Department of Information Engineering Technology

The Superior University, Lahore

Artificial Intelligence Lab

Experiment No.2

Data Analysis with Python

Prepared for

_____

By:

Name: _____

ID: _____

Section: _____

Semester: _____

Total Marks: _____

Obtained Marks: _____

Signature: _____

Date: _____

Experiment No. 2

Data Analysis with Python

(Rubrics)

Name: _____ **Roll No. : _____**

A. PSYCHOMOTOR

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Follow Procedures	1	0	0.25	0.5	0.75	1	
2	Software and Simulations	2	0	0.5	1	1.5	2	
3	Accuracy in Output Results	3	0	0.75	1.5	2.25	3	
Sub Total		**6**	**Sub Total marks Obtained in Psychomotor(P)**					

B. AFFECTIVE

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Respond to Questions	1	0	0.25	0.5	0.75	1	
2	Lab Report	1	0	0.25	0.5	0.75	1	
3	Assigned Task	2	0	0.5	1	1.5	2	
Sub Total		**4**	**Sub Total marks Obtained in Affective (A)**					

Instructor Name:_____ **Total Marks (P+A): _____**

Instructor Signature:_____ **Obtained Marks (P+A): _____**

Introduction to Data Analysis

Data analysis is the process of inspecting, cleaning, transforming, and modeling data to discover meaningful insights, draw conclusions, and support decision-making. It plays a crucial role in various fields, including business, science, healthcare, finance, and many others, as it helps organizations and individuals make informed choices and better understand their surroundings. Python is a popular programming language for data analysis due to its extensive libraries, user-friendly syntax, and strong community support.

Here's how data analysis is typically performed with Python:

- **Data Collection:** Python offers several libraries and packages for data collection. You can use libraries like pandas to read data from various sources such as **CSV, Excel, SQL databases, JSON,** and more. For web scraping, you might use libraries like **BeautifulSoup and Scrapy.**
- **Data Cleaning:** The Pandas library is a powerful tool for data cleaning. You can use it to handle missing values, remove duplicates, filter data, and perform other data cleaning operations.
- **Exploratory Data Analysis (EDA):** Libraries like matplotlib, **Seaborn, and Plotly** are commonly used for creating visualizations during EDA. These libraries allow you to create histograms, scatter plots, bar charts, and other types of plots to explore your data.
- **Data Transformation:** pandas also provide functions for data transformation, such as reshaping data with pivot, melt, and performing calculations on columns.
- **Machine Learning:** Python's **scikit-learn library** is widely used for machine learning tasks. It provides a range of machine learning algorithms for classification, regression, clustering, and more. Other libraries like **TensorFlow and PyTorch** are used for deep learning tasks.
- **Data Visualization:** Libraries like **matplotlib, Seaborn, and Plotly** are used to create customized and informative data visualizations. Jupyter notebooks are often used to combine code with visualizations for interactive analysis.
- **Iterative Process:** Data analysis is often an iterative process, and Python's scripting capabilities make it easy to modify and rerun code as needed when new data is available or additional questions arise.

In summary, data analysis is a multifaceted process that involves collecting, cleaning, exploring, and analyzing data to extract meaningful information. It empowers organizations and individuals to make data-driven decisions, solve problems, and gain valuable insights into various domains. Effective data analysis requires a combination of domain expertise, statistical knowledge, and proficiency in data analysis tools and programming languages.

Data Analysis with Python

Python is a popular programming language for data analysis due to its extensive libraries, user-friendly syntax, and strong community support.

NumPy:

NumPy (Numerical Python) is a fundamental Python library for scientific and numerical computing. It provides support for working with large, multi-dimensional arrays and matrices of data, as well as a variety of high-level mathematical functions to operate on these arrays.

1. **Creating NumPy Arrays:** NumPy arrays (ndarrays) are the primary data structure for storing numerical data. You can create arrays from Python lists, using functions like **np.array().** Arrays can be one-dimensional (1D) or multi-dimensional (2D or higher). NumPy provides functions for creating arrays filled with zeros, ones, and generating sequences of values.

```
import numpy as np

# Creating a 1D array
arr1d = np.array([1, 2, 3, 4, 5])
print(arr1d) # Output  [1 2 3 4 5]
print(arr1d.shape) #Output (5,)
print(arr1d.size) # Output 5
# Creating a 2D array (matrix)
arr2d = np.array([[1, 2, 3], [4, 5, 6]])
print(arr2d)
""" Output  [[1 2 3]
             [4 5 6]] """
# Creating a range of values
range_arr = np.arange(10)  # Array with values 0, 2, 4, 6, 8
print(range_arr) #Output [0 1 2 3 4 5 6 7 8 9]
```

2. **Array Operations:** NumPy allows you to perform various operations on arrays, including arithmetic operations like addition, subtraction, multiplication, and division. Element-wise operations apply a mathematical function to each element of the array.

```
import numpy as np

# Arithmetic operations
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
addition = a + b
subtraction = a - b
multiplication = a * b
division = a / b
```

```
# Element-wise operations
squared = np.square(a)
print(squared)
square_root = np.sqrt(a)
exponential = np.exp(a)
```

3. **Indexing and Slicing:** You can access individual elements of an array using indexing. Slicing allows you to extract a subset of an array by specifying a range of indices.

```
import numpy as np
arr = np.array([0, 1, 2, 3, 4, 5])

# Accessing elements
element = arr[2]  # Get the element at index 2
print(element) #Output 2
# Slicing
slice_arr = arr[2:5]  # Get elements from index 2 to 4 (exclusive)
print(slice_arr) #Output [2 3 4]
```

4. **Aggregation and Statistics:** NumPy provides functions for calculating summary statistics on arrays, such as mean, median, sum, maximum, and minimum. These functions are useful for data analysis and summarizing data

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5])

mean_value = np.mean(arr)      # Mean (average)
print(mean_value) # Output 3.0
median_value = np.median(arr) # Median
print(median_value) # Output 3.0
average =np.average(arr) # Average
print(average) # Output 3.0
sum_value = np.sum(arr)        # Sum
print(sum_value) # Output 15
max_value = np.max(arr)        # Maximum
print(max_value ) # Output 5
min_value = np.min(arr)        # Minimum
print(min_value) # Output 1
variance=np.var(arr)          #variance
print(variance) # Output 2.0
Standard_deviation= np.std(arr) #Standard deviation
print(Standard_deviation) # Output 1.414
```

5. **Reshaping and Transposing:** Reshaping changes the dimensions of an array while preserving the total number of elements. Transposing swaps, the rows and columns of a 2D array

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
""" Output: [[1 2 3]
             [4 5 6]] """
# Reshaping
reshaped_arr = arr.reshape(3, 2)  # Reshape to a 3x2 matrix
print(reshaped_arr)
""" Output: [[1 2]
             [3 4]
             [5 6]] """
# Transposing
transposed_arr = arr.T  # Transpose the matrix
print(transposed_arr)
""" Output: [[1 4]
             [2 5]
             [3 6]] """
```

Pandas:

Pandas is a widely used open-source Python library for data manipulation and analysis. It provides easy-to-use data structures and functions for working with structured data, such as tables and time series data. Pandas is an essential tool in the field of data science and is often used in combination with other libraries like NumPy, Matplotlib, and Scikit-Learn.

1. **Creating a DataFrame:** Create a Pandas DataFrame from data, which can be a dictionary, a list of dictionaries, or from other data sources

```
import pandas as pd
data = {'Name': ['Waqar', 'Rehan', 'Manzoor'],
        'Age': [25, 30, 35]}
df = pd.DataFrame(data)
print(df)
""" Output    Name    Age
          0  Waqar   25
          1  Rehan   30
          2  Manzoor 35 """
```

2. **Reading Data from a File:** Read data from an external file, such as a CSV file, Excel spreadsheet, or SQL database, into a DataFrame.

```
df= pd.read_csv('/content/drive/MyDrive/weather.csv')
```

3. **Basic DataFrame Operations:** Perform basic operations like displaying the first few rows, checking data types, and getting summary statistics.

```
print(df.head())         # Display the first few rows
print(df.tail())         # Display the last few rows
print(df.dtypes)         # Check data types of columns
print(df.describe())     # Get summary statistics
```

4. **Data Selection and Indexing:** Select specific columns or rows using indexing and conditions.

```
selected_column = df['Name']  # Select a specific column
filtered_data = df[df['Age'] > 30]  # Filter rows based on a condition
```

5. **Data Cleaning and Handling Missing Values:** Handle missing values by dropping or filling them python

```
df.fillna(0,inplace=True) # Fill missing values with a specific value
```

6. **Correlation:** You can use the .corr() method in Pandas to calculate the Pearson correlation coefficient between two or more columns in a DataFrame.

```
df['Apparent Temperature (C)'].corr(df['Temperature (C)'])#Showing
relation b/w Apparent Temperature (C) column & Temperature (C) column
```

7. **Saving Data to a File:** Save a Pandas DataFrame to an external file.

```
df.to_csv('new_data.csv', index=False)  # Save as CSV
df.to_excel('new_data.xlsx', index=False)  # Save as Excel
```

Matplotlib:

Matplotlib is a popular Python library for creating static, animated, and interactive visualizations in various formats, such as charts, plots, graphs, and more. It is widely used in data visualization, scientific research, and other fields where visualizing data is essential.

1. **Importing Matplotlib:** Import the Matplotlib library and, if needed, a specific module within Matplotlib, such as pyplot, which provides a MATLAB-like interface for creating plots..

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

2. **Basic Line Plot:** Create a simple line plot with x and y data.

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 1, 3, 5]
# Create a line plot
plt.plot(x, y)
# Adding labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Line Plot')
plt.show()  # Display the plot
```

3. **Scatter Plot:** Create a scatter plot to visualize individual data points.

```
x = [1, 2, 3, 4, 5]
y = [2, 4, 1, 3, 5]
plt.scatter(x, y)
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Scatter Plot')
plt.show()
```

4. **Bar Chart:** Create a bar chart to represent categorical data.

```
categories = ['A', 'B', 'C', 'D', 'E']
values = [10, 15, 7, 12, 9]
plt.bar(categories, values)
plt.xlabel('Categories')
plt.ylabel('Values')
plt.title('Bar Chart')
plt.show()
```

5. **Creating a Line Plot with Pandas:** Creating a Line Plot with Pandas is a common data visualization task in which you use the Pandas library in Python to generate a line plot to visualize the relationship between two variables.

```
import pandas as pd
import matplotlib.pyplot as plt

# Sample DataFrame
data = {'Number': [1, 2, 3, 4, 5],
        'Age': [25, 30, 35, 40, 45]}
df = pd.DataFrame(data)

# Create a line plot
df.plot(kind='line', x='Number', y='Age', figsize=(4, 4), grid=True)

# Adding labels and title
plt.xlabel('Number')
plt.ylabel('Age')
plt.title('Line Plot of Age vs. Number')

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

Lab Task

1. A dataset (e.g., CSV file) with information about students' exam scores in different subjects. You have to load the dataset using Pandas, calculate summary statistics (mean, median, standard deviation) for each subject using NumPy, and create a bar chart using Matplotlib to visualize the average scores for each subject.
2. A dataset containing information about stock prices for a set of companies over time. You have to use Pandas to filter the data for a specific company, select a particular date range, and then use Matplotlib to create a line chart to visualize the stock price trends for that company within the chosen date range.
3. A dataset with information about monthly sales for multiple product categories. You have to use Pandas to aggregate the data to find the total sales for each category over time. Then, use Matplotlib to create a stacked area chart to visualize the sales trends for each category over a specific period.

Home Task

1. A dataset containing information about weather conditions over the course of a year. You have to use Pandas to calculate the monthly average temperature and precipitation, and then create a custom bar chart using Matplotlib that displays both temperature and precipitation for each month.
2. A dataset with information about customer orders, including order date, customer demographics, and purchase details. You have to use Pandas to analyze customer purchasing behavior (e.g., total spending by customer segment) and create a customized visual representation (e.g., a grouped bar chart or a heatmap) using Matplotlib to illustrate their findings.