Paper SY08

Stata® for the Struggling SAS® Mind

Dan Blanchette, Carolina Population Center, UNC-CH, Chapel Hill, NC

ABSTRACT

This paper shows how to "do" in Stata what you know how to "program" in SAS. For those SAS programmers who need the statistical software Stata for certain analysis commands or have been given Stata code that they need to use, this is basically a formal presentation of notes to myself that I made as I started learning how to use Stata. What is covered:

1) How to get your SAS brain to conceptualize the different way Stata is set up. In essence, how to think Stata.
2) How to navigate the Stata window environment.
3) How to translate your SAS code into Stata code.

With a SAS programmer in mind, this paper makes use of your SAS knowledge to explain how to use Stata. This makes learning Stata a lot easier as familiar SAS terms and concepts are used in the explanations. Not all SAS statements are translated into Stata commands. Just the basics are covered so that a new-to-Stata user can learn enough to get started. Perhaps the handiest feature is the table of SAS code matched to Stata code.

INTRODUCTION

For some reason you need to learn how to use Stata. Having already gone through perhaps a long and arduous period learning SAS, you are likely not eager to do the same to learn another software package. This paper is intended to ease the pain by making use of what you already know in SAS.

STATA AND SAS ARE SIMILAR

Stata is also a statistical software package. Stata is very much like SAS in that you can read in a data set, manipulate the values of variables and analyze the data with it. For the most part, it is the case that anything you can do in SAS you can also do in Stata. There are many tools available to create complex programs, stylized reports and graphs. This paper is not going to cover all of that, but you should know there is much more to Stata than this paper talks about.

STATA GOES ABOUT THINGS DIFFERENTLY

Stata is more than just a different programming language. The whole software package is set up differently than SAS which spawns many nuances that you need to keep in mind while interacting with it. At first you are likely to find this frustrating, but you may find certain aspects of Stata preferable to SAS.

DATA

The biggest difference between Stata and SAS is that Stata loads the entire data set into memory where SAS typically only has the current observation in memory. In Stata you can access the values of variables in any observation in the whole data set at almost any time. For this reason a RETAIN statement would make no sense in Stata.

SETTING MEMORY

Stata requires you to be aware of how much of the computer's memory will be required to load the data set into memory. Stata's default setting for memory frequently is large enough to keep this from being an issue, but you should be aware that the memory setting might need to be tweaked. A quick rule of thumb would be to set Stata's memory to 20% more than the size of the data set. If your data sets tend to be smaller than 30 megabytes, set your default memory setting to 40m. Stata allows you to modify your initial memory setting when no data are in memory.

COMMANDS VS. STATEMENTS

In Stata one command modifies the entire data set one observation at a time, then the next command modifies the entire data set, etc. In SAS a series of statements modifies one observation at a time, then the entire series modifies the next observation, etc. Think of each Stata command as a SAS data step containing only one statement:

```
data mydata;
set mydata;
var3=var1/var2*100;
run;

data mydata;
set mydata;
agegroup=(0<=age<2)*1+(2<=age<8)*2+(8<=age<18)*3;
run;
```

MISSING VALUES

For numeric variables, missing values in SAS are the SMALLEST values whereas in Stata they are the LARGEST values. Starting with Version 8, Stata allows for special missing values .a-.z. Unlike SAS where special missing value dot underscore (._) is smaller than dot missing (.), followed by .a alphabetically up to the largest missing value .z, Stata's dot missing (.) is the smallest missing value and .z is the largest missing value (Stata has no dot underscore). Thus the expression "<." in Stata excludes all missing values whereas ">.z" in SAS excludes all missing values.

IN CASE, IT MATTERS

Capital or lower case SAS code does not matter. For example the variable "Age" is the same as variable "age" or "aGe", etc. In Stata lower case is most often used because case does matter. For example the variables "Age", "age" and "aGe" are all different variables, albeit not very good names for different variables. All commands in Stata are in lower case so beware the Caps Lock key.

THE END OF THE LINE

The end-of-line delimiter in SAS is the semi-colon (;).

```
data one;
set two;
 if oldvar1=2 and oldvar2=1 then
  newvar=1;
run;
```

Stata's default end-of-line delimiter is the carriage return, which is invisible to us. You can set the end-of-line delimiter to be the semi-colon with the " #delimit ; " command. You might find this a helpful crutch at first, but I encourage you to give the carriage return a chance since it is less to type and it can be a hassle to keep track of what the end-of-line delimiter is in your code. Here are some examples in Stata that show how to span more than one line in one command:

```
generate newvar=1 if oldvar1==2 /* comments out the carriage return
  this ends the comment */ & oldvar2==1
```

Or you can use 3 forward slashes preceded by at least one blank space to comment out the rest of the line and to continue the command on the next line.

```
generate newvar=1 if oldvar1==2 /// also comments out the carriage return
                  & oldvar2==1
```

CONDITIONS

In SAS, conditions come first and then something happens. For example:

```
if age>10 then age1=1;
```

Conditions in Stata come at the end of a command. For example:

```
generate age1=1 if age>10 & age<.
```

(Remember to consider that age may equal missing for some cases and that in Stata missing values are greater than any other number.) If you are used to conditionally processing statements in a do-loop in SAS, there is no real substitute for that in Stata. Depending on the problem you are trying to solve, one of Stata's looping commands may come to the rescue. If you find yourself wanting to write a SAS-like conditional do-loop, remind yourself that

Stata processes each command for all observations before processing the next command. Remember: Think Stata.

NO NEED TO SPELL IT OUT

All SAS statements have to be spelled out completely and usually spelled correctly for SAS to recognize them. Stata recognizes a command when it is spelled out enough for Stata to understand what command you intend. For example, the command “generate” can be typed as “g” although “gen” has somehow become its most common abbreviation. Stata documentation of commands shows the first few letters underlined to indicate what characters must be typed. It is common practice to use the abbreviation of the command.

LOGGING

If you want to keep a record of your session in a log file, Stata requires you to decide to do that at the beginning instead of after the fact as in SAS. There is no option to save the contents of the Results window, though you can copy and paste a large amount from the Results window to your favorite text editor. You can also save the contents of the Review window if you forgot to start a log file.

STATA'S WINDOWS MODE

The Windows mode of Stata may initially look vastly different than SAS's, but what you might expect to see is there.

RESULTS WINDOW

Stata's Results window is a mix of SAS's log and output windows. The Results window prints the command and Stata's reaction to that command. The default setting for the maximum number of characters to be stored in the Results window is 32,000. This may not be enough to store all your results, as the SAS output and log windows do, so it is better to log your Stata session instead of relying on the Results window.

COMMAND WINDOW

The Stata Command window is almost like SAS's program editor except that it is designed for the user to submit only one command at a time. (Helpful tip: to scroll back to previously submitted commands use the Page Up key.)

DO-FILE EDITOR

The Stata Do-file Editor is like SAS's program editor. To launch it, click the icon in the tool bar of an open envelope. Here you can submit all the commands in the window, or by highlighting a selection you can submit a single command or several lines of code at a time. To submit commands from the Do-file Editor, click the icon in the tool bar that looks like a piece of paper with lines on it and an arrow pointing down. In SAS you write programs, but in Stata you write do-files. A Stata program is equivalent to a SAS macro.

VARIABLES WINDOW

The Variables window lists the names and labels of the variables in the data set that is currently in memory. If you click on a variable it will appear in the command line.

REVIEW WINDOW

The Review window lists recently submitted commands from the Command window. If you click on a previous command, it will appear in the Command window. If you double click, it will be executed. (Helpful tip: If you want to save the commands in the Review window, perhaps because you did not start logging your Stata session, click to the left of the word "Review" in the title of the window and select "Save Review Contents". This saves the commands in the order they were submitted in the form of a do-file (*.do), which is equivalent to a SAS program (*.sas) file.)

DATA EDITOR AND BROWSER

Stata also offers a Data Editor and a Data Browser just like SAS. To access them there are icons in the tool bar that look like spread sheets. The icon for the data browser is the one with the magnifying glass.

VIEWER

The Stata Viewer is for viewing existing log files as well as help files. To launch the viewer click the icon in the tool bar that is a picture of an eye.

CLICKING YOUR WAY THROUGH STATA

In the tool bar above the Results window there are drop-down menus for Data, Graphics, and Statistics. Here you can click your way through most any Stata command using Stata's dialog boxes and have Stata write the code for you. If you are logging your Stata session, the log will contain all the Stata code as if you had typed it yourself. The Page Up key will also recall your last command even if it was generated by the Stata dialog box.

SAS CODE MATCHED TO STATA CODE

The following table shows you what the equivalent Stata code is for the SAS code you have in mind. This is intended to acquaint you with some of the basics to get you started.

SAS Code Matched to Stata Code

Note: Stata commands are partially underlined to show the minimum characters that need to be typed for Stata to recognize the command. Not all commands can be abbreviated.

SAS	Stata
In SAS operators can be symbols or mnemonic equivalents such as: `&` or: `and` For many situations in SAS order doesn't matter: `<=` can be: `=<` and `>=` can be: `=>`	Most operators are the same in Stata as in SAS, but in Stata operators do not have mnemonic equivalents. For example, you have to use the ampersand (&) and not the word "and": This works: `var_a>=1 & var_b<=10` where this does not: `var_a>=1 and var_b<=10` These are the operators that are different in Stata: `Symbol Definition` `& and` `| or` `>= greater than or equal to` `<= less than or equal to` `== equality` `!= does not equal` `~ not` `^ power` NOTE: Symbols have to be in the order shown: " >= " not " => " .
`/* comment */` `* comment ;`	`/* comment */` `* comment` `// comment` To continue a line: `///` For example: `list hhid personid gender age ///` `weight height race income date`
Range of values: `if 1<=var_a<=10` or: `if var_a in(1,2,3,4,5,6,7,8,9,10)`	`if var_a>=1 & var_a<=10` or: `if inrange(var_a,1,10)` or: `if inlist(var_a,1,2,3,4,5,6,7,8,9,10)`

SAS	Stata
Referencing multiple variables at a time: Say the following variables in the data file in the order shown: `var1 var2 var3 age var4 var5` You could refer to them as: `var1--var5` To SAS, this means "all variables that are positionally between var1 and var5," which would include the variable age.	Referencing multiple variables at a time: `var1-var5` To Stata, this means "all variables that are positionally between var1 and var5." Notice that there is only one hyphen (-).
Referencing multiple variables at a time: `var1-var5` is the same as: `var1 var2 var3 var4 var5` no matter the positions of the variables are in the observation. Using a colon selects variables containing the same prefix: `var:` could represent: `var1 var2 var10 variable varying var_1`	Referencing multiple variables at a time: `var?` The question mark (?) is a wild card that represents one character in the variable name. It could be a number, a letter, or an underscore (_). `var*` The asterisk (*) is a wild card that represents many characters in the variable name. They could be numbers, letters, or underscores. Thus `var*` could represent: `var1 var2 var10 variable varying var_1`
To save the contents of the Log window and/or Output window, go to that window and click on the menu bar's "File", "Save". In SAS batch mode these files are automatically generated for you.	To save the contents of the Results window, start logging to a log file BEFORE you submit commands that you want logged. Open a Log file by clicking on the icon in the tool bar that looks like a scroll and a traffic light. A "*.log" file is a simple ASCII text file; a " *.smcl" file is formatted with html-like tags. You can also use the log command: `log using "d:\mydofile.log", replace` NOTE: The "replace" option simply tells Stata to overwrite the log file if it already exists. This is helpful when you have to run a do-file over and over again.
`libname in8 v8 "d:\mydata\";` `data new;` `set in8.mySASfile;` `run;` or, starting in SAS 8: `data new;` `set "d:\mydata\mysasfile.sas7bdat";` `run;`	`use "d:\mydata\myStataFile.dta"` You can also click on the "open file" icon and select your data set.
Save the data set newer to d:\mydata\ : `data in8.newer;` `set new;` `run;`	`save "d:\mydata\newer.dta"` To overwrite the data set newer if it already exists: `save "d:\mydata\newer.dta" , replace` You can also click on the "save" icon to save your data set.

SAS	Stata
Save a SAS data set as a Stata data set with the SAS macro SAVASTATA, which can be downloaded from the web: http://www.cpc.unc.edu/services/computer/presentations/sas_to_stata/savastata.html `savastata("d:\mydata\", -replace)`	Save a Stata data set as a SAS data set with the Stata command SAVASAS, which can be downloaded from the web: http://www.cpc.unc.edu/services/computer/presentations/sas_to_stata/savasas.html `savasas "d:\mydata\", replace` Or use a SAS data set using the Stata command USESAS, which can be downloaded from the web: http://www.cpc.unc.edu/services/computer/presentations/sas_to_stata/usesas.html `usesas using "d:\mydata\mySAS.sas7bdat"`
`proc contents;` On selected variables: `proc contents data=in8.newer (keep=id age height); run;`	`describe` On selected variables: `describe id age height`
`proc means;` On selected variables: `proc means; var age height; run;` or `proc univariate; var age height; run;`	`summarize` On selected variables: `summarize age height` If you want variable labels and a proc univariate style output try: `summarize age height, detail` or: `codebook age height`
`proc surveymeans; cluster sampunit; strata stratum; var age height; weight sampwt; run;`	`svyset [pweight=sampwt], /// psu(sampunit) strata(stratum)` `svymean age height`
Analyze a subpopulation by implementing the domain option: `proc surveymeans; cluster sampunit; strata stratum; domain female; var age height; weight sampwt; run;`	Analyze a subpopulation by implementing the subpop option: `svymean age height, subpop(females)` NOTE: Options come after a comma (,).
`proc freq;`	`tabulate` or, for just checking out your data set, try: `codebook`

SAS	Stata
A series of 1-way tables: `proc freq;` `tables var1 var2;` `run;`	A series of 1-way tables: `tab1 var1 var2`
A 2-way table: `proc freq;` `tables var1*var2;` `run;`	A 2-way table: `tab2 var1 var2`
Starting in SAS Version 9: `proc surveyfreq;` `cluster sampunit;` `strata stratum;` `tables females*var1*var2;` `weight sampwt;` `run;`	`svyset [pweight=sampwt], ///` `psu(sampunit) strata(stratum)` `svytab var1 var2, subpop(females)` NOTE: The svytab command requires two variables. Currently it is not possible to do a one-way tab with svytab. A work-around for this is to generate a new variable with a value equal to 1 and use that variable as the second variable in your svytab. Stata plans to allow svytab to do one-way tabs in the future.
`proc surveyreg;` `cluster sampunit;` `strata stratum;` `model depvar=indvar1 indvar2;` `weight sampwt;` `run;` Proc surveyreg does not have a way of dealing with subpopulations. Using “by” or “where” will not suffice as they will compute incorrect standard errors.	`svyset [pweight=sampwt], ///` `psu(sampunit) strata(stratum)` `svyreg depvar indvar1 indvar2, /*` `*/ subpop(females)`
Starting in SAS Version 9: `proc surveylogistic;` `cluster sampunit;` `strata stratum;` `model depvar=indvar1 indvar2 indvar3;` `weight sampwt;` `run;` Proc surveylogistic does not have a way of dealing with subpopulations. Using "by" or "where" will not suffice as they will compute incorrect standard errors.	`svyset [pweight=sampwt], ///` `psu(sampunit) strata(stratum)` `svylogit depvar indvar1 indvar2, /*` `*/ subpop(females)`
`proc print;` On selected variables: `proc print;` `var id age height;` `run;` On selected variables and a limited range of observations: `proc print data=new (firstobs=1 obs=20);` `var id age height;` `run;`	`list` On selected variables: `list id age height` On selected variables and a limited range of observations: `list id age height in 1/20`

SAS	Stata
Create a numeric variable with a default length of 8 bytes: `var1=1234;` Create a numeric variable with the minimum allowable length (3 bytes): `length var1 3;` `var1=1234;`	`generate var1=1234` NOTE: The default numeric type is "float." The statement above is relying on that default. It could have been written explicitly as: `generate float var1=1234` "float" stands for "floating point decimal." You could more wisely save storage space by specifying: `gen int var1=1234` "int" stands for "integer."
Create a character variable with a length of 3 bytes: `name="Bob";`	Generate a string variable with a length of 3 bytes: `gen str3 name="Bob"`
Increase the variable length to allow for 5 characters: `data new;` `length name $5;` `set new;` Change the values of numeric and character variables. `var1=123456;` `name="Bobby";` `run;`	`replace var1=123456` Stata automatically increases the storage type if necessary. `replace name="Bobby"` Stata automatically increases the length to 5. To change the storage of a variable manually, use the recast command. `recast int age`
Example of an if-then statement: `if var1=123456 then var2=1;`	The condition follows the command: `replace var2=1 if var1==123456` NOTE: Notice that Stata requires two equals signs when testing equality.
Example of an if-then do loop: `if age<=10 then do;` `child=1;` `parent=0;` `end;`	`replace child=1 if age<=10` `replace parent=0 if age<=10` Since each command is executed on all observations before the next command is executed, the "if-then do loop" is not an option. Stata does have excellent looping tools: foreach, forvalues, and while.
Drop variables var1, var2, and var3: `data new(drop=var1 var2 var3);` `set new;` `run;`	Drop variables var1, var2, and var3: `drop var1 var2 var3`
Keep variables var1, var2, and var3: `data new(keep=var1 var2 var3);` `set new;` `run;`	Keep variables var1, var2, and var3: `keep var1 var2 var3`
Keep observations / subsetting if statement: `data new;` `set new;` `if var1=1;` `run;`	Keep observations `keep if var1==1`

SAS	Stata
Delete observations: `data new;` `set new;` `if var1=1 then delete ;` `run;`	Drop observations: `drop if var1==1`
`data new(drop=i);` `set new;` `array raymond {4} var1 var2 var3 var4;` `do i=1 to 4;` `if raymond{i}=99 then raymond{i}=.;` `end;` `run;`	`foreach i in var1 var2 var3 var4 {` `replace `i' = . if `i' == 99` `}` NOTE: Notice that the quote to the left of the letter " i " is a left quote (`). The left quote is located at the top of your keyboard next to the " ! 1 " key. In this example i is a local macro variable that exists only for the duration of the foreach command so it does not need to be dropped like the variable i in the SAS code.
`label age="age in years"` `height="height in inches";`	`label var age "age in years"` `label var height "height in inches"`
Define a format: `proc format;` `value yesno` `1="yes"` `2="no";` `run;` `data newer;` `set newer;` `format smokes yesno.;` `run;`	Define a format. These are called "value labels": `label define yesno 1 "yes" /*` `*/ 2 "no"` Assign the value label to a variable: `label value smokes yesno`
Assign formats defined by SAS to a variable: `format interview_date mmddyy8.;`	Assign formats defined by Stata to a variable: `format interview_date %n/d/y` NOTE: The "n" in "%n/d/y" stands for "number of the month".
`title "Nutritional Intakes ages 12-18";`	Since the Results window/log file is both the log and the Output window Stata does not need a title statement. Titling can be accomplished with a comment. `/* Nutritional Intakes ages 12-18 */`
`proc sort data=new out=newer;` `by id;` `run;`	`sort id`
`proc transpose data=new` `(keep=age edu rel sex id lineno)` `out=tr_new;` `by id;` `run;`	`reshape long age edu rel sex, ///` `i(id) j(lineno)`

SAS	Stata
`` data newer1; set newer; by id; if first.id=1 then f_num=1; if first.id=1 and last.id=1 then s_num=1; if last.id=1 then l_num=1; run; ```	``` by id: gen f_num=1 if _n==1 by id: gen s_num=1 if _n==1 & _N==1 by id: gen l_num=1 if _n==_N ``` Stata's "_n" is equivalent to SAS's "_n_" in that it is equal to the observation number; but when inside a by command "_n" is equal to 1 for the first observation of the by-group, 2 for the second observation of the by-group, etc. Stata's "_N" is equal to the number of observations in the data set except in a by-command when it is equal to the total number of observations in the by-group.
Count the number of boys within an id by-group: ``` data new_c; set newer; by id; retain count 0; if first.id then count=0; if gender=1 and age<=18 then count=count+1 run; ```	Count the number of boys by id: ``` by id: gen count=sum(gender==1 & /// age<=18) ``` The sum function creates a running sum of the expression inside it.
`` data both; merge new(in=a) in8.newer(in=b); by id; if a=1 and b=1; run; ```	``` merge id using "d:\mydata\newer.dta" keep if _merge==3 ``` Stata automatically creates the variable "_merge" after a merge. Stata will not merge on another data set if _merge already exists on one of the data sets. The data set in memory is the "master" data set. The data set that is being merged on is the "using" data set. Unlike SAS, variables shared by the master data set and the using data set will not be updated (values overwritten) by the using data set. Like SAS, the formats, labels, and informats of variables shared by the master data set and the using data set will be defined by the master data set. Remember that the master always wins.
Concatenate two data sets: ``` data both; set new in8.newer; run; ```	``` append using "d:\mydata\newer.dta" ```

SAS	Stata
Sort data sets in order to prepare them for a merge: Sort permanently stored data sets and create new, sorted copies in the WORK library: `proc sort data=in8.individual out=indiv; by id; run;` `proc sort data=in8.household out=house; by id; run;` `data temp2; merge house(in=a) indiv(in=b); by id; run;`	Sort data sets in order to prepare them for a merge: Create a local macro variable to represent a filename for Stata to use in temporarily storing a data file on the computer's hard drive if requested to do so later: `tempfile indiv` `use "d:\mydata\individual.dta"` `sort id` Save the data set that is currently in memory to a temporary filename in Stata's temp directory. This file will be deleted when Stata is exited just like a data set in SAS's work library: `save `indiv'` `use "d:\mydata\household.dta"` `sort id` `merge id using `indiv'`
Note: Stata commands are partially underlined to show the minimum characters that need to be typed for Stata to recognize that command. Not all commands can be abbreviated.	

CONCLUSION

Stata can be learned using your SAS knowledge, but it does require knowing that you need to have in mind the differences between Stata and SAS. There will be times when you type a semi-colon at the end of the line or forget to use the double equals signs (==) when testing for equality, but with practice you will become proficient. I hope this paper helps you learn Stata and saves you many hours of frustration that you would have encountered otherwise.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Dan Blanchette
Carolina Population Center
University of North Carolina at Chapel Hill
CB# 8120, University Square
123 West Franklin St.
Chapel Hill NC 27516-2524

Work Phone: 919-966-1714
Fax: 919-966-6638
Email: Dan_Blanchette@unc.edu
Web: http://www.cpc.unc.edu/