Name : Advik Ashok Hegde **Div : D15C** **Roll No : 15**

Experiment No. - 1 :

Aim :

- 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

Problem Statement : Introduction to Data science and Data preparation using Pandas steps.

Introduction :

Q.What is Data Science and Data Preparation ?

1. Data Science

Data Science is the process of extracting insights from data using statistical and computational techniques. It involves:

- **Data Processing** – Collecting, cleaning, and organizing raw data.
- **Analysis & Modeling** – Applying machine learning and statistical methods to identify patterns.
- **Decision Making** – Using data-driven insights to solve real-world problems.

2. Data Preparation

Data Preparation ensures data quality for analysis and modeling by refining raw data. It includes:

- **Cleaning** – Handling missing values, duplicates, and inconsistencies.
- **Transformation** – Normalizing, scaling, and encoding data for better model performance.
- **Feature Selection** – Choosing relevant data attributes to improve accuracy.

Dataset Used : **Car features and their corresponding MSRP.**

The dataset titled "Car Features and MSRP" provides detailed information on various car attributes and their corresponding Manufacturer's Suggested Retail Prices (MSRP). This dataset is valuable for analyzing how different features influence car pricing

.

Key Features of the Dataset:

- **Make and Model:** Identifies the manufacturer and specific model of each car.
- **Year:** Indicates the production year of the vehicle.
- **Engine Type:** Details about the engine, such as displacement and configuration.
- **Fuel Type:** Indicates the kind of fuel the car uses, such as gasoline, diesel, or electric.
- **MSRP:** Lists the Manufacturer's Suggested Retail Price for each vehicle.

This dataset is structured to facilitate analysis of how these features correlate with car pricing, making it a valuable resource for studies in automotive market trends and pricing strategies.

1.Loading Data into Pandas

```
import pandas as pd
df = pd.read_csv('Car_Features.csv')
df.info()
df.describe()
```

	Year	Engine HP	Engine Cylinders	Number of Doors	highway MPG	city mpg	Popularity	MSRP
count	11914.000000	11845.00000	11884.000000	11908.000000	11914.000000	11914.000000	11914.000000	1.191400e+04
mean	2010.384338	249.38607	5.628829	3.436093	26.637485	19.733255	1554.911197	4.059474e+04
std	7.579740	109.19187	1.780559	0.881315	8.863001	8.987798	1441.855347	6.010910e+04
min	1990.000000	55.00000	0.000000	2.000000	12.000000	7.000000	2.000000	2.000000e+03
25%	2007.000000	170.00000	4.000000	2.000000	22.000000	16.000000	549.000000	2.100000e+04
50%	2015.000000	227.00000	6.000000	4.000000	26.000000	18.000000	1385.000000	2.999500e+04
75%	2016.000000	300.00000	6.000000	4.000000	30.000000	22.000000	2009.000000	4.223125e+04
max	2017.000000	1001.00000	16.000000	4.000000	354.000000	137.000000	5657.000000	2.065902e+06

All of the data from the dataset file of ‘Car_Features.csv’ was loaded onto pandas and the successful loading of the file was verified by using the df.describe() command that displays he data within the file.

2.Description of the Dataset

```
import pandas as pd
df = pd.read_csv('Car_Features.csv')
df.info()
df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Make               11914 non-null  object 
 1   Model              11914 non-null  object 
 2   Year               11914 non-null  int64  
 3   Engine Fuel Type   11911 non-null  object 
 4   Engine HP          11845 non-null  float64
 5   Engine Cylinders   11884 non-null  float64
 6   Transmission Type  11914 non-null  object 
 7   Driven_Wheels      11914 non-null  object 
 8   Number of Doors    11908 non-null  float64
 9   Market Category    8172 non-null   object 
 10  Vehicle Size       11914 non-null  object 
 11  Vehicle Style      11914 non-null  object 
 12  highway MPG        11914 non-null  int64  
 13  city mpg           11914 non-null  int64  
 14  Popularity         11914 non-null  int64  
 15  MSRP               11914 non-null  int64  
dtypes: float64(3), int64(5), object(8)
memory usage: 1.5+ MB
```

The df.describe() command is used to obtain a description of the data inside of the dataset.

3.Drop columns that are not useful.(**Dropping Column "Popularity"**)

Dropping Column "Popularity"

```
[ ]  import pandas as pd
     df = pd.read_csv('Car_Features.csv')
     df = df.drop('Popularity', axis=1)
     df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Make               11914 non-null  object
 1   Model              11914 non-null  object
 2   Year               11914 non-null  int64
 3   Engine Fuel Type   11911 non-null  object
 4   Engine HP          11845 non-null  float64
 5   Engine Cylinders   11884 non-null  float64
 6   Transmission Type  11914 non-null  object
 7   Driven_Wheels      11914 non-null  object
 8   Number of Doors    11908 non-null  float64
 9   Market Category    8172 non-null   object
 10  Vehicle Size       11914 non-null  object
 11  Vehicle Style      11914 non-null  object
 12  highway MPG        11914 non-null  int64
 13  city mpg           11914 non-null  int64
 14  MSRP               11914 non-null  int64
dtypes: float64(3), int64(4), object(8)
memory usage: 1.4+ MB
```

The column of 'Popularity' which is not really all that useful from the perspective of analysis of the data is removed from the dataset as a part of its processing phase.

4.Dropping rows with missing values

Dropping rows with Missing values.

```
import pandas as pd
df = pd.read_csv('Car_Features.csv')
df = df.dropna()
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8084 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Make               8084 non-null   object 
 1   Model              8084 non-null   object 
 2   Year               8084 non-null   int64  
 3   Engine Fuel Type   8084 non-null   object 
 4   Engine HP          8084 non-null   float64
 5   Engine Cylinders   8084 non-null   float64
 6   Transmission Type  8084 non-null   object 
 7   Driven_Wheels      8084 non-null   object 
 8   Number of Doors    8084 non-null   float64
 9   Market Category    8084 non-null   object 
 10  Vehicle Size       8084 non-null   object 
 11  Vehicle Style      8084 non-null   object 
 12  highway MPG        8084 non-null   int64  
 13  city mpg           8084 non-null   int64  
 14  Popularity         8084 non-null   int64  
 15  MSRP               8084 non-null   int64  
dtypes: float64(3), int64(5), object(8)
memory usage: 1.0+ MB
None
```

`dropna()` removes rows or columns containing missing (NaN) values cleaning the dataset of all of the missing values that do not exist which provides us with more consistent data values and accurate analysis.

5.Taking care of missing values by replacing it with Mean

Taking care of misssing values by putting Mean

```
import pandas as pd
df = pd.read_csv('Car_Features.csv')
df.fillna(df.mean(numeric_only=True), inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11914 entries, 0 to 11913
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Make               11914 non-null  object 
 1   Model              11914 non-null  object 
 2   Year               11914 non-null  int64  
 3   Engine Fuel Type   11911 non-null  object 
 4   Engine HP          11914 non-null  float64
 5   Engine Cylinders   11914 non-null  float64
 6   Transmission Type  11914 non-null  object 
 7   Driven_Wheels      11914 non-null  object 
 8   Number of Doors    11914 non-null  float64
 9   Market Category    8172 non-null   object 
 10  Vehicle Size       11914 non-null  object 
 11  Vehicle Style      11914 non-null  object 
 12  highway MPG        11914 non-null  int64  
 13  city mpg           11914 non-null  int64  
 14  Popularity         11914 non-null  int64  
 15  MSRP               11914 non-null  int64  
dtypes: float64(3), int64(5), object(8)
memory usage: 1.5+ MB
```

All of the missing values are replaced by the mean of that corresponding column to get more accurate analysis and make sure that the data is consistent.

6.Creating Dummy variables for the Transmission type

```
import pandas as pd
df = pd.read_csv('Car_Features.csv')
transmission_dummies = pd.get_dummies(df['Transmission Type'])
df_with_dummies = pd.concat([df, transmission_dummies], axis=1)
df_with_dummies.info()
df_with_dummies.head(10)
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	...	Vehicle Style	highway MPG	city mpg	Popularity	MSRP	AUTOMATED_MANUAL	AUTOMATIC	DIRECT_DRIVE	MANUAL
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	...	Coupe	26	19	3916	46135	False	False	False	True
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	...	Convertible	28	19	3916	40650	False	False	False	True
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	...	Coupe	28	20	3916	36350	False	False	False	True
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	...	Coupe	28	18	3916	29450	False	False	False	True
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	...	Convertible	28	18	3916	34500	False	False	False	True
5	BMW	1 Series	2012	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	...	Coupe	28	18	3916	31200	False	False	False	True

Creating dummy variables for the transmission type converts categorical data into a numeric format for machine learning models. Using `pd.get_dummies(df['Transmission'])`, each unique transmission type (e.g., Automatic, Manual) becomes a separate column with binary values (0 or 1), allowing models to interpret the categorical feature effectively without introducing ordering bias.

7.Find out outliers

```python
import pandas as pd
df = pd.read_csv('Car_Features.csv')
column_to_check = 'MSRP'

Q1 = df[column_to_check].quantile(0.25)
Q3 = df[column_to_check].quantile(0.75)

IQR = Q3 - Q1

lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outliers = df[(df[column_to_check] < lower_bound) | (df[column_to_check] > upper_bound)]
print(f"Outliers in {column_to_check}:\n", outliers.head(10))
print("\nNumber of outliers:", outliers.shape[0])
```

```
Outliers in MSRP:
         Make Model  Year             Engine Fuel Type  Engine HP  \
294  Ferrari   360  2002  premium unleaded (required)      400.0
295  Ferrari   360  2002  premium unleaded (required)      400.0
296  Ferrari   360  2002  premium unleaded (required)      400.0
297  Ferrari   360  2002  premium unleaded (required)      400.0
298  Ferrari   360  2003  premium unleaded (required)      400.0

     Engine Cylinders Transmission Type     Driven_Wheels  Number of Doors  \
294               8.0            MANUAL  rear wheel drive              2.0
295               8.0            MANUAL  rear wheel drive              2.0
296               8.0  AUTOMATED_MANUAL  rear wheel drive              2.0
297               8.0  AUTOMATED_MANUAL  rear wheel drive              2.0
298               8.0            MANUAL  rear wheel drive              2.0

              Market Category Vehicle Size Vehicle Style  highway MPG  \
294  Exotic,High-Performance      Compact   Convertible           15
295  Exotic,High-Performance      Compact         Coupe           15
296  Exotic,High-Performance      Compact         Coupe           15
297  Exotic,High-Performance      Compact   Convertible           15
298  Exotic,High-Performance      Compact   Convertible           15

     city mpg  Popularity    MSRP
294        10        2774  160829
295        10        2774  140615
296        10        2774  150694
297        10        2774  170829
298        10        2774  165986

Number of outliers: 996
```

8.Standardization and normalization of columns

```
import pandas as pd
from sklearn.preprocessing import StandardScaler
df = pd.read_csv('Car_Features.csv')
column_to_standardize = "MSRP"
scaler = StandardScaler()
df[column_to_standardize + " Standardized"] = scaler.fit_transform(df[[column_to_standardize]])
print(df.head(10).to_string())
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible
5	BMW	1 Series	2012	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe
6	BMW	1 Series	2012	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible
7	BMW	1 Series	2012	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe
8	BMW	1 Series	2012	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible
9	BMW	1 Series	2013	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

df = pd.read_csv('Car_Features.csv')
column_to_normalize = "MSRP"
scaler = MinMaxScaler()
df[column_to_normalize + " Normalized"] = scaler.fit_transform(df[[column_to_normalize]])
print(df.head(10).to_string())
```

	Make	Model	Year	Engine Fuel Type	Engine HP	Engine Cylinders	Transmission Type	Driven_Wheels	Number of Doors	Market Category	Vehicle Size	Vehicle Style	hi
0	BMW	1 Series M	2011	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Factory Tuner,Luxury,High-Performance	Compact	Coupe	
1	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	
2	BMW	1 Series	2011	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	
3	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	
4	BMW	1 Series	2011	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	
5	BMW	1 Series	2012	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Coupe	
6	BMW	1 Series	2012	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,Performance	Compact	Convertible	
7	BMW	1 Series	2012	premium unleaded (required)	300.0	6.0	MANUAL	rear wheel drive	2.0	Luxury,High-Performance	Compact	Coupe	
8	BMW	1 Series	2012	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	
9	BMW	1 Series	2013	premium unleaded (required)	230.0	6.0	MANUAL	rear wheel drive	2.0	Luxury	Compact	Convertible	

Standardization – This process transforms numerical features to have a **mean of 0** and a **standard deviation of 1** using the **Z-score formula**:

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}$$

where **XXX** is the original value, **μ\muμ** is the mean, and **σ\sigmaσ** is the standard deviation. This method ensures that features with different units are comparable, making it useful for models like linear regression and SVM.

Normalization : his scales values between a fixed range, typically **[0,1]**, using **Min-Max scaling**:

$$X_{\text{normalized}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$$

where **XminX_{\text{min}}Xmin** and **XmaxX_{\text{max}}Xmax** are the minimum and maximum values of the feature. This helps models like neural networks that require inputs within a specific range.

Conclusion : Thus we have successfully applied all of the basic commands on out chosen dataset of Car Features and MSRP and have learned the basic process of modifying the data,cleaning it and preparing it for processing.

Name : Advik Ashok Hegde **Div : D15C** **Roll No : 15**

Experiment No. - 2 :

Aim :
Perform following data visualization and exploration on your selected dataset.
1. Create bar graph, contingency table using any 2 features.
2. Plot Scatter plot, box plot, Heatmap using seaborn.
3. Create histogram and normalized Histogram.
4. Describe what this graph and table indicates.
5. Handle outlier using box plot and Inter quartile range.

Problem Statement : Data Visualization/ Exploratory data Analysis using Matplotlib and Seaborn.

Introduction :

Data Visualization with Matplotlib

Matplotlib is a comprehensive Python library for creating static, animated, and interactive visualizations. It provides control over plot elements like axes, labels, and colors. Key features include:

- **Line, bar, scatter, and histogram plots**
- **Customizable visual styles** (e.g., figure size, colors, titles)
- **Subplot capabilities** for multi-plot grids
  Matplotlib is ideal for creating basic and detailed visual representations of data.

Exploratory Data Analysis (EDA) with Seaborn

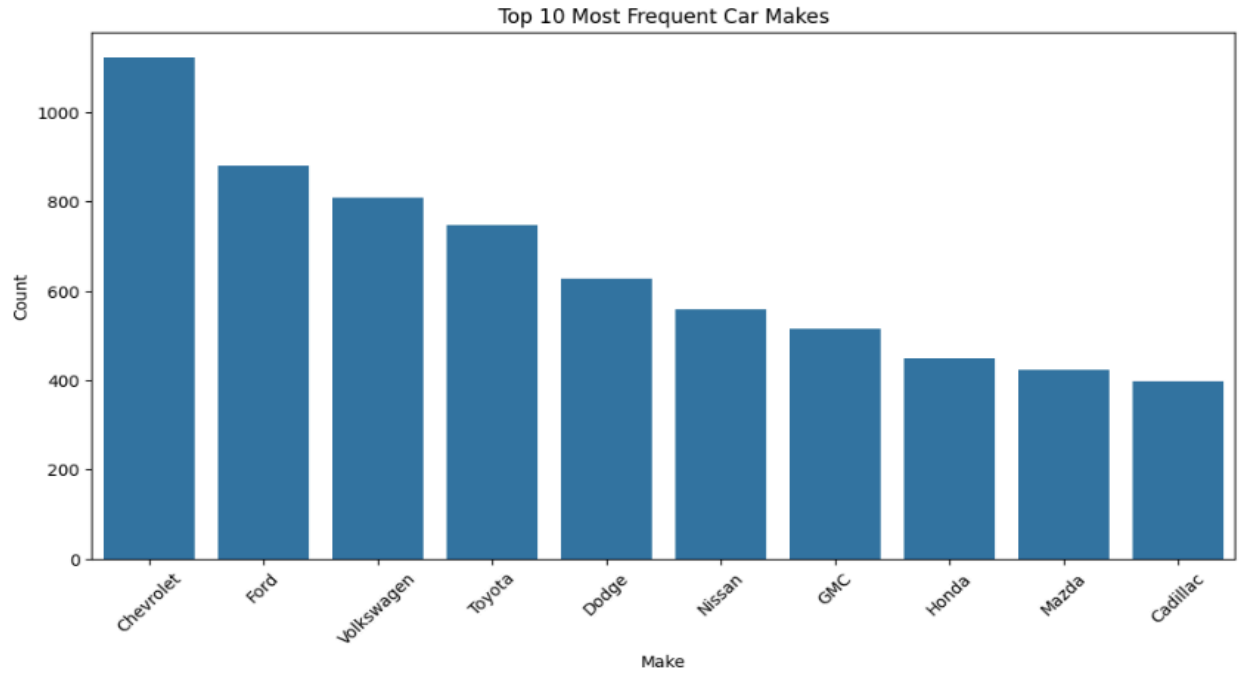
Seaborn is built on top of Matplotlib and provides a high-level interface for creating visually appealing and informative statistical graphics. Key capabilities include:

- **Visualizing distributions** (e.g., histograms, box plots, violin plots)
- **Correlation and relationships** (e.g., scatter plots, pair plots)
- **Easy integration with Pandas DataFrames**
  Seaborn simplifies the process of exploring data relationships and distributions, making it a powerful tool for EDA.

1.Bar Graph (top 10 car makes vs count of vehicles for make)

This graph provides a clear representation of the distribution of car brands within the dataset. By plotting the **Make** column's value counts, we can see which car brands are most prevalent in the dataset. The x-axis shows the top 10 car makes, while the y-axis indicates the count of vehicles for each make.

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

df = pd.read_csv("aids_2.csv")

plt.figure(figsize=(12, 6))
sns.barplot(x=df['Make'].value_counts().index[:10], y=df['Make'].value_counts().values[:10])
plt.xticks(rotation=45)
plt.xlabel("Make")
plt.ylabel("Count")
plt.title("Top 10 Most Frequent Car Makes")
plt.show()

contingency_table = pd.crosstab(df['Transmission Type'], df['Driven_Wheels'])
print("Contingency Table:\n", contingency_table)
```



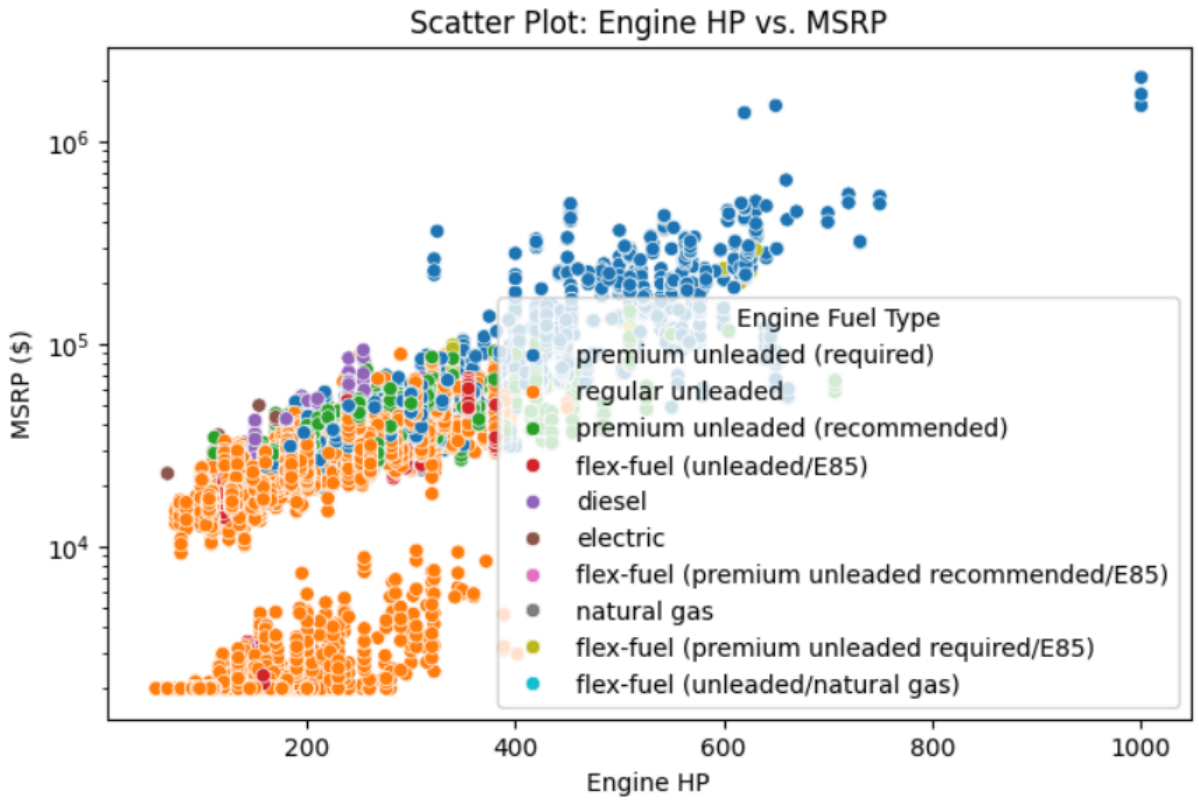

```
Contingency Table:
 Driven_Wheels      all wheel drive  four wheel drive  front wheel drive  \
Transmission Type
AUTOMATED_MANUAL               198                 0                304
AUTOMATIC                     1940              1056               3056
DIRECT_DRIVE                    11                 0                 43
MANUAL                         204               345               1380
UNKNOWN                          0                 2                  4

Driven_Wheels      rear wheel drive
Transmission Type
AUTOMATED_MANUAL                124
AUTOMATIC                      2214
DIRECT_DRIVE                     14
MANUAL                         1006
UNKNOWN                          13
```

2.Scatter Plot,box plot, Heatmap using seaborn.

```
## 2. Scatter Plot, Box Plot, Heatmap
# Scatter Plot: Engine HP vs. MSRP
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df['Engine HP'], y=df['MSRP'], hue=df['Engine Fuel Type'])
plt.xlabel("Engine HP")
plt.ylabel("MSRP ($)")
plt.title("Scatter Plot: Engine HP vs. MSRP")
plt.yscale('log')  # Log scale to handle large MSRP values
plt.show()

# Box Plot: Highway MPG by Vehicle Size
plt.figure(figsize=(8, 5))
sns.boxplot(x=df['Vehicle Size'], y=df['highway MPG'])
plt.title("Box Plot: Highway MPG by Vehicle Size")
plt.show()

# Heatmap: Correlation between numerical features
numeric_df = df.select_dtypes(include=['number'])
plt.figure(figsize=(10, 6))
sns.heatmap(numeric_df.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Heatmap of Feature Correlations")
plt.show()
```

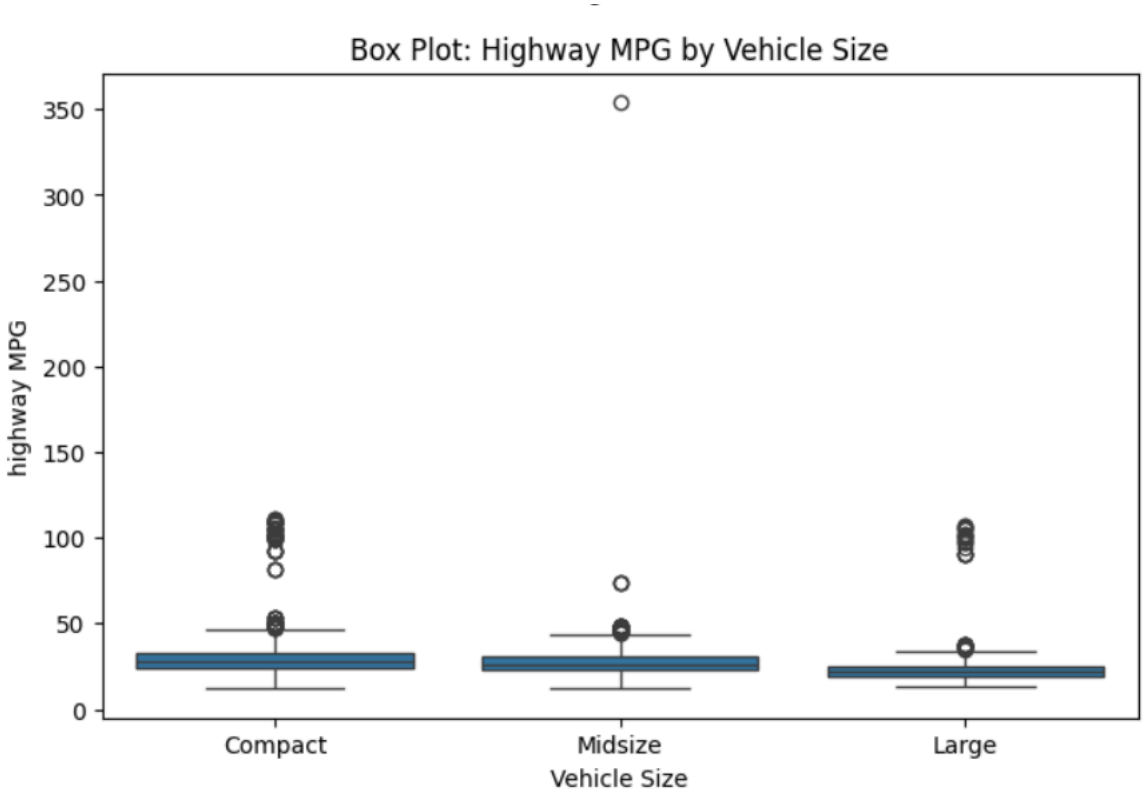
A **scatter plot** is a graph used to visualize the relationship between two continuous variables. It helps to identify correlations and trends in data. In this dataset, the scatter plot visualizes the relationship between Engine HP and MSRP, showing whether more powerful engines are associated with higher prices.

The plot uses color to represent Engine Fuel Type, adding another layer of insight. A logarithmic scale on the y-axis helps manage the wide range of MSRP values, making patterns clearer, especially in higher price ranges.



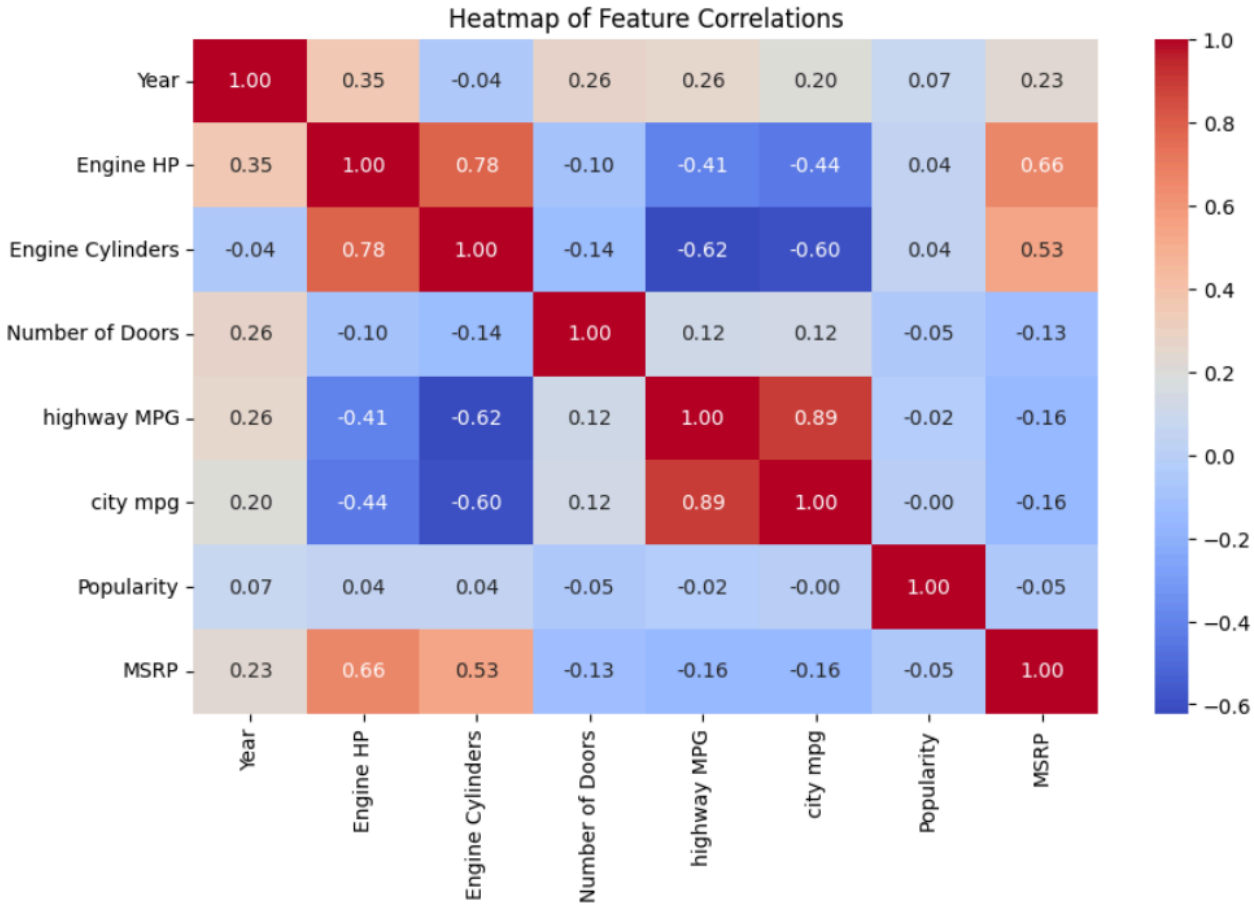
A **box plot** is used to show the distribution of data based on quartiles, highlighting the median, spread, and potential outliers.

In the dataset, the box plot visualizes the distribution of Highway MPG by Vehicle Size. It helps identify if larger vehicles tend to have lower fuel efficiency and points out any extreme outliers in highway MPG for different vehicle categories.



A **heatmap** is a graphical representation of data where values are represented by color, often used to show correlations between variables.

In the dataset, the heatmap visualizes the correlation between numerical features like Engine HP, MSRP, and Highway MPG. The color intensity indicates the strength of relationships, helping identify which variables are strongly correlated, such as whether Engine HP and MSRP are positively correlated.



3. Histogram and normalized Histogram.

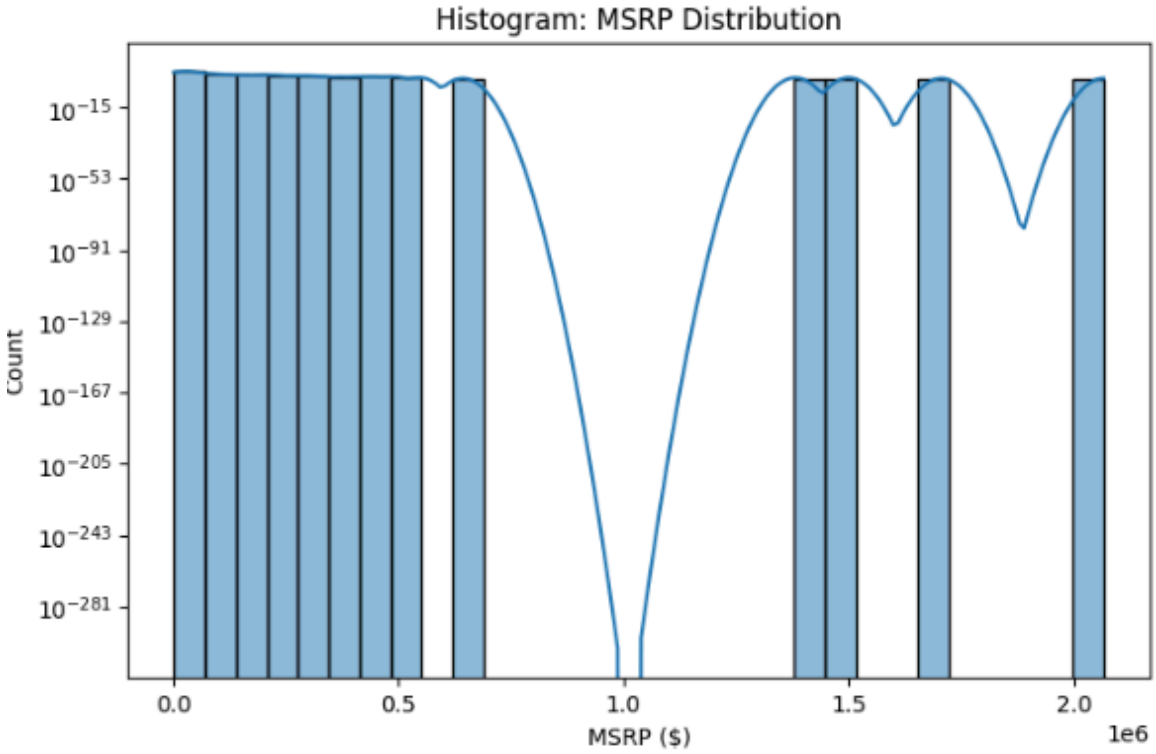
```
## 3. Histogram and Normalized Histogram
# Histogram: MSRP Distribution
plt.figure(figsize=(8, 5))
sns.histplot(df['MSRP'], bins=30, kde=True)
plt.xlabel("MSRP ($)")
plt.ylabel("Count")
plt.title("Histogram: MSRP Distribution")
plt.yscale('log')  # Log scale for better visualization
plt.show()

# Normalized Histogram
plt.figure(figsize=(8, 5))
sns.histplot(df['MSRP'], bins=30, kde=True, stat="density")
plt.xlabel("MSRP ($)")
plt.ylabel("Density")
plt.title("Normalized Histogram: MSRP")
plt.yscale('log')
plt.show()
```

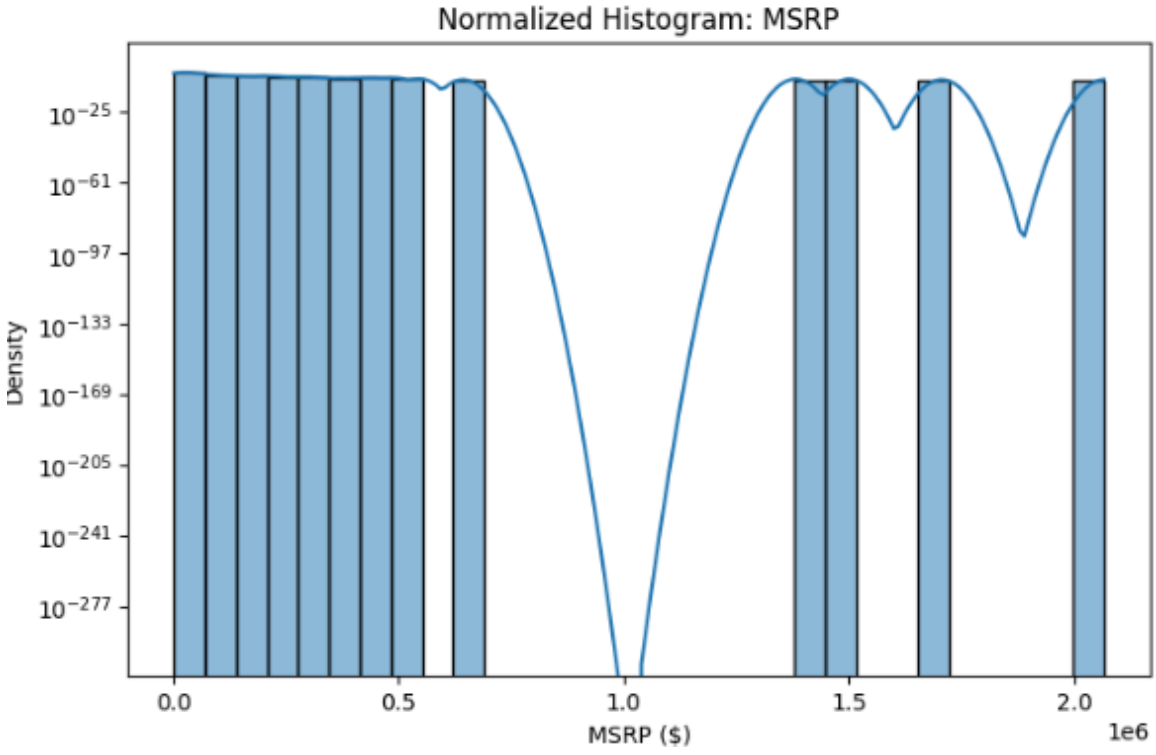
A histogram is a graphical representation that shows the distribution of a dataset by grouping data into bins. It is particularly useful for visualizing the frequency of values within a continuous range.

In this dataset, the histogram is used to display the distribution of MSRP (Manufacturer's Suggested Retail Price). The x-axis represents the range of MSRP values, while the y-axis shows the count of vehicles within each bin. The log scale on the y-axis helps visualize the distribution more effectively, especially with large values of MSRP, making it easier to observe the frequency of cars in different price ranges.

A normalized histogram represents the relative density (probability) rather than the raw count, showing how the distribution of data is spread out across the range of values.

In this dataset, the normalized histogram of MSRP helps to understand the probability distribution of car prices. The y-axis now represents density instead of count, providing a clearer view of the distribution's shape and allowing for easier comparison between different datasets or features. Like the histogram, the log scale is applied to the y-axis to help manage the skewed distribution of MSRP values.


Histogram: MSRP Distribution
Count
10^{-15}
10^{-53}
10^{-91}
10^{-129}
10^{-167}
10^{-205}
10^{-243}
10^{-281}
0.0
0.5
1.0
1.5
2.0
MSRP ($)
1e6


Normalized Histogram: MSRP
Density
10^{-25}
10^{-61}
10^{-97}
10^{-133}
10^{-169}
10^{-205}
10^{-241}
10^{-277}
0.0
0.5
1.0
1.5
2.0
MSRP ($)
1e6

4.Outlier using box plot and Inter quartile range.

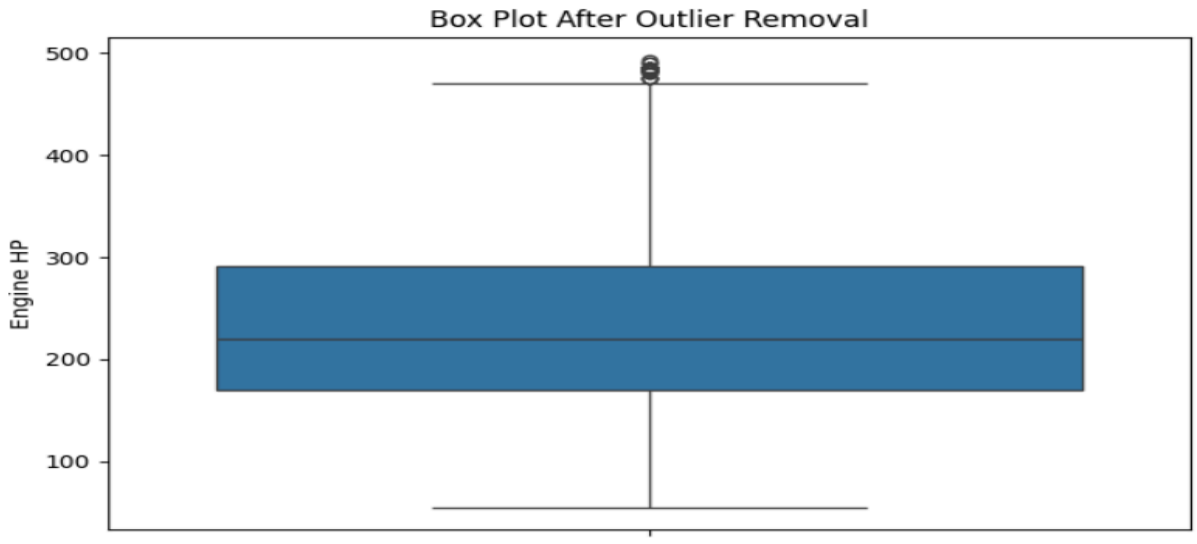
```
## 4. Handling Outliers using Box Plot and IQR
# Detecting Outliers in Engine HP using IQR
Q1 = df['Engine HP'].quantile(0.25)
Q3 = df['Engine HP'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Removing outliers
df_cleaned = df[(df['Engine HP'] >= lower_bound) & (df['Engine HP'] <= upper_bound)]

# Box Plot after Outlier Removal
plt.figure(figsize=(8, 5))
sns.boxplot(y=df_cleaned['Engine HP'])
plt.title("Box Plot After Outlier Removal")
plt.show()
```

A box plot is a useful tool for visualizing the distribution of a dataset and detecting outliers. It displays the median, quartiles, and potential outliers, providing a clear overview of the data's spread and central tendency.

In this dataset, the box plot is applied to the Engine HP feature, helping identify any extreme values or outliers in engine horsepower. The whiskers of the box plot indicate the range of data within the interquartile range (IQR), while points outside this range are considered outliers.

To handle outliers in Engine HP, the Interquartile Range (IQR) method is used. The first step is to calculate the IQR by subtracting the 25th percentile (Q1) from the 75th percentile (Q3). Outliers are typically defined as values outside the range of 1.5 times the IQR above Q3 or below Q1.

In this case, any Engine HP values outside the calculated bounds (lower and upper) are considered outliers and are removed from the dataset. The box plot is then regenerated to display the distribution of Engine HP after removing the outliers, allowing for a cleaner view of the data without extreme values skewing the results.



Conclusion :

In conclusion, using Matplotlib and Seaborn for Exploratory Data Analysis (EDA) helps uncover key insights in a dataset. Bar graphs reveal the distribution of categorical features, while contingency tables show relationships between them. Scatter plots identify correlations between continuous variables, and box plots detect outliers, which are handled using the IQR method. Heatmaps visualize feature correlations, and histograms provide insights into variable distributions. These tools are essential for understanding data patterns, ensuring clean datasets, and guiding further analysis or modeling.

Name : Advik Ashok Hegde **Div : D15C** **Roll No : 15**

Experiment No. - 3 :

Aim: Perform Data Modeling.
Problem Statement:

a. 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.
b. Use a bar graph and other relevant graph to confirm your proportions.
c. Identify the total number of records in the training data set.
d. Validate partition by performing a two-sample Z-test.

Introduction : Data modeling is a crucial step in machine learning and statistical analysis. It involves structuring data in a way that facilitates efficient processing, analysis, and prediction. One of the key aspects of data modeling is partitioning the dataset into training and testing subsets. This ensures that the model can be trained effectively while also being evaluated on unseen data to measure its performance.

The dataset that was used was first cleaned and pre-processed. All of the columns were checked for missing values and were replaced with mean for numerical data and mode for categorical data.One of the columns called “Market Category” had too many missing values so was completely dropped.

The column names were standardized with all of them being converted to lowercase and the blank spaces being replaced with “_”.All duplicates were removed and the datatypes of the columns was explicitly assigned to avoid any future problems.

Step 1: Data Partitioning

The dataset was divided into two subsets:

- **Training Set (75%)**: Used for model training.
- **Testing Set (25%)**: Used for evaluation and performance measurement.

Partitioning was done using a random sampling method to ensure an unbiased split.

```
from sklearn.model_selection import train_test_split
# Splitting dataset: 75% training, 25% testing
train_df, test_df = train_test_split(df, test_size=0.25, random_state=42)
# Verify the split
print("Training set size:", train_df.shape)
print("Testing set size:", test_df.shape)
```

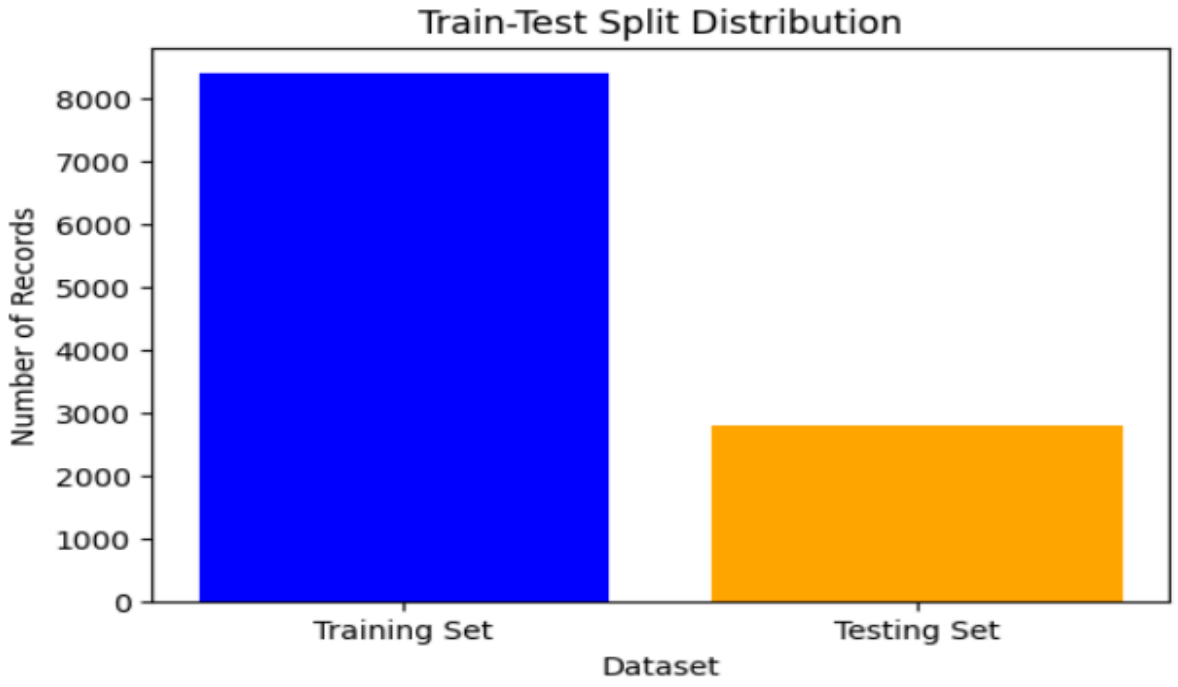
```
Training set size: (8395, 15)
Testing set size: (2799, 15)
```

Step 2: Visualizing the Data Split

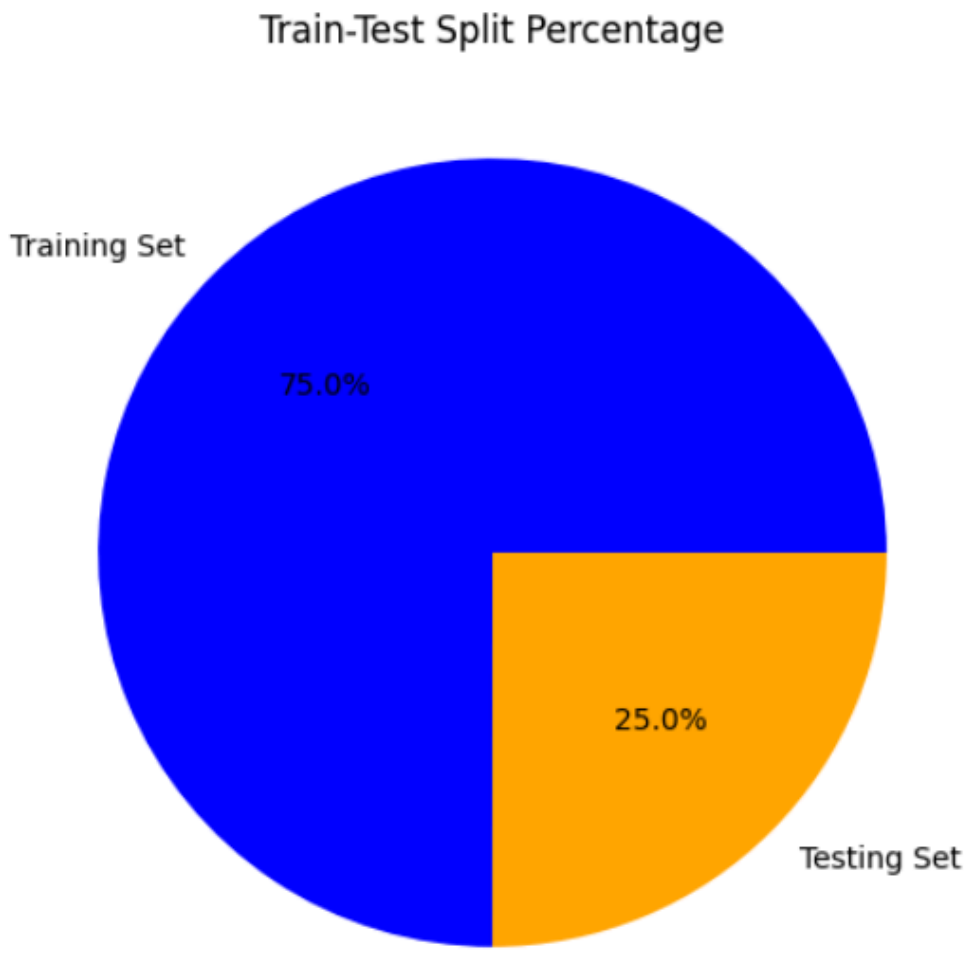
To confirm the partitioning proportions, the following visualizations were generated:

- **Bar Graph:** Representing the count of records in training and testing sets.
- **Pie Chart:** Showing the proportional distribution.

```
import matplotlib.pyplot as plt
# Bar graph for train-test split
labels = ['Training Set', 'Testing Set']
sizes = [len(train_df), len(test_df)]
plt.figure(figsize=(6, 4))
plt.bar(labels, sizes, color=['blue', 'orange'])
plt.xlabel("Dataset")
plt.ylabel("Number of Records")
plt.title("Train-Test Split Distribution")
plt.show()
```



```
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['blue', 'orange'])
plt.title("Train-Test Split Percentage")
plt.show()
```



Step 3: Identifying the Training Set Size

The total number of records in the dataset was determined, and 75% of these records were counted to confirm the training dataset size.

```
print("Total records in training dataset:", len(train_df))
```

```
Total records in training dataset: 8395
```

Step 4: Validating Partition with a Two-Sample Z-Test

A two-sample Z-test was performed to verify whether the training and testing subsets are statistically similar. The hypothesis for the test is:

- **Null Hypothesis (H0):** The mean of the training set is equal to the mean of the testing set.
- **Alternative Hypothesis (H1):** The means of the two sets are significantly different.

```
from scipy import stats

# Perform Z-test on MSRP column
train_mean = train_df["msrp"].mean()
test_mean = test_df["msrp"].mean()
train_std = train_df["msrp"].std()
test_std = test_df["msrp"].std()
n_train = len(train_df)
n_test = len(test_df)

# Compute Z-score
z_score = (train_mean - test_mean) / ((train_std**2 / n_train) + (test_std**2 / n_test))**0.5
p_value = stats.norm.sf(abs(z_score)) * 2  # Two-tailed test

print(f"Z-score: {z_score}")
print(f"P-value: {p_value}")

# Interpretation
if p_value > 0.05:
    print("No significant difference between training and testing sets (p > 0.05).")
else:
    print("Significant difference detected (p < 0.05). Data might not be well-distributed.")
```

```
Z-score: 1.9539196496714206
P-value: 0.05071072035766937
No significant difference between training and testing sets (p > 0.05).
```

A significance level of 0.05 was chosen, and the computed Z-score was compared against the critical Z-value to determine whether to reject the null hypothesis.

Conclusion Through data partitioning, visualization, and statistical validation, we ensured that our training and testing datasets were correctly proportioned and statistically similar. This process is essential for building reliable machine learning models that generalize well to unseen data.

Name : Advik Ashok Hegde **Div : D15C** **Roll No : 15**

Experiment No. - 4 :

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

Problem Statement: Perform the following Tests:Correlation Tests:
a) Pearson's Correlation Coefficient
b) Spearman's Rank Correlation
c) Kendall's Rank Correlation
d) Chi-Squared Test

Introduction Statistical hypothesis testing is a fundamental concept in data analysis and machine learning. It helps in determining relationships between variables and making data-driven decisions. In this experiment, we implement various statistical hypothesis tests using Python libraries such as SciPy and Scikit-learn.

For this experiment ,we are working with the same dataset that we obtained after cleaning in the last experiment named "cleaned_vehivles.csv".

Since all data cleaning and preprocessing operations are already performed ,we can directly start with performing the operations of Statistical Hypothesis Testing.

Pearson's Correlation Coefficient

- Measures the linear relationship between two continuous variables.
- Values range from -1 to 1, where 1 indicates a strong positive correlation, -1 indicates a strong negative correlation, and 0 indicates no correlation.

```
from scipy.stats import pearsonr

# Calculate Pearson correlation
pearson_corr, pearson_p = pearsonr(df["engine_hp"], df["msrp"])

print(f"Pearson Correlation Coefficient: {pearson_corr}")
print(f"P-value: {pearson_p}")
```

```
Pearson Correlation Coefficient: 0.6587937229804306
P-value: 0.0
```

A value of **0.6588** suggests a **moderately strong positive linear relationship** between the two variables.This means that as one variable increases, the other tends to increase as well.

The **p-value of 0.0** (or a very small value close to zero) suggests that the correlation is **highly statistically significant**.This means there is strong evidence to reject the null hypothesis (which assumes no correlation between the variables).

Spearman's Rank Correlation

- A non-parametric test that assesses the monotonic relationship between two variables.
- Useful for measuring correlations in ordinal or non-normally distributed data.

```
from scipy.stats import spearmanr

# Calculate Spearman correlation
spearman_corr, spearman_p = spearmanr(df["popularity"], df["city_mpg"])

print(f"Spearman's Rank Correlation: {spearman_corr}")
print(f"P-value: {spearman_p}")
```

```
Spearman's Rank Correlation: 0.027328076748436195
P-value: 0.003833171587034418
```

A value of **0.0273** is **very close to 0**, indicating an **extremely weak positive association** between the variables.Since **p < 0.05**, the correlation is **statistically significant**.This means that, despite being weak, the relationship is unlikely to be due to random chance.

Kendall's Rank Correlation

- Another non-parametric test that measures the strength of association between two variables.
- More robust for small datasets compared to Spearman's correlation.

```
from scipy.stats import kendalltau

# Calculate Kendall correlation
kendall_corr, kendall_p = kendalltau(df["number_of_doors"], df["highway_mpg"])

print(f"Kendall's Rank Correlation: {kendall_corr}")
print(f"P-value: {kendall_p}")
```

```
Kendall's Rank Correlation: 0.1119635631132
P-value: 6.856199111675777e-47
```

A value of **0.1119** indicates a **very weak positive association** between the variables.The **p-value is extremely small** (almost 0), meaning the correlation is **highly statistically significant**.This suggests that the observed weak correlation is **unlikely to be due to random chance**.

Chi-Squared Test

- Used to test the independence between categorical variables.
- Helps in determining whether distributions of categorical variables differ from one another.

```
from scipy.stats import chi2_contingency

# Create a contingency table
contingency_table = pd.crosstab(df["transmission_type"], df["driven_wheels"])

# Perform Chi-Squared test
chi2_stat, chi2_p, chi2_dof, chi2_expected = chi2_contingency(contingency_table)

print(f"Chi-Squared Statistic: {chi2_stat}")
print(f"P-value: {chi2_p}")
print(f"Degrees of Freedom: {chi2_dof}")
print(f"Expected Frequencies Table:\n {chi2_expected}")
```

```
Chi-Squared Statistic: 526.7198264496208
P-value: 4.5300427647599666e-105
Degrees of Freedom: 12
Expected Frequencies Table:
 [[1.14018581e+02 6.54569412e+01 2.14896373e+02 1.58628104e+02]
 [1.63460997e+03 9.38413436e+02 3.08082902e+03 2.27414758e+03]
 [1.40203681e+01 8.04895480e+00 2.64248705e+01 1.95058067e+01]
 [5.42876898e+02 3.11660264e+02 1.02318653e+03 7.55276309e+02]
 [2.47418260e+00 1.42040379e+00 4.66321244e+00 3.44220118e+00]]
```

Chi-Squared Statistic (526.72) : The Chi-Squared statistic measures the difference between observed and expected frequencies in a contingency table.A higher value indicates a greater deviation from expected frequencies, meaning the variables are likely dependent.

P-value (4.53×10^{-105}) : Since $p < 0.05$, we reject the null hypothesis, meaning there is a significant relationship between the categorical variables.

Degrees of Freedom (12) : More degrees of freedom generally indicate a more complex relationship being tested.

There is strong statistical evidence that the two categorical variables are not independent. The difference between observed and expected values is significant, meaning there is a meaningful association between them.

Conclusion Through these statistical tests, we evaluated relationships between variables and determined their significance. Pearson's test was used for linear relationships, while Spearman and Kendall's tests were used for rank-based correlations. The Chi-Squared test helped assess categorical variable dependencies. These analyses are essential in feature selection and model evaluation in machine learning.

Name : Advik Ashok Hegde **Div : D15C** **Roll No : 15**

Experiment No. - 5 :

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

Problem Statement:
a) Perform Logistic regression to find out relation between variables
b) Apply regression model technique to predict the data on the above dataset.

Introduction :
Logistic Regression : Logistic regression is a widely used statistical method for analyzing and modeling relationships between a dependent variable and one or more independent variables. Unlike linear regression, logistic regression is applied when the target variable is categorical, typically binary (0 or 1). It uses the logistic function to estimate probabilities, making it suitable for classification tasks.

In this implementation, we explore the relationship between variables using logistic regression to understand their influence on the target variable. Additionally, we apply this model to predict outcomes based on the dataset, leveraging techniques like model fitting, evaluation metrics, and performance assessment to validate the predictions.

For this dataset , we will be working on an AQI dataset that contains values of various pollutants contributing to the overall AQI value.

Implementation Process :

Step 1 **:** We start with performing pre-processing operations on the dataset by eliminating all of the duplicate values and missing values.

```
[ ]  # Preprocessing the Data
     import pandas as pd

     df = pd.read_csv("city_hour.csv")
     df = df.drop_duplicates()
     df = df.dropna()

     print("Number of rows after cleaning:", len(df))

     df.to_csv("aqidata.csv", index=False)
     df.info()
     df.head(10)
```

```
Number of rows after cleaning: 129277
<class 'pandas.core.frame.DataFrame'>
Index: 129277 entries, 50888 to 707867
Data columns (total 16 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   City        129277 non-null  object
 1   Datetime    129277 non-null  object
 2   PM2.5       129277 non-null  float64
 3   PM10        129277 non-null  float64
 4   NO          129277 non-null  float64
 5   NO2         129277 non-null  float64
 6   NOx         129277 non-null  float64
 7   NH3         129277 non-null  float64
 8   CO          129277 non-null  float64
 9   SO2         129277 non-null  float64
 10  O3          129277 non-null  float64
 11  Benzene     129277 non-null  float64
 12  Toluene     129277 non-null  float64
 13  Xylene      129277 non-null  float64
 14  AQI         129277 non-null  float64
 15  AQI_Bucket  129277 non-null  object
dtypes: float64(13), object(3)
```

Step 2 : The categorical values of the AQI_Bucket column that categorizes the AQI value are converted into binary target variables.

Mapping categories to binary labels:

- 0 for 'Good', 'Satisfactory', 'Moderate' (considered as good/acceptable air quality).
- 1 for 'Poor', 'Very Poor', 'Severe' (considered as bad/unacceptable air quality).

This creates a new column 'AQI_Binary' with binary values (0 or 1) for classification.

```python
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, confusion_matrix, classification_report
import seaborn as sns
import matplotlib.pyplot as plt

# Load the dataset
df = pd.read_csv("aqidata.csv")
print(df['AQI_Bucket'].unique())

df['AQI_Binary'] = df['AQI_Bucket'].map({
    'Good': 0,         # Define "Good" as 0
    'Satisfactory': 0,
    'Moderate': 0,
    'Poor': 1,         # Define "Bad" as 1
    'Very Poor': 1,
    'Severe': 1
})

df = df.dropna(subset=['AQI_Binary'])
df.info()
```

```
['Moderate' 'Poor' 'Very Poor' 'Satisfactory' 'Good' 'Severe']
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129277 entries, 0 to 129276
Data columns (total 17 columns):
```

Step 3 : The numerical values in the dataset are selected for the purposes of prediction, the dataset is then split into a 70 : 30 ratio, the features selected are standardized and then the logistic regression model is trained.

```
[ ]  # Select numerical features for prediction
     df_selected = df[['PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3', 'Benzene', 'Toluene', 'Xylene']]

     df_selected = df_selected.dropna()

     # Define X (features) and y (target)
     X = df_selected
     y = df.loc[df_selected.index, 'AQI_Binary']
```

```
[ ]  # Split into training and testing sets (70% train, 30% test)
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
[ ]  # Standardize features
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)
```

```
[ ]  # Train logistic regression model
     logreg = LogisticRegression()
     logreg.fit(X_train_scaled, y_train)
```

Step 4 : The model is then used to make predictions for the classification on the basis of AQI into a score of 0 for low AQI and 1 for high and dangerous levels of AQI and compare it with the actual values for the first 5 rows.

```
# Predict using trained model
y_pred = logreg.predict(X_test_scaled)
import pandas as pd
df = pd.DataFrame({"Actual": y_test, "Predicted": y_pred})
print(df.head(5))
```

```
        Actual  Predicted
22160        0          0
115833       0          0
124772       0          0
51509        0          0
5464         0          0
```

Step 5 : The model is then evaluated to check for accuracy along with the confusion matrix.

```
# Print evaluation metrics
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))
```

```
Accuracy: 0.9171823432343235
Confusion Matrix:
 [[29531   914]
 [ 2298  6041]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     30445
           1       0.87      0.72      0.79      8339

    accuracy                           0.92     38784
   macro avg       0.90      0.85      0.87     38784
weighted avg       0.92      0.92      0.91     38784
```

Linear Regression : Linear regression is a fundamental statistical technique used to model the relationship between a dependent variable and one or more independent variables. It assumes a linear relationship between the input features and the target variable, aiming to find the best-fitting line that minimizes the difference between predicted and actual values. Linear regression is widely used for predictive analysis in fields like finance, healthcare, and social sciences to forecast continuous outcomes like prices, scores, or measurements. The model's simplicity, interpretability, and efficiency make it a popular choice for regression tasks in data science.

For our dataset,we make use of linear regression to make predictions about the AQI values

Step 1 :
We prepare the dataset for applying linear regression to predict the Air Quality Index (AQI) based on various pollutants. It begins by cleaning column names, ensuring the presence of the target variable AQI, and verifying that all required feature columns (like PM2.5, NO2, CO, etc.) are present. Missing values in the target or feature columns are removed to maintain data consistency. The feature matrix (X_reg) and target variable (y_reg) are then finalized for modeling. This setup ensures accurate and reliable predictions in the subsequent regression analysis.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

df.columns = df.columns.str.strip()

if 'AQI' not in df.columns:
    raise KeyError("Column 'AQI' not found. Check dataset structure.")

y_reg = df['AQI']  # AQI is the target variable
feature_columns = ['PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3', 'Benzene', 'Toluene', 'Xylene']

missing_features = [col for col in feature_columns if col not in df.columns]
if missing_features:
    raise KeyError(f"Missing feature columns: {missing_features}")

X_reg = df[feature_columns]

df_selected = df.dropna(subset=['AQI'] + feature_columns)

X_reg = df_selected[feature_columns]
y_reg = df_selected['AQI']
```

Step 2 : The dataset is split into a 70 : 30 ratio for training and testing.The linear regression model is applied and then used to predict values and compare them with the actual values from the dataset.

```python
[ ]  # Split data into training (70%) and testing (30%) sets
    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)

    # Predict AQI values
    y_pred_reg = linreg.predict(X_test_reg_scaled)

    import pandas as pd
    df_results = pd.DataFrame({'Actual AQI': y_test_reg[:10], 'Predicted AQI': y_pred_reg[:10]})
```

	Actual AQI	Predicted AQI
22160	44.0	88.561925
115833	69.0	69.859090
124772	101.0	89.095485
51509	134.0	165.851721
5464	69.0	83.331704
2571	44.0	49.108907
125801	117.0	153.129551
101139	62.0	60.028689
75492	81.0	120.020093
93810	48.0	47.966254

Step 3 : The prepared model is evaluated for its performance on the basis of evaluation metrics like Mean Absolute Error,Mean Squared Error and the R2 Error.

Mean Absolute Error (MAE):
Measures the average absolute difference between actual and predicted values. It is simple and interpretable but doesn't emphasize large errors.

Mean Squared Error (MSE):
Computes the average of squared differences between actual and predicted values. Squaring emphasizes larger errors, making it sensitive to outliers.

R^2 Score (Coefficient of Determination):
Indicates the proportion of variance in the target variable explained by the model. Ranges from 0 to 1 (or negative for poor models). Higher R^2 suggests better fit.

```
[ ]  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))
```

```
Mean Absolute Error: 32.60513742490674
Mean Squared Error: 2270.1515445187156
R² Score: 0.7620454246256327
```

Low metric values in predicting **AQI** using linear regression are likely due to the complex, non-linear nature of air quality data, which a simple linear model may struggle to capture. Additionally, the dataset may lack crucial features like **weather conditions** or **traffic patterns**, leading to incomplete modeling.

Step 4 :

Root Mean Squared Error (RMSE):
The square root of MSE, maintaining the same unit as the target variable. It balances sensitivity to large errors with interpretability.

Baseline MSE:
The MSE is calculated by predicting the mean of the target variable for all inputs. It serves as a benchmark; if a model's MSE is significantly lower than the baseline, it's considered effective.

```
print(df['AQI'].min(), df['AQI'].max())

import numpy as np
rmse = np.sqrt(2270.15)  # Square root of MSE
print("RMSE:", rmse)
print(df['AQI'].std())  # Standard deviation of AQI

y_baseline = df['AQI'].mean()  # Predicting the mean AQI for all values
mse_baseline = np.mean((df['AQI'] - y_baseline) ** 2)
print("Baseline MSE:", mse_baseline)
```

```
18.0 760.0
RMSE: 47.64609113033303
97.01102086065393
Baseline MSE: 9411.065370185535
```

The minimum and maximum AQI values (18.0 and 760.0) show a wide range of air quality data. The Root Mean Squared Error (RMSE) of 47.65 is relatively small compared to the AQI range, suggesting that while the model makes errors, they are not extreme. However, the standard deviation of 97.01 indicates significant variability in AQI, implying that the model might struggle with accurately predicting all values. The Baseline MSE of 9411.07, based on predicting the mean AQI, is substantially higher than the model's MSE of 2270.15, indicating that the linear regression model does perform better than a naive prediction. Despite this, the high variability and potential data complexities suggest that a more advanced model may better capture the relationships within the dataset.

Conclusion :

In this analysis, we applied both **logistic regression** and **linear regression** to understand the relationships within the air quality dataset and make predictions. Logistic regression was effective in classifying air quality as **"Good"** or **"Bad"** based on pollutant levels, demonstrating the model's capability in binary classification tasks. Linear regression aimed to predict the **AQI** numerically but faced challenges due to the data's high variability and complex non-linear patterns. Despite achieving better-than-baseline performance, the model's low **R^2 score** and relatively high **error metrics** suggest that more advanced techniques or additional features may be necessary for improved predictions. The analysis showcased the practical applications of regression techniques in environmental data analysis while highlighting their limitations.

Name : Advik Ashok Hegde **Div : D15C** **Roll No : 15**

Experiment No. - 6 :

Problem Statement: Classification modelling
a. Choose a classifier for a classification problem.
b. Evaluate the performance of the classifier.
Perform Classification using the below 4 classifiers on the same dataset which you have used for

- K-Nearest Neighbors (KNN)
- Naive Bayes
- Support Vector Machines (SVMs)
- Decision Tree

Introduction :

Classification algorithms are crucial in predicting categorical outcomes and analyzing labeled data. In this experiment, we applied two supervised learning techniques — Naive Bayes and K-Nearest Neighbors (KNN) — to classify air quality based on the Air Quality Index (AQI) dataset.

Naive Bayes is a probabilistic classifier based on Bayes' theorem, known for its simplicity, efficiency, and effectiveness, especially with small datasets or when feature independence is assumed. On the other hand, KNN is a distance-based classifier that assigns a class label based on the majority of the nearest neighbors, making it intuitive yet sensitive to feature scaling and noise.

The primary objective was to classify AQI levels as either "Good" or "Bad" and evaluate the effectiveness of these algorithms. By comparing their accuracy, precision, recall, and F1-score, we gained insights into the suitability of these methods for air quality classification.

Implementation Process :

Step 1 : We perform a series of data pre-processing operations that involve calculating the missing percentage for each column ,using mean imputation for columns that have less than 20 % missing values,using iterative imputation methods for columns between 20 - 50% missing values and dropping the xylene column that had around 60 % missing values.

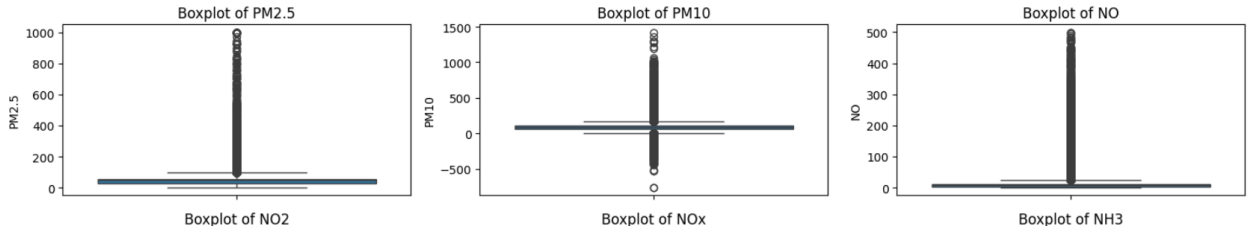
```python
from sklearn.impute import SimpleImputer
# Mean Imputation - <20% Missing
mean_imputer = SimpleImputer(strategy='mean')
for col in ['PM2.5', 'NO', 'NO2', 'NOx', 'CO']:
    df[col] = mean_imputer.fit_transform(df[[col]])
```

```python
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# Impute all columns with missing data (20%-50%) in one go
# More efficient in comparision to KNN
iter_imputer = IterativeImputer(max_iter=10, random_state=42)
columns_to_impute = ['PM10', 'NH3', 'Benzene', 'Toluene']
df[columns_to_impute] = iter_imputer.fit_transform(df[columns_to_impute])
print("Remaining Missing Values:\n", df[columns_to_impute].isnull().sum())
```

```python
[ ]  df.drop('Xylene', axis=1, inplace=True)
```

Step 2 : We verify that all of the missing values have been removed and then move to identifying and eliminating the outliers using boxplot analysis.

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

numeric_columns = ['PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3', 'Benzene', 'Toluene', 'AQI']

plt.figure(figsize=(15, 10))
for i, col in enumerate(numeric_columns, 1):
    plt.subplot(4, 3, i)
    sns.boxplot(df[col])
    plt.title(f'Boxplot of {col}')
plt.tight_layout()
plt.show()
```



Step 3 : Apply standardizing operations using the z-score method.

```
import numpy as np

# Define threshold for z-scores
threshold = 3

# Loop through each numeric column for efficient capping
for col in numeric_columns:
    mean_val = df[col].mean()
    std_dev = df[col].std()

    # Using np.clip for faster operations
    df[col] = np.clip(df[col], mean_val - threshold * std_dev, mean_val + threshold * std_dev)
```

Step 4 : We then encode categorical features in the dataset to prepare it for machine learning algorithms, as our model requires numerical data.

The LabelEncoder converts the categorical values of the AQI_Bucket column (like "Good", "Satisfactory", "Poor", etc.) into integer labels (0, 1, 2, etc.).

One-Hot Encoding creates binary columns for each unique value in the City column (like Delhi, Mumbai, etc.).

```
from sklearn.preprocessing import LabelEncoder

# Label Encoding for AQI_Bucket (ordinal)
le = LabelEncoder()
df['AQI_Bucket'] = le.fit_transform(df['AQI_Bucket'])

# One-Hot Encoding for City (nominal)
df = pd.get_dummies(df, columns=['City'], drop_first=True)

# Display the first few rows to verify
print("Encoded DataFrame (first 5 rows):")
print(df.head())
```

Step 5 : We then perform splitting of the dataset into a 70 : 30 split to proceed with the preparation and evaluation of the models.

```
[ ]  from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler

     # Dropping unnecessary columns
     X = df.drop(['Datetime', 'AQI'], axis=1)  # Features
     y = df['AQI']  # Target variable

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

     # Standardizing the data
     scaler = StandardScaler()
     X_train_scaled = scaler.fit_transform(X_train)
     X_test_scaled = scaler.transform(X_test)

     # Converting back to DataFrame for readability (Optional)
     X_train_scaled = pd.DataFrame(X_train_scaled, columns=X.columns)
     X_test_scaled = pd.DataFrame(X_test_scaled, columns=X.columns)

     print("Scaled Features (First 5 rows):")
     print(X_train_scaled.head())
```

Step 6 : The preparation of the Naive Bayes Classifier for application on to the dataset.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Handle missing values and format 'AQI_Bucket'
df['AQI_Bucket'] = df['AQI_Bucket'].fillna('Unknown').astype(str)

# Drop 'Datetime' from features
X = df.drop(columns=['AQI_Bucket', 'Datetime'])
y = df['AQI_Bucket']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardizing only numeric features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train.select_dtypes(include=[float, int]))
X_test_scaled = scaler.transform(X_test.select_dtypes(include=[float, int]))

# Naive Bayes Model
nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train)
nb_pred = nb_model.predict(X_test_scaled)

# Evaluation
print(" 🔹 Naive Bayes Evaluation 🔹 ")
print(f"Accuracy: {accuracy_score(y_test, nb_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, nb_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, nb_pred))
```

This classifier is then evaluated for accuracy and a classification report and a confusion matrix is prepared for the same.

```
 🔹 Naive Bayes Evaluation 🔹
Accuracy: 0.88
Classification Report:
              precision    recall  f1-score   support

         0.0       0.84      0.89      0.86      4601
         1.0       0.92      0.87      0.90     31435
         2.0       0.78      0.76      0.77      5237
         3.0       0.85      0.90      0.88     23026
         4.0       0.98      0.95      0.97      3461
         5.0       0.82      0.88      0.85      2859

    accuracy                           0.88     70619
   macro avg       0.87      0.88      0.87     70619
weighted avg       0.88      0.88      0.88     70619

Confusion Matrix:
 [[ 4096     3     0   502     0     0]
 [    0 27457   847  3131     0     0]
 [    0   862  4003     0     0   372]
 [  800  1485     0 20741     0     0]
 [    0     0     0     0  3296   165]
 [    0     0   286     0    69  2504]]
```

Step 7 : The preparation of the KNN (K-Nearest Neighbors) for application on to the dataset.

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# KNN Classifier
knn_model = KNeighborsClassifier(n_neighbors=5, weights='distance', n_jobs=-1)
knn_model.fit(X_train_scaled, y_train)
knn_pred = knn_model.predict(X_test_scaled)

print(" 🔹 Optimized KNN Evaluation 🔹 ")
print(f"Accuracy: {accuracy_score(y_test, knn_pred):.2f}")
print("Classification Report:\n", classification_report(y_test, knn_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, knn_pred))
```

This classifier is then evaluated for accuracy and a classification report and a confusion matrix is prepared for the same.

```
🔹 Optimized KNN Evaluation 🔹
Accuracy: 0.90
Classification Report:
              precision    recall  f1-score   support

         0.0       0.94      0.84      0.89      4601
         1.0       0.93      0.93      0.93     31435
         2.0       0.79      0.76      0.77      5237
         3.0       0.90      0.93      0.91     23026
         4.0       0.92      0.92      0.92      3461
         5.0       0.77      0.73      0.75      2859

    accuracy                           0.90     70619
   macro avg       0.88      0.85      0.86     70619
weighted avg       0.90      0.90      0.90     70619

Confusion Matrix:
 [[ 3864     3     0   734     0     0]
 [    0 29296   540  1581     2    16]
 [    0   889  3973     1    10   364]
 [  253  1409     0 21364     0     0]
 [    0     3    24     0  3197   237]
 [    0    19   507     0   248  2085]]
```

Conclusion :

Based on the classification results obtained from both Naive Bayes and the optimized K-Nearest Neighbors (KNN) algorithms, the following conclusions can be drawn:

1. Model Performance: The optimized KNN model achieved a higher accuracy of 90% compared to 88% for Naive Bayes. This indicates that KNN is slightly better at correctly predicting the AQI categories for this dataset.

2. Evaluation Metrics: The KNN model consistently outperforms Naive Bayes across key metrics like precision, recall, and F1-score. Particularly, the recall scores indicate that KNN is better at correctly identifying instances for most classes.

3. Confusion Matrix Analysis: The confusion matrix for KNN shows fewer misclassifications across most categories, highlighting its robustness in handling complex decision boundaries for this dataset.

4. Model Suitability: Given the high dimensionality of the dataset, the performance of Naive Bayes is still noteworthy, as it is computationally efficient. However, KNN's higher overall metrics suggest better suitability for this dataset when classification accuracy is prioritized.

Both models show strong performance, but KNN, with optimized parameters, proves to be more effective for the classification task of predicting AQI categories. Depending on the use case—whether quick, interpretable results (Naive Bayes) or higher accuracy (KNN) are preferred—either model could be selected.

Name : Advik Hegde **Div : D15C** **Roll No. : 15**

AIDS 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 :

Clustering Overview
Clustering is a form of unsupervised machine learning, where the model learns patterns in data that lack predefined labels. Rather than using target outcomes to guide learning, clustering identifies underlying structures and inherent groupings within datasets. The goal is to categorize data points into collections (clusters) such that items in the same group are more alike to each other than to those in other groups. For example, a visual plot might clearly reveal three distinct clusters where data points are closely packed, indicating they share certain attributes.

Practical Applications of Clustering
Clustering has wide-ranging use cases across different industries:

1. **Marketing** – Helps segment consumers based on purchasing behavior or preferences.

2. **Biological Research** – Aids in classifying species of flora and fauna.

3. **Library Management** – Books can be grouped by themes, topics, or genres.

4. **Insurance Sector** – Useful for profiling clients, detecting policy anomalies, or spotting fraud.

5. **Urban Development** – Assists in zoning properties based on value or location.

6. **Seismology** – Helps isolate earthquake-prone regions using historical tremor data.

Types of Clustering Techniques

When picking a clustering method, it's essential to consider the data volume and structure. Some algorithms involve pairwise similarity checks, resulting in a time complexity of $O(n^2)$, which may not scale well with large datasets. Various clustering methods are classified as follows:

1. **Density-Oriented Techniques**
   These techniques identify groups as areas of higher point density separated by sparser regions. They offer flexibility and can merge neighboring clusters effectively. Examples include DBSCAN (Density-Based Spatial Clustering of Applications with Noise) and OPTICS.

2. **Hierarchical Approaches**
   Clusters are constructed based on hierarchical relationships, forming a tree-like structure. This is split into:

   - Agglomerative: Starts with individual points and merges them upward.

   - Divisive: Begins with the full dataset and splits it recursively.
     Tools like CURE and BIRCH fall under this category.

3. **Partition-Based Techniques**
   These methods divide the data into a pre-set number of clusters. They focus on optimizing a distance metric to ensure tight clusters. K-Means and CLARANS are popular algorithms in this group.

4. **Grid-Oriented Methods**
   Here, the data space is divided into a grid structure, with operations applied on these cells. This approach is efficient and scales independently of data size. Notable algorithms include STING, CLIQUE, and WaveCluster.

Dataset Description :

The dataset used in ‘cleaned_ciy_hour.csv’,contains data about various pollutants that are used to determine the value of AQI for major Indian cities. It has about 20000 entries and contains data for a large number of cities like Mumbai,Delhi,Chennai, Jaipur etc.

In this case , we are using K - means clustering :

1. K-Means Clustering

Definition:
K-Means is a popular **unsupervised learning algorithm** used for grouping data into **k distinct, non-overlapping clusters**. The main idea is to minimize the distance between the data points and their respective cluster centroids (means).

How It Works:

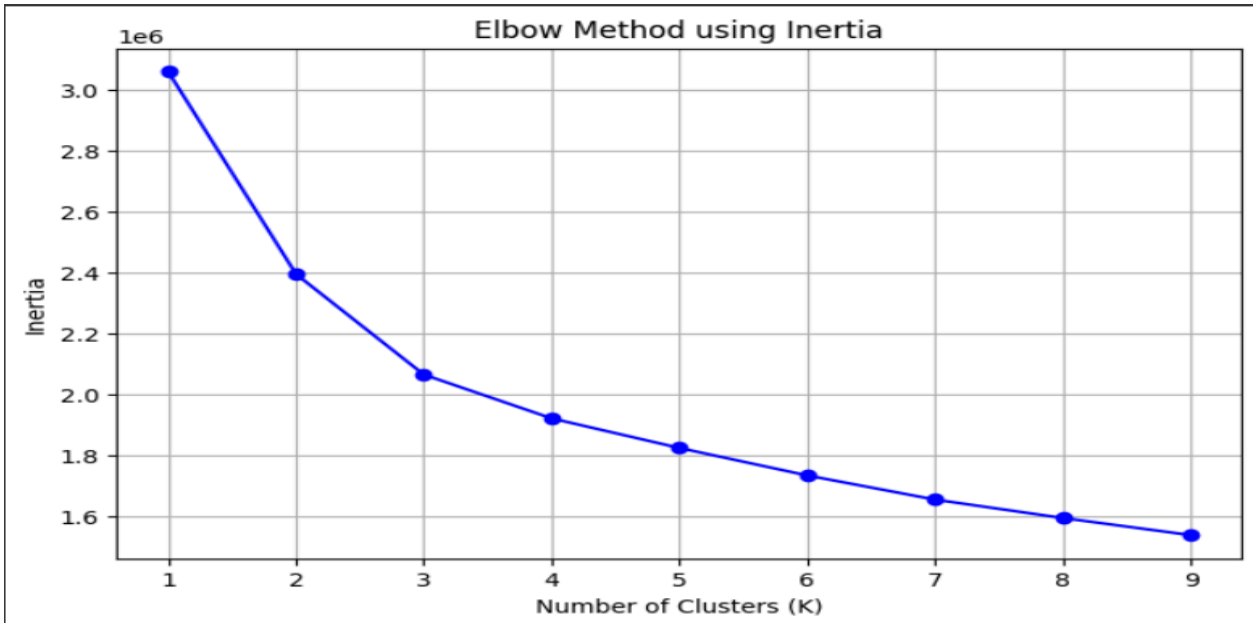
1. **Choose the number of clusters (k).**
2. **Initialize k centroids** randomly in the feature space.
3. **Assign each data point** to the nearest centroid (using distance measures like Euclidean distance).
4. **Recompute the centroid** of each cluster by calculating the mean of all points in that cluster.
5. **Repeat steps 3–4** until the centroids stop changing significantly (i.e., convergence is achieved).

Steps :

First, missing or null values are handled by either removing the affected records or replacing them with statistical estimates like mean or median. Then, irrelevant or redundant attributes are eliminated to reduce noise and focus on important features. The data is also checked for inconsistent formats or incorrect entries, which are corrected or standardized. After that, normalization or scaling is applied to ensure all numerical features are within the same range, which helps algorithms treat each feature equally.

After this, we move forward with k-means clustering for which the first step is to make use of the elbow method to identify the ideal number of clusters that the dataset should be divided into.

```
# Finding optimal number of clusters using the Elbow Method
inertia_values = []
K_range = range(1, 10)  # Trying K from 1 to 9

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(df_scaled)
    inertia_values.append(kmeans.inertia_)  # Store inertia (sum of squared distances)

# Plot the Elbow Method graph
plt.figure(figsize=(8, 5))
plt.plot(K_range, inertia_values, marker='o', linestyle='-', color="b")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Inertia")
plt.title("Elbow Method using Inertia")
plt.grid()
plt.show()
```

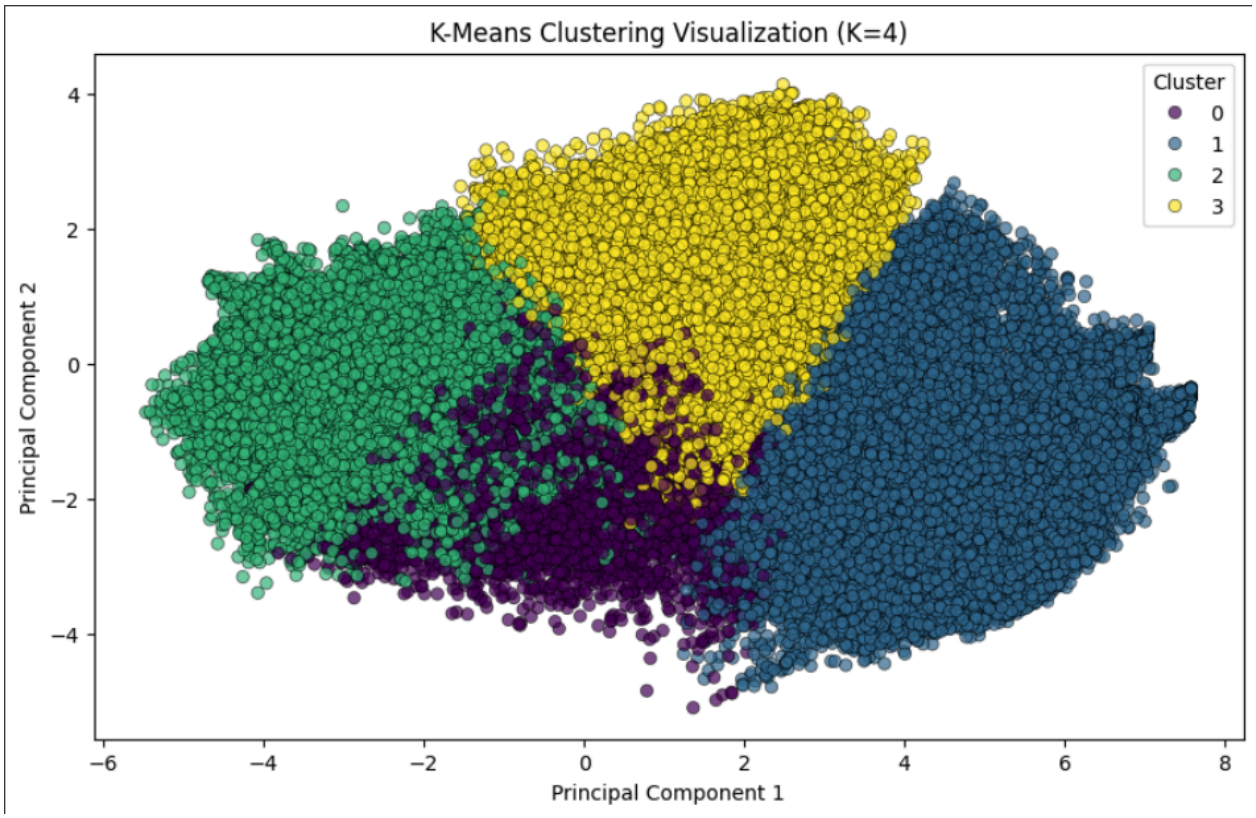


From this plot , we can identify that the ideal number of clusters for our dataset is 4 as the graph becomes smooth beyond that point and no longer bends that sharply .

So we move ahead and divide the dataset into 4 clusters :

The results of clustering are plotted below after applying PCA (Principal Component Analysis) which helps in numerosity and dimension reduction to visualize the results of clustering more clearly.

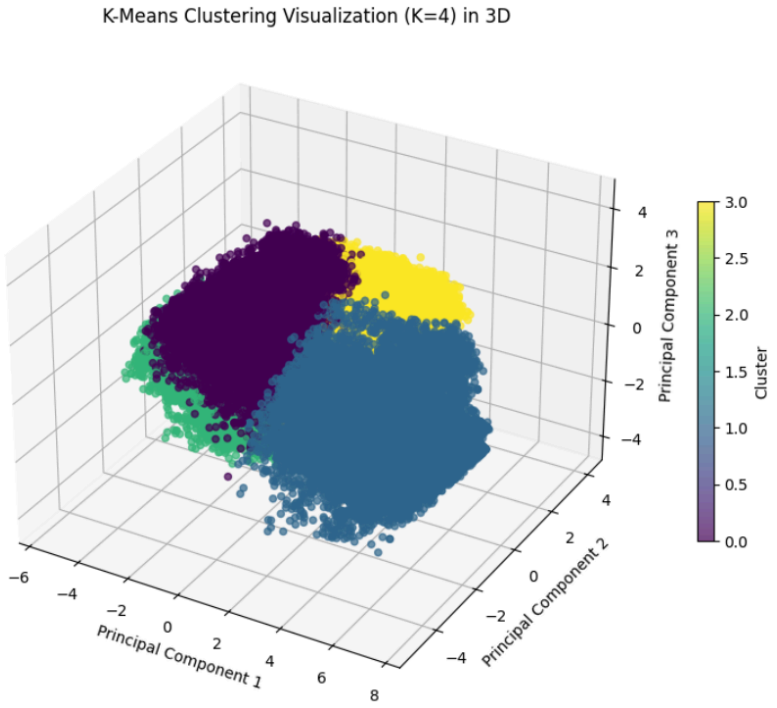
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

# Apply PCA to reduce dimensions to 2D
pca = PCA(n_components=2)
pca_features = pca.fit_transform(df_scaled.drop(columns=["Cluster_K4"]))  # Drop cluster column

# Create DataFrame for visualization
df_pca = pd.DataFrame(pca_features, columns=["PC1", "PC2"])
df_pca["Cluster"] = df_scaled["Cluster_K4"]

# Plot the clusters
plt.figure(figsize=(10, 6))
sns.scatterplot(x="PC1", y="PC2", hue=df_pca["Cluster"], palette="viridis", data=df_pca, alpha=0.7, edgecolor="k")
plt.title("K-Means Clustering Visualization (K=4)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.legend(title="Cluster")
plt.show()
```



After visualizing clustering results in 2D, plotting them in 3D provides a deeper understanding of how data points are grouped across three features. This involves selecting three relevant numerical attributes from the dataset and using a 3D scatter plot, where each axis represents one attribute. The data points are then plotted in 3D space, with colors or shapes indicating their cluster assignments. This helps in visually

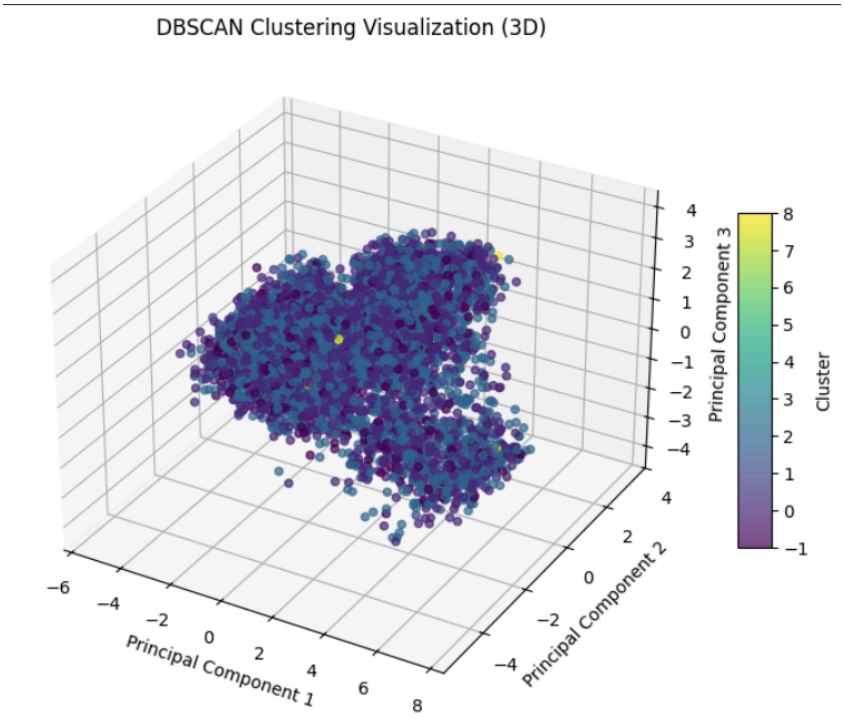
assessing cluster separation, compactness, and overlap that may not be fully visible in 2D plots.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D

# Apply PCA to reduce dimensions to 3D for K=4 clusters
pca = PCA(n_components=3)
pca_features = pca.fit_transform(df_scaled.drop(columns=["Cluster_K4"]))  # Drop cluster column

# Create DataFrame for visualization
df_pca = pd.DataFrame(pca_features, columns=["PC1", "PC2", "PC3"])
df_pca["Cluster"] = df_scaled["Cluster_K4"]  # Use K=4 clusters

# Plot the clusters in 3D
fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

# Scatter plot without outlines (edgecolors=None)
scatter = ax.scatter(df_pca["PC1"], df_pca["PC2"], df_pca["PC3"], c=df_pca["Cluster"], cmap="viridis", alpha=0.7)

# Labels and title
ax.set_title("K-Means Clustering Visualization (K=4) in 3D")
ax.set_xlabel("Principal Component 1")
ax.set_ylabel("Principal Component 2")
ax.set_zlabel("Principal Component 3")

# Colorbar for clusters
cbar = plt.colorbar(scatter, ax=ax, shrink=0.5)
cbar.set_label("Cluster")

plt.show()
```



As the next step after performing K-Means clustering, DBSCAN clustering is applied to the same dataset using three selected numerical features to explore density-based grouping. Unlike K-Means, which assumes spherical clusters and a fixed number of clusters, DBSCAN identifies clusters based on the density of data points, allowing it to detect irregularly shaped clusters and outliers. The three attributes used are plotted in a 3D scatter plot, with each point colored according to its DBSCAN-assigned cluster or marked as noise. This provides a more flexible view of the data structure, helping to compare how DBSCAN captures natural groupings differently than K-Means in a 3-dimensional space.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D

# Apply PCA to reduce dimensions to 3D
pca = PCA(n_components=3)
pca_features = pca.fit_transform(df_sample.drop(columns=["Cluster"]))  # Drop cluster column

# Create DataFrame for visualization
df_pca = pd.DataFrame(pca_features, columns=["PC1", "PC2", "PC3"])
df_pca["Cluster"] = df_sample["Cluster"]

# Plot the clusters in 3D
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection="3d")

# Scatter plot for clusters
scatter = ax.scatter(df_pca["PC1"], df_pca["PC2"], df_pca["PC3"],
                     c=df_pca["Cluster"], cmap="viridis", alpha=0.7)

# Labels and title
ax.set_title("DBSCAN Clustering Visualization (3D)")
ax.set_xlabel("Principal Component 1")
ax.set_ylabel("Principal Component 2")
ax.set_zlabel("Principal Component 3")

# Add color bar
legend1 = fig.colorbar(scatter, ax=ax, shrink=0.5, aspect=10)
legend1.set_label("Cluster")

plt.show()
```



Conclusion:

In this experiment, we implemented and compared two popular unsupervised clustering algorithms: **K-Means** and **DBSCAN**, using the adult.csv dataset.

- **K-Means** clustering required us to select the number of clusters (k) using the **Elbow Method**, which helped in identifying the optimal k by observing the point where WCSS started to level off.
- **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) automatically detected clusters based on data density, without requiring k. It also identified outliers (labeled as -1), which K-Means cannot do.

This comparison highlighted the strengths of both algorithms — **K-Means** works well for well-separated spherical clusters, while **DBSCAN** is more robust in handling **noise and arbitrary-shaped clusters**, making it suitable for more complex data distributions.

Name : Advik Hegde **Div : D15C** **Roll No. : 15**

AIDS Experiment 8

Aim: To implement a recommendation system on your dataset using the following machine learning techniques.
o Regression
o Classification
o Clustering
o Decision tree
o Anomaly detection
o Dimensionality Reduction
o Ensemble Methods.

Theory :

A recommendation system leverages various machine learning techniques to predict user preferences and suggest items. Techniques like regression and classification predict numerical ratings or categorize items for users, while clustering groups similar items for more relevant suggestions. Anomaly detection identifies outliers in user behavior, and dimensionality reduction simplifies data to improve efficiency. Ensemble methods combine multiple models to enhance prediction accuracy.

Decision trees play a crucial role in recommendation systems by modeling the relationship between user features and item preferences. These trees split data based on specific features, creating a clear path for predictions. In the context of recommendations, decision trees can classify or predict which items a user is most likely to prefer by analyzing patterns in user-item interactions. This method is highly interpretable and efficient, especially for datasets with numerous attributes, providing transparent decision-making for personalized suggestions.

Dataset Description :

The **Books.csv** dataset contains metadata for a collection of books, including information such as titles, authors, publication years, publishers, and cover images. This data provides a comprehensive view of the books in the system, which can be leveraged to categorize and recommend books based on various attributes.

The **Ratings.csv** dataset contains user interactions with the books, specifically user ratings for each book. This dataset reflects user preferences and allows for the creation of personalized recommendations by analyzing how users rate different books. By combining both datasets, a recommendation system can suggest books based on user ratings and book attributes.

Steps to perform :

1. First, missing or null values are handled by either removing the affected records or replacing them with statistical estimates like mean or median. Then, irrelevant or redundant attributes are eliminated to reduce noise and focus on important features. The data is also checked for inconsistent formats or incorrect entries, which are corrected or standardized. After that, normalization or scaling is applied to ensure all numerical features are within the same range, which helps algorithms treat each feature equally.

```
# Install if needed:
# pip install pandas scikit-learn matplotlib seaborn

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

```
[ ] # Load datasets
books = pd.read_csv('Books.csv', encoding='latin-1')
ratings = pd.read_csv('Ratings.csv', encoding='latin-1')

# Merge books and ratings on ISBN
data = pd.merge(ratings, books, on='ISBN')

# Drop rows with missing values in critical columns
data.dropna(subset=['Book-Rating', 'Year-Of-Publication'], inplace=True)

# Preview merged data
data.head()
```

2. To develop a decision tree, we need to make sure that we establish a boundary of rating beyond which it can be said that a user does or does not like a book. In our case we decide that rating number oto be “7”.So , if a user rates a book 7 or above, we assume that it can be said that the user likes a book and for all ratings below it,we assume that the user does not like that book.

```
# Define like = 1 if rating >= 7, else dislike = 0
data['Liked'] = data['Book-Rating'].apply(lambda x: 1 if x >= 7 else 0)
```

3. We clip extreme years to ensure that the values fall within a valid range, between 1000 and 2025, to handle any outliers or invalid years. For the feature matrix X, we select the relevant columns (Year-Of-Publication, Author, and Publisher) and apply one-hot encoding to the categorical variables (Author and Publisher), converting them into numerical format. Finally, we set the target variable y as the Liked column, which indicates the user's preference.

```
# Convert year to numeric
data['Year-Of-Publication'] = pd.to_numeric(data['Year-Of-Publication'], errors='coerce')

# Replace missing values and clip extreme years
data['Year-Of-Publication'] = data['Year-Of-Publication'].fillna(0)
data['Year-Of-Publication'] = data['Year-Of-Publication'].clip(lower=1000, upper=2025)

# Feature matrix (X) and target (y)
X = data[['Year-Of-Publication', 'Author', 'Publisher']]
X = pd.get_dummies(X, drop_first=True)  # One-hot encoding for categorical

y = data['Liked']
```

4. After this, we split the dataset in a 20:80 split to train the model and after training , we evaluate the performance of the model .The evaluation metrics of the model are :

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(class_weight='balanced', max_depth=10, random_state=42)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

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

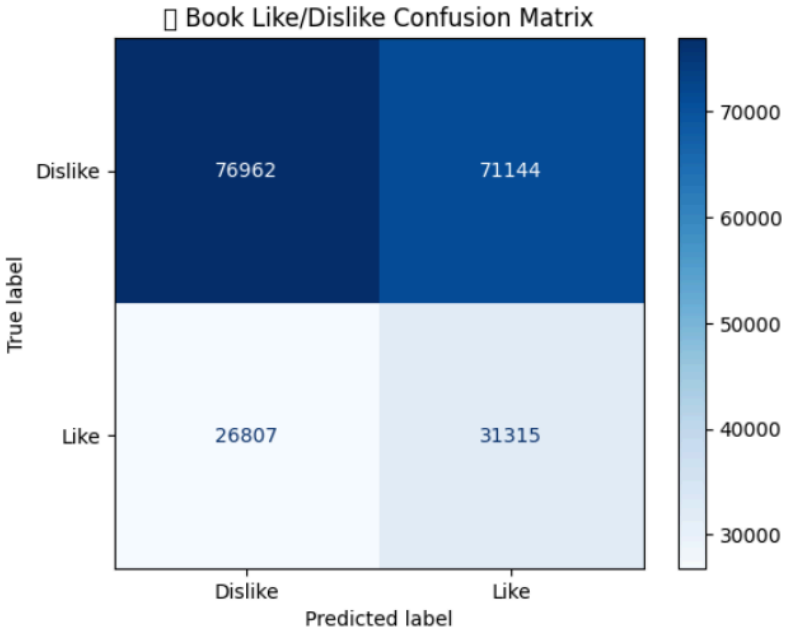
```
Accuracy: 0.5250353977151502
              precision    recall  f1-score   support

           0       0.74      0.52      0.61    148106
           1       0.31      0.54      0.39     58122

    accuracy                           0.53    206228
   macro avg       0.52      0.53      0.50    206228
weighted avg       0.62      0.53      0.55    206228
```

5. We can then use this model to make predictions for whether or not a user will like a book from various years

```
# Predict whether user might like a book from year 2010
new_books = pd.DataFrame({
    'Year-Of-Publication': [1985, 2000, 2010, 2015]
})

predictions = model.predict(new_books)

for year, pred in zip(new_books['Year-Of-Publication'], predictions):
    print(f"Book published in {year} → {'Like' if pred == 1 else 'Dislike'}")
```

```
Book published in 1985 → Dislike
Book published in 2000 → Like
Book published in 2010 → Like
Book published in 2015 → Dislike
```

The confusion matrix for the model can also be used to evaluate it's performance



Conclusion : The implementation of a recommendation system using machine learning techniques, specifically decision trees, has proven effective in predicting user preferences for books. By leveraging book metadata and user ratings, we created a model that classifies whether a user will like a book based on certain features like publication year, author, and publisher. The decision tree model provides transparency in its decision-making process, allowing for clear interpretations of user-item interactions. Through data preprocessing, feature encoding, and model evaluation using performance metrics like the confusion matrix, the system demonstrates how machine learning can be used to make personalized recommendations with accuracy.

Name: Div: Roll no:

Experiment 9

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

Theory:

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

Apache Spark is an open-source, distributed computing system designed for big data processing and analytics. It supports programming languages like Python, Scala, Java, and R. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance.

Key components and working:

- **Driver Program**: Coordinates the execution of Spark applications by creating SparkContext and handling task scheduling.
- **Cluster Manager**: Allocates resources to the Spark applications across the cluster.
- **Executors**: Run on worker nodes to execute tasks assigned by the driver.
- **RDD (Resilient Distributed Dataset)**: A fault-tolerant collection of elements that can be operated on in parallel.
- **Lazy Evaluation**: Spark doesn't execute transformations until an action is triggered, which optimizes execution.

Spark also includes high-level APIs for structured data processing via **DataFrames** and **Spark SQL**, enabling SQL-like queries on distributed datasets. It integrates well with other big data tools and supports various data sources like HDFS, Cassandra, HBase, and Amazon S3. Spark's ability to process data in-memory significantly reduces disk I/O, making it much faster than traditional frameworks like Hadoop MapReduce. This speed, combined with scalability and ease of use, makes Spark ideal for data exploration, machine learning, and real-time analytics on large datasets.

2. How is data exploration done in Apache Spark?

Step 1: Loading the Data
Use `spark.read.csv()` or similar methods to load data from different sources such as CSV, JSON, or Parquet.

Step 2: Inspecting the Schema
Use `.printSchema()` or `.dtypes` to view the structure and data types of each column.

Step 3: Viewing Sample Data
Use `.show()` or `.head()` to look at a few initial records and get an idea of the data.

Step 4: Summary Statistics
Apply `.describe()` or `.summary()` to generate basic statistics like mean, standard deviation, min, max, and count.

Step 5: Checking for Missing Values
Use `.filter()` or `.where()` with `.isNull()` to identify null or missing entries.

Step 6: Counting Unique Values
Use `.groupBy(column).count()` to see frequency distributions of categorical or numerical features.

Step 7: Correlation Analysis
Use `.corr()` method to measure the linear correlation between numerical variables.

Step 8: Visualization
Since Spark doesn't support advanced visualizations directly, convert Spark DataFrames to Pandas using `.toPandas()` and use libraries like Matplotlib or Seaborn for plotting.

Conclusion:

Apache Spark is a powerful tool for performing EDA on large datasets thanks to its distributed architecture and in-memory processing. When combined with Pandas and Python's visualization libraries, it becomes possible to efficiently analyze and visualize even large datasets. Spark enables scalable, fast, and interactive exploration that helps identify data quality issues, trends, and patterns before modeling. It supports parallel processing, making it ideal for handling large-scale data without performance bottlenecks. Overall, Spark greatly enhances the EDA process by accelerating insights and enabling more informed decision-making.

Name: Div: Roll no:

Experiment 10

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

Theory:

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

Streaming refers to the continuous and unbounded flow of data generated in real-time from various sources such as sensors, logs, online transactions, or social media platforms. Unlike traditional processing methods that wait for the data to be collected, streaming processes each piece of data almost instantly as it arrives, often within milliseconds or seconds. This allows for real-time analytics, monitoring, and event-driven applications.

- Batch Data is collected and stored over a time period, then processed as a single unit. This method is effective when immediate results are not required. It is widely used for ETL processes, periodic reporting, and historical data analysis, where latency is acceptable.

- Stream Data, on the other hand, is continuously produced and must be processed in near real-time. It is ideal for applications that require instant feedback, like fraud detection systems, stock trading platforms, live metrics dashboards, or user activity tracking. The main advantage of stream processing is its ability to provide timely insights and actions.

2. How Data Streaming Takes Place Using Apache Spark.

Apache Spark Streaming is a powerful component built on top of the core Spark engine, enabling real-time stream processing by using a micro-batch architecture. It processes data streams in small batches, which balances performance and latency. Here's how it works:

- Input Sources: Spark Streaming can ingest data from a variety of real-time sources such as Apache Kafka, Apache Flume, Amazon Kinesis, TCP sockets, or even files monitored in real-time.

- Micro-batch Architecture: Spark collects the data for a short interval (e.g., every 1 or 2 seconds) and creates a batch. These batches are then processed using the same

APIs used for batch jobs, which makes it easier for developers to switch between batch and stream use cases.

- DStreams (Discretized Streams): Spark represents incoming streaming data as DStreams, which are essentially a continuous series of RDDs. This abstraction allows developers to apply familiar operations such as map(), reduceByKey(), filter(), etc., to process data.

- Transformations and Actions: Developers can perform complex operations on the stream such as aggregations, joins, and windowed computations. The results can be sent to databases, file systems, or real-time dashboards.

- Output: The final output of stream processing can be stored or visualized in real-time, enabling instant feedback and data-driven decisions. Spark also supports stateful operations to maintain data across batches.

Conclusion:

Apache Spark offers a robust and unified platform for handling both batch and stream data processing. While batch processing is efficient for handling large volumes of historical data, Spark Streaming brings the capability to process data in near real-time, helping organizations gain insights instantly. Its micro-batch architecture strikes a good balance between performance and complexity, making it suitable for both small-scale and enterprise-grade real-time applications. Spark's ability to connect to a variety of input/output systems, combined with fault tolerance and scalability, makes it a leading choice for modern data processing needs. Moreover, Spark Structured Streaming now provides even more powerful and user-friendly APIs for developing continuous applications with built-in support for event time, stateful processing, and fault recovery.

Name : Advik . Hegde

Div : D15C R.No: 15 AIDS - Assignment - 1

Assignment - I : →

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

→ Artificial Intelligence (AI) enables machines to think, learn and make decisions like humans. It includes technologies like machine learning, NLP and robotics.

Applications:
1) Healthcare: AI helped in early diagnosis, vaccine development, and chatbot-based health assistance.
2) Contact Tracing: AI-powered apps tracked COVID-19 exposure ensuring public safety.
3) Remote work and education: AI enhanced logistics and virtual meetings, online learning and productivity tools.
4) Supply Chain and Delivery: AI optimized logistics and enabled autonomous deliveries.
5) Mental Health Support: AI-driven apps provided emotional and fitness assistance.

Q2) What are agents terminology, explain with examples.
→ 1) Agent: An entity that interacts with the environment and makes decision based on inputs.
ex: a self-driving car perceives traffic signals and adjusts speed accordingly.

2) Performance measures: Defines how successful an agent is in achieving it's goal.

ex: A self-driving cars performance measures, cou minimizing accidents, fuel efficiency and travel time

b) Behavior action of agent: The action an agent takes based on its percepts.
ex. a robotic vacuum cleaner moves around obstacl after detecting them.

4) percept: The data an agent receives at a specific moment from sensors. ex: A spam filter receives an email and detects keywords, sensor info, and attachmen

5) percept sequence: The entire history of percepts receive by an agent. ex: A chess playing AI remembers all previ moves in the game before making its next move.

6) Agent Function: A mapping from the percept sequence to an action. ex: A smart thermostat analyzes past temperature changes and adjusts heating accordingly.

Q3) How AI technique is used to solve the 8 puzzle problem?
→ It consists of a 3×3 grid with numbered tiles and one empty space, where the objective is to move the tiles around to match a predefined goal configuration.

Initial state:

1	2	3
4		6
7	5	8

This is the random starting configuration of the 8-puzzle

with the tiles placed in a non-goal configuration.

2) Goal state: The goal is to arrange the tiles in a specific order with the blank space at bottom right

Goal state:
1 2 3
4 5 6
7 8

Solving the puzzle problem
- AI search algorithms, such as Breadth-first search (BFS), depth first search (DFS) and A* are commonly used.

1) Breadth First Search (BFS):
- BFS is an uninformed search algorithm that explores all possible states level by level, starting from initial state.
- BFS guarantees that the solution found is the shortest in terms of no. of moves, but it can be very slow.

Advantages: Guaranteed to find optimal solution
Disadvantages: BFS has a high memory requirement, as it must store all the states at each level of explora-tion.

2) Depth-First Search (DFS)
DFS is another uninformed search algorithm that explores one branch of the state space tree as deep as possible, before backtracking.

Advantages: DFS is more memory efficient than BFS.
Disadvantages: DFS can get stuck in deep, non-optimal paths and may not find the shortest solution.

Steps using A*
- compute manhattan distance for each possible move
- choose best move (lowest F(n))
- repeat until reaching goal state.

Q7 What is PEAS descriptor? Give peas descripter for following

→ 1) Taxi Driver
P: Minimize travel time, fuel efficiency, passenger safety, obey traffic rules.
E: Roads, traffic, passengers, weather, ~~obedient~~ obstacles, pedestrians.
A: Steering, accelerator, brakes, turn signals, horn
S: Camera, GPS, speedometer, radar, LiDAR, microphone

2) Medical Diagnostic System
P: Accuracy of diagnosis, treatment success rate, response time
E: Patient records, symptoms, medical tests, hospital database.
A: Display screen, printed prescriptions, notifications.
S: Patient input, lab reports, electronic health records.

3) Medical diagnosis system
P: Accuracy of diagnosis, speed, reduced misdiagnosis rate.
E: Hospital database, patient records, symptoms, medical history.
A: Display screen, reports, alerts to doctors/patients

then BFS
2-optim

S: User inputs, patient history, test results, electronic health records.

3) Music composer.
P: Harmony, creativity, user satisfaction, adherence to style.
E: Musical styles, melody rules.
A: ~~M~~ Music notes' generation, play sounds
S: User feedback, existing music datasets

4) Aircraft Autolander.
P: Safe landing, smooth descent, fuel efficiency
E: Airport runway, wind speed, air traffic
A: adjust flaps, landing gear, braking system.
S: Altimeter, radar, GPS.

5) An Essay Evaluator (Automated Grading System)
P: Accuracy in grading, fairness, consistency.
E: Essays, grammar rules, grading rubrics.
A: ~~P~~s Feedback generation, score assignment, highlighting errors, suggesting improvement.
S: optical character recognition, NLP, gramar and spell checkers

6) A robotic sentry gun for the Keck Lab.
P: Target accuracy, threat deflection efficiency, response spee
E: Keck lab premises, intruders, lighting conditions
A: Gun aiming system, camera panning, alert syster
S: Motion detectors, infrared sensors, cameras, LIDAR.

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

→ 1) Partially observable: Bot may not have complete visibility.
2) Stochastic: The environment is unpredictable.
3) Sequential: Each decision bot makes affects future states.
4) Dynamic: The bookstore environment changes over time.
5) Discrete: Bot chooses discrete choices (selecting books)
6) Multi-agent: Bot interacts with multiple agents.

Q.6) Differentiate b/w Model based and utility based agent.

→

Model Based Agent	Utility Based Agent
1) Maintains an internal model of env. to make decisions	1) uses a utility function to measure performance and make choices
2) Relies on stored knowledge and updates model.	2) Chooses actions based on maximizing expected utility.
3) can adapt to changing environment by updating internal mode	3) More flexible and goal-oriented, adapting to changes dynamically.
4) Moderate complexity due to model maintainance.	4) Higher complexity due to the need to compute utilities for different actions.

5) Ex - self-driving car that predicts pedestrian move-ment

5) A self driving car that evaluates and selects the best one.

Q.7) Explain architecture of Knowledge based agent and learning agent.

→ 1) Knowledge based Agent Architecture.

A Knowledge based agent is an intelligent agent that makes decisions using Knowledge base (KB) and reasoning mechanism.

Architecture component:

1) Knowledge Base: Stores fact, rules and heuristics about the world.

2) Inference Engine: Use logical reasoning (FOL) to derive new knowledge from the KB.

3) Perception Module: Collects data from sensor and update the KB.

4) Action Selection Module: Chooses appropriate actions based on reasoning outcomes

5) Communication Module: Allows interaction with other agents.

Working Process:

- The agent perceives the environment and updates its KB
- The Interference engine applies logical rules to infer new knowledge.
- The agent decides an action and executes it.
- The KB is continuously updated to improve decision making.

2) Learning Agent Architecture:
A Learning agent improves it's performance over by learning from past experiences and interactions the environment.

Architecture components
1) Learning Element: Analyzes feedback from the environment and improves Knowledge.
2) Performance: Make decisions and executes actions
3) Critic: Evaluates the agent's action and provides feedback
4) Problem generator: Suggests exploratory actions to improve learning.

Working process:
- The performance element selects an action.
- The critic evaluates the action and provides feedback.
- The learning element updates the agents' knowledge to improve future decisions.
- The problem generator suggests new strategies to explore better solutions.

Q8) What is AI? Considering the COVID-19 pandemic situation, how has AI helped to survive and renovated the way of life with diff. applications.

→ Artificial Intelligence is the simulation of human intelligence in machines that can learn, reason and make decisions. AI systems process large datasets, recognize patterns and automate tasks, enhancing efficiency across industries.

AI's role in COVID-19 pandemic.
1) Healthcare & Diagnosis: AI analyzed CT scans and detected COVID-19 faster.
2) Chatbots and virtual assistants: provided instant medical advice.

Q.9) Convert the following to predicates:
a) Anita travels by car if possible, otherwise by bus.
car available → travels by Car(Anita)
¬ car Available → Travels by Bus(Anita)

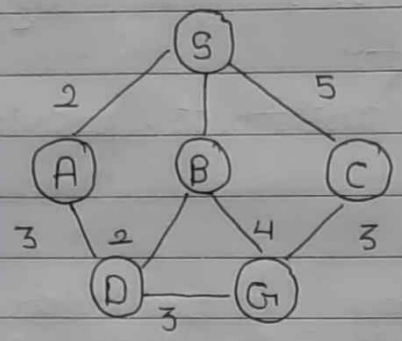
Bus goes via Andheri and goregaon
goes Via(Bus, Andheri) ∧ goes Via(Bus, Goregaon)

car has puncture, so not available
puncture(car), puncture(car) → ∅ ¬ car Available

will Anita travel via gurgaon? use backward ~~chaining~~ reasoning.
From (c)
puncture(car) is true
As puncture(car) → ¬ (car Available
From (a)
¬ Car Available, we use ¬ car Available → Travels by Bus (Anita)
from (b)

goes via(Bus, Goregaon)
Since, Anita travels by bus, she will follow this route
Thus, Anita will travel via gurgaon.

Q.10) Find the route from S to G using BFS



Correct Node	Queue	Visited Node
S	A B C	S
A	B C D G	S → A
B	C D G	S → A → B
C	D G	S → A → B → C
D	G	S → A → B → C → D
G		S → A → B → C → D → G

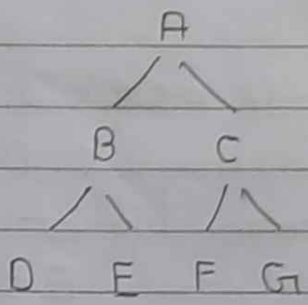
The path is : S → A → B → C → D → G

Q.11) What do you mean by depth limited search? Explain iterative deepening search with example.

→ Depth limited search (DLS) is an uninformed search algorithm that modifies DFS by introducing a depth limit L preventing exploration beyond the defined level. This prevents infinite loops in graphs but risks missing goals beyond L.

Iterative Deepening search (IDS) combines DF DLS with BFS by incrementally increasing the search level.

example:



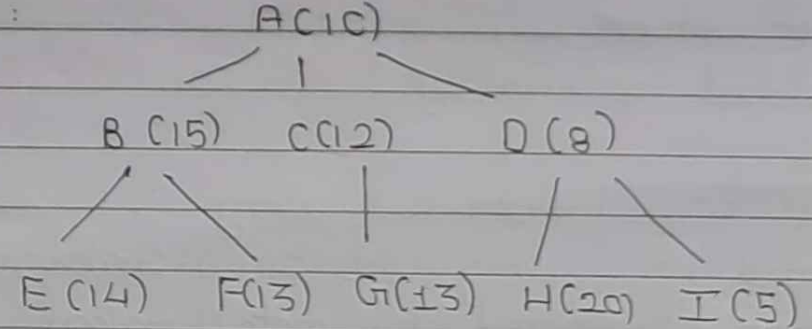
goal = G

Initially, the depth level is 0 for iteration - 1.
Nodes visited = A (goal not found)
Iteration 2, limit = 1
Nodes visited = A → B → C (goal not found)
Iteration 3, limit = 2
Nodes visited : A → B → D → E → C → F → G
Goal : G is found.

12] Explain Hill climbing and its drawbacks in detail with example. Also ~~state~~ limitations of steeper ascent hill climbing.

→ Hill climbing is a local search optimization algorithm which moves forward towards a better neighboring solution until it reaches a peak.

example:



goal : G.

Steps:
1) Start at root node A(10)
2) Compare its children B, C, D.
3) Move to child with highest value i.e. B(15).
4) Repeat for B's children E and F.
5) Terminate at E(14)

The algorithm stops at E(14) not reaching goal G

Drawbacks:
1) Local maxima: The algorithm greedily selects the best immediate child and can thus get stuck on local maxima.

2) plateaus: If siblings have equal values, the algorithm can't decide the next step and gets stuck.

3) ridges: narrow uphill paths require backtracking which hill climbing algorithm does not support.

4) Limitations of steepest ascent hill climbing.
- Computationally expensive: evaluates all neighbors before selecting the best.
- can get stuck.
- no global optimality.

13) Explain simulated annealing and write the algorithm.
→ Simulated annealing is a probabilistic optimization algorithm inspired by metallurgical process of annealing where materials are heated and cooled to

reduce defects. It escapes the local optima by temporarily accepting worse solution with a probability.

Algorithm

1) Initialize: Set an initial solution and define an initial temperature T.
2) Repeat until stopping condition.
   - Generate new neighbour soln.
   - compute changes in cost.
   - If new soln is better then accept it.
   - If worse, accept it with probability.
   - Decrease temp. T.
3) Return best solution.

ex: travelling salesman problem

---

14) Explain A* algorithm with an example.

→ A* is a best first search algorithm used in pathfinding and graph traversal. It uses the foll. formulas:

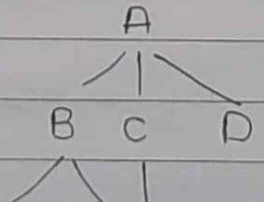
$f(n) = g(n) + h(n)$

$g(n)$ → cost to reach n from start.

$h(n)$ → heuristic estimate

$f(n)$ → total estimated cost.

Goal G:



Node	g(A,n)	h(n, G)
A	0	6
B	1	4
C	2	2
D	~~3~~ 4	7
E	3	5
F	5	3
G	6	0

Steps

1) Start at root node A

$f(A) = g(A) + h(A) = 0 + 6 = 6$

2) expand neighbors B, C, D

$f(B) = 1 + 4 = 5$, $f(C) = 2 + 2 = 4$, $f(D) = 4 + 7 = 11$

3) Choose lowest value that is f(C)

4) Expand neighbors of C.

$f(G) = 2 + 4 + 0 = 6$

5. Goal reached at G with total cost 6.

Advantages :→

Efficient for finding shortest path in weighted graphs.

balances exploration by considering both g(n) and h(n)

17 Explain minmax algorithm and draw game trees for tic-tac-toe.

→ The minmax algo is a decision making algorithm used in 2 players games. It assumes

- one player (MAX) tries to maximize score
- other player (min) tries to minimize score.

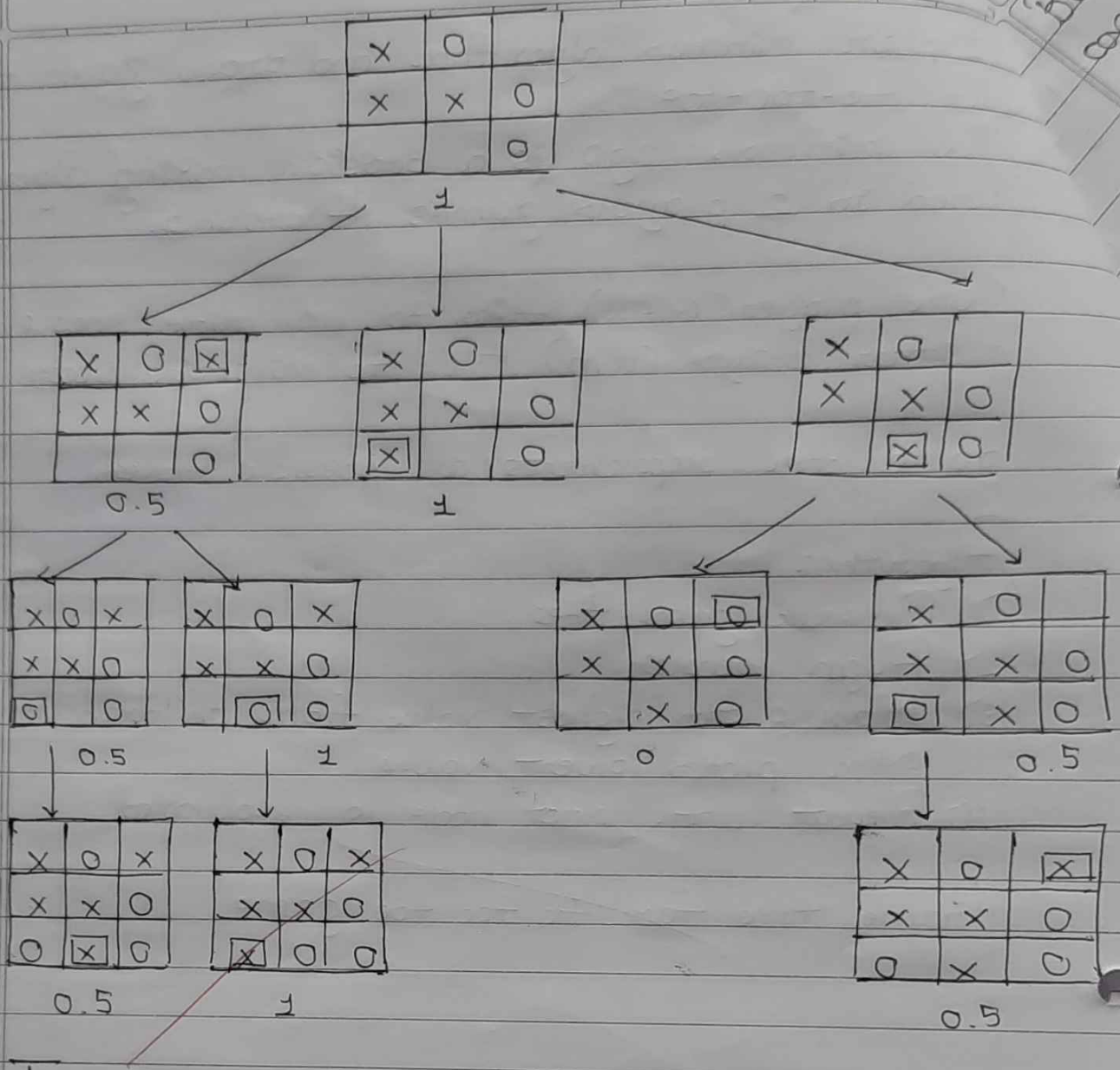
game tree represents all possible moves.

Algorithm

1) Generate game tree.
2) Align scores.
3) Max picks highest value from children
   Min picks lowest value.
4) Repeat until root node is evaluated.

Game tree for tic tac toe game.

P.T.O →



16) Explain alpha beta pruning algorithm for adversarial search with example.

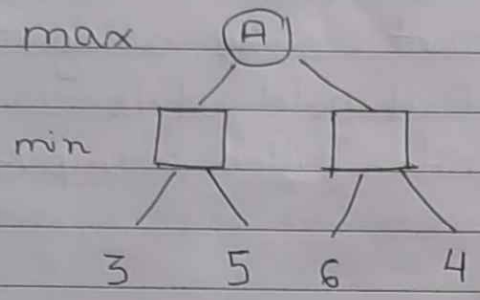
→ Alpha beta pruning is an optimization technique used in minimum algorithm to reduce the no. of nodes evaluated in adversarial search problems like game-playing AI (eg chess, tic tac toe).

Alpha beta pruning includes:

Alpha (α): Best max score that max player can guarantee so far.

Beta (β): The best min score that MIN player can guarantee.

ex:



1) Start at root node A.
$\alpha = -\infty$, $\beta = +\infty$

2) Check left min node (child of A)
check first child value → update $\beta = 3$
check second value = 5 → β remains 3
min node returns 3 to max.

3) right min node (child of A)
check first child value = 6 → $\beta = 6$
pruning ~~of~~
here $\alpha = 3$ at max but $\beta(6) > \alpha(3)$ so, no pruning.
explore 2nd child (9) → here pruning will occur
min node already has value ≤ 6, it will never choose 9 and so, we prune node with value 9.

4) Max value = 6

17) Explain wumpus world environment, giving its PEAS description. explain how percept seq. is generated.

→ The WUMPUS world environment is a simple based environment used in AI to study intelli agents behavior in uncertain environments. It is turn based environment where an agent must navigate a cave to find gold while avoiding hazard like pits and a monster called wumpus.

PEAS:

P: the agent is rewarded for grabbing gold and exiting safely. Penalty is imposed for falling into pits and getting eaten by wumpus.

E: 4 × 4 grid world containing agent, wumpus, pits, gold.

A: agent can move forward, left, right, shoot, climb

S: agent perceives stench, breeze, glitter, bump and scream.

example percept sequence:

1) Agent starts at (1, 1)
no breeze, no stench, no glitter → safe square.

2) agent moves to (2, 1)
breeze detected → A pit is nearby but not in current square.

3) moves to (1, 2)
Stench detected → wumpus in adjacent cell.

4) moves to (2, 2)
glitter detected → gold is here.

5] agent moves back to (1,1), climbs out.

18] Solve the foll. crypto-arithmetic problem.
SEND + MORE = MONEY.

→ Step 1:
M must be 1. The sum of 2 4-digit nos cannot be more than 10,000.

```
 SEND
+MORE
-----
10NEY
```

2] S must be 8, as 1 is carry over from EON.
O must be 0 and there is a 1 carried or S=9 and there is no 1 carry. But 1 is already taken, so O must be 0.

```
  SEND
+ 10RE
-------
10NEY
```

3] There cannot be carry from EON because any digit + 0 < 10, unless there is carry from column NRE and E=9. But this cannot be the case because then N would be 0 and 0 is already taken. So, E<9 and there is no carry from this column. Therefore S=9 because 9+1=10

Step 4:
case 1:

No carry : $N+R = 10+(N-1) = N+9$

$R = 9$

but 9 is already taken so will not work

case 2:

carry : $N+R+1 = 9$

$R = 9-1 = 8$. This must be solution of R.

Step 5:

lets consider $E = 5$ or 6

$E = 5$

then $D = 7$ and $Y = 3$, so this part will work but look at column N8E, There is carry from DEY, $N+8+1 = 16$ but then $N = 7$ and 7 is taken by D, therefore $E = 5$

```
 9 5 N D
+1 0 8 5
-------
1 0 N 5 Y
```

Now, $N+8+1 = 15$, $N = 6$

```
 9 5 6 D
+1 0 8 5
-------
1 0 6 5 Y
```

Step 6:

The digits left are 7, 4, 3 and 2. We know there is carry from column D5Y, so only pair that works is $D = 7$ and $Y = 2$

$$\begin{array}{r} 9\ 5\ 6\ 7 \\ +\ 1\ 0\ 8\ 5 \\ \hline 1\ 0\ 6\ 5\ 2 \end{array}$$

19] Consider the foll. axioms:
All people who are graduating are happy.
All people happy people are smiling.
Someone is graduating.

→ ① Represent there axioms in first order predicate logic.

In clause form: $\{G(a)\}$

② Prove "is Someone smiling?" using resolution.
collect of clauses
(1) $\{\neg G(x), H(x)\}$
(2) $\{\neg H(x), S(x)\}$
(3) $\{G(a)\}$

Apply resolution
resolve (1) $\{\neg G(x), H(x)\}$ with (3) $\{G(a)\}$
Substituting $x = a$.
$\{\neg G(a), H(a)\}$
∵ we have $G(a)$, resolving gives
$\{H(a)\}$
Resolve (2) $\{\neg H(x), S(x)\}$ with $\{H(a)\}$
Subs. $x = a$
$\{\neg H(a), S(a)\}$
Since, we have $H(a)$, resolving given $\{S(a)\}$.

Since we have derived S(a), we conclude th someone(a) is smiling.

20) Explain modus ponen with suitable ex.

→ Modus ponen is a fundamental rule of inferen in propopositional logic that allows us to deduc a conclusion from a conditional statement and th antecedent.

It follows the form
1) $P \rightarrow Q$ (if P, then Q)
2) P (P is true)
   Q (Q must be true)

ex. If it rains, ground will be wet.
→ $P \rightarrow Q$

It is raining → P, ground is wet → Q

21) Explain forward chaining and backward chaining algo with the help of an example.

→

Forward chaining: It starts with given facts and applies inference rules to derive new facts until goal is reached. It is a data driven approach because it begins with known data and works forward to reach a concln.

ex: diagnosing a disease
Rules:
1) If a person has fever & cough, they might have flu.
2) If a person has sore throat and fever, they might have cold.

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

1) Fever + cough → flu (rule 1 applies)
2) conclusion: patient might have flu.

Backward chaining: It starts with goal and works backwards by checking what facts are needed to support it. It is a goal driven approach.

ex: diagnosing a disease.
goal: determine if patient has flu

Rules:
1) (Fever ∧ cough) → flu
2) (sore throat ∧ Fever) → cold.

Process using backward chaining:
1) We want to prove flu.
2) Looking at rule 1 (Fever ∧ cough) → Flu, we need to check if patient has fever and cough.

3) We check our known facts.
- patient has fever
- patient has cough

4) Since, both conditions are met, we confirm flu is true.

Name : Advik Hegde **Div : D15C** **Roll No. : 15**

AIDS Assignment -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 (10pts)
2. Find the Median (10pts)
3. Find the Mode (10pts)
4. Find the Interquartile range (20pts)

Answer :

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

1. Mean (10 pts)

To find the mean, sum all numbers and divide by the total count.

Mean = (82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90) ÷ 20
Mean = 1621 ÷ 20
Mean = 81.05

2. Median (10 pts)

Step 1: Sort the data:
59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Step 2: Median is the average of the 10th and 11th values:
Median = (81 + 82) ÷ 2
Median = 81.5

3. Mode (10 pts)

The number that appears most frequently is:

- 76 (appears 3 times) and Others appear only once or twice.

Mode = 76

4. Interquartile Range (IQR) (20 pts)

Step 1: Use the sorted data again:
59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Step 2:

- Q1 (25th percentile) = average of 5th and 6th values
  Q1 = (76 + 76) ÷ 2 = 76

- Q3 (75th percentile) = average of 15th and 16th values
  Q3 = (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?

   - Predictive analytic
   - Descriptive analytic

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

   - Supervised learning
   - Unsupervised learning
   - Reinforcement learning

Answer :

1) Machine Learning for Kids

- **Target Audience:**
  Primarily designed for school-age students (ages 8–16) and beginners in machine learning and AI.

- **Use by Target Audience:**
  Students use this tool to create simple machine learning models by training them with labeled examples. They often apply it in projects using platforms like Scratch or Python to build games, quizzes, or basic chatbots.

- **Benefits:**

    - Easy-to-understand interface tailored for young learners.
    - Encourages learning by doing hands-on experiments.
    - Helps demystify AI concepts in a classroom setting.

- **Drawbacks:**
    - Limited to basic models and functionality; not suitable for advanced projects.
    - Depends heavily on pre-built templates and may not provide much flexibility for deeper exploration.

2) Teachable Machine (by Google)

- **Target Audience:**
  Designed for general users, educators, students, and creatives who want to experiment with machine learning without coding.

- **Use by Target Audience:**
  Users train models using webcam images, sound, or pose data directly in the browser. It's often used in classrooms, art projects, or personal explorations to classify images, gestures, or sounds.

- **Benefits:**
    - No coding skills required.
    - Extremely user-friendly and quick to get results.
    - Allows export of models for real-world applications (e.g., TensorFlow.js or Arduino).

- **Drawbacks:**
    - Models are simple and may not generalize well to complex data.
    - Limited customization or control over algorithm details.

2. Analytic Type

- **Machine Learning for Kids:**
  Predictive Analytic
  Because it is used to build models that predict outcomes based on training data (e.g.,

predicting the correct answer based on input text or images).

- **Teachable Machine:**
  Predictive Analytic
  It also creates models that make predictions based on user-provided data (e.g., recognizing if the image is of a cat or a dog).

3. Learning Type

- **Machine Learning for Kids:**
  Supervised Learning
  The tool works by training models using labeled data (e.g., “this is happy,” “this is sad”), which is the core concept of supervised learning.

- **Teachable Machine:**
  Supervised Learning
  Users label their own training data (e.g., “Class 1,” “Class 2”), and the model learns to distinguish between them based on these labels.

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.

Anwer :

1. Key Takeaways from the Articles

a. "What’s in a Chart? A Step-by-Step Guide to Identifying Misinformation in Data Visualization" by Arthur Kakande

Article Link

Arthur Kakande emphasizes that data visualizations can be misleading due to several common practices:

- **Truncated Y-Axis:** Starting the Y-axis at a value other than zero can exaggerate differences between data points, leading to misinterpretation.

- **Correlation vs. Causation:** Visuals may imply a causal relationship where only a correlation exists, misleading viewers about the true nature of the data.
- **Color Misuse:** Using colors contrary to common associations (e.g., red for positive outcomes) can confuse the audience.
- **Trend Manipulation:** Altering scales or omitting data points can distort the perceived trend.

Kakande advocates for critical evaluation of charts, encouraging viewers to scrutinize axes, scales, and color usage to detect potential misinformation.

b. "How Bad COVID-19 Data Visualizations Mislead the Public" by Katherine Ellen Foley

Article Link

Katherine Ellen Foley discusses how, during the COVID-19 pandemic, some state health departments in the U.S. presented data in ways that hindered public understanding:

- **Snapshot Data Without Trends:** Presenting daily case numbers without historical context made it difficult to discern whether situations were improving or worsening.
- **Cluttered Visuals:** Overloading charts with information without clear organization led to confusion.
- **Ineffective Use of Pie Charts:** Pie charts were used inappropriately, making it hard for viewers to accurately interpret proportions.

Foley suggests that clear, context-rich, and appropriately designed visualizations are crucial for effective public communication.

2. Recent Event Example: Misleading COVID-19 Visualizations by Anti-Mask Groups

A study by MIT's Visualization Group highlighted how anti-mask communities created misleading data visualizations to challenge public health guidelines. These groups produced charts that mimicked scientific rigor but were based on flawed interpretations

Article Link

- **Selective Data Representation:** Focusing on data that supported their views while ignoring contradictory information.

- **Misinterpretation of Data:** Drawing conclusions that were not supported by the data, such as underestimating the severity of the pandemic.

- **Aesthetic Mimicry:** Designing visuals that appeared scientifically credible to lend unwarranted legitimacy to their claims.

These practices contributed to public confusion and resistance to health measures.

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

- Data File: Classification data.csv
- Class Label: Last Column
- Use any Machine Learning model (SVM, Naïve Base Classifier)

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used
- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized
- Use any Python library to present the accuracy measures of trained model

Pima Indians Diabetes Database

Name : Advik Hegde Div : D15C Roll No. : 15

Step 1: Set up and import the necessary libraries and upload the dataset into the runtime environment.

```
# Install necessary packages
!pip install -q imbalanced-learn scikit-learn

# Import required libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[2] # Upload the file manually if needed
from google.colab import files
uploaded = files.upload()

# Read the CSV
df = pd.read_csv('diabetes.csv')  # Change this to the uploaded filename
df.head()
```

Step 2 : Data Preprocessing

```
# Check for missing values
print(df.isnull().sum())

# Replace 0s in some columns with NaN where 0 is not a valid value
columns_with_zeros = ['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI']
df[columns_with_zeros] = df[columns_with_zeros].replace(0, np.nan)

# Fill missing values with the median
df.fillna(df.median(), inplace=True)

# Separate features and target
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Step 3 : Handling Class Imbalances using the SMOTE technique

```
# Apply SMOTE to balance the classes
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Check class distribution
print("Class distribution after SMOTE:")
print(pd.Series(y_resampled).value_counts())
```

```
Class distribution after SMOTE:
Outcome
1    500
0    500
Name: count, dtype: int64
```

Step 4: Train-Validation-Test Split (70/20/10)

```
[5] # First split into 90% train_val and 10% test
    X_train_val, X_test, y_train_val, y_test = train_test_split(
        X_resampled, y_resampled, test_size=0.10, random_state=42, stratify=y_resampled)

    # Now split 90% into 70% train and 20% validation (from the 90%)
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_val, y_train_val, test_size=2/9, random_state=42, stratify=y_train_val)
```

Step 5: Feature Scaling

```
[6] scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)
```

Step 7: Model Training with Hyperparameter Tuning

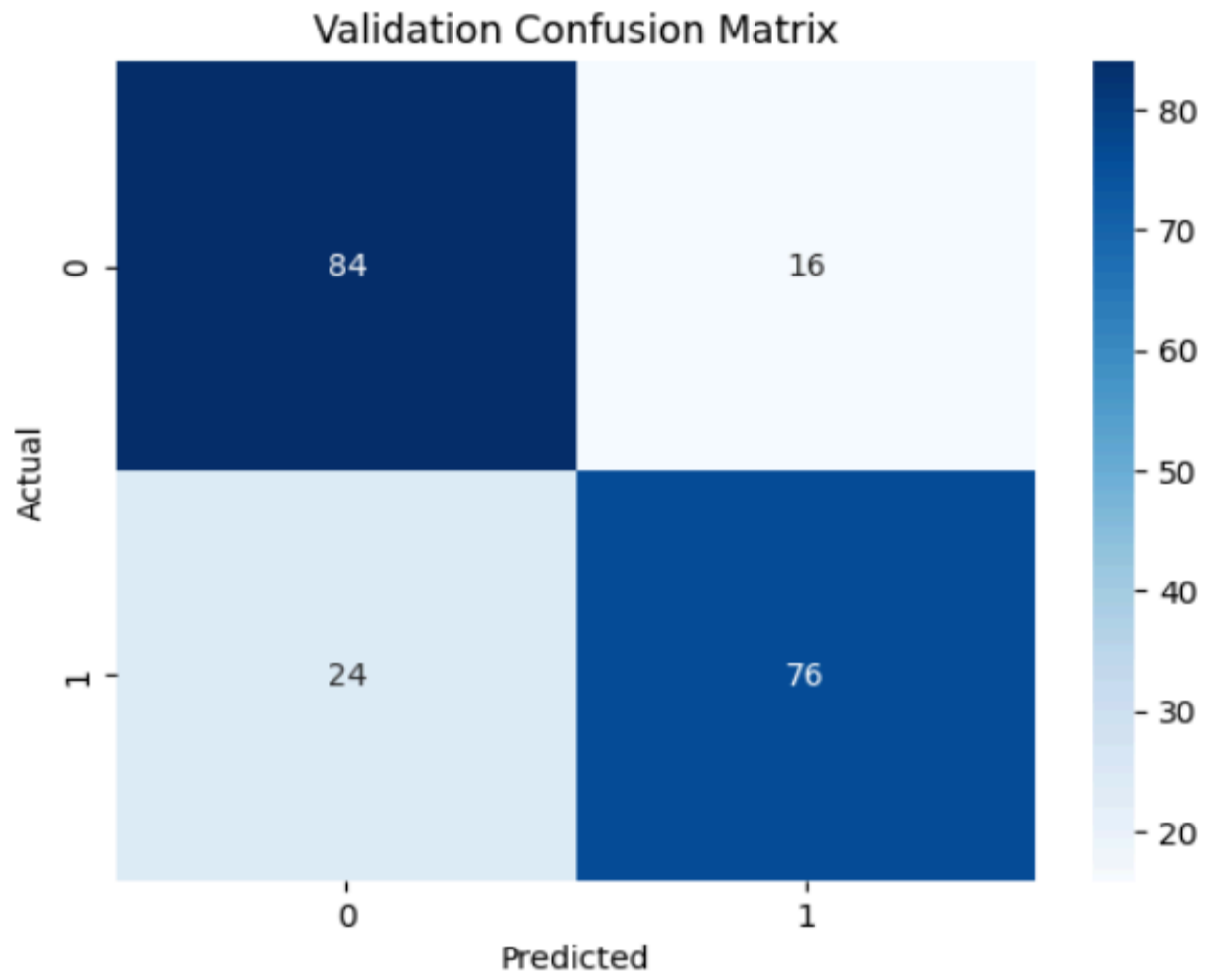
```
[7]  param_grid = {
         'C': [0.1, 1, 10],
         'kernel': ['linear', 'rbf'],
         'gamma': ['scale', 'auto']
     }

     svm = SVC()
     grid = GridSearchCV(svm, param_grid, cv=5)
     grid.fit(X_train, y_train)

     print("Best SVM Parameters:", grid.best_params_)

     # Validate
     y_val_pred = grid.predict(X_val)
```

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

Step 8: Evaluate the Model (Validation)

```
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print("\nClassification Report:\n", classification_report(y_val, y_val_pred))

# Confusion Matrix
conf_matrix = confusion_matrix(y_val, y_val_pred)
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title("Validation Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Validation Accuracy: 0.8

Classification Report:
              precision    recall  f1-score   support

           0       0.78      0.84      0.81       100
           1       0.83      0.76      0.79       100

    accuracy                           0.80       200
   macro avg       0.80      0.80      0.80       200
weighted avg       0.80      0.80      0.80       200
```



Step 9: Final Test Evaluation

```
y_test_pred = grid.predict(X_test) if 'grid' in locals() else nb.predict(X_test)

print("Test Accuracy:", accuracy_score(y_test, y_test_pred))
print("\nTest Classification Report:\n", classification_report(y_test, y_test_pred))
```

```
Test Accuracy: 0.85

Test Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.84      0.85        50
           1       0.84      0.86      0.85        50

    accuracy                           0.85       100
   macro avg       0.85      0.85      0.85       100
weighted avg       0.85      0.85      0.85       100
```

Name : Advik Hegde **Div : D15C** **Roll No. : 15**

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 1st column.

Requirements to satisfy:

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

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

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

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

(Refer any one)

Answer :

Import Libraries:

- Essential libraries like pandas, numpy, train_test_split, GridSearchCV, StandardScaler, XGBRegressor, and r2_score are imported. These libraries handle data processing, model training, hyperparameter tuning, and performance evaluation.

Load Dataset:

- The dataset is read from the Excel file Real estate valuation data set.xlsx using pd.read_excel().
- The column No (index column) is dropped as it's not relevant for the model.

Separate Features and Target:

- The features (X) are separated by dropping the target column 'Y house price of unit area', which is stored in y.

- X contains the input variables, and y contains the output (target variable).

Standardize Features:

- The features X are standardized using StandardScaler(). This step ensures that all features have a mean of 0 and a standard deviation of 1, which is important for models like gradient boosting.

Train/Test Split:

- The dataset is split into training (70%) and test (30%) sets using train_test_split(). This step is crucial for evaluating the model's performance on unseen data.

Define Model and Hyperparameters:

- An XGBRegressor model is defined, and hyperparameters like learning_rate, max_depth, and n_estimators are set for tuning.
- A GridSearchCV is used to search through different combinations of these hyperparameters to find the best performing model, with cross-validation (5-fold) and r2 scoring.

Train the Model:

- The model is trained using the training data (X_train, y_train) and the GridSearchCV method. The grid search will try different hyperparameter combinations and select the best one based on R^2 score.

Evaluate Performance:

- The best model from grid_search is used to make predictions on the test data (X_test).
- The R^2 score (r2_score()) is computed to measure how well the model explains the variance in the test set.
- The Adjusted R^2 score is calculated to adjust the R^2 score for the number of features used, providing a more reliable performance metric for models with multiple predictors.

Name : Advik Hegde **Div : D15C** **Roll No. : 15**

```python
# 1. Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
from xgboost import XGBRegressor

# 2. Load the dataset
data = pd.read_excel('Real estate valuation data set.xlsx')
data = data.drop(columns=['No'])  # Remove index column if present

# 3. Separate features and target
X = data.drop(columns=['Y house price of unit area'])
y = data['Y house price of unit area']

# 4. Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 5. Train/test split (70/30)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42
)
```

```python
# 6. Define model and hyperparameters
param_grid = {
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'n_estimators': [100, 150, 200]
}

grid_search = GridSearchCV(
    XGBRegressor(objective='reg:squarederror', random_state=42),
    param_grid,
    scoring='r2',
    cv=5,
    verbose=1,
    n_jobs=-1
)

# 7. Train the model
grid_search.fit(X_train, y_train)

# 8. Evaluate performance
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)

r2 = r2_score(y_test, y_pred)
n = X_test.shape[0]
p = X_test.shape[1]
adjusted_r2 = 1 - (1 - r2) * ((n - 1) / (n - p - 1))

print("R² Score:", round(r2, 4))
print("Adjusted R² Score:", round(adjusted_r2, 4))
print("Best Parameters:", grid_search.best_params_)
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
R² Score: 0.7258
Adjusted R² Score: 0.7119
Best Parameters: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100}
```

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).

Answer :

Key Features of the Wine Quality Dataset

The **Wine Quality Dataset** includes several key features that influence the overall quality of the wine. These features are:

- **Fixed Acidity**: Affects the wine's freshness and preservation.
- **Volatile Acidity**: Contributes to a sharp, vinegar-like taste.
- **Citric Acid**: Enhances the freshness and balance of the wine.
- **Residual Sugar**: Determines the sweetness level and balance.
- **Chlorides**: Influences the salty taste and flavor balance.
- **Free Sulfur Dioxide**: Preserves the wine and affects stability.
- **Total Sulfur Dioxide**: Affects preservation and taste.
- **Density**: Indicates alcohol content and the mouthfeel of the wine.
- **pH**: Impacts the wine's stability and aging potential.
- **Sulphates**: Affects preservation and aroma.
- **Alcohol**: Contributes to the body and flavor complexity of the wine.
- **Quality (Target)**: A score from 0 to 10 representing the wine’s overall quality.

Importance of Features in Predicting Wine Quality

- **Acidity-related features** like **Fixed Acidity**, **Volatile Acidity**, and **pH** are crucial for determining the balance and freshness of wine, which are key indicators of quality.

- **Residual Sugar** and **Alcohol** significantly impact sweetness, body, and the perceived quality of the wine.

- **Sulfur Dioxide**, **Chlorides**, and **Sulphates** play an important role in wine preservation and stability.

- **Density** and **Citric Acid** provide information about the mouthfeel, aroma, and overall balance, which also contribute to quality.

Handling Missing Data

During the feature engineering process, **missing data** in the Wine Quality Dataset can be handled using the following techniques:

- **Mean/Median Imputation**:
    - **Advantages**: Simple and quick, good for numerical features with small amounts of missing data.
    - **Disadvantages**: Can introduce bias, especially if data isn't missing randomly, and reduces variance.
- **KNN Imputation**:
    - **Advantages**: Utilizes the similarity between data points, providing more accurate imputation.
    - **Disadvantages**: Computationally expensive, especially for large datasets.
- **Regression Imputation**:
    - **Advantages**: Predicts missing values based on relationships between features, more accurate than mean/median imputation.
    - **Disadvantages**: Requires building a predictive model, which can lead to overfitting.
- **Model-based Imputation** (e.g., using Random Forest):
    - **Advantages**: Very accurate as it leverages complex patterns in the data.

- **Disadvantages**: Computationally intensive and requires more resources.

In summary, each feature in the dataset is vital in determining the quality of wine, and the choice of imputation technique depends on the dataset's nature and the desired trade-off between computational efficiency and accuracy.