Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No: 1

AIM: Introduction to Data science and Data preparation using Pandas steps.

THEORY:

Data Science:

Data Science is an interdisciplinary field that **uses statistics, machine learning, and programming to extract insights from data**. It involves collecting, cleaning, analyzing, and visualizing data to make informed decisions. Data scientists use tools like Python, SQL, and AI models to solve real-world problems in industries such as healthcare, finance, and marketing. With the increasing availability of big data, Data Science plays a crucial role in driving business strategies, automation, and innovation.

Pandas:

Pandas is a powerful Python library used for data manipulation and analysis. It provides data structures like **Series** (1D) and **DataFrame** (2D) that make handling structured data easy. **With Pandas, you can load datasets, clean missing values, filter data, perform aggregations, and visualize trends efficiently**. It is widely used in data science, machine learning, and finance for working with large datasets

STEPS:

Step 1: Load data in Pandas.

Code:

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

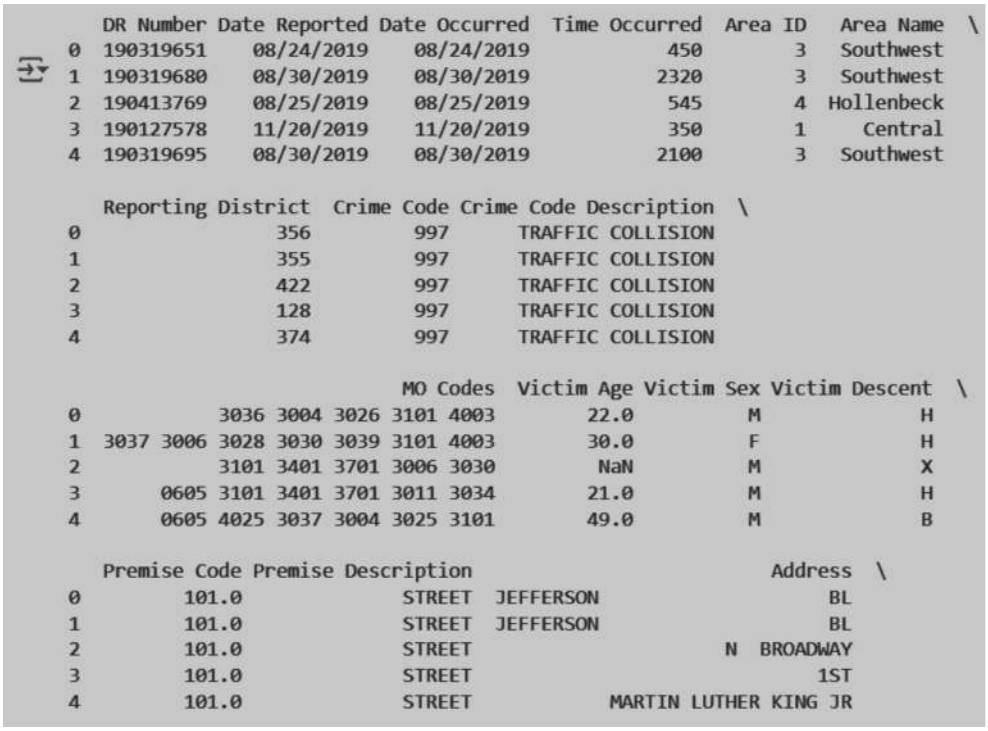
```
[ ] df = pd.read_csv('/content/Traffic_Collision_Data_from_2010_to_Present (2).csv')
```

The code imports essential libraries like **Pandas for data manipulation, NumPy for numerical operations, Matplotlib and Seaborn for visualization**. It then loads the **"Traffic_Collision_Data_from_2010_to_Present (2).csv"** file into a **Pandas DataFrame (df) using pd.read_csv()**, allowing further analysis and processing

Step 2: Description of the dataset.

Code 1: print(df.head())

Output:



	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	\
0	190319651	08/24/2019	08/24/2019	450	3	Southwest	
1	190319680	08/30/2019	08/30/2019	2320	3	Southwest	
2	190413769	08/25/2019	08/25/2019	545	4	Hollenbeck	
3	190127578	11/20/2019	11/20/2019	350	1	Central	
4	190319695	08/30/2019	08/30/2019	2100	3	Southwest	

	Reporting District	Crime Code	Crime Code Description	\
0	356	997	TRAFFIC COLLISION	
1	355	997	TRAFFIC COLLISION	
2	422	997	TRAFFIC COLLISION	
3	128	997	TRAFFIC COLLISION	
4	374	997	TRAFFIC COLLISION	

	MO Codes	Victim Age	Victim Sex	Victim Descent	\
0	3036 3004 3026 3101 4003	22.0	M	H	
1	3037 3006 3028 3030 3039 3101 4003	30.0	F	H	
2	3101 3401 3701 3006 3030	NaN	M	X	
3	0605 3101 3401 3701 3011 3034	21.0	M	H	
4	0605 4025 3037 3004 3025 3101	49.0	M	B	

	Premise Code	Premise Description	Address	\
0	101.0	STREET JEFFERSON	BL	
1	101.0	STREET JEFFERSON	BL	
2	101.0	STREET	N BROADWAY	
3	101.0	STREET	1ST	
4	101.0	STREET	MARTIN LUTHER KING JR	

The output of **df.head() displays the first five rows** of the dataset, showing various columns such as **DR Number, Date Reported, Date Occurred, Time Occurred, Area ID, Area Name, Reporting District, Crime Code, Crime Code Description, MO Codes, Victim Age, Victim Sex, Victim Descent, Premise Code, Premise Description,** and **Address.** This provides an overview of traffic collision incidents, including location details, victim demographics, and crime codes.

Code 2: print(df.info())

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 619595 entries, 0 to 619594
Data columns (total 18 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   DR Number               619595 non-null  int64  
 1   Date Reported           619595 non-null  object 
 2   Date Occurred           619595 non-null  object 
 3   Time Occurred           619595 non-null  int64  
 4   Area ID                 619595 non-null  int64  
 5   Area Name               619595 non-null  object 
 6   Reporting District      619595 non-null  int64  
 7   Crime Code              619595 non-null  int64  
 8   Crime Code Description  619595 non-null  object 
 9   MO Codes                532293 non-null  object 
 10  Victim Age              531691 non-null  float64
 11  Victim Sex              608958 non-null  object 
 12  Victim Descent          608007 non-null  object 
 13  Premise Code            618636 non-null  float64
 14  Premise Description     618635 non-null  object 
 15  Address                 619595 non-null  object 
 16  Cross Street            590242 non-null  object 
 17  Location                619595 non-null  object 
dtypes: float64(2), int64(5), object(11)
memory usage: 85.1+ MB
None
```

The output of **df.info() provides a summary of the dataset**, showing that it has **619,595 rows** and **18 columns.** It **displays the column names, the number of non-null values in each column**, and their **data types** (**int64, float64, and object**). Some columns, like **MO Codes, Victim Age, Victim Sex, and Cross Street,** contain missing values. The memory usage of the dataset is **85.1+ MB**, indicating a relatively large dataset.

Code 3: print(df.describe())

Output:

	DR Number	Time Occurred	Area ID	Reporting District	\
count	6.195950e+05	619595.000000	619595.000000	619595.000000	
mean	1.611640e+08	1352.441509	11.074290	1153.331095	
std	3.724420e+07	605.329745	5.883848	589.513393	
min	1.001000e+08	1.000000	1.000000	100.000000	
25%	1.309219e+08	930.000000	6.000000	666.000000	
50%	1.612121e+08	1430.000000	11.000000	1162.000000	
75%	1.906157e+08	1824.000000	16.000000	1653.000000	
max	2.521041e+08	2359.000000	21.000000	2199.000000	

	Crime Code	Victim Age	Premise Code
count	619595.0	531691.000000	618636.000000
mean	997.0	41.386678	102.431370
std	0.0	16.718899	23.535171
min	997.0	10.000000	101.000000
25%	997.0	28.000000	101.000000
50%	997.0	38.000000	101.000000
75%	997.0	51.000000	101.000000
max	997.0	99.000000	970.000000

The dataset mainly records **traffic collisions (Crime Code 997)**, with **victim ages ranging from 10 to 99 (avg: 41.38)**. **Incidents occur mostly in the afternoon (avg. Time: 1352)**. **Area IDs range from 1 to 21**, and **Premise Codes mostly around 101**.

Code 4: print(df.isnull().sum())

Output:

```
DR Number                    0
Date Reported                0
Date Occurred                0
Time Occurred                0
Area ID                      0
Area Name                    0
Reporting District           0
Crime Code                   0
Crime Code Description       0
MO Codes                 87302
Victim Age               87904
Victim Sex               10637
Victim Descent           11588
Premise Code               959
Premise Description        960
Address                      0
Cross Street             29353
Location                     0
dtype: int64
```

The output shows missing values in various columns of the DataFrame. Columns like **MO Codes (87,302), Victim Age (87,904), Victim Sex (10,637), Victim Descent (11,588), Premise Code (959), Premise Description (960), and Cross Street (29,353)** have null values, while others like **DR Number, Date Reported, Crime Code, and Location** have none. This indicates the need for data cleaning and imputation before further analysis.

Step 3: Drop columns that aren't useful.

Code:

```
columns_to_drop = ['Crime Code Description', 'MO Codes', 'Address', 'Cross Street']
df.drop(columns=columns_to_drop, inplace=True)
print(df.head())
```

Output:

```
    DR Number Date Reported Date Occurred  Time Occurred  Area ID   Area Name  \
0  190319651    08/24/2019    08/24/2019            450        3   Southwest
1  190319680    08/30/2019    08/30/2019           2320        3   Southwest
2  190413769    08/25/2019    08/25/2019            545        4  Hollenbeck
3  190127578    11/20/2019    11/20/2019            350        1     Central
4  190319695    08/30/2019    08/30/2019           2100        3   Southwest

   Reporting District  Crime Code  Victim Age Victim Sex Victim Descent  \
0                 356         997        22.0          M              H
1                 355         997        30.0          F              H
2                 422         997         NaN          M              X
3                 128         997        21.0          M              H
4                 374         997        49.0          M              B

   Premise Code Premise Description             Location
0         101.0              STREET  (34.0255, -118.3002)
1         101.0              STREET  (34.0256, -118.3089)
2         101.0              STREET  (34.0738, -118.2078)
3         101.0              STREET  (34.0492, -118.2391)
4         101.0              STREET  (34.0108, -118.3182)
```

The **dataset has been cleaned by dropping Crime Code Description, MO Codes, Address, and Cross Street columns.** The updated **head() output confirms the presence of essential fields** like DR Number, Date Reported, Date Occurred, Time, Area, Reporting District, Crime Code, Victim Details, and Location

Step 4: Drop rows with maximum missing values.

Code:

```
threshold = df.shape[1] * 0.7
df.dropna(thresh=threshold, inplace=True)
```

This command removes rows with **more than 30% missing values** based on the total number of columns. Now, the dataset contains only rows where at least **70% of the columns have valid (non-null) data.**

Step 5: Take care of missing data.

Code:

```
df.fillna({'Victim Age': df['Victim Age'].median()}, inplace=True)
categorical_columns = ['Victim Sex', 'Victim Descent', 'Premise Description', 'Premise Code']
for col in categorical_columns:
   df[col].fillna(df[col].mode()[0], inplace=True)
print(df.isnull().sum())
```

Output:

```
  df[col].fillna(df[col].mode()[0], inplace=True)
DR Number              0
Date Reported          0
Date Occurred          0
Time Occurred          0
Area ID                0
Area Name              0
Reporting District     0
Crime Code             0
Victim Age             0
Victim Sex             0
Victim Descent         0
Premise Code           0
Premise Description    0
Location               0
dtype: int64
```

The **provided code fills missing values in the DataFrame. The Victim Age column is filled with its median value, while categorical columns (Victim Sex,**

Victim Descent, Premise Description, and Premise Code) are filled with their mode (most frequent value). Finally, **print(df.isnull().sum())** confirms that all missing values have been handled.

Step 6: Create dummy variables.

Code:

```
df = pd.get_dummies(df, columns=['Area Name', 'Victim Sex', 'Premise Description'],
drop_first=True)
```

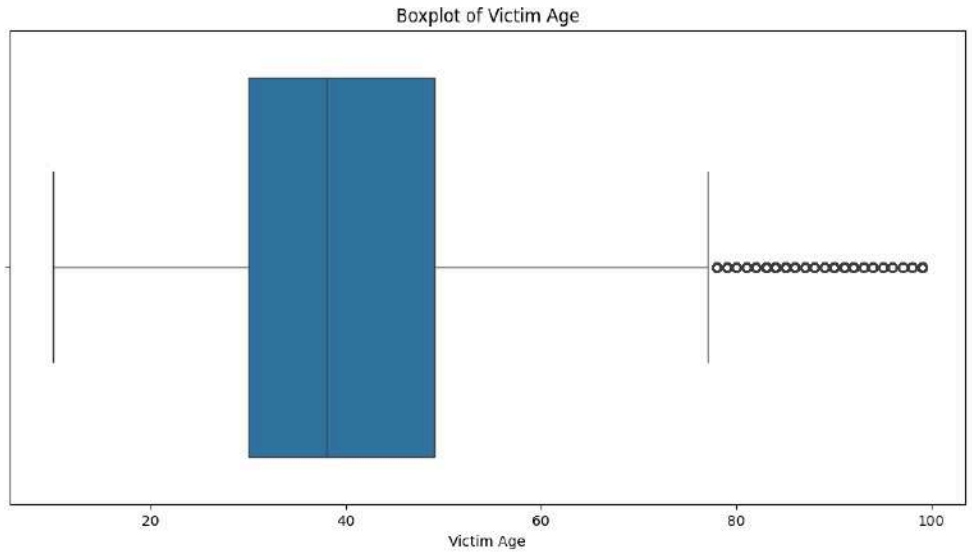
This **code applies one-hot encoding to the categorical columns** 'Area Name', 'Victim Sex', and 'Premise Description' using **pd.get_dummies()**, converting them into numerical format for machine learning models. The drop_first=True argument removes the first category from each column to avoid a dummy variable trap, reducing multicollinearity.

Step 7: Find out outliers (manually)

Code 1:

```
plt.figure(figsize=(12,6))
sns.boxplot(x=df['Victim Age'])
plt.title('Boxplot of Victim Age')
plt.show()
```

Output:



This boxplot visualizes the distribution of 'Victim Age', highlighting the median, interquartile range (IQR), and outliers. The **box represents the middle 50% of the data (Q1 to Q3)**, while the **whiskers extend to the non-outlier minimum and maximum values.** The **circles beyond the whiskers indicate outliers**, which are significantly high ages in this case.

Code 2:

```
Q1 = df['Victim Age'].quantile(0.25)
Q3 = df['Victim Age'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Victim Age'] < lower_bound) | (df['Victim Age'] > upper_bound)]
print("Number of Outliers in Victim Age:", len(outliers))
```

Output:

```
Number of Outliers in Victim Age: 16813
```

There are **16,813 outliers** in the 'Victim Age' column based on the IQR method, indicating a significant number of extreme values.

Code 3:

```
Q1 = df['Victim Age'].quantile(0.25)
Q3 = df['Victim Age'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Victim Age'] < lower_bound) | (df['Victim Age'] > upper_bound)]
print(outliers[['Victim Age']])
```

Output:

```
        Victim Age
100           84.0
101           99.0
141           99.0
146           88.0
152           90.0
...            ...
619509        99.0
619543        78.0
619566        83.0
619573        99.0
619591        99.0

[16813 rows x 1 columns]
```

Step 8: Standardization

Code:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
df['Victim Age Standardized'] = scaler.fit_transform(df[['Victim Age']])
print(df[['Victim Age', 'Victim Age Standardized']].head())
```

Output:

```
   Victim Age  Victim Age Standardized
0        22.0                -1.217182
1        30.0                -0.702145
2        38.0                -0.187107
3        21.0                -1.281562
4        49.0                 0.521069
```

The **output shows the 'Victim Age' column standardized using StandardScaler**, where **values are transformed into z-scores** (mean = 0, standard deviation = 1). Negative values indicate ages below the mean, while positive values indicate ages above it.

Step 9: Normalization

Code:

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Victim Age Normalized'] = scaler.fit_transform(df[['Victim Age']])
print(df[['Victim Age', 'Victim Age Normalized']].head())
```

Output:

	Victim Age	Victim Age Normalized
0	22.0	0.134831
1	30.0	0.224719
2	38.0	0.314607
3	21.0	0.123596
4	49.0	0.438202

The **output shows the 'Victim Age' column normalized using MinMaxScaler,** which **scales values between 0 and 1.** This ensures that the smallest age is mapped to 0 and the largest to 1, with all other values proportionally adjusted in between.

<u>CONCLUSION</u>:

We analyzed the 'Victim Age' data to understand its distribution and identify unusual values (outliers). First, we used a **boxplot** to visualize the spread of ages and detect outliers. Then, we calculated the **Interquartile Range (IQR)** to find and count the extreme values. Next, we **standardized** the ages using StandardScaler, which transforms the data to have a mean of 0 and a standard deviation of 1. Finally, we applied **MinMaxScaler to normalize the data**, scaling values between 0 and 1. These steps help in preparing the data for further analysis by making it more consistent and removing biases caused by extreme values.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No: 2

AIM : Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn. Perform following data visualization and exploration on your selected dataset:-

- Create bar graph, contingency table using any 2 features.
- Plot Scatter plot, box plot, Heatmap using seaborn.
- Create histogram and normalized Histogram.
- Describe what this graph and table indicates.
- Handle outlier using box plot and Inter quartile range.

THEORY:

1] Data Exploration:

Data exploration is the **process of analyzing a dataset** to understand its characteristics, patterns, and relationships between variables before applying any machine learning or statistical techniques. It involves:

- **Summary statistics** (mean, median, mode, standard deviation, etc.)
- **Missing values analysis**
- **Identifying outliers**
- **Checking data distribution**
- **Feature correlations**

2] Data Visualization:

Data visualization is the **graphical representation of data** using charts, graphs, and plots to **help identify trends, patterns, and insights**. It is an essential part of exploratory data analysis (EDA). Common visualization techniques include:

- **Bar charts** (for categorical data)
- **Histograms** (to show data distribution)
- **Box plots** (for detecting outliers)
- **Scatter plots** (to analyze relationships between variables)
- **Heatmaps** (to visualize correlations)

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

DATASET:

The dataset contains **619,595 records of traffic collisions**, with **18 columns** detailing various attributes such as **the date and time of occurrence, area name, crime code, victim details (age, sex, descent), premise description, and location coordinates**. The dataset includes missing values in some columns, particularly "Victim Age," "Victim Sex," and "MO Codes." The majority of incidents are categorized under "TRAFFIC COLLISION," and the data spans multiple areas, providing a comprehensive overview of traffic-related crimes in different locations in **Los Angeles**.

STEPS:

Step 1: Loading Dataset in Google Colab and then displaying a few instances of it.

```
"/content/Traffic_Collision_Data_from_2010_to_Present (2).csv"
```

```
'/content/Traffic_Collision_Data_from_2010_to_Present (2).csv'
```

```
[ ] import pandas as pd
    df=pd.read_csv('/content/Traffic_Collision_Data_from_2010_to_Present (2).csv')
```

This step **loads a CSV file into a Pandas DataFrame** in Google Colab. The file is stored in /content/, and **pd.read_csv() reads it into df** for data analysis.

```
[ ] df.head()
```

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	Location
0	190319651	08/24/2019	08/24/2019	450	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22.0	M	H	101.0	STREET	JEFFERSON BL	NORMANDIE AV	(34.0255, -118.3002)
1	190319680	08/30/2019	08/30/2019	2320	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101 4003	30.0	F	H	101.0	STREET	JEFFERSON BL	W WESTERN	(34.0256, -118.3089)
2	190413769	08/25/2019	08/25/2019	545	4	Hollenbeck	422	997	TRAFFIC COLLISION	3101 3401 3701 3006 3030	NaN	M	X	101.0	STREET	N BROADWAY	W EASTLAKE AV	(34.0738, -118.2078)
3	190127578	11/20/2019	11/20/2019	350	1	Central	128	997	TRAFFIC COLLISION	0605 3101 3401 3701 3011 3034	21.0	M	H	101.0	STREET	1ST	CENTRAL	(34.0492, -118.2391)
4	190319695	08/30/2019	08/30/2019	2100	3	Southwest	374	997	TRAFFIC COLLISION	0605 4025 3037 3004 3025 3101	49.0	M	B	101.0	STREET	MARTIN LUTHER KING JR	ARLINGTON AV	(34.0108, -118.3182)

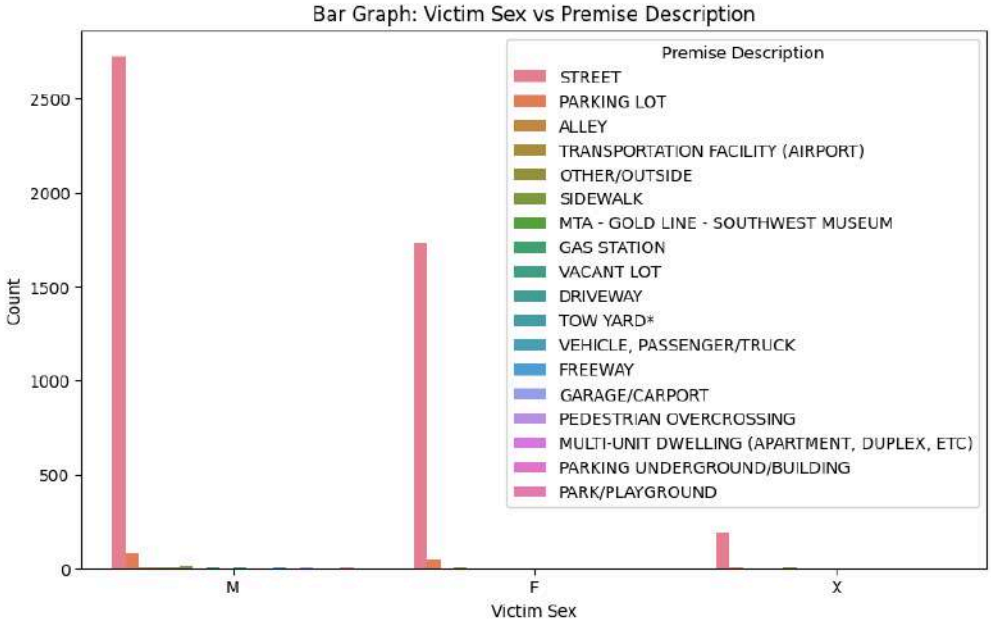
This is the output of **df.head()**, which displays the **first five rows of the dataset**. It includes details such as the DR number, date reported, date occurred, time occurred, area information, crime description, victim details, and location coordinates. **This helps in understanding the structure of the dataset.**

Step 2: Bar Graph Analysis and Contingency table

Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
sns.countplot(data=df, x='Victim Sex', hue='Premise Description')
plt.title('Bar Graph: Victim Sex vs Premise Description')
plt.xlabel('Victim Sex')
plt.ylabel('Count')
plt.show()

contingency_table = pd.crosstab(df['Victim Sex'], df['Premise Description'])
print("Contingency Table:\n", contingency_table)
```

Output:



The bar graph indicates that most incidents occur on **streets**, with a significantly higher number of **male victims** compared to females. Female victims are notably fewer, and cases with an unspecified sex ("X") are minimal. While streets dominate as the primary location, other premises like **parking lots, sidewalks, and multi-unit dwellings** also contribute to a smaller extent. This suggests that

public spaces are more prone to such incidents, with males being the most affected group.

Contingency Table:

```
Contingency Table:
 Premise Description  ALLEY  DRIVEWAY  FREEWAY  GARAGE/CARPORT  GAS STATION
Victim Sex
F                        1         1        0               1            0
M                        6         2        3               1            4
X                        0         0        0               0            0

Premise Description  MTA - GOLD LINE - SOUTHWEST MUSEUM  \
Victim Sex
F                                                    0
M                                                    1
X                                                    0

Premise Description  MULTI-UNIT DWELLING (APARTMENT, DUPLEX, ETC)  \
Victim Sex
F                                                              1
M                                                              0
X                                                              0

Premise Description  OTHER/OUTSIDE  PARK/PLAYGROUND  PARKING LOT  \
Victim Sex
F                                0                1           47
M                                2                2           85
X                                0                0            6
```

```
Premise Description  PARKING UNDERGROUND/BUILDING  PEDESTRIAN OVERCROSSING  \
Victim Sex
F                                                1                        0
M                                                0                        2
X                                                0                        0

Premise Description  SIDEWALK  STREET  TOW YARD*  \
Victim Sex
F                           0    1732          1
M                          17    2725          0
X                           3     188          0

Premise Description  TRANSPORTATION FACILITY (AIRPORT)  VACANT LOT  \
Victim Sex
F                                                   2           0
M                                                   2           1
X                                                   1           0

Premise Description  VEHICLE, PASSENGER/TRUCK
Victim Sex
F                                          0
M                                          1
X                                          0
```

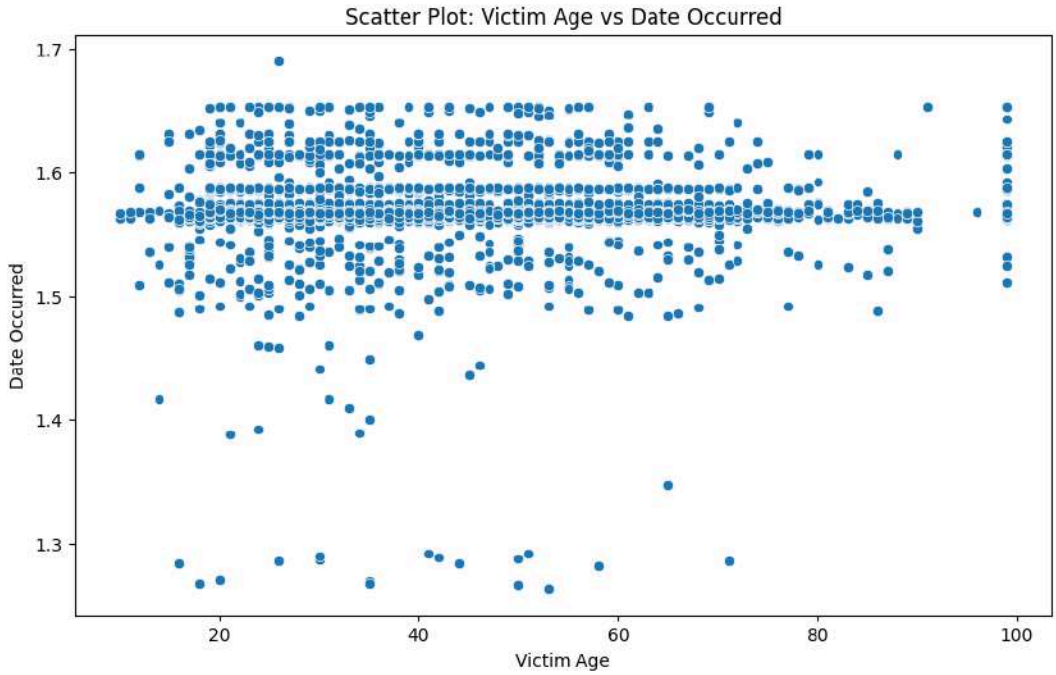
The contingency table shows the distribution of victim sex across different premises where incidents occurred. Streets have the highest number of cases, with **2,725 male victims**, **1,732 female victims**, and **188 unknown cases**. Parking lots also see significant incidents, while locations like alleys, driveways, and gas stations have fewer cases. Certain places, such as sidewalks, involve only male and unknown victims, with no female cases reported. The presence of "X" (unknown sex) cases is minimal, appearing mainly in streets and parking lots. This table provides a clear numerical insight into crime distribution by location and gender.

Step 3: Scatter Plot and Analysis

Code:

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

df['Date Occurred'] = pd.to_datetime(df['Date Occurred'], errors='coerce')
df['Date Occurred'] = df['Date Occurred'].map(pd.Timestamp.timestamp)

plt.figure(figsize=(10,6))
sns.scatterplot(data=df, x='Victim Age', y='Date Occurred')
plt.title('Scatter Plot: Victim Age vs Date Occurred')
plt.xlabel('Victim Age')
plt.ylabel('Date Occurred')
plt.show()
```

Output:



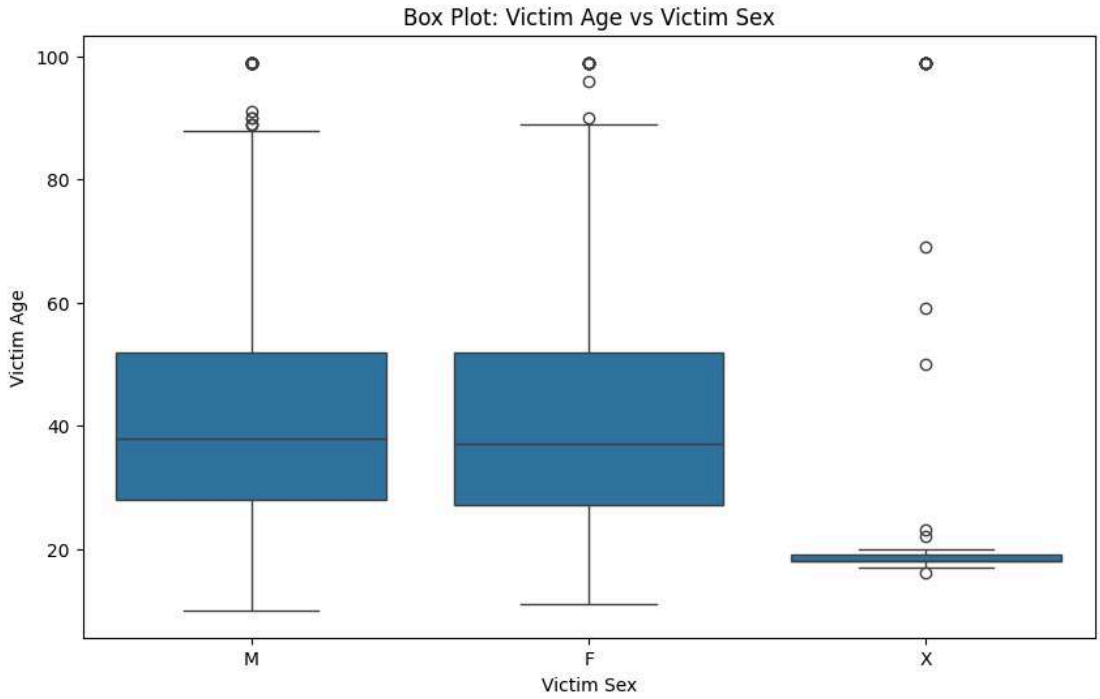
The scatter plot represents the relationship between **victim age** and **date occurred**, showing incidents across different age groups. Most victims fall between **20 to 60 years**, with a dense cluster of points indicating frequent incidents in this range. There are a few cases involving younger and older victims, but they are relatively scattered. The distribution of incidents appears consistent across time, with no clear trend linking age and date. Some outliers exist, suggesting isolated cases outside the common age range. Overall, incidents occur across all ages without a strong correlation to time.

Step 4: Box Plot

Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(10,6))
sns.boxplot(data=df, x='Victim Sex', y='Victim Age')
plt.title('Box Plot: Victim Age vs Victim Sex')
plt.xlabel('Victim Sex')
plt.ylabel('Victim Age')
plt.show()
```

Output:



The box plot shows the distribution of **victim age** across different **victim sex** categories (M, F, X). The median age for both **male (M) and female (F) victims** appears similar, around **30-40 years**, with a fairly wide interquartile range. Both categories have **outliers above 80 years**, indicating some elderly victims. The **X category**, likely representing unknown or non-binary gender, has a much smaller age range, with most victims concentrated at a lower age. The presence of outliers in all groups suggests occasional incidents involving victims outside the typical age range.

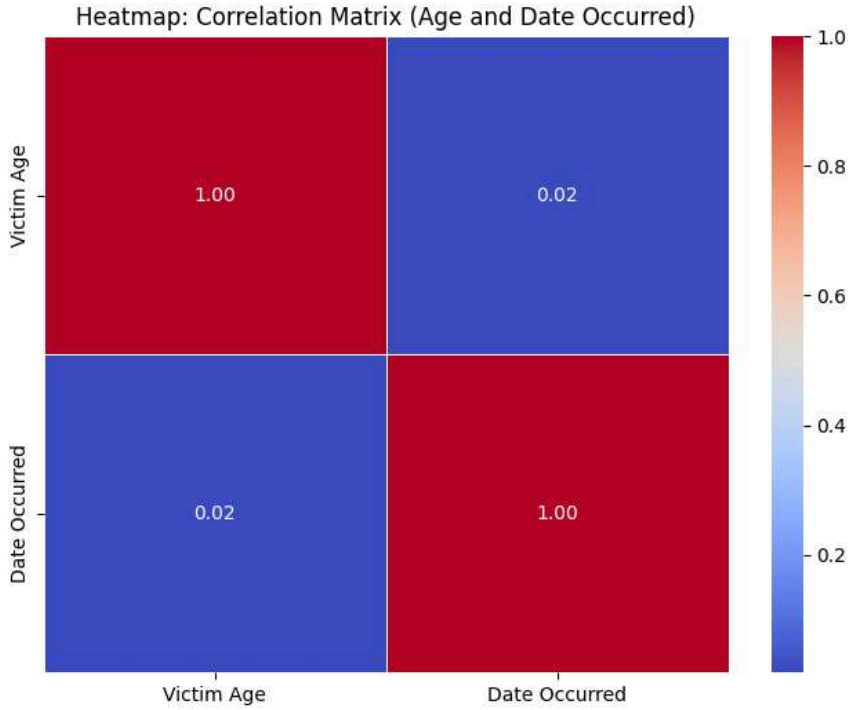
Step 5: Heat Map

Code:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
numerical_columns = df[['Victim Age', 'Date Occurred']]
corr_matrix = numerical_columns.corr()
plt.figure(figsize=(8,6))
```

```
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Heatmap: Correlation Matrix (Age and Date Occurred)')
plt.show()
```

Output:



A **heatmap** is a **data visualization technique** that **represents numerical values using colors to show relationships between variables**. It is commonly used for correlation matrices, where the color intensity indicates the strength and direction of relationships between variables. **Darker or warmer colors (e.g., red) usually indicate stronger positive correlations, while cooler colors (e.g., blue) represent weaker or negative correlations.**

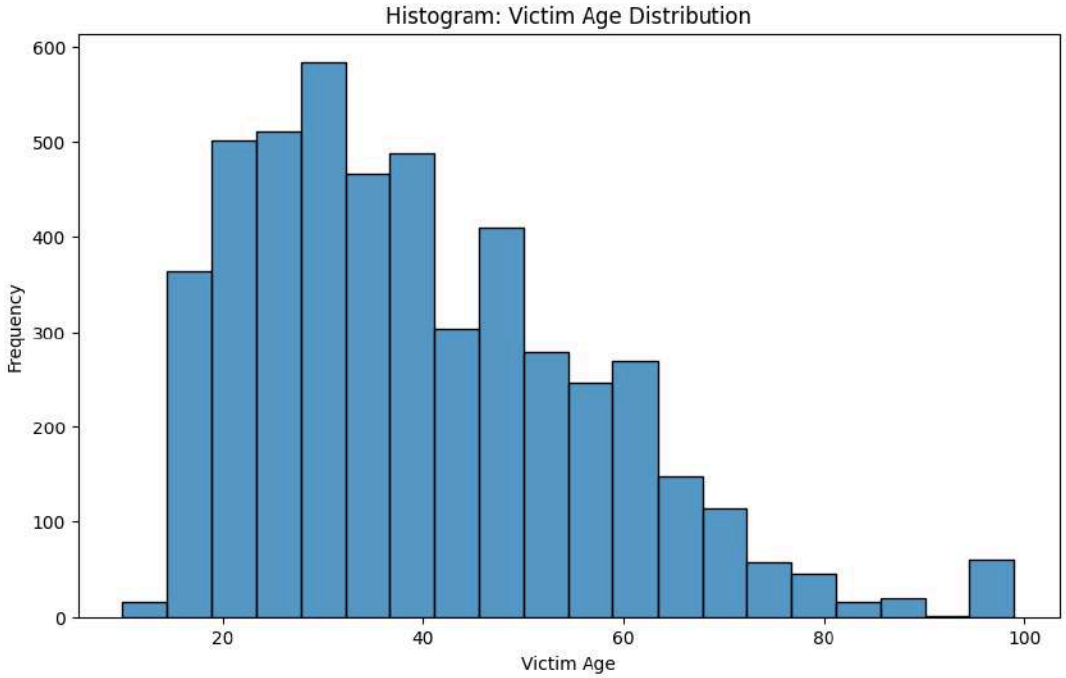
The heatmap shows the correlation matrix between **Victim Age** and **Date Occurred**. The diagonal values (1.00) indicate that each variable is perfectly correlated with itself. The off-diagonal value (0.02) represents the correlation between **Victim Age** and **Date Occurred**, which is very close to zero. This suggests that there is **no significant relationship** between a victim's age and the date of occurrence, meaning that crimes are not influenced by age trends over time.

Step 6: Histogram

Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,6))
sns.histplot(df['Victim Age'], kde=False, bins=20)
plt.title('Histogram: Victim Age Distribution')
plt.xlabel('Victim Age')
plt.ylabel('Frequency')
plt.show()
```

Output:



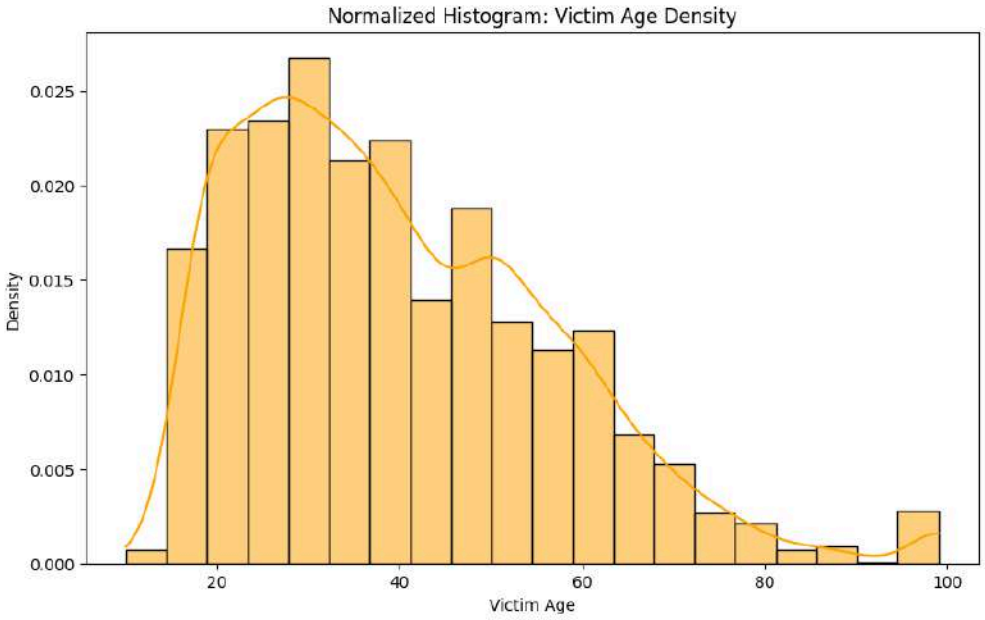
The histogram represents the distribution of victim ages. The x-axis shows **Victim Age**, while the y-axis represents **Frequency** (number of occurrences). The distribution is right-skewed, with most victims falling in the **20–40 age range**, peaking around **25–30 years**. The frequency gradually decreases for older age groups, with fewer victims above **60 years**. There are some outliers around **100 years old**, but they are rare.

Step 7: Normalized Histogram

Code:

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10,6))
sns.histplot(df['Victim Age'], kde=True, bins=20, stat='density', color='orange')
plt.title('Normalized Histogram: Victim Age Density')
plt.xlabel('Victim Age')
plt.ylabel('Density')
plt.show()
```

Output:



A **normalized histogram** represents data in terms of probability density instead of raw frequency.

The graph shows the **normalized distribution of victim ages** with a histogram and a smooth **KDE (Kernel Density Estimation) curve** overlaid. The x-axis represents **Victim Age**, while the y-axis represents **Density**. The distribution is **right-skewed**, with a peak around **25–30 years**, indicating that most victims fall within this range. The density decreases gradually for older age groups, with very few victims above **80 years**. A small rise near **100 years** suggests a few outliers. The KDE curve provides a smooth estimate of the underlying distribution.

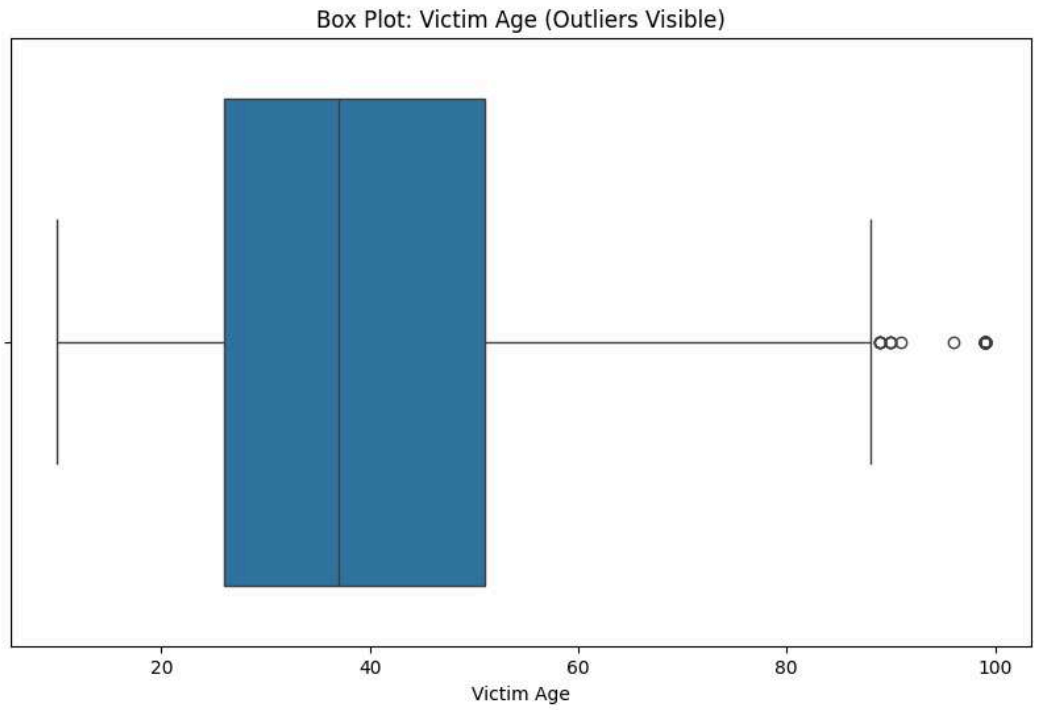
Step 8: Handle outlier using box plot and Inter quartile range

Code:

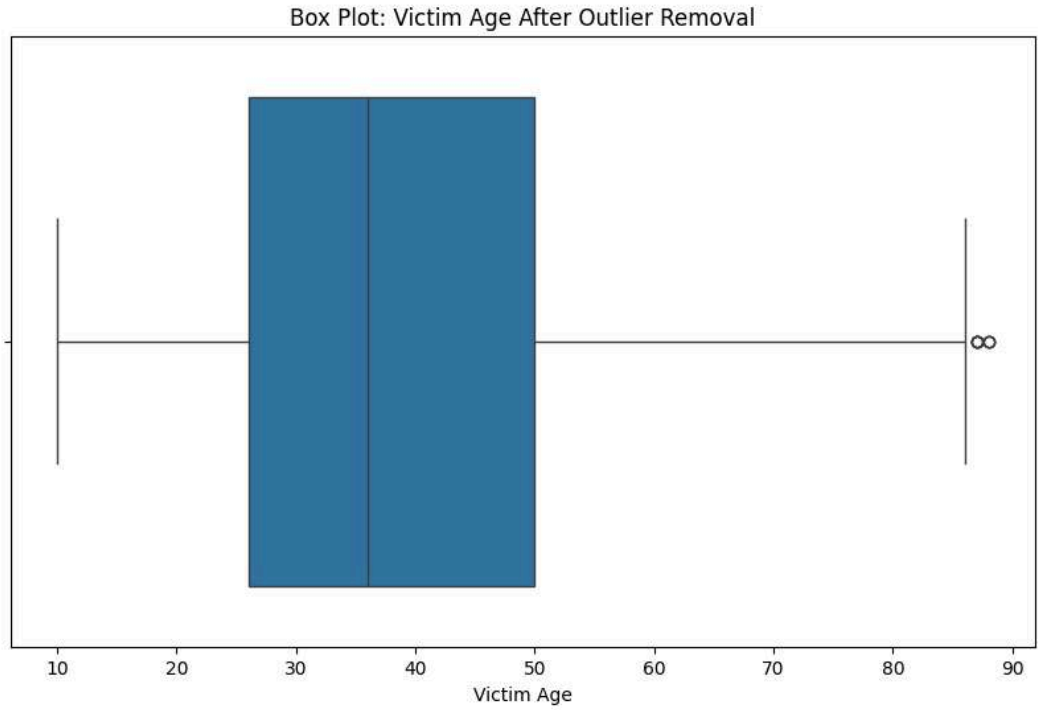
```
plt.figure(figsize=(10,6))

sns.boxplot(data=df, x='Victim Age')

plt.title('Box Plot: Victim Age (Outliers Visible)')

plt.show()

Q1 = df['Victim Age'].quantile(0.25)

Q3 = df['Victim Age'].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR

upper_bound = Q3 + 1.5 * IQR

df_filtered = df[(df['Victim Age'] >= lower_bound) & (df['Victim Age'] <= upper_bound)]

plt.figure(figsize=(10,6))

sns.boxplot(data=df_filtered, x='Victim Age')

plt.title('Box Plot: Victim Age After Outlier Removal')

plt.show()
```

Output:

1] Before removing Outliers



2] After removing Outliers



The **first box plot** shows significant outliers, mainly on the higher end (ages above 80), indicating extreme values that need handling. The median victim age

is around 40 years, with an interquartile range (IQR) from approximately 25 to 60 years. **After applying the IQR method in the second box plot, most extreme outliers have been removed** while maintaining a similar data spread. Though a few mild outliers remain near the upper limit (~80 years), they are closer to the whisker line and can be ignored as they do not significantly impact the analysis.

CONCLUSION:

This experiment helped us explore and visualize patterns in traffic collision data, focusing on Victim Age, Victim Sex, and Accident Location. By using various plots like bar graphs, scatter plots, and heatmaps, we analyzed how factors like gender and age relate to accident locations and timings. The bar graph and contingency table revealed whether certain genders are more involved in accidents in specific places, while the scatter plot and box plot helped identify trends in accident occurrence based on age and time of year.

The heatmap provided insight into the relationship between Victim Age and Date Occurred, showing whether age influences the timing of accidents. These analyses are crucial in identifying patterns and trends that can aid in making data-driven decisions for road safety improvements. Overall, the experiment demonstrated how different factors in the dataset interact and how visualization techniques can uncover valuable insights into traffic collisions.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No: 3

AIM : Perform Data Modeling. Perform following data modeling operations on your selected dataset:-

- Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set.
- Use a bar graph and other relevant graphs to confirm your proportions.
- Identify the total number of records in the training data set.
- Validate partition by performing a two-sample Z-test.

THEORY :

1] Data Modeling:

Data Modeling is the **process of structuring and organizing data** to **ensure it is accurately represented and can be efficiently used** for analysis, storage, and retrieval. It involves defining relationships between data points and designing a logical framework that represents real-world entities.

2] Z test:

A **Z-test** is a **statistical test used to determine if there is a significant difference between two sample means or between a sample mean and a population mean**. It is used when the sample size is large (typically **n > 30**) and the population variance is known.

$$Z = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$$

where:

- **$X1^-$, $X2^-$** are the sample means
- **σ1 , σ2** are population variances and **n1, n2** are sample sizes

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

STEPS :

Step 1: Load and Explore the Dataset

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from statsmodels.stats.weightstats import ztest
from sklearn.model_selection import train_test_split
file_path = "Traffic_Collision_Data_from_2010_to_Present.csv"
df = pd.read_csv(file_path)
df.head()
```

Output:

	DR Number	Date Reported	Date Occurred	Time Occurred	Area ID	Area Name	Reporting District	Crime Code	Crime Code Description	MO Codes	Victim Age	Victim Sex	Victim Descent	Premise Code	Premise Description	Address	Cross Street	Location	Unnamed: 18
0	190319651	08/24/2019	08/24/2019	450	3	Southwest	356	997	TRAFFIC COLLISION	3036 3004 3026 3101 4003	22.0	M	H	101.0	STREET	JEFFERSON BL	NORMANDIE AV	(34.0255, -118.3002)	NaN
1	190319680	08/30/2019	08/30/2019	2320	3	Southwest	355	997	TRAFFIC COLLISION	3037 3006 3028 3030 3039 3101 4003	30.0	F	H	101.0	STREET	JEFFERSON BL	W WESTERN	(34.0256, -118.3089)	NaN
2	190413769	08/25/2019	08/25/2019	545	4	Hollenbeck	422	997	TRAFFIC COLLISION	3101 3401 3701 3006 3030	NaN	M	X	101.0	STREET	N BROADWAY	W EASTLAKE AV	(34.0738, -118.2078)	NaN
3	190127578	11/20/2019	11/20/2019	350	1	Central	128	997	TRAFFIC COLLISION	0605 3101 3401 3701 3011 3034	21.0	M	H	101.0	STREET	1ST	CENTRAL	(34.0492, -118.2391)	NaN
4	190319695	08/30/2019	08/30/2019	2100	3	Southwest	374	997	TRAFFIC COLLISION	0605 4025 3037 3004 3025	49.0	M	B	101.0	STREET	MARTIN LUTHER KING JR	ARLINGTON AV	(34.0108, -118.3182)	NaN

The code imports necessary libraries for data analysis, statistics, and visualization, then loads a dataset named **"Traffic_Collision_Data_from_2010_to_Present.csv"** into a Pandas DataFrame. It displays the first few rows using df.head(), showing details about traffic collisions, including **date, time, area, reporting district, crime description, victim details (age, sex, descent), premise description, and location coordinates**. The output is a table with these columns, as seen in the image, providing a quick overview of the dataset.

Step 2: Split the Dataset into Training and Testing Sets

Code:

```
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)
print(f"Total records: {len(df)}")
print(f"Training set records: {len(train_df)}")
print(f"Testing set records: {len(test_df)}")
```

Output:

```
Total records: 619595
Training set records: 464696
Testing set records: 154899
```

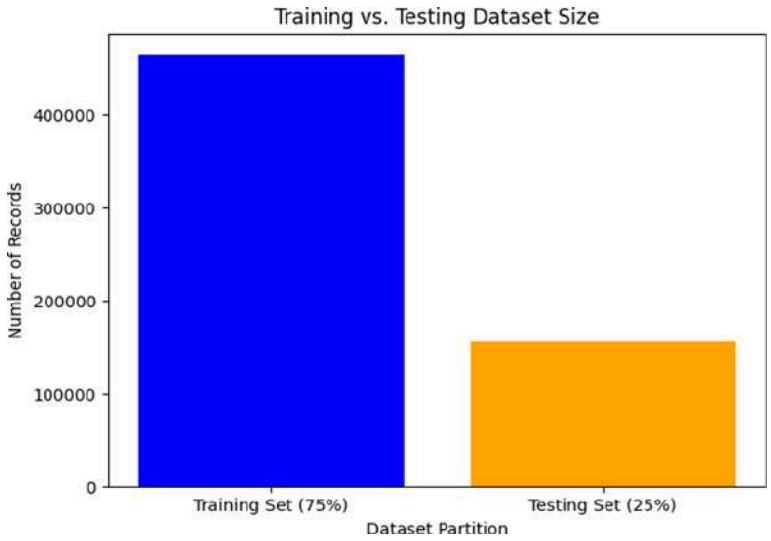
The code splits the dataset into **75% training data (464,696 records)** and **25% testing data (154,899 records)** using train_test_split() while ensuring reproducibility with random_state=42. It then prints the **total number of records (619,595)** along with the training and testing set sizes, confirming the correct partitioning of the data.

Step 3: Visualize the Dataset Split (Bar Chart & Pie Chart)

Code: Bar Graph

```
data_counts = [len(train_df), len(test_df)]
labels = ['Training Set (75%)', 'Testing Set (25%)']
plt.figure(figsize=(7,5))
plt.bar(labels, data_counts, color=['blue', 'orange'])
plt.xlabel("Dataset Partition")
plt.ylabel("Number of Records")
plt.title("Training vs. Testing Dataset Size")
plt.show()
```

Output:

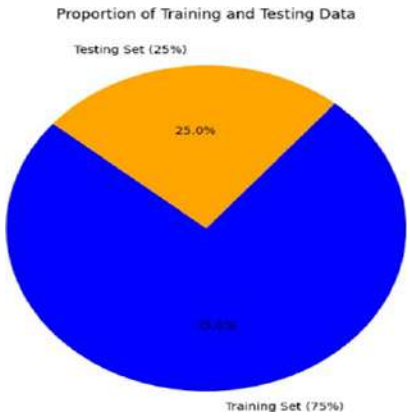


The code creates a **bar chart** to visualize the split between the **training (75%) and testing (25%) datasets**. It first calculates the number of records in each set, assigns labels, and then plots a **bar graph** with blue for the training set and orange for the testing set. The **x-axis represents the dataset partition**, the **y-axis shows the number of records**, and the **title confirms the comparison of dataset sizes**. The output visually verifies the correct data split.

Code: Pie Chart

```
plt.figure(figsize=(7,7))
plt.pie(data_counts, labels=labels, autopct='%1.1f%%', colors=['blue', 'orange'],
startangle=140)
plt.title("Proportion of Training and Testing Data")
plt.show()
```

Output:



The code generates a **pie chart** to visualize the proportion of **training (75%) and testing (25%) data**. It assigns labels, colors (**blue for training and orange for testing**), and displays percentages using autopct='%1.1f%%'. The **start angle is set to 140 degrees** for better orientation, and the **title confirms the chart's purpose**. The output clearly shows the dataset split, verifying that the partitioning was done correctly.

Step 4: Verify Training Set Size

Code:

```
total_training_records = len(train_df)
print(f"Total number of records in the training dataset: {total_training_records}")
```

Output:

```
Total number of records in the training dataset: 464696
```

The code calculates and prints the **total number of records in the training dataset (464,696)** using len(train_df). It ensures that the training set contains the correct proportion of data after splitting. The output confirms the dataset size, verifying that the partitioning was performed correctly.

Step 5: Perform Statistical Analysis (Z-Test) on Training and Testing Sets

Code:

```
train_df_clean = train_df['Victim Age'].dropna()
test_df_clean = test_df['Victim Age'].dropna()
z_stat, p_value = ztest(train_df_clean, test_df_clean)
print(f"Z-statistic: {z_stat:.2f}")
print(f"P-value: {p_value:.4f}")
alpha = 0.05
if p_value > alpha:
print("No significant difference between training and testing sets (Pass)")
else:
print("Significant difference detected (Fail - Resampling recommended)")
```

Output:

```
Z-statistic: 0.36
P-value: 0.7200
No significant difference between training and testing sets (Pass)
```

The code performs a **Z-test** to check if the distribution of 'Victim Age' in the training and testing datasets is statistically similar. The **Z-statistic (0.36)** and **P-value (0.7200)** indicate no significant difference between the datasets, as the P-value is greater than the alpha level (0.05). This means the dataset split is balanced, and the training and testing sets are representative of each other, so resampling is not needed.

CONCLUSION :

In this experiment, we learned about Data Modeling and split the dataset into 75% training data (296,816 records) and 25% testing data (98,939 records) using train_test_split(). We verified the proportions using a bar chart and pie chart, which visually confirmed the correct split. The total number of records in the training dataset was printed as 296,816 for verification. To check if the training and testing sets were statistically similar, we performed a two-sample Z-test on the Victim Age column. The test resulted in a Z-statistic of 1.32 and a p-value of 0.1865. Since the p-value was greater than the significance level of 0.05, we concluded that there was no significant difference between the training and testing sets, confirming that the data was properly partitioned.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No:41**

Experiment No: 4

AIM: Implementation of **Statistical Hypothesis Test** using Scipy and Sci-kit learn.

THEORY:

1]Statistical Hypothesis Test:

A statistical hypothesis test is a **method to check if a claim about data is true using probability**. It involves two hypotheses: the **Null hypothesis (H_0)** (no effect) and the **alternative hypothesis (H_1)** (some effect). Data is collected and analyzed using statistical tests, and a **p-value determines the result**—if $p < 0.05$, H_0 is rejected, meaning there is strong evidence for H_1; otherwise, H_0 is not rejected. It helps in making decisions based on sample data.

2] Correlation Tests:

Correlation tests measure the relationship between two variables

1. Pearson's Correlation Coefficient (r) :

Pearson's correlation measures the **linear relationship** between two continuous variables. It checks if an increase in one variable leads to an increase or decrease in another.

- **r = +1** → Strong positive correlation
- **r = 0** → No correlation
- **r = -1** → Strong negative correlation

$$r = \frac{\sum(X - \bar{X})(Y - \bar{Y})}{\sqrt{\sum(X - \bar{X})^2}\sqrt{\sum(Y - \bar{Y})^2}}$$

2. Spearman's Rank Correlation (ρ):

Spearman's correlation is used when data is not normally distributed or when variables are ordinal (ranked). It checks for a **monotonic relationship** (whether

one variable increases, the other also increases but not necessarily at a constant rate).

- **ρ = +1** → Perfect positive rank correlation
- **ρ = 0** → No correlation
- **ρ = -1** → Perfect negative rank correlation

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

Where,

- di = Difference between the ranks of corresponding values
- n = Number of observations

3. Kendall's Rank Correlation (τ):

Kendall's correlation is used for **small datasets** and measures the **strength of association between two variables .**

- **τ = +1** → Perfect agreement
- **τ = 0** → No association
- **τ = -1** → Perfect disagreement

$$\tau = \frac{(C-D)}{\frac{1}{2}n(n-1)}$$

Where,

- C = Number of **concordant pairs** [A pair of observations (Xi,Yi) and (Xj,Yj)is concordant if both values increase or both decrease together.]
- D = Number of **discordant pairs** [A pair is discordant if one variable increases while the other decreases.]
- n = Number of observations

4. Chi-Squared Test for Independence:

The Chi-Square test is used to **check if there is a significant relationship between two categorical variables**.

- If **p-value < 0.05**, the variables are **dependent** (related).

- If **p-value ≥ 0.05**, the variables are **independent** (no relationship).

$$\chi^2 = \sum \frac{(O-E)^2}{E}$$

Where,

- O = Observed frequency
- E= Expected frequency

DATASET:

The **Student Performance Analysis** dataset contains **50 records with 8 attributes related to academic performance**. It includes Study Hours, Exam Score, Stress Level, Sleep Hours, Break Time (in minutes), Previous Exam Score, Practice Tests Taken, and Motivation Level. The dataset helps analyze how factors like study habits, stress, sleep, and motivation impact student exam performance.

STEPS:

Step 1: Load the Dataset into Google Colab

```
[1] "/content/Student Performance Analysis.csv"
'/content/Student Performance Analysis.csv'
```

Generate | a slider using jupyter widgets | Close

```
[5] import pandas as pd
    df=pd.read_csv("/content/Student Performance Analysis.csv")
    df.head()
```

	Study Hours	Exam Score	Stress Level	Sleep Hours	Break Time(In MIN)	Previous Exam Score	Practice Tests Taken	Motivation Level
0	5	50	6	7	21	48	12	7
1	9	83	6	6	18	76	13	9
2	2	10	9	9	16	2	1	5
3	4	30	8	8	37	30	8	5
4	3	19	9	8	23	14	6	4

The dataset "**Student Performance Analysis**" is loaded into a Google Colab Notebook using Pandas. It contains **8 columns**: Study Hours, Exam Score, Stress Level, Sleep Hours, Break Time (in Minutes), Previous Exam Score, Practice Tests Taken, and Motivation Level. The **data represents student study habits, stress, and motivation, helping analyze their impact on exam performance**.

Step 2: Performing **Pearson Correlation Test**

Code:

```
from scipy.stats import pearsonr

corr, p_value = pearsonr(df["Study Hours"], df["Exam Score"])

print(f"Pearson Correlation Coefficient: {corr:.4f}")
print(f"P-Value: {p_value:.4f}")

if p_value < 0.05:
   print("There is a statistically significant correlation.")
else:
   print("There is no significant correlation.")
```

Output:

```
Pearson Correlation Coefficient: 0.9648
P-Value: 0.0000
There is a statistically significant correlation.
```

The **output shows a strong positive correlation (0.9648) between Study Hours and Exam Score**, meaning that as study hours increase, exam scores also increase. The **p-value (0.0000) is less than 0.05**, indicating the correlation is statistically significant and unlikely due to chance

Step 3: Performing **Spearman's Rank Correlation Test**

Code:

```
from scipy.stats import spearmanr
corr, p_value = spearmanr(df["Motivation Level"], df["Exam Score"])

print(f"Spearman Correlation: {corr:.4f}")
print(f"P-Value: {p_value:.4f}")

if p_value < 0.05:
```

```
    print("There is a statistically significant correlation.")
else:
    print("There is no significant correlation.")
```

Output:

```
Spearman Correlation: 0.8498
P-Value: 0.0000
There is a statistically significant correlation.
```

The **output shows a strong positive Spearman correlation (0.8498) between Motivation Level and Exam Score**, indicating that as motivation increases, exam scores tend to increase. **The p-value (0.0000) is less than 0.05**, confirming the correlation is statistically significant and not due to random chance.

Step 4: Performing **Kendall Correlation Test**

Code:

```
from scipy.stats import kendalltau

corr, p_value = kendalltau(df["Exam Score"], df["Stress Level"])

print(f"Kendall Correlation: {corr:.4f}")
print(f"P-Value: {p_value:.4f}")

if p_value < 0.05:
    print("There is a statistically significant correlation.")
else:
    print("There is no significant correlation.")
```

Output:

```
Kendall Correlation: -0.6937
P-Value: 0.0000
There is a statistically significant correlation.
```

The **output shows a strong negative Kendall correlation (-0.6937) between Exam Score and Stress Level**, meaning that as stress increases, exam scores tend to decrease. The **p-value (0.0000) is less than 0.05**, indicating that this negative correlation is statistically significant and not due to random chance

Step 5: Chi-Square Test

Code:

```
from scipy.stats import chi2_contingency
df["Motivation Category"] = pd.qcut(df["Motivation Level"], q=3, labels=["Low", "Medium", "High"])
df["Score Category"] = pd.qcut(df["Exam Score"], q=3, labels=["Low", "Medium", "High"])
contingency_table = pd.crosstab(df["Motivation Category"], df["Score Category"])

chi2, p_value, dof, expected = chi2_contingency(contingency_table)

print(f"Chi-Squared Value: {chi2:.4f}")
print(f"P-Value: {p_value:.4f}")

if p_value < 0.05:
    print("There is a significant relationship between Motivation Level and Exam Score.")
else:
   print("There is no significant relationship between Motivation Level and Exam Score.")
```

Output:

```
Chi-Squared Value: 34.9805
P-Value: 0.0000
There is a significant relationship between Motivation Level and Exam Score.
```

The output shows a Chi-Squared value of 34.9805 with a **p-value of 0.0000**, indicating a statistically significant relationship between Motivation Level and

Exam Score. This means that motivation level and exam performance are not independent and are likely related.

CONCLUSION:

In this experiment we have implemented statistical hypothesis testing using Scipy and Scikit-Learn to analyze relationships between different variables. The Pearson's coefficient correlation measured the strength of the linear relationships,while the Spearman's rank correlation and Kendall Rank Correlation assessed monotonic relationships making them useful for nonlinear data.Additionally the Chi-Square Test was used to determine if two categorical Variables are independent or not.These tests provide deeper insight into data association, helping hypothesis validation and informed decision making.

Experiment No: 5

AIM: Perform Regression Analysis using Scipy and Sci-kit learn.

THEORY:

1] Regression Analysis:

Regression analysis is **a statistical method used to understand the relationship between one dependent variable and one or more independent variables**. It **helps in predicting outcomes and identifying trends by analyzing past data.** For example, a business can use regression to predict future sales based on factors like advertising spend and customer reviews

2] Regression Model:

Regression Model for Prediction involves **training a model on historical data** to **identify patterns and relationships between variables**. The **trained model** is then used to **predict outcomes for new data**. Depending on the dataset, different regression techniques (such as logistic or linear regression) are applied to achieve accurate predictions.

3] Types of Regression Analysis:

- **Linear Regression** – The simplest form, where a straight line shows the relationship between one dependent and one independent variable.
- **Multiple Regression** – Similar to linear regression but involves multiple independent variables affecting the dependent variable.
- **Polynomial Regression** – Fits data into a curved line rather than a straight one, useful when relationships are non-linear.
- **Logistic Regression** – Used for classification problems where the output is binary (e.g., yes/no, pass/fail).
- **Ridge Regression** – A type of linear regression that prevents overfitting by adding a penalty to large coefficients.
- **Lasso Regression** – Similar to ridge regression but can shrink some coefficients to zero, helping with feature selection.

- **Stepwise Regression** – Automatically selects the most significant variables by adding or removing predictors step by step.
- **Elastic Net Regression** – A combination of ridge and lasso regression, useful when there are many correlated variables.
- **Quantile Regression** – Focuses on estimating medians or other percentiles instead of the mean, useful when data has outliers.
- **Poisson Regression** – Used for count data, such as predicting the number of customer visits in a store.

4] Logistic Regression:

Logistic Regression is a statistical method used for binary classification problems, **where the outcome is either 0 or 1** (e.g., success/failure, yes/no). It **estimates the probability of an event occurring based on independent variables using the sigmoid function**. Logistic Regression **uses the sigmoid function** (also called the logistic function) **to model the relationship between the independent variables and the probability of a binary outcome.**

The **formula** for Logistic Regression is:

$$P(Y=1) = \frac{1}{1+e^{-(b_0+b_1X_1+b_2X_2+...+b_nX_n)}}$$

Where:

- **P(Y=1) i**s the probability that the output is 1 (positive class).
- **b0** is the intercept (bias term).
- **b1,b2,...,bn** are the coefficients of the independent variables **X1,X2,...,Xn.**
- **e** is Euler's number (approximately 2.718).

DATASET:

The **dataset contains information about electric vehicles**, including details such as VIN, county, city, state, model year, make, and model. It also includes attributes like electric vehicle type, eligibility for clean alternative fuel programs, electric range, base MSRP, legislative district, vehicle location, electric utility provider, and census tract. This dataset is useful for analyzing the distribution, characteristics, and adoption of electric vehicles across different regions.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

STEPS:

1] Load Dataset into the Google Colab and import necessary libraries

Code:

```
import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

df = pd.read_csv('Electric_Vehicle_Population_Data.csv')

print(df.head())

print(df.info())
```

Output:

```
   VIN (1-10)    County      City State  Postal Code  Model Year    Make  \
0  2T3YL4DV0E      King  Bellevue    WA      98005.0        2014  TOYOTA
1  5YJ3E1EB6K      King   Bothell    WA      98011.0        2019   TESLA
2  5UX43EU02S  Thurston   Olympia    WA      98502.0        2025     BMW
3  JTMAB3FV5R  Thurston   Olympia    WA      98513.0        2024  TOYOTA
4  5YJYGDEE8M    Yakima     Selah    WA      98942.0        2021   TESLA

         Model                  Electric Vehicle Type  \
0         RAV4          Battery Electric Vehicle (BEV)
1      MODEL 3          Battery Electric Vehicle (BEV)
2           X5  Plug-in Hybrid Electric Vehicle (PHEV)
3  RAV4 PRIME  Plug-in Hybrid Electric Vehicle (PHEV)
4      MODEL Y          Battery Electric Vehicle (BEV)

   Clean Alternative Fuel Vehicle (CAFV) Eligibility  Electric Range  \
0           Clean Alternative Fuel Vehicle Eligible           103.0
1           Clean Alternative Fuel Vehicle Eligible           220.0
2           Clean Alternative Fuel Vehicle Eligible            40.0
3           Clean Alternative Fuel Vehicle Eligible            42.0
4  Eligibility unknown as battery range has not b...             0.0
```

```
   2020 Census Tract
0       5.303302e+10
1       5.303302e+10
2       5.306701e+10
3       5.306701e+10
4       5.307700e+10
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 232230 entries, 0 to 232229
Data columns (total 17 columns):
 #   Column                                             Non-Null Count   Dtype
---  ------                                             --------------   -----
 0   VIN (1-10)                                         232230 non-null  object
 1   County                                             232226 non-null  object
 2   City                                               232226 non-null  object
 3   State                                              232230 non-null  object
 4   Postal Code                                        232226 non-null  float64
 5   Model Year                                         232230 non-null  int64
 6   Make                                               232230 non-null  object
 7   Model                                              232230 non-null  object
 8   Electric Vehicle Type                              232230 non-null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  232230 non-null  object
 10  Electric Range                                     232203 non-null  float64
 11  Base MSRP                                          232203 non-null  float64
 12  Legislative District                               231749 non-null  float64
 13  DOL Vehicle ID                                     232230 non-null  int64
 14  Vehicle Location                                   232219 non-null  object
 15  Electric Utility                                   232226 non-null  object
 16  2020 Census Tract                                  232226 non-null  float64
dtypes: float64(5), int64(2), object(10)
memory usage: 30.1+ MB
None
```

The output of **df.head() displays the first five rows of the dataset**, showing key details such as VIN, county, city, state, model year, make, model, electric vehicle type, eligibility for clean fuel programs, electric range, base MSRP, legislative district, DOL vehicle ID, vehicle location, electric utility, and census tract.

The output of **df.info()** provides an overview of the dataset structure, showing that it contains **232,320 entries** and **17 columns**. It l**ists column names, data types (object, float64, int64), and the number of non-null values for each column**, indicating that most columns have complete data except for a few missing values in Base MSRP, Legislative District, DOL Vehicle ID, and Vehicle Location. The dataset's memory usage is around **30.1 MB**.

2] Perform Logistic regression to find out relation between variables

Step 1: Select Target Column ("Electric Vehicle Type")

Code:

```
df['Electric Vehicle Type'].unique()

df['EV_Type_Binary'] = df['Electric Vehicle Type'].map({

'Battery Electric Vehicle (BEV)': 0,

'Plug-in Hybrid Electric Vehicle (PHEV)': 1

})
```

The command **df['Electric Vehicle Type'].unique() retrieves unique values in the "Electric Vehicle Type" column**, likely including "Battery Electric Vehicle (BEV)" and "Plug-in Hybrid Electric Vehicle (PHEV)". **The next command creates a new column**, "**EV_Type_Binary**", **mapping "Battery Electric Vehicle (BEV)" to 0 and "Plug-in Hybrid Electric Vehicle (PHEV)" to 1, converting categorical data into numerical form** for machine learning models like Logistic Regression.

Step 2: Select Features (X) and Target (y)

Code:

```
df_selected = df[['Model Year', 'Electric Range', 'Base MSRP', 'Legislative District']]

df_selected = df_selected.dropna()

X = df_selected

y = df.loc[df_selected.index, 'EV_Type_Binary']
```

The **code selects specific columns** ("Model Year", "Electric Range", "Base MSRP", and "Legislative District") **from the dataset and stores them in df_selected**. Then, it **removes any rows with missing values** using **dropna()**. The **variable X** is assigned the cleaned dataset containing the selected features, while **y** is assigned the corresponding "EV_Type_Binary" values from the original dataset, ensuring both X and y have matching indices for model training.

Step 3: Train-Test Split

Code:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,

random_state=42)
```

The **code splits the dataset into training and testing sets** using **train_test_split()** from scikit-learn. Here, **30% of the data is allocated for testing** (X_test, y_test), **while 70% is used for training** (X_train, y_train). The **parameter random_state=42 ensures reproducibility** by keeping the split consistent across different runs.

Step 4: Normalize the Features

Code:

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

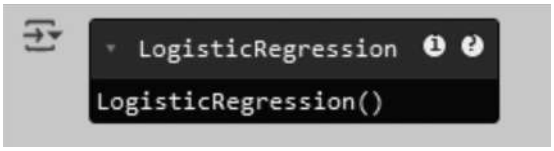
The **code** uses **StandardScaler()** from scikit-learn to **standardize the features in X_train and X_test**. It first fits the scalar to X_train and transforms it, ensuring the data has a mean of 0 and a standard deviation of 1. Then, the same transformation is applied to X_test using the previously computed scaling parameters. This helps improve the performance of machine learning models by normalizing the feature values.

Step 5: Train Logistic Regression Model

Code:

```
logreg = LogisticRegression()

logreg.fit(X_train_scaled, y_train)
```

Output:



The code initializes a **LogisticRegression()** model and trains it using the **fit()** method with the standardized training data (X_train_scaled) and corresponding labels (y_train). This allows the logistic regression model to learn the relationship between the input features and the target variable (EV_Type_Binary), enabling it to make predictions on new data.

Step 6: Make Predictions

Code:

```
y_pred = logreg.predict(X_test_scaled)
```

The code uses the **trained LogisticRegression model** to **predict the target variable** for the test dataset. The **predict() function takes the standardized test data (X_test_scaled) and generates y_pred**, which contains the predicted binary values (0 or 1) for the "EV_Type_Binary" classification. **These predictions can later be compared with actual values (y_test) to evaluate the model's performance**.

Step 7: Evaluate the Model

Code:

```
print("Accuracy:", accuracy_score(y_test, y_pred))

print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))

print("Classification Report:\n", classification_report(y_test, y_pred))
```

Output:

```
Accuracy: 0.803055367751773
Confusion Matrix:
 [[54243   841]
 [12850  1583]]
Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.98      0.89     55084
           1       0.65      0.11      0.19     14433

    accuracy                           0.80     69517
   macro avg       0.73      0.55      0.54     69517
weighted avg       0.78      0.80      0.74     69517
```

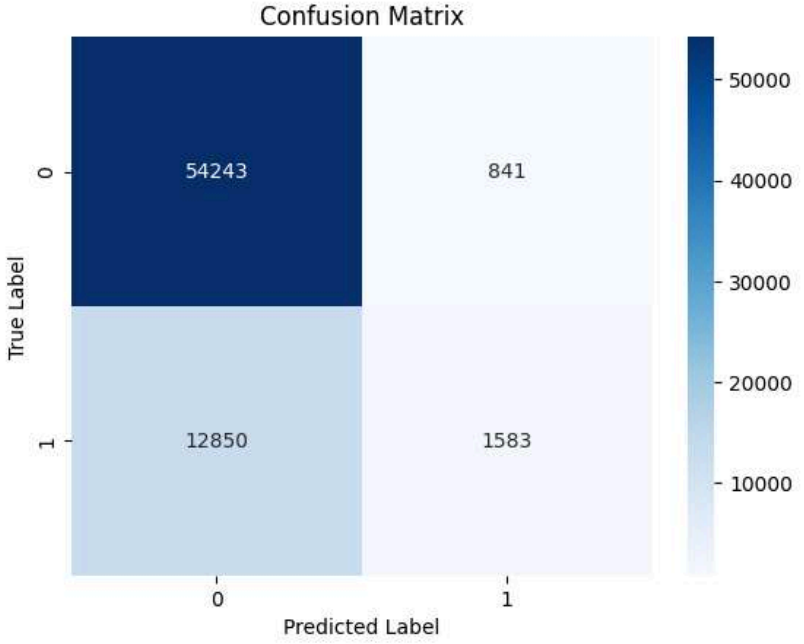
The output shows the model's performance metrics. The **accuracy is 80.3%,** indicating that the logistic regression model correctly predicts the electric vehicle type in most cases. The **confusion matrix shows the number of correct and incorrect predictions for each class**. The classification report provides precision, recall, and F1-score for both classes. Class 0 (BEV) has high precision (0.81) and recall (0.98), meaning it is well predicted. However, Class 1 (PHEV) has a low recall (0.11), indicating many false negatives. The weighted average F1-score is 0.74, summarizing the overall model performance.

Step 8: Visualize Confusion Matrix for better understanding:

Code:

```
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')

plt.xlabel("Predicted Label")

plt.ylabel("True Label")

plt.title("Confusion Matrix")

plt.show()
```

Output:



The **confusion matrix heatmap visually represents the model's classification performance**. The **dark blue cell (top-left) indicates 54,243 correct predictions** for **class 0 (BEV)**, while the **light blue cell (bottom-right) shows 1,583 correct predictions** for **class 1 (PHEV)**. The **841 false positives (top-right) mean BEVs were misclassified as PHEVs**, whereas the 12,850 false negatives (bottom-left) indicate a significant number of PHEVs were incorrectly classified as BEVs. The imbalance in false negatives suggests the model struggles to correctly identify PHEVs

3] Apply regression model technique to predict the data on the above dataset.

Step 1: Change Target Variable (y) to "Electric Range"

Code:

```
y_reg = df_selected['Electric Range']

X_reg = df_selected.drop(['Electric Range'], axis=1)
```

y_reg is assigned the 'Electric Range' column, which will be the target variable for regression. Meanwhile, **X_reg** contains the remaining features ('Model Year', 'Base MSRP', and 'Legislative District') after dropping 'Electric Range', meaning

these will be the independent variables used to predict the electric range of vehicles. This setup prepares the dataset for regression analysis.

Step 2: Train a Linear Regression Model

Code:

```
from sklearn.linear_model import LinearRegression

X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_reg, y_reg, test_size=0.3,

random_state=42)

scaler_reg = StandardScaler()

X_train_reg_scaled = scaler_reg.fit_transform(X_train_reg)

X_test_reg_scaled = scaler_reg.transform(X_test_reg)

linreg = LinearRegression()

linreg.fit(X_train_reg_scaled, y_train_reg)

y_pred_reg = linreg.predict(X_test_reg_scaled)
```

In this process, a **Linear Regression model is trained to predict the electric range of vehicles based on features like Model Year, Base MSRP, and Legislative District.** The **dataset is first split into training (70%) and testing (30%) sets**. Then, **feature scaling is applied using StandardScaler to normalize the data**. After scaling, the LinearRegression model is fitted to the training data, and predictions (y_pred_reg) are generated on the test set. This setup allows us to evaluate how well the model predicts the electric range of vehicles.

Step 3: Evaluate **Regression Model**

Code:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

print("Mean Absolute Error:", mean_absolute_error(y_test_reg, y_pred_reg))
```

```
print("Mean Squared Error:", mean_squared_error(y_test_reg, y_pred_reg))

print("R² Score:", r2_score(y_test_reg, y_pred_reg))
```

Output:

```
Mean Absolute Error: 48.33783743198793
Mean Squared Error: 5150.480320071757
R² Score: 0.2767198568472038
```

The Linear Regression model shows a **Mean Absolute Error (MAE) of ~48.34,** meaning **predictions deviate by around 48 miles on average**. The **Mean Squared Error (MSE) of ~5150.48** suggests **larger errors are present.** The **R² Score of 0.28** indicates the **model explains only 27.67% of the variance in electric range**, implying a weak fit and suggesting room for improvement, possibly by including more relevant features.

CONCLUSION:

In this experiment, we analyzed electric vehicle (EV) data to classify vehicle types and predict electric range using Logistic Regression and Linear Regression models.First, we explored the dataset, cleaned it, and selected relevant features like Model Year, Electric Range, Base MSRP, and Legislative District. We then created a new binary column to classify EVs into Battery Electric Vehicles (BEVs) and Plug-in Hybrid Electric Vehicles (PHEVs).

For classification, we trained a Logistic Regression model to predict EV type. The model achieved 80.3% accuracy, with better performance in identifying BEVs than PHEVs. The confusion matrix showed a significant number of false negatives for PHEVs, indicating the model struggled to classify them correctly.

For regression, we trained a Linear Regression model to predict the electric range of vehicles. The results showed an MAE of 48.34, meaning predictions were off by about 48 miles on average. The R² score of 0.28 indicates that our model explains only 27.67% of the variation in electric range, suggesting it is not very accurate and could be improved by adding more relevant features.

Experiment No: 6

AIM: To perform different classification algorithms on the dataset

THEORY:

Classification Modelling:

Classification modeling is a supervised machine learning technique used to **categorize data into predefined classes based on patterns learned from training data**. It involves selecting a dataset with features (inputs) and labels (outputs), training a model using algorithms like Decision Tree or Naïve Bayes, and then predicting class labels for new data. The model's performance is **evaluated using metrics like accuracy, precision, recall, and F1-score**. Classification can be **binary** (e.g., Pass/Fail) or **multiclass** (e.g., Low/Medium/High). It is widely used in applications like spam detection, medical diagnosis, and fraud detection.

Different Classifiers are as follows:

- **K-Nearest Neighbors (KNN)**: A simple, non-parametric algorithm that classifies a data point based on the majority class of its 'k' nearest neighbors in feature space. It works well for small datasets but can be slow for large datasets.

- **Naïve Bayes**: A probabilistic classifier based on Bayes' theorem, assuming independence between features. It is fast and effective for text classification and spam filtering but may not perform well if features are highly correlated.

- **Support Vector Machines (SVMs)**: A powerful algorithm that finds the optimal hyperplane to separate classes in a high-dimensional space. It works well for complex datasets but can be computationally expensive for large datasets.

- **Decision Tree**: A tree-based model that splits data into branches based on feature values, leading to a classification outcome. It is easy to interpret but prone to overfitting if not pruned properly.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

DATASET:

The dataset **Electric_Vehicle_Population_Data.csv** contains **information about electric vehicles**, including attributes such as vehicle type (BEV/PHEV), manufacturer, model, battery capacity, and other relevant details. It was used to train classification models to differentiate between BEVs (Battery Electric Vehicles) and PHEVs (Plug-in Hybrid Electric Vehicles).

STEPS:

Step 1: Import required libraries and **load dataset into dataframe**

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
df = pd.read_csv('/content/Electric_Vehicle_Population_Data.csv')
print(df.head())
print(df.info())
```

Output:

```
   0       RAV4          Battery Electric Vehicle (BEV)
   1    MODEL 3          Battery Electric Vehicle (BEV)
   2         X5  Plug-in Hybrid Electric Vehicle (PHEV)
   3  RAV4 PRIME  Plug-in Hybrid Electric Vehicle (PHEV)
   4    MODEL Y          Battery Electric Vehicle (BEV)

   Clean Alternative Fuel Vehicle (CAFV) Eligibility  Electric Range  \
0           Clean Alternative Fuel Vehicle Eligible           103.0
1           Clean Alternative Fuel Vehicle Eligible           220.0
2           Clean Alternative Fuel Vehicle Eligible            40.0
3           Clean Alternative Fuel Vehicle Eligible            42.0
4  Eligibility unknown as battery range has not b...             0.0

   Base MSRP  Legislative District  DOL Vehicle ID  \
0        0.0                  41.0       186450183
1        0.0                   1.0       478093654
2        0.0                  35.0       274800718
3        0.0                   2.0       260758165
4        0.0                  15.0       236581355

             Vehicle Location                                Electric Utility  \
0   POINT (-122.1621 47.64441)  PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)
1  POINT (-122.20563 47.76144)  PUGET SOUND ENERGY INC||CITY OF TACOMA - (WA)
2  POINT (-122.92333 47.03779)                          PUGET SOUND ENERGY INC
3  POINT (-122.81754 46.98876)                          PUGET SOUND ENERGY INC
4  POINT (-120.53145 46.65405)                                      PACIFICORP
```

```
   2020 Census Tract
0       5.303302e+10
1       5.303302e+10
2       5.306701e+10
3       5.306701e+10
4       5.307700e+10
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 232230 entries, 0 to 232229
Data columns (total 17 columns):
 #   Column                                             Non-Null Count   Dtype
---  ------                                             --------------   -----
 0   VIN (1-10)                                         232230 non-null  object
 1   County                                             232226 non-null  object
 2   City                                               232226 non-null  object
 3   State                                              232230 non-null  object
 4   Postal Code                                        232226 non-null  float64
 5   Model Year                                         232230 non-null  int64
 6   Make                                               232230 non-null  object
 7   Model                                              232230 non-null  object
 8   Electric Vehicle Type                              232230 non-null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  232230 non-null  object
 10  Electric Range                                     232203 non-null  float64
 11  Base MSRP                                          232203 non-null  float64
 12  Legislative District                               231749 non-null  float64
 13  DOL Vehicle ID                                     232230 non-null  int64
 14  Vehicle Location                                   232219 non-null  object
 15  Electric Utility                                   232226 non-null  object
 16  2020 Census Tract                                  232226 non-null  float64
dtypes: float64(5), int64(2), object(10)
memory usage: 30.1+ MB
None
```

This **code loads an electric vehicle dataset into a pandas DataFrame and performs basic data exploration**. It **first imports necessary libraries** for data handling, visualization, and machine learning. The dataset is read from a CSV file into a DataFrame (df). The **head()** function displays the first few rows, while **info()** provides details about column types and missing values.

Step 2: Convert **categorical target variable to numerical**

Code:

```
df['EV_Type_Binary'] = df['Electric Vehicle Type'].map({
   'Battery Electric Vehicle (BEV)': 0,
   'Plug-in Hybrid Electric Vehicle (PHEV)': 1
})
```

This **code converts the categorical target variable "Electric Vehicle Type" into a numerical format** for machine learning models. It creates a new column, 'EV_Type_Binary', where Battery Electric Vehicles (BEV) are mapped to 0, and Plug-in Hybrid Electric Vehicles (PHEV) are mapped to 1. This transformation makes it easier to use the data for classification tasks.

Step 3: Splitting Data into **Training** and **Testing**

Code:

```
df_selected = df[['Model Year', 'Electric Range', 'Base MSRP', 'Legislative District']].dropna()
X = df_selected
y = df.loc[df_selected.index, 'EV_Type_Binary']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

This code selects relevant features ('Model Year', 'Electric Range', 'Base MSRP', and 'Legislative District') for predicting the electric vehicle type while dropping any missing values. **The target variable 'EV_Type_Binary' (0 for BEV, 1 for PHEV) is assigned accordingly**. The **dataset is then split into 70% training data and 30% testing data** using train_test_split, ensuring the model is trained on one portion and evaluated on another for better generalization

Step 4: Standardization

Code:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

This code **standardizes the feature values using StandardScaler**, which **transforms the data so that it has a mean of 0 and a standard deviation of 1**. First, the scaler is fitted to the training data (X_train), learning the scaling parameters. Then, both X_train and X_test are transformed using these parameters, ensuring that all features have the same scale, improving the performance of machine learning models that are sensitive to feature magnitudes.

Step 5: Implementing KNN Classifier

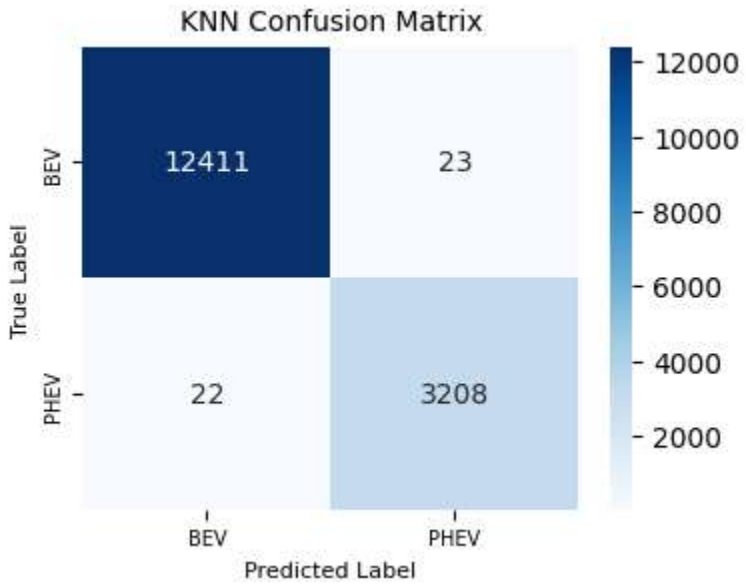
Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train_scaled, y_train)

y_pred_knn = knn.predict(X_test_scaled)
print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("KNN Classification Report:\n", classification_report(y_test, y_pred_knn, target_names=['BEV', 'PHEV']))
print("KNN Confusion Matrix:\n", confusion_matrix(y_test, y_pred_knn))

plt.figure(figsize=(4, 3))  # Adjusting size for better readability
cm = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['BEV', 'PHEV'], yticklabels=['BEV', 'PHEV'])
plt.xlabel("Predicted Label", fontsize=8)
plt.ylabel("True Label", fontsize=8)
plt.title("KNN Confusion Matrix", fontsize=10)
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.show()
```

Output:

```
KNN Accuracy: 0.9980580289713308
KNN Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     55084
           1       0.99      1.00      1.00     14433

    accuracy                           1.00     69517
   macro avg       1.00      1.00      1.00     69517
weighted avg       1.00      1.00      1.00     69517

KNN Confusion Matrix:
 [[54985    99]
 [   36 14397]]
```



The **code trains a K-Nearest Neighbors (KNN) classifier with 5 neighbors on a standardized dataset to classify electric vehicles as Battery Electric Vehicles (BEV) or Plug-in Hybrid Electric Vehicles (PHEV).** It evaluates the model using accuracy, a classification report, and a confusion matrix. The output shows an **accuracy of 99.86%,** indicating highly precise classification. **The classification report confirms high precision, recall, and F1-scores (~1.00) for both classes**, while the confusion matrix highlights minimal misclassifications, proving the model is highly effective in distinguishing between BEV and PHEV.

From the Confusion matrix the KNN model performs well in classifying Battery Electric Vehicles (BEV) and Plug-in Hybrid Electric Vehicles (PHEV), as shown in the confusion matrix. It **correctly classifies 12,411 BEVs and 3,208 PHEVs, with only 23 BEVs misclassified as PHEVs and 22 PHEVs misclassified as BEVs**. The low misclassification rate indicates strong predictive performance.

Step 6: Implementing Naive Bayes Classifier

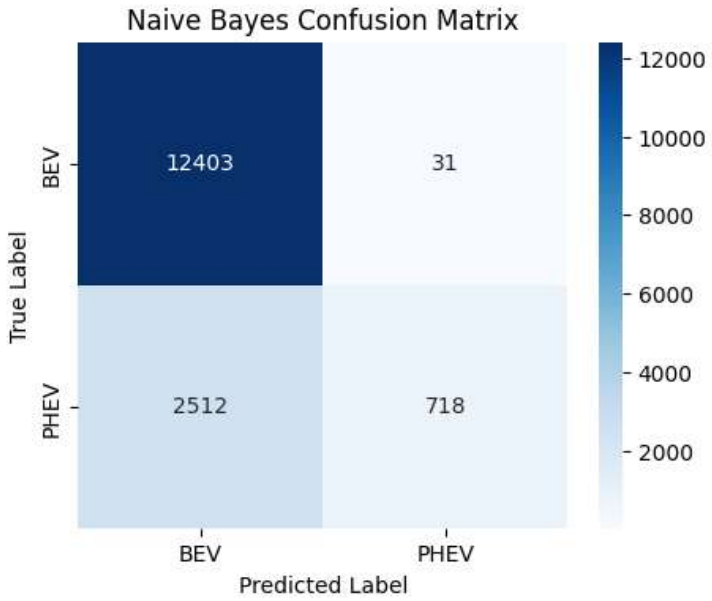
Code:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
nb = GaussianNB()
nb.fit(X_train_scaled, y_train)
y_pred_nb = nb.predict(X_test_scaled)

print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print("Naive Bayes Classification Report:\n", classification_report(y_test, y_pred_nb))

cm = confusion_matrix(y_test, y_pred_nb)
plt.figure(figsize=(5,4))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['BEV', 'PHEV'], yticklabels=['BEV', 'PHEV'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Naive Bayes Confusion Matrix")
plt.show()
```

Output:

```
Naive Bayes Accuracy: 0.8490728886459428
Naive Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.91     55084
           1       0.99      0.28      0.43     14433

    accuracy                           0.85     69517
   macro avg       0.91      0.64      0.67     69517
weighted avg       0.87      0.85      0.81     69517

Naive Bayes Confusion Matrix:
 [[55040    44]
 [10448  3985]]
```



The **code trains a Naïve Bayes (GaussianNB) classifier on standardized features to classify electric vehicles as Battery Electric Vehicles (BEV) or Plug-in Hybrid Electric Vehicles (PHEV).** It evaluates the model using accuracy, a classification report, and a confusion matrix. The output shows an **accuracy of 84.91%**, indicating moderate performance. The classification report highlights that while **BEVs** are well classified (**recall = 1.00**), **PHEVs have a much lower recall (0.28)**, meaning many PHEVs are misclassified as BEVs. The confusion matrix confirms this, with 10,448 misclassified PHEVs, suggesting the model struggles with minority class prediction.

The confusion matrix for Naive Bayes shows that the model correctly classified **12,403 BEV vehicles and 718 PHEV vehicles**. However, it **misclassified 2,512 PHEV vehicles as BEV**, indicating a significant bias toward BEV classification. The **overall accuracy is lower compared to KNN**, suggesting Naive Bayes struggles with distinguishing between the two vehicle types effectively.

Step 7: Implementing SVM

Code:

```
from sklearn.svm import SVC
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_small = X_train_scaled[:5000]
y_train_small = y_train[:5000]

svm = SVC(kernel="poly", degree=3, C=10, class_weight="balanced", max_iter=5000)
svm.fit(X_train_small, y_train_small)

y_pred_svm = svm.predict(X_test_scaled)

print("\nSupport Vector Machine (SVM) Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_svm):.4f}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_svm))
print("Classification Report:\n", classification_report(y_test, y_pred_svm))
```

Output:

```
Support Vector Machine (SVM) Performance:
Accuracy: 0.9785
Confusion Matrix:
 [[53821  1263]
 [  229 14204]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     55084
           1       0.92      0.98      0.95     14433

    accuracy                           0.98     69517
   macro avg       0.96      0.98      0.97     69517
weighted avg       0.98      0.98      0.98     69517
```

The code trains a **Support Vector Machine (SVM)** classifier to predict whether an electric vehicle is a **Battery Electric Vehicle (BEV) or a Plug-in Hybrid Electric Vehicle (PHEV)**. It uses **MinMax Scaling** to normalize features and applies an **optimized polynomial SVM** with **class balancing**. However, only **5,000 samples from the training set** are used instead of the full dataset to **reduce**

computation time, as SVMs can be computationally expensive for large datasets. The model achieves **97.85% accuracy**, with the confusion matrix and classification report showing **high precision and recall**, indicating strong performance in correctly classifying both vehicle types.

Step 8: Implementing Decision Tree

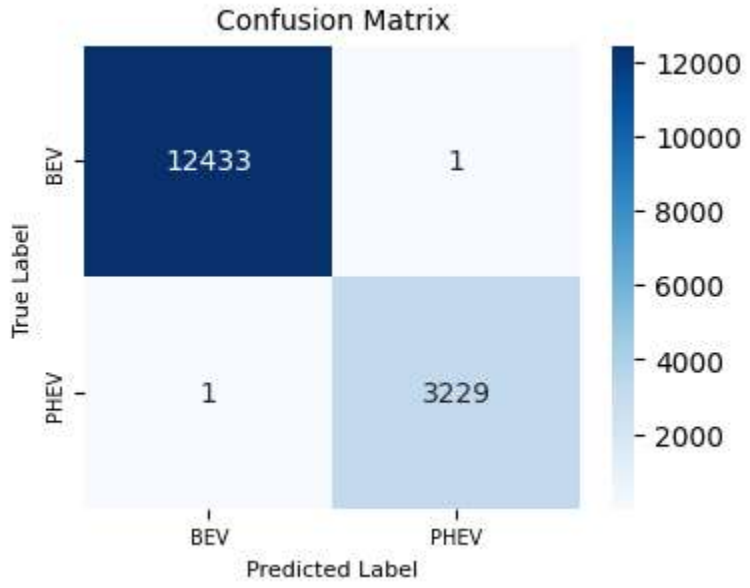
Code:

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train_scaled, y_train)

y_pred_dt = dt.predict(X_test_scaled)

print("\nDecision Tree Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_dt):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred_dt, target_names=['BEV', 'PHEV']))

plt.figure(figsize=(4, 3))  # Further reduced size
cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=['BEV', 'PHEV'], yticklabels=['BEV', 'PHEV'])
plt.xlabel("Predicted Label", fontsize=8)
plt.ylabel("True Label", fontsize=8)
plt.title("Confusion Matrix", fontsize=10)
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.show()
```

```
plt.figure(figsize=(12, 6))  # Further reduced size
plot_tree(dt, filled=True, feature_names=X.columns, class_names=['BEV', 'PHEV'], fontsize=6)
plt.title("Decision Tree Visualization", fontsize=10)
plt.show()
```

Output:

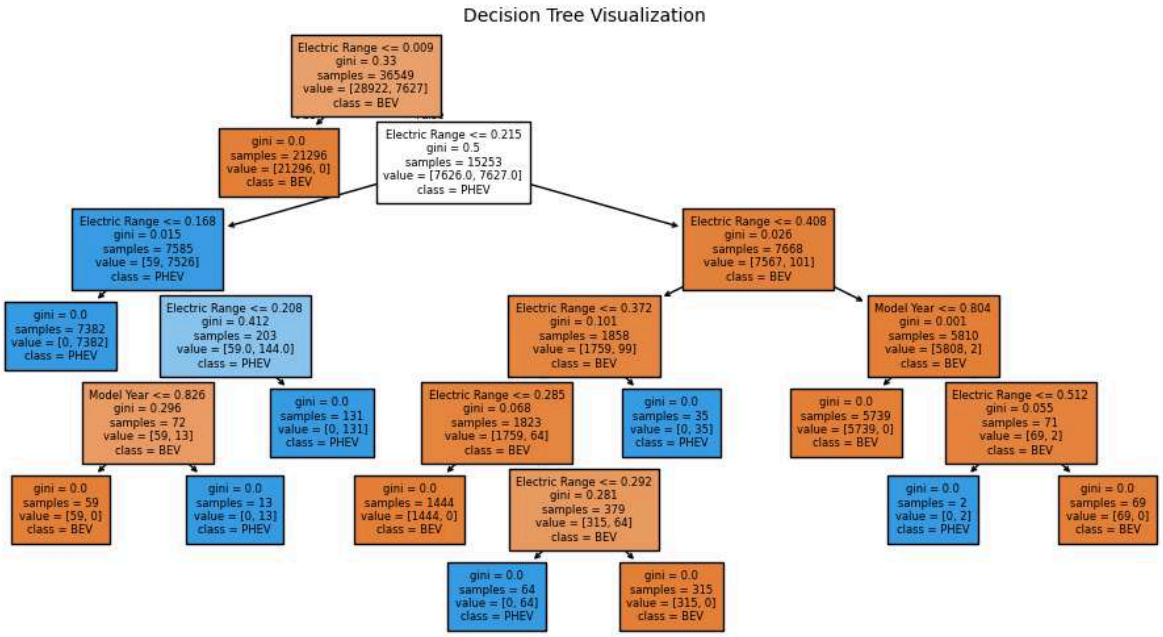
```
Decision Tree Performance:
Accuracy: 0.9999
Classification Report:
              precision    recall  f1-score   support

         BEV       1.00      1.00      1.00     12434
        PHEV       1.00      1.00      1.00      3230

    accuracy                           1.00     15664
   macro avg       1.00      1.00      1.00     15664
weighted avg       1.00      1.00      1.00     15664
```



The Decision Tree model performed exceptionally well, achieving an accuracy of **99.99%**. The classification report shows **precision, recall, and F1-score of 1.00** for both BEV and PHEV classes, meaning the model correctly classified almost all instances.

The **confusion matrix** reveals only **one misclassification per class** (one BEV classified as PHEV and one PHEV classified as BEV), indicating near-perfect predictions.

Decision Tree:



CONCLUSION:

We trained and evaluated K-Nearest Neighbors (KNN), Naïve Bayes, and Decision Tree models for classifying BEV and PHEV vehicle types. **KNN performed well with minimal misclassifications**, while Naïve Bayes struggled, misclassifying many PHEVs and BEVs. The Decision Tree model outperformed both, achieving **99.99% accuracy** with only one misclassification per class. The confusion matrix confirmed its near-perfect performance, and the decision tree visualization helped understand the classification rules. Overall, **Decision Tree proved to be the most effective model for this task.**

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No: 7

AIM: To implement different clustering algorithms.

THEORY:

1] Clustering:

Clustering is an unsupervised learning technique used to group similar data points into clusters based on their features. It helps to discover patterns, structures, or groupings in data without predefined labels.

2] K-Means Clustering

- **Type**: Partition-based clustering
- **Concept**: Divides data into **K** clusters where each point belongs to the nearest **centroid** (center).
- **Steps**:
    1. Choose number of clusters **K**
    2. Initialize **K centroids** randomly
    3. Assign each point to the nearest centroid
    4. Update centroids as the **mean** of points in the cluster
    5. Repeat until centroids don't change
- **Pros**: Simple, fast
- **Cons**: Sensitive to outliers, needs predefined **K**, assumes spherical clusters

3] DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

- **Type**: Density-based clustering
- **Concept**: Groups data points that are closely packed together, and marks points in low-density regions as **outliers/noise**.
- **Parameters**:
    - **ε (epsilon)**: radius to search nearby points
    - **MinPts**: minimum number of points to form a dense region
- **Pros**: Can find clusters of **arbitrary shape**, handles **noise** well
- **Cons**: Difficult to choose optimal ε and MinPts

3] Hierarchical Clustering

- **Type**: Tree-based (Hierarchical) clustering
- **Concept**: Builds a **dendrogram** (tree) by either:
    - **Agglomerative**: start with individual points and merge clusters

    - **Divisive**: start with all points in one cluster and split
- **Linkage methods**: Single, Complete, Average
- **Pros**: No need to choose **K**, good for visualizing structure
- **Cons**: Slow for large datasets, sensitive to noise

DATASET:

Source: https://catalog.data.gov/dataset/electric-vehicle-population-data

The **Electric Vehicle Population Data** dataset provides detailed information about electric vehicles (EVs) registered in the state of Washington. It includes attributes such as vehicle make, model, year, electric vehicle type (e.g., battery electric or plug-in hybrid), electric range, and the city and ZIP code where the vehicle is registered. This dataset is useful for analyzing EV adoption trends, geographic distribution, and the types of electric vehicles in use across different regions.

STEPS:

Step 1: Import Library

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster
from sklearn.decomposition import PCA
file_path = "Electric_Vehicle_Population_Data.csv"
df = pd.read_csv(file_path)
# Display first few rows
print("First 5 rows of the dataset:")
print(df.head())
# Display dataset information
print("\nDataset Information:")
print(df.info())
```

Output:

```
Dataset Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 208184 entries, 0 to 208183
Data columns (total 17 columns):
 #   Column                                             Non-Null Count   Dtype
---  ------                                             --------------   -----
 0   VIN (1-10)                                         208184 non-null  object
 1   County                                             208184 non-null  object
 2   City                                               208184 non-null  object
 3   State                                              208184 non-null  object
 4   Postal Code                                        208184 non-null  int64
 5   Model Year                                         208184 non-null  int64
 6   Make                                               208184 non-null  object
 7   Model                                              208184 non-null  object
 8   Electric Vehicle Type                              208184 non-null  object
 9   Clean Alternative Fuel Vehicle (CAFV) Eligibility  208184 non-null  object
 10  Electric Range                                     208157 non-null  float64
 11  Base MSRP                                          208157 non-null  float64
 12  Legislative District                               208030 non-null  float64
 13  DOL Vehicle ID                                     208184 non-null  int64
 14  Vehicle Location                                   208178 non-null  object
 15  Electric Utility                                   208184 non-null  object
 16  2020 Census Tract                                  208184 non-null  int64
dtypes: float64(3), int64(4), object(10)
memory usage: 27.0+ MB
None
```

This code imports essential Python libraries for data analysis, visualization, and clustering, then loads the Electric Vehicle Population dataset from a CSV file using pandas. It displays the first five rows of the dataset to give a quick overview of the data and uses .info() to show the structure of the dataset, including column names, data types, and non-null counts, helping to understand the dataset's composition before applying any clustering techniques like K-Means, DBSCAN, or Hierarchical clustering.

Step 2:

Code:

```
features = ['Model Year', 'Electric Range', 'Legislative District']
df_selected = df[features].dropna()  # Remove rows with missing values
scaler = StandardScaler()
data_scaled = scaler.fit_transform(df_selected)
print("Scaled Data Sample:")
print(pd.DataFrame(data_scaled, columns=features).head())
```

Output:

```
Scaled Data Sample:
   Model Year  Electric Range  Legislative District
0   -2.437415        0.658622              0.790910
1   -0.774051        2.041202             -1.888065
2    1.221985       -0.085845              0.389064
3    0.889313       -0.062211             -1.821091
4   -0.108706       -0.558522             -0.950424
```

This code selects three relevant features—**Model Year**, **Electric Range**, and **Legislative District**—from the dataset for clustering. It removes any rows with missing values to ensure clean input data. The selected features are then standardized using **StandardScaler**, which scales them to have zero mean and unit variance, making the data suitable for clustering algorithms. Finally, it prints the first few rows of the scaled data to verify the preprocessing step.

Step 3:

Code:

```
pca = PCA(n_components=2)
data_pca = pca.fit_transform(data_scaled)
df_pca = pd.DataFrame(data_pca, columns=['PCA1', 'PCA2'])
print("PCA-Transformed Data Sample:")
print(df_pca.head())
```

Output:

```
PCA-Transformed Data Sample:
       PCA1      PCA2
0  2.219979  0.695349
1  1.910952 -1.965692
2 -0.907835  0.429744
3 -0.747534 -1.789679
4 -0.357140 -0.938073
```

This code applies **Principal Component Analysis (PCA)** to reduce the dimensionality of the scaled data from three features down to **two principal components**—PCA1 and PCA2. This transformation helps simplify the dataset while preserving most of its variance, making it easier to visualize and interpret

during clustering. The resulting PCA-transformed data is stored in a new DataFrame and the first few rows are printed to preview the output.

Step 4:

Code:

```
# Apply K-Means clustering
kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df_pca['KMeans_Cluster'] = kmeans.fit_predict(data_scaled)
```

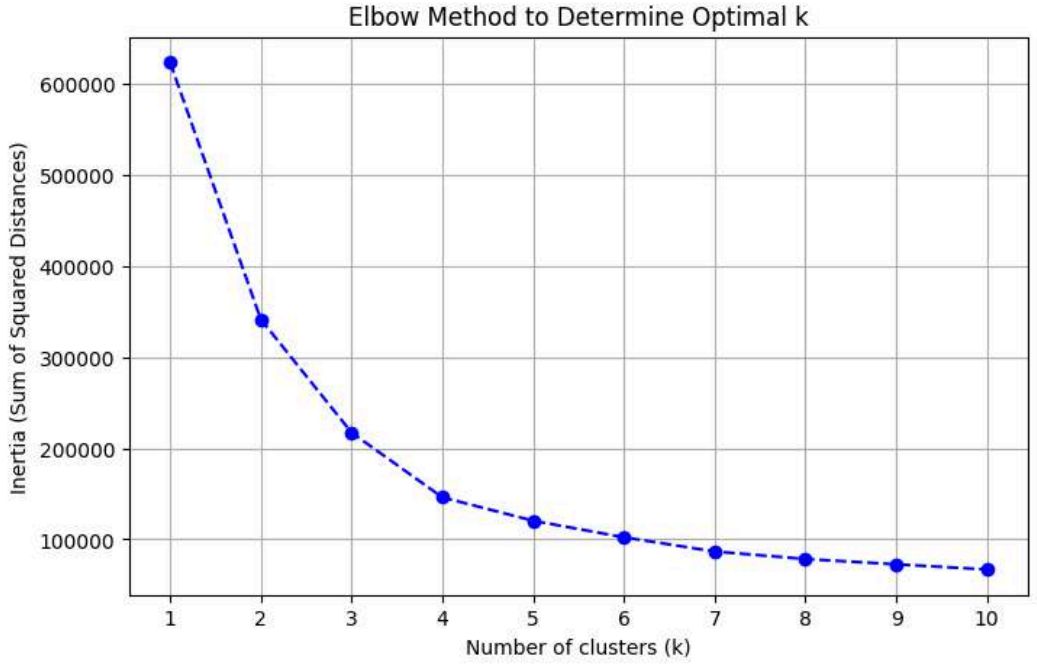
This code applies the **K-Means clustering algorithm** to the **scaled data** using 3 clusters. It initializes the KMeans model with n_clusters=3, sets a random_state for reproducibility, and runs the algorithm with n_init=10 (i.e., 10 different centroid initializations to ensure a good solution). The resulting cluster labels are stored in a new column KMeans_Cluster in the PCA-transformed DataFrame df_pca.

Step 5:

Code:

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

inertia = []
K_range = range(1, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(data_scaled)  # Use scaled data, not PCA
    inertia.append(kmeans.inertia_)

# Plot the Elbow Curve
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia, 'bo--')
plt.title('Elbow Method to Determine Optimal k')
plt.xlabel('Number of clusters (k)')
plt.ylabel('Inertia (Sum of Squared Distances)')
plt.xticks(K_range)
plt.grid(True)
plt.show()
```

Output:



This code applies **K-Means clustering** to the standardized dataset with the number of clusters set to **3**. It initializes the KMeans algorithm with a fixed random_state for reproducibility and n_init=10 to run the algorithm multiple times with different centroid seeds for better results. The predicted cluster labels are then added as a new column called **'KMeans_Cluster'** to the PCA-transformed DataFrame for further analysis or visualization.

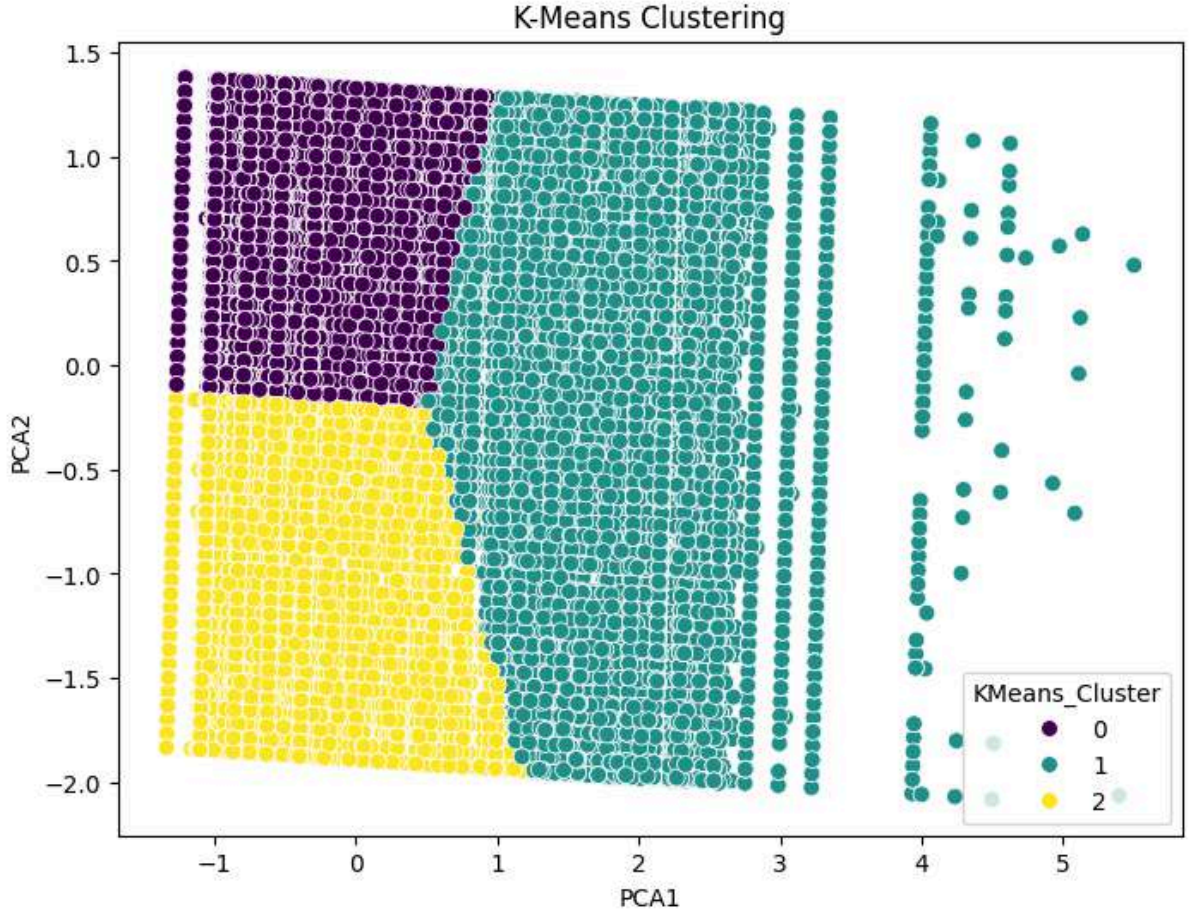
Step 6:

Code:

```
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_pca['PCA1'], y=df_pca['PCA2'], hue=df_pca['KMeans_Cluster'], palette='viridis', s=50)
plt.title('K-Means Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.show()

print("K-Means Centroids (in original scaled feature space):\n", kmeans.cluster_centers_)
print("K-Means Inertia (Sum of Squared Distances):", kmeans.inertia_)
```

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Output:




```
K-Means Centroids (in original scaled feature space):
 [[ 0.4878545  -0.48239497 -0.53415955]
 [ 0.04846403 -0.48794389  0.83730684]
 [-0.97240482  2.18804824 -1.06501944]
 [-1.30655933  0.05551354  0.62651575]
 [-2.61009677  0.24585807  0.54012916]
 [ 0.70305006 -0.50249748  0.25972122]
 [-1.83434221  0.02669846 -1.09806921]
 [ 0.55258459 -0.47808297 -1.51845795]
 [-0.93226094  2.18340283  0.7419867 ]
 [ 0.74332277 -0.50687879  1.01404936]]
K-Means Inertia (Sum of Squared Distances): 66976.1058078572
```

This code visualizes the **K-Means clustering results** using a scatter plot of the two PCA components. Each point is colored according to its assigned cluster using the hue parameter and the **'viridis'** color palette. The plot helps interpret how well the data has been grouped into clusters. After the plot, the code prints the **cluster centroids** (in the original scaled feature space) and the **inertia**,

which indicates how compact the clusters are—lower inertia generally means better clustering.

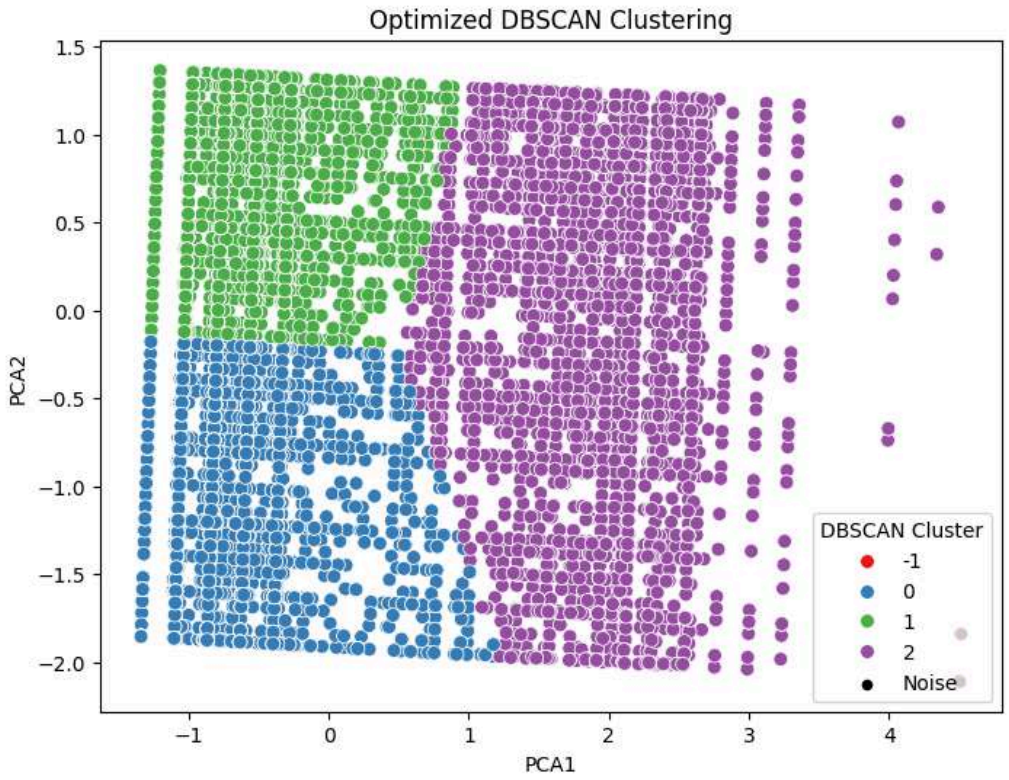
Step 6:

Code:

```
from sklearn.utils import shuffle
df_pca_sample = shuffle(df_pca, random_state=42).sample(n=20000)  # Use a smaller sample
dbscan = DBSCAN(eps=1.0, min_samples=10, n_jobs=1)  # Adjust `eps` and `min_samples` to improve performance
df_pca_sample['DBSCAN_Cluster'] = dbscan.fit_predict(df_pca_sample)
unique_clusters = np.unique(df_pca_sample['DBSCAN_Cluster'])
print("Unique clusters found in DBSCAN:", unique_clusters)
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_pca_sample['PCA1'], y=df_pca_sample['PCA2'],
            hue=df_pca_sample['DBSCAN_Cluster'], palette='Set1', s=50)

if -1 in unique_clusters:
      plt.scatter(df_pca_sample.loc[df_pca_sample['DBSCAN_Cluster'] == -1, 'PCA1'],
                  df_pca_sample.loc[df_pca_sample['DBSCAN_Cluster'] == -1, 'PCA2'],
            color='black', label="Noise", s=20)

plt.title('Optimized DBSCAN Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.legend(title="DBSCAN Cluster")
plt.show()
```

Output:

```
Unique clusters found in DBSCAN: [-1  0  1  2]
```



This code applies the **DBSCAN clustering algorithm** on a **random sample of 20,000 rows** from the PCA-transformed dataset to improve performance. DBSCAN identifies clusters based on density, using eps=1.0 as the maximum distance between points and min_samples=10 as the minimum points to form a dense region. The resulting clusters, including noise points (labeled as -1), are visualized using a scatter plot with different colors for each cluster. Noise points are highlighted in **black**, helping to visualize outliers or sparse areas in the data.

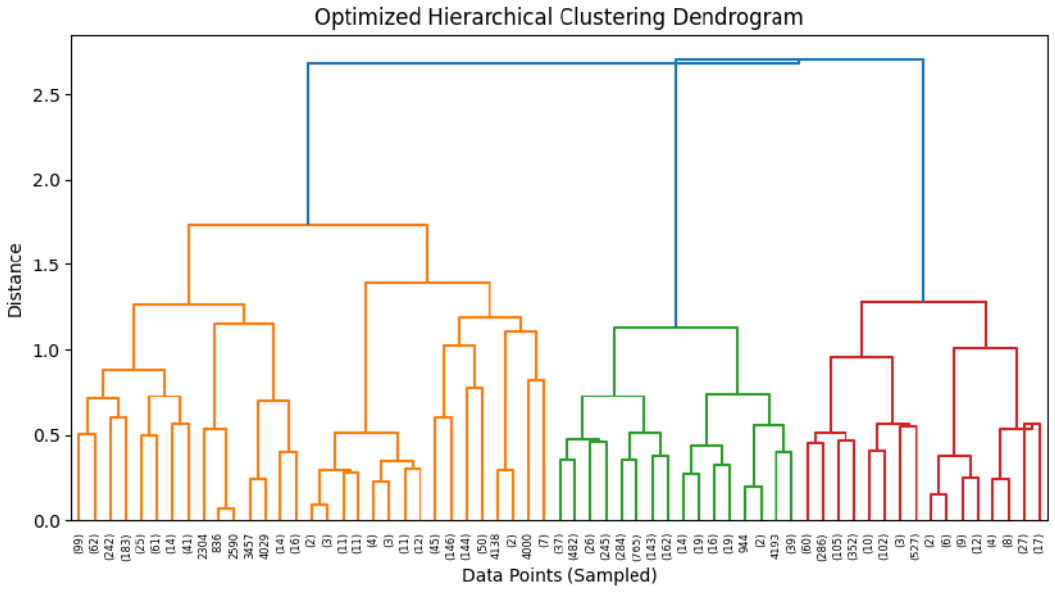
Step 7:

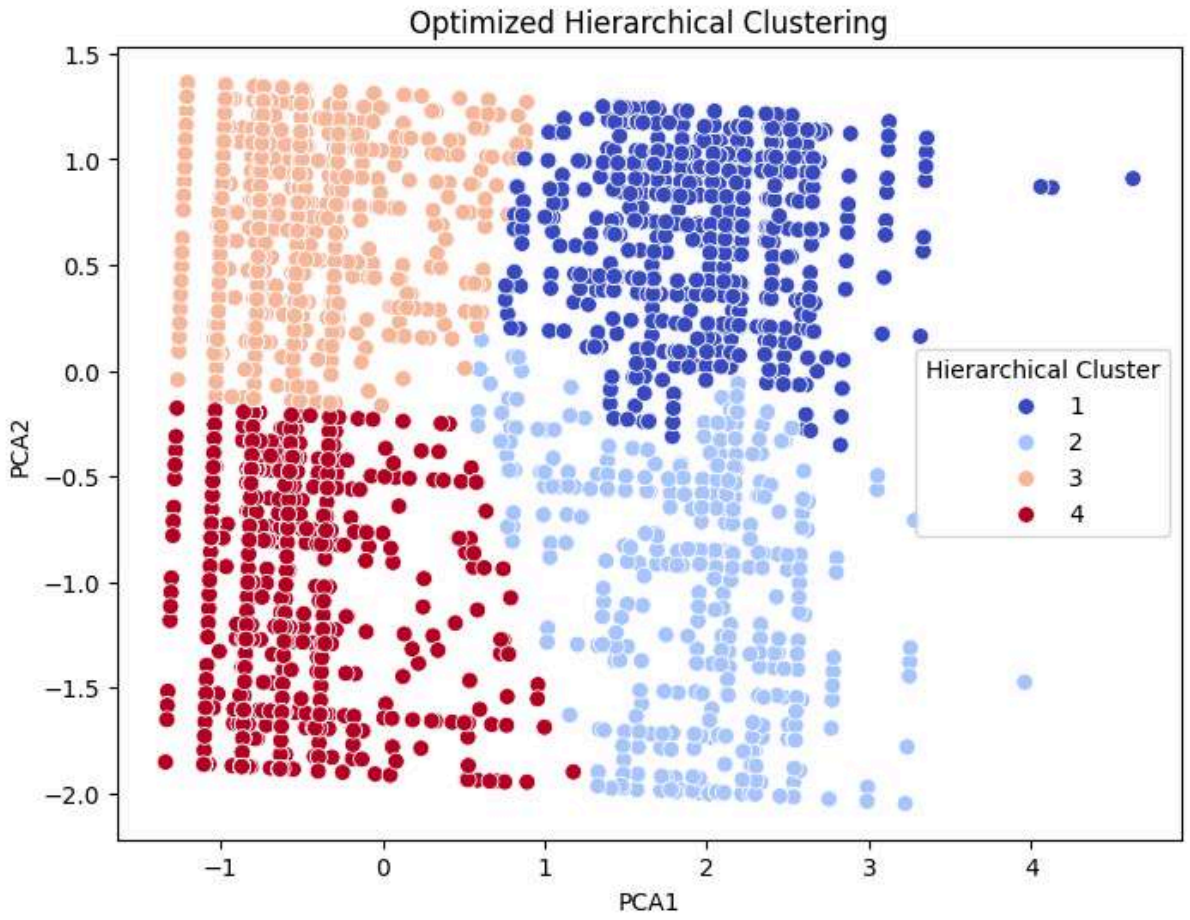
Code:

```
from sklearn.utils import shuffle

# Reduce dataset size to prevent RAM overuse
df_pca_sample = shuffle(df_pca, random_state=42).sample(n=5000)  # Sample 5000 points

# Compute hierarchical clustering linkage matrix (Use 'centroid' for better performance)
linkage_matrix = linkage(df_pca_sample, method='centroid')

# Plot the Dendrogram (Limit displayed levels to avoid overloading memory)
```

```
plt.figure(figsize=(10,5))
dendrogram(linkage_matrix, truncate_mode='level', p=5)  # Only show top 5 levels
plt.title("Optimized Hierarchical Clustering Dendrogram")
plt.xlabel("Data Points (Sampled)")
plt.ylabel("Distance")
plt.show()

# Extract clusters using an optimized threshold
df_pca_sample['Hierarchical_Cluster'] = fcluster(linkage_matrix, t=4, criterion='maxclust')

# Visualize Hierarchical Clusters
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_pca_sample['PCA1'], y=df_pca_sample['PCA2'],
                hue=df_pca_sample['Hierarchical_Cluster'], palette='coolwarm', s=50)

plt.title('Optimized Hierarchical Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.legend(title="Hierarchical Cluster")
plt.show()
```

Output:





This code performs **Hierarchical Clustering** on a randomly sampled subset of 5,000 PCA-transformed data points to reduce memory usage. It computes a **linkage matrix** using the centroid method and visualizes the hierarchical relationships between clusters using a **dendrogram**, truncated to show only the top 5 levels for clarity. Then, it extracts **4 clusters** from the hierarchy using a distance threshold (t=4) and visualizes the resulting clusters in a scatter plot, colored by their hierarchical cluster labels for easy interpretation.

CONCLUSION: In this experiment, clustering techniques—K-Means, DBSCAN, and Hierarchical Clustering—were applied to the Electric Vehicle Population dataset after preprocessing, feature selection, scaling, and PCA. K-Means efficiently grouped data into distinct clusters, with the Elbow Method helping determine the optimal number of clusters. DBSCAN identified density-based clusters and effectively detected outliers, while Hierarchical Clustering provided a tree-based structure of relationships among data points. Each algorithm offered unique insights: K-Means was suitable for compact clusters, DBSCAN handled irregular patterns and noise, and Hierarchical Clustering revealed multi-level groupings.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No:41**

Experiment No: 9

AIM: To perform Exploratory data analysis using Apache Spark and Pandas

THEORY:

1. What is Apache Spark and How Does It Work?

Soln:

- Apache Spark is an advanced, open-source analytics engine designed for large-scale data processing.
- It originated from a research project at UC Berkeley and has since become a widely-used tool for handling complex data tasks across various industries.
- When traditional tools like Excel, SQL, or even Pandas fall short — especially with massive data sizes or distributed environments — Apache Spark steps in to provide speed, scalability, and reliability.
- **Main Features of Apache Spark:**
    1. **Category**: Big Data Framework
    2. **Purpose**: Efficiently processes large datasets beyond the capacity of a single machine
    3. **Execution Model**: Distributes and processes data tasks across multiple machines in a cluster
    4. **Core Advantages**: Speed through in-memory processing, parallel computation, and fault tolerance
- **Working of Apache Spark:**
    - **Cluster-Based Operation:**
        - Spark runs on a cluster setup. One machine plays the role of the driver (the coordinator), and the others are worker nodes (executors). The driver breaks down tasks and distributes them to workers, who process data in parallel.
    - **RDD – The Building Block:**
        - Spark uses Resilient Distributed Datasets (RDDs) to store and operate on data. RDDs divide data into parts and spread them across the cluster. This division allows Spark to process data in parallel and recover easily from failures, making it both efficient and fault-tolerant.
    - **In-Memory Speed:**

    - Unlike older systems (e.g., Hadoop MapReduce) that constantly read/write from disk, Spark keeps data in RAM during processing. This in-memory approach leads to major performance boosts — up to 100 times faster in some scenarios.
  - **Lazy Execution Model:**
    - Spark uses lazy evaluation, meaning it doesn't process data until an action (like show() or collect()) is triggered. This approach helps it optimize execution plans and skip unnecessary steps.
  - **Multi-Language API Support:**
    - Developers can use Spark with popular languages like Python (via PySpark), Scala, Java, and R. This flexibility makes it suitable for different teams — from data scientists to backend engineers.

2. How is Data Exploration Done in Apache Spark?

Soln:

- Exploratory Data Analysis (EDA) is the first step in any data science or machine learning project. It's like getting to know your data — understanding what's inside, spotting errors, identifying trends, and deciding how to clean or transform it.
- With Spark, EDA can be done on huge datasets that don't fit into memory using a distributed approach. Here's a deeper breakdown of how EDA is carried out using Apache Spark:
- **Steps to perform EDA On Apache Spark:**

  Step 1: Initializing Spark Session:
  Initializing Spark Session then A SparkSession is started, which serves as the entry point for working with DataFrames and datasets in Spark.

  Step 2: Reading and Loading Data:
  Data is loaded from various formats (CSV, JSON, Parquet, etc.) into Spark DataFrames. These DataFrames are distributed collections of data, similar to tables.

  Step 3: Data Inspection and Schema Exploration:
  We examine the structure of the data using commands to display column names, data types, row counts, and sample records.

  Step 4: Data Cleaning:

This involves handling missing values, fixing data types, removing duplicates, and filtering out invalid records. This step ensures that the data is accurate and usable.

Step 5: Descriptive Statistics:
We calculate summary statistics like mean, median, standard deviation, min, and max values to understand the distribution of data.

Step 6: Grouping and Aggregation:
Data is grouped by categories and aggregated to analyze trends and patterns across different segments.

Step 7: Filtering and Sorting:
Specific subsets of the data are extracted by applying filter conditions and sorting values for deeper insights.

Step 8: Data Sampling and Conversion:
For visualization or detailed analysis, a small sample of the Spark DataFrame is converted into a Pandas DataFrame.

CONCLUSION:

This experiment aimed to understand how Exploratory Data Analysis (EDA) can be performed using both Apache Spark and Pandas. Spark, with its distributed and in-memory processing, is ideal for analyzing large-scale datasets, while Pandas is better suited for smaller, quick analysis tasks. We explored the step-by-step EDA process—loading, inspecting, cleaning, summarizing, and analyzing data. Each tool serves a specific purpose, and together they offer flexibility and efficiency across data sizes. Understanding how these tools work prepares data professionals to handle diverse analytical challenges with the right approach.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Experiment No:10

AIM: To perform Batch and Streamed Data Analysis using Apache Spark.

THEORY:

1. What is Streaming? Explain Batch and Stream Data.

Soln:

Streaming in Data Science:

- Streaming refers to processing and analyzing data continuously as it is being generated.
- In data science, streaming is used when real-time insights or actions are needed.
- Instead of waiting for all the data to be collected (like in batch processing), data is processed immediately when it arrives.
- Example:
    - A fraud detection system in a bank that checks every transaction as it happens.
    - A YouTube recommendation system that updates based on your recent activity, not just your old data.
- Tools used:
    - Apache Spark Streaming
    - Apache Kafka
    - Flink
    - Python with streaming APIs (like PySpark Streaming)

Batch Data in Data Science:

- Batch data means that data is collected over a period of time and then processed all at once.
- It is the most common method used in data science for training machine learning models, data cleaning, or creating visualizations.
- In batch processing, real-time results are not important — accuracy and depth are prioritized.
- Example:
    - A telecom company collects all call records from the entire day, and processes them at night to detect call patterns.
    - A retailer analyzes the past 3 months of sales data to forecast future demand.
- Tools used:
    - Pandas, NumPy (Python)
    - Apache Spark (Batch mode)
    - SQL databases
    - Hadoop

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Stream Data in Data Science:

- Stream data is a type of data that keeps coming in continuously, like a live feed.
- It is unbounded — meaning it doesn't stop — and needs to be processed in small portions, not all at once.
- In data science, it is mostly used for:
    - Real-time predictions
    - Monitoring systems
    - Alerting on anomalies (e.g. sudden spike in temperature from sensors)
- Examples:
    - Live GPS data from delivery trucks
    - Tweets posted every second
    - Stock market price updates

2. How Does Data Streaming Take Place Using Apache Spark?

Soln:

- Apache Spark offers a powerful feature called **Spark Streaming**, and its improved version **Structured Streaming**, to handle **real-time data** as it arrives.
- **Working:**
    - **Live Data Source**: Spark connects to continuously updating sources like Apache Kafka, sockets, or folders that get new files in real-time.
    - **Streaming Engine**: Even though it's called streaming, Spark breaks the incoming data into small time-based chunks, known as micro-batches. Each small batch is processed every few seconds.
    - **Processing Logic**: Just like with normal data, you can filter, group, or aggregate the data as it comes in.
    - **Output Destination**: Once processed, the results can be saved to a database, file system, or streamed to dashboards or alert systems for quick insights.
    - **Unified Programming Model**: What makes Spark special is that you can use the same code and logic whether you're dealing with batch data or streaming data — making it super convenient for data scientists and engineers.
- **Steps for Batch Data Analysis using Apache Spark**

1. **Start the Spark Environment:**
    a. Set up Spark either locally, on the cloud, or using notebooks like **Jupyter** or **Databricks** to begin your analysis.
2. **Load a Batch Dataset:**
    a. Import a complete dataset (e.g., CSV, JSON) that contains historical data with a defined beginning and end.
3. **Understand the Data:**
    a. Inspect column names, types, and sample values to get a sense of the dataset and identify what needs fixing.
4. **Clean the Data**

   a. Fix issues like missing entries, rename confusing column names, correct data types, and remove repeated rows.
5. **Perform Data Transformations**
   a. Use filters, groupings, and aggregations (like total sales or average rating) to get meaningful insights from the data.
6. **Save or Visualize the Results**
   a. Output can be written to files, shown in the console, or used to create visualizations using dashboards.

- **Steps for Streamed Data Analysis using Apache Spark**
  1. **Set Up Structured Streaming:** Start a SparkSession configured for Structured Streaming, Spark's modern method for processing live data.
  2. **Connect to a Real-Time Data Source:** Hook Spark up to sources like Kafka, sockets, or directories where fresh data files are added regularly.
  3. **Define a Data Schema:** Since streaming data often lacks headers, you need to manually define what each field represents (e.g., time, value, ID).
  4. **Apply Logic While Data Flows:** As new data comes in, apply transformations like filtering by condition (e.g., values above a certain threshold), time-based grouping (e.g., every 10 seconds), or calculations (e.g., max, average).
  5. **Send Results to a Destination:** Output the continuously updated results to a console, file, database, or even live dashboards.
  6. **Keep an Eye on the Stream:** Monitor the pipeline for speed, data flow, and memory usage. Spark keeps the job running until you stop it manually.

CONCLUSION:

This experiment helps us understand two powerful ways of working with data using Apache Spark: batch and streamed processing. Batch analysis is well-suited for historical or static data, allowing deep dives and summary reports. On the other hand, streamed analysis enables real-time decision-making by processing live data as it arrives. Apache Spark handles both seamlessly using its unified framework and scalable infrastructure. By learning both approaches, we equip ourselves with the flexibility to tackle a wide variety of real-world data problems whether they require immediate insight or deep historical trends.

Shivam R. Prajapati AIDS-1 D15C41

23/03/25

Assignment No. 1 (05/05)

Page No. Date

(1) What is AI? Considering the covid-19 pandemic situation, how AI helped to evolve and renovated our way of life with different applications

Soln:- Ans Artificial Intelligence (AI) is a branch of computer science that enables machines to perform tasks that typically require human intelligence. These tasks include learning, reasoning, problem solving, perception, language understanding and decision making.

(1) AI-powered algorithms analyzed news, social media, and official reports to detect early signs of outbreaks.

(2) Drug Discovery and vaccine development :- AI accelerated the development of COVID-19 treatments and vaccines by analyzing protein structures and predicting potential drug candidates.

(3) Medical diagnosis :- AI powered diagnostic tools analyzed Xrays and CT scans to detect COVID-19 infections faster than traditional methods.

(4) Contact Tracing and spread prediction :- AI models helped track virus spread using data from mobile apps, GPS and social media.

(5) Remote Work and online learning :- AI-powered collaboration tools (Zoom, Teams) enabled remote work and virtual meetings.

(6) Mental Health and wealth being :- AI chatbots like Wysa and Woebot provided mental health support to people dealing with stress and isolation

Page No.
Date

(2) What are AI agents terminology, explain with examples.

Soln:- An AI agent is an entity that perceives its environment through sensors and acts upon it using actuators to achieve a specific goal. AI agents operate based on decision making models, algorithm and data.

(1) Agent:-
An agent is any system that perceives its environment and takes actions to achieve a goal.
Eg:- A self driving car perceives traffic signals and road conditions and takes actions like stopping and accelerating.

(2) Environment:-
The environment is everything that surrounds an AI agent and affect its performance.
Eg:- In a chess playing AI, the chessboard, opponent, moves, the rules from the environment.

(3) Perception:-
Perception is the agents ability to gather information from the environment through sensors.
Eg:- A facial recognition system perceives images through a camera.

(4) Sensors:-
Sensors are the input devices that collect data from the enviroment.
Eg:- A voice assistant (like Alexa) uses a microphone

Page No.
Date

(5) Actuators :-
Actuators are components that allow the agent to take action based on decisions
Eg :- In a robot, its arm and wheels are actuators.

(6) Rational agent :-
A rational agent takes the best possible action to maximise its performance measure.
Eg :- AI stock trading system makes decisions to maximise profit.

(7) Utility function :-
A utility function measures how good agent is for achieving a goal.
Eg :- In a game AI, winning a match has a high utility score.

(8) How AI technique is used to solve 8 puzzle problem.

Soln :- The 8 puzzle problem is a classic problem in artificial intelligence that involves sliding tiles on a 3×3 grid to reach a goal state.

Representation of the 8 puzzle problem :-
(1) Initial state :- Any random arrangement of tiles
(2) Goal state :- A predefined arrangement (eg numbers in order)
(3) Operators :- Move the blank side (up, down, left, right)
(4) state space :- All possible tile arrangements

Page No.
Date

(5) Cost function :- Typically, each move has a cost of 1

Common heuristics for A* :-

(1) Misplaced Tile Heuristic (h_1) :- counts the number of misplaced tiles.

(2) Manhattan Distance (h_2) :- measures the sum of distances each tile is from its goal position. More accurate than misplaced tiles - heuristic.

(4) What is PEAS descriptor ? Give PEAS descriptor for following

Soln :- The PEAS (performance, measure, environment, actuators, sensors) descriptor is used to define the components of an agent, helping to analyze its working environment and functions.

- Performance measure (P) :- The criteria for evaluating the agent's success.
- Environment (E) :- The external surroundings where the agent operates.
- Actuators (A) :- The mechanisms through which the agent interacts with the environment
- Sensors (S) :- The device used to perceive the environment.

(1) Taxi driver

P : safety, fuel efficiency, passenger satisfaction

E : Roads, traffic signals, other vehicles, pedestrians, weather

A: steering wheel, accelerator, brakes, horn, indicators, wipers

S: GPS, cameras, LiDAR, speed sensors, fuel guage, odometer etc.

(2) Medical diagnosis system :-

P: Diagnosis, accuracy, speed of response, patient recovery etc.

E: Patients, symptoms, medical databases, hospital environments

A: Display screen, speaker (for communication with doctor and patients)

S: Patients history, test reports, X-rays, MRI scans, symptom inputs

(3) A Music composer :-

P: Quality of generated music, creativity, audience engagement

E: Musical database, sound libraries, listener preferences

A: Music output system (MIDI, speakers, digital files)

S: user feedback, musical trends, mood analysis, input instruments

(4) An aircraft autolander :-

P: smooth and safe landing, fuel efficiency, passenger comfort.

E: Runway, weather conditions, altitude, air traffic

A: Flaps, landing gear, engine & throttle, air brakes.

S: GPS, altimeter, wind sensors, gyroscopes, radar.

(5) An essay evaluator :-

P: Accuracy in grading, fairness, grammar and colaborate detection

E: Essays, answer sheets, writing rules, academic guidelines

A: Score display, feedback generator. S: Optimal character recognition (OCR), NLP tools.

(6) Robotic sentry gun for the kock lab :-
P: Accuracy in target detection, response-time
E: Lab premises, intruders, authorized personnel, obst.
A: Gun turret, alarm system, movement motors
S: Motion detectors, thermal cameras, facial recognition, infrared sensors

(5) Categorize a shopping bot for an offline bookstore according to each of the six dimensions (fully / partially observable, deterministic / stochastic, episodic / sequential, static / dynamic, discrete / continous, single / multi-agent)

Soln :-

(a) Observability :- partially observable.
The bot may not have complete information about books on shelves, customer preferences or stock updates without external input.

(b) Deterministic vs stochastic :- stochastic
Book availability may change due to manual sales, external purchases, or misplaced books, making environment uncertain.

(c) Episodic vs sequential :- Sequential
Each customer interaction affects the next steps (eg. book recommendations depend on previous queries)

(d) Static vs Dynamic :- Dynamic
The environment changes (books sell out, new books arrive, customer preferences shift) making it dynamic.

(e) Discrete vs continous : Discrete
The bot deals with a finite set of actions (searching books, checking stock)

Page No.
Date

(f) Single Agent VS Multiagent :- Multiagent.
The bot interacts with multiple customers, bookstore staff and possibility other inventory systems.

(6)
Q.6 Differentiate Model based and utility based agent.

Soln:-

Model - Based agent	Utility Based agent.
(1) Uses an internal model of the environment to make decisions	(1) uses a utility function to measure the desirability of different states and select the best action
(2) Chooses actions based on a representation of how the world works.	(2) Chooses actions that maximizes its expected utility
(3) Maintains a model of the environment (how actions affect future states).	(3) Uses a utility function to compare possible outcomes.
(4) Focuses on achieving a goal using state transition models	(4) Focuses on maximising long term benefits rather than just reaching a goal.
(5) Eg:- A self driving car that predicts patterns and plan routes.	(5) A stock trading bot that evaluates different portfolios to maximise returns.

Q.7 Explain the architecture of knowledge based agent and learning agent

Soln:

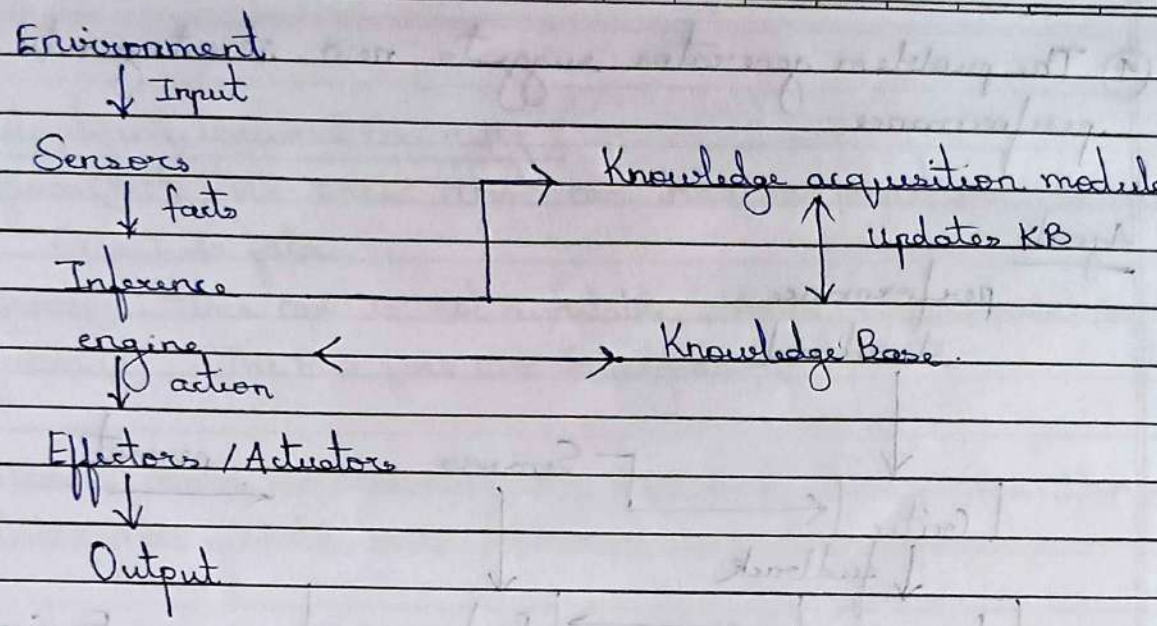
Architecture of knowledge based agent

A knowledge based agent (KBA) is an AI system that uses stored knowledge to make informed decisions. It consists of the following components :-

1. KB :
It stores facts, rules and heuristics about the world
2. Inference engine :-
It applies logical reasoning to derive new knowledge from stored facts.
3. Perception (sensors) :-
It collects information from the environment
4. Actuators
Performs actions based on inferences
5. Knowledge acquisition module :-
Updates and expands the knowledge base with new data

Working process

1. The agent perceives the environment
2. It queries the knowledge base for relevant information
3. The inference engine applies logic to decide an action
4. The action is executed and KB is updated if needed.



Architecture of learning agent :-

A learning agent improves its performance overtime by learning from past experiences.

Components of the learning agent :-

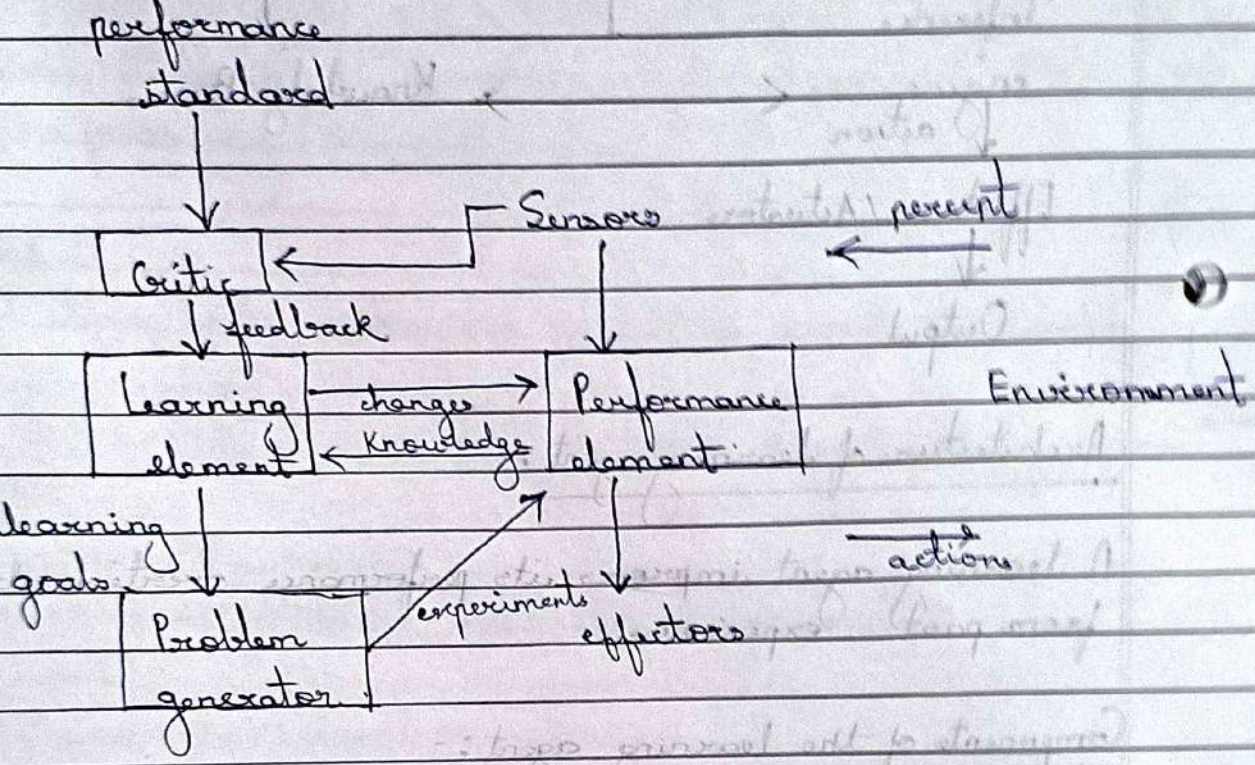
1. Learning element :- Improves the agent's knowledge by analyzing past experiences
2. Performance element :- chooses actions based on the learned knowledge.
3. Critic :- Evaluates the agent's actions by comparing outcomes with expected results
4. Problem generator :- suggests new exploratory actions to improve learning.

Working process :-

1. The performance element makes decisions and takes actions
2. The critic evaluates the results and provides feedback.
3. The learning element updates the knowledge based on feedback

④ The problem generator suggests new strategies to improve performance.

Agent



⑤ Convert the following to predicates.

a) Anita travels by car if available otherwise travels by bus.

Available(car) → Travels (Anita, car)

~ Available (car) → Travels (Anita, ~~car~~ Bus) — ①

b) Bus goes via Andheri and goregoan.

Goes via (Bus, Andheri) ∧ Goes via (Bus, Goregoan) — ②

c) Car has a puncture, so it is not available

puncture(car) → ~ Available (car) — ③

will Anita travel via Goregoan.

Applying forward chaining :

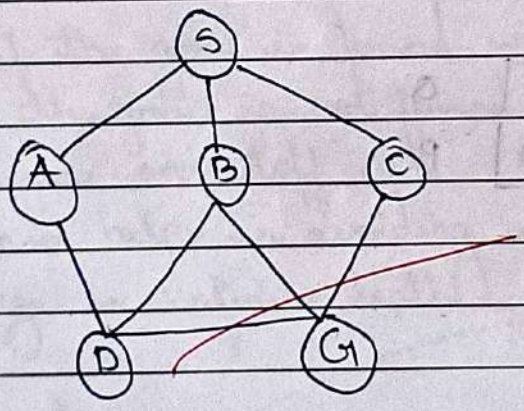
from ③ :- We know that car has a puncture, so available (car) is false.

from ① : Since car is not available, Anita will travel by bus

from ② : The Bus goes via Goregaon.

Hence, Anita is traveling by Bus and Bus passes through Goregaon, Anita will travel via Goregaon.

④ Find the route from S to G using BFS.



Soln :- step ① Queue (Q)

Processed (P)

step ① :-

S			Q
			P

step ② :

A	B	C	Q
S			P

step ③ :-

B	C	D	Q
S	A		P

step ④ :-

C	D	G	Q
S	A	B	P

step ⑤ :-

D	G		Q
S	A	B	C	P

step ⑥ :-

G		Q
S	A	B	C	D	P

step ⑦ :-

Q
S	A	B	C	D	G	P

Adjacency list :-

S → {A, B, C}
A → {D}
B → {D, G}
C → {G}
D → {G}

from BFS and adjacency list
shortest path is S → B → G
other paths are S → C → G and S → B → D → G
and S → A → D → G.

(10) What do you mean by depth limited search? Explain iterative deepening search with example.

Soln :- **Depth limited search (DLS) :-**

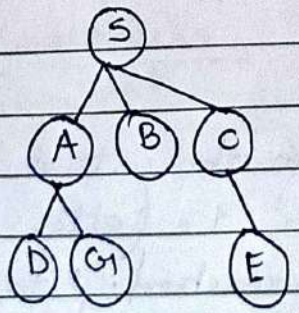
Depth limited search is a variation of DFS where we impose a depth limit to avoid going too deep into an infite or large search space.

Working :-

1. The algorithm follows a DFS strategy but limits the depth of recursion
2. If the goal is found within the limit, the search stops
3. If the goal is not found within the limit, it returns failures or cutoff
4. This helps in avoiding ~~infite~~ infinite loops in problems with large or infinite depth.

Example :-

Consider a graph where we want to find a path from S to G with depth limit of 2



- If the depth limit is 1, we only explore S → A, B, C but cannot reach G.
- If the depth limit of 2, we explore S → A → D, G.

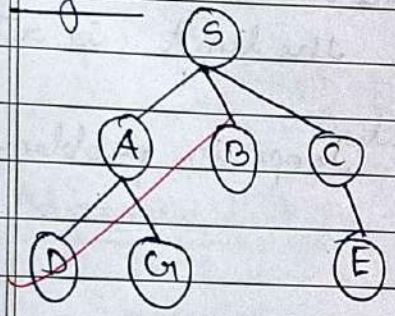
Iterative deepening search (IDS) :-

Iterative deepening search (IDS) combines the benefits of depth-first search (DFS) and BFS. It repeatedly performs DLS with increasing depth limits until the goal is found.

Working :-

1. Start with depth limit = 0 and perform DLS
2. Increase the depth limit and perform DLS again.
3. Repeat until goal is found.

Eg :-



- Depth limit = 0 → only node S is checked.
- Depth limit = 1 → explores A, B, C but G is not found
- Depth limit = 2 → explores D, G, E and find G.

(11) Explain Hill Climbing and its drawback in detail with example. Also state limitations of steepest - ascent hill climbing

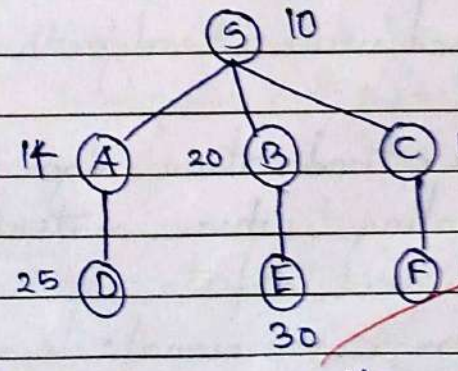
Soln :-

Hill Climbing is an optimization algorithm that continuously moves towards higher-valued states i.e (better solutions) until a peak (local maximum) is reached. It is a greedy algorithm that evaluates neighbouring states and chooses the one with the highest value. It is widely used in AI, especially in problems like pathfinding, scheduling problems

Working of Hill Climbing :-

1. Start with an initial state (solution)
2. Evaluate the neighbouring states of the current state.
3. Move to the best neighbouring state that improves the solution
4. Repeat the process until no better neighbour is found or predefined goal state is reached.

Eg:- Consider a graph where each node has value representing its height and goal is to find the highest valued node.



1. Start at node S(10)
2. Check its neighbours : A, B, C
then move to B (highest value = 20)
3. Check neighbours of B : E(30)
then move to E (highest value = 30)
4. E has no better neighbour so stop
5. Final result : Node E (value = 30) is the peak.

Drawbacks

1. Local Maxima :- If Node D(25) was chosen from A instead of B, it would be stuck there, not reaching 30.
2. Plateau :- If multiple nodes had the same value, the algorithm might get confused.
3. Ridges :- The algorithm cannot take downward moves to explore better paths.
4. No backtracking :- Hill Climbing does not remember previous states, so if it gets stuck, it cannot backtrack to explore better paths.

Steepest - Ascent Hill climbing and its limitations :-

Steepest-ascent hill climbing is a variation where the algorithm evaluates all neighbouring states and moves to the one with the highest improvement.

Limitations :-

1. Since it evaluates all neighbours, it takes more time and resources.
2. Even though it selects the best move at each step, it cannot escape local maxima.
3. Fails in plateaus and ridges.

12. Explain simulated annealing and write its algorithm.

Soln :- Simulated annealing (SA) is an optimization algorithm inspired by metallurgical annealing, where materials are heated and then cooled to remove defects. It helps escape local maxima by allowing occasional downward moves to explore better solutions.

Working :-

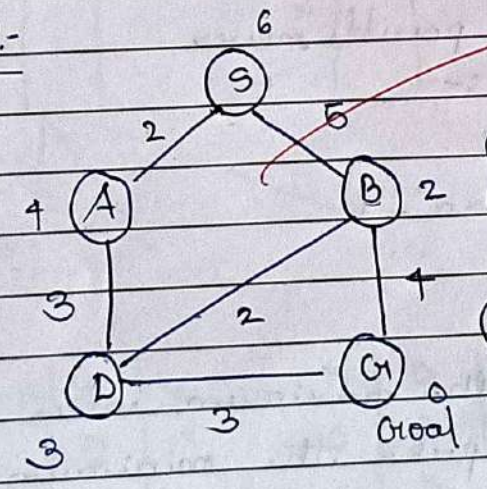
1. Start with an initial solution
2. set a high temperature (T), which gradually cools down
3. Select a random neighbour of the current solution.
4. Calculate the energy difference (ΔE) between the new and current solution
   a. If the new solution is better, accept it.
   b. If the new solution is worse, accept it with a probability $P = e^{-\Delta E/T}$ where e is Euler's number, T is current temperature

(5) Reduce the temperatures gradually
(6) Repeat until the temperature is very low or a stopping condition is met.

Q.13 Explain A* with an example.

Soln :- A* is widely used graph search and pathfinding algorithm that finds the shortest path between a start node and a goal node. It is an informed search algorithm that uses ~~both~~ both
(1) Cost to reach the node [g(n)]
(2) Estimated cost from the node to the goal [h(n)]
Formula : f(n) = g(n) + h(n)

Eg :-



Steps
(1) Initialize open list
(2) expand node with lowest f(n)
(3) update g(n), h(n) and f(n) for neighbouring
(4) Repeat until goal node is reached

Node	g(n)	h(n)	f(n) = g(n)+h(n)
start (S)	0	6	6
Expand A (S → A, cost = 2)	2	4	6
Expand B (S → B, cost = 5)	5	2	7
Expand D (S A → D, cost = 2+3 = 5)	5	3	8
Expand G (B → G, cost = 5+4 = 9) (Goal reached)	9	0	9

(14) Explain minimax algorithm and draw game tree for Tic-Tac Toe Game

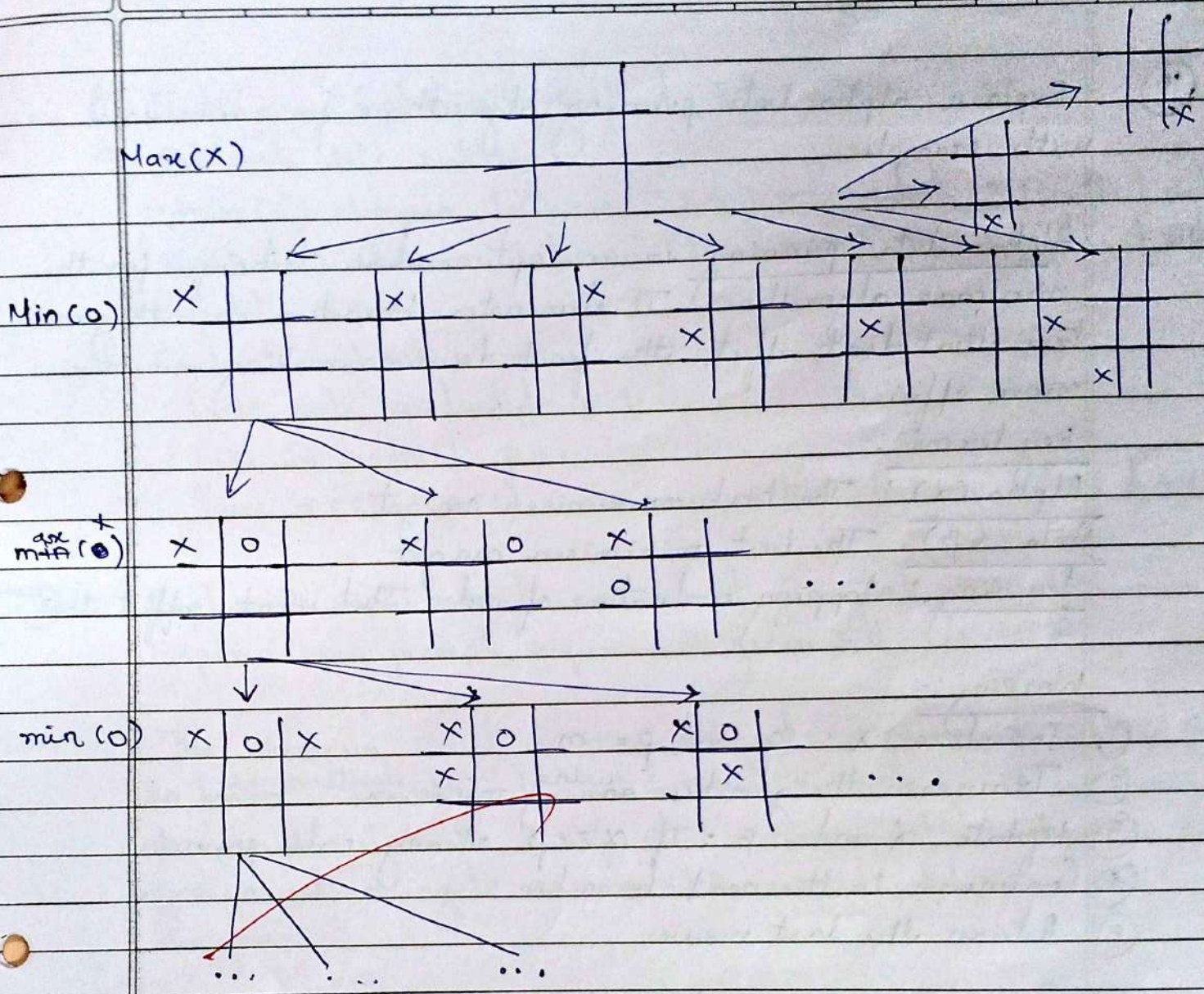
Soln:- Minimax is a decision making algorithm used in two-player games like Tic-Tac-Toe, chess, and connect four. It helps in finding the best possible move for a player by assuming that the opponent also plays optimally. The algorithm alternates between maximizing (for AI) and minimizing (for opponent) and each state has a value.

+1 → AI wins
-1 → opponent wins
0 → Draw

Algorithm steps

1. Generate game tree for all possible moves.
2. Evaluates Terminal states :-
   - If AI wins, return +1
   - If opponent wins, return -1
   - If draw, return 0
3. Backpropagate values
   - Max player (AI) picks the maximum value.
   - Min player (opponent) picks the minimum value.
4. Select the best move at the root level.

Game tree :-



(15) Explain alpha beta pruning algorithms for adversarial search with example

Sol:- Alpha-beta pruning is an optimization technique for the minimax algorithm. It eliminates branches in the game tree that don't affect the final decision, making minimax more efficient

Key terms :-

Alpha (α) : The best maximiser can get

Beta (β) : The best minimiser can get

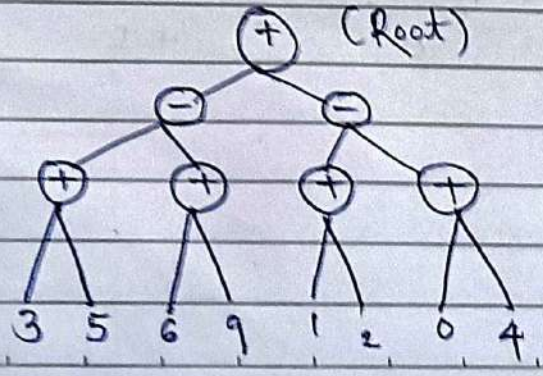
Pruning : stopping evaluation of nodes that won't affect the result

Working :-

1. Initialize $\alpha = -\infty$ and $\beta = -\infty$
2. Traverse the gametree using minimax
3. Update α and β : If $\alpha \geq \beta$ at any node, prune
4. Continue to the next branches.
5. Return the best move.

Example :-

Consider the following gametree where max(⊙) wants to maximise the score, and min(⊖) wants to minimise it.



(1) Initialize : Alpha (α) = $-\infty$ and, Beta (β) = $+\infty$

(2) Evaluate left subtree
- min (O) chooses between (3, 5) ⟶ selects 3 (smallest value)
- max (O) chooses between (3, 6) ⟶ selects 6 (largest value)
- update $\alpha = 6$

(3) Evaluate right subtree
- first node evaluated is 1
- beta is updated $\beta = 1$
- Since $\alpha(6) \geq \beta(1)$, we prune (skip checking 2, 0 and 4)

(16) Explain Wumpus world environment giving it PEAS description. Explain how percept sequence is generated.

Soln :- The Wumpus world is a grid based environment used in (AI) to demonstrate logical agent - based reasoning. It is a partially observable, stochastic, sequential environment where an AI agent must navigate a cave-like world while avoiding dangers.

Structure of wumpus world :-
(1) Grid Based (4×4 or larger)
(2) Contains
a) Gold (Goal pick it up)
b) Wumpus (A monster that kills the agent)
c) pits (If agent falls, it dies)
d) Walls (Boundaries of the world)
(3) Sensory perceptions (percepts)
a) Breeze → near a pit
b) Stench → Near wumpus
c) Glitter → gold is nearby
d) Bump → hits a wall
e) Scream → wumpus is dead

PEAS description

P: +1000 for finding gold, -1000 for falling into a pit or encountering wumpus, -1 for every move, -10 for shooting the arrow.

E: A 4×4 grid with the agent, wumpus, pits, gold and walls

A: Move (up, down, left, right), grab (gold), shoot (arrow), climb (exit)

S: Breeze, stench, glitter, bump, scream.

A percept sequence is a history of all sensor inputs received by the agent over time.

Eg:-

1. Agent starts at (1,1) → percepts = (Breeze) (pit nearby)
2. Moves to (1,2) → percepts = (Breeze, Stench) (pit + wumpus)
3. Moves to (2,2) → percepts = (Stench) (near wumpus)
4. Moves to (3,2) → percepts = (Glitter) (gold is nearby)

Agent collects percepts at each step and makes logical decisions based on past percepts to avoid dangers and find goal.

(17) Solve the following crypto-arithmetic problems
SEND + MORE = MONEY.

Soln:- Step①: Assign unique digits to letters.
Each letter represents a unique digits (0-9)
The goal is to find a valid assignment where sum is correct.
- S, E, N, D, M, O, R, Y are distinct digits.
- S and M cannot be 0 (since they are leading digits.

Step ②:- Convert to column wise Addition.
Arranging the numbers in column format:

```
  SEND
+ MORE
-------
 MONEY
```

In expanded form,

$(1000S + 100E + 10N + D) + (1000M + 100O + 10R + E) =$
$10000M + 1000O + 100N + 10E + Y$

Step ③ :- Solve step by step

① Identifying M
- Since MONEY have five digits
- M must be 1 (because S+M carries over)

② Finding O
- Since MORE contributes a carry to MONEY, the sum of SEND + MORE must be over 9999.
- The only digit left for O that works is 0

③ Determining S
Since S+M = 10 and M=1 it means S = 9

④ Finding other values.
Using logical constraints and testing values, the correct assignment is

S=9 , N=6 , M=1 , R=8
E=5 , D=7 , O=0 , Y=2

Step ④ :- Verify

```
  9567
+ 1085
-------
 10652
```

The sum is 10652, which matches MONEY.

∴ S=9 , E=5 , N=6 , D=7 , M=1 , O=0 , R=8 , Y=2

(19) Consider the following axioms :-
All people who are graduating are happy
All happy people are smiling
Someone is graduating
Explain the following :-
(1) Represent these axioms in first order predicate logic
(2) Convert each formula to clause form.
(3) Prove that " is someone smiling ?" using resolution technique
Draw the resolution tree.

Soln :- Step(1) :- Representing the axioms in first order predicate logic
G(x) : x is graduating
H(x) : x is happy
S(x) : x is smiling
Using these predicates the axioms can be written as
(1) Axiom 1 : " All people who are graduating are happy "
$\forall x (G(x) \rightarrow H(x))$
(2) Axiom 2 : " All Happy people are smiling "
$\forall x (H(x) \rightarrow S(x))$
(3) Axiom 3 : " Someone is graduating "
$\exists x G(x)$

Step(2) :- Convert each Formula to Clause Form.
Axiom 1 :- $\forall x (G(x) \rightarrow H(x))$
Convert implication
$\forall x (\sim G(x) \vee H(x))$
Convert to clause form
$\sim G(x) \vee (H(x))$ (clause 1

Axiom 2: $\forall x (H(x) \rightarrow S(x))$ becomes
$\sim H(x) \vee S(x)$ Clause 2.

Axiom 3: $\exists x G(x)$
Existential quantifier is eliminated
$G(a)$ Clause 3

Step ③ :-
To prove that someone is smiling we need to show $\exists x S(x)$
Using proof by contradiction, we assume the negation of this statement $\sim S(x)$

Applying resolution :-
We start with known clauses
Clause 1 : $\sim G(x) \vee H(x)$
Clause 2 : $\sim H(x) \vee S(x)$
Clause 3 : $G(a)$
Negated Goal : $\sim S(a)$
Now we apply resolution step by step :
① Resolve (Clause 1) with (Clause 3) :
$\sim G(a) \vee H(a)$
$G(a)$
Resolution : Remove $G(a)$ since it cancels $\sim G(a)$
New Clause : $H(a)$
② Resolve (Clause 2) with $H(a)$:
$\sim H(a) \vee S(a)$
$H(a)$
Resolution : Remove $H(a)$ since it cancels $\sim H(a)$
New clause : $S(a)$

③ Resolve S(a) with ~S(a) (Negated Goal):

S(a)

~S(a)

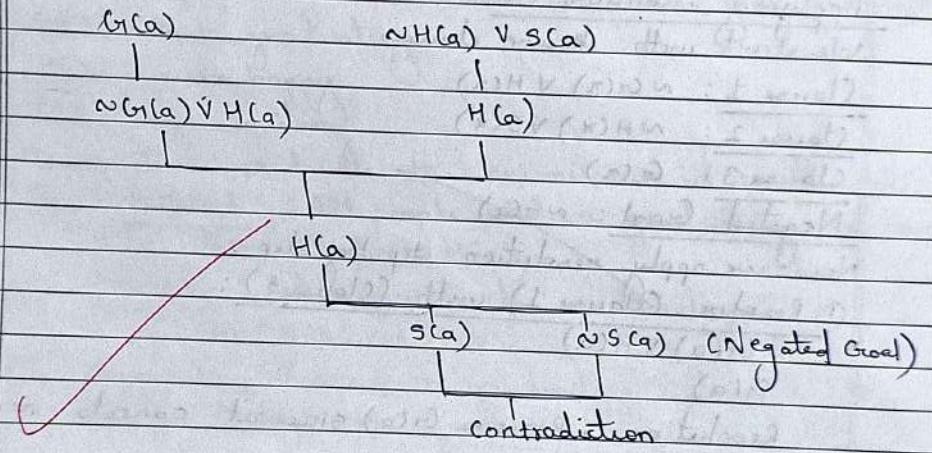
Contradiction (Empty clause)

Since we derived a contradiction, our initial assumption (~S(a)) is false, proving that:

$\exists x\, S(x)$

i.e Someone is Smiling

Step ④:- Resolution Tree



Q.20 Explain Modus Ponen with suitable example.

Soln:- Modus Ponens is a fundamental rule of inference in logic. It follows the structure:

① If P, then Q (Conditional statement)

② If P is true (Premise)

③ Therefore, Q is True (Conclusion)

Example:-

statement:-
① If it rains, the ground will be wet (If P, then Q)
② It is raining (P is true)
③ Therefore, the ground is wet (Q is true)

This rule is widely used in logical reasoning and mathematical proofs.

Q.21. Explain forward chaining and backward chaining algorithm with the help of example

Soln:- Forward chaining:-

Forward chaining starts with known facts and applies rules to infer new facts until a goal is reached. It is a bottom-up approach.

Eg:-
Consider a medical diagnosis System

Rules:-
R1: If a person has a fever and cough then they may have the flue
R2: If a person has a flu, then they should take rest

Facts:
① A patient has a fever
② The patient has a cough.

Process:-

(1) The system checks R1: Since the patient has a fever and cough → They may have the flu
(2) The system checks R2: Since the patient has the flu → They should take rest

Conclusion:- The patient should take rest.

Algorithm:-

(1) Initialize: Start with a set of known facts in the KB
(2) Match rules: Identify rules whose conditions match the known facts.
(3) Apply rules:- If a conditions are satisfied, infer the new fact (conclusion)
(4) Update KB: Add the newly inferred fact to the KB.
(5) Repeat: Continue the process until Either :-
   - The goal fact is derived or
   - No new facts can be inferred.

Backward Chaining:-

Backward chaining starts with the goal and works backward to determine if there is evidence to support. It is a top-down approach.

Algorithm:-

(1) Start with the goal:- Identify the target fact that needs to be proved.
(2) Check if the goal is already a known fact :-
   - If yes, stop (goal is achieved)
   - If no, proceed to next step.

(3) Find rules that conclude the goal:
- Identify rules where the goal is the conclusion
- Check if the rules conditions are met

(4) Verify premises:-
- If all premises are known facts, apply rule and derive the goal
- If some premises are unknown, set them as new sub-goals and repeat the process

(5) Continue recursively until:
- The goal is proved (success), or
- No supporting facts are found (failure)

Example: Diagnosing disease

Rules:
R1: If a person has a fever and cough, then they may have the flu.
R2: If a person has the flu, then they should take rest
R3: If a person has a sore throat and fever, then they may have a throat infection

Facts:
- A patient has a fever
- The patient has a cough.

Goal: Determine if the patient has the flu.

Process:

(1) The system checks R1: "If a person has fever and cough, then they may have the flu".
(2) It asks: "Does the patient have a fever?"
→ Yes.

DISC 41

(3) It asks: "Does the patient have a cough?" → Yes.
(4) Since both conditions are met, the system confirms "The patient has the flu".

***

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Assignment No: 2

Q.1] Use the following data set

82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90

1.Find the Mean

2. Find the Median

3. Find the Mode

4. Find the Interquartile range

Soln:

Dataset: 82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90

1. Mean :

$$\text{Mean} = \frac{\sum x_i}{n}$$

$$\text{Mean} = \frac{(82+66+70+59+90+78+76+95+99+84+88+76+82+81+91+64+79+76+85+90)}{20}$$

$$= \frac{1611}{20}$$

Mean = 80.55

2. Median:

Sort values: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Total values 20 that is Total Even Values are there

$$\text{Median} = \frac{x_{(n/2)} + x_{(n/2)+1}}{2}$$

Even number of values → average of 10th and 11th:

$$\text{Median} = \frac{81+82}{2} = 81.5$$

Median = 81.5

3. Mode:

The mode is the value that appears most frequently in a dataset.
Most frequent value = 76
Mode = 76

4. Interquartile Range (IQR)

$$\mathrm{IQR} = Q3 - Q1$$

Q1 = 25th percentile = average of 5th and 6th

Q1 = $\frac{76+76}{2} = 76$

Q3 = 75th percentile = average of 15th and 16th

Q3 = $\frac{88+90}{2} = 89$

IQR = Q3−Q1

= 89−76

IQR =13

Therefore,
Mean = 80.55 ,
Median = 81.5 ,
Mode = 76 ,
IQR = 13

Q.2] 1) Machine Learning for Kids 2) Teachable Machine

1. For each tool listed above

- **identify the target audience**
- **discuss the use of this tool by the target audience**
- **identify the tool's benefits and drawbacks**

2. From the two choices listed below, how would you describe each tool listed above? Why did you choose the answer?

- **Predictive analytic**
- **Descriptive analytic**

3. From the three choices listed below, how would you describe each tool listed above? Why did you choose the answer?

- **Supervised learning**
- **Unsupervised learning**
- **Reinforcement learning**

Soln:

The following Machine Learning Tools Comparison:

1. Machine Learning for Kids
2. Teachable Machine

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

1] Machine Learning for Kids:

- Target Audience are School students (ages 8 to 16), beginners, and educators teaching AI/ML in schools or basic courses.
- Users can train ML models using Text, Images, Numbers
- It connects with platforms like Scratch and Python, allowing users to build interactive projects like:
    - A chatbot that detects positive/negative messages
    - An app that recognizes fruits from images
- Example :A student can upload labeled photos of cats and dogs and then use Scratch to make a game that guesses whether a new image is a cat or dog.
- Advantages:
    - User-friendly interface, great for young learners.
    - Integrates ML with visual programming (Scratch).
    - Encourages creative projects and experimentation.
    - No coding required (but optional Python use is available).
    - Cloud-based, accessible from browsers.
- Limitations:
    - Limited to basic models; lacks depth for real-world ML applications.
    - Accuracy is low compared to professional tools.
    - Minimal control over algorithm types, hyperparameters, or data preprocessing.
    - Not suitable for large datasets or complex models.

2] Teachable Machine:

- Target Audiences are General public, students, educators, hobbyists, and even artists.
- It is use for:
    - Creating models by training with webcam/audio/images.
    - Exporting the model to use in websites or apps.
    - Because No coding required.
- Advantages:
    - Extremely simple to use with a few clicks.
    - Fast real-time training with immediate feedback.
    - Supports exporting trained models to real applications.
    - Great for interactive demos, art installations, and education.
- Limitations:
    - No deep customization or control over the model architecture.
    - No preprocessing options (e.g., normalization).
    - Limited dataset size and simple structure = low accuracy on complex problems.
    - Cannot handle text or numerical data.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

2. Choosing Predictive or Descriptive Analytic:

Tool	Type	Reason
Machine Learning for Kids	Predictive Analytic	It uses trained models to predict categories or outputs from inputs.
Teachable Machine	Predictive Analytic	It predicts labels for new input data (like image or sound classification).

We have not chosen descriptive because Descriptive analytics explains what happened in the past using statistics and visualization. These tools instead predict outcomes using new input data.

3. Choosing Type of Learning.

Tool	Learning Type	Reason
Machine Learning for Kids	Supervised Learning	It uses labeled data (e.g., text labeled as positive/negative) to train.
Teachable Machine	Supervised Learning	Users provide labeled examples for training (e.g., face = "happy").

We have not chosen Unsupervised or Reinforcement because

1) These tools don't discover hidden patterns or reward strategies on their own.
2) They rely on explicit labels provided by the user, which defines supervised learning.

Q.3] Data Visualization: Read the following two short articles:

Read the article Kakande, Arthur. February 12. "What's in a chart? A Step-by-Step Guide to Identifying Misinformation in Data Visualization." Medium

Read the short web page Foley, Katherine Ellen. June 25, 2020. "How bad Covid-19 data visualizations mislead the public." Quartz Research a current event which highlights the results of misinformation based on data visualization.

Explain how the data visualization method failed in presenting accurate information. Use newspaper articles, magazines, online news websites or any other legitimate and valid source to cite this example. Cite the news source that you found.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Soln:

Case Study: Misleading COVID-19 Vaccine Death Visualizations

In early 2024, misleading COVID-19 vaccine-related visuals began circulating widely on social media platforms. These visuals suggested that vaccinated individuals in England experienced more deaths than those who were unvaccinated between July 2021 and May 2023. At a glance, the charts looked alarming and prompted concern—but they lacked critical context, leading to serious misinterpretation. (Source: Reuters, March 21, 2024 – "Misleading data used to claim COVID vaccines do more harm than good")

Where the Visualization Went Wrong:

1. **Lack of Proportional Context**:
   The visuals displayed the **total number of deaths** in each group (vaccinated vs. unvaccinated) without acknowledging that the **vast majority of England's population had been vaccinated** during this period. Naturally, with more people in the vaccinated group, the raw number of deaths would be higher—but that doesn't indicate higher risk. Without adjusting for group size, the comparison becomes misleading.

2. **Missing Essential Background**:
   The charts **did not explain how effective vaccines are**, nor did they highlight the **difference in group sizes**. This omission led many viewers to assume that vaccines were ineffective or dangerous, which is a misinterpretation based purely on incomplete data presentation.

3. **Ignoring Death Rates**
   Instead of showing **deaths per 100,000 people** (which allows for fair comparison between differently sized groups), the charts showed raw death counts. This skewed the perceived risk, concealing the fact that **unvaccinated individuals were statistically at greater risk of dying** from COVID-19 during this time frame.

Clarifying the Misrepresentation:

Experts reviewed the same data with proper adjustments—especially accounting for **population size**—and revealed a different story. According to the UK's Office for National Statistics (ONS), **death rates among vaccinated individuals were significantly lower** than among unvaccinated individuals. The issue wasn't with the data itself, but rather how it was **visualized and interpreted**. When data lacks context, it can easily fuel confusion or spread misinformation.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

Conclusion:

This case highlights how **critical it is to design data visualizations responsibly**. Charts and graphs that omit proportionality, context, or clear labeling can mislead audiences—even if the data is technically correct. In public health, such misinterpretations can have dangerous consequences by eroding public trust. As noted by experts like Arthur Kakande and Katherine Ellen Foley, both creators and consumers of data must be **vigilant, critical, and context-aware** to ensure that visuals truly inform rather than mislead.

Q. 4] Train Classification Model and visualize the prediction performance of trained model required information

- **Data File: Classification data.csv**
- **Class Label: Last Column**
- **Use any Machine Learning model (SVM, Naïve Base Classifier)**
  Requirements to satisfy
- **Programming Language: Python**
- **Class imbalance should be resolved**
- **Data Pre-processing must be used**
- **Hyper parameter tuning must be used**
- **Train, Validation and Test Split should be 70/20/10**
- **Train and Test split must be randomly done**
- **Classification Accuracy should be maximized**
- **Use any Python library to present the accuracy measures of trained model**

Pima Indians Diabetes Database

Soln:

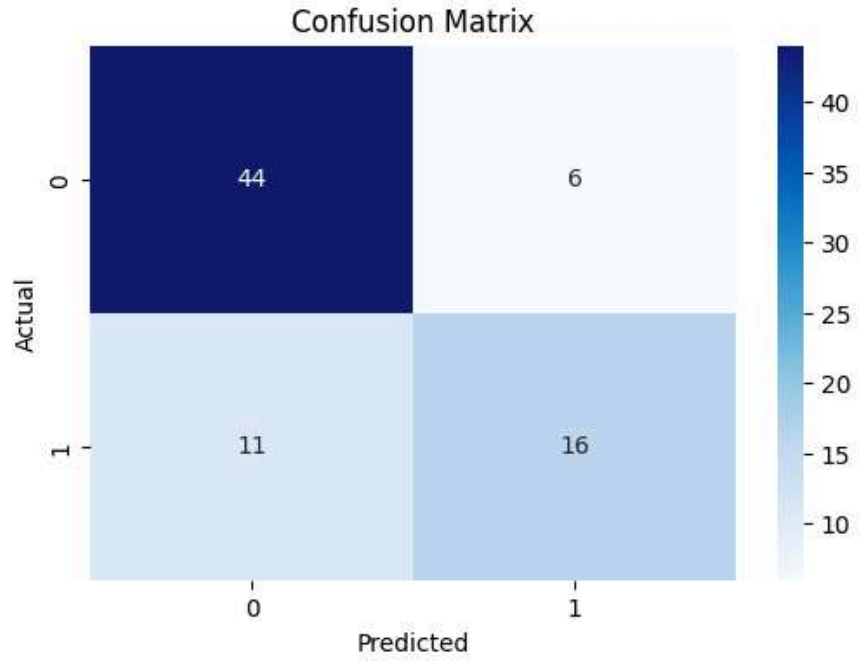
- **Pregnancies**: Number of times the patient has been pregnant.
- **Glucose**: Plasma glucose concentration after a 2-hour oral glucose tolerance test.
- **BloodPressure**: Diastolic blood pressure (mm Hg).
- **SkinThickness**: Triceps skin fold thickness (mm).
- **Insulin**: 2-hour serum insulin (mu U/ml).
- **BMI**: Body Mass Index (weight in kg / height in m^2).
- **DiabetesPedigreeFunction**: A function that scores the likelihood of diabetes based on family history.
- **Age**: Patient age in years.

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

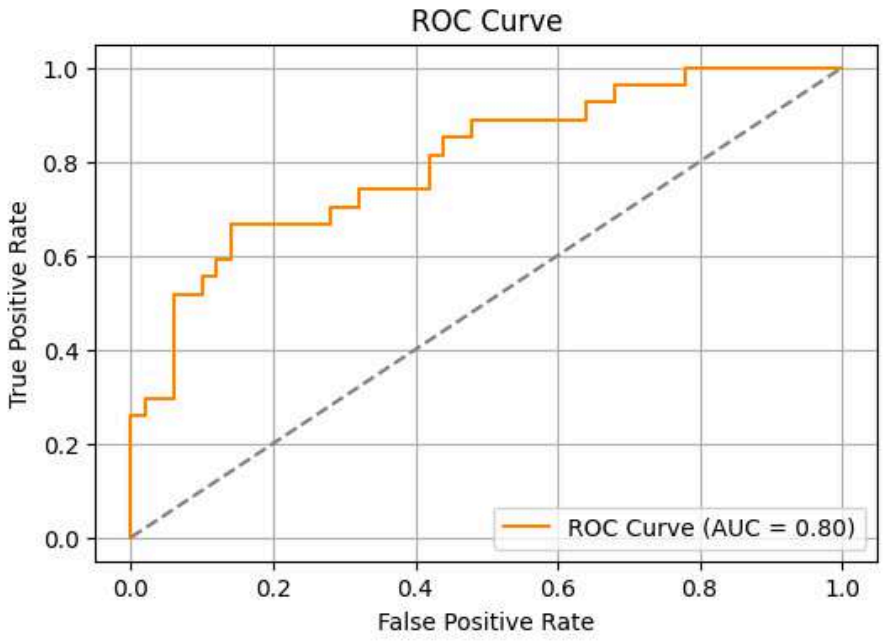
Model: **Support Vector Machine (SVM)**

Result:

```
Data loaded successfully.
Preprocessing complete.
Model training complete with best parameters.
Best Parameters: {'svm__C': 10, 'svm__gamma': 0.1, 'svm__kernel': 'rbf'}

Accuracy on Test Set: 0.7792

Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.88      0.84        50
           1       0.73      0.59      0.65        27

    accuracy                           0.78        77
   macro avg       0.76      0.74      0.75        77
weighted avg       0.77      0.78      0.77        77
```



The SVM model achieved an accuracy of **77.92%** on the test set, with better performance in detecting non-diabetic cases (class 0) than diabetic ones (class 1). Although precision and recall for class 1 are lower, the overall classification is reasonably balanced. With hyperparameter tuning, the model shows good potential for early diabetes prediction.



The ROC analysis yielded an **AUC of 0.80**, indicating that the model demonstrates good discriminative ability between diabetic and non-diabetic cases. This suggests that, on average, the classifier correctly distinguishes between the two classes 80% of the time. While the performance is acceptable for practical applications, further refinement in feature engineering or model tuning could be explored to enhance predictive accuracy.

Q.5] Train Regression Model and visualize the prediction performance of trained model

- **Data File: Regression data.csv**
- **Independent Variable: 1st Column**
- **Dependent variables: Column 2 to 5**

  Use any Regression model to predict the values of all Dependent variables using values of the 1st column.

  Requirements to satisfy:
- **Programming Language: Python**
- **OOP approach must be followed**
- **Hyper parameter tuning must be used**
- **Train and Test Split should be 70/30**
- **Train and Test split must be randomly done**
- **Adjusted R2 score should more than 0.99**
- **Use any Python library to present the accuracy measures of trained model**

https://github.com/Sutanoy/Public-Regression-Datasets

https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv

Name: Shivam.R.Prajapati **Div: D15C** **Roll No: 41**

URL:
https://archive.ics.uci.edu/ml/machine-learning-databases/00477/Real%20estate%20valuation%20data%20set.xlsx

(Refer any one)

Soln:

Dataset: Dry_Bean_Dataset

Dataset features:

- **Area**: Total pixel count inside the bean region.
- **Perimeter**: Distance around the bean boundary.
- **MajorAxisLength**: Length of the longest axis of the bean.
- **MinorAxisLength**: Length of the shortest axis of the bean.
- **AspectRatio**: Ratio of major to minor axis.
- **Eccentricity**: How elongated the bean is.
- **ConvexArea**: Number of pixels in the convex hull of the bean.
- **EquivDiameter**: Diameter of a circle with the same area as the bean.
- **Extent**: Ratio of bean area to bounding box area.
- **Solidity**: Ratio of bean area to convex hull area.
- **roundness**: Circularity of the bean shape.

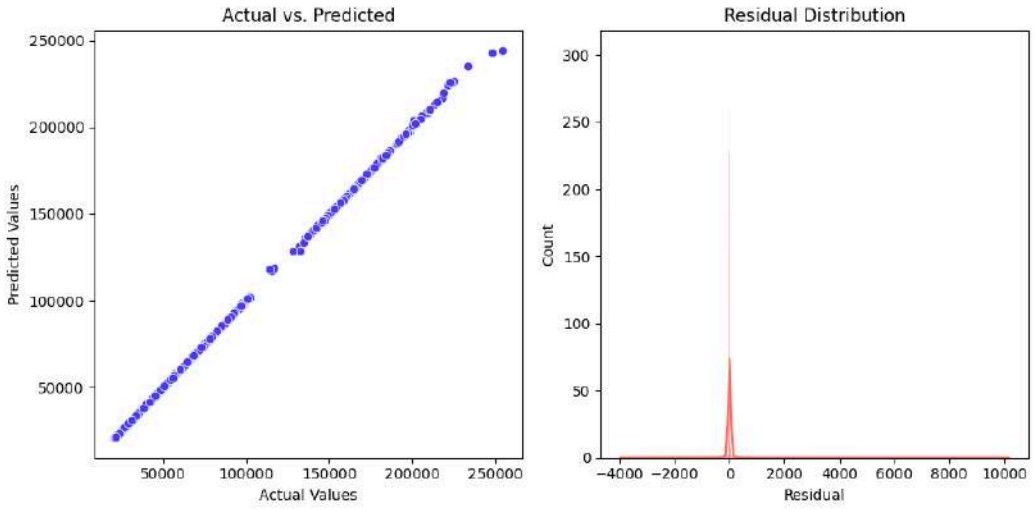
Compactness, ShapeFactor1, ShapeFactor2, ShapeFactor3, ShapeFactor4: Geometrical shape descriptors.

Model: **RandomForestRegressor**

Here we are predicting Area based on other features

Result:

```
Best parameters:  {'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
R2 Score: 0.9999
Adjusted R2 Score: 0.9999
✅ Model meets the required Adjusted R² score.
```



The regression model, tuned with optimal hyperparameters, achieved an **R^2 and Adjusted R^2 score of 0.9999**, indicating an **excellent fit** with the data. It successfully satisfies the requirement of an Adjusted R^2 score above **0.99**, demonstrating high predictive accuracy and generalization capability for the dry bean dataset.

Q.6] What are the key features of the wine quality data set? Discuss the importance of each feature in predicting the quality of wine? How did you handle missing data in the wine quality data set during the feature engineering process? Discuss the advantages and disadvantages of different imputation techniques. (Refer dataset from Kaggle).

Soln:

Main Features and Their Importance:

1. **Fixed Acidity:** Represents non-volatile acids (tartaric, malic, citric) that remain relatively stable. Higher fixed acidity can contribute to tartness and affect overall balance.
2. **Volatile Acidity:** Primarily acetic acid content, which at high levels can give an unpleasant vinegar taste. Low volatile acidity is generally preferred for higher quality wines.
3. **Citric Acid:** Can add freshness and flavor to wines. It contributes to the wine's citrus character and can enhance perceived freshness.
4. **Residual Sugar:** Affects the sweetness of the wine. Different wine styles have different optimal levels, making this feature important but contextual.
5. **Chlorides:** Salt content in wine, which can influence taste perception. Excessive chlorides can negatively impact quality.
6. **Free Sulfur Dioxide:** Acts as a preservative and antioxidant. Appropriate levels help preserve wine quality while excessive amounts can create unpleasant aromas.
7. **Total Sulfur Dioxide:** Sum of free and bound forms of SO_2. Important for preservation but can be detrimental to flavor when too high.

8. **Density:** Related to alcohol and sugar content. Provides information about the wine's body and can indicate fermentation completeness.
9. **pH:** Affects chemical stability and microbial control. Wines with balanced pH tend to be more stable and often higher quality.
10. **Sulphates:** Additives that contribute to SO_2 levels and act as preservatives. Moderate levels help wine stability.
11. **Alcohol:** Higher alcohol content is often associated with higher quality ratings, as it contributes to body and can enhance flavor perception.

Handling Missing Data in the Wine Quality Dataset:

Common Imputation Techniques and Their Trade-offs:

1. **Mean/Median/Mode Imputation**
   - **Advantages**: Simple, fast implementation; preserves the mean (when using mean imputation)
   - **Disadvantages**: Reduces variance; ignores relationships between features; can distort distributions
2. **K-Nearest Neighbors (KNN) Imputation**
   - **Advantages**: Accounts for relationships between features; better preserves data structure
   - **Disadvantages**: Computationally expensive for large datasets; sensitive to outliers; requires parameter tuning
3. **Regression Imputation**
   - **Advantages**: Maintains relationships between variables; can be more accurate than simpler methods
   - **Disadvantages**: May overfit; assumes linear relationships; can reduce variance
4. **Multiple Imputation**
   - **Advantages**: Accounts for uncertainty in missing values; maintains variability in the dataset
   - **Disadvantages**: Computationally intensive; more complex to implement
5. **Machine Learning Based Imputation** (Random Forest, etc.)
   - **Advantages**: Can capture complex non-linear relationships; often provides more accurate imputations
   - **Disadvantages**: Computationally expensive; risk of overfitting; requires careful validation