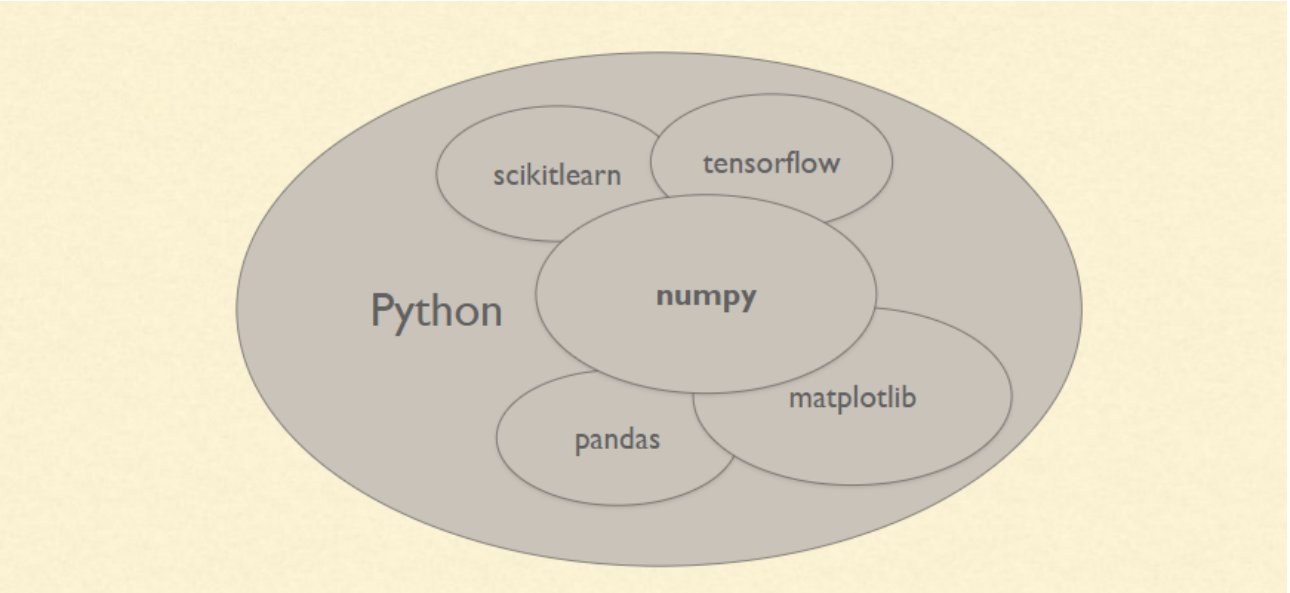
NUMPY, PANDAS, MATPLOTLIB



Python is increasingly being used as a scientific language. Matrix and vector manipulations are extremely important for scientific computations. Both NumPy and Pandas have emerged to be essential libraries for any scientific computation, including machine learning, in python due to their intuitive syntax and high-performance matrix computation capabilities.

What is NumPy?

NumPy stands for 'Numerical Python' or 'Numeric Python'. It is an open source module of Python which provides fast mathematical computation on arrays and matrices. Since, arrays and matrices are an essential part of the Machine Learning ecosystem, NumPy along with Machine Learning modules like Scikit-learn, Pandas, Matplotlib, TensorFlow, etc. complete the Python Machine Learning Ecosystem.

NumPy provides the essential multi-dimensional array-oriented computing functionalities designed for high-level mathematical functions and scientific computation. Numpy can be imported into the notebook using

```
>>> import numpy as np
```

NumPy's main object is the homogeneous multidimensional array. It is a table with same type elements, i.e, integers or string or characters (homogeneous), usually integers. In NumPy, dimensions are called axePIP. The number of axes is called the rank.

There are several ways to create an array in NumPy like np.array, np.zeros, no.ones, etc. Each of them provides some flexibility.

Command to create an array	Example

np.array	`>>> a = np.array([1, 2, 3])`
	`>>> type(a)`
	`<type 'numpy.ndarray'>`
	
	`>>> b = np.array((3, 4, 5))`
	`>>> type(b)`
	`<type 'numpy.ndarray'>`

np.ones	`>>> np.ones( (3,4), dtype=np.int16 )`
	`array([[ 1, 1, 1, 1],`
	`[ 1, 1, 1, 1],`
	`[ 1, 1, 1, 1]])`

np.full	`>>> np.full( (3,4), 0.11 )`
	`array([[ 0.11, 0.11, 0.11, 0.11],`
	`[ 0.11, 0.11, 0.11, 0.11],`
	`[ 0.11, 0.11, 0.11, 0.11]])`

np.arange	`>>> np.arange( 10, 30, 5 )`
	`array([10, 15, 20, 25])`
	
	`>>> np.arange( 0, 2, 0.3 )`
	`# it accepts float arguments`
	`array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])`

Some of the important attributes of a NumPy object are:

1. **Ndim:**displays the dimension of the array
2. **Shape:**returns a tuple of integers indicating the size of the array
3. **Size:**returns the total number of elements in the NumPy array
4. **Dtype**: returns the type of elements in the array, i.e., int64, character
5. **Itemsize:**returns the size in bytes of each item
6. **Reshape**: Reshapes the NumPy array

NumPy array elements can be accessed using indexing. Below are some of the useful examples:

A[2:5] will print items 2 to 4. Index in NumPy arrays starts from 0

A[2::2] will print items 2 to end skipping 2 items

A[::-1] will print the array in the reverse order

A[1:] will print from row 1 to end

What is Pandas?

Similar to NumPy, Pandas is one of the most widely used python libraries in data science. It provides high-performance, easy to use structures and data analysis tools. Unlike NumPy library which provides objects for multi-dimensional

arrays, Pandas provides in-memory 2d table object called Dataframe. It is like a spreadsheet with column names and row labels.
Hence, with 2d tables, pandas is capable of providing many additional functionalities like creating pivot tables, computing columns based on other columns and plotting graphs. Pandas can be imported into Python using:

```
>>> import pandas as pd
```

Some commonly used data structures in pandas are:

1. **Series objects**: 1D array, similar to a column in a spreadsheet
2. **DataFrame objects:**2D table, similar to a spreadsheet
3. **Panel objects:**Dictionary of DataFrames, similar to sheet in MS Excel

Pandas Series object is created using pd.Series function. Each row is provided with an index and by defaults is assigned numerical values starting from 0. Like NumPy, Pandas also provide the basic mathematical functionalities like addition, subtraction and conditional operations and broadcasting.
Pandas dataframe object represents a spreadsheet with cell values, column names, and row index labels. Dataframe can be visualized as dictionaries of Series. Dataframe rows and columns are simple and intuitive to access. Pandas also provide SQL-like functionality to filter, sort rows based on conditions. For example,

```
>>> people_dict = { "weight": pd.Series([68, 83, 112],index=["alice", "bob",
"charles"]), "birthyear": pd.Series([1984, 1985, 1992], index=["bob", "alice",
"charles"], name="year"),
"children": pd.Series([0, 3], index=["charles", "bob"]),
"hobby": pd.Series(["Biking", "Dancing"], index=["alice", "bob"]),}
>>> people = pd.DataFrame(people_dict)
>>> people
```

```
>>> p
```

	birthyear	children	hobby	weight
alice	1985	NaN	Biking	68

	birthyear	children	hobby	weight
alice	1985	NaN	Biking	68
bob	1984	3.0	Dancing	83
charles	1992	0.0	NaN	112

Create dataframe from list

```
1.import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
       'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
```

```
df = pd.DataFrame(lst)
df
```

2.
```
import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is', 'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
# with indices and columns specified
df = pd.DataFrame(lst, index =['a', 'b', 'c', 'd', 'e', 'f', 'g'],columns
=['Names'])
df
```

3.
```
import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is', 'portal', 'for', 'Geeks']

# list of int
lst2 = [11, 22, 33, 44, 55, 66, 77]

# Calling DataFrame constructor after zipping
# both lists, with columns specified
df = pd.DataFrame(list(zip(lst, lst2)),
        columns =['Name', 'val'])
df
```

4.
```
import pandas as pd

# List1
lst = [['tom', 25], ['krish', 30],
    ['nick', 26], ['juli', 22]]

df = pd.DataFrame(lst, columns =['Name', 'Age'])
df
```

5.
```
# import pandas as pd
import pandas as pd

# List1
lst = [['tom', 'reacher', 25], ['krish', 'pete', 30],
    ['nick', 'wilson', 26], ['juli', 'williams', 22]]
```

```
df = pd.DataFrame(lst, columns =['FName', 'LName', 'Age'], dtype =
float)
df
```

6. Python code demonstrate creating
```
# pandas DataFrame with indexed by

# DataFrame using arrays.
import pandas as pd

# initialize data of lists.
data = {'Name':['Tom', 'Jack', 'nick', 'juli'],
     'marks':[99, 98, 95, 90]}

# Creates pandas DataFrame.
df = pd.DataFrame(data, index =['rank1',
                   'rank2',
                   'rank3',
                   'rank4'])

# print the data
df
```

Dataframes can also be easily exported and imported from CSV, Excel, JSON, HTML and SQL database. Some other essential methods that are present in dataframes are:

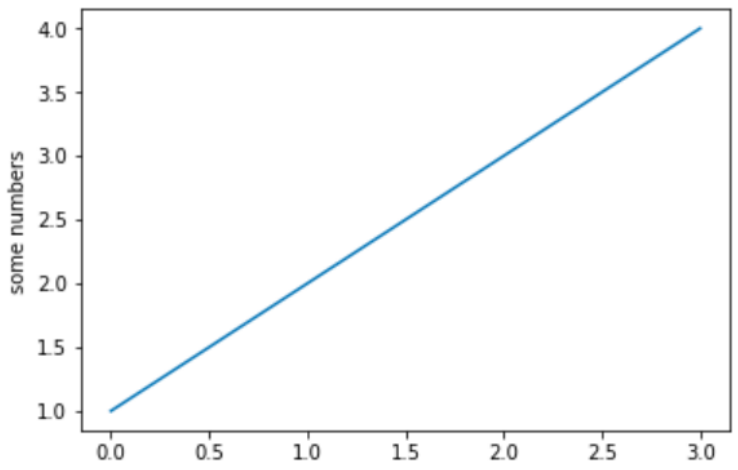
1. **head():** returns the top 5 rows in the dataframe object
2. **tail():** returns the bottom 5 rows in the dataframe
3. **info():** prints the summary of the dataframe
4. **describe():** gives a nice overview of the main aggregated values over each column

What is matplotlib?

Matplotlib is a 2d plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments. Matplotlib can be used in Python scripts, Python and IPython shell, Jupyter Notebook, web application servers and GUI toolkits.
matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Majority of plotting commands in pyplot have MATLAB analogs with similar arguments. Let us take a couple of examples:

Example 1: Plotting a line graph	Example 2: Plotting a histogram
`>>> import matplotlib.pyplot as plt`	`>>> import matplotlib.pyplot as plt`

```
>>> plt.plot([1,2,3,4])
>>> plt.ylabel('some numbers')
>>> plt.show()
```




```
>>> x = [21,22,23,4,5,6,77,8,9,10,31,32,33,34,3
>>> num_bins = 5
>>> plt.hist(x, num_bins, facecolor='blue')
>>> plt.show()
```