Advanced Topics in Stata

Kerry L. Papps

1. Overview

- Basic commands for writing do-files
- Accessing automatically-saved results generated by Stata commands
- Matrices
- Macros
- Loops
- Writing programmes
- Ado-files

2. Comment on notation used

- Consider the following syntax description:

  `list [`*varlist*`] [in` *range*`]`

  – Text in `typewriter-style font` should be typed exactly as it appears (although there are possibilities for abbreviation).

  – *Italicised* text should be replaced by desired variable names *etc*.

  – Square brackets (*i.e.* `[]`) enclose optional Stata commands (do not actually type these).

- This notation is consistent with notation in Stata Help menu and manuals.

3. Writing do-files

- These commands are normally used in Stata do-files (although most can also be used interactively).
- We will write do-files in the Stata do-file editor. (Go to Window → Do-File Editor or click  .)
- Type each line of code on a new line of the do-file.
- Alternatively, to use a semi-colon (;) as the command delimiter, start the do-file with the command:

```
#delimit ;
```

- This allows multiple-line commands. To return to using the Return key at the end of each line, type:

```
#delimit cr
```

4. Writing do-files (cont.)

- To prevent Stata from pausing each time the Results window is full of output, type:

  `set more off`

- To execute a do-file without presenting the results of any output, use:

  `run` *dofilename*

- To execute *any* Stata command while suppressing the output, use:

  `quietly` *command*

5. Types of Stata commands

- Stata commands (and new commands that you and others write) can be classified as follows:
  - r-class: General commands such as `summarize`. Results are returned in `r()` and generally must be used before executing more commands.
  - e-class: Estimation commands such as `regress`, `logistic` *etc*., that fit statistical models. Results are returned in `e()` and remain there until the next model is estimated.
  - s-class: Programming commands that assist in parsing. These commands are relatively rare. Results are returned in `s()`.

6. Types of Stata commands (cont.)

- n-class: Commands that do not save results at all, such as `generate` and `replace`.
- c-class: Values of system parameters and settings and certain constants, such as the value of π, which are contained in `c()`.

7. Accessing returned values

- `return list`, `ereturn list`, `sreturn list` and `creturn list` return all the values contained in the `r()`, `e()`, `s()` and `c()` vectors, respectively.
- For example, after using `summarize`, `r()` will contain `r(N)`, `r(mean)`, `r(sd)`, `r(sum)` *etc*.
- Elements of each of the vectors can be used when creating new variables. They can also be saved as macros (see later section).
- `e(sample)` is a useful function that records the observations used in the most recent model, *e.g.*:

```
summarize varlist if e(sample)==1
```

8. Accessing returned values (cont.)

- Although coefficients and standard errors from the most recent model are saved in `e()`, it is quicker to refer to them by using `_b[`*varname*`]` and `_se[`*varname*`]`, respectively.
- For example:

```
gen fitvals = educ*_b[educ] +
  _cons*_b[_cons]
```

EXERCISE 1

9. Regression results

- Note that all solutions to the exercises are contained in:
  `http://people.bath.ac.uk/klp33/advanced_stata.do`
- Start a do-file and change the working directory to a folder of your choice (*myfolder*) using:
  `cd c:\`*myfolder*
- Open (with `use`) the file:
  `http://people.bath.ac.uk/klp33/advanced_stata_data.dta`

EXERCISE 1 (cont.)

10. Regression results

- Create the total crime rate (`totcrimerate`), imprisonment rate (`imprisrate`) and execution rate (`execrate`) by dividing `totcrime`, `impris` and `exec`, respectively, by `population` and multiplying by 100,000.
- Create the unemployment rate (`unemplrate`) by dividing `unempl` by `lf` and multiplying by 100.
- Create `youthperc` by dividing `youthpop` by `population` and multiplying by 100.
- Create `year2` by squaring `year`.
- Regress `totcrimerate` on `inc`, `unemplrate`, `imprisrate`, `execrate`, `youthperc`, `year` and `year2`.

EXERCISE 1 (cont.)

11. Regression results

- Look at the results that are saved in `e()` by using:
  `ereturn list`
- Create a variable that measures the (quadratic) trend in crime:

```
gen trend = _b[year]*year +
  _b[year2]*year2
```

- Plot this against time by using:

```
scatter trend year
```

- Save the modified dataset as "`Crime data`".

12. Creating matrices

- In addition to the following, a complete matrix language, Mata, is incorporated in Stata.
- Matrices are not stored in the spreadsheet.
- Matrices can be inputted manually using:

```
matrix [input] matname = (#[,#...][\
  #[,#...][\[...]]])
```

- For example, to create $\mathbf{A} = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ type:

```
matrix A = (1,2 \ 3,4)
```

13. Creating matrices (cont.)

- To create a matrix with existing variables as columns, type:

  `mkmat` *varlist*`[, matrix(`*matname*`)]`

- If the matrix option is omitted, the variables in *varlist* will be stored as separate column vectors with the same names as the variables.
- To create new matrices from existing matrices:

  `matrix [define]` *matname* `=` *exp*

14. Matrix operators and functions

- Some operators and functions that may be used in *exp*:
  - `+` means addition
  - `-` means subtraction or negation
  - `*` means multiplication
  - `/` means matrix division by a scalar
  - `'` means transpose
  - `#` means Kronecker product
  - `inv(`*matname*`)` gives the inverse of *matname*

15. Submatrices

- To obtain submatrices, type:

  `matrix` *newmat* `=` *oldmat*`[`*rowrange*`,` *colrange*`]`
- *rowrange* and *colrange* can be single numbers or ranges with start and finish positions separated by two periods.
- For example, to create a matrix **B** containing the second through fourth rows and first through fifth columns of **A**, type:

```
matrix B = A[2..4,1..5]
```

- To take all rows after the second, use three periods:

```
matrix B = A[2...,1..5]
```

16. Cross-product matrices

- To create cross-product matrices (`X'X`) it is convenient to use the following code:

  `matrix accum` *matname* `=` *varlist*`[, noconstant]`

- A constant will be added unless `noconstant` is specified.
- For example, `matrix accum XX = age educ` would create a 3×3 matrix of cross-products.

17. Managing matrices

- To list a matrix, type:

  `matrix list` *matname*
- To rename a matrix, type:

  `matrix rename` *oldname newname*
- To drop one or more matrices, type:

  `matrix drop [`*matlist*`]`

EXERCISE 2

18. Regression with matrices

- Start a new do-file and open "`Crime data.dta`".
- Suppose we wanted to perform the regression from Exercise 1 manually. Calculate the estimated coefficient vector: $\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$.
- To do this, first construct a general cross-product matrix **Z** by typing:

```
matrix accum Z = totcrimerate inc
  unemplrate imprisrate execrate
  youthperc year year2
```

- Display **Z** using `matrix list`.

EXERCISE 2 (cont.)

19. Regression with matrices

- Next, construct the matrix **X′X** by selecting all but the first row and column of **Z** and save it as `XX`.
- Construct **X′y** by selecting only the first column of **Z** below the first row and save it as `Xy`.
- Construct the vector `b` using the `matrix` command, the `inv()` function and the matrices `XX` and `Xy`.
- Display the contents of `b` using `matrix list` and verify that the coefficients are the same as those generated by `regress` in Exercise 1 (within acceptable rounding error limits).
- Save your do-file in the working directory.

20. Macros

- A macro is a string of characters (the macro name) that stands for another string of characters (the macro contents).
- Macros allow you to avoid unnecessary repetition in your code.
- More importantly, they are also the variables (or “building blocks”) of Stata programmes.
- Macros are classified as either global or local.

21. Macro assignment

- Global macros exist for the remainder of the Stata session and are defined using:

  `global` *gblname* `[`*exp*`]`
- Local macros exist solely within a particular programme or do-file:

  `local` *lclname* `[`*exp*`]`
- When *exp* is enclosed in double quotes, it is treated as a string; when *exp* begins with =, it is evaluated as an expression.

22. Macro assignment (cont.)

- For example, consider:

```
local problem “2+2”
local solution = 2+2
```

- `problem` contains `2+2`, `solution` contains `4`.

23. Referring to macros

- To substitute the contents of a global macro, type the macro name preceded by `$`.
- To substitute the contents of a local macro, type the macro name enclosed in single quotes (`` `' ``).
- For example, the following are all equivalent once *gblname* and *lclname* have been defined as `newvar` using `global` and `local`, respectively:

```
gen newvar = oldvar
gen $gblname = oldvar
gen `lclname' = oldvar
```

24. Temporary variables

- `tempvar` creates a local macro with a name different to that of any variable. This can then be used to define a new variable. For example:

```
tempvar sumsq
gen `sumsq' = var1^2 + var2^2
```

- Temporary variables are dropped as soon as a programme terminates.
- Similarly, it is possible to define temporary files.

25. Manipulating macros

- `macro list` displays the names and contents of all defined macros.
- Note that local macros are stored with an underscore (_) at the beginning of their names.
- When working with multiple folders, global macros can be used to avoid typing full file names, *e.g.*:

```
global mypath "c:\Stata files"
use "$mypath\My Stata data"
```

26. Looping over items

- The `foreach` command allows one to repeat a sequence of commands over a set of variables:

```
foreach lclname of listtype list {
  Stata commands referring to `lclname'
}
```

- Stata repeatedly sets *lclname* equal to each element in *list* and executes the commands enclosed in braces.
- *lclname* is a local macro, so should be enclosed in single quotes when referred to within the braces.
- *listtype* may be: `local`, `global`, `varlist`, `newlist`, `numlist`.

27. Looping over items (cont.)

- With `local` and `global`, *list* should already be defined as a macro. For example:

```
local listname "age educ inc"
foreach var of local listname {
```

- With `varlist`, `newlist` and `numlist`, the actual list is written in the `foreach` line, *e.g.*:

```
foreach var of varlist age educ inc {
```

- `foreach` may also be used with mixed lists of variable names, numbers, strings *etc.*:

```
foreach x in educ 5.8 a b inc {
```

- You can nest any number of `foreach` loops (with unique local names) within each other.

28. Looping over values

- To loop over consecutive values, use:

  `forvalues` *lclname* `=` *range* `{`
- For example, to loop from 1 to 1000 in steps of 1, use:

```
forvalues i = 1/1000 {
```
- To loop from 1 to 1000 in steps of 2, use:

```
forvalues i = 1(2)1000 {
```
- This is quicker than `foreach` with `numlist` for a large number of regularly-spaced values.

29. More complex loops

- `while` allows one to repeat a series of commands as long as a particular restriction is true:

```
while exp {
  Stata commands
}
```

- For example:

```
local i "7 6 5 4 3 2 1"
while `i'>4 {
```

- This will only set `` `i' `` equal to 7, 6 and 5.
- Sometimes it is useful to refer to elements by their position in the list ("token"). This can be done with `tokenize`:

```
tokenize string
```

30. More complex loops (cont.)

- *string* can be a macro or a list of words.
- `` `1' `` will contain the first list item, `` `2' `` the second item and so on, *e.g.*:

```
local listname "age educ inc"
tokenize `listname'
```

- `` `1' `` will contain `age`, `` `2' `` `educ` and `` `3' `` `inc`.
- To work through each item in the list one at a time, use `macro shift` at the end of a loop, *e.g.*:

```
while "`1'" ~= "" {
  Commands using `1'
  macro shift
}
```

31. More complex loops (cont.)

- At each repetition, this will discard the contents of `` `1′ ``, shift `` `2′ `` to `` `1′ ``, `` `3′ `` to `` `2′ `` and so on.
- Where possible, use `foreach` instead of `while`.

EXERCISE 3

32. Using loops in regression

- Use `foreach` with `varlist` to create a loop that generates the rate per 100,000 people for each crime category and names the new variables by adding “rate” to the end of the old variable names.
- Save the updated dataset.
- Use `forvalues` to create a loop that repeats the regression from Exercise 1 (minus `imprisrate`) separately for observations with imprisonment rates in each interval of 50 between 0 and 250.
- Hint: use an `if` restriction with the regression after starting with the following line:

```
forvalues i = 50(50)250 {
```

33. Writing programmes

- To create your own Stata commands that can be executed repeatedly during a session, use the `program` command:

```
program progname
  args arg1 arg2…
  Commands using `arg1', `arg2' etc.
end
```

- `args` refers to the words that appear after *progname* whenever the programme is executed.

34. Writing programmes (cont.)

- For example, you could write a (pointless) programme that added two numbers together:

```
program mysum
   args a b
   local c = `a'+`b'
   display `c'
end
```

- Following this, `mysum` followed by two numbers can be used just like any other Stata command.
- For example, typing `mysum 3 9` would return the output `12`.

35. Writing programmes (cont.)

- If the number of arguments varies, use `syntax` instead of `args`.
- `syntax` stores all arguments in a single local macro.
- For example, to add any number of numbers together, use the following code (`anything` is one of three available format options):

36. Writing programmes (cont.)

```
program mysum
  syntax anything
  local c = 0
  foreach num of local anything {
    local c = `c'+`num'
  }
  display `c'
end
```

37. Writing programmes (cont.)

- To list all current programmes, type:
  `program dir`
- To drop a previously-defined programme, use:
  `program drop` *progname*
- By default, Stata does not display the individual lines of your programme as it executes them, however to debug a programme, it is useful to do so, using `set trace on`.
- `set trace off` undoes this command.

EXERCISE 4

38. Creating a programme

- Take the code that created the estimated coefficient vector `b` from Exercise 2 and turn it into a Stata programme called `myreg` that regresses any dependent variable on the set of 7 independent variables used.
- You should be able to invoke `myreg` by typing `myreg` *depvarname*.
- Hint: Use `args depvar` to create a macro called `depvar` and use this instead of `totcrimerate` in the existing code.
- Make sure that the `b` vector is displayed by the programme by using `matrix list b`.

EXERCISE 4 (cont.)

39. Creating a programme

- Check that `myreg` gives the same results as `regress` when a couple of different crime categories are used as the dependent variable.

40. Ado-files

- An ado-file ("automatic do-file") is a do-file that defines a Stata command. It has the file extension .ado.
- Not all Stata commands are defined by ado-files: some are built-in commands.
- The difference between a do-file and an ado-file is that when the name of the latter is typed as a Stata command, Stata will search for and run that file.
- For example, the programme `mysum` could be saved in `mysum.ado` and used in future sessions.
- Ado-files often have help (.hlp) files associated with them.

41. Ado-files (cont.)

- There are three main sources of ado-files:
  - Official updates from StataCorp.
  - User-written additions (*e.g.* from the Stata Journal).
  - Ado-files that you have written yourself.
- Stata stores these in different locations, which can be reviewed by typing `sysdir`.
- Official updates are saved in the folder associated with `UPDATES`.
- User-written additions are saved in the folder associated with `PLUS`.
- Ado-files written by yourself should be saved in the folder associated with `PERSONAL`.

42. Installing ado-files

- If you have an Internet connection, official updates and user-written ado-files can be installed easily.
- To install official updates, type:

```
update from http://www.stata.com
```

- Next, follow the recommendations in the Results window.
- Users on the University network should not need to do this as Stata is regularly updated.
- To install a specific user-written addition, type:

```
net from http://www.stata.com
```

- Next, click on one of the listed options and follow the links to locate the required file.

43. Installing ado-files (cont.)

- To search for an ado-file with an unknown name and location, type:

  `net search` *keywords*
- Equivalently, go to Help → Search and click "Search net resources".
- For example, `estout.ado` is a very convenient user-written ado-file that saves Stata regression output in a form that can be displayed in academic tables.
- Since network users do not have generally have access to the c:\ drive, they must first choose another location in which to save additional ado-files:

  `sysdir set PLUS` *yourfoldername*

44. Installing ado-files (cont.)

- Finally, to add an ado-file of your own, simply write the code defining a programme and save the file with the same name as the programme and the extension .ado in the folder associated with `PERSONAL`.
- Once again, network users will have to change the location of this folder with:

  `sysdir set PERSONAL` *yourfoldername*