NUMPY & PANDAS DOCUMENTATION

Created by:

Srinidhi Devan

I. NUMPY:

- Python's package to do math and is more advanced than +/-/* etc.,
- Special functions: Cosine, Exponential, Square Root etc., are included.
- It is also useful in generating samples/arrays from many types of random variables.
- It also has powerful data types to define vectors, matrices and tensors.

```
#Numpy (Numerical Python) is imported in the notebook using:
import numpy as np
```

A. NumPy Basics:

Python Command	Purpose	Illustration
np.array	**To create a 1-D/2-D array**	#CREATING A 1-D NUMPY ARRAY
	Code-->	**list_1 = [101,102,103]**
		array_1 = np.array(list_1)
	Output-->	[101 102 103]
		#CREATING A 2-D NUMPY ARRAY
	Code-->	**list_2 = [201,202,203]**
		array_2 = np.array([list_1,list_2])
	Output-->	[[101 102 103]
		[201 202 203]]
np.zeros	**To create an array (1-D/2-D) of zero**	#CREATING A 1-D ARRAY OF ZEROS
	Code-->	**zero_array_1D = np.zeros(5)**
	Output-->	[0. 0. 0. 0. 0.]
		#CREATING A 2-D ARRAY OF ZEROS
	Code-->	**zero_array_2D = np.zeros([5,5])**
	Output-->	[[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.]]
np.ones	**To create an array (1-D/2-D) of ones**	#CREATING A 1-D ARRAY OF ZEROS
	Code-->	**ones_array_1D = np.ones(5)**
	Output-->	[1. 1. 1. 1. 1.]
		#CREATING A 1-D ARRAY OF ZEROS
	Code-->	**ones_array_2D = np.ones([5,5])**
	Output-->	[[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.]]
np.eye	**To create an identity matrix**	**identity_array = np.eye(3)**
	Output-->	[[1. 0. 0.]
		[0. 1. 0.]
		[0. 0. 1.]]
np.arange	**With defined intervals, it contains evenly spaced values**	**arange_array = np.arange(0,50,2)**
	Output-->	[0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48]

B. Scalar Operations on NumPy:

Python Command	Purpose	Illustration
*	**Multiplying two scalars in NumPy**	
	Code-->	arr_1 = np.array([[1,2,3],[5,6,7]])
		arr_2 = arr_1 * arr_1
	Output-->	[[1 4 9] [25 36 49]]
**	**Exponential operation in NumPy**	
	Code-->	arr_3 = arr_1**3
	Output-->	[[1 8 27] [125 216 343]]
–	**Subtraction of array elements**	
	Code-->	arr_4 = arr_3 - arr_1
	Output-->	[[0 6 24] [120 210 336]]
/	**Scalar Division of array elements**	
	Code-->	arr_5 = 1/arr_1
	Output-->	[[1. 0.5 0.33333333] [0.2 0.16666667 0.14285714]]

C. NumPy Array Indexes:

Python Command	Purpose	Illustration
	Consider an array	
1-D ARRAY	Code-->	arr = np.arange(100,160,2)
	Output-->	[100 102 104 106 108 110 112 114 116 118 120 122 124 126 128 130 132 134 136 138 140 142 144 146 148 150 152 154 156 158]
[index_number]	To access the individual array elements	
	Code-->	print("First Element:",arr[0])
		print("Fifth Elemnet:",arr[4])
	Output-->	First Element: 100
		Fifth Element: 108
[Start:Stop:Step]	Slicing the array indexes	
	Code-->	arr[0:5:1]
	Output-->	[100, 102, 104, 106, 108]
array[index_number]=new_number	Update the array using slices	
	Code-->	arr[0:5] = 0
	Output-->	[0 0 0 0 0 110 112 114 116 118 120 122 124 126 128 130 132 134 136 138 140 142 144 146 148 150 152 154 156 158]

Python Command	Purpose	Illustration
	Consider an array	
2-D ARRAY	Code-->	arr_2d = np.array([[1,2,3],[4,5,6],[7,8,9]])
	Output-->	[[1 2 3]
		[4 5 6]
		[7 8 9]]
[index_number]	To access different rows	
	Code-->	print(arr_2d[0])
	Output-->	[1 2 3]
	Code-->	print(arr_2d[1])
	Output-->	[4 5 6]
array[row][column]	To access elements of an array	
	Code-->	print(arr_2d[0][1])
	Output-->	2

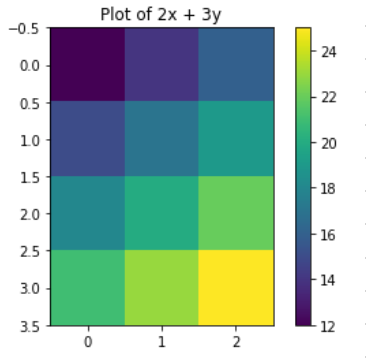
D. Functions of NumPy Array:

Python Command	Purpose	Illustration
np.arange	**With defined intervals, it contains evenly spaced values**	
	Code-->	A = np.arange(15)
	Output-->	[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]
np.sqrt()	**To find the square root of any number**	
	Code-->	D = np.sqrt(A)
	Output-->	[0. 1. 1.41421356 1.73205081 2. 2.23606798 2.44948974 2.64575131 2.82842712 3. 3.16227766 3.31662479 3.46410162 3.60555128 3.74165739]
np.exp()	**To find the Euler's number**	
	Code-->	E = np.exp(A)
	Output-->	[1.00000000e+00 2.71828183e+00 7.38905610e+00 2.00855369e+01 5.45981500e+01 1.48413159e+02 4.03428793e+02 1.09663316e+03 2.98095799e+03 8.10308393e+03 2.20264658e+04 5.98741417e+04 1.62754791e+05 4.42413392e+05 1.20260428e+06]
np.random	**To generate random numbers**	
	Code-->	F = np.random.rand(5)
	Output-->	[0.93222504 0.74206534 0.02859682 0.52828864 0.89512233]
np.add()	**To add two numbers/array**	
	Code-->	G = np.add(A,D)
	Output-->	[0. 2. 3.41421356 4.73205081 6. 7.23606798 8.44948974 9.64575131 10.82842712 12. 13.16227766 14.31662479 15.46410162 16.60555128 17.74165739]
np.maximum()	**To find the maximum of the numbers**	
	Code-->	I = np.array([1,5,7,10])
		J = np.array([0,6,8,9])
		K = np.maximum(I,J)
	Output-->	[1 6 8 10]

E. Statistical Processing & Sketching Graph:

Matplotlib is used for Visualization purposes in Python.

```
#Matplotlib is imported in the notebook using:
import matplotlib.pyplot as plt
```

Python Command	Purpose	Illustration
np.arange	With defined intervals, it contains evenly spaced values	
	Code-->	x = np.arange(3)
	Output-->	[0 1 2]
	Code-->	y = np.arange(4,8)
	Output-->	[4 5 6 7]
np.meshgrid()	Facilitates in creating a grid in rectangular form	
	Code-->	x2,y2 = np.meshgrid(x,y)
	Output of x2-->	x2: [[0 1 2] [0 1 2] [0 1 2] [0 1 2]]
	Output of y2-->	y2: [[4 4 4] [5 5 5] [6 6 6] [7 7 7]]
z = m*x +n*y	To create a linear function	
	Code-->	z = 2*x2 + 3*y2
	Output-->	[[12 14 16] [15 17 19] [18 20 22] [21 23 25]]
plt.imshow()	To plot linear function heatmap	
	Code-->	plt.imshow(z) plt.title('Plot of 2x + 3y') plt.colorbar()
	Output-->	

F. Conditional Clauses & Boolean Operations:

- We impose certain conditions to get the desired output.
- Disadvantage: The statements gets longer with the increase in the number of variables and arrays.
- To tackle this problem, we use 'np.where()' function.

Python Command	Purpose	Illustration
np.array()	**To create a 1-D/2-D array**	
	Code-->	x = np.array([100,400,-50,-40])
		y = np.array([10,15,20,25])
conditional statement	**To impose a condition to get the desired output**	
	Code-->	condition = np.array([True,True,False,False])
		z = [a if cond else b for a,cond,b in zip(x,condition,y)]
	Output-->	[100, 400, 20, 25]
np.where(condition, value if true, value if false)	**To locate the conditional statement and fill the respective values**	
	Code-->	z2 = np.where(condition,x,y)
	Output-->	[100 400 20 25]
.sum()	**To get the sum of an array**	
	Code-->	x.sum()
	Output-->	410
.mean()	**To get the average of an array**	
	Code-->	x.mean()
	Output-->	102.5
.std()	**To get the standard deviation**	
	Code-->	x.std()
	Output-->	181.7106216
.var()	**To get the variance**	
	Code-->	x.var()
	Output-->	33018.75
.unique()	**To get the unique elements from an array**	
	Code-->	arr = np.array(['liquid','liquid','liquid','gas','gas','solid','solid'])
		np.unique(arr)
	Output-->	['gas' 'liquid' 'solid']
.in1d()	**To check whether list of elements are there in the array or not**	
	Code-->	np.in1d(['solid','liquid','plasma'],arr)
	Output-->	[True, True, False]

Additional Note:

Python Command	Purpose
.reshape()	To reshape a matrix
np.matmul	To add 2 matrices
np.unique()	To find the unique value(s) in an array
np.random.rand	To generate random samples from Uniform Distribution
np.random.randn	To generate random samples from Standard Normal Distribution
np.save	To save a single matrix
np.savez	Zipped file to save many matrices
np.load	To load files

II. PANDAS:

- It is a high-level data manipulation tool.
- It is built on NumPy library and extends the functionality of it.
- It has 2 main components: Series & DataFrame.

```
#Pandas is imported in the notebook using:
import pandas as pd
```

A. Pandas Basics- SERIES:

Python Command	Purpose	Illustration
pd.Series()	1-D labelled array and can hold any type of data	
	Code-->	s1 = Series([5,10,15,20])
	Output-->	0 5 1 10 2 15 3 20
np.array()	To create a 1-D/2-D array	
	Code-->	rev_array = np.array([400,300,200,100])
	Output-->	[400 300 200 100]
pd.Series()	To create series from NumPy	
	Code-->	rev_series = Series(rev_array)
	Output-->	0 400 1 300 2 200 3 100
pd.Series(array,index)	To create series with custom indexes	
	Code-->	revenue = Series(rev_array,index=['Uber','Ola','Lyft','Gojek'])
	Output-->	Uber 400 Ola 300 Lyft 200 Gojek 100
series[series>condition]	To filter a series based on condition(s)	
	Code-->	revenue[revenue>250]
	Output-->	Uber 400 Ola 300
series + series	To add 2 series	
	Code-->	revenue+revenue
	Output-->	Uber 800 Ola 600 Lyft 400 Gojek 200

B. Pandas Basics- DATAFRAME:

- 2-D data structure with three principal components - Rows, Columns & Values.
- Similar to a table in excel sheets; Rows & Columns are labelled.
- Size of a dataframe can be changed anytime.

Python Command	Purpose	Illustration
pd.read_clipboard()	**To create a DataFrame from ClipBoard**	
	Code-->	age_df = pd.read_clipboard()
	Output-->	First name Last name Age 0 Tinu Elejogun 14 1 Blaszczyk Kostrzewski 25 2 Lily McGarrett 18 3 Olatunkbo Chijiaku 22 4 Adrienne Anthoula 22 5 Axelia Athanasios 22 6 Jon-Kabat Zinn 22 7 Thabang Mosoa 15 8 Kgaogelo Mosoa 11
df['Column_Name']	**To access one or more column(s)**	
	Code-->	age_df['First name']
	Output-->	0 Tinu 1 Blaszczyk 2 Lily 3 Olatunkbo 4 Adrienne 5 Axelia 6 Jon-Kabat 7 Thabang 8 Kgaogelo
.head()	**To view the first few rows of the data**	
	Code-->	age_df.head(2)
	Output-->	First name Last name Age 0 Tinu Elejogun 14 1 Blaszczyk Kostrzewski 25
.tail()	**To view the last few rows of the data**	
	Code-->	age_df.tail(2)
	Output-->	First name Last name Age 7 Thabang Mosoa 15 8 Kgaogelo Mosoa 11

- Combining DataFrame in Pandas are done using: Concat, Join and/or Merge.

C. Index in Pandas:

- In both Series & DataFrame structure of Pandas, we use Index to refer to the row & column which is crucial in data analytics.

Python Command	Purpose	Illustration
pd.Series(array,index)	**To customise the index**	
	Code-->	s1 = Series([10,20,30,40], index=['a','b','c','d'])
	Output-->	a 10 b 20 c 30 d 40
.index()	**To view the index elements**	
	Code-->	index_object = s1.index
	Output-->	['a', 'b', 'c', 'd']
index[index_number]	**To access the index elements**	
	Code-->	index_object[0]
	Output-->	'a'
index[-index_number:]	**To access elements using negative indexes**	
	Code-->	index_object[-2:]
	Output-->	['c', 'd']

- 'reindex() is done using:
  i. 'fill_value' method: Wherein values can be filled with a given number
  ii. 'ffill' method: Forward fill is an auto-fill method

D. Dropping Entries in Series & DataFrame:

Python Command	Purpose	Illustration
pd.Series	To create a series	
	Code-->	cars = pd.Series(['BMW','Audi','Merc'],index=['a','b','c'])
	Output-->	a BMW b Audi c Merc
.drop()	To drop value(s) from series	
	Code-->	cars = cars.drop('a')
	Output-->	b Audi c Merc
pd.DataFrame	To create a data frame	
	Code-->	cars_df = DataFrame(np.random.randn(9).reshape(3,3),index=['BMW','Audi','Merc'],columns=['test1','test2','test3'])
	Output-->	test1 test2 test3 BMW -0.027516 0.750294 -1.322448 Audi -0.067386 1.597434 -0.399865 Merc 0.962175 -0.584543 0.416939
.drop()	To drop rows from data frame	
	Code-->	cars_df = cars_df.drop('BMW')
	Output-->	test1 test2 test3 Audi -0.067386 1.597434 -0.399865 Merc 0.962175 -0.584543 0.416939
.drop(,axis=1)	To drop columns from data frame	
	Code-->	cars_df = cars_df.drop('test3',axis=1)
	Output-->	test1 test2 BMW -0.027516 0.750294 Audi -0.067386 1.597434 Merc 0.962175 -0.584543

E. Handling Null/NaN Values in Pandas:

Python Command	Purpose	Illustration
pd.Series	To create a series	
	Code-->	rev = Series([100,200,300,np.nan],index=['Audi','Merc','BMW','Toyota'])
	Output-->	Audi 100.0 Merc 200.0 BMW 300.0 Toyota NaN
.isnull()	To check for null values	
	Code-->	rev.isnull()
	Output-->	Audi False Merc False BMW False Toyota True
.dropna()	To drop a null value in a series	
	Code-->	rev.dropna()
	Output-->	Audi 100.0 Merc 200.0 BMW 300.0
pd.DataFrame	To create a data frame	
	Code-->	df2 = DataFrame([[1,2,3,np.nan],[4,5,6,7],[8,9,np.nan,np.nan],[12,np.nan,np.nan,np.nan]])
	Output-->	0 1 2 3 0 1 2.0 3.0 NaN 1 4 5.0 6.0 7.0 2 8 9.0 NaN NaN 3 12 NaN NaN NaN
.dropna(thresh)	If 3 or more values (in this case) in a row is non-null, then the row should exist	
	Code-->	df2.dropna(thresh=3)
	Output-->	0 1 2 3 0 1 2.0 3.0 NaN 1 4 5.0 6.0 7.0
.fillna({})	if we want to fill each column of missing values by different number, we use dictionary function	
	Code-->	df2.fillna({0:0,1:50,2:100,3:200})
	Output-->	0 1 2 3 0 1 2.0 3.0 200 1 4 5.0 6.0 7.0 2 8 9.0 100 200 3 12 50 100 200

F. Selecting & Modifying Entries in Pandas:

Python Command	Purpose	Illustration
pd.Series	**To create a series**	
	Code-->	s1 = Series([100,200,300],index=['a','b','c'])
	Output-->	a 100
		b 200
		c 300
series['index_name']	**To access an element of series**	
	Code-->	s1['a']
	Output-->	100
series['index_number']	**To access an element using index number**	
	Code-->	s1[0]
	Output-->	100
series[series(condition)]	**To access series using condition**	
	Code-->	s1[s1>150]
	Output-->	b 200
		c 300

- DataFrame is accessed using '.loc()' method.

G. Ranking & Sorting in Pandas Series:

i. 'sort_index()': This will facilitate sorting in ascending order
ii. 'sort_values()': To sort values in ascending order
iii. '.rank()': To rank the observations

H. Graphs in Pandas DataFrame:

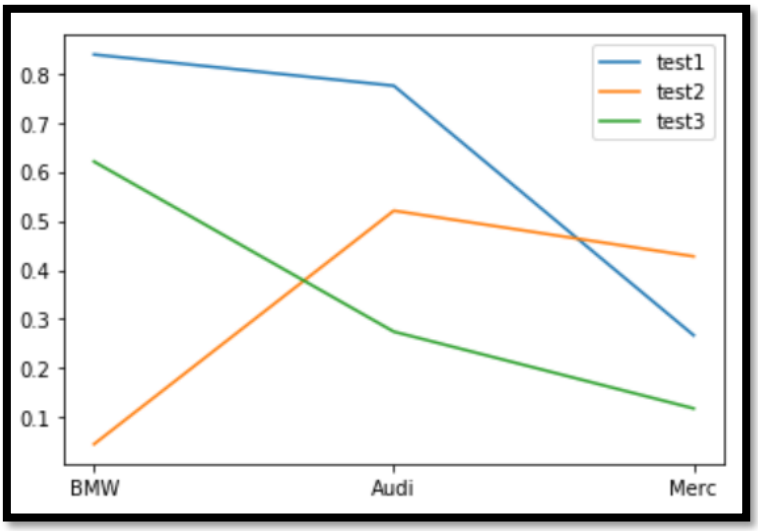
Matplotlib:

- Visualization tool in Python that contains various types of graphs for numerical as well as categorical variables.
- Pyplot is a sub element of Matplotlib.

```
#Matplotlib is imported in the notebook using:
import matplotlib.pyplot as plt
%matplotlib inline
```

```
#GRAPH OF A DATAFRAME
cars_df = pd.DataFrame(np.random.rand(9).reshape(3,3),index=['BMW','Audi','Merc'],columns=['test1','test2','test3'])
print(cars_df)
```

```
plt.plot(cars_df)
plt.legend(cars_df.columns,loc="upper right")
```



I. Statistics in Pandas DataFrame:

Python Command	Purpose	Illustration
np.array	**To create an array**	
	Code-->	array = np.array([[10,np.nan,20],[30,40,np.nan],[50,np.nan,60]])
pf.DataFrame	**To create a data frame**	
	Code-->	df = pd.DataFrame(arr_1,columns=list('ABC'))
	Output-->	A B C 0 10.0 NaN 20.0 1 30.0 40.0 NaN 2 50.0 NaN 60.0
.sum()	**To add across columns**	
	Code-->	df.sum()
	Output-->	A 90.0 B 40.0 C 80.0
.sum(axis=1)	**To add across rows**	
	Code-->	df.sum(axis=1)
	Output-->	0 30.0 1 70.0 2 110.0
.cumsum()	**To find the cummulative summation**	
	Code-->	df.cumsum()
	Output-->	A B C 0 10.0 NaN 20.0 1 40.0 40.0 NaN 2 90.0 NaN 80.0
.describe()	**To view the brief statistics of the data**	
	Code-->	df.describe()
	Output-->	

	A	B	C
count	3.0	1.0	2.000000
mean	30.0	40.0	40.000000
std	20.0	NaN	28.284271
min	10.0	40.0	20.000000
25%	20.0	40.0	30.000000
50%	30.0	40.0	40.000000
75%	40.0	40.0	50.000000
max	50.0	40.0	60.000000

Additional Note:

Python Command	Purpose
.columns	To see the column names
.groupby()	To group the data based on a particular variable
.iloc	To access numerical index
pd.join()	To combine the dataframe based on particular index
pd.merge()	To combine the dataframe based on a particular column
pd.concat()	To concatenate two dataframes