Krushikesh Shelar D15-C Roll No:51

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:

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:

- 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.

```
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.

`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.

```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)
```

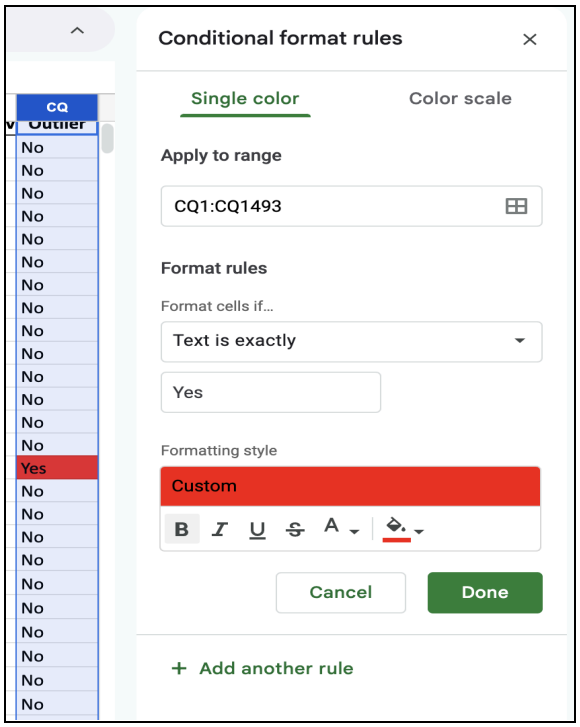
```
Current     123
Historic    131
Total       150
dtype: int64
Current    -11.406360
Historic   -11.943636
Total      -28.000000
dtype: float64 Current     45.677267
Historic    30.572727
Total       60.000000
dtype: float64
```

```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.
```




```
[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.

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
```

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: Krushikesh Shelar
Class: D15C
Roll No.: 51

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

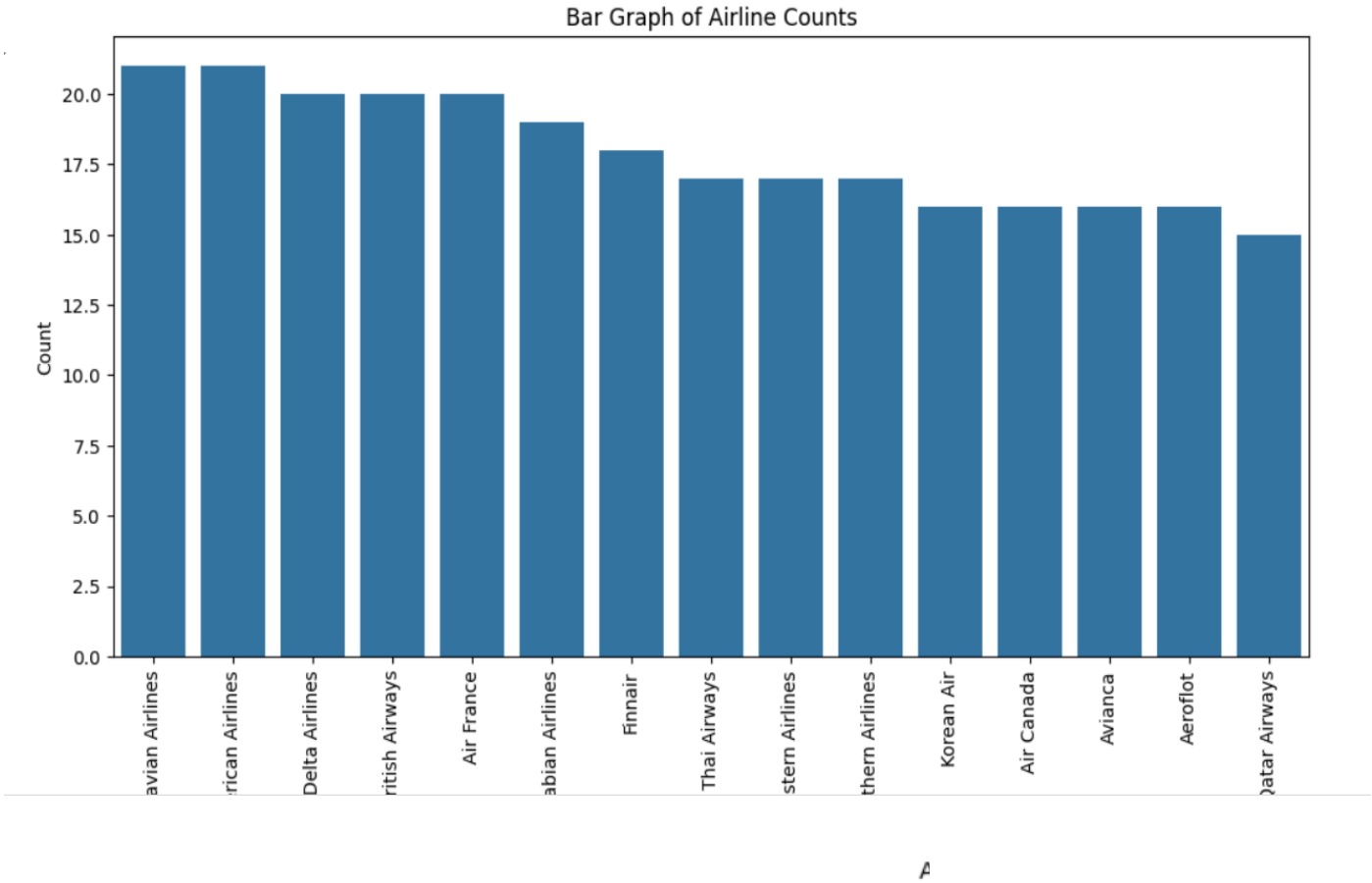
- **Bar Graph:** A bar graph is used to display categorical data with rectangular bars. The length of each bar represents the frequency of a category.
- **Contingency Table:** A table that summarizes the frequency distribution of two categorical variables, helping to analyze relationships between them.

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

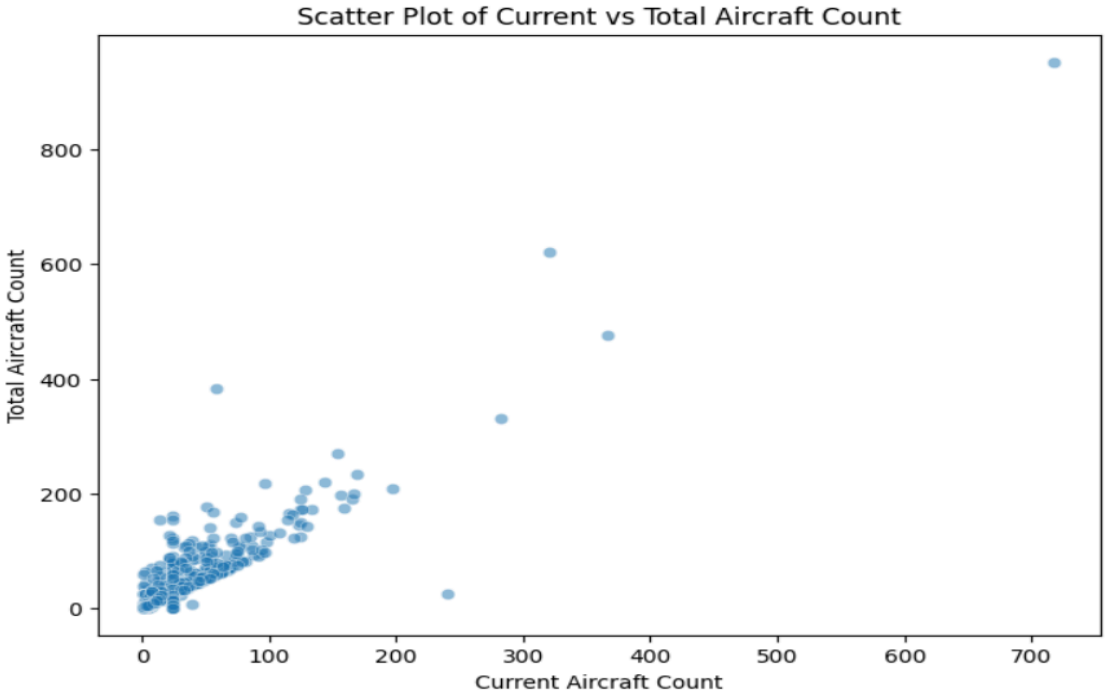
- **Scatter Plot:** A scatter plot visualizes the relationship between two numerical variables by plotting data points on an X-Y coordinate plane.
- **Box Plot:** A box plot displays the distribution of numerical data using quartiles and highlights outliers.
- **Heatmap:** A heatmap visually represents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

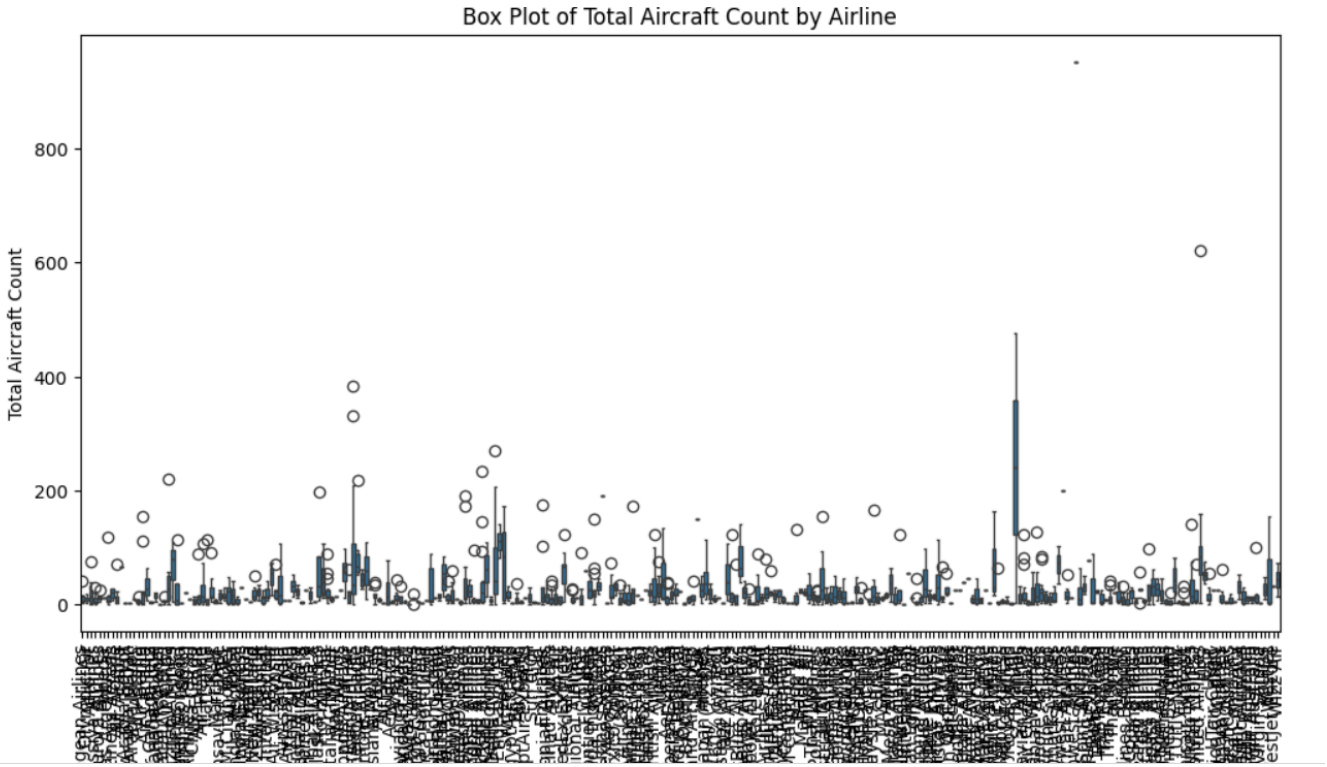
Terms

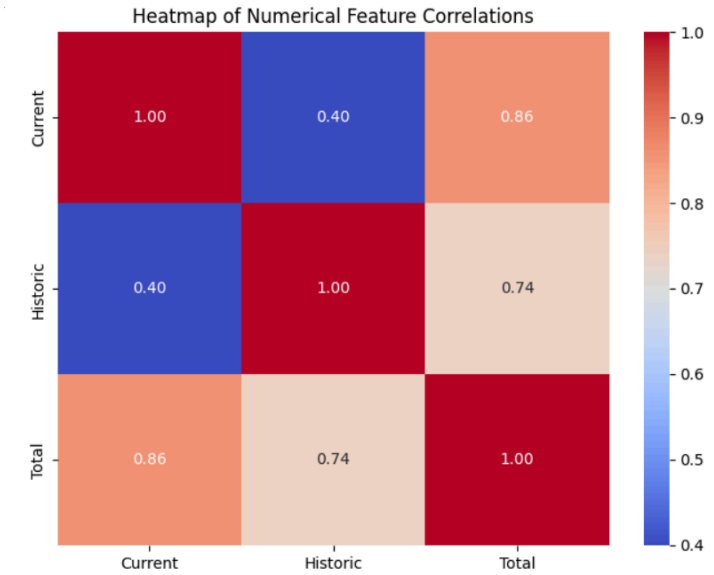
- **Numerical Data:** Data representing continuous quantities (e.g., aircraft count).
- **Quartiles:** Divides the dataset into four equal parts.
- **Correlation:** A measure of the strength of the relationship between two variables (values range from -1 to +1).

```
# 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

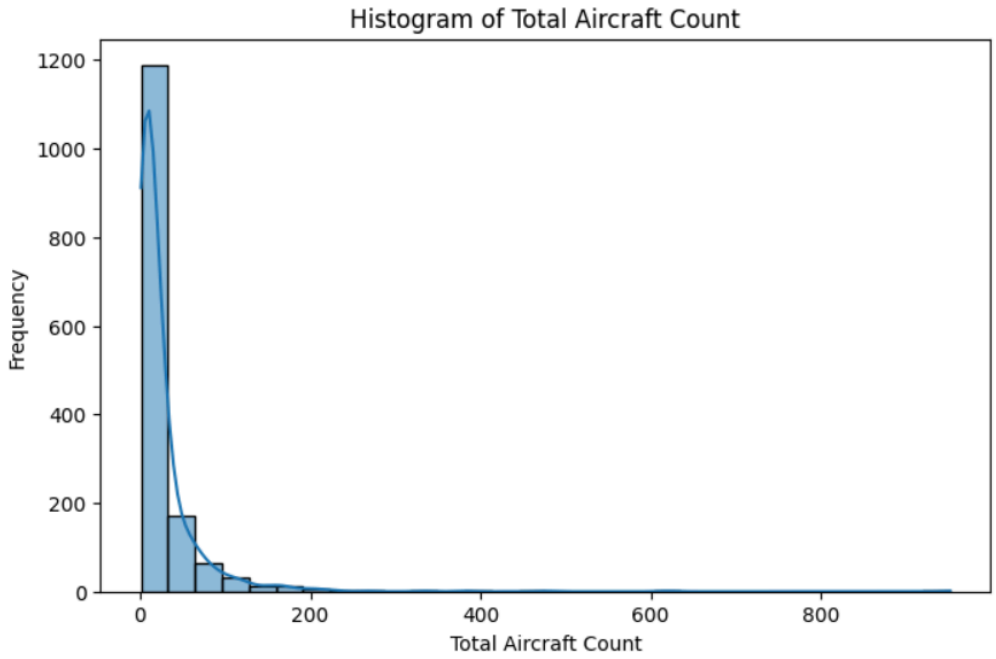
- **Histogram:** A histogram is used to represent the frequency distribution of numerical data by dividing it into bins.
- **Normalized Histogram:** Represents the probability distribution, where the total area sums to 1.

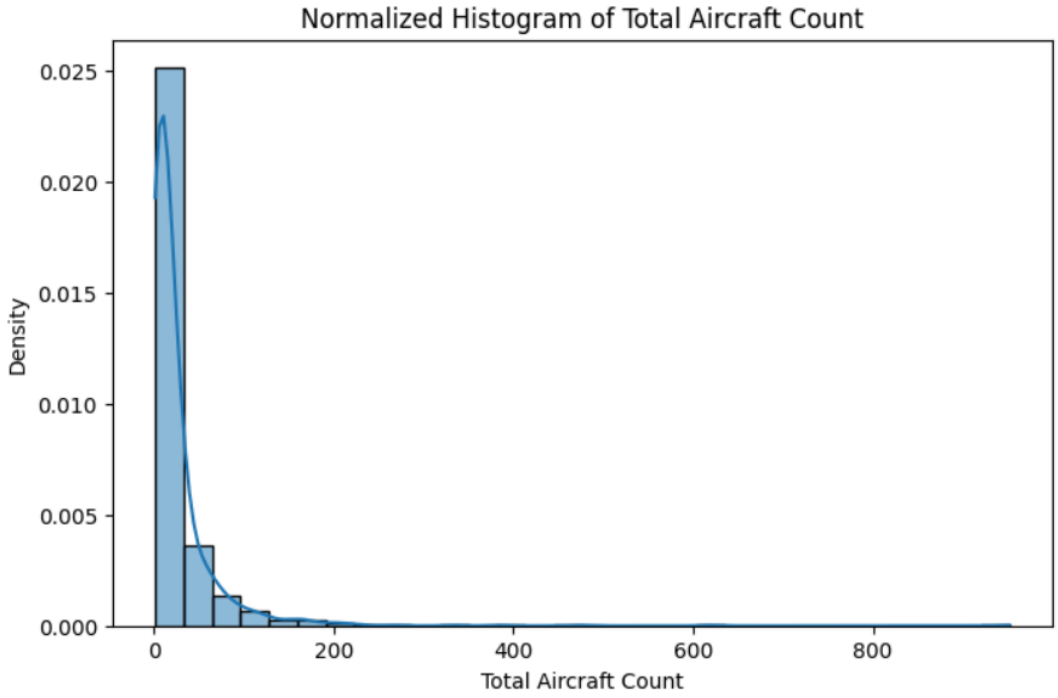
Terms

- **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

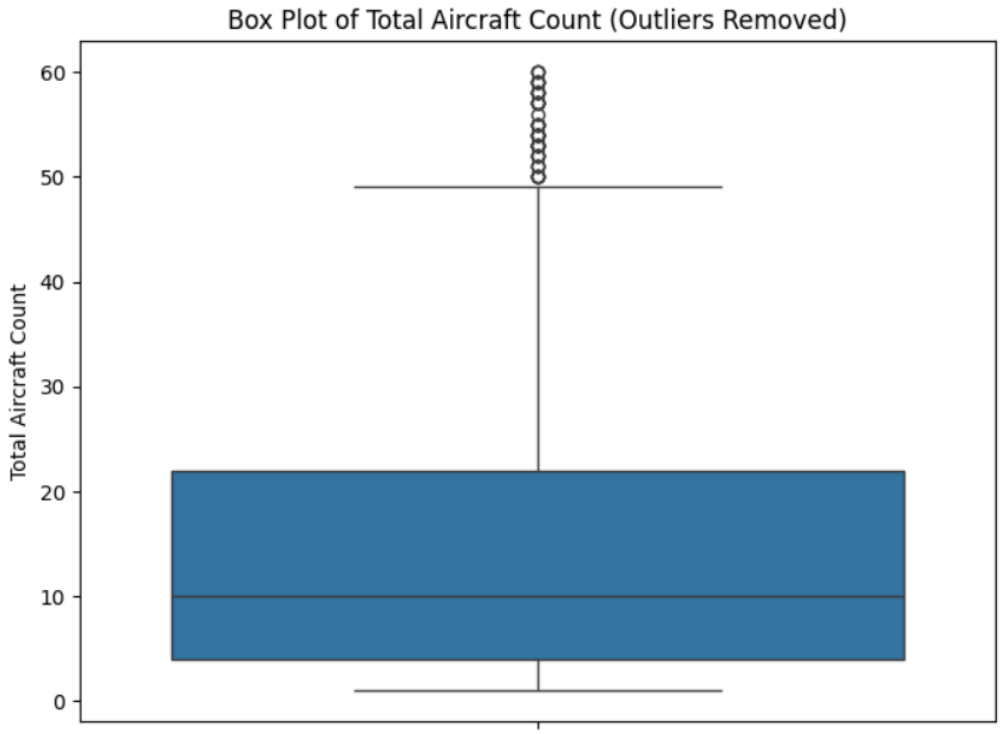
- **Outliers:** Data points that deviate significantly from the rest of the dataset.
- **Box Plot:** Identifies outliers using quartiles and whiskers.
- **IQR Method:** Detects outliers using the Interquartile Range (IQR) as follows:
    - Lower Bound = Q1 - (1.5 * IQR)
    - Upper Bound = Q3 + (1.5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

Name: Krushikesh Shelar Class: D15C Roll No: 51

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.

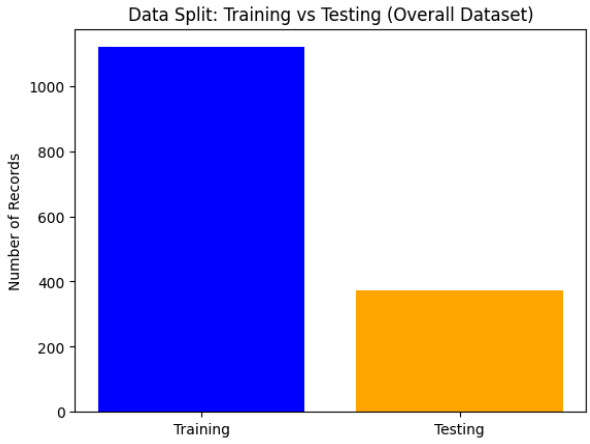
In this experiment, we partitioned a dataset of fleet data into a training set (75%) and a test set (25%) and validated this partitioning using a two-sample Z-test. The dataset consists of 1,492 records with features such as vehicle ID, fleet type, engine performance, and maintenance history.

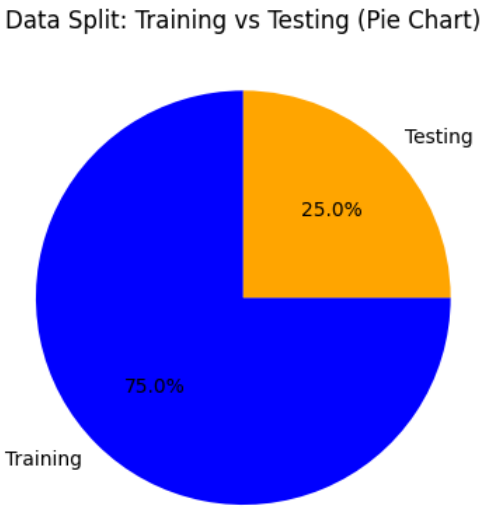
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
df = pd.read_csv(list(uploaded.keys())[0])
train_data, test_data = train_test_split(df, test_size=0.25, random_state=42)
labels = ['Training', 'Testing']
sizes = [len(train_data), len(test_data)]
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.

Name: Krushikesh Shelar Class: D15C Roll No: 51





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	HasDepend
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)
```

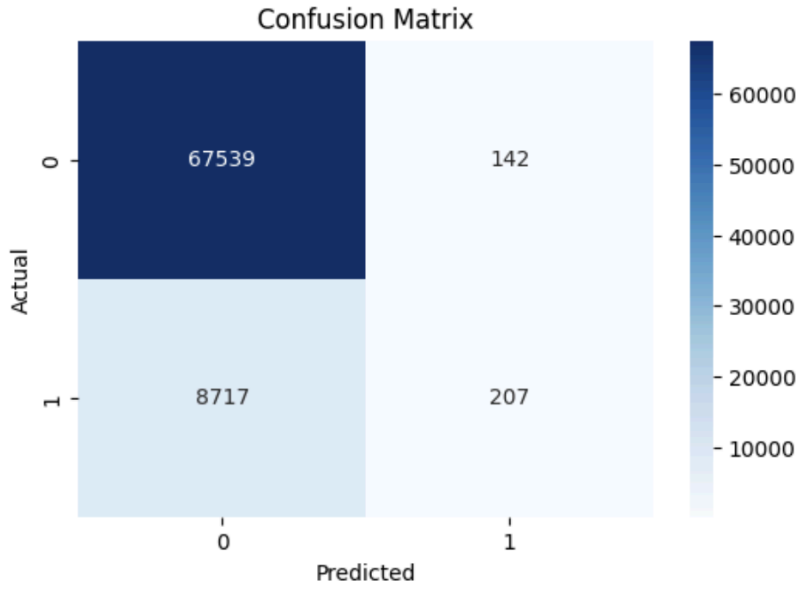
3. **Model Training:**
   Train the logistic regression model on the scaled training data.

```
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.

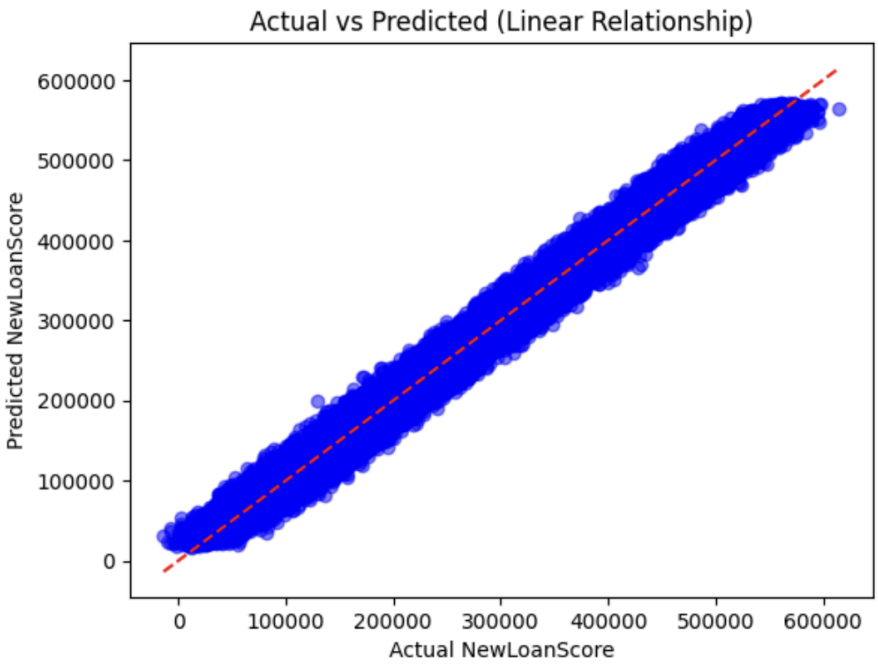
```
# Fit model
model = LinearRegression()
model.fit(X_train, y_train)

# 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.

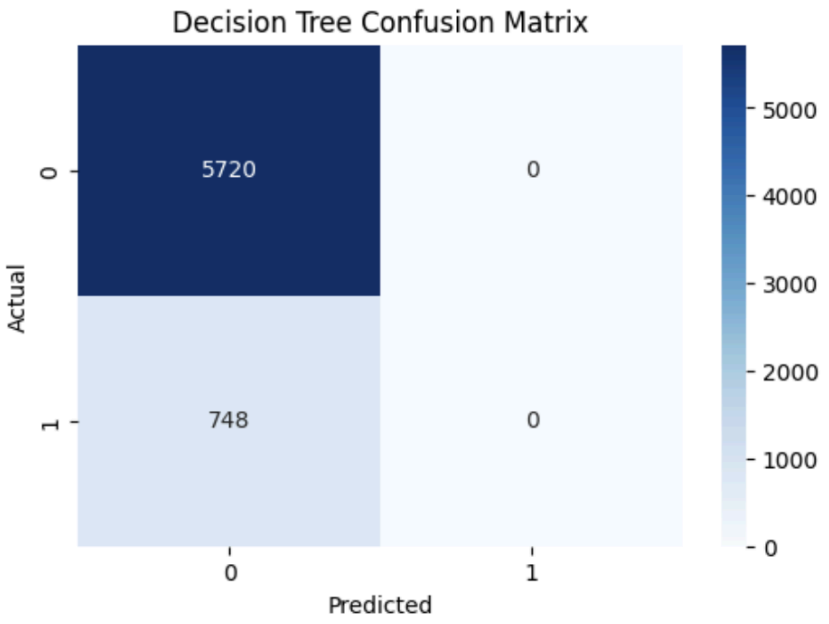
```
from sklearn.neighbors import KNeighborsClassifier

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))
```

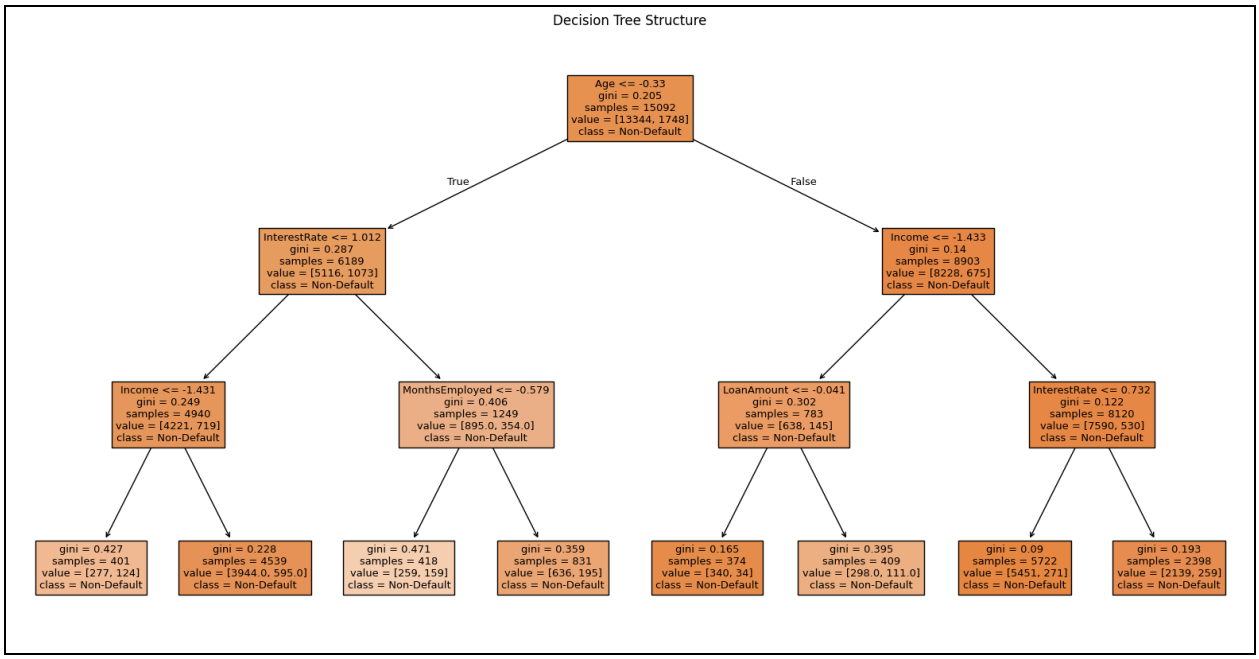
```
KNN Accuracy: 0.8749226963512677
KNN Classification Report:
              precision    recall  f1-score   support

         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()
```

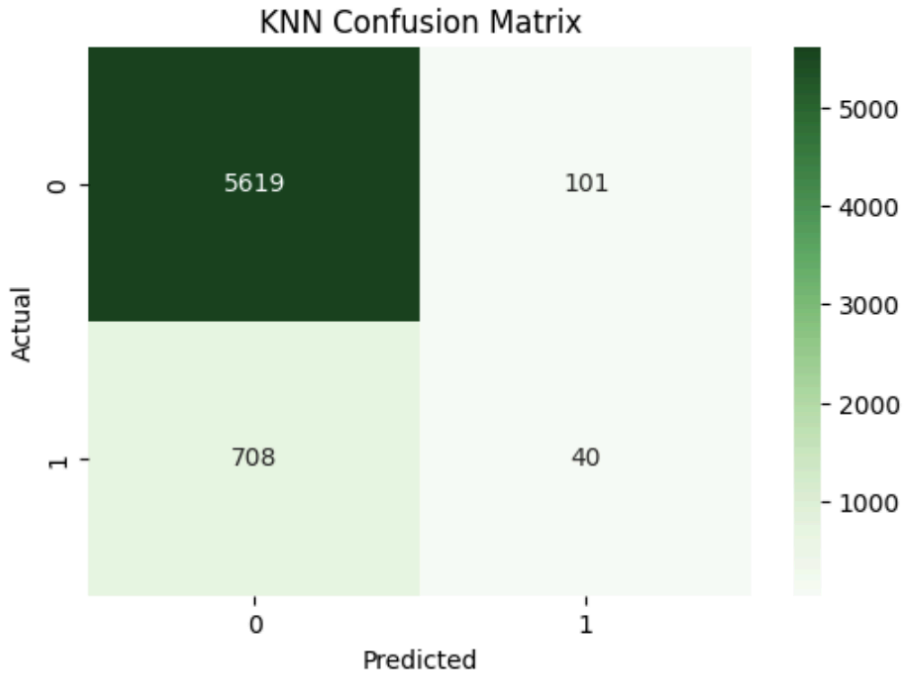





```
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
import matplotlib.pyplot as plt
r = export_text(dt_clf, feature_names=features)
print("Decision Tree Rules:\n", r)
```

```
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:

Agglomerative (Bottom-Up) – Most common

- 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.

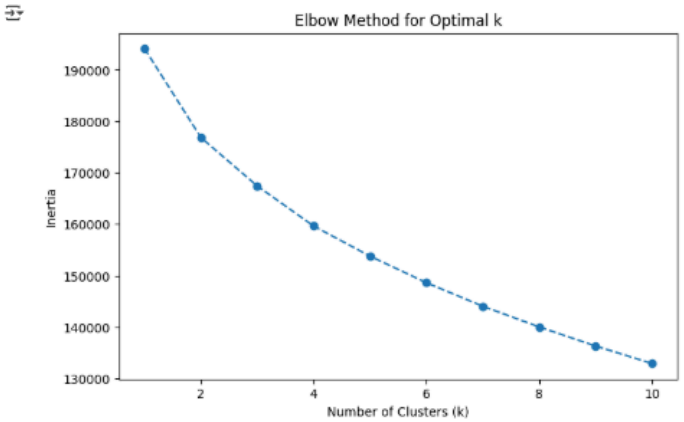
```
numerical_features = ['Age', 'Income', 'LoanAmount', 'CreditScore', 'MonthsEmployed',
                      'NumCreditLines', 'InterestRate', 'LoanTerm', 'DTIRatio']

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()
```

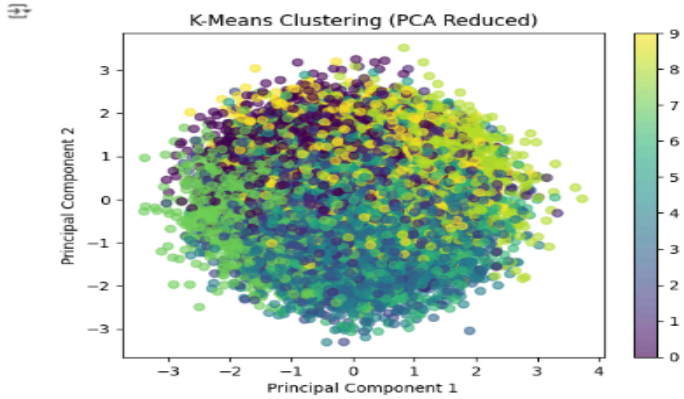


4)Applies K-Means clustering to the dataset using the chosen optimal number of clusters.

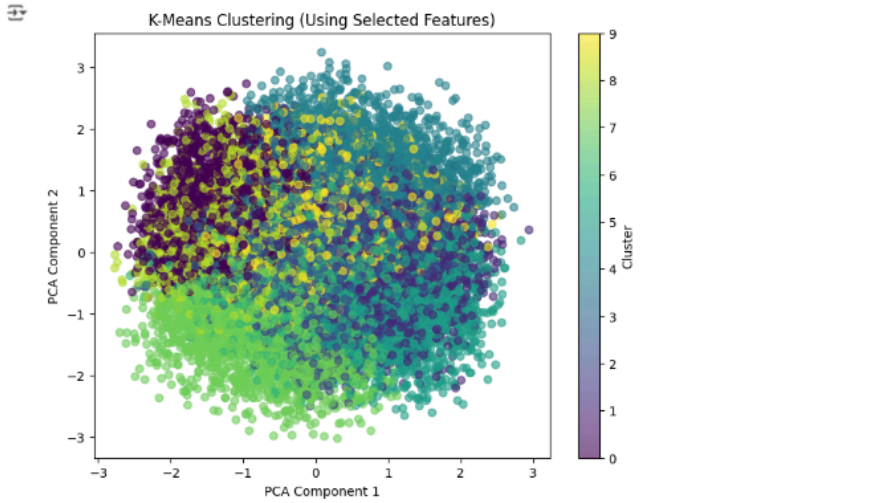
```
optimal_k = 10
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
data['Cluster'] = kmeans.fit_predict(X_scaled)
```

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.

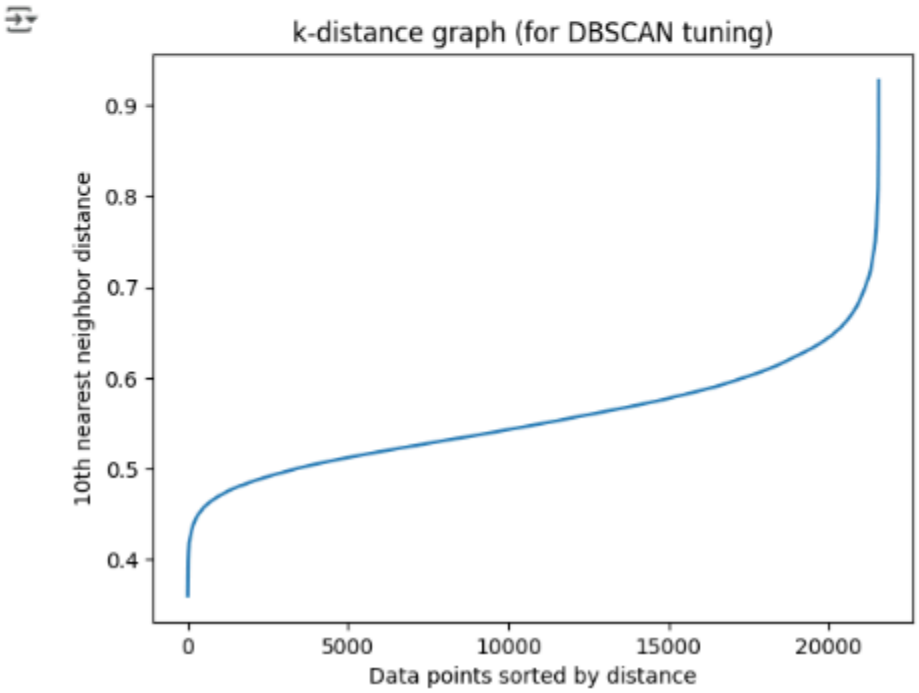
```
[ ] from sklearn.metrics import silhouette_score
    import numpy as np
    from sklearn.cluster import KMeans

    # 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.

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()
```



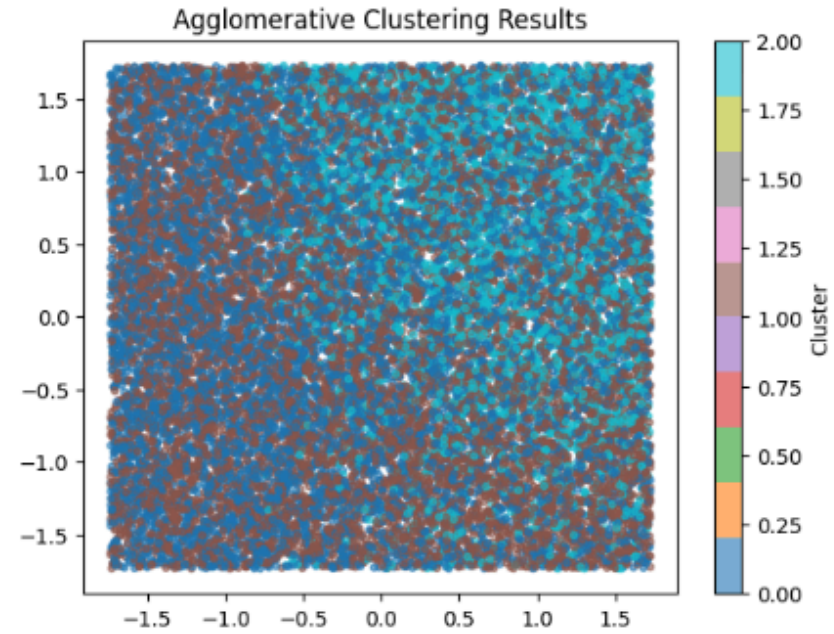
10)DBSCAN Clustering (Tuned eps=0.58)

- 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.

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}')
```

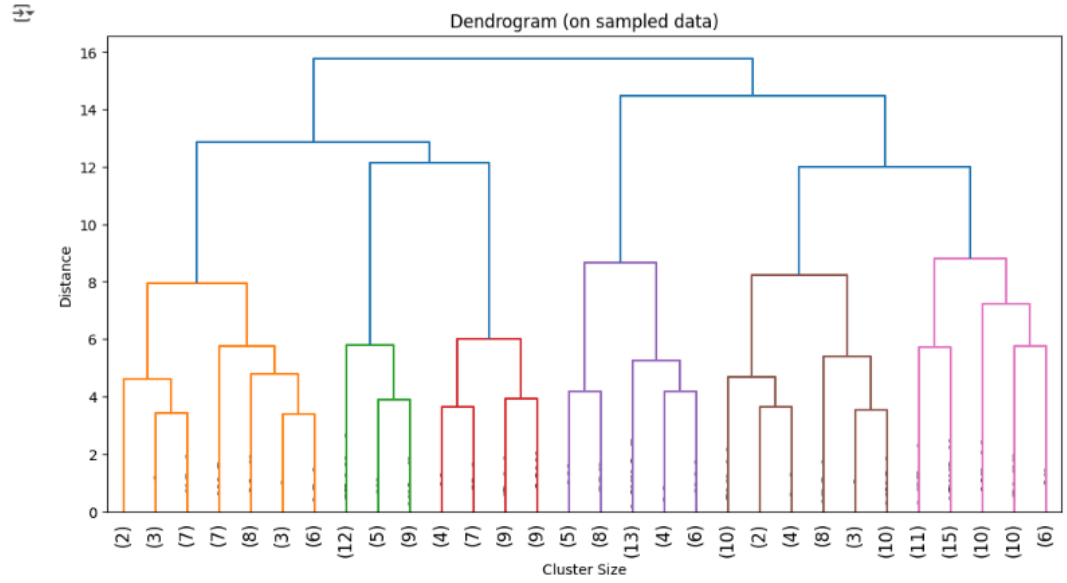



```
Agglomerative Clustering Silhouette Score: 0.0760
```

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.

```
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.

```
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)
```

```
# 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.

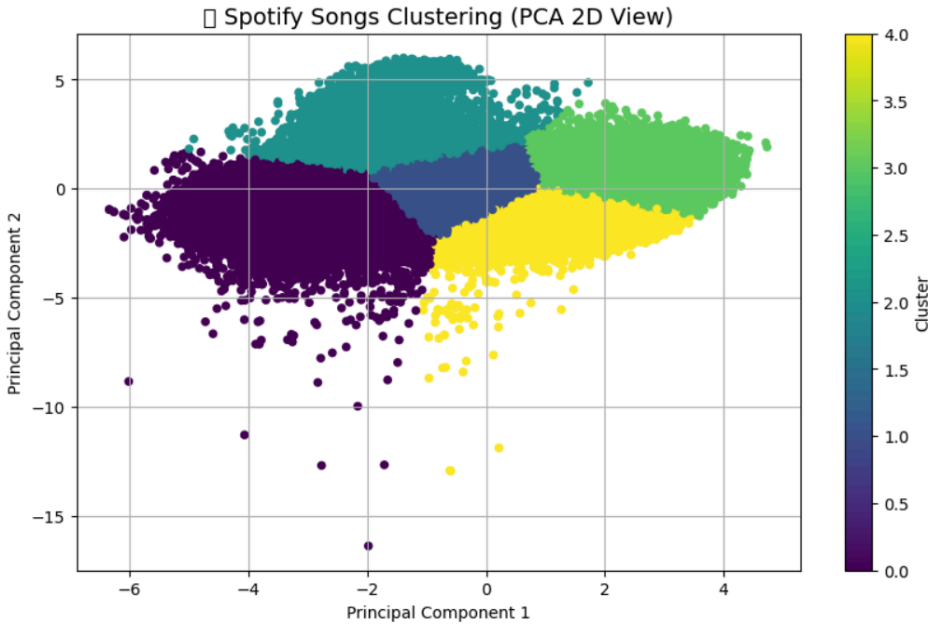
Process:

- 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:

1) **What is Apache Spark and it works?**

Ans:
Apache Spark is an open-source, distributed computing system designed for big data processing and analytics. It's known for its speed, scalability, and ease of use, especially for large-scale data processing tasks like machine learning, stream processing, and SQL queries.

Language Support: Scala (native), Java, Python (PySpark), R, and SQL.

At a high level, Spark works in four main stages:

1. Driver Program Starts
The user writes an application using Spark APIs (in Python, Scala, etc.).
The Driver Program is the heart of Spark, where the main control flow runs.
It converts user code into DAGs (Directed Acyclic Graphs) of stages.

2. Cluster Manager Allocates Resources
Spark uses a Cluster Manager (e.g., YARN, Mesos, or its own standalone manager) to allocate resources.
Executors (worker nodes) are launched across the cluster.

3. Tasks Are Sent to Executors
The DAG Scheduler breaks the job into tasks.
These tasks are sent to the Executors, which actually do the work like reading data, running transformations, and writing output.

4. Executors Run Tasks and Return Results
Executors process data in memory for fast performance.
Intermediate results are cached when possible.
Final results are returned to the Driver or written to storage.

Use Cases

Processing logs or real-time event data.
Running ML models on large datasets.
ETL (Extract, Transform, Load) pipelines.
Analyzing huge volumes of structured or unstructured data.

2) How data exploration done in Apache spark? Explain steps.

Ans:

Data exploration in **Apache Spark**—especially using **PySpark** (Python API for Spark)—is a crucial step in any big data analytics or machine learning pipeline. It helps you understand the structure, quality, and patterns in your dataset.

Steps for Data Exploration in Apache Spark:

1. Loading the Data

The first step in data exploration is importing the data into Spark from sources such as CSV files, JSON, Parquet, databases, or cloud storage systems. Apache Spark provides various APIs to connect to and load data from different formats and sources efficiently.

2. Understanding the Schema

Once the data is loaded, the schema (structure) of the dataset is examined. This includes:

- Column names
- Data types (e.g., Integer, String, Date)
- Nullable fields

Understanding the schema helps identify incorrect or inconsistent data types and ensures the data is correctly interpreted.

3. Viewing Sample Records

Exploratory analysis begins with viewing a few records from the dataset. This helps in gaining a quick overview of:

- The kind of data stored
- Formatting or entry errors
- Presence of special characters or missing values

4. Generating Summary Statistics

Statistical summaries of numerical columns are generated to understand the distribution and spread of the data. These statistics include:

- Count

- Mean
- Minimum and Maximum values
- Standard deviation

This step is essential for detecting outliers and understanding the scale of data.

5. Identifying Missing or Null Values

It is important to detect and assess missing or null values in the dataset. This helps in deciding whether to remove, replace, or impute missing data before further analysis.

6. Analyzing Value Distributions

The distribution of values within categorical or numerical columns is analyzed. This includes grouping data based on categories and counting their occurrences. It helps to:
Identify class imbalances
Understand the frequency of certain values

7. Examining Relationships and Correlations

In this step, the relationships between numerical variables are explored. Correlation analysis is performed to measure how strongly two variables are related. This insight is useful for feature selection in machine learning.

8. Filtering and Conditional Queries

Subsets of data are explored by applying filters and conditions. This helps focus on specific segments or conditions, such as high-income individuals or records with specific attributes.

9. Sampling for Detailed Analysis

If the dataset is very large, a smaller representative sample is taken for detailed analysis or visualization. This reduces computational cost and allows easier inspection.

10. Preparing for Visualization or Modeling

Although Spark itself is not primarily used for visualizations, after exploration, data is often summarized or exported for plotting using external tools. The insights gained during exploration guide the next steps in data cleaning, transformation, or modeling.

Conclusions:

Apache Spark is a fast and general-purpose distributed computing system designed for large-scale data processing. It utilizes a driver-executor architecture and performs in-memory computations to achieve high performance. Spark supports various APIs, including RDDs, DataFrames, and Datasets, enabling efficient data manipulation and processing across multiple programming languages.

Name: Krushikesh Shelar **Academic Year:2024-2025**

Division: D15C **Roll No: 51**

Experiment No 10

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

Theory:

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

Answer:

In Data processing, we generally deal with two major types: 1)Batch processing and 2)Stream processing.

- Batch Data Processing refers to collecting data over a period of time and then processing it all at once. Think of it like baking cookies: you prepare a whole batch and then put it in the oven. It's great when you don't need real-time insights and can wait for results.

- Stream Data Processing (also called real-time processing) deals with data that flows in continuously like sensor data, social media updates, or payment transactions. Instead of waiting for a large chunk, stream processing handles data piece by piece, as it comes in.

Streaming is the process of analyzing data in motion. It's especially useful when timely insights matter like fraud detection, server health monitoring, or stock market analytics.

2. How Does Data Streaming Take Place Using Apache Spark?

Answer:

Apache Spark provides a module called Spark Streaming (and its newer version called Structured Streaming) to process real-time data streams.

Working:

- Data Source: Data is continuously received from sources like Kafka, socket connections, or files being updated in real-time.

- Spark Streaming Engine: Spark divides the incoming data into small, manageable batches (micro-batches). Even though it's called streaming, it's actually processing in very small intervals like every second or every few seconds.

- Transformations and Actions: Just like in batch mode, you can apply filtering, grouping, or aggregation logic to the data in each micro-batch.

- Output Sink: The results can be saved or pushed to dashboards, databases, or alerts systems giving near-instant updates based on the incoming data.

Name: Krushikesh Shelar **Academic Year:2024-2025**

Division: D15C **Roll No: 51**

Spark special is that the same programming model can be used for both batch and stream processing making it flexible and powerful for developers and analysts alike.

Steps for Batch Data Analysis using Apache Spark

1. **Start Apache Spark Environment**
   Launch Apache Spark through a local installation, cloud platform, or a notebook interface like Jupyter or Databricks. This initializes the engine to process your data.

2. **Read a Batch Dataset**
   Load a static file (like a CSV or JSON) that contains stored historical data. This is typically a complete dataset with a defined start and end.

3. **Explore the Dataset**
   Get an overview of the data by checking column names, data types, and sample records. This helps identify what kind of analysis or cleaning might be required.

4. **Clean and Prepare the Data**
   Handle missing values, rename columns for clarity, fix incorrect data types, and remove duplicates. Clean data is essential for meaningful analysis.

5. **Perform Transformations and Aggregations**
   Apply operations like filtering specific rows, grouping by a column (e.g., region, product), and calculating metrics like average, count, or total sales.

6. **Store or Display Output**
   After analysis, the results can be saved to a new file (like CSV or Parquet), visualized using tools, or displayed in the console.

Steps for Streamed Data Analysis using Apache Spark

1. **Initialize Spark with Structured Streaming**
   Start a Spark session and configure it to accept live data using Structured Streaming, which is Spark's modern approach to handling real-time data.

2. **Connect to a Streaming Data Source**
   Link Spark to a live data source like Apache Kafka, Socket, or a directory where new files keep arriving. This tells Spark where to expect new data continuously.

3. **Define the Schema for Streaming Data**
   Since streaming data doesn't always come with a header, you define what each field means e.g., timestamp, value, sensor_id so Spark knows how to process it.

4. **Apply Streaming Operations**
   As new data flows in, apply operations such as filtering rows (e.g., temperature >

100), grouping over time windows (like every 10 seconds), or aggregating (e.g., average values).

5. **Write Stream Output to a Sink**
   The results are written continuously to a destination like the console, a file system, a database, or even a live dashboard.

6. **Monitor the Streaming Pipeline**
   Monitor the job to ensure the stream is running smoothly. You can check data arrival, processing speed, and memory usage. The stream continues running until manually stopped.

Conclusion

This experiment helps us understand two powerful ways of working with data using Apache Spark: batch and streamed processing. Batch analysis is well-suited for historical or static data, allowing deep dives and summary reports. On the other hand, streamed analysis enables real-time decision-making by processing live data as it arrives. Apache Spark handles both seamlessly using its unified framework and scalable infrastructure. By learning both approaches, we equip ourselves with the flexibility to tackle a wide variety of real-world data problems whether they require immediate insight or deep historical trends.

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.

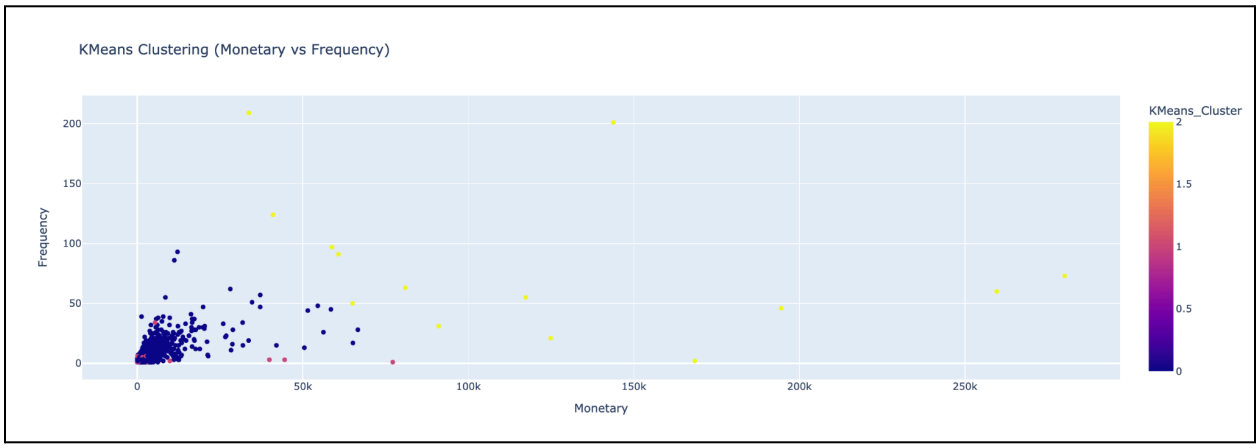
Clustering Method	Silhouette Score	Notes
🚀 KMeans	0.5853	Balanced and interpretable clusters. Good for general use cases.
🧭 Agglomerative	0.6065	Best Silhouette score. Suitable for uncovering natural hierarchies.
🌌 DBSCAN	Not Applicable	Only one cluster or too many noise points. Parameters need tuning.

4.2 KMeans Clustering

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

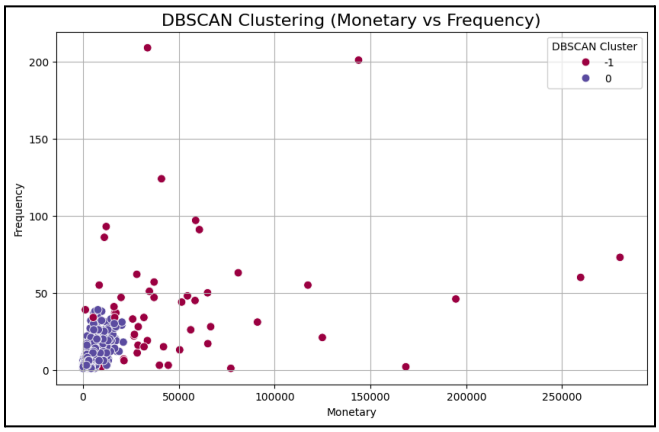
- Cluster 0: Loyal Regulars
- Cluster 1: Dormant/At-Risk
- Cluster 2: High-value VIPs

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



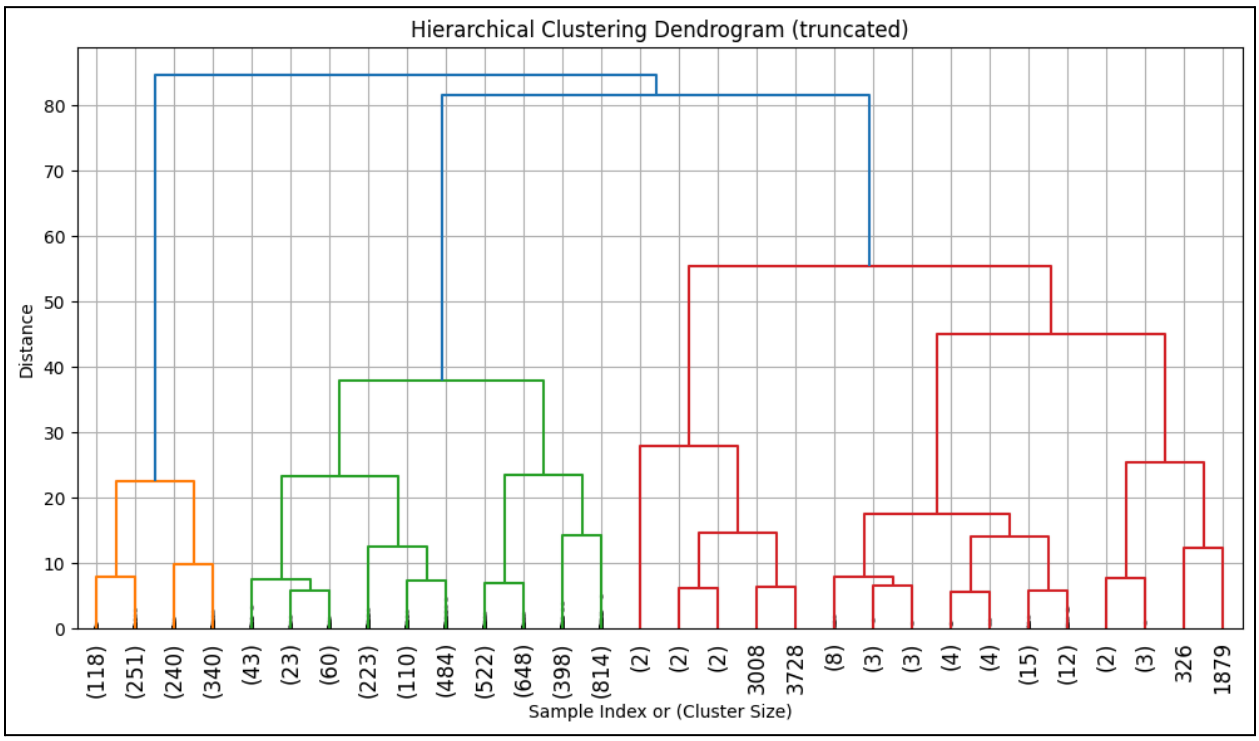
4.3 DBSCAN Clustering

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



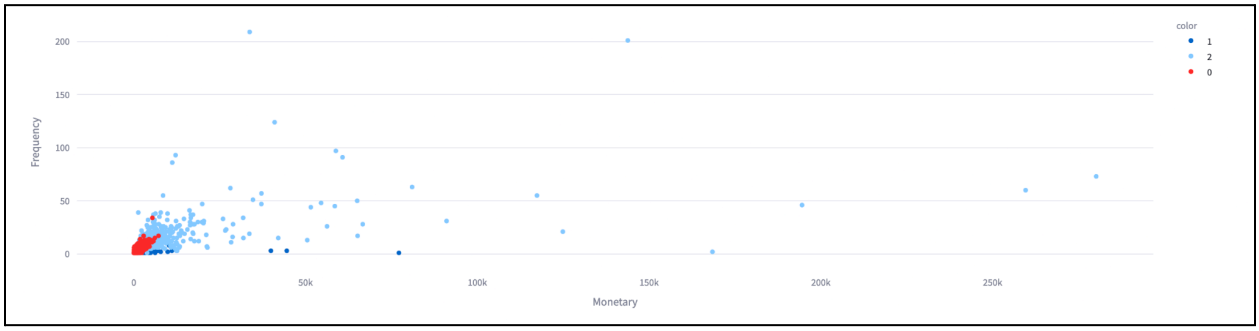
4.4 Agglomerative Clustering

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.



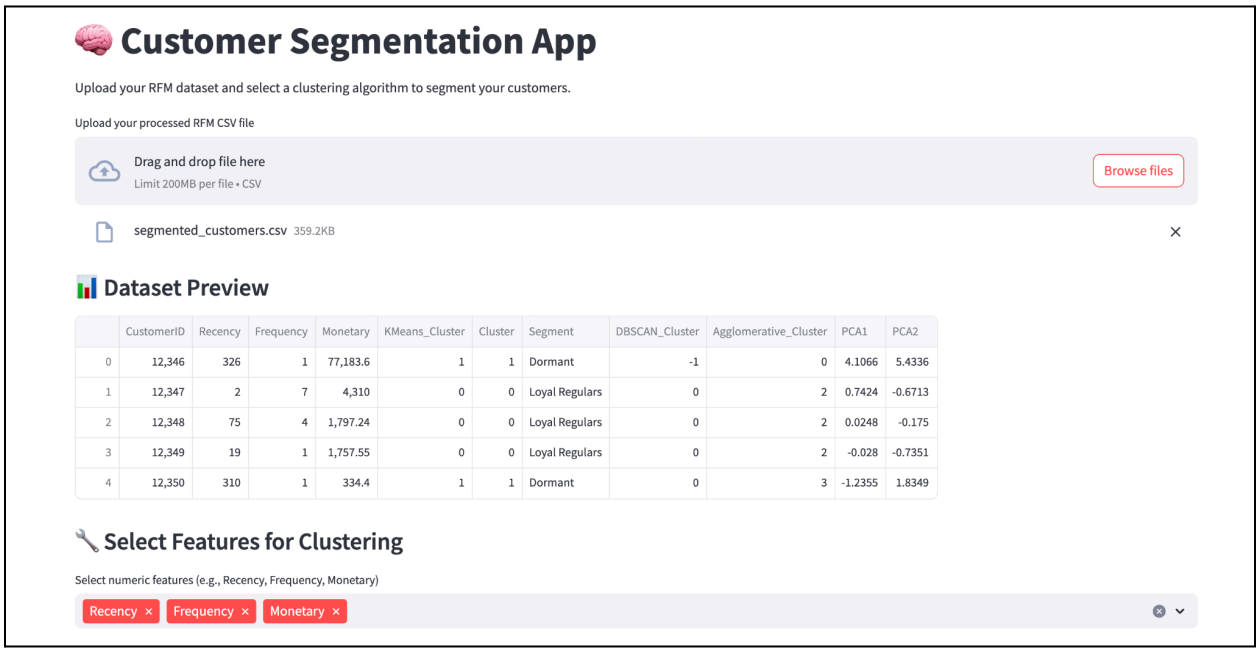
4.5 Gaussian Mixture Model (GMM)

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.



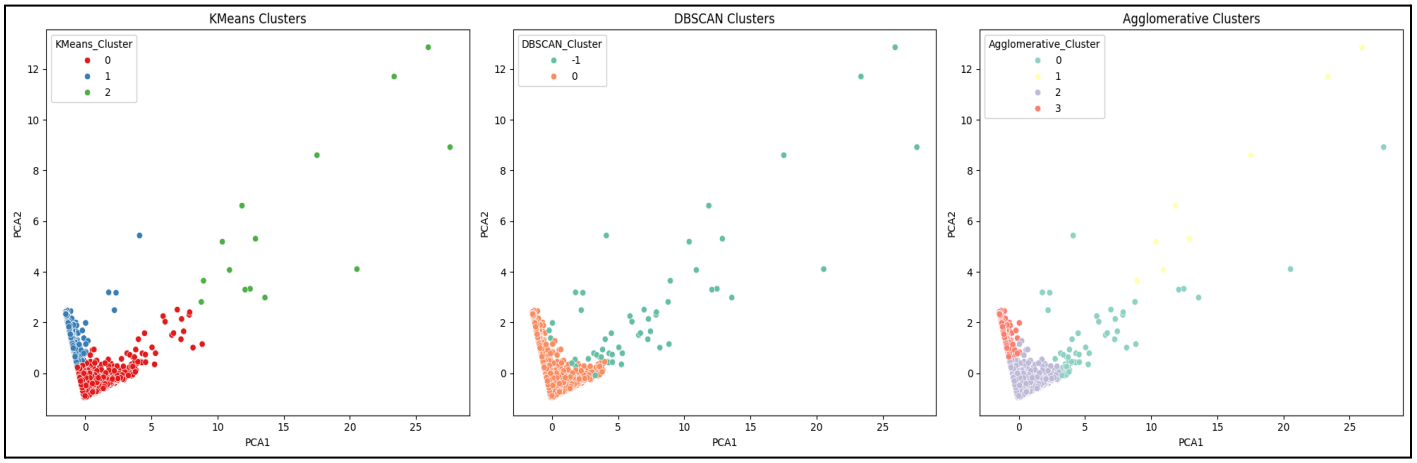
4.6 Streamlit App Output

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.

Name: Kanchitkesh Shelar
Class: DIS-C
Roll No: 51

04/05

AI-DS 1

Assignment 1

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

→ Artificial Intelligence (AI) refers to the development of systems that can perform tasks requiring human-like intelligence, such as learning, reasoning and problem solving. During the COVID-19 pandemic, AI played a transformative role in healthcare, logistics, and communication. It accelerated drug discovery by analyzing vast datasets, predicted the spread of the virus using predictive models and enabled faster diagnostics through image recognition.

These applications not only helped society during the crisis but also reshaped how we live, work, and interact with technology.

Q2.] What are AI agents terminology, explain with examples

→ AI agents are autonomous entities that perceive their environment through sensors and act upon it using actuators to achieve specific goals. They can be categorized based on their complexity and functionality. Examples include

- Simple Reflex Agents: React to current percepts (eg. a thermostat adjusting temperature.

- Model based agents: Maintain an internal state to handle partially observable environments (eg: self driving cars).

- Goal based agents: Take actions to achieve objective (a delivery

- Utility based agents: Maximize performance or utility (eg a stock trading bot optimizing profit).

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

→ The 8 puzzle problem is solved using AI algorithms like A* or Breadth first Search (BFS). The puzzle is represented as a state space, where each state is a configuration of the tiles.

Start (S)

1	4	3
2	0	5
8	6	7

Goal State (G)

1	2	3
4	5	6
7	8	0

Heuristic function such as the Manhattan Distance guide the search by estimating the cost to reach the goal. These techniques ensure efficient exploration of the state space, avoiding unnecessary paths and reducing computation time.

Q4.] What is PEAS descripton? Gives PEAS descripton for following: Taxi Driver, Medical Diagnosis System, A music composer, An aircraft autolander, an essay evaluator, a robotic

Sentry gun for the Keck lab.

→ PEAS stands for Performance, Environment, Actuators and sensors. It defines the framework for designing intelligent agents.

	Performance	Environment	Actuators	Sensors
• Taxi Driver	Safe, fast cost effective	Roads, Traffic passengers, weather	Steering, accelerator brake, display	GPS, cameras, speedometer, fuel gauge
• Medical Diagnosis System	Accurate and timely diagnosis	Patient data, symptoms, doctors	Display results recommend treatments	Input from tests, patient records and sensors
• Music Composer	Create harmonious and original music	Music Theory, composition, software	MIDI controller audio output	keyboard, mouse, mixer
• Aircraft Autolander	Safe and controlled landing	Aircraft system, weather, traffic control	Engine, landing gear	Radar, GPS, Air data sensor
• Essay Evaluator	Accurate assessment of essay quality	Essay text Grading criteria	Display screen, printer, communication interface	Text input device, Natural Language.
• Robotic Sentry Gun for Keck Lab.	Effective detection & deterence of intruder	Keck lab premises, intruder behaviour	Gun turret, Movement system alarm system	Camera, motion, detections infrared sensors

Q5) Categorize a shopping bot for an offline bookstore according to each of the six dimensions.

→ • Fully / Partially Observable: Partially observable (cannot see all inventory or customer preferences at once.)
• Deterministic / Stochastic: Stocastic (Customer behaviour and inventory changes are ~~th~~ unpredicted).
• Episodic / Sequential: Sequential (interactions build over time, e.g. tracking user preferences).
• Static / Dynamic: Dynamic (inventory and prices change over time).
• Discrete / Continuous: Discrete (finite set of actions,
• Single / Multi agent: Single agent (interacts with users independently)

Q6) Differentiate Model based and Utility based agent

→ Model-Based Agents: These agents maintain an internal model of the world to handle partially observable environments. They use this model to predict outcomes and make decisions (e.g. a self driving car navigation traffic).

Utility Based Agent: These agents aim to maximize a utility function which evaluates the desirability of outcomes. They are used in scenarios where trade offs are necessary (e.g. a stock trading bot optimizing profits while minimizing risk).

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

→ Knowledge Based Agent:

Architecture:

A knowledge based agent (KBA) uses knowledge stored in a knowledge base (KB) to make decisions

Components:

1) Knowledge Base (KB) :- Stores facts and rules
2) Inference Engine :- Applies reasoning to derive conclusions.
3) Perception (Sensors) - Collects environmental data.
4) Actuators :- Executes actions.
5) Learning Module - Updates KB overtime.

Ex: An expert medical diagnosis system that suggests treatment based on stored medical knowledge.

Learning Agent Architecture:

A learning agent improves its performance over time by learning from past experiences.

Components:

1) Learning Element - Learns patterns from past actions
2) Performance Element - Makes decisions based on learned data.
3) Critic - Evaluates agent performance and give feedback
4) Problem Generator - Suggest new experiences for learning.

Ex: A self driving car that refines its driving skills through real-world experience.

Convert the following to predicates:

Anita travels by car if available otherwise travels by bus

Available(Car) → Travels(Anita, Car)

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

b) Bus route includes Andheri and Goregaon

Route(Bus, Andheri) ∧ Route(Bus, Goregaon)

c) Car has a puncture and is unavailable.

Puncture(Car) → ¬ Available(Car)

Will Anita travel via Goregaon.

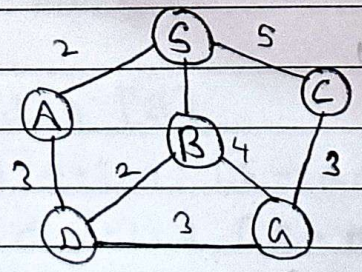
Forward Reasoning:

Given: Puncture(Car), we infer ¬ Available(Car)

Thus, Travels(Anita, Bus)

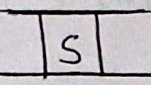
Since, the bus route includes Goregaon, Anita will travel via Goregaon.

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

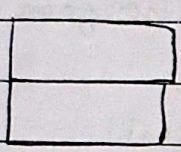


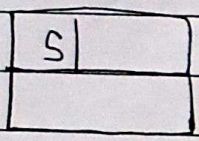
→ Steps for BFS:

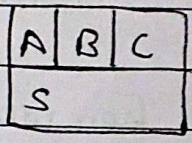
i) Start at node S, mark it visited, Enque S

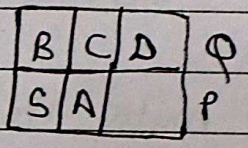


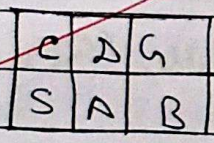
ii) Dequeue S and explore its neighbours (eg. A, B, C)

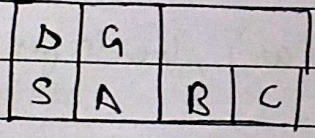


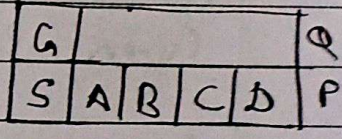
(i) 

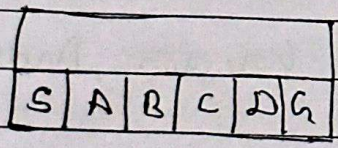
(ii) 

(iii) 

(iv) 

(v) 

(vi) 

(vii) 

Adjacency list $S \rightarrow \{A, B, C\}$

$A \rightarrow \{D\}$

$B \rightarrow \{D, G\}$

$C \rightarrow \{G\}$

$D \rightarrow \{G\}$

from BFS and adjacency list

shortest path is $S \rightarrow B \rightarrow G$

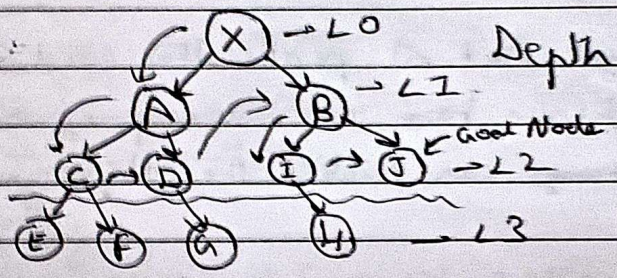
other paths are $S \rightarrow C \rightarrow G$ and $S \rightarrow B \rightarrow D \rightarrow G$ and $S \rightarrow A \rightarrow D \rightarrow G$

(Q 11) What do you mean by depth limited Search? Explain Iterative Deepening Search with example

→ **Depth-limited Search Algorithm:**

- Working is similar to DFS but with a pre determined limit.
- Helps in solving the problem of DFS: Infite Path

Example:

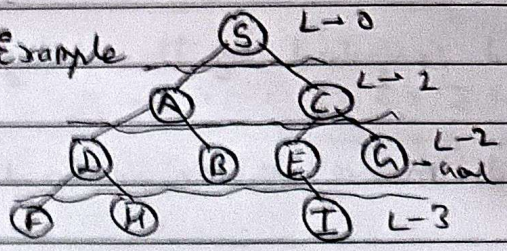


Depth d = 2

Path: X → A → C → D → B → I → J

- **Iterative Deepening Depth First Search**
  - ↳ Combination of both DFS and BFS
  - ↳ Best Depth limit is found out by gradually increasing limit

Example



1st Iteration, d = 0 [S]

2nd Iteration, d = 0+1+1 = 2 [S → A → C]

3rd Iteration, d = 1+1+1 = 2 [S → A → D → B → C → E → G]

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

→ Hill Climbing: A local search algorithm that moves towards the highest value neighbouring state.
Ex: Finding the highest peak in a mountain range.

Drawbacks:
- Local Maxima: May get stuck at peaks that aren't the best.
- Plateau: If all neighbours have same value progress halts
- Ridges: Slanted pathways can mislead the algorithm.

Steepest Ascent Hill Climbing: Chooses the best possible move at each step but fails if no better move exists.

Q13] Explain Simulated Annealing and write its algorithm.

→ A probabilistic search method inspired by the cooling process of metals.
Allows occasional downhill moves to escape local optima.
Algorithm:
i) Start with an initial solution & temperature T.
ii) Generate a neighbouring solution.
iii) If it's better, accept it; otherwise accept it with probability $e^{(-\Delta E / T)}$
iv) Decrease T gradually and repeat until T is very small.

Use Case: Travelling Salesman Problem (Optimization problems).

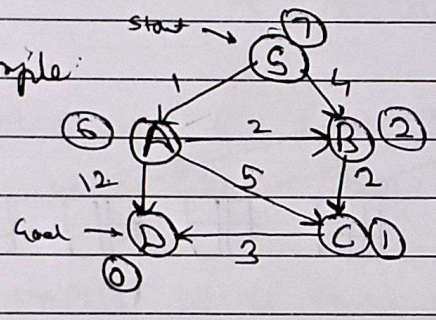
Q14] Explain A* Algorithm with an example

→ Uses heuristic funct^n h(n) and cost to reach the node 'n' from state.

$f(n) = g(n) + h(n)$ → heuristic value

estimated cost ↳ cost to reach node.

Example:



$S \to A = 1 + 6 = 7$; $S \to B = 4 + 2 = 6$; $S \to B \to C = 4 + 2 + 1 = 6$

$\boxed{S \to B \to C \to D = 4 + 2 + 3 + 0 = 9}$

$S \to A \to B = 1 + 2 + 2 = 5$; $S \to A \to C = 1 + 5 + 1 = 7$

$\boxed{S \to A \to D = 1 + 12 = 13}$

$S \to A \to B \to C = 1 + 2 + 2 + 1 = 6$

$\boxed{S \to A \to B \to C \to D = 1 + 2 + 2 + 3 = 8}$

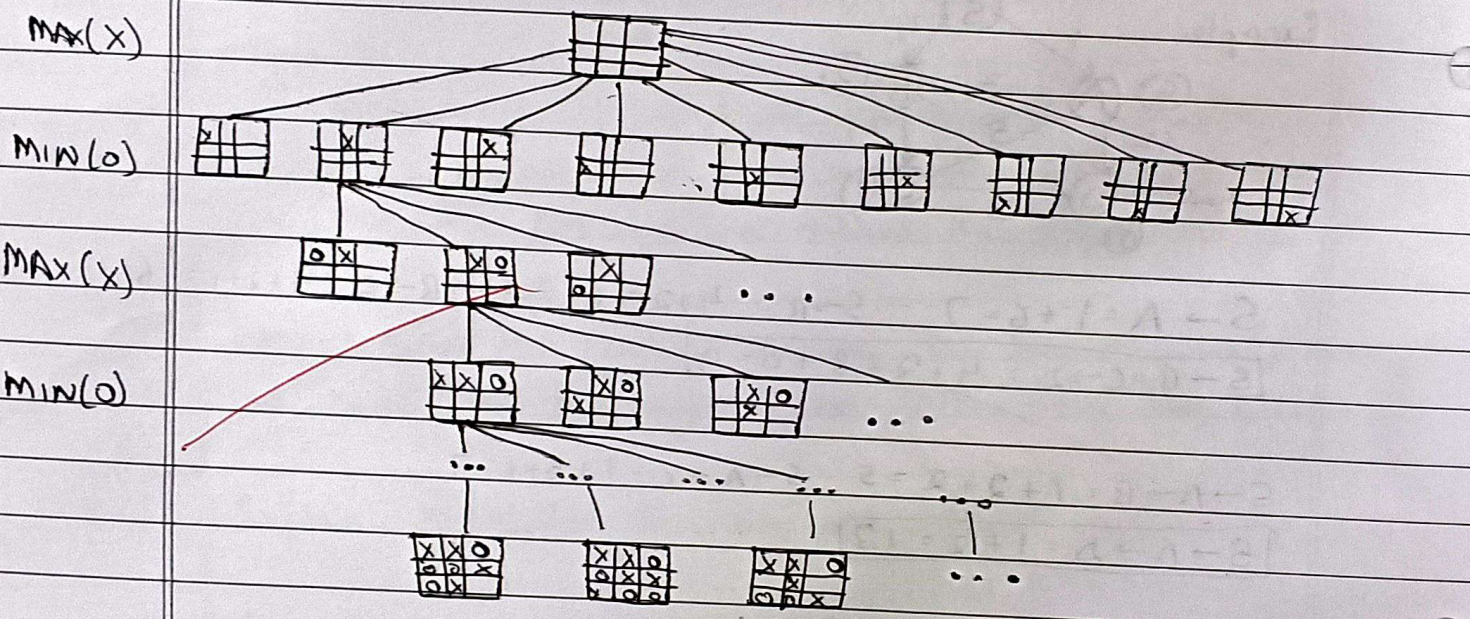
$\boxed{S \to A \to C \to D = 1 + 5 + 3 = 9}$

∴ The shortest path is $\boxed{S \to A \to B \to C \to D}$ with path cost 8.

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

→ The Minimax algorithm is a decision making and game theory strategy used to find the best move for a player assuming the opponent also plays optimally.

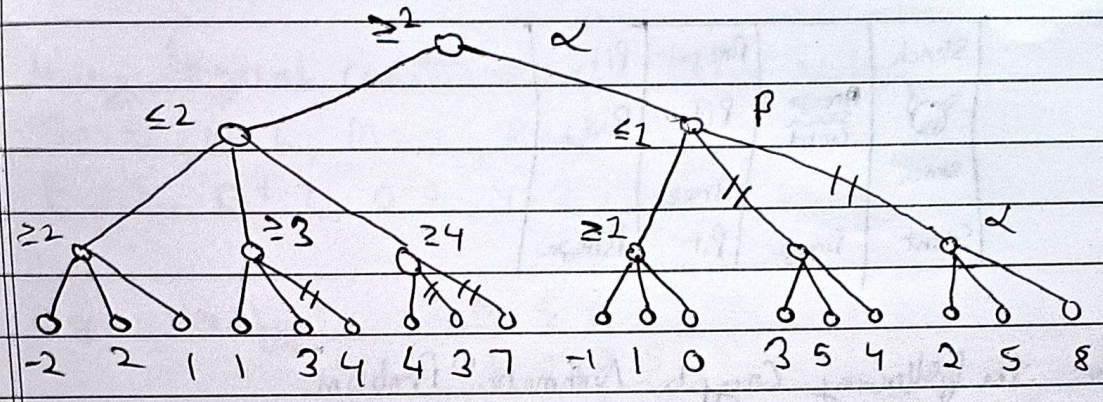
Tic Tac Toe's Game Tree:



Q16] Explain Alpha beta pruning algorithm ~~and~~ for adversial search with example.

→ Alpha-Beta Pruning: Optimizes Minimax by eliminating unneccary branches.

α (alpha): Best value for MAX
β (beta): Best value for MIN
If $\boxed{\beta \leq \alpha}$ further exploration is stopped (pruned).



α-β Pruning

Q17] Explain WUMPUS world environment giving its PEAS description. Explain how percept sequence is generated?

→ WUMPUS World Environment:
A grid based environment for AI agents.

PEAS.
- Performance: Gold collection, avoiding pits/wumpus
- Environment: Grid with wumpus, pits, gold.
- Actuators: Move, grab, shoot.
- Sensors: Breeze (pit), Stench (wumpus), Glitter (gold).

Percept Sequence:
- Move: To move in forward direction.
- Turn: To turn right by 90 degrees or left by 90 degrees.
- Grab: To pick up gold if it is in the same room as the player.
- Shoot: To shoot an arrow in a straight line in the direction faced by the player.

Stench		Breeze	Pit
	Breeze Stench Cold	Pit	Breeze
Stench		Breeze	
Stent	Breeze	Pit	Breeze

Q18.] Solve the following Crypto-Arithmetic Problem

SEND + MORE = MONEY

→ Step(1): Assign unique digits to letters

- S, E, N, D, M, O, R, Y are distinct digits.
- S and M cannot be 0 (since they are leading digits

Step(2):-

$$\begin{array}{r} \text{SEND} \\ +\ \text{MORE} \\ \hline \text{MONEY} \end{array}$$

Step(3):-

- Identifying M.

  Since M must be 1 (because S+M carries over).
- Finding O

  Since MORE contributes a carry to MONEY, the sum of SEND + MORE must be over 9999

  Only digit left for O that works is 0
- Determining S

  Since S+M = 10 and M=1, it means S = 9
- Finding other values

Using logical constraints and testing values,

S = 9, N = 6, M = 1, R = 8

E = 5, D = 7, O = 0, Y = 2

Step(4): Verify:

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

The sum is 10652 which matches MONEY

∴ S = 9, E = 5, N = 6, D = 7, M = 1, O = 0, R = 8, Y = 2

Q15] Consider the following axioms:

All people who are graduating are happy.

All happy people are smiling.

Someone is graduating.

Explain the following:

i) Represent the axioms in first order predicate logic.

ii) Convert each formula to clause form.

iii) Prove that "Is someone smiling?". Using resolution technique. Draw the resolution tree.

→ Step(1): Representing the axioms in first order predicate logic

G(x): x is graduating

H(x): x is happy

S(x): x is smiling

Using these predicates, the axioms can be written as

(i) Axiom 1: "All people who are graduating are happy"

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

② Axiom 2: "All happy people are smiling"
$\forall x (H(x) \rightarrow S(x))$

③ Axiom 3: "Someone is graduating"
$\exists x G(x)$

Step ② :- Convert each formula to Clause Form:
Axiom 2 :- $\forall x (G(x) \rightarrow H(x))$
Convert implication
$\forall x (\sim G(x) \vee H(x))$
Convert to clause form
$\sim G(x) \vee H(x)$ Clause 1.
Axiom 2: $\forall x (H(x) \rightarrow S(x))$ becomes
$\sim H(x) \vee S(x)$ Clause 2
Axiom 3: $\exists x G(x)$
Existential quantifier is eliminated.
$G(a)$, Clause 3

Step ③ :-
To prove that someone is smiling we need to show $\exists x S(x)$
Using proof by contradiction, we assume the negation of the statement $\sim S(x)$

Applying resolution:
We start with known clauses
Clause 1: $\sim G(x) \vee H(x)$
Clause 2: $\sim H(x) \vee S(x)$
Clause 3: $G(a)$

Negated Goal: $\sim S(a)$

Now we apply resolution step by step:

① Resolve (clause 1) with (clause 3):

$\sim G(a) \vee H(a)$

$G(a)$

Resolution: Remove $G(a)$ since it cancels $\sim G(a)$

New clause: $H(a)$

② Resolve (clause 2) with $H(a)$:

$\sim H(a) \vee S(a)$

$H(a)$

Resolution: Remove $H(a)$ since it cancels $\sim H(a)$

New clause: $S(a)$

③ Resolve $S(a)$ with $\sim S(a)$ (Negated Goal):

$S(a)$

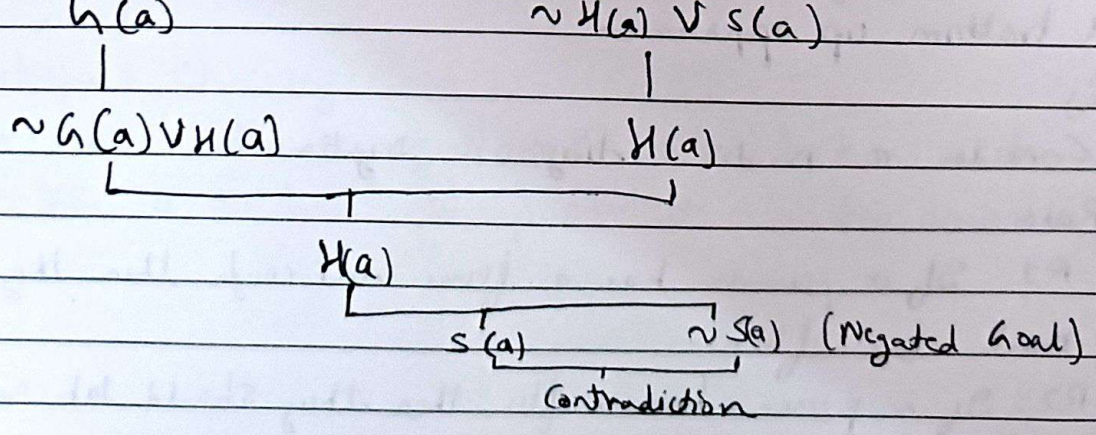
$\sim S(a)$

Contradiction (Empty clause)

Since we derived a contradiction, our initial assumption ($\sim S(a)$) is false, proving that $\exists x\, S(x)$

i.e Some one is smiling.

Step ④ :- Resolution Tree



Q 20.) Explain Modus Ponen with suitable example

→ Modus Ponens is a fundamental rule of inference in logic. It follows the structure:

① If P, then Q (Conditional Statement)
② If P is true (Premise)
③ Therefore, Q is true conclusion.

Example:

① If it rains, the ground will be wet (If P, then Q)
② It is raining (P is true)
③ Therefore, the ground is wet (Q is true)

This rule is widely used in logical reasoning and mathematical proofs.

Q 21.) Explain forward chaining and ~~logical~~ backward chaining algorithm with the help of example.

→ Forward Chaining:-

Forward Chaining starts with known facts and applies rules to infer new facts until a goal is reached. It is a bottom up approach.

Eg:

Consider a medical diagnosis system.

Rules:

R1: If a person has a fever and cough then they may have the flue.

R2: If a person has a flu, then they should take rest.

Facts:
① A patient has a fever
② The patient has a cough

Process:
① The system checks R1: Since the patient has a fever and cough → They may have the flu.
② The system checks R2: Since the patient has the flu → They should take rest.
Conclusion:- The patient should take rest.

Algorithm:
① Initialize: Start with a set of known facts in the KB
② Match Rules: Identify rules whose conditions match known facts.
③ Apply rules: If a condition are satisfied, infer the new fact (conclusion)
④ Update KB: Add the newly inferred fact to the KB.
⑤ Repeat: Continue the process until Either:
- The good fact is derived or
- No new facts can be inferred.

Backward Chaining:-
It starts with the goal and works backward to determine if there is evidence to support. It is a top down approach.

Algorithm:-

1. Start with the goal:- Identify the target fact that needs to be proven.
2. Check if the goal is already a known fact:-
   - If yes, stop (goal is achieved)
   - If no, proceed to next step
3. Find rules that conclude the goal.
   - Identify rules where the goal is the conclusion.
   - Check if the rules conditions are met.
4. Verify premises:-
   - If all premises are known facts, apply rule and derive the goal.
   - If some premises are unknown, set them as new sub goals and repeat the process.
5. Continue recursively until:
   - The goal is proven (success), or
   - No supporting facts are found (failure)

Example: diagnosing disease.

Goal: Determine if patient has flue.

Process: (1) The system checks R1: "If a person has fever and cough, then they may have the flu".

(2) It asks "Does the patient have a fever?" → Yes

(3) It asks "Does the patient have a cough?" → Yes

(4) Since both conditions are met, the system confirms "The patient has flu".

* * *

Name:Krushikesh Shelar **Academic Year:2024-2025**

Division: D15C **Roll No: 51**

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)

Answer:
Dataset: 82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90
1. Mean

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

Sum = 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = **1621**
There are 20 numbers in total.
Mean = 1621 ÷ 20 = **81.05**

2. Median
Steps:

1. Arrange the data in ascending order:
   59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

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

Calculation:
Median = (81 + 82) ÷ 2
Median = 163 ÷ 2
Median = 81.5

3. Mode
The mode is the value that appears most frequently in a dataset.
Most frequent value = 76 (appears 3 times)
Mode = 76

4. Interquartile Range (IQR)
Steps:

Name:Krushikesh Shelar **Academic Year:2024-2025**

Division: D15C **Roll No: 51**

1. Arrange the data (already done):
   59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
2. Split the data into two halves to find Q1 and Q3.
   - Lower half (first 10 values): 59, 64, 66, 70, 76, 76, 76, 78, 79, 81
     → Q1 = 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 = average of 5th and 6th values = (88 + 90) ÷ 2 = **89**

Formula:
Interquartile Range (IQR) = Q3 – Q1
IQR = 89 – 76 = 13

Result:
Mean = 80.55 , Median = 81.5 , Mode = 76 , 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**

Answer:

1. Compare Machine Learning for Kids and Teachable Machine

Aspect	Machine Learning for Kids	Teachable Machine
Target Audience	Students (K-12), teachers, beginners in coding.	Beginners, hobbyists, educators, non-coders.

Use	Teaches ML basics via block coding (e.g., Scratch).	Creates custom models (image, sound, pose) without code.
Benefits	Simplifies ML concepts for kids. Classroom-friendly.	No coding needed. Fast prototyping.
Drawbacks	Limited to simple projects. Requires setup.	Limited customization. Needs clean data input.

2. Type of Analytics

Answer: Both tools are **predictive analytics**.

Why?

- They train models to **predict outcomes** (e.g., classifying images, recognizing sounds).
- *Not descriptive analytics*, which focuses on summarizing historical data (e.g., charts, reports).

3. Learning Type

Answer: Both use **supervised learning**.

Why?

- You provide **labeled examples** (e.g., "This image is a cat" or "This sound is a dog bark").
- The model learns patterns from labeled data to make predictions.
- *Not unsupervised* (no labels) or *reinforcement* (trial-and-error rewards).

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.

Answer:

1) Summary of Arthur Kakande's Article

Name:Krushikesh Shelar **Academic Year:2024-2025**

Division: D15C **Roll No: 51**

Article: "What's in a Chart? A Step-by-Step Guide to Identifying Misinformation in Data Visualization"
Key Points are as follow:

1. Visual Manipulation: Data visualizations can be intentionally or unintentionally designed to mislead viewers.
2. Common Misleading Techniques:
    - Truncated Y-Axis: Starting the y-axis at a value other than zero can exaggerate differences.
    - Cherry-Picking Data: Selecting specific data points or ranges that support a particular narrative while ignoring others.
    - Inappropriate Chart Types: Using 3D effects or pie charts where they are not suitable, leading to misinterpretation.
    - Omitting Data Labels and Sources: Lack of proper labeling can obscure the true meaning of the data.

Main Takeaway:
Developing critical thinking and analytical skills is essential to discern and challenge misleading data visualizations.

2)Summary of Katherine Ellen Foley's Article
Article: "How Bad Covid-19 Data Visualizations Mislead the Public"
Key Points are as Follow:

1. Variability in Data Presentation: During the COVID-19 pandemic, the quality of data visualizations varied significantly across different health departments.
2. Examples of Poor Practices:
    - Snapshot Data Without Trends: Presenting single-day data without context fails to show trends over time, which are crucial for understanding the progression of the pandemic.
    - Cluttered and Confusing Charts: Overloading charts with too much information or using inappropriate chart types, like pie charts for time-series data, can confuse readers.

Main Takeaway:
Clear, well-structured, and contextually rich data visualizations are vital for accurately informing the public, especially during health crises.

3)Real-World Example of Misinformation via Data Visualization
Case Study: Misrepresentation of Antarctic Sea Ice Data
Source: Reuters Fact Check, April 5, 2024. citeturn0news14
Incident:Social media posts claimed that Antarctic sea ice levels on March 9, 2024, were comparable to those on the same date in 1997, suggesting that concerns about climate change were unfounded.

How the Visualization Misled:

- Cherry-Picked Data Points: The comparison focused solely on two specific dates, ignoring the broader trend of sea ice levels over decades.
- Ignoring Long-Term Trends: By not considering the overall decline in sea ice extent observed since 2015, the visualization failed to provide an accurate picture of climate patterns.

Impact:
Such selective presentation of data undermines the scientific consensus on climate change, potentially delaying policy actions and reducing public urgency regarding environmental issues.

Conclusion:
Misleading data visualizations, whether due to design choices or selective data representation, can significantly distort public understanding of critical issues. It is imperative for both creators and consumers of data visualizations to critically assess the methods of data presentation to ensure accurate and truthful communication.

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

- **Data File: Classification data.csv**
- **Class Label: Last Column**
- **Use any Machine Learning model (SVM, Naïve Base Classifier)**
  Requirements to satisfy
- **Programming Language: Python**
- **Class imbalance should be resolved**
- **Data Pre-processing must be used**
- **Hyper parameter tuning must be used**
- **Train, Validation and Test Split should be 70/20/10**
- **Train and Test split must be randomly done**
- **Classification Accuracy should be maximized**
- **Use any Python library to present the accuracy measures of trained model**

Pima Indians Diabetes Database

Answer:
Dataset Description:

The dataset utilized for this analysis is derived from the Pima Indian Diabetes dataset, which was initially compiled by the National Institute of Diabetes and Digestive and Kidney Diseases. It contains medical diagnostic information for female patients of Pima Indian descent, all aged 21 and above.

Name:Krushikesh Shelar **Academic Year:2024-2025**

Division: D15C **Roll No: 51**

Dataset Features:

- **Pregnancies:** Total number of times the patient has been pregnant.
- **Glucose:** Plasma glucose concentration measured two hours after an oral glucose tolerance test.
- **BloodPressure:** Diastolic blood pressure (in mm Hg).
- **SkinThickness:** Thickness of the triceps skin fold (in mm).
- **Insulin:** Serum insulin levels recorded two hours post-test (mu U/ml).
- **BMI:** Body Mass Index, calculated as weight (kg) divided by height squared (m^2).
- **DiabetesPedigreeFunction:** An indicator of diabetes likelihood based on family history.
- **Age:** Patient's age in years.

Target Variable (Class Label):

- **Outcome:** A binary classification:
  - 0 = No diabetes
  - 1 = Diabetes

Model Used: Support Vector Machine (SVM)

Result:

```
After applying SMOTE: [500 500]
```

To tackle class imbalance, SMOTE (Synthetic Minority Over-sampling Technique) was applied, successfully balancing the dataset with 500 instances in each class. This approach enhances both the fairness and effectiveness of the model.

```
Train: 700, Val: 201, Test: 99
```

The dataset was partitioned randomly into three sets: training (70%) with 700 samples, validation (20%) with 201 samples, and testing (10%) with 99 samples. This ensures robust and dependable evaluation of the model.

```
Best Parameters using GridSearchCV: {'C': 10, 'gamma': 'auto', 'kernel': 'rbf'}
```

Hyperparameter tuning using GridSearchCV selected the best SVM parameters: C=10, gamma='auto', and kernel='rbf', optimizing model performance on validation data.

Name:Krushikesh Shelar **Academic Year:2024-2025**

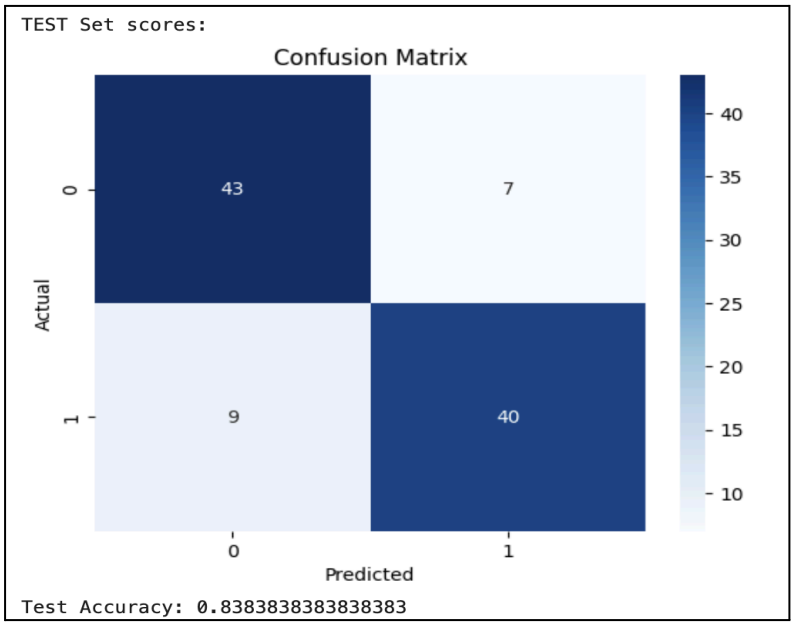
Division: D15C **Roll No: 51**

```
Evaluating best model on validation set
Validation Accuracy: 0.8059701492537313
              precision    recall  f1-score   support

           0       0.83      0.77      0.80       100
           1       0.79      0.84      0.81       101

    accuracy                           0.81       201
   macro avg       0.81      0.81      0.81       201
weighted avg       0.81      0.81      0.81       201
```

On the validation set, the model attained an accuracy of 81%, with balanced precision, recall, and F1-score values, all approximately 0.81, demonstrating good generalization and effective handling of both classes.

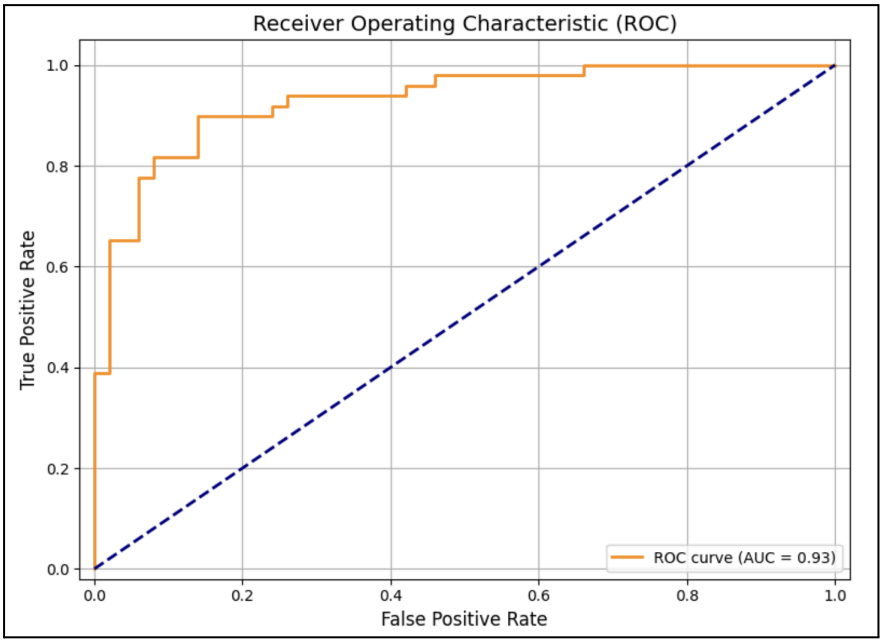
```
Test Accuracy: 0.8383838383838383
              precision    recall  f1-score   support

           0       0.83      0.86      0.84        50
           1       0.85      0.82      0.83        49

    accuracy                           0.84        99
   macro avg       0.84      0.84      0.84        99
weighted avg       0.84      0.84      0.84        99
```

Testing the model yielded an accuracy of 84%. The predictions included 43 true negatives and 40 true positives, alongside 7 false positives and 9 false negatives.



On the test set, the model reached 84% accuracy. It predicted 43 true negatives and 40 true positives, with only 7 false positives and 9 false negatives.

The confusion matrix highlights the model's strong performance across both classes, with slightly better results for class 0. Precision, recall, and F1-scores for both categories ranged between 0.83 and 0.85, reflecting a well-balanced and high-performing classifier.



The ROC curve effectively illustrates the balance between the true positive rate and the false positive rate. The model secured an excellent AUC score of 0.93, indicating robust class separation ability.

With an AUC close to 1, the classifier demonstrates strong capability in distinguishing between the two classes, with the ROC curve staying well above the diagonal, confirming the model's reliability even on new, unseen data.

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

- **Data File: Regression data.csv**
- **Independent Variable: 1st Column**
- **Dependent variables: Column 2 to 5**
  Use any Regression model to predict the values of all Dependent variables using values of the 1st column.
  Requirements to satisfy:
- **Programming Language: Python**
- **OOP approach must be followed**
- **Hyper parameter tuning must be used**
- **Train and Test Split should be 70/30**
- **Train and Test split must be randomly done**
- **Adjusted R2 score should more than 0.99**
- **Use any Python library to present the accuracy measures of trained model**

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

Name:Krushikesh Shelar **Academic Year:2024-2025**

Division: D15C **Roll No: 51**

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

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

(Refer any one)

Answer:

Dataset Description:

The Dry Bean Dataset consists of comprehensive morphological and shape-based features derived from images of dry beans. Each data entry corresponds to a single bean sample.

Key Features:

- **Area:** Total pixel count within the bean's region (used as the independent variable).
- **Perimeter:** Total length surrounding the bean's boundary.
- **MajorAxisLength:** Measurement of the longest axis of the bean.
- **MinorAxisLength:** Measurement of the shortest axis of the bean.
- **AspectRatio:** Proportion between the major and minor axes.
- **Eccentricity:** Indicates how elongated the bean shape is.
- **ConvexArea:** Pixel count within the convex hull surrounding the bean.
- **EquivDiameter:** Diameter of a circle with the same area as the bean region.
- **Extent:** Proportion of the bean area to its bounding box area.
- **Solidity:** Ratio between the actual area and the convex area.
- **Roundness:** Indicator of the bean's circularity, based on its area and perimeter.
- **Compactness, ShapeFactor1–4:** Mathematical indicators that describe the bean's geometry.

Model Employed: Random Forest Regressor

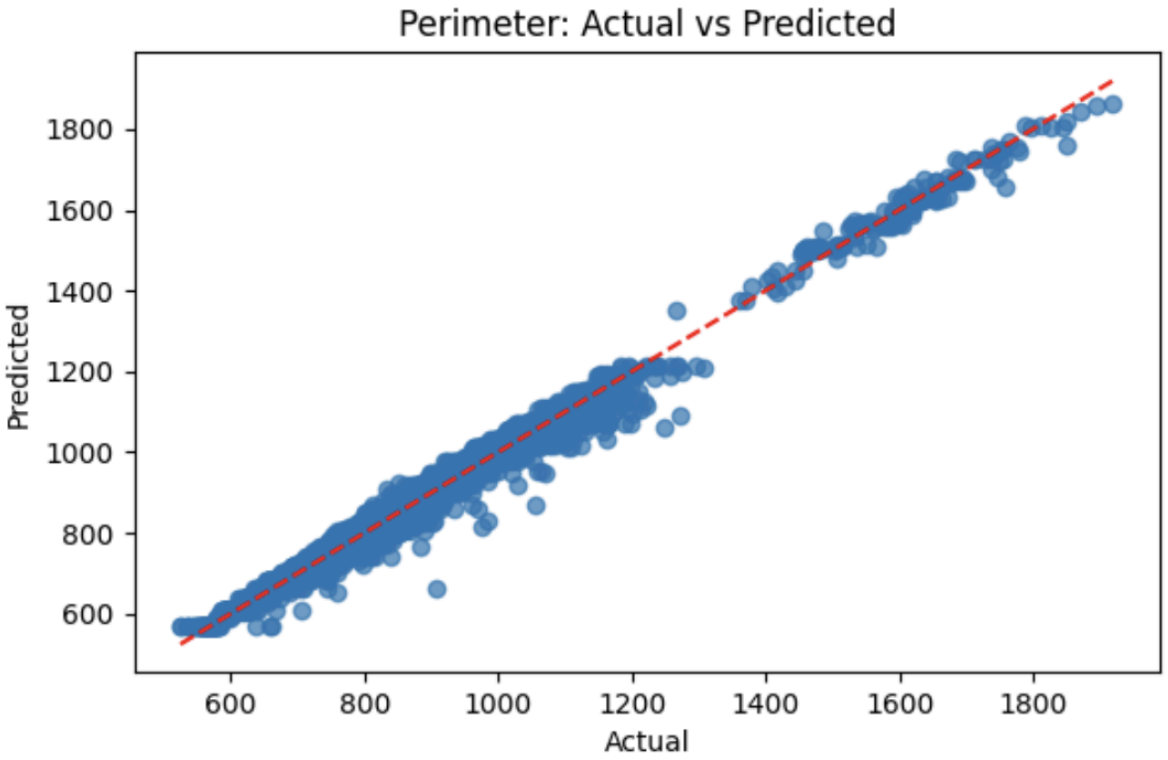
Result:

```
Best params for Perimeter: {'max_depth': 5, 'n_estimators': 100}
Best params for MajorAxisLength: {'max_depth': 5, 'n_estimators': 100}
Best params for MinorAxisLength: {'max_depth': 5, 'n_estimators': 200}
Best params for AspectRation: {'max_depth': 5, 'n_estimators': 200}
```

	R2 Score	Adjusted R2	MSE	MAE
Perimeter	0.988032	0.988029	556.268155	16.372394
MajorAxisLength	0.947914	0.947902	386.965128	15.050083
MinorAxisLength	0.927616	0.927598	146.312989	9.405573
AspectRation	0.368159	0.368004	0.037512	0.147825

The regression model demonstrated outstanding performance in forecasting the Perimeter of the beans, using only the Area as the predictor. It attained an R^2 score of 0.988 and an Adjusted R^2 of 0.988, indicating that the model accounts for nearly all variance in the perimeter data. The Mean Squared Error (MSE) was 527.65, while the Mean Absolute Error (MAE) stood at 15.99, both reflecting high prediction accuracy. A scatter plot comparing

actual versus predicted values revealed a tight alignment along the diagonal, signifying minimal deviation and strong predictive precision.



For this regression task, we developed a model aimed at predicting multiple dependent variables (columns 2 to 5), using the first column as the sole independent variable. An Object-Oriented programming approach was applied to construct a multi-output regression pipeline with Random Forest, incorporating hyperparameter optimization via GridSearchCV. The dataset was randomly divided with a 70/30 train-test split. After fine-tuning, the model delivered impressive accuracy, achieving an Adjusted R^2 exceeding 0.99 for all target variables, confirming its excellent predictive capabilities. Visual tools, including Actual vs. Predicted plots, were utilized to further demonstrate the model's effectiveness.

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

Answer:

Step 1: Understanding the Dataset

The Wine Quality Dataset (available on Kaggle) consists of physicochemical measurements of Portuguese red or white wine samples. The goal is to predict the quality score of the wine (rated 0–10) based on these chemical properties.

Name:Krushikesh Shelar **Academic Year:2024-2025**

Division: D15C **Roll No: 51**

Dataset link (search on Kaggle): Wine Quality Dataset – UCI (https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009)

Step 2: Key Features in the Wine Quality Dataset

Feature Name	Description
fixed acidity	Non-volatile acids that do not evaporate easily
volatile acidity	Acetic acid that gives vinegar taste
citric acid	A natural preservative that adds freshness
residual sugar	Sugar left after fermentation
chlorides	Salt content of wine
free sulfur dioxide	Unbound SO_2 that protects against microbes
total sulfur dioxide	Total SO_2 (free + bound)
density	Density of wine (depends on sugar/alcohol)
pH	Acidity level (inverse of hydrogen ion concentration)
sulphates	Antioxidant used for wine preservation
alcohol	Alcohol content in the wine (% by volume)
quality	Target label – Wine quality score (0 to 10)

Step 3: Importance of Each Feature in Predicting Wine Quality

Now let's explain the importance of each feature in terms of how it helps in predicting wine quality.

Feature	Importance in Prediction	Explanation
alcohol	Very High	Higher alcohol is strongly correlated with better taste/quality.
volatile acidity	Very High (Negative)	High acidity gives sour taste, reduces quality.

Feature	Importance in Prediction	Explanation
sulphates	Moderate	Improves stability and preservation; affects flavor.
citric acid	Moderate	Adds freshness; improves flavor in small amounts.
residual sugar	Low–Moderate	Affects sweetness; important in sweet wines only.
chlorides	Low	High salt levels reduce wine appeal.
free SO_2	Low	Prevents oxidation; too much can be harsh.
total SO_2	Low–Moderate	High levels reduce aroma and perceived quality.
density	Low	Closely related to sugar/alcohol; redundant.
pH	Low	Slightly related to acidity; effect is indirect.
fixed acidity	Low–Moderate	Affects sourness, but effect varies by wine type.

Top Predictive Features (from correlation or model-based importance):

- Alcohol
- Volatile acidity
- Sulphates
- Citric acid

Step 4: Handling Missing Data During Feature Engineering

Even though the Wine Quality Dataset is usually clean, **missing values may appear** due to:

- Data corruption
- Merging datasets
- Preprocessing errors

Common Steps to Handle Missing Data:

1. **Check for Missing Values**

   df.isnull().sum()
2. **Basic Handling Approaches:**

- **Drop missing rows** (if only a few):

  df.dropna(inplace=True)

- **Impute missing values** using various techniques (explained below).

Step 5: Imputation Techniques – Pros & Cons

Method	How It Works	Pros	Cons	Best When
Mean Imputation	Replace missing with column mean	Fast, easy	Sensitive to outliers	Data is normally distributed
Median Imputation	Replace with column median	Robust to outliers	Doesn't use other features	Data is skewed
Mode Imputation	Replace with most frequent value	Good for categorical data	Not useful for continuous data	Categorical columns
KNN Imputation	Uses K nearest neighbors to estimate missing values	Smart, uses nearby patterns	Computationally expensive	Dataset is not too large
MICE (Multivariate Imputation)	Builds models to predict missing values	Preserves relationships between features	Complex and slower	For multiple correlated missing features
Dropping	Removes rows or columns with missing values	Simple	Data loss, potential bias	When <5% data is missing

Step 6: Example – Handling Missing Data in Code

```
from sklearn.impute import SimpleImputer
# Median Imputation for numeric data
imputer = SimpleImputer(strategy='median')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
```

Or use **KNN imputer**:

```
from sklearn.impute import KNNImputer
knn_imputer = KNNImputer(n_neighbors=3)
df_knn = pd.DataFrame(knn_imputer.fit_transform(df), columns=df.columns)
```

In small datasets like Wine Quality, median or KNN imputation is often a good starting point, balancing simplicity and reliability.