University of Washington
CSSS 506

Using SAS with UNIX

A Beginner's Guide

Mark S. Handcock
handcock@u.washington.edu

Table of Contents

INTRODUCTION

This document serves as an introduction to some of the basic principles of SAS.

SAS programs have two major building blocks:

1. DATA step

   The DATA step is used to describe and modify your data. In it you tell SAS how to read data; create variables; delete observations; change variable values; and limit analysis to relevant data.

   The DATA step transforms your raw data into a SAS dataset (if it is not already in one). SAS datasets are the common element between DATA steps and PROC steps: DATA steps create SAS datasets and PROC steps analyze SAS datasets.

2. PROC step

   The PROC step (or steps) tells SAS what analysis to perform on the data.

Some basic syntax rules for SAS program statements

All software has rules that must be followed in order for the program to work efficiently. These are the basic SAS rules that must be followed:

1. All SAS statements (a SAS "statement" is an instruction or command) must end with a semicolon (;).
2. Most SAS statements begin with a SAS keyword (e.g., DATA, PROC, BY).
3. SAS statements may be entered in free form in columns 1 to 80 (That is, SAS statements may cover multiple lines on the screen or a line on the screen may contain multiple SAS statements. But all statements must be separated by semicolons.)
4. A period (.) is used to represent a missing value
5. The asterisk (*) is used as a keyword to introduce a COMMENT statement, comments are ended with a semi-colon (;), as:

   ```
   * This is a comment that gives some information about the program *
   * Comments are ignored by SAS but are useful in following the logic *
   * of the program                                                    *;
   ```

   Comments are not required but help you or others follow the logic of the program. Comments are extremely helpful in debugging and finding program errors.[1]

6. Many things in SAS have to be given proper names, e.g., dataset names, variables, and formats. In general, a proper name in SAS must begin with a letter and can be no more than 8 characters in length. SAS names may use the alphabetic characters, the numbers 0-9, and the underscore character.

---

1. Comments in the PL/1 programming style, i.e. in the form /*.....*/, should be avoided in the Display Manager environment.

Reading data with the DATA step

There are two ways of getting a DATA step. You can create one using an editor or you can copy an already existing DATA step. Both of these methods have the same result: a complete DATA step describes your data to SAS.

Data to be analyzed

The data to be analyzed should be viewed as a column or matrix of numbers. A single column is appropriate if there is a single variable being analyzed. The matrix is a table-like arrangement used when the data has more than one variable.

A sample data matrix is shown below. The three variables (X,Y,Z) form the columns and the rows represent observations. If, in this example, the observations are computer users, then the first row of the matrix gives the scores of the first computer user on the three variables.

X	Y	Z
27	118	63
24	170	70
25	173	73
23	183	68
19	203	78

It helps to interpret the SAS results if the three variables are labelled with meaningful names. For example, if the data came from a study of the ages, weights, and heights of individuals, then it would be better to name the variables AGE, WEIGHT, and HEIGHT.

When entering data, type in the numeric values "row-wise." That is, the values of all the variables for a single observation will be entered on the same line. Variables are separated by spaces. For example, when entering the first observation's scores from the above matrix, you type:

```
27 118 63
```

It is very important that the same order be used for each observation. Here, the first number is the AGE value, followed by WEIGHT, then HEIGHT.

DATA step

There is more to the DATA step than just the data, however. Three SAS commands need to be included as well: (1) the DATA statement, (2) the INPUT statement, and (3) the CARDS statement.

The **DATA statement**

DATA *datasetname*;

The DATA statement must come first. It names the dataset. If you have several DATA steps during one session, the *datasetname* is used by SAS to distinguish among them. It's good practice to use a *datasetname* that somehow describes the data. It can be from 1 to 8 characters and it must begin with a letter. The DATA statement MUST end with a semicolon.

The **INPUT statement**

INPUT *variablenames*;

The INPUT statement is next and consists of the word INPUT followed by the names of the variables in the data. Each *variablename* can be from 1 to 8 characters and must begin with a letter. *Variablenames* are separated with a blank space. The INPUT statement MUST end with a semicolon.
For the example, a free form INPUT statement is:

```
input age weight height;
```

A second approach is to use column input. In the INPUT statement you specify the columns where each data value is found. For example, the column INPUT statement for this example is:

```
input age 1-2 weight 3-5 height 6-7;
```

In this case the data are not separated by spaces and the data line is:

```
2711863
2417070
2517373
2318368
1920378
```

A third approach is to use formats.[2]

2. See the SAS manuals for details.

The **CARDS statement**

CARDS;	The CARDS statement is next. It tells SAS that the actual data begins on the next line. The CARDS statement MUST end with a semicolon.

Combining these statements with the data, the entire input to SAS would be:

Free Form Example:

```
DATA newstudy;
INPUT age weight height;
CARDS;
27   118 63
24 170       70
25    173     73
23   183      68
19 203 78
```

Column Direct Example:

```
DATA newstudy;
INPUT age 1-2 weight 3-5 height 6-7;
CARDS;
2711863
2417070
2517373
2318368
1920378
```

Note that in the free form example the actual data values do not have to be "neat." All that matters is that the three values on each line be entered in the correct order and that at least one space separates them. In the column example the data must be in the correct column.

Sometimes the data already exists in a UNIX file, and so there will be no CARDS statement in the DATA step. In these cases you must use the SAS INFILE statement to tell SAS where to find the data. The following illustrates the use of a data file already existing in UNIX. That file is referred to in the SAS DATA step by the SAS INFILE command (note the use of comments):

```
*=========================================================*;
* The  INFILE statement tells SAS where  to find the data.  *;
*=========================================================*;
DATA  NEW;
    INFILE '/class/mhandcoc/data/mardiv.dat';
    INPUT age weight height;
```

Note: You must still include the INPUT statement to tell SAS the variable names of the data in the referenced file.

PROC step

The SAS procedure (PROC) step tells SAS what calculations to perform on the data. Every one of SAS's many PROC steps begins with a line containing the PROC statement. That may be followed by one or more lines further refining the procedure.

PROC *procedurename* DATA = *datasetname [option]* ;

Where *procedurename* is the name of the procedure to use and *datasetname* is the name of the dataset. Some procedures allow certain options to be specified as well (see the example below). If options are included they are typed one after the other with a space between them. The PROC statement **must** end with a semicolon.

additional statements;

Some PROCs require additional statements to further refine the analysis. These are described more fully when the associated PROC is described. All additional statements **must** end with a semicolon.

An example of a complete PROC step is:

```
PROC MEANS DATA = JUNIOR MEAN VAR SKEWNESS;
```

This PROC step computes the mean, variance, and skewness of the data contained in a dataset named "junior." Note the options listed in the PROC statement and the semicolon at the end of each statement. This example does not have any additional statements because PROC MEANS does not require them.

Some common SAS procedures are[3]

PROC name	Brief Description
CHART	Draws histograms, pie charts and other descriptive statistics.
CORR	Computes Pearson product moment correlation coefficient and other related statistics.
FREQ	Prints "contingency" tables of frequency counts. Computes percentages, expected frequencies, Chi square and other related statistics.
GLM	General Linear Model. Computes simple and multiple regression. GLM produces the equation of the best fitting line and all associated statistics.
MEANS	Produces a default set of statistics, or you can specify a selection of the following:
	CSS — corrected sums of squares KURTOSIS — kurtosis MAX — highest value MEAN — arithmetic mean MIN — lowest value N — same size NMISS — number of missing values RANGE — range of scores SKEWNESS — skewness SUM — sum of scores STD — standard deviation USS — uncorrected sums of squares VAR — variance
PLOT	Produces bivariate scatterplots
PRINT	Prints the observations in a dataset, using all or some of the variables.
REG	Computes multiple regression (includes SYSREG and STEPWISE from older versions).
TTEST	Gives descriptive statistics plus a t test.
UNIVARIATE	Gives many descriptive statistics, quartiles (called "quantiles" in SAS); boxplots and stem-and-leaf displays with the PLOT option

3. For detailed directions for each of these PROC statements, see the *SAS User's Guide: Basics* or *SAS System for Elementary Statistical Analysis.*

Temporary and permanent SAS datasets

While this is a slightly more advanced topic, it is a concept important to using SAS effectively.

The DATA statement names the SAS dataset to create. Normally SAS dataset names begin with a letter and are no more than eight characters. Using this approach creates a temporary dataset that exists only as long as the program runs.

It is possible to create a SAS dataset and permanently store it for future use. This has several advantages: (1) SAS datasets are compact and can be stored in less space than regular data files, thereby saving disk space; (2) permanent SAS datasets save execution time because SAS doesn't have to convert raw data into SAS format every time the dataset is used; and (3) finding data with PROCs is easier. One disadvantage is that a permanent SAS dataset cannot be directly edited with **emacs**.

To create a permanent SAS dataset use a two level name in the DATA statement when creating the dataset. This makes the SAS dataset a permanent file written to your UNIX directory. Examples of two level dataset names are NEW.DATA and REGRESS.ALLONE. When such files are stored on your UNIX directory, their names will appear to be reversed: the first level SAS name becomes the suffix and the second level SAS name becomes the file name. Thus, if the dataset is called NEW.DATA in the DATA statement it appears as DATA.NEW in your directory. An example of a SAS program which generates a permanent SAS dataset:

```
    *=================================================*;
    * This program generates a permanent SAS DATA SET  *;
    *  stored on your UNIX directory                   *;
    *=================================================*;
DATA REGRESS.CARS;                ⇐ refers to UNIX file: CARS.REGRESS
    INFILE CAR.DATA;
    INPUT YEAR NCARS PINCOME;
PROC PRINT DATA=REGRESS.CARS;
```

To use a permanent SAS dataset you only need to add a "DATA=" option to a PROC step, as shown in the following example:

```
PROC MEANS  DATA=REGRESS.CARS;      ⇐ using the "DATA=" option
```

To access and modify an existing permanent SAS dataset in a program, use the SET statement:

```
DATA NEW;                         ⇐ "NEW" is the DDNAME
    SET REGRESS.CARS;             ⇐  defines the dataset to use
    C=B+D;                        ⇐ data transformation statement
PROC PRINT DATA=NEW;              ⇐ using the "DATA=" option
```

RUNNING SAS ON UNIX

There are two common ways to run a SAS job on UNIX. In one, often called "batch processing," you use **emacs** to create a SAS program file containing all the SAS statements your job will require, and possibly all the data as well. Or, the program file may refer to another UNIX file containing the data the program needs to run. The second method takes advantage of a relatively new feature of SAS called the Display Manager.

Batch processing

Batch processing means that you have created a file with the suffix **.sas** that contains all the SAS commands necessary to complete the job. This file can have any file name but should have a suffix of **.sas**. Some examples are **myjob**..**sas** or **prog1**..**sas**.

To get this program to run, log on to UNIX and at the prompt give the command:

sales% sas *file name*

Where: *file name* is the name of your program file, for example **sas myjob** or **sas prog1**. This command sends a copy of the file containing the program instructions to be executed by SAS.

If the program runs successfully, two new files are created and stored on your UNIX directory. Both these new files have the same name as the program file, but one will have the suffix .**lst** and the other the suffix **.log**. For example, if you gave the command **sas myjob** (meaning, run the program stored as **myjob .sas**) successful completion produces the two files **myjob.lst** and **myjob .log** . The **.lst** file contains the results of the SAS analysis, and the **.log** file contains your program statements and notes from SAS about the running of the job. If the program did not run successfully, only the **.log** file appears. In this case it contains diagnostic information that will help you correct the errors.

Printing SAS files from UNIX

SAS program (**.sas**), log (**.log**) files and output (**.lst**) files created can be printed from UNIX with the command:

sales% saslpr *file name*

This command will print your program file on the default UNIX lineprinter 2-up on letter-sized paper.

It is not necessary to print the program, listing, and log files for a SAS program every time it runs. It is rarely necessary to print the log file; usually that is done only when there is a major problem. SAS listing files can use large quantities of paper. To avoid excessive paper usage, Academic Computing Services advises reviewing and editing your output files before printing them. The procedures in your program may generate many pages of output, but you may need only three tables out of all that output. Print wisely.

SAS PROGRAMMING STYLE RECOMMENDATIONS

The following recommendations for programming in SAS have been recommended by Research Consulting Services of Academic Computing Services. These are general guidelines to make your programs clearer and easier to follow. They will also make it easier for a consultant to help you if you have problems.

1. Begin your program with a header. Example:

```
  ************************************************************;
*          DESCRIPTIVE NAME OF THE PROGRAM                    *;
*                        AUTHOR                               *;
*                      (AFFILIATION)                          *;
*                                                             *;
*     DATE:                        VERSION:                  *;
*                                                             *;
  ************************************************************;
```

2. Reserve column 1 for DATA, PROC and OPTIONS statements. All other SAS statements should be indented.

3. Develop a system of indentation that works for you to identify DO groups. A common practice is to indent them 5 columns per level.

4. Continuation lines of multiline statements should be indented.

5. Use comments to explain what you're doing. Comment lines begin and end with an asterisk (*****).

6. Select meaningful names for variables, datasets, data fields, and the like.

7. End DATA steps and PROCs with a RUN; statement.

8. Never rely on the implicit numeric-char (or vice versa) conversions.

9. Never take default formats - always specify lengths.

10. Never use implicit RETAIN statements; say

    count = count + 1;

    rather than:

    count + 1;

11. Always use the LENGTH or ATTRIB statements for character variable lengths.

12. Avoid "tricks", especially if someone else is going to maintain the code that you write. Instead of:

    x = y > 0;

    be a bit verbose and write something like:

    if y > 0 then x = 1; else x = 0;

13. Use formats and the PUT function to group/recode variables (do this instead of coding a series of IF-THEN statements).

14. Ensure closure of IF-THEN-ELSE sequences. Don't let missing values be generated by default. Instead of:

    if x > 10 then y = .8 * z; else if x = 10 then y = .6 * z;

    write something like:

    if x> 10 then y = .8 * z; else if x = 10 then y = .6 * z; else y = .;

15. Use functions to calculate statistics - avoid "hard-coding". This is true even if performance/resource usage is negatively affected by using the functions.

16. Use PROC SUMMARY rather than a hand-coded DATA step to calculate univariate statistics. The PROC, even if it has to be followed by a PRINT or another DATA step is almost always faster and more reliable than if you calculate the numbers yourself.

17. When developing programs, don't DROP/KEEP variables until you're sure the dataset "looks" acceptable.

18. Avoid taking default dataset names.

19. Even though it is less compact and not "elegant," split up the work into two or more statements when doing complicated (like "nesting") function calls and calculations, . Rather than writing:

    dayname = put(weekday(put(input(cdate,$7.),mmddyy7.)),weekfmt9.);

    (which is "sexy" but unreadable), write something that will be understood later, like:

    ndate = put(cdate,$7.); date = input(ndate,mmddyy7.);
    daynum = weekday(date); dayname = put(daynum,weekfmt9.);

    Later on, when you are sure that DAYNAME is correct, you can DROP the intermediate variables. This rule is purely a diagnostic aid.

REFERENCES

- *SAS System for Elementary Statistical Analysis.* Available for purchase at local bookstores. An excellent introduction to learning both SAS and statistics. Published by the SAS Institute, © 1987. ISBN 1-55544-076-2.
- *SAS User's Guide: Basics, Version 6 Sixth Edition* . Available for purchase at local bookstores. Published by the SAS Institute, © 1985. ISBN 0-917382-65-X.
- *SAS /STAT User's Guide: Statistics, Version 6 Vol. 1-2* . Available for purchase at local bookstores. Published by the SAS Institute, © 1985. ISBN 0-917382-66-8.
- *SAS Introductory Guide Third Edition.* Available for purchase at local bookstores. Published by the SAS Institute, © 1985. ISBN 0-917382-73-0.

This document was is based on IST101-1768 by Bruce W. Derr and Catherine L. Ladd of Syracuse University, Academic Computing services. Thanks for this resource!