Chapter_06_additional_lectures

June 4, 2022

1 PREREQUISITES - Python basics

This chapter will give you the python basics you need to follow the rest of the course. These are only reminders of what will be useful. That's why I advise you to follow a free quick Python training for your personal projects because if you are totally strangers to it, it can be hard to do your own project.

1.0.1 Part 1: The Python basics

- Data operations: > * Numbers , Strings, Booleans > * Tuples, Lists > * Variable assignment
- Python Structures: > * If/Elif/Else > * FOR > * While
- The Functions

1.0.2 Part 2: Data Management

- Numpy: > * Numpy table creation > * Random with Numpy > * Indexing and slixing
- Pandas: > * Series and dataframe > * Useful pandas functions > * Indexing and slixing
- Matplotlib > * Graphs > * Point clouds > * Useful functions

2 PART 1: The python basics

2.0.1 Data operations

Numbers

[2]:
```
# Addition
1 + 1
```

[2]:
```
2
```

[3]:
```
# Subtraction
5 - 2
```

[3]:
```
3
```

[4]:
```
# Multiplication
5 * 3
```

[4]:
```
15
```

[5]:
```
# Division
15 / 3
```

[5]:
```
5.0
```

[6]:
```
# Power
2**3
```

[6]:
```
8
```

[7]:
```
# Whole part | 5.33 --> 5
16 // 3
```

[7]:
```
5
```

[8]:
```
# Modulo | 16 = 15*3 + 1
16 % 3
```

[8]:
```
1
```

String

[9]:
```
# Simple string
"Hello"
```

[9]:
```
'Hello'
```

[10]:
```
# F-string (first possibility)
"Tesla stock price is {} at date {}".format(515, "01-01-2019")
```

[10]:
```
'Tesla stock price is 515 at date 01-01-2019'
```

[11]:
```
# F-string (second possibility)
f"Tesla stock price is {515} at date {'2019-01-01'}"
```

[11]:
```
'Tesla stock price is 515 at date 2019-01-01'
```

[12]:
```
# Slicing on string
"Hello"[0:3]
```

[12]:
```
'Hel'
```

Booleans / logical operations

[13]:
```
# Boolean type (True)
True
```

[13]:
```
True
```

[14]:
```python
# Boolean type (False)
False
```

[14]:
```
False
```

[15]:
```python
# Equality
5 == 6
```

[15]:
```
False
```

[16]:
```python
# Sup
5 < 6
```

[16]:
```
True
```

[17]:
```python
# Inf
5 > 6
```

[17]:
```
False
```

[18]:
```python
# Not Equal
5 != 6
```

[18]:
```
True
```

[19]:
```python
# And
(1<2) and (5>6)
```

[19]:
```
False
```

[20]:
```python
# Or
(1<2) or (5>6)
```

[20]:
```
True
```

[21]:
```python
# Not
not (1==15)
```

[21]:
```
True
```

Variable assignment

[22]:
```python
x = 5
y = 6
print(x + y)
```

```
11
```

[23]:

```
price = 515
date = "01-01-2019"
information = f"Tesla stock price is {price} at date {date}"

print(price)
print(date)
print(information)
```

```
515
01-01-2019
Tesla stock price is 515 at date 01-01-2019
```

[24]:

```
market_open = True
market_open
```

[24]:

```
True
```

Tuples, lists

[25]:

```
# Tuple
mean_variance_couple = (6,11)
print(mean_variance_couple)
```

```
(6, 11)
```

[26]:

```
# List creation
my_list = [1,2,3,4,5,6,7,8,9,10]
print(my_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

[27]:

```
# Adding value
my_list.append(11)
print(my_list)
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

[28]:

```
# Indexing
my_list[0]
```

[28]:

```
1
```

[29]:

```
# Value range selection
my_list[0:6]
```

[29]:

```
[1, 2, 3, 4, 5, 6]
```

[30]:

```
# Delete a value
del my_list[0]
print(my_list)
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
```

[31]:

```
# Bonus: Nested List
my_list = [[1,15], 3]

# Print the sublist
print(my_list[0])

# Print value 15
my_list[0][1]
```

```
[1, 15]
```

[31]:

```
15
```

Dictionary

[32]:

```
# Initialize a dictionnary
dictio = {}
dictio
```

[32]:

```
{}
```

[33]:

```
# Initialize with values
dictio = {"TSLA Price": 1500}
dictio
```

[33]:

```
{'TSLA Price': 1500}
```

[34]:

```
# Add a value
dictio["GOOG Price"] = 1300
dictio
```

[34]:

```
{'GOOG Price': 1300, 'TSLA Price': 1500}
```

[35]:

```
# Extract a value
dictio["TSLA Price"]
```

[35]:

```
1500
```

Sets

[36]:

```
# Create a set
{1,3,9}
```

[36]:

```
{1, 3, 9}
```

[37]:

```
# View a property
{1,1,1,1,1,1,1,1,3,3,3,9,9,9}
```

[37]:

```
{1, 3, 9}
```

[38]:

```
# Create a list
my_list = ["Finance", "Finance", "Finance"]

# Transform the list into a set
set(my_list)
```

[38]:

```
{'Finance'}
```

[39]:

```
# Add a value
s = {1,3,15}
print(s)

s.add(56)
print(s)
```

```
{1, 3, 15}
{56, 1, 3, 15}
```

2.0.2 Python Structures

If/ Elif/ Else

[40]:

```
# Conditionnal structure IF
if 5<6:
  print("Yes")
```

```
Yes
```

[41]:

```
# Conditionnal structure IF/ ELSE
if 5>6:
  print("Yes")
else:
  print("No")
```

```
No
```

[42]:

```
# Conditionnal structure IF/ ELIF/ ELSE
x = 12

if x>15:
  print("X>15")

elif x>10:
  print("15>x>10")

else:
  print("x<10")
```

```
15>x>10
```

Loop for

[43]:

```
# Loop for with a sequence
sequence = [1,2,3,4,5,6]

for item in sequence:
  print(item)
```

```
1
2
3
4
5
6
```

[44]:

```
# Loop for with a range
for item in range(7):
  print(item)
```

```
0
1
2
3
4
5
6
```

[45]:

```
# Loop in list
seq = [i for i in range(7)]
print(seq)
```

```
[0, 1, 2, 3, 4, 5, 6]
```

Loop While

[46]:

```
# Loop while
i = 1
while i <= 15:
  print(i)
  i = i+1
```

```
1
2
3
4
5
6
7
8
9
10
11
```

```
12
13
14
15
```

2.0.3 Functions

Basics function

[47]:

```
# Basic function
def my_function(param1, param2, param3):
    """
    documentation
    """
    return param1 + param2 + param3
```

[48]:

```
my_function(1,2,3)
```

[48]:

```
6
```

[49]:

```
# default setting
def function_bis(param1, param2 = 3, param3 = 5):
  """
  documentation
  """
  return param1 + param2 + param3
```

[50]:

```
function_bis(1)
```

[50]:

```
9
```

Lambda

[51]:

```
# Creation lambda oject
lambda x: x**2
```

[51]:

```
<function __main__.<lambda>>
```

[52]:

```
# Create a list to apply a lambda function
lis = [1,2,3]

# Use map function to apply lambda function to a list
generator = map(lambda x: x**2,lis)

generator
```

[52]:

```
<map at 0x7f1546f59110>
```

[53]:

```
# Map the list
list(generator)
```

```
[53]: [1, 4, 9]
```

```
[54]: # Create a function wich verify if the number is odd or not
      def f(num):
        return num%2==0

      # Filter take only the values wich return True
      list(filter(f, lis))
```

```
[54]: [2]
```

Local Variable

```
[55]: # variable
      glo = 50
      glo
```

```
[55]: 50
```

```
[56]: # Local variable
      def my_function():
        loc = 60
        print(glo)
```

```
[57]: # Glo can go in the function
      my_function()

      # But loc can not go out this function it is local to the function
      print(loc)
```

```
50
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-57-c4ba15d3d69c> in <module>()
      3
      4 # But loc can not go out this function it is local to the function
----> 5 print(loc)

NameError: name 'loc' is not defined
```

Global Variable

```
[118]: # Global variable

       def my_function():
         global x
         x = 3
```

[119]:

```
# Rune the function and print the global variable
my_function()
print(x)
```

```
3
```

3 Part 2: Data Management

3.0.1 Numpy

Numpy table creation

[120]:

```
# Import of numpy
import numpy as np

# 1D array
arr = np.array([1,2,3])
print(arr)
```

```
[1 2 3]
```

[121]:

```
# Linspace function
np.linspace(0,50,51)
```

[121]:

```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
       13., 14., 15., 16., 17., 18., 19., 20., 21., 22., 23., 24., 25.,
       26., 27., 28., 29., 30., 31., 32., 33., 34., 35., 36., 37., 38.,
       39., 40., 41., 42., 43., 44., 45., 46., 47., 48., 49., 50.])
```

[122]:

```
# Arrange function
np.arange(0,50)
```

[122]:

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49])
```

[123]:

```
# 2D array
arr_bis = np.array([[1,2,3],
                    [5,6,9],
            [7,6,9]])
arr_bis
```

[123]:

```
array([[1, 2, 3],
       [5, 6, 9],
       [7, 6, 9]])
```

[124]:

```
# Null matrix
O = np.zeros([5,5])
O
```

```
[124]: array([[0., 0., 0., 0., 0.],
              [0., 0., 0., 0., 0.],
              [0., 0., 0., 0., 0.],
              [0., 0., 0., 0., 0.],
              [0., 0., 0., 0., 0.]])
```

```
[125]: # Identity matrix
       I = np.identity(5)
       I
```

```
[125]: array([[1., 0., 0., 0., 0.],
              [0., 1., 0., 0., 0.],
              [0., 0., 1., 0., 0.],
              [0., 0., 0., 1., 0.],
              [0., 0., 0., 0., 1.]])
```

```
[126]: # Ones matrix
       One = np.ones([5,5])
       One
```

```
[126]: array([[1., 1., 1., 1., 1.],
              [1., 1., 1., 1., 1.],
              [1., 1., 1., 1., 1.],
              [1., 1., 1., 1., 1.],
              [1., 1., 1., 1., 1.]])
```

```
[127]: # Add two matrix
       O + I
```

```
[127]: array([[1., 0., 0., 0., 0.],
              [0., 1., 0., 0., 0.],
              [0., 0., 1., 0., 0.],
              [0., 0., 0., 1., 0.],
              [0., 0., 0., 0., 1.]])
```

```
[128]: # Soubstract two matrix
       O-I
```

```
[128]: array([[-1.,  0.,  0.,  0.,  0.],
              [ 0., -1.,  0.,  0.,  0.],
              [ 0.,  0., -1.,  0.,  0.],
              [ 0.,  0.,  0., -1.,  0.],
              [ 0.,  0.,  0.,  0., -1.]])
```

```
[129]: # Scalar multiplication
       I * 5
```

```
[129]: array([[5., 0., 0., 0., 0.],
               [0., 5., 0., 0., 0.],
               [0., 0., 5., 0., 0.],
               [0., 0., 0., 5., 0.],
               [0., 0., 0., 0., 5.]])
```

Random

```
[130]: # Random 1d array in the value 0 and 1 (Ideal to simulate a random weight of a␣
        ↪wallet for example)
       ran = np.random.rand(5)
       print(ran)
```

```
[0.03879199 0.31330032 0.8882369  0.84822129 0.70954671]
```

```
[131]: # BONUS: How to find the shape of a array and transorfm 1d array to 2d array
       print(f"Shape is: {np.shape(ran)}")

       # Reshape
       ran = ran.reshape(-1,1)

       # New shape
       print(f"New shape is: {np.shape(ran)}")
```

```
Shape is: (5,)
New shape is: (5, 1)
```

```
[132]: # Random integer value
       arr_int = np.random.randint(100, size=(5,5))
       print(arr_int)
```

```
[[57 69 38 77 11]
 [62 11 90 29 52]
 [93 90 64 11  6]
 [94 68 62 30 80]
 [83 50 74 35 61]]
```

```
[133]: # Normal 1 Dim
       np.random.randn(3)
```

```
[133]: array([-1.25729761,  0.13152329, -0.20864375])
```

```
[134]: # Normal 2 Dim
       np.random.randn(3,3)
```

```
[134]: array([[-0.08786695,  0.75704674, -1.13198515],
               [ 0.69740263,  1.00887717, -0.07221112],
               [-2.0695567 , -1.88590475,  0.22580599]])
```

[135]:

```
# Set the seed
print(np.random.rand(3))
print(np.random.rand(3))

np.random.seed(seed = 32)
print(np.random.rand(3))

np.random.seed(seed = 32)
print(np.random.rand(3))

np.random.seed(seed = 32)
print(np.random.rand(3))
```

```
[0.46310814 0.98478429 0.50113492]
[0.39807245 0.72790532 0.86333097]
[0.85888927 0.37271115 0.55512878]
[0.85888927 0.37271115 0.55512878]
[0.85888927 0.37271115 0.55512878]
```

Indexing Slicing Transformation

[136]:

```
# Choose one value in a matrix
arr_bis = np.array([[1,2,3],
                    [7,1,6],
                    [9,6,3]])

arr_bis[0][0]
```

[136]:

```
1
```

[137]:

```
arr_bis[0,0]
```

[137]:

```
1
```

[138]:

```
# Choose sub matrix in the matrix
arr_bis[0:2,0:3]
```

[138]:

```
array([[1, 2, 3],
       [7, 1, 6]])
```

[139]:

```
# Choose one columns or one row
print(arr_bis[1,:])
print(arr_bis[:,1])
```

```
[7 1 6]
[2 1 6]
```

[140]:

```
# Max
# On the matrix
```

```
print(arr_bis.max())

# By the rows
print(arr_bis.max(axis=1))

# By the columns
print(arr_bis.max(axis=0))
```

```
9
[3 7 9]
[9 6 6]
```

[141]:
```
# Min
# On the matrix
print(arr_bis.min())

# By the rows
print(arr_bis.min(axis=1))

# By the columns
print(arr_bis.min(axis=0))
```

```
1
[1 1 3]
[1 1 3]
```

[142]:
```
# Mean
# On the matrix
print(arr_bis.mean())

# By the rows
print(arr_bis.mean(axis=1))

# By the columns
print(arr_bis.mean(axis=0))
```

```
4.222222222222222
[2.         4.66666667 6.        ]
[5.66666667 3.         4.        ]
```

[143]:
```
# Std
# On the matrix
print(arr_bis.std())

# By the rows
print(arr_bis.std(axis=1))

# By the columns
print(arr_bis.std(axis=0))
```

```
2.698879511442471
[0.81649658 2.62466929 2.44948974]
[3.39934634 2.1602469  1.41421356]
```

[144]:

```python
# Other way available
np.max(arr_bis, axis=1)
```

[144]:

```
array([3, 7, 9])
```

[145]:

```python
# Log function
np.log(arr_bis)
```

[145]:

```
array([[0.        , 0.69314718, 1.09861229],
       [1.94591015, 0.        , 1.79175947],
       [2.19722458, 1.79175947, 1.09861229]])
```

[146]:

```python
# Exponential function
np.exp(arr_bis)
```

[146]:

```
array([[2.71828183e+00, 7.38905610e+00, 2.00855369e+01],
       [1.09663316e+03, 2.71828183e+00, 4.03428793e+02],
       [8.10308393e+03, 4.03428793e+02, 2.00855369e+01]])
```

[147]:

```python
# Squared root function
np.sqrt(arr_bis)
```

[147]:

```
array([[1.        , 1.41421356, 1.73205081],
       [2.64575131, 1.        , 2.44948974],
       [3.        , 2.44948974, 1.73205081]])
```

[148]:

```python
# Concatenate
# Params: tuple of arrays and the axis of the concatenation
# Concat 1 dimension
arr1 = arr_bis[:,1]
arr2 = arr_bis[:,2]

print(f"ARR1 {arr1}")

print(f"ARR2 {arr2}")

print(np.concatenate((arr1, arr2), axis=0))
```

```
ARR1 [2 1 6]
ARR2 [3 6 3]
[2 1 6 3 6 3]
```

[149]:

```python
# Concat 2 dimension
arr1 = arr_bis[:,1].reshape(-1,1)
arr2 = arr_bis[:,2].reshape(-1,1)
```

```python

print(f"ARR1 {arr1}")

print(f"ARR2 {arr2}")


print(np.concatenate((arr1, arr2), axis=0))
```

```
ARR1 [[2]
 [1]
 [6]]
ARR2 [[3]
 [6]
 [3]]
[[2]
 [1]
 [6]
 [3]
 [6]
 [3]]
```

[150]:

```python
# Concat 2 dimension
arr1 = arr_bis[:,1].reshape(-1,1)
arr2 = arr_bis[:,2].reshape(-1,1)

print(f"ARR1 {arr1}")

print(f"ARR2 {arr2}")


print(np.concatenate((arr1, arr2), axis=1))
```

```
ARR1 [[2]
 [1]
 [6]]
ARR2 [[3]
 [6]
 [3]]
[[2 3]
 [1 6]
 [6 3]]
```

3.0.2 Pandas

[151]:

```python
import pandas as pds
```

Series and dataframes **Series**

```
[152]: labels = ["label 1", "label 2", "label 3"]
       list_values = [1,2,3]
       arr_values = np.array([1,2,3])
```

```
[153]: # Create a serie from a list
       pds.Series(list_values, index=labels)
```

```
[153]: label 1    1
       label 2    2
       label 3    3
       dtype: int64
```

```
[154]: # Create a serie from a array
       pds.Series(arr_values, index=labels)
```

```
[154]: label 1    1
       label 2    2
       label 3    3
       dtype: int64
```

dataframes

```
[155]: list_columns = ["col_1", "col_2", "col_3"]
       list_index = ["row_1", "row_2", "row_3"]
       arr_bis = np.array([[1,2,3],
                           [7,1,6],
                           [9,6,3]])
```

```
[156]: # Create a dataframe from a array
       pds.DataFrame(arr_bis)
```

```
[156]:    0  1  2
       0  1  2  3
       1  7  1  6
       2  9  6  3
```

```
[157]: # Specify rows/columns
       pds.DataFrame(arr_bis, columns = list_columns, index = list_index)
```

```
[157]:        col_1  col_2  col_3
       row_1      1      2      3
       row_2      7      1      6
       row_3      9      6      3
```

Cleaning and selection

```
[158]: # Import of csv
       assets = pds.read_csv("/content/assets.csv", parse_dates=True, index_col="time")
       assets
```

```
[158]:             Open MSFT  High MSFT  …  Close INTEL  Volume INTEL
       time                              …
       2020-03-18      138.0   146.0000  …        29.20      192337.0
       2020-03-17      140.0   147.4998  …        32.40      273967.0
       2020-03-16      140.0   149.3500  …        29.01       97928.0
       2020-03-13      147.5   161.9100  …        33.00      126388.0
       2020-03-12      145.3   153.4700  …        32.43      145336.0
       …               …        …  …          …           …
       2000-07-04        NaN        NaN  …          NaN           NaN
       2000-05-29        NaN        NaN  …          NaN           NaN
       2000-04-21        NaN        NaN  …          NaN           NaN
       2000-02-21        NaN        NaN  …          NaN           NaN
       2000-01-17        NaN        NaN  …          NaN           NaN

       [5732 rows x 80 columns]
```

```
[159]: # Ordonning
       assets = assets.sort_index(ascending=True)
       assets
```

```
[159]:             Open MSFT  High MSFT  …  Close INTEL  Volume INTEL
       time                              …
       2000-01-03        NaN        NaN  …          NaN           NaN
       2000-01-04        NaN        NaN  …          NaN           NaN
       2000-01-05        NaN        NaN  …          NaN           NaN
       2000-01-06        NaN        NaN  …          NaN           NaN
       2000-01-07        NaN        NaN  …          NaN           NaN
       …               …        …  …          …           …
       2020-03-15        NaN        NaN  …          NaN           NaN
       2020-03-16      140.0   149.3500  …        29.01       97928.0
       2020-03-17      140.0   147.4998  …        32.40      273967.0
       2020-03-18      138.0   146.0000  …        29.20      192337.0
       2020-03-19        NaN        NaN  …          NaN           NaN

       [5732 rows x 80 columns]
```

```
[160]: # Select a column
       assets = assets[["Close DJI30", "Close CAC40", "Close SP500"]]
       assets
```

```
[160]:             Close DJI30  Close CAC40  Close SP500
       time
       2000-01-03   11357.5098    5917.3701    1455.2200
       2000-01-04   10997.9297    5672.0200    1399.4200
       2000-01-05   11122.6504    5479.7002    1402.1100
       2000-01-06   11253.2598    5450.1099    1403.4500
       2000-01-07   11522.5596    5539.6099    1441.4700
```

```
…                  …          …          …
2020-03-15         NaN        NaN        NaN
2020-03-16  20188.5195  3881.4600  2426.6181
2020-03-17  21237.3809  3991.7800  2436.5000
2020-03-18  19898.9199  3754.8401  2398.1001
2020-03-19         NaN        NaN        NaN

[5732 rows x 3 columns]
```

[161]:

```
# Reset index
assets.reset_index(drop=True)
```

[161]:

```
      Close DJI30  Close CAC40  Close SP500
0      11357.5098    5917.3701    1455.2200
1      10997.9297    5672.0200    1399.4200
2      11122.6504    5479.7002    1402.1100
3      11253.2598    5450.1099    1403.4500
4      11522.5596    5539.6099    1441.4700
…             …          …          …
5727          NaN          NaN          NaN
5728   20188.5195    3881.4600    2426.6181
5729   21237.3809    3991.7800    2436.5000
5730   19898.9199    3754.8401    2398.1001
5731          NaN          NaN          NaN

[5732 rows x 3 columns]
```

[162]:

```
# Missing values NaN
assets = assets.dropna()
assets
```

[162]:

```
            Close DJI30  Close CAC40  Close SP500
time
2000-01-03   11357.5098    5917.3701    1455.2200
2000-01-04   10997.9297    5672.0200    1399.4200
2000-01-05   11122.6504    5479.7002    1402.1100
2000-01-06   11253.2598    5450.1099    1403.4500
2000-01-07   11522.5596    5539.6099    1441.4700
…                  …          …          …
2020-03-12   21200.6191    4044.2600    2420.8827
2020-03-13   23185.6191    4118.3599    2418.0191
2020-03-16   20188.5195    3881.4600    2426.6181
2020-03-17   21237.3809    3991.7800    2436.5000
2020-03-18   19898.9199    3754.8401    2398.1001

[5047 rows x 3 columns]
```

[163]:

```
# Rolling
assets["SMA15"] = assets["Close SP500"].rolling(15).mean()
assets
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

[163]:

```
            Close DJI30  Close CAC40  Close SP500        SMA15
time
2000-01-03   11357.5098    5917.3701    1455.2200          NaN
2000-01-04   10997.9297    5672.0200    1399.4200          NaN
2000-01-05   11122.6504    5479.7002    1402.1100          NaN
2000-01-06   11253.2598    5450.1099    1403.4500          NaN
2000-01-07   11522.5596    5539.6099    1441.4700          NaN
…                     …          …           …          …
2020-03-12   21200.6191    4044.2600    2420.8827  2445.100287
2020-03-13   23185.6191    4118.3599    2418.0191  2444.029520
2020-03-16   20188.5195    3881.4600    2426.6181  2443.003900
2020-03-17   21237.3809    3991.7800    2436.5000  2442.964567
2020-03-18   19898.9199    3754.8401    2398.1001  2440.551540

[5047 rows x 4 columns]
```

[164]:

```
# Shift
assets["Close SP500"].shift(1)
```

[164]:

```
time
2000-01-03          NaN
2000-01-04    1455.2200
2000-01-05    1399.4200
2000-01-06    1402.1100
2000-01-07    1403.4500
                …
2020-03-12    2423.1280
2020-03-13    2420.8827
2020-03-16    2418.0191
2020-03-17    2426.6181
2020-03-18    2436.5000
Name: Close SP500, Length: 5047, dtype: float64
```

[165]:

```
# Groupby by mean
arr = [[1,10],
       [1,10],
       [3,15]]

df = pds.DataFrame(arr, columns=["num", "val"])

df.groupby(by="num").mean()
```

[165]:

```
     val
num
1     10
3     15
```

[166]:

```
# Groupby by sum
df.groupby(by="num").sum()
```

[166]:

```
     val
num
1     20
3     15
```

[167]:

```
# Groupby by std
df.groupby(by="num").std()
```

[167]:

```
     val
num
1    0.0
3    NaN
```

Iloc & Loc

[168]:

```
# Slicing
assets.iloc[0:1500,0:10]
```

[168]:

```
            Close DJI30  Close CAC40  Close SP500        SMA15
time
2000-01-03   11357.5098    5917.3701    1455.2200          NaN
2000-01-04   10997.9297    5672.0200    1399.4200          NaN
2000-01-05   11122.6504    5479.7002    1402.1100          NaN
2000-01-06   11253.2598    5450.1099    1403.4500          NaN
2000-01-07   11522.5596    5539.6099    1441.4700          NaN
…                     …          …          …          …
2005-12-13   10823.7197    4693.3999    1267.4301  1261.032680
2005-12-14   10883.5098    4674.8501    1272.7400  1261.800013
2005-12-15   10881.6699    4673.1401    1270.9399  1262.155340
2005-12-16   10875.5898    4704.4102    1267.3199  1262.093333
2005-12-19   10836.5303    4694.8599    1259.9200  1262.257333

[1500 rows x 4 columns]
```

```
[169]: # Conditonal by dates
       assets.loc["2010-01-01":"2015-01-01",:]
```

```
[169]:             Close DJI30  Close CAC40  Close SP500        SMA15
       time
       2010-01-04   10583.9600    4013.9700   1132.9900  1116.254013
       2010-01-05   10572.0195    4012.9099   1136.5200  1118.261347
       2010-01-06   10573.6797    4017.6699   1137.1400  1119.796680
       2010-01-07   10606.8604    4024.8000   1141.6899  1122.047333
       2010-01-08   10618.1904    4045.1399   1144.9800  1124.433993
       …                   …          …           …           …
       2014-12-23   18024.1699    4314.9702   2082.1699  2044.893340
       2014-12-24   18030.2109    4295.8501   2081.8799  2045.396660
       2014-12-29   18038.2305    4317.9302   2090.5701  2046.640007
       2014-12-30   17983.0703    4245.5400   2080.3501  2046.972007
       2014-12-31   17823.0703    4272.7500   2058.8999  2046.877993

       [1245 rows x 4 columns]
```

```
[170]: # Conditonal by values
       assets.loc[assets["Close DJI30"] > 15000]
```

```
[170]:             Close DJI30  Close CAC40  Close SP500        SMA15
       time
       2013-05-07   15056.2002    3921.3201   1625.9600  1583.837320
       2013-05-08   15105.1201    3956.2800   1632.6899  1587.711987
       2013-05-09   15082.6201    3928.5801   1626.6700  1592.689320
       2013-05-10   15118.4902    3953.8301   1633.7000  1598.828653
       2013-05-13   15091.6797    3945.2000   1633.7700  1604.063320
       …                   …          …           …           …
       2020-03-12   21200.6191    4044.2600   2420.8827  2445.100287
       2020-03-13   23185.6191    4118.3599   2418.0191  2444.029520
       2020-03-16   20188.5195    3881.4600   2426.6181  2443.003900
       2020-03-17   21237.3809    3991.7800   2436.5000  2442.964567
       2020-03-18   19898.9199    3754.8401   2398.1001  2440.551540

       [1687 rows x 4 columns]
```

```
[171]: # Concat some dataframes

       df1 = assets.iloc[:,:3]
       df2 = assets.iloc[:,3:]

       pds.concat((df1,df2), axis=1)
```

```
[171]:             Close DJI30  Close CAC40  Close SP500        SMA15
       time
```

```
2000-01-03   11357.5098    5917.3701    1455.2200          NaN
2000-01-04   10997.9297    5672.0200    1399.4200          NaN
2000-01-05   11122.6504    5479.7002    1402.1100          NaN
2000-01-06   11253.2598    5450.1099    1403.4500          NaN
2000-01-07   11522.5596    5539.6099    1441.4700          NaN
…                  …          …           …           …
2020-03-12   21200.6191    4044.2600    2420.8827  2445.100287
2020-03-13   23185.6191    4118.3599    2418.0191  2444.029520
2020-03-16   20188.5195    3881.4600    2426.6181  2443.003900
2020-03-17   21237.3809    3991.7800    2436.5000  2442.964567
2020-03-18   19898.9199    3754.8401    2398.1001  2440.551540

[5047 rows x 4 columns]
```

[172]:

```
# Concat some dataframes

df1 = assets.iloc[1500:,:]
df2 = assets.iloc[:1500,:]

pds.concat((df1,df2), axis=0)
```

[172]:

```
            Close DJI30  Close CAC40  Close SP500        SMA15
time
2005-12-20   10805.5498    4703.4800    1259.6200  1262.400000
2005-12-21   10833.7305    4752.4102    1262.7900  1263.287333
2005-12-22   10889.4404    4751.9600    1268.1200  1263.517333
2005-12-23   10883.2695    4757.7402    1268.6600  1263.756000
2005-12-27   10777.7695    4769.3799    1256.5400  1263.386000
…                  …          …           …           …
2005-12-13   10823.7197    4693.3999    1267.4301  1261.032680
2005-12-14   10883.5098    4674.8501    1272.7400  1261.800013
2005-12-15   10881.6699    4673.1401    1270.9399  1262.155340
2005-12-16   10875.5898    4704.4102    1267.3199  1262.093333
2005-12-19   10836.5303    4694.8599    1259.9200  1262.257333

[5047 rows x 4 columns]
```

3.0.3 Matplotlib

[173]:

```
import matplotlib.pyplot as plt
```

[174]:

```
import numpy as np
```

[175]:

```
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib import cycler
```

```
colors = cycler('color',
                ['#669FEE', '#66EE91', '#9988DD',
                 '#EECC55', '#88BB44', '#FFBBBB'])
plt.rc('figure', facecolor='#313233')
plt.rc('axes', facecolor="#313233", edgecolor='none',
       axisbelow=True, grid=True, prop_cycle=colors,
       labelcolor='gray')
plt.rc('grid', color='474A4A', linestyle='solid')
plt.rc('xtick', color='gray')
plt.rc('ytick', direction='out', color='gray')
plt.rc('legend', facecolor="#313233", edgecolor="#313233")
plt.rc("text", color="#C9C9C9")
plt.rc('figure', facecolor='#313233')
```

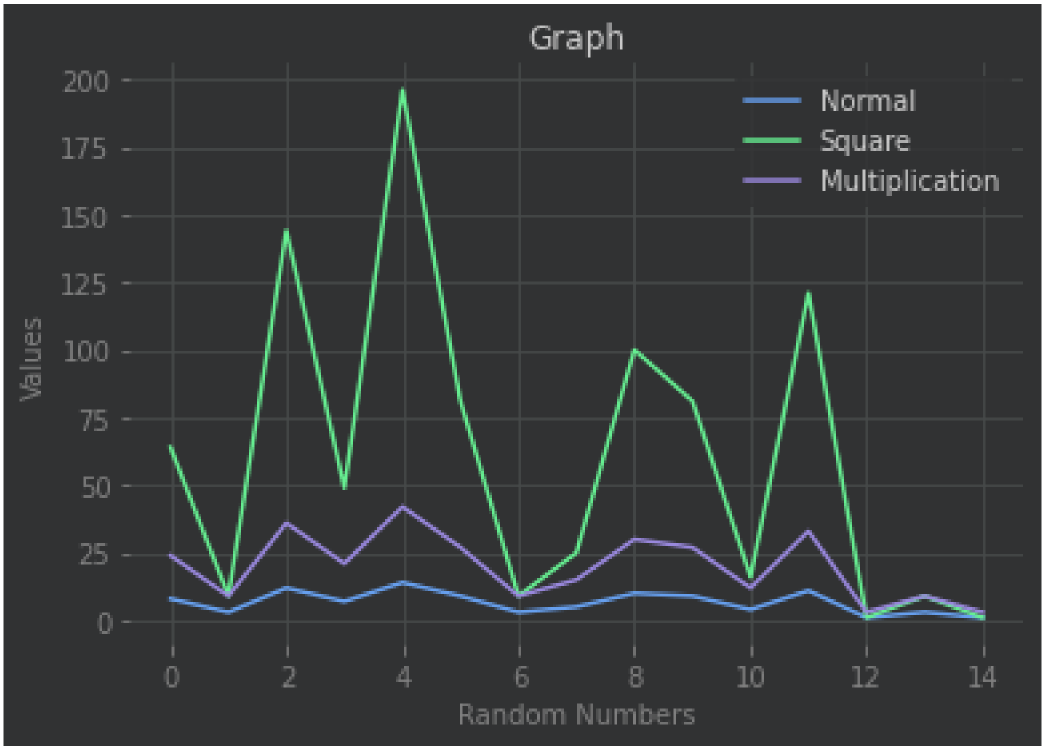
Graphs

[176]:
```
# data simulation
arr = np.random.randint(0,15,size=(15,))

# square of arr
square = arr**2

# Mutltiplication of arr
mut = arr*3

print(arr)
print(square)
print(mut)
```

```
[ 8  3 12  7 14  9  3  5 10  9  4 11  1  3  1]
[ 64   9 144  49 196  81   9  25 100  81  16 121   1   9   1]
[24  9 36 21 42 27  9 15 30 27 12 33  3  9  3]
```

[177]:
```
# Plot the data
plt.plot(arr)
plt.plot(square)
plt.plot(mut)

# Change the xlabel
plt.xlabel("Random Numbers")

# Change the ylabel
plt.ylabel("Values")

# Change the title
plt.title("Graph")

# Put a legend
```

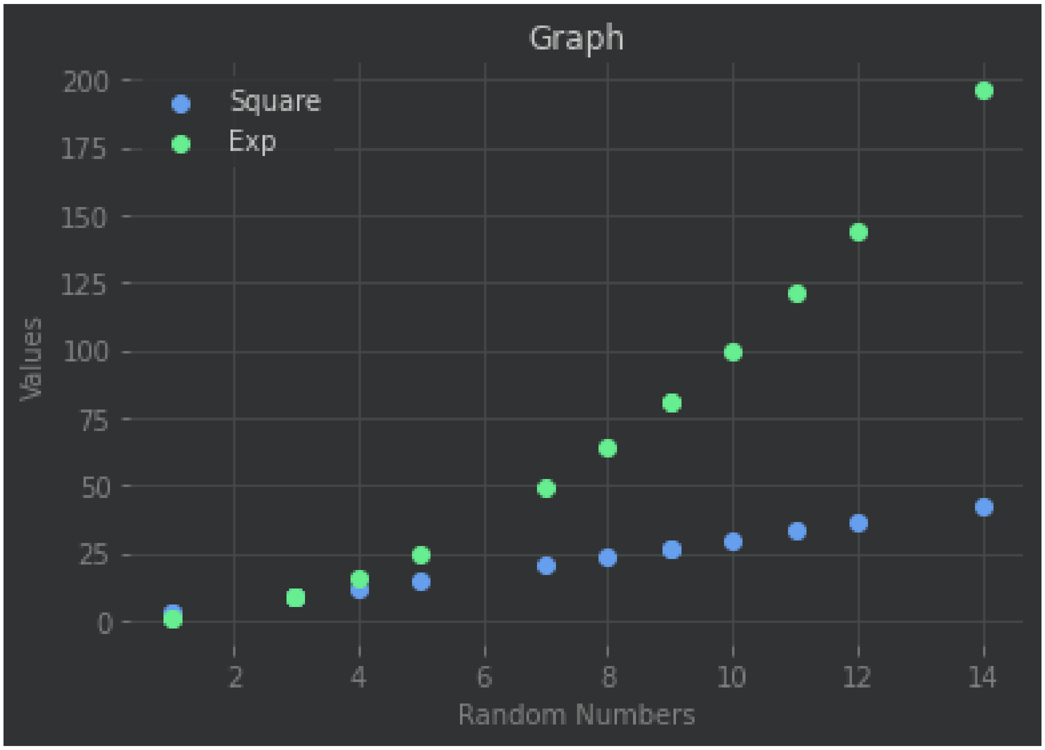
```
plt.legend(["Normal", "Square", "Multiplication"])

# Show the chart
plt.show()
```



Scatter

[178]:
```
# Point clouds
plt.scatter(arr, mut)
plt.scatter(arr, square)

# Change the ylabel
plt.xlabel("Random Numbers")

# Change the title
plt.ylabel("Values")

# Change the title
plt.title("Graph")

# Put a legend
plt.legend(["Square", "Exp"])

# Show the plot
```

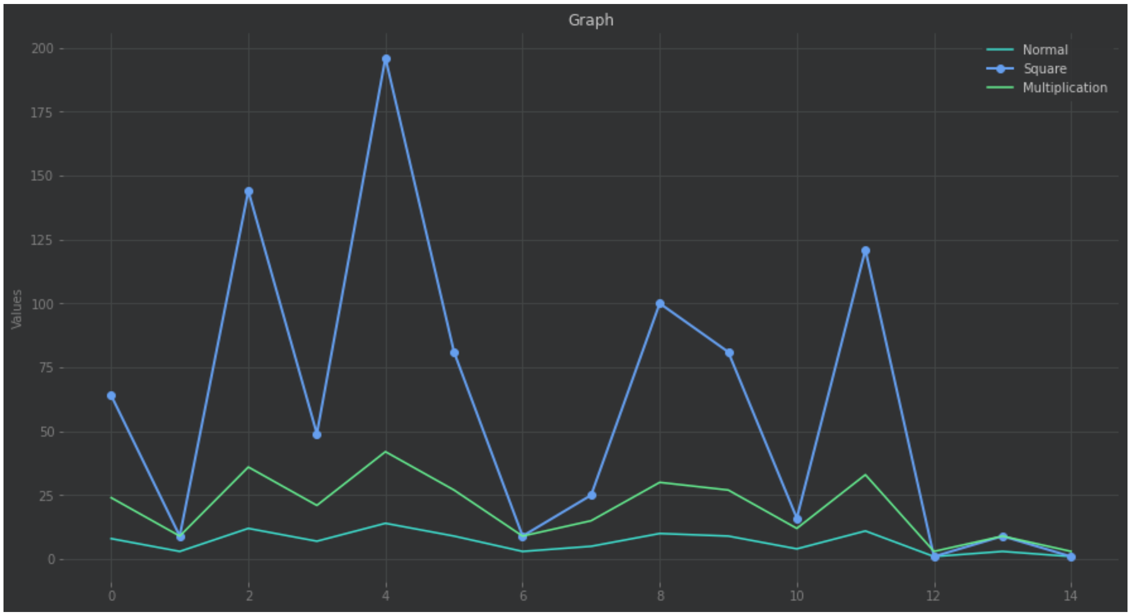
```
plt.show()
```



Tools

[179]:
```
# Adapt the size
plt.figure(figsize=(15,8))

# Plot the line
plt.plot(arr, alpha=1, color="turquoise")

# Plot the line
plt.plot(square, "-o", linewidth=2)

# Plot the line
plt.plot(mut)

# Change the ylabel
plt.ylabel("Values")

# Change the title
plt.title("Graph")

# Put a legend
plt.legend(["Normal", "Square", "Multiplication"])
```

```python
# show the plot
plt.show()
```


Graph
Values
Normal
Square
Multiplication