Introduction to SAS – STA2453H

What is SAS?

SAS is a software package, which is used primarily for data analysis. It has the capability to perform a wide range of statistical functions, but is not limited to statistics alone. SAS is also equipped with graphical tools, data management tools, and more, and has a wide range of possible uses. For this reason, it is widely used in many organizations today, and SAS programming is a valuable skill to acquire.

SAS can be thought of as a programming language. Many new users find it to be unfriendly and difficult to work with, but once they become familiar with the basics, it becomes quite simple.

Today's lecture will cover some of the preliminary things you will need to know about the SAS System before you can begin to program. If you are interested, the Information Commons at Robarts Library will also be conducting some short courses in SAS this year. Check their website at www.utoronto.ca/cat/whatson/stats.html for more details. Or ask me about them. I teach them.

Starting a SAS Session

The easiest way to write a SAS program is to create it first using a text editor (such as pico), and to submit it as a batch job. As an example, suppose we have written a program called reg.sas . To run this program, we would type:

```
sas reg.sas
```

at the unix prompt.

After a few moments, two or more files will be created. These are your log and output files. They will have the same name as your program file, but will be given the file extensions .log and .lst respectively. If your program involves the creation of graphs, other files will be created as well.

The output file will contain the results of your analysis. If there are errors in your program, you may not have any output. However, just because some results have appeared in the file, it does not necessarily imply that your program ran correctly.

This is what the SAS log is for. Every time you run a SAS job, the system writes messages in this file. It is critical that you read through this whenever you execute a program, so that you can ensure that your program had the results you had intended. This window will contain each statement of your program, followed by SAS comments upon executing the statement. When something goes terribly, terribly wrong and SAS cannot perform a certain task, it will contain error messages. If something dubious happens in your program that may not be correct but that does not prevent the program from being executed, warning messages will appear in this file.

** IT IS VERY IMPORTANT TO READ THROUGH YOUR LOG AFTER EACH SAS JOB **

SAS Language

Each statement in your program must end with a semi-colon. It does not matter if the command is broken over more than one line. SAS does not recognize the end of a statement until it reaches a semi-colon. For example:

```
proc print data=mydata;
run;
```

will produce the same results as

```
proc print
  data=mydata;
run;
```

There are two basic types of SAS program steps: data steps and proc steps. Data steps are used to create or modify a SAS data set. Proc (procedure) steps are used to process the data set. For example, reading in a text file would be accomplished through a data step, whereas performing a logistic regression would be done through a proc step.

A Few Words about Style

Proper programming style is highly encouraged in this course for a number of reasons. It makes your programs easier for other people to understand, as well as making it easier to de-bug your programs when things aren't working properly.

Good programming style includes the following:

1 Comments embedded within the program detailing the purpose of each step.
In SAS, you can include comments on a separate line by starting the line with a * and ending with a semi-colon.
Alternatively, you can include comments within your coding by surrounding the comments with a /* …. */.

2 Each statement in SAS should be contained on a separate line. You should include a blank line between each data or proc step in your code. Within a given step, it is highly recommended that you indent statements in a logical manner (i.e. after the initial line, and within if loops). For example:

** create a new data set transforming variables in temp1 based on the value of a;

```
data temp2;
  set temp1;
  if a=1 then do;
    newvar=oldvar*2;
    oldvar=oldvar+1 /* add 1 to the value of oldvar */;
  end;
  else do;
    newvar=oldvar/2;
  end;
run;

proc print data=temp2;
run;
```

3 When producing output, it is a good idea to include titles, footnotes etc. The commands for these options can be embedded within your proc steps. For example:

```
proc freq data=temp3;
  title ‘Cross-Tabulation of Age by Sex’;
  footnote ‘File=temp3’;
  table age*sex;
run;
```

SAS Data Steps

These steps are used to create a new SAS data set, either from an existing data set or from a raw data file. Default data sets are temporary and are deleted upon completion of your SAS session. To create a permanent data set, you will need to define a SAS library in which to store your data. This is done through the libname command:

`libname mydata ‘/u/consult/temp’;` or, more generally,
`libname <library name> ‘<directory path>’;`

If you wanted to save a SAS data set named data1 in this directory, you would name it mydata.data1.

Reading Raw Data:

There are two ways to create a SAS data set from raw data.

The first method involves directly entering the data into the program. The syntax is as follows:

```
data <filename>
  input var1 var2 $ var3;
/* lists the names of the variables you wish to enter into the data set.
Note that a $ sign follows var2, this is necessary because it is a
character (i.e. non-numeric) variable */;
  cards;                /* This statement tells SAS that the following
                           lines contain the raw data */;
  1 John 2
  1 Mary 5
  1 Ed 10
  0 Joe 5
;
/* note that each record is on a separate line, and the final line of
input contains only a semi-colon */
run;
```

The other means of entering raw data into a SAS data set is more commonly used. This involves reading in a pre-existing file and saving it in SAS format. The syntax is as follows:

```
data <filename>;
  infile '/u/consult/mydata.txt' <options>;
  input var1 var2 $8. var3;
run;
```

<u>Other Issues in Reading Raw Data:</u>

<u>COLUMN INPUT:</u>

Consider the following example data set:

Var	Address	Postal
111	133 Main Street, Toronto, Ontario	M1P 1T1
222	200 Elm Street, Toronto, Ontario	M2L 2X2

Here, each variable is separated by spaces, but always begin at the same place in each record. Files of these types can be read in as follows:

```
data temp;
  infile ‘u/consult/temp/mydata.txt’;
  input var 1-5 address $ 6-40 postal $ 41-47;
run;
```

The numbers after each variable name in the input statement indicate the columns in which this variable can be found. The $ indicates that the variable is character.

RESTRICTING INPUT RANGE:

Among the available options in the infile statement, the firstobs and obs commands are very useful. Firstobs= tells SAS which line to begin reading on, and obs= tells when to stop reading. If the datafile contains 5 lines of description before the data begins, then you could use the option firstobs=6 to begin reading at the correct place. If you wanted to read in 5 observations, beginning on line 6, then you would also use the obs=11 option (note that the number of observations read equals obs-firstobs).

DELIMITERS:

A delimiter is a character which separates variable values. In the above example, each variable is separated by a space in the raw data file. This is the default for SAS. However, if your data is not separated by spaces, you may use the DLM option to cope with this. For example, the dataset

```
1000,apple,book
28.5555,banana,chair
```

uses commas as delimiters and can be read in as follows:

```
data temp2;
  infile ‘/u/consult/comma.txt’ dlm=’,’;
  input var1 fruit $ var2
run;
```

If two consecutive delimiters are present in the data, SAS assumes that the variable in between them is missing. For example, the data set:

```
1000,,apple,book
28.5555,banana,chair
```

would be read in as:

VAR1	VAR2	VAR3
1000		apple
28.5555	banana	chair

If you do not want two consecutive delimiters to indicate a missing variable, then use the DSD option, which will then treat these two as a single delimiter, resulting in:

VAR1	VAR2	VAR3
1000	apple	book
28.5555	banana	chair

<u>Multiple Observations per Line or Multiple Lines per Observation:</u>

If more than one observation exists on a line, use the @@ option at the end of the input statement. This tells SAS to hold the line and continue to read the data.

eg.

```
data temp;
  input a b c @@;
  cards;
  1 1 1 2 2 2 3 3 3
  4 4 4
;
run;
```

would result in the dataset:

a	b	c
1	1	1
2	2	2
3	3	3
4	4	4

If a single observation is spread across more than one line, the @ option will hold the current record until all variables have been read in.

For example:

```
data temp2;
  input a b c @;
  cards;
  1 1
  1
  2 2 2
  3
  3
  3
  4 4
  4
;
```

Would again give us:

a	b	c
1	1	1
2	2	2
3	3	3
4	4	4

These two options can be used together if you have a really ugly dataset!

Creating New Variables:

To modify existing variables or to create new variables, simply tell SAS within the data step what you want to do. Some examples are:

```
data olddata;
  input a;
  cards;
  0
  0
  1
  1
  2
;
run;



data newdata;
  input olddata; /* we assume olddata is a previously
existing SAS data set */;
  a = a * 8;
  if a > 10 then do;
     b=1;
     c='Yes';
  end;
  else do;
     b=0;
     c='No';
     d=100;
  end;
run;
```

Our final data set appears as follows:

A	B	C	D
0	0	No	100
0	0	No	100
8	0	No	100
8	0	No	100
16	1	Yes	

Note that D is missing for the last observation, since our code does not assign a value to D in the case where a>10.

Combining Data Sets:

To combine two or more data sets into one larger data set, you may use the set command, as follows:

```
data file4;
  set file1 file2 file3;
run;
```

Suppose our files contain the following data:

File1	File2	File3
1 0	2 0	3 0
1 1	2 1	3 3

The set command described above would result in the following file:

File4
1 0
1 1
2 0
2 1
3 0
3 3

Another useful command to combine data sets is the merge command. Consider the following example:

Data1		Data2	
ID	Age	ID	Grade
111	20	222	80
100	26	222	75
222	35	100	90

```
proc sort data=data1;
  by ID;
run;

proc sort data=data2;
  by ID;
run;

data data3;
  merge data1 data2;
  by ID;
run;
```

This would result in the following file:

Data3

ID	Age	Grade
100	26	90
111	20	.
222	35	80
222	35	75

Note that grade is missing for ID 111, since this record was not contained in Data2. Also note that there are two instances of ID 222 in the new file, since it was duplicated in Data2.

Warning: Merging data sets is often useful, and works well when you are combining data sets in which the ‘by’ variable is one to one or one to many. However, if the by variable is replicated in the first data set named in the merge statement, the merge command leads to unpredictable results. In these cases, it is better to use proc sql.

SAS Proc Steps

As a general rule, the format of proc steps is as follows:

```
proc <procname> data=<filename> / <options>
  var <list of variables>
run;
```

A few of the more commonly used procedures are described below:

Proc Univariate

Produces summary statistics for a given set of variables. The general form is:

```
proc univariate data=<dataset> <options>;
  var <var1> <var2> … <varn>;
  by <optional by variables>;
  freq <optional frequency variable>;
  weight <optional weight variable>;
  output <output dataset, if desired> <list of statistics to
be written out to data set>;
run;
```

Some useful options you may wish to include in the proc statement include:

OPTION	DESCRIPTION
Normal	Tests the assumption that the data are normally distributed
Plot	Produces a stem-and-leaf, box plot and normal probability plot

Proc Plot / Proc Gplot

The general format is:

```
proc gplot data=<filename> <options>;
  plot <yvariable>*<xvariable>;
run;
```

This procedure is used primarily for scatterplots of x and y. Several other types of plots are available through proc capability.

Proc Capability

```
proc capability data=<filename> <options>;
  cdfplot <variables> / <distribution options>;
  histogram <variables> / <distribution options>;
  ppplot <variables> / <distribution options>;
  probplot <variables> / <distribution options>;
  qqplot <variables> / <distribution options>;
<other optional statements>;
run;
```

Not all plot statements are necessary, just choose the ones you want. The plot statements produce the following types of graphs:

OPTION	DESCRIPTION
Cdfplot	Cumulative distribution function plot
Histogram	Histograms of selected variables
Ppplot	Probability-probability (percent) plots
Probplot	Probability plots
Qqplot	Quantile-quantile plots

The distribution options can be used to compare your plot to a theoretical distribution. For example:

```
proc capability data=tmp1;
  cdfplot x / normal(mu=0 sigma=1);
run;
```

Will create a cdf plot comparing the distribution of x to a N(0,1) distribution.

NOTE: To display a more detailed plot on your monitor, use the graphics option in the proc statement. If you do not select graphics, your graph will appear in the output window, along with other statistics. By selecting graphics, the chart will appear in a new, gchart window. These charts are easier to read, but rather difficult to print out from unix. When using proc capability, run the procedure twice, once with the graphics option and once without. You can then print out the results from your output window.

In addition to plots, proc capability also produces a number of statistics, similar to the output of the univariate procedure.

Proc Freq

This procedure will create frequency tables on a given data set. It also has the capability to perform some statistical testing of tables of data, such as chi-squared tests and Fisher's exact test.
The general form is:

```
proc freq data=<dataset>
  tables <var1> <var1*var2> <var1*var2*var3> / <options>;
  format <variable> <format>.;
run;
```

You may include as many tables as you wish in a single ‘tables’ statement. The above commands would produce a one-way, two-way and three-way frequency table.

Several options are available, including the following commands:

Option	Result
Missing	Includes missing values in the frequency tables
Chisq	Performs chi-square tests of independence on the contents of the table
Exact	Performs Fisher’s exact test
All	Performs all tests available in SAS for analysis of frequency tables
Alpha=p	Alters the p value to be used in any testing. The default = 0.05
Out=<newfile>	Allows you to save the output to a new SAS data set

Several other options are available, refer to the SAS Stat manual or help files if you are interested in learning more about these.

Proc Means

This is a good procedure to help describe the distribution of one or more variables, as well as to produce descriptive statistics for your data. The format is as follows:

```
proc means data=<filename> <options>;
  var <variable list>;
run;
```

A CLASS or BY statement may also be included, which will perform separate analyses for each level of the variables specified in these statements. Options include:

OPTION	DESCRIPTION
N	Number of non-missing values
Min	Minimum value
Max	Maximum value
Sum	Sum of values
Mean	Mean value
Median	Median value
Stderr	Standard error of the mean
Stddev	Standard deviation

If you want to save the statistics to a data set, you must include the following statement in the body of the procedure:

```
output <newvariable name>=<option>(<oldvarname>) out=<output
dataset filename>;
```

For example,

```
output num=n(var1) sum1=sum(var1) out=newdata;
```

would result in the variables num and sum1 being written to the new data set newdata.

Proc GLM

This is useful for most types of linear models. GLM can be used for ANOVA, regression, MANOVA, etc. The basic form is as follows:

```
PROC GLM DATA=<DATASET>;
  BY <VARIABLE LIST>;
  CLASS <LISTING OF CLASS VARIABLES TO BE INCLUDED IN
ANALYSIS>;
  MODEL <RESPONSE(S)> = <INDEPENDENT VARIABLES> / <OPTIONS>;
  RANDOM <LISTING OF RANDOM EFFECTS>;
  REPEATED <FACTOR NAME>;
  FREQ <VARIABLE REPRESENTING FREQUENCY OF OBSERVATIONS>;
  ID <IDENTIFICATION VARIABLE LIST>;
  OUTPUT <OUT DATA SET> KEYWORD=NAMES … KEYWORD=NAMES;
RUN;
```

The first statement calls the GLM procedure, and indicates the name of the dataset to be used in the analysis.

The BY statement is optional. When used, a separate analysis will be performed for observations taking on different values of the BY variable. For example, if we select GENDER as a BY variable, separate models will be built for males and females. Note that to use the BY statement, the data must first be sorted using PROC SORT.

The CLASS statement is necessary only if classification (or categorical) variables are included as independent variables in your model. All classification variables which are incorporated into the model must be listed in this statement.

The MODEL statement is the one which defines the model you wish to build in order to carry out an analysis. The form of this statement can vary, depending on the type of analysis being run. Several options are available, including NOINT which requests that an intercept term not be included in the model. Some examples are given in the following sections.

The RANDOM statement is necessary only if random effects are included in the model. Any random effects which are listed in the MODEL statement must also be specified on this line.

The REPEATED statement is used when the response variables represent repeated measurements on the same experimental unit. For example,

```
MODEL Y1 Y2 Y3 = TREATMENT;
REPEATED TIME;
```

would indicate that our three responses reflect measures taken at time1, time2 and time3.

The FREQ statement is used in cases where some variable represents the frequency of an observed outcome (e.g. Poisson regression).

The ID statement affects only the format of your output. If you specify PATIENT as an ID variable, for example, than the PATIENT name will be printed beside each observed, predicted, and residual value.

The OUTPUT statement is used when you wish to save the results of your analysis to a new data set. For example, suppose we wish to save the residuals from our model to a data set called DATA2. This statement would then read:

```
OUTPUT OUT=DATA2 residual=RES;
```

Our new data set would include a variable called RES, which contains the residual values from our model. A complete listing of the keywords can be found in the SAS STAT manual.

Other statements are available, including CONTRAST, MEANS, and other statements, but are not necessary in order for an analysis to be run. Descriptions of these statements will be given later in this document.

Proc Reg

Used for linear regression. The basic syntax is as follows:

```
PROC REG < options > ;
MODEL dependents=<regressors> < / options > ;
VAR variables ;
PLOT <yvariable*xvariable> <=symbol>
      < ...yvariable*xvariable> <=symbol> < / options > ;
RUN;
```

Further options are available as well, but will not be covered in these notes. Please refer to the SAS STAT manuals, or ask me about them.

Proc Boxplot

This procedure is new to version 8 of SAS, and enables you to easily produce side-by-side boxplots. The general syntax is as follows:

```
PROC BOXPLOT < options > ;
PLOT analysis-variable*group-variable < (block-variables ) >
< =symbol-variable > < / options > ;
BY variables;
ID variables;
RUN;
```

Note that your data must contain two or more groups which you wish to compare, in order to carry out this procedure.

Sample Programs

Creating a Permanent Data Set

```
options pagesize=120 linesize=80;

libname mydata 'c:\My Documents\My SAS Files7';

data mydata.demo1;
  length first $9. last $15.; /* the default length is 8, so some names
would be cut off
                                 if we did not reset the length */
  input first $ last $ colour $ age gender;
  label first ='First Name'
        last  ='Last Name'
        colour='Favourite Colour'
        age   ='Age of individual'
        Gender='Gender of Individual'
        ;
  cards;
  John Smith blue 25 0
  Andrew Jones orange 43 0
  Mary Andrews purple 38 1
  Anastasia Reynolds red 21 1
  Susan Wilson blue 49 1
  Jason Chun green 19 0
  Laurie Wagner red 55 1
  Chris Foster yellow 46 0
  Richard Thomas black 60 0
  Pam Bentley blue 31 1
  Martha Porter orange 34 1
  Debbie Graham blue 43 1
  ;
run;
```

Using the Infile Statement to Read Raw Data

```
data demo2;
  infile 'a:/textfile.txt' firstobs=14 dlm=',';
  input first $ last $ age subject $ mark1 mark2 exam @@;
run;
```

Modifying Data Sets

```
data mydata.demo2 (keep=name age_gp);
  set mydata.demo1 (drop=colour rename=(first=first_name
last=last_name));
  length name $ 50;
  name = last_name!!','!!first_name;  /* the !! command indicates
concatination.  Here
                                  we've included a comma in between last
and first
                                        names, resulting in Smith,John
for example. */
  if age ^=. then do;
        if       age<20 then age_gp ='Under 20';
        else if age <40 then age_gp ='20-39';
        else if age <60 then age_gp ='40-59';
        else                 age_gp ='60+';
  end;
run;
```

Examining Your Data

```
proc contents data=mydata.demo1;
  title 'Contents of Demo1';
run;

proc format;
  value sexfmt 0='Male'
               1='Female';
run;

proc print data=demo1;
  format gender sexfmt.;
run;

proc freq data=demo1;
  table gender*colour;
  format gender sexfmt.;
run;
```

Combining Data Sets

```
** sort each of the data sets by name;

proc sort data=demo1;
  by last first;
run;

proc sort data=demo2;
  by last first;
run;

data temp3;
  merge demo1 (in=a) demo2 (in=b);
  by last first;
  if a or b;  /* includes any records found in either of these files */
run;

data temp4
  merge demo1 (in=a) demo2 (in=b);
  by last first;
  if a and b  /* includes only the records common to both files */
run;
```