University of California, Santa Cruz
Department of Economics
ECON 294A (Fall 2014)- Stata Lab
Instructor: Manuel Barron[1]

Stata Programming 1

1 Introduction

In the first three lab sessions we learned how to use Stata to manage variables and datasets. In this and the next lecture we will cover some basics of programming in Stata. From my point of view, programming consists in writing code that will repeat commands a number of time. Today and next week we'll cover some toy examples, so that you get familiar with the types of loops and some of the other basic programming tools that are available in Stata.

Starting on week 6 of the course we will start applying econometric tools, and we'll start using real world data.

2 Macros

In Stata, *macros* are shorthand for any type of expressions. Stata has two types of macros: local and global macros. We'll start with global macros and will cover local macros afterwards. I'll refer to local macros as "locals" and to global macros as "globals". The main difference is that local macros stay in memory only temporarily (details in the next section), while global macros stay in memory until you clear them from memory. It sounds convenient, but it can be a bit risky.

2.1 Global Macros

To define a global macro, we type "global" , then the macro name, and then the contents of the macro, which can be the name of a variable, a list of variables, a name, a number, or anything really. To use the macro in our code, we type the name of the macro preceded by a dollar sign: $macroname. This example shows how macros work:

[1]Please contact me with any comments (typos, errors, unclear stuff, or other suggestions on how to improve these notes) at mbarron4 [at] ucsc

do-file

```
sysuse auto, clear

global covariates "mpg headroom"

summarize $covariates
* Stata replaces $covariates by "mpg headroom"
* Stata executes: summarize mpg headroom

regress price $covariates
* Stata replaces $covariates by "mpg headroom"
* Stata executes regress price mpg headroom

regress price covariates
* There is no $ before covariates, so Stata looks for a variable named
* covariates. Since that variable doesn't exist, Stata reports an error.
```

We can define anything as a macro, even if its not a variable name. Stata just keeps the macro in memory without evaluating its contents until we actually use the macro to execute a command.

do-file

```
global covariates prices
summarize $covariates
```

Here are some more examples of the use of global macros. In particular, notice that if we don't type "clear all", Stata keeps the "covariates" macro we had defined in the previous paragraphs. This is partly convenient, but it is also risky. We need to be sure that the contents of the global "covariates" are what we think they are. This is not so salient in small examples like the ones we see in class, but it will become important when working with long do-files or with multiple do-files in a larger project, where two different macros may have the same name.

do-file

```
* We can use more than one global in the same command:
global covariates2 "weight length"

regress price $covariates $covariates2

* We can define a local with another global:
global covariates3 "$covariates $covariates2"
regress price $covariates3

* We can save a whole expression as a global:
global regression "reg price $covariates3 if foreign==0"
$regression
```

2.2 Local Macros

To define a local macro, type "local" then the macro name, and then the contents. When we refer to the local macro, we need to place the name within ‘ ’. The opening bracket is the key to the left of the number 1 (top left in your keyboard). The closing symbol is the single quotation mark that usually appears next to the enter (PC) or return (Mac) key.

Local macros stay in memory temporarily, only while executing a given set of commands (hence the name local). When you run a do-file using local macros, local macros must be defined within the section you wish to execute. If you execute the command lines that define the local macros first, and then highlight and execute the commands lines that use those local macros, Stata will have dumped the macros from memory.

do-file

```
clear all
sysuse auto, clear
local covariates "mpg headroom"

summarize ‘covariates’
```

OK, now lets just run a single line calling the covariates local macro to see what happens.

do-file

```
regress price ‘covariates’
```

Since we did not define the local macro covariates when we executed the do-file, Stata replaces ‘covariates’ with empty space, so it executes "regress price", which amounts to running the regression of price on a constant.

do-title

```
* Now lets display the contents of the local "covariates"
display "‘covariates’"

* Now, lets use the contents of the local "covariates" in commands:

summarize ‘covariates’

regress price ‘covariates’

* We can use more than one local in the same command:
local covariates2 "weight length":

regress price ‘covariates’ ‘covariates2’

* We can define a local with another local:
local covariates3 ‘covariates’ ‘covariates2’
regress price ‘covariates3’

* We can save a whole expression as a local:
local regression "reg price ‘covariates3’ if foreign==0"
‘regression’
```

Normally, you will use locals to store covariates when you want to run many regressions with the same covariates. Some people use them to store conditions as well.

do-file

```
local covariates "mpg weight length"
local if "if foreign==1"

regress price weight
regress price weight ‘covariates’
regress price weight ‘covariates’ ‘if’


regress headroom weight
regress headroom weight ‘covariates’
regress headroom weight ‘covariates’ ‘if’
```

Be careful, always check the output to be sure that your locals were used as you intended them to be used.

3 Control Structures: (*Loops)*

Stata has some very useful control structures, more commonly known as *loops*. When we start using loops is when you will see that there are plenty of ways of doing the same task. It will largely depend on the type of loop with which you are more comfortable.

Loops in Stata generally have the following structure:

[control structure name] [local macro] [list of values or variables] [opening bracket]

← space → lines of code over which Stata loops through

[closing bracket]

The space is important because it makes it easier for a human to understand what's going on in the loop (its all the same to the computer). This will become clearer when we start nesting loops within loops.

3.1 foreach

The *foreach* loop allows you to run a command (or a set of commands) a number of times following the pattern specified by the user. For example

do-file

```
foreach y in price headroom {
     regress ‘y’ weight
}

foreach x in weight mpg foreign {
     regress price ‘x’
}
```

Note that the stubs over which the loop runs need not be variables

do-file

```
foreach j in 11 12 13 {
     sum x‘j’
}
```

3.2 forvalues

forvalues executes the commands for a range of values specified by the user. So, instead of typing,

do-file

```
gen x1 = rnormal()
gen x2 = rnormal()
gen epsilon = rnormal()

gen y = 0.5 + x1 + x2 + epsilon
reg y x1 x2
```

we can generate the x variables with a loop:

do-file

```
drop x*

forvalues i = 1(1)100 {
     generate x‘i’ =  rnormal()
}
```

3.3 while

do-file

```
reg y x1
outreg2 using tables/table1, replace

local i = 2
while ‘i’<= 100{
     reg y x‘i’
     outreg2 using tables/table1, append
     local i = ‘i’ + 1
}
```

Let’s open the resulting table. What happened? Well, since each time the covariate has a different name, Stata treats it as a different covariate (as it should do). So each time we run the command Stata adds a row to the table. Since we probably want to compare the coefficients, one way around it is to create a fake variable that takes on the value of x1 in the first run, x2 in the second,

do-file

```
g random_covariate = x1
reg y random_covariate
outreg2 using tables/table1, replace

local i = 2
while `i'<= 100{
     replace random_covariate = x`i'
     reg y random_covariate
     outreg2 using tables/table1, append
     local i = `i' + 1
}
```

The following example shows a mistake I've made tons of times. Can you see what's wrong with this? If I gave you the following instructions, what would you do in the first run of the loop? what would you do in the second?

do-file

```
local i = 1
while `i'<= 100{
     reg y x`i'
}
```

4 Nested Loops

You can also write a loop (or multiple loops) within a loop. This is useful in cases in which there are two things changing. We'll see more of these next week. By now I just want you to know that they exist and may become handy. Note how the spacing helps identify what is going on in which loop.

do-file

```
foreach y in price headroom {
     foreach x in weight mpg foreign {
          regress `y' `x'
     }
}
```

As an exercise, write down each of the regress commands that stata runs, in the order that this loop implies.

5 Example

For this example we will use the *predict* command, which generates the predicted values of the outcome variable that result from running a regression. (see help predict).

We want to see if the model has good "out of sample" prediction power (i.e., if it can predict fairly well the observed values of observations that are not in the sample). For this,

1. Estimate the model with all observations except the first one.
2. Use the model to predict the outcome variable in the first observation
3. Store the predicted value for the first observation
4. Proceed to the next observation in the dataset

Once you are done with all the observations in the dataset, use the commands we've seen to compare the predicted values of the data to the observed values.

6 Exercises

With the nlsw88.dta dataset:

- Use a foreach loop to run a regression of wage on hours (reg wage hours) by marital status. Use outreg2 to save your results to an excel file.
- Use a while loop to run a regression of wage on hours for each occupation in the sample. Use outreg2 to save your results to an excel file. (Don't forget to update the counter at the end of your loop!)
- Use a nested loop to run a regression of wage on hours for each occupation and by marital status. Use outreg2 to save your results to an excel file.