From: Eric So
Re: A Brief Introduction to SAS
Date: 7/29/09

Section 1: A Few Beginning Notes

1. SAS is a powerful software package that uses its own programming language for data processing and statistical analysis. Programming in SAS requires an understanding of the basic structure of the syntax as well as the many built-in procedures that make data manipulation and analysis easier than coding from scratch in matrix notation. Learning the language can be initially cumbersome but this upfront cost is rewarded with a seemingly endless list of procedures that greatly simplify statistical analysis.
2. A few quick notes:
   (a) A SAS program consists of a sequence of commands executed in order of their appearance in the program. Consequently, the ordering of commands is particularly important.
   (b) Extremely Important: every SAS command ends with a semicolon: ;
   (c) SAS is not case sensitive.
   (d) Statements can run over to the next line as long as a semicolon separates each distinct command.
   (e) You can comment out a line by adding an asterisk at the beginning. Example: `*this line is not read by SAS;`
   (f) You can also comment out a section by bracketing the section with /* and */ Example: `/* this line is also not read by SAS;*/`
   (g) SAS reads and processes datasets row by row rather than using an entire dataset at the same time. This is important to keep in mind when structuring one's code. For example, if you write a program that manipulates the time series of Microsoft's daily returns in the month of December by (i) adding 1 to each daily return, (ii) subtracting the market return, and (iii) taking the log, SAS will first do all three operations on Microsoft's December 1st return, then December 2nd, and so forth.
   (h) There are two primary methods for running SAS code: (1) Interactively: using the SAS interface on your computer and (2) Batch Mode: submitting a SAS program to run "out of sight" (typically on a server such as Stanford's or Wharton's Unix server). Today we will focus on the former approach.
   (i) SAS code can be executed using either an entire program or individual sections so long as each section is self-contained.
   (j) The SAS system is comprised of five main sections:
      (i) Explorer Window: allows you to see libraries of existing datasets
      (ii) Program Editor: allows you to write and submit lines of code
      (iii) Log: allows you to see notes regarding submitted processes. This will contain confirmations, errors, and/or warnings as well as the program statements that you have already submitted.
      (iv) Output: allows you to see printable results & specific details on the results of submitted code
      (v) Toolbox/Toolbar: allows you to enter additional option commands

Section 2: Importing Data

1. It is particularly helpful for researchers in Accounting, Finance, and Economics to learn SAS since all of the major datasets available from the Wharton Researcher Data Services (WRDS) are stored natively in the SAS format. This allows researchers to directly and easily access SAS datasets using a variety of remote connections (note that the use of remote connections is somewhat advanced procedural and will be discussed in later sessions). The use of SAS to access the WRDS server dramatically simplifies exploratory data analysis by allowing the researcher to examine the structure and content of data sets without the use of the WRDS website or conversion of existing datasets into alternative formats.
2. The file extension for SAS datasets is sas7bdat. So for example a SAS dataset named Return would be listed as follows: Returns.sas7bdat
3. Datasets are organized using SAS libraries. A library is simply a file path or folder that contains the data that you hope to work with in SAS. There are no limits to the number of libraries that one can use at the same time. So for example, when working off of the AFS server, I can using the following commands to tell SAS where to look for data:
   ```
   libname a '/afs/ir/data/gsb/eso/';
   libname b '/afs/ir/data/gsb/eso2/';
   ```
   The first part of each line is the libname command, which tells SAS that you want to assign a library reference name (e.g. ‘a’, ‘b’) to particular folders. The second part of each line is the actual reference name of each library. The reference names are not limited to single letters (‘a’,’b’) and can instead be any string without spaces (e.g. libname ILoveSAS '/afs/ir/data/gsb/eso/'; also an acceptable library reference command while libname I Love SAS '/afs/ir/data/gsb/eso/'; is not).
   Suppose I stored the file Returns.sas7bdat in the directory /afs/ir/data/gsb/eso/ and another file Analysts.sas7bdat in /afs/ir/data/gsb/eso2/. Then I can access these datasets by telling SAS to find a.Returns and b.Analysts. Note that I do not include the file extensions within the SAS code. Of course, data file can be similarly stored on your personal computer and the process for setting a library is the same—simply set the path directory of the file’s location.
4. For today’s session, we will assume that you have your dataset in the SAS format. If this not the case, there are a number of additional options that you have available to you including the use of a Proc Import statement and conversion using a secondary program such as Stat Transfer (available on Stanford’s AFS server).
5. There are two major components of a SAS program:
   (a) DATA steps: (i) begin with DATA statements, (ii) read and modify data, (ii) create new SAS datasets. Example of situation requiring a DATA statement: create a data set that takes adds together two columns and takes the log of another.
   (b) PROC steps: (i) begin with PROC statements, (ii) perform specific analysis or function, (iii) produce results or reports. Example of situation requiring a PROC statement: generate descriptive statistics of Microsoft’s historical earnings; generate the mean, median, and standard deviation of earnings per share.

Section 3: Data Step Examples

1. First, a quick note about terminology: I will refer to a dataset's columns as variables and to a dataset's rows as observations.
2. A Basic Example:

```
*this tells SAS where to find data;
libname a '/afs/ir/data/gsb/eso/';

*this tells SAS what to call the new dataset;
data new_dataset_name;

     *the SET statement tells SAS which dataset to use;
     set a.old_dataset_name;

     *this next line will create a new variable called LogRet that takes the log of 1
     plus a variable called RET;
     LogRet=log(1+RET);

     *this next line will set LogRet to equal a missing value if it is greater than 1;
     if LogRet gt 1 then LogRet=.;

     *this next three lines will both create a dummy variable BigReturn if LogRet
     is greater than or equal to 0.02. Note that these commands accomplish the
     same thing and only one is necessary;
     if LogRet ge .02 then BigReturn=1; else BigReturn=0;
     if LogRet >= .02 then BigReturn=1; else BigReturn=0;
     BigReturn=(LogRet>=.02);

     *this next line will create a character variable (also called a string or non-
     numeric variable) that indicates the category of return;
     if LogRet lt -.02 then RetCategory='Low';
     else if -.02 le LogRet le .02 then RetCategory='Middle';
     else if LogRet gt .02 then RetCategory='High';

*Finally, we conclude the DATA step with a run command to tell SAS to execute the
above code;
run;
```

A few notes are in order:

(a) Note that the above code results in a new dataset called 'new_dataset_name'. Obviously, you can call this whatever you want. Since no library was specified for 'new_dataset_name' this file would be stored in your 'working' SAS directory, which SAS automatically creates for you. If you wanted to save the resulting dataset to the folder /afs/ir/data/gsb/eso/, you can simply rewrite the initial DATA command as: `data a.new_dataset_name;` Here the 'a.' appended to the file name tells SAS to store the file in the library referenced by the name 'a'.

(b) Recall that the comment lines that start with * and end with ; will not affect SAS's processes.

(c) To create a non-numeric variable such as `RetCategory` in the above example the resulting values must be placed within single quotes (e.g. 'Low').

(d) The operations =, >=, <=, >, <, and ~= used to denote equals, greater than or equal to, less than or equal to, greater than, less than, or not equal to (respectively), can, in general, be replaced by `eq`, `ge`, `le`, `gt`, `lt`, and `ne` (respectively). One of the few exceptions is when working with quoted strings such as a variable called `Univ_Name` containing 'Stanford University'. In this case, you would need to use `eq` instead of an equal sign such as: `if UNIV_Name eq 'Stanford University' then…`[1]

(e) Other commonly used operations: `x**2` produces x-squared, `exp(x)` produces the exponential of x, `sqrt(x)` results in the square-root of x.

(f) Extremely Useful: to examine the resulting dataset, type in the command `FSV` (no semincolon needed) into the Toolbar/Toolbox command line *after* the dataset has been created. This command opens the dataset for viewing the resulting rows and commands.

(g) Also Extremely Useful: to examine the characteristics of the resulting dataset (e.g. complete variable list, variable names, number of rows), type in the command `VARS` into the Toolbar/Toolbox command line *after* the dataset has been created.

3. Merging Data: Suppose you have two datasets that you want to join according to one or more variables. For example, suppose you two datasets: the first is a dataset containing Micrsoft's daily returns and a second dataset containing the daily returns of the S&P500. If you wanted to align the data by date and place them both into one dataset, you would need to know how to merge datasets.

```
*This first line specifies the name of the file to be created;
data Merged_Dataset_Name;

    /*The next line specifies the datasets that you want to merge. When creating
    a new dataset that is the result of a merge, use the command MERGE in
    place of a SET command within a datastep. You can specify more than two
    datasets to merge as long as you want all of them merged on the same
    variable (e.g. date). Appending the code (in=a) to the first dataset name
    provides SAS with a reference name for the dataset—this is particularly
    useful if you only want to keep observations that are present on one or both
    datasets;*/
    merge new_dataset_name(in=a) a.SNP(in=b);
```

[1] Note that while SAS is not generally case sensitive, the case of a phrase matters when matching a quoted string. So for example, the following two lines of code would not produce the same code:

```
if UNIV_Name eq 'Stanford University' then GOOD=1;
if UNIV_Name eq 'sTanFord univeRsity' then GOOD=1;
```

```
        *the next line tells SAS the variable upon which to merge. In this case, we
        will be merged by the date variable.
        by date;

        /*this next line is completely optional but tells SAS to only keep those
        observations that were successfully merged from BOTH datasets specified in
        the MERGE command. Alternatives include: (i) if a;, (ii) if b;, (iii) if a and
        ~b*/
        if a and b;

        *the next line tells SAS to create a logged version of the market return.
        LogMktRet=log(1+MktRet);

        *As usual, we end the statement block with a run command;
    run;
```

A quick note on preparing dataset to merge:

- Note that in order to merge the above two datasets, they MUST be sorted by the variable that you intend to merge it on. In the above example, both datasets must be sorted by the DATE variable. To accomplish this, one can use the following sort procedure:

```
proc sort data=<input dataset name> out=<output dataset name>;
        by <variable to be sorted by>;
run;
proc sort data= a.SNP out=SNP_sorted;
        by date;
run;
```

Section 4: Procedural Analysis

1. Procedures can be used to synthesize data and produce results or reports. Given that we have already seen how to use the SORT procesdure, we will discuss two other commonly used procedures: SUMMARY, CORR, and GLM (for regression).
2. The SUMMARY procedure results in summary statistics such as means, medians, and standard deviations of the input dataset. Consider the following code:

```
proc summary data= Merged_Dataset_Name;
        Class RetCategory;
        Output out=Summary_Dataset_Name
                mean(LogRet)=MeanLogRet sum(LogRet)=SumLogRet;
run;
```

Note the following:

- The above summary procedure results in the dataset Summary_Dataset_Name that contains the mean of the variable LogRET for each value of RetCategory.

- A class statement is not necessary if you want the mean of the entire sample
- The proc summary procedure also counts the number of observations for each unique value of the variable specified in the class statement
- The mean() function can be replaced, or supplemented, by other statistical functions such as median(), std(), max(), min(), and sum().
- Multiple summary functions can be used at the same time by augmenting the output line: Output out=Summary_Dataset_Name mean(LogRet)=MeanLogRet median(LogRet)=MediamLogRet std(LogRet)=StdLogRet;
- A common option used along with the summary procedure is NWAY:

```
proc summary data= Merged_Dataset_Name NWAY;
        Class RetCategory;
        Output out=Summary_Dataset_Name
                mean(LogRet)=MeanLogRet sum(LogRet)=SumLogRet;
run;
```

The NWAY option suppresses additional summary data incremental to the mean of each RetCategory (try adding this option and note the difference).[2]

3. The CORR procedure is used to produce a table of correlation coefficients among the variables in an existing dataset. The syntax of the CORR statement is relatively simple. The following code will produce a table containing correlations between LogRet and LogMktRet:

```
proc corr data=Merged_Dataset_Name out=Corr_Dataset;
        var logRet logMktRet;
run;
```

The above code produces a dataset named Corr_Dataset that contains both correlations among each variable listed in the var statement as well as descriptive statistics (i.e. means, standard deviations) of each variable

4. The GLM (short for Generalized Linear Model) procedure can be used to estimate basic linear OLS regression models. Of course, more advanced procedures are available but we will start with the basics. Consider the following example:

```
proc glm data= Merged_Dataset_Name;
        model LogRet=LogMktRet;
        ods output parameterestimates=OLS_Coeff_Dataset_Name;
run;
```

[2] Note that by summing logged returns, one can easily create cumulative holding period returns by taking the exponential of the sum and subtracting one. That is, cumulative returns can be obtained within a data set as: CumulativeRet=exp(SumLogRet)-1;

The above code produces a dataset named OLS_Coeff_Dataset_Name that contains OLS coefficients from regressing LogRet on LogMktRet. Of course, the glm procedure can handle multiple RHS variables so feel free to add as many as you would like.

Future References and Things to Try

- The absolute best way to learn to program in SAS is to practice on your own. Practicing on your own and learning from your mistakes is extremely important. Don't be discouraged by errors or mistakes. This is part of the learning process and is important for developing your understanding of some the nuanced aspects of SAS.
- The following website is particularly helpful and provides a comprehensive set of SAS references and tutorials (even some in video!): www.ats.ucla.edu/stat/sas Bookmark this webpage and make sure to spend time going through their archive. I am confident that you will find it extremely helpful and user friendly.
- Ask your colleagues for bits of code when you approach a particularly difficult problem. It is important to try to address the problem on your own but it can also be extremely helpful to learn from your peers and more advanced SAS programmers.