FOURTH EDITION

Applied Statistics and the SAS Programming Language

Ronald P. Cody
ROBERT WOOD JOHNSON MEDICAL SCHOOL

Jeffrey K. Smith
RUTGERS UNIVERSITY

PRENTICE HALL, Upper Saddle River, New Jersey 07458

Library of Congress Cataloging-in-Publication Data

Cody, Ronald P.
Applied statistics and the SAS programming language / Ronald P. Cody, Jeffrey K. Smith – 4th ed.
p. cm.
Includes bibliographical references (p. –) and index.
ISBN 0-13-743642-4 (pbk.)
1. SAS (Computer file) 2. Mathematical statistics–Data processing. I. Smith, Jeffrey K. II. Title.
QA276.4.C53 1997
519.5'0285'5369–dc21 97-1737
CIP

Acquisition editor: Ann Heath
Editorial assistant: Mindy Ince
Editorial director: Tim Bozik
Editor-in-chief: Jerome Grant
Assistant vice president of production and manufacturing: David W. Riccardi
Editorial/production supervision: Nicholas Romanelli
Managing editor: Linda Mihatov Behrens
Executive managing editor: Kathleen Schiaparelli
Manufacturing buyer: Alan Fischer
Manufacturing manager: Trudy Pisciotti
Marketing manager: Melody Marcus
Marketing assistant: Jennifer Pan
Creative director: Paula Maylahn
Cover designer: Jayne Conte

Cover illustration: From the manuscripts of Leonardo DaVinci; published by Charles Raviasson-Mollien, 6 vols., Paris, 1881-91.
Back cover photo of Ron Cody by Russ Cody.

Printed in the United States of America

10 9 8 7 6

ISBN: 0-13-743642-4

Prentice-Hall, International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada, Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Simon & Schuster Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

To our parents,

Ralph and Bettie Smith

and

Philip and Margaret Cody

Contents

Applied Statistics and SAS Software

SAS Programming

Preface to the Fourth Edition

When we began creating this fourth edition, several facts were clear: First, SAS software continues to evolve and improve. Second, our programming techniques have also improved. Third, several statistical techniques (such as logistic regression) have become popular and required coverage in this edition.

We have met many readers of earlier editions at meetings and conferences and were delighted to hear good things and constructive criticisms of the book. These we have taken to heart and attempted to improve old material and add relevant new topics. This fourth edition is the result of such reader reaction.

Most researchers are inherently more interested in the substance of their research endeavors than in statistical analyses or computer programming. Yet, conducting such analyses is an integral link in the research chain (all too frequently, the weak link). This condition is particularly true when there is no resource for the applied researcher to refer to for assistance in running computer programs for statistical analyses. *Applied Statistics and the SAS Programming Language* is intended to provide the applied researcher with the capacity to perform statistical analyses with SAS software without wading through pages of technical documentation.

The reader is provided with the necessary SAS statements to run programs for most commonly used statistics, explanations of the computer output, interpretations of results, and examples of how to construct tables and write up results for reports and journal articles. Examples have been selected from business, medicine, education, psychology, and other disciplines.

SAS software is a combination of a statistical package, a data-base management system, and a high-level programming language. Like SPSS, BMDP, Systat, and other statistical packages, SAS software can be used to describe a collection of data and produce a variety of statistical analyses. However, SAS software is much more than just a statistical package. Many companies and educational institutions use SAS software as a high-level data-management system and programming language. It can be used to organize and transform data and to create reports of all kinds. Also, depending on which portions of the SAS system you have installed in your computer (and what type of computer system you are running), you may be using the SAS system for interactive data entry or an on-line system for order entry or retrieval.

This book concentrates on the use of the SAS system for the statistical analysis of data and the programming capabilities of SAS software most often used in educational and research applications.

The SAS system is a collection of products, available from the SAS Institute in Cary, North Carolina. The major products available from the SAS Institute are:

Base SAS ®	The main SAS module, which provides some data manipulation and programming capability and some elementary descriptive statistics
SAS/STAT ®	The SAS product that includes all the statistical programs except the elementary ones supplied with the base package
SAS/GRAPH ®	A package that provides high-quality graphs and maps. Note that "line graphics" (the graphs and charts that are produced by normal character plots) are available in the base and SAS/STAT packages. SAS/GRAPH adds the ability to produce high-quality camera-ready graphs, maps, and charts.
SAS/FSP ®	These initials stand for the Full Screen Product. This package allows you to search, modify, or delete records directly from a SAS data file. It also provides for data entry with sophisticated data checking capabilities. Procedures available with FSP are FSBROWSE, FSEDSIT, FSPRINT, FSLIST, and FSLETTER.
SAS/AF ®	AF stands for the SAS Applications Facility. This product is used by data processing professionals to create "turn key" or menu systems for their users. It is also used to create instructional modules relating to the SAS system.
SAS/ETS ®	The Econometric and Time Series package. This package contains specialized programs for the analysis of time series and econometric data.
SAS/OR ®	A series of operations research programs.
SAS/QC ®	A series of programs for quality control.
SAS/IML ®	The Interactive Matrix Language module. The facilities of IML used to be included in PROC MATRIX in the version 5 releases. This very specialized package allows for convenient matrix manipulation for the advanced statistician.

SAS software now runs on a variety of computers, from personal computers to large multimillion dollar mainframes. The original version of the SAS system was written as a combination of PL/1 and IBM assembly language. Today, SAS software runs under Windows and Windows 95 on IBM compatible minicomputers, on most Macintosh computers, under UNIX on a large number of minicomputers and workstations, on IBM computers under a variety of operation systems, on Digital Equipment VAX computers, and others too numerous to mention. The major reason for the ability of SAS software to run on such a variety of machines is that all SAS software was rewritten in C and designed so that most of the code was system-independent. The conversion of the entire system to the C programming language was one of the largest (and most successful) programming projects ever undertaken. To migrate the SAS system to another computer or operating system, only a small system-dependent portion of code needs to be rewritten. The result is that new versions of SAS software

are made available for all computers very quickly, and the versions of SAS systems from one computer to another are much alike.

Learning to program is a skill that is difficult to teach well. While you will find our examples "reader-friendly" and their logic easy to follow, learning to write your own programs is another matter. Only through lots of practice will your programming skills develop. So, when you finish a chapter, please spend the time doing as many problems as you can. We wish you happy programming.

We express our gratitude to two colleagues who reviewed the manuscript and made many comments beneficial to this revision of the text. Our thanks, therefore, to Sylvia Brown and Robert Hamer, at the Robert Wood Johnson Medical School. Our sincere thanks also to Ann Heath, acquisitions editor for statistics at Prentice Hall, for her encouragement and support, and to Nicholas Romanelli for his superb editing, patience, and good cheer.

RON CODY
JEFFREY SMITH

CHAPTER

1

A SAS Tutorial

A. *Introduction*

For the novice, engaging in statistical analysis of data can seem as appealing as going to the dentist. If that pretty much describes your situation, perhaps you can take comfort in the fact that this is the fourth edition of this book—meaning that the first three editions sold pretty well, and this time we may get it right. Our purpose for this tutorial is to get you started using SAS software. The key objective is to get one program to run successfully. If you can do that, you can branch out a little bit at a time. Your expertise will grow.

The SAS System is a combination of programs originally designed to perform statistical analysis of data. Other programs you may have heard of are SPSS, BMDP, or SYSTAT. If you look at personal computer magazines, you might run across other programs, primarily designed to run on personal computers. Since its inception, the SAS system has grown to where it can perform a fairly impressive array of nonstatistical functions. We'll get into a little of that in later chapters. For now, we want to learn the most basic rudiments of the SAS system. If you skipped over it, the Preface to the fourth edition contains some history of SAS software development and a more complete overview of the capabilities of SAS software.

To begin, SAS software runs on a wide variety of computers and operating systems (computer people call these platforms), and we don't know which one you have. You may have an IBM compatible personal computer running Windows or, perhaps, Windows 95. You may have a Macintosh computer, or you may be connected to a network or a mainframe computer by way of a modem or network connection. You may only have a sophisticated VCR, which you think is a computer. If you are unsure of what platform you are using or what version of SAS software you are using, ask someone. As a matter of fact, right now would be a good time to invite the best computer person you know to lunch. Have that person arrive at your office about an hour before lunch so you can go over some basic elements of your system. You need to find out what is necessary on your computer to get the SAS system running. What we can teach you here is how to use the SAS system, and how to adapt to your computer system.

If you are running on a mainframe, you may well be submitting what are called "batch" jobs. When you run batch jobs, you send your program (across a phone line or a network from your personal computer or terminal) to the computer. The computer runs your program and holds it until you ask for it or prints out the results on a high-speed printer. You may even need to learn some Job Control Language (which you have to get from your local computer folks), and then you can proceed.

If you are running on a personal computer, or running in what is called interactive mode on a minicomputer or mainframe, then you need to learn how to use the SAS Display Manager. The look and feel of SAS once you are in the Display Manager is pretty much the same whatever platform you are using. If you are undaunted, take a deep breath and plunge into the real content in the next section. If you are already daunted, take a minute and get that lunch scheduled, then come back to this.

B. *Computing with SAS Software: An Illustrative Example*

SAS programs communicate with the computer by SAS "statements." There are several kinds of SAS statements, but they share a common feature—they end in a semicolon. A semicolon in a SAS program is like a period in English. Probably the most common error found in SAS programs is the omission of the semicolon. This omission causes the computer to read two statements as a run-on statement and invariably fouls things up.

SAS programs are comprised of SAS statements. Some of these statements provide information to the system, such as how many lines to print on a page and what title to print at the top of the page. Other statements act together to create SAS data sets, while other SAS statements act together to run predefined statistical or other routines. Groups of SAS statements that define your data and create a SAS data set are called a DATA step; SAS statements that request predefined routines are called a PROC (pronounced "prock") step. DATA steps tell SAS programs about your data. They are used to indicate where the variables are on data lines, what you want to call the variables, how to create new variables from existing variables, and several other functions we mention later. PROC (short for PROCEDURE) steps indicate what kind of statistical analyses to perform and provide specifications for those analyses. Let's look at an example. Consider this simple data set:

SUBJECT NUMBER	GENDER (M or F)	EXAM 1	EXAM 2	HOMEWORK GRADE
10	M	80	84	A
7	M	85	89	A
4	F	90	86	B
20	M	82	85	B
25	F	94	94	A
14	F	88	84	C

We have five variables (SUBJECT NUMBER, GENDER, EXAM 1, EXAM 2, and HOMEWORK GRADE) collected for each of six subjects. The unit of analysis, people for this example, is called an observation in SAS terminology. SAS software uses the term "variable" to represent each piece of information we collect for each observation. Before we can write our SAS program, we need to assign a variable name to each variable. We do this so that we can distinguish one variable from another when doing computations or requesting statistics. SAS variable names must conform to a few simple rules: They must start with a letter, be not more than eight characters (letters or numerals) in length, and cannot contain blanks or certain special characters such as commas, semicolons, etc. (The underscore character (_) is a valid character for SAS variable names and can be used to make variable names more readable.) Therefore, our column headings of "SUBJECT NUMBER," or "EXAM 1" are not valid SAS variable names. Logical SAS variable names for this collection of data would be:

```
SUBJECT    GENDER    EXAM1    EXAM2    HWGRADE
```

It is prudent to pick variable names that help you remember which name goes with which variable. We could have named our five variables VAR1, VAR2, VAR3, VAR4, and VAR5, but we would then have to remember that VAR1 stands for "SUBJECT NUMBER," and so forth.

To begin, let's say we are interested only in getting the class means for the two exams. In reality it's hardly worth using a computer to add up six numbers, but it does provide a nice example. In order to do this, we could write the following SAS program:

```
DATA TEST;  ①
   INPUT SUBJECT 1-2 GENDER $ 4 EXAM1 6-8 EXAM2 10-12  ②
   HWGRADE $ 14;
DATALINES;  ③
10 M  80  84 A
 7 M  85  89 A
 4 F  90  86 B
20 M  82  85 B
25 F  94  94 A
14 F  88  84 C
;
PROC MEANS DATA=TEST;  ④
RUN;  ⑤
```

The first four lines make up the DATA step. In this example, the DATA step begins with the word DATA and ends with the word DATALINES. Older versions of SAS software used the term CARDS instead of DATALINES. Either term is still valid. (If you don't know what a computer card is, ask an old person.) Line ① tells the program that we want to create a SAS data set called TEST. The next two lines ② show an INPUT statement which gives the program two pieces of information: what to call the variables and where to find them on the data line. Notice that this single SAS statement occupies two lines. The SAS system understands this is a single SAS statement because there is a single semicolon at the end of it. The first variable is SUBJECT and can be found in columns 1 and 2 of the data line. The second variable is GENDER and can be found in column 4. The dollar sign after GENDER means that GENDER is a character (alphanumeric) variable, that is, a variable that can have letters or numbers as data values. (More on this later.) EXAM1 is in columns 6–8, etc. The DATALINES statement ③ says that the DATA statements are done and the next thing the program should look for are the data themselves. The next six lines are the actual data. In this example, we are including the data lines directly in the program. Later on in this book, we will show you how to read data from external files.

Great latitude is possible when putting together the data lines. Using a few rules will make life much simpler for you. These are not laws; they are just suggestions. First, put each new observation on a new line. Having more than one line per observation is often necessary (and no problem), but don't put two observations on one line (at least for now). Second, line up your variables. Don't put EXAM1 in columns 6–8 on one line and in columns 9–11 on the next. SAS software can actually handle some degree of sloppiness here, but sooner or later it'll cost you. Third, right-justify your numeric data values. If you have AGE as a variable, record the data as follows:

Correct	Problematic
87	87
42	42
9	9
26	26
4	4
Right-justified	Left-justified

Again, SAS software doesn't care whether you right-justify numeric data or not, but other statistical programs will, and right justification is standard. Fourth, use character variables sparingly. Take HWGRADE, for example. We have HWGRADE recorded as a character value. But we could have recorded it as 0–4 (0 = F, 1 = D, etc.). As it stands, we cannot compute a mean grade. Had we coded HWGRADE numerically, we could get an average grade. Enough on how to code data for now.

Back to our example. A SAS program knows that the data lines are completed when it finds a SAS statement or a single semicolon. When you include your data lines in the program, as in this example, we recommend placing a single semicolon on the line directly below your last line of data. The next SAS statement ④ is a PROC statement. PROC says "Run a procedure" to the program. We specify which procedure right after the word PROC. Here we are running a procedure called MEANS. Following the procedure name (MEANS), we place the option DATA= and specify that this procedure should compute statistics on the data set called TEST. In this example, we could omit the option DATA=TEST, and the procedure would operate on the most recently created SAS data set, in this case, TEST. We recommend that you include the DATA= option on every procedure since, in more advanced SAS programs, you can have procedures that create data sets as well as many data sets "floating around." By including the DATA= option, you can always be sure your procedure is operating on the correct data set.

The MEANS procedure calculates the mean for any variables you specify. The RUN statement ⑤ is necessary only when SAS programs are run under the Display Manager. The RUN statement tells SAS that there are no more statements for the preceding procedure and to go ahead and do the calculations. If we have several PROCs in a row, we only need a single RUN statement at the end of the program. However, as a stylistic standard, we prefer to end every procedure with a RUN statement and to separate procedures by a blank line to make the programs more readable.

When this program is executed, it produces something called the SAS LOG and the SAS OUTPUT. The SAS LOG is an annotated copy of your original program (without the data listed). It's a lot like a phone book: Usually it's pretty boring, but sometimes you need it. Any SAS error messages will be found there, along with information about the data set that was created. The SAS LOG for this program is shown below:

```
NOTE: Copyright (c) 1989-1995 by SAS Institute Inc., Cary, NC, USA.
NOTE: SAS (r) Proprietary Software Release 6.11  TS020
      Licensed to RON CODY/ROBERT WOOD JOHNSON MEDICAL SCHOOL, Site XXX.

NOTE: Release 6.11 of the SAS(R) System for Windows(R).

NOTE: AUTOEXEC processing completed.

1
2         DATA TEST;
3            INPUT SUBJECT 1-2 GENDER $ 4 EXAM1 6-8 EXAM2 10-12
4            HWGRADE $ 14;
5         DATALINES;

NOTE: The data set WORK.TEST has 6 observations and 5 variables.
NOTE: The DATA statement used 0.27 second.


12        ;
13        PROC MEANS DATA=TEST;
14        RUN;

NOTE: The PROCEDURE MEANS used 0.17 second.
```

The more important part of the output is found in the OUTPUT window if you are using the Display Manager. It contains the results of the computations and procedures requested by our PROC statements. This portion of the output from the program above is shown next:

Variable	N	Mean	Std Dev	Minimum	Maximum
SUBJECT	6	13.3333333	7.9916623	4.0000000	25.0000000
EXAM1	6	86.5000000	5.2057660	80.0000000	94.0000000
EXAM2	6	87.0000000	3.8987177	84.0000000	94.0000000

If you don't specify which variables you want, SAS software will calculate the mean and several other statistics for every numeric variable in the data set. Our program calculated means for SUBJECT, EXAM1, and EXAM2. Since SUBJECT is just an arbitrary ID number assigned to each student, we aren't really interested in its mean. We can avoid getting it (and using up extra CPU cycles) by adding a new statement under PROC MEANS:

```
PROC MEANS DATA=TEST;
   VAR EXAM1 EXAM2;   ⑥
RUN;
```

The indentation used is only a visual aid. The VAR statement ⑥ specifies on which variables to run PROC MEANS. PROC MEANS not only gives you means, it gives you the number of observations used to compute the mean, the standard deviation, the minimum score found, and the maximum score found. PROC MEANS can compute many other statistics such as variance and standard error. You can specify just which pieces you want in the PROC MEANS statement. For example:

```
PROC MEANS DATA=TEST N MEAN STD STDERR MAXDEC=1;
   VAR EXAM1 EXAM2;
RUN;
```

will get you only the number of observations with no missing values (N), mean (MEAN), standard deviation (STD), and standard error (STDERR) for the variables EXAM1 and EXAM2. In addition, the statistics will be rounded to one decimal place (because of the MAXDEC=1 option). Chapter 2 describes most of the commonly requested options used with PROC MEANS.

C. Enhancing the Program

The program as it is currently written provides some useful information but, with a little more work, we can put some "bells and whistles" on it. The bells and whistles version below adds the following features: It computes a final grade, which we will let be the average of the two exam scores; it assigns a letter grade based on that final score; it lists the students in student number order, showing their exam scores, their final grades and homework grades; it computes the class average for the exams and final grade and a frequency count for gender and homework grade; finally, it gets you a cup of coffee and tells you that you are a fine individual.

```
DATA EXAMPLE;  ①
   INPUT SUBJECT GENDER $ EXAM1 EXAM2 ②
   HWGRADE $;
FINAL = (EXAM1 + EXAM2) / 2; ③
IF FINAL GE 0 AND FINAL LT 65 THEN GRADE='F';  ④
⑤ ELSE IF FINAL GE 65 AND FINAL LT 75 THEN GRADE='C';
   ELSE IF FINAL GE 75 AND FINAL LT 85 THEN GRADE='B';
   ELSE IF FINAL GE 85 THEN GRADE='A';
DATALINES;  ⑥
10 M  80  84 A
 7 M  85  89 A
 4 F  90  86 B
20 M  82  85 B
25 F  94  94 A
14 F  88  84 C
;
PROC SORT DATA=EXAMPLE;  ⑦
   BY SUBJECT;  ⑧
RUN;  ⑨

PROC PRINT DATA=EXAMPLE;  ⑩
   TITLE 'Roster in Student Number Order';
   ID SUBJECT;
   VAR EXAM1 EXAM2 FINAL HWGRADE GRADE;
RUN;

PROC MEANS DATA=EXAMPLE N MEAN STD STDERR MAXDEC=1; ⑪
   TITLE 'Descriptive Statistics';
   VAR EXAM1 EXAM2 FINAL;
RUN;

PROC FREQ DATA=EXAMPLE; ⑫
   TABLES GENDER HWGRADE GRADE;
RUN;
```

As before, the first four lines constitute our DATA step. Line ① is an instruction for the program to create a data set whose data set name is "EXAMPLE." (Remember that data set names follow the same conventions as variable names.) Line ② is an INPUT statement which is different from the one in the previous example.

We could have used the same INPUT statement as in the previous example but wanted the opportunity to show you another way that SAS programs can read data. Notice that there are no column numbers following the variable names.

This form of an INPUT statement is called list input. To use this form of INPUT, the data values must be separated by one or more blanks (or other separators which computer people call delimiters). If you use one of the other possible delimiters, you need to modify the program accordingly (see Chapter 12, Section C). The order of the variable names in the list corresponds to the order of the values in the line of data. In this example, the INPUT statement tells the program that the first variable in each line of data represents SUBJECT values, the next variable is GENDER, the third EXAM1, and so forth. If your data conform to this "space-between-each-variable" format, then you don't have to specify column numbers for each variable listed in the INPUT statement. You may want to anyway, but it isn't necessary. (You still have to follow character variable names with a dollar sign.) If you are going to use "list input," then every variable on your data lines must be listed. Also, since the order of the data values is used to associate values with variables, we have to make special provisions for missing values. Suppose that subject number 10 (the first subject in our example) did not take the first exam. If we listed the data like this:

```
10 M    84 A
```

with the EXAM1 score missing, the 84 would be read as the EXAM1 score, the program would read the letter "A" as a value for EXAM2 (which would cause an error since the program was expecting a number), and, worst of all, the program would look on the next line for a value of homework grade. You would get an error message in the SAS LOG telling you that you had an invalid value for EXAM2 and that SAS went to a new line when the INPUT statement reached past the end of a line. You wouldn't understand these error messages and might kick your dog.

To hold the place of a missing value when using a list INPUT statement, use a period to represent the missing value. The period will be interpreted as a missing value by the program and will keep the order of the data values intact. When we specify columns as in the first example, we can use blanks as missing values. Using periods as missing values when we have specified columns in our INPUT statement is also OK, but not recommended. The correct way to represent this line of data, with the EXAM1 score missing, is:

```
10 M . 84 A
```

Since list input requires one or more blanks between data values, we need at least one blank before and after the period. We may choose to add extra spaces in our data to allow the data values to line up in columns.

Line ③ is a statement assigning the average of EXAM1 and EXAM2 to a variable called FINAL. The variable name "FINAL" must conform to the same naming conventions as the other variable names in the INPUT statement. In this example, FINAL is calculated by adding together the two exam scores and dividing by 2. Notice that we indicate addition with a + sign and division by a / sign. We need the parentheses because, just the same as in handwritten algebraic expressions, SAS computations are performed according to a hierarchy. Multiplication and division are performed before addition and subtraction. Thus, had we written:

```
FINAL = EXAM1 + EXAM2 / 2;
```

the FINAL grade would have been the sum of the EXAM1 score and half of EXAM2. The use of parentheses tells the program to add the two exam scores first, and then divide by 2. To indicate multiplication, we use an asterisk (*); to indicate subtraction, we use a - sign. Exponentiation, which is performed before multiplication or division, is indicated by two asterisks. As an example, to compute A times the square of B we write:

```
X = A * B**2;
```

The variable FINAL was computed from the values of EXAM1 and EXAM2. That does not, in any way, make it different from the variables whose values were read in from the raw data. When the DATA step is finished, the SAS procedures that follow will not treat variables such as FINAL any differently from variables such as EXAM1 and EXAM2.

The IF statement ④ and the ELSE IF statements ⑤ are logical statements that are used to compute a letter grade. They are fairly easy to understand. When the condition specified by the IF statement is true, the instructions following the word THEN are executed. The logical comparison operators used in this example are GE (greater than or equal to) and LT (less than). So, if a FINAL score is greater than or equal to 0, and less then 65, a letter grade of 'F' is assigned. The ELSE statements are only executed if a previous IF statement is not true. For example, if a FINAL grade is 73, the first IF statement ⑤ is not true, so the ELSE IF statement ⑥ is tested. Since this statement is true, a GRADE of 'C' is assigned, and all the following ELSE IF statements are skipped.

Other logical operators and their equivalent symbolic form are shown in the table below:

Expression	Symbol	Meaning
EQ	=	Equal
LT	<	Less than
LE	< =	Less than or equal
GT	>	Greater than
GE	> =	Greater than or equal
NE	^=	Not equal
NOT	^	Negation

NOTE: The symbols for NOT and NE may vary, depending on your system.

The "DATALINES" statement ⑥ indicates that the DATA step is complete and that the following lines contain data.

Notice that each SAS statement ends with a semicolon. As mentioned before, the semicolon is the logical end of a SAS statement. We could have written the first four lines like this:

```
DATA EXAMPLE; INPUT SUBJECT GENDER $
EXAM1 EXAM2 HWGRADE $; FINAL =
(EXAM1 + EXAM2) / 2;
```

The program would still run correctly. Using a semicolon as a statement delimiter is convenient since we can write long SAS statements on several lines and simply put a semicolon at the end of the statement. However, if you omit a semicolon at the end of a SAS statement, the program will attempt to read the next statement as part of previous statement, causing an error. This may not only cause your program to die, it may result in a bizarre error message emanating from the SAS system. Omission of one or more semicolons is the most common programming error for novice SAS programmers. Remember to watch those semicolons! Notice also that the data lines, since they are not SAS statements, do not end with semicolons. In fact, under most usual circumstances, data are not allowed to contain semicolons.

Following the DATALINES statement are our data lines. Remember that if you have data that have been placed in preassigned columns with no spaces between the data values, you must use the form of the INPUT shown earlier, with column specifications after each variable name. This form of data is discussed further in Chapter 12. We have used a RUN statement to end every procedure. Each RUN statement tells the system that we are finished with a section of the program and to do the computations just concluded. Remember, when using the Display Manager, only the last RUN statement is absolutely necessary; the others are really only a matter of programming style.

D. SAS Procedures

Immediately following the data is a series of PROCs. They perform various functions and computations on SAS data sets. Since we want a list of subjects and scores in subject order, we first include a SORT PROCEDURE ⑦, ⑧, and ⑨. Line ⑦ indicates that we plan to sort our data set; line ⑧ indicates that the sorting will be by SUBJECT number. Sorting can be multilevel if desired. For example, if we want separate lists of male and female students in subject number order, we write:

```
PROC SORT DATA=EXAMPLE;
   BY GENDER SUBJECT;
RUN;
```

This multilevel sort indicates that we should first sort by GENDER (F's followed by M's—character variables are sorted alphabetically), then in SUBJECT order within GENDER.

The PRINT procedure ⑩ requests a listing of our data (which is now in SUBJECT order). The PRINT procedure is used to list the data values in a SAS data set. We have followed our PROC PRINT statement with three statements that supply information to the procedure. These are the TITLE, ID, and VAR statements. As with many SAS procedures, the supplementary statements following a PROC can be placed in any order. Thus:

```
PROC PRINT DATA=EXAMPLE;
   ID SUBJECT;
   TITLE 'Roster in Student Number Order';
   VAR EXAM1 EXAM2 FINAL HWGRADE GRADE;
RUN;
```

is equivalent to

```
PROC PRINT DATA=EXAMPLE;
   TITLE 'Roster in Student Number Order';
   ID SUBJECT;
   VAR EXAM1 EXAM2 FINAL HWGRADE GRADE;
RUN;
```

SAS programs recognize the keywords TITLE, ID, and VAR and interpret what follows in the proper context. Notice that each statement ends with its own semicolon. The words following TITLE are placed in single (or double) quotes and will be printed across the top of each of the SAS output pages. The ID variable, SUBJECT in this case, will cause the program to print the variable SUBJECT in the first column of the report, omitting the column labeled OBS (observation number) which the program will print when an ID variable is absent. The variables following the keyword VAR indicate which variables, besides the ID variable, we want in our report. The order of these variables in the list also controls the order in which they appear in the report.

The MEANS procedure ⑪ is the same as the one we used previously. Finally, the FREQ procedure ⑫ (you're right: pronounced "PROC FREAK") requests a frequency count for the variables GENDER, HWGRADE, and GRADE. That is, what is the number of Males and Females, the number of A's, B's, etc., as well as the percentages of each category. PROC FREQ will compute frequencies for the variables listed on the TABLES statement. The reason that SAS uses the keyword TABLES instead of VAR for this list of variables is that PROC FREQ can also produce n-way tables (such as 2×3 tables).

Output from the complete program is shown below:

```
Roster in Student Number Order
                                                        1
                              13:15 Wednesday, July 31, 1996

SUBJECT   EXAM1   EXAM2   FINAL   HWGRADE   GRADE

   4        90      86     88.0      B        A
   7        85      89     87.0      A        A
  10        80      84     82.0      A        B
  14        88      84     86.0      C        A
  20        82      85     83.5      B        B
  25        94      94     94.0      A        A

Descriptive Statistics        13:15 Wednesday, July 31, 1996   2

Variable  N        Mean       Std Dev      Std Error
-----------------------------------------------------
EXAM1     6        86.5         5.2           2.1
EXAM2     6        87.0         3.9           1.6
FINAL     6        86.8         4.2           1.7
-----------------------------------------------------

Descriptive Statistics        13:15 Wednesday, July 31, 1996   3

                               Cumulative  Cumulative
GENDER   Frequency   Percent   Frequency    Percent
-----------------------------------------------------
F               3      50.0          3        50.0
M               3      50.0          6       100.0

                               Cumulative  Cumulative
HWGRADE  Frequency   Percent   Frequency    Percent
-----------------------------------------------------
A               3      50.0          3        50.0
B               2      33.3          5        83.3
C               1      16.7          6       100.0

                               Cumulative  Cumulative
GRADE    Frequency   Percent   Frequency    Percent
-----------------------------------------------------
A               4      66.7          4        66.7
B               2      33.3          6       100.0
```

The first part of the output (the page number is shown at the extreme right of each page) is the result of the PROC PRINT on the sorted data set. Each column is labeled with the variable name. Because we used an ID statement with SUBJECT as the ID variable, the left-most column shows the SUBJECT number instead of the default OBS column, which would have been printed if we did not have an ID variable.

Page 2 of the output lists each of the variables listed on the VAR statement and produces the requested statistics (N, mean, standard deviation, and standard error) all to the tenths place (because of the MAXDEC=1 option).

Page 3 is the result of the PROC FREQ request. Notice that the title "Descriptive Statistics" is still printed at the top of each page. Titles remain in effect until the end of the current SAS session or if you change it to another title line. This portion of the listing gives you frequencies (the number of observations with particular values) as well as percentages. The two columns labeled "Cumulative Frequency" and "Cumulative Percent" are not really useful in this example. In other cases, where a variable represents an ordinal quantity, the cumulative statistics may be more useful.

E. Overview of the SAS DATA Step

Let's spend a moment examining what happens when we execute a SAS program. This discussion is a bit technical and can be skipped, but an understanding of how SAS software works will help when you are doing more advanced programming. When the DATA statement is executed, SAS software allocates a portion of a disk and names the data set "EXAMPLE," our choice for a data set name. Before the INPUT statement is executed, each of the character and numeric variables is assigned a missing value. Next, the INPUT statement reads the first line of data and substitutes the actual data values for the missing values. These data values are not yet written to our SAS data set EXAMPLE but to a place called the Program Data Vector (PDV). This is simply a "holding" area where data values are stored before they are transferred to the SAS data set. The computation of the final grade comes next ③, and the result of this computation is added to the PDV. Depending on the value of the final grade, a letter grade is assigned by the series of IF and ELSE IF statements. The DATALINES line triggers the end of the DATA step, and the values in the PDV are transferred to the SAS data set. The program then returns control back to the INPUT statement to read the next line of data, compute a final grade, and write the next observation to the SAS data set. This reading, processing, and writing cycle continues until no more observations remain. Wasn't that interesting?

F. Syntax of SAS Procedures

As we have seen above, SAS procedures can have options. Also, procedures often have statements, like the VAR statement above, which supply information to the procedure. Finally, statements can also have options. We will show you the general syntax of SAS procedures and then illustrate it with some examples. The syntax for all SAS procedures is:

```
PROC PROCNAME options;
   STATEMENTS / statement options;
        •
        •
        •
   STATEMENTS / statement options;
RUN;
```

First, all procedures start with the word PROC followed by the procedure name. If there are any procedure options, they are placed, in any order, between the procedure name and the semicolon, separated by spaces. If we refer to a SAS manual, under PROC MEANS we will see a list of options to be used with the procedure. As mentioned, N, MEAN, STD, STDERR, and MAXDEC= are some of the available options. A valid PROC MEANS request for statistics from a data set called EXAMPLE, with options for N, MEAN, and MAXDEC would be:

```
PROC MEANS DATA=EXAMPLE N MEAN MAXDEC=1;
RUN;
```

Next, most procedures need statements to supply more information about which type of analysis to perform. An example would be the VAR statement used with PROC MEANS. Statements follow the procedure, in any order. They each end with a semicolon. So, to run the PROC MEANS statement above, on the variables EXAM1 and EXAM2, and to supply a title, we would enter:

```
PROC MEANS DATA=EXAMPLE N MEAN STD MAXDEC=1;
   TITLE 'Descriptive Statistics on Exam Scores';
   VAR EXAM1 EXAM2;
RUN;
```

The order of the TITLE and VAR statements can be interchanged with no change in the results. Finally, some procedure statements also have options. Statement options are placed between the statement keyword and the semicolon and separated from the statement by a slash. To illustrate, we need to choose a procedure other than PROC MEANS. Let's use PROC FREQ as an example. As we saw, PROC FREQ will usually have one or more TABLES statements following it. There are TABLES options that control which statistics can be placed in a table. For example, if we do not want the cumulative statistics printed, the statement option NOCUM is used. Since this is a statement option, it is placed between the TABLES statement and the semicolon, separated by a slash. The PROC FREQ request in the earlier example, modified to remove the cumulative statistics, would be:

```
PROC FREQ DATA=EXAMPLE;
   TABLES GENDER HWGRADE GRADE/ NOCUM;
RUN;
```

To demonstrate a procedure with procedure options and statement options, we use the ORDER= option with PROC FREQ. This useful option controls the order that the values can be arranged in our frequency table. One option is

ORDER=FREQ, which enables the frequency table to be arranged in frequency order, from the highest frequency to the lowest. So, to request frequencies in descending order of frequency and to omit cumulative statistics from the output, we write our PROC FREQ statements as follows:

```
PROC FREQ DATA=EXAMPLE ORDER=FREQ;
   TABLES GENDER HWGRADE GRADE/ NOCUM;
RUN;
```

G. Comment Statements

Before concluding this chapter, we introduce one of the most important SAS statements—the comment statement. (Yes, we're not kidding!) A properly commented program indicates that a true professional is at work. A comment, inserted in a program is one or more lines of text that are ignored by the program—they are there only to help the programmer or researcher when he or she reads the program at a later date.

There are two ways to insert comments into a SAS program. One is to write a comment statement. Begin it with an asterisk (*) and end it with a semicolon. There are many possible styles of comments using this method. For example:

```
*Program to Compute Reliability Coefficients
 Ron Cody
 September 18, 1997
 Program Name: FRED stored in directory C:\MYDATA

 Contact Fred Cohen at 555-4567;
```

Notice the convenience of this conclusion. Just enter the * and type as many lines as necessary, ending with the semicolon. Just make sure the comment statement doesn't contain a semicolon. Some programmers get fancy and make pretty boxes for their comments, like this:

```
*-----------------------------------------------------*
| Program Name: FRED  stored in C:\MYDATA             |
| Purpose: To compute reliability coefficients        |
| Contact: Fred Cohen at 555-4567                     |
| Date: September 18, 1997                            |
| Programmer: Ron Cody                                |
*-----------------------------------------------------*;
```

Notice that the entire box is a SAS comment statement since it begins with an asterisk and ends in a semicolon. Notice also that the box literally cries out, "I need a life!"

You may also choose to comment individual lines by resorting to one of the following three ways:

```
QUES = 6 - QUES;  *Transform QUES VAR;
X = LOG(X);  *LOG Transform of X;
```

or

```
*Transform the QUES Variable;
QUES = 6 - QUES;
*Take the LOG of X;
X = LOG(X);
*True professonal at work;
```

or

```
*
*Transform the QUES Variable
*;
QUES = 6 - QUES;
*
*Take the LOG of X
*;
X = LOG(X);
*
*True professonal at work
*;
```

The last example uses more than one asterisk to set off the comment, for visual effect. Note however, that each group of three lines is a single comment statement since it begins with an asterisk and ends with a semicolon.

An alternative commenting method begins a comment with a /* and ends with a */. This form of comment can be embedded within a SAS statement and can include semicolons within the comment itself. It can occur any place a blank can occur. A few examples:

```
/* This is a comment line */
```

or


```
/*----------------------------------------------------*
 | This is a pretty comment box using the slash star  |
 | method of commenting.  Notice that it begins with  |
 | a slash star and ends with a star slash.           |
 *----------------------------------------------------*/
```

or

```
DATA EXAMPLE;  /* The data statement */
   INPUT SUBJECT GENDER $
         EXAM1 /* EXAM1 is the first exam score */
         EXAM2 /* EXAM2 is the second exam score */
         HWGRADE $;
   FINAL = (EXAM1 + EXAM2)/2;  /* Compute a composite grade */
DATALINES;
```

Let us show you one final, very useful trick using a comment statement, before concluding this chapter. Suppose you have written a program and run several procedures. Now, you return to the program and wish to run additional procedures. You could edit the program, remove the old procedures, and add the new ones. Or, you could "comment them out" by preceding the section with a /* and ending with a */, making the entire section a comment. As an example, our commented program could look like this:

```
DATA MYPROG;
   INPUT X Y Z;
DATALINES;
1 2 3
3 4 5
;
/*****************************************************
PROC PRINT DATA=MYPROG;
   TITLE 'MY TITLE';
   VAR X Y Z;
RUN;
*****************************************************/
PROC CORR DATA=MYPROG;
   VAR X Y Z;
RUN;
```

The print procedure is not executed since it is treated as a comment; the correlation procedure will be run.

One final point: when running a batch program on IBM mainframes under MVS, a /* in columns 1 and 2 causes the program to terminate.

A few extra minutes are needed to comment a SAS program, but that time is well spent. You will be thankful that you added comments to a program when it comes time to modify the program or if you expect other people to understand how your program works. To repeat, comments in your program take a little time but are usually worth it.

H. References

One of the advantages of using SAS software is the variety of procedures that can be performed. We cannot describe them all here nor can we explain every option of the procedures we do describe. You may, therefore, want to obtain one or more of the following manuals available from the SAS Institute Inc., Book Sales Department, SAS Campus Drive, Cary, NC 27513-2414. The SAS Institute also takes phone orders. Call (919)677-8000 or (800)727-3228.

SAS Language and Procedures: Introduction, Version 6, First Edition (order number #P56074)

SAS Language and Procedures Usage, Version 6, First Edition (order number #P56075)

SAS Language and Procedures: Usage 2, Version 6, First Edition (order number #P560078)

SAS Language: Reference, Version 6, First Edition (order number #P56076)

SAS Procedures Guide, Version 6, Third Edition (order number #P560080)

SAS/STAT User's Guide, Version 6, Fourth Edition, Volumns 1 and 2 (order number #P56045)

SAS/STAT Software: Changes and Enhancements through Release 6.11 (order number #P55356)

SAS Programming by Example, by Ron Cody and Ray Pass (order number #P55126)

The SAS Workbook, by Ron Cody (order number #P55473)

The SAS Workbook: Solutions, by Ron Cody (order number #P55475)

The SAS Workbook and Solutions (both books sold together at a discount), by Ron Cody (order number #55594)

Below are some statistics books that we recommend:

Statistical Principles in Experimental Design, by B. J. Winer (McGraw-Hill, New York, 1991)

Statistical Methods, by Snedecor and Cochran (Iowa State University Press, Iowa, 1980)

Multiple Regression in Behavioral Research: Explanation and Prediction, by Elazar J. Pedhazur (Holt, Rinehart and Winston, New York, 1982)

Experimental Design in Psychology Research, by Edwards (Harper & Row, New York, 1975)

Multivariate Statistics in Behavioral Research, by R. Darrell Bock (McGraw, New York, 1975)

Problems

1-1. We have collected the following data on five subjects:

ID	AGE	GENDER	Grade Point Average (GPA)	College Entrance Exam Score (CSCORE)
1	18	M	3.7	650
2	18	F	2.0	490
3	19	F	3.3	580
4	23	M	2.8	530
5	21	M	3.5	640

(a) Write the SAS statements necessary to create a SAS data set.

(b) Add the statement(s) necessary to compute the mean grade point average and mean college entrance exam score.

(c) We want to compute an index for each subject, as follows:

```
INDEX = GPA + 3 x CSCORE/500
```

Modify your program to compute this INDEX for each student and to print a list of students in order of increasing INDEX. Include in your listing the student ID, GPA, CSCORE, and INDEX.

1-2. Given the following set of data:

Social Security Number	Annual Salary	Age	Race
123874414	28,000	35	W
646239182	29,500	37	B
012437652	35,100	40	W
018451357	26,500	31	W

(a) Create a SAS data set using the data above. Compute the average annual salary and average age. NOTE: Since you don't know yet how to read numeric values containing commas, you may enter the values without commas.

(b) If all subjects were in a 30% tax bracket, compute their taxes (based on annual salary) and print out a list, in Social Security number order, showing the annual salary and the tax.

1-3. What's wrong with the following program?

```
DATA MISTAKE;
   INPUT ID 1-3 TOWN 4-6 REGION 7-9 YEAR 11-12 BUDGET 13-14
   VOTER TURNOUT 16-20
(data lines)
;
PROC MEANS DATA=MISTAKE;
   VAR ID REGION VOTER TURNOUT;
   N,STD,MEAN;
RUN;
```

*1-4. A large corporation is interested in who is buying their product. What the CEO wants is a profile of the "typical buyer." The variables collected on a sample of buyers are: age, gender, race, income, marital status, and homeowner/ renter. Set up a layout for the observations, and write a SAS program to obtain a profile of the "typical buyer." NOTE: Slightly more difficult problems are preceded by an asterisk (*).

HINTS AND COMMENTS:

(1) For variables such as "homeowner," it is easier to remember what you have if you let negative responses be 0 and positive responses be 1.

(2) When grouping numerical variables into categories, make sure your grouping suits your needs and your data. For example, if your product is denture cream, a grouping of age,

1 = < 21 2 = 21–35 3 = 36–50 4= > 50

is virtually useless. You know such people are mostly over 50. You might use groupings such as

1 = < 50, 2 = 50–59 3 = 60–69 4 = > 69.

1-5. Given, the data set:

ID	RACE	SBP	DBP	HR
001	W	130	80	60
002	B	140	90	70
003	W	120	70	64
004	W	150	90	76
005	B	124	86	72

(NOTE: SBP is systolic blood pressure, DBP is diastolic blood pressure, and HR is heart rate.)

Write the SAS statements to produce a report as follows:

Race and Hemodynamic Variables

ID	RACE	SBP	DBP
003	W	120	70
005	B	124	86
001	W	130	80
002	B	140	90
004	W	150	90

NOTE: 1. To omit the default "OBS" column, use the PROC PRINT option NOOBS.
2. Data is in increasing order of SBP.
3. The variable HR is not included in the report.
4. The report has a title.

1-6. Given the data set of problem 1-5, modify that program to compute the "average" blood pressure. (ABP) defined as a weighted average of the diastolic blood pressure and the systolic blood pressure. Since the heart spends more time in its relaxed state (diastole), the diastolic pressure is weighted two-thirds, and the systolic blood pressure is weighted one-third. Therefore, the average blood pressure could be computed by multiplying the diastolic blood pressure by 2/3, and the systolic blood pressure by 1/3 and adding the two. An equivalent expression would be the diastolic pressure plus one-third of the difference between the systolic and diastolic pressures. Using either definition, add APB to the data set.

CHAPTER

2

Describing Data

A. *Introduction*

Did you work the problems at the end of Chapter 1 as we asked you? We didn't think so. So, go back and do them now, and then send in a check to your local Public Broadcasting station.

One of the first steps for any data analysis project is to generate simple descriptive statistics for all the continuous variables. Besides the traditional mean and standard deviation, you may want to use histograms, stem-and-leaf plots, test for normality of distributions, and a variety of other descriptive measures.

B. *Describing Data*

Even for complex statistical analysis, you must be able to describe the data in a straightforward, easy-to-comprehend fashion. This is typically accomplished in one of several ways. The first way is through descriptive summary statistics. Probably the most popular way to describe a sample of scores is by reporting: (1) the number of people in the sample (called the sample size and referred to by "n" in statistics books and SAS printouts); (2) the mean (arithmetic average) of the scores; and (3) the standard deviation of the scores. The standard deviation is a measure of how widely

spread the scores are. Roughly speaking, if the scores form a "bell-shaped" (normal) distribution, about 68% of the scores will fall within 1 standard deviation of the mean (plus or minus) and 95% of the scores within 2 standard deviations.

Let's create a SAS data set to introduce some concepts related to descriptive statistics. Suppose we conducted a survey (albeit a very small one) where we record the gender, height, and weight of seven subjects. We collect the following data:

GENDER	HEIGHT	WEIGHT
M	68.5	155
F	61.2	99
F	63.0	115
M	70.0	205
M	68.6	170
F	65.1	125
M	72.4	220

There may be several questions we want to ask about these data. Perhaps we want to count how many males and females are in our sample. We might also want means and standard deviations for the variables HEIGHT and WEIGHT. Finally, we may want to see a histogram of our continuous variables and, perhaps, determine if the data can be considered to have come from a normal distribution. These are fairly simple tasks if only seven people are involved. However, we are rarely interested in such small samples. Once we begin to consider as many as 20–30 people, statistical analysis by hand becomes quite tedious. A SAS program to read these data and to compute some descriptive statistics is shown next:

```
DATA HTWT; ①
   INPUT SUBJECT GENDER $ HEIGHT WEIGHT; ②
DATALINES; ③
1 M 68.5 155
2 F 61.2  99
3 F 63.0 115
4 M 70.0 205
5 M 68.6 170
6 F 65.1 125
7 M 72.4 220
;
PROC MEANS DATA=HTWT; ④
   TITLE 'Simple Descriptive Statistics'; ⑤
RUN; ⑥
```

Once again, in this example, our data are placed "in-stream" directly into our program. For small data sets, this is an appropriate method. For larger data sets, we usually place our data in a separate file and instruct our SAS program where to look to find the data (see Chapter 13, Section C). In this program, we have chosen to use the list form of input where each data value is separated from the next by one or more spaces. The first three lines define our DATA step. In ①, we indicate that we are creating a SAS data set called HTWT. As mentioned in the tutorial, SAS data set names as well as SAS variable names are from one to eight characters in length. They must start with a letter or underscore(_). The other characters in SAS data set names or SAS variable names can, in addition, include numerals. Thus, our name HTWT meets these criteria and is a valid SAS data set name. Statement ② is the INPUT statement. This statement gives names to the variables we are going to read. In this form of INPUT, the order of the variable names corresponds to the order of the data values. Since our data values are arranged in SUBJECT, GENDER, HEIGHT, and WEIGHT order, our INPUT statement lists variable names in the same order. If we had any missing values in our data, with this form of input we would have had to use a period (.) to hold the place of a missing value. Again, in review, the $ following the variable name GENDER indicates that we are using character values for GENDER ('M' and 'F'). Statement ③, DATALINES, indicates that the in-stream lines of data will follow and that the DATA step is finished. Following the data is a lone semicolon (;), which is a convenient way to indicate there are no more data lines. Statement ④ is our request for descriptive statistics. Since we did not tell it to do otherwise, PROC MEANS will give us the number of observations used to calculate the descriptive statistics for each of our numeric variables, the mean, standard deviation, minimum, and maximum. In a moment, we will show you how to request only those statistics that you want rather than accepting the defaults. Let's look at the results of running this program:

```
Simple Descriptive Statistics

Variable  N        Mean        Std Dev       Minimum       Maximum
-------------------------------------------------------------------
SUBJECT   7     4.0000000     2.1602469     1.0000000     7.0000000
HEIGHT    7    66.9714286     4.0044618    61.2000000    72.4000000
WEIGHT    7   155.5714286    45.7961321    99.0000000   220.0000000
-------------------------------------------------------------------
```

For the variable of height in our sample, we see that there are seven people (n = 7); that the shortest person is 61.2 inches tall and the tallest 72.4 inches (from the "minimum value" and "maximum value" columns); their mean height is 66.971 (rounded off); that the standard deviation is 4.004. Notice that we also obtain statistics on the variable called SUBJECT. The mean subject number is probably not going to be of interest to us. We will show you, shortly, how to compute statistics only for those variables of interest.

You can specify which statistics you want to compute by specifying options for PROC MEANS. Most SAS procedures have options which are placed between the procedure name and the semicolon. Many of these options are listed in this text; a complete list of options for all SAS procedures can be found in the manuals available from the SAS Institute. As mentioned in Chapter 1, the option MAXDEC=n controls the number of decimal places for the printed statistics, N prints the number of (nonmissing) observations, and MEAN produces the MEAN. So, if you want only the N and MEAN and you want three places to the right of the decimal, you would write:


```
PROC MEANS DATA=HTWT N MEAN MAXDEC=3;
```

You may also wish to specify for which numeric variables in your data set you want to compute descriptive statistics (so we can avoid getting descriptive statistics on numeric variables such as SUBJECT). We make this specification with a VAR statement. VAR (short for VARIABLES) is a statement that gives additional information to PROC MEANS (and many other procedures as well). The syntax is the word VAR followed by a list of variable names. So, if we want descriptive statistics only on HEIGHT and we want the sample size, the mean, and three decimal places, we would write:

```
PROC MEANS DATA=HTWT N MEAN MAXDEC=3;
   TITLE 'Simple Descriptive Statistics';
   VAR HEIGHT;
RUN;
```

The order of the options is irrelevant. A list of the commonly requested options for PROC MEANS is as follows:

Option	Description
N	Number of observations for which the statistic was computed
NMISS	Number of observations with missing values for the variable of interest
MEAN	Arithmetic mean
STD	Sample standard deviation
STDERR	Standard error
CLM	Lower and upper two-sided 95% confidence interval (CI) for the mean
LCLM	Lower one-sided 95% CI for the mean. If both LCLM and UCLM are requested, a two-sided CI is computed; otherwise, this option gives you a one-sided interval.
UCLM	Upper one-sided 95% CI for the mean. If both LCLM and UCLM are requested, a two-sided CI is computed; otherwise, this option gives you a one-sided interval.
MIN	Minimum: lowest score for the data
MAX	Maximum: highest score for the data
SUM	Sum
VAR	Variance
CV	Coefficient of variation
SKEWNESS	Skewness
KURTOSIS	Kurtosis
T	Student's t-test, testing the null hypothesis that the population mean is zero.
PRT	Probability of obtaining a larger absolute value of t under the null hypothesis.
MAXDEC=n	Where n specifies the number of decimal places for printed statistics

One further example. Suppose we want the sample size, mean, standard deviation, standard error, and a 95% confidence interval about the sample mean. In addition, we want the statistics rounded to two decimal places. We would write:

```
PROC MEANS DATA=HTWT MAXDEC=2 N MEAN STD STDERR CLM;
```

Output from this request would look as follows:

```
Simple Descriptive Statistics          13:46 Monday, June 17,  1996  10

Variable  N          Mean        Std Dev      Std Error
--------------------------------------------------------
HEIGHT    7         66.97          4.00          1.51
WEIGHT    7        155.57         45.80         17.31
--------------------------------------------------------

Variable   Lower 95.0% CLM   Upper 95.0% CLM
--------------------------------------------
HEIGHT              63.27             70.67
WEIGHT             113.22            197.93
--------------------------------------------
```

The standard error of the mean is used to indicate a "confidence interval" about the mean, which is useful when our scores represent a sample of scores from some population. For example, if our seven subjects were a random sample of high school juniors in New Jersey, we could use the sample mean (66.97) as an estimate of the average height of all New Jersey high school juniors. The standard error of the mean tells us how far off this estimate might be. If our population is roughly normally distributed, the sample estimate of the mean (based on a random sample) will fall within one standard error (1.51) of the actual, or "true," mean 68% of the time and within two standard errors (3.02) of the mean 95% of the time. By specifying the CLM option, PROC MEANS will print a 95% confidence interval about our sample mean. Looking at our listing, we are 95% "confident" that the interval from 63.27 to 70.67 contains the true population mean for height. For weight, this interval is from 113.22 to 197.93. If you were to compute a 95% confidence interval by hand, you would need to add and subtract a t-value (based on the degrees of freedom) times the standard error of the mean.

As mentioned earlier in this chapter, PROC MEANS produces, by default, the N, mean, minimum, maximum, and standard deviation. Suppose you want standard error added to that list. If you request any statistic (MAXDEC= is not a statistic), PROC MEANS will print only that statistic. Therefore, if you decide to override the system defaults and request an additional statistic, you must specify them all. As an example, to add standard error to the default list we would write:

```
PROC MEANS DATA=HTWT N MEAN STD MIN MAX STDERR;
```

C. More Descriptive Statistics

If you would like a more extensive list of statistics, including tests of normality, stem-and-leaf plots, and box plots, PROC UNIVARIATE is the way to go. This extremely useful procedure can compute, among other things:

1. The number of observations (nonmissing)
2. Mean
3. Standard deviation
4. Variance
5. Skewness
6. Kurtosis
7. Uncorrected and corrected sum of squares
8. Coefficient of variation
9. Standard error of the mean
10. A t-test comparing the variable's value against zero
11. Maximum (largest value)
12. Minimum (smallest value)
13. Range
14. Median, 3rd, and 2nd quartiles
15. Interquartile range
16. Mode
17. 1st, 5th, 10th, 90th, 95th, and 99th percentiles
18. The five highest and five lowest values (useful for data checking)
19. W or D statistic to test whether data are normally distributed
20. Stem-and-leaf plot
21. Boxplot
22. Normal probability plot, comparing your cumulative frequency distribution to a normal distribution

To run PROC UNIVARIATE for our variables HEIGHT and WEIGHT, we would write:

```
PROC UNIVARIATE DATA=HTWT;
   TITLE 'More Descriptive Statistics';
   VAR HEIGHT WEIGHT;
RUN;
```

By default, we get the first 18 items of the list of statistics above. To request, additionally, a test of normality, a stem-and-leaf plot, and a box plot, we would add the options NORMAL and PLOT as follows:

```
PROC UNIVARIATE DATA=HTWT NORMAL PLOT;
   TITLE 'More Descriptive Statistics';
   VAR HEIGHT WEIGHT;
RUN;
```

A portion of the output from the request above is shown next:

More Descriptive Statistics
Univariate Procedure
Variable=HEIGHT

Moments

N	7	Sum Wgts	7
Mean	66.97143	Sum	468.8
Std Dev	4.004462	Variance	16.03571
Skewness	-0.23905	Kurtosis	-1.16132
USS	31492.42	CSS	96.21429
CV	5.979358	Std Mean	1.513544
T:Mean=0	44.24808	Pr >|T|	0.0001
Num ^= 0	7	Num > 0	7
M(Sign)	3.5	Pr >= |M|	0.0156
Sgn Rank	14	Pr >= |S|	0.0156
W:Normal	0.954727	Pr < W	0.7849

Quantiles(Def=5)

100% Max	72.4	99%	72.4
75% Q3	70	95%	72.4
50% Med	68.5	90%	72.4
25% Q1	63	10%	61.2
0% Min	61.2	5%	61.2
		1%	61.2
Range	11.2		
Q3-Q1	7		
Mode	61.2		

Extremes

Lowest	Obs	Highest	Obs
61.2(	2)	65.1(	6)
63(	3)	68.5(	1)
65.1(	6)	68.6(	5)
68.5(	1)	70(	4)
68.6(	5)	72.4(	7)

More Descriptive Statistics
Univariate Procedure
Variable=HEIGHT

```
Stem Leaf                        #  Boxplot
  72 4                           1     |
  70 0                           1  +-----+
  68 56                          2  *-----*
  66                                |  +  |
  64 1                           1  |     |
  62 0                           1  +-----+
  60 2                           1     |
     ----+----+----+----+
```


[Continued]

```
                    Normal Probability Plot
73+                                               +*+++
   |                                           *++++
   |                                   *  +*+++
67+                                 +++++
   |                            +++*+
   |                        ++++*
61+                    +++*+
    +----+----+----+----+----+----+----+----+----+----+
        -2        -1         0        +1        +2
```

More Descriptive Statistics
Univariate Procedure
Variable=WEIGHT

Moments

N	7	Sum Wgts	7
Mean	155.5714	Sum	1089
Std Dev	45.79613	Variance	2097.286
Skewness	0.278915	Kurtosis	-1.46723
USS	182001	CSS	12583.71
CV	29.43737	Std Mean	17.30931
T:Mean=0	8.987731	Pr > |T|	0.0001
Num ^= 0	7	Num > 0	7
M(Sign)	3.5	Pr >= |M|	0.0156
Sgn Rank	14	Pr >= |S|	0.0156
W:Normal	0.941255	Pr < W	0.6674

Quantiles(Def=5)

100% Max	220	99%	220
75% Q3	205	95%	220
50% Med	155	90%	220
25% Q1	115	10%	99
0% Min	99	5%	99
		1%	99
Range	121		
Q3-Q1	90		
Mode	99		

Extremes

Lowest	Obs	Highest	Obs
99(	2)	125(	6)
115(	3)	155(	1)
125(	6)	170(	5)
155(	1)	205(	4)
170(	5)	220(	7)

[Continued]

```
More Descriptive Statistics
Univariate Procedure
Variable=WEIGHT

  Stem Leaf                    #  Boxplot
    22 0                       1     |
    20 5                       1  +-----+
    18                            |     |
    16 0                       1  |     |
    14 5                       1  *--+--*
    12 5                       1  |     |
    10 5                       1  +-----+
     8 9                       1     |
       ----+----+----+----+
  Multiply Stem.Leaf by 10**+1


                     Normal Probability Plot
  230+                                              *++++
     |                                         * +++++
     |                                       ++++
     |                                  +++*+
     |                              +++*
     |                          +++*
     |                      ++++*
   90+                  ++*+
      +----+----+----+----+----+----+----+----+----+----+
          -2        -1         0        +1        +2
```

There is a lot of information in a PROC UNIVARIATE output. Under the title "Moments" you will see a number of statistics. Most of them are self explanatory. Here is a list with explanations:

N	Number of nonmissing observations
Sum Wgts	Sum of weights (if WEIGHT statement used)
Mean	Arithmetic mean
Sum	Sum of the scores
Std Dev	Standard deviation
Variance	Variance
Skewness	Skewness (measure of the symmetry or asymmetry of the distribution)
Kurtosis	Kurtosis (measure of the flatness or the distribution)
USS	Uncorrected sum of squares (the sum of the scores squared—each score is squared and the squares are added together)
CSS	Corrected sum of squares (sum of squares about the mean, usually more useful then USS)
CV	Coefficient of variation
Std Mean	Standard error of the mean (the standard deviation divided by the square root of n)

T:Mean=0	Student's t-test for testing the hypothesis that the population mean is zero.
Prob>|T|	The p-value for the t-statistic (two-tailed)
Sgn Rank	The Wilcoxon signed rank sum (usually used for difference scores)
Prob>|S|	The p-value for the Sign Rank test
Num ^= 0	Number of nonzero observations
W:Normal (D:Normal)	Shapiro-Wilk statistic for a test of normality (SAS will produce the Kolomogorov D:Normal test when n is larger than 2000)
Prob<W (Prob>D)	P-value testing the null hypothesis that the population is normally distributed (when the D:Normal test is done, the statistic is Prob>D)

Looking further at the output, we find, under the heading "Quantiles(Def=5)," two sets of useful information. The left-hand set lists quantiles, showing the highest score (100%), the score at the third quartile (75%), the median (50%), the score at the first quartile, and the lowest score (0%). We point out that this is the only place we know of where you can compute a median. (Wouldn't a median be a nice option for PROC MEANS?) The "Def=5" in the heading indicates that SAS is using definition 5 listed in the SAS Procedures manual. This definition is described as an "empirical distribution function with averaging." We refer those interested in the subtle, yet fascinating differences of the five available definitions, to consult the SAS Procedures manual under PROC UNIVARIATE. For the other 99.6% of our readers, we recommend DEF=5.

The right-hand column lists other percentiles that are often of interest (99%, 95%, etc.).

Below these two columns you will find the range, the interquartile range (Q3-Q1), and the mode.

The list of Extremes, which comes next, is extremely useful (sorry, bad pun, but an excellent mnemonic). This list also comes without our specifically asking for it. It lists the five lowest and five highest values in the data set. We find this useful for data checking. Obviously incorrect values can be spotted easily. Next to each extreme value is the corresponding observation number. This can be made more useful if an ID statement is used with PROC UNIVARIATE. The ID variable (usually a subject number) is printed next to each extreme value, instead of the observation number. That way, when a data error is spotted, it is easier to locate the incorrect value in the data by referring to the ID variable. We can use our variable SUBJECT as an ID variable to demonstrate this. Our complete PROC UNIVARIATE request would be:

```
PROC UNIVARIATE DATA=HTWT NORMAL PLOT;
   TITLE 'More Descriptive Statistics';
   VAR HEIGHT WEIGHT;
   ID SUBJECT;
RUN;
```

The next portion of the output from PROC UNIVARIATE is a result of the PLOT option we used. The left side of the page is a Tukey-style stem-and-leaf plot. This can be thought of as a sideways histogram. However, instead of using X's to represent the bars, the next digit of the number after the "stem" is used. For example, the smallest height is 72.4. Next to the stem value of 72, we see a '4', which tells us not only is there a value between 72 and 74 but that the actual value is 72.4. Look at the stem labeled "68." Notice that there were two scores between 68 and 70. The "leaf" values of '5' and '6' indicate that there is one height equal to 68.5 and another height equal to 68.6. This style of "histogram" supplies us with additional information concerning the structure of values within a bar. In this small example, the stem-and-leaf plot is not too impressive, but with samples from about 20 to 200, it is very useful.

Another Tukey invention, the boxplot, is displayed on the right side of the page. The SAS conventions are as follows: The bottom and top of the box represent the sample 25th and 75th percentiles. The median is represented by the dashed line inside the box, and the sample mean is shown as a + sign. The vertical lines at the top and bottom of the box are called whiskers and extend as far as the data, to a maximum distance of 1.5 times the interquartile range. Data values beyond the whiskers, but within three interquartile ranges of the box boundaries, are shown with o's. Still more extreme values are shown with an asterisk (*). The purpose of the boxplot is to present a picture of the data and is useful for comparing several samples side-by-side. In the next section we show you how to produce descriptive statistics broken down by one or more variables. When you do this with PROC UNIVARIATE (using a BY statement), you will obtain side-by-side boxplots automatically, for each value of the BY variable or each combination of the BY variables if you have more than one.

The Normal Probability Plot, shown last, is also a result of the PLOT option and represents a plot of the data compared to a normal distribution. The y-axis displays our data values, and the x-axis is related to the inverse of the standard normal function. The asterisks (*) mark the actual data values, the plus signs (+) provide a reference straight line based on the sample mean and standard deviation. Basically, as the sample distribution deviates from the normal, the more the asterisks deviate from the plus signs. You will, no doubt, find the stem-and-leaf and boxplots a more intuitive way to inspect the shape of the distribution.

D. Descriptive Statistics Broken Down by Subgroups

Instead of computing descriptive statistics for every subject in your data set, you may want these statistics for specific subgroups of your data. For example, in the data set of Section B, you might want the number, the mean, and the standard deviation of HEIGHT and WEIGHT separately for males and females. We can accomplish this two ways. First, we can sort the data set by GENDER and then include a BY statement with PROC MEANS, like this:

```
*Sort the data by GENDER;
PROC SORT DATA=HTWT;
   BY GENDER;
RUN;

*Run PROC MEANS for each value of GENDER;
PROC MEANS DATA=HTWT N MEAN STD MAXDEC=2;
   BY GENDER;  *This is the statement that gives the breakdown;
   VAR HEIGHT WEIGHT;
RUN;
```


The resulting output is shown below:

```
GENDER=F

Variable  N          Mean       Std Dev
---------------------------------------
HEIGHT    3         63.10          1.95
WEIGHT    3        113.00         13.11
---------------------------------------

 GENDER=M

Variable  N          Mean       Std Dev
---------------------------------------
HEIGHT    4         69.88          1.82
WEIGHT    4        187.50         30.14
---------------------------------------
```

Most SAS procedures allow you to include a BY statement that runs the procedure for each level of the BY variable. Remember to always sort the data set first by the same BY variable (or know in advance that the data set is already sorted). Sometimes in a long program, you ask for a data set to be sorted which you previously sorted (because you sorted it earlier and forgot that you did it). In SAS versions 6.08 and above, if you request a sort on a data set that has been previously sorted by SAS, the sort will not be performed, and you will be notified in the SAS LOG that the data set was already sorted in the order indicated.

An alternative to using a BY statement with PROC MEANS is the use of a CLASS statement. If you use a CLASS statement instead of a BY statement, the printed output will be similar, as you can see in the output below where a CLASS statement is used:

```
Simple Descriptive Statistics

GENDER    N Obs  Variable  N         Mean       Std Dev
-------------------------------------------------------
F             3  HEIGHT    3        63.10          1.95
                 WEIGHT    3       113.00         13.11

M             4  HEIGHT    4        69.88          1.82
                 WEIGHT    4       187.50         30.14
-------------------------------------------------------
```

The advantage of using a CLASS statement is that you do not have to sort the data set first. For large data sets, this can mean a large saving of processing time (and possibly money). On the down side, use of a CLASS statement requires considerably more memory than a BY statement, especially if there are several CLASS variables and many levels of each. In general, try using a CLASS statement first, and resort to use of a BY statement only if memory limitations force you to. We amplify the discussion about using a CLASS statement with PROC MEANS in Section H of this chapter.

E. Frequency Distributions

Let's look at how to get SAS software to count how many males and females there are in our sample. The following SAS program does this:

```
DATA HTWT;
   INPUT SUBJECT GENDER $ HEIGHT WEIGHT;
DATALINES;
1 M 68.5 155
2 F 61.2  99
3 F 63.0 115
4 M 70.0 205
5 M 68.6 170
6 F 65.1 125
7 M 72.4 220
;
PROC FREQ DATA=HTWT;
   TITLE 'Using PROC FREQ to Compute Frequencies';
   TABLES GENDER;
RUN;
```

This time, instead of PROC MEANS, we are using a procedure (PROC) called FREQ. PROC FREQ is followed by a request for a table of frequencies for the variable GENDER. The word TABLES (or TABLE), used with PROC FREQ, is followed by a list of variables for which we want to count occurrences of particular values (e.g., how many males and how many females for the variable GENDER). Be sure to include a TABLES statement and to list only variables with a reasonable number of levels when you use PROC FREQ. That is, don't ask for frequencies on continuous variables such as AGE since you will get the frequency for every distinct value. This may kill several trees.

Note that PROC FREQ does not use a VAR statement to specify on which variables to compute frequencies (as we did with PROC MEANS and PROC UNIVARIATE). This may seem inconsistent to you. You need to learn (or look up) the appropriate statements that can be used with each particular procedure. Chapter 20 includes some syntax examples that may help you.

Notice that the two statements, TITLE and TABLES, are indented several spaces right of the other lines. The starting column of any SAS statement does not affect the program in any way. These lines are indented only to make it clear to the programmer that the TITLE and TABLES statements are part of PROC FREQ.

The next table shows the output from PROC FREQ. The column labeled "FREQUENCY" lists the number of people who are males or females; the column labeled "PERCENT" is the same information expressed as a percentage of the total number

of people. The "CUM FREQ" and "CUM PERCENT" columns give us the cumulative counts (the number and percentage respectively) for each category of gender.

```
Using PROC FREQ to compute frequencies
                                     Cumulative    Cumulative
GENDER    Frequency    Percent    Frequency     Percent
---------------------------------------------------------
  F           3          42.9         3            42.9
  M           4          57.1         7           100.0
```

If you do not need (or want) the cumulative statistics that are automatically produced by a TABLES request, you may use the NOCUM statement option to omit the cumulative statistics. To review, statement options follow a slash (/) after the appropriate statement. Thus, to omit cumulative statistics, you would write:

```
TABLES GENDER / NOCUM;
```

To omit both cumulative statistics and percentages you would include the NOPERCENT TABLES option as well, like this:

```
TABLES GENDER / NOCUM NOPERCENT;
```

The order in which you place these options does not matter. The output from the request above is shown next:

```
Using PROC FREQ to compute frequencies

GENDER     Frequency
--------------------
   F           3
   M           4
```

F. Bar Graphs

We have seen the statistics that are produced by running PROC MEANS and PROC FREQ. It is an excellent way to get a summarization of our data. But then, a picture is worth a thousand words (p=1000w) so let's move on to presenting pictures of our data. SAS software can generate a frequency bar chart showing the same information as PROC FREQ, using PROC CHART.

The statements:

```
PROC CHART DATA=HTWT;
   VBAR GENDER;
RUN;
```

were used to generate the frequency bar chart shown below:

```
Frequency

4 +                    *****
  |                    *****
  |                    *****
  |                    *****
3 +         *****      *****
  |         *****      *****
  |         *****      *****
  |         *****      *****
2 +         *****      *****
  |         *****      *****
  |         *****      *****
  |         *****      *****
1 +         *****      *****
  |         *****      *****
  |         *****      *****
  |         *****      *****
  ----------------------------
              F          M

               GENDER
```

The term HBAR in place of VBAR will generate a chart with horizontal bars instead of the vertical bars obtained from VBAR. When HBAR is used, frequency counts and percents are also presented alongside each bar (see below). Also, the HBAR approach often allows for more groups to be presented as it takes up less space.

```
FREQUENCY BAR CHART

GENDER                                FREQ  CUM.  PERCENT   CUM.
                                            FREQ           PERCENT
  |
F | ***************                     3     3    42.86    42.86
  |
M | ********************                4     7    57.14   100.00
  |
  ----+----+----+----+
      1    2    3    4

        FREQUENCY
```

Now, what about the distribution of heights or weights? If we use PROC FREQ to calculate frequencies of heights, it will compute the number of subjects for every

value of height (how many people are 60 inches tall, how many are 61 inches tall, etc.). If we use PROC CHART instead, it will automatically place the subjects into height groups (unless we specified options to control how we wanted the data displayed). Since our sample is so small, a frequency distribution of heights or weights will look silly, but we'll show it to you anyway. The SAS statements below will generate a vertical bar chart:

```
PROC CHART DATA=HTWT;
   TITLE 'Distribution of Heights';
   VBAR HEIGHT / LEVELS=6;
RUN;
```

The option LEVELS=6 is an instruction to group the heights so that there will be six equally spaced intervals for the variable HEIGHT. If we leave out any options when we are charting a continuous variable, PROC CHART will use its own grouping algorithm to select the number of levels and the midpoints for the plot. The output from the program above is:

```
Distribution of Heights

Frequency
  |
  |                             *****
2 +                             *****
  |                             *****
  |                             *****
  |                             *****
  |                             *****
1 +  *****    *****    *****    *****    *****    *****
  |  *****    *****    *****    *****    *****    *****
  |  *****    *****    *****    *****    *****    *****
  |  *****    *****    *****    *****    *****    *****
  |  *****    *****    *****    *****    *****    *****
  -------------------------------------------------------
      62       64       66       68       70       72

                      HEIGHT Midpoint
```

The VBAR and HBAR statements of PROC CHART have a variety of options. The general form of the VBAR and HBAR statements is:

```
VBAR variable(s) / list of options ;
```

An alternative to the LEVELS= option provides the procedure with specified midpoints. This is done with the MIDPOINTS option. The form is:

```
MIDPOINTS= lower_limit TO upper_limit BY interval;
```

An example would be:

```
VBAR HEIGHT / MIDPOINTS=50 TO 80 BY 10;
```

There are times when we do not want PROC CHART to divide our numerical variable into intervals. Suppose we had a variable called WEEK that was a numeric variable and represented the day of the week (from 1 to 7). The statement:

```
VBAR WEEK;
```

would most likely produce a chart with midpoints that were not integers. To avoid this and instruct PROC CHART to use the actual values of a variable, the option DISCRETE is added to the option list. To be sure that the frequency chart for WEEK is printed correctly, the statement:

```
VBAR WEEK / DISCRETE;
```

should be used. (Remember that statement options are placed between the statement and the semicolon, separated by a slash.)

Before leaving PROC CHART, we demonstrate a few of the other options that are available. To do this, we have constructed another data set which contains the variables DEPT (department), YEAR, QUARTER, and SALES. The statement:

```
VBAR DEPT;
```

will produce a simple frequency bar graph as shown below:

```
Frequency
   |
   |      *****
   |      *****
10 +      *****
   |      *****
   |      *****         *****
   |      *****         *****
   |      *****         *****
 5 +      *****         *****
   |      *****         *****
   |      *****         *****
   |      *****         *****
   |      *****         *****
   --------------------------------
          FOOD          TOYS

                 DEPT
```

The statement:

```
VBAR SALES;
```

will produce a frequency distribution of SALES for all years and all departments.


```
Frequency
  |
7 +                         *****     *****
  |                         *****     *****
6 +                         *****     *****
  |                         *****     *****
5 +                         *****     *****
  |                         *****     *****
4 +                         *****     *****
  |                         *****     *****
3 +               *****     *****     *****
  |               *****     *****     *****
2 +               *****     *****     *****     *****
  |               *****     *****     *****     *****
1 +     *****     *****     *****     *****     *****
  |     *****     *****     *****     *****     *****
  ----------------------------------------------------
         1500      2500      3500      4500      5500

                        SALES Midpoint
```

To see the sales distributions of each department side-by-side, we can use the GROUP option available with both VBAR and HBAR. The statement:

```
VBAR SALES / GROUP=DEPT;
```

will produce the side-by-side graph like the one below:

```
Frequency
  |
6 +                                                  ***
  |                                                  ***
5 +          ***                                     ***
  |          ***                                     ***
4 +          ***                                     ***
  |          ***                                     ***
3 +      *** ***                                     ***
  |      *** ***                                     ***
2 +      *** ***     ***                         *** ***
  |      *** ***     ***                         *** ***
1 +  *** *** *** *** ***                         *** ***
  |  *** *** *** *** ***                         *** ***
  ---------------------------------------------------------
   1500 2500 3500 4500 5500  1500 2500 3500 4500 5500   SALES Midpoint

  |--------FOOD--------|    |--------TOYS--------|      DEPT
```

Another way to display these data is to have the y-axis represent a sales sum, rather than a frequency or count. This is done by using a SUMVAR option with VBAR or HBAR. The keyword SUMVAR is followed by a variable whose sum we want displayed on the y-axis. We also use the SUMVAR option to display a mean value on the y-axis by adding the keyword TYPE=MEAN to the list of VBAR or HBAR options. We will show you a chart using the SUMVAR and TYPE options. Since we are displaying a sum of sales for each department, the TYPE= option is redundant but is included to remind you that it is available to display other statistics on the y-axis. The statement:

```
VBAR DEPT / GROUP=YEAR SUMVAR=SALES TYPE=SUM;
```

produces the graph below:

```
SALES Sum

20000 +                                    *****
      |                                    *****
      |         *****             *****    *****
      |         *****    *****    *****    *****
      |         *****    *****    *****    *****
10000 + *****   *****    *****    *****    *****
      | *****   *****    *****    *****    *****
      | *****   *****    *****    *****    *****
      | *****   *****    *****    *****    *****
      | *****   *****    *****    *****    *****
      --------------------------------------------------------
        FOOD    TOYS     FOOD     TOYS     FOOD     TOYS    DEPT

        |---1988---|     |---1989---|      |---1990---|     YEAR
```

Other valid values for the TYPE= option are:

Option	Result
TYPE=FREQ	Frequency counts
TYPE=PCT	Percentages
TYPE=CFREQ	Cumulative frequencies
TYPE=CPCT	Cumulative percentages
TYPE=SUM	Totals
TYPE=MEAN	Means

One final option used with VBAR and HBAR is SUBGROUP. The first character of a SUBGROUP variable is used as the character making up the bars in the bar graph. If we write:

```
VBAR SALES / SUBGROUP=DEPT;
```

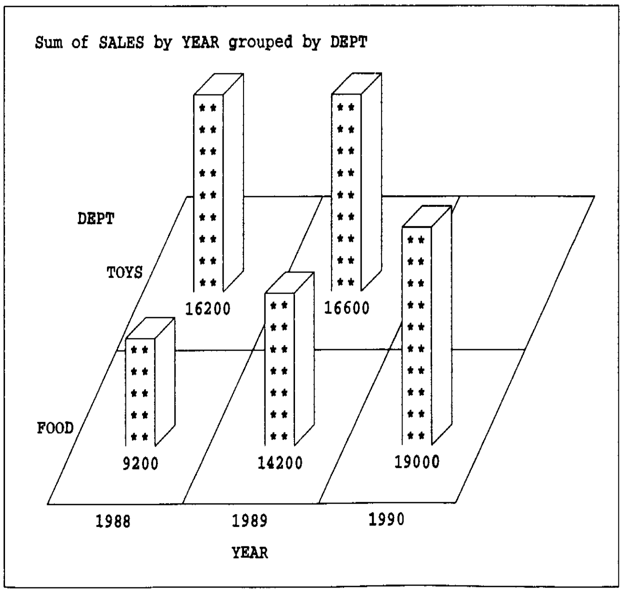
the department values (A's and B's) will show us which departments are contributing to the sales frequencies. See the chart below for an example:


```
Frequency

  |                          TTTTT    TTTTT
6 +                          TTTTT    TTTTT
  |                          FFFFF    TTTTT
4 +                          FFFFF    TTTTT
  |                 FFFFF    FFFFF    TTTTT
2 +                 FFFFF    FFFFF    TTTTT    FFFFF
  |       FFFFF     FFFFF    FFFFF    FFFFF    FFFFF
  -----------------------------------------------------
          1500      2500     3500     4500     5500

                       SALES Midpoint

Symbol DEPT     Symbol DEPT

  F    FOOD       T    TOYS
```

Before we leave PROC CHART, it's hard to resist showing you the fancy three-dimensional graphs produced by the BLOCK statement. The BLOCK variable defines the x-axis, the GROUP option defines the y-axis, and SUMVAR variables (with any of the TYPE= options) defines the z-axis, represented by the heights of the bars. The block chart resulting from the statement:

```
BLOCK YEAR / GROUP=DEPT SUMVAR=SALES TYPE=SUM DISCRETE;
```

is shown below:

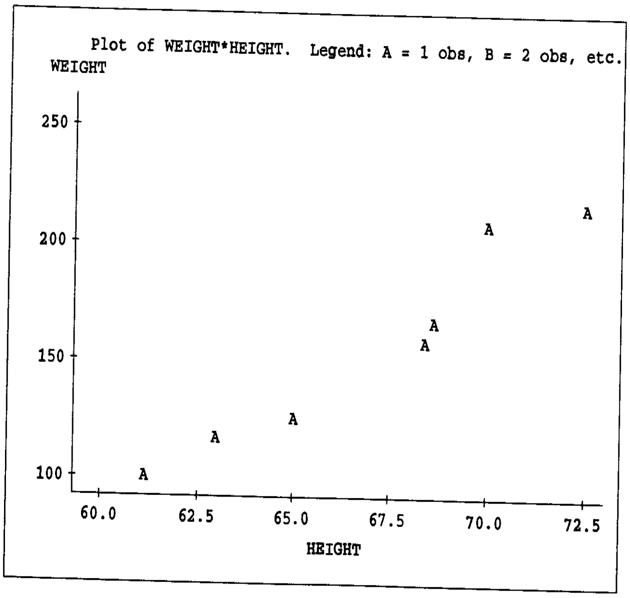


G. Plotting Data

We now investigate the relationship between height and weight. Our intuition tells us that these two variables are related: The taller a person is, the heavier (in general). The best way to display this relationship is to draw a graph of height versus weight. We can have our SAS program generate this graph by using PROC PLOT. The statements:

```
PROC PLOT DATA=HTWT;
   PLOT WEIGHT*HEIGHT;
RUN;
```

generate the graph that follows: (NOTE: This is a plot using the original data set of seven people.)



The general form of the plot procedure is:

```
PROC PLOT DATA=data_set_name;
   PLOT Y variable * X variable;
        (Vertical)     (Horizontal)
RUN;
```


Notice that SAS software automatically chooses appropriate scales for the x- and y-axes. Unless you specify otherwise, PROC PLOT uses letters (A, B, C, etc.) as plotting symbols. Since a computer line printer is restricted to printing characters in discrete locations across or down the page, two data values that are very close to each other would have to print in the same location. If two data points do occur at one print position, the program prints the letter "B"; for three data points, the letter "C," and so forth.

Can we obtain a plot of height versus weight for males, and one for females? The answer is "yes," and it is quite easy to do. Just as we used a BY variable with PROC MEANS earlier, we can use the same BY statement with PROC PLOT to create separate plots for males and females. First, we need to have the SAS program sort our data by GENDER. Once this is done, we can use PROC PLOT to produce the desired graphs.

Our program will look as follows:

```
DATA HTWT;
   INPUT SUBJECT GENDER $ HEIGHT WEIGHT;
DATALINES;
1 M 68.5 155
2 F 61.2  99
3 F 63.0 115
4 M 70.0 205
5 M 68.6 170
6 F 65.1 125
7 M 72.4 220
;
PROC SORT DATA=HTWT;
   BY GENDER;
RUN;

PROC PLOT DATA=HTWT;
   BY GENDER;
   PLOT WEIGHT*HEIGHT;
RUN;
```

The result of this program will be a separate graph for males and another for females. If we omit a BY statement with PROC PLOT, the program will ignore the fact that the data set is now sorted by GENDER.

We can generate another graph that displays the data for males and females on a single graph but, instead of the usual plotting symbols of A, B, C, etc., we will use F's and M's (for females and males). The statements:

```
PROC PLOT DATA=HTWT;
   PLOT WEIGHT*HEIGHT=GENDER;
RUN;
```

accomplishes this. The data set does not have to be sorted to use this form of PROC PLOT. The "=GENDER" after our plot request specifies that the first letter of each of the GENDER values will be used as plotting symbols. In essence, this allows us to look at three variables (height, weight, and gender) simultaneously. The result of running this last procedure is shown below:

```
          Plot of WEIGHT*HEIGHT.  Symbol is value of GENDER.
WEIGHT
    |
250 +
    |
    |
    |
    |                                                         M
200 +                                              M
    |
    |
    |                                        M
150 +                                       M
    |
    |                             F
    |                     F
100 +        F
    ---+---------+---------+---------+---------+---------+--
     60.0      62.5      65.0      67.5      70.0      72.5
                              HEIGHT
```

Since we are not using the standard plotting symbols (A, B, C, etc.), multiple observations at a single print location will not be shown on the graph (except in the case of one male and one female, in which case the M and F will overprint if the OVP option is set and you have an output device, such as a lineprinter, that can overprint). The program will print a message indicating the number of "hidden" observations, at the bottom of the graph in this case.

If you would like to choose a plotting symbol, instead of the SAS default of A, B, C, etc., you may follow the PLOT request by an equal sign and a plotting symbol of your choice in single quotes. If you wanted an asterisk as your plotting symbol, the plot request would read:

```
PLOT WEIGHT*HEIGHT='*';
```

As for the case of a variable name following the equal sign, choosing a plotting symbol will not allow hidden observations to be displayed, and you will see a message to that effect if there are any hidden observations.

H. *Creating Summary Data Sets with PROC MEANS and PROC UNIVARIATE*

Besides providing a printed output of descriptive statistics, broken down by one or more CLASS (or BY) variables, PROC MEANS and PROC UNIVARIATE can produce new SAS data sets containing any of the statistics produced by these procedures. This might be useful, for example, in educational research where the original data were collected on individual students, but you want to use classroom means as the unit of observation (to compare teachers), or in business (to compare sales from various quarters when the original data are collected daily). In medicine, you may have clinical data on patient visits, with a different number of visits for each patient, and want to compute patient means for later analysis.

To demonstrate how this is done, suppose we have collected data on several students. We have a student number, gender, the teacher's name, the teacher's age, and two test scores (a pre-test and a posttest). We use the following data for our example:

SUBJECT	GENDER	TEACHER	T_AGE	PRETEST	POSTTEST
1	M	JONES	35	67	81
2	F	JONES	35	98	86
3	M	JONES	35	52	92
4	M	BLACK	42	41	74
5	F	BLACK	42	46	76
6	M	SMITH	68	38	80
7	M	SMITH	68	49	71
8	F	SMITH	68	38	63
9	M	HAYES	23	71	72
10	F	HAYES	23	46	92
11	M	HAYES	23	70	90
12	F	WONG	47	49	64
13	M	WONG	47	50	63

NOTES: (1) T_AGE is the teacher's age. (2) In a "real" study, we would probably enter the teacher's name and age only once in a separate data set and combine that data set with the student data later on, saving some typing. However, for this example, it is simpler to include the teacher's age for every observation.

As a first step, let's see how we can compute the mean pre-test, post-test, and gain scores for each teacher. Look at the following program:

```
DATA SCHOOL;
   LENGTH GENDER $ 1 TEACHER $ 5;  ①
   INPUT SUBJECT GENDER $ TEACHER $ T_AGE PRETEST POSTTEST;
   GAIN = POSTTEST - PRETEST;
DATALINES;
(lines of data)
;
PROC MEANS DATA=SCHOOL N MEAN STD MAXDEC=2;
   CLASS TEACHER;
   TITLE 'Means Scores for Each Teacher';
   VAR PRETEST POSTTEST GAIN;
RUN;
```

This program is straightforward. The DATA step computes a gain score, and PROC MEANS requests statistics for each teacher by including TEACHER as a CLASS variable. The LENGTH statement ① is used to specify how many characters are needed for the alphanumeric variables GENDER and TEACHER (1 and 5, respectively). We include this because, with the space-between-the-values form of data entry, a default length of eight is used for all character variables. Here is the output:

Means Scores for Each Teacher

TEACHER	N Obs	Variable	N	Mean	Std Dev
BLACK	2	PRETEST	2	43.50	3.54
		POSTTEST	2	75.00	1.41
		GAIN	2	31.50	2.12
HAYES	3	PRETEST	3	62.33	14.15
		POSTTEST	3	84.67	11.02
		GAIN	3	22.33	22.59
JONES	3	PRETEST	3	72.33	23.46
		POSTTEST	3	86.33	5.51
		GAIN	3	14.00	26.00
SMITH	3	PRETEST	3	41.67	6.35
		POSTTEST	3	71.33	8.50
		GAIN	3	29.67	10.79
WONG	2	PRETEST	2	49.50	0.71
		POSTTEST	2	63.50	0.71
		GAIN	2	14.00	1.41

Instead of just printing out the results, what we want to do is to create a new data set that has TEACHER as the unit of observation instead of SUBJECT. In our example, we only have five teachers, but we might have 100 and they might be using different teaching methods, be in different schools, etc. To create the new data set, we do the following:

```
PROC MEANS DATA=SCHOOL NOPRINT NWAY;  ①
   CLASS TEACHER;
   VAR PRETEST POSTTEST GAIN;
   OUTPUT OUT=TEACHSUM  ②
         MEAN=M_PRE M_POST M_GAIN;
RUN;
*To get a list of what was produced and therefore what
 is contained in the data set TEACHSUM, add the following:;
PROC PRINT DATA=TEACHSUM;
   TITLE 'Listing of Data Set TEACHSUM';
RUN;
*Hey! This is a good example of why comments
 are useful. ;
```


The NOPRINT option on the first line ① tells the program not to print the results of this procedure (since we already have them from the last run—or, the listing would be too large to want to look at). We want the computed statistics (means in this case) in the new data set. To do this, we include an OUTPUT statement ② in PROC MEANS. The OUTPUT statement creates a new data set. We have to give it a name of our choosing (by saying OUT=TEACHSUM), tell it what statistics to put in it, and what names to give those statistics.

We can output any statistics available with PROC MEANS by using the PROC MEANS options (N, MEAN, STD, etc.) as keywords in the OUTPUT statement. These statistics will be computed for all the variables in the VAR list and broken down by the CLASS variable. Since we want only the score means in this new data set, we said "MEAN = M_PRE M_POST M_GAIN." These new variables represent the means of each of the variables listed in the VAR statement, in the same order the variables are listed. Thus, M_PRE will represent the mean value of PRETEST, M_POST will represent the mean value of POSTEST, and M_GAIN will represent the mean value of GAIN. You could have named these new variables MANNY, MOE, and JACK. SAS doesn't care what you call them. We used M_PRE, M_POST, and M_GAIN because it helps us remember that they represent Means of PREtest, POSTtest, and GAIN.

Finally, we need to explain the NWAY option in line ①. This tells the procedure to only give us results for each TEACHER (the CLASS variable) and not to include the grand mean in the new data set. Don't forget this. We will explain what happens if you leave this out later. Your new data set (the listing from PROC PRINT) will look like this:

Listing of Data Set TEACHSUM

OBS	TEACHER	_TYPE_	_FREQ_	M_PRE	M_POST	M_GAIN
1	BLACK	1	2	43.5000	75.0000	31.5000
2	HAYES	1	3	62.3333	84.6667	22.3333
3	JONES	1	3	72.3333	86.3333	14.0000
4	SMITH	1	3	41.6667	71.3333	29.6667
5	WONG	1	2	49.5000	63.5000	14.0000

Let's leave the explanations of the _TYPE_ variable for our next example. The variable _FREQ_ gives us the number of observations (missing or nonmissing) for each value of the CLASS variable. If you go back to the original data values, you will see that teacher BLACK had two students, HAYES three students, and so forth.

What if you wanted the teacher's age in this new data set (so you could compare age to gain score, for example)? This is easily accomplished by including an ID statement as part of PROC MEANS. So, to include the teacher's age in this data set, you would use the following code:

```
PROC MEANS DATA=SCHOOL NOPRINT NWAY;  ①
   CLASS TEACHER;
   ID T_AGE;
   VAR PRETEST POSTTEST GAIN;
   OUTPUT OUT=TEACHSUM  ②
          MEAN=M_PRE M_POST M_GAIN;
RUN;
```

The resulting data set (TEACHSUM) will now contain the variable T_AGE. As an alternative, you could have included both variables, TEACHER and T_AGE, as CLASS variables with the same result.

We now turn to a more complex example where we create an output data set, using PROC MEANS and a CLASS statement with more than one CLASS variable. Yes folks, hold on to your hats, this gets tricky. First, the raw data:

SUBJ	GENDER	REGION	HEIGHT	WEIGHT
01	M	North	70	200
02	M	North	72	220
03	M	South	68	155
04	M	South	74	210
05	F	North	68	130
06	F	North	63	110
07	F	South	65	140
08	F	South	64	108
09	F	South	.	220
10	F	South	67	130

Next, we create a SAS data set as follows:

```
DATA DEMOG;
  LENGTH GENDER $ 1 REGION $ 5;
   INPUT SUBJ GENDER $ REGION $ HEIGHT WEIGHT;
DATALINES;
01 M North 70 200
02 M North 72 220
03 M South 68 155
04 M South 74 210
05 F North 68 130
06 F North 63 110
07 F South 65 140
08 F South 64 108
09 F South  . 220
10 F South 67 130
;
```

To compute the number of subjects, the mean, and the standard deviation for each combination of GENDER and REGION, include a CLASS statement with PROC MEANS like this:


```
PROC MEANS DATA=DEMOG N MEAN STD MAXDEC=2;
   TITLE 'Output from PROC MEANS';
   CLASS GENDER REGION;
   VAR HEIGHT WEIGHT;
RUN;
```

Remember that you do not have to sort your data set when you use a CLASS statment with PROC MEANS. In this example we have two CLASS variables instead of one. The output from this procedure is shown next:

```
Output from PROC MEANS

GENDER  REGION  N Obs  Variable   N          Mean        Std Dev

F       North       2  HEIGHT     2         65.50           3.54
                       WEIGHT     2        120.00          14.14

        South       4  HEIGHT     3         65.33           1.53
                       WEIGHT     4        149.50          48.86

M       North       2  HEIGHT     2         71.00           1.41
                       WEIGHT     2        210.00          14.14

        South       2  HEIGHT     2         71.00           4.24
                       WEIGHT     2        182.50          38.89
```

Since we now have two CLASS variables, the requested statistics are computed for each combination of GENDER and REGION.

We first demonstrate what happens when we use PROC MEANS to create an output data set with GENDER and REGION as CLASS variables. Here is the code:

```
PROC MEANS DATA=DEMOG NOPRINT;  ①
   CLASS GENDER REGION;
   VAR HEIGHT WEIGHT;
② OUTPUT OUT=SUMMARY
      ③ MEAN=M_HEIGHT M_WEIGHT;
RUN;

***Add a PROC PRINT to list the observations in SUMMARY;
PROC PRINT DATA=SUMMARY;
   TITLE 'Listing of Data Set SUMMARY';
RUN;
```

Don't be confused by the three asterisks in the comment above. Remember that the first asterisk starts the comment, and the semicolon ends it. We added the other two asterisks to make the comment stand out better. (One of the programming challenged authors thought you might like to know this.)

As before, the NOPRINT option on the first line ① tells the procedure not to print any output. Rather, you want these values in a SAS data set. To do this, you add an OUTPUT statment ② to PROC MEANS. The OUTPUT statement allows you to create a new data set, to select which statistics to place in this data set, and what names to give to each of the requested statistics. The name of the output data set is placed after the OUT= keyword. The request to output means is indicated by the keyword MEAN= ③. The two variable names following the keyword MEAN= are names you choose to represent the mean HEIGHT and WEIGHT, respectively. The order of the names following MEAN= corresponds to the order of the variable names in the VAR statement. In this example, the variable M_HEIGHT will represent the mean height and the variable M_WEIGHT will represent the mean weight. Other keywords (chosen from the list of statistics available with PROC MEANS earlier in this chapter) can be used to output statistics such as standard deviation (STD=) or sums (SUM=).

Using a PROC PRINT with DATA=SUMMARY to see the contents of this new data set, we obtain the listing below:

Listing of Data Set SUMMARY

OBS	GENDER	REGION	_TYPE_	_FREQ_	M_HEIGHT	M_WEIGHT
1			0	10	67.8889	162.300
2		North	1	4	68.2500	165.000
3		South	1	6	67.6000	160.500
4	F		2	6	65.4000	139.667
5	M		2	4	71.0000	196.250
6	F	North	3	2	65.5000	120.000
7	F	South	3	4	65.3333	149.500
8	M	North	3	2	71.0000	210.000
9	M	South	3	2	71.0000	182.500

Besides the mean for each combination of GENDER and REGION, we see there are five additional observations and two addition variables, _TYPE_ and _FREQ_. Here's what they're all about. The first observation with a value of 0 for _TYPE_ is the mean of all nonmissing values (9 for HEIGHT and 10 for WEIGHT) and is called the grand mean. The two observations with _TYPE_ equal to 1 are the mean HEIGHT and WEIGHT for each REGION; the next two observations with _TYPE_ equal to two are the mean HEIGHT and WEIGHT for each GENDER. Finally, the last four observations with _TYPE_ equal to 3 are the means by GENDER and REGION (sometimes called cell means). This is getting complicated! Relax, there is actually a way to tell which _TYPE_ value corresponds to which breakdown of the data.

Let's look carefully at our CLASS statement. It is written:

```
CLASS GENDER REGION;
```

First, we count in binary (remember, 0, 1, 10, 11, 100, 101, etc.) and place the binary numbers below the CLASS variables like this:

CLASS GENDER REGION;

Binary		_TYPE_	Interpretation
0	0	0	Mean over all GENDERS and REGIONS
0	1	1	Mean for each value of REGION
1	0	2	Mean for each value of GENDER
1	1	3	Mean for each combination of GENDER and REGION (cell means)

Next, we can come up with a simple rule. Whenever the _TYPE_ value, written in binary gives you a "1" beneath a CLASS variable, the statistics are broken down by that variable. If we look at _TYPE_ = 1 we write that in binary (not too hard) as 01 and realize that the _TYPE_ = 1 statistics represent each REGION and so forth. Still confused? It's OK, this is not easy.

For most applications, you don't even need to look at the _TYPE_ values. Since most applications call for cell means (the values broken down by each of the class variables), you will want the highest value of the _TYPE_ variable. If you include the option NWAY on the PROC MEANS statement, only cell means will be output to the new data set. So, if you only want the mean HEIGHT and WEIGHT for each combination of GENDER and REGION, you would write the PROC MEANS statements like this:

```
PROC MEANS DATA=DEMOG NOPRINT NWAY;
   CLASS GENDER REGION;
   VAR HEIGHT WEIGHT;
   OUTPUT OUT=SUMMARY
          MEAN=M_HEIGHT M_WEIGHT;
RUN;

PROC PRINT DATA=SUMMARY;
   TITLE 'Listing of Data Set SUMMARY with NWAY Option';
RUN;
```

The resulting data set (shown below) contains only the _TYPE_ = 3 values.

Listing of Data Set SUMMARY with NWAY Option

OBS	GENDER	REGION	_TYPE_	_FREQ_	M_HEIGHT	M_WEIGHT
1	F	North	3	2	65.5000	120.0
2	F	South	3	4	65.3333	149.5
3	M	North	3	2	71.0000	210.0
4	M	South	3	2	71.0000	182.5

The value of the variable _FREQ_ is the number of observations (missing or nonmissing) in each subgroup. For example, there were two females from the North so _FREQ_ = 2 in observation 1 in the summary data set. If you need to know the number of nonmissing values that were used in computing the requested statistics, include a request for N= in your output data set. Let's demonstrate this with the code below:

```
PROC MEANS DATA=DEMOG NOPRINT NWAY;
   CLASS GENDER REGION;
   VAR HEIGHT WEIGHT;
   OUTPUT OUT=SUMMARY
          N=N_HEIGHT N_WEIGHT
          MEAN=M_HEIGHT M_WEIGHT;
RUN;

PROC PRINT DATA=SUMMARY;
   TITLE1 'Listing of Data Set SUMMARY with NWAY Option';
   TITLE2 'with Requests for N= and MEAN=';
RUN;
```

In this program, we have chosen the variable names N_HEIGHT and N_WEIGHT to represent the number of nonmissing observations. The resulting output, shown below, makes the difference between the value of _FREQ_ and the N= variable clear:

```
Listing of Data Set SUMMARY with NWAY Option
with Requests for N= and MEAN=

OBS GENDER REGION _TYPE_ _FREQ_ N_HEIGHT N_WEIGHT M_HEIGHT M_WEIGHT

 1    F    North    3      2       2        2      65.5000   120.0
 2    F    South    3      4       3        4      65.3333   149.5
 3    M    North    3      2       2        2      71.0000   210.0
 4    M    South    3      2       2        2      71.0000   182.5
```

Observe that the value for N_HEIGHT is 3 for females from the South, while the value of _FREQ_ is 4 (there was a missing HEIGHT for a female from the South).

Finally, if you use the NWAY option, there is not much need to keep the _TYPE_ variable in the output data set. You can use a DROP= data set option to omit this variable. The program, modified to do this, is shown below:

```
PROC MEANS DATA=DEMOG NOPRINT NWAY;
   CLASS GENDER REGION;
   VAR HEIGHT WEIGHT;
   OUTPUT OUT=SUMMARY(DROP=_TYPE_)
          N=N_HEIGHT N_WEIGHT
          MEAN=M_HEIGHT M_WEIGHT;
RUN;
```


Some lazy programmers sometimes omit the variable list following a request for a statistic when only one statistic is requested. For example, if you only want means for each combination of GENDER and REGION, you would write:

```
PROC MEANS DATA=DEMOG NOPRINT NWAY;
   CLASS GENDER REGION;
   VAR HEIGHT WEIGHT;
   OUTPUT OUT=SUMMARY(DROP=_TYPE_)
          MEAN=;
RUN;
```

Using this method, the variable names in the new summary data set will be the same as listed on the VAR statement. That is, the variable name representing the mean height will be HEIGHT, and the variable name representing the mean weight will be WEIGHT. This is probably a bad idea since you may get confused and not realize that a variable name represents a summary statistic and not the original value. (Actually that other author would not even put in (DROP=_TYPE_) since it takes up too much time and he doesn't mind the extra variable in the printout.)

I. *Outputting Statistics Other Than Means*

We saw that PROC MEANS can create an output data set, using an OUTPUT statement, which contains means for each variable in the VAR list for each level of the variables in the CLASS or BY statement. We used the keyword MEAN= to indicate that we wanted to output means. Any of the options that are available to be used with PROC MEANS (see Section B) can also be used to create variables in the data set created by PROC MEANS. For example, if we wanted our new data set to contain the mean, standard deviation, and maximum value, we would write:

```
PROC MEANS DATA=DEMOG NOPRINT NWAY;
   CLASS GENDER REGION;
   VAR HEIGHT WEIGHT;
   OUTPUT OUT =STATS
          MEAN=M_HEIGHT M_WEIGHT
          STD =S_HEIGHT S_WEIGHT
          MAX =MAX_HT MAX_WT;
RUN;
```

Notice that we MUST include a list of variable names after the keywords MEAN=, STD=, and MAX= since we need to have a different variable name for each of these statistics. The resulting data set (STATS) will include the variables GENDER and REGION as well as the variables representing the mean, standard deviation, maximum, and the two variables _TYPE_ and _FREQ_ created by the procedure.

J. Creating a Summary Data Set Containing a Median

Since PROC MEANS does not compute medians, you will need to use PROC UNIVARIATE if you want to construct a summary data set containing that statistic. The statements to create an output data set using PROC UNIVARIATE are identical to the ones you used with PROC MEANS except that you must use a BY statement rather than a CLASS statement since PROC UNIVARIATE does not support a CLASS statement (too bad).

The keywords used to output statistics when using PROC UNIVARIATE are the same ones listed in Section C of this chapter. To output both means and medians to a summary data set, you could write:

```
PROC SORT DATA=DEMOG;
   BY GENDER REGION;
RUN;

PROC UNIVARIATE DATA=DEMOG NOPRINT;
   BY GENDER REGION;
   VAR HEIGHT WEIGHT;
   OUTPUT OUT=SUM
          N      = N_HT N_WT
          MEDIAN = MED_HT MED_WT
          MEAN   = MEAN_HT MEAN_WT;
RUN;

PROC PRINT DATA=SUM;
   TITLE 'Listing of Data Set SUM';
RUN;
```

The resulting output data set is listed below:

```
Listing of Data Set SUM

OBS  GENDER  REGION  N_HT  N_WT  MEAN_HT  MEAN_WT  MED_HT  MED_WT

 1     F     North     2     2   65.5000   120.0    65.5   120.0
 2     F     South     3     4   65.3333   149.5    65.0   135.0
 3     M     North     2     2   71.0000   210.0    71.0   210.0
 4     M     South     2     2   71.0000   182.5    71.0   182.5
```


Problems

2-1. Add the necessary statements to compute the number of males and females in problem 1-1.

2-2. Given the data set from problem 1-2, use SAS software to compute the number of Whites (W) and Blacks (B). Use the appropriate option to omit cumulative statistics from the output.

2-3. Run the program below to create a SAS data set called PROB2_3, containing variables X, Y, Z, and GROUP:

```
DATA PROB2_3;
   LENGTH GROUP $ 1;
   INPUT X Y Z GROUP $;
DATALINES;
2 4 6 A
3 3 3 B
1 3 7 A
7 5 3 B
1 1 5 B
2 2 4 A
5 5 6 A
;
```

(a) Write the SAS statements to generate a frequency bar chart (histogram) for GROUP.

(b) Write the SAS statements to generate a plot of Y vs. X (with "Y" on the vertical axis and "X" on the horizontal).

(c) Write the SAS statements to generate a separate plot of Y vs. X for each value of the GROUP variables.

2-4. We have recorded the following data from an experiment:

SUBJECT	DOSE	REACT	LIVER_WT	SPLEEN
1	1	5.4	10.2	8.9
2	1	5.9	9.8	7.3
3	1	4.8	12.2	9.1
4	1	6.9	11.8	8.8
5	1	15.8	10.9	9.0
6	2	4.9	13.8	6.6
7	2	5.0	12.0	7.9
8	2	6.7	10.5	8.0
9	2	18.2	11.9	6.9
10	2	5.5	9.9	9.1

Use PROC UNIVARIATE to produce histograms, normal probability plots, and box plots, and test the distributions for normality. Do this for the variables REACT, LIVER_WT, and SPLEEN, first for all subjects and then separately for each of the two DOSES.

2-5. What's wrong with the following program?

```
DATA 123;
   INPUT AGE STATUS PROGNOSIS DOCTOR GENDER STATUS2
   STATUS3;
(data lines)
;
PROC CHART DATA=123 BY GENDER;
   VBAR STATUS
   VBAR PROGNOSIS;
RUN;

PROC PLOT DATA=123;
   DOCTOR BY PROGNOSIS;
RUN;
```

2-6. Given the data set:

Salesperson	Target company	Number of visits	Number of phone calls	Units sold
Brown	American	3	12	28,000
Johnson	VRW	6	14	33,000
Rivera	Texam	2	6	8,000
Brown	Standard	0	22	0
Brown	Knowles	2	19	12,000
Rivera	Metro	4	8	13,000
Rivera	Uniman	8	7	27,000
Johnson	Oldham	3	16	8,000
Johnson	Rondo	2	14	2,000

(a) Write a SAS program to compare the sales records of the company's three sales people. (Compute the sum and mean for the number of visits, phone calls, and units sold for each salesman.) Note that the values for "Units sold" contain commas. Since you don't know how to read data values with commas (you use a COMMA informat), omit the commas when you enter your data values.

(b) Plot the number of visits against the number of phone calls. Use the value of Salesperson (the first character in the name) as the plotting symbol (instead of the usual A, B, C, etc.).

(c) Make a frequency bar chart for each Salesperson for the sum of "units sold."

***2-7.** You have completed an experiment and recorded a subject ID, and values for variables A, B, and C. You want to compute means for A, B, and C but, unfortunately, your lab technician, who didn't know SAS programming, arranged the data like this:

ID	TYPE	SCORE
1	A	44
1	B	9
1	C	203
2	A	50
2	B	7
2	C	188
3	A	39
3	B	9
3	C	234

Write a program to read these data and produce means. (HINT: A CLASS or BY statement might come in handy.)

CHAPTER

3

Analyzing Categorical Data

A. *Introduction*

This chapter explains ways of analyzing categorical data as well as step-by-step instructions for analyzing questionnaires. Variables such as gender, sick or well, success or failure, and age group represent categories rather than numerical values. We use a number of statistical techniques such as tests of proportions and chi-square with variables of this type.

You may notice a substantial enlargement of this topic from our previous edition. This is the result of expanded techniques from the SAS procedures used on categorical data and some techniques such as meta analysis that have become increasingly popular in many fields.

B. *Questionnaire Design and Analysis*

A common way to collect certain types of data is by using a questionnaire. Although these can be designed in many ways, the accompanying Sample Questionnaire contains features that make it especially useful when the collected data are to be entered into a computer. We present an approach based on data being manually entered into the computer, which is probably the most common approach for most

For office use only

ID

SAMPLE QUESTIONNAIRE

1. Age in years ____

(Questions 2–4: Please check the appropriate category.)

2. Gender ____ 1=Male
____ 2=Female

3. Race ____ 1=White
____ 2=African American
____ 3=Hispanic
____ 4=Other

4. Marital Status:
____ 1=Single
____ 2=Married
____ 3=Widowed
____ 4=Divorced

5. Education Level:
____ 1=High school or less
____ 2=Two year college
____ 3=Four year college (B.A. or B.S)
____ 4=Post graduate degree

For each of the following statements, please enter a NUMBER from the list below, on the line to the LEFT of each question. Use the following codes:

1=Strongly disagree 2=Disagree 3=Neutral
4=Agree 5=Strongly agree

____ **6.** The president of the U.S. has been doing a good job.

____ **7.** The arms budget should be increased.

____ **8.** There should be more federal aid to big cities.

researchers. However, software is available that allows researchers to design questionnaires which are scannable on standard PC scanners. Researchers doing a lot of work with questionnaires should investigate that possibility.

Notice that every response in this questionnaire is placed in a box by a coding clerk, and that the boxes are all on the right side of the page. This will facilitate the job of transferring the data from the survey instrument to our computer. One should be careful, however, not to ignore the person who is filling out the questionnaire. If the questionnaire confuses the respondent, it will not matter how easy the data are to enter. With this in mind, many experienced questionnaire designers would place the choices for questions 6 through 8 below each of these items and have the respondent check his choice.

The typical method of coding data from a questionnaire of this type is to enter the data into a computer by using a wordprocessor, a specialized data entry program, a spread sheet (such as Lotus (TM) or Excel (TM)), or by using a data-base management program (such as Paradox (TM), Access (TM), or DBase (TM)). In the case of the wordprocessor or data entry program, we would probably set aside certain columns for each variable. Where a data base management system is used, the data can either be written to a text file or converted directly to a SAS system file if the appropriate software is available. There are also key-to-tape systems for large commercial applications. Another option is the use of an optical mark sense reader for large volume data entry requirements. It is preferable to design the questionnaire so that the data can be entered directly from the questionnaire, rather than having to be transcribed first to a coding form.

We might decide to enter our questionnaire data as follows:

Column	Description	Variable Name
1-3	Subject ID	ID
4-5	Age in years	AGE
6	Gender	GENDER
7	Race	RACE
8	Marital status	MARITAL
9	Education level	EDUC
10	President doing good job	PRES
11	Arms budget increased	ARMS
12	Federal aid to cities	CITIES

Typical lines of data would look like:

```
001091111232
002452222422
```

Notice that we have not left any spaces between the values for each variable. This is a common method of data entry since it saves space and extra typing. Therefore, we must specify the column location for each variable. Our INPUT statement for this questionnaire would be written:

```
INPUT ID 1-3 AGE 4-5 GENDER $ 6 RACE $ 7 MARITAL $ 8 EDUC $ 9
      PRES 10 ARMS 11 CITIES 12;
```


Some programmers find it easier to read INPUT statements if written one variable per line, thus:

```
INPUT ID        1-3
      AGE       4-5
      GENDER  $ 6
      RACE    $ 7
      MARITAL $ 8
      EDUC    $ 9
      PRES      10
      ARMS      11
      CITIES    12;
```

Don't forget the semicolon at the end of the INPUT statement if you use this approach.

Each variable name is followed by its column designation. We chose to store data values for GENDER, RACE, MARITAL, and EDUC as characters, even though we are coding the values as numbers. One reason for doing this is to save storage space on the computer disk. Another reason is that we would be unlikely to perform any type of arithmetic operation on a variable such as RACE. Storing these values as characters helps remind us that the numerical values are merely the names of categories.

A common occurrence with questionnaires is that some respondents do not answer all the questions. With a list INPUT statement (one in which we list only the variable names and not the column designations) we use a period to represent a missing value; with our column INPUT statement we leave the column(s) blank. We can do this since it is the columns, not the order of the data, that determine which variable is being read.

A complete SAS program shown below, with some sample lines of data, calculates the mean age of the respondents and compute frequencies for all the other variables:

```
DATA QUEST;
   INPUT ID 1-3 AGE 4-5 GENDER $ 6 RACE $ 7 MARITAL $ 8 EDUC $ 9
         PRES 10 ARMS 11 CITIES 12;
DATALINES;
001091111232
002452222422
003351324442
004271111121
005682132333
006651243425
;
PROC MEANS DATA=QUEST MAXDEC=2 N MEAN STD;
   TITLE 'Questionnaire Analysis';
   VAR AGE;
RUN;
```

[Continued]

```
PROC FREQ DATA=QUEST;
   TITLE 'Frequency Counts for Categorical Variables';
   TABLES GENDER RACE MARITAL EDUC PRES ARMS CITIES;
RUN;
```

We have chosen to supply PROC MEANS with options to print statistics to two decimal places and to compute N, the number of nonmissing observations, the mean, and standard deviation.

We listed the variables GENDER, RACE, MARITAL, EDUC, PRES, ARMS, and CITIES in our PROC FREQ TABLES statement. A shortcut method for specifying a list of variables in a SAS data or PROC step is the -- (two dashes together) notation. The convention

```
variable_name_1 -- variable_name_2
```

means to include all the variables from variable_name_1 to variable_name_2 in the order they exist in the SAS data set. A TABLES statement equivalent to the one above, using this shortcut method, is:

```
TABLES GENDER -- CITIES;
```

The statement "in the order they exist in the SAS data set" is particularly important when using this particular short-cut method of specifying variable lists. In this case, the order is the same as the order on the INPUT statement. As you will see later, such SAS statements as LENGTH, ARRAY, and RETAIN can affect this order.

While we are on the topic of variable list notation, ROOTn-ROOTm is used to refer to all variables with the same alphabetic root, from the nth to the mth numeric ending. For example, ABC1-ABC5 is equivalent to: ABC1 ABC2 ABC3 ABC4 ABC5. It is convenient to name certain variables using the same root with a number ending so we can use the single dash notation any time we want to refer to part or all of the list. If we recorded the response to 50 multiple-choice questions in a test, convenient variable names would be QUES1, QUES2, . . . up to QUES50. Then, if we wanted frequencies on all 50 variables, the tables request:

```
TABLES QUES1-QUES50;
```

would do the trick. It is not necessary for the variables to be in any particular order in the data set when this notation is used.

In this questionnaire, we have not requested any statistics for the variable ID. The ID number only serves as an identifier if we want to go back to the original questionnaire to check data values. This is a highly recommended procedure. Without an ID variable of some sort, when we discover an error in the data, it is difficult to find the original questionnaire to check on the correct value. Even if you did not include an ID on the questionnaire, number the questionnaires as they are returned, and enter this number in the computer along with the other responses.

A sample of the output from PROC FREQ is shown below:

```
Frequency Counts for Categorical Variables

                                     Cumulative  Cumulative
GENDER   Frequency    Percent        Frequency     Percent
-----------------------------------------------------------
1               4       66.7                4        66.7
2               2       33.3                6       100.0

                                     Cumulative  Cumulative
RACE     Frequency    Percent        Frequency     Percent
-----------------------------------------------------------
1               3       50.0                3        50.0
2               2       33.3                5        83.3
3               1       16.7                6       100.0

                                     Cumulative  Cumulative
MARITAL  Frequency    Percent        Frequency     Percent
-----------------------------------------------------------
1               2       33.3                2        33.3
2               2       33.3                4        66.7
3               1       16.7                5        83.3
4               1       16.7                6       100.0
```

C. Adding Variable Labels

We can improve considerably on this output. First, we have to refer back to our coding scheme to see the definition of each of the variable names. Some variable names like GENDER and RACE need no explanation; others like PRES and CITIES do. We can associate a variable label with each variable name by using a LABEL statement. These labels will be printed along with the variable name in certain procedures such as PROC FREQ and PROC MEANS. The general form of a LABEL statement is

```
LABEL variable_name = 'description'
         .
         .
         .
      variable_name = 'description';
```

The "description" can contain up to 40 characters (each blank counts as a character) and must be enclosed in single or double quotes (but not a mixture of both, e.g. "Description'). The LABEL statement can be placed anywhere in the DATA step. Our program, rewritten to include variable labels, follows:

```
DATA QUEST;
   INPUT ID 1-3 AGE 4-5 GENDER $ 6 RACE $ 7 MARITAL $ 8 EDUC $ 9
         PRES 10 ARMS 11 CITIES 12;

   LABEL  MARITAL='Marital Status'
          EDUC='Education Level'
          PRES='President Doing a Good Job'
          ARMS='Arms Budget Increase'
          CITIES='Federal Aid to Cities';

DATALINES;
001091111232
002452222422
003351324442
004271111121
005682132333
006651243425
;
PROC MEANS DATA=QUEST MAXDEC=2 N MEAN STD;
   TITLE 'Questionnaire Analysis';
   VAR AGE;
RUN;

PROC FREQ DATA=QUEST;
   TITLE 'Frequency Counts for Categorical Variables';
   TABLES GENDER RACE MARITAL EDUC PRES ARMS CITIES;
RUN;
```

Notice that we did not supply variable labels for all our variables. The ones you choose are up to you. Now, when we run our program, the labels will be printed along with the variable names in our PROC FREQ output. A sample of the output from this program is shown below:

Frequency Counts for Categorical Variables

GENDER	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	4	66.7	4	66.7
2	2	33.3	6	100.0

RACE	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	3	50.0	3	50.0
2	2	33.3	5	83.3
3	1	16.7	6	100.0

[Continued]

Marital Status

MARITAL	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	2	33.3	2	33.3
2	2	33.3	4	66.7
3	1	16.7	5	83.3
4	1	16.7	6	100.0

Education Level

EDUC	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	1	16.7	1	16.7
2	2	33.3	3	50.0
3	2	33.3	5	83.3
4	1	16.7	6	100.0

President Doing a Good Job

PRES	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	1	16.7	1	16.7
2	1	16.7	2	33.3
3	1	16.7	3	50.0
4	3	50.0	6	100.0

Arms Budget Increase

ARMS	Frequency	Percent	Cumulative Frequency	Cumulative Percent
2	3	50.0	3	50.0
3	2	33.3	5	83.3
4	1	16.7	6	100.0

Federal Aid to Cities

CITIES	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	1	16.7	1	16.7
2	3	50.0	4	66.7
3	1	16.7	5	83.3
5	1	16.7	6	100.0

D. Adding "Value Labels" (Formats)

We would like to improve the readability of the output one step further. The remaining problem is that the values for our variables (1=male 2=female etc.) are printed on the output, not the names that we have assigned to these values. We would like the output to show the number of males and females, for example, not the number of 1's and 2's for the variable GENDER. We can supply the "value labels" in two steps.

The first step is to define our code values for each variable. For example, 1=Male, 2=Female will be used for our variable GENDER. The codes 1=str disagree, 2=disagree, etc. will be used for three variables: PRES, ARMS, and CITIES. SAS software calls these codes formats, and we define the formats in a procedure by that name.

The second step, shown later, will be to associate a FORMAT with one or more variable names. Below is an example of PROC FORMAT used for our questionnaire:

```
PROC FORMAT;
   VALUE $SEXFMT    '1'='Male' '2'='Female';
   VALUE $RACE      '1'='White'  '2'='African Am.' '3'='Hispanic'
                    '4'='Other';
   VALUE $OSCAR     '1'='Single' '2'='Married' '3'='Widowed'
                    '4'='Divorced';
   VALUE $EDUC      '1'='High Sch or Less'
                    '2'='Two Yr. College'
                    '3'='Four Yr. College'
                    '4'='Graduate Degree';
   VALUE LIKERT      1='Str Disagree'
                     2='Disagree'
                     3='Neutral'
                     4='Agree'
                     5='Str Agree';
RUN;
```

The names $SEXFMT, $RACE, $OSCAR, $EDUC, and LIKERT are format names. (An aside: We chose the name LIKERT since scales such as 1=strongly disagree, 2=disagree, 3=neutral, etc. are called Likert scales by psychometricians.) You may choose any name (consistent with naming conventions, with the exception that they cannot end in a number or be identical to a SAS supplied format) for your formats. It is best to give names that help you remember what the format will be used for. As a matter of fact, you can use the same name for a format and for a variable without confusion. Notice the silly format name $OSCAR used to format values of the variable called MARITAL. We did this just to emphasize the fact that format names do not have to be related to the variables they will be used to format. Format values can be up to 40 characters long but should be restricted to 16 characters since this is the number of characters that will appear in a cross tabulation or frequency table.

Format names for character variables must start with a $ sign. Notice the format names $SEXFMT, $RACE, $OSCAR, and $EDUC. These formats will all be used to define values for character variables. Notice that the value to the left of the = sign must also be enclosed in single quotes for character variables. If your format value contains any single quotes, enclose it with double quotes.

Once we have defined a set of formats (such as $SEXFMT, $RACE, LIKERT, etc.), we need to assign the formats to the appropriate variables. Just because we have named a format $EDUC, for example, does not mean that this format has be used with the variable we have called EDUC. We need another SAS statement that indicates which formats will be used or associated with which variables. Format statements start with the word FORMAT, followed by a single variable or a list of variables, followed by a format name to be used with the preceding variables. The list of variable and format names continues on as many lines as necessary and ends with a semicolon. SAS software knows the difference between our variable names and our format names since we place a period after each format name in our format statement.

Format statements can be placed within a DATA step or as a statement in a PROC step. If we choose to place our format statement in the DATA step, the formatted values will be associated with the assigned variable(s) for all PROCS that follow. If we place a format statement in a PROC step, the formatted values will be used only for that procedure. In this example we will place our format statement in the DATA step. Therefore, we will define our formats with PROC FORMAT before we write our DATA step. We will associate the format $SEXFMT with the variable GENDER, $RACE with the variable RACE, and so forth. The variables PRES, ARMS, and CITIES are all "sharing" the LIKERT format. If you look at the completed questionnaire program below, the use of PROC FORMAT and its application in other PROCs should become clear.

```
PROC FORMAT;
   VALUE $SEXFMT    '1'='Male' '2'='Female';
   VALUE $RACE      '1'='White'  '2'='African Am.' '3'='Hispanic'
                    '4'='Other';
   VALUE $OSCAR     '1'='Single' '2'='Married' '3'='Widowed'
                    '4'='Divorced';
   VALUE $EDUC      '1'='High Sch or Less'
                    '2'='Two Yr. College'
                    '3'='Four Yr. College'
                    '4'='Graduate Degree';
   VALUE LIKERT      1='Str Disagree'
                     2='Disagree'
                     3='Neutral'
                     4='Agree'
                     5='Str Agree';
RUN;

DATA QUEST;
   INPUT ID 1-3 AGE 4-5 GENDER $ 6 RACE $ 7 MARITAL $ 8 EDUC $ 9
         PRES 10 ARMS 11 CITIES 12;
```

[Continued]

```
   LABEL MARITAL='Marital Status'
         EDUC='Education Level'
         PRES='President Doing a Good Job'
         ARMS='Arms Budget Increase'
         CITIES='Federal Aid to Cities';

   FORMAT GENDER $SEXFMT. RACE $RACE. MARITAL $OSCAR.
          EDUC $EDUC. PRES ARMS CITIES LIKERT.;

DATALINES;
001091111232
002452222422
003351324442
004271111121
005682132333
006651243425
;
PROC MEANS DATA=QUEST MAXDEC=2 N MEAN STD;
   TITLE 'Questionnaire Analysis';
   VAR AGE;
RUN;

PROC FREQ DATA=QUEST;
   TITLE 'Frequency Counts for Categorical Variables';
   TABLES GENDER RACE MARITAL EDUC PRES ARMS CITIES;
RUN;
```

Output from this program is shown below:

```
Questionnaire Analysis

Analysis Variable : AGE

N        Mean       Std Dev
-----------------------------
6       41.50         22.70
-----------------------------

Frequency Counts for Categorical Variables

                                Cumulative  Cumulative
GENDER  Frequency   Percent     Frequency    Percent
-----------------------------------------------------
Male           4      66.7            4        66.7
Female         2      33.3            6       100.0
```


[Continued]

```
                                Cumulative  Cumulative
RACE         Frequency  Percent  Frequency    Percent
-----------------------------------------------------
White               3     50.0          3       50.0
African Am.         2     33.3          5       83.3
Hispanic            1     16.7          6      100.0

                    Marital Status

                                Cumulative  Cumulative
MARITAL      Frequency  Percent  Frequency    Percent
-----------------------------------------------------
Single              2     33.3          2       33.3
Married             2     33.3          4       66.7
Widowed             1     16.7          5       83.3
Divorced            1     16.7          6      100.0

                    Education Level

                                       Cumulative  Cumulative
EDUC                Frequency  Percent  Frequency    Percent
------------------------------------------------------------
High Sch or Less           1     16.7          1       16.7
Two Yr. College            2     33.3          3       50.0
Four Yr. College           2     33.3          5       83.3
Graduate Degree            1     16.7          6      100.0

               President Doing a Good Job

                                  Cumulative  Cumulative
        PRES   Frequency  Percent  Frequency    Percent
-------------------------------------------------------
Str Disagree          1     16.7          1       16.7
Disagree              1     16.7          2       33.3
Neutral               1     16.7          3       50.0
Agree                 3     50.0          6      100.0

                  Arms Budget Increase

                                  Cumulative  Cumulative
        ARMS   Frequency  Percent  Frequency    Percent
-------------------------------------------------------
Disagree              3     50.0          3       50.0
Neutral               2     33.3          5       83.3
Agree                 1     16.7          6      100.0
```

[Continued]

```
                    Federal Aid to Cities

                                        Cumulative  Cumulative
      CITIES   Frequency   Percent    Frequency     Percent
-----------------------------------------------------------
Str Disagree          1      16.7             1       16.7
Disagree              3      50.0             4       66.7
Neutral               1      16.7             5       83.3
Str Agree             1      16.7             6      100.0
```

Notice how much easier it is to read this listing compared to the earlier version which did not contain labels or formats.

E. Recoding Data

In the previous questionnaire example, we coded the respondents' actual age in years. What if we want to look at the relationship between age and the questions 6-8 (opinion questions)? It might be convenient to have a variable that indicated an age group rather than the person's actual age. We will look at two ways of accomplishing this.

Look at the following SAS statements:

```
PROC FORMAT;
   VALUE $SEXFMT '1'='Male' '2'='Female';
         .
         .
         .
   VALUE AGEFMT 1='0-20'  2='21-40'  3='41-60'
                4='Greater than 60';
RUN;

DATA QUEST;
   INPUT ID 1-3 AGE 4-5 GENDER $ 6 RACE $ 7 MARITAL $ 8 EDUC $ 9
         PRES 10 ARMS 11 CITIES 12;
   IF 0 <= AGE <= 20 THEN AGEGRP=1;
   IF 20 < AGE <= 40 THEN AGEGRP=2;
   IF 40 < AGE <= 60 THEN AGEGRP=3;
   IF AGE > 60 THEN AGEGRP=4;

   LABEL  MARITAL='MARITAL STATUS'
         .
         .
         .
         AGEGRP='AGE GROUP';

   FORMAT GENDER $SEXFMT. RACE $RACE. MARITAL $OSCAR.
```


[Continued]

```
          EDUC $EDUC. PRES ARMS CITIES LIKERT.
          AGEGRP AGEFMT.;
DATALINES;
(data lines)
;
PROC FREQ DATA=QUEST;
  TABLES GENDER -- AGEGRP;
RUN;
```

Several new features have been added to the program. The major additions are the four IF statements following the INPUT. With the DATA statement, the program begins to create a data set. When the INPUT step is reached, the program will read a line of data according to the INPUT specifications. Next, each IF statement is evaluated. If the condition is true, then the variable AGEGRP will be set to 1, 2, 3, or 4. Finally, when the DATALINES statement is reached, an observation is added to the SAS data set. The variables in the data set will include all the variables listed in the INPUT statement as well as the variable AGEGRP. The variable AGEGRP may be used in any PROC just like any of the other variables. Be sure there are no "cracks" in your recoding ranges. That is, make sure you code your IF statements so that there isn't a value of AGE that is not recoded. If that happens, the variable AGEGRP for that person will have a missing value. Notice that the first IF statement is written:

```
IF 0 <= AGE <= 20 THEN AGEGRP=1;
```

and not like this:

```
IF AGE <= 20 then AGEGRP=1;
```

The reason is subtle. SAS stores missing values as −0 raised to a power. You don't need to understand what this means but you do need to know that SAS programs treat missing values as negative infinity for ordering purposes. That is, a missing value will be considered lower than any numeric value. Thus, the above IF statement will be true for missing values as well as the valid ages from 0 to 20. Another way of writing the statement:

```
IF 0 <= AGE <= 20 THEN AGEGRP=1;
```

is:

```
IF AGE <= 20 AND AGE NE . THEN AGEGRP=1;
```

or

```
IF AGE >= 0 AND AGE <= 20 THEN AGEGRP=1;
```

Feel free to choose whichever method makes most sense to you. The other IF statments can also be written using an alternative syntax. For example, you could write the IF statement for AGEGRP = 2 as:

```
IF AGE > 20 AND AGE <= 40 THEN AGEGRP=2;
```

A complete list of the SAS comparison operators is shown below: Either the two-letter abbreviations or the symbols may be used.

Definition	Symbol(s)
Equal to	= or EQ
Greater than	> or GT
Less than	< or LT
Greater than or equal to	>= or GE
Less than or equal to	<= or LE
Not equal to	^= or NE

(NOTE: ~= is an alternative not equal symbol)

A better way to write multiple IF statements is to use an ELSE before all but the first IF. The four IF-THEN/ELSE statements would then look like this:

```
IF 0 <= AGE <= 20 THEN AGEGRP=1;
   ELSE IF 20 < AGE <= 40 THEN AGEGRP=2;
   ELSE IF 40 < AGE <= 60 THEN AGEGRP=3;
   ELSE IF AGE > 60 THEN AGEGRP=4;
```

The effect of the ELSE statements is that when any IF statement is true, all the following ELSE statements are skipped. The advantage is to reduce computer time (since all the IF do not have to be tested) and to avoid the following type of problem. Can you see what will happen with the statements below? (Assume X can have values of 1,2,3,4, or 5.)

```
IF X=1 THEN X=5;
IF X=2 THEN X=4;
IF X=4 THEN X=2;
IF X=5 THEN X=1;
```

What happens when X is 1? The first IF statement is true. This causes X to have a value of 5. The next two IF statements are false, but the last IF statement is true. X is back to 1! The ELSE statements, besides reducing computer time, prevent the problem above. The correct coding is:

```
IF X=1 THEN X=5;
   ELSE IF X=2 THEN X=4;
   ELSE IF X=4 THEN X=2;
   ELSE IF X=5 THEN X=1;
```

One final note: If all we wanted to do was recode X so that 1 = 5, 2 = 4, 3 = 3, 4 = 2, and 5 = 1, the statement:


```
X = 6 - X;
```

would be the best way to recode the X values. Check it out.

Notice that we added a line to the LABEL section and to PROC FORMAT to supply a variable label and a format for our new variable.

F. *Using a Format to Recode a Variable*

There is another way of recoding our AGE variable without creating a new variable. We use a "trick." By defining a format and associating it with a variable, we can have SAS software assign subjects to age categories. The formats can be associated with a variable in the DATA step or directly in a PROC. If you include a format statement in the DATA step, the format will remain associated with the variable for all following procedures. If you include a format statement within a procedure, that association remains only for that particular procedure. Also, remember that the actual "internal" value of the original variable has not changed. So, if you place a format statement within a DATA step and associate a format with a variable, all computations regarding that variable still involve the original value. You can still compute means (with PROC MEANS), for example, on AGE even though it has an associated format. It is only in such procedures as PROC FREQ or when used as a CLASS variable that the associated format has an effect. So, to continue with our example, we write:

```
PROC FORMAT;
   VALUE AGROUP LOW-20   = '0-20'
                21-40    = '21-40'
                41-60    = '41-60'
                61-HIGH  = 'Greater than 60';
```

As you can see, instead of single values to the left of the equals sign, we are supplying a range of values. SAS software will not let us specify overlapping ranges when defining formats. Thus, LOW- 20='Very Young' and 15-30='Young' are not allowed. The special words HIGH and LOW are available to indicate the highest and lowest values in the data set (not counting missing values) respectively. Thus, the term LOW-20 refers to all values from the lowest value up to and including 20. Remember, the keyword LOW does not include missing values.

One additional keyword, OTHER, can be used to match any data value not included in any of the other format ranges. Thus, you can use the form:

```
VALUE AGROUP LOW-20   = '0-20'
             21-40    = '21-40'
             41-60    = '41-60'
             60-HIGH  = 'Greater than 60'
             .        = 'Did Not Anewer'
             OTHER    = 'Out of Range';
```

As you would expect, the . = 'format label' allows you to supply a label to missing values.

Once we have defined a format, we can then associate it with a variable, either in the DATA step or in the procedure itself. For example, we can write a TABLES statement on the original variable (such as AGE). By supplying the format information using the format AGROUP for the variable AGE, frequencies will be computed for the new categories instead of the original AGE values. In this example, we place the format statement in the appropriate PROC rather than in the DATA statement since we want to use the recoded values only for PROC FREQ.

Thus, the SAS statements:

```
PROC FREQ DATA=QUEST;
   TABLES AGE;
   FORMAT AGE AGROUP.;
RUN;
```

produce the following output:

```
                                       Cumulative  Cumulative
          AGE   Frequency   Percent    Frequency    Percent
-------------------------------------------------------------
0-20                  1      16.7           1        16.7
21-40                 2      33.3           3        50.0
41-60                 1      16.7           4        66.7
Greater than 60       2      33.3           6       100.0
```

While on the topic of creating formats, we mention a few other ways to represent format values. You can combine specific codes and code ranges in a single format statement. Suppose we assigned the codes 1-3 to the colors 'Red', 'White', and 'Blue', and codes 0, and 4 through 9 to other colors. The codes 0, and 4 through 9 are to be lumped together and called 'Other Colors', and any other codes are to be labeled as 'Out of Range'. The VALUE statement to create a SAS format called COLOR, which meets these specifications, is shown below:

```
VALUE COLOR 1      = 'Red'
            2      = 'White'
            3      = 'Blue'
            0,4-9  = 'Other Colors'
            OTHER  = 'Out of Range';
```

A series of values or ranges of values can be separated by commas. In this example, either a 0 or any number from 4 through 9 will be formatted as 'Other Colors'. A

good application of this would be to regroup a large number of categories into a smaller number of categories. Suppose we had coded our original questionnaire with five levels of RACE:

```
1=WHITE  2=SPANISH AMERICAN  3=ORIENTAL  4=AFRICAN AM.  5=OTHER
```

Now, for a particular analysis, we want to have only three groups, WHITE, AFRICAN AM., and OTHER. A quick and easy way to do this is to supply a format like the following:

```
VALUE $RACE '1'          = 'WHITE'
            '4'          = 'AFRICAN AM.'
            '2','3','5'  = 'OTHER RACES';
```

The advantage of this method of grouping values is that you don't have to create a new data set or make new variables. All that is necessary is to write the PROC FORMAT statements to create a new format and add a format statement to the procedure where you want to use the new grouping.

G. Two-way Frequency Tables

Besides computing frequencies on individual variables, we may have occasion to count occurrences of one variable at each level of another variable. An example will make this clear. Suppose we took a poll of presidential preference and also recorded the gender of the respondent. Sample data might look like this:

GENDER	CANDIDATE
M	DEWEY
F	TRUMAN
M	TRUMAN
M	DEWEY
F	TRUMAN
etc.	

We would like to know (1) how many people were for Dewey and how many for Truman, (2) how many males and females were in the sample, and (3) how many males and females were for Dewey and Truman, respectively.

A previous example of PROC FREQ shows how to perform tasks (1) and (2). For (3) we would like a table that looks like this:

	GENDER F	GENDER M	
DEWEY	70	40	110
TRUMAN	30	40	70
	100	80	180

If this were our table, it would show that females favored Dewey over Truman 70 to 30, while males were split evenly. A SAS program to solve all three tasks is:

```
DATA ELECT;
   INPUT GENDER $ CANDID $ ;
DATALINES;
M DEWEY
F TRUMAN
M TRUMAN
M DEWEY
F TRUMAN
(more data lines)
;
PROC FREQ DATA=ELECT;
   TABLES GENDER CANDID
          CANDID*GENDER;
RUN;
```

Notice that since the variables GENDER and CANDID are coded as character values (alphanumeric), we follow each variable name with a $ in the INPUT statement. Another fact that we have not mentioned thus far is that the VALUES of our character variables also cannot be longer than eight letters in length when using the "list" form of the INPUT statement, unless we modify our INPUT statement to indicate this change. So, for the time being, we cannot use this program for the Eisenhower/Stevenson election (without using nicknames).

The first two TABLES are one-way frequency tables, the same type we have seen before; the TABLE specification, CANDID*GENDER is a request for a two-way table.

What would a table like the one above tell us? If it were based on a random sample of voters, we might conclude that gender affected voting patterns. Before we conclude that this is true of the nation as a whole, it would be nice to see how likely it was that these results were simply due to a quirky sample. A statistic called chi-square will do just this.

Consider the table again. There were 180 people in our sample, 110 for Dewey and 70 for Truman; 100 females and 80 males. If there were no gender bias, we would expect the proportion of the population who wanted Dewey (110/180) to be the same for the females and males. Therefore, since there were 100 females, we could expect (110/180) of 100 (approximately 61) females to be for Dewey. Our expectations (in statistics called expected values) for all the other cells can be calculated in the same manner. Once we have observed and expected frequencies for each cell (each combination of gender and candidate), the chi-square statistic can be computed. By adding an option for chi-square on our TABLES request, we can have our program compute chi-square and the probability of obtaining a value as large or larger by chance alone. Remembering that statement options follow a slash, the request for chi-square is written as:

```
TABLES CANDID*GENDER / CHISQ;
```

Output from the request above is shown next:


```
TABLE OF CANDID BY GENDER

CANDID     GENDER

Frequency|
Percent  |
Row Pct  |
Col Pct  |F       |M       |  Total
---------+--------+--------+
DEWEY    |     70 |     40 |    110
         |  38.89 |  22.22 |  61.11
         |  63.64 |  36.36 |
         |  70.00 |  50.00 |
---------+--------+--------+
TRUMAN   |     30 |     40 |     70
         |  16.67 |  22.22 |  38.89
         |  42.86 |  57.14 |
         |  30.00 |  50.00 |
---------+--------+--------+
Total         100       80      180
            55.56    44.44   100.00

STATISTICS FOR TABLE OF CANDID BY GENDER

Statistic                     DF     Value       Prob
------------------------------------------------------
Chi-Square                     1     7.481      0.006
Likelihood Ratio Chi-Square    1     7.493      0.006
Continuity Adj. Chi-Square     1     6.663      0.010
Mantel-Haenszel Chi-Square     1     7.439      0.006
Fisher's Exact Test (Left)                      0.998
                    (Right)                  4.91E-03
                    (2-Tail)                 8.73E-03
Phi Coefficient                      0.204
Contingency Coefficient              0.200
Cramer's V                           0.204

Sample Size = 180
```

The key to the table is found in its upper left-hand corner. It tells you what all the numbers in the cells are. By FREQUENCY, we mean the number of subjects in the cell. For example, 70 females favored Dewey for president. The second number in each cell shows the PERCENT of the total population. The third number, labeled ROW PCT, gives the percent of each row. For example, of all the people for Dewey (row 1), $70/110 \times 100$, or 63.64%, were female. The last number, COL PCT, is the column percent. Of all the females, 70% were for Dewey and 30% were for Truman. In the TABLES request for a two-way cross tabulation, the variable that forms the columns is placed second (e.g., CANDID*GENDER). In our statistical requests, rows come first, then columns.

For our example, chi-square equals 7.48, and the probability of obtaining a chi-square this large or larger by chance alone is .006. Therefore, we can say that, based on our data, there appears to be a gender bias in presidential preference: There is a tendency for females to show greater preference for Dewey than males do or, put another way, for males to prefer Truman.

The number of degrees of freedom (df) in a chi-square statistic is equal to the number of rows minus one multiplied by the number of columns minus one ($(R - 1) \times (C - 1)$). Thus, our 2×2 chi-square has 1 df. Whenever a chi-square table has 1 df and the expected value of any cell is less than 5, a "correction for continuity," called Yates' correction, is often applied. SAS software prints out a corrected chi-square value and its associated probability beside the heading Continuity Adj. Chi-square. Another alternative, when you have small expected values, is to use Fisher's exact test, which is included in the list of statistics for the table. Remembering that the chi-square test is nondirectional, you will probably want to use the two-tailed Fisher probability. When the degrees of freedom are greater than 1, it is desirable that no more than 20% of the cells have expected values less than 5. The program will print a warning when this condition occurs. This does not mean that you have to throw your data out if you fall into this situation. This is a situation that is complex and about which statisticians don't always agree. Below, we will suggest one alternative if your df are greater than 1. If you are in doubt, consult your friendly statistician.

For larger tables (more than four cells) the usual alternative when faced with small expected cell values is to combine, or collapse, cells. If we had four categories of age: 0-20, 21-40, 41-60, and over 60, we might combine 0-20 and 21-40 as one group, and 41-60 and 60+ as another. Another example would be combining categories such as "strongly disagree" and "disagree" on an opinion questionnaire. We can use either method of recoding shown in the previous section to accomplish this.

H. A Short-cut Way to Request Multiple Tables

We can use the questionnaire program at the beginning of this chapter to see another example of a two-way table. Suppose we wanted crosstabulations of AGEGRP against the three variables PRES, ARMS, and CITIES. We could code:

```
TABLES (PRES ARMS CITIES)*AGEGRP;
```

This will generate three tables and is a short way of writing:

```
TABLES PRES*AGEGRP ARMS*AGEGRP CITIES*AGEGRP;
```

We can also have multiple-column variables in a TABLE request. Thus:

```
TABLES (PRES ARMS) * (AGEGRP Gender);
```

would produce four tables, PRES*AGEGRP, PRES*GENDER, ARMS*AGEGRP, and ARMS*GENDER.

When you use this method, be sure to enclose the list of variables within parentheses.

One of the tables generated from this program is shown next:


```
TABLE OF PRES BY AGEGRP

PRES(President Doing a Good Job)     AGEGRP

Frequency    |
Percent      |
Row Pct      |
Col Pct      |       1|       2|       3|       4|  Total
-------------+--------+--------+--------+--------+
Str Disagree |      0 |      1 |      0 |      0 |      1
             |   0.00 |  16.67 |   0.00 |   0.00 |  16.67
             |   0.00 | 100.00 |   0.00 |   0.00 |
             |   0.00 |  50.00 |   0.00 |   0.00 |
-------------+--------+--------+--------+--------+
Disagree     |      1 |      0 |      0 |      0 |      1
             |  16.67 |   0.00 |   0.00 |   0.00 |  16.67
             | 100.00 |   0.00 |   0.00 |   0.00 |
             | 100.00 |   0.00 |   0.00 |   0.00 |
-------------+--------+--------+--------+--------+
Neutral      |      0 |      0 |      0 |      1 |      1
             |   0.00 |   0.00 |   0.00 |  16.67 |  16.67
             |   0.00 |   0.00 |   0.00 | 100.00 |
             |   0.00 |   0.00 |   0.00 |  50.00 |
-------------+--------+--------+--------+--------+
Agree        |      0 |      1 |      1 |      1 |      3
             |   0.00 |  16.67 |  16.67 |  16.67 |  50.00
             |   0.00 |  33.33 |  33.33 |  33.33 |
             |   0.00 |  50.00 | 100.00 |  50.00 |
-------------+--------+--------+--------+--------+
Total               1        2        1        2        6
                16.67    33.33    16.67    33.33   100.00
```

I. Computing Chi-square from Frequency Counts

When you already have a contingency table and want to use SAS software to compute a chi-square statistic, there is a WEIGHT statement that makes this task possible. Suppose someone gave you the 2×2 table below and wanted to compute chi-square:

		OUTCOME	
		Dead	Alive
GROUP	Control	20	80
	Drug	10	90

We could code this by reading in values for GROUP, OUTCOME, and the number of subjects, COUNT, in the appropriate cell. Thus, we would have:

```
DATA CHISQ;
   INPUT GROUP $ OUTCOME $ COUNT;
DATALINES;
CONTROL DEAD 20
CONTROL ALIVE 80
DRUG DEAD 10
DRUG ALIVE 90
;
PROC FREQ DATA=CHISQ;
   TABLES GROUP*OUTCOME / CHISQ;
   WEIGHT COUNT;
RUN;
```

The WEIGHT statement tells the procedure how many subjects there are for each combination of OUTCOME and GROUP.

J. A Useful Program for Multiple Chi-square Tables

Even though it is a bit early in this book to present more complicated programs, we present a short program that allows you to enter the counts for as many 2 × 2 tables as you wish, and compute chi-square statistics for each table. You may copy the program below "as is" and substitute your data lines in place of our sample lines. So, here is the program without any detailed explanations of how it works:

```
*------------------------------------------------------------*
| Program to compute Chi-square for any number of 2 x 2 tables |
| where the data lines consist of the cell frequencies.  The   |
| order of the cell counts is upper left, upper right, lower   |
| left, and lower right.  To use this program, substitute your |
| cell frequencies for the sample data lines in this program.  |
*------------------------------------------------------------*;
DATA CHISQ;
   N + 1;
   DO ROW = 1 TO 2;
      DO COL = 1 TO 2;
         INPUT COUNT @;
         OUTPUT;
      END;
   END;
DATALINES;
3 5 8 6
10 20 30 40
;
PROC FREQ DATA=CHISQ;
   BY N;
   TABLES ROW*COL / CHISQ;
   WEIGHT COUNT;
RUN;
```


K. McNemar's Test for Paired Data

Suppose you want to determine the effect of an anti-cigarette advertisement on people's attitudes towards smoking. In this hypothetical example, we ask 100 people their attitude towards smoking (either positive or negative). We then show them the anti-cigarette advertisement and again ask their smoking attitude. This experimental design is called a paired or matched design since the same subjects are responding to a question under two different conditions (before and after an advertisement). Paired designs are also used when a specific person is matched on some criteria, such as age and gender, to another person for purposes of analysis. Suppose you collected the data below in your cigarette study (P=Positive; N=Negative):

Subject	Before	After
001	P	P
002	P	N
003	N	N
.		
.		
.		
100	N	P

A chi-square test of the type described in Section G is not appropriate here. Instead, a McNemar test for paired designs is needed instead. A SAS program to create the data set and perform the McNemar test is shown next:

```
*----------------------------------------------------*
| Program Name: MCNEMAR.SAS  in C:\APPLIED           |
| Purpose: To perform Mc'Nemars Chi-square test for  |
|          paired samples                            |
*----------------------------------------------------*;
PROC FORMAT;
   VALUE $OPINION 'P'='Positive'
                  'N'='Negative';
RUN;

DATA MCNEMAR;
   LENGTH BEFORE AFTER $ 1;
   INPUT SUBJECT BEFORE $ AFTER $;
   FORMAT BEFORE AFTER $OPINION.;
DATALINES;
001    P    P
002    P    N
003    N    N
 (more data lines)
100    N    P
;
PROC FREQ DATA=MCNEMAR;
   TITLE "McNemar's Test for Paired Samples";
   TABLES BEFORE*AFTER / AGREE;
RUN;
```

Some comments about this program before we study the output. Notice the LENGTH statement immediately before the INPUT statement. By setting the length of the two variables BEFORE and AFTER to 1, we only need a single byte to store the values. Without the LENGTH statement, the default 8 bytes would be used. To make the output more readable, we created a format for the two variables BEFORE and AFTER. Since the title contains a single quote, we used double quotes to surround the title. Finally, the AGREE option on the TABLES statement produces both McNemar's chi-square as well as a measure of agreement called Kappa. (Coefficient Kappa is often used as a measure of interrater reliability.) Now for the output:

```
McNemar's Test for Paired Samples

TABLE OF BEFORE BY AFTER

BEFORE      AFTER

Frequency|
Percent  |
Row Pct  |
Col Pct  |Negative|Positive|  Total
---------+--------+--------+
Negative |     30 |     10 |     40
         |  30.00 |  10.00 |  40.00
         |  75.00 |  25.00 |
         |  40.00 |  40.00 |
---------+--------+--------+
Positive |     45 |     15 |     60
         |  45.00 |  15.00 |  60.00
         |  75.00 |  25.00 |
         |  60.00 |  60.00 |
---------+--------+--------+
Total          75       25      100
            75.00    25.00   100.00

STATISTICS FOR TABLE OF BEFORE BY AFTER

McNemar's Test
--------------
Statistic = 22.273        DF = 1          Prob = 0.001

Simple Kappa Coefficient
------------------------
                                         95% Confidence Bounds
Kappa = -0.000      ASE = 0.077          -0.151      0.151

Sample Size = 100
```

This output shows us that the McNemar's chi-square statistic is 22.273, with a corresponding p-value of .001. We conclude that the anti-cigarette advertisement was effective in changing people's attitudes towards smoking.

If you already had frequency counts for the 2×2 table and wanted SAS to compute McNemar's chi-square, you could use the same technique as in Section I.

L. *Odds Ratios*

Suppose we want to determine if people with a rare brain tumor are more likely to have been exposed to benzene than people without a brain tumor. One experimental design used to answer this question is called a case-control design. As the name implies, you first start with cases, people with a disease or condition (in this example, a brain tumor) and find people who are as similar as possible but who do not have brain tumors. Those people are the controls. We provide some data below to demonstrate some features of a case-control study:

		OUTCOME		
		Case	Control	
EXPOSURE	Yes	50	20	70
	No	100	130	230
		150	150	300

Inspection of the table shows a higher percentage of Cases being exposed to benzene than Controls. To test this, we can compute a chi-square statistic. The results of a case-control study are frequently reported by an odds ratio and a 95% confidence interval about the odds ratio. Briefly, the odds of a Case being exposed to benzene is 50/100. The odds of a Control being exposed to benzene is 20/130. Therefore, the odds ratio is

$$\frac{50/100}{20/130} = \frac{.5}{.155} = 3.25.$$

If we run PROC FREQ with the CHISQ and CMH (Cochran-Mantel-Haenszel) options, we obtain a chi-square statistic, the odds ratio, a 95% CI on the odds ratio, and quite a few additional statistics. A program to analyze the table above and the resulting output is shown next:

```
*Program to compute an Odds Ratio and the 95% CI;
DATA ODDS;
   INPUT OUTCOME $ EXPOSURE $ COUNT;
DATALINES;
CASE 1-YES 50
CASE 2-NO 100
CONTROL 1-YES 20
CONTROL 2-NO 130
;
PROC FREQ DATA=ODDS;
   TITLE 'Program to Compute an Odds Ratio';
   TABLES EXPOSURE*OUTCOME / CHISQ CMH;
   WEIGHT COUNT;
RUN;
```

We used an expedient here to ensure that the 'Yes' row came before the 'No' row. We want the 'Yes' group on top so that the odds ratio will be the odds that a case (in the first column) is more (or less) likely to be exposed (i.e., EXPOSURE = 'Yes'). Since SAS, by default, will order the values in a frequency table by the alphabetical order of character variables, by using the names '1-YES' and '2-NO' for the variable EXPOSURE, we forced the program to place the 'Yes' row on top (since '1' comes before '2', "alphabetically"). Another useful trick is to use formats for the row or column variables, choosing values that result in the desired ordering of rows and columns, and using the ORDER=FORMATTED option with PROC FREQ. This option causes PROC FREQ to use the formatted values rather than the internal raw values when ordering values for tables. The output from this program is as follows:

```
Program to compute an Odds Ratio

TABLE OF EXPOSURE BY OUTCOME

EXPOSURE     OUTCOME

Frequency|
Percent  |
Row Pct  |
Col Pct  |CASE    |CONTROL |  Total
---------+--------+--------+
1-YES    |     50 |     20 |     70
         |  16.67 |   6.67 |  23.33
         |  71.43 |  28.57 |
         |  33.33 |  13.33 |
---------+--------+--------+
2-NO     |    100 |    130 |    230
         |  33.33 |  43.33 |  76.67
         |  43.48 |  56.52 |
         |  66.67 |  86.67 |
---------+--------+--------+
Total         150      150      300
            50.00    50.00   100.00


STATISTICS FOR TABLE OF EXPOSURE BY OUTCOME

Statistic                     DF     Value        Prob
------------------------------------------------------
Chi-Square                     1    16.770       0.001
Likelihood Ratio Chi-Square    1    17.207       0.001
Continuity Adj. Chi-Square     1    15.671       0.001
Mantel-Haenszel Chi-Square     1    16.714       0.001
Fisher's Exact Test (Left)                       1.000
                    (Right)                   3.12E-05
                    (2-Tail)                  6.25E-05
```


[Continued]

```
Phi Coefficient                         0.236
Contingency Coefficient                 0.230
Cramer's V                              0.236

Sample Size = 300

Program to compute an Odds Ratio

SUMMARY STATISTICS FOR EXPOSURE BY OUTCOME


  Cochran-Mantel-Haenszel Statistics (Based on Table Scores)

Statistic   Alternative Hypothesis    DF       Value      Prob
--------------------------------------------------------------
    1       Nonzero Correlation        1      16.714     0.001
    2       Row Mean Scores Differ     1      16.714     0.001
    3       General Association        1      16.714     0.001


      Estimates of the Common Relative Risk (Row1/Row2)
                                                         95%
Type of Study     Method              Value   Confidence Bounds
---------------------------------------------------------------
Case-Control      Mantel-Haenszel     3.250       1.847     5.719
  (Odds Ratio)    Logit               3.250       1.819     5.807

Cohort            Mantel-Haenszel     1.643       1.295     2.084
  (Col1 Risk)     Logit               1.643       1.333     2.025

Cohort            Mantel-Haenszel     0.505       0.364     0.701
  (Col2 Risk)     Logit               0.505       0.343     0.745

The confidence bounds for the M-H estimates are test-based.

Total Sample Size = 300
```

The chi-square value is 16.770, which is significant at the .001 level. As we calculated earlier, the odds ratio is 3.25. To the right of this value is the 95% confidence interval (1.847 to 5.719). We interpret this to mean that if we took many similar samples from the given population, 95% of the computed confidence intervals would contain the true population odds ratio. More practically, we can say that we are 95% confident that the true population odds ratio is in the interval 1.847 to 5.719 (but don't say it too loudly near a statistician). Also, since the interval does not contain 1, we conclude that the odds ratio of 3.25 is significant at the .05 level. (In this case we have a chi-square and p-value we can use.) For a case-control study, the relative risks (labeled cohort) are not interpretable and should be ignored. For very low incidence rates, the odds ratio is an acceptable estimate of the relative risk, discussed next.

M. Relative Risk

Just as an odds ratio is the appropriate statistic when dealing with case-control studies, relative risk is the appropriate statistic when dealing with cohort studies. As an example, suppose we conducted a prospective cohort study to investigate the effect of aspirin on heart attacks. A group of patients who are at risk for a heart attack are randomly assigned to either a placebo (sugar pill) or aspirin. At the end of one year, the number of patients suffering a heart attack (MI, or myocardial infarction) is recorded. Some fictitious data are presented below:

		OUTCOME		
		MI	No MI	
GROUP	Placebo	20	80	100
	Aspirin	15	135	150
		35	215	250

We see that out of 100 patients on placebo, 20 had an MI, giving us an incidence rate of 20/100, or .20. For patients on aspirin, the incidence rate is 15/150, or .10. The ratio of the incidence rates is called relative risk. In this example it is .20/.10 = 2.00. We can say that the risk of a heart attack for people on placebo is twice that of people on aspirin. Another approach would be to place the Aspirin group as the top row and the Placebo group as the bottom row. The column 1 risk (col1 risk in the output) would then represent the protective effect of using aspirin (giving a RR of .5) rather than the increased risk for people who do not take aspirin. We can use basically the same program as we used above to compute the odds ratio to compute the relative risk. Here it is:

```
*Program to compute a Relative Risk and a 95% CI;
DATA RR;
   LENGTH GROUP $ 9;
   INPUT GROUP $ OUTCOME $ COUNT;
DATALINES;
1-PLACEBO MI 20
1-PLACEBO NO-MI 80
2-ASPIRIN MI 15
2-ASPIRIN NO-MI 135
;
PROC FREQ DATA=RR;
   TITLE 'Program to Compute a Relative Risk';
   TABLES GROUP*OUTCOME / CMH;
   WEIGHT COUNT;
RUN;
```

```
Program to Compute a Relative Risk

TABLE OF GROUP BY OUTCOME

GROUP       OUTCOME

Frequency  |
Percent    |
Row Pct    |
Col Pct    |MI      |NO-MI   |  Total
-----------+--------+--------+
1-PLACEBO  |     20 |     80 |    100
           |   8.00 |  32.00 |  40.00
           |  20.00 |  80.00 |
           |  57.14 |  37.21 |
-----------+--------+--------+
2-ASPIRIN  |     15 |    135 |    150
           |   6.00 |  54.00 |  60.00
           |  10.00 |  90.00 |
           |  42.86 |  62.79 |
-----------+--------+--------+
Total            35      215      250
              14.00    86.00   100.00


Program to Compute a Relative Risk

SUMMARY STATISTICS FOR GROUP BY OUTCOME

  Cochran-Mantel-Haenszel Statistics (Based on Table Scores)

Statistic   Alternative Hypothesis    DF       Value       Prob
---------------------------------------------------------------
    1       Nonzero Correlation        1       4.963      0.026
    2       Row Mean Scores Differ     1       4.963      0.026
    3       General Association        1       4.963      0.026

       Estimates of the Common Relative Risk (Row1/Row2)
                                                     95%
Type of Study     Method             Value    Confidence Bounds
---------------------------------------------------------------
Case-Control      Mantel-Haenszel    2.250      1.102    4.592
  (Odds Ratio)    Logit              2.250      1.090    4.643

Cohort            Mantel-Haenszel    2.000      1.087    3.680
  (Col1 Risk)     Logit              2.000      1.076    3.717

Cohort            Mantel-Haenszel    0.889      0.801    0.986
  (Col2 Risk)     Logit              0.889      0.795    0.994

The confidence bounds for the M-H estimates are test-based.

Total Sample Size = 250
```

We used the same labeling trick here with the groups to control the order of the rows in the table. The columns were OK since 'MI' comes before 'NO-MI' alphabetically. The LENGTH statement assigned a length of 9 for the variable GROUP. Had we not done this, the values of GROUP would have defaulted to 8 characters and the last letter of the GROUP values would have been chopped off. An alternative approach is to provide an INFORMAT statement or an INFORMAT on the INPUT statement like this:

```
INFORMAT GROUP $ 9.; /* Separate INFORMAT statement */
INPUT GROUP OUTCOME $ COUNT; /* $ after GROUP optional */
```

or

```
INPUT GROUP : $9. OUTCOME $ COUNT; /* INFORMAT on INPUT line */
```

This time we include only the CMH option, so chi-square and associated statistics are not computed. Since we want to see the "risk" of not using aspirin, we want to look at the cohort (Col1 Risk) results in the output. The Col1 relative risk is how much more or less likely you are to be in the column 1 category (in this case, MI) if you are in the row 1 group (in this case, placebo). The computed relative risk is 2.00 with the 95% confidence interval of 1.087 to 3.680.

N. Chi-square Test for Trend

If the categories in a 2 × N table represent ordinal levels, you may want to compute what is called a chi-square test for trend. That is, are the proportions in each of the N levels increasing or decreasing in a linear fashion? Consider the table below:

		GROUP			
		A	B	C	D
TEST RESULT	Fail	10	15	14	25
	Pass	90	85	86	75
		100	100	100	100

Notice that the proportions of 'Fail' in groups A through D is increasing (except from group B to C). To test if there is a significant linear trend in proportions, we can use the CHISQ option of PROC FREQ and look at the statistic labeled "Mantel-Haenszel chi-square." A program to enter the table values and compute the chi-square test for trend is:

```
*-----------------------------------------------------*
|              Chi-square Test for Trend              |
*-----------------------------------------------------*;
DATA TREND;
   INPUT RESULT $ GROUP $ COUNT @@;
DATALINES;
FAIL A 10 FAIL B 15 FAIL C 14 FAIL D 25
PASS A 90 PASS B 85 PASS C 86 PASS D 75
;
```


[Continued]

```
PROC FREQ DATA=TREND;
   TITLE 'Chi-square Test for Trend';
   TABLES RESULT*GROUP / CHISQ;
   WEIGHT COUNT;
RUN;
```

If you are not familiar with the double "at" sign (@@) notation, see Chapter 12 for more information. Fortunately, the GROUP and RESULT values are already in the proper alphabetical order so we don't have to resort to any expedients. The output is shown below:

```
Chi-square Test for Trend

TABLE OF RESULT BY GROUP

RESULT     GROUP

Frequency|
Percent  |
Row Pct  |
Col Pct  |A       |B       |C       |D       |  Total
---------+--------+--------+--------+--------+
FAIL     |     10 |     15 |     14 |     25 |     64
         |   2.50 |   3.75 |   3.50 |   6.25 |  16.00
         |  15.63 |  23.44 |  21.88 |  39.06 |
         |  10.00 |  15.00 |  14.00 |  25.00 |
---------+--------+--------+--------+--------+
PASS     |     90 |     85 |     86 |     75 |    336
         |  22.50 |  21.25 |  21.50 |  18.75 |  84.00
         |  26.79 |  25.30 |  25.60 |  22.32 |
         |  90.00 |  85.00 |  86.00 |  75.00 |
---------+--------+--------+--------+--------+
Total         100      100      100      100      400
            25.00    25.00    25.00    25.00   100.00


STATISTICS FOR TABLE OF RESULT BY GROUP

Statistic                      DF     Value      Prob
------------------------------------------------------
Chi-Square                      3     9.077     0.028
Likelihood Ratio Chi-Square     3     8.718     0.033
Mantel-Haenszel Chi-Square      1     7.184     0.007
Phi Coefficient                       0.151
Contingency Coefficient               0.149
Cramer's V                            0.151

Sample Size = 400
```

From this output, you can see that the M-H chi-square test for trend is 7.184 (p = .007). There may be times when your table chi-square is not significant but, since the test for trend is using more information (the order of the columns), it may be significant.

O. Mantel-Haenszel Chi-square for Stratified Tables and Meta Analysis

You may have a series of 2 × 2 tables for each level of another factor. This may be a confounding factor such as age, or you may have a 2 × 2 table at each site in a multi-site study. In any event, one way to analyze multiple 2 × 2 tables of this sort is to compute a Mantel-Haenszel chi-square. This same technique can also be used to combine results from several studies identified in a literature search on a specific topic. Although the studies may have some minor differences, you may prefer to ignore those differences and combine the results anyway. This technique is sometimes referred to as meta-analysis and is becoming quite popular in medicine, education, and psychology. There are **lots** of cautions concerning this technique, but this is neither the time nor the place to discuss them. The Mantel-Haenszel statistic is also used frequently for item bias research.

It is actually easy to compute a chi-square for stratified tables using SAS software. All that is necessary is to request a three-way table with PROC FREQ and to include the option ALL.

As an example, suppose we have just two 2 × 2 tables. One for boys and the other for girls. Each table represents the relationship between hours of sleep and the chance of failing a test of physical ability. We want to investigate the risk factor (lack of sleep) on the outcome (failing a test). The hours of sleep variable has been dichotomized as 'LOW' (less than 8 hours) and 'HIGH' (more than 8 hours). Here are the two tables:

		Test Results, Boys	
		FAIL	PASS
SLEEP	LOW	20	100
	HIGH	15	150

	Test Results, Girls	
	FAIL	PASS
LOW	30	100
HIGH	25	200

Our SAS program is straightforward:

```
*Program to compute a Mantel-Haenszel Chi-square Test
 for Stratified Tables;
DATA ABILITY;
   INPUT GENDER $ RESULTS $ SLEEP $ COUNT;
DATALINES;
BOYS FAIL 1-LOW 20
BOYS FAIL 2-HIGH 15
BOYS PASS 1-LOW 100
```


[Continued]

```
BOYS PASS 2-HIGH 150
GIRLS FAIL 1-LOW 30
GIRLS FAIL 2-HIGH 25
GIRLS PASS 1-LOW 100
GIRLS PASS 2-HIGH 200
;
PROC FREQ DATA=ABILITY;
   TITLE 'Mantel-Haenszel Chi-square Test';
   TABLES GENDER*SLEEP*RESULTS / ALL;
   WEIGHT COUNT;
RUN;
```

Since the output from this program is voluminous, we edit it and show you the relevant portions:

```
Mantel-Haenszel Chi-square Test

TABLE 1 OF SLEEP BY RESULTS
CONTROLLING FOR GENDER=BOYS

Statistic                      DF      Value        Prob
--------------------------------------------------------
Chi-Square                      1      3.701       0.054

      Estimates of the Relative Risk (Row1/Row2)

                                              95%
Type of Study            Value       Confidence  Bounds
--------------------------------------------------------
Case-Control             2.000        0.978      4.091
Cohort (Col1 Risk)       1.833        0.980      3.431
Cohort (Col2 Risk)       0.917        0.835      1.006

TABLE 2 OF SLEEP BY RESULTS
CONTROLLING FOR GENDER=GIRLS

Statistic                      DF      Value        Prob
--------------------------------------------------------
Chi-Square                      1      9.011       0.003

      Estimates of the Relative Risk (Row1/Row2)

                                              95%
Type of Study            Value       Confidence  Bounds
--------------------------------------------------------
Case-Control             2.400        1.340      4.297
Cohort (Col1 Risk)       2.077        1.279      3.373
Cohort (Col2 Risk)       0.865        0.779      0.961
```

[Continued]

```
SUMMARY STATISTICS FOR SLEEP BY RESULTS
CONTROLLING FOR GENDER

  Cochran-Mantel-Haenszel Statistics (Based on Table Scores)

Statistic  Alternative Hypothesis   DF      Value      Prob
------------------------------------------------------------
   1       Nonzero Correlation       1     12.477     0.001
   2       Row Mean Scores Differ    1     12.477     0.001
   3       General Association       1     12.477     0.001

       Estimates of the Common Relative Risk (Row1/Row2)

                                                  95%
Type of Study   Method              Value   Confidence Bounds
-------------------------------------------------------------
Case-Control    Mantel-Haenszel     2.229     1.429    3.477
  (Odds Ratio)  Logit               2.232     1.421    3.506
Cohort          Mantel-Haenszel     1.977     1.355    2.887
  (Col1 Risk)   Logit               1.982     1.351    2.909
Cohort          Mantel-Haenszel     0.889     0.833    0.949
  (Col2 Risk)   Logit               0.894     0.833    0.958
The confidence bounds for the M-H estimates are test-based.

Breslow-Day Test for Homogeneity of the Odds Ratios

Chi-Square =    0.150          DF =    1         Prob = 0.698

Total Sample Size = 640
```

As you can see, the results for boys gives us a chi-square of 3.701 (p = .054) with a relative risk of 1.833 (95% CI .980, 3.431). (We choose the Col1 Risk value since we want the RR for failing the test, which is the value for column 1.) For girls, the results are a chi-square of 9.011 (p = .003) and a relative risk of 2.077 (95% CI 1.279, 3.373).

The results of combining the two tables is found under the heading "SUMMARY STATISTICS." Looking at the Cohort Mantel-Haenszel statistics (col1 risk), we find a p-value of .001 and a relative risk of 1.977 (95% CI 1.355, 2.887). Notice that p-value from the combined genders is lower than that for either the boys or girls alone. The Breslow-Day test for homogeneity of the odds ratio is not significant (p = .698), so we can be comfortable in combining these two tables.

P. "Check All That Apply" Questions

A common problem in questionnaire analysis is "check all that apply" questions. For example, suppose we had a question asking respondents which course or courses they were interested in taking. It might be written:


```
Which course, or courses, would you like to see
offered next semester?

                  (Check ALL that apply)

___ 1. Micro-computers             ___ 4. Job Control Language
___ 2. Intro to SAS                ___ 5. FORTRAN
___ 3. Advanced SAS                ___ 6. PASCAL
```

Insofar as our analysis is concerned, this is not one question with up to six answers, but six yes/no questions. Each course offering is treated as a variable with values of YES or NO (coded as 1 or 0, for example). Our questionnaire would be easier to analyze if it were arranged thus:

```
Please indicate which of the following courses you
would like to see offered next semester:

                                                 | For office
                           (1) yes    (0) no     |  use only
a) Micro-computers           ___        ___      |    ___
b) Intro to SAS              ___        ___      |    ___
c) Advanced SAS              ___        ___      |    ___
d) Job Control Language      ___        ___      |    ___
e) FORTRAN                   ___        ___      |    ___
f) PASCAL                    ___        ___      |    ___
```

Our INPUT statement would have six variables (COURSE1-COURSE6 for instance), each with a value of 1 or 0. A format matching 1 = yes and 0 = no would lend readability to the final analysis.

This approach works well when there is a limited number of choices. However, when we are choosing several items from a large number of possible choices, this approach becomes impractical since we need a variable for every possible choice. A common example in the medical field would be the variable "diagnosis" on a patient record. We might have a list of hundreds or even thousands of diagnosis codes and want to consider a maximum of two or three diagnoses for each patient. Our approach in this case would be to ask for up to three diagnoses per patient, using a diagnosis code from a list of standardized codes. Our form might look like this:

```
Enter up to 3 diagnosis codes for the patient.

Diagnosis 1 ____________________    | [ ] [ ] [ ]

Diagnosis 2 ____________________    | [ ] [ ] [ ]

Diagnosis 3 ____________________    | [ ] [ ] [ ]
```

Our INPUT statement would be straightforward:

```
DATA DIAG1;
   INPUT ID 1-3 . . . DX1 20-22 DX2 23-25 DX3 26-28;
```

Suppose we had the following data:

OBS	ID	DX1	DX2	DX3
1	1	3	4	.
2	2	1	3	7
3	3	5	.	.

Notice that one patient could have a certain diagnosis code as his first diagnosis whereas another patient might have the same code as his second or third diagnosis. If we want a frequency distribution of diagnosis codes, what can we do? We could try this:

```
PROC FREQ DATA=DIAG1;
   TABLES DX1-DX3;
RUN;
```

Using this method, we would have to add the frequencies from three tables to compute the frequency for each diagnosis code. A better approach would be to create a separate data set that is structured differently. Our goal would be a data set like:

OBS	ID	DX
1	1	3
2	1	4
3	2	1
4	2	3
5	2	7
6	3	5

The program statements to create the data set above are:


```
DATA DIAG2;
   SET DIAG1;
   DX=DX1;
   IF DX NE . THEN OUTPUT;
   DX=DX2;
   IF DX NE . THEN OUTPUT;
   DX=DX3;
   IF DX NE . THEN OUTPUT;
   KEEP ID DX;
RUN;
```

Each observation from our original data set (DIAG1) will create up to three observations in our new data set (DIAG2). The SET statement (line 2) reads an observation from our original data set (DIAG1). The first observation will contain:

ID	AGE	GENDER	DX1	DX2	DX3
1	23	M	3	4	.

To these values, we add a new variable called DX, and set it equal to DX1. The Program Data Vector now contains:

ID	AGE	GENDER	DX1	DX2	DX3	DX
1	23	M	3	4	.	3

Since DX is not missing, we output (line 4) these values to the first observation in data set DIAG2 (for now, let's ignore the KEEP statement in line 9). Next, DX is set to DX2, and another observation is written to data set DIAG2:

ID	AGE	GENDER	DX1	DX2	DX3	DX
1	23	M	3	4	.	3
1	23	M	3	4	.	4

The values for ID, AGE, GENDER, DX1, DX2, and DX3 are still the same, only DX has changed. Finally, DX is set equal to DX3, but since DX3, and therefore DX, is missing, no observation is written to DIAG2.

We are now at the bottom of the DATA step, and control is returned to the top of the DATA step, where the SET statement will read another observation from data set DIAG1. This routine continues until all the observations from DIAG1 have been read. Since we had a KEEP statement in the program, only the variables ID and DX actually exist in data set DIAG2:

OBS	ID	DX
1	1	3
2	1	4
3	2	1
4	2	3
5	2	7
6	3	5

We can now count the frequencies of DX codes using PROC FREQ with DX as the TABLE variable.

This program can be made more compact by using an ARRAY and a DO loop. (See Chapter 15 for a detailed explanation of ARRAYS, and Chapter 16, Section C, for details on restructuring SAS data sets using arrays.)

```
DATA DIAG2;
   SET DIAG1;
   ARRAY D[*] DX1-DX3;
   DO I = 1 TO 3;
     DX=D[I];
     IF D[I] NE . THEN OUTPUT;
   END;
KEEP ID DX;
RUN;
```

Problems

3-1. Suppose we have a variable called GROUP that has numeric values of 1, 2, or 3. Group 1 is a control group, group 2 is given aspirin, and group 3 is given ibuprofen. Create a format to be assigned to the GROUP variable.

3-2. A survey is conducted and data are collected and coded. The data layout is shown below:

Variable	Description	Columns	Coding Values
ID	Subject identifier	1-3	
GENDER		4	M=Male F=Female
PARTY	Political party	5	'1'=Republican '2'=Democrat '3'=Not registered
VOTE	Did you vote in the last election?	6	0=No 1=Yes
FOREIGN	Do you agree with the government's foreign policy?	7	0=No 1=Yes
SPEND	Should we increase domestic spending?	8	0=No 1=Yes

Collected data are shown next:


```
007M1110
013F2101
137F1001
117 1111
428M3110
017F3101
037M2101
```

(a) Create a SAS data set, complete with labels and formats for this questionnaire.

(b) Generate frequency counts for the variables GENDER, PARTY, VOTE, FOREIGN, and SPEND.

(c) Test if there is a relationship between voting in the last election versus agreement with spending and foreign policy. (Have SAS compute chi-square for these relationships.)

3-3. Run the program below to create a SAS data set called DEMOG:

```
DATA DEMOG;
   INPUT WEIGHT HEIGHT GENDER $;
DATALINES;
155 68 M
 98 60 F
202 72 M
280 75 M
130 63 F
;
```

We want to recode WEIGHT and HEIGHT as follows. (Assume that WEIGHT and HEIGHT are integer values only.):

```
WEIGHT    0-100   = 1
          101-150 = 2
          151-200 = 3
            >200  = 4

HEIGHT    0-70    = 1
           >70    = 2
```

We then want to generate a table of WEIGHT categories (rows) by HEIGHT categories (columns). Recode these variables in two ways: (1) with "IF" statements; (2) with formats. Then write the necessary statements to generate the table.

3-4. A friend gives you some summary data on the relationship between socio-economic status (SES) and asthma, as follows:

Asthma	Yes	No
Low SES	40	100
High SES	30	130

Create a SAS data set from these data and compute chi-square.

3-5. A matched-pairs case-control study is conducted. Each case (a person with disease X) is matched to a single control, based on age (plus or minus 2 years) and gender. Each person is then asked if he/she used multivitamins regularly in the past year. The results are:

Case Use of Vitamins	Matched Controls Use of Vitamins	Count
Yes	Yes	100
Yes	No	50
No	Yes	90
No	No	200

Remembering that this is a matched study, compute a McNemar chi-square. Are the subjects more or less likely to have used vitamins?

3-6. A researcher wants to determine if there is a relationship between use of computer terminals (VDT—video display terminals) and miscarriages. An unpaired, case-control study is conducted. Of the cases (women with miscarriages) there were 30 women who used VDT's and 50 who did not. Among the controls, there were 90 women who used VDT's and 200 who did not. Compute chi-square, the odds ratio, and a 95% confidence interval for the OR.

3-7. A researcher wants to determine if sound-proofing a classroom leads to better behavior among the students. In the standard classrooms, there were 30 behavioral problems from a total of 250 children. In the sound-proofed classrooms, there were 20 behavioral problems from a total of 300 children. Test if room noise results in an increased number of behavioral problems, by computing the relative risk (of noisy classrooms) for producing behavioral problems. Have SAS produce a 95% CI for the relative risk as well.

3-8. A school administrator believes that larger class sizes lead to more discipline problems. Four class sizes (small, medium, large, and gigantic) are tested. The table below summarizes the problems recorded for each of the class sizes. Treating class size as an ordinal variable, test if there is a linear increase in the proportion of class problems. (Be careful to arrange the order of the class size columns so that they range from small to gigantic, or your test for trend will not be valid.)

Class Size	Small	Medium	Large	Gigantic
Problem	3	6	17	80
No Problem	12	22	38	120
Total	15	28	55	200

3-9. The relationship between office temperature and head colds is tested for smokers and nonsmokers. Since smoking is assumed to be a confounding factor in this relationship, use a Mantel-Haenszel chi-square for stratified tables to analyze the two tables below:

	SMOKERS Colds	SMOKERS No Colds	NONSMOKERS Colds	NONSMOKERS No Colds
Poor Temperature Control	30	50	40	100
Good Temperature Control	20	55	35	150

*3-10. A physical exam is given to a group of patients. Each patient is diagnosed as having none, one, two, or three problems from the code list below:

Code	Problem Description
1	Cold
2	Flu
3	Trouble sleeping
4	Chest pain
5	Muscle pain
6	Headaches
7	Overweight
8	High blood pressure
9	Hearing loss

The coding scheme is as follows:

Variable	Description	Column (s)
SUBJ	Subject number	1–2
PROB1	Problem 1	3
PROB2	Problem 2	4
PROB3	Problem 3	5
HR	Heart rate	6–8
SBP	Systolic blood pressure	9–11
DBP	Diastolic blood pressure	12–14

Using the sample data below:

```
Columns     12345678901234
------------------------------
            11127 78130 80
            1787  82180110
            031   62120 78
            4261  68130 80
            89    58120 76
            9948  82178100
```

(a) Compute the mean HR, SBP, and DBP.

(b) Generate frequency counts for each of the nine medical problems.

3-11. What's wrong with this program?

```
DATA IGOOFED;
INPUT #1 ID 1-3 GENDER 4 AGE 5-6 RACE 7 (QUES1-QUES10) (1.)
      #2 @4 (QUES11-QUES25) (1.);
FORMAT GENDER SEX. RACE RACE. QUES1-QUES25 YESNO.;
DATALINES;
00112311010011101
   1100111001101011
00224421011101110O
   0111011101111010
;
PROC FORMAT;
   VALUE SEX 1='MALE' 2='FEMALE';
   VALUE RACE 1='WHITE' 2='AFRICAN AM.' 3='HISPANIC';
   VALUE YESNO 0='NO' 1='YES';
RUN;
PROC FREQ DATA=IGOOFED;
   VAR GENDER RACE QUES1-QUES25 / CHISQ;
RUN;
PROC MEANS MAXDEC=2 N MEAN STD MIN MAX;
   BY RACE;
   VAR AGE;
RUN;
```

HINTS AND COMMENTS: The INPUT statement is correct. The pointers (@ signs) and format lists (1.) are explained in Chapter 12. There are four errors.

CHAPTER

4

Working with Date and Longitudinal Data

A. *Introduction*

Working with dates is a task which some data analysts face frequently. SAS software contains many powerful resources for working with dates. These include the ability to read dates in almost any form, or to compute the number of days, months, or years between any two dates.

Data collected for the same set of subjects at different times are sometimes called longitudinal data. These data require specialized techniques for analysis. Let's begin by seeing how date values are handled with SAS software.

B. *Processing Date Variables*

Suppose you want to read the following information into a SAS data set:

ID	DOB	Admitting Date	Discharge Date	DX	Fee
001	102150	091090	091390	8	3000
002	050540	090190	090590	7	5000

We can arrange this data in columns, thus:

Variable Name	Description	Column(s)
ID	Patient ID	1–3
DOB	Date of Birth	5–10
ADMIT	Date of Admission	12–17
DISCHRG	Discharge Date	18–23
DX	Diagnosis	25
FEE	Hospital Fee	26–30

You might be tempted to write an input statement like:

```
INPUT ID 1-3 DOB 5-10 ADMIT 12-17 DISCHRG 18-23 DX 25 FEE 26-30;
```

However, the six digits of the date values have no meaning when read as a six-digit number. SAS software includes extensive provisions for working with date and time variables. The first step in reading date values is to tell the program how the date value is written. Common examples are:

Example	Explanation	SAS INFORMAT
102150	Month – Day – 2 digit Year	MMDDYY6.
10211950	Month – Day – 4 digit Year	MMDDYY8.
10/21/50	Month – Day – 2 digit Year	MMDDYY8.
10/21/1950	Month – Day – 4 digit Year	MMDDYY10.
211050	Day – Month – 2 digit Year	DDMMYY6.
21101950	Day – Month – 4 digit Year	DDMMYY8.
501021	2 digit Year – Month – Day	YYMMDD6.
19501021	4 digit Year – Month – Day	YYMMDD8.
21OCT50	Day, 3 character Month, 2 digit Year	DATE7.
21OCT1950	Day, 3 character Month, 4 digit Year	DATE9.
OCT50	Month and 4 digit Year only	MONYY5.
OCT1950	Month and 4 digit Year only	MONYY7.

SAS programs can read any of these dates, provided we instruct it which form of date we are using. Once SAS knows it's reading a date, that date is converted into the number of days from a fixed point in time: January 1, 1960. It doesn't matter if your date comes before or after this date. Dates before January 1, 1960 will be negative numbers. Therefore we can subtract any two dates to find the number of days in between. We can also convert the SAS internal date value back to any of the allowable SAS date formats for reporting purposes. As we saw in the last chapter, we can associate formats with variables for printing. We can specify how to read values by specifying an INFORMAT to give the program instructions on how to read a data value. The SAS date informat MMDDYY6., for example, is used to read dates in month-day-year order. The 6 in the informat refers to the number of columns occupied by the date. Thus far, we have used space-between-the-numbers and column specifications to instruct SAS how to read our data values.

To read date values, we can use pointers and informats. A column pointer (@) first tells the program which column to start reading. We follow this with the variable name and a specification of what type of data we are reading, called an informat. Two very common informats are W.n and $W. W.n is a numeric informat which says to read W columns of data and to place n digits to the right of the decimal point. For example, the informat 6.1 says to read six columns and to place a decimal point before the last digit. If you are not specifying a decimal point, you do not have to specify the value to the right of the period. Thus, an informat of 6. is equivalent to 6.0 The $W. informat is used to read W columns of character data. The informat MMDDYY8. is used for dates in the form 10/21/50 or 10–21–50. Using our column assignments above, a valid INPUT statement would be:

```
INPUT @1 ID 3. @5 DOB MMDDYY6. @12 ADMIT MMDDYY6.
      @18 DISCHRG MMDDYY6. @25 DX 1. @26 FEE 5.;
```


Or to make it more readable:

```
INPUT @1  ID      3.
      @5  DOB     MMDDYY6.
      @12 ADMIT   MMDDYY6.
      @18 DISCHRG MMDDYY6.
      @25 DX      1.
      @26 FEE     5.;
```

The @ signs, referred to as column pointers, tell the program at which column to start reading the next data value. Our three dates are all in the form MMDDYY (month-day-year) and occupy six columns, so the MMDDYY6. informat is used. Remember that all of our formats and informats end with periods to distinguish them from variable names. All the column pointers in the program above are not necessary. For example, the program would start reading in column 1 without the @1 pointer. Also, since the date of admission ends in column 17 and the discharge date starts in column 18, the pointer @18 is redundant. Good programming practice suggests that using a column pointer with every variable is a good idea. (See Chapter 12 for more details on how to use the INPUT statement.)

Let's calculate two new variables from these data. First, we compute our subject's age at the time of admission. Next, we compute the length of stay in the hospital. All we have to do is subtract any two dates to obtain the number of days in between. Therefore, our completed program looks like:

```
DATA HOSPITAL;
INPUT @1  ID      3.
      @5  DOB     MMDDYY6.
      @12 ADMIT   MMDDYY6.
      @18 DISCHRG MMDDYY6.
      @25 DX      1.
      @26 FEE     5.;
LEN_STAY = DISCHRG - ADMIT + 1;
AGE = ADMIT - DOB;
DATALINES;
(data lines)
```

The calculation for length of stay (LEN_STAY) is relatively straightforward. We subtract the admission date from the discharge date and add 1 (we want to count both the admission day and the discharge day in the length of stay computation). The age calculation is a little more complicated.

By subtracting the date of birth from the admission date, we have the number of days between these two dates. We could convert this to years by dividing by 365.25 (approximately correct since there is a leap year every 4 years). This would give us:

```
AGE = (ADMIT - DOB) / 365.25;
```

We may want to define age so that a person is not considered, say 18 years old, until his 18th birthday. That is, we want to omit any fractional portion of his/her age in years. A SAS function that does this is the INT (integer) function. We can write:

```
AGE = INT((ADMIT - DOB) / 365.25);
```

The INT function, like all SAS functions, has a pair of parentheses after the function name. If we wanted to round the age to the nearest tenth of a year, we would use the ROUND function. This function has two arguments: the number to be rounded and the roundoff unit. To round to the nearest tenth of a year, we use:

```
AGE = ROUND (((ADMIT - DOB) / 365.25),.1);
```

To the nearest year, the function would be:

```
AGE = ROUND (((ADMIT - DOB) / 365.25),1);
```

To compute an AGE as of a specific date, say January 1, 1997, you can use a SAS date literal. A date literal is of the form 'nnMMMyy'D where nn is a two-digit day, MMM is a three-character month abbreviation, and yy is a two- or four-digit year. This string is placed in single or double quotes and followed by an upper or lower case 'D'. To compute the age of a subject as of January 1, 1997, whose date of birth is stored in the variable DOB, you write:

```
AGE = INT (('01JAN97'D - DOB) / 365.25);
```

It is important to remember that once SAS has converted our dates into the number of days from January 1, 1960, it is stored just like any other numeric value. Therefore, if we print it out (with PROC PRINT, for example), the result will not look like a date. We need to supply SAS with a format to use when printing our dates. This format does not have to be the same one we used to read the date in the first place. Some useful date formats and their results are shown in the table below:

The Date 10/21/50 Printed with Different Date Formats:

Format	Result
MMDDYY6.	102150
MMDDYY8.	10/21/50
DATE7.	21OCT50
DATE9.	21OCT1950
WORDDATE.	October 21, 1950

You may also use any of the INFORMATS listed earlier in this section as formats for date values.

In the age calculation above, what age would be computed for a person born on January 1, 1899, who was admitted on January 1, 1990? If we entered his/her date of birth as 01/01/99, the computer would assume birth in 1999, not 1899, and compute an age of −9 years. A person born in 1880, admitted on the same date, would be listed as 10 years old. Clearly, we need a way to include century data for people born before 1900. We can do this in several ways. First, we can use a four-digit year. We could code the original date in a form such as:

01/01/1899 or 01011899

We could read the first date with the MMDDYY10. informat, and the second with the MMDDYY8. informat. You may recall that MMDDYY8. is also used to read a date such as:

10/21/50

However, the system is smart enough to realize that an eight-digit date without slashes must be using a four-digit year.

Another method is to provide a variable that indicates the century, and include it in our calculation. Suppose a variable called CENTURY has values of 18 or 19 to indicate if the subject was born in the 1800's or the 1900's. One way to compute the person's age would be:

```
AGE = (ADMIT - DOB) / 365.25;
IF CENTURY = 18 THEN AGE = AGE + 100;
```

A more compact equation would be:

```
AGE = (ADMIT - DOB) / 365.25 + 100 * (CENTURY=18);
```

Notice the expression (CENTURY=18). This is called a logical expression; it is the first time we have combined a logical expression in a calculation. The result of the logical expression (CENTURY=18) is evaluated. A true value equals 1, and a false value equals 0. So, if a subject was born in the 1800's, we would add 100 to their age. This is a good place to point out that the two-line computation of age is perfectly fine. Only a compulsive programmer (like one of the authors) would feel the need to write a more elegant, one-line expression. There are times when more lengthy, straightforward approaches are best. They are less likely to contain errors, are easier for other people to understand and, finally, easier for the original programmer to modify or debug at a later date.

If you know something about your data and know, for example, that there are no subjects over 100 years old, you can simply check if any ages are negative (subjects born before 1900) and add 100. However, if your data base could conceivably contain 2 year olds and 102 year olds, you must include a four-digit year or a century variable in your calculations.

A fairly recent addition to the SAS language is the YEARCUTOFF option. If you are using two-digit dates, you can indicate what range of 100 years to assume your date values represent. For example, if you include the statement:

```
OPTIONS YEARCUTOFF=1910;
```

in a SAS program, it assumes that any two-digit date falls within the range 1910 to 2010. So, the date 10/21/05 would be assumed to be 10/21/2005. The date 10/21/46 would be set to 10/21/1946. So, with a little care you don't have to worry about the dreaded year 2000 problem!

Chapter 17, Section D, contains a list of SAS functions that are useful when working with dates. For example, month, day, and year values can be combined to form a SAS date, or you can extract a month or year from a SAS date.

C. Longitudinal Data

There is a type of data, often referred to as longitudinal data, that needs special attention. Longitudinal data are collected on a group of subjects over time. CAUTION: This portion of the chapter is difficult and may be "hazardous to your health"!

To examine the special techniques needed to analyze longitudinal data, let's follow a simple example. Suppose we are collecting data on a group of patients. (The same scheme would be applicable to periodic data in business or data in the social sciences with repeated measures.) Each time the patients come in for a visit, we fill out an encounter form. The data items we collect are:

```
PATIENT ID
DATE OF VISIT (Month Day Year)
HEART RATE
SYSTOLIC BLOOD PRESSURE
DIASTOLIC BLOOD PRESSURE
DIAGNOSIS CODE
DOCTOR FEE
LAB FEE
```

Now, suppose each patient's visits are a maximum of four times a year. One way to arrange our SAS data set is as follows (each visit to occupy a separate line):

```
DATA HOSP;
  INPUT #1 ID1 1-3 DATE1 MMDDYY6. HR1 10-12 SBP1 13-15 DBP1 16-18
           DX1 19-21 DOCFEE1 22-25 LABFEE1 26-29
        #2 ID2 1-3 DATE2 MMDDYY6. HR2 10-12 SBP2 13-15 DBP2 16-18
           DX2 19-21 DOCFEE2 22-25 LABFEE2 26-29
        #3 ID3 1-3 DATE3 MMDDYY6. HR3 10-12 SBP3 13-15 DBP3 16-18
           DX3 19-21 DOCFEE3 22-25 LABFEE3 26-29
        #4 ID4 1-3 DATE4 MMDDYY6. HR4 10-12 SBP4 13-15 DBP4 16-18
           DX4 19-21 DOCFEE4 22-25 LABFEE4 26-29;
 FORMAT DATE1-DATE4 MMDDYY8.;
DATALINES;
00710218307012008001400400150
00712018307213009002000500200
007
007
00909038306611007013700300000
009
```


[Continued]

```
009
009
00507058307414008201300900000
00501158208018009601402001500
00506188207017008401400800400
00507038306414008401400800200
;
```

The number signs (#) in the INPUT statement signify multiple lines per subject. Since our date is in the form month-day-year, we use the MMDDYY6. informat. Notice that we can combine pointers and informats with the other style of column specification. We also included an output format for our dates with a FORMAT statement. This FORMAT statement uses the same syntax as the earlier examples in this chapter, where we created our own formats. The output format MMDDYY8. specifies that the date variables be printed in month-day-year form.

With this method of one line per patient visit, we would need to insert BLANK lines of data for any patient who had less than four visits, to fill out four lines per subject. This is not only clumsy but occupies a lot of space unnecessarily. If we want to compute within-subject means, we continue (before the DATALINES statement):

```
AVEHR = MEAN (OF HR1-HR4);
AVESBP = MEAN (OF SBP1-SBP4);
AVEDBP = MEAN (OF DBP1-DBP4);
        etc.
```

"MEAN" is one of the SAS built-in functions that will compute the mean of all the variables listed within parentheses. (NOTE: If any of the variables listed as arguments of the MEAN function have missing values, the result will be the mean of the nonmissing values. See Chapter 17, Section B for more details.) A much better approach is to treat each visit as a separate observation. Our program would then look like:

```
DATA PATIENTS;
   INPUT @1 ID      3.
         @4 DATE    MMDDYY6.
         @10 HR     3.
         @13 SBP    3.
         @16 DBP    3.
         @19 DX     3.
         @22 DOCFEE 4.
         @26 LABFEE 4.;
   FORMAT DATE MMDDYY8.;
DATALINES;
```

Now we need to include only as many lines of data as there are patient visits; blank lines to fill out four lines per subject are not needed. Our variable names are also simpler since we do not need to keep track of HR1, HR2, etc. How do we analyze on this data set?

A simple PROC MEANS on a variable such as HR,SBP, or DOCFEE will not be particularly useful since we are averaging one to four values per patient together. Perhaps the average of DOCFEE would be useful since it represents the average doctor fee per PATIENT VISIT, but statistics for heart rate or blood pressure would be a weighted average, the weight depending on how many visits we had for each patient. How do we compute the average heart rate or blood pressure per patient? The key is to use ID as a CLASS or BY variable.

Here is our program (with sample data):

```
DATA PATIENTS;
   INPUT @1  ID      3.
         @4  DATE    MMDDYY6.
         @10 HR      3.
         @13 SBP     3.
         @16 DBP     3.
         @19 DX      3.
         @22 DOCFEE 4.
         @26 LABFEE 4.;
  FORMAT DATE MMDDYY8.;
DATALINES;
00710218307012008001400400150
00712018307213009002000500200
00909038306611007013700300000
00507058307414008201300900000
00501158208018009601402001500
00506188207017008401400800400
00507038306414008401400800200
;
PROC MEANS DATA=PATIENTS NOPRINT NWAY;
  CLASS ID;
  VAR HR -- DBP DOCFEE LABFEE;
  OUTPUT OUT=STATS MEAN=M_HR M_SBP M_DBP M_DOCFEE M_LABFEE;
RUN;
```

The result is the mean HR, SBP, etc., per patient, which is placed in a new data set STATS, with variable names M_HR, etc. (See Chapter 2, Sections H and I, for more about using PROC MEANS to create output data sets.) Each variable listed after "MEAN=" in the OUTPUT statement will be the mean of the variables listed in the VAR statement, in the order they appear. Thus, M_HR in the data set STATS is the mean HR in the data set PATIENTS. In this example, the data set STATS would appear as follows (we can always test this with a PROC PRINT statement):

Listing of STATS data set

OBS	ID	_TYPE_	_FREQ_	M_HR	M_SBP	M_DBP	M_DOCFEE	M_LABFEE
1	5	1	4	72	157.5	86.5	112.5	525
2	7	1	2	71	125.0	85.0	45.0	175
3	9	1	1	66	110.0	70.0	30.0	0

This data set contains the mean HR, SBP, etc., per patient. We could analyze this data set with additional SAS procedures to investigate relationships between variables or compute descriptive statistics where each data value corresponds to a single value (the MEAN) from each patient.

D. *Most Recent (or Last) Visit per Patient*

What if we want to analyze the most recent visit for each patient in the data set PATIENTS above? If we sort data set in patient-date order, the most recent visit would be the last observation for each patient ID. We can extract these observations with the following SAS program:

```
DATA PATIENTS;
   INPUT @1  ID      3.
         @4  DATE    MMDDYY6.
         @10 HR      3.
         @13 SBP     3.
         @16 DBP     3.
         @19 DX      3.
         @22 DOCFEE 4.
         @26 LABFEE 4.;
     FORMAT DATE MMDDYY8. ;
DATALINES;
(data lines)
;
PROC SORT DATA=PATIENTS;
   BY ID DATE;
RUN;

DATA RECENT; ①
   SET PATIENTS; ②
   BY ID; ③
   IF LAST.ID; ④
RUN;
```

Statement ① creates a new data set called RECENT. As we explained previously, the SET statement ② acts like an INPUT statement except that observations are read one by one from the SAS data set PATIENTS instead of our original raw data.

We are permitted to use a BY ③ variable following our SET statement, provided our data set has been previously sorted by the same variable (it has). The effect of adding the BY statement permits the use of the special FIRST. and LAST. internal SAS logical variables. Since ID is our BY variable, FIRST.ID will be a logical variable (i.e., true or false: 1 or 0) that will be part of each observation as it is being processed but will not remain with the final data set RECENT. FIRST.ID will be TRUE (or 1) whenever we are reading a new ID; LAST.ID will be TRUE whenever we are reading the last observation for a given ID. To clarify this, here are our observations and the value of FIRST.ID and LAST.ID for each case:

OBS	ID	DATE	HR	SBP	DBP	DX	DOCFEE	LABFEE	FIRST.ID	LAST.ID
1	5	01/15/82	80	180	96	14	200	1500	1	0
2	5	06/18/82	70	170	84	14	80	400	0	0
3	5	07/03/83	64	140	84	14	80	200	0	0
4	5	07/05/83	74	140	82	13	90	0	0	1
5	7	10/21/83	70	120	80	14	40	150	1	0
6	7	12/01/83	72	130	90	20	50	200	0	1
7	9	09/03/83	66	110	70	137	30	0	1	1

By adding the IF statement ④, we can select the last visit for each patient (in this case, observations 4, 6, and 7). By the way, this IF statement may also look strange—there is no logical expression following it. Since the value of FIRST. and LAST. variables are true (1) or false (0), they can be placed directly in an IF statement without the more traditional form:

```
IF LAST.ID = 1;
```

You would normally expect to see a subsetting IF statement with the equals sign included. We could also select the first visit for each patient with the statement:

```
IF FIRST.ID;
```

E. Computing Frequencies on Longitudinal Data Sets

To compute frequencies for our diagnoses, we use PROC FREQ on our original data set (PATIENTS). We would write:

```
PROC FREQ DATA=PATIENTS ORDER=FREQ;
   TITLE 'Diagnoses in Decreasing Frequency Order';
   TABLES DX;
RUN;
```


Notice we use the DATA= option on the PROC FREQ statement to make sure we were counting frequencies from our original data set. The ORDER= option allows us to control the order of the categories in a PROC FREQ output. Normally, the diagnosis categories are listed in numerical order. The ORDER=FREQ option lists the diagnoses in frequency order from the most common diagnosis to the least. While we are on the subject, another useful ORDER= option is ORDER=FORMATTED. This will order the diagnoses alphabetically by the diagnosis formats (if we had them). Remember that executing a PROC FREQ procedure on our original data set (the one with multiple visits per patient) has the effect of counting the number of times each diagnosis is made. That is, if a patient came in for three visits, each with the same diagnosis, we would add 3 to the frequency count for that diagnosis. If, for some reason, we want to count a diagnosis only once for a given patient, even if this diagnosis is made on subsequent visits, we can sort our data set by ID and DX. We then have a data set such as the following:

ID	DX	FIRST.ID	FIRST.DX
5	13	1	1
5	14	0	1
5	14	0	0
5	14	0	0
7	14	1	1
7	20	0	1
9	137	1	1

If we now use the logical FIRST.DX and FIRST.ID variables, we can accomplish our goal of counting a diagnosis only once for a given patient. The logical variable FIRST.DX will be true each time a new ID-diagnosis combination is encountered. The data set and procedure would look as follows (assume we have previously sorted by ID and DX):

```
DATA DIAG;
   SET PATIENTS;
   BY ID DX;
   IF FIRST.DX;
RUN;

PROC FREQ DATA=DIAG ORDER=FREQ;
   TABLES DX;
RUN;
```

We have accomplished our goal of counting a diagnosis code only once per patient. As you can see, the SAS internal variables FIRST. and LAST. are extremely useful. Think of using them anytime you need to do something special for the first or last occurrence of another variable.

Problems

4-1. We have collected data on a questionnaire as follows:

Variable	Starting Column	Length	Description
ID	1	3	Subject ID
DOB	5	6	Date of Birth in MMDDYY format
ST_DATE	11	6	Start Date in MMDDYY format
END_DATE	17	6	Ending Date in MMDDYY format
SALES	23	5	Total Sales

Here is some sample data:

```
         1         2         3
123456789012345678901234567890  Column Indicators
------------------------------
001 1021461112801228888734 3
002 0913550202800204908812 3
005 0606400312810312855000 0
003 0705441115801113908954 4
```

(a) Write a SAS program to read these data.

(b) Compute age (in years) at time work was started and length of time between ST_DATE and END_DATE (also in years).

(c) Compute the sales per year of work.

(d) Print out a report showing:

ID DOB AGE LENGTH SALES_YR

where LENGTH is the time at work computed in (b) and SALES_YR is the sales per year computed in part (c). Use the MMDDYY8. format to print the DOB.

(e) Modify the program to compute AGE as of the last birthday and sales per year rounded to the nearest 10 dollars. Try using the DOLLAR6. format for SALES_YR.

4-2. For each of eight mice, the date of birth, date of disease, and date of death is recorded. In addition, the mice are placed into one of two groups (A or B). Given the data below, compute the time from birth to disease, the time from disease to death, and age at death. All times can be in days. Compute the mean, standard deviation, and standard error of these three times for each of the two groups. Here are the data:

RAT_NO	DOB	DISEASE	DEATH	GROUP
1	23MAY90	23JUN90	28JUN90	A
2	21MAY90	27JUN90	05JUL90	A
3	23MAY90	25JUN90	01JUL90	A
4	27MAY90	07JUL90	15JUL90	A
5	22MAY90	29JUN90	22JUL90	B
6	26MAY90	03JUL90	03AUG90	B
7	24MAY90	01JUL90	29JUL90	B
8	29MAY90	15JUL90	18AUG90	B

Arrange these data in any columns you wish. HINT: Use the DATE7. informat to read dates in this form.

***4-3.** Using the data set (PATIENTS) described in Section C of this chapter (use the program with sample data), write the necessary SAS statements to create a data set (PROB4_3) in which the first visit for each patient is omitted. Then, using that data set, compute the mean HR, SBP, and DBP for each patient. (Patient 9 with only one visit will be eliminated.)

***4-4.** Write a program similar to problem 4-3 except that we want to include all the data for each patient except for any patient that has had only one visit.

***4-5.** We have a data set called BLOOD that contains from one to five observations per subject. Each observation contains the variables ID, GROUP, TIME, WBC (white blood cells), and RBC (red blood cells). Run the program below to create this data set.

```
***Program to create data set BLOOD;
DATA BLOOD;
   LENGTH GROUP $ 1;
   INPUT ID GROUP $ TIME WBC RBC @@;
DATALINES;
1 A 1 8000 4.5 1 A 2 8200 4.8  1 A 3 8400 5.2
1 A 4 8300 5.3 1 A 5 8400 5.5
2 A 1 7800 4.9 2 A 2 7900 5.0
3 B 1 8200 5.4 3 B 2 8300 5.4  3 B 3 8300 5.2
3 B 4 8200 4.9 3 B 5 8300 5.0
4 B 1 8600 5.5
5 A 1 7900 5.2 5 A 2 8000 5.2  5 A 3 8200 5.4
5 A 4 8400 5.5
;
```

We want to create a data set that contains the mean WBC and RBC for each subject. This new data set should contain the variables ID, GROUP, WBC, and RBC where WBC and RBC are the mean values for the subject. Finally, we want to exclude any subjects from this data set that have two or fewer observations in the original data set (assume there are no missing values).

HINT: We will want to use PROC MEANS with a CLASS statement. Since we want both ID and GROUP in the new data set, you can make them both CLASS variables or include an ID statement (ID GROUP;) to cause the variable GROUP to be present in the output data set. Also, remember the _FREQ_ variable that PROC MEANS creates. It will be useful for creating a data set that meets the last condition of excluding subjects with two or fewer observations.

4-6. Modify the program in problem 4-5 to include the standard deviation of WBC and RBC for each subject.

CHAPTER

5

Correlation and Regression

A. Introduction

A common statistic for indicating the strength of a linear relationship existing between two continuous variables is called the Pearson correlation coefficient, or correlation coefficient (there are other types of correlation coefficients). The correlation coefficient is a number ranging from −1 to +1. A positive correlation means that as values of one variable increase, values of the other variable also tend to increase. A small or zero correlation coefficient tells us that the two variables are unrelated. Finally, a negative correlation coefficient shows an inverse relationship between the variables: as one goes up, the other goes down. Before we discuss correlation any further, let's begin with a simple example.

B. Correlation

We have recorded the GENDER, HEIGHT, WEIGHT, and AGE of seven people and want to compute the correlations between HEIGHT and WEIGHT, HEIGHT and AGE, and WEIGHT and AGE. The program to accomplish this is as follows:

```
DATA CORR_EG;
   INPUT GENDER $ HEIGHT WEIGHT AGE;
DATALINES;
M 68 155 23
F 61  99 20
F 63 115 21
M 70 205 45
M 69 170  .
F 65 125 30
M 72 220 48
;
PROC CORR DATA=CORR_EG;
   TITLE 'Example of a Correlation Matrix';
   VAR HEIGHT WEIGHT AGE;
RUN;
```

The output from this program is:

```
Example of a Correlation Matrix

Correlation Analysis

   3 'VAR' Variables: HEIGHT WEIGHT AGE


                    Simple Statistics

Variable   N     Mean   Std Dev      Sum   Minimum   Maximum

HEIGHT     7  66.8571    3.9761    468.0   61.0000   72.0000
WEIGHT     7    155.6   45.7961   1089.0   99.0000     220.0
AGE        6  31.1667   12.4164    187.0   20.0000   48.0000


Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0
/ Number of Observations

               HEIGHT        WEIGHT          AGE

HEIGHT        1.00000       0.97165      0.86614
                  0.0        0.0003       0.0257
                    7             7            6

WEIGHT        0.97165       1.00000      0.92496
               0.0003           0.0       0.0082
                    7             7            6

AGE           0.86614       0.92496      1.00000
               0.0257        0.0082          0.0
                    6             6            6
```


PROC CORR gives us some simple descriptive statistics on the variables in the VAR list along with a correlation matrix. If you examine the intersection of any row or column in this matrix, you will find the correlation coefficient (top number), the p-value (the second number), and the number of data pairs used in computing the correlation (third number). If the number of data pairs are the same for every combination of variables, the third number in each group is not printed. Instead, this number is printed at the top of the table.

In this listing, we see that the correlation between HEIGHT and WEIGHT is .97165, and the significance level is .0003; the correlation between HEIGHT and AGE is .86614 (p = .0257); the correlation between WEIGHT and AGE is .92496 (p = .0082). Let's concentrate for a moment on the HEIGHT and WEIGHT correlation. The small p-value computed for this correlation indicates that it is unlikely to have obtained a correlation this large, by chance, if the sample of seven subjects were taken from a population whose correlation was zero. Remember that this is just an example. Correlations this large are quite rare in social science data.

To generate a correlation matrix (correlations between every pairwise combination of the variables), use the following general syntax:

```
PROC CORR options;
   VAR list-of-variables;
RUN;
```

The term "list-of-variables" should be replaced with a list of variable names, separated by spaces. If no options are selected, a Pearson correlation coefficient is computed, and simple descriptive statistics are computed. As we discuss later, several nonparametric correlation coefficients are also available. The option SPEARMAN will produce Spearman rank correlations, KENDALL will produce Kendall's tau-b coefficients, and HOEFFDING will produce Hoeffding's D statistic. If you use one of these options, SPEARMAN for example, you must use the PEARSON option as well, if you also want Pearson correlations to be computed. If you do not want simple statistics, include the option NOSIMPLE in your list of options.

For example, if you wanted both Pearson and Spearman correlations and did not want simple statistics to be printed, you would submit the following code:

```
PROC CORR DATA=CORR_EG PEARSON SPEARMAN NOSIMPLE;
   TITLE 'Example of a Correlation Matrix';
   VAR HEIGHT WEIGHT AGE;
RUN;
```

The CORR procedure will compute correlation coefficients between all pairs of variables in the VAR list. If the list of variables is large, this results in a very large number of coefficients. If you only want to see a limited number of correlation coefficients (the ones with the highest absolute values), include the BEST=number option with PROC CORR. This results in these correlations listed in descending order.

If all you want is a number of correlations between one or more variables against another set of variables, a WITH statement is available. PROC CORR will then compute a correlation coefficient between every variable in the WITH list against every variable in the VAR list. For example, suppose we had the variables IQ and GPA (grade point average) in a data set called RESULTS. We also recorded a student score on 10 tests (TEST1-TEST10). If we only want to see the correlation between the IQ and GPA versus each of the 10 tests, the syntax is:

```
PROC CORR DATA=RESULTS;
   VAR IQ GPA;
   WITH TEST1-TEST10;
RUN;
```

This notation can save considerable computation time.

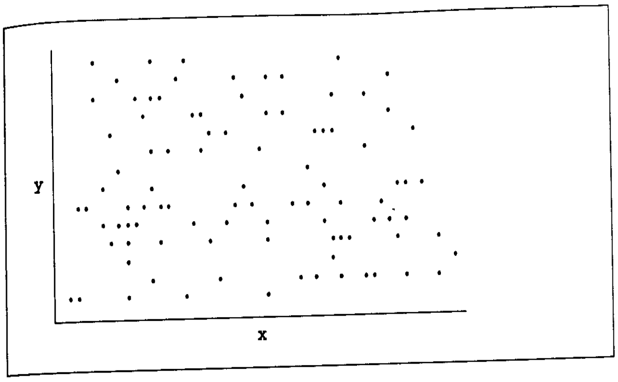
C. Significance of a Correlation Coefficient

You may ask, "How large a correlation coefficient do I need to show that two variables are correlated?" Each time PROC CORR prints a correlation coefficient, it also prints a probability associated with the coefficient. That number gives the probability of obtaining a sample correlation coefficient as large as or larger than the one obtained by chance alone (i.e., when the variables in question actually have zero correlation).

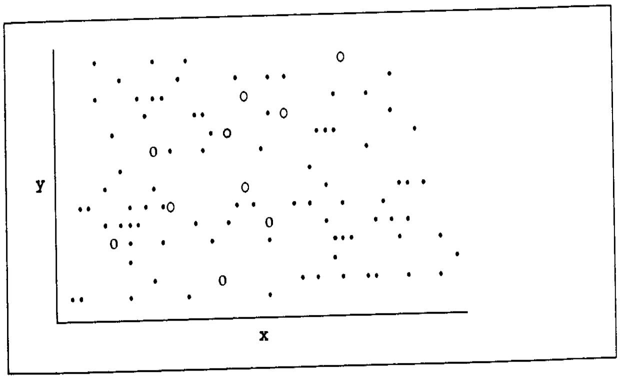
The significance of a correlation coefficient is a function of the magnitude of the correlation and the sample size. With a large number of data points, even a small correlation coefficient can be significant. For example, with 10 data points, a correlation coefficient of .63 or larger is significant (at the .05 level); with 100 data points, a correlation of .195 would be significant. Note that a negative correlation shows an equally strong relationship as a positive correlation (although the relationship is inverse). A correlation of −.40 is as strong as one of +.40.

It is important to remember that correlation indicates only the strength of a relationship—it does not imply causality. For example, we would probably find a high positive correlation between the number of hospitals in each of the 50 states versus the number of household pets in each state. Does this mean that pets make people sick and therefore make more hospitals necessary? Doubtful. The most plausible explanation is that both variables (number of pets and number of hospitals) are related to population size.

Being SIGNIFICANT is not the same as being IMPORTANT or STRONG. That is, knowing the significance of the correlation coefficient does not tell us very much. Once we know that our correlation coefficient is significantly different from zero, we need to look further in interpreting the importance of that correlation. Let us digress a moment to ask what we mean by the significance of a correlation coefficient. Suppose we have a population of x and y values in which the correlation is zero. We could imagine a plot of this population as shown in the graph below:



Suppose further that we choose a small sample of 10 points from this population. In the plot below, the o's represent the x, y pairs we choose "at random" from our population.



In this sample, a nonzero correlation would occur, but probably not large enough to be significant because the sample is too small. The significance level must be tested anew for each correlation.

D. How to Interpret a Correlation Coefficient

One of the best ways to interpret a correlation coefficient (r) is to look at the square of the coefficient (r-squared); r-squared can be interpreted as the proportion of variance in one of the variables that can be explained by variation in the other variable. As an example, our height/weight correlation is .97. Thus, r-squared is .94. We can say that 94% of the variation in weights can be explained by variation in height (or vice versa). Also, (1 − .94), or 6%, of the variance of weight, is due to factors other than height variation. With this interpretation in mind, if we examine a correlation coefficient of .4, we see that only .16, or 16%, of the variance of one variable is explained by variation in the other.

Another consideration in the interpretation of correlation coefficients is this: Be sure to look at a scatter plot of the data (using PROC PLOT). It often turns out that one or two extreme data points can cause the correlation coefficient to be much larger than expected. One keypunching error can dramatically alter a correlation coefficient.

An important assumption concerning a correlation coefficient is that each pair of x,y data points is independent of any other pair. That is, each pair of points has to come from a separate subject. Suppose we had 20 subjects, and we measured two variables (say blood pressure and heart rate) for each of the 20 subjects. We could then calculate a correlation coefficient using 20 pairs of data points. Suppose instead that we made 10 measurements of blood pressure and heart rate for each subject. We cannot compute a valid correlation coefficient using all 200 data points (10 points for each of 20 subjects) because the 10 points for each subject are not independent. The best method would be to compute the mean blood pressure and heart rate for each subject, and use the mean values to compute the correlation coefficient. Having done so, we'll keep in mind that we correlated mean values when we are interpreting the results.

E. Partial Correlations

A researcher may wish to determine the strength of the relationship between two variables when the effect of other variables has been removed. One way to accomplish this is by computing a partial correlation. To remove the effect of one or more variables from a correlation, use a PARTIAL statement to list those variables whose effects you want to remove. Using the CORR_EG data set from Section B, we can compute a partial correlation between HEIGHT and WEIGHT with the effect of AGE removed as:

```
PROC CORR DATA=CORR_EG NOSIMPLE;
   TITLE 'Example of a Partial Correlation';
   VAR HEIGHT WEIGHT;
   PARTIAL AGE;
RUN;
```

As you can see in the listing below, the partial correlation between HEIGHT and WEIGHT is now lower than before (.91934) although still significant (p = .0272).

```
Example of a Partial Correlation

Correlation Analysis

   1 'PARTIAL' Variables:  AGE
   2 'VAR'     Variables:  HEIGHT    WEIGHT
```



```
[Continued]

Pearson Partial Correlation Coefficients
/ Prob > |R| under Ho: Partial Rho=0 / N = 6

                  HEIGHT        WEIGHT

HEIGHT           1.00000       0.91934
                  0.0           0.0272

WEIGHT           0.91934       1.00000
                  0.0272        0.0
```

F. Linear Regression

Given a person's height, what would be their predicted weight? How can we best define the relationship between height and weight? By studying the graph below we see that the relationship is approximately linear. That is, we can imagine drawing a straight line on the graph with most of the data points being only a short distance from the line. The vertical distance from each data point to this line is called a residual.

```
Plot of WEIGHT*HEIGHT    Legend: A = 1 obs, B = 2 obs, etc.
Plot of PRED_WT*HEIGHT   Symbol used is P

WEIGHT
     |
 240 +
 220 +                                                      A
 200 +                                              A
 180 +
     |                                          A
 160 +
     |                                     A
 140 +
 120 +                         A
     |               A
 100 +    A
  80 +
  60 +
     |
     ---+----+----+----+----+----+----+----+----+----+----+----+
       61   62   63   64   65   66   67   68   69   70   71   72
                                 HEIGHT
```

How do we determine the "best" straight line to fit our height/weight data? The method of least squares is commonly used, which finds the line (called the regression line) that minimizes the sum of the squared residuals. An equivalent way to define a residual is the difference between a subject's predicted score and his/her actual score.

PROC REG (short for regression) has the general form:

```
PROC REG options;
   MODEL dependent variable(s) = independent variable / options;
RUN;
```

Using our height/weight program, we add the following PROC to give us the equation for the regression line:

```
PROC REG DATA=CORR_EG;
   TITLE 'Regression Line for Height-Weight Data';
   MODEL WEIGHT=HEIGHT;
RUN;
```

The output from the procedure above is as follows:

```
Model: MODEL1
Dependent Variable: WEIGHT

Analysis of Variance

                        Sum of          Mean
Source          DF     Squares        Square      F Value       Prob>F

Model            1  11880.32724   11880.32724      84.451       0.0003
Error            5    703.38705     140.67741
C Total          6  12583.71429

     Root MSE       11.86075     R-square        0.9441
     Dep Mean      155.57143     Adj R-sq        0.9329
     C.V.            7.62399

Parameter Estimates

                  Parameter         Standard      T for H0:
Variable  DF       Estimate            Error    Parameter=0    Prob > |T|

INTERCEP   1    -592.644578      81.54217462         -7.268        0.0008
HEIGHT     1      11.191265       1.21780334          9.190        0.0003
```


Look first at the last two lines. There is an estimate for two parameters, INTERCEPT and HEIGHT. The general equation for a straight line can be written as:

$$y = a + bx$$

where a = intercept, b = slope.

We can write the equation for the "best" straight line defined by our height/weight data as:

$$\text{WEIGHT} = -592.64 + 11.19 \times \text{HEIGHT}$$

Given any height, we can now predict the weight. For example, the predicted weight of a 70-inch-tall person is:

$$\text{WEIGHT} = -592.64 + 11.19 \times 70 = 190.66 \text{ lb.}$$

To the right of the parameter estimates, are columns labeled Standard Error, T for H_0: Parameter = 0, and Prob > |T|. The T values and the associated probabilities (Prob > |T|) test the hypothesis that the parameter is actually zero. That is, if the true slope and intercept were zero, what would the probability be of obtaining, by chance alone, a value as large as or larger than the one actually obtained? The standard error can be thought of in much the same way as the standard error of the mean. It reflects the accuracy with which we know the true slope and intercept.

In our case, the slope is 11.19, and the standard error of the slope is 1.22. We can therefore form a 95% confidence interval for the slope by taking two (approximately) standard errors above and below the mean. The 95% confidence interval for our slope is 8.75 to 13.63. Actually, since the number of points in our example is small ($n = 7$) we really should go to a t-table to find the number of standard errors above and below the mean for a 95% confidence interval. (This is true when n is less than 30.) Going to a t-table, we look under degrees of freedom (df) equal to $n - 2$ and level of significance (two-tail) equal to .05. The value of t for df = 5 and $p = .05$ is 2.57. Our 95% confidence interval is then 11.19 plus or minus $2.57 \times 1.22 = 3.14$.

Inspecting the indented portion of the output, we see values for Root MSE, R-square, Dep Mean, Adj R-sq, and C.V. Root MSE is the square root of the error variance. That is, it is the standard deviation of the residuals. R-square is the square of the multiple correlation coefficient. Since we have only one independent variable (HEIGHT), R-square is the square of the Pearson correlation coefficient between HEIGHT and WEIGHT. As mentioned in the previous section, the square of the correlation coefficient tells us how much variation in the dependent variable can be accounted for by variation of the independent variable. When there is more than one independent variable, R-square will reflect the variation in the dependent variable accounted for by a linear combination of all the independent variables (see Chapter 9 for a more complete discussion). The Dep Mean is the mean of the dependent variable: WEIGHT in this case. C.V. is the coefficient of variation. Finally, Adj R-sq is the squared correlation coefficient corrected for the number of independent variables in the equation. This adjustment has the effect of decreasing the value of R-squared. The difference is typically small but becomes larger and more important when dealing with multiple independent variables (see Chapter 9).

G. Partitioning the Total Sum of Squares

The top portion of the printout of page 122 presents what is called the analysis of variance table for the regression. It takes the variation in the dependent variable and breaks it out into various sources. To understand this table, think about an individual weight. That weight can be thought of as the mean weight for all individuals plus (or minus) a certain amount because the individual is taller (or shorter) than average. The regression tells us that taller people are heavier. Finally, there is a portion attributable to the fact that the prediction is less than perfect. So we start at the mean, move up or down due to the regression, and then up or down again due to error (or residual). The analysis of variance table breaks these components apart and looks at the contribution of each through the sum of squares.

The total sum of squares is the sum of squared deviations of each person's weight from the grand mean. This total sum of squares (SS) can be divided into the two portions: the sum of squares due to regression (or model), and the sum of squares error (sometimes called residual). One portion, called the Sum of Squares (ERROR) in the output, reflects the deviation of each weight from the PREDICTED weight. The other portion reflects deviations between the PREDICTED values and the MEAN. This is called the Sum of Squares due to the MODEL in the output. The column labeled Mean Square is the Sum of Squares divided by the degrees of freedom. In our case, there are seven data points. The total degrees of freedom is equal to $n - 1$, or 6. The model here has 1 df. The error degrees of freedom is the total df minus the model df, which gives us 5. We can think of the Mean Square as the respective variance (square of the standard deviation) of each of these two portions of variation. Our intuition tells us that if the deviations about the regression line are small (error mean square) compared to the deviation between the predicted values and the mean (mean square model) then we have a good regression line. To compare these mean squares, the ratio

$$F = \frac{\text{Mean square model}}{\text{Mean square error}}$$

is formed. The larger this ratio, the better the fit (for a given number of data points). The program prints this F value and the probability of obtaining an F this large or larger by chance alone. In the case where there is only one independent variable, the probability of the F statistic is exactly the same as the probability associated with testing for the significance of the correlation coefficient. If this probability is "large" (over .05) then our linear model is not doing a good job of describing the relationship between the variables.

H. Plotting the Points on the Regression Line

To plot out height/weight data, we can use PROC PLOT as follows:

```
PROC PLOT DATA=CORR_EG;
   PLOT WEIGHT*HEIGHT;
RUN;
```

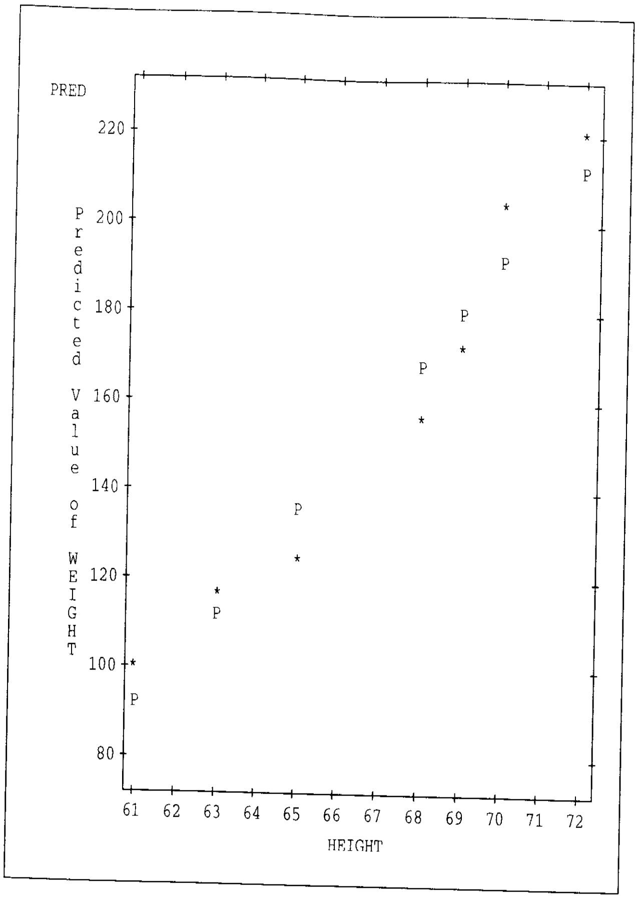
However, we would have to draw in the regression line by hand, using the slope and intercept from PROC REG. It is desirable to have SAS software plot the regression line (actually, the regression predicted values) for us. PROC REG allows the use of a PLOT statement. The form is:

```
PLOT y_variable * x_variable = symbol / options;
```

Y_variable and x_variable are the names of the variables to be plotted on the y- and x-axes, respectively. The symbol can be either a single character in single quotes or a variable name whose value is to be the plotting symbol (as in PROC PLOT). There are some special "variable names" that can be used with a PLOT statement. In particular, the names PREDICTED. and RESIDUAL. (the periods are part of these keywords) are used to plot predicted and residual values. A common option is OVERLAY, which is used to plot more than one graph on a single set of axes.

Suppose we want to see a plot of WEIGHT versus HEIGHT in the PROC REG example above. In addition, we want to see the points on the regression line. We want the plotting symbols for the WEIGHT versus HEIGHT graph to be asterisks (*) and the symbols showing the predicted values (the regression line) to be the letter 'P'. The procedure, complete with the PLOT statement is shown next, followed by the plot:

```
PROC REG DATA = CORR_EG;
   MODEL WEIGHT = HEIGHT;
   PLOT PREDICTED. *HEIGHT = 'P' WEIGHT*HEIGHT='*' / OVERLAY;
RUN;
```



I. Plotting Residuals and Confidence Limits

Before leaving this section, we mention that SAS software has the ability to PLOT several computed values other than the PREDICTED value of the dependent variable. The most useful statistics besides the predicted values are:

RESIDUAL.	The residual (i.e., difference between actual and predicted values for each observation).
L95. U95.	The lower or upper 95% confidence limit for an individual predicted value. (i.e., 95% of the dependent data values would be expected to be between these two limits).
L95M. U95M.	The lower or upper 95% confidence limit for the mean of the dependent variable for a given value of the independent variable.

To demonstrate some of these options in action, we produce the following:

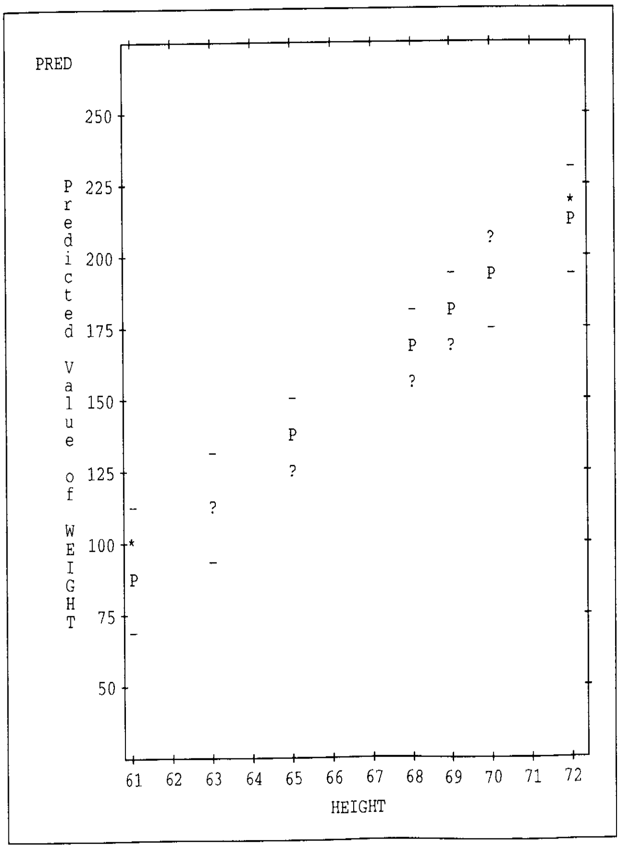
(a) A plot of the original data, predicted values, and the upper and lower 95% confidence limits for the mean weight.

(b) A plot of residuals versus HEIGHT. NOTE: If this plot shows some systematic pattern (rather than random points), one could try to improve the original model.

Here is the program to accomplish the requests above:

```
PROC REG DATA = CORR_EG;
   MODEL WEIGHT = HEIGHT;
   PLOT PREDICTED.*HEIGHT = 'P'
        U95M.*HEIGHT = '-' L95M.*HEIGHT='-'
        WEIGHT*HEIGHT='*' / OVERLAY;
   PLOT RESIDUAL.*HEIGHT = 'o';
RUN;
```

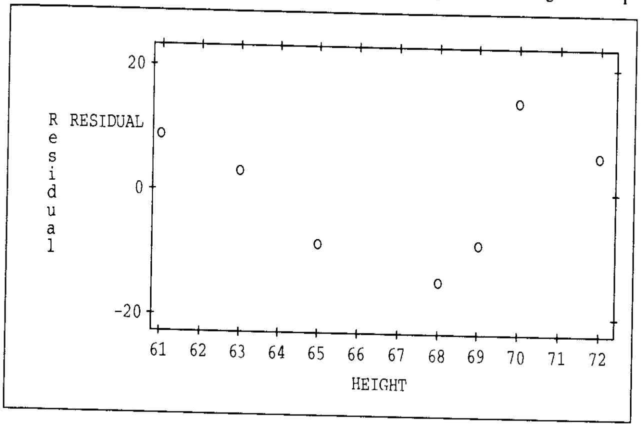
The two graphs are shown below:



In this graph, the *'s are the original data points, the P's represent the predicted values, the -'s the upper and lower 95% confidence intervals about the mean weight for a given height. The question marks represent multiple observations (perhaps a raw data value and a predicted value were very close together). The next graph is discussed in the following section.

J. Adding a Quadratic Term to the Regression Equation

The plot of residuals, shown in the graph below, suggests that a second-order term (height squared) might improve the model since the points do not seem random but, rather, form a curve that could be fit by a second-order equation. Although this chapter deals mostly with linear regression, let us show you quickly how you might explore this possible quadratic relationship between height and weight. First, we need to add a line in the original DATA step to compute a variable that represents the height squared. After the INPUT statement, include a line such as the following:




```
HEIGHT2 = HEIGHT * HEIGHT;
```

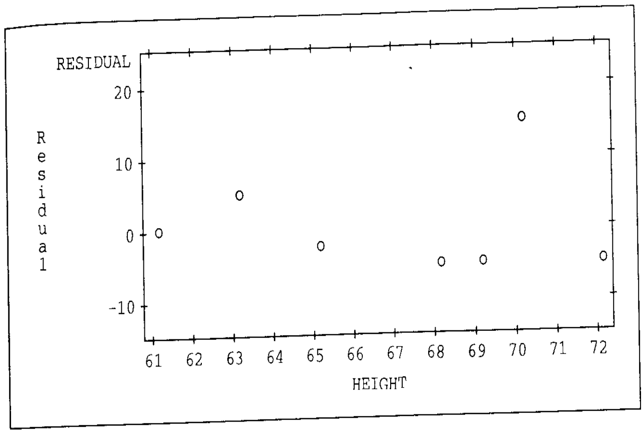
or

```
HEIGHT2 = HEIGHT**2;
```

Next, write your MODEL and PLOT statements:

```
PROC REG DATA=CORR_EG;
   MODEL WEIGHT = HEIGHT HEIGHT2;
   PLOT RESIDUAL.*HEIGHT = 'o';
RUN;
```


When you run this model, you will get an r-squared of .9743, an improvement over the .9441 obtained with the linear model. Studying the residual plot below shows that the distribution of the residuals is more random than the earlier plot. A caution or two is in order here. First, remember that this is an example with a very small data set and that the original correlation is quite high. Second, although it is possible to also enter cubic terms, etc., one should keep in mind that results need to be interpretable.



K. Transforming Data

Another regression example is provided here to demonstrate some additional steps that may be necessary when doing regression. Shown below are data collected from 10 people:

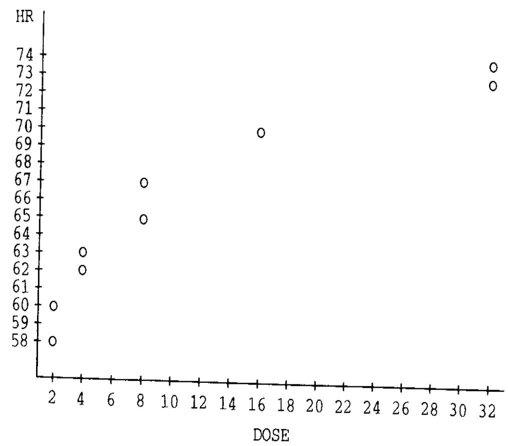
Subject	Drug Dose	Heart Rate
1	2	60
2	2	58
3	4	63
4	4	62
5	8	67
6	8	65
7	16	70
8	16	70
9	32	74
10	32	73

Let's write a SAS program to define this collection of data and plot drug dose by heart rate.

```
DATA HEART;
   INPUT DOSE HR;
DATALINES;
2  60
   .
   .
   .
32 73
;
PROC PLOT DATA=HEART;
   PLOT HR*DOSE='o';
RUN;
PROC REG DATA=HEART;
   MODEL HR = DOSE;
RUN;
```

The resulting graph and the PROC REG output are shown below:

```
Plot of HR*DOSE.  Symbol used is 'o'.
```




```
NOTE: 1 obs hidden.
Regression Line for Height-Weight Data

Model: MODEL1
Dependent Variable: HR
```


[Continued]

Analysis of Variance

Source	DF	Sum of Squares	Mean Square	F Value	Prob>F
Model	1	233.48441	233.48441	49.006	0.0001
Error	8	38.11559	4.76445		
C Total	9	271.60000			

Root MSE	2.18276	R-square	0.8597
Dep Mean	66.20000	Adj R-sq	0.8421
C.V.	3.29722		

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	1	60.708333	1.04491764	58.099	0.0001
DOSE	1	0.442876	0.06326447	7.000	0.0001

Either by clinical judgment or by careful inspection of the graph, we decide that the relationship is not linear. We see an approximately equal increase in heart rate each time the dose is doubled. Therefore, if we plot log dose against heart rate we can expect a linear relationship. SAS software has a number of built-in functions such as logarithms and trigonometric functions, described in Chapter 17. We can write mathematical equations to define new variables by placing these statements between the INPUT and DATALINES statements. In SAS programs, we represent addition, subtraction, multiplication, and division by the symbols +, − , *, and /, respectively. Exponentiation is written as **. To create a new variable which is the log of dose, we write:

```
DATA HEART;
   INPUT DOSE HR;
   LDOSE = LOG(DOSE);
DATALINES;
```

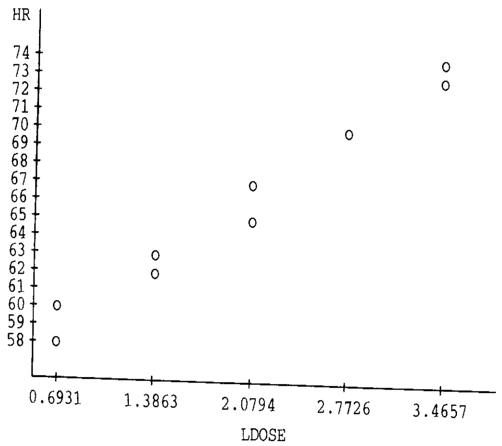
LOG is a SAS function that yields the natural (base e) logarithm of whatever value is within the parentheses.

We can now plot log dose versus heart rate and compute a new regression line.

```
PROC PLOT DATA=HEART;
   PLOT HR*LDOSE='o';
RUN;

PROC REG DATA=HEART;
   MODEL HR=LDOSE;
RUN;
```

Output from the statements above is shown on the following pages. Approach transforming variables with caution. Keep in mind that when a variable is transformed, one should not refer to the variable as in the untransformed state. That is, don't refer to the "log of dosage" as "dosage." Some variables are frequently transformed: Income, sizes of groups, and magnitudes of earthquakes are usually presented as logs, or in some other transformation.

```
Plot of HR*LDOSE. Symbol used is 'o'.
```




```
NOTE: 1 obs hidden.
Regression Line for Height-Weight Data

Model: MODEL1
Dependent Variable: HR
```


[Continued]

```
Analysis of Variance

                       Sum of         Mean
Source        DF      Squares       Square     F Value    Prob>F

Model          1    266.45000    266.45000     413.903    0.0001

Error          8      5.15000      0.64375
C Total        9    271.60000

     Root MSE       0.80234     R-square       0.9810
     Dep Mean      66.20000     Adj R-sq       0.9787
     C.V.           1.21199


Parameter Estimates

                  Parameter     Standard      T for H0:
Variable   DF      Estimate        Error    Parameter=0    Prob > |T|

INTERCEP    1     55.250000   0.59503151         92.852        0.0001
LDOSE       1      5.265837   0.25883212         20.345        0.0001
```

Notice that the data points are now closer to the regression line. The MEAN SQUARE ERROR term is smaller and r-square is larger, confirming our conclusion that dose versus heart rate fits a logarithmic curve better than a linear one.

L. Computing Within-subject Slopes

There are times when we need to compute slopes for each subject (on multiple measurements) or slopes for a subset of an entire population. An example would be a study where each subject is assigned to group A or group B. We then measure his or her performance on a test three times, and the rate of learning (the slope of the score versus time curve) is compared for the two groups. Prior to version 6, this was a difficult problem to solve. Thanks to the response of the SAS Institute to user requests, we can now output a data set containing slopes with PROC REG.

The procedure option, OUTEST=data-set-name, will create a data set containing all variables in the BY statement as well as the slopes (coefficients) of each of the independent variables in the model. To illustrate, here is a program to analyze the study design just described:

```
DATA TEST;
   INPUT ID GROUP $ TIME SCORE;
DATALINES;
1 A 1 2
1 A 2 5
1 A 3 7
2 A 1 4
2 A 2 6
2 A 3 9
3 B 1 8
3 B 2 6
3 B 3 2
4 B 1 8
4 B 2 7
4 B 3 3
;
PROC SORT DATA=TEST;
   BY ID;
RUN;

PROC REG OUTEST=SLOPES DATA=TEST;
   BY ID;
   ID GROUP;  ***Note: This ID statement is used so that the
               GROUP variable will be in the SLOPES data
               set. An alternative would be to include it
               in the BY list.;
   MODEL SCORE = TIME / NOPRINT;
RUN;

PROC PRINT DATA=SLOPES;
   TITLE 'Listing of the Data Set SLOPES';
RUN;

PROC TTEST DATA=SLOPES;
   TITLE 'Comparing Slopes Between Groups';
   CLASS GROUP;
   VAR TIME;
RUN;
```

The data set SLOPES, produced by PROC REG, and the results from the t-test are shown next. (We are getting a bit ahead of ourseleves here. You can read about how to compare means with PROC TTEST in the next chapter.) The variable TIME represents the slope of the SCORE by TIME graph (i.e., the coefficient of TIME). The reason that the GROUP variable is included in the ID statement is so that it will be included in the output data set. One could also include other variables where each subject can have only one value (such as GENDER). Since slope in this example is a computation over observations for an individual, any other variable which


```
Listing of the Data Set SLOPES

OBS ID GROUP _MODEL_ _TYPE_ _DEPVAR_  _RMSE_  INTERCEP TIME SCORE

 1   1   A   MODEL1  PARMS   SCORE   0.40825  -0.3333  2.5   -1
 2   2   A   MODEL1  PARMS   SCORE   0.40825   1.3333  2.5   -1
 3   3   B   MODEL1  PARMS   SCORE   0.81650  11.3333 -3.0   -1
 4   4   B   MODEL1  PARMS   SCORE   1.22474  11.0000 -2.5   -1


Comparing Slopes Between Groups

TTEST PROCEDURE

Variable: TIME

GROUP        N              Mean            Std Dev          Std Error
----------------------------------------------------------------------
A            2        2.50000000        0.00000E+00        0.00000E+00
B            2       -2.75000000        3.53553E-01        2.50000E-01

Variances        T       DF    Prob>|T|
---------------------------------------
Unequal    21.0000      1.0      0.0303
Equal      21.0000      2.0      0.0023

NOTE: All values are the same for one CLASS level.
```

could have different values for an individual cannot be used. The dependent variable in each model is assigned a value of −1.

Problems

5-1. Given the following data:

X	Y	Z
1	3	15
7	13	7
8	12	5
3	4	14
4	7	10

(a) Write a SAS program and compute the Pearson correlation coefficient between X and Y; X and Z. What is the significance of each?

(b) Change the correlation request to produce a correlation matrix, that is, the correlation coefficient between each variable versus every other variable.

5-2. Given the following data:

AGE	SBP
15	116
20	120
25	130
30	132
40	150
50	148

How much of the variance of SBP (systolic blood pressure) can be explained by the fact that there is variability in AGE? (Use SAS to compute the correlation between SBP and AGE.)

5-3. From the data for X and Y in problem 5-1:

(a) Compute a regression line (Y on X). Y is the dependent variable, X the independent variable.

(b) What is slope and intercept?

(c) Are they significantly different from zero?

5-4. Using the data of problem 5-1, compute three new variables LX, LY, and LZ which are the natural logs of the original values. Compute a correlation matrix for the three new variables. HINT: The function to compute a natural log is the LOG function (see Chapter 17 for details).

5-5. Generate:

(a) A plot of Y versus X (data from problem 5-1).

(b) A plot of the regression line and the original data on the same set of axes.

5-6. Given the data set:

COUNTY	POP	HOSPITAL	FIRE_CO	RURAL
1	35	1	2	YES
2	88	5	8	NO
3	5	0	1	YES
4	55	3	3	YES
5	75	4	5	NO
6	125	5	8	NO
7	225	7	9	YES
8	500	10	11	NO

(a) Write a SAS program to create a SAS data set of the data above.

(b) Run PROC UNIVARIATE to check the distributions for the variables POP, HOSPITAL, and FIRE_CO.

(c) Compute a correlation matrix for the variables POP, HOSPITAL, and FIRE_CO. Produce both Pearson and Spearman correlations. Which is more appropriate?

(d) Recode POP, HOSPITAL, and FIRE_CO so that they each have two levels (use a median cut or a value somewhere near the 50th percentile). Compute crosstabulations between the variable RURAL and the recoded variables.

5-7. What's wrong with the following program? (NOTE: There may be missing values for X, Y, and Z.)

```
DATA MANY-ERR;
   INPUT X Y Z;
   IF X LE 0 THEN X=1;
   IF Y LE 0 THEN Y=1;
   IF Z LE 0 THEN Z=1;
   LOGX=LOG(X);
   LOGY=LOG(Y);
   LOGZ=LOG(Z);
DATALINES;
1 2 3
. 7 8
4 . 10
7 8 11
;
PROC CORR DATA=MANY-ERR / PEARSON SPEARMAN;
   VAR X - LOGZ;
RUN;
```

CHAPTER

6

T-tests and Nonparametric Comparisons

A. *Introduction*

Our first topic concerning hypothesis testing is the comparison of two groups. When certain assumptions are met, the popular t-test is used to compare means. When these assumptions are not met, there are several nonparametric methods that can be used. This chapter shows you how to conduct all of these two group comparisons using SAS software. In addition, we show you how to write a simple SAS program to randomly assign subjects to two or more groups.

B. *T-test: Testing Differences between Two Means*

A common experimental design is to assign subjects randomly to a treatment or a control group and then measure one or more variables that would be hypothesized to be affected by the treatment. To determine whether the means of the treatment and control groups are significantly different, we set up what is called a null hypothesis (H_0). It states that the treatment and control groups would have the same mean if we repeated the experiment a large (infinite) number of times and that the differences in any one trial are attributable to the "luck of the draw" in assigning subjects to treatment and control groups. The alternative hypotheses (H_A) to the null hypothesis are that one particular mean will be greater than the other (called a one-tailed test) or that the two means will be different, but the researcher cannot say a priori which will be greater (called a two-tailed test). The researcher can specify either of the two alternative hypotheses.

When the data have been collected, a procedure called the t-test is used to determine the probability that the difference in the means that is observed is due to chance. The lower the likelihood that the difference is due to chance, the greater the likelihood that the difference is due to there being real differences in treatment and control. The following example demonstrates a SAS program that performs a t-test.

Students are randomly assigned to a control or treatment group (where a drug is administered). Their response times to a stimulus is then measured. The times are as follows:

Control	Treatment
(response time in millisec)	
80	100
93	103
83	104
89	99
98	102

Do the treatment scores come from a population whose mean is different from the mean of the population from which the control scores were drawn? A quick calculation shows that the mean of the control group is 88.6, and the mean of the treatment group is 101.6. You may recall the discussion in Chapter 2 about the standard error of the mean. When we use a sample to estimate the population mean, the standard error of the mean reflects how accurately we can estimate the population mean. As you will see when we write a SAS program to analyze this experiment, the standard error of the control mean is 3.26, and the standard error of the treatment mean is .93. Since the means are 11.8 units apart, even if each mean is several standard errors away from its true population mean, they would still be significantly different from each other.

There are some assumptions that should be met before we can apply this test. First, the two groups must be independent. This is ensured by our method of random assignment. Second, the theoretical distribution of sampling means should be normally distributed (this is ensured if the sample size is sufficiently large), and third, the variances of the two groups should be approximately equal. This last assumption is checked automatically each time a t-test is computed by the SAS system. The SAS t-test output contains t-values and probabilities for both the case of equal group variances and unequal group variances. We will see later how to test whether the data are normally distributed. Finally, when the t-test assumptions are not met, there are other procedures that can be used; these are demonstrated later in this chapter.

It's time now to write a program to describe our data and to request a t-test. We know which group each person belongs to and what his or her response time is. The program can thus be written as follows:

```
DATA RESPONSE;
   INPUT GROUP $ TIME;
DATALINES;
C 80
C 93
```

[Continued]

```
C 83
C 89
C 98
T 100
T 103
T 104
T 99
T 102
;
PROC TTEST DATA=RESPONSE;
   TITLE 'T-test Example';
   CLASS GROUP;
   VAR TIME;
RUN;
```

PROC TTEST uses a CLASS statement to identify the independent variable—the variable that identifies the two groups of subjects. In our case, the variable GROUP has values of C (for the control group) and T (for the treatment group).

The variable or variable list that follows the word VAR identifies the dependent variable(s), in our case, TIME. When more than one dependent variable is listed, a separate t-test is computed for each dependent variable in the list.

Look at the TTEST output below:

```
T-test Example

TTEST PROCEDURE

Variable: TIME

GROUP      N              Mean            Std Dev           Std Error
-----------------------------------------------------------------------
C          5       88.60000000         7.30068490          3.26496554
T          5      101.60000000         2.07364414          0.92736185

Variances        T         DF       Prob>|T|
---------------------------------------------
Unequal    -3.8302        4.6         0.0145
Equal      -3.8302        8.0         0.0050

For H0: Variances are equal, F' = 12.40 DF = (4,4) Prob>F' = 0.0318
```

We see the mean value of TIME for the control and treatment groups, along with the standard deviation and standard error. Below this are two sets of t-values, degrees of freedom, and probabilities. One is valid if we have equal variances, the other if we have unequal variances. You will usually find these values very close unless the variances differ widely. The bottom line gives us the probability that the

variances are unequal due to chance. If this probability is small (say less than .05) then we are going to reject the hypothesis that the variances are equal. We then use the t-value and probability labeled Unequal. If the PROB>F' value is greater than .05, we use the t-value and probability for equal variances.

In this example, we look at the end, the t-test output, and see that the F ratio (larger variance divided by the smaller variance) is 12.40. The probability of obtaining, by chance alone, a ratio this large or larger is 0.0318. That is, if the two samples came from populations with equal variance, there is a small probability (.0318) of obtaining a ratio of our sample variances of 12.40 or larger by chance. We should therefore use the t-value appropriate for groups with unequal variance. The rule of thumb here is to use the t-value (df) and probability labeled unequal if the probability from the F-test is less then .05. We point out here that, as with most rules of thumb, there is a weakness in the rule. That is, when the sample sizes are small, the test of equal variances has low power and may fail to reject the null hypothesis of equal variances. This is just the situation where the assumption is most important to performing a proper t-test. Also, when the sample sizes are large (where the assumptions of equal variance are less important to the t-test), the null hypothesis of equal variances is frequently rejected.

C. *Random Assignment of Subjects*

In our discussion of t-tests, we said that we randomly assigned subjects to either a treatment or control group. This is actually a very important step in our experiment and we can use SAS software to provide us with a method for making the assignments.

We could take our volunteers one by one, flip a coin, and decide to place all the "heads" in our treatment group and the "tails" in our control group. This is acceptable, but we would prefer a method that ensures an equal number of subjects in each group. One method would be to place all the subjects' names in a hat, mix them up, and pull out half the names for treatment subjects and the others for controls. This is essentially what we will do in our SAS program. The key to the program is the SAS random number function.

The function RANUNI(seed) will generate a pseudorandom number in the interval from 0 to 1. The argument of this function, called a seed, can either be a zero or a number of your choice. A zero seed specifies that the random number function use the time clock to generate a random seed to initiate the random number sequence. If you use a zero seed, you will obtain a different series of random numbers every time the program is run (unless you run the program at EXACTLY the same time every day). You may want to supply your own seed instead. It can be any number you wish. Each time you run the program with your own seed, you will generate the same series of random numbers, which is sometimes desirable. You will have to decide whether to supply your own seed or use the time clock.

We will create a SAS data set with our subjects' names (or we could use subject numbers instead) and assign a random number to each subject. We can then split the group in half (or in any number of subgroups) using a special feature of PROC RANK. Here is the program:

```
PROC FORMAT;
    VALUE GRPFMT 0='CONTROL' 1='TREATMENT';
RUN;

DATA RANDOM;
   INPUT SUBJ NAME $20.;
   GROUP=RANUNI(0);
DATALINES;
1 CODY
2 SMITH
3 HELM
4 GREGORY
   (more data lines)
;
PROC RANK DATA=RANDOM GROUPS=2 OUT=SPLIT;
   VAR GROUP;
RUN;

PROC SORT DATA=SPLIT;
   BY NAME;
RUN;

PROC PRINT DATA=SPLIT;
   TITLE 'Subject Group Assignments';
   ID NAME;
   VAR SUBJ GROUP;
   FORMAT GROUP GRPFMT.;
RUN;
```

The key to this program is the RANUNI function, which assigns a random number from 0 to 1 to each subject. The GROUPS=2 option of PROC RANK divides the subjects into two groups (0 and 1) depending on the value of the random variable GROUP. Values below the median become 0; those at or above the median become 1. The GRPFMT format assigns the labels "CONTROL" and "TREATMENT" using values of 0 and 1, respectively. We can use this program to assign our subjects to any number of groups by changing the "GROUPS=" option of PROC RANK to indicate the desired number of groups. Sample output from this program is shown below:

```
Subject Group Assignments

NAME        SUBJ      GROUP

CODY          1      CONTROL
GREGORY       4      CONTROL
HELM          3      TREATMENT
SMITH         2      TREATMENT
  (more names and groups)
```


Although this may seem like a lot of work just to make random assignments of subjects, we recommend that this or an equivalent procedure be used for assigning subjects to groups. Other methods such as assigning every other person to the treatment group can result in unsuspected bias.

D. *Two Independent Samples: Distribution Free Tests*

There are times when the assumptions for using a t-test are not met. One common problem is that the data are not normally distributed, and your sample size is small. For example, suppose we collected the following numbers in a psychology experiment that measured the response to a stimulus:

```
0 6 0 5 7 6 9 4 8 0 7 0 5 6 6 0 0
```

A frequency distribution would look like this:

```
x
x
x     x
x     x
x    xxx
x   xxxxxx
---------------
0123456789
```

What we are seeing is probably due to a threshold effect. The response is either 0 (the stimulus is not detected) or, once the stimulus is detected, the average response is about 6. Data of this sort would artificially inflate the standard deviation (and thus the standard error) of the sample and make the t-test more conservative. However, we would be safer to use a nonparametric test (a test that does not assume a normal distribution of data).

Another common problem is that the data values may only represent ordered categories. Scales such as 1 = very mild, 2 = mild, 3 = moderate, 4 = strong, 5 = severe reflect the strength of a response, but we cannot say that a score of 4 (strong) is worth twice the score of 2 (mild). Scales like these are referred to as ordinal scales. (Most of the scales we have been using until now have been interval or ratio scales.) We need a nonparametric test to analyze differences in central tendencies for ordinal data. Finally, for very small samples, nonparametric tests are often more appropriate since assumptions concerning distributions are difficult to determine.

SAS software provides us with several nonparametric two-sample tests. Among these are the Wilcoxon rank-sum test (equivalent to the Mann-Whitney U-test) for two samples.

Consider the following experiment. We have two groups, A and B. Group B has been treated with a drug to prevent tumor formation. Both groups are exposed to a chemical that encourages tumor growth. The masses (in grams) of tumors in groups A and B are

```
A:  3.1 2.2 1.7 2.7 2.5
B:  0.0 0.0 1.0 2.3
```

Are there any differences in tumor mass between groups A and B? We will choose a nonparametric test for this experiment because of the absence of a normal distribution and the small sample sizes involved. The Wilcoxon test first puts all the data (groups A and B) in increasing order (with special provisions for ties), retaining the group identity. In our experiment we have

```
MASS    0.0  0.0  1.0  1.7  2.2  2.3  2.5  2.7  3.1
GROUP    B    B    B    A    A    B    A    A    A
RANK    1.5  1.5   3    4    5    6    7    8    9
```

The sums of ranks for the A's and B's are then computed. We have

SUM RANKS A = 4+5+7+8+9 = 33
SUM RANKS B = 1.5+1.5+3+6 = 12

If there were smaller tumors in group B, we would expect the B's to be at the lower end of the rank ordering and therefore have a smaller sum of ranks than the A's. Is the sum of ranks for group A sufficiently larger than the sum of ranks for group B so that the probability of the difference occurring by chance alone is small (less than .05)? The Wilcoxon test gives us the probability that the difference in rank sums that we obtained occurred by chance.

For even moderate sample sizes, the Wilcoxon test is almost as powerful as its parametric equivalent, the t-test. Thus, if there is a question concerning distributions or if the data are really ordinal, you should not hesitate to use the Wilcoxon test instead of the t-test.

The program to analyze this experiment using the Wilcoxon test is shown below:

```
DATA TUMOR;
   INPUT GROUP $ MASS @@;
DATALINES;
A 3.1 A 2.2 A 1.7 A 2.7 A 2.5
B 0.0 B 0.0 B 1.0 B 2.3
;
PROC NPAR1WAY DATA=TUMOR WILCOXON;
   TITLE 'Nonparametric Test to Compare Tumor Masses';
   CLASS GROUP;
   VAR MASS;
   EXACT WILCOXON;
RUN;
```

First, we have introduced a new feature on the INPUT statement. Normally, when a SAS program has finished reading an observation, it goes to a new line of data for the next observation. The two '@' signs at the end of the INPUT statement instruct the program to "hold the line" and not automatically go to a new line of data for the next observation. This way, we can put data from several observations on one line. By the way, a single @ sign will hold the line for another INPUT statement but goes to the next line of data when the DATALINES statement is encountered. See Chapter 12 for more details on the use of single and double trailing @ signs.

PROC NPAR1WAY performs the nonparametric tests. The options WILCOXON and MEDIAN request these particular tests. The CLASS and VAR statements are identical to the CLASS and VAR statements of the t-test procedure. A recent addition to the NPAR1WAY procedure is the EXACT statement. Inclusion of this statement causes the program to compute exact p-values (in addition to the asymptotic approximations usually computed) for the tests listed after this statement. In this example, we requested exact p-value computations for the Wilcoxon Rank Sum Test. We suggest that you include the EXACT statement when you have relatively small sample sizes.

The output from the NPAR1WAY procedure follows:

```
Nonparametric Test to Compare Tumor Masses

N P A R 1 W A Y  P R O C E D U R E

Wilcoxon Scores (Rank Sums) for Variable MASS
Classified by Variable GROUP

                    Sum of    Expected      Std Dev          Mean
GROUP       N       Scores    Under H0      Under H0         Score

A           5        33.0        25.0     4.06543697    6.60000000
B           4        12.0        20.0     4.06543697    3.00000000
Average Scores Were Used for Ties

Wilcoxon 2-Sample Test     S = 12.0000

    Exact P-Values
     (One-sided) Prob <= S          = 0.0317
     (Two-sided) Prob >= |S - Mean| = 0.0635

    Normal Approximation (with Continuity Correction of .5)
    Z = -1.84482        Prob > |Z| = 0.0651

    T-Test Approx. Significance = 0.1023

Kruskal-Wallis Test (Chi-Square Approximation)
CHISQ = 3.8723      DF = 1      Prob > CHISQ = 0.0491
```

The sum of ranks for groups A and B are shown, as well as their expected values. The exact two-tailed p-value for this test is .0635 which is quite close to the Normal Approximation value of .0651.

E. *One-tailed versus Two-tailed Tests*

When we conduct an experiment like the tumor example of Section D, we have a choice for stating the alternate hypothesis. In our example, the null hypothesis is that the mass of tumors is the same for groups A and B. The alternative hypothesis is that groups A and B are not the same. We would reject the null hypothesis if A > B or B > A. This type of hypothesis requires a two-tailed test. If we were using a t-test, we would have to consider absolute values of t greater than the critical value. If our alpha level is .05, then we have .025 from each tail of the t-distribution.

In some research studies, the researcher has a reasonable expectation of the results. If we are confirming the results of a previous positive drug study we may have an expectation that the drug will perform better than a placebo. If we are testing whether people prefer blue widgets to red widgets, then we probably do not have an expectation either way. Whenever a directional alternative hypothesis (e.g., B > A) can be justified from the substantive issues in the study, then a one-tailed test can be used. With a one-tailed test, the 5% of the curve associated with the .05 alpha level can all be located in one tail, which increases the power of the study (i.e., makes it more likely of finding a significant difference if, in fact, one exists). If our tumor example had been stated as a one-tailed test, we could have divided the p-value by 2, giving p=.0317 for the Wilcoxon test probability. The decision to do a one-tailed test should be based on an understanding of the theoretical considerations of the study and not as a method of reducing the p-value below the .05 level.

F. Paired T-tests (Related Samples)

Our t-test example in Section B had subjects randomly assigned to a control or treatment group. Therefore, the groups could be considered to be independent.

There are many experimental situations where each subject receives both treatments. For example, each subject in the earlier example could have been measured in the absence of the drug (control value) and after having received the drug (treatment value). The response times for the control and treatment groups would no longer be independent. We would expect a person with a very short response time in the control situation to also have a short response time after taking the drug (compared to the other people who took the drug). We would expect a positive correlation between the control and treatment values.

Our regular t-test cannot be used here since the groups are no longer independent. A variety of the t-test, referred to as a paired t-test, is used instead. The differences between the treatment and control times are computed for each subject. If most of the differences are positive, we suspect that the drug lengthens reaction time. If most are about zero, the drug has no effect. The paired t-test computes a mean and standard error of the differences and determines the probability that the absolute value of the mean difference was greater than zero by chance alone.

Before we program this problem, it should be mentioned that this would be a very poor experiment. What if the response time increased because the subjects became tired? If each subject were measured twice, without being given a drug, would the second value be different because of factors other than the treatment? One way to control for this problem is to take half of the subjects and measure their drug response time first and the control value later, after the drug has worn off. We show a

better way to devise experiments to handle this problem in Chapter 8 (Repeated Measures Designs).

Experiments that measure the same subject under different conditions are sometimes called repeated measures experiments. They do have the problem just stated: one measurement might affect the next. However, if this can be controlled, it is much easier to show treatment effects with smaller samples compared to a regular t-test.

If we used two independent groups of people, we would find that within the control group or the treatment group there would be variation in response times because of individual differences. However, if we measure the same subject under two conditions, even if that person has much longer or shorter response times than the other subjects, the difference between the scores should approximate the difference for other subjects. Thus, each subject acts as his/her own control and therefore controls some of the natural variation between subjects.

The SAS system does not include a paired t-test as part of PROC TTEST. We will need to compute the difference scores ourselves and then use PROC MEANS to compute the probability that the difference is significantly different from zero.

Our data are arranged like this:

Subject	Control Value	Treatment Value
1	90	95
2	87	92
3	100	104
4	80	89
5	95	101
6	90	105

The program is written as follows:

```
DATA PAIRED;
   INPUT CTIME TTIME;
   DIFF = TTIME - CTIME;
DATALINES;
90 95
87 92
100 104
80 89
95  101
90  105
;
PROC MEANS DATA=PAIRED N MEAN STDERR T PRT;
   TITLE 'Paired T-test Example';
   VAR DIFF;
RUN;
```

The variable names CTIME and TTIME were chosen to represent the response times in the control and treatment conditions, respectively. For each observation, we are calculating the difference between the treatment and control times, by creating a new variable called DIFF.

PROC MEANS is followed by a list of options. N, MEAN, and STDERR cause the number of subjects, the mean, and the standard error of the mean to be printed. In our case, these statistics will be computed for the variable DIFF. The options T and PRT will give us the t-value and its associated probability, testing the null hypothesis that the variable DIFF comes from a population whose mean is zero. The output is shown below:

```
Paired T-test Example

Analysis Variable : DIFF

N          Mean    Std Error              T      Prob>|T|
---------------------------------------------------------
6     7.3333333    1.6865481     4.3481318         0.0074
---------------------------------------------------------
```

In this example, the mean difference is positive (7.3) and the probability of the difference occurring by chance is .0074. We can state that response times are longer under the drug treatment compared to the control values. Had the mean difference been negative, we would state that response times were shorter under the drug treatment, because DIFF was computed as treatment time minus control time.

Problems

6-1. The following table shows the time for subjects to feel relief from headache pain:

Aspirin	Tylenol
(Relief time in minutes)	
40	35
42	37
48	42
35	22
62	38
35	29

Write a SAS program to read these data and perform a t-test. Is either product significantly faster than the other (at the .05 level)?

6-2. Using the same data as for problem 6-1, perform a Wilcoxon rank-sum test. Include a request for an exact p-value.

6-3. Eight subjects are tested to see which of two medications (A or B) works best for headaches. Each subject tries each of the two drugs (for two different headaches) and the time span to pain relief is measured. Ignoring an order effect, what type of test would you use to test if one drug is faster than the other? Below are some made-up data: Write the SAS statements to run the appropriate analysis.

Subject	Drug A	Drug B
1	20	18
2	40	36
3	30	30
4	45	46
5	19	15
6	27	22
7	32	29
8	26	25

***6-4.** A researcher wants to randomly assign 30 patients to one of three treatment groups. Each subject has a unique subject number (SUBJ). Write a SAS program to assign these subjects to a treatment group and produce a listing of subjects and groups in subject order (ascending).

6-5. What's wrong with this program?

```
1   DATA DRUGSTDY;
2      INPUT SUBJ 1-3 DRUG 4 HEARTRATE 5-7 SBP 8-10
3      DBP 11-13;
4      AVEBP=DBP + (SBP-DBP)/3;
5   DATALINES;
    0011064130080
    0021068120076
    0031070156090
    0042080140080
    0052088180092
    0062098178094
    ;
6   PROC NPAR1WAY DATA=DRUGSTDY WILCOXON MEDIAN;
7      TITLE 'MY DRUG STUDY';
8      CLASS DRUG;
9      VAR HEARTRATE SBP DBP AVEBP;
10  RUN;

11  PROC T-TEST DATA=DRUGSTDY;
12     CLASS DRUG;
13     VAR HEARTRATE SBP DBP AVEBP;
14  RUN;
```

CHAPTER

7

Analysis of Variance

A. *Introduction*

When you have more than two groups, a t-test (or the nonparametric equivalent) is no longer applicable. Instead, we use a technique called analysis of variance. This chapter covers analysis of variance designs with one or more independent variables, as well as more advanced topics such as interpreting significant interactions, unbalanced designs, and analysis of covariance. You will have to wait until the next chapter to see how to analyze repeated measures designs.

B. *One-way Analysis of Variance*

We have analyzed experiments with two groups using a t-test. Now, what if we have more than two groups? Take the situation where we have three treatment groups: A, B, and C. It was once the practice to use t-tests with this design, comparing A with B, A with C, and B with C. With four groups (ABCD) we would have comparisons AB, AC, AD, BC, BD, and CD. As the number of groups increases, the number of possible comparisons grows rapidly. What is wrong with this procedure of multiple comparisons? Any time we show the means of two groups to be significantly different, we state a probability that the difference occurred by chance alone. Suppose we make 20 multiple comparisons and each comparison is made at the .05 level. Even if there are no real differences between the groups, it is likely that we will find one comparison significant at the .05 level. (The actual probability of at least one significant

difference by chance alone is .64.) The more comparisons that are made, the greater the likelihood of finding a pair of means significantly different by chance alone.

The method used today for comparisons of three or more groups is called analysis of variance (ANOVA). This method has the advantage of testing whether there are any differences between the groups with a single probability associated with the test. The hypothesis tested is that all groups have the same mean. Before we present an example, note that there are several assumptions that should be met before an analysis of variance is used.

Essentially, the same assumptions for a t-test need to be met when conducting an ANOVA. That is, we must have independence between groups (unless a repeated measures design is used); the sampling distributions of sample means must be normally distributed; and the groups should come from populations with equal variances (called homogeneity of variance).

The analysis of variance technique is said to be "robust," a term used by statisticians which means that the assumptions can be violated somewhat, but the technique can still be used. So, if the distributions are not perfectly normal or if the variances are unequal, we may still use analysis of variance. The judgment of how serious a violation to permit is subjective, and, if in doubt, see your local statistician. (Winer has an excellent discussion of the effect of homogeneity of variance violations and the use of analysis of variance.) Balanced designs (those with the same number of subjects under each of the experimental conditions) are preferred to unbalanced designs, especially when the group variances are unequal.

Consider the following experiment:

We randomly assign 15 subjects to three treatment groups X, Y, and Z (with five subjects per treatment). Each of the three groups has received a different method of speed-reading instruction. A reading test is given, and the number of words per minute is recorded for each subject. The following data are collected:

X	Y	Z
700	480	500
850	460	550
820	500	480
640	570	600
920	580	610

The null hypothesis is that mean (X) = mean (Y) = mean (Z). The alternative hypothesis is that the means are not all equal. The means of groups X, Y, and Z are 786, 518, and 548, respectively. How do we know if the means obtained are different because of differences in the reading programs or because of random sampling error? By chance, the five subjects we choose for group X might be faster readers than those chosen for groups Y and Z.

In our example, the mean reading speed of all 15 subjects (called the GRAND MEAN) is 617.33. Now, we normally think of a subject's score as whatever it happens to be, 580 is 580. But we could also think of 580 as being 37.33 points lower than the grand mean.

We might now ask the question, "What causes scores to vary from the grand mean?" In this example, there are two possible sources of variation, the first source is the training method (X, Y, or Z). If X is a far superior method, then we would expect subjects in X to have higher scores, in general, than subjects in Y or Z. When we say "higher scores in general" we mean something quite specific. We mean that being a member of group X causes one's score to increase by so many points.

The second source of variation is due to the fact that individuals are different. Therefore, within each group there will be variation. We can think of a formula to represent each person's score:

The person's score = The grand mean	+	An addition or subtraction from the grand mean depending on which group the person is in,	+	An addition or subtraction depending on the individual's variability.

Now that we have the ideas down, let's return briefly to the mathematics.

It turns out that the mathematics are simplified if, instead of looking at differences in scores from the grand mean, we look instead at the square of the differences. The sum of all the squared deviations is called the total SUM OF SQUARES or, SS, total.

To be sure this is clear, we calculate the total SS in our example. Subtracting the grand mean (617.33) from each score, squaring the differences (usually called deviations), and adding up all the results, we have:

$$\text{SS total} = (700 - 617.33)^2 + (850 - 617.33)^2 + \cdots + (610 - 617.33)^2.$$

As mentioned earlier, we can separate the total variation into two parts: one due to differences in the reading methods (often called SUM OF SQUARES BETWEEN (groups)) and the other due to the normal variations between subjects (often called the SUM OF SQUARES ERROR). Note that the word ERROR here is not the same as "mistake." It simply means that there is variation in the scores that we cannot attribute to a specific variable. Some statisticians call this RESIDUAL instead of ERROR.

Intuitively, we know that if there is no difference between the group means in the populations then, on the average, the group means should differ from each other about as much as the observations differ from each other. In fact, the logical argument that this is the case is straightforward, but it would take several pages to explain. If we take the "average" sum of squares due to group differences (MEAN SQUARE between) divided by the "average" sum of squares due to subject differences (MEAN SQUARE error), the result is called an F ratio:

$$F = \frac{\text{MS between}}{\text{MS error}} = \frac{\text{SS between}/(k-1)}{\text{SS error}/(N-k)}, \quad k = \text{number of groups}$$

If the variation between the groups is large compared to the variation within the groups, this ratio will be larger than 1. If the null hypothesis is true, the expected value for the two mean squares will be equal, and the F statistic will be equal to 1.00. Just how far away from 1.00 is too far away to be attributable to chance is a function of the number of groups and the number of subjects in each group. SAS analysis of variance procedures will give us the F ratio and the probability of obtaining a value of F this large or larger by chance alone, when the null hypothesis is true.

The following box contains a more detailed explanation of how an F ratio is computed. (If you prefer, you may wish to skip the box for now.)

Consider a one-way ANOVA with three groups (A, B, C) and three subjects within each group. The design is as follows:

```
                  A    B    C

             ↑   50   70   20
   Within    |   40   80   15
   Group     ↓   60   90   25

   Means         50   80   20    50 Grand Mean
                 ←———————→
                  Between
                  Groups
```

If we want to estimate the within-group variance (also called ERROR variance), we take the deviation of each score from its group mean, square the result, and add up the squared deviations for each of the groups, then divide the result by the degrees of freedom. (The number of degrees of freedom is N − k where N is the total number of subjects and k is the number of groups.) In the example above, the within-group variance is equal to:

$$[(50-50)^2 + (40-50)^2 + (60-50)^2 + (70-80)^2 + \cdots + (25-20)^2]/6 = 450/6 = 75.0.$$

This within-group variance estimate (75.0) can be compared to the between-group variance, the latter obtained by taking the squared deviations of each group mean from the grand mean, multiplying each deviation by the number of subjects in a group, and dividing by the degrees of freedom (k − 1). In our example the between-group variance is

$$[3(50-50)^2 + 3(80-50)^2 + 3(20-50)^2]/2 = 2700$$

If the null hypothesis is true, the between-group variance estimate will be close to the within-group variance and the ratio

$$F = \frac{\text{Between-group Variance}}{\text{Within-group Variance}}$$

will be close to 1. In our example, F = 2700/75 = 36.0 with probability of .0005 of obtaining a ratio this large or larger by chance alone.

We can write the following program (using our reading-speed data):

```
DATA READING;
   INPUT GROUP $ WORDS @@;
DATALINES;
X 700   X 850   X 820   X 640   X 920
Y 480   Y 460   Y 500   Y 570   Y 580
Z 500   Z 550   Z 480   Z 600   Z 610
;
PROC ANOVA DATA=READING;
   TITLE 'Analysis of Reading Data';
   CLASS GROUP;
   MODEL WORDS = GROUP;
   MEANS GROUP;
RUN;
```

If you forgot what the double trailing @ signs at the end of the INPUT statement do, check Chapter 12 for an explanation. This style of arranging the data values is not necessary to run PROC ANOVA; we're just saving space. One observation per line will do fine.

Now, look at the section of the program beginning with the statement "PROC ANOVA." The first thing we want to indicate is which variable(s) is/are going to be the independent variable(s). We do this with a "CLASS" statement. (NOTE: The term CLASSES is equivalent and may also be used.) We are using the variable named GROUP as our independent variable. It could have been GENDER or TREAT or any other variable in a SAS data set. It is almost always the case that the independent variable(s) in an ANOVA will have a relatively small number of possible values. Next, we want to specify what our MODEL is for the analysis. Basically, we are saying what the dependent and independent variables are for this analysis. Following the word MODEL, is our dependent variable or a list of dependent variables, separated by spaces. When we have more than one dependent variable, a separate analysis of variance will be performed on each of them. For the simplest analysis, a one-way ANOVA, following the dependent variable(s) is an equal sign, followed by the independent variable. You must be sure to list any of the independent variables in the MODEL statement in the previous CLASS statement. In the next line, MEANS GROUP will give us the mean value of the dependent variable (WORDS) for each level of GROUP. Output from this program is shown below:

```
Analysis of Reading Data

Analysis of Variance Procedure
Class Level Information

Class    Levels    Values

GROUP         3    X Y Z

Number of observations in data set = 15

Analysis of Reading Data

Analysis of Variance Procedure

Dependent Variable: WORDS

                            Sum of          Mean
Source              DF     Squares        Square   F Value   Pr > F
Model                2   215613.33     107806.67     16.78   0.0003
Error               12    77080.00       6423.33
Corrected Total     14   292693.33

            R-Square        C.V.    Root MSE      WORDS Mean

            0.736653    12.98256      80.146          617.33

Source              DF   Anova SSMean Square   F Value   Pr > F

GROUP                2   215613.33  107806.67    16.78   0.0003
```



```
[Continued]

Analysis of Reading Data

Analysis of Variance Procedure

Level of        ------------WORDS-----------
GROUP       N        Mean              SD

X           5   786.000000      113.929803
Y           5   518.000000       54.037024
Z           5   548.000000       58.051701
```

The output begins by recapitulating the details of the analysis. This is particularly helpful if you are running a number of ANOVAs at the same time. Pay attention to the levels of each CLASS variable to be sure that there are no data errors, resulting in extraneous levels of one or more CLASS variables. Then we are given the number of cases (observations) in the data set and the name of the dependent variable.

Next comes the stuff of the analysis. There are usually two sections here: an analysis for the MODEL as a whole and a breakdown according to the contribution of each independent variable. Where the ANOVA only has one independent variable (a "one-way ANOVA") these two sections are quite similar. Let's look at the top section first. We see the terms "Source," "DF," "Sum of Squares," "Mean Square," "F Value," and "Pr > F = ." Source tells us what aspect of the analysis we are considering. We have "Model," "Error," and "Corrected Total" as categories here. Model means all of the independent variables and their interactions added together. Error means the residual variation after the Model variation has been removed. In our one-way ANOVA, we have only GROUP to consider. It has two degrees of freedom (DF). The sum of squares is 215613.33. The mean square (SS/DF) is 107806.67. The next row contains the same information for the error term. In the third line we see the DF and SUM OF SQUARES for the CORRECTED TOTAL (which just means the total sum of squares about the grand mean). To the right, we find the F statistic and the probability of it having occurred by chance. Below this is the R-SQUARE for the Model, the coefficient of variation (C.V.) and the mean and standard deviation for the dependent variable.

The next section uses the same terms described above only now each independent variable or interaction is listed separately. Since we only have one independent variable, the results look identical to the ones above. In this example, we would therefore reject the null hypothesis since our F (with 2 and 12 degrees of freedom) is 16.78 and the p-value is .0003 and conclude that the reading instruction methods were not all equivalent.

Now that we know the reading methods are different, we want to know what the differences are. Is X better than Y or Z? Are the means of groups Y and Z so close that we cannot consider them different? In general, methods used to find group differences after the null hypothesis has been rejected are called post hoc, or multiple comparison tests. SAS software provides us with a variety of these tests to investigate differences between levels of our independent variable. These include Duncan's multiple-range test, the Student-Newman-Keuls' multiple-range test, least significant-difference test, Tukey's studentized range test, Scheffe's multiple-comparison procedure, and others. To request a post hoc test, place the SAS option name for the test you want, following a slash (/) on the MEANS statement. The SAS names for the post hoc tests previously listed are DUNCAN, SNK, LSD, TUKEY, and SCHEFFE, respectively. In practice, it is easier to include the request for a multiple-comparison test at the same time we request the analysis of variance. If the analysis of variance is not significant, WE SHOULD NOT LOOK FURTHER AT THE POST HOC TEST RESULTS. (This is our advice. Some statisticians may not agree, especially when certain post hoc tests are used.) Our examples will use Duncan's multiple-range test for post hoc comparisons. You may use any of the available methods in the same manner. Winer (see Chapter 1) is an excellent reference for analysis of variance and experimental design. A discussion of most of these post hoc tests can be found there.

For our example we have:

```
MEANS GROUP / DUNCAN;
```

Unless we specify otherwise, the differences between groups are evaluated at the .05 level. Alpha levels of .1 or .01 may be specified by following the post hoc option name with ALPHA=.1 or ALPHA=.01. For example, to specify an alpha level of .1 for a Scheffe test, we would have

```
MEANS GROUP / SCHEFFE ALPHA=.1;
```

Here is the output from the Duncan procedure in our example:

```
Analysis of Reading Data

Analysis of Variance Procedure

Duncan's Multiple Range Test for variable: WORDS

NOTE: This test controls the type I comparisonwise error rate,
      not the experimentwise error rate

Alpha= 0.05      df= 12   MSE= 6423.333

Number of Means       2      3
Critical Range    110.4  115.6

Means with the same letter are not significantly different.

Duncan Grouping           Mean     N  GROUP

              A          786.00    5  X

              B          548.00    5  Z
              B
              B          518.00    5  Y
```


The listing above uses the following method to show group differences:

On the right are the group identifications. The order is determined by the group means, from highest to lowest. At the far left is a column labeled "Duncan Grouping." Any groups that are not significantly different from one another will have the same letter in the Grouping column. In our example, the Y and Z groups both have the letter 'B' in the GROUPING column and are therefore not significantly different. The letter 'B' between GROUP Z and GROUP Y is there for visual effect. It helps us realize that groups Y and Z are not significantly different (at the .05 level). Group X has an A in the grouping column and is therefore significantly different ($p < .05$) from the Y and Z groups.

From this Duncan's test we conclude that

1. Method X is superior to both methods Y and Z.
2. Methods Y and Z are not significantly different.

How would we describe the statistics used and the results of this experiment in a journal article? Although there is no "standard" format, we suggest one approach here. The key is clarity. Here is our suggestion:

Method. We compared three reading methods: (1) Smith's Speed-Reading Course, (2) Standard Method, and (3) Evelyn Tree's Institute. Fifteen subjects were randomly assigned to one of the three methods. At the conclusion of training, a standard reading test (Cody Count the Words Test version 2.1) was administered.

Results. The mean reading speed for the three methods is

Method	Reading Speed (words per minute)
1. Smith's	786
2. Standard	518
3. Tree's	548

A one-way analysis of variance was performed. The F-value was 16.78 (df = 2, 12, $p = .0003$). A Duncan multiple-range test ($p = .05$) shows that Smith's method is significantly superior to either Tree's or the Standard method. Tree's and the Standard method are not significantly different from each other at the .05 level.

The results of the Duncan multiple-range test can readily be described in words when there are only three groups. With four or more groups, especially if the results are complicated, we can use another method. Consider the following results of a Duncan test on an experiment with four treatment groups:

Grouping	Mean	N	Group
A	80	10	1
A			
A B	75	10	3
B			
B	72	10	2
C	60	10	4

To begin, notice that the groups are now ordered from highest mean to lowest mean, 1, 3, 2, and 4. We see that groups 1 and 3 are not significantly different. (They both have the letter "A" in the Grouping column). Neither are groups 2 and 3. Remember that "not significantly different" does not mean "equal." But 1 and 2 are significantly different!

Finally, group 4 is significantly different from all the other groups ($p < .05$). We can describe these results in a journal article as represented above, or like this:

	Group			
	1	3	2	4
Mean	80	75	72	60

Duncan multiple-range test example

Any two groups with a common underscore are not significantly different ($p < .05$).

A final possibility is to simply put the findings within the text of the article; e.g., "group 4 is different from 1, 2, and 3; group 1 is different from group 2."

C. Computing Contrasts

Now back to our reading example. Suppose you decide, before you perform this experiment, that you want to make some specific comparisons. For example, if method X is a new method and methods Y and Z are more traditional methods, you may decide to compare method X to the mean of method Y and method Z to see if there is a difference between the new and traditional methods. You may also want to compare method Y to method Z to see if there is a difference between the two traditional methods. These comparisons are called contrasts, planned comparisons, or a priori comparisons. Note that the use of CONTRASTS does not give you any protection against a type I error, as do the various multiple-comparison methods described earlier.

To specify comparisons using SAS software, you need to use PROC GLM (General Linear Model) instead of PROC ANOVA. PROC GLM is similar to PROC ANOVA and uses many of the same options and statements. However, PROC GLM is a more generalized program and can be used to compute contrasts or to analyze unbalanced designs (See Section G).

Here are the CONTRAST statements for making the planned comparisons described above:

```
PROC GLM DATA=READING;
   TITLE 'Analysis of Reading Data - Planned Comparions';
   CLASS GROUP;
   MODEL WORDS = GROUP;
   CONTRAST 'X VS. Y AND Z' GROUP -2 1 1;
   CONTRAST 'METHOD Y VS Z' GROUP  0 1 -1;
RUN;
```

For this one-way design, the syntax of a CONTRAST statement is the word CONTRAST, followed by a label for this contrast (placed in quotes), the

independent (CLASS) variable name and a set of k coefficients (where k is the number of levels of the class variable).

The rules are simple: (1) The sum of the coefficients must add to zero. (2) The order of the coefficients matches the alphanumeric order of the levels of the CLASS variable if it is not formatted. If you associated a format with the CLASS variable, the order is determined by the formatted values (you can override this by specifying the PROC GLM option, ORDER=DATA, which will order the levels of the CLASS variable by the data values, not the formatted values). (3) A zero coefficient means that you do not want to include the corresponding level in the comparison. (4) Levels with negative coefficients are compared to levels with positive coefficients.

The first CONTRAST statement in the program above gives you a comparison of method X against the mean of methods Y and Z. The second CONTRAST statement will only perform a comparison between methods Y and methods Z.

Here is a portion of the output showing the results of the comparisons requested:

```
Analysis of Reading Data - Planned Comparisons

Contrast          DF    Contrast SS   F Value    Pr > F

X VS. Y AND Z      1  213363.333333     33.22    0.0001
METHOD Y VS Z      1    2250.000000      0.35    0.5649
```

Notice that method X is shown to be significantly different from methods Y and Z combined, and there is no difference between methods Y and Z at the .05 level.

D. Analysis of Variance: Two Independent Variables

Suppose we ran the same experiment for comparing reading methods, but using 15 male and 15 female subjects. In addition to comparing reading-instruction methods, we could compare male versus female reading speeds. Finally, we might want to see if the effects of the reading methods are the same for males and females.

This experimental design is called a two-way analysis of variance. The "two" refers to the fact that we have two independent variables: GROUP and GENDER. We can picture this experiment as follows:

		GROUP		
		X	Y	Z
GENDER	Male	700	480	500
		850	460	550
		820	500	480
		640	570	600
		920	580	610
	Female	900	590	520
		880	540	660
		899	560	525
		780	570	610
		899	555	645

This design shows that we have each of the three reading-instruction methods (specified by the variable GROUP) for each level of GENDER (male/female). Designs of this sort are called factorial designs. The combination of GROUP and GENDER is called a cell. For this example, males in group X constitute a cell. In general, the number of cells in a factorial design would be the number of levels of each of the independent variables multiplied together. In this case, three levels of GROUP times two levels of GENDER equals six cells.

The total sum of squares is now divided or partitioned into four components. We have the sum of squares due to GROUP differences and the sum of squares due to GENDER differences. The combination of GROUP and GENDER provides us with another source of variation (called an interaction), and finally, the remaining sum of squares is attributed to error. We discuss the interaction term later in this chapter.

Since there are the same number of subjects in each cell, the design is said to be "balanced" (some statisticians call this "orthogonal"). When we have more than one independent variable in our model, we cannot use PROC ANOVA if our design is unbalanced. For unbalanced designs, PROC GLM (general linear model) is used instead. The programming of our balanced design experiment is similar to the one-way analysis of variance. Here is the program:

```
DATA TWOWAY;
   INPUT GROUP $ GENDER $ WORDS;
DATALINES;
X M 700
   .
   .
Z F 645
;
PROC ANOVA DATA=TWOWAY;
   TITLE 'Analysis of Reading Data';
   CLASS GROUP GENDER;
   MODEL WORDS = GROUP | GENDER;
   MEANS GROUP | GENDER / DUNCAN;
RUN;
```

As before, following the word CLASS is a list of independent variables. The vertical line between GROUP and GENDER in the MODEL and MEANS statements indicates that we have a factorial design (also called a crossed design). If we don't include the vertical line, none of the interaction terms will be estimated. Some computer terminals may not have the "|" symbol on the keyboard. In this case, the term GROUP | GENDER can be written as

```
GROUP GENDER GROUP*GENDER
```

The "|" symbol is especially useful when we have higher-order factorial designs such as GROUP | GENDER | DOSE. Written the long way, this would be

```
GROUP GENDER DOSE GROUP*GENDER GROUP*DOSE
      GENDER*DOSE GROUP*GENDER*DOSE
```


That is, each variable, and every two- and three-way interaction term has to be specified.

Let's study the output of the previous example carefully to see what conclusions we can draw about our experiment. The first portion of the output is shown below:

```
Analysis of Reading Data

Analysis of Variance Procedure
Class Level Information

Class    Levels  Values

GROUP        3   X Y Z

GENDER       2   FEMALE MALE


Number of observations in data set = 30

Dependent Variable: WORDS

Source               DF  Sum of Squares  F Value  Pr > F

Model                 5   531436.166667    23.92  0.0001

Error                24   106659.200000

Corrected Total      29   638095.366667

               R-Square            C.V.              WORDS Mean

               0.832848        10.31264              646.433333

Source               DF        Anova SS  F Value  Pr > F

GROUP                 2   503215.266667    56.62  0.0001
GENDER                1    25404.300000     5.72  0.0250
GROUP*GENDER          2     2816.600000     0.32  0.7314
```

The top portion labeled "Class Level Information" indicates our two independent variables and the levels of each. The analysis of variance table shows us the sum of squares and mean square for the entire model and the error. Note: Depending on the value of your LINESIZE option, the output may also include mean squares. This overall F value (23.92) and the probability p = .0001 shows us how well the model (as a whole) explains the variation about the grand mean. This could be very important in certain types of studies where we want to create a general, predictive model. In this case, we are more interested in the detailed sources of variation (GROUP, GENDER, and GROUP*GENDER).

Each source of variation in the table has an F value and the probability of obtaining by chance a value of F this large or larger. In our example, the GROUP variable is significant at .0001, and GENDER at .0250. Since there are only two levels of GENDER, we do not need the Duncan test to claim that males and females are significantly different with respect to reading speed (p = .025).

In a two-way analysis of variance, when we look at GROUP effects, we are comparing GROUP levels without regard to GENDER. That is, when the groups are compared we combine the data from both genders. Conversely, when we compare males to females, we combine data from the three treatment groups.

The term GROUP*GENDER is called an interaction term. If group differences were not the same for males and females, we would have a significant interaction. For example, if males did better with method A compared to method B, while females did better with B compared to A, we would expect a significant interaction. In our example, the interaction between GROUP and GENDER was not significant ($p = .73$). (Our next example shows a case where there is a significant interaction.)

The portion of the output resulting from the "MEANS GROUP | GENDER / DUNCAN" request is shown next:

```
Analysis of Variance Procedure

Duncan's Multiple Range Test for variable: WORDS

NOTE: This test controls the type I comparisonwise error rate,
      not the experimentwise error rate

Alpha= 0.05   df= 24   MSE= 4444.133

Number of Means      2       3
Critical Range   61.53   64.63

Means with the same letter are not significantly different.

Duncan Grouping              Mean     N   GROUP

              A            828.80    10   X

              B            570.00    10   Z
              B
              B            540.50    10   Y

Duncan's Multiple Range Test for variable: WORDS

NOTE: This test controls the type I comparisonwise error rate,
      not the experimentwise error rate

Alpha= 0.05      df= 24       MSE= 4444.133

Number of Means      2
Critical Range   50.24

Means with the same letter are not significantly different.

Duncan Grouping              Mean     N   GENDER

              A            675.53    15   FEMALE

              B            617.33    15   MALE
```



```
[Continued]

Level of   Level of         ---------WORDS--------
GROUP      GENDER      N     Mean            SD

X          FEMALE      5    871.600000     51.887378
X          MALE        5    786.000000    113.929803
Y          FEMALE      5    563.000000     18.574176
Y          MALE        5    518.000000     54.037024
Z          FEMALE      5    592.000000     66.011363
Z          MALE        5    548.000000     58.051701
```

The first comparison shows group X significantly different ($p < .05$) from Y and Z. The second table shows that females have significantly higher reading speeds than males. We already know this because GENDER is a significant main effect ($p = .025$), and there are only two levels of GENDER. Following the two Duncan tests are the mean reading speeds (and standard deviations) for each combination of GROUP and GENDER. These values are the means of the six cells in our experimental design.

E. *Interpreting Significant Interactions*

Now, consider an example that has a significant interaction term. We have two groups of children. One group is considered normal; the other, hyperactive. (Hyperactivity is often referred to as attention-deficit hyperactivity disorder, or ADHD; we simply use the term hyperactive.) Each group of children is randomly divided, with one-half receiving a placebo and the other a drug called ritalin. A measure of activity is determined for each of the children, and the following data are collected:

	PLACEBO	RITALIN
NORMAL	50	67
	45	60
	55	58
	52	65
HYPERACTIVE	70	51
	72	57
	68	48
	75	55

We name the variables in this study GROUP (NORMAL or HYPER), DRUG (PLACEBO or RITALIN), and ACTIVITY (activity score). Since the design is balanced (same number of subjects per cell), we can use PROC ANOVA. The DATA step and PROC statements are written like this:

```
DATA RITALIN;
   DO GROUP = 'NORMAL','HYPER ';
      DO DRUG = 'PLACEBO','RITALIN';
         DO SUBJ = 1 TO 4;
            INPUT ACTIVITY @;
```

[Continued]

```
            OUTPUT;
         END;
      END;
   END;
DATALINES;
50 45 55 52  67 60 58 65  70 72 68 75  51 57 48 55
;
PROC ANOVA DATA=RITALIN;
   TITLE 'Activity Study';
   CLASS GROUP DRUG;
   MODEL ACTIVITY=GROUP | DRUG;
   MEANS GROUP | DRUG;
RUN;
```

Before discussing the ANOVA results, let us precede that with a few words about the DATA step above. One way to read the data values is a straightforward INPUT DRUG $ GROUP $ ACTIVITY; statement. Instead we use nested DO loops, more as a demonstration of SAS programmming rather than to shorten the program. Notice two things: One, you can write DO loops in a SAS DATA step with character values for the loop variable. This is a very useful and powerful feature of the language. Second, when we nest DO loops (place one inside the other) the rule is to complete all the iterations of an inner loop before dropping back to an outer loop. So, in the example above, we set the value of GROUP to 'NORMAL', the value of DRUG to 'PLACEBO', and then read in four values of ACTIVITY. If you have trouble understanding this DATA step, you can always resort to the simpler INPUT statement without loops.

This ANOVA design is another example of a two-way analysis of variance factorial design. The vertical bar between GROUP and DRUG in the MODEL and MEANS statements indicates that we have a factorial design and GROUP and DRUG are crossed. Notice that we do not need to request a multiple-comparison test since there are only two levels of each independent variable.

A portion of the output is shown below:

```
Activity Study

Analysis of Variance Procedure

Source        DF        Anova SS    F Value    Pr > F

GROUP          1    121.00000000       8.00    0.0152
DRUG           1     42.25000000       2.79    0.1205
GROUP*DRUG     1    930.25000000      61.50    0.0001
```

The first thing to notice is that there is a strong GROUP*DRUG interaction term (p = .0001). When this occurs, we must be careful about interpreting any of the

main effects (GROUP and DRUG in our example). That is, we must first understand the nature of the interactions before we examine main effects.

By looking more closely at the interaction between GROUP and DRUG, we see why the main effects shown in the analysis of variance table can be misleading. The best way of explaining a two-way interaction is to take the cell means and plot them. These means can be found in the portion of the output from the MEANS request. The portion of the output containing the cell means is shown below:

```
Level of   Level of          -------- ACTIVITY --------
GROUP      DRUG        N       Mean          SD

HYPER      PLACEBO     4   71.2500000    2.98607881
HYPER      RITALIN     4   52.7500000    4.03112887
NORMAL     PLACEBO     4   50.5000000    4.20317340
NORMAL     RITALIN     4   62.5000000    4.20317340
```

We can use this set of means to plot an interaction graph. We choose one of the independent variables (we choose DRUG) to go on the x-axis and then plot means for each level of the other independent variable (GROUP). We can either do this by hand or have SAS plot it for us. To have SAS plot the interaction graph, we first have to use PROC MEANS to create a data set containing the cell means. The SAS statements to create a data set of cell means is shown next:

```
PROC MEANS DATA=RITALIN NWAY NOPRINT;
   CLASS GROUP DRUG;
   VAR ACTIVITY;
   OUTPUT OUT=MEANS MEAN=;
RUN;
```

Notice that we use GROUP and DRUG as CLASS variables and the NWAY option of PROC MEANS since this will restrict the output data set to the highest order interaction (the cell means). Next, we use PROC PLOT to plot the interaction graph. We can choose to place either of the independent variables on the x-axis and plot a separate graph for each level of the other independent variable. We choose DRUG to be the x-axis variable and plot a separate graph for each level of GROUP. A shortcut, using the values of GROUP as the plotting symbol, make the SAS statements simple. We write:

```
PROC PLOT DATA=MEANS;
   PLOT ACTIVITY*DRUG=GROUP;
RUN;
```

The resulting graph is shown below:

```
Plot of ACTIVITY*DRUG. Symbol is value of GROUP.

        80 +
           |
           |  H
        70 +
           |
ACTIVITY   -
           |
        60 +                                    N
           |
           |
        50 +  N                                 H
           ---+---------------------------------+--
           PLACEBO                           RITALIN

                          DRUG
```

To begin, most interaction plots would include straight lines connecting the normal groups and the hyperactive groups. However, unless we use more sophisticated programs like SAS Graph®, we have to draw in the lines by hand.

The graph shows that normal children increase their activity when given ritalin, while hyperactive children are calmed by ritalin. In the analysis of variance, the comparison of placebo to ritalin values is done by combining the data from normal and hyperactive children. Since these values tend to cancel each other, the average activity with placebo and ritalin is about the same. What we have found here is that it is not possible to understand the activity level of children just by knowing whether they had ritalin. One must also know whether they are hyperactive. This is why it is critical to understand the interaction before looking at main effects. If we really want to study the effect of ritalin, we should look separately at normal and hyperactive children. For each of these groups we have two levels of the DRUG. We can therefore do a t-test between placebo and ritalin within the normal and hyperactive groups. As we know from Chapter 6, this is accomplished by first sorting the data set by GROUP and then including a BY variable in the t-test request. We have:

```
PROC SORT DATA=RITALIN;
   BY GROUP;
RUN;

PROC TTEST DATA=RITALIN;
   BY GROUP;
   CLASS DRUG;
   VAR ACTIVITY;
RUN;
```


The output from these statements is shown below:

```
Activity Study

TTEST PROCEDURE

GROUP=HYPER*****************************************************

Variable: ACTIVITY

DRUG          N           Mean          Std Dev        Std Error
-----------------------------------------------------------------
PLACEBO       4      71.25000000     2.98607881      1.49303941
RITALIN       4      52.75000000     4.03112887      2.01556444

Variances       T        DF  Prob > | T |
------------------------------------------
Unequal     7.3755      5.5       0.0005
Equal       7.3755      6.0       0.0003

For H0: Variances are equal, F' = 1.82 DF = (3,3) Prob>F' = 0.6343

GROUP=NORMAL****************************************************

Variable: ACTIVITY

DRUG             N           Mean      Std Dev       Std Error
-----------------------------------------------------------------
PLACEBO          4      50.50000000   4.20317340   2.10158670
RITALIN          4      62.50000000   4.20317340   2.10158670

Variances        T               DF      Prob>| T |
----------------------------------------------------
Unequal      -4.0376            6.0         0.0068
Equal        -4.0376            6.0         0.0068

For H0: Variances are equal, F'=1.00 DF=(3,3) Prob>F'=1.0000
```

Notice that in both groups the two drug means are significantly different ($p < .05$). However, in the normal group, the ritalin mean is higher than the placebo mean, while in the hyperactive group the reverse is true. So, watch out for those interactions!

An alternative to the t-tests above is to break down the two-way ANOVA into a one-way ANOVA by creating an independent (CLASS) variable that has a level for each combination of the original independent variables. In our case, we create a variable (let's call it COND) that has a level for each combination of DRUG and GROUP. Thus, we have NORMAL-PLACEBO, NORMAL-RITALIN, HYPER-PLACEBO, and HYPER-RITALIN as levels of our COND variable. A quick and easy way to create this variable is to concatenate (join) the two original independent variables. This will be performed directly for character variables. In the case of numeric variables, SAS software will convert them to character and perform the concatenation. In the SAS system, concatenation is accomplished with the concatenation operator, '| |'. To create the COND variable, we add a single line to our data step like this:

```
COND = GROUP || DRUG;
```

This line creates a new variable (COND) which has four values (HYPER PLACEBO, HYPER RITALIN, NORMAL PLACEBO, and NORMAL RITALIN). We can remove the extra spaces between the words if we wish, using the COMPRESS or TRIM functions described in Chapter 18, but let's leave it alone for now. We simply have a one-way design with the single factor (COND) having four levels. The SAS statements to produce the one-way ANOVA are:

```
PROC ANOVA DATA=RITALIN;
   TITLE 'One-way ANOVA Ritalin Study';
   CLASS COND;
   MODEL ACTIVITY = COND;
   MEANS COND / DUNCAN;
RUN;
```

The results of running this procedure are:

```
Class Level Information

Class      Levels    Values

COND          4      HYPER   PLACEBO  HYPER    RITALIN
                     NORMAL  PLACEBO  NORMAL   RITALIN

Number of observations in data set = 16

Dependent Variable: ACTIVITY

Source                  DF   Sum of Squares    F Value     Pr>F

Model                    3    1093.50000000      24.10   0.0001

Error                   12     181.50000000

Corrected Total         15    1275.00000000

               R-Square            C.V.        ACTIVITY Mean

               0.857647       6.5638604          59.25000000

Source                  DF          Anova SS   F Value   Pr > F

COND                     3     1093.50000000     24.10   0.0001
```



```
[Continued]

Duncan's Multiple Range Test for variable: ACTIVITY

NOTE: This test controls the type I comparisonwise error rate,
      not the experimentwise error rate

Alpha= 0.05   df= 12   MSE= 15.125

Number of Means          2            3            4
Critical Range   5.9802927    6.2645621    6.4545514

Means with the same letter are not significantly different.

Duncan Grouping        Mean        N COND

              A      71.250        4 HYPER   PLACEBO

              B      62.500        4 NORMAL  RITALIN

              C      52.750        4 HYPER   RITALIN
              C
              C      50.500        4 NORMAL  PLACEBO
```

Notice that this analysis tells us more than the two t-tests. Besides verifying that PLACEBO is different from RITALIN within each GROUP (NORMAL and HYPER), we can also see the other pairwise comparisons.

There is another way of comparing the two drugs within each level of GROUP without using either of the two methods just described. If we run PROC GLM (general linear model) instead of PROC ANOVA, we can issue two CONTRAST statements that will make the two within-group comparisons for us. This method is considered more correct statistically than the two t-tests by some statisticians since it uses all the data to estimate the error variance. It is, however, equivalent to the one-way analysis above. We present it here without too much explanation. It is difficult, and you will need to seek help beyond this book. First, the program:

```
PROC GLM DATA=RITALIN;
   CLASS GROUP DRUG;
   MODEL ACTIVITY = GROUP | DRUG;
   CONTRAST 'Hyperactive only' DRUG 1 -1
                               GROUP*DRUG 1 -1 0 0;
   CONTRAST 'Normals only'     DRUG 1 -1
                               GROUP*DRUG 0 0 1 -1;
RUN;
```

The first contrast compares PLACEBO to RITALIN only for the hyperactive children. The second, only for the normal children. The "real" methodology for writing these CONTRAST statements is quite complicated. We present a simple algorithm just for this simple two-way design. The order of the variables in the CLASS statement is very important. You must also recognize that the order of the levels of each CLASS variable will be determined by the formatted values (if a format is provided) or by the actual raw data values (if a format is not supplied). In this example we have:

```
GROUP (HYPER - NORMAL) and DRUG (PLACEBO - RITALIN)
```

Note that HYPER is listed before NORMAL because, alphabetically, 'H' comes before 'N'. For each level of GROUP, we list all the levels of DRUG like this:

```
HYPER-PLACEBO HYPER-RITALIN NORMAL-PLACEBO NORMAL-RITALIN
```

We pick the first level of GROUP and then list all the levels of DRUG before going to the next value of GROUP. Now, since we want to compare drugs, we first list DRUG in the CONTRAST statement as 1 −1. Next, for the interaction term GROUP* DRUG, we place our 1 and −1 in the locations that we want to compare. Thus, to compare HYPER-PLACEBO to HYPER-RITALIN we code 1 −1 0 0. To compare NORMAL-PLACEBO to NORMAL-RITALIN we code 0 0 1 −1. OK, if this doesn't make any sense, go ahead and use one of the two methods we presented earlier or consult your friendly statistician. Below is the result of running these two contrasts:

Contrast	DF	Contrast SS	Mean Square	F Value	Pr > F
Hyperactive only	1	684.5000000	684.5000000	45.26	0.0001
Normals only	1	288.0000000	288.0000000	19.04	0.0009

We could obtain the identical results if we used a CONTRAST statement in the one-way model described above where we created a variable (COND) which had four levels representing all combinations of GROUP and DRUG. We would use PROC GLM instead of PROC ANOVA and add two CONTRAST statements following the MODEL statement, thus:

```
PROC GLM DATA=RITALIN;
   TITLE 'One-way ANOVA Ritalin Study';
   CLASS COND;
   MODEL ACTIVITY = COND;
   CONTRAST 'Hyperactive only'   COND  1  -1   0   0;
   CONTRAST 'Normals only'   COND   0   0  1  -1;
RUN;
```

This may be simpler and easier to understand than the CONTRAST statements for two-way designs. Take your choice.

F. N-way Factorial Designs

The method we used to perform a two-way analysis of variance can be extended to cover any number of independent variables. An example with three independent variables (GROUP GENDER DOSE) is shown below (we didn't actually create a data set called THREEWAY):


```
PROC ANOVA DATA=THREEWAY;
   TITLE 'Three-way Analysis of Variance';
   CLASS GROUP GENDER DOSE;
   MODEL ACTIVITY = GROUP | GENDER | DOSE;
   MEANS GROUP | GENDER | DOSE;
RUN;
```

With three independent variables, we have three main effects (GROUP GENDER DOSE), three two-way interactions (GROUP*GENDER GROUP*DOSE GENDER*DOSE), and one three-way interaction (GROUP*GENDER*DOSE). One usually hopes that the higher-order interactions are not significant since they complicate the interpretation of the main effects and the lower-order interactions. (See Winer for a more complete discussion of this topic.)

It clearly becomes difficult to perform factorial design experiments with a large number of independent variables without expert assistance. The number of subjects in the experiment also must be large so that there are a reasonable number of subjects per cell.

G. Unbalanced Designs: PROC GLM

As we mentioned before, designs with an unequal number of subjects per cell are called unbalanced designs. For all designs that are unbalanced (except for one-way designs), we cannot use PROC ANOVA; PROC GLM (general linear model) is used instead. CLASS, MEANS, and MODEL statements for PROC GLM are identical to those used with PROC ANOVA. The only difference between the procedures is the mathematical methods used for each and some additional information that is computed when PROC GLM is used. The real differences come in interpreting results.

Here is an example of a two-way analysis of variance that is unbalanced:

A pudding company wants to test-market a new product. Three levels of sweetness and two flavors are produced. Each subject is given a pudding to taste and is asked to rate the taste on a scale from 1 to 10. The following data are collected:

	Sweetness Level 1	2	3
	9	8	6
	7	7	5
Vanilla	8	8	7
	7		
	9	8	4
Chocolate	9	7	5
	7	6	6
	7	8	4
	8		4

The SAS INPUT statement is written:

```
INPUT SWEET FLAVOR : $9. RATING;
```

Since the number of subjects in each cell is unequal, we use PROC GLM.

```
PROC GLM DATA=PUDDING;
   TITLE 'Pudding Taste Evaluation';
   TITLE3 'Two-way ANOVA - Unbalanced Design';
   TITLE5 '---------------------------------';
   CLASS SWEET FLAVOR;
   MODEL RATING = SWEET | FLAVOR;
   MEANS SWEET | FLAVOR;
   LSMEANS SWEET | FLAVOR / PDIFF;
RUN;
```

Notice a new statement, LSMEANS. With unbalanced designs the MEANS statement will give us unadjusted means, LSMEANS will produce least-square, adjusted means for main effects. We added the PDIFF option which computes probabilities for all pairwise differences. These pairwise comparisons are essentially a series of t-tests and have all the problems of running multiple t-tests.

Portions of the output are shown below:

```
Pudding Taste Evaluation
Two-way ANOVA - Unbalanced Design
---------------------------------

General Linear Models Procedure

Dependent Variable: RATING

Source              DF   Sum of Squares   F Value    Pr > F

Model                5      39.96666667      9.36    0.0002
Error               18      15.36666667
Corrected Total     23      55.33333333

              R-Square             C.V.       RATING Mean

              0.722289         13.52138        6.83333333

Source              DF      Type I SS    F Value    Pr > F

SWEET                2    35.85515873      21.00    0.0001
FLAVOR               1     1.33341530       1.56    0.2274
SWEET*FLAVOR         2     2.77809264       1.63    0.2241

Source              DF    Type III SS    F Value    Pr > F

SWEET                2    29.77706840      17.44    0.0001
FLAVOR               1     1.56666667       1.84    0.1923
SWEET*FLAVOR         2     2.77809264       1.63    0.2241

Level of         -----------RATING----------
SWEET       N        Mean             SD

1           9    7.88888889      0.92796073
2           7    7.42857143      0.78679579
3           8    5.12500000      1.12599163
```

[Continued]

```
Level of                RATING
FLAVOR      N       Mean              SD
                ---------        ------------
Chocolate  14   6.57142857       1.78516475
Vanilla    10   7.20000000       1.13529242

Level of     Level of                       RATING
SWEET        FLAVOR         N        Mean              SD
                                  ----------      ------------
1            Chocolate      5     8.00000000      1.00000000
1            Vanilla        4     7.75000000      0.95742711
2            Chocolate      4     7.25000000      0.95742711
2            Vanilla        3     7.66666667      0.57735027
3            Chocolate      5     4.60000000      0.89442719
3            Vanilla        3     6.00000000      1.00000000

General Linear Models Procedure
Least Squares Means

SWEET      RATING     Pr > |T| H0: LSMEAN (i)=LSMEAN(j)
           LSMEAN     i/j     1        2       3

1      7.87500000     1   .       0.3866  0.0001
2      7.45833333     2  0.3866   .       0.0003
3      5.30000000     3  0.0001  0.0003   .

NOTE: To ensure overall protection level, only probabilities
      associated with pre-planned comparisons should be used.


FLAVOR          RATING    Pr > |T| H0:
                LSMEAN    LSMEAN1=LSMEAN2

Chocolate   6.61666667          0.1923
Vanilla     7.13888889

SWEET   FLAVOR          RATING    LSMEAN
                        LSMEAN    Number

  1     Chocolate   8.00000000      1
  1     Vanilla     7.75000000      2
  2     Chocolate   7.25000000      3
  2     Vanilla     7.66666667      4
  3     Chocolate   4.60000000      5
  3     Vanilla     6.00000000      6

Pr > |T| H0: LSMEAN(i)=LSMEAN(j)

i/j      1       2       3       4       5       6
1        .     0.6914  0.2419  0.6273  0.0001  0.0083
2     0.6914     .     0.4540  0.9073  0.0001  0.0233
3     0.2419  0.4540     .     0.5622  0.0005  0.0934
4     0.6273  0.9073  0.5622     .     0.0003  0.0404
5     0.0001  0.0001  0.0005  0.0003     .     0.0526
```

[Continued]

```
General Linear Models Procedure
Least Squares Means

Least Squares Means for effect SWEET*FLAVOR
Pr > | T | H0: LSMEAN(i)=LSMEAN(j)

Dependent Variable: RATING

   i/j      1      2      3      4      5      6
   6   0.0083 0.0233 0.0934 0.0404 0.0526   .

NOTE: To ensure overall protection level, only probabilities
      associated with pre-planned comparisons should be used.
```

Notice that there are two sets of values for SUM OF SQUARES, F VALUES, and probabilities; one labeled TYPE I, the other labeled TYPE III. When designs do not have equal cell sizes, the TYPE I and TYPE III sums of squares may not be equal for all variables. The difference between TYPE I and TYPE III sum of squares is that TYPE I lists the sums of squares for each variable as if it were entered one at a time into the model, in the order they are specified in the MODEL statement. Hence they can be thought of as incremental sums of squares. If there is any variance that is common to two or more variables, the variance will be attributed to the variable that is entered first. This may or may not be desirable. The TYPE III sum of squares gives the sum of squares that would be obtained for each variable if it were entered last into the model. That is, the effect of each variable is evaluated after all other factors have been accounted for. In any given situation, whether you want to look at TYPE I or TYPE III, sum of squares will vary; however, for most analysis of variance applications, you will want to use TYPE III sum of squares.

Just to keep you on your toes, we have added to the program a new form of the TITLE statement. As you probably can guess, TITLE3 provides a third title line; TITLE5, a fifth. Since TITLE2 and TITLE4 are missing, lines 2 and 4 will be blank. In general, TITLEn will be the nth title line on the SAS output, where n is an integer. Note that TITLE is equivalent to TITLE1.

In our example, the sweetness factor was significant (p = .0001). The probabilities for FLAVOR and the interaction between FLAVOR and SWEETNESS were .1923 and .2241, respectively.

H. *Analysis of Covariance*

If you have a variable, IQ for example, which may be correlated with your dependent measure, you may want to adjust for possible differences due to the confounding variable before analyzing your dependent variable. Consider the following data set:

GROUP			
A		B	
Math Score	IQ	Math Score	IQ
260	105	325	126
325	115	440	135
300	122	425	142
400	125	500	140
390	138	600	160

We want to compare groups A and B on math scores. However, we notice that there seems to be a relationship between math scores and IQ and that group B seems to have higher IQ scores. We can test the relationship between math score and IQ by computing a correlation coefficient, and we can see if there is a significant difference in IQ scores between groups A and B with a t-test. Here is a program that does just that:

```
DATA COVAR;
   LENGTH GROUP $ 1;
   INPUT GROUP MATH IQ @@;
DATALINES;
A 260 105  A 325 115  A 300 122  A 400 125  A 390 138
B 325 126  B 440 135  B 425 142  B 500 140  B 600 160
;
PROC CORR DATA=COVAR NOSIMPLE;
   TITLE 'Covariate Example';
   VAR MATH IQ;
RUN;

PROC TTEST DATA=COVAR;
   CLASS GROUP;
   VAR IQ MATH;
RUN;
```

Notice that we requested a t-test for math scores as well (while we were at it). Here are the results:

```
Covariate Example

Correlation Analysis

   2 'VAR' Variables: MATH IQ

Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0
/ N=10
```

[Continued]

```
         MATH       IQ

MATH   1.00000    0.92456
        0.0        0.0001

IQ     0.92456    1.00000
       0.0001     0.0

Covariate Example

TTEST PROCEDURE

Variable: IQ

GROUP      N            Mean          Std Dev       Std Error
-------------------------------------------------------------
A          5    121.00000000      12.22701926      5.46808925
B          5    140.60000000      12.48198702      5.58211429

Variances         T        DF    Prob>|T|
-----------------------------------------
Unequal     -2.5083       8.0      0.0365
Equal       -2.5083       8.0      0.0365

For H0: Variances are equal, F' = 1.04  DF = (4,4)
                             Prob>F' = 0.9691

Variable: MATH

GROUP      N            Mean          Std Dev       Std Error
-------------------------------------------------------------
A          5    335.00000000      59.58187644     26.64582519
B          5    458.00000000     101.27931674     45.29348739

Variances         T        DF    Prob>|T|
-----------------------------------------
Unequal     -2.3406       6.5      0.0550
Equal       -2.3406       8.0      0.0474

For H0: Variances are equal, F' =2.89 DF = (4,4)
                             Prob>F' =0.3286
```

We see that IQ and math scores are highly correlated (r = .92456, p = .0001) and that there is a significant difference in IQ (p = .0365) and math scores (p = .0474) between groups. We want to correct for the IQ mismatch by running an analysis of covariance.

The first step is to test an important assumption that must be verified before running an analysis of covariance. That is, the slope of the covariate by independent variable must be the same for all levels of the independent variable. We can test for heterogeneity of slopes using the following MODEL statement with PROC GLM:


```
PROC GLM DATA=COVAR;
   CLASS GROUP;
   MODEL MATH=IQ GROUP IQ*GROUP;
RUN;
```

The term IQ*GROUP will test if there are different regression coefficients for the two groups. Running this, we get (partial listing):

```
Covariate Example

Dependent Variable: MATH

Source      DF      Type I SS    F Value    Pr>F

IQ           1  79541.5882838      49.73  0.0004
GROUP        1     96.5979265       0.06  0.8141
IQ*GROUP     1   3816.9637225       2.39  0.1734
```

We see from this that there is no significant difference in the MATH/IQ relationship as a function of GROUP (from the IQ*GROUP interaction term: F = 2.39, p = .1734). We can go ahead and run the analysis of covariance as follows:

```
PROC GLM DATA=COVAR;
   CLASS GROUP;
   MODEL MATH=IQ GROUP;
   LSMEANS GROUP;
RUN;
```

The results (below) show that when we adjust for IQ, there are no longer any differences between the groups on math scores. Notice the LSMEANS output shows the math scores for the two groups adjusted for IQ:

```
Covariate Example
Source      DF     Type III SS    F Value    Pr > F

IQ           1  41815.6862103       21.82    0.0023
GROUP        1     96.5979265        0.05    0.8288
```

```
[Continued]

Least Squares Means

GROUP          MATH
               LSMEAN

A         392.345889
B         400.654111
```

Problems

7-1. The next two questions were inspired by one of the authors (Cody) watching the French Open Tennis tournament while working on problem sets. (McEnroe versus Lendl (1984). Lendl won in five sets.)

Three brands of tennis shoes are tested to see how many months of playing would wear out the soles. Eight pairs of brands A, N, and T are randomly assigned to a group of 24 volunteers. The table below shows the results of the study:

	Brand A	Brand N	Brand T
	8	4	12
	10	7	8
Wear time,	9	5	10
in months	11	5	10
	10	6	11
	10	7	9
	8	6	9
	12	4	12

Are the brands equal in wear quality? Write a SAS program to solve this problem, using ANOVA.

7-2. Tennis balls are tested in a machine to see how many bounces they can withstand before they fail to bounce 30% of their dropping height. Two brands of balls (W and P) are compared. In addition, the effect of shelf life on these brands is tested. Half of the balls of each brand are 6 months old, the other half, fresh. Using a two-way analysis of variance, what conclusions can you reach? The data are shown below:

		Brand W	Brand P
		67	75
		72	76
	New	74	80
		82	72
Age		81	73
		46	63
		44	62
	Old	45	66
		51	62
		43	60

7-3. A taste test is conducted to rate consumer preference between brands C and P of a popular soft drink. In addition, the age of the respondent is recorded (1 = less than 20, 2 = 20 or more). Preference data (on a scale of 1–10) is as follows:

		Brand C	Brand P
		7	9
		6	8
		6	9
	<20	5	9
		6	8
Age			
		9	6
		8	7
		8	6
	>=20	9	6
		7	5
		8	
		8	

(a) Write a SAS program to analyze these data with a two-way analysis of variance. (Careful: Is the design balanced?) NOTE: Go ahead and treat these data as interval data even though some analysts would prefer that you use a nonparametric analysis.

(b) Use SAS software to plot an interaction graph.

(c) Follow up with a t-test comparing brand C to brand P for each age group separately.

7-4. A manufacturer wants to reanalyze the data of problem 7-1, omitting all data for brand N. Run the appropriate analysis.

7-5. What's wrong with this program?

```
1  DATA TREE;
2     INPUT TYPE $ LOCATION $ HEIGHT;
3  DATALINES;
   PINE NORTH 35
   PINE NORTH 37
   PINE NORTH 41
   PINE NORTH 41
   MAPLE NORTH 44
   MAPLE NORTH 41
   PINE SOUTH 53
   PINE SOUTH 55
   MAPLE SOUTH 28
   MAPLE SOUTH 33
   MAPLE SOUTH 32
   MAPLE SOUTH 22
   ;
4  PROC ANOVA DATA=TREE;
5     CLASS TYPE LOCATION;
6     MODEL HEIGHT = TYPE | LOCATION;
7     MEANS TYPE LOCATION TYPE*LOCATION;
8  RUN;
```

*7-6. You want to determine if the mean score on a standardized math test is different among three groups of school children, ranging in age from 12 to 18. Although the test covers only simple math topics, easily understood by a 12 year old, you want to perform the analysis with and without an adjustment based on age.

(a) Using the sample data below, first perform a one-way ANOVA comparing the math scores and ages among the three groups.

(b) Test if the relationship between age and math score is homogeneous among the three groups.

(c) If the test in part B permits, perform an analysis of covariance, comparing the math scores among the three groups, adjusted for age.

Math Scores and Ages for Groups A, B, and C

Group A		Group B		Group C	
Math Score	Age	Math Score	Age	Math Score	Age
90	16	92	18	97	18
88	15	88	13	92	17
72	12	76	12	88	16
82	14	78	14	92	17
65	12	90	17	99	17
74	13	68	12	82	14

CHAPTER

8

Repeated Measures Designs

A. *Introduction*

This chapter covers the analysis of repeated measures designs. First, a few words about terminology before we begin this topic. Our use of the term "repeated" is based on a common use in the medical field and described in the text, *Statistical Principles in Experimental Design*, Second Edition, by B.J. Winer (1991). We use the term "repeated" in this chapter to mean any factor where each subject is measured at every level for that factor. For example, if a runner is timed running on two different types of track surfaces, we are considering "surface" as a repeated measure factor. Other authors use the term "repeated" to refer only to factors that cannot be assigned in random order, such as time. When treatments are randomized, the interpretation of a significant effect can be attributed to treatments and not to the order of presentation. This is often referred to as a split-plot or within-subjects design. If you use the latter meaning of "repeated," feel free to substitute your design terminology in the examples in this chapter. These designs often fall into the category that statisticians call mixed designs, or designs with within-subjects factors.

Designs in this chapter involve a repeated measurement on the unit of analysis (usually subjects) in one or more of the independent variables. For example, an experiment where each subject receives each of four drugs or an experiment where each subject is measured each hour for five hours would need a repeated measures design. With the introduction of version 6 of SAS software, a REPEATED statement was added to the analysis of variance procedures (ANOVA and GLM) which greatly simplified the coding of repeated measures designs. As you will see, there are times when you will want to analyze your data using the REPEATED statement, and there will be times when you will choose not to. For each of the repeated measures designs in this chapter, we demonstrate both methods of analysis.

B. One-factor Experiments

Consider the following experiment. We have four drugs (1, 2, 3, and 4) that relieve pain. Each subject is given each of the four drugs. The subject's pain tolerance is then measured. Enough time is allowed to pass between successive drug administrations so that we can be sure there is no residual effect from the previous drug. In clinical terms, this is called a "wash-out" period.

The null hypothesis is

```
MEAN(1)=MEAN(2)=MEAN(3)=MEAN(4)
```

If the analysis of variance is significant at $p < .05$, we want to look at pairwise comparisons of the drugs using Duncan's multiple-range test or other post hoc tests.

Notice how this experiment differs from a one-way analysis of variance without a repeated measure. With the designs we have discussed thus far, we would have each subject receive only one of the four drugs. In this design, each subject is measured under each of the drug conditions. This has several important advantages.

First, each subject acts as his own control. That is, drug effects are calculated by recording deviations between each drug score and the average drug score for each subject. The normal subject-to-subject variation can thus be removed from the error sum of squares. Let's look at a table of data from the pain experiment:

SUBJECT	DRUG 1	DRUG 2	DRUG 3	DRUG 4
1	5	9	6	11
2	7	12	8	9
3	11	12	10	14
4	3	8	5	8

To analyze this experiment, we consider the subject to be an independent variable. We therefore have SUBJECT and DRUG as independent variables.

One way of arranging our data and writing our INPUT statement would be like this:

```
DATA PAIN;
   INPUT SUBJ DRUG PAIN;
DATALINES;
1 1 5
1 2 9
1 3 6
1 4 11
2 1 7
(more data lines)
```


It is usually more convenient to arrange all the data for each subject on one line:

SUBJ	DRUG 1	DRUG 2	DRUG 3	DRUG 4
1	5	9	6	11
2	7	12	8	9
3	11	12	10	14
4	3	8	5	8

We can read the data as arranged above, but restructure it to look as if we had read it with the first program as follows:

```
DATA PAIN;
   INPUT SUBJ @; ①
   DO DRUG = 1 to 4; ②
      INPUT PAIN @; ③
      OUTPUT;  ④
   END;  ⑤
DATALINES;
1 5 9 6 11
2 7 12 8 9
3 11 12 10 14
4 3 8 5 8
;
```

The first INPUT statement ① reads the subject number. The "@" sign following SUBJ is an instruction to keep reading from the same line of data. (See Chapter 12 for a more detailed discussion of the trailing @ sign.) Statement ② starts an iterative loop. The value of DRUG is first set to 1. Next, the input statement ③ is executed. Again, if the "@" were omitted, the program would go to the next data line to read a value (which we don't want). The OUTPUT statement ④ causes an observation to be written to the SAS data set. Look at our first line of data. We would have as the first observation in the SAS data set SUBJ = 1, DRUG = 1, and PAIN = 5. When the END statement ⑤ is reached, the program flow returns to the DO statement ② where DRUG is set to 2. A new PAIN value is then read from the data (still from the first line because of the trailing @) and a new observation is added to the SAS data set. This continues until the value of DRUG = 4. Normally, when a DATA step reaches the end, signalled by a DATALINES or RUN statement, an observation is automatically written out to the SAS data set being created. However, since we included an OUTPUT statement in the DATA step to write out our observations, the automatic writing out of an observation at the end of the DATA step does not occur. (If it did, we would get a duplicate of the observation where DRUG = 4.) Since the DO loop has completed, the program control returns to the top of the DATA step to line ①. The program will read the subject number from the next line of data.

The general form of a DO statement is

```
DO variable = start TO end BY increment;
   (SAS Statements)
END;
```

Where "start" is the initial value for "variable," "end" is the ending value, and "increment" is the increment. If "increment" is omitted, it defaults to 1.

The first few observations in the SAS data set created from this program are listed below:

```
OBS   SUBJ   DRUG   PAIN
1      1      1      5
2      1      2      9
3      1      3      6
4      1      4     11
5      2      1      7
6      2      2     12
7      2      3      8
8      2      4      9
9      3      1     11
        etc.
```

We can make one small modification to the program and, by so doing, avoid having to enter the subject numbers on each line of data. The new program looks as follows:

```
DATA PAIN;
   SUBJ+1; ①
   DO DRUG=1 TO 4;
      INPUT PAIN @;
      OUTPUT;
   END;
DATALINES;
5 9 6 11
7 12 8 9
11 12 10 14
3 8 5 8
;
```

Statement ① creates a variable called SUBJ, which starts at 1 and is incremented by 1 each time the statement is executed.

We are ready to write our PROC statements to analyze the data. With this design there are several ways to write the MODEL statement. One way is like this:

```
PROC ANOVA DATA=PAIN;
   TITLE 'One-way Repeated Measures ANOVA';
   CLASS SUBJ DRUG;
   MODEL PAIN = SUBJ DRUG;
   MEANS DRUG/DUNCAN;
RUN;
```


Notice that we are not writing SUBJ | DRUG. We are indicating that SUBJ and DRUG are each main effects and that there is no interaction term between them. Once we have accounted for variations from the grand mean due to subjects and drugs, the remaining deviations will be our source of error.

Below is a portion of the output from the one-way repeated measures experiment:

```
One-way Repeated Measures ANOVA

Analysis of Variance Procedure
Class Level Information

Class    Levels    Values

SUBJ          4    1 2 3 4

DRUG          4    1 2 3 4


Number of observations in data set = 16


One-way Repeated Measures ANOVA

Analysis of Variance Procedure

Dependent Variable: PAIN
                                Sum of           Mean
Source                DF       Squares         Square    F Value    Pr > F

Model                  6     120.50000       20.08333      13.64    0.0005

Error                  9      13.25000        1.47222

Corrected Total       15     133.75000

             R-Square           C.V.       Root MSE              PAIN Mean

             0.900935       14.06785         1.2134                 8.6250


Source                DF       Anova SS   Mean Square    F Value    Pr > F

SUBJ                   3      70.250000     23.416667      15.91    0.0006
DRUG                   3      50.250000     16.750000      11.38    0.0020


Duncan's Multiple Range Test for variable: PAIN

NOTE: This test controls the type I comparisonwise error rate,
      not the experimentwise error rate
```

[Continued]

```
Alpha= 0.05   df= 9   MSE= 1.472222

Number of Means        2     3     4
Critical Range     1.941 2.026 2.075

Means with the same letter are not significantly different.
Duncan Grouping              Mean      N DRUG

            A             10.5000      4  4
            A
            A             10.2500      4  2

            B              7.2500      4  3
            B
            B              6.5000      4  1
```

What conclusions can we draw from these results? Looking at the very bottom of the analysis of variance table, we find an F value of 11.38 with an associated probability of .0020. We can therefore reject the null hypothesis that the means are equal. Another way of saying this is that the four drugs are not equally effective for reducing pain. Notice that the SUBJ term in the analysis of variance table also has an F value and a probability associated with it. This merely tells us how much variability there was from subject to subject. It is not really interpretable in the same fashion as the drug factor. We include it as part of the model because we don't want the variability associated with it to go into the ERROR sum of squares.

Now that we know that the drugs are not equally effective, we can look at the results of the Duncan Multiple-Range Test. This shows two drug groupings. Assuming that a higher mean indicates greater pain, we can say that drugs 1 and 3 were more effective in reducing pain than drugs 2 and 4. We cannot, at the .05 level, claim any differences between drugs 1 and 3 or between drugs 2 and 4.

Looking at the error SS and the SS due to subjects, we see that SUBJECT SS (70.25) is large compared to the ERROR SS (13.25). Had this same set of data been the result of assigning 16 subjects to the four different drugs (instead of repeated measures), the error SS would have been 83.5 (13.25 + 70.25). The resulting F and p values for the DRUG effect would have been 2.41 and .118, respectively. (Note that the degrees of freedom for the error term would be 12 instead of 9.)

We see, therefore, that controlling for between-subject variability can greatly reduce the error term in our analysis of variance and allow us to identify small treatment differences with relatively few subjects.

C. *Using the REPEATED Statement of PROC ANOVA*

This same design can be analyzed using the REPEATED option, first introduced with version 6 of SAS software. When the REPEATED statement is used, we need our data set in the form:


```
SUBJ PAIN1 PAIN2 PAIN3 PAIN4
```

where PAIN1-PAIN4 are the pain levels at each drug treatment. Notice that the data set does not have a DRUG variable. The ANOVA statements to analyze this experiment are:

```
DATA REPEAT1;
   INPUT PAIN1-PAIN4;
DATALINES;
5 9 6 11
7 12 8 9
11 12 10 14
3 8 5 8
;
PROC ANOVA DATA=REPEAT1;
   TITLE 'One-way ANOVA Using the REPEATED Statement';
   MODEL PAIN1-PAIN4 = / NOUNI;
   REPEATED DRUG 4 (1 2 3 4);
RUN;
```

Notice several details. First, there is no CLASS statement; our data set does not have an independent variable. We specify the four dependent variables to the left of the equal sign in the MODEL statement. Since there is no CLASS variable, we have nothing to place to the right of the equals sign. The option NOUNI (no univariate) is a request not to conduct a separate analysis for each of the four PAIN variables. Later, when we have both repeated and nonrepeated factors in the design, this option will be especially important. The repeated statement indicates that we want to call the repeated factor DRUG. The "4" following the variable name indicates that DRUG has four levels. This is optional. Had it been omitted, the program would have assumed as many levels as there were dependent variables in the MODEL statement. The number of levels needs to be specified only when we have more than one repeated measure factor. Finally, "(1 2 3 4)" indicates the labels we want printed for each level of DRUG. The labels also act as spacings when polynomial contrasts are requested. (See the SAS/STAT manual for more details on this topic.) We omit the complete output from running this procedure, but will show you some excerpts and leave the details of the output to the next model.

```
One-way ANOVA Using the REPEATED Statement

Analysis of Variance Procedure
Repeated Measures Analysis of Variance
Repeated Measures Level Information

Dependent Variable     PAIN1    PAIN2    PAIN3    PAIN4

    Level of DRUG          1        2        3        4
```

```
[Continued]

Univariate Tests of Hypotheses for Within Subject Effects

Source: DRUG
                                                    Adj    Pr > F
    DF   Anova SS Mean Square  F Value  Pr > F   G - G    H - F
     3 50.2500000  16.7500000    11.38  0.0020  0.0123   0.0020

Source: Error(DRUG)
    DF   Anova SS Mean Square
     9 13.2500000   1.4722222

Greenhouse-Geisser Epsilon = 0.5998
     Huynh-Feldt Epsilon = 1.4433
```

The F value and probabilities (F = 11.38, p = .002) are identical to those in the previous output. Notice that two additional p-values are included in this output. The Adjusted p-values (labeled 'Adj. Pr > F') shown to the right are more conservative, the G-G representing the Greenhouse-Geisser correction, and the H-F representing the Huynh-Feldt value. (See Edwards for an explanation; reference in Chapter 1.) Here is a brief explanation: There are some assumptions in repeated measures designs which are rather complicated. You may see these mentioned as symmetry tests or as sphericity tests. Somewhat simplified, what is being tested is the assumption that the variances and correlations are the same among the various dependent variables. It is more or less an extension of the assumption of equal variances in t-tests or ANOVA or equal slopes in ANCOVA. The SAS output above shows the unadjusted p-values along with the G-G and H-F adjusted values, the more conservative adjustment being the Greenhouse-Geisser.

When we use the REPEATED statement, we cannot use a MEANS statement with the repeated factor name. The only way to compute pairwise comparisons in this case is to use the keyword CONTRAST(n) with the REPEATED statement. The form is:

```
REPEATED factor_name CONTRAST(n);
```

where n is a number from 1 to k, with k being the number of levels of the repeated factor. CONTRAST(1) compares the first level of the factor with each of the other levels. If the first level were a control value, for example, the CONTRAST(1) statement would compare the control to each of the other drugs. If we want all of the pairwise contrasts, we need to write k − 1 repeated statements. In our DRUG example, where there are four levels of DRUG, we write:

```
PROC ANOVA DATA=REPEAT1;
   TITLE 'One-way ANOVA Using the Repeated Statement';
   MODEL PAIN1-PAIN4 = / NOUNI;
   REPEATED DRUG 4 CONTRAST(1)/ NOM SUMMARY;
   REPEATED DRUG 4 CONTRAST(2)/ NOM SUMMARY;
   REPEATED DRUG 4 CONTRAST(3)/ NOM SUMMARY;
RUN;
```


These three CONTRAST statements produce all the two-way comparisons. (CONTRAST(1) gives us 1 vs. 2, 3, 4; CONTRAST(2) gives us 2 vs. 1, 3, 4; and CONTRAST(3) gives us 3 vs. 1, 2, 4.) The contrasts are equivalent to multiple t-tests between the levels, and you may want to protect yourself against a type I error with a Bonferroni correction or some other method. There is no provision for making these corrections using existing SAS procedures. The option NOM asks that no multivariate statistics be printed; the option SUMMARY requests analysis of variance tables for each contrast defined by the repeated factor.

D. *Two-factor Experiments with a Repeated Measure on One Factor*

One very popular form of a repeated measures design is the following:

	SUBJ	PRE	POST
Control	1		
	2		
	3		
Treatment	4		
	5		
	6		

Subjects are randomly assigned to a control or treatment group. Then, each subject is measured before and after treatment. Obviously, in this case, the TIME factor (PRE and POST) cannot be randomized. The "treatment" for the control group can either be a placebo or no treatment at all. The goal of an experiment of this sort is to compare the pre/post changes of the control group to the pre/post changes of the treatment group. This design has a definite advantage over a simple pre/post design where one group of subjects is measured before and after a treatment (such as having only a treatment group in our design). Simple pre/post designs suffer from the problem that we cannot be sure if it is our treatment that causes a change (e.g., TIME may have an effect). By adding a pre/post control group, we can compare the pre/post control scores to the pre/post treatment scores and thereby control for any built-in, systematic, pre/post changes.

A simple way to analyze our design is to compute a difference score (post minus pre) for each subject. We then have two groups of subjects with one score each (the difference score). Then we use a t-test to look for significant differences between the difference scores of the control and treatment groups. With more than two levels of time, however, we will need to use analysis of variance.

Here are some sample data and a SAS program that calculates difference scores and computes a t-test:

	SUBJ	PRE	POST
Control	1	80	83
	2	85	86
	3	83	88
Treatment	4	82	94
	5	87	93
	6	84	98

```
DATA PREPOST;
   INPUT SUBJ GROUP $ PRETEST POSTEST;
   DIFF = POSTEST-PRETEST;
DATALINES;
1 C 80 83
2 C 85 86
3 C 83 88
4 T 82 94
5 T 87 93
6 T 84 98
;
PROC TTEST DATA=PREPOST;
   TITLE 'T-test on Difference Scores';
   CLASS GROUP;
   VAR DIFF;
RUN;
```

Results of this analysis show the treatment mean difference to be significantly different from the control mean difference (p=.045). See below:

```
T-test on Difference Scores

TTEST PROCEDURE

Variable: DIFF

GROUP   N          Mean          Std Dev       Std Error
---------------------------------------------------------
C       3    3.00000000       2.00000000      1.15470054
T       3   10.66666667       4.16333200      2.40370085

Variances        T      DF      Prob>|T|
----------------------------------------
Unequal    -2.8750     2.9        0.0686
Equal      -2.8750     4.0        0.0452

For H0: Variances are equal, F' = 4.33 DF = (2,2) Prob>F' = 0.3750
```

We can alternatively treat this design as a two-way analysis of variance (GROUP × TIME) with TIME as a repeated measure. This method has the advantage of analyzing designs with more than two levels on one or both factors.

We first write a program using the REPEATED statement of ANOVA. No changes in the data set are necessary. The ANOVA statements are:

```
PROC ANOVA DATA=PREPOST;
   TITLE1 'Two-way ANOVA with a Repeated Measure on One Factor';
   CLASS GROUP;
   MODEL PRETEST POSTEST = GROUP / NOUNI;
   REPEATED TIME 2 (0 1);
   MEANS GROUP;
RUN;
```


The REPEATED statement indicates that we want to call the repeated factor TIME, it has two levels, and we want to label the levels 0 and 1.

Output from these procedure statements are shown below:

```
Two-way ANOVA with a Repeated Measure on One Factor

Analysis of Variance Procedure
Class Level Information

Class     Levels    Values

GROUP     2         C T

Number of observations in data set = 6

Two-way ANOVA with a Repeated Measure on One Factor

Analysis of Variance Procedure
Repeated Measures Analysis of Variance
Repeated Measures Level Information

Dependent Variable  PRETEST  POSTEST

   Level of TIME          0        1


Manova Test Criteria and Exact F Statistics for
the Hypothesis of no TIME Effect
H = Anova SS&CP Matrix for TIME    E = Error SS&CP Matrix

S=1   M=-0.5   N=1

Statistic                   Value        F    Num DF  Den DF  Pr > F

Wilks' Lambda           0.1321631   26.266       1       4  0.0069
Pillai's Trace          0.8678369   26.266       1       4  0.0069
Hotelling-Lawley Trace  6.5664063   26.266       1       4  0.0069
Roy's Greatest Root     6.5664063   26.266       1       4  0.0069

Manova Test Criteria and Exact F Statistics for
the Hypothesis of no TIME*GROUP Effect
H = Anova SS&CP Matrix for TIME*GROUP      E = Error SS&CP Matrix

S=1   M=-0.5   N=1

Statistic                   Value        F    Num DF  Den DF  Pr > F

Wilks' Lambda           0.3261146   8.2656       1       4  0.0452
Pillai's Trace          0.6738854   8.2656       1       4  0.0452
Hotelling-Lawley Trace  2.0664063   8.2656       1       4  0.0452
Roy's Greatest Root     2.0664063   8.2656       1       4  0.0452
```

[Continued]

```
Two-way ANOVA with a Repeated Measure on One Factor
Analysis of Variance Procedure
Repeated Measures Analysis of Variance
Tests of Hypotheses for Between Subjects Effects

Source          DF        Anova SS     F Value    Pr > F

GROUP            1     90.75000000       11.84    0.0263

Error            4     30.66666667


Univariate Tests of Hypotheses for Within Subject Effects

Source: TIME
                                                     Adj  Pr > F
  DF    Anova SS  Mean Square  F Value  Pr > F   G - G    H - F
   1  140.083333   140.083333    26.27  0.0069     .        .

Source: TIME*GROUP
                                                     Adj  Pr > F
  DF    Anova SS  Mean Square  F Value  Pr > F   G - G    H - F
   1   44.083333    44.083333     8.27  0.0452     .        .

Source: Error(TIME)

  DF    Anova SS  Mean Square
   4   21.333333     5.333333
                 --------              --------       --------             --------
Level of                     PRETEST                          POSTEST
GROUP      N         Mean               SD              Mean                SD

C          3   82.6666667       2.51661148        85.6666667        2.51661148
T          3   84.3333333       2.51661148        95.0000000        2.64575131
```

Finally, it should be noted that these data can also be analyzed through an analysis of covariance using the pretest as a covariate. The choice of analysis depends in part on the precise nature of the question being asked. Bock (1975) discusses this issue.

We will discuss the output from PROC ANOVA after we show an alternative method of analyzing this experiment. However, a portion of the output above is unique and is discussed here. Notice the rows labeled Wilks' Lambda, Pillai's Trace, Hotelling-Lawley Trace, and Roy's Greatest Root. These are multivariate statistics which are of special interest when more than one dependent variable is indicated. Unlike univariate statistics, when you use multivariate procedures, there is no single test analogous to the F-test. Instead, there are about half a dozen. A question arises as to which one to use. Multivariate statisticians spend many pleasant hours investigating this question. The answer to the question is unambiguous: It depends. Bock (1975; see Chapter 1 for references) presents a nice discussion of the options. Our advice is: When there are only very small differences among the p-values, it doesn't really matter which one you use. If in doubt, we suggest two alternatives: (1) Use Wilks' Lamda. It is a likelihood ratio test that is often appropriate, or (2) when there are differences among the p-values, find a consultant.

We now analyze the same experiment as a two-factor analysis of variance without using the REPEATED statement of PROC ANOVA. We may want to do this so that we can use the "built-in" multiple-comparison tests. (You also may not want to do this if you feel you need the "protection" of the more conservative F-values computed in the multivariate model.) To do this, we must first create a new variable—say TIME—which will have two possible values: PRE or POST. Each subject will then have two observations, one with TIME = PRE and one with TIME = POST.

As with our one-way, repeated measures design, the method of creating several observations from one is with the OUTPUT statement.

We can add the following SAS statements to the end of the previous program:

```
DATA TWOWAY;
   SET PREPOST;  ①
   TIME = 'PRE ';  ②
   SCORE = PRETEST;  ③
   OUTPUT;  ④
   TIME = 'POST';  ⑤
   SCORE = POSTEST; ⑥
   OUTPUT;  ⑦
   DROP PRETEST POSTEST DIFF; ⑧
RUN;
```

This section of the program creates a SAS data set called TWOWAY, which has variables SUBJ GROUP TIME and SCORE. The first few observations in this data set are:

SUBJ	GROUP	TIME	SCORE
1	C	PRE	80
1	C	POST	83
2	C	PRE	85
2	C	POST	86

Let's follow this portion of the SAS program step by step to see exactly how the new data set is created.

The SET statement ① causes observations to be read from the original data set, PREPOST. The first observation is

```
SUBJ=1 GROUP=C PRETEST=80 POSTEST=83 DIFF=3.
```

Statement ② creates a new variable called TIME and sets the value of TIME to 'PRE '. Note that there is a space after the 'E' in PRE. The reason is that the length of the variable TIME is defined by the first value that is assigned to it. Had we coded

TIME = 'PRE', the length would be equal to three and the statement TIME = 'POST' would have assigned the value 'POS' to TIME instead of 'POST'.

Statement ③ creates a new variable, SCORE, which is equal to the PRETEST value. When ④ is executed, the first observation of the data set called TWOWAY becomes:

```
SUBJ=1 GROUP=C PRETEST=80 POSTEST=83 DIFF=3 TIME=PRE SCORE=80.
```

However, since we included a DROP statement in ⑧, the first observation in data set TWOWAY only contains the variables SUBJ, GROUP, TIME, and SCORE.

Next, ⑤ sets TIME='POST', and ⑥ sets the variable SCORE to the POSTEST value. A new observation is added to the data set TWOWAY with the second OUTPUT statement ⑦. The second observation has SUBJ=1 GROUP=C TIME=POST SCORE=83.

The RUN statement ends the DATA step and control returns to the top of the DATA step where a new observation is read from data set PREPOST.

We are now ready to write our ANOVA statements. Unlike any of our previous examples, we have to specify all the terms, including the sources of error, in the MODEL statement. This is necessary because our main effects and interaction terms are not tested by the same error term. Therefore, we need to specify each of these terms in the MODEL statement so they can be used later in tests of our hypotheses. In this design, we have one group of subjects assigned to a control group and another group assigned to a treatment group. Within each group, each subject is measured at TIME=PRE and TIME=POST. In this design, the subjects are said to be nested within the GROUP. In SAS programs, the term subjects nested within group is written:

```
SUBJ(GROUP)
```

Since the model statement defines ALL sources of variation about the grand mean, the ERROR SUM OF SQUARES printed in the ANOVA table will be zero. To specify which error term to be used to test each hypothesis in our design, we use TEST statements following the MODEL specification. A TEST statement consists of a hypothesis to be tested (H=) and the appropriate error term (E=). The entire ANOVA procedure looks as follows:

```
PROC ANOVA DATA=TWOWAY;
   TITLE 'Two-way ANOVA with TIME as a Repeated Measure';
   CLASS SUBJ GROUP TIME;
   MODEL SCORE = GROUP SUBJ(GROUP) TIME
                 GROUP*TIME TIME*SUBJ(GROUP);
   MEANS GROUP|TIME;
   TEST H=GROUP                  E=SUBJ(GROUP);
   TEST H=TIME GROUP*TIME        E=TIME*SUBJ(GROUP);
RUN;
```

Notice that the error term for GROUP is SUBJ(GROUP) (subject nested within group), and the error term for TIME and the GROUP*TIME interaction is TIME*SUBJ(GROUP).

Below are portions of the PROC ANOVA output:

```
Two-way ANOVA with TIME as a Repeated Measure

Analysis of Variance Procedure
Class Level Information

Class     Levels    Values

SUBJ           6    1 2 3 4 5 6

GROUP          2    C T

TIME           2    POST PRE

Number of observations in data set = 12


Dependent Variable: SCORE

Source                 DF    Sum of Squares   F Value   Pr > F

Model                  11      326.91666667         .        .

Error                   0                 .

Corrected Total        11      326.91666667

                R-Square            C.V.            SCORE Mean

                1.000000               0            86.9166667

Source                 DF          Anova SS   F Value   Pr > F

GROUP                   1       90.75000000         .        .
SUBJ(GROUP)             4       30.66666667         .        .
TIME                    1      140.08333333         .        .
GROUP*TIME              1       44.08333333         .        .
SUBJ*TIME(GROUP)        4       21.33333333         .        .


Tests of Hypotheses using the Anova MS for
SUBJ(GROUP) as an error term

Source                 DF          Anova SS   F Value   Pr > F

GROUP                   1       90.75000000     11.84   0.0263


Tests of Hypotheses using the Anova MS for
SUBJ*TIME(GROUP) as an error term

Source                 DF          Anova SS   F Value   Pr > F

TIME                    1      140.08333333     26.27   0.0069
GROUP*TIME              1       44.08333333      8.27   0.0452
```

Since all sources of variation were included in the MODEL statement, the error sum of squares is zero, and the F value is undefined (it prints as a missing value '.'). The requested tests are shown at the bottom of the listing. Group differences have an F = 11.84 and p = .0263. TIME and GROUP*TIME have F-values of 26.27 and 8.27 and probabilities of .0069 and .0452, respectively.

In this experimental design, it is the interaction of GROUP and TIME that is of primary importance. This interaction term tells us if the pre/post changes were the same for control and treatment subjects. An interaction graph will make this clear. The output from the MEANS request is shown below:

```
Level of                  ------------SCORE-----------
GROUP       N             Mean             SD

C           6             84.1666667       2.78687400
T           6             89.6666667       6.28225013

Level of                  ------------SCORE-----------
TIME        N             Mean             SD

POST        6             90.3333333       5.60951572
PRE         6             83.5000000       2.42899156

Level of    Level of                ------------SCORE-----------
GROUP       TIME       N            Mean              SD

C           POST       3            85.6666667        2.51661148
C           PRE        3            82.6666667        2.51661148
T           POST       3            95.0000000        2.64575131
T           PRE        3            84.3333333        2.51661148
```

We can use the last set of means (interaction of GROUP and TIME) to plot the interaction graph. We pick one of the independent variables (we will use TIME) to go on the x-axis and then plot means for each of the levels of the other independent variable (GROUP). A few lines of SAS code will produce the desired interaction plot. Here is the code:

```
PROC MEANS DATA=TWOWAY NOPRINT NWAY;
   CLASS GROUP TIME;
   VAR SCORE;
   OUTPUT OUT=INTER
          MEAN=;
RUN;
OPTIONS LINESIZE=68 PAGESIZE=24;
PROC PLOT DATA=INTER;
   TITLE 'Interaction Plot';
   PLOT SCORE*TIME=GROUP;
RUN;
```


The resulting graph is shown below:

```
Interaction Plot

  Plot of SCORE*TIME.  Symbol is value of GROUP

          95 +  T
             |
             |
          90 +
SCORE        |
             |  C
          85 +                                    T
             |                                    C
             |
          80 +
             ---+---------------------------------+--
               POST                              PRE
                               TIME
```

A significant interaction term shows us that the two pre/post lines are not parallel. This tells us that the change from pre to post was different, depending on which GROUP a subject was in, which is precisely what we wanted to know. The treatment group and control group were quite similar in terms of pain relief before the drug was administered (mean = 84.33 and 82.67). After the drug was given (the POST measure), the treatment group showed dramatic gains and the control group only modest gains. The F-statistic for GROUP × TIME (8.27) and its p-value (.045) tell us that this difference in improvement is greater than could be expected by chance alone. The F-statistic for GROUP (F = 11.84, p = .0263) tells us that if we summed over the pre and post tests, the groups were different. This isn't of use to us since it combines the pre measure (where we anticipated them being the same) with the post measure (where we anticipated a difference). The same logic is true for TIME. Here we are summing over the control and treatment groups. Finally, note that the p-value for GROUP × TIME is the same as for the t-test of the difference scores, because we are essentially making the same test in both analyses. Next, we move to a somewhat more complex setting.

E. *Two-factor Experiments with Repeated Measures on Both Factors*

This design is similar to the previous design except that each subject is measured under all levels of both factors. An example follows:

A group of subjects is tested in the morning and afternoon of two different days. On one of the days, the subjects receive a strong sleeping aid the night before the experiment is to be conducted; on the other, a placebo. The subjects' reaction time to a stimulus is measured. A diagram of the experiment is shown below:

	TREAT			
	CONTROL		DRUG	
TIME	SUBJ	REACT	SUBJ	REACT
A.M.	1	65	1	70
	2	72	2	78
	3	90	3	97
P.M.	1	55	1	60
	2	64	2	68
	3	80	3	85

We would like to see if the drug had any effect on the reaction time and if the effect was the same for the whole day. We can use the AM/PM measurements on the control day as a comparison for the AM/PM changes on the drug day.

Since each subject is measured under all levels of treatment (PLACEBO or DRUG) and TIME (AM/PM), we can treat this experiment as a SUBJ by TREATMENT by TIME factorial design. However, we must specify the error terms to test our hypotheses.

To create our SAS data set, we could use the following statements:

```
DATA SLEEP;
   INPUT SUBJ TREAT $ TIME $ REACT;
DATALINES;
1 CONT AM 65
1 DRUG AM 70
1 CONT PM 55
1 DRUG PM 60
2 CONT AM 72
2 DRUG AM 78
2 CONT PM 64
2 DRUG PM 68
3 CONT AM 90
3 DRUG AM 97
3 CONT PM 80
3 DRUG PM 85
;
```

The ANOVA statements can be written:

```
PROC ANOVA DATA=SLEEP;
   CLASS SUBJ TREAT TIME;
   MODEL REACT = SUBJ|TREAT|TIME;
   MEANS TREAT|TIME;
   TEST H=TREAT        E=SUBJ*TREAT;
   TEST H=TIME         E=SUBJ*TIME;
   TEST H=TREAT*TIME   E=SUBJ*TREAT*TIME;
RUN;
```


Before we investigate the output from the program above, we would like to show an alternative way of programming this problem. This method differs from the one above only in the way that data are read; the PROC statements are exactly the same. The purpose of the alternate programming method is to simplify data entry. Please feel free to skip this discussion if you wish; it is useful only if you will be using SAS software frequently with moderate to large amounts of data. In that case you will save considerable time. Here is the program:

```
    ALTERNATIVE PROGRAM FOR SLEEP STUDY
DATA SLEEP;
   SUBJ+1; ①
   DO TIME=1 to 2;  ②
      DO TREAT=1 TO 2; ③
         INPUT REACT @;  ④
         OUTPUT; ⑤
      END;
   END;
DATALINES;
65 70 55 60
72 78 64 68
90 97 80 85
;
PROC ANOVA DATA=SLEEP;
   CLASS SUBJ TREAT TIME;
   MODEL REACT = SUBJ|TREAT|TIME;
   MEANS TREAT|TIME;
   TEST H=TREAT        E=SUBJ*TREAT;
   TEST H=TIME         E=SUBJ*TIME;
   TEST H=TREAT*TIME   E=SUBJ*TREAT*TIME;
RUN;
```

This program allows us to place all the data for one subject on a single line. We begin creating our data set with the DATA statement. Since we are not explicitly entering a subject number, statement ① will provide us with a SUBJ variable.

The reaction times for each subject are arranged as follows:

```
CONTROL AM - DRUG AM - CONTROL PM - DRUG PM
```

We want to create four observations for each subject (one for each combination of treatment and time). The outer loop ② sets the TIME values while the inner loop ③ sets the TREAT values. Since we used a trailing @ sign ④ the pointer does not move to the next line of data until the first four data values have been read and four observations have been written in line ⑤. At this point SUBJ is incremented, and four more values of reaction time are read with the appropriate values of TIME and TREAT.

A FORMAT statement to assign formats to the variables TREAT and TIME would make output from the statistical procedures easier to read. The complete program, modified to include formats, is shown next:

```
PROC FORMAT;
   VALUE FTREAT 1='Control' 2='Drug';
   VALUE FTIME 1='AM' 2='PM';
RUN;

DATA SLEEP;
   SUBJ+1;
   DO TIME=1 TO 2;
      DO TREAT=1 TO 2;
         INPUT REACT @;
         OUTPUT;
      END;
   END;
   FORMAT TREAT FTREAT. TIME FTIME.;
DATALINES;
65 70 55 60
72 78 64 68
90 97 80 85
;
PROC ANOVA DATA=SLEEP;
   CLASS SUBJ TREAT TIME;
   MODEL REACT = SUBJ |TREAT| TIME;
   MEANS TREAT|TIME;
   TEST H=TREAT          E=SUBJ*TREAT;
   TEST H=TIME           E=SUBJ*TIME;
   TEST H=TREAT*TIME     E=SUBJ*TREAT*TIME;
RUN;

   END OF ALTERNATIVE PROGRAM EXPLANATION
```

Which method you choose to create the SAS data set will not affect the PROC ANOVA statements. In any design where ALL factors are repeated, such as this one, we can treat the SUBJ variable as being crossed by all other factors (as opposed to nested). The MODEL statement is therefore the same as our factorial design. However, by including the SUBJ term in our model, the error term will be zero (as in our previous example). Thus, our ANOVA table will not show F-values or probabilities. These are obtained by specifying TEST statements following the MODEL statement, as described previously.

The error terms to test each hypothesis are simple to remember: For factor X, the error term is SUBJ*X. For example, the error term to test TREAT is SUBJ *TREAT; the error term to test the interaction TREAT*TIME is SUBJ *TREAT*TIME. To specify the correct error term for each main effect and interaction, the three TEST statements following the MODEL statement were added, each specifying a hypothesis to be tested and the error term to be used in calculating the F-ratio.

A portion of the output from PROC ANOVA is shown below:

```
Analysis of Variance Procedure

Dependent Variable: REACT

Source                DF   Sum of Squares   F Value    Pr > F

Model                 11    1750.66666667        .         .

Error                  0         .

Corrected Total       11    1750.66666667

                 R-Square            C.V.          REACT Mean

                 1.000000               0          73.6666667


Source                DF         Anova SS   F Value    Pr > F

SUBJ                   2    1360.66666667        .         .
TREAT                  1      85.33333333        .         .
SUBJ*TREAT             2       0.66666667        .         .
TIME                   1     300.00000000        .         .
SUBJ*TIME              2       2.00000000        .         .
TREAT*TIME             1       1.33333333        .         .
SUBJ*TREAT*TIME        2       0.66666667        .         .

Tests of Hypotheses using the Anova MS for
SUBJ*TREAT as an error term

Source                DF         Anova SS   F Value    Pr > F

TREAT                  1      85.33333333    256.00    0.0039

Tests of Hypotheses using the Anova MS for
SUBJ*TIME as an error term

Source                DF         Anova SS   F Value    Pr > F

TIME                   1     300.00000000    300.00    0.0033

Tests of Hypotheses using the Anova MS for
SUBJ*TREAT*TIME as an error term

Source                DF         Anova SS   F Value    Pr > F

TREAT*TIME             1       1.33333333      4.00    0.1835
```

What conclusions can we draw? (1) The drug increases reaction time ($F = 256.00$, $p = .0039$); (2) reaction time is longer in the morning compared to the afternoon ($F = 300.00, p = .0033$); and (3) we cannot conclude that the effect of the drug on reaction time is related to the time of day (the interaction of TREAT and TIME is not significant $F = 4.00, p = 0.1835$). Note that this study is not a pre/post study as in the previous example. Even so, had the TREAT by TIME interaction been significant, we would have been more cautious in looking at the TREAT and TIME effects.

If this were a real experiment, we would have to control for the learning effect that might take place. For example, if we measure each subject in the same order, we might find a decrease in reaction time from CONTROL AM to DRUG PM because the subject became more familiar with the apparatus. To avoid this, we would either have to acquaint the subject with the equipment before the experiment begins to assure ourselves that the learning has stabilized, or to measure each subject using TREATMENT and TIME in random order.

This design may also be analyzed using the REPEATED statements of PROC ANOVA. If we read in the four reaction times for each subject in the order: AM Control - AM Drug - PM Control - PM Drug, and name our variables REACT1-REACT4, the SAS statements are:

```
DATA REPEAT2;
   INPUT REACT1-REACT4;
DATALINES;
65 70 55 60
72 78 64 68
90 97 80 85
;
PROC ANOVA DATA=REPEAT2;
   MODEL REACT1-REACT4 = / NOUNI;
   REPEATED TIME 2 , TREAT 2  / NOM;
RUN;
```

When there is more than one repeated factor, we need to specify the number of levels of each factor after the factor name. The factor we name first changes the slowest. That is, the first two reaction times are for TIME=AM with REACT1 associated with TREAT=control and REACT2 associated with TREAT=drug. We have now exhausted all levels of TREAT and set TIME=PM; REACT3 and REACT4 are both PM measurements.

F. Three-factor Experiments with a Repeated Measure on the Last Factor

For this example, we consider a marketing experiment. Male and female subjects are offered one of three different brands of coffee. Each brand is tasted twice; once immediately after breakfast, the other time after dinner (the order of presentation is randomized for each subject). The preference of each brand is measured on a

scale from 1 to 10 (1 = lowest, 10 = highest). The experimental design is shown below:

		BRAND (of coffee)								
		A:	Brkfst	Dinner	B:	Brkfst	Dinner	C:	Brkfst	Dinner
		subj			subj			subj		
GENDER	Male	1	7	8	7	4	6	13	8	9
		2	6	7	8	3	5	14	6	9
		3	6	8	9	3	5	15	5	8
	Female	4	5	7	10	3	4	16	6	9
		5	4	7	11	4	4	17	6	9
		6	4	6	12	2	3	18	7	8

In this experiment, the factors BRAND and GENDER are crossed factors while MEAL is a repeated measure factor (each subject tastes coffee after breakfast and dinner). Since a single subject tastes only one brand of coffee and is clearly only one gender, the subject term is said to be nested within BRAND and GENDER (written SUBJ(BRAND GENDER)). We could arrange our data several ways. First, we arrange data so that we can take advantage of the REPEATED statement of ANOVA. To do this, we place all data for each subject on one line. Thus, our program and data will look as follows:

```
DATA COFFEE;
   INPUT SUBJ BRAND $ GENDER $ SCORE_B SCORE_D;
DATALINES;
1 A M 7 8
2 A M 6 7
3 A M 6 8
4 A F 5 7
5 A F 4 7
6 A F 4 6
7 B M 4 6
8 B M 3 5
9 B M 3 5
10 B F 3 4
11 B F 4 4
12 B F 2 3
13 C M 8 9
14 C M 6 9
15 C M 5 8
16 C F 6 9
17 C F 6 9
18 C F 7 8
;
PROC ANOVA DATA=COFFEE;
   TITLE 'Coffee Study';
   CLASS BRAND GENDER;
   MODEL SCORE_B SCORE_D = BRAND|GENDER / NOUNI;
   REPEATED MEAL;
   MEANS BRAND|GENDER;
RUN;
```

Notice that BRAND and GENDER are crossed while MEAL is the repeated measures factor. As before, the option NOUNI on the MODEL statement indicates that we do not want UNIvariate statistics for SCORE_B and SCORE_D.

Selected portions of the output from the above program are shown below:

```
Coffee Study

Analysis of Variance Procedure
Class Level Information

Class     Levels    Values

BRAND          3    A B C

GENDER         2    F M


Number of observations in data set = 18

Analysis of Variance Procedure
Repeated Measures Analysis of Variance
Repeated Measures Level Information

Dependent Variable    SCORE_B   SCORE_D

    Level of MEAL           1         2


Analysis of Variance Procedure
Repeated Measures Analysis of Variance
Tests of Hypotheses for Between Subjects Effects

Source          DF      Anova SS     F Value    Pr > F

BRAND            2   83.38888889       51.76    0.0001
GENDER           1    6.25000000        7.76    0.0165
BRAND*GENDER     2    3.50000000        2.17    0.1566

Error           12    9.66666667

Analysis of Variance Procedure
Repeated Measures Analysis of Variance
Univariate Tests of Hypotheses for Within Subject Effects

Source: MEAL
                                                         Adj  Pr > F
  DF   Anova SS  Mean Square  F Value   Pr > F   G - G   H - F
   1 30.2500000   30.2500000    99.00   0.0001     .       .

Source: MEAL*BRAND
                                                         Adj  Pr > F
  DF   Anova SS  Mean Square  F Value   Pr > F   G - G   H - F
   2  1.5000000    0.7500000     2.45   0.1278     .       .
```

```
[Continued]

Source: MEAL*GENDER
                                                       Adj Pr > F
  DF   Anova SS  Mean Square  F Value   Pr > F   G - G H - F
   1  0.0277778    0.0277778     0.09   0.7682     .     .

Source: MEAL*BRAND*GENDER
                                                       Adj Pr > F
  DF   Anova SS  Mean Square  F Value   Pr > F   G - G H - F
   2  2.0555556    1.0277778     3.36   0.0692     .     .

Source: Error(MEAL)

  DF   Anova SS  Mean Square
  12  3.6666667    0.3055556

Analysis of Variance Procedure

Level of      ---------SCORE_B-------    ---------SCORE_D-------
BRAND      N      Mean          SD          Mean          SD

A          6  5.33333333  1.21106014   7.16666667  0.75277265
B          6  3.16666667  0.75277265   4.50000000  1.04880885
C          6  6.33333333  1.03279556   8.66666667  0.51639778

Level of      ---------SCORE_B-------    ---------SCORE_D-------
GENDER     N      Mean          SD          Mean          SD

F          9  4.55555556  1.58989867   6.33333333  2.23606798
M          9  5.33333333  1.73205081   7.22222222  1.56347192

Level of  Level of            ---------SCORE_B--------
BRAND     GENDER      N         Mean          SD

A         F           3     4.33333333   0.57735027
A         M           3     6.33333333   0.57735027
B         F           3     3.00000000   1.00000000
B         M           3     3.33333333   0.57735027
C         F           3     6.33333333   0.57735027
C         M           3     6.33333333   1.52752523

Level of  Level of            ---------SCORE_D--------
BRAND     GENDER      N         Mean          SD

A         F           3     6.66666667   0.57735027
A         M           3     7.66666667   0.57735027
B         F           3     3.66666667   0.57735027
B         M           3     5.33333333   0.57735027
C         F           3     8.66666667   0.57735027
C         M           3     8.66666667   0.57735027
```

We shall explain the results after the alternative program below:

An alternative program can be written that does not use the REPEATED statement of PROC ANOVA. Although in some models this does not give you the protection of the more conservative multivariate model, you still may want to run such a model. To do so, it is more useful if the data is arranged with two observations per subject and a MEAL variable already in the data set. So, if the data is arranged like this:

SUBJ	BRAND	GENDER	MEAL	SCORE
1	A	M	BRKFST	7
1	A	M	DINNER	8
2	A	M	BRKFST	6
		etc.		

Your INPUT statement would look like this:

```
INPUT SUBJ BRAND $ GENDER $ MEAL $ SCORE;
```

The ANOVA statements are written:

```
PROC ANOVA DATA=COFFEE;
   CLASS SUBJ BRAND GENDER MEAL;
   MODEL SCORE = BRAND GENDER BRAND*GENDER SUBJ(BRAND GENDER)
                 MEAL BRAND*MEAL GENDER*MEAL BRAND*GENDER*MEAL
                 MEAL*SUBJ(BRAND GENDER);
   MEANS BRAND|GENDER / DUNCAN E=SUBJ(BRAND GENDER);
   MEANS MEAL BRAND*MEAL GENDER*MEAL BRAND*GENDER*MEAL;
*-----------------------------------------------------------*
| The following TEST statements are needed to obtain the     |
| correct F and p-values:                                    |
*-----------------------------------------------------------*;
TEST H=BRAND GENDER BRAND*GENDER
     E=SUBJ(BRAND GENDER);
TEST H=MEAL BRAND*MEAL GENDER*MEAL BRAND*GENDER*MEAL
     E=MEAL*SUBJ(BRAND GENDER);
RUN;
```

The first test statement will test each of the terms (BRAND GENDER and BRAND*GENDER) with the error term SUBJ(BRAND GENDER). The effects MEAL, BRAND*MEAL, GENDER*MEAL, and BRAND*GENDER*MEAL will all be tested with the error term MEAL*SUBJ(BRAND GENDER). We have also made a change in the way the MEANS statements were written. Included after the DUNCAN option is an "E=" specification. This is done because the DUNCAN procedure will use the residual mean square as the error term unless otherwise instructed. Since we have completely defined every source of variation in our model, the residual mean square is zero. The "E=error term" option uses the same error term as the "H=" option of the corresponding TEST statement. Also, since different error terms are used to test different hypotheses, a separate MEANS statement is required each time a different error term is used. Note that we did not need to perform a DUNCAN test for MEAL since this variable has only two levels.

A portion of the results of running this alternative program are shown below:

```
Analysis of Variance Procedure Class Level Information

Coffee Study

Analysis of Variance Procedure
Class Level Information

Class     Levels    Values

SUBJ          18    1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

BRAND          3    A B C

GENDER         2    F M

MEAL           2    BRKFST DINNER


Number of observations in data set = 36

Analysis of Variance Procedure

Dependent Variable: SCORE

Source                    DF    Sum of Squares   F Value   Pr > F

Model                     35      140.30555556         .        .

Error                      0                 .

Corrected Total           35      140.30555556

                    R-Square              C.V.        SCORE Mean

                    1.000000                 0        5.86111111

Source                    DF          Anova SS   F Value   Pr > F

BRAND                      2       83.38888889         .        .
GENDER                     1        6.25000000         .        .
BRAND*GENDER               2        3.50000000         .        .
SUBJ(BRAND*GENDER)        12        9.66666667         .        .
MEAL                       1       30.25000000         .        .
BRAND*MEAL                 2        1.50000000         .        .
GENDER*MEAL                1        0.02777778         .        .
BRAND*GENDER*MEAL          2        2.05555556         .        .
SUBJ*MEAL(BRAN*GEND)      12        3.66666667         .        .

Duncan's Multiple Range Test for variable: SCORE

NOTE: This test controls the type I comparisonwise error rate,
      not the experimentwise error rate
```

[Continued]

```
Alpha= 0.05    df= 12   MSE= 0.805556

Number of Means        2      3
Critical Range     .7983  .8356

Means with the same letter are not significantly different.
```

Duncan Grouping	Mean	N	BRAND
A	7.5000	12	C
B	6.2500	12	A
C	3.8333	12	B

Level of BRAND	Level of GENDER	N	SCORE Mean	SCORE SD
A	F	6	5.50000000	1.37840488
A	M	6	7.00000000	0.89442719
B	F	6	3.33333333	0.81649658
B	M	6	4.33333333	1.21106014
C	F	6	7.50000000	1.37840488
C	M	6	7.50000000	1.64316767

Level of MEAL	N	SCORE Mean	SCORE SD
BRKFST	18	4.94444444	1.66175748
DINNER	18	6.77777778	1.92676369

Level of BRAND	Level of MEAL	N	SCORE Mean	SCORE SD
A	BRKFST	6	5.33333333	1.21106014
A	DINNER	6	7.16666667	0.75277265
B	BRKFST	6	3.16666667	0.75277265
B	DINNER	6	4.50000000	1.04880885
C	BRKFST	6	6.33333333	1.03279556
C	DINNER	6	8.66666667	0.51639778

Level of BRAND	Level of MEAL	N	SCORE Mean	SCORE SD
F	BRKFST	9	4.55555556	1.58989867
F	DINNER	9	6.33333333	2.23606798
M	BRKFST	9	5.33333333	1.73205081
M	DINNER	9	7.22222222	1.56347192

[Continued]

Level of BRAND	Level of GENDER	Level of MEAL	N	SCORE Mean	SCORE SD
A	F	BRKFST	3	4.33333333	0.57735027
A	F	DINNER	3	6.66666667	0.57735027
A	M	BRKFST	3	6.33333333	0.57735027
A	M	DINNER	3	7.66666667	0.57735027
B	F	BRKFST	3	3.00000000	1.00000000
B	F	DINNER	3	3.66666667	0.57735027
B	M	BRKFST	3	3.33333333	0.57735027
B	M	DINNER	3	5.33333333	0.57735027
C	F	BRKFST	3	6.33333333	0.57735027
C	F	DINNER	3	8.66666667	0.57735027
C	M	BRKFST	3	6.33333333	1.52752523
C	M	DINNER	3	8.66666667	0.57735027

```
Dependent Variable: SCORE

Tests of Hypotheses using the Anova MS for
SUBJ(BRAND*GENDER) as an error term
```

Source	DF	Anova SS	F Value	Pr > F
BRAND	2	83.38888889	51.76	0.0001
GENDER	1	6.25000000	7.76	0.0165
BRAND*GENDER	2	3.50000000	2.17	0.1566

```
Tests of Hypotheses using the Anova MS for
SUBJ*MEAL(BRAN*GEND) as an error term
```

Source	DF	Anova SS	F Value	Pr > F
MEAL	1	30.25000000	99.00	0.0001
BRAND*MEAL	2	1.50000000	2.45	0.1278
GENDER*MEAL	1	0.02777778	0.09	0.7682
BRAND*GENDER*MEAL	2	2.05555556	3.36	0.0692

What conclusions can we draw from these results? First, we notice that the variables BRAND, MEAL, and GENDER are all significant effects (BRAND and MEAL at p = .0001, GENDER at p = .016). We see, from the Duncan test, that brand C is the preferred brand, followed by A and B. The fact that MEAL (breakfast or dinner) is significant and that BRAND*MEAL is not, tells us that all three brands of coffee are preferred after dinner.

G. *Three-factor Experiments with Repeated Measures on Two Factors*

As an example of a three-factor experiment with two repeated measures factors, we have designed a hypothetical study involving reading comprehension and a concept called slippage. It is well known that many students will do less well on a reading comprehension test in the early fall compared to the previous spring because of "slippage" during the summer vacation. As children grow older, the slippage should decrease. Also, slippage tends to be smaller with high-SES (socio-economic status—roughly speaking, "wealthier") children compared to low-SES children, since high-SES children typically do more reading over the summer.

To test these ideas, the following experiment was devised:

A group of high- and low-SES children is selected for the experiment. Their reading comprehension is tested each spring and fall for three consecutive years. A diagram of the design is shown below:

Reading Comprehension Scores

	Years:	1		2		3	
		SPRING	FALL	SPRING	FALL	SPRING	FALL
HIGH	subj						
SES	1	61	50	60	55	59	62
	2	64	55	62	57	63	63
	3	59	49	58	52	60	58
	4	63	59	65	64	67	70
	5	62	51	61	56	60	63
LOW	6	57	42	56	46	54	50
SES	7	61	47	58	48	59	55
	8	55	40	55	46	57	52
	9	59	44	61	50	63	60
	10	58	44	56	49	55	49

Notice that each subject is measured each spring and fall and each year so that the variables SEASON and YEAR are both repeated measures factors. In this design each subject belongs to either the high-SES or the low-SES group. Therefore, subjects are nested within SES.

We show three ways of writing a SAS program to analyze this experiment. First, using the REPEATED statement of PROC ANOVA:

```
DATA READ_1;
   INPUT SUBJ SES $ READ1-READ6;
   LABEL READ1 = 'SPRING YR 1'
         READ2 = 'FALL YR 1'
         READ3 = 'SPRING YR 2'
         READ4 = 'FALL YR 2'
         READ5 = 'SPRING YR 3'
         READ6 = 'FALL YR 3';
DATALINES;
1 HIGH 61 50 60 55 59 62
2 HIGH 64 55 62 57 63 63
3 HIGH 59 49 58 52 60 58
4 HIGH 63 59 65 64 67 70
5 HIGH 62 51 61 56 60 63
6 LOW  57 42 56 46 54 50
7 LOW  61 47 58 48 59 55
8 LOW  55 40 55 46 57 52
```


[Continued]

```
9 LOW  59 44 61 50 63 60
10 LOW 58 44 56 49 55 49
;

PROC ANOVA DATA=READ_1;
   TITLE 'Reading Comprehension Analysis';
   CLASS SES;
   MODEL READ1-READ6 = SES / NOUNI;
   REPEATED YEAR 3, SEASON 2;
   MEANS SES;
RUN;
```

Since the REPEATED statement is confusing when we have more than one repeated factor, we again show you how to determine the order of the factor names. The variables listed on the MODEL statement are in the following order:

```
    YEAR 1          YEAR 2          YEAR 3
SPRING  FALL    SPRING  FALL    SPRING  FALL
   1      2        3      4        5      6
```

There are three levels of YEAR and two levels of SEASON. The factors following the keyword REPEATED are placed in order from the one that varies the slowest to the one that varies the fastest. For example, the first number (READ1) is from YEAR 1 in the SPRING. The next number (READ2) is still YEAR 1 but in the FALL. Thus, we say that SEASON is varying faster than YEAR. We must also be sure to indicate the number of levels of each factor following the factor name on the REPEATED statement.

```
REPEATED YEAR 3, SEASON 2;
```

This statement instructs the ANOVA procedure to choose the first level of YEAR (1), then loop through two levels of SEASON (SPRING FALL), then return to the next level of YEAR (2), followed by two levels of SEASON, etc. The product of the two levels must equal the number of variables in the dependent variable list of the MODEL statement. To check, 3 * 2 = 6 and we have READ1 to READ6 on the MODEL statement.

Results of running this program are shown next (some sections omitted):

```
Reading Comprehension Analysis

Analysis of Variance Procedure
Class Level Information

Class   Levels   Values

SES          2   HIGH LOW
```

[Continued]

```
Number of observations in data set = 10

Repeated Measures Analysis of Variance
Repeated Measures Level Information

Dependent Variable  READ1  READ2  READ3  READ4  READ5  READ6

      Level of YEAR     1      1      2      2      3      3
    Level of SEASON     1      2      1      2      1      2

Tests of Hypotheses for Between Subjects Effects

Source            DF       Anova SS  F Value     Pr > F

SES                1    680.0666667    13.54     0.0062

Error              8    401.6666667

Univariate Tests of Hypotheses for Within Subject Effects

Source: YEAR
                                                      Adj  Pr > F
    DF     Anova SS  Mean Square  F Value  Pr > F   G - G   H - F
     2  252.033333   126.016667    26.91  0.0001  0.0002  0.0001

Source: YEAR*SES
                                                      Adj  Pr > F
    DF     Anova SS  Mean Square  F Value  Pr > F   G - G   H - F
     2    1.033333     0.516667     0.11  0.8962  0.8186  0.8700

Source: Error(YEAR)

    DF     Anova SS  Mean Square
    16   74.933333     4.683333

Greenhouse-Geisser Epsilon = 0.6757
       Huynh-Feldt Epsilon = 0.8658


Source: SEASON
                                                      Adj  Pr > F
    DF     Anova SS  Mean Square  F Value  Pr > F   G - G   H - F
     1  680.066667   680.066667   224.82  0.0001    .       .

Source: SEASON*SES
                                                      Adj  Pr > F
    DF     Anova SS  Mean Square  F Value  Pr > F   G - G   H - F
     1  112.066667   112.066667    37.05  0.0003    .       .
```


[Continued]

```
Source: Error(SEASON)

    DF     Anova SS  Mean Square
     8   24.200000     3.025000

Source: YEAR*SEASON
                                                      Adj  Pr > F
    DF     Anova SS  Mean Square  F Value  Pr > F   G - G   H - F
     2  265.433333   132.716667   112.95  0.0001  0.0001  0.0001

Univariate Tests of Hypotheses for Within Subject Effects

Source: YEAR*SEASON*SES
                                                      Adj  Pr > F
    DF     Anova SS  Mean Square  F Value  Pr > F   G - G   H - F
     2    0.433333     0.216667     0.18  0.8333  0.7592  0.8168

Source: Error(YEAR*SEASON)

    DF     Anova SS  Mean Square
    16   18.800000     1.175000

Greenhouse-Geisser Epsilon = 0.7073
       Huynh-Feldt Epsilon = 0.9221
                ----------   ----------   ----------   ----------
Level of              READ1                      READ2
SES      N        Mean          SD           Mean          SD

HIGH     5   61.8000000   1.92353841   52.8000000   4.14728827
LOW      5   58.0000000   2.23606798   43.4000000   2.60768096
                ----------   ----------   ----------   ----------
Level of              READ3                      READ4
SES      N        Mean          SD           Mean          SD

HIGH     5   61.2000000   2.58843582   56.8000000   4.43846820
LOW      5   57.2000000   2.38746728   47.8000000   1.78885438
                ----------   ----------   ----------   ----------
Level of              READ4                      READ5
SES      N        Mean          SD           Mean          SD

HIGH     5   61.8000000   3.27108545   63.2000000   4.32434966
LOW      5   57.6000000   3.57770876   53.2000000   4.43846820
```

We discuss the statistical results later, after two alternate programs have been presented.

We now present the other two programs that analyze this experiment without use of the REPEATED statement. Here is a second method: We have arranged our data so that each line represents one cell of our design. In practice, this would be tedious, but it will help you understand the last program for this problem in which all data for a subject are read on one line and the data set is transformed to look like this one.

```
DATA READ_2;
   INPUT SUBJ SES $ YEAR SEASON $ READ;
DATALINES;
1 HIGH 1 SPRING 61
1 HIGH 1 FALL 50
1 HIGH 2 SPRING 60
1 HIGH 2 FALL 55
1 HIGH 3 SPRING 59
1 HIGH 3 FALL 62
2 HIGH 1 SPRING 64
 (more data lines)
```

To simplify data entry (with the consequence of making the program more complicated) we can place all the data for each subject on one line. As we have mentioned before, since there is an alternative easier method (above), you may skip the more elaborate program below and not sacrifice anything in the way of statistical understanding:

```
*-------------------------------------------------------------*
| Alternative Program for reading in the data for the         |
| reading experiment with all the data for one subject on     |
| one line.                                                   |
*-------------------------------------------------------------*;
DATA READ_3;
   DO SES = 'HIGH','LOW';  ①
      SUBJ = 0;  ②
      DO N = 1 TO 5;  ③
        SUBJ + 1;  ④
        DO YEAR = 1 TO 3;  ⑤
           DO SEASON = 'SPRING','FALL';  ⑥
              INPUT SCORE @;  ⑦
              OUTPUT;  ⑧
           END;
        END;
      END;
   END;
   DROP N;  ⑨
DATALINES;
61 50 60 55 59 62
64 55 62 57 63 63
59 49 58 52 60 58
63 59 65 64 67 70
62 51 61 56 60 63
57 42 56 46 54 50
61 47 58 48 59 55
55 40 55 46 57 52
59 44 61 50 63 60
58 44 56 49 55 49
;
```

(NOTE: The indentation is not necessary. It is used as a visual aid to keep the DO loops straight.)

This program is not as complicated as it may seem at first glance. Our data will be arranged in the same order as they appear in the diagram of the experimental design. All high-SES students will be read followed by the low-SES students. Each student will have a spring/fall set of reading comprehension scores for each of the three years.

The data are arranged with all the high-SES students followed by all the low-SES students. The DO loop ① sets the values of SES appropriately. The use of character values in a DO loop may be new to you. This very useful feature of SAS software saves you the trouble of using numbers in DO loops and then formatting the numeric values to the character labels. Since there are five students in each SES group, the DO loop ③ goes from 1 to 5. The sum statement ④ will generate a subject number from 1 to 5 (it gets reset after all the data values for the HIGH-SES subjects have been read, statement ②). Since the order of the data for each subject is YEAR, SEASON, the two DO loops ⑤ and ⑥ set the values of YEAR and SEASON before reading in a score in line ⑦. The trailing at sign (@) in ⑦ is necessary to prevent the pointer from going to a new line until all six scores have been read. The OUTPUT statement ⑧ will output an observation with the variables:

```
SES SUBJ YEAR SEASON READ
```

Note that the variable N is not included because of the DROP statement ⑨. It is not necessary to drop N; we simply don't need it. We could leave it in the data set and just not use it.

One final note: Be careful when using character values with DO loops because if the length of the first value in the DO loop is shorter than the other levels, the program will truncate the length of the character variable to the first length it encounters. To avoid this problem, either pad the first value with blanks to be equal to the length of the longest value or use a LENGTH statement to define the length of the character variable.

This ends the discussion of the alternative program.

Now, regardless of the SAS data statements you used, the ANOVA statements will be the following:

```
PROC ANOVA DATA=READ_3;
   TITLE 'Reading Comprehension Analysis';
   CLASS SUBJ SES YEAR SEASON;
   MODEL SCORE = SES SUBJ(SES)
            YEAR SES*YEAR YEAR*SUBJ(SES)
            SEASON SES*SEASON SEASON*SUBJ(SES)
            YEAR*SEASON SES*YEAR*SEASON YEAR*SEASON*SUBJ(SES);
   MEANS YEAR / DUNCAN E=YEAR*SUBJ(SES);
   MEANS SES SEASON SES*YEAR SES*SEASON YEAR*SEASON
         SES*YEAR*SEASON;
   TEST H=SES                 E=SUBJ(SES);
   TEST H=YEAR SES*YEAR       E=YEAR*SUBJ(SES);
   TEST H=SEASON SES*SEASON   E=SEASON*SUBJ(SES);
   TEST H=YEAR*SEASON SES*YEAR*SEASON
                              E=YEAR*SEASON*SUBJ(SES);
RUN;
```

Output from this procedure is shown next:

```
     SOURCE                  F VALUE        PR > F
 ---------------------------------------------------------
     SES                      13.54          .0062
     YEAR                     26.91          .0001
     SEASON                  224.82          .0001
     SES*YEAR                   .11          .8962
     SES*SEASON               37.05          .0003
     YEAR*SEASON             112.95          .0001
     SES*YEAR*SEASON           0.18          .8333

     Next are the means for each of the main effects and two-way
 interactions:

 Level of          ----------READ----------
 SES        N        Mean             SD

 1         30    59.6000000     4.92425384
 2         30    52.8666667     6.23523543

 Level of          ----------READ----------
 SEASON     N        Mean             SD

 1         30    59.6000000     3.22276386
 2         30    52.8666667     7.26224689

 Level of Level of          ----------READ----------
 SES      YEAR       N       Mean             SD

 1        1         10    57.3000000     5.63816361
 1        2         10    59.0000000     4.13655788
 1        3         10    62.5000000     3.68932394
 2        1         10    50.7000000     8.02842174
 2        2         10    52.5000000     5.33853913
 2        3         10    55.4000000     4.45221543

 Level of Level of          ----------READ----------
 SES      SEASON     N       Mean             SD

 1        1         15    61.6000000     2.47270817
 1        2         15    57.6000000     5.96178305
 2        1         15    57.6000000     2.61315354
 2        2         15    48.1333333     5.06904706

 Level of Level of          ----------READ----------
 YEAR     SEASON     N       Mean             SD

 1        1         10    59.9000000     2.80673792
 1        2         10    48.1000000     5.93389510
 2        1         10    59.2000000     3.15524255
 2        2         10    52.3000000     5.71644800
 3        1         10    59.7000000     3.91719855
 3        2         10    58.2000000     6.69659947
```


As before, we specify hypotheses and error terms with TEST statements following our MODEL, and we include the appropriate error terms with the Duncan requests.

Have our original ideas about "slippage" been confirmed by the data?

First, let us examine each of the main effects and their interactions:

What conclusions can we draw from these results?

1. High-SES students have higher reading comprehension scores than low-SES students ($F = 13.54, p = .0062$).
2. Reading comprehension increases with each year ($F = 26.91, p = .0001$). However, this increase is due partly to the smaller "slippage" in the later years [see (5) below].
3. Students had higher reading comprehension scores in the spring compared to the following fall ($F = 224.82, p = .0001$).
4. The "slippage" was greater for the low-SES students (there was a significant SES*SEASON interaction $F = 37.05, p = .0003$).
5. "Slippage" decreases as the students get older (YEAR*SEASON is significant $F = 112.95, p = .0001$).

Repeated measures designs can be a powerful ally for the applied researcher. They can also be a little bit tricky. For example, in our coffee study, even though we randomized the order of first drinking the coffee with dinner or breakfast, there may be an effect we're overlooking. It may be that one (or all) of the brands take a little "getting used to." This could result in subjects preferring their second drinking of the coffee (whether breakfast or dinner). We are ignoring this in our study and maybe we shouldn't be. Had we not randomized which drinking came first, we would have confounded drinking order with MEAL. The best way to make sure that you are getting what you want out of a repeated measures design is to consult a text which deals solely with the design and statistical issues involved. (Winer does an excellent job of this.)

Problems

8-1. A marketing survey is conducted to determine sport shirt preference. A questionnaire is presented to a panel of four judges. Each judge rates the three shirts presented, of three brands (X, Y, and Z). The data entry form is shown below:

MARKETING SURVEY FORM

1. Judge ID	☐	1
2. Brand (1=X, 2=Y, 3=Z)	☐	2
3. Color rating 9=Best, 1=Worst	☐	3
4. Workmanship rating	☐	4
5. Overall preference	☐	5

An index is computed as follows:

```
INDEX=(3*OVERALL PREFERENCE + 2*WORKMANSHIP +
      COLOR RATING) / 6.0
```

The collected data follow:

```
11836
21747
31767
41846
12635
22534
32546
42436
13988
23877
33978
43887
```

Compare the color rating, workmanship, overall preference, and index among the three brands, using analysis of variance. (HINT: This is a repeated measures design—each judge rates all three brands.)

8-2. A taste test is conducted to determine which city has the best-tasting tap water. A panel of four judges tastes each of the samples from the four cities represented. The rating scale is a Likert scale with 1 = worst to 9 = best. Sample data and the coding scheme are shown below:

COLUMN	DESCRIPTION
1-3	Judge identification number
4	City code: 1 = New York, 2 = New Orleans 3 = Chicago, 4 = Denver
5	Taste rating, 1 = worst 9 = best

Data:

```
00118
00126
00138
00145
00215
00226
00235
00244
00317
00324
00336
00344
00417
00425
00437
00443
```


Write a SAS program to describe these data and to perform an analysis of variance. Treat the ratings as interval data. Remember that we have a repeated measures design.

***8-3.** The same data as in problem 8-2 are to be analyzed. However, they are arranged so that the four ratings from each judge are on one line. Thus, columns 1-3 are for the judge ID, column 4 is the rating for New York, column 5 for New Orleans, column 6 for Chicago, and column 7 for Denver. Our reformed data are shown below:

```
0018685
0025654
0037464
0047573
```

Write the DATA statements to analyze this arrangement of the data. You will need to create a variable for CITY and to have one observation per city. Also run these data using the REPEATED statement of PROC ANOVA. How do the two solutions compare?

***8-4.** A study is conducted to test the area of nerve fibers in NORMAL and DIABETIC rats. A sample from the DISTAL and PROXIMAL ends of each nerve fiber is measured for each rat. Therefore, we have GROUP (Normal versus CONTROL) and LOCATION (Distal versus proximal) as independent variables, with location as a repeated measure (each rat nerve is measured at each end of the nerve fiber). The data are shown below:

	RATNO	DISTAL	PROXIMAL
	1	34	38
Normal	2	28	38
	3	38	48
	4	32	38
	5	44	42
Diabetic	6	52	48
	7	46	46
	8	54	50

Write a SAS program to enter these data and run a two-way analysis of variance, treating the location as a repeated measure. Use the REPEATED option for the LOCATION variable. Is there any difficulty in interpreting the main effects? Why?

8-5. What's wrong with this program?

```
DATA FINDIT;
   DO GROUP='CONTROL','DRUG';
      DO TIME='BEFORE','AFTER';
         DO SUBJ=1 TO 3;
            INPUT SCORE @;
         END;
      END;
   END;
DATALINES;
10 13 15 20 (data for subject 1) Order is CONTROL TIME 1,
12 14 16 18 (data for subject 2) CONTROL TIME 2, DRUG TIME 1,
15 18 22 28 (data for subject 3) and DRUG TIME 2
;
```

[Continued]

```
PROC ANOVA DATA=FINDIT;
   TITLE 'ANALYSIS OF VARIANCE';
   CLASS SUBJ GROUP TIME;
   MODEL SCORE = GROUP SUBJ(GROUP)
      TIME GROUP*TIME TIME*SUBJ(GROUP);
   TEST H=GROUP E=SUBJ(GROUP);
   TEST H=TIME GROUP*TIME E=TIME*SUBJ(GROUP);
   MEANS GROUP|TIME;
RUN;
```

NOTE: The comments within parentheses are not part of the program.

CHAPTER

9

Multiple-Regression Analysis

A. Introduction
B. Designed Regression
C. Nonexperimental Regression
D. Stepwise and Other Variable Selection Methods
E. Creating and Using Dummy Variables
F. Logistic Regression

A. *Introduction*

Multiple-regression analysis is a method for relating two or more independent variables to a dependent variable. While the dependent variable (the variable you want to predict) must be a continuous variable (except with logistic regression), the independent variables may either be continuous or categorical variables such as "gender" or "type of medication." In the case of categorical independent variables, we need to create "dummy" variables rather than using the actual character values (more on this later). If all of your independent variables are categorical (or most of them) you may be better off using analysis of variance techniques.

There are two rather distinct uses of multiple regression, and they will be addressed separately. The first use is for studies where the levels of the independent variables have been experimentally controlled (such as amount of medication and number of days between dosages). This use will be referred to as "designed regression." The second use involves settings where a sample of subjects have been observed on a number of naturally occurring variables (age, income, level of anxiety, etc.) which are then related to some outcome of interest. This use of regression will be referred to as "nonexperimental regression."

It is fairly easy to misuse regression. We will try to note some popular pitfalls, but we cannot list them all. A rule of thumb is to use your common sense. If the results of an analysis don't make any sense, get help. Ultimately, statistics is a tool employed to help us understand life. Although understanding life can be tricky, it is not usually perverse. Before accepting conclusions which seem silly based on statistical analyses, consult with a veteran data analyst. Most truly revolutionary results from data analyses are based on data entry errors.

B. Designed Regression

Imagine a researcher interested in the effects of scheduled exercise and the use of a stimulant for weight loss. She constructs an experiment using 24 college sophomores where four levels of stimulant and three levels of exercise are used. There are 24 subjects in the experiment and each is randomly assigned to a level of exercise and stimulant such that two students are in each of the 12 (3 × 4) possible combinations of exercise and stimulant. After 3 weeks of participation, a measure of weight loss (post − pre weight) is obtained for each subject. The data for the experiment might look as shown below:

Data for Weight Loss Experiment

Subject	Stimulant (mg/day)	Exercise (hr/week)	Weight Loss (pounds)
1	100	0	−4
2	100	0	0
3	100	5	−7
4	100	5	−6
5	100	10	−2
6	100	10	−14
7	200	0	−5
8	200	0	−2
9	200	5	−5
10	200	5	−8
11	200	10	−9
12	200	10	−9
13	300	0	1
14	300	0	0
15	300	5	−3
16	300	5	−3
17	300	10	−8
18	300	10	−12
19	400	0	−5
20	400	0	−4
21	400	5	−4
22	400	5	−6
23	400	10	−9
24	400	10	−7

These data can be analyzed either as a 3 × 4 analysis of variance, or as a two-variable multiple regression. The regression approach typically assumes that the effects of exercise and medication increase linearly (i.e., in a straight line); the ANOVA model makes no such assumption. If we use the multiple-regression approach, the following program will provide the desired results:


```
DATA REGRESSN;
   INPUT ID DOSAGE EXERCISE LOSS;
DATALINES;
1 100 0  -4
2 100 0  0
3 100 5  -7
(more data lines)
;
PROC REG DATA=REGRESSN;
   TITLE 'Weight Loss Experiment - Regression Example';
   MODEL LOSS = DOSAGE EXERCISE / P R;
RUN;
QUIT;
```

The first three lines create the data set. PROC REG performs a wide variety of regression models. The MODEL statement indicates that the dependent variable (the one to the left of the equals sign) is LOSS, and the two independent (or predictor) variables (the ones to the right of the equals sign) are DOSAGE and EXERCISE. The options "P" and "R" specify that we want predicted values and residuals to be computed. Note the use of a QUIT statement at the end of this procedure. PROC REG, as well as PROC ANOVA and PROC GLM, are considered to be "interactive" procedures. That is, after a RUN statement has been encountered and the procedure has executed, you may submit additional statements (new models, for example). The top line of the SAS Display Manager will continue to show the procedure "Running" until a new procedure is submitted or until a QUIT statement is submitted. The use of a QUIT statement is therefore, optional.

The output from this program is presented below:

```
Weight Loss Experiment - Regression Example

Model: MODEL1
Dependent Variable: LOSS

Analysis of Variance

                       Sum of          Mean
Source       DF       Squares        Square      F Value       Prob>F

Model         2     162.97083      81.48542       11.185       0.0005
Error        21     152.98750       7.28512
C Total      23     315.95833

    Root MSE        2.69910        R-square       0.5158
    Dep Mean       -5.45833        Adj R-sq       0.4697
    C.V.          -49.44909
```

[Continued]

Parameter Estimates

Variable	DF	Parameter Estimate	Standard Error	T for H0: Parameter=0	Prob > |T|
INTERCEP	1	-2.562500	1.50884052	-1.698	0.1042
DOSAGE	1	0.001167	0.00492785	0.237	0.8151
EXERCISE	1	-0.637500	0.13495480	-4.724	0.0001

Obs	Dep Var LOSS	Predict Value	Std Err Predict	Residual	Std Err Residual	Student Residual	-2-1-0	1 2	Cook's D
1	-4.0000	-2.4458	1.142	-1.5542	2.445	-0.636	*		0.029
2	0	-2.4458	1.142	2.4458	2.445	1.000		**	0.073
3	-7.0000	-5.6333	0.922	-1.3667	2.537	-0.539	*		0.013
4	-6.0000	-5.6333	0.922	-0.3667	2.537	-0.145			0.001
5	-2.0000	-8.8208	1.142	6.8208	2.445	2.789		*****	0.566
6	-14.0000	-8.8208	1.142	-5.1792	2.445	-2.118	****		0.326
7	-5.0000	-2.3292	0.905	-2.6708	2.543	-1.050	**		0.047
8	-2.0000	-2.3292	0.905	0.3292	2.543	0.129			0.001
9	-5.0000	-5.5167	0.604	0.5167	2.631	0.196	*		0.001
10	-8.0000	-5.5167	0.604	-2.4833	2.631	-0.944			0.016
11	-9.0000	-8.7042	0.905	-0.2958	2.543	-0.116			0.001
12	-9.0000	-8.7042	0.905	-0.2958	2.543	-0.116			0.001
13	1.0000	-2.2125	0.905	3.2125	2.543	1.263		**	0.067
14	0	-2.2125	0.905	2.2125	2.543	0.870		*	0.032
15	-3.0000	-5.4000	0.604	2.4000	2.631	0.912		*	0.015
16	-3.0000	-5.4000	0.604	2.4000	2.631	0.912		*	0.015
17	-8.0000	-8.5875	0.905	0.5875	2.543	0.231			0.002
18	-12.0000	-8.5875	0.905	-3.4125	2.543	-1.342	**		0.076
19	-5.0000	-2.0958	1.142	-2.9042	2.445	-1.188	**		0.103
20	-4.0000	-2.0958	1.142	-1.9042	2.445	-0.779	*		0.044
21	-4.0000	-5.2833	0.922	1.2833	2.537	0.506		*	0.011
22	-6.0000	-5.2833	0.922	-0.7167	2.537	-0.283			0.004
23	-9.0000	-8.4708	1.142	-0.5292	2.445	-0.216			0.003
24	-7.0000	-8.4708	1.142	1.4708	2.445	0.601		*	0.026

Sum of Residuals	0
Sum of Squared Residuals	152.9875
Predicted Resid SS (Press)	212.0359

NOTE: The output is truncated somewhat so that it fits conveniently on the page width.

The output begins with an analysis of variance table, which looks much as it would from a standard ANOVA. We can see that there are two degrees of freedom in this model, one for EXERCISE and one for DOSAGE. There is only one degree of freedom for each since the regression estimates a single straight line for each variable rather than estimating a number of cell means.

The sum of squares for the model (162.971) tells us how much of the variation in weight loss is attributable to EXERCISE and DOSAGE. The mean square for the model (81.485) is the sum of squares (162.971) divided by the degrees of freedom for the model (2). This mean square is then divided by the mean square error (7.285) to produce the F-statistic for the regression (11.185). The p-value for this is reported as .0005. "C TOTAL" means "corrected total" and indicates the total degrees of freedom (23) and sum of squares (315.958) in the dependent variable. In regression, the corrected total degrees of freedom is always one less than the total sample size since one degree of freedom is used to estimate the grand mean. The ROOT MSE (2.699) stands for the square root of the mean square error and represents, in standard deviation units, the variation in the system not attributable to EXERCISE or DOSAGE. DEP Mean (–5.458) is simply the mean of the dependent variable (LOSS). The R-SQUARE (.5158) is the square of the multiple correlation of EXERCISE and DOSAGE with LOSS. It is the proportion of variance in LOSS explained by (attributable to) the independent variables. ADJ R-SQ (.4697) is the adjusted R-square. The adjusted R-square takes into account how many variables were used in the equation and slightly lowers the estimate of explained variance. C.V. (–49.449) stands for coefficient of variation and is calculated by dividing the ROOT MSE by the mean, and multiplying by 100. The C.V. is sometimes useful when the mean and standard deviation are related (such as in income data).

The bottom part of the output shows us, observation by observation, the actual LOSS, the predicted value and the difference between the two (residual). In addition, the column labeled "Student Residuals," expresses the residual as a t-score and Cook's D is a distance measure that helps us determine how strongly a particular data point affects the overall regression. Large absolute values of D (2 or more) indicate possible problems with your model or data points that require some careful scrutiny.

Having explained the terms in the analysis of variance table for the regression, let's summarize what meaning we can infer. Basically, the table indicates that the independent variables were related to the dependent variable (since the F was significant at p=.0005). Furthermore, we find that about 50% of the variation in weight loss is explained by the two experimental treatments. Many researchers are more interested in the R-square statistic than in the p-value since the R-square represents an estimate of how strongly related the variables were. The bottom half of the printout contains the estimates of the parameters of the regression equation. Three parameters are estimated: (1) the intercept, or constant, term (2) the coefficient for DOSAGE, and (3) the coefficient for EXERCISE. Each parameter estimate was based on one degree of freedom (always the case in regressions). For each parameter estimate, a standard error was estimated along with a t-statistic and a p-value for the t-statistic. The t-statistic is simply the parameter estimate divided by its standard error, and it is based on the number of degrees of freedom for the error term (21 for this example).

This half of the printout tells us that it was really EXERCISE that caused the weight loss. The regression coefficient for DOSAGE is not statistically significantly different from zero (p=.8151). The fact that the intercept was not significantly different from zero is irrelevant here. The intercept merely tells us where the regression line (or plane, in this case) crosses the y-axis, and does not explain any variation.

At this point, many researchers would run a new regression with DOSAGE eliminated, to refine the estimate of EXERCISE. Since this was a designed experiment, we would recommend leaving the regression as is for purposes of reporting. Dropping DOSAGE won't affect the estimated impact of EXERCISE since DOSAGE and EXERCISE are uncorrelated (by design). When the independent variables in a regression are uncorrelated, the estimates of the regression coefficients are unchanged by adding or dropping independent variables. When the independent variables are correlated, dropping or adding variables strongly affects the regression estimates and hypothesis tests.

C. *Nonexperimental Regression*

Many, if not most, regression analyses are conducted on data sets where the independent variables show some degree of correlation. These data sets, resulting from nonexperimental research, are common in all fields. Studies of factors affecting heart disease or the incidence of cancer, studies relating student characteristics to student achievement, and studies predicting economic trends all utilize nonexperimental data. The potential for a researcher to be misled by a nonexperimental data set is high; for a novice researcher, it is near certainty. We strongly urge consultation with a good text in this area (Pedhazur's *Multiple Regression in Behavioral Research* is excellent) or with a statistical consultant. Having made this caveat, let's venture into regression analysis for nonexperimental data sets.

The Nature of the Data. There are many surface similarities between experimental and nonexperimental data sets. First, there are one or more outcome or dependent variables. Second, there are several independent variables (sometimes quite a few). The basic difference here is that the independent variables are correlated with one another. This is because in nonexperimental studies one defines a population of interest (people who have had heart attacks, sixth grade students, etc.), draws a sample, and measures the variables of interest. The goal of the study is usually to explain variation in the dependent variable by one or more of the independent variables. So far it sounds simple.

The problem is that correlation among the independent variables causes the regression estimates to change depending on which independent variables are being used. That is, the impact of B on A depends on whether C is in the equation or not. With C omitted, B can look very influential. With C included, the impact of B can disappear completely! The reason for this is as follows: A regression coefficient tells us the unique contribution of an independent variable to a dependent variable. That is, the coefficient for B tells us what B contributes all by itself with no overlap with any other variable. If B is the only variable in the equation, this is no problem. But if we add C, and if B and C are correlated, then the unique contribution of B on A will be changed. Let's see how this works in an example.

The subjects are a random sample of sixth grade students from Metropolitan City School District. The following measures have been taken on the subjects:

1. ACH6: Reading achievement at the end of sixth grade.
2. ACH5: Reading achievement at the end of fifth grade.

3. APT: A measure of verbal aptitude taken in the fifth grade.
4. ATT: A measure of attitude toward school taken in fifth grade.
5. INCOME: A measure of parental income (in thousands of dollars per year).

Our data set is listed below. (NOTE: These are not actual data.)

ID	ACH6	ACH5	APT	ATT	INCOME
1	7.5	6.6	104	60	67
2	6.9	6.0	116	58	29
3	7.2	6.0	130	63	36
4	6.8	5.9	110	74	84
5	6.7	6.1	114	55	33
6	6.6	6.3	108	52	21
7	7.1	5.2	103	48	19
8	6.5	4.4	92	42	30
9	7.2	4.9	136	57	32
10	6.2	5.1	105	49	23
11	6.5	4.6	98	54	57
12	5.8	4.3	91	56	29
13	6.7	4.8	100	49	30
14	5.5	4.2	98	43	36
15	5.3	4.3	101	52	31
16	4.7	4.4	84	41	33
17	4.9	3.9	96	50	20
18	4.8	4.1	99	52	34
19	4.7	3.8	106	47	30
20	4.6	3.6	89	58	27

The purpose of the study is to understand what underlies the reading achievement of the students in the district. The following program was written to analyze the data:

```
DATA NONEXP;
   INPUT ID ACH6 ACH5 APT ATT INCOME;
DATALINES;
1 7.5 6.6 104 60 67
2 6.9 6.0 116 58 29
(more data lines)
;
PROC REG DATA=NONEXP;
   TITLE 'Nonexperimental Design Example';
   MODEL ACH6 = ACH5 APT ATT INCOME /
         SELECTION = FORWARD;
   MODEL ACH6 = ACH5 APT ATT INCOME /
         SELECTION = MAXR;
RUN;
QUIT;
```

By using the SELECTION=FORWARD option of PROC REG, we have specified that a forward regression is to be run with ACH6 as the dependent variable, and ACH5, APT, ATT, and INCOME as independent variables. Each variable will be tested, and the one that produces the largest F-value will be entered first (if the p-value for entry is less than the specified or default value). Since we did not specify a criteria for entry into the model, the default value of .50 is used. If we want to change the p-value for entry into the model, we can include the MODEL option SLENTRY= our p-value. Although we did not choose to show it, we could run a stepwise regression (where variables that enter the model can also leave the model later) where we can specify both entry level p-value (with SLENTRY= our p-value) and a p-value for staying in the model (SLSTAY= our-p-value). For selection method STEPWISE, the default entry and staying p-values are both .15.

We also want to run the model using the MAXR technique. Before examining the output, we should discuss briefly stepwise regression and nonexperimental data.

D. Stepwise and Other Variable Selection Methods

As mentioned earlier, with nonexperimental data sets, the independent variables are not truly "independent" in that they are usually correlated with one another. If these correlations are moderate to high (say 0.50 and above), then the regression coefficients are greatly affected by that particular subset of independent variables that are in the regression equation. If there are a number of independent variables to consider, coming up with the best subset can be difficult. Variable selection methods, including stepwise, were developed to assist researchers in arriving at this optimal subset. Unfortunately, many of these methods are frequently misused. The problem is that the solution from a purely statistical point of view is often not the best from a substantive perspective. That is, a lot of variance is explained but the regression doesn't make much sense and isn't very useful. We'll discuss this more when we examine the printout.

Stepwise regression examines a number of different regression equations. Basically, the goal of stepwise techniques is to take a set of independent variables and put them into a regression one at a time in a specified manner until all variables have been added or until a specified criterion has been met. The criterion is usually one of statistical significance or the improvement in the explained variance.

SAS software allows for a number of variable selection techniques. Among them are:

1. FORWARD: Starts with the best single regressor, then finds the best one to add to what exists; the next best, etc.
2. BACKWARD: Starts with all variables in the equation, then drops the worst one, then the next, etc.
3. STEPWISE: Similar to FORWARD except that there is an additional step where all variables in each equation are checked again to see if they remain significant after the new variable has been entered.
4. MAXR: A rather complicated procedure, but basically it tries to find the one-variable regression with the highest r-square, then the two-variable regression with the highest r-square, etc.
5. MINR: Very similar to the MAXR, except that the selection system is slightly different.

Now, let's examine the printout from the program:

```
Nonexperimental Design Example

Forward Selection Procedure for Dependent Variable ACH6

Step 1 Variable ACH5 Entered  R-square = 0.66909805  C(p) = 1.87549647

               DF     Sum of Squares    Mean Square          F     Prob>F

Regression      1        12.17624633    12.17624633      36.40     0.0001
     Error     18         6.02175367     0.33454187
     Total     19        18.19800000

            Parameter         Standard          Type II
Variable     Estimate            Error   Sum of Squares        F   Prob>F

INTERCEP   1.83725236       0.71994457       2.17866266     6.51   0.0200
ACH5       0.86756297       0.14380353      12.17624633    36.40   0.0001

Bounds on condition number:          1,              1
------------------------------------------------------------------------------
Step 2 Variable APT Entered   R-square = 0.70817380    C(p) = 1.76460424

              DF          Sum of Squares      Mean Square        F     Prob>F

Regression     2             12.88734675       6.44367337    20.63     0.0001
Error         17              5.31065325       0.31239137
Total         19             18.19800000

            Parameter         Standard          Type II
Variable     Estimate            Error   Sum of Squares        F   Prob>F

INTERCEP   0.64269963       1.05397972       0.11615840     0.37   0.5501
ACH5       0.72475202       0.16813652       5.80435251    18.58   0.0005
APT        0.01824901       0.01209548       0.71110042     2.28   0.1497

Bounds on condition number:  1.463985,      5.855938
------------------------------------------------------------------------------

No other variable met the 0.5000 significance level for entry into the model.

Summary of Forward Selection Procedure for Dependent Variable ACH6

        Variable  Number    Partial      Model
Step    Entered       In       R**2       R**2     C(p)           F     Prob>F

  1     ACH5           1     0.6691     0.6691   1.8755     36.3968     0.0001
  2     APT            2     0.0391     0.7082   1.7646      2.2763     0.1497
```

[Continued]

Maximum R-square Improvement for Dependent Variable ACH6

Step 1 Variable ACH5 Entered R-square = 0.66909805 C(p) = 1.87549647

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	1	12.17624633	12.17624633	36.40	0.0001
Error	18	6.02175367	0.33454187		
Total	19	18.19800000			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	1.83725236	0.71994457	2.17866266	6.51	0.0200
ACH5	0.86756297	0.14380353	12.17624633	36.40	0.0001

Bounds on condition number: 1, 1

The above model is the best 1-variable model found.

Step 2 Variable APT Entered R-square = 0.70817380 C(p) = 1.76460424

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	2	12.88734675	6.44367337	20.63	0.0001
Error	17	5.31065325	0.31239137		
Total	19	18.19800000			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	0.64269963	1.05397972	0.11615840	0.37	0.5501
ACH5	0.72475202	0.16813652	5.80435251	18.58	0.0005
APT	0.01824901	0.01209548	0.71110042	2.28	0.1497

Bounds on condition number: 1.463985, 5.855938

The above model is the best 2-variable model found.

Step 3 Variable ATT Entered R-square = 0.71086255 C(p) = 3.61935632

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	3	12.93627670	4.31209223	13.11	0.0001
Error	16	5.26172330	0.32885771		
Total	19	18.19800000			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	0.80013762	1.15586303	0.15758855	0.48	0.4987
ACH5	0.74739939	0.18222852	5.53198290	16.82	0.0008
APT	0.01972808	0.01298905	0.75861687	2.31	0.1483
ATT	-0.00797735	0.02068119	0.04892995	0.15	0.7048

Bounds on condition number: 1.633564, 14.15998

[Continued]

The above model is the best 3-variable model found.

Step 4 Variable INCOME Entered R-square = 0.72232775 C(p) = 5.00000000

	DF	Sum of Squares	Mean Square	F	Prob>F
Regression	4	13.14492048	3.28623012	9.76	0.0004
Error	15	5.05307952	0.33687197		
Total	19	18.19800000			

Variable	Parameter Estimate	Standard Error	Type II Sum of Squares	F	Prob>F
INTERCEP	0.91164562	1.17841159	0.20161506	0.60	0.4512
ACH5	0.71373964	0.18932981	4.78747493	14.21	0.0019
APT	0.02393740	0.01419278	0.95826178	2.84	0.1124
ATT	-0.02115577	0.02680560	0.20983199	0.62	0.4423
INCOME	0.00898581	0.01141792	0.20864378	0.62	0.4435

Bounds on condition number: 2.431593, 31.79315

The above model is the best 4-variable model found.

No further improvement in R-square is possible.

Since a forward selection was requested first, that is what was run first. In step 1, the technique picked ACH5 as the first regressor since it had the highest correlation with the dependent variable ACH6. The r-square (variance explained) is 0.669, which is quite high. "C_p" is a statistic used in determining how many variables to use in the regression. You will need to consult one of the references in Chapter 1 or see your friendly statistician for help in interpreting Mallow's C_p statistic. The remaining statistics are the same as for the PROC REG program run earlier. On step 2, the technique determined that adding APT would lead to the largest increase in r-square. We notice however, that r-square has only moved from 0.669 to 0.708, a slight increase. Furthermore, the regression coefficient for APT is nonsignificant (p=.1497). This indicates that APT doesn't tell us much more than we already knew from ACH5. Most researchers would drop it from the model and use the one-variable (ACH5) model. After step 2 has been run, the forward technique indicates that no other variable would come close to being significant. In fact, no other variable would have a p-value less than .50 (we usually require less than .05, although in regression analysis, it is not uncommon to set the inclusion level at .10).

The MAXR approach finds the best one-variable model, then the best two-variable model, etc., until the full model (all variables included) is estimated. As can be seen, with these data, both of these techniques lead to the same conclusions: ACH5 is far and away the best predictor; it is a strong predictor; and no other variables would be included with the possible exception of APT.

There is a problem here, however. Any sixth grade teacher could tell you that the best predictor of sixth grade performance is fifth grade performance. But it doesn't really tell us very much else. It might be more helpful to look at APT, ATT, and INCOME without ACH5 in the regression. Also, it could be useful to make ACH5 the dependent variable and have APT, ATT, and INCOME be regressors. Of course, this is suggesting quite a bit in the way of regressions. There is another regression technique which greatly facilitates looking at a large number of possibilities quickly. This is the RSQUARE selection method of PROC REG. The RSQUARE method will give us the multiple r-square value for every one, two, three, . . . , n way combinations of the variables in the independent variable list. The following lines will generate all of the regressions mentioned so far as well as the model with ACH5 as the dependent variable:

```
PROC REG DATA=NONEXP;
   MODEL ACH6  = INCOME ATT APT ACH5 / SELECTION=RSQUARE;
   MODEL ACH5  = INCOME ATT APT / SELECTION=RSQUARE;
RUN;
```

The output from PROC REG with RSQUARE the selection option is shown next:

```
N = 20       Regression Models for Dependent Variable: ACH6

Number in      R-square      Variables in Model
  Model

    1        0.66909805      ACH5
    1        0.38921828      APT
    1        0.18113085      ATT
    1        0.10173375      INCOME
-----------------------------------------
    2        0.70817380      APT ACH5
    2        0.66964641      INCOME ACH5
    2        0.66917572      ATT ACH5
    2        0.45629702      INCOME APT
    2        0.40687404      ATT APT
    2        0.18564520      INCOME ATT
-----------------------------------------
    3        0.71086255      ATT APT ACH5
    3        0.71079726      INCOME APT ACH5
    3        0.66967022      INCOME ATT ACH5
    3        0.45925077      INCOME ATT APT
-----------------------------------------
    4        0.72232775      INCOME ATT APT ACH5
-------------------------------------------------

N = 20       Regression Models for Dependent Variable: ACH5

Number in      R-square      Variables in Model
  Model

    1        0.31693268      APT
    1        0.26115761      ATT
    1        0.13195687      INCOME
-----------------------------------------
```



```
[Contiuued]
-----------------------------------------
    2        0.41273318      INCOME APT
    2        0.38784142      ATT APT
    2        0.26422291      INCOME ATT
-----------------------------------------
    3        0.41908115      INCOME ATT APT
-----------------------------------------
```

The top part contains all of the RSQUARE's for every possible one-, two-, three-, and four-variable regression with ACH6 as the outcome variable. It is possible to glean quickly, a lot of information from this table.

Let us say that you have just decided that you don't want ACH5 as a regressor. You can see quickly from the one-variable regressions that APT is the next best regressor (r-square=.389). The next question is, "What is the best two-variable regression?" and "Is the improvement large enough to be worthwhile?" Let's look at the two-variable regressions which have APT in them:

```
R-square for     APT + ATT = .407
                 APT + INCOME = .456
```

(Remember, we're eliminating ACH5 for now.)

APT and INCOME is best, and the gain is .067 (which is equal to 0.456 – 0.389). Is a 6.7% increase in variance explained worth including? Probably it is, although it may not be statistically significant with our small sample size. In explaining the regressions using ACH5 as an outcome variable, we can see that APT and INCOME look like the best bet there also. In interpreting these data, we might conclude that aptitude combined with parental wealth are strong explanatory variables in reading achievement. It is important to remember that statistical analyses must make substantive sense. The question arises here as to how these two variables work to influence reading scores. Some researchers would agree that APT is a psychological variable and INCOME is a sociological variable and the two shouldn't be mixed in a single regression. It's a bit beyond the scope of this book to speculate on this, but when running nonexperimental regressions, it is best to be guided by these two principles:

1. Parsimony: Less is more in terms of regressors. Another regressor will always explain a little bit more, but it often confuses our understanding of life.
2. Common Sense: The regressors must bear a logical relationship to the dependent variable in addition to a statistical one. (ACH6 would be a great predictor of ACH5, but it is a logical impossibility.)

Finally, whenever regression analysis is used, the researcher should examine the simple correlations among the variables. The statements below will generate a correlation matrix among all the variables of interest:

```
PROC CORR DATA=NONEXP NOSIMPLE;
   VAR APT ATT ACH5 ACH6 INCOME;
RUN;
```

The output from running this procedure is:

```
Pearson Correlation Coefficients / Prob > |R| under Ho: Rho=0
/ N = 20

            APT        ATT        ACH5       ACH6       INCOME

APT       1.00000    0.49741    0.56297    0.62387    0.09811
          0.0        0.0256     0.0098     0.0033     0.6807

ATT       0.49741    1.00000    0.51104    0.42559    0.62638
          0.0256     0.0        0.0213     0.0614     0.0031

ACH5      0.56297    0.51104    1.00000    0.81798    0.36326
          0.0098     0.0213     0.0        0.0001     0.1154

ACH6      0.62387    0.42559    0.81798    1.00000    0.31896
          0.0033     0.0614     0.0001     0.0        0.1705

INCOME    0.09811    0.62638    0.36326    0.31896    1.00000
          0.6807     0.0031     0.1154     0.1705     0.0
```

An examination of the simple correlations often leads to a better understanding of the more complex regression analyses. Here we can see why ATT, which shows a fairly good correlation with ACH6, was never included in a final model. It is highly related to INCOME (r = .626) and also to APT (r = .497). Whatever relationship it had with ACH6 was redundant with APT and INCOME. Also notice that INCOME and APT are unrelated, which contributes to their inclusion in the model. The simple correlations also protect against misinterpretation of "suppressor variables." These are a little too complex to discuss here. However, they can be spotted when a variable is not significantly correlated with the dependent variable but in the multiple regression has a significant regression coefficient (usually negative). You should get help (from Pedhazur or from another text or a consultant) with interpreting such a variable.

E. Creating and Using Dummy Variables

As we mentioned in the introduction to this chapter, categorical variables such as gender or race can be used as independent variables in a multiple regression, provided you first create dummy variables. When a categorical variable has only two levels, such as gender, this is very easy. You code the dummy variable as either FEMALE or NOT FEMALE (or MALE versus NOT MALE). It is traditional to use the values of 0 and 1 for these dummy variables. So, if you have a variable called GENDER with values of 'F' and 'M' you would create a dummy variable with the following statements:

```
IF GENDER = 'F' THEN DUMMY_G = 1;
   ELSE IF GENDER = 'M' THEN DUMMY_G = 0;
```

When this dummy variable is used in a regression, the coefficient of DUMMY_G will show how much to add or subtract (if the coefficient is negative)

from the predicted value of the dependent variables if the subject is a female. The reason for this is that we have chosen the 0 level for males which makes the males the "referrence" level (anything times 0 is still 0). A compact SAS statement to create DUMMY_G would be:

```
DUMMY_G = INPUT (TRANSLATE(GENDER,'01','MF'),1.);
```

See Chapter 17 for an explanation of the INPUT function and Chapter 18 for an explanation of the TRANSLATE function. This statement is not preferrable to the two lines above (we actually prefer the two-line method) but it is useful if you want to impress your boss!

What do you do when your independent categorical variable has more than two levels? You choose one of the levels as your "reference" and create k–1 dummy variables where k is the number of levels for your categorical variable. As an example, suppose a variable RACE has levels of 'WHITE', 'AFRICAN AM', and 'HISPANIC'. Arbitrarily choosing WHITE as the referrence level, you would create two dummy variables, one representing African American or not; the other representing Hispanic or not. Here is one way to code this:

```
IF RACE = 'AFRICAN AM' THEN AF_AM = 1;
   ELSE AF_AM = 0;
IF RACE = 'HISPANIC' THEN HISPANIC = 1;
   ELSE HISPANIC = 0;
```

This code assumes that if the value of RACE is missing, the dummy variables will be set to 0. Instead, you may want to set the dummy variables to missing when RACE is missing. To do this, modify the lines above like this:

```
IF RACE = 'AFRICAN  AM' THEN AF_AM = 1;
   ELSE IF RACE NE ' ' THEN AF_AM = 0;
IF RACE = 'HISPANIC' THEN HISPANIC = 1;
   ELSE IF RACE NE ' ' THEN HISPANIC = 0;
```

You may find it impractical to create dummy variables for categorical variables with a large number of values.

F. Logistic Regression

When you have a dependent variable with only two levels (such as dead/alive; sick/well), multiple-regression techniques are not appropriate. Suppose you coded your dependent variable as a 1 for SICK and a 0 for WELL. You would like the regression equation to predict a number between 0 and 1 which could be interpreted as the probability that the subject was sick or well. However, using the multiple-regression methods described in the sections above, the prediction equation could result in negative values or values greater than 1. This would be difficult to interpret.

A regression method called logistic regression was developed to handle this problem. Logistic regression uses a transformation (called a logit) which forces the prediction equation to predict values between 0 and 1. A logistic regression equation predicts the natural log of the odds for a subject being in one category or another. In addition, the regression coefficients in a logistic regression equation can be used to estimate odds ratios for each of the independent variables.

Although the details of logistic regression are beyond the scope of this book, we will demonstrate several ways to run logistic regression with the following data set.

We have recorded the age, vision status, driver education status, and accident status (did the subject have an accident in the past year?) of a number of individuals. The sample data, stored in a data set called C:\APPLIED\ACCIDENT.DTA is listed below (0 = No, 1 = Yes):

Accident Statistics Based on Age, Vision, and Driver's Education:

Accident in Past Year?	Age	Vision Status	Driver Education
1	17	1	1
1	44	0	0
1	48	1	0
1	55	0	0
1	75	1	1
0	35	0	1
0	42	1	1
0	57	0	0
0	28	0	1
0	20	0	1
0	38	1	0
0	45	0	1
0	47	1	1
0	52	0	0
0	55	0	1
1	68	1	0
1	18	1	0
1	68	0	0
1	48	1	1
1	17	0	0
1	70	1	1
1	72	1	0
1	35	0	1
1	19	1	0
1	62	1	0
0	39	1	1
0	40	1	1
0	55	0	0
0	68	0	1
0	25	1	0
0	17	0	0
0	45	0	1
0	44	0	1
0	67	0	0
0	55	0	1

[Continued]

Accident in Past Year?	Age	Vision Status	Driver Education
1	61	1	0
1	19	1	0
1	69	0	0
1	23	1	1
1	19	0	0
1	72	1	1
1	74	1	0
1	31	0	1
1	16	1	0
1	61	1	0

Our aim is to see if age, vision status, and driver education can be used to predict if the subject had an accident in the past year. Below is a program to create a SAS data set called LOGISTIC which includes the variables listed above plus three new variables (AGEGROUP, YOUNG, and OLD) which will be used in later sections. The statements to run a forward stepwise logistic regression are included:

```
*-----------------------------------------------------------*
| Program Name: LOGISTIC.SAS in C:\APPLIED                  |
| Purpose: To demonstrate logistic regression               |
| Date: June 6, 1996                                        |
*-----------------------------------------------------------*;
PROC FORMAT;
   VALUE AGEGROUP 0 = ' < =20 and <=65'
                  1 = ' <20 or >65';
   VALUE VISION   0 = 'No Problem'
                  1 = 'Some Problem';
   VALUE YES_NO   0 = 'No'
                  1 = 'Yes';

RUN;

DATA LOGISTIC;
   INFILE 'C:\APPLIED\ACCIDENT.DTA' MISSOVER;
   INPUT ACCIDENT AGE VISION DRIVE_ED;
   ***Note: No missing ages;
   IF AGE < 20 OR AGE > 65 THEN AGEGROUP = 1;
   ELSE AGEGROUP=0;
   IF AGE < 20 THEN YOUNG = 1;
   ELSE YOUNG = 0;
   IF AGE > 65 THEN OLD = 1;
   ELSE OLD = 0;

   LABEL ACCIDENT = 'Accident in Last Year?'
         AGE      = 'Age of Driver'
         VISION   = 'Vision Problem?'
         DRIVE_ED = 'Driver Education?';
```

[Continued]

```
   FORMAT ACCIDENT
          DRIVE_ED
          YOUNG
          OLD       YES_NO.
          AGEGROUP  AGEGROUP.
          VISION    VISION.;
RUN;

PROC LOGISTIC DATA=LOGISTIC DESCENDING;
   TITLE 'Predicting Accidents Using Logistic Regression';
   MODEL ACCIDENT = AGE VISION DRIVE_ED /
                    SELECTION = FORWARD
                    CTABLE PPROB = (0 to 1 by .1)
                    LACKFIT
                    RISKLIMITS;
RUN;
QUIT;
```

The DATA Step statements are straightforward. Let's explain the PROC LOGISTIC statements. One somewhat peculiar "feature" of PROC LOGISTIC is that the resulting equation predicts the log odds for the LOWER value of the dependent variable. So, if we follow tradition and code Yes as a 1 and No as a 0, the equation would predict the log odds of NOT having an accident. One easy way to reverse this is to use the option DESCENDING. In our example, the use of this option will cause the program to predict the log odds of having an accident given a certain set of predictor or explanatory values. (Another way is to use the options ORDER=FORMATTED and provide a format with values in the correct direction.)

Next, our MODEL statement looks just like the ones we used with PROC REG. This logistic regression example includes several MODEL options: The selection method is chosen to be FORWARD (the same as for regular regression); a classification table (CTABLE) is requested for all probabilities from 0 to 1 by .1; the Hosmer and Lemeshow Goodness-of-Fit test (LACKFIT); and the Odds Ratios for each variable in the equation with their 95% confidence limits (RISKLIMITS) are requested. Here are the results:

```
Predicting Accidents Using Logistic Regression

The LOGISTIC Procedure

Data Set: WORK.LOGISTIC
Response Variable: ACCIDENT Accident in Last Year?
Response Levels: 2
Number of Observations: 45
Link Function: Logit
```

[Continued]

```
      Response Profile ①

Ordered
  Value   ACCIDENT    Count

      1   Yes            25
      2   No             20


Forward Selection Procedure ②

Step 0.  Intercept entered:

Residual Chi-Square = 10.7057 with 3 DF (p=0.0134)

Step 1. Variable VISION entered:

Model Fitting Information and Testing Global Null Hypothesis BETA=0

                            Intercept
             Intercept         and
Criterion       Only       Covariates  Chi-Square for Covariates

AIC            63.827         59.244     .
SC             65.633         62.857     .
-2 LOG L       61.827         55.244     6.583 with 1 DF (p=0.0103)
Score            .              .        6.421 with 1 DF (p=0.0113)

Residual Chi-Square = 4.9818 with 2 DF (p=0.0828)

Predicting Accidents Using Logistic Regression

The LOGISTIC Procedure

Step 2. Variable DRIVE_ED entered:

Model Fitting Information and Testing Global Null Hypothesis BETA=0

                            Intercept
             Intercept         and
Criterion       Only       Covariates  Chi-Square for Covariates

AIC            63.827         56.287     .
SC             65.633         61.707     .
-2 LOG L       61.827         50.287   11.539 with 2 DF (p=0.0031)
Score            .              .      10.598 with 2 DF (p=0.0050)

Residual Chi-Square = 0.1293 with 1 DF (p=0.7191)

NOTE: No (additional) variables met the 0.05 significance
      level for entry into the model.
```

[Continued]

Summary of Forward Selection Procedure ③

Step	Variable Entered	Number In	Score Chi-Square	Pr > Chi-Square	Variable Label
1	VISION	1	6.4209	0.0113	Vision Problem?
2	DRIVE_ED	2	4.8680	0.0274	Driver Education?

Analysis of Maximum Likelihood Estimates ④

Variable	DF	Parameter Estimate	Standard Error	Wald Chi-Square	Pr > Chi-Square	Standardized Estimate
INTERCPT	1	0.1110	0.5457	0.0414	0.8388	.
VISION	1	1.7139	0.7049	5.9120	0.0150	0.477689
DRIVE_ED	1	-1.5001	0.7037	4.5447	0.0330	-0.417273

Association of Predicted Probabilities and Observed Responses ⑤

Concordant	= 67.2%	Somers' D	= 0.532
Discordant	= 14.0%	Gamma	= 0.655
Tied	= 18.8%	Tau-a	= 0.269
(500 pairs)		c	= 0.766

Conditional Odds Ratios and 95% Confidence Intervals ⑥

Variable	Unit	Odds Ratio	Wald Confidence Limits Lower	Upper
VISION	1.0000	5.551	1.394	22.098
DRIVE_ED	1.0000	0.223	0.056	0.886

Hosmer and Lemeshow Goodness-of-Fit Test ⑦

Group	Total	ACCIDENT = Yes Observed	Expected	ACCIDENT = No Observed	Expected
1	11	2	2.20	9	8.80
2	11	6	5.80	5	5.20
3	10	6	5.80	4	4.20
4	13	11	11.20	2	1.80

Goodness-of-fit Statistic = 0.0756 with 2 DF (p=0.9629)

[Continued]

Classification Table ⑧

Prob Level	Correct Event	Correct Non-Event	Incorrect Event	Incorrect Non-Event	Percentages Correct	Sensi-tivity	Speci-ficity	False POS	False NEG
0.000	25	0	20	0	55.6	100.0	0.0	44.4	.
0.100	25	0	20	0	55.6	100.0	0.0	44.4	.
0.200	23	0	20	2	51.1	92.0	0.0	46.5	100.0
0.300	23	9	11	2	71.1	92.0	45.0	32.4	18.2
0.400	23	9	11	2	71.1	92.0	45.0	32.4	18.2
0.500	17	9	11	8	57.8	68.0	45.0	39.3	47.1
0.600	11	14	6	14	55.6	44.0	70.0	35.3	50.0
0.700	11	18	2	14	64.4	44.0	90.0	15.4	43.8
0.800	11	18	2	14	64.4	44.0	90.0	15.4	43.8
0.900	0	18	2	25	40.0	0.0	90.0	100.0	58.1
1.000	0	20	0	25	44.4	0.0	100.0	.	55.6

Explanation of the Output. Let's examine the salient sections of this output. First is the "Response Profile" ① which lists the number of observations in each category of the outcome variable (ACCIDENT). Pay careful attention to this, especially the ordered value information. Because we used the DESCENDING option on the PROC LOGISTIC statement, the value of 1 (formatted as 'Yes') is first in the list of ordered values. As we mentioned before, this means that this logistic model will be predicting the odds and probabilities of having an accident based on the explanatory variables.

The next section shows the order that the independent or explanatory variables entered the model ②. We see VISION entered first, with several criteria for assessing the importance of this variable in predicting accidents. The two criteria "–2 LOG L" and "Score" are both used to test whether the independent variable(s) is significant, based on a chi-squared distribution. We see that VISION is a significant explanatory variable using either of these two criteria (p approximately .01). The other two criteria, "AIC" (Akaike Information Criterion) and "SC" (Schwartz Criterion) serve a similar purpose except they adjust for the number of explanatory variables and the number of observations used in the model. These statistics are useful for comparing different models; lower values of these statistics indicate a better-fitting model.

Looking farther down the output, we see that DRIVE_ED (driver education) entered next. The overall model improved (based on a lower AIC and SC and a smaller p-value for –2 LOG L). Since no other variables met the default entry level significance of .05, the model building stopped at this point.

The "Summary" section ③ is printed only for stepwise (FORWARD, BACKWARD, or STEPWISE) selection methods. It summarizes the order in which the explanatory variables entered the model and the chi-square and p-value for each variable.

Section ④ gives us the parameter estimates for the logistic regression model. In this example, the equation is:

$$\log(\text{odds of having an accident}) = .1110 + 1.7139 \times \text{VISION} - 1.5001 \times \text{DRIVE\_ED}$$

If we substitute values for VISION and DRIVE_ED into this equation, the results are the log (odds) of having an accident. To determine the odds, raise e (the base of common logorithms) to this power. To compute the probability that a person will have an accident, based on vision and driver education, you can use the relationship that:

$$\text{Odds} = \frac{P}{1 - P} \quad \text{where P is the probability.}$$

Solving for P, we get:

$$P = \frac{\text{Odds}}{1 + \text{Odds}}.$$

Let's use this equation to predict the odds and the probability of a person having an accident for given values of VISION and DRIVE_ED. For a person with no vision problem (VISION=0) and who never took a driver education course (DRIVE_ED=0), the calculation would be:

$$\log(\text{odds}) = .1110 + 1.7139 \times 0 - 1.5001 \times 0 = .1110.$$

Therefore, the odds of having an accident for this person are:

$$\text{Odds (of having an accident)} = \exp(.1110) = 1.1174.$$

And the probability of having an accident is:

$$p\text{ (having an accident)} = \frac{1.1174}{1 + 1.1174} = .5277.$$

Taking a similar person, except one with a vision problem (VISION=1), we again compute odds and probabilities:

$$\log(\text{odds}) = .1110 + 1.7139 \times 1 - 1.5001 \times 0 = 1.8249,$$

$$\text{Odds (of having an accident)} = \exp(1.8249) = 6.2022,$$

$$p\text{ (having an accident)} = \frac{6.2022}{1 + 6.2022} = .8612.$$

You can see that the odds of having an accident increase dramatically (from 1.1174 to 6.2022) when a person has a vision problem. We often look at the ratio of these odds,

$$\frac{6.2022}{1.1174} = 5.5506,$$

to describe the effect of an explanatory variable on the odds for an event. This ratio is called the odds ratio and is shown in a later section of the output.

The section labeled "Association of Predicted Probabilities" ⑤ is somewhat complicated. It works this way: Take all possible pairs of observations in which the outcomes are not the same. In this example, there are 500 pairs where one unit of the pair had an accident and the other did not. Then compute the probability of each unit of the pair having an accident. If the unit with the higher computed probability is the one that actually experienced the event in question (an accident), this pair is labeled "Concordant." When a pair is in the "wrong" order, it is labeled "Discordant." If both units of the pair have the same probability, the pair is labeled "Tied." It is desirable to have a high concordant percentage and a low discordant percentage.

The "Conditional Odds Ratios and 95% Confidence Intervals" ⑥ is the result of the RISKLIMITS option. For each variable, it lists the odds ratio and the 95% confidence interval for this ratio. Notice that the odds ratio for VISION is 5.551, which is the same as we computed earlier (if we round our result). Since the 95% confidence interval does not contain one, we have additional confirmation that vision is a significant explanatory variable in our model.

The fact that the odds ratio for DRIVE_ED is less than 1 tells us that driver education helps reduce accidents.

The "Hosmer and Lemeshow Goodness-of-Fit" statistics ⑦ is a chi-square based test to assess goodness of fit. Since you probably do not want to reject the null hypothesis that your data fit the specified model, you would like a high p-value for this test. In this example, the chi-square value of .0756 with two degrees of freedom gives us a p-value of .9629, which means that we do not reject the null hypothesis that these data fit this model.

We finally get to the "Classification Table" ⑧, which gives us the sensitivity, specificity, false positive rate, and false negative rate for several levels of probability. Suppose, for example, if you decide that any predicted probability greater than .3 should be considered a "positive diagnosis" for having an accident. In other words, you want to be somewhat conservative and consider a person an accident risk even though the probability of having an accident is less than .5. Based on the classification table, this cutoff for a "positive diagnosis" would have a high sensitivity (92%) and a relatively low specificity (45%). "Sensitivity," for those not familiar with the term, is the proportion of people who have the event in question (an accident) and are predicted to have one ($p > .3$ in this case). Specificity would be the proportion of people who did not have an accident and who had a probability less than .3. Looking at this table, you can decide what a convenient cutoff for a "positive diagnosis" might be, depending on your desired sensitivity and specificity. We will discuss a graphical way of looking at this later in this section, when we show you how to produce a receiver operator characteristic curve (ROC).

Creating a Categorical Variable from AGE. Either by inspection of the data or by experience, you may be surprised to find that age did not enter into the equation. To investigate this further, let's look at the age distributions for those who had accidents and those who did not. A simple PROC CHART can quickly do this for us. Here is the code:

```
OPTIONS PS=30;
PROC CHART DATA=LOGISTIC;
   TITLE 'Distribution of Ages by Accident Status';
   VBAR AGE / MIDPOINTS=10 TO 90 BY 10
              GROUP=ACCIDENT;
RUN;
```

Here is the resulting output:

```
Distribution of Ages by Accident Status

Frequency

  8 +                          *
    |                          *
  7 +                          *         *
    |                          *         *
  6 +       *                  *         *
    |       *                  *         *
  5 +       *                  *         *
    |       *                  *         *
  4 +       * * *              *       * *
    |       * * *              *       * *
  3 +       * * *              *       * *
    |       * * *              *       * *
  2 +   * * * * * *            *   * * * *
    |   * * * * * *            *   * * * *
  1 +   * * * * * *            * * * * * * *
    |   * * * * * *            * * * * * * *
    --------------------------------------------
      1 2 3 4 5 6 7 8 9     1 2 3 4 5 6 7 8 9   AGE Midpoint
      0 0 0 0 0 0 0 0 0     0 0 0 0 0 0 0 0 0

      |-------No-------|    |------Yes-------|   ACCIDENT
```

Notice that for the nonaccident group, there are more subjects in the center of the distribution but in the accident group, there seems to be an excess of very young and older individuals. Based on this finding, we can create a new variable (AGEGROUP) which will have a value of 0 for subjects between 20 and 65 (inclusive) and a value of 1 otherwise. With foresight, we already created this variable in the original data set. We can therefore use the new variable AGEGROUP instead of AGE and rerun the regression. Here is the modified program with the added option to create an output data set which will contain the sensitivity and 1-specificity (the false positive rate) so we can plot an ROC (receiver operator characteristic curve) later on:


```
PROC LOGISTIC DATA = LOGISTIC DESCENDING;
   TITLE 'Predicting Accidents Using Logistic Regression';
   MODEL ACCIDENT = AGEGROUP VISION DRIVE_ED /
                    SELECTION = FORWARD
                    CTABLE PPROB =(0 to 1 by .1)
                    LACKFIT
                    RISKLIMITS
                    OUTROC=ROC;
RUN;
QUIT;
OPTIONS LS=64 PS=32;
PROC PLOT DATA=ROC;
   TITLE 'ROC Curve';
   PLOT _SENSIT_ * _1MSPEC_ = 'o';
   LABEL _SENSIT_ = 'Sensitivity'
         _1MSPEC_ = '1 - Specificity';
RUN;
```

The PROC LOGISTIC statements are basically the same as before except that we substituted AGEGROUP for AGE and included the MODEL option OUTROC= to create an output data set (ROC) with the data necessary to plot an ROC curve. We then used PROC PLOT to plot the ROC curve. For a nicer looking graph, PROC GPLOT, part of the SAS Graph® package, could be used. Here are some edited portions of the output from running the procedures above:

Predicting Accidents Using Logistic Regression

The LOGISTIC Procedure

Summary of Forward Selection Procedure

Step	Variable Entered	Number In	Score Chi-Square	Pr > Chi-Square	Variable Label
1	AGEGROUP	1	9.3750	0.0022	
2	VISION	2	5.3447	0.0208	Vision Problem?

Analysis of Maximum Likelihood Estimates

Variable	DF	Parameter Estimate	Standard Error	Wald Chi-Square	Pr > Chi-Square	Standardized Estimate
INTERCPT	1	-1.3334	0.5854	5.1886	0.0227	.
AGEGROUP	1	2.1611	0.8014	7.2712	0.0070	0.590289
VISION	1	1.6258	0.7325	4.9265	0.0264	0.453130

```
[Continued]

  Conditional Odds Ratios and 95% Confidence Intervals

                                      Wald
                                Confidence Limits
                         Odds
Variable      Unit       Ratio     Lower      Upper

AGEGROUP     1.0000      8.680     1.805     41.756
VISION       1.0000      5.083     1.209     21.359

ROC Curve

        Plot of _SENSIT_*_1MSPEC_.  Symbol used is 'o'.

     1.0 +
         |                                                o
         |
 S       |
 e       |
 n 0.8 +                       o
 s       |
 i       |
 t       |
 i       |
 v       |
 i       |
 t 0.6 +         o
 y       |
         |
         |
         |o
     0.4 +
         -+---------+---------+---------+---------+---------+-
         0.0       0.2       0.4       0.6       0.8       1.0
                            1-Specificity

NOTE: 2 obs hidden.
```

What a difference! Now AGEGROUP enters first, followed by VISION. See how important it is to look at and understand your data before jumping in with both feet and running procedures.

The ROC curve is a traditional method for showing the relationship between sensitivity and the false positive rate. The variables _SENSIT_ and _1MSPEC_ in the output data set (ROC) represent the sensitivity and one minus the specificity (false positive rate) respectively. As mentioned earlier, we can arbitrarily decide what value to call a positive prediction—it doesn't have to be .5. You could declare any value greater than .3 to be a positive prediction. This would increase your sensitivity (anyone who actually had an accident would likely be predicted to be positive) but increase your false positive rate (many who did not have accidents would also be predicted to have one). As you can see, one minus the specificity gives us the false positive rate. (Well, maybe you can't see it. Unless you work with these definitions often, it is very easy to get confused.)

Creating Two Dummy Variables from AGE. For our final trick, let's create two dummy variables from AGE, one called YOUNG, which will be true (1) for anyone less than 20 years old, and false (0) otherwise. In a similar manner, the variable OLD will be defined as true for people over 65, and false otherwise. The code to create these two additional variables is included in the original DATA Step. The PROC LOGISTIC statements to run an analysis based on the use of these new variables are:

```
PROC LOGISTIC DATA = LOGISTIC DESCENDING;
   TITLE 'Predicting Accidents Using Logistic Regression';
   TITLE 'Using Two Dummy Variables (YOUNG and OLD) for AGE';
   MODEL ACCIDENT = YOUNG OLD VISION DRIVE_ED /
                    SELECTION = FORWARD
                    CTABLE PPROB=(0 to 1 by .1)
                    LACKFIT
                    RISKLIMITS
                    OUTROC=ROC;
RUN;
QUIT;
```

Unfortunately, because of the fairly small sample size, there aren't enough subjects in the young and old age groups so that these two variables are not included in the model. However, with a larger data set, this approach may be preferable to the AGEGROUP approach used earlier since the odds ratios for being young and old can be determined separately.

Problems

9-1. We want to test the effect of light level and amount of water on the yield of tomato plants. Each potted plant receives one of three levels of light (1 = 5 hours, 2 = 10 hours, 3 = 15 hours) and one of two levels of water (1 = 1 quart, 2 = 2 quarts). The yield, in pounds, is recorded. The results are as follows:

Yield	Light	Water	Yield	Light	Water
12	1	1	20	2	2
9	1	1	16	2	2
8	1	1	16	2	2
13	1	2	18	3	1
15	1	2	25	3	1
14	1	2	20	3	1
16	2	1	25	3	2
14	2	1	27	3	2
12	2	1	29	3	2

Write a SAS program to read these data, and perform a multiple regression.

9-2. We would prefer to estimate the number of books in a college library without counting them. Data are collected from colleges across the country of the number of volumes, the student enrollment (in thousands), the highest degree offered (1 = B.A., 2 = M.A., 3 = Ph.D.), and size of the main campus (in acres). Results of this (hypothetical) study are displayed below:

Books (millions)	Student Enrollment (in thousands)	Degree	Area (acres)
4	5	3	20
5	8	3	40
10	40	3	100
1	4	2	50
.5	2	1	300
2	8	1	400
7	30	3	40
4	20	2	200
1	10	2	5
1	12	1	100

Using a forward stepwise regression, show how each of the three factors affects the number of volumes in a college library. Treat DEGREE as a continuous variable.

9-3. We want to predict a student's success in college by a battery of tests. Graduating seniors volunteer to take our test battery, and their final grade point average is recorded. Using a MAXR technique, develop a prediction equation for final grade point average using the test battery results. The data are as follows:

GPA	HS GPA	College Board	IQ Test
3.9	3.8	680	130
3.9	3.9	720	110
3.8	3.8	650	120
3.1	3.5	620	125
2.9	2.7	480	110
2.7	2.5	440	100
2.2	2.5	500	115
2.1	1.9	380	105
1.9	2.2	380	110
1.4	2.4	400	110

9-4. Take a sample of 25 people and record their height, waist measurement, length of their right leg, length of their arm, and their weight. Write a SAS program to create a SAS data set of these data, and compute a correlation matrix of these variables. Next, run a stepwise multiple regression using weight as the dependent variable and the other variables as independent.

9-5. What's wrong with this program?

```
DATA MULTREG;
   INPUT HEIGHT WAIST LEG ARM WEIGHT;
DATALINES;
(data lines)
;
PROC CORR DATA = MULTREG;
   VAR HEIGHT -- WEIGHT;
```


[Continued]

```
RUN;
PROC REG DATA=MULTREG;
   MODEL WEIGHT = HEIGHT WAIST LEG ARM / SELECTION=STEPWISE;
RUN;
```

9-6. Repeat problem 9-2 except treat DEGREE as a categorical variable. You will need to create two dummy variables. Use the B.A. degree as your reference level.

9-7. Accident data, similar to Section F, are presented below. This time, we recorded accidents that occurred in the past year, based on the presence of a drinking problem and whether the driver had one or more accidents in the previous year. Run a forward stepwise logistic regression on these data, and write the resulting logistic regression equation. Compute the odds and the probability of an accident for two cases: (1) a person with no drinking problem or a previous accident; (2) a person with a drinking problem but no previous accidents. Take the ratio of the odds for person (2) divided by person (1) and confirm that the odds ratio is the same as listed in the SAS output (use the MODEL option RL to obtain the risk limits). Here are the data:

Accident Statistics Based on Drinking and Accident History

(1 = Yes, 0 = No)

Accident in Past Year	Drinking Problem	Previous Accident	Accident in Past Year	Drinking Problem	Previous Accident
1	0	1	1	1	0
1	1	1	1	1	1
1	1	1	0	1	1
1	0	0	0	1	1
1	1	1	0	0	1
0	0	1	0	0	1
0	1	0	0	1	0
0	1	0	0	0	0
0	0	1	0	0	1
0	0	0	0	1	0
0	1	0	0	0	0
0	0	1	0	0	0
0	1	0	1	1	1
0	0	0	1	1	0
0	0	0	1	0	1
1	1	0	1	1	1
1	0	1	1	1	0
1	1	1	1	1	1
1	1	1	1	1	0
1	0	1	1	1	1
1	1	1	1	1	1
1	1	0	1	1	1
1	0	1			

CHAPTER

10

Factor Analysis

A. *Introduction*

Welcome to the wacky world of factor analysis, a family of statistical techniques developed to allow researchers to reduce a large number of variables, such as questions on a questionnaire, to a smaller number of broad concepts called factors. The factors can then be used in subsequent analyses. But beware. Factor analysis can be fairly seductive in that it holds out the promise of taking a large number of baffling variables and turning them into a clear-cut set of just a few factors.

B. *Types of Factor Analysis*

Before proceeding, a word or two on different types of factor analysis. Generally speaking, there are two types of factor analysis: exploratory and confirmitory. SAS does either, but we are only going to discuss exploratory here as confirmatory is too complex for the scope of this book. Within exploratory factor analysis, there are also two basic approaches: principal components analysis and factor analysis. Factor analysis' purists get a little crazed over the use of principal components analysis and some even refuse to consider it as a part of factor analysis. We are going to show you a principal components analysis, then a more proper factor analysis of the same data set, show you how to rotate the results (doesn't that sound fun), and then talk about interpretation of the results. By the way, Pedhazur and Schmelkin (1991) is a very readable presentation of factor analysis for beginners.

C. *Principal Components Analysis*

We start with principal components analysis since it is conceptually somewhat simpler than factor analysis. This does not mean that we are recommending it; it just allows for a better pedagogical flow. Imagine you are trying to develop a new measure of depression and paranoia (how pleasant). Your measure contains six questions. For each question, the subject is to respond using the following Likert scale:

1 = Very Strongly Disagree
2 = Strongly Disagree
3 = Disagree
4 = No Opinion
5 = Agree
6 = Strongly Agree
7 = Very Strongly Agree

The six questions are:

1. I usually feel blue.
2. People often stare at me.
3. I think that people are following me.
4. I am usually happy.
5. Someone is trying to hurt me.
6. I enjoy going to parties.

As stated, this example was created with two psychological problems in mind: depression and paranoia. Someone who is depressed will likely agree with questions 1 and 6 and disagree with question 3. Someone who is paranoid will probably agree with questions 2, 4, and 5. Therefore, we would expect the factor analysis to come up with two factors. One we can label depression, the other, paranoia.

Some sample data are shown below:

	Question					
SUBJ	**1**	**2**	**3**	**4**	**5**	**6**
1	7	2	3	4	5	6
2	6	3	2	1	3	2
3	3	6	7	3	6	3
4	2	2	2	5	3	4
5	3	4	2	4	2	3
6	6	3	4	2	3	2
7	1	2	3	7	2	2
8	3	3	2	3	4	3
9	2	1	1	6	2	5
10	6	2	3	2	2	2
11	3	5	4	2	3	3
12	6	7	6	2	6	2
13	5	1	1	2	6	2
14	2	1	1	6	1	5
15	1	2	1	7	1	7

Assume that we place these data in a file called FACTOR.DTA with the subject number in columns 1 and 2 and the six questions in columns 3 through 8 (i.e., no spaces between any values). Some people say that you should have about 10 times the number of subjects as you have variables to be factor analyzed. However, you probably want a minimum of 50 subjects and do not want huge numbers of variables. We violate that rule for this simple example where we have six variables and 15 subjects.

Our first step is to create a SAS data set containing the responses to the six questions, as shown below:

```
*-------------------------------------------------------------*
|Program Name: FACTOR.SAS in C:\APPLIED                        |
|Purpose: To perform a factor analysis on psychological Data   |
*-------------------------------------------------------------*;
PROC FORMAT;
  VALUE LIKERT
   1 = 'V. Strong Dis.'
   2 = 'Strongly Dis.'
   3 = 'Disagree'
   4 = 'No Opinion'
   5 = 'Agree'
   6 = 'Strongly Agree'
   7 = 'V. Strong Agree';
RUN;

DATA FACTOR;
   INFILE 'C:\APPLIED\FACTOR.DTA' PAD;
   INPUT SUBJ 1-2 @3 (QUES1-QUES6) (1.);

LABEL   QUES1='Feel Blue'
        QUES2='People Stare at Me'
        QUES3='People Follow Me'
        QUES4='Basically Happy'
        QUES5='People Want to Hurt Me'
        QUES6='Enjoy Going to Parties';
RUN;
```

The INPUT statement in the example above uses a list of variables in parentheses (QUES1-QUES6), followed by an informat list. The informat "1." means one column for each of the six responses. If you prefer, you may separate each data item from the next by a space, and use the free form or "list" input method. As always, there are several ways to accomplish our goal with SAS software. Now, back to our factor analysis example. The SAS statements to perform the factor analysis are:

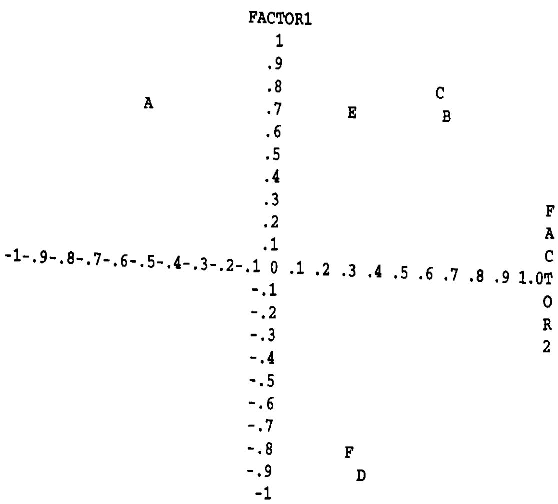
```
PROC FACTOR DATA=FACTOR PREPLOT PLOT ROTATE=VARIMAX
            NFACTORS=2 OUT=FACT SCREE;
   TITLE 'Example of Factor Analysis';
   VAR QUES1-QUES6;
RUN;
```


We have selected several of the more popular PROC FACTOR options in this example. The PREPLOT option will show us a factor plot before rotation; PLOT will produce a factor plot after rotation; ROTATE=VARIMAX requests a Varimax rotation; NFACTORS=2 specifies that a maximum of two factors should be extracted; OUT=FACT specifies that you want the factor scores (in this case FACTOR1 and FACTOR2) to be placed in the named data set along with all the variables in the DATA= data set. The SCREE option gives you a Scree plot (discussed later). Here is the output from PROC FACTOR:

```
Example of Factor Analysis

Initial Factor Method: Principal Components

Prior Communality Estimates: ONE ①

Eigenvalues of the Correlation Matrix: Total = 6 Average = 1

                      1          2          3     ②
Eigenvalue         3.6683     1.2400     0.5313
Difference         2.4283     0.7087     0.1873
Proportion ③       0.6114     0.2067     0.0886
Cumulative         0.6114     0.8180     0.9066

                      4          5          6
Eigenvalue         0.3440     0.1547     0.0617
Difference         0.1893     0.0931
Proportion         0.0573     0.0258     0.0103
Cumulative         0.9639     0.9897     1.0000

2 factors will be retained by the NFACTOR criterion.

Initial Factor Method: Principal Components

Scree Plot of Eigenvalues ④

      4 +
        |           1
  E     |
  i     |
  g     |
  e     |
  n     |
  v   2 +
  a     |
  l     |
  u     |                2
  e     |
  s     |                     3
        |                          4     5
      0 +                                     6
        ------+-----+-----+-----+-----+-----+-----+
              0     1     2     3     4     5     6

                          Number
```

[Continued]

```
Initial Factor Method: Principal Components ⑤

Factor Pattern

            FACTOR1     FACTOR2

QUES1       0.76843    -0.54767    Feel Blue
QUES2       0.72985     0.59840    People Stare at Me
QUES3       0.77904     0.50692    People Follow Me
QUES4      -0.87354     0.36879    Basically Happy
QUES5       0.72583     0.26237    People Want to Hurt Me
QUES6      -0.80519     0.34660    Enjoy Going to Parties

Variance explained by each factor ⑥

 FACTOR1   FACTOR2
3.668279  1.239976

Final Communality Estimates: Total = 4.908255

   QUES1     QUES2     QUES3     QUES4     QUES5     QUES6
0.890425  0.890758  0.863868  0.899078  0.595662  0.768463

Initial Factor Method: Principal Components

Plot of Factor Pattern for FACTOR1 and FACTOR2 ⑦
```




```
QUES1  =A   QUES2  =B   QUES3  =C   QUES4  =D   QUES5  =E
QUES6  =F
```


[Continued]

```
Rotation Method: Varimax ⑧

Orthogonal Transformation Matrix

              1           2
1      -0.73625     0.67671
2       0.67671     0.73625

Rotated Factor Pattern

            FACTOR1     FACTOR2

QUES1      -0.93637     0.11677    Feel Blue
QUES2      -0.13241     0.93446    People Stare at Me
QUES3      -0.23053     0.90040    People Follow Me
QUES4       0.89271    -0.31960    Basically Happy
QUES5      -0.35684     0.68434    People Want to Hurt Me
QUES6       0.82737    -0.28969    Enjoy Going to Parties

Variance explained by each factor

  FACTOR1   FACTOR2
 2.556277  2.351977

Rotation Method: Varimax

Final Communality Estimates: Total = 4.908255

   QUES1     QUES2     QUES3     QUES4     QUES5     QUES6
0.890425  0.890758  0.863868  0.899078  0.595662  0.768463

Scoring Coefficients Estimated by Regression

Squared Multiple Correlations of the Variables with each Factor

  FACTOR1   FACTOR2
 1.000000  1.000000

Standardized Scoring Coefficients ⑨

            FACTOR1     FACTOR2

QUES1      -0.45312    -0.18343    Feel Blue
QUES2       0.18008     0.48994    People Stare at Me
QUES3       0.12029     0.44471    People Follow Me
QUES4       0.37659     0.05783    Basically Happy
QUES5      -0.00249     0.28968    People Want to Hurt Me
QUES6       0.35076     0.05726    Enjoy Going to Parties
```

[Continued]

```
Plot of Factor Pattern for FACTOR1 and FACTOR2 ⑩
                              FACTOR1
                                 1
                        D       .9
                         F      .8
                                .7
                                .6
                                .5
                                .4
                                .3
                                .2                          F
                                .1                          A
                                                            C
-1 -.9-.8-.7-.6-.5-.4-.3-.2-.1 0 .1 .2 .3 .4 .5 .6 .7 .8 .9 1.0T
                                                            O
                               -.1                          R
                               -.2                     B    2
                               -.3                    C
                               -.4               E
                               -.5
                               -.6
                               -.7
                               -.8
                               -.9
                               -1  A

QUES1   =A   QUES2   =B   QUES3   =C   QUES4   =D   QUES5   =E
QUES6   =F
```

Well, that all looks fairly confusing. Let's walk through what we have. The first thing we learn is that our prior communality estimates were one ①. This means that each variable was given one full unit of variance to be factored in the original correlation matrix. Many factor analysts argue that one should only factor the variance that is shared among the variables. When the prior communalities equal one, the analysis factors all of the variance in all variables and the approach is called principal components analysis. More on this later.

The second thing we encounter are the eigenvalues of the correlation matrix ②. Eigenvalue is a term from matrix algebra. In this analysis, since we let each variable have one unit of variance, and since there are six variables, the total of the eigenvalues equals six. As mentioned earlier, factor analysis tries to reduce variables into a smaller set of factors which "explain" the variance in the original variables. Of the original "6 units" of variance, 3.6683 of these units are explained by the first factor (or more properly, component, since this is a principal components analysis). The second factor explains 1.2400 units of variance, and so on.

It is useful to think about eigenvalues in the following fashion. Imagine the first factor is a new variable, and everyone in the data set had a score on it. We could correlate their score on the factor with their score on each of the six variables. If we then squared those correlations, it would tell us how much of the variance in each original variable was explained by the factor. If we then added those six squared correlations together, the sum would be the eigenvalue for that factor. Thus, the first factor explains 3.6683 units of variance, or 3.6683/6 = .6114, of the total variance of the original six variables. This .6114 figure is listed in the printout under "Proportion ③." The "Difference" heading tells us the difference in the proportion explained from the previous total, and the "Cumulative" row gives the cumulative total.

The next thing we see is that two factors will be retained for rotation because that is what we told the program to do.

The next section of printout is the "scree plot ④." The scree plot is used to help determine how many factors to keep in the analysis. It is simply a plot of the eigenvalue against the number of the factor. One looks for breaks, or "elbows," in the curve. In this graph, it is easy to see that factors 1 and 2 are very different from 3-6, so two factors should be retained. In our example, we knew from a theoretical perspective that we wanted two factors. The results support our notion. In determining the number of factors to keep, it is always best to combine theory and data. If you don't have a strong theory to rely on beforehand, you will have to do one run just to get an idea of how many factors to keep.

The next section ⑤ presents the initial solution for the analysis. Oversimplified, what the factor analysis tries to do is first find a factor (think of it as a new variable) which will provide the highest set of correlations with the original variables (actually, with the squares of these variables) thus producing the largest eigenvalue. Then it finds a second factor which will correlate as highly as possible with the original variables once the variation from the first factor has been removed. Another way of thinking about this is to say that the factors have to be uncorrelated (or orthogonal). Then a third factor is extracted, which works with the remaining variance, and so on.

What is presented under the heading "Factor Pattern" is the result of this process for factor 1 and factor 2. Had we said we wanted to keep three factors, there would have been a factor 3 here. In the case of principal components analysis, if we don't specify the number of factors to keep, PROC FACTOR will keep as many factors as there are variables. If we specify PRIORS SMC (see Section E below), the number of factors retained will be determined by the proportion of the total variance explained and may be less than the total number of variables. For more details on controlling the number of factors to keep, see the SAS/Stat User's Guide, Volume 1, Version 6, Fourth Edition (SAS Institute, Cary NC). The factor pattern displays what are called "factor loadings" for each of the variables. At this point in the analysis, these loadings are the simple correlations of the variables with the factor.

The next part of the printout ⑥ shows the variance explained by each factor (just the first two eigenvalues again) and then the communalities of the variables. The what? The communalities. Communalities represent how much variance in the original variable is explained by the total of all the factors which are kept in the analysis. We see here that 89% (actually, .890425) of the variance in the first question is explained (or attributable to) the first two factors. Communalities are for original variables what eigenvalues are for factors (more or less).

The next portion of the printout is a plot of the two factors retained ⑦. Note that at the bottom of the plot is a key to what the letters are. There are basically two clusters of variables here: A, B, C, and E; and F and D. One might look at this plot and wonder if the axes could be rotated so that they ran directly through the clusters. What a good idea! The only question would be how to determine just how to do the rotation. SAS software provides a number of alternative rotation methods. We look at two here.

The first is called VARIMAX. It maintains the orthogonal (uncorrelated) nature of the factors and tries to get the original variables to load high on one of the factors and low on the rest. When this occurs, it is called simple structure. Other rotation methods also attempt to obtain simple structure. The results of a varimax rotation are presented next ⑧.

There are several issues of note here. First, the factor loadings obtained from a rotation of the axes almost always result in a more readily interpretable solution. Here we see that our original notion of two factors with three variables (questions in our case) on each factor is strongly supported by the data. Second, notice that some of the loadings are negative while others are positive. Although these factor loadings can technically no longer be interpreted as correlation coefficients, it is useful to think of them in the same fashion. We would expect "Feel blue" and "Basically happy" to have opposite signs in their relationship to a depression scale. Next, notice that the variance explained by each factor has changed. Their sum is still the same, but the distribution is more equal now. The communality estimates have not changed since just as much variance is explained in each original variable as before rotation; now, however, more is attributable to the second factor (than before) and less to the first.

There are two more pieces left from this analysis. The first is a standardized scoring coefficient matrix ⑨. You would use this if you actually wanted to calculate factor scores for your subjects (PROC FACTOR will also do this for us automatically with the OUT= procedure option). Although many researchers find this useful, we don't recommend it. Instead, we recommend you simply construct scales using raw data from the variables which load on each factor. There are a number of reasons for this. First, it is much simpler and more straightforward. Second, factor analysis is highly "sample dependent." That is, results tend to be changeable from one data set to the next. Remember, what you are factoring is a matrix of correlations and, we all know how sample dependent they are.

The FINAL (yeah!) piece of this analysis is a plot of the rotated factors ⑩. You can see how the new axes really do run right through the clusters. If rotation is confusing you a little, think of it in the three factor case. Here we would have a three-dimensional swarm of points. Now think of looking at any three-dimensional object: a football, a pen, or a diamond ring. It looks different, depending on how you hold it in front of you. Rotating axes is just changing your perspective until you can get the best "look" at something.

D. *Oblique Rotations*

Well, that was entertaining. But, we said there were several rotation methods possible. VARIMAX is one popular method, PROMAX is another. They are similar in some respects, but different in one important aspect. PROMAX does not maintain the orthogonality of the factors. It allows the factors to be correlated. Why should a researcher prefer one over the other? In favor of orthogonal rotation is that it tends to keep things cleaner. If your factors are orthogonal and you then want to use them as independent variables in a regression, you will have fewer problems of collinarity. In favor of a nonorthogonal ("oblique") rotation is the argument that it is usually

silly to think that the underlying constructs which are represented by the factors are, in fact, uncorrelated. In our example, it is unreasonable to think that depression and paranoia aren't correlated. Also, it is sometimes easier to obtain simple structure with an oblique rotation. Here is the program to get the oblique rotation called PROMAX from our data set:

```
PROC FACTOR DATA=FACTOR ROTATE=PROMAX NFACTORS=2;
   TITLE 'Example of Factor Analysis - Oblique Rotation';
   VAR QUES1-QUES6;
RUN;
```

Now, much of the printout is similar to the orthogonal case, so we will just focus on one section here:

```
Rotation Method: Promax

Inter-factor Correlations

             FACTOR1     FACTOR2

FACTOR1      1.00000     -0.44010
FACTOR2     -0.44010      1.00000

Rotated Factor Pattern (Std Reg Coefs)

             FACTOR1     FACTOR2

QUES1       -0.98795    -0.11319    Feel Blue
QUES2        0.08709     0.97888    People Stare at Me
QUES3       -0.02787     0.91684    People Follow Me
QUES4        0.89043    -0.11782    Basically Happy
QUES5       -0.21841     0.65033    People Want to Hurt Me
QUES6        0.82687    -0.10212    Enjoy Going to Parties
```

Here we have the correlation between the factors and the rotated factor loading matrix. The correlation between the factors is –.44010; if we had four factors, this would be a 4 × 4 matrix. Looking at the factor loadings, we see that, in fact, we are closer to simple structure than we were before, even though the results from the orthogonal rotation were quite good.

E. *Using Communalities Other Than One*

The final stop on our journey brings us back to the notion of communalities. When factor analysts get upset over what is really factor analysis, much of the issue has to do with what is placed on the "main diagonal" of the correlation matrix that is to be factored. When ones (1.0) are used, we are basically factoring what we all know and love as correlation matrices. This approach assumes that each variable is equally as important as the others and has the same amount of interrelatedness with the other variables. In our example, this is a fairly reasonable assumption but, in most cases, some variables are more important than others, have stronger relationships with the variables in the analysis than others, or are measured with less error than others. In this case, it would be better conceptually to have some indication of how much each variable "fits in" with the others. This idea is realized by changing the communalities on the main diagonal to be less than one. Now, there is a whole science to this, but one popular approach is to take each variable and regress all the other variables against it. Then the squared multiple correlation resulting from this regression is used as the communality estimate. "Whoa," you're saying, "That's a lot of work." Indeed, but PROC FACTOR does it all for you. All we have to do is include the statement, PRIORS SMC, in our PROC FACTOR procedure, and it's done. Here is an example:

```
PROC FACTOR DATA=FACTOR PREPLOT PLOT ROTATE=VARIMAX
          NFACTORS=2 OUT=FACT SCREE;
     TITLE 'Example of Factor Analysis';
     VAR QUES1-QUES6;
     PRIORS SMC; ***This is the new line;
RUN;
```

Here is a portion of the output from this modified program:

```
Example of Factor Analysis

Initial Factor Method: Principal Factors ①

Prior Communality Estimates: SMC

      QUES1      QUES2      QUES3      QUES4      QUES5      QUES6
   0.827935   0.807363   0.806758   0.870613   0.485777   0.628900

Eigenvalues of the Reduced Correlation Matrix: ②
  Total = 4.42734575  Average = 0.73789096

                    1          2          3
Eigenvalue     3.4196     1.0322     0.1191
Difference     2.3874     0.9131     0.0525
Proportion     0.7724     0.2331     0.0269
Cumulative     0.7724     1.0055     1.0324

                    4          5          6
Eigenvalue     0.0666    -0.0934    -0.1167
Difference     0.1600     0.0232
Proportion     0.0150    -0.0211    -0.0263
Cumulative     1.0475     1.0263     1.0000
```

```
[Continued]

2 factors will be retained by the NFACTOR criterion.

Factor Pattern ③

            FACTOR1     FACTOR2

QUES1       0.76246    -0.50359   Feel Blue
QUES2       0.71563     0.56368   People Stare at Me
QUES3       0.76325     0.48653   People Follow Me
QUES4      -0.87428     0.35487   Basically Happy
QUES5       0.64510     0.17642   People Want to Hurt Me
QUES6      -0.75035     0.25896   Enjoy Going to Parties

Variance explained by each factor ④

    FACTOR1     FACTOR2
   3.419580    1.032161

Final Communality Estimates: Total = 4.451740

      QUES1      QUES2      QUES3      QUES4      QUES5      QUES6
   0.834952   0.829864   0.819257   0.890306   0.447272   0.630089

Rotation Method: Varimax

Orthogonal Transformation Matrix

              1           2
    1   -0.74526     0.66678
    2    0.66678     0.74526

Rotated Factor Pattern ⑥

            FACTOR1     FACTOR2

QUES1      -0.90401     0.13309   Feel Blue
QUES2      -0.15749     0.89725   People Stare at Me
QUES3      -0.24441     0.87151   People Follow Me
QUES4       0.88819    -0.31848   Basically Happy
QUES5      -0.36313     0.56161   People Want to Hurt Me
QUES6       0.73187    -0.30733   Enjoy Going to Parties

Variance explained by each factor

    FACTOR1     FACTOR2
   2.358154    2.093586

Final Communality Estimates: Total = 4.4517401

      QUES1      QUES2      QUES3      QUES4      QUES5      QUES6
   0.834952   0.829864   0.819257   0.890306   0.447272   0.630089
```

In all honesty, the results here are not too different than they were before but, remember, this is a simple example. There are important theoretical distinctions among the three analyses we have shown here but, if you'll notice "Factor Analysis" is only a chapter here; it isn't the whole book. Try Gorsuch (1983) for a thorough discussion of factor analysis or Pedhazur and Schmelkin (1991) for a couple of solid chapters.

F. How to Reverse Item Scores

Some researchers prefer to leave the positive and negative forms of questions as they are and interpret the sign of the factor loadings based on that information. Others prefer to modify the scores first so that all scores are in the same direction. For example, a high score (agree) to question 1 implies depression. We can reverse the scoring for questions 4 and 6 so that high scores also imply depression. One way to do this is as follows:

```
DATA FACTOR;
   INFILE 'C:\APPLIED\FACTOR.DTA' PAD;
   INPUT SUBJ 1-2 @3 (QUES1-QUES6) (1.);
   ***Reverse the scores for questions 4 and 6;
   QUES4 = 8 - QUES4;
   QUES6 = 8 - QUES6;

LABEL QUES1='Feel Blue'
      QUES2='People Stare at Me'
      QUES3='People Follow Me'
      QUES4='Basically Happy'
      QUES5='People Want to Hurt Me'
      QUES6='Enjoy Going to Parties';
RUN;
```

If you had a large number of questions that needed reversing, you could use an ARRAY to do the job (see Chapter 15).

Let's look at the rotated factor loadings when we reverse these two questions:

Rotated Factor Pattern

	FACTOR1	FACTOR2	
QUES1	0.90401	0.13309	Feel Blue
QUES2	0.15749	0.89725	People Stare at Me
QUES3	0.24441	0.87151	People Follow Me
QUES4	0.88819	0.31848	Basically Happy
QUES5	0.36313	0.56161	People Want to Hurt Me
QUES6	0.73187	0.30733	Enjoy Going to Parties

As you can quickly see, the factor loadings are identical to the one above (with PRIORS=SMC) where the scores were not reversed, except that none of the loadings are now negative.

We originally requested that PROC FACTOR create a new data set for us containing all the original values plus two factor scores. You may choose to use these factor scores in further analyses, but see the caution in Section C above. We have reduced the number of variables from six to two. In addition, these two factor scores are uncorrelated to each other, making them particularly useful in regression models. Finally, each of the two factors spans a single psychological dimension (depression or paranoia). Let's run a PROC PRINT on the new data set and see what it contains:

```
PROC PRINT DATA=FACT NOOBS;
   TITLE 'Output Data Set (FACT) Created by PROC FACTOR';
   TITLE2 'Questions 4 and 6 Reversed';
RUN;
```

Output Data Set (FACT) Created by PROC FACTOR
Questions 4 and 6 Reversed

SUBJ	QUES1	QUES2	QUES3	QUES4	QUES5	QUES6	FACTOR1	FACTOR2
1	7	3	4	6	2	6	1.13994	-0.11342
2	6	3	2	7	3	6	1.24019	-0.43330
3	3	6	7	5	6	5	-0.35471	2.11191
4	2	2	2	3	3	4	-0.64087	-0.32356
5	3	4	2	4	2	5	-0.39275	0.07976
6	6	3	4	6	3	6	0.96485	0.05673
7	1	2	3	1	2	6	-1.23125	-0.01079
8	3	3	2	5	4	5	0.05043	-0.07278
9	2	1	1	2	2	3	-0.87196	-0.93028
10	6	2	3	6	2	6	1.10334	-0.51853
11	3	5	4	6	3	5	0.05298	0.94688
12	6	7	6	6	6	6	0.39438	1.79914
13	5	1	1	6	6	6	1.07234	-0.97856
14	2	1	1	2	1	3	-0.87503	-0.98015
15	1	2	1	1	1	1	-1.65188	-0.63303

If factor 1 is depression and factor 2 is paranoia, you can readily spot those subjects who are the most depressed or are paranoid.

Problems

10-1. Run a factor analysis on the questionnaire data in Chapter 3, Section B. Use only the variables PRES, ARMS, and CITIES. Request two factors, VARIMAX rotation method, and set the PRIORS estimate of communality to SMC.

10-2. Using the test scores in Chapter 11, Section B, run a factor analysis on the test, using the scored responses to each of the five questions as the items to factor analyze. Request only one factor, and create an output data set containing that one factor score. Do not request any rotations. Use PROC PRINT to list the contents of this output data set. The first unrotated factor from a test is sometimes related to IQ and called factor G (for "general"). Note that there are far too few observations to run a meaningful factor analysis—it is for instructional purposes only.

CHAPTER

11

Psychometrics

A. *Introduction*

This chapter contains programs to score a test, to perform item analysis, test reliability (Cronbach's Alpha), and interrater reliability (Coefficient Kappa). In Section E, you will find a complete program for item analysis that you are free to use or incorporate in a larger test scoring and item analysis program.

B. *Using SAS Software to Score a Test*

We start with a simple program that will score a five question multiple-choice test. Later sections enhance this program so that it will be more general and will work with any number of questions. First the program, then the explanation:

```
*------------------------------------------------------------*
| Program Name: SCORE1.SAS in C:\APPLIED                      |
| Purpose: To score a five item multiple choice exam.         |
| Data: The first line is the answer key, remaining lines     |
|       contain the student responses                         |
| Date: July 23, 1996                                         |
*------------------------------------------------------------*;
```

[Continued]

```
DATA SCORE;
   ARRAY ANS[5] $ 1 ANS1-ANS5; ***Student answers;
   ARRAY KEY[5] $ 1 KEY1-KEY5; ***Answer key;
   ARRAY S[5] 3 S1-S5; ***Score array 1=right,0=wrong;
   RETAIN KEY1-KEY5; ①

   ***Read the answer key;
   IF _N_ = 1 THEN INPUT (KEY1-KEY5)($1.); ②

   ***Read student responses;
   INPUT @1 ID 1-9 ③
         @11 (ANS1-ANS5)($1.);

   ***Score the test;
   DO I = 1 TO 5; ④
      S[I] = KEY[I] EQ ANS[I]; ⑤
   END;

   ***Compute Raw and Percentage scores;
   RAW = SUM (OF S1-S5); ⑥
   PERCENT = 100*RAW / 5; ⑦

   KEEP ID RAW PERCENT;

   LABEL ID = 'Social Security Number'
         RAW = 'Raw Score'
         PERCENT = 'Percent Score';
DATALINES;
ABCDE
123456789 ABCDE
035469871 BBBBB
111222333 ABCBE
212121212 CCCDE
867564733 ABCDA
876543211 DADDE
987876765 ABEEE
;
PROC SORT DATA=SCORE;
   BY ID;
RUN;

PROC PRINT DATA=SCORE LABEL;
   TITLE 'Listing of SCORE data set';
   ID ID;
   VAR RAW PERCENT;
   FORMAT ID SSN11.;
RUN;
```


Here is the output from the PROC PRINT:

```
Listing of SCORE data set

     Social
   Security   Raw    Percent
     Number   Score   Score

035-46-9871    1       20
111-22-2333    4       80
123-45-6789    5      100
212-12-1212    3       60
867-56-4733    4       80
876-54-3211    2       40
987-87-6765    3       60
```

Since the answer key for our test is contained in the first line of data, and the student responses in the remaining lines, we can use the automatic SAS DATA step variable _N_, which is incremented by one for each iteration of the DATA step. When the program starts, the variable _N_, will have a value of 1 and the first line of data will be read into the KEY variables ②. By retaining the KEYn variables ①, their value will be available to compare to each of the student responses. Remember that the SAS system normally sets the value of each variable to missing before a new data line is read. By retaining the KEYn variables, this initialization does not occur.

The program continues with the next INPUT statement ③ and reads a line of student data. For all subsequent iterations of the data step, _N_ will be greater than 1 and statement ② will not execute again. Thus, the answer key is skipped and the student ID and responses are read ③.

The scoring DO loop ④ compares each of the student responses with the answer key. Statement ⑤ is somewhat unusual and needs some explanation. The right-most portion of the statement (KEY[I] EQ ANS[I]) is a logical comparison. If the student answer (ANS[I]) is equal to the answer key (KEY[I]), then the value of S[I] will be 1 (true). Otherwise, it will be a 0 (false). A value of 1 or 0 will then be assigned to the variable S[I]. Instead, you could also score the test with two lines, like this:

```
IF KEY[I] = ANS[I] THEN S[I] = 1;
ELSE ANS[I] = 0;
```

The SUM statement ⑥ gives us the number of correct answers on the test. A percentage score is computed by dividing the number of correct responses by the number of items on the test and multiplying by 100 ⑦.

We make use of the built-in format SSN11. to print the student social security numbers in standard format (which also ensures that the leading zeros in the number are printed).

C. Generalizing the Program for a Variable Number of Questions

This next program extends the program above in two ways: First, it can be used to score tests with different numbers of items. Second, several procedures are added to produce class summary reports. Here is the program (to stay with our previous example, the number of questions is set to five):

```
*-------------------------------------------------------*
| Program Name: SCORE2.SAS in C:\APPLIED                |
| Purpose: To score a multiple-choice exam with an arbitrary |
|          number of items                              |
| Data: The first line is the answer key, remaining lines |
|       contain the student responses                   |
|       Data in file C:\APPLIED\TEST.DTA                |
| Date: July 23, 1996                                   |
*-------------------------------------------------------*;

%LET NUMBER = 5; ***The number of items on the test; ①

DATA SCORE;
   INFILE 'C:\APPLIED\TEST.DTA'; ②
   ARRAY ANS[&NUMBER] $ 1 ANS1-ANS&NUMBER; ***Student answers;
   ARRAY KEY[&NUMBER] $ 1 KEY1-KEY&NUMBER; ***Answer key;
   ARRAY S[&NUMBER] 3 S1-S&NUMBER; ***Score array 1=right,0=wrong;
   RETAIN KEY1-KEY&NUMBER;

   IF _N_ = 1 THEN INPUT (KEY1-KEY&NUMBER)($1.);

   INPUT @1 ID 1-9
         @11 (ANS1-ANS&NUMBER)($1.);

   DO I = 1 TO &NUMBER;
      S[I] = KEY[I] EQ ANS[I];
   END;

   RAW = SUM (OF S1-S&NUMBER);

   PERCENT = 100*RAW/&NUMBER;

   KEEP ANS1-ANS&NUMBER ID RAW PERCENT;

   LABEL ID = 'Social Security Number'
         RAW = 'Raw Score'
         PERCENT = 'Percent Score';
RUN;

PROC SORT DATA=SCORE; ③
   BY ID;
RUN;

PROC PRINT DATA=SCORE LABEL; ④
   TITLE 'Listing of SCORE data set';
   ID ID;
   VAR RAW PERCENT;
```


[Continued]

```
   FORMAT ID SSN11.;
RUN;

PROC MEANS DATA=SCORE MAXDEC=2 N MEAN STD RANGE MIN MAX; ⑤
   TITLE 'Class Statistics';
   VAR RAW PERCENT;
RUN;

PROC CHART DATA=SCORE; ⑥
   TITLE 'Histogram of Student Scores';
   VBAR PERCENT/MIDPOINTS=50 TO 100 BY 5;
RUN;

PROC FREQ DATA=SCORE; ⑦
   TITLE 'Frequency Distribution of Student Answers';
   TABLES ANS1-ANS&NUMBER/NOCUM;
RUN;
```

This program uses a macro variable (&NUMBER) which is assigned in the %LET statement ①. Each occurrence of &NUMBER is replaced by this assigned value before the program executes.

One other change from the previous program is that this program reads data from an external file, which is accomplished by the INFILE statement ②. The remainder of the DATA step portion of the program is identical to the previous program.

The first several PROCs are straightforward. We want a student roster in ID order ③ ④, the class statistics ⑤, a histogram ⑥, and the frequencies of A's, B's, etc., for each of the questions of the test ⑦.

A portion of the output from this program is shown next:

```
Listing of SCORE data set

   Social
  Security    Raw    Percent
   Number    Score    Score

111-22-2333    4        80
113-45-4545    4        80
132-43-4567    3        60
345-45-6233    4        80
386-54-7098    5       100
   . . .
Class Statistics

Variable   Label             N     Mean    Std Dev     Range
-------------------------------------------------------------
RAW        Raw Score        34     2.91       1.38      5.00
PERCENT    Percent Score    34    58.24      27.58    100.00
-------------------------------------------------------------
```

[Continued]

```
Variable   Label              Minimum    Maximum
--------------------------------------------------
RAW        Raw Score            0.00       5.00
PERCENT    Percent Score        0.00     100.00
--------------------------------------------------


Histogram of Student Scores

Frequency
   | ***
   | ***
10 + ***                 ***
   | ***                 ***
   | ***                 ***
   | ***     ***         ***
   | ***     ***         ***
 5 + ***     ***         ***
   | ***     ***         ***
   | ***     ***         ***
   | ***     ***         ***         ***
   | ***     ***         ***         ***
   ------------------------------------------
     50  55  60  65  70  75  80  85  90  95  100
                    Percent Score

Frequency Distribution of Student Answers

ANS1   Frequency   Percent
--------------------------
A           19       55.9
B            6       17.6
D            5       14.7
E            4       11.8

ANS2   Frequency   Percent
--------------------------
A            4       11.8
B           17       50.0
C            3        8.8
D            3        8.8
E            7       20.6
  . . .
```

D. Creating a Better Looking Table Using PROC TABULATE

We can produce a compact table showing answer-choice frequencies using PROC TABULATE. To do this efficiently, we will restructure the data set so that we have a variable called QUESTION, which is the question number; and CHOICE, which is the answer choice for that question for each student. We will be fancy and create CHOICE as a character variable that shows the letter choice (A, B, C, D, or E) with

an asterisk (*) next to the correct choice for each question. Again, we offer the program here without much explanation for those who might find the program useful or those who would like to figure out how it works. One of the authors (Smith) insists that good item analysis includes the mean test score for all students choosing each of the multiple-choice items. Therefore, the code to produce this statistic is included as well. The details of TABULATE are too much to describe here, and we refer you to the SAS Guide to Tabulate Processing, available from the SAS Institute, Cary, NC. This is one of the best manuals from the SAS Institute; and if you plan to use PROC TABULATE (it's very powerful) we highly recommend this manual.

The complete program to restructure the data set and produce the statistics described above is shown next:

```
*---------------------------------------------------------*
| Program Name: SCORE3.SAS in C:\APPLIED                  |
| Purpose: To score a multiple-choice exam with an arbitrary
|          number of items and compute item statistics    |
| Data: The first line is the answer key, remaining lines |
|       contain the student responses. Data is located in |
|       file C:\APPLIED\TEST.DTA                          |
| Date: July 23, 1996                                     |
*---------------------------------------------------------*;
OPTIONS LS=64 PS=59 NOCENTER;

PROC FORMAT; ①
   PICTURE PCT LOW-<0=' ' 0-HIGH='00000%';
RUN;

%LET NUMBER = 5; ***The number of items on the test;

DATA SCORE;
   INFILE 'C:\APPLIED\TEST.DTA';
②  ARRAY ANS[&NUMBER] $ 2 ANS1-ANS&NUMBER; ***Student answers;
   ARRAY KEY[&NUMBER] $ 1 KEY1-KEY&NUMBER; ***Answer key;
   ARRAY S[&NUMBER] 3 S1-S&NUMBER; ***Score array 1=right,0=wrong;
   RETAIN KEY1-KEY&NUMBER;

   IF _N_ = 1 THEN INPUT (KEY1-KEY&NUMBER)($1.);

   INPUT @1 ID 1-9
         @11 (ANS1-ANS&NUMBER)($1.);

   DO I = 1 TO &NUMBER;
      IF KEY[I] EQ ANS[I] THEN DO;
         S[I] = 1;
③        SUBSTR(ANS[I],2,1) = '*'; ***Place an asterisk next
                                      to correct answer;
      END;
      ELSE S[I] = 0;
   END;

   RAW = SUM (OF S1-S&NUMBER);
   PERCENT = 100*RAW / &NUMBER;
```

[Continued]

```
   KEEP ANS1-ANS&NUMBER ID RAW PERCENT;

   LABEL ID = 'Social Security Number'
         RAW = 'Raw Score'
         PERCENT = 'Percent Score';
RUN;

DATA TEMP; ④
   SET SCORE;
   ARRAY ANS[*] $ 2 ANS1-ANS&NUMBER;
   DO QUESTION=1 TO &NUMBER;
      CHOICE=ANS[QUESTION];
      OUTPUT;
   END;
   KEEP QUESTION CHOICE PERCENT;
RUN;

PROC TABULATE DATA=TEMP;
   TITLE 'Item Analysis Using PROC TABULATE';
   CLASS QUESTION CHOICE;
   VAR PERCENT;
   TABLE QUESTION*CHOICE,
      PERCENT=' '*(PCTN<CHOICE>*F=PCT. MEAN*F=PCT.
      STD*F=10.2) / RTS=20 MISSTEXT=' ';
   KEYLABEL ALL='Total' MEAN='Mean Score' PCTN='FREQ'
         STD= 'Standard Deviation';
RUN;
```

A brief explanation of the program follows: PROC FORMAT ① is used to create a picture format so that we can print scores as percentages. (NOTE: There is a PERCENTn. format available as part of the SAS system, but it multiplies by 100 as well as placing a percent sign after a number.) The remainder of the SCORE DATA step is the same as the previous program; with the exception that the ANS1-ANSn variables are now two bytes in length. We will use this second byte later to place an asterisk next to the correct answer for each item. The SUBSTR function on the left of the equals sign ③ is used to place an asterisk in the second position of the correct answer choice. A DATA step ④ is needed to restructure the data set so that it will be in a convenient form for PROC TABULATE. This data set contains n observations per student, where n is the number of items on the test. Selected portions of the output from these procedures are shown in the table on page 273.

The frequency column shows the percentage of students selecting each item choice. The frequency next to the correct answer (marked by an *) is the item's difficulty (percentage of students answering the item correctly). The column labeled MEAN SCORE shows the mean test score for all students answering each answer choice. For example, for item 1, 55% of the students chose A, which is the correct answer. The students who chose A had a mean score of 71% on the test. Seventeen percent of the students picked choice B, and the mean score of all students who chose B was 26%, and so

Item Analysis Using PROC TABULATE

QUESTION	CHOICE	FREQ	Mean Score	Standard Deviation
1	A*	55%	71%	23.40
	B	17%	26%	24.22
	D	14%	52%	22.80
	E	11%	50%	11.55
2	A	11%	40%	28.28
	B*	50%	75%	16.63
	C	8%	40%	20.00
	D	8%	73%	11.55
	E	20%	28%	22.68
3	A	8%	20%	20.00
	B	17%	46%	24.22
	C*	52%	68%	22.98
	D	8%	53%	46.19
	E	11%	60%	16.33

forth. Looking briefly at item 2, we see that the students who picked choice D did fairly well on the test overall (73% correct). We might want to look at choice D to make sure it is not misleading.

E. A Complete Test Scoring and Item Analysis Program

We present here a complete test scoring and item analysis program. This is a relatively complex program, and we will not go into any detail about its inner workings. We present it so that you may copy pieces of it, or all of it, and use it to analyze your multiple-choice tests. If you examine the sample output below the program, you will see that a lot of information is presented in a compact table. Each row of the table shows an item number, the answer key, the percentage of the students who chose each of the answer choices, a difficulty (the proportion of students answering the item correctly), a point-biserial correlation coefficient, and the proportion of students (in each quartile of the class) answering the item correctly. Here is the complete program:

```
*------------------------------------------------------------*
| Program Name: SCORE4.SAS in C:\APPLIED                     |
| Purpose: To score a multiple-choice exam with an arbitrary |
|          number of items                                   |
| Data: The first line is the answer key, remaining lines   |
|       contain the student responses                        |
|       Data in file C:\APPLIED\TEST.DTA                     |
| Date: July 23, 1996                                        |
*------------------------------------------------------------*;
```

[Continued]

```
%LET NUMBER = 5; ***The number of items on the test;

DATA SCORE;
   INFILE 'C:\APPLIED\TEST.DTA';
   ARRAY ANS [&NUMBER] $ 1 ANS1-ANS&NUMBER; ***Student answers;
   ARRAY KEY [&NUMBER] $ 1 KEY1-KEY&NUMBER; ***Answer key;
   ARRAY S [&NUMBER] S1-S&NUMBER; ***Score array 1=right,0=wrong;
   RETAIN KEY1-KEY&NUMBER;

   IF _N_ = 1 THEN INPUT (KEY1-KEY&NUMBER)($1.);

   INPUT @1 ID 1-9
         @11 (ANS1-ANS&NUMBER)($1.);

   DO I = 1 TO &NUMBER;
      S [I] = KEY [I] EQ ANS [I];
   END;

   RAW = SUM (OF S1-S&NUMBER);
   PERCENT = 100*RAW / &NUMBER;

   KEEP ANS1-ANS&NUMBER S1-S&NUMBER KEY1-Key&NUMBER
        ID RAW PERCENT;
   ***Note: ANS1-ANSn, S1-Sn, KEY1-KEYn
        are needed later on;
   LABEL ID = 'Social Security Number'
         RAW = 'Raw Score'
         PERCENT = 'Percent Score';
RUN;

*-------------------------------------------------------*
| You may want to include the procedures in Section C  |
| which print student rosters, histograms, and class   |
| statistics.                                          |
*-------------------------------------------------------*;

***Section to prepare data sets for PROC TABULATE;

***Write correlation coefficients to a data set;
PROC CORR DATA=SCORE NOSIMPLE NOPRINT
          OUTP=CORROUT(WHERE = (_TYPE_='CORR'));
   VAR S1-S&NUMBER;
   WITH RAW;
RUN;

***Reshape the data set;
DATA CORR;
   SET CORROUT;
   ARRAY S{*} 3 S1-S&NUMBER;
   DO I=1 TO &NUMBER;
```


[Continued]

```
      CORR=S[I];
      OUTPUT;
   END;
   KEEP I CORR;
RUN;

***Compute quartiles;
PROC RANK DATA=SCORE GROUPS=4 OUT=QUART(DROP=PERCENT ID);
   RANKS QUARTILE;
   VAR RAW;
RUN;

***Create ITEM variable and reshape again;
DATA TAB;
   SET QUART;
   LENGTH ITEM $ 5 QUARTILE CORRECT I 3 CHOICE $ 1;
   ARRAY S{*} S1-S&NUMBER;
   ARRAY ANS{*} $ 1 ANS1-ANS&NUMBER;
   ARRAY KEY{*} $ 1 KEY1-KEY&NUMBER;
   QUARTILE=QUARTILE+1;
   DO I = 1 TO &NUMBER;
      ITEM = RIGHT(PUT(I,3.)) || " " || KEY[I];
      CORRECT = S[I];
      CHOICE = ANS[I];
      OUTPUT;
   END;
   KEEP I ITEM QUARTILE CORRECT CHOICE;
RUN;

PROC SORT DATA=TAB;
   BY I;
RUN;

***Combine correlations and quartile information;
DATA BOTH;
   MERGE CORR TAB;
   BY I;
RUN;

***Print ont a pretty table;
OPTIONS LS=72;
PROC TABULATE FORMAT=7.2 DATA=BOTH ORDER=INTERNAL NOSEPS;
   TITLE 'Item Statistics';
   LABEL QUARTILE = 'Quartile'
         CHOICE = 'Choices';
   CLASS ITEM QUARTILE CHOICE;
   VAR CORRECT CORR;
   TABLE ITEM='# Key'*F=6.,
   CHOICE*(PCTN<CHOICE>)*F=3. CORRECT=' '*MEAN='Diff.'*F=PERCENT5.2
      CORR=' '*MEAN='Corr.'*F=5.2
```

[Continued]

```
      CORRECT=' '*QUARTILE*MEAN='Prop. Correct'*F=PERCENT7.2/
      RTS=8;
   KEYLABEL PCTN='%' ;
RUN;
```

Here is a sample of the output from this program:

Item Statistics

	Choices							Quartile			
								1	2	3	4
	A	B	C	D	E			Prop.	Prop.	Prop.	Prop.
	%	%	%	%	%	Diff.	Corr.	Correct	Correct	Correct	Correct
# Key											
1 A	56	18	.	15	12	56%	0.55	33.3%	28.6%	90.9%	100%
2 B	12	50	9	9	21	50%	0.63	0.00%	42.9%	72.7%	100%
3 C	9	18	53	9	12	53%	0.42	16.7%	50.0%	63.6%	100%
4 D	15	6	6	65	9	65%	0.68	0.00%	64.3%	90.9%	100%
5 E	9	3	15	6	68	68%	0.51	16.7%	71.4%	81.8%	100%

To make this listing clear, let's look at item 1. The correct answer is 'A', which 56% of the students chose. Eighteen percent of the class chose 'B', and so forth. The item difficulty is 56%, and the point-biserial coefficient is .55. Thirty-three percent of the bottom quartile (lowest) answered this item correctly; 29% of the next quartile; 91% of the third quartile; and 100% of the top quartile answered this item correctly.

F. Test Reliability

As of version 6.06, PROC CORR has had the ability to compute Cronbach's Coefficient Alpha. For test items that are dichotomous, this coefficient is equivalent to the popular Kuder-Richardson formula 20 coefficient. These are the most popularly used estimates of the reliability of a test. They basically assess the degree to which the items on a test are all measuring the same underlying concept. The lines below show how to compute Coefficient Alpha from the data set SCORE in Section C of this chapter:

```
PROC CORR DATA=SCORE NOSIMPLE ALPHA;
   TITLE 'Coefficient Alpha from Data Set SCORE';
   VAR S1-S5;
RUN;
```


Since each test item is either right (1) or wrong (0), Coefficient Alpha is equivalent to KR-20. Here is a partial listing:

```
Coefficint Alpha from Data Set SCORE

Correlation Analysis

Cronbach Coefficient Alpha

for RAW variables        : 0.441866
for STANDARDIZED variables: 0.444147


                     Raw Variables                Std. Variables

Deleted     Correlation                     Correlation
Variable     with Total         Alpha        with Total        Alpha
-----------------------------------------------------------------------
S1             0.219243      0.395887          0.211150     0.404775
S2             0.316668      0.321263          0.317696     0.325870
S3             0.053238      0.511023          0.049289     0.513369
S4             0.404102      0.256819          0.414362     0.248868
S5             0.189630      0.415648          0.196747     0.414978
```

G. Interrater Reliability

In studies where more than one rater rates subjects, you may want to determine how well the two raters agree. Suppose each rater is rating a subject as normal or not normal. By chance alone, the two raters will agree from time to time, even if they are both assigning ratings randomly. To adjust for this, a test statistic called Kappa was developed. If you are running SAS version 6.10 or later, Kappa is requested using the AGREE option on the TABLE statement of PROC FREQ. Suppose each of two raters rated 10 subjects as shown below:

Outcome (N = Normal, X = Not Normal)

Subject	Rater 1	Rater 2
1	N	N
2	X	X
3	X	X
4	X	N
5	N	X
6	N	N
7	N	N
8	X	N
9	X	X
10	N	N

The program to compute Kappa is:

```
DATA KAPPA;
   INPUT SUBJECT RATER_1 $ RATER_2 $ @@;
DATALINES;
1 N N  2 X X  3 X X  4 X N  5 N X
6 N N  7 N N  8 X N  9 X X  10 N N
;
PROC FREQ DATA=KAPPA;
   TITLE 'Coefficient Kappa Calculation';
   TABLE RATER_1 * RATER_2 / NOCUM NOPERCENT KAPPA;
RUN;
```

The output is shown below:

```
Coefficient Kappa Calculation

TABLE OF RATER_1 BY RATER_2

RATER_1     RATER_2

Frequency|      |      |
Row Pct  |      |      |
Col Pct  |N     |X     |  Total
---------+------+------+
N        |     4|     1|    5
         | 80.00| 20.00|
         | 66.67| 25.00|
---------+------+------+
X        |     2|     3|    5
         | 40.00| 60.00|
         | 33.33| 75.00|
---------+------+------+
Total          6      4     10

STATISTICS FOR TABLE OF RATER_1 BY RATER_2

McNemar's Test
--------------
Statistic = 0.333    DF = 1    Prob = 0.564

Simple Kappa Coefficient
------------------------
                                    95% Confidence Bounds
Kappa = 0.400      ASE = 0.284      -0.157      0.957

Sample Size = 10
```

As you can see, Kappa is .4 between these two raters. This may not seem like a terrific reliability, and indeed it isn't. You might say, "But they were in agreement for 7 of 10 cases," and indeed they were. But we would expect 5 out of 10 agreements by

flipping a coin (there are only two ratings possible here), so 7 out of 10 is not a wonderful improvement over chance.

Problems

11-1. Given the answer key below:

Question 1 = 'B' Question 2 = 'C' Question 3 = 'D'
Question 4 = 'A' Question 5 = 'A'

Write a SAS program to grade the six students whose data are shown below. Provide one listing in Social Security number order and another in decreasing test-score order. Compute both a raw score (the number of items correct) and a percentage score. (HINT: Use the SSN11. format for the Social Security number. Be sure to read it as a numeric if you do this.)

Student Data:

Social Security No.	Responses to Five Items
123-45-6789	B C D A A
001-44-5559	A B C D E
012-12-1234	B C C A B
135-63-2837	C B D A A
005-00-9999	E C E C E
789-78-7878	B C D A A

11-2. Using the test data from problem 11-1, compute the KR-20 (or Cronbach's Alpha) for the test. Also, compute a point-biserial correlation coefficient for each item. Remember that a point-biserial correlation is equivalent to a Pearson Correlation coefficient when one of the scores has values of 0 or 1.

11-3. Two pathologists viewed 14 slides and made a diagnosis of cancer or not cancer. Using the data below, compute Kappa, an index of interrater reliability: (C = Cancer, X = Not cancer)

Rater 1	Rater 2	Rater 1	Rater 2
C	C	C	X
X	X	C	C
X	X	X	X
C	X	C	C
X	C	C	C
X	X	X	X
X	X	C	C

CHAPTER

12

The SAS INPUT Statement

A. Introduction

Throughout the examples in the statistics section of this book, we have seen some of the power of the SAS INPUT statement. In this chapter, the first in a section on SAS Programming, we explore the power of the INPUT statement. (NOTE: To learn the basics of the INPUT statement, return to Chapter 1.)

B. List Directed Input: Data values separated by spaces

SAS can read data values separated by one or more spaces. This form of input is sometimes referred to as list directed. The rules here are that we must read every variable on a line, the data values must be separated by one or more spaces, and missing values are represented by periods. A simple example is shown below:

```
DATA QUEST;
   INPUT ID GENDER $ AGE HEIGHT WEIGHT;
DATALINES;
1 M 23 68 155
2 F . 61  102
3   M  55   70    202
;
```


Notice that character variable names are followed by a $. The multiple spaces between the data values in the third line of data will not cause a problem.

C. Reading Comma-delimited Data

Sometimes we are given data that are comma-delimited (i.e., separated by commas) instead of the spaces that SAS is expecting. We have two choices here: We can use an editor and replace all the commas with blanks, or we can leave the commas in the data and use the DLM= option of the INFILE statement to make the comma the data delimiter. (See Chapter 13 for details on the INFILE statement and its options.) As an example, suppose you were given a file on a floppy diskette called SURVEY.DTA. All the data values are separated by commas. The first three lines are shown next:

```
1,M,23,68,155
2,F,.,61,102
3,  M, 55, 70,  202
```

To read this file we code:

```
DATA HTWT;
   INFILE 'A:SURVEY.DTA' DLM=',';
   INPUT ID GENDER $ AGE HEIGHT WEIGHT;
RUN;
```

The INFILE statement directs the INPUT statement to read an external file called SURVEY on the floppy diskette in drive A and to use commas as the data delimiter.

Another useful INFILE option for reading comma-delimited files is DSD. Besides treating commas as delimiters, this option performs several other functions. First, if it finds two adjacent commas, it will assign a missing value to the corresponding variable in the INPUT list. Second, it will allow text strings surrounded by quotes to be read into a character variable and will strip off the quotes in the process. To illustrate the DSD option, suppose the three lines of data below are stored in the file A:SURVEY.DTA.

```
1,"M",23,68,155
2,F,,61,102
3,  M, 55, 70,  202
```

A SAS DATA step to read these three lines of data is shown next:

```
DATA HTWT;
   INFILE 'A:SURVEY.DTA' DSD;
   INPUT ID GENDER $ AGE HEIGHT WEIGHT;
RUN;
```

The resulting data set will be identical to the earlier example using the DLM option. Notice that the GENDER for the first person will be the value 'M' without the quotes, and the AGE for the second person will be assigned a missing value.

D. Using INFORMATS with List-directed Data

We may have data such as date values which we want to read with a date informat, but still want to use the list-directed form of input. We have two choices here. One is to precede the INPUT statement with an INFORMAT statement, assigning an informat to each variable. An INFORMAT statement uses the same syntax as a FORMAT statement but is used to supply an input format instead of an output format for a variable. An example, using an INFORMAT statement, is shown below:

```
DATA INFORM;
   INFORMAT DOB VISIT MMDDYY8.;
   INPUT ID DOB VISIT DX;
DATALINES;
1 10/21/46 6/5/89 256.20
2 9/15/44 4/23/89 232.0
   etc.
```

An alternative to the INFORMAT statement is to supply the informats directly in the INPUT statement. We do this by following the variable name with a colon, followed by the appropriate informat. A program using this method with the same data set is:

```
DATA FORM;
   INPUT ID DOB : MMDDYY8. VISIT : MMDDYY8. DX;
DATALINES;
1 10/21/46 6/5/89 256.20
2 9/15/44 4/23/89 232.0
   etc.
```

Either method can also be used to override the default eight character limit for character variables. So, to read a file containing last names (some longer than eight characters) we could use either of the next two programs:

```
*Example with an INFORMAT statement;
DATA LONGNAME;
   INFORMAT LAST $20.;
   INPUT ID LAST SCORE;
DATALINES;
1 STEVENSON 89
2 CODY 100
3 SMITH 55
4 GETTLEFINGER 92
   etc.
```


[Continued]

```
*Example with INPUT informats;
DATA LONGNAME;
   INPUT ID LAST : $20. SCORE;
DATALINES;
1 STEVENSON 89
2 CODY 100
3 SMITH 55
4 GETTLEFINGER 92
   etc.
```

Before we leave "list-directed" reads, there is one more "trick" you should know about. Suppose you wanted to read a first and last name into a single variable. If we used spaces as data delimiters, we could not have a blank between the first and last name. However, the very clever people at the SAS Institute have thought about this problem and have come up with the & sign in a list-directed INPUT statement. An ampersand (&) modifier following a variable name changes the default delimiter of one space to two or more spaces. You may also follow the ampersand with an INFORMAT. To see how this works, look at the next program:

```
DATA FIRSTLST;
   INPUT ID NAME & $30. SCORE1 SCORE2;
DATALINES;
1 RON CODY  97 98
2 JEFF SMITH  57 58
   etc.
```

Notice that there are at least two spaces between the name and the first score.

E. Column Input

Most of the examples in this book use INPUT statements that specify which columns to read for each data value. The syntax is to list the variable names, followed by the column or columns to read. In addition, we follow the variable name by a $ sign if we are reading character values. A simple example is:

```
DATA COL;
   INPUT ID 1-3 GENDER $ 4 HEIGHT 5-6 WEIGHT 7-11;
DATALINES;
001M68155.5
2  F61 99.0
  3M  233.5
(more data lines)
```

Notice that the ID number for subject number 2 is not right-adjusted. In some programming languages this would cause a problem; not so for SAS software. We could have placed the '2' in any of the first three columns, and it would have been read properly. Notice also that we can include decimal points in numeric fields. Just remember to leave the extra columns for them. Finally, remember that we can simply leave columns blank when we have missing values.

If you want to make this program a bit easier to read, you can spread out the INPUT statement on several lines like this:

```
DATA COL;
   INPUT ID       1-3
         GENDER $   4
         HEIGHT   5-6
         WEIGHT   7-11;
DATALINES;
001M68155.5
2 F61 99.0
  3M  233.5
(more data lines)
```

F. Pointers and Informats

An alternative to specifying the starting and ending columns (column input) is to specify a starting column and an INFORMAT (which also specifies how many columns to read). This is especially useful when we are given a coding layout like the following:

Variable	Starting Column	Length	Type	Description
ID	1	3	NUM	SUBJECT ID
GENDER	4	1	CHAR	GENDER M=MALE F=FEMALE
AGE	9	2	NUM	AGE IN YEARS
HEIGHT	11	2	NUM	HEIGHT IN INCHES
V_DATE	15	6	DATE	VISIT DATE IN MMDDYY FORMAT

Rather than doing all the high-level arithmetic to compute ending columns for each of these variables, we can use a pointer (@ sign) to specify the starting column and an informat, which will not only tell SAS how to read the data value, but how many columns to read. Here is the program to read the data layout above:

```
DATA POINT;
  INPUT @1 ID 3.
        @4 GENDER $1.
        @9 AGE 2.
        @11 HEIGHT 2.
        @15 V_DATE MMDDYY6.;
```


The @ symbol, called an absolute column pointer, indicates the starting column for each variable. In the INPUT statement above, some of the pointers are redundant, such as the @4 before GENDER. As data are read, an internal pointer moves along the data line. Since ID started in column 1 and was three columns in length, this internal pointer was ready to read data in column 4 next. We recommend using an absolute column pointer before every variable as in this example; it makes for a neater program and reduces the possibility of reading the wrong column. The informat N. is used for a numeric variable of length N; $N. is the informat used for a character variable of length N. The general form for a SAS numeric informat is N.n where N is the number of columns to read and n is the number of digits to the right of an implied decimal point (if a decimal point is not included in the data value). For example, the number 123 read with a format 3.2 would be interpreted as 1.23 (we are reading three columns, and there are two digits to the right of the decimal point). Using this notation, we can read numbers with an "implied" decimal point. By the way, we can also read numbers with decimal points. The number 1.23 read with the format 4. (remember the decimal point takes up a column) would be interpreted as 1.23. The informat MMDDYY6. was one of the date formats we used in Chapter 4 to read date values. We used a separate line for each variable simply as a matter of programming style.

G. Reading More Than One Line per Subject

When we have more than one line of data for each subject, we can use the row pointer, #, to specify which row of data we want to read. Just as with the column pointer, we can move anywhere within the multiple rows of data per subject. Keep in mind that we must have the same number of rows of data for each subject. Below is an example where two lines of data were recorded for each subject:

```
DATA COLUMN;
   INPUT #1 ID 1-3 AGE 5-6 HEIGHT 10-11 WEIGHT 15-17
         #2 SBP 5-7 DBP 8-10;
DATALINES;
001 56   72   202
    140080
002 45   70   170
    130070
;
```

If you have N lines of data per subject but don't want to read data from all N lines, make sure to end your input statement with #N where N is the number of data lines per subject. For example, if you have six lines of data per subject but only want to read two lines (as in the example above), you would write:

```
INPUT #1 ID 1-3 AGE 5-6 HEIGHT 10-11 WEIGHT 15-17
      #2 SBP 5-7 DBP 8-10   #6;
```

H. Changing the Order and Reading a Column More Than Once

It is possible to move the absolute column pointer to any starting column, in any order. Thus, we can read variables in any order, and we can read columns more than once. Suppose we had a six-digit ID where the last two digits represented a county code. We could do this:

```
INPUT @1 ID 6. @5 COUNTY 2. etc.;
```

or

```
INPUT ID 1-6 COUNTY 5-6 etc.;
```

We can also read variables in any order. The following INPUT statements are valid:

```
INPUT ID 1-3 HEIGHT 5-6 GENDER $ 4 WEIGHT 7-9;

INPUT @1 ID 3. @5 HEIGHT 2. @4 GENDER $1. @7 WEIGHT 3.;
```

I. Informat Lists

You may place a group of variables together within parentheses in an INPUT statement and follow this list by one or more INFORMATS, also in parentheses. Below is a simple example:

```
INPUT (X Y Z C1-C3)(1. 2. 1. $3. $3. $3.);
```

Now, this isn't very useful as shown. Where you save time is when several variables in a row all use the same INFORMAT. You can have fewer informats in the INFORMAT list than there are variables in the variable list. When this happens, the INFORMAT list is recycled as many times as necessary to provide an INFORMAT for each of the variables in the variable list. The INPUT statement below illustrates this:

```
INPUT (X1-X50)(1.);
```

This indicates to read X1, X2, X3, . . . , X50, all with the 1. INFORMAT. Below is an example showing how you can shorten a fairly long INPUT statement using variable lists and INFORMAT lists. First, the INPUT statement without using variable and INFORMAT lists:

```
DATA NOVICE;
INPUT ID 1-3 QUES1 4 QUES2 5 QUES3 6 QUES4 7 QUES4B 8 QUES4C 9
   QUES5 10 QUES6 11 QUES7 12 QUES8 13 QUES9 14
   @20 DOB MMDDYY6. @26 ST_DATE MMDDYY6. @32 END_DATE
   MMDDYY6.;
```


The program above, rewritten to use informat lists is:

```
DATA ADVANCED;
INPUT ID  1-3
      @4  (QUES1-QUES4 QUES4B QUES4C QUES5-QUES9)(1.)
      @20 (DOB ST_DATE END_DATE)(MMDDYY6.);
```

Well, it didn't really save much typing, but if you had hundreds or thousands of variables, the savings would be considerable.

The informat list can also contain something called relative column pointers. Using + and − signs, we can move the pointer forward or backward in the record. Next, we show you a novel INPUT statement where relative pointers saved a lot of coding.

A researcher coded 12 systolic and diastolic blood pressures for each subject (the number was reduced for illustrative purposes). They were entered in pairs. A straightforward INPUT statement would be:

```
INPUT ID 1-3 SBP1 4-6 DBP1 7-9 SBP2 10-12 DBP2 13-15 etc. ;
```

A more compact method, using relative pointers is:

```
INPUT ID 1-3 @4 (SBP1-SBP12)(3. +3)
             @7 (DBP1-DBP12)(3. +3);
```

The INFORMAT list (3. + 3) says to read a variable using the 3. INFORMAT and then move the pointer over three spaces. Thus, we "skip over" the diastolic pressures the first time, set the pointer back to column 7, and repeat the same trick with the diastolic pressures.

J. "Holding the Line"—Single- and Double-trailing @'s

There are times when we want to read one variable and, based on its value, decide how to read other variables. To do this, we need more than one INPUT statement. Normally, when SAS finishes an INPUT statement, the pointer is moved to the next line of data. So, if we had more than one INPUT statement, the second INPUT statement would be reading data from the next record. We have two ways to prevent this: the single- and double-trailing @ symbols. A single @ sign, placed at the end of an INPUT statement, means to "hold the line." That is, do not move the pointer to the next record until the end of the DATA step is reached. The double-trailing @ symbol "holds the line strongly." That is, the pointer will not be moved to the next record, even if a DATALINES statement (end of the DATA step) is encountered. It will move to the next record only if there are no more data values to be read on a line.

Here are some examples to help make this clear. In the first example, we ran a survey for 1989 and 1990. Unfortunately, in 1990, an extra question was added, in the middle of the questionnaire (where else!). We want to be able to combine these data in a single file and read each record according to the data layout for that year. The year of the survey was coded in columns 79 and 80 (89 for 1989 and 90 for 1990). In 1989 we had 10 questions in columns 1-10. In 1990 there was an extra question (let's call it 5B) placed in column 6 and questions 6 through 10 wound up in columns 7–11. We will use a trailing @ to read these data:

```
DATA QUEST;
   INPUT YEAR 79-80 @; *** HOLD THE LINE;
   IF YEAR = 89 THEN INPUT @1 (QUES1-QUES10)(1.);
   ELSE IF YEAR = 90 THEN INPUT @1 (QUES1-QUES5)(1.)
      @6 QUES5B 1. @7 (QUES6-QUES10)(1.);
DATALINES;
```

A simple example where a double-trailing @@ is needed is shown next. Suppose we want to read in pairs of X's and Y's and want to place several X Y pairs per line. We could read these data with a double-trailing @:

```
DATA XYDATA;
   INPUT X Y @@;
DATALINES;
1 2 7 9 3 4 10 12
15 18 23 67
;
```

The data set XYDATA would contain six X,Y pairs, namely (1,2), (7,9), (3,4) (10,12), (15,18), and (23,67). Without the double-trailing @, the data set would contain only two X,Y pairs, (1,2) and (15,18).

K. Suppressing the Error Messages for Invalid Data

If invalid data values are read by the SAS system (such as character data in a numeric field or two decimal points in a number), an error message is placed in the SAS Log, the offending record is listed, and a missing value is assigned to the variable. Below is an example of a SAS Log where a character value ('a') was read into a numeric field:


```
   1    data example;
   2    input x y z;
   3    datalines;
NOTE: Invalid data for X in line 5 1-1.
RULE:----+----1----+----2----+----3----+----4----+----5----+----6----+----7
   5 a 3 4
X=. Y=3 Z=4 _ERROR_=1 _N_=2
   6    run;
NOTE: The data set WORK.EXAMPLE has 2 observations and 3 variables.
NOTE: The DATA statement used 2.00 seconds.
```

Although this information can be very useful, there are times where we know that certain fields contain invalid data values, and we want missing values to be substituted for the invalid data. For large files, the SAS Log may become quite large, and the processing time will be longer when these error messages are processed. There are two ways to reduce the error message handling. First, a single question mark placed after the variable name will suppress the invalid data message but still print the offending line of data. Two question marks following the variable name will suppress all error messages and prevent the automatic _ERROR_ variable from being incremented. Here is the INPUT statement to suppress error messages when an invalid value is encountered for X:

```
INPUT X ?? Y Z;
```

If you are using column input, you would write:

```
INPUT X ?? 1-2 Y 3-4 Z 5-6;
```

To allow invalid values for X, Y, and Z, we could write:

```
INPUT @1 (X Y Z)(?? 2.);
```

L. Reading "Unstructured" Data

Almost all the examples in this text have been either small data sets or balanced data sets that were relatively easy to read using standard INPUT statements. However, in the real world, we often encounter data sets that are not so clean. For example, we might have a varying number of records for each subject in a study. Another example would be an unbalanced design where there were different numbers of subjects in each treatment. As these data sets become large, reading them without error sometimes becomes the most difficult part of the data processing problem. The techniques shown in this section will allow you to read almost any type of unstructured data easily.

The key to all the examples that follow is to embed "tags" in the data to indicate to the program what type of data to read. A t-test example with unequal n's and an unbalanced ANOVA will serve to illustrate the use of tags and stream data input.

Example 1. Unbalanced T-test

The amount and complexity of the data have been reduced to make the examples short and easy to follow. The strength of the techniques is their use with larger, more complicated data sets.

We want to analyze an experiment where we had five control and three treatment subjects and we recorded a single variable per subject. The data are shown below:

GROUP	
Control	Treatment
20	40
25	42
23	35
27	
30	

The simplest, most straightforward method to read these data is shown next:

Example 1-A

```
***Traditional INPUT Method;
DATA EX1A;
   INPUT GROUP $ X @@;
DATALINES;
C 20 C 25 C 23 C 27 C 30
T 40 T 42 T 35
;
PROC TTEST DATA=EX1A;
   CLASS GROUP;
   VAR X;
RUN;
```

For larger amounts of data, this program contains some problems. It is tedious and time-consuming to repeat the group identification before each variable to be read. This can be corrected in two ways: First, we can put the information concerning the number of observations per group in the program (Example 1-B) or we can put this information in the data itself (Example 1-C). As mentioned above, if the number of observations were large (several hundred or more), a single mistake in counting would have disastrous consequences.

Example 1-B

```
DATA EX1B;
   GROUP='C';
   DO I=1 TO 5;
      INPUT X @;
      OUTPUT;
```

[Continued]

```
   END;
   GROUP='T';
   DO I=1 TO 3;
      INPUT X @;
      OUTPUT;
   END;
   DROP I;
DATALINES
20 25 23 27 30
40 42 35
;
PROC TTEST DATA=EX1B;
   CLASS GROUP;
   VAR X;
RUN;
```

Example 1-C

```
DATA EX1C;
   DO GROUP='C','T';
      INPUT N;
      DO I=1 TO N;
         INPUT X @;
```

```
         OUTPUT;
      END;
   END;
   DROP N I;
DATALINES;
5
20 25 23 27 30
3
40 42 35
;
PROC TTEST DATA=EX1C;
   CLASS GROUP;
   VAR X;
RUN;
```


The method we are suggesting for large data sets is shown in Example 1-D below:

Example 1-D

```
***Reading the Data with Tags;
DATA EX1D;
   RETAIN GROUP;
   INPUT DUMMY $ @@;
   IF DUMMY='C' OR DUMMY='T' THEN GROUP=DUMMY;
   ELSE DO;
      X=INPUT (DUMMY,5.0);
      OUTPUT;
   END;
   DROP DUMMY;
DATALINES;
C 20 25 23 27 30
T 40 42 35
;
PROC TTEST DATA=EX1D;
   CLASS GROUP;
   VAR X;
RUN;
```

With this program we can add or delete data without making any changes to our program. The three important points in the program are:

1. All data items are read as character and interpreted. If a 'C' or 'T' is found, the variable GROUP is set equal to DUMMY, and the DATA step returns to read the next number. The RETAIN statement prevents the variable

GROUP from being reinitialized to missing each time the INPUT statement reads a new number—it will keep its value of 'C' or 'T' until reset.

2. The INPUT function is used to "read" a character variable with a numeric informat. The INPUT function takes two arguments. The first is the variable to "reread," the second is the informat with which to read that value. Thus, although DUMMY is a character variable, X will be stored as a numeric. We chose the informat 5. since we knew it would be larger than any of our data values.
3. Because there is an OUTPUT statement in the ELSE DO block, the program will not output an observation when DUMMY is equal to a 'C' or a 'T'. Whenever an explicit OUTPUT statement is used in a DATA step, the automatic implied OUTPUT does not occur at the end of the DATA step.

This same program can read data that are not as ordered as Example 1-D. For instance, the data set

```
C 20 25 23 T 40 42
C 30 T 35
```

will also be read correctly. For large data sets, this structure is less prone to error than Examples 1-A through 1-C. (Of course, we pay additional processing costs for the alternative program, but the ease of data entry and the elimination of counting errors is probably worth the extra cost.)

Example 2. Unbalanced Two-way ANOVA.

The next example is an unbalanced design for which we want to perform an analysis of variance. Our design is as follows:

	GROUP		
	A	**B**	**C**
	20	70	90
	30	80	90
M	40	90	80
	20		90
	50		
Gender			
	25	70	20
	30	90	20
F	45	90	30
	30	80	
	65	85	
	72		

The straightforward method of entering these data would be:

```
DATA EX2A;
   INPUT GROUP $ GENDER $ SCORE;
DATALINES;
A M 20
A M 30
  etc.
```


This is a lengthy and wasteful data entry method. For small data sets of this type, we could follow the example of the unbalanced t-test problem and enter the number of observations per cell, either in the program or embedded in the data. A preferable method, especially for a large number of observations per cell where counting would be inconvenient, is shown in Example 2-B below:

```
***First Method of Reading ANOVA Data with Tags;
DATA EX2A;
   DO GENDER='M', 'F';
      DO GROUP='A', 'B', 'C';
         INPUT DUMMY $ @;
         DO WHILE (DUMMY NE '#');
            SCORE=INPUT(DUMMY, 6.0);
            OUTPUT;
            INPUT DUMMY $ @;
         END;
      END;
   END;
   DROP DUMMY;
DATALINES;
20 30 40 20 50 # 70 80 90
# 90 90 80 90 # 25 30 45 30
65 72 # 70 90 90 80 85 # 20 20 30 #
;
PROC GLM DATA=EX2A;
   etc.
```

This program reads and assigns observations to a cell until a "#" is read in the data stream. The program then finishes the innermost loop, and the next cell is selected. We can read as many lines as necessary for the observations in a given cell.

An improved version of this program is shown next (Example 2-B). With this program, we can read the cells in any order, and do not have to supply the program with the cell identification since it is incorporated right in the tags. Let's look over the program first, and then we will discuss the salient features:

```
***More Elegant Method for Unbalanced ANOVA Design;
DATA EX2B;
   RETAIN GROUP GENDER;
   INPUT DUMMY $ @@;
   IF VERIFY (DUMMY,'ABCMF ') = 0 THEN DO;
      GROUP = SUBSTR (DUMMY,1,1);
      GENDER = SUBSTR (DUMMY,2,1);
      DELETE;
   END;
   ELSE SCORE = INPUT (DUMMY,6.);
   DROP DUMMY;
DATALINES;
AM 20 30 40 20 50
BM 70 80 90
```

[Continued]

```
CM 90 80 80 90
AF 25 30 45 30 65 72
BF 70 90 90 80 85
CF 20 20 30
;
PROC GLM DATA=EX2B;
  etc.
```

This program allows us to enter the cells in any order and even use as many lines as necessary for the observations from a cell. This form of data entry is also convenient when we will be adding more data at a later time. The analysis can be rerun without any changes to the program. Additional observations can even be added at the end of the original data.

Special features of this program are the use of the VERIFY and SUBSTR functions. The VERIFY function returns 0 if all the characters in the variable DUMMY can be found as one of the characters in the second argument of the function. Note that a blank is included in argument 2 of the VERIFY function since the length of DUMMY is, by default, equal to eight bytes, which means that it will contain two letters and six blanks. The SUBSTR function picks off the GROUP and GENDER values from the DUMMY string, and the INPUT function converts all character values back to numeric. (See Chapters 17 and 18 for a more detailed discussion of SAS functions.) A CONCLUDING NOTE: One of the authors of this book writes these programs with relative ease. The other author calls him when in need of help. You can't call either one of us. So, be careful about how you structure your data sets.

Problems

12-1. You have five subjects in a placebo group and five in a drug group. One way to structure your data is like this:

GROUP	SCORE
P	77
P	76
P	74
P	72
P	78
D	80
D	84
D	88
D	87
D	90

(a) Write an INPUT statement to read these data, assuming you have one or more spaces between the group designation and the score.

(b) Suppose you prefer to arrange your values on two lines like this:

```
P 77 P 76 P 74 P 72 P 78
D 80 D 84 D 88 D 87 D 90
```

Write an INPUT statement for this arrangement.

(c) This time, the five scores for the placebo group are on the first line and the five scores for the drug group are on another like this:

```
77 76 74 72 78
80 84 88 87 90
```

Write a DATA step to read these data. Be sure the data set contains a GROUP as well as a SCORE variable.

(d) Modify the program in part (c) so that each of the 10 subjects has a subject number from 1 to 10.

12-2. Given the three lines of data:

```
1,3,5,7
2,4,6,8
9,8,7,6
```

Write a SAS DATA step to read these data, assigning the four data values to the variables X1 to X4.

12-3. Given the three lines of data:

```
1,,"HELLO",7
2,4,TEXT,8
9,,,6
```

Write a SAS DATA step to read these data, assigning the four data values to the variables X, Y, C, and Z. Variable C should be a character variable. The double quotes should be stripped off the text strings, and two adjacent commas should be interpreted as containing a missing value.

***12-4.** You are given data values, separated by one or more spaces, representing a patient ID, date of visit, DX code, and cost. Create a SAS data set called OFFICE from these data:

```
1 10/01/96 V075 $102.45
2      02/05/97     X123456789 $3,123
3  07/07/96      V4568
4      11/11/96     A123      $777.
```

NOTES: (1) There is no cost value for patient 3, and the line is not padded with blanks. (2) The maximum length for a DX code is 10. (3) The largest cost is $99,999. (4) Use the INFORMAT DOLLAR8 for the variable cost.

12-5. Given the data layout:

Variable	Starting Column	Ending Column	Type
SUBJECT	1	3	Char
A1	5	5	Char
X	7	8	Num
Y	9	10	Num
Z	11	12	Num

Write a SAS DATA step to read these data using starting and ending column specifications in your INPUT statement. Here are some sample lines of data for you to test your program:

```
         1         2
12345678901234567890
--------------------
A12 X 111213
A13 W 102030
```

12-6. Create a SAS data set, using the same data as above, except use column pointers and INFORMATS instead of starting and ending columns.

***12-7.** Someone gives you the following data layout:

Variable	Start Column	Length	Description
ID	1	3	Num
GENDER	4	1	Char
DOB	10	6	MMDDYY
VISIT	16	6	MMDDYY
DISCHRG	22	6	MMDDYY
SBP1	30	3	Num
DBP1	33	3	Num
HR1	36	2	Num
SBP2	38	3	Num
DBP2	41	3	Num
HR2	44	2	Num
SBP3	46	3	Num
DBP3	49	3	Num
HR3	52	2	Num

Write a SAS DATA step to read the two lines of sample data below. Use variable lists and informat lists, to read these data. See if you can find a way to read the SBP's and DBP's other than the straightforward @30 SBP1 3. @33 DBP1 3. @36 HR1 2. etc. That is, try to read all the SBP's together (SBP1-SBP3)(your informat) and so on.

```
         1         2         3         4         5
1234567890123456789012345678901234567890123456789012345
-------------------------------------------------------
123M     102146111196111396  130 8668134 8872136 8870
456F     010150122596020597  220110822101028424012084
```

***12-8.** Write a SAS DATA step to read the following data:

	Variable	Starting Column	Length	Type
Line 1	ID	1	2	Num
	X	4	2	Num
	Y	6	2	Num
Line 2	A1	3	3	Char
	A2	6	1	Char

Here are some sample data lines on which to test your program:

```
         1         2
12345678901234567890
--------------------
01 2345
  AAAX
02 9876
  BBBY
```

12-9. Write a SAS DATA step to read a series of X,Y pairs where there are several X,Y pairs per line. Each X,Y pair forms one observation. Here are some sample lines of data on which to test your program:

```
1 2  3 4  5 6  7 8
11 12   13 14
21 22  23 24  25 26  27 28
```

***12-10.** You conducted two surveys.

Survey ONE format is:

Variable	Starting Column	Length	Type
ID	1	3	Char
HEIGHT	4	2	Num
WEIGHT	6	3	Num

Survey TWO format is:

Variable	Starting Column	Length	Type
ID	1	3	Char
AGE	4	2	Num
HEIGHT	6	2	Num
WEIGHT	8	3	Num

All lines of data using the ONE format have a '1' in column 12. Lines of data using the TWO format have a '2' in column 12. Create a data set using the sample data below:

```
         1         2
12345678901234567890
--------------------
00168155   1
00272201   1
0034570170 2
0045562 90 2
```

CHAPTER

13

External Files: Reading and Writing Raw and System Files

A. *Introduction*

New SAS users are often confused by the different ways SAS software can read and write data to external files. This is due to the fact that SAS programs can read and write many different types of data files. For example, simple ASCII files (or EBCDIC text files on IBM-compatible mainframes) are read with INFILE and INPUT statements, whereas SAS data sets use two-level SAS data set names and do not require INPUT statements. This chapter discusses several ways that SAS software can read and write a variety of data types. The use of temporary and permanent SAS data sets is discussed and the advantages and disadvantages of each type.

B. *Data in the Program Itself*

Before discussing how to read data from external files, let's review how SAS reads data lines that are part of the program itself, following a DATALINES statement. For example:

```
DATA EX1;
   INPUT GROUP $ X Y Z;
DATALINES;
CONTROL 12 17 19
```


[Continued]

```
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
;
PROC MEANS DATA=EX1 N MEAN STD STDERR MAXDEC=2;
   TITLE 'MEANS FOR EACH GROUP';
   CLASS GROUP;
   VAR X Y Z;
RUN;
```

The INPUT method used, whether column specification, informats, pointers, etc., will not change any of our examples so, for the most part, simple list input is used. The DATALINES statement tells the program that the data lines will follow. The program reads data lines until it encounters a line that ends in a semicolon. The examples in this book use a semicolon on a line by itself (called a null statement) to end instream data. The word DATALINES replaces an older term, CARDS, obviously a throwback to the past when actual punched cards were read into a card reader. However, since your children have probably never even heard of computer cards (maybe you haven't either), the SAS Institute decided that the statement DATALINES was more appropriate than CARDS.

Before we leave this topic, here is one more (and rare) possibility you may encounter. What happens when your data contains semicolons? For example, suppose you had:

```
DATA TEST;
   INPUT AUTHOR $10. TITLE $40.;
DATALINES;
SMITH    The Use of the ; in Writing
FIELD    Commentary on Smith's Book
;
```

The program, recognizing the semicolon in the first line of data, treats the line as a SAS statement and generates more error messages than you can "shake a stick at." The solution to this rare problem is to use the special SAS statement DATALINES4, which requires four semicolons in a row ";;;;" to indicate the end of your data. The corrected example would look like this:

```
DATA TEST;
   INPUT AUTHOR $10. TITLE $40.;
DATALINES4;
SMITH    The Use of the ; in Writing
FIELD    Commentary on Smith's Book
;;;;
```

C. Reading ASCII Data from An External File

It's not unusual to receive data in an external file to be analyzed with SAS software. Whether on a floppy diskette on a microcomputer or on a tape used with a mainframe computer, we want a way to have our SAS program read data from an external source. For this example, we assume that the data file is either an ASCII (American Standard Code for Information Interchange) file or a "card image" file on tape (also called raw data). To read this file is surprisingly easy.

The only changes to be made to a program that reads "instream" data with a DATALINES statement are: (1) Precede the INPUT statement with an INFILE statement. (2) Omit the DATALINES statement, and, of course, the lines of data. An INFILE statement is the way we tell a SAS program where to find external raw data.

If you are running a batch version of SAS software on a platform where JCL (Job Control Language) is needed, the INFILE name will correspond to a DDname (DD stands for Data Definition) which gives information on where to find the file. On MVS batch systems, the DDname is included in the JCL (Job Control Language). On systems such as VM, the DDname is defined with a FILEDEF statement. For any of these mainframe systems, a FILENAME statement can also be used (and is preferrable). For a microcomputer or UNIX system, the INFILE statement can either name a file directly (placed within single quotes) or it can be a fileref defined with a FILENAME statement. We show examples of all of these variations.

```
*----------------------------------------------*
| Personal Computer or UNIX Example            |
| Reading ASCII data from an External Data File |
*----------------------------------------------*;

DATA EX2A;
   INFILE 'B:MYDATA';
   ****This INFILE statement tells the program that
       our INPUT data is located in the file MYDATA
       on a floppy diskette in the B drive;
   INPUT GROUP $ X Y Z;
RUN;

PROC MEANS DATA=EX2A N MEAN STD STDERR MAXDEC=2 ;
   VAR X Y Z;
RUN;
```

File MYDATA (located on the floppy diskette in drive B) looks like this:

```
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
```


An alternative way of writing the INFILE statement in the example above is to use a FILENAME statement to create an alias, or "fileref," for the file (a short "nickname" which we associate with the file). This is shown below:

```
DATA EX2B;
   FILENAME GEORGE 'B:MYDATA';
   INFILE GEORGE;
   ****This INFILE statement tells the program that
       our INPUT data is located in the file MYDATA
       on a floppy diskette in the B drive;
   INPUT GROUP $ X Y Z;
RUN;

PROC MEANS DATA=EX2B N MEAN STD STDERR MAXDEC=2 ;
   VAR X Y Z;
RUN;
```

Note the difference between these two INFILE statements. The first INFILE statement refers to the external file directly, and the filename is placed within single quotes. The second INFILE example defines an alias first with a FILENAME statement and then uses the alias with the INFILE statement. Notice that when we use a fileref it is not in single quotes. This point is important since it is the only way that the program can distinguish between an actual file name and a fileref.

A Mainframe Example Using JCL. The mainframe example shown next is basically the same as the microcomputer example shown above. The only difference is in the way we create the fileref. On an MVS batch system, we would create the fileref with a DD statement in the JCL like this:

```
//JOBNAME JOB (ACCT,BIN),'RON CODY'
//   EXEC SAS
//SAS.GEORGE DD DSN=ABC.MYDATA,DISP=SHR
//SAS.SYSIN DD *
DATA EX2C;
   INFILE GEORGE;
   ***This INFILE statement tells the program that the
      file ABC.MYDATA contains our external data file
      (Assume it is catalogued);
   INPUT GROUP $ X Y Z;
RUN;

PROC MEANS DATA=EX2C N MEAN STD STDERR MAXDEC=2 ;
   VAR X Y Z;
RUN;
```

This example on a VM system would be the same except that a FILEDEF statement would be used to associate the DDname with the file instead of the DD statement in the JCL. Here it is:

```
CMS FILEDEF GEORGE DISK MYDATA DATA B;
***The file MYDATA DATA is on the B minidisk;
DATA EX2D;
   INFILE GEORGE;
   ***This INFILE statement tells the program that the
      data is located in the file with FILENAME MYDATA,
      FILETYPE DATA, and FILEMODE B.;
   INPUT GROUP $ X Y Z;
RUN;

PROC MEANS DATA=EX2D N MEAN STD STDERR MAXDEC=2 ;
   VAR X Y Z;
RUN;
```

These last two examples could also use a FILENAME statement to point to our data source. Whatever method we use, once we know how to create a DDname or a fileref on our particular platform, the SAS statements for reading the files are the same. You will need to refer to your manual on how to create a fileref with TSO or VSE. Again, the SAS statements will not change.

D. INFILE Options

There are a variety of options that can be used with an INFILE statement to control how data are read and to allow the SAS program more control over the input operation. These options are placed after the word INFILE and before the semicolon. We now demonstrate several useful options:

Option END=variable name. This option will automatically set the value of "variable name" to 0 (false) until the INPUT statement has finished reading the last data record. This option can be used when you want to read several different files and combine their data, or when you want to do special processing after you've read the last record in a file. (An alternative is to use the EOF=label option which branches to "label" when the end of file is reached.)

The next example first reads data from a file called OSCAR on a floppy diskette in the B: drive and then from a file called BIGBIRD.TXT located in a subdirectory C:\DATA.

```
DATA EX2E;
   IF TESTEND NE 1 THEN INFILE 'B:OSCAR' END=TESTEND;
   ELSE INFILE 'C:\DATA\BIGBIRD.TXT';
   INPUT GROUP $ X Y Z;
RUN;

PROC MEANS DATA=EX2E N MEAN STD STDERR MAXDEC=2;
   VAR X Y Z;
RUN;
```


Notice here that we can conditionally execute an INFILE statement, thus giving us complete control over the file-reading operation.

Option MISSOVER. The MISSOVER option is very useful when you have records of different length and have missing values at the end of a record. This is frequently the case when a text file is created with a word processor and the records are not padded on the right with blanks. Suppose our file called MYDATA has a short record and looks like:

```
CONTROL 1 2 3
TREAT 4 5
CONTROL 6 7 8
TREAT 8 9 10
```

The program EX2A or EX2B would have a problem reading the second record of this file. Instead of assigning a missing value to the variable Z, it would go to the next record and read "CONTROL" as the value for Z and print an error message (since CONTROL is not a numeric value). The SAS LOG would also contain a NOTE telling us that SAS went to a new line when the INPUT statement reached past the end of a line. The remainder of the third record would not be read, and the next observation in our data set would be GROUP=TREAT, X=8, Y=9, and Z=10. To avoid this problem, use the MISSOVER option on the INFILE statement. This will set all variables to missing if any record is short. The entire program would look like:

```
DATA EX2F;
   INFILE 'B:MYDATA' MISSOVER;
   INPUT GROUP $ X Y Z;
RUN;

PROC MEANS DATA=EX2F N MEAN STD STDERR MAXDEC=2 ;
   VAR X Y Z;
RUN;
```

Option PAD. When your INPUT statement uses columns or pointers and informats, the option PAD can be used to prevent problems with short records. For example, to read data values from fixed columns from a file called C:\DATA\MYDATA.TXT, you could write:

```
DATA EX2G;
   INFILE 'C:\DATA\MYDATA.TXT' PAD;
   INPUT GROUP $ 1
         X       2-3
         Y       4-5
         Z       6-7;
RUN;
```

```
[Continued]

PROC MEANS DATA=EX2G N MEAN STD STDERR MAXDEC=2 ;
   VAR X Y Z;
RUN;
```

Option LRECL=record-length. You may need to specify your logical record length (the number of columns on a line, roughly speaking) if it exceeds the default value for your system. When in doubt, add the LRECL (this stands for logical record length and is pronounced El-Rec-el) option to the INFILE statement. It will not cause a problem if you specify an LRECL larger than your actual record length. For example, suppose you have an ASCII file on a floppy diskette with 210 characters per line and your system default LRECL is 132. To read this file, you would write the INFILE statement thus:

```
INFILE fileref LRECL=210;
```

There are many other INFILE options that allow you more control over how data is read from external files. They can be found in the SAS Language, Reference Version 6 (see Chapter 1 for complete references).

There will be times where you have data within the SAS program itself (following a DATALINES; statement) and not in an external file, yet you want to use one or more of the INFILE options to control the input data. We can still use these options by specifying a special fileref called DATALINES, followed by any options you wish. Suppose you want to use MISSOVER and you have included the data within the program. You would proceed as follows:

```
DATA EX2H;
   INFILE DATALINES MISSOVER;
   INPUT X Y Z;
DATALINES;
1 2 3
4 5
6 7 8
;
PROC MEANS DATA=EX2H;
 etc.
```

E. Writing ASCII or Raw Data to An External File

We may have reason to have our SAS program write data to an external file in "card image," or ASCII format. Writing raw data to a file would have the advantage of being somewhat "universal" in that most software packages would be able to read it. On most microcomputer systems, as ASCII file could be read by a wordprocessing program, a spread-sheet program, or a data-base management package. Writing raw data to a file is very much like reading data from an external file. We use one statement, FILE, to tell the program where to send the data and another, PUT, to indicate which variables and in what format to write them. Thus, to read raw data files we use the statements: INFILE and INPUT; to write raw data files we use the statements: FILE and PUT. Here is a simple example of a program that reads a raw file (MYDATA), creates new variables, and writes out the new file (NEWDATA) to a floppy disk:


```
DATA EX3A;
  ***This program reads a raw data file, creates a new
     variable, and writes the new data set to another file;
   INFILE 'C:MYDATA';     ***Input file;
   FILE 'A:NEWDATA';      ***Output file;
   INPUT GROUP $ X Y Z;
   TOTAL = SUM (OF X Y Z);
   PUT GROUP $ 1-10 @12 (X Y Z TOTAL) (5.);
RUN;
```

Running this program produces a new file called NEWDATA, which looks like this:

```
         1         2         3
12345678901234567890123456789012
--------------------------------
CONTROL       12   17   19   48
TREAT         23   25   29   77
CONTROL       19   18   16   53
TREAT         22   22   29   73
```

Notice that we can employ any of the methods of specifying columns or formats that are permissible with an INPUT statement, with the PUT statement. In the example above, we specified columns for the first variable (GROUP) and a format (in a format list) for the remaining four variables. This gives us complete control over the structure of the file to be created. It goes without saying that this example will work just the same on a mainframe under MVS or VM, provided that the correct filename statements are issued. Note that on an MVS system, if we are creating a new file, we have to provide all the parameters (such as RECFM, DISP, UNIT, DSN, DCB, etc.) necessary for your system.

F. Creating a Permanent SAS Data Set

Thus far, we have seen how to read raw "card image" data and to write the same type of data to an external file. We now demonstrate how to create a permanent SAS data set.

A SAS data set, unlike a raw data file, is not usable by software other than SAS. It contains not only the data, but such information as the variable names, labels, and formats (if any). Before we show you an example, let's first discuss the pros and cons of using permanent SAS data sets.

First, some cons: As we mentioned, non-SAS programs cannot read SAS data sets. You cannot use a typical editor to read or modify a SAS data set. To read, update, or modify a SAS data set requires using SAS software and either writing a program (for example, to change a data value) or using SAS/FSP(r) (Full Screen Product) to display and/or update an observation. When you write a SAS program or use SAS/FSP to modify a SAS data set, you must keep in mind that the original raw data are not modified, and you can no longer recreate the SAS data set from the raw data without making the modification again. Finally, in the minus column, SAS data sets typically use more storage than the original data set and are usually kept in addition to the original raw data, thus more than doubling the system storage requirements.

With all these negatives, why create permanent SAS data sets? Probably the most compelling reason is speed. Considerable computer resources are required to read raw data and create a SAS data set. If you plan to be running many different analyses on a data set that will not be changing often, it is a good idea to make the data set permanent for the duration of the analyses. SAS data sets are also a good way to transfer data to other users provided they have SAS software available. Knowing the data structure is no longer necessary since all the variables, labels, and formats have already been defined. We will see shortly how to use PROC CONTENTS to determine what is contained in a SAS data set.

Our first example in this section is to write a SAS program which has the data in the program itself, and to create a permanent SAS data set.

```
*-----------------------------------------------------------*
| This program reads data following the datalines statement |
| and creates a permanent SAS data set in a subdirectory    |
| called C:\SASDATA                                         |
*-----------------------------------------------------------*;
LIBNAME FELIX 'C:\SASDATA';

DATA FELIX.EX4A;
   INPUT GROUP $ X Y Z;
DATALINES;
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
;
```

The way we distinguish between temporary and permanent SAS data sets is by the SAS data set name. If we have a two-level name (two names separated by a period), we are defining a permanent SAS data set name. With a single-level SAS data set name, we are defining a temporary data set which will disappear when we exit the SAS environment.

The first part of the two-level name (the part before the period) names a location where the SAS data set will be written. In a PC or UNIX environment, this location is usually a subdirectory. SAS uses a LIBNAME statement to assign an alias or nickname, referred to as a libref in SAS manuals, for this directory. We can have many SAS data sets contained within a single subdirectory.

When this program executes, the data set EX4A will be a permanent SAS data set located in the C:\SASDATA subdirectory. If we look at a list of files in the C: \SASDATA subdirectory, there will be a file called EX4A.SD2. The extension SD2 may be different for different platforms. Note that on any SAS system, the first-level name does not remain with the data set; it is only used to point to a SAS library. The only requirement is that the first-level name match the LIBNAME within a program.

Now that we have created a permanent SAS data set, let's see how to read it and determine its contents.

G. *Reading Permanent SAS Data Sets*

Once we have created a permanent SAS data set, we can use it directly in a procedure, once we have defined a libref. We now show you a SAS program which uses this permanent data set.

```
LIBNAME ABC 'C:\SASDATA';

PROC MEANS DATA=ABC.EX4A N MEAN STD STDERR MAXDEC=3;
   VAR X Y Z;
RUN;
```

You can see right away how useful it is to save SAS data sets. Notice that there is no DATA step at all in the program above. All that is needed is to define a SAS library (where the SAS data set is located) and to use a DATA= option with PROC MEANS to indicate on which data set to operate. First, observe that the libref ABC is not the same name we used when we created the data set. The libref ABC is defined with the LIBNAME statement and indicates that we are using the subdirectory C: \SASDATA. Therefore, the first part of the two-level SAS data set name is ABC. The second part of the two-level name tells the system which of the SAS data sets located in C:\SASDATA is to be used. It is important to remember that we must use the DATA= option with any procedure where we are accessing previously stored SAS data sets, because the program will not know which data set to use (when we create a SAS data set in a DATA step, the system keeps track of the "most recently created data set" and uses that data set with any procedure where you do not explicitly indicate which data set to use with a DATA= option). Just so that we don't shortchange the mainframe users, the same program, written on an MVS system, would look something like this:

```
//GROUCH JOB (1234567,BIN),'OSCAR THE'
//   EXEC SAS
//SAS.ABC DD DSN=OLS.A123.S456.CODY,DISP=SHR
//SAS.SYSIN DD *
PROC MEANS DATA=ABC.EX4A N MEAN STD STDERR MAXDEC=3;
   VAR X Y Z;
/*
//
```

The DDname was defined in the JCL, indicating my SAS data set was stored in the MVS data set called OLS.A123.S456.CODY which was catalogued. On a VM system, the DDname, or first part of a two-level name, is what VM calls the filetype in the general, filename filetype filemode method of defining a file. The filename corresponds to the SAS second-level name. Thus, without even issuing a FILEDEF command, we could write:

```
PROC MEANS DATA=ABC.EX4A N MEAN STD STDERR MAXDEC=3;
```

which is valid as long as we have a filetype of ABC and a filename of EX4A.

H. How to Determine the Contents of a SAS Data Set

As mentioned earlier, we cannot use our system editor to list the contents of a SAS data set. How can we "see" what is contained in a SAS data set? We use PROC CONTENTS. This very useful procedure will tell us important information about our data set: the number of observations, the number of variables, the record length, and an alphabetical listing of variables (which includes labels, length, and formats). As an option, you can obtain a list of variables in the order of their position in the data set. Here are the statements to display the contents of the permanent SAS data set EX4A created above:

```
LIBNAME SUGI 'C:\SASDATA';
PROC CONTENTS DATA=SUGI.EX4A POSITION;
RUN;
```

Output from this procedure is shown below:

```
CONTENTS PROCEDURE

Data Set Name: SUGI.EX4A                      Observations:          4
Member Type:   DATA                           Variables:             4
Engine:        V611                           Indexes:               0
Created:       10:01 Thu, Jul 25, 1996        Observation Length:    32
Last Modified: 10:01 Thu, Jul 25, 1996        Deleted Observations:  0
```



```
[Continued]

Protection:                                   Compressed:    NO
Data Set Type:                                Sorted:        NO
Label:

-----Engine/Host Dependent Information-----

Data Set Page Size:      8192
Number of Data Set Pages:1
File Format:             607
First Data Page:         1
Max Obs per Page:        254
Obs in First Data Page:  4

-----Alphabetic List of Variables and Attributes-----

#   Variable   Type   Len   Pos
-------------------------------
1   GROUP      Char    8     0
2   X          Num     8     8
3   Y          Num     8    16
4   Z          Num     8    24

-----Variables Ordered by Position-----

#   Variable   Type   Len   Pos
-------------------------------
1   GROUP      Char    8     0
2   X          Num     8     8
3   Y          Num     8    16
4   Z          Num     8    24
```

One final point of information, the DATA= option of PROC CONTENTS can be used to list all the SAS data sets contained in a SAS library instead of a single data set. Use the form libref._ALL_ instead of libref.data_set_name. This will display all the SAS data sets stored in the library referred to by the libref.

If you are working in an interactive, windows type environment, the two commands DIR and VAR (or pointing and clicking appropriately) will show you any permanent SAS data set and a variable list.

I. Permanent SAS Data Sets with Formats

One special note is needed to caution you about saving permanent SAS data sets in which you have assigned user-created formats to one or more of the variables in the DATA step. If you try to use that data set (in a procedure for example), you will get an error that formats are missing. The important thing to remember is this: If you create a permanent SAS data set which assigns user-created formats to variables, you must make the format library permanent as well. Also, if you give someone else the data set, make sure you give him or her the format library.

To make your format library permanent, add the LIBRARY=libref option to PROC FORMAT. You may either use a special libref called LIBRARY or one of your own choosing. If you choose your own libref, you need to supply a system option called FMTSEARCH=(libref) to tell the program where to look for user-defined formats. Since this sounds rather complicated, we show the code to create a permanent format library and the code to access a permanent data set where formats were used.

Code to Create a Permanent Format Library and Assign the Format to a Variable:

```
LIBNAME FELIX 'C:\SASDATA';
OPTIONS FMTSEARCH=(FELIX);
***We will place the permanent SAS data sets and the
   formats in C:\SASDATA;
PROC FORMAT LIBRARY=FELIX;
   VALUE $XGROUP 'TREAT'='TREATMENT GRP'
                 'CONTROL'='CONTROL GRP';
RUN;

DATA FELIX.EX4A;
   INPUT GROUP $ X Y Z;
   FORMAT GROUP $XGROUP.;
DATALINES;
CONTROL 12 17 19
TREAT 23 25 29
CONTROL 19 18 16
TREAT 22 22 29
;
```

Program to Read a Permanent SAS Data Set with Formats

```
LIBNAME C 'C:\SASDATA';
OPTIONS FMTSEARCH=(C) ;
***Tell the program to look in C:\SASDATA for user
   defined formats;
PROC PRINT DATA=C.EX4A;
RUN;
```

In this example, the libref FELIX was used when the SAS data set and the permanent SAS format was created. A different libref, C, was used in the subsequent program when the data set was accessed. This was for illustrative purposes only. In practice, most SAS programmers use the same libref to point to a specific subdirectory.

If someone gives you a SAS data set that has user formats defined and does not give you the format library, don't despair! You can at least get the procedures to work if you use the system option NOFMTERR. This option will allow you to process the data set that contains missing formats by supplying SAS system defaults to the character and numeric variables.

J. Working with Large Data Sets

Special consideration is necessary when we process large data sets. Of course, "large" is a relative term. On a small microcomputer, 50,000 observations with 50 variables might be considered large. On a mainframe, users frequently process data sets with millions of observations. A few simple techniques described here can reduce the processing time and memory usage (and cost if you're paying for it) for processing a large file.

1. Don't read a file unnecessarily. For example:
   Inefficient Way:

```
LIBNAME INDATA 'C:\MYDATA';
DATA TEMP;
   SET INDATA.STATS;
RUN;

PROC PRINT;
   VAR X Y Z;
RUN;
```

   Efficient Way:

```
LIBNAME INDATA 'C:\MYDATA';
PROC PRINT DATA=INDATA.STATS;
   VAR X Y Z;
RUN;
```

The DATA step in the "Inefficient" example is unnecessary. It simply copies one data set into another so that the PROC PRINT can use the default, most recently created data set. Surprisingly, this is a common error.

2. Drop all unnecessary variables. Not only do more variables take up more space, they slow down DATA step processing as well.
   Inefficient Way (if all you want is the quiz average):

```
DATA QUIZ;
   INPUT @1 (QUIZ1-QUIZ10) (3.);
   QUIZAVE = MEAN (OF QUIZ1-QUIZ10);
DATALINES;
```

   Efficient Way:

```
DATA QUIZ;
   INPUT @1 (QUIZ1-QUIZ10) (3.);
   QUIZAVE = MEAN (OF QUIZ1-QUIZ10);
   DROP (QUIZ1-QUIZ10);
DATALINES;
```

3. Use a DROP (or KEEP) option on the SET statement rather than a DROP (or KEEP) statement in the DATA step. When a DROP option is used, only the variables still in the data set will be brought into the Program Data Vector, which can result in a significant decrease in processing time.
   Inefficient Way:

```
DATA NEW;
   SET OLD;
DROP X1-X20 A B;
etc.
```

   Efficient Way:

```
DATA NEW
   SET OLD (DROP=X1-X20 A B);
etc.
```

4. Do not sort data sets more than necessary. For example, if you need your data in DAY order and know that later in the program you need it in DAY-HOUR order, do the two-level sort first.
   Inefficient Way:

```
PROC SORT DATA=MYDATA;
   BY DAY;
RUN;
etc.
PROC SORT DATA=MYDATA
   BY DAY HOUR;
RUN;
etc.
```

   Efficient Way:

```
PROC SORT DATA=MYDATA;
   BY DAY HOUR;
RUN;
etc.
```

5. Think about using a CLASS statement instead of a BY statement with PROC MEANS. This will eliminate the need to sort the data but will require more memory to run.

   Inefficient Way:

```
PROC SORT DATA=MYDATA;
   BY DAY;
RUN;

PROC MEANS DATA=MYDATA N MEAN NWAY;
   BY DAY;
   VAR . . . ;
RUN;
```

   Efficient Way:

```
PROC MEANS DATA=MYDATA N MEAN NWAY;
   CLASS DAY;
   VAR . . . ;
RUN;
```

6. When a small subset is selected from a large file, use the WHERE statement instead of a subsetting IF statement.
   Inefficient Way:

```
DATA ALPHA;
   SET BETA;
   IF X GE 20;
RUN;
```

   Efficient Way:

```
DATA ALPHA;
   SET BETA;
   WHERE X GE 20;
RUN;
```

   or

```
DATA ALPHA;
   SET BETA(WHERE=(X GE 20));
RUN;
```

7. Use a WHERE statement in a PROC when you only need to run a single procedure on a subset of the data.
Inefficient Way:

```
DATA TEMP DATA=MYDATA;
   SET OLD;
   WHERE AGE GE 65;
RUN;

PROC MEANS DATA=MYDATA N MEAN STD;
   VAR . . . ;
RUN;
```

Efficient Way:

```
PROC MEANS DATA=MYDATA N MEAN STD;
   WHERE AGE GE 65;
   VAR . . . ;
RUN;
```

or

```
PROC MEANS DATA=MYDATA(WHERE=(AGE GE 65)) N MEAN STD;
   VAR . . . ;
RUN;
```

8. Use ELSE IF instead of multiple IF statements.
Inefficient Way:

```
DATA SURVY;
   INPUT ID AGE HEIGHT WEIGHT;
   IF 0 LE AGE LT 20 THEN AGEGRP=1;
   IF 20 LE AGE LT 30 THEN AGEGRP=2;
   IF 30 LE AGE LT 40 THEN AGEGRP=3;
   IF AGE GE 40 THEN AGEGRP=4;
RUN;
```


Efficient Way:

```
DATA SURVY;
   INPUT ID AGE HEIGHT WEIGHT;
   IF 0 LE AGE LT 20 THEN AGEGRP=1;
   ELSE IF 20 LE AGE LT 30 THEN AGEGRP=2;
   ELSE IF 30 LE AGE LT 40 THEN AGEGRP=3;
   ELSE IF AGE GE 40 THEN AGEGRP=4;
RUN;
```

9. When using multiple IF statements, place first the one most likely to be true.
Inefficient Way (most of the subjects are over 65):

```
DATA SURVEY;
   SET OLD;
   IF 0 LE AGE LT 20 THEN AGEGRP=1;
   ELSE IF 20 LE AGE LT 30 THEN AGEGRP=2;
   ELSE IF 30 LE AGE LT 40 THEN AGEGRP=3;
   ELSE IF AGE GE 40 THEN AGEGRP=4;
RUN;
```

Efficient Way (most of the subjects are over 65):

```
DATA SURVEY;
   SET OLD;
   IF AGE GE 40 THEN AGEGRP=4;
   ELSE IF 30 LE AGE LT 40 THEN AGEGRP=3;
   ELSE IF 20 LE AGE LT 30 THEN AGEGRP=2;
   ELSE IF 0 LE AGE LT 20 THEN AGEGRP=1;
RUN;
```

10. Save summary statistics in a permanent SAS file if you plan to do further computations with it.
Inefficient Way:

```
LIBNAME C 'C:\MYDATA';
PROC MEANS DATA=C.INDATA;
   CLASS RACE GENDER;
   VAR . . .
RUN;
```

Efficient Way:

```
LIBNAME C 'C:\MYDATA';
PROC MEANS DATA=C.INDATA NWAY;
   CLASS RACE GENDER;
   VAR . . . ;
   OUTPUT OUT=C.SUMMARY MEAN= ;
RUN;
```

11. Use _NULL_ as a data set name when you only want to process records from a file (such as data cleaning) but do not want to keep the resulting data set.
    Inefficient Way:

```
DATA TEMP;
   SET OLD;
   FILE 'ERRORS';
   IF AGE GT 110 THEN PUT ID= AGE= ;
RUN;
```

Efficient Way:

```
DATA _NULL_;
   SET OLD;
   FILE 'ERRORS';
   IF AGE GT 110 THEN PUT ID= AGE= ;
RUN;
```

12. Save your SAS system files if you plan to do further processing of the data. Reading a system file is much more efficient than reading raw data.
13. Use "OPTIONS OBS=n", where n is either zero or a small number when testing your code. Remember to set it back with "OPTIONS OBS=MAX" before you do any more processing. BE VERY CAREFUL WITH THIS OPTION. You may wind up replacing an existing SAS data set with an empty one.
14. Use PROC DATASETS to rename variables or change variable labels.
15. Use PROC APPEND to add new data to a large file.

Inefficient Way:

```
DATA COMBINE;
   SET BIGFILE NEWFILE;
RUN;
```

Efficient Way:

```
PROC APPEND BASE=BIGFILE DATA=NEWFILE;
RUN;
```

Problems

13-1. You receive a text (ASCII) file, called FRODO, on a floppy diskette. The data layout is as follows:

Variable	Col(s)
ID	1-3
AGE	5-6
HR	8-10
SBP	12-14
DBP	16-18

A few sample records are shown below:

```
123456789012345678        (Columns listed here,
------------------         not on the diskette)
001 56  64 130  80
002 44  72 180             Note: No DBP recorded for
003 64  78 140  88         this ID (short record)
```

You place this diskette in the A: drive of your computer. Write a SAS program that will read this file and do the following:

(a) Create a permanent SAS data set called BILBO on the floppy diskette in drive A. This data set should contain AGE, HR, SBP, and AVEBP, where AVEBP is defined as two-thirds of the diastolic blood pressure (DBP) plus one-third of the systolic blood pressure (SBP). (This is actually a weighted average of the DBP and SBP with weights of 2/3 and 1/3, since the heart spends more time in diastole than systole.)

(b) Create another SAS system file called HIBP, which contains only records of subjects with AVEBP greater than or equal to 100.

13-2. You are given a diskette with two files: SURVEY.SD2 and FORMATS.SC2. The former is a SAS system file (compatible with your version of SAS software), and FORMATS.SC2 is a user-defined format library. You copy these two files to a subdirectory called C:\SASDATA on your hard disk. Two variables of special interest in this data set are ICD_9 and AGE. The format library contains a format for ICD_9, and that format has been assigned in the data set to ICD_9. Write a SAS program, for a computer system you use (mainframe MVS, VM, Windows, Windows95, UNIX, etc.) that will read that data set, recognize the format library, and produce a frequency distribution of the ICD_9 codes in decreasing order of frequency (PROC FREQ option ORDER = FREQ). Also, compute descriptive statistics for AGE (n, mean, standard deviation, standard error, minimum, and maximum).

13-3. You have collected demographic data for the years 1996 and 1997. The data for 1996 is placed in a file called DEM_1996, and the data for 1997 is placed in a file called DEM_1997. These two files use the same data layout (see below). Both files are located on a floppy disk that you place in the A: drive of your computer. Write a program to read all the data from both files, and create a single, permanent SAS data set to be located in C:\MYDATA. Call the SAS data set DEM_9697. The data layout for both files is:

Variable	Starting Column	Length	Type
ID	1	3	Char
AGE	4	2	Num
JOB_CODE	6	1	Char
SALARY	7	6	Num

13-4. You have a raw data file called SAMPLE.DTA on a floppy disk. The file contains 100 numbers per line, and the values are separated by one or more spaces. The length of the longest line is 320 bytes, and some lines contain fewer than 100 numbers. Write a SAS DATA step that will read this file from the floppy disk in drive B: and will assign the 100 values to the variables X1-X100. Assume that the default logical record length for your system is less than 320 bytes.

13-5. Run the program below to create a SAS data set called MILTON. Next, write a SAS DATA step that will read the values from MILTON and write the data for variables A, B, and C to a raw data file to the subdirectory C:\MYDATA. Call the output data file OUTDATA, and write the values for A, B, and C to columns 1–3, 4–6, and 7–9, respectively.

```
***DATA step to create MILTON;
DATA MILTON;
   INPUT X Y A B C Z;
DATALINES;
1 2 3 4 5 6
11 22 33 44 55 66
;
```

CHAPTER

14

Data Set Subsetting, Concatenating, Merging, and Updating

A. Introduction

This chapter covers some basic data set operations. Subsetting is an operation where we select a subset from one data set to form another. We may also want to combine data from several data sets into a single SAS data set; this chapter explores several ways of doing this. Let's take these topics one at a time.

B. Subsetting

We have already seen some examples of data subsetting. The key here is the SET statement which "reads" observations from a SAS data set to form a new SAS data set. As we process the observations from the original data set, we can modify any of the values, create new variables, or make a decision whether to include the observation in the new data set. A simple example:

```
DATA WOMEN;
   SET ALL;
   IF GENDER = 'F';
RUN;
```

In this example, data set ALL contains a variable, GENDER, which has values of 'M' and 'F'. The IF statement, used in this context, is called a subsetting IF. To those of you familiar with other programming languages, this is a "funny" looking IF statement-there is no THEN clause. If the IF statement is NOT true, the observation will be deleted and the DATA step will return to the top. If the IF statement is true, the DATA step will continue in the normal fashion. Therefore, any observation with GENDER equal to 'F' will be written to the new data set WOMEN. You could also write (assuming you have values of 'M' and 'F' in your data set):

```
IF GENDER NE 'M';
```

Be careful here. If there are any observations with missing or miscoded values (i.e., any value that is not an 'M') for GENDER, the statement above would add those observations to data set WOMEN. It is usually better to indicate what you want, rather than what you do not want.

We can use any logical expression in the IF statement to subset the data set. For example, we could have:

```
DATA OLDWOMEN;
   SET ALL;
   IF GENDER = 'F' AND AGE > 65;
RUN;
```

With the release of version 6 of SAS software, an alternative to the subsetting IF statement, called the WHERE statement, became available. Although there are several subtle differences between using IF and WHERE statements, we can subset a data set just as easily by substituting WHERE for IF in the programs above. When the data set we are creating is a small subset of the original data set, the WHERE statement is usually faster and more efficient. In addition, we also have the option of using the WHERE statement in a SAS PROCEDURE. So, to compute frequencies of RACE and INCOME only for females, we could write:

```
PROC FREQ DATA=ALL;
   WHERE GENDER = 'F';
   TABLES RACE INCOME;
RUN;
```

If we want to run a procedure for all levels of a variable, we should use a BY statement instead. We have found the WHERE statement particularly useful when we run t-tests or ANOVAs and we want to eliminate one or more groups from the analysis. Suppose, the variable GROUP has three levels (A, B, and C) and we want to run a t-test between groups A and B. Using the WHERE statement greatly simplifies the job:

```
PROC TTEST DATA=data_set_name;
   WHERE GROUP='A' OR GROUP='B';
   CLASS GROUP;
   VAR . . . ;
RUN;
```


C. Combining Similar Data from Multiple SAS Data Sets

Assume that we have several SAS data sets, each containing the same variables. To create a SAS data set from multiple SAS data sets, use the SET statement to name each of the data sets to be combined. For example, to combine the data from two SAS data sets MEN and WOMEN into a single data set called ALLDATA, the following program could be used:

```
DATA ALLDATA;
  SET MEN WOMEN;
RUN;
```

Data set ALLDATA will contain all the observations from the data set MEN, followed by all the observations from the data set WOMEN.

D. Combining Different Data from Multiple SAS Data Sets

In this section, we demonstrate how to combine different information from multiple SAS data sets. Suppose we have a master student data set that contains Social Security numbers (SS) and student names (NAME). We then give a test and create a data set that contains student SS numbers and test scores. We now want to print out a list of student numbers, names, and scores. Below are sample master and test listings:

SS	Name	
123-45-6789	CODY	
987-65-4321	SMITH	
111-22-3333	GREGORY	Master Data
222-33-4444	HAMER	
777-66-5555	CHAMBLISS	

SS	Score	
123-45-6789	100	
987-65-4321	67	Test Data
222-33-4444	92	

To merge the student names in the MASTER data set with the SS numbers in the TEST data set, we first sort both data sets by SS:

```
PROC SORT DATA=MASTER;
  BY SS;
RUN;

PROC SORT DATA=TEST;
  BY SS;
RUN;
```

To merge these two data sets into a new data set, we use a MERGE statement:

```
DATA BOTH;
   MERGE MASTER TEST;
      BY SS;
   FORMAT SS SSN11.;
RUN;
```

Since the MASTER and TEST data sets are now sorted by SS, the MERGE operation will attempt to combine observations from each of the two data sets when the SS has the same value in each observation. Let's look at the observations from these two data sets, in sorted order, side by side:

Data Set MASTER

SS	Name
111-22-3333	GREGORY
123-45-6789	CODY
222-33-4444	HAMER
777-66-5555	CHAMBLISS
987-65-4321	SMITH

Data Set TEST

SS	Score
123-45-6789	100
222-33-4444	92
987-65-4321	67

The first SS number in data set MASTER is not found in data set TEST. Therefore, when the MERGE takes place, the first observation in data set BOTH will have a missing value for SCORE. The next observation in MASTER has a SS number of 123–45–6789. This number is found in both data sets, so the second observation in data set BOTH will contain a value of 123–45–6789 for SS, the NAME 'CODY' and a SCORE of 100. This process continues until all the observations in data sets MASTER and TEST have been processed. The resulting data set BOTH will have the following observations:

Data Set BOTH

SS	Name	Score
111-22-3333	GREGORY	.
123-45-6789	CODY	100
222-33-4444	HAMER	92
777-66-5555	CHAMBLISS	.
987-65-4321	SMITH	67

Most likely, we would like the merged data set to contain only those observations for which there was a test score. The data set option IN=logical_variable following either (or both) of the data set names listed in the MERGE statement gives us control over which observations are added to the merged data set. The value of logical_variable will be true (1) if that data set has made a contribution to (has a nonmissing value for the BY variable) the current observation being created. If not, it has a value of false (0). The logical_variable created by the IN= data set option is

a temporary variable that can be used anywhere in the DATA step but is not added to the new data set. Let's see how we can use the IN= data set option to omit those observations in data set MASTER who did not take the test. We modify the program above as follows:

```
DATA BOTH;
   MERGE MASTER TEST (IN=FRODO);
      BY SS;
   IF FRODO;
   FORMAT SS SSN11.;
RUN;
```

The IN= option following the data set name TEST creates the logical variable FRODO. To limit the merged data set to only those students in the TEST data set, we use a subsetting IF statement to make sure that the student had an observation in the TEST data set. The resulting merged data set (BOTH) is:

Data Set BOTH

SS	Name	Score
123-45-6789	CODY	100
222-33-4444	HAMER	92
987-65-4321	SMITH	67

Suppose there were observations in the TEST data set without a corresponding one in the MASTER data set. How would we use the IN= data set options to create a merged data set that only contained observations where there was a contribution from both of the data sets? We could use an IN= option for each of the data sets and test that both logical variables were true, like this:

```
DATA BOTH;
   MERGE MASTER (IN=BILBO) TEST (IN=FRODO);
      BY SS;
   IF BILBO AND FRODO;
   FORMAT SS SSN11.;
RUN;
```

The general syntax for the MERGE statement is:

```
MERGE data_set_one (IN=var_name) data_set_two (IN=var_name);
   BY match_vars;
```

In this syntax example, data_set_one and data_set_two are the two data sets to be merged; the IN= option that follows each data set name, if used, can control which observations will be included in the merged data set. The BY statement will tell the program how to select observations from the two data sets.

If the BY variable has a different variable name in one of the data sets, you can use a RENAME option in the MERGE statement. For example, if SS were called ID in the TEST data set, we would write:

```
MERGE MASTER TEST (IN=INTEST RENAME=(ID=SS));
```

This brings in an observation from data set TEST and renames ID to SS for purposes of the MERGE. Note that the variable name in the data set TEST remains ID.

We should mention that MERGE can be used without a BY statement. When that is done, the observations are combined in the order they appear in the two data sets. **(This is extremely risky and we recommend that you never do it.)**

E. *"Table Look Up"*

This section explores some other ways that merging can be used to perform a "table look up." By table look up, we mean that one can pull information from a data set based on one or more criteria and add that information to the current data set. Some simple examples will make this clear. We have one data set which contains ID numbers, YEAR, and white blood count (WBC). Some sample observations are shown here:

ID	Year	WBC
1	1940	6000
2	1940	8000
3	1940	9000
1	1941	6500
2	1941	8500
3	1941	8900

Data set WORKER

Next, we have a data set that tells us the benzene exposure for these subjects for each year.

Year	Exposure
1940	200
1941	150
1942	100
1943	80

Data set EXP

What we want is to add the correct exposure to each observation in the WORKER data set. The SAS statements to perform the merge are:

```
PROC SORT DATA=WORKER;
   BY YEAR;
RUN;

PROC SORT DATA=EXP;
   BY YEAR;
RUN;
```


[Continued]

```
DATA COMBINE;
   MERGE WORKER (IN=INWORK) EXP;
      BY YEAR;
   IF INWORK;
RUN;
```

The resulting data set, COMBINE is shown next:

ID	Year	WBC	Exposure
1	1940	6000	200
2	1940	8000	200
3	1940	9000	200
1	1941	6500	150
2	1941	8500	150
3	1941	8900	150

Data set COMBINE

We now extend this problem to include two BY variables. We want to assign an exposure based on the YEAR and the WORK assignment. Our look up table consists of years, work codes, and exposures. Here is the look up table:

Year	Work	Exposure
1940	MIXER	190
1940	SPREADER	200
1941	MIXER	140
1941	SPREADER	150
1942	MIXER	90
1942	SPREADER	100
1943	MIXER	70
1943	SPREADER	80

Data set EXP

The WORKER data set now contains the YEAR, WORK code, and WBC counts:

ID	Year	Work	WBC
1	1940	MIXER	6000
2	1940	SPREADER	8000
3	1940	MIXER	9000
1	1941	MIXER	6500
2	1941	MIXER	8500
3	1941	SPREADER	8900

Data set WORKER

To add the correct exposure to each observation in the WORKER data set, we have to "look up" the exposure for the correct YEAR and WORK code. A MERGE statement with two BY variables will accomplish this for us:

```
PROC SORT DATA=WORKER;
   BY YEAR WORK;
RUN;
```

[Continued]

```
PROC SORT DATA=EXP;
   BY YEAR WORK;
RUN;

DATA COMBINE;
   MERGE WORKER (IN=INWORK) EXP;
      BY YEAR WORK;
   IF INWORK;
RUN;
```

The merged data set (COMBINE) is shown next:

ID	Year	Work	WBC	Exposure
1	1940	MIXER	6000	190
3	1940	MIXER	9000	190
2	1940	SPREADER	8000	200
1	1941	MIXER	6500	140
2	1941	MIXER	8500	140
3	1941	SPREADER	8900	150

F. Updating a Master Data Set from An Update Data Set

Many business applications contain a master data set that needs to be updated with new information. For example, we might have part numbers and prices in the master data set. The update data set would contain part numbers and new, updated prices. Typically, the update data set would be smaller than the master data set and contain observations only for those part numbers with new prices. In addition, the update data set may contain part numbers that are not present in the master data set. Here is an example:

MASTER data set

PART_NO	Price
1	19
4	23
6	22
7	45

UPDATE data set

PART_NO	Price
4	24
5	37
7	.

We sort both data sets by PART_NO and then perform the UPDATE:

```
DATA NEWMASTR;
   UPDATE MASTER UPDATE;
      BY PART_NO;
RUN;
```


The result is:

NEWMASTR data set

PART_NO	Price
1	19
4	24
5	37
6	22
7	45

Note that the missing value in UPDATE does NOT replace the corresponding observation in MASTER.

Problems

14-1. You have a file containing gymnastic scores for boys and girls, as follows:

ID	Gender	Age	Vault	Floor	P_BAR
3	M	8	7.5	7.2	6.5
5	F	14	7.9	8.2	6.8
2	F	10	5.6	5.7	5.8
7	M	9	5.4	5.9	6.1
6	F	15	8.2	8.2	7.9

(a) Create a SAS data set called GYM from these data.

(b) Create a subset of these data for males only. Call it MALE_GYM.

(c) Create another subset of GYM for all females greater than or equal to 10 years of age. Call it OLDER_F.

14-2. You have two data files, one from the year 1996 and the other from the year 1997, as follows:

File for 1996 (DATA96)

ID	Height	Weight
2	68	155
1	63	102
4	61	111

File for 1997 (DATA97)

ID	Height	Weight
7	72	202
5	78	220
3	66	105

Create a SAS data set from each file (call them YEAR1996 and YEAR1997, respectively). Combine the data from each data set into a single file (call it BOTH).

14-3. You have a separate file on the children in problem 14-1. This file contains ID numbers, income ranges, and the parents' last name as follows:

ID	Income	L_NAME
3	A	Klein
7	B	Cesar
8	A	Solanchick
1	B	Warlock
5	A	Cassidy
2	B	Volick

Note that there are ID's for which there is no GYM data and vice versa. First, create a SAS data set called MONEY from the data above. Next, merge the two data sets (call the merged data set GYMMONEY), including only those ID's that are present in the GYM data set. Next, print out a list showing ID, last name, gender, and age. Have this list in ID order.

14-4. Combine the GYMMONEY data set from problem 14-3 with the data set BOTH from problem 14-2. Call the resulting data set FREDDY. Include only those ID's with data in both data sets. List the contents of this data set.

14-5. You have a finacial plan based on income range and gender. Using the GYMMONEY data set from problem 14-3, create a new data set called FINAL, which contains all the data from GYMMONEY along with the correct financial plan based on the table below:

Income Range	Gender	Financial Plan
A	M	W
A	F	X
B	M	Y
B	F	Z

Produce a listing of this data set.

14-6. You have some new information on the gymnasts in problem 14-1. Subject 3 now has a 6.7 on P_BAR; subject 5 is now 15 and has scores of 8.1 and 7.2 on VAULT and P_BAR, respectively; subject 7 was incorrectly entered as a male and should be female. Create an update data set of this new information and update the GYM file from problem 14-1. Call the updated data set GYM_2, and provide a listing.

CHAPTER

15

Working with Arrays

A. *Introduction*

SAS arrays are a facility that can reduce the amount of coding in a SAS DATA step. Although often thought of as an advanced programming tool, there are many applications of arrays that can be easily mastered. This chapter demonstrates some of the more common uses of SAS arrays.

One of the most common uses of arrays is to shorten a program that repeats one or more lines of code with the only change being the variable names referenced in each of the lines. You will see that you can write "model" lines, replacing the variable names with array names and, by placing these model lines in a looping structure, you can effectively replace hundreds or thousands or millions or ... lines of code with just a few lines. O.K., we sometimes get carried away with how useful arrays can be!

B. *Substituting One Value for Another for a Series of Variables*

One of the best ways to learn how to use arrays is to first write a few lines of SAS code without them. Once you see the pattern, you can write your array statement, your "model" lines of code, and decide how to place those model lines in a DO loop.

So, for this example and for most of the examples in this chapter, you will see some lines of SAS code without using arrays and the corresponding coding using arrays.

For this first example, imagine that you have been given a SAS data set where a value of 999 was entered whenever there was a missing numeric value. (Yes, you guessed it, probably a converted SPSS data set!) Here is a DATA step that converts the values of 999 to missing, without using SAS arrays:

```
*----------------------------------------------------------*
| Example 1: Converting 999 to missing without using an array |
*----------------------------------------------------------*;
DATA MISSING;
   SET OLD;
   IF A = 999 THEN A = .;
   IF B = 999 THEN B = .;
   IF C = 999 THEN C = .;
   IF D = 999 THEN D = .;
   IF E = 999 THEN E = .;
RUN;
```

Do you see a pattern here? Good. Here is the same program using arrays:

```
*----------------------------------------------------------*
|    Example 1: Converting 999 to missing using an array    |
*----------------------------------------------------------*;
DATA MISSING;
   SET OLD;
   ARRAY X[5] A B C D E;

   DO I = 1 TO 5;
      IF X[I] = 999 THEN X[I] = .;
   END;

   DROP I;
RUN;
```

O.K., it's not that much shorter. But, if we had to recode hundreds of variables (or thousands or millions!) you would clearly see the advantage. Here's how the program works:

You first need to create an array to represent the five variables A, B, C, D, and E. You can choose an array name using the same rules you use for SAS variable names. However, be sure not to use the same name for a SAS array as for a variable in the same DATA step. In the example above, the array name is X. Next, in square brackets [ ], curly brackets { }, or parentheses ( ), you enter the number of elements (variables) in the array. You may, if you are "counting challenged" (as is one of the authors), use an asterisk in place of the number. Also note that, on some operating systems and versions of SAS software, parentheses are not acceptable. We usually use, and prefer, square brackets. Following the brackets is a list of SAS variable names. You may list them explicitly or use any of the SAS DATA step conventions that refer to a group of variable names such as the single dash, double dash, or the reserved names _CHARACTER_ or _NUMERIC_. Be forewarned that an array cannot contain a mixture of character and numeric variables. A later example will demonstrate how to create an array of character variables.

Having created your array, you can refer to any of the array elements by using the array name and the appropriate subscript within brackets. In this first example, the element X[3] would represent the third element in the array, C. Just substituting an array element for a variable name in a DATA step would accomplish little. The most common use of an array is to place it in an iterative loop such as a DO loop, a DO WHILE, or a DO UNTIL structure. In the example above, for each of the iterations of the DO loop, you are setting values of 999 to missing for each of the elements of the array and, therefore, each of the variables A through E.

If you are not familiar with DO loops, the syntax is:

```
DO COUNTER = START TO END BY INCREMENT;
   (lines of sas code)
END;
```

The SAS statements between the DO and END statement will be repeated according to the directions specified in the DO statement. In the first example, the index variable I was used as the counter, and the iteration went from 1 to 5. Since the INCREMENT value was omitted, it defaulted to 1. Also, don't forget to DROP DO loop counters in your DATA step.

C. Extending Example 1 to Convert All Numeric Values of 999 to Missing

Here is a useful extension of Example 1 that converts values of 999 to missing for all numeric variables:

```
*----------------------------------------------------------*
| Example 2: Converting 999 to missing for all numeric vars |
*----------------------------------------------------------*;
DATA ALLNUMS;
   SET ALL;
   ARRAY PRESTON[*] _NUMERIC_ ;

   DO I = 1 TO DIM(PRESTON);
      IF PRESTON[I] = 999 THEN
         PRESTON[I] = .;
   END;

   DROP I;
RUN;
```

First, you may wonder where the array name PRESTON came from. That's easy. It is the name of one of the authors' youngest boy, who loves to program (and has visited the SAS Institute). The reserved name _NUMERIC_ refers to all the numeric variables in the data set ALL. Since it might be a lot of trouble to count how many numeric variables there are, you use the asterisk (*) instead of the actual number. The only trick here is that you don't know the ending value for the DO loop. Luckily, the DIM function comes to the rescue. The argument of the DIM function is an array name, and it returns the number of elements in the array.

D. *Converting the Value of N/A (Not Applicable) to a Character Missing Value*

For this problem, you have a SAS data set called OLD where the character string 'N/A' (not applicable) was used in place of a character missing value (a blank). You want to convert the values of 'N/A' to missing for several character variables. As before, here is the program without arrays:

```
*----------------------------------------------------------*
| Example 3: Converting 'N/A' to Missing for character vars |
*----------------------------------------------------------*;
DATA NOTAPPLY;
   SET OLD;
   IF S1 = 'N/A' THEN S1 = ' ';
   IF S2 = 'N/A' THEN S2 = ' ';
   IF S3 = 'N/A' THEN S3 = ' ';
   IF X  = 'N/A' THEN X  = ' ';
   IF Y  = 'N/A' THEN Y  = ' ';
   IF Z  = 'N/A' THEN Z  = ' ';
RUN;
```

And here is the same program using a character array:

```
*----------------------------------------------------------*
| Example 3: Converting 'N/A' to Missing for character vars |
*----------------------------------------------------------*;
DATA NOTAPPLY;
   SET OLD;
   ARRAY RUSSELL[*] $ S1-S3 X Y Z;

   DO J = 1 TO DIM(RUSSELL);
      IF RUSSELL[J] = 'N/A' THEN
         RUSSELL[J] = ' ';
   END;

   DROP J;
RUN;
```


This time, I'm sure you guessed that the array name RUSSELL is the name of another son. (He is a PADI certified SCUBA diver and a musician.) Since you want to declare the array as a character array, you place a dollar sign ($) following the brackets. In this case, since you are reading the observations from an existing SAS data set, the variables S1, S2, S3, X, Y, and Z are already declared as character variables, and you could actually omit the $ in the array statement. However, a good programming practice is always to include a $ in the definition of an array of character variables. Now is a good time to mention that you can also indicate a LENGTH for the elements (if these variables don't already exist) of either a numeric or character array following immediately before the list of variables. In the examples so far, the variables have come from an already created SAS data set and had predefined lengths. To create an array of character variables Q1-Q50 with lengths of one byte, the syntax would be:

```
ARRAY Q[50] $ 1 Q1-Q50;
```

E. *Converting Heights and Weights from English to Metric Units*

For this example, you want to input three heights and five weights in English units and create new variables that represent these same values in metric units.

Here is the program without arrays:

```
*-------------------------------------------------*
| Example 4: Metric conversion without using arrays |
*-------------------------------------------------*;
DATA CONVERT;
   INPUT HT1-HT3 WT1-WT5;
   HTCM1 = 2.54 * HT1;
   HTCM2 = 2.54 * HT2;
   HTCM3 = 2.54 * HT3;
   WTKG1 = WT1 / 2.2;
   WTKG2 = WT2 / 2.2;
   WTKG3 = WT3 / 2.2;
   WTKG4 = WT4 / 2.2;
   WTKG5 = WT5 / 2.2;
DATALINES;
(data goes here)
RUN;
```

And now, the array version:

```
*-------------------------------------------------*
|      Example 4: Metric conversion using arrays    |
*-------------------------------------------------*;
DATA CONVERT;
   INPUT HT1-HT3 WT1-WT5;
   ARRAY HT[3];
   ARRAY HTCM[3];
```

```
[Continued]

   ARRAY WT[5];
   ARRAY WTKG[5];
   *** Yes, we know the variable
       names are missing, read on;

   DO I = 1 TO 5;
      IF I LE 3 THEN
         HTCM[I] = 2.54 * HT[I];
      WTKG[I] = WT[I] / 2.2;
   END;

DATALINES;
(data goes here)
RUN;
```

There are a few things to notice about this program. First, you may have observed that there are no variable names following the array names! (This is a shortcut that you may want to use to impress your boss.) Anytime you write an array statement where there are no variable names listed, the names default to the array name, followed by the range of numbers cited in the brackets. For example, the array statement:

```
ARRAY QUES[3];
```

is equivalent to:

```
ARRAY QUES[3] QUES1-QUES3;
```

Next, you have the problem that the number of heights and weights are not the same. There are several solutions to this problem. The example above used a single DO loop with the index going from 1 to 5. An IF statement inside the loop ensured that the index for the height variables would not exceed 3. An alternative would be for two DO loops, one going from 1 to 3 and another going from 4 to 5. For example:

```
DO I = 1 TO 3;
   HTCM[I] = 2.54 * HT[I];
   WTKG[I] = WT[I] / 2.2;
END;

DO I = 4 TO 5;
   WTKG[I] = WT[I] / 2.2;
END;
```

Pay your money and take your choice!

F. Temporary Arrays

The arrays you have encountered thus far all represent a list of numeric or character variables. There is a special type of array (called a temporary array) which does not actually refer to a list of variables at all! Instead, you can declare an array to be temporary and use the array elements in their subscripted form in the DATA step.

No real variables are created when you use a temporary array. You can also provide initial values for each of the array elements. Let's look at an example first, and then discuss the advantages and disadvantages of temporary arrays.

This first example uses a temporary array to hold the passing scores on five exams. Students' scores are then read and compared to these passing scores, and the number of failed courses is recorded. Now for the program:

```
*---------------------------------------------------------*
| Example 5: Using a temporary array to determine the number |
|   of tests passed                                        |
*---------------------------------------------------------*;
DATA PASSING;

   ARRAY PASS[5] _TEMPORARY_ (65 70 65 80 75);
   ARRAY SCORE[5];

   INPUT ID $ SCORE[*];
   PASS_NUM = 0;

   DO I = 1 TO 5;
      IF SCORE[I] GE PASS[I] THEN
         PASS_NUM + 1;
   END;

   DROP I;
DATALINES;
001 64 69 68 82 74
002 80 80 80 60 80
;
PROC PRINT DATA = PASSING;
   TITLE 'Passing Data Set';
   ID ID;
   VAR PASS_NUM SCORE1-SCORE5;
RUN;
```

The main feature of this program is the array PASS, defined as a temporary array by the key word _TEMPORARY_ following the brackets. Notice the five scores within parentheses following the _TEMPORARY_ key word. These are initial values assigned to the five array elements PASS[1] through PASS[5]. Since we never change them, they do not change. Also, they are automatically retained. It is important to note that the variables PASS1 through PASS5 are not created by this array and do not exist in the data set.

Again, notice that we used the "short-cut" method of defining the SCORE array. As mentioned previously, the array statement:

```
ARRAY SCORE[5];
```

is equivalent to

```
ARRAY SCORE[5] SCORE1-SCORE5;
```

We just never seem to be able to pass up the chance to save a few keystrokes. One final, very important point to make concerning temporary arrays: Remember that the elements of temporary arrays are automatically retained. This is why you can compare each of the scores to PASS[1] through PASS[5] in the DATA step.

The most compelling reason to use temporary arrays is for efficiency. Since they do not create actual data set variables, they do not increase the length of the PDV (program data vector), they are automatically retained, and you do not have to bother dropping useless variables.

G. Using a Temporary Array to Score a Test

This example also uses temporary arrays. Instead of assigning initial values to the array elements when defining the array, the initial values are read as raw data, which causes some interesting problems and innovative solutions. The object of this sample program is to score a ten-question multiple-choice test. The first line of data contains the answer key, and the remaining lines contain student ID's and student answers to the ten questions. Here is a test-scoring program that makes good use of temporary arrays:

```
*-------------------------------------------------*
| Example 6: Using a temporary array to score a test |
*-------------------------------------------------*;
DATA SCORE;

   ARRAY KEY[10] $ 1 _TEMPORARY_;
   ARRAY ANS[10] $ 1;
   ARRAY SCORE[10] _TEMPORARY_;

   IF _N_ = 1 THEN
      DO I = 1 TO 10;
         INPUT KEY[I] @;
      END;

   INPUT ID $ @5 (ANS1-ANS10) ($1.);
   RAWSCORE = 0;
   DO I = 1 TO 10;
      SCORE[I] = ANS[I] EQ KEY[I];
      RAWSCORE + SCORE[I];
   END;

   PERCENT = 100*RAWSCORE/10;
   DROP I;
DATALINES;
A B C D E E D C B A
001 ABCDEABCDE
002 AAAAABBBBB
;
PROC PRINT;
   TITLE 'SCORE Data Set';
   ID ID;
   VAR RAWSCORE PERCENT;
RUN;
```


Let's take it step-by-step. The two arrays, KEY and SCORE are declared to be temporary. In addition, the elements of the KEY array are to be one byte characters. The elements in the KEY array hold the answer key (which will be retained by the nature of temporary array elements), and the elements of the SCORE array will be either 1's (correct answer) or 0's (incorrect answer). The array ANS is not a temporary array, and the variables ANS1-ANS10 will contain the student answers to the ten questions comprising the test.

Unlike the previous example, you do not initialize the values of the KEY array with the ARRAY statement. Instead, you read the values from the first line of data. That's what the group of code starting with IF _N_ = 1 is all about. Since the first line of data is the answer key, this DO group will only execute once for the first line of data. Notice that instead of reading in the data with:

```
INPUT KEY[1] KEY[2] KEY[3] ... KEY[10];
```

The easier (and more generalizable form)

```
DO I = 1 TO 10;
   INPUT KEY[I]@;
END;
```

is used instead. Remember that there are no variables with names KEY1, KEY2, etc., in this data set. Also note that it is not proper to write:

```
INPUT KEY[1]- KEY[10];
```

since the form BASEn-BASEm works only for real variables and not elements of arrays.

For the remaining lines of data, the statement IF _N_ = 1 is false, and the program drops down to the INPUT ID $ ANS1-ANS10; statement. Scoring is performed by the somewhat unusual statement:

```
SCORE[I] = ANS[I] EQ KEY[I];
```

This statement causes the student answer, ANS [I], to be compared to the value of KEY [I]. If they are equal, this part of the statement is true (equal to 1) and SCORE [I] is set equal to 1. Otherwise, SCORE [I] will be set to false (equal to 0). Finally, the statement:

```
RAWSCORE + SCORE[I];
```

accumulates the raw score for each student. You could have used a SUM function outside this loop like this:

```
RAWSCORE = SUM (OF SCORE[1] SCORE[2] ... SCORE[10]);
```

This, like the alternative INPUT statement discussed earlier is not as easy to generalize (for tests of different length) as the structure used here.

For those truly compulsive programmers (like one the authors), you can omit the SCORE array entirely and simply code the following:

```
DO I = 1 TO 10;
   RAWSCORE + (KEY[I] EQ ANS[I]);
END;
```

However, we wanted the excuse to show you a numeric temporary array.

H. Specifying Array Bounds

All of the arrays you have seen thus far had array elements starting from 1. So, for example, if you wanted an array to represent the five variables, YR1993, YR1994, YR1995, YR1996, and YR1997, you could write your array statement like this:

```
ARRAY YR[5] YR1993-YR1997;
```

This is OK, but you would have to remember that YR [1] represents the variable YR1993, YR [2] represents the variable YR1994 and so on. You might like to have the array element YR [1993] associated with the variable YR1993; YR [1994] associated with the variable YR1994 instead. You can specify starting and ending boundaries in your array statement by entering the starting value, a colon, and the ending value within the brackets following the array name. For the problem just discussed, the array statement:

```
ARRAY YR[1993:1997] YR1993-YR1997;
```

gives you just what you want. You separate the lower and upper bounds of the array index by a colon. Another application where specifying array bounds is useful is when you are counting from zero instead of one. For example, you may be measuring heart rate (HR) at times 0, 1, 2, and 3. A convenient array statement would be:

```
ARRAY HR[0:3] HR0-HR3;
```

Now that you see that array bounds do not have to run from 1 to n, you need to rethink the use of the DIM function, which returns the number of elements in an array. If the array starts from 1, the DIM function will also represent the upper bound of the array. However, in the array YR above, the DIM function would return a five! To extract the lower and upper bounds of an array, use the LBOUND and UBOUND functions instead. They return the lower and upper bounds of an array, respectively.

I. Temporary Arrays and Array Bounds

Here is an interesting program that converts plain text to Morse code. The program uses a temporary array to store the Morse equivalents of the letters, and subscripts the elements of the array, starting at 65 since the RANK function, which returns the location of a letter in the ASCII collating sequence, returns a 65 for a capital "A," a 66 for a capital "B," etc. Here then, is the program to convert plain text to Morse code:

```
*-----------------------------------------------------*
| Example 8: Coverting plain text to morse code        |
*-----------------------------------------------------*;
DATA _NULL_;
   FILE PRINT;
   ARRAY M[65:90] $ 4 _TEMPORARY_

   ('.-' '-...' '-.-.' '-..' '.'
    '..-.' '--.' '....' '..' '.---'
    '-.-' '.-..' '--' '-.' '---'
    '.--.' '--.-' '.-.' '...' '-'
```


[Continned]

```
    '..-' '...-' '.--' '-..-' '-.--'
    '--..');

   INPUT LETTER $1. @@;
   LETTER = UPCASE(LETTER);

   IF LETTER EQ ' ' THEN DO;
      PUT ' ' @;
      RETURN;
   END;

   MORSE = M[RANK(LETTER)];
   PUT MORSE @;
DATALINES;
This is a test
;
```

J. Implicitly Subscripted Arrays

Before leaving the topic of arrays, we should mention the alternate type of array which does not explicitly show the subscript when you refer to an array element in the DATA step. This was the original form of the array statement in version 5 that was superseded by the explicit subscript form that we have discussed until now. We strongly recommend that you use the explicit form when writing any new programs. However, since the implicit form is still supported and you may have to maintain older SAS code that contains this type of array, we will briefly show you how it works. The form the ARRAY statement is:

```
ARRAY ARRAYNAME(index_variable) list-of-sas-variables;
```

Length and $ attributes are also available and are placed before the list of SAS variables. When using the array name in a DATA step, the index variable is first set to a value (usually in a DO loop) and the array name is used without a subscript. For example:

```
*-----------------------------------------------------*
| Example 9: demonstrating the older implicit array    |
*-----------------------------------------------------*;
DATA OLDARRAY;
   ARRAY MOZART(I) A B C D E;
   INPUT A B C D E;
   DO I = 1 TO 5;
      IF MOZART = 999 THEN
         MOZART = .;
   END;
DROP I;
DATALINES;
(data lines)
;
```

Notice that inside the DO loop the array name MOZART is used without a subscript. When I = 1, MOZART will represent the variable A; when I = 2, MOZART will represent the variable B; and so forth. A default subscript _I_ is used if no subscript is indicated in the ARRAY statement. The DO loop would then read:

```
DO _I_ = 1 to 5;
```

A very useful form of the DO loop, "DO OVER," is available when the implicit subscript form of the array is used. DO OVER automatically computes the lower and upper bounds of the array and loops over all the elements in the array. The program above can be written using a DO OVER structure like this:

```
*-----------------------------------------------------------*
| Example 10: Demonstrating the Older Implicit ARRAY        |
*-----------------------------------------------------------*;
DATA OLDARRAY;
   ARRAY MOZART A B C D E;
   INPUT A B C D E;
   DO OVER MOZART;
      IF MOZART = 999 THEN
         MOZART = .;
   END;
DATALINES;
(data lines)
;
```

As convenient as this may seem, we still recommend the explicit arrays of version 6.

Yes, you can live without arrays, but a thorough understanding of them can substantially reduce the size of a SAS program. We hope that the examples offered here will give you the courage to try arrays in your next program.

Problems

15-1. Rewrite this program, using arrays:

```
DATA PROB15_1;
   INPUT (HT1-HT5)(2.) (WT1-WT5)(3.);
   DENS1 = WT1 / HT1**2;
   DENS2 = WT2 / HT2**2;
   DENS3 = WT3 / HT3**2;
   DENS4 = WT4 / HT4**2;
   DENS5 = WT5 / HT5**2;
DATALINES;
6862727074150090208230240
64 68 70140   150   170
;
```


15-2. Rewrite the following program, using arrays:

```
DATA OLDMISS;
   INPUT A B C X1-X3 Y1-Y3;
   IF A=999 THEN A=.;
   IF B=999 THEN B=.;
   IF C=999 THEN C=.;
   IF X1=999 THEN X1=.;
   IF X2=999 THEN X2=.;
   IF X3=999 THEN X3=.;
   IF Y1=777 THEN Y1=.;
   IF Y2=777 THEN Y2=.;
   IF Y3=777 THEN Y3=.;
DATALINES;
1 2 3 4 5 6 7 8 9
999 4 999 999 5 999 777 7 7
;
```

15-3. You are given the SAS data set SPEED, created by running the program below. Create a new data set SPEED2 from SPEED, with some new variables. The new variables LX1-LX5 are the natural (base e) logs of the variables X1-X5, and the variables SY1-SY3 are the square roots of the variables Y1-Y3. Use arrays to create the new variables. (Note: See Chapter 17 for how to take a natural log of a number.) In case you don't want to turn to Chapter 17 now, the statement to take the natural log of X and assign the value to a variable LOG_X is: LOG_X = LOG(X);

```
***Program to create the data set SPEED;
DATA SPEED;
   INPUT X1-X5 Y1-Y3;
DATALINES;
1 2 3 4 5 6 7 8
11 22 33 44 55 66 77 88
;
```

15-4. Rewrite the program below, using arrays:

```
DATA PROB15_4;
   LENGTH C1-C5 $ 2;
   INPUT C1-C5 $ X1-X5 Y1-Y5;
   IF C1 = 'NA' THEN C1 = ' ';
   IF C2 = 'NA' THEN C2 = ' ';
   IF C3 = 'NA' THEN C3 = ' ';
   IF C4 = 'NA' THEN C4 = ' ';
   IF C5 = 'NA' THEN C5 = ' ';
   IF X1 = 999 OR Y1 = 999 THEN DO;
      X1 = .; Y1 = .;
END;
```

[Continued]

```
IF X2 = 999 OR Y2 = 999 THEN DO;
   X2 = .; Y2 = .;
END;
IF X3 = 999 OR Y3 = 999 THEN DO;
   X3 = .; Y3 = .;
END;
IF X4 = 999 OR Y4 = 999 THEN DO;
   X4 = .; Y4 = .;
END;
IF X5 = 999 OR Y5 = 999 THEN DO;
   X5 = .; Y5 = .;
END;
DATALINES;
AA BB CC DD EE 1 2 3 4 5 6 7 8 9 10
NA XX NA YY NA 999 2 3 4 999 999 4 5 6 7
;
```


CHAPTER

16

Restructuring SAS Data Sets Using Arrays

A. *Introduction*

This chapter describes how to restructure data sets by using arrays. First, what do we mean by the term restructuring? You may want to create multiple observations from a single observation (or vice versa) for several possible reasons: You may want to create multiple observations from a single observation to count frequencies or to allow for BY variable processing or to restructure SAS data sets for certain statistical analyses. Creating a single observation from multiple observations may make it easier for you to compute differences between values without resorting to LAG functions or perhaps to use the REPEATED statement in PROC GLM.

PROC TRANSPOSE may come to mind as a solution to these transforming problems, but using arrays in a DATA step can be more flexible and allow you to have full control over the transformation process.

B. *Creating a New Data Set with Several Observations per Subject from a Data Set with One Observation per Subject*

Suppose you have a data set called DIAGNOSE, with the variables ID, DX1, DX2, and DX3. The DX variables represent three diagnosis codes. The observations in data set DIAGNOSE are:

Data set DIAGNOSE

ID	DX1	DX2	DX3
01	3	4	
02	1	2	3
03	4	5	
04	7		

As you can see, some subjects have only one diagnosis code, some two, and some all three. Suppose you want to count how many subjects have diagnosis 1, how many have diagnosis 2, and so on. You don't care if the diagnosis code is listed as DX1, DX2, or DX3. In the example here, you would have a frequency of one for diagnosis codes 1, 2, 5, and 7, and a frequency of two for diagnosis codes 3 and 4.

One way to accomplish this task is to restructure the data set DIAGNOSE, which has one observation per subject and three diagnosis variables, to a data set that has a single diagnosis variable and as many observations per subject as there are diagnoses for that subject. This new data set (call it NEW_DX) would look as follows:

Restructured Data Set (NEW_DX)

ID	DX
01	3
01	4
02	1
02	2
02	3
03	4
03	5
04	7

It is now a simple job to count diagnosis codes using PROC FREQ on the single variable DX. Let us first write a SAS DATA step that accomplishes this task and does not use arrays. Here is the code:

```
*-------------------------------------------------------------*
| Example 1A: Creating multiple observations from a single    |
|   observation without using an array                        |
*-------------------------------------------------------------*;
DATA NEW_DX;
   SET DIAGNOSE;
   DX = DX1;
   IF DX NE . THEN OUTPUT;
   DX = DX2;
   IF DX NE . THEN OUTPUT;
   DX = DX3;
   IF DX NE . THEN OUTPUT;
   KEEP ID DX;
RUN;
```

As you read each observation from data set DIAGNOSE, you create from one to three observations in the new data set NEW_DX. The KEEP statement is needed

since you only want the variables ID and DX in the new data set.

Notice the repetitive nature of the program and your array light bulb should "turn on." Here is the program rewritten using arrays:

```
*-------------------------------------------------------------*
| Example 1B: Creating multiple observations from a single    |
|   observation using an array                                |
*-------------------------------------------------------------*;
DATA NEW_DX;
   SET DIAGNOSE;
   ARRAY DXARRAY[3] DX1 - DX3;
   DO I = 1 TO 3;
      DX = DXARRAY[I];
      IF DX NE . THEN OUTPUT;
   END;

   KEEP ID DX;
RUN;
```

In this program, you first create an array called DXARRAY, which contains the three numeric variables DX1, DX2, and DX3. The two lines of code inside the DO loop are similar to the repeated lines in the nonarray example with the variable names DX1, DX2, and DX3 replaced by the array elements. For a more detailed discussion of array processing, refer to the previous chapter.

To count the number of subjects with each diagnosis code, you can now use PROC FREQ like this:

```
PROC FREQ DATA=NEW_DX;
   TABLES DX / NOCUM;
RUN;
```

In this example, you saved only one line of SAS code. However, if there were more variables, DX1 to DX50 for example, the savings would be substantial.

C. Another Example of Creating Multiple Observations from a Single Observation

Here is an example that is similar to Example 1. You start with a data set that contains an ID variable and three variables S1, S2, and S3, which represent a score at times 1, 2, and 3, respectively. The original data set, called ONEPER, looks as follows:

Data Set ONEPER

ID	S1	S2	S3
01	3	4	5
02	7	8	9
03	6	5	4

You want to create a new data set called MANYPER, which looks like this:

Data set MANYPER

ID	Time	Score
01	1	3
01	2	4
01	3	5
02	1	7
02	2	8
02	3	9
03	1	6
03	2	5
03	3	4

The program to restructure data set ONEPER to data set MANYPER is similar to the program in Example 1 except that you need to create the TIME variable in the restructured data set. This is easily accomplished by naming the DO loop counter TIME as follows:

```
*----------------------------------------------------------------*
| Example 2: Creating multiple observations from a single       |
|   observation using an array                                   |
*----------------------------------------------------------------*;
DATA MANYPER;
   SET ONEPER;
   ARRAY S[3];

   DO TIME = 1 TO 3;
      SCORE = S[TIME];
      OUTPUT;
   END;

   KEEP ID TIME SCORE;
RUN;
```

Notice that the ARRAY statement does not have a variable list. This was done to demonstrate another way of writing an array statement. When this list is omitted, the variable names default to the array name, followed by the numbers from the lower bound to the upper bound. In this case, the statement

```
ARRAY S[3];
```

is equivalent to

```
ARRAY S[3] S1-S3;
```

Still going in the direction of creating multiple observations from a single observation, let us extend this program to include an additional dimension.

D. *Going from One Observation per Subject to Many Observations per Subject Using Multi-dimensional Arrays*

Suppose you have a SAS data set (call it WT_ONE) that contains an ID and six weights on each subject in an experiment. The first three values represent weights at times 1, 2, and 3 under condition 1; the next three values represent weights at times 1, 2, and 3 under condition 2. To clarify this, suppose that data set WT_ONE contained two observations:

Data Set WT_ONE

ID	WT1	WT2	WT3	WT4	WT5	WT6
01	155	158	162	149	148	147
02	110	112	114	107	108	109

You want a new data set called WT_MANY to look like this:

Data set WT_MANY

ID	COND	Time	Weight
01	1	1	155
01	1	2	158
01	1	3	162
01	2	1	149
01	2	2	148
01	2	3	147
02	1	1	110
02	1	2	112
02	1	3	114
02	2	1	107
02	2	2	108
02	2	3	109

A convenient way to make this conversion is to create a two-dimensional array, with the first dimension representing condition and the second representing time. So, instead of having a one-dimensional array like this:

```
ARRAY WEIGHT[6] WT1-WT6;
```

You could create a two-dimensional array like this:

```
ARRAY WEIGHT[2,3] WT1-WT6;
```

The comma between the 2 and 3 separates the dimensions of the array. This is a 2 by 3 array. Array element WEIGHT[2,3], for example, would represent a subject's weight under condition 2 at time 3.

Let us use this array structure to create the new data set which contains six observations for each ID. Each observation is to contain the ID and one of the six weights, along with two new variables, COND and TIME, which represent the condition and the time at which the weight was recorded. Here is the restructuring program:

```
*-----------------------------------------------------------*
| Example 3: Using a multi-dimensional array to restructure |
|    a data set                                             |
*-----------------------------------------------------------*;
DATA WT_MANY;
   SET WT_ONE;

   ARRAY WTS[2,3] WT1-WT6;

   DO COND = 1 TO 2;
      DO TIME = 1 TO 3;
         WEIGHT = WTS[COND,TIME];
         OUTPUT;
      END;
   END;

   DROP WT1-WT6;

RUN;
```

To cover all combinations of condition and time, you use "nested" DO loops, that is, a DO loop within a DO loop. Here's how it works: COND is first set to 1 by the outer loop. Next, TIME is set to 1, 2, and 3 while COND remains at 1. Each time a COND and TIME combination is selected, a WEIGHT is set equal to the appropriate array element and the observation is written out to the new data set.

E. *Creating a Data Set with One Observation per Subject from a Data Set with Multiple Observations per Subject*

It's now time to reverse the restructuring process. We will do the reverse of Example 2 to demonstrate how to create a single observation from multiple observations. This time, we start with data set MANYPER and create data set ONEPER. First the program, then the explanation:

```
*-----------------------------------------------------------*
| Example 4A: Creating a data set with one observation per  |
|    subject from a data set with multiple observations per |
|    subject. (Caution: This program will not work if there |
|    are any missing time values.)                          |
*-----------------------------------------------------------*;
PROC SORT DATA=MANYPER;
   BY ID TIME;
RUN;
```


[Continued]

```
DATA ONEPER;
   ARRAY S[3] S1-S3;
   RETAIN S1-S3;
   SET MANYPER;
   BY ID;
   S[TIME] = SCORE;
   IF LAST.ID THEN OUTPUT;
   KEEP ID S1-S3;
RUN;
```

First, you sort data set MANYPER to be sure that the observations are in ID and TIME order. In this example, data set MANYPER is already in the correct order, but the SORT procedure makes the program more general. Next, you need to create an array containing the variables you want in the ONEPER data set, namely, S1, S2, and S3. You can "play computer" to see how this program works. The first observation in data set MANYPER is:

```
ID = 01    TIME = 1    SCORE = 4
```

Therefore, S [TIME] will be S [1], representing the variable S1 and set to the value of SCORE, which is 3. Since LAST.ID is false, the OUTPUT statement does not execute. However, the value of S1 is retained. In the next observation, time is 2 and SCORE is 4, so the variable S2 is assigned a value of 4. Finally, the third and last observation is read for ID 01. S3 is set to the value of SCORE, which is 5 and, since LAST.ID is true, the first observation in data set ONEPER is written. Everything seems fine. Almost.

What if data set MANYPER did not have an observation at all three values of time for each ID? Use the data set MANYPER2 shown next to see what would happen:

Data set MANYPER2

ID	Time	Score
01	1	3
01	2	4
01	3	5
02	1	7
02	3	9
03	1	6
03	2	5
03	3	4

Notice that ID number 02 does not have an observation with TIME=2. What will happen if you run program Example 4A? Since you retained the values of S1, S2, and S3, and never replaced the value of S2 for ID number 02, ID number 02 will be given the value of S2 from the previous subject! Not what you want. You must always be careful when you retain variables. To be sure that this will not happen, you need to set the values of S1, S2, and S3 to missing each time you encounter a new subject. This is readily accomplished by checking the value of FIRST.ID. The corrected program is:

```
*-------------------------------------------------------*
| Example 4B: Creating a data set with one observation per |
|    subject from a data set with multiple observations per |
|    subject (corrected version)                            |
*-------------------------------------------------------*;
PROC SORT DATA=MANYPER2;
   BY ID TIME;
RUN;

DATA ONEPER;
   ARRAY S[3] S1-S3;
   RETAIN S1-S3;
   SET MANYPER2;
   BY ID;
   IF FIRST.ID THEN DO I = 1 TO 3;
      S[I] = .;
      END;
   S[TIME] = SCORE;
   IF LAST.ID THEN OUTPUT;
   KEEP ID S1-S3;
RUN;
```

This program will now work correctly whether or not there are missing TIME values.

F. Creating a Data Set with One Observation per Subject from a Data Set with Multiple Observations per Subject Using a Multi-dimensional Array

This example is the reverse of Example 3. That is, you want to start from data set WT_MANY and wind up with data set WT_ONE. The solution to this problem is similar to Example 4, except that we use a multi-dimensional array. Instead of writing the program in two steps, as we did in Examples 4A and 4B, we present the general solution that will work whether or not there are any missing observations in the data set. Here is the program:

```
*-------------------------------------------------------*
| Example 5: Creating a data set with one observation per |
|    subject from a data set with multiple observations per |
|    subject using a Multi-dimensional array                |
*-------------------------------------------------------*;
```


[Continued]

```
PROC SORT DATA=WT_MANY;
   BY ID COND TIME;
RUN;

DATA WT_ONE;
   ARRAY WT[2,3] WT1-WT6;
   RETAIN WT1-WT6;
   SET WT_MANY;
   BY ID;
   IF FIRST.ID THEN
   DO I = 1 TO 2;
      DO J = 1 TO 3;
         WT[I,J] = .;
      END;
   END;
   WT[COND,TIME] = WEIGHT;
   IF LAST.ID THEN OUTPUT;
   KEEP ID WT1-WT6;
RUN;

PROC PRINT DATA=WT_ONE;
   TITLE 'WT_ONE Again';
RUN;
```

You have seen how to restructure SAS data sets, going from one to many, or from many to one observation per subject, using arrays. You may want to keep these examples handy for the next time you have a restructuring job to be done.

Problems

***16-1.** We have a data set called FROG, which looks like this:

ID	X1	X2	X3	X4	X5	Y1	Y2	Y3	Y4	Y5
01	4	5	4	7	3	1	7	3	6	8
02	8	7	8	6	7	5	4	3	5	6

We want a data set that has an observation for each subject (ID) at each time interval (X1 represents X at time 1, etc.). Write a program, using arrays, to accomplish this objective. The new data set (TOAD) should look like:

ID	Time	X	Y
01	1	4	1
01	2	5	7
01	3	4	3
01	4	7	6
01	5	3	8
02	1	7	5
02	2	7	4
02	3	8	3
02	4	6	5
02	5	7	6

Run the program below to create data set FROG:

```
DATA FROG;
   INPUT ID X1-X5 Y1-Y5;
DATALINES;
01  4  5  4  7  3  1  7  3  6  8
02  8  7  8  6  7  5  4  3  5  6
;
```

***16-2.** We have data set (called STATE) that contains an ID variable, and up to five states (two letter codes) where an individual may have visited last year. Three observations from this data set are shown:

ID	STATE1	STATE2	STATE3	STATE4	STATE5
1	NY	NJ	PA	TX	GA
2	NJ	NY	CA	XX	XX
3	PA	XX	XX	XX	XX

As you can see, "XX" was used as a missing value. Write a program to: (a) read these records and replace the values of "XX" with blanks, and (b) Compute frequency counts showing how many people visited each state. Present the frequency list in decreasing order of frequency (use the ORDER=FREQ option of PROC FREQ). Run the program below to create data set STATE:

```
DATA STATE;
   INFORMAT STATE1-STATE5 $ 2;
   INPUT ID STATE1-STATE5;
DATALINES;
1  NY  NJ  PA  TX  GA
2  NJ  NY  CA  XX  XX
3  PA  XX  XX  XX  XX
;
```

***16-3.** You have inherited an old SAS program (shown below) and want to convert it to one using explicit array subscripts. Rewrite the program to do this.

```
DATA OLDFASH;
   SET BLAH;
   ARRAY JUNK(J) X1-X5 Y1-Y5 Z1-Z5;
   DO OVER JUNK;
      IF JUNK = 999 THEN JUNK=.;
   END;
   DROP J;
RUN;
```

CHAPTER

17

A Review of SAS Functions

Part I. Functions Other Than Character Functions

A. *Introduction*

Throughout this book we have used functions to perform a variety of tasks. We use a LOG function to transform variables, and various DATE functions to convert a date into an internal SAS date. We will see in this chapter that the SAS programming language has a rich assortment of functions that can greatly simplify some complex programming problems. Take a moment to browse through this chapter to see what SAS functions can do for you.

B. *Arithmetic and Mathematical Functions*

The functions you are probably most familiar with are ones that perform arithmetic or mathematical calculations. Remember that all SAS functions are recognized as such because the function name is always followed by a set of parentheses containing one or more arguments. This way, the program can always differentiate between a variable name and a function. Here is a short program that computes a new variable, called LOGLOS, which is the natural log of LOS (length of stay). This is a common way to "pull in the tail" of a distribution skewed to the right. We have:

```
DATA FUNC_EG;
   INPUT ID SEX $ LOS HEIGHT WEIGHT;
   LOGLOS = LOG(LOS);
DATALINES;
```

[Continued]

```
1 M 5 68 155
2 F 100 62 98
3 M 20 72 220
;
```

The new variable (LOGLOS) will be in the data set FUNC_EG, and its values will be the natural (base e) log of LOS. Note that a zero value for LOS will result in a missing value for LOGLOS. When zeros are possible values, and you still want a log transformation, it is common practice to add a small number (usually .5 or 1) to the variable before taking the log.

We now list some of the more common arithmetic and mathematical functions and their purposes:

Function Name	Action
LOG	Base e log
LOG10	Base 10 log
SIN	Sine of the argument (in radians)
COS	Cosine (in radians)
TAN	Tangent (in radians)
ARSIN	Arcsine (inverse sine) of argument (in radians)
ARCOS	Arccosine (in radians)
ARTAN	Arctangent (in radians)
INT	Drops the fractional part of a number
SQRT	Square root

Some functions can accommodate more than one argument. For example, the ROUND function can take two arguments, separated by commas. The first argument is the number to be rounded, and the second argument indicates the round-off unit. Here are some examples:

```
ROUND (X,1)      Round X to the nearest whole number
ROUND (X,.1)     Round X to the nearest tenth
ROUND (X,100)    Round X to the nearest hundred
ROUND (X,20)     Round X to the nearest twenty
```

(NOTE: If you omit the second argument of the ROUND function, it rounds to the nearest integer.)

Other functions operate on a list of arguments. A good example of this is the MEAN function. If we have a series of variables (say X1 to X5) for each subject, and we want the mean of these five numbers, we write:

```
MEAN_X = MEAN (OF X1-X5);
```

We may use any variable list following the word OF. An important difference between the MEAN function and the alternative expression:

```
MEAN_X = (X1 + X2 + X3 + X4 + X5)/5;
```


is that the MEAN function returns the mean of the nonmissing values. Thus, if we had a missing value for X5, the function would return the mean of X1, X2, X3, and X4. Only if all the variables listed as arguments of the MEAN function are missing, will the MEAN function return a missing value. Our equation for the mean above, would return a missing value if any of the X values were missing.

The MIN, MAX, SUM, STD, and STDERR functions work the same way as the MEAN function, except that a minimum, maximum, sum, standard deviation, or a standard error, respectively, is computed instead of a mean.

Two useful functions are N and NMISS. They return, as you would expect, the number of nonmissing (N) or the number of missing (NMISS) values in a list of variables. Suppose we have recorded 100 scores for each subject and want to compute the mean of those 100 scores. However, we want to compute the mean score only if there are 75 or more nonmissing responses. Without the N function, we would have to do a bit of programming. Using the N function, the computation is much simpler:

```
DATA EASYWAY;
   INPUT (X1-X100)(2.);
   IF N(OF X1-X100) GE 75 THEN
      AVE = MEAN(OF X1-X100);
DATALINES;
(lines of data)
;
```

The NMISS function is used in a similar fashion.

C. Random Number Functions

In Chapter 6, we saw how we could use random numbers to assign subjects to groups. The function RANUNI will generate uniform random numbers in the range from 0 to 1. Random-number generators (more properly called pseudo random number generators) require an initial number, called a seed, used to calculate the first random number. From then on, each random number is used in some way to generate the next. A zero seed will cause the function to use a seed derived from the time clock, thus generating a different random series each time it is used. RANUNI can also be seeded with any number of your choosing. If you supply the seed, the function will generate the same series of random numbers each time. A simple example follows where we use a uniform random number to put a group of subjects in random order:

```
DATA SHUFFLE;
   INPUT NAME : $20.;
   X = RANUNI (0);
DATALINES;
```

[Continued]

```
CODY
SMITH
MARTIN
LAVERY
THAYER
;
PROC SORT DATA=SHUFFLE;
   BY X;
RUN;

PROC PRINT DATA=SHUFFLE;
   TITLE 'Names in Random Order';
   VAR NAME;
RUN;
```

To generate a series of random numbers from n to m, we need to scale our 0 to 1 random numbers accordingly. To generate a series of random numbers from 1 to 100, we can write:

```
X = 1 + 99 * RANUNI (0);
```

For purposes of statistical modeling, we might also want a series of random numbers chosen from a normal distribution (mean=0, variance=1). The RANNOR function will generate such variables. The allowable seeds for RANNOR follow the same rules as for RANUNI.

D. Time and Date Functions

We saw some examples of date functions in Chapter 4. We summarize the time and date functions here.

There are several extremely useful date functions. One, MDY (month, day, year) will convert a month, day, and year value into a SAS date variable. Suppose, for example, that we want to know a subject's age as of July 15, 1990, and we know his/her date of birth. We could use the MDY function to compute the age, thus:

```
AGE = (MDY (7, 15, 90) - DOB)/365.25;
```

Although a more efficient method would be to use a SAS date literal, as shown below:

```
AGE = ('15JUL90'D - DOB)/365.25;
```

A SAS date literal is always represented by a two-digit day, a three-letter month, and a two- or four-digit year, all placed within single or double quotes, followed by an upper- or lower-case 'D'.

Another possible use of the MDY function is when date information is not recorded by one of the standard methods for which SAS has a date informat. If we can read the month, day, and year into variables, we can then use the MDY function to compute the date. For example:


```
DATA DATES;
   INPUT ID 1-3 MONTH 4-5 DAY 10-11 YEAR 79-80;
   DATE = MDY (MONTH, DAY, YEAR);
   DROP MONTH DAY YEAR;
   FORMAT DATE MMDDYY8.;
DATALINES;
(data lines)
;
```

There are several date functions that extract information from a SAS date. For example, the YEAR function returns a four-digit year from a SAS date. The MONTH function returns a number from 1 to 12, which represents the month for a given date. There are two functions which return day information. The DAY function returns the day of the month (i.e., a number from 1 to 31) and the WEEKDAY function returns the day of the week (a number from 1 to 7, 1 being Sunday). As an example, suppose we want to see distributions by month and day of the week for hospital admissions. The variable ADMIT is a SAS date variable:

```
PROC FORMAT;
   VALUE DAYWK 1='SUN' 2='MON' 3='TUE' 4='WED' 5='THU'
               6='FRI' 7='SAT';
   VALUE MONTH 1='JAN' 2='FEB' 3='MAR' 4='APR' 5='MAY' 6='JUN'
               7='JUL' 8='AUG' 9='SEP' 10='OCT' 11='NOV' 12='DEC';
RUN;

DATA HOSP;
   INPUT @1 ADMIT MMDDYY6. etc. ;
   DAY = WEEKDAY(ADMIT);
   MONTH = MONTH(ADMIT);
   FORMAT ADMIT MMDDYY8. DAY DAYWK. MONTH MONTH.;
DATALINES;
(data lines)
;
PROC CHART DATA=HOSP;
   VBAR DAY / DISCRETE;
   VBAR MONTH / DISCRETE;
RUN;
```

Later in this chapter, look for a short-cut method for producing the day of the week or month name in the discussion of the PUT function.

Besides working with date values, SAS has a corresponding set of functions to work with "time." For example, we can read a time (in "military" form from 00:00 to 24:00) in hh:mm:ss (hours, minutes, seconds) or hh:mm (hours and minutes) format using the time8. informat. We can then extract hour, minute, or second information from the time variable using the HOUR, MINUTE, or SECOND functions, just the way we used the YEAR, MONTH, and WEEKDAY functions above.

Before we leave the date and time functions, let's discuss two very useful functions, INTCK and INTNX. They may save you pages of SAS coding. INTCK returns the number of intervals between any two dates. Valid, interval choices are: DAY, WEEK, MONTH, QTR, YEAR, HOUR, MINUTE, and SECOND. The syntax of the INTCK function is:

```
INTCK (interval, start, end);
```

Interval is one of the choices above, placed within single quotes; start is the starting date; and end is the ending date. As an example, suppose we want to know now many quarters our employees have worked. If START is the SAS variable that holds the starting date, the number of quarters worked would be:

```
NUM_QTR = INTCK ('QTR', START, TODAY());
```

(NOTE: the TODAY function, which has no argument, returns today's date.)

Since the algorithms used to compute the number of intervals can be confusing, we recommend reading the section on SAS functions in the appropriate SAS manual.

The INTNX function can be thought of as the inverse of the INTCK function; it returns a date, given an interval, starting date, and the number of intervals elapsed. Suppose we know the date of hire and want to compute the date representing the start of the third quarter. You would use:

```
DATE3RD = INTNX ('QTR', HIRE, 3);
FORMAT DATE3RD MMDDYY8.;
```

If HIRE were 01/01/90, 01/05/90, or 03/30/90, the value of DATE3RD would be 10/01/90. If a person were hired on April 1, 1990, his/her third-quarter date would be 01/01/91.

E. The INPUT and PUT Functions: Converting Numerics to Character and, Character to Numeric Variables

While the INPUT and PUT functions (not to be mistaken for INPUT and PUT statements) have many uses, one common application is to convert numeric and character values.

The PUT function uses the formatted value of a variable to create a new variable. For example, suppose we have recorded the AGE of each subject. We also have a format that groups the ages into four groups:

```
PROC FORMAT;
   VALUE AGEGRP LOW-20='1' 21-40='2' 41-60='3' 61-HIGH='4';
RUN;

DATA PUTEG;
   INPUT AGE @@;
   AGE4 = PUT (AGE,AGEGRP.);
DATALINES;
5 10 15 20 25 30 66 68 99
;
```


In this example, the variable AGE4 is a character variable with values of '1','2', '3', or '4'. As another example, suppose we want a variable to contain the three-letter day of the week abbreviations (MON, TUE, etc.). One of the SAS built-in formats is WEEKDATEn. which returns values such as: WEDNESDAY, SEPTEMBER 12, 1990 (if we use WEEKDATE29.). The format WEEKDATE3. is the first three letters of the day name (SUN, MON, etc.). To create our character day variable, we can use the PUT function:

```
DAYNAME = PUT(DATE,WEEKDATE3.);
```

There are some useful tricks that can be accomplished using the PUT function. Consider one file that has social security numbers as nine-digit numerics. Another file has the social security numbers coded as 11-digit character strings (123-45-6789). Our job is to merge the two files, based on the social security numbers. There are many ways to solve this problem, either pulling out the three numbers between the dashes and recombining them to form a numeric, or converting the nine-digit number to a character string and placing the dashes in the proper places using the appropriate string functions. One method is to use the fact that SAS has a built-in format, SSN11., which formats nine-digit numerics to 123-45-6789 style social security numbers. Therefore, using the PUT function, we can create a character variable in the form 123-45-6789, like this:

```
SS = PUT (ID,SSN11.);
```

In general, to convert a numeric variable to a character variable, we can use the PUT function, with the appropriate format. If we have a file where group is a numeric variable and we want a character variable instead, we can write:

```
GROUPCHR = PUT (GROUP,1.);
```

We use the INPUT function in a similar manner, except that we can "reread" a variable according to a new informat. The most common use of the function is to convert character values into numeric values. There are several examples of such conversions in the last section of Chapter 12, Reading "Unstructured" Data. We will show you two examples here.

In the first example, we convert a character representation of a social security number (in the form 123-45-6789) into a nine-digit numeric, the reverse of the PUT example above. First, we have to remove the dashes from the character variable. We use the COMPRESS function to do this. COMPRESS takes the form:

```
char_var = COMPRESS (char_var,'list_of_characters');
```

If the second argument, the list_of_characters is omitted, the COMPRESS function will, by default, remove blanks from a character value. To remove the dashes from a social security number (SS) we write:

```
char_var = COMPRESS(SS,'-');
```

To create a numeric variable, we can use the INPUT function to perform the character to numeric conversion:

```
ID = INPUT (COMPRESS(SS,'-'),9.);
```

where ID is a SAS numeric variable.

For the second example, we want to read data values that may either represent groups ('A' or 'B') or numeric scores. Since we don't know if we will be reading a character or a number, we read every value as a character, and test if it is an 'A' or a 'B'. If not, we assume it is a score and use the INPUT function to convert the character variable to numeric. Here is the code:

```
DATA FREEFORM;
   INPUT TEST $ @@;
   RETAIN GROUP;
   IF TEST = 'A' OR TEST='B' THEN DO;
      GROUP = TEST;
      DELETE;
      RETURN;
   END;
   ELSE SCORE = INPUT (TEST, 5.);
   DROP TEST;
DATALINES;
A 45 55 B 87 A 44 23 B 88 99
;
PROC PRINT DATA=FREEFORM;
   TITLE 'Listing of Data Set FREEFORM';
RUN;
```

To help make this example clearer, the data set formed by running this program is shown below:

OBS	Group	Score
1	A	45
2	A	55
3	B	87
4	A	44
5	A	23
6	B	88
7	B	99

As you can see, the INPUT function provides a flexible way of reading data.

F. The LAG and DIF Functions

A "lagged" value is one from an earlier time. In SAS, we may want to compare a data value from a current observation with a value from a previous observation. We may also want to look back several observations. Without the LAG function, this is a difficult task—with it, it's simple. The value returned by the LAG function is the value of the argument the last time the function was invoked (see a more complete explanation in Chapter 19, Section H). In more complicated DATA steps, this can be very tricky. If we invoke the LAG function for each observation, then the value of the LAG function will be the value of the argument in the previous observation. The value LAGn (X) where n is a number, is the value from the nth previous observation. A common application of the LAG function is to take differences between observations. Suppose each subject was measured twice, and each measurement was entered as a separate observation. We want to take the value of X at time 1 and subtract it from the value of X at time 2. We proceed as follows:

Data set ORIG looks like this:

SUBJ	TIME	X
1	1	4
1	2	6
2	1	7
2	2	2
etc.		

To subtract the X at time 1 from the X at time 2, we write:

```
DATA LAGEG;
   SET ORIG;
   ***Note: Data Set ORIG is Sorted by SUBJ and TIME;
   DIFF = X-LAG(X);
   IF TIME=2 THEN OUTPUT;
RUN;
```

You could shorten this program even further by using the DIFn function, which returns the difference between a value from the current observation and the nth previous observation. The calculation above would be:

```
DIFF = DIF(X);
```

Chapter 19, Section H, shows how to use the LAG function to compute moving averages.

Problems

17-1. You have a SAS data set called HOSP, which contains a patient ID, gender, date of Birth (DOB), date of service (DOS), length of stay (LOS), systolic blood pressure (SBP), diastolic blood pressure (DBP), and heart rate (HR). Run the program below to create this data set:

```
DATA HOSP;
  INFORMAT ID $3. GENDER $1. DOB DOS MMDDYY8.;
  INPUT ID GENDER DOB DOS LOS SBP DBP HP;
  FORMAT DOB DOS MMDDYY10.;
```

[Continued]

```
DATALINES;
1 M 10/21/46 3/17/97 3 130 90 68
2 F 11/1/55 3/1/97 5 120 70 72
3 M 6/6/90 1/1/97 100 102 64 88
4 F 12/21/20 2/12/97 10 180 110 86
;
```

Create a new data set (NEW_HOSP) that contains all of the variables in HOSP, plus the following:

(a) The base 10 log of LOS (call it LOG_LOS).

(b) The patient's age as of his/her last birthday, on the date of service (call it AGE_LAST).

(c) A new variable (X) computed as the square root of the mean of the systolic and diastolic blood pressures, rounded to the nearest tenth.

17-2. A data set (MANY) contains the variables X1-X5, Y1-Y5. First, run the program below to create this data set:

```
DATA MANY;
   INPUT X1-X5 Y1-Y5;
DATALINES;
1 2 3 4 5   6 7 8 9 10
3 . 5 . 7   5 . . . 15
9 8 . . .   4 4 4 4 1
;
```

Write a program to include the following in data set MANY:

(a) The mean (average) of the X's (call it MEAN_X) and the mean of the Y's (call it MEAN_Y).

(b) The minimum value of the X's (call it MIN_X) and the minimum value of the Y's (call it MIN_Y).

(c) A new variable (CRAZY) which is the maximum of the X's times the minimum of the Y's times the sum of (the number of nonmissing X's plus the number of missing Y's). In other words:

```
CRAZY = (Maximum of X'S) × (Minimum of Y's) ×
        (Number of nonmissing X's + Number of missing
        Y's).
```

(d) Compute a variable MEAN_X_Y that is the mean of all the X's and Y's (the mean of all 10 numbers) with the provision that there be three or more nonmissing X values and four or more nonmissing Y values.

***17-3.** Create a SAS data set (UNI) that contains 1000 random numbers in the range of 1 to 5. Use the INT function to give you only integers, and be sure that there is an equal likelihood of choosing each of the five integers. Be careful that the values of 1 and 5 have the same probability of being chosen as 2 and 4. Run PROC FREQ to count the number of 1's, 2's, and so forth, and compute chi-square by hand to test if the distribution differs from uniform. Note that this goodness-of-fit test has four degrees of freedom.

17-4. Use the data set HOSP from problem 17-1, and create two vertical bar charts: one for the day of the week (formatted please) and one for the month of the year (no need to format) of the date of service.

17-5. Run the program below to create a SAS data set called MIXED:

```
DATA MIXED;
  INPUT X Y A $ B $;
DATALINES;
1 2 3 4
5 6 7 8
;
```

Create a new data set (NUMS) containing all four variables (you can use a new name for A and B) with only numeric variables.

***17-6.** Using the data set NEW_HOSP created in problem 17-1, create a new character variable called AGEGROUP, with the following groupings:

```
1 = Ages less than 20 (but not missing)
2 = 20 to 40
3 = 41 to highest
```

Do this with a user-defined format and a PUT function.

CHAPTER

18

A Review of SAS Functions:

Part II. Character Functions

A. *Introduction*

SAS software is rich in its assortment of functions that deal with character data. This class of functions is sometimes called STRING functions. In this chapter, we demonstrate some of the more useful string functions.

Some of the functions we discuss are: VERIFY, TRANSLATE, TRANWRD, COMPRESS, COMPBL, LENGTH, SUBSTR, INPUT, PUT, SCAN, TRIM, UPCASE, LOWCASE, REPEAT, || (concatenation), INDEX, INDEXC, AND SOUNDEX. Did you realize there were so many string functions? Let's get started.

B. *How Lengths of Character Variables Are Set in a SAS DATA Step*

Before we actually discuss these functions, we need to understand how SAS software assigns storage lengths to character variables and what the LENGTH function does for us. Look at the following program:


```
DATA EXAMPLE1;
   INPUT GROUP $ @10 STRING $3.;
   LEFT  = 'X    '; *X AND 4 BLANKS;
   RIGHT = '    X'; *4 BLANKS AND X;
   C1 = SUBSTR(GROUP,1,2) ;
   C2 = REPEAT(GROUP,1);
   LGROUP = LENGTH(GROUP) ;
   LSTRING = LENGTH(STRING) ;
   LLEFT = LENGTH(LEFT);
   LRIGHT = LENGTH(RIGHT);
   LC1 = LENGTH(C1);
   LC2 = LENGTH(C2);
DATALINES;
ABCDEFGH 123
XXX       4
Y        5
;
PROC CONTENTS DATA=EXAMPLE1 POSITION;
   TITLE 'Output from PROC CONTENTS';
RUN;

PROC PRINT DATA=EXAMPLE1 NOOBS;
   TITLE 'Listing of Example 1';
RUN;
```

One purpose of this example is to clarify the term LENGTH. If you look at the output from PROC CONTENTS, each of the variables is listed, along with a TYPE and LENGTH. Take a moment and look at the output from PROC CONTENTS below:

```
CONTENTS PROCEDURE

-----Variables Ordered by Position-----

 #    Variable    Type    Len    Pos
-------------------------------------
 1    GROUP       Char      8      0
 2    STRING      Char      3      8
 3    LEFT        Char      5     11
 4    RIGHT       Char      5     16
 5    C1          Char      8     21
 6    C2          Char    200     29
 7    LGROUP      Num       8    229
 8    LSTRING     Num       8    237
 9    LLEFT       Num       8    245
10    LRIGHT      Num       8    253
11    LC1         Num       8    261
12    LC2         Num       8    269
```

The column labeled LEN (length) is the number of bytes needed to store the values for each of the variables listed. By default, all the numeric variables are stored in eight bytes.

But, what about the storage lengths of the character variables? Look first at the two variables listed in the INPUT statement; GROUP and STRING. Since this is the first mention of these variables in this DATA step, their lengths are assigned by the rules governing INPUT statements. Since no columns or informats were associated with the variable GROUP, its length is set to eight bytes, the default length for character variables in this situation. The variable STRING uses a $3. INFORMAT so its length will be set to three bytes. The length of LEFT and RIGHT are determined by the assignment statement. The storage lengths of C1 and C2 are more difficult to understand.

The variable C1 is defined to be a substring of the variable GROUP. The SUBSTR function takes the form:

```
SUBSTR(char_var,start,length);
```

This function says to take a substring from char_var, starting at the position indicated by the start argument, for a length indicated by the length argument. Why then, is the length of C1 equal to 8 and not 2? The SAS compiler determines lengths at compile time. Since the starting position and length arguments of the SUBSTR function can be variable expressions, the compiler must set the length of C1 equal to the largest possible value it can ever attain, the length of GROUP.

The same kind of logic controls the length of C2, defined by the REPEAT function. Since the number of addition replications is defined by the second argument of the REPEAT function, and this argument can be a variable expression, the compiler sets the length of C2 to the largest possible value, 200. Why 200? Because that is the maximum length of a character variable in the SAS system.

There is a lesson here: Always use a LENGTH statement for any character variables that do not derive their length elsewhere. For example, to set the length of C2 to 16, you would write:

```
LENGTH C2 $ 16;
```

The LENGTH function does not, as you might guess, return the storage length of a character variable. Instead, it returns the position of the right-most nonblank character. Thus, trailing blanks are excluded in its computation.

The value of LLEFT and LRIGHT are 1 and 5, respectively, for every observation. This demonstrates that the trailing blanks in LEFT are not counted by the LENGTH function, while the leading blanks in RIGHT are. The table below summarizes the lengths for the remaining variables:

Obs	GROUP	LGROUP	STRING	LSTRING	C1	LC1	C2	LC2
1	abcdefgh	8	123	3	ab	2	abcdefghabcdefgh	16
2	xxx	3	4	1	xx	2	xxx xxx	11
3	y	1	5	1	y	1	y y	9

The values of LGROUP and LSTRING are straightforward. The value of LC1 is 1 for the third observation since C1 is only 1 byte in length in the third observation. The values for LC2 are more complicated. The REPEAT function says to take the original value and repeat it n times. So, for the first observation, LC2 is 16 (2 times 8). For observations 2 and 3, the trailing blanks come into play. In observation 2, the value of GROUP is 'XXXbbbbb' (where the b's stand for blanks). When we repeat this string one additional time, we get: 'XXXbbbbbXXXbbbbb'. Not counting the trailing blanks, we have a length of 8 + 3 = 11. Using the same logic for the third observation, we have a 'Y' followed by seven blanks, repeated once. Not counting the last seven trailing blanks, we have a length of 8 + 1 = 9.

With these preliminaries out of the way, we can now begin our tour of some of the very useful string functions available in SAS software.

C. *Working with Blanks*

This example demonstrates how to convert multiple blanks to a single blank. Suppose you have some names and addresses in a file. Some of the data-entry clerks placed extra spaces between the first and last names and in the address fields. You prefer to store all names and addresses with single blanks. Here is an example of how this conversion is accomplished:

```
DATA EXAMPLE2;
   INPUT #1 @1  NAME      $20.
         #2 @1  ADDRESS   $30.
         #3 @1  CITY      $15.
            @20 STATE     $2.
            @25 ZIP       $5.;
   NAME = COMPBL(NAME);
   ADDRESS = COMPBL(ADDRESS);
   CITY = COMPBL(CITY);
DATALINES;
RON CODY
89 LAZY BROOK ROAD
FLEMINGTON        NJ   08822
BILL      BROWN
28   CATHY   STREET
NORTH   CITY      NY   11518
;
PROC PRINT DATA=EXAMPLE2;
   TITLE 'Example 2';
   ID NAME;
   VAR ADDRESS CITY STATE ZIP;
RUN;
```

Thus, a seemingly difficult task is accomplished in a single line by using the COMPBL (COMPress BLank) function, compressing multiple blanks to a single blank. How useful!

D. How to Remove Characters from a String

A more general problem is the removal (deletion) of selected characters from a string. For example, suppose you want to remove blanks, parentheses, and dashes from a phone number that has been stored as a character value. Here comes the COMPRESS function to the rescue! The COMPRESS function can remove any number of specified characters from a character variable. The program below uses the COMPRESS function twice. The first time, to remove blanks from the string; the second, to remove blanks plus the other characters mentioned above. Here is the code:

```
DATA EXAMPLE3;
   INPUT PHONE $ 1-15;
   PHONE1 = COMPRESS(PHONE);
   PHONE2 = COMPRESS(PHONE,'(-) ');
DATALINES;
(908)235-4490
(201) 555-77 99
;
PROC PRINT DATA=EXAMPLE3;
   TITLE 'Listing of Example 3';
RUN;
```

The variable PHONE1 has just blanks removed. Notice that the COMPRESS function does not have a second argument here. When it is omitted, the COMPRESS function removes only blanks. For the variable PHONE2, the second argument of the COMPRESS function contains a list of the characters to remove: left parenthesis, dash, right parenthesis, and blank. This string is placed within single or double quotes.

E. Character Data Verification

A common task in data processing is to validate data. For example, you may want to be sure that only certain values are present in a character variable. In the example below, only the values 'A', 'B', 'C', 'D', and 'E' are valid data values. A very easy way to test if there are any invalid characters present is as follows:

```
DATA EXAMPLE4;
   INPUT ID $ 1-4 ANSWER $ 5-9;
   P = VERIFY(ANSWER,'ABCDE');
   OK = P EQ 0;
DATALINES;
001 ACBED
002 ABXDE
003 12CCE
004 ABC E
;
PROC PRINT DATA=EXAMPLE4 NOOBS;
   TITLE 'Listing of Example 4';
RUN;
```


The "workhorse" of this example is the VERIFY function, which is a bit complicated. It inspects every character in the first argument and, if it finds any value not in the verify string (the second argument), it will return the position of the first offending value. If all the values of the string are located in the verify string, a value of 0 is returned. In the first observation, P will be 0 and OK will be 1; in the second observation, P will be a 3 (the position of the 'X') and OK will be 0; in the third observation, P will be 1 and OK will be 0; finally, in the fourth observation, P will be a 4 and OK will be 0.

Another way to solve the same problem is the following: Suppose someone gave you the data set EXAMPLE5, created by running the short DATA step below:

```
DATA EXAMPLE5;
   INPUT STRING $3.;
DATALINES;
ABC
[illegible]
[illegible]
[illegible]
;
```

To list any observation with values for STRING that are not the letters A-E or blank, the following DATA step could be used:

```
[illegible]
   [illegible]
      [illegible]
      [illegible]
   IF VERIFY(STRING,CHECK) NE 0 THEN
      PUT 'Error in Record ' _N_ STRING=;
RUN;
```

F. Substring Example

We mentioned in the Introduction that a substring is a part of a longer string (although it can actually be the same length but this would not be too useful). In this example, you have ID codes that contain a state abbreviation in the first two positions. Furthermore, in positions 7-9 is a numeric code. You want to create two new variables; one containing the two-digit state codes, and the other, a numeric variable constructed from the three numerals in positions 7, 8, and 9. Here goes:

```
DATA EXAMPLE6;
   INPUT ID $ 1-9;
   LENGTH STATE $ 2;
   STATE = SUBSTR(ID,1,2);
   NUM = INPUT(SUBSTR(ID,7,3),3.);
```

[Continued]

```
DATALINES;
NYXXXX123
NJ1234567
;
PROC PRINT DATA=EXAMPLE6 NOOBS;
   TITLE 'LISTING OF EXAMPLE 6';
RUN;
```

Creating the state code is easy. We use the SUBSTR function. The first argument is the variable from which we want to extract the substring; the second argument is the starting position of the substring; and the last argument is the length of the substring (not the ending position as you might guess). Also note the use of the LENGTH statement to set the length of STATE to 2 bytes.

Extracting the three-digit number code is more complicated. First, we use the SUBSTR function to extract the three numerals (numerals are character representations of numbers). However, the result of a SUBSTR function is always a character value. To convert the character value to a number, we use the INPUT function. The INPUT function takes the first argument and "reads" it as if it were coming from a file, according to the INFORMAT listed as the second argument. So, for the first observation, the SUBSTR function would return the string '123', and the INPUT function would convert this to the number 123. As a point of interest, you may use a longer INFORMAT as the second argument without any problems. For example, the INPUT function could have been written as:

```
INPUT (SUBSTR(ID,7,3),8.);
```

and everything would have worked out fine. This fact is useful in situations where you do not know the length of the string ahead of time.

G. Using the SUBSTR Function on the Left-hand Side of the Equals Sign

There is a particularly useful and somewhat obscure use of the SUBSTR function that we would like to discuss next. You can use this function to place characters in specific locations within a string by placing the SUBSTR function on the left-hand side of the equals sign (in the older SAS manuals we believe this was called a SUBSTR pseudo variable).

Suppose you have some systolic blood pressures (SBP) and diastolic blood pressures (DBP) in a SAS data set. You want to print out these values and star high values with an asterisk. Here is a program that uses the SUBSTR function on the left of the equals sign to do that:


```
DATA EXAMPLE7;
   INPUT SBP DBP @@;
   LENGTH SBP_CHK DBP_CHK $ 4;
   SBP_CHK = PUT(SBP,3.);
   DBP_CHK = PUT(DBP,3.);
   IF SBP GT 160 THEN SUBSTR(SBP_CHK,4,1) = '*';
   IF DBP GT 90  THEN SUBSTR(DBP_CHK,4,1) = '*';
DATALINES;
120 80 180 92 200 110
;
PROC PRINT DATA=EXAMPLE7 NOOBS;
   TITLE 'Listing of Example 7';
RUN;
```

We first need to set the lengths of SBP_CHK and DBP_CHK to four (three spaces for the value plus one for the possible asterisk). Next, we use a PUT function to perform a numeric to character conversion. The PUT function is, in some ways, similar to the INPUT function. It "writes out" the value of the first argument, according to the FORMAT specified in the second argument. By "write out" we actually mean assign the value to the variable on the left of the equals sign. The SUBSTR function then places an asterisk in the fourth position when a value of SBP is greater than 160 or a value of DBP is greater than 90.

H. Doing the Previous Example Another Way

It is both interesting and instructive to obtain the results above without using the SUBSTR function on the left-hand side of the equals sign. We are not doing this just to show you a hard way to accomplish something we already did. Rather, this alternative solution uses a number of character functions that can be demonstrated. Here is the program:

```
DATA EXAMPLE8;
   INPUT SBP DBP @@;
   LENGTH SBP_CHK DBP_CHK $ 4;
   SBP_CHK = PUT(SBP,3.);
   DBP_CHK = PUT(DBP,3.);
   IF SBP GT 160 THEN SBP_CHK = SUBSTR(SBP_CHK,1,3) || '*';
   IF DBP GT  90 THEN DBP_CHK = TRIM(DBP_CHK) || '*';
DATALINES;
120 80 180 92 200 110
;
PROC PRINT DATA=EXAMPLE8 NOOBS;
   TITLE 'Listing of Example 8';
RUN;
```

This program is not really more complicated but maybe just not as elegant as the first program. This program uses the concatenation operator (| |) to join the three-character blood-pressure value with an asterisk. Since SBP_CHK and DBP_CHK were both assigned a length of 4, we wanted to be sure to concatenate, at most, the first 3 bytes with the asterisk. Just for didactic purposes, we did this two ways. For the SBP_CHK variable, we used a SUBSTR function to extract the first 3 bytes. For the DBP_CHK variable, the TRIM function was used. The TRIM function removes trailing blanks from a character string.

I. Unpacking a String

To save disk storage, you may wish to store several single-digit numbers in a longer character string. For example, storing five numbers as numeric variables with the default 8 bytes each would take up 40 bytes of disk storage per observation. Even reducing this to 3 bytes each would result in 15 bytes of storage. If, instead, you store the five digits as a single character value, you need only 5 bytes.

That is fine, but at some point, you may need to get the numbers back out for computation purposes. Here is a nice way to do this:

```
DATA EXAMPLE9;
   INPUT STRING $ 1-5;
DATALINES;
12345
8 642
;
DATA UNPACK;
   SET EXAMPLE9;
   ARRAY X[5];
   DO J = 1 TO 5;
      X[J] = INPUT(SUBSTR(STRING,J,1),1.);
   END;
   DROP J;
RUN;

PROC PRINT DATA=UNPACK NOOBS;
   TITLE 'Listing of UNPACK';
RUN;
```

We first created an array to hold the five numbers, X1 to X5. Don't be alarmed if you don't see any variables listed on the ARRAY statement. ARRAY X[5]; is equivalent to ARRAY X[5] X1-X5; We use a DO loop to cycle through each of the five starting positions corresponding to the five numbers we want. As mentioned before, since the result of the SUBSTR function is a character value, we need to use the INPUT function to perform the character to numeric conversion.

J. Parsing a String

"Parsing" a string means to take it apart based on some rules. In the example to follow, five separate character values are placed together on a line, with either a space, a comma, a semicolon, a period, or an explanation mark between them. You would like to extract the five values and assign them to five character variables. With the SCAN function this difficult task is simplified:

```
DATA EX_10;
   INPUT LONG_STR $ 1-80;
   ARRAY PIECES[5] $ 10
      PIECE1-PIECE5;
   DO I = 1 TO 5;
      PIECES[I] = SCAN(LONG_STR,I,',;.! ');
   END;
   DROP LONG_STR I;
DATALINES4;
THIS LINE,CONTAINS!FIVE.WORDS
ABCDEFGHIJKL XXX;YYY
;;;;
PROC PRINT DATA=EX_10 NOOBS;
   TITLE 'Listing of Example 10';
RUN;
```

Before we discuss the SCAN function, a brief word about DATALINES4 and the four semicolons ending our data. If you have data values that include semicolons, you cannot use a simple DATALINES (or CARDS) statement since the semicolon would signal the end of your data. Instead, the statement DATALINES4 (or CARDS4) is used, causing the program to continue reading data values until four semicolons are read.

The function:

```
SCAN(char_var,n,'list-of-delimiters');
```

returns the nth "word" from the char_var, where a "word" is defined as anything between two delimiters. If there are fewer than n words in the character variable, the SCAN function will return a blank.

By placing the SCAN function in a DO loop, we can pick out the nth word in the string.

K. Locating the Position of One String within Another String

Two somewhat similar functions, INDEX and INDEXC, can be used to locate a string or one of several strings within a larger string. For example, if you have a string 'ABCDEFG' and want the location of the letters DEF (starting position 4), the following INDEX function could be used:

```
INDEX('ABCDEFG','DEF');
```

This would return a value of 4. If you want to know the starting position of any one of several strings, the INDEXC function can be used. As an example, if you wanted the starting position of either 'BC', or 'FG' in the string 'ABCDEFG', you would code:

```
INDEXC('ABCDEFG','BC','FG');
```

The function would return a value of 2, the starting position of 'BC'. Here is a short program which demonstrates these two functions:

```
DATA EX_11;
   INPUT STRING $ 1-10;
   FIRST = INDEX(STRING,'XYZ');
   FIRST_C = INDEXC(STRING,'X','Y','Z');
DATALINES;
ABCXYZ1234
1234567890
ABCX1Y2Z39
ABCZZZXYZ3
;
PROC PRINT DATA=EX_11 NOOBS;
   TITLE 'Listing of Example 11';
RUN;
```

FIRST and FIRST_C for each of the 4 observations are:

OBS	FIRST	FIRST_C
1	4	4
2	0	0
3	0	4
4	7	4

When the search fails, both functions return a zero.

L. *Changing Lower Case to Upper Case and Vice Versa*

The two companion functions UPCASE and LOWCASE do just what you would expect. These two functions are especially useful when data-entry clerks are careless and a mixture of upper- and lower-case values are entered for the same variable. You may want to place all of your character variables in an array and UPCASE (or LOWCASE) them all. Here is an example of such a program:

```
DATA EX_12;
   LENGTH A B C D E $ 1;
   INPUT A B C D E X Y;
DATALINES;
M f P p D 1 2
m f m F M 3 4
;
```


[Continued]

```
DATA UPPER;
   SET EX_12;
   ARRAY ALL_C[*] _CHARACTER_;
   DO I = 1 TO DIM(ALL_C);
      ALL_C[I] = UPCASE(ALL_C[I]);
   END;
   DROP I;
RUN;

PROC PRINT DATA=UPPER NOOBS;
   TITLE 'Listing of UPPER';
RUN;
```

This program uses the _CHARACTER_ keyword to select all the character variables. The result of running this program is to convert all values for the variables A, B, C, D, and E to upper case. The LOWCASE function could be used in place of the UPCASE function if you wanted all your character values in lower case.

M. *Substituting One Character for Another*

A very handy character function is TRANSLATE. It can be used to convert one character to another in a string. For example, suppose you recorded multiple choices on a test as 1, 2, 3, 4, or 5, which represented the letters 'A' through 'E', respectively. When you print out the character values, you want to see the letters rather than the numerals. While formats would accomplish this very nicely, it also serves as an example for the TRANSLATE function. Here is the code:

```
DATA EX_13;
   INPUT QUES : $1. @@;
   QUES = TRANSLATE(QUES,'ABCDE','12345');
DATALINES;
1 4 3 2 5
5 3 4 2 1
;
PROC PRINT DATA=EX_13 NOOBS;
   TITLE 'LISTING OF EXAMPLE 13';
RUN;
```

The syntax for the TRANSLATE function is:

```
TRANSLATE(char_var,to_string,from_string);
```

Each value in from_string is translated to the corresponding value in the to_string.

Another interesting application of the TRANSLATE function is the creation of dichotomous numeric variables from character variables. For example, you may wish to set values of 'N' to 0 and values of 'Y' to 1. Although this is easily done with IF-THEN/ELSE statements, let's see if we can do it using the TRANSLATE function. Here goes:

```
DATA EX_14;
   LENGTH CHAR $ 1;
   INPUT CHAR @@;
   X = INPUT(TRANSLATE(UPCASE(CHAR),'01','NY'),1.);
DATALINES;
N Y n y A B 0 1
;
PROC PRINT DATA=EX_14 NOOBS;
   TITLE 'Listing of Example 14';
RUN;
```

The UPCASE function sets all values to upper case. Next, the TRANSLATE function converts values of 'N' to '0' and 'Y' to '1'. Finally, the INPUT function converts the numerals '0' and '1' to the numbers 0 and 1, respectively.

N. Substituting One Word for Another in a String

A relatively new function (as of version 6.07 on personal computers), TRANWRD (translate word), can perform a search and replace operation on a string variable. For example, you may want to standardize addresses by converting the words 'Street', 'Avenue', and 'Road' to the abbreviations 'St.', 'Ave.', and 'Rd.', respectively. Look at the following program:

```
DATA CONVERT;
   INPUT @1 ADDRESS $20. ;
   *** Convert Street, Avenue, and Boulevard to
       their abbreviations;
   ADDRESS = TRANWRD (ADDRESS,'Street','St.');
   ADDRESS = TRANWRD (ADDRESS,'Avenue','Ave.');
   ADDRESS = TRANWRD (ADDRESS,'Road','Rd.');
DATALINES;
89 Lazy Brook Road
123 River Rd.
12 Main Street
;
PROC PRINT DATA=CONVERT;
   TITLE 'Listing of Data Set CONVERT';
RUN;
```

The syntax of the TRANWRD function is:

```
TRANWRD (char_var,'find_string','replace_string');
```


That is, the function will replace every occurrence of find_string with replace_string. Notice that the order of the find and replace strings are reversed compared to the TRANSLATE function where the to_string comes before the from_string as arguments to the function. In this example, 'Street' will be converted to 'St.', 'Avenue' to 'Ave.', and 'Road' to 'Rd.'. The listing below confirms this fact:

Listing of Data Set CONVERT

OBS	ADDRESS
1	89 Lazy Brook Rd.
2	123 River Rd.
3	12 Main St.

O. Concatenating (Joining) Strings

Since we are discussing strings, we should mention the concatenation operation. Although this is not a function, it is a useful string operation and this seems as good as anywhere to tell you about it! In computer jargon, "concatenate" means to join. So, if we concatenate the string 'ABC' with the string 'XYZ', the result is 'ABCXYZ'. Pretty clever, eh? Things can get "a bit sticky" if we forget that SAS character variables may be padded on the right with blanks to fill out the predefined length of the variable. If this is the case, we can use the TRIM function to remove trailing blanks before we concatenate the strings. The concatenation operator is | | (two vertical bars). Suppose we had social security numbers in a file and, instead of the usual—separators, the digit groups were separated by colons, 123:45:6789, for example. One way to read this string and convert it into the more common format would be:

```
DATA EX_15;
   INPUT PART1 $ 1-3 PART2 $ 5-6 PART3 $ 8-11;
   SS = PART1 || '-' || PART2 || '-' || PART3;
   KEEP SS;
DATALINES;
123:45:6789
;
PROC PRINT DATA=EX_15;
   TITLE 'Listing of Data Set EX_15';
RUN;
```

The output (just one observation) is:

```
Listing of EX_15

OBS         SS

 1      123-45-6789
```

The compulsive programmer in one of us (Cody), will not let this program stand without mentioning that the TRANSLATE function discussed previously would be a better way to solve this problem. (Your humble second author, Smith, cannot believe anyone is actually reading this section.) The solution to the social security problem above is:

```
INPUT SS $ 1-11;
SS = TRANSLATE (SS,'-',':');
```

P. Soundex Conversion

SAS software provides a soundex function which returns the soundex equivalent of a name. Soundex equivalents of names allow you to match two names from two different sources even though they might be spelled differently. Great care is needed when using this function since many very dissimilar looking names may translate to the same soundex equivalent. The soundex equivalent of most names will result in strange looking codes such as C3 or A352. Here is a sample program and the results of the soundex translations:

```
DATA EX_16;
   LENGTH NAME1-NAME3 $ 10;
   INPUT NAME1-NAME3;
   S1 = SOUNDEX(NAME1);
   S2 = SOUNDEX(NAME2);
   S3 = SOUNDEX(NAME3);
DATALINES;
cody Kody cadi
cline klein clana
smith smythE adams
;
PROC PRINT DATA=EX_16 NOOBS;
   TITLE 'Listing of Example 16';
RUN;
```

This program will result in the following soundex matches:

Name	Soundex Equivalent
CODY	C3
KODY	K3
CADI	C3
CLINE	C45
KLEIN	K45
CLANA	C45
SMITH	S53
SMYTHE	S53
ADAMS	A352

Problems

18-1. First, create a data set called CHAR1 by running the DATA step below:

```
DATA CHAR1;
   INPUT STRING1 $1.
         STRING2 $5.
         STRING3 $8.
         (C1-C5)($1.);
DATALINES;
XABCDE12345678YNYNY
YBBBBB12V56876yn YY
ZCCKCC123-/. ,WYNYN
;
```

Create a new data set ERROR containing any observations in CHAR1 that meet any one of the following conditions:

1. A value other than an 'X' or a 'Y' for the variable STRING1.
2. A value other than an 'A', 'B', 'C', 'D', or 'E' in STRING2.
3. A value other than an upper or lower case 'N' or 'Y' for the variables C1–C5. (NOTE: Blank values of C1–C5 will place the observation in the ERROR data set.)

18-2. Using the data set CHAR1 from the previous problem, create a new variable (NEW3) from the variable STRING3, based on the following rules: First, remove embedded blanks and the characters '-', '/', '.', and ','. Next, substitute the letters A-H for the numerals 1–8. Finally, set NEW3 equal to a missing value if there are any characters other than A–H (trailing blanks are OK) in the string. The value of NEW3 for the three observations should be: 'ABCDEFGH', missing, and 'ABC'.

18-3. For the variables C1–C5 in data set CHAR1 (problem 18-1), change all lower-case values to upper case, and set any remaining values other than 'Y', 'N', or blank to missing. You may wish to use an ARRAY to solve this problem.

18-4. You are given a data set (PHONE) with phone numbers stored in a variety of formats. Run the program below to create the PHONE data set and create a new variable, NUMBER, that is, a numerical variable derived from the character variable CHAR_NUM, with all extraneous symbols and blanks removed. Assume that the longest phone number contains 10 digits.

```
DATA PHONE;
   INPUT CHAR_NUM $20.;
DATALINES;
(908)235-4490
(800) 555 - 1 2 1 2
203/222-4444
;
```

18-5. The data set (EXPER) created by running the program below, contains the variables ID, GROUP, and DOSE. Create two new variables as follows: One is a 2-byte character variable created from the second and third bytes of ID. The other is a variable created by concatenating GROUP and DOSE. This variable should be 6 bytes long with a '-' between the GROUP and DOSE values, and it should not contain any embedded blanks. Call the two variables SUB_ID and GRP_DOSE, respectively:

```
DATA EXPER;
   INPUT ID    $ 1-5
         GROUP $ 7
         DOSE  $ 9-12;
DATALINES;
1NY23 A HIGH
3NJ99 B HIGH
2NY89 A LOW
5NJ23 B LOW
;
```

*18-6. Using the data set EXPER from the previous problem, create a new variable (ID2) from ID. Make this new variable 6 bytes in length and place an asterisk in the sixth byte if the fourth byte of ID has a value of 5 or more. For this solution, create a numeric variable based on the fourth byte of ID, and check if it is greater than or equal to 5. Do not check this byte as a character value against the numerals 5-9.

*18-7. Merge the two data sets created by running the program below, using the GENDER, date of birth (DOB), and the SOUNDEX equivalent of the NAME to determine matches. Keep only those observations where a match is successful. Call the new data set COMBINED. There should be three observations in data set COMBINED.

```
DATA ONE;
   INPUT @1  GENDER  $1.
         @2  DOB     MMDDYY8.
         @10 NAME    $11.
         @21 STATUS  $1.;
   FORMAT DOB MMDDYY8.;
DATALINES;
M10/21/46CADY       A
F11/11/50CLINE      B
M11/11/52SMITH      A
F10/10/80OPPENHEIMERB
M04/04/60JOSE       A
;
DATA TWO;
   INPUT @1  GENDER  $1.
         @2  DOB     MMDDYY8.
         @10 NAME    $11.
         @21 WEIGHT  3.;
```


[Coutinued]

```
   FORMAT DOB MMDDYY8.;
DATALINES;
M10/21/46CODY       160
F11/11/50CLEIN      102
F11/11/52SMITH      101
F10/10/80OPPENHAIMER120
M02/07/60JOSA       220
;
```

CHAPTER

19

Selected Programming Examples

A. *Introduction*

This chapter contains a number of common applications and serves two functions: One is to allow you to use any of the programs here, with modification, if you have a similar application; the other is to demonstrate SAS programming techniques.

B. *Expressing Data Values as a Percentage of the Grand Mean*

A common problem is to express data values as percentages of the grand mean, rather than in the original raw units. In this example, we record the heart rate (HR), systolic blood pressure (SBP), and diastolic blood pressure (DBP) for each subject. We want to express these values as percentages of the mean HR, SBP, and DBP for all subjects. For example, in the data set below, the mean heart rate for all subjects is 70 (mean of 80, 70, and 60). The first subject's score of 80, expressed as a percentage, would be 100% × 80/70 = 114.28%. The approach here will be to use PROC MEANS to compute the means and to output them in a data set which we then combine with the original data so that we can perform the required computation.


```
DATA TEST;
   INPUT HR SBP DBP;
DATALINES;
80 160 100
70 150 90
60 140 80
;
PROC MEANS NOPRINT DATA=TEST;
   VAR HR SBP DBP;
   OUTPUT OUT=MOUT(DROP=_TYPE_ _FREQ_)
          MEAN=MHR MSBP MDBP;
RUN;

DATA NEW (DROP=MHR MSBP MDBP);
   SET TEST;
   IF _N_ = 1 THEN SET MOUT;
   HRPER=100*HR/MHR;
   SBPPER=100*SBP/MSBP;
   DBPPER=100*DBP/MDBP;
RUN;

PROC PRINT DATA=NEW NOOBS;
   TITLE 'Listing of Data Set NEW';
RUN;
```

Description. We use the NOPRINT option with PROC MEANS because we do not want the procedure to print anything but, rather, to create a data set of means. In this case, the output data set from PROC MEANS will consist of one observation. The variables _TYPE_ and _FREQ_, which are normally added to the output data set, are dropped (using a DROP= data set option) since they are not needed. _TYPE_ is useful when a CLASS statement is used with PROC MEANS. We show examples of this later. The single observation in the data set MOUT is shown below:

```
OBS   MHR   MSBP   MDBP
 1    70    150    90
```

We want to add the three variables MHR, MSBP, and MDBP to every observation in the original data set so that we can divide each value by the mean and multiple by 100%. Since our original data set has three observations and the mean data set contains only one observation, we use a trick: We use the SAS internal variable _N_ to conditionally execute the SET statement. The Program Data Vector now contains the variables HR, SBP, DBP, MHR, MSBP, and MDBP. Any variables coming in from a SET statement are automatically retained so the values of MHR, MSBP, and MDBP will not change and will be available for every iteration of the DATA step. We can now divide HR by MHR, SBP by MSBP, and DBP by MDBP. Each time we bring in another observation from the original (TEST) data set, the values of MHR, MSBP, and MDBP will remain in the Program Data Vector (they are automatically RETAINED because of the SET statement). The final data set created by this program (NEW) is shown next:

HR	SBP	DBP	HRPER	SBPPER	DBPPER
80	160	100	114.286	106.667	111.111
70	150	90	100.000	100.000	100.000
60	140	80	85.714	93.333	88.889

C. Expressing a Value as a Percentage of a Group Mean

This example is an extension of the previous problem. Here we have two groups (A and B), and we want to express each measurement as a percentage of the GROUP mean.

```
DATA TEST;
   INPUT GROUP $ HR SBP DBP @@;
DATALINES;
A 80 160 100 A 70 150 90 A 60 140 80
B 90 200 180 B 80 180 140 B 70 140 80
;
PROC SORT DATA=TEST;
   BY GROUP;
RUN;

PROC MEANS DATA=TEST NOPRINT NWAY;
   CLASS GROUP;
   VAR HR SBP DBP;
   OUTPUT OUT=MOUT(DROP=_TYPE_ _FREQ_)
          MEAN=MHR MSBP MDBP;
RUN;

DATA NEW (DROP=MHR MSBP MDBP);
   MERGE TEST MOUT;
      BY GROUP;
   HRPER=100*HR/MHR;
   SBPPER=100*SBP/MSBP;
   DBPPER=100*DBP/MDBP;
RUN;

PROC PRINT NOOBS;
   TITLE 'Listing of Data Set NEW';
RUN;
```

Description. Since the output data set from PROC MEANS contains GROUP, we can MERGE it with the original data set, using GROUP as the BY variable. Data set MOUT will contain two observations, one for each value of GROUP. The contents of MOUT are shown below:

OBS	GROUP	MHR	MSBP	MDBP
1	A	70	150.000	90.000
2	B	80	173.333	133.333

Data set NEW, which contains both the original values and the percentage values is shown next:

GROUP	HR	SBP	DBP	HRPER	SBPPER	DBPPER
A	80	160	100	114.286	106.667	111.111
A	70	150	90	100.000	100.000	100.000
A	60	140	80	85.714	93.333	88.889
B	90	200	180	112.500	115.385	135.000
B	80	180	140	100.000	103.846	105.000
B	70	140	80	87.500	80.769	60.000

D. Plotting Means with Error Bars

When we plot a set of means, we may want to include error bars, which represent one standard error above and below the mean. The program below shows how we can use PROC MEANS to output the standard errors and then use PROC PLOT to plot the means and the error bars.

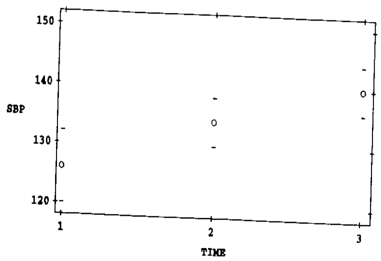
```
DATA ORIG;
   INPUT SUBJ TIME DBP SBP;
DATALINES;
[illegible]

[illegible] NOPRINT NWAY;
[illegible]
[illegible] STDERR=SE_SBP [illegible]
RUN;

[illegible]
RUN;
```

[Continued]

```
PROC PLOT DATA=TMP;
   PLOT SBP*TIME='o' SBPHI*TIME='-' SBPLO*TIME='-' / OVERLAY BOX;
   PLOT DBP*TIME='o' DBPHI*TIME='-' DBPLO*TIME='-' / OVERLAY BOX;
   TITLE 'Plot of Mean Blood Pressures at Each Time';
   TITLE2 'Error bars represent +- 1 standard error';
RUN;
```

Below is the first of the two plots (SBP by TIME) produced by this program:

```
Plot of Mean Blood Pressures at Each Time
Error bars represent +- 1 standard error

             Plot of SBP*TIME.     Symbol used is 'o'
             Plot of SBPHI*TIME.   Symbol used is '-'.
             Plot of SBPLO*TIME.   Symbol used is '-'.
```



Description. The original data set contained the variable TIME as well as the two blood pressures (SBP and DBP). The data set produced by PROC MEANS (with the CLASS statement) will have as many observations as there are values of TIME (note the NWAY option). We are seeking a plot of mean SBP (and DBP) versus time with a lower-case 'o' as the plotting symbol. The data set TMP adds and subtracts the standard errors from the means so that we can plot them along with the means. We are using a "-" as a plotting symbol to represent error bars.

E. Using a Macro Variable to Save Coding Time

Programmers are always looking for a way to make their programs more compact (and to avoid typing). While there is an extensive macro language as part of the SAS system, we will use only a macro variable in this example. Macro statements begin with % signs. A macro variable is defined with a %LET function. The expression to the right of the = sign will be assigned to the macro variable. We precede the macro variable name with an ampersand (&) in the program so that the system knows we

are referring to a macro variable. In this example, we are using a macro variable to take the place of a variable list. Any time we want to refer to the variables ONE, TWO, and THREE, we can use the macro variable LIST instead. (See Chapter 11, Section C for another example of a macro variable.)

```
DATA TEST;
   %LET LIST=ONE TWO THREE;
   INPUT &LIST FOUR;
DATALINES;
1 2 3 4
4 5 6 6
;
PROC FREQ DATA=TEST;
   TABLES &LIST;
RUN;
```

F. Computing Relative Frequencies

In this example, we have an ICD (International Classification of Diseases) code for each subject as well as the year that diagnosis was made. We want to know what percentage of the observations contain a particular code for each year. That is, we want to express the frequency of each ICD code as a percentage of all codes for that year. For example, in the data below, there were three recorded codes in 1950 (note they are not in YEAR order). Code 450 occurred twice, giving us a relative frequency of 2/3, or 66.6%. We will use an output data set from PROC FREQ to compute these relative frequencies, as follows:

```
DATA ICD;
   INPUT ID YEAR ICD;
DATALINES;
001 1950 450
002 1950 440
003 1951 460
004 1950 450
005 1951 300
;
PROC FREQ DATA=ICD;
   TABLES YEAR*ICD / OUT=ICDFREQ NOPRINT;
   ***Data set ICDFREQ contains the counts
   for each CODE in each YEAR;
RUN;

PROC FREQ DATA=ICD;
   TABLES YEAR / OUT=TOTAL NOPRINT;
   ***Data set ICD contains the total number
   of obs for each YEAR;
RUN;
```

[Coutinued]

```
DATA RELATIVE;
   MERGE ICDFREQ TOTAL (RENAME=COUNT=TOT_CNT));
   ***We need to rename COUNT in one of the two data sets
      so that we can have both values in data set RELATIVE;
   BY YEAR;
   RELATIVE=100*COUNT/TOT_CNT;
   DROP PERCENT;
RUN;

PROC PRINT DATA=RELATIVE;
   TITLE 'Relative Frequencies of ICD Codes by Year';
RUN;
```

Description. The first PROC FREQ creates an output data set (ICDFREQ) that looks like the following:

ICDFREQ data set

YEAR	ICD	COUNT	PERCENT
1950	440	1	20
1950	450	2	40
1951	300	1	20
1951	460	1	20

Notice that the data set created by PROC FREQ contains all the TABLES variables as well as the two variables COUNT and PERCENT. The COUNT variable in this data set tells us how many times a given ICD code appeared in each year.

Next, we need the total number of ICD codes for each year to be used in the denominator to create a relative incidence of each ICD code for each year. Running PROC FREQ with only YEAR as a TABLE variable will give us the number of ICD codes there were for each year. Data set TOTAL is shown next:

YEAR	COUNT	PERCENT
1950	3	60
1951	2	40

All we have to do now is to merge the two data sets so that we can divide the COUNT variable in the ICDFREQ data set by the COUNT variable in the TOTAL data set. When we do the merging, we will rename COUNT in the TOTAL data set to TOT_CNT since we can't have two values for a single variable in one observation. Finally, we can divide COUNT by TOT_CNT to obtain our desired result.

Relative Frequencies of ICD Codes by Year

YEAR	ICD	COUNT	TOTAL	RELATIVE
1950	440	1	3	33.3333
1950	450	2	3	66.6667
1951	300	1	2	50.0000
1951	460	1	2	50.0000

G. *Computing Combined Frequencies on Different Variables*

In this example questionnaires are issued to people to determine to which chemicals they are sensitive. Each subject replies yes or no (1 or 0) to each of the ten chemicals in the list. We want to list the chemicals in decreasing order of sensitivity. If we compute frequencies for each of the 10 variables, we will be unable to display a list showing the chemicals in decreasing order of frequency. Our first step is to restructure the data set into one with up to 10 observations per subject. Each observation will include a chemical number (from 1 to 10) indicating which chemical was selected. Here is the program:

```
PROC FORMAT;
   VALUE SYMPTOM 1='ALCOHOL' 2='INK' 3='SULPHUR' 4='IRON' 5='TIN'
                 6='COPPER'  7='DDT' 8='CARBON' 9='SO2' 10='NO2';
RUN;

DATA SENSI;
   INPUT ID 1-4 (CHEM1-CHEM10)(1.);
   ARRAY CHEM[*]  CHEM1-CHEM10;
   DO I=1 TO 10;
      IF CHEM[I]=1 THEN DO;
         SYMPTOM=I;
         OUTPUT;
      END;
   END;
   KEEP ID SYMPTOM;
   FORMAT SYMPTOM SYMPTOM.;
DATALINES;
00011010101010
00021000010000
00031100000000
00041001001111
00051000010010
;
PROC FREQ DATA=SENSI ORDER=FREQ;
   TABLES SYMPTOM;
RUN;
```

Description. For a more detailed description of how to restructure data sets using arrays, see the relevant section of Chapter 16. Data set SENSI will have as many observations per person as the number of 1's on the list of chemicals (CHEM1 to CHEM10). The variable SYMPTOM is set equal to the DO loop counter, I, which tells us which of the ten chemicals was selected. The observations from data set SENSI are shown here to help clarify this point:

ID	Symptom
1	ALCOHOL
1	SULPHUR
1	TIN
1	DDT
1	SO2
2	ALCOHOL
2	COPPER
3	ALCOHOL
3	INK
4	ALCOHOL
4	IRON
4	DDT
4	CARBON
4	SO2
4	NO2
5	ALCOHOL
5	COPPER
5	SO2

Notice that the formatted values for SYMPTOM are displayed since we assigned a format to the variable. A simple PROC FREQ will now tell us the frequencies for each of the 10 chemicals. The ORDER=FREQ option of PROC FREQ will produce a frequency list in decreasing order of frequencies. The output from PROC FREQ is as follows:

SYMPTOM	Frequency	Percent	Cumulative Frequency	Cumulative Percent
ALCOHOL	5	27.8	5	27.8
SO2	3	16.7	8	44.4
COPPER	2	11.1	10	55.6
DDT	2	11.1	12	66.7
INK	1	5.6	13	72.2
SULPHUR	1	5.6	14	77.8
IRON	1	5.6	15	83.3
TIN	1	5.6	16	88.9
CARBON	1	5.6	17	94.4
NO2	1	5.6	18	100.0

H. *Computing a Moving Average*

Suppose we record the COST of an item each day. For this example, we want to compute a moving average of the variable COST for a three-day interval. On day 3, we will average the COST for days 1, 2, and 3; on day 4, we will average the COST on days 2, 3, and 4, etc.

```
*--------------------------------------*
| Program to compute a moving average  |
*--------------------------------------*;
DATA MOVING;
   INPUT COST @@;
   DAY+1;
   COST1=LAG(COST);
   COST2=LAG2(COST);
   IF _N_ GE 3 THEN MOV_AVE=MEAN (OF COST COST1 COST2);
   DROP COST1 COST2;
DATALINES;
1 2 3 4 . 6 8 12 8
;
PROC PRINT DATA=MOVING NOOBS;
   TITLE 'Listing of Data Set MOVING';
RUN;
```

The data set MOVING is:

OBS	COST	DAY	MOV_AVE
1	1	1	.
2	2	2	.
3	3	3	2.00000
4	4	4	3.00000
5	.	5	3.50000
6	6	6	5.00000
7	8	7	7.00000
8	12	8	8.66667
9	8	9	9.33333

Description. The LAG function returns the value of the argument the last time the function was executed. If you place the LAG function in a DATA step where it will be executed for every iteration of the DATA step, it will give you the value of the argument from the previous observation. There are also a family of LAG functions, LAG1, LAG2, LAG3, etc., which will return the value of the argument from the nth previous execution of the function. Thus, LAG (which is equivalent to LAG1) will return the value from a previous observation; LAG2 will return the value from the next earlier observation, and so forth. Notice how the use of a moving average "smooths out" the abrupt change on day 8. A note of caution here: it is usually inadvisable to execute the LAG function conditionally. Consider the following example:

```
DATA NEVER;
   INPUT X @@;
   IF X GT 3 THEN X_LAG = LAG(X);
DATALINES;
5 7 2 1 4
;
```

What are the values of X_LAG in the five observations? Answer: Missing, 5, missing, missing, and 7! Read the definition of the LAG function carefully, and see if you can understand what is happening here.

I. *Sorting Within an Observation*

We use PROC SORT to sort observations in a SAS data set. However, we may have occasion to sort within an observation. In the example that follows, five values (L1-L5) are recorded for each subject. We want to arrange the five values from highest to lowest. The method used here is known as a "bubble sort," because the highest (or lowest) values "bubble," or move closer, to the top each time the program returns to the computation loop.

```
*-----------------------------------------*
| Program to Sort within an Observation   |
*-----------------------------------------*;
DATA TEST;
   INPUT ID L1-L5;
DATALINES;
1 1 2 3 4 5
2 1 3 5 2 4
3 . 7 9 . .
;
DATA NEW;
   SET TEST;
   ARRAY L[5] L1-L5;
   LOOP: FLAG=0; ①
   DO I=1 TO 4; ③
      IF L[I+1] GT L[I] THEN DO; ④
         HOLD=L[I];
         L[I]=L[I+1];
         L[I+1]=HOLD;
         FLAG=1;
      END;
   END;
   IF FLAG=1 THEN GO TO LOOP; ②
   DROP I FLAG HOLD;
RUN;
```


Description. Statement ① contains a SAS label. A statement in a SAS program can be labeled by a one- to eight-character label, followed by a colon. We can then use a GO TO statement ② to control the logical flow of the program. The DO loop ③ runs through the data values pairwise, reversing the order of a pair if two values are not in descending order. A flag then gets set, so that the program knows to repeat the process until no more pair reversals are made. Notice that we have five variables to be sorted, and the DO loop runs from 1 to 4 since we have I+1 as a subscript inside the loop. To reverse the sorting order, substitute a LT operator for the GT operator in line ④.

For those of you who don't like GO TO statements, the program below also sorts values within an observation but doesn't use any GO TO statements:

```
DATA NEW;
   SET TEST;
   ARRAY L[5] L1-L5;
   FLAG = 1;
   DO UNTIL (FLAG = 0);
      FLAG = 0;
      DO I = 1 TO 4;
         IF L[I+1] GT L[I] THEN DO;
            HOLD = L[I];
            L[I] = L[I+1];
            L[I+1] = HOLD;
            FLAG = 1;
         END;
      END;
   END;
   DROP I FLAG HOLD;
RUN;
```

J. *Computing Coefficient Alpha (or KR-20) in a DATA Step*

This program computes a test statistic called coefficient alpha, which, for a test item that is dichotomous, is equivalent to the Kuder-Richardson formula 20. This statistic is available in PROC CORR with option ALPHA (Cronbach's alpha—see Chapter 11, Section F.). You may still want to use the code below to compute your KR-20. At the very least, it serves as a good programming example. The formula for Cronbach's alpha is:

$$\text{Alpha (or KR-20 if dichotomous)} = \frac{k}{k-1}\left(1 - \frac{\Sigma\text{ Item variances}}{\text{Test variance}}\right)$$

where k is the number of items in the test.

The key here is to output a data set that contains the item and test variances. Here is the program:

```
*-----------------------------------------------------*
| Assume that data set SCORE (see Chapter 11) contains |
| variables S1-S5 which are the scored responses for   |
| each of the 5 items on a test. S=1 for a correct     |
| answer, S=0 for an incorrect response                |
*-----------------------------------------------------*;
PROC MEANS NOPRINT DATA=SCORE;
   VAR S1-S5 RAW;
   OUTPUT OUT=VAROUT VAR=VS1-VS5 VRAW;
RUN;

DATA _NULL_; ①
   FILE PRINT; ②
   SET VAROUT;
   SUMVAR = SUM (OF VS1-VS5); ③
   KR20 = (5/4)*(1-SUMVAR/VRAW); ④
PUT KR20=; ⑤
RUN;
```

We use PROC MEANS to output a data set containing the item variances. The keyword VAR= computes variances for the variables listed in the VAR statement. This data set contains only one observation. In order to sum the item variances, we need to use another DATA step. You may not be familiar with the special SAS data set name _NULL_ ①. This reserved data set name instructs the SAS system to process the observations as they are encountered but not to write them to a temporary or permanent SAS data set. This saves time and, perhaps, money. Line ③ computes the sum of the item variances, and line ④ is the formula for coefficient alpha.

We get the program to print the results for us by using a PUT statement ⑤. The results of this PUT are sent to the same place that normal SAS output goes, because of the FILE PRINT statement ②.

Problems

As a special treat to you, this chapter does not contain any SAS programming problems (since they are not really appropriate). So your assignment, should you decide to accept it, is to go out and have a good time (on us of course)!

CHAPTER

20

Syntax Examples

A. *Introduction*

This chapter presents examples of each of the procedures listed above. Instead of a generalized, hard-to-understand syntax reference, we present several concrete examples that cover the majority of options and statements that you may want for each procedure. Simply replace our highlighted variable names and data set names with your own, and you are ready to go.

B. PROC ANOVA

One-way Design:

```
PROC ANOVA DATA=MYDATA;
   CLASS RACE;
   MODEL SCORE FINAL = RACE;
   MEANS RACE / SNK; ***SNK is Student Newman Keuls;
RUN;
```

Two-way Factorial Design (balanced only):

```
PROC ANOVA DATA=MYDATA;
   CLASS GROUP DOSE;
   MODEL HR SBP DBP = GROUP | DOSE;
   MEANS GROUP|DOSE / DUNCAN; ***DUNCAN multiple range test;
RUN;
```

One-way Repeated Measures Design (using the REPEATED statement):

```
PROC ANOVA DATA=MYDATA;
   MODEL SCORE1-SCORE4 = / NOUNI;
   REPEATED TIME;
RUN;
```

One-way Repeated Measures Design (without using the REPEATED statement):

```
PROC ANOVA DATA=MYDATA;
   CLASS SUBJ TIME;
   MODEL SCORE = SUBJ TIME;
   MEANS TIME / SNK;
RUN;
```

For more advanced factorial and repeated measures designs, refer to Chapters 7 and 8.

C. PROC APPEND

```
PROC APPEND BASE=BIG DATA=NEWDATA;
RUN;
```

D. PROC CHART

Vertical Bar Chart (Frequencies):

```
PROC CHART DATA=MYDATA;
   VBAR GENDER;
RUN;

PROC CHART DATA=MYDATA;
   VBAR DAY / DISCRETE;
RUN;
```


Horizontal Bar Chart:

```
PROC CHART DATA=MYDATA;
   HBAR GROUP;
RUN;
```

Bar Chart Where Bars Represent Sums or Means of a Variable:

```
PROC CHART DATA=MYDATA;
   VBAR REGION / SUMVAR=SALES TYPE=SUM;
RUN;
```

For more details on PROC CHART see Chapter 2.

E. PROC CONTENTS

```
PROC CONTENTS DATA=MYDATA POSITION;
RUN;

LIBNAME XXX 'C:\SASDATA';
PROC CONTENTS DATA=XXX._ALL_ POSITION;
RUN;
```

F. PROC CORR

Correlation Matrix:

```
PROC CORR DATA=MYDATA NOSIMPLE;
   VAR A B C X Y Z;
RUN;
```

Correlate One Variable with Several Others:

```
PROC CORR DATA=MYDATA NOSIMPLE;
   WITH QUES1-QUES50;
   VAR GRADE;
RUN;
```

For additional examples, see Chapter 5.

G. PROC DATASETS

```
LIBNAME XXX 'C:\SASDATA';
PROC DATASETS LIBRARY=XXX;
   MODIFY MYDATA;
      LABEL CANDID = 'CANDIDATE';
      RENAME OLDWT = WEIGHT;
      FORMAT COST DOLLAR7.;
RUN;
```

H. PROC FACTOR

Principal Components Analysis (with rotation):

```
PROC FACTOR DATA=FACTOR PREPLOT PLOT ROTATE=VARIMAX
          NFACTORS=2 OUT=FACT SCREE;
   VAR QUES1-QUES6;
RUN;
```

Factor Analysis with Squared Multiple Correlations and Oblique Rotation:

```
PROC FACTOR DATA=FACTOR PREPLOT PLOT ROTATE=PROMAX
          NFACTORS=2 OUT=FACT SCREE;
   VAR QUES1-QUES6;
   PRIORS SMC;
RUN;
```

See Chapter 10 for more examples and explanations.

I. PROC FORMAT

Temporary Character and Numeric Formats:

```
PROC FORMAT;
   VALUE $GENDER 'M' = 'Male'
                 'F' = 'Female';
   VALUE LIKERT   1 = 'Strongly Disagree'
                  2 = 'Disagree'
                  3 = 'No Opinion'
                  4 = 'Agree'
                  5 = 'Strongly Agree';
   VALUE WTGRP    LOW-<20 = 'Zero to 20'
                  20-<40  = '20 TO 40'
                  40-HIGH = '40 and Above';
   VALUE $CODES   'A','C','E' = 'Group 1'
                  'X','Y','Z' = 'Group 2';
   VALUE NUMS     1,4-8       = 'Range One'
                  2,3,9-11    = 'Range Two';
RUN;
```

Permanent Formats:

```
LIBNAME XXX 'C:\SASDATA';
OPTIONS FMTSEARCH=(XXX);
PROC FORMAT LIBRARY=XXX;
   VALUE $YESNO '1' = 'Yes'
                '0' = 'No';
RUN;
```


J. PROC FREQ

One-way Frequencies:

```
PROC FREQ DATA = MYDATA ORDER=FREQ;
   TABLES GENDER RACE GROUP / NOCUM;
RUN;
```

Two-way Frequencies (with request for chi-square):

```
PROC FREQ DATA=MYDATA;
   TABLES TREAT*OUTCOME / CHISQ;
RUN;
```

Three-way Frequencies (with a request for all statistics):

```
PROC FREQ DATA=MYDATA;
   TABLES STRATA*GROUP*OUTCOME / ALL;
RUN;
```

See Chapter 3 for more examples.

K. PROC GLM

One-way Design:

```
PROC GLM DATA=MYDATA;
   CLASS TREAT;
   MODEL Y = TREAT;
   CONTRAST 'A VS. B AND C' TREAT 2 -1 -1;
   CONTRAST 'B VS. C'       TREAT 0 1 -1;
   MEANS TREAT / SNK;
RUN;
```

Two-way Factorial Design (balanced or unbalanced):

```
PROC GLM DATA=MYDATA;
   CLASS TREAT GENDER;
   MODEL Z = TREAT | GENDER;
   LSMEANS TREAT | GENDER / SCHEFFE ALPHA=.1;
RUN;
```

For more examples of factorial designs, contrast statements, and repeated measures designs, see Chapters 7 and 8.

L. PROC LOGISTIC

```
PROC LOGISTIC DATA=LOGISTIC DESCENDING;
   MODEL ACCIDENT = AGE VISION DRIVE_ED /
                    SELECTION = FORWARD
                    CTABLE PPROB=(0 to 1 by .1)
                    LACKFIT
                    RISKLIMITS;
RUN;
```

For other examples of PROC LOGISTIC, see Chapter 9, Section F.

M. PROC MEANS

Descriptive Statistics on All Subjects Together:

```
PROC MEANS DATA=MYDATA N MEAN STD STDERR MAXDEC=2;
   VAR HR SBP DBP;
RUN;
```

Descriptive Statistics Broken Down by One Variable:

```
PROC MEANS DATA=MYDATA N MEAN STD STDERR MIN MAX MAXDEC=2;
   CLASS GROUP;
   VAR X Y Z;
RUN;
```

Creating an Output Data Set Containing Means and Variances:

```
PROC MEANS DATA=MYDATA NOPRINT NWAY;
   CLASS GENDER GROUP;
   ID SUBJ;
   VAR X Y Z;
   OUTPUT OUT=SUMMARY
          MEAN=M_X M_Y M_Z
          VAR =V_X V_Y V_Z;
RUN;
```

Using PROC MEANS to Run a Paired t-Test:

```
PROC MEANS DATA=MYDATA N MEAN STD STDERR T PRT;
   VAR DIFF;
RUN;
```

For more examples of PROC MEANS with and without creating an OUTPUT data set, see Chapter 2.

N. PROC NPAR1WAY

```
PROC NPAR1WAY DATA=MYDATA WILCOXON;
   CLASS GROUP;
   VAR WEIGHT;
   EXACT WILCOXON;
RUN;
```

See Chapter 6, Section D, for more details.

O. PROC PLOT

Simple X–Y Plot:

```
PROC PLOT DATA=MYDATA;
   PLOT Y*X;
RUN;
```

Choosing o's as Plotting Symbols:

```
PROC PLOT DATA=MYDATA;
   PLOT Y*X = 'o';
RUN;
```

Using the Value of Gender ('M' or 'F') as the Plotting Symbol:

```
PROC PLOT DATA=MYDATA;
   PLOT Y*X = GENDER;
RUN;
```

For more details, see Chapter 2, Section G.

P. PROC PRINT

Simple Listing with Variable Names as Column Headings:

```
PROC PRINT DATA=MYDATA;
   TITLE 'This is the Title of My Report';
   ID SUBJ_ID;
   VAR DATE HR SBP DBP;
RUN;
```

Simple Listing with Variable Labels as Column Headings:
(NOTE: In this example, the LABEL statement is included in the procedure. In other situations, the labels may be assigned in the DATA step and you would not need an additional LABEL statement in the PROC. The formats may have also been assigned previously.)

```
PROC PRINT DATA=MYDATA LABEL;
   TITLE1 'Fancier Report';
   TITLE2 'Compiled by J. Smith and R. Cody';
   TITLE3 '--------------------------------';
   FOOTNOTE 'Printed on recycled paper';
   ID SUBJ_ID;
   VAR DATE COST SBP DBP;
   FORMAT DATE MMDDYY8. COST DOLLAR8. SBP DBP 4.;
   LABEL SUBJ ID = 'Subject ID'
         DATE    = 'Date of Visit'
         COST    = 'Cost of Treatment';
RUN;
```

Q. PROC RANK

Create a New Data Set with R_X Representing the Rank of X:

```
PROC RANK DATA=MYDATA OUT=RANKDATA;
   VAR X;
   RANKS R_X;
RUN;
```

Using PROC RANK to Split the Group in Two (median split):

```
PROC RANK DATA=MYDATA OUT=NEWDATA GROUPS=2;
   VAR X;
RUN;
```

R. PROC REG

Simple Linear Regression:

```
PROC REG DATA=MYDATA;
   MODEL Y = X;
RUN;
```

Two Variable Regression:

```
PROC REG DATA=MYDATA;
   MODEL LOSS = DOSAGE EXERCISE / P R;
RUN;
```

Stepwise Multiple Regression:

```
PROC REG DATA=MYDATA;
   MODEL OUTCOME = INCOME SES AGE IQ /
                   SELECTION = STEPWISE;
RUN;
```

See Chapter 9 for more examples of PROC REG.

S. PROC SORT

Sorting a Data Set "In Place":

```
PROC SORT DATA=MYDATA;
   BY ID DATE;
RUN;
```

Example Creating an Output Data Set (Using KEEP and WHERE= data set options):

```
PROC SORT DATA=MYDATA(KEEP=ID HR SBP DBP GENDER
                      WHERE=(SBP GT 140)) OUT=OUTDATA;
   BY ID DATE;
RUN;
```

T. PROC TTEST;

```
PROC TTEST DATA=MYDATA;
   CLASS GENDER;
   VAR HR SBP DBP;
RUN;
```

U. PROC UNIVARIATE

```
PROC UNIVARIATE DATA=MYDATA NORMAL PLOT;
   VAR X Y Z;
RUN;
```

Solutions to Problems

CHAPTER 1

1-1 (a)
```
DATA COLLEGE;
   INPUT ID AGE GENDER $ GPA CSCORE;
DATALINES;
1 18 M 3.7 650
2 18 F 2.0 490
3 19 F 3.3 580
4 23 M 2.8 530
5 21 M 3.5 640
;
```
(b)
```
PROC MEANS DATA=COLLEGE;
   VAR GPA CSCORE;
RUN;
```
(c) Between the "INPUT" and "DATALINES" lines insert:
```
INDEX = GPA + 3*CSCORE/500;
```
Add to the end of the program:
```
PROC SORT DATA=COLLEGE;
   BY INDEX;
RUN;

PROC PRINT DATA = COLLEGE;
   TITLE 'Students in Index Order'; /* (optional) */
   ID ID;
   VAR GPA CSCORE INDEX;
RUN;
```

1-2 (a)
```
DATA TAXPROB;
   INPUT SS SALARY AGE RACE $;
   FORMAT SS SSN11.; /* (See Chapter 3 about FORMATS) */
DATALINES;
123874414 28000 35 W
646239182 29500 37 B
012437652 35100 40 W
018451357 26500 31 W
;
PROC MEANS DATA = TAXPROB N MEAN MAXDEC=0;
   TITLE 'Descriptive Statistics for Salary and Age';
   VAR SALARY AGE;
RUN;
```
(b) Add a line after the INPUT statement:
```
TAX = .30 * SALARY;
```
Add to the end of the program:


```
PROC SORT DATA=TAXPROB;
   BY SS;
RUN;

PROC PRINT DATA=TAXPROB;
   TITLE 'Listing of Salary and Taxes';
   ID SS;
   VAR SALARY TAX;
RUN;
```

1-3
```
1 DATA MISTAKE;
2    INPUT ID 1-3 TOWN 4-6 REGION 7-9 YEAR 11-12 BUDGET 13-14
3    VOTER TURNOUT 16-20
  (data lines go here)
  ;
4 PROC MEANS DATA=MISTAKE;
5    VAR ID REGION VOTER TURNOUT;
6    N,STD,MEAN;
7 RUN;
```

Line 3: Variable name cannot contain a blank. Variable name too long. (Actually, if we had two variables, VOTER and TURNOUT, the INPUT statement above would work since we can combine LIST input with column specifications. However, for this problem, we intended VOTER TURNOUT to represent a single variable.) Semicolon missing after TURNOUT 16-20.

Line 5: We probably don't want the mean ID. Also, would be more meaningful to use PROC FREQ for a categorical variable such as REGION.

DATALINES; missing.

Line 6: Options for PROC MEANS go on the PROC line between the word MEANS and the semicolon. The options must have a space between them, not a comma.

```
PROC MEANS DATA=MISTAKE N MEAN STD;
   VAR ---- ;
RUN;
```

1-4 We have a SAS data set with the variables AGE, GENDER, RACE, INCOME, MARITAL, and HOME (homeowner versus renter).

Code Book

Variable Name	Col(s)	Description and Formats
AGE	1	Age group of subject 1 = 10 to 19 2 = 20–29 3 = 30–39 4 = 40–49 5 = 50–59 6 = 60+
GENDER	2	Gender, 1 = male 2 = female
RACE	3	Race, 1 = white 2 = African Am. 3 = hispanic 4 = other
INCOME	4	Income group, 1 = 0 to $9,999 2 = 10,000 to 19,999 3 = 20,000 to 39,000 4 = 40,000 to 59,000 5 = 60,000 to 79,000 6 = 80,000 and over
MARITAL	5	Marital status, 1 = single 2 = married 3 = separated 4 = divorced 5 = widowed
HOME	6	Homeowner or renter, 1 = homeowner 0 = renter

```
DATA CEO;
   INPUT AGE 1 GENDER 2 RACE 3 INCOME 4 MARITAL 5 HOME 6;
DATALINES;
311411
122310
411221
(more data lines)
RUN;

PROC FREQ DATA=CEO ORDER=FREQ;
   *Note, the ORDER=FREQ option will list the frequencies in
    decreasing frequency order, i.e. the most frequent first;
   TITLE 'Frequencies and Contingency Tables for CEO Report';
   TABLES AGE GENDER RACE INCOME MARITAL HOME
          AGE*GENDER*RACE INCOME*AGE*GENDER MARITAL*HOME;
   *or whatever other combinations you are interested in;
RUN;

PROC CHART DATA=CEO;
   TITLE 'Histograms';
   VBAR AGE GENDER RACE INCOME MARITAL HOME / DISCRETE;
RUN;
```

1-5
```
DATA PROB1_5;
   INPUT ID RACE $ SBP DBP HR;
DATALINES;
001   W    130   80   60
002   B    140   90   70
003   W    120   70   64
004   W    150   90   76
005   B    124   86   72
;
PROC SORT DATA=PROB1_5;
   BY SBP;
RUN;

PROC PRINT DATA=PROB1_5 NOOBS;
   TITLE 'Race and Hemodynamic Variables';
   VAR ID RACE SBP DBP;
RUN;
```

1-6 Add the following line after the INPUT statement:

```
ABP = 2*DBP/3 + SBP/3;
or
ABP = DBP + (SBP-DBP)/3;
```

CHAPTER 2

2-1
```
PROC FREQ DATA=COLLEGE;
   TABLES GENDER;
RUN;
```

2-2
```
PROC FREQ DATA=TAXPROB;
   TABLES RACE / NOCUM;
RUN;
```

2-3 (a)
```
PROC CHART DATA=PROB2_3;
   VBAR GROUP;
RUN;
```


(b)
```
PROC PLOT DATA=PROB2_3;
   PLOT Y*X;
RUN;
```

(c)
```
PROC SORT DATA=PROB2_3;
   BY GROUP;
RUN;

PROC PLOT DATA=PROB2_3;
   BY GROUP;
   PLOT Y*X;
RUN;
```

Don't forget that you must have your data set sorted by the BY variables before you can use a BY statement in a PROC.

2-4 Program to read liver data and produce statistics:

```
DATA LIVER;
   INPUT SUBJ DOSE REACT LIVER_WT SPLEEN;
DATALINES;
1 1 5.4 10.2 8.9
2 1 5.9 9.8 7.3
(more data lines)
;
PROC SORT DATA=LIVER;
   BY DOSE; *Note, optional since already in dose order;
RUN;

PROC UNIVARIATE DATA=LIVER NORMAL PLOT;
   TITLE 'Distributions for Liver Data';
   VAR REACT -- SPLEEN;
RUN;

PROC UNIVARIATE DATA=LIVER NORMAL PLOT;
   BY DOSE;
   TITLE 'Distributions for Liver Data by Dose';
   VAR REACT -- SPLEEN;
RUN;
```

2-5
```
1   DATA 123;
2      INPUT AGE STATUS PROGNOSIS DOCTOR GENDER STATUS2
3      STATUS3;
4   (data lines)
    ;
5   PROC CHART DATA=123 BY GENDER;
6      VBAR STATUS
7      VBAR PROGNOSIS;
8   RUN;

9   PROC PLOT DATA=123;
10     DOCTOR BY PROGNOSIS;
11  RUN;
```

Line 1: Invalid data set name, cannot start with a number.
Line 2: PROGNOSIS has nine letters.
Line 2: Not really an error, but it would be better to list GENDER with the other demographic variables.
Line 2: Again, not an error, but an ID variable is desirable.
Lines 2 and 3: Boy, we're picky. If you have STATUS2 and STATUS3, STATUS should be STATUS1.

DATALINES; or CARDS; statement missing between lines 3 and 4.
Line 5: Two things wrong here: One, If you use a BY variable, the data set must be sorted in order of the BY variable; two, a semicolon is missing between PROC CHART and BY GENDER.
Line 6: Missing a semicolon at the end of the line.
Line 7: In case you thought this was an error, it isn't. You can have two (or more) VBAR statements with one PROC CHART.
Line 8: Missing the keyword PLOT before the plot request. Also, the plot request is of the form Y*X not Y BY X.

2-6 (a)
```
DATA SALES;
   INPUT PERSON $ TARGET $ VISITS CALLS UNITS;
DATALINES;
Brown    American   3  12  28000
Johnson  VRW        6  14  33000
Rivera   Texam      2   6   8000
Brown    Standard   0  22      0
Brown    Knowles    2  19  12000
Rivera   Metro      4   8  13000
Rivera   Uniman     8   7  27000
Johnson  Oldham     3  16   8000
Johnson  Rondo      2  14   2000
;
PROC MEANS DATA=SALES N SUM MEAN STD MAXDEC=0;
   CLASS PERSON;
   TITLE 'Sales Figures for Each Salesperson';
   VAR VISITS CALLS UNITS;
RUN;
```
(b)
```
PROC PLOT DATA=SALES;
   TITLE 'Sales Plots';
   PLOT VISITS*CALLS=PERSON;
RUN;
```
(c)
```
PROC CHART DATA=SALES;
   TITLE 'Distribution of Units Sold by Salesperson';
   VBAR PERSON /SUMVAR=UNITS TYPE=SUM;
RUN;
```
or
```
PROC CHART DATA=SALES;
   TITLE 'Distribution of Units Sold by Salesperson';
   VBAR UNITS /GROUP=PERSON;
RUN;
```
The first PROC CHART in part (c) above will produce a single bar for each salesperson, the height representing the total (sum) of the units sold. The alternate statements will produce an actual frequency distribution of the number of units sold, for each salesperson, in a side-by-side fashion.

2-7 A program to read these data and compute means would be:
```
DATA PROB2_7;
   INPUT ID TYPE $ SCORE;
DATALINES;
1   A    44
1   B     9
1   C   203
2   A    50
2   B     7
2   C   188
3   A    39
3   B     9
3   C   234
;
```



```
PROC MEANS DATA=PROB2_7;
   CLASS TYPE;
   VAR SCORE;
RUN;
```
If you use a BY statement instead of a CLASS statement, remember to sort your data set first.

CHAPTER 3

3-1
```
PROC FORMAT;
   VALUE FGROUP 1='CONTROL' 2='DRUG A'
                3='DRUG B';
RUN;
```
3-2
```
PROC FORMAT;
   VALUE $GENDER 'M'='Male'
                 'F'='Female';
   VALUE $PARTY '1'='Republican'
                '2'='Democrat'
                '3'='Not Registered';
   VALUE YESNO 0='No' 1='Yes';
RUN;

DATA SURVEY;
   INPUT ID 1-3 GENDER $ 4 PARTY $ 5
         VOTE 6 FOREIGN 7 SPEND 8;
   LABEL PARTY   = 'Political Party'
         VOTE    = 'Vote in Last Election?'
         FOREIGN = 'Agree with Government Policy?'
         SPEND   = 'Should we Increase Domestic Spending?';
   FORMAT GENDER $GENDER. PARTY $PARTY. VOTE FOREIGN SPEND YESNO.;
DATALINES;
007M1110
013F2101
137F1001
117 1111
428M3110
017F3101
037M2101
;
PROC FREQ DATA=SURVEY;
   TITLE 'Political Survey Results';
   TABLES GENDER PARTY VOTE FOREIGN SPEND;
   TABLES VOTE*(SPEND FOREIGN) / CHISQ;
RUN;
```
3-3 Method 1:
```
DATA DEMOG;
   INPUT WEIGHT HEIGHT GENDER $;
   *Create weight groups;
   IF 0 LE WEIGHT LT 101 THEN WTGRP=1;
   ELSE IF 101 LE WEIGHT LT 151 THEN WTGRP=2;
   ELSE IF 151 LE WEIGHT LE 200 THEN WTGRP=3;
   ELSE IF WEIGHT GT 200 THEN WTGRP=4;
   *Create height groups;
   IF 0 LE HEIGHT LE 70 THEN HTGRP=1;
   ELSE IF HEIGHT GT 70 THEN HTGRP=2;
DATALINES;
155 68 M
 98 60 F
```

```
202 72 M
280 75 M
130 63 F
;
PROC FREQ DATA=DEMOG;
   TABLES WTGRP*HTGRP;
RUN;
```

(NOTE: You may use <= instead of LE, < instead of LT, and > instead of GT.)

Method 2:

```
PROC FORMAT;
   VALUE WTFMT 0-100='1' 101-150='2' 151-200='3' 201-HIGH='4';
   VALUE HTFMT 0-70='1' 71-HIGH='2';

DATA DEMOG;
   INPUT WEIGHT HEIGHT GENDER $;
DATALINES;
155 68 M
 98 60 F
202 72 M
280 75 M
130 63 F
;
PROC FREQ DATA=DEMOG;
   TABLES WEIGHT*HEIGHT;
   FORMAT WEIGHT WTFMT. HEIGHT HTFMT.;
RUN;
```

3-4
```
DATA ASTHMA;
   INPUT ASTHMA $ SES $ COUNT;
DATALINES;
YES LOW   40
NO  LOW  100
YES HIGH  30
NO  HIGH 130
;
PROC FREQ DATA=ASTHMA;
   TITLE 'Relationship between Asthma and SES';
   TABLES SES*ASTHMA / CHISQ;
   WEIGHT COUNT;
RUN;
```

Chi-square = 4.026, p = .045.

3-5
```
DATA VITAMIN;
   INPUT V_CASE $ V_CONT $ COUNT;
   LABEL V_CASE = 'Case Use Vitamins'
         V_CONT = 'Control Use Vitamins';
***Note: Values of V_CASE and V_CONT chosen so that 1-YES
   will come before 2-NO in the table;
DATALINES;
1-YES 1-YES 100
1-YES 2-NO   50
2-NO  1-YES  90
2-NO  2-NO  200
;
PROC FREQ DATA=VITAMIN;
   TITLE 'Matched Case-control Study';
   TABLES V_CASE * V_CONT / AGREE;
   WEIGHT COUNT;
RUN;
```


Chi-square (McNemar) = 11.429, p = .001 More likely to develop disease X if you do not use vitamins. (Remember, it is only the discordant pairs (yes/no or no/yes) that contribute to the McNemar Chi-square.)

3-6
```
DATA VDT_USE;
   INPUT GROUP $ VDT $ COUNT;
DATALINES;
CASE  1-YES 30
CASE  2-NO  50
CONTROL 1-YES  90
CONTROL 2-NO   200
;
PROC FREQ DATA=VDT_USE;
   TITLE 'Case-control study of VDT Use';
   TABLES VDT * GROUP / CHISQ CMH;
   WEIGHT COUNT;
RUN;
```

Chi-square = .274, p > .05 OR = 1.333, 95% CI (.796, 2.234).

3-7
```
DATA CLASS;
   INPUT TYPE : $10. PROBLEM $ COUNT;
DATALINES;
1-STANDARD 1-YES 30
1-STANDARD 2-NO 220
2-PROOFED  1-YES 20
2-PROOFED  2-NO 280
;
PROC FREQ DATA=CLASS;
   TITLE 'Sound Proofing Study';
   TABLES TYPE * PROBLEM / CHISQ CMH;
   WEIGHT COUNT;
RUN;
```

RR = 1.800 (room noise increases the incidence of problems), 95% CI (1.057, 3.065).

3-8
```
PROC FORMAT;
   VALUE SIZE 1 = 'Small' 2 = 'Medium' 3 = 'Large' 4 = 'Gigantic';
RUN;

DATA CLASS;
   INPUT SIZE PROBLEM $ COUNT @@;
   FORMAT SIZE SIZE.;
DATALINES;
1 1-YES 3    1 2-NO 12    2 1-YES 6     2 2-NO 22
3 1-YES 17   3 2-NO 38    4 1-YES 80    4 2-NO 120
;
PROC FREQ DATA=CLASS;
   TITLE 'Relationship Between Class Size and Behavior';
   TABLES PROBLEM * SIZE / CHISQ;
   WEIGHT COUNT;
RUN;
```

Chi-square test for trend = 6.038, p = .014.

NOTE: The chi-square for the 2 by 4 table is 6.264, with p = .094.

3-9
```
DATA TEMP;
   INPUT T_CONTRL $ GROUP : $10. COLD $ COUNT;
DATALINES;
1-POOR SMOKERS 1-YES 30
1-POOR SMOKERS 2-NO 50
1-POOR NONSMOKERS 1-YES 40
1-POOR NONSMOKERS 2-NO 100
```

```
2 -GOOD SMOKERS 1-YES 20
2 -GOOD SMOKERS 2-NO 55
2 -GOOD NONSMOKERS 1-YES 35
2 -GOOD NONSMOKERS 2-NO 150
;
PROC FREQ DATA=TEMP;
   TITLE 'Relationship Between Temperature Control and Colds';
   TABLES GROUP * T_CONTRL * COLD / ALL;
   WEIGHT COUNT;
RUN;
```

The overall RR for the combined tables = 1.468.
The 95% CI is (1.086, 1.985).
The p-value is .013.

3-10
```
PROC FORMAT;
   VALUE PROB 1 = 'Cold' 2 = 'Flu' 3 = 'Trouble Sleep'
              4 = 'Chest Pain'  5 = 'Muscle Pain' 6 = 'Headache'
              7 = 'Overweight'  8 = 'High BP' 9 = 'Hearing Loss';
RUN;

DATA PATIENT;
   INPUT SUBJ 1-2 PROB1 3 PROB2 4 PROB3 5 HR 6-8 SBP 9-11 DBP 12-14;
DATALINES;
11127 78130 80
1787  82180110
031   62120 78
4261  68130 80
89    58120 76
9948  82178100
;
PROC MEANS DATA=PATIENT N MEAN STD MAXDEC = 1;
   TITLE 'Statistics from Patient Data Base';
   VAR HR SBP DBP;
RUN;
```

For part (b) add:

(Solution without arrays)

```
DATA PROBLEM;
     SET PATIENT;
     PROB = PROB1;
     IF PROB NE . THEN OUTPUT;
     PROB = PROB2;
     IF PROB NE . THEN OUTPUT;
     PROB = PROB3;
     IF PROB NE . THEN OUTPUT;
     FORMAT PROB PROB.;
     KEEP PROB;
RUN

PROC FREQ DATA=PROBLEN;
     TABLES PROB;
RUN;
```

(Solution with arrays)

```
DATA PROBLEM;
   SET PATIENT;
   ARRAY XPROB[3] PROB1-PROB3;
   DO I = 1 TO 3;
      PROB = XPROB[I];
      IF PROB NE . THEN OUTPUT;
   END;
   FORMAT PROB PROB.;
   KEEP PROB;
RUN;

PROC FREQ DATA=PROBLEM;
   TABLES PROB;
RUN;
```

3-11 **Line 3:** The formats cannot be assigned to variables before they have been defined. Therefore, move lines 5 through 8 to the beginning of the program (before line 1).
Line 11: PROC FREQ uses the keyword TABLES, not VAR, to specify a list of variables.
Line 11: You cannot use the CHISQ option unless a two-way table (or higher order) is specified. That is, we could have written:


```
PROC FREQ DATA=IGOOFED;
   TABLES GENDER*RACE / CHISQ;
RUN;
```

Line 14: You cannot use a BY statement unless the data set has been sorted first by the same variable.

CHAPTER 4

4-1
```
DATA PROB4_1;
   INPUT @1 ID 3.
         @5 (DOB ST_DATE END_DATE)(MMDDYY6.)
         @23 SALES 4.;
   AGE = (ST_DATE - DOB) / 365.25;
   *For section E, substitute the line below for AGE;
   AGE = INT((ST_DATE - DOB) / 365.25);
   LENGTH = (END_DATE - ST_DATE) / 365.25;
   *or LENGTH = (END_DATE - ST_DATE + 1) / 365.25;
   SALES_YR = SALES / LENGTH;
   *For section, E substitute the line below for SALES_YR;
   SALES_YR = ROUND ((SALES/LENGTH),10);
   FORMAT DOB MMDDYY8. SALES_YR DOLLAR6.;
DATALINES;
001 1021461112801228888 7343
002 0913550202800204908 8123
005 0606400312810312855 0000
003 0705441115801113908 9544
;
PROC PRINT DATA=PROB4_1;
TITLE 'Report for Homework Problem 4-1';
   ID ID;
   VAR DOB AGE LENGTH SALES_YR;
RUN;
```

4-2
```
DATA RATS;
   INPUT @1  RAT_NO  1.
         @3  DOB     DATE7.
         @11 DISEASE DATE7.
         @19 DEATH   DATE7.
         @27 GROUP   $1.;
   BIR_TO_D = DISEASE - DOB;
   DIS_TO_D = DEATH - DISEASE;
   AGE = DEATH - DOB;
   FORMAT DOB DISEASE DEATH MMDDYY8.;
DATALINES;
1 23MAY90 23JUN90 28JUN90 A
2 21MAY90 27JUN90 05JUL90 A
3 23MAY90 25JUN90 01JUL90 A
4 27MAY90 07JUL90 15JUL90 A
5 22MAY90 29JUN90 22JUL90 B
6 26MAY90 03JUL90 03AUG90 B
7 24MAY90 01JUL90 29JUL90 B
8 29MAY90 15JUL90 18AUG90 B
;
PROC MEANS DATA=RATS MAXDEC=1 N MEAN STD STDERR;
   CLASS GROUP;
   VAR BIR_TO_D -- AGE;
RUN;
```

4-3
```
DATA PATIENTS;
    INPUT @1  ID      3.
          @4  DATE    MMDDYY6.
          @10 HR      3.
          @13 SBP     3.
          @16 DBP     3.
          @19 DX      3.
          @22 DOCFEE  4.
          @26 LABFEE  4.;
  FORMAT DATE MMDDYY8.;
DATALINES;
00710218307012008001400400150
00712018307213009002000500200
00909038306611007013700300000
00507058307414008201300900000
00501158208018009601402001500
00506188207017008401400800400
00507038306414008401400800200
;
PROC SORT DATA=PATIENTS;
   BY ID DATE;
RUN;
DATA PROB4_3;
   SET PATIENTS;
   BY ID;
*Omit the first VISIT for each patient;
IF NOT FIRST.ID;
RUN;

PROC MEANS DATA=PROB4_3 NOPRINT NWAY;
   CLASS ID;
   VAR HR SBP DBP;
   OUTPUT OUT=PAT_MEAN MEAN=;
RUN;
```

4-4
```
PROC SORT DATA=PATIENTS; ***From problem 4-3;
   BY ID;
RUN;
DATA PROB4_4;
   SET PATIENTS;
   BY ID;
   *Omit patients with only one visit;
IF FIRST.ID AND LAST.ID THEN DELETE;
RUN;

PROC MEANS DATA=PROB4_4 NOPRINT NWAY;
   CLASS ID;
   VAR HR SBP DBP;
   OUTPUT OUT=PAT_MEAN MEAN=;
RUN;
```

4-5
```
***Program to create data set BLOOD;
DATA BLOOD;
   LENGTH GROUP $ 1;
   INPUT ID GROUP $ TIME WBC RBC @@;
DATALINES;
1 A 1 8000 4.5  1 A 2 8200 4.8  1 A 3 8400 5.2
1 A 4 8300 5.3  1 A 5 8400 5.5
2 A 1 7800 4.9  2 A 2 7900 5.0
3 B 1 8200 5.4  3 B 2 8300 5.4  3 B 3 8300 5.2
3 B 4 8200 4.9  3 B 5 8300 5.0
4 B 1 8600 5.5
5 A 1 7900 5.2  5 A 2 8000 5.2  5 A 3 8200 5.4
5 A 4 8400 5.5
;
PROC MEANS DATA=BLOOD NWAY NOPRINT;
   CLASS ID;
   ID GROUP;
   VAR WBC RBC;
   OUTPUT OUT=TEMP(WHERE=(_FREQ_ GT 2) DROP=_TYPE_) MEAN=;
RUN;

PROC PRINT DATA=TEMP NOOBS;
   TITLE 'Listing of data set TEMP';
RUN;
```


4-6 Replace the OUTPUT statement of PROC MEANS with:

```
OUTPUT OUT=TEMP(WHERE=(_FREQ_ GT 2) DROP= _TYPE_)
       MEAN=  STD=SD_WBC SD_RBC;
```

CHAPTER 5

5-1 (a)
```
DATA PROB5_1;
   INPUT X Y Z;
DATALINES;
1 3 15
7 13 7
8 12 5
3 4 14
4 7 10
;
PROC CORR DATA=PROB5_1;  /* x vs. y r=  .965 p=.0078 */
   VAR X;                /* x vs. z r= -.975 p=.0047 */
   WITH Y Z;
RUN;
```

(b)
```
PROC CORR DATA=PROB5_1;  /* y vs. z r= -.963 p=.0084 */
   VAR X Y Z;
RUN;
```

5-2
```
DATA PRESSURE;
   INPUT AGE SBP;
DATALINES;
15 116
20 120
25 130
30 132
40 150
50 148
;
PROC CORR DATA=PRESSURE;
   VAR AGE SBP;
RUN;
```

5-3 (a)
```
PROC REG DATA=PROB5_1;
   MODEL Y = X;
RUN;
```

Intercept = .781, prob > $|T|$ = .5753,
Slope = 1.524, prob > $|T|$ = .0078.

5-4
```
DATA PROB5_4;
   INPUT X Y Z;
   LX = LOG(X);
   LY = LOG(Y);
   LZ = LOG(Z);
DATALINES;
1 3 15
7 13 7
8 12 5
3 4 14
4 7 10
;
PROC CORR DATA=PROB5_4;
   VAR LX LY LZ;
RUN;
```

5-5 (a)
```
PROC PLOT DATA=PROB5_1;
   PLOT Y*X;
RUN;
```

(b)
```
PROC REG DATA=PROB5_1;
   MODEL Y = X;
   PLOT PREDICTED.*X='P' Y*X='o'/ OVERLAY;
RUN;
```

You may use any plotting symbol you wish for the two plots; we used P's and o's.

5-6 Sections (a–c):

```
DATA PROB5_6;
  INPUT COUNTY POP HOSPITAL FIRE_CO RURAL $;
DATALINES;
1  35  1  2   YES
2  88  5  8    NO
3   5  0  1   YES
4  55  3  3   YES
5  75  4  5    NO
6 125  5  8    NO
7 225  7  9   YES
8 500 10 11    NO
;
PROC UNIVARIATE DATA=PROB5_6 NORMAL PLOT;
  TITLE 'Checking the Distributions';
  VAR POP HOSPITAL FIRE_CO;
RUN;

PROC CORR DATA=PROB5_6 NOSIMPLE PEARSON SPEARMAN;
  TITLE 'Correlation Matrix';
  VAR POP HOSPITAL FIRE_CO;
RUN;
```

Because of the outliers in the population variable, we prefer the Spearman correlation for this problem.

(d) We can use the output from UNIVARIATE to find the medians and do the recoding. In Chapter 6 we will see that PROC RANK can be used to produce a median cut automatically by using the GROUPS=2 option. For now, we will recode the variables using formats. You can also create new variables in the data step with IF statements.

```
PROC FORMAT;
  VALUE POP LOW-81='Below median' 82-HIGH='Above Median';
  VALUE HOSPITAL LOW-4='Below Nedian' 5-HIGH='Above Median';
  VALUE FIRE_CO LOW-6='Below Median' 7-HIGH='Above Median';
RUN;
```



```
PROC FREQ DATA=PROB5_6; ***Data set from above;
  TITLE 'Cross Tabulations';
  FORMAT POP POP. HOSPITAL HOSPITAL. FIRE_CO FIRE_CO.;
  TABLES RURAL*(POP HOSPITAL FIRE_CO) / CHISQ;
RUN;
```

5-7 **Line 1:** Incorrect data set name, cannot contain a dash.

Lines 3-5: These lines will recode missing values to 1, which we probably do not want to do. the correct form of these statements is:

```
IF X LE 0 AND X NE . THEN X=1;
```

Line 10: The options PEARSON and SPEARMAN do not follow a slash. The line should read:

```
PROC CORR DATA=MANY_ERR PEARSON SPEARMAN;
```

Line 11: The correct form for a list of variables where the "root" is not the same is:

```
VAR X -- LOGZ;
```

Remember, the single dash is used for a list of variables such as ABC1 - ABC25.

CHAPTER 6

6-1
```
DATA HEADACHE;
   INPUT TREAT $ TIME @@;
DATALINES;
A 40 A 42 A 48 A 35 A 62 A 35
T 35 T 37 T 42 T 22 T 38 T 29
;
PROC TTEST DATA=HEADACHE;
   CLASS TREAT;
   VAR TIME;
RUN;
```

Not significant at the .05 level (t = 1.93, p = .083).

6-2
```
PROC NPAR1WAY DATA=HEADACHE WILCOXON;
   TITLE 'Nonparametric Comparison';
   CLASS TREAT;
   VAR TIME;
   EXACT WILCOXON;
RUN;
```

Sum of ranks for A = 48.5; for B, 29.5.
Exact two-sided p = .1385
Approximation using a normal approximation with a continuity correction z = 1.45, p = .146.

6-3 Use a paired t-test. We have

```
DATA PAIR;
   INPUT SUBJ A_TIME B_TIME;
   DIFF = A_TIME - B_TIME;
DATALINES;
1 20 18
2 40 36
3 30 30
4 45 46
5 19 15
6 27 22
7 32 29
8 26 25
;
```

```
PROC MEANS DATA=PAIR N MEAN STD STDERR T PRT MAXDEC=3;
   VAR DIFF;
RUN;
```

t = + 3.00, p = .0199; drug B works faster.

6-4
```
PROC FORMAT;
   VALUE GROUP 0='A' 1='B' 2='C';
RUN;

DATA RANDOM;
   INPUT SUBJ @@;
   GROUP = RANUNI(0); *NOTE: CAN ALSO USE UNIFORM FUNCTION;
DATALINES;
001 137 454 343 257 876 233 165 002
;
PROC RANK DATA=RANDOM OUT=RANKED GROUP=3;
   VAR GROUP;
RUN;

PROC SORT DATA=RANKED;
   BY SUBJ;
RUN;

PROC PRINT DATA=RANKED;
   TITLE 'Listing of Subject Numbers and Group Assignments';
   FORMAT GROUP GROUP.;
   ID SUBJ;
   VAR GROUP;
RUN;
```

6-5 **Line 2:** Variable name HEARTRATE too long.
Line 11: Correct procedure name is TTEST.

CHAPTER 7

7-1
```
DATA BRANDTST;
   DO BRAND='A','N','T';
      DO SUBJ=1 TO 8;
         INPUT TIME @;
         OUTPUT;
      END;
   END;
DATALINES;
8 10 9 11 10 10 8 12
4 7 5 5 6 7 6 4
12 8 10 10 11 9 9 12
;
PROC ANOVA DATA=BRANDTST;
   CLASSES BRAND;
   MODEL TIME=BRAND;
   MEANS BRAND / DUNCAN;
RUN;
```

F = 28.89, p = .0001; N is significantly lower than either T or A (p < .05). T and A are not significantly different (p > .05).

7-2
```
DATA BOUNCE;
   DO AGE = 'NEW', 'OLD';
      DO BRAND = 'W','P';
         DO I = 1 TO 5;
            INPUT BOUNCES @;
            OUTPUT;
         END;
      END;
   END;
   DROP I;
DATALINES;
67 72 74 82 81 75 76 80 72 73
46 44 45 51 43 63 62 66 62 60
;
PROC ANOVA DATA=BOUNCE;
   TITLE 'Two-way ANOVA (AGE by BRAND) for Tennis Balls';
   CLASSES AGE BRAND;
   MODEL BOUNCES = AGE | BRAND;
   MEANS AGE | BRAND;
RUN;
```


NOTE: A simpler INPUT statement could have been used:

```
INPUT BRAND $ AGE $ BOUNCES;
```

With the data listed one number per line such as:

```
W NEW 67
P NEW 75
   etc.
```

Both main effects (AGE and BRAND) are significant (p = .0001 and .0002, respectively). The interaction is also significant, p = .0002.

7-3 (a)
```
DATA SODA;
   INPUT BRAND $ AGEGRP RATING;
DATALINES;
C 1 7
C 1 6
C 1 6
C 1 5
C 1 6
P 1 9
P 1 8
P 1 9
P 1 9
P 1 9
P 1 8
C 2 9
C 2 8
C 2 8
C 2 9
C 2 7
C 2 8
C 2 8
P 2 6
P 2 7
P 2 6
P 2 6
P 2 5
;
PROC GLM DATA=SODA;
   TITLE 'Two-way Unbalanced ANOVA';
   CLASSES BRAND AGEGRP;
   MODEL RATING = BRAND | AGEGRP;
   MEANS BRAND | AGEGRP;
RUN;
```

(b)
```
PROC MEANS DATA=SODA NWAY NOPRINT;
   CLASS BRAND AGEGRP;
   VAR RATING;
   OUTPUT OUT=MEANS MEAN=;
RUN;

PROC PLOT DATA=MEANS;
   PLOT RATING*AGEGRP=BRAND;
RUN;
```

(c)
```
PROC SORT DATA=SODA;
   BY AGEGRP;
RUN;

PROC TTEST DATA=SODA;
   BY AGEGRP;
   CLASS BRAND;
   VAR RATING;
RUN;
```

7-4
```
PROC TTEST DATA=BRANDTST;
   WHERE BRAND='A' OR BRAND='T';
   /* Alternative: WHERE BRAND IN ('A','T'); */
   /* WHERE BRAND NE 'N'; is not as desirable, since
      in a more general data set, there may be missing
      or miscoded values */
   CLASS BRAND;
   VAR TIME;
RUN;
```

7-5 **Line 4:** Since this is a two-way unbalanced design, PROC GLM should be used instead of PROC ANOVA.

7-6
```
***Part A;
DATA PROB7_6;
   DO GROUP = 'A','B','C';
      INPUT M_SCORE AGE @;
      OUTPUT;
   END;
DATALINES;
90 16 92 18 97 18
88 15 88 13 92 17
72 12 76 12 88 16
82 14 78 14 92 17
65 12 90 17 99 17
74 13 68 12 82 14
;
PROC ANOVA DATA=PROB7_6;
   CLASS GROUP;
   MODEL M_SCORE AGE = GROUP;
   MEANS GROUP /SNK;
RUN;

***Part B;
PROC GLM DATA=PROB7_6;
   TITLE 'Testing Assumption of Homogeneity of Slope';
   CLASS GROUP;
   MODEL M_SCORE = AGE GROUP AGE*GROUP;
RUN;
/* Interaction term not significant. OK to do analysis of
   covariance */
```



```
***Part C;
PROC GLM DATA=PROB7_6;
   TITLE 'Analysis of Covariance';
   CLASS GROUP;
   MODEL M_SCORE = AGE GROUP;
   LSMEANS GROUP /PDIFF;
RUN;
```

In the unadjusted analysis, the groups are significantly different (p = .0479) and the ages are nearly significant (p = .0559). The null hypothesis that the slopes are equal among the three groups is not rejected (AGE*GROUP interaction p = .1790). Adjusting for age, the group differences on math scores disappears completely (p = .7606).

CHAPTER 8

8-1
```
DATA SHIRT;
   INPUT (JUDGE BRAND COLOR WORK OVERALL)(1.);
   INDEX = (3*OVERALL + 2*WORK + COLOR) /6.0;
DATALINES;
(data lines)
;
PROC ANOVA DATA=SHIRT;
   CLASSES JUDGE BRAND;
   MODEL COLOR WORK OVERALL INDEX = JUDGE BRAND;
   MEANS BRAND / DUNCAN;
RUN;
```

8-2
```
DATA WATER;
   INPUT ID 1-3 CITY $ 4 RATING 5;
DATALINES;
(data lines)
;
PROC ANOVA DATA=WATER;
   CLASSES ID CITY;
   MODEL RATING = ID CITY;
   MEANS CITY /SNK;
RUN;
```

8-3
```
PROC FORMAT;
   VALUE CITY 1='New York' 2='New Orleans'
              3='Chicago' 4='Danver';
RUN;

DATA PROB8_3;
   INPUT JUDGE 1-3 @;
   DO CITY=1 TO 4;
      INPUT TASTE 1. @;
      OUTPUT;
   END;
   FORMAT CITY CITY.;
DATALINES;
0018685
0025654
0037464
0047573
;
***Same PROC ANOVA statements as problem 7-2 except for the
Data Set Name;
```

```
***Solution using the REPEATED statement of PROC ANOVA;
DATA REPEAT;
   INPUT ID 1-3 @4 (RATING1-RATING4)(1.);
DATALINES;
0018685
0025654
0037464
0047573
;

PROC ANOVA DATA=REPEAT;
   MODEL RATING1-RATING4 = /NOUNI;
   REPEATED CITY;
RUN;
```

The unadjusted comparison shows that the cities are not all equal (p = .0067). Using the Greenhouse-Geisser correction, the p-value is .0375 and, using the Huynh-Feldt correction, the p-value is .0108. Therefore, you should feel comfortable in rejecting the null hypothesis at the .05 level, regardless of which correction (if any) you use.

8-4
```
DATA RATS;
   INPUT GROUP $ RATNO DISTAL PROXIMAL;
DATALINES;
N 1 34 38
N 2 28 38
N 3 38 48
N 4 32 38
D 5 44 42
D 6 52 48
D 7 46 46
D 8 54 50
;
PROC ANOVA DATA=RATS;
   CLASSES GROUP;
   MODEL DISTAL PROXIMAL = GROUP / NOUNI;
   REPEATED LOCATION 2;
RUN;
```

Although the main effects are significant (GROUP p = .01, LOCATION p = .0308) the interaction term is highly significant (GROUP*LOCATION interaction F = 31.58, p = .0014). We should look carefully at the interaction graph to see exactly what is going on.

8-5 The DO loops are in the wrong order, and the OUTPUT statement is missing. Lines 2 through 8 should read:

```
DO SUBJ=1 TO 3;
   DO GROUP='CONTROL','DRUG';
      DO TIME='BEFORE','AFTER';
         INPUT SCORE @;
         OUTPUT;
         END;
      END;
   END;
```

There are no other errors.

CHAPTER 9

9-1
```
DATA TOMATO;
   DO LIGHT=1 TO 3;
      DO WATER=1 TO 2;
         DO I=1 TO 3;
            INPUT YIELD @;
```



```
            OUTPUT;
         END;
      END;
   END;
   DROP I;
DATALINES;
12 9 8 13 15 14 16 14 12 20 16 16 18 25 20 25 27 29
;
PROC REG DATA=TOMATO;
   MODEL YIELD = LIGHT WATER;
RUN;
```

9-2
```
DATA LIBRARY;
   INPUT BOOKS ENROLL DEGREE AREA;
DATALINES;
 4  5  3  20
 5  8  3  40
10 40  3 100
 1  4  2  50
.5  2  1 300
 2  8  1 400
 7 30  3  40
 4 20  2 200
 1 10  2   5
 1 12  1 100
;
PROC REG DATA=LIBRARY;
   MODEL BOOKS = ENROLL DEGREE AREA / SELECTION = FORWARD;
RUN;
```

9-3
```
DATA PROB9_3;
   INPUT GPA HS_GPA BOARD IQ;
DATALINES;
3.9    3.8   680    130
3.9    3.9   720    110
3.8    3.8   650    120
3.1    3.5   620    125
2.9    2.7   480    110
2.7    2.5   440    100
2.2    2.5   500    115
2.1    1.9   380    105
1.9    2.2   380    110
1.4    2.4   400    110
;
PROC REG DATA=PROB9_3;
   MODEL GPA = HS_GPA BOARD IQ / SELECTION=MAXR;
RUN;
```

9-4
```
DATA PEOPLE;
   INPUT HEIGHT WAIST LEG ARM WEIGHT;
DATALINES;
(data lines)
;
PROC CORR DATA=PEOPLE NOSIMPLE;
   VAR HEIGHT--WEIGHT;
RUN;

PROC REG DATA=PEOPLE;
   MODEL WEIGHT = HEIGHT WAIST LEG ARM /
                  SELECTION = STEPWISE;
RUN;
(You may also use FORWARD, BACKWARD, or MAXR instead of STEPWISE)
```

9-5 Ha! No errors here. As a matter of fact, you can use this program for problem 9-4.

9-6
```
DATA CATEGOR;
   SET LIBRARY;***From Problem 9-2;
   MASTERS = 0;
   PH_D = 0;
   IF DEGREE = 2 THEN MASTERS = 1;
   ELSE IF DEGREE = 3 THEN PH_D = 1;
RUN;

PROC REG DATA=CATEGOR;
   MODEL BOOKS = ENROLL AREA MASTERS PH_D /
      SELECTION = FORWARD;
RUN;
```

9-7
```
*----------------------------------------*
|Program Name: PROB9_7.SAS in C:\APPLIED |
|Purpose: Solution to homework problem 9-7|
|Date: June 29, 1996                      |
*----------------------------------------*;
PROC FORMAT;
   VALUE YES_NO  0 = 'No'
                 1 = 'Yes';
   RUN;

DATA LOGISTIC;
   INPUT ACCIDENT DRINK PREVIOUS;

   LABEL ACCIDENT = 'Accident in Last Year?'
         DRINK    = 'Drinking Problem?'
         PREVIOUS = 'Accident in Previous Year?';

   FORMAT ACCIDENT DRINK PREVIOUS YES_NO.;
DATALINES;
(data lines)
;
PROC LOGISTIC DATA=LOGISTIC DESCENDING;
   TITLE 'Predicting Accidents Using Logistic Regression';
   MODEL ACCIDENT = DRINK PREVIOUS /
                    SELECTION = FORWARD
                    RISKLIMITS;
RUN;
QUIT;
```

The logistic regression equation is:

```
LOG(odds of accident) = -1.9207 + 1.9559 (DRINK)
                                + 1.7770 (PREVIOUS).
```

The odds and probability of an accident for person 1 (no drinking history, no previous accidents) are .1465 and .1278 respectively. For person 2 (history of a drinking problem but no previous accident history), they are 1.0358 and .5088 respectively. The odds ratio is 1.0358.1465 = 7.07, which agrees with the PROC LOGISTIC output.

CHAPTER 10

10-1
```
DATA QUEST;
   INPUT ID 1-3 AGE 4-5 GENDER $ 6 RACE $ 7 MARITAL $ 8
         EDUC $ 9 PRES 10 ARMS 11 CITIES 12;
DATALINES;
001091113232
002452222422
003351324442
004271111121
005682132333
006651243425
;
PROC FACTOR DATA=QUEST ROTATE=VARIMAX
                 NFACTORS=2 OUT=FACT;
   TITLE 'Example of Factor Analysis';
   VAR PRES ARMS CITIES;
   PRIORS SMC;
RUN;
```


10-2
```
DATA SCORE;
   ARRAY ANS[5] $ 1 ANS1-ANS5;
   ARRAY KEY[5] $ 1 KEY1-KEY5;
   ARRAY S[5] 3 S1-S5; ***Score array 1=right,0=wrong;
   RETAIN KEY1-KEY5;

   IF _N_ = 1 THEN INPUT (KEY1-KEY5)($1.);

   INPUT @1 ID 1-9
         @11 (ANS1-ANS5)($1.);

   DO I = 1 TO 5;
      S[I] = KEY[I] EQ ANS[I];
   END;

DATALINES;
ABCDE
123456789 ABCDE
035469871 BBBBB
111222333 ABCBE
212121212 CCCDE
867564733 ABCDA
876543211 DADDE
987876765 ABEEE
;
PROC FACTOR DATA=SCORE OUT=FACTDATA NFACTORS=1;
   TITLE 'Factor Analysis of Test Data';
   VAR S1-S5;
   PRIORS SMC;
RUN;

PROC PRINT DATA=FACTDATA;
   TITLE 'Listing of Data Set FACTDATA';
RUN;
```

CHAPTER 11

11-1
```
*-------------------------------------------------------*
| Program to score a five item multiple choice exam.    |
| Data: The first line is the answer key, remaining lines|
|       contain the student responses                   |
*-------------------------------------------------------*;
DATA SCORE;
   ARRAY ANS[5] $ 1 ANS1-ANS5; ***Student answers;
   ARRAY KEY[5] $ 1 KEY1-KEY5; ***Answer key;
   ARRAY S[5] 3 S1-S5; ***Score array 1=right,0=wrong;
   RETAIN KEY1-KEY5;
```

```
***Read the answer key;
IF _N_ = 1 THEN INPUT (KEY1-KEY5)($1.);

***Read student responses;
INPUT @1 SS 1-9
      @11 (ANS1-ANS5) ($1.);

***Score the test;
DO I = 1 TO 5;
   S[I] = KEY[I] EQ ANS[I];
END;

***Compute Raw and Percentage scores;
RAW = SUM (OF S1-S5);
PERCENT = 100*RAW / 5;

KEEP SS RAW PERCENT S1-S5; ***S1-S5 needed for 11-2;

LABEL SS = 'Social Security Number'
      RAW = 'Raw Score'
      PERCENT = 'Percent Score';
DATALINES;
BCDAA
123456789 BCDAA
001445559 ABCDE
012121234 BCCAB
135632837 CBDAA
005009999 ECECE
789787878 BCDAA
;
PROC SORT DATA=SCORE;
   BY SS;
RUN;

PROC PRINT DATA=SCORE LABEL;
   TITLE 'Listing of Student Scores in SS Order';
   ID SS;
   VAR RAW PERCENT;
   FORMAT SS SSN11.;
RUN;

PROC SORT DATA=SCORE;
   BY DESCENDING RAW;
RUN;

PROC PRINT DATA=SCORE LABEL;
   TITLE 'Listing of Student Scores in Decreasing Order';
   ID SS;
   VAR RAW PERCENT;
   FORMAT SS SSN11.;
RUN;
```

11-2
```
PROC CORR DATA=SCORE ALPHA NOSIMPLE;
   TITLE 'Computing KR-20';
   VAR S1-S5;
RUN;

PROC CORR DATA=SCORE NOSIMPLE;
   TITLE 'Point-biserial Correlations';
   VAR S1-S5;
   WITH RAW; ***Same results if you use PERCENT;
RUN;
```


11-3
```
DATA KAPPA;
   LENGTH RATER_1 RATER_2 $ 1;
   INPUT RATER_1 RATER_2 @@;
DATALINES;
C C  X X  X X  C X  X C  X X  X X
C X  C C  X X  C C  C C  X X  C C
;
PROC FREQ DATA=KAPPA;
   TITLE 'Inter-rater Reliability: Coefficient Kappa';
   TABLES RATER_1 * RATER_2 / AGREE NOCUM NOPERCENT;
RUN;
```

CHAPTER 12

12-1 (a)
```
DATA PROB12_1;
   INPUT GROUP $ SCORE;
DATALINES;
P       77
P       76
P       74
P       72
P       78
D       80
D       84
D       88
D       87
D       90
;
```

(b)
```
DATA PROB10_1;
   INPUT GROUP $ SCORE @@;
DATALINES;
P 77 P 76 P 74 P 72 P 78
D 80 D 84 D 88 D 87 D 90
;
```

(c)
```
DATA PROB10_1;
   DO GROUP= 'P','D';
      DO I = 1 TO 5;
         INPUT SCORE @@;
         OUTPUT;
      END;
   END;
   DROP I;
DATALIMES;
77 76 74 72 78
80 84 88 87 90
;
```

(d)
```
DATA PROB10_1;
   DO GROUP= 'P','D';
      DO I = 1 TO 5;
         SUBJ+1;
         INPUT SCORE @@;
         OUTPUT;
      END;
   END;
   DROP I;
DATALINES;
77 76 74 72 78
80 84 88 87 90
;
```

12-2
```
DATA PROB12_2;
   INFILE DATALINES DLM=',';
   INPUT X1-X4;
DATALINES;
1,3,5,7
2,4,6,8
9,8,7,6
;
```

12-3
```
DATA PROB12_3;
   INFILE DATALINES DSD;
   INPUT X Y C $ Z;
DATALINES;
1,,"HELLO",7
2,4,TEXT,8
9,,,6
;
```

12-4
```
DATA OFFICE;
   INFORMAT VISIT MMDDYY8. DX $10. COST DOLLAR8.;
   INFILE DATALINES MISSOVER;
   INPUT ID VISIT DX COST;
DATALINES;
1 10/01/96 V075 $102.45
2     02/05/97    X123456789  $3,123
3  07/07/96    V4568
4     11/11/96    A123      $777.
;
```

12-5
```
DATA PROB12_5;
   INPUT  SUBJECT $ 1-3
          A1      $ 5
          X         7-8
          Y         9-10
          Z         11-12;
DATALINES;
A12 X 111213
A13 W 102030
;
```

12-6
```
DATA PROB12_6;
   INPUT @1  SUBJECT  $3.
         @5  A1       $1.
         @7  (X Y Z)  (2.);  /* OK to specify X, Y, and Z */
DATALINES;                   /* separately                */
A12 X 111213
A13 W 102030
;
```

12-7
```
DATA PROB12_7;
   INPUT @1  ID    3.
         @4  GENDER  $1.
         @10 (DOB VISIT DISCHRG) (MMDDYY6.)
         @30 (SBP1-SBP3) (3. + 5)
         @33 (DBP1-DBP3) (3. + 5)
         @36 (HR1-HR3) (2. + 6);
   FORMAT DOB VISIT DISCHRG MMDDYY8.;
DATALINES;
123M      102146111196111396  130 8668134 8872136 8870
456F      010150122596020597  220110822101028424012084
;
```


12-8
```
DATA PROB12_8;
   INPUT #1  @1   ID  2.
             @4   X   2.
             @6   Y   2.
         #2  @3   A1  $3.
             @6   A2  $1.;
DATALINES;
01 2345
  AAAX
02 9876
  BBBY
;
```

12-9
```
DATA PROB12_9;
   INPUT X Y @@;
DATALINES;
1 2  3 4  5 6  7 8
11 12   13 14
21 22  23 24  25 26  27 28
;
```

12-10
```
DATA SURVEY;
   INPUT @12 TEST 1. @;
   IF TEST = 1 THEN
   INPUT @1  ID     $3.
         @4  HEIGHT 2.
         @6  WEIGHT 3.;
   ELSE IF TEST = 2 THEN
   INPUT @1  ID     $3.
         @4  AGE     2.
         @6  HEIGHT 2.
         @8  WEIGHT 3.;
   DROP TEST;
DATALINES;
00168155    1
00272201    1
0034570170 2
0045562 90 2
;
```

CHAPTER 13

13-1
```
LIBNAME A 'A:\';
DATA A.BILBO;
   INFILE 'A:FRODO' PAD;  *(Don't forget the PAD);
   INPUT ID 1-3 AGE 5-6 HR 8-10 SBP 12-14 DBP 16-18;
   AVEBP = 2*DBP/3 + SBP/3;
RUN;

DATA A.HIBP;
   SET A.BILBO;
   IF AVEBP GE 100;
RUN;
```

Alternative solutions using a WHERE statement or WHERE data set option:

```
DATA A.HIBP;
   SET A.BILBO;
   WHERE AVEBP GE 100;
RUN;
```

or

```
DATA A.HIBP;
   SET A.BILBO(WHERE= (AVEBP GE 100));
RUN;
```

13-2
```
LIBNAME INDATA 'C:\SASDATA';
OPTIONS FMTSEARCH= (INDATA);
***Alternative is to use the default library name LIBRARY;
PROC FREQ DATA=INDATA.SURVEY ORDER=FREQ;
   TITLE 'Frequencies for ICD_9 codes from the 1990 Survey';
   TABLES ICD_9;
RUN;

PROC MEANS DATA=INDATA.SURVEY N MEAN
           STD STDERR MIN MAX MAXDEC=2;
   TITLE 'Descriptive Statistics for the Survey';
   VAR AGE;
RUN;
```

13-3
```
LIBNAME C 'C:\MYDATA;
DATA C.DEM_9697;
   IF END96 NE 1 THEN INFILE 'A:DEM_1996' END = END96;
   ELSE INFILE 'A:DEM_1997';
   INPUT @1  ID       $3.
         @4  AGE       2.
         @6  JOB_CODE $1.
         @7  SALARY    6.;
RUN;
```

13-4
```
DATA PROB13_4;
   INFILE 'B:SAMPLE.DTA' LRECL=320 MISSOVER;
   INPUT X1-X100;
RUN;
```

13-5
```
***DATA step to create MILTON;
DATA MILTON;
   INPUT X Y A B C Z;
DATALINES;
1 2 3 4 5 6
11 22 33 44 55 66
;
DATA _NULL_; ***No need to create a SAS data set;
   SET MILTON;
   FILE 'C:\MYDATA\OUTDATA';
   PUT @1 (A B C) (3.);
RUN;
```

CHAPTER 14

14-1 (a)
```
DATA GYM;
   LENGTH GENDER $ 1;
   INPUT ID GENDER AGE VAULT FLOOR P_BAR;
   ***GENDER is already declared a character variable by
      the LENGTH statement so a $ is not needed in the INPUT
      statement;
DATALINES;
3   M   8   7.5   7.2   6.5
5   F   14  7.9   8.2   6.8
2   F   10  5.6   5.7   5.8
7   M   9   5.4   5.9   6.1
6   F   15  8.2   8.2   7.9
;
```


(b)
```
DATA MALE_GYM;
   SET GYM;
   IF GENDER = 'M';
RUN;
```

or

```
DATA MALE_GYM;
   SET GYM(WHERE= (GENDER= 'M'));
RUN;
```

(c)
```
DATA OLDER_F;
   SET GYM;
   IF GENDER = 'F' AND AGE GE 10; ***WHERE statement OK;
RUN;
```

or

```
DATA OLDER_F;
   SET GYM(WHERE= (GENDER = 'F' AND AGE GE 10));
RUN;
```

14-2
```
DATA YEAR1996;
   INPUT ID HEIGHT WEIGHT;
DATALINES;
2 68 155
1 63 102
4 61 111
;
DATA YEAR1997;
   INPUT ID HEIGHT WEIGHT;
DATALINES;
7 72 202
5 78 220
3 66 105
;
DATA BOTH;
   SET YEAR1996 YEAR1997;
RUN;
```

14-3
```
DATA MONEY;
   INPUT ID INCOME : $1. L_NAME : $10.;
DATALINES;
3 A Klein
7 B Cesar
8 A Solanchick
1 B Warlock
5 A Cassidy
2 B Volick
;
PROC SORT DATA=GYM;
   BY ID;
RUN;

PROC SORT DATA=MONEY;
   BY ID;
RUN;

DATA GYMMONEY;
   MERGE GYM(IN=IN_GYM) MONEY;
   BY ID;
   IF IN_GYM;
RUN;
```

```
PROC PRINT DATA=GYMMONEY;
   ***Note: GYMMONEY already in ID order;
   TITLE 'Listing of Gym and Financial Data';
   ID ID;
   VAR L_NAME GENDER AGE;
RUN;
```

14-4
```
PROC SORT DATA=BOTH;
   BY ID;
RUN;

DATA FREDDY;
   MERGE GYMMONEY (IN=ONE)
         BOTH (IN=TWO);
   BY ID;
   IF ONE AND TWO;
RUN;

PROC PRINT DATA=FREDDY NOOBS;
   TITLE 'Listing of Data Set FREDDY';
RUN;
```

14-5
```
DATA FINANCE;
   LENGTH INCOME GENDER PLAN $ 1;
   INPUT INCOME GENDER PLAN @@;
DATALINES;
A N W A F X B M Y B F Z
;
PROC SORT DATA=FINANCE;
   BY GENDER INCOME;
RUN;

PROC SORT DATA=GYMMONEY;
   BY GENDER INCOME;
RUN;

DATA FINAL;
   MERGE FINANCE GYMMONEY;
   BY GENDER INCOME;
RUN;

PROC PRINT DATA=FINAL NOOBS;
   TITLE 'Listing of Data Set FINAL';
RUN;
```

14-6
```
DATA NEW;
   INFILE DATALINES MISSOVER;
   ***MISSOVER needed because of short lines;
   INPUT ID GENDER : $1. AGE VAULT P_BAR;
DATALINES;
3 . . . 6.7
5 . 15 8.1 7.2
7 F
;
PROC SORT DATA=NEW;
   BY ID;
RUN;

PROC SORT DATA=GYM;
   BY ID;
RUN;
```



```
DATA GYM_2;
   UPDATE GYM NEW;
   BY ID;
RUN;

PROC PRINT DATA=GYM_2 NOOBS;
   TITLE 'Listing of Data Set GYM_2';
RUN;
```

An alternative way to create the update (NEW) data set is:

```
DATA NEW;
   LENGTH GENDER $ 1;
   INPUT ID= GENDER = $ AGE= VAULT= P_BAR=;
DATALINES;
ID=3 P_BAR=6.7
ID=5 AGE=15 VAULT=8.1 P_BAR=7.2
ID=7 GENDER=F
;
```

This is called NAMED input and is discussed in the SAS Language Reference, version 6, First Edition.

CHAPTER 15

15-1
```
DATA PROB15_1;
   INPUT (HT1-HT5)(2.) (WT1-WT5)(3.);
   ARRAY HT[*] HT1-HT5;
   ARRAY WT[*] WT1-WT5;
   ARRAY DENS[*] DENS1-DENS5;
   DO I = 1 TO 5;
      DENS[I] = WT[I]/ HT[I]**2;
   END;
   DROP I;
DATALINES;
6862727074150090208230240
64 68 70140   150   170
;
```

15-2
```
DATA OLDMISS;
   INPUT A B C X1-X3 Y1-Y3;
   ARRAY NINE[*] A B C X1-X3;
   ARRAY SEVEN[*] Y1-Y3;
   DO I = 1 TO 6;
      IF NINE[I] = 999 THEN NINE[I] = .;
   END;
   DO I = 1 TO 3;
      IF SEVEN[I] = 777 THEN SEVEN[I] = .;
   END;
   DROP I;
DATALINES;
1 2 3 4 5 6 7 8 9
999 4 999 999 5 999 777 7 7
;
```

Alternative:

```
DATA OLDMISS;
   INPUT A B C X1-X3 Y1-Y3;
   ARRAY NINE[*] A B C X1-X3;
   ARRAY SEVEN[*] Y1-Y3;
   DO I = 1 TO 6;
      IF NINE[I] = 999 THEN NINE[I] = .;
```

```
      IF I LE 3 AND SEVEN[I] = 777 THEN SEVEN[I] = .;
   END;
   DROP I;
DATALINES;
1 2 3 4 5 6 7 8 9
999 4 999 999 5 999 777 7 7
;
```

15-3
```
DATA SPEED;
   INPUT X1-X5 Y1-Y3;
DATALINES;
1 2 3 4 5 6 7 8
11 22 33 44 55 66 77 88
;
DATA SPEED2;
   SET SPEED;
   ARRAY X[5] X1-X5;
   ARRAY Y[3] Y1-Y3;
   ARRAY LX[5] LX1-LX5;
   ARRAY SY[3] SY1-SY3;
   DO I = 1 TO 5;
      LX[I] = LOG(X[I]);
      IF I LE 3 THEN SY[I] = SQRT(Y[I]);
   END;
   DROP I;
RUN;
```

15-4
```
DATA PROB15_4;
   LENGTH C1-C5 $ 2;
   INPUT C1-C5 $ X1-X5 Y1-Y5;
   ARRAY C[5] $ C1-C5;
   ARRAY X[5] X1-X5;
   ARRAY Y[5] Y1-Y5;
   DO I = 1 TO 5;
      IF C[I] = 'NA' THEN C[I] = ' ';
      IF X[I] = 999 OR Y[I] = 999 THEN DO;
      X[I] = .; Y[I] = .;
      END;
   END;
   DROP I;
DATALINES;
AA BB CC DD EE 1 2 3 4 5 6 7 8 9 10
NA XX NA YY NA 999 2 3 4 999 999 4 5 6 7
;
```

CHAPTER 16

16-1
```
DATA FROG;
   INPUT ID X1-X5 Y1-Y5;
DATALINES;
01  4  5  4  7  3  1  7  3  6  8
02  8  7  8  6  7  5  4  3  5  6
;
DATA TOAD;
   SET FROG;
   ARRAY XX[5] X1-X5;
   ARRAY YY[5] Y1-Y5;
   DO TIME = 1 TO 5;
      X = XX[TIME];
      Y = YY[TIME];
```



```
      OUTPUT;
   END;
   DROP X1-X5 Y1-Y5;
RUN;
```

16-2
```
DATA STATE;
   INFORMAT STATE1-STATE5 $2.;
   INPUT ID STATE1-STATE5;
DATALINES;
1  NY   NJ   PA   TX   GA
2  NJ   NY   CA   XX   XX
3  PA   XX   XX   XX   XX
;
DATA NEWSTATE;
   SET STATE;
   ARRAY XSTATE[*] $ STATE1-STATE5;
   DO I = 1 TO 5;
      IF XSTATE[I] = 'XX' THEN XSTATE[I] = ' ';
      STATE = XSTATE[I];
      OUTPUT;
   END;
   DROP I;
RUN;

PROC FREQ DATA=NEWSTATE ORDER=FREQ;
   TABLES STATE;
RUN;
```

16-3
```
DATA NEW;
   SET BLAH;
   ARRAY JUNK[*] X1-X5 Y1-Y5 Z1-Z5;
   DO J = 1 TO DIM(JUNK);
      IF JUNK[J] = 999 THEN JUNK[J] = .;
   END;
   DROP J;
RUN;
```

CHAPTER 17

17-1
```
DATA HOSP;
   INFORMAT ID $3. GENDER $1. DOB DOS MMDDYY8.;
   INPUT ID GENDER DOB DOS LOS SBP DBP HP;
   FORMAT DOB DOS MMDDYY10.;
DATALINES;
1 M 10/21/46 3/17/97 3 130 90 68
2 F 11/1/55 3/1/97 5 120 70 72
3 N 6/6/90 1/1/97 100 102 64 88
4 F 12/21/20 2/12/97 10 180 110 86
;
DATA NEW_HOSP;
   SET HOSP;
   LOG_LOS = LOG10(LOS); ***Part A;
   AGE_LAST = INT((DOS - DOB)/365.25); ***Part B;
   X = ROUND(SQRT(MEAN(OF SBP DBP)),.1); ***Part C;
RUN;
```

17-2
```
DATA MANY;
   INPUT X1-X5 Y1-Y5;
***Part A;
MEAN_X = MEAN(OF X1-X5);
MEAN_Y = MEAN(OF Y1-Y5);
```

```
***Part B;
MIN_X = MIN(OF X1-X5);
MIN_Y = MIN(OF Y1-Y5);
***Part C;
CRAZY = MAX(OF X1-X5) * MIN_Y * (N(OF X1-X5) + NMISS(OF Y1-Y5));
***Part D;
IF N(OF X1-X5) GE 3 AND N(OF Y1-Y5) GE 4 THEN
   MEAN_X_Y = MEAN(OF X1-X5 Y1-Y5);
DATALINES;
1 2 3 4 5   6 7 8 9 10
3 . 5 . 7   5 . . . 15
9 8 . . .   4 4 4 4 1
;
```

17-3
```
DATA UNI;
   DO I = 1 TO 1000;
      N = INT(RANUNI(0)*5 + 1);
      OUTPUT;
   END;
   DROP I;
RUN;

PROC FREQ DATA=UNI;
   TABLES N / NOCUM;
RUN;
```

17-4
```
PROC FORMAT;
   VALUE DAYFMT 1='SUN' 2='MON' 3='TUE' 4='WED' 5='THU'
                6='FRI' 7='SAT';
RUN;

DATA DATES;
   SET HOSP; ***From 17-1;
   DAY = WEEKDAY(DOS);
   MONTH = MONTH(DOS);
   FORMAT DAY DAYFMT.;
RUN;

PROC CHART DATA=DATES;
   VBAR DAY MONTH / DISCRETE;
RUN;
```

17-5
```
DATA MIXED;
   INPUT X Y A $ B $;
DATALINES;
1 2 3 4
5 6 7 8
;
DATA NUMS;
   SET MIXED;
   A_NUM = INPUT(A,8.);
   B_NUM = INPUT(B,8.);
   DROP A B; ***Don't forget this;
RUN;
```

17-6
```
PROC FORMAT;
   VALUE AGEGRP LOW-< 20 = '1'
                20-40    = '2' /*OK SINCE INTEGERS*/
                41-HIGH  = '3';
RUN;

DATA NEWER;
   SET NEW_HOSP; ***From 17-1;
   AGEGROUP = PUT(AGE_LAST,AGEGRP.);
RUN;
```


CHAPTER 18

18-1
```
DATA CHAR1;
   INPUT STRING1 $1.
         STRING2 $5.
         STRING3 $8.
         (C1-C5)($1.);
DATALINES;
XABCDE12345678YNYNY
YBBBBB12V56876yn YY
ZCCKCC123-/. ,WYNYN
;
DATA ERROR;
   SET CHAR1;
   LENGTH DUMMY $ 5;
   DUMMY = C1 || C2 || C3 || C4 || C5;
   IF VERIFY(STRING1,'XYZ') NE 0   OR
      VERIFY(STRING2,'ABCDE') NE 0  OR
      VERIFY(UPCASE(DUMMY),'NY') NE 0 THEN OUTPUT;
DROP DUMMY;
RUN;
```

18-2
```
DATA PROB18_2;
   SET CHAR1; ***From 18-1;
   NEW3 = TRANSLATE(
         COMPRESS(STRING3,' -/.,'),'ABCDEFGH','12345678');
   IF VERIFY(NEW3,'ABCDEFGH ') NE 0 THEN NEW3 = ' ';
RUN;
```

18-3
```
DATA PROB18_3;
   SET CHAR1; ***From 18-1;
   ARRAY C[5] $ 1 C1-C5; ***Create a character array;
   DO I = 1 TO 5;
      C[I] = UPCASE(C[I]);
      IF VERIFY(C[I],'NY ') NE 0 THEN C[I] = ' ';
   END;
   DROP I;
RUN;
```

18-4
```
DATA PHONE;
   INPUT CHAR_NUM $20.;
   NUMBER = INPUT( COMPRESS(CHAR_NUM,' ()-/'),10.);
DATALINES;
(908)235-4490
(800) 555 - 1 2 1 2
203/222-4444
;
```

18-5
```
DATA EXPER;
   INPUT ID    $ 1-5
         GROUP $ 7
         DOSE  $ 9-12;

   LENGTH SUB_ID $ 2 GRP_DOSE $ 6;
   SUB_ID = SUBSTR(ID,2,2);
   GRP_DOSE = GROUP  | | '-' | |  DOSE;

DATALINES;
1NY23 A HIGH
3NJ99 B HIGH
2NY89 A LOW
5NJ23 B LOW
;
```

18-6
```
DATA PROB18_6;
   SET EXPER; ***From the previous problem;
   LENGTH ID2 $ 6;
   ID2 = ID;
   IF INPUT(SUBSTR(ID,4,1),1.) GE 5 THEN SUBSTR(ID2,6,1) = '*';
RUN;
```

18-7
```
DATA ONE;
   INPUT @1  GENDER  $1.
         @2  DOB     MMDDYY8.
         @10 NAME    $11.
         @21 STATUS  $1.;
   FORMAT DOB MMDDYY8.;
DATALINES;
M10/21/46CADY       A
F11/11/50CLINE      B
M11/11/52SMITH      A
F10/10/80OPPENHEIMERB
M04/04/60JOSE       A
;
DATA TWO;
   INPUT @1  GENDER  $1.
         @2  DOB     MMDDYY8.
         @10 NAME    $11.
         @21 WEIGHT  3.;
   FORMAT DOB MMDDYY8.;
DATALINES;
M10/21/46CODY       160
F11/11/50CLEIN      102
F11/11/52SMITH      101
F10/10/80OPPENHAIMER120
M02/07/60JOSA       220
;
DATA ONE_TMP;
   SET ONE;
   S_NAME = SOUNDEX(NAME);
RUN;

DATA TWO_TMP;
   SET TWO;
   S_NAME = SOUNDEX(NAME);
RUN;

PROC SORT DATA=ONE_TMP;
   BY GENDER DOB S_NAME;
RUN;

PROC SORT DATA=TWO_TMP;
   BY GENDER DOB S_NAME;
RUN;

DATA COMBINED;
   MERGE ONE_TMP(IN=INONE) TWO_TMP(IN=INTWO);
   BY GENDER DOB S_NAME;
   IF INONE AND INTWO;
RUN;
```

NOTE: There are no problems for chapters 19 and 20.

INDEX