**Chapter 8**

**Lecture Notes**

**Introduction:**

In this module, we shall explore two powerful libraries for data analysis and visualization: NumPy and Pandas. NumPy provides support for efficient numerical operations and multi-dimensional arrays, while Pandas offers high-performance data manipulation and analysis tools. Additionally, we will introduce the Matplotlib and Seaborn libraries for data visualization. By the end of this module, you will have a solid understanding of these essential Python libraries for data science.

**1. NumPy:**

**1.1 Introduction to NumPy:**

NumPy is a fundamental library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is widely used for scientific and data analysis tasks due to its speed and versatility.

**1.2 Creating and Printing Ndarray:**

The core object in NumPy is the ndarray (N-dimensional array), which represents a grid of values. We can create ndarrays using various methods, such as initializing from lists or using built-in functions like zeros() and ones(). Printing ndarrays provides insights into their shape, dimensions, and contents.

**1.3 Class and Attributes of Ndarray:**

Ndarrays have several attributes that provide information about the array, including its shape, size, data type, and more. Understanding these attributes is crucial for effectively working with ndarrays.

**1.4 Basic Operations:**

NumPy enables various basic operations on ndarrays, such as element-wise arithmetic operations, array indexing, slicing, and reshaping. These operations allow us to perform computations efficiently and manipulate data effectively.

**1.5 Mathematical Functions of NumPy:**

NumPy provides a vast array of mathematical functions to perform operations on ndarrays. These functions include mathematical operations (e.g., sin(), cos(), sqrt()), statistical functions (e.g., mean(), std(), median()), and linear algebra operations (e.g., dot(), transpose(), inverse()). Understanding and utilizing these functions is essential for data analysis tasks. Some mathematical NumPy functions are as follows:

1. Trigonometric Functions:

- `numpy.sin(x)`: Returns the sine of the input array.

- `numpy.cos(x)`: Returns the cosine of the input array.

- `numpy.tan(x)`: Returns the tangent of the input array.

- Example usage:

import numpy as np

x = np.array([0, np.pi/2, np.pi])

print(np.sin(x))

# Output: [0. 1. 1.2246468e-16]

2. Exponential and Logarithmic Functions:

- `numpy.exp(x)`: Returns the exponential of the input array.

- `numpy.log(x)`: Returns the natural logarithm of the input array.

- `numpy.log10(x)`: Returns the base 10 logarithm of the input array.

- Example usage:

import numpy as np

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

print(np.exp(x))

# Output: [ 2.71828183 7.3890561 20.08553692]

3. Statistical Functions:

- `numpy.mean(x)`: Returns the arithmetic mean along the specified axis of the input array.

- `numpy.median(x)`: Returns the median value along the specified axis of the input array.

- `numpy.std(x)`: Returns the standard deviation along the specified axis of the input array.

- Example usage:

import numpy as np

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

print(np.mean(x))

# Output: 3.0

4. Linear Algebra Functions:

- `numpy.dot(a, b)`: Returns the dot product of two arrays.

- `numpy.linalg.inv(a)`: Computes the multiplicative inverse of a matrix.

- `numpy.linalg.eig(a)`: Computes the eigenvalues and eigenvectors of a square matrix.

- Example usage:

import numpy as np

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

b = np.array([[5, 6], [7, 8]])

print(np.dot(a, b))

# Output: [[19 22] [43 50]]

5. Random Number Generation:

- `numpy.random.rand(shape)`: Generates random numbers in the range [0, 1] of the specified shape.

- `numpy.random.randint(low, high, size)`: Generates random integers between low (inclusive) and high (exclusive) of the specified size.

- Example usage:

import numpy as np

print(np.random.rand(3, 2))

# Output: [[0.12345678 0.23456789] [0.34567891 0.45678901] [0.56789012 0.67890123]]

NumPy offers many more mathematical functions for various purposes, such as array manipulation, Fourier transformations, polynomial operations, and more. It is recommended to explore the NumPy documentation to gain a comprehensive understanding of the available functions and their usage.

**2. Pandas:**

**2.1 Understanding DataFrames:**

Pandas introduces the DataFrame, a powerful data structure for data manipulation and analysis. It provides a tabular representation of data with labeled rows and columns. Understanding DataFrames and their components is crucial for working with structured data effectively.

**2.2 View and Select Data:**

Pandas offers various methods to view and select data from DataFrames. This includes techniques like head, tail, indexing, slicing, and filtering based on conditions. These operations enable us to explore and extract relevant information from large datasets.

**2.3 Missing Values and Data Operations:**

Handling missing values is a common challenge in data analysis. Pandas provides functions and methods to identify, handle, and impute missing values. Additionally, we will explore data operations such as sorting, merging, and grouping data, which are vital for data manipulation tasks.

**2.4 File Read and Write Operations:**

Pandas allows seamless reading and writing of data from various file formats, including CSV, Excel, SQL databases, and more. Understanding how to import data into DataFrames and export DataFrames to different formats is essential for data preprocessing and analysis.

**3. Matplotlib**

**Introduction to Matplotlib Library:**

Matplotlib is a comprehensive visualization library in Python, pivotal for producing static, animated, and interactive visualizations. Rooted in the scientific computing community, it serves as a foundational package for most Python-based data plotting libraries.

Here is the brief overview of popular chart in matplotlib

**1. Line Plot:**

Used to display information as a series of data points connected by straight line segments.

import matplotlib.pyplot as plt

x = [1, 2, 3, 4]

y = [10, 12, 15, 17]

plt.plot(x, y)

plt.show()

**2. Bar chart**:

Represents data with rectangular bars with heights or lengths proportional to the values they represent.

bars = ['A', 'B', 'C']

values = [10, 23, 17]

plt.bar(bars, values)

plt.show()

**3. Histogram**

An accurate representation of the distribution of numerical data.

data = [21, 22, 23, 4, 5, 6, 77, 8, 9, 10, 31, 32, 33, 34, 35, 36, 37, 18, 49, 50, 100]

plt.hist(data, bins=5)

plt.show()

**4. Scatter Chart:**

Uses dots to represent the values obtained for two different variables, depicting how much one variable is affected by another.

x = [10, 20, 30, 40]

y = [5, 15, 10, 25]

plt.scatter(x, y)

plt.show()

**5. Pie Chart:**

labels = 'A', 'B', 'C', 'D'

sizes = [15, 30, 45, 10]

plt.pie(sizes, labels=labels)

plt.axis('equal')

plt.show()

**4. Seaborn**

**Introduction to the Seaborn:**

Seaborn is a data visualization library in Python that provides a high-level interface for creating attractive graphs. Built on top of Matplotlib, it is integrated with Pandas data structures. Seaborn comes with several built-in themes and color palettes to enhance the visual appeal of plots. It excels in creating statistical graphics and simplifies the process of deriving insights from data.

**1. Distplot:**

Combines a histogram and a kernel density estimate to represent the distribution of data.

import seaborn as sns

data = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9]

sns.distplot(data)

**2. Boxplot:**

Represents data in terms of quartiles and helps in identifying outliers.

import seaborn as sns

sns.set\_style("whitegrid")

tips = sns.load\_dataset("tips")

sns.boxplot(x=tips["total\_bill"])

**3. Pairplot:**

Plots pairwise relationships in a dataset by creating scatterplots. Useful for preliminary exploratory data analysis.

import seaborn as sns

iris = sns.load\_dataset("iris")

sns.pairplot(iris)

**4. Heatmap:**

Represents data as colors in a matrix which is useful for visualizing frequency or correlation.

import seaborn as sns

data = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

sns.heatmap(data)

**5. Barplot:**

Displays the relationship between two variables. One variable is plotted on the x-axis and another on the y-axis.

import seaborn as sns

tips = sns.load\_dataset("tips")

sns.barplot(x="day", y="total\_bill", data=tips)

**6. Scatterplot:**

Plots two variables against each other, displaying how much one variable is affected by another.

import seaborn as sns

tips = sns.load\_dataset("tips")

sns.scatterplot(x="total\_bill", y="tip", data=tips)