

Name :Shreyash Kamat

Div : D15C

Roll No : 22

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	11845.000000	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 the 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	Series M	1	premium unleaded (required)	335.0	6.0	MANUAL	rear wheel drive	2.0	Tuner,Luxury,High-Performance	...	Coupe	26	19	3916	46135	False	False	False	True
1	BMW	Series 1	1	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	Series 1	1	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	Series 1	1	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	Series 1	1	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	Series 1	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

```
▶ 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("Number 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 is the mean, and σ 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 : this scales values between a fixed range, typically [0,1], using **Min-Max scaling**:

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

where X_{\min} and X_{\max} 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 our 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 : Shreyash Kamat

Div : D15C

Roll No : 22

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.

```

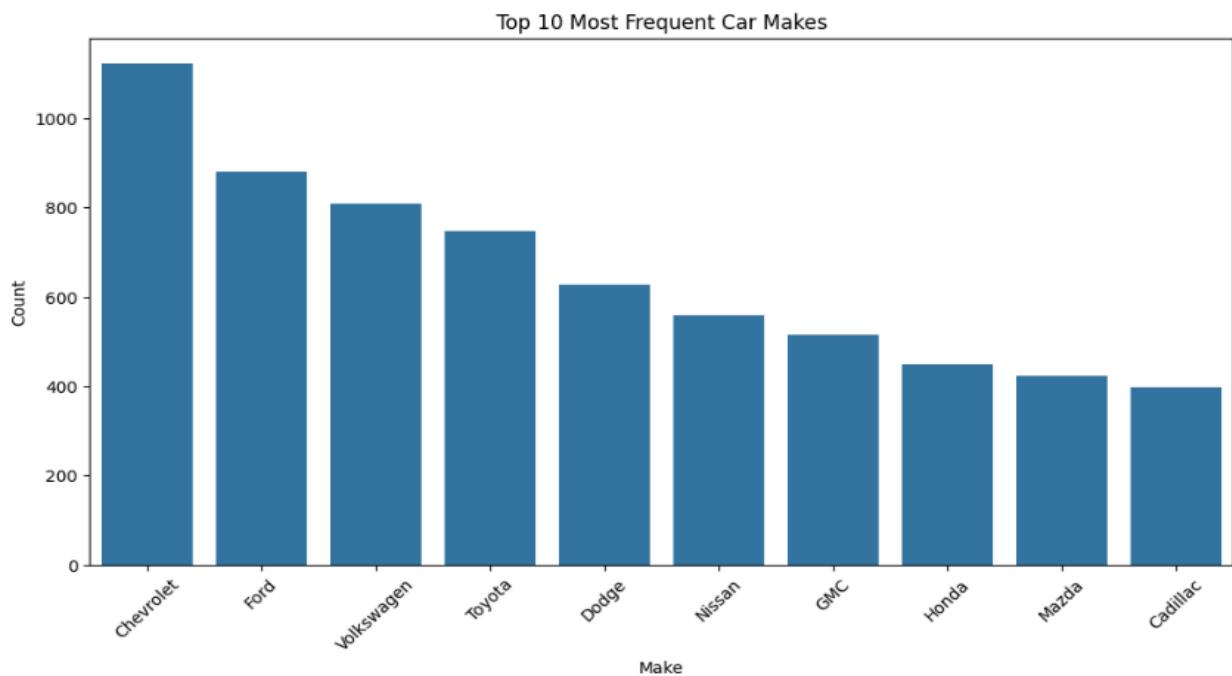
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

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.

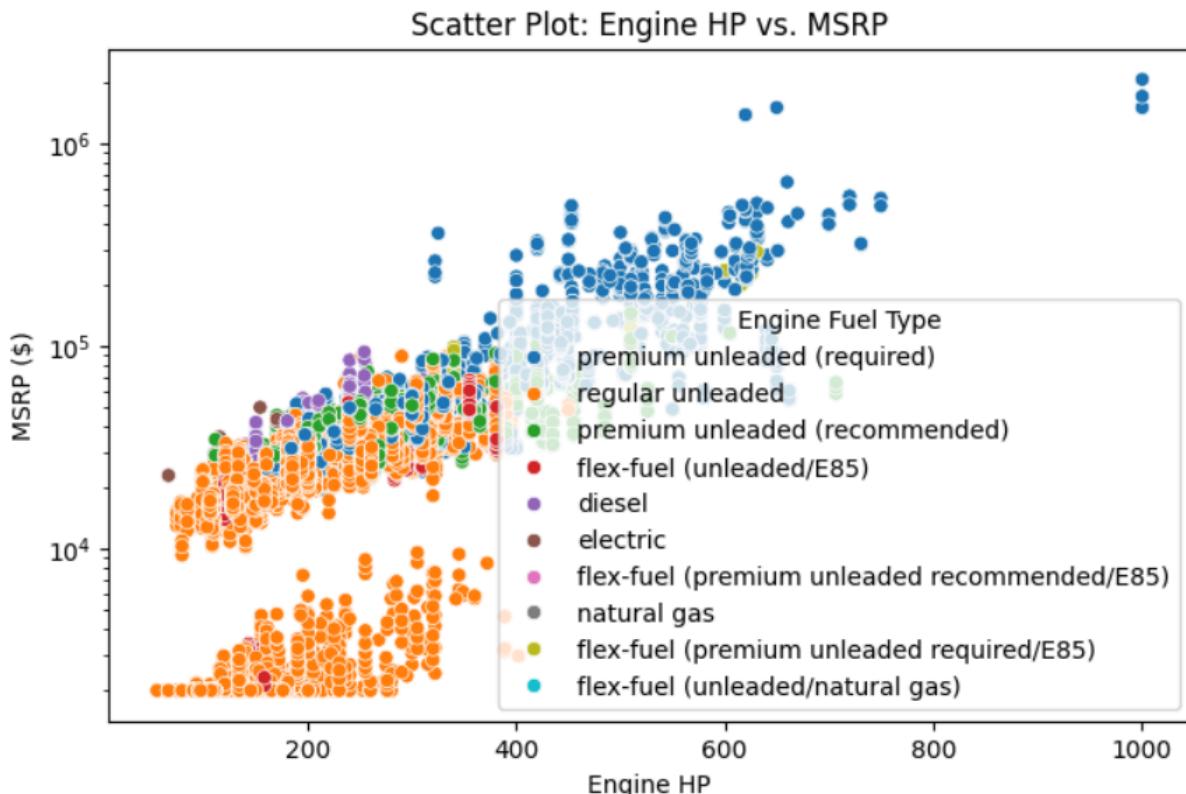
```
## 2. Scatter Plot, Box Plot, Heatmap
# Scatter Plot: Engine HP vs. MSRP
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df['Engine HP'], y=df['MSRP'], hue=df['Engine Fuel Type'])
plt.xlabel("Engine HP")
plt.ylabel("MSRP ($)")
plt.title("Scatter Plot: Engine HP vs. MSRP")
plt.yscale('log') # Log scale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

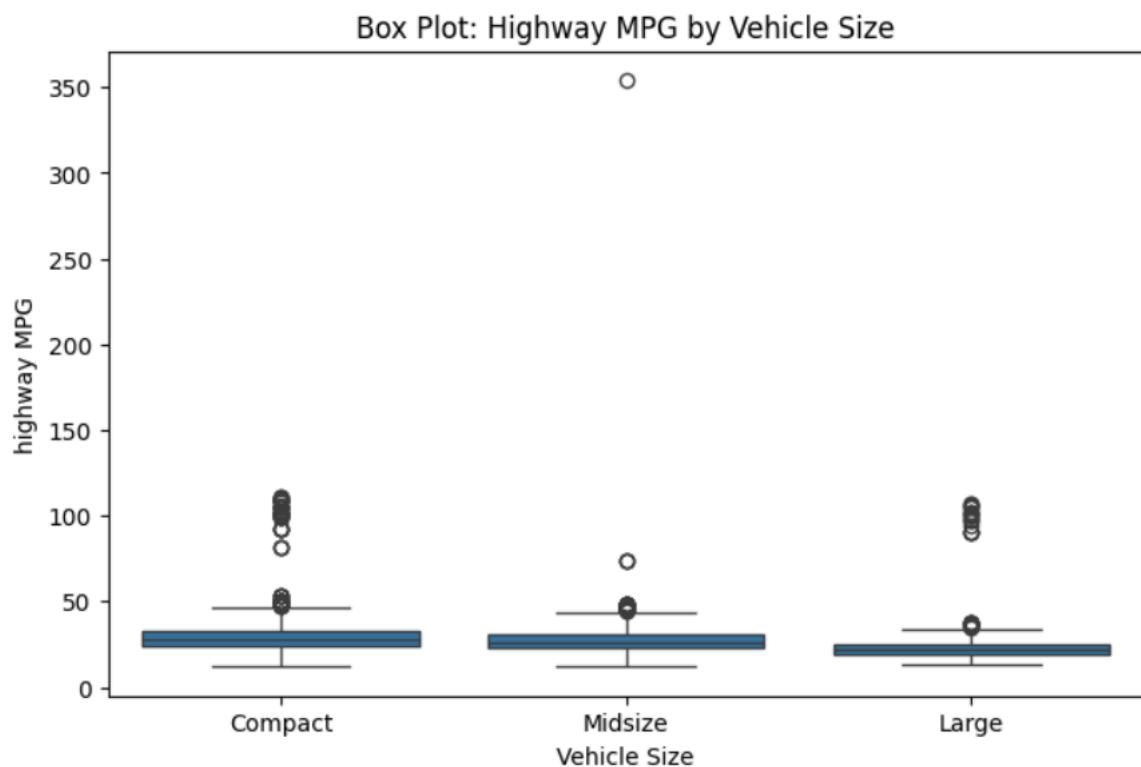
A **scatter plot** is a graph used to visualize the relationship between two continuous variables. It helps to identify correlations and trends in data. In this dataset, the scatter plot visualizes the relationship between Engine HP and MSRP, showing whether more powerful engines are associated with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



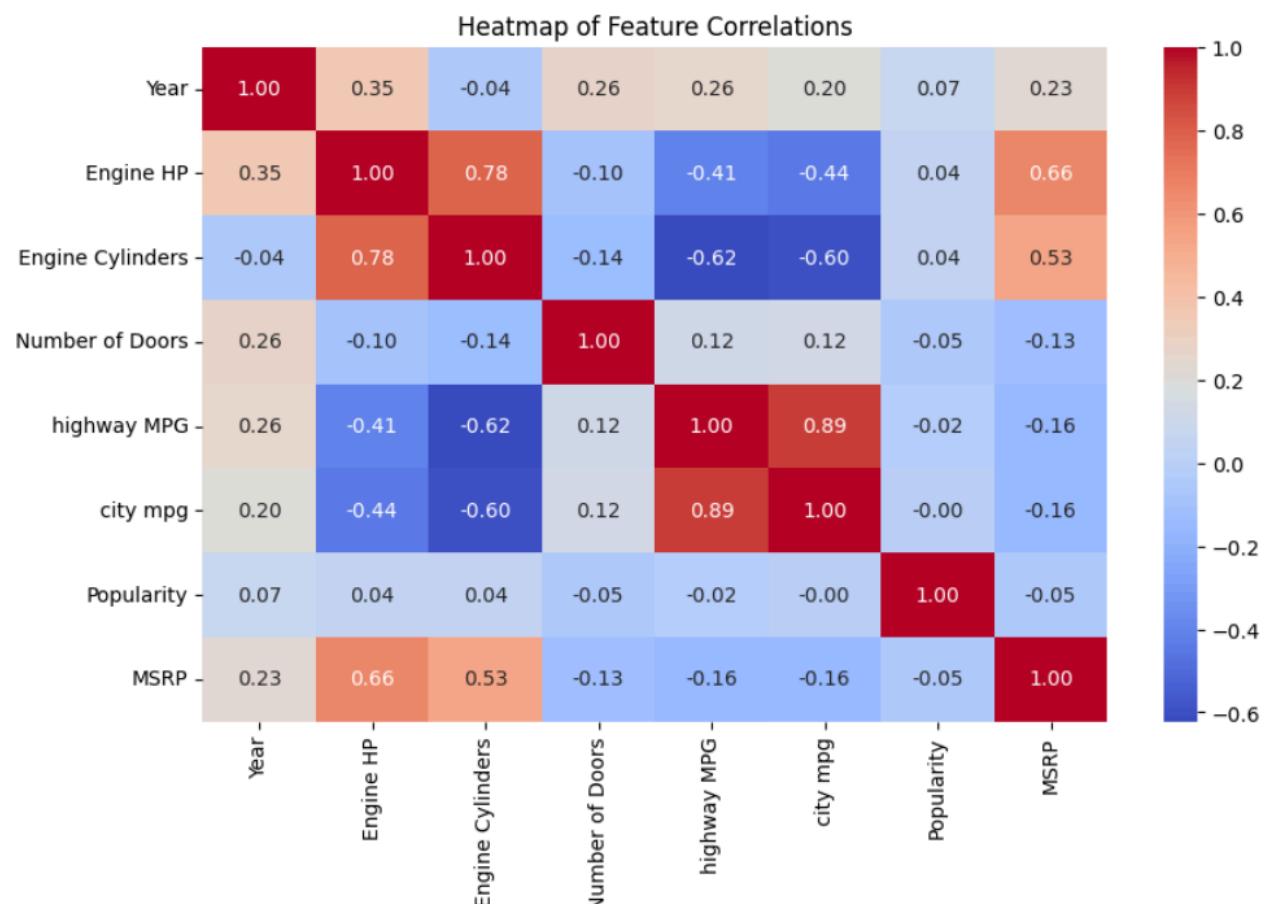
A **box plot** is used to show the distribution of data based on quartiles, highlighting the median, spread, and potential outliers.

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()
```

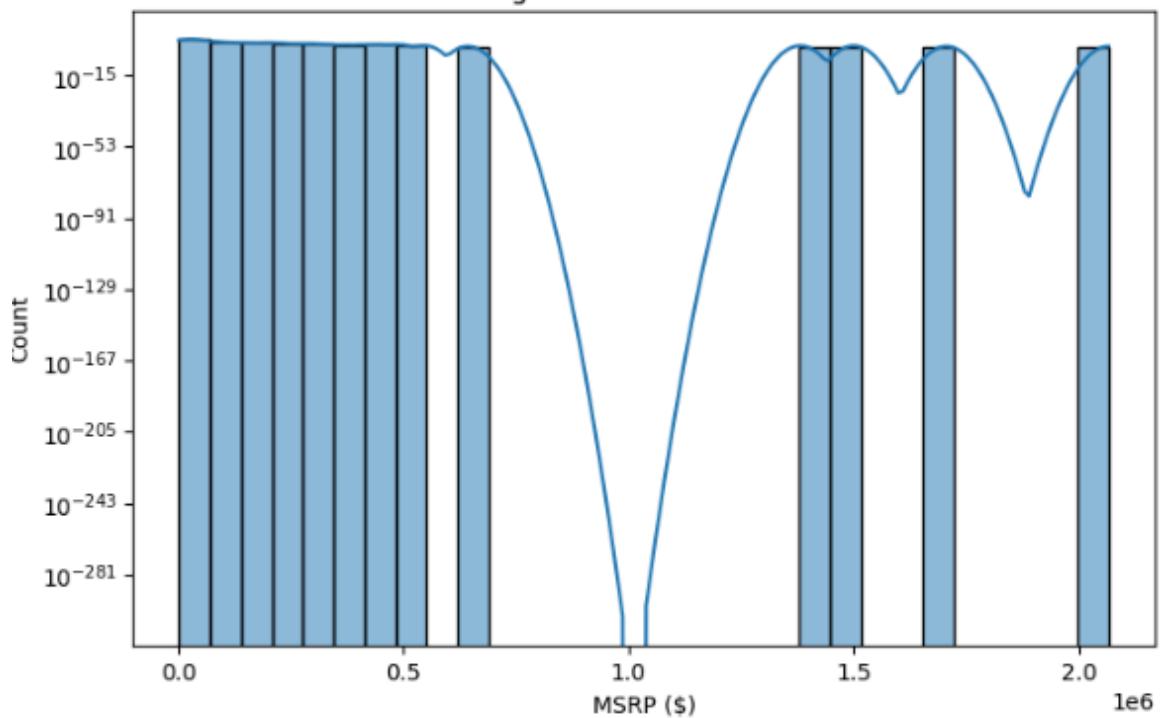
A histogram is a graphical representation that shows the distribution of a dataset by grouping data into bins. It is particularly useful for visualizing the frequency of values within a continuous range.

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.

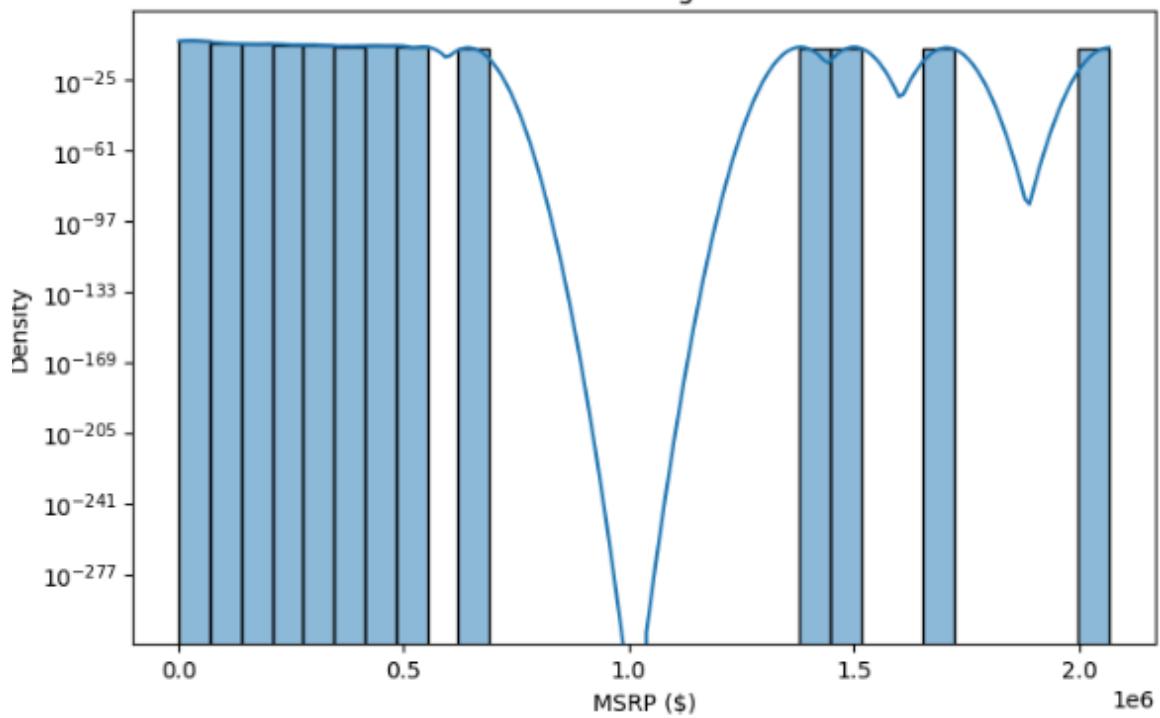
A normalized histogram represents the relative density (probability) rather than the raw count, showing how the distribution of data is spread out across the range of values.

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



Normalized Histogram: MSRP



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)]

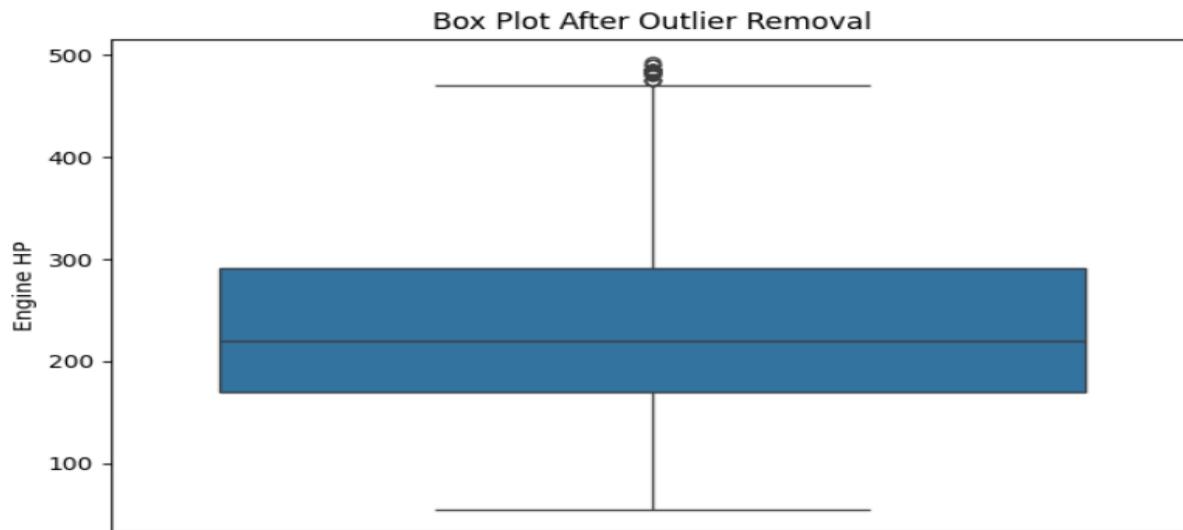
# Box Plot after Outlier Removal
plt.figure(figsize=(8, 5))
sns.boxplot(y=df_cleaned['Engine HP'])
plt.title("Box Plot After Outlier Removal")
plt.show()
```

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.

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)

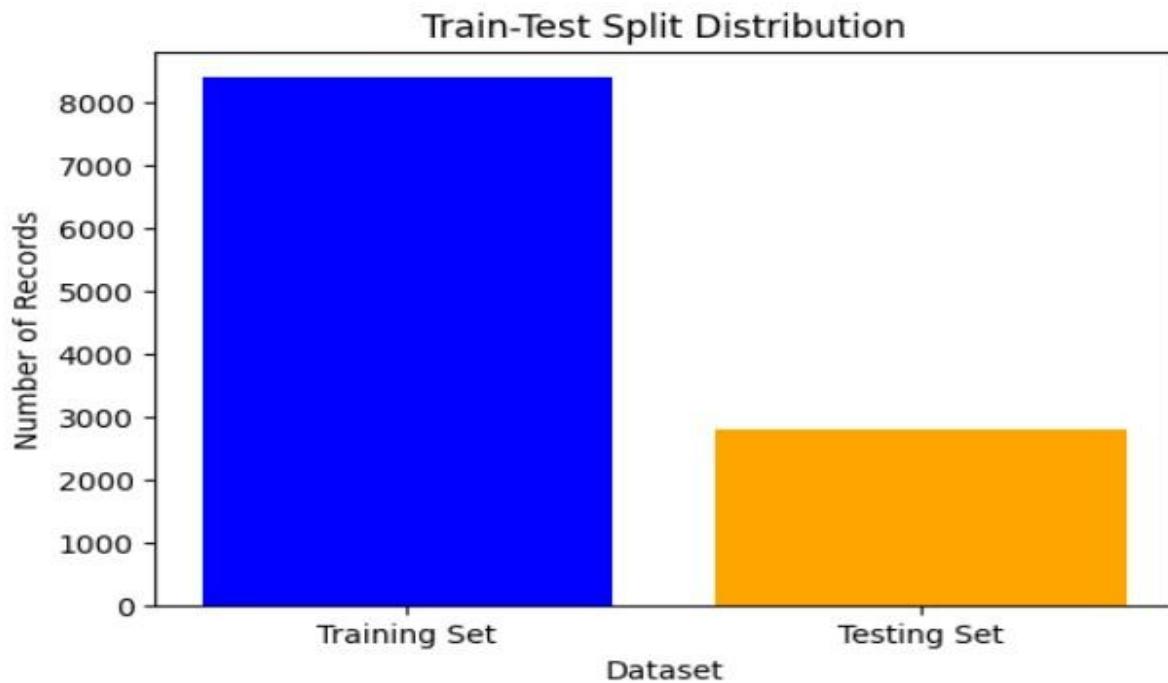
Training set size: (8395, 15)
Testing set size: (2799, 15)
```

Step 2: Visualizing the Data Split

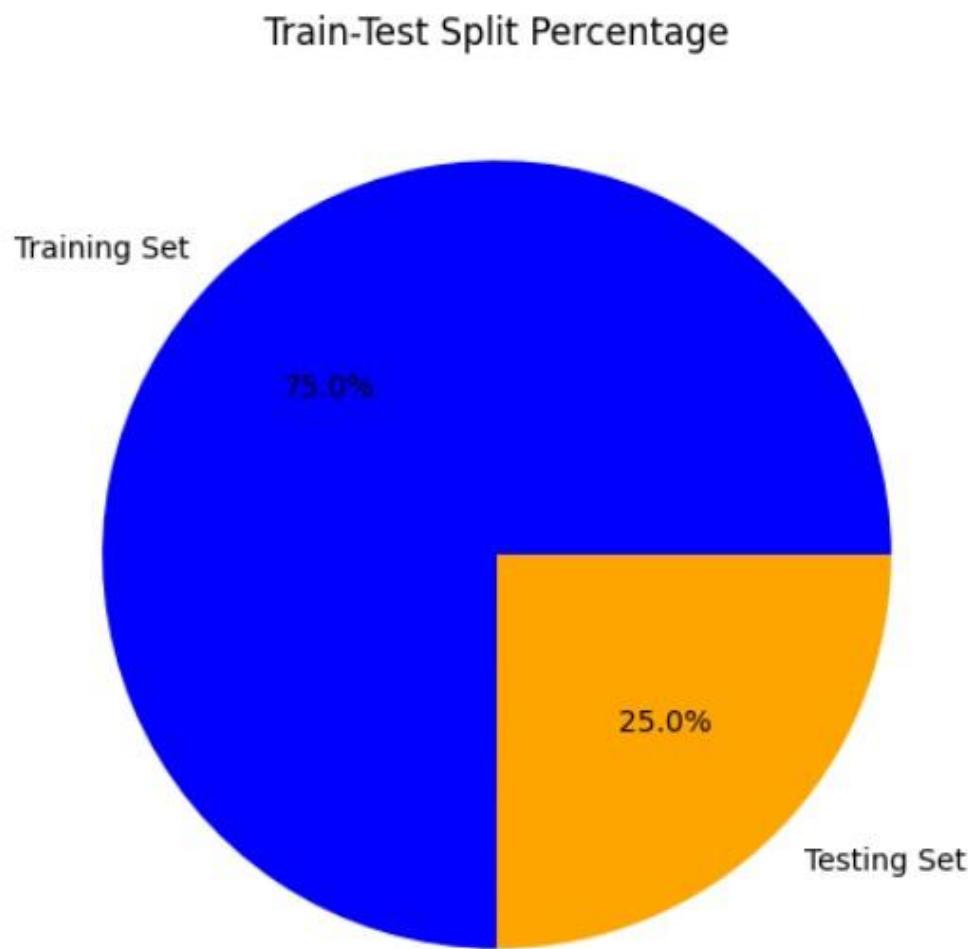
To confirm the partitioning proportions, the following visualizations were generated:

- **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.

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.

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.

```

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
Code 115833 but actions	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:

Code cell output

```
[[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.

```

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.

```

[ ] # 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.

R² 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 R² 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 **R² 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.

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.

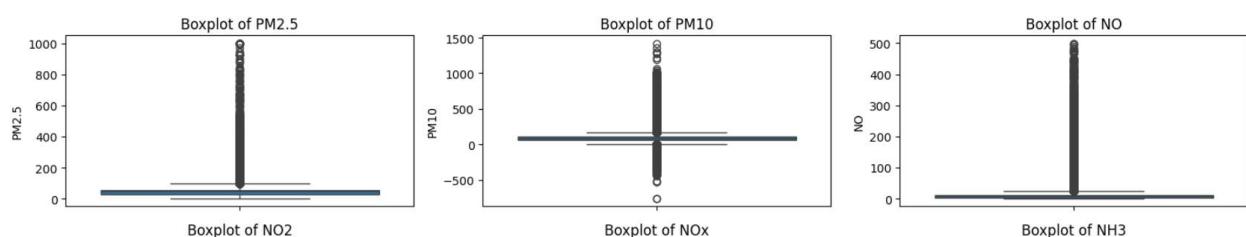
```

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.

Experiment 7

Aim: To implement different clustering algorithms.

Problem statement:

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

Theory:

CLUSTERING

It is basically a type of unsupervised learning method. An unsupervised learning method is a method in which we draw references from datasets consisting of input data without labelled responses. Generally, it is used as a process to find meaningful structure, explanatory underlying processes, generative features, and groupings inherent in a set of examples.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters.

For ex— The data points in the graph below clustered together can be classified into one single group. We can distinguish the clusters, and we can identify that there are 3 clusters in the below picture.

Applications of Clustering in different fields

1. Marketing: It can be used to characterize & discover customer segments for marketing purposes.
2. Biology: It can be used for classification among different species of plants and animals.
3. Libraries: It is used in clustering different books on the basis of topics and information.
4. Insurance: It is used to acknowledge the customers, their policies and identifying the frauds.
5. City Planning: It is used to make groups of houses and to study their values based on their geographical locations and other factors present.

6. Earthquake studies: By learning the earthquake-affected areas we can determine the dangerous zones

Clustering Algorithms

When choosing a clustering algorithm, you should consider whether the algorithm scales to your dataset. Datasets in machine learning can have millions of examples, but not all clustering algorithms scale efficiently. Many clustering algorithms work by computing the similarity between all pairs of examples. This means their runtime increases as the square of the number of examples n , denoted as $O(n^2)$ in complexity notation. $O(n^2)$ algorithms are not practical when the number of examples are in millions.

1. Density-Based Methods:

These methods consider the clusters as the dense region having some similarities and differences from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example DBSCAN (Density-Based Spatial Clustering of Applications with Noise), OPTICS (Ordering Points to Identify Clustering Structure), etc.

2. Hierarchical Based Methods:

The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two category

1. Agglomerative (bottom-up approach) 2.

Divisive (top-down approach)

examples CURE (Clustering Using Representatives), BIRCH (Balanced Iterative Reducing Clustering and using Hierarchies), etc.

3. Partitioning Methods:

These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example K-means, CLARANS (Clustering Large Applications based upon Randomized Search), etc.

4. Grid-based Methods:

In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operations done on these grids are fast and independent of the number of data objects, for example STING (Statistical Information Grid), wave cluster, CLIQUE (Clustering In Quest), etc.

Dataset Description: [adult.csv](#)

The **Adult Income dataset** contains demographic data of individuals, sourced from the 1994 US Census. It includes features like **age**, **education**, **occupation**, **hours per week**, etc., and aims to predict whether a person earns **more than \$50K or not** annually. This is a classic dataset used for **classification tasks** in machine learning.

```
[1] import pandas as pd

file_path = "adult.csv"
df = pd.read_csv(file_path)

df.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	40	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40	United-States	<=50K

Next steps: [Generate code with df](#) [View recommended plots](#) [New interactive sheet](#)

```
[2] df.dtypes
```

	dtype
age	int64
workclass	object
fnlwgt	int64
education	object
education.num	int64
marital.status	object
occupation	object
relationship	object
race	object
sex	object
capital.gain	int64
capital.loss	int64
hours.per.week	int64
native.country	object

Here we are going to see implementation of K-means and DB-SCAN clustering algorithms.

1) K-means clustering

1. Objective

To group data points into distinct clusters based on feature similarity using the K-Means algorithm.

2. Why the Elbow Method?

The **Elbow Method** helps determine the **optimal number of clusters (k)** by plotting:

- **X-axis:** Number of clusters (**k**)
- **Y-axis:** Within-Cluster Sum of Squares (WCSS / Inertia)

We select the “**elbow point**” – where the decrease in WCSS slows down – as the best **k**.

Now lets see the actual implementation.

```
▶ # we are selecting 2 columns....hrs per week and age for kmeans clusterization
data = df[['age', 'hours.per.week']]

[4] # missing values with the median..not mode or mean
data = data.fillna(data.median())

[5] from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data)
```

We are primarily selecting 2 columns on which our clustering will be based i.e Age and Working hours per week. Filling the missing values with median here.

```

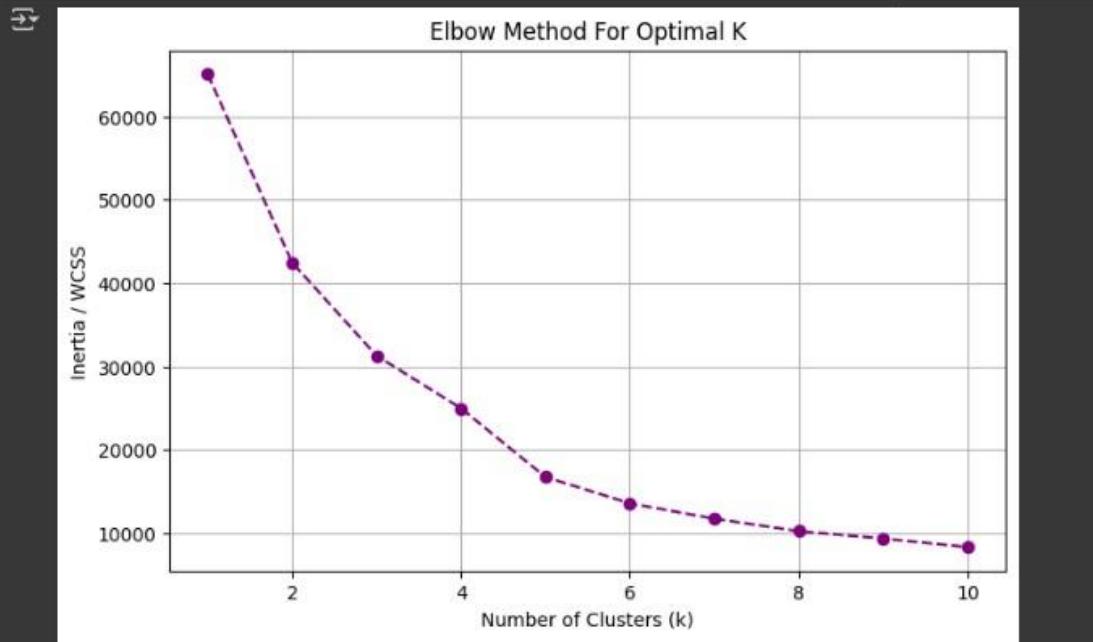
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Elbow method to find the best value for k
inertia = [] # to store WCSS (Within-Cluster-Sum-of-Squares)

# Try from k=1 to k=10
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(scaled_data) # use scaled data
    inertia.append(kmeans.inertia_)

# Plotting the elbow graph
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker='o', linestyle='--', color='purple')
plt.title('Elbow Method For Optimal K')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia / WCSS')
plt.grid(True)
plt.show()

```



Using the elbow method to find the optimal number clusters(k) that we need . We have chosen k=5.

```

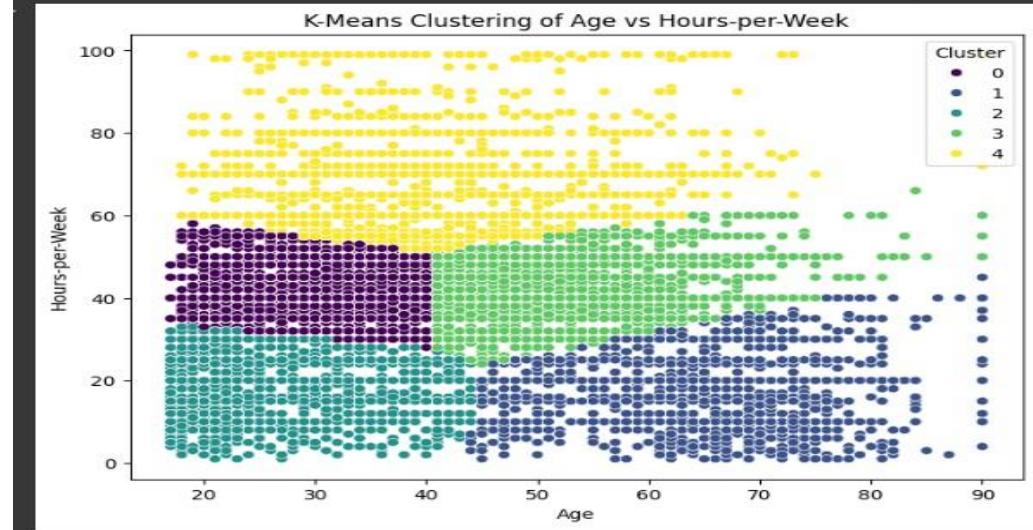
from sklearn.cluster import KMeans

# using 5 clusters for now
kmeans = KMeans(n_clusters=5, random_state=42)
df['Cluster'] = kmeans.fit_predict(scaled_data)

# visualizing
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8,6))
sns.scatterplot(x=df['age'], y=df['hours.per.week'], hue=df['Cluster'], palette='viridis')
plt.title('K-Means Clustering of Age vs Hours-per-Week')
plt.xlabel('Age')
plt.ylabel('Hours-per-Week')
plt.show()

```



The **elbow point** (e.g., at $k = 5$) indicates that 3 clusters give a good balance between **model accuracy** and **simplicity**.

Each data point is assigned to the nearest cluster based on **Euclidean distance**. The result reveals **natural groupings** or patterns in the dataset.

Silhouette Score

- **Purpose:** Measures how well each data point fits within its assigned cluster compared to other clusters.
- **Range:** **-1 to +1**
 - **+1** → Perfect clustering
 - **0** → Overlapping clusters
 - **Negative** → Misclassified points

```
▶ from sklearn.metrics import silhouette_score  
  
# Assuming `X_scaled` is your feature data and `kmeans.labels_` has your cluster labels  
score = silhouette_score(scaled_data, kmeans.labels_)  
print("Silhouette Score:", score)  
  
⇒ Silhouette Score: 0.4413614980396684
```

Double-click (or enter) to edit

```
▶ from sklearn.metrics import davies_bouldin_score  
  
# Make sure these are defined:  
# X_scaled → Scaled feature data (used to fit KMeans)  
# kmeans.labels_ → Labels assigned by KMeans  
  
# Compute Davies-Bouldin Score  
db_score_kmeans = davies_bouldin_score(scaled_data, kmeans.labels_)  
print("Davies-Bouldin Score (K-Means):", db_score_kmeans)
```

⇒ Davies-Bouldin Score (K-Means): 0.7348734337435234

Double-click (or enter) to edit

The **Davies-Bouldin Score** obtained is **0.73**, which is **fairly low** and indicates that the clusters are **compact** (low intra-cluster distance) and **well-separated** (high inter-cluster distance). This suggests that **K-Means has performed reasonably well** in forming distinct clusters. Additionally, if your **Silhouette Score** is also high (close to 1), it further confirms that the clustering is effective.

2) DB-SCAN

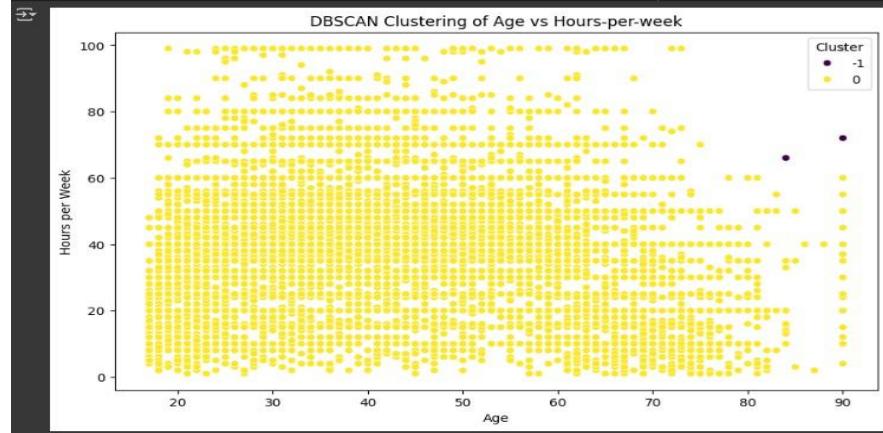
Feature	Description
Type	Density-based clustering algorithm
Assumption	Clusters are dense regions separated by low-density areas

Input Required	<code>eps</code> (radius), <code>min_samples</code> (minimum points in a dense region)
How It Works	Groups points closely packed together (high density), and labels others as noise
Performance	Slower for large, high-dimensional data
Handles Noise	<input checked="" type="checkbox"/> Yes (labels outliers as noise)
Shape of Clusters	Can handle arbitrary shapes (not just circular)
Scalability	Moderate; better for smaller or medium datasets
Use Case	When clusters have irregular shapes or you expect noise/outliers

Implementation:

```
[1] dbscan = DBSCAN(eps=0.5, min_samples=5) # Adjust eps and min_samples based on data
clusters = dbscan.fit_predict(df_selected_scaled)

# Add cluster labels to DataFrame
df_selected["Cluster"] = clusters
```



Age (X-axis)

Hours-per-week (Y-axis)

Points within **0.5 distance** of each other are considered neighbors

A point needs **at least 5 neighbors** to be a **core point**

Most points belong to Cluster 0 (yellow):

- This means **almost the entire dataset** is treated as **one dense cluster**.
- So people across all ages and working hour ranges generally fall into the same cluster.

A few outliers (Cluster -1, purple dots):

- These are individuals whose **Age** and **Hours-per-week** values are **unusual compared to others**.
- Example: A 90-year-old working 70+ hours — rare, hence considered noise.

Conclusion:

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

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

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

Experiment 8

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:

```

❶ 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))

❷ 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%20...      0      0
1  http://us.imdb.com/M/title-exact?GoldenEye%20...      0      1
2  http://us.imdb.com/M/title-exact?Four%20Rooms%20...      0      0
3  http://us.imdb.com/M/title-exact?Get%20Shorty%20...      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.

```

❷ 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)

```

```

❸ 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:

Div:

Roll no:

Experiment 9

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

Theory:

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

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

Key components and working:

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

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

2. How is data exploration done in Apache Spark?

Step 1: Loading the Data

Use `spark.read.csv()` or similar methods to load data from different sources such as CSV, JSON, or Parquet.

Step 2: Inspecting the Schema

Use `.printSchema()` or `.dtypes` to view the structure and data types of each column.

Step 3: Viewing Sample Data

Use `.show()` or `.head()` to look at a few initial records and get an idea of the data.

Step 4: Summary Statistics

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

Step 5: Checking for Missing Values

Use `.filter()` or `.where()` with `.isNull()` to identify null or missing entries.

Step 6: Counting Unique Values

Use `.groupBy(column).count()` to see frequency distributions of categorical or numerical features.

Step 7: Correlation Analysis

Use `.corr()` method to measure the linear correlation between numerical variables.

Step 8: Visualization

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

Conclusion:

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

Name:

Div:

Roll no:

Experiment 10

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

Theory:

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

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

- Batch Data is collected and stored over a time period, then processed as a single unit. This method is effective when immediate results are not required. It is widely used for ETL processes, periodic reporting, and historical data analysis, where latency is acceptable.
- Stream Data, on the other hand, is continuously produced and must be processed in near real-time. It is ideal for applications that require instant feedback, like fraud detection systems, stock trading platforms, live metrics dashboards, or user activity tracking. The main advantage of stream processing is its ability to provide timely insights and actions.

2. How Data Streaming Takes Place Using Apache Spark.

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

- Input Sources: Spark Streaming can ingest data from a variety of real-time sources such as Apache Kafka, Apache Flume, Amazon Kinesis, TCP sockets, or even files monitored in real-time.
- Micro-batch Architecture: Spark collects the data for a short interval (e.g., every 1 or 2 seconds) and creates a batch. These batches are then processed using the same

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

- DStreams (Discretized Streams): Spark represents incoming streaming data as DStreams, which are essentially a continuous series of RDDs. This abstraction allows developers to apply familiar operations such as map(), reduceByKey(), filter(), etc., to process data.
- Transformations and Actions: Developers can perform complex operations on the stream such as aggregations, joins, and windowed computations. The results can be sent to databases, file systems, or real-time dashboards.
- Output: The final output of stream processing can be stored or visualized in real-time, enabling instant feedback and data-driven decisions. Spark also supports stateful operations to maintain data across batches.

Conclusion:

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

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://bmccinfectdis.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.

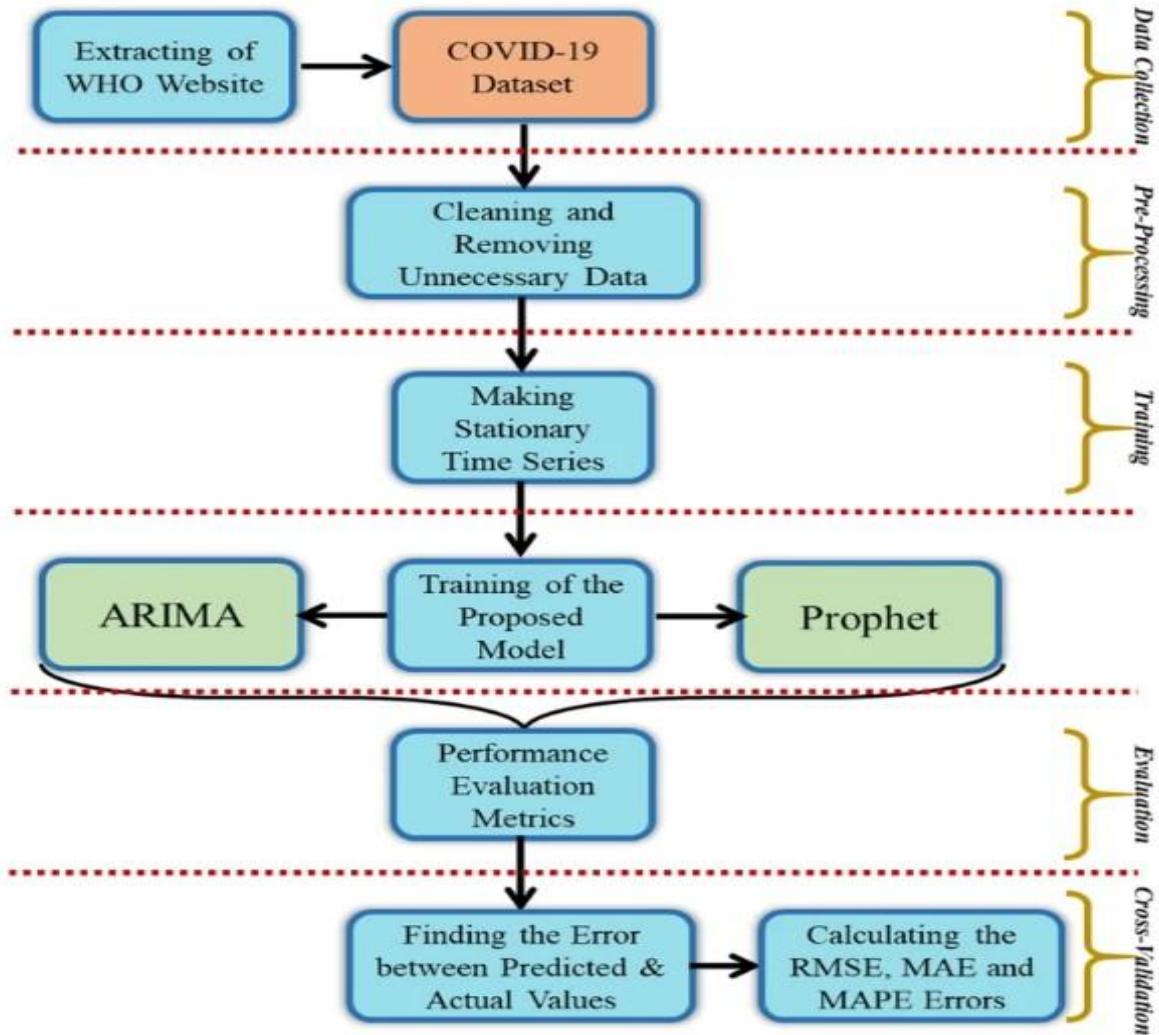
Evaluation Metrics:

- **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.

Q1 What is AI? Considering the COVID is pandemic situation how AI helped to survive and revolutionized our way of life with different applications?

→ Artificial Intelligence (AI) enables machines to think, learn and make decisions like humans. It includes technologies like machine learning, NLP & 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 & Education: AI enhanced virtual meeting, online learning & productivity tools.
- 4) Supply chain & Delivery: AI optimized logistics & enabled autonomous deliveries.
- 5) Mental Health Support: AI driven apps provided emotional & fitness assistance.

Q2 What are AI Agents terminology; explain with examples

→ 1) Agent: An entity that interacts with the environment & makes decision based on inputs

ex: A self-driving car perceives traffic signal & adjust speed accordingly.

2) Performance measures: Defines how successful an agent is in achieving its goals

ex: A self-driving car's performance measures could be minimizing accidents, fuel efficiency & travel time.

3) Behaviour / Action of Agent: The action an agent makes take based on its percepts
ex: A robotic vacuum cleaner moves around obstacles after detecting them.

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

5) Percept Sequence: The entire history of percepts received by an agent
ex: A chessplaying AI remembers all previous moves in the game before making the next moves

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

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

→ It consists of a 3x3 grid with 8 numbered tiles & one empty space where the objective is to move the tiles around to match a predefined goal configuration.

Initial State

1	2	3
4		6
7	5	8

This is the random starting configuration of the 8 puzzle with the tiles placed in a non-goal configuration

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

Coal 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

Disadvantage:

- DFS can get stuck in deep non-optimal paths & 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

Q4 What is PEAS descriptor? Give PEAS descriptor for following

1) Taxidriver

- P: Minimize travel time, fuel efficiency, passenger safety, obey traffic rules
- E: Roads, traffic, passengers, weather, obstacles
- A: Steering, accelerator, brakes, turn signals, horn
- S: Camera, GPS, speedometer, radar, 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, notification
- 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 setting
- A: Audio output, digital instrument selection, file saving/export
- S: User inputs; style preferences, tempo, feedback from listeners, music theory constraints

4) An aircraft Autolander

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

E: Airspace runway, weather, wind, speed, visibility

A: Flight control, landing gears brakes

S: GPS airspeed indicators, gyroscope, radar, weather sensors

5) An essay essay Evaluator

P: Accuracy of grading consistency fairness grammar

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

A: Feedback generation score assignment highlighting error suggesting improvement

S: Optical character recognition : NLP grammar & spell check

6) A robotic sentry gun for the keck lab

P: Target accuracy threat detection efficiency response speed

E: Keck lab premises, intruders, lighting condition obstacles

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

S: Motion detectors, infrared sensors, camera, LIDAR, radar

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 / multi agent

→ i) Partially Observable: The bot may not have complete visibility

2) Stochastic: The environment is unpredictable

3) Sequential: Each decision bot makes affect future states -

4) Dynamic: The bookstore environment changes over time

- 5) Discrete: Bot chooses discrete choice (selecting books)
6) Multiagent: The bot interacts with multiple entities

Q6 Differentiate Model based & Utility based agent

Model Based Agent

Utility Based Agent

- | | |
|---|---|
| i) Maintains an internal model of the environment to make decision | Use a utility function to measure performance & make option choice |
| 2) Relies on stored knowledge & Updates the model | Choose actions based on maximizing expected utility |
| 3) Can adapt to changing environment by updating the internal model | More flexible & goal oriented
adapting changes dynamically |
| 4) Moderate complexity due to model maintenance | Higher complexity due to the need to compute utilities for different action |
| 5) A Self driving car that predicts pedestrian movement | A self driving car that evaluate route & selects the best one |

Q7 Explain the architecture of a knowledge based agent & Learning Agents

→ 1. Knowledge-Based Agent Architecture

- A knowledge-based agent is an intelligent that makes decisions using knowledge (KB) and reasoning mechanism

Architecture Component :

- 1) Knowledge base: Stores fact, rules & heuristic about the world
- 2) Inference Engine: Use logical reasoning (FOL) to derive new knowledge from the KB
- 3) Perception Module: Collects data from sensor & update the KB
- 4) Action Selection Module: Chooses appropriate actions based on reasoning outcomes
- 5) Communication Module: Allows interaction with other agents

Working Process

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

2) Learning Agent Architecture

- A learning agent improves its performance over time by learning from past experiences & interaction with the environment

Architecture components

- 1) Learning Element: Analyzes feedback from the environment and improves knowledge
- 2) Performance Element: Make decisions & execute action
- 3) Critic: Evaluates the agent's action & provide feedback
- 4) Problem Generator: Suggest exploratory action to improve learning

Working Process:

- The performance element select an action
- The critic evaluates the action & provides feedback
- The learning element updates the agent knowledge to improve future decision
- The problem generator suggests new strategies to explore better solution

a) Convert the following to predicate

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

$$\rightarrow \text{Car Available} \Rightarrow \text{Travel by Car (Anita)} \\ \neg \text{Car Available} \rightarrow \text{Travel by Bus (Anita)}$$

b. Bus goes via Andheri and Goregaon

$$\rightarrow \text{Goes Via (Bus, Andheri)} \wedge \text{Goes Via (Bus, Goregaon)}$$

c. Car has puncture \rightarrow it is not available

Puncture (car)

$$\text{Puncture (car)} \rightarrow \neg \text{Car Available}$$

will Anita travel via Goregaon use forward reasoning

From (c)

Puncture (car) is true

$$\text{As Puncture (car)} \rightarrow \neg \text{Car Available}$$

From (a)

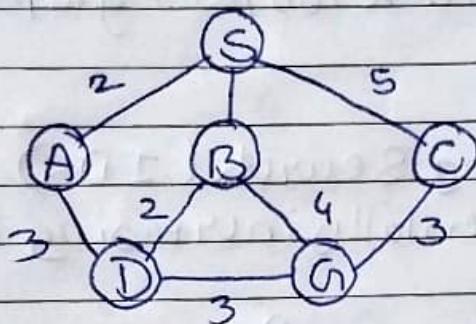
$\neg \text{Car Available}$, we use $\neg \text{Car Available} \rightarrow \text{Travel By Bus}$
(Auto)

From (b)

Coes via (Bus, Coregaon)

Since Anita travels by bus she will follow this route
Thus Anita will travel via Coregaon

Q10 Find the route from S to G using BFS



Current Node

1. Start at S

Queue = [S]

2. Dequeue S and explore its neighbouring nodes

Queue = [A, B, C]

3. Dequeue A and explore neighbour

Queue = [B, C, D]

4. Dequeue B and explore neighbour

Queue = [C, D, G]

5. Dequeue C and explore neighbour

Queue = [D, G]

6. Dequeue D

Queue = [G]

7. Dequeue G

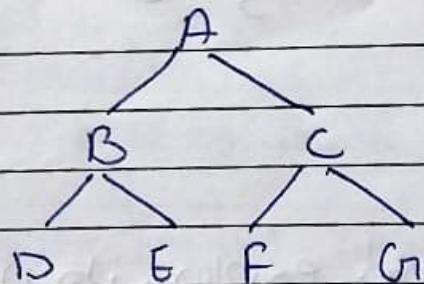
∴ G is our destination node BFS will stop here

Route from S to G : S → B → G

11. What do you mean by depth limited search? Explain Iterative deepening search with example
- Depth limited search (DLS) is an uninformed search algorithm that modifies DFS by introducing a depth limit L , preventing exploration beyond the defined level. This prevents infinite loop in graphs but risk missing beyond L

Iterative Deepening Search (IDS) combines DLS with BFS by incrementally increasing the depth bound

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

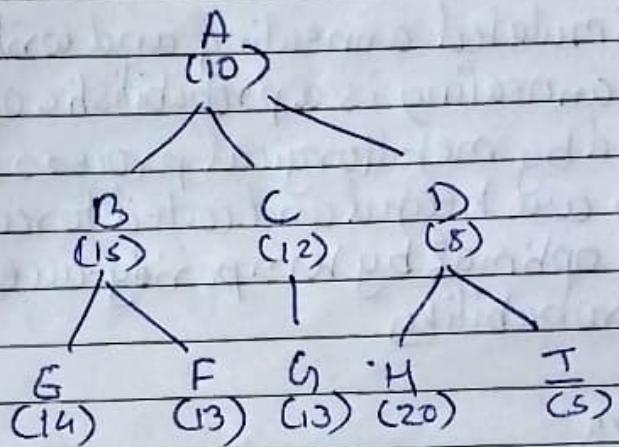
Nodes visited = A → B → D → E → C → F → G

Goal G is found

12. Explain hill climbing and its drawbacks in details with example. Also state limitation of steepest-ascent hill climbing

→ 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 ie 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

Drawback

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

2. Plateau - If sibling 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

Limitation 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 improvement

(3) Explain simulated annealing and write its algorithm

→ Simulated annealing is a probabilistic optimization algorithm

inspired by metallurgical process of annealing where materials are heated and cooled to reduce defects. It escape the local optimal 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

- Decrease temperature, T

3 Return best solution

Example: Travelling salesman problem

14 Explain A* algorithm with an example

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

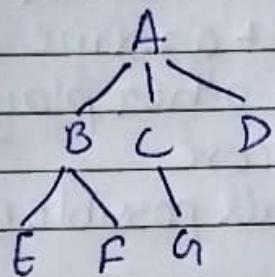
$$f(n) = g(n) + h(n)$$

$g(n)$ → 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	1
E	3	5
F	5	3
G	6	0

Steps

1. Start at root node A $f(A) = g(A) + h(A) = 0 + 6 = 6$

$f(B) = 1 + 4 = 5$ 2. Expand neighbour B, C, D

$$f(C) = 2 + 2 = 4$$

$$f(D) = 4 + 1 = 5$$

3. Choose lowest value that is $f(C)$

4. Expand neighbours of C

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

5. Coal reached at G with total cost 6

Advantages ->

- Efficient for finding the shortest path in weighted graph
- balances exploration by considering both $g(n)$ and $h(n)$

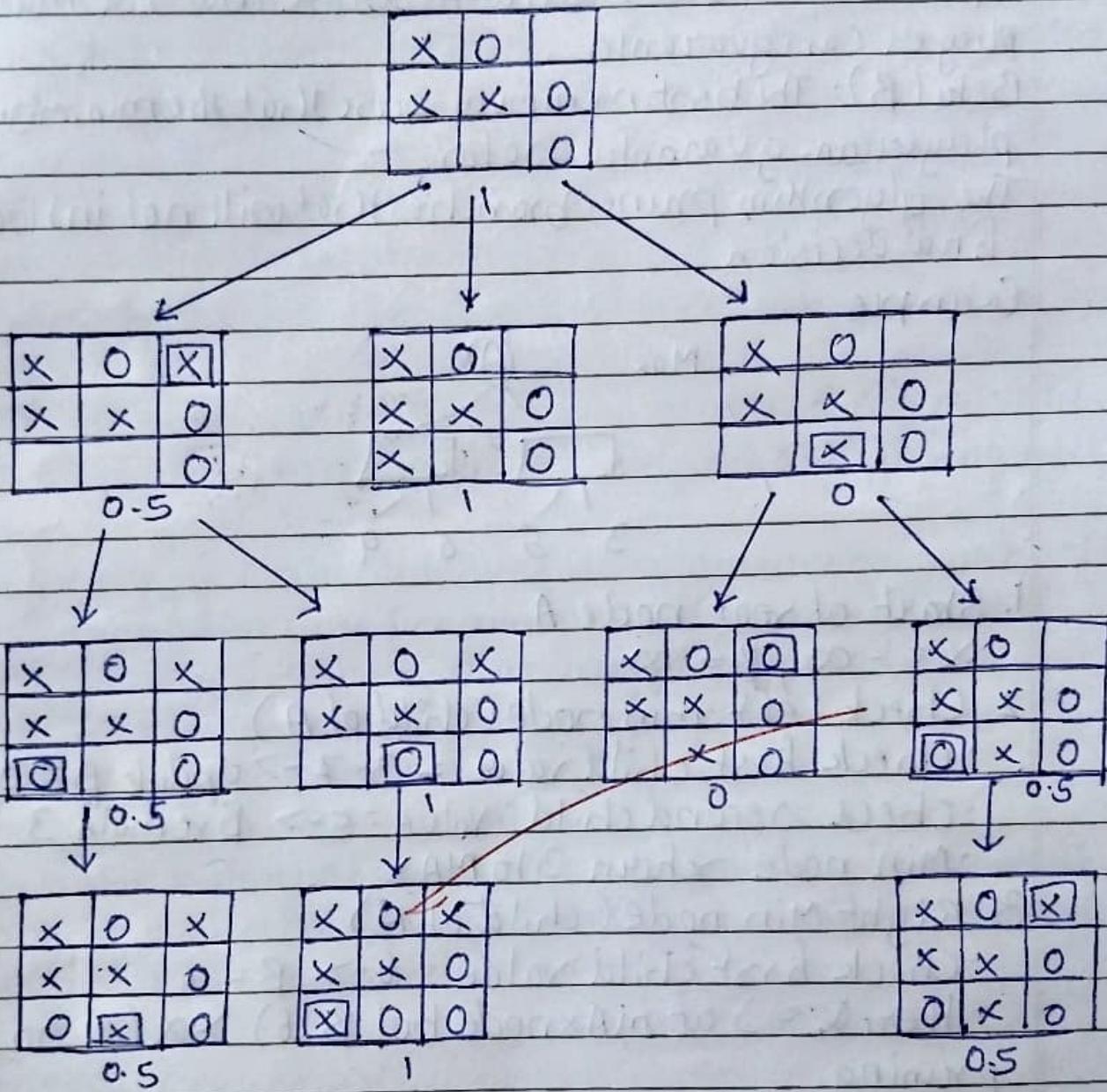
15 Explain MinMax algorithm and draw game tree for tic tac toe game

- > The min max algorithm is a decision making algorithm used in 2 players games. It assume 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 score
3. Max picks highest values from children MIN picks lowest values
4. Repeat until root node is evaluated

Game tree for Tic tac toe game:



16 Explain alpha-beta pruning algorithm for adversarial search with example

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

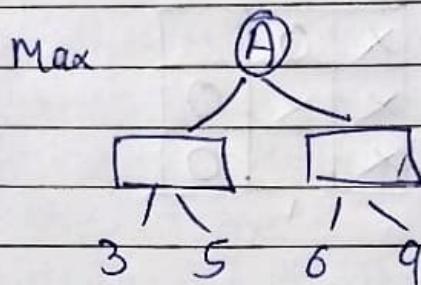
Alpha-beta pruning includes:

Alpha(α): The best maximum score that the maximising player can guarantee

Beta(β): The best minimum score that the minimising player can guarantee 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~~ \rightarrow update $\beta = 3$

• check second child: value = 5 $\rightarrow \beta$ remain 3

• Min node return 3 to MAX

3 Right Min node (child of A)

• check first child value = 6 $\rightarrow \beta = 6$

• Here $\alpha = 3$ at MAX node but $\beta(6) > \alpha(3)$ so no pruning

• Explore 2nd child (9) \rightarrow Here pruning will occur

\therefore 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

Q) Explain WUMPUS world environment giving its PEAS description. Explain how percept sequence is generated

→ The Wumpus world environment is a simple grid-based environment used in AI to study intelligent agent behaviour in uncertain environment. It is a turn based environment where an agent must navigate a cave to find gold while avoiding hazards like pits and a monster called wumpus.

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

G: 4x4 grid world containing the agent, wumpus, pit, gold

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

S: Agent perceives stench (near wumpus), breeze (near a pit), glitter (near gold), bump and scream

Percept Sequence generation

If is the history of all perception received by the agent. At each time step the agent perceives information based on its current location and surrounding

Example percept sequence

1. Agent starts at (1, 1)

- No breeze, no stench, no glitter → Safe square

2. Agent moves to (2, 1)

- Breeze detected → A pit is nearby but not in (current square)

3. Agent moves to (1, 2)

- Stench detected → wumpus is in adjacent cell

4. Agent moves to (2, 2)

- Glitter detected → gold is here

5. Agent moves back to (1, 1) and climb out

18 Solve the following crypto arithmetic problems

$$1 \cdot SEND + MORE = MONEY$$

→ Step 1:

M must be 1. The sum of 2 four digit number cannot be more than than 10000

$$\begin{array}{r} SEND \\ + MORE \\ \hline 1 ONEY \end{array}$$

Step 2

Now S must be 8 because there is 1 carry over from column E ON. 0 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 0 must be 0

~~$$\begin{array}{r} SEND \\ + MORE \\ \hline 1 ONEY \end{array}$$~~

Step 3

There cannot be a carry from column E ON because any digit to < 10 unless there is a carry from the column N R E and $E=9$. But this cannot be the case because then N would be 0 and 0 is already taken so $E < 9$ and there is not carry from this column therefore $S=9$ because $9+1=10$

Step 4

Case 1

$$\text{No carry } N+R=10+(N-1)=N+9$$

$$R=9$$

But 9 is already taken so will not work

Case 2

$$\text{Carry} : N + R + 1 = 9$$

$$R = 9 - 1 = 8 \text{ This must be the solution of } R$$

StepsLet's consider $E = 5$ or 6

$$E = 5$$

Then $D = 7$ and $Y = 3$ so this part will work but look at the column $N+E$. There is a carry from the column $D+Y$
 $N+8+1=16$ But then $N=7$ and 7 is taken by D therefore

$$E = 5$$

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

Now,

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

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

Step 6

The digits left are 7, 4, 3 and 2 are known. Then there is carry from column $D+Y$ so only pair that work is $D=7$ and $Y=2$

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

19 Consider the following axioms

All people who are graduating are happy

All happy people are smiling

Someone is graduating

- 1) Represent these axioms in first order predicate logic
2) Convert each formula to clause form
3) Prove that "Is someone smiling?" using resolution technique
Draw the resolution tree

1) In clause form = $\{\exists x \Gamma(x)\}$

we define the following predicate

$G(x)$ = x is graduating

$H(x)$ = x is happy

$S(x)$ = $\exists z$ 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 implication to clausal form

$\forall x (G(x) \rightarrow H(x))$

• 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 clauses

1) $\{\neg G(x), H(x)\}$

2) $\{\neg H(x), S(x)\}$

3) $\{G(a)\}$

2) Apply resolution

Resolve (1) $\{\neg G(x), H(x)\}$ with (3)
 $\{\neg G(a)\}$

Substituting $x = a$

$\{\neg G(a), H(a)\}$

\therefore 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

20 Explain Modus ponens with suitable example

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

It follows the form

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

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 $\rightarrow Q$

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

→ Forward chaining : It starts with given facts and applies inference rules to derive new fact 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 a cough

Inference

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

Backward chaining: If starts with goal and work backward by checking 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 \wedge Cough) \rightarrow Flu
2. (Sore Throat \wedge Fever) \rightarrow Cold

Process using backward chaining

1. We want to prove flu
2. Looking at rule 1: (Fever \wedge Cough) \rightarrow 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 condition are met we confirm Flu is true



AIDS-1
Assignment 02

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)

Dataset:

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

1. Mean

Step 1: Count the values

Total numbers = 20

Step 2: Calculate the sum

$$\begin{aligned} 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + \\ 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = 1611 \end{aligned}$$

Step 3: Apply formula

Mean = Sum of values ÷ Number of values

Mean = 80.55

2. Median

Step 1: Sort the data

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

Step 2: Since there are 20 values (even), find the average of 10th and 11th values

10th value = 81

11th value = 82

Median = 81.5

3. Mode

Frequency count:

- 76 appears 3 times
- 82 appears 2 times
- 90 appears 2 times
- Others appear once

Mode = 76

4. Interquartile Range (IQR)

Step 1: Sorted data (again)

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

Step 2: Divide into halves

- Lower half: 59, 64, 66, 70, 76, 76, 76, 78, 79, 81
- Upper half: 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

$Q_1 = \text{Median of lower half} = (5\text{th} + 6\text{th})/2 = (76 + 76)/2 = 76$

$Q_3 = \text{Median of upper half} = (5\text{th} + 6\text{th})/2 = (88 + 90)/2 = 89$

IQR = Q3 - Q1 = 89 - 76 = 13

Final Answers

- Mean: 81.05
- Median: 81.5
- Mode: 76
- Interquartile Range (IQR): 13

Q.2 1) [Machine Learning for Kids](#) 2) [Teachable Machine](#)

1. For each tool listed above
 - identify the target audience
 - discuss the use of this tool by the target audience
 - identify the tool's benefits and drawbacks
2. From the two choices listed below, how would you describe each tool listed above? Why did you choose the answer?
 - Predictive analytic
 - Descriptive analytic
3. From the three choices listed below, how would you describe each tool listed above? Why did you choose the answer?
 - Supervised learning
 - Unsupervised learning
 - Reinforcement learning

Tool 1: Machine Learning for Kids

Target Audience

- Primarily designed for kids, educators, and absolute beginners.
- Ideal for use in schools, coding clubs, and introductory AI workshops.

How They Use It

- Provides hands-on experience by letting users train models using examples.
- Users can build projects like:
 - Text classifiers (e.g., spam or not spam)
 - Image recognition (e.g., recognize types of food)
 - Chatbots using Natural Language Processing (NLP)
- The tool integrates with platforms like Scratch and Python to help visualize and use trained models in fun, interactive ways.

- Learning Goal: Understand the basic principles of AI and machine learning through experimentation.

Benefits

- Simple drag-and-drop interface, no coding required (optional Python).
- Supports real ML algorithms (powered by IBM Watson).
- Encourages experimentation and creativity.
- Great documentation and curriculum support for teachers.

Drawbacks

- Limited algorithm choices – not for advanced projects.
- May oversimplify concepts, missing depth for serious learners.
- Data privacy concerns when uploading images/texts (especially with kids).

Analytics Type

Predictive Analytic – Models are trained to make predictions based on labeled training data (e.g., predict whether an input is a cat or dog based on previous examples).

Learning Type

Supervised Learning – Users label examples during training (e.g., tagging images or texts), and the model learns to predict similar labels.

Tool 2: Teachable Machine (by Google)

Target Audience

- Geared toward students, artists, educators, and hobbyists.
- Suitable for people with no programming background.

How They Use It

- Users can create machine learning models by providing examples through webcam, microphone, or file uploads.
- Project types include:
 - Image classification (e.g., gestures, objects)
 - Audio recognition (e.g., speech commands)
 - Pose detection (e.g., yoga poses, dance moves)
- Learning Goal: Help users understand how training and prediction work in ML without needing code.

Benefits

- No installation or signup required, runs in the browser.
- Extremely intuitive interface – great for quick demos.
- Can export models to TensorFlow.js, TensorFlow Lite, or embed in websites/apps.
- Supports real-time feedback, making it engaging and interactive.

Drawbacks

- Limited customization – can't tweak model architecture or hyperparameters.
- Doesn't support advanced datasets or complex use cases.
- Data privacy issues if webcam/mic is used without consent.

Analytics Type

Predictive Analytic – Models are trained to make real-time predictions (e.g., detect a specific pose or sound based on prior training data).

Learning Type

Supervised Learning – Models are trained using labeled input (e.g., label each pose/image/audio before training).

Summary Table

Tool Name	Target Audience	Analytics Type	Learning Type	Benefits	Drawbacks
Machine Learning for Kids	Kids, educators, beginners	Predictive Analytic	Supervised Learning	Easy Scratch integration, great schools UI, for	Limited models, privacy concerns
Teachable Machine	Students, hobbyists	Predictive Analytic	Supervised Learning	No-code, real-time training, easy exports	No deep customization, privacy risks

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.

Real-World Example of Misleading Visualization**Example: Misleading COVID-19 Vaccine Uptake Surveys****Description:**

During the COVID-19 pandemic, several large-scale surveys aimed to estimate vaccine uptake among U.S. adults. Notably, the Delphi-Facebook survey, with approximately 250,000 responses per week, and the Census Household Pulse survey, with about 75,000 weekly responses, sought to provide insights into vaccination rates.

What Went Wrong:

Despite their extensive sample sizes, these surveys significantly overestimated vaccine uptake. By May 2021, the Delphi-Facebook survey overestimated uptake by 17 percentage points, and the Census Household Pulse survey by 14 percentage points, compared to benchmarks from the Centers for Disease Control and Prevention (CDC).

The primary issue was **sampling bias**. The surveys failed to adequately represent certain demographics, leading to skewed results. Additionally, the large sample sizes gave a false sense of precision, as the confidence intervals were narrow, yet centered around inaccurate estimates.

Impact:

These overestimations had significant implications:

- **Policy Decisions:** Public health officials relying on these surveys might have believed that vaccine uptake was higher than it actually was, potentially leading to premature relaxation of safety measures.
- **Resource Allocation:** Misjudging vaccination rates could have affected the distribution of vaccines and resources, leaving some communities underserved.
- **Public Perception:** Inaccurate data might have influenced public opinion, leading individuals to believe that herd immunity was closer than it truly was, possibly reducing the urgency to get vaccinated.

Citation:

Bradley, V. C., Kuriwaki, S., Isakov, M., Sejdinovic, D., Meng, X.-L., & Flaxman, S. (2021). *Unrepresentative Big Surveys Significantly Overestimate US Vaccine Uptake*. arXiv preprint arXiv:2106.05818

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

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

Requirements to satisfy

- Programming Language: Python
- Class imbalance should be resolved
- Data Pre-processing must be used
- Hyper parameter tuning must be used

- Train, Validation and Test Split should be 70/20/10
- Train and Test split must be randomly done
- Classification Accuracy should be maximized
- Use any Python library to present the accuracy measures of trained model

[Pima Indians Diabetes Database](#)

Objective:

Train a classification model (SVM / Naïve Bayes) on the given dataset, resolve class imbalance, perform preprocessing and hyperparameter tuning, and evaluate its prediction performance.

Step 1: Import Required Libraries

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

Step 2: Load the Dataset

```
df = pd.read_csv("Classification data.csv")
```

Step 3: Separate Features and Class Label

```
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

Step 4: Handle Class Imbalance

```
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

Step 5: Data Splitting (Train/Validation/Test = 70/20/10)

```
# Train (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X_resampled, y_resampled, test_size=0.3,
random_state=42, stratify=y_resampled)

# Validation (20%) and Test (10%) split from 30%
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, random_state=42,
stratify=y_temp)
```

Step 6: Data Preprocessing

```
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 7: Model Selection and Hyperparameter Tuning

```
params = {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
svc = SVC()
grid = GridSearchCV(svc, params, cv=3)
grid.fit(X_train_scaled, y_train)
```

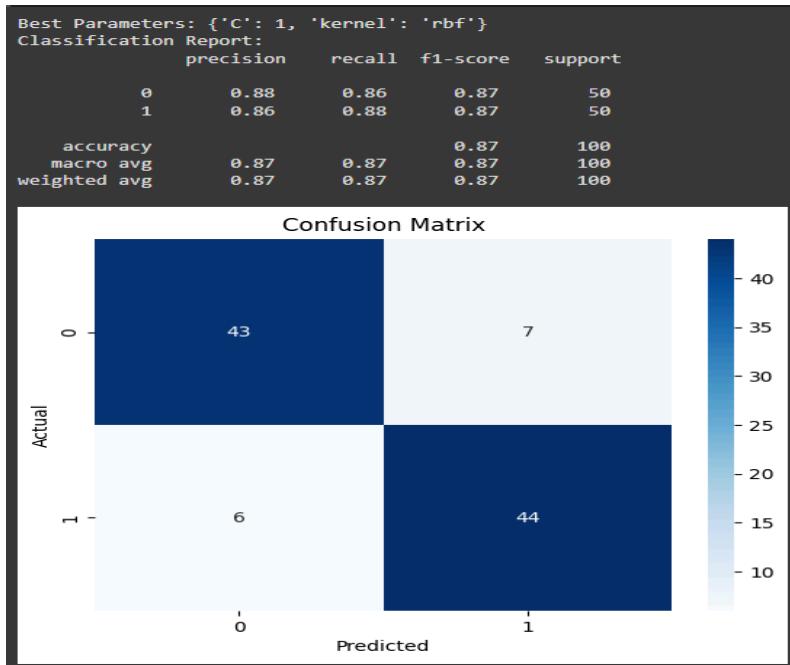
Step 8: Evaluate the Model

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

```
print("Best Parameters:", grid.best_params_)
print("Classification Report:\n", classification_report(y_test, y_pred))
```

Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



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

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

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

Requirements to satisfy:

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

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

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

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

(Refer any one)

Objective

To train a regression model using Ridge Regression with Polynomial Feature Expansion to predict real estate prices based on several features (like distance to MRT station, number of convenience stores, etc.). We aim to:

- Tune hyperparameters (alpha and polynomial degree)
- Use a modular, object-oriented design
- Evaluate using R², Adjusted R², and MSE
- Visualize predicted vs actual prices

Step-by-Step Explanation

1. Importing 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 use:

- Pipeline to streamline polynomial features → standardization → ridge regression.
- GridSearchCV for hyperparameter tuning
- r2_score, mean_squared_error for evaluation

2. Defining the RegressionModel Class

```
class RegressionModel:
    def __init__(self, model_pipeline, param_grid):
        ...
        • Encapsulates model training, evaluation, and hyperparameter tuning.
        • Reusable and extensible for other regression problems.
```

3. Loading the Dataset

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

The dataset contains real estate records including:

- Distance to MRT station
- Number of nearby convenience stores
- Age of building
- Geographic coordinates

4. Data Preprocessing

```
df = df.drop(columns=['No']) # Drop irrelevant index
X = df.drop(columns=['Y house price of unit area']) # Features
y = df['Y house price of unit area'] # Target
```

We define:

- X: all columns except house price
- y: the target variable (house price)

5. Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(...)
    • 70% training data, 30% testing
    • random_state=42 ensures reproducibility
```

6. Model Pipeline & Hyperparameter Grid

```
pipeline = Pipeline([
    ('poly', PolynomialFeatures()),
    ('scaler', StandardScaler()),
    ('ridge', Ridge())
])
    • Pipeline Components:
        ○ poly: adds non-linearity (degree=2 or more)
        ○ scaler: standardizes features (important for ridge!)
        ○ ridge: regularized regression
    • Parameter Grid:
param_grid = {
```

```

'poly_degree': [2, 3, 4],
'ridge_alpha': [0.1, 1, 10, 100]
}

```

We let GridSearchCV try different combinations of:

- Polynomial degrees: 2 to 4
- Ridge regularization strengths (alpha): 0.1 to 100

7. Training and Evaluation

```

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

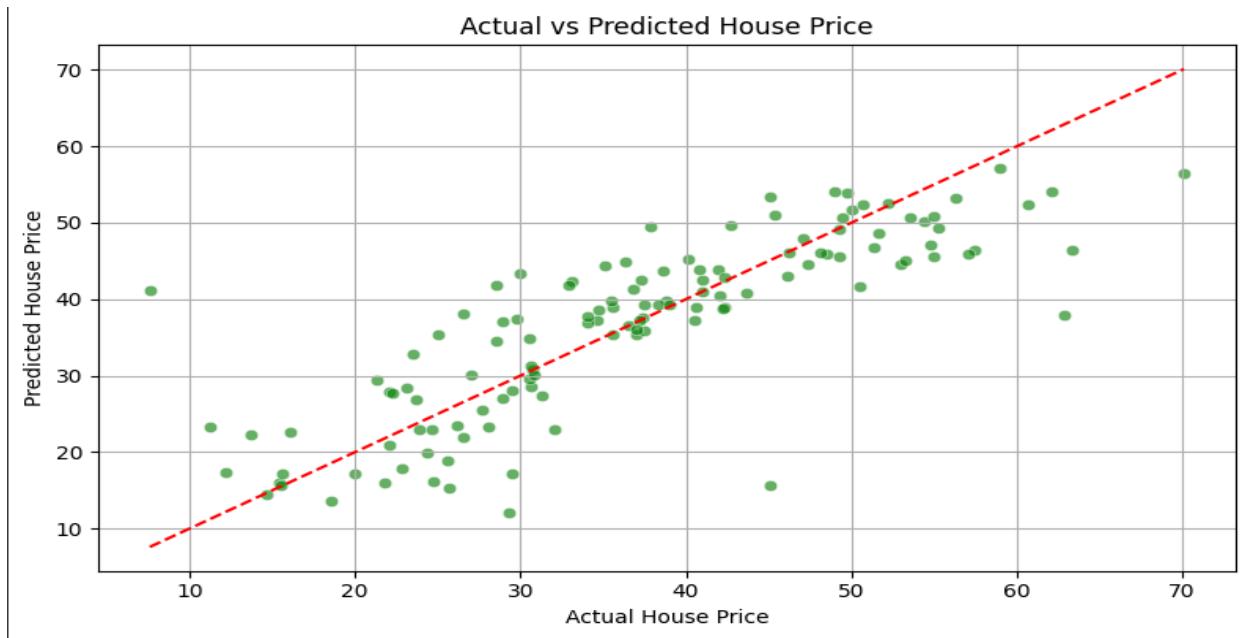
```

- The model is trained with 5-fold cross-validation
- Best estimator is used for prediction
- We calculate:
 - R²: proportion of variance explained
 - Adjusted R²: penalizes for extra features
 - MSE: average squared prediction error

8. Visualization

```
sns.scatterplot(x=y_test_actual, y=y_pred)
```

- Compares actual vs predicted prices
- Red dashed line shows ideal predictions (perfect match)



Final Results:

Best Params: {'poly_degree': 2, 'ridge_alpha': 1}

R² Score: 0.6552

Adjusted R² Score: 0.6376

Mean Squared Error: 57.6670

- The model captures ~66% of the variance in house prices.
- There's room for improvement, but this is reasonable for real-world data.
- Adjusted R² shows performance after accounting for model complexity.

Why we May Not Reach R² > 0.99

- Real-world datasets include noise, missing features, and non-linear interactions.
- Polynomial features help but too high a degree → overfitting.
- Ridge helps reduce overfitting, but can't add missing signal.

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).

1. Introduction

The Wine Quality dataset available on Kaggle is a well-known dataset used for regression and classification tasks. It includes physicochemical properties of red or white vinho verde wine samples, along with a quality score rated by wine tasters.

Dataset Link:

[Wine Quality Dataset on Kaggle \(Red and White\)](#)

2. Key Features of the Wine Quality Dataset

Feature	Description
fixed acidity	Tartaric acid content. Too much affects the taste, while too little may reduce the wine's stability.
volatile acidity	Acetic acid content. High levels give an unpleasant vinegar taste.
citric acid	Adds freshness and flavor. Small amounts are desirable.
residual sugar	Sugar remaining after fermentation. Influences sweetness and body of wine.
chlorides	Salt content. High amounts can negatively affect taste.
free sulfur dioxide	Prevents microbial growth. Too much affects flavor.
total sulfur dioxide	Sum of free and bound forms. High levels can cause off-flavors.
density	Indicates sugar and alcohol content. Important for fermentation control.
pH	Acidity level. Affects freshness, taste, and preservation.

sulphates	Wine preservative. Also affects flavor.
alcohol	One of the strongest predictors of quality. Higher alcohol usually improves perception.
quality (target)	Score between 0–10 given by tasters. This is what we aim to predict.

3. Importance of Features in Predicting Wine Quality

Based on correlation studies and model performance, some features contribute more to wine quality prediction:

- Alcohol: Strongest positive correlation with quality.
- Volatile Acidity: Strong negative impact on taste and quality.
- Sulphates: Moderate positive impact; enhances flavor.
- Citric Acid: Mild but noticeable effect on freshness.
- Density: Inversely related to alcohol; indirectly affects quality.

Using models like Random Forest or XGBoost, feature importance plots show alcohol, volatile acidity, sulphates, and citric acid as top predictors.

Handling Missing Data During Feature Engineering

The wine quality dataset from Kaggle does not contain missing values in its original form. However, if encountered (e.g., in modified or extended versions), these are the standard approaches:

Techniques for Handling Missing Data

Method	Description	Advantages	Disadvantages
Drop rows	Remove rows with missing values	Simple, avoids introducing bias	Risk of losing valuable data
Mean/Median Imputation	Replace missing values with mean/median	Easy to implement, preserves dataset size	Can distort distribution; ignores variability
Mode Imputation	Replace with most frequent value (used for categorical data)	Maintains consistency	Can bias the data if mode is too dominant
KNN Imputation	Use nearest neighbors to predict missing values	Considers feature relationships	Computationally expensive; sensitive to outliers
Multivariate Imputation (MICE)	Iteratively predicts missing data using regression models	Accurate and flexible	Complex and slow on large datasets

Recommended Technique:

- For this dataset, mean or median imputation would be ideal if data were missing, as most features are continuous.
- For advanced cases, KNN imputation or MICE could improve model performance by preserving relationships among variables.

Conclusion

- The most important predictors of wine quality are alcohol, volatile acidity, and sulphates.
- Although the original dataset is clean, it is essential to understand missing value imputation techniques.
- The choice of imputation method depends on the nature of data, the amount of missingness, and the balance between performance and simplicity.