Vanderbilt University
Leadership, Policy and Organizations
Class Number 9951
Fall 2017

Programming in Stata

The three virtues of a computer programmer are laziness, impatience, and hubris.

Laziness: The programmer wants to write as little code as is humanly possible.

Impatience The programmer does not have the patience to undertake a tedious task.

Hubris: The programmer is proud enough to believe that she can make the computer accomplish seemingly impossible tasks.

What's a macro? A way of storing information in Stata.

Why? Simplification. Lots of times we use lists of things. Say we need to use a list of terms that would influence college choice. This could be financial, academic, and family influences. We choose indicators to represent variables in each of these areas. What if we change one of these? We could change it each and every time, or if we had it stored in a macro we change it just once.

Macros are also used so that commands don't need to be repeated again and again, and instead can be written just once. This cuts down on mistakes and allows the analyst to focus on the analysis. The whole goal here is to get the computer to do the boring (repetitive) tasks, while the analyst does the interesting (analytical and interpretive) tasks.

There are two types of macos in Stata, local and global macros. Global macros should basically never be used.

So, let's do a macro: this macro will contain two variables from the plans dataset, math and reading test scores

```
local tests bynels2m bynels2r
```

What can we do now that we have a macro? Any command that can be run on the object can now be run on the macro. However, the macro must be referenced corectly. Referring to the macro without quotes will result in an error:

```
summarize tests
```

Why didn't this work? Without proper specification, a macro can not be accessed. The macro must be *dereferenced*. For STATA to know it's dealing with a macro, you must put it in single quotes, meaning that you start with the left tick (‘) and close with the apostrophe (’). Most of the curse words directed at STATA have come about as a result of this syntax. To use our macro, we would do the following:

```
summarize ‘tests’
```

Notice the construction of the quotes.

Quick Exercise

Create a macro that contains two variables. Run a summarize command on the macro.

A Note on Macros and Do-Files

When you run a do file with a macro, Stata will hold that macro in memory only while the do file is running. After it stops, the macro is dropped. This is important. Say you had a do file with a local named `family`, because it contained variables relating to a student's family. After running your do file, you'd like to summarize the family variables.

```
. sum `family´
```

You'll get back an error message because the `family` macro is no longer held in memory. For this reason, when using macros, it's a good idea to run the do file as a whole each time, instead of just running pieces of it.

Scalars

In the language of matrix algebra, a scalar is a single number. In STATA a scalar is a value that can only hold one value at a time. The value can be numeric or a character.

To define a scalar, use the following syntax:

```
scalar pi=3.14159
```

More usefully we can define scalars to take on the value of a result. For instance, to calculate a standardized transformation of the variable `income` we could do the following:

```
summarize income
scalar mean_income=r(mean)
scalar sd_income=r(sd)
gen stand_income = (income-mean_income)/sd_income
```

Scalars are also quite useful if you have a constant in a do file that you may wish to change. For instance, if you'd like to limit your analysis to a certain age group, but you might change that age group as you go through different iterations.

Quick Exercise

Generate scalars for a variable's sum and a variable's total number of units from the plans dataset. Divide the sum by the total number of units to obtain the mean.

The *varlist* Concept

A varlist is a list of variables (of all things). Say for instance you wanted a local that was equal to just data elements that were in the base year. We know from NCES nomenclature that all base year data elements in ELS are preceded by "by". We can use this, plus the wild card operator *, to create a varlist in the following way:

```
local bydata by*
```

This tells STATA to include every variable in the local bydata that begins with by.

Say you wanted to create a local that included the first five variables in the dataset. This can be done using the - as part of the command:

```
local first_five stu_id-f1sch_id
```

If you wanted every variable that had ses, and you knew that variables could only have one letter or number at the end, you could do something like this:

```
local myses *ses?
```

The *Numlist* concept

A numlist is a way of constructing a pattern of numbers. Stata recognizes several

Quick Exercise

Generate a varlist that contains only nels related variables, without naming the variables themselves.

Loops

A loop construct is the basic stepping stone to a life of laziness, impatience and hubris.

All loop constructs follow the same basic format:

```
 (A pattern goes here) {
(A series of commands for each step in the pattern goes here)
}
```

Note the braces: these always denote the beginning and end of a loop. The brace must follow the pattern command, and must always be closed after the body of the loop is complete.

With a loop construct, if you can figure out the underlying set of commands that you'd like to repeat, and if you can figure out the pattern that you'd like to apply them, you can simplify some pretty daunting tasks down to something rather simple. There are three basic ways to run loops in STATA: the `forvalues, foreach` and `while commands`.

Here's an example: Missing data, as you probably know, are a hassle when working with NCES datasets. They can be listed as -4, -8, or -9. Replacing this for every single variable in your dataset with a . would be time consuming and error prone. The following loop structure (which I will explain later) can accomplish it for you in just a few lines of code.

```
   foreach myvar of varlist stu_id-f1psepln{
foreach i of numlist= -4 -8 -9{
replace ‘myvar’=.  if ‘myvar’==-‘i’
}
}
```

The forvalues structure

The `forvalue` command tells STATA to execute the series of commands within the braces in a numerical format defined by a numlist.

The general structure of a forvalues command is:

foreach [local_name] of [number pattern]
(run the following commands on [local_name])

Here's a simple example:

```
   forvalues i = 1/10{
di ‘‘This is number ‘i’’’
}
```

In the example above, I defined the placeholder macro i to be equal to the numlist 1-10, starting at 1 and moving up by one for each run through the loop. The braces define the body of the loop. The command is a simple print command, asking STATA to display the text and the value of the placeholder macro i.

There are any number of applications where this could be useful. Say that you had data for test scores, and you wanted to regress test scores for each grade separately on a set of school characteristics that you had cleverly stored in a macro named `school`

```
   forvalues i = 1/6{
reg test‘i’ ‘school’ }
```

A more complex example is to convert the date of birth variable into an age, and then convert the result into a series of dummy variables for 14, 15, 16, 17 or 18 years old(you'll need to download and install the nsplit command).

nsplit bydob_p, digits (4 2) gen (newdobyr newdobm)

gen myage= 2002-newdobyr

forvalues i = 14/18 {
gen age‘i’=0

```
replace age‘i’=1 if myage==‘i’
}
```

Quick Exercise

Write a loop that counts from 1 to 100 by 3's. `help forvalues` is your friend here.

The foreach structure

The foreach structure is a more general version of the fovralues command. The general pattern for a foreach structure is:

```
foreach [local_name] of [varlist, local numlist, etc] {
(run the following commands on [local_name])
}
```

In the example on missing data, I used a foreach command to recode the variables. Let's use one now to standardize two test variables by subtracting the mean and dividing by 2 times their standard deviation (which is recommended by many statisticians).

```
foreach test of *nels* {
sum ‘test’
gen stand_‘test’=(‘test’-r(mean))/(2*r(sd))
}
```

Quick Exercise

Create a macro that contains only base year variables, with the exception of the two test variables (bynels2m and bynels2r). Write a loop that tabulates every variable in this macro.

The while command

The while command is a little outdated. It used to be the main way to construct loops in Stata, but the forvalues and foreach command have since superseded it i in most cases. However, it can still be useful, mainly when you're running complex code that you want to stop if something bad happens.

The general format of the while command is:

```
while (a condition is true) {
(run these commands)
}
```

So, we can repeat the counting program from above, but use the while command:

```
local i = 1
while i< 11 {
“I have not yet reached 10, instead the counter is now ‘i’ ”
local i= ‘i’+1
}
```

Quick Exercise Repeat the forvalues exercise, using the while command.

Nested Loops

You can run loops within loops, which is actually a very powerful function. Here's a simple example:

```
   forvalues i = 1/10 {
for values j =1/10 {
di ‘‘This is outer loop, inner loop ‘i’’’ }
}
```

The motivating example on missing data uses a nested loop structure. The outer loop consists of all of the variables, while the inner loop iterates over the possible missing value codes (-4,-8,-9).

Writing loops

The important thing about writing loops is to recognize patterns in your coding early on, and make sure that you get the underlying structure right first. The way to think about this is 0, 1,...1 million. That is, the core code either doesn't work (0) or works (1). If you can do it once, you can do it a million times. Start out by writing the simplest possible version, and making sure it works. Then slowly build the loop structure around the code.

Debugging loops

Your best friend here is `set trace=on`. Also, I've found STATA's foreach, forvalues commands to be really tricky. If you know that your "core" code is running fine, the main problem with loops is probably going to be in the syntax for your forvalues or foreach command.

It's also a really good idea to build in sanity checks if you're running complex programs. Small mistakes can really compound when you're using these powerful tools.

In Class Exercise

Use the plans dataset. Create an algorithm that will convert a continous variable into a series of dummy variables, one dummy variable for each qunitile. Now, run this for every continuous variable in the dataset, using a loop structure.