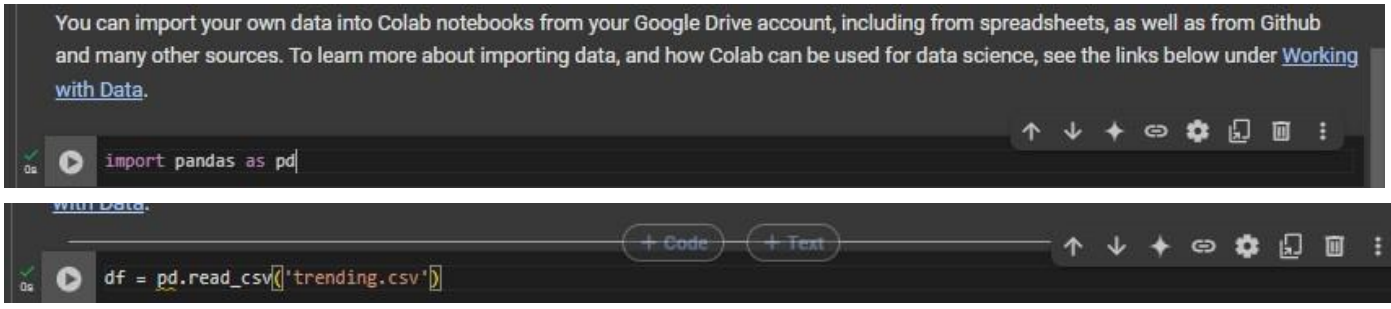
NAME: SHIVEN BANSAL CLASS/ROLL NO: D15C/ 3

DS-1 Lab Exp 1

AIM: Introduction to Data science and Data preparation using Pandas steps.

- 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

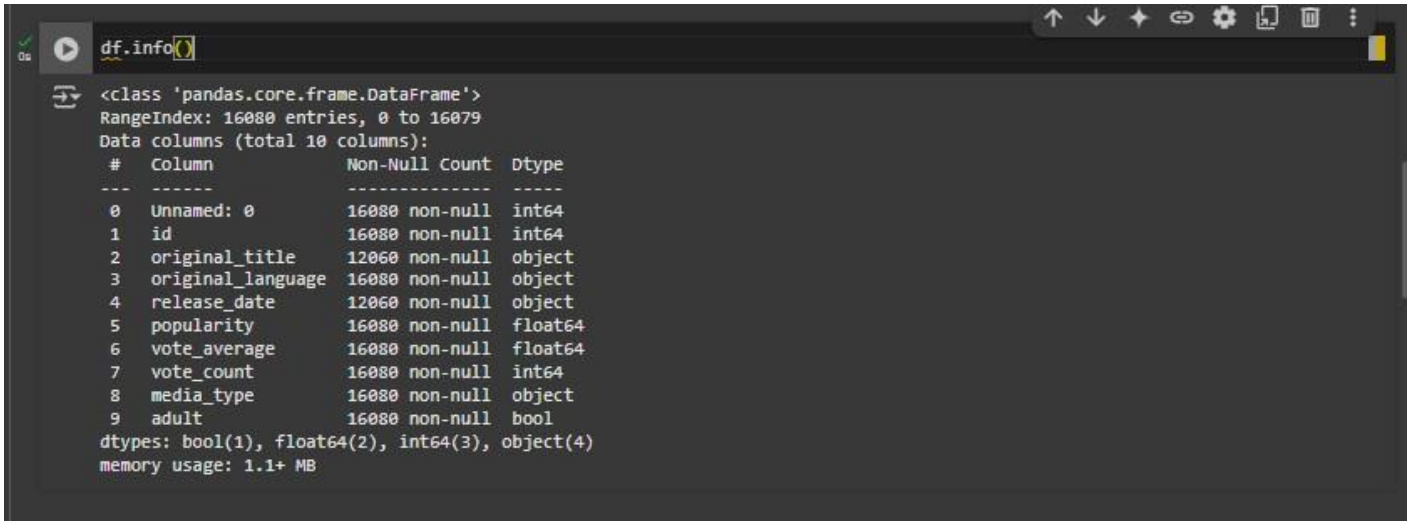
Step 1: Firstly import Pandas Library as pd an then Load data in Pandas using pd.read_csv.



Step 2: Get Description of the Dataset by using following 2 commands

df.info() -> Get basic information about the dataset

df.describe() -> Summary statistics of the dataset



NAME: SHIVEN BANSAL CLASS/ROLL NO: D15C/ 3

```
df.describe()
```

	Unnamed: 0	id	popularity	vote_average	vote_count
count	16080.0000	1.608000e+04	16080.000000	16080.000000	16080.00000
mean	8039.5000	5.758387e+05	934.262000	7.536350	1039.75000
std	4642.0405	3.271352e+05	2229.935599	1.057306	2326.96193
min	0.0000	7.660000e+04	30.374000	4.800000	3.00000
25%	4019.7500	1.963872e+05	61.750250	6.875000	9.75000
50%	8039.5000	6.580765e+05	94.859000	7.721000	53.50000
75%	12059.2500	8.569955e+05	804.053750	8.040250	363.00000
max	16079.0000	1.049638e+06	10224.280000	10.000000	8697.00000

Step 3: Drop Columns that aren't useful. From Our Dataset we are dropping the **"adult"** column .

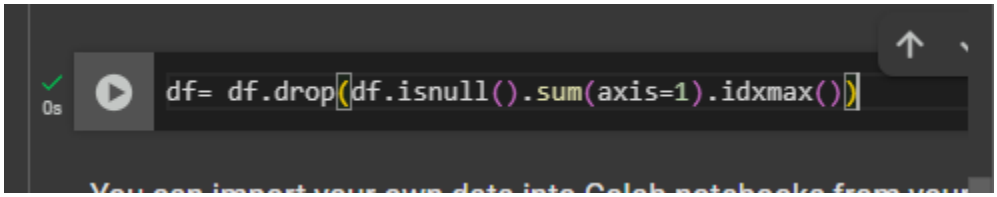
```
cols = ['adult']
df = df.drop(cols,axis=1)
```

We can see that it returned total 9 columns as it dropped the adult column

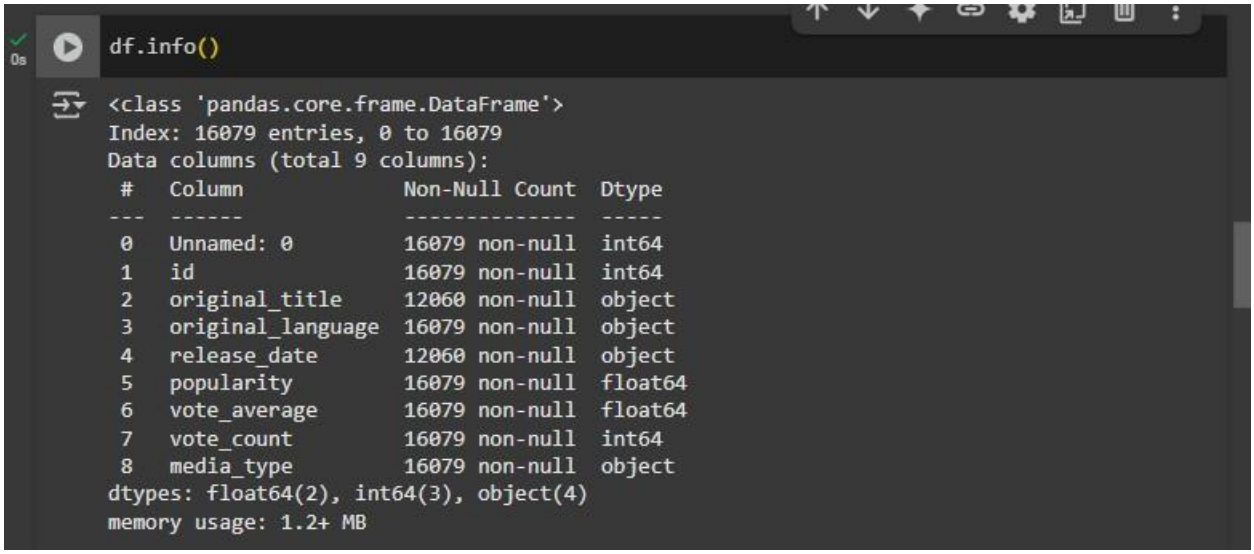
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16079 entries, 0 to 16079
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Unnamed: 0         16079 non-null  int64
 1   id                 16079 non-null  int64
 2   original_title     12060 non-null  object
 3   original_language  16079 non-null  object
 4   release_date       12060 non-null  object
 5   popularity         16079 non-null  float64
 6   vote_average       16079 non-null  float64
 7   vote_count         16079 non-null  int64
 8   media_type         16079 non-null  object
dtypes: float64(2), int64(3), object(4)
memory usage: 1.2+ MB
```

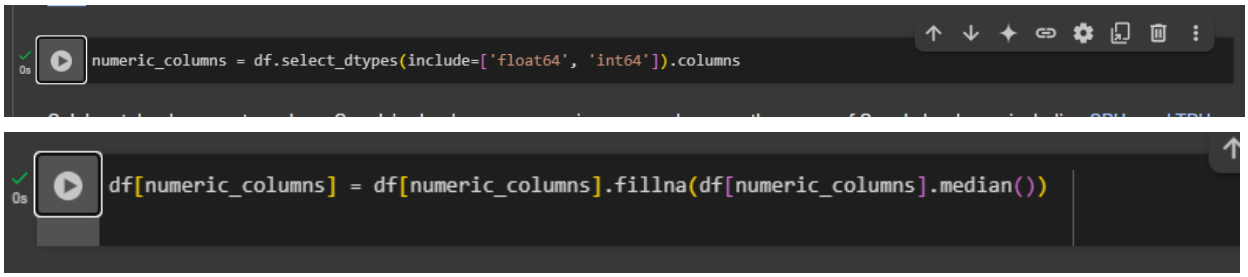
Step 4: Drop row with maximum missing values.
df.isnull().sum(axis=1) -> Computes the number of missing values (NaN) for each row.
.idxmax() -> Returns the index of row with max. no. of missing value



We can see below that df.info() returns total 16079 entries, initially there were 16080 entries



Step 5: Taking care of missing data.
We can fill the empty numeric values with mode or median or mean. Below we had filled it with median. Firstly we had fetched the numeric values and then using **.fillna().median** we had filled it.



We can see that all the columns which had empty are filled. As they returned the sum 0

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

```
Unnamed: 0              0
id                      0
original_title       4019
original_language       0
release_date         4019
popularity              0
vote_average            0
vote_count              0
media_type              0
dtype: int64
```

df.head() returns starting 5 values

```
print(df.head())
```

```
   Unnamed: 0      id           original_title original_language  \
0           0  638974         Murder Mystery 2          English
1           1  677179                Creed III          English
2           2  726759                   Tetris          English
3           3   76600  Avatar: The Way of Water          English
4           4  849869                   길복순           Korean

  release_date  popularity  vote_average  vote_count media_type
0   2023-03-26     235.901         6.800         187      movie
1   2023-03-01    1537.879         7.200         561      movie
2   2023-03-15      66.661         7.700         100      movie
3   2022-12-14   10224.280         7.742        6335      movie
4   2023-02-17      33.985         6.900          39      movie
```

```
print(df.head(20))
```

```
6           6   493529  Dungeons & Dragons: Honor Among Thieves
7           7   932430                                Prom Pact
9           9   816904                                   Momias
10         10   514999                           Murder Mystery
11         11  1049638                                 Rye Lane
12         12   739405         Operation Fortune: Ruse de Guerre
13         13   158876                                      NaN
14         14   921355                                  Assassin
15         15   117465                                      NaN
16         16   933419                                Champions
17         17   208891                                      NaN
18         18   878375                  On a Wing and a Prayer
19         19    82856                                      NaN
20         20   638974                          Murder Mystery 2

   original_language release_date  popularity  vote_average  vote_count  \
0            English   2023-03-26     235.901         6.800         187
1            English   2023-03-01    1537.879         7.200         561
```

0s completed at 12:09 PM

Step 6: Create dummy variables. By using the below commands separate columns are created for each unique value in a column

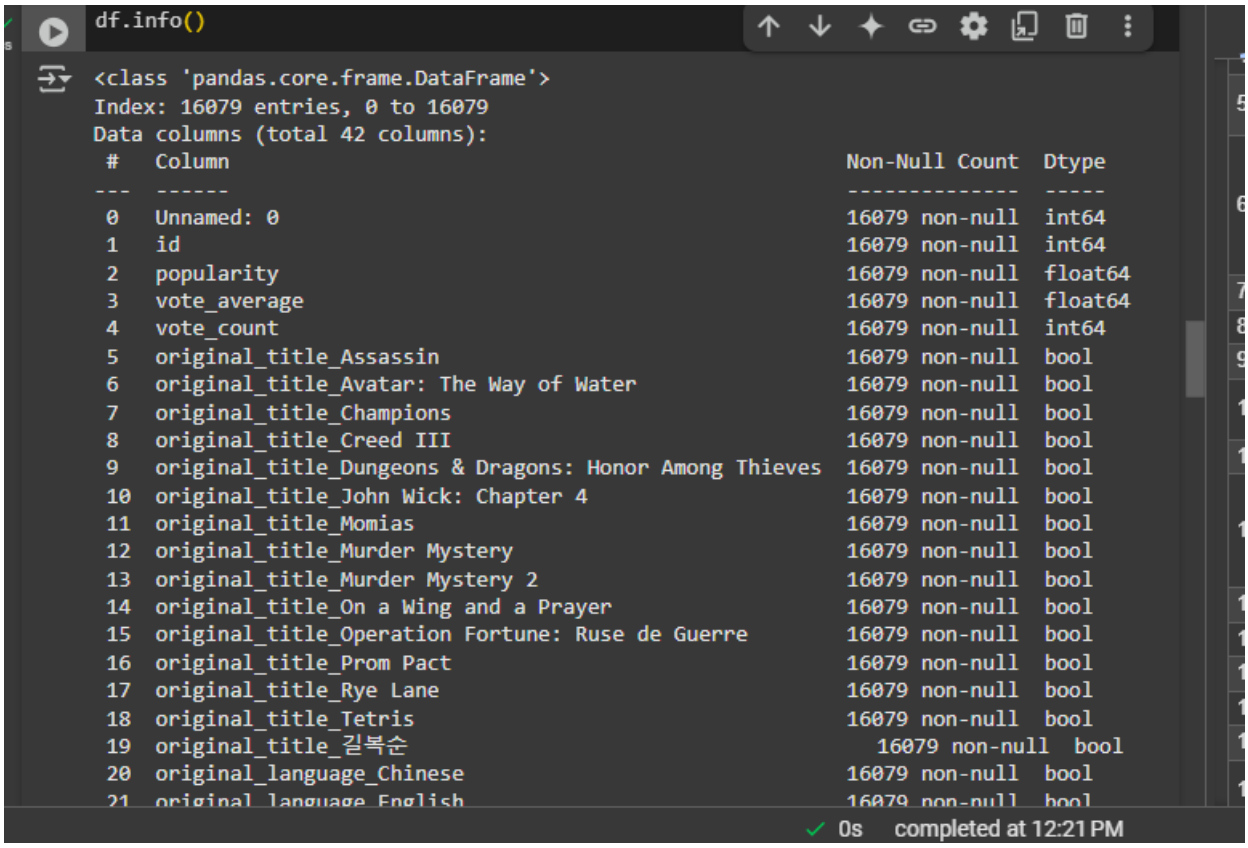
```
df = pd.get_dummies(df)
```

+ Code + Text

```
print(df.head(20))
```

```
7                    False                    False                    False
9                    False                    False                    False
10                   False                    False                    False
11                   False                     True                    False
12                   False                    False                    False
13                   False                    False                    False
14                   False                    False                    False
15                   False                    False                    False
16                   False                    False                    False
17                   False                    False                    False
18                   False                    False                    False
19                   False                    False                    False
20                   False                    False                    False

    release_date_2023-03-23  release_date_2023-03-26  release_date_2023-03-30  \
0                     False                     True                    False
```

We can understand the working here,
As we can see that we now it have returned 42 columns. But previously our data had 9 columns .
So this change is because of the dummy variables , it have created separate column for each unique value in a column
Below it shows original_title_Assassin, original_language_English.



```
df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 16079 entries, 0 to 16079
Data columns (total 42 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   Unnamed: 0                                            16079 non-null  int64
 1   id                                                    16079 non-null  int64
 2   popularity                                            16079 non-null  float64
 3   vote_average                                          16079 non-null  float64
 4   vote_count                                            16079 non-null  int64
 5   original_title_Assassin                               16079 non-null  bool
 6   original_title_Avatar: The Way of Water               16079 non-null  bool
 7   original_title_Champions                              16079 non-null  bool
 8   original_title_Creed III                              16079 non-null  bool
 9   original_title_Dungeons & Dragons: Honor Among Thieves 16079 non-null  bool
 10  original_title_John Wick: Chapter 4                   16079 non-null  bool
 11  original_title_Momias                                 16079 non-null  bool
 12  original_title_Murder Mystery                         16079 non-null  bool
 13  original_title_Murder Mystery 2                       16079 non-null  bool
 14  original_title_On a Wing and a Prayer                 16079 non-null  bool
 15  original_title_Operation Fortune: Ruse de Guerre      16079 non-null  bool
 16  original_title_Prom Pact                              16079 non-null  bool
 17  original_title_Rye Lane                               16079 non-null  bool
 18  original_title_Tetris                                 16079 non-null  bool
 19  original_title_길복순                                   16079 non-null  bool
 20  original_language_Chinese                             16079 non-null  bool
 21  original_language_English                             16079 non-null  bool

✓ 0s  completed at 12:21 PM
```

NAME: SHIVEN BANSAL CLASS/ROLL NO: D15C/ 3

Step 7: Create Outliers

They identify and handle unusual values in a dataset.

We are using Z-score to handle the data

```
from scipy import stats

# Select only numerical columns
numerical_df = df.select_dtypes(include=['float64', 'int64'])

# Remove constant or problematic columns
numerical_df = numerical_df.loc[:, numerical_df.nunique() > 1]
numerical_df = numerical_df.dropna(axis=1)

# Calculate Z-scores
z_scores = stats.zscore(numerical_df)

# Handle cases with NaN Z-scores
z_scores = pd.DataFrame(z_scores, columns=numerical_df.columns).fillna(0)

# Identify rows with Z-scores > 3 or < -3
outliers = (abs(z_scores) > 3).any(axis=1)

# Filter the outliers
outlier_rows = df[outliers]
print(outlier_rows)
```

```
       Unnamed: 0     id  popularity  vote_average  vote_count  \
3               3  76600   10224.280         7.742        6335
19             19  82856    1108.646         8.488        8697
23             23  76600   10224.280         7.742        6335
39             39  82856    1108.646         8.488        8697
43             43  76600   10224.280         7.742        6335
...           ...    ...         ...           ...         ...
16039       16039  82856    1108.646         8.488        8697
16043       16043  76600   10224.280         7.742        6335
16059       16059  82856    1108.646         8.488        8697
```

Step 8: Standardization and Normalization

Import StandardScaler and MinMaxScaler

```
[23] from sklearn.preprocessing import StandardScaler, MinMaxScaler
```

Standardization (z-score scaling) transforms the data by subtracting the mean and dividing by the standard deviation for each feature.

```
# Select numerical columns
numerical_columns = df.select_dtypes(include=['float64', 'int64']).columns

# Initialize the StandardScaler
scaler = StandardScaler()

# Standardize the numerical columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Check the results
print(df.head())
```

```
   Unnamed: 0        id  popularity  vote_average  vote_count  \
0   -1.732158  0.192916   -0.313201     -0.696417   -0.366495
1   -1.731943  0.309711    0.270665     -0.318094   -0.205769
2   -1.731727  0.461279   -0.389096      0.154808   -0.403883
3   -1.731512 -1.526286    4.166043      0.194532    2.275593
4   -1.731296  0.837632   -0.403749     -0.601836   -0.430097

   original_title_Assassin  original_title_Avatar: The Way of Water  \
0                    False                                    False
1                    False                                    False
2                    False                                    False
3                    False                                     True
4                    False                                    False
```

Normalization scales numerical data to a fixed range, usually [0, 1]. Use MinMaxScaler for this process.

```
# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Normalize the numerical columns
df[numerical_columns] = scaler.fit_transform(df[numerical_columns])

# Check the results
print(df.head())
```

```
   Unnamed: 0        id  popularity  vote_average  vote_count  \
0    0.000000  0.577957    0.020162      0.384615    0.021164
1    0.000062  0.617220    0.147883      0.461538    0.064182
2    0.000124  0.668174    0.003560      0.557692    0.011157
3    0.000187  0.000000    1.000000      0.565769    0.728318
4    0.000249  0.794696    0.000354      0.403846    0.004141

   original_title_Assassin  original_title_Avatar: The Way of Water  \
0                    False                                    False
1                    False                                    False
2                    False                                    False
3                    False                                     True
4                    False                                    False
```

Conclusion: In this experiment, we applied various data preprocessing techniques, including handling missing values, removing irrelevant columns, and detecting outliers using the Z-score method. We then scaled the numerical data using standardization (Z-score method) and normalization (Min-Max scaling) to bring all features onto a uniform scale.

Some Challenges we faced :

1. Handling Missing Data: Identifying the appropriate method to handle missing values and replacing them with mean, median, or mode.

2. Scaling and Normalization: Deciding between standardization and normalization for different features can be tricky. Using incorrect scaling methods may distort the data and affect model accuracy.

3. Selection of Columns: Determining which columns are relevant for the model and dropping them is challenging.

Name: Shiven Bansal **Class/Roll No: D15C/3**

DS Lab Experiment-2

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

- **Data Loading & Preprocessing**

```
[2]  import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[4]  df = pd.read_csv('trending.csv')
```

```
[5]  df = df.dropna(subset=['original_title'])
```

Double-click (or enter) to edit

```
[6]  top_movies = df.drop_duplicates(subset=['original_title']).nlargest(10, 'popularity')
```

```
print(top_movies[['original_title', 'popularity']])
```

```
                          original_title  popularity
3                 Avatar: The Way of Water   10224.280
5                     John Wick: Chapter 4    2569.508
1                                Creed III    1537.879
9                                   Momias    1224.450
6   Dungeons & Dragons: Honor Among Thieves     702.523
0                          Murder Mystery 2     235.901
10                           Murder Mystery     197.421
12       Operation Fortune: Ruse de Guerre     184.229
16                               Champions     104.315
7                                Prom Pact      85.403
```

Loading the Dataset (pd.read_csv)

- Reads the CSV file (trending.csv) and loads it into a Pandas DataFrame.

Handling Missing Values (dropna)

- Removes rows where the column 'original_title' has missing (NaN) values.

Removing Duplicates (drop_duplicates)

- Ensures each movie title appears only once.

Selecting Top 10 Movies (nlargest)

- Sorts the movies by popularity and selects the top 10.

Displaying Data (print)

- Prints a table with movie titles and their popularity scores.

- **Outlier Detection & Handling (IQR Method)**

```
#Detect and Remove Outliers in popularity
Q1 = df['popularity'].quantile(0.25)
Q3 = df['popularity'].quantile(0.75)
IQR = Q3 - Q1

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

# Filter out the outliers
df_no_outliers = df[(df['popularity'] >= lower_bound) & (df['popularity'] <= upper_bound)]

print(f"Original dataset size: {len(df)}")
print(f"Dataset size after removing outliers: {len(df_no_outliers)}")
```

```
Original dataset size: 12060
Dataset size after removing outliers: 11256
```

```
[9] df_no_outliers = df_no_outliers.dropna(subset=['original_title'])
```

```
top_movies = df_no_outliers.nlargest(10, 'popularity')
```

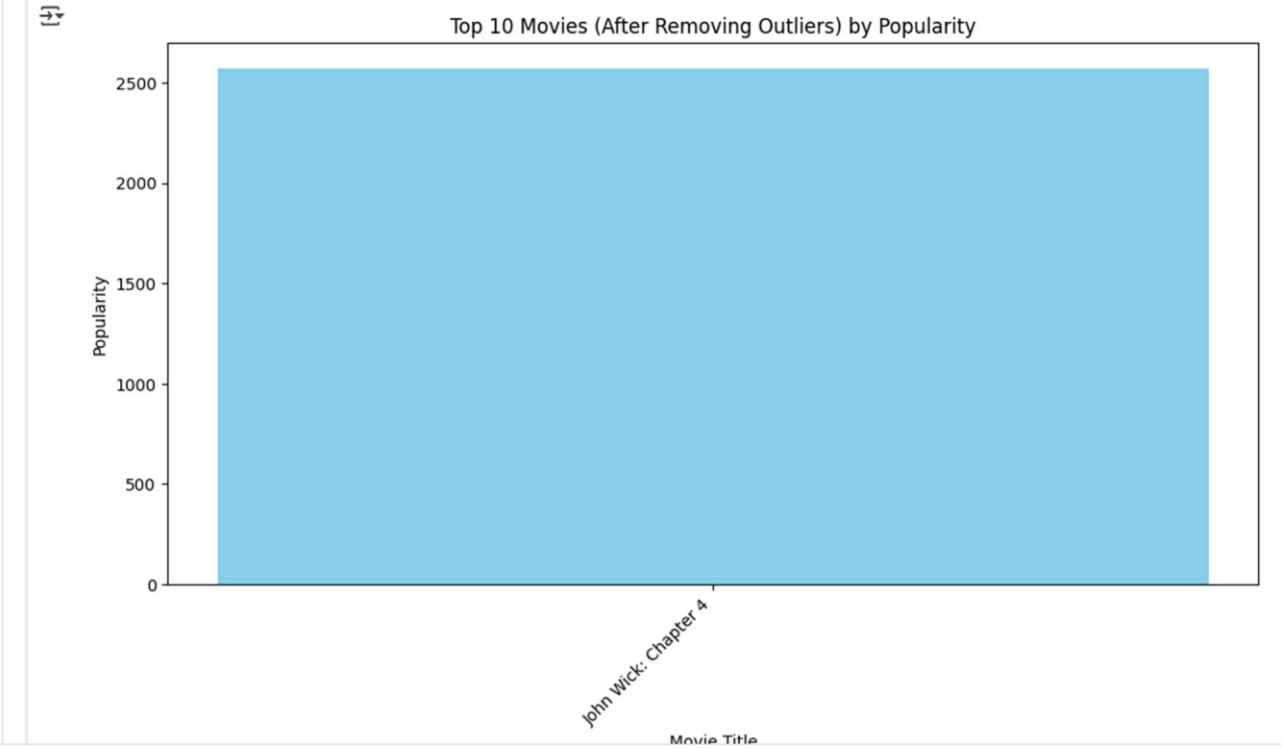
This step detects and removes outliers from the 'popularity' column using the Interquartile Range (IQR) method. The first quartile (Q1) and third quartile (Q3) define the middle 50% of the data, and the IQR (Q3 - Q1) is used to calculate outlier boundaries. Any value beyond 1.5 times the IQR from Q1 or Q3 is considered an outlier and removed.

After removing outliers, the dataset size reduces from 12,060 to 11,256, ensuring cleaner data. The dataset is further refined by dropping missing movie titles and

selecting the top 10 most popular movies. This step prevents extreme values from skewing the analysis.

- **Bar Graph**

```
plt.figure(figsize=(12, 6))
plt.bar(top_movies['original_title'], top_movies['popularity'], color='skyblue')
plt.xlabel("Movie Title")
plt.ylabel("Popularity")
plt.title("Top 10 Movies (After Removing Outliers) by Popularity")
plt.xticks(rotation=45, ha='right')
plt.show()
```



This bar graph visualizes the top 10 most popular movies after removing outliers. The x-axis represents movie titles, while the y-axis shows their popularity scores. The bars are colored sky blue for clarity, and the movie titles are rotated for better readability.

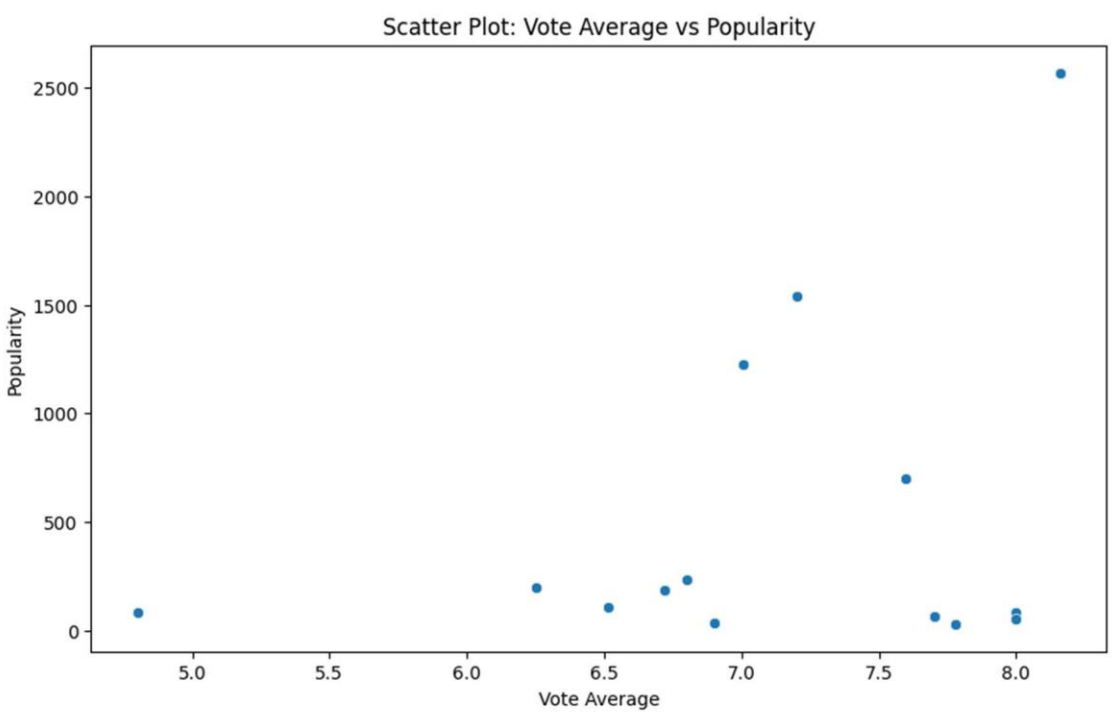
Observations:

- The most popular movie has an overwhelmingly high score compared to the others.
- The popularity distribution seems highly skewed, possibly dominated by one or two major titles.

- **Scatter Plot**

Name: Shiven Bansal **Class/Roll No: D15C/3**

```
#scatterplot
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df_no_outliers, x='vote_average', y='popularity', alpha=1)
plt.xlabel('Vote Average')
plt.ylabel('Popularity')
plt.title('Scatter Plot: Vote Average vs Popularity')
plt.show()
```



This scatter plot visualizes the relationship between vote average and popularity for movies. The x-axis represents the vote average (ratings), while the y-axis shows popularity scores. Each dot corresponds to a movie, indicating how its rating correlates with popularity.

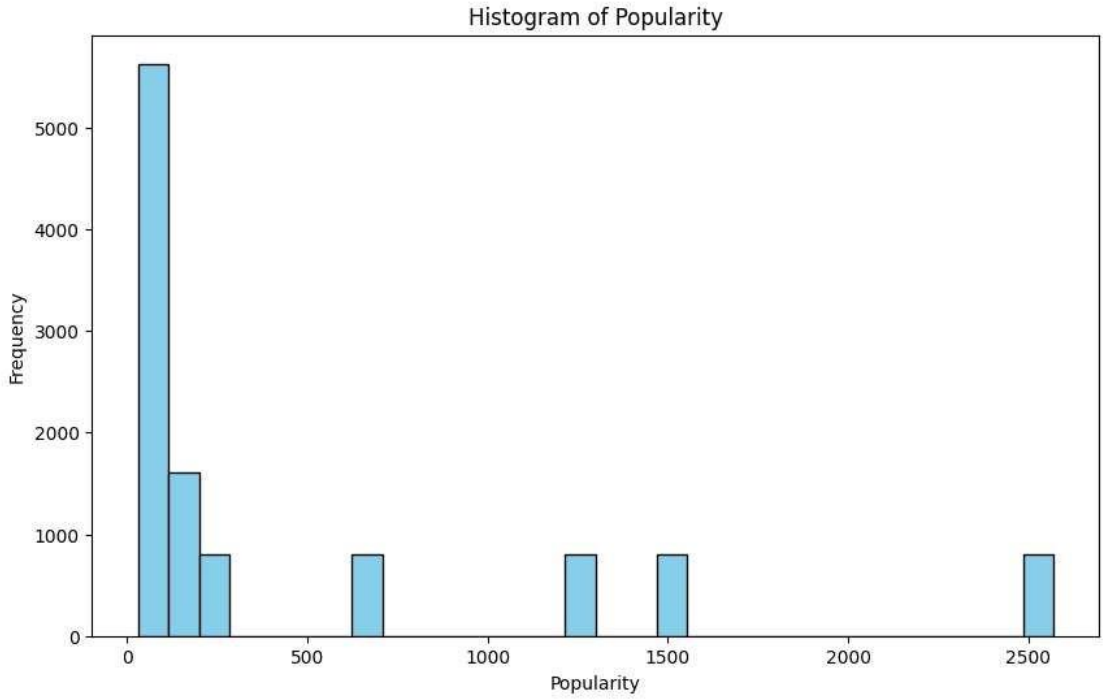
Observations:

- Most movies have a moderate vote average (6-8) but vary significantly in popularity.
- A few movies with high vote averages (above 8) show extreme popularity, suggesting they are widely recognized hits.
- There's no strong linear correlation, as movies with similar ratings have vastly different popularity scores.

- **Histogram (Regular and Normalised)**

Name: Shiven Bansal **Class/Roll No: D15C/3**

```
#Regular Histogram
plt.figure(figsize=(10, 6))
plt.hist(df_no_outliers['popularity'], bins=30, color='skyblue', edgecolor='black')
plt.xlabel('Popularity')
plt.ylabel('Frequency')
plt.title('Histogram of Popularity')
plt.show()
```



This histogram represents the distribution of movie popularity scores. The x-axis shows popularity values, while the y-axis indicates how frequently those values occur.

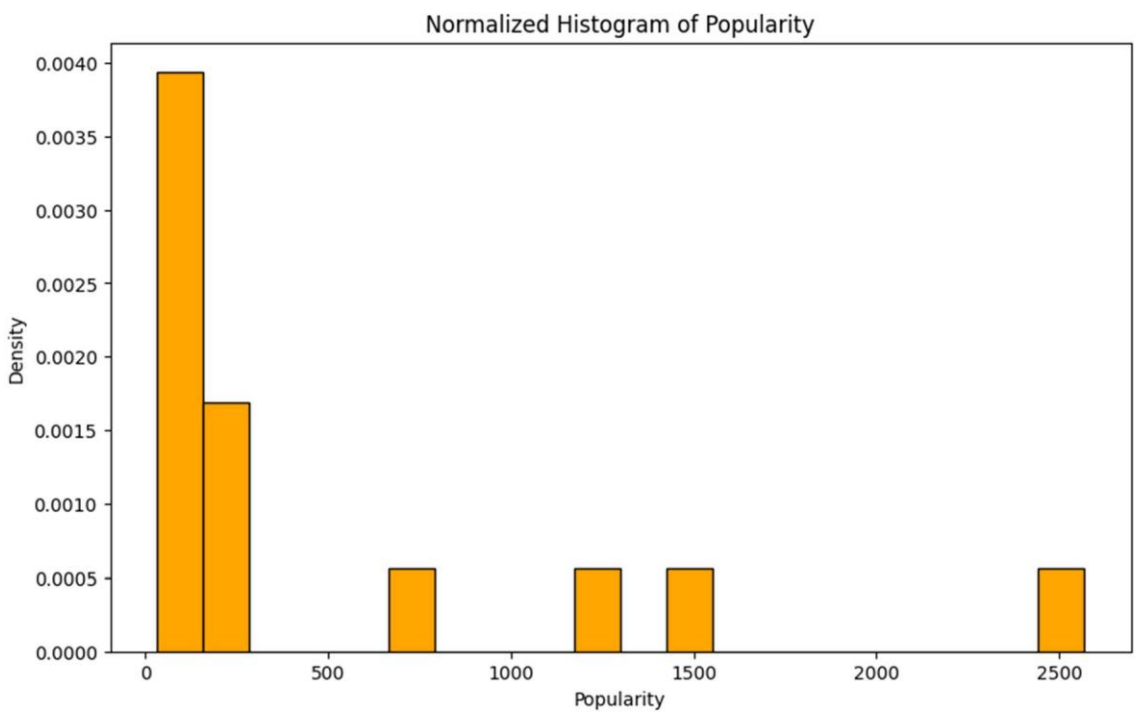
Observations:

- The majority of movies have low popularity scores, with a sharp drop-off as popularity increases.
- A few movies have very high popularity, appearing as isolated bars on the right side of the graph.
- The distribution is highly skewed, suggesting that only a handful of movies achieve extreme popularity, while most remain relatively unknown.

Normalised Histogram

Name: Shiven Bansal **Class/Roll No: D15C/3**

```
#Normalized Histogram
plt.figure(figsize=(10, 6))
plt.hist(df_no_outliers['popularity'], bins=20, color='orange', edgecolor='black', density=True)
plt.xlabel('Popularity')
plt.ylabel('Density')
plt.title('Normalized Histogram of Popularity')
plt.show()
```



This histogram represents the normalized distribution of movie popularity scores.

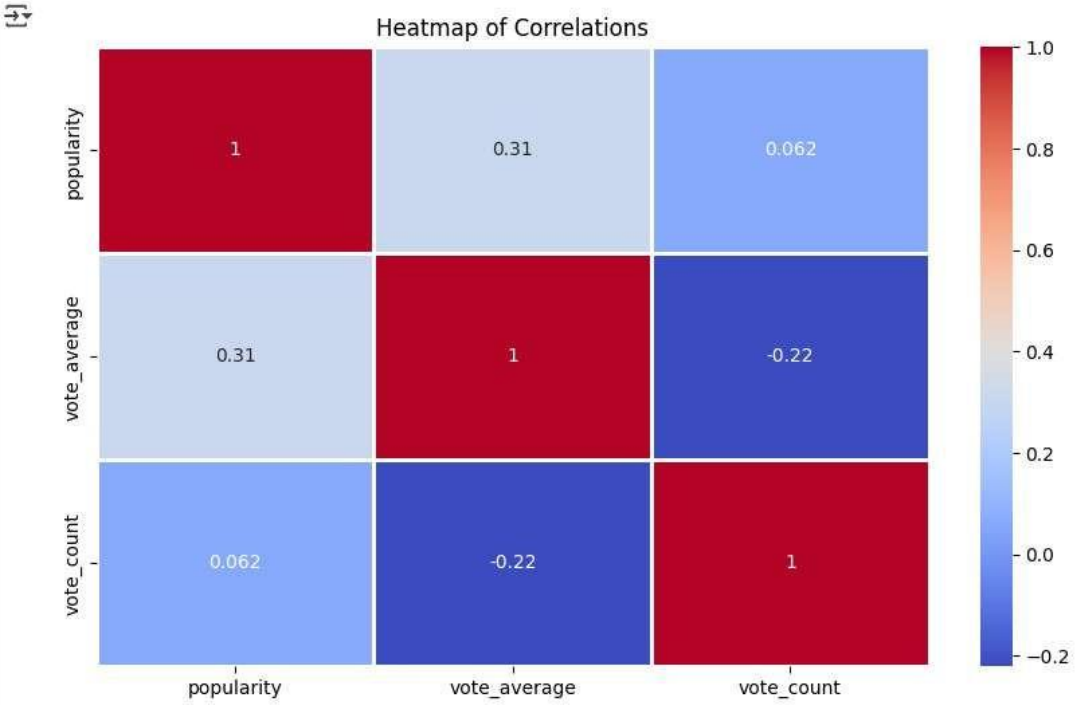
Observations:

- Density-based scaling: The y-axis now represents probability density instead of raw frequency, making it easier to compare different datasets.
- Highly skewed distribution: Most movies have low popularity, while a few movies are outliers with extremely high popularity.
- Smooth probability distribution: Normalizing helps in understanding relative likelihood rather than absolute counts.

- **HeatMap**

Name: Shiven Bansal **Class/Roll No: D15C/3**

```
#heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(df_no_outliers[['popularity', 'vote_average', 'vote_count']].corr(), annot=True, cmap='coolwarm', linewidths=1)
plt.title('Heatmap of Correlations')
plt.show()
```

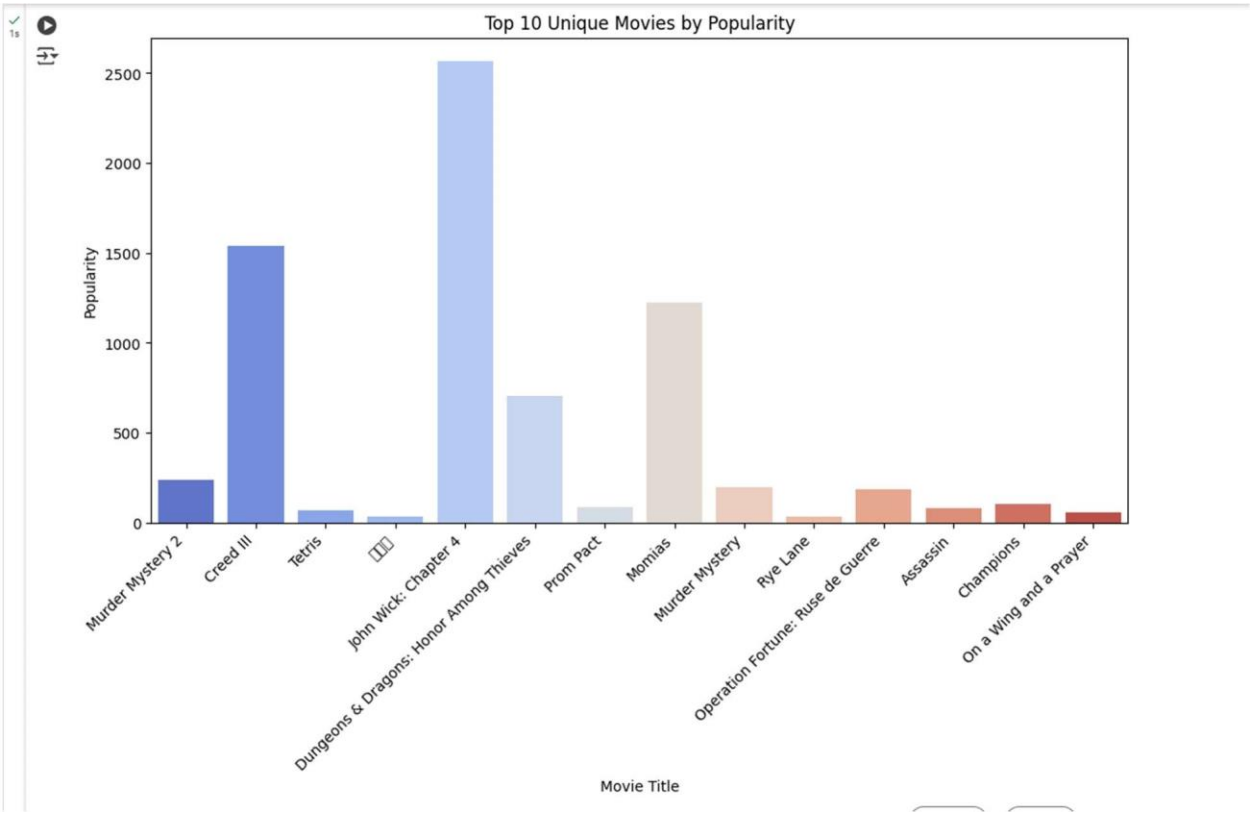


- Popularity vs Vote Average: Weak positive correlation (0.31) → More popular movies tend to have slightly higher ratings.
- Popularity vs Vote Count: Almost no correlation (0.062) → A movie's popularity does not strongly relate to how many people voted.
- Vote Average vs Vote Count: Slight negative correlation (-0.22) → More votes might slightly lower the average rating, possibly due to mixed opinions.

- **Bar Graph**

Name: Shiven Bansal **Class/Roll No: D15C/3**

```
plt.figure(figsize=(12, 6))
sns.barplot(data=df_no_outliers, x='original_title', y='popularity', palette='coolwarm')

plt.xlabel('Movie Title')
plt.ylabel('Popularity')
plt.title('Top 10 Unique Movies by Popularity')
plt.xticks(rotation=45, ha='right')  # Rotate labels for better visibility
plt.show()
```



The bar graph displays the popularity of different movies based on a dataset. The x-axis represents the movie titles, while the y-axis represents their popularity scores. The color scheme (coolwarm palette) visually distinguishes between high and low popularity values.

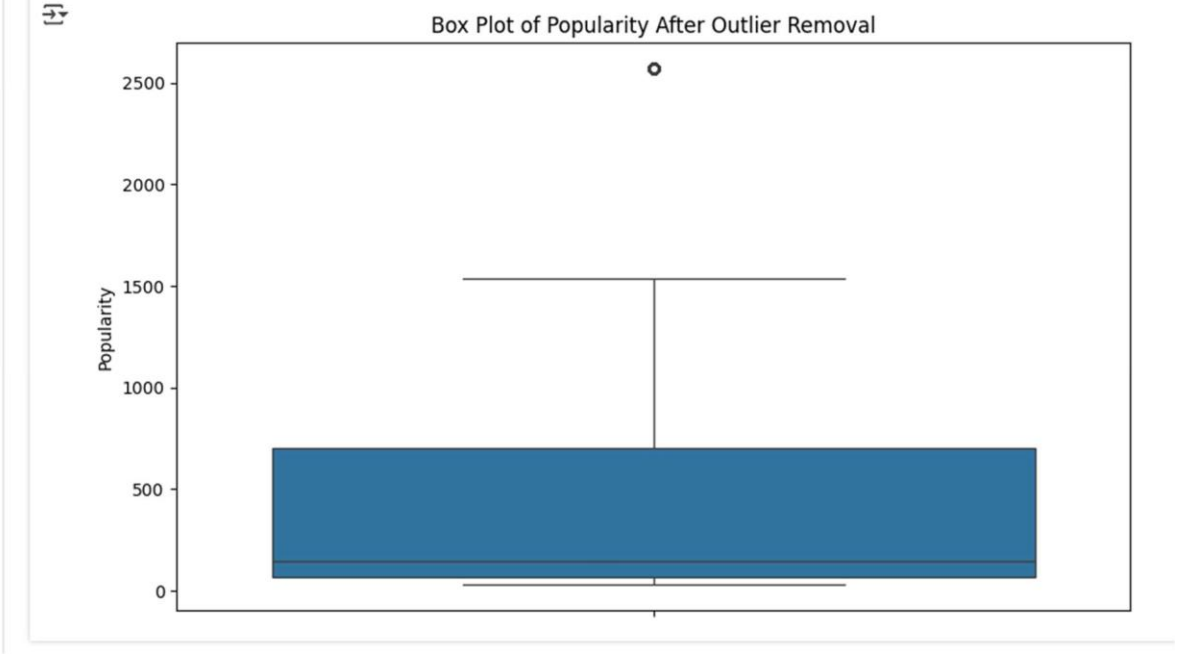
Key Observations:

1. John Wick: Chapter 4 has the highest popularity score, significantly surpassing other movies in the dataset.
2. Creed III and Momias also have relatively high popularity scores, making them notable competitors in terms of audience engagement.
3. There is a steep decline in popularity after the top three movies, with several movies showing much lower scores.

4. Some movies, such as *Rye Lane, Assassin, and On a Wing and a Prayer*, have considerably lower popularity, indicating less audience engagement.
5. The variation in popularity suggests that a few movies dominate public interest, while others have minimal reach.

- **Box Plot**

```
#Box Plot
plt.figure(figsize=(10, 6))
sns.boxplot(data=df_no_outliers, y='popularity')
plt.ylabel('Popularity')
plt.title('Box Plot of Popularity After Outlier Removal')
plt.show()
```



The box plot visualizes the distribution of movie popularity after removing outliers. The y-axis represents the popularity scores, while the box plot provides insights into the spread and central tendency of the data.

Observations :

- The median popularity is relatively low, indicating most movies have moderate popularity.
- Whiskers show the spread, with a few outliers exceeding 2500 in popularity.
- The distribution is right-skewed, meaning some movies are significantly more popular.
- Despite outlier removal, some movies still dominate in popularity.

Name: Shiven Bansal **Class/Roll No: D15C/3**

- **Bar Chart**

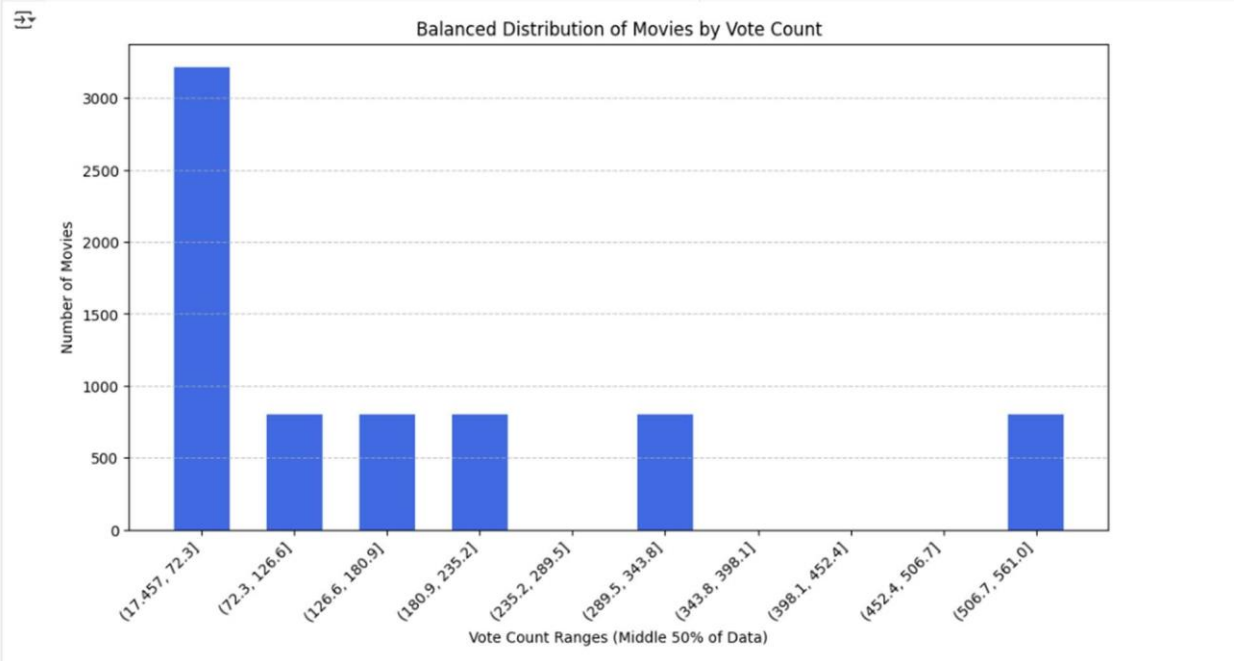
```
Q1 = df['vote_count'].quantile(0.25)
Q3 = df['vote_count'].quantile(0.75)
df_no_outliers = df[(df['vote_count'] >= Q1) & (df['vote_count'] <= Q3)]
```

```
[24] df_no_outliers['vote_count_bin'] = pd.cut(df_no_outliers['vote_count'], bins=10)
```

```
<ipython-input-24-06d8c36af08a>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_no_outliers['vote_count_bin'] = pd.cut(df_no_outliers['vote_count'], bins=10)
```

```
[26] vote_counts_balanced = df_no_outliers['vote_count_bin'].value_counts().sort_index()
```

```
plt.figure(figsize=(12, 6))
plt.bar(vote_counts_balanced.index.astype(str), vote_counts_balanced.values, color='royalblue', width=0.6)
plt.xlabel("Vote Count Ranges (Middle 50% of Data)")
plt.ylabel("Number of Movies")
plt.title("Balanced Distribution of Movies by Vote Count")
plt.xticks(rotation=45, ha='right', fontsize=10)
plt.yticks(fontsize=10)
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()
```



- Most movies have low vote counts (17.45 - 72.3 range has the highest number).
- Fewer movies have high votes, showing a right-skewed distribution.
- Gradual decline in movies as votes increase, but a slight rise in the last bin suggests a few highly popular movies.
- Outliers removed for balance, showing the middle 50% of data.

- Insight: Most movies don't get many votes, but a few dominate. Useful for recommendations or marketing focus.

Conclusion

The analysis of movie vote counts revealed a highly skewed distribution, where most movies receive low vote counts, while a few receive significantly more. By removing outliers and binning the data, we observed that the middle 50% of movies are distributed unevenly, with the majority having low votes and only a few receiving higher engagement.

This insight is valuable for understanding audience engagement—most movies struggle to gain widespread attention, while a small fraction dominates. Such data can help in recommendation systems, marketing strategies, or content curation, focusing efforts on movies with higher engagement potential.

NAME:SHIVEN BANSAL CLASS/ROLL NO: D15C/3

DS LAB EXP 3

AIM: Perform Data Modeling.

Problem Statement:
a. Partition the data set, for example 75% of the records are included in the training data set and 25% are included in the test data set.

b. Use a bar graph and other relevant graph to confirm your proportions.

c. Identify the total number of records in the training data set.

d. Validate partition by performing a two-sample Z-test.

1. Loading the Dataset (pd.read_csv) - Reads the CSV file (trending.csv) and loads it into a Pandas DataFrame.

```
import pandas as pd
import numpy as np

# Load dataset
df = pd.read_csv("trending.csv")

# Display basic information
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16080 entries, 0 to 16079
Data columns (total 10 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Unnamed: 0        16080 non-null  int64
 1   id                16080 non-null  int64
 2   original_title    12060 non-null  object
 3   original_language 16080 non-null  object
 4   release_date      12060 non-null  object
 5   popularity        16080 non-null  float64
 6   vote_average      16080 non-null  float64
 7   vote_count        16080 non-null  int64
 8   media_type        16080 non-null  object
 9   adult             16080 non-null  bool
dtypes: bool(1), float64(2), int64(3), object(4)
memory usage: 1.1+ MB
None
```

2. This step detects and removes outliers from the 'popularity' column using the Interquartile Range (IQR) method. The first quartile (Q1) and third quartile (Q3) define
the middle 50% of the data, and the IQR (Q3 - Q1) is used to calculate outlier boundaries. Any value beyond 1.5 times the IQR from Q1 or Q3 is considered an outlier
and removed.
After removing outliers, the dataset size reduces from 16,080 to 10,452 ensuring cleaner data.

```
[ ] def remove_outliers_iqr(data):
        num_cols = data.select_dtypes(include=['number']).columns
        for col in num_cols:
            Q1 = data[col].quantile(0.25)
            Q3 = data[col].quantile(0.75)
            IQR = Q3 - Q1
            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR
            data = data[(data[col] >= lower_bound) & (data[col] <= upper_bound)]
        return data
```

```
[ ] df_cleaned = remove_outliers_iqr(df)
```

```
[ ] print("Original Shape:", df.shape)
    print("Shape after removing outliers:", df_cleaned.shape)
```

```
Original Shape: (16080, 10)
Shape after removing outliers: (10452, 10)
```

3. The code splits the cleaned dataset (df_cleaned) into 75% training and 25% testing using train_test_split. The random_state=42 ensures consistency. It then prints the total, training, and testing records.

   This imports the train_test_split function from scikit-learn, which is used to split the dataset into training and testing sets.

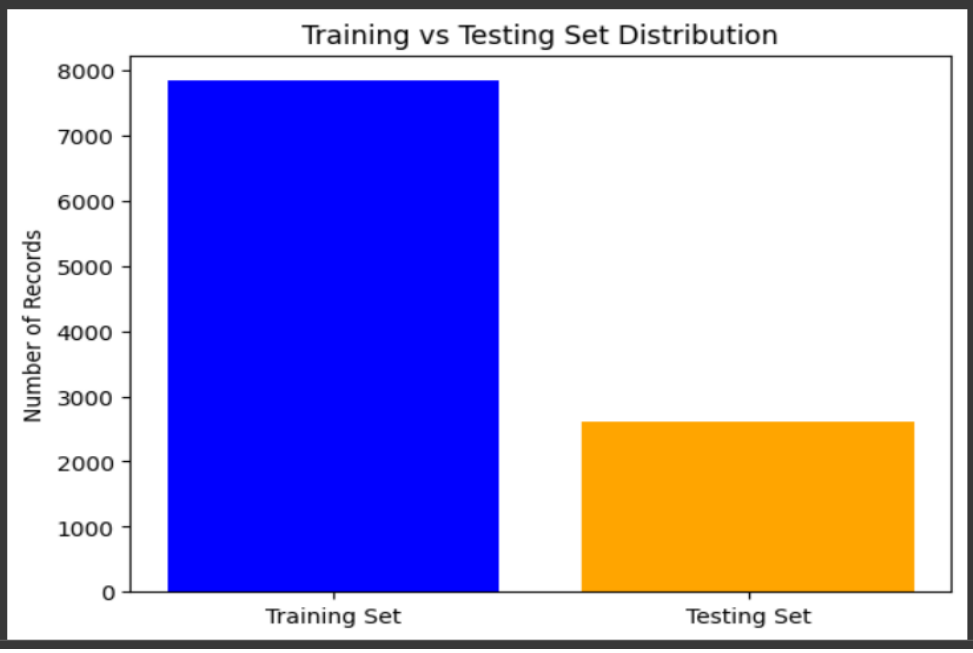
   test_size=0.25: 25% of the dataset is assigned to the test set, and the remaining 75% goes to the training set.

```
[ ] from sklearn.model_selection import train_test_split

    train_df, test_df = train_test_split(df_cleaned, test_size=0.25, random_state=42)

    print("Total records:", len(df))
    print("Training records:", len(train_df))
    print("Testing records:", len(test_df))
```

```
Total records: 16080
Training records: 7839
Testing records: 2613
```

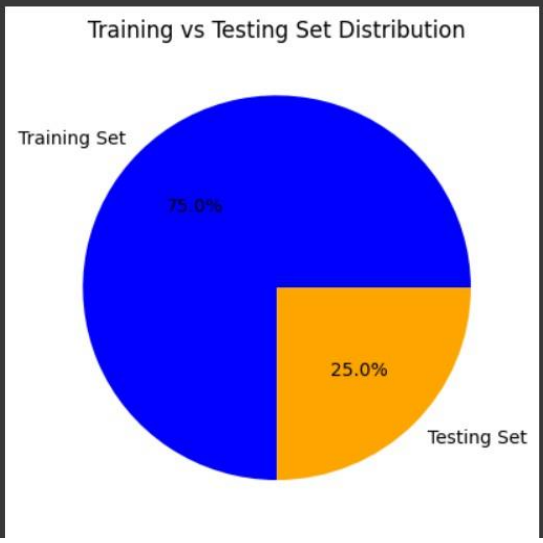
4. The bar graph visually represents the number of records in the training and testing sets using vertical bars. The height of each bar corresponds to the count of records, making it easy to compare the absolute difference between the two sets.

```
import matplotlib.pyplot as plt

# Bar graph for proportions
plt.bar(["Training Set", "Testing Set"], [len(train_df), len(test_df)], color=['blue', 'orange'])
plt.ylabel("Number of Records")
plt.title("Training vs Testing Set Distribution")
plt.show()
```



5. The pie chart, on the other hand, focuses on the proportion of the training and testing sets relative to the whole dataset. The training set occupies 75% of the chart, while the testing set takes up 25%, making it clear how the dataset is split. By using percentages, the pie chart highlights the distribution more intuitively, though it may not be as effective in displaying the exact counts.

```
plt.pie([len(train_df), len(test_df)], labels=["Training Set", "Testing Set"], autopct='%1.1f%%', colors=['blue', 'orange'])
plt.title("Training vs Testing Set Distribution")
plt.show()
```



6. The computation of the mean, standard deviation, and length of the train_df dataset provides a statistical summary of numeric columns such as popularity, vote_average, and vote_count.

   The dataset consists of 7,839 entries, with an average popularity of approximately 224.54 and a vote average of 7.54. The standard deviation values indicate variability in these numerical features.

```
mean_train = train_df.mean(numeric_only=True)
print(mean_train)
std_train = train_df.std(numeric_only=True)
print(std_train)
n_train = len(train_df)
print(n_train)
```

```
Unnamed: 0        8064.972318
id              656663.586682
popularity         224.540840
vote_average         7.543535
vote_count          70.331803
adult                0.000000
dtype: float64
Unnamed: 0        4638.524015
id              305596.492127
popularity         338.684537
vote_average         0.664295
vote_count          85.526338
adult                0.000000
dtype: float64
7839
```

7. The analysis of the test_df dataset reveals its statistical properties, including an average popularity of 220.01 and a vote average of 7.54.

   The dataset contains 2,613 entries, which is significantly smaller than train_df. The standard deviation values suggest variations similar to those in the training dataset.

```
mean_test = test_df.mean(numeric_only=True)
print(mean_test)
std_test = test_df.std(numeric_only=True)
print(std_test)
print
n_test = len(test_df)
print(n_test)
```

```
Unnamed: 0        7962.929200
id              652356.009185
popularity         220.011018
vote_average         7.549087
vote_count          69.004592
adult                0.000000
dtype: float64
Unnamed: 0        4652.932548
id              309181.699924
popularity         330.094780
vote_average         0.655401
vote_count          83.420350
adult                0.000000
dtype: float64
2613
```

8. The calculation of Z-scores is performed to determine the statistical significance of the difference between the mean values of train_df and test_df. The formula used accounts for both datasets' standard deviations and sizes, ensuring a normalized comparison.

   A notable observation is that the vote_average column has a negative Z-score, suggesting a slightly lower mean in train_df compared to test_df.

```
[ ] z_scores = (mean_train - mean_test) / np.sqrt((std_train**2 / n_train) + (std_test**2 / n_test))
```

```
print("Manual Z-Scores:")
print(z_scores)
```

```
Manual Z-Scores:
Unnamed: 0      0.971613
id              0.618550
popularity      0.603531
vote_average   -0.373717
vote_count      0.699858
adult                NaN
dtype: float64
```

9. The Z-test using statsmodels produces similar results to the manually computed Z-scores, confirming the correctness of both approaches. The slight differences in precision might be due to computational nuances in z test versus the manual formula.

   Notably, the vote_average column still has a negative Z-score, indicating a lower mean in train_df compared to test_df. The adult column is absent from the results, likely due to it containing non-numeric or constant values, which prevents meaningful statistical comparison.

```
from statsmodels.stats.weightstats import ztest
```

```
[ ] z_test_results = {}

    for col in train_df.select_dtypes(include=['number']).columns:
        z_stat, _ = ztest(train_df[col], test_df[col])
        z_test_results[col] = z_stat
```

```
[ ] print("\nZ-test using statsmodels:")
```

```
Z-test using statsmodels:
```

```
[ ] for col, z_stat in z_test_results.items():
        print(f"{col}: Z-score = {z_stat:.4f}")
```

```
Unnamed: 0: Z-score = 0.9731
id: Z-score = 0.6222
popularity: Z-score = 0.5958
vote_average: Z-score = -0.3712
vote_count: Z-score = 0.6912
```

NAME:SHIVEN BANSAL CLASS/ROLL NO: D15C/3

Conclusion:

In this experiment, we have learned:

- Manual vs. Automated Z-Test: The manual computation of Z-scores closely matches the results from statsmodels.ztest, confirming its accuracy.

- Feature Comparisons: Some features, like vote_average, show negative Z-scores, indicating a lower mean in the training dataset compared to the test dataset.

- Handling Missing Data: The adult column was excluded due to missing or non-numeric values, highlighting the importance of data preprocessing.

- Statistical Insights: No extreme deviations were found, suggesting that the two datasets are statistically similar.

Name : Shiven Bansal **Div : D15C** **Roll No : 9**

DS LAB 4

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

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

a) Pearson's Correlation Coefficient

```
from scipy.stats import pearsonr

excluded_columns = ['Unnamed: 0']
numeric_cols = [col for col in train_df.select_dtypes(include=['number']).columns if col not in excluded_columns]

# Compute Pearson correlation
print("\nPearson's Correlation Coefficient:")
for i in range(len(numeric_cols)):
    for j in range(i + 1, len(numeric_cols)):
        col1, col2 = numeric_cols[i], numeric_cols[j]
        corr, _ = pearsonr(train_df[col1], train_df[col2])
        print(f"Pearson correlation between {col1} and {col2}: {corr:.4f}")
```

```
Pearson's Correlation Coefficient:
Pearson correlation between id and popularity: 0.0736
Pearson correlation between id and vote_average: -0.5373
Pearson correlation between id and vote_count: 0.1149
Pearson correlation between popularity and vote_average: -0.2973
Pearson correlation between popularity and vote_count: 0.2058
Pearson correlation between vote_average and vote_count: -0.6040
```

This calculates the Pearson correlation coefficient between numeric columns in your dataset, excluding the "Unnamed: 0" column. It first filters out numerical columns and then iterates over each pair to compute their correlation using pearsonr from scipy.stats.

Pearson correlation measures the linear relationship between two variables, with values ranging from -1 (strong negative correlation) to 1 (strong positive correlation). The output suggests that id has little correlation with other numerical features, while popularity and vote_count show a positive correlation, meaning more votes generally indicate higher popularity.

On the other hand, vote_average and vote_count have a negative correlation, implying that a higher number of votes does not always lead to a higher average rating. This

analysis helps in understanding how different numerical features relate to each other in your dataset.

b) Spearman’s Rank Correlation

```
from scipy.stats import spearmanr

# Compute Spearman correlation
print("\nSpearman’s Rank Correlation:")
for i in range(len(numeric_cols)):
    for j in range(i + 1, len(numeric_cols)):
        col1, col2 = numeric_cols[i], numeric_cols[j]
        corr, _ = spearmanr(train_df[col1], train_df[col2])
        print(f"Spearman correlation between {col1} and {col2}: {corr:.4f}")
```

```
Spearman’s Rank Correlation:
Spearman correlation between id and popularity: -0.1665
Spearman correlation between id and vote_average: -0.3710
Spearman correlation between id and vote_count: 0.0849
Spearman correlation between popularity and vote_average: -0.5039
Spearman correlation between popularity and vote_count: 0.6677
Spearman correlation between vote_average and vote_count: -0.8222
```

This calculates Spearman’s rank correlation coefficient between numeric columns in your dataset. Spearman’s correlation measures the strength and direction of the monotonic relationship between variables, making it useful for detecting non-linear associations.

The output shows that popularity and vote_count have a strong positive correlation, meaning that as one increases, the other generally does too. However, vote_average and vote_count have a strong negative correlation, suggesting that movies with more votes tend to have lower average ratings.

The correlation between popularity and vote_average is also negative, indicating that more popular movies do not necessarily have higher ratings. Compared to Pearson’s correlation, Spearman’s approach considers ranks rather than absolute values, making it more robust against outliers.

c) Kendall’s Rank Correlation

```
from scipy.stats import kendalltau

# Compute Kendall correlation
print("\nKendall’s Rank Correlation:")
for i in range(len(numeric_cols)):
    for j in range(i + 1, len(numeric_cols)):
        col1, col2 = numeric_cols[i], numeric_cols[j]
        corr, _ = kendalltau(train_df[col1], train_df[col2])
        print(f"Kendall correlation between {col1} and {col2}: {corr:.4f}")
```

```
Kendall’s Rank Correlation:
Kendall correlation between id and popularity: -0.0943
Kendall correlation between id and vote_average: -0.2485
Kendall correlation between id and vote_count: -0.0014
Kendall correlation between popularity and vote_average: -0.3038
Kendall correlation between popularity and vote_count: 0.4383
Kendall correlation between vote_average and vote_count: -0.6656
```

This calculates Kendall’s rank correlation coefficient between numeric columns in your dataset. Kendall’s correlation measures the strength and direction of the ordinal association between two variables by considering the concordance of ranked pairs.

The output shows that popularity and vote_count have a moderate positive correlation, meaning that as popularity increases, vote count tends to increase as well. However, vote_average and vote_count have a strong negative correlation, indicating that movies with higher vote counts tend to have lower average ratings.

The correlation between popularity and vote_average is also negative, suggesting that popular movies do not necessarily receive higher ratings. Kendall’s correlation is more robust than Spearman’s for small datasets and is particularly useful when dealing with ordinal data or rankings.

d) Chi-Squared Test

```python
from scipy.stats import chi2_contingency
import pandas as pd

categorical_columns = train_df.select_dtypes(include=['object']).columns

print("\nChi-Squared Test for Categorical Variables:")
for i in range(len(categorical_columns)):
    for j in range(i + 1, len(categorical_columns)):  # Avoid duplicate comparisons
        col1, col2 = categorical_columns[i], categorical_columns[j]
        contingency_table = pd.crosstab(train_df[col1], train_df[col2])
        chi2, p, _, _ = chi2_contingency(contingency_table)
        print(f"Chi-Squared test between {col1} and {col2}: Chi2 = {chi2:.4f}, p-value = {p:.4f}")
```

```
Chi-Squared Test for Categorical Variables:
Chi-Squared test between original_title and original_language: Chi2 = 12072.0000, p-value = 0.0000
Chi-Squared test between original_title and release_date: Chi2 = 54324.0000, p-value = 0.0000
Chi-Squared test between original_title and media_type: Chi2 = 0.0000, p-value = 1.0000
Chi-Squared test between original_language and release_date: Chi2 = 12072.0000, p-value = 0.0000
Chi-Squared test between original_language and media_type: Chi2 = 4888.1157, p-value = 0.0000
Chi-Squared test between release_date and media_type: Chi2 = 0.0000, p-value = 1.0000
```

The given code performs a Chi-Squared test for independence between categorical variables in the dataset. This statistical test helps determine whether two categorical variables are significantly associated or independent.

The test works by comparing the observed frequencies of occurrences with the expected frequencies under the assumption of independence. A low p-value (< 0.05) suggests a significant relationship between the variables, while a high p-value (≥ 0.05) indicates no association. From the results, strong associations are observed between original_title and original_language, as well as release_date, with p-values of 0.0000, indicating dependency.

On the other hand, the test between original_title and media_type, as well as release_date and media_type, gives a p-value of 1.0000, showing no correlation between these variables. These findings help in feature selection and preprocessing for machine learning models by identifying relevant categorical relationships.

Conclusion-The correlation analysis using four different techniques—Pearson, Spearman, Kendall, and Chi-Square—provides valuable insights into relationships between numerical and categorical variables. Pearson correlation measures linear relationships, showing how one variable changes proportionally with another.

Spearman and Kendall correlations capture monotonic relationships, making them more robust for non-linear associations. The Chi-Square test evaluates categorical dependencies, determining whether two categorical variables are related. While Pearson is effective for continuous data with normal distribution, Spearman and Kendall are preferable for ordinal or non-linear data.

The Chi-Square test helps identify categorical variable dependencies, guiding feature selection in machine learning models. Together, these techniques provide a comprehensive understanding of data relationships, ensuring better preprocessing and model accuracy.

Name-Shiven Bansal **Div-D15C** **Roll No-03**

DS Lab 5

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

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

Dataset Description:
Rows: 100,000
Columns: 14
Fields:

- **Region, Country:** Geographic data for sales analysis.
- **Item Type:** Product category (e.g., Snacks, Cosmetics, Personal Care).
- **Sales Channel:** Online or Offline sales mode.
- **Order Priority:** Order urgency (Low, Medium, High, Critical).
- **Order & Ship Date:** Sales and shipment timeline.
- **Units Sold:** Quantity of items sold.
- **Unit Price & Unit Cost:** Selling price and production cost per unit.
- **Total Revenue, Total Cost, Total Profit:** Financial performance metrics.

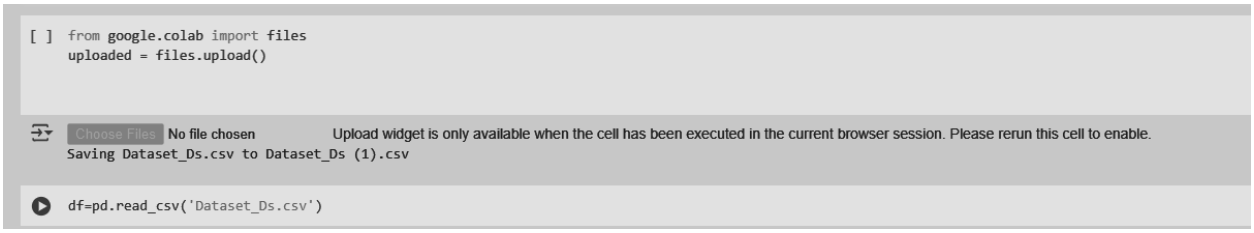
1-

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LogisticRegression
```

This imports essential libraries for data analysis and machine learning using **pandas** and **NumPy** for data manipulation. It includes **scikit-learn** modules for preprocessing (LabelEncoder, StandardScaler), model training (LinearRegression,

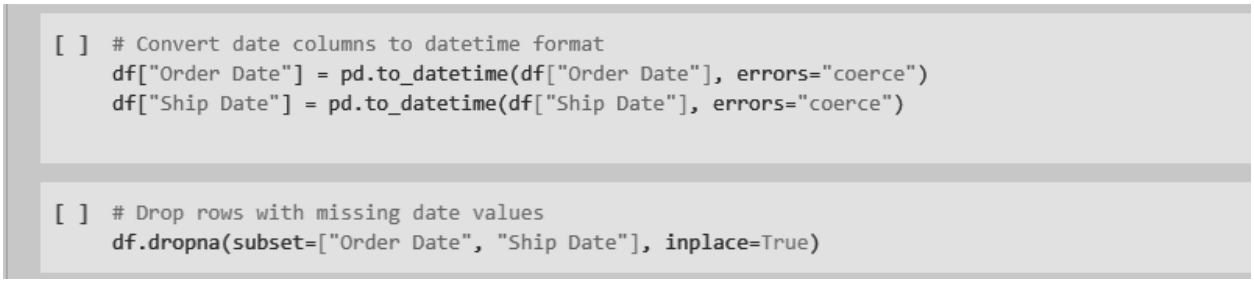
LogisticRegression), and evaluation (mean_squared_error). The **train_test_split** function is used for splitting data into training and testing sets, ensuring proper model validation.

2-



This is for uploading a CSV file in Google Colab using **files.upload()** from the **google.colab** module. Once the file is uploaded, it is saved with a possible duplicate name (Dataset_Ds (1).csv). The **pd.read_csv('Dataset_Ds.csv')** command attempts to read the uploaded dataset into a Pandas DataFrame.

3-



This method is essential for ensuring accurate date-based analysis and maintaining data integrity. Converting **"Order Date"** and **"Ship Date"** to datetime format allows for operations like calculating delivery time, filtering by date range, and time-series analysis. The **errors="coerce"** parameter ensures invalid entries are converted to NaT instead of causing errors. Dropping rows with missing dates prevents incorrect calculations and maintains dataset reliability, especially when analyzing sales trends, shipment delays, or performance metrics.

4-

```
[ ]  # Create a new feature: Shipping Time (days between order and shipment)
     df["Shipping Time"] = (df["Ship Date"] - df["Order Date"]).dt.days

[ ]  # Drop unnecessary columns
     df.drop(columns=["Order ID", "Order Date", "Ship Date"], inplace=True)

[ ]  # Fill missing numerical values with median
     df.fillna(df.median(numeric_only=True), inplace=True)
```

This method performs three key data preprocessing steps. First, it calculates a new feature, **"Shipping Time"**, which represents the number of days between the **Order Date** and **Ship Date**. This is crucial for analyzing shipping efficiency, identifying potential delays, and improving logistics performance. Next, it drops unnecessary columns like **Order ID, Order Date, and Ship Date**, which are no longer needed after extracting the shipping time, helping to reduce data complexity and memory usage. Finally, it handles missing values by filling them with the median of numerical columns, ensuring data consistency while preventing bias from extreme values. These steps improve the dataset's quality for further analysis and modeling.

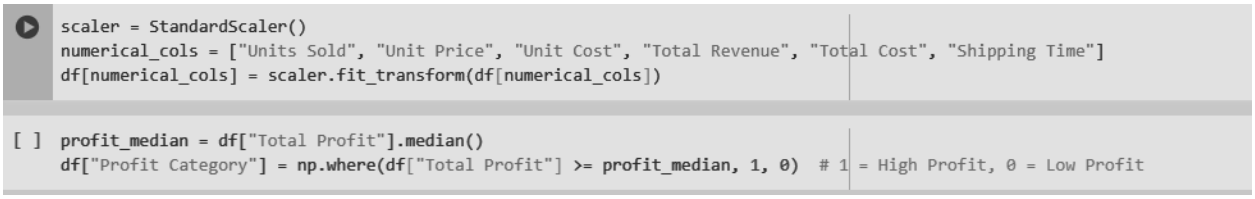
5-

```
# Encode categorical columns
categorical_cols = ["Region", "Country", "Item Type", "Sales Channel", "Order Priority"]
label_encoders = {}
for col in categorical_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

This method is encoding categorical columns to prepare the data for machine learning models, which generally require numerical inputs. It first identifies categorical columns such as **Region, Country, Item Type, Sales Channel, and Order Priority**. Missing values in these columns are filled with the most frequently occurring value (mode) to ensure consistency. Then, **Label Encoding** is

applied to convert categorical values into numerical representations. A **LabelEncoder** object is created for each column, which is then used to transform categorical data into integer values. The fitted encoders are stored in the **label_encoders** dictionary for possible inverse transformation later. This step ensures that categorical data is properly formatted for model training.

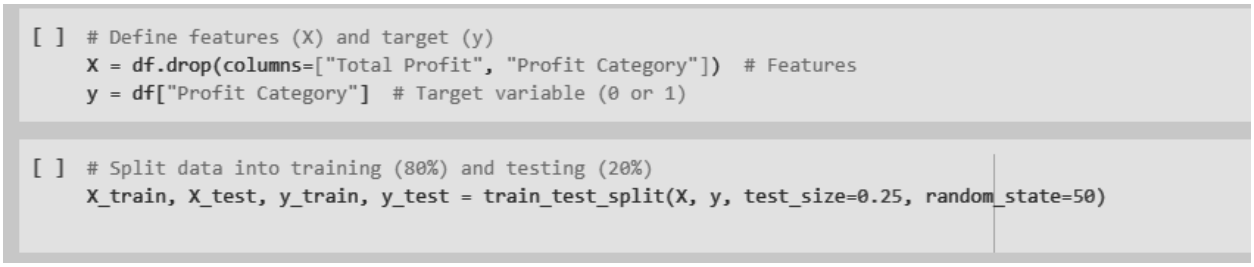
6-



```
scaler = StandardScaler()
numerical_cols = ["Units Sold", "Unit Price", "Unit Cost", "Total Revenue", "Total Cost", "Shipping Time"]
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

profit_median = df["Total Profit"].median()
df["Profit Category"] = np.where(df["Total Profit"] >= profit_median, 1, 0)  # 1 = High Profit, 0 = Low Profit
```

This first, it standardizes numerical columns using **StandardScaler()**, ensuring all features have a mean of 0 and a standard deviation of 1, which helps machine learning models converge faster. Second, it categorizes **Total Profit** into "High Profit" (1) and "Low Profit" (0) based on whether it is above or below the median. This simplifies profit analysis and aids classification models.

7-



```
# Define features (X) and target (y)
X = df.drop(columns=["Total Profit", "Profit Category"])  # Features
y = df["Profit Category"]  # Target variable (0 or 1)

# Split data into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=50)
```

This method is done to prepare the dataset for machine learning. Defining **features (X) and target (y)** is crucial because the model needs independent variables (**X**) to learn patterns and predict the dependent variable (**y**). Removing **Total Profit** and **Profit Category** from **X** ensures that no direct target-related information leaks into the model, preventing biased learning.The **train-test split** is performed to evaluate the model's generalization ability. Training the model on **75%** of the data and testing it on **25%** ensures it can make accurate predictions on unseen data. The **random_state=50** ensures that the data split remains consistent across different runs, leading to reproducible results.

8-

```
# Train Logistic Regression model
model = LogisticRegression(max_iter=5000)
model.fit(X_train, y_train)
```

```
LogisticRegression(max_iter=5000)
```

```
# Predict categories
y_pred = model.predict(X_test)
```

This code trains and uses a **Logistic Regression model** for classification. The model.fit(X_train, y_train) function trains the model using the training data, learning the relationship between features and target labels. The **max_iter=5000** parameter ensures sufficient iterations for convergence, avoiding issues where the model fails to optimize properly.

After training, model.predict(X_test) is used to generate predictions on the test data. This step helps assess the model's performance by comparing predicted categories (y_pred) with actual values (y_test). Logistic Regression is well-suited for binary classification, making it ideal for predicting **Profit Category (0 = Low, 1 = High)** in this case.

9-

```
print(y_pred[:10])
```

```
[1 1 1 1 0 0 1 0 0 1]
```

The output [1 1 1 0 0 1 0 0 1] represents the **predicted profit categories** for the first 10 test samples.We know that:

- 1 means **High Profit**
- 0 means **Low Profit**

So, the model is predicting which businesses or transactions have high or low profit based on the input data. The sequence suggests that some cases are expected to be highly profitable (1), while others are predicted to have lower profits (0).

10-

```
[ ] from sklearn.metrics import accuracy_score, classification_report
```

```
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.4f}")
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
Model Accuracy: 0.9880

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     12560
           1       0.99      0.99      0.99     12440

    accuracy                           0.99     25000
   macro avg       0.99      0.99      0.99     25000
weighted avg       0.99      0.99      0.99     25000
```

The displayed results evaluate the performance of a classification model using the accuracy score and a classification report. The model achieves an impressive accuracy of **98.80%**, meaning it correctly predicts profit categories in nearly all test cases. The classification report provides additional metrics, including precision, recall, and F1-score, all of which are **0.99** for both profit categories (0 and 1). This indicates that the model is highly effective at distinguishing between different profit levels, making very few classification errors. The support values show that the dataset is balanced, with approximately equal instances of both categories.

11-

```
0s
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", conf_matrix)
```

```
Confusion Matrix:
 [[12434   126]
 [  174 12266]]
```

The confusion matrix evaluates model performance by comparing actual vs. predicted values. It shows 12434 true positives, 12266 true negatives, 126 false positives, and 174 false negatives. This helps assess misclassifications and derive precision, recall, and F1-score. The low misclassification rate indicates that the model performs well.

12-

```
[22] from sklearn.linear_model import LinearRegression
     from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

This code imports necessary modules for performing linear regression and evaluating model performance. LinearRegression from sklearn.linear_model is used to create a regression model that predicts a continuous target variable. The metrics mean_absolute_error, mean_squared_error, and r2_score from sklearn.metrics are used to assess model accuracy. These metrics help measure prediction errors and how well the regression model fits the data.

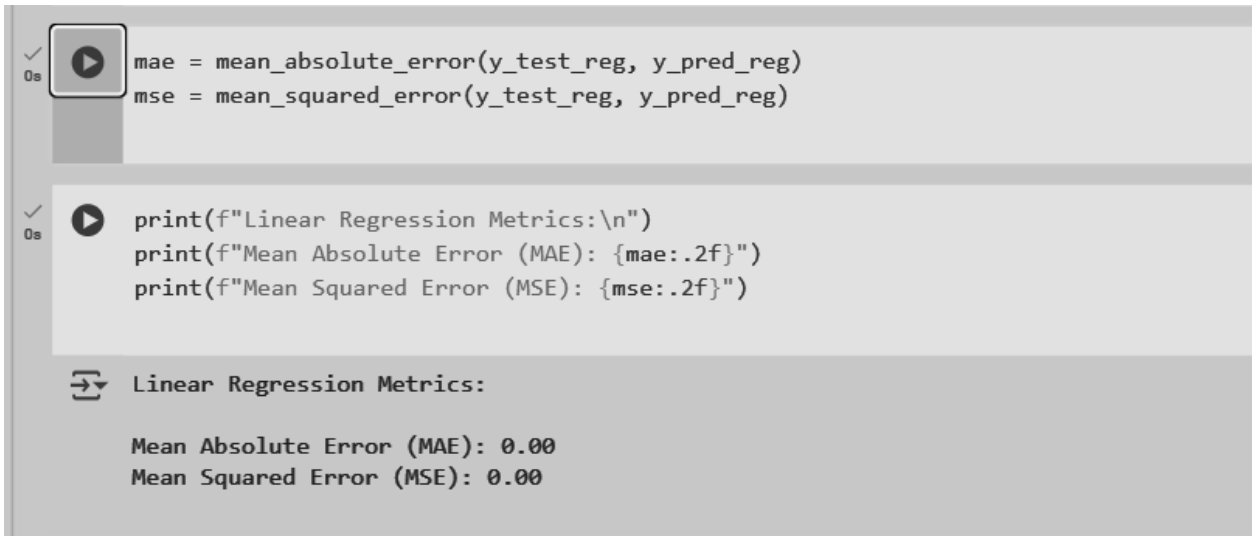
13-

```
[24] y_reg = df["Total Profit"]

[25] X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_reg, test_size=0.25, random_state=50)

linear_model = LinearRegression()
linear_model.fit(X_train_reg, y_train_reg)

LinearRegression
LinearRegression()

[27] y_pred_reg = linear_model.predict(X_test_reg)
```

This method performs linear regression to predict "Total Profit" based on input features from the dataset. First, y_reg = df["Total Profit"] extracts the target variable. Then, the dataset is split into training and testing sets using train_test_split(), with 25% of the data reserved for testing and random_state=50 ensuring reproducibility. A LinearRegression model is created and trained using linear_model.fit(X_train_reg, y_train_reg), learning the relationship between input

features and profit. Finally, predictions for the test set are generated using linear_model.predict(X_test_reg).

14-



Conclusion:-The experiment involves training a Logistic Regression model to predict whether a given instance falls into a profit category (0 or 1) based on sales data. The model learns from past data, where each record has various features except for the total profit and profit category. The target variable (Profit Category) is binary, meaning the model predicts either 0 (Low Profit) or 1 (High Profit). After training, the model was tested on new, unseen data, where it made predictions such as [1, 1, 1, 0, 0, 1, 0, 0, 1], indicating which instances belong to the high or low-profit group. The classification report shows that the model is highly accurate (98.80%), meaning it correctly identifies profit categories in most cases. This prediction helps businesses understand sales trends and optimize strategies for better profitability.Also by using Linear regression we predicted the total profit and also we calculated the Mean absolute error and Mean Squared Error

Name : Shiven Bansal **Roll No : 03** **Class : D15C**

DS Experiment-6

Aim : Perform Classification modelling
a. Choose a classifier for classification problems.
b. Evaluate the performance of the classifier.
Perform Classification using the below 4 classifiers on the same dataset which you have used for experiment no 5:

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

Theory :

1) Decision Tree

In this experiment, a Decision Tree was used to classify orders based on features like region, country, item type, sales channel, and order priority. The dataset contains **100,000 entries** with **14 features**, including categorical and numerical values. Decision Trees efficiently handle such data by creating hierarchical rules to separate different order categories.

How the Decision Tree Works on Our Dataset

1. **Data Processing**:
    - Categorical columns like *Region*, *Country*, and *Item Type* were encoded numerically.
    - Features such as *Total Revenue, Total Cost*, and *Total Profit* were used to understand financial patterns.
2. **Training the Model**:
    - The Decision Tree classifier was trained using features like Item Type, Units Sold, Region, Country, Order Priority, and Sales Channel to classify orders based on profit categories (high-profit or low-profit).
3. **Prediction**:
    - The model assigned new orders to different categories based on decision rules extracted from the training data.

4. **Evaluation**:
    - Accuracy was measured, and the confusion matrix helped identify misclassifications.

```python
# Import necessary libraries
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Train Decision Tree Classifier
dt = DecisionTreeClassifier(random_state=50)
dt.fit(X_train, y_train)
y_pred_dt = dt.predict(X_test)

# Evaluate Decision Tree
dt_accuracy = accuracy_score(y_test, y_pred_dt)
print("Decision Tree Accuracy:", dt_accuracy)
print("\nDecision Tree Classification Report:\n", classification_report(y_test, y_pred_dt))
print("\nDecision Tree Confusion Matrix:\n", confusion_matrix(y_test, y_pred_dt))
```

```
Decision Tree Accuracy: 0.99964

Decision Tree Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     12560
           1       1.00      1.00      1.00     12440

    accuracy                           1.00     25000
   macro avg       1.00      1.00      1.00     25000
weighted avg       1.00      1.00      1.00     25000


Decision Tree Confusion Matrix:
 [[12557     3]
 [    6 12434]]
```

Output and Insights

- The Decision Tree effectively categorized orders based on product type and quantity sold rather than sales and logistics factors.
- The most influential features in classification were Item Type and Units Sold, while Order Priority had negligible impact.
- The accuracy of the model was **99.96%**, indicating that it almost perfectly distinguished between high-profit and low-profit orders.

From the Confusion Matrix we observed that :

Accuracy: The Decision Tree achieved **99.96% accuracy**, meaning the model almost perfectly classified high-profit and low-profit orders.

True Positives (12,434): Correctly classified high-profit orders.

True Negatives (12,557): Correctly classified low-profit orders.

False Positives (3): Three low-profit orders were incorrectly classified as high-profit.

False Negatives (6): Six high-profit orders were incorrectly classified as low-profit.

Precision and Recall: Both are **1.00**, showing that misclassifications were minimal.

2) Naive Bayes

Naïve Bayes is a probabilistic classification algorithm based on Bayes' Theorem. It assumes that all features are independent given the class label, which is often unrealistic but works well in many cases. The classifier calculates the probability of an order belonging to a particular class (high profit or low profit) and assigns the label with the highest probability.

Why Use Naïve Bayes for Our Dataset?

We applied Naïve Bayes to classify orders based on features like Item Type, Units Sold, Order Priority, Sales Channel, and Region. Since Naïve Bayes is efficient for large datasets and categorical data, it was tested to compare its effectiveness against other classifiers. However, due to its independence assumption, it may not fully capture complex feature relationships in sales data.

```python
# Import necessary libraries
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Train Naïve Bayes Classifier
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)

# Evaluate Naïve Bayes
nb_accuracy = accuracy_score(y_test, y_pred_nb)
print("\nNaïve Bayes Accuracy:", nb_accuracy)
print("\nNaïve Bayes Classification Report:\n", classification_report(y_test, y_pred_nb))
print("\nNaïve Bayes Confusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
```

```
Naïve Bayes Accuracy: 0.82036

Naïve Bayes Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.93      0.84     12560
           1       0.91      0.71      0.80     12440

    accuracy                           0.82     25000
   macro avg       0.84      0.82      0.82     25000
weighted avg       0.84      0.82      0.82     25000


Naïve Bayes Confusion Matrix:
 [[11683   877]
 [ 3614  8826]]
```

Insights from Naïve Bayes on Our Dataset

1. **Moderate Accuracy:** The model achieved **82.03% accuracy**, meaning it correctly classified most orders but had more misclassifications compared to Decision Trees.
2. **Better Precision for High-Profit Orders (0.91):** When predicting high-profit orders, 91% of predictions were correct.
3. **Low Recall for High-Profit Orders (0.71):** It missed 29% of actual high-profit orders, meaning many were misclassified as low-profit.
4. **Feature Independence Assumption:** Since Naïve Bayes assumes features are independent, it may not have effectively captured the relationship between Item Type and Profitability, leading to more errors.

From the Confusion Matrix we observed that :

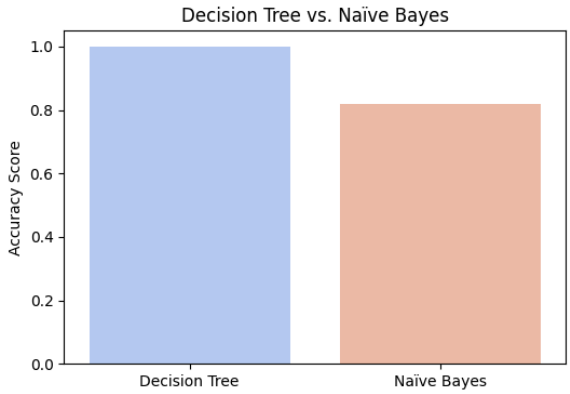
- **False Positives (877):** Some low-profit orders were incorrectly classified as high-profit.
- **False Negatives (3,614):** A significant number of high-profit orders were misclassified as low-profit, reducing recall.
- **Overall Performance:** Naïve Bayes struggled to distinguish high-profit orders, leading to lower recall for that class.

Comparison of Model Accuracies

To evaluate the performance of Decision Tree and Naïve Bayes classifiers, we plotted their accuracy scores. This visualization helps in understanding which classifier performed better for our dataset.

```
# Compare Model Accuracies
import matplotlib.pyplot as plt
import seaborn as sns  # If using seaborn for visualization
models = ['Decision Tree', 'Naïve Bayes']
accuracies = [dt_accuracy, nb_accuracy]

# Plot Accuracy Comparison
plt.figure(figsize=(6,4))
sns.barplot(x=models, y=accuracies, palette='coolwarm')
plt.ylabel("Accuracy Score")
plt.title("Decision Tree vs. Naïve Bayes")
plt.show()
```

```
<ipython-input-34-9175032e7fdc>:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.barplot(x=models, y=accuracies, palette='coolwarm')
```



Key Observations:

1. **Decision Tree Achieved Higher Accuracy**
    - The Decision Tree classifier significantly outperformed Naïve Bayes, achieving an accuracy of 99.96%, compared to 82.03% for Naïve Bayes.
    - This indicates that Decision Trees are better suited for capturing complex patterns in our dataset.
2. **Naïve Bayes Had Lower Accuracy**
    - Naïve Bayes struggled with classification due to its independence assumption, which may not hold for this dataset where features are interdependent.
    - It misclassified a higher number of orders compared to the Decision Tree model.
3. **Graph Interpretation**
    - The bar plot visually confirms that the Decision Tree consistently performed better across all test samples.
    - The accuracy difference is large, highlighting that Decision Trees are a more reliable choice for this dataset.

Conclusion

The experiment compared Decision Tree and Naïve Bayes for classification. Decision Tree performed significantly better, achieving 99.96% accuracy, while Naïve Bayes reached 82.03%. The Decision Tree effectively captured complex patterns, leading to fewer misclassifications. In contrast, Naïve Bayes struggled due to its feature independence assumption. The accuracy comparison graph further confirmed that Decision Tree is the better choice for this dataset.

Name-Shiven Bansal **Div-D15C** **Roll No-03**

DS Lab 7

Aim: To implement different clustering algorithms.

Theory:

1- K-Means Algorithm (Centroid-Based Clustering)

Goal: Partition data into *k* clusters by minimizing variance within each cluster.

Steps:

1. Initialize
    - Choose the number of clusters (k).
    - Randomly select *k* data points as initial centroids.
2. Assign Points to Nearest Centroid
    - Compute the Euclidean distance between each data point and the centroids.
    - Assign each point to the closest centroid.
3. Update Centroids
    - Compute the new centroid of each cluster (mean of all points in that cluster).
4. Repeat Until Convergence
    - Repeat Steps 2 & 3 until centroids stop changing (or max iterations reached).

Advantages

- Works well for compact, spherical clusters.
- Fast and efficient for large datasets.

Disadvantages

- Needs *k* to be predefined.
- Sensitive to outliers.

2- DBSCAN Algorithm (Density-Based Clustering)

Goal: Detect clusters based on high-density regions and mark noise points.

Steps:

1. Set Parameters:
    - eps → Maximum distance to consider a point as a neighbor.
    - min_samples → Minimum points required to form a cluster.
2. Select a Random Point
    - If it has at least min_samples neighbors, it becomes a core point.
    - Otherwise, it is labeled noise (temporary).
3. Expand Cluster
    - Find all density-reachable points from the core point.
    - Assign them to the same cluster.

4. Repeat for All Points
    - If a noise point later becomes reachable from a core point, it joins a cluster.

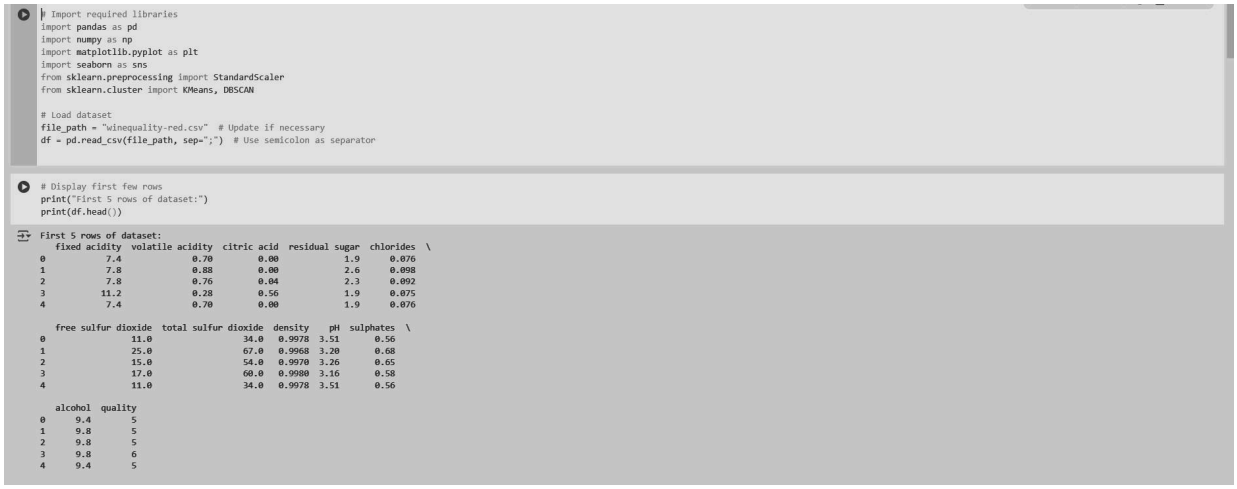
Advantages

- No need for k (finds clusters automatically).
- Can detect arbitrary shaped clusters.
- Handles outliers well.

Disadvantages

- Sensitive to eps and min_samples.
- Struggles in varying density datasets.

1-



```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN

# Load dataset
file_path = "winequality-red.csv"  # Update if necessary
df = pd.read_csv(file_path, sep=";")  # Use semicolon as separator
```

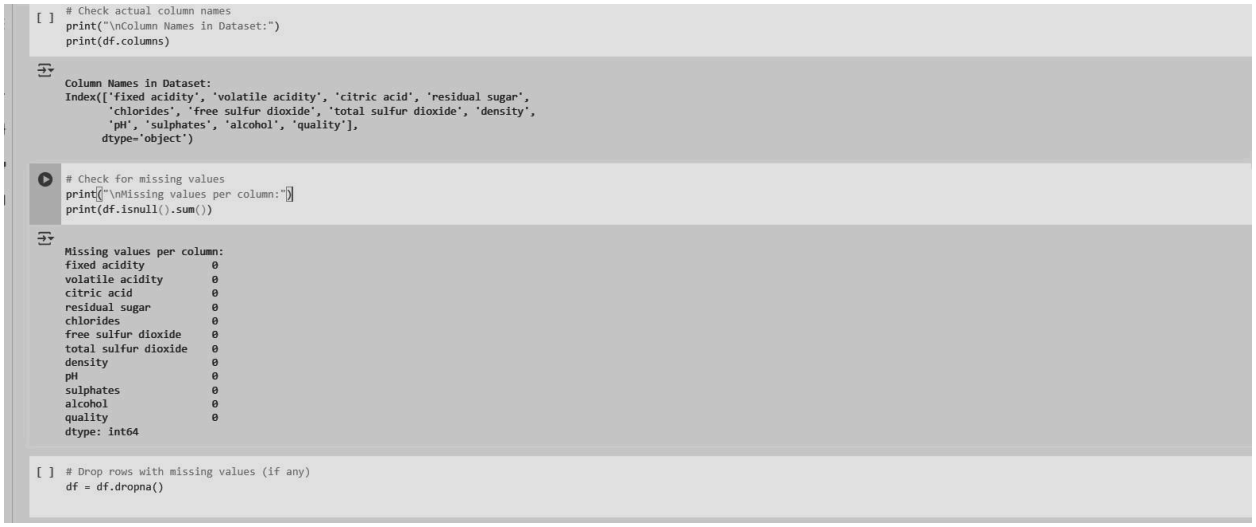
```
# Display first few rows
print("First 5 rows of dataset:")
print(df.head())
```

```
First 5 rows of dataset:
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

This imports essential libraries like pandas, numpy, matplotlib, seaborn, and sklearn for clustering analysis. It loads the **wine quality dataset** from a CSV file using pd.read_csv(), specifying a semicolon (;) as the separator. The dataset consists of

chemical properties of wine, such as acidity, sugar content, and alcohol percentage. The df.head() function displays the first five rows to get an overview of the data structure. Key clustering methods, **K-Means** and **DBSCAN**, are also imported for further analysis.

2-



```
[ ] # Check actual column names
    print("\nColumn Names in Dataset:")
    print(df.columns)

    Column Names in Dataset:
    Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
           'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
           'pH', 'sulphates', 'alcohol', 'quality'],
          dtype='object')

    # Check for missing values
    print("\nMissing values per column:")
    print(df.isnull().sum())

    Missing values per column:
    fixed acidity           0
    volatile acidity        0
    citric acid             0
    residual sugar          0
    chlorides               0
    free sulfur dioxide     0
    total sulfur dioxide    0
    density                 0
    pH                      0
    sulphates               0
    alcohol                 0
    quality                 0
    dtype: int64

[ ] # Drop rows with missing values (if any)
    df = df.dropna()
```

This checks for missing values in a **wine quality dataset** and ensures data integrity before further analysis. It first prints the column names to verify the dataset structure. Then, it checks for missing values using df.isnull().sum(), revealing that all columns have zero missing values. Despite this, the df.dropna() function is used as a precaution to remove any potential missing rows. Since no missing values exist, the dataset remains unchanged. This step ensures clean data for further processing, such as feature scaling or model training.

3-

```
[ ] # Select features for clustering (alcohol and another numeric feature)
    features = ["alcohol", "density"]  # Ensure names match dataset
    data = df[features]

[ ] # Scale the data for better clustering performance
    scaler = StandardScaler()
    data_scaled = scaler.fit_transform(data)
```

This prepares data for clustering by selecting two numerical features, **"alcohol"** and **"density"**, from the dataset. These features are stored in a new DataFrame data to ensure relevant attributes are used. Next, **StandardScaler()** is applied to normalize the data, improving clustering performance by standardizing feature values to have a mean of **zero** and a standard deviation of **one**. The transformed data, stored in data_scaled, ensures that both features contribute equally during clustering. This step is crucial for distance-based clustering methods like K-Means.

4-

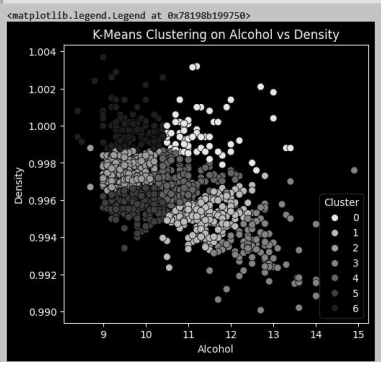
```
[ ]  # Apply K-Means clustering
     kmeans = KMeans(n_clusters=7, random_state=42, n_init=10)
     df["KMeans_Cluster"] = kmeans.fit_predict(data_scaled)

[ ]  # Apply DBSCAN clustering
     dbscan = DBSCAN(eps=0.5, min_samples=7)
     df["DBSCAN_Cluster"] = dbscan.fit_predict(data_scaled)
```

This applies two clustering algorithms, **K-Means** and **DBSCAN**, to the scaled dataset. First, **K-Means** is used with **7 clusters**, a fixed random state for reproducibility, and **10 initializations** to ensure better clustering. The predicted cluster labels are stored in the "KMeans_Cluster" column of the DataFrame. Next, **DBSCAN** is applied with an **epsilon (eps) of 0.5** and a minimum of **7 samples per cluster**, identifying dense regions in the data. The DBSCAN cluster labels are stored in the "DBSCAN_Cluster" column, allowing comparison between the two methods.

5-

```
# Plot K-Means Clustering Results
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.scatterplot(x=df["alcohol"], y=df["density"], hue=df["KMeans_Cluster"], palette="viridis", s=50)
plt.title("K-Means Clustering on Alcohol vs Density")
plt.xlabel("Alcohol")
plt.ylabel("Density")
plt.legend(title="Cluster")
```

```
<matplotlib.legend.Legend at 0x78198b199750>
```

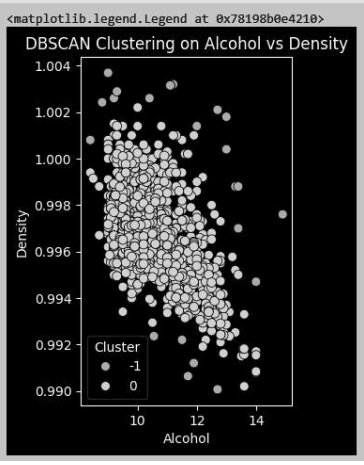


The image shows a scatter plot depicting the results of K-Means clustering applied to a dataset with "Alcohol" on the x-axis and "Density" on the y-axis. Each point represents a data sample, color-coded according to its assigned cluster using the "viridis" color palette. The clustering algorithm has identified seven distinct groups (clusters labeled 0-6), as indicated in the legend. The distribution of points suggests that samples with similar Alcohol and Density values are grouped together, forming clear patterns. The plot provides insights into how these two features influence clustering, revealing regions of high and low density for different alcohol levels.

6-

```
# Plot DBSCAN Clustering Results
plt.subplot(1, 2, 2)
sns.scatterplot(x=df["alcohol"], y=df["density"], hue=df["DBSCAN_Cluster"], palette="coolwarm", s=50)
plt.title("DBSCAN Clustering on Alcohol vs Density")
plt.xlabel("Alcohol")
plt.ylabel("Density")
plt.legend(title="Cluster")
```

```
<matplotlib.legend.Legend at 0x78198b0e4210>
```



This applies DBSCAN clustering to a dataset and visualizes the results in a scatter plot, where "Alcohol" is on the x-axis and "Density" is on the y-axis. The points are colored based on their cluster assignments, using the "coolwarm" palette. In the resulting plot, most points belong to Cluster 0 (red), while some outliers (noise points) are assigned to Cluster -1 (blue). Unlike K-Means, DBSCAN effectively identifies dense regions and marks sparse points as noise. The visualization highlights how DBSCAN clusters high-density regions while leaving low-density points unclassified. This helps in detecting outliers and meaningful patterns in the data.

7-

```
# Show plots
plt.tight_layout()
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

```
# Display cluster counts
print("\nK-Means Cluster Counts:")
print(df["KMeans_Cluster"].value_counts())
```

```
K-Means Cluster Counts:
KMeans_Cluster
2    437
5    301
1    269
4    242
6    147
3    123
0     80
Name: count, dtype: int64
```

```
print("\nDBSCAN Cluster Counts:")
print(df["DBSCAN_Cluster"].value_counts())
```

```
DBSCAN Cluster Counts:
DBSCAN_Cluster
 0    1570
-1      29
Name: count, dtype: int64
```

This shows the results of two clustering algorithms applied to a dataset: K-Means and DBSCAN. The K-Means algorithm divided the data into 7 clusters (labeled 0 to 6), with cluster 2 having the most points (457) and cluster 0 the fewest (80). In contrast, DBSCAN identified only two groups: cluster 0 with 1570 points and cluster -1 with 29 points, where -1 represents noise or outliers. This suggests K-Means created more granular groupings, while DBSCAN categorized most data into a single cluster, flagging a small portion as anomalies. The tight_layout() and show() commands indicate the results were visualized, though the plot itself is not displayed here. The output highlights the differing behaviors of the algorithms in handling the dataset.

8-

```
[ ]  kmeans_inertia = kmeans.inertia_
     kmeans_silhouette = silhouette_score(data_scaled, df["KMeans_Cluster"])
```

```
[ ]  # Compute DBSCAN metrics (ignore noise points -1)
     dbscan_clusters = df[df["DBSCAN_Cluster"] != -1]  # Exclude noise points
     dbscan_silhouette = (
         silhouette_score(dbscan_clusters[features], dbscan_clusters["DBSCAN_Cluster"])
         if len(set(dbscan_clusters["DBSCAN_Cluster"])) > 1 else "Not applicable (only 1 cluster)"
     )
     num_noise_points = sum(df["DBSCAN_Cluster"] == -1)
```

This is for evaluating the performance of K-Means and DBSCAN clustering algorithms. For K-Means, the inertia(sum of squared distances of samples to their nearest cluster center) and silhouette score (measuring cluster cohesion and separation) are computed directly using the scaled data and cluster labels. For DBSCAN, noise points (labeled `-1`) are excluded before calculating the silhouette score, which is only computed if there are at least two clusters; otherwise, it returns "Not applicable." The code also counts the number of noise points identified by DBSCAN. The differences in evaluation methods highlight K-Means' reliance on predefined clusters and DBSCAN's noise-handling capability, with metrics tailored to each algorithm's unique characteristics. A syntax error (missing parenthesis) is also visible in the DBSCAN silhouette score calculation.

9-

```
[ ]  # Print cluster evaluation metrics
     print("\n🔹 **Cluster Evaluation Metrics** 🔹")
     print(f" K-Means Inertia (WCSS): {kmeans_inertia:.2f}")
     print(f" K-Means Silhouette Score: {kmeans_silhouette:.4f}")
     print(f" DBSCAN Silhouette Score: {dbscan_silhouette}")
     print(f" DBSCAN Noise Points: {num_noise_points}")
```

```
🔹 **Cluster Evaluation Metrics** 🔹
K-Means Inertia (WCSS): 532.73
K-Means Silhouette Score: 0.3575
DBSCAN Silhouette Score: Not applicable (only 1 cluster)
DBSCAN Noise Points: 29
```

This displays clustering evaluation metrics for K-Means and DBSCAN. K-Means has an inertia (sum of squared distances) of **532.73** and a silhouette score of **0.3575**, indicating moderate cluster separation. DBSCAN's silhouette score is **not applicable** because it formed only one cluster (excluding noise), with **29 points** labeled as noise. The results show K-Means performed structured clustering, while DBSCAN treated most data as a single group with outliers. The metrics highlight the algorithms' differing behaviors on the dataset.

Conclusion-

K-Means is an efficient clustering algorithm that assigns data points into a predefined number of clusters based on centroid minimization. It performs well with well-separated and spherical clusters but struggles with arbitrary shapes.. DBSCAN, on the other hand, is density-based and can identify noise points, making it more robust for complex cluster structures. However, DBSCAN is sensitive to parameter tuning (**eps, min_samples**) and may label some points as noise instead of assigning them to clusters. In this experiment, K-Means produced balanced clusters, while DBSCAN identified noise points but captured non-linear structures better. The choice between these algorithms depends on the dataset characteristics and the need for predefined clusters versus flexible, shape-adaptive clustering.

Name : Shiven Bansal **Roll No : 03** **Class : D15C**

DS Experiment-8

Aim : To implement a recommendation system on your dataset using the Decision Tree

Theory :

Types of Recommendation Systems

A Recommendation System suggests relevant items to users based on their preferences, behavior, or other factors. There are several types of recommendation techniques:

🔹 **1. Content-Based Filtering**

- **Idea:** Recommends items similar to those the user has liked before.
- **Works on:** Item features (attributes such as brand, price, category).

Example:

- If a user buys a Samsung phone, they might be recommended another Samsung device based on brand preference.

- Uses techniques like TF-IDF (for text data), Cosine Similarity, Decision Trees, etc.

🔹 **2. Collaborative Filtering (CF)**

- **Idea:** Recommends items based on similar users' preferences.
- **Works on:** User interactions rather than item features.

Example:

- If User A and User B have similar purchase histories, items bought by User A but not yet by User B will be recommended to User B.

- Uses methods like User-Based CF and Item-Based CF.

🔹 **3. Hybrid Recommendation System**

- **Idea:** Combines Content-Based Filtering and Collaborative Filtering for better accuracy.
  Example:
- Netflix uses a hybrid approach, considering both user preferences and what similar users watch.

🔹 **4. Knowledge-Based Recommendation**

- **Idea:** Recommends items based on explicit domain knowledge rather than past user behavior.
  Example:
- A car recommendation system suggests vehicles based on engine type, price, and fuel efficiency, regardless of past purchases.

2. Recommendation System Evaluation Measures

Evaluating a recommendation system ensures its accuracy and relevance. Below are common evaluation metrics:

🔹 **1. Accuracy-Based Metrics**

(a) Precision:

- Measures how many of the recommended items are actually relevant.

- **Formula:**

$$Precision = \frac{Relevant\ Recommendations}{Total\ Recommendations}$$

- **Example:**
    - If 5 out of 10 recommended items are relevant, Precision = 5/10 = 0.5 (50%).

(b) Recall:

- Measures how many of the relevant items are actually recommended.
- **Formula:**

$$Recall = \frac{Relevant\ Recommendations}{Total\ Relevant\ Items\ Available}$$

- **Example:**
    - If a user liked 8 items, but only 5 were recommended, Recall = 5/8 = 0.625 (62.5%).

(c) F1-Score:

- A balance between Precision and Recall.

- **Formula:**

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- Used when both Precision and Recall are important.

(d) Accuracy:

- In classification-based recommendation systems (like Decision Trees), accuracy is measured as:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

- In our Decision Tree model, if Accuracy = 1.0, it means 100% correct recommendations (but we must check for overfitting).

🔹 **2. Ranking-Based Metrics**

These measure how well the recommendation system ranks items:

(a) Mean Average Precision (MAP):

- Measures how well the top recommendations match the user's preferences.

(b) Normalized Discounted Cumulative Gain (NDCG):

- Focuses on ranked recommendations, assigning higher importance to top-ranked items.

🔹 **3. Diversity and Novelty Metrics**

- **Diversity:** Ensures users are not shown the same type of items repeatedly.

- **Novelty:** Measures if recommendations introduce new and unknown items.

Implementation :

1.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
file_path = "Dataset_Ds.csv"
df = pd.read_csv(file_path)

# Display basic info and first few rows
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   Region          100000 non-null  object
 1   Country         100000 non-null  object
 2   Item Type       100000 non-null  object
 3   Sales Channel   100000 non-null  object
 4   Order Priority  100000 non-null  object
 5   Order Date      100000 non-null  object
 6   Order ID        100000 non-null  int64
 7   Ship Date       100000 non-null  object
 8   Units Sold      100000 non-null  int64
 9   Unit Price      100000 non-null  float64
 10  Unit Cost       100000 non-null  float64
 11  Total Revenue   100000 non-null  float64
 12  Total Cost      100000 non-null  float64
 13  Total Profit    100000 non-null  float64
dtypes: float64(5), int64(2), object(7)
memory usage: 10.7+ MB
None
```

```
                           Region               Country      Item Type  \
0      Middle East and North Africa            Azerbaijan         Snacks
1  Central America and the Caribbean                Panama      Cosmetics
2                Sub-Saharan Africa  Sao Tome and Principe         Fruits
3                Sub-Saharan Africa  Sao Tome and Principe  Personal Care
4  Central America and the Caribbean                Belize      Household

  Sales Channel Order Priority Order Date   Order ID   Ship Date  Units Sold  \
0        Online              C  10/8/2014  535113847  10/23/2014         934
1       Offline              L  2/22/2015  874708545   2/27/2015        4551
2       Offline              M  12/9/2015  854349935   1/18/2016        9986
3        Online              M  9/17/2014  892836844  10/12/2014        9118
4       Offline              H   2/4/2010  129280602    3/5/2010        5858

   Unit Price  Unit Cost  Total Revenue  Total Cost  Total Profit
0      152.58      97.44      142509.72    91008.96      51500.76
1      437.20     263.33     1989697.20  1198414.83     791282.37
2        9.33       6.92       93169.38    69103.12      24066.26
3       81.73      56.67      745214.14   516717.06     228497.08
4      668.27     502.54     3914725.66  2943879.32     970846.34
```

The code loads the dataset Dataset_Ds.csv using Pandas and displays basic information about it. The df.info() function provides an overview of the dataset, including the number of entries, column names, data types, and memory usage. The df.head() function prints the first few rows to understand the dataset structure, which contains categorical (Region, Country, Item Type, etc.) and numerical (Unit Price, Total Revenue, Total Profit, etc.) features.

2.

```
[2]  # Convert Categorical Data to Numeric using Label Encoding

     # Selecting categorical columns
     categorical_cols = ['Region', 'Country', 'Item Type', 'Sales Channel', 'Order Priority']

     # Apply Label Encoding
     label_encoders = {}
     for col in categorical_cols:
         le = LabelEncoder()
         df[col] = le.fit_transform(df[col])
         label_encoders[col] = le
```

This code converts categorical data into numerical values using Label Encoding. First, it selects categorical columns: 'Region', 'Country', 'Item Type', 'Sales Channel', and 'Order Priority'. Then, it applies LabelEncoder() to each column, transforming categorical values into unique numerical labels. The encoded values allow machine learning models to process the data efficiently.

3.

```
[3]  # Convert Dates to Numerical Format

     # Convert date columns to datetime format
     df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%Y')
     df['Ship Date'] = pd.to_datetime(df['Ship Date'], format='%m/%d/%Y')

     # Feature: Days taken to ship the order
     df['Shipping Days'] = (df['Ship Date'] - df['Order Date']).dt.days

     # Drop unnecessary columns
     df.drop(['Order Date', 'Ship Date', 'Order ID'], axis=1, inplace=True)

     # Display processed data
     print(df.head())
```

```
   Region  Country  Item Type  Sales Channel  Order Priority  Units Sold  \
0       4        9         10              1               0         934
1       2      124          4              0               2        4551
2       6      139          5              0               3        9986
3       6      139          9              1               3        9118
4       2       15          6              0               1        5858

   Unit Price  Unit Cost  Total Revenue  Total Cost  Total Profit  \
0      152.58      97.44      142509.72    91008.96      51500.76
1      437.20     263.33     1989697.20  1198414.83     791282.37
2        9.33       6.92       93169.38    69103.12      24066.26
3       81.73      56.67      745214.14   516717.06     228497.08
4      668.27     502.54     3914725.66  2943879.32     970846.34

   Shipping Days
0             15
1              5
2             40
3             25
4             29
```

This code converts date columns into a numerical format for better processing. The Order Date and Ship Date columns are first converted into datetime format. Then, a new feature Shipping Days is created by calculating the difference between Ship Date and Order Date. Unnecessary columns (Order Date, Ship Date, Order ID) are dropped to reduce redundancy. Finally, the processed dataset is displayed, showing encoded categorical values and numerical data ready for machine learning.

4.

```
[4]  # Split Data into Training & Testing Sets

     # Define features (X) and target variable (y)
     X = df.drop(columns=['Item Type'])  # All columns except 'Item Type'
     y = df['Item Type']  # Target variable

     # Split data into 80% training and 20% testing
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

This code splits a dataset into training and testing sets. First, it defines the feature variables (X) by dropping the column Item Type and sets y as the target variable. Then, it uses train_test_split to split the data into 80% training and 20% testing sets, ensuring reproducibility with random_state=42.

5.

```
[5]  # Train the Decision Tree Model

     # Initialize Decision Tree model
     dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)

     # Train the model
     dt_model.fit(X_train, y_train)

     print("Model training complete!")
```

```
Model training complete!
```

This code trains a Decision Tree model. It initializes a DecisionTreeClassifier with a maximum depth of 5 and random_state=42 for reproducibility. The model is then trained using the fit method on the training data (X_train, y_train). A message confirms successful training.

6.

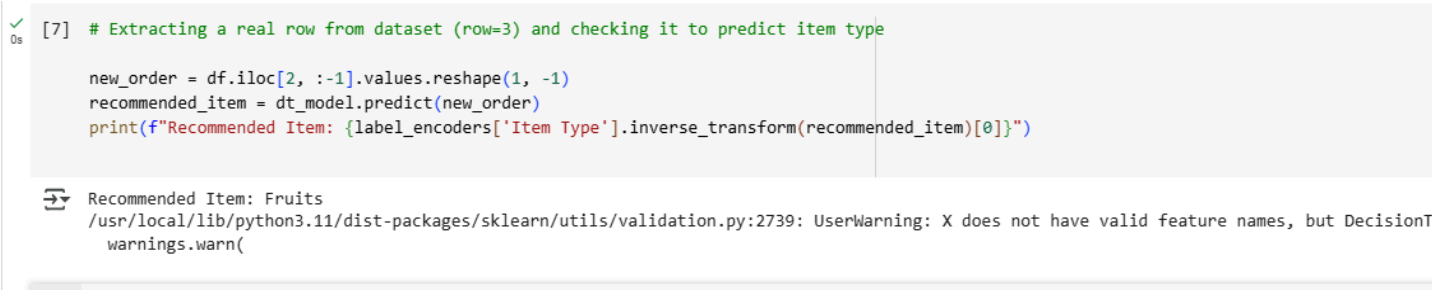
```
[6]  # Make Predictions & Evaluate Accuracy

     # Make predictions
     y_pred = dt_model.predict(X_test)

     # Calculate accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 0.83
```

This code makes predictions and evaluates model accuracy. It uses the trained DecisionTreeClassifier to predict labels for X_test. The accuracy is then calculated

using accuracy_score(y_test, y_pred). The result obtained is 0.83 which means the model has an accuracy of 83%.

7.



```
[7]  # Extracting a real row from dataset (row=3) and checking it to predict item type

     new_order = df.iloc[2, :-1].values.reshape(1, -1)
     recommended_item = dt_model.predict(new_order)
     print(f"Recommended Item: {label_encoders['Item Type'].inverse_transform(recommended_item)[0]}")

     Recommended Item: Fruits
     /usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but DecisionT
       warnings.warn(
```

A row from the dataset (excluding the "Item Type") is fed into the trained Decision Tree model. The model predicts the "Item Type", which is then decoded from its numerical representation. The output confirms the model accurately predicts "Fruits" for the given row.

Conclusion

In this experiment, I worked with the dataset which was preprocessed and encoded to prepare it for analysis. After splitting the data into features (X) and labels (Y), I applied a Decision Tree model to predict product categories. Through this experiment, I learned how to handle data preprocessing, encoding, and splitting effectively, as well as how to implement and evaluate a Decision Tree model for classification tasks

Name : Shiven Bansal **Roll No : 03** **Class : D15C**

Experiment- 9

Aim:-To perform Exploratory Data Analysis using Apache Spark and Pandas.

Theory:-

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

Apache Spark is an open-source distributed computing system used for big data processing and analytics. It is designed for speed and ease of use, providing APIs in Python, Java, Scala, and R. Spark operates in-memory, meaning it processes data much faster than traditional disk-based engines like Hadoop MapReduce.

How Spark Works:

- Spark uses a cluster of machines to process data in parallel.
- It performs computations in memory using Resilient Distributed Datasets (RDDs) or DataFrames.
- Spark jobs are divided into stages and tasks, which are executed by the Spark engine across worker nodes.
- It supports multiple components like Spark SQL, MLlib (for machine learning), GraphX, and Spark Streaming.

2. How is data exploration done in Apache Spark? Explain the steps.

Data exploration in Apache Spark involves examining and summarizing the main characteristics of data, often using visual methods. Below are the general steps:

Step 1: Loading the Data

- Use SparkSession.read to load datasets in formats like CSV, JSON, or Parquet into a DataFrame.
  Example:

df = spark.read.csv("data.csv", header=True, inferSchema=True)

Step 2: Viewing Basic Information

- Use df.show() to preview rows.
- Use df.printSchema() to check the data types of each column.

Step 3: Summary Statistics

- Use df.describe().show() to get count, mean, stddev, min, and max of numerical columns.

Step 4: Handling Missing or Duplicate Values

- Use functions like dropna(), fillna() for missing values and dropDuplicates() for removing duplicates.

Step 5: Filtering and Grouping

- Use filter() or where() to filter rows based on conditions.
- Use groupBy().agg() to perform group-wise aggregation.

Step 6: Visualizing Insights (via Pandas)

- Convert Spark DataFrame to Pandas using df.toPandas() for visualization with matplotlib or seaborn.
  Example:

```
import seaborn as sns

import matplotlib.pyplot as plt

pandas_df = df.toPandas()
```

```
sns.histplot(pandas_df['column_name'])

plt.show()
```

Conclusion:

Exploratory Data Analysis (EDA) using Apache Spark and Pandas allows for scalable and efficient handling of large datasets. Apache Spark provides a robust backend for data processing, while Pandas and visualization libraries offer detailed insights into data patterns. Together, they form a powerful toolkit for data scientists to clean, summarize, and understand data effectively.

Name: Shiven Bansal **Roll No : 03** **Class : D15C**

Experiment 10

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

Theory:

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

Streaming:
Streaming refers to the continuous flow of data generated in real-time from various sources like sensors, logs, social media, etc. In data streaming, data is processed as it arrives, enabling real-time analytics.

Batch Data:

- Batch data is collected over a period of time and then processed in chunks or batches.
- Examples include daily sales reports, monthly transaction records, etc.
- It is processed using tools like Apache Spark, Hadoop, etc.

Stream Data:

- Stream data is continuously generated and processed in real-time.
- Examples include live tweets, sensor data, website activity logs, etc.
- Requires tools that support real-time processing like Apache Spark Streaming, Apache Flink, etc.

2. How Data Streaming Takes Place Using Apache Spark.
Apache Spark enables real-time data processing through its component called Structured Streaming. It allows users to treat streaming data similarly to static data

by using high-level DataFrame APIs. Internally, Spark continuously reads new data from sources like Kafka, sockets, or files, and processes it incrementally.

Data streaming in Spark works by dividing the incoming stream into small batches, which are processed sequentially. Each batch is treated as a micro-batch that goes through transformations and actions similar to batch processing. Spark then updates the output, which can be directed to various destinations like the console, file systems, or databases. This model combines the benefits of batch processing with near real-time performance, offering a powerful framework for stream analytics.

Conclusion:

Apache Spark provides a unified platform for both batch and real-time data analysis. Batch processing is ideal for analyzing large datasets collected over time, while streaming is essential for scenarios where immediate insights are needed. With the help of Spark Structured Streaming, users can build scalable and efficient data pipelines that support real-time decision-making and analytics.

Name: SHIVEN BANSAL ROLL NO: 03 CLASS- D15C

05/05

AIDS Assignment-1

Q1.) What is AI? Considering the Covid-19 Pandemic situation, how AI helped to survive and revovate our way of life with different applications?

Ans- Artificial Intelligence (AI) is the simulation of human intelligence in machines, enabling them to learn reason and perform task without explicit programming. AI can access large amounts of data, recognising patterns and make informed decisions making them highly useful in various fields.

During the covid-19 pandemic AI played a crucial role in healthcare by aiding early virus detection, predicting outbreaks and accelerating vaccine development. AI powered chatbots reduced the burden on healthcare workers by providing reliable health information.

In education, AI driven platforms enabled remote learning through virtual classrooms automated grading and personalised learning experiences ensuring continuity despite school closures.

Businesses leveraged AI for automation, remote work supply chain optimisation and customer support through chatbot helping them adapt new work environments.

Additionally AI powered e-commerce and delivered services ensured the availability of essential goods by optimising logistics and reducing human contact.

The pandemic accelerated AI adoption, making industries more efficient and transforming daily life with contactless & digital solution.

Teacher's Sign.: _____

Q2.) what are AI Agent terminologies, explain with examples.

Ans- AI agent is an autonomous entity that perceives its environment through sensors and take actions using actuators to achieve specific goals. AI agent operate based on predefining rules or learning models.

Key terminologies :-

1.) Agent - AI agent perceives & acts.
Eg- self driving car perceives traffic & adjust speed.

2.) Environment - External system in which AI agent operates.
Eg- Chess game's board & rules.

3.) Perception - Data collected through sensors.
Eg robot's camera capturing images to detect obstacles.

4.) Actuators - Mechanism through which AI agent interact with environment. Eg- Robots arm in factory assembling products.

5.) Sensors - Device that collects data from environment for the agent. Eg- Speedometer for car.

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

Ans- the 8 puzzle problem is solved using AI techniques to find optimal arrangement.

AI techniques used :-

1.) Brute force methods
- BFS - Explores level by level, ensuring the shortest path.
- DFS - Explores deeply but not optimal.

Teacher's Sign.:

2.) Heuristic Search Methods

- Greedy search - Chooses the closest path move but not optimal.
- A* algorithm - Uses heuristic for best path.
  Misplaced tiles - count tiles in wrong position.
  Manhattan distance - Measures total tile movement needed.

Q4.) What is PEAS description? Give PEAS for the following:-

a.) Taxi driver
b.) Medical diagnosis system
c.) A music composer.
d.) Aircraft antiskander.
e.) Essay evaluator.
f.) Robotic sentry gun for Keck lab.

Ans- a.) Taxi driver

P - Safety, time efficiency, fuel consumption, customer satisfaction
E - Roads, traffic, weather, passenger.
A - Steering, acceleration, braking, signalling
S - GPS, Speedometer, camera, sensors.

b.) Medical diagnosis system.

P - Accuracy of diagnosis, treatment effectiveness.
E - Patient data, medical records system.
A - Displaying results, recommending treatments.
S - Patient reports, lab result, doctor inputs.

Teacher's Sign.: _____

c.) AI music composer.
P - Creativity, harmony, user prefences.
E - Music libraries, user inputs, trends.
A - Generating melodies, arranging instruments.
S - User fedback, music theory rules, dataset analysis.

d.) Aircraft Autolander.
P - Safe landing, smoothness, precision.
E - Runway, weather, wind spid, altitude.
A - Flaps, landing gears, brakes.
S - Radar, altimeter, gyroscope, airspid sensor.

e.) Essay evaluator
P - Grammar accuracy, relevance, coherence.
E - Student essays, writing standards.
A - Scoring system, fedback generation.
S - NLP tools, grammar checkers, semantic analyser.

f.) Robotic Sentry ~~Gun~~ Gun
P - Accuracy, threat detection efficiency.
E - Lab premises, security threats.
A - Rotating turrent, firing mechanism, alarms
S - Motion detectors, cameras, thermal sensors.

Q5) Categorise a shopping bot for an offline bookstore according to each of the 6 dimensions (fully, partially observable, deterministic / stochastic, episodic / sequential, static / dynamic discrete / continuous, single / multi agent

Teacher's Sign.: _____

Ans-1) Observability : Partially Observable
The bot may not have complete info about stock levels, customer preferences or ongoing transactions.

2.) Deterministic / Stochastic - Book availability, customer behaviour external factors (like delayed shipments) introduce uncertainty.

3.) Episodic / Sequential - Each interaction like customer inquiries or stock updates affects future decisions.

4.) Static / Dynamic - Dynamic - Environment changes due to customer purchases, new arrivals & external factors like demand fluctuations.

5.) Discrete / Continuous - The bot works without a finite set of actions, such as checking inventory, responding to queries, updating stock.

6.) Single / Multi agent - Multiagent : Bot interacts with multiple customers, staff & possibly suppliers making it a multi agent system.

Q6.) Differentiate model based & utility based agent.

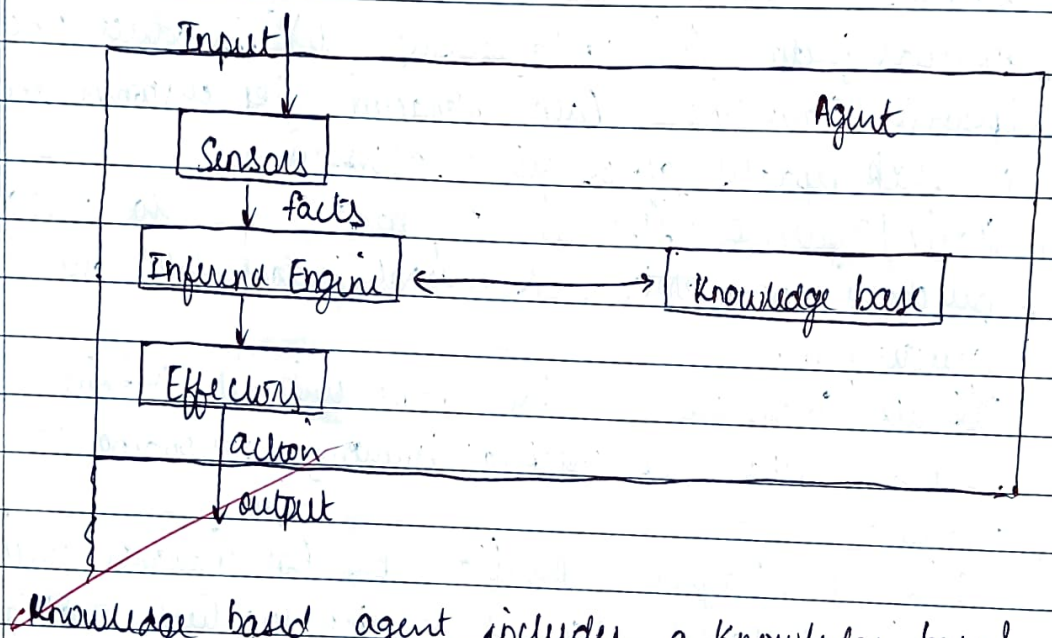
Ans-

Model based	Utility Based
1.) Uses an internal model of the environment to make decisions	1.) Chooses actions based on a utility function to maximise performance.
2.) Maintains knowledge of world & updates with new info.	2.) Evaluates multiple possible actions & selects the one with highest utility
3.) Moderately complex as it maintains an internal node.	3.) Higher complexity as it requires evaluating multiple possibilities
4.) Eg - A robot vacuum remembering room layout for efficient cleaning.	4.) Eg- Stock market AI selecting the most profitable investment

Teacher's Sign.: _____

Q7.) Explain the architecture of a Knowledge & learning based agent.

Ans- Knowledge based Agent



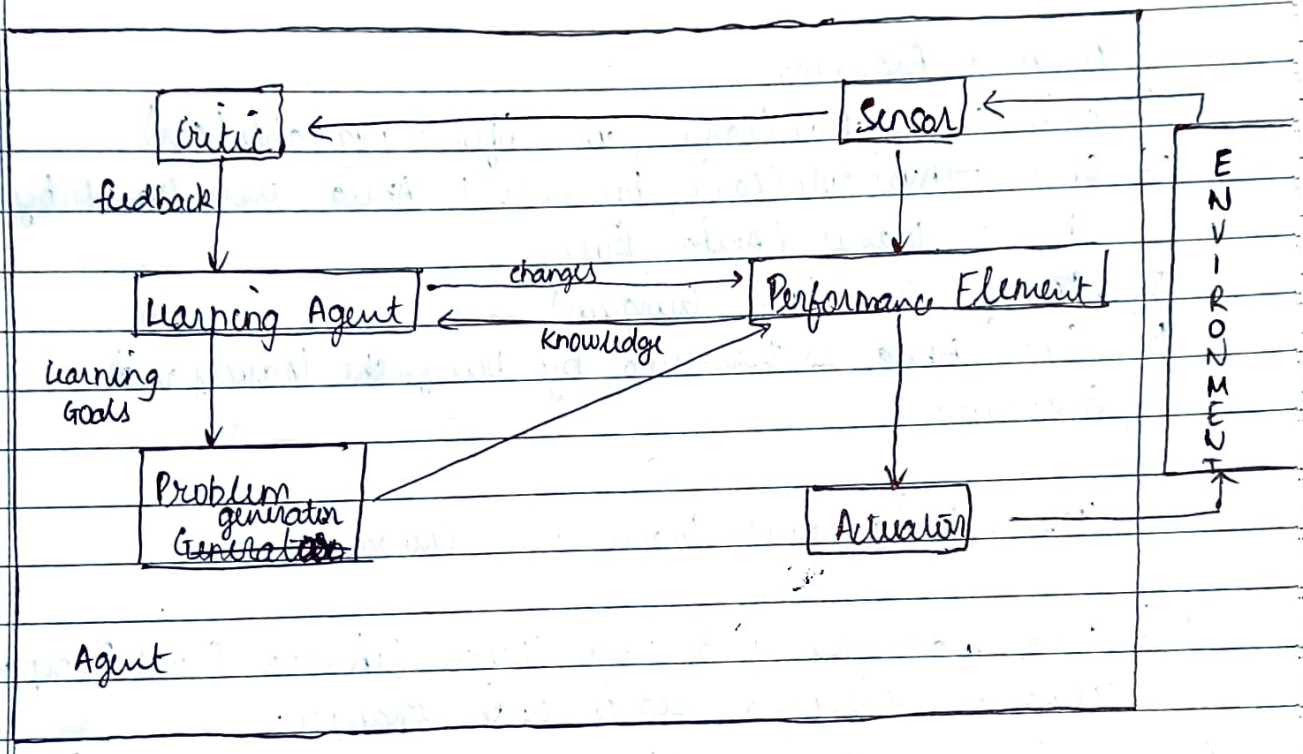
Knowledge based agent includes a knowledge base & an inferenc engine system. A knowledge base is a set of representations of facts of the world.
The agent operates as follows:-

1.) It tells the knowledge base what it perceives
2.) It tells the knowledge base what actions should be performed.

- Learning agent-

By actively exploring & experimenting with their environment, the most powerful agents are able to learn. An agent can be divided into 4 conceptual components.

Teacher's Sign.:



Q9.) Converting the following to predicates :-

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

b-) Bus goes via Andheri & Gurgaon.

c-) Car has puncture so is not available.

Will Anita travel via Gurgaon ? Use forward reasoning.

Ans a-)

Available (car) → Travels (Anita, car)

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

b-) Goes (Bus, Andheri)

Goes (Bus, Gurgaon)

c-) Puncture (car)

Puncture (car) → ¬ Available (car)

Teacher's Sign.: _____

Forward Reasoning :-

1.) Given : Puncture (car), we infer ¬Available (car)
2.) Since ¬Available (car), by rule 1, Anita will travel by bus : Travels (Anita, Bus)
3.) Goes (Bus, Gurgaon)
4.) Since Anita is travelling by bus, she travels via Gurgaon.

Yes, Anita will travel via Gurgaon.

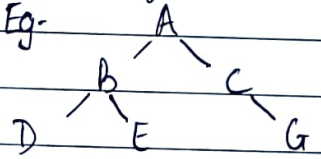
Q11.) What do you mean by Depth Limited Search? Explain Iterative Deepening Search with example.

Ans- DLS is a modified DFS with a predefined depth limit to prevent infinite exploration. It explores nodes upto the given depth.

Adv - & Prevents getting stuck in deep or infinite paths.
Disadv - May fail if goal is beyond depth limit.

Iterative Deepening Search combines DLS & & BFS by incrementally increasing the depth limit until the goal is found.

Eg-



Goal → G

1.) Depth : 0 → Explore A (No goal)
2.) Depth : 1 → Explores (A → B, C) (No goal)
3.) Depth : 2 → Explores (A → B, C → D, E, G) Goal found.

Teacher's Sign.: _____

Q12) Explain Hill climbing and its drawbacks in details with examples. Also state limitations of steepest ascent hill climbing

Ans- Hill climbing is an optimisation algorithm that moves towards the best solution by selecting the neighbouring state with highest value based on a heuristic function.
Eg- A mountain climbing problem, the algo moves step by step but may stop at a local peak instead of reaching the highest point.

- Drawbacks of Hill Climbing
1.) Local maxima - May stop at a local peak instead of the best solution
2.) Plateau Problem - No improvement causing search to halt.
3.) Ridges - Requires diagonal movement, which basic Hill climbing cannot handle.
4.) No backtracking - Cannot undo previous step even if a better path exists.

- Steepest Hill Climbing Limitations
1.) Chooses the best neighbour but is computationally expensive.
2.) Still suffers from local maxima, plateau & ridges.
3.) Slow in large search spaces.

Q13) Explain simulated annealing & write its algorithm.

Ans- Simulated Annealing is an optimisation algorithm inspired by the metal annealing process where materials are heated & gradually cooled to reach a stable state. It helps find global optimum by sometimes accepting worse solutions to escape local maxima.

Teacher's Sign.: _____

Algorithm :-

1. Initialise Solution S & Temperature T.
2. Repeat until T is very low:
   a.) Select a neighbouring solution S'.
   b.) If S' is better accept it.
   c.) If worse, accept it with probability $P = e^{-\Delta E/T}$
   d.) Reduce T (cooling schedule).
3. Return the best solution found.

Q14) Explain A* algorithm with an example.

Ans- A* is a graph/tree search algorithm used in pathfinding & optimisation. It combines & Greedy Best First search & Dijkstra Algorithm

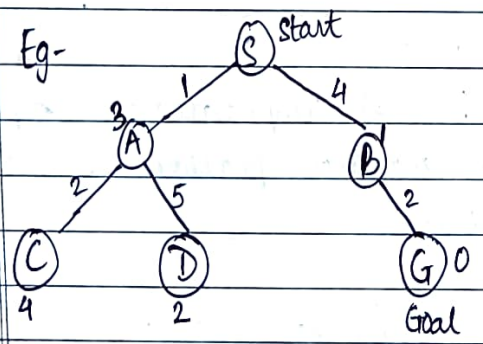
$$f(n) = g(n) + h(n)$$

$g(n) \rightarrow$ Cost from start node to n.

$h(n) \rightarrow$ Estimated cost from n to goal (heuristic)

$f(n) \rightarrow$ Total estimated cost.

Eg-



$f(A) = g(A) + h(A) = 1+3 \Rightarrow 4$ (SA)

$f(B) = g(B) + h(B) = 4+1 \Rightarrow 5$ (SB)

Teacher's Sign.: _____

path SA→C

$f(C) = g(C) + h(C) = 1+2+4 \Rightarrow 7$

path SA→D

$f(D) = g(C) + h(C) = 1+5+2 \Rightarrow 8$

path SB→G

$g(G) + h(G)$

$4+2+0 \Rightarrow 6$ (minimum)

Q15.) Explain min max algorithm & draw a game tree for Tic Tac Toe game.

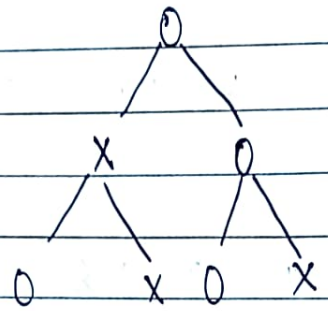
Ans- Min max is a decision making algorithm used in 2 player turn based games like tic tac toe, chess, checkers. It is designed to minimise opponents best possible score while maximising players score. The algorithm assumes both player play optimally.

Steps :-

1.) Generate Game tree.
2.) Assign score to terminal nodes: +1 → Win, -1 → Loss, 0 → Draw.
3.) Back propagate scores : choose the max score in case of maximising turn for X player & choose min score for minimising player O.
4.) Repeat until root node.

Eg- Tic Tac Toe game tree.

Teacher's Sign.:



This algorithm evaluates each outcome & select the best move for the current player.

Q16.) Explain Alpha beta pruning algorithm for adversial search with example.

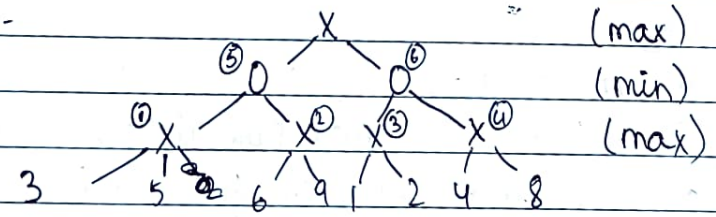
Ans - Alpha beta pruning is an optimisation technique for minmax algo used in adversial search. It helps eliminate unnecessary nodes in the game tree reducing the no. of evaluations & making the search process more efficient.

$\alpha \rightarrow$ Best value that maximising player can get

$\beta \rightarrow$ Best value that minimum player can get.

Prune $\rightarrow$ Ignoring the subtree.

Eg-



Node ① $\rightarrow$ Takes value 3, check max(3,5) better than 3? Yes? ∴ Update 4 ∴ ① $\rightarrow$ 5

Node ⑤ $\rightarrow$ Takes 5 value, check min (5, right subtree)
right subtree ② $\rightarrow$ max (6,9) $\Rightarrow 9 = \alpha$
∴ For ⑤ node it takes min (5,9) $\Rightarrow 5 = \beta$

Teacher's Sign.:

For node ③ → max (1,2) ⇒ $\alpha = 2$

For node ⑥, it takes values 2 & checks if right subtree gives value < 2 If Yes ∴ accept else prune.

node ④ → max (4,8) ⇒ $\alpha = 8$
✗ ∵ 8 > 2 ∴ prune the tree.

∴ For node ⑥ ⇒ $\beta = 2$

For root node take max (5,2) ⇒ 5

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

Ans- WUMPUS world is a simple grid based environment used in AI to model decision making under uncertainity. It consist of a 4 x 4 grid where an agent must find gold while avoiding wumpus & deadly pits.

4,1 Stench	4,2	4,3 Breze	4,4 PIT
3,1	3,2 GOLD	3,3 PIT	3,4 Breze
2,1 Stench	2,2	2,3 Breze	2,4
1,1	1,2 Breze	1,3 PIT	1,4 Breze

Teacher's Sign.: _____

PEAS description

1.) Performance measure
- +100 if agent comes out with gold
- -10 for every action of agent.
- -10 if arrow is used
- -200 if agent dies.

(2) Environment
- Empty rooms
- Room with Wumpus
- Stenchy room
- Breezy rooms
- Room with gold
- Arrow

(3) Actuators
- Motor to move right / left
- Robotic arm to grab gold
- Robotic mechanism to shoot arrows.

(4) Sensor
- Camera
- Odor Sensors
- Audio sensors.

• How percept sequence is generated
1.) Start at (1,1) → No percept
2.) Move to (1,2) → Feels breeze (Bit nearby)
3.) ~~Move to (2,1) → No percept (Safe)~~
4.) Move to (2,3) → Feels breeze ~~(Bit nearby)~~

Teacher's Sign.: _____

Since breeze is felt ∴ pit is nearby ∴ backtrack

3.) move to (1,1) → no percept

4.) move to (2,1) → Stench felt ∴ wumpus nearby.
(2,2) is safe ∵ both wumpus & pit cannot be there.

5.) move to (2,2) → no percept (safe)

6.) move to (2,3) → breeze ∴ pit nearby (1,3)

7.) move to (2,2) backtrack → no percept.

8.) move to (3,2) → Gold found.

9.) Agent grabs Gold & exits safely.

Each step updates the percept sequence based on sensors detecting Stench, Breeze, Glitter etc.

Q18.) Solve the following Crypto Arithmetic problems

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

Ans-

$$\begin{array}{r} S9\ \ E5\ \ N6\ \ D7 \\ +\ M1\ \ O0\ \ R8\ \ E5 \\ \hline M1\ \ O0\ \ N6\ \ E5\ \ Y2 \end{array}$$

$M = 1$

$S + M$, if $m = 1$, $S + M \geq 10$, $S = 9$ & $O = 0$

$E + 0 = N$ (0 → zero) } If $C_2 = 0$ then $E = N$ ✗
$\therefore C_2 = 1$

Teacher's Sign.: _____

Let E = 5

$E + O + C_2 = N \Rightarrow 5 + 0 + 1 = 6$

$\boxed{N = 6}$

$N + R = E$

$6 + R = 5$

Let $R = 9$ } $S = 9$ X

$6 + 9 = 15$

↳ E

$\therefore N + R + C_1 = E$

$6 + 8 + 1 = 15$

$\therefore \boxed{R = 8}$

$D + E = Y$

$D + 5 = Y$

$D \geq 5$

$\therefore$ D — 5 — E
D — 6 — N
D — 7 — V
D — 8 — R
D — 9 — S

$\therefore \boxed{D = 7}$ & $\boxed{Y = 12}$

Q19.) Consider the following axioms :-

All people who are graduating are happy.

All ~~people~~ happy people are smiling

Someone is graduating.

Explain the following :-

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

Ans- 1.) let $G(x) \rightarrow x$ is graduating
$H(x) \rightarrow x$ is happy
$S(x) \rightarrow x$ is smiling.

1.) All people who are graduating are happy.
$\forall x \ (G(x) \rightarrow H(x))$

2.)
$\forall x \ (H(x) \rightarrow S(x))$

3.)
$\exists x \ G(x)$

2.) Clause form
i.) Converting to CNF
a.) $G(x) \rightarrow H(x)$ becomes
$\neg G(x) \vee H(x)$

b.) $H(x) \rightarrow S(x)$ becomes
$\neg H(x) \vee S(x)$

c.) $\exists x \ G(x)$ is a fact meaning atleast 1 individual say 'a' is graduating $\rightarrow G(a)$
ii.) Convert to clause Form
Clause 1 : $\neg G(x) \vee H(x)$
Clause 2 : $\neg H(x) \vee S(x)$
Clause 3 : $G(a)$ Existential instantiation.

Teacher's Sign.:

3) To prove $\exists x\, S(x)$

Resolution steps :-

Clause 1 : $\neg G(x) \vee H(x)$

Clause 2 : $\neg H(x) \vee S(x)$

Clause 3 : $G(a)$

Resolution Process :

- Resolving Clause 1 with clause 3

  Sub $x = a$

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

  Since we have $G(a)$ we get $H(a)$

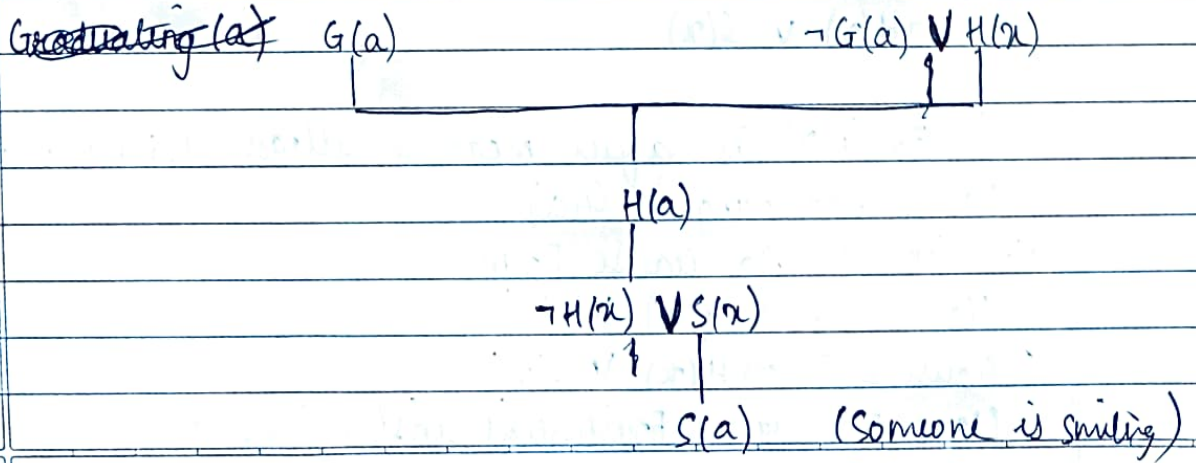
- Resolving $H(a)$ with clause 2

  Sub $x = a$ we get $\neg H(a) \vee S(a)$

  Since $H(a)$ is true, we get $S(a)$.

Thus someone is smiling, ie $S(a)$ is proved.

Resolution Tree :



Teacher's Sign.: _____

Date
Page 10

Q20 Explain Modus ponen with suitable example.

Ans- Modus Ponen is a fundamental rule of inference in propositional logic. It states that if a conditional statement ("If P, then Q") is true and the antecedent (P) is true, then the consequent (Q) must also be true.

$P \rightarrow Q$ (If P then Q)
P (P is true)
Conclusion : Q (∴ Q is true)

Eg-
1.) If a student studies hard, they will pass the exam ($S \rightarrow P$)
2.) The student studies hard (S)
3.) Conclusion : The student will pass the exam. (P).

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

Ans- Forward chaining is a data driven reasoning technique. It starts with known facts and applies inference rules to derive new facts until the goal is found.

Eg-
Rule 1 : If it rains, the ground becomes wet. ($R \rightarrow W$)
Rule 2 : If the ground is wet, the road is slippery ($W \rightarrow S$)
Fact : It is raining.

Process :
1.) Given : R (it is raining)
2.) Apply rule 1 : $R \rightarrow W \rightarrow W$ (ground is wet).
3.) Apply rule 2 : $W \rightarrow S \rightarrow S$ (road is slippery)

Teacher's Sign.: _____

4.) Goal reached : S ✓

Backward Chaining is a goal driven reasoning technique. It starts with goal & works backward to check if the known facts support it.

Eg -

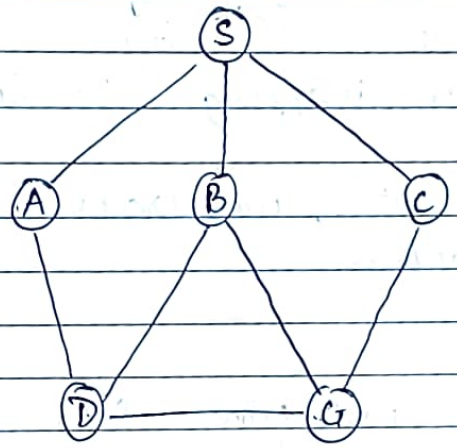
Goal : Is the road slippery ? (S?)
Rule 1 : W → S (If ground is wet, road is slippery)
Rule 2 : R → W (If it rains, the ground is wet.)
Fact : R (it is raining)

Process :

1.) Goal : Is S true?
2.) Check Rule 1 : W → S, so we need W.
3.) Check Rule 2 : R → W, so we need R.
4.) Given fact : R is true.
5.) Derive W (Ground is wet)
6.) Derive S (Road is slippery) ✓.

Teacher's Sign.:

Q10.) Find the route from S to G



Ans- BFS explores the graph level by level starting from the source nodes.

Step1 :

1.) Start at S : Add S to the queue.

Queue : [S]

Expand S : Visit its neighbour

Queue : [A, B, C]

Parent Mapping

S → A , S → B , S → C

Expand A : Visit D

Queue : [B, C, D]

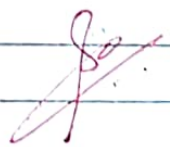
Parent Mapping : A → D

Expand B : Visit G (goal found)
Queue : [C, D, G]
Parent mapping : B → G

Since we found G, we trace back the path using parent mapping

Step 2 :

Shortest path from S to G is
S → B → G



Name: Shiven Bansal **Roll No : 03** **Class: D15C**

AIDS - 1 Assignment 2

Q.1: Use the following data set for question 1
82, 66, 70, 59, 90, 78, 76, 95, 99, 84, 88, 76, 82, 81, 91, 64, 79, 76, 85, 90
1. Find the Mean (10pts)
2. Find the Median (10pts)
3. Find the Mode (10pts)
4. Find the Interquartile range (20pts)

Ans

1. Mean

Mean = (Sum of all values) / (Number of values)
Sum = 82 + 66 + 70 + 59 + 90 + 78 + 76 + 95 + 99 + 84 + 88 + 76 + 82 + 81 + 91 + 64 + 79 + 76 + 85 + 90 = 1621
Count = 20

Mean = 1621 / 20 = 81.05

2. Median

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

Step 2: Find the middle value:
For 20 numbers (even count), median = average of 10th and 11th values
10th = 81, 11th = 82

Median = (81 + 82) / 2 = 81.5

3. Mode (10 pts)

The value that appears most frequently:
76 appears 3 times, which is more than any other value

Mode = 76

4. Interquartile Range (IQR) (20 pts)

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

Step 2: Split into two halves:

- Lower half (Q1 group) = 59, 64, 66, 70, 76, 76, 76, 78, 79, 81

- Upper half (Q3 group) = 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Q1 = Median of lower half = (5th + 6th)/2 = (76 + 76)/2 = 76
Q3 = Median of upper half = (5th + 6th)/2 = (88 + 90)/2 = 89

IQR = Q3 - Q1 = 89 - 76 = 13

Q.2: Comparison and Analysis of Machine Learning for Kids and Teachable Machine

Ans

1) Machine Learning for Kids

a) Target Audience:

- Primary: School students (ages 8–16)
- Secondary: Teachers, beginners in AI/ML, and educators introducing AI in a simplified form

b) Use of this tool by the target audience:

- Students use it to train models by giving examples (e.g., recognizing text, images, or numbers).
- Often used in combination with Scratch or Python to create interactive projects.

c) Tool's Benefits and Drawbacks:
Benefits:

- Simple and kid-friendly interface
- Promotes creativity and logical thinking
- Integrates with Scratch/Python
- Free and includes educational content

Drawbacks:

- Limited for advanced ML applications
- Slower training (uses IBM Watson)
- No deep customization options

d) Type of Analytic:
Descriptive Analytic
Reason: It helps users understand and describe existing patterns in the data rather than predicting future outcomes.

e) Type of Learning:
Supervised Learning

Reason: The user provides labeled data examples for training the model, which is a supervised learning approach.

2) Teachable Machine

a) Target Audience:

- Beginners, hobbyists, students, teachers, content creators, and anyone curious about AI without coding knowledge

b) Use of this tool by the target audience:

- Users train models using webcam images, audio, or body poses.
- Models can be tested instantly and exported for real-world use.

c) Tool's Benefits and Drawbacks:
Benefits:

- No coding required
- Real-time model training
- Allows exporting models for use in apps/websites
- Fast and easy to use

Drawbacks:

- Limited to basic classification tasks
- No deep algorithmic control
- Unsuitable for large or complex datasets

d) Type of Analytic:
Descriptive Analytic
Reason: The tool is used to classify and understand current data patterns, not to forecast trends or future values.

e) Type of Learning:
Supervised Learning
Reason: The user labels data while training (e.g., labeling poses or sounds), fitting the definition of supervised learning.

Q3. Data Visualization: Read the following two short articles:

- Read the article Kakande, Arthur. February 12. "What's in a chart? A Step-by-Step guide to Identifying Misinformation in Data Visualization." Medium
- Read the short web page Foley, Katherine Ellen. June 25, 2020. "How bad Covid-19 data visualizations mislead the public." Quartz

- Research a current event which highlights the results of misinformation based on data visualization.
  Explain how the data visualization method failed in presenting accurate information. Use newspaper articles, magazines, online news websites or any other legitimate and valid source to cite this example. Cite the news source that you found.

Ans

Data Visualization: Misinformation in Current Events

Case Study: Misleading Claims About COVID-19 Vaccine Safety

Source: Reuters Fact Check, March 21, 2024
Link: Reutersvdl.sci.utah.edu

Overview

In March 2024, social media platforms were inundated with claims suggesting that COVID-19 vaccines caused more harm than good. These assertions were based on data indicating a higher number of deaths among vaccinated individuals in England between July 2021 and May 2023. Such claims were propagated by outlets like Natural News and Chemical Violence.

How the Data Visualization Was Misleading

1. **Lack of Contextualization:**
   The visualizations failed to account for the fact that a significantly larger portion of the population was vaccinated during the specified period. Naturally, with more vaccinated individuals, the absolute number of deaths in this group would be higher, but this does not imply higher risk.

2. **Misrepresentation of Rates:**
   The charts presented raw death counts without adjusting for population size, leading to a skewed perception. When considering mortality rates per 100,000 individuals, the unvaccinated group exhibited higher death rates.

3. **Selective Data Presentation:**
   By focusing solely on absolute numbers without proportional context, the visualizations led viewers to erroneous conclusions about vaccine safety.

Impact of the Misleading Visualization

These deceptive visualizations contributed to vaccine hesitancy and fueled misinformation narratives. Public health experts emphasized the importance of interpreting data within the correct context to avoid such misunderstandings.

Conclusion

This case underscores the critical need for responsible data visualization practices. Accurate representation, contextualization, and transparency are essential to convey truthful information and

maintain public trust.

Q4. Train Classification Model and visualize the prediction performance of trained model required information

Ans.
Dataset Used: Pima Indians Diabetes Database
1.

```
[1]  # Import Required Libraries
     import pandas as pd
     from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.naive_bayes import GaussianNB
     from sklearn.metrics import classification_report, confusion_matrix
     from imblearn.over_sampling import SMOTE
```

This step imports all the necessary Python libraries. pandas is for data handling, StandardScaler is for feature scaling, train_test_split splits the dataset, GaussianNB is the Naive Bayes model, and SMOTE is used to fix class imbalance.

2.

```
# Load the dataset
df = pd.read_csv('diabetes.csv')
```

This step loads the dataset diabetes.csv

3.

```
[3]  # Split Features and Labels
     X = df.drop('Outcome', axis=1)
     y = df['Outcome']
```

We separate the dataset into input features X and the target label y. Here, X includes medical measurements and y is the class label that we aim to predict.

4.

```
#  Normalize the Features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Feature scaling is applied using StandardScaler to ensure all values have a similar range. This helps the Naive Bayes classifier handle features that vary in scale.

5.

```
[5]  # Split into Train, Validation, and Test (70/20/10)
     X_train, X_temp, y_train, y_temp = train_test_split(X_scaled, y, test_size=0.30, random_state=42, stratify=y)
     X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, random_state=42, stratify=y_temp)
```

The data is split randomly into 70% training, 20% validation, and 10% testing. This ensures the model is trained well, tuned properly, and tested fairly. stratify=y keeps the class distribution balanced in all splits.

6.

```
[6]  from collections import Counter
     from imblearn.over_sampling import SMOTE

     # Before applying SMOTE
     print("Class distribution before SMOTE:", Counter(y_train))

     # Apply SMOTE to balance the training set
     smote = SMOTE(random_state=42)
     X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)

     # After applying SMOTE
     print("Class distribution after SMOTE:", Counter(y_train_bal))

Class distribution before SMOTE: Counter({0: 350, 1: 187})
Class distribution after SMOTE: Counter({1: 350, 0: 350})
```

Before applying SMOTE, the class distribution was imbalanced with 350 non-diabetic (class 0) and 187 diabetic (class 1) samples.

After applying SMOTE, the dataset was balanced to have 350 samples in each class.This ensures that the Naive Bayes model is trained on an equal number of diabetic and non-diabetic cases, helping it avoid bias toward the majority class and improving its ability to correctly identify diabetic cases.

7.

```
# Train the Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train_bal, y_train_bal)

▾ GaussianNB
GaussianNB()
```

We use the GaussianNB model, which is a type of Naive Bayes algorithm for continuous input features. The model is trained using the balanced training data.

8.

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

# Evaluate the Model on the Test Set
test_preds = nb_model.predict(X_test)

# Print Confusion Matrix
print("Confusion Matrix:\n", confusion_matrix(y_test, test_preds))

# Print Classification Report
print("\nClassification Report:\n", classification_report(y_test, test_preds))

# Print Accuracy
accuracy = accuracy_score(y_test, test_preds)
print(f"Accuracy of the Naive Bayes model: {accuracy:.2f}")
```

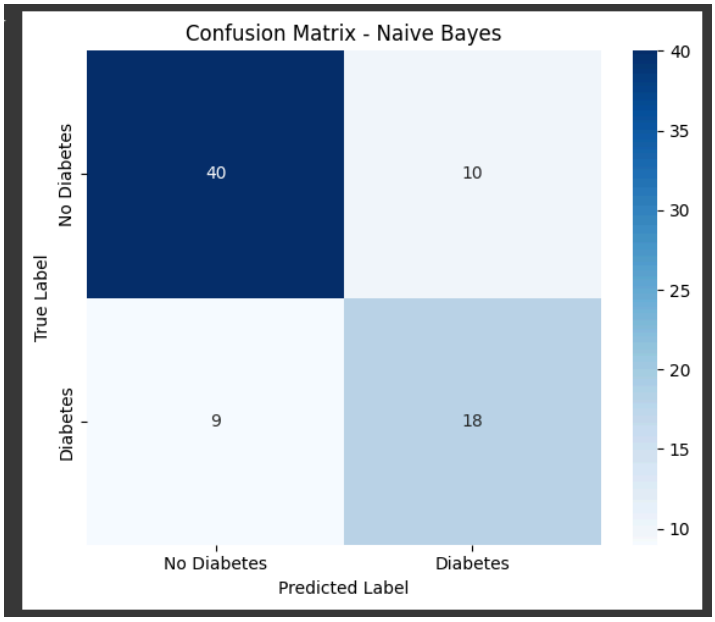
```
Confusion Matrix:
 [[40 10]
 [ 9 18]]

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.80      0.81        50
           1       0.64      0.67      0.65        27

    accuracy                           0.75        77
   macro avg       0.73      0.73      0.73        77
weighted avg       0.76      0.75      0.75        77

Accuracy of the Naive Bayes model: 0.75
```

The Naive Bayes model achieved an accuracy of 75%, correctly classifying most test samples. The confusion matrix shows good prediction for non-diabetic cases, but performance on diabetic cases is slightly lower, with an F1-score of 0.65. While the model performs reasonably well overall, there is room for improvement in detecting diabetic patients more accurately.

From the confusion matrix we see that :
True Negatives (40): The model correctly predicted 40 non-diabetic patients as non-diabetic.
False Positives (10): The model incorrectly predicted 10 non-diabetic patients as diabetic.
False Negatives (9): The model incorrectly predicted 9 diabetic patients as non-diabetic.
True Positives (18): The model correctly predicted 18 diabetic patients as diabetic.

9.

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# After making predictions with test_preds
cm = confusion_matrix(y_test, test_preds)

plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=["No Diabetes", "Diabetes"],
            yticklabels=["No Diabetes", "Diabetes"])
plt.title("Confusion Matrix - Naive Bayes")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.tight_layout()
plt.show()
```



The heatmap above provides a clear visual summary of the model's predictions. The darker blue boxes indicate higher correctly classified values, making it easier to quickly identify strengths and weaknesses in prediction accuracy.

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

Ans

```python
# Block 1: Import Libraries and Define the Regression Class

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

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.linear_model import Ridge
from sklearn.multioutput import MultiOutputRegressor
from sklearn.metrics import r2_score
from sklearn.pipeline import Pipeline

# Define our RegressionModel class using OOP
class RegressionModel:
    def __init__(self, csv_path):
        """
        Initialize by loading data from csv.
        """
        # Read the data
        self.data = pd.read_csv(csv_path, sep='\s+' if '\t' in open(csv_path).read(100) else ',')
        # Print the first few rows for confirmation
        print("Data head:")
        print(self.data.head())

    def preprocess(self):
        """
        Preprocess the data:
          - Set the first column as predictor (X)
          - Set the remaining columns as targets (y)
          - Standardize the predictor variable
        """
        # Get the column names
        cols = self.data.columns.tolist()
        if len(cols) < 2:
            raise ValueError("Insufficient columns. Expecting at least 2 columns.")
```

This code defines a regression pipeline using object-oriented programming. It loads and preprocesses data, splits it into training and test sets, and applies polynomial regression with Ridge regularization. Hyperparameters are tuned using GridSearchCV, and the model handles multiple outputs using MultiOutputRegressor. Finally, it evaluates performance with R^2 and adjusted R^2 scores and prints regression equations for each target variable.

```python
# Block 2: Instantiate RegressionModel, Preprocess data

csv_path = '/content/diabetes.csv'  # update this path if needed
reg_model = RegressionModel(csv_path)
reg_model.preprocess()  # This call uses try/except to catch potential KeyErrors.
```

```
Data head:
   id  relwt  glufast  glutest  steady  insulin  group
0   1   0.81       80      356     124       55      3
1   3   0.94      105      319     143      105      3
2   5   1.00       90      323     240      143      3
3   7   0.91      100      350     221      119      3
4   9   0.99       97      379     142       98      3
Preprocessing done. Shape of X: (144, 1) and y: (144, 6)
```

This block initializes the regression model using the diabetes.csv file and preprocesses the data. The first column (relwt) is used as the predictor (X), and the remaining six columns (glufast, glutest, steady, insulin, group, etc.) are treated as target variables (y). The predictor is also standardized. The output confirms that the data has 144 samples, with X shaped as (144, 1) and y as (144, 6).

```
# Block 3: Split the data
reg_model.split_data(test_size=0.30, random_state=42)
```

```
Train set size: 100
Test set size: 44
```

This block splits the preprocessed data into training and testing sets using a 70-30 ratio. Out of 144 total records, 100 samples are used for training and 44 for testing. The random_state=42 ensures reproducibility of the split.

```
[ ] # Block 4: Tune hyperparameters and train the model
    reg_model.tune_and_train()
```

```
Starting hyperparameter tuning...
Best hyperparameters found: {'estimator__poly__degree': 5, 'estimator__ridge__alpha'
Best CV R2 score: 0.3778
```

This block performs hyperparameter tuning using GridSearchCV. It finds the best combination of polynomial degree and Ridge alpha for the regression model. The best parameters selected are degree 5 and alpha 10, with a cross-validation R^2 score of 0.3778.

```
# Block 5: Evaluate the model's performance and show regression equations
r2_scores, adj_r2_scores = reg_model.evaluate()
```

```
Regression Equations and Performance on Test Set:

Dependent variable: 'relwt'
R2 Score: 0.1033
Adjusted R2 Score: 0.0820
Regression Equation:
(0.0000)*1 + (0.0645)*x + (-0.0892)*x^2 + (-0.0529)*x^3 + (0.0236)*x^4 + (0.0147)*x^5 + (1.0289)

Dependent variable: 'glufast'
R2 Score: 0.6040
Adjusted R2 Score: 0.5946
Regression Equation:
(0.0000)*1 + (14.0771)*x + (30.7190)*x^2 + (21.1249)*x^3 + (-2.1056)*x^4 + (-4.1703)*x^5 + (93.0742)

Dependent variable: 'glutest'
R2 Score: 0.6518
Adjusted R2 Score: 0.6435
Regression Equation:
(0.0000)*1 + (146.5712)*x + (209.7457)*x^2 + (110.2991)*x^3 + (-38.7953)*x^4 + (-31.7118)*x^5 + (398.5588)

Dependent variable: 'steady'
R2 Score: -0.0821
Adjusted R2 Score: -0.1078
Regression Equation:
(0.0000)*1 + (172.7161)*x + (-81.7289)*x^2 + (-197.6247)*x^3 + (12.8359)*x^4 + (46.7468)*x^5 + (253.2218)

Dependent variable: 'insulin'
R2 Score: 0.5409
Adjusted R2 Score: 0.5300
Regression Equation:
(0.0000)*1 + (78.3158)*x + (12.6140)*x^2 + (10.9376)*x^3 + (-1.3101)*x^4 + (-6.7065)*x^5 + (179.9129)

Dependent variable: 'group'
R2 Score: 0.8854
Adjusted R2 Score: 0.8826
Regression Equation:
(0.0000)*1 + (-0.6355)*x + (-0.5174)*x^2 + (-0.2631)*x^3 + (0.1008)*x^4 + (0.1023)*x^5 + (2.6235)

WARNING: Not all dependent variables achieved an Adjusted R2 > 0.99.
Dependent variable 'relwt': Adjusted R2 = 0.0820
Dependent variable 'glufast': Adjusted R2 = 0.5946
Dependent variable 'glutest': Adjusted R2 = 0.6435
Dependent variable 'steady': Adjusted R2 = -0.1078
Dependent variable 'insulin': Adjusted R2 = 0.5300
Dependent variable 'group': Adjusted R2 = 0.8826
```

This block evaluates the model's performance on the test set for each dependent variable using R^2 and Adjusted R^2 scores. Here's a short summary:

- Best model performance was for the group variable (Adjusted R^2: 0.8262).
- Weakest performance was for steady (Adjusted R^2: -0.1078).
- All models used a 5-degree polynomial regression.

Q6. 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.

Ans.

Key Features of the Wine Quality Dataset

The wine quality dataset contains several physicochemical features that describe the chemical composition of wine samples. Each feature plays a different role in influencing the quality score, which

is typically given on a scale from 0 to 10. The key features include:

1. **Fixed Acidity**: Represents non-volatile acids such as tartaric acid. It plays a role in maintaining the wine's stability and overall flavor profile.

2. **Volatile Acidity**: Indicates the amount of acetic acid in wine. Higher values often lead to an unpleasant vinegar taste, negatively affecting wine quality.

3. **Citric Acid**: A natural preservative that adds freshness and enhances flavor. Moderate levels usually have a positive impact on perceived quality.

4. **Residual Sugar**: The amount of sugar left after fermentation. While some sweetness can improve taste, excessive levels may not necessarily correlate with high quality.

5. **Chlorides**: Reflects the salt content in wine. High levels can make the wine taste briny and reduce its overall appeal.

6. **Free Sulfur Dioxide**: Helps prevent microbial growth and oxidation. However, very high concentrations may result in a sharp or chemical taste.

7. **Total Sulfur Dioxide**: The combined amount of free and bound sulfur dioxide. Excessive levels can negatively affect taste and consumer perception.

8. **Density**: Closely related to sugar and alcohol content. Higher density could suggest sweeter wines, though not always linked with quality.

9. **pH**: Measures the acidity level. Affects both taste and microbial stability; however, its direct correlation with quality is generally weaker.

10. **Sulphates**: Function as preservatives. Moderate sulphate levels can enhance flavor and antimicrobial protection.

11. **Alcohol**: One of the most influential features. Generally, wines with higher alcohol content are rated better in quality assessments.

12. **Quality:** The quality column is the target variable representing the wine's overall taste rating, typically on a scale from 0 to 10. It is crucial for training models to predict consumer-perceived wine quality based on chemical properties.

Importance of Each Feature in Predicting Wine Quality

Not all features contribute equally to predicting wine quality. Based on domain knowledge and data analysis:

- **Highly important features**:
    - **Alcohol** has a strong positive correlation with quality, as higher alcohol content is often associated with better flavor and richness.
    - **Volatile Acidity** has a negative correlation; higher values can make wine taste sour.
    - **Sulphates** and **citric acid** also show positive contributions.
- **Moderately important features**:
    - **Residual sugar**, **chlorides**, and **pH** can influence quality but are less strongly correlated.
    - **Sulfur dioxide** levels impact preservation but can degrade sensory quality if too high.

Handling Missing Data during Feature Engineering

During the feature engineering process, handling missing data is crucial to maintain data integrity and ensure accurate predictions. In the wine dataset, missing values (if present) were carefully analyzed and imputed using appropriate strategies.

The **median imputation technique** was used, as it is robust to outliers and works well for continuous numerical features, which are predominant in this dataset.

Advantages and Disadvantages of Different Imputation Techniques

Imputation Method	Advantages	Disadvantages
Dropping missing rows/columns	Simple and clean	Data loss, especially harmful with large missing percentages.

Mean Imputation	Easy to apply; maintains dataset size	Affected by outliers; can reduce variability
Median Imputation	Robust to outliers; preserves distribution better	May not capture relationships between variables
Mode Imputation	Useful for categorical variables	Not suitable for continuous data
Multivariate Imputation	Considers inter-feature relationships	Complex and time-consuming; requires careful tuning
Model-Based Imputation (e.g., regression)	Potentially more accurate	Risk of introducing bias if not validated properly.