Using Stata to Create Graphics: Programming for Exploratory Data Visualization and Reporting

Strategic Data Project Convening 2015
Boston, MA

Billy Buchanan

Mississippi Department of Education

May 20, 2015

Purpose

Why write programs in the first place?

- Efficiency . . . Let's look at what the MDE did previously for accountability purposes.

Area	Scripts	Data Files
Growth	32	80
Graduation Rates	23	75
AYP	81	119
ESEA (Federal Law)	16	22

- And Now . . . 24 .ado files that consume 5 input files and/or ODBC connections to produce results for $>$ 800 schools.
- Rather than copying, pasting, and trying to remember what changes to hard code into scripts, we used `mde_fay, tsd(01sep2013) ted(28apr2014) fbed(01dec2013) sbsd(01feb2014)` last year for Full Academic Year determinations and can use `mde_fay, tsd(01sep2014) ted(28apr2015 ) fbed(01dec2013) sbsd(01feb2014)` this year for the same task.

Purpose

- Well written and conceived programs have some characteristics that are helpful for data analysis, changing business requirements, and reproducibility.
  - Extensibility (e.g., you can extend the functionality of the code/program without going back to the drawing board)
  - Consistent Performance (e.g., as much as we may love and use copy/paste patterns, they leave us much more exposed to errors)
  - Consistent Interface (e.g., a new feature shouldn't change the API/syntax used to call the program)
- The hope today is that you'll get some experience/exposure to some Stata programming concepts that you can use to improve your own, and your team's, productivity.

Other info

- The majority of the programs we’ll be working through today are still very early in their development, but if you’re familiar with the College Going Toolkit it should be relatively easy to follow along.
- You can install `brewscheme` by typing:
  `net inst brewscheme, from("http://www.paces-consulting.org/stata")`
- The program above isn’t required, but can make your life significantly easier when creating data visualizations in Stata.
- `brewscheme` is also used in several of the examples of `cgbubble` and `cgwaterfall` to give you an idea of how it can help you to simplify your workflow here and in general.
- To install `cgbubble` and `cgwaterfall` type:
  `net inst cgwaterfall, from("http://www.paces-consulting.org/stata")`
- The ‘ character (left-single quote in Stata speak) is to the left of the #1 key
- The ’ character (right-single quote in Stata speak) is to the left of the carriage return key
- If you hear reference to ‘"compound double quotes"’, use the quotes around the phrase in this bullet
- If you hear a reference to "quotes", use the quotes around the phrase in this bullet
- To reference ‘local macros’ you need a left-single quote the name of the macro and a right-single quote
- To reference $global macros you would use the $ before the name of the macro
- If you need to embed quotation marks or spaces in a value, use ‘"compound double quotes"’ to be safe
- If you installed the programs above before this morning, use :
  `net inst cgwaterfall, from("http://www.paces-consulting.org/stata") replace`
- The installer used above will also place a copy of the slides on your system. I tested it on a window’s machine this morning and it should appear in ‘"‘c(sysdir_plus)’/s/sdpworkshop.pdf"’

The Basics

Programming Paradigms in Stata

- Stata has two distinct programming languages.
  - Stata - what you're probably familiar with already to some degree
  - Mata - Stata's compiled language for matrix algebra/lower level programming
- Both languages support object oriented programming.
- That brings us to the different file extensions and how they are used by Stata.
  - .do files - scripts with commands and comments that do a very specific task
  - .ado files - executable scripts with commands and comments that can do a single or several tasks

The Basics
Scoping/Language

- Unlike R/Python/Julia, the scoping rules in Stata are fairly loose.
- To create the equivalent of a local variable/scalar, you need to be explicit in your use of `tempvars` and `local` macros.
- People are regularly admonished for using global macro's in the Stata community because they are not immutable (e.g., if a global with the same name gets created by an unrelated program it would change things permanently).
- Although it may take a more time and keystrokes, `tempvars` also have a nice advantage of automatic garbage collection (e.g., when the program finishes any/all of the memory needed for that temporary variable is freed up).

Following Along/More Information

- In the next set of slides you'll see references to Stata help files (e.g., `help syntax`) as well as subtitles that reference the example program `cgbubble`.
- If you want to follow along with the program references you can run

  ```
  doedit `"`c(sysdir_plus)'/c/cgbubble.ado"'.
  ```

  - If you get an error message, try

    ```
    doedit `"`c(sysdir_plus)'c/cgbubble.ado"'
    ```

    instead.
- If you wanted to see the source code from the SDP College Going Toolkit, you can use

  ```
  doedit `"`c(sysdir_plus)'/c/cgbubble1.do"'
  ```

  to view that in your do file editor.

Defining the program (help program)

cap prog drop cgbubble

- Although not strictly required, common practice is to start your program with the line:
  `cap prog drop my_program_name`
- This clears the program from the cache if it is there so any changes can be interpreted.
- Then the formal declaration of the program can happen
  `prog def my_program_name`
- If the program has an properties (e.g., can work with survey weighted data, return values, return estimates, etc. . .) you would include that here as well.

Program properties

`prog def cgbubble, rclass`

- If you've ever tried to do something like
  `bys this_group:  xtile test_scores = qtilescore`
  and received an error message, it is because the `xtile` program does not have the byable property included in its definition.
- The only programming property I want to talk about today is the `rclass` property.
  - When you use a command like `summarize` it leaves values behind in a combination of macros, scalars, and/or matrices.
  - If you have scripts/programs that you need to run sequentially (e.g., *Clean, Connect, Analyze*, . . .), you can use the return values to dynamically generate the code you need in the next step.
- For our purposes today, the utility of the rclass property is how we can use it to return the results from our data manipulations without ever making any permanent changes to the source data.
- You can use
  `viewsource cgbubble.ado`
  or
  `viewsource cgwaterfall.ado`
  to view the sourcecode in a help window.

Program properties (continued)

`help return`

- If you have data that were manipulated and wanted to make sure you could retrieve it after the program finishes executing, you can use
  `mkmat varlist, mat(matrixname)`
  to define a matrix from your data set then use
  `ret mat thedata = matrixname`
  and you can retrieve it in `r(thedata)` after the program executes.
- Other times, you may want to use subroutines (e.g., a program defined within a program) that calculate a single value and you can use
  `ret sca scalar_name = thevalue`
- The last bit of usefulness, is
  `ret loc macroname = macrocontents`
- One of the reasons why using global macros is generally discouraged is the ability to return local macros from programs/functions and in doing so, preventing overwriting the global namespace (e.g., if you create a global macro and my program creates a global with the same name, the results now depend on the order in which the programs are run).

Versioning (`help version`)

`version 13.1`

- Stata put significant effort into backwards compatibility since there are a significant number of contributions available from the user community.
- One way they do this is through their interpreter, which is set by a program using the line `version ##`.
- If you wanted to see it in action enter
  `doedit `"`c(sysdir_plus)'c/cgbubble.ado"'`
  and change the line from `version 13.1` to `version 14.0` if you don't have Stata 14 yet you won't be able to run the program until you undo the change.

Defining the API (help syntax)

```
syntax [using/] [if] , COHortid(string asis) ...[STRetch scheme(passthru) ...]
```

- This is both the place where you get to determine how other users and you will interact with your program.
- This is also where you can add support/options for conditional statements, weights, etc…
- After either a square bracket and comma, or a comma by itself, you would specify the optional options (those inside of the square brackets) and required options (outside of the square brackets).

Handling Conditional Statements (help marksample)

[if]...marksample touse

Overview
Intro to Stata Programming
Structure of Stata Programs
API Design
Making .do Files
Flexible .ado Files
General Tips/advice
Invaluable tools
Deciding on Parameters
Worked Example
Advanced Examples
Handling Multiple Use Cases With a Single Program
Automating Reports using LaTeX2ε and Stata
Additional References
Stata Specific
Data Visualization
Questions/Contact

- Although there are other low-level methods for handling cases when you want to allow if and/or in arguments to your program,
  `marksample touse`
  is what you'd be most likely to see looking at other peoples' code.
- marksample takes the arguments provided to the if/in qualifier and creates a temporary variable that identifies the observations that satisfy your condition; touse just happens to be the macro that everyone uses (probably because of the nice semantics of the command and macroname).
- Inside of your program, you'd want to use if `touse' to reference only the records that satisfy the condition.

Required Arguments

```
COHortid(string asis) ...QTILEmath(string asis) SCHname(string asis)
DIPLoma(string asis) ENRollyear1(string asis)
```

- You're probably wondering about the mixed case above; when you're entering the named arguments for your program, capital letters are used to define the fewest number of characters the user would need to type to correctly pass their argument to the parameter (e.g., `qtile(qrt_8_math)` and `qtilemath(qrt_8_math)` are the same for the person using the program, but ‘qtile’ and ‘qtilemath’ are not the same within the program).
- If you're wondering about the argument type `string asis`, it is to make sure there aren't any issues if someone wants to call the command on a dataset that isn't currently in memory (since the variable wouldn't exist, it would issue an error, so this type of argument gives you a bit of flexibility in that sense).

Required Arguments

Years(numlist min=2 max=2 sort)

- A numlist is kind of like a variable list, but accepts numeric values instead.
- Setting min=2 max=2 prevented me from having to use a starting and ending year value since any number of values more or less than two would raise a syntax error from Stata.
- The sort option makes it even easier since it will take the values you enter and sort them from lowest to highest (and if you know the first value is going to be less than the second you can use the extended macro functions (see help extended_fcn) to access those values directly .
- In this case you could use
  ```
  `:  word 1 of `years”
  ```
  to get the starting year and
  ```
  `:  word 2 of `years”
  ```
  to get the end year.

Optional Arguments

`STRetch scheme(passthru) SAVing(string asis) JITter(passthru)`

- The subtitle above shows a few different types of optional arguments:
  - `STRetch` is the equivalent of a boolean in Stata. If the user enters `str, stre, stret, stretc,` or `stretch` as an optional argument the value of the local macro ‘stretch’ would be "stretch".
  - `scheme(passthru)` is exactly the same optional argument that exists in the various graph commands in Stata. If you’re familiar with R, an optional argument of type passthru is the equivalent of quoting an expression to prevent it from being evaluated immediately.
  - `JITter(passthru)` is another example like scheme above, but limited to certain types of graphs.
- There are several other optional argument types available that each have their own tradeoffs.
- Hopefully, this gives you some ideas for how different optional parameters can be used/modified.

Things to Avoid . . .

or not

- The string asis parameter type is a double edged sword of sorts.
- You can use it to pass variable names to your program to avoid errors that would be caused if the data were not loaded and the parameter is a variable name/list type. In these cases, **do not** wrap the macro in quotes.
- If you’re using this type of argument for file names/paths, wrap the macro in compound double quotes (e.g., ‘"‘macroname’"’); This will make sure embedded spaces in strings don’t break your program.
- Be `assert`ive. In otherwords, if the use case you’re designing the program for would not allow string variables, you could add tests that make sure string variables are not being used :

  ```
  if substr("`:  type `theVariable”", 1, 3) == "str" {
  err 198
  }
  ```

(continued)

- Create an undocumented optional argument to your programs called `DBug`.
  - Then use
    ```
    if "`dbug'" != "" pause on
    ```
  - Add
    ```
    pause "A reminder message to yourself"
    ```
    before/after any/every line of code that you would want to interrupt for inspection.
  - Before closing the program add
    ```
    pause off
    ```
    to turn the feature off and prevent it from being triggered by other programs.

OR

- Learn to love a combination of
  ```
  set tracedepth #
  ```
  and
  ```
  set trace on
  ```
  to find and/or diagnose errors in your code.

VCS

- Version Control Systems can quite literally save the life of a project you are working on when/if a change goes awry and breaks your code base.
- There are two general families of VCS and currently three massive VCS projects in widespread use:
  - Distributed Systems (e.g., everyone gets their own copy of the entire codebase and then requests for their changes to be incorporated into the master) - Git and Mercurial are the two major platforms for distributed VCS.
  - Centralized Systems (e.g., one item can be checked out from the server by one person at any one time); If you've ever been hit with the cannot edit this file because it is already open by another use message on sharepoint, you've experienced something similar to a centralized VCS. The centralized VCS currently in widest use is Subversion.
- If your agency has QA/QC Engineers/Developers, consulting with them is the best way to go when selecting a VCS.

Editors

- If you currently work in multiple languages, and/or believe you will in the near future, a good text editor will be an invaluable tool to prevent you from making some of the simpler errors.
- One way it can do this is by supporting multiple language specific tools (e.g., syntax highlighting, code completion, linters/formatting checkers, etc . . .).
- Depending on your needs and/or comfort level learning new tools there is quite a lot to choose from. Emacs and vi(m) are probably the two oldest, and potentially most extensible, text editors and both have varying degrees of support for integration with Stata and/or to support syntax highlighting.
- Other editors, like Sublime Text, Notepad++, etc . . . can sometimes be supported by Stata, the Stata user community, and/or the developers of the editors themselves (see Chuck Huebler's Blog for additional details).

File names/paths

- Many people regularly use file names in their day-to-day work that are less than friendly to programming languages (e.g., using spaces, unicode characters, etc. . .).
- Only you . . . can prevent bad file names from breaking your software.
- `confirm new file` will have a return code (`_rc == 0`) if the file (or directory) **does not** currently exist.
- `confirm file` will have a return code (`_rc == 0`) if the file (or directory) **does** currently exist.
- You can also use some extended macro functions to check for the existence of files/directories (e.g.,
  ```
  ‘:  dir ‘"‘c(pwd)’"’ files diff_bubble.txt’
  ```
  would return nothing if not found or the name of the file/directory you were looking for if found).

Variables

- We all have our own styles and preferences for naming conventions.
- I, for example, find underscores painful due to their location on the keyboard, but can appreciate the clarity that it brings to situations.
- People familiar with other languages like Java, R, Python, Julia, etc. . . may use styles that are typical in those languages (e.g., like lowerCamelCase, UpperCamelCase, underscores_to_the_rescue_for_everything, or chained.methods.that.use.dots).
- You can use extended macro functions to automate well defined cleaning tasks based on the input type (e.g., run part of a cleaning script conditional on the variable being a string, else use a different process, . . .).

Methods/Estimators

- You've probably come across a graph at some point that left you wondering why the author took one approach vs another?
- If you have, you can parameterize the methods for creating the graphs (we'll see some of that not too much longer from now).
- Sometimes there are several different methods and you may want to check several of them out (e.g., using a linear smoother, loess smoother, quadratic smoother, or fractional polynomial smoother over a scatterplot).

Options to functions/programs called from your program

- You can easily add support for any existing options to underlying functions/programs using the passthru type options.
- While this gives the end user additional flexibility, it restricts the amount you are able to control and test.
- If you're programming something primarily for internal use, you shouldn't need to allow too many different options since you'd likely want some form of standardized layout/template.

cgbubble.ado

- I created a diff file between the .ado file I created and the .do file that was included in the toolkit.
- To find where it is located on your system enter: `findfile c/diff_bubble.txt, all`
- You can open the file in a text editor, or the do-file editor, to see the two files side by side.

cgwaterfall.ado

- Sometimes you want to have a single interface to handle several different types of use cases.
- This is what I was trying to accomplish with the cgwaterfall program.
- Three of the Four waterfall charts in the Toolkit use the same aggregations, layout, etc. . .
- The last waterfall chart uses different aggregation/graphing.
- If you want to view the code use

  ```
  viewsource cgwaterfall.ado
  ```

  or to see it the do file editor use :

  ```
  doedit ‘"‘c(sysdir_plus)’/c/cgwaterfall.ado"’
  ```

- You can also see the help file to get a better idea of the different inputs and to see some examples (*Note: if the example uses the using argument in the program it is assuming that you are in the root directory of the SDP College Going Toolkit*).

Additional Requirements

LaTeX2ε

- Similar to R, Perl, and other languages, LaTeX also has an online repository system complete with binaries for multiple operating systems called CTAN.
- In addition to the LaTeX installer, you also need to make sure that the program **pdflatex** is included with the binary (since it will be used to compile the LaTeX source code into a pdf).
- It may be possible to do something similar with Pandoc, but I've not experimented with it yet.

The file & filefilter commands

Building LaTeX2ε Source Dynamically

```
// Create a LaTeX source document
qui: file open texreport using `"`c(pwd)'/stataout.tex"', w replace

// Add in typical header material/packages that are generally useful
file write texreport `"\documentclass[12pt,oneside,final,letterpaper]{article}"' _n
file write texreport `"\usepackage{pdflscape}\usepackage{tabulary}"' _n
file write texreport `"\usepackage{graphicx}\usepackage{longtable}"' _n
file write texreport `"\usepackage[hidelinks]{hyperref}"' _n

// Tell LaTeX what type of graph files to look for and where to find them
file write texreport `"\DeclareGraphicsExtensions{.pdf, .png}"' _n
file write texreport `"\graphicspath{{"`c(pwd)'"}}"' _n

// Add some metadata to build a title page
file write texreport `"\title{Stata Output Example: \\ `c(current_date)'}"' _n
file write texreport `"\author{`c(username)'}"' _n

/* Start the document, create a title page, add a table of contents, add a
list with links to each of the figures, and the same for each of the tables */
file write texreport `"\begin{document}"' _n
file write texreport `"\begin{titlepage} \maketitle \end{titlepage}"' _n
file write texreport `"\newpage\clearpage \tableofcontents \newpage\clearpage"' _n
file write texreport `"\listoffigures \newpage\clearpage"' _n
file write texreport `"\listoftables \newpage\clearpage"' _n

/* Now you can insert graphs into the file using commands like:
// Export the graph to a pdf or png
gr export `"`districtID'.pdf"', as(pdf) replace

// Add an include statement to the LaTeX file
file write texreport `"\includegraphics[width=\textwidth]{`districtID'.pdf}"' _n

When completed, make sure to close the file connection:
file close texreport
*/
```

Making Stata Write Compile Scripts for you

```
// Check the operating system to know what type of script to create
if regexm(`"`c(os)'"', "[xX]") == 1 {

    // Create a shell script file called makeReport with write permissions
    file open script using makeReport.sh, w replace

    // Add a Bash header and move to the next line of the file
    file write script "#!/bin/bash" _n

    // Create a local macro with the execute string
    loc exec ! chmod +x makeReport.sh && ./makeReport.sh

} // End IF Block for *nix based systems

// For windows
else {

    // Create a batch file with the same settings as above
    file open script using makeReport.bat, w replace

    // Add a Windows Batch header to the file
    file write script ":: Compile LaTeX source from Stata" _n

    // Create a local macro with the execute string
    loc exec ! makeReport.bat

} // End ELSE Block for Windows systems

// Usually helpful to make three passes to make sure everything is rendered properly
file write script `"pdflatex "`c(pwd)'/`saving'.tex""' _n
file write script `"pdflatex "`c(pwd)'/`saving'.tex""' _n
file write script `"pdflatex "`c(pwd)'/`saving'.tex""' _n
file close script

// Then execute
`exec'
```

Stata Specific

- Baum, C. F. (2009). *An Introduction to Stata Programming*. College Station, TX: StataPress
    - This book inconjunction with the NC-151 and NC-152 Stata NetCourses are what I used to get started with Stata programming
    - The book is an excellent resource written by a fairly prolific Stata programmer (if you've ever thought about doing Instrumental Variables Regression in Stata, you've come across his work for sure)
- Cox, N. J. (2014). *Speaking Stata graphics: A collection from the Stata Journal*. College Station, TX: StataPress
    - Nick is the editor of the Stata Journal for a very good reason; This book contains a little over 100 reasons why
- Mitchell, M. N. (2012). *Interpreting and Visualizing Regression Models Using Stata*. College Station, TX: StataPress
    - If you do any work with logit/probit or other types of non-continuous outcomes, this book will be invaluable for you
    - It will show you how to make the `help margins` command provide you with the estimated marginal effects with lots of flexibility to estimate different types of marginal effects adjusted for different covariates in your model and/or at different levels of the covariates
    - Then it shows you how to make extremely useful graphs to help you interpret those graphs (e.g., you could graph the marginal effect of 8th grade math scores adjusted for FRL/Sex on On-Time Graduation to get a better understanding about the point where the probability of graduating on time is $>$ chance)
- Mitchell, M. N. (2012). *A Visual Guide to Stata Graphics*. 3rd Ed. College Station, TX: StataPress
    - You can almost think of this book like your McDonalds menu of Stata graphs; Each of the graphs shows how individual options affect the rendering of the graph and is an invaluable resource when you're stumped on how to create a specific type of graph

Data Visualization

- Cleveland, W. (1993). *Visualizing Data*. Summit, NJ: Hobart Press
- Cleveland, W. (1994). *The Elements of Graphing Data*. Summit, NJ: Hobart Press
  - Bill Cleveland worked at Bell Labs (the same Bell Labs where Tukey worked) and did tons of research on the visual perception of data visualizations as well as developing the framework of "lattice" type graphs
  - The books are great references for understanding what makes graphs easier for your audience to percieve (e.g., perception of area/volume = bad, perception of location/distance between locations = good)
- Tufte, E. R. (2001). *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press
  - If you're wondering why the background is all white and there aren't grid lines when using the scheme files created with the `help brewscheme` program, Tufte is the reason why
  - He is a huge advocate of removing "chart-junk" from graphs and creating clean looking graphs
- Tukey, J. W. (1977). *Exploratory Data Analysis*. Boston, MA: Addison-Wesley Publishing
  - John Tukey = The Man.
  - W/O Tukey Exploratory Data Analysis might not exist, neither would things like the Box-Plot, the terms Bit/Byte, the Stem and Leaf Plot, the five-number summary, and quite a bit of statistics developed in the first half of the 20th century
- Wilkinson, L. (2005). *The Grammar of Graphics*. New York City, NY: Springer
  - SPSS's current data visualization framework is based on Wilkinson's work, and the `ggplot2` package in R is an implementation of his framework in R
  - If you really want to dig into the minute details of data visualization, this tome would be a great place to start (as a heads up it is > 600 pages long and is fairly dense, but extremely helpful and informative when thinking about statistical graphics)

Questions & Contact Info

If you have any questions, comments, and/or wanted to follow up with me about things, feel free to contact me at:

william@williambuchanan.net

I'm still working on the `help brewscheme` program, but already have some things in the works to add an additional 900+ named colors to Stata's color definitions based on the XKCD survey a few years ago and am incorporating color translations that will allow you to view your graph based on colors transposed to show how people with any of the three major types of color blindness would see your graph (e.g., so you can proof things before going to press and make sure it is clear to any/all audiences).