Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Experiment 1

Aim: Introduction to Data science and Data preparation using Pandas steps. Solve the following questions:-

- Load data in Pandas.
- Description of the dataset.
- Drop columns that aren't useful.
- Drop rows with maximum missing values.
- Take care of missing data.
- Create dummy variables.
- Find out outliers (manually)
- Standardization and Normalization of columns

Steps:

- Load data in Pandas:-

Step 1: Run the following commands:

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

These commands import essential libraries for data manipulation (pandas), numerical computations (numpy), and data visualization using matplotlib for basic plotting and seaborn for enhanced statistical graphics.

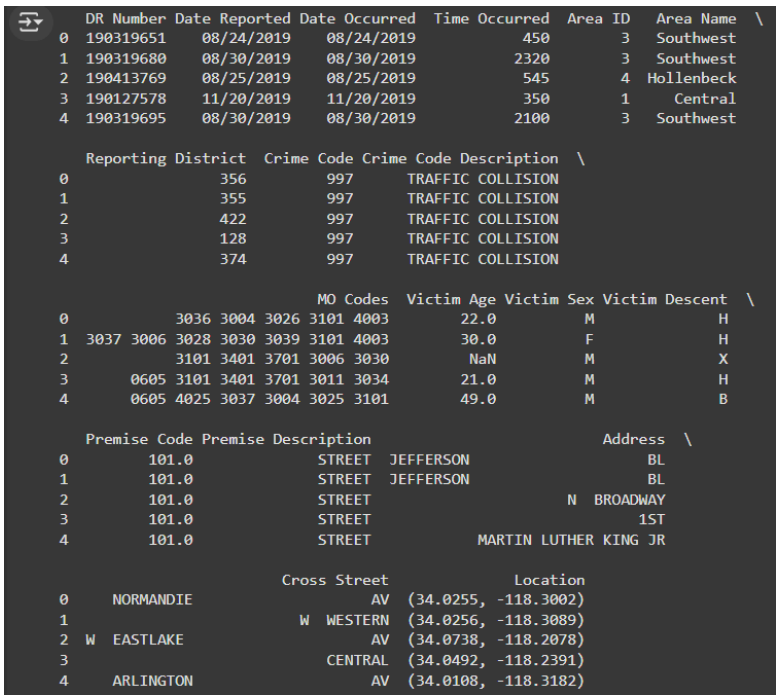
Step 2: Run the following command:

```
df = pd.read_csv('Traffic_Collision_Data_from_2010_to_Present.csv')
```

This command reads the Traffic_Collision_Data_from_2010_to_Present.csv file into a pandas DataFrame (df), allowing for structured data manipulation and analysis.

- Description of the dataset:-

Command 1: print(df.head())
This command prints the first five rows of the DataFrame df, giving a quick preview of the dataset's structure.

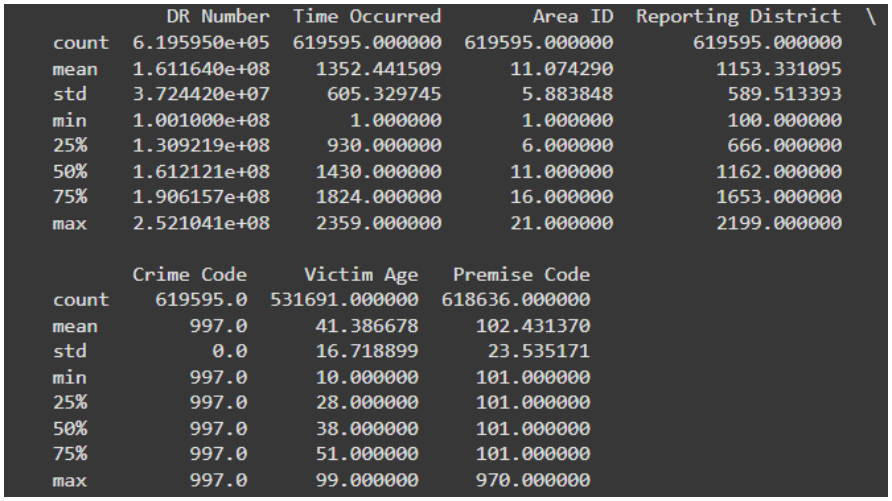


```
   DR Number Date Reported Date Occurred  Time Occurred  Area ID   Area Name  \
0  190319651    08/24/2019    08/24/2019            450        3   Southwest
1  190319680    08/30/2019    08/30/2019           2320        3   Southwest
2  190413769    08/25/2019    08/25/2019            545        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

                     Cross Street             Location
0    NORMANDIE                 AV  (34.0255, -118.3002)
1                    W  WESTERN  (34.0256, -118.3089)
2  W  EASTLAKE                 AV  (34.0738, -118.2078)
3                       CENTRAL  (34.0492, -118.2391)
4    ARLINGTON                 AV  (34.0108, -118.3182)
```

Command 2: print(df.info())
This command displays a summary of the DataFrame df, including the number of rows and columns, data types, and non-null value counts for each column.

```
<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
```

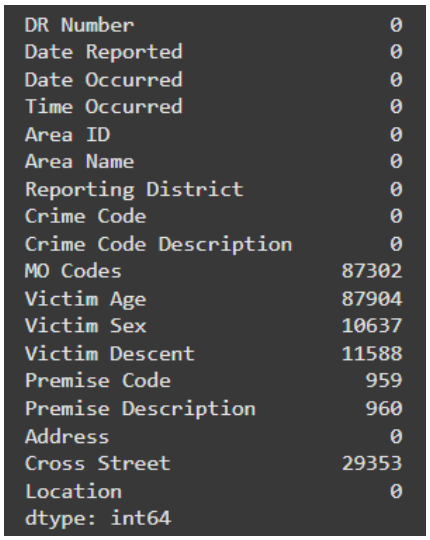
Command 3: print(df.describe())
This command generates summary statistics for numerical columns in the DataFrame df, including count, mean, standard deviation, minimum, maximum, and quartile values.



	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

Command 4: print(df.isnull().sum())
This command prints the total number of missing (null) values in each column of the DataFrame df, helping identify incomplete data.



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

- Drop columns that aren't useful:-

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

The above drops the columns ('Crime Code Description', 'MO Codes', 'Address', and 'Cross Street') from the DataFrame df, as they are deemed not useful for the analysis, and updates the DataFrame in place. The print(df.head()) command then displays the first five rows of the modified DataFrame.

	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)

- Drop rows with maximum missing values:-

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

The above calculates a threshold for the minimum number of non-null values required in a row (70% of the total columns) and drops rows in the DataFrame df that have fewer non-null values than this threshold, updating the DataFrame in place.

```
Dataset shape before dropping rows with maximum missing values: (619595, 14)
Dataset shape after dropping rows with maximum missing values: (619572, 14)
```

- Take care of missing data:-

Command: 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())
```

The above code fills missing values in the DataFrame df as follows:

- ☐ Numerical missing values in the 'Victim Age' column are filled with the median of that column.
- ☐ Categorical missing values in the specified columns ('Victim Sex', 'Victim Descent', 'Premise Description', 'Premise Code') are filled with the mode (most frequent value) of each respective column.

print(df.isnull().sum()) is used to verify that there are no more missing values in the DataFrame.

```
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
```

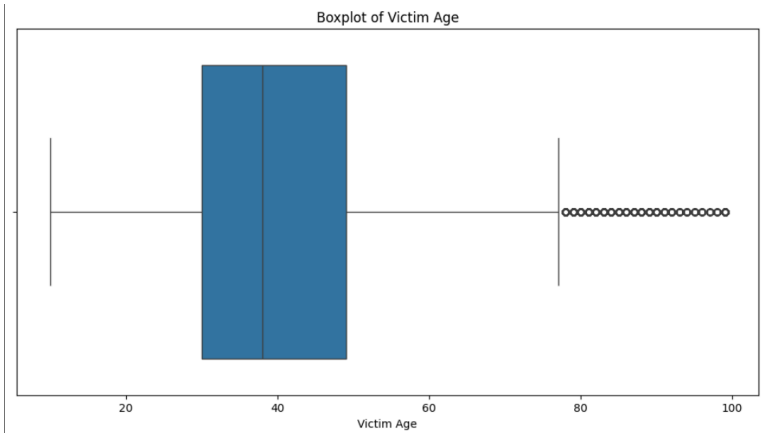
- Create dummy variables:-

Command: df = pd.get_dummies(df, columns=['Area Name', 'Victim Sex', 'Premise Description'], drop_first=True)
The above code changes the categorical columns ('Area Name', 'Victim Sex', and 'Premise Description') into separate columns with 0s and 1s to represent each category, and removes the first category in each to prevent confusion during analysis.

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

   Reporting District  Crime Code  Victim Age Victim Descent  Premise Code  \
0                 356         997        22.0              H         101.0
1                 355         997        30.0              H         101.0
2                 422         997        38.0              X         101.0
3                 128         997        21.0              H         101.0
4                 374         997        49.0              B         101.0

   ... Premise Description_TRAM/STREETCAR(BOXLIKE WAG ON RAILS)*  \
0  ...                                                 False
1  ...                                                 False
2  ...                                                 False
3  ...                                                 False
4  ...                                                 False

   Premise Description_TRANSPORTATION FACILITY (AIRPORT)  \
0                                              False
1                                              False
2                                              False
3                                              False
4                                              False
```

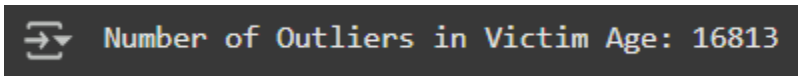
- Find out outliers (manually):-

Command 1: plt.figure(figsize=(12,6))
sns.boxplot(x=df['Victim Age'])
plt.title('Boxplot of Victim Age')
plt.show()
The first method uses a boxplot to visually identify outliers in the 'Victim Age' column by showing the distribution and extreme values.



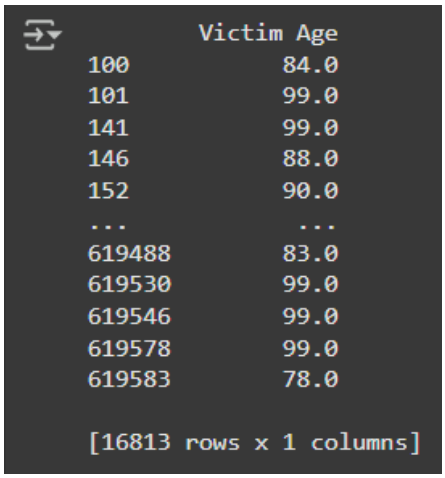
Command 2: Q1 = df['Victim Age'].quantile(0.25)
Q3 = df['Victim Age'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR
outliers = df[(df['Victim Age'] < lower_bound) | (df['Victim Age'] > upper_bound)]
print("Number of Outliers in Victim Age:", len(outliers))

The second method calculates the IQR (Interquartile Range), defines a range using 1.5 times the IQR, and identifies outliers numerically by checking values outside this range.



Command 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']])

Instead of just counting the outliers, we can also list the actual outlier values from the 'Victim Age' column as seen below.

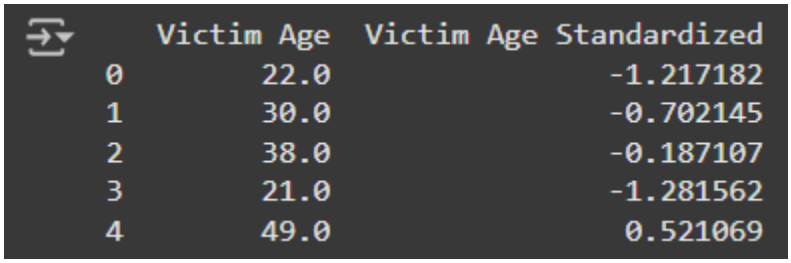


	Victim Age
100	84.0
101	99.0
141	99.0
146	88.0
152	90.0
...	...
619488	83.0
619530	99.0
619546	99.0
619578	99.0
619583	78.0

[16813 rows x 1 columns]

- Standardization of columns:-

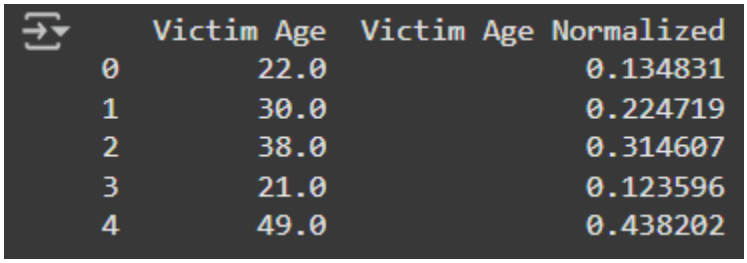
Command: from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['Victim Age Standardized'] = scaler.fit_transform(df[['Victim Age']])
The above code applies standardization to the 'Victim Age' column using StandardScaler from scikit-learn, transforming the data so that it has a mean of 0 and a standard deviation of 1, and stores the standardized values in a new column 'Victim Age Standardized' for better comparability and normalization.



	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

- Normalization of columns:-

Command: from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df['Victim Age Normalized'] = scaler.fit_transform(df[['Victim Age']])
The above code applies normalization, a technique that rescales the values of the 'Victim Age' column to a range between 0 and 1, using MinMaxScaler from scikit-learn, and stores the normalized values in a new column 'Victim Age Normalized' to ensure the data is on a consistent scale for improved model performance.



	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

Conclusion:

1. In this experiment, we learned how to preprocess data using Pandas steps.
2. By loading data into Pandas, we successfully imported the dataset into a Pandas DataFrame for analysis.
3. By describing the dataset, we understood the structure and statistics of the data using .info() and .describe().

4. By dropping unnecessary columns, we removed columns that were not relevant to the analysis to reduce complexity.
5. By dropping rows with maximum missing values, we eliminated rows where a high percentage of data was missing to improve data quality.
6. By handling missing data, we filled missing values appropriately to ensure data consistency.
7. By creating dummy variables, we converted categorical columns ('Area Name', 'Victim Sex', and 'Premise Description') into numerical representations using one-hot encoding.
8. By detecting outliers, we identified outliers using a Boxplot and Interquartile Range (IQR). We also displayed the outliers.
9. By standardizing and normalizing a numerical column, we ensured uniform data distribution by applying feature scaling techniques (standardization and normalization).
10. After preprocessing the data (removing rows with maximum missing values, removing outliers, etc), the number of rows in the dataset reduced from 619,595 to 602,702 (decrease of 2.72%).

Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Experiment 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.

Performance:

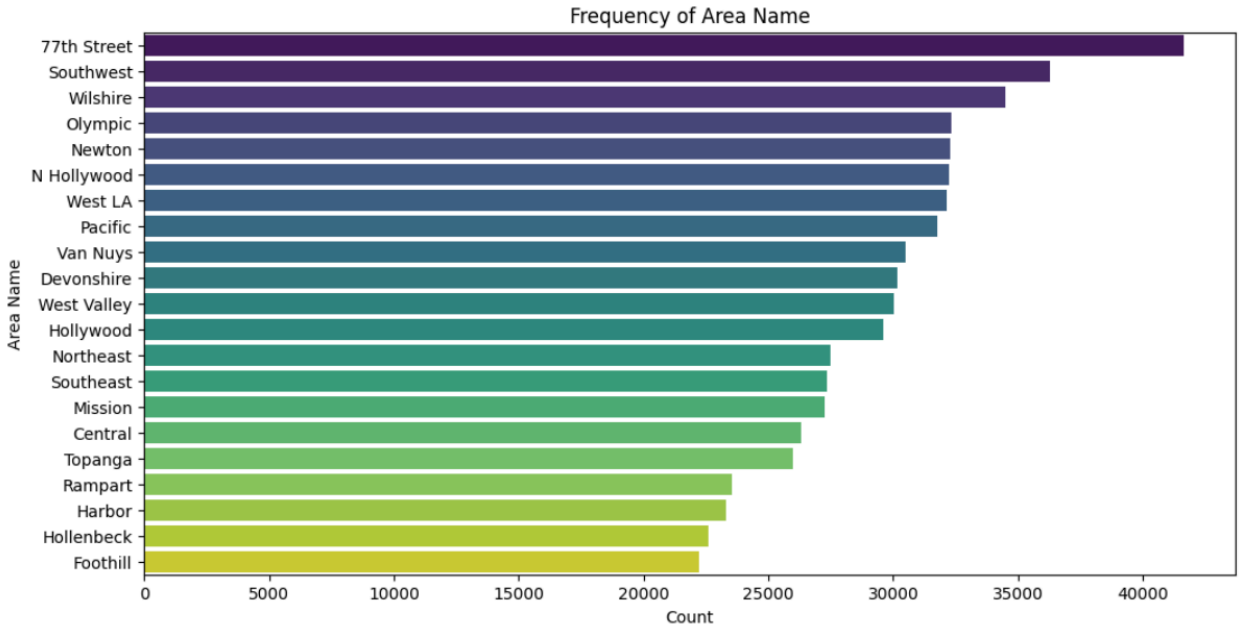
- Prerequisite: Import all the required libraries (pandas for data manipulation, numpy for numerical computations, and data visualization using matplotlib for basic plotting and seaborn for enhanced statistical graphics) and load data into Pandas:

Command: import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
df = pd.read_csv('Traffic_Collision_Data_from_2010_to_Present.csv')
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	49.0	M	B	101.0	STREET	MARTIN LUTHER KING JR	ARLINGTON AV	(34.0108, -118.3182)

- Create bar graph, contingency table using any 2 features:

Command: feature_x = "Area Name"
feature_y = "Crime Code Description"
plt.figure(figsize=(12, 6))
sns.countplot(y=df[feature_x], order=df[feature_x].value_counts().index, palette="viridis")
plt.title(f"Frequency of {feature_x}")
plt.xlabel("Count")
plt.ylabel(feature_x)
plt.show()
contingency_table = pd.crosstab(df[feature_x], df[feature_y])
print("Contingency Table:")
print(contingency_table)



The above bar graph represents the frequency of vehicle collisions across different areas. The x-axis shows the number of collisions, while the y-axis lists the area names. 77th Street has the highest number of reported collisions, followed by Southwest and Wilshire, while Foothill has the lowest. This suggests that certain areas experience significantly more traffic collisions, which could indicate high traffic density, accident-prone roads, or other contributing factors.

Contingency Table:

Area Name	TRAFFIC COLLISION
77th Street	41631
Central	26309
Devonshire	30191
Foothill	22215
Harbor	23307
Hollenbeck	22594
Hollywood	29601
Mission	27235
N Hollywood	32259
Newton	32282
Northeast	27508
Olympic	32316
Pacific	31787
Rampart	23541
Southeast	27351
Southwest	36285
Topanga	25979
Van Nuys	30518
West LA	32129
West Valley	30047
Wilshire	34510

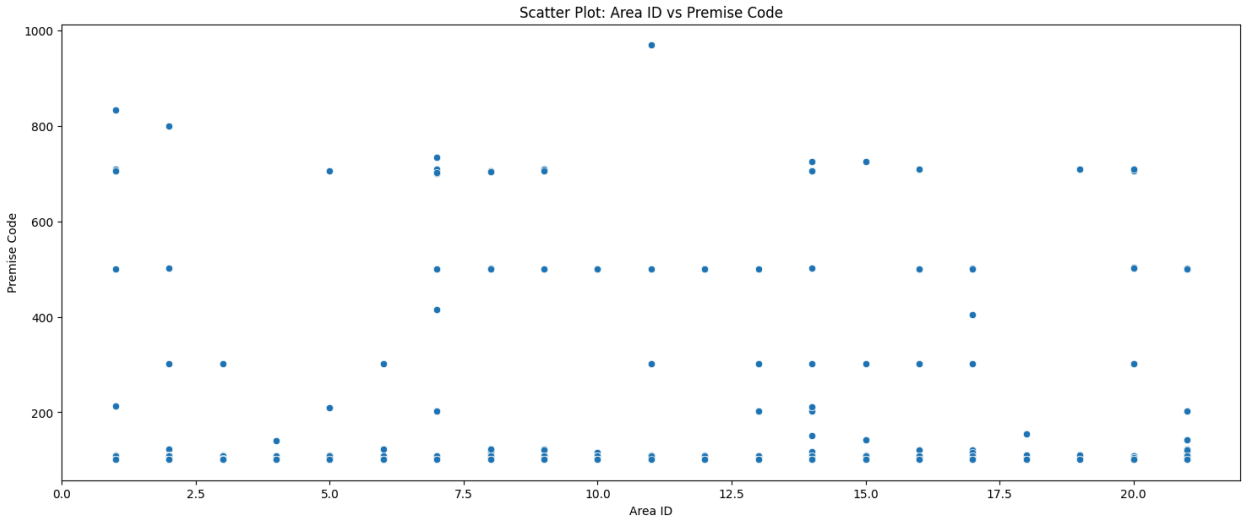
The contingency table displays the number of traffic collisions across different areas. Each row represents an area name, while the corresponding value indicates the number of reported traffic collisions in that area. 77th Street has the highest number of collisions, followed by Wilshire and Southwest, indicating higher traffic incidents in these regions. Conversely, areas like Foothill, Harbor, and Rampart have relatively fewer collisions. This distribution suggests variations in traffic density, road conditions, or reporting frequency across different areas.

- Plot Scatter plot, box plot, Heatmap using seaborn:

1. Scatter plot:-

Command:

```
plt.figure(figsize=(18, 7))
sns.scatterplot(x=df["Area ID"], y=df["Premise Code"])
plt.title("Scatter Plot: Area ID vs Premise Code")
plt.xlabel("Area ID")
plt.ylabel("Premise Code")
plt.show()
```

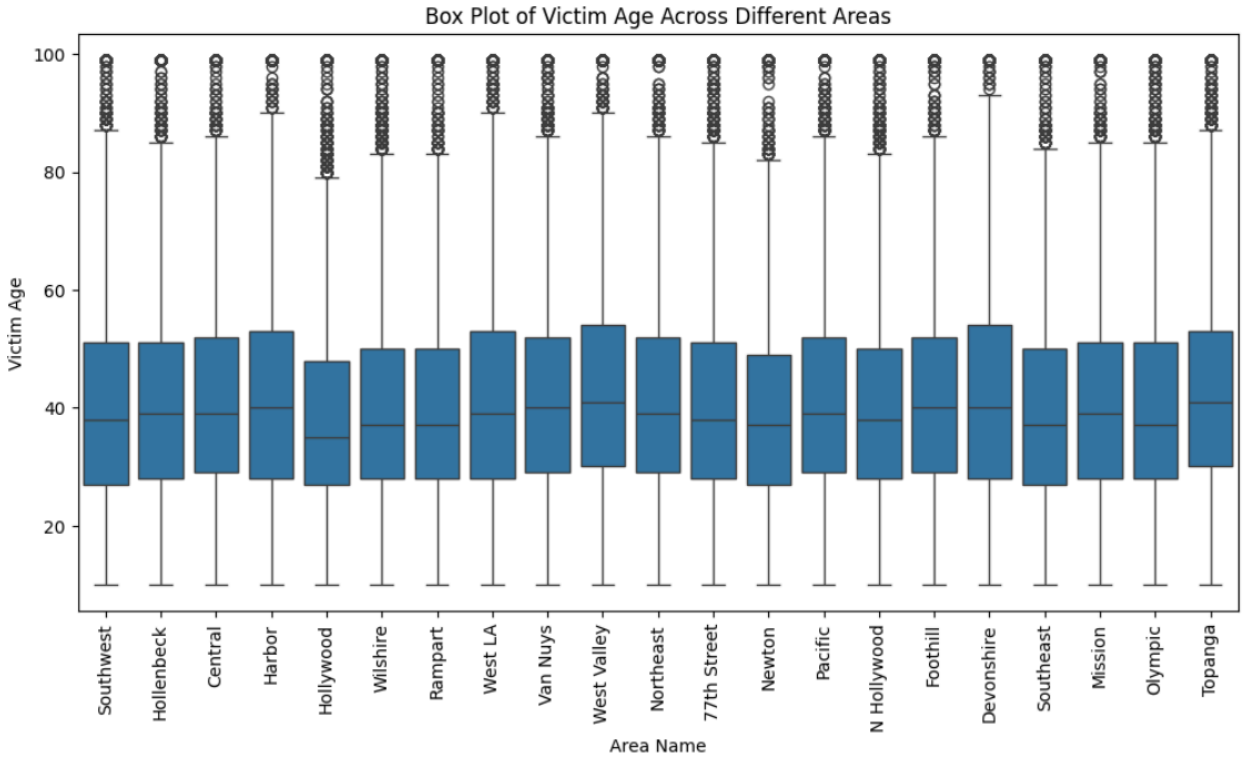


The scatter plot visualizes the relationship between Area ID and Premise Code in vehicle collision data. The x-axis represents different area IDs, while the y-axis represents premise codes, which categorizes the type of location where the collision occurred. The scattered points suggest that collisions happen across various premises in all areas, with some areas showing higher concentrations at specific premise codes. There are a few outliers, indicating locations where collisions are significantly more or less frequent.

2. Box Plot:-

Command:

```
plt.figure(figsize=(12, 6))
sns.boxplot(x=df["Area Name"], y=df["Victim Age"])
plt.xticks(rotation=90)
plt.title("Box Plot of Victim Age Across Different Areas")
plt.xlabel("Area Name")
plt.ylabel("Victim Age")
plt.show()
```

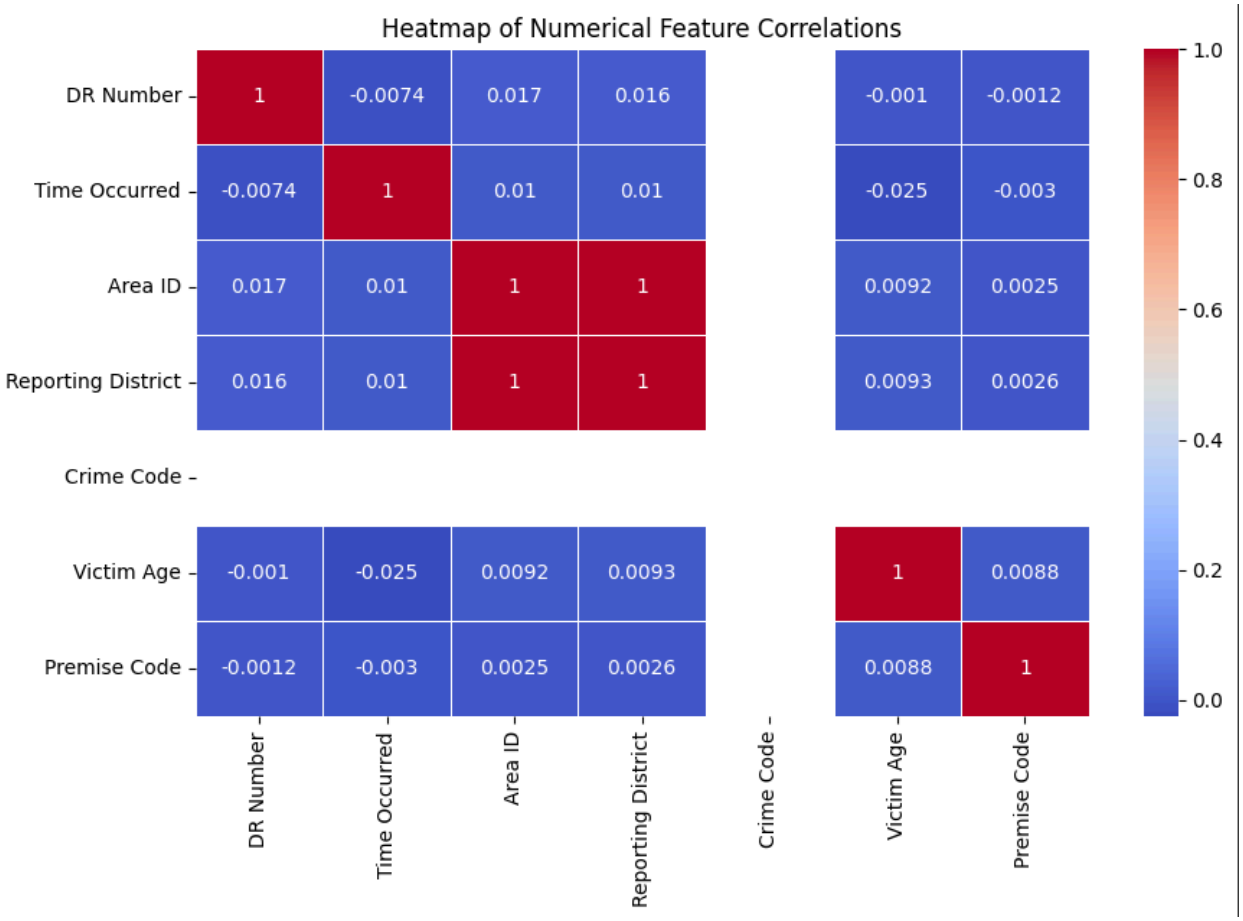


The box plot visualizes the distribution of victim ages across different areas, highlighting variations in age demographics. The median victim age appears to be around 35-45 years in most areas, with interquartile ranges spanning from approximately 25 to 55 years. The whiskers extend towards younger and older victims, with numerous outliers above 80 years, indicating some elderly victims involved in incidents. The overall distribution remains fairly consistent across areas, suggesting similar age patterns in reported cases regardless of location.

3. Heatmap:

Command:

```
plt.figure(figsize=(10, 6))
sns.heatmap(df.select_dtypes(include=np.number).corr(), annot=True,
cmap="coolwarm", linewidths=0.5)
plt.title("Heatmap of Numerical Feature Correlations")
plt.show()
```



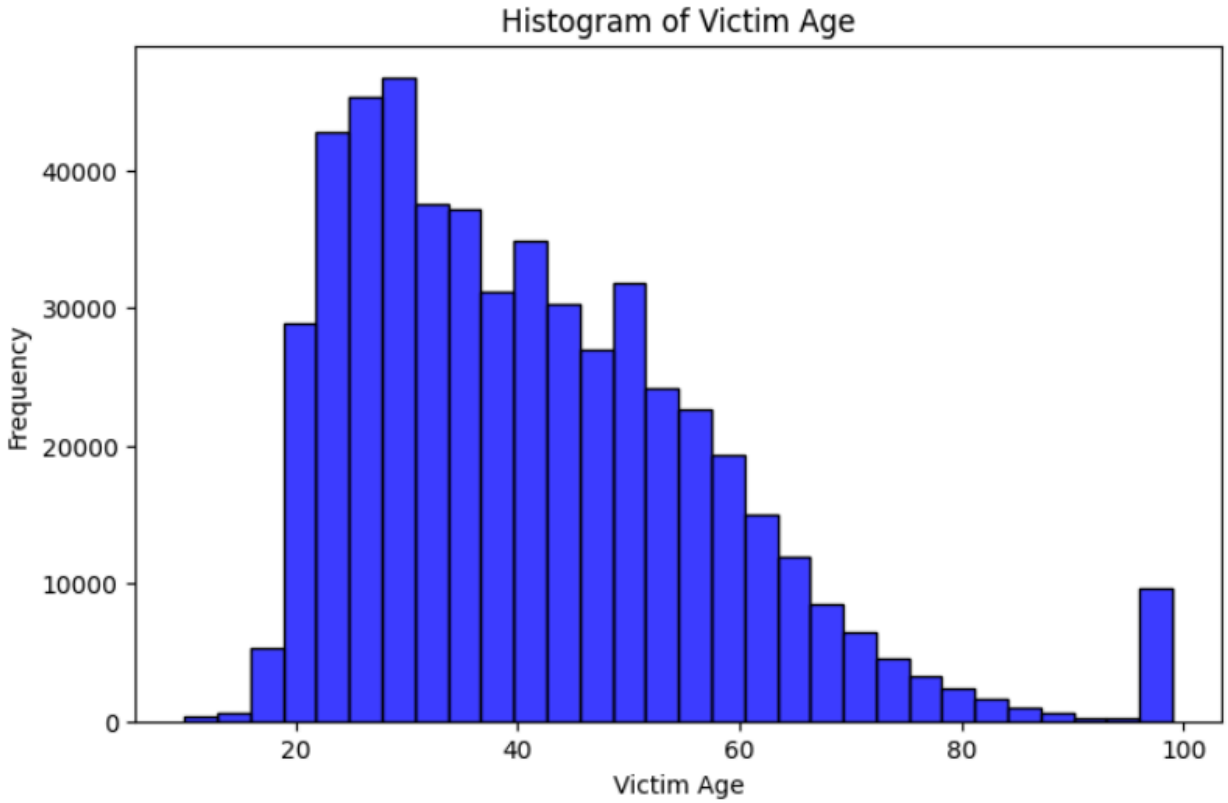
The heatmap visualizes the correlation matrix of numerical features, where values range from -1 to 1. A strong correlation (value = 1) is observed between Area ID and Reporting District, indicating they are closely related. Most other features exhibit weak or near-zero correlations, suggesting minimal linear relationships. Victim Age shows little correlation with Time Occurred and Premise Code, implying that age does not significantly influence when or where incidents occur. Similarly, DR Number and Crime Code have no meaningful correlation with other variables, indicating they function as independent identifiers. Overall, the heatmap suggests that most numerical features are weakly correlated, except for geographical identifiers, which show a strong relationship.

- Create histogram and normalized Histogram:-

1. Histogram:

Command:

```
plt.figure(figsize=(8, 5))
sns.histplot(df["Victim Age"], bins=30, kde=False, color="blue")
plt.title("Histogram of Victim Age")
plt.xlabel("Victim Age")
plt.ylabel("Frequency")
plt.show()
```

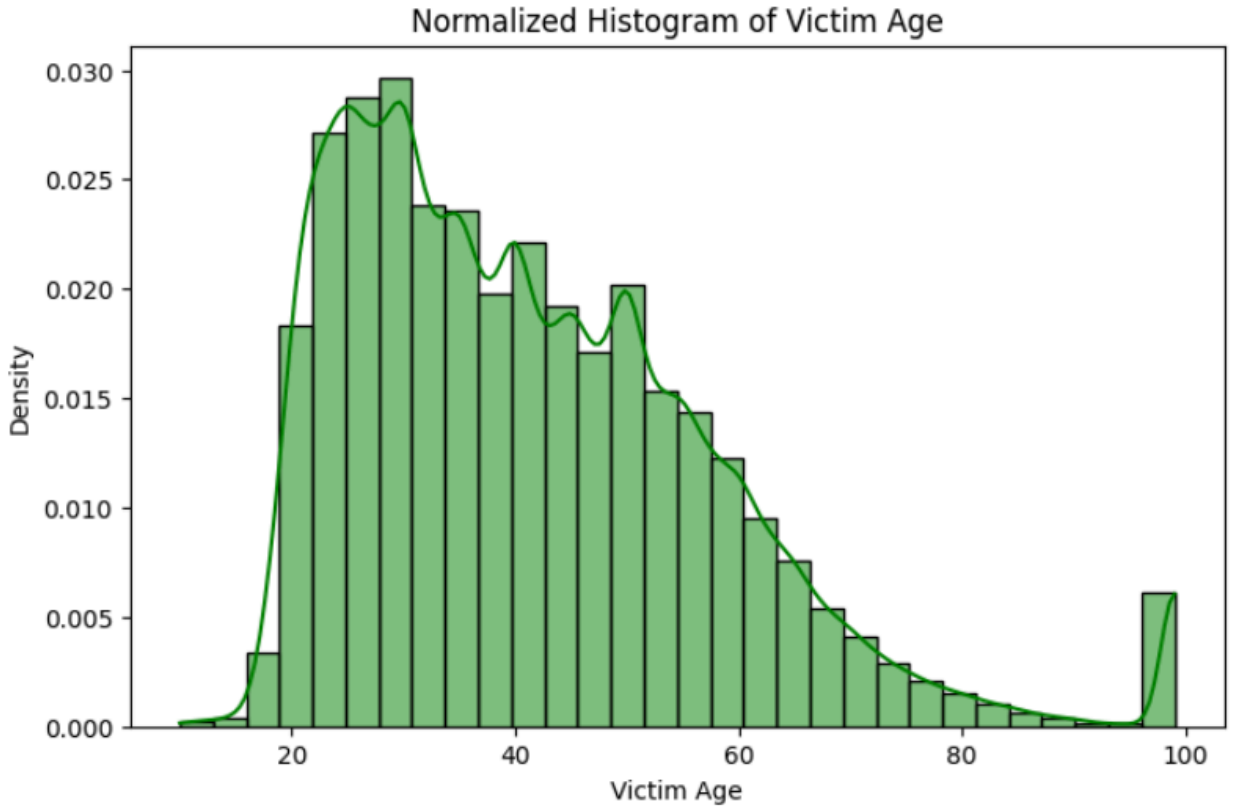


The histogram displays the distribution of Victim Age, showing a right-skewed pattern where most victims fall within the 20 to 40-year-old range, with the highest frequency occurring around the mid-20s to early 30s. The frequency gradually declines as age increases, with fewer incidents reported among victims over 60 years old. A notable spike is observed at 100 years, which may indicate data anomalies or specific reporting issues. Overall, the distribution suggests that younger adults are more frequently involved in incidents, while elderly victims are significantly less common.

2. Normalized Histogram:

Command:

```
plt.figure(figsize=(8, 5))
sns.histplot(df["Victim Age"], bins=30, kde=True, color="green", stat="density")
plt.title("Normalized Histogram of Victim Age")
plt.xlabel("Victim Age")
plt.ylabel("Density")
plt.show()
```



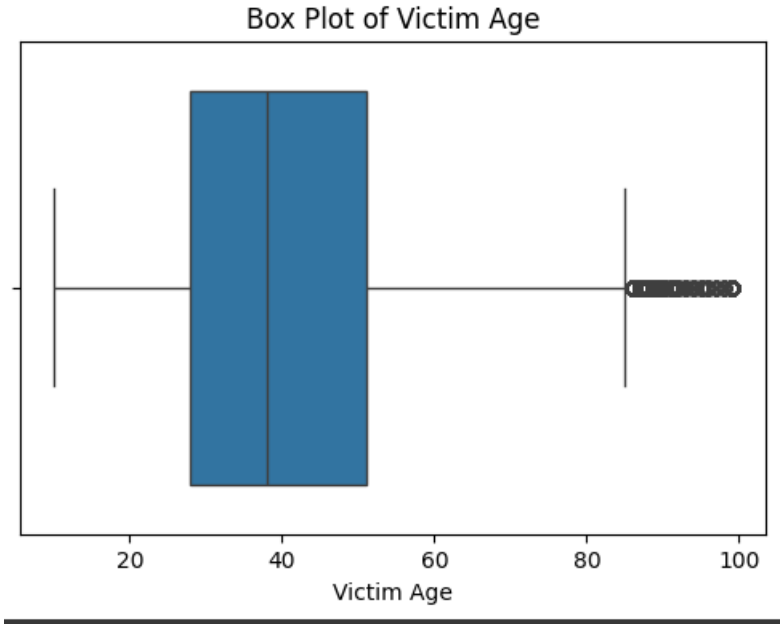
The normalized histogram of Victim Age represents the relative density of victims across different age groups rather than absolute counts, making it easier to compare distributions. The density peaks around the mid-20s to early 30s, indicating that this age range has the highest proportion of victims. As age increases, the density gradually declines, showing fewer cases among older individuals. The kernel density estimate (KDE) curve provides a smooth approximation of the distribution, highlighting fluctuations and reinforcing the overall right-skewed pattern. A small but noticeable spike at 100 years suggests an anomaly or special case in the data.

- Handle outlier using box plot and Inter quartile range:

1. Using box plot:-

Command:

```
plt.figure(figsize=(6, 4))
sns.boxplot(x=df["Victim Age"])
plt.title("Box Plot of Victim Age")
plt.show()
```



The box plot of Victim Age provides a summary of the age distribution, highlighting key statistical measures such as the median, interquartile range (IQR), and outliers. The median age is around the mid-40s, with the interquartile range (IQR) spanning from approximately mid-20s to mid-60s, indicating that the majority of victims fall within this age range. The whiskers extend to the minimum and maximum values within 1.5 times the IQR, while outliers, represented as individual points beyond the whiskers, appear around age 100, suggesting extreme values or possible data anomalies.

To remove outliers using a box plot, one common approach is to filter out values beyond 1.5 times the IQR from both the lower and upper quartiles. This helps in reducing the influence of extreme values on analysis, ensuring a more robust representation of the central data distribution.

2. Using Interquartile range:-

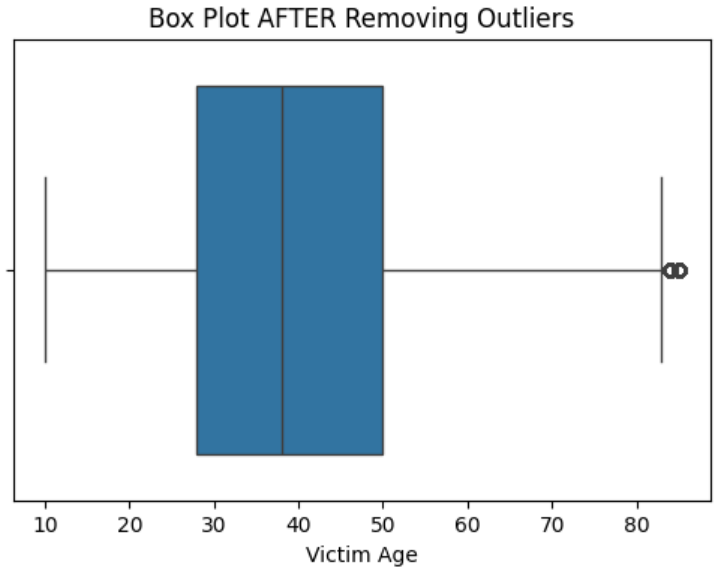
Command:

```
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 in Victim Age Column:\n", outliers)
df_cleaned = df[(df["Victim Age"] >= lower_bound) & (df["Victim Age"] <= upper_bound)]
print(f"Original dataset size: {df.shape[0]} rows")
print(f"Dataset size after removing outliers: {df_cleaned.shape[0]} rows")
```

```
Outliers in Victim Age Column:
          DR Number Date Reported Date Occurred  Time Occurred  Area ID  \
101     190814470    08/21/2019    08/21/2019           1220        8
141     190915755    08/24/2019    08/24/2019           1655        9
146     190916045    08/30/2019    08/30/2019             10        9
152     191008351    04/11/2019    04/11/2019            540       10
250     191418726    08/25/2019    08/19/2019           1230       14
...           ...           ...           ...            ...      ...
619427  240713209    12/12/2024    12/11/2024           1230        7
619432  241415646    12/07/2024    12/07/2024              5       14
619530  240613544    11/25/2024    11/24/2024           1600        6
619546  240812440    12/09/2024    12/09/2024            335        8
619578  241714453    11/24/2024    11/24/2024             45       17

[11396 rows x 18 columns]
Original dataset size: 619595 rows
Dataset size after removing outliers: 520295 rows
```

The Interquartile Range (IQR) is used to detect outliers by measuring the spread of the middle 50% of the data. First, the first quartile (Q1) and third quartile (Q3) are calculated, representing the 25th and 75th percentiles, respectively. The IQR is then determined as the difference between Q3 and Q1 (IQR = Q3 - Q1). To identify outliers, a lower bound is set at Q1 - 1.5 * IQR, and an upper bound is set at Q3 + 1.5 * IQR. Any values outside this range are considered outliers and can be removed from the dataset. The code first extracts these outliers and then filters the dataset to retain only values within the acceptable range, resulting in a cleaned dataset with reduced extreme values.

Box plot after removing outliers:



The box plot after removing outliers provides a clearer representation of the central distribution of victim ages without extreme values distorting the spread. Compared to the original box plot, the whiskers now extend only to the adjusted lower and upper bounds, ensuring that only values within the 1.5*IQR range are included. While a few mild outliers may still be present, the overall data distribution appears more compact and balanced. This refinement helps in more accurate analysis by reducing the influence of extreme values while preserving the essential characteristics of the dataset.

Conclusion:

1. In this experiment, we learned about Data Visualization / Exploratory Data Analysis using Matplotlib and Seaborn.
2. The bar graph showed that 77th Street and Wilshire have the highest number of collisions, likely due to high traffic density, accident-prone roads, or urban congestion.
3. The scatter plot indicated that some areas experience more incidents at specific premises, with a few outliers suggesting unusual collision patterns in certain locations.
4. The contingency table confirmed that certain areas, such as 77th Street and Wilshire, report significantly more collisions, reinforcing the findings from the bar graph analysis.
5. The box plot revealed that most victims are between 25-55 years old, while outliers above 80 suggest that elderly individuals are occasionally involved in incidents.
6. The heatmap showed that most numerical features have weak correlations, except for strong relationships between geographical identifiers like Area ID and Reporting District.
7. The histogram displayed a right-skewed victim age distribution, with the highest frequency in the mid-20s to early 30s, gradually declining for older age groups.

8. Outlier removal using the IQR method helped refine data accuracy by eliminating extreme victim ages that could distort statistical analysis and overall conclusions.

Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Experiment 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.

Performance:

- Prerequisite: Import all the required libraries: pandas for data manipulation and analysis, numpy for numerical computations, matplotlib.pyplot for basic plotting and data visualization, and seaborn for enhanced statistical graphics. Additionally, use statsmodels' ztest for performing statistical hypothesis testing and sklearn's train_test_split for splitting data into training and testing sets. Also, load data into Pandas:

Command:

```
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()
```

	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

- 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:

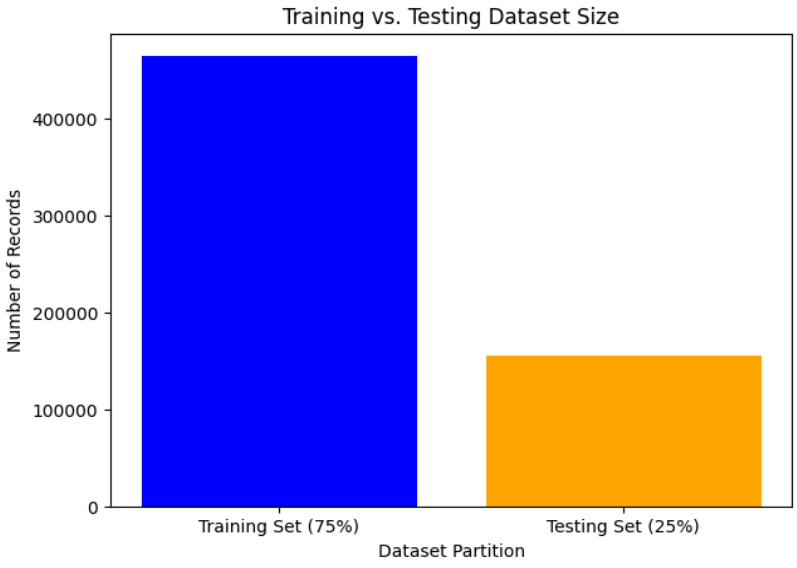
Command:
```
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)
print(f"Total records: {len(df)}")
print(f"Training set records: {len(train_df)}")
print(f"Testing set records: {len(test_df)}")
```
The above code splits the dataset into 75% training data and 25% testing data using train_test_split(), with random_state=42 ensuring the split remains consistent across runs, and then prints the total number of records along with the sizes of the training and testing sets.

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

- Use a bar graph and other relevant graphs to confirm your proportions:

1. Bar graph:-

Command:
```
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()
```
The above code generates a bar chart to visualize the dataset split, comparing the number of records in the training set (75%) and testing set (25%), helping to confirm the partitioning.

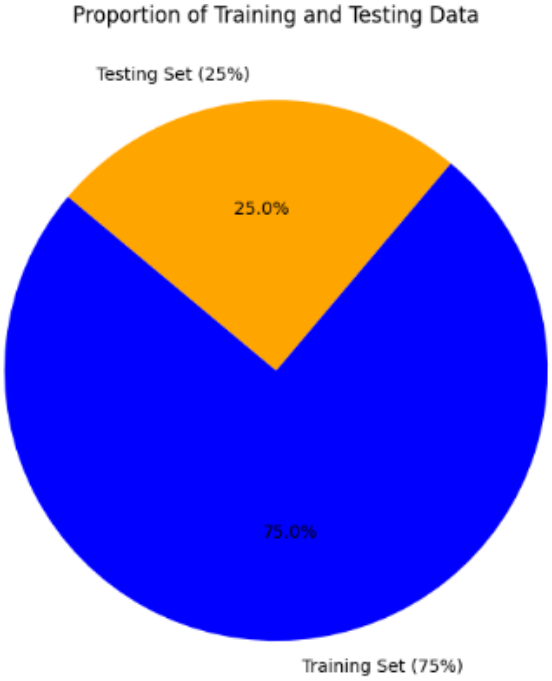


2. Pie Chart:-

Command:

```
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()
```

The above code generates a pie chart displaying the proportion of records in the training (75%) and testing (25%) datasets, confirming that the data was correctly partitioned.



- Identify the total number of records in the training data set:

Command:

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

The above code calculates and prints the total number of records in the training dataset by determining the length of train_df.

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

- Validate partition by performing a two-sample Z-test:

Command:

```
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)")
```

The above code performs a two-sample Z-test on the Victim Age column to compare the means of the training and testing datasets, printing the Z-statistic (which measures the difference between the sample means in terms of standard errors) and the p-value (which indicates the probability of observing such a difference by chance). If the p-value is greater than 0.05, the partitioning is considered statistically valid.

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

Conclusion:

1. In this experiment, we learned about Data Modeling.
2. The dataset was split into 75% training data (296,816 records) and 25% testing data (98,939 records) using train_test_split().
3. A bar chart and pie chart confirmed the correct proportions, showing 75% training data and 25% testing data visually.
4. The total number of records in the training dataset was 296,816, which was printed for verification.
5. A two-sample Z-test on the Victim Age column was performed, yielding a Z-statistic of 1.32 and a p-value of 0.1865.
6. Since the p-value (0.1865) was greater than the significance level (0.05), the test confirmed no significant difference between the training and testing sets, validating the partitioning.

Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Experiment 4

Aim: Implementation of Statistical Hypothesis Test using Scipy and Sci-kit learn.
Perform the following correlation tests:
a) Pearson's Correlation Coefficient
b) Spearman's Rank Correlation
c) Kendall's Rank Correlation
d) Chi-Squared Test

Performance:

- Prerequisite: We import necessary libraries such as pandas for data manipulation, numpy for numerical operations, scipy.stats for statistical calculations, and seaborn and matplotlib.pyplot for visualization and load data into Pandas. To understand the dataset structure, we print its basic information using df.info() to check column types (numerical or categorical) and df.head() to preview the first few rows:

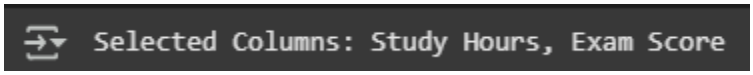
Command:
```
import pandas as pd
import numpy as np
import scipy.stats as stats
import seaborn as sns
import matplotlib.pyplot as plt
df = pd.read_csv('Student Performance Analysis.csv')
print("Dataset Info:")
df.info()
df.head()
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Study Hours           50 non-null     int64
 1   Exam Score            50 non-null     int64
 2   Stress Level          50 non-null     int64
 3   Sleep Hours           50 non-null     int64
 4   Break Time(In MIN)    50 non-null     int64
 5   Previous Exam Score   50 non-null     int64
 6   Practice Tests Taken  50 non-null     int64
 7   Motivation Level      50 non-null     int64
dtypes: int64(8)
memory usage: 3.3 KB
```

	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

- To perform correlation tests on specific features, we manually select two columns (col1 and col2), which should be numerical for Pearson, Spearman, and Kendall correlation. It is then checked whether the selected column names exist in the dataset to prevent errors. This step ensures we are working with the correct variables for our analysis.

```
Command: col1 = 'Study Hours'
col2 = 'Exam Score'
if col1 not in df.columns or col2 not in df.columns:
    raise ValueError("One or both selected columns do not exist in the dataset!")
print(f"Selected Columns: {col1}, {col2}")
```



a) Pearson's Correlation Coefficient:

```
Command: pearson_corr, _ = stats.pearsonr(df[col1], df[col2])
print(f"Pearson Correlation Coefficient between {col1} and {col2}: {pearson_corr:.4f}")
```

Pearson's correlation measures the linear relationship between two continuous variables. We compute it using 'stats.pearsonr(df[col1], df[col2])', which returns a correlation coefficient ranging from -1 (perfect negative correlation) to +1 (perfect positive correlation), with 0 indicating no correlation.
The Pearson correlation of 0.9648 indicates a strong positive linear relationship between Study Hours and Exam Score. This means that as students study more hours, their exam scores tend to increase proportionally, showing a near-perfect linear trend.

b) Spearman's Rank Correlation:

```
Command: spearman_corr, _ = stats.spearmanr(df[col1], df[col2])
print(f"Spearman's Rank Correlation between {col1} and {col2}: {spearman_corr:.4f}")
```

Spearman's correlation measures the monotonic relationship between two variables, making it more robust to outliers and non-linear relationships than Pearson's. It is calculated using stats.spearmanr(df[col1], df[col2]), which ranks the values before computing the correlation. Like Pearson's, the coefficient ranges from -1 to +1, where higher absolute values indicate stronger relationships. This test is useful when data is not normally distributed.
With a Spearman correlation of 0.9671, there is a strong monotonic relationship between Study Hours and Exam Score. Even if the relationship is not strictly linear, the ranking of students based on study hours aligns closely with their exam performance.

c) Kendall’s Rank Correlation:

Command: kendall_corr, _ = stats.kendalltau(df[col1], df[col2])
print(f"Kendall's Rank Correlation between {col1} and {col2}: {kendall_corr:.4f}")

```
Kendall's Rank Correlation between Study Hours and Exam Score: 0.8861
```

Kendall’s correlation is another non-parametric test that assesses the strength of association between two variables based on the concordance of pairs. It is calculated using stats.kendalltau(df[col1], df[col2]), which measures the agreement between ranked data points. Kendall’s method is particularly useful for smaller datasets and when working with ordinal data. The correlation coefficient, like the previous tests, ranges between -1 and +1, with values closer to ±1 indicating stronger relationships.
The Kendall correlation of 0.8861 suggests a very strong agreement in ranking between Study Hours and Exam Score. Students who study more consistently rank higher in exam scores, reinforcing the predictability of performance based on study time.

d) Chi-Squared Test:

Command:contingency_table = pd.crosstab(df[col1], df[col2])
chi2_stat, p_val, dof, expected = stats.chi2_contingency(contingency_table)
print(f"Chi-Squared Test Statistic: {chi2_stat}")
print(f"Degrees of Freedom: {dof}")
print(f"p-value: {p_val}")
if p_val < 0.05:
    print(f"There is a significant association between {col1} and {col2}.")
else:
    print(f"There is NO significant association between {col1} and {col2}.")

```
Chi-Squared Test Statistic: 411.89814814814815
Degrees of Freedom: 350
p-value: 0.012573183158872888
There is a significant association between Study Hours and Exam Score.
```

The Chi-Squared test is used to examine the association between two categorical variables by creating a contingency table using pd.crosstab(df[col1], df[col2]). The test statistic, degrees of freedom, and p-value are computed using stats.chi2_contingency(contingency_table). If the p-value is less than 0.05, we reject the null hypothesis, indicating a significant association between the two variables. Otherwise, there is no significant relationship. This test is particularly useful in analyzing dependencies between categorical-like numerical data, such as grouped scores or study hour categories
The Chi-Squared test result shows a statistically significant association ($p < 0.05$) between Study Hours and Exam Score. This means that the two variables are not independent, and study time likely influences exam performance.

Conclusion:

1. In this experiment, we learned about the implementation of Statistical Hypothesis Test using Scipy and Sci-kit learn.
2. The Pearson correlation coefficient (0.9648) indicates a strong positive linear relationship between Study Hours and Exam Score, meaning that as students study more, their exam scores tend to increase proportionally.
3. The Spearman's rank correlation (0.9671) suggests that the ranking of students based on study hours closely matches their ranking in exam scores, even if the relationship is not perfectly linear.
4. The Kendall's rank correlation (0.8861) confirms a strong agreement in ranking, showing that students who study more are highly likely to rank higher in exam performance.
5. The Chi-Squared test result ($\chi^2 = 411.90$, $p = 0.0126$) indicates a statistically significant association between Study Hours and Exam Score, proving that study time plays an important role in influencing exam performance.
6. Overall, all tests confirm a strong positive relationship between study hours and exam scores, suggesting that increasing study time is likely to improve exam performance

Experiment 5

Aim: Perform Regression Analysis using Scipy and Sci-kit learn.
a) Perform Logistic regression to find out relation between variables
b) Apply regression model technique to predict the data on the above dataset.

Performance:

- Prerequisite: Import essential libraries: pandas for data manipulation, numpy for numerical computations, matplotlib.pyplot and seaborn for data visualization, sklearn.model_selection for dataset splitting, sklearn.preprocessing for data scaling, sklearn.linear_model for logistic regression, and sklearn.metrics for model evaluation. Next, load the Electric Vehicle Population Dataset into a Pandas DataFrame using pd.read_csv(). Finally, explore the dataset by displaying the first few rows with df.head() and checking column names, data types, and missing values using df.info():

Command:
```
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())
```

```
   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

      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
```

a) Perform Logistic regression to find out relation between variables:

Step 1: Select Target Column ("Electric Vehicle Type"):-
Command: 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
})
```

```
array(['Battery Electric Vehicle (BEV)',
       'Plug-in Hybrid Electric Vehicle (PHEV)'], dtype=object)
```

First, df['Electric Vehicle Type'].unique() retrieves and displays the unique values in the Electric Vehicle Type column, helping to identify the different categories present in the dataset. Then, a new binary column, EV_Type_Binary, is created by mapping Battery Electric Vehicles (BEV) to 0 and Plug-in Hybrid Electric Vehicles (PHEV) to 1 using the map() function. This transformation converts categorical data into a numerical format, making it suitable for machine learning models.

Step 2: Select Features (X) and Target (y):-
Command: df_selected = df[['Model Year', 'Electric Range', 'Base MSRP', 'Legislative District']]

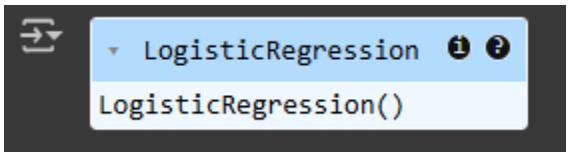
```
df_selected = df_selected.dropna()
X = df_selected
y = df.loc[df_selected.index, 'EV_Type_Binary']
```

This step selects specific numerical columns—Model Year, Electric Range, Base MSRP, and Legislative District—from the dataset and stores them in df_selected. It then removes any rows with missing values using dropna() to ensure the data is clean for modeling. The feature matrix X is assigned the cleaned df_selected, while the target variable y is extracted from the EV_Type_Binary column, ensuring that both X and y have matching indices. This prepares the dataset for training a machine learning model.

Step 3: Train-Test Split:-
Command: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
This step splits the dataset into training and testing sets using the train_test_split() function. The feature matrix X and target variable y are divided into X_train, X_test, y_train, and y_test, where 30% of the data is allocated for testing (test_size=0.3), and 70% for training. Setting random_state=42 ensures reproducibility by making the split consistent across different runs. This step is essential for evaluating the model's performance on unseen data.

Step 4: Normalize the Features:-
Command: scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
This step standardizes the feature values using StandardScaler. First, an instance of StandardScaler is created. Then, fit_transform(X_train) computes the mean and standard deviation from the training data and scales it, ensuring all features have a mean of 0 and a standard deviation of 1. The same transformation is applied to X_test using transform(X_test), maintaining consistency. Standardization improves model performance by preventing features with larger ranges from dominating those with smaller ones.

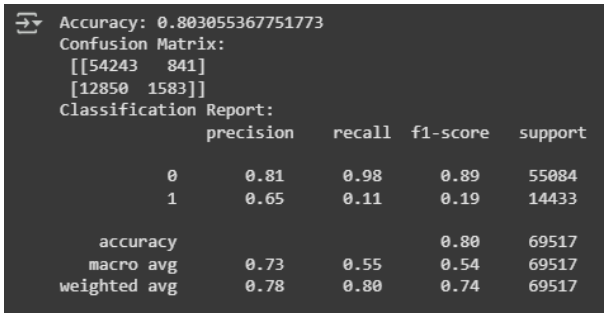
Step 5: Train Logistic Regression Model:-
Command: logreg = LogisticRegression()
logreg.fit(X_train_scaled, y_train)



This step initializes a Logistic Regression model using LogisticRegression(). The model is then trained on the standardized training data using fit(X_train_scaled, y_train), where it learns the relationship between the features and the target variable. This trained model can later be used to make predictions on new data.

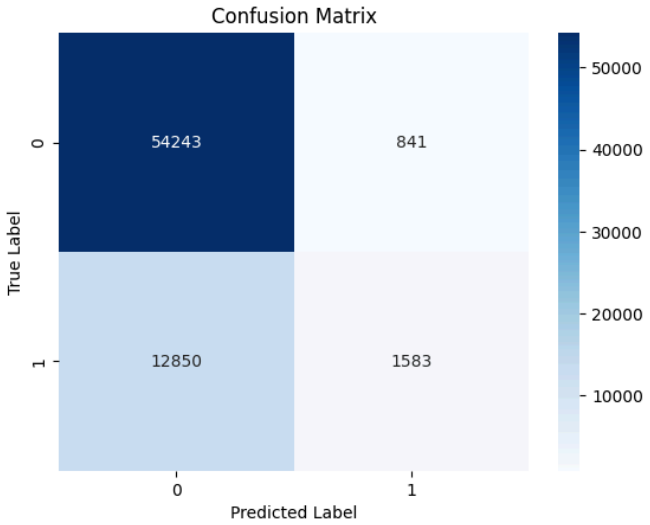
Step 6: Make Predictions:-
Command: y_pred = logreg.predict(X_test_scaled)
This step uses the trained Logistic Regression model to make predictions on the standardized test data. The predict(X_test_scaled) function generates predicted labels (y_pred) based on the learned relationships from the training phase. These predictions will be compared with actual values to evaluate the model's performance.

Step 7: Evaluate the Model:-
Command: 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))



This step evaluates the Logistic Regression model's performance by calculating key metrics. The accuracy score measures the percentage of correct predictions, while the confusion matrix displays the distribution of true positives, true negatives, false positives, and false negatives. Finally, the classification report provides detailed metrics such as precision, recall, and F1-score for each class, offering insights into the model's effectiveness.

Step 8: Visualize Confusion Matrix for better understanding:-
Command: 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()



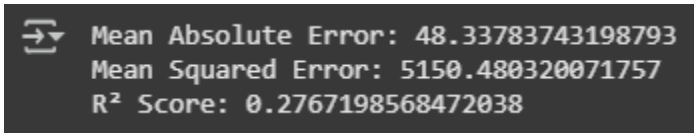
This step visualizes the confusion matrix using Seaborn's heatmap() function. The confusion matrix is plotted with numerical values (annot=True) in a blue color scheme (cmap='Blues'). Labels for the x-axis (Predicted Label) and y-axis (True Label) are added for clarity, along with a title (Confusion Matrix). Finally, plt.show() displays the heatmap, making it easier to interpret the model's classification performance.

b) Apply regression model technique to predict the data on the above dataset:

Step 1: Change Target Variable (y) to "Electric Range":-
Command: y_reg = df_selected['Electric Range']
X_reg = df_selected.drop(['Electric Range'], axis=1)
This step prepares the dataset for regression analysis by defining the target variable (y_reg) and feature matrix (X_reg). The Electric Range column is selected as the target variable (y_reg) since the goal is to predict it. The remaining columns in df_selected are assigned to X_reg by dropping Electric Range using drop(axis=1), ensuring that only independent variables are used as input features for the regression model.

Step 2: Train a Linear Regression Model:-
Command: 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)
This step applies Linear Regression to predict Electric Range using the selected features. The dataset is split into training (70%) and testing (30%) sets, ensuring reproducibility with random_state=42. The features are standardized using StandardScaler() to prevent numerical imbalances. A Linear Regression model is then initialized and trained on the scaled training data with fit(). Finally, predictions are made on the test set using predict(), generating y_pred_reg, which contains the predicted electric range values.

Step 3: Evaluate Regression Model:-
Command: 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))



This step evaluates the Linear Regression model's performance using three key metrics. Mean Absolute Error (MAE) measures the average absolute difference between actual and predicted values, while Mean Squared Error (MSE) penalizes larger errors more heavily. The R² Score indicates how well the model explains the variance in Electric Range, with values closer to 1 signifying better performance. These metrics provide insights into the model's accuracy and effectiveness.

Conclusion:

1. In this experiment, we learned to perform Regression Analysis using Scipy and Sci-kit learn.
2. The dataset was preprocessed by handling missing values, encoding categorical variables, and selecting relevant numerical features.
3. A Logistic Regression model was trained to classify electric vehicle types based on selected features.
4. The classification model achieved an accuracy of 80.3%, indicating a relatively strong ability to distinguish between Battery Electric Vehicles (BEVs) and Plug-in Hybrid Electric Vehicles (PHEVs).

5. The confusion matrix showed that the model correctly classified 54,243 BEVs but misclassified 841 as PHEVs, while it correctly identified only 1,583 PHEVs and misclassified 12,850 as BEVs.
6. The classification report revealed high precision (81%) and recall (98%) for BEVs, but low recall (11%) for PHEVs, indicating the model struggles to correctly identify PHEVs.
7. A Linear Regression model was applied to predict Electric Range using features like Model Year, Base MSRP, and Legislative District.
8. The regression model resulted in a Mean Absolute Error (MAE) of 48.34 and a Mean Squared Error (MSE) of 5150.48, indicating notable prediction variations.
9. The R^2 score of 0.277 suggests that the regression model explains only 27.7% of the variance in Electric Range, meaning other influential factors are missing from the dataset.
10. The classification model performed well overall but had difficulty identifying PHEVs, while the regression model had limited predictive power for Electric Range.

Experiment 6

Aim: Perform Classification Modeling:
a) Choose a classifier for a classification problem.
b) Evaluate the performance of the classifier.
Perform Classification using the following 3 classifiers:

1. K-Nearest Neighbors (KNN)
2. Naive Bayes
3. Decision Tree

Performance:

- Prerequisite: Import essential libraries: pandas for data manipulation, numpy for numerical computations, seaborn and matplotlib.pyplot for data visualization, and sklearn modules for dataset splitting, preprocessing, and classification using K-Nearest Neighbors, Naïve Bayes, SVM, and Decision Tree models. Load the Electric Vehicle Population Dataset into a Pandas DataFrame using pd.read_csv(). Finally, explore the dataset by displaying the first few rows with df.head() and checking column names, data types, and missing values using df.info():

Command:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
df = pd.read_csv('Electric_Vehicle_Population_Data.csv')
print(df.head())
print(df.info())
```

```
    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
```

```
   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'>
```

```
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)
```

Step 1: Data Preprocessing and Encoding:-

```
Command: df_filtered = df[["Model Year", "Make", "Model", "Electric Vehicle Type",
                "Clean Alternative Fuel Vehicle (CAFV) Eligibility",
                "Electric Range", "Base MSRP"]].dropna()
label_encoders = {}
for col in ["Make", "Model", "Clean Alternative Fuel Vehicle (CAFV) Eligibility"]:
    le = LabelEncoder()
    df_filtered[col] = le.fit_transform(df_filtered[col])
    label_encoders[col] = le
target_encoder = LabelEncoder()
df_filtered["Electric Vehicle Type"] = target_encoder.fit_transform(df_filtered["Electric Vehicle
Type"])
df_filtered.head()
```

	Model Year	Make	Model	Electric Vehicle Type	Clean Alternative Fuel Vehicle (CAFV) Eligibility	Electric Range	Base MSRP
0	2014	41	128	0	0	103.0	0.0
1	2019	39	97	0	0	220.0	0.0
2	2025	5	163	1	0	40.0	0.0
3	2024	41	129	1	0	42.0	0.0
4	2021	39	100	0	1	0.0	0.0

The code selects key columns for classification, including Model Year, Make, Model, Electric Vehicle Type, CAFV Eligibility, Electric Range, and Base MSRP, while removing rows with missing values using dropna(). Categorical variables (Make, Model, CAFV Eligibility) are encoded into numerical values with LabelEncoder, making them suitable for machine learning. The target variable (Electric Vehicle Type) is also encoded for classification. Finally, the first five rows of the preprocessed dataset are displayed using df_filtered.head().

Step 2: Splitting Data into Training and Testing Sets:-

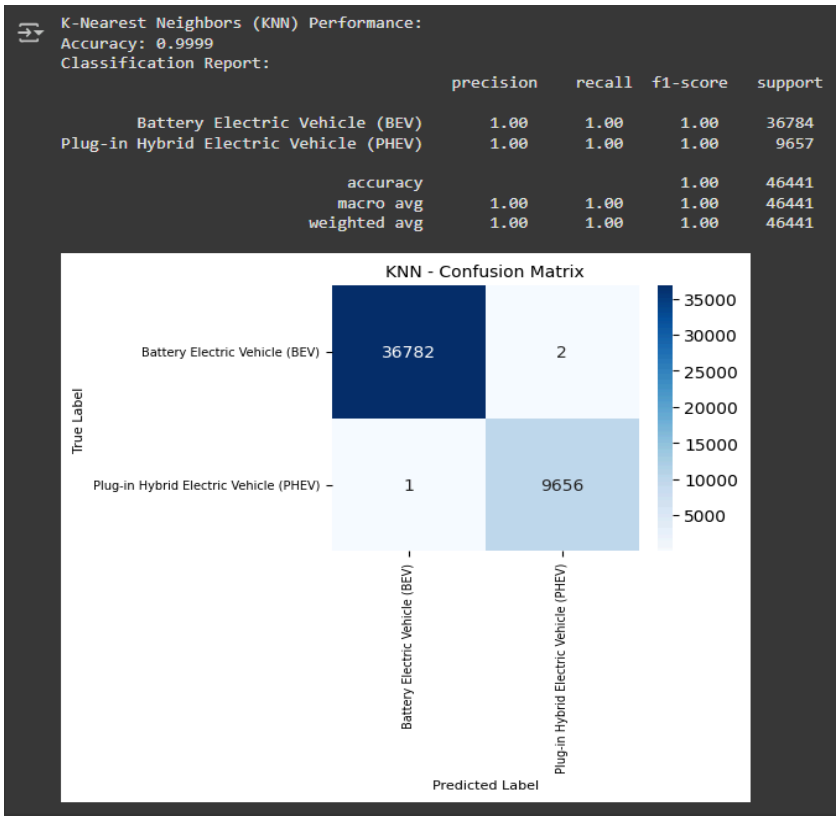
Command:

```
X = df_filtered.drop(columns=["Electric Vehicle Type"])
y = df_filtered["Electric Vehicle Type"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print(f"Training Data Shape: {X_train.shape}")
print(f"Testing Data Shape: {X_test.shape}")
```

```
Training Data Shape: (185762, 6)
Testing Data Shape: (46441, 6)
```

The code defines features (X) by dropping "Electric Vehicle Type" and sets y as the target variable. It then splits the dataset into 80% training and 20% testing sets using train_test_split(), ensuring reproducibility with random_state=42. Finally, it prints the shapes of the training and testing sets.

Step 3: Training and Evaluating K-Nearest Neighbours (KNN):-
Command: knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print("\nK-Nearest Neighbors (KNN) Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_knn):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred_knn,
target_names=target_encoder.classes_))
plt.figure(figsize=(4, 3)) # Compact size
cm = confusion_matrix(y_test, y_pred_knn)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=target_encoder.classes_,
yticklabels=target_encoder.classes_)
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()

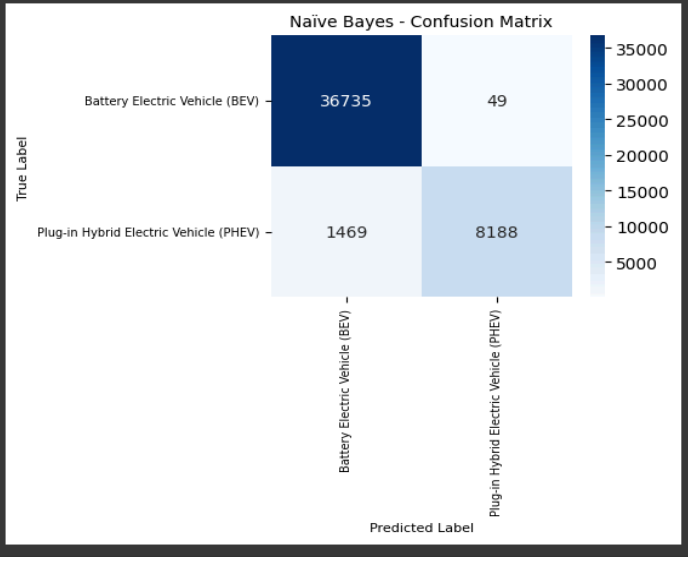


The code trains a K-Nearest Neighbors (KNN) model with n_neighbors=5 using the training data and makes predictions on the test set. It evaluates performance by computing accuracy and displaying a classification report. Additionally, it visualizes the confusion matrix using a heatmap to show the distribution of correct and incorrect predictions, making model performance easier to interpret.

Step 4: Training and Evaluating Naive Bayes:

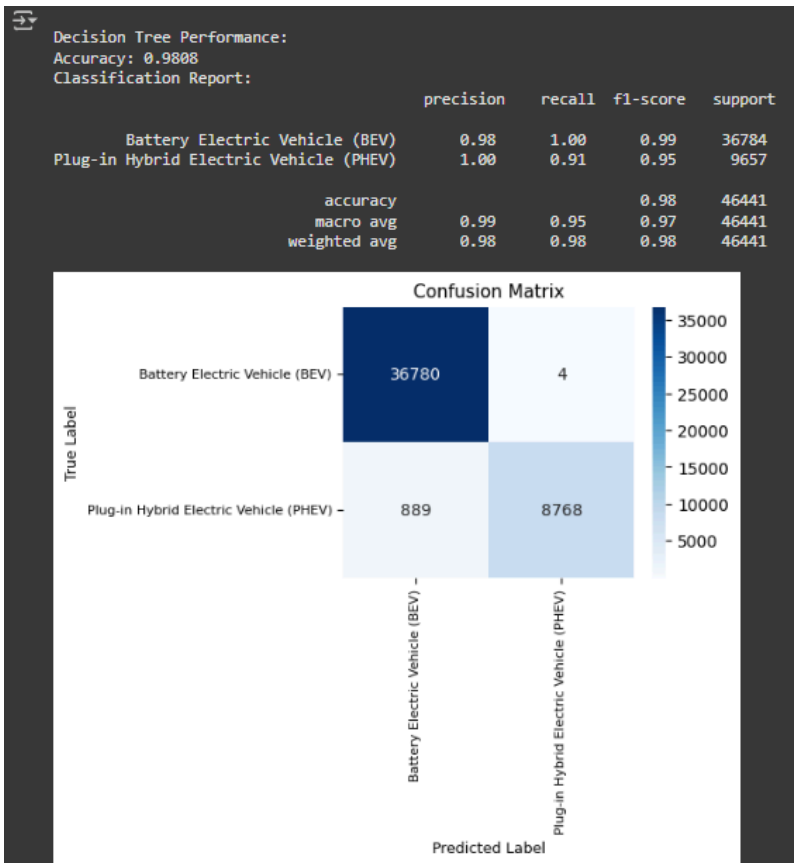
```
Command: nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
print("\nNaïve Bayes Performance:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_nb):.4f}")
print("Classification Report:\n", classification_report(y_test, y_pred_nb,
target_names=target_encoder.classes_))
plt.figure(figsize=(4, 3))  # Smaller size
cm = confusion_matrix(y_test, y_pred_nb)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=target_encoder.classes_,
yticklabels=target_encoder.classes_)
plt.xlabel("Predicted Label", fontsize=8)
plt.ylabel("True Label", fontsize=8)
plt.title("Naïve Bayes - Confusion Matrix", fontsize=10)
plt.xticks(fontsize=7)
plt.yticks(fontsize=7)
plt.show()
```

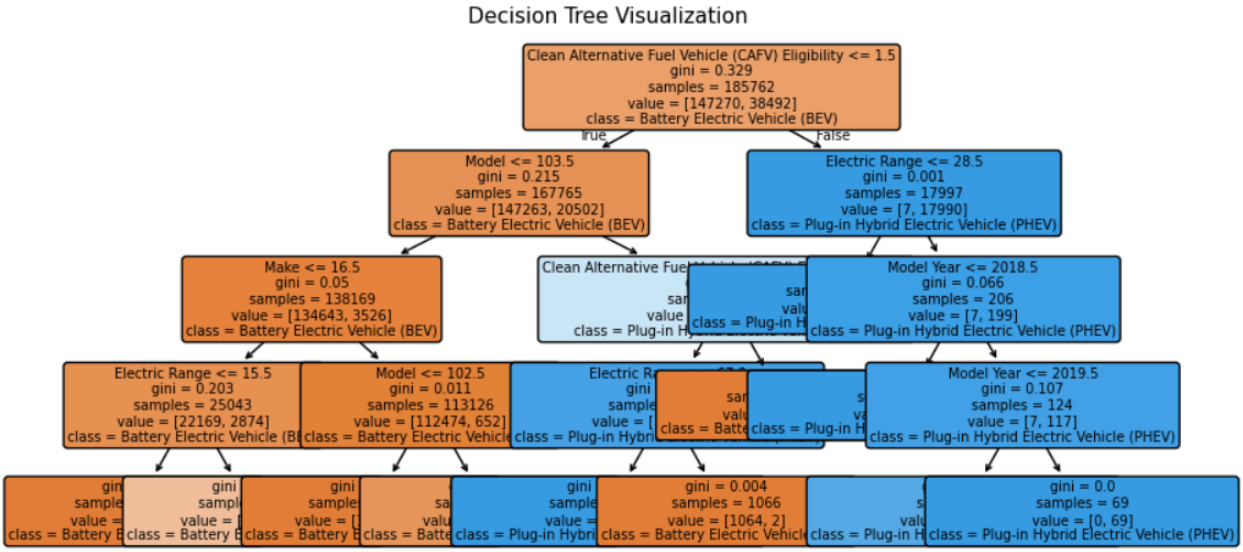
```
Naïve Bayes Performance:
Accuracy: 0.9673
Classification Report:
                              precision    recall  f1-score   support

       Battery Electric Vehicle (BEV)       0.96      1.00      0.98     36784
Plug-in Hybrid Electric Vehicle (PHEV)       0.99      0.85      0.92      9657

                            accuracy                           0.97     46441
                           macro avg       0.98      0.92      0.95     46441
                        weighted avg       0.97      0.97      0.97     46441
```



The code trains a Naïve Bayes classifier using the training dataset and makes predictions on the test set. Model performance is evaluated using accuracy and a classification report, which provide insights into precision, recall, and F1-score. Additionally, a confusion matrix is visualized with a heatmap to highlight correct and incorrect predictions, helping to assess the classifier's effectiveness and potential areas for improvement.

Step 5: Decision Tree Model Training, Evaluation and Visualization:-
Command: from sklearn.tree import plot_tree
dt = DecisionTreeClassifier(random_state=42, max_depth=3, min_samples_split=10, min_samples_leaf=5)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)
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=target_encoder.classes_))
plt.figure(figsize=(4, 3))
cm = confusion_matrix(y_test, y_pred_dt)
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=target_encoder.classes_, yticklabels=target_encoder.classes_)
plt.xlabel("Predicted Label", fontsize=9)
plt.ylabel("True Label", fontsize=9)
plt.title("Confusion Matrix", fontsize=11)
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.show()
plt.figure(figsize=(10, 5))
plot_tree(dt, filled=True, feature_names=X.columns, class_names=target_encoder.classes_, fontsize=7, rounded=True)
plt.title("Decision Tree Visualization", fontsize=11)
plt.show()





The code trains a Decision Tree classifier on the training dataset and makes predictions on the test set. Model performance is assessed using accuracy and a classification report to evaluate precision, recall, and F1-score. A confusion matrix is visualized using a heatmap to analyze correct and incorrect predictions. Additionally, a decision tree diagram is plotted, offering a detailed view of the model's decision-making process by showing feature splits and class assignments, aiding in model interpretability.

Step 6: Model Performance Comparison:-
Command:
```
model_performances = {
    "KNN": accuracy_score(y_test, y_pred_knn),
    "Naïve Bayes": accuracy_score(y_test, y_pred_nb),
    "Decision Tree": accuracy_score(y_test, y_pred_dt)
}
print("\nModel Performance Summary:")
for model, acc in model_performances.items():
    print(f"{model}: Accuracy = {acc:.4f}")
```

```
Model Performance Summary:
KNN: Accuracy = 0.9999
Naïve Bayes: Accuracy = 0.9673
Decision Tree: Accuracy = 0.9808
```

The code stores and compares the accuracy scores of different machine learning models, including K-Nearest Neighbors (KNN), Naïve Bayes and Decision Tree. These accuracy values are saved in a dictionary, model_performances, and then printed in a structured format to provide a quick overview of how well each model performed on the test dataset. This step helps in identifying the most effective model for classification.

Conclusion:

1. In this experiment, we learned how to perform classification modeling using different classifiers.
2. Decision Tree and KNN performed exceptionally well, achieving near perfect accuracy (0.9808 and 0.9999) respectively.
3. Naive Bayes had the lowest accuracy (0.9673), struggling with Plug-in Hybrid Electric Vehicles (PHEVs), misclassifying 1469 instances.
4. Decision Tree showed perfect classification, with only 4 misclassified BEVs, indicating clear decision boundaries.
5. KNN also performed nearly perfectly, misclassifying just 3 instances, proving its effectiveness for our dataset.
6. The Decision Tree visualization highlights key decision factors like ‘Electric Range’ and ‘CAFV Eligibility’ for classification.
7. Overall, Decision Tree and KNN are the best models while Naive Bayes is less suitable due to lower recall for PHEVs.

Experiment 7

Aim: To implement different clustering algorithms.

Problem Statement:
a) Clustering algorithm for unsupervised classification (K-means, density based (DBSCAN), Hierarchical clustering)
b) Plot the cluster data and show mathematical steps.

Theory:

- **K-Means Clustering:** K-Means is a centroid-based algorithm that partitions data into *k* clusters by minimizing the variance within each cluster. It starts by randomly placing *k* centroids, assigns each point to the nearest centroid, then updates the centroids based on the mean of assigned points. This process repeats until convergence. K-Means is efficient and works best when clusters are spherical and similarly sized, but it requires specifying *k* in advance and is sensitive to outliers.

- **DBSCAN Clustering:** DBSCAN groups together points that are closely packed (i.e., high-density regions), while marking points in low-density areas as outliers (noise). It uses two parameters: eps (neighborhood radius) and min_samples (minimum points to form a dense region). Unlike K-Means, it does not require specifying the number of clusters and can find arbitrarily shaped clusters. It's robust to noise but sensitive to parameter choice and struggles with varying densities.

- **Hierarchical clustering**: Hierarchical clustering builds a tree-like structure (dendrogram) of clusters by either merging smaller clusters (agglomerative) or splitting larger ones (divisive). The algorithm does not require the number of clusters upfront and allows visualization of clustering at different levels. Common linkage methods include ward, average, and centroid. It’s easy to interpret but can be computationally expensive for large datasets and sensitive to noisy data.

Performance:

- Prerequisite: Import essential libraries: pandas for data manipulation, numpy for numerical computations, matplotlib.pyplot and seaborn for data visualization, and sklearn for clustering using K-Means and DBSCAN. Also, use scipy for hierarchical clustering and PCA for dimensionality reduction. Load the Electric Vehicle Population Dataset into a Pandas DataFrame using pd.read_csv(). Finally, explore the dataset by displaying the first few rows with df.head() and checking column names, data types, and missing values using df.info():

Command:
```
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)
print("First 5 rows of the dataset:")
print(df.head())
print("\nDataset Information:")
print(df.info())
```

```
First 5 rows of the dataset:
   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

   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
```

```
Dataset Information:
<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
```

Step 1: Feature Selection and Preprocessing:-

```
Command: 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())
```

```
Scaled Data Sample:
   Model Year  Electric Range  Legislative District
0   -2.455485        0.666844              0.813147
1   -0.786089        2.053754             -1.870647
2    1.217186       -0.079953              0.410578
3    0.883307       -0.056245             -1.803552
4   -0.118331       -0.554111             -0.931319
```

The above code selects three relevant numerical features—'Model Year', 'Electric Range', and 'Legislative District'—from the electric vehicle dataset for clustering analysis. It removes any rows containing missing values to ensure clean data. The selected features are then standardized using StandardScaler, which scales the values to have a mean of 0 and standard deviation of 1. This normalization is important because it ensures that all features contribute equally to the clustering algorithms, preventing features with larger numeric ranges from dominating the results.

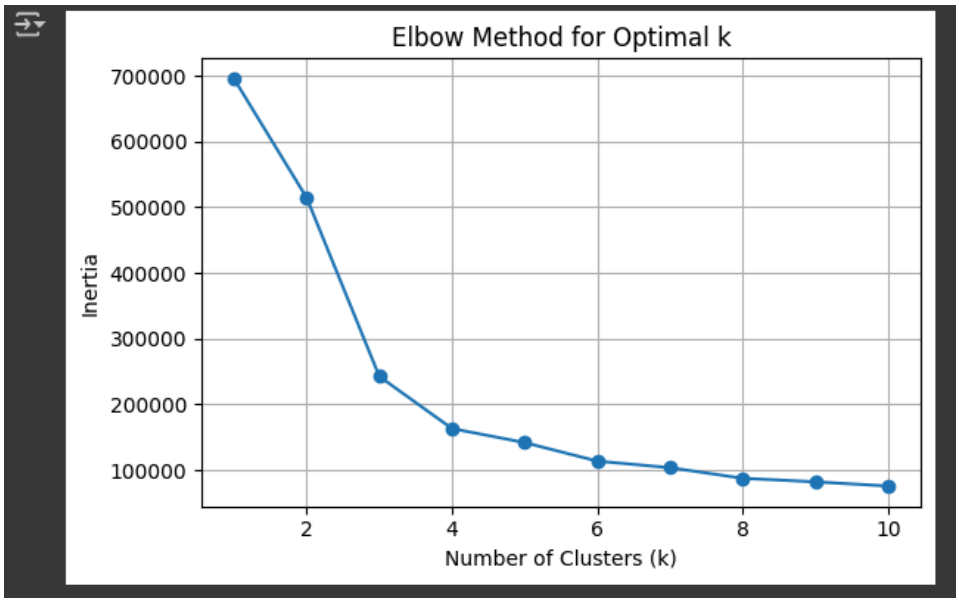
Step 2: Apply PCA for Dimensionality Reduction (for visualization):-

```
Command: 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())
```

```
PCA-Transformed Data Sample:
       PCA1      PCA2
0  2.238305  0.721200
1  1.932234 -1.946371
2 -0.900138  0.448977
3 -0.735446 -1.774065
4 -0.344920 -0.919696
```

The code applies Principal Component Analysis (PCA) to reduce the standardized dataset to two principal components—PCA1 and PCA2. This helps simplify the dataset while retaining most of its important information (variance), making it suitable for 2D visualization of clustering results. The transformed data is stored in a new DataFrame df_pca, and the first few rows are printed to preview the reduced dataset.

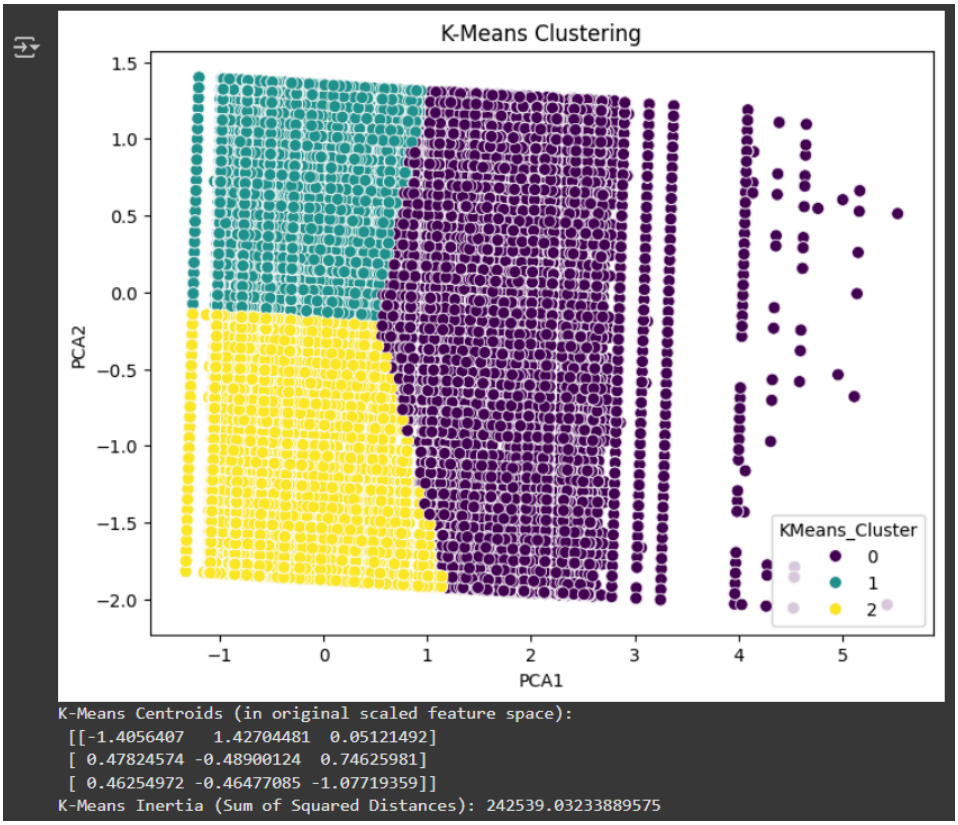
Step 3: Determining Optimal Number of Clusters Using the Elbow Method:-
Command: inertia = []
k_range = range(1, 11)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
plt.figure(figsize=(6, 4))
plt.plot(k_range, inertia, marker='o')
plt.title("Elbow Method for Optimal k")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.grid(True)
plt.tight_layout()
plt.show()



The code implements the Elbow Method to identify the optimal number of clusters (k) for K-Means clustering by plotting inertia values (sum of squared distances from points to their closest cluster center) against various k values from 1 to 10. The resulting plot shows a sharp drop in inertia from k=1 to k=3, after which the curve starts to flatten, forming an "elbow" around k=3 or k=4. This suggests that choosing 3 or 4 clusters would provide a good balance between model accuracy and simplicity—beyond this point, adding more clusters offers diminishing returns in reducing inertia. Thus, based on the curve, k=3 appears to be a strong candidate for optimal clustering.

Step 4: Apply K-Means Clustering:-

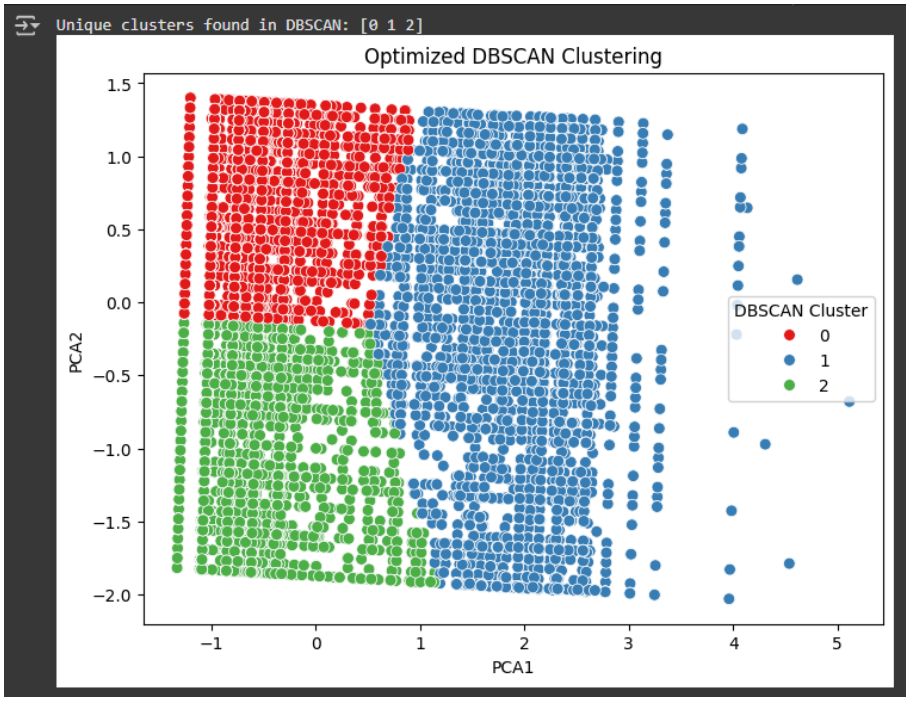
```
Command: kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
df_pca['KMeans_Cluster'] = kmeans.fit_predict(data_scaled)
plt.figure(figsize=(8,6))
sns.scatterplot(x=df_pca['PCA1'], y=df_pca['PCA2'], hue=df_pca['KMeans_Cluster'],
palette='viridis', s=50)
plt.title('K-Means Clustering')
plt.xlabel('PCA1')
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_)
```



This code performs K-Means clustering on the standardized dataset with 3 clusters, using a fixed random seed for reproducibility and n_init=10 to ensure stable results. The resulting cluster labels are added to the df_pca DataFrame for visualization. A scatter plot is then generated using the two PCA components to visually display how the data points are grouped into clusters. Finally, it prints the cluster centroids (in the original scaled feature space) and the inertia, which is the total within-cluster sum of squared distances—a measure of how well the clusters fit the data.

Step 5: Apply DBSCAN Clustering:

```
Command: 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()
```

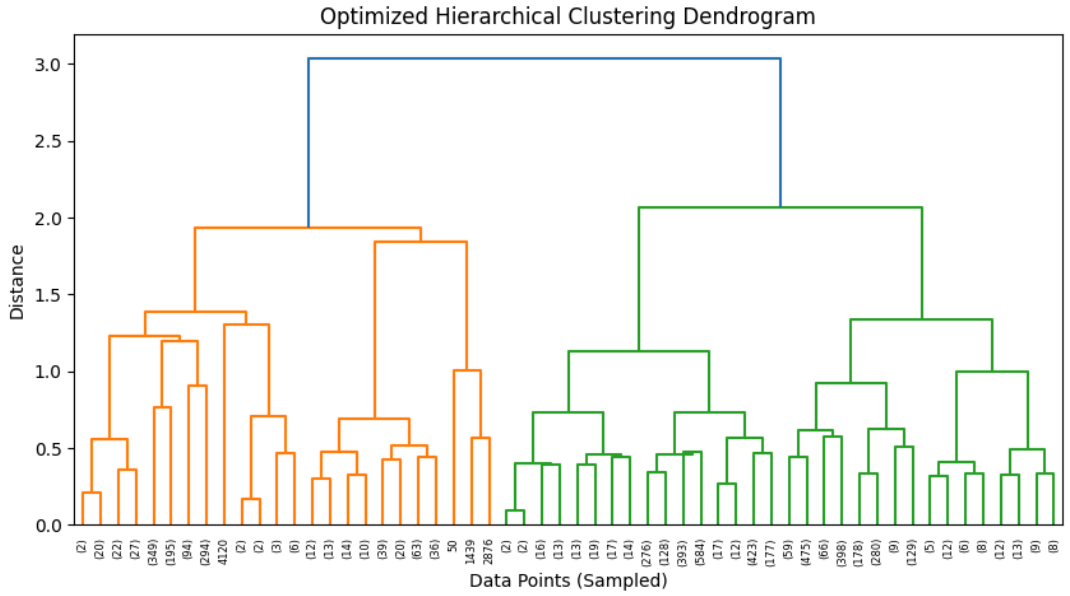


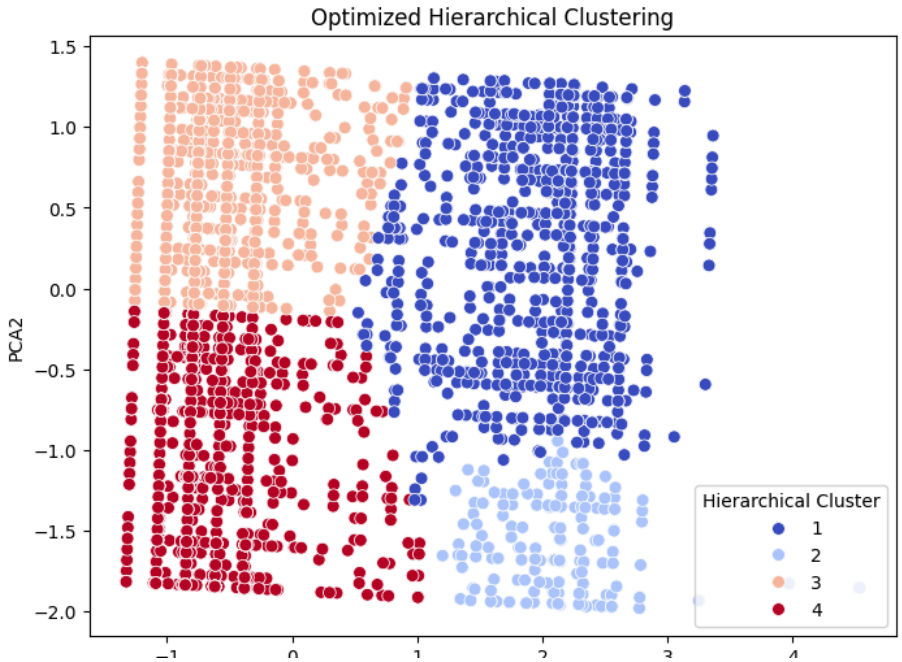
This code applies DBSCAN clustering to a reduced sample (20,000 rows) of the dataset to prevent Google Colab from crashing due to memory overload. It randomly shuffles the full PCA-transformed DataFrame (df_pca) and selects a sample to work with. DBSCAN is then applied with optimized parameters (eps=1.0, min_samples=10, n_jobs=1 to control CPU usage). After clustering, the unique cluster labels—including -1 for noise or outliers—are printed. A scatter plot is generated to visualize the clusters in 2D using the PCA components, with noise

points highlighted in black. This approach ensures DBSCAN runs efficiently while still providing meaningful clustering output.

Step 6: Apply Hierarchical clustering:

```
Command: from sklearn.utils import shuffle
df_pca_sample = shuffle(df_pca, random_state=42).sample(n=5000)  # Sample 5000 points
linkage_matrix = linkage(df_pca_sample, method='centroid')
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()
df_pca_sample['Hierarchical_Cluster'] = fcluster(linkage_matrix, t=4, criterion='maxclust')
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()
```





In this code, hierarchical clustering is performed on a random sample of 5,000 PCA-reduced data points to reduce memory usage and prevent system crashes. The 'centroid' linkage method is used, which clusters data points based on the centroid (mean) of clusters rather than individual distances, making it more efficient for larger datasets. A dendrogram is plotted with truncate_mode='level' and p=5 to visualize only the top 5 levels of the hierarchy, giving an overview without overloading the plot. Clusters are then extracted using fcluster with t=4, meaning the data is grouped into 4 final clusters. Finally, a scatter plot shows how these clusters are distributed in the reduced 2D PCA space. This method gives both a visual and analytical view of how the data groups together hierarchically.

Conclusion:

1. In this experiment, we learned how to perform different clustering algorithms.
2. K-Means clustering identified three distinct clusters but exhibited high inertia (242,539), indicating that it struggled with optimal separations, particularly in areas with dense data points.
3. DBSCAN successfully handled noise and discovered three clusters, but its clustering structure was highly dependent on parameter tuning, leading to an uneven distribution of cluster sizes.
4. Hierarchical clustering provided an interpretable dendrogram, effectively showcasing relationships between data points, but the final clustering outcome resulted in four distinct clusters, differing from the other methods.

5. Compared to K-Means, DBSCAN performed better in detecting non-uniform cluster densities, while hierarchical clustering provided more structured and detailed insights into data grouping.
6. K-Means is ideal for well-separated, structured clusters, DBSCAN excels with complex shapes and noise, and hierarchical clustering offers valuable visualization but becomes computationally expensive for large datasets.

Experiment 8

Aim: To implement recommendation system on your dataset using the following machine learning techniques:

- Regression
- Classification
- Clustering
- Decision tree
- Anomaly detection
- Dimensionality Reduction
- Ensemble Methods

Theory:

- **Recommendation types:** Recommendation systems help users discover relevant items by predicting their preferences. The main types include content-based filtering, collaborative filtering, and hybrid methods. Content-based filtering recommends items similar to those the user has liked before, using item features such as category, description, or brand. Collaborative filtering, on the other hand, leverages the preferences of similar users or the patterns in user-item interactions without requiring item-specific data. It can be user-based or item-based, depending on whether it finds similarities between users or between items. Hybrid systems combine both methods to overcome limitations such as data sparsity or the cold-start problem, often resulting in better performance.

- **Recommendation measures:** Evaluating recommendation systems is essential to measure their effectiveness and accuracy. For prediction-based systems, RMSE (Root Mean Square Error) and MAE (Mean Absolute Error) are widely used to quantify the difference between predicted ratings and actual user ratings. Lower values indicate better accuracy. For ranking-based recommendations, metrics like precision, recall, and F1-score assess the system's ability to rank relevant items higher in a list. Additionally, Mean Reciprocal Rank (MRR) and Normalized Discounted Cumulative Gain (NDCG) are used to measure the quality of ranked recommendations. These metrics help determine how well the system delivers useful and personalized results to users.

Performance:

- Prerequisite: The code starts by importing essential libraries for data manipulation (pandas, numpy), visualization (matplotlib.pyplot, seaborn), and modeling (sklearn for K-Means clustering and scaling, and surprise for collaborative filtering using SVD). It then loads six CSV files using pd.read_csv(): orders.csv, order_products__prior.csv, order_products__train.csv, products.csv, aisles.csv, and departments.csv. These datasets contain user order history and product details, which are used to build clustering and recommendation models.

Command:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from surprise import accuracy
orders = pd.read_csv("orders.csv")
order_products_prior = pd.read_csv("order_products__prior.csv")
order_products_train = pd.read_csv("order_products__train.csv")
products = pd.read_csv("products.csv")
aisles = pd.read_csv("aisles.csv")
departments = pd.read_csv("departments.csv")
```

- Preprocessing and Merging separate csv files into a single cohesive dataset:

Command:
```
import pandas as pd
products = products[["product_id", "aisle_id", "department_id", "product_name"]]
aisles = aisles[["aisle_id"]]
departments = departments[["department_id"]]
products = products.merge(aisles, on="aisle_id", how="left")
products = products.merge(departments, on="department_id", how="left")
order_products_prior = order_products_prior[["order_id", "product_id", "add_to_cart_order",
"reordered"]]
order_products_train = order_products_train[["order_id", "product_id", "add_to_cart_order",
"reordered"]]
order_products_prior = order_products_prior.merge(products, on="product_id", how="left")
order_products_train = order_products_train.merge(products, on="product_id", how="left")
order_products_prior = order_products_prior.sample(n=100000, random_state=42)
order_products_train = order_products_train.sample(n=50000, random_state=42)
all_orders = pd.concat([order_products_prior, order_products_train], ignore_index=True)
final_dataset = all_orders.merge(orders, on="order_id", how="left")
final_dataset.drop(columns=["eval_set", "aisle_id", "department_id"], inplace=True)
```

```
print("Final Dataset Columns:", final_dataset.columns)
print("Final Dataset Shape:", final_dataset.shape)
final_dataset.head()
```

```
Final Dataset Columns: Index(['order_id', 'product_id', 'add_to_cart_order', 'reordered',
       'product_name', 'user_id', 'order_number', 'order_dow',
       'order_hour_of_day', 'days_since_prior_order'],
      dtype='object')
Final Dataset Shape: (150000, 10)
```

	order_id	product_id	add_to_cart_order	reordered	product_name	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	3109255	34099	16	0	Crushed Red Chili Pepper	135284	9	0	19	8.0
1	301098	41950	5	0	Organic Tomato Cluster	7293	2	4	15	1.0
2	1181866	45066	8	0	Honeycrisp Apple	111385	2	1	17	8.0
3	1678630	8859	2	1	Natural Spring Water	147365	7	0	14	26.0
4	644090	24781	2	0	PODS Laundry Detergent, Ocean Mist Designed fo...	99290	7	0	19	30.0

The code efficiently preprocesses the Instacart dataset by selecting only essential columns, including product_name, to minimize memory usage while preserving key product details. It merges the products dataset with aisles and departments to enrich each product entry with hierarchical information. The merging of order_products_prior and order_products_train with product details is performed to associate each ordered product with its name and category, enabling meaningful insights and recommendations. Sampling is used to reduce dataset size for faster processing. Finally, merging with the orders dataset adds user-level and temporal context to each transaction, which is crucial for building personalized recommendation systems.

Step 1: Clustering Users Based on Shopping Behavior:

```
Command: from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
user_data = final_dataset.groupby("user_id").agg({
    "order_number": "mean",
    "days_since_prior_order": "mean",
    "add_to_cart_order": "mean"
}).reset_index()
imputer = SimpleImputer(strategy="mean")
user_data[["order_number", "days_since_prior_order", "add_to_cart_order"]] =
imputer.fit_transform(
    user_data[["order_number", "days_since_prior_order", "add_to_cart_order"]]
)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(user_data[["order_number", "days_since_prior_order",
"add_to_cart_order"]])
kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)
user_data["Cluster"] = kmeans.fit_predict(scaled_features)
print("Cluster Counts:\n", user_data["Cluster"].value_counts())
```

```
Cluster Counts:
 Cluster
2    30293
1    22321
4    16973
0    11447
3     6634
Name: count, dtype: int64
```

This code clusters users based on their shopping behavior to uncover distinct patterns among customer segments. First, it aggregates user-level features like the average number of orders, time between orders, and cart position. Missing values in these features are handled using mean imputation to ensure model reliability. The features are then normalized using StandardScaler to bring all variables to a common scale. Finally, K-Means clustering with 5 clusters is applied to group users with similar purchasing habits, and the distribution of users across clusters is displayed to understand segment sizes. This clustering can later help tailor recommendations to user groups with shared behaviors.

Step 2: Collaborative Filtering Using Matrix Factorization (SVD):

```
Command: cf_data = final_dataset[["user_id", "product_id", "reordered"]].dropna()
reader = Reader(rating_scale=(0, 1))
data = Dataset.load_from_df(cf_data, reader)
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)
svd = SVD(n_factors=50, random_state=42)
svd.fit(trainset)
predictions = svd.test(testset)
rmse = accuracy.rmse(predictions)
print("Collaborative Filtering RMSE:", rmse)
```

```
RMSE: 0.4800
Collaborative Filtering RMSE: 0.47996185938330604
```

This code implements collaborative filtering to recommend products based on user-product interactions. It begins by preparing a dataset with user_id, product_id, and reordered as the implicit rating, where a reorder indicates user preference. The Surprise library's Reader and Dataset modules convert the DataFrame into a suitable format, followed by an 80-20 train-test split. A Singular Value Decomposition (SVD) model is trained on the training set to learn latent features of users and products. Finally, predictions are made on the test set and evaluated using RMSE (Root Mean Square Error) to measure the model's accuracy in predicting reorder behavior.

The model achieved a Collaborative Filtering RMSE of 0.47996, which suggests a relatively good level of predictive accuracy. A lower RMSE indicates that the predicted reorder values are close to the actual values, meaning the model is effective at capturing user preferences and recommending products that align with their past shopping behavior.

Step 3: Generating Personalized Product Recommendations:
Command:

```
def recommend_products(user_id, num_recommendations=5):
    unique_products = final_dataset["product_id"].unique()
    predicted_ratings = [(product, svd.predict(user_id, product).est) for product in
unique_products]
    top_products = sorted(predicted_ratings, key=lambda x: x[1],
reverse=True)[:num_recommendations]
    recommended_product_ids = [prod[0] for prod in top_products]
    recommended_products =
products[products["product_id"].isin(recommended_product_ids)][["product_id",
"product_name"]]
    return recommended_products
recommendations = recommend_products(user_id=1)
print("\nRecommended Products:\n", recommendations)
```

```
Recommended Products:
        product_id                          product_name
12383       12384     Organic Lactose Free 1% Lowfat Milk
24023       24024                          1% Lowfat Milk
45444       45445  Organic Bagged Mini Dark Peanut Butter
45602       45603                 Trilogy Kombucha Drink
48108       48109  Tortilla Chips, Clasico, Jalapeno Lime
```

The above function leverages the trained SVD collaborative filtering model to recommend products to a specific user. It begins by identifying all unique products in the dataset and predicts the reorder likelihood for each using the model. The top-N products with the highest predicted scores (by default, 5) are selected and mapped to their respective names using the products DataFrame. This process ensures that the recommendations are both accurate and interpretable.
As shown above, when executed for user_id=1, the model recommended five specific items based on the user's shopping behavior: Organic Lactose Free 1% Lowfat Milk, 1% Lowfat Milk, Organic Bagged Mini Dark Peanut Butter, Trilogy Kombucha Drink, and Tortilla Chips, Clasico, Jalapeno Lime. These results illustrate how collaborative filtering can successfully generate personalized product suggestions that are relevant and likely to be of interest to the user.

Conclusion:

1. In this experiment, we learned how to perform a recommendation system on our dataset using the 'Clustering' machine learning technique.
2. K-Means clustering was effectively applied to group users based on behavioral patterns such as order frequency, recency, and cart activity, allowing us to uncover distinct user segments with similar shopping habits.

3. Collaborative Filtering using Matrix Factorization (SVD) demonstrated strong predictive performance by achieving a low RMSE of 0.47996, indicating the model’s ability to accurately estimate user preferences and reorder likelihood.
4. The recommendation engine successfully generated relevant product suggestions such as Organic Lactose Free 1% Lowfat Milk, Trilogy Kombucha Drink, and Tortilla Chips, reflecting the system's strength in delivering personalized and appealing recommendations.
5. By leveraging user purchase history and clustering, the system enhanced the understanding of user needs, which is crucial for building intelligent recommendation systems that can increase user engagement and satisfaction.
6. In conclusion, the integration of clustering for user segmentation and collaborative filtering for recommendation generation forms a powerful hybrid approach that not only improves accuracy but also helps tailor the shopping experience more effectively.

Experiment 9

Aim: To perform exploratory data analysis using Apache Spark and Pandas.

Theory:

1. What is Apache Spark and how does it work?

Ans) Apache Spark is an open-source data processing engine designed for speed and scalability. It supports batch processing, real-time streaming, machine learning, and graph analytics in one unified platform.
Unlike Hadoop MapReduce, Spark processes data in memory, making it much faster for iterative and complex tasks. It supports languages like Python, Scala, Java, and SQL, and can run on various cluster managers like YARN, Kubernetes, or standalone.
Spark is widely used in big data environments for its performance, ease of use, and flexibility.

Core Components of Apache Spark are:

- **Spark Core**: Core engine (scheduling, memory, etc.)
- **Spark SQL**: For SQL queries and DataFrames
- **Structured Streaming**: Real-time processing
- **MLlib**: Machine learning
- **GraphX**: Graph processing

It works in the following ways:

1. **Driver Program**: Starts the app, defines the workflow (DAG), and manages coordination.
2. **Cluster Manager**: Allocates resources (Standalone, YARN, Mesos, Kubernetes).
3. **Executors**: Run tasks on worker nodes and store data.
4. **In-Memory Computing**: Keeps data in memory for fast processing.
5. **APIs**: Uses RDDs for low-level tasks, DataFrames/Datasets for optimized, SQL-like operations.

2. How is data exploration done in Apache Spark? Explain with steps.

Ans) Data exploration in Apache Spark is done as follows:

1. **Initialize Spark Session**: Start a Spark session to enable interaction with Spark's features and APIs.
2. **Load Data**: Import your dataset into Spark from sources like CSV, JSON, databases, or Parquet files.
3. **View Data Sample**: Look at the first few rows to get a quick sense of the data content and format.
4. **Check Schema**: Examine the structure of the dataset, including column names and data types.
5. **Summary Statistics**: Generate basic statistics like mean, count, min, and max for numerical columns to understand distributions.
6. **Check for Missing Values**: Identify columns that have null or missing values to assess data quality.
7. **Value Counts / Grouping**: Group data by specific columns to analyze category frequencies or distributions.
8. **Filter and Query Data**: Apply filters or queries to focus on specific subsets of the data for deeper analysis.

Conclusion: In this experiment, we learned how to perform exploratory data analysis using Apache Spark and Pandas. By examining the data's structure, statistics, missing values, and groupings, we gain key insights to guide cleaning, feature selection, and modeling. Spark's speed and scalability make it ideal for exploring large datasets efficiently.

Experiment 10

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

Theory:

1. What is streaming? Explain batch and stream data.

Ans) Streaming is the process of transmitting and processing data continuously and in real-time. Instead of waiting for all the data to be collected (like in batch processing), streaming handles data as soon as it's generated. This allows for immediate analysis and actions — useful in scenarios like fraud detection, live video, and real-time analytics.

A. **Batch Data Analysis**:

- Processed in groups after collecting over time.
- Not real-time, higher latency.
- Examples: Monthly billing, daily sales reports.
- Tools: Hadoop, Hive.

B. Streamed Data Analysis:

- Processed in real-time as it's generated.
- Low latency, continuous flow.
- Examples: Live chat, stock market feeds.
- Tools: Kafka, Spark Streaming.

The main difference is that in batch processing, data is processed later in chunks, while in stream processing, data is processed continuously in real-time as it arrives.

2. How does data streaming take place in Apache Spark?

Ans) In Apache Spark, data streaming is handled by Spark Streaming, which allows for the processing of real-time data in a distributed and fault-tolerant manner. Here's how the process works:

1. **Data Ingestion**: Spark Streaming can ingest real-time data from various sources like Kafka, Flume, Sockets, or file systems like HDFS or Amazon S3. The data is continuously received in small chunks.
2. **Micro-batches**: Spark Streaming divides the incoming data stream into micro-batches, which are small, manageable batches that are processed in intervals (typically in milliseconds or seconds). This approach allows Spark to handle streaming data while leveraging its batch processing engine.
3. **Processing**: Each micro-batch is processed using Spark's standard transformations (like map, filter, etc.) and actions (like reduce, count, etc.). These operations are applied on the data in the batch, similar to how traditional Spark processes batch data.
4. **Output**: After processing, the results of the micro-batches are typically written to external storage systems (such as HDFS, databases, or dashboards) or used to trigger further actions.
5. **Fault Tolerance:** Spark Streaming ensures fault tolerance by replicating the data across multiple nodes and periodically checkpointing the data, so that the system can recover from failures without losing progress.

Conclusion: In this experiment, we learned how Apache Spark Streaming processes real-time data using micro-batches. It enables efficient, scalable, and fault-tolerant real-time analytics, making it ideal for applications like monitoring and fraud detection. Spark Streaming provides a powerful solution for large-scale data processing.

Name - Anish Kulkarni
Roll No. - 29
Class - D15C

05/05

AIDS Assignment - 1

Q1) What is AI? Considering the COVID-19 pandemic situation, how AI helped to survive and renovated our way of life with different applications?

Ans) Artificial Intelligence (AI) enables machines to perform human-like tasks such as problem-solving, learning, and decision-making using algorithms and data-driven techniques.

AI helped during the COVID-19 pandemic in the following ways:-

i) Healthcare: Predicted outbreaks, diagnosed cases, and aided drug discovery.

ii) Contact Tracing: AI apps and thermal cameras tracked infections.

iii) Remote Work: AI chatbots and tools improved virtual collaboration.

Post-pandemic, AI transformed telemedicine, remote work, e-learning, automation, and entertainment, becoming a vital part of daily life.

Q2) What are AI agents terminology? Explain with examples.

Ans) AI agents are systems that perceive their environment, process information, and take actions to achieve specific goals. Key terminologies include:

i) Agent - An entity that interacts with the environment (e.g. a self driving car).

iii) Environment - The external system in which the agent operates (e.g. traffic conditions for a self-driving car).

iii) Perception - Data collected from sensors (e.g. a robot's camera detecting obstacles).

iv) Actuators - Components that execute actions (e.g. robotic arms in manufacturing).

v) Rationality - The ability to make optimal decisions based on available information (e.g. AI trading bots adjusting stock investments).

vi) Autonomy - The degree of independence an agent has (e.g. voice assistants like Alexa operating without human input).

viii) Types of Agents:

a) Simple Reflex Agents - Act based on current perceptions (e.g. thermostats).

b) Model-Based Agents - Use internal models to predict future states (e.g. GPS navigation systems).

c) Goal-Based Agents - Take actions to achieve a specific goal (e.g. chess-playing AI).

d) Utility-Based Agents - Optimise outcomes for maximum benefit (e.g. recommendation systems in e-commerce).

e) Learning Agents - Improve performance over time using past experiences (e.g. self-learning robots in warehouses).

Q2) How is AI technique used to solve 8-puzzle problem?

Ans) The 8-puzzle problem involves arranging numbered tiles in a 3x3 grid using AI to find the shortest solution. The AI techniques used are:

1. Uninformed Search (Explores all moves blindly) :-

i) BFS: Finds the shortest path by exploring level-by-

level.

ii) DFS: Explores deeply but may not be optimal.

2. Informed Search (Uses heuristics for efficiency):-

i) A*: Combines path cost and heuristics for optimal moves.

ii) Heuristics (Best-first): Calculates the estimated distance from the goal node based on two points of the original position.

3. Optimisation Methods:-

i) Genetic Algorithms: Evolves solutions over multiple iterations.

ii) Simulated Annealing: Avoids getting stuck in bad solutions.

Qu) What is PEAS descriptor? Give PEAS descriptor for following:

i) Taxi Driver

ii) Medical diagnosis system

iii) A music composer

iv) An aircraft autolander.

v) An essay evaluator.

vi) A robotic sentry gun for the Keck Lab.

Ans) PEAS (Performance Measure, Environment, Actuators, and Sensors) is a framework used to define the structure and functioning of an AI agent. It describes how an agent interacts with its environment to achieve its goal.

Teacher's Sign.: _____

	AI System	Performance Measure (P)	Environment (E)	Actuators (A)	Sensors (S)
i)	Taxi Driver AI	Safety, efficiency, fuel usage, passenger comfort	Roads, traffic, passengers, weather.	Steering wheel, brakes, signals, accelerator.	GPS, cameras, speedometer, proximity sensors, microphone.
ii)	Medical Diagnosis System	Accuracy, speed, reliability, patient satisfaction	Patient symptoms, medical history, lab reports	Display screen, reports, prescriptions	Patient input, medical test results, wearable health monitors
iii)	AI Music Composer	Melody quality, harmony, originality	Music theory rules, user preferences, compositions	Music generation software, speakers.	User feedback, genre data, music databases.
iv)	Aircraft Autolander	Smooth landing, accuracy, passenger safety	Airplane altitude, weather, runway conditions	Flaps, landing gear, thrust control, brakes	Altimeter, GPS, radar, wind sensors, airspeed indicators.
v)	AI Essay Evaluator	Grammar accuracy, coherence, structure, relevance.	Essays, grammar rules, grading rubrics.	Screen output, grading suggestions, feedback	Text input, grammar checking tools.
vi)	Robotic Sentry Gun	Accuracy, response time, threat detection efficiency	Lab surroundings, potential threats, movement patterns	Gun turret, alert system, camera movement	Motion detectors, thermal sensors, facial recognition cameras.

Q5] Categorise a shopping bot for an offline bookstore according to each of the six dimensions (fully / partially observable, deterministic / stochastic, episodic / sequential, static / dynamic, discrete / continuous, single / multi agent).

Ans] A shopping bot for an offline bookstore can be categorised using the six AI environment dimensions as follows:

- Partially observable: The bot does not have complete visibility into real-time inventory, customer preferences, or book availability.
- Stochastic: Customer choices, stock updates, and staff interventions introduce uncertainty, making the environment unpredictable.
- Sequential: Each customer interaction depends on prior actions such as recommendations based on past selections.
- Dynamic: The environment changes as books are purchased, restocked, or reordered, affecting the bot's decision making.
- Discrete: The bot operates through distinct actions like checking stock, suggesting books, and processing purchases.
- Multi-Agent: The bot interacts with multiple entities, including customers, bookstore staff, and inventory management systems.

Q6] Differentiate between Model-based and Utility-based agent.

Ans]

Model-based Agent	Utility-based agent
i) It uses an internal model of the environment to make decisions.	i) It chooses actions based on maximising a utility function.

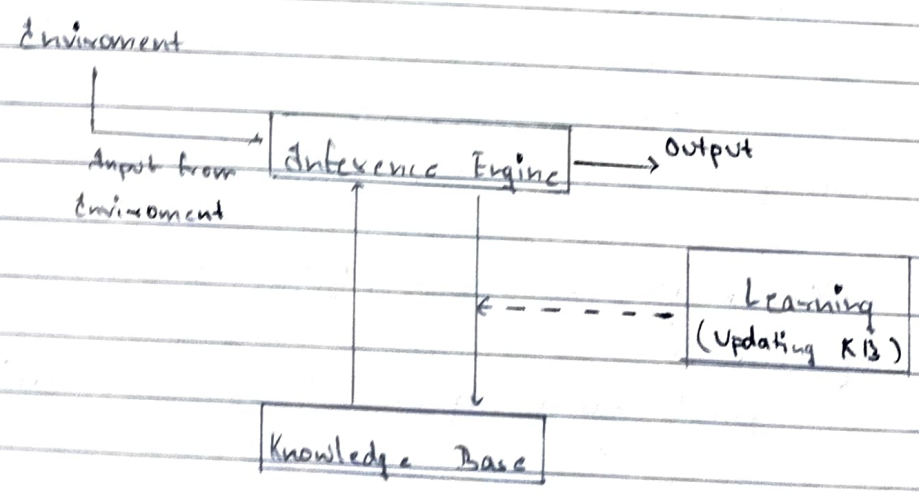
Teacher's Sign.: _____

ii) It predicts future states using the model before acting.	ii) It evaluates multiple possible actions and selects the best based on utility.
iii) It is goal-oriented but focuses on how the environment works.	iii) It is goal-oriented but aims to achieve the best possible outcome.
iv) It may not always take the best possible action, only a feasible one.	iv) It always selects the action that provides the highest benefit.
v) For example, a self-driving car using a traffic model to predict congestion.	v) For example, a stock trading AI selecting the trade with the highest expected profit.

Q7) Explain the architecture of a knowledge based agent and learning agent.

Ans)

- Architecture of a Knowledge-Based Agent:-



Teacher's Sign.:

A Knowledge-Based Agent uses stored knowledge to make decisions and reason about the environment. Its architecture consists of:

i) Knowledge Base (KB): Stores facts, rules, and background knowledge in a structured format.

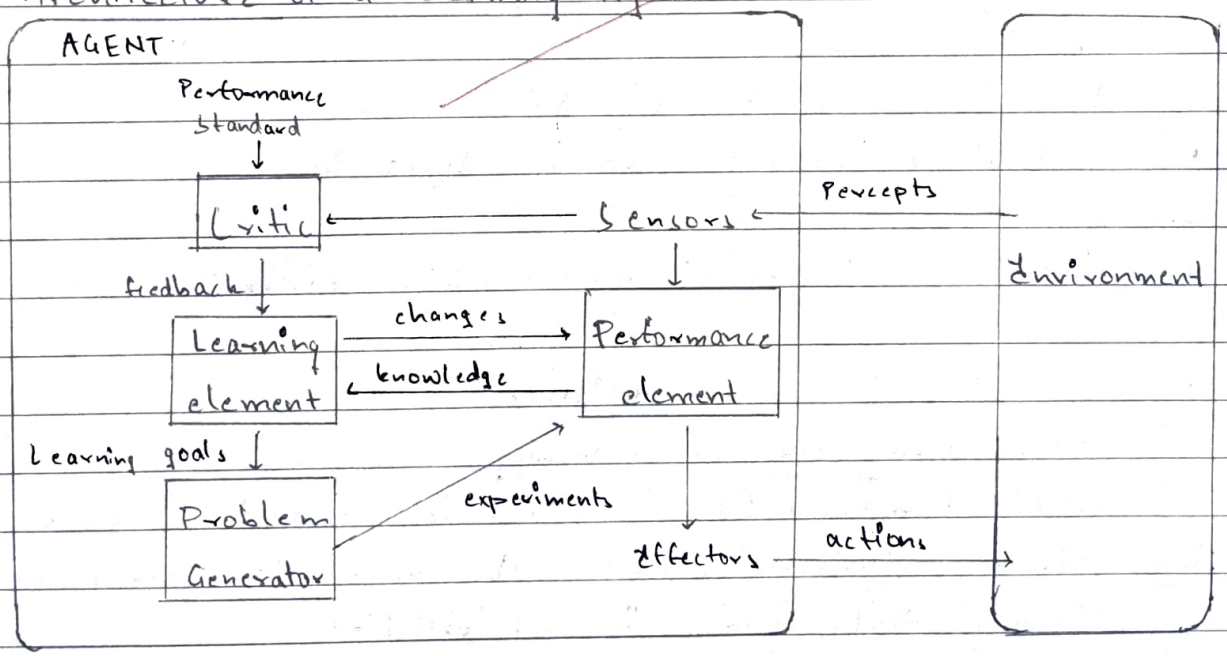
ii) Inference Engine: Applies logical reasoning to derive conclusions from the knowledge base.

iii) Perception Module: Gathers inputs from sensors or external sources.

iv) Action Module: Executes actions based on derived conclusions.

v) Learning Mechanism: Updates the knowledge base with new information.

- Architecture of a Learning Agent:-



A Learning Agent improves its performance over time by learning from past experiences. It has four main components:

i) Learning Element: Learns from interactions and updates

Teacher's Sign.: _____

knowledge.

ii) Performance Element: Uses the learned knowledge to make decisions.

iii) Critic: Evaluates the agent's performance by comparing actions with desired outcomes.

iv) Problem Generator: Suggests new exploratory actions to improve learning.

Q9) Convert the following to predicates:

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

b. Bus goes via Andheri and Goregaon.

c. Car has puncture so it is not available.

Will Anita travel via Goregaon? Use forward reasoning.

Ans) a.

- ~~Available (Car)~~ → Travels (Anita, Car)
- ¬ Available (Car) → Travels (Anita, Bus)

b.

- ~~Route (Bus, Andheri)~~ GoesVia (Bus, Andheri ∧ Goregaon)

c.

- Puncture (Car) → ¬ Available (Car)

Now, using forward reasoning:-

i) Given Punctured (Car), we infer: ¬ Available (Car)

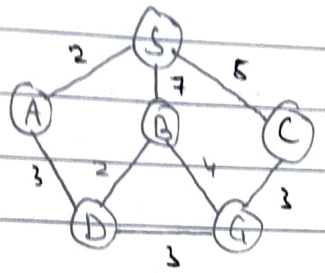
ii) Since ¬ Available (Car), the rule Available (Car) → Travels (Anita, Car) is not satisfied.

iii) Therefore, the alternative rule applies: ¬ Available (Car) → Travels (Anita, Bus)

iv) Since GoesVia (Bus, Goregaon) and Anita is travelling by bus, she will travel via Goregaon.

Teacher's Sign.: _____

Q10) Find the route from S to G using BFS.



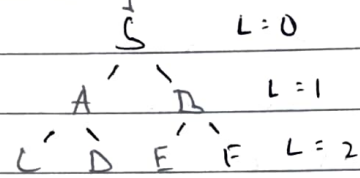
Ans) We know that, in BFS, queue is used.
So starting S, it gets popped from queue and its neighbours get added in order of priority:-
[(A, 2), (C, 5), (B, 7)]
Then, A gets ~~popped~~ dequeued and its neighbours get added:-
[(D, 3), (C, 5), (B, 7)]
Then, D gets dequeued and its neighbours get added to the queue:-
[(B, 2), (G, 3), (C, 5), (B, 7)]
We see that, the target node G is found.
Thus, route $S \rightarrow A \rightarrow D \rightarrow G$ is correct (path length: $2+3+3=8$)
Similarly,
$S \rightarrow C \rightarrow G$ (path length = 8)
Thus, either of the two paths are optimal.

Q11) What do you mean by Depth Limited Search? Explain Iterative Deepening Search with example.

Ans) Depth-limited search is a variation of Depth-First Search where the search is restricted to a specific depth limit L. If a goal is not found within this limit, the search terminates. It helps in preventing the search from

being infinitely stuck in a branch. For ex: if L = 2, only nodes upto depth = 2 are checked through DFS.

ii) Iterative Deepening Search (IDS) combines memory efficiency of DFS and completeness of BFS by repeatedly running DLS with increasing depth limits until the goal is found. For ex (searching for E):-



→ At DLS with L=0, checks S, goal isn't found.
→ At DLS with L=1, check A, B, goal isn't found.
→ DLS with L=2, checks C, D, E, F, goal is found.

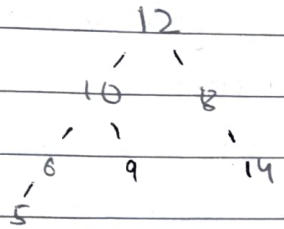
Q12) Explain Hill climbing and its drawbacks in detail with example. Also state limitations of steepest-ascent hill climbing.

Ans) Hill climbing is an iterative optimisation algorithm that starts with an initial solution and makes incremental changes (using greedy method) to improve it. The goal is to reach the best possible solution by always moving towards higher values in the search space.

Drawbacks of Hill Climbing:-

i) Local Maxima - It may stop at a node which is the local maxima but not the global maxima.
ii) Plateau - Flat regions can prevent further movement in Hill Climbing algorithm.
iii) Ridges - It struggles if progress requires a temporary decline because it assumes it as a peak.

For ex:-



→ Starts at 5, neighbours are 6.
→ Moves to 6, neighbours are 10, 9.
→ Chooses 10 as it is higher than to 9.
→ Then chooses 12 as it is highest value and then stops.

Here, we see that 12 is local maxima and 14 is global maxima but search stops at 12. This is the major drawback.

Limitations of Steepest-Ascent Hill Climbing:-

i) High computational cost as evaluating all neighbours is expensive.
ii) Still prone to getting stuck in local maxima.
iii) It can take longer than simpler versions.

Q12) Explain simulated annealing and write its algorithm.

Ans) Simulated annealing is a probabi~~stic~~listic optimisation algorithm inspired by the annealing process in metallurgy. It helps in finding a global optimum by allowing occasional moves to worse solutions to escape local optima. It uses Metropolis algorithm, which is based on Monte Carlo technique. Here, a solution for a problem is equated to states in a physical system and the cost of a solution is equated to the energy of a state.

Algorithm of simulated annealing:-

1. Initialize the current state and temperature T.

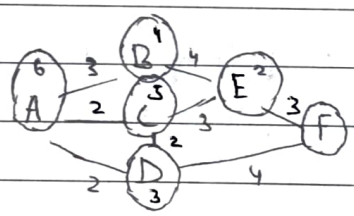
2. Repeat until stopping condition is reached:
   i) Select a random neighbour state.
   ii) Compute energy difference: $\Delta E = E_{new} - E_{current}$
   iii) If $\Delta E < 0$, accept the new state.
   iv) Else, accept it with probability $e^{-\Delta E/T}$ (Metropolis criterion)
   v) Reduce the temperature 'T' according to cooling schedule.
3. Return the best-found solution.

Qn) Explain A* algorithm with an example.

Ans) The A* algorithm is an informed search algorithm that finds the optimal path by considering both the cost to reach a node ($g(n)$) and the estimated cost to the goal ($h(n)$) using the formula:

$$f(n) = g(n) + h(n)$$

For ex:-



To go from A to F,

→ at A, there are 3 neighbours:-

i) B: $h(n) = 4$, $g(n) = 3 \rightarrow \therefore f(n) = 4+3 = 7$

ii) C: $h(n) = 5$, $g(n) = 2 \rightarrow \therefore f(n) = 5+2 = 7$

iii) D: $h(n) = 3$, $g(n) = 2 \rightarrow \therefore f(n) = 3+2 = 5$

Thus, as $f(n)$ is minimum for D, we move to D.

→ at D, there are 2 neighbours:-

i) C: $h(n) = 5$, $g(n) = 2 \rightarrow \therefore f(n) = 5+2 = 7$.

ii) F: $h(n) = 0$, $g(n) = 4 \rightarrow \therefore f(n) = 0+4 = 4$

Now, F is goal node. Thus, the search is complete.

Q15] Explain Minimax algorithm and draw game tree for Tic-Tac-Toe game.

Ans] The Minimax algorithm is a decision-making strategy used in two-player games like chess and tic-tac-toe. It helps find the best move by simulating all possible future moves and assuming that the opponent plays optimally.

It works in the following way :-

i] Maximising Player (MAX) : Tries to maximise the score.

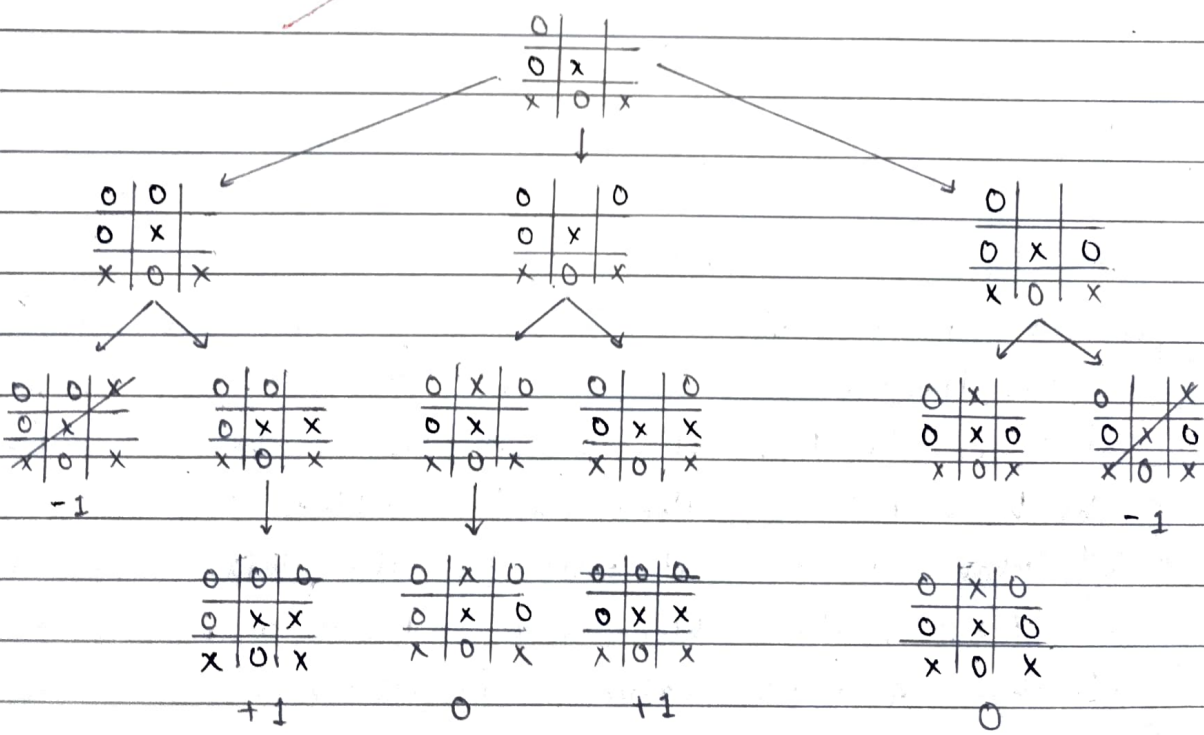
ii] Minimising Player (MIN) : Tries to minimise the score.

iii] It generates all possible game states upto a certain depth.

iv] It assigns scores to terminal states based on winning chances.

v] The best move is chosen assuming both players play optimally.

Game tree for Tic-Tac-Toe :-



Q16) Explain Alpha-Beta Pruning Algorithm for ~~adversial~~ adversarial search with an example.

Ans) Alpha-Beta Pruning is an optimisation technique for the Minimax algorithm that reduces the no. of nodes evaluated, making it more efficient. Instead of exploring all game states, the algorithm prunes branches that cannot influence the final decision. This is how it works:

i) Alpha(α): Best value MAX can guarantee.

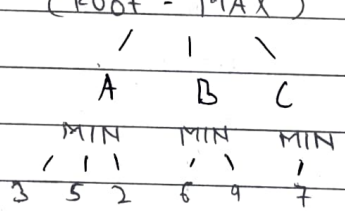
ii) Beta(β): Best value MIN can guarantee.

iii) While traversing the game tree:

a) If a node's value is worse than the best alternative, stop exploring ($\alpha > \beta$).

b) This avoids evaluating unnecessary nodes.

For ex:-



i) At Root, $\alpha = -\infty$, $\beta = +\infty$.

ii) At A, it evaluates the child values (3, 5, 2) and selects the lowest value(2). Thus, $\beta = 2$ for this branch.

iii) Move to B. It evaluates the child values (6, 9). As both 6, 9 > 2, the entire subtree is pruned (as $\alpha > \beta$).

iv) Move to C. It evaluates the child value (7). As 7 > 2, the subtree is pruned (as $\alpha > \beta$).

v) Thus, A with value 2 is chosen.

Q17 Explain WUMPUS World environment giving its PEAS description. Explain how percept sequence is generated.

Ans The Wumpus World is a grid-based AI environment where an agent explores a cave while avoiding hazards like pits and the Wumpus monster.

PEAS Descriptor :-

- Performance Measure : +1000 points for reaching the gold and returning back safely, -1000 for dying, -1 for every move of the agent, -10 points when the agent uses the arrow.
- Environment : A 4x4 grid with pits, gold, and Wumpus.
- Actuators : Move forward, turn left, turn right, grab, shoot, climb.
- Sensors : Perceive stench (Wumpus in an adjacent room), & breeze (Pit in an adjacent room), bump (if agent runs into a wall), a scream (if Wumpus is killed), glitter (gold is found).

Percept Sequence Generation :-

i) Agent starts at (1,1), sensing its surroundings.
ii) If stench is detected, Wumpus is in an adjacent room.
iii) If breeze is detected, a pit is in an adjacent room.
iv) The agent infers safe paths and navigates toward the gold while avoiding hazards.

Q18 Solve the following crypto-arithmetic problem: SEND + MORE = MONEY.

Ans

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

→ As M is the carry, it's value can only be 1.
∴ M=1.

→ Now, S+M=O. As M=1, S+1=O. So, S is a number which creates a carry when added with 1. Thus, S=9, O=0.

→ Now, E+O=N. But, O=0. Thus, E=N. But, this is not possible. Thus, we infer that a carry is involved.
Thus, 1+E=N. —(i)

→ Next, N+R=E —(ii)

We solve (i), (ii), we get E=5, N=6.

→ Now, from above, N should be 9 but it is already assigned. Thus, R=R and a 1 comes from a carry.

→ Thus, D+E=y should generate a carry. But, E=5.
∴ D+5=y and a carry is generated.

→ Thus, D is greater than 4. But 5, 6, 8 and 9 are taken already. Thus, D=7 and y=2.

Thus, solution:
M=1, S=9, O=0, E=5, N=6, R=8, D=7, y=2.

In the given form:-

```
  9 5 6 7
  1 0 8 5
-------
1 0 6 5 2
```

Q19] 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.

Ans] i] FOL :-

i] $\forall x$ (Graduating (x) $\rightarrow$ Happy (x))

ii] $\forall x$ (Happy (x) $\rightarrow$ Smiling (x))

iii] $\exists x$ (Graduating (x))

2] Converting above to clauses form :-

i] - Graduating (x) v Happy (x)

ii] - Happy (x) v Smiling (x)

iii] Graduating (A)

3. Resolution :-

→ We have Graduating (A). From Graduating (A) and - Graduating (x) v Happy (x), we infer :-

((Graduating (A)), (- Graduating (x) v Happy (x))) $\Rightarrow$ Happy $\left(\frac{A}{x}\right)$

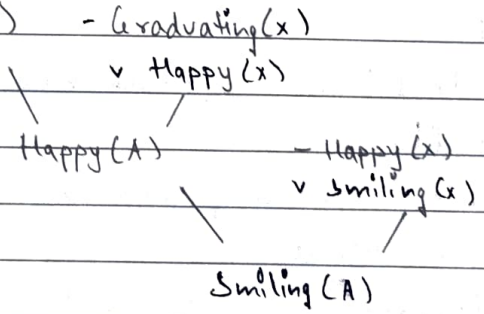
This is because an empty clause is created for Graduating (A).

→ We have Happy (A) and - Happy (x) v Smiling (x). From this, we infer :-

((Happy (A)), (- Happy (x) v Smiling (x))) $\Rightarrow$ Smiling (A).

Thus, someone is smiling.

Resolution Tree :-



Q20) Explain Modus ponen with suitable example.

Ans) Modus Ponens is a fundamental rule in propositional logic that states:

$P \rightarrow Q$ (if P then Q),

and P is true, then Q must be true.

For ex:-

→ Premise 1: If it rains, the ground will be wet ($P \rightarrow Q$)

→ Premise 2: It is raining (P is true)

→ Conclusion: The ground is wet (Q is true).

Q21) Explain Forward Chaining and Backward Chaining algorithm with an example.

Ans) • Forward ~~Electin~~ Chaining: It is a method of inference that starts with known facts and applies rules to derive new conclusions. The process continues until the goal is reached or no new facts can be inferred.

For ~~ex~~ ex: Diagnosing a disease:-

1. Given facts:-

i) Fact 1: If a person has a fever and cough, they may have the flu.

ii) Fact 2: John has a fever.

iii) Fact 3: John has a cough.

2. Rules:

i) Rule 1: If a person has a fever and cough, they have the flu.

ii) Rule 2: If a person has the flu, they should rest and drink fluids.

3. Inference using forward reasoning:-

i) Step 1: John has a fever and cough.

ii) Step 2: Apply Rule 1 → John has the flu.
iii) Step 3: Apply Rule 2 → John should rest and drink fluids.

Thus, conclusion: John should rest and drink fluids.

- Backward Chaining: It is an inference method that starts with the goal and works backward to check if the given facts support it.

For ex: Some example as Forward Chaining:

1. Facts and Rules same as that of Forward Chaining.
2. Goal: Prove that John has the flu.
3. Backward Chaining steps:-
   i) To prove John has the flu, check if he has a fever and a cough (from Rule 1)
   ii) Check if John has a fever (Fact 1: Yes).
   iii) Check if John has a cough (Fact 2: Yes).
   iv) Since both conditions are met, we can conclude that John has the flu.

Thus, conclusion: The hypothesis is proven (John has the flu).

Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

AIDS-I Assignment No: 2

Q.1: Use the following data set for question 1
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

Ans)
1. **Mean**:-

Mean = (Sum of all values) ÷ (Number of values)

- Sum = 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = 1611
- Number of values = 20

Mean = 1611 / 20 = 80.55

2. **Median:-**

First, sort the data into ascending order: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

There are 20 values (even number), so the median is the average of the 10th and 11th values:
10th value = 81, 11th value = 82

Median = (81 + 82) / 2 = 81.5

3. **Mode:-**

Mode = Most frequently occurring number(s)

From the sorted list:
76 appears **3 times**, more than any other value.

Mode = 76

4. **Interquartile Range (IQR):-**

Q1 = 25th percentile of the given values
= Median of the first half (first 10 numbers):
59, 64, 66, 70, 76, 76, 76, 78, 79, 81
Q1 = (5th + 6th) / 2 = (76 + 76) / 2 = **76**

Q3 = 75th percentile of the given values
= Median of the second half (last 10 numbers):
82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Q3 = (5th + 6th) / 2 = (88 + 90) / 2 = **89**

IQR = Q3 - Q1 = 89 - 76 = 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?
   a. Predictive analytic
   b. Descriptive analytic
3. From the three choices listed below, how would you describe each tool listed above? Why did you choose the answer?
   a. Supervised learning
   b. Unsupervised learning
   c. Reinforcement learning

Ans)
1) **Tool Comparison:-**

1. Machine Learning for Kids

   a. Target Audience:
   Primarily designed for young students (ages 8–16) and educators introducing machine learning concepts in schools.

   b. Use by Target Audience:
   - Students use visual interfaces to create models by training them on labeled data (e.g., text, images, numbers).
   - Integrates with Scratch and Python to build interactive ML-based projects (e.g., chatbots, games).
   - Teachers use it in classrooms to explain AI and machine learning fundamentals.

   c. Benefits:
   - User-friendly and tailored for education.
   - Visual and coding integration helps in real application.
   - No programming background needed for basic use.

   d. Drawbacks:
   - Limited complexity — not suitable for advanced ML tasks.
   - Mainly focuses on classification — fewer regression or clustering tasks.

2. Teachable Machine

a. Target Audience:
Aimed at students, educators, artists, hobbyists, and beginners in machine learning.

b. Use by Target Audience:
- Users upload or record images, audio, or pose data.
- Train models directly in the browser without writing code.
- Export models to use in websites or projects (e.g., with TensorFlow.js).

c. Benefits:
- Extremely easy to use, fast model training.
- Supports multiple input types (image, audio, pose).
- No sign-in or installation required.

d. Drawbacks:
- Limited customization or control over model parameters.
- Models may lack robustness for real-world deployment.

2) **Predictive Analytic vs Descriptive Analytic:-**
- Machine Learning for Kids: Predictive Analytic
  This is because it focuses on training models to predict outcomes (e.g., classifying emotions, sorting objects based on input), which is a form of predictive analysis.
- Teachable Machine: Predictive Analytic
  This is because it trains models to make real-time predictions based on user inputs (e.g., classifying audio, images, or poses).

3) **Type of Learning:-**
- Machine Learning for Kids: Supervised Learning
  This is because students provide labeled data (e.g., “happy” vs. “sad” texts) for training which is a clear example of supervised learning.
- Teachable Machine: Supervised Learning
  This is because users label data themselves (e.g., Image A = "Cat", Image B = "Dog"), which the model uses to learn which is a clear example of 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.

Ans) A significant example of misleading data visualization occurred in May 2020 when the Georgia Department of Public Health released a graph showing COVID-19 cases across five counties. The graph misrepresented the data in several critical ways:

1. **Non-Sequential Ordering of Dates:** The x-axis dates were presented out of sequence, which created an artificial sense of a consistent decline in cases. This misrepresentation led viewers to believe the situation was improving steadily when, in reality, the data was distorted by improper sequencing, making it appear as if the decline in cases was more consistent than it was. This error in chronological ordering significantly skewed the interpretation of the trends.
2. **Reordering of Counties:** The counties were reordered daily, making it difficult to track trends for individual counties over time. By constantly changing the order, the graph obscured meaningful comparisons and created a false narrative of a uniform decline across all counties, even though each county had its own unique trends. This also made it harder for viewers to understand the performance of each county separately.
3. **Lack of Contextual Information:** The graph lacked key details such as the dates of significant changes or explanations for data fluctuations. This omission gave the misleading impression that cases were declining significantly, downplaying the ongoing risks and complexities of the pandemic. Without these critical details, viewers were left with an incomplete picture.

This misleading visualization created a false perception of the situation, leading many viewers to believe that the pandemic was under control, which could have contributed to misguided confidence. This, in turn, may have influenced public decisions regarding safety measures like social distancing and mask-wearing. Misleading visualizations like this can have serious consequences for public health policy, as they may encourage premature relaxation of restrictions, which could lead to further outbreaks and unnecessary health risks.

This case raises important ethical concerns regarding data visualization, particularly in public health contexts. While data visualization is a powerful tool for communication, when used improperly, it can distort the interpretation of critical data, mislead the public, and undermine trust in the authorities. Ethical data visualization practices are essential

to ensure clarity, accuracy, and transparency. Such practices help the public and policymakers make well-informed decisions based on a truthful interpretation of the data, which ultimately leads to better public health outcomes.

In conclusion, the Georgia Department of Public Health's misleading graph highlights the dangers of poorly constructed visualizations and their potential to skew perceptions and influence public behavior. It underscores the need for accurate, transparent, and ethically designed data visualizations, particularly in public health crises, to avoid harmful consequences. Misleading visualizations undermine public trust and hinder informed decision-making in critical situations.

Source: Anderson Review. (2020). Graphic Presentation of COVID-19 Data Can Skew Perceptions of Risk. Retrieved from Anderson Review.

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

Dataset: [Pima Indians Diabetes Database]

Ans) Steps to carry out the above mentioned operations:

Step 1: Import required libraries and load the dataset:

```
Code: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDisplay
from imblearn.over_sampling import SMOTE
import warnings
warnings.filterwarnings("ignore")
data = pd.read_csv("diabetes.csv")
print(data.head())
print(data['Outcome'].value_counts())
```

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
Outcome
0    500
1    268
Name: count, dtype: int64
```

Step 2: Data Preprocessing:

Code: print(data.isnull().sum())
z_scores = np.abs(stats.zscore(data.drop('Outcome', axis=1)))
data = data[(z_scores < 3).all(axis=1)]
print("Data shape after outlier removal:", data.shape)
X = data.drop('Outcome', axis=1)
y = data['Outcome']
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
Data shape after outlier removal: (688, 9)
```

Step 3: Train/Val/Test Split (70/20/10) - Randomized:

Code: X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.3, random_state=42, stratify=y)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, random_state=42, stratify=y_temp)
print("Train size:", len(X_train), "Validation size:", len(X_val), "Test size:", len(X_test))

```
Train size: 481 Validation size: 138 Test size: 69
```

Step 4: Resolve Class Imbalance Using SMOTE:

Code: sm = SMOTE(random_state=42)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)
print("Balanced classes in training set:", np.bincount(y_train_res))

```
Balanced classes in training set: [322 322]
```

The code uses SMOTE to fix class imbalance by creating fake examples of the smaller class in the training data. It adjusts the training set to have an equal number of examples for each class and then shows the new class distribution to confirm the balance.

Step 5: Hyperparameter Tuning with GridSearchCV (SVM):

Code:

```
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
```

```
grid = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid.fit(np.vstack((X_train_res, X_val)), np.hstack((y_train_res, y_val)))
print("Best Parameters:", grid.best_params_)
print("Best Score:", grid.best_score_)
```

```
Best Parameters: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
Best Score: 0.8108198595459741
```

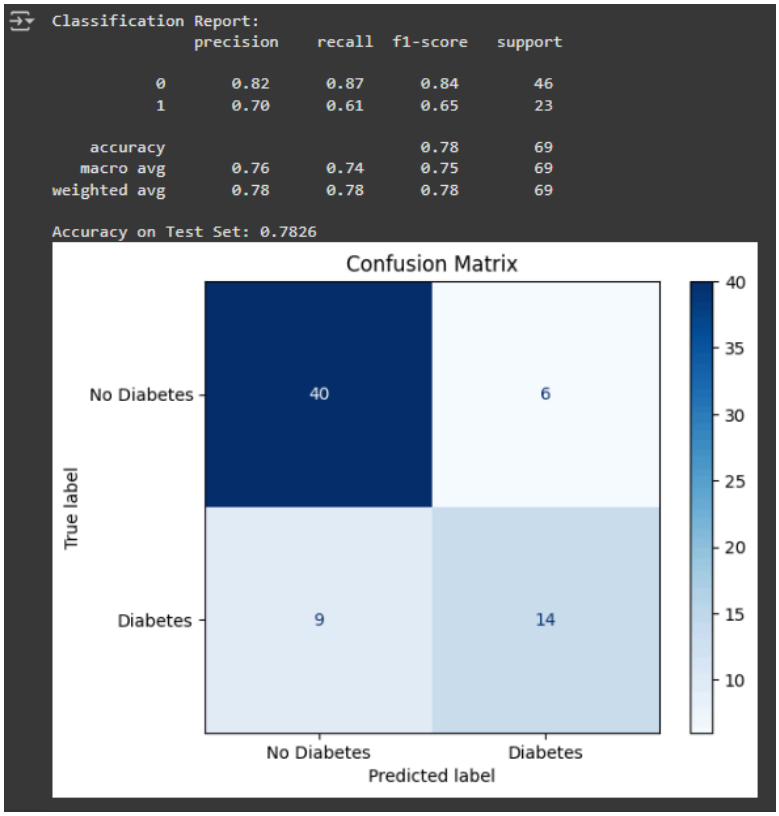
The code performs hyperparameter tuning for an SVM model using GridSearchCV. It tests different values for the C, kernel, and gamma parameters, using 5-fold cross-validation to find the best combination that maximizes accuracy. The best parameters and score are then printed.

Step 6: Train Final Model with Best Parameters:

Code: final_model = grid.best_estimator_
final_model.fit(np.vstack((X_train_res, X_val)), np.hstack((y_train_res, y_val)))

Step 7: Evaluate on Test Set:

Code: from sklearn.metrics import accuracy_score
y_pred = final_model.predict(X_test)
print("Classification Report:\n", classification_report(y_test, y_pred))
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy on Test Set: {accuracy:.4f}")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["No Diabetes", "Diabetes"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()



The classification model achieves an overall accuracy of 78.26% on the test set. It performs better at identifying non-diabetic cases (class 0) with a precision of 0.82 and recall of 0.87, compared to diabetic cases (class 1) where precision drops to 0.70 and recall to 0.61. The confusion matrix shows the model correctly classified 40 non-diabetic and 14 diabetic cases, but it misclassified 6 non-diabetics as diabetic and 9 diabetics as non-diabetic, indicating a need for improvement in detecting diabetes cases.

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

Dataset: https://github.com/Sutanoy/Public-Regression-Datasets/blob/main/Heart.csv

Ans) Steps to carry out the above mentioned operations:

Step 1: Import libraries and load the dataset:

Code:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("Heart.csv")  # replace with path if needed
df.head()
```

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slope	Ca	Thal	AHD
0	1	63	1	typical	145	233	1	2	150	0	2.3	3	0.0	fixed	No
1	2	67	1	asymptomatic	160	286	0	2	108	1	1.5	2	3.0	normal	Yes
2	3	67	1	asymptomatic	120	229	0	2	129	1	2.6	2	2.0	reversable	Yes
3	4	37	1	nonanginal	130	250	0	0	187	0	3.5	3	0.0	normal	No
4	5	41	0	nontypical	130	204	0	2	172	0	1.4	1	0.0	normal	No

Step 2: OOP Approach – Define a Regression Pipeline Class:

Code:
```
class HeartRegressionModel:
    def __init__(self, data):
        self.df = data.copy()
        self.model = None
        self.X_train = None
        self.X_test = None
```

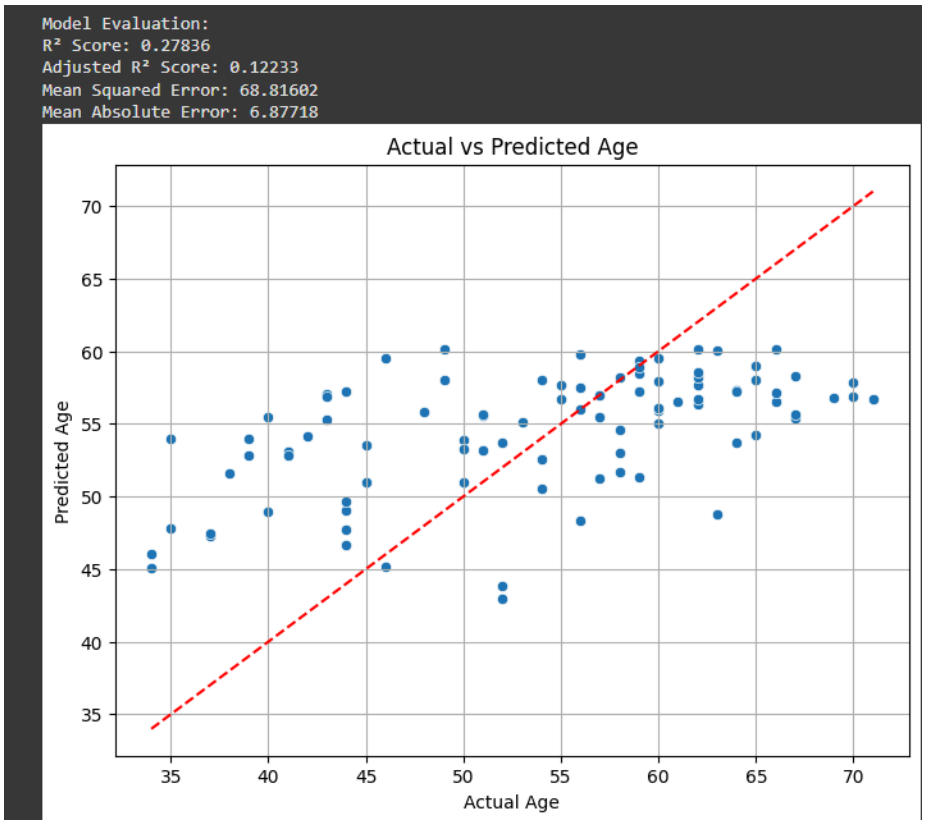
```
    self.y_train = None
    self.y_test = None
    self.y_pred = None
  def preprocess(self):
   if 'Unnamed: 0' in self.df.columns:
    self.df.drop(columns=['Unnamed: 0'], inplace=True)
   self.df = pd.get_dummies(self.df, drop_first=True)
   self.X = self.df.drop(columns=['Age'])  # Predicting 'age'
   self.y = self.df['Age']
   scaler = StandardScaler()
   self.X = scaler.fit_transform(self.X)
  def split_data(self):
    self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(
        self.X, self.y, test_size=0.3, random_state=None
    )
  def tune_model(self):
   model = XGBRegressor(objective='reg:squarederror', random_state=42)
   params = {
      'n_estimators': [100, 200],
      'max_depth': [3, 5, 7],
      'learning_rate': [0.01, 0.1, 0.2],
      'subsample': [0.8, 1],
     'colsample_bytree': [0.8, 1]
}
   grid = GridSearchCV(model, params, cv=5, scoring='r2', n_jobs=-1)
   grid.fit(self.X_train, self.y_train)
   self.model = grid.best_estimator_
   print(" Best Parameters:", grid.best_params_)
  def train_model(self):
    self.model.fit(self.X_train, self.y_train)
  def evaluate(self):
    self.y_pred = self.model.predict(self.X_test)
    r2 = r2_score(self.y_test, self.y_pred)
    n = len(self.y_test)
    p = self.X_test.shape[1]
    adjusted_r2 = 1 - (1 - r2) * (n - 1) / (n - p - 1)
    print("\nModel Evaluation:")
    print(f"R² Score: {r2:.5f}")
    print(f"Adjusted R² Score: {adjusted_r2:.5f}")
    print(f"Mean Squared Error: {mean_squared_error(self.y_test, self.y_pred):.5f}")
    print(f"Mean Absolute Error: {mean_absolute_error(self.y_test, self.y_pred):.5f}")
    return adjusted_r2
  def visualize_predictions(self):
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=self.y_test, y=self.y_pred)
    plt.xlabel("Actual Age")
    plt.ylabel("Predicted Age")
    plt.title("Actual vs Predicted Age")
```

```
        plt.plot([self.y_test.min(), self.y_test.max()],
                 [self.y_test.min(), self.y_test.max()],
                 color='red', linestyle='--')
        plt.grid(True)
        plt.show()
```

The HeartRegressionModel class performs regression on the heart dataset to predict the target variable (Age). It includes methods for preprocessing the data (handling categorical features and scaling), splitting it into training and test sets, tuning hyperparameters using XGBRegressor with GridSearchCV, training the model, and evaluating its performance (including R^2, adjusted R^2, MSE, and MAE). Additionally, it visualizes the model's predictions against the actual values using a scatter plot.

Step 3: Run the Complete Pipeline:

Code: heart_model = HeartRegressionModel(df)
heart_model.preprocess()
heart_model.split_data()
heart_model.tune_model()
heart_model.train_model()
adjusted_r2 = heart_model.evaluate()
heart_model.visualize_predictions()



The regression model performs poorly, with an R^2 score of 0.28 and an adjusted R^2 of 0.12, suggesting it explains little variance in the data. The mean absolute error of 6.88 indicates significant deviations between actual and predicted ages. The scatter plot shows that predictions are biased toward the mean, often underestimating higher ages and overestimating lower ones.

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).

Ans)

- **Key Features of the Wine Quality Dataset**:

1. Fixed Acidity – Tartaric acid; affects taste and stability.
2. Volatile Acidity – High levels give vinegar taste; lowers quality.
3. Citric Acid – Adds freshness; improves quality in moderation.
4. Residual Sugar – Impacts sweetness; minimal effect on quality.
5. Chlorides – Salt content; too much is bitter.
6. Free & Total Sulfur Dioxide – Preserves wine; excess affects taste.
7. Density – Related to sugar/alcohol; useful with other features.
8. pH – Acidity level; extremes can reduce quality.
9. Sulphates – Antimicrobial; moderate levels improve quality.
10. Alcohol – Higher alcohol often means better quality.
11. Quality – Target variable (score 0–10) from human tasters.

- **Importance of Features in Predicting Quality**:

a. Highly Influential: Alcohol, volatile acidity, sulphates
b. Moderately Influential: Citric acid, density, pH
c. Low Influence: Residual sugar, chlorides, total sulfur dioxide

Importance is determined using feature correlation, model coefficients (like in logistic regression), or feature importance from tree-based models.

- **Handling Missing Data During Feature Engineering:**

Step 1: Detect Missing Data

- Use .isnull().sum() or .info() to identify missing values.

Step 2: Strategy Based on Data Type & Distribution:

- For numerical columns:
    - Impute with mean if data is normally distributed.
    - Use median for skewed distributions (robust to outliers).
    - In some models, missing values are imputed using KNN Imputer or IterativeImputer for better estimation.
- For outlier-influenced features:
    - Median is usually preferred to reduce distortion.

Step 3: Advanced Imputation (if needed):

- Use KNN imputation to estimate values based on the nearest neighbors.

- Use regression imputation where a feature is predicted using a model trained on the other features.

- **Advantages and Disadvantages of Imputation Techniques:**

Technique	Advantages	Disadvantages
Mean Imputation	Simple and fast	Affected by outliers; reduces data variance
Median Imputation	Robust to outliers	Can still distort the distribution
Mode Imputation	Good for categorical data	Not suitable for continuous variables
KNN Imputer	Maintains feature relationships	Computationally expensive; sensitive to outliers
Regression Imputer	Captures complex relationships	Assumes linearity; can overfit
Drop Missing Rows	Simplifies dataset	Risk of losing valuable data and reducing model performance