November 2013 SAS Workshop H. Seltman

I. What is SAS?

Along with SPSS, Minitab, Stata, Splus, R, JMP, and others, SAS is full featured statistical package used by academia, industry, and government for data management and analysis. It has a 2007 market share of 32%, which is twice the nearest competitor, SPSS. SAS also has a very long history (since 1976), with many organizations tied to "legacy code" decades old. In specific areas, other programs are used more, e.g., Stata in survey sampling. SAS works on Windows or Unix, but not Macs. SAS is very efficient with large datasets.

Basic SAS includes data management, graphing, standard statistical analyses, custom reporting, an interactive matrix language (IML) for general purpose analyses, and a macro facility. Additional packages include genetics, spatial data, data mining, forecasting, inventory management, quality control, image analysis, etc.

Links: http://www.stat.cmu.edu/~hseltman/SASTips.html

See Appendix 2, below, for SAS books.

II. SAS vs. R

SAS is more standardized and more thoroughly tested than R. It is more commonly used among well-established companies, particularly for those that must demonstrate that they follow government regulations, e.g., drug companies. R is more nimble, and tends to be used by more nimble companies. R is a complete programming, statistics, and graphics language. SAS is not really a programming language and the graphics are just beginning to catch up with R in terms of flexibility. SAS has a macro language, its PROC IML (interactive matrix language), and a seldom-used C interface for programming. The actual statistical procedures are not user-modifiable, while many of the R ones are.

R has a rich set of data types, while SAS supports only numeric and character. SAS has better date handling capabilities.

SAS code is entered block-by-block (PROCs and DATA steps), while R is line by line. SAS is case insensitive, while R is case sensitive. R lines end when "syntactically complete", while SAS lines end with a semicolon. SAS datasets live in "libraries" on the disk and no variables are persistent between sessions. R datasets live in "environments" that must be loaded into memory (but are stored in .RData files along with all other variables between sessions), and variables are persistent between sessions. SAS uses "." for numeric missing.

III. Getting started

To get started with this lab, download workshop.zip onto your computer and unzip the files into a directory that you create for that purpose. Just for this lab create a new subfolder named "mydata" in the folder containing the unzipped files.

One standard approach to working in SAS is to use its GUI (graphical user interface), being sure to save all program statements to files once they are debugged. Start SAS on your computer or a cluster computer. (There are some minor differences between platforms (Unix vs. Windows) and versions.) The GUI will appear.

As an alternative to using a downloaded version of SAS, you can use Virtual Andrew. Or you can remotely login to unix.andrew.cmu.edu using SSH, and then type “sas&” to start SAS (the optional ampersand is helpful because you get the Unix prompt back immediately).

When you start SAS, you will see these windows, along with menus and a toolbar:

A. **Explorer / Results** window(s) to navigate results and datasets
B. Program **Editor** window to enter or load SAS commands (programs)
C. **Log** window to view important messages about your programs
D. **Output** (listing) window to view statistical results (now unused by default)
E. **Graphics** window (not seen initially) to view certain types of graphs
F. **Toolbox** window or toolbar to carry out common actions

- Remember that at any time you can change the size and position of each window to whatever works best for you.
- Use the View menu item or the bottom tabs to switch to another window.
- Quit SAS with File/Exit from the Results, Explorer or Editor windows.
- Edit/ClearAll from the Program Editor, Log, or Output window is very useful when switching tasks. Try it in the log window now.
- It is usually convenient to use the menu item Tools/Options/ChangeDirectory at the beginning of your session to make the directory with your data be the working directory. Do it now.

The basic procedure for doing anything in SAS is:

A. Type the commands into the Program Editor or load an existing file (using File / Open program on the menu).
B. Run all or some of the program. Select some program lines to execute only those or don’t select any lines to execute the entire program. Click the Submit button (which looks like a running person), or use right click and choose “Submit selection”.
C. Check the Log Window for errors and warnings (always!!!).
D. Read the results in the Output Window (old style) or in the Results Viewer (new html output)

Now try running the **example program in the file “nrs.sas”** (from the lab website, and also reproduced below). Think/ask about all of the comments in the code – these contain critical information for learning SAS. Examine the Log and Results windows after running the program. The tasks accomplished by this program are to read the data from the text file “nrs.dat” into SAS and to print it out.

```
/*****************************************/
/* File: nrs.sas                         */
/* Read name, rank and serial number data */
/*****************************************/

/* Comments can be anything starting with slash-star and
   ending with star-slash. */
```

```
* Anything starting with a star and ending with a semicolon
  is also a comment (but this style is less preferred);

/* SAS commands end with a semicolon and can be written
   over multiple lines.  My convention is to use upper
   case for words that must match the example exactly
   and lower case for arbitrary names.
*/

/* SAS programs often start by setting some options that
   affect the entire program. Titles are useful! */
OPTIONS LINESIZE=74;
TITLE "Simple test of name, rank, serial number";

/* General rules: SAS variable and dataset names start with a letter,
   can be up to 32 characters, may include numbers and underscores,
   and are not case sensitive.  Commas are NOT used to separate list
   items.  In data files, a period (.) is used to indicate missing
   data.
*/

/* The DATA step(s) name and read in data.
   The INPUT command of a DATA step names the variables.  A dollar
   sign is needed to identify a string (non-numeric) variable.
   In this first example, column positions are used to specify
   where each variable is in the input file.  This approach is
   only used if you happen to have "fixed-column" data files.
   Standard math plus special functions (see appendix) can be used
   to create new variables from old variables.
   Use 'RUN;' to complete a DATA step or PROC step (see below).
   You can enter as many DATA steps as you like to have multiple
   datasets "open" at one time.
*/

DATA military;
    INFILE "nrs.dat";
    INPUT last $ 1-14 first $ 15-24 rank $ 25-35
        serialNum 36-42 IQ 46-48 bdate $ 51-60 sex 62;
    serIQrat = serialNum / IQ;
RUN;

/* PROC steps analyze data. */
/* They use the most recently used data set unless explicitly
   overridden with the optional data= option.
*/
PROC PRINT DATA=military;
RUN;
```

IV. The OPTIONS statement

This precedes DATA and PROC steps. Most options are rather technical. LINESIZE= is useful for restricting the number of characters per line in output. You may also prefer NOCENTER over the default setting that centers output.

V. Loading data in SAS: The DATA step

A SAS program's DATA step(s) load and manipulate data. As opposed to Excel, SPSS, R, etc., all data manipulation is kept separate from data analyses. The data can be in an external file or included in the SAS program. The DATA step produces a "SAS data set" which is in a special SAS format. The SAS data set may be temporary (existing only during the current session) or permanent (stored on disk). If a file is permanent, future SAS programs can skip the DATA step and access that data set directly.

A DATA step typically consists of some of these:

A. A (required) DATA statement which specifies the SAS name for the data and, optionally, the location for a permanent dataset.
B. INFORMAT statements to specify input formats, i.e., to explain to SAS how to read special kinds of data
C. An INFILE statement or a DATALINES block to specify the data
D. An INPUT statement to identify variables (because SAS cannot read variable names from your data file!)
E. LABEL statements to specify clearer (usually longer) variable descriptions
F. FORMAT statements to specify output formats
G. Expressions to calculate new variables
H. Expressions to conditionally alter variables
I. Expressions for dropping cases (rows or observations)

- Consider using edit/clear for the Log and Editor before running the examples in readData.sas and readData2.sas
- To clear the Results Viewer use Output Delivery System commands:
  ```
  ODS HTML CLOSE;  ODS FILE="myfile.html";
  ```
- Or to set output to the Output Window (which can be cleared at any time):
  ```
  ODS HTML CLOSE;   ODS LISTING;
  ```

```
/************************************/
/* File: readData.sas               */
/* LIBNAME and permanent data files */
/************************************/

/**************************************************
INPUT statements read data and save the data to
SAS-format files, but without use of "libraries"
these files are "temporary" and automatically deleted
when you exit SAS.

Use "libraries" to make your SAS-format data files
"permanent", i.e., persistent from session to session.
If a file is permanent, then you don't need to repeat
the DATA and INPUT portions of your program the
next time you run SAS, you don't need to keep your
original data files around (though it is always good
practice to keep a copy), and the data will load
much faster.

Permanent SAS data files have an extension like
"sas7bdat".  Besides SAS, SPSS, the "foreign" package,
of R and several other programs can read these files.
**************************************************/

OPTIONS LINESIZE=74;

/* A LIBNAME statement, used before the DATA or PROC
   steps, is used (as many times as needed) to attach
   a temporary (for one SAS session) SAS name to a
   folder or subdirectory where SAS data files are
   permanently stored.

   The first argument to LIBNAME is the name used to
   identify the library (subdirectory) for the current
   SAS session.  The second argument is a relative or
   absolute description of a subdirectory location, in
   quotes.  Note that you can use "." to store data in
   the current directory.

   In this example "mydata" is a subdirectory of the
   current directory.  By explicitly using
   "mylib.military" in the DATA statement, the "military"
   dataset is stored as a permanent file in the "mydata"
   folder.  Remember: the form "lib.name" refers to a
   permanent data file, while the form "name" refers to a
   temporary one.
*/

LIBNAME mylib "mydata";
```

```
/* Read name, rank and serial number data by column position */
DATA mylib.military;
    INFILE "nrs.dat";
    INPUT last $ 1-14 first $ 15-24 rank $ 25-35
        serialNum 36-42 IQ 46-48 bdate $ 51-60 sex 62;
    serIQrat = serialNum / IQ;
RUN;

/* Now you should end your SAS session, then come back
   and try the program dataRead2.sas. */

/********************************************/
/* File: readData2.sas                      */
/* DATA reading, continued.                 */
/********************************************/

/* Use an existing permanent SAS data file */
OPTIONS LINESIZE=74;
LIBNAME libForMe "mydata";
TITLE ‘Read data examples’;
PROC PRINT DATA=libForMe.military;
RUN;

/* printing only a few lines */
PROC PRINT DATA=libForMe.military(OBS=4);
RUN;

/* Input a data file that has a row of column headers. */
TITLE2 ‘With column headers’;
DATA temp;
  INFILE "nrsH.dat" FIRSTOBS=2;
  /* Note that the SAS program can only skip, not use, the header. */
  INPUT last $ 1-14 first $ 15-24 rank $ 25-35
      serialNum 36-42 IQ 46-48 bdate $ 51-60 sex 62;
RUN;
PROC PRINT DATA=temp;
RUN;


/* Input a data file with spaces between the columns. */
TITLE2 'As space separated variables';
DATA temp;
  INFILE "nrs.dat";
  INPUT last $ first $ rank $ serialNum IQ bdate $ sex;
RUN;
PROC PRINT DATA=temp;
RUN;
```

```
/* By default, the above chops strings off at 8 characters (and  */
/* does not treat dates as meaningful).                           */
/* INFORMAT's solve both problems.                                */
TITLE2 'With informat for dates and wider last and rank';
DATA temp;
  INFILE "nrs.dat";
  INFORMAT last $9. first $8. rank $10. bdate MMDDYY10. ;
  *** Note: MMDDYY10 represents MM/DD/YYYY (10 characters);
  INPUT last first rank serialNum IQ bdate sex;
  age = (TODAY()-bdate)/365.25;
RUN;
PROC PRINT DATA=temp;
RUN;

/* Now we have a problem with dates not printing as desired. */
/* We specified the input format, but not the output format. */
/* Also numeric coded categorical variables can be better    */
/* shown if a PROC FORMAT is used to define VALUES for later */
/* use in FORMAT statements. (A final period (.) is needed   */
/* when referring back to these FORMATs.)                    */
TITLE2 'With formatted output';
PROC FORMAT;  /* optional LIBRARY= makes this permanent */
  VALUE sexfmt 1='Female' 2='Male';
RUN;
PROC PRINT DATA=temp;
  FORMAT bdate MMDDYY. sex sexfmt.;
RUN;


/* Input a data file with commas between the columns (DSD). */
/* Sequential commas gives missing data (without the usual  */
/* "." missing value code).  Quotes are removed from quoted */
/* strings.                                                 */
/* For any DATA step, FORMAT's can be included, then they   */
/* are used for all subsequent PROC;s.                      */
TITLE2 'Using DSD for commas';
DATA temp;
  INFILE "nrsC.dat" DSD;
  INFORMAT last $9. first $8. rank $10. bdate MMDDYY10. ;
  INPUT last $ first $ rank $ serialNum IQ bdate sex;
  FORMAT bdate MMDDYY. sex sexfmt.;
RUN;
PROC PRINT DATA=temp;
RUN;

/* Demonstrate conditionally dropping cases. */
TITLE2 'Dropping privates';
DATA temp;
  INFILE "nrs.dat";
  INFORMAT last $9. first $8. rank $10. bdate MMDDYY10. ;
  INPUT last first rank serialNum IQ bdate sex;
  if rank="Private" then delete;
RUN;
PROC PRINT;
RUN;
```

```
/* Demonstrate conditionally altering variables. */
TITLE2 'IQgroup and funny created with if statements';
DATA temp;
  INFILE "nrs.dat";
  INFORMAT last $9. first $8. rank $10. bdate MMDDYY10. ;
  INPUT last first rank serialNum IQ bdate sex;
  IF IQ GE 100 THEN IQgroup = 'Normal';
    ELSE IQgroup='Dull';
  IF IQ GT 130 THEN IQgroup = 'Genius';
  funny = 0;
  IF first = 'Jay' | last = 'Maher' THEN funny = 1;
RUN;
PROC PRINT;
RUN;
```

Try navigating in the Results window. Try some minor modifications of the above code.

VI. Centering variables, etc.: PROC SQL

A. Syntax:

```
PROC SQL;
  CREATE TABLE myNewData AS
    SELECT var1, var2, … (or * for all)  /*Big exception: commas between variables!!! */
      FROM myOldData;
QUIT;  /* Big exception: this PROC ends with QUIT, not RUN!!! */
```

B. See this SAS pdf for more details. Here, we are most interested in a variant of SELECT that creates centered variables as in the example below.

C. Try this example from sql.sas:

```
/**********************************/
/* File sql.sas                   */
/* Demonstrate variable centering. */
/**********************************/
OPTIONS LINESIZE=68;
TITLE 'Centering of fake mixed data with PROC SQL';

DATA fakeMixed;
  INFILE 'fakeMixed.txt' FIRSTOBS=2;
  INPUT SubjID ClassID male ses score;

RUN;

PROC SQL;
    CREATE TABLE fakeMixed2 AS
      SELECT *, ses - mean(ses) as cses /*centered ses*/
      FROM fakeMixed;
QUIT;

PROC PRINT DATA=fakeMixed2(OBS=4);
RUN;
```

VII. Sorting

A. Required if grouped analyses using BY (see below) are desired

B. Syntax:

```
PROC SORT DATA=indata OUT=outdata NODUPKEY;
  BY var1 var2 …;
RUN;
```

C. OUT= is optional; dropping it sorts "in-place". Optional NODUPKEY drops duplicates.

D. Try sorting the nrs data (in the "temp" dataset) by rank then last name. Be sure to examine the resulting dataset using PROC PRINT.

E. As an alternative, examine your sorted data by navigating to it in the Explorer window. Like all temporary data files, it is in the "Work" folder which is in the "Libraries" folder (file cabinet). Double clicking on a data set opens it for viewing. **Be sure to close it when you are done looking at it** or there will be problems if you try to access it later!

VIII. EDA in SAS: PROC FREQ and PROC UNIVARIATE

A. Syntax for PROC FREQ (EDA and testing for categorical data):

```
PROC FREQ DATA=indata;
  TABLES v1 v2 … / options;   /* v1, v2, etc. are variables in the data set */
  TABLES v1*v2 / options;
RUN;
```

B. Some options: CHISQ, CELLCHI2, FISHER, NOPERCENT, NOROW, NOCOL, NOCUM, EXPECTED, DEVIATION, OUT=dsname

C. Try some one-way and two way tables for rank and sex.

D. Optional WEIGHT statement is needed if counts instead of cases are in the data file. Try weightFreq.sas. This also demonstrates how to include your data inside your program, rather than in a separate file, by using the DATALINES (or CARDS) statement.

```
/************************************/
/* File weightFreq.sas              */
/* Demonstrate chi2 with count data */
/************************************/

TITLE 'Two way table with Chi-square';
TITLE2 'Using weights (counts)';
DATA wtfreq;
  INPUT agegrp$ gender$ count;
  DATALINES;
  young male 10
  young female 15
  med male 20
  med female 24
  old male 30
  old female 42
RUN;
```

```
PROC PRINT DATA=wtfreq;
RUN;

PROC FREQ DATA=wtfreq;
  TABLES agegrp*gender / EXPECTED CHISQ CELLCHI2;
  WEIGHT count;
RUN;
```

E. Syntax for PROC UNIVARIATE (standard EDA for quantitative variables):

```
PROC UNIVARIATE;
  VAR v1 v2 …;
RUN;
```

F. Useful option: NORMALTEST

G. Try running nrsUnivariate.sas:

```
/* Fancier plotting for name/rank/serial number data */

OPTIONS LINESIZE=74;
LIBNAME mydata "mydata";
TITLE 'EDA for quantitative variables';

/* Basic statistics for quantitative data */
/* Test of Normality is an option. */
TITLE2 'Simple univariate analysis';
PROC UNIVARIATE DATA=mydata.military NORMALTEST;
  VAR IQ serIQrat;
RUN;

/* Additional options make plots */
TITLE2 'With histogram and QQ-plot';
PROC UNIVARIATE DATA=mydata.military;
    VAR IQ serIQrat;
    HISTOGRAM serIQrat / NORMAL;
    HISTOGRAM IQ / NORMAL;
    QQPLOT IQ;
RUN;


/* Additional plot options make things nicer */
TITLE2 'Add QQ reference line; control binning';
PROC UNIVARIATE DATA=mydata.military;
    VAR IQ;
    QQPLOT / NORMAL (MU=117.1 SIGMA=22.44 COLOR=red);
    HISTOGRAM / NORMAL ENDPOINTS = 90 to 160 by 5;
    HISTOGRAM / NORMAL ENDPOINTS = 90 to 160 by 10;
RUN;

/* Most procs allow BY statements to analyze for */
/* separate categories.                          */
/* This only works correctly for sorted data.    */
TITLE2 'By rank';
PROC SORT DATA=mydata.military;
  BY rank;
RUN;
PROC UNIVARIATE DATA=mydata.military  NORMALTEST;
  BY rank;
  VAR IQ;
RUN;

/* But without sorting you can use: */
PROC UNIVARIATE DATA=mydata.military  NORMALTEST;
  CLASS rank;
  VAR IQ;
RUN;
```

IX. Graphical EDA: PROC SGPLOT

A. Syntax (greatly abbreviated)

```
PROC SGPLOT DATA=indata;
  SCATTER X=xvar Y=yvar;  /* to make a scatter plot */
  VBOX var; /* to make a boxplot */
  VBOX var / CATEGORY=var2;  /* to make clustered boxplots */
RUN;
```

B. Examples:

```
PROC SGPLOT DATA=temp;
  SCATTER X=bdate Y=IQ;
RUN;
PROC SGPLOT DATA=temp;
  VBOX IQ;
RUN;
PROC SGPLOT DATA=temp;
  VBOX IQ / CATEGORY=rank;
RUN;
```

X. ANOVA in SAS: PROC ANOVA

A. Syntax:

```
PROC ANOVA;
  CLASS x1 x2 …;
  MODEL y = x1 x2;
  MEANS x1 x2 x1*x2 / options;
RUN;
```

In the model statement "*" is for interaction and, e.g., A|B produces A B A*B.

B. Some MEANS options: TUKEY, SCHEFFE, DUNNET('control'), BON, ALPHA=p

C. Try the examples in anovaTest.sas:

```
/**********************/
/* File anovaTest.sas */
/* Demonstrate ANOVA. */
/**********************/

DATA fake;
  INPUT agegrp$ gender$ score;
  DATALINES;
  young male 11
  young male 15
  ...
  old female 48
  old female 53
  old female 51
RUN;

PROC ANOVA DATA=fake;
  CLASS agegrp gender;
  MODEL score = agegrp gender agegrp*gender;
  MEANS agegrp / TUKEY;
RUN;
```

XI. Getting help: Click on the Help menu item

A. Choose "SAS Help and Documentation" (note other choices).

B. Example: search for "mixed models PROC". (Adding PROC usually helps.)

   i. On the (long) list go to "The MIXED Procedure: Overview". Usually you want to find a "Procedure" matching what you want to do.

   ii. Use Previous Page to get to the documentation outline. From here you can jump to any needed documentation page. The SYNTAX pages are often very helpful.

   iii. You can go to the Favorites tab and click Add to "bookmark" this help page.

XII. Regression in SAS: PROC GLM (General Linear Model)

A. Syntax:

```
PROC REG;
  MODEL y =x1 x2 …;
  PLOT RESIDUAL.*PREDICTED.;
RUN;
```

B. Categorical variables must be recoded as indicators. PROC GLM is similar, but has a CLASS statement to automatically recode categorical variables. It uses OUTPUT and GPLOT instead of PLOT.

C. Try the examples in regTest.sas:

```
/***************************/
/* File regTest.sas        */
/* Demonstrate regression. */
/***************************/
TITLE 'Algebra data analyzed as ANCOVA with GLM';
/* Note: DELIMITER='09'X designates tab separated data */
PROC FORMAT;
  VALUE schoolfmt 1="elem" 2="middle" 3="high";
RUN;
DATA algebra;
  INFILE 'algebra.txt' FIRSTOBS=2 DELIMITER='09'X;
  INPUT id$ school experience score;
  FORMAT school schoolfmt.;
RUN;
PROC PRINT DATA=algebra;
RUN;

/* Important note: WHERE can restrict cases for any PROC. */
/* The CLASS statement creates indicator variables automatically. */
/* A "*" can be used to create interaction variables. */
TITLE2 'ANCOVA with interaction';
PROC GLM DATA=algebra;
  WHERE experience<22;  * drop one outlier with experience=22;
  CLASS school;
  /* Solution is needed to get the coefficient table */
  MODEL score = school experience school*experience / SOLUTION;
RUN;


TITLE2 'No interaction model; save and plot residuals';
PROC GLM DATA=algebra PLOT=(ALL);  /* PLOT=(ALL) adds diagnostics */
  CLASS school;
  MODEL score = school experience / SOLUTION;
  OUTPUT OUT=algdiag RESIDUAL=res PREDICTED=fit;
RUN;

/* Residual vs. fit plot */
PROC GPLOT DATA=algdiag;
  PLOT res*fit / VREF=0;
RUN;



/* Find sd of residuals for QN plot */
PROC MEANS DATA=algdiag;
  VAR res;
RUN;
```

```
/* QN plot */
PROC UNIVARIATE DATA=algdiag NOPRINT;
  VAR res;
  QQPLOT / NORMAL (MU=0 SIGMA=0.43253 COLOR=red);
RUN;
```

XIII. Proc Mixed: random effects / hierarchical linear models

A. Syntax:

```
PROC MIXED  COVTEST NOCLPRINT NOITPRINT METHOD=ML/REML;
  CLASS catvar1 catvar2 …;
  MODEL y = x1 x2 … / SOLUTION OUTPRED=dataset DDFM=SAT;
  RANDOM INTERCEPT var1 var2 … / SUBJECT=subvar TYPE=UN/VC/etc.;
  REPEATED var1 var2 … / TYPE=AR(1)/etc.
  ESTIMATE ‘label’ <fixed-effect-values> | <random-effect-values>;
  LSMEANS ‘label’ <fixed-effects> / ADJUST=BON/SCHEFFE/etc.;
RUN;
```

B. The PROC options listed above allow for a covariance test, suppression of the list of all class levels, suppression of the iteration towards the maximum likelihood, and using the ML instead of the default REML likelihood maximization method.   The DDFM= option should be used to get a better method for calculation of denominator degrees of freedom. For possible TYPEs for the RANDOM and REPEATED statements see type list.

C. Try the examples in mixed.sas. The data are simulated, but are similar to a real dataset with multiple subjects (students) for each classroom, with gender and socio-economic status as covariates, and with test score as the outcome.

```
/*****************************/
/* File mixed.sas            */
/* Demonstrate mixed models. */
/*****************************/
OPTIONS LINESIZE=68;
TITLE 'Fake mixed data';
DATA fakeMixed;
  INFILE 'fakeMixed.txt' FIRSTOBS=2;
  INPUT SubjID ClassID male ses score;

RUN;

/* Create dataset with additional variable: centered ses */
PROC SQL;
    CREATE TABLE fakeMixed2 AS
      SELECT *, ses - mean(ses) as cses
      FROM fakeMixed;
QUIT;
PROC PRINT DATA=fakeMixed2(OBS=4);
RUN;
```

```
TITLE2 'Fixed effect model (just a grand mean)';
PROC MIXED DATA=fakeMixed2;
  MODEL score =  / SOLUTION;
RUN;

TITLE2 'Random intercept model';
PROC MIXED DATA=fakeMixed2 COVTEST;
  CLASS ClassID;
  MODEL score = / SOLUTION;
  RANDOM INTERCEPT / SUBJECT=ClassID;
RUN;


TITLE2 'Add centered ses';
PROC MIXED DATA=fakeMixed2 COVTEST NOCLPRINT PLOTS=(ALL);
  CLASS ClassID;
  MODEL score = cses / SOLUTION;
  RANDOM INTERCEPT / SUBJECT=ClassID;
RUN;

TITLE2 'Add gender';
PROC MIXED DATA=fakeMixed2 COVTEST NOCLPRINT PLOTS=(ALL);
  CLASS ClassID;
  MODEL score = cses male / SOLUTION;
  RANDOM INTERCEPT / SUBJECT=ClassID;
RUN;
```

Interpretation of these results is a key part of any HLM course.

XIV. Other common PROCs:

A. CLUSTER: cluster analysis
B. FACTOR: factor analysis
C. GLMPOWER: power calculations for linear models
D. LIFETEST: Survival models
E. LOGISTIC: Logistic regression models
F. PHREG: Cox proportional hazards model
G. POWER: power calculations
H. PRINCOMP: principal components analysis
I. SURVEYFREQ/SURVEYLOGISTIC/SURVEYMEANS/SURVEYREG: complex surveys
J. TTEST: t-test
K. TREE: Hierarchical clustering

XV. The Macro Language

A. Overview (in SAS doc's words):

The macro facility is a tool for extending and customizing SAS and for reducing the amount of text you must enter to do common tasks. The macro facility enables you to assign a name to character strings or groups of SAS programming statements. From that point on, you can work with the names rather than with the text itself.

When you use a macro facility name in a SAS program or from a command prompt, the macro facility generates SAS statements and commands as needed. The rest of SAS receives those statements and uses them in the same way it uses the ones you enter in the standard manner.

The macro facility has two components:

- The macro processor is the portion of SAS that does the work.
- The macro language is the syntax that you use to communicate with the macro processor.

When SAS compiles program text, two delimiters trigger macro processor activity:

&name refers to a macro variable. The form &name is called a macro variable reference.

%name refers to a macro. The form %name is called a macro call.

The text substitution produced by the macro processor is completed before the program text is compiled and executed. The macro facility uses statements and functions that resemble those that you use in the DATA step. An important difference, however, is that macro language elements can only trigger text substitution and are not present during program or command execution.

B. Example (macro.sas):

```
/* Create a macro to do EDA, regression, and residual plotting */
%macro regAndPlot(data, y, x);
    /* Double quotes are needed to allow substitution */
    TITLE "EDA of &data: &y on &x";
    /* RL is regression, linear for adding a fit line to a plot */
    SYMBOL INTERPOL=RL VALUE=plus;
    PROC GPLOT DATA=&data;
      PLOT &y * &x;
    RUN;

    TITLE "Regression analysis for &data";
    PROC GLM DATA=&data;
      MODEL &y = &x / SOLUTION;
      OUTPUT OUT=_temp RESIDUAL=res PREDICTED=pred;
    RUN;
    TITLE;
    PROC UNIVARIATE DATA=_temp NOPRINT;
      VAR res;
      OUTPUT OUT=_tmpstd STD=stdev;
    RUN;

    DATA _NULL_;
      SET _tmpstd;
      CALL SYMPUT('resSD',stdev);
    RUN;

    TITLE "Regression with &data data: &y on &x";
    TITLE2 'Quantile normal plot of residuals';
    PROC UNIVARIATE DATA=_temp NOPRINT;
      VAR res;
      QQPLOT / NORMAL (MU=0 SIGMA=&resSD COLOR=red);
    RUN;

    TITLE2 'Residual vs. fit plot';
    PROC GPLOT DATA=_temp;
      PLOT res*pred / VREF=0;
    RUN;
%mend regAndPlot;

/* Test the macro on the algebra data */
/* Note: call does not end with a semicolon. */
%regAndPlot(algebra, score, experience)
```

C. Further comments: Macros range from very simple to very complex. The learning curve is fairly steep, but relatively simple time savers like the above example are not too difficult to write. There are many examples on the Web that can be used as templates for creating your own macros or for learning about macro programming.

XVI. Appendix 1: SAS operators in expressions

A. Basic Math: + - / * and, for powers, **

B. Precedence: **, then * and /, then + and -

C. Selected useful functions: abs, int, log, log10, round(x,2), sqrt

D. Statistical functions (per subject)

   i. Y = mean (x1, x2, x3)

   ii. Also min, max, std, sum

   iii. missCount = Nmiss(x1, x2, x3)

   iv. notMissingCount = N(x1, x2, x3)

E. Comparison operators

   i. Eq or =, Ne or ~=

   ii. Lt or <, Gt or >

   iii. Le or <=, Ge or >=

F. Logical operators

   i. & or AND

   ii. | or OR

   iii. ~ or NOT

XVII. Appendix 2: SAS Books

A. Best intro/reference: SAS Application Programming: A Gentle Introduction , by Frank C. Dilorio, SAS Institute

B. Lots of examples with online data: Handbook of Statistical Analyses using SAS by B.S. Everitt and G. Der, Chapman & Hall

C. The Little SAS Book: A Primer by L.D. Delwiche and S. J. Slaughter, SAS Institute

D. Categorical Data Analysis Using the SAS System by Stokes, Davis and Koch, SAS Institute

E. SAS System for Mixed Models by Littell, Milliken, Stroup and Wolfinger, SAS Institute

F. SAS Macro Facility, Tips and Techniques, SAS Institute

G. SAS Procedures Guide, SAS Institute

H. SAS System for Statistical Graphics, by Michael Friendly, SAS Institute