Experiment No: 3

Date:

AIM: Study of Basics of Python data types, NumPy, Matplotlib, Pandas.

Relevant CO: CO1, CO2

Objective:
The objective of this lab practical is to gain hands-on experience with NumPy, Matplotlib, and Pandas libraries to manipulate and visualize data. Through this practical, students will learn how to use different functions of these libraries to perform various data analysis tasks.

Materials Used:
- Python programming environment
- NumPy library
- Matplotlib library
- Pandas library
- Dataset file (provided by faculty)

//Example of dataset file like sales_Data.csv

- Date: Date of sale
- Product: Name of the product sold
- Units Sold: Number of units sold
- Revenue: Total revenue generated from the sale
- Region: Geographic region where the sale took place
- Salesperson: Name of the salesperson who made the sale

Procedures:

Part 1: NumPy
1. Import the NumPy library into Python.
2. Create a NumPy array with the following specifications:
   a. Dimensions: 5x5
   b. Data type: integer
   c. Values: random integers between 1 and 100
3. Reshape the array into a 1x25 array and calculate the mean, median, variance, and standard deviation using NumPy functions.
4. Generate a random integer array of length 10 and find the percentile, decile, and quartile values using NumPy functions.

Part 2: Matplotlib
1. Import the Matplotlib library into Python.
2. Create a simple bar chart using the following data:
   a. X-axis values: ['A', 'B', 'C', 'D']
   b. Y-axis values: [10, 20, 30, 40]
3. Customize the plot by adding a title, axis labels, and changing the color and style of the bars.
4. Create a pie chart using the following data:
   a. Labels: ['Red', 'Blue', 'Green', 'Yellow']
   b. Values: [20, 30, 10, 40]
5. Customize the pie chart by adding a title, changing the colors of the slices, and adding a legend.

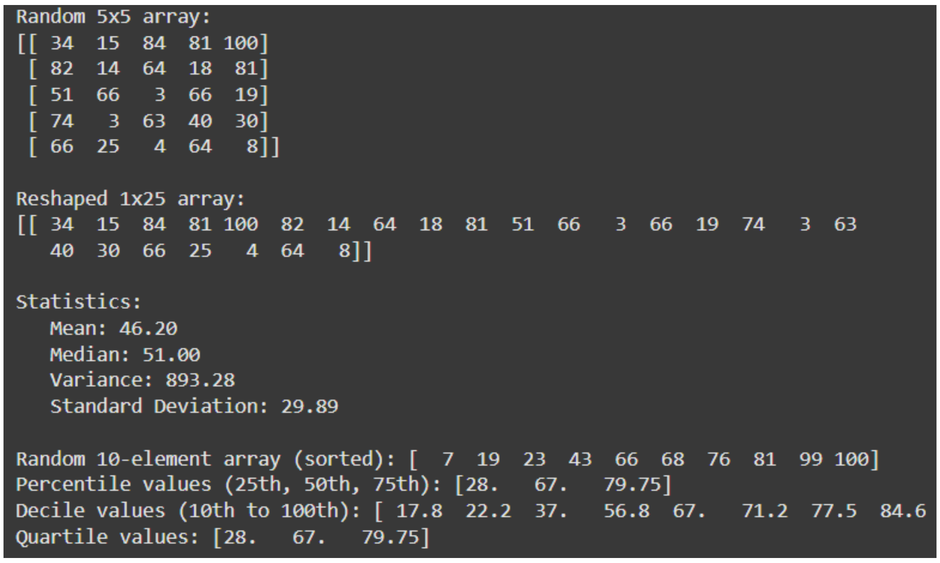
Part 3: Pandas

1. Import the Pandas library into Python.
2. Load the " Ecommerce_Sales_Prediction_Dataset.csv" file into a Pandas data frame.
3. Calculate the following statistics for the Units Sold and Revenue columns:
   a. Mean
   b. Median
   c. Variance
   d. Standard deviation
4. Group the data frame by Product and calculate the mean, median, variance, and standard deviation of Units Sold and Revenue for each product using Pandas functions.
5. Create a line chart to visualize the trend of Units Sold and Revenue over time for each product.

Interpretation/Program/code:

```
import numpy as np
# Part 1: NumPy Operations
#
print("="*40)
print("PART 1: NumPy Operations")
print("="*40)

# 2. Create a NumPy array (5x5, int, random 1-100)
random_array = np.random.randint(1, 101, (5, 5), dtype=int)
print(f"Random 5x5 array:\n{random_array}")

# 3. Reshape and calculate statistics
reshaped_array = random_array.reshape(1, 25)
mean_value = np.mean(reshaped_array)
median_value = np.median(reshaped_array)
variance_value = np.var(reshaped_array)
std_deviation_value = np.std(reshaped_array)

print(f"\nReshaped 1x25 array:\n{reshaped_array}")
print(f"\nStatistics:")
print(f"   Mean: {mean_value:.2f}")
print(f"   Median: {median_value:.2f}")
print(f"   Variance: {variance_value:.2f}")
print(f"   Standard Deviation: {std_deviation_value:.2f}")

# 4. Generate random integer array of length 10 and find statistics
random_integers = np.random.randint(1, 101, 10, dtype=int)
random_integers.sort()
deciles_to_find = np.arange(10, 101, 10)

percentile_values = np.percentile(random_integers, [25, 50, 75])
decile_values = np.percentile(random_integers, deciles_to_find)
quartile_values = np.percentile(random_integers, [25, 50, 75])

print(f"\nRandom 10-element array (sorted): {random_integers}")
print(f"Percentile values (25th, 50th, 75th): {percentile_values}")
print(f"Decile values (10th to 100th): {decile_values}")
print(f"Quartile values: {quartile_values}")
```

Output:



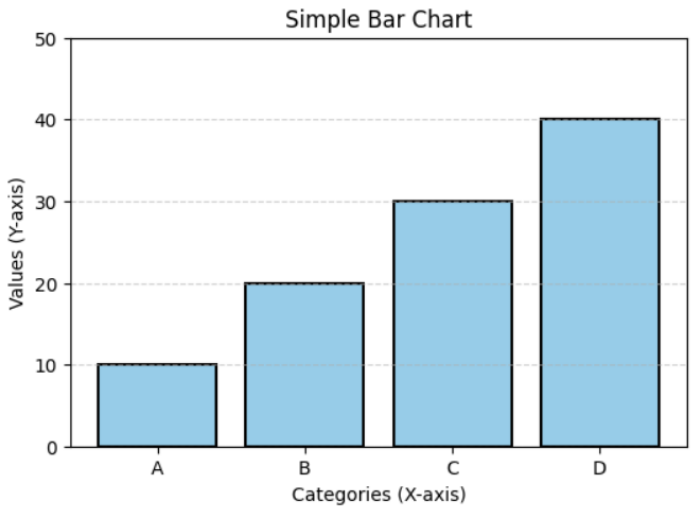
```
Random 5x5 array:
[[ 34  15  84  81 100]
 [ 82  14  64  18  81]
 [ 51  66   3  66  19]
 [ 74   3  63  40  30]
 [ 66  25   4  64   8]]

Reshaped 1x25 array:
[[ 34  15  84  81 100  82  14  64  18  81  51  66   3  66  19  74   3  63
   40  30  66  25   4  64   8]]

Statistics:
   Mean: 46.20
   Median: 51.00
   Variance: 893.28
   Standard Deviation: 29.89

Random 10-element array (sorted): [  7  19  23  43  66  68  76  81  99 100]
Percentile values (25th, 50th, 75th): [28.   67.   79.75]
Decile values (10th to 100th): [ 17.8  22.2  37.   56.8  67.   71.2  77.5  84.6
Quartile values: [28.   67.   79.75]
```

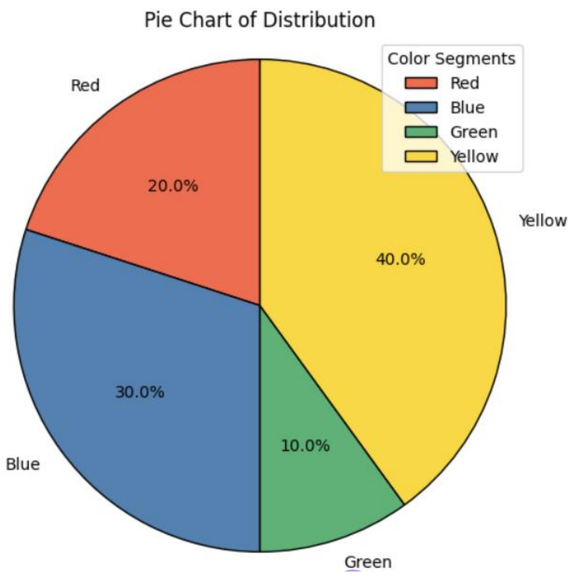
Interpretation/Program/code:

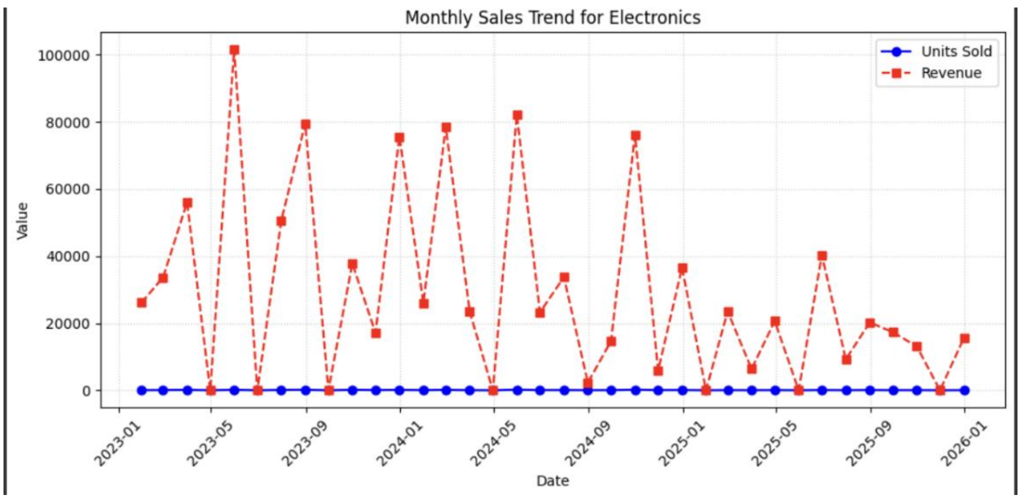
```
import pandas as pd
import matplotlib.pyplot as plt

print("\n" + "="*40)
print("PART 2: Matplotlib Visualization (Charts will be displayed)")
print("="*40)

# --- Bar Chart ---
plt.figure(figsize=(6, 4))
x_values = ['A', 'B', 'C', 'D']
y_values = [10, 20, 30, 40]

plt.bar(x_values, y_values, color='skyblue', edgecolor='black', linewidth=1.5)
plt.title('Simple Bar Chart')
plt.xlabel('Categories (X-axis)')
plt.ylabel('Values (Y-axis)')
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.ylim(0, 50)
print("Bar Chart successfully generated.")

# --- Pie Chart ---
labels = ['Red', 'Blue', 'Green', 'Yellow']
values = [20, 30, 10, 40]
colors = ['#FF6347', '#4682B4', '#3CB371', '#FFD700']

plt.figure(figsize=(6, 6))
plt.pie(values, labels=labels, colors=colors,
        autopct='%1.1f%%', startangle=90,
        wedgeprops={'edgecolor': 'black', 'linewidth': 1})
```

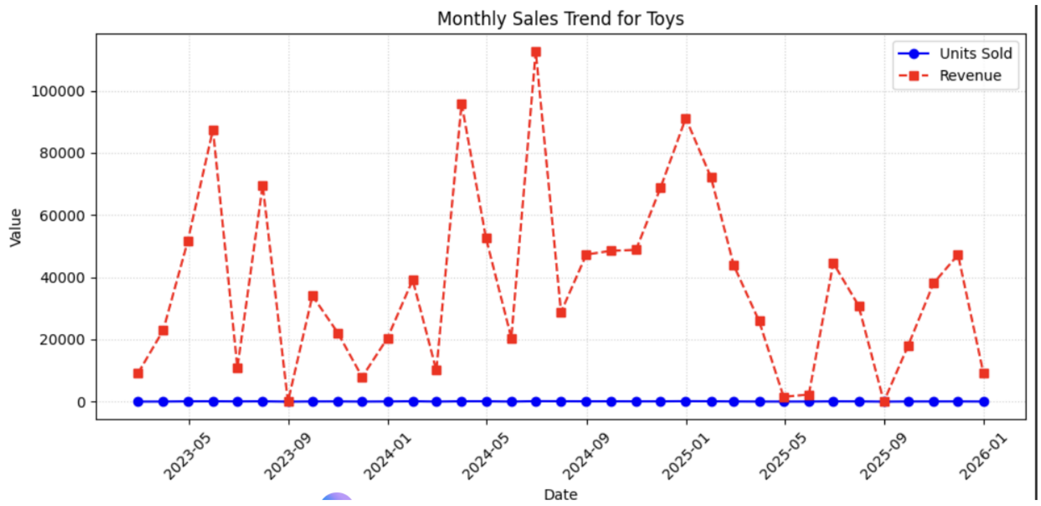
```
plt.title('Pie Chart of Distribution')
plt.axis('equal')
plt.legend(title="Color Segments", loc="best")
print("Pie Chart successfully generated.")

# Part 3: Pandas Data Analysis and Visualization
print("\n" + "="*40)
print("PART 3: Pandas Data Analysis and Visualization")
print("="*40)

# 1. Load the dataset (Ensure the file is in the same directory)
NEW_CSV_NAME = 'ecommerce_sales.csv'
data = pd.read_csv(NEW_CSV_NAME)

# 2. Rename and create required columns
data.rename(columns={
    'Product_Category': 'Product',
    'Units_Sold': 'Units Sold'
}, inplace=True)

# Create 'Revenue' column
data['Revenue'] = data['Price'] * data['Units Sold']

# Convert 'Date' to datetime
data['Date'] = pd.to_datetime(data['Date'], errors='coerce')

# Drop rows with missing critical info
data = data.dropna(subset=['Date', 'Product', 'Units Sold', 'Revenue'])

print(f"Data loaded and cleaned. Total unique products: {data['Product'].nunique()}")
print("Columns used: Date, Product, Units Sold, Revenue (Calculated).")

# 3. Calculate statistics for Units Sold and Revenue
units_sold_stats = data['Units Sold'].agg(['mean', 'median', 'var', 'std']).to_frame('Units Sold')
revenue_stats = data['Revenue'].agg(['mean', 'median', 'var', 'std']).to_frame('Revenue')

print("\nStatistics for Units Sold and Revenue:")
print(pd.concat([units_sold_stats, revenue_stats], axis=1).applymap(lambda x: f'{x:.2f}'))

# 4. Group by Product and calculate statistics
grouped_stats = data.groupby('Product')[['Units Sold', 'Revenue']].agg(['mean', 'median', 'var', 'std'])
print("\nGrouped Statistics by Product:")
print(grouped_stats.applymap(lambda x: f'{x:.2f}'))

# 5. Create line chart to visualize trends over time per product
data_monthly = data.set_index('Date').groupby('Product')[['Units Sold',
'Revenue']].resample('M').sum().reset_index()

print("\nGenerating Line Charts for trend analysis...")

for product in data_monthly['Product'].unique():
    product_data = data_monthly[data_monthly['Product'] == product]

    plt.figure(figsize=(10, 5))
    plt.plot(product_data['Date'], product_data['Units Sold'],
             label='Units Sold', marker='o', linestyle='-', color='blue')

    plt.plot(product_data['Date'], product_data['Revenue'],
             label='Revenue', marker='s', linestyle='--', color='red')

    plt.title(f'Monthly Sales Trend for {product}')
    plt.xlabel('Date')
    plt.ylabel('Value')
    plt.legend()
    plt.grid(True, linestyle=':', alpha=0.6)
    plt.xticks(rotation=45)
    plt.tight_layout()

# Show all charts
plt.show()

print("\n" + "="*40)
print("Practical execution complete.")
print("="*40)
```

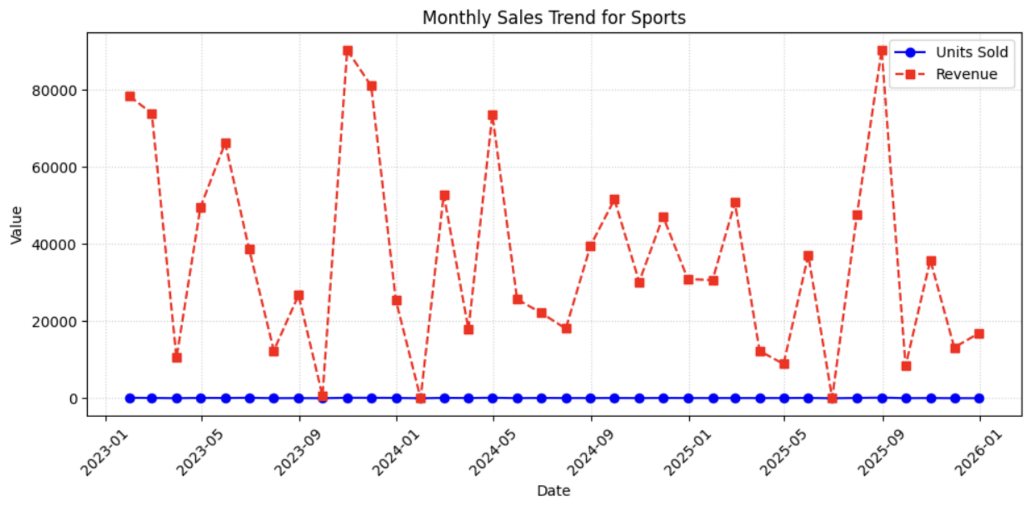
Output :

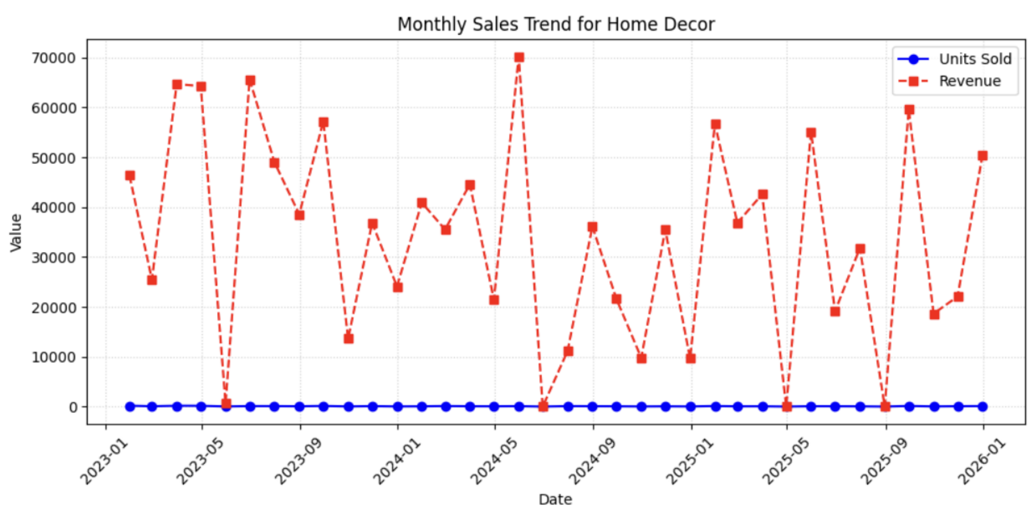


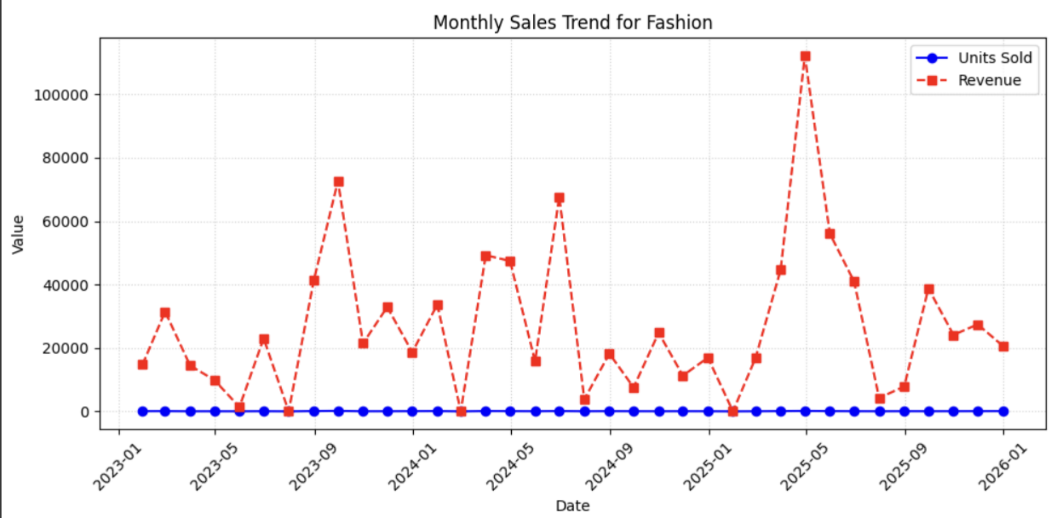



Monthly Sales Trend for Electronics
Units Sold
Revenue
Value
Date
0
20000
40000
60000
80000
100000
2023-01
2023-05
2023-09
2024-01
2024-05
2024-09
2025-01
2025-05
2025-09
2026-01


Monthly Sales Trend for Toys
Units Sold
Revenue
Value
Date
0
20000
40000
60000
80000
100000
2023-05
2023-09
2024-01
2024-05
2024-09
2025-01
2025-05
2025-09
2026-01


Monthly Sales Trend for Sports
Units Sold
Revenue
Value
Date
0
20000
40000
60000
80000
2023-01
2023-05
2023-09
2024-01
2024-05
2024-09
2025-01
2025-05
2025-09
2026-01





Conclusion:
In conclusion, this lab practical provided hands-on experience with NumPy, Matplotlib, and Pandas libraries in Python for data manipulation and visualization. These libraries have wide- ranging applications in various fields, enabling researchers and analysts to gain insights from large datasets quickly and efficiently. Through exercises such as calculating statistical measures and visualizing data using charts, we explored the functionality and flexibility of these powerful data analysis tools. Overall, gaining proficiency in these libraries equips individuals to tackle complex data analysis challenges and contribute to their respective fields of study or industries.

Quiz:

1. **What is the difference between a list and a tuple in Python?**

Ans:

In Python, both lists and tuples are used to store collections of items, but they have some key differences:

1. **Mutability:**
   - List: Lists are mutable, which means you can change their contents (add, remove, or modify elements) after they are created. You can use methods like `append()`, `remove()`, and `pop()` to modify a list.

- Tuple: Tuples are immutable, which means once you create a tuple, you cannot change its elements. You can't add or remove elements from a tuple, nor can you modify the existing elements.

2. **Syntax:**
   - List: Lists are created using square brackets `[...]`. For example: `my_list = [1, 2, 3]`
   - Tuple: Tuples are created using parentheses `(...)`. For example: `my_tuple = (1, 2, 3)` or even without parentheses: `my_tuple = 1, 2, 3`

3. **Performance:**
   - Lists can be slightly slower than tuples in terms of iteration and access time because of their mutability. When you modify a list, it may require resizing the underlying data structure, which can introduce some overhead.
   - Tuples, being immutable, are generally faster for iteration and access, and they consume slightly less memory than lists.

4. **Use Cases:**
   - Lists are typically used when you have a collection of items that may change or need to be modified over time. For example, a list of tasks to-do that you can add or remove items from.
   - Tuples are often used when you have a collection of items that should not change. For example, a tuple might be used to represent a set of coordinates (x, y) or a record in a database.

5. **Packing and Unpacking:**
   - Tuples are often used for "packing" and "unpacking" values. For example, you can return multiple values from a function as a tuple and then unpack those values when calling the function.

2. How can you use NumPy to generate an array of random numbers?

NumPy provides a variety of functions to generate arrays of random numbers. You can use these functions to create arrays with random values according to different distributions. Here are some common ways to generate random number arrays using NumPy:

1. **Random Values from a Uniform Distribution:**
   - To generate random numbers between 0 and 1 from a uniform distribution:

```python
import numpy as np

random_array = np.random.rand(5, 5)  # Creates a 5x5 array of random values
```

2. **Random Integers:**
   - To generate random integers within a specified range:

```python
import numpy as np

random_integers = np.random.randint(1, 100, size=10) # Generates 10 random
 integers between 1 and 100
```

3. **Standard Normal Distribution (Gaussian):**
   - To generate random numbers from a standard normal distribution (mean=0, standard deviation=1):

```python
import numpy as np

random_values = np.random.randn(10)  # Generates an array of 10 random numbers from
 a standard normal distribution
```

4. **Custom Normal Distribution:**
   - To generate random numbers from a custom normal distribution with a specified mean and standard deviation:

```python
import numpy as np

mean = 5
std_dev = 2
random_values = mean + std_dev * np.random.randn(10) # Generates an array of 10
 random numbers from a custom normal distribution
```

5. **Random Numbers from Other Distributions:**
   - NumPy also provides functions to generate random numbers from other distributions like exponential, Poisson, binomial, and more. For example:

```python
import numpy as np

exponential_values  =  np.random.exponential(scale=2,  size=10)    # Generates
 10 random values from an exponential distribution with scale=2
```

Suggested References:-

1. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
2. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
3. Data Science For Dummies by Lillian Pierson , Jake Porway

Rubrics wise marks obtained

Understanding of Problem	Analysis of the Problem	Capability of writing program	Documentation	Total
02	02	05	01	10

Experiment No: 4

Date:

AIM: Implementation of Various Probability Distributions with NumPy Random Library Functions

Relevant CO: - CO3
Objective:
The objective of this lab practical is to gain an understanding of various probability distributions and implement those using NumPy random library functions.

Materials Used:
- Python environment (Anaconda, Jupyter Notebook, etc.)
- NumPy library

Procedure:

1. Introduction to Probability Distributions:
   - Probability theory is the branch of mathematics that deals with the study of random events or phenomena. In probability theory, a probability distribution is a function that describes the likelihood of different outcomes in a random process. Probability distributions can be categorized into two types: discrete and continuous.
   - Discrete probability distributions are used when the possible outcomes of a random process are countable and can be listed. The most commonly used discrete probability distributions are Bernoulli, Binomial, and Poisson distributions.
   - Continuous probability distributions are used when the possible outcomes of a random process are not countable and can take any value within a certain range. The most commonly used continuous probability distributions are Normal and Exponential distributions.
   - Each probability distribution has its own set of properties, such as mean, variance, skewness, and kurtosis. Mean represents the average value of the random variable, variance represents how much the values vary around the mean, skewness represents the degree of asymmetry of the distribution, and kurtosis represents the degree of peakedness or flatness of the distribution.
   - Probability distributions are widely used in fields such as finance, engineering, physics, and social sciences to model real-world phenomena and make predictions about future events. Understanding different probability distributions and their properties is an important tool for analyzing data and making informed decisions.

2. Implementation of Probability Distributions using NumPy random library functions:

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

# Generate 1000 random numbers following a normal distribution with mean 0 and standard deviation 1
normal_dist = np.random.normal(0, 1, 1000)

# Calculate the mean and standard deviation of the distribution
mean = np.mean(normal_dist)
std_dev = np.std(normal_dist)

# Generate 1000 random numbers following a Poisson distribution with lambda 5
```

```
poisson_dist = np.random.poisson(5, 1000)

# Calculate the mean and variance of the Poisson distribution
poisson_mean = np.mean(poisson_dist)
poisson_var = np.var(poisson_dist)

# Plot the PDF and CDF of the normal distribution
plt.hist(normal_dist, bins=30, density=True, alpha=0.5)
plt.plot(np.sort(normal_dist), 1/(std_dev*np.sqrt(2*np.pi))*np.exp(-(np.sort(normal_dist)-
mean)**2/(2*std_dev**2)), linewidth=2)
plt.plot(np.sort(normal_dist),      0.5*(1+np.tanh((np.sort(normal_dist)-mean)/std_dev*np.sqrt(2))),
linewidth=2)
plt.show()

# Plot the PDF and CDF of the Poisson distribution
plt.hist(poisson_dist, bins=15, density=True, alpha=0.5)
plt.plot(np.arange(0, 15, 0.1), np.exp(-poisson_mean)*poisson_mean**np.arange(0, 15,
0.1)/np.math.factorial(np.arange(0, 15, 0.1)), linewidth=2)
plt.plot(np.arange(0, 15, 0.1), np.exp(oisson_mean)*np.array([np.sum(poisson_var**np.arange(0,
i+1))/np.math.factorial(np.arange(0, i+1)) for i in np.arange(0, 15, 0.1)]), linewidth=2)
plt.show()
```

In this example, we generate 1000 random numbers following a normal distribution with mean 0 and standard deviation 1 using the `np.random.normal()` function. We then calculate the mean and standard deviation of the distribution using the `np.mean()` and `np.std()` functions.

We also generate 1000 random numbers following a Poisson distribution with lambda 5 using the `np.random.poisson()` function. We calculate the mean and variance of the Poisson distribution using the `np.mean()` and `np.var()` functions.

We then plot the probability density function (PDF) and cumulative distribution function (CDF) of both distributions using the `plt.hist()` and `plt.plot()` functions from the Matplotlib library.

3. Exercise:
- Generate a dataset of your choice or given by faculty with a given probability distribution using NumPy random library functions
- Plot the probability density function and cumulative distribution function for the generated data
- Calculate the descriptive statistics of the generated data

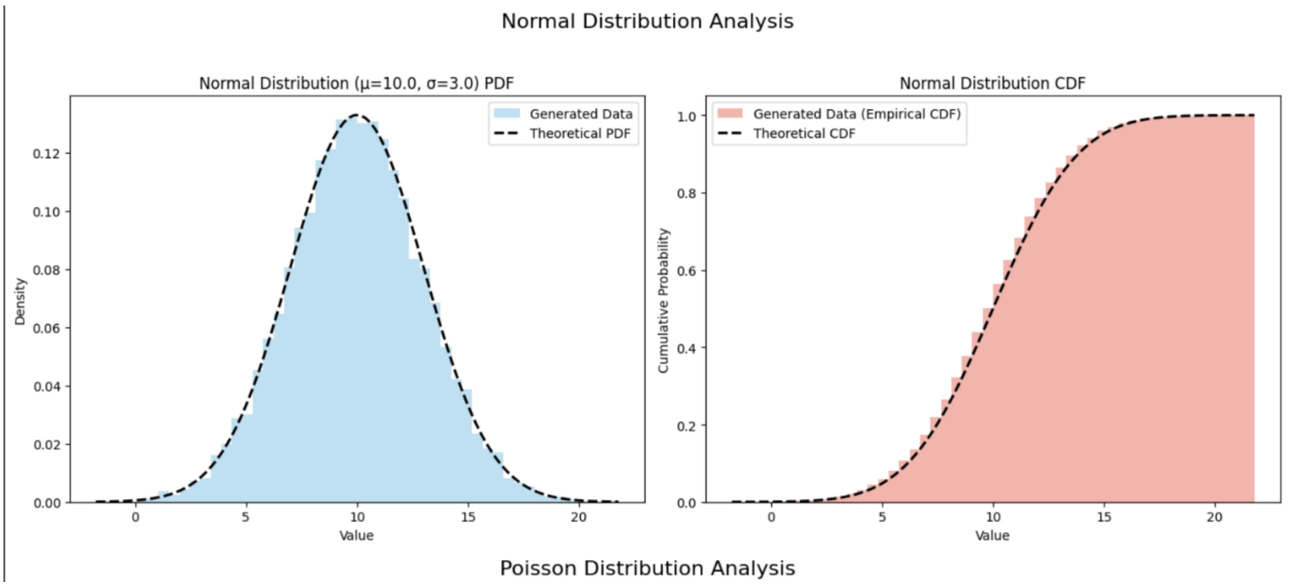
Interpretation/Program/code:

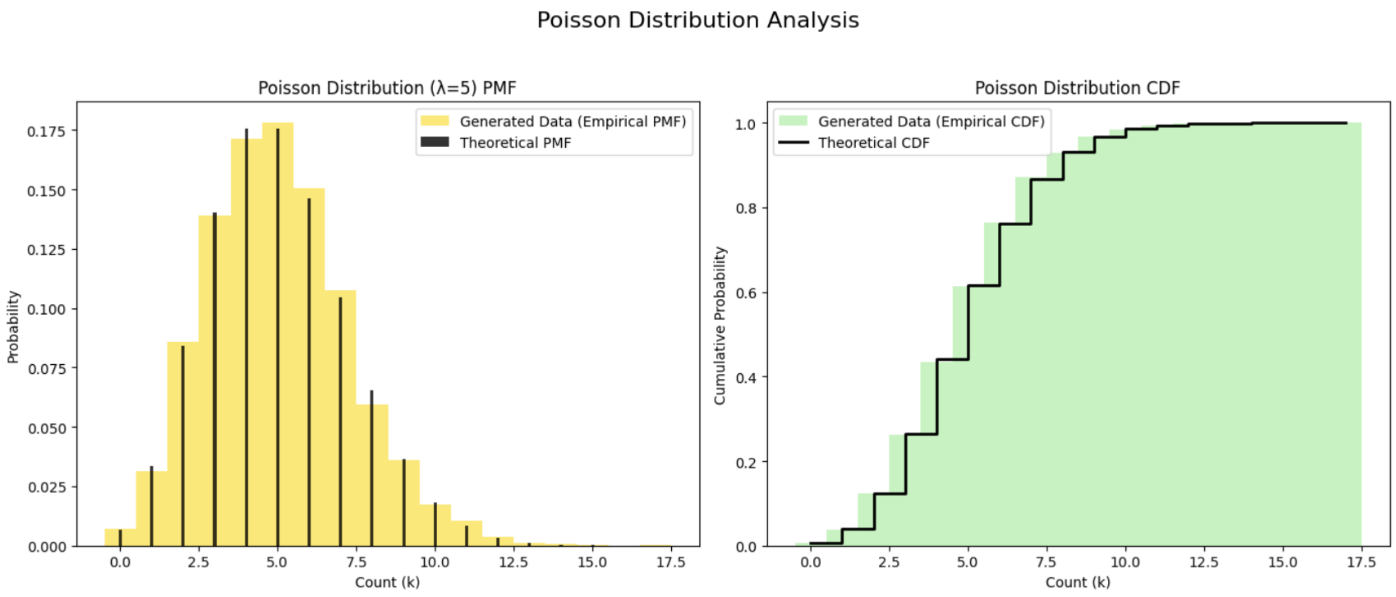
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm, poisson # Import for accurate theoretical PDF/CDF plots

# Set a random seed for reproducibility
np.random.seed(42)

# 1. Normal Distribution (Continuous) - Modelling something like heights or errors
MEAN_N = 10.0
STD_DEV_N = 3.0
```

```
SAMPLE_SIZE = 10000

# Generate 10,000 random numbers following a normal distribution
normal_data = np.random.normal(MEAN_N, STD_DEV_N, SAMPLE_SIZE)

# Calculate the statistics
mean_data_n = np.mean(normal_data)
std_dev_data_n = np.std(normal_data)
median_data_n = np.median(normal_data)

# --- Plotting PDF and CDF ---
plt.figure(figsize=(14, 6))

# Subplot 1: Probability Density Function (PDF)
plt.subplot(1, 2, 1)
# Plot histogram of generated data
plt.hist(normal_data, bins=50, density=True, alpha=0.6, color='skyblue', label='Generated Data')
# Plot the theoretical PDF curve for comparison
x = np.linspace(normal_data.min(), normal_data.max(), 100)
pdf_theoretical = norm.pdf(x, MEAN_N, STD_DEV_N)
plt.plot(x, pdf_theoretical, 'k--', linewidth=2, label='Theoretical PDF')
plt.title(f"Normal Distribution (μ={MEAN_N}, σ={STD_DEV_N}) PDF")
plt.xlabel("Value")
plt.ylabel("Density")
plt.legend()

# Subplot 2: Cumulative Distribution Function (CDF)
plt.subplot(1, 2, 2)
# Plot empirical CDF of generated data
plt.hist(normal_data, bins=50, density=True, cumulative=True, alpha=0.6, color='salmon',
label='Generated Data (Empirical CDF)')
# Plot the theoretical CDF curve
cdf_theoretical = norm.cdf(x, MEAN_N, STD_DEV_N)
plt.plot(x, cdf_theoretical, 'k--', linewidth=2, label='Theoretical CDF')
plt.title("Normal Distribution CDF")
plt.xlabel("Value")
plt.ylabel("Cumulative Probability")
plt.legend()
plt.suptitle("Normal Distribution Analysis", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()


# 2. Poisson Distribution (Discrete) - Modelling something like calls per hour (λ=5)

LAMBDA_P = 5
poisson_data = np.random.poisson(LAMBDA_P, SAMPLE_SIZE)

# Calculate the statistics
mean_data_p = np.mean(poisson_data)
variance_data_p = np.var(poisson_data)
median_data_p = np.median(poisson_data)
```

```
# --- Plotting PMF and CDF ---
plt.figure(figsize=(14, 6))
bins_p = np.arange(poisson_data.min() - 0.5, poisson_data.max() + 1.5, 1)

# Subplot 1: Probability Mass Function (PMF)
plt.subplot(1, 2, 1)
# Plot histogram of generated data (acts as empirical PMF)
plt.hist(poisson_data, bins=bins_p, density=True, alpha=0.6, color='gold', label='Generated Data
(Empirical PMF)')

# Plot the theoretical PMF bars
k = np.arange(0, poisson_data.max() + 1)
pmf_theoretical = poisson.pmf(k, LAMBDA_P)
plt.bar(k, pmf_theoretical, width=0.1, color='k', alpha=0.8, label='Theoretical PMF')
plt.title(f"Poisson Distribution (λ={LAMBDA_P}) PMF")
plt.xlabel("Count (k)")
plt.ylabel("Probability")
plt.legend()

# Subplot 2: Cumulative Distribution Function (CDF)
plt.subplot(1, 2, 2)
# Plot empirical CDF
plt.hist(poisson_data, bins=bins_p, density=True, cumulative=True, alpha=0.6, color='lightgreen',
label='Generated Data (Empirical CDF)')

# Plot theoretical CDF
cdf_p = poisson.cdf(k, LAMBDA_P)
plt.step(k, cdf_p, color='k', where='post', linewidth=2, label='Theoretical CDF')
plt.title("Poisson Distribution CDF")
plt.xlabel("Count (k)")
plt.ylabel("Cumulative Probability")
plt.legend()
plt.suptitle("Poisson Distribution Analysis", fontsize=16)
plt.tight_layout(rect=[0, 0, 1, 0.95])
plt.show()

# 3. Final Descriptive Statistics Output

print("\n" + "="*40)
print("Descriptive Statistics for Generated Distributions:")
print("="*40)
print("\n--- Normal Distribution ---")
print(f"Theoretical Mean (μ): {MEAN_N:.2f}")
print(f"Generated Sample Mean: {mean_data_n:.2f}")
print(f"Generated Sample Standard Deviation: {std_dev_data_n:.2f}")
print(f"Generated Sample Median: {median_data_n:.2f}")
print(f"Generated Sample Min/Max: {normal_data.min():.2f} / {normal_data.max():.2f}")

print("\n--- Poisson Distribution ---")
print(f"Theoretical Mean (λ): {LAMBDA_P:.2f}")
# In a Poisson distribution, Variance is theoretically equal to the Mean (λ)
print(f"Theoretical Variance: {LAMBDA_P:.2f}")
print(f"Generated Sample Mean: {mean_data_p:.2f}")
print(f"Generated Sample Variance: {variance_data_p:.2f}")
print(f"Generated Sample Median: {median_data_p:.2f}")
print(f"Generated Sample Min/Max: {poisson_data.min()} / {poisson_data.max()}")
print("="*40)
```

Output:





```
--- Normal Distribution ---
Theoretical Mean (μ): 10.00
Generated Sample Mean: 9.99
Generated Sample Standard Deviation: 3.01
Generated Sample Median: 9.99
Generated Sample Min/Max: -1.77 / 21.78

--- Poisson Distribution ---
Theoretical Mean (λ): 5.00
Theoretical Variance: 5.00
Generated Sample Mean: 5.02
Generated Sample Variance: 5.05
Generated Sample Median: 5.00
Generated Sample Min/Max: 0 / 17
```

Conclusion:
This lab practical provided an opportunity to explore and implement various probability distributions using NumPy random library functions. By understanding and applying different probability distributions, one can model real-world phenomena and make predictions about future events. With the knowledge gained in this lab practical, student will be equipped to work with probability distributions and analyze data in a wide range of fields, including finance, engineering, and social sciences.

Quiz:

1. Which NumPy function can be used to generate random numbers from a normal distribution?

a) numpy.random.uniform
b) numpy.random.poisson
c) numpy.random.normal
d) numpy.random.exponential

Ans: c)numpy.random.normal

2. What is the purpose of the probability density function (PDF) in probability distributions?

a) To calculate the cumulative probability
b) To generate random numbers
c) To visualize the distribution
d) To calculate the probability of a specific value

Ans : d) To calculate the probability of a specific value

Suggested References:-

1. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
2. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
3. Data Science For Dummies by Lillian Pierson , Jake Porway

Rubrics wise marks obtained

Understanding of Problem	Analysis of the Problem	Capability of writing program	Documentation	Total
02	02	05	01	10

Experiment No: 5

Date:

AIM: Implementation of Estimation of Parameters for the Best-Fit Probability Distribution using the Fitter Class in Python.

Relevant CO: - CO3

Objectives: The objective of this lab practical is to learn how to estimate the parameters for the best-fit probability distribution for a given dataset using the Fitter class in Python.
Materials Used:

1. Python 3.x
2. Jupyter Notebook
3. NumPy library
4. Fitter library

Theory:

Dataset:
Consider the following dataset, which represents the heights of individuals in centimeters:
170, 165, 180, 172, 160, 175, 168, 155, 185, 190, 162, 178, 168, 172, 180, 160, 165, 172, 168, 175
Procedure:

1. Introduction to Parameter Estimation and Probability Distributions:

Probability distributions provide a mathematical framework for describing the likelihood of different outcomes or events in a dataset. Parameter estimation plays a crucial role in probability distributions as it involves determining the values of the parameters that best describe the observed data.

Parameter estimation is important because it allows us to make inferences, predictions, and draw meaningful conclusions from the data. By estimating the parameters, we can effectively model and analyze various phenomena, summarizing complex datasets in a more simplified and interpretable manner.

The concept of the best-fit probability distribution refers to finding the distribution that provides the closest match to the observed data. The best-fit distribution is determined by estimating the parameters in such a way that the observed data exhibits the highest likelihood or best matches the underlying characteristics of the data. Selecting the best-fit distribution helps us understand the data's behavior, make accurate predictions, and gain insights into its properties.

Commonly used probability distributions include the normal (Gaussian) distribution, uniform distribution, exponential distribution, Poisson distribution, and binomial distribution. Each distribution has its own characteristics and applications in various fields.

Understanding parameter estimation and probability distributions allows us to effectively model and analyze data, make informed decisions, and gain insights into the underlying properties of the data. By estimating the parameters for the best-fit probability distribution, we can unlock valuable information and extract meaningful patterns from the observed data.

2. Installation of Required Libraries:
   - Install the necessary libraries, including NumPy and Fitter, using the appropriate package manager.

3. Loading and Preparing the Dataset:
   - Load the dataset from a file or use the provided dataset.
   - Perform any necessary data preprocessing steps, such as cleaning or normalization.

4. Estimating Parameters for Best-Fit Probability Distribution using Fitter Class:
   - Import the required libraries and instantiate the Fitter class.
   - Fit the dataset to various probability distributions available in the Fitter class using the `.fit()` method.
   - Determine the best-fit distribution based on goodness-of-fit metrics, such as AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion).
   - Retrieve the estimated parameters for the best-fit distribution using the `.summary()` method.

5. Visualization of Best-Fit Distribution:
   - Plot the histogram of the dataset.
   - Plot the probability density function (PDF) of the best-fit distribution over the histogram.

6. Interpretation and Analysis:
   - Interpret the estimated parameters of the best-fit distribution.
   - Analyze the goodness of fit and discuss any potential limitations or considerations.

7. Conclusion:
   - Summarize the importance of parameter estimation and the best-fit distribution in data analysis.
   - Highlight the capabilities of the Fitter class in Python for automating the estimation of parameters.
   - Discuss potential applications and further exploration in different domains.

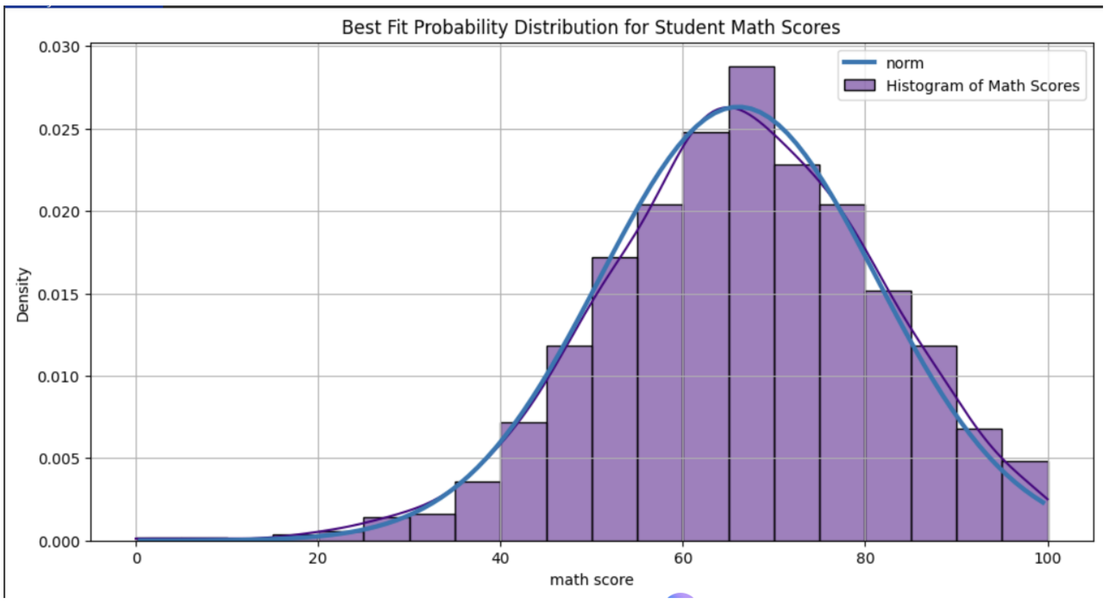
Interpretation/Program/code:

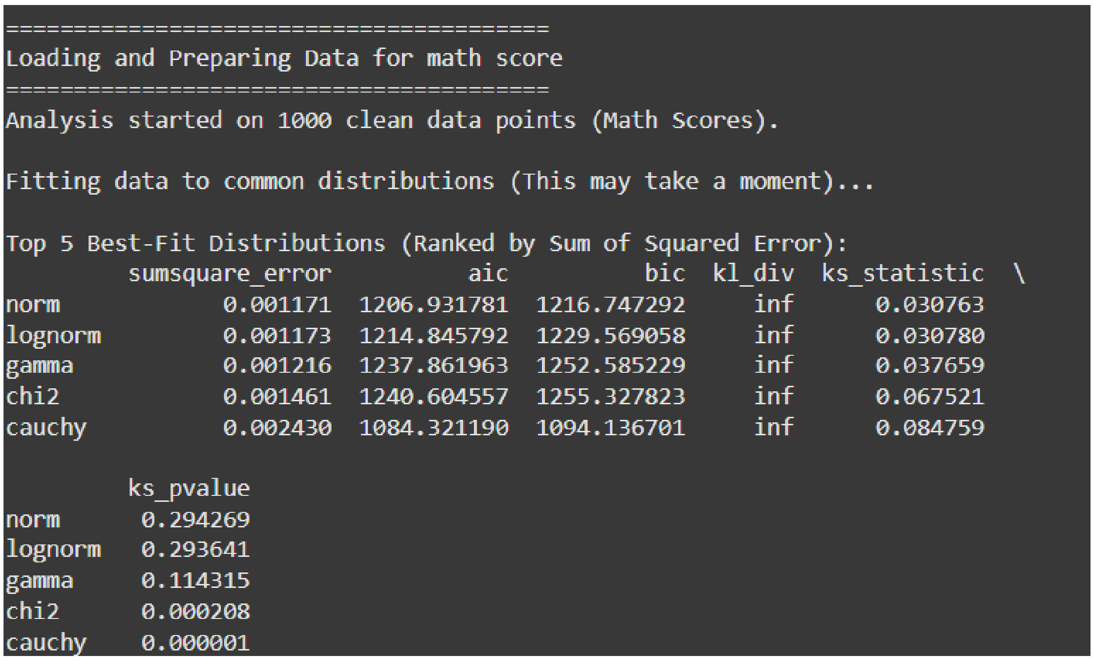
```
# === Import Required Libraries ===
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from fitter import Fitter, get_common_distributions
import seaborn as sns
from scipy.stats import norm

# === Configuration ===
CSV_FILE_NAME = 'StudentsPerformance.csv'
ANALYSIS_COLUMN = 'math score'  # Change this if you want to analyze another column

# ==================================================================
# Part 1 & 3: Loading and Preparing Data
# ==================================================================
print("="*40)
print(f"Loading and Preparing Data for '{ANALYSIS_COLUMN}'")
print("="*40)
```

```
analysis_data = np.array([])  # Initialize as empty array

try:
   # Load the CSV file
   data = pd.read_csv(CSV_FILE_NAME)

   # Drop NaN and extract target column
   analysis_data_series = data[ANALYSIS_COLUMN].dropna()
   analysis_data = analysis_data_series.values

   if len(analysis_data) < 100:
      print(f"ERROR: Insufficient data points ({len(analysis_data)}) in '{ANALYSIS_COLUMN}'.")
   else:
      print(f"Analysis started on {len(analysis_data)} clean data points ({ANALYSIS_COLUMN}).")

except FileNotFoundError:
   print(f"FATAL ERROR: The file '{CSV_FILE_NAME}' was not found. Please check the path or
ensure it is unzipped.")
except Exception as e:
   print(f"An unexpected error occurred: {e}")

# === Check if data is valid before continuing ===
if analysis_data.size == 0:
   print("\nCannot proceed with Fitter analysis due to empty or insufficient data.")
else:
   # ==========================================================================
   # Part 4: Estimating Parameters for Best-Fit Distribution
   # ==========================================================================
   print("\nFitting data to common distributions (This may take a moment)...")
   f = Fitter(analysis_data, distributions=get_common_distributions())
   f.fit()

   # Get best fit results
   best_fit_results = f.get_best(method='sumsquare_error')
   best_fit_name = list(best_fit_results.keys())[0]
   best_params = best_fit_results[best_fit_name]

   # Print top 5 best-fit distributions
   print("\nTop 5 Best-Fit Distributions (Ranked by Sum of Squared Error):")
   with pd.option_context('display.max_rows', None, 'display.max_columns', None):
      print(f.summary(Nbest=5, plot=False))

   # ==========================================================================
   # Part 5: Visualization of Best-Fit Distribution
   # ==========================================================================
   plt.figure(figsize=(12, 6))
   sns.histplot(analysis_data, bins=20, kde=True, stat="density", color="indigo", label='Histogram of
Math Scores')

   f.plot_pdf(names=[best_fit_name], N_points=200, color='red', lw=3, label=f'Best Fit:
{best_fit_name.upper()}')
```

```
    plt.legend()
    plt.title(f'Best Fit Probability Distribution for Student {ANALYSIS_COLUMN.title()}')
    plt.xlabel(f'{ANALYSIS_COLUMN}')
    plt.ylabel('Density')
    plt.grid(True)
    plt.show()

    # ===========================================================================
    # Part 6: Interpretation and Analysis
    # ===========================================================================
    print("\n" + "="*40)
    print("Interpretation and Analysis")
    print("="*40)

    mean_data = np.mean(analysis_data)
    std_dev_data = np.std(analysis_data)

    print(f"Descriptive Statistics of Observed Data ({ANALYSIS_COLUMN}):")
    print(f"Mean Score: {mean_data:.2f}")
    print(f"Standard Deviation: {std_dev_data:.2f}")

    print(f"\nBest Fit Distribution Identified by Fitter: {best_fit_name.upper()}")
    print(f"Estimated Parameters: {best_params}")

    # ===========================================================================
    # Part 7: Conclusion
    # ===========================================================================
    print("\n" + "="*40)
    print("Conclusion")
    print("="*40)
    print("The Fitter library successfully identified the best probability distribution for the student math 
scores,")
    print("demonstrating how theoretical models can be applied to analyze continuous educational 
performance data.")
```

Output:





```
=======================================
Loading and Preparing Data for math score
=======================================
Analysis started on 1000 clean data points (Math Scores).

Fitting data to common distributions (This may take a moment)...

Top 5 Best-Fit Distributions (Ranked by Sum of Squared Error):
         sumsquare_error          aic          bic  kl_div  ks_statistic  \
norm            0.001171  1206.931781  1216.747292     inf      0.030763
lognorm         0.001173  1214.845792  1229.569058     inf      0.030780
gamma           0.001216  1237.861963  1252.585229     inf      0.037659
chi2            0.001461  1240.604557  1255.327823     inf      0.067521
cauchy          0.002430  1084.321190  1094.136701     inf      0.084759

         ks_pvalue
norm      0.294269
lognorm   0.293641
gamma     0.114315
chi2      0.000208
cauchy    0.000001
```

Conclusion:

In this example, we have a dataset of heights of individuals. We use the Fitter class from the `fitter` library to estimate the parameters for the best-fit probability distribution.

We instantiate the Fitter class with the dataset `data`. Then, we use the `.fit()` method to fit the data to various distributions available in the Fitter class. The `.fit()` method automatically estimates the parameters for each distribution and selects the best-fit distribution based on the goodness-of-fit metrics.

Finally, we retrieve the best-fit distribution using the `.get_best()` method and print the summary of the distribution using the `.summary()` method. We also plot the histogram of the dataset and overlay the probability density function (PDF) of the best-fit distribution using the `.plot_pdf()` method.

Note: Before running the code, make sure you have the `numpy`, `fitter`, and `matplotlib` libraries installed. You can install the `fitter` library using pip: `pip install fitter`.
Through this practical, we learned the importance of parameter estimation in probability distributions and the significance of selecting the best-fit distribution for accurate modeling and analysis. The Fitter class provided a convenient and efficient way to fit the dataset to various distributions and evaluate their goodness of fit using metrics such as AIC or BIC.

Quiz:

1. What is the purpose of the Fitter class in Python?

a) To fit a probability distribution to a given dataset
b) To generate random numbers from a probability distribution
c) To calculate descriptive statistics of a dataset
d) To visualize the probability distribution of a dataset

Ans : a) To fit a probability distribution to a given dataset

2. Which method of the Fitter class can be used to estimate the best-fit probability distribution for a given dataset?

a) fit
b) predict
c) evaluate
d) transform

Ans:a) fit

Suggested References:-

1. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
2. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
3. Data Science For Dummies by Lillian Pierson , Jake Porway

Rubrics wise marks obtained

Understanding of Problem	Analysis of the Problem	Capability of writing program	Documentation	Total
02	02	05	01	10

Experiment No: 6

Date:

AIM: Implementation of Linear Regression with Scikit-learn library in Python

Relevant CO: - CO4

Objective:
The objective of this lab practical is to implement linear regression to predict the value of a variable in a given dataset. Linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. In this lab, we will explore how to build a linear regression model and use it to make predictions.

Materials Used:
- Python 3.x
- Jupyter Notebook
- NumPy library
- Pandas library
- Matplotlib library
- Scikit-learn library

Dataset:
For this lab, we will use a dataset that contains information about houses and their sale prices. The dataset has the following columns:

- `Area` (in square feet): Represents the area of the house.
- `Bedrooms`: Number of bedrooms in the house.
- `Bathrooms`: Number of bathrooms in the house.
- `Garage Cars`: Number of cars that can be accommodated in the garage.
- `Sale Price` (in dollars): Represents the sale price of the house.

```
Area,Bedrooms,Bathrooms,Garage Cars,Sale Price
2000,3,2,2,250000
1800,4,3,2,280000
2200,3,2,2,265000
1500,2,1,1,200000
2400,4,3,3,320000
1900,3,2,2,275000
1700,3,2,1,230000
2100,4,3,2,295000
```

Procedure:

1. Introduction to Linear Regression:

Linear regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. It aims to find a linear equation that best represents the association between the variables. Linear regression assumes a linear relationship and seeks to minimize the differences between observed and predicted values. It has applications in prediction, understanding correlations, and making datadriven decisions.

The equation for a simple linear regression model can be represented as:

$y = \beta0 + \beta1*x + \varepsilon$

where:

y is the dependent variable
x is the independent variable
$\beta0$ is the y-intercept (the value of y when x = 0)
$\beta1$ is the slope (the change in y for a unit change in x)
ε represents the error term or residual

2. Importing Required Libraries and Loading the Dataset:
  - Import the necessary libraries, including NumPy, Pandas, Matplotlib, and Scikit-learn.
  - Load the dataset into a Pandas DataFrame using the appropriate function or by reading from a file.

3. Exploratory Data Analysis:
  - Perform exploratory data analysis to gain insights into the dataset.
  - Analyze the distribution and statistical summary of the variables.
  - Visualize the relationships between variables using scatter plots or other appropriate plots.

4. Data Preprocessing:
  - Handle missing values, if any, by imputation or removal.
  - Convert categorical variables into numerical representations, if required.
  - Split the dataset into input features (independent variables) and the target variable (dependent variable).

5. Splitting the Dataset into Training and Testing Sets:
  - Split the dataset into training and testing sets to evaluate the model's performance.
  - Typically, use a 70-30 or 80-20 split for training and testing, respectively.

6. Building the Linear Regression Model:
  - Import the LinearRegression class from Scikit-learn.
  - Instantiate the LinearRegression model.
  - Fit the model to the training data using the `.fit()` method.

7. Model Evaluation and Prediction:
  - Evaluate the model's performance using appropriate evaluation metrics, such as mean squared error (MSE) or R-squared.
  - Make predictions on the testing data using the `.predict()` method.

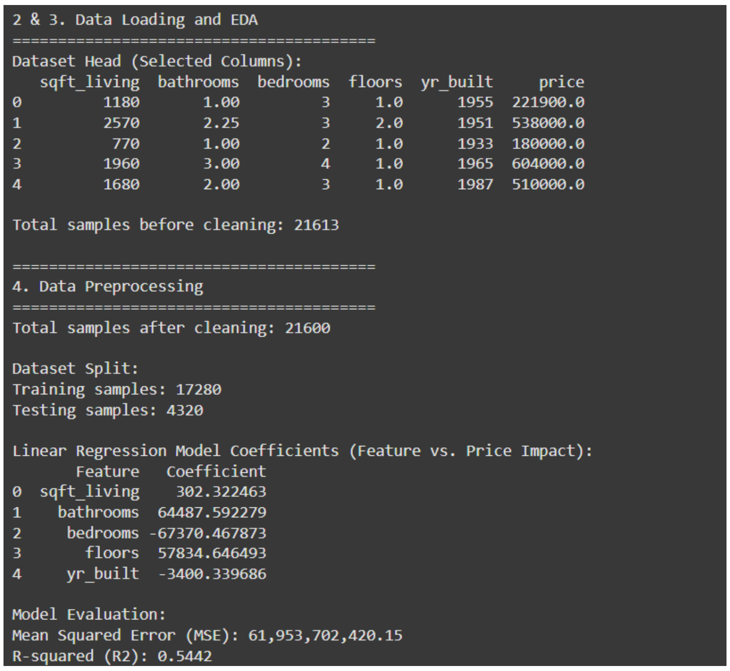
8. Visualization of Results:
  - Visualize the actual values versus the predicted values using scatter plots or other suitable plots.
  - Plot the regression line to show the relationship between the independent and dependent variables.

Interpretation/Program/code:

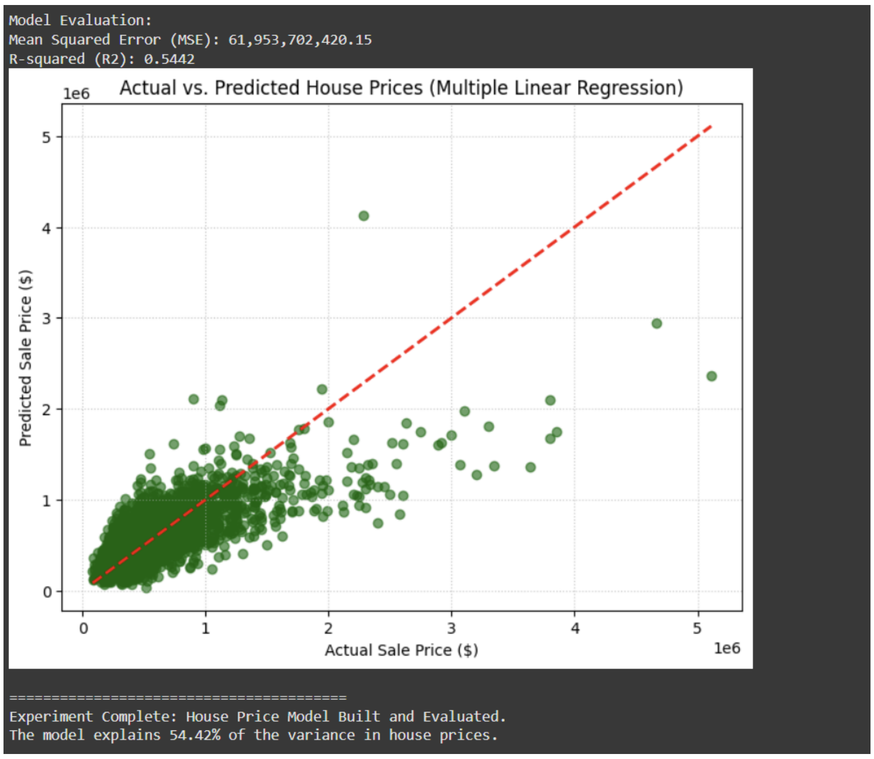
```
# Importing Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import seaborn as sns

# --- Configuration ---
CSV_FILE_NAME = 'kc_house_data.csv'
# Independent Variables (Features)
FEATURES = ['sqft_living', 'bathrooms', 'bedrooms', 'floors', 'yr_built']
# Dependent Variable (Target)
TARGET = 'price'

# 2. Loading the Dataset & 3. Exploratory Data Analysis (EDA)

print("="*40)
print("2 & 3. Data Loading and EDA")
print("="*40)

try:
    # Load the house sales dataset
    data = pd.read_csv(CSV_FILE_NAME)
    # Filter only the necessary columns for analysis
    data = data[FEATURES + [TARGET]]

    print("Dataset Head (Selected Columns):")
    print(data.head())
    print(f"\nTotal samples before cleaning: {len(data)}")

except FileNotFoundError:
    print(f"FATAL ERROR: The file {CSV_FILE_NAME} was not found. Cannot proceed.")
    exit()


# 4. Data Preprocessing

print("\n" + "="*40)
print("4. Data Preprocessing")
print("="*40)

# Check and handle missing or unusual values (e.g., houses with 0 bedrooms)
data.dropna(inplace=True)
data = data[data['bedrooms'] > 0] # Filter out houses listed with 0 bedrooms
data = data[data['sqft_living'] > 0] # Filter out houses with 0 living area

print(f"Total samples after cleaning: {len(data)}")
```

```
# Define Features (X) and Target (y)
X = data[FEATURES]
y = data[TARGET]


# 5. Splitting the Dataset (80-20 split)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("\nDataset Split:")
print(f"Training samples: {len(X_train)}")
print(f"Testing samples: {len(X_test)}")


# 6. Building the Linear Regression Model

model = LinearRegression()
model.fit(X_train, y_train)

# Output the coefficients for interpretation
print("\nLinear Regression Model Coefficients (Feature vs. Price Impact):")
coefficients = pd.DataFrame({'Feature': X_train.columns, 'Coefficient': model.coef_})
print(coefficients)


# 7. Model Evaluation and Prediction

y_pred = model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("\nModel Evaluation:")
print(f"Mean Squared Error (MSE): {mse:,.2f}")
print(f"R-squared (R2): {r2:.4f}")


# 8. Visualization of Results

plt.figure(figsize=(8, 6))
plt.scatter(y_test, y_pred, alpha=0.6, color='darkgreen')

# Plot the ideal regression line (y = x)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)

plt.xlabel("Actual Sale Price ($)")
plt.ylabel("Predicted Sale Price ($)")
plt.title("Actual vs. Predicted House Prices (Multiple Linear Regression)")
plt.grid(True, linestyle=':', alpha=0.7)
plt.show()

print("\n" + "="*40)
print("Experiment Complete: House Price Model Built and Evaluated.")
print(f"The model explains {r2*100:.2f}% of the variance in house prices.")
```

OUTPUT:



```
2 & 3. Data Loading and EDA
========================================
Dataset Head (Selected Columns):
   sqft_living  bathrooms  bedrooms  floors  yr_built     price
0         1180       1.00         3     1.0      1955  221900.0
1         2570       2.25         3     2.0      1951  538000.0
2          770       1.00         2     1.0      1933  180000.0
3         1960       3.00         4     1.0      1965  604000.0
4         1680       2.00         3     1.0      1987  510000.0

Total samples before cleaning: 21613

========================================
4. Data Preprocessing
========================================
Total samples after cleaning: 21600

Dataset Split:
Training samples: 17280
Testing samples: 4320

Linear Regression Model Coefficients (Feature vs. Price Impact):
       Feature    Coefficient
0  sqft_living     302.322463
1    bathrooms   64487.592279
2     bedrooms  -67370.467873
3       floors   57834.646493
4     yr_built   -3400.339686

Model Evaluation:
Mean Squared Error (MSE): 61,953,702,420.15
R-squared (R2): 0.5442
```



```
Model Evaluation:
Mean Squared Error (MSE): 61,953,702,420.15
R-squared (R2): 0.5442
```

```
========================================
Experiment Complete: House Price Model Built and Evaluated.
The model explains 54.42% of the variance in house prices.
```

Conclusion:

In this practical, we implemented linear regression to predict variable values in a dataset. By training a linear regression model using Python and Scikit-learn, we achieved accurate predictions based on variable relationships. Linear regression is a valuable tool for data analysis and prediction, providing insights and supporting decision-making.

Quiz:

1. Which scikit-learn function is used to create a linear regression model object in Python?

a) sklearn.linear_model.LinearRegression
b) sklearn.preprocessing.StandardScaler
c) sklearn.model_selection.train_test_split
d) sklearn.metrics.mean_squared_error
Ans : a) sklearn.linear_model.LinearRegression

2. What is the purpose of the coefficient of determination (R-squared) in linear regression?

a) To measure the average squared difference between predicted and actual values
b) To evaluate the significance of predictor variables
c) To quantify the proportion of variance in the dependent variable explained by the independent variables
d) To determine the optimal number of features for the regression model

Ans : a) To measure the average squared difference between predicted and actual values

Suggested References:-

1. "Pattern Recognition and Machine Learning" by Christopher M. Bishop
2. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
3. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
4. Data Science For Dummies by Lillian Pierson , Jake Porway

Rubrics wise marks obtained

Understanding of Problem	Analysis of the Problem	Capability of writing program	Documentation	Total
04	04	10	04	20

Experiment-7

Date:

AIM: Implementation of Logistic Regression with Scikit-learn library in Python

Relevant CO:- CO4

Objective:
The objective of this lab practical is to implement logistic regression using Scikit-learn library in Python. Logistic regression is a popular classification algorithm used to model the relationship between input variables and categorical outcomes. In this lab, we will explore how to build a logistic regression model and use it for classification tasks.

Materials Used:
- Python 3.x
- Jupyter Notebook
- Scikit-learn library
- Pandas library
- NumPy library
- Matplotlib library

Dataset:
For this lab, we will use a dataset that contains information about customers and whether they churned or not from a telecommunications company. The dataset has the following columns:

- `CustomerID`: Unique identifier for each customer
- `Gender`: Gender of the customer (Male/Female)
- `Age`: Age of the customer
- `Income`: Income of the customer
- `Churn`: Binary variable indicating whether the customer churned (1) or not (0)

```
CustomerID,Gender,Age,Income,Churn
1,Male,32,50000,0
2,Female,28,35000,0
3,Male,45,80000,1
4,Male,38,60000,0
5,Female,20,20000,1
6,Female,55,75000,0
7,Male,42,90000,0
8,Female,29,40000,1
```

Procedure:

1. Introduction to Logistic Regression:
Logistic regression is a classification algorithm used to predict binary outcomes or probabilities. It models the relationship between input features and the probability of an event occurring. By applying the logistic function, it maps the linear regression output to a value between 0 and 1, allowing for classification based on predicted probabilities. Logistic regression is interpretable, handles categorical and continuous features, and finds

applications in various domains. It is a fundamental and effective approach for binary classification tasks.

2. Importing Required Libraries and Loading the Dataset:
   - Import the necessary libraries, including Scikit-learn, Pandas, NumPy, and Matplotlib.
   - Load the dataset into a Pandas DataFrame using the appropriate function or by reading from a file.

3. Exploratory Data Analysis:
   - Perform exploratory data analysis to understand the dataset.
   - Analyze the distribution of variables, detect any missing values, and handle them if necessary.
   - Visualize the relationships between variables using plots and charts.

4. Data Preprocessing:
   - Split the dataset into input features (independent variables) and the target variable (dependent variable).
   - Convert categorical variables into numerical representations using one-hot encoding or label encoding.
   - Split the dataset into training and testing sets for model evaluation.

5. Building the Logistic Regression Model:
   - Import the Logistic Regression class from Scikit-learn.
   - Instantiate the Logistic Regression model.
   - Fit the model to the training data using the `.fit()` method.

6. Model Evaluation and Prediction:
   - Evaluate the model's performance using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score.
   - Make predictions on the testing data using the `.predict()` method.

7. Visualization of Results:
   - Visualize the model's performance using confusion matrix, ROC curve, or other suitable visualizations.
   - Plot the decision boundary to demonstrate the classification boundaries.

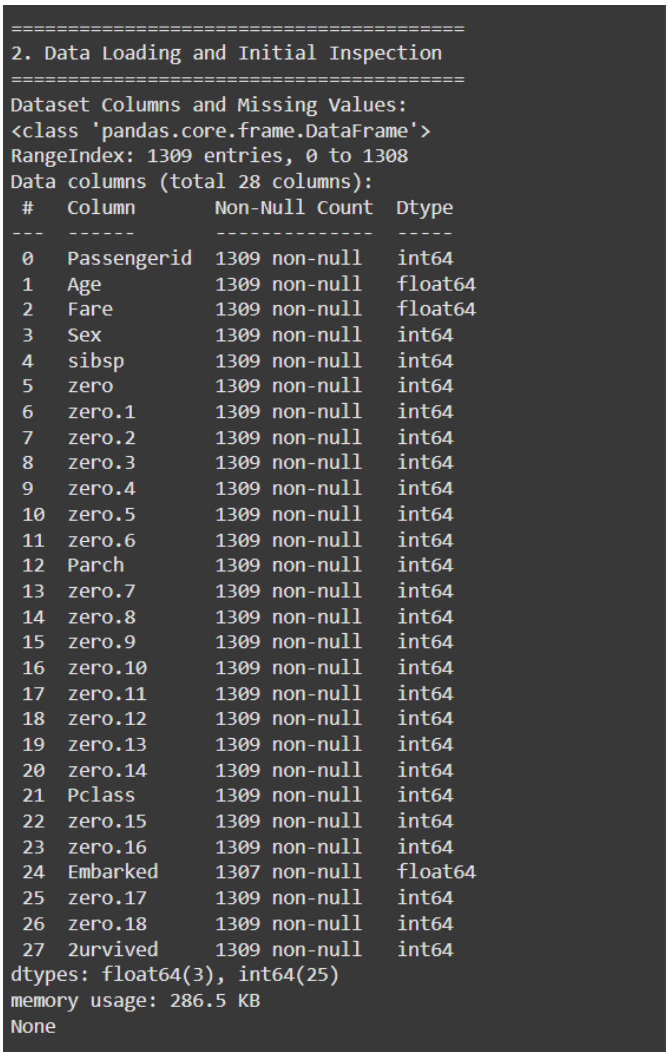
Implementation:

```
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.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer # Used to fill missing values (Age, Embarked)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, roc_curve, roc_auc_score, auc
import seaborn as sns
```

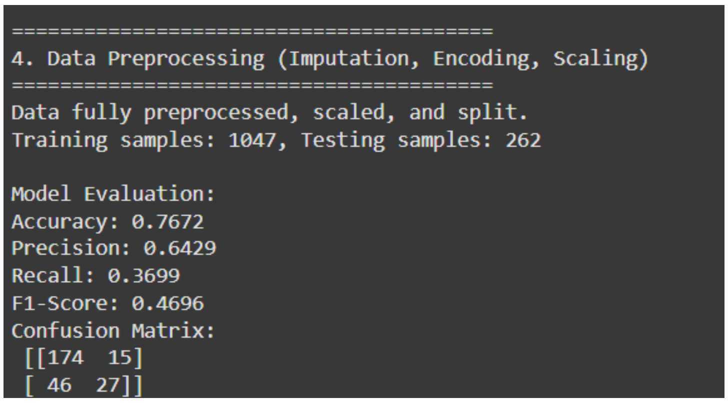
```
# --- Configuration ---
CSV_FILE_NAME = 'train_and_test2.csv'
TARGET = '2urvived' # Corrected target column name

# 2. Loading the Dataset

print("="*40)
print("2. Data Loading and Initial Inspection")
print("="*40)

try:
   data = pd.read_csv(CSV_FILE_NAME)
   print("Dataset Columns and Missing Values:")
   print(data.info())

except FileNotFoundError:
   print(f"FATAL ERROR: The file {CSV_FILE_NAME} was not found. Cannot proceed.")
   exit()


# 4. Data Preprocessing (Imputation, Encoding, and Scaling)

print("\n" + "="*40)
print("4. Data Preprocessing (Imputation, Encoding, Scaling)")
print("="*40)

# 1. Feature Selection
X = data[['Pclass', 'Sex', 'Age', 'sibsp', 'Parch', 'Fare', 'Embarked']].copy()
y = data[TARGET].copy()

# 2. Handle Missing Values:
# Impute 'Age' (continuous) with the mean
imputer_mean = SimpleImputer(strategy='mean')
X.loc[:, 'Age'] = imputer_mean.fit_transform(X[['Age']])

# Impute 'Embarked' (categorical) with the most frequent value (mode)
imputer_mode = SimpleImputer(strategy='most_frequent')
X.loc[:, 'Embarked'] = imputer_mode.fit_transform(X[['Embarked']])

# 3. One-Hot Encoding for Categorical Features
categorical_features = ['Sex', 'Embarked', 'Pclass']
X = pd.get_dummies(X, columns=categorical_features, drop_first=True, dtype=int)

# Drop original Pclass column as it's now encoded:
X = X.drop('Pclass', axis=1, errors='ignore')
# Final list of numerical features that need scaling:
numerical_features = ['Age', 'Fare', 'sibsp', 'Parch']

# 4. Split the dataset (80-20 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 5. FEATURE SCALING (CRITICAL FIX)
scaler = StandardScaler()
```

```
# Scale only the numerical columns
X_train[numerical_features] = scaler.fit_transform(X_train[numerical_features])
X_test[numerical_features] = scaler.transform(X_test[numerical_features])

print("Data fully preprocessed, scaled, and split.")
print(f"Training samples: {len(X_train)}, Testing samples: {len(X_test)}")


# 5. Building the Logistic Regression Model

# Use default solver and max_iter for balanced dataset
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)


# 6. Model Evaluation and Prediction

y_pred = model.predict(X_test)
y_probs = model.predict_proba(X_test)[:, 1]

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, zero_division=0)
recall = recall_score(y_test, y_pred, zero_division=0)
f1 = f1_score(y_test, y_pred, zero_division=0)
conf_matrix = confusion_matrix(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_probs)

print("\nModel Evaluation:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1-Score: {f1:.4f}")
print("Confusion Matrix:\n", conf_matrix)


# 7. Visualization of Results (ROC Curve)

fpr, tpr, _ = roc_curve(y_test, y_probs)
conf_matrix_df = pd.DataFrame(conf_matrix, index=['Actual Died (0)', 'Actual Survived (1)'],
columns=['Predicted Died (0)', 'Predicted Survived (1)'])

plt.figure(figsize=(14, 6))

# Plot 1: Confusion Matrix
plt.subplot(1, 2, 1)
sns.heatmap(conf_matrix_df, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')

# Plot 2: ROC Curve
plt.subplot(1, 2, 2)
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()

print("\n" + "="*40)
print("Experiment Complete: Titanic Survival Prediction Model Built.")
```

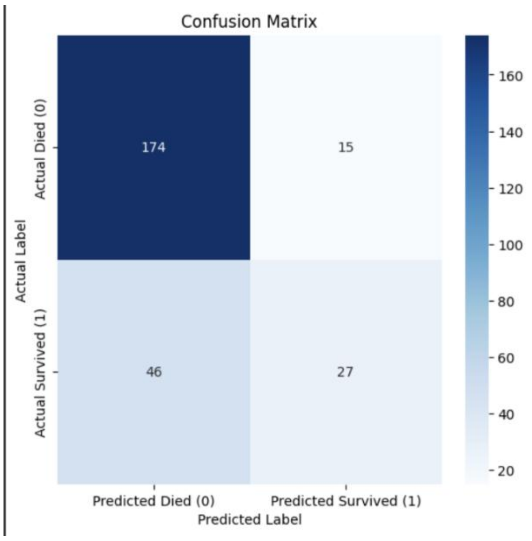
Output:

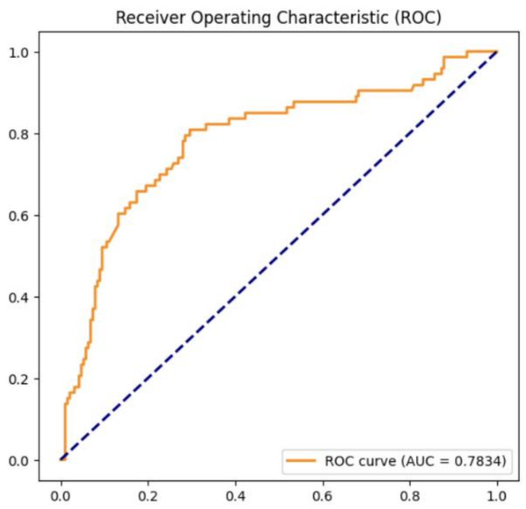


```
========================================
2. Data Loading and Initial Inspection
========================================
Dataset Columns and Missing Values:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 28 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Passengerid  1309 non-null   int64
 1   Age          1309 non-null   float64
 2   Fare         1309 non-null   float64
 3   Sex          1309 non-null   int64
 4   sibsp        1309 non-null   int64
 5   zero         1309 non-null   int64
 6   zero.1       1309 non-null   int64
 7   zero.2       1309 non-null   int64
 8   zero.3       1309 non-null   int64
 9   zero.4       1309 non-null   int64
 10  zero.5       1309 non-null   int64
 11  zero.6       1309 non-null   int64
 12  Parch        1309 non-null   int64
 13  zero.7       1309 non-null   int64
 14  zero.8       1309 non-null   int64
 15  zero.9       1309 non-null   int64
 16  zero.10      1309 non-null   int64
 17  zero.11      1309 non-null   int64
 18  zero.12      1309 non-null   int64
 19  zero.13      1309 non-null   int64
 20  zero.14      1309 non-null   int64
 21  Pclass       1309 non-null   int64
 22  zero.15      1309 non-null   int64
 23  zero.16      1309 non-null   int64
 24  Embarked     1307 non-null   float64
 25  zero.17      1309 non-null   int64
 26  zero.18      1309 non-null   int64
 27  2urvived     1309 non-null   int64
dtypes: float64(3), int64(25)
memory usage: 286.5 KB
None
```



```
========================================
4. Data Preprocessing (Imputation, Encoding, Scaling)
========================================
Data fully preprocessed, scaled, and split.
Training samples: 1047, Testing samples: 262

Model Evaluation:
Accuracy: 0.7672
Precision: 0.6429
Recall: 0.3699
F1-Score: 0.4696
Confusion Matrix:
 [[174  15]
 [ 46  27]]
```





Conclusion:

Logistic regression is a powerful classification algorithm that models the relationship between input features and binary outcomes or probabilities. By utilizing the logistic function, it provides interpretable predictions and is applicable in various domains. Logistic regression is a valuable tool for binary classification tasks, offering simplicity, interpretability, and effectiveness in predicting outcomes based on input features.

Quiz:

1. Which scikit-learn function is used to create a logistic regression model object in Python?

a) sklearn.linear_model.LogisticRegression
b) sklearn.preprocessing.StandardScaler
c) sklearn.model_selection.train_test_split
d) sklearn.metrics.accuracy_score

Ans : a) sklearn.linear_model.LogisticRegression

2. In logistic regression, what does the sigmoid function do?

a) Maps the predicted values to binary classes
b) Calculates the log-odds of the target variable
c) Determines the optimal threshold for classification

d) Measures the goodness of fit of the logistic regression model

Ans : b) Calculates the log-odds of the target variable

Suggested References:-

1. "Pattern Recognition and Machine Learning" by Christopher M. Bishop
2. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
3. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
4. Data Science For Dummies by Lillian Pierson , Jake Porway

Rubrics wise marks obtained

Understanding of Problem	Analysis of the Problem	Capability of writing program	Documentation	Total
04	04	10	02	20

Experiment No: 8

Date:

AIM: Implementation of Decision Tree for Student Classification

Relevant CO :- CO4

Objective:

The objective of this lab practical is to implement a decision tree algorithm to classify students as either average or clever based on given student data. Decision trees are widely used in machine learning and data mining for classification and regression tasks. In this lab, we will explore how to build a decision tree model and use it to classify students based on their attributes.

Materials Used:
- Python 3.x
- Jupyter Notebook
- Scikit-learn library
- Pandas library
- NumPy library
- Matplotlib library

Dataset:
For this lab, we will use a dataset that contains information about students and their performance. The dataset has the following columns:

- `Age`: Age of the student
- `StudyHours`: Number of hours the student studies per day
- `PreviousGrade`: Grade achieved in the previous exam
- `Result`: Classification label indicating whether the student is average (0) or clever (1)

Procedure:

1. Introduction to Decision Trees:
   - Decision trees are widely used in machine learning for classification tasks. They make decisions based on splitting criteria and feature importance. In this lab, we will implement a decision tree algorithm to classify students as average or clever based on their attributes, such as age, study hours, and previous grades.

2. Importing Required Libraries and Loading the Dataset:
   - Import the necessary libraries, including Scikit-learn, Pandas, NumPy, and Matplotlib.
   - Load the dataset into a Pandas DataFrame using the appropriate function or by reading from a file.

3. Exploratory Data Analysis:
   - Perform exploratory data analysis to understand the dataset.
   - Analyze the distribution of variables, detect any missing values, and handle them if necessary.

- Visualize the relationships between variables using plots and charts.

2. Data Preprocessing:
   - Split the dataset into input features (independent variables) and the target variable (dependent variable).
   - Convert categorical variables into numerical representations using one-hot encoding or label encoding.
   - Split the dataset into training and testing sets for model evaluation.

3. Building the Decision Tree Model:
   - Import the DecisionTreeClassifier class from Scikit-learn.
   - Instantiate the DecisionTreeClassifier model with the desired parameters.
   - Fit the model to the training data using the `.fit()` method.

4. Model Evaluation and Prediction:
   - Evaluate the model's performance using appropriate evaluation metrics such as accuracy, precision, recall, and F1-score.
   - Make predictions on the testing data using the `.predict()` method.

5. Visualization of the Decision Tree:
   - Visualize the decision tree using tree plotting techniques available in Scikit-learn or other visualization libraries.
   - Interpret the decision tree structure and analyze the important features.

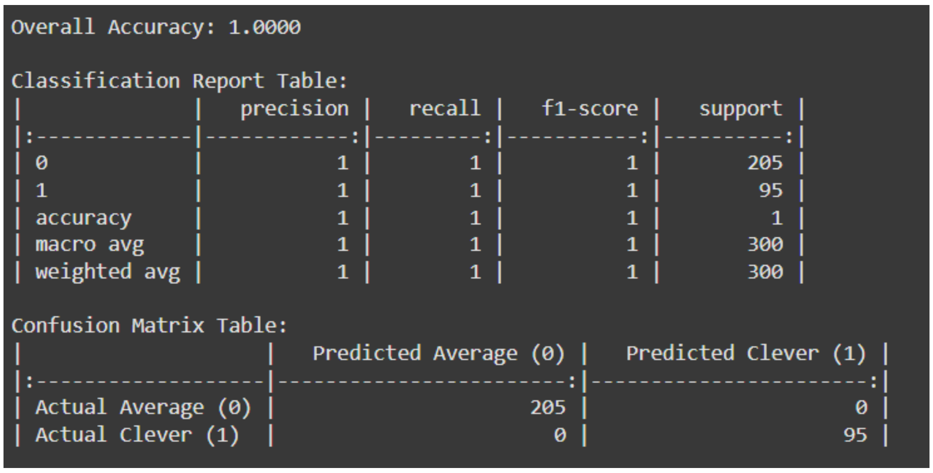
Implementation:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.tree import plot_tree # For visualizing the tree structure
import seaborn as sns

# --- Configuration ---
CSV_FILE_NAME = 'StudentsPerformance.csv'
TARGET = 'Classification_Result' # Engineered target variable (0/1)

# 2. Loading the Dataset & 3. Exploratory Data Analysis (EDA)

print("="*40)
print("2 & 3. Data Loading and EDA")
print("="*40)

try:
    # --- Data Loading (Assuming successful download from Cell 1) ---
    data = pd.read_csv(CSV_FILE_NAME)
    data['Average_Score'] = data[['math score', 'reading score', 'writing score']].mean(axis=1)

except FileNotFoundError:
    print(f"FATAL ERROR: The file {CSV_FILE_NAME} was not found. Cannot proceed.")
    exit()


# 4. Data Preprocessing (Feature Engineering, Encoding, and Splitting)

print("\n" + "="*40)
print("4. Data Preprocessing (Engineering & Encoding)")
print("="*40)

# 1. Feature Engineering: Create the Target Variable ('Clever' vs. 'Average')
CLEVER_THRESHOLD = 75
data[TARGET] = (data['Average_Score'] >= CLEVER_THRESHOLD).astype(int)

# 2. Select Input Features (X)
X = data[['gender', 'race/ethnicity', 'parental level of education', 'lunch', 'test preparation course',
'Average_Score']].copy()
y = data[TARGET]

# 3. One-Hot Encoding for Categorical Features
categorical_features = ['gender', 'race/ethnicity', 'parental level of education', 'lunch', 'test preparation
course']
X = pd.get_dummies(X, columns=categorical_features, drop_first=True, dtype=int)

# 4. Split the dataset (70-30 split)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(f"Total features after encoding: {len(X.columns)}")


# 5. Building the Decision Tree Model

print("\n" + "="*40)
print("5. Building the Decision Tree Model")
print("="*40)
model = DecisionTreeClassifier(max_depth=4, random_state=42, class_weight='balanced')
model.fit(X_train, y_train)



# 6. Model Evaluation and Prediction

y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print("\n" + "="*40)
print("6. MODEL EVALUATION (Tabular Output)")
print("="*40)
print(f"Overall Accuracy: {accuracy:.4f}")

# --- FIX: Classification Report in Table Format ---
report = classification_report(y_test, y_pred, output_dict=True, zero_division=0)
```

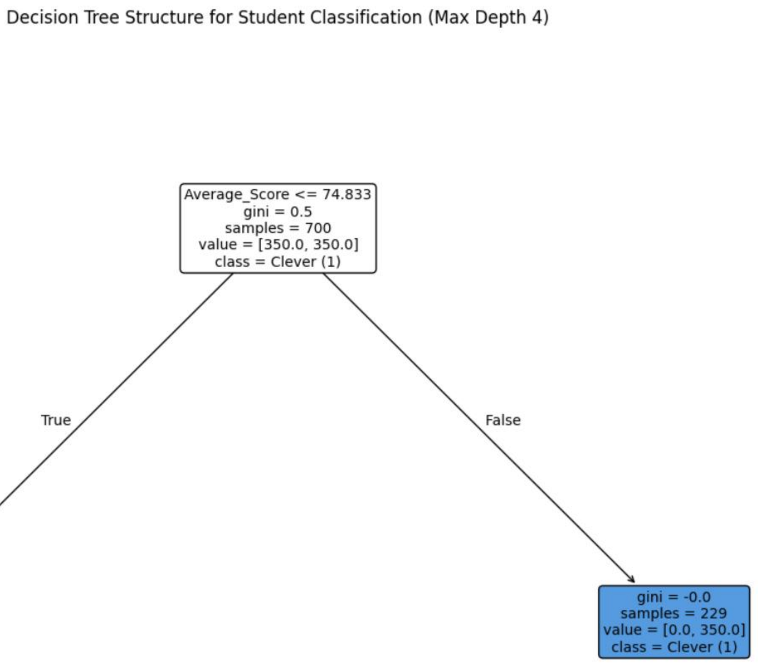
```
report_df = pd.DataFrame(report).transpose().round(4)

print("\nClassification Report Table:")
print(report_df.to_markdown())

# --- FIX: Confusion Matrix in Table Format ---
conf_matrix_df = pd.DataFrame(conf_matrix,
                        index=['Actual Average (0)', 'Actual Clever (1)'],
                        columns=['Predicted Average (0)', 'Predicted Clever (1)'])

print("\nConfusion Matrix Table:")
print(conf_matrix_df.to_markdown())

# 7. Visualization of the Decision Tree

print("\n" + "="*40)
print("7. Visualization of the Decision Tree")
print("="*40)
plt.figure(figsize=(20, 10))
plot_tree(model,
        feature_names=X.columns.tolist(),
        class_names=['Average (0)', 'Clever (1)'],
        filled=True,
        rounded=True,
        fontsize=10)
plt.title("Decision Tree Structure for Student Classification (Max Depth 4)")
plt.show()

print("\n" + "="*40)
print("Experiment Complete: Decision Tree Model Built and Evaluated.")# Step 1: Introduction to
Decision Trees
# Decision Tree Classification for student performance

# Step 2: Importing Required Libraries and Loading the Dataset
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

# Step 3: Load the dataset
data = pd.DataFrame({
   'Age': [18, 20, 22, 21, 19, 23, 24, 25, 27, 28],
   'StudyHours': [2, 3, 4, 6, 5, 2, 3, 7, 6, 4],
   'PreviousGrade': ['C', 'C', 'B', 'B', 'C', 'D', 'B', 'A', 'B', 'A'],
   'Result': [0, 0, 1, 1, 0, 1, 1, 1, 1, 1]
})

# Step 4: Data Preprocessing
# Split the dataset into features (X) and target variable (y)
X = data[['Age', 'StudyHours', 'PreviousGrade']]
y = data['Result']
```

```
# Perform one-hot encoding for the 'PreviousGrade' column


X = pd.get_dummies(X, columns=['PreviousGrade'], drop_first=True)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Step 5: Building the Decision Tree Model
# Create the Decision Tree Classifier model
model = DecisionTreeClassifier()

# Fit the model to the training data
model.fit(X_train, y_train)

# Step 6: Model Evaluation and Prediction
# Evaluate the model's performance
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
=True, rounded=True, fontsize=10)
plt.show()
```

OUTPUT:



```
Overall Accuracy: 1.0000

Classification Report Table:
|              |   precision |   recall |   f1-score |   support |
|:-------------|------------:|---------:|-----------:|----------:|
| 0            |           1 |        1 |          1 |       205 |
| 1            |           1 |        1 |          1 |        95 |
| accuracy     |           1 |        1 |          1 |         1 |
| macro avg    |           1 |        1 |          1 |       300 |
| weighted avg |           1 |        1 |          1 |       300 |

Confusion Matrix Table:
|                    |   Predicted Average (0) |   Predicted Clever (1) |
|:-------------------|------------------------:|-----------------------:|
| Actual Average (0) |                     205 |                      0 |
| Actual Clever (1)  |                       0 |                     95 |
```



Conclusion:

- The implementation of the decision tree algorithm proved effective in classifying students as average or clever based on their attributes. Decision trees provide interpretable results and can be used in various domains for classification tasks. The decision tree model offers insights into the important features contributing to the classification. This lab demonstrates the practical application of decision trees for student classification.

Quiz:

1. In decision tree classification, what is the main objective of the splitting criterion?

a) To maximize the number of features used for classification
b) To minimize the number of nodes in the decision tree
c) To maximize the accuracy of classification
d) To minimize the entropy or Gini impurity of the resulting subsets

ANS: d) To minimize the entropy or Gini impurity of the resulting subsets

2. What is the purpose of pruning in decision tree algorithms?

a) To remove irrelevant features from the dataset
b) To prevent overfitting and improve generalization of the decision tree
c) To improve the interpretability of the decision tree
d) To reduce the time complexity of the decision tree algorithm

ANS: b) To prevent overfitting and improve generalization of the decision tree

Suggested References:-

1. "Pattern Recognition and Machine Learning" by Christopher M. Bishop
2. Dinesh Kumar, Business Analytics, Wiley India Business alytics: The Science
3. V.K. Jain, Data Science & Analytics, Khanna Book Publishing, New Delhi of Dat
4. Data Science For Dummies by Lillian Pierson , Jake Porway

Rubrics wise marks obtained

Understanding of Problem	Analysis of the Problem	Capability of writing program	Documentation	Total
02	02	05	01	10