MPP-C6: Statistics II

Programming with Stata

Max Callaghan

Hertie School of Governance

?? February 2015

Outline

- Why Program?
- Reproducible Research
- Programming in Stata to accomplish difficult tasks and simplify repetitive tasks

Why Program?

- Computers can perform calculations faster and more accurately than humans
- When a task is difficult or repetitive, if often makes sense to instruct the computer to do it
- When those instructions are written down, we and others can see exactly what has been done
  - It's easier to repeat the work
  - It's easier to spot errors
  - It's easier to repeat the work, changing just one detail, without doing every subsequent step again.

Reproducible Research

- “The standard of reproducibility calls for the data and the computer code used to analyse the data to be made available to others” [**?**]
- Literate programming ties together data, code and the actual research output, enhancing reproducibility

Reproducible Research with Stata

- What do we already do that helps to keep our research reproducible?
- How do we ensure that what's in our research output reflects the calculations we report making?

Reproducible Research with Stata

- The ideal for perfect reproducibility would be to have a single document that contains instructions for performing calculations as well as for producing the research output we present.
- This is not possible in Stata without LaTeX but we can at least make the way we include Stata output in Word documents more systematic

Reproducible Research with Stata

[Include instructions to do this, with a simple example of output that changes to reflect a change in a do file]

- Stata basics
- Directory structure
- Reading data
- Transforming and processing data
- Presenting results with Stata

Stata Basics

- Pointing and clicking is fine for exploring data
- The command line is fine for trying out commands
- Anything you want to be able to reproduce, you should put in the do file

Stata Basics

Writing a good do file

A good do file should be readable by humans as well as computers.

- Use comments to explain what each line is doing
- Empty lines are free, space makes your code easier to read
- Use meaningful names when you create them, and write them consistently (variable_name, variableName or VariableName)
- Follow indentation conventions e.g.

```
forvalues i in 1/5 {
  display `i'
}
```

Stata commands are preprogrammed functions that take information we give to them, do something with the information, then output something. We pass information to commands with *arguments* and *options*.
If you are not sure how to use a command you can get help by typing "help" and then the name of the command. For example,

```
help regress
```

will take you to the regress command's manual

Stata Basics

Reading the Stata manual

The Syntax `regress` *depvar [indepvars] [if] [in] [weight] [,options]* describes the basic use of the command

- Pay attention to the order of the arguments. This is how Stata knows which arguments are which
- Items in square brackets are optional
- Options come after a comma. Possible options are described in the help file
- You don't always need to set a lot of options, but you should pay atention to what the defaults are

If you don't know the command you want to use, you will have try and describe your problem to google.

Directory Structure

File paths tell the computer where to read and write information. They differ between Windows and Mac/Linux. (\or /)

- Absolute paths start from the top of the tree and specify each subdirectory until the file e.g. "C:bla\bla\data.dta" (or "/bla/bla/data.dta", or even "http://bla.com/data.dta")
- Relative paths start from the current working directory

Directory Structure

If you refer to more than one resource, it makes sense to set the working directory at the top of your do file and use relative paths. You'll want to think about how you structure your directory so that you can access items easily.

- If your do file produces output, think about where you want to save it so that you can access it automatically with another program
- This also allows you to change computers easily. Dropbox is an easy way to carry entire directories between computers. Git/Github is even better as it incorporates version control.

Reading Data

Data doesn't always come in nicely formatted dta files. Sometimes you have a data source or sources in files that aren't set up for stata to read. You often have to do a bit of work to get things into the format you want: the more of that work is recorded the better.

Reading Data

Some things to pay attention to when reading data

- Keep an original copy of the data exactly as you found it, if you make changes, save to a new name
- Try and make changes with Stata in your do file. If you have to change in Excel, write down what you did
- Check the data has been imported properly before you use it
    - You may need to specify what character signifies missing values in your data
    - You might need to specify the delimiter in csv or txt files
    -

Data types

Stata stores data in various different data types. Each variable can only be one data type. Some operations can only be done on data of certain types

- Numeric data can be stored in various degrees of precision: check -help data types- for more information
- Anything with non-numeric characters will be saved as a string (text)

It's easy for data to arrive in the wrong format when we read from other sources.

We can use -tostring- and -destring- to convert between string and numeric data, as well as -encode- to create a numeric variable out of non-numeric string data.

Programming with Stata

Macros

Stata already helps us to perform calculations quickly, but we can speed up how we interact with Stata by using some simple programming to avoid repetition. The most simple concept is storing something as a macro.

- Macros can tie a name to some text
- `local controls age gender incCat` ties the word ”controls” to “age gender inc_cat”
- Now, everytime we type `` `controls' ``, stata understands “age gender inc_cat” (note the backtick, which is under the tilde)
- If we type “age gender inc_cat” a lot, then we save ourselves time by defining it once and referring to the definition the other times
- This also reduces the risk of errors. Why?

Loops

Loops can speed up our work by repeating tasks while changing one thing.

```
forvalues i in 1/20 {
  display "`i'"
}
```

```
foreach control of local controls {
  display "`control'"
}
```

We can loops within loops, and apply conditions within loops. It's often helpful to try the operation once outside the loop, and it's often helpful to display the iterator so that you know what is happening in your loop. Can you write a fizz-buzz loop?

Programming with Stata

The -by- command

Reading Data

Example

In our first example, we want to read data from a source we found on the web (link). We want to look at some crime statistics by borough, and we'll start with fear of crime by borough.
We copy it onto our computer and use -import excel- to read into stata the data we want

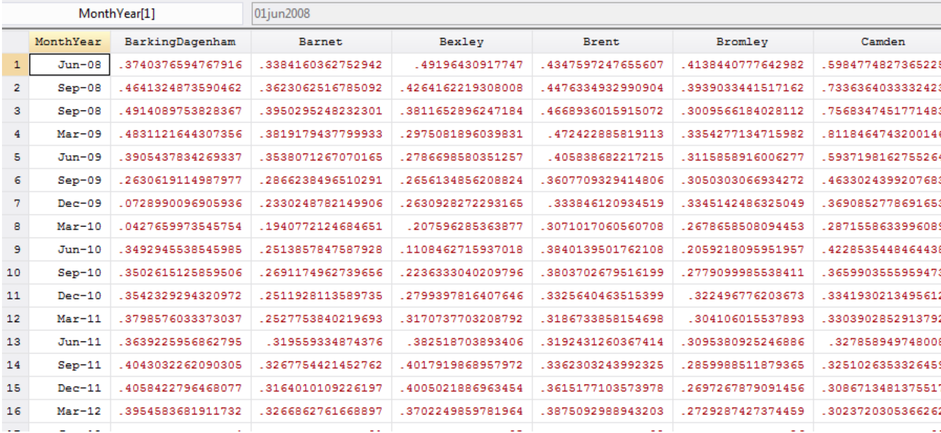
Interactive Stata Example

```
. *** Copy the dataset onto our computer (if it doesn't exist already
. capture confirm file data/crime.xls

. if _rc==601 {
.         copy https://files.datapress.com/london/dataset/metropolitan-police-service-recorde
> -figures.xls ///
>                 data/crime.xls, replace
. }

. import excel data/crime.xls, ///
>         sheet("Fear of Crime-Borough") /// Tell stata which sheet to import
>         cellrange(A3:AG31) /// Specify the cells we want to import
>         firstrow // tell stata that variable names are in the first row

.
. cap rename BarkingandDagenham BarkingDagenham // Inconsistent name

. cap rename A MonthYear // Merged cell caused problem
```

Transforming data

In stata, columns are called variables and rows are observations. We don't always receive data like that.

How would you reformat this data? What variables do we have?



MonthYear[1] 01jun2008

	MonthYear	BarkingDagenham	Barnet	Bexley	Brent	Bromley	Camden
1	Jun-08	.3740376594767916	.3384160362752942	.49196430917747	.4347597247655607	.4138440777642982	.598477482736522
2	Sep-08	.4641324873590462	.3623062516785092	.4264162219308008	.4476334932990904	.3939033441517162	.733636403333242
3	Sep-08	.4914089753828367	.3950295248232301	.3811652896247184	.4668936015915072	.3009566184028112	.756834745177148
4	Mar-09	.4831121644307356	.3819179437799933	.2975081896039831	.472422885819113	.3354277134715982	.811846474320014
5	Jun-09	.3905437834269337	.3538071267070165	.2786698580351257	.405838682217215	.3115858916006277	.593719816275526
6	Sep-09	.2630619114987977	.2866238496510291	.2656134856208824	.3607709329414806	.3050303066934272	.463302439920768
7	Dec-09	.0728990096905936	.2330248782149906	.2630928272293165	.333846120934519	.3345142486325049	.369085277869165
8	Mar-10	.0427659973545754	.1940772124684651	.207596285363877	.3071017060560708	.2678658508094453	.287155863399608
9	Jun-10	.3492945538545985	.2513857847587928	.1108462715937018	.3840139501762108	.2059218095951957	.422853544846443
10	Sep-10	.3502615125859506	.2691174962739656	.2236333040209796	.3803702679516199	.2779099985538411	.365990355595947
11	Dec-10	.3542329294320972	.2511928113589735	.2799397816407646	.3325640463515399	.322496776203673	.334193021349561
12	Mar-11	.3798576033373037	.2527753840219693	.3170737703208792	.3186733858154698	.304106015537893	.330390285291379
13	Jun-11	.3639225956862795	.319559334874376	.382518703893406	.3192431260367414	.3095380925246886	.32785894974800
14	Sep-11	.4043032262090305	.3267754421452762	.4017919868957972	.3362303243992325	.2859988511879365	.325102635332645
15	Dec-11	.4058422796468077	.3164010109226197	.4005021886963454	.3615177103573978	.2697267879091456	.308671348137551
16	Mar-12	.3954583681911732	.3266862761668897	.3702249859781964	.3875092988943203	.2729287427374459	.302372030536626

Transforming Data

Our data is too wide: we can use the -reshape- command to switch between “wide” and “long” formats. First we need to give the borough coloums a common prefix, then we can tell reshape to create a new variable. Attempting this also lets us know that we have two records for September 2008

Interactive Stata Example

```
. rename(BarkingDagenham-Westminster) foCrime=

. keep if MonthYear > td(2sep2008) | MonthYear < td(2aug2008) // There were 2 values for sep
(2 observations deleted)

. reshape long foCrime, i(MonthYear) j(Borough) string
(note: j = BarkingDagenham Barnet Bexley Brent Bromley Camden Croydon Ealing Enfield Greenwic
>  Hounslow Islington KensingtonandChelsea KingstonuponThames Lambeth Lewisham Merton Newham
> Forest Wandsworth Westminster)

Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                       26   ->     832
Number of variables                  33   ->       3
j variable (32 values)                    ->   Borough
xij variables:
foCrimeBarkingDagenham foCrimeBarnet ... foCrimeWestminster->foCrime
-----------------------------------------------------------------------------


.
```

Data types

The reason why Stata thinks our fear of crime variable is a string is that some of the values have % characters (probably an artefact of inconsistent Excel cell formatting).

We can remove these using the -subinstr- command

Interactive Stata Example

```
. gen foCrime2 = subinstr(foCrime,"%","",.)
(2,219 missing values generated)

. destring foCrime2, replace
foCrime2 has all characters numeric; replaced as double
(2219 missing values generated)

.
```

-

References

Roger D. Peng.
Reproducible research in computational science.
Science, 334(6060):1226–1227, 2011.