

An Introductory SAS Course

For use with Version 9.3

Updated by

**Cong Cao
Yating Cao
Quan Hu
Xiaosu Tong**

Originally created by

**Christina Wassel
Chenghong Li
January 2002**

Statistical Consulting Service

Purdue University

August, 2012

I. Introduction

What is SAS?

SAS is a statistical software package that allows the user to manipulate and analyze data in many different ways. Because of its capabilities, this software package is used in many disciplines, including the medical sciences, biological sciences, and social sciences. Knowing the SAS programming language will likely not only help you in your current class or research, but also in obtaining a job.

How to obtain SAS

SAS Version 9.3 is installed on all ITaP (Information Technology at Purdue) machines in all ITaP labs around campus. To get into the program, click Start, All Programs, Standard Software, Statistical Packages, and finally SAS.

If you would like to install SAS Version 9.3 on your home computer or laptop, ITaP Software Loans has SAS CDs available free of charge to students, faculty, and staff to do this. Go to Young Hall 5th floor reception area. (Remember to take your student ID to sign out the disks over night!)

Those with a Purdue career account can also access SAS version 9.3 on a personal computer through ITaP's software remote. After logging in with your career account, go to the *Standard Software* folder, then the *Statistical Packages* folder.

After you open SAS

After you open SAS, you will see four windows, the explorer window on the left hand side, the output window, the log window, and the program editor on the right hand side. You may have to change the size of the 3 right side windows to display all of them together.

In the program editor, you will type the program (i.e. SAS commands) that you will eventually run. It works similarly to Microsoft Word. (You can cut, paste, move the cursor, etc.) The enhanced program editor will give you color-coded procedures, statements, and options (more on these later) that will help you to find errors in your program before you even run it.

The log window will inform you of any errors in your program and the reason for the errors. The log window is EXTREMELY IMPORTANT to help figure out any errors in your program. Always check the log window first to determine if your program ran properly!

The output window is where your output appears after the program has been run. Unlike previous versions of SAS, version 9.3 output by default uses HTML. In case you do not like using HTML, we will discuss later how to get output similar to previous versions of SAS.

With the explorer window, you can open/view data that is read into SAS. In the explorer window, click on libraries, then the work folder, and this will show you any datasets you have read or created in SAS for that session. Be careful, all the datasets you created in work folder are temporary, which means once you close the current SAS session; all temporary datasets created will be deleted automatically.

Reading data into SAS

There are three basic options (others do exist) for reading datasets into SAS.

1) With an `infile` statement:

Using an `infile` statement will bring text file (.txt) data in from a different drive (e.g., A:, H:, C:, F:) and make them available for the entire SAS session that you run. (To use data in an Excel spreadsheet, use 2) Importing the data set listed below.) To read in the text file, the commands would be, assuming you were reading in a data set from drive A:

```

data a1;
infile 'A:\filename';
input x1 x2;

```

You must also name your dataset. You can name it anything you like; here it is named a1. The `input` statement identifies the variables in your dataset so you can use them for analysis. They can also be named whatever you would like; here they are named x1 and x2. Software remote users may encounter difficulties using an `infile` statement and may only be able to access data sets on your career account (H: drive).

2) Importing the dataset:

To import the data from another drive, go to “File”, then “Import data”. An Import wizard window will pop up and ask you the format of the file to import. Click on the pull-down menu and select your file type (i.e. Excel, Access, text, etc.). Then, click “Next” which will take you to a window that asks the file’s pathname. Typing in the place where your file exists (e.g., H:\StatHW\data). Or, if you are not exactly sure where your file is, click on “Browse”, and this will help you to locate it, then “Ok”. Click “Next” again, and then it will ask you to name your data set. This can be anything you would like. Once you name it, you must continue to use this name in your program to reference this particular data set. Click “Next” one more time, and then click “Finish”.

Please note that SAS 9.2 can only read Excel files saved under Excel 2003 or earlier, (i.e. with .xls extension). With 9.2, if you have a .xlsx files, just open the file in Excel and save it in an earlier version before reading it into SAS. With SAS 9.3, the Import Wizard will allow you to choose the correct type of file and does allow .xlsx.

3) Using the `cards` or `datalines` statements:

Another option is to put your dataset directly into the program editor. This generally works best when your dataset is fairly small (e.g., for a class assignment). The code for this is:

```

data a1;
input x1 x2 @@;
cards;
2 9.2
3 11
2 10.1
1 8.5
;

```

or

```

data a1;
input x1 x2 @@;
datalines;
2 9.2
3 11
2 10.1
1 8.5
;

```

It is very important that the last semi-colon goes on the next line after all of the data (as shown above), otherwise your last observation will be deleted! The reason we put “@@” in the end of the input statement is that we want to let SAS read two data points each time. This means that when we are typing the data after `cards` or `datalines` statement, we can type all data points just in one line or in multiple lines. Note that there are only spaces between each data point, no commas or other characters. Using the `cards` or `datalines` statement will give the same results.

Important basic syntax to know:

In order to successfully run any program, you need the following basic elements:

- 1) a semi-colon at the end of every line
- 2) a **data** statement that names your data set (unless you import the data set)
- 3) **input** statement (unless you import the data set)
- 4) at least one space between each word or statement
- 5) a **run** statement

A semi-colon is the way to tell SAS that a particular operation, procedure, or statement is finished, and tells SAS to look for the next one. The **data** statement names your data set so you can reference it later in your program. The **input** statement tells SAS the names of the variables in your data set so that they can also be referenced later. For the names of variables, it is case sensitive in SAS, however it is not for the name of dataset. Only one space is required to tell SAS that things are separate. If you have more than one space, that is fine too. A **run** statement tells SAS to process the previous bit of code that you wrote. If there is no **run** statement, SAS will not process anything. (Lack of semi-colons and run statements are two most common mistakes in a program.)

After you finish writing the code, you should click “” icon in the toolbar at the top of the screen to get the result. (This tells SAS to run the code.)

An example of this follows:

```
data yourdatasetname;  
  infile 'H:\StatHW\yourfilename.dat';  
  input variable1 variable2 (up to the number of variables in the set);
```

If you use the **cards** or **datalines** statements instead, they must both be **preceded** by the **input** statement. An example of using the **cards** statement to read in data is on the following page.

II. Data Steps and Procedures

What is a data step? How are they useful?

A SAS program is composed of two parts: data steps that deal with data cleaning and data format, and procedures that perform required statistical analyses and/or graphically present the results. Data steps are important for several reasons. First, the dataset may not be in a SAS compatible format, although this is usually not the case for the datasets in class examples or exercises. Second, sometimes you need to extract some of the variables or some of the observations from the dataset to perform analysis. Third, different procedures may require the same dataset in different format. A data step is needed to transform the dataset into the appropriate format for a procedure.

Mathematical operations are listed in the following table:

Function	Operator	Example
Addition	+	Height + weight
Subtraction	-	Height – weight
Multiplication	*	Height * age
Division	/	Weight / height
Power	** or ^	Weight ** 2
Equal	= or eq	Weight = 120
Unequal	<> or ne	Weight <> 120
Less than	< or lt	Weight < 120 or weight lt 120
Less than or equal to	<= or le	Weight le 120
Greater than	> or gt	Weight gt 80
Greater than or equal to	>= or ge	Weight ge 80

Manipulating variables in a data step (recoding, if/then statements)

To illustrate the data manipulation, let's take a sample data set:

```
data a1;
  input gender $ age weight;
  cards;
M 13 143
M 16 132
F 19 140
M 20 120
M 15 110
F 18 95
F 22 105
;
```

Suppose you want a data set of females only. The following SAS code will create a new data set call aa and store those observations whose value for the variable gender is not 'M'. The `set a1` statement after the `data aa` statement tells SAS to make a copy of the dataset a1 and save it as aa. The `if/then` statement deletes the observations in dataset aa whose gender variable has a value 'M'. Quotation marks are used on M because gender is a categorical variable. The dollar sign (\$) is used when you have a text variable rather than a numerical variable (i.e., gender coded as M, F rather than as 1 denoting male and 2 denoting female).

```
data aa;
set a1;
if gender eq 'M' then delete;
run;

or

data aa;
set a1;
if gender eq 'F';
run;

or

data aa;
set a1;
if gender = 'M' then delete;
run;

or

data aa;
set a1;
if gender = 'F';
run;
```

If you want to include only those who are 16 years or older:

```
data ab;
set a1;
if age lt 16 then delete;
run;

or

data ab;
set a1;
if age ge 16;
run;
```

You can also select variables from a dataset for analysis. The statement is `keep` or `drop`. For example, if you do not need the variable `age` in your analysis:

```
data ac;
set a1;
drop age;
run;

or

data ac;
set a1;
keep gender weight;
run;
```

This last statement will create a dataset that only contains the two variables specified, `gender` and `weight`.

What is a procedure?

A SAS program is composed of one or more (statistical) procedures. Each procedure is a unit, although some are needed to run others. Some frequently used procedures for statistical analysis are explained in detail below. Running a procedure typically results in output going to the output window. There are ways to suppress this output if you do not want to see each step of an analysis.

As mentioned earlier, SAS 9.3 by default has output in HTML and ODS Graphics is enabled. This results in well-organized tables of output that are not necessarily easy to manipulate. To get output similar to that in earlier versions can be done in two ways:

- 1) Modify the default setting by selecting Tools→Options→Preferences→Results from the menu at the top of the main SAS window. Check the “Create Listing” box and uncheck the “Create HTML” box. Also uncheck the “Use ODS Graphics”.
- 2) You can also control these via the following lines of code in the Editor window prior to running the first procedure:

```
ods html close;
ods listing;
```

The output below is shown using the listing output (i.e., default output from Version 9.2 and earlier).

Proc print

The output of this procedure is the data set that you specified by writing `data=dataset` option after the print key word. This option, `data = dataset`, is common for almost every SAS procedure to specify which dataset you want to work with. It is a good habit to use this option all the time and is very helpful especially when there are multiple datasets, as is usually the case when you are performing statistical analysis using SAS. Here’s an example of how `proc print` works. In the data step section, we created a data set called `a1` with three variables (`gender`, `age`, `weight`), and seven observations. It’s a good idea to always check if SAS has read in your dataset correctly before performing any analyses on the data.

```
proc print data=a1;
run;
```

If you highlight this section of code and click on the run button, you’ll see the dataset in the output window as follows:

Obs	gender	age	weight
1	M	13	143
2	M	16	132
3	F	19	140
4	M	20	120
5	M	15	110
6	F	18	95
7	F	22	105

If you do not highlight a section of code, SAS will run all commands in the program editor window.

If you want to see only some variables in the data set, you could add a statement after the `proc print` line in the format of `var gender age`. This would generate output similar to that shown above except the weight variable would not be included. For example:

```
proc print data=a1;
var gender age;
run;
```

Proc univariate

This is one of the most important procedures for elementary statistical analysis. It outputs the basic statistics of one or more variables, and has optional statements to generate qqplots and histograms. Sample code follows:

```
proc univariate data=a1;
var weight;
qqplot;
histogram;
run;
```

The `var` statement is optional. Without this statement, a univariate analysis is performed for all numeric variables in the order they appear in the dataset.

Proc sort

`Proc sort` sorts the observations in a dataset by some variables in either ascending or descending order. For example:

```
proc sort data=a1 out=a2;
by gender;
run;
```

The observations of dataset a1 are sorted in ascending order, by default, of the variable gender, and the sorted data is saved in a dataset named a2. Without the `out=a2` option, the unsorted dataset named a1 will be replaced by the sorted dataset. You can also sort the observations in the descending order of some variable by specifying the descending option in the by statement, e.g. `by descending gender`. If you need to sort by more than one variable, you can list all the variables in the by statement. For example, `by gender age` will sort in the ascending order by gender, and then the observations with the same gender value will be sorted in the ascending order by the values of age.

Proc means

This procedure produces simple univariate descriptive statistics for numeric variables. It also calculates confidence limits for the mean, and identifies extreme values and quartiles. Here's an example for mean and its confidence limit calculation:

```
proc means data=a2 alpha=0.01 clm mean median n min max;
run;
```

The mean, median, sample size, minimal value, maximal value, and 99% confidence intervals will be computed for variables age and weight. The `alpha` option specifies the confidence level for the confidence limit, and the default value is 0.05. `clm` tells SAS to calculate the confidence interval of the mean. `n` is used to count the sample size. Since gender is a categorical variable, no mean will be computed for it.

If you have a lot of variables and you only want to calculate the mean for some of them, use the `var` option and list the variables after the keyword `var`. If you want the means of the variables by group, use the `by` option. For example,

```
proc means data=a2 alpha=0.05 clm mean;
var weight;
by gender;
run;
```

tells SAS to compute the mean and confidence interval of weight for each value of gender, i.e. male and female. If the `by` statement is used, the observations need to be sorted by the same variable before the `proc means` procedure. Note data a2, the sorted dataset, was used in our `proc means` example.

Proc summary

It computes descriptive statistics on numeric variables in a SAS dataset and outputs the results to a new SAS dataset. The syntax of `proc summary` is the same as that of `proc means`. An example follows:

```
proc summary data=a2 print;
var weight;
by gender;
output out=3;
run;
```

`Proc summary` will not run without either the print option or the output statement.

Proc corr

This procedure is used for calculating the correlation between numeric variables. In addition, the output will give the simple summary statistics for each numeric variable. For example, the Pearson correlation coefficient and its P-value can be computed.

```
proc corr data=a1;
var age weight;
run;
```

A correlation coefficient matrix is created:

Pearson Correlation Coefficients, N = 7		
Prob > r under H0: Rho=0		
	age	weight
age	1.00000	-0.43017 0.3354
weight	-0.43017 0.3354	1.00000

The correlation coefficient between age and weight in this example is -0.43017, and 0.3354 is the P-value for testing the null hypothesis that the coefficient is zero. In this case, the P-value is greater than 0.05, and the null hypothesis of zero coefficient cannot be rejected.

Proc glm

It performs simple and multiple regression, analysis of variance (ANOVA), analysis of covariance (ANCOVA), multivariate analysis of variance, and repeated measures analysis of variance. E. g.,

```
proc glm data=a1;
model weight=age;
output out=a3 p=pred r=resid;
run;
```

will perform a simple linear regression with `weight` as the dependent variable and `age` the independent variable. The predicted values of `weight` (the dependent variable) and the residuals are saved in a new dataset called `a3` using the `output` statement.

For multiple regression where you have more than one independent variable, simply list in the model statement all the variables on the right hand side of the equal sign with one space in between, e.g.

```
model weight=age height age*height;
```

where the term `age*height` represents the interaction between variables `age` and `height`.

In the case of ANOVA, a class statement is needed for categorical variables before the model statement. The following code is an ANOVA analyzing the effect of gender on weight. It tests whether the weight is the same for females and males.

```
proc glm data=a1;
class gender;
model weight=gender;
run;
```

Proc reg

Proc reg is a procedure for regression. It is capable of more regression tasks than **proc glm**. It allows multiple model statements in one procedure, can do model selection, and even plots summary statistics and normal qq-plots.

You can specify several `plot` statements for each `model` statement, and you can specify more than one plot in each `plot` statement.

```
proc reg data=a1;
model weight=age;
plot weight*age;
plot predicted.*age;
plot residual.*age;
plot nqq.*residual.;
run;
```

In the above example, a simple regression is performed with `weight` as the response and `age` as the explanatory variable. The `plot` statements request four plots: weight versus age, predicted values of weight versus age, residuals versus age, and normal qq plot versus residuals. `predicted.`, `residual.` and `nqq.` are keywords that SAS recognizes. Make sure you keep a dot after these keywords.

IMPORTANT NOTE: regular simple and multiple regression (described in **proc glm** and **proc reg**) use numeric variables only for both response variable and the explanatory variable(s).

Basic Options and Statements within the Procedures

What is an option or statement?

A statement is a command nested within the procedure commands that tell SAS a bit more about the procedure you want to perform or in some cases, allows you to make your analysis more specific. An option is something that even further describes a statement, or in some cases, it may also further describe a procedure. Some statements are necessary while others are optional.

The `var` Statement

In many of the above SAS procedures, a `var` statement is either required or may be needed if you are dealing with a large data set with many variables. For example, if you are using `proc corr` procedure (outlined above), you may want to tell SAS which variables in your dataset you are interested in obtaining correlations for. It would work as follows if you had three variables for which you needed to obtain the correlations:

```
proc corr data=yourdatasetname;  
var V1 V2 V3;  
run;
```

If you have a dataset with many variables, but you only want to check normality assumptions for a few of them, use:

```
proc univariate data=yourdataset;  
var response1 response2;  
run;
```

The `by` Statement

The `by` statement is required for the `proc sort` procedure. After using it in `proc sort`, you can then use it in other procedures such as `proc means`. For example, say you were interested in performing regressions of height on weight by gender. First, you would want to sort your dataset by gender as follows:

```
proc sort data=yourdataset;  
by gender;  
run;
```

Then, you can use the sorted data to obtain two separate regressions, one for males and one for females as follows:

```
proc reg data=yourdataset;  
model weight=height;  
by gender;  
run;
```

After running the above SAS commands, `yourdataset` is not ordered by gender. If you wish to keep a copy of `yourdataset` in the original order, use an `output` statement (see page 11).

Note: In `proc reg`, the `by` statement can be used without a prior `proc sort`.

The `class` Statement

The `class` statement tells SAS that you have a variable in your data set that is categorical. For example, if you had data from an experiment with 20 subjects where five subjects received treatment 1, five received treatment 2, five received treatment 3, and the final five received treatment 4, treatment would be considered a categorical variable, and thus must appear in the `class` statement of the `glm` procedure. The most common usage of the `class` statement for you will most likely be in the `univariate`, `means`, and `glm` procedures. It is required for the `glm`

procedure only if you have a categorical variable such as gender. The coding of the above example could look as follows

```
proc glm data=yourdataset;  
class treatment;  
model resp=treatment;  
run;
```

where `resp` is the response for each of the 20 subjects.

The `model` Statement

By now, you have already seen the `model` statement in a few of the above examples. The `model` statement tells SAS which model you would like to use for your data. The dependent or response variable always goes on the left side of the equals sign while the independent variable(s) come after the equals sign on the right. The above `glm` example shows how the `model` statement works. For the *procedure* statements you have learned thus far, the `model` statement is only required (and accepted) in the `glm` and `reg` procedures.

The `model` statement also supports many options in both `glm` and `reg`. For example, in the `glm` model statement, options exist for choosing the types of sums of squares and asking for confidence and prediction intervals. In `proc reg`, the `model` statement has options for these same things, plus many other options such as standard errors for the regression coefficients, step-wise regression and specialized regression diagnostics. An example of how to use options in the model statement is as follows:

```
proc reg data=yourdataset;  
model weight=height / stb;  
run;
```

(following the earlier example of weight and height). You must always use the forward slash to tell SAS that there are options coming after the `model` statement. You can use as many options as you need in one `model` statement, but just make sure that all of them are separated by one space. The option `stb` asks for the standardized regression coefficients.

The `means` and `lsmeans` Statements

Often in an analysis, once differences are found among groups, we would like to see exactly where those differences occur; this is done in SAS by the use of the `means` and `lsmeans` statements in `proc glm` or `proc reg`. Both the `means` and `lsmeans` statements can be used in conjunction with a variety of options. If you have no missing values in your data set, your design is a balanced one and you use no covariates, you can use the `means` statement. However, if missing values exist or there is an imbalance in your design, or you have covariates on your model, you must use `lsmeans` to obtain the proper means and comparisons. An example follows:

```
proc glm data=yourdataset;  
class treatment;  
model resp=treatment;  
means treatment / lines tukey bon;  
run;
```

The `means` statement will perform means comparisons for all four treatment groups in this case. The options `lines`, `tukey`, and `bon` are used. The `lines` option displays the means comparisons in a more readable format. The `tukey` and `bon` options correspond to *Tukey* and *Bonferroni* comparisons procedures, respectively. Many other options for different means comparison procedures also exist (i.e. `Dunnett(dunnett)`, least squared differences (`lsd`), `Duncan(duncan)`, `Scheffe(scheffe)`, `Student-Newman-Kuels(snk)`). When using the `lsmeans` statement, the syntax is a bit different.

```
lsmeans treatment / adj=tukey stderr;
```

When using `lsmeans`, you must use the “`adj=`” option to obtain Tukey and Bonferroni comparisons, for example. The `stderr` option gives the standard errors for the least squares (ls) means.

Options in the Procedures

Some options contained in the procedures come not in the `model` or the `means` statements, but directly after the `proc` statement. An example of this is:

```
proc glm data=yourdataset alpha=.05;
class treatment;
model resp=treatment;
means treatment / lines tukey bon;
run;
```

In this example, it becomes apparent that the “`data=`” option is really an option in the procedures statement. The `alpha=.05` option tells SAS that for any confidence intervals, significance testing, etc. you want an alpha of .05. (This option is such that any tests in the model statement, `lsmeans`, `means`, and any confidence intervals outputted with the output statement are performed at the .05 level).

Another useful example of options in the `proc` statement is with `proc univariate`. By using options in the procedures statement, you can obtain stem-and-leaf plots, normal probability plots, boxplots, and tests for normality.

```
proc univariate data=yourdataset normal plot;
var response1 response2;
run;
```

The `normal` option gives the Shapiro-Wilks test of normality, while the `plot` option produces the stem-and-leaf plot, boxplot, and normal probability plot.

Output Statements (used in many procedures)

How does the output statement normally work?

In SAS, all the outputs that appear in the output window are reports not files. So, we can see these reports but they are not actually saved in SAS. The basic function of the `output` statement is to create a new dataset containing both the information in the old dataset plus any new diagnostics or statistics that the procedure has created. For example, if you specify a dataset for your `proc reg` procedure, you may want to output that dataset along with predicted values and residual values. While some analysis results appear in the output window cannot be saved using the `output` statement, the tables and plots can be saved in other ways. Details can be found in section “IV. Miscellaneous SAS Issues” (See page 14).

Options for obtaining predicted values, residual values, and other statistics and diagnostics

This is how it works:

```
proc reg data=one;
model response=var1 var2;
output out=two r=res p=pred;
run;
```

So, now you have a data set named “two” which contains everything that dataset one contains, plus the predicted and residual values from your `proc reg` model. Now, you can make diagnostic plots as follows:

```
proc gplot data=two;
plot res*pred;
plot res*var1;
plot res*var2;
run;
```

These plots can help to assess normality, independence of observations, and constancy of variance.

There are many other options besides residual and predicted values depending on which procedure you are using for your analysis. By looking in the SAS help menu, you can find the keywords (e.g., for residuals, the keyword is just `r=`) for other diagnostics such as Cook's distance, standard errors, prediction, etc.

Another example of an `output` statement used with `proc univariate` statement:

```
proc univariate normal plot data=old;
var y1;
output out=new max=maximum min=minimum mean=mean;
run;
```

This will give the mean, maximum, and minimum values for `y1` in the data set "new". Note that `max`, `min`, and `mean` are how SAS recognizes that you are asking for these values. What comes after the equals sign (=) is whatever YOU choose to name that new value or variable.

How can I be sure that correct values and variables were output?

The best way to assess whether your output statement worked is to use the `proc print` procedure as follows (building from the univariate example above):

```
proc print data=new;
run;
```

This will print out all variables and values in your new data set.

How does SAS know which dataset to use?

If you are working with multiple datasets that you have output from multiple procedures (e.g., you have one data set that SAS created from a `proc glm` and another data set from a `proc reg`), you must always name the data set you wish to use, otherwise SAS will default to the most recently used dataset.

III. Working with Graphics in SAS

The two basic graphic procedures in SAS are `proc gplot` and `ODS graphics`. In Version 9.3, ODS graphics are by default on so when certain procedures are run, sets of graphs are automatically generated. If you don't want these plots you can type the command

```
ods graphics off;
```

Proc gplot

`Proc gplot` has more options and can produce fancier, color graphics. The basics to know about `gplot` are how to choose symbols and how to draw regression lines. The following example will introduce you to a few of the options in `gplot`.

```
symbol value=circle i=r ci=red cv=blue;
proc gplot data=new;
plot y*x1;
run;
```

The `symbol value` statement has many other options other than circle (e.g., triangle). The `i=r` statement draws the linear regression line and gives the linear regression equation (in the log window, not the output window). The

`ci=red` option makes the regression line red and the `cv=blue` makes the plotted points show up as blue. The remaining statements are similar to `proc plot`. Also you can specify multiple different symbol statements and let them be represented in the same graph, like:

```
symbol1 value=circle i=r ci=red cv=blue;
symbol2 value=plus i=r ci=black cv=blue;
symbol3 value=star i=join ci=yellow cv=green;

proc gplot data=diag;
plot res*x1=1 res*x2=2 res*x3=3 / overlay;
run;
```

As example shown above, we can also specify the joint line instead of linear regression line between each point. So the graph between residuals and `x1` will be drawn with `symbol1`. `Proc gplot` also has the capability to overlay plots and many other options for adjusting axes values, changing colors, changing the legend, etc. which can be found in the help menu.

Exporting graphs

Often, it is helpful to export SAS graphics to a Word and/or a Power Point document. Graphs export best from `proc gplot`, but it is also possible to export graphs constructed with `proc plot`, but they may not look as nice in the Word or Power Point documents. There are many different formats in which to save graphs and many options for exporting graphs. The ones presented here are in no way exhaustive of all options. Sometimes, it just takes trial and error to find the best way to export a graph from SAS.

Exporting graphs/plots to Word:

- 1) From `proc gplot`, click in the graph you wish to export, pull down the Edit menu, and click on copy. Then, go into Word, pull down the Edit menu, and click on “paste special”. Use the option “Picture” to paste the graph. This is probably the simplest way.
- 2) Click on the graph you wish to export, pull down the File menu, and go to “Export as Image”. You can choose a variety of different formats in which to save the graph. After you choose your format and save it, go to the Word document. Pull down the Insert menu, click on “object”, click on “from file”, and put the pathname where your file is located. Note: some file formats may be difficult to export.

For both of these ways, using the `gplot` options first to control the size of the graph may produce better results, although you can size the graph somewhat once it is in Word.

Exporting graphs/plots to Power Point:

- 1) Save the graph first as a Bitmap file (.bmp) by going to the “Export as Image” as described above. Then, go to your Power Point document and choose the blank slide format. Pull down the “Insert” menu, and go to the “picture” option, and then to the “from file” option. Browse to find your file and then click “insert”. This will fit the graph nicely to the slide size.
- 2) The same process can also be achieved by using the “paste special” and “picture” or “bitmap” options as described above for the Word documents.

IV. Miscellaneous SAS Issues

Saving files

Now you are familiar with program editor window, log window, and output window. If you want to save the work you’ve done in a session, you’ll need to save the contents of each window separately. Usually, you only need to save the program; you can always run the program to get the log and output. To save a program file, you’ll need to first make sure the program editor is the active window by clicking in that window, then go to file and select the “save” or “save as” command. Similarly, you can save a log file when a log window is active, or an output file when the output window is active.

To save the result showing in output window, you can right click to get the menu, go to the “file” option, then go to “save” option, choose the location you want and click “save”. The file will be saved as list file(.lst), which can be opened in SAS.

There is no direct way to save the output result in other format. If you want do this indirectly, here we introduce two methods. First is to copy the output you need and then paste it into word file or text file. Second way is to save the output window as an image. When the output window is active, press “Alt” and “Print Screen” on the keyboard at the same time, then paste it into word file. Also you can paste the result in Paint tool, then edit the image and save it as .png or .jpeg format. If you are using a laptop, to print the screen you need to press “Fn” and “Print Screen” together instead of “Alt” and “Print Screen”.

You do not have to run the entire program every time you make a correction to your SAS program. Each SAS procedure is relatively independent of other procedures. As long as you have the dataset you need in this procedure in SAS, you can run only part of the program by highlighting the part of the program you want to run and then clicking the run button in the tool bars.

Missing values

- In SAS, a numeric missing value is represented by a dot or single period (.), character missing value is represented by a single blank enclosed in quotes (' '), and special numeric missing values are represented by a single period followed by a single letter or an underscore (for example .a, .b). If your data set has missing values, you’ll need to specify them as dots or blank enclosed in quotes in the SAS dataset.
- What if data set does not have dots (or blanks)? You can add a dot to the corresponding missing value locations within a data step. For example, if you have two variables, X and Y, in your data set, and 10 observations. The ninth value of Y is missing. The following code with an `If` statement will do:

```
data a2; set a1;
If _n_ eq 9 then Y=.
```

- When importing an Excel data file, SAS will automatically recognize missing values that are blank (i.e. empty cells) and replace them with dots or blank enclosed in quotes.

Other Issues

- Reading in data @@. You’ve already learned that when you input your dataset after a `cards` or `datalines` statement, every observation needs to be on an individual line. In case you want to make better use of the window and want to have more than one observation per line, @@ is the syntax that tells SAS where the end of one observation is. For example:

```
data b1;
input x y z @@;
cards;
1.1 2.2 3.3 4.4 5.5 6.5
;
```

It may be that your variables are data strings instead of numbers, for example gender or disease type. We call these variables categorical. In this case, SAS wants you to specify which variables are categorical by adding a \$ sign right after the name of the variable in the input statement. Sample code follows:

```
data b1;
input state $ county $ name $ gender $ weight;
cards;
indiana tipp brown female 125
;
```

- b. What if my Excel data file is not reading properly into SAS or not at all? If the Excel data file is not reading into SAS at all, most likely it's because your Excel data file is open. The Excel file must be closed before you import it into SAS. There are other reasons that the Excel data file is not reading in properly. It could be that the data type of your Excel cells is not correctly defined. Inappropriate reading also happens when you do not have a header in the first row, since the import procedure takes the first row as header by default. However, this can be changed during the import procedure under options.

How do you know if SAS is reading your dataset correctly? Use the `proc print` procedure and see if the dataset in SAS is what you expect.

Exporting to Excel, Access, or SPSS (.txt, .xls, .xlsx, .sav)

Exporting a data set to another program is the reverse of the import process. If you go to "File" and then select "Export Data", an export wizard window pops up. Then just follow the wizard through the following steps.

Step 1: Choose a data set that you created in the WORK library (where the SAS datasets are stored automatically by SAS). Click next button when you are done.

Step 2: Choose the file type you want to export to. Available types include Excel, Access, dBase, delimited file, and many others. Then click next.

Step 3: Type in the directory path where you want to save your data file. If you are not sure of the path, click on the browse button and find the location. Then click OK. If exporting to Excel, the wizard will ask you to assign a name to the exported table. This name will appear as the Sheet name tab at the bottom of the Excel workbook. At this time, you may click on the FINISH button. This method is similar to the second method of importing data mentioned in the Introduction chapter on page 2 of this document.

How to use the help menu

The SAS help menu is helpful if you want to improve your knowledge of SAS procedures. There are two ways of getting SAS help. One is to go to the help menu and choose one of the options listed. Under *SAS Help and Documentation* you can go to the *Index* tab or *Search* tab and type in the name of the procedure or other keyword. You can also work through the SAS tutorial provided in the *Help* menu under *Learning to use SAS*. The tutorial is also available through a pop-up window that appears when you launch a SAS session.

There are many other sources for help with SAS online and in printed sources. For Purdue users of SAS, you can visit the Statistical Software consulting desk. See:

http://www.stat.purdue.edu/scs/help/software_consulting_schedule.html for availability and location.

The color coding

When coding in Editor window, SAS will use different colors to distinguish different parts in SAS syntax. It's easy for you to check if the code is correct. Here are some rules:

Color	Explanation	Example
Black	General part of codes	Variable names, dataset names
Dark Blue	Procedure names	data, proc sort, proc glm
Green	Number	See in the following example
Light Blue	Statements and options in procedure	Statements and options in procedure
Green	Comment	/* comment */
Purple	Quote	See in the following example
Yellow highlighted area	Data input when using "datalines" or "cards"	See in the example on page 4

Here is a SAS code example:

```
data a1;
infile 'C:\Users\hp\Desktop\CH16PR11.txt';
input y machine no @@;
run;

proc glm data=a1;
class machine;
model y=machine / clparm alpha=0.01428;
estimate 'd1' machine 1 -1 0 0 0 0; /*compare the first two levels of
machine*/
run;
```