Writing `Stata` packages

Gonzalo Vazquez-Bare
gvazquez@econ.ucsb.edu

UC Santa Barbara

April 22, 2022

Why write Stata packages?

- Make your empirical analysis more efficient
  - Data handling (e.g. preparing your data set for estimation)
  - Estimation, inference
  - Control the output of your code
- Sharing with coauthors, other researchers
- Making your methods more widely available
  - And get more citations in the process!
- It only takes a few step to go from a do-file to a command / package

Writing `Stata` packages

- Writing `Stata` packages is relatively straightforward:
  1. Write the code
  2. Save it as an .ado file in the appropriate folder
  3. Write the help file
  4. Add auxiliary files (functions, data sets), if any
- This talk goes over a few useful commands and examples on how to write a `Stata` program

Some useful resources

- User manuals

  `https://www.stata.com/features/documentation/`

- Statalist (`Stata` forum)

  `https://www.statalist.org/`

- Help files:

```
help cmd
```

Very brief introduction to `Stata`

- `Stata` is a statistical analysis software
- Most operations in `Stata` involve *variables*
- A `Stata` data set looks like this:

	var 1	var 2	⋯	var k
obs 1				
obs 2				
⋮				
obs n				

- We look at data sets by columns (vertically)

Very brief introduction to Stata

- Typical syntax of a Stata command:

```
cmd [varlist] [if] [in] [, options]
```

- Some examples:

```
describe

summarize x

regress y x if z==1, vce(robust)
```

- Command (and variable) names can be abbreviated:

```
d

sum x

reg y x if z==1, vce(robust)
```

Very brief introduction to `Stata`

- Running a `Stata` command produces output and stored results
- Two main classes of commands:
  - r-class: general commands (`describe, summarize, count`)
  - e-class: estimation commands (`regress, logit, gmm`)
- After running an r-class command, see stored results by typing:

  ```
  return list
  ```

- After running an e-class command, see stored results by typing:

  ```
  ereturn list
  ```

Very brief introduction to `Stata`

- Script files in `Stata` are called do-files
- Do-files can be saved in .do format
- Programs written in do-files are deleted when the session ends
- To write a command, use ado-files

Macros in `Stata`

- Scalars and strings are handled with *macros*
- There are three types of macros in `Stata`:
  - Locals
  - Globals
  - Tempvars, tempnames, tempfiles

Macros: locals

- Locals can contain numbers or strings

```
local a "x y z"

local b = 1
```

- The contents of a local are accessed using single quotes (‘ ’)

```
display "‘a’"

display ‘b’

local c = ‘b’ + 1

local d "‘a’ w"
```

Macros: locals

- Locals exist within the program or do-file in which they are defined
- A local defined in a do-file does not exist in the interactive session
    - And vice versa

Macros: globals

- Globals work like locals, but they are available anywhere in `Stata`
  - E.g. a global defined in a do-file can be accessed interactively
- The contents of a global are accessed using a dollar sign ($)

```
global a = 4

display $a
```

Macros: tempvars, etc

- Tempvars define temporary variables

```
tempvar auxvar
```

```
generate ‘auxvar’ = x^2
```

- Tempvars are deleted as soon as the program stops running
- Tempname does the same for scalars and matrices
- Tempfile does the same for files

Scalars

- Scalars are variables that contain single numbers or strings

```
scalar k = 6
display k + 4
```

- A scalar can have the same name as a variable
    - `Stata` gives priority to variables

```
generate k = runiform()
display k
```

- To avoid confusion, use the `scalar()` pseudofunction

```
display k
display scalar(k)
```

Writing programs in Stata

- A program in Stata typically looks like this:

```
program define myprog [, class]
      syntax ...
      code
end
```

Some useful programming commands

- `capture`
    - Runs a command omitting the output and error messages
    - Avoids terminating execution after a nonzero error code

```
capture drop x
gen x = runiform()
```

Some useful programming commands

- `marksample`
  - Generates a temporary binary variable indicating the observations to be used in subsequent code
  - Useful when the command allows for `if` and `in` options

```
marksample touse

reg y x z if ‘touse’
```

Some useful programming commands

- `quietly`
  - Runs a command omitting the output
  - Useful for controlling the output of your program

```
quietly regress y x z

quietly {
        summarize y x z
        regress y x z
}
```

Some useful programming commands

- `tokenize`
  - Parses a string into tokens
  - Useful to split a varlist into multiple variable names

```
local numbers "one two three"
tokenize ‘numbers’
display "‘1’"
display "‘2’"
display "‘3’"
```

Mata

- Mata is Stata's matrix programming language
- Mata's syntax is more similar to other languages like R
- It handles vectors and matrices in a more “standard” way
- Using Mata interactively:

```
mata
x = 1
x + 2
M = (1,2,3 \ 4,5,6)
M
end
```

Using Mata in ado-files

- Add `mata:` in front of each command line

```
gen x = runiform()
mata: st_view(x=.,.,"x")
mata: mean(x)
sum x
```

- Wrap Mata code in `mata {...}`

```
gen x = runiform()
mata {
   st_view(x=.,.,"x")
   mean(x)
}
sum x
```

Using Mata in ado-files

- Write a separate Mata function and call it from within the ado-file

```
gen x = runiform()
mata: mymatafun(args)
sum x

********************

capture mata: drop mymatafun()
mata:
   (function code)
end
```

Debugging

- `set trace on|off`
  - Traces the execution of a program
- Tracing an error “by hand”: add

  ```
  display "line ..."
  ```

  at different lines of the code and see where it stops

Help files

- All commands need an associated help file describing the syntax, options, default parameters, etc
- Help files are written using `Stata`'s output language: `smcl`
  - "Stata Markup and Control Language"
- Can be written in the do-file editor, then saved as .sthlp
- For a help file template, type:

```
help examplehelpfile

viewsource examplehelpfile.sthlp
```

Sharing your package online

- Upload all the ado-files, .sthlp files, .mo files, data sets and auxiliary files to your website or repository
- Add two more files:
  - `stata.toc`: content file
  - *pkgname*`.pkg`: package description

Sharing your package online

- Minimal content of the `stata.toc` file:

```
v 3
p pkgname
```

- Can also include descriptions, links, etc

- See `Stata` manual for further details:

  `https://www.stata.com/manuals/rnet.pdf`

Sharing your package online

- Example of *pkgname*`.pkg` file:

```
v 3
d description line 1
d description line 2...
d Distribution-Date: yyyymmdd
f myprog.ado
f myprog.sthlp
f myfun.mo
f mydata.dta
```

- See `Stata` manual for further details:

`https://www.stata.com/manuals/rnet.pdf`

Sharing your package online

- Then the package can be installed using the `net install` command

```
net install pkgname, from(url) replace
```

Thank you!