Using Python in a Stata estimation command

David M. Drukker
Executive Director of Econometrics

Cass University
4 September 2019

Contents

Part I
Why use Python in Stata?

1 Why use Python in Stata?

Why use Python in Stata?

- Stata has many commands doing data science
- Many data science and numerical methods have been implemented in Python but not (yet) in Stata
- We want to use methods coded in Python in Stata

What kind of method?

- Some parts of data science are methods for data management, graphical analysis, statistical estimation, and prediction
- In this talk, I focus on statistical estimation

Do today

- Write a Stata command that estimates the mean and stores an estimate of its VCE
- Rewrite this command using Python to do the numerical computations

Stata's estimation/post estimation framework

- After a Stata estimation command you can
  - Use `test` or `testnl` to do a Wald test of a hypothesis
  - Use `predict` to predict observation-level functions of the estimated parameters and the data
  - Use `margins` to estimate average-partial effects
  - Use `estat` post-estimation tests and inference
  - Other cool things

A mean example

```
. sysuse auto
(1978 Automobile Data)
. mean mpg rep78
Mean estimation                   Number of obs    =         69
```

	Mean	Std. Err.	[95% Conf.	Interval]
mpg	21.28986	.7062326	19.88059	22.69912
rep78	3.405797	.1191738	3.167989	3.643605

```
. test (mpg=20) (rep78=3)
 ( 1)  mpg = 20
 ( 2)  rep78 = 3
       F(  2,    68) =    5.92
            Prob > F =    0.0043
```

A mean example

```
. ereturn list
scalars:
               e(df_r) =  68
             e(N_over) =  1
                 e(N) =  69
               e(k_eq) =  1
               e(rank) =  2
macros:
            e(cmdline) : "mean mpg rep78"
                e(cmd) : "mean"
                e(vce) : "analytic"
              e(title) : "Mean estimation"
          e(estat_cmd) : "estat_vce_only"
            e(varlist) : "mpg rep78"
       e(marginsnotok) : "_ALL"
         e(properties) : "b V"
matrices:
```

```
                e(b) :  1 x 2
                e(V) :  2 x 2
              e(_N) :  1 x 2
           e(error) :  1 x 2
functions:
           e(sample)
```

A mean example

Wald test statistic of the q-dimensional hypothesis that $\widehat{\boldsymbol{\beta}} = \boldsymbol{\beta}_0$ is

$$w = (\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}_0)(\mathbf{V})^{-1}(\widehat{\boldsymbol{\beta}} - \boldsymbol{\beta}_0)$$

where

- $\mathbf{V}$ is the VCE
- The F-statistic version is $f = (1/q)w$

```
. matrix bhat = e(b)
. matrix b0   = (20, 3)
. matrix vhat = e(V)
. matrix c    = bhat-b0
. matrix f    = (1/2)*c*invsym(vhat)*c´
. scalar f    = f[1,1]
. scalar d1   = 2
. scalar d2   = e(N)-1
. display "F is " scalar(f)
F is 5.9215615
. display "p is " Ftail(scalar(d1), scalar(d2), scalar(f))
p is .00425821
```

Bring in the Python

```
. python:
------------------------------------------------- python (type end to exit) -----
>>> from sfi import Matrix
>>> import numpy as np
>>> b  = Matrix.get(´e(b)´)
>>> V  = Matrix.get(´e(V)´)
>>> print(b)
[[21.28985507246377, 3.4057971014492754]]
>>> print(V)
[[0.498764471131868, 0.033862757453327896], [0.033862757453327896, 0.0142024043
> 39177386]]
>>> end
--------------------------------------------------------------------------------
```

- Using Python interactively
- sfi (Stata Function Interface) module has classes that make Stata and Python talk to each other
  See https://www.stata.com/python/api16/Data.html
- importing Matrix class to get started

Bring in the Python

```
. python:
------------------------------------------------------- python (type end to exit) -----
>>> print(b)
[[21.28985507246377, 3.4057971014492754]]
>>> print(V)
[[0.498764471131868, 0.033862757453327896], [0.033862757453327896, 0.0142024043
> 39177386]]
>>> end
-------------------------------------------------------------------------------
```

- The Python session is persistent
- Persistence is good for simple interactive examples
- Persistence is not good when writing commands for users
  - The __main__ module belongs to users
  - Programmers should never destroy anything or leave behind anything in __main__

Create numpy arrays

- Work in do files
- Python is also persistent (__main__) between do files

```
python:
from sfi import Matrix
import numpy as np
b  = Matrix.get('e(b)')
V  = Matrix.get('e(V)')
b  = np.array(b,dtype='float64')
V  = np.array(V,dtype='float64')
print(b)
print(V)
end
```

numpy arrays

```
. do np1.do
. python:
------------------------------------------------------- python (type end to exit) -----
>>> from sfi import Matrix
>>> import numpy as np
>>> b  = Matrix.get(´e(b)´)
>>> V  = Matrix.get(´e(V)´)
>>> b  = np.array(b,dtype=´float64´)
>>> V  = np.array(V,dtype=´float64´)
>>> print(b)
[[21.28985507  3.4057971 ]]
>>> print(V)
[[0.49876447 0.03386276]
 [0.03386276 0.0142024 ]]
>>> end
-------------------------------------------------------------------------------

.
end of do-file
```

Calculate the Wald statistic

```
python:
from sfi import Matrix, Scalar
import numpy as np
from scipy.stats import f
bh  = Matrix.get('e(b)')
Vh  = Matrix.get('e(V)')
bh  = np.array(b,dtype='float64')
Vh  = np.array(V,dtype='float64')
b0  = np.array([20, 3])
c   = bh - b0
Vi  = np.linalg.inv(Vh)
```

```
p1  = np.matmul(Vi,np.transpose(c))
fv  = (1/2)*np.matmul(c,p1)
d1  = 2
d2  = Scalar.getValue('e(N)') - 1
p   = f.sf(fv, d1, d2)
print(fv)
print(p)
end
```

Calculate the Wald statistic

```
. do np2.do
. python:
```

```
------------------------------------------------------------------------------ python (type end to exit) ------
>>> from sfi import Matrix, Scalar
>>> import numpy as np
>>> from scipy.stats import f
>>> bh  = Matrix.get(´e(b)´)
>>> Vh  = Matrix.get(´e(V)´)
>>> bh  = np.array(b,dtype=´float64´)
>>> Vh  = np.array(V,dtype=´float64´)
>>> b0  = np.array([20, 3])
>>> c   = bh - b0
>>> Vi  = np.linalg.inv(Vh)
>>> p1  = np.matmul(Vi,np.transpose(c))
>>> fv  = (1/2)*np.matmul(c,p1)
>>> d1  = 2
>>> d2  = Scalar.getValue(´e(N)´) - 1
>>> p   = f.sf(fv, d1, d2)
>>> print(fv)
[[5.92156154]]
>>> print(p)
[[0.00425821]]
>>> end
------------------------------------------------------------------------------------------------------------------
```

```
.
end of do-file
```

Part II
Back to Stata

2 Back to Stata

How to store stuff in Stata

- Scope
  - local: within a .do or .ado file
  - global: anywhere in a current session
- Store a dataset in variables
  - variables names and contents are global
- Store a matrix in a matrix
  - `. matrix b = (1, 2, 3)`
  - matrix names and contents are global
- Store a scalar in a scalar
  - `. scalar a = invnorm(.975)`

  - scalar names and contents are global

- Store lists, string scalars and numeric scalars in macros

Macros are a way of storing and retrieving values

- Scope
  - local: within a .do or .ado file
  - global: anywhere in a current session
- Store lists, string scalars and numeric scalars in macros
- See my blog post
  Programming an estimation command in Stata: Where to store your stuff
  `https://t.co/TIJqkSScvc`
  for another introduction to macros

Storing stuff in Macros

Macros store information as strings

- `local` macros are local
- `global` macros are global
- There are three syntaxes for storing information in macros

`local` *lclname* `=` *exp*
`global` *gblname* `=` *exp*

`local` *lclname* `"`*string*`"`
`global` *gblname* `"`*string*`"`

`local` *lclname* `:` *extended_fcn*
`global` *gblname* `:` *extended_fcn*

Retrieving stuff stored in macros

- Everywhere a punctuated macro name appears, its contents are substituted for the macro name.
  - The names of local macros are punctuated by enclosing them between single left quotes (`) and single right quotes (')
  - The names of global macros are punctuated by preceding them with a dollar sign ($).

Examples of local macros

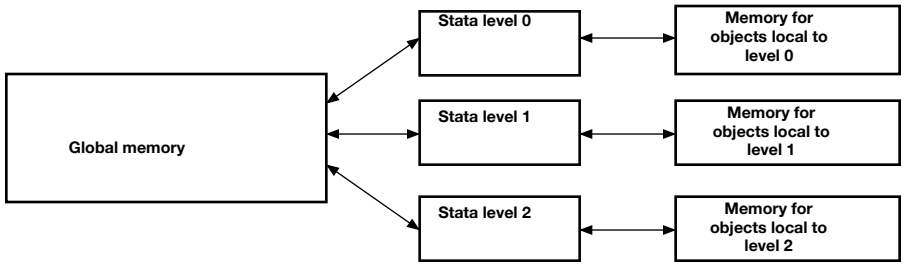
```
. local value = invnorm(.975)
. display "punctuating value yields `value´"
punctuating value yields 1.959963984540054
. local vlist "y x1 x2"
. display "punctuating vlist yields `vlist´"
punctuating vlist yields y x1 x2
. local cnt : word count y x1 x2
. display "punctuating cnt yields `cnt´"
punctuating cnt yields 3
```

Examples of global macros

```
. global value = invnorm(.975)
. display "punctuating value yields $value"
punctuating value yields 1.959963984540054
. global vlist "y x1 x2"
. display "punctuating vlist yields $vlist"
punctuating vlist yields y x1 x2
. global cnt : word count y x1 x2
. display "punctuating cnt yields $cnt"
punctuating cnt yields 3
```

Levels of Stata

- The notion that there are levels of Stata can help explain the difference between global boxes and local boxes



Global macros are accessible across do-files

- In the main do-file we define a global macro and then execute another do-file to do the work
- The work do-file can access the information stored in the global macro by the main do-file

```
*-------------------------------Begin globala.do ---------------
*! globala.do
*  In this do-file we define the global macro vlist, but we
*  do not use it
global vlist y x1 x2

do globalb
*-------------------------------End globala.do ---------------
```

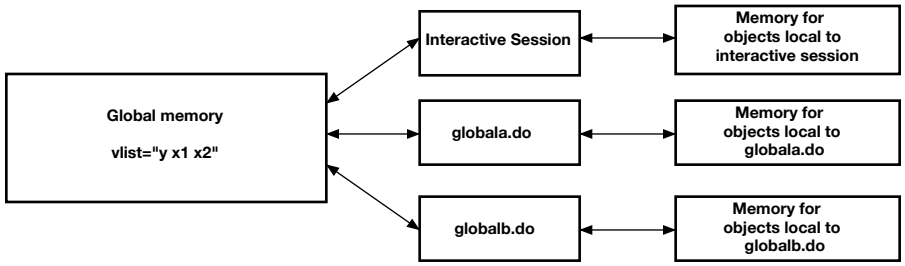
```
*-------------------------------Begin globalb.do ---------------
*! globalb.do
*  In this do-file, we use the global macro vlist, defined in globala.do

display "The global macro vlist contains |$vlist|"
*-------------------------------End globalb.do ---------------
```

Globals are global example

```
. do globala
. *-------------------------------Begin globala.do ---------------
. *! globala.do
. *  In this do-file we define the global macro vlist, but we
. *  do not use it
. global vlist y x1 x2
.
. do globalb
. *-------------------------------Begin globalb.do ---------------
. *! globalb.do
. *  In this do-file, we use the global macro vlist, defined in globala.do
.
. display "The global macro vlist contains |$vlist|"
The global macro vlist contains |y x1 x2|
. *-------------------------------End globalb.do ---------------
.
end of do-file
. *-------------------------------End globala.do ---------------
.
end of do-file
```

A global macro in global memory

- Global macros live in global memory



Local macros are local

- Each do-file has a separate name space local macros
- Each do-file can have a local macro calls, say `mylist`, and they not interfere with each other

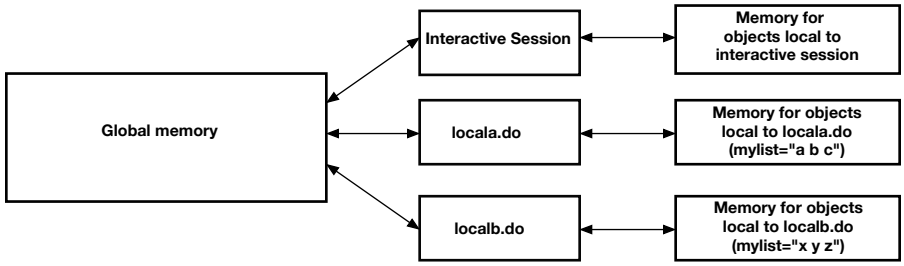
```
*-------------------------------Begin locala.do ---------------
*! locala.do
local mylist "a b c"
display "mylist contains |`mylist'|"

do localb

display "mylist contains |`mylist'|"
*-------------------------------End locala.do ---------------


*-------------------------------Begin localb.do ---------------
*! localb.do
local mylist "x y z"
display "mylist contains |`mylist'|"
*-------------------------------End localb.do ---------------
```

Local macros are local

```
. do locala
. *------------------------------Begin locala.do ---------------
. *! locala.do
. local mylist "a b c"
. display "mylist contains |`mylist´|"
mylist contains |a b c|
.
. do localb
. *------------------------------Begin localb.do ---------------
. *! localb.do
. local mylist "x y z"
. display "mylist contains |`mylist´|"
mylist contains |x y z|
. *------------------------------End localb.do -----------------
.
end of do-file
.
. display "mylist contains |`mylist´|"
mylist contains |a b c|
. *------------------------------End locala.do -----------------
.
.
end of do-file
```

A local macro in level-specific memory

- Local macros live in level-specific memory



More macro tricks

- For more about local versus global macros, see my blog post

  Programming an estimation command in Stata: Global macros versus local macros

  `http://bit.ly/1ksYHrI`

- Macro evaluation is recursive

```
. local macro1 "hello"
. display "`macro1´"
hello
. local i 1
. display "`macro`i´´"
hello
```

`forvalues`

`forvalues` *lname* = #/# {

`commands referring to` ‘*lname*’

}

```
// forvalues.do
forvalues i = 1/3 {
      display "i is now `i'"
}
```

```
. do forvalues
. // forvalues.do
. forvalues i = 1/3 {
  2.       display "i is now `i´"
  3. }
i is now 1
i is now 2
i is now 3
.
end of do-file
```

foreach

foreach *lname* in *list* {
    commands referring to 'lname'
}

foreach *lname* of local *lmacname* {
    commands referring to 'lname'
}

foreach II

```
// foreach.do
local vlist y x1 x2
foreach v of local vlist {
      display "v is now `v'"
}
```

```
. do foreach
. // foreach.do
. local vlist y x1 x2
. foreach v of local vlist {
  2.       display "v is now `v´"
  3. }
v is now y
v is now x1
v is now x2
.
end of do-file
```

foreach III

```
// foreach2.do
local v "3"
display "v is now `v'"
local vlist y x1 x2
foreach v of local vlist {
      display "v is now `v'"
}
display "v is now |`v'|"
```

```
. do foreach2
. // foreach2.do
. local v "3"
. display "v is now `v´"
v is now 3
```

```
. local vlist y x1 x2
. foreach v of local vlist {
  2.       display "v is now `v´"
  3. }
v is now y
v is now x1
v is now x2
. display "v is now |`v´|"
v is now ||
.
end of do-file
```

if

- The are two type of `if` in Stata

  1. *command* `if` *exp*

     restricts the sample to those observations for which `if` *exp* is true and *command* works on the restricted sample

     ```
     . poisson accidents traffic tickets if male==1
     ```

  2. In do files and ado files,

     `if` *exp* { *commands* }

     will only execute *commands* `if` *exp* is true

if II

```
. local test 0
. if `test´ < 1 {
.         display "expression is true!"
expression is true!
. }
```

Now that we can program ...

- Write a do file to estimate the mean of variable
- Write an ado command that estiamtes the mean of a variable

summarize leaves results in r()

```
// version 1.0.0 09Jun2019 (This comment is ignored by Stata)
version 15              // version #.# fixes the version of Stata
sysuse auto
summarize price
return list             // summarize stores its results in r()
```

summarize leaves results in r()

```
. do meanb
. // version 1.0.0 09Jun2019 (This comment is ignored by Stata)
. version 15              // version #.# fixes the version of Stata
. sysuse auto
(1978 Automobile Data)
. summarize price
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906
. return list             // summarize stores its results in r()
scalars:
                  r(N) =  74
```

```
               r(sum_w) =  74
                r(mean) =  6165.256756756757
                 r(Var) =  8699525.974268788
                  r(sd) =  2949.495884768919
                 r(min) =  3291
                 r(max) =  15906
                 r(sum) =  456229

.
end of do-file
```

Using summarize to compute estimates

- We can use `summarize` to compute the sample-average estimator for the mean and its standard error

```
// version 1.0.0 09Jun2019
version 15
sysuse auto
quietly summarize price
return list
local sum          = r(sum)
local N            = r(N)
local mu           = (1/`N')*`sum'
generate double e2 = (price - `mu')^2
quietly summarize e2
local V            = (1/((`N')*(`N'-1)))*r(sum)
display "muhat   = " `mu'
display "sqrt(V) = " sqrt(`V')
mean price
```

```
. do meanc
. // version 1.0.0 09Jun2019
. version 15
. sysuse auto
(1978 Automobile Data)
. quietly summarize price
. return list
scalars:
                  r(N) =  74
               r(sum_w) =  74
                r(mean) =  6165.256756756757
                 r(Var) =  8699525.974268788
                  r(sd) =  2949.495884768919
                 r(min) =  3291
                 r(max) =  15906
                 r(sum) =  456229
. local sum          = r(sum)
. local N            = r(N)
. local mu           = (1/`N´)*`sum´
. generate double e2 = (price - `mu´)^2
. quietly summarize e2
. local V            = (1/((`N´)*(`N´-1)))*r(sum)
. display "muhat   = " `mu´
muhat   = 6165.2568
. display "sqrt(V) = " sqrt(`V´)
sqrt(V) = 342.87193
. mean price
Mean estimation                   Number of obs   =         74

------------------------------------------------------------------------------
             |       Mean   Std. Err.      [95% Conf. Interval]
-------------+----------------------------------------------------------------
       price |   6165.257   342.8719      5481.914      6848.6
------------------------------------------------------------------------------

.
end of do-file
```

3 Programming an ado file

Syntax of Stata estimation commands

`cmdname` *depvar varlist* [*weight*] [*if*] [*in*] [`,` *options*]

- Standard options
  - noconstant
  - vce(robust)
  - vce(cluster *clustervar*)
  - level(#)
- Maximize options
  - iterate(#)
  - from(*init_spec*)
  - nrtolerance(#)
  - constraints(*numlist*)

Examples of Stata estimation commands

- `regress lnwage educ momed daded neighqual, vce(robust)`
- `xtreg lnwage educ momed daded neighqual, vce(cluster id)`
- `var dlinvestment dlincome dlconsumpton, constraints(1 2) iterate(30)`

Make your command work like other commands

- Return results in `e()`
- Display results in a standard output table
- `test` works automatically
- Make `predict` work with command
- `margins` uses your `predict`
- Document your command
  - help file
  - Stata Journal article

Defining a new command

Putting the following code into a file called mymean.ado

```
program define mymean

end
```

defines the program `mymean`.

An ado that always computes the same thing

File `mymean2/mymean.ado`

```
*! version 2.0.0  09Jun2019
program define mymean
        version 15

        quietly summarize price
        local sum          = r(sum)
        local N            = r(N)
        local mu           = (1/`N')*`sum'
        generate double e2 = (price - `mu')^2
        quietly summarize e2
        local V            = (1/((`N')*(`N'-1)))*r(sum)
        display "muhat   = " `mu'
        display "sqrt(V) = " sqrt(`V')
end
```

```
. sysuse auto, clear
(1978 Automobile Data)
. quietly cd mymean2
. capture program drop mymean
. mymean
muhat   = 6165.2568
sqrt(V) = 342.87193
. quietly cd ..
```

Parsing Stata syntax

- Use the `syntax` command in your ado program to parse what it was passed
- In your ado program, `syntax` will parse the assorted pieces of Stata syntax passed to your command and store these items in local macros for you to manipulate.
- Example:
  - `mymean` needs to take a varlist

Add `syntax` statement to `mymean`

File `mymean3/mymean.ado`

```
*! version 3.0.0  09Jun2019
program define mymean
        version 15

        syntax varlist
        display "varlist contains `varlist'"
        quietly summarize `varlist'
        local sum          = r(sum)
        local N            = r(N)
        local mu           = (1/`N')*`sum'
        capture drop e2
        generate double e2 = (`varlist' - `mu')^2
        quietly summarize e2
        local V            = (1/((`N'-1)*(`N')))*r(sum)
        display "muhat   = " `mu'
        display "sqrt(V) = " sqrt(`V')
end
```

```
. quietly cd mymean3
. program drop mymean
. mymean price
varlist contains price
muhat   = 6165.2568
sqrt(V) = 342.87193
. quietly cd ..
```

- `syntax` can parse any standard Stata syntax

Put results into matrices b and V

File `mymean5/mymean.ado`

```
*! version 5.0.0  09Jun2019
program define mymean
        version 15

        syntax varlist

        quietly summarize `varlist'
        local sum          = r(sum)
        local N            = r(N)
        matrix b           = (1/`N')*`sum'
        matrix colnames b  = mu
        capture drop e2
        generate double e2 = (`varlist' - b[1,1])^2
        quietly summarize e2
        matrix  V          = (1/((`N')*(`N'-1)))*r(sum)
        matrix colnames V  = mu
        matrix rownames V  = mu

        matrix list b
        matrix list V
end
```

`mymean` **now produces**

```
. quietly cd mymean5
. program drop mymean
. mymean price
symmetric b[1,1]
            mu
r1  6165.2568
symmetric V[1,1]
            mu
mu  117561.16
. quietly cd ..
```

What type of varlist?

- `syntax` allows you to specify extentions or restrictions on the type of varlist allowed
  - You can extend the default to allow for time-series or factor-variable operators
  - You can restrict the number or type of variables allowed
- In the case hand, we want to restrict the variables to be numeric and we want only one variable specified

Restrict varlist

File `mymean5a/mymean.ado`

```
*! version 5.1.0  09Jun2019
program define mymean
        version 15

        syntax varlist(max=1 numeric)

        quietly summarize `varlist'
        local sum          = r(sum)
        local N            = r(N)
        matrix b           = (1/`N')*`sum'
        matrix colnames b  = mu
        capture drop e2
        generate double e2 = (`varlist' - b[1,1])^2
        quietly summarize e2
        matrix  V          = (1/((`N')*(`N'-1)))*r(sum)
        matrix colnames V  = mu
        matrix rownames V  = mu

        matrix list b
        matrix list V
end
```

`mymean` **now produces**

```
. quietly cd mymean5a
. program drop mymean
. capture noisily mymean price mpg
too many variables specified
. capture noisily mymean make
string variables not allowed in varlist;
make is a string variable
. mymean price
symmetric b[1,1]
            mu
r1  6165.2568
symmetric V[1,1]
            mu
mu  117561.16
. quietly cd ..
```

Tempnames

- Recall that variable, matrix and scalar names are global in Stata
- This implies that there are problems with our current version of `mymean`
  - `mymean` will overwrite any Stata matrices named `b` or `V`
  - `mymean` will drop any variable named `e2` in the dataset
  - The locals `sum` and `N` are fine; they are local
- The solution is to use temporary names stored in local macros
- The Stata command `tempname` creates a list of local macros, each of which contains a name that is not used elsewhere
- The Stata command `tempvar` creates a list of local macros, each of which contains a name that is not used elsewhere

File `myean6/myregress.ado`

```
*! version 6.0.0  09Jun2019
program define mymean
        version 15

        syntax varlist(max=1 numeric)

        tempname b V
        tempvar  e2

        quietly summarize `varlist'
        local sum            = r(sum)
        local N              = r(N)
        matrix `b'           = (1/`N')*`sum'
        matrix colnames `b'  = mu
        generate double `e2' = (`varlist' - `b'[1,1])^2
        quietly summarize `e2'
        matrix  `V'          = (1/((`N')*(`N'-1)))*r(sum)
        matrix colnames `V'  = mu
        matrix rownames `V'  = mu

        matrix list `b'
        matrix list `V'
end
```

Example of `mymean` with tempnames

```
. quietly cd mymean6
. program drop mymean
. mymean price
symmetric __000000[1,1]
            mu
r1  6165.2568
symmetric __000001[1,1]
            mu
mu  117561.16
. quietly cd ..
```

- Safe program
- We cannot access our results
- Need to store the results somewhere

4 Making mymean an estimation command

Command classes in Stata

- All Stata commands are either e-class, r-class, s-class or n-class.
  - e-class commands return results in e()
  - r-class commands return results in r()
  - s-class commands return results in s()
  - n-class commands do not return results.
- By convention, Stata estimation commands are e-class commands

e-class commands

- e-class commands return
  - e(b), the vector of parameter estimates
  - e(V), the VCE of e(b)
  - e(sample), a function that equals 1 if the observation is part of the estimation sample and 0 otherwise.
  - e(N), the number of observations in the sample

File `mymean7/mymean.ado`

```
*! version 7.0.0  09Jun2019
program define mymean, eclass
        version 15

        syntax varlist(max=1 numeric)

        tempname b V
        tempvar  e2

        quietly summarize `varlist'
        local sum            = r(sum)
        local N              = r(N)
        matrix `b'           = (1/`N')*`sum'
        matrix colnames `b'  = mu
        generate double `e2' = (`varlist' - `b'[1,1])^2
        quietly summarize `e2'
        matrix  `V'          = (1/((`N')*(`N'-1)))*r(sum)
        matrix colnames `V'  = mu
        matrix rownames `V'  = mu

        ereturn post `b' `V'
        ereturn scalar   N   = `N'
        ereturn display
end
```

eclass version of mymean

```
. quietly cd mymean7
. program drop mymean
. mymean price
```

	Coef.	Std. Err.	z	P>|z|	[95% Conf.	Interval]
mu	6165.257	342.8719	17.98	0.000	5493.24	6837.273

```
. ereturn list
scalars:
                  e(N) =  74
macros:
         e(properties) : "b V"
matrices:
                  e(b) :  1 x 1
                  e(V) :  1 x 1
. quietly cd ..
```

`if` and `in` sample restrictions

- use `syntax` to parse the input to the command
- use `marksample` to create a temporary variable that identifies the sample

```
*! version 8.0.0  09Jun2019
program define mymean, eclass
        version 15

        syntax varlist(max=1 numeric) [if] [in]
        marksample touse

        tempname b V
        tempvar  e2
        quietly summarize `varlist' if `touse'==1
        local sum            = r(sum)
        local N              = r(N)
        matrix `b'           = (1/`N')*`sum'
        matrix colnames `b'  = mu
        generate double `e2' = (`varlist' - `b'[1,1])^2 if `touse'==1
        quietly summarize `e2' if `touse'==1
        matrix  `V'          = (1/((`N')*(`N'-1)))*r(sum)
        matrix colnames `V'  = mu
        matrix rownames `V'  = mu

        ereturn post `b' `V', esample(`touse')
        ereturn scalar   N   = `N'
        ereturn scalar df_r  = `N'-1
        ereturn display
end
```

mymean with if restriction

```
. quietly cd mymean8
. program drop mymean
. mymean price if mpg>20
(38 missing values generated)
```

	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
mu	5350.306	393.102	13.61	0.000	4552.266	6148.345

```
. ereturn list
scalars:
                  e(N) =  36
                e(df_r) =  35
macros:
         e(properties) : "b V"
matrices:
                  e(b) :  1 x 1
                  e(V) :  1 x 1
functions:
              e(sample)
. mean price if mpg>20
Mean estimation                   Number of obs    =         36
```

	Mean	Std. Err.	[95% Conf.	Interval]
price	5350.306	393.102	4552.266	6148.345

```
. quietly cd ..
```

Part III
Python Stata programming

5 Python Stata programming

```
*! version 1.0.0
program define pmean, eclass
        version 16.0
        tempname b
        matrix `b' = (1, 2, 3)
        python: MyMeanWork("`b'")
        matrix list `b'
end

version 16.0
python:
from sfi import Matrix
import numpy as np

def MyMeanWork(bname ):
        b     = Matrix.get(bname)
        b     = np.array(b,dtype='float64')
        b     = b*b
        Matrix.store(bname, b)
end
```

Call a Python function from ado

```
. program drop _all
. cd pmean1
/Users/dmd/Dropbox/projects/talks/2019/uk19/cass/pythonp/tex/examples/pmean1
. pmean
__000000[1,3]
    c1  c2  c3
r1   1   4   9
. cd ..
/Users/dmd/Dropbox/projects/talks/2019/uk19/cass/pythonp/tex/examples
```

```
*! version 2.0.0
// compute mean using python
program define pmean, eclass
        version 16.0

        syntax varlist(numeric) [if] [in]
        marksample touse

        tempname b v N
        python: MyMeanWork("`varlist'", "`touse'", "`b'", "`v'", "`N'")
        ereturn post `b' `v', esample(`touse')
        ereturn scalar   N   = scalar(`N')
        ereturn scalar df_r  = scalar(`N')-1
        ereturn display
end

version 16.0

python:

from sfi import Data, Matrix, Missing, SFIToolkit, Scalar
import numpy as np

def MyMeanWork(varlist, touse, bname, vname, nname):
        data  = Data.get(var=varlist, selectvar=touse)
```

```
import numpy as np

def MyMeanWork(varlist, touse, bname, vname, nname):
        data  = Data.get(var=varlist, selectvar=touse)
        data  = np.array(data, dtype='float64')
        vlist = varlist.split()
        p     = len(vlist)
        shape = data.shape
        n     = shape[0]
        cols  = shape[1]

        Matrix.create(bname, 1, p, Missing.getValue())
        Matrix.create(vname, p, p, Missing.getValue())
```

```
        m    = np.mean(data, axis=0)
        E    = data - m
        Ep   = np.transpose(E)
        E2   = np.matmul(Ep,E)
        E2   = (1/n)*(1/(n-1))*E2

        Matrix.store(bname, m)
        Matrix.setRowNames(bname,['mean'])
        Matrix.setColNames(bname,vlist)
        Matrix.store(vname, E2)
        Matrix.setRowNames(vname,vlist)
        Matrix.setColNames(vname,vlist)
        Scalar.setValue(nname,n)
end
```

```
. program drop _all
. cd pmean2
/Users/dmd/Dropbox/projects/talks/2019/uk19/cass/pythonp/tex/examples/pmean2
. sysuse auto
(1978 Automobile Data)
. pmean mpg rep78
```

	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
mpg	21.28986	.7062326	30.15	0.000	19.88059	22.69912
rep78	3.405797	.1191738	28.58	0.000	3.167989	3.643605

```
. test (mpg=20) (rep78=3)
 ( 1)  mpg = 20
 ( 2)  rep78 = 3
       F(  2,    68) =    5.92
            Prob > F =    0.0043
. cd ..
/Users/dmd/Dropbox/projects/talks/2019/uk19/cass/pythonp/tex/examples
```

```
*! version 3.0.0
// compute mean using python
program define pmean, eclass
        version 16.0

        syntax varlist(numeric) [if] [in]
        marksample touse

        qui count  if `touse'==1
        if r(N) < 1 {
                di "{err}no observations"
                exit(2000)
        }

        tempname b v N
        python: MyMeanWork("`varlist'", "`touse'", "`b'", "`v'", "`N'")
        ereturn post `b' `v', esample(`touse')
        ereturn scalar   N   = scalar(`N')
        ereturn scalar df_r  = scalar(`N')-1
        ereturn display
end

version 16.0

python:
```

```
version 16.0

python:

from sfi import Data, Matrix, Missing, SFIToolkit, Scalar
import numpy as np

def MyMeanWork(varlist, touse, bname, vname, nname):
        data  = Data.get(var=varlist, selectvar=touse)
        data  = np.array(data, dtype='float64')
        vlist = varlist.split()
        p     = len(vlist)

        if p < 1:
                SFIToolkit.errprint('Bad varlist in work function')
                SFIToolkit.exit(498)

        if data.ndim != 2:
                SFIToolkit.errprint('Bad array in work function')
                SFIToolkit.exit(498)

        shape = data.shape
        n     = shape[0]
        cols  = shape[1]
```

```
        if cols != p :
                SFIToolkit.errprint('Wrong number of columns in work function data')




        if cols != p :
                SFIToolkit.errprint('Wrong number of columns in work function data')
                SFIToolkit.exit(498)

        Matrix.create(bname, 1, p, Missing.getValue())
        Matrix.create(vname, p, p, Missing.getValue())

        m    = np.mean(data, axis=0)
        E    = data - m
        Ep   = np.transpose(E)
        E2   = np.matmul(Ep,E)
        E2   = (1/n)*(1/(n-1))*E2

        Matrix.store(bname, m)
        Matrix.setRowNames(bname,['mean'])
        Matrix.setColNames(bname,vlist)
        Matrix.store(vname, E2)
        Matrix.setRowNames(vname,vlist)
        Matrix.setColNames(vname,vlist)
        Scalar.setValue(nname,n)
end
```