Unit – 3

Libraries for Data Analysis

3.1 Introduction to NumPy & Pandas

3.1.1 NumPy :

NumPy, which stands for **Numerical Python**, is an open-source Python library consisting of multidimensional and single-dimensional array elements. It's a standard that computes numerical data in Python. NumPy is most widely used in almost every domain where numerical computation is required, like science and engineering; hence, the NumPy API functionalities are highly utilized in data science and scientific Python packages,
including Pandas, SciPy, Matplotlib, scikit-learn, scikit-image, and many more.

This NumPy tutorial explains the basics of NumPy, such as its architecture and environment. It also discusses array functions, types of indexing, etc., and then extends to learn Matplotlib, Pandas, SciPy, and other important Python libraries. All this is explained with the help of examples for better understanding.

KEY POINTS

- NumPy is a Python library.
- NumPy is used for working with arrays.
- NumPy is short for "Numerical Python".

- **Why NumPy - Need of NumPy**

NumPy is a fundamental package for numerical computation in Python. It provides mathematical functions to compute data as well as functions to operate multi-dimensional arrays and matrices efficiently. Here are some reasons why NumPy is essential:

- NumPy includes a wide range of mathematical functions for basic arithmetic, linear algebra, Fourier analysis, and more.
- NumPy performs numerical operations on large datasets efficiently.
- NumPy supports multi-dimensional arrays, allowing for the representation of complex data structures such as images, sound waves, and tensors in machine learning models.
- It supports the writing of concise and readable code for complex mathematical computations.

- NumPy integrates with other libraries to do scientific computation; these are SciPy (for scientific computing), Pandas (for data manipulation and analysis), and scikit-learn (for machine learning).
- Many scientific and numerical computing libraries and tools are built on top of NumPy.
- Its widespread adoption and stability make it a standard choice for numerical computing tasks.

Overall, NumPy plays a crucial role in the Python ecosystem for scientific computing, data analysis, machine learning, and more. Its efficient array operations and extensive mathematical functions make it an indispensable tool for working with numerical data in Python.

- **NumPy Applications - Uses of NumPy**

The NumPy API in Python is used primarily for numerical computing. It provides support for a wide range of mathematical functions to operate on data efficiently. The following are some common application areas where NumPy is extensively used:

1. **Data Analysis**: NumPy offers rapid and effective array operations, rendering it well-suited for tasks like data cleansing, filtering, and transformation. It is predominantly used in the analysis and scientific handling of data, particularly when working with extensive, large datasets.
2. **Machine Learning and Artificial Intelligence**: Different machine learning and deep learning frameworks in Python, such as TensorFlow, and PyTorch, rely on NumPy arrays for handling input data, model parameters, and outputs.
3. **Image and Signal Processing**: NumPy is extensively used in processing and analysing images and signals.
4. **Data Visualisation**: NumPy doesn't provide data visualisation but supports Matplotlib and Seaborn libraries to generate plots and visualisations from numerical data.

EXAMPLE 1

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

OUTPUT

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

EXAMPLE 2: NumPy array indexing

```
import numpy as np

arr = np.array([1, 2, 3, 4])

print(arr[0])
```

OUTPUT

1

EXAMPLE 3: NumPy using 2-D Array

```
import numpy as np

arr = np.array([[1,2,3,4,5], [6,7,8,9,10]])

print(arr)
```

OUTPUT

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

[6,7,8,9,10]]

3.1.2 Pandas :

Pandas is an open-source, BSD-licensed Python library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc. Pandas is a powerful Python library used for data manipulation and analysis. It provides easy-to-use data structures and tools for handling structured data.

KEY POINTS

- Pandas is a Python library.
- Pandas is used to analyze data.
- The name Pandas derived from Panel Data &Python Data Analysis

- **What is Pandas?**

Pandas is a powerful Python library that is specifically designed to work on data frames that have "relational" or "labeled" data. Its aim aligns with doing real-world data analysis using Python. Its flexibility and functionality make it indispensable for various data-related tasks. Hence, this Python package works well for data manipulation, operating a dataset, exploring a data frame, data analysis, and machine learning-related tasks. To work on it we should first install it using a pip command like "pip install pandas" and then import it like "import pandas as pd". After successfully installing and importing, we can enjoy the innovative functions of pandas to work on datasets or data frames. Pandas versatility and ease of use make it a go-to tool for working with structured data in Python.

Generally, Pandas operates a data frame using Series and DataFrame; where Series works on a one-dimensional labeled array holding data of any type like integers, strings, and objects, while a DataFrame is a two-dimensional data structure that manages and operates data in tabular form (using rows and columns).

- **Why Pandas?**

The beauty of Pandas is that it simplifies the task related to data frames and makes it simple to do many of the time-consuming, repetitive tasks involved in working with data frames, such as:

- **Import datasets** - available in the form of spreadsheets, comma-separated values (CSV) files, and more.
- **Data cleansing** - dealing with missing values and representing them as NaN, NA, or NaT.
- **Size mutability** - columns can be added and removed from DataFrame and higher-dimensional objects.
- **Data normalization** – normalize the data into a suitable format for analysis.

- Applications of Pandas

The most common applications of Pandas are as follows:

- **Data Cleaning**: Pandas provides functionalities to clean messy data, deal with incomplete or inconsistent data, handle missing values, remove duplicates, and standardise formats to do effective data analysis.
- **Data Exploration**: Pandas easily summarise statistics, find trends, and visualise data using built-in plotting functions, Matplotlib, or Seaborn integration.
- **Data Preparation**: Pandas may pivot, melt, convert variables, and merge datasets based on common columns to prepare data for analysis.
- **Data Analysis**: Pandas supports descriptive statistics, time series analysis, group-by operations, and custom functions.

- **Data Visualisation**: Pandas itself has basic plotting capabilities; it integrates and supports data visualisation libraries like Matplotlib, Seaborn, and Plotly to create innovative visualisations.

EXAMPLE :

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a)

print (myvar)
```

OUTPUT

```
0  1

1  7

2  2
```

```
import pandas as pd

a = [1, 7, 2]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
```

OUTPUT

```
x  1

y  7

z  2
```

➢ What is a DataFrame?

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

EXAMPLE :

```
import pandas as pd
data = {
  "marks": [420, 380, 390],
  "rollno": [50, 40, 45]
}
df = pd.DataFrame(data)
print(df)
```

OUTPUT

	marks	rollno
0	420	50
1	380	40
2	390	45

3.2 Data Manipulation with NumPy & Pandas

3.2.2 NumPy

NumPy (Numerical Python) provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.

a) Example

```
# using array-scalar type
import numpy as np
dt = np.dtype(np.int32)
print(dt)
```

The output is as follows – int32

b) ndarray.shape

This array attribute returns a tuple consisting of array dimensions. It can also be used to resize the array.

Example

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
print(a.shape)
```

The output is as follows –

(2, 3)

c) numpy.arange

```
import numpy as np
x = np.arange(5)
print(x)
```

Its output would be as follows –

[0 1 2 3 4]

3.2.3 Pandas

Pandas is built on top of NumPy and provides data structures (like Series and DataFrame) to manipulate and analyze structured data more easily.

Sr.No	Parameter & Description
1	**Data** : data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	**Index** :For the row labels, the Index to be used for the resulting frame is Optional Default np.arange(n) if no index is passed.
3	**Columns** :For column labels, the optional default syntax is - np.arange(n). This is only true if no index is passed.
4	**Dtype** :Data type of each column.

5 **Copy** :This command (or whatever it is) is used for copying of data, if the default is False.

a) Create an Empty DataFrame

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
df = pd.DataFrame()
print df
```

Its **output** is as follows –

Empty DataFrame

Columns: [ ]

Index: [ ]

b) Create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists.

Example

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print (df)
```

Its **output** is as follows –

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

c) Create a DataFrame from Dict of ndarrays / Lists

All the **ndarrays** must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.

If no index is passed, then by default, index will be range(n), where **n** is the array length.

Example

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve',
'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print (df)
```

Its **output** is as follows –

	Age	Name
0	28	Tom
1	34	Jack
2	29	Steve
3	42	Ricky

Note – Observe the values 0,1,2,3. They are the default index assigned to each using the function range(n).

3.3 Data Visualization with Matplotlib & Seabom

3.3.1 Data Visualization

It may sometimes seem easier to go through a set of data points and build insights from it but usually this process may not yield good results. There could be a lot of things left undiscovered as a result of this process. Additionally, most of the data sets used in real life are too big to do any analysis manually. This is essentially where data visualization steps in.

Data visualization is an easier way of presenting the data, however complex it is, to analyze trends and relationships amongst variables with the help of pictorial representation.

The following are the advantages of Data Visualization

- Easier representation of compels data
- Highlights good and bad performing areas
- Explores relationship between data points
- Identifies data patterns even for larger data points

While building visualization, it is always a good practice to keep some below mentioned points in mind

- Ensure appropriate usage of shapes, colors, and size while building visualization
- Plots/graphs using a co-ordinate system are more pronounced
- Knowledge of suitable plot with respect to the data types brings more clarity to the information
- Usage of labels, titles, legends and pointers passes seamless information the wider audience

3.3.2 Matplotlib

It is an amazing visualization library in Python for 2D plots of arrays, It is a multi-platform data visualization library built on *NumPy* arrays and designed to work with the broader *SciPy* stack. It was introduced by John Hunter in the year 2002. Some of the benefits and features of *matplotlib*

- It's fast, efficient as it is based on *numpy* and also easier to build
- Has undergone a lot of improvements from the open source community since inception and hence a better library having advanced features as well
- Well maintained visualization output with high quality graphics draws a lot of users to it
- Basic as well as advanced charts could be very easily built
- From the users/developers point of view, since it has a large community support, resolving issues and debugging becomes much easier

Example

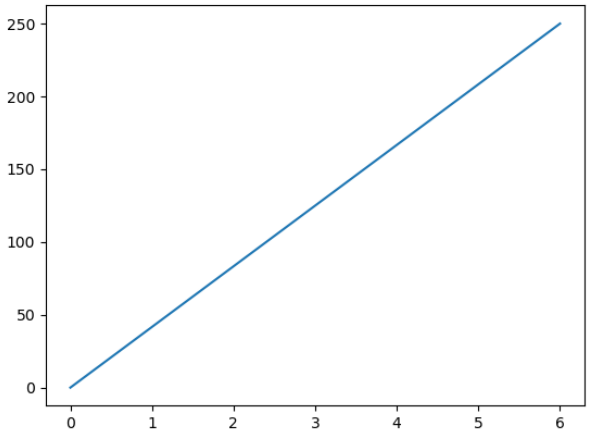
Draw a line in a diagram from position (0,0) to position (6,250):

```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```

Output :



Seaborn : Conceptualized and built originally at the Stanford University, this library sits on top of *matplotlib*. In a sense, it has some flavors of *matplotlib* while from the visualization point, it is much better than *matplotlib* and has added features as well. Below are its advantages.

- Built-in themes aid better visualization
- Statistical functions aiding better data insights
- Better aesthetics and built-in plots.

	Matplotlib	**Seaborn**
Level of Abstraction	Matplotlib is more low-level and requires more code for customizations.	Seaborn abstracts some complexities by enabling easier creation of complex statistical plots.
Default Styles	Matplotlib doesn't have better default styles and color palettes when compared to seaborn.	Seaborn has better default styles and color palettes by making its plots visually appealing without much customization.
Specialized Plots	Matplotlib require more effort to plot certain plots readily.	Seaborn offers certain types of plots that are not readily available or require more effort in Matplotlib.
When to use each library	We can use this library when we need fine-grained control over the appearance of our plots or when creating non-standard plots that may not be available in other libraries.	We can use this library when working with statistical data especially for quick exploration and visualization of distributions, relationships and categories within the data. Seaborn's high-level abstractions and default styles make it convenient for this purpose.