Intermediate Stata Programming

Jacob Fowles
Assistant Professor
School of Public Affairs and Administration
Affiliated Faculty, CRMDA
University of Kansas
Jacob.fowles@ku.edu
Overview

• Stata's interface is designed to accommodate user groups of differing expertise
• Users of Stata (independent of degree, field, and tenure status) frequently do more much work than necessary
• Like most things in life, there are diminishing marginal returns to Stata technical training. . .
• But there are large benefits for most researchers in moving from "novice" to "intermediate"
Caveat Emptor

• If you currently:
  – Rely heavily on Stata's point-and-click interface
  – Don't typically use do-files
  – Have zero experience with any modern (1960s +) programming languages

• Then:
  – This talk will be less of an incremental step than a leap from the proverbial cliff, but
  – You can change your approach to statistical analysis in Stata before your (bad) habits become ingrained
Why Bother?

• Avoiding typographical errors and simple coding mistakes
• You can't make days longer, but you can make them more productive
• Reviewers and professors frequently (always?) suggest tweaks to empirical models that change every coefficient and associated statistic in every table
• Being good at something is good for your ego
A Typical Workflow

- Setup and Organization
- Cleaning Data
- Performing Analysis
- Presenting Findings
- Saving your work
### Directory Structures

- **Careful use of directories makes life easier**

<table>
<thead>
<tr>
<th>Directory</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>\project_1</td>
<td>Project name</td>
</tr>
<tr>
<td>\administration</td>
<td>Correspondence, budgets, IRB approval, etc.</td>
</tr>
<tr>
<td>\documentation</td>
<td>Research logs, codebooks, etc.</td>
</tr>
<tr>
<td>\work</td>
<td>Datasets, do-files an log-files</td>
</tr>
<tr>
<td>\analysis</td>
<td>Do-files and logs for statistical analysis</td>
</tr>
<tr>
<td>\data_cleaning</td>
<td>Do-files and logs for data management</td>
</tr>
<tr>
<td>\datasets</td>
<td>Datasets</td>
</tr>
<tr>
<td>\derived</td>
<td>Datasets constructed from the source data</td>
</tr>
<tr>
<td>\source</td>
<td>Original, unchanged data</td>
</tr>
<tr>
<td>\figures</td>
<td>Figure and chart outputs</td>
</tr>
<tr>
<td>\manuscripts</td>
<td>Drafts of papers</td>
</tr>
</tbody>
</table>
Do-Files

• Simple text files to contain your commands that should be:
  – Legible
    • Must contain the documentation and be formatted so that it is easy to follow what is being done
  – Robust
    • Must produce the same results when run on a different machine or at a different time
Making Do-Files Legible

Use a header at the beginning of do-files:

*****************************************************************************
* PURPOSE:  * Do-file for CRMDA advanced Stata Seminar
* 
* AUTHOR: 
* Jacob Fowles
* 
* DATE MODIFIED: 
* 4/4/14
* 
* Notes: 
* 1) The "estout" package must be installed
*****************************************************************************

There are 78 *s in the header, which mirrors the width of a standard piece of paper using Courier New, 10 point font
"Clearing and Setting the Table"
Using Do-Files

- You can should a standard set of commands at the beginning of do-files to set up your "workspace" in Stata. Here’s mine:

```stata
***INITIAL SETUP***
clear all
macro drop _all
eststo clear
set linesize 80
capture log close
set more off
cd "C:\users\jacob\documents\dropbox\project_1"
log using "work\data_cleaning\log1", replace
```
Setting the Working Directory

Establish your working directory using the \texttt{cd} command. For example:
\begin{verbatim}
. cd "c:\mydocs\research\project_1"
\end{verbatim}

Now, all file and directory references you make are relative to the directory you set using \texttt{cd}:
\begin{verbatim}
. save "datasets\derived\full_4-5-14.dta"
\end{verbatim}

This saves your dataset in:
\begin{verbatim}
c:\mydocs\research\project_1\datasets\derived
\end{verbatim}
Creating the Log-File

To close any open log-file, start your do-file with:
```
capture log close
```

Create a log-file using:
```
.log using "act_exp.log", replace text
```

This creates a plain text copy of everything that appears in your output window. It is saved in folder established using the `cd` command

- This file will contain all commands that were run and the output generated by those commands
- The file will be called `act_exp.log`
- The `replace` option overwrites a previous log-file with the same name
- The text can be copied into a do-file in case of emergencies (user-written ado-file `log2do2` can do this for you)
Documenting Within Do-Files (Do-Files as Research Logs)

• If a line starts with an *, everything on that line is treated as a comment
  – * can also be used to temporarily stop a command from being executed
  – A line of ********* can be used to separate different parts of your program.

• You can add comments to a line of code after a //:
  . reg price mpg displacement // model with controls only

• Everything between /* and */ is treated as a comment. This is useful when typing a long description or explanation.
Making Do-Files Legible

- Alignment and indentation can also be used to help legibility.
  - For example
    - rename make model
    - rename headroom space
    - rename rep78 repair
      or
    - rename make model
    - rename headroom space
    - rename rep78 repair

    . reg dv var1 var2 var3 var4 ///
    var5 var6 var7 var8 var9 ///
    var10 var11
      or
    . reg dv var1 var2 var3 var4 ///
      var5 var6 var7 var8 var9 ///
      var10 var11
Datasets: To Save, or Not to Save?

• In general, I *almost never* save "analysis" datasets. Instead, my do-file re-creates the "analysis" dataset from scratch each time. Why?
  – No chance that I will overwrite my raw data by accident
  – No problem with version control (different results using different "analysis" datasets)
  – Saves valuable space on Dropbox and my backups
Sharing Datasets with Colleagues

Sharing data with colleagues may require saving “analysis” datasets. Conclude the “cleaning and recoding” part of your do-file with the following command:

```
datasignature set
```

A checksum is added to the meta-data of your dataset. Then later if you type:

```
datasignature
```

the dataset’s current checksum is displayed. If the dataset is different in any way since you originally set the datasignature, these checksums will not match.

If checksums do not match, you can use:

```
datasignature confirm
```

to get a list of data elements that have changed since you set the datasignature.

This can save you a lot of headaches when a colleague accidentally changes the dataset and subsequently produces estimates that you cannot duplicate.
Other Helpful Hints

• An asterisk on the Stata do-file editor window means your do-file has changed since it was last saved:
Efficient Stata Programming
Two Stata Programming Commandments

1. You should *never* manually copy/paste results from the output window to somewhere else, manually key in estimated coefficients or statistics, or manually add significance stars.

2. You should *never* write duplicate lines of code to perform the same procedure to multiple variables, multiple datasets, etc.

*Ok, almost never.*
Local and Global Macros*

• A macro in Stata refers to an item stored in memory that takes on a determined (numeric or string) value

• Macros can be:
  – single items or lists of items
  – local (exist only within a do-file) or global (persist in memory for the duration of the interactive session)
  – referenced by name, which returns the stored item(s)
  – modified by and included in other appropriate mathematical or string operations

*Much of the text and examples that follow are drawn from:
Global Macros

• Don't use them (unless you have an overriding reason to need to do so)
Local Macros

Why use them? Generally, to save effort and reduce opportunities for typos

If I type in Stata:

. local controls "trunk weight length"

I have assigned the name controls to the string of characters appearing within the quotes.

We refer to macros using single quotes (‘ ’):

. regress mpg ‘controls’
Local Macros

• What can go wrong?
  – Macro names, like everything in Stata, are case sensitive. ‘Controls’ and ‘controls’ are two different macros.
  – You may mistype the name of the macro. An undefined macro that refers to nothing is substituted by nothing (the empty string """)
  – You reference them after they cease to exist. Local macros exist only within the Stata program (do-file, interactive session) in which they are created
Macros Can Contain Numbers, Too

Defining numeric macros doesn’t require quotes:

`. local i 1`

We can redefine the macro by incrementing it by one:

`. local i = ‘i’ + 1`
Using Macros with `foreach`

Suppose I want to generate a series of powers of a variable y:
```
. generate y_2 = y^2
. generate y_3 = y^3
. generate y_4 = y^4
```

We can use a local macro combined with a loop:
```
. foreach i in 2 3 4 {
    .    gen y`i' = y`i'
    . }
```
foreach using in

. foreach macro_name in list_of_values {
.  one or more command defined using that macro name
.
}  

This requires you to:

– Create a named macro
– Directly input a list comprised of *individual* elements
– Define each command to be executed by the loop (as many as necessary)
foreach using of

. foreach macro_name of list_type
list_of_values {
  .   one or more command defined using that
       macro name
. }

This requires you to:
  – Create a named macro
  – Directly input a list of the type that is named
  – Define each command to be executed by the loop
    (as many as necessary)
foreach using of

varlist example:
. foreach j of varlist * {
.   capture gen log`j' = log(`j')
.   capture gen rec`j' = 1/`j'
. }

numlist example:
. foreach k of numlist 2/4 {
.   gen y_`k' = y`k'
. }

local example:
. local controls "price mpg weight"
. foreach j of local controls {
.     summarize `j', detail
.     egen z_`j' = std(`j')
.     label var z_`j' "Z-scored `j'"
.     summarize z_`j'
. }

foreach using of
Imagine you want to run separate regressions on subsamples of your data, distinguished by the values taken by a categorical variable:

```
.sysuse auto, clear
(1978 Automobile Data)
.regress price mpg rep78 displacement if foreign==0
.regress price mpg rep78 displacement if foreign==1
```
forvalues

Or you could use a forvalues loop to accomplish the same task:

```
. forvalues i = 0/1 {
    regress price mpg rep78 displacement if foreign==`i'
}
```

Syntax:

- 0/1 is shorthand for "evaluate all integers between 0 and 1"
- 20(10)500 is shorthand for "evaluate integers from 20 to 500, in steps of 10"
In-Class Exercise

Create a do-file that:

1. Opens the auto dataset *(sysuse auto, clear)*
2. Uses a *foreach* loop to create logged versions of mpg, weight, and displacement
3. Creates two local macros that contain the variables from Step 2. One should only include the logged versions (name it *control_1*) and the other the non-logged versions (name it *control_2*)
4. Uses a *forvalues* loop to:
   1. Regress *price* on *foreign*, controlling for the variables in *control_1*
   2. Regress *price* on *foreign*, controlling for the variables in *control_2*
Simplifying Your Life with Loops: An Advanced Example

• The following website contains county-level population growth data, disaggregated by state: http://bit.ly/popdata

• It would be nice to have these data aggregated into a single Stata dataset. So I can:
  – Right-click and save each file to a working directory
  – Import each CSV into Stata
  – Save each worksheet
  – Append the individual sheets together

• Where's my graduate research assistant?
Simplifying Your Life with Loops: An Advanced Example

• Remember **Stata Commandment #2**?

• Do some sleuthing:
  – Each link opens a consistently-formatted CSV file. Stata can import those using `insheet`.
  – Each CSV file is named according to a simple convention:
    • CO-EST2011-POPCHG2010_2011-‘x’.csv
    • Where ‘x’ is the state FIPS code. These are numeric, but are not sequentially 1-50 (so a `forvalues x = 1/50` loop will stop with an error). But all is not lost.
Constructing loops with \texttt{levelsof}

\texttt{levelsof} displays (and optionally saves as a macro) a sorted list of the distinct values taken by a variable, which you can then use to construct loops.

Create (find) a dataset of all FIPS codes and open it:
\begin{verbatim}
. use statefips.dta
\end{verbatim}

Use \texttt{levelsof} to create a macro that you can reference in a loop:
\begin{verbatim}
. levelsof fips, local(fips)
1 2 4 5 6 8 9 10 12 13 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 49 50 51 53 54 55 56
\end{verbatim}
Now we are getting somewhere...

Stata's `copy` command can download files of any type from the internet.* Putting it all together:

```
  . foreach x of local fips {
    .    insheet using data/csv/'x'.csv, clear
    .    save data/dta/'x', replace
  }
```

* If you download files that are archived, `unzipfile` can automate that process as well.
One more quick loop... 

. clear
. foreach x of local fips {
.     append using data/dta/`x'
. }
<output omitted>
. describe
Contains data
    obs:        3,192
    vars:       21
    size: 325,584

**Bottom line: Your graduate research assistant gets a well-deserved evening off.**
Accessing Stored Information after Running a Stata Command
Returned Results

• In addition to the output displayed in the output window, Stata saves a lot of information about the command and its results in memory.

• These returned results are useful when you want to use information produced by a Stata command to do something else in Stata
An example

```
. sysuse auto.dta, clear
. quietly sum mpg
. return list

scalars:

    r(N) =  74
    r(sum_w) =  74
    r(mean) =  21.2972972972973
    r(Var) =  33.47204738985561
    r(sd) =  5.785503209735141
    r(min) =  12
    r(max) =  41
    r(sum) =  1576

. local mpg_mean r(mean)
. di `mpg_mean'
21.297397
```
Using These Results

• Stata distinguishes two types of returned results:
  – E-class (accessed by `ereturn`) commands perform estimation (regression, factor analysis, ANOVA, etc.)
  – R-class (accessed by `return`) commands are pretty much everything else (summarize, tabulate, describe, etc.)
Using Stored Estimation Information

Run a regression:
. sysuse auto.dta, clear
. quietly regress price mpg if foreign==1
. ereturn list
<output omitted>

Do some useful things using stored information:
1. Mark the observations used in the regression:
   . gen sample_dum = e(sample)
2. Calculate the error variance:
   . display e(rss)/(e(N)-1)
Using Stored Estimation Information, Continued

You can also directly access the coefficients and standard errors estimated by the regression:

```stata
. qui reg price mpg trunk foreign
```

Save the coefficient of the constant term in a macro:

```stata
. local b_cons _b[_cons]
```

Manually calculate a $t$-statistic for `mpg`:

```stata
. di _b[mpg]/_se[mpg]
```

Create a fitted value for the price of a high mpg, foreign-made car with a small trunk:

```stata
. di _b[_const] + _b[mpg]*35 + _b[trunk]*5 + _b[foreign]*1
```

3169.15
Exporting Stata Output Tables
Saving and Displaying Estimation Results with `esttab`*

The `estimates` suite of commands and the `estout` ado-package facilitates the creation of advanced output tables.

```
. sysuse auto, clear
. gen price_1k = price/1000
. label var price_1k "Price (000)"
. quietly regress price_1k mpg
. estimates store model_1
. generate mpg_sq = mpg*mpg
. label var mpg_sq "Square of mpg"
. quietly regress price_1k mpg mpg_sq
. estimates store model_2
```

*`esttab` is part of the "estout" ado package (ssc install estout)
```

```

```

```
```
. esttab model_1 model_2, stat(N F r2) star(.1 .05 01) b(3) se(2)

<table>
<thead>
<tr>
<th></th>
<th>(1)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>price_1k</td>
<td>price_1k</td>
</tr>
<tr>
<td>mpg</td>
<td>-0.239***</td>
<td>-1.265***</td>
</tr>
<tr>
<td></td>
<td>(0.05)</td>
<td>(0.29)</td>
</tr>
<tr>
<td>mpg_sq</td>
<td></td>
<td>0.021***</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0.01)</td>
</tr>
<tr>
<td>_cons</td>
<td>11.253***</td>
<td>22.716***</td>
</tr>
<tr>
<td></td>
<td>(1.17)</td>
<td>(3.37)</td>
</tr>
</tbody>
</table>

N       74.000     74.000
F       20.258     18.277
r2      0.220      0.340

Standard errors in parentheses
* p<.1, ** p<.05, *** p<.01
```
Table 2: Regression Results

<table>
<thead>
<tr>
<th></th>
<th>(1)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mileage (mpg)</td>
<td>-0.239*** (-4.50)</td>
<td>-1.265*** (-4.37)</td>
</tr>
<tr>
<td>Square of mpg</td>
<td></td>
<td>0.021*** (3.60)</td>
</tr>
<tr>
<td>Constant</td>
<td>11.253*** (9.61)</td>
<td>22.716*** (6.75)</td>
</tr>
<tr>
<td>N</td>
<td>74.000</td>
<td>74.000</td>
</tr>
<tr>
<td>F</td>
<td>20.258</td>
<td>18.277</td>
</tr>
<tr>
<td>r2</td>
<td>0.220</td>
<td>0.340</td>
</tr>
</tbody>
</table>

t statistics in parentheses
* p<.1, ** p<.05, *** p<.01
esttab

. esttab model_1 model_2 using filename, stat(N F r2) star(* .1 ** .05 *** .01) b(3) t(2) label nomtitles title("Table 2:  Regression Results") rtf

<table>
<thead>
<tr>
<th></th>
<th>(1)</th>
<th>(2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mileage (mpg)</td>
<td>-0.239***</td>
<td>-1.265***</td>
</tr>
<tr>
<td></td>
<td>(-4.50)</td>
<td>(-4.37)</td>
</tr>
<tr>
<td>Square of mpg</td>
<td></td>
<td>0.021***</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3.60)</td>
</tr>
<tr>
<td>Constant</td>
<td>11.253***</td>
<td>22.716***</td>
</tr>
<tr>
<td></td>
<td>(9.61)</td>
<td>(6.75)</td>
</tr>
<tr>
<td>N</td>
<td>74.000</td>
<td>74.000</td>
</tr>
<tr>
<td>F</td>
<td>20.258</td>
<td>18.277</td>
</tr>
<tr>
<td>r2</td>
<td>0.220</td>
<td>0.340</td>
</tr>
</tbody>
</table>

\textit{t} statistics in parentheses
\(^* p < .1\), \(^** p < .05\), \(^*** p < .01\)
Summary Statistics using `estpost` and `esttab`

```stata
. qui reg price mpg weight displacement if foreign==1
. estpost summarize price mpg weight displacement if e(sample)
<output suppressed>
. esttab, cells("mean sd min max") nomtitle nonumber
title("Table 1: Summary Statistics")
Table 1: Summary Statistics

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>price</td>
<td>6384.682</td>
<td>2621.915</td>
<td>3748</td>
<td>12990</td>
</tr>
<tr>
<td>mpg</td>
<td>24.77273</td>
<td>6.611187</td>
<td>14</td>
<td>41</td>
</tr>
<tr>
<td>weight</td>
<td>2315.909</td>
<td>433.0035</td>
<td>1760</td>
<td>3420</td>
</tr>
<tr>
<td>disp.</td>
<td>111.2273</td>
<td>24.88054</td>
<td>79</td>
<td>163</td>
</tr>
</tbody>
</table>

| N     | 22      |
```

More `estout` examples can be found at [http://repec.org/bocode/e/estout/index.html](http://repec.org/bocode/e/estout/index.html)
In-Class Exercise

• Return to the previous exercise. Add code that:
  1. Creates a table of descriptive statistics in MS Word format displaying means, standard deviations, minimum and maximum values for the variables (including the dependent variable) from your regression.
  2. Creates a single output table in MS Word format that contains both sets of regression results. It should:
     • Use variable labels, not names
     • Report standard errors, not t-statistics
     • Include a table title and appropriate titles for both models (you can make them up)
And a Random Bonus Trick

- Stata’s graphing capabilities are robust but often frustrating and counter-intuitive
- Once you create a graph, you often have to resort to the graph editor to "fix" it
- You can use the graph editor recorder to log your manual changes in order to repeat them later
- Then you can include `play(filename)` at the end of the command that generates your graph
In Summary and in Conclusion

• There is a fundamental, enduring tension in programming: The Lazy Way vs. The Elegant Way
• Over the long run, the two paths converge
• Murphy’s Law of Quantitative Research: reviewers will typically only ask for changes that your do-files cannot easily accommodate
• Long’s Rule of Stata Programming: it is easier to do it right today than go back and fix it tomorrow