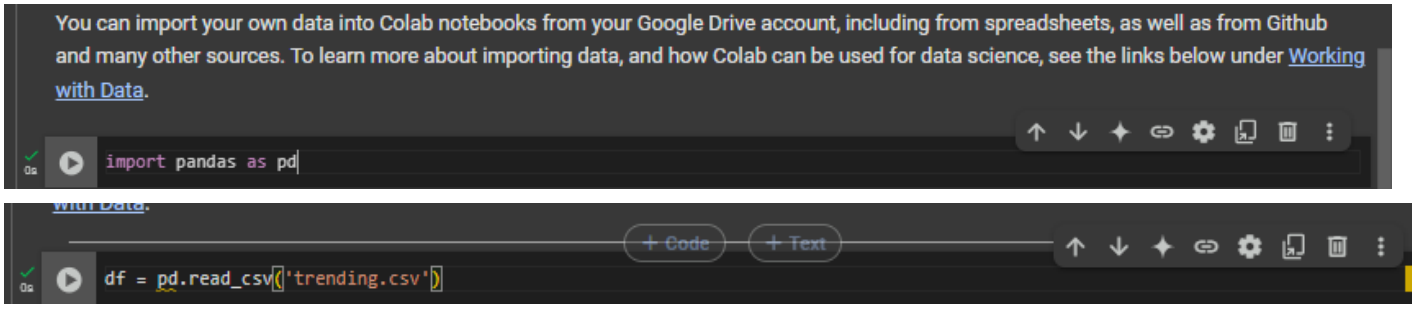
NAME: VEDANT SURESH DHOKE CLASS/ROLL NO: D15C/ 9

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

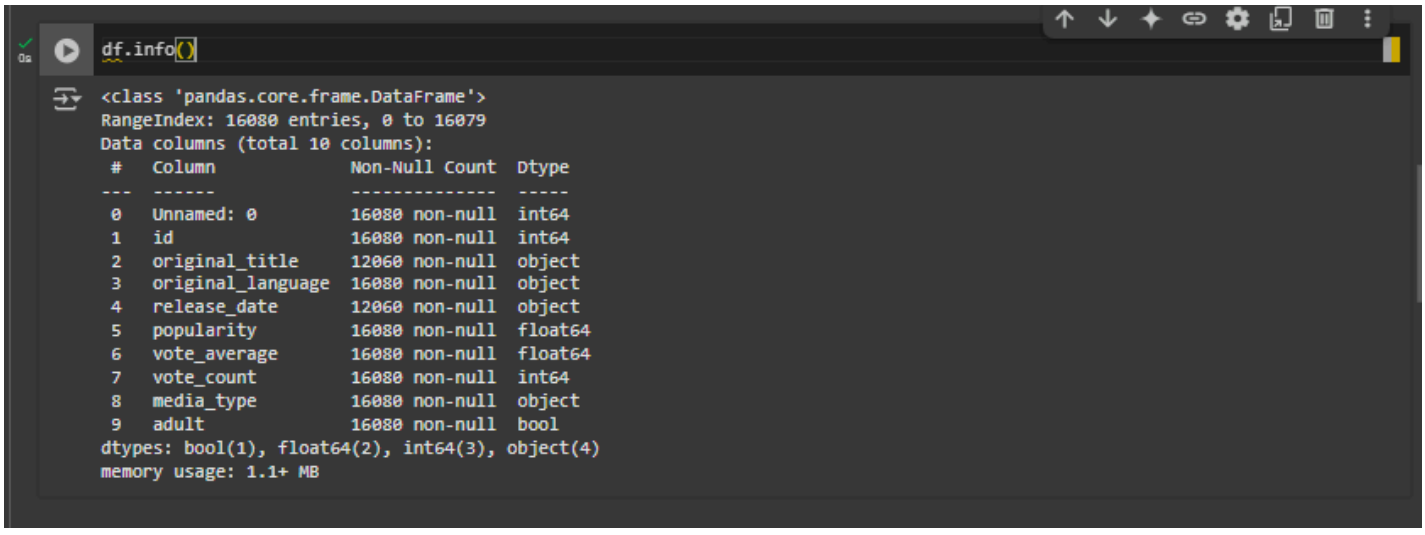
Step 1: Firstly import Pandas Library as pd an then Load data in Pandas using pd.read_csv.



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: VEDANT SURESH DHOKE CLASS/ROLL NO: D15C/ 9

```
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

```
df= df.drop(df.isnull().sum(axis=1).idxmax())
```

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

```
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 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.

```
numeric_columns = df.select_dtypes(include=['float64', 'int64']).columns
```

```
df[numeric_columns] = df[numeric_columns].fillna(df[numeric_columns].median())
```

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

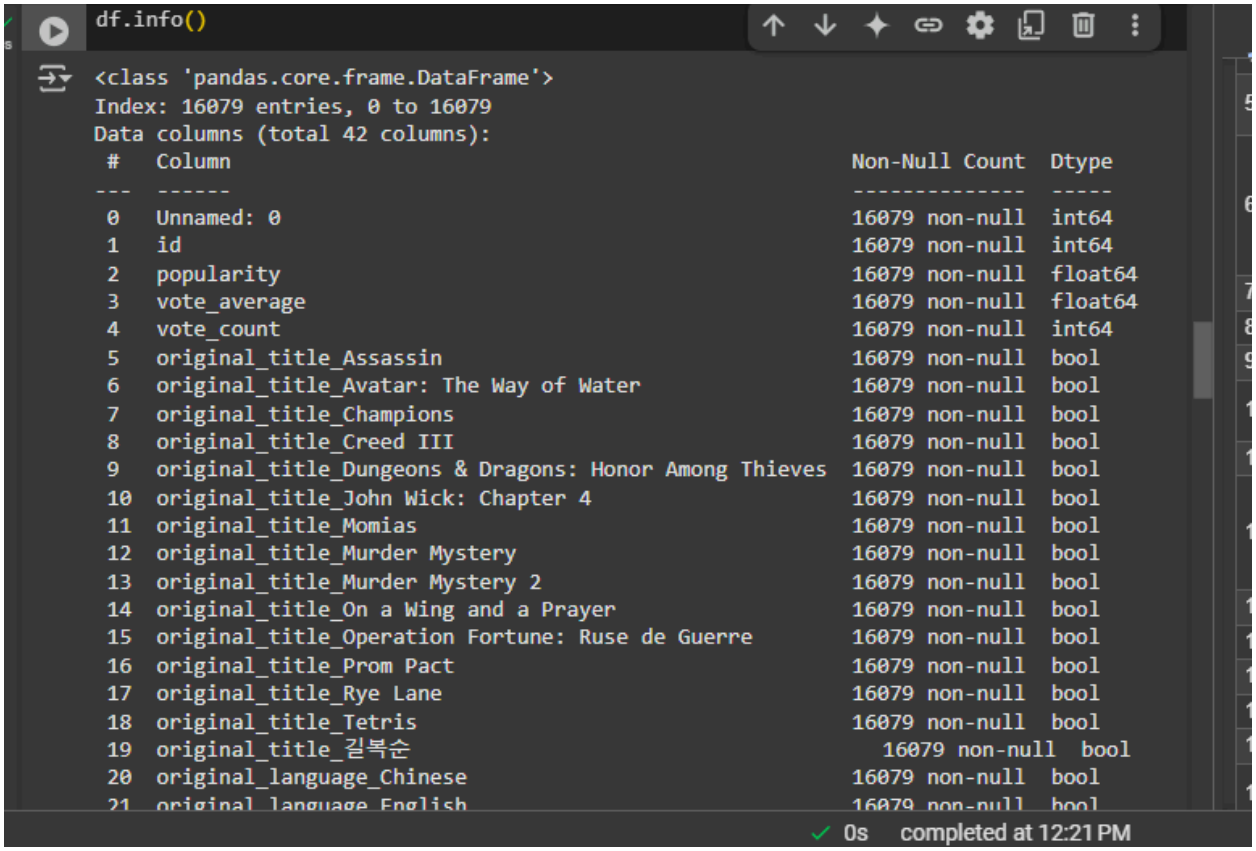
```
df = pd.get_dummies(df)
```

+ 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: VEDANT SURESH DHOKE CLASS/ROLL NO: D15C/ 9

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:Vedant Dhoke **Class/Roll No: D15C/9**

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')
```

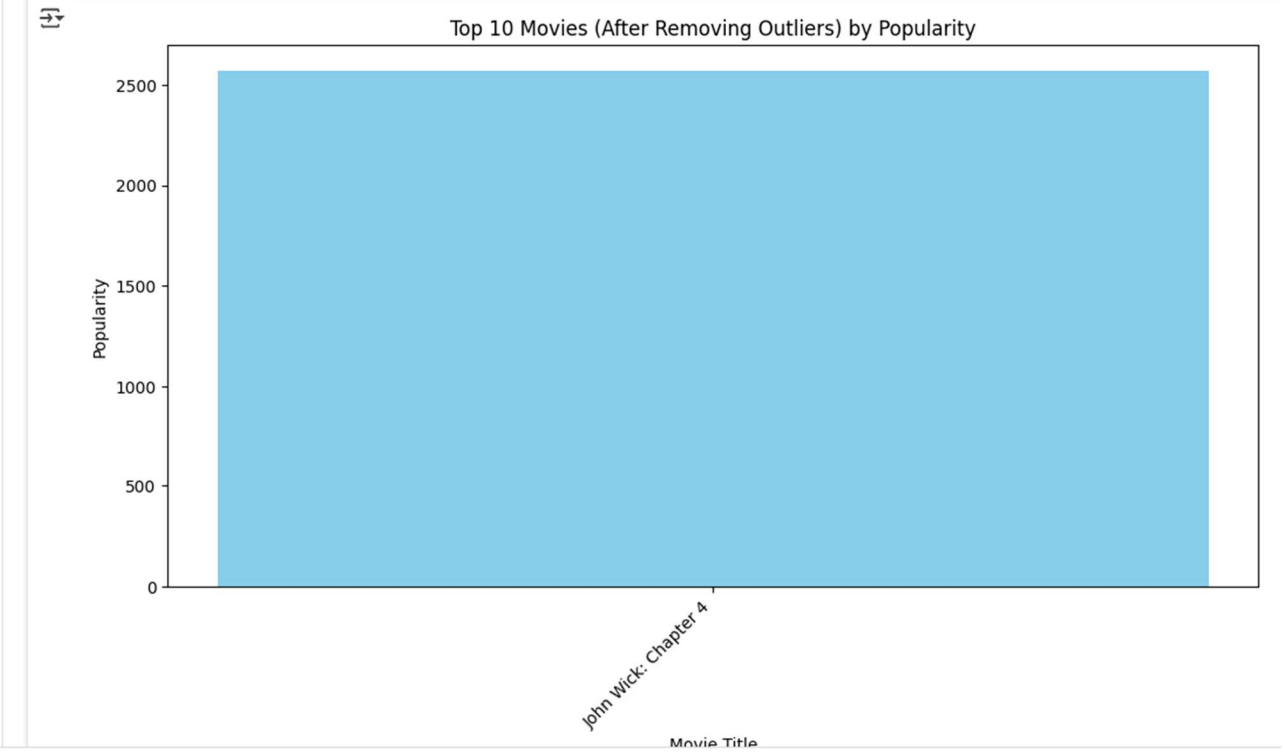
This step detects and removes outliers from the 'popularity' column using the Interquartile Range (IQR) method. The first quartile (Q1) and third quartile (Q3) define the middle 50% of the data, and the IQR (Q3 - Q1) is used to calculate outlier boundaries. Any value beyond 1.5 times the IQR from Q1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

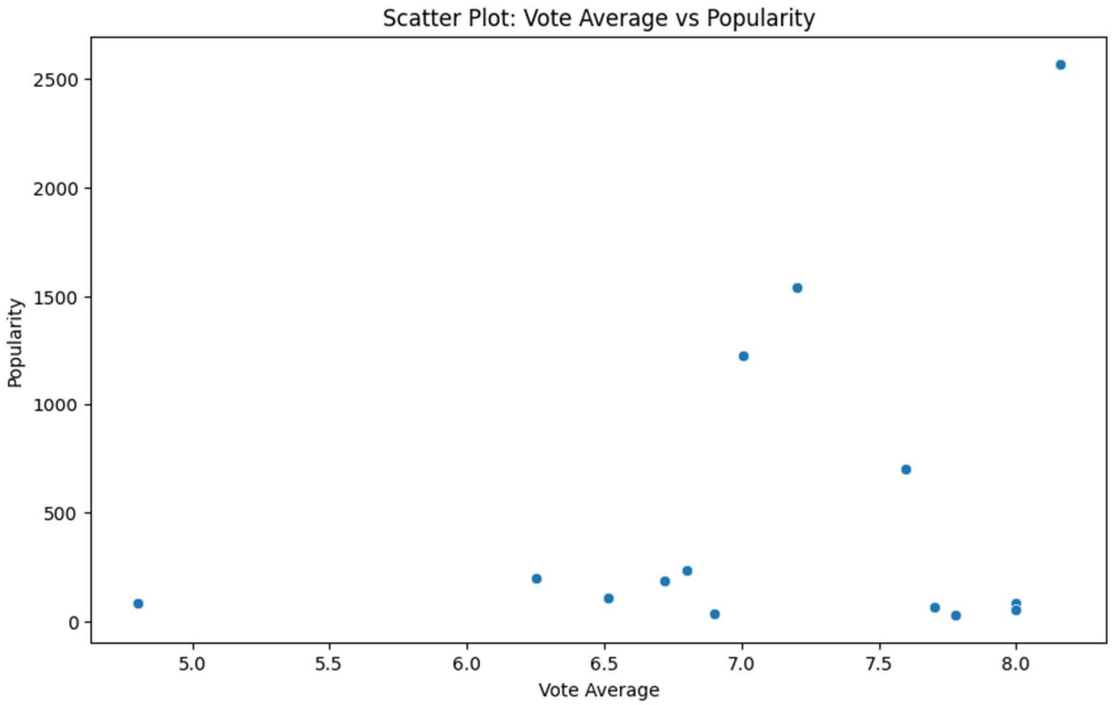
Observations:

- 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:Vedant Dhoke **Class/Roll No: D15C/9**

```
#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.

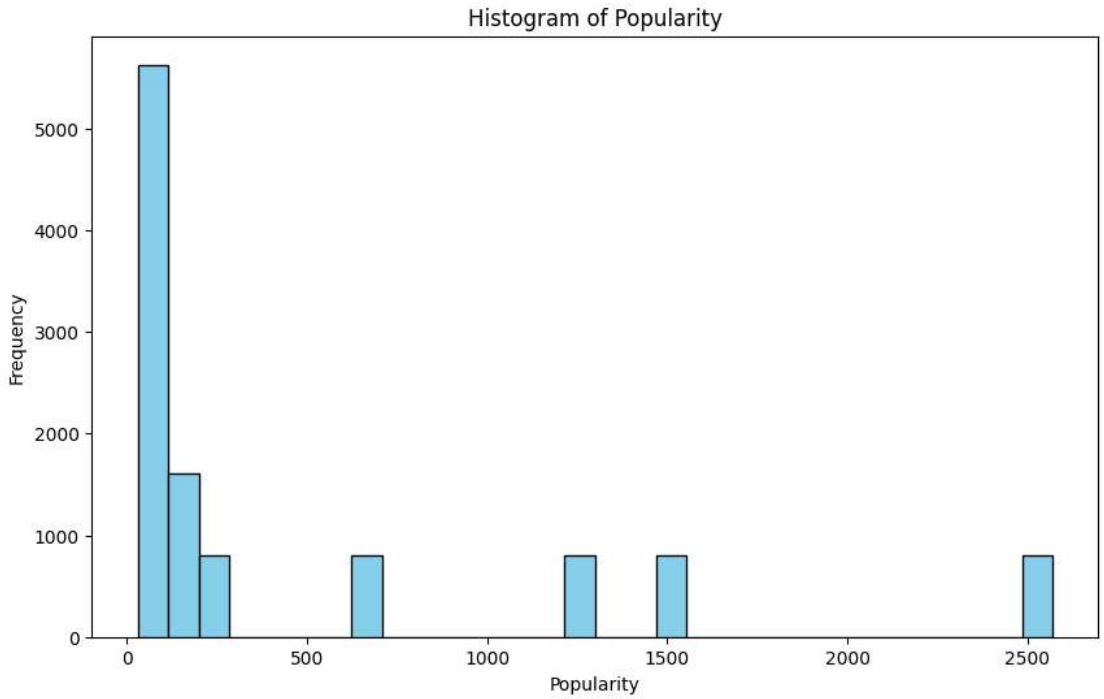
Observations:

- 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:Vedant Dhoke **Class/Roll No: D15C/9**

```
#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.

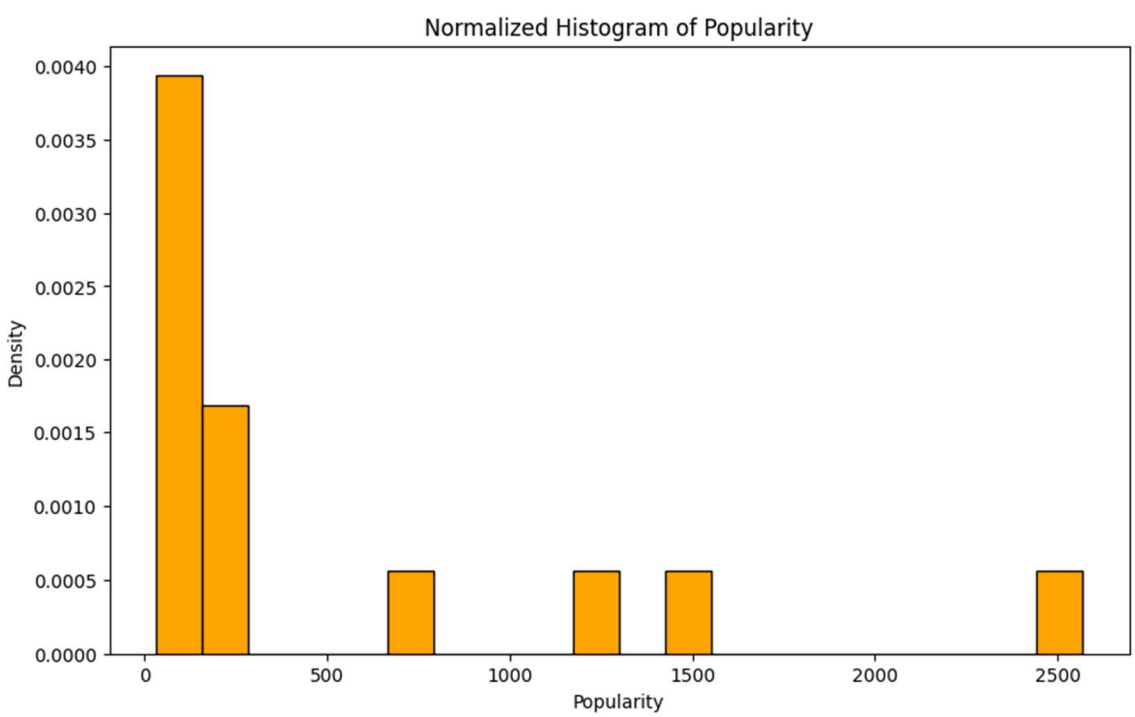
Observations:

- 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:Vedant Dhoke **Class/Roll No: D15C/9**

```
#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.

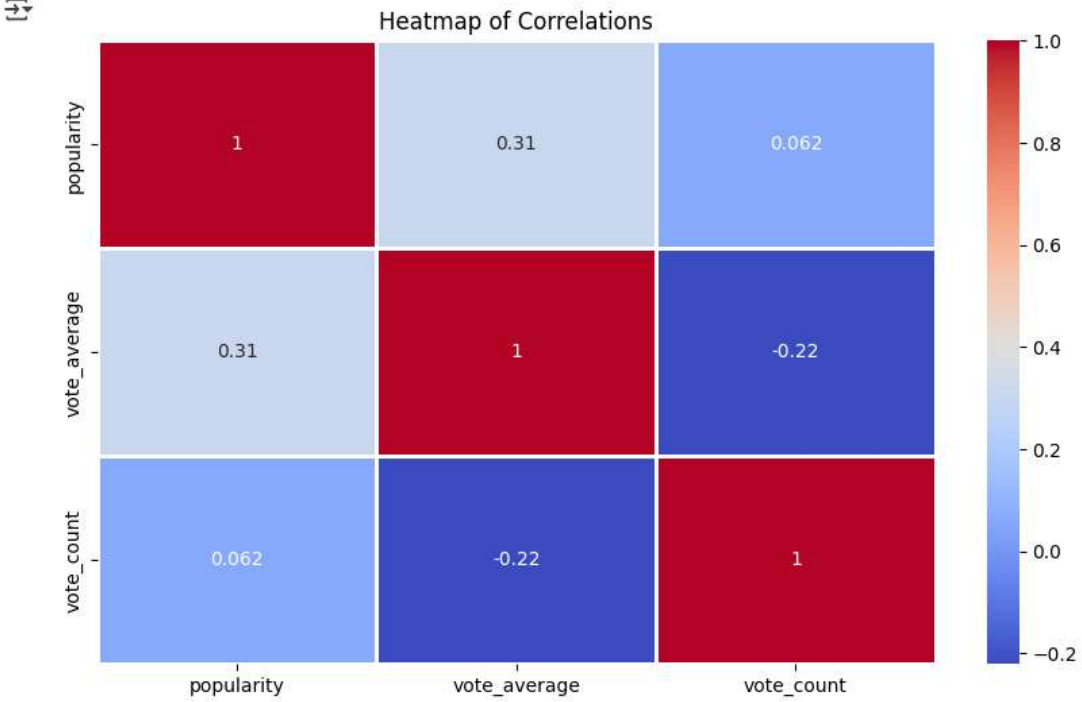
Observations:

- 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:Vedant Dhoke **Class/Roll No: D15C/9**

```
#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()
```

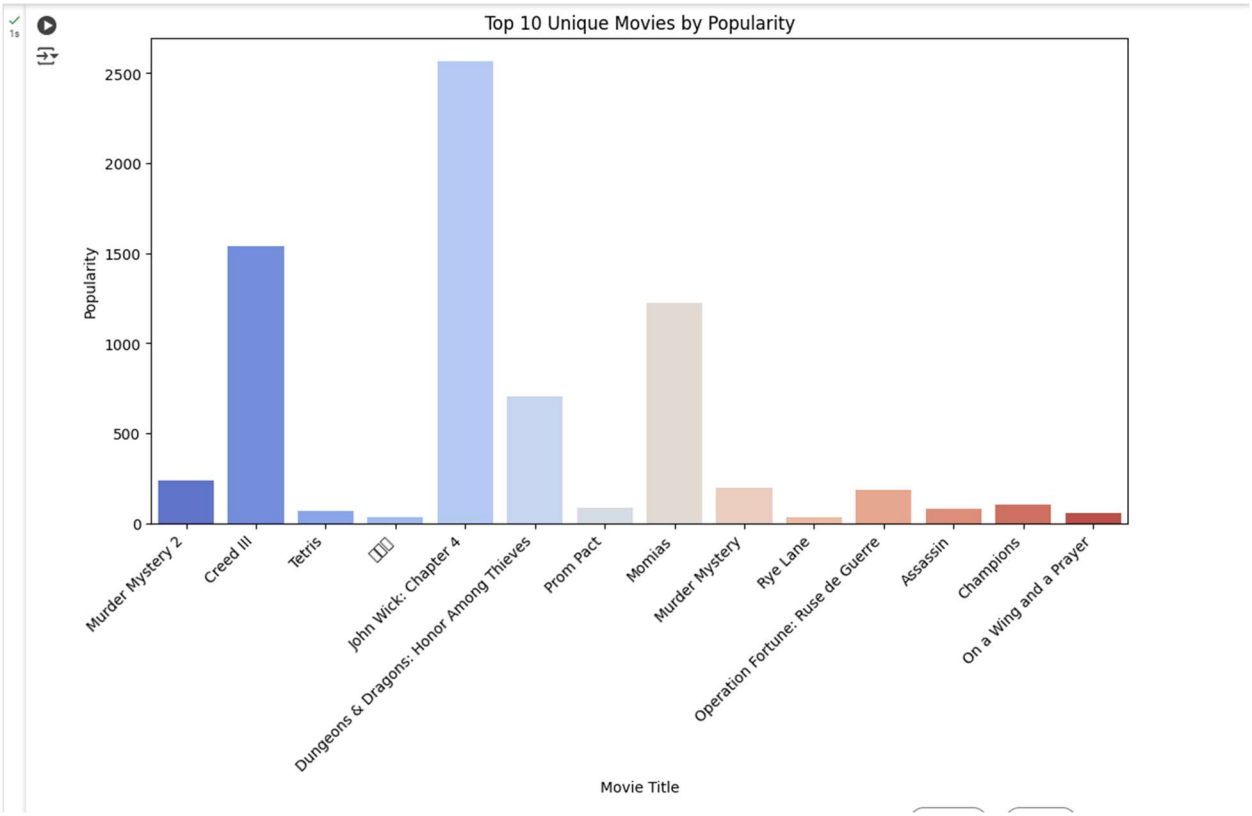


- Popularity vs Vote Average: Weak positive correlation (0.31) → More popular movies tend to have slightly higher ratings.
- Popularity vs Vote Count: Almost no correlation (0.062) → A movie's popularity does not strongly relate to how many people voted.
- Vote Average vs Vote Count: Slight negative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

- **Bar Graph**

Name:Vedant Dhoke **Class/Roll No: D15C/9**

```
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.

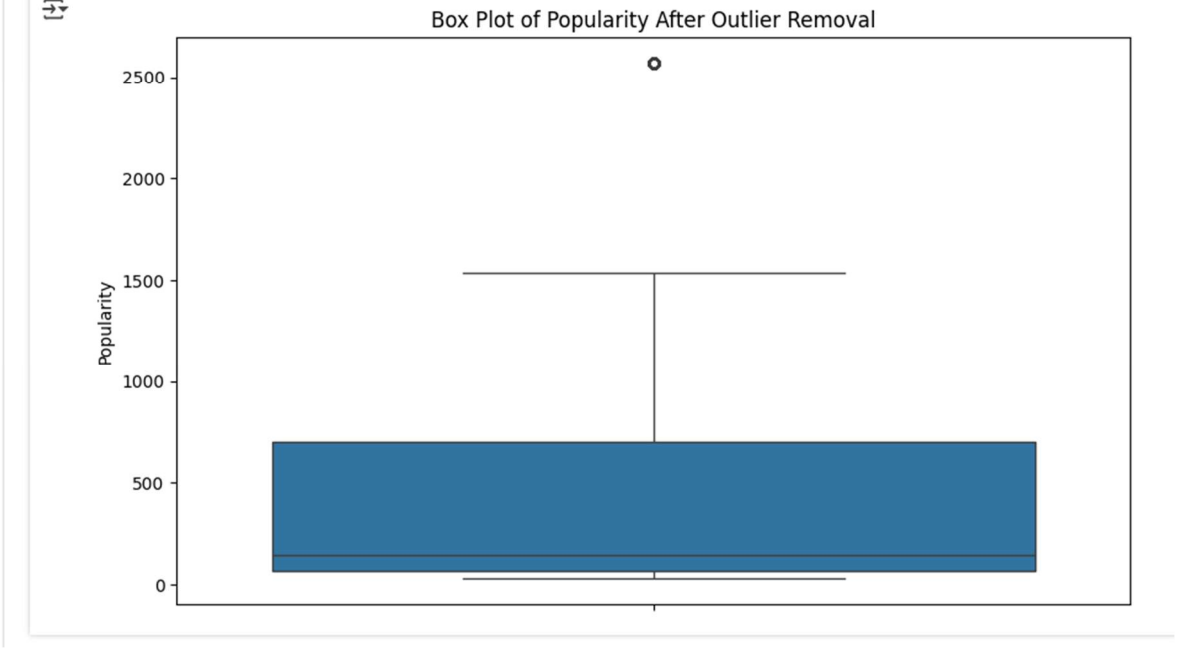
Key Observations:

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:Vedant Dhoke **Class/Roll No: D15C/9**

- **Bar Chart**

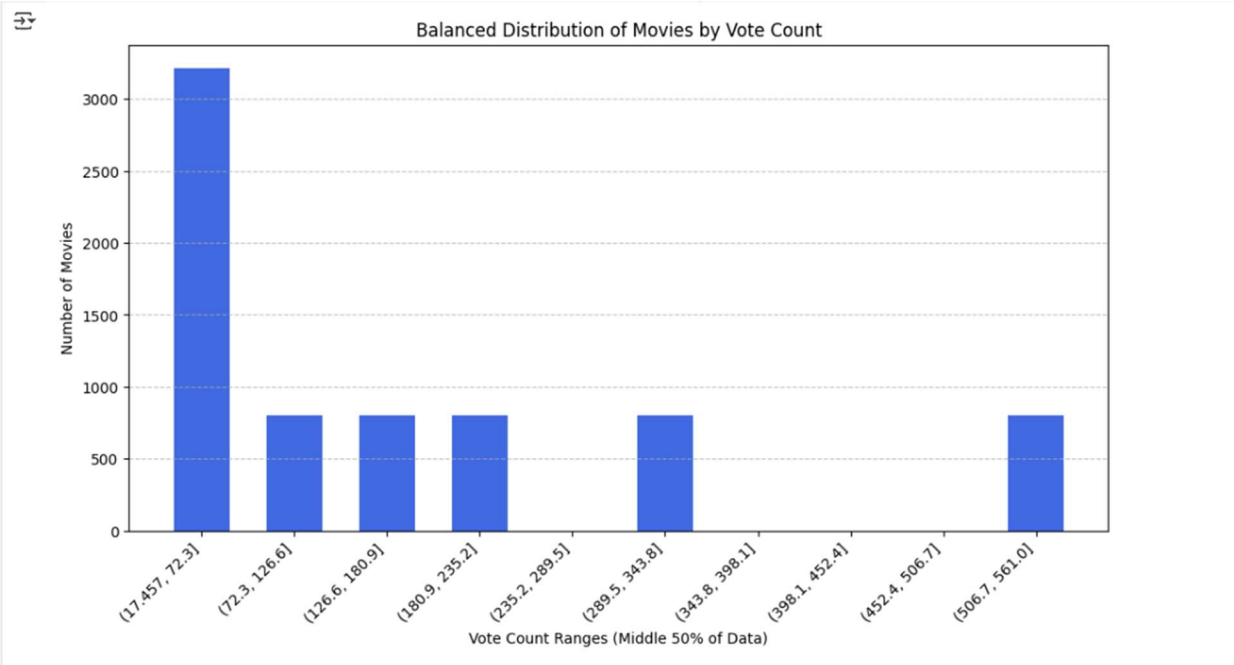
```
Q1 = df['vote_count'].quantile(0.25)
Q3 = df['vote_count'].quantile(0.75)
df_no_outliers = df[(df['vote_count'] >= Q1) & (df['vote_count'] <= Q3)]
```

```
[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:Vedant Dhoke Div-D15C Roll No-9 A.Y.-2024-25

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.

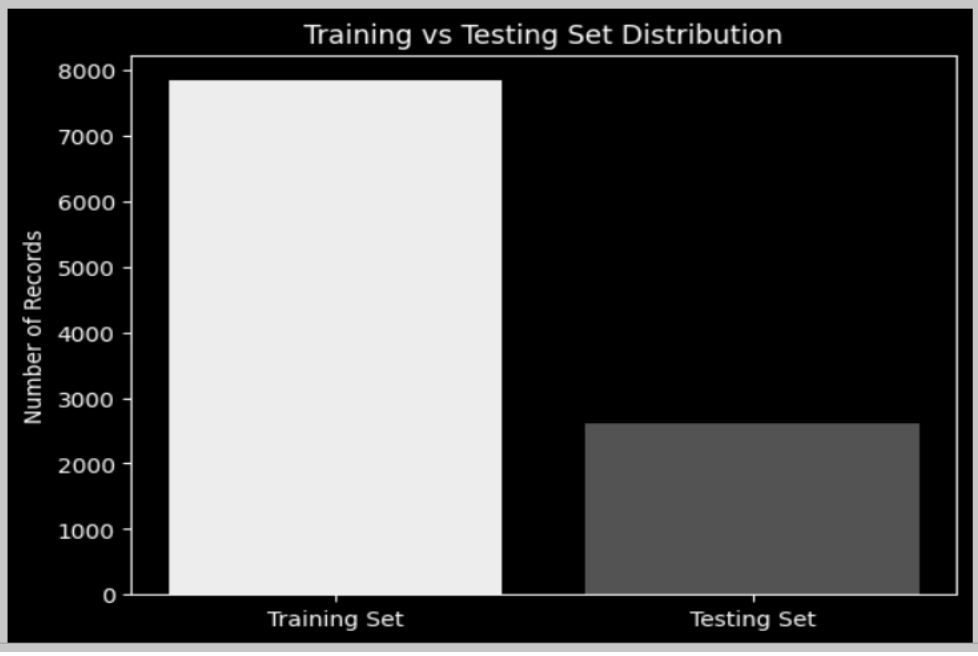
   test_size=0.25: 25% of the dataset is assigned to the test set, and the remaining 75% goes to the training set.

```
[ ] 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: Vedant Dhoke Div-D15C Roll No-9 A.Y.-2024-25

* Manual Z-score calculation

Column Name	mean_train	mean_test	std_train	std_test	n_train	n_test
id	656663.5	652356	305596.4	309181	7839	2613
popularity	224.5	220	338.6	330	7839	2613
vote_average	7.54	7.54	0.66	0.65	7839	2613
vote_count	70.33	69	85.52	83	7839	2613

$$Z = \frac{\text{mean\_train} - \text{mean\_test}}{\sqrt{\frac{\text{std}^2_{\text{train}}}{n_{\text{train}}} + \frac{\text{std}^2_{\text{test}}}{n_{\text{test}}}}}$$

$$Z_{\text{popularity}} = \frac{224.5 - 220}{\sqrt{\frac{(338.6)^2}{7839} + \frac{(330)^2}{2613}}}$$

$$= \frac{224.5 - 220}{7.5}$$

$$= \frac{4.5}{7.5}$$

$$Z_{\text{popularity}} = 0.6$$

Z-score for popularity is 0.6

Teacher's Sign.: _____

Name:Vedant Dhoke Div-D15C Roll No-9 A.Y.-2024-25

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-Vedant Dhoke Div:D15C Roll No-9 A.Y.-2024-25

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-Vedant Suresh Dhoke Div-D15C Roll No-9 A.Y.-2024-25

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 above dataset.

Dataset Desciption:
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.

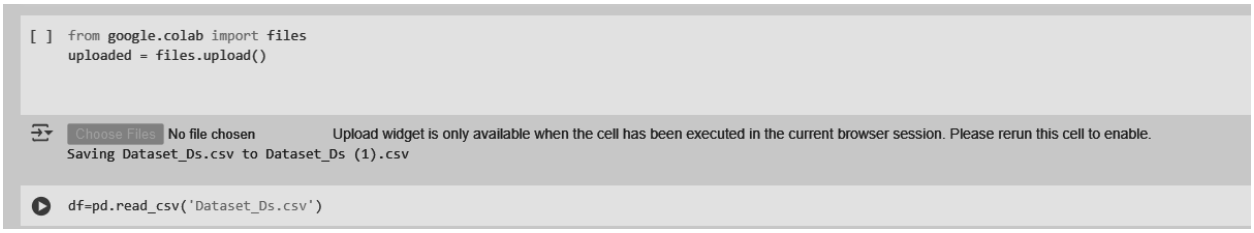
1-

```
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,

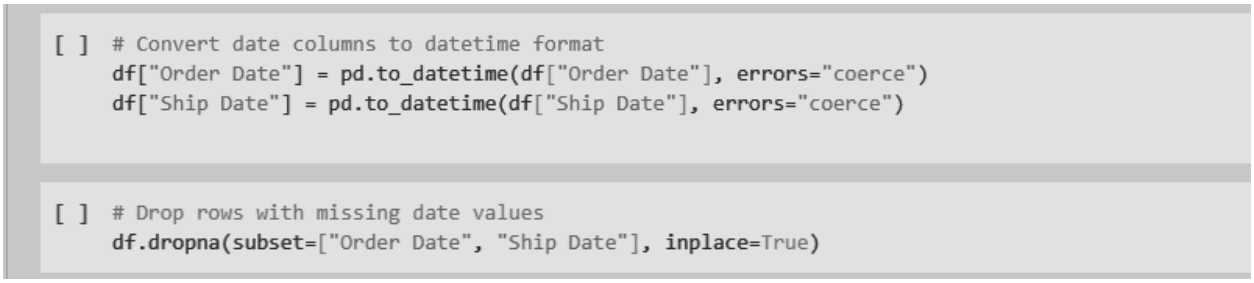
LogisticRegression), and evaluation (mean_squared_error). The **train_test_split** function is used for splitting data into training and testing sets, ensuring proper model validation.

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.

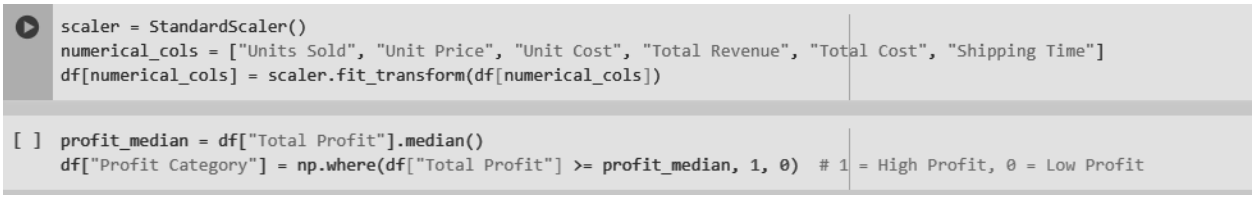
5-

```
# 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.

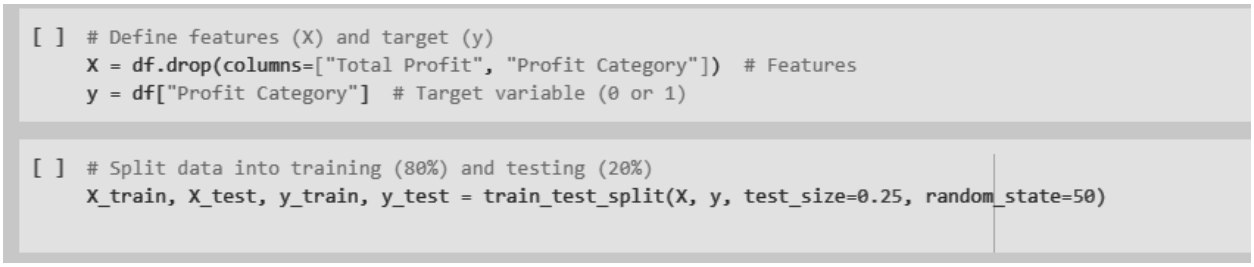
6-



```
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.

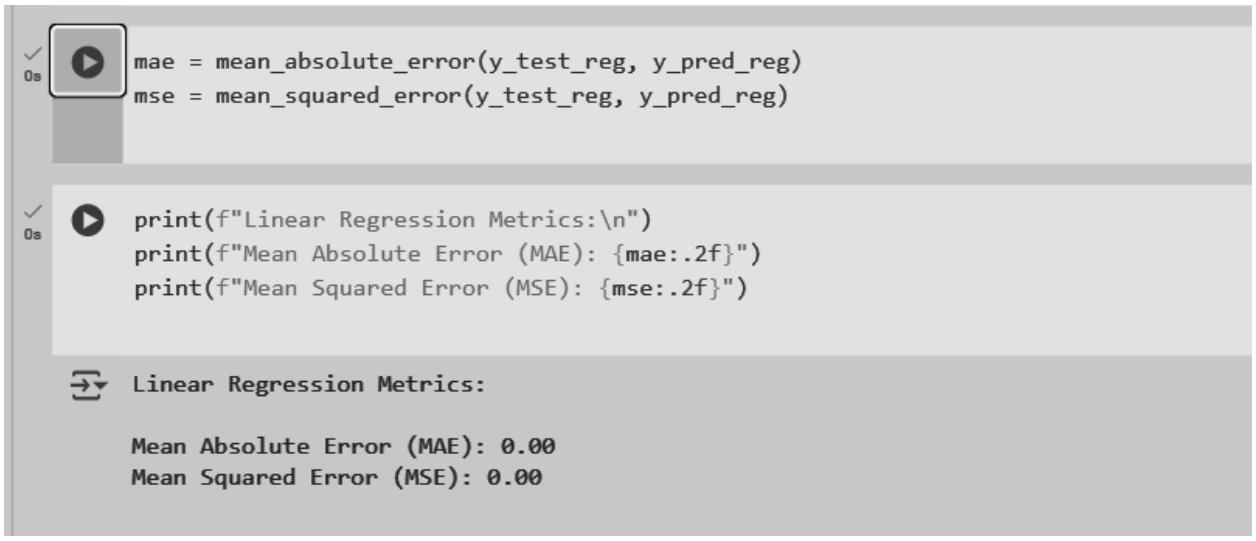
13-

```
[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 : Vedant Dhoke **Roll No : 9** **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 :

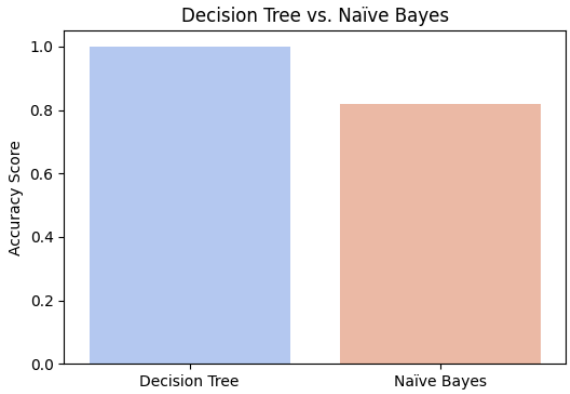
- **False Positives (877):** Some low-profit orders were incorrectly classified as high-profit.
- **False Negatives (3,614):** A significant number of high-profit orders were misclassified as low-profit, reducing recall.
- **Overall Performance:** Naïve Bayes struggled to distinguish high-profit orders, 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-Vedant Dhoke Div-D15C Roll No-9 A.Y.-2024-25

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.

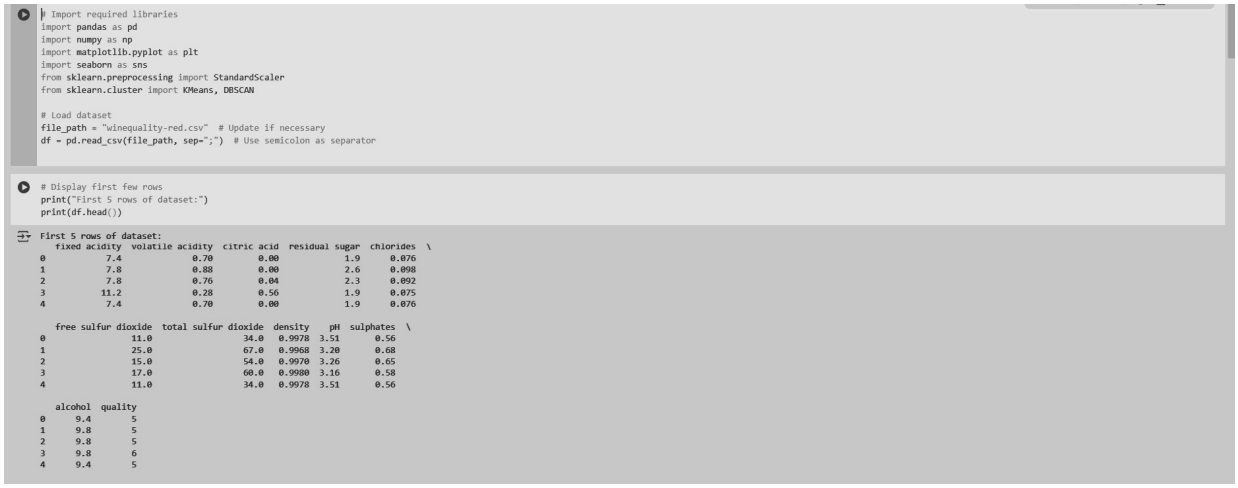
Advantages

- 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-

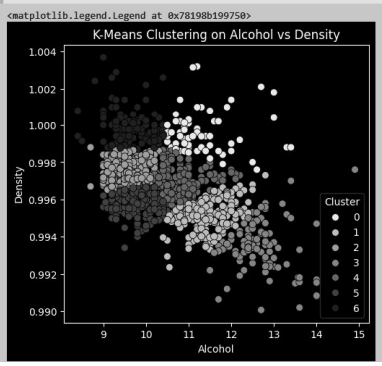
```
[ ]  # Apply K-Means clustering
     kmeans = KMeans(n_clusters=7, random_state=42, n_init=10)
     df["KMeans_Cluster"] = kmeans.fit_predict(data_scaled)

[ ]  # 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>
```

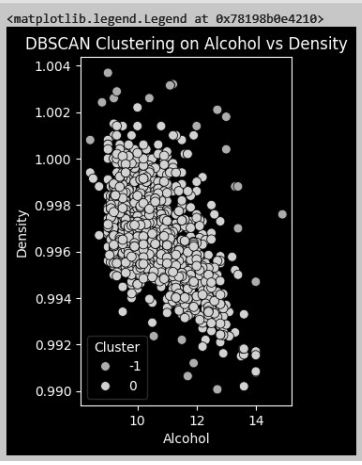


The image shows a scatter plot depicting the results of K-Means clustering applied to a dataset with "Alcohol" on the x-axis and "Density" on the y-axis. Each point represents a data sample, color-coded according to its assigned cluster using the "viridis" color palette. The clustering algorithm has 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 : Vedant Dhoke **Roll No : 09** **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.

```python
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-Vedant Dhoke Div-D15C Roll No-9
A.Y.-2024-25

EXP 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-Vedant Dhoke Div-D15C Roll No-9 A.Y.-2024-25

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.

Chapter 1

Introduction

1.1 Introduction

In today's digital age, mobile applications play a crucial role in various aspects of life, such as communication, entertainment, productivity, and more. With over 3 million apps on the Google Play Store, users often face difficulties in selecting the most suitable and high-quality apps. App ratings are instrumental in helping users make informed download decisions, with these ratings influenced by factors such as user interface, performance, features, and reliability. However, predicting an app's potential rating based on its characteristics remains a challenge for developers. Factors like the app's category, size, number of installs, and user feedback significantly impact its rating. This project aims to address this challenge by leveraging machine learning to create a predictive model that forecasts app ratings using historical data from the Google Play Store. The objective is to provide developers and businesses with data-driven insights, which can enhance design decisions, improve app performance, and ultimately lead to higher user satisfaction.

1.2 Objectives

To train a regression or classification model on the Play Store dataset, machine learning techniques are employed to predict app ratings. Key features such as app category, reviews, installs, price, and size are analyzed to understand their impact on ratings. Feature engineering is applied to extract meaningful information from raw data, while hyperparameter tuning ensures the model performs optimally. Ensemble learning techniques, such as Random Forest or Gradient Boosting, are used to improve prediction accuracy by combining multiple models. Additionally, model interpretability is a priority, as identifying the most influential factors affecting app ratings helps developers make informed decisions. By refining the model through iterative improvements and data analysis, developers can predict app ratings with greater accuracy and actionable insights.

1.3 Motivation

The motivation behind this project stems from the overwhelming number of mobile applications available in the market, with millions of apps competing for users' attention.

With so many options, users often rely on app ratings to make decisions, but these ratings can be influenced by numerous factors that are not immediately apparent. For developers, predicting an app's rating based on its characteristics such as category, reviews, installs, and size remains a complex challenge.This project is motivated by the need to create a machine learning model that can accurately predict app ratings, offering valuable insights into app performance. By leveraging historical data from the Google Play Store, the goal is to equip developers with a data-driven tool to improve app design, optimize user experience, and ultimately increase app success. Furthermore, understanding the key features that impact app ratings will not only help developers improve their offerings but also enhance the overall user satisfaction and trust in the app marketplace.

1.4 Scope of the Work

The scope of this project involves building a machine learning model to predict app ratings based on various app characteristics using data from the Google Play Store. It covers analyzing key features such as app category, reviews, installs, price, and size, to identify their impact on ratings. The project also includes optimizing model performance through techniques like feature engineering, hyperparameter tuning, and ensemble learning. Additionally, the scope extends to ensuring model interpretability by highlighting the most influential factors affecting app ratings. By providing developers with data-driven insights, the project aims to enhance app design and user experience. The outcome of this project will support decision-making for developers, businesses, and marketers seeking to improve app quality. Finally, the project's findings can be applied to the broader mobile app market, providing actionable recommendations to enhance app success.

1.5. Feasibility Study

The feasibility of this project is high, given the availability of comprehensive datasets from the Google Play Store, which provide essential features like app category, reviews, installs, price, and size. Machine learning techniques are well-established for regression and classification tasks, making it technically feasible to predict app ratings accurately. Tools like Python, with libraries such as scikit-learn, pandas, and TensorFlow, offer powerful frameworks for model building and optimization. The project also benefits from clear objectives, ensuring focused development efforts. While challenges may arise in handling missing data or outliers, these can be addressed through data preprocessing and feature engineering techniques. The project is feasible within a reasonable timeframe and can provide valuable insights for developers. Furthermore, it can be scaled or adapted to various other app marketplaces beyond the Play Store, enhancing its long-term applicability.

Chapter 6

Conclusion

6.1. Conclusion

The rapid expansion of the mobile app ecosystem has made it increasingly challenging for developers to gain visibility and for users to identify quality applications. In this project, we addressed this challenge by leveraging machine learning algorithms to predict app ratings based on structured data from the Google Play Store. Our analysis demonstrated that features such as category, number of installs, size, reviews, and price significantly influence an app's rating, and that these attributes can be effectively modeled using regression-based techniques.

Through systematic preprocessing, encoding, scaling, and training of predictive models, we developed a reliable system that provides accurate predictions of app ratings. This not only aids in assessing current applications but also helps developers estimate the potential rating of new apps under development. The results validate the potential of data-driven approaches in guiding strategic decisions related to app design, feature selection, and marketing.

6.2. Future Scope

While the current model lays a strong foundation for predicting app ratings based on structured app metadata, there are multiple opportunities for extending and enhancing the work:

1. **Incorporating User Review Text Analysis:** Future iterations can integrate Natural Language Processing (NLP) to analyze review content, sentiment, and keywords. This would enrich the model with qualitative data, offering a deeper understanding of user perceptions and concerns.
2. **Real-Time Prediction and Monitoring:** Deploying the model as a real-time dashboard or API service could allow developers to monitor changes in app ratings as features or user engagement evolves, enabling quicker adjustments and performance optimization.

3. **Deep Learning Techniques:** Implementing deep learning models such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM), or Transformers can potentially improve prediction accuracy, especially in scenarios involving large and unstructured datasets.
4. **Explainable AI:** Adding interpretability to the model can help developers understand why certain features lead to higher or lower predicted ratings, making the system more transparent and actionable.
5. **Broader Feature Integration**: Future studies can incorporate additional features such as in-app purchase data, update frequency, crash reports, or user demographics to further enhance predictive capability.

6.3. Societal Impact

The implementation of intelligent systems for predicting app ratings offers considerable societal benefits, especially in a world increasingly dependent on mobile applications for day-to-day activities. By enabling developers to make data-informed decisions, this project helps promote the creation of high-quality, user-centered applications. This leads to improved user satisfaction, reduced app abandonment rates, and overall better digital experiences.

For users, such predictive models assist in identifying reliable and relevant apps more efficiently, reducing the risk of downloading low-quality or insecure software. In turn, this contributes to building trust within the app ecosystem, especially for vulnerable or less tech-savvy users who rely heavily on ratings for decision-making.

From a business and economic standpoint, enhanced app quality translates to better market performance, higher user retention, and more efficient use of development resources. On a broader scale, these tools empower startups, individual developers, and small businesses to compete more effectively by providing them with access to predictive insights that were previously limited to larger organizations with advanced analytics capabilities.

Lastly, this project aligns with the larger vision of responsible AI — using machine learning not just for automation or prediction, but for solving real-world problems in ways that are transparent, fair, and beneficial to society at large.

Name → Vedant Suresh Dhoke
DISC Rollno → 9

05/05

AIDS Assignment - 1

Q.1] What is AI? Considering the Covid-19 pandemic situation, how AI helped to survive and renovated our way of life with different applications?

→ AI (Artificial intelligence) refers to the simulation of human intelligence in machines, enabling them to learn, reason and solve problems. AI encompasses machine learning, natural language processing, robotics and expert systems.

AI's role in the COVID-19 pandemic:

1] Healthcare → AI powered tools helped in diagnosing COVID-19 using chest X-rays, predicting outbreaks and accelerating vaccine development

2] Chatbots → AI drive chatbots provided real-time information and assisted in symptom checking.

3] Supply Chain Management → AI optimized supply chains by predicting shortages of medical equipment.

4] Drug discovery → AI assisted in analyzing protein structure to identify potential treatments

Teacher's Sign.: _____

Q.2] What are AI agents terminology, explain with example?

→ An AI agents perceives its environment using sensors and acts upon it using actuators.

Key terminologies:

1] Agent → An entity that senses and acts. Ex:- a self driving car.

2] Sensors → Devices used to perceive the environment (ex:- cameras).

3] Actuators → Components responsible for actions (eg:- robot arms in automation).

4] Rationality → An agent acts rationally to achieve its goal.

5] Performance measure → Criteria to evaluate agent performance.

Ex:- A robot vacuum cleaner uses sensors to detect obstacles and moves using actuators based on programmed logic.

Q.3] How AI technique is used to solve 8 puzzle problem?

→ The 8-puzzle consists of a 3x3 grid with numbered tiles and an empty space and the goal is to arrange the tiles in order.

AI techniques used:-

Teacher's Sign.:

1) Breadth-First Search (BFS) → Explore All possible moves at each step.

2) Depth-First search (DFS) → Explores a single path deeply before backtracking.

3) A* Algorithm → Uses heuristics to find the shortest path.

4) Best-First Search → Select moves that bring it closest to the goal state.

AI ensures the problem is solved optimally using search & heuristic based approach.

Q4) ⇒ What is PEAS descriptor? Give PEAS for the following
PEAS is used to describe and analyze the task environment of an AI agent.

System	Measure	Environment	Actuators	Sensors
Taxi drives	Safety time, customer satisfaction	Roads, traffic passengers	Steering, breaking, acceleration	GPS, speedometer, camera
Medical diagnosis	Accuracy of diagnosis speed	Patient data, symptoms	Display recommendations	Patient records, medical tests
Music Composer	Quality of composition popularity	Music industry, users	Sound generation, editing tools	Musical notes, user preferences.

Teacher's Sign.: _____

Environment	Actuators	Sensors	Performance measure

System	Performance measure	Environment	Actuators	Sensors
Aircraft Autolander	Smooth landing, safety	Airport, weather	Display feedback	Text analysis grammar
Essay Evaluator	Grammar accuracy, relevance	Essays writing styles	Aiming, filing mechanism	Motion sensors, infrared cameras

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

→

Dimension	Categorization
Observability	Partially observable (it doesn't know all book stocks instantly)
Determinism	Stochastic (stock availability changes unpredictably)
Episodic vs sequential	Sequential (it learns from past user interactions)
Static vs Dynamic	Dynamic (book availability and price changes)

Teacher's Sign.: _____

Discrete vs Continuous	Discrete (fixed set of book categories & user queries)
Single vs Multi-Agent	Multiagent (interacts with customers and inventory databases)

Q.6 Differentiate model based and utility based?

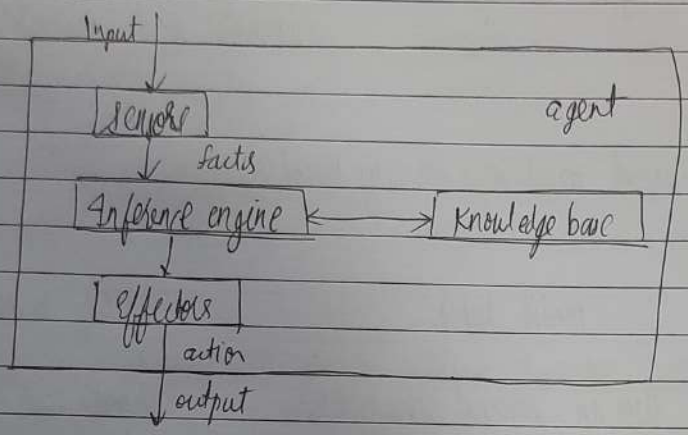
Feature	model based	Utility based
Definition	Uses an internal representation of the environment to make decisions	Chooses actions based on a utility based functions that maximizes the best possible outcome
Decision making	Relies on stored knowledge about how the environment behaves	Selects actions based on numerical values assigned to outcomes
Complexity	moderate, as it requires a well-defined model of the environment	High, as it involves calculating and comparing multiple possible outcomes.
Adaptability	Less adaptive, as it depends on pref defined rules.	more adaptive, as it evaluates outcomes dynamically.
Example	A chess playing AI tracking opponent moves based on past strategies	A self driving car choosing the safest and fastest route.

Teacher's Sign.: _____

Q.7] Explain the architecture of a knowledge based agent & learning Agent

→

• Knowledge based Agent



A knowledge based agent includes a knowledge base and an inference engine system. A knowledge base is a set of representations of facts of the world.

The agent operates as follows:-

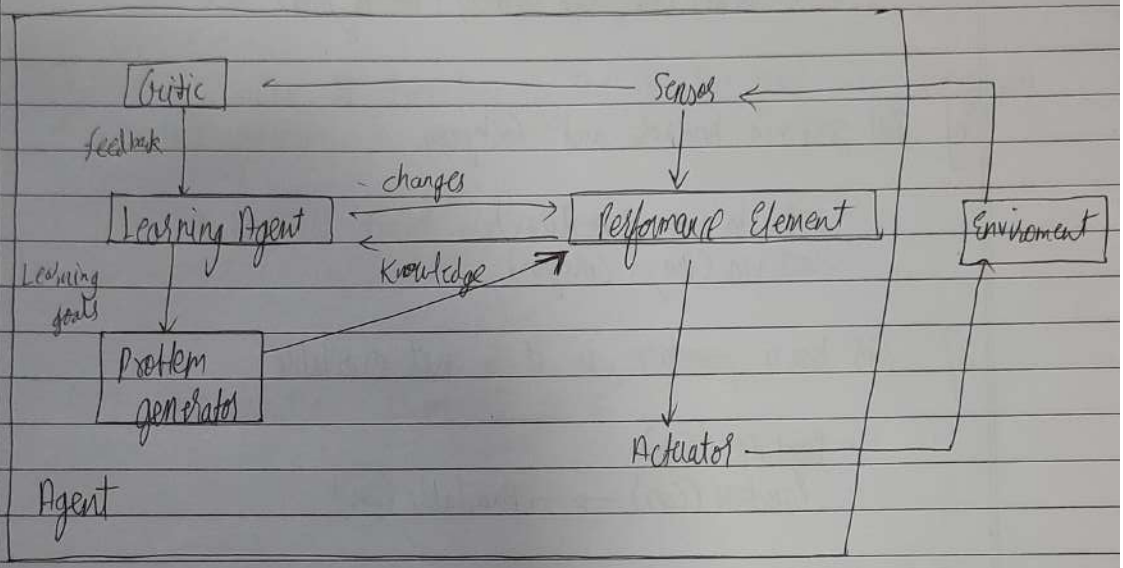
1) It TELLS the knowledge base what it perceives.
2) It ASKs the knowledge base what action it should perform.
3) It performs the chosen action.

• Learning Agent

By actively exploring and experimenting with their environment, the most powerful agents are able to learn.

Teacher's Sign.: _____

A agents can be further divided into 4 conceptual components



Q.8 Convert the following to predicates

→

Lets define predicates

- Travels(x,y) → x travels by y
- Available(y) → y is available
- Goes_via(y,z) → Vehicle y goes via location z.
- Puncture(y) → y has a puncture
- Not(Available(y)) → y is not available

Teacher's Sign.: _____

a] Anita travels by car if available, otherwise by bus

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

b] Bus goes via Andheri and Goregaon

- Goes_via (Bus, Andheri)
- Goes_via (Bus, Goregaon)

c] Car has a puncture, so it is not available

- Puncture (car)
- Puncture (car) → ¬ Available (car)

- Forward Reasoning to determine Anita's Travel Route

1] Given Puncture (car), we apply the rule Punctured (car) → ¬ Available (car), so:-

- ¬ Available (car) (car is not available)

2] From ¬ Available (car) → Travels (Anita, Bus), we infer:

- Travels (Anita, Bus) (Anita travels by Bus)

Teacher's Sign.:

3] Given Goes_via (Bus, Goregaon), we know

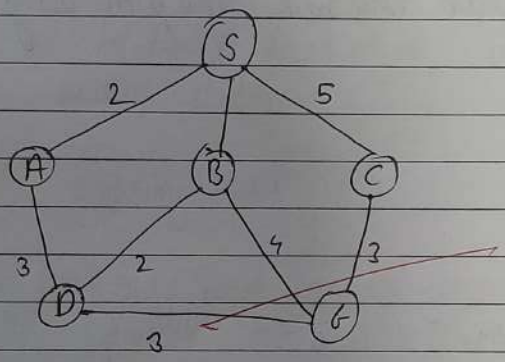
- ~~The buses pass~~
- The bus passes through goregaon

4] Since Anita is travelling by Bus and Bus goes via Goregaon we will conclude:

- Anita will travel via Goregaon

~~Q9~~

Q10] Find the route from S to G



BFS explores the graph level by level, starting from the source nodes

Step 1:

1] Start at S: Add S to the queue

- Queue: [S]

Teacher's Sign.:

- Expand S: Visit its neighbors A, B, C
  - 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 the parent mapping.

~~Step 3:~~

Step 2:

The ~~shortest~~ path from S to G is
S → B → G

Teacher's Sign.: _____

Q.11 What do you mean by depth limited search? Explain Iterative deepening search with example.

→

- Depth limited search (DLS)

It is a variant of depth first search that restricts the depth of recursion to fixed limit L. It avoids infinite loop search spaces.

Algorithm:-

1) start from the initial node and explore deeper until depth L is reached
2) If goal is found, return the solution
3) If the depth limit is reached without finding the goal, backtrack.
4) If no solution is found at all, increase the depth limit or choose another strategy

- Iterative Deepening Search (IDS)

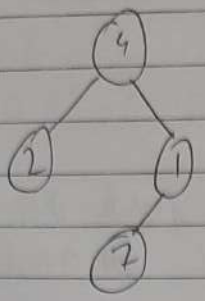
Iterative Deepening Search combines the advantages of both BFS and DFS. It repeatedly runs Depth Limited Search with increasing depth limits until the goal is found.

Algorithm:

1) Set depth limit L=0
2) Run depth-Limited Search (DLS) up to depth L.
3) If the goal is not found, increase L and repeat DLS
4) Continue until the goal is found or all nodes are explored.

Teacher's Sign.:

Example :- Consider tree we need to find 7 at depth 2



For L=0, 4 → goal not found search further by increasing depth
For L=1 2, 1 → goal not found search further by increasing depth.
For L=2 7 → goal found.

12] Explain Hill climbing and its drawbacks in detail with example. Also state limitations.

→ Hill climbing is a local search algorithm that continuously moves toward higher-valued states in search space to find the optimal solution. It is widely used in optimization problems.

Algorithm :-

1) Start from an initial state
2) Evaluate all neighbouring states.
3) Move to the neighbour with highest value
4) Repeat until no better neighbour is found

Teacher's Sign.: _____

- Drawbacks of Hill climbing:

i) Local Maxima → It can get stuck at a peak that is not global maximum

ii) Plateau Problem → If all neighbours have the same value, the algorithm may stop

- Limitations of Steepest-Ascent Hill climbing

i) Evaluates all neighbors, which may be computationally expensive
ii) Can get stuck in local maxima instead of finding the global optimum
iii) Not useful for complex, non-smooth search spaces.

Q.B] Explain simulated annealing and write it's algorithm.

→

Simulated Annealing is an optimization technique inspired by the cooling process of metals. Unlike Hill climbing it can escape local maxima by allowing some "bad" moves with a probability that decreases over time

Algorithm:-

1] Start with an initial state S and high temperature T.

2] Generate a random neighbour S'.

Teacher's Sign.:

3] If S' is better than S, move to S'.

4] If S' is worse, accept it with probability $e^{(-\Delta E/T)}$, where ΔE is difference in values.

5] Reduce T gradually

6] Repeat until $T \to 0$ or a solution is found

Example:- In Travelling Salesman Problem (TSP) to optimize the shortest route while avoiding local optima.

Q.14] Explain A* algorithm with an example.

→ A* is a search algorithm used in pathfinding and graph traversal. It finds the shortest path from the start node to the goal node using both:

- $g(n)$ → the actual cost from the start node to node n.
- $h(n)$ → the estimated cost from node n to goal.
- $f(n) = g(n) + h(n)$ → The total estimated cost of the cheapest path through n

Algorithm:-

1] Place the start node in an open list
2] move the node with the lowest $f(n)$ from the open list to the closed list.

Teacher's Sign.:

3) Expand the node's neighbors, update their $g(n)$ & $h(n)$ values

4) Repeat until the goal node is reached.

- Advantages →

i) Guarantees the shortest path if $h(n)$ is admissible
ii) More efficient than Dijkstra's algorithm in many cases

Q.15) Explain Minmax. Explain Minimax Algorithm & draw game tree for TIC TAC TOE.

→ Minimax is an adversarial search algorithm used in zero sum games like Tic-Tac-Toe, chess and checkers. It assumes two players:

- Maximizer (tries to maximize score)
- Minimizer (tries to minimize score)

- Algorithms steps:-

1) Generate the game tree upto depth where a terminal state is reached.

2) Assign scores:
   - +1 for a win, -1 for a loss, 0 for a draw

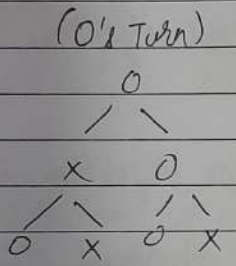
Teacher's Sign.: _____

3] Propagate the scores back up the tree:

- Maximizer chooses the max value
- Minimizer chooses the min value

4] The root node selects the best move for the player.

Example:- Tic-Tac-Toe Game tree



- The algorithm evaluates each outcome and select the best move for the current player.

Q.16] Explain Alpha beta pruning algorithm for adversarial search with example.

→ Alpha beta pruning improves Minimax by pruning unnecessary branches, reducing computation time without affecting the final result.

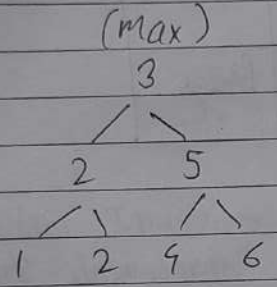
- Alpha (α): Best choice for the maximizer
- Beta (β): Best choice for the minimizer

Teacher's Sign.: _____

Algorithm Steps:-

1) Traverse the game like minimax
2) If a branch's worst case is worse than an already explored branch, prune it.
3) Continue evaluating only necessary branches.

Example: Pruning in Tic Tac Toe



- If left subtree evaluates to 2, and we know the right subtree has a move ≥ 3, we prune the left subtree.

Advantages:-

- Reduces computation from $O(b^d)$ to $O(b^{d/2})$
- Faster decision making in real time games like chess.

Teacher's Sign.:

Q.17) Explain WUMPUS world environment giving it's PEAS description. Explain how percept sequence is generated.

→

Stench		Breeze	Pit
Wumpus	Breeze Stench Gold	Pit	Breeze
Stench		Breeze	
start (Agent)	Breeze	Pit	Breeze

The WUMPUS world is a grid based environment used to study AI agent behavior in uncertain environments. The agent navigates a cave, avoiding dangers and trying out to find gold.

Environment Details:

- The grid consists of rooms connected by doors (4x4)
- The Wumpus is in one of the rooms
- There are pits which the agent must avoid.
- The agent can sense danger using percepts

i) Stench → Wumpus nearby
ii) Breeze → Pit is nearby
iii) Glitter → Gold is present

Teacher's Sign.: _____

PEAS Description for Wumpus world

• Performance measure	+1000 if agent comes out with the gold -1000 if Wumpus eats the agent +1 for each action
• Environment	A 4x4 grid world with pits, wumpus, gold and walls.
• Actuators	move forward, turn left, turn right, ~~grab~~ grab (gold), shoot (arrow), climb (exit)
• Sensors	Perceive stench (wumpus), breeze (pit), glitter (gold), bump (wall) and scream (wumpus killed)

Percept sequence

At each step, the agent receives a 5-symbol percept sequence based on the current room.
For example:-

- If agent is in a room next to a pit and gold, the percept is:- [Breeze, None, Glitter, None, None]
- The agent updates its knowledge base & decides on the next move.

Teacher's Sign.: _____

Q.18) Solve the following Crypto-arithmetic problems

```
  SEND
  MORE
 MONEY
```

→

C_4	C_3	C_2	C_1	
	S 9	E 5	N 6	D 7
	M 1	O 0	R 8	E 5
M 1	O 0	N 6	E	Y 2

As M is leftmost digit, $M = 1$ because it will get carry value

- $S + M = O$
  $S + 1 = O$ we know $M = 1$ and we want $S+1 \geq 10$, as we want carry to be generated
  $\therefore S = 9$, $O = 0$

- $E + O = N$
  $E + 0 = N$ .. $10 \rightarrow 0$ (zero)
  $E = N$ .. {If $C_2 = 0$, $E = N$ it is violating condition there should be a carry}

Let assume $E = 5$

$\therefore E + O + C_2 = N$

$\boxed{N = 6}$

Teacher's Sign.: _____

- $N + R = E$

  $6 + R = 5$ $\quad (N = 6, E = 5)$

If we take $R = 9$, then it will satisfy but already we have $S = 9$ so it's not possible

Now we want a carry i.e $C_1 = 1$

$6 + R + 1 = 15$

$\therefore R = 8$

- $D + E = Y$

  $D + 5 = Y$ -- This equation will generate a carry C_1

  $\therefore D \geq 5$ we left with only 7

If $D = 7$

$7 + 5 = 12$

So $D = 7$ & $Y = 2$ with carry

Then it matches

Teacher's Sign.: _____

Q.19] Consider the following axioms

All people who are ~~happy~~ graduating are happy
All happy people are smiling
Someone is graduating

Also explain the following

→

Given axioms

1] All people who are graduating are happy → $\forall x$ Graduating(x) → Happy(x)

2] All happy people are smiling → $\forall x$ Happy(x) → Smiling(x)

3] Someone is graduating → $\exists x$ Graduating(x)

Explain :-

Step 1 :- Convert to Clause Form

1] Graduating(x) → Happy(x) → ¬ Graduating(x) ∨ Happy(x)

2] Happy(x) → Smiling(x) → ¬ Happy(x) ∨ Smiling(x)

3] $\exists x$ Graduating(x) → Introduce constant 'a' → Graduating(a).

Teacher's Sign.: _____

Step 2: Prove "Is someone smiling?"

1] Graduating (a) -- Given

2] ¬ Graduating (x) ∨ Happy (x)

- Substitute x=a: ¬ Graduating (a) ∨ Happy (a)
- Since Graduating (a) is true, resolve to Happy(a)

3] ¬ Happy(x) ∨ Smiling (x)

- Substituting x = a: ¬ Happy(a) ∨ Smiling (a)
- Since Happy(a) is true, resolve to Smiling (a)

Thus, someone is Smiling (Smiling (a)).

Resolution tree for proof

Graduating(a)

|— (Graduating (a) ∨ Happy(a)) → Resolve → Happy(a)

|— (¬Happy (a) ∨ Smiling (a)) → Resolve → Smiling(a)

Teacher's Sign.: _____

Q.20] Explain Modus Ponen with suitable example.

→

Modus Ponen is a fundamental rule of inference in logic that states:

If "P → Q" (If P then Q) is true and P is true, then Q must be true.

It follows this logical form

1] P → Q (If P then Q)
2] P (P is true)
∴ Q (Therefore, Q is true)

- Example:
  - If it rains, the ground will be wet (P → Q)
  - It is raining. (P is true)
  - Therefore, the ground is wet. (Q is true)

- Symbolic Representation:

1] Rain → Wet Ground
2] Rain
∴ Wet Ground

Teacher's Sign.: _____

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

→

- Forward chaining

Starts from known facts and applies inferance rules to ~~reach~~ reach a conclusion.
It is also called as data driven approach.

Algorithm

1) Start with given facts in a knowledge base
2) Apply rules to infer new facts.
3) Repeat until the goal/conclusion is ~~reached~~ reached.

Example :-

Rule:-
1) If a person has a fever and cough, then may have the flu.
2) If a person has a sore throat, they may have a cold

Facts :
1) John has a fever and cough
2) Apply rule1 → John may have the flu.

Teacher's Sign.: _____

Conclusion → The system infers John has the flu.

- Backward chaining → Starts from the goal & works backward to check if facts support it.
  It is also called "goal-driven approach"

Algorithm:

1) Start with the goal.
2) check if there are rules to support it.
3) See if known facts confirm those rules.
4) Repeat until a match is found or disproved.

Example:-

Goal: Does John have the flu?

- Flu :→ Fever and cough (Rule)
- Does John have fever and cough? (Check facts)
- Yes, John has a fever and cough

Conclusion → John has the flu.

Teacher's Sign.: _____

NAME: VEDANT DHOKE CLASS/ROLL NO: D15C/ 9

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

Formula:
Mean =Sum of all data points / Total number of data points

Calculation:
Sum = 82+66+70+ 59+ 90+ 78+ 76+ 95+ 99+ 84+ 88+ 76+ 82+ 81+ 91+ 64+ 79+ 76+ 85+ 90
Sum = 1621
Mean = 1621/20 = 81.05

Mean = 81.05

2. Median

Steps:
First, arrange the data in ascending order:
59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99
Since the number of values (n = 20) is even, the median is the average of the 10th and 11th terms.

Median = 81 + 82 / 2 = 163/2= 81.5
Median = 81.5

3. Mode

Steps:

Look for the number(s) that occur most frequently.

From the sorted list:
76 occurs 3 times, while all other numbers occur once or twice.

Mode = 76

4. Interquartile Range

Steps:
The Interquartile Range (IQR) is defined as: IQR=Q3−Q1

From the ordered dataset:
59, 64, 66, 70, 76, 76, 76, 78, 79, 81, 82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Q1 (First Quartile):
Lower half (first 10 numbers):
59, 64, 66, 70, 76, 76, 76, 78, 79, 81

Q1 is the median of the first half = average of 5th and 6th values:
Q1 = 76+76 / 2 =76

Q3 (Third Quartile):
Upper half (last 10 numbers):
82, 82, 84, 85, 88, 90, 90, 91, 95, 99

Q3 is the median of the second half = average of 5th and 6th values:
Q3= 88+90 / 2 = 89

IQR Calculation:

IQR=89−76=13
Interquartile Range = 13

Q.2: Comparison and Analysis of Machine Learning for Kids and Teachable Machine
Ans ->
1. For each tool listed below:

A. Machine Learning for Kids

- Target Audience: Primarily designed for school-aged students, beginners, and educators who want to introduce AI and machine learning concepts in an accessible, visual, and engaging way.
- Use by Target Audience: Students use the platform to train machine learning models using examples (text, numbers, images, or sound) and apply those models to interactive projects using Scratch or Python. Teachers utilize it as a classroom teaching tool to demonstrate AI concepts with hands-on projects.
- Benefits: Intuitive, beginner-friendly interface.
  Encourages experiential learning through integration with Scratch.
  Enables students to create real AI models using their own data.
  Free and web-based — no installation required.
- Drawbacks: Limited to basic functionalities — not suitable for advanced machine learning. Less control over model parameters or algorithms. Requires internet connection for use.

B. Teachable Machine

- Target Audience: Targeted towards students, educators, hobbyists, and non-programmers interested in quickly building and testing machine learning models, especially for image, audio, and pose detection.
- Use by Target Audience: Users can create classification models by uploading their own data (images, audio, or pose samples), train models quickly, and export them for use in websites, apps, or physical devices.
- Benefits: No coding required.
  Allows real-time training and testing of models.
  Provides export options to TensorFlow.js and TensorFlow Lite for deployment.
  Highly engaging and interactive interface.
- Drawbacks: Not suitable for complex or large-scale projects. Limited model customization. May not perform well with small or unbalanced datasets.

2. Classification of Tools: Predictive or Descriptive Analytics

Tool	Classification	Reason
Machine Learning for Kids	Predictive Analytic	It uses training data to build models that predict outcomes based on new input (e.g., classify images or text).

Teachable Machine	Predictive Analytic	It allows users to train models that predict or classify new inputs, such as recognizing gestures, faces, or sounds.

3. Classification of Tools: Type of Machine Learning

Tool	Learning Type	Reason
Machine Learning for Kids	Supervised Learning	Users train models using labeled data, where the input data is tagged with the correct output.
Teachable Machine	Supervised Learning	It also uses labeled data (e.g., tagging images with class names) to train models for classification.

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 ->

1. **Key Insights from Assigned Readings**

a. "What’s in a Chart? A Step-by-Step Guide to Identifying Misinformation in Data Visualization" by Arthur Kakande.

Arthur Kakande emphasizes the importance of critically evaluating data visualizations to detect misinformation. Key takeaways include:

- Assessing Data Sources: Verify the credibility and origin of the data presented.
- Examining Axes and Scales: Check for manipulated scales that can exaggerate or downplay trends.
- Identifying Omissions: Be alert to missing data or context that could alter interpretation.
- Recognizing Misleading Chart Types: Some chart types, like pie charts, can be easily misinterpreted if not used appropriately.

By applying these principles, readers can better discern the accuracy and intent behind data visualizations.

b. "How Bad Covid-19 Data Visualizations Mislead the Public" by Katherine Ellen Foley
Katherine Ellen Foley discusses specific examples from the COVID-19 pandemic where data visualizations misled the public:

- Snapshot Data Without Trends: Presenting single-day data without historical context can misrepresent the severity or improvement of situations.
- Overuse of Pie Charts: Complex data shown in pie charts can confuse readers, especially when depicting parts of a whole that don't sum intuitively.
- Missing Axes and Labels: Omitting axes or labels deprives viewers of necessary reference points, leading to potential misinterpretation.

These examples underscore the necessity for clarity, context, and completeness in data visualizations to ensure accurate public understanding.

2. Case Study: Misleading Data Visualization in COVID-19 Vaccine Efficacy Reporting.

Incident Overview

In March 2024, social media platforms circulated claims suggesting that COVID-19 vaccines were causing more harm than good. These assertions were based on data showing higher absolute numbers of deaths among vaccinated individuals compared to unvaccinated ones between July 2021 and May 2023. The data was presented in bar charts highlighting the total number of deaths in each group without considering the proportion of vaccinated versus unvaccinated individuals in the population.

Analysis of the Misleading Visualization

- Lack of Proportional Context: The visualizations displayed raw death counts without accounting for the fact that a significantly larger portion of the population was vaccinated.

Without normalizing the data (e.g., deaths per 100,000 individuals), the charts misleadingly suggested that vaccination increased mortality.

- Omission of Confounding Variables: Factors such as age, underlying health conditions, and exposure risks were not represented in the visualizations. These variables are critical in understanding mortality rates and the effectiveness of vaccines.

- Absence of Time Trends: The charts did not show how death rates changed over time in relation to vaccination rollouts, missing an opportunity to illustrate potential correlations between vaccination and mortality reductions.

Impact and Consequences

The misleading visualizations contributed to vaccine hesitancy among the public, undermining efforts to achieve widespread immunity. Public health officials had to invest additional resources to counteract the misinformation, emphasizing the importance of accurate data interpretation and presentation.

Conclusion

This case study highlights the critical importance of responsible data visualization. Misrepresenting data, whether intentionally or inadvertently, can have significant real-world consequences, particularly in public health. It is imperative for data communicators to present information transparently, ensuring that visualizations accurately reflect the underlying data and context.

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

Ans ->

Dataset Used: Pima Indians Diabetes Database

```python
# Step 1: Import required libraries
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
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from sklearn.naive_bayes import GaussianNB
from imblearn.over_sampling import SMOTE
```

- Import all the required libraries

NAME: VEDANT DHOKE CLASS/ROLL NO: D15C/ 9

```
# Step 2: Load dataset
df = pd.read_csv("diabetes.csv")  # Pima Indians Diabetes

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

```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

- In this step, the Pima Indians Diabetes dataset was loaded using pandas. The dataset contains 768 rows and 9 columns, including medical features like Glucose, BMI, Age, and the target variable Outcome. Using head() and info(), it was confirmed that there are no missing values and all columns are numeric. The dataset is clean and ready for further preprocessing and modeling.

```
# Step 3: Feature-Label separation
X = df.iloc[:, :-1]  # All columns except last
y = df.iloc[:, -1]   # Last column (Outcome)
```

+ Code + Text

```
[4] # Step 4: Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_balanced, y_balanced = smote.fit_resample(X, y)

print("Class distribution after SMOTE:\n", pd.Series(y_balanced).value_counts())
```

```
Class distribution after SMOTE:
 Outcome
1    500
0    500
Name: count, dtype: int64
```

- In this step, the dataset was split into features (X) and the target variable (y) using .iloc. To address the issue of class imbalance in the target variable Outcome, the SMOTE (Synthetic

Minority Over-sampling Technique) method was applied. After applying SMOTE, the class distribution was balanced to 500 samples for each class (0 and 1), ensuring that the model does not become biased toward the majority class during training.

```
# Step 5: Train/Validation/Test Split (70/20/10)
X_train, X_temp, y_train, y_temp = train_test_split(X_balanced, y_balanced, test_size=0.3, stratify=y_bala
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=1/3, stratify=y_temp, random_sta

print(f"Train: {len(X_train)}, Validation: {len(X_val)}, Test: {len(X_test)}")
```

```
Train: 700, Validation: 200, Test: 100
```

- In this step, the dataset was split into training, validation, and testing sets using a 70/20/10 ratio. Initially, 70% of the data was allocated to the training set, and the remaining 30% was further split into validation and test sets in a 2:1 ratio. The train_test_split function was used with stratification to preserve the class distribution. As a result, the final data was divided into 700 training samples, 200 validation samples, and 100 testing samples, ensuring a balanced and representative dataset for model training and evaluation.

```
# Step 6: Preprocessing using StandardScaler
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

```
# Step 7: Train Naive Bayes with Hyperparameter Tuning
param_grid = {
    'var_smoothing': np.logspace(-9, -5, 10)
}

model = GaussianNB()
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'var_smoothing': np.float64(1e-09)}
```

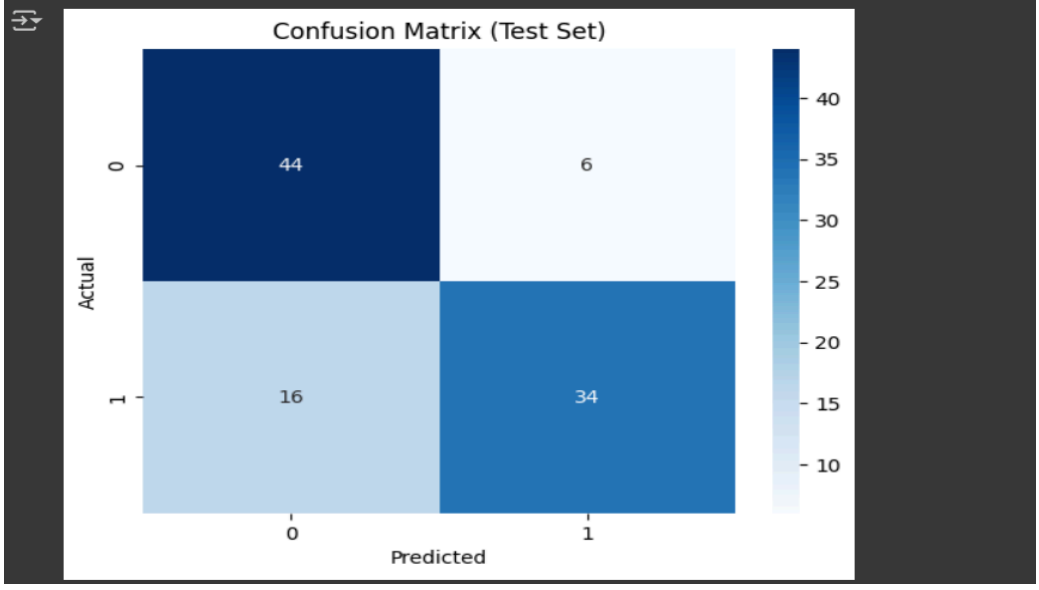
- In this step, data preprocessing was performed using StandardScaler to standardize the features, ensuring they have a mean of 0 and standard deviation of 1, which helps improve model performance. After scaling, a Naive Bayes classifier was trained using GridSearchCV to tune the var_smoothing parameter over a logarithmic range. The best parameter found was var_smoothing = 1e-09, which helps control numerical stability in the model and improves classification accuracy.

```
# Step 8: Evaluate on Validation and Test
val_preds = best_model.predict(X_val_scaled)
test_preds = best_model.predict(X_test_scaled)

print("Validation Accuracy:", accuracy_score(y_val, val_preds))
print("Test Accuracy:", accuracy_score(y_test, test_preds))

# Classification report
print("\nClassification Report (Test):")
print(classification_report(y_test, test_preds))
```

```
Validation Accuracy: 0.7
Test Accuracy: 0.78

Classification Report (Test):
              precision    recall  f1-score   support

           0       0.73      0.88      0.80        50
           1       0.85      0.68      0.76        50

    accuracy                           0.78       100
   macro avg       0.79      0.78      0.78       100
weighted avg       0.79      0.78      0.78       100
```

- In the final step, the trained Naive Bayes model was evaluated on the validation and test sets. The validation accuracy was 70%, while the test accuracy reached 78%. The classification report for the test set showed a good balance between precision and recall, with an overall F1-score of 0.78. This indicates that the model performs reasonably well on unseen data.

```
# Step 9: Confusion Matrix Visualization
cm = confusion_matrix(y_test, test_preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix (Test Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



- The confusion matrix visualization in Step 9 offers a deeper insight into the model's performance:

- True Positives (1 predicted as 1): 34
- True Negatives (0 predicted as 0): 44
- False Positives (0 predicted as 1): 6
- False Negatives (1 predicted as 0): 16

- This shows that the model correctly identified a majority of both classes but is slightly better at predicting class 0. The higher number of false negatives (16) suggests that the model sometimes struggles to detect class 1 instances correctly, which aligns with the recall score for class 1 being lower than that of class 0. Overall, the confusion matrix confirms the classification report and highlights areas for potential improvement, such as reducing false negatives.

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.

```
# 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.

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

NAME: VEDANT DHOKE CLASS/ROLL NO: D15C/ 9

Key Features of the Wine Quality Dataset

The Wine Quality Dataset contains physicochemical properties of red or white wine samples. Each row in the dataset is a different wine sample, and the target variable is the quality score (rated from 0 to 10).

Feature Name	Description
fixed acidity	Non-volatile acids that do not evaporate during fermentation.
volatile acidity	Amount of acetic acid; high levels can result in an unpleasant taste.
citric acid	Adds freshness and flavor; small quantities are usually favorable.
residual sugar	Sugar remaining after fermentation; affects sweetness.
chlorides	Salt content; affects taste and stability of wine.
free sulfur dioxide	Protects wine from microbial growth and oxidation.
total sulfur dioxide	Sum of free and bound forms; high levels can affect taste and health.
density	Affected by sugar and alcohol content; used to monitor fermentation.
pH	Indicates acidity level; affects taste and microbial stability.
sulphates	Used as an antioxidant and antimicrobial agent.
alcohol	Alcohol content of wine; typically has a strong positive correlation with quality.
quality (target)	Score assigned to the wine sample (0–10) based on sensory data.

Importance of Features in Predicting Wine Quality

- Alcohol: Positively correlated with quality. Higher alcohol usually improves flavor.
- Volatile Acidity: Negative impact; too much leads to vinegar-like taste.
- Sulphates & Citric Acid: Moderate to high correlation; help in preservation and flavor enhancement.
- pH & Chlorides: Can influence taste, but usually show lower correlation with quality.
- Residual Sugar: Important for sweetness but has lower impact unless extreme.
- Fixed Acidity, Density: Help in fermentation monitoring; moderate importance.

Feature importance can be confirmed through correlation analysis and model-based feature selection techniques (e.g., feature importance from Random Forest).

Handling Missing Data During Feature Engineering

In the original Kaggle dataset, missing values are rare or often not present. However, if missing data does exist, it can be handled in several ways:

Steps Taken:

NAME: VEDANT DHOKE CLASS/ROLL NO: D15C/ 9

1.Data Exploration:
Checked for NaN or missing values using df.isnull().sum()

2.Visual Inspection:
Used heatmaps (e.g., Seaborn) to visualize missing data patterns.

3. Applied Imputation Techniques:
For numerical features, used mean/median imputation.
For non-linear relationships, used K-Nearest Neighbors (KNN) imputation.

Advantages and Disadvantages of Imputation Techniques

Technique	Advantages	Disadvantages
Mean/Median Imputation	Simple, fast, works well with symmetric distributions	Can reduce variance; not suitable for skewed or non-normal distributions
Mode Imputation	Works well for categorical data	Not useful for continuous/numerical features
KNN Imputation	Uses data similarity; retains data variance	Computationally expensive, sensitive to outliers
Regression Imputation	Uses correlations between features to predict missing values	Can introduce bias if model is overfit or assumptions are incorrect
Multiple Imputation	Accounts for uncertainty; gives multiple datasets for better estimation	Complex and computationally intensive
Dropping Rows	Easy and clean	Can lead to data loss if many rows are missing