Applied Econometrics using Python

Alexandre Rodrigues Loures
Federal University of Paraiba
Center of Applied Social Sciences
Graduate Program in Economics

January 13, 2017

Chapter 1

Introduction

1.1 Basic Python Functions

1.1.1 Using Python as a Calculator

You can use the Python command line to perform the four basic math operations and other simple operations.

Basic Mathematical Operations with Python

```
>>> ## Examples:

>>> ## making a sum

>>> 2 + 2
4
```

Basic Mathematical Operations with Python

```
>>> ## making a subtraction

>>> 15 - 3
12
```

Basic Mathematical Operations with Python

```
>>> ## making a multiplication

>>> 2 * 8
16
```

Basic Mathematical Operations with Python

```
>>> ## making a division

>>> 35/7
5
```

Basic Mathematical Operations with Python

```
>>> ## extracting the square root
>>> import math

>>> math.sqrt (16)
4.0
```

It is also possible to perform more complex operations.

More Complex Operations with Python

```
>>> ## Examples:

>>> ## calculating the neperian of 8

>>> math.log (8)
2.0794415416798357
```

More Complex Operations with Python

```
>>> ## calculating the logarithm in base 10 of 8

>>> math.log (8, 10)
0.9030899869919434
```

More Complex Operations with Python

```
>>> ## calculating the logarithm in base 5 of 8

>>> math.log (8, 5)
1.2920296742201791
```

More Complex Operations with Python

```
>>> ## calculating the tangent of 9

>>> math.tan (9)
-0.45231565944180985
```

Another feature of Python is being able to assign any value to an object and then use it to do both simple and complex operations.

Assigning any value to an object with Python

```
>>> ## Examples:

>>> ## equaling the divison of 200 by 10 to 'q'

>>> q = 200/10

>>> ## obtaining the result of the previous divison

>>> q
20
```

```
>>> ## multiplying the object 'q' by 2

>>> q * 2
40

>>> ## getting the sin of object 'q'

>>> math.sin (q)
0.9129452507276277

>>> ## adding 30 to object 'q'

>>> q + 30
50

>>> ## subtracting object 'q' from 100

>>> 100 - q
80
```

Observations with Python

```
>>> ## constants:

>>> ## obtaining the value of 'Pi'

>>> math.pi
3.141592653589793

>>> ## obtaining the value of 'e'

>>> math.e
2.718281828459045

>>> ## powers with Python

>>> x = 2

>>> pow (x, 3)
8

>>> ## or

>>> x**3
8

>>> ## the logarithm in base 10 can also be obtained as follows:

>>> math.log10 (8)
0.9030899869919435
```

Step-by-Step Solve Nonlinear Equations with Python

```
>>> from numpy import *
>>> from scipy.optimize import *

>>> def myFunction (z):
...             x = z[0]
...             y = z[1]
...             w = z[2]
...             F = empty ((3))
...             F[0] = pow (x,2) + pow (y,2) - 20
...             F[1] = y - pow (x,2)
...             F[2] = w + 5 - x*y
...             return F
...
>>> zGuess = array ([1,1,1])

>>> z = fsolve (myFunction, zGuess)

>>> print(z)
[ 2.  4.  3.]
```

Chapter 2

Basic Statistics

2.1 Measures of Position

2.1.1 Average

Simple arithmetic mean

Simple arithmetic mean is the sum of the values of a series divided by the total number of elements of that series. This is the most common day-to-day average. The mathematical representation being as follows:

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n} \tag{2.1}$$

where:

$x_i \rightarrow$ is each element of the series; and

$n \rightarrow$ is the number of elements in the series.

To calculate the simple arithmetic mean in Python, the following syntax is used:

Calculating the Simple Arithmetic Mean with Python

```
>>> ## Example:

>>> import statistics

>>> ## creating any data series

>>> x = [15, 40, 10, 25, 26, 33]

>>> ## syntax to calculate the simple arithmetic mean

>>> statistics.mean (x)
24.833333333333332
```

Geometric Mean

Geometric mean is the mean of the means and is equal to the nth root of the product (multiplication) among the elements of a series whose mathematical representation is as follows:

$$\overline{g} = \sqrt[n]{x_1 * x_2 * \cdots * x_n} \tag{2.2}$$

or

$$\overline{g} = (x_1 * x_2 * \cdots * x_n)^{\frac{1}{n}} \tag{2.3}$$

where:

$x_i \rightarrow$ is each element of the series; and

$n \rightarrow$ is the number of elements in the series.

The calculation of the geometric mean can be done manually, just remembering the formula of this magnitude. However, in this manual a Python module will be used to perform this calculation.

Calculating the Geometric Mean with Python

```
>>> ## Example:

>>> from scipy import stats as scistats

>>> ## creating any data series

>>> x = [15, 40, 10, 25, 26, 33]

>>> ## syntax to calculate the simple arithmetic mean

>>> scistats.gmean (x)
22.469656044541605
```

Harmonic Mean

When it comes to inversely proportional quantities (for example, cost and quantity), the harmonic meedia is used. That is, it is applied to calculate the average cost of goods bought with a fixed monetary amount, the average speed, etc. Because average cost is equal to C = Pq and average speed is equal to V = dt, that is, Cost is inversely proportional to quantity and velocity is inversely proportional to time. The harmonic mean formula is:

$$\overline{h} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}} \tag{2.4}$$

where:

$x_i \rightarrow$ is each element of the series; and

$n \rightarrow$ is the number of elements in the series.

Just as for the geometric mean just remember the formula to apply it manually. However, once again, a Python module will be used in this manual to perform the harmonic mean calculation.

Calculating the Harmonic Mean with Python

```
>>> ## Example:

>>> from scipy import stats as scistats

>>> ## creating any data series

>>> x = [15, 40, 10, 25, 26, 33]

>>> ## syntax to calculate the simple arithmetic mean

>>> scistats.hmean (x)
19.971292237265782
```

Median

The median of any data series separates the lower half of the upper half. That is, 50% of the series will have values less than or equal to the median and the othres 50% of the series will have values greater than or equal to the median. There are two observations that need to be done. First, the data must be processed orderly (it can be ascending or descending order), that is, one should not work with raw data, that is, without ordering. For example, a series of raw data $\{7, 9, 1, 5, 3\}$ needs to be sorted $\{1, 3, 5, 7, 9\}$ or $\{9, 7, 5, 3, 1\}$. Second, you should check whether the number of terms in the series is even or odd, as there will be a separate calculation formula for each of the situations.
As a last observation, the formulas applied in the calculation of the median do not report the median value, but rather, the position in which the median value is. And possessing these positions returns to the data series to locate the median.

1. if the number of terms in the given series is even the median is the order term given by the formula: $P_{M_d} = \frac{n+1}{2}$.
2. if the number of terms in the given series is odd, the median is the mean simple arithmetic of the order terms given by the formulas: $P_{M_d} = \frac{n}{2}$ and $P_{M_d} = \frac{n}{2} + 1$.

where:

$P_{M_d} \rightarrow$ is the position of the median value in the series; and

$n \rightarrow$ is the number of elements in the series.

Example 2.1: What is the median of the series $1, 3, 5, 7, 9$?[*]. Since the number of terms in the series is odd, only formula $P_{M_d} = \frac{n+1}{2}$. So,

$$P_{M_d} = \frac{5+1}{2} = \frac{6}{2} = 3$$

Therefore, the median value is in the 3rd position, that is, the median is $M_d = 5$.

Example 2.2: What is the median of the series $1, 3, 5, 7, 9, 10$?[†]. Now the number of terms in the series is even, so the two formulas apply: $P_{M_d} = \frac{n}{2}$ e $P_{M_d} = \frac{n}{2} + 1$. Therefore,

$$P_{M_d} = \frac{6}{2} = 3$$

and

[*]Note that the series is already ordered, that is, they are not raw data.
[†]Note that the series is already ordered, that is, they are not raw data.

$$P_{M_d} = \frac{6}{2} + 1 = 3 + 1 = 4$$

Then, the median value will be the simple arithmetic mean of the values that are in the $3rd$ and $4th$ positions and which are, respectively, 5 and 7.

$$\bar{x} = \frac{5+7}{2} = \frac{12}{2} = 6$$

Thus, the median is equal to $M_d = 6$.

However, in Python the values reported for the median syntax are the values corresponding to the data series.

Calculating the Median with Python

```
>>> ## Example:

>>> import statistics

>>> ## creating any data series

>>> a = [1, 3, 5, 7, 9]

>>> ## syntax to calculate the median of a series

>>> statistics.median (a)
5
```

For another series we have:

Calculating the Median with Python

```
>>> ## Example:

>>> import statistics

>>> ## creating any data series

>>> b = [1, 3, 5, 7, 9, 10]

>>> ## syntax to calculate the median of a series

>>> statistics.median (b)
6.0
```

Mode

Mode is the value of the series that occurs the most, that is, the most frequently. However, in a series it may be that there is no repeating term, and so, such series is termed amodal. In turn, if two elements occur more frequently the series is called bimodal and in cases where there are more than two elements that repeat a series has multimodal or polymodal.

Calculating the Mode with Python

```
>>> ## Example:

>>> import statistics

>>> ## creating any data series

>>> w = [1, 2, 3, 4, 4, 4, 5, 6, 7]

>>> ## syntax to calculate the mode of a series

>>> statistics.mode (w)
4
```

Chapter 3

Simple Linear Regression

Simple linear regression analysis studies the linear relationship between two quantitative variables. Being one denominated of dependent variable and the one of independent variable. This analysis is carried out from two different points of view:

1. regression $\rightarrow$ which expresses the form of the linear relationship between the two variables; and
2. correlation $\rightarrow$ that quantifies the strength of this relationship.

This relation is represented by a mathematical model, that is, by an equation which will associate the explained variable with the explanatory variable. The mathematical representation of this association is as follows:

$$Y = \beta_0 + \beta_1 X + \mu \tag{3.1}$$

where:

$Y \rightarrow$ is the explained or dependent variable that will be calculated and therefore is random;

β_0 and $\beta_1 \rightarrow$ are the unknown parameters of the model that will be calculated. When you are working with the population it is said that these are the estimates, however, if you are working with a sample it is said that these are the estimators of the true values;

$X \rightarrow$ is the explanatory or independent variable measured without error, that is, without randomness; and

$\mu \rightarrow$ is the residual random variable in which all other variables that influence the behavior of the dependent variable Y are found and which were not included in the mathematical model. That is, they are influences on the explained variable Y that can not be explained linearly by the behavior of the explicative variable X.

Example 3.1: Table I.1 of the book Basic Econometrics, translation of the 4th edition, by Damodar Gujarati will be used.

Simple Linear Regression with Python

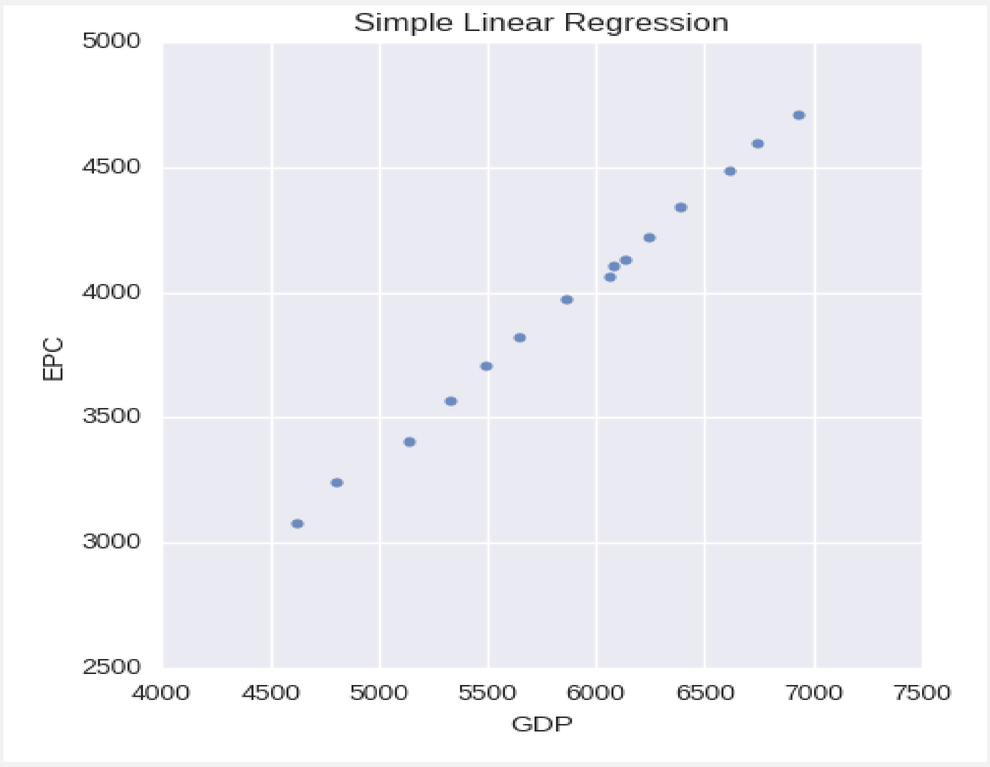
```
>>> ## Examples:

>>> ## creating a data.frame with data from Table I.1

>>> import pandas as pd

>>> pers_con = pd.DataFrame ({'year': (1982, 1983, 1984, 1985, 1986, 1987,
...                                    1988, 1989, 1990, 1991, 1992, 1993,
...                                    1994, 1995, 1996),
...                             'epc': (3081.5, 3240.6, 3407.6, 3566.5,
...                                     3708.7, 3822.3, 3972.7, 4064.6,
...                                     4132.2, 4105.8, 4219.8, 4343.6,
...                                     4486.0, 4595.3, 4714.1),
...                             'gdp': (4620.3, 4803.7, 5140.1, 5323.5,
...                                     5487.7, 5649.5, 5865.2, 6062.0,
...                                     6136.3, 6079.4, 6244.4, 6389.6,
...                                     6610.7, 6742.1, 6928.4)},
...                              columns = ['year', 'epc', 'gdp']
...                           )

>>> ## calling the created data.frame

>>> pers_con
    year     epc     gdp
0   1982  3081.5  4620.3
1   1983  3240.6  4803.7
2   1984  3407.6  5140.1
3   1985  3566.5  5323.5
4   1986  3708.7  5487.7
5   1987  3822.3  5649.5
6   1988  3972.7  5865.2
7   1989  4064.6  6062.0
8   1990  4132.2  6136.3
9   1991  4105.8  6079.4
10  1992  4219.8  6244.4
11  1993  4343.6  6389.6
12  1994  4486.0  6610.7
13  1995  4595.3  6742.1
14  1996  4714.1  6928.4

>>> ## obtaining the descriptive statistics of the data

>>> pers_con.describe ().transpose ()
      count         mean         std     min     25%     50%     75%     max
year     15  1989.000000    4.472136  1982.0  1985.5  1989.0  1992.5  1996.0
epc      15  3964.086667  489.661393  3081.5  3637.6  4064.6  4281.7  4714.1
gdp      15  5872.193333  692.618195  4620.3  5405.6  6062.0  6317.0  6928.4

>>> pers_con.corr ()
          year       epc       gdp
year  1.000000  0.988875  0.987103
epc   0.988875  1.000000  0.999203
gdp   0.987103  0.999203  1.000000

>>> ## estimating the simple regression between epc and gdp

>>> from statsmodels.formula.api import ols

>>> reg = ols ("epc ~ gdp", pers_con).fit()

>>> ## calling the simple linear regression between epc and gdp

>>> print (reg.summary ())
                            OLS Regression Results
```

```
==============================================================================
Dep. Variable:                    epc   R-squared:                       0.998
Model:                            OLS   Adj. R-squared:                  0.998
Method:                 Least Squares   F-statistic:                     8145.
Date:                Fri, 13 Jan 2017   Prob (F-statistic):           1.42e-19
Time:                        21:18:09   Log-Likelihood:                -65.359
No. Observations:                  15   AIC:                             134.7
Df Residuals:                      13   BIC:                             136.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
Intercept   -184.0780     46.262     -3.979      0.002      -284.021   -84.135
gdp            0.7064      0.008     90.247      0.000         0.689     0.723
==============================================================================
Omnibus:                        1.347   Durbin-Watson:                   2.082
Prob(Omnibus):                  0.510   Jarque-Bera (JB):                0.900
Skew:                          -0.572   Prob(JB):                        0.638
Kurtosis:                       2.637   Cond. No.                     5.22e+04
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 5.22e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

Simple Linear Regression with Python

```
>>> import seaborn as sns

>>> import matplotlib.pyplot as plt

>>> ## first without the estimated regression line

>>> sns.lmplot ('gdp', 'epc', pers_con, fit_reg = False)
<seaborn.axisgrid.FacetGrid object at 0x7fc1931d4f50>

>>> ## naming the axes

>>> plt.title ('Simple Linear Regression')
<matplotlib.text.Text object at 0x7fc18d6dbdd0>

>>> plt.xlabel ('GDP')
<matplotlib.text.Text object at 0x7fc18e5001d0>

>>> plt.ylabel ('EPC')
<matplotlib.text.Text object at 0x7fc18d6ab150>

>>> plt.show ()
```

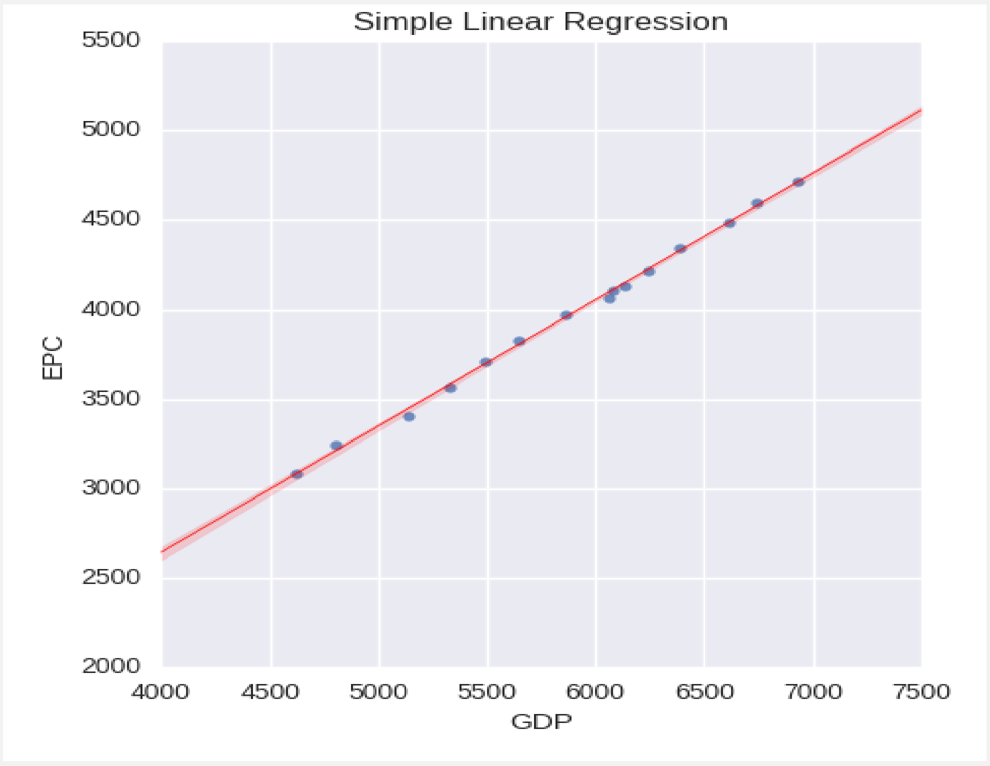
Simple Linear Regression with Python



Simple Linear Regression with Python

```
>>> ## adding the estimated regression line

>>> sns.lmplot ('gdp', 'epc', pers_con, line_kws = {'color': 'red', 'lw': 0.5})
<seaborn.axisgrid.FacetGrid object at 0x7fc1931d4f50>

>>> ## naming the axes

>>> plt.title ('Simple Linear Regression')
<matplotlib.text.Text object at 0x7fc1893cca90>

>>> plt.xlabel ('GDP')
<matplotlib.text.Text object at 0x7fc189e21e50>

>>> plt.ylabel ('EPC')
<matplotlib.text.Text object at 0x7fc189416dd0>

>>> plt.show ()
```

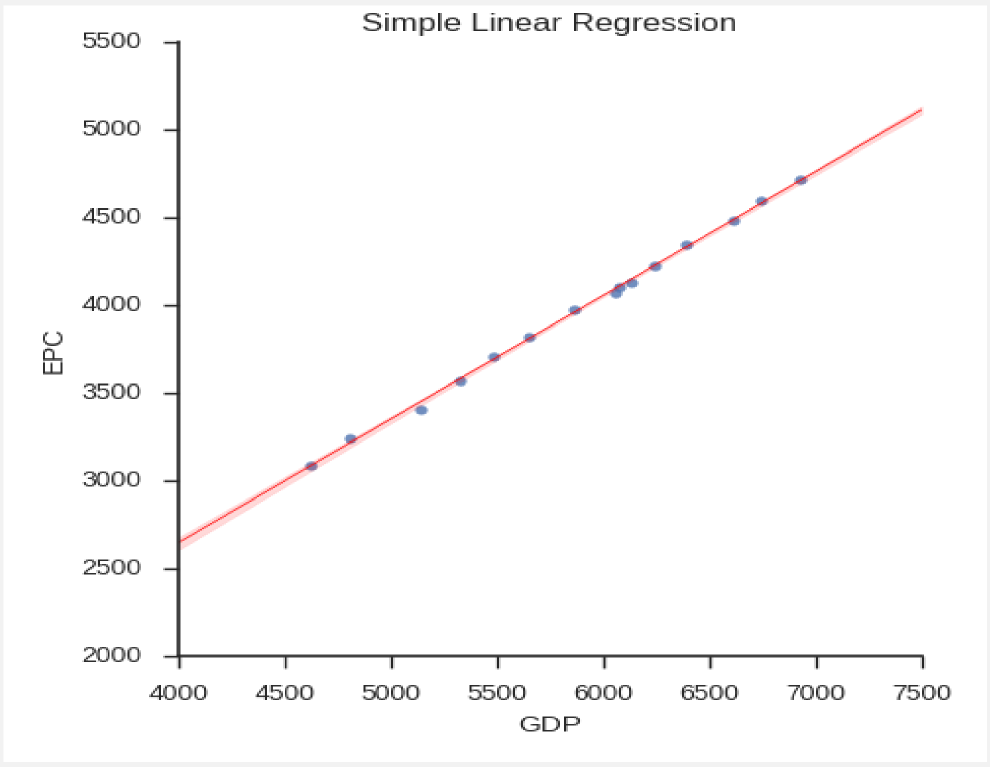
Simple Linear Regression with Python



Simple Linear Regression with Python

```
>>> import seaborn as sns

>>> import matplotlib.pyplot as plt

>>> ## creating another style of scatterplots with seaborn

>>> sns.set_style ('ticks')

>>> sns.lmplot ('gdp', 'epc', pers_con, line_kws = {'color': 'red', 'lw': 0.5})
<seaborn.axisgrid.FacetGrid object at 0x7fc189dffe50>

>>> ## naming the axes

>>> plt.title ('Simple Linear Regression')
<matplotlib.text.Text object at 0x7fc189221bd0>

>>> plt.xlabel ('GDP')
<matplotlib.text.Text object at 0x7fc18940b910>

>>> plt.ylabel ('EPC')
<matplotlib.text.Text object at 0x7fc18935ff90>

>>> plt.show ()
```

Simple Linear Regression with Python


Simple Linear Regression
EPC
GDP
5500
5000
4500
4000
3500
3000
2500
2000
4000
4500
5000
5500
6000
6500
7000
7500