MOVIE RATING PREDICTION

This project involves building a machine learning model to predict movie ratings based on various features such as **genre**, **director**, **actors**, and other metadata. The project focuses on analyzing historical movie data, preprocessing it, engineering features, and applying regression techniques.

Objective

- To build a machine learning model that predicts the **rating of a movie** based on its features.
- Analyze the relationship between features (genre, director, actors, etc.) and ratings given by **users** or **critics**.
- Provide insights into the factors influencing movie ratings.

Data Dictionary

Name: The title of the movie.

Year: The year the movie was released.

Genre: The primary genre(s) of the movie (e.g., Action, Comedy, Drama). Multiple genres may be included.

Rating: The average rating of the movie given by critics or users (e.g., IMDb or Rotten Tomatoes rating).

Votes: The total number of votes or reviews that contributed to the movie's rating.

Director: The name of the director who directed the movie.

Actor 1: The name of the lead actor in the movie.

Actor 2: The name of the second lead actor or supporting actor.

Actor 3: The name of another key actor, often a supporting role.

Task Steps

1. Setup Environment
2. Data Exploration
3. Data Cleaning
4. Feature Engineering
5. Split Data
6. Model Building
7. Model Evaluation
8. Insights and Visualization

Python Packages

- **Pandas** (data manipulation)
- **Numpy** (numerical operations)
- **Matplotlib** (basic plotting)
- **Seaborn** (statistical visualizations)
- **scikit-learn** (for preprocessing, regression models, and evaluation)

1. Loading the Dataset and Libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_squared_log_error, r2_score
```

2. Load the dataset

```
df = pd.read_csv('IMDB Movies India.csv', encoding='latin1')

# Dataset first look
print('First few rows:\n',df.head(5))
print('\nDataset Information:\n',df.info())
```

```
First few rows:
                                 Name    Year  ...           Actor 2          Actor 3
0                                         NaN  ...            Birbal  Rajendra Bhatia
1  #Gadhvi (He thought he was Gandhi)  (2019)  ...     Vivek Ghamande    Arvind Jangid
2                         #Homecoming  (2021)  ...  Plabita Borthakur       Roy Angana
3                             #Yaaram  (2019)  ...         Ishita Raj  Siddhant Kapoor
4                 ...And Once Again  (2010)  ...  Rituparna Sengupta      Antara Mali

[5 rows x 10 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15509 entries, 0 to 15508
Data columns (total 10 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Name      15509 non-null  object
 1   Year      14981 non-null  object
 2   Duration  7240 non-null   object
 3   Genre     13632 non-null  object
```

4 Rating 7919 non-null float64

5 Votes 7920 non-null object

6 Director 14984 non-null object

7 Actor 1 13892 non-null object

8 Actor 2 13125 non-null object

9 Actor 3 12365 non-null object

dtypes: float64(1), object(9)

memory usage: 1.2+ MB

Dataset Information:

None

3. Data Cleaning

```python
# Initial DataFrame inspection
print("\nMissing Values Before Cleaning:\n",df.isnull().sum())

# Checking Duplicate Rows
print("\nDuplicate Rows Before Cleaning:\n",df.duplicated().sum())

# Drop rows with missing values
df.dropna(inplace=True)

# Recheck for missing values
print("\nMissing Values After Dropping:\n",df.isnull().sum())

# Drop duplicate rows
df.drop_duplicates(inplace=True)

# Final shape of the DataFrame
print("Final Shape of the DataFrame:", df.shape)
```

Missing Values Before Cleaning:

Name 0

Year 528

Duration 8269

Genre 1877

Rating 7590

Votes 7589

Director 525

Actor 1 1617

Actor 2 2384

Actor 3 3144

dtype: int64

Duplicate Rows Before Cleaning:
6

Missing Values After Dropping:
Name 0
Year 0
Duration 0
Genre 0
Rating 0
Votes 0
Director 0
Actor 1 0
Actor 2 0
Actor 3 0
dtype: int64
Final Shape of the DataFrame: (5659, 10)

4. Data Pre-processing

```
# replacing the brackets from year column
df['Year'] = df['Year'].str.replace(r'[()]', '', regex=True).astype(int)

# Remove the min word from 'Duration' column and convert all values to numeric
df['Duration'] = pd.to_numeric(df['Duration'].str.replace(' min', ''))

# Splitting the genre by, to keep only unique genres and replacing the null values with mode
df['Genre'] = df['Genre'].str.split(', ')
df = df.explode('Genre')
df['Genre'].fillna(df['Genre'].mode()[0], inplace=True)

# Convert 'Votes' to numeric and replace the to keep only numeric part
df['Votes'] = pd.to_numeric(df['Votes'].str.replace(',', ''))

# Checking the dataset is there any null values present and data types of the feature present
print('\nDataset Information:\n',df.info())
```

FutureWarning: A value is trying to be set on a copy of a DataFrame

or Series through chained assignment using an inplace method.

The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or

df[col] = df[col].method(value) instead, to perform the operation inplace on the original object

```
  df['Genre'].fillna(df['Genre'].mode()[0], inplace=True)

<class 'pandas.core.frame.DataFrame'>

Index: 11979 entries, 1 to 15508

Data columns (total 10 columns):

 #   Column    Non-Null Count  Dtype

---  ------    --------------  -----

 0   Name      11979 non-null  object

 1   Year      11979 non-null  int64

 2   Duration  11979 non-null  int64

 3   Genre     11979 non-null  object

 4   Rating    11979 non-null  float64

 5   Votes     11979 non-null  int64

 6   Director  11979 non-null  object

 7   Actor 1   11979 non-null  object

 8   Actor 2   11979 non-null  object

 9   Actor 3   11979 non-null  object

dtypes: float64(1), int64(3), object(6)

memory usage: 1.0+ MB
```

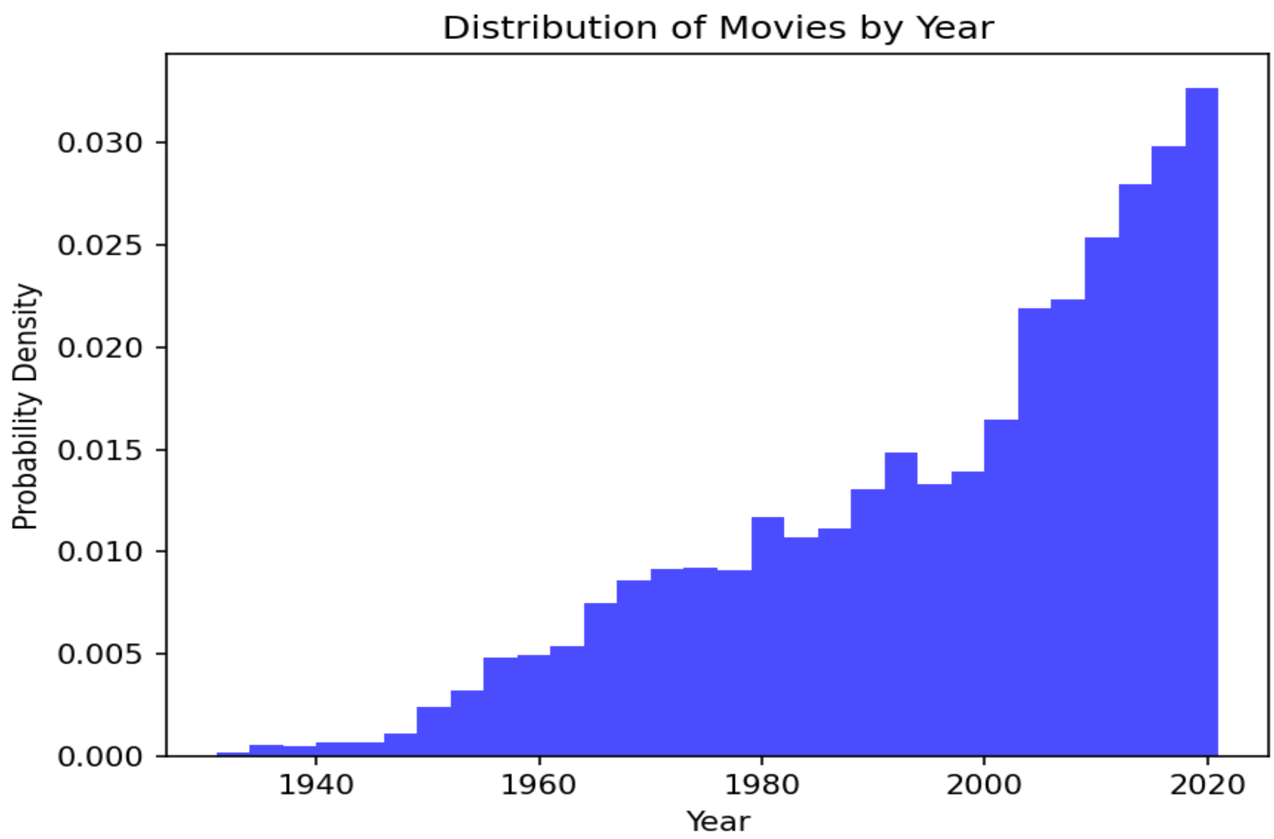
Dataset Information:

 None

5. Data Visualizing

Creating histogram plot

```
# Here we have created a histogram over the years in the data
plt.hist(df['Year'], bins=30, density=True, alpha=0.7, color='blue')
plt.title('Distribution of Movies by Year')
plt.xlabel('Year')
plt.ylabel('Probability Density')
plt.show()
```



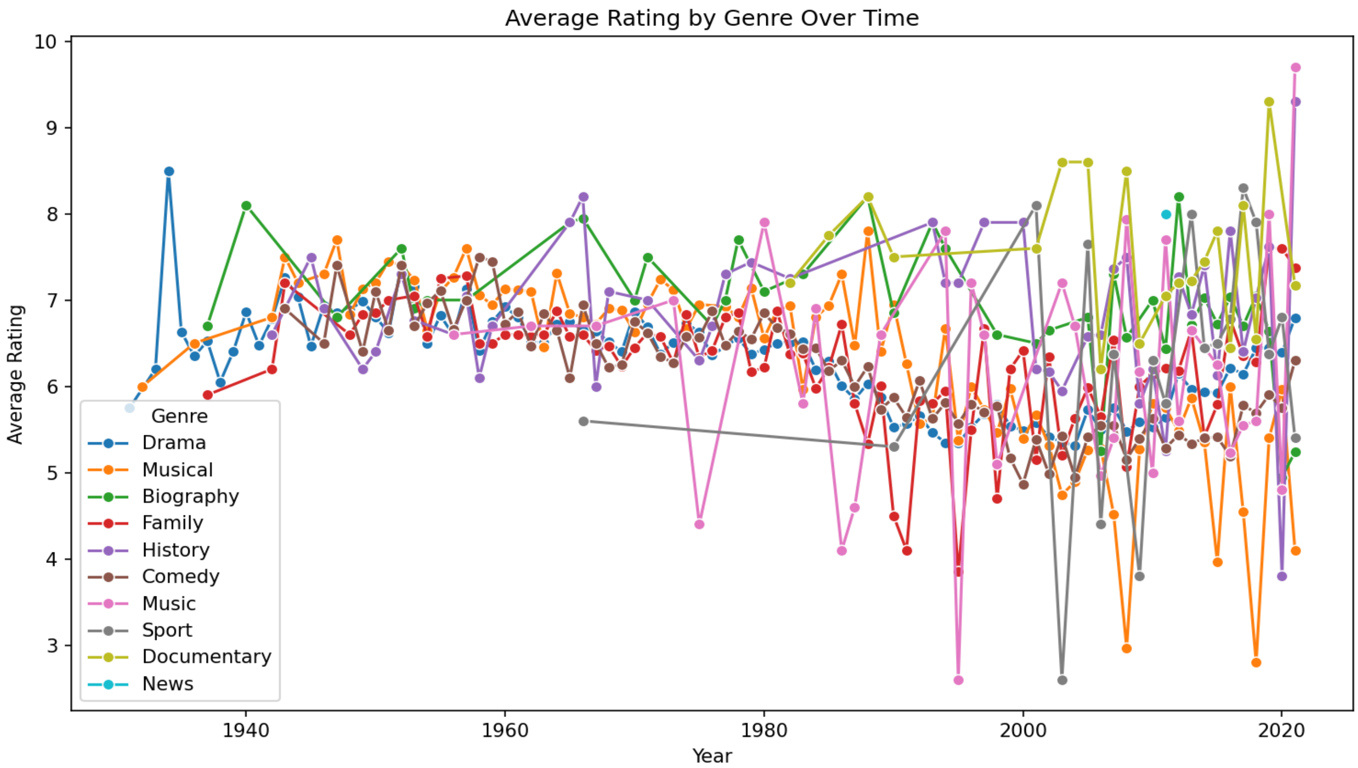

```
# Group data by Year and calculate the average rating
avg_rating_by_year = df.groupby(['Year', 'Genre'])['Rating'].mean().reset_index()
print(avg_rating_by_year.head(10))

# Get the top 10 genres by average rating
top_generes =
avg_rating_by_year.groupby('Genre')['Rating'].mean().sort_values(ascending=False).head(10).
index
print(top_generes)

# Filter the data to include only the top 3 genres
average_rating_by_year = avg_rating_by_year[avg_rating_by_year['Genre'].isin(top_generes)]
```

```
print(average_rating_by_year.head(10))
```

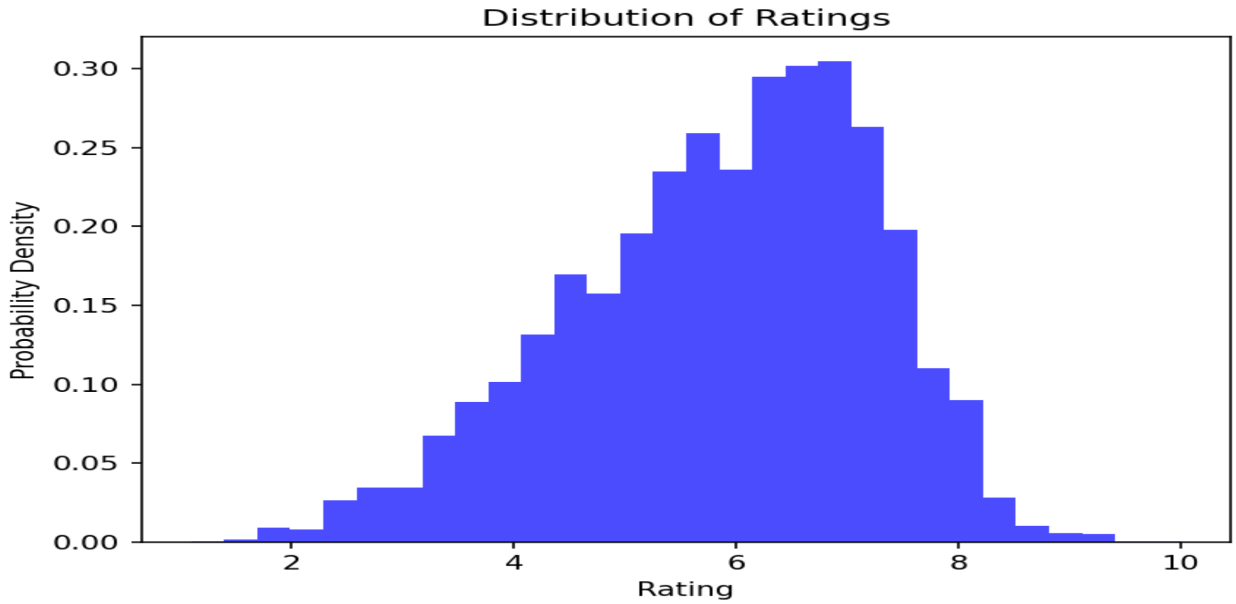
```
   Year     Genre  Rating
0  1931     Drama    5.75
1  1931   Fantasy    6.20
2  1932   Musical    6.00
3  1932   Romance    6.00
4  1933     Drama    6.20
5  1933   Romance    6.20
6  1934 Adventure    2.70
7  1934     Drama    8.50
8  1934   Fantasy    2.70
9  1935    Action    4.50
Index(['News', 'Documentary', 'Biography', 'History', 'Musical', 'Drama',
       'Family', 'Sport', 'Music', 'Comedy'],
      dtype='object', name='Genre')
    Year     Genre    Rating
0   1931     Drama  5.750000
2   1932   Musical  6.000000
4   1933     Drama  6.200000
7   1934     Drama  8.500000
11  1935     Drama  6.633333
13  1936     Drama  6.350000
14  1936   Musical  6.500000
17  1937  Biography  6.700000
18  1937     Drama  6.525000
19  1937    Family  5.900000
```

Creating line plot

```
plt.figure(figsize=(12, 8))
sns.lineplot(data=average_rating_by_year, x='Year', y='Rating', hue='Genre',
marker='o')
plt.title('Average Rating by Genre Over Time')
plt.xlabel('Year')
plt.ylabel('Average Rating')
plt.show()
```

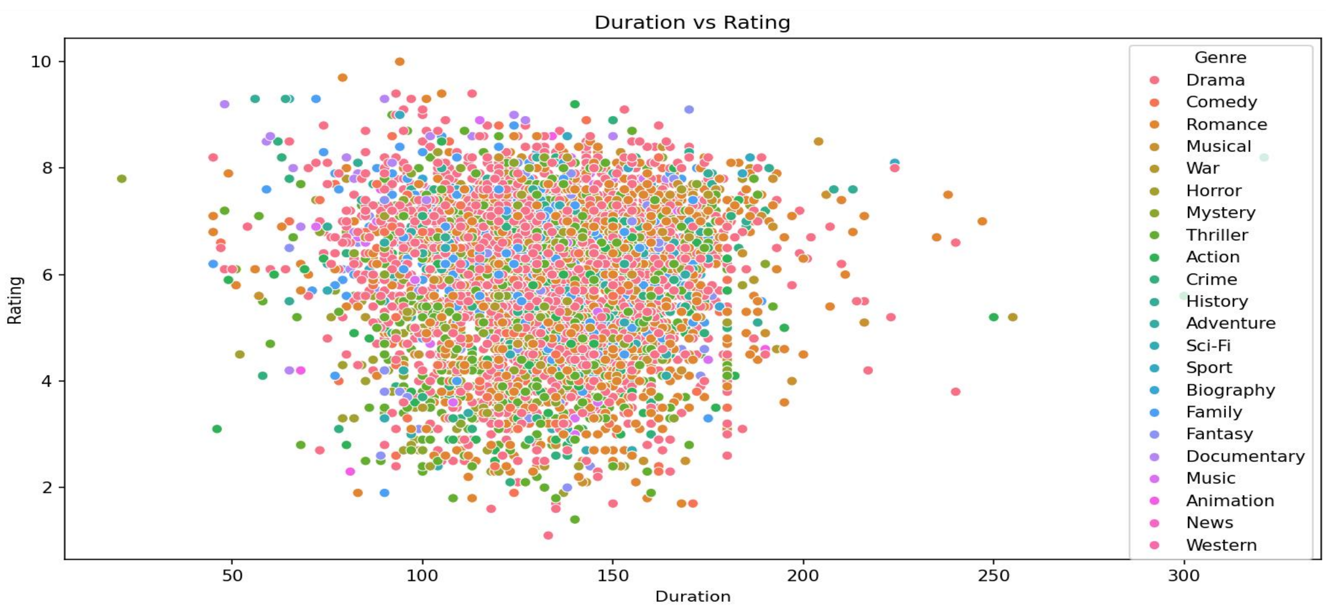



```
# This histogram shows the distribution of ratings and its probability density
plt.hist(df['Rating'], bins=30, density=True, alpha=0.7, color='blue')
plt.title('Distribution of Ratings')
plt.xlabel('Rating')
plt.ylabel('Probability Density')
plt.show()
```



Creating a scatter plot

```
# This scatter plot shows the relationship between the duration of the movie and its
rating
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df, x='Duration', y='Rating', hue='Genre')
plt.title('Duration vs Rating')
plt.xlabel('Duration')
plt.ylabel('Rating')
plt.show()
```



6.Feature Engineering

```
# Drop the Name column because it doesn't impact the outcome
df.drop('Name', axis=1, inplace=True)

# Grouping the columns with their average ratings and then creating a new feature
genre_mean_rating = df.groupby('Genre')['Rating'].transform('mean')
df['Genre_mean_rating'] = genre_mean_rating

director_mean_rating = df.groupby('Director')['Rating'].transform('mean')
df['Director_encoded'] = director_mean_rating

actor1_mean_rating = df.groupby('Actor 1')['Rating'].transform('mean')
df['Actor1_encoded'] = actor1_mean_rating

actor2_mean_rating = df.groupby('Actor 2')['Rating'].transform('mean')
df['Actor2_encoded'] = actor2_mean_rating

actor3_mean_rating = df.groupby('Actor 3')['Rating'].transform('mean')
df['Actor3_encoded'] = actor3_mean_rating

# Display the updated DataFrame
print(df.head())
```

Year Duration Genre Rating Votes ... Genre_mean_rating Director_encoded Actor1_encoded Actor2_encoded Actor3_encoded

1 2019 109 Drama 7.0 8 ... 6.056744 7.000000 6.850000 7.00 7.00

3 2019 110 Comedy 4.4 35 ... 5.751042 4.400000 5.250000 4.40 4.46

3 2019 110 Romance 4.4 35 ... 5.811087 4.400000 5.250000 4.40 4.46

5 1997 147 Comedy 4.7 827 ... 5.751042 5.335135 4.793617 5.73 5.93

5 1997 147 Drama 4.7 827 ... 6.056744 5.335135 4.793617 5.73 5.93

[5 rows x 14 columns

7. Split Data

```
# Keeping the predictor and target variables
X = df[['Year', 'Votes', 'Duration', 'Genre_mean_rating', 'Director_encoded',
'Actor1_encoded', 'Actor2_encoded', 'Actor3_encoded']]
y = df['Rating']

# Splitting the data into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print(f"Train shapes: X_train: {X_train.shape}, y_train: {y_train.shape}")
```

```
print(f"Test shapes: X_test: {X_test.shape}, y_test: {y_test.shape}")
```

Train shapes: X_train: (9583, 8), y_train: (9583,)

Test shapes: X_test: (2396, 8), y_test: (2396,)

8. Model Building

```
# Initializing and training the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions
y_pred = model.predict(X_test)

# Calculating evaluation metrics
print('Accuracy (R2 Score):', model.score(X_test, y_test))
print('Mean Squared Error:', mean_squared_error(y_test, y_pred))

# Handling potential errors for negative predictions in mean_squared_log_error
y_pred = [max(val, 0) for val in y_pred]
print('Mean Squared Log Error:', mean_squared_log_error(y_test, y_pred))

print('R2 Score:', r2_score(y_test, y_pred))
```

Accuracy (R2 Score): 0.7641133663863862

Mean Squared Error: 0.4465441653985704

Mean Squared Log Error: 0.012458877846073142

R2 Score: 0.7641133663863862

Model Testing

```
print(X.head())
print(y.head())

# For testing , we create a new dataframe with values close to the any of our existing data
to evaluate
data = {'Year': [2019], 'Votes': [36], 'Duration': [111], 'Genre_mean_rating': [5.8],
'Director_encoded' : [4.5], 'Actor1_encoded' : [5.3], 'Actor2_encoded' : [4.5],
'Actor3_encoded' : [4.5]}
trail = pd.DataFrame(data)

# Predict the movie rating by entered data
rating = model.predict(trail)
```

```
# Displat the predicted rating
print('Predicted Rating:', rating[0])
```

Year	Votes	Duration	Genre_mean_rating	Director_encoded	Actor1_encoded	Actor2_encoded	Actor3_encoded
1 2019	8	109	6.056744	7.000000	6.850000	7.00	7.00
3 2019	35	110	5.751042	4.400000	5.250000	4.40	4.46
3 2019	35	110	5.811087	4.400000	5.250000	4.40	4.46
5 1997	827	147	5.751042	5.335135	4.793617	5.73	5.93
5 1997	827	147	6.056744	5.335135	4.793617	5.73	5.93

1 7.0

3 4.4

3 4.4

5 4.7

5 4.7

Name: Rating, dtype: float64

Predicted Rating: 4.2074589621343295

Conclusion:

The movie rating prediction program provides a comprehensive approach to analyze, clean, and model movie data to predict ratings based on various features like year, votes, duration, genre, director, and actors.

Insights:

- The program highlights the importance of feature engineering and cleaning in building effective predictive models.
- Linear regression works well for predicting continuous variables like movie ratings, but its performance could be further enhanced using more complex models like Random Forest or Gradient Boosting if required.