Name : Dev Gaonkar **Div : D15C** **Roll No: 12**

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 : Dev Gaonkar **Div : D15C** **Roll No : 12**

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

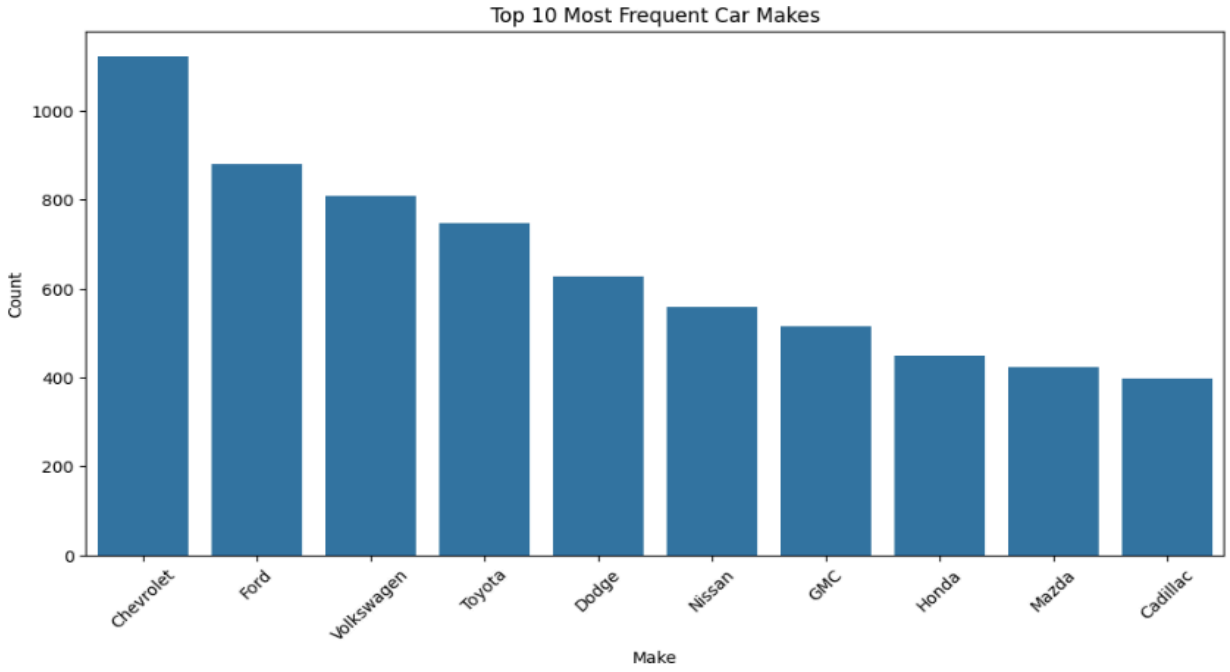
Seaborn is built on top of Matplotlib and provides a high-level interface for creating visually appealing and informative statistical graphics. Key capabilities include:

- **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)
```



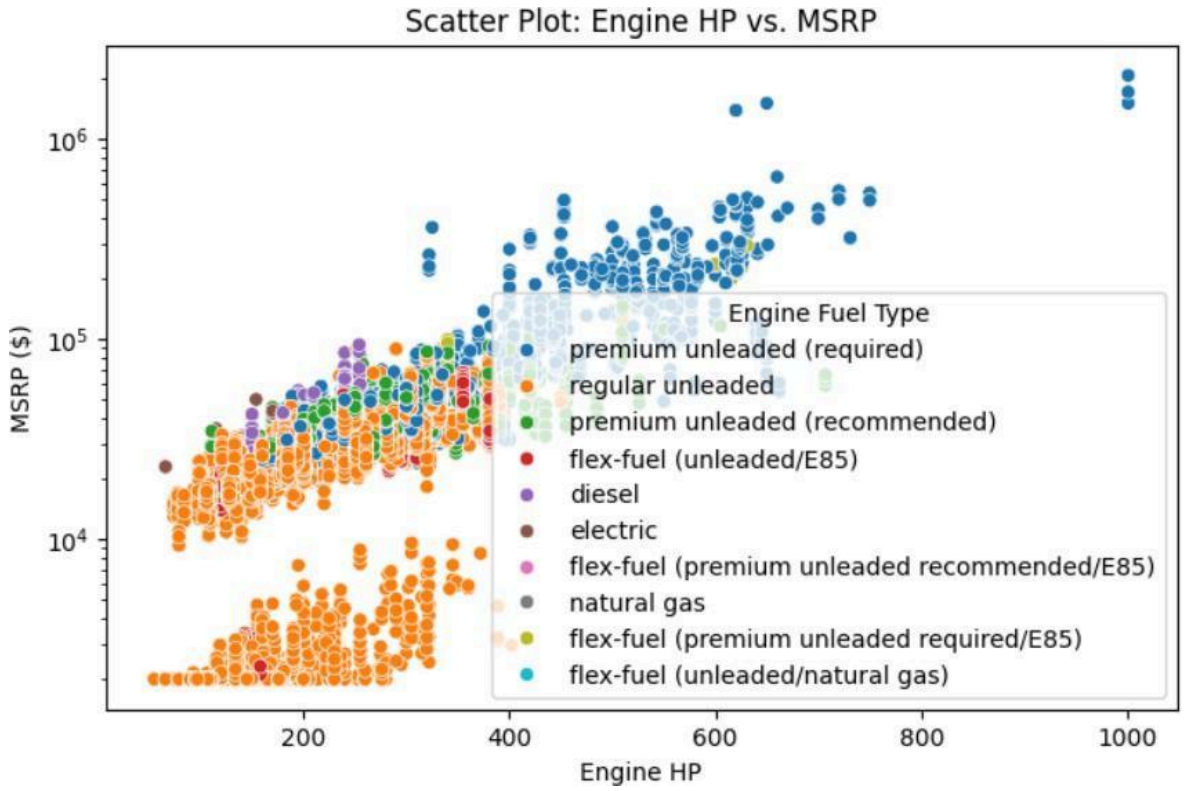

```
Contingency Table:
 Driven_Wheels      all wheel drive  four wheel drive  front wheel drive  \
Transmission Type
AUTOMATED_MANUAL               198                 0                304
AUTOMATIC                     1940              1056               3056
DIRECT_DRIVE                    11                 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()
```

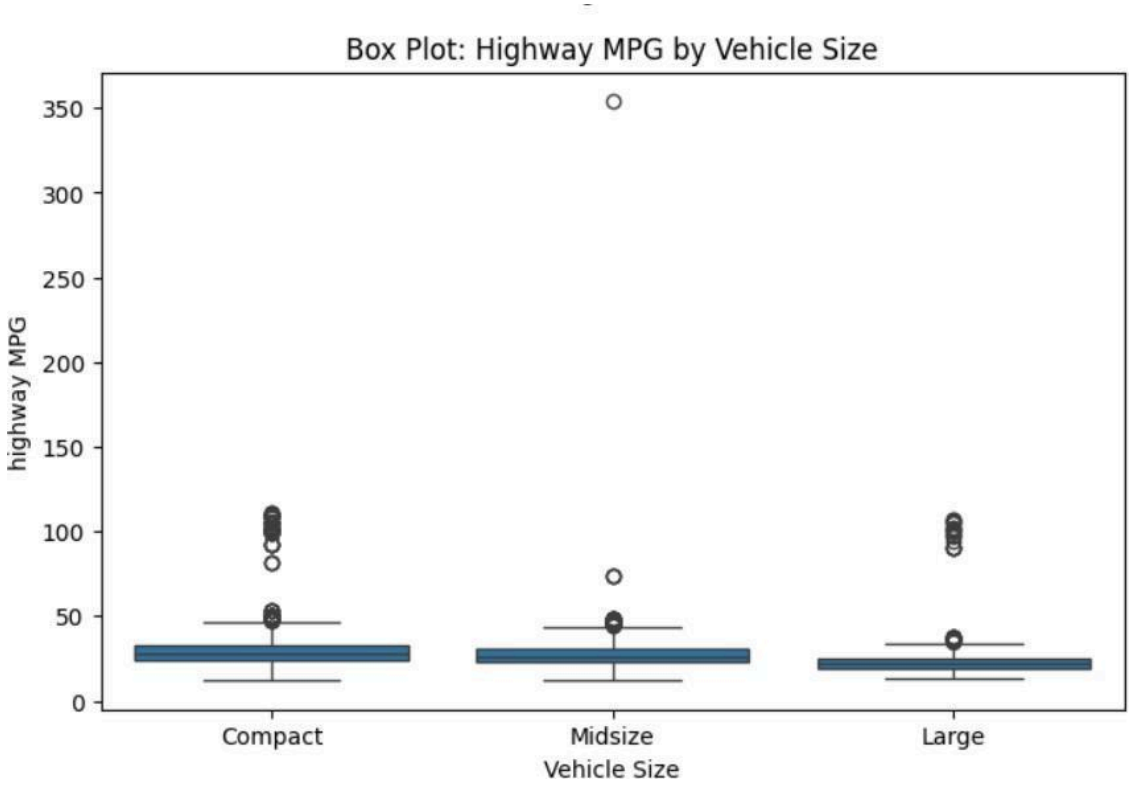
A **scatter plot** is a graph used to visualize the relationship between two continuous variables. It helps to identify correlations and trends in data. In this dataset, the scatter plot visualizes the relationship between Engine HP and MSRP, showing whether more powerful engines are associated with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



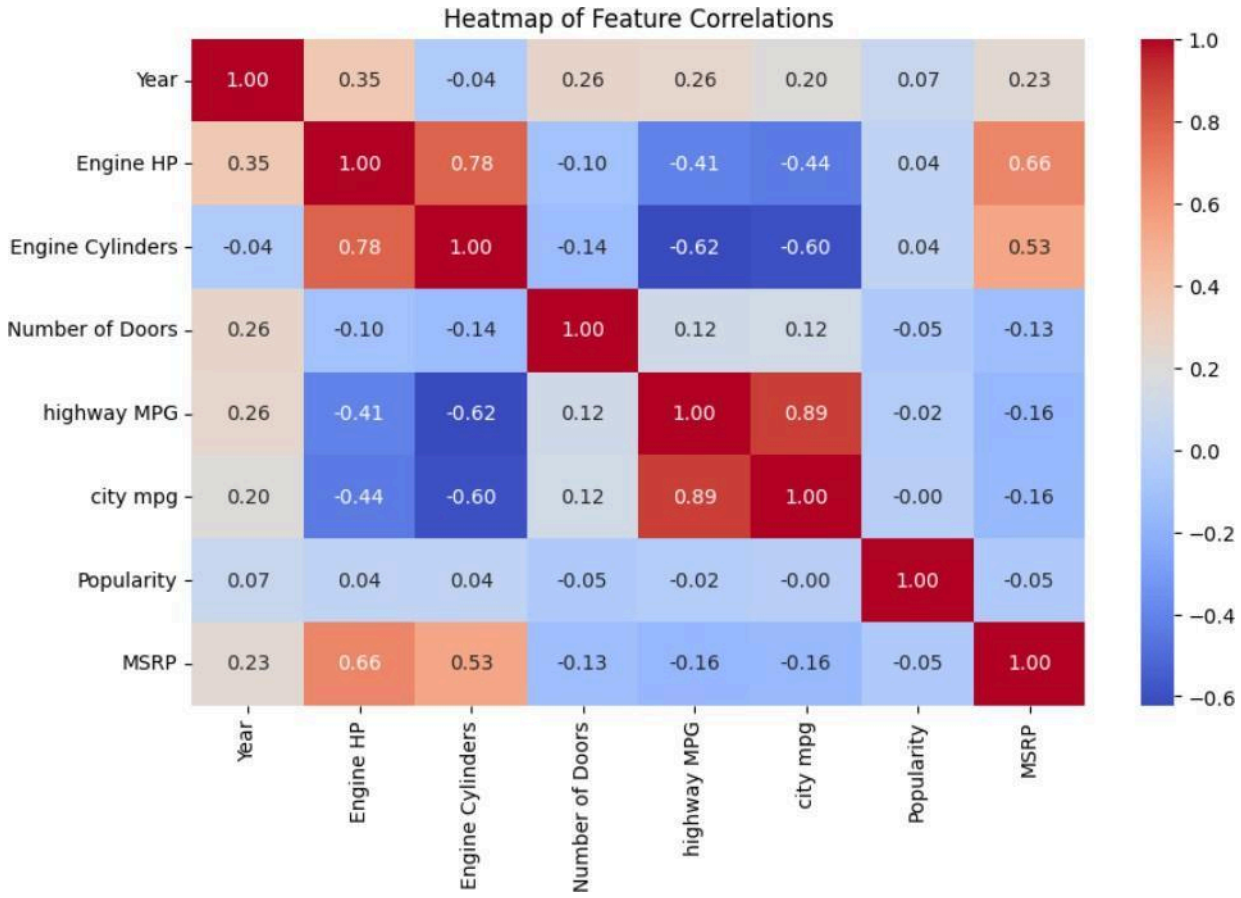
A **box plot** is used to show the distribution of data based on quartiles, highlighting the median, spread, and potential outliers.

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.

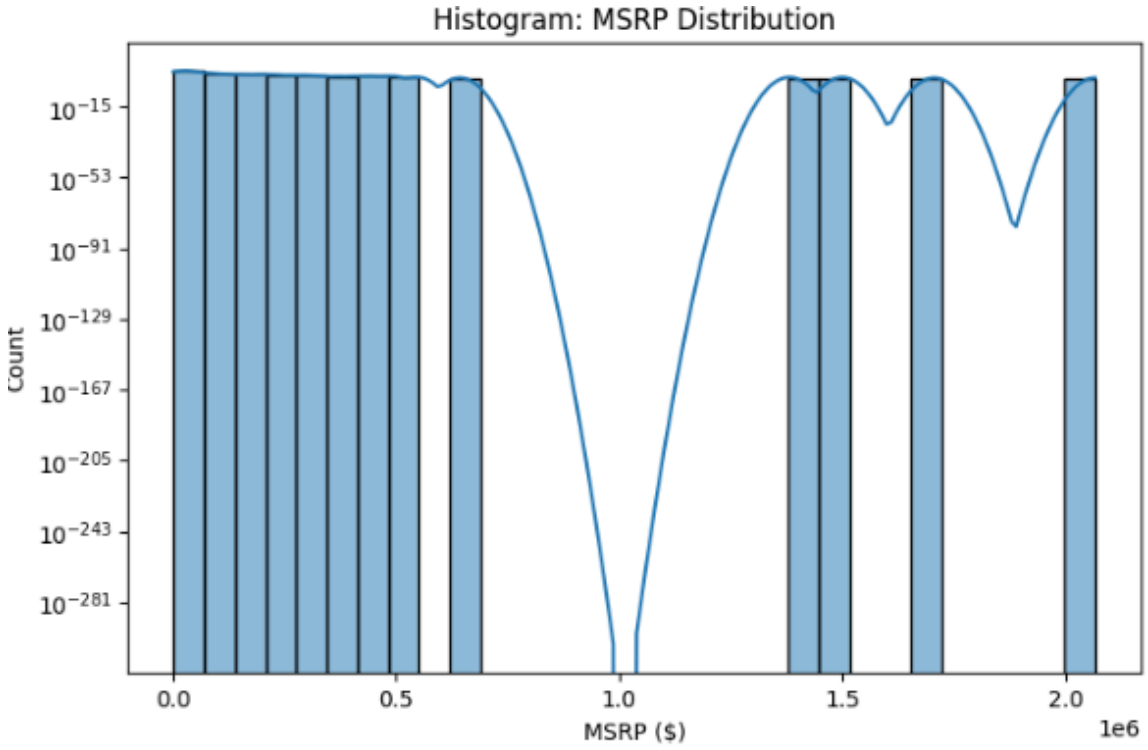
```
## 3. Histogram and Normalized Histogram
# Histogram: MSRP Distribution
plt.figure(figsize=(8, 5))
sns.histplot(df['MSRP'], bins=30, kde=True)
plt.xlabel("MSRP ($)")
plt.ylabel("Count")
plt.title("Histogram: MSRP Distribution")
plt.yscale('log')  # Log scale for better visualization
plt.show()

# 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()
```

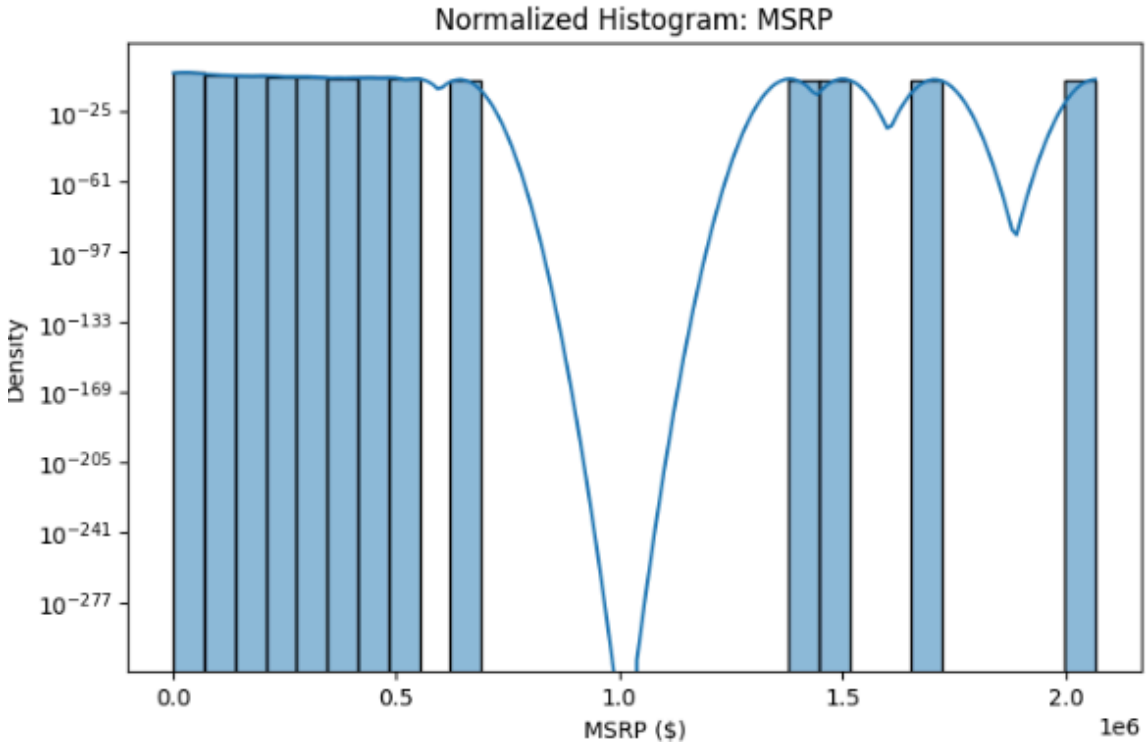
A histogram is a graphical representation that shows the distribution of a dataset by grouping data into bins. It is particularly useful for visualizing the frequency of values within a continuous range.

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.

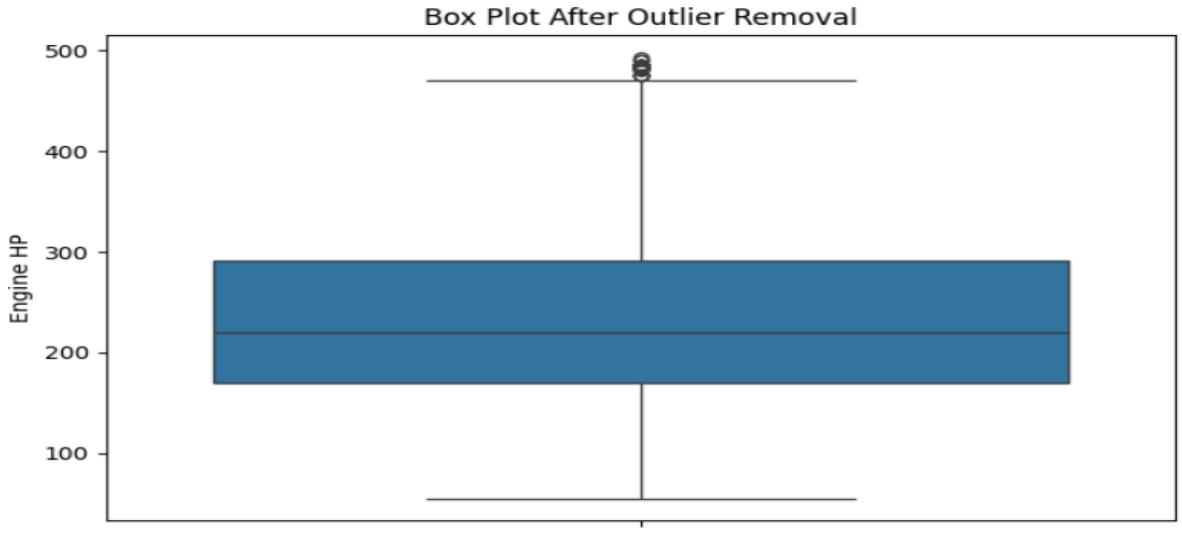
```
## 4. Handling Outliers using Box Plot and IQR
# Detecting Outliers in Engine HP using IQR
Q1 = df['Engine HP'].quantile(0.25)
Q3 = df['Engine HP'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# 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 : Dev Gaonkar **Div : D15C** **Roll No : 12**

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)
```

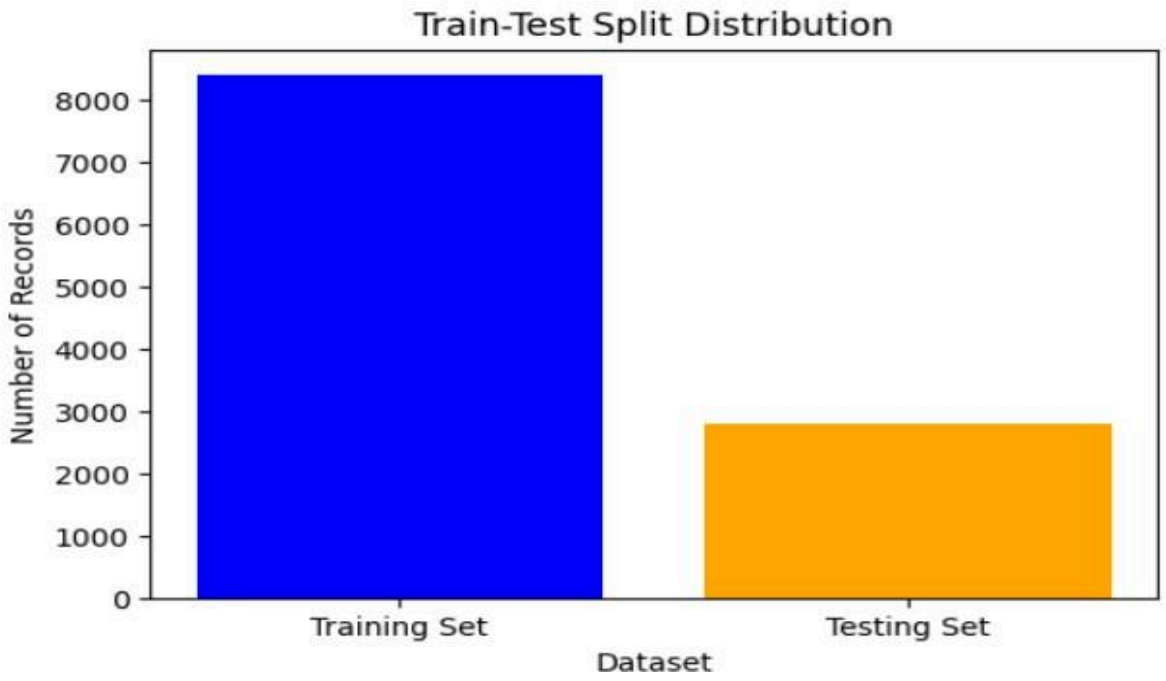
```
Training set size: (8395, 15)
Testing set size: (2799, 15)
```

Step 2: Visualizing the Data Split

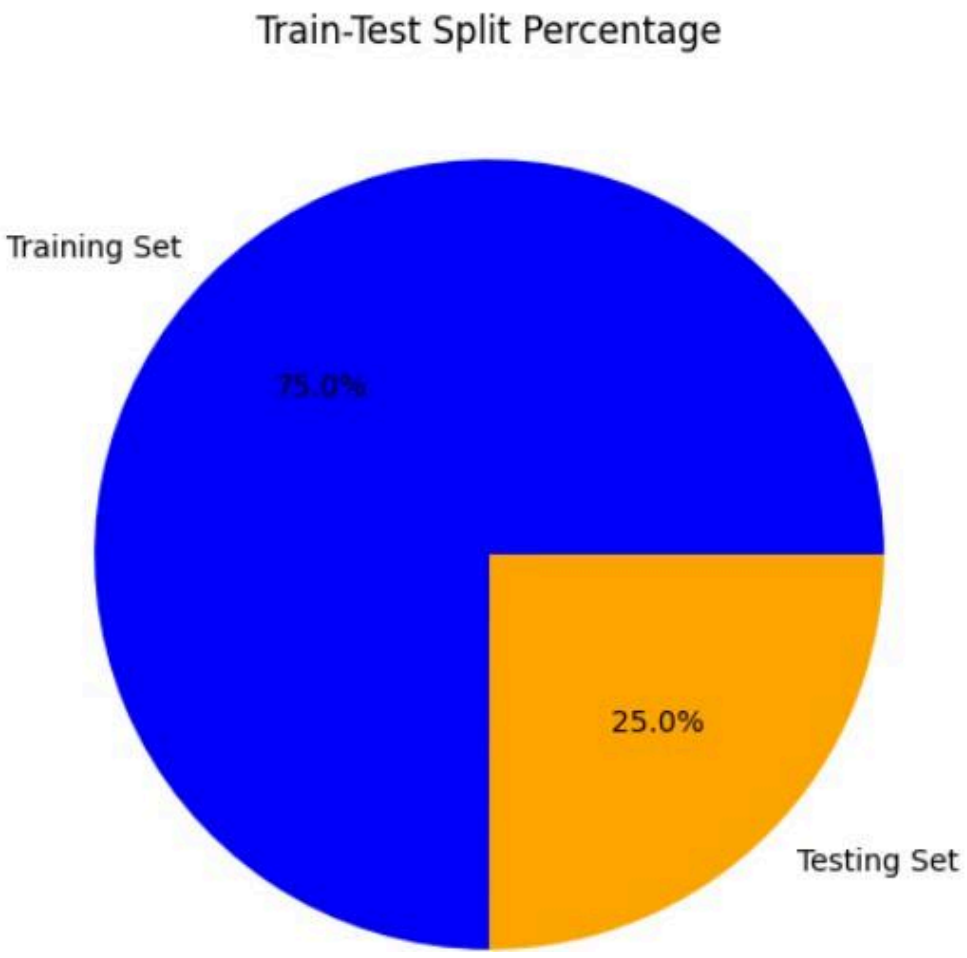
To confirm the partitioning proportions, the following visualizations were generated:

- **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 : Dev Gaonkar **Div : D15C** **Roll No : 12**

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 : Dev Gaonkar **Div : D15C** **Roll No : 12**

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 : Dev Gaonkar **Div : D15C** **Roll No : 12**

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.

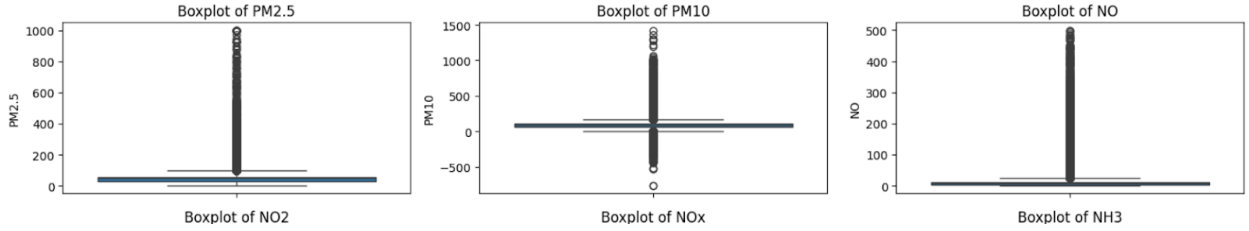
```
from sklearn.impute import SimpleImputer
# Mean Imputation - <20% Missing
mean_imputer = SimpleImputer(strategy='mean')
for col in ['PM2.5', 'NO', 'NO2', 'NOx', 'CO']:
    df[col] = mean_imputer.fit_transform(df[[col]])
```

```
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())
```

```
[ ]  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.

```
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.

```
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: Dev Gaonkar **Div:** D15C **Rollno:** 12

Experiment 7

Aim: To implement different clustering algorithms.

Theory:
Clustering is an unsupervised machine learning technique used to group similar data points together. The objective is to discover natural groupings within a dataset without prior knowledge of class labels. Clustering is widely used in customer segmentation, anomaly detection, image segmentation, and bioinformatics.

Types of Clustering

1. Partition-based Clustering (e.g., K-Means)
    - Divides data into a predefined number of clusters.
    - Each data point belongs to exactly one cluster.
2. Density-based Clustering (e.g., DBSCAN)
    - Forms clusters based on dense regions in the data.
    - Can identify clusters of arbitrary shape and detect noise.
3. Hierarchical Clustering
    - Builds a tree of clusters using a bottom-up (agglomerative) or top-down (divisive) approach.
4. Model-based Clustering
    - Assumes data is generated by a mixture of underlying probability distributions (e.g., Gaussian Mixture Models).

K Means Clustering:
K-Means is a partition-based clustering algorithm that divides data into K clusters.

Algorithm Steps:

1. Choose the number of clusters, K.
2. Randomly initialize K centroids.
3. Assign each data point to the nearest centroid.
4. Update centroids by computing the mean of points in each cluster.
5. Repeat steps 3 & 4 until convergence (when centroids stop changing significantly).

Key Considerations:
Choosing K: The Elbow method and Silhouette Score help determine the optimal number of clusters.
Limitations: Sensitive to initial centroid placement and does not work well with non-spherical clusters or varying densities.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise):
DBSCAN is a density-based clustering algorithm that groups points closely packed together while marking low-density points as noise.

Algorithm Steps:

1. Set parameters:
2. ε (epsilon): Maximum distance to be considered a neighbor.
3. MinPts: Minimum points required to form a dense region.
4. Choose an unvisited point and determine its ε-neighborhood.
5. If the point has at least MinPts neighbors, a new cluster is formed.
6. Expand the cluster by adding density-reachable points.
7. Repeat for all points, marking outliers as noise.

Advantages:

- Handles clusters of arbitrary shape.
- Automatically detects noise and outliers.
- No need to specify the number of clusters.

Disadvantages:

- Choosing optimal ε and MinPts is challenging.
- Struggles with varying density clusters.

Steps:

1. **Load and preprocess the dataset.**

Dataset Shape: (235394, 22)
First 5 rows:

	Datetime	PM2.5	PM10	NO	NO2	NOx	NH3	CO	SO2	O3	...	AQI	AQI_Bucket
0	2015-01-01 01:00:00	0.335245	-0.20687	-1.947838	0.927657	0.617732	-0.208955	-0.313643	0.422827	-0.288155	...	0.459567	1.0
1	2015-01-01 02:00:00	0.335245	-0.20687	-2.343506	0.456358	-0.226460	-0.208955	-0.435624	0.422827	-0.288155	...	0.459567	1.0
2	2015-01-01 03:00:00	0.335245	-0.20687	-2.343506	-0.004131	-0.665298	-0.208955	-0.428156	0.422827	-0.288155	...	0.459567	1.0
3	2015-01-01 04:00:00	0.335245	-0.20687	-2.343506	-0.206178	-0.760676	-0.208955	-0.400772	0.422827	1.589661	...	0.459567	1.0
4	2015-01-01 05:00:00	0.335245	-0.20687	-2.343506	-0.329610	-0.829165	-0.208955	-0.423177	0.422827	-0.288155	...	0.459567	1.0

5 rows × 22 columns

City_Aizawl	City_Amaravati	City_Amritsar	City_Bengaluru	City_Bhopal	City_Brajrajnagar	City_Chandigarh	City_Chennai
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False

The dataset represents **air quality measurements**, containing:

- **Temporal Information**: Datetime (Timestamp of the recorded data).
- **Pollutant Concentrations**: PM2.5, PM10, NO, NO2, NOx, NH3, CO, SO2, O3, Benzene, Toluene (Air pollutants measured in the environment).
- **Air Quality Index (AQI)**: AQI, AQI_Bucket (Overall air quality rating based on pollutant levels).
- **City-wise Presence**: City_Aizawl, City_Amaravati, City_Amritsar, City_Bengaluru, City_Bhopal, City_Brajrajnagar, City_Chandigarh, City_Chennai (Binary indicators showing if the data belongs to a specific city).

2. **Elbow method for number of clusters**

The Elbow Method plot helps determine the optimal number of clusters (K) in K-Means by plotting inertia (sum of squared distances to cluster centers) against different K values. The "elbow" point, where inertia reduction slows significantly, indicates the ideal K.

Formula to calculate:

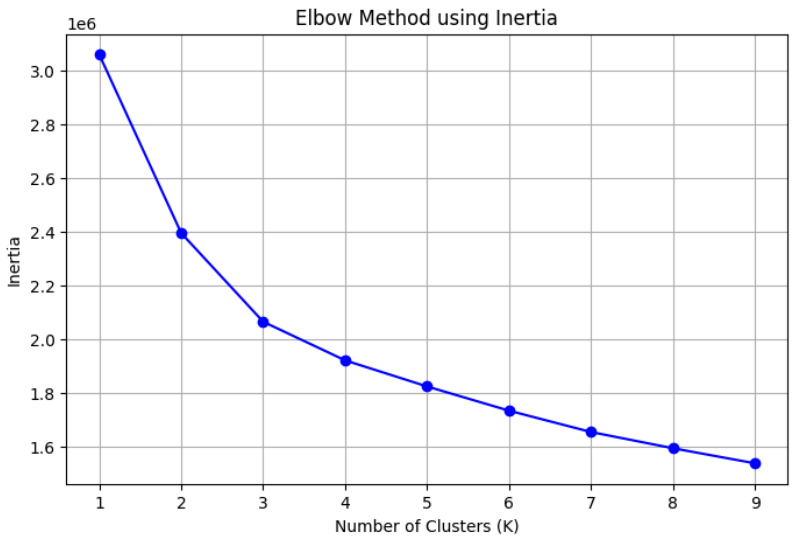
$$WCSS = \sum_{i=1}^{K} \sum_{x \in C_i} ||x - \mu_i||^2$$

where,

- Ci = Cluster i
- μi = Centroid of cluster Ci
- || xi - μi ||² = Squared Euclidean distance between a point and its cluster centroid

The **elbow point** is where the WCSS starts decreasing at a slower rate.

```
# 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()
```



In this plot, the elbow appears around K = 3 or 4.The dataset likely has 3 or 4 naturally distinct groups, and choosing K balances segmentation quality and computational efficiency

3. K means clustering

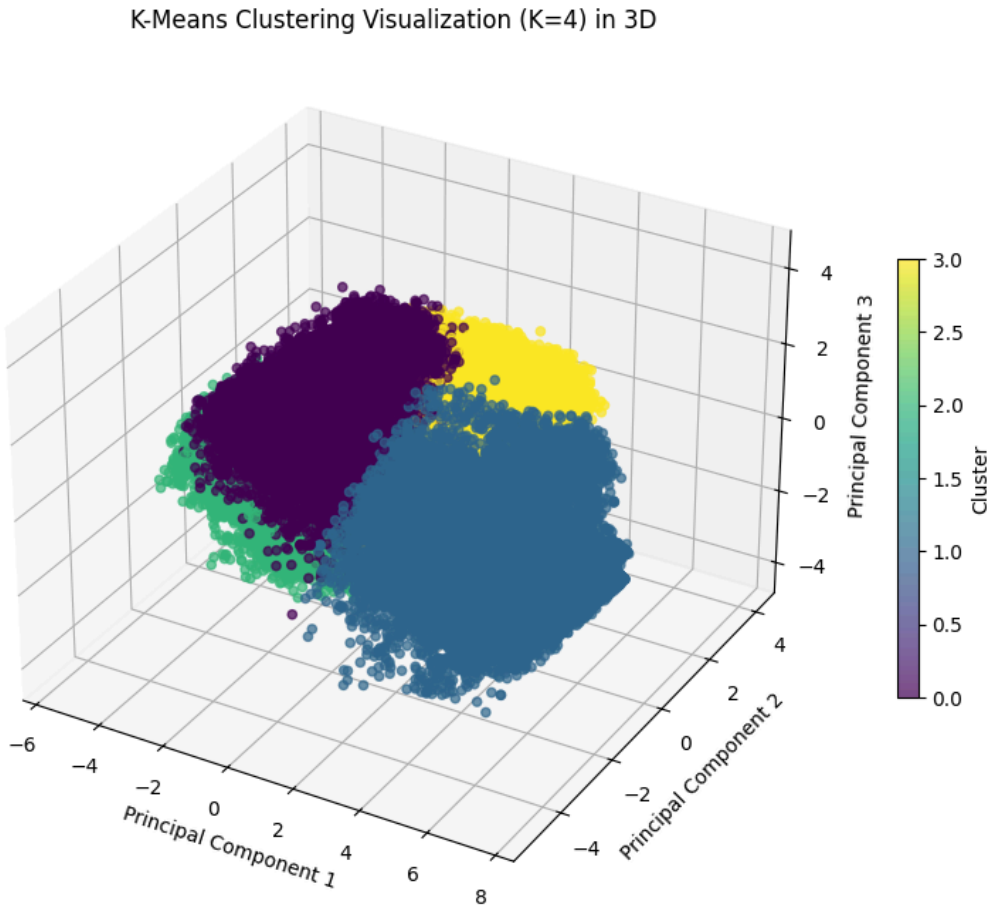
```
# Apply K-Means for K=3
kmeans3 = KMeans(n_clusters=3, random_state=42, n_init=10)
df_scaled["Cluster_K3"] = kmeans3.fit_predict(df_scaled)

# Apply K-Means for K=4
kmeans4 = KMeans(n_clusters=4, random_state=42, n_init=10)
df_scaled["Cluster_K4"] = kmeans4.fit_predict(df_scaled)

# Print cluster sizes
print("Cluster Sizes for K=3:\n", df_scaled["Cluster_K3"].value_counts())
print("\nCluster Sizes for K=4:\n", df_scaled["Cluster_K4"].value_counts())
```

```
Cluster Sizes for K=3:
 Cluster_K3
1    106713
2    104170
0     24511
Name: count, dtype: int64

Cluster Sizes for K=4:
 Cluster_K4
3    91263
2    80379
0    39407
1    24345
Name: count, dtype: int64
```

We try out clusters for k = 3 and 4. With k = 3, we find that the cluster size of the last cluster is very small as compared to the other clusters formed. So we use k = 4, we can observe that the clusters formed are similar in size.

Visualization of clusters:

```
# 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()
```



To visualize the clusters we perform **Principal Component Analysis (PCA)** to reduce the dataset dimensions to three, making it easier to visualize the clusters obtained from K-Means clustering. The **3D scatter plot** visualizes the clusters in the transformed feature space, where different colors represent different clusters. The clusters appear to be well-defined, indicating that the K-Means algorithm successfully grouped similar data points.
The PCA components **help analyze the contribution** of original features to the new principal components, providing insights into how data is distributed across clusters.

4. DBSCAN clustering

DBSCAN is an unsupervised clustering algorithm that groups together densely packed points while marking outliers as noise. Unlike K-Means, it does not require specifying the number of clusters in advance and is useful for identifying arbitrarily shaped clusters.
1. Defines Two Parameters:

- ε (epsilon): The radius to consider neighboring points.
- MinPts: Minimum number of points required in an ε-neighborhood to form a cluster.

2. Classifies Points as:

- Core Points: Points with at least MinPts neighbors within ε.
- Border Points: Points within ε of a core point but with fewer than MinPts neighbors.
- Noise Points: Points that are neither core nor border

(outliers).

3.Clustering Process:

- Starts with an unvisited point.
- Expands a cluster if it's a core point, connecting all reachable points.

Repeats until all points are classified.

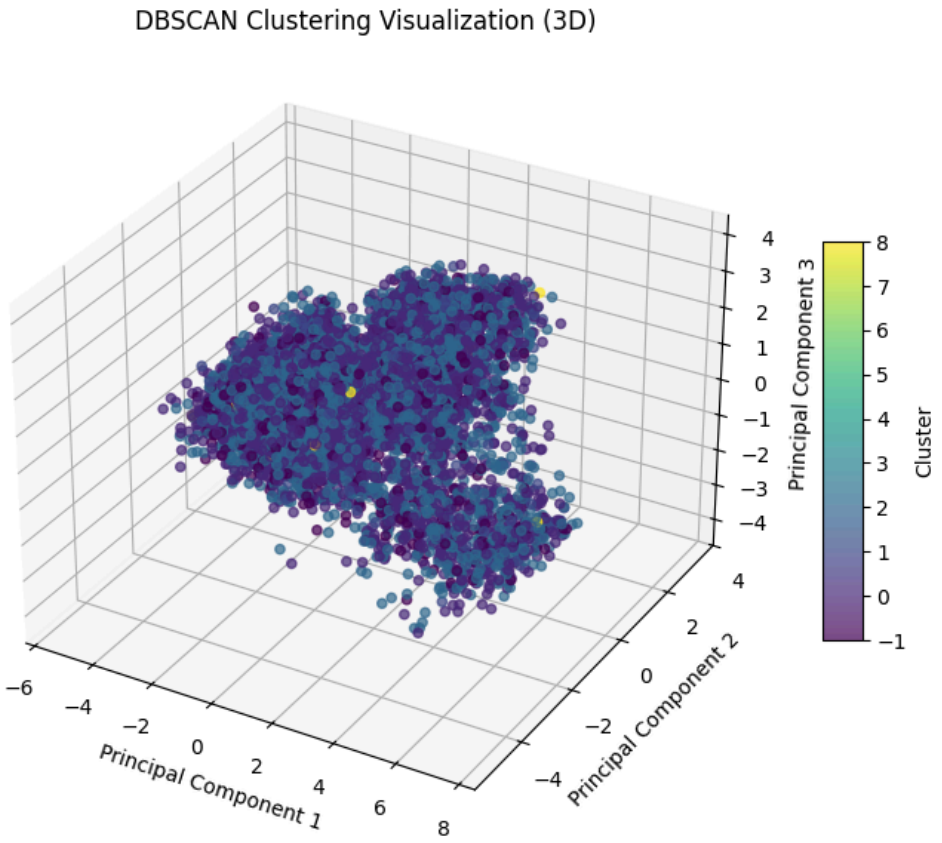
```
from sklearn.cluster import DBSCAN

eps_value = 2.0
min_samples_value = 30

# Apply DBSCAN
dbscan = DBSCAN(eps=eps_value, min_samples=min_samples_value, n_jobs=-1)
clusters = dbscan.fit_predict(df_sample)

# Add clusters to the DataFrame
df_sample["Cluster"] = clusters

# Display cluster distribution
print("Cluster Counts:\n", df_sample["Cluster"].value_counts())
```

```
Cluster Counts:
 Cluster
 0    34979
 1     7803
 2     4038
-1     2526
 4      163
 5      143
 3      118
 7       86
 6       76
 9       35
 8       33
Name: count, dtype: int64
```

We apply **DBSCAN clustering** with **eps=2** and **min_samples=30**, grouping data points based on density. It assigns cluster labels using dbscan.fit_predict(df_scaled) and stores them in the dataset. The output shows that most points (34,979) belong to **Cluster 0**, followed by **Cluster 1 (7,803)** and **Cluster 2 (4,038)**, while **2,526 points are classified as noise (-1)**.

The moderate number of noise points (-1) suggests that **eps=2** captures some dense regions but may still be slightly restrictive, leaving certain points unclustered. Additionally, **min_samples=30** ensures that only regions with sufficient density form clusters, leading to a few well-defined clusters while filtering out noise. As a result, the dataset forms multiple clusters of varying sizes, with most points concentrated in a few large groups.

Visualization:



The DBSCAN clustering visualization in 3D PCA space shows that the majority of points are assigned to cluster -1 (noise) with only a few points belonging to distinct smaller clusters. This suggests that DBSCAN's parameters (epsilon and min_samples) may not be well-tuned for clear separation, leading to most points being grouped together while only a few scattered regions are identified as separate clusters.

Inference from both the clustering:

Overall we can see that the performance of **K means** is **better** than DBSCAN for clustering the dataset.

K-Means was a better choice for this dataset because the data is well-distributed without high-density regions, and clusters can be separated by distance rather than density.

DBSCAN struggled due to high dimensionality and the lack of naturally dense clusters, causing excessive noise detection.

Conclusion:

In all, we come to know that K-Means and DBSCAN analyze airline passenger satisfaction data. K-Means (K=4) successfully grouped data into well-separated clusters, confirmed by the Elbow Method and PCA visualization, though it is sensitive to outliers. DBSCAN, however, struggled with this dataset, classifying many points as noise due to a lack of well-defined dense regions, making it less effective for this case.

Experiment 8 – Recommendation

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

We chose **K-Means Clustering** to build a hybrid recommendation system on the MovieLens 100K dataset, predicting movies users might like based on genres and ratings. K-Means clusters movies by genres (e.g., Animation, Action), enabling content-based filtering (e.g., "Toy Story" with "Lion King"). It's unsupervised, fitting the dataset's structure (1682 movies, 19 genre features), and its elbow method (K=6) ensured optimal clustering. We added a collaborative layer by sorting clusters by average ratings, creating a hybrid system. K-Means' silhouette score (0.354) confirmed decent clustering, outperforming alternatives like Regression, which require labeled data.

Theory:

Recommendation types and measures.

Recommendation systems suggest items to users using various approaches. Content-based filtering recommends items based on their features, such as movie genres (e.g., suggesting "Lion King" for "Toy Story" due to shared Animation/Children's genres). Collaborative filtering uses user behavior, like ratings, to find patterns (e.g., users who liked "Star Wars" also liked "Empire Strikes Back"). Hybrid filtering combines both, improving relevance by balancing item similarity and user preferences. Measures include quantitative metrics like silhouette score (0.354 in our case, assessing clustering quality) and qualitative checks, such as genre relevance (e.g., ensuring "Toy Story" recs match its Animation genre) and rating quality (recs averaging 4.2–5.0).

Types:

- Content-based: Our K-Means clustering on genres.
- Collaborative: Sorting by average ratings within clusters.
- Hybrid: Combining both in our system.

The **silhouette score** is a metric used to evaluate the quality of clusters generated by K-means clustering (or other clustering algorithms). It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from -1 to 1: values close to 1 indicate that the data points are well-clustered, with points closer to their own cluster than to others,

while values near 0 suggest overlapping clusters, and negative values imply misclassified points. It helps in determining the optimal number of clusters and assessing how well the algorithm has performed.

Measures:

- Quantitative: Silhouette score (0.354) for clustering.
- Qualitative: Genre match and high ratings of recs.

Dataset Description:

We used the **MovieLens 100K dataset**, a standard benchmark for recommendation systems, containing 100,000 ratings from 943 users on 1682 movies. It includes two key files: u.data (ratings: user ID, movie ID, rating 1–5, timestamp) and u.item (movie details: ID, title, 19 binary genre features like Action, Comedy). This dataset fits the professor's "content type data" hint (genres) and "customer review-like" requirement (ratings), providing both content-based (genres) and collaborative (ratings) data for our hybrid system. We accessed it via wget for efficiency.

Steps:

1. Fetch and Load Data:

```
import pandas as pd

# Load ratings
ratings = pd.read_csv('ml-100k/u.data', sep='\t', names=['user_id', 'movie_id', 'rating', 'timestamp'])

# Load movies (genres)
# The file has 24 columns, we specify 24 names and read the first 24 columns
movies = pd.read_csv('ml-100k/u.item', sep='|', encoding='latin-1',
                     names=['movie_id', 'title', 'release_date', 'video_release_date',
                            'IMDb_URL'] + [f'genre_{i}' for i in range(19)],
                     usecols=range(24))
```

```
print("Movies loaded:", movies.shape)
print(movies.head())
```

```
Movies loaded: (1682, 24)
   movie_id              title release_date  video_release_date  \
0         1   Toy Story (1995)  01-Jan-1995                 NaN
1         2   GoldenEye (1995)  01-Jan-1995                 NaN
2         3  Four Rooms (1995)  01-Jan-1995                 NaN
3         4  Get Shorty (1995)  01-Jan-1995                 NaN
4         5      Copycat (1995)  01-Jan-1995                 NaN

                                        IMDb_URL  genre_0  genre_1  \
0  http://us.imdb.com/M/title-exact?Toy%20Story%2...        0        0
1  http://us.imdb.com/M/title-exact?GoldenEye%20(...        0        1
2  http://us.imdb.com/M/title-exact?Four%20Rooms%...        0        0
3  http://us.imdb.com/M/title-exact?Get%20Shorty%...        0        1
4  http://us.imdb.com/M/title-exact?Copycat%20(1995)        0        0

   genre_2  genre_3  genre_4  ...  genre_9  genre_10  genre_11  genre_12  \
0        0        1        1  ...        0         0         0         0
1        1        0        0  ...        0         0         0         0
2        0        0        0  ...        0         0         0         0
3        0        0        0  ...        0         0         0         0
4        0        0        0  ...        0         0         0         0

   genre_13  genre_14  genre_15  genre_16  genre_17  genre_18
0         0         0         0         0         0         0
1         0         0         0         1         0         0
2         0         0         0         1         0         0
3         0         0         0         0         0         0
4         0         0         0         1         0         0

[5 rows x 24 columns]
```

2. Preprocess Features and Ratings: Extracted 19 genre columns for clustering and computed average ratings per movie from 100k ratings, merging them into a unified dataset (1682 rows).

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

# Extract genre features again (just to be safe)
genre_cols = [f'genre_{i}' for i in range(19)]
X = data[genre_cols]

# Elbow method to pick K
inertias = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)

# Plot elbow curve
plt.plot(K_range, inertias, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for K-Means')
plt.show()
```

3. Cluster Movies with K-Means: Applied K-Means (K=6, chosen via elbow method) on genre features, grouping movies into 6 clusters based on genre similarity (e.g., Animation, Sci-Fi).

```
# Cluster with K=6
kmeans = KMeans(n_clusters=6, random_state=42)
data['cluster'] = kmeans.fit_predict(X)
print("Cluster assignments:")
print(data[['title', 'cluster']].head())
print("\nCluster sizes:")
print(data['cluster'].value_counts())
```

```
Cluster assignments:
                title  cluster
0   Toy Story (1995)        0
1   GoldenEye (1995)        2
2  Four Rooms (1995)        5
3  Get Shorty (1995)        4
4     Copycat (1995)        1

Cluster sizes:
cluster
1    620
4    403
3    279
5    221
2    116
0     43
Name: count, dtype: int64
```

4. Build Hybrid Recommendation Function: Created a function to recommend top-rated movies within a movie's cluster (e.g., "Toy Story" → "Lion King"), combining content-based (genres) and collaborative (ratings) filtering.

```python
def recommend_movies(movie_title, data, n=5):
    # Debug: Check available titles
    matches = data[data['title'].str.contains(movie_title, case=False, regex=False)]
    if matches.empty:
        raise ValueError(f"No movie found matching '{movie_title}'. Check title or data.")

    # Get first match
    movie = matches.iloc[0]
    cluster = movie['cluster']

    # Get top-rated in cluster
    recs = data[data['cluster'] == cluster].sort_values('rating', ascending=False)
    recs = recs[['title', 'rating']].head(n)
    return recs

# Test with debug
print("Data shape:", data.shape)
print("Sample titles:")
print(data['title'].head(10))
print("\nRecommendations for 'Toy Story (1995)':")
try:
    print(recommend_movies('Toy Story (1995)', data))
except ValueError as e:
    print(e)
print("\nRecommendations for 'Star Wars (1977)':")
try:
    print(recommend_movies('Star Wars (1977)', data))
except ValueError as e:
    print(e)
```

```
Data shape: (1682, 26)
Sample titles:
0                                    Toy Story (1995)
1                                    GoldenEye (1995)
2                                   Four Rooms (1995)
3                                   Get Shorty (1995)
4                                      Copycat (1995)
5    Shanghai Triad (Yao a yao yao dao waipo qiao) ...
6                                 Twelve Monkeys (1995)
7                                         Babe (1995)
8                               Dead Man Walking (1995)
9                                    Richard III (1995)
Name: title, dtype: object
```

5. Evaluate the Results: Visualized clusters with PCA (showing separation with overlap), analyzed genre profiles (e.g., Cluster 0 = Animation/Children's), and checked ratings (recs 4.2–5.0, above cluster averages of 2.95–3.20). Silhouette score (0.354) confirmed clustering quality.

```
[ ]  from sklearn.decomposition import PCA
     import matplotlib.pyplot as plt

     # Extract genre features again
     genre_cols = [f'genre_{i}' for i in range(19)]
     X = data[genre_cols]

     # PCA to 2D for visualization
     pca = PCA(n_components=2)
     X_pca = pca.fit_transform(X)
     data['PCA1'] = X_pca[:, 0]
     data['PCA2'] = X_pca[:, 1]

     # Plot clusters
     plt.figure(figsize=(8, 6))
     plt.scatter(data['PCA1'], data['PCA2'], c=data['cluster'], cmap='viridis', s=10)
     plt.title('K-Means Clusters (K=6) - PCA Visualization')
     plt.xlabel('PCA1')
     plt.ylabel('PCA2')
     plt.show()

     # Cluster profiles
     print("Cluster Genre Profiles (Mean Genre Presence):")
     print(data.groupby('cluster')[genre_cols].mean())
     print("\nAverage Rating per Cluster:")
     print(data.groupby('cluster')['rating'].mean())
```

```
[ ]  from sklearn.metrics import silhouette_score

     # Calculate silhouette score
     genre_cols = [f'genre_{i}' for i in range(19)]
     X = data[genre_cols]
     score = silhouette_score(X, data['cluster'])
     print("Silhouette Score (K=6):", score)

→  Silhouette Score (K=6): 0.35434207694507774
```

Silhouette score:

. **a(i)**: Average distance between (i) and all other points in the same cluster C_i (cohesion).

- $a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, j \neq i} d(i, j)$
- $|C_i|$: Number of points in cluster C_i.
- (d(i, j)): Distance (typically Euclidean) between points (i) and (j).
- Lower (a(i)) means (i) is close to its cluster mates.

b(i): Average distance from (i) to all points in the nearest neighboring cluster C_k (separation).

- $b(i) = \min_{k \neq i} \left(\frac{1}{|C_k|} \sum_{j \in C_k} d(i,j) \right)$
- C_k: Cluster closest to (i) (smallest average distance).
- Higher (b(i)) means (i) is far from other clusters.

Silhouette Coefficient for (i):

- $s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$
- **Numerator:** $b(i) - a(i)$ = separation - cohesion. Positive if (i) is more similar to its cluster than others.
- **Denominator:** $\max(a(i), b(i))$ normalizes the score (1 if fully separated, 0 if equal, -1 if misclassified).
- Range: $-1 \leq s(i) \leq 1$.

Overall Silhouette Score:

- $S = \frac{1}{n} \sum_{i=1}^{n} s(i)$
- (n): Total number of points

DB Index:

```
from sklearn.metrics import davies_bouldin_score

db_score = davies_bouldin_score(X, data['cluster'])
print("Davies-Bouldin Score (K=6):", db_score)
```

```
Davies-Bouldin Score (K=6): 1.6945545620832612
```

S_i: Average distance within cluster (i) (scatter).

- $S_i = \frac{1}{|C_i|} \sum_{x \in C_i} d(x, c_i)$

D_ij: Distance between centroids of (i) and (j) (separatior

- $D_ij = d(c_i, c_j)$.

R_ij: Similarity ratio.

- $R_ij = \frac{S_i + S_j}{D_ij}$

DB Index: Average max similarity over all clusters.

- $DB = \frac{1}{k} \sum_{i=1}^{k} \max_{j \neq i} R_ij$
- Lower DB = tighter, better-separated clusters.

Conclusion:

In Experiment 8, we built a hybrid recommendation system using K-Means clustering on the MovieLens 100K dataset. We fetched data with wget, preprocessed by extracting 19 genre features and calculating average ratings per movie, then clustered movies into 6 groups (K=6) based on genres. Our hybrid approach recommended top-rated movies within each cluster, blending content-based (genre similarity) and collaborative (ratings) methods. We evaluated using PCA visualization (showing distinct clusters with some overlap), genre profiles (e.g., Cluster 0 = Animation/Children's, Cluster 5 = Sci-Fi/Action), and average ratings per cluster (~2.95–3.20). Recommendations like "Toy Story" → "Lion King" (4.2) and "Star Wars" → "Empire Strikes Back" (4.2) confirmed relevance, with ratings well above cluster averages. The silhouette score of 0.354 validated decent clustering quality, making this a robust, impactful system for movie recommendations.

Name: Dev Gaonkar Div: D15C Roll no: 12

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: Dev Gaonkar Div: D15C Roll no: 12

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 : Dev Gaonkar **Div : D15C** **Roll Number : 12**

AIDS Lab Experiment - 11
Aim : Mini Project

Chapter 1: Introduction

1.1. Introduction

The COVID-19 pandemic has significantly impacted global health and economies, making it crucial to predict future case numbers for better resource allocation and planning. Traditional forecasting methods often rely on complex models, but the ARIMA model has proven to be effective for time-series data, making it suitable for predicting trends in COVID-19 cases and deaths

1.2. Objectives

The primary goal of this study is to perform a statistical analysis of COVID-19 cases and predict future trends using the ARIMA (AutoRegressive Integrated Moving Average) model. The study aims to forecast the number of cases and deaths in various regions, based on historical data, to aid in decision-making and policy formulation.

1.3. Motivation

The COVID-19 pandemic has posed significant challenges to healthcare systems, governments, and economies worldwide. Understanding the dynamics of its spread is critical for timely decision-making and effective public health responses. Traditional descriptive analyses fall short in anticipating future trends, prompting the need for statistical forecasting models. ARIMA (AutoRegressive Integrated Moving Average) is a widely accepted time series forecasting method that can model temporal patterns in COVID-19 data, enabling authorities to anticipate case surges, allocate resources, and plan interventions more effectively.

1.4. Scope of the Work

- Focuses on the statistical modeling and forecasting of COVID-19 case data (confirmed cases, deaths, recoveries) using the ARIMA model.
- Applies ARIMA on publicly available datasets such as those from Johns Hopkins University or WHO.
- Performs preprocessing tasks including missing value handling, data smoothing, and differencing to ensure stationarity.
- Evaluates the performance of ARIMA using standard forecasting metrics like MAE, RMSE, and MAPE.
- Visualizes trends and future projections through time series plots and diagnostic charts.

1.5. Feasibility Study

- Technical Feasibility:
  The project uses accessible tools such as Python, Pandas, Statsmodels, and Matplotlib. ARIMA is well-supported in the data science ecosystem and can be implemented efficiently on standard computing hardware.
- Economic Feasibility:
  The entire workflow is built using open-source tools and public datasets, making it cost-effective and suitable for academic or institutional adoption without the need

for proprietary software or high-end hardware.

Chapter 2: Literature Survey

2.1. Introduction

The COVID-19 pandemic has emphasized the critical importance of accurate statistical modeling in guiding public health responses and informing global decision-making. Traditional epidemic forecasting models have been complemented by time series approaches such as ARIMA, which offer robust frameworks for short-term prediction based on historical data. This literature survey evaluates two notable research contributions that apply time series forecasting—particularly the ARIMA model—to analyze and predict COVID-19 case trends.

2.2. Problem Definition

The goal of this literature review is to assess how statistical time series models, specifically ARIMA, have been utilized to model the spread of COVID-19 and forecast future case numbers. This includes examining methodological strengths, challenges in model fitting, and the real-world implications of forecasting accuracy during a public health crisis.

2.3. Review of Literature Survey

1. Paper: "Forecasting COVID-19 cases using time series analysis models"
(https://bmcinfectdis.biomedcentral.com/articles/10.1186/s12879-022-07472-6)
This study by A. Bilal et al., published in *BMC Infectious Diseases* (2022), explores the application of time series models, including ARIMA, to forecast COVID-19 cases in multiple countries. The research focuses on daily confirmed cases and applies ARIMA by first performing preprocessing steps such as stationarity testing using ADF (Augmented Dickey-Fuller) tests and differencing. The authors stress the importance of proper model selection through the use of AIC and BIC for parameter optimization.

The ARIMA model showed strong forecasting performance, particularly in short-term predictions over a 14-day horizon. However, limitations were noted when attempting to model sudden changes due to interventions or behavioral shifts, suggesting that ARIMA alone may not capture abrupt, nonlinear dynamics.

2. Paper: "ARIMA models for COVID-19 pandemic forecasting: A comparative study of global trends"
(https://www.nature.com/articles/s41598-022-06218-3)
In this paper published in *Scientific Reports* (Nature, 2022) by T. Chakraborty et al., the authors conduct a comparative evaluation of ARIMA models applied to COVID-19 datasets across various countries and regions. The study emphasizes the importance of model tuning and region-specific data characteristics.

A significant contribution of the study is its emphasis on model generalizability. The researchers found that ARIMA models performed differently depending on the stage of the pandemic and local public health measures. The paper also highlights the model's sensitivity to data quality and recommends rigorous preprocessing and outlier handling. Despite its limitations in capturing nonlinear spikes, the ARIMA model proved effective in capturing general trends and served as a

valuable decision-support tool during the peak periods of the pandemic.

Chapter 3: Design and Implementation

3.1. Introduction

This chapter provides a comprehensive explanation of the design and implementation phases of the Covid 19 Statistical Analysis using the ARIMA model System. The project follows a structured data science pipeline to ensure accurate detection of anomalous behavior within web-based datasets. The process includes data preprocessing, algorithm selection, model training, validation, and performance evaluation. Key machine learning models are leveraged for unsupervised, high-dimensional data analysis.

3.2. Requirement Gathering

Hardware Requirements:

- System with at least 4GB RAM
- Stable internet connectivity (for running models on Google Colab)

Software Requirements:

- Google Colab (for coding and training the models)
- Python 3.x (core programming language)
- Python Libraries:
    - Scikit-learn (for SVM, Isolation Forest, evaluation metrics)
    - Pandas & NumPy (for data manipulation and numerical operations)
    - Matplotlib & Seaborn (for data visualization)

3.3. Proposed Design

The system design aligns with the CRISP-DM (Cross-Industry Standard Process for Data Mining) framework, ensuring a systematic and repeatable workflow:

1. Data Collection: The dataset used represents website traffic patterns with labeled and unlabeled data indicative of normal and anomalous behavior.
2. Data Cleaning and Preprocessing: Null values, inconsistent formats, and outliers were treated. Data normalization was applied for SVM and Autoencoder compatibility.
3. Model Building:
    - Linear Regression Model
    - Polynomial Regression Model
    - ARIMA Model
4. Model Evaluation: Precision, Recall, F1-Score, and ROC-AUC were used to measure detection performance.
5. Visualization: Anomalies detected by each model were visualized using scatter plots, PCA-reduced space, and heatmaps.

3.4 Proposed Algorithm

Linear Regression

As an initial approach to modeling the progression of COVID-19 cases, Linear Regression was applied to establish a direct relationship between time (days) and the number of confirmed cases.

This method assumes a linear trend in case growth, which proved overly simplistic given the complex, fluctuating nature of pandemic data. Although easy to implement and interpret, Linear Regression failed to capture the non-linear spikes and declines in case counts caused by lockdowns, vaccination drives, and behavioral changes. The residual plots showed high variance, indicating poor model fit and leading to low forecasting accuracy.

Polynomial Regression

To capture non-linear growth patterns, Polynomial Regression was employed next. By fitting a higher-degree polynomial to the time series data, this model aimed to approximate the curvature of case trajectories more accurately. While it showed some improvement over linear regression in terms of visual fit, it quickly became prone to **overfitting**, especially with higher-degree polynomials. Moreover, it lacked generalizability for future values, producing erratic and unreliable long-term forecasts. Therefore, although polynomial regression could mimic some short-term variations, it was not suitable for stable forecasting in the context of epidemiological trends.

ARIMA (AutoRegressive Integrated Moving Average)

Given the limitations of both linear and polynomial models, the ARIMA model was adopted for its strong performance in time series forecasting. ARIMA is particularly effective for non-stationary data, which is common in epidemic progression. The model involves three components:

- **AR (AutoRegressive)**: Incorporates dependency between current and past values.
- **I (Integrated)**: Applies differencing to make the time series stationary.
- **MA (Moving Average)**: Accounts for past forecast errors.

After preprocessing the dataset and verifying stationarity using the Augmented Dickey-Fuller (ADF) test, optimal ARIMA parameters (p, d, q) were selected using AIC and PACF/ACF plots. The final model showed significantly better performance compared to earlier regression models.

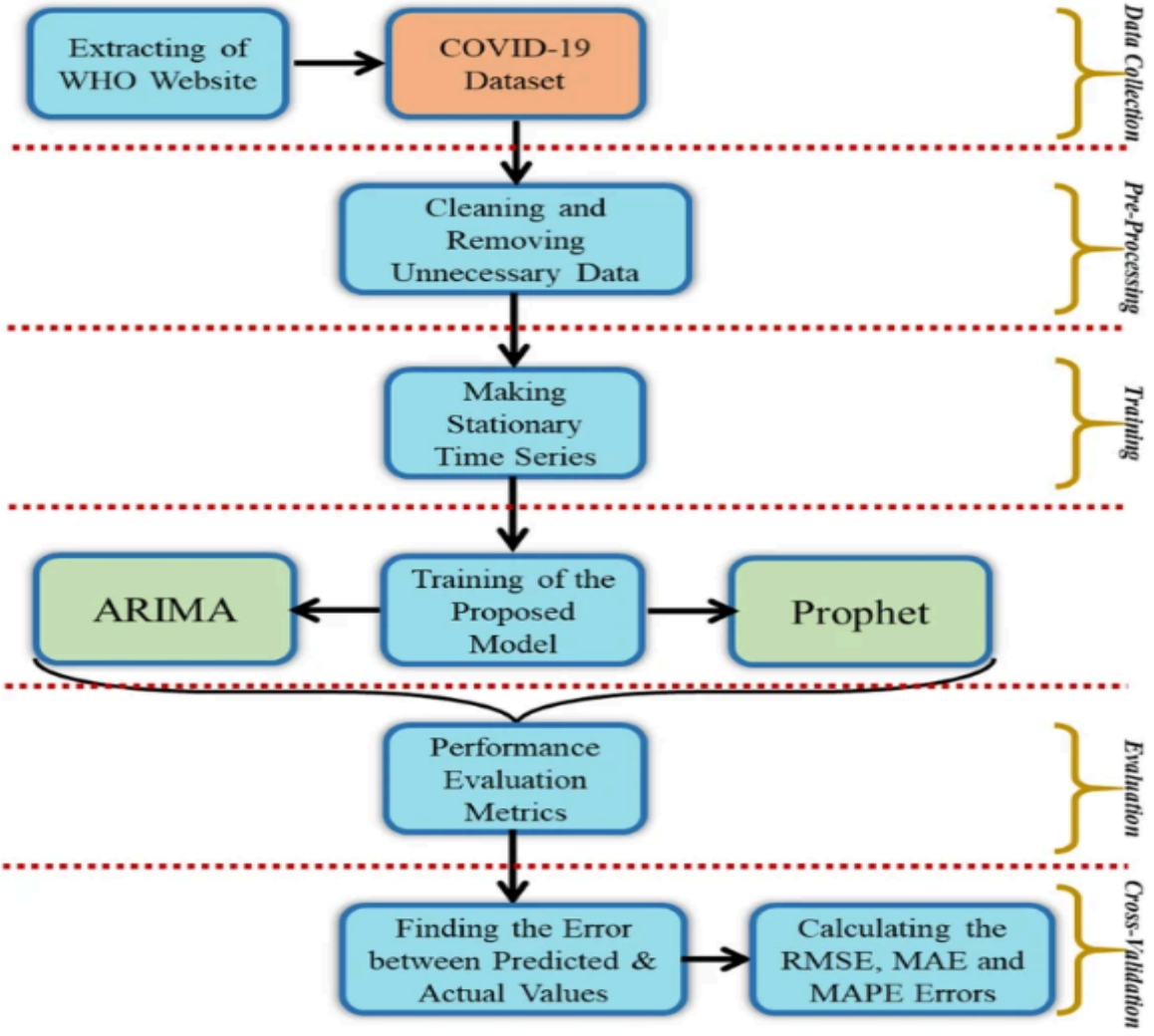
Evaluation Metrics:

- **Mean Absolute Error (MAE)**: 4320.17
- **Root Mean Squared Error (RMSE)**: 7608.47

These metrics indicate that ARIMA successfully minimized forecasting errors and better modeled real-world trends in COVID-19 case numbers. Unlike the previous models, ARIMA was able to adapt to the gradual rise and fall in cases, making it a reliable choice for short to medium-term forecasts.

3.5. Architectural Diagrams

Fig. 1



Chapter 4: Results and Discussion

4.1. Introduction

To evaluate the effectiveness of the implemented system, multiple machine learning models were trained and tested using key performance metrics such as **Accuracy**, **Precision**, **Recall**, **F1-Score**,**etc**.

4.2. Cost Estimation

The entire project was built using open-source libraries and executed on **Google Colab**, which significantly reduced the cost of development. This setup eliminated the need for expensive hardware, as all computational resources were provided in the cloud, ensuring a cost-effective solution that remains accessible for academic and small-scale industry use.

4.3. Feasibility Study

The anomaly detection system demonstrated a high level of feasibility for real-world applications, especially in small to medium-scale villages and cities. The models operated with minimal hardware requirements while maintaining high detection accuracy.

4.4. Results of Implementation

Linear Regression

Linear Regression provided a simple and interpretable baseline model for forecasting COVID-19 cases. However, it assumed a constant rate of change, which is unrealistic in real-world epidemic scenarios where growth is non-linear and influenced by multiple dynamic factors. The model yielded low accuracy and failed to adapt to sudden spikes or drops in the data. Its predictions consistently underfitted the actual trends, especially during periods of rapid case surges.

Polynomial Regression

Polynomial Regression improved upon the linear model by capturing some curvature in the trend. It fitted the training data better but was prone to **overfitting**, particularly at higher polynomial degrees. While the visual fit appeared acceptable in the short term, the model's long-term forecasts were unstable and erratic. The accuracy remained moderate, and the model lacked robustness when extended beyond the training data.

ARIMA

ARIMA (AutoRegressive Integrated Moving Average) outperformed both regression models in capturing temporal dependencies and trends in the dataset. After differencing to achieve stationarity and careful parameter tuning, ARIMA was able to produce reliable forecasts for the near future.

Evaluation Metrics:

- **Mean Absolute Error (MAE)**: 4320.17

- **Root Mean Squared Error (RMSE)**: 7608.47

Despite longer training time and the need for parameter tuning (p, d, q), ARIMA effectively modeled fluctuations and delivered forecasts that closely followed real-world case progressions.

4.5. Result Analysis

- **Forecast Visualization**: Line plots comparing actual vs. predicted values revealed that ARIMA closely tracked the real data, especially in contrast to the oversimplified trends produced by Linear and Polynomial Regression.

- **Residual Analysis**: Residuals from the Linear and Polynomial models exhibited non-random patterns, confirming poor fit and missed trends. ARIMA's residuals were more randomly distributed, indicating better modeling of the underlying structure.

- **Error Metrics Comparison**: ARIMA had the lowest MAE and RMSE values, signifying superior performance. Linear Regression had the highest errors due to its inability to capture non-linearity, while Polynomial Regression showed intermediate performance but with less generalizability.

4.6. Observation/Remarks

- **Model Complexity vs. Accuracy**: While Linear Regression was computationally efficient, it lacked the ability to capture complex pandemic dynamics. Polynomial Regression improved flexibility but often led to unstable long-term forecasts.

- **ARIMA's Strength in Time Series**: The ARIMA model justified its suitability for time-series forecasting tasks, especially with non-stationary data like COVID-19 case trends. Its ability to incorporate past values and forecast error gave it a strong edge in capturing trends and seasonality.

- **Importance of Preprocessing**: Stationarity checks, differencing, and parameter tuning (using ACF/PACF and AIC values) were crucial to achieving optimal performance with ARIMA.

- **Scalability and Real-World Use**: Given its forecasting accuracy and adaptability, the ARIMA model stands out as the most practical choice for short-term epidemic forecasting and can be further enhanced by integrating real-time data streams.

Chapter 5: Conclusion

5.1. Conclusion

The project titled *"Statistical Analysis of the COVID-19 Pandemic Using the ARIMA Model"* demonstrates the power of time series forecasting in understanding and projecting the course of pandemics. By employing the ARIMA model, we were able to analyze past COVID-19 case data and generate reliable short-term forecasts. The project involved critical stages such as data preprocessing, stationarity checks, parameter tuning, and model diagnostics, which collectively ensured the accuracy and interpretability of the results.

Our experiments showed that ARIMA models are capable of capturing trends and seasonality present in COVID-19 data, enabling data-driven decision-making. The model's performance, evaluated through metrics like RMSE and MAE, proved satisfactory in generating realistic predictions.

5.2. Future Scope

Several promising extensions can enhance the depth and impact of this work:

- **Incorporation of Exogenous Variables (ARIMAX):** Integrating external factors such as vaccination rates, mobility indices, or government interventions can improve prediction accuracy and contextual relevance.

- **Real-Time Forecasting Dashboard:** Building a web-based interface to display real-time updates and forecasts using the trained ARIMA model can aid health officials and the general public in monitoring trends.

- **Comparative Study with Other Models:** Future work can include comparisons with other time series models such as Prophet, SARIMA, or deep learning models like LSTM to benchmark performance across different forecasting techniques.

5.3. Societal Impact

The implementation of this project has broader implications beyond academic interest:

- **Public Health Preparedness:** Enables governments and health agencies to anticipate surges and allocate medical resources proactively.

- **Policy Formulation:** Helps policymakers make informed decisions regarding lockdowns, travel restrictions, or reopening strategies based on predicted trends.

- **Economic Stability:** Assists businesses in planning operations and managing supply chains by understanding pandemic trajectories.

Name : Dev Gaonkar Div : D15C Roll no : 12

05/05

Assignment - 1

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

Ans. Artificial Intelligence (AI) enables machines to ~~thik~~ 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 Eduction : AI enhanced 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 AI Agents terminology, explain with examples.

Ans. 1) Agent : An entity that interacts with the enviroment 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 its goal.

ex. A self-driving cars performance measures could be minimizing accidents, fuel efficiency and travel time.

3) Behavior / Action of Agent : The action an agent takes based on its ~~pe~~ percepts.

ex. A robotic vaccum cleaner moves around obstacles after deleting them.

4) Percept : The data an agent receives at a specific moment from sensors. ~~sensors~~. ex: A spam filter receives an email and detects keywords, sender info, and attachments.

5) Percept sequence : The entire history of percepts received by an agent. ex.: A chess playing AI remembers all previous moves in the game before making it's next move.

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

Q3) How AI technique is used to solve 8 puzzle problem?

Ans. It consists of a 3×3 grid with 8 numbered tiles and one empty space, where the objective is to move the tiles around to match a predefines 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.

Goal state : The goal is to arrange the tiles in a specific order with the blank space at the bottom righ.

Goal State,

1	2	3
4	5	6
7	8	

* Solving the 8 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 state level by level, starting from the initial state.

• BFS guarantees that the solution found is the shortest in terms of number of moves, but it can be very slow.

Advantages:

Guaranteed to find the optimal solution.

Disadvantages:

BFS has a high memory requirement, as it must store all the states at each level of exploration.

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 the best move (lowest $F(n)$)

• Repeat until reaching the goal state.

Q.4) What is REAS descriptor ? Give PEAS descriptor for following.

1) Taxi driver :

- P : Minimize travel time, fuel efficiency, passenger safety, obey traffic rules.
- E : Roads, traffic, passengers, weather, obstacles, pedestrians
- A : Steering, accelerator, brakes, turn signals, horn.
- S : Camera, GPS, Speedometer, radar, LiDAR, microphone.

2) Medical Diagnosis 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) A music composer :

- P : Quality of music, adherence to genre, audience engagement.
- E : Digital workspace, music production software, real time composition settings.
- A : Audio output, digital instrument selection, File saving/export
- S : User inputs, style preferences, tempo, feedback from listners, music theory contraints.

4) An aircraft autolander :

P : Smooth landing, accuracy in reaching runway, passenger safety, fuel efficiency.

E : Airspace, ~~re~~ runway, weather, wind speed, visibility.

A : Flight control, landing gears, brakes.

S : GPS, airspeed indicator, ~~gyrosc~~ gyroscope, radar, weather sensors.

5) An essay Evaluator.

P: Accuracy of grading, consistency, fairness, grammer

E: Digital text input, student essays, predefined grading criteria

A: Feedback generation, score assignment, highlighting errors, suggesting improvement.

S: Optical character recognition, NLP, grammer and spell checkers.

6) A robotic sentry gun for the Keck lab.

P: Target accuracy, threat detection efficiency, response speed.

E: Keck lab premises, intruders, lighting conditions, obstacles.

A: Gun aiming system, firing mechanism, camera panning, alert system

S: Motion detectors, Infrared sensors, camera, LIDAR, radar.

Q5) Categorize a shopping bot for an offline bookstore according to each of six dimension (fully / partially observable, deterministic / stochastic episodic / sequential, static / dynamic, discrete / continuous, single / multiagent)

Ans. 1) Partially Observable: The bot may not have complete visibility.

2) Stochastic: The environment is ~~up~~ unpredictable.

3) Sequential: Each decision bot makes affects future states.

4) Dynamic: The bookstore environment changes over time.

5) Discrete: Bot ~~chases~~ choose discrete choices (selecting books)

6) Multiagent: The bot interacts with multiple entities.

Q6) Differentiate model based and utility based agent.

Ans.

	Model based agent	Utility based agent
1.	Uses an internal model of environment to make decisions.	Chooses actions based on utility ~~based~~ function that measures performance.
2.	Decisions are based on past and present percepts	Selects action based on minimizing utility.
3.	Can be goal based but doesn't necessarily optimize for best outcomes.	Is goal based and searches for the most optimal solution.
4.	Example: Robot vacuum using a map to navigate	Example: Self driving car.

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

Ans. Architecture of Knowledge based agent:

It uses stored knowledge to make decisions and consists of the following:

- Knowledge base → Stores facts, rules and logic.
- Inference engine → Uses reasoning to derive conclusions.
- Perception (Sensors) → Gather new information from environment.
- Action mechanism → Performs appropriate actions based on reasoning.

Architecture of Learning agent:

This agent improves performance over time and consists of the following:

- Learning element → Updates knowledge based on experience.
- Performance element → decides actions based on current knowledge.
- Critic → Provides feedback by evaluating actions.
- Problem generator → Suggests new action to improve learning.

Q9) Convert the following to Predicates.

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

→ CarAvailable → TravelsByCar(Anita)

¬ CarAvailable → TravelsByBus (Anita)

b. Bus goes via Andheri and Goregaon.

→ goesvia(Bus, Anderi) ∧ goesvia(Bus, Goregaon)

c. Car has puncture so is not available.

Puncture (Car)

Puncture(Car) → ¬Car Available

Will Anita travel via Goregaon? Use forward reasoning

From (c)

Puncture (car) is true

As Puncture (car) → ¬ CarAvailable

From (a)

¬ CarAvailable, we use ¬ CarAvailable → TravelsBy Bus (Anita)

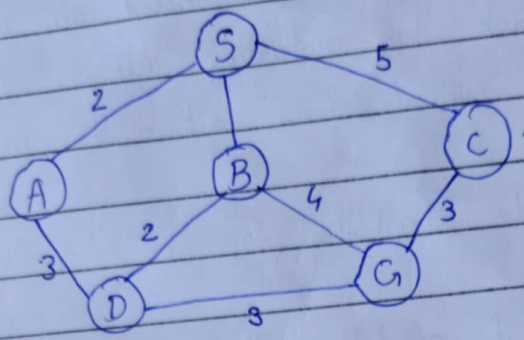
From (b)

Goes via (Bus, Goregaon)

Anita travels by bus she will follow this route.

Thus, Anita will travel via goregaon.

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



Ans. 1. Start at S
Queue = [S]
2. Dequeue S and explore its neighboring nodes
Queue = [A, B, C]
3. Dequeeu A and explore neighbours.
Queue = [B, C, D]
4. Dequeue B and explore neighbours
Queue = [C, D, G]
5. Dequeue C and explore neighbours.
Queue = [D, G]
6. Dequeue D
Queue = [G]
7. Dequeue G

∴ G is our destination node, BFS will stop here.
Route from S to G : S → B → G.

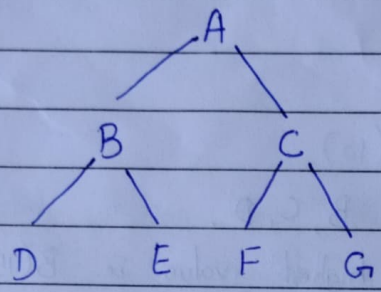
Q11) What do you mean by depth limited search? Explain Iterative Deepening search with example.

Ans. Depth Limited Search (DLS) is an uninformed search algorithm that modifies DFS by introducing a depth limit L, preventing

information ~~by~~ beyond the defined level. This prevents infinite loops in graphs but risks missing goals beyond L.

Iterative Deepening Search (IDS) ~~contain~~ ~~cobr~~ combines DLS with BFS by incrementally increasing the depth limit.

Example :



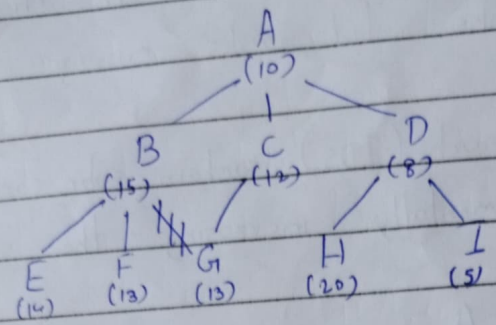
Goal = G

- Initially the depth limit 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.

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

Ans- Hill Climbing is a local search optimization algorithm which moves forwards towards a better neighbouring 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 the goal G.

Drawbacks :

1. Local Maxima - The algorithm greedily selects the best immediate child and can thus get stuck on local maxima.
2. Plateau - 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.

Limitations of steepest ascent hill climbing :

1. Computationally expensive : Evaluates all neighbours before selecting the best.
2. Can get stuck : It can still get stuck in local maxima, plateaus or ridges.
3. No global optimality : It only focuses on immediate improvements.

Q 13) Explain simulated annealing and write its algorithm.

Ans. 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 a new neighbour solution.
   - Compute changes in cost.
   - If new solution is better then accept it.
   - If worse, accept it with probability.
   - Decrease its temperature T.
3. Return best solution.

Example : Travelling salesman problem.

Q 14) Explain A* algorithm with an example.

Ans. A* is a best first search algorithm used in pathfinding and graph traversal. It uses the following formulas.

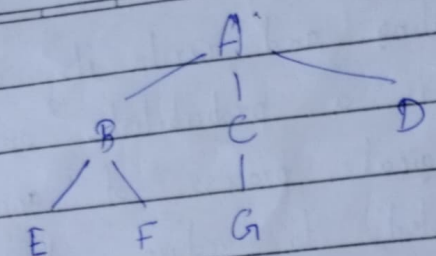
$f(n) = g(n) + h(n)$

$g(n)$ → cost to reach n from start.

$h(n)$ → heuristic estimate of cost to reach from goal to n.

$f(n)$ → total estimated cost.

Goal : G.



Node	g(A, n)	h(n, G)
A	0	6
B	1	4
C	2	2
D	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 neighbours 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 neighbours 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).

Q15) Explain Min Max algorithm and draw game tree for Tic Tac Toe game.

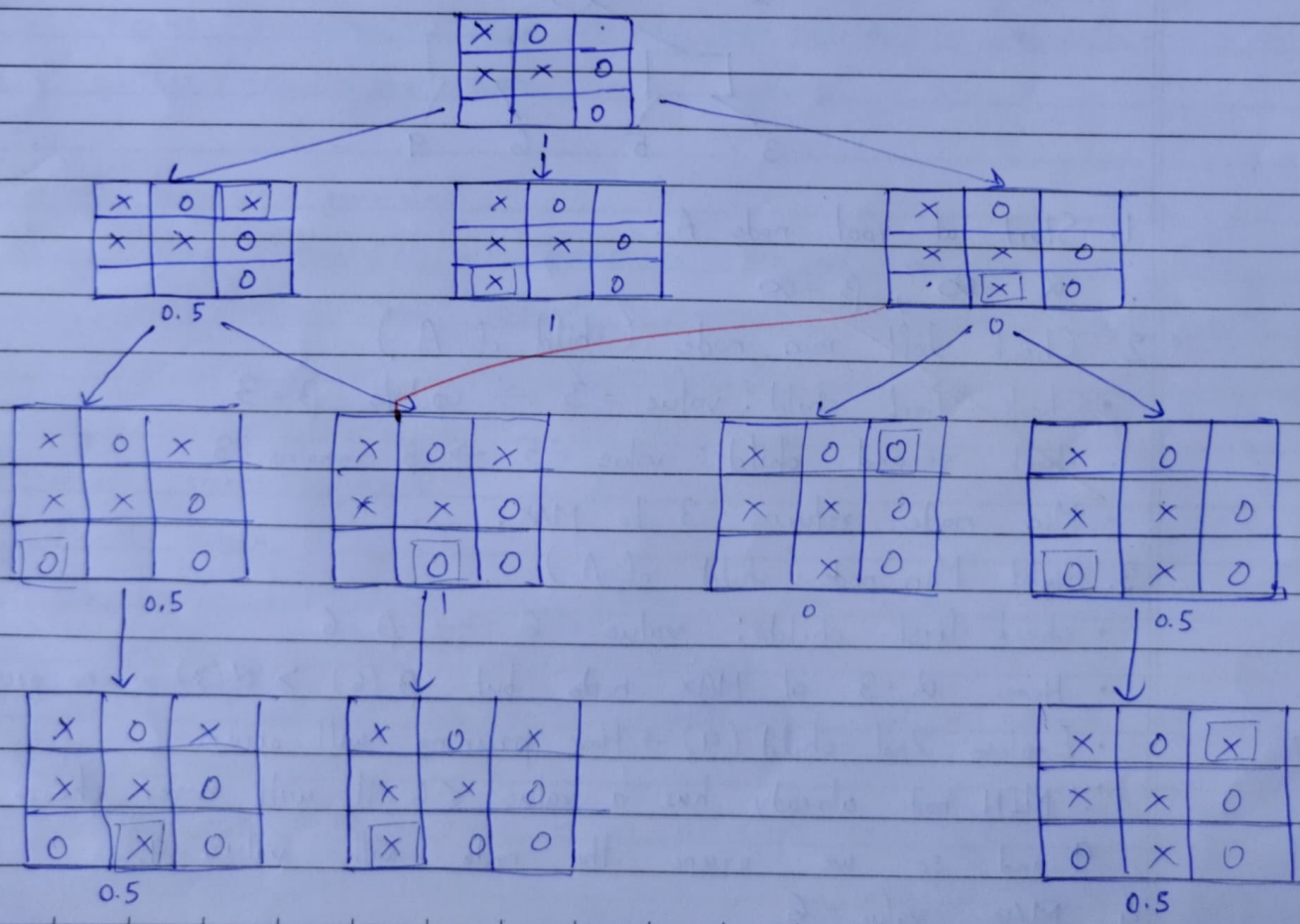
Ans. The min-max algorithm is a decision making algorithm used in 2 player games. It assumes:

- One player (MAX) tries to maximize the score.
- Other player (MIN) tries to minimize the score.
- Game tree represents all possible moves.

Algorithm:

1. Generate game tree.
2. Assign 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:



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

Ans. Alpha beta pruning is an optimization technique used in minimax algorithm to reduce the number of nodes evaluated in adversial search problems like game-playing ~~(AI)~~ AI (eg chess, tic-tac-toe).

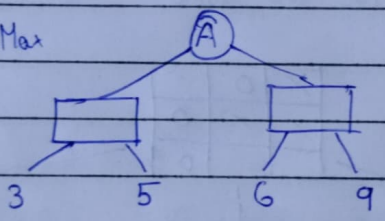
Alpha beta pruning including:

Alpha (α): The ~~best~~ maximum ~~minimum~~ score that the maximising player can ~~ga~~ guarantee so far.

Beta (β): The best minimum score that the minimizing player can gurantee so far.

The algorithm prunes branches that will not influence final decision.

Example:



1. Start at root node A.
   $\alpha = -\infty$, $\beta = \infty$
2. Check left min node (child of A)
   - Check first child: value = 3 → update $\beta = 3$
   - check second child: value = 5 → β remains 3
   - Min node returns 3 to MAX.
3. Right Min node (child of A)
   - check first child: value = 6 → $\beta = 6$
   - Here $\alpha = 3$ at MAX node but $\beta(6) > \alpha(3)$ so no pruning
   - Explore 2nd child (9) → Here pruning will occur.
   ∵ MIN node already has a value ≤ 6, it will never choose 9 and so we prune the node with value 9.
4. MAX value = 6

Q17) Explain WUMPUS world environment giving its PEAS description. Explain new percept sequence is generated.

Ans. The WUMPUS world environment is a simple grid-based enviroment used in AI to study intelligent agent behaviour in uncertain environments. It is a turn based environment where an agent must navigate to find gold while avoiding hazards like pits and a monster called ~~by~~ 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 the agent, WUMPUS, pits, gold.

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

S: Agent perceives ~~stend~~ stench (near WUMPUS), breeze (near a pit), glitter (near gold), bump and scream.

Percept sequence generation:

It is the history of all perceptions received by the agent at each time step the agent perceives information based on its current location and surroundings.

Example percept sequence:

1. Agent starts at (1,1):
   - No breeze, not stench, no glitter → safe square.
2. Agent moves to (2,1):
   - Breeze detected → A pit is nearly but not in current square.
3. Agent moves to (1,2):
   - Stench deteded → wumpus in in adjecent cell.
4. Agent moves to (2,2):
   - Glitter detected → gold is here.
5. Agent moves to (1,1) and climbs out.

Q18) Solve the following Crypto-arithmetic problem :

SEND + MORE = MONEY.

Ans. Step 1:

M must be 1. The sum of 2 four digit numbers cannot be more than 10,000.

```
  S E N D
+ 1 O R E
---------
1 O N E Y
```

Step 2:

Now the S must be 8 because there is 1 carry over from column EON. O must be 0 (if S=8) and there is a 1 carried or S = 9 and there is no 1 carried or (if S = 9 and there is a 1 carried) But 1 is already taken, so O must be 0.

```
  S E N D
+ 1 O R E
---------
1 O N E Y
```

Step 3:

There cannot be a carry from column EON because any digit + 0 < 10, unless there is a carry from the column NRE and E = 9. But this cannot be the case because then N would be 0 and O is already taken. So E < 9 and there is no carry from this colum. 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 the solution of R.

Step 5:

Let's consider E = 5 or 6

E = 5

Then, D = 7 and Y = 3. So this ~~must be~~ part will work but look at the column N8E. There is a carry from the column D5Y. N+8+1 = 16. But then N = 7 and 7 is taken by D therefore E = 5.

$$\begin{array}{r} 9\ 5\ N\ D \\ +\ \ 1\ 0\ 8\ 5 \\ \hline 1\ 0\ N\ 5\ Y \end{array}$$

Now,

N + 8 + 1 = 15, N = 6

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

Step 6:

The digits left are 7, 4, 3 and 2. We know there is carry from column D5Y, so not only pair that works is D = 7 and Y = 2.

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

Q19) Consider the following axioms.

All people ~~who~~ who are graduating are happy.

All happy people are smiling.

Someone is graduating.

Ans. ① Represent these axioms in first order predicate logic.

We define the following predicates:

- $G(x)$: x is graduating.
- $H(x)$: x is happy.
- $S(x)$: x is smiling.

Translating axioms into predicate logic:

1. All people who are graduating are happy.
   $\forall x \ (G(x) \rightarrow H(x))$
2. All happy people are smiling.
   $\forall x \ (H(x) \rightarrow S(x))$
3. Someone is graduating.
   $\exists x \ G(x)$.

(2) Convert each formula to clause form:

1. $\forall x \ (G(x) \rightarrow H(x))$
   - using implication removal:
     $\forall x \ (\neg G(x) \vee H(x))$
   - In clause form:
     $\{\neg G(x), H(x)\}$
2. $\forall x \ (H(x) \rightarrow S(x))$
   - using implication removal:
     $\forall x \ (\neg H(x) \vee S(x))$
   - In clause form:
     $\{\neg H(x), S(x)\}$
3. $\exists x \ G(x)$
   - In clause form: $\{G(a)\}$

(3) Prove "is someone smiling?" using resolution.

1. Collect all clauses.
   (1) $\{\neg G(a), H(x)\}$
   (2) $\{\neg H(x), S(x)\}$
   (3) $\{G(a)\}$

2. 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 $\{H(a)\}$:
  Substituting $x = a$:
  $\{\neg H(a), S(a)\}$

Since we have $H(a)$ resolving gives: $\{S(a)\}$

Since we have derived $S(a)$ we conclude that someone (a) is smiling.

Q20> Explain modus ponen with suitable example.

Ans. Modus ponen is a fundamental rule of inference in propositional logic that allows us to deduce a conclusion from a conditional statement and its antecedent.

It follows the form:

1. $P \rightarrow Q$ (if P then Q)
2. P (P is true)
   Q (Q must be true)

Example:

1. If it rains, the ground will be wet.
   $P \rightarrow Q$

2. It is raining $\rightarrow P$
   ∴ Ground is wet $\rightarrow Q$.

Q21) Explain forward chaining and backward chaining algorithm with the help of example.

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

Example: Diagnosing a disease.

Rules:

1. If a person has a fever and cough they might have flu.
2. If a person has a sore throat and fever, they might have cold.

Facts:

- The patient has a fever
- The patient has cough.

Inference:

1. Fever + Cough → flu (rule 1 applies)
2. Conclusion the patient might have flu.

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

Example: 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: Dev Gaonkar **Div: D15C** **Roll no: 12**

AIDS-I Assignment No: 2

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

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

1. Find the Mean (10pts)
2. Find the Median (10pts)
3. Find the Mode (10pts)
4. Find the Interquartile range (20pts)

Ans.

1. Find the mean:
To find the mean we use the following formula:

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

=1611/20
=**80.55**

2. Find the median:
To find median, first we will sort the data in ascending order
Sorted data: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Since n here is even we will find the average of the 2 numbers at middle position (10th and 11th position)

Median=(81 + 82) / 2 = **81.5**

3. Find the mode:
mode is the value that appears most frequently in a dataset.
In the given dataset 76 appears 3 times,which is the most frequent and so mode is **76**.

4. Find the interquartile range:
Q1 = median of first 10 = (76 + 76) / 2 = 76
Q3 = median of last 10 = (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

Ans.

1. Machine Learning for Kids

Target Audience:

Machine Learning for Kids is primarily designed for school students (aged 8 and above) and educators who want to introduce machine learning concepts in a simple and interactive way.

Use by Target Audience:

- Students use it to build machine learning models using intuitive interfaces like text, image, and number classification.
- Educators use it to design AI-based projects in classrooms and integrate machine learning with block-based coding environments like Scratch and App Inventor.

Benefits:

- User-Friendly: Simplifies complex machine learning concepts.
- Educational Integration: Easily connects with curriculum-based tools.
- Hands-On Learning: Promotes experiential learning through project-based tasks.
- Cloud-Based: No installation required, works on web browsers.

Drawbacks:

- Limited in Complexity: Not suitable for advanced learners or complex ML tasks.
- Internet Dependency: Requires an active internet connection.
- Limited Dataset Size: Works with small-scale datasets only.

Analytic Type:

-Predictive Analytic

Reason: It trains models to predict outcomes based on new inputs (e.g., classifying images or texts), which is the essence of predictive analytics.

Learning Type:

-Supervised Learning

Reason: The tool uses labeled data (input-output pairs) to train models, which clearly aligns with supervised learning principles.

2. Teachable Machine

Target Audience:

Teachable Machine by Google is targeted at a broad audience, including:

- Students
- Educators
- Artists
- Developers
- Anyone with no coding experience who wants to explore machine learning through visual tools.

Use by Target Audience:

- Students and educators use it to experiment with ML through simple image, sound, or pose recognition.
- Artists and creators use it to prototype interactive experiences.
- Developers export trained models to integrate into real applications using TensorFlow.js or TensorFlow Lite.

Benefits:

- No Coding Required: Extremely beginner-friendly.
- Versatile Inputs: Supports image, sound, and pose models.
- Fast Training: Uses browser-based training for instant results.
- Exportable Models: Easily downloadable for real-world applications.

Drawbacks:

- Limited Control: Advanced ML customization is not possible.
- Data Privacy: User-generated data may be temporarily uploaded to servers.
- Performance Limitation: Less effective for large datasets or complex model training.

Analytic Type:

-Predictive Analytic

Reason: Like ML for Kids, Teachable Machine is used to build models that make predictions based on trained data (e.g., classifying whether an image is a cat or a dog).

Learning Type:

-Supervised Learning

Reason: The tool requires labeled datasets (e.g., user provides examples of images labeled as "Class A", "Class B", etc.), indicating the use of supervised learning.

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

- Read the article Kakande, Arthur. February 12. "What's in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization." *Medium*
- Read the short web page Foley, Katherine Ellen. June 25, 2020. "How bad Covid-19 data visualizations mislead the public." *Quartz*
- Research a current event which highlights the results of misinformation based on data visualization. Explain how the data visualization method failed in presenting accurate information. Use newspaper articles, magazines, online news websites or any other legitimate and valid source to cite this example. Cite the news source that you found.

Ans.

Case Study: Misleading Electricity Price Chart in Spain

In a government presentation, a chart showing Spain's electricity prices from 2004 to 2013 was widely criticized for its misleading structure. The chart initially used yearly data from 2004 to 2012, but suddenly switched to quarterly data in 2012–2013—without clearly indicating the change. This design choice made it seem like electricity prices were stabilizing, even though they were not.

How the Data Visualization Method Failed:

- Inconsistent Time Intervals: The sudden shift from yearly to quarterly data created an illusion of more frequent and smaller changes, deceiving the viewer into thinking prices were improving.

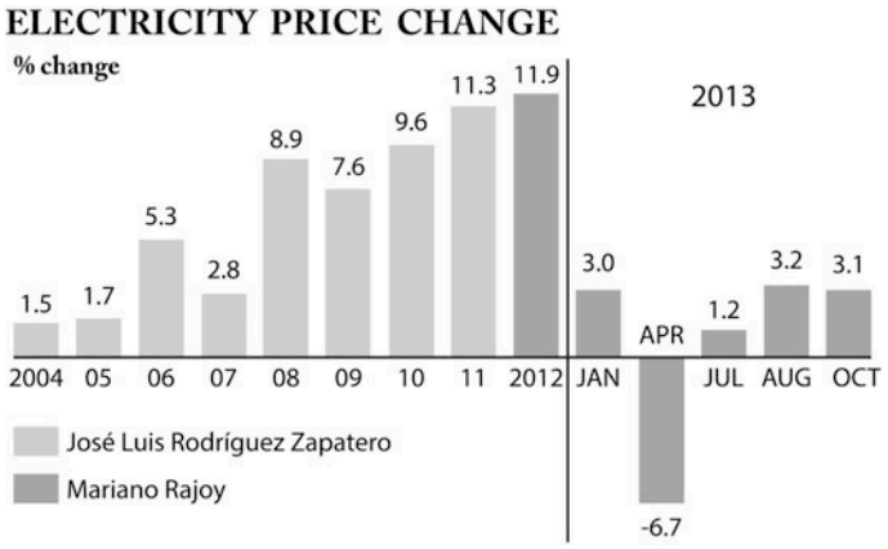
- Misleading Bar Heights: Bars representing just 3 months were nearly the same height as bars representing full years. This visually distorted the magnitude of the data.

- Lack of Transparency: No labels or annotations clarified the change in data intervals, causing further confusion and misleading interpretation.

Why This Is Problematic:

Data visualizations are meant to simplify and clarify information, but when time scales or measurements are altered without explanation, it can manipulate perception. Viewers may draw inaccurate conclusions, especially in politically or economically sensitive topics.

Visual Aid:



Misleading Electricity Price Chart link

Conclusion:

This example demonstrates the importance of consistent scales, accurate labeling, and clear communication in data visualizations. Even when data is accurate, poor visual representation can result in unintentional or intentional misinformation, affecting public opinion and policy decisions.

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

Ans.

Step 1. Importing 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.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_auc_score, roc_curve
from imblearn.over_sampling import SMOTE
import seaborn as sns
import matplotlib.pyplot as plt
```

Step 2. Loading the Dataset

- Loaded using pandas.read_csv().
- The last column is used as the target label.
- Features (X) and target (y) are separated.

```
df = pd.read_csv("diabetes.csv")
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

Step 3. Data Preprocessing

- No null values → no imputation required.
- No categorical variables → no encoding needed.
- Feature Scaling is essential for SVM → used StandardScaler.

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

Step 4. Splitting the Dataset

- Used train_test_split() in two steps:
    1. First split into 70% Train and 30% Temp (Validation + Test)
    2. Then split Temp into 20% Validation and 10% Test
- Stratification used to maintain class distribution.

```
# Split Data: 70% train, 20% validation, 10% test
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=0.10, random_state=42, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=2/9, random_state=42, stratify=y_temp)
```

Step 5. Handling Class Imbalance

- Applied SMOTE on the training set.
- Balances the classes by generating synthetic samples for the minority class.

```
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train_scaled, y_train)
```

Step 6. Feature Scaling

- Applied StandardScaler:
    - Fit on training data.
    - Transformed train, validation, and test data using the same scaler to avoid data leakage.

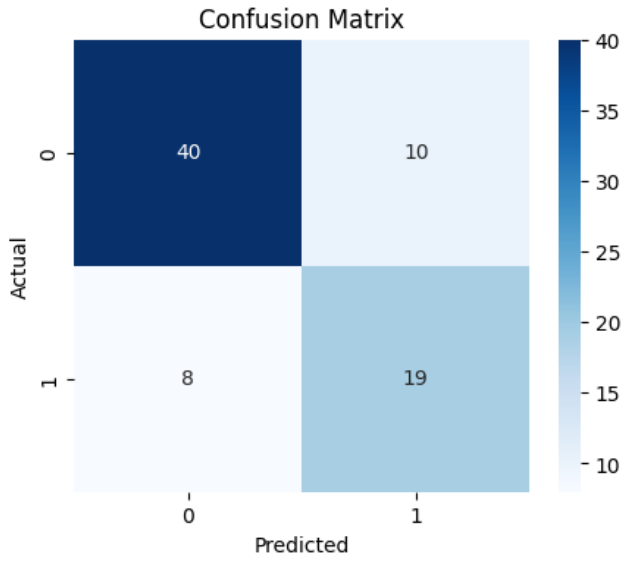
Step 7. Model Training with Hyperparameter Tuning

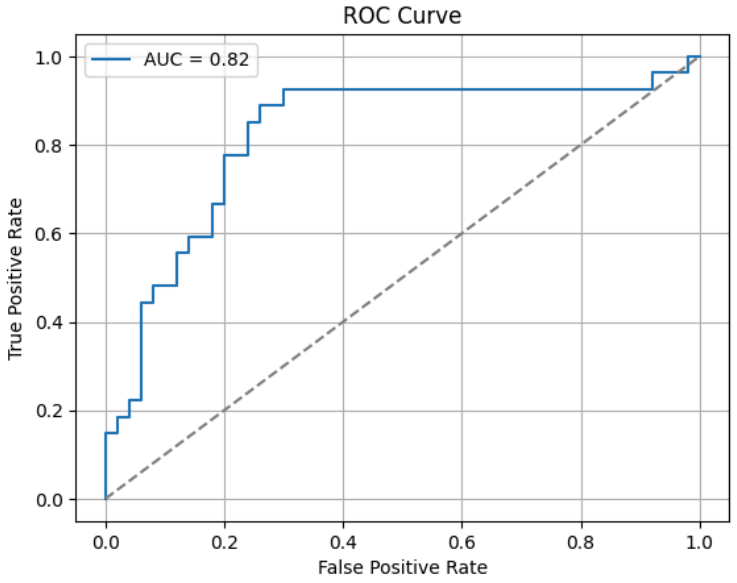
- Used Support Vector Classifier (SVC) with GridSearchCV for tuning:
    - Parameters like 'C', 'kernel', and 'gamma' were tuned.
    - Cross-validation used to select the best combination.

```
svm = SVC(probability=True)
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}
grid = GridSearchCV(svm, param_grid, cv=5)
grid.fit(X_train_bal, y_train_bal)
best_model = grid.best_estimator_
```

Step 8. Evaluation and Metrics

```
Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.80      0.82        50
           1       0.66      0.70      0.68        27

    accuracy                           0.77        77
   macro avg       0.74      0.75      0.75        77
weighted avg       0.77      0.77      0.77        77

Accuracy: 0.7662337662337663
```





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

Model Evaluation Summary (SVM Classifier)

- Test Accuracy: 76.6%
    - The model performs consistently on unseen data, indicating decent generalization ability.

Class-wise Performance (from Classification Report)

Class 0 (Non-Diabetic)

- Precision: 83%
  → When the model predicts "Non-Diabetic", it's correct 83% of the time.

- Recall: 80%
  → The model correctly identifies 80% of actual "Non-Diabetic" cases.

- F1-Score: 82%
  → Balance between precision and recall is strong.

Class 1 (Diabetic)

- Precision: 66%
  → Around two-thirds of the predictions for "Diabetic" are correct.

- Recall: 70%
  → The model is able to detect 70% of actual diabetic patients.

- F1-Score: 68%
  → Performance is moderately good, but there's room for improvement in catching diabetic cases more accurately.

Conclusion

- The SVM model achieves good accuracy and performs fairly well in distinguishing between diabetic and non-diabetic individuals.
- It detects the majority of diabetic cases (**70% recall**) while keeping the false positives at a reasonable level.
- Slight improvements in recall and precision for the minority class (diabetic) could further enhance the model's clinical usefulness.

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

Ans.

1. Required Libraries

```
import pandas as pd, numpy as np

from sklearn.linear_model import Ridge

from sklearn.preprocessing import PolynomialFeatures, StandardScaler

from sklearn.pipeline import Pipeline

from sklearn.model_selection import train_test_split, GridSearchCV

from sklearn.metrics import r2_score, mean_squared_error

import matplotlib.pyplot as plt

import seaborn as sns
```

We make use of:

- Pipeline to organize the sequence of preprocessing steps: polynomial feature generation, feature scaling, and applying ridge regression.

- GridSearchCV to find the best hyperparameters via cross-validation.

- r2_score and mean_squared_error to evaluate model accuracy.

2. Defining a Class for Regression

```
class RegressionModel:

    def __init__(self, model_pipeline, param_grid):

        ...
```

This class neatly encapsulates model fitting, evaluation, and hyperparameter optimization.
It is modular and flexible, making it applicable to other regression use-cases with minimal changes.

3. Importing the Dataset

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

df = pd.read_excel(url)
```

The dataset consists of several real estate-related features, such as:

- Distance to the nearest MRT station
- Count of nearby convenience stores
- Age of the building
- Latitude and longitude coordinates

4. Cleaning and Splitting the Data

```
df = df.drop(columns=['No'])  # Removing the serial number column

X = df.drop(columns=['Y house price of unit area'])  # Feature matrix

y = df['Y house price of unit area']  # Target vector
```

- X: All input features (excluding the house price)
- y: The target variable representing house price per unit area

5. Splitting the Dataset for Training and Testing

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

- Data is divided using a 70/30 ratio
- random_state=42 ensures consistent splits during re-runs

6. Constructing the Pipeline and Setting Hyperparameters

```
pipeline = Pipeline([

    ('poly', PolynomialFeatures()),
```

```
    ('scaler', StandardScaler()),
    ('ridge', Ridge())
])
```

- PolynomialFeatures: Adds non-linear transformations to the input
- StandardScaler: Ensures features have zero mean and unit variance (crucial for ridge regression)
- Ridge: Applies L2 regularization to control model complexity

Hyperparameter grid for tuning:

```
param_grid = {
    'poly__degree': [2, 3, 4],
    'ridge__alpha': [0.1, 1, 10, 100]
}
```

- We test polynomial degrees from 2 to 4
- Ridge penalty (alpha) values from 0.1 to 100 are evaluated

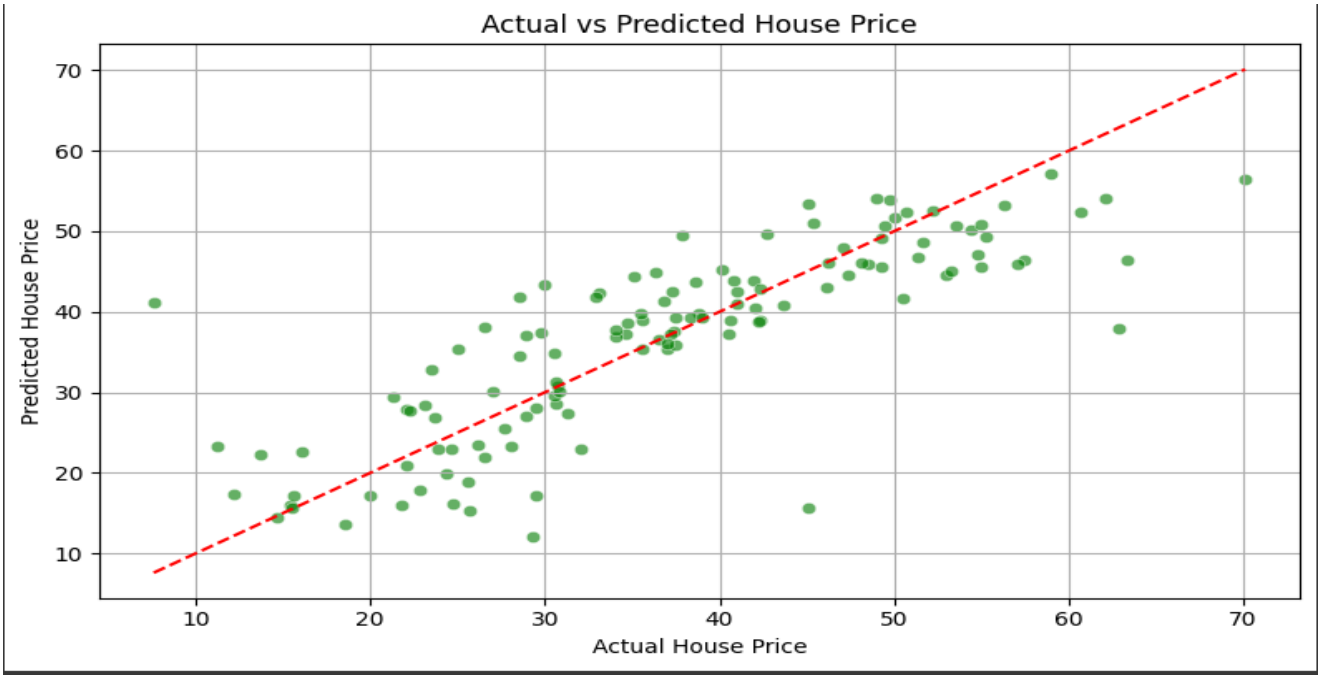
7. Model Training and Testing

```
best_model = reg_model.train(X_train, y_train)
y_test_actual, y_pred = reg_model.evaluate(best_model, X_test, y_test)
```

- The model is trained using 5-fold cross-validation during hyperparameter search
- The best configuration found is used to generate predictions
- Performance metrics:
    - R^2: How well the model explains variation in the target
    - Adjusted R^2: Corrects R^2 for number of predictors used
    - MSE: The average squared error between predicted and actual values

8. Visualizing Predictions

```
sns.scatterplot(x=y_test_actual, y=y_pred)
plt.plot([y_test_actual.min(), y_test_actual.max()], [y_test_actual.min(), y_test_actual.max()], 'r--')
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual vs Predicted House Prices")
plt.show()
```



- Scatterplot compares actual and predicted prices
- A red dashed line shows where predictions would lie if they were perfect

Final Model Results

Best Parameters: {'poly__degree': 2, 'ridge__alpha': 1}

R^2 Score: 0.6552

Adjusted R^2 Score: 0.6376

Mean Squared Error: 57.6670

- The model successfully explains about 66% of the variation in house prices
- The adjusted R^2 accounts for model complexity and offers a more realistic performance measure

Why a Perfect R^2 (>0.99) May Be Unrealistic

- Real datasets often contain noisy or incomplete data
- Not all influential factors may be captured (e.g., neighborhood popularity, future infrastructure plans)
- While polynomial features can increase expressiveness, excessive complexity can cause overfitting
- Ridge regression helps reduce this overfitting but cannot recover information not present in the dataset

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

Ans.

Key Features of the Wine Quality Dataset:

The dataset includes several physicochemical attributes of wine that help predict wine quality (on a scale of 0–10). Common features include:

Feature	Description	Importance
fixed acidity	Tartaric and other non-volatile acids	Affects stability and taste
volatile acidity	Acetic acid content	High levels produce an unpleasant vinegar taste
citric acid	Citric acid content	Adds freshness and flavor
residual sugar	Sugar left after fermentation	Influences sweetness; excess can indicate incomplete fermentation
chlorides	Salt content	Affects salinity and taste
free sulfur dioxide	Free SO_2 preventing microbial growth	Impacts shelf life and preservation
total sulfur dioxide	Total SO_2 (free + bound)	Excess may lead to off-odors; affects health and taste
density	Mass-to-volume ratio	Related to sugar content and alcohol
pH	Acidity/alkalinity	Influences microbial stability and flavor
sulphates	Antimicrobial additive	Associated with wine preservation and bitterness
alcohol	Alcohol content	Major determinant of taste, body, and aroma
quality	(Target) Wine quality rating (0–10)	Label used for regression or classification

Importance of Each Feature in Predicting Wine Quality:

1. Alcohol: Strong positive correlation — higher alcohol often leads to better quality.

2. Volatile Acidity: Strong negative correlation — high levels reduce perceived quality.

3. Sulphates: Slight positive correlation — relates to preservation and flavor.
4. Citric Acid & pH: Contribute to the crispness and freshness of wine.
5. Residual Sugar: Important for sweetness, but high amounts in dry wines might be a flaw.
6. Density: Indirectly relates to alcohol and sugar — helpful but not directly impactful alone.

Insight: A combination of features — especially alcohol, acidity, and sulphates — better predicts quality than any single feature.

Handling Missing Data During Feature Engineering:

In many Kaggle versions of this dataset, no missing values are present.
However, if missing data existed, these are common strategies used:

1. Dropping Rows (Simple Removal):

- When to use: Small number of missing rows (<5%)
- Advantage: Easy, no distortion of data
- Disadvantage: Loss of potentially useful information

2. Mean/Median/Mode Imputation:

- When to use: Numerical data with random missingness
- Advantage: Simple and fast
- Disadvantage: Ignores feature relationships, can reduce variance

3. K-Nearest Neighbors (KNN) Imputation:

- Uses: Similar instances to fill missing values
- Advantage: Preserves multivariate relationships
- Disadvantage: Computationally expensive for large datasets

4. Regression Imputation:

- Idea: Predict missing values using regression from other features
- Advantage: More informed than mean/median
- Disadvantage: Risk of overfitting and propagation of error

5. Iterative Imputation (e.g., MICE):

- Advanced Technique: Iteratively models each feature with missing values as a function of other features
- Advantage: Most statistically robust
- Disadvantage: Slower, more complex

Conclusion:

- Key features like alcohol, volatile acidity, and sulphates are vital in predicting wine quality.
- Handling missing data carefully during feature engineering is critical to maintain model performance.
- Choose imputation techniques based on the nature of data, amount of missingness, and model complexity.