Call Stata from Python

Zhao Xu

Principal Software Engineer
StataCorp LLC

Introduction

Stata 16 introduces tight integration with Python allowing users to embed and execute Python code from within Stata. In this talk, I will demonstrate new functionality we have been working on: calling Stata from within Python. Note that this functionality is not available yet and is still a work in progress.

Contents

How it works

The Python package **pystata** provides two ways to interact with Stata:

- IPython magic commands
  - **%%stata** and **%stata**
  - **%%mata** and **%mata**
  - **%pystata**
- A suite of API functions
  - The **stata** module
  - The **config** module

The magic commands can be used to access Stata and Mata interactively in an IPython kernel-based environment.

- Jupyter Notebook/console
- Jupyter Lab/console
- Other environments that support the IPython kernel, such as Spyder IDE, PyCharm IDE, etc

The API functions can be used to interact with Stata and Mata in a command-line Python environment.

- Windows Command Prompt
- Unix terminal
- Python built-in IDLE, etc

The API functions can also be used together with the magic commands in the IPython environment. Both of them can be used with **Stata's Function Interface (sfi) module** to access Stata and Mata.

Benefits

In Python, with this integration, you can now:

- Use Stata's broad suite of estimation and post-estimation commands
  - Model estimations for various disciplines
  - Statistical inferences and predictions
  - Marginal effects and interaction analysis
  - Model specification, diagnostic, and goodness-of-fit analysis
- Create hundreds of thousands of publication-quality and distinctly styled graphics
- Make your research and work reproducible all the time using Stata's integrated version control
- Write and execute Stata and Python code in one environment
- Interact with each other by passing data and results back and forth
- And more...

Configuration and initialization

To get started, we need to configure the **pystata** package within Python so that it can be found and imported by Python. Suppose we have Stata installed in **C:/Program Files/Stata/**, it then can be initialized in Python as follows:

In [1]:

```
import sys
sys.path.append("C:/Program Files/Stata/utilities")
from pystata import config
config.init()
```

```
  ___  ____  ____  ____  ____ (R)
 /__    /   ____/   /   ____/  Stata Embedded
___/   /   /___/   /   /___/          Copyright 1985-2019 StataCorp LLC
  Statistics/Data analysis            StataCorp
                                      4905 Lakeway Drive
     MP - Parallel Edition            College Station, Texas 77845 USA
                                      800-STATA-PC        https://www.stata.com
                                      979-696-4600        stata@stata.com
                                      979-696-4601 (fax)

Stata license: 10-user 4-core network perpetual
Serial number: 1
  Licensed to: Stata Developer
               StataCorp LLC

Notes:
      1. Unicode is supported; see help unicode_advice.
      2. More than 2 billion observations are allowed; see help obs_advice.
      3. Maximum number of variables is set to 5,000; see help set_maxvar.
```

Examples

Example 1: Basic usage

To illustrate the general usage of calling Stata from Python, we use the automobile data. The data has mileage rating and weight of 74 automobiles. The variables of interest in the data are **mpg**, **weight**, and **foreign**. The **foreign** variable assumes the value 1 for foreign and 0 for domestic automobiles. We wish to analysis the relationship among the mileage rating, weight, and whether the automobile is foreign or domestic.

In [2]:

```
%%stata
use https://www.stata-press.com/data/r16/auto, clear
describe
```

```
. use https://www.stata-press.com/data/r16/auto, clear
(1978 Automobile Data)

. describe

Contains data from https://www.stata-press.com/data/r16/auto.dta
  obs:            74                          1978 Automobile Data
 vars:            12                          13 Apr 2018 17:45
                                              (_dta has notes)
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
make            str18   %-18s                 Make and Model
price           int     %8.0gc                Price
mpg             int     %8.0g                 Mileage (mpg)
rep78           int     %8.0g                 Repair Record 1978
headroom        float   %6.1f                 Headroom (in.)
trunk           int     %8.0g                 Trunk space (cu. ft.)
weight          int     %8.0gc                Weight (lbs.)
length          int     %8.0g                 Length (in.)
turn            int     %8.0g                 Turn Circle (ft.)
displacement    int     %8.0g                 Displacement (cu. in.)
gear_ratio      float   %6.2f                 Gear Ratio
foreign         byte    %8.0g      origin     Car type
-------------------------------------------------------------------------------
Sorted by: foreign
```

Then we obtain summaries of **mpg** and **weight** for the foreign and domestic cars.

In [3]:

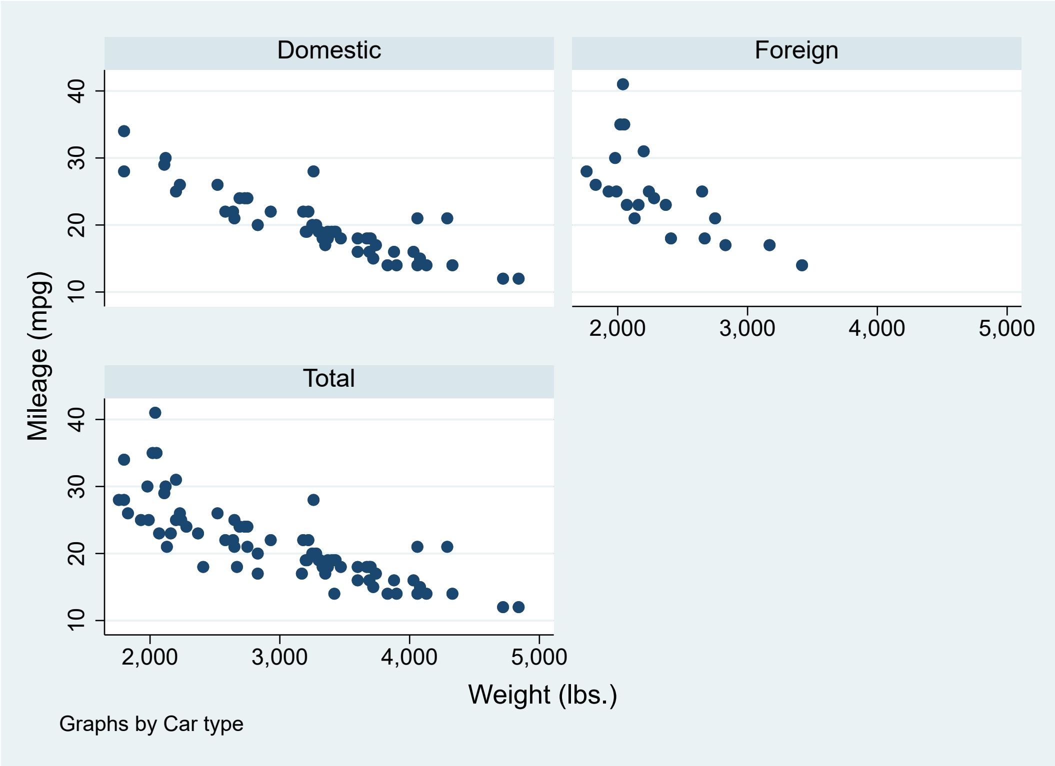
```
%stata by foreign: summarize mpg weight
```

```
-------------------------------------------------------------------------------
-> foreign = Domestic

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         52    19.82692    4.743297         12         34
      weight |         52    3317.115    695.3637       1800       4840

-------------------------------------------------------------------------------
-> foreign = Foreign

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         22    24.77273    6.611187         14         41
      weight |         22    2315.909    433.0035       1760       3420
```

We visualize **mpg** and **weight** for each group of cars using the scatter plot.

In [4]:

```
%%stata
scatter mpg weight, by(foreign, total)
```



Then we fit a linear regression model of **mpg** on **weight** and **foreign**.

In [5]:

```
%%stata
regress mpg weight i.foreign
```

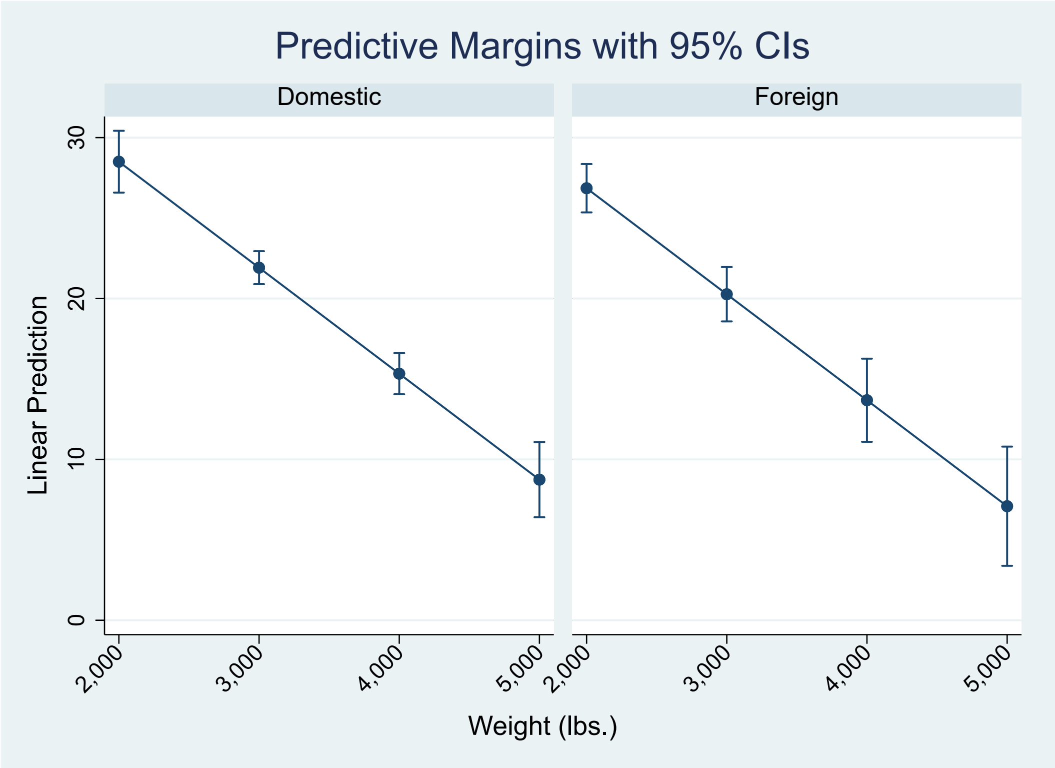
```
      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(2, 71)        =     69.75
       Model |  1619.2877         2  809.643849   Prob > F        =    0.0000
    Residual |  824.171761        71   11.608053   R-squared       =    0.6627
-------------+----------------------------------   Adj R-squared   =    0.6532
       Total |  2443.45946        73  33.4720474   Root MSE        =    3.4071

------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      weight |  -.0065879   .0006371   -10.34   0.000    -.0078583   -.0053175
             |
     foreign |
     Foreign |  -1.650029   1.075994    -1.53   0.130      -3.7955    .4954422
       _cons |    41.6797   2.165547    19.25   0.000     37.36172    45.99768
------------------------------------------------------------------------------
```

Next, we use **margins** to calculate the mean predicted values for various values of **weight** in increments of 1,000 between 2,000 and 5,000 and each group of cars. We then use **marginsplot** to show the results graphically.

In [6]:

```
%%stata
margins, at(weight=(2000(1000)5000)) over(foreign)
marginsplot, by(foreign) xlabel(, angle(forty_five))
```

```
. margins, at(weight=(2000(1000)5000)) over(foreign)

Predictive margins                              Number of obs     =         74
Model VCE    : OLS

Expression   : Linear prediction, predict()
over         : foreign

1._at        : 0.foreign
                   weight          =        2000
               1.foreign
                   weight          =        2000

2._at        : 0.foreign
                   weight          =        3000
               1.foreign
                   weight          =        3000

3._at        : 0.foreign
                   weight          =        4000
               1.foreign
                   weight          =        4000

4._at        : 0.foreign
                   weight          =        5000
               1.foreign
                   weight          =        5000

------------------------------------------------------------------------------
             |            Delta-method
             |     Margin   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
 _at#foreign |
 1#Domestic  |   28.50393   .9630195    29.60   0.000     26.58372    30.42414
  1#Foreign  |    26.8539   .7537561    35.63   0.000     25.35095    28.35685
 2#Domestic  |   21.91604   .5138592    42.65   0.000     20.89144    22.94065
  2#Foreign  |   20.26601   .8471116    23.92   0.000     18.57692    21.95511
 3#Domestic  |   15.32816   .6422785    23.87   0.000     14.04749    16.60882
  3#Foreign  |   13.67813   1.295714    10.56   0.000     11.09455    16.26171
 4#Domestic  |    8.74027   1.171673     7.46   0.000     6.404021    11.07652
  4#Foreign  |   7.090241   1.857949     3.82   0.000     3.385596    10.79489
------------------------------------------------------------------------------

. marginsplot, by(foreign) xlabel(, angle(forty_five))

  Variables that uniquely identify margins: weight foreign
```

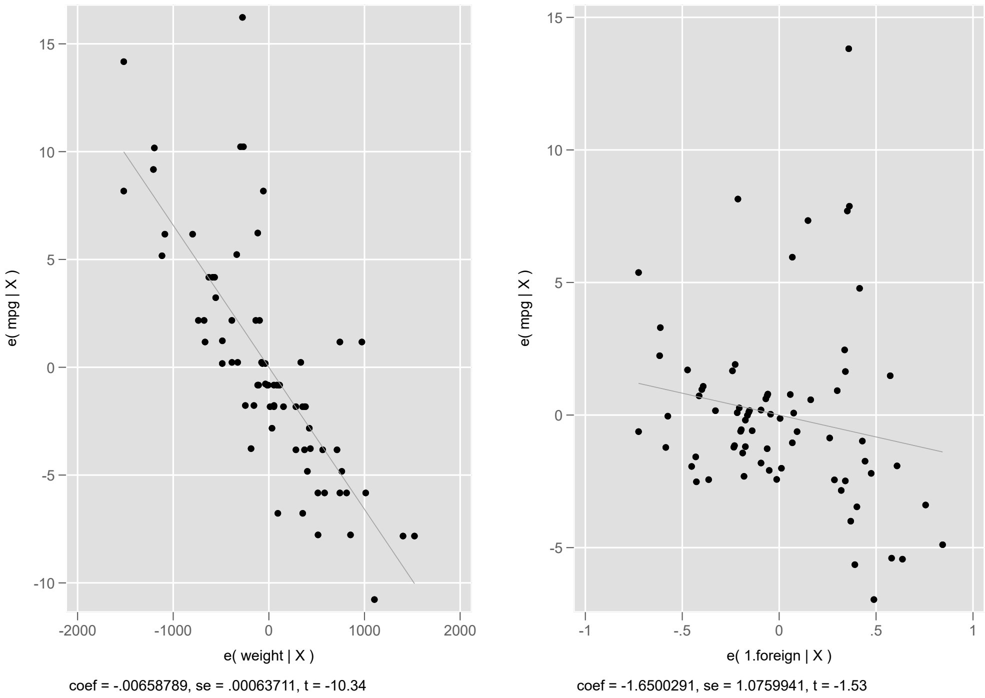


Last, we create a partial-regression leverage plot for all the regressors using the **avplots** command.

In [7]:

```
%%stata
set scheme plottigblind
avplots
```

```
. set scheme plottigblind

. avplots
```



Example 2: Load dataset from Python

There are many ways to load data from Python into Stata's current dataset in memory. For example

1. Pandas dataframes and Numpy arrays can be loaded directly into Stata.
2. The **Data** and **Frame** classes within the **Stata Function Interface (sfi)** module provide multiple methods for loading data from Python.
3. Stata can read in data from a variety of sources, many of which can be created in Python: Excel files, CSV files, SPSS and SAS dataset, and various databases.

We have data from the **National Longitudinal Survey** on young women's wages reported from 1968 through 1988. This dataset is stored in a csv file named **nlswork.csv**.

The goal is to use Stata to fit a model of wage as a function of each woman's age, job tenure, birth year, and level of education. We believe that random shocks that affect a woman's wage also affect her job tenure, so we treat tenure as endogenous. As additional instruments, we use her union status, number of weeks worked in the past year, and a dummy indicating whether she lives in a metropolitan area.

The plan is to load the data using Pandas dataframe and fit a single-equation instrumental-variables regression via the **ivregress** command.

In [8]:

```
import pandas as pd
nlswork = pd.read_csv('nlswork.csv')
nlswork
```

Out[8]:

	idcode	year	birth_yr	age	race	msp	nev_mar	grade	collgrad	not_smsa	...	so
0	1	70	51	18.0	black	0.0	1.0	12.0	0	0.0	...	
1	1	71	51	19.0	black	1.0	0.0	12.0	0	0.0	...	
2	1	72	51	20.0	black	1.0	0.0	12.0	0	0.0	...	
3	1	73	51	21.0	black	1.0	0.0	12.0	0	0.0	...	
4	1	75	51	23.0	black	1.0	0.0	12.0	0	0.0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
28529	5159	80	44	35.0	black	0.0	0.0	12.0	0	0.0	...	
28530	5159	82	44	37.0	black	0.0	0.0	12.0	0	0.0	...	
28531	5159	83	44	38.0	black	0.0	0.0	12.0	0	0.0	...	
28532	5159	85	44	40.0	black	0.0	0.0	12.0	0	0.0	...	
28533	5159	88	44	43.0	black	0.0	0.0	12.0	0	0.0	...	

28534 rows × 21 columns

Then we load the dataframe into Stata as current dataset and specify the labels to the variables of interest within Stata.

In [9]:

```
%%stata -d nlswork -force
label variable ln_wage "ln(wage/GNP deflator)"
label variable age "age in current year"
label variable birth_yr "birth year"
label variable grade "current grade completed"
label variable tenure "job tenure, in years"
label variable union "weeks unemployed last year"
label variable wks_work "weeks worked last year"
label variable msp "1 if married, spouse present"

describe
```

```
. label variable ln_wage "ln(wage/GNP deflator)"

. label variable age "age in current year"

. label variable birth_yr "birth year"

. label variable grade "current grade completed"

. label variable tenure "job tenure, in years"

. label variable union "weeks unemployed last year"

. label variable wks_work "weeks worked last year"

. label variable msp "1 if married, spouse present"

.
. describe

Contains data
  obs:        28,534
 vars:            21
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
idcode          long    %12.0g
year            long    %12.0g
birth_yr        long    %12.0g                birth year
age             double  %10.0g                age in current year
race            str9    %9s
msp             double  %10.0g                1 if married, spouse present
nev_mar         double  %10.0g
grade           double  %10.0g                current grade completed
collgrad        long    %12.0g
not_smsa        double  %10.0g
c_city          double  %10.0g
south           double  %10.0g
ind_code        double  %10.0g
occ_code        double  %10.0g
union           double  %10.0g                weeks unemployed last year
wks_ue          double  %10.0g
ttl_exp         double  %10.0g
tenure          double  %10.0g                job tenure, in years
hours           double  %10.0g
wks_work        double  %10.0g                weeks worked last year
ln_wage         double  %10.0g                ln(wage/GNP deflator)
-------------------------------------------------------------------------------
Sorted by:
     Note: Dataset has changed since last saved.
```

Next, we fit the model and push Stata's estimation results into Python, such as the coefficent vector **e(b)** and variance-covariance matrix **e(V)**. The estimation results is stored in **steret**, which is a Python dictionary.

In [10]:

```
%%stata -eret steret
// fit the model using the gmm estimator
ivregress gmm ln_wage age c.age#c.age birth_yr grade  ///
    (tenure = union wks_work msp), wmatrix(cluster idcode)

// e() results
ereturn list
```

```
. // fit the model using the gmm estimator
. ivregress gmm ln_wage age c.age#c.age birth_yr grade  ///
>     (tenure = union wks_work msp), wmatrix(cluster idcode)

Instrumental variables (GMM) regression          Number of obs   =     18,625
                                                 Wald chi2(5)    =    1807.17
                                                 Prob > chi2     =     0.0000
                                                 R-squared       =          .
GMM weight matrix: Cluster (idcode)              Root MSE        =     .46951

                             (Std. Err. adjusted for 4,110 clusters in idcode)
------------------------------------------------------------------------------
             |               Robust
     ln_wage |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      tenure |    .099221   .0037764    26.27   0.000     .0918194    .1066227
         age |   .0171146   .0066895     2.56   0.011     .0040034    .0302259
             |
 c.age#c.age |  -.0005191    .000111    -4.68   0.000    -.0007366   -.0003016
             |
    birth_yr |  -.0085994   .0021932    -3.92   0.000     -.012898   -.0043008
       grade |    .071574   .0029938    23.91   0.000     .0657062    .0774417
       _cons |   .8575071   .1616274     5.31   0.000     .5407231    1.174291
------------------------------------------------------------------------------
Instrumented:  tenure
Instruments:   age c.age#c.age birth_yr grade union wks_work msp


.
. // e() results
. ereturn list

scalars:
                  e(J) =  11.88787679472382
                e(rss) =  4105.610892967217
                  e(N) =  18625
               e(df_m) =  5
               e(rmse) =  .4695055741800723
                e(mss) =  -63.29808734918015
                 e(r2) =  .
               e(r2_a) =  .
               e(chi2) =  1807.171481540709
         e(iterations) =  1
            e(N_clust) =  4110
               e(rank) =  6

macros:
            e(cmdline) : "ivregress gmm ln_wage age c.age#c.age birth_yr gr.."
                e(cmd) : "ivregress"
          e(estat_cmd) : "ivregress_estat"
            e(wmatrix) : "cluster idcode"
                e(vce) : "cluster"
           e(clustvar) : "idcode"
            e(vcetype) : "Robust"
          e(estimator) : "gmm"
           e(footnote) : "ivreg_footnote"
      e(marginsnotok) : "Residuals SCores"
          e(marginsok) : "XB default"
            e(predict) : "ivregress_p"
             e(depvar) : "ln_wage"
              e(exogr) : "age c.age#c.age birth_yr grade"
              e(insts) : "age c.age#c.age birth_yr grade union wks_work msp"
```

```
           e(instd) : "tenure"
           e(title) : "Instrumental variables (GMM) regression"
      e(properties) : "b V"

matrices:
               e(b) :  1 x 6
               e(V) :  6 x 6
               e(S) :  8 x 8
               e(W) :  8 x 8

functions:
          e(sample)
```

Below, we list the entire contents of the Python dictionary **steret**.

In [11]:

```
steret
```

Out[11]:

```
{'e(J)': 11.887876794723823,
 'e(rss)': 4105.610892967217,
 'e(N)': 18625.0,
 'e(df_m)': 5.0,
 'e(rmse)': 0.46950557418007227,
 'e(mss)': -63.29808734918015,
 'e(r2)': 8.98846567431158e+307,
 'e(r2_a)': 8.98846567431158e+307,
 'e(chi2)': 1807.1714815407095,
 'e(iterations)': 1.0,
 'e(N_clust)': 4110.0,
 'e(rank)': 6.0,
 'e(cmdline)': 'ivregress gmm ln_wage age c.age#c.age birth_yr grade      (tenure
= union wks_work msp), wmatrix(cluster idcode)',
 'e(cmd)': 'ivregress',
 'e(estat_cmd)': 'ivregress_estat',
 'e(robust_epilog)': 'ivregress_epilog',
 'e(robust_prolog)': 'ivregress_prolog',
 'e(wmatrix)': 'cluster idcode',
 'e(vce)': 'cluster',
 'e(clustvar)': 'idcode',
 'e(vcetype)': 'Robust',
 'e(estimator)': 'gmm',
 'e(footnote)': 'ivreg_footnote',
 'e(marginsprop)': 'nolinearize',
 'e(marginsnotok)': 'Residuals SCores',
 'e(marginsok)': 'XB default',
 'e(predict)': 'ivregress_p',
 'e(depvar)': 'ln_wage',
 'e(exogr)': 'age c.age#c.age birth_yr grade',
 'e(insts)': 'age c.age#c.age birth_yr grade union wks_work msp',
 'e(instd)': 'tenure',
 'e(title)': 'Instrumental variables (GMM) regression',
 'e(properties)': 'b V',
 'e(b)': array([[ 9.92210073e-02,  1.71146216e-02, -5.19104153e-04,
        -8.59936559e-03,  7.15739528e-02,  8.57507064e-01]]),
 'e(V)': array([[ 1.42613627e-05, -9.76241713e-07, -2.63864849e-08,
        -7.77159613e-07, -3.01563638e-06,  8.56749367e-05],
       [-9.76241713e-07,  4.47498135e-05, -7.33276730e-07,
         2.72940218e-06, -1.83647532e-06, -7.58296139e-04],
       [-2.63864849e-08, -7.33276730e-07,  1.23108900e-08,
        -3.57163864e-08,  3.86949853e-08,  1.17566367e-05],
       [-7.77159613e-07,  2.72940218e-06, -3.57163864e-08,
         4.81015453e-06,  1.16191117e-07, -2.80185164e-04],
       [-3.01563638e-06, -1.83647532e-06,  3.86949853e-08,
         1.16191117e-07,  8.96286674e-06, -9.00370663e-05],
       [ 8.56749367e-05, -7.58296139e-04,  1.17566367e-05,
        -2.80185164e-04, -9.00370663e-05,  2.61234292e-02]]),
 'e(S)': array([[1.84225991e+07, 6.48996714e+08, 2.54724104e+07, 7.01502864e+06,
        5.36806324e+05, 1.53398668e+05, 3.94294503e+07, 3.22229084e+05],
       [6.48996714e+08, 2.31708366e+10, 8.79392757e+08, 2.44144659e+08,
        1.86264122e+07, 5.37264373e+06, 1.38258192e+09, 1.12612714e+07],
       [2.54724104e+07, 8.79392757e+08, 3.63515621e+07, 9.88420993e+06,
        7.59073876e+05, 2.14187357e+05, 5.50583611e+07, 4.50186699e+05],
       [7.01502864e+06, 2.44144659e+08, 9.88420993e+06, 2.78761599e+06,
        2.07158962e+05, 5.83301079e+04, 1.51171391e+07, 1.23720912e+05],
       [5.36806324e+05, 1.86264122e+07, 7.59073876e+05, 2.07158962e+05,
        1.59149056e+04, 4.49290588e+03, 1.15376185e+06, 9.47072185e+03],
       [1.53398668e+05, 5.37264373e+06, 2.14187357e+05, 5.83301079e+04,
```

```
          4.49290588e+03,  3.80211614e+03,  3.26881494e+05,  2.51143324e+03],
         [3.94294503e+07,  1.38258192e+09,  5.50583611e+07,  1.51171391e+07,
          1.15376185e+06,  3.26881494e+05,  8.75368616e+07,  6.78521312e+05],
         [3.22229084e+05,  1.12612714e+07,  4.50186699e+05,  1.23720912e+05,
          9.47072185e+03,  2.51143324e+03,  6.78521312e+05,  8.64778234e+03]]),
 'e(W)': array([[ 7.64077092e+00, -1.16467817e-01, -3.45417830e-01,
          -8.73750534e-02, -1.00703558e+02, -3.57303530e-01,
          -3.75313140e-02, -4.81605062e-01],
         [-1.16467817e-01,  1.81080290e-03,  7.50889019e-03,
           9.15320423e-04,  1.41514893e+00,  3.91377503e-03,
           2.69838887e-04,  5.60894455e-03],
         [-3.45417830e-01,  7.50889019e-03,  2.77106682e-01,
          -2.59628258e-02, -8.66908704e+00, -6.90985307e-02,
          -1.81124339e-02, -3.10466629e-02],
         [-8.73750534e-02,  9.15320423e-04, -2.59628258e-02,
           2.03171641e-01,  5.43785072e-01,  2.37092939e-02,
          -9.75767432e-04, -1.57125159e-02],
         [-1.00703558e+02,  1.41514893e+00, -8.66908704e+00,
           5.43785072e-01,  2.05170227e+03,  7.23678800e+00,
           1.26215920e+00,  5.53726790e+00],
         [-3.57303530e-01,  3.91377503e-03, -6.90985307e-02,
           2.37092939e-02,  7.23678800e+00,  6.94839823e+00,
           1.33578947e-02,  4.59819858e-01],
         [-3.75313140e-02,  2.69838887e-04, -1.81124339e-02,
          -9.75767432e-04,  1.26215920e+00,  1.33578947e-02,
           7.48017149e-03,  2.96991881e-02],
         [-4.81605062e-01,  5.60894455e-03, -3.10466629e-02,
          -1.57125159e-02,  5.53726790e+00,  4.59819858e-01,
           2.96991881e-02,  5.96402115e+00]])}
```

You can also access specific elements of the dictionary. For example, you can access **e(b)** and **e(V)** by typing **steret['e(b)']** and **steret['e(V)']** in Python.

In [12]:

```
steret['e(V)']
```

Out[12]:

```
array([[ 1.42613627e-05, -9.76241713e-07, -2.63864849e-08,
        -7.77159613e-07, -3.01563638e-06,  8.56749367e-05],
       [-9.76241713e-07,  4.47498135e-05, -7.33276730e-07,
         2.72940218e-06, -1.83647532e-06, -7.58296139e-04],
       [-2.63864849e-08, -7.33276730e-07,  1.23108900e-08,
        -3.57163864e-08,  3.86949853e-08,  1.17566367e-05],
       [-7.77159613e-07,  2.72940218e-06, -3.57163864e-08,
         4.81015453e-06,  1.16191117e-07, -2.80185164e-04],
       [-3.01563638e-06, -1.83647532e-06,  3.86949853e-08,
         1.16191117e-07,  8.96286674e-06, -9.00370663e-05],
       [ 8.56749367e-05, -7.58296139e-04,  1.17566367e-05,
        -2.80185164e-04, -9.00370663e-05,  2.61234292e-02]])
```

You can also access the above matrix using the **get()** method of the **Matrix** class in the **sfi** module.

In [13]:

```
from sfi import Matrix
import numpy as np

np.array(Matrix.get('e(V)'))
```

Out[13]:

```
array([[ 1.42613627e-05, -9.76241713e-07, -2.63864849e-08,
        -7.77159613e-07, -3.01563638e-06,  8.56749367e-05],
       [-9.76241713e-07,  4.47498135e-05, -7.33276730e-07,
         2.72940218e-06, -1.83647532e-06, -7.58296139e-04],
       [-2.63864849e-08, -7.33276730e-07,  1.23108900e-08,
        -3.57163864e-08,  3.86949853e-08,  1.17566367e-05],
       [-7.77159613e-07,  2.72940218e-06, -3.57163864e-08,
         4.81015453e-06,  1.16191117e-07, -2.80185164e-04],
       [-3.01563638e-06, -1.83647532e-06,  3.86949853e-08,
         1.16191117e-07,  8.96286674e-06, -9.00370663e-05],
       [ 8.56749367e-05, -7.58296139e-04,  1.17566367e-05,
        -2.80185164e-04, -9.00370663e-05,  2.61234292e-02]])
```

Example 3: Work with multiple datasets (frames)

Stata 16 introduced frames, allowing you to simultaneously work with multiple datasets in memory. The following example illustrates simple usage of multiple frames in Stata and how to switch frames between Stata and Python.

First, we load the *iris* datateset in Stata. This dataset is used in Fisher's (1936) article. Fisher obtained the iris data from Anderson (1935). The data consist of four features measured on 50 samples from each of three Iris species.

In [14]:

```
%%stata
use http://www.stata-press.com/data/r16/iris, clear
describe
label list species
```

```
. use http://www.stata-press.com/data/r16/iris, clear
(Iris data)

. describe

Contains data from http://www.stata-press.com/data/r16/iris.dta
  obs:           150                          Iris data
 vars:             5                          18 Jan 2018 13:23
                                              (_dta has notes)
--------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
iris            byte    %10.0g     species    Iris species
seplen          double  %4.1f                 Sepal length in cm
sepwid          double  %4.1f                 Sepal width in cm
petlen          double  %4.1f                 Petal length in cm
petwid          double  %4.1f                 Petal width in cm
--------------------------------------------------------------------------------
Sorted by:

. label list species
species:
           1 setosa
           2 versicolor
           3 virginica
```

Our goal is to build a classifier using those four features to detect the Iris type. We will use the **Random Forest** classification model within the **scikit-learn** Python package to achieve this goal. First, we split the data into the training and test datasets, and store the datasets in the **training** and **test** frames in Stata. The training dataset contains 80% of the observations and the test dataset contains 20% of the observations. Then we store the two frames into Python as Pandas dataframes with the same name.

In [15]:

```
%%stata -fouta training,test
// Split the original dataset into training and test
// dataset which contains 80% and 20% of observations respectively
splitsample, generate(svar, replace) split(0.8 0.2) show rseed(16)

// create two frames holding the the two datasets
frame put iris seplen sepwid petlen petwid if svar==1, into(training)
frame put iris seplen sepwid petlen petwid if svar==2, into(test)
```

```
. // Split the original dataset into training and test
. // dataset which contains 80% and 20% of observations respectively
. splitsample, generate(svar, replace) split(0.8 0.2) show rseed(16)

       svar |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |        120       80.00       80.00
          2 |         30       20.00      100.00
------------+-----------------------------------
      Total |        150      100.00

.
. // create two frames holding the the two datasets
. frame put iris seplen sepwid petlen petwid if svar==1, into(training)

. frame put iris seplen sepwid petlen petwid if svar==2, into(test)
```

Now we have 2 NumPy arrays in Python, **training** and **test**. Below we split each array into two sub-arrays to store the features and labels separately.

In [16]:

```
X_train = training[:, 1:]
y_train = training[:, 0]
X_test = test[:, 1:]
y_test = test[:, 0]
```

Then we use **X_train** and **y_train** to train the classification model.

In [17]:

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(max_depth=2, random_state=16)
clf.fit(X_train, y_train)
```

Out[17]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=2, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=16, verbose=0,
                       warm_start=False)
```

Next we use **X_test** and **y_test** to evaluate the performace of the training model. We also predict the species type of each flower and the probabilities that it belongs to the three species in the test dataset.

In [18]:

```
from sklearn import metrics

y_pred = clf.predict(X_test)
y_pred_prob = clf.predict_proba(X_test)
```

Next, in the **test** frame, we create a Byte variable **irispr** to store the predicted species types and three float variables to store the probabilities that each flower belongs to the three species types from the array **y_pred_prob**.

In [19]:

```
from sfi import Frame

fr = Frame.connect('test')
fr.addVarByte('irispr')
fr.addVarFloat('pr1')
fr.addVarFloat('pr2')
fr.addVarFloat('pr3')

fr.store('irispr', None, y_pred)
fr.store('pr1 pr2 pr3', None, y_pred_prob)
```

In Stata, we change the current working frame to **test**. We attach the value label **species** to **irispr**, and use the **tabulate** command to display a classification table. We also list the flowers that have been misclassified.

In [20]:

```
%%stata
frame change test
label values irispr species
label variable irispr predicted
tabulate iris irispr, row
list iris irispr pr1 pr2 pr3 if iris!=irispr
```

```
. frame change test

. label values irispr species

. label variable irispr predicted

. tabulate iris irispr, row

+----------------+
| Key            |
|----------------|
|   frequency    |
| row percentage |
+----------------+

      Iris |             predicted
   species |    setosa  versicolo  virginica |     Total
-----------+---------------------------------+----------
    setosa |        11          0          0 |        11
           |    100.00       0.00       0.00 |    100.00
-----------+---------------------------------+----------
versicolor |         0          9          3 |        12
           |      0.00      75.00      25.00 |    100.00
-----------+---------------------------------+----------
 virginica |         0          1          6 |         7
           |      0.00      14.29      85.71 |    100.00
-----------+---------------------------------+----------
     Total |        11         10          9 |        30
           |     36.67      33.33      30.00 |    100.00

. list iris irispr pr1 pr2 pr3 if iris!=irispr

     +-------------------------------------------------------+
     |       iris       irispr        pr1        pr2        pr3 |
     |-------------------------------------------------------|
 16. | versicolor    virginica   .0036523   .3280099   .6683378 |
 18. | versicolor    virginica   .0004762   .0687575   .9307663 |
 19. | versicolor    virginica   .0031241   .4210142   .5758616 |
 26. |  virginica   versicolor   .0046717   .5435431   .4517851 |
     +-------------------------------------------------------+
```

Example 4: Integrate with Mata

This example will illustrate how to use the **%%mata** magic command to combine Python's capabilities with features of **Mata**, Stata's matrix programming language.

For illustrative purposes, we generate two random NumPy arrays in Python, **X** and **y**, representing the observations of the independent variables and the response variable. Our goal is to fit a linear regression and get the coefficients b, their standard errors, t statistics and p-values.

In [21]:

```
import numpy as np
np.random.seed(16)

# 100,000 observations and 7 variables
X = np.random.random((100000, 7))
y = np.random.random((100000, 1))
```

Then we push **X** and **y** to Mata using the magic command **%%mata**, and fit the model.

In [22]:

```
%%mata -m X,y
//add the constant term to X
cons = J(100000,1,1)
X = (X, cons)

// calculate b
b = invsym(X'X)*X'y

// calcuate residuals and square of s
e  = y - X*b
n  = rows(X)
k  = cols(X)
s2 = (e'e)/(n-k)

// calculate V
V = s2*invsym(X'X)
```

```
. mata
------------------------------------------------- mata (type end to exit) -----
: //add the constant term to X
: cons = J(100000,1,1)

: X = (X, cons)

:
: // calculate b
: b = invsym(X'X)*X'y

:
: // calcuate residuals and square of s
: e  = y - X*b

: n  = rows(X)

: k  = cols(X)

: s2 = (e'e)/(n-k)

:
: // calculate V
: V = s2*invsym(X'X)

: end
--------------------------------------------------------------------------------
```

Next we calculate the standard errors, t statistics and p-values. Note that the standard errors are just the square roots of the diagonals of **V**.

In [23]:

```
%%mata
se = sqrt(diagonal(V))
tstat = b:/se
pval = 2*ttail(n-k, abs(b:/se))
```

```
. mata
------------------------------------------------- mata (type end to exit) -----
: se = sqrt(diagonal(V))

: tstat = b:/se

: pval = 2*ttail(n-k, abs(b:/se))

: end
-------------------------------------------------------------------------------
```

Next, we store the above results into a Mata matrix **ols_est** and push it back to Python.

In [24]:

```
%%mata -outm ols_est
ols_est = (b, se, tstat, pval)
ols_est
```

```
. mata
------------------------------------------------- mata (type end to exit) -----
: ols_est = (b, se, tstat, pval)

: ols_est
                  1              2              3              4
    +-------------------------------------------------------------+
  1 |  -.0028814142    .0031559598   -.9130072687      .36124092  |
  2 |   .0021227247    .0031590697    .6719461302     .501619544  |
  3 |  -.0001837604    .0031580164    -.058188558     .953598551  |
  4 |   .0002157194    .0031606842     .068250868     .945586071  |
  5 |    .001669965     .003153824    .5295048033    .5964564888  |
  6 |  -.0009795437    .0031563296   -.3103426322    .7563010614  |
  7 |   .0029998483     .003158516    .9497650963    .3422339202  |
  8 |   .4988382533    .0042774646    116.6200762              0  |
    +-------------------------------------------------------------+

: end
-------------------------------------------------------------------------------
```

In Python, we now have a NumPy array named **ols_est** with the same contents as the above Mata matrix.

In [25]:

```
ols_est
```

Out[25]:

```
array([[-2.88141423e-03,  3.15595979e-03, -9.13007269e-01,
         3.61240920e-01],
       [ 2.12272468e-03,  3.15906973e-03,  6.71946130e-01,
         5.01619544e-01],
       [-1.83760422e-04,  3.15801643e-03, -5.81885580e-02,
         9.53598551e-01],
       [ 2.15719443e-04,  3.16068425e-03,  6.82508680e-02,
         9.45586071e-01],
       [ 1.66996498e-03,  3.15382404e-03,  5.29504803e-01,
         5.96456489e-01],
       [-9.79543651e-04,  3.15632965e-03, -3.10342632e-01,
         7.56301061e-01],
       [ 2.99984826e-03,  3.15851600e-03,  9.49765096e-01,
         3.42233920e-01],
       [ 4.98838253e-01,  4.27746465e-03,  1.16620076e+02,
         0.00000000e+00]])
```

We can compare **ols_est** with the results reported by Stata's **regress** command. Below, we will use the **%%stata** magic command to execute it. Before we do that, we combine **X** and **y** into one NumPy array called **data**.

In [26]:

```
data = np.concatenate((X, y), axis=1)
```

Then we push **data** to Stata. When reading a NumPy array into Stata, the variables are named **v1**, **v2**, ... by default, so we rename them. Then we fit the regression model.

In [27]:

```
%%stata -d data -force
describe

// rename the last variable to y
rename v8 y

// rename the other variables, prefixing them with an x
rename v* x*

regress y x1-x7
```

```
. describe

Contains data
  obs:       100,000
 vars:             8
--------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
v1              double  %10.0g
v2              double  %10.0g
v3              double  %10.0g
v4              double  %10.0g
v5              double  %10.0g
v6              double  %10.0g
v7              double  %10.0g
v8              double  %10.0g
--------------------------------------------------------------------------------
Sorted by:
     Note: Dataset has changed since last saved.

.
. // rename the last variable to y
. rename v8 y

.
. // rename the other variables, prefixing them with an x
. rename v* x*

.
. regress y x1-x7

      Source |       SS           df       MS      Number of obs   =   100,000
-------------+----------------------------------   F(7, 99992)     =      0.37
       Model |  .213224988         7  .030460713   Prob > F        =    0.9218
    Residual |  8297.31719    99,992   .08297981   R-squared       =    0.0000
-------------+----------------------------------   Adj R-squared   =   -0.0000
       Total |  8297.53042    99,999  .082976134   Root MSE        =    .28806

------------------------------------------------------------------------------
           y |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
          x1 |  -.0028814    .003156    -0.91   0.361    -.0090671    .0033042
          x2 |   .0021227   .0031591     0.67   0.502     -.004069    .0083145
          x3 |  -.0001838    .003158    -0.06   0.954    -.0063734    .0060059
          x4 |   .0002157   .0031607     0.07   0.946    -.0059792    .0064106
          x5 |     .00167   .0031538     0.53   0.596    -.0045115    .0078514
          x6 |  -.0009795   .0031563    -0.31   0.756    -.0071659    .0052068
          x7 |   .0029998   .0031585     0.95   0.342    -.0031908    .0091905
       _cons |   .4988383   .0042775   116.62   0.000     .4904545     .507222
------------------------------------------------------------------------------
```

Example 5: Call Stata using API functions

In addition to the magic commands, you can also call Stata using the API functions defined in the **stata** module. For illustration purpose, we use the Boston housing price dataset from the **scikit-learn** package.

In [28]:

```
from sklearn import datasets

bos = datasets.load_boston()
print(bos.DESCR)
```

```
.. _boston_dataset:

Boston house prices dataset
---------------------------

**Data Set Characteristics:**

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive. Median Value (attrib
ute 14) is usually the target.

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.f
t.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 othe
rwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
https://archive.ics.uci.edu/ml/machine-learning-databases/housing/


This dataset was taken from the StatLib library which is maintained at Carnegie Me
llon University.

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostics
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers that add
ress regression
problems.

.. topic:: References

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential Data
and Sources of Collinearity', Wiley, 1980. 244-261.
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. In Proc
eedings on the Tenth International Conference of Machine Learning, 236-243, Univer
sity of Massachusetts, Amherst. Morgan Kaufmann.
```

We store the dataset into a Pandas dataframe named **boston**.

In [29]:

```
import pandas as pd
boston = pd.DataFrame(bos.data)
boston.columns = bos.feature_names
boston['MEDV'] = bos.target
boston.head()
```

Out[29]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90

Then we load the Pandas dataframe into Stata using the **dataframe_to_data()** function of the **stata** module.

In [30]:

```
from pystata import stata
stata.dataframe_to_data(boston, force=True)
```

Next, we specify the variable labels and attach a value label to **CHAS**, which indicates whether the tract bounds the Charles River, using the **run()** method.

In [31]:

```
stata.run('''
label variable MEDV "Median value of owner-occupied homes in $1000s"
label variable CRIM "Town crime rate, per capita"
label variable RM "Average number of rooms per dwelling"
label variable CHAS "Tract bounds the Charles River (= 1 if tract bounds river; 0 otherwise)"

label define bound 1 "Yes" 0 "No", replace
label values CHAS bound
''')
```

```
.
. label variable MEDV "Median value of owner-occupied homes in $1000s"

. label variable CRIM "Town crime rate, per capita"

. label variable RM "Average number of rooms per dwelling"

. label variable CHAS "Tract bounds the Charles River (= 1 if tract bounds rive
> r; 0 otherwise)"

.
. label define bound 1 "Yes" 0 "No", replace

. label values CHAS bound
```

Afterwards, we fit a linear regression model of the median home value (**MEDV**) on the town crime rate (**CRIM**), the average number of rooms per dwelling (**RM**), and whether the house is close to the Charles River (**CHAS**). Then we predict the median home values, storing these values in the variable **midval**.

In [32]:

```
stata.run('''
regress MEDV CRIM RM i.CHAS
predict midval
''')
```

```
.
. regress MEDV CRIM RM i.CHAS

      Source |       SS           df       MS      Number of obs   =       506
-------------+----------------------------------   F(3, 502)       =    206.73
       Model |  23607.3566         3  7869.11888   Prob > F        =    0.0000
    Residual |  19108.9388       502  38.0656151   R-squared       =    0.5527
-------------+----------------------------------   Adj R-squared   =    0.5500
       Total |  42716.2954       505  84.5867236   Root MSE        =    6.1697

------------------------------------------------------------------------------
        MEDV |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
        CRIM |  -.2607244   .0327369    -7.96   0.000    -.3250427   -.1964061
          RM |    8.27818   .4018204    20.60   0.000     7.488723    9.067637
             |
        CHAS |
         Yes |   3.763037   1.086199     3.46   0.001     1.628981    5.897093
       _cons |  -28.81068   2.563308   -11.24   0.000    -33.84682   -23.77455
------------------------------------------------------------------------------

. predict midval
(option xb assumed; fitted values)
```

Then we use the **get_ereturn()** function to push all the **e()** results to Python as a dictionary named steret. And we use the **data_to_array()** function to store the predicted values (**midval**) in a NumPy array named **stpred**.

In [33]:

```
steret = stata.get_ereturn()
stpred = stata.data_to_array('midval')
```

Conclusion

- The **pystata** package provides tight integration bewtween Stata and Python using IPython magic commands and a suite of API functions.
- Users can execute Stata code in all Python's environment to leverage Stata's capabilities.
- The **pystata** package can be used together with the [Stata Function Interface (sfi) (https://www.stata.com/python/api16/index.html)](https://www.stata.com/python/api16/index.html) module to access Stata's core features.