Stata and Python Integration

Zhao Xu

Principal Software Engineer
StataCorp LLC

Nov 19, 2019

Introduction

Stata 16 introduces tight integration with Python. With the new **python** suite of commands, users can

- Embed and execute Python code from within Stata
  - Invoke Python interactively
  - Embed Python code in do-files
  - Embed Python code in ado-files
  - Run a Python script file (**.py**) within Stata

Introduction (cont.)

- Interact with Stata through the Stata Function Interface (**sfi**) module
  - Access Stata's core features within Python
  - Pass data and results back and forth between Stata and Python
- Leverage the language features and capabilities of both in one environment
- Output Python results directly within Stata

Introduction (cont.)

Users can use any Python packages directly within Stata with the integration.

- Numpy and Pandas (Numerical Analysis)
- Matplotlib and seaborn (Visualization)
- BeautifulSoup, lxml, and Scrapy (Web Scraping)
- Scikit learn, Tensorflow, and Keras (Machine Learning)
- NLTK and spaCy (Natural Language Processing)

Setup Python environment

- Search for Python installations on the current system

  ```
  . python search
  ```

- Set which version of Python to use

  ```
  . python set exec pyexecutable [, permanently]
  ```

- Query current Python settings and system information

  ```
  . python query
  ```

Use Python interactively

Type **python** or **python:** in the Command window to enter the interactive environment. It works like a **Read-Eval-Print-Loop (REPL)** environment.

```
. python
---------------------------------------- python (type end to exit) -----------
>>> print("Hello World!")
Hello World!
>>> str = "This is a Python string"
>>> str.split(" ")
['This', 'is', 'a', 'Python', 'string']
>>>
>>> mysum = 0
>>> for i in range(10):
...     mysum+=i
...
>>> mysum
45
>>> end
------------------------------------------------------------------------------
```

Use Python interactively (cont.)

Within the interactive environment, you can directly

- Access Stata macros
- Execute Stata code

```
. local anum = 3
. local astr = "This is a Stata string"
. python
---------------------------------------- python (type end to exit) -----------
>>> ‘anum’
3
>>> "‘astr’".split(" ")
[’This’, ’is’, ’a’, ’Stata’, ’string’]
>>>
>>> stata: sysuse auto, clear
(1978 Automobile Data)
>>> stata: summarize mpg

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         74     21.2973    5.785503         12         41
>>> end
------------------------------------------------------------------------------
```

Use Python interactively (cont.)

You can also access Stata through the Stata Function Interface **(sfi)** Python module.

```
. sysuse auto, clear
. python:
------------------------------------------ python (type end to exit) -----------
>>> from sfi import Data
>>> pymake = Data.get('make')
>>> type(pymake)
<class 'list'>
>>> pybrand = [m.split(" ")[0] for m in pymake]
>>> Data.addVarStr('brand', 9)
>>> Data.store('brand', None, pybrand)
>>> end
--------------------------------------------------------------------------------
. list make brand in 1/5, noobs
  +----------------------+
  | make            brand |
  |----------------------|
  | AMC Concord       AMC |
  | AMC Pacer         AMC |
  | AMC Spirit        AMC |
  | Buick Century   Buick |
  | Buick Electra   Buick |
  +----------------------+
```

Choose between python and python:

Distinction betwen **python** or **python:**

```
. python
---------------------------------------- python (type end to exit) -----------
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
r(7102);
>>>
```

```
. python:
---------------------------------------- python (type end to exit) -----------
>>> a
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
------------------------------------------------------------------------------
r(7102);

.
```

Web scraping population of Lafayette

Demographics [edit]

As of the census[15] of 2010, there were 120,623 people, 43,506 households, and 27,104 families residing in the city. The population density was 2,316.7 people per square mile (894.5/km²). There were 46,865 housing units at an average density of 984.7 per square mile (380.2/km²). The racial makeup of the city was 68.23% White, 28.51% African American, 0.25% Native American, 1.44% Asian, 0.02% Pacific Islander, 0.58% from other races, and 0.97% from two or more races. Hispanic or Latino of any race were 7.88% of the population. In 2010, 84.2% of the population over the age of five spoke English at home, and 11.5% of the population spoke French or Cajun French, a dialect that developed in Louisiana.[16]

There were 43,506 households out of which 31.3% had children under the age of 18 living with them, 43.9% were married couples living together, 14.6% had a female householder with no husband present, and 37.7% were non-families. Nearly 29.4% of all households were made up of individuals, and 8.0% had someone living alone who was 65 years of age or older. The average household size was 2.43 and the average family size was 3.07.

Historical population

Census	Pop.	%±
1860	498	—
1870	777	56.0%
1880	815	4.9%
1890	2,106	158.4%
1900	3,314	57.4%
1910	6,392	92.9%
1920	7,855	22.9%
1930	14,635	86.3%
1940	19,210	31.3%
1950	33,541	74.6%
1960	40,400	20.4%
1970	68,908	70.6%
1980	80,584	16.9%
1990	94,440	17.2%
2000	110,257	16.7%

Web scraping population of Lafayette

Step 1: Extract the data

```
. python:
------------------------------------- python (type end to exit) -----------
>>> import pandas as pd
>>> page = pd.read_html("https://en.wikipedia.org/wiki/Lafayette,_Louisiana")
>>> tbl = page[5].iloc[0:-1, [0,1,3]]
>>> tbl_val = tbl.values.tolist()
>>> end
------------------------------------------------------------------------------
```

Web scraping population data

Step 2: Store data into Stata

```
. python:
------------------------------------- python (type end to exit) -----------
>>> from sfi import Data
>>> Data.setObs(len(tbl_val))
>>> stata: gen census = ""
>>> stata: gen pop = ""
>>> stata: gen increase = ""
>>> Data.store(None, None, tbl_val)
>>> end
------------------------------------------------------------------------------

. list in 1/5, clean

     census    pop   increase
  1.   1860    498          -
  2.   1870    777      56.0%
  3.   1880    815       4.9%
  4.   1890   2106     158.4%
  5.   1900   3314      57.4%
```

Embed Python code in do-files

Python code can also be embedded in a do-file so that you can run multiple Python statements sequentially. All you need to do is to place the Python code within the **python**/**end** block or **python:**/**end** block.

```
/* pyex1.do */
/* Stata code block */
some Stata code

/* Python code block */
python:

end

/* Stata code block */
some Stata code
```

Embed Python code in do-files (cont.)

- Include Stata code and Python code in a do-file, say **pydo1.do**

```
sysuse auto, clear

python:
from sfi import Data
pymake = Data.get('make')
pybrand = [m.split(" ")[0] for m in pymake]
Data.addVarStr('brand', 9)
Data.store('brand', None, pybrand)
end

list make brand in 1/5, noobs
```

- Execute the do-file as you normally would

```
. do pydo1
```

Classify Iris flowers

Iris Versicolor

Iris Setosa

Iris Virginica

The *Iris* dataset was introduced in Fisher's (1936) article. It consists of **4** features measured on **50** samples from each of three species: *Iris Versicolor*, *Iris Setosa*, and *Iris Virginica*.

Classify Iris flowers (.cont)

```
. 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:
```

Classify Iris flowers (.cont)

Step 1: split the original dataset

```
clear all
use http://www.stata-press.com/data/r16/iris, clear
describe

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

// create two frames holding the the two datasets
frame put if group==1, into(training)
frame put if group==2, into(test)
```

Classify Iris flowers (.cont)

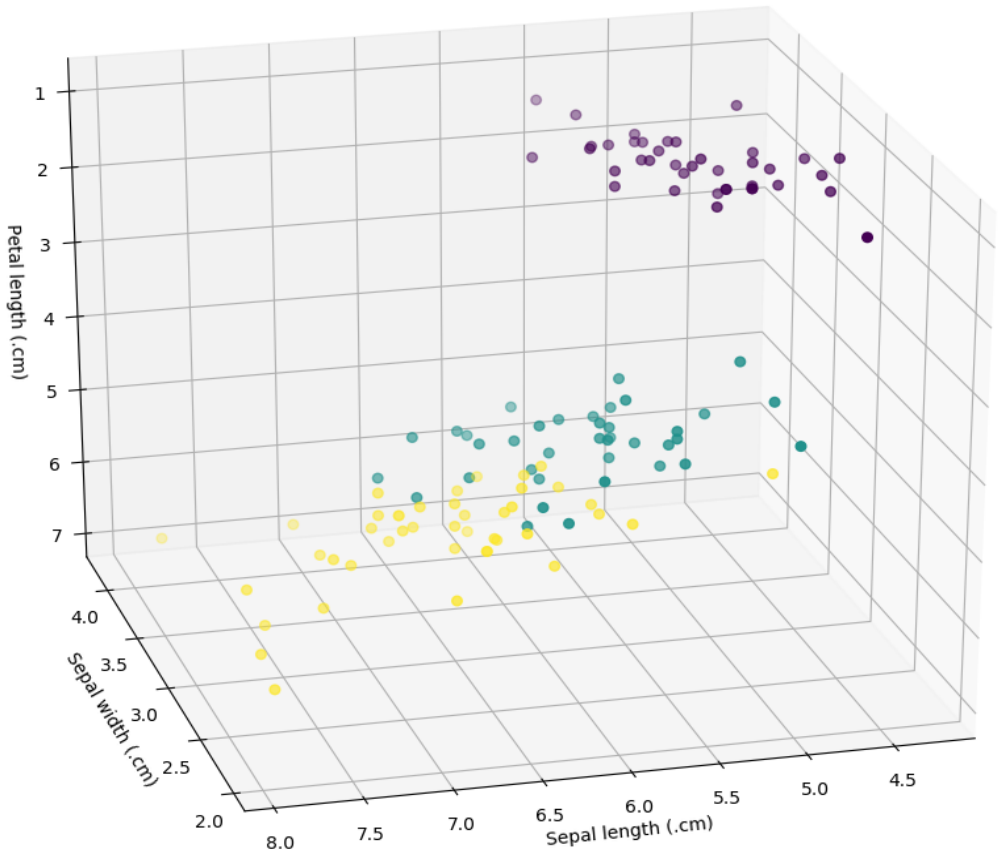
Step 2: draw a 3D plot using the training dataset

```
python:
from sfi import Frame
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Use the sfi Frame class to pull data from Stata training frame into Python
train = Frame.connect('training')
train_X = np.array(train.get("seplen sepwid petlen petwid"))
train_y = np.array(train.get("iris"))

fig = plt.figure(1, figsize=(10, 8))
ax = Axes3D(fig, elev=-155, azim=105)
ax.scatter(train_X[:, 0], train_X[:, 1], train_X[:, 2], c=train_y, s=30)
ax.set_xlabel("Sepal length (.cm)")
ax.set_ylabel("Sepal width (.cm)")
ax.set_zlabel("Petal length (.cm)")
plt.savefig("img2.png")
end
```

Classify Iris flowers (.cont)



Classify Iris flowers (.cont)

Step 3: train the model with the training dataset using Support Vector Machine classifier

```
python:
from sklearn.svm import SVC
svc_clf = SVC(gamma='auto', random_state=16)
svc_clf.fit(train_X, train_y)
end
```

Classify Iris flowers (.cont)

Step 4: make predictions with the test dataset and store results back into the test frame

```
python:
test = Frame.connect('test')
test_X = np.array(test.get("seplen sepwid petlen petwid"))
irispr = svc_clf.predict(test_X)

test.addVarByte('irispr')
test.store('irispr', None, irispr)
end
```

Classify Iris flowers (.cont)

Step 5: see prediction results in Stata

```
frame change test
label values irispr species
label variable irispr predicted
tabulate iris irispr, row
```

Classify Iris flowers (.cont)

```
+----------------+
| 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          0          7 |         7
           |      0.00       0.00     100.00 |    100.00
-----------+---------------------------------+----------
     Total |        11          9         10 |        30
           |     36.67      30.00      33.33 |    100.00
```

A simple example: calculate the cumulative sum of a variable

Python code can be embedded and executed in ado-files too. This is useful when you are interested in extending Stata by adding a new command.

```
program varsum
    version 16.0
    syntax varname [if] [in]

    marksample touse
    python: calcsum("`varlist'", "`touse'")
    display as txt " sum of `varlist': " as res r(sum)
end

python:
from sfi import Data, Scalar
def calcsum(varname, touse):
    x = Data.get(varname, None, touse)
    Scalar.setValue("r(sum)", sum(x))
end
```

A simple example: calculate the cumulative sum of a variable (.cont)

```
. sysuse auto, clear
(1978 Automobile Data)

. varsum price
 sum of price: 456229

. varsum price if foreign
 sum of price: 140463

. varsum price if mpg > 25 & foreign
 sum of price: 29921
```

Revisit the classification example

General idea:

- Split the **iris** dataset into training and test datasets.
- One command, **pysvm**, trains a Support Vector Machine classification model using the training dataset.
- The other command, **pysvmpredict**, predicts the test dataset.

Revisit the classification example (.cont)

pysvm.ado: training command

```
program pysvm, eclass
    version 16
    syntax varlist [if] [in]

    marksample touse
    gettoken label feature : varlist

    python: dosvm("`label'", "`feature'", "`touse'")
    ereturn local features `feature'
end

python:
from sfi import Data
import numpy as np
from sklearn.svm import SVC
import __main__

def dosvm(label, features, touse):
    X = np.array(Data.get(features, None, touse))
    y = np.array(Data.get(label, None, touse))

    svm_clf = SVC(gamma='auto', random_state=16)
    svm_clf.fit(X, y)

    __main__.svm_clf = svm_clf

end
```

Revisit the classification example (.cont)

pysvmpredict.ado: prediction command

```
program pysvmpredict
    version 16
    syntax, predict(name)

    // get the feature variables stored by pysvm
    local features = e(features)

    python: dopredict("`features'", "`predict'")
end

python:
from sfi import Data
import numpy as np
from sklearn.svm import SVC
import __main__

def dopredict(features, predict):
    svm_clf = __main__.svm_clf

    X = np.array(Data.get(features))
    y_pred = svm_clf.predict(X)

    Data.addVarByte(predict)
    Data.store(predict, None, y_pred)

end
```

Revisit the classification example (.cont)

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

. splitsample, generate(group, replace) split(0.8 0.2) show rseed(16)

. pysvm iris seplen sepwid petlen petwid if group==1

. pysvmpredict, predict(irispr)

. label values irispr species

. label variable irispr predicted

. tabulate iris irispr if group==2, row
```

Revisit the classification example (.cont)

```
+----------------+
| 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          0          7 |         7
           |      0.00       0.00     100.00 |    100.00
-----------+---------------------------------+----------
     Total |        11          9         10 |        30
           |     36.67      30.00      33.33 |    100.00
```

Run a Python script file (.py) within Stata

A Python script **(.py)** file can be executed directly from within Stata using the **python script** command.

It is useful when you want to isolate Stata code and Python code so that you do not need to bother with the language differences between them, such as the indentations, comments, etc.

Revisit the classification example

pyex.py

```
from sfi import Frame
import numpy as np
import sys

# get the training and test frames from Stata
train_frame = sys.argv[1]
test_frame = sys.argv[2]

# use the sfi Frame class to pull data from Stata training frame into Python
train = Frame.connect(train_frame)
train_X = np.array(train.get("seplen sepwid petlen petwid"))
train_y = np.array(train.get("iris"))

# train the model with the training dataset using SVC classifier
from sklearn.svm import SVC
svc_clf = SVC(gamma='auto', random_state=16)
svc_clf.fit(train_X, train_y)

# make predictions with the test dataset and store results back into the test frame
test = Frame.connect(test_frame)
test_X = np.array(test.get("seplen sepwid petlen petwid"))
irispr = svc_clf.predict(test_X)
test.addVarByte('irispr')
test.store('irispr', None, irispr)
```

Revisit the classification example (.cont)

pyex.do

```
clear all
use http://www.stata-press.com/data/r16/iris

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

// create two frames holding the the two datasets
frame put if group==1, into(training)
frame put if group==2, into(test)

// pass the two frame names into the .py script
// train the model and make predictions
python script pyex.py, args(training test)

// see prediction results in Stata
frame change test
label values irispr species
label variable irispr predicted
tabulate iris irispr, row
```

Revisit the classification example (.cont)

```
+----------------+
| 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          0          7 |         7
           |      0.00       0.00     100.00 |    100.00
-----------+---------------------------------+----------
     Total |        11          9         10 |        30
           |     36.67      30.00      33.33 |    100.00
```

Introduction to sfi module

The Stata Function Interface **(sfi)** module allows users to interact Python's capabilities with core features of Stata. The module can be used interactively or in do-files and ado-files or in a .py file.

Class	Description
Characteristic	This class provides access to Stata characteristics.
Data	This class provides access to the Stata dataset in memory.
Datetime	This class provides access to Stata date and time values.
Frame	This class provides access to a Stata data frame.
Macro	This class provides access to Stata macros.
Mata	This class provides access to global Mata matrices.
Matrix	This class provides access to Stata matrices.
Missing	This class provides tools for handling Stata missing values.
Platform	A set of utilities for getting platform information.
Preference	A set of utilities for loading and saving preferences.
Scalar	This class provides access to Stata scalars.
SFIToolkit	This class provides a set of tools for interacting with Stata.
StrLConnector	This class facilitates access to Stata's **strL** datatype.
ValueLabel	This class provides access to Stata's value labels.

Introduction to sfi module (cont.)

Each class defines various functions letting you access Stata's characteristics, current dataset, frames, data and time, macros, scalars, matrices, value labels, global Mata matrices, and so on.

Within Python, you can access those classes in the following three ways:

- import the whole module

```
>>> import sfi
>>> sfi.Data.getObsTotal()
>>> sfi.Matrix.get("e(V)")
```

Introduction to sfi module (cont.)

- import all classes in the module at once

```
>>> from sfi import *
>>> Data.getObsTotal()
>>> Matrix.get("e(V)")
```

- import specific class(es) in the module

```
>>> from sfi import Data, Matrix
>>> Data.getObsTotal()
>>> Matrix.get("e(V)")
```

Conclusion

- Embed and execute Python code within Stata
- Interact Python's capabilities with Stata's core features through the Stata Function Interface (**sfi**) module
- Some examples include
  - Web scraping
  - Creating 3D plots
  - Training machine learning models

Additional resources

- Quick overview of Python integration in Stata
  https://www.stata.com/new-in-stata/python-integration/
- Full documentation
  https://www.stata.com/manuals/ppython.pdf
- Stata Function Interface (sfi) documentation and examples
  https://www.stata.com/python/api16/index.html

Thank You!