Getting Started with SAS

SAS stands for ***Statistical Analysis System***. SAS is a computer software system for data analysis and presentation. SAS can be used to produce:

- Descriptive statistics
- Statistical analyses
- Printed reports
- Graphs

1. To start the SAS system:

If starting SAS under Windows,

- Click the Start button in the lower left corner of the screen.
- From the Start menu choose Programs and slide the mouse to the right.
- From the Programs menu choose SAS.
- From the SAS menu choose the SAS system. (You will see an upside down pyramid logo next to this item.)
- When SAS has been properly invoked, you will see the screen divided into two halves, with the Log window on top and the Program Editor window on bottom.

If starting SAS from a file server,

- Logon to the file server.
- Issue the command to start SAS (usually SAS).
- OR see you systems administrator for the appropriate command sequence to start SAS.

Below the Windows title bar you will see a list of commands (FILE, EDIT, VIEW, LOCALS, GLOBALS, OPTIONS, WINDOW, AND HELP) and a row of buttons. Buttons are shortcuts to frequently used commands such as submit, cut, copy, paste, open, close, undo, and help. To determine the function of each button, hold the mouse pointer over a button (without

clicking). A small text balloon will appear to remind you what the button is for.

2. To create a new SAS program:

- Position the mouse pointer in the Program Editor window (the bottom window on the screen) and click. This makes the Program Editor window the active window.
- Type the SAS program into the Program Editor window. (Don't forget the semicolons after each SAS statement.)
- You can use typical Windows editing actions such as cut, copy, paste.
- You can click on the middle button in the right hand corner of the Program Editor title bar to zoom the window to fill the entire screen or to reduce the window back to a portion of the screen.

3. To save a SAS program:

- Select SAVE AS from the FILE menu. A popup box will appear.
- Specify the location (disk/subdirectory) where the program file will be stored.
- Choose FILENAME and type in the filename.
- Choose FILETYPE and verify that the filetype is *.sas
- Click on the word SAVE in the popup box.

Note that it is wise to save your work on a regular basis (every 10-15 minutes), even if you have not yet finished typing your SAS program.

After the program has been saved once, you can save updated versions (under the same filename) by selecting SAVE from the FILE menu. SAS will ask if you want to replace the old file, append to the old file, or cancel the save operation. Click on the appropriate button (usually the replace button).

4. To run a SAS program:

1. Make sure that the SAS program you wish to execute appears in the Program Editor window.
2. Choose SUBMIT from the LOCALS menu (or click the button that looks like a running man).

Note that the program you submitted disappears from the Program Editor window, upon submission.

5. To view the results produced by the SAS program:

1. Click on WINDOW from the command menu.
2. Choose OUTPUT.

The OUTPUT window will appear on top of the Log and Program Editor window. You can use the horizontal and vertical scroll bars, as well as the arrow keys, and the PageUp and PageDown keys to review the results produced by SAS.

6. To save the results in the Output window:

- Make the OUTPUT window the active window.
- Select SAVE AS from the FILE menu. A popup box will appear.
- Specify the location (disk/subdirectory) where the file containing the results will be stored.
- Choose FILENAME and type in the filename.
- Choose FILETYPE and verify that the filetype is *.lst
- Click on the word SAVE in the popup box.

7. To print the results:

- Make the OUTPUT window the active window.
- Select the PRINT option from the FILE menu (or click on the Printer button).

8. To return to the Program Editor window:

- Click on the WINDOW command.
- Click on PROGRAM EDITOR.

9. To recover a closed window:

If you have clicked on the X button in the upper right corner of a window, the window disappears. You can recover the closed window as follows:

- Click on the GLOBALS command.
- Click the name (Program Editor, Log, Output) of the window you wish to recover.

10. To recall the text of the program you submitted:

Remember, the program submitted disappears from the Program Editor window upon submission. To recall the SAS program without opening the file that contains the SAS program ,

- Make the Program Editor window the active window.
- Click on LOCALS.
- Click on RECALL TEXT.

11. To edit your SAS program:

Recall the text of the SAS program into the Program Editor window. The Program Editor allows editing in a fashion similar to Notepad, Eudora, or a stripped down version of WordPerfect.

- **To insert a blank line**, position the cursor at the end of a line and hit ENTER.

- **To insert characters in a line**, position the cursor to the desired location, and enter the characters you wish to insert. By default the

editor is in INSERT mode. To toggle between insert mode and typeover mode, hit the INSERT key.

- **To delete characters**,

  1. Position the cursor in front of the character(s) to be deleted and press the DELETE key.
  2. OR position the cursor after the character(s) to be deleted, and press the BACKSPACE key.
  3. OR Use the mouse to highlight the character(s) to be deleted, and press the DELETE key.

- **To copy text,**

  1. Use the mouse to highlight the material to be copied.
  2. Choose COPY from the EDIT menu.
  3. Position the cursor to the location where the copied material will appear, and click.
  4. Choose PASTE from the EDIT menu.

- **To move text,**

  1. Use the mouse to highlight the material to be moved.
  2. Choose CUT from the EDIT menu.
  3. Position the cursor to the location where the material will appear, and click.
  4. Choose PASTE from the EDIT menu.

3. **To undo an action,**

   1. Click on the UNDO button.
   2. OR Click on the UNDO option of the EDIT menu.

   Note that you can undo (at least) the six most recent actions that you have taken.

12. To save the modified program:

- Select SAVE from the FILE menu.

Note that the first time you do this, SAS will ask if you want to Replace or Append the file. If you want the modified version of the program to completely replace the previous version, click on REPLACE. If you want to append additional program statements to the end of the old program, choose APPEND.

13. To use the online HELP facility:

- Click on the HELP menu, then click on EXTENDED HELP.
- OR click on the ? button.

You may click on any of the green underlined hotspots to get a new screen of information (or options). You may again click on a topic or procedure name to get details and options.

You can also review and copy sample SAS programs by selecting that item from the main HELP screen. Select the procedure of interest to you. If you want to run a sample program, first copy the program, using the mouse and the COPY option of the EDIT menu. Next click in the Program Editor window. Then choose PASTE from the EDIT menu. The sample program will appear in the Program Editor window. Click SUBMIT to run the sample program.

While in HELP, you can click the BACK button to go back to a previous help screen.

Exercise 1: Invoke the SAS system on your computer.
Access each SAS window.
In the Program Editor window, type anything. Practice inserting and deleting characters.
Practice copying and moving blocks of text.
Try out the help facility.

SAS Statements

SAS programs are made up of SAS statements. SAS statements ask SAS to perform some activity. The first word of a SAS statement tells SAS what action to perform - create a SAS data set, run a statistical procedure, print a line of data, etc. In the rest of the SAS statement, you give SAS more information about how you want the activity performed - what to name the new data set, which procedure to run, how to arrange the line you want printed, etc.

A SAS statement may begin and end in any column. SAS statements may span more than one line. Two or more SAS statements may share the same line. In general, variable names, options and parameters may appear in SAS statements, and are separated by blanks (not commas). **Every SAS statement ends with a semicolon**.

Variables and data sets must have names. Allowable names are from one to 32 characters long, and must start with a letter. You can use the letters A through Z, digits 0 through 9, and the underscore character (_). Examples of allowable names include

AGE
SEX
YR_INCOM
Q1_1999

The Data Step

The first step in most SAS jobs is to get your data into a SAS data set. This is accomplished by using the DATA step. At a minimum you will need:

- DATA statement - begins the creation of a SAS data set, and allows you to name the data set.
- INPUT statement - names the variables, and allows you to specify the location of the data values.
- DATALINES statement - indicates that the data follows.

The DATA statement

The DATA statement is usually the first statement in a SAS job, although the DATA statement may appear in other places in the SAS job. It has the form

DATA datasetname;

The data set name must follow the standard naming conventions. An example is

DATA example1;

If you omit the data set name from the DATA statement, as follows

DATA;

SAS will assign the data set a name (usually work.data1, work.data2, etc.).

The INPUT Statement

The INPUT statement assigns names to the variables and describes the location of their values in the data records. Before you can write the INPUT statement you must know what the information on your data records represents. The INPUT statement is important because SAS reads the data records using the description you have provided. If this information is not correct, subsequent SAS procedures may produce incorrect results. The INPUT statement usually follows the DATA statement.

To write an INPUT statement:

1. Begin with the word INPUT.
2. Choose names for each of the variables (remember to follow the rules for naming variables).
3. Determine whether each variable contains numeric data or character data. If the variable's values contain letters or other non-numeric characters, the variable is a character variable. To indicate this to SAS, place a dollar sign ($) after the variable name in the INPUT statement.

4. For each variable, write the number of the column where the data values begin, followed by a dash (-), then the number of the column where the data values end. It is important to give the entire range of columns where the variable's values may appear, even if not all values occupy all the columns specified.
5. If a numeric variable contains decimal positions, it is easiest to include the decimal point in the values on the data records. However, SAS can be told to insert a decimal point in the appropriate position. Specify the number of digits that should follow the decimal point after the column specifications.

As an example, suppose that columns 1-10 contain the name of a person, column 12 contains the person's sex (M or F), columns 14-15 contains the age, columns 17-18 contains the height, and columns 20-22 contains the weight. Then our INPUT statement would look like

```
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22;
```

If the variable weight has values to tenths of a pound, and the decimal point is omitted from the data records, our INPUT statement would look like

```
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-23  1;
```

This type of INPUT statement uses fixed column input, i.e., the values of the variables must appear in the columns specified in the INPUT statement. There is an easier way to write the INPUT statement, called free-format input (or list input). You may omit the column specifications if your data satisfies the following conditions:

- Each of the values on the data records is separated from the next value by at least one blank column.
- Character variables have values consisting of 8 characters or fewer.
- The data have no missing values, or the missing values are represented by periods.
- Numeric values include all necessary decimal points.

Suppose that our data meets these conditions. Then the INPUT statement could have been written as

INPUT name $ sex $ age height weight ;

There may be some **special situations** that must be addressed.

- **Skipping data values**. If the data record contains values of variables that you don't want SAS to read, don't describe these variables in the INPUT statement. For example, suppose that you only need two of the variables, name and weight. You could use the INPUT statement:

  INPUT name $ 1-10 weight 20-22;

- **Long INPUT statements**. If the INPUT statement is so long that you need more than one line, just continue onto the next line. Don't split a variable name between two lines.

- **Missing data values**. Often one or more data values are missing. If the columns that contain the values of a variable are blank, or if those columns contain a period, SAS treats that value as a missing value. Each SAS procedure has provisions for handling missing values in the data. When SAS prints data values, it uses a period to indicate a missing value.

- **When data values for one observation (record) occupy more than one line**. When you write the INPUT statement for a set of data with several lines per record, you must tell SAS which variables are on which lines. The symbol # followed by a number tells SAS which line contains the next group of variables. For example the SAS statement

  INPUT a 3-5 b 45-50 #2 x 10-15;

  tells SAS to read the values of variables a and b on the first line, then go to line 2 to read the value of x. You don't need to put the symbol #1 before the first group of variables, since SAS automatically begins reading on the first line.

The DATALINES Statement

When you enter lines of data at a terminal, the DATALINES statement follows the DATA statement and the INPUT statement. The data lines follow the DATALINES statement. The end of the data records should be indicated by a data line containing only a semicolon. For example,

```
DATA example1;
INPUT name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22;
DATALINES;
data lines go here
;
```

The INFILE statement

When the data records are stored on disk (or tape) you must tell both the computer's operating system and SAS where to find the data. This is done by using an INFILE statement in you SAS program. The form of the INFILE statement is

```
INFILE 'path\filename' ;
```

For example, suppose that the data set is located on your C: drive in the directory sasuser. The INFILE statement would look like

```
INFILE 'C:\sasuser\mydata';
```

The INFILE statement is placed before the INPUT statement that describes the data. For example,

```
DATA example1;
INFILE 'C:\sasuser\mydata';
INPUT name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22 ;
SAS programming statements go here
PROC PRINT;
RUN;
```

Getting Your Data Into Shape

Data are often not in the shape you desire when you get them. You can use SAS program statements to work with the observations as they are read into SAS -- for example, to create new variables, to delete observations, to modify the existing values of variables, etc. Program statements are optional in SAS jobs. You are not required to include them. But as you handle sophisticated data analysis tasks, you will come to rely on program statements to help you tailor your SAS programs to the requirements of the task at hand. SAS program statements go after the INPUT statement and before the DATALINES statement (or before the first PROC statement, if you are using an INFILE statement). For example,

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
PROC PRINT;
RUN;
```

In order to understand how to use program statements, you first need to know how SAS creates a data set. When SAS creates a data set, it goes through the following steps for each observation in the data:

1. SAS uses the description in the INPUT statement to read each observation.
2. SAS use the data values in the observation to carry out any program statements that are present.
3. SAS adds the observation to the data set being created.

The SAS program statements are executed once for each observation. When the next observation is read, the program statements are executed again for the current observation.

Creating New Variables

When you create a new variable, you add another set of data values to your data set. You can create a new variable by using a program statement called an assignment statement. To create a new variable:

1. Choose a name for the new variable.
2. Figure out a formula necessary to create the new variable.
3. Write the formula as an assignment statement, putting the new variable's name on the left of the equal sign.

As an example, suppose that you wish to create a new variable containing the weight in kilograms.

1. The name of the new variable will be wtkg.
2. In order to convert weight in pounds to weight in kilograms, you need to know that one pound is .45 kilogram. The formula you need to use is

   weight in kilograms = weight in pounds * .45.
3. In the SAS assignment statement, put the name of the new variable on the left of the equal sign.

   wtkg = weight * .45 ;

The assignment statement tells SAS, for each observation, to multiply the value of the variable weight by .45, and make the result the value of the variable wtkg for each observation.

Modifying Values of Existing Variables

You can also use program statements to modify the values of a variable that already exists in your data set. Suppose that you want the value of height in feet rather than in inches, but you don't want to create a new variable. This can be accomplished by dividing the value of the variable height by 12, and assigning the result to the existing variable height. The program statement to do this is

```
height  = height / 12;
```

This statement looks as if you are saying that height equals height divided by 12 -- a clear absurdity. But in an assignment statement, the equals sign means "assign the computed value on the right of the equals sign to the variable on the left of the equals sign".

Allowable arithmetic symbols in assignment statements include:

+	addition
-	subtraction
*	multiplication
/	division
**	exponentiation
( )	grouping symbols

Exercise 2: If necessary, clear the text from your Program Editor window. (Do this by clicking on EDIT and then click on CLEAR TEXT.) Type in a SAS program, similar to the examples above, that reads (at least) 10 observations consisting of (at least) 4 variables. Create a new variable. Modify the value of an existing variable. Use PROC PRINT to examine your results. Save your SAS program to disk.

IF Statements and ELSE Statements

Sometimes you may want SAS to execute an action for certain observations in the data set, but not for all observations. This can be accomplished by using an IF statement. The IF statement has the form:

```
IF condition THEN statement;
```

As SAS reads each observation, it evaluates the condition to determine if the condition is true or false for the current observation. If the condition is true SAS executes the statement following the word THEN. An ELSE statement may be used in conjunction with an IF statement:

```
IF condition THEN statement1;
ELSE statement2;
```

As SAS reads each observation, it evaluates the condition to determine if the condition is true or false for the current observation. If the condition is true SAS executes the statement following the word THEN. If the condition is false, SAS executes the statement following the word ELSE.

The IF condition may be a simple comparison of a variable and a value:

```
IF height  >  72  THEN  tall = "yes";
```

The IF condition may be a comparison of two variables:

```
IF pretest < posttest THEN  improve='yes';
```

The IF condition may involve several comparisons joined by AND and OR operators:

```
IF age < 10  AND  height > 72 THEN LIST;
```

There may be occasions when you want to execute more than one program statement as a result of a logical condition. This can be accomplished by using a DO…END structure. The general form is :

```
IF  condition  THEN  DO;
            Program statements to be executed when the
            condition is evaluated true.
            END;
ELSE   DO;
            Program statements to be executed when the
            condition is evaluated false.
            END;
```

Allowable comparison operators that can be used in IF conditions include:

Symbol	Abbreviation	Comparison
<	LT	less than
>	GT	greater than
<=	LE	less than or equal to
>=	GE	greater than or equal to

=	EQ	equal to
~=	NE	not equal to

The Delete Statement

When you want to discard certain observations (you may not need them for the current analysis, or they may contain invalid data), you can use the DELETE Statement:

```
DELETE;
```

When this statement is executed, SAS stops working on the current observation, does not add it to the SAS data set being created, and begins immediately on the next observation. The DELETE statement usually appears as part of an IF statement. For example,

```
IF sex = 'M' THEN DELETE;
```

When an observation meets the condition, SAS carries out the DELETE statement. When the condition is not satisfied, SAS does not delete the current observation.

The DELETE statement should not be used alone. If it is used alone, all observations will be deleted from the data set being created.

The List Statement

If you want SAS to print the data records on the SAS log, you can use a LIST statement:

```
LIST ;
```

This will cause all records to be printed in the SAS log. Frequently, you will only want to print the records which contain invalid or questionable data.You can use the LIST statement with an IF statement:

```
IF age < 0 THEN LIST;
```

Subsetting IF Statements

Sometimes you want only certain observations included in the data set being created. You can use a subsetting IF statement to include only those observations that satisfy a specified condition:

```
IF  sex = 'F' ;
```

The statement means "If the value of the variable sex for the current observation is 'F' add that observation to the data set being created. Otherwise, stop carrying out program statements for the current observation, and do not add it to the data set."

The subsetting IF statement has the opposite effect of an IF statement containing the DELETE statement. As an example, the following two statements have exactly the same effect:

```
IF  sex = 'F' ;
```

and

```
IF   sex   NE  'F'   THEN   DELETE;
```

Concatenating Data Sets

Occasions may arise where similar data (i.e., the same variables on different observations) are stored in two or more files. You may wish to use all the data for a statistical analysis. To do this, create two SAS data sets using the DATA statement, INPUT statement, and INFILE statement. Then use the SET statement to create a third data set that concatenates the first two data sets. For example,

```
DATA  ex1;
INFILE 'C:\sasuser\mydata1';
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
RUN;

DATA  ex2;
INFILE 'C:\sasuser\mydata2';
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
```

```
RUN;

DATA both;
SET  ex1  ex2;
RUN;
```

The data set named both will contain all observations from the data sets ex1 and ex2.

Exercise 3: Create two data sets with the same variables. Concatenate the two data sets. Print the result.

Merging Data Sets

Occasions may arise where distinct data (i.e., different variables on similar observations) are stored in two or more files. You may combine or merge the data for a statistical analysis. To do this, create two SAS data sets using the DATA statement, INPUT statement, and INFILE statement. Next sort each data set on the same variable (using PROC SORT). Then use the MERGE statement to create a third data set that combines the first two data sets. For example,

```
DATA  ex1;
INFILE 'C:\sasuser\mydata1';
INPUT  name $ 1-10 sex $ 12 age 14-15;
RUN;

DATA  ex2;
INFILE 'C:\sasuser\mydata2';
INPUT  name $ 1-10 height 12-13 weight 15-17  ;
RUN;
PROC SORT DATA=ex1; BY name;
RUN;

PROC SORT DATA=ex2; BY name;
RUN;
```

```
DATA both;
MERGE  ex1  ex2;  BY name;
RUN;
```

In this example, an observation from data set ex1 will be matched with the observation from data set ex2 with the same value of the variable name. The data set both will consist of records that have values for the variables name, sex, age, height, and weight, matched up according to the value of the variable name. This is called a "matched merge ".

Note that in order to perform a "matched merge" each data set to be merged must contain a "common" variable, i.e. the variable that SAS uses to match records from different data sets must appear in each data set.

Exercise 4: Create two data sets with different variables. Each data set must contain a "matching" variable. Merge the two data sets (don't forget to sort them first). Print the result.

Comment Statements

Comment statements may be used to document the SAS program. The COMMENT statement has the form

```
COMMENT   text  ;
```

The text of the comment follows the word COMMENT. The text of the COMMENT statement is terminated when SAS encounters the next semicolon.

A "better" way to specify comments in a SAS program is to imbed the comment between the comment identifiers /* and */ as follows:

```
/*  text of comment  */
```

This method allows the use of semicolons within the text of the comment.

Common Program Statement Errors

- Omitting the semicolon at the end of a program statement
- Misspelling/mistyping a variable name, data set name, SAS keyword or option
- Using an invalid variable name or data set name
- Forgetting a closing quote mark
- Forgetting the END; statement (used with DO)
- Missing the */ when writing a comment

Common Data Errors

- Using the wrong INPUT format
- Specifying the wrong columns in an INPUT statement
- Using an alphabetic O instead of a numeric 0 in numeric data
- Failing to use a period (.) to represent missing data when a free-format INPUT statement is used
- Using .. for missing data
- Including a blank line in the data stream
- Placing a semicolon in the data stream

The Procedure Step

Once you have created a SAS data set, you are ready to use SAS procedures to analyze and process that data set. SAS procedures are computer programs that read your SAS data set, perform various computations, and print the results of the computations. SAS procedures can also create SAS data sets containing the results of the computations.

The statements that ask SAS to run a procedure make up the PROC step of a SAS job. The DATA step and the PROC step get their names from the SAS statements that begin these steps. You can use the SAS system by stringing together DATA steps and PROC steps. DATA steps and PROC steps don't have to appear in any special order. You may start with a DATA step, then sue a PROC step, then another PROC step, then a DATA step, and so on. However, the DATA step is usually the first step of most SAS jobs.

The PROC step always begins with a PROC statement that specifies the name of the procedure that you want to run. The general form of a PROC statement is

PROC procedurename options ;

For example, if you want to print a data set, you would begin the PRINT procedure with the statement

PROC PRINT;

All SAS procedures for processing data are similar in the respect that you can tell the procedure

- What data set you want processed
- Which variables you want processed
- Whether you want the data processed in subsets

Most SAS procedures automatically handle the most common processing situation, where

- You want to process the most recently created SAS data set

- You want all the variables processed (or all numeric variables, for a computational procedure)
- You want the entire data set processed together, rather than in subsets

Since SAS handles this situation automatically, much of the time your PROC step need only include a PROC statement giving the name of the procedure you want to run.

Telling the PROC Which Data Set to Process

If you want the procedure to process the last data set you created, or if you have created only one data set, you need give only the procedure name in the PROC statement. For example,

```
PROC PRINT ;
```

If you want the procedure to use a data set that is not the most recently created data set, or if you have created several data sets and you want to minimize confusion, you can specify the name of the data set to be processed by using the option DATA= in the PROC statement, as follows:

```
PROC procedurename DATA=datasetname ;
```

For example,

```
PROC PRINT DATA = example1 ;
```

Telling the PROC What Variables to Process

If you want the procedure to use all variables in the data set, you need do nothing since SAS will automatically use all the variables. (For computational procedures, SAS will automatically use all the numeric variables only.) But if you want to analyze only certain variables, you can use the VARIABLES statement (usually abbreviated as VAR) to specify the

names of the variables to be processed. The VAR statement is placed after the PROC statement to which it applies. The VAR statement has the form

```
VAR  variablenames ;
```

For example, if we want to print only the variables height and weight from the data set example1, we would use the statements

```
PROC PRINT DATA= example1;
     VAR   height  weight ;
```

Processing the Data in Subsets

If you want the entire data set processed at once, SAS will do this automatically. Often, however, you may want to process the data set in subsets. You can ask SAS to process the data set in subsets by including a BY statement after the PROC statement. The BY statement has the form

```
BY variablename(s);
```

For example to compute the mean height and mean weight for the subset of females and for the subset of males, you would specify the variable sex in the BY statement:

```
PROC MEANS DATA= example1;
     VAR  height  weight;
     BY  sex;
```

These three statements tell SAS to

- Run the procedure MEANS on the data set example1
- Analyze only the variables height and weight
- Run the procedure once using only the observations for which the value of the variable sex is 'F', then run the procedure again using only the observations for the which the variable sex has the value 'M'

Note that before you can use a BY statement in a SAS procedure, you must first make sure that the observations contained in the data set are arranged in sorted order of the BY variables. This can be accomplished by using the SORT procedure. For our example the SORT step would look like

```
PROC SORT DATA= example1;
        BY sex;
```

These statements would be placed before the PROC MEANS statement:

```
PROC SORT DATA= example1;
        BY sex;
RUN;
PROC MEANS DATA= example1;
        VAR  height  weight;
        BY  sex;
RUN;
```

Now that we have discussed the general specifications of PROC statements, we can focus on specific SAS procedures and how they work.

Rearranging Your Data Using PROC SORT

The order of the data observations usually doesn't matter for many of the statistical analyses researchers perform. Often, however, you may want the data arranged in some specific order. For example, you may wish to print the data set example1 in alphabetical order by name. This will aid in finding records for specific individuals. Or you may wish to perform a "matched merge" on two data sets. In this situation, both sets of data need to be arranged in the same sorted order. Or you may wish to analyze your data in subsets.

To sort the data use a PORC SORT statement followed by a BY statement that specifies the variable(s) by which you wish to sort the data. For example, to sort the data set example1 in alphabetical order using the variable name:

```
PROC SORT DATA=  example1;
        BY name;
```

You may specify more than one variable in the BY statement. The first variable in the BY statement is the primary sort key, the second variable in the BY statement is the secondary sort key, and so on. The data set is sorted according to the primary sort key. For records with the same value for the primary sort key, these records will be sorted using the secondary sort key. An example:

```
PROC SORT;
        BY lastname  firstnam ;
```

The sorted data set will be arranged similar to the list of names in the telephone directory. Those records with the same lastname will be sorted alphabetically by firstnam.

SAS normally reads the data set identified by the DATA= option in the PROC SORT statement (or the most recently created data set if the DATA= option is omitted from the PROC SORT statement). SAS rearranges the data set in the order of the variable(s) specified in the BY statement and stores the rearranged data set with the same name as the original (unsorted) data set. The unsorted version of the data "disappears". You may wish to keep both the sorted and unsorted versions of the data set. This can be accomplished by using the OUT= option in the PROC SORT statement. For example,

```
PROC SORT DATA=example1  OUT= ex1sort ;
        BY sex;
```

The data set ex1sort contains the same observations as the data set example1, but the observations are sorted by the values of the variable sex.

If the sort key is a character variable, SAS will sort the data in alphabetic order (ascending order). If you wish to sort the data in reverse alphabetic order, you must specify the option DESCENDING before the name of the character variable in the BY statement. For example,

```
PROC SORT DATA= example1;
```

```
BY  DESCENDING  name;
```

will rearrange the observations such that those observations whose value for name begins with a "Z" will be placed first in the sorted data set, and those observations whose value for name begins with an "A" will be placed last in the sorted data set. The option DESCENDING applies only to the variable which immediately follows.

Similarly, data sorted using a numeric variable are arranged in ascending order (low values to high values). The DESCENDING option may also be used for a numeric sort key:

```
PORC SORT DATA= example1;
     BY DESCENDING height;
```

It should be noted that PROC SORT does not produce any printed output, although it does print a message in the SAS log file indicating the amount of storage and time was used to sort the data, and how many observations the sorted data set contains. In order to print the sorted data set you must use the PRINT procedure.

Printing Your Data Using PROC PRINT

PROC PRINT produces a listing of the values of some or all of the variables in a SAS data set. Totals and subtotals for numeric variables can also be printed. The general form of the PROC PRINT statement is

```
PROC PRINT  options;
```

The simplest form

```
PROC PRINT;
```

will print the all observations and variables in the most recently created SAS data set. The observations and variables are printed in the same order as in the SAS data set.

If the SAS job creates several data sets, you can specify which data set you want printed by using the DATA= option in the PROC PRINT statement:

```
PROC PRINT DATA= datasetname;
```

The VAR statement may be used in conjunction with the PROC PRINT statement to specify which variables you want printed, or to specify the order in which the variables will be printed. Recall the example

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC PRINT DATA= example1;
RUN;
```

All records in the data set example1 will be printed to the OUTPUT window, and will appear in the same order as in the data set example1.

Suppose that you want only the variables name and height to be printed. You can do this using a VAR statement:

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC PRINT DATA= example1;
        VAR  name  height;
RUN;
```

Suppose that you wish to print all the variables, but change the order in which they appear on the printout. This can also be done using a VAR statement:

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC PRINT DATA= example1;
        VAR  name  sex  height  weight  age ;
RUN;
```

The printed output will show the variables in the order specified in the VAR statement.

A BY statement can be used with PROC PRINT to obtain separate analyses on observations in groups defined by the variable(s) in the BY statement. When a BY statement appears as part of the PRINT procedure, the procedure expects the data to be sorted in the order of the BY variables.

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC SORT DATA =  example1;
        BY sex;
RUN;
PROC PRINT DATA= example1;
        VAR  name  sex  height  weight  age ;
        BY sex;
RUN;
```

This example will produce a listing for females (sex='F') and a separate listing for males (sex='M').

The SUM statement may also be used with the PROC PRINT statement to specify variables whose values are to be totaled. The SUM statement has the form

```
SUM   variablename(s)  ;
```

As an example, suppose that you wish to total the values of the variable age.

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC PRINT DATA= example1;
        VAR  name  sex  height  weight  age ;
        SUM  age;
RUN;
```

If a SUM statement and a BY statement are used in the same PRINT procedure, totals for the variables in the SUM statement are computed for each BY group. The following example will total the ages for the males, and compute a total of the ages for the females separately.

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC SORT DATA =  example1;
        BY sex;
RUN;
PROC PRINT DATA= example1;
```

```
        VAR  name  sex  height  weight  age ;
        BY sex;
        SUM  age ;
RUN;
```

You can specify a variable in the SUM statement which does not appear in the VAR statement.

Exercise 5:

a) Create a data set (or use a data set from a prior example). Sort the data using a character variable. Obtain an overall total for (at least) one numeric variable (Use a SUM statement). Print the sorted data set.

b) Obtain subtotals for (at least) one numeric variable for each BY group. (Use a SUM statement). Print the sorted data using a BY statement.

Describing Your Data Using PROC MEANS

PROC MEANS can be used to compute various univariate descriptive statistics for specified variables including the number of observations, mean, standard deviation, variance, minimum and maximum values, the standard error of the mean, uncorrected sum of squares, corrected sum of squares, the coefficient of variation, and confidence limits for the mean. You can also ask PROC MEANS to perform a t-test on the hypothesis that the population mean is zero.

The general form of the PROC MEANS statement is

```
PROC MEANS   options;
```

The simplest form

```
PROC MEANS;
```

will automatically compute and print the mean, standard deviation, minimum and maximum values for each of the numeric variables in the most recently created data set. If the linesize of the output window is large enough, PROC MEANS will also print the standard error of the mean, the

variance, the sum, and the coefficient of determination. You can specify the data set which PROC MEANS will process by using the DATA= option in the PROC MEANS statement:

```
PROC MEANS DATA= datasetname;
```

The VAR statement may be used in conjunction with the PROC MEANS statement to specify the variables for which you want descriptive statistics computed.

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC MEANS DATA= example1;
        VAR  age  height;
RUN;
```

Descriptive statistics will be printed only for the variables specified in the VAR statement.

A BY statement can be used with PROC MEANS to obtain separate analyses on observations in groups defined by the variable(s) in the BY statement. When a BY statement appears as part of the MEANS procedure, the procedure expects the data to be sorted in the order of the BY variables.

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC SORT DATA =  example1;
        BY sex;
RUN;
```

```
PROC MEANS DATA= example1;
        VAR   height  weight  age ;
        BY sex;
RUN;
```

This example will produce a listing of descriptive statistics for females (sex='F') and a separate listing for males (sex='M').

You can ask PROC MEANS to produce only the descriptive statistics you desire. Corresponding to each statistic is a keyword that can be specified in the PROC MEANS statement:

N	number of nonmissing observations (in a subgroup)
NMISS	number of observations with missing values (in a subgroup)
MEAN	sample mean
STD	sample standard deviation
MIN	minimum value
MAX	maximum value
RANGE	range
SUM	sum
VAR	variance
USS	uncorrected sum of squares
CSS	corrected sum of squares
CV	coefficient of variation
STDERR	standard error of the mean
CLM	confidence limits on the mean

LCLM	lower confidence limit on the mean (one-sided)
UCLM	upper confidence limit on the mean (one-sided)
T	Student's t statistic for testing H_0: $\mu = 0$
PRT	probability of a greater absolute value for Student's t statistic

The confidence intervals have a 95% confidence level (by default). If you desire a different confidence level, use the ALPHA= option on the PROC MEANS statement. For example, to produce 99% confidence limits,

```
PROC MEANS CLM ALPHA=0.01;
```

Now for a more complicated example, including a t-test, we might use the following program:

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC SORT DATA =  example1;
        BY sex;
RUN;
PROC MEANS DATA= example1  N MEAN STD STDERR CLM T
        PRT ALPHA=0.10;
        VAR   height  weight  age ;
        BY sex;
RUN;
```

With some clever program statements, you can "trick" PROC MEANS into performing tests involving hypotheses other than H_0: $\mu = 0$. Suppose that

you wish to test H_0: $\mu = 60$ for the variable height. This can be accomplished by

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
/* set up variable for hypothesis test H0: μ = 60  */
htdiff = height  -  60 ;
DATALINES;
Data records go here
;
RUN;

PROC MEANS DATA= example1  N MEAN STD STDERR T PRT;
        VAR   htdiff ;
RUN;
```

PROC MEANS can also produce an output data set containing the computed descriptive statistics by using an OUTPUT statement, which has the form

```
OUTPUT  OUT= datasetname    statistics ;
```

For example,

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS program statements go here
DATALINES;
Data records go here
;
RUN;

PROC MEANS DATA= example1  N MEAN STD STDERR;
        VAR   weight ;
        OUTPUT  OUT= calcstat  N MEAN STD STDERR;
RUN;
```

The data set calcstat will contain the specified statistics for the variable weight.

Exercise 6: Create a data set (or use one you have already created) and compute descriptive statistics (means, standard deviations, etc.) on all numeric variables. Interpret the output.

Frequencies and Crosstabulations Using PROC FREQ

Frequency tables and crosstabulation tables provide a way to summarize data for ordinal and categorical variables. Frequency tables show the distribution of a variable's values. Crosstabulation tables show joint distributions for two or more variables. The SAS procedure FREQ will produce one-way to n-way frequency and crosstabulation tables. PROC FREQ can also compute chi-square statistics, the phi coefficient, the contingency coefficient, and Cramer's V statistic. These statistics measure the degree of association of the values of the variable in a contingency table. Tests for independence can be performed in PROC FREQ. In addition, the procedure can compute kappa statistics, which can be used as a measure of agreement between two classification systems.

The general form of the PROC FREQ statement is

```
PROC FREQ   options;
```

Available options include the DATA= option, which will tell PROC FREQ which data set to process. You can request that PROC FREQ print only one table per page by using the PAGE option (otherwise multiple tables per page will be printed, as space permits).

The simplest form is

```
PROC FREQ;
```

and will produce frequency tables for all the variables in the most recently created data set. There is typically little purpose in obtaining frequency tables for continuous numeric variables, or for character variables whose values are unique, e.g., name. You can ask PROC FREQ to construct and print frequency and crosstab tables for selected variables in the data set by using the TABLES statement. Any number of TABLES statements may be

included in one execution of PROC FREQ. If no TABLES statement is included in the procedure, PROC FREQ will construct one-way frequency tables for each of the variables in the data set. The TABLES statement has the form

```
TABLES   requests  /  options  ;
```

To request a one-way frequency table on the variable sex :

```
PROC FREQ;
TABLES     sex  ;
```

If you request a one-way table and specify no options, PROC FREQ produces frequencies, cumulative frequencies, percentages of the total frequency, and cumulative percentages for each level of the variable specified in the TABLES statement.

To request a two-way table on the variables sex and religion :

```
PROC FREQ;
TABLES     sex  * religion;
```

If you request a two-way table and specify no options, PROC FREQ produces a crosstabulation table that includes cell frequencies, cell percentages of the total frequency, cell percentages of the row frequencies, and cell percentages of the column frequencies.

Three - way and general n- way tables requests are specified in a similar fashion:

```
PROC FREQ;
TABLES     a * b * c;
```

will result in a three-way crosstab table.

For each frequency table or crosstabulation table you want, put a table request in the TABLES statement.

Missing values of each variable are typically excluded from the table that PROC FREQ produces, but the total frequency of missing values is printed below each table.

You can include options in the TABLES statement after the slash (/). If you use the option MISSING, PROC FREQ will treat missing values as nonmissing values and include them in the calculation of percentages and other statistics. The OUT= option may be included in a TABLES statement to produce a data set containing the levels of the variables and counts and percentages.

Options can be specified to request additional table information. The option EXPECTED will cause expected cell frequencies (under the hypothesis of independence) to be printed. The option CELLCHI2 tells PROC FREQ to print each cell's contribution to the total chi-square statistic.

Options to request additional statistical analyses can also be specified in the TABLES statement. The option CHISQ requests a chi-square test of homogeneity or independence, along with measures of association based on the chi-square. These include the Pearson chi-square, liklihood ratio chi-square, Mantel-Haenszel chi-square, the phi coefficient, the contingency table coefficient, and Cramer's V. The option EXACT requests Fisher's exact test. The option MEASURES will produce Pearson and Spearman correlation coefficients, Kendall's tau-b, Stuart's tau-c, and Somer's d statistics. The option AGREE can be used to produce kappa statistics.

An example of a program that use PROC FREQ follows:

```
DATA  socio1;
INPUT  name $ 1-10 sex $ 12 race $ 15-25  religion  $ 30-35  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC FREQ DATA =  socio1;
        TABLES     sex  ;
        TABLES     religion   race  / MISSING ;
        TABLES      sex* race   sex*religion  race*religion /
```

```
                EXPECTED  CELLCHI2  CHISQ  ;
        TABLES  sex*race*religion /  EXACT ;
RUN;
```

Exercise 7: Create a SAS program similar to the program example above. Run the program. Change some of the options. Interpret the output.

Getting Correlations Using PROC CORR

Correlation analysis provides a method to measure the strength of a linear relationship between two numeric variables. PROC CORR can be used to compute Pearson product-moment correlation coefficient between variables, as well as three nonparametric measures of association, Spearman's rank-order correlation, Kendall's tau-b, and Hoeffding's measure of dependence D. PROC CORR also computes simple descriptive statisticsThe general form of the PROC CORR statement is

```
PROC CORR  options;
```

The simplest form

```
PROC CORR;
```

will compute pairwise Pearson correlation coefficients for all numeric variables in the most recently created SAS data set. Allowable options in the PROC CORR statement include the DATA= option, as well as options to produce an output data set. The OUTP = option will create a data set containing Pearson correlations; the OUTS = option will create a data set containing Spearman correlations; the OUTK = option will create a data set containing Kendall correlations; and the OUTH = option will create a data set containing Hoeffding statistics.

Options to select the kinds of correlations to be computed include PEARSON (these are computed by default), KENDALL, SPEARMAN, and HOEFFDING.

A VAR statement can be used with PROC CORR to specify which variables are to be analyzed. Pairwise correlation coefficients will be computed for all variables in the VAR statement. For example,

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
DATALINES;
Data records go here
;
RUN;

PROC CORR  DATA= example1  PEARSON   SPEARMAN
        VAR   weight  height ;
RUN;
```

PROC CORR also computes probabilities to test the null hypothesis H_0: $\rho=0$. If three or more numeric variables are in the input data set or are specified in the VAR statement, the correlations and probabilities are printed in matrix form.

The WITH statement can be used to obtain correlations for specific combinations of variables. Suppose you wish to compute a correlation for age with height, and another for age with weight, but you are not interested in a correlation of weight with height. You can accomplish this by using the WITH statement in conjunction with the VAR statement as follows:

```
PROC CORR  DATA= example1  PEARSON   SPEARMAN
        VAR   weight  height ;
        WITH  age;
RUN;
```

A BY statement can be used with PROC CORR to obtain separate analyses on observations in the groups defined by the BY statement. When a BY statement is used, PROC CORR expects the data to be sorted in the order specified in the BY statement. As an example suppose that you want separate analyses for males and females. A SAS program to accomplish this follows:

```
DATA example1;
```

```
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC SORT DATA =  example1;
        BY sex;
RUN;
PROC CORR DATA=example1;
RUN;
PROC CORR DATA= example1 ;
        VAR   height  weight  age ;
        BY sex;
RUN;
```

The default method for handling missing values is to use all nonmissing pairs of values for each pair of variables in the VAR or WITH statements. This implies that differing numbers of observations may be used to compute correlation coefficients for different pairs of variables. If the NOMISS option is used in the PROC CORR statement, any observations with missing values for any of the variables in the VAR or WITH statements are excluded from the analysis, i.e., all pairwise correlations are computed using the same number of observations.

Exercise 8: Use one of your previous examples and compute pairwise correlations for all numeric variables. Run another analysis using a VAR and a WITH statement. Next run another analysis using a BY statement. Interpret the results.

Producing Graphs Using PROC PLOT

PROC PLOT can be used to generate a graph of the values of one variable plotted against values of another variable. The general form of the PROC PLOT statement is

```
PROC PLOT   options;
```

The simplest form is

```
PROC PLOT;
```

Allowable options include the DATA= option. Another useful option is the UNIFORM option. This option requests uniform axes scaling when a BY statement is used. This allows you to easily compare plots for different levels of the BY variables.

You can ask PROC PLOT to construct and print plots for specified variables in the data set by using the PLOT statement. Any number of PLOT statements may be included in one execution of PROC PLOT. The PLOT statement has the form

```
PLOT  requests  /  options  ;
```

The requests in the PLOT statement indicate the variables to be plotted and the plotting characters to be used to mark points on the plot. The first variable in a request will be represented on the vertical axis (y-axis) and the second variable in the request will be represented by the horizontal axis (x-axis). The plot requests can have the form

```
var1 * var2
```

When this form is used, SAS determines the plotting character to be used to mark points on the plot. When a point on the plot represents one observation PROC PLOT uses the plotting character A to mark that point; when a point on the plot represents two observations PROC PLOT uses the plotting character B to mark that point; and so on.

Plot requests may also have the form

```
var1 * var2  = 'character'
```

This form allows the user to specify the plotting character to be used to mark points on the plot. The character you choose can represent values form one observation, or more than one observation. As an example,

```
PROC PLOT;
      PLOT  weight * height = '*' ;
```

The third form that a plot request may have is

var1 * var2 = var3

This form specifies that points on the plot of var1 * var2 will be indicated by the value of var3. As an example, suppose that the variable sex has the values 'F' or 'M'. Then the statements

```
PROC PLOT;
        PLOT  weight * height = sex ;
```

Will cause the points on the plot to be marked with either an F or an M.

Any number of plot requests can appear in a single PLOT statement. The plots will appear on separate graphs unless the OVERLAY option is specified in the PLOT statement. The OVERLAY tells PROC PLOT to print the plot requests specified in the PLOT statement on a single graph. This can be useful for plots in a regression analysis.

If values of any of the variables in a plot request are missing, PROC PLOT does not include the observation in the plot.

A BY statement can be used with PROC PLOT to obtain separate plots on observations in groups defined by the BY variable(s). When a BY statement appears, PROC PLOT expects the data to be sorted in the order of the BY variables.

Now for a fuLl-blown example:

```
DATA example1;
INPUT  name $ 1-10 sex $ 12 age 14-15 height 17-18 weight 20-22  ;
SAS programming statements go here
DATALINES;
Data records go here
;
RUN;
PROC SORT DATA =  example1;
        BY sex;
```

```
RUN;
PROC PLOT DATA=example1;
        PLOT  weight * height = sex   height *age  weight*age ;
RUN;
PROC PLOT  DATA= example1 ;
        PLOT   age* height  ='H'  age*weight ='W'  /  OVERLAY ;
        BY sex;
RUN;
```

Exercise 9: Generate some plots. Try various plotting symbols and options.

Linear Regression Analysis using PROC GLM

Regression analysis is a statistical method of obtaining an equation that represents a linear relationship between two variables (simple linear regression), or between a single dependent and several independent variables (multiple linear regression). Either the GLM procedure or the REG procedure can be used to perform simple and multiple linear regression. PROC GLM can also handle a wide variety of other linear models. The statements required to perform a regression analyses in either procedure are similar.

The general form of the PROC GLM statement is

```
PROC  GLM    options  ;
```

Options which you may need to run a regression analysis include the DATA= option. Other options apply to other types of linear models.

The simplest form of the PROC GLM statement is

```
PROC GLM ;
```

A MODEL statement is also required in order to perform a regression analysis. The form of the MODEL statement is

```
MODEL  dependent = independents   /  options;
```

The dependent variable is specified on the left of the equal sign, and the independent variable(s) on the right of the equal sign, separated by blanks. Allowable options include INTERCEPT (or just INT), which tells PROC GLM to print hypothesis tests associated with the intercept as an effect in the model. By default, PROC GLM includes the intercept in the model, but does not print associated tests of hypotheses. The option NOINT tells PROC GLM to omit the intercept term from the model. The CLI option tells PROC GLM to print confidence limits for individual predicted values for each observation. The ALPHA= option specifies the alpha level for confidence intervals. By default the alpha level is 0.05. The option P will cause the observed, predicted, and residual values to be printed for each observation that does not contain missing values. An example of the MODEL statement is

```
MODEL weight = height age / CLI P ;
```

The OUTPUT statement can be used to create a SAS data set that contains all the input data, as well as predicted values, confidence limits, residuals, and regression diagnostics. The form of the OUTPUT statement is

```
OUTPUT OUT=datasetname keyword=name ;
```

The keywords are used to specify which values to store in the output data set. Useful keyword options include PREDICTED (or P) for predicted values, L95 and U95 for lower and upper confidence limits for an individual prediction, and RESIDUAL (or R) for residual values. An example of an OUTPUT statement is

```
OUTPUT OUT=stats P=pred R=res L95=lower U95=upper;
```

The OUTPUT statement is useful when creating a data set that will be used later by another SAS procedure (such as PROC PLOT).

A BY statement can be used with PROC GLM to obtain separate plots on observations in groups defined by the BY variables. When a BY statement appears, PROC GLM expects the data to be sorted in the order of the BY variables.

Example: Simple Linear Regression

Develop a simple linear regression equation to predict the age (days) of a rat from its body weight (grams). The data is given below in the following SAS program:

```
DATA  rats;
INPUT   weight  age ;
DATALINES;
76  28
89  33
154  35
189  37
180  38
180  47
241 66
320 67
271 70
370 82
;
PROC GLM  DATA=rats;
        MODEL   age = weight  / p;
        OUTPUT  OUT=stats  P=pred  R=res  L95=lower  U95=upper;
RUN;

PROC PLOT DATA=stats;
        PLOT   age *weight  pred*weight='*'  / OVERLAY  ;
RUN;
```

At the top of the OUTPUT window we see information on the number of observations in the data set. Next we see an overall analysis of variance table for the specified model. This table shows the breakdown of the total sum of squares for the dependent variable (age) into orthogonal components: the portion of variation accounted for by the model (the model sum of squares), and the portion that the model does not account (the error sum of squares). These are found under the column "Sum of Squares"

In this section, we also see a column labeled "F Value" and another labeled "Pr > F". The F value for the model is calculated by dividing the mean square for the model by the error mean square. For the example above, the model F value is 61.56, and the probability of getting a greater F value is

0.0001. This indicates that the model explains a significant portion of the variation in the ages.

Beneath the analysis of variance table are some computed statistics. The R-Square value is computed as the ratio of the sum of squares for the model divided by the sum of squares for the corrected total. R^2 measures how much variation in the dependent variable (age) can be explained by the model. The value of R^2 can range from 0 to 1. In general the larger the value of R^2 the better the fit of the model. In our printout we see a value of 0.884995, indicating that the model can account for over 88% of the value of age, just by knowing the weight.

Also printed are the coefficient of variation (C.V.), the root mean square error (Root MSE), which is the square root of the error mean square and is also known as the standard deviation of the dependent variable (age).The mean of the dependent variable is also printed.

In the next section of output we see the sum of squares attributable to each variable in the model. The TYPE I SS measures the increment in the sum of squares for the model as each variable is added to the model. The TYPE III SS measures the sum of squares due to adding that variable last in the model. In this example, since only one independent variable was included in the model, these sums of squares are equal. The "F Value" and "Pr > F" for the type III sum of squares are equivalent to the results of a t-test for testing that the regression coefficient equals zero. In our example, we see that this probability is 0.0001, indicating that the coefficient of the variable weight is significantly different from zero.

The next section is a report on the parameter estimates. The intercept is estimated to be 10.866, and the results of the t-test (testing the null hypothesis that the parameter equals zero) yield a p-value of 0.0819, indicating that the intercept is "somewhat" useful in the model. The coefficient of the weight term is 0.190404, with a p-value of 0.001, indicating that the coefficient of weight does not equal zero. Hence the independent variable weight does contribute significantly to the model. Standard errors of each parameter estimate are also printed. The next part of the printed output shows the observed and predicted values of age, along with the residual vales. The First Order Autocorrelation and Durbin-Watson D statistics measure the presence of a first -order atucorrelation are also computed.

A plot of predicted ages and actual ages vs. weight appear on the next section of output. The plot of residuals that appears next can be used to detect patterns in the residuals, and may be used to determine if a nonlinear model may be more appropriate. In our plot, the residuals appear to be randomly scattered about the line r=0. There appears to be no pattern in the residuals plot. A linear model seems appropriate.

Example: Multiple Linear Regression

In a study of grade school children, ages, heights, weights and scores on a physical fitness exam were obtained from a random sample of 20 children. The data is given below. Find a multiple linear regression equation relating the scores to the ages, heights, and weights of the children.

Our SAS program might look like:

```
DATA  phys;
INPUT  score age height weight;
DATALINES;
58  7  47.5  53
54  7   45  50
55  9  52.5  85
74  7  48  52
86  9  55  76
98  8  51  64
96  9  53  75
70  7  46  75
40  7  48  68
67  9  50.5  74
41  6  45  40
41  7  48.5  66
47  8  50.5  65
45  8  49.0  70
92  9 51.5 70
50  7  46.5  60
98  9  53.5  77
42  8  45  65
64  8  52.5  65
70  8  51.5  67
```

```
;
PROC GLM DATA = phys;
        MODEL   score = age   height   weight    / p;
RUN;
```

When we look at the analysis of variance table in the output, we see that the model does seem to explain a significant portion of the variation in the physical fitness scores, as indicated by a p-value of 0.0143 under "Pr > F" . Although the model is significant in explaining scores, the model is not a very good fit to the data. The r-square value of 0.4738 indicates that less than half of the variation in scores is accounted for by the model.

In the next section of output we see the sum of squares attributable to each variable in the model. The TYPE I SS measures the increment in the sum of squares for the model as each variable is added to the model. The TYPE III SS measures the sum of squares due to adding that variable last in the model. In this example, we have three independent variables included in the model. Hence these sums of squares are different. The "F Value" and "Pr > F" for the type III sum of squares are equivalent to the results of a t-test for testing that the regression coefficient equals zero. In our example, it appears that none of the variables is significant in explaining the test scores. However, looking at the TYPE I SS we see that age is significant when it is the first variable added to the model. These results may seem contradictory. But you have to remember how TYPE I and TYPE III SS are computed. A reasonable explanation for this seeming discrepancy is that age is likely to be significantly correlated with both height and weight. Thus when age is the last variable entered into the model, most of the information contained in the variable age has already been "used" by the variables height and weight in explaining test scores. When this type of situation arises you may want to repeat the analysis using some type of selection procedure.

The section for parameter estimates also indicates that none of the variables are significant in explaining test scores.

Example: Stepwise Selection Multiple Regression

Run the same model using the stepwise selection method.

Note that PROC GLM will not perform model selection methods. We must use SAS's regression procedure (PROC REG) to do this. The SAS program is

```
DATA  phys;
INPUT  score age height weight;
DATALINES;
58  7  47.5  53
54  7   45  50
55  9  52.5  85
74  7  48  52
86  9  55  76
98  8  51  64
96  9  53  75
70  7  46  75
40  7  48  68
67  9  50.5  74
41  6  45  40
41  7  48.5  66
47  8  50.5  65
45  8  49.0  70
92  9 51.5 70
50  7  46.5  60
98  9  53.5  77
42  8  45  65
64  8  52.5  65
70  8  51.5  67
;
PROC REG DATA = phys;
        MODEL   score = age   height   weight    / p
                SELECTION = STEPWISE
                SLENTRY =0.3
                SLSTAY = 0.3;
RUN;
```

The STEPWISE selection method begins by fitting an intercept term to the data. If you do not want an intercept term in your model, use the option NOINT in the MODEL statement. After fitting the intercept, SAS performs an "Analysis of Variables Not in the Model". For STEPWISE, the

first variable selected for entry into the model will be the variable with the lowest p-value (assuming that this p-value is less than the value of SLENTRY). The option SLENTRY=.3 specifies the significance level for entry into the model. The default value is 0.50 for FORWARD selection and 0.15 for STEPWISE selection. The option SLSTAY=.3 specifies the significance level for remaining in the model. The default value is 0.10 for BACKWARD selection and 0.15 for STEPWISE selection.

On the SAS output (Step 1), we see that the variable height has entered the model with a p-value of 0.0019.

After entering a new variable into the model, SAS performs a BACKWARD elimination step to see if any of the variables in the current model can be removed. SAS will next determine which (if any) of the remaining explanatory variables should be entered into the model. None of the remaining variables meets the criterion for entry into the model. The STEPWISE selection procedure terminates. A summary of the variables selected for entry and the variables removed from the model at each step of the process is given in the SAS output.

We have mentioned three selection methods available in PROC REG. These three methods are FORWARD for forward selection, BACKWARD for backward selection, and STEPWISE for stepwise selection. These methods are specified using the SELECTION= option in the MODEL statement. Intercept parameters are always forced to stay in the model unless the NOINT option is specified in the MODEL statement.

When SELECTION=FORWARD, PROC REG first estimates parameters for variables forced into the model, i.e. the intercept. (There is a way to force some of the explanatory variables into the model. Use the INCLUDE= variable(s) option in the MODEL statement) Next SAS computes the adjusted Chi-square statistics for all variables not in the model and examines the largest of these statistics. If it meets the criterion for entry (SLENTRY) into the model, the variable with this largest adjusted Chi-square is entered into the model. Once a variable is entered into the model it is never removed from the model. The process is repeated until none of the remaining variables meets the specified entry level.

When SELECTION=BACKWARD, parameters for the complete model as specified in the MODEL statement are estimated. The univariate tests based on the Maximum Likelihood Estimates are examined. The least significant variable that does not meet the criterion for staying (SLSTAY) in the model is removed. Once a variable is removed from the model it remains excluded. The process is repeated until no other variable meets the specified level for removal.

SELECTION=STEPWISE is similar to SELECTION=FORWARD except that variables already in the model do not necessarily remain in the model. Variables are entered into the model and removed from the model in such a way that each forward selection step is followed by one or more backward elimination steps. The stepwise selection process terminates if no further variable can be added to the model, or if the variable just entered into the model is the only variable removed in the subsequent backward elimination.

Other selection methods are available, for example RSQUARE. The RSQUARE selection procedure performs an "all subsets regression" and prints the models in decreasing order of R^2 magnitude within each subset size. The subset models selected by RSQUARE are optimal in terms of R^2 for the given sample, but are not necessarily optimal for the population from which the sample was obtained, or any other sample for which you may wish to make predictions.

While model selection techniques are useful for exploratory model building, no statistical method can be relied upon to identify the "true" model. Effective model building requires substantive theory to suggest relevant predictors and plausible functional forms for the model.

Exercise 10: Use the RSQUARE selection method on the physical fitness data. Determine the best one-variable, two-variable and three-variable linear models.