Shram Sadhana Bombay Trust's

COLLEGE OF ENGINEERING AND TECHNOLOGY,

BAMBHORI POST BOX NO. 94, JALGAON – 425001. (M.S.)

Included under section 2 (f) & 12 (B) of the UGC Act, 1956

NAAC Re-accredited Grade: A (3.14) (2nd Cycle)

ISO 9001: 2008 Certified

Phone No. (0257) 2258393, Fax No. (0257) 2258392

Website- www.sscoetjalgaon.ac.in

Email: sscoetjal@gmail.com

ISO 9001:2008

DEPARTMENT OF COMPUTER APPLICATIONS (MC.A.)

MCA
SSBT
COET

Laboratory Journal

Class: M.C.A - I**, Sem - I**

Subject Code: MCA-419(B)

Subject: Lab on Data Science - I

Academic Year: 2024 - 25

University Exam Seat No:

Name of the Faculty: Asst. Prof. Arsalan A. Shaikh

Vision of the Institute

Today we carry the flame of quality education, knowledge and progressive technology for global societal development; tomorrow the flame will glow even brighter.

Mission of the Institute

To provide conducive environment for preparing competent, value added and patriotic engineers of integrity of par excellence to meet global standards for societal development.

DEPARTMENT OF COMPUTER APPLICATIONS (M.C.A.)

Vision of the Department

To develop skilled professionals with social values as per industry needs.

Mission of the Department

To provide a student-centered, conducive environment for preparing knowledgeable, skilled and value addedprofessionals to stand out in a competitive world.

Department of Computer Applications (M.C.A.)

Program Outcomes (POs)

PO1: Computational Knowledge: Understand and apply mathematical foundation, computing and domain knowledge for the conceptualization of computing models from defined problems.

PO2: Problem Analysis: Ability to identify, critically analyze and formulate complex computing problems using fundamentals of computer science and application domains.

PO3: **Design / Development of Solutions:** Ability to transform complex business scenarios and contemporary issues into problems, investigate, understand and propose integrated solutions using emerging technologies.

PO4: Conduct Investigations of Complex Computing Problems: Ability to devise and conduct experiments, interpret data and provide well informed conclusions.

PO5: Modern Tool Usage: Ability to select modern computing tools, skills and techniques necessary for innovative software solutions.

PO6: Professional Ethics: Ability to apply and commit professional ethics and cyber regulations in a global economic environment.

PO7: Life-long Learning: Recognize the need for and develop the ability to engage in continuous learning as a Computing professional.

PO8: Project Management and Finance: Ability to understand, management and computing principles with computing knowledge to manage projects in multidisciplinary environments.

PO9: Communication Efficacy: Communicate effectively with the computing community as well as society by being able to comprehend effective documentations and presentations.

PO10: Societal & Environmental Concern: Ability to recognize economic, environmental, social, health, legal, ethical issues involved in the use of computer technology and other consequential responsibilities relevant to professional practice.

PO11: Individual and Team Work: Ability to work as a member or leader in diverse teams in multidisciplinary environment.

PO12: Innovation and Entrepreneurship: Identify opportunities, entrepreneurship vision and use of innovative ideas to create value and wealth for the betterment of the individual and society.

Department of Computer Applications (M.C.A.)

Program Educational Objectives (PEOs)

PEO 1: Core Knowledge -The computing professional graduate will have the knowledge of basic science and Engineering skills, Humanities, social science, management and conceptual and practical understanding of core computer applications area with project development.

PEO 2: Employment/ Continuing Education – The computing professional graduate will have the knowledge of Industry-based technical skills to succeed in entry level professional position at various industries as well as in academics.

PEO 3: Professional Competency - The computing professional graduate will have the ability to communicate effectively in English, to accumulate and disseminate the knowledge and to work effectively in a team with a sense of social awareness.

Department of Computer Applications (M.C.A.)

Program Specific Outcomes (PSOs)

PSO 1: The computing professional graduate will be able to apply knowledge of computer science in practice to identify, critically analyze, formulate and develop computer applications using modern computing tools and techniques and will use these tools with dexterity.

PSO 2: The computing professional graduate will be able to design computing systems to meet desired needs within realistic constraints such as safety, security and applicability. These systems will function professionally with ethical responsibility as an individual as well as in multidisciplinary teams with positive attitude.

PSO 3: The computing professional graduate will be able to appreciate the importance of goal setting and recognize the need for life-long learning with good communication skills.

Department of Computer Applications (M.C.A.)
MCA-419(B) Lab on Data Science -I

M.C.A-I Sem-I

Course Outcomes:

CO1: Demonstrate proficiency in setting up and using Python data science tools.

CO2: Apply data preprocessing techniques for machine learning.

CO3: Conduct statistical analysis and interpret results.

CO4: Develop and evaluate machine learning models.

CO-PO and PSO Mappings

Course Outcomes	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
CO1:	3	2	3		3		2		3				3	2	3
CO2:	3	3	3		3								3	3	3
CO3:	2	3		3									2	3	
CO4:	2	3	3	3	3						2		2	3	3

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 01

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Setting Up Python for Data Science & Install and configure Anaconda and Jupiter Notebook.

Steps:

Step 1: Download and Install Anaconda

Anaconda is a distribution of Python that includes a variety of data science tools, including Jupyter Notebook, NumPy, Pandas, Matplotlib, and more.

1.1 Download Anaconda:

- Go to the official Anaconda website: Anaconda Downloads
- Click the **Download** button for **Windows** under the "Anaconda Distribution" section.
- Choose the **Python 3.x version** (the latest one is recommended).

1.2 Install Anaconda:

- Locate the downloaded file (`Anaconda3-x.x.x-Windows-x86_64.exe`) in your downloads folder and **double-click** it to start the installation process.
- Click **Next** in the installation wizard.
- Accept the license agreement by selecting **I Agree**.

1.3 Select Installation Type:

- **Choose Installation Type**:
  - For all users: Select the "All Users" option if you have admin rights and want to install it for all users on your computer.
  - For a single user: Select "Just Me" if you only want to install it for yourself.
- Click **Next**.

1.4 Choose Installation Folder:

- You can leave the default path (`C:\Users\<your-username>\Anaconda3`) or choose a different location by clicking **Browse**.
- Click **Next**.

1.5 Advanced Installation Options:

- You will see two checkboxes:
    - **Add Anaconda3 to my PATH environment variable**: It's **not recommended** to check this option because it can cause conflicts with other Python installations.
    - **Register Anaconda3 as my default Python 3.x**: It's recommended to leave this box checked if you want Anaconda to be your default Python.
- Click **Install**.

1.6 Finish Installation:

- Wait for the installation to complete. This may take a few minutes.
- When done, click **Next** and then **Finish**.

Step 2: Open Anaconda Navigator

Anaconda Navigator is a GUI that simplifies managing data science tools.

2.1 Open Anaconda Navigator:

- Go to the **Start Menu** and search for **Anaconda Navigator**.
- Click on **Anaconda Navigator** to open it.

2.2 Explore Anaconda Navigator:

- Once Anaconda Navigator opens, you will see several tools listed, including **Jupyter Notebook**, **Spyder**, **VS Code**, etc.

Step 3: Launch Jupyter Notebook

Jupyter Notebook is an interactive Python environment that runs in your browser.

3.1 Launch Jupyter Notebook:

- In **Anaconda Navigator**, find **Jupyter Notebook** and click on the **Launch** button next to it.
- This will open a Jupyter Notebook in your default web browser.

3.2 Create a New Notebook:

- Once Jupyter Notebook opens, you will see a file browser.
- Navigate to the folder where you want to create a new notebook, or create a new folder by clicking the **New** button in the top-right corner and selecting **Folder**.
- To create a new notebook, click the **New** button in the top-right corner and select **Python 3 (ipykernel)**.

Step 4: Install Additional Python Packages

Although Anaconda comes with most of the required libraries, you may still need to install additional packages for specific projects.

4.1 Install Packages Using Anaconda Navigator:

- Open **Anaconda Navigator** and click on the **Environments** tab on the left panel.
- Select the environment you are working with (the default is `base`).
- Click on the **Search Packages** text box at the bottom, type the package name (e.g., `scikit-learn`, `tensorflow`), and check the box next to it.
- Click the **Apply** button on the bottom-right corner.

4.2 Install Packages Using Conda in the Command Prompt:

- Open the **Anaconda Prompt** (search for it in the Start Menu).
- Type the following command to install a package, for example:

```
conda install scikit-learn
```

- Press **Enter** and wait for the package to install.

4.3 Install Packages Using pip:

If you cannot find a package via Conda, you can install it using `pip`:

- In the **Anaconda Prompt** or **Command Prompt**, type:

```
pip install package-name
```

Step 5: Running Jupyter Notebook for Data Science Projects

Now that everything is set up, you can start running your Data Science projects in Jupyter Notebook.

5.1 Open Jupyter Notebook from Anaconda:

- You can always launch **Jupyter Notebook** from **Anaconda Navigator** or use the Anaconda Prompt:

```
jupyter notebook
```

- Your browser will open with the Jupyter Notebook dashboard.

5.2 Create Notebooks:

- Create new Python notebooks, write your code, and start executing cells.
- You can perform Data Science tasks such as data cleaning with Pandas, visualization with Matplotlib, and modeling with Scikit-learn.

Step 6: Verifying the Installation

To check that your Python environment is ready for Data Science, you can test it by running the following commands in Jupyter Notebook:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

If no errors occur, your setup is complete!

Submitted By

Sign :

Name :

Roll No :

Checked By :

Asst. Prof. Arsalan A. Shaikh

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 02

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Create a Jupiter Notebook and perform basic Python operations & Calculate statistics (mean, median, variance) using NumPy functions.

Code:

```
# Basic Python Operations
print("=== Basic Python Operations ===")

# Arithmetic operations
x = 10
y = 5
print("\nArithmetic Operations:")
print("Addition (x + y):", x + y)
print("Subtraction (x - y):", x - y)
print("Multiplication (x * y):", x * y)
print("Division (x / y):", x / y)
print("Floor Division (x // y):", x // y)
print("Modulus (x % y):", x % y)
print("Exponentiation (x ** y):", x ** y)

# Logical operations
a = True
b = False
print("\nLogical Operations:")
print("Logical AND (a and b):", a and b)
print("Logical OR (a or b):", a or b)
print("Logical NOT (not a):", not a)

# Comparison operations
print("\nComparison Operations:")
print("x > y:", x > y)
print("x < y:", x < y)
print("x == y:", x == y)
print("x != y:", x != y)
print("x >= y:", x >= y)
print("x <= y:", x <= y)

# String operations
str1 = "Hello"
str2 = "World"
```

```
print("\nString Operations:")
print("String Concatenation (str1 + str2):", str1 + " " + str2)
print("String Repetition (str1 * 3):", str1 * 3)

# List operations
lst = [1, 2, 3, 4, 5]
print("\nList Operations:")
print("Original List:", lst)
print("Append 6 to List (lst + [6]):", lst + [6])
print("Slicing List (First 3 elements - lst[:3]):", lst[:3])

# NumPy Statistical Operations
import numpy as np

print("\n=== NumPy Statistical Operations ===")

# Create a NumPy array
data = np.array([10, 20, 30, 40, 50])
print("\nDataset:", data)

# Calculate Mean
mean_value = np.mean(data)
print("Mean of the data:", mean_value)

# Calculate Median
median_value = np.median(data)
print("Median of the data:", median_value)

# Calculate Variance
variance_value = np.var(data)
print("Variance of the data:", variance_value)
```

Output:

```
=== Basic Python Operations ===

Arithmetic Operations:
Addition (x + y): 15
Subtraction (x - y): 5
Multiplication (x * y): 50
Division (x / y): 2.0
Floor Division (x // y): 2
Modulus (x % y): 0
Exponentiation (x ** y): 100000
```

Logical Operations:
Logical AND (a and b): False
Logical OR (a or b): True
Logical NOT (not a): False

Comparison Operations:
x > y: True
x < y: False
x == y: False
x != y: True
x >= y: True
x <= y: False

String Operations:
String Concatenation (str1 + str2): Hello World
String Repetition (str1 * 3): HelloHelloHello

List Operations:
Original List: [1, 2, 3, 4, 5]
Append 6 to List (lst + [6]): [1, 2, 3, 4, 5, 6]
Slicing List (First 3 elements - lst[:3]): [1, 2, 3]

=== NumPy Statistical Operations ===

Dataset: [10 20 30 40 50]
Mean of the data: 30.0
Median of the data: 30.0
Variance of the data: 200.0

Submitted By

Sign :

Name :

Roll No :

Checked By :

Asst. Prof. Arsalan A. Shaikh

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 03

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Load datasets into Pandas Data Frames from CSV files &Perform basic Data Frame operations (e.g., filtering, sorting).

Code:

```
import pandas as pd
# Load dataset from a CSV file
# Replace 'sample_data.csv' with the path to your CSV file
# Sample CSV assumed to have columns: 'Name', 'Age', 'Salary', 'Department'
data = {
   'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eva'],
   'Age': [25, 30, 35, 40, 28],
   'Salary': [50000, 60000, 70000, 80000, 55000],
   'Department': ['HR', 'IT', 'Finance', 'IT', 'HR']
}
df = pd.DataFrame(data)

# 1. View the first few rows of the dataset
print("Dataset Preview:")
print(df.head())

# 2. Information about the dataset
print("\nDataset Information:")
print(df.info())

# 3. Descriptive statistics
print("\nDescriptive Statistics:")
print(df.describe())

# 4. Accessing a specific column
print("\nNames of Employees:")
print(df['Name'])

# 5. Filtering data - employees with Salary > 60000
print("\nEmployees with Salary > 60000:")
high_salary = df[df['Salary'] > 60000]
print(high_salary)

# 6. Sorting data - by Age in ascending order
print("\nDataset sorted by Age:")
```

```
sorted_by_age = df.sort_values(by='Age')
print(sorted_by_age)

# 7. Adding a new column - Bonus (10% of Salary)
df['Bonus'] = df['Salary'] * 0.10
print("\nDataset after Adding Bonus Column:")
print(df)

# 8. Dropping a column - 'Bonus'
df_without_bonus = df.drop(columns=['Bonus'])
print("\nDataset after Dropping the Bonus Column:")
print(df_without_bonus)

# 9. Grouping data - Average Salary by Department
print("\nAverage Salary by Department:")
avg_salary_dept = df.groupby('Department')['Salary'].mean()
print(avg_salary_dept)

# 10. Checking for missing data
print("\nChecking for Missing Values:")
print(df.isnull().sum())
```

Output:

```
Dataset Preview:
      Name  Age  Salary Department
0    Alice   25   50000         HR
1      Bob   30   60000         IT
2  Charlie   35   70000    Finance
3    David   40   80000         IT
4      Eva   28   55000         HR

Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5 entries, 0 to 4
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Name        5 non-null      object
 1   Age         5 non-null      int64
 2   Salary      5 non-null      int64
 3   Department  5 non-null      object
dtypes: int64(2), object(2)
memory usage: 288.0 bytes
```

Descriptive Statistics:

	Age	Salary
count	5.000000	5.000000
mean	31.600000	63000.000000
std	6.658328	12041.594578
min	25.000000	50000.000000
25%	28.000000	55000.000000
50%	30.000000	60000.000000
75%	35.000000	70000.000000
max	40.000000	80000.000000

Names of Employees:

```
0      Alice
1        Bob
2    Charlie
3      David
4        Eva
Name: Name, dtype: object
```

Employees with Salary > 60000:

	Name	Age	Salary	Department
2	Charlie	35	70000	Finance
3	David	40	80000	IT

Dataset sorted by Age:

	Name	Age	Salary	Department
0	Alice	25	50000	HR
4	Eva	28	55000	HR
1	Bob	30	60000	IT
2	Charlie	35	70000	Finance
3	David	40	80000	IT

Dataset after Adding Bonus Column:

	Name	Age	Salary	Department	Bonus
0	Alice	25	50000	HR	5000.0
1	Bob	30	60000	IT	6000.0
2	Charlie	35	70000	Finance	7000.0
3	David	40	80000	IT	8000.0
4	Eva	28	55000	HR	5500.0

Dataset after Dropping the Bonus Column:

	Name	Age	Salary	Department
0	Alice	25	50000	HR
1	Bob	30	60000	IT
2	Charlie	35	70000	Finance
3	David	40	80000	IT
4	Eva	28	55000	HR

Average Salary by Department:
Department
Finance 70000.0
HR 52500.0
IT 70000.0
Name: Salary, dtype: float64

Checking for Missing Values:
Name 0
Age 0
Salary 0
Department 0
Bonus 0

dtype: int64

Submitted By

Sign :

Name :

Roll No :

Checked By :

Asst. Prof. Arsalan A. Shaikh

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 04

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Write a program to implement Normalize and standardize numerical data using Scikit-learn.

Code:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Define the dataset
data = {
    'Age': [25, 30, 35, 40, 45],
    'Salary': [50000, 60000, 70000, 80000, 90000],
    'Experience': [1, 2, 3, 4, 5]
}

df = pd.DataFrame(data)
print("=== Original Data ===")
print(df)

# Basic statistics
print("\n=== Basic Statistics ===")
print("Mean:\n", df.mean())
print("Standard Deviation:\n", df.std())
print("Minimum:\n", df.min())
print("Maximum:\n", df.max())

# Normalization (Min-Max Scaling)
min_max_scaler = MinMaxScaler()
normalized_data = min_max_scaler.fit_transform(df)
normalized_df = pd.DataFrame(normalized_data, columns=df.columns)
print("\n=== Normalized Data (Min-Max Scaling) ===")
print(normalized_df)

# Standardization (Z-score Scaling)
standard_scaler = StandardScaler()
standardized_data = standard_scaler.fit_transform(df)
standardized_df = pd.DataFrame(standardized_data, columns=df.columns)
print("\n=== Standardized Data (Z-score Scaling) ===")
print(standardized_df)
```

```
# Dataset with outliers
data_with_outlier = {
    'Age': [25, 30, 35, 40, 150],
    'Salary': [50000, 60000, 70000, 80000, 1000000],
    'Experience': [1, 2, 3, 4, 50]
}
df_outlier = pd.DataFrame(data_with_outlier)
print("\n=== Dataset with Outliers ===")
print(df_outlier)

# Normalized and standardized data with outliers
normalized_data_outlier = min_max_scaler.fit_transform(df_outlier)
standardized_data_outlier = standard_scaler.fit_transform(df_outlier)

normalized_outlier_df = pd.DataFrame(normalized_data_outlier, columns=df_outlier.columns)
standardized_outlier_df = pd.DataFrame(standardized_data_outlier,
columns=df_outlier.columns)

print("\n=== Normalized Data (With Outliers) ===")
print(normalized_outlier_df)

print("\n=== Standardized Data (With Outliers) ===")
print(standardized_outlier_df)
```

Output:

```
=== Original Data ===
   Age  Salary  Experience
0   25   50000           1
1   30   60000           2
2   35   70000           3
3   40   80000           4
4   45   90000           5

=== Basic Statistics ===
Mean:
Age           35.0
Salary     70000.0
Experience     3.0
dtype: float64
```

Standard Deviation:
Age 7.905694
Salary 15811.388301
Experience 1.581139
dtype: float64

Minimum:
Age 25
Salary 50000
Experience 1
dtype: int64

Maximum:
Age 45
Salary 90000
Experience 5
dtype: int64

=== Normalized Data (Min-Max Scaling) ===

	Age	Salary	Experience
0	0.00	0.0	0.0
1	0.25	0.25	0.25
2	0.50	0.50	0.50
3	0.75	0.75	0.75
4	1.00	1.0	1.0

=== Standardized Data (Z-score Scaling) ===

	Age	Salary	Experience
0	-1.4142	-1.4142	-1.4142
1	-0.7071	-0.7071	-0.7071
2	0.0000	0.0000	0.0000
3	0.7071	0.7071	0.7071
4	1.4142	1.4142	1.4142

=== Dataset with Outliers ===

	Age	Salary	Experience
0	25	50000	1
1	30	60000	2
2	35	70000	3
3	40	80000	4
4	150	1000000	50

=== Normalized Data (With Outliers) ===

	Age	Salary	Experience
0	0.000000	0.000000	0.000000
1	0.034483	0.010101	0.020408
2	0.068966	0.020202	0.040816
3	0.103448	0.030303	0.061224
4	1.000000	1.000000	1.000000

=== Standardized Data (With Outliers) ===

	Age	Salary	Experience
0	-0.824163	-0.828046	-0.805386
1	-0.796210	-0.821711	-0.781662
2	-0.768258	-0.815376	-0.757939
3	-0.740305	-0.809041	-0.734215
4	3.129936	4.274174	4.079201

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Arsalan A. Shaikh**

Roll No :

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 05

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Write a program to implement Encode categorical variables using One-Hot Encoding and Label.

Code:

```
# Import required libraries
import pandas as pd
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

# Define a dataset with categorical variables
data = {
    'Color': ['Red', 'Blue', 'Green', 'Blue', 'Green', 'Red'],
    'Size': ['S', 'M', 'L', 'M', 'S', 'L']
}

# Create a DataFrame
df = pd.DataFrame(data)
print("=== Original DataFrame ===")
print(df)

# 1. Label Encoding
label_encoder_color = LabelEncoder()
label_encoder_size = LabelEncoder()

# Apply Label Encoding to 'Color' and 'Size'
df['Color_Label'] = label_encoder_color.fit_transform(df['Color'])
df['Size_Label'] = label_encoder_size.fit_transform(df['Size'])

print("\n=== DataFrame after Label Encoding ===")
print(df)

# Label Encoding Mappings
color_mapping = dict(zip(label_encoder_color.classes_,
label_encoder_color.transform(label_encoder_color.classes_)))
size_mapping = dict(zip(label_encoder_size.classes_,
label_encoder_size.transform(label_encoder_size.classes_)))
print("\n=== Label Encoding Mapping for 'Color' ===")
print(color_mapping)
print("\n=== Label Encoding Mapping for 'Size' ===")
print(size_mapping)
```

```
# 2. One-Hot Encoding using Pandas
color_one_hot = pd.get_dummies(df['Color'], prefix='Color')
size_one_hot = pd.get_dummies(df['Size'], prefix='Size')
df_one_hot = pd.concat([df, color_one_hot, size_one_hot], axis=1)
print("\n=== DataFrame after One-Hot Encoding (Using Pandas) ===")
print(df_one_hot)

# 3. One-Hot Encoding using Scikit-learn
one_hot_encoder = OneHotEncoder(sparse=False)
color_encoded = one_hot_encoder.fit_transform(df[['Color']])
color_encoded_df = pd.DataFrame(color_encoded,
columns=one_hot_encoder.get_feature_names_out(['Color']))
size_encoded = one_hot_encoder.fit_transform(df[['Size']])
size_encoded_df = pd.DataFrame(size_encoded,
columns=one_hot_encoder.get_feature_names_out(['Size']))
df_one_hot_sklearn = pd.concat([df, color_encoded_df, size_encoded_df], axis=1)
print("\n=== DataFrame with One-Hot Encoding (Using Scikit-learn) ===")
print(df_one_hot_sklearn)
```

Output:

```
Original DataFrame:
   Color Size
0    Red    S
1   Blue    M
2  Green    L
3   Blue    M
4  Green    S
5    Red    L

DataFrame after Label Encoding:
   Color    Size  Color_Label  Size_Label
0    Red     S            2             2
1    Blue    M            0             1
2   Green    L            1             0
3   Blue     M            0             1
4   Green    S            1             2
5    Red     L            2             0

Label Encoding Mapping for 'Color':
{'Blue': 0, 'Green': 1, 'Red': 2}

Label Encoding Mapping for 'Size':
{'L': 0, 'M': 1, 'S': 2}
```

DataFrame after One-Hot Encoding (Using Pandas):

	Color	Size	Color_Label	Size_Label	Color_Blue	Color_Green	Color_Red	Size_L	Size_M	Size_S
0	Red	S	2	2	0	0	1	0	0	1
1	Blue	M	0	1	1	0	0	0	1	0
2	Green	L	1	0	0	1	0	1	0	0
3	Blue	M	0	1	1	0	0	0	1	0
4	Green	S	1	2	0	1	0	0	0	1
5	Red	L	2	0	0	0	1	1	0	0

DataFrame with One-Hot Encoding (Using Scikit-learn):

	Color	Size	Color_Label	Size_Label	Color_Blue	Color_Green	Color_Red	Size_L	Size_M	Size_S
0	Red	S	2	2	0.0	0.0	1.0	0.0	0.0	1.0
1	Blue	M	0	1	1.0	0.0	0.0	0.0	1.0	0.0
2	Green	L	1	0	0.0	1.0	0.0	1.0	0.0	0.0
3	Blue	M	0	1	1.0	0.0	0.0	0.0	1.0	0.0
4	Green	S	1	2	0.0	1.0	0.0	0.0	0.0	1.0
5	Red	L	2	0	0.0	0.0	1.0	1.0	0.0	0.0

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Arsalan A. Shaikh**

Roll No :

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 06

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

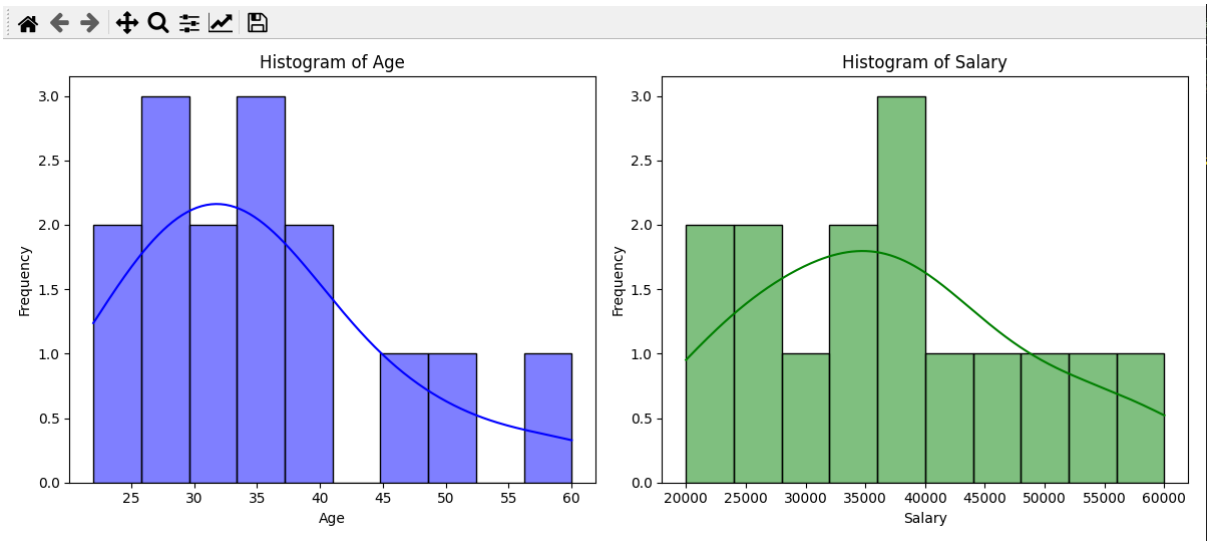
Title: Write a program to Plot histograms to visualize data distribution and calculate skewness and kurtosis of given dataset.

Code:

```
# Importing required libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import skew, kurtosis

# Sample dataset - Create a DataFrame
data = {
    'Age': [23, 26, 30, 22, 29, 35, 32, 40, 34, 36, 38, 45, 50, 60, 28],
    'Salary': [20000, 25000, 30000, 22000, 27000, 40000, 35000, 45000, 37000, 38000, 39000,
50000,
            55000, 60000, 32000]
}
df = pd.DataFrame(data)

# Step 1: Plot histograms to visualize data distribution
plt.figure(figsize=(12, 5))

# Histogram for 'Age'
plt.subplot(1, 2, 1)
sns.histplot(df['Age'], bins=10, kde=True, color='blue')
plt.title('Histogram of Age')
plt.xlabel('Age')
plt.ylabel('Frequency')

# Histogram for 'Salary'
plt.subplot(1, 2, 2)
sns.histplot(df['Salary'], bins=10, kde=True, color='green')
plt.title('Histogram of Salary')
plt.xlabel('Salary')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

```
# Step 2: Calculate skewness and kurtosis
age_skewness = skew(df['Age'])
salary_skewness = skew(df['Salary'])
age_kurtosis = kurtosis(df['Age'])
salary_kurtosis = kurtosis(df['Salary'])

# Print skewness and kurtosis values
print("Skewness and Kurtosis of the Dataset:")
print(f"Skewness of Age: {age_skewness:.4f}")
print(f"Skewness of Salary: {salary_skewness:.4f}")
print(f"Kurtosis of Age: {age_kurtosis:.4f}")
print(f"Kurtosis of Salary: {salary_kurtosis:.4f}")
```

Output:




```
Skewness and Kurtosis of the Dataset:
Skewness of Age: 0.9273
Skewness of Salary: 0.4207
Kurtosis of Age: 0.3455
Kurtosis of Salary: -0.6568
```

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Arsalan A. Shaikh**

Roll No :

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 07

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Write a program to Perform a t-test (one-sample, independent) on sample data and interpret the results.

Code:

```
from scipy import stats import numpy as np

# One-sample t-test
# Sample data: Let's say we have test scores of a group and we want to test if the mean score is significantly different from a hypothetical population mean.
sample_data_one = [85, 89, 78, 92, 88, 91, 84, 90] population_mean = 80 # Hypothetical population mean

# Perform the one-sample t-test
t_stat_one, p_value_one = stats.ttest_1samp(sample_data_one, population_mean) print("One-sample t-test:")
print(f"T-statistic: {t_stat_one:.2f}, P-value: {p_value_one:.3f}")

# Interpret results for one-sample t-test alpha = 0.05 # Significance level
if p_value_one < alpha:
print("Result: Reject the null hypothesis. The sample mean is significantly different from the population mean.")
else:
print("Result: Fail to reject the null hypothesis. The sample mean is not significantly different from the population mean.")

# Independent (two-sample) t-test
# Sample data for two groups (e.g., scores of students from two different classes) class_a_scores = [85, 88, 78, 92, 88, 91, 84, 90]
class_b_scores = [82, 86, 80, 85, 87, 79, 83, 84]

# Perform the independent (two-sample) t-test
t_stat_ind, p_value_ind = stats.ttest_ind(class_a_scores, class_b_scores) print("\nIndependent (two-sample) t-test:")
print(f"T-statistic: {t_stat_ind:.2f}, P-value: {p_value_ind:.3f}")

# Interpret results for independent (two-sample) t-test if p_value_ind < alpha:
print("Result: Reject the null hypothesis. There is a significant difference between the two groups.")
else:
```

```
print("Result: Fail to reject the null hypothesis. There is no significant difference between the two groups.")
```

Output:

```
One-sample t-test:
T-statistic: 5.16, P-value: 0.002
Result: Reject the null hypothesis. The sample mean is significantly different from the population mean.

Independent (two-sample) t-test:
T-statistic: 1.53, P-value: 0.142
Result: Fail to reject the null hypothesis. There is no significant difference between the two groups.
```

Submitted By

Sign :

Name :

Roll No :

Checked By :

Asst. Prof. Arsalan A. Shaikh

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 08

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Write a program to Calculate and interpret the Pearson and Spearman correlation coefficients.

Code:

```
# Import necessary libraries
import numpy as np
from scipy.stats import pearsonr, spearmanr

# **DATA PREPARATION**
print("Data Preparation")
# Example data (e.g., study hours and exam scores)
x = [10, 12, 13, 15, 16, 18, 20]  # Hours studied
y = [85, 87, 88, 90, 92, 95, 98]  # Exam scores

print(f"Variable X (Hours Studied): {x}")
print(f"Variable Y (Exam Scores): {y}")
print("\n" + "-"*50 + "\n")

# **PEARSON CORRELATION COEFFICIENT**
print("Pearson Correlation Coefficient")
# Calculate Pearson correlation
pearson_corr, pearson_p_value = pearsonr(x, y)

# Results
print(f"Pearson Correlation Coefficient: {pearson_corr:.3f}")
print(f"P-Value: {pearson_p_value:.3f}")

# Interpretation
if pearson_corr > 0:
   print("Positive correlation: As X increases, Y tends to increase.")
elif pearson_corr < 0:
   print("Negative correlation: As X increases, Y tends to decrease.")
else:
   print("No correlation: X and Y are not linearly related.")

if pearson_p_value < 0.05:
   print("The correlation is statistically significant.")
else:
   print("The correlation is not statistically significant.")
```

```
print("\n" + "-"*50 + "\n")

# **SPEARMAN CORRELATION COEFFICIENT**
print("Spearman Correlation Coefficient")
# Calculate Spearman correlation
spearman_corr, spearman_p_value = spearmanr(x, y)

# Results
print(f"Spearman Correlation Coefficient: {spearman_corr:.3f}")
print(f"P-Value: {spearman_p_value:.3f}")

# Interpretation
if spearman_corr > 0:
    print("Positive monotonic relationship: As X increases, Y tends to increase.")
elif spearman_corr < 0:
    print("Negative monotonic relationship: As X increases, Y tends to decrease.")
else:
    print("No monotonic relationship between X and Y.")

if spearman_p_value < 0.05:
    print("The monotonic relationship is statistically significant.")
else:
    print("The monotonic relationship is not statistically significant.")
```

Output:

```
 Data Preparation
Variable X (Hours Studied): [10, 12, 13, 15, 16, 18, 20]
Variable Y (Exam Scores): [85, 87, 88, 90, 92, 95, 98]

--------------------------------------------------

Pearson Correlation Coefficient
Pearson Correlation Coefficient: 0.993
P-Value: 0.000
Positive correlation: As X increases, Y tends to increase.
The correlation is statistically significant.

--------------------------------------------------

Spearman Correlation Coefficient
Spearman Correlation Coefficient: 1.000
P-Value: 0.000
Positive monotonic relationship: As X increases, Y tends to increase.
The monotonic relationship is statistically significant.
```

Submitted By

Sign :

Name :

Roll No :

Checked By :

Asst. Prof. Arsalan A. Shaikh

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 09

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Write a program to implement Conduct ANOVA (One-Way and Two-Way) test.

Code:

```
# Import necessary libraries
import numpy as np
import pandas as pd
from scipy.stats import f_oneway
from statsmodels.stats.anova import AnovaRM
from statsmodels.formula.api import ols
import statsmodels.api as sm

# **ONE-WAY ANOVA**

# Data for three groups (e.g., exam scores from three teaching methods)
group1 = [85, 86, 88, 75, 78]
group2 = [89, 92, 91, 90, 87]
group3 = [82, 80, 85, 86, 84]

print("One-Way ANOVA")
print("-" * 50)

# Step 1: Display data
print("Group 1:", group1)
print("Group 2:", group2)
print("Group 3:", group3)

# Step 2: Perform One-Way ANOVA
f_statistic, p_value = f_oneway(group1, group2, group3)

# Step 3: Print Results
print(f"\nF-Statistic: {f_statistic:.3f}")
print(f"P-Value: {p_value:.3f}")

# Step 4: Interpret Results
alpha = 0.05  # Significance level
if p_value < alpha:
    print("Reject the null hypothesis: There is a significant difference between the group means.")
else:
    print("Fail to reject the null hypothesis: No significant difference between the group means.")
```

```
print("\n" + "=" * 50 + "\n")

# **TWO-WAY ANOVA**

# Step 1: Create a dataset for two-way ANOVA
# Example: Two factors - Teaching Method (A, B) and Gender (Male, Female)
data = {
    "Scores": [85, 86, 88, 75, 78, 89, 92, 91, 90, 87, 82, 80, 85, 86, 84],
    "Teaching_Method": ["A"] * 5 + ["B"] * 5 + ["C"] * 5,
    "Gender": ["Male", "Male", "Female", "Female", "Male", "Female", "Male", "Male",
"Female", "Female", "Male", "Female", "Male", "Male", "Female"]
}

# Step 2: Convert data to a pandas DataFrame
df = pd.DataFrame(data)

# Display the dataset
print("Two-Way ANOVA Dataset")
print(df)
print("\n" + "-" * 50 + "\n")

# Step 3: Perform Two-Way ANOVA using statsmodels
# Fit the model
model = ols('Scores ~ C(Teaching_Method) + C(Gender) + C(Teaching_Method):C(Gender)',
data=df).fit()

# Perform ANOVA
anova_results = sm.stats.anova_lm(model, typ=2)

# Step 4: Display Results
print("Two-Way ANOVA Results")
print(anova_results)

# Step 5: Interpret Results
print("\nInterpretation:")
for effect, p_value in anova_results["PR(>F)"].items():
    if p_value < alpha:
        print(f"The effect of {effect} is statistically significant (p-value: {p_value:.3f}).")
    else:
        print(f"The effect of {effect} is not statistically significant (p-value: {p_value:.3f}).")
```

Output:

One-Way ANOVA

Group 1: [85, 86, 88, 75, 78]
Group 2: [89, 92, 91, 90, 87]
Group 3: [82, 80, 85, 86, 84]

F-Statistic: 5.926
P-Value: 0.016
Reject the null hypothesis: There is a significant difference between the group means.

==========

Two-Way ANOVA Dataset

	Scores	Teaching_Method	Gender
0	85	A	Male
1	86	A	Male
2	88	A	Female
3	75	A	Female
4	78	A	Male
5	89	B	Female
6	92	B	Male
7	91	B	Male
8	90	B	Female
9	87	B	Female
10	82	C	Male
11	80	C	Female
12	85	C	Male
13	86	C	Male
14	84	C	Female

Two-Way ANOVA Results

	sum_sq	df	F	PR(>F)
C(Teaching_Method)	175.881349	2.0	5.483599	0.027712
C(Gender)	17.777778	1.0	1.108545	0.319844
C(Teaching_Method):C(Gender)	1.088889	2.0	0.033949	0.966744
Residual	144.333333	9.0	NaN	NaN

Interpretation:
The effect of C(Teaching_Method) is statistically significant (p-value: 0.028).
The effect of C(Gender) is not statistically significant (p-value: 0.320).
The effect of C(Teaching_Method):C(Gender) is not statistically significant (p-value: 0.967).
The effect of Residual is not statistically significant (p-value: nan).

Submitted By

Sign :

Name :

Roll No :

Checked By :

Asst. Prof. Arsalan A. Shaikh

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 10

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Write a program to implement simple linear regression, Multiple Linear Regression on given dataset.

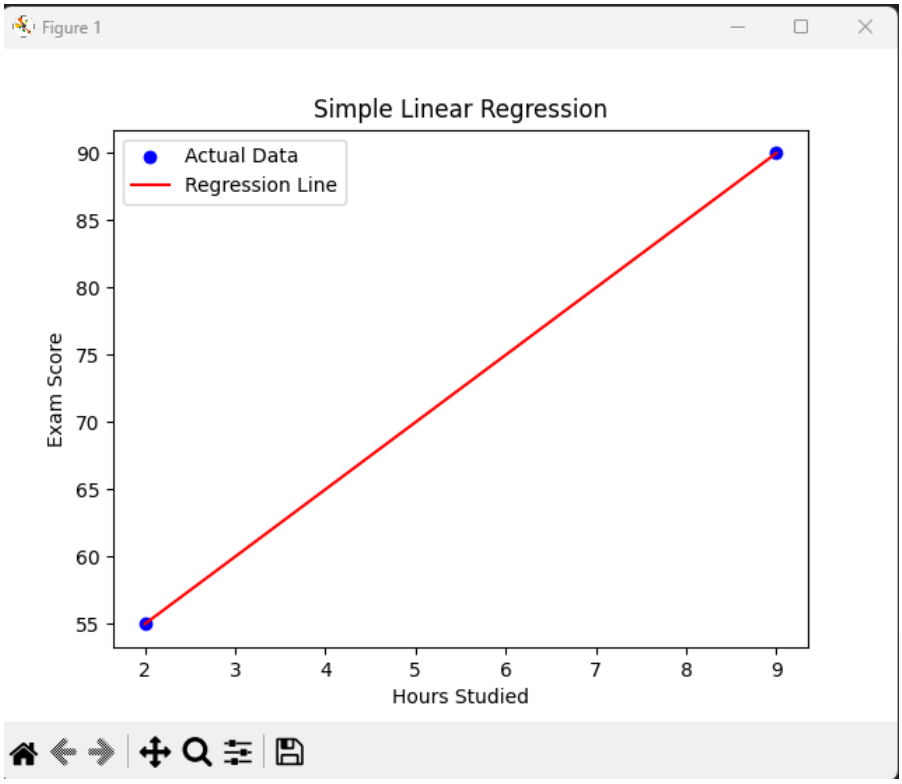
Code:

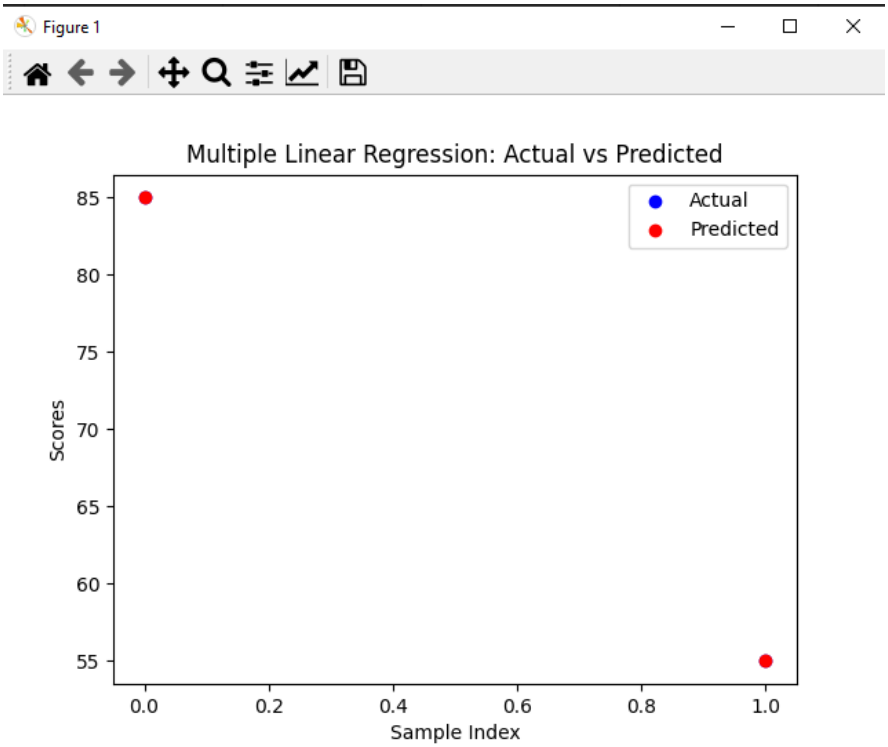
```
# Step 1: Import Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Step 2: Create a Sample Dataset
# Example dataset for both Simple and Multiple Linear Regression
data = {
    "Hours_Studied": [2, 3, 4, 5, 6, 7, 8, 9, 10],
    "Online_Tutorials": [1, 2, 1, 3, 2, 3, 4, 5, 5],
    "Scores": [50, 55, 60, 65, 70, 75, 80, 85, 90]
}

# Convert data to DataFrame
df = pd.DataFrame(data)

# Display the dataset
print("Dataset:")
print(df)

# Step 3: Separate Dependent (Y) and Independent Variables (X)
X_simple = df[["Hours_Studied"]]  # For Simple Linear Regression
X_multiple = df[["Hours_Studied", "Online_Tutorials"]]  # For Multiple Linear Regression
Y = df["Scores"]

# Step 4: Split the Dataset into Training and Testing Sets
# For Simple Regression
X_train_simple, X_test_simple, Y_train, Y_test = train_test_split(X_simple, Y, test_size=0.2,
random_state=42)

# For Multiple Regression
X_train_multiple, X_test_multiple, Y_train_multiple, Y_test_multiple =
```

```
train_test_split(X_multiple, Y, test_size=0.2, random_state=42)

# **Simple Linear Regression**
print("\nSimple Linear Regression")
print("-" * 50)

# Step 5: Train the Model
model_simple = LinearRegression()
model_simple.fit(X_train_simple, Y_train)

# Step 6: Test the Model
Y_pred_simple = model_simple.predict(X_test_simple)

# Step 7: Print the Coefficient and Intercept
print(f"Coefficient (Slope): {model_simple.coef_[0]:.2f}")
print(f"Intercept: {model_simple.intercept_:.2f}")

# Step 8: Calculate Performance Metrics
mse_simple = mean_squared_error(Y_test, Y_pred_simple)
r2_simple = r2_score(Y_test, Y_pred_simple)

print(f"Mean Squared Error: {mse_simple:.2f}")
print(f"R-Squared Value: {r2_simple:.2f}")

# Step 9: Visualize Results
plt.scatter(X_test_simple, Y_test, color="blue", label="Actual")
plt.plot(X_test_simple, Y_pred_simple, color="red", label="Predicted (Line of Best Fit)")
plt.xlabel("Hours Studied")
plt.ylabel("Scores")
plt.title("Simple Linear Regression")
plt.legend()
plt.show()

# **Multiple Linear Regression**
print("\nMultiple Linear Regression")
print("-" * 50)

# Step 10: Train the Model
model_multiple = LinearRegression()
model_multiple.fit(X_train_multiple, Y_train_multiple)

# Step 11: Test the Model
Y_pred_multiple = model_multiple.predict(X_test_multiple)

# Step 12: Print Coefficients and Intercept
print(f"Coefficients (Slopes): {model_multiple.coef_}")
print(f"Intercept: {model_multiple.intercept_:.2f}")

# Step 13: Calculate Performance Metrics
mse_multiple = mean_squared_error(Y_test_multiple, Y_pred_multiple)
```

```
r2_multiple = r2_score(Y_test_multiple, Y_pred_multiple)

print(f"Mean Squared Error: {mse_multiple:.2f}")
print(f"R-Squared Value: {r2_multiple:.2f}")

# Step 14: Visualize Results
# Since multiple regression involves multiple dimensions, we'll display actual vs predicted values
plt.scatter(range(len(Y_test_multiple)), Y_test_multiple, color="blue", label="Actual")
plt.scatter(range(len(Y_test_multiple)), Y_pred_multiple, color="red", label="Predicted")
plt.xlabel("Sample Index")
plt.ylabel("Scores")
plt.title("Multiple Linear Regression: Actual vs Predicted")
plt.legend()
plt.show()
```

Output:





Dataset:

	Hours_Studied	Online_Tutorials	Scores
0	2	1	50
1	3	2	55
2	4	1	60
3	5	3	65
4	6	2	70
5	7	3	75
6	8	4	80
7	9	5	85
8	10	5	90

Simple Linear Regression

Coefficient (Slope): 5.00
Intercept: 40.00
Mean Squared Error: 0.00
R-Squared Value: 1.00

Multiple Linear Regression

Coefficients (Slopes): [5. 0.]
Intercept: 40.00
Mean Squared Error: 0.00
R-Squared Value: 1.00

Submitted By

Sign :

Name :

Roll No :

Checked By :

Asst. Prof. Arsalan A. Shaikh

SSBT's College of Engineering & Technology, Bambhori,

JalgaonDepartment of Computer Applications

Practical: 11

DATE OF PERFORMANCE: **DATE OF COMPLETION:**

Title: Setting Up Python for Data Science & Install and configure Anaconda and Jupiter Notebook.

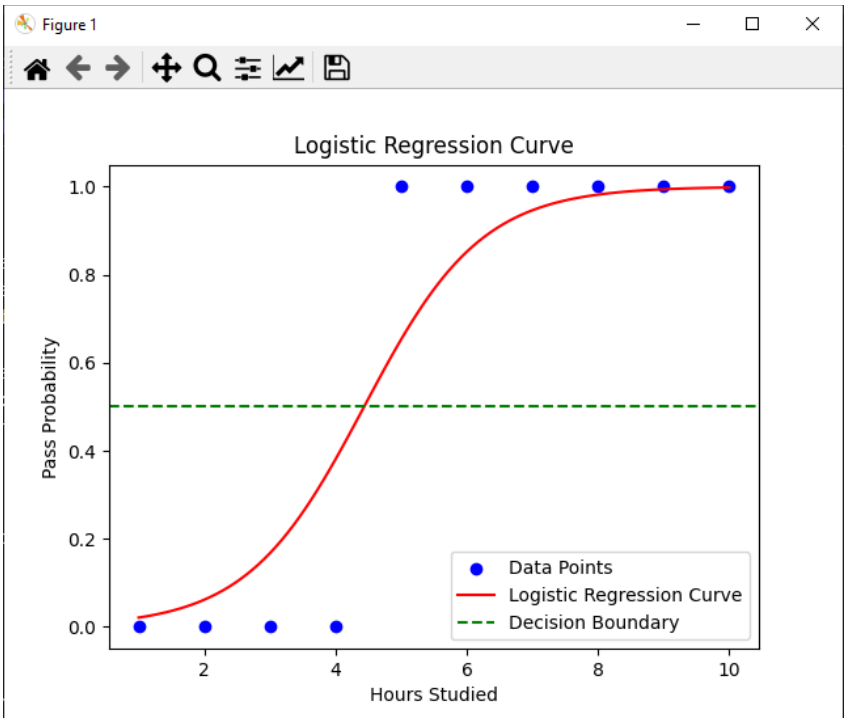
Code:

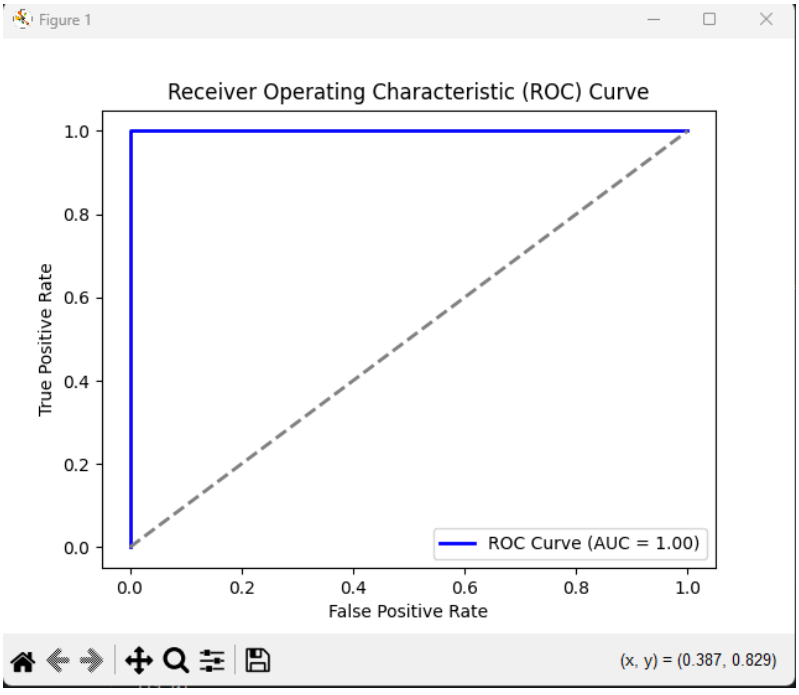
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_curve, auc

# ============================
# Example Dataset
# ============================
# Creating a sample dataset
data = {
'Study_Hours': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
'Pass_Exam': [0, 0, 0, 0, 1, 1, 1, 1, 1, 1] # Binary output: 0 -> Fail, 1 -> Pass
}

# Convert to DataFrame
df = pd.DataFrame(data)

# ============================
# Split Dataset into Features and Target # ============================
# Features and target variable
X = df[['Study_Hours']] # Independent variable
y = df['Pass_Exam'] # Dependent variable (binary classification)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ============================
# Logistic Regression Model
# ============================
# Initialize and train the model
logistic_model = LogisticRegression()
```

```
logistic_model.fit(X_train, y_train)

# Predict on test data
y_pred = logistic_model.predict(X_test)
y_prob = logistic_model.predict_proba(X_test)[:, 1] # Probabilities for class 1 #
==============================
# Model Evaluation
# ==============================

print("=== Logistic Regression Evaluation ===") # Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
print(f"Confusion Matrix:\n{cm}")

# Classification Report
cr = classification_report(y_test, y_pred)
print(f"Classification Report:\n{cr}")

# ==============================
# Visualization
# ==============================
# Scatter plot of data
plt.scatter(X, y, color='blue', label='Actual Data')

# Line plot of predicted probabilities
x_vals = np.linspace(X.min(), X.max(), 100).reshape(-1, 1)
y_vals = logistic_model.predict_proba(x_vals)[:, 1]
plt.plot(x_vals, y_vals, color='red', label='Logistic Curve')

plt.xlabel("Study Hours")
plt.ylabel("Probability of Passing")
plt.title("Logistic Regression Curve")
plt.legend()
plt.show()

# ==============================
# ROC Curve and AUC
# ==============================
fpr, tpr, _ = roc_curve(y_test, y_prob)
roc_auc = auc(fpr, tpr)

# Plot ROC Curve plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label=f'ROC Curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--', lw=2)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
```

```
plt.legend(loc='lower right')
plt.show(
```

Output:





=== Logistic Regression Evaluation ===
Accuracy: 1.00
Confusion Matrix:
[[1 0]
 [0 1]]
Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	1
1	1.00	1.00	1.00	1
accuracy		1.00	2	
macro avg	1.00	1.00	1.00	2
weighted avg	1.00	1.00	1.00	2

Submitted By **Checked By :**

Sign :

Name : **Asst. Prof. Arsalan A. Shaikh**

Roll No :