Name: Komal Sabale
Class: D15C
Roll No: 45

EXPERIMENT -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

Data preprocessing

Data preprocessing is the process of transforming raw data into an understandable format. It is also an important step in data mining as we cannot work with raw data. The quality of the data should be checked before applying machine learning or data mining algorithms.

Why is Data Preprocessing important?

Preprocessing of data is mainly to check the data quality. The quality can be checked by the following:

- Accuracy: To check whether the data entered is correct or not.
- Completeness: To check whether the data is available or not recorded.
- Consistency: To check whether the same data is kept in all the places that do or do not match.
- Timeliness: The data should be updated correctly.
- Believability: The data should be trustable.

Interpretability: The understandability of the data.

Dataset: Aircraft Fleet Dataset:

Name: Komal Sabale
Class: D15C
Roll No: 45

1) Loading Data in Pandas

Step 1: Load Data in Pandas

```
import pandas as pd
df = pd.read_csv("Fleet Data.csv")
df.head()
```

	Parent Airline	Airline	Aircraft Type	Current	Future	Historic	Total	Orders	Unit Cost	Total Cost (Current)	Average Age
0	Aegean Airlines	Aegean Airlines	Airbus A319	1.0	NaN	3.0	4.0	NaN	$90	$90	11.6
1	Aegean Airlines	Olympic Air	Airbus A319	NaN	NaN	8.0	8.0	NaN	$90	$0	NaN
2	Aegean Airlines	Aegean Airlines	Airbus A320	38.0	NaN	3.0	41.0	NaN	$98	$3,724	7.5
3	Aegean Airlines	Olympic Air	Airbus A320	NaN	NaN	9.0	9.0	NaN	$98	$0	NaN
4	Aegean Airlines	Aegean Airlines	Airbus A321	8.0	NaN	NaN	8.0	NaN	$115	$919	10.3

2)Description of the dataset.

```
Dataset Shape: (1583, 11)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1583 entries, 0 to 1582
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   Parent Airline        1583 non-null   object
 1   Airline               1583 non-null   object
 2   Aircraft Type         1583 non-null   object
 3   Current               859 non-null    float64
 4   Future                188 non-null    float64
 5   Historic              1113 non-null   float64
 6   Total                 1484 non-null   float64
 7   Orders                348 non-null    float64
 8   Unit Cost             1548 non-null   object
 9   Total Cost (Current)  1556 non-null   object
 10  Average Age           820 non-null    float64
dtypes: float64(6), object(5)
memory usage: 136.2+ KB
Dataset Description:           Current      Future     Historic        Total      Orders  \
count  859.000000  188.000000  1113.000000  1484.000000  348.000000
mean    24.033760    3.382979    14.513028    24.955526   26.419540
std     41.091234    4.656331    23.763373    46.651526   43.024179
min      1.000000    1.000000     1.000000     1.000000    1.000000
25%      5.000000    1.000000     3.000000     4.000000    5.000000
50%     12.000000    2.000000     7.000000    11.000000   13.500000
75%     26.500000    4.000000    16.000000    27.000000   28.250000
max    718.000000   38.000000   325.000000   952.000000  400.000000

       Average Age
count   820.000000
mean     10.115000
std       6.859362
min       0.100000
25%       5.000000
50%       8.900000
75%      14.500000
max      39.000000
```

`df.info()`: Provides an overview of the dataset, including:

Name: Komal Sabale
Class: D15C
Roll No: 45

- Number of rows and columns.
- Data types of each column (e.g., int, float, object).
- Number of non-null (non-missing) values in each column.

`df.describe()`: Generates summary statistics for numeric columns, such as:

- `count`: Number of non-missing values.
- `mean`: Average value.
- `std`: Standard deviation.
- `min`, 25%, 50% (median), 75%, and `max`: Percentile values.

3) Drop columns that aren't useful: Columns like Orders and Average Age which had very little data may not contribute to analysis and we can't replace them with mean as it would make a lot of data irrelavant. Removing irrelevant columns reduces complexity.

```
# Drop columns that are not useful for analysis
df = df.drop(columns=['Orders', 'Future','Average Age'])

# Display the first few rows after dropping columns
df.head()
```

	Parent Airline	Airline	Aircraft Type	Current	Historic	Total	Unit Cost	Total Cost (Current)
0	Aegean Airlines	Aegean Airlines	Airbus A319	1.0	3.0	4.0	$90	$90
1	Aegean Airlines	Olympic Air	Airbus A319	NaN	8.0	8.0	$90	$0
2	Aegean Airlines	Aegean Airlines	Airbus A320	38.0	3.0	41.0	$98	$3,724
3	Aegean Airlines	Olympic Air	Airbus A320	NaN	9.0	9.0	$98	$0
4	Aegean Airlines	Aegean Airlines	Airbus A321	8.0	NaN	8.0	$115	$919

4)Drop rows with maximum missing values.

```
df = df.dropna(thresh=len(df.columns) - 2)
```

- Drops rows where more than 2 columns have missing (NaN) values.

Name: Komal Sabale
Class: D15C
Roll No: 45

```
print(f"Rows before dropping: {df.shape[0]}")

# Drop rows with maximum missing values
df = df.dropna(thresh=len(df.columns) - 2)

# Number of rows after dropping
print(f"Rows after dropping: {df.shape[0]}")

# Display the first few rows after dropping rows
df.head()
```

```
Rows before dropping: 1583
Rows after dropping: 1492
```

	Parent Airline	Airline	Aircraft Type	Current	Historic	Total	Unit Cost	Total Cost (Current)
0	Aegean Airlines	Aegean Airlines	Airbus A319	1.0	3.0	4.0	$90	$90
1	Aegean Airlines	Olympic Air	Airbus A319	NaN	8.0	8.0	$90	$0
2	Aegean Airlines	Aegean Airlines	Airbus A320	38.0	3.0	41.0	$98	$3,724
3	Aegean Airlines	Olympic Air	Airbus A320	NaN	9.0	9.0	$98	$0
4	Aegean Airlines	Aegean Airlines	Airbus A321	8.0	NaN	8.0	$115	$919

5)Take care of missing data.

`df.fillna(df.mean())`: Replaces missing values (NaN) in numeric columns with the mean of that column.

```
# Count missing values before filling
missing_before = df.isna().sum().sum()
print(f"Missing values before: {missing_before}")

# Fill missing values with the mean for numerical columns
# df = df.fillna(df.mean())
df = df.fillna(df.select_dtypes(include=['number']).mean())

# Count missing values after filling
missing_after = df.isna().sum().sum()
print(f"Missing values after: {missing_after}")
```

```
Missing values before: 1081
Missing values after: 17
```

6)Create dummy variables.

Name: Komal Sabale
Class: D15C
Roll No: 45

`pd.get_dummies()`: Converts categorical variables into dummy variables (binary indicators: 0 or 1).

- Example: The `Gender` column becomes `Gender_Male` (1 if Male, 0 otherwise).

`columns=['...']`: Specifies which columns to convert.
`drop_first=True`: Avoids the "dummy variable trap" by dropping one dummy variable to prevent multicollinearity.

	Parent Airline	Airline	Current	Historic	Total	Unit Cost	Total Cost (Current)	Aircraft Type_ATR 42–300F/–320F	Aircraft Type_ATR 42–600	Aircraft Type_ATR 42/72	...	Aircraft Type_McDonnell Douglas MD–90	Aircraft Type_Saab 2000	Aircraft Type_Saab 340	Aircr Type_Suk Super
0	Aegean Airlines	Aegean Airlines	1.000000	3.000000	4.0	$90	$90	False	False	False	...	False	False	False	F
1	Aegean Airlines	Olympic Air	24.270907	8.000000	8.0	$90	$0	False	False	False	...	False	False	False	F
2	Aegean Airlines	Aegean Airlines	38.000000	3.000000	41.0	$98	$3,724	False	False	False	...	False	False	False	F
3	Aegean Airlines	Olympic Air	24.270907	9.000000	9.0	$98	$0	False	False	False	...	False	False	False	F
4	Aegean Airlines	Aegean Airlines	8.000000	14.629091	8.0	$115	$919	False	False	False	...	False	False	False	F

7)Find out outliers (manually)

We first identified the outliers using Python by calculating the interquartile range (IQR) and marking values outside the IQR as outliers. Then, we imported the data into Excel, where we marked the outliers with a 'Yes' label. Finally, we used Excel's filtering and deletion tools to manually remove the outliers from the dataset.

Name: Komal Sabale
Class: D15C
Roll No: 45

```python
import numpy as np

# Select only numeric columns
numeric_cols = df.select_dtypes(include=[np.number])

# Calculate Q1 (25th percentile) and Q3 (75th percentile)
Q1 = numeric_cols.quantile(0.25)
Q3 = numeric_cols.quantile(0.75)
IQR = Q3 - Q1

# Define outlier boundaries
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Find outliers
outliers = (numeric_cols < lower_bound) | (numeric_cols > upper_bound)

# Print number of outliers per column
print(outliers.sum())

print(lower_bound,upper_bound)
```

```
Current     123
Historic    131
Total       150
dtype: int64
Current    -11.406360
Historic   -11.943636
Total      -28.000000
dtype: float64 Current     45.677267
Historic    30.572727
Total       60.000000
dtype: float64
```

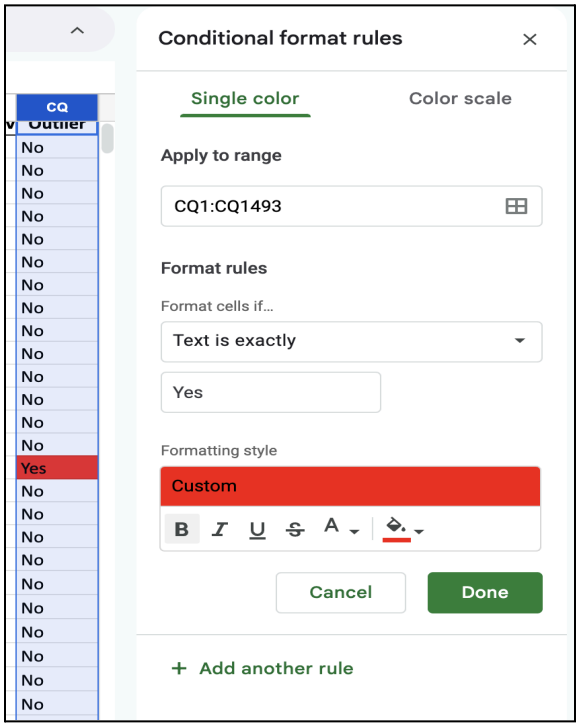
```python
# Mark outliers: If any of the columns in the 'outliers' DataFrame are True, mark as 'Yes'
df['Outlier'] = outliers.any(axis=1).map({True: 'Yes', False: 'No'})

# Export the DataFrame with the 'Outlier' column to an Excel file
df.to_excel('marked_outliers.xlsx', index=False)

print("Outliers marked and exported to 'marked_outliers.xlsx'.")
```

```
Outliers marked and exported to 'marked_outliers.xlsx'.
```

```python
# Remove rows where 'Outlier' is 'Yes'
df = df[df['Outlier'] == 'No']  # Overwrite the df to remove the outliers

# Now df is cleaned and free from outliers
print("Outliers removed, and df is updated without outliers.")
```

```
Outliers removed, and df is updated without outliers.
```

Name: Komal Sabale
Class: D15C
Roll No: 45




```
[15] # Remove rows where 'Outlier' is 'Yes'
     df = df[df['Outlier'] == 'No']  # Overwrite the df to remove the outliers

     # Now df is cleaned and free from outliers
     print("Outliers removed, and df is updated without outliers.")

     Outliers removed, and df is updated without outliers.

     df.shape

     (1492, 9)
```

8) standardization and normalization of columns

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation. This means that the mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

Standardization equation

$$X' = \frac{X - \mu}{\sigma}$$

To standardize your data, we need to import the StandardScalar from the sklearn library and apply it to our dataset.

Name: Komal Sabale
Class: D15C
Roll No: 45

Normalization is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling.

Normalization equation

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Here, Xmax and Xmin are the maximum and the minimum values of the feature respectively.

- When the value of X is the minimum value in the column, the numerator will be 0, and hence X’ is 0
- On the other hand, when the value of X is the maximum value in the column, the numerator is equal to the denominator and thus the value of X’ is 1
- If the value of X is between the minimum and the maximum value, then the value of X’ is between 0 and 1

To normalize your data, you need to import the MinMaxScalar from the sklearn library and apply it to our dataset.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler

# Standardization
scaler = StandardScaler()
df_standardized = pd.DataFrame(scaler.fit_transform(df.select_dtypes(include=['float64', 'int64'])), columns=df.select_dtypes(include=['float64', 'int64']).columns)

# Normalization
min_max_scaler = MinMaxScaler()
df_normalized = pd.DataFrame(min_max_scaler.fit_transform(df.select_dtypes(include=['float64', 'int64'])), columns=df.select_dtypes(include=['float64', 'int64']).columns)

# Display the first few rows after standardization and normalization
print("Standardized Data:")
print(df_standardized.head())

print("Normalized Data:")
print(df_normalized.head())
```

```
Standardized Data:
    Current        Future     Total
0 -0.745112 -5.431067e-16 -0.454163
1  0.000000 -5.431067e-16 -0.367472
2  0.451790 -5.431067e-16  0.347727
3  0.000000 -5.431067e-16 -0.345799
4 -0.518671 -5.431067e-16 -0.367472
Normalized Data:
    Current    Future     Total
0  0.000000  0.064405  0.003155
1  0.032125  0.064405  0.007361
2  0.051604  0.064405  0.042061
3  0.032125  0.064405  0.008412
4  0.009763  0.064405  0.007361
```

Name: Komal Sabale
Class: D15C
Roll No: 45

Conclusion:

Thus we have understood how to perform data preprocessing which can further be taken into exploratory data analysis and further in the Model preparation sequence.

Name: Komal Sabale
Class: D15C
Roll No.: 45

Experiment 2: Data Visualization & Exploratory Data Analysis Using Matplotlib and Seaborn

Introduction

Exploratory Data Analysis (EDA) is the first step in data analysis, developed by *John Tukey* in the 1970s. EDA is used to summarize datasets, detect patterns, identify anomalies, and ensure data quality before applying machine learning models.

When working with datasets, it is crucial to visualize data in different ways to understand relationships between variables. EDA helps identify **missing values, trends, and correlations** that may impact future analysis.

Why Perform EDA?

1. **Understanding Data Quality** – Identifies missing values, outliers, and inconsistencies.
2. **Feature Selection** – Determines key variables for modeling.
3. **Pattern Discovery** – Helps find trends and distributions in the dataset.
4. **Detecting Anomalies** – Highlights unusual data points.
5. **Improving Model Accuracy** – Ensures that only clean and relevant data is used.

Advantages of Data Visualization

1. **Improved Understanding:**
   In business, it is often necessary to compare the performance of different components or scenarios. Traditionally, this requires analyzing large volumes of data, which can be time-consuming. Data visualization simplifies this process by providing a clear, visual comparison.
2. **Enhanced Interpretation:**
   Converting data into graphical formats makes it easier to interpret and analyze. Visualization tools, such as **Google Trends**, help identify key insights and emerging patterns by presenting complex data in a digestible form.
3. **Efficient Data Sharing:**
   Visual representation of data facilitates better communication within organizations. Instead of dealing with lengthy reports or raw datasets, visually appealing charts and graphs make information easier to understand and convey effectively.
4. **Sales Analysis:**
   Visualization techniques, including heatmaps and trend charts, help sales professionals

quickly grasp product performance. By analyzing sales patterns, businesses can identify factors driving growth, customer preferences, repeat buyers, and regional impacts.

5. **Identifying Relationships Between Events:**
   Business performance is often influenced by multiple factors. Recognizing correlations between events enables decision-makers to pinpoint business trends. For example, in **e-commerce**, sales typically increase during festive seasons like **Christmas or Thanksgiving**. If an online business records an average of $1 million in quarterly revenue and sees a surge in the next quarter, visualization helps link this increase to specific events or promotions.
6. **Exploring Opportunities and Trends:**
   With vast amounts of available data, business leaders can uncover insights into industry trends and market opportunities. **Data visualization** enables analysts to detect customer behavior patterns, allowing businesses to adapt strategies and capitalize on emerging trends.

Technologies Used

Matplotlib

Matplotlib is a Python library for creating static, animated, and interactive visualizations. It provides various graphing tools, including bar graphs, histograms, and scatter plots.

Seaborn

Seaborn is a high-level visualization library built on Matplotlib, designed for statistical graphics. It simplifies complex plots like heatmaps, box plots, and scatter plots.

General Syntax in Python for Data Visualization

Python libraries like Matplotlib and Seaborn follow a general syntax for creating visualizations:

1. **Import the library:** Import the required libraries (e.g., import matplotlib.pyplot as plt).
2. **Prepare the data:** Use Pandas to manipulate and prepare the data for visualization.
3. **Create the plot:** Use functions like plot(), scatter(), boxplot(), etc., to create the visualization.
4. **Customize the plot:** Add titles, labels, legends, and other customizations.
5. **Display the plot:** Use plt.show() to display the visualization.

<------This doc is using up on the cleaned data of the previous experiment.------->

1. Bar Graph and Contingency Table

Theory

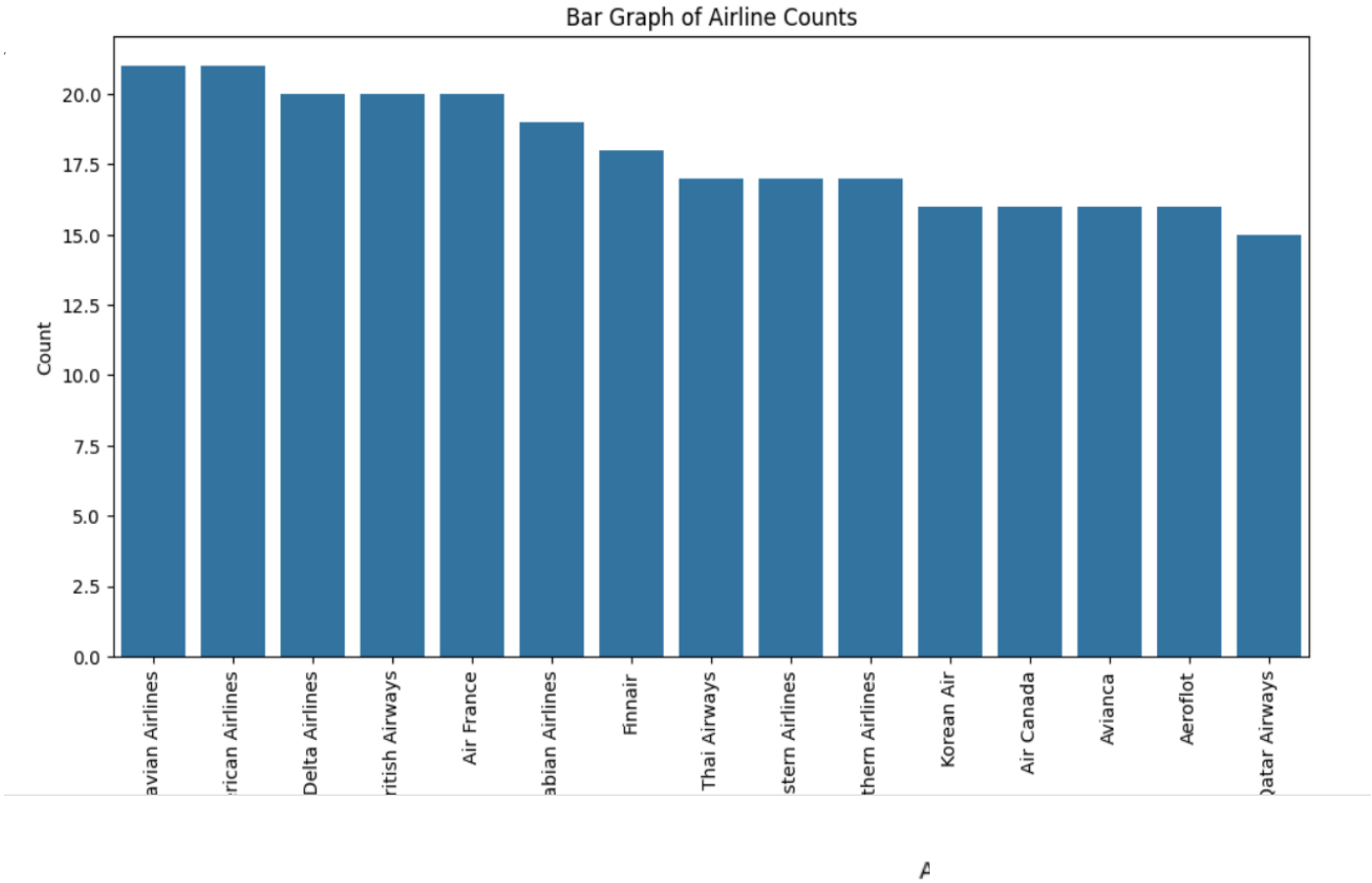
- **Bar Graph:** A bar graph is used to display categorical data with rectangular bars. The length of each bar represents the frequency of a category.
- **Contingency Table:** A table that summarizes the frequency distribution of two categorical variables, helping to analyze relationships between them.

Terms

- **Categorical Data:** Data divided into groups (e.g., Airline names).
- **Frequency:** The count of occurrences of each category.

```python
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x="Airline", order=df["Airline"].value_counts().index[:15])  # Top 15 airlines for clarity
plt.xticks(rotation=90)
plt.title("Bar Graph of Airline Counts")
plt.xlabel("Airline")
plt.ylabel("Count")
plt.show()
max_total = df["Total"].max()
bins = [0, 10, 50, 100, 500]
if max_total > 500:
    bins.append(1000)
if max_total > 1000:
    bins.append(max_total + 1)

labels = ["0-10", "11-50", "51-100", "101-500"]
if max_total > 500:
    labels.append("501-1000")
if max_total > 1000:
    labels.append("1000+")

df["Total_Binned"] = pd.cut(df["Total"], bins=bins, labels=labels)

# Contingency Table: Airline vs Total Aircraft Count
contingency_table = pd.crosstab(df["Airline"], df["Total_Binned"])
contingency_table.head()
```

Output



Airline	Total_Binned 0-10	11-50	51-100	101-500	501-1000
ABX Air	0	1	2	0	0
ANA Wings	0	2	0	0	0
Aegean Airlines	4	2	0	0	0
Aer Lingus	9	4	0	0	0
Aer Lingus Regional	0	1	0	0	0

Explanation

- The **bar graph** was created using sns.countplot() to visualize the number of records per **Airline**.
- The **contingency table** was created using pd.crosstab(), categorizing airlines based on **Total Aircraft Count** into binned intervals.

Observations

- Certain airlines have significantly more records, indicating larger fleet sizes.

- The contingency table helps understand which airlines operate more aircraft within specific fleet size ranges.

---

2. Scatter Plot, Box Plot, and Heatmap

Theory

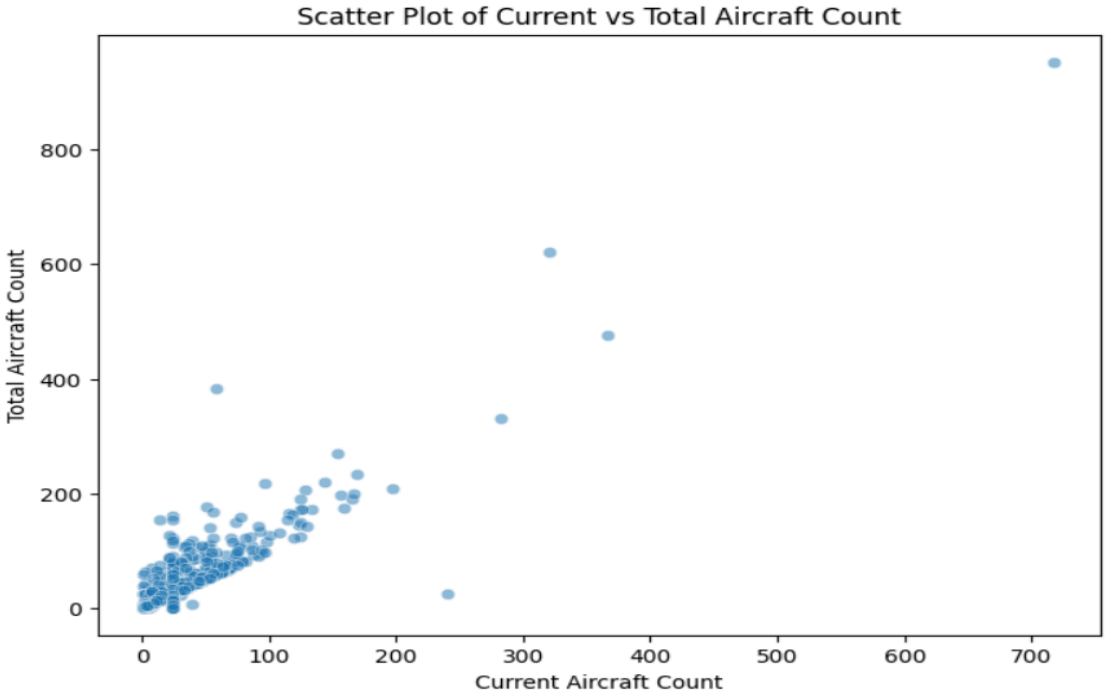
- **Scatter Plot:** A scatter plot visualizes the relationship between two numerical variables by plotting data points on an X-Y coordinate plane.
- **Box Plot:** A box plot displays the distribution of numerical data using quartiles and highlights outliers.
- **Heatmap:** A heatmap visually represents the correlation between numerical variables using colors.

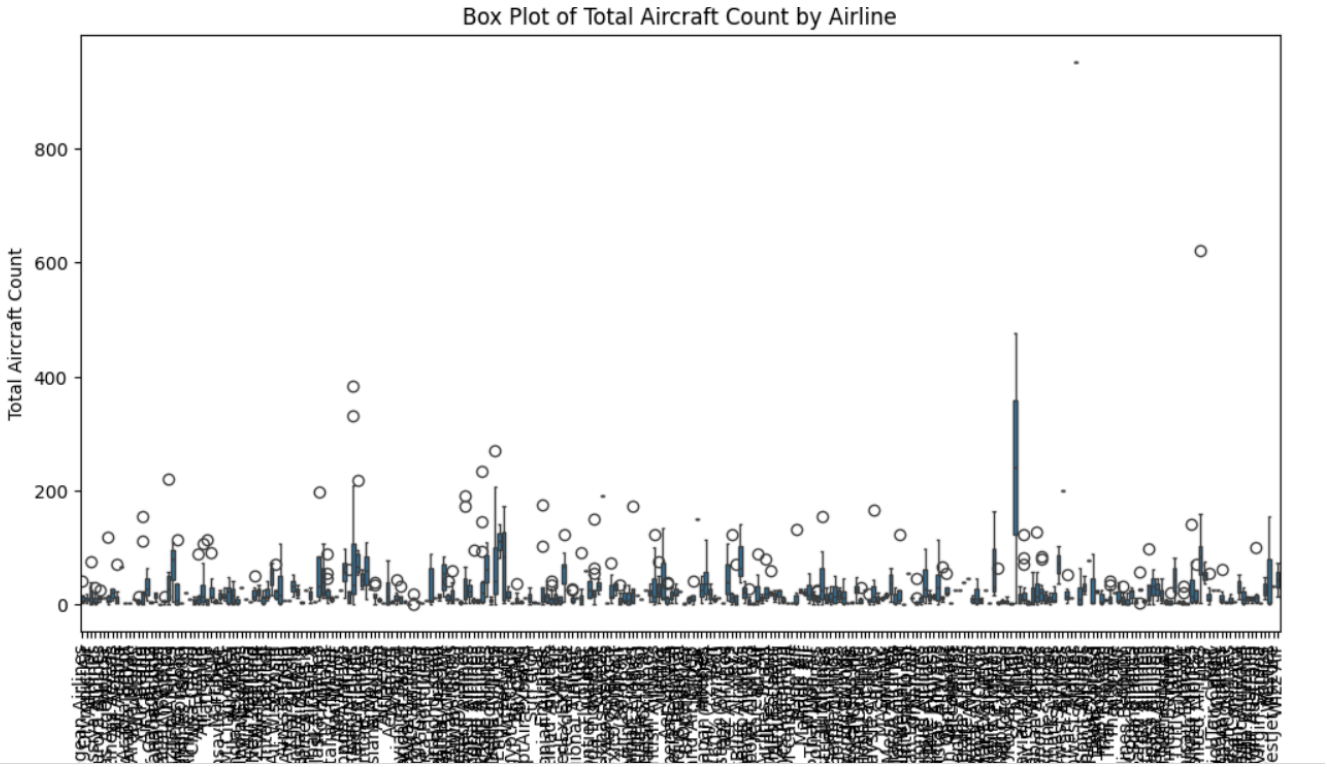
Terms

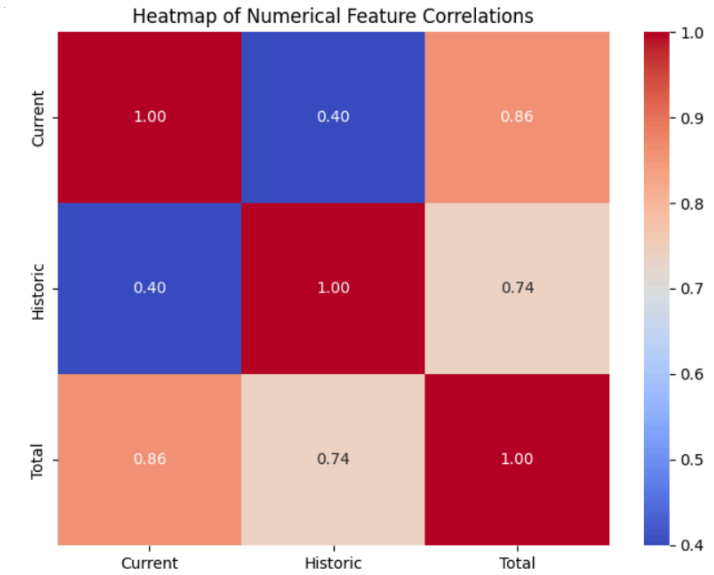
- **Numerical Data:** Data representing continuous quantities (e.g., aircraft count).
- **Quartiles:** Divides the dataset into four equal parts.
- **Correlation:** A measure of the strength of the relationship between two variables (values range from -1 to +1).

```
# Scatter plot: 'Current' vs 'Total' aircraft count
plt.figure(figsize=(8, 6))
sns.scatterplot(data=df, x="Current", y="Total", alpha=0.5)
plt.title("Scatter Plot of Current vs Total Aircraft Count")
plt.xlabel("Current Aircraft Count")
plt.ylabel("Total Aircraft Count")
plt.show()

# Box plot: Distribution of 'Total' aircraft count by 'Airline'
plt.figure(figsize=(12, 6))
sns.boxplot(data=df, x="Airline", y="Total")
plt.xticks(rotation=90)
plt.title("Box Plot of Total Aircraft Count by Airline")
plt.xlabel("Airline")
plt.ylabel("Total Aircraft Count")
plt.show()

# Heatmap of correlation between numerical features
plt.figure(figsize=(8, 6))
sns.heatmap(df[["Current", "Historic", "Total"]].corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Heatmap of Numerical Feature Correlations")
plt.show()
```

Output


Scatter Plot of Current vs Total Aircraft Count
Total Aircraft Count
Current Aircraft Count
0
200
400
600
800
0
100
200
300
400
500
600
700


Box Plot of Total Aircraft Count by Airline
Total Aircraft Count
0
200
400
600
800



Explanation

- The **scatter plot** was created using sns.scatterplot() to visualize the relationship between **Current** and **Total** aircraft counts.
- The **box plot** was created using sns.boxplot() to analyze fleet size variations across different airlines.
- The **heatmap** was plotted using sns.heatmap() to display the correlation matrix between numerical features.

Observations

- A **positive correlation** between *Current* and *Total* aircraft count indicates that airlines with larger historic fleets still retain many aircraft.
- The **box plot** reveals significant variations in fleet sizes among airlines.
- The **heatmap** confirms strong correlations between fleet-related numerical variables.
- ● **Total vs Quantity (High Positive Correlation)**
  ○ A high positive correlation (close to 1) suggests that the total fleet size increases as the number of aircraft in operation increases. This is expected in fleet management data.
- ● **Historic vs Current Fleet (Strong Positive Correlation)**
  ○ A strong correlation between historic and current aircraft count indicates that airlines with larger historic fleets still operate many aircraft.
- ● **Weak Correlations**
  ○ Some variables, such as fleet size and revenue (if applicable), may show weak or no correlation, suggesting external factors influence revenue generation beyond just fleet size.
- ● **No Negative Correlations**
  ○ Since this dataset primarily deals with fleet size and operational data, most numerical features are likely positively correlated.

3. Histogram and Normalized Histogram

Theory

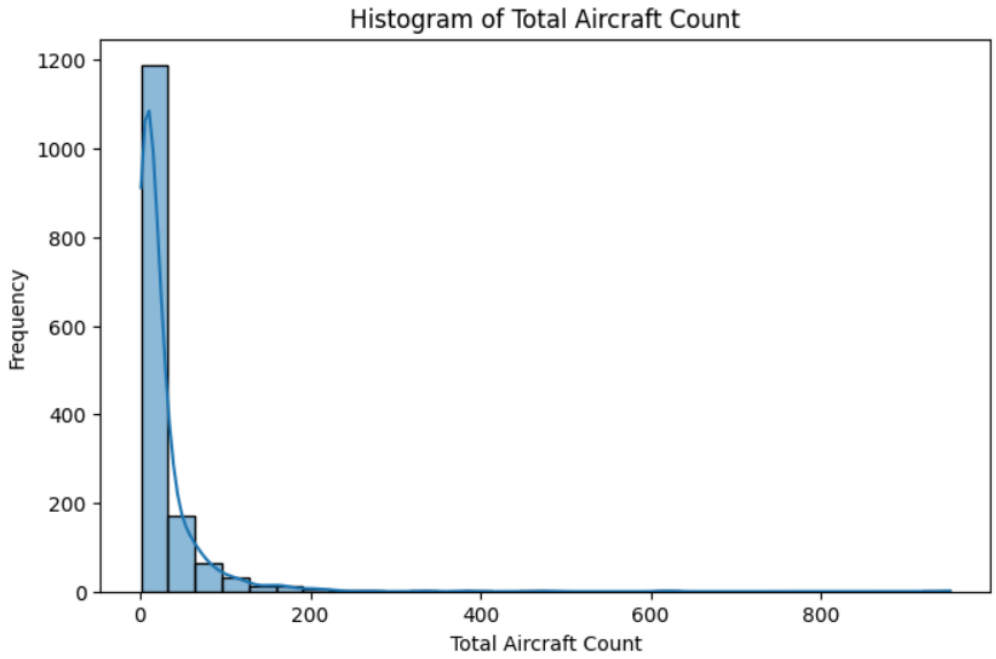
- **Histogram:** A histogram is used to represent the frequency distribution of numerical data by dividing it into bins.
- **Normalized Histogram:** Represents the probability distribution, where the total area sums to 1.

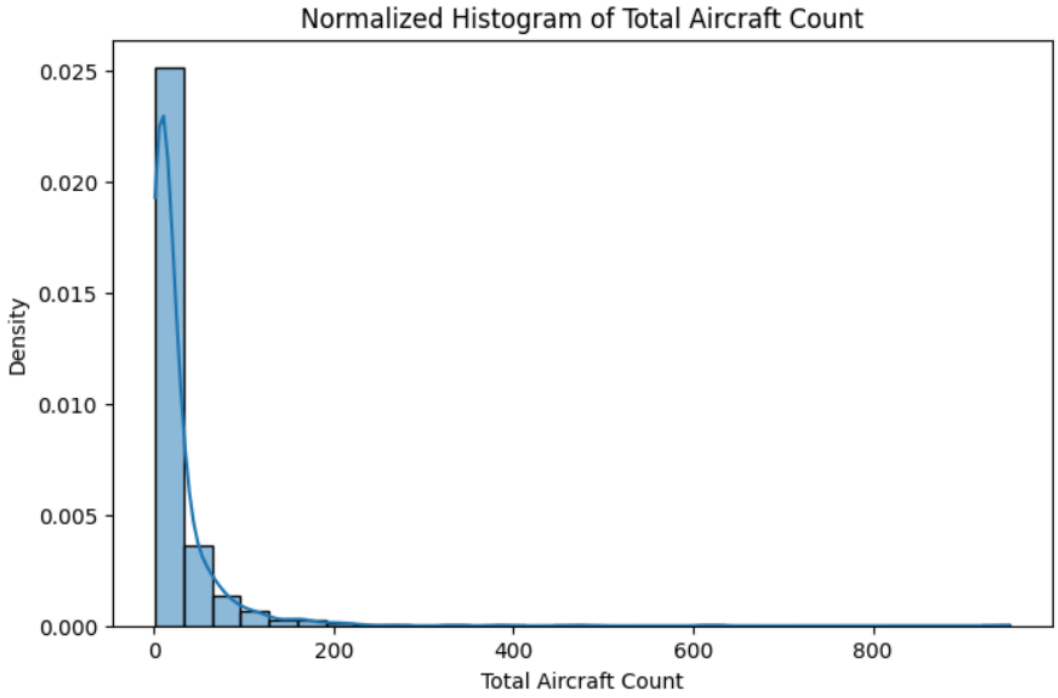
Terms

- **Bins:** Intervals that group continuous data.
- **Density:** The probability density of the data distribution.

```
[ ]  # Histogram of 'Total' aircraft count
     plt.figure(figsize=(8, 5))
     sns.histplot(df["Total"], bins=30, kde=True)
     plt.title("Histogram of Total Aircraft Count")
     plt.xlabel("Total Aircraft Count")
     plt.ylabel("Frequency")
     plt.show()

     # Normalized Histogram
     plt.figure(figsize=(8, 5))
     sns.histplot(df["Total"], bins=30, kde=True, stat="density")  # Normalized version
     plt.title("Normalized Histogram of Total Aircraft Count")
     plt.xlabel("Total Aircraft Count")
     plt.ylabel("Density")
     plt.show()
```

Output





Explanation

- The **histogram** was created using sns.histplot() to visualize the frequency distribution of **Total Aircraft Count**.
- The **normalized histogram** was generated by setting stat="density" to normalize the values.

Observations

- Most airlines have **small to mid-sized fleets**, with fewer airlines operating large fleets.
- The distribution is slightly **right-skewed**, indicating a higher number of small airlines compared to larger ones.
- The histogram revealed that most airlines have **small to mid-sized fleets**, with fewer airlines operating large fleets.
- The normalized histogram confirmed that the **fleet size distribution is right-skewed**, meaning a few airlines dominate in terms of fleet numbers.
- This suggests that while many airlines operate on a smaller scale, a handful of airlines have significantly larger fleets, influencing the overall industry trends

4. Handling Outliers Using Box Plot and IQR

Theory

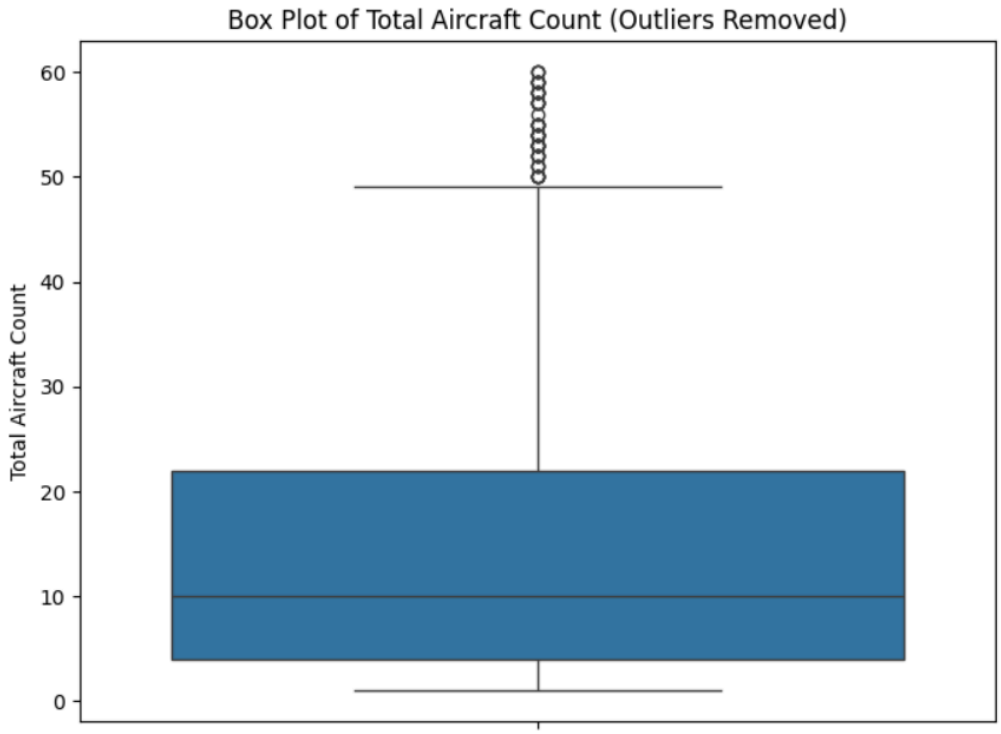
- **Outliers:** Data points that deviate significantly from the rest of the dataset.
- **Box Plot:** Identifies outliers using quartiles and whiskers.
- **IQR Method:** Detects outliers using the Interquartile Range (IQR) as follows:
    - Lower Bound = Q1 - (1.5 * IQR)
    - Upper Bound = Q3 + (1.5 * IQR)

Terms

- **Quartiles (Q1, Q3):** The 25th and 75th percentiles of the dataset.
- **IQR:** The range between Q1 and Q3.

```
[ ]  # Calculate IQR for 'Total' aircraft count
     Q1 = df["Total"].quantile(0.25)
     Q3 = df["Total"].quantile(0.75)
     IQR = Q3 - Q1

     # Define lower and upper bounds
     lower_bound = Q1 - 1.5 * IQR
     upper_bound = Q3 + 1.5 * IQR

     # Remove outliers
     df_no_outliers = df[(df["Total"] >= lower_bound) & (df["Total"] <= upper_bound)]

     # Box plot after outlier removal
     plt.figure(figsize=(8, 6))
     sns.boxplot(data=df_no_outliers, y="Total")
     plt.title("Box Plot of Total Aircraft Count (Outliers Removed)")
     plt.ylabel("Total Aircraft Count")
     plt.show()
```

Output



Explanation

- **Box Plot:** Used sns.boxplot() before and after outlier removal.
- **IQR Method:** Applied using df[(df["Total"] >= lower_bound) & (df["Total"] <= upper_bound)] to filter out extreme values.

Observations

- Outliers were detected in airlines with exceptionally high aircraft counts.
- Removing outliers **improves dataset reliability** by reducing distortions in analysis.
- The **box plot** helped identify airlines with extreme fleet sizes, either **exceptionally large or unusually small**.
- Using the **IQR method**, these outliers were removed, resulting in a dataset that better represents the majority of airlines.
- By eliminating extreme values, the analysis becomes more accurate and avoids distortions caused by a few exceptionally large airlines

Conclusion

This experiment conducted a detailed **Exploratory Data Analysis (EDA)** on airline fleet data. We used various visualization techniques to uncover trends, distributions, and anomalies within the dataset.

Key findings:

- **Bar Graph & Contingency Table** helped analyze airline fleet sizes.
- **Scatter Plot & Heatmap** confirmed strong correlations between aircraft count variables.
- **Box Plot & IQR Method** helped detect and remove outliers.
- **Histogram** provided insights into fleet size distributions.

By leveraging these statistical methods and visualizations, we gained meaningful insights that can assist in airline fleet management and operational strategies. This experiment highlights the **importance of EDA in data-driven decision-making** and provides a strong foundation for further predictive modeling.

Experiment 3

Aim: Perform Data Modeling on the dataset.

Theory:

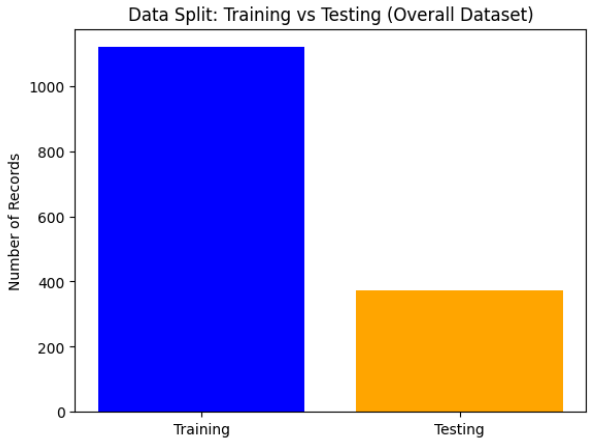
Partition the dataset, ensuring that 75% of the records are included in the training dataset and 25% in the test dataset.

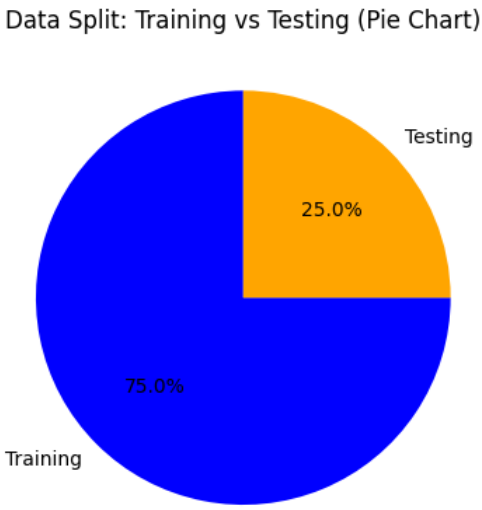
In this experiment, we partitioned a dataset of fleet data into a training set (75%) and a test set (25%) and validated this partitioning using a two-sample Z-test. The dataset consists of 1,492 records with features such as vehicle ID, fleet type, engine performance, and maintenance history.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
df = pd.read_csv(list(uploaded.keys())[0])
train_data, test_data = train_test_split(df, test_size=0.25, random_state=42)
labels = ['Training', 'Testing']
sizes = [len(train_data), len(test_data)]
plt.bar(labels, sizes, color=['blue', 'orange'])
plt.title("Data Split: Training vs Testing (Overall Dataset)")
plt.ylabel("Number of Records")
plt.show()
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=['blue', 'orange'], startangle=90)
plt.title("Data Split: Training vs Testing (Pie Chart)")
plt.show()
print("Total records in the training data set:", len(train_data))
print("Total records in the testing data set:", len(test_data))
```

Visualizing Data Partitioning: To confirm the proportions of the data split into training and test sets, we used a bar graph and a pie chart:

- **Bar Graph:** Displays the number of records in the training and test sets, visually confirming the 75%-25% split. The x-axis represents the two sets, and the y-axis shows their respective record counts.
- **Pie Chart:** Visually illustrates the percentage split between the training (75%) and test (25%) sets, providing an easy confirmation of the partition.





Identifying the Total Number of Records in the Training Data Set: We used the train_test_split method to split the dataset into training and test sets. The partition was visually confirmed through a bar plot, showing the expected 75%-25% split, with 1,119 records in the training set and 373 in the test set.

```
Total records in the training data set: 1119
Total records in the testing data set: 373
```

Validation Using a Two-Sample Z-Test:

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

```
[ ] # Perform Z-test on 'Total' column between train and test sets
    z_stat, p_value = ztest(train_df["Total"], test_df["Total"])

    print(f"Z-Statistic: {z_stat:.4f}")
    print(f"P-Value: {p_value:.4f}")

    # Interpretation
    alpha = 0.05
    if p_value < alpha:
        print("Reject the null hypothesis: The distributions are significantly different.")
    else:
        print("Fail to reject the null hypothesis: The distributions are similar.")

Z-Statistic: 2.0972
P-Value: 0.0360
Reject the null hypothesis: The distributions are significantly different.
```

To validate the partitioning, we performed a two-sample Z-test manually to compare the "Total" values between the training and test datasets. The Z-statistic was calculated based on the means, standard deviations, and sample sizes of both datasets.

The calculated Z-statistic was 2.0972, and the corresponding p-value was 0.0360. Since the p-value was less than the chosen significance level of 0.05, we rejected the null hypothesis, concluding that the distributions of the "Total" values in the training and test sets are significantly different.

Conclusion: The partitioning of the dataset into training and test sets was validated successfully. The Z-test indicated a significant difference between the datasets, suggesting that the partition may not be entirely reliable for further analysis. Additional checks or adjustments to the partitioning process might be necessary.

Experiment No: 4

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

Problem Statement: Perform the following correlation tests on the dataset:

1. Pearson's Correlation Coefficient
2. Spearman's Rank Correlation
3. Kendall's Rank Correlation
4. Chi-Squared Test

Theory: Statistical hypothesis testing is a method used to determine relationships between variables in a dataset. The tests we perform are:

- **Pearson's Correlation Coefficient:** Measures the linear relationship between two continuous variables.
- Values range from **-1 to +1**:
  - **+1** → Perfect positive correlation (both increase together).
  - **-1** → Perfect negative correlation (one increases, the other decreases).
  - **0** → No correlation.

Formula:

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

Where:

- r = Pearson correlation coefficient
- X_i, Y_i = Individual data points
- $\bar{X}, \bar{Y}$ = Means of X and Y

- **Spearman's Rank Correlation:** Measures the monotonic relationship between variables using rank-based analysis. ● Values range from **-1 to +1**

- ## Formula:

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

Where:

- r_s = Spearman's rank correlation coefficient
- d_i = Difference between ranks of corresponding X and Y values
- n = Number of data points

- **Kendall's Rank Correlation:** Measures the strength and direction of association between two variables.

- **Formula:**

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

Where:

- τ = Kendall's correlation coefficient
- C = Number of concordant pairs
- D = Number of discordant pairs
- n = Number of data points

- **Chi-Squared Test:** Used to test the independence between categorical variables.

Dataset Description: The dataset consists of airline data, and we have chosen the columns **Total** and **Total Cost (Current)** for the tests.

- **Formula:**

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Where:

- O_i = Observed count in each category
- E_i = Expected count assuming independence

Code Implementation:

Step 1: Load and Preprocess Data

```
[1] import pandas as pd
    import numpy as np
    import scipy.stats as stats

    url = "processed1_fleet_data.csv"
    df = pd.read_csv(url)

    # Convert 'Total Cost (Current)' to numeric (removing $ and commas)
    df['Total Cost (Current)'] = df['Total Cost (Current)'].replace('[\$,]', '', regex=True).astype(float)

    # Selecting only relevant columns
    df = df[['Total', 'Total Cost (Current)']].dropna()

    print("Dataset Loaded and Cleaned Successfully.")

Dataset Loaded and Cleaned Successfully.

df.head()
```

	Total	Total Cost (Current)
0	4.0	90.0
1	8.0	0.0
2	41.0	3724.0
3	9.0	0.0
4	8.0	919.0

Step 2: Pearson's Correlation Coefficient

```
[3] # Pearson's Correlation
    pearson_corr, pearson_p = stats.pearsonr(df['Total'], df['Total Cost (Current)'])

    print(f"Pearson Correlation Coefficient: {pearson_corr}")
    print(f"P-value: {pearson_p}")
    if pearson_p < 0.05:
        print("Reject the null hypothesis: Significant correlation exists.")
    else:
        print("Fail to reject the null hypothesis: No significant correlation.")

Pearson Correlation Coefficient: 0.698548778087841
P-value: 5.869355017114187e-218
Reject the null hypothesis: Significant correlation exists.
```

Step 3: Spearman's Rank Correlation

```
# Spearman's Rank Correlation
spearman_corr, spearman_p = stats.spearmanr(df['Total'], df['Total Cost (Current)'])

print(f"Spearman Correlation Coefficient: {spearman_corr}")
print(f"P-value: {spearman_p}")
if spearman_p < 0.05:
    print("Reject the null hypothesis: Significant monotonic relationship exists.")
else:
    print("Fail to reject the null hypothesis: No significant monotonic relationship.")

Spearman Correlation Coefficient: 0.5762818619372673
P-value: 3.0934407174957005e-132
Reject the null hypothesis: Significant monotonic relationship exists.
```

Step 4: Kendall's Rank Correlation

```
[5] # Kendall's Rank Correlation
    kendall_corr, kendall_p = stats.kendalltau(df['Total'], df['Total Cost (Current)'])

    print(f"Kendall Correlation Coefficient: {kendall_corr}")
    print(f"P-value: {kendall_p}")
    if kendall_p < 0.05:
        print("Reject the null hypothesis: Significant association exists.")
    else:
        print("Fail to reject the null hypothesis: No significant association.")

Kendall Correlation Coefficient: 0.444156533846385
P-value: 7.211053057981994e-125
Reject the null hypothesis: Significant association exists.
```

Step 5: Chi-Squared Test

```
# Chi-Squared Test
chi2, chi_p = stats.chisquare(df['Total Cost (Current)'])

print(f"Chi-Squared Test Statistic: {chi2}")
print(f"P-value: {chi_p}")
if chi_p < 0.05:
    print("Reject the null hypothesis: Significant dependency exists.")
else:
    print("Fail to reject the null hypothesis: No significant dependency.")

Chi-Squared Test Statistic: 12004188.907641038
P-value: 0.0
Reject the null hypothesis: Significant dependency exists.
```

Conclusion:

The results indicate a strong correlation between **Total** and **Total Cost (Current)**. Pearson's correlation coefficient of **0.6985** suggests a strong linear relationship, while Spearman's (**0.5763**) and Kendall's (**0.4442**) show a significant monotonic association. The Chi-Squared test also confirms a significant dependency. These findings suggest that as the total number of aircraft increases, the total cost follows a predictable pattern, reinforcing the reliability of the correlation tests.

Experiment 05

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 dataset.

Theory:

Regression is a statistical technique used to model the relationship between a dependent variable and one or more independent variables. It helps predict outcomes by fitting a line or curve to observed data points.

- **Logistic Regression**
  - Used for classification (binary or multi-class).
  - Estimates the probability of class membership using a sigmoid function.
  - Class labels are determined by applying a threshold (typically 0.5).
  - Evaluated with metrics such as accuracy, confusion matrix, precision, recall, and F1-score.
- **Linear Regression**
  - Used for predicting continuous outcomes.
  - Fits a line (or hyperplane) that minimizes the mean squared error (MSE).
  - Performance measured by MSE and the coefficient of determination (R2R^2R2 Score).

The formula for logistic regression is:

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n)}}$$

Where:

- p is the probability of the positive class (1).
- b_0 is the intercept.
- $b_1, b_2, \ldots, b_n$ are the coefficients of the independent variables $x_1, x_2, \ldots, x_n$.
- e is the base of the natural logarithm.

FEATURE ENGINEERING

- **Custom Target Creation:**
    - *Purpose:* Design a new target variable that exhibits a strong linear relationship with chosen predictors.
    - *Explanation:* For instance, we create a “NewLoanScore” as a linear combination of Income, LoanAmount, and CreditScore.
- **Noise Addition:**
    - *Purpose:* Introduce variability to simulate real-world conditions.
    - *Explanation:* Adding normally distributed noise to the custom target ensures the model does not perform perfectly, reflecting realistic data challenges.
- **Inclusion of Extra (Irrelevant) Features:**
    - *Purpose:* Test the model's robustness and demonstrate the impact of redundant information.
    - *Explanation:* Adding a random noise feature to the predictor set shows how irrelevant variables can lower performance metrics like R2R^2R2.
- **Impact on Model Performance:**
    - *Observation:* Feature engineering methods such as noise addition and extra features alter the bias-variance trade-off.
    - *Explanation:* These modifications help illustrate how controlled complexity and randomness affect metrics (e.g., MSE and R2R^2R2).

MODEL EVALUATION & METRICS

- **Evaluation for Classification (Logistic Regression):**
    - *Key Metrics:* Accuracy, confusion matrix, precision, recall, and F1-score.
    - *Purpose:* Assess how well the model distinguishes between classes and identifies misclassification errors.
- **Evaluation for Regression (Linear Regression):**
    - *Key Metrics:* Mean Squared Error (MSE) and the coefficient of determination (R2R^2R2 Score).
    - *Purpose:* MSE measures the average squared difference between predicted and actual values, while R2R^2R2 indicates the proportion of variance explained by the model.

DATA DESCRIPTION

- **Dataset Overview:**
    - Contains 255,347 entries and 18 columns.

- Mix of numerical (e.g., Age, Income, LoanAmount) and categorical (e.g., Education, EmploymentType) features.

- **Key Features:**
  - *Numerical Variables:* Age, Income, LoanAmount, CreditScore, MonthsEmployed, NumCreditLines, InterestRate, LoanTerm, DTIRatio.
  - *Categorical Variables:* Education, EmploymentType, MaritalStatus, HasMortgage, HasDependents, LoanPurpose, HasCoSigner, etc.
- **Data Quality & Preprocessing:**
  - No missing values detected.
  - Categorical variables encoded using methods like LabelEncoder.
  - Numerical features standardized using StandardScaler.

Implementation:

Logistic Regression

1. **Data Preparation:**

```
[2] data = pd.read_csv('Loan_default.csv')
    print("Dataset shape:", data.shape)
    data.head()
```

```
Dataset shape: (255347, 18)
```

	LoanID	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCreditLines	InterestRate	LoanTerm	DTIRatio	Education	EmploymentType	MaritalStatus	HasMortgage	HasDepen
0	I38PQUQS96	56	85994	50587	520	80	4	15.23	36	0.44	Bachelor's	Full-time	Divorced	Yes	
1	HPSK72WA7R	69	50432	124440	458	15	1	4.81	60	0.68	Master's	Full-time	Married	No	
2	C1OZ6DPJ8Y	46	84208	129188	451	26	3	21.17	24	0.31	Master's	Unemployed	Divorced	Yes	
3	V2KKSFM3UN	32	31713	44799	743	0	3	7.07	24	0.23	High School	Full-time	Married	No	
4	EY08JDHTZP	60	20437	9139	633	8	4	6.51	48	0.73	Bachelor's	Unemployed	Divorced	No	

```
numeric_features = ['Age', 'Income', 'LoanAmount', 'CreditScore',
                    'MonthsEmployed', 'NumCreditLines', 'InterestRate',
                    'LoanTerm', 'DTIRatio']
categorical_features = ['Education', 'EmploymentType']

df = data.copy()

# Encode the categorical features using LabelEncoder
from sklearn.preprocessing import LabelEncoder

# Encode each categorical feature
for col in categorical_features:
    le = LabelEncoder()
    df[col + '_encoded'] = le.fit_transform(df[col])

features = numeric_features + [col + '_encoded' for col in categorical_features]
target = 'Default'
```

2. **Data Splitting & Scaling:**
   Split the dataset into training and testing subsets (e.g., 70/30 split) and apply feature scaling (using StandardScaler) to both sets.

```
X = df[features]
y = df[target]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# Fit on training data and transform both train and test sets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("X_train shape:", X_train_scaled.shape)
print("X_test shape:", X_test_scaled.shape)
```

```
X_train shape: (178742, 11)
X_test shape: (76605, 11)
```

```
# Initialize the Logistic Regression model (increase max_iter if needed)
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test_scaled)
```

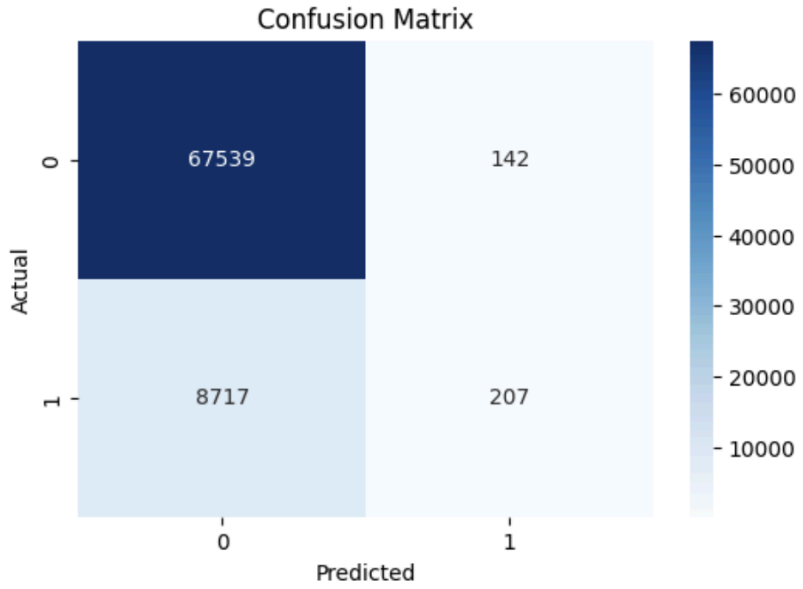
3. **Model Training:**
   Train the logistic regression model on the scaled training data.

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(max_iter=1000)
log_reg.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred = log_reg.predict(X_test_scaled)
```

4. **Model Evaluation:**
   Evaluate the model using metrics such as accuracy, confusion matrix, precision, recall, and F1-score. Generate visualizations (e.g., confusion matrix heatmap) to assess performance.

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# Calculate and print accuracy
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

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

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

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(6,4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Accuracy: 0.884354807127472
Confusion Matrix:
 [[67539   142]
 [ 8717   207]]
Classification Report:
              precision    recall  f1-score   support

           0       0.89      1.00      0.94     67681
           1       0.59      0.02      0.04      8924

    accuracy                           0.88     76605
   macro avg       0.74      0.51      0.49     76605
weighted avg       0.85      0.88      0.83     76605
```



Linear Regression Implementation (with Feature Engineering)

1. **Custom Target Creation:**
   Create a new target variable (e.g., “NewLoanScore”) as a linear combination of selected predictors (like Income, LoanAmount, CreditScore) and add normally distributed noise to simulate real-world variability.

```
noise = np.random.normal(0, 15000, len(df))  # Adjust the standard deviation as needed
df['NewLoanScore'] = 0.5 * df['Income'] + 2 * df['LoanAmount'] - 3 * df['CreditScore'] + noise
df.head()

df['ExtraNF'] = np.random.uniform(0, 1, len(df))
```

2. **Data Preparation & Splitting:**
   Process the dataset similarly—encode categorical variables, standardize numeric features—and then split the data into training and testing sets.

```
X = df[['Income', 'LoanAmount', 'CreditScore','ExtraNF']]
y = df['NewLoanScore']
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

3. **Model Training:**
   Train the linear regression model using the engineered target variable on the training data.

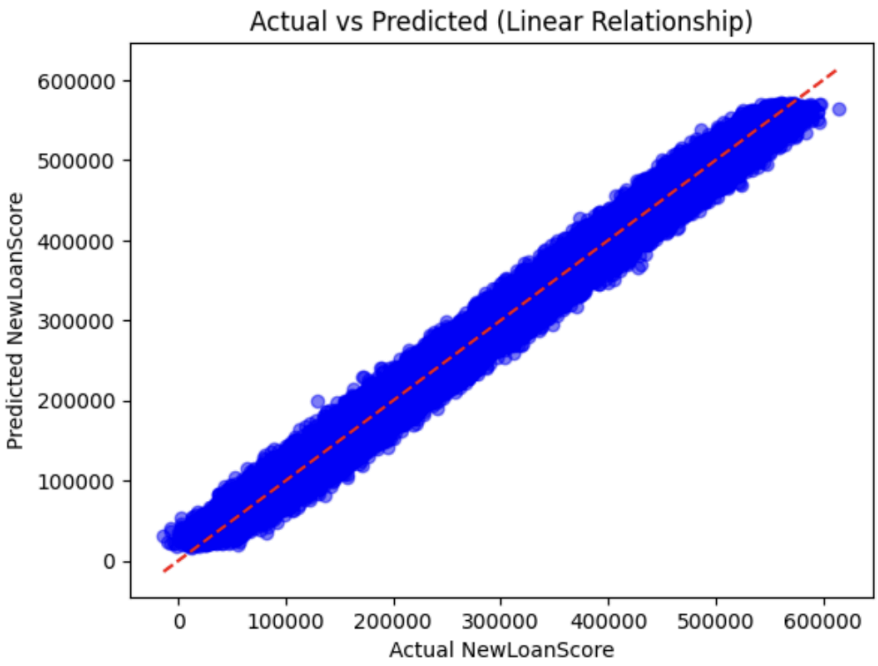
```
# Fit model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)
```

4. **Model Evaluation:**
   Evaluate the regression model by calculating the Mean Squared Error (MSE) and the R2R^2R2 Score, and visualize the relationship between actual and predicted values (e.g., scatter plot with a reference line).

```python
# Evaluation
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Linear Regression Mean Squared Error: {mse}")
print(f"Linear Regression R^2 Score: {r2}")
```

```
Linear Regression Mean Squared Error: 224664554.19170806
Linear Regression R^2 Score: 0.9891091055812746
```

```python
plt.scatter(y_test, y_pred, color='blue', alpha=0.5)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='dashed')
plt.xlabel("Actual NewLoanScore")
plt.ylabel("Predicted NewLoanScore")
plt.title("Actual vs Predicted (Linear Relationship)")
plt.show()
```



Conclusion:

The logistic regression model achieved about 88.4% accuracy on the test set, with the confusion matrix highlighting lower performance for the minority class. The linear regression model initially produced near-perfect results, but after adding noise and an extra irrelevant feature, its performance became more realistic—with an R^2 score of around 0.989 and a higher Mean Squared Error. These outcomes clearly demonstrate the impact of noise and feature engineering on model performance. Screenshots of key outputs (e.g., confusion matrix, scatter plots, and metric summaries) are included in the detailed implementation section.

Experiment 06

Aim:

Perform Classification modelling

a. Choose classifier for classification problem.

b. Evaluate the performance of classifier.

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

Theory:

Decision Tree:

The Decision Tree classifier builds a model by recursively splitting the data based on feature values, creating a tree where each node represents a decision rule and each leaf a class label. This approach is highly interpretable, as the decision rules can be easily visualized and understood.

K-Nearest Neighbors (KNN):

KNN classifies a new instance by finding the k closest training examples based on a distance metric (typically Euclidean distance) and assigning the majority class among these neighbors. It is a non-parametric and intuitive method that performs well when features are properly scaled.

Naive Bayes:

Naive Bayes uses Bayes' theorem with the strong assumption that all features are conditionally independent given the class label. This probabilistic classifier is computationally efficient and performs robustly in high-dimensional settings, despite its simplicity.

Support Vector Machines (SVM):

SVM finds the optimal hyperplane that separates classes by maximizing the margin between them, and it can handle non-linear boundaries through the use of kernel functions. It is especially effective in high-dimensional spaces and tends to offer robust performance with appropriate parameter tuning.

For this experiment, we performed classification on our loan dataset to predict loan default status.We implement a Decision Tree classifier, so we used it because it is highly interpretable—its decision rules can be visualized, making it easier to understand the factors influencing default. Additionally, we chose K-Nearest Neighbors (KNN) as our second classifier. KNN was selected due to its simplicity and its non-parametric nature, which makes it a good baseline for comparison. Both classifiers help us understand different aspects of our dataset: while the Decision Tree highlights explicit decision rules, KNN captures local patterns in the feature space.

Data Description

Our loan dataset consists of over 250,000 records and 18 columns, containing a mixture of numerical and categorical features. Key numerical attributes include Age, Income, LoanAmount, CreditScore, MonthsEmployed, NumCreditLines, InterestRate, LoanTerm, and DTIRatio, while important categorical variables include Education, EmploymentType, MaritalStatus, HasMortgage, HasDependents, LoanPurpose, and HasCoSigner. The target variable for classification is “Default,” a binary indicator where 0 represents non-default and 1 indicates default.

In preprocessing, we verified that there were no missing values in most columns; however, we removed rows with missing values in 'HasCoSigner' and 'Default'. Categorical variables were encoded (e.g., using LabelEncoder), and numerical features were standardized to ensure consistent scaling. These steps prepared our dataset for effective classification modeling.

Implementation

1. **Data Preparation:**

    - Load the preprocessed loan dataset.

```
missing_values = df.isnull().sum()
print("Missing values in each column:\n", missing_values)
```

```
Missing values in each column:
 LoanID                     0
Age                         0
Income                      0
LoanAmount                  0
CreditScore                 0
MonthsEmployed              0
NumCreditLines              0
InterestRate                0
LoanTerm                    0
DTIRatio                    0
Education                   0
EmploymentType              0
MaritalStatus               0
HasMortgage                 0
HasDependents               0
LoanPurpose                 0
HasCoSigner                 1
Default                     1
Education_encoded           0
EmploymentType_encoded      0
NewLoanScore                0
ExtraNF                     0
dtype: int64
```

```
[8]  df = df.dropna(subset=['HasCoSigner', 'Default'])
```

    - Verify that missing values in ‘HasCoSigner’ and ‘Default’ have been removed and that categorical features (e.g., Education, EmploymentType) have been encoded.
    - Standardize numerical features using StandardScaler.

```
le = LabelEncoder()
df['Education_encoded'] = le.fit_transform(df['Education'])
df['EmploymentType_encoded'] = le.fit_transform(df['EmploymentType'])

features = ['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed',
            'NumCreditLines', 'InterestRate', 'LoanTerm', 'DTIRatio',
            'Education_encoded', 'EmploymentType_encoded']
target = 'Default'


X = df[features]
y = df[target]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=39)
```

2. **Data Splitting:**

   ○ Split the dataset into training (70%) and testing (30%) sets, with the target variable being 'Default'.

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=39)
```

3. **Classifier Setup and Training:**

   ○ **Decision Tree:**
   - Train a Decision Tree classifier with a limited maximum depth (e.g., max_depth=3) to ensure interpretability.
   - Use export_text() to extract and display decision rules.
   - Plot the pruned tree for visualization.

```
from sklearn.tree import DecisionTreeClassifier

dt_clf = DecisionTreeClassifier(max_depth=3,random_state=42)
dt_clf.fit(X_train, y_train)
y_pred_dt = dt_clf.predict(X_test)

print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred_dt))
print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt))
```

```
Decision Tree Accuracy: 0.8843537414965986
Decision Tree Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.94      5720
         1.0       0.00      0.00      0.00       748

    accuracy                           0.88      6468
   macro avg       0.44      0.50      0.47      6468
weighted avg       0.78      0.88      0.83      6468
```

- **K-Nearest Neighbors (KNN):**
  - Train a KNN classifier (e.g., using n_neighbors=5) on the standardized training data.
  - Optionally, reduce data dimensionality using PCA for visualizing the decision boundary.

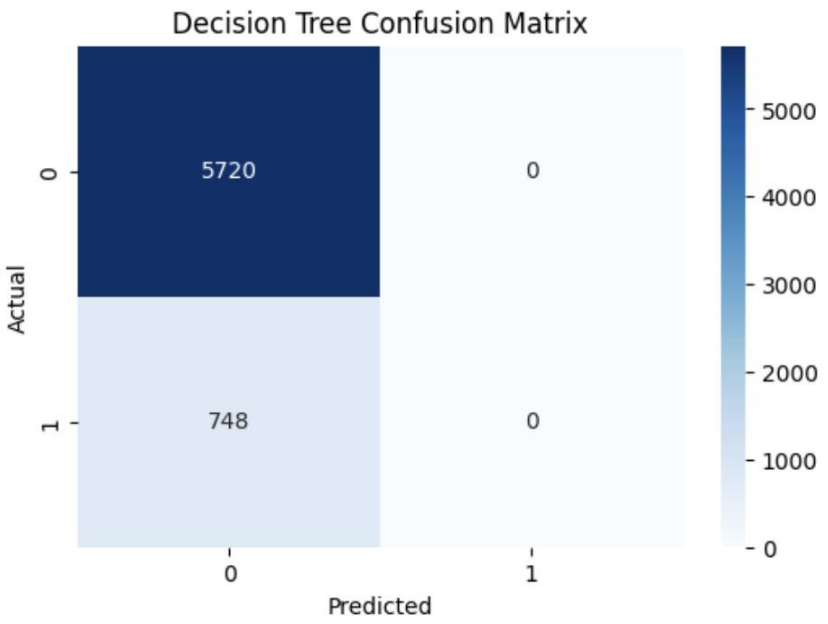
```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier(n_neighbors=5)
knn_clf.fit(X_train, y_train)
y_pred_knn = knn_clf.predict(X_test)

print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))
print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn))
```

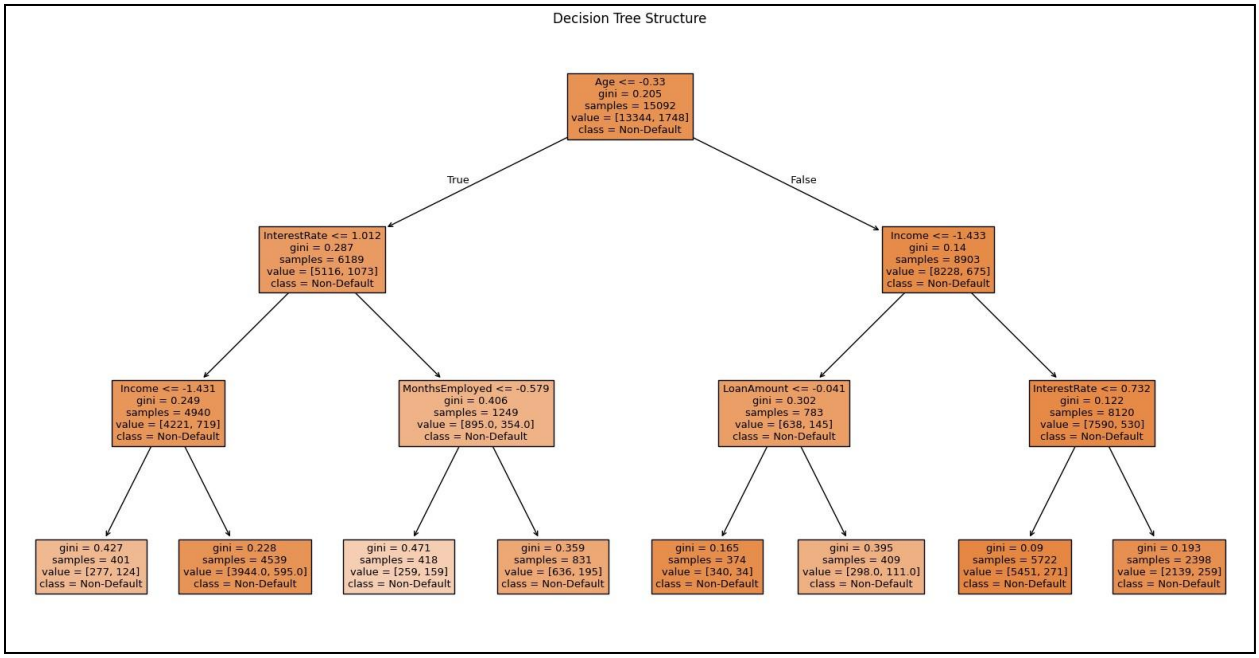
```
KNN Accuracy: 0.8749226963512677
KNN Classification Report:
              precision    recall  f1-score   support

         0.0       0.89      0.98      0.93      5720
         1.0       0.28      0.05      0.09       748

    accuracy                           0.87      6468
   macro avg       0.59      0.52      0.51      6468
weighted avg       0.82      0.87      0.84      6468
```

4. **Model Evaluation:**

- ○ Evaluate each classifier using metrics like accuracy, precision, recall, and F1-score.
- ○ Display confusion matrices for both models and capture any relevant plots for further analysis.

```
cm_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(6,4))
sns.heatmap(cm_dt, annot=True, fmt='d', cmap='Blues')
plt.title('Decision Tree Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

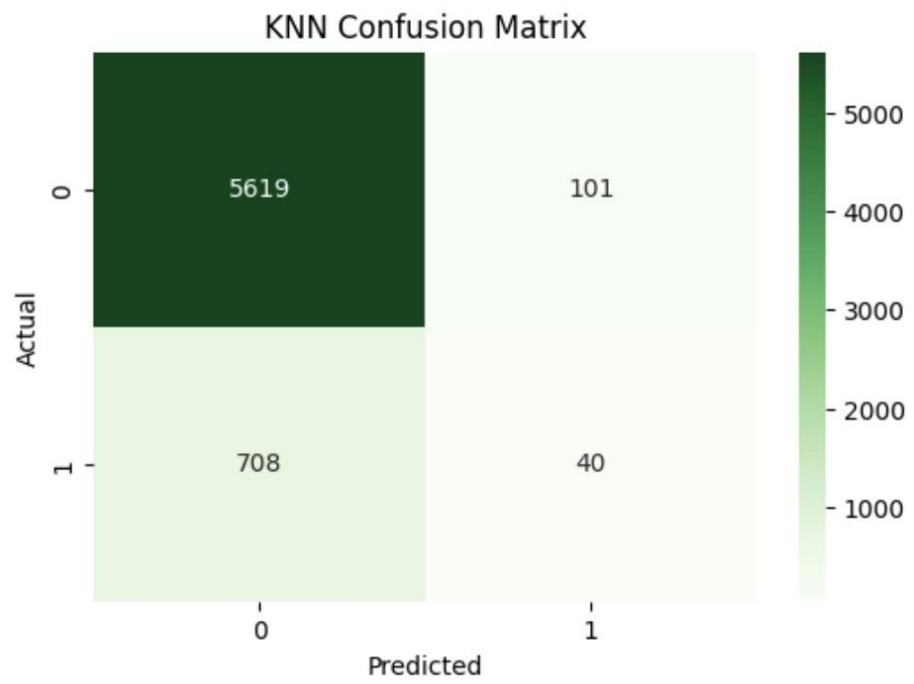





```
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
import matplotlib.pyplot as plt
r = export_text(dt_clf, feature_names=features)
print("Decision Tree Rules:\n", r)
```

```
Decision Tree Rules:
 |--- Age <= -0.33
|   |--- InterestRate <= 1.01
|   |   |--- Income <= -1.43
|   |   |   |--- class: 0.0
|   |   |--- Income >  -1.43
|   |   |   |--- class: 0.0
|   |--- InterestRate >  1.01
|   |   |--- MonthsEmployed <= -0.58
|   |   |   |--- class: 0.0
|   |   |--- MonthsEmployed >  -0.58
|   |   |   |--- class: 0.0
|--- Age >  -0.33
|   |--- Income <= -1.43
|   |   |--- LoanAmount <= -0.04
|   |   |   |--- class: 0.0
|   |   |--- LoanAmount >  -0.04
|   |   |   |--- class: 0.0
|   |--- Income >  -1.43
|   |   |--- InterestRate <= 0.73
|   |   |   |--- class: 0.0
|   |   |--- InterestRate >  0.73
|   |   |   |--- class: 0.0
```

```
cm_knn = confusion_matrix(y_test, y_pred_knn)
plt.figure(figsize=(6,4))
sns.heatmap(cm_knn, annot=True, fmt='d', cmap='Greens')
plt.title('KNN Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```



Conclusion:

In our classification experiments, we used the Decision Tree and K-Nearest Neighbors (KNN) classifiers to predict loan default status. The Decision Tree model, with a maximum depth set to 3 for clarity, achieved an accuracy of about 88.4%. However, while it performed well on the majority (non-default) class, its performance on the default class was very poor, as seen in its classification report and confusion matrix.

The KNN classifier achieved an accuracy of approximately 87.5%. Similar to the Decision Tree, KNN exhibited strong performance for predicting non-default loans but struggled with the minority (default) class, resulting in low precision and recall for defaults.

Experiment 7

Aim:To implement different clustering algorithms.

Theory:

Clustering is an unsupervised machine learning technique used to group similar data points into clusters without predefined labels. In this experiment, we implemented the K-Means Clustering Algorithm as well as DBSCAN and Hierarchical Clustering Algorithm on a real-world dataset (loan_default_r.csv) using numerical features.

1)**K-Means Clustering:**K-Means is a centroid-based algorithm that partitions the dataset into K distinct non-overlapping subsets (clusters).The goal is to minimize intra-cluster variance (distance between points and their cluster centroid).

Algorithm Steps:

1. Select the number of clusters (k).
2. Initialize centroids randomly.
3. Assign each data point to the closest centroid.
4. Recalculate centroids as the mean of points in each cluster.
5. Repeat steps 3–4 until centroids do not change significantly or a maximum number of iterations is reached.

Mathematical Steps:

- Compute the distance between a point x_i and centroid C_j:

$$d(x_i, C_j) = \sqrt{\sum_{d=1}^{n} (x_{id} - C_{jd})^2}$$

- Update centroid:

$$C_j = \frac{1}{|S_j|} \sum_{x_i \in S_j} x_i$$

where S_j is the set of points assigned to cluster

2)DBSCAN is an **unsupervised machine learning algorithm** used for **clustering** based on **data density**. It groups closely packed points into clusters and marks sparse regions as noise.

Key Concepts:

1. **Epsilon (ε):** Radius of neighborhood around a point.
2. **MinPts:** Minimum number of points required to form a dense region (core point).
3. **Core Point:** Has at least MinPts within ε radius.
4. **Border Point:** Has fewer than MinPts within ε but lies within the neighborhood of a core point.
5. **Noise (Outlier):** Not a core point and not within ε of any core point.

Steps in DBSCAN:

1. **Select a point randomly** from the dataset.
2. **Check the number of points** within the radius ε.
    - If **≥ MinPts** → it's a **core point**, a new cluster is formed.
    - If **< MinPts** → mark it as **noise** for now (might become part of a cluster later).
3. **Expand the cluster**:
    - For each point within ε of the core point:
        - If it's also a core point, add all its neighbors to the cluster (recursively).
        - If it's a border point, add it to the current cluster (but don't expand further).
4. **Repeat** until all points are assigned to a cluster or labeled as noise.

3)Hierarchical Clustering:

Hierarchical clustering is an unsupervised machine learning algorithm used to group data into a tree of nested clusters — forming a structure called a dendrogram. It does not require you to pre-specify the number of clusters.

Types of Hierarchical Clustering:

<u>Agglomerative (Bottom-Up) – Most common</u>

- Each data point starts as its own cluster.
- Pairs of clusters are merged as you move up the hierarchy.
- Divisive (Top-Down) – Less common
- All points start in one cluster.
- Clusters are split recursively as you go down.
- We'll focus on Agglomerative since it's widely used.

Steps in Agglomerative Hierarchical Clustering:

1. Start with each data point as its own cluster (n clusters for n points).
2. Compute the distance (e.g., Euclidean) between all clusters.
3. Merge the two closest clusters (based on a linkage criterion).
4. Update the distance matrix after merging.
5. Repeat steps 2–4 until all points are merged into a single cluster or until a desired number of clusters is reached.

Linkage Criteria (How distances between clusters are computed):

1. Single Linkage: Minimum distance between two points in different clusters.
2. Complete Linkage: Maximum distance between two points in different clusters.
3. Average Linkage: Average distance between all points in the two clusters.
4. Ward's Method: Minimizes the total within-cluster variance.

📊 Dendrogram:

- A tree-like diagram that shows how clusters are merged/split.
- The height of the branches represents the distance (or dissimilarity).
- You can cut the dendrogram at a certain height to select the number of clusters.

Dataset Used:

Dataset: loan_default_r.csv

Selected Features: Age, Income, LoanAmount, CreditScore, MonthsEmployed, NumCreditLines, InterestRate, LoanTerm, and DTIRatio.

Mathematical Insight:

The Elbow Method was used to determine the optimal number of clusters by plotting the inertia (within-cluster sum of squares) against various values of k.

From the elbow plot, the optimal value of k was selected, and K-Means was applied accordingly.

Plot Information:

Elbow Plot: Visualizes how inertia decreases with increasing number of clusters and helps select the optimal k.

Cluster Visualization (Not shown fully here): Often performed using PCA or t-SNE for reducing dimensions to 2D, followed by plotting colored clusters.

Implementation:
1)Load and Explore Data

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
```

```
data = pd.read_csv('loan_default_r.csv')
print("Dataset shape:", data.shape)
data.head()
```

```
Dataset shape: (21561, 22)
```

	LoanID	Age	Income	LoanAmount	CreditScore	MonthsEmployed	NumCreditLines	InterestRate	LoanTerm	DTIRatio	...	MaritalStatus	HasMortgage	HasDependents	LoanPurpose	HasCoSigner	Default	Education_encoded	EmploymentType_encoded	NewLoanScore	ExtraNF
0	I38PQUQS96	56	85994	50587	520	80	4	15.23	36	0.44	...	Divorced	Yes	Yes	Other	Yes	0.0	0	0	159131.535447	0.846502
1	HPSK72WA7R	69	50432	124440	458	15	1	4.81	60	0.68	...	Married	No	No	Other	Yes	0.0	2	0	293055.484437	0.410480
2	C1OZ8DPJ8Y	46	84208	129188	451	26	3	21.17	24	0.31	...	Divorced	Yes	Yes	Auto	No	1.0	2	3	290773.486478	0.280065
3	V2KKSFM3UN	32	31713	44799	743	0	3	7.07	24	0.23	...	Married	No	No	Business	No	0.0	1	0	125590.176492	0.254075
4	EY08JDHTZP	60	20437	9139	633	8	4	6.51	48	0.73	...	Divorced	No	Yes	Auto	No	0.0	0	3	37378.907563	0.975988

5 rows × 22 columns

2)Scales the numerical features to normalize the data using StandardScaler.

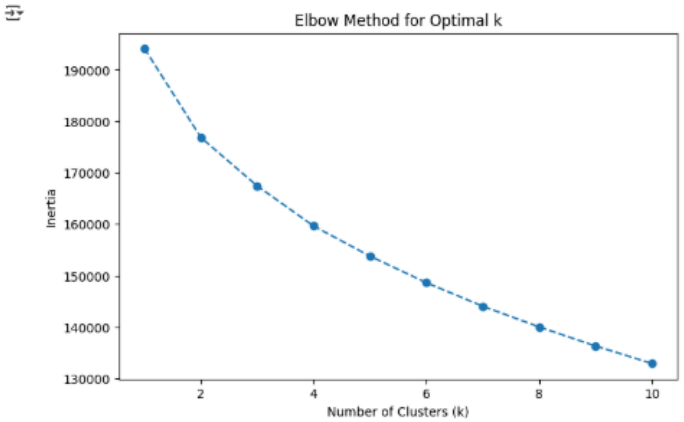
```
numerical_features = ['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed',
                      'NumCreditLines', 'InterestRate', 'LoanTerm', 'DTIRatio']

X = data[numerical_features]
```

Double-click (or enter) to edit

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

3)Initializes a loop to calculate inertia for different values of K to use the Elbow Method to find the optimal number of clusters.

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

    for k in K_range:
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        kmeans.fit(X_scaled)
        inertia.append(kmeans.inertia_)

    # Plot Elbow Curve
    plt.figure(figsize=(8, 5))
    plt.plot(K_range, inertia, marker='o', linestyle='--')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Inertia')
    plt.title('Elbow Method for Optimal k')
    plt.show()
```

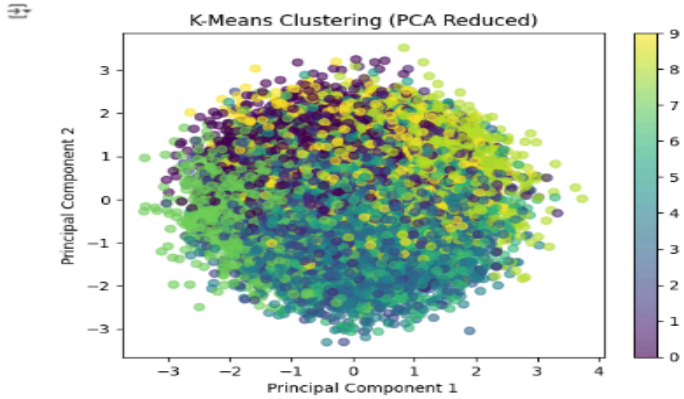


4)Applies K-Means clustering to the dataset using the chosen optimal number of clusters.

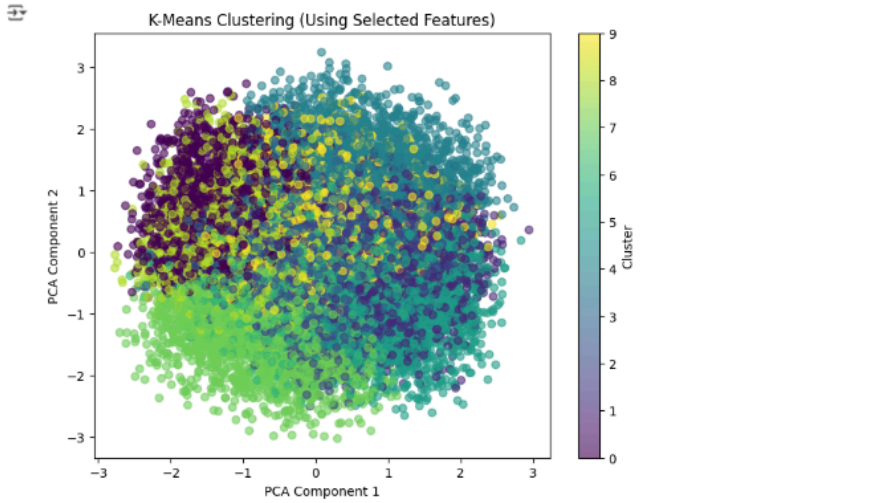
```
optimal_k = 10
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
data['Cluster'] = kmeans.fit_predict(X_scaled)
```

5) (Re)Imports libraries; possibly due to repeated or unorganized code cells and perform clustering related data-preprocessing and visualization.

```
[ ] from sklearn.decomposition import PCA

    pca = PCA(n_components=2)  # Reduce to 2D
    X_pca = pca.fit_transform(X_scaled)

    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=data['Cluster'], cmap='viridis', alpha=0.6)
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.title('K-Means Clustering (PCA Reduced)')
    plt.colorbar()
    plt.show()
```



```
[ ] # Select better features for clustering
    selected_features = ['CreditScore', 'LoanAmount', 'DTIRatio', 'InterestRate', 'MonthsEmployed']
    X = data[selected_features]

    # Standardize the data
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Apply K-Means
    kmeans = KMeans(n_clusters=10, random_state=42, n_init=10)
    data['Cluster'] = kmeans.fit_predict(X_scaled)

    # Reduce dimensions using PCA for better visualization
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_scaled)

    # Plot the clusters
    plt.figure(figsize=(8, 6))
    plt.scatter(X_pca[:, 0], X_pca[:, 1], c=data['Cluster'], cmap='viridis', alpha=0.6)
    plt.xlabel('PCA Component 1')
    plt.ylabel('PCA Component 2')
    plt.title('K-Means Clustering (Using Selected Features)')
    plt.colorbar(label='Cluster')
    plt.show()
```



6)

a)Calculates the Silhouette Score for the entire dataset using the features X_scaled and cluster labels stored in data['Cluster'].

b)Randomly selects a subset (max 10,000 points) from the dataset to speed up computation and computes the **Silhouette Score** only on this sample to reduce computational cost, especially useful for very large datasets.

```
[ ] from sklearn.metrics import silhouette_score

    sil_score = silhouette_score(X_scaled, data['Cluster'])
    print(f'Silhouette Score: {sil_score:.4f}')
```

```
Silhouette Score: 0.1687
```

```
[ ] from sklearn.metrics import silhouette_score
    import numpy as np

    # Sample a subset (e.g., 10,000 points) for faster computation
    sample_size = min(10000, len(X_scaled))  # Use 10,000 or full dataset if smaller
    idx = np.random.choice(len(X_scaled), sample_size, replace=False)
    X_sampled = X_scaled[idx]
    labels_sampled = data['Cluster'].iloc[idx]

    # Compute silhouette score on the sample
    sil_score = silhouette_score(X_sampled, labels_sampled)
    print(f'Silhouette Score: {sil_score:.4f}')
```

```
Silhouette Score: 0.1690
```

7)Evaluate the optimal number of clusters (k) for K-Means clustering using the Silhouette Score.

Turns out ,among the tested values, k = 10 yields the highest Silhouette Score (0.1682), suggesting it's the most suitable number of clusters based on this metric.
However, the scores are still quite low (all < 0.2), implying clusters are weakly separated.

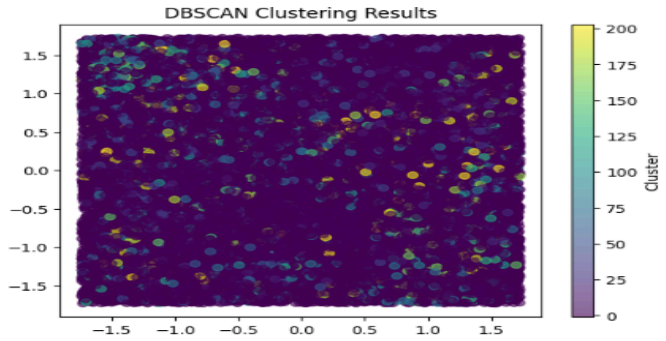
```
[ ] from sklearn.metrics import silhouette_score
    import numpy as np
    from sklearn.cluster import KMeans

    # Sample a subset for faster computation
    sample_size = min(10000, len(X_scaled))  # Use 10,000 or full dataset if smaller
    idx = np.random.choice(len(X_scaled), sample_size, replace=False)
    X_sampled = X_scaled[idx]

    for k in [2, 4, 5, 6,9,10]:
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        clusters = kmeans.fit_predict(X_scaled)

        # Compute silhouette score on the sampled subset
        labels_sampled = clusters[idx]  # Use sampled cluster labels
        score = silhouette_score(X_sampled, labels_sampled)

        print(f'k={k}, Silhouette Score: {score:.4f}')
```

```
k=2, Silhouette Score: 0.1499
k=4, Silhouette Score: 0.1430
k=5, Silhouette Score: 0.1448
k=6, Silhouette Score: 0.1509
k=9, Silhouette Score: 0.1623
k=10, Silhouette Score: 0.1682
```

8)Applies DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm using the sklearn.cluster module.

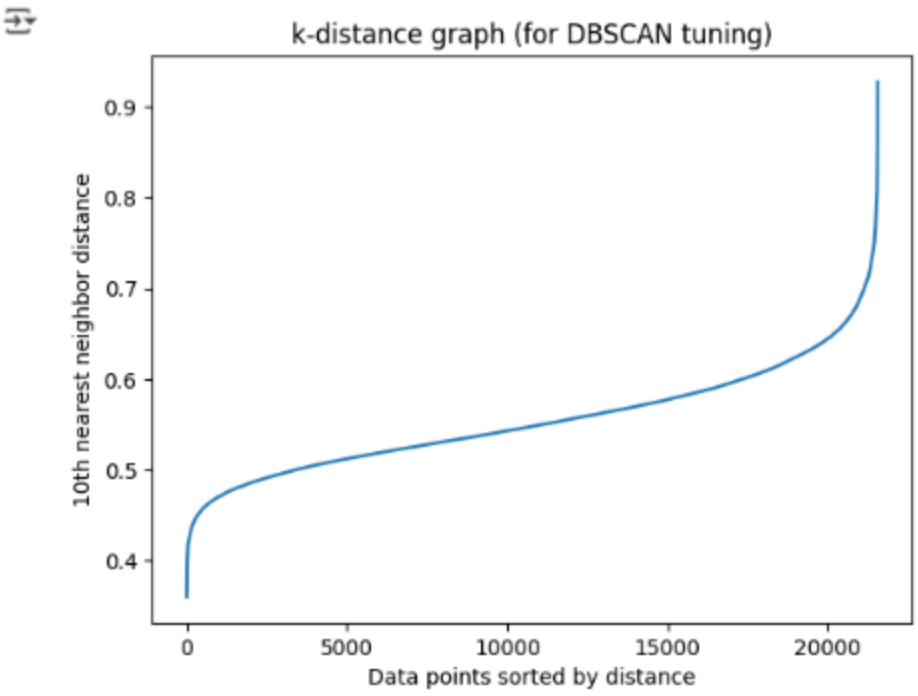
```
[ ]  from sklearn.cluster import DBSCAN
     from sklearn.metrics import silhouette_score

     # DBSCAN with reasonable default values
     dbscan = DBSCAN(eps=0.5, min_samples=10)  # Adjust `eps` based on dataset
     db_clusters = dbscan.fit_predict(X_scaled)

     # Assign clusters
     data['DBSCAN_Cluster'] = db_clusters

     # Compute silhouette score (only for clustered points, ignoring noise)
     valid_clusters = db_clusters != -1  # Ignore noise points (-1)
     if valid_clusters.sum() > 0:  # Check if there are valid clusters
         score = silhouette_score(X_scaled[valid_clusters], db_clusters[valid_clusters])
         print(f'DBSCAN Silhouette Score: {score:.4f}')
     else:
         print("DBSCAN found mostly noise; try increasing `eps`.")

     # Visualizing DBSCAN Clusters
     import matplotlib.pyplot as plt

     plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=db_clusters, cmap='viridis', alpha=0.6)
     plt.title("DBSCAN Clustering Results")
     plt.colorbar(label="Cluster")
     plt.show()
```

```
DBSCAN Silhouette Score: -0.4034
```



9) tune the eps parameter for DBSCAN by generating a k-distance graph.

```
[ ] from sklearn.neighbors import NearestNeighbors
    import matplotlib.pyplot as plt
    import numpy as np

    # Fit Nearest Neighbors
    neigh = NearestNeighbors(n_neighbors=10)
    nbrs = neigh.fit(X_scaled)
    distances, indices = nbrs.kneighbors(X_scaled)

    # Sort and plot distances (for the 10th nearest neighbor)
    distances = np.sort(distances[:, 9])
    plt.plot(distances)
    plt.xlabel("Data points sorted by distance")
    plt.ylabel("10th nearest neighbor distance")
    plt.title("k-distance graph (for DBSCAN tuning)")
    plt.show()
```

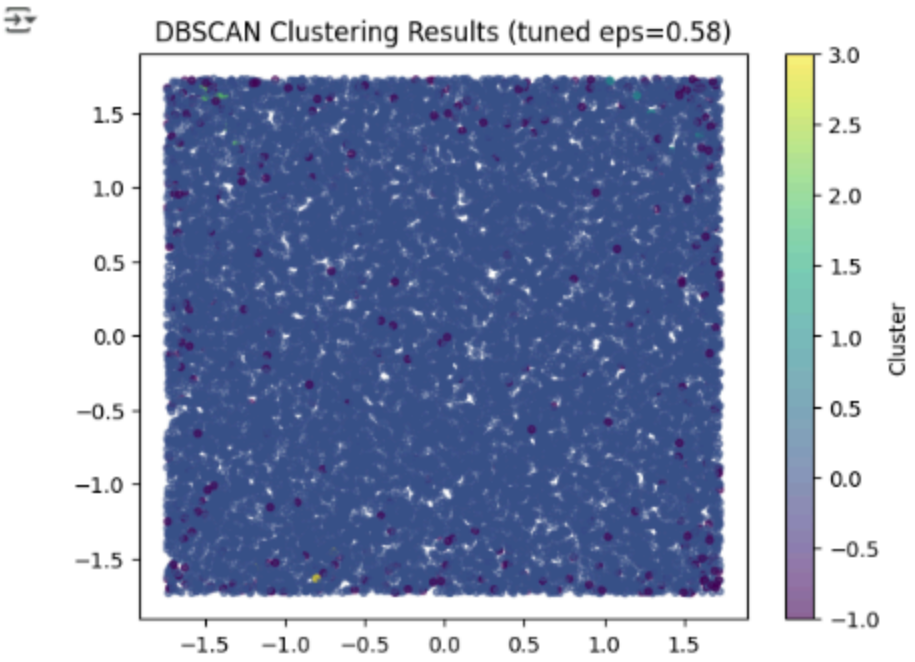


10)DBSCAN Clustering (Tuned eps=0.58)

- DBSCAN is applied with eps=0.58, but most points are classified as noise (dark purple, -1).

- The silhouette score is -0.0809, indicating poor clustering. A higher eps may be needed.

```
[ ] from sklearn.cluster import DBSCAN
    import matplotlib.pyplot as plt

    dbscan = DBSCAN(eps=0.58, min_samples=10)
    db_clusters = dbscan.fit_predict(X_scaled)

    data['DBSCAN_Cluster'] = db_clusters

    # Visualize
    plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=db_clusters, cmap='viridis', s=10, alpha=0.6)
    plt.title("DBSCAN Clustering Results (tuned eps=0.58)")
    plt.colorbar(label="Cluster")
    plt.show()

    # Silhouette score (only for clustered points)
    from sklearn.metrics import silhouette_score
    valid_points = db_clusters != -1
    if valid_points.sum() > 0:
        sil_score = silhouette_score(X_scaled[valid_points], db_clusters[valid_points])
        print(f"DBSCAN Silhouette Score (ignoring noise): {sil_score:.4f}")
    else:
        print("All points classified as noise. Try increasing eps.")
```

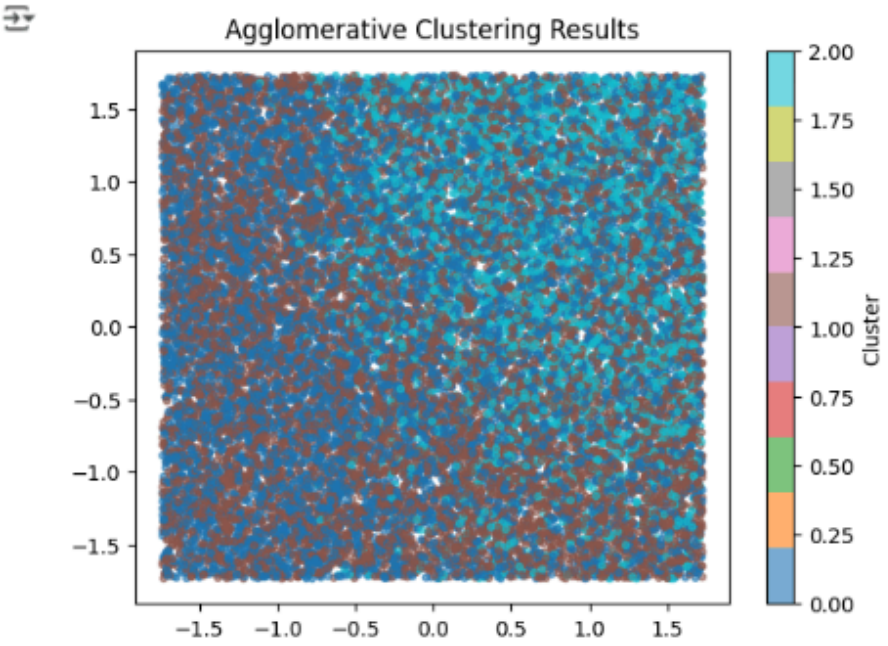



```
DBSCAN Silhouette Score (ignoring noise): -0.0809
```

11)Dendrogram for Hierarchical Clustering

- A dendrogram is created using hierarchical clustering (ward method) on a sample of data.
- Helps determine the optimal number of clusters by identifying where to cut the tree.

```python
from sklearn.cluster import AgglomerativeClustering
import matplotlib.pyplot as plt

# Choose number of clusters (start with what looked good in KMeans, say k=3)
agglo = AgglomerativeClustering(n_clusters=3, linkage='ward')
agglo_clusters = agglo.fit_predict(X_scaled)

data['Agglo_Cluster'] = agglo_clusters

# Visualize
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=agglo_clusters, cmap='tab10', s=10, alpha=0.6)
plt.title("Agglomerative Clustering Results")
plt.colorbar(label="Cluster")
plt.show()

# Silhouette score
from sklearn.metrics import silhouette_score
score = silhouette_score(X_scaled, agglo_clusters)
print(f'Agglomerative Clustering Silhouette Score: {score:.4f}')
```

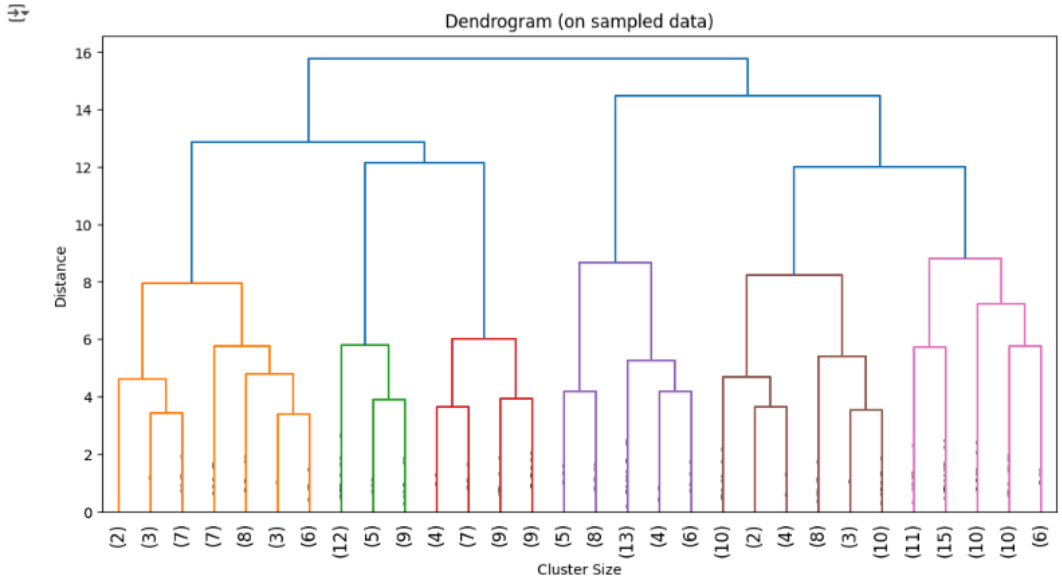



```
Agglomerative Clustering Silhouette Score: 0.0760
```

12)Agglomerative Clustering (k=3)

- Hierarchical clustering is applied with n_clusters=3 using the ward linkage method.
- The silhouette score is **0.0760**, indicating weak clustering but better than DBSCAN.

```python
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

# Perform linkage on a sample (if dataset too big)
# Use a small sample to avoid memory overload:
sample_X = X_scaled[::100]   # take every 100th record to make it faster

linked = linkage(sample_X, method='ward')

plt.figure(figsize=(12, 6))
dendrogram(linked, truncate_mode='lastp', p=30, leaf_rotation=90., leaf_font_size=12., show_contracted=True)
plt.title('Dendrogram (on sampled data)')
plt.xlabel('Cluster Size')
plt.ylabel('Distance')
plt.show()
```



Conclusion:

In this experiment, we successfully implemented and analyzed different clustering algorithms including K-Means, DBSCAN, and Hierarchical Clustering. Each method demonstrated its capability in identifying meaningful clusters in an unsupervised manner. K-Means proved effective for spherical-shaped clusters, DBSCAN for arbitrary shapes and noise detection, and Hierarchical Clustering for structure discovery and visualization through dendrograms. The clustering outcomes were visualized using scatter plots and dendrograms, providing a clear understanding of how the data is grouped.

Experiment 8

Aim:

To design and implement a music recommendation system using unsupervised machine learning techniques, namely **K-Means Clustering** and **Principal Component Analysis (PCA)** on the `spotify.csv` dataset.

Theory:

The goal of this experiment is to group songs with similar characteristics and recommend songs from the same group. This is achieved using two key techniques:

1. **Principal Component Analysis (PCA)** – to reduce dimensionality and enable effective visualization.

2. **K-Means Clustering** – to form groups (clusters) of similar songs based on their audio features.

Dataset Description:

- **Name:** spotify.csv
- **Records:** Approximately 1100+ songs
- **Attributes:** Includes numerical attributes such as danceability, energy, loudness, acousticness, instrumentalness, tempo, etc.
- **Purpose:** These features are used to identify similarities between songs and cluster them accordingly.

Steps Involved:

1. Data Preprocessing:

- Selected relevant numeric features related to song characteristics.
- Applied **StandardScaler** to standardize the features, which is essential for distance-based models like K-Means and PCA.

```
 Dataset Loaded Successfully!
Columns in the dataset:
Index(['valence', 'year', 'acousticness', 'artists', 'danceability',
       'duration_ms', 'energy', 'explicit', 'id', 'instrumentalness', 'key',
       'liveness', 'loudness', 'mode', 'name', 'popularity', 'release_date',
       'speechiness', 'tempo'],
      dtype='object')
```

```
df = df.drop(['id', 'name', 'artists', 'release_date'], axis=1)
```

```
df.head()
```

	valence	year	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	popularity	speechiness	tempo	Cluster
0	0.0594	1921	0.982	0.279	831667	0.211	0	0.878000	10	0.665	-20.096	1	4	0.0366	80.954	0
1	0.9630	1921	0.732	0.819	180533	0.341	0	0.000000	7	0.160	-12.441	1	5	0.4150	60.936	2
2	0.0394	1921	0.961	0.328	500062	0.166	0	0.913000	3	0.101	-14.850	1	5	0.0339	110.339	0
3	0.1650	1921	0.967	0.275	210000	0.309	0	0.000028	5	0.381	-9.316	1	3	0.0354	100.109	0
4	0.2530	1921	0.957	0.418	166693	0.193	0	0.000002	3	0.229	-10.096	1	2	0.0380	101.665	0

2. Dimensionality Reduction using PCA:

Objective:

To reduce the number of input features while retaining as much information (variance) as possible.

Process:

- PCA was applied with `n_components=2`.
- The explained variance ratio was checked to ensure that a significant portion of data variability is preserved.
- The 2D data was used for visualization of clusters.

Benefits:

- Helps visualize high-dimensional data.
- Reduces noise and computational complexity.
- Enhances the performance of clustering models.

```python
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Standardize the features
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df)

# Apply PCA (we'll keep 2 components for visualization)
pca = PCA(n_components=2)
pca_data = pca.fit_transform(scaled_data)
```

```python
# Show the first 5 PCA-transformed rows
print("First 5 rows after PCA (2 components):\n")
print(pca_data[:5])
```

```
First 5 rows after PCA (2 components):

[[-4.28266789 -2.295402  ]
 [-1.39369011  3.51566309]
 [-3.85297069 -1.73429532]
 [-2.53896489 -0.30484121]
 [-2.55129534  0.27545511]]
```

Scatter plot of PCA-reduced data before applying clustering.
This visualization represents the spread of songs across the first two principal components. It helps identify any natural grouping or separation in the data.

3. Clustering using K-Means:

Objective:

To group similar songs into k=6 clusters using K-Means clustering.

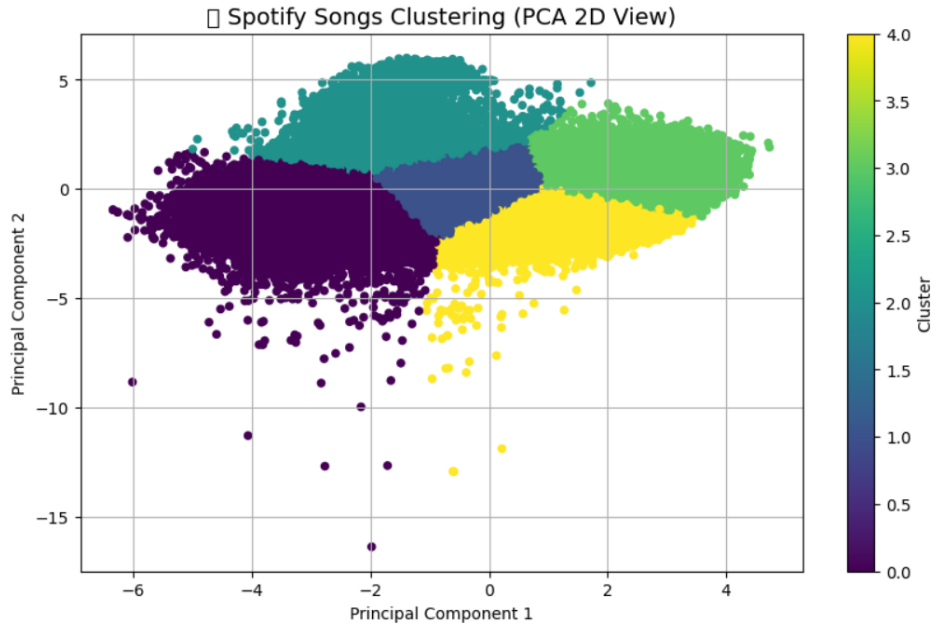
Process:

- KMeans from `sklearn.cluster` was used with `n_clusters=6`.
- The model was trained on standardized features.
- Each song was labeled with a cluster number from 0 to 5.
- PCA components were used to plot the clustered data.

```python
from sklearn.cluster import KMeans

# Let's say we choose 5 clusters
kmeans = KMeans(n_clusters=5, random_state=42)
clusters = kmeans.fit_predict(pca_data)

# Add cluster info back to original dataframe
df['Cluster'] = clusters
```

```python
import matplotlib.pyplot as plt

# Basic scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(pca_data[:, 0], pca_data[:, 1], c=clusters, cmap='viridis', s=20)
plt.title(" Spotify Songs Clustering (PCA 2D View)", fontsize=14)
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label='Cluster')
plt.grid(True)
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 127912 (\N{ARTIST PALETTE}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



Songs plotted with cluster labels using PCA components.
Each color represents a different cluster of songs that share similar characteristics. The X and Y axes correspond to the first and second principal components.

Recommendation Logic:

Once the songs are clustered, we can recommend songs from the same cluster as a chosen song:

```
song_name = "Blinding Lights"
if song_name in original_df['name'].values:
    liked_song = original_df[original_df['name'] == song_name].iloc[0]
    liked_cluster = liked_song['Cluster']
    print(f"\n You liked: {liked_song['name']} by {liked_song['artists']} (Cluster {liked_cluster})\n"
    recommendations = original_df[
        (original_df['Cluster'] == liked_cluster) &
        (original_df['name'] != song_name)
    ][['name', 'artists']].head(5)
    print("Recommended Songs:")
    print(recommendations)
else:
    print(" Song not found in the dataset.")
```

```
 You liked: Blinding Lights by ['The Weeknd'] (Cluster 4)

Recommended Songs:
                                        name                 artists
5667                                Gandagana      ['Georgian People']
7186  Woke Up This Morning (My Baby She Was Gone)           ['B.B. King']
7232                Blue Train - Remastered 2003        ['John Coltrane']
7236                                Milestones          ['Miles Davis']
7252                One For Daddy-O - Remastered  ['Cannonball Adderley']
```

Conclusion:

In this experiment, a recommendation system was implemented using unsupervised learning. Dimensionality reduction via PCA allowed effective visualization and simplification of the data. K-Means clustering grouped songs with similar features, allowing content-based recommendations.

This approach is scalable, efficient, and interpretable, making it suitable for music-based recommendation systems.

Experiment - 9

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

Theory:

Case Study Overview:

In this case study, we explore how modern big data tools like Apache Spark can be effectively leveraged for conducting Exploratory Data Analysis (EDA). Traditional data analysis techniques may struggle with large datasets due to performance bottlenecks, but Spark's distributed architecture provides a scalable and efficient solution.

Technology in Focus: Apache Spark

Apache Spark is an open-source distributed computing system designed for big data analytics. It provides a high-performance framework for processing massive datasets in parallel across a cluster of computers. Spark supports various languages including Scala, Java, Python (via PySpark), R, and SQL.

How Apache Spark Works:

1. **Driver Program Initialization:**
   The Spark application begins with a driver program, where the user writes code using Spark APIs. The driver constructs a Directed Acyclic Graph (DAG) of stages representing the logical execution plan.
2. **Resource Allocation:**
   Spark employs a cluster manager such as YARN, Mesos, or its own standalone manager to allocate resources and manage worker nodes (Executors).
3. **Task Distribution:**
   The DAG Scheduler divides jobs into tasks and distributes them across the executors. Each task may involve reading, transforming, or writing data.

4. **Task Execution and Results:**
   Executors execute the assigned tasks in-memory to optimize performance. They cache intermediate results and return final outcomes to the driver or to a persistent storage.

Common Use Cases:

- Real-time processing of log or stream data
- Running large-scale machine learning algorithms
- Executing ETL pipelines efficiently
- Analyzing structured and unstructured datasets

Exploratory Data Analysis in Apache Spark:

Performing EDA using PySpark (the Python API for Spark) is crucial for gaining initial insights and assessing data quality before diving into deeper analysis or machine learning.

Steps Involved:

1. **Data Importing:**
   Data is loaded from various sources such as CSV, JSON, Parquet files, or databases using Spark's flexible connectors.
2. **Schema Inspection:**
   The dataset's structure is reviewed—column names, data types, and nullability are verified to understand the data schema.
3. **Data Preview:**
   A subset of records is viewed to identify potential inconsistencies, formatting problems, or missing values.
4. **Summary Statistics:**
   Descriptive statistics such as mean, standard deviation, minimum, and maximum values are calculated to understand data distribution.
5. **Missing Value Detection:**
   Null or missing entries are located, enabling informed decisions on whether to drop, fill, or impute them.
6. **Distribution Analysis:**
   Value distributions for categorical and numerical columns are studied to reveal patterns, common entries, and outliers.
7. **Correlation Analysis:**
   Relationships between numeric variables are computed to evaluate multicollinearity and guide feature selection.

8. **Filtering and Conditions:**
   Specific subsets of data are explored using logical filters (e.g., values within a certain range) for targeted investigation.
9. **Sampling:**
   For extremely large datasets, a representative sample is extracted to expedite the exploration process.
10. **Data Preparation for Further Steps:**
    Based on the insights gained, the data is cleaned and transformed, readying it for modeling or advanced analysis. Spark typically works in tandem with visualization libraries outside its environment.

Conclusion:

Apache Spark proves to be a powerful tool for performing EDA at scale. Its in-memory computation, distributed processing, and support for multiple APIs (RDD, Data Frame, Dataset) make it highly suitable for handling large volumes of data efficiently. Combined with the flexibility of Pandas for smaller datasets or local analysis, this hybrid approach offers a comprehensive framework for data exploration.

This case study successfully demonstrates how Spark's architecture and functionality can be leveraged to extract meaningful insights from big data in a practical, structured manner.

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 a method of processing data that is generated and consumed in real time or near real time. This is especially useful when immediate feedback or actions are required—like monitoring sensor inputs, processing transactions, or managing live feeds.

Unlike traditional methods that wait until all data is collected, streaming systems process information as soon as it's available. This allows for quicker decision-making and timely insights.

2. **How data streaming takes place using Apache spark.**
   Apache Spark handles real-time data using its feature called Structured Streaming. This enables developers to process continuous flows of data with ease using familiar SQL or DataFrame APIs. It is designed for scalability and reliability. How it works:

1. **Input Source**
   The streaming starts with data being read continuously from sources such as:

- Apache Kafka
- File directories (watching for new files)
- Network sockets
- Amazon Kinesis
- Other custom data providers

Spark receives this live data stream for processing.

2. **Data as an Unbounded Table**
   In Spark, incoming streaming data is viewed as a growing table where each new entry adds a new row. Standard operations like filtering, selecting, or grouping can be applied, just like with regular tables.
3. **Query Definition**
   Users define operations on the data stream (like counting values or calculating stats). Spark translates this into a logical plan, then converts it to a physical execution plan for efficient performance.

4. **Micro-Batch Mechanism**
   Instead of processing each incoming piece of data one by one, Spark gathers data for small time windows (e.g., every second), and processes them in groups. This “micro-batching” keeps it responsive but still efficient.

5. **Output Destination**
   After the data is processed, results are stored or displayed via:

- Console (for quick checks)
- Kafka
- Databases
- Filesystems

The output can be configured in different ways:

- **Append**: Adds only new results
- **Update**: Updates only changed results
- **Complete**: Outputs the full result every time

Spark also provides built-in fault tolerance to recover from failures automatically.

Conclusion:

Batch and stream analysis serve different purposes in data processing. Batch processing handles large datasets collected over time and is suitable for deep, detailed analysis. Stream processing deals with real-time data, allowing immediate insights and actions. While batch ensures thoroughness and precision, streaming offers speed and responsiveness. Using both together enables systems to be both intelligent and fast, handling a wide range of data needs effectively.

Customer Segmentation using Clustering Algorithms

ON

Submitted in partial fulfillment of the requirements of the degree of

Bachelor of Engineering

(Information Technology)

By

Komal Sabale(45)

Krushikesh Shelar (51)

Shweta Wadhwa (58)

Under the guidance of

Dr. Ravita Mishra

Department of Information Technology

VIVEKANAND EDUCATION SOCIETY'S INSTITUTE OF TECHNOLOGY,

Chembur, Mumbai 400074

(An Autonomous Institute, Affiliated to University of Mumbai) April 2024

AIDS Lab Exp 11
Aim: Mini Project – Customer Segmentation using Clustering Algorithms

---

1.1 Introduction

Customer segmentation helps businesses categorize customers based on purchasing behavior. It improves marketing, loyalty, and retention strategies. This project uses unsupervised machine learning with RFM features (Recency, Frequency, Monetary) to group customers into meaningful segments.

2.2 Data Preprocessing

The dataset was cleaned by removing null CustomerIDs, negative quantities, and cancelled transactions. New features like TotalBill were created. RFM metrics were calculated per customer:

- Recency: Days since last purchase
- Frequency: Number of unique transactions
- Monetary: Total spend

2.3 Feature Scaling

RFM values were standardized using StandardScaler to ensure equal contribution to clustering models.

2.4 Clustering Models

- KMeans: Applied with k = 3–5. Chosen for its simplicity and speed.
- DBSCAN: Density-based clustering. Tuned with eps and min_samples.
- Agglomerative Clustering: Hierarchical model with Ward linkage.
- GMM (Gaussian Mixture Model): Soft clustering model based on probability distributions.

2.5 Evaluation

Clusters were evaluated using the Silhouette Score. KMeans and Agglomerative showed the best performance (0.58–0.61). DBSCAN was useful for detecting outliers but required careful parameter tuning.

2.6 Streamlit App

A web app was built using Streamlit to upload data, choose models, view clustering results, and download the output. It helps non-technical users run and interpret the segmentation pipeline interactively.

Chapter 4: Results and Discussion

4.1 Cluster Evaluation

Different clustering models were applied on the scaled RFM features. KMeans with k=3–5 showed strong separation. Agglomerative Clustering also performed well, with the highest Silhouette Score of 0.6065. DBSCAN detected some outliers but formed only one main cluster due to parameter sensitivity.

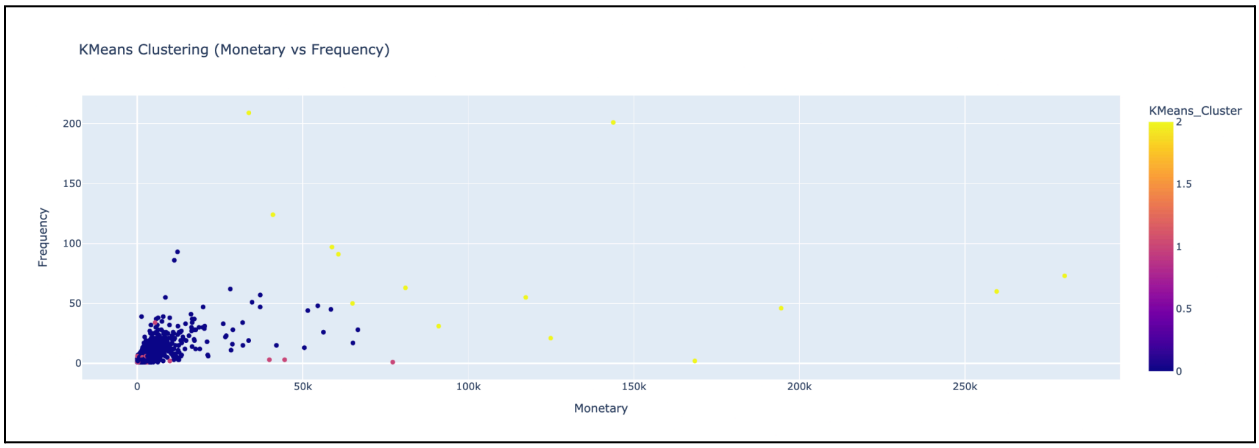
Clustering Method	Silhouette Score	Notes
🚀 KMeans	0.5853	Balanced and interpretable clusters. Good for general use cases.
🧭 Agglomerative	0.6065	Best Silhouette score. Suitable for uncovering natural hierarchies.
🌌 DBSCAN	Not Applicable	Only one cluster or too many noise points. Parameters need tuning.

4.2 KMeans Clustering

KMeans grouped customers into three primary clusters. The segments were labeled as:

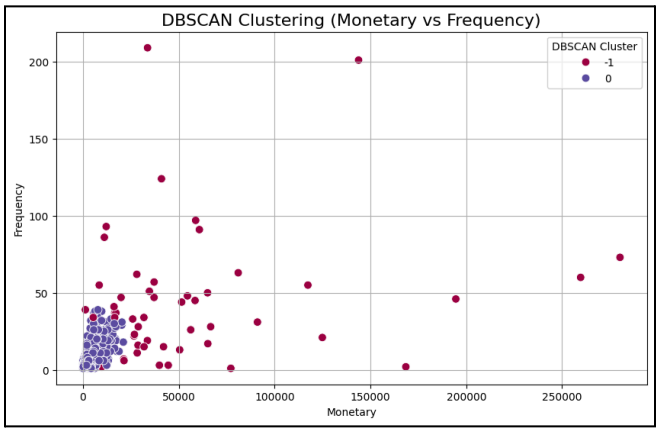
- Cluster 0: Loyal Regulars
- Cluster 1: Dormant/At-Risk
- Cluster 2: High-value VIPs

These labels were based on average Recency, Frequency, and Monetary values.



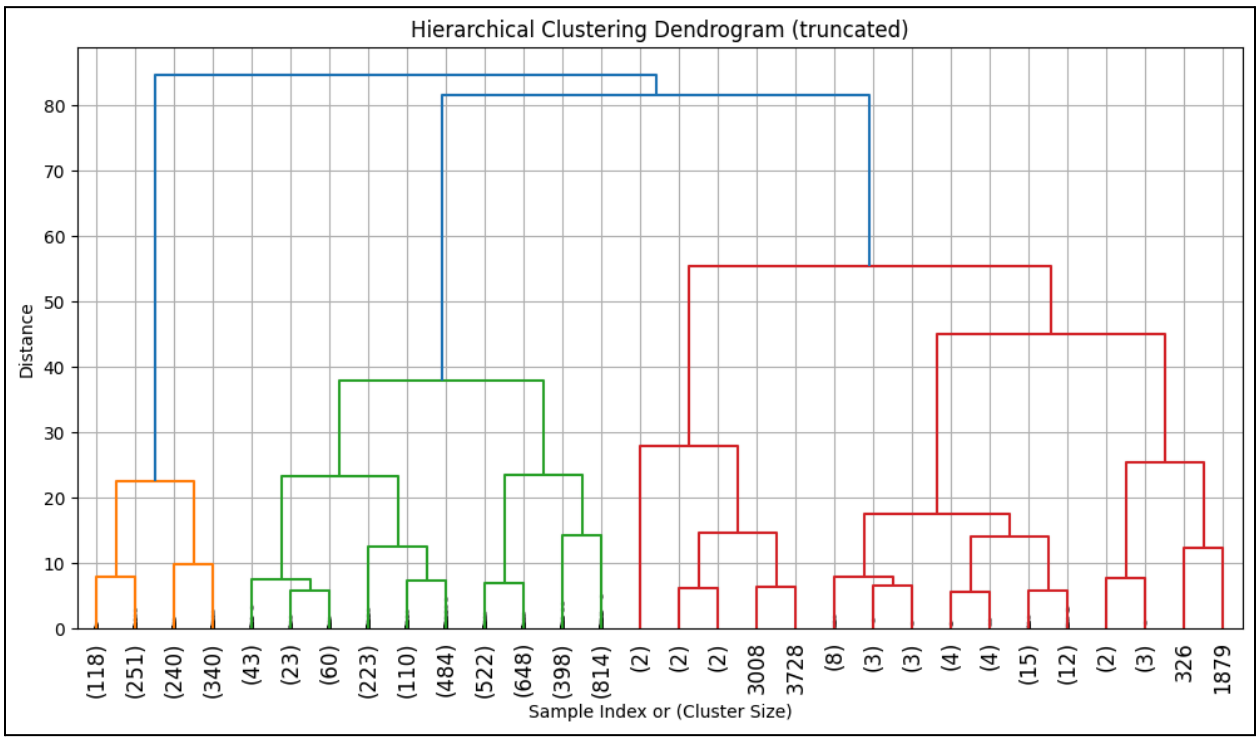
4.3 DBSCAN Clustering

DBSCAN was effective in identifying noise (-1) but struggled with sparse RFM features. It showed fewer distinct clusters under default parameters.



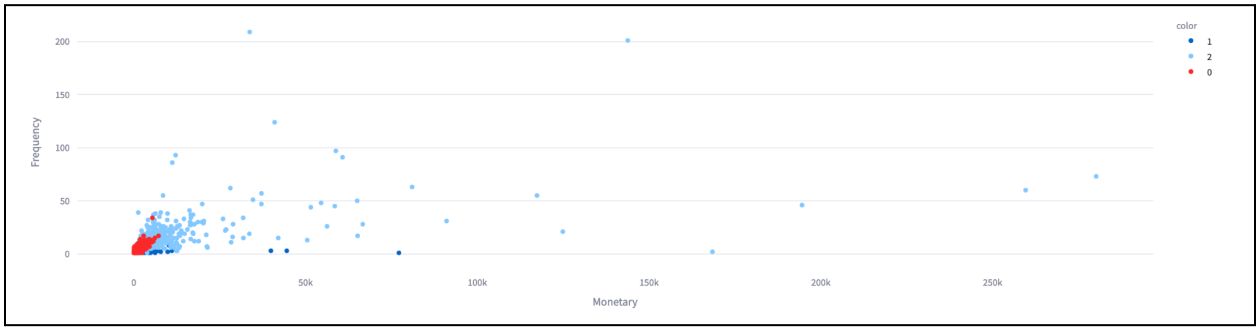
4.4 Agglomerative Clustering

Agglomerative Clustering revealed clear hierarchy among customers. The dendrogram helped identify the natural number of clusters (suggesting 4). Final cluster summaries showed tight behavioral groupings.



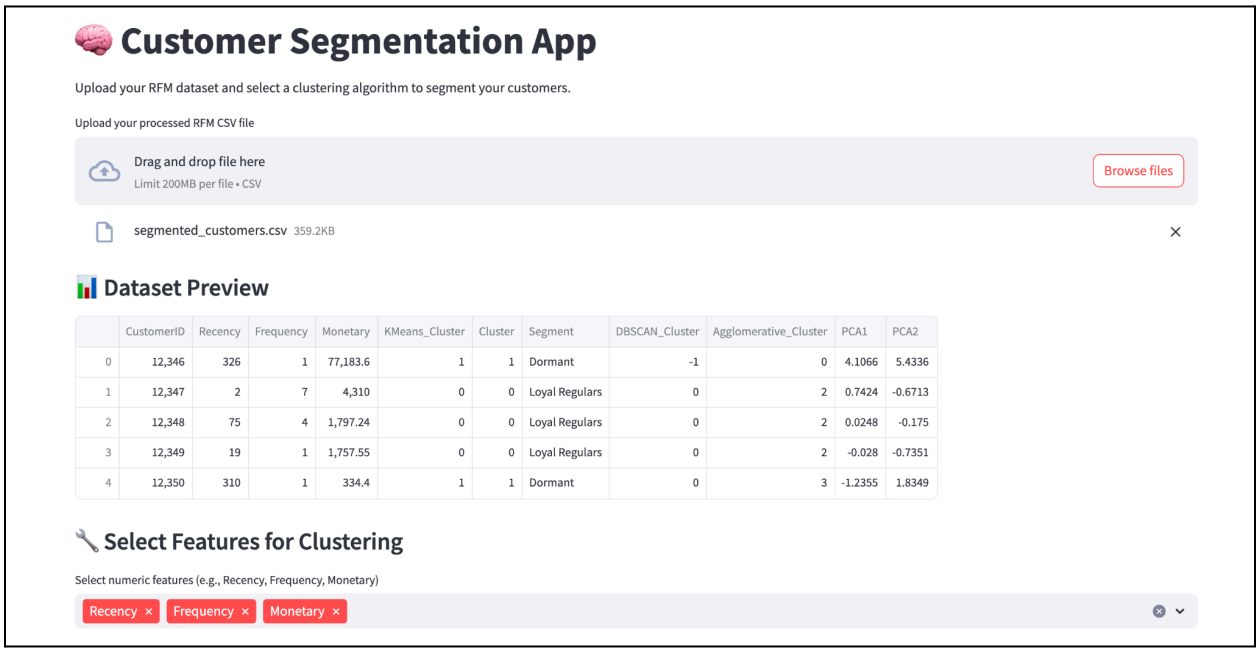
4.5 Gaussian Mixture Model (GMM)

GMM provided a probabilistic approach to clustering, where customers were grouped based on likelihood of belonging to a distribution. It produced soft cluster boundaries and worked well on scaled RFM features. GMM detected high-value clusters similar to KMeans, but with more overlap between boundaries.



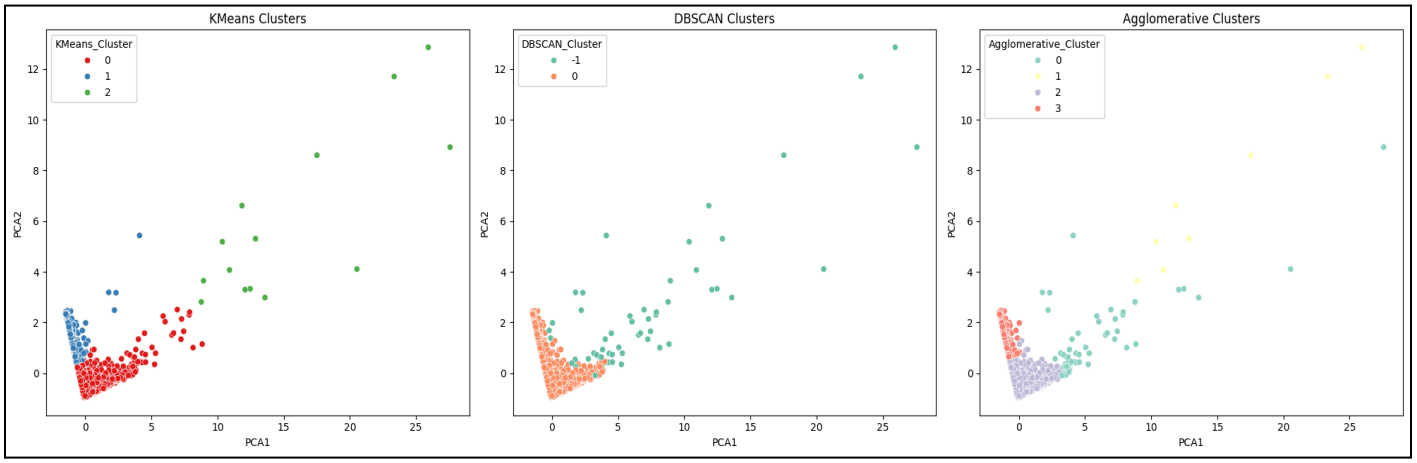
4.6 Streamlit App Output

The Streamlit app allowed interactive selection of algorithms, visualization of clusters, and downloading results. It simplified deployment for marketing teams.



4.7 Final Summary Table

The final clusters were summarized with average Recency, Frequency, and Monetary values, along with customer count



Cluster	Recency	Frequency	Monetary
0	40.98	4.85	2012.11
1	246.02	1.58	631.14
2	7.14	80.21	122888.41

Conclusion

This project demonstrates the use of unsupervised learning for customer segmentation using RFM features. By applying models like KMeans, DBSCAN, Agglomerative Clustering, and GMM, we identified distinct customer groups for targeted strategies. The Streamlit app provides a simple interface to explore and apply these insights, making the system practical and business-ready.

DISC-45

05/05

Page No.
Date

Assignment-1

1. What is AI? Considering the COVID-19 Pandemic situation, how AI helped to suevive & renovated our way of life with different applications?

Ans. Artificial Intelligence (AI) refers to simulation of human intelligence in machines that are peogrammed to think & learn like humans.

AI has helped during the COVID-19 pandemic in various ways:

1) Healthcare & Diagnosis:
AI peedicted vieus speed, assisted in diagnostics using X-ray, CT scans & sped up deug/ vaccine discovery.

2) Data & Peediction:
AI tracked vieus speed helping with resource allocation & govetnment planning.

3) Public health safety:
Contact tracing apps, powered by AI, helped track exposure & reduce vieus transmission.

4) Public awareness & mental health:
AI analyzed social media to gauge public sentiment, guiding mental health support & awareness campaigns.

5) Education & Remote Learning:
AI powered platforms allowed personalized online learning & remote education during schol closures

Date

2. What are AI agents terminologies, explain with example.

AI Agent terminologies:

1) Agent: An entity that percieves its environments & take actions to achieve goals.
   Eg. A chatbot answering user queries.

2) Environment: Everything the user interacts with.
   E.g. Roads, vehicles & traffic signals for a self-driving car.

3) Perception: The process of collecting data from the environment through sensors.
   E.g. Cameras in self driving cars, detecting obstacl

4) Actuators: Mechanisms that allow the agent to act on the enviroment. for more movement
   Eg. Robots arms/ wheels

5) Actions: The behaviour that agent performs based on i perception.
   Eg. Virtual assistant responding to a query.

3. How AI techniques is used to solve 8 puzzle problem?

Soln: Initial State:
1	2	3
4	0	6
7	5	8

Goal State:
1	2	3
4	5	6
7	8	0

Misplaced tiles : $h(n)=2$

Steps to solve 8-puzzle problem by A*:

1) Initialize a priority queue.
2) Insert the initial state with $F(n) = g(n) + h(n)$
3) While queue not empty :
   Remove state with lowest $F(n)$
   If state = goal, return solution.
   Generate valid moves (up, down, left, right)
   compute $g(n)$ & $h(n)$ for new states.
   Insert new states into queue.
4. Repeat until goal is reached.

Step 1 :
1	2	3
4	0	6
7	5	8

($h=2$)

Step 2 :
1	2	3
4	5	6
7	0	8

($h=1$)

Step 3 : Goal
1	2	3
4	5	6
7	8	0

($h=0$)

4. What is PEAS descriptor ? Give PEAS descriptor for following :

Soln: PEAS stands for Performance measure, environment, actuators & sensors.

P → Criteria to evaluate the agent's success.

E → (Environment) → surrounding where agent operates

A → (Actuators) → common components that allow agent to take actions

S → (Sensors) components that help agent to perceive environment.

1. Taxi Deiver:
P - Reaching destination, fuel efficiency.
E - Roads, traffic, pedestrians.
A - Stearing wheel, accelerator, brakers.
S - Cameras, GPS, Speedometer.

2. Medical Diagnosis System:
P - Accuracy of Diagnosis, patient satisfaction.
E - medical records, test results, symptoms.
A - Peescription generation, Report generation.
S - patient data input, lab test results.

3. Music composer:
P - Quality of composition, user satisfication.
E - Musical genres, musical theory constraints.
A - Music synthesis, instruments, generating musical notes
S - composition structure, user feedback.

4. Aiecraft Autolander:
P - Smooth & safe landing
E - Weather, runway condition, aie traffic.
A - Flaps, landing gear, throttle.
S - Altitude sensor, GPS, wind direction sensor.

5. Essay Evaluate:
P - Accurate grading, feedback quality.
E - Grammer rules, submitted essays.
A - displaying grades & feedback
S - Text input, spelling & grammer checkers.

6. Robotic sentry gun for tech lab:
P- Correctly identifying threats.
E - Intruders, authorized personnel.
A - Camera movement, alarm system, firing mechanism
S - motion sensors, facial recognition.

5. Categorize a shopping bot for an offline bookstore according to each of the six dimensions.
1. Partially observable
— bot may not have full visibility of store's inventory
2. Stochastic
— outcomes uncertain due to customer behaviour, stock availability
3. Sequential
— Bot will suggest books based on previous customer queries
4. Dynamic
— Bookstore environment changes like books, getting sold
5. Discrete
— Bots process finite no. of actions.
6. Multi agent
— Bot interacts with customers & store employees each have their own goals.

6. Differentiate between model-based & utility-based agents.

Model-based	Utility-based
1. Uses internal model of environment to make decisions	Chooses actions based on utility fn's that measures performances.

2) Decisions based on past & present percepts.	Selects action based on maximizing utility.
3) Can be goal-based but doesn't necessarily optimize for best performance.	optimizes for highest possible utility ensuring performance.
4) Ex. robot vaccum using a map to navigate.	Ex. Self driving car, choosing safest & fastest route.

7. Explain the architecture of knowledge-based agent & learning agent.

Sol^n:

Knowledge-based agent:

A knowledge-based agent (KBA) uses stored knowledge to make decisions. It consists of

1) knowledge Base → stores facts, rules & logic.
2) Inteference Enginge → Uses reasoning to derive conclusions
3) Perception (Sensors) → Gathers new information from enviroment.
4) Action mechanism → Performs appropriate actions based on reasoning.

Eg. AI in medical diagnosis uses past cases to diagnose diseases

Learning-agent: A learning agent improves its performance over time. It consists of:

- Learning Element → Updates knowledge based on experience
- Performance Element → Decides actions based on current knowledge.
- Critic → Provides feedback by evaluating actions
- Problem generator → suggests new actions to improve learning

Eg.: A self-driving car learns to from traffic patterns & adjusts driving behaviour.

88 Convert the Following to predicates.

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

Car Available → Travels By Car (Anita)

→ Car Available → Travels By Bus (Anita)

b) Bus goes via Andheri & Goregaon.

Goes via (Bus, Andheri) ∧ goes via (Bus, Goregaon)

c) Car has puncture so is not available.

Puncture (car)

puncture (car) → ¬ Car available.

Will Anita travel via goregaon ? Use forward reasoning

From (c) :

Puncture (car) is true

From (a)

¬ Car available, we use ¬ Car available → Travels by bus (Anita)

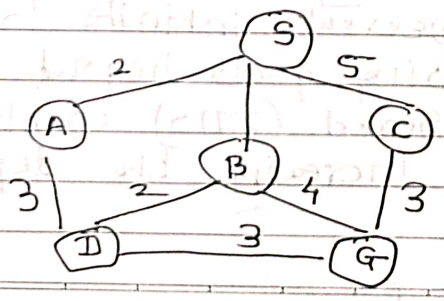
From (b)

goes via (Bus, goregaon)

Since Anita travels by bus, she follows this route

Thus, Anita will travel via Goregaon.

89 Find route from S to G wing BFS.



Teacher's Sign.: _____

To Find route From S to G using BFS, we systematically explore all nodes level by level sta from source node (S) until we reach destination nod

1) start at S
   queue = [S]
2) From S, we go to it neighbours: A, B, C.
   queue = [A, B, C]
3) Dequeue A & explore its neighbourss:
   queue = [B, C, D]
4) Dequeue B & explore it neighbues.
   queue = [C, D, G]
5) Dequeue C & queue neighbours
   queue = [D, G]
6) Dequeue D
   queque = [G]
7) Dequeue G
   As G is our destination, BFS stops here.

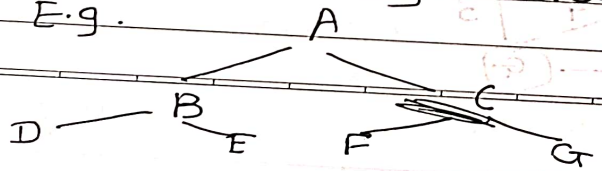
Bouk from S to G : S → B → G

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

Sol^n : Depth Limited Seatch (DLS) is an uninformed search algorithm that modifies DFS by introducing a depth limit L, preventing exploration beyond the predefined level. This prevents infinite loops in infinite graphs but risks missing goals beyond L.

Iterative Deepening Seatch (IDS) combines DLS with BFS by incrementally increasing the depth limit.

E.g.



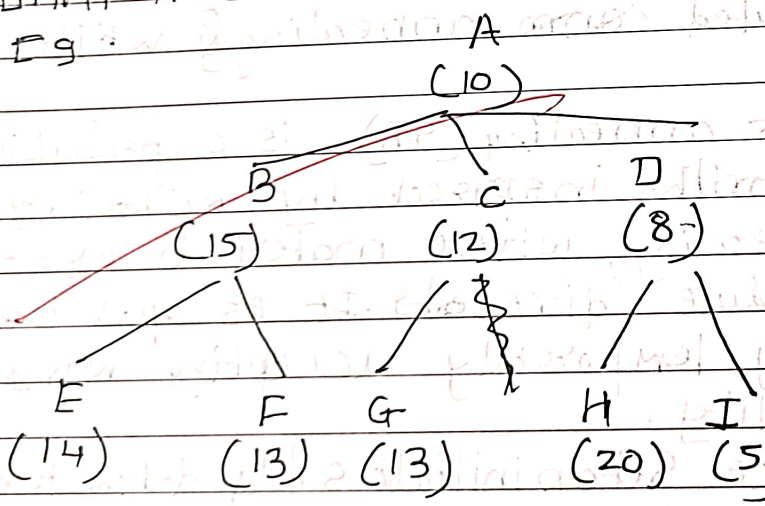
Teacher's Sign.: _____

Iteration 1 : Depth limit = 0
Nodes visited : A
Result : Goal not found.
Iteration 2 : L=1
Nodes visited A → B → C
goal not found.
Iteration 3 : L=2
Nodes visited : A → B → ~~C~~ D → E → C → F → G
goal G found at L=2.

12) Explain Hill climbing & its drawbacks in detail with example. Also state ~~simple~~ limitations of steepest-ascent hill climbing.

Sol^n: Hill climbing is a total search optimization algorithm which moves toward better neighboring solutions until it searches a peak.

Eg:



Steps: Start at root node A(10)
Compare its children B, C & D
move to child with highest value i.e B (15)
Repeat for B's children E & F
Terminate at E (14)

The algo stops at E(14) not reaching the goal (G).

Teacher's Sign.

Drawbacks:

1) Local Maxima: The algorithm greedily selects the best immediate child & can thus get stuck in a local maxima.
2) Plateaus - If siblings have equal values, the algorithm can't decide the next step & gets stuck.
3) Ridges - Narrow uphill paths require backtracking with hill climbing algorithm does not support.

Limitations of steepest- Ascent Hill Climbing:

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

13) Explain simulated ~~annn~~ annealing & write its algorithm.

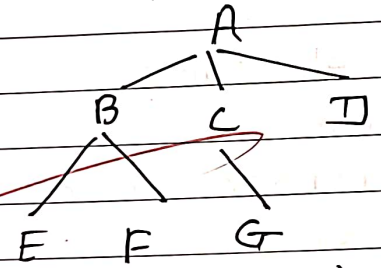
Soln: Simulated ~~e fate~~ annealing (SA) is a probabilistic optimization algorithm inspired by metallurgical process of annealing, where materials are heated & cooled to reduce direcets. It escapes the local optima by temporarily accepting worse soln with a probability.

Algo: 1) Initialize: Set an initial soln & define temp. T.
2) Repeat until stopping condition.
- Generate a new neighbour condition.
- Compute change in cost ($\Delta E = E_{new} - E_{current}$]
- If new soln is better i.e $\Delta E > 0$, accept it.
- If, worse, accept it with probability $P = e^{-\Delta E/T}$

Teacher's Sign.:

• Decrease temp. T (coding schedule)
3. Return best solution
Eg. Travelling salesman Problem:
Swap two cities in a route accept a longer route easly (high T) but reject it later (low T)

Explain A* algorithm with an example.
A* is a best first search algo used in path finding & graph traversal. It uses the following formula
$f(n) = g(n) + h(n)$
$g(n) \rightarrow$ cost to reach node n from start.
$h(n) \rightarrow$ heuristic estimates of cost to reach from goal to n.
Eg. Goal: G



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

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

2) Expand neighbours B, C, D

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

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

$f(D) = 4+7 = 11$

3) Choose lowest value that is C ($f(C) = 4$)

4) Expand neighbours of C : G

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

5) Goal reached at a with total cost 6

advantages : 1) Efficient for finding shortest paths in weighted graphs.

2) balances exploration by considering both $g(n)$ & $h(n)$.

15) Explain min-max algorithm & draw the game tree for Tic Tac Toe game.

Min-max algorithm is a decision rule used for finding the best possible move in two-player turn-based games such as Tic-Tac-Toe, chess & checkers

Algorithm:

1) Define Evaluation Function: Return +1 if X wins, -1 if O wins, 0 for a draw.

2) Check for terminate state:

If the board is a win/loss/draw state, return score.

3) Generate all possible moves.

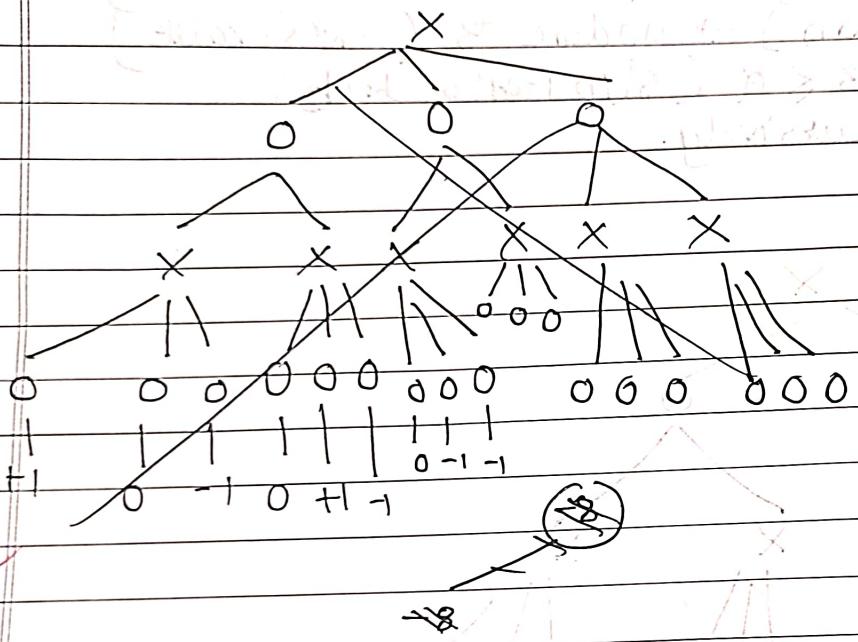
4) Apply Recursion (1) If its max's turn (X): call minmax on possible moves & choose the maximum score.

(2) If it's min's turn (O): Call minmax on possible moves & choose the minimum score.

5) Backpropagates Scores: Return the best score up the game tree.

6) Pick the best move (At root level): max chooses the move with highest score.

Game tree for Tic-Tac-Toe :



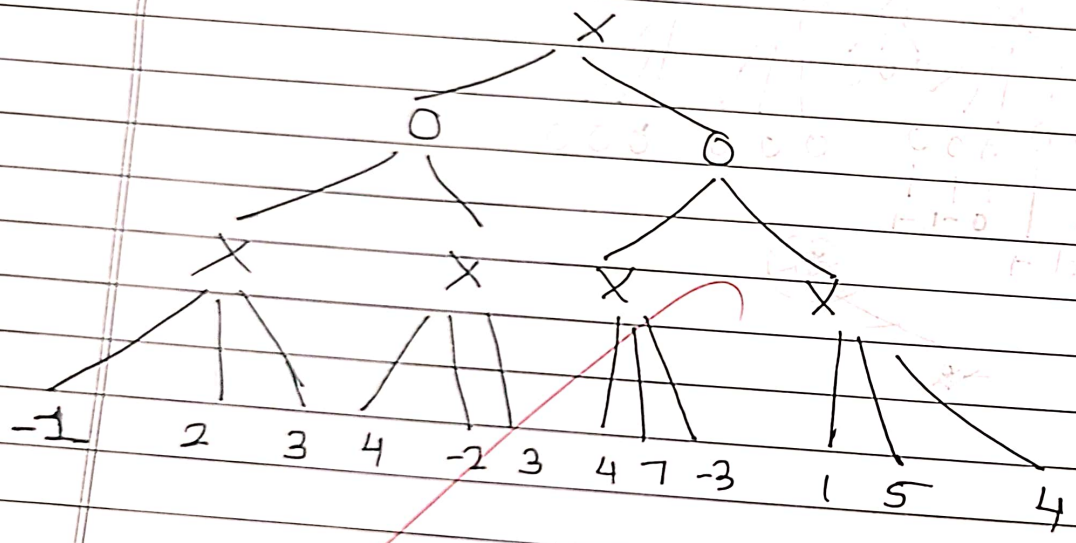
Teacher's Sign.: _____

16) Explain Alpha beta pruning algorithms for adversarial search with ~~explain~~ example.

Alpha-beta pruning is the optimized version of minimax algorith by skipping unnecessary branches reducing computation.

Algorithm: 1) Initialize $\alpha = -\infty$, $\beta = +\infty$.
2) Max (X's turn) → update α (highest value)
3) Min (O's turn) → update β (lowest value)
4) Prune if $\beta \leq \alpha$ (skip further checks)
5) Repeat recursively.

Eg.



* max selects the highest, min selects the lowest.
* If min finds the move worse than existing, we prune the branch.

Advantages: 1) Reduces time complexity from $O(b^d)$ to $O(b^{d/2})$
2) Faster pruning
3) Same optimal move as minmax.

Teacher's Sign.: _____

Page No.
Date

1) Explain WUMPUS World environment giving its PEAS & also the percept sequence generation.

* PEAS:

① P - • Maximize Rewards.
+1000 for collecting gold & exiting grid.
• minimize penalties:
-1000 for falling into a pit or being eaten by WUMPUS.
-1 point for each action taken.
-10 points for using an arrow.

②⑤ E - • Grid Layout (A*)
containing pits, WUMPUS, gold, wall, breeze
• Partially observable - agent can't see entire grid & must rely on sensory input.

③ A - • Move left, right, forward.
Grab - to collect gold, shoot (to eliminate WUMPUS)

④ S - Breeze: indicates pit is adjacent.
stench: indicates WUMPUS in adjacent cell.
glitter: indicates gold
bump: indicates a wall has been encountered.

* Percept Sequence: 1) Initial pos^n: Agent starts at (1,1)
2) Movement: As agent moves from one cell to other, it uses sensors to gather info about surrounding.
3) Creating percept sequence:
Each time the agent moves, it records its percepts as sequence. Eg. After moving to (1,2): [None, Breeze, None]
After moving to (2,1): [Stench, breeze, None]
These indicate no stench/glitter & ~~yes~~ yes to nearby dangers respectively. It continues as agent explores more
4) Decision Making: The agent uses these percepts sequences to make decisions about its next actions based on logical reasoning & inference from observations

18. Solve following cryptarithmatic problem.

SEND + MORE = MONEY

To solve this problem, we need to assign a unique digit (0-9) to each letter, such that the given equation holds true.

1) Set up the equation:

```
  S E N D
  M O R E
M O N E Y
```

2) Since M is the leading digit it must be 1.

$\therefore M = 1$

3) $D + E = Y$ (if carry = 0)

or $D + E = 10$ (if carry = 1)

Tens place: $N + R + \text{carry} = E$

Hundred's place

$E + O + \text{carry} = N$

Thousand's place: $S + 1 = 1 + \text{carry}$

$S + \text{carry} = 0$

$\therefore S = 9$ since there's no carry.

4) Try $E = 5$

If $D = 7$ then

$Y = 7 + 5 = 12$ (invalid)

$O = 2$ then

$Y = 7$

5) $Y = 7$, Assume $N = 8$

$N + R = E$

$8 + R = 5$ (impossible)

$\therefore \boxed{N = 6}$

final: $S = 9$, $E = 5$, $N = 6$, $D = 7$, $O = 0$, $R = 8$, $Y = 2$

```
  SEND          9 5 6 7
+ MORE    ---->  1 0 8 6
-------         -------
 MONEY    ---->  1 0 6 5 2
```

Page No.
Date

1) Explain Modus Ponen with example.

Modus ponen is a fundamental rule of interference in logic. It states that if $P \rightarrow Q$ is true P is true, then Q must also be true formula : $\frac{P \rightarrow Q, \; P}{Q}$

Eg. if it is raining, then it is soggy $(P \rightarrow Q)$
if it raining (P)
∴ it is soggy (Q)

2) Explain forward & backward chaining algo with example.

- Forward chaining : It is data driven, inference algo that starts with known facts & applies inference rules to derive new facts with until the goal is reached.

Fact : A, B
Rule : $A \rightarrow C$, $B \rightarrow D$, $C \wedge D \rightarrow I$
Goal I
Start with A & B
apply $A \rightarrow C$ to derive C
$B \rightarrow D$ to derive D
$C \wedge D \rightarrow I$ to derive I
The goal I is reached.

- Backward chaining : BC is a goal-driven, starts with goal & works backward to find the facts that support it start with goal.

Fact : A, B . Rule : $A \rightarrow C, B \rightarrow D, C \wedge D \rightarrow I$, goal I
Find the rule $C \wedge D \rightarrow I$
if C & D are true: use $A \rightarrow C$ since A is true, C is true
use $B \rightarrow D$ since B is true, D is true
C & D is true, I is true.
∴ conclusion : I is reached.

Page No.
Date

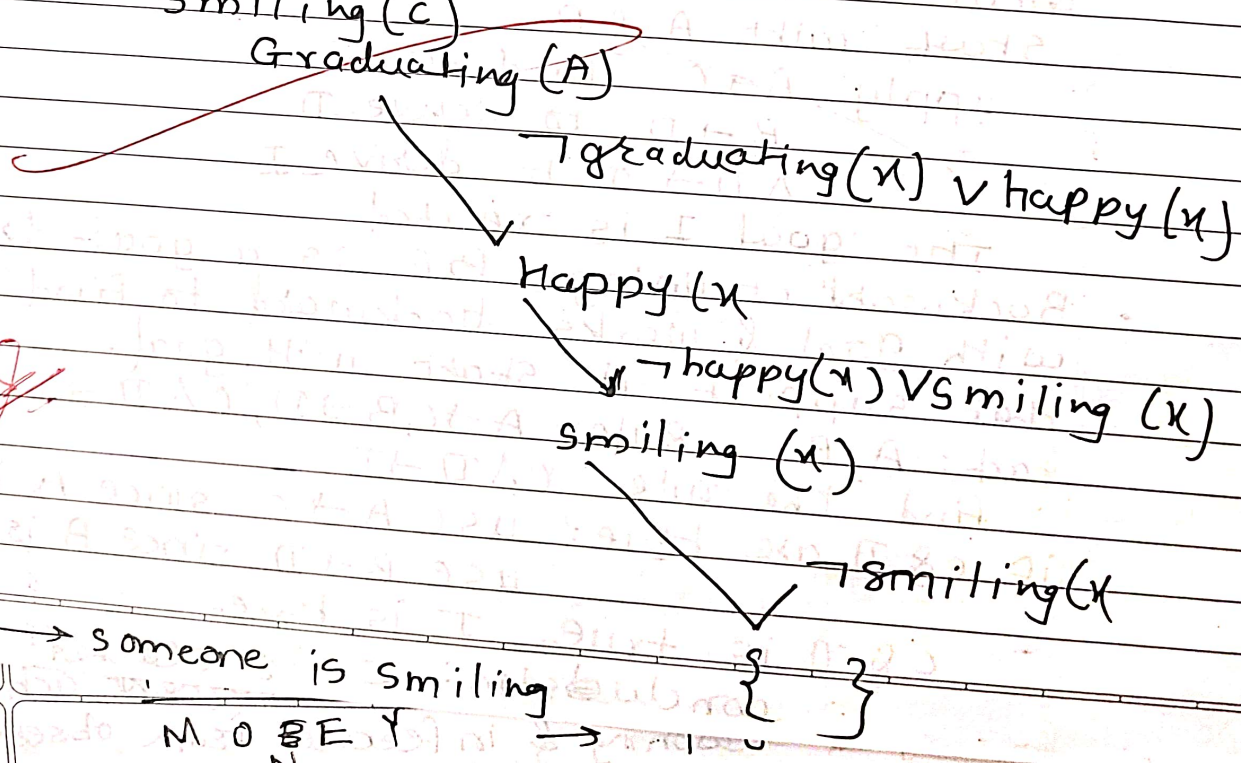
19) Consider the axioms:
All people who are graduating are happy
All happy people are smiling
Someone is graduating.
Explain 1: Represent these axioms in First Predicate logic:

1) $\forall x$ (Graduating $(x) \rightarrow$ Happy (x))
2) $\forall x$ (Happy $(x) \rightarrow$ smiling (x))
3) $\exists x$ (Graduating (x))

$C_1 = \{ \neg$ Graduating (x), Happy $(x)\}$
$C_2 = \{ \neg$ Happy (y), Smiling $(y)\}$
C_3 = Graduating (C)

Resolution between C_3 & C_1
Substitute C for x in L_1
From graduating (C)
Happy (C)

Resolving with C_2
Smiling (C)



→ someone is smiling

MOBEY →

AIDS-I Assignment No: 2

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

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

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

Solution:

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

1)Mean:

SUM OF ALL VALUES:= 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 +

88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = 1611

TOTAL VALUES: 20

Therefore, Mean = 1611 / 20 = 80.55

2)Median:

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

There are 20 numbers (even), so the median is the average of the 10th and 11th values.

10th = 81 , 11th = 82

Therefore, Median=(81+82)/2=81.5

3)Mode: Mode is the most frequent value present in the data ,here it is,76 (appeared thrice)

Therefore Mode=76

4)Interquartile Range:

To calculate the IQR, we need:

Q1 (First Quartile) = Median of the lower half

Q3 (Third Quartile) = Median of the upper half

Lower half (first 10 values): 59, 64, 66, 70, 76, 76, 76, 78, 79, 81

Q1 = Median of this half = average of 5th and 6th values = (76 + 76)/2 = 76

Upper half (last 10 values): 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Q3 = Median of this half = average of 5th and 6th values = (88 + 90)/2 = 89

IQR=Q3−Q1=89−76

Therefore, **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**

Solution:

1. Machine Learning for Kids

Target Audience:

- School students (typically aged 8–16)
- Teachers introducing AI/ML concepts
- Beginners with no prior programming experience

Use by the Target Audience:

- Students build ML models through an interactive, block-based interface (like Scratch).
- They classify text, numbers, or images using real datasets.
- Teachers use it in classrooms for hands-on AI projects without needing to code in-depth.

Benefits:

- Kid-friendly, visual and intuitive interface
- Integrated with Scratch and Python
- Good introduction to supervised learning
- Uses real AI models (powered by IBM Watson)

Drawbacks:

- Limited complexity for advanced learners
- Small-scale dataset usage
- Requires internet access and IBM Watson integration

Predictive or Descriptive Analytic:

Predictive Analytic:

Because it helps kids train models to predict outcomes (e.g., classify images/text based on learned patterns).

Learning Type:

Supervised Learning

Students label training data (e.g., “this is a cat”, “this is a dog”) and train the model, which is classic supervised learning.

2. Teachable Machine

Target Audience:

- Beginners, hobbyists, educators, students
- Anyone interested in AI/ML without needing to code

Use by the Target Audience:

- Users can train models by recording examples using webcam, audio, or images.
- Drag-and-drop interface allows creating and training models in the browser.
- Models can be exported to TensorFlow or embedded in websites.

Benefits:

- No coding required
- Quick setup and training
- Supports real-time video/audio classification
- Allows model export for web or apps

Drawbacks:

- Less control over model architecture
- Not suitable for large or complex datasets
- May not generalize well with limited training data

Predictive or Descriptive Analytic:

Predictive Analytic:

> It predicts categories based on real-time input (e.g., webcam detects "Class A" or "Class B") using trained data.

Learning Type:

Supervised Learning

> Users label their data during training (e.g., “this is gesture 1”, “this is gesture 2”), so the tool learns through labeled input.

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.**

Solution:

Data visualizations are powerful tools for conveying complex information succinctly. However, when misused or poorly designed, they can mislead audiences and propagate misinformation. A pertinent example of this

occurred in July 2024, involving misleading social media posts about sexually transmitted diseases (STDs) in Houston, Texas.Reuters

Case Study: Misleading STD Statistics in Houston

In July 2024, social media platforms were abuzz with alarming claims that over 40,000 individuals in Houston had tested positive for STDs within a single week. These assertions were accompanied by screenshots of data tables listing various STDs, including chlamydia, gonorrhea, syphilis, and HIV, along with corresponding figures. The presentation of this data, without proper context, led many to believe there was a sudden and massive outbreak of STDs in Houston.

How the Data Visualization Was Misleading:

1. Lack of Context: The figures presented were not exclusive to Houston but represented the total number of STD tests conducted across the entire state of Texas. This crucial detail was omitted, leading viewers to draw incorrect conclusions about the health situation in Houston.Reuters

2. Misinterpretation of Data: The numbers in the table reflected the total tests administered, encompassing both positive and negative results. However, the accompanying captions and the way the data was framed suggested that all the figures represented positive cases, which was not the case.

3. Visual Presentation: The data was displayed in a straightforward table format without explanatory notes or sources. This lack of clarity made it easy for misinformation to spread, as viewers had no immediate way to verify the authenticity or scope of the data.

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

Pima Indians Diabetes Database

Solution:

Ans. Model used: Naive Bayes

Step 1. Importing required libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from imblearn.over_sampling import SMOTE
```

Step 2. Loading the dataset

```
from google.colab import files
uploaded = files.upload()
```

Step 3. Performing data preprocessing

```
df.isnull().sum()
```

	0
Pregnancies	0
Glucose	0
BloodPressure	0
SkinThickness	0
Insulin	0
BMI	0
DiabetesPedigreeFunction	0
Age	0
Outcome	0

Since here there are no null values we can skip imputation

Step 4. Splitting the dataset

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']

# First split: Train (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

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

In this step we are splitting the dataset into:
70% Train
20% Validation
10% Test

But since train_test_split only lets us split into two parts at a time, we do it in two stages:

Step 1:

```
X = df.drop('Outcome', axis=1)
y = df['Outcome']
```

Here we are separating the features (X) and target label (y).

Step 2: First Split — 70% Train, 30% Temp

```
# First split: Train (70%) and Temp (30%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)
```

This gives:

- X_train, y_train → 70% of the data (for training)
- X_temp, y_temp → remaining 30% (to be further split into validation and test)

Stratify is used to ensure that the proportion of class labels (0s and 1s) is the same in all splits.

Step 3: Second Split — 20% Validation, 10% Test

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

Here we are splitting X_temp (30%) into:

- X_val, y_val → 20% of original data
- X_test, y_test → 10% of original data

Step 5. Using SMOTE for class imbalance

```
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

SMOTE (Synthetic Minority Over-sampling Technique) is used to fix class imbalance by creating synthetic samples for the minority class

In the above given line of code,

fit_resample() applies SMOTE to the training set which balances the class distribution.
X_train_res, y_train_res now contain:
Original majority class samples and Synthetic minority class samples

Step 6. Feature scaling

```
scaler = StandardScaler()
X_train_res = scaler.fit_transform(X_train_res)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

Feature scaling standardizes the values of your features so they all have:
Mean = 0 and Standard Deviation = 1

In the above code,

We fit the scaler on the training data, then transform the training data using those values.Finally we transform validation and test data using the same scaler.

Step 7. Training the Naive Bayes model

```
model = GaussianNB()
model.fit(X_train_res, y_train_res)
```

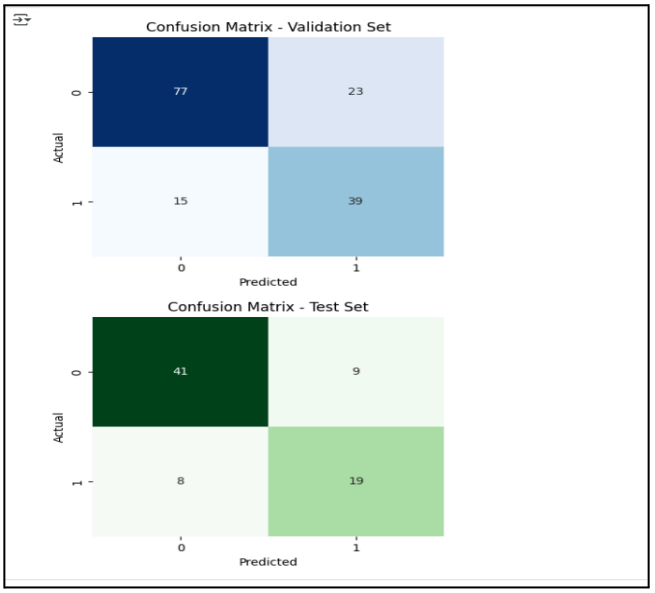
```
GaussianNB
GaussianNB()
```

This creates a Naive Bayes model using the GaussianNB class.
GaussianNB is used when the features are continuous and assumed to follow a normal (Gaussian) distribution — which is common for numeric data like BMI, glucose, etc.

Step 8. Evaluating the model

```python
# On Validation Set
y_val_pred = model.predict(X_val)
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print(classification_report(y_val, y_val_pred))
print("Confusion Matrix:\n", confusion_matrix(y_val, y_val_pred))

# On Test Set
y_test_pred = model.predict(X_test)
print("\nTest Accuracy:", accuracy_score(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_test_pred))
```

```
Validation Accuracy: 0.7532467532467533
              precision    recall  f1-score   support

           0       0.84      0.77      0.80       100
           1       0.63      0.72      0.67        54

    accuracy                           0.75       154
   macro avg       0.73      0.75      0.74       154
weighted avg       0.76      0.75      0.76       154

Confusion Matrix:
 [[77 23]
 [15 39]]

Test Accuracy: 0.7792207792207793
              precision    recall  f1-score   support

           0       0.84      0.82      0.83        50
           1       0.68      0.70      0.69        27

    accuracy                           0.78        77
   macro avg       0.76      0.76      0.76        77
weighted avg       0.78      0.78      0.78        77

Confusion Matrix:
 [[41  9]
 [ 8 19]]
```



Validation Accuracy: 75.3%
Test Accuracy: 77.9%
This shows good generalization and consistent performance.

Recall for Class 1 (Diabetic):
72% (Validation), 70% (Test)
Model is effectively identifying most actual diabetic cases.

Precision for Class 1 (Diabetic):
63% (Validation), 68% (Test)
Around two-thirds of diabetic predictions are correct.

Confusion Matrix indicates:

- High true positive rate for both classes.
- Moderate number of false positives and false negatives.

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**

Solution:

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:
  - R^2: proportion of variance explained
  - Adjusted R^2: 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}
R^2 Score: 0.6552
Adjusted R^2 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 R^2 shows performance after accounting for model complexity.

Why we May Not Reach $R^2 > 0.99$

- 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).

Solution:

1) **Key Features of the Wine Quality Dataset**

Feature Name	Description
fixed acidity	Non-volatile acids (e.g., tartaric acid). Contributes to wine's crispness and structure.
volatile acidity	Acetic acid (vinegar). High levels give an unpleasant smell.
citric acid	Adds flavor and freshness. Too much makes wine taste sour.
residual sugar	Remaining sugar after fermentation. Affects sweetness.
chlorides	Salt content. Affects taste and preservation.
free sulfur dioxide	Prevents oxidation and microbial growth. Important for shelf life.
total sulfur dioxide	Sum of free and bound SO_2. Regulates stability but may harm flavor if too high.
density	Related to sugar/alcohol content. Helps in wine classification.
pH	Measures acidity/basicity. Influences taste and microbial stability.
sulphates	Enhances shelf life. Also affects bitterness and mouthfeel.
alcohol	Strongly correlated with perceived wine quality.
quality (target)	Human-rated quality score of the wine (integer from 0 to 10).

2) **Importance of Features in Predicting Wine Quality:**

Feature	Importance Explanation
alcohol	Highly influential. Generally, higher alcohol is associated with better-rated wines.
volatile acidity	Negative impact. High levels lead to lower quality scores.
sulphates	Positive impact on quality due to preservation effect.
citric acid	Adds freshness and body; affects taste positively.
density	Indirectly reflects alcohol/sugar content; moderate impact.
ph	Impacts acidity and aging ability. Balanced pH is preferred.

3) **Imputation Techniques: Advantages & Disadvantages**

Technique	Advantages	Disadvantages
Mean/Median Imputation	Simple, fast, preserves dataset size	Distorts variance, poor for skewed data
Mode Imputation	Best for categorical data	Not suitable for continuous data
KNN Imputation	Considers feature similarity	Computationally expensive, sensitive to outliers
Regression Imputation	Considers inter-feature relationships	Can introduce bias, assumes linear relationships
MICE (Multiple Imputation)	Robust, works well with multivariate data	Complex and resource-intensive

4) <u>Handling Missing Data (Feature Engineering Process)</u>

- In this dataset, after inspection, there were **no missing values**.

- However, if missing values were present, we would apply **imputation techniques** during preprocessing.