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

DS LAB EXP 3

AIM: Perform Data Modeling.

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

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

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

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

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

```
import pandas as pd
import numpy as np

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

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

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

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

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

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

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

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

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

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

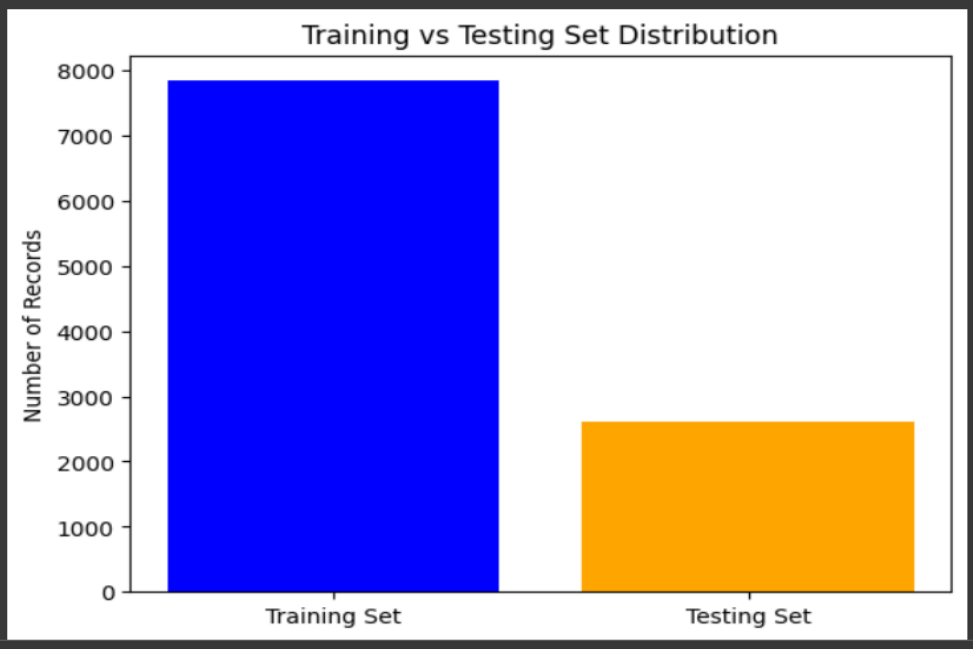
   test_size=0.25: 25% of the dataset is assigned to the test set, and the remaining 75% goes to the training set.

```
[ ] from sklearn.model_selection import train_test_split

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

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

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

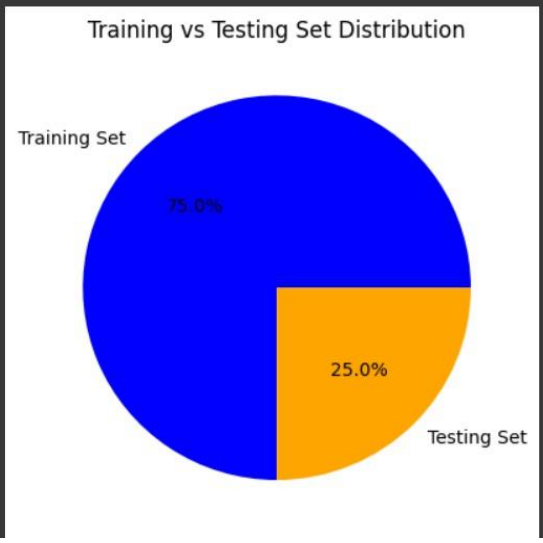
4. The bar graph visually represents the number of records in the training and testing sets using vertical bars. The height of each bar corresponds to the count of records, making it easy to compare the absolute difference between the two sets.

```
import matplotlib.pyplot as plt

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



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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
[ ] z_test_results = {}

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

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

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

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

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

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

Conclusion:

In this experiment, we have learned:

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

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

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

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