MOVIE RATING PREDICTION WITH PYTHON

Objective:

The primary objective of this study is to predict the ratings of movies based on their attributes such as year, duration, genre, director, cast, and number of votes. This analysis also aims to identify trends and patterns in the movie dataset, such as top-rated genres, directors, and actors, and explore the factors influencing movie ratings

Data Dictionary:

The IMDB movies India dataset contains the following features

- **Name**: Name of the movie.
- **Year**: Year of release.
- **Duration**: Duration of the movie in minutes.
- **Rating**: IMDB rating of the movie.
- **Votes:** Number of votes received.
- **Genre**: Movie genre(s).
- **Director**: Director(s) of the movie.
- **Actor 1, Actor 2, Actor 3**: Leading actors in the movie.

Key Steps:

- **Data Preparation**: Handle missing values, duplicates, and convert data types.
- **EDA**: Analyze trends, correlations, and top genres, directors, and actors.
- **Preprocessing**: Scale numeric data and one-hot encode categorical features.
- **Modeling**: Train a RandomForestRegressor on split data.
- **Evaluation**: Assess with MSE, MAE, and visualize predictions

Packages:

- **pandas**: For data manipulation (loading, cleaning, inspecting datasets).
- **numpy**: For numerical operations (calculations, handling NaN values).
- **train_test_split**: Splits the data into training and testing sets for model evaluation.
- **SimpleImputer**: Handles missing data by replacing it with the mean, median, or most frequent value.
- **OneHotEncoder**: Converts categorical features (e.g., genre) into numerical format.
- **StandardScaler**: Standardizes numeric features to improve model performance.
- **ColumnTransformer**: Applies different preprocessing to numeric and categorical columns.

- **Pipeline**: Combines preprocessing and modeling steps for efficiency and reproducibility.
- **RandomForestRegressor**: A powerful machine learning model for predicting ratings.
- **LinearRegression**: A simpler model for identifying trends in the data.

Step 1: Load the Dataset and Libraries

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
import matplotlib.pyplot as plt
import seaborn as sns

# Step 1: Load Dataset

df = pd.read_csv('imbd movies.csv', encoding='ISO-8859-1')
print("Dataset Head:\n",df.head())
```

output:

```
Dataset Head:
                                Name    Year Duration          Genre  ...          Director      Actor 1          Actor 2         Actor 3
0                                     NaN      NaN          Drama  ...      J.S. Randhawa     Manmauji           Birbal  Rajendra Bhatia
1  #Gadhvi (He thought he was Gandhi) -2019.0  109 min          Drama  ...      Gaurav Bakshi  Rasika Dugal     Vivek Ghamande   Arvind Jangid
2                     #Homecoming -2021.0   90 min  Drama, Musical  ... Soumyajit Majumdar  Sayani Gupta  Plabita Borthakur       Roy Angana
3                         #Yaaram -2019.0  110 min  Comedy, Romance  ...        Ovais Khan       Prateik        Ishita Raj  Siddhant Kapoor
4                ...And Once Again -2010.0  105 min          Drama  ...      Amol Palekar  Rajat Kapoor  Rituparna Sengupta     Antara Mali
```

Data Exploration

```
print("\nMissing Values:\n", df.isnull().sum())
print("Summary of the statistics of numerical columns:")
print(df.describe())
print("Dimensions of the DataFrame:")
print(df.shape)
print("Number of duplicate rows:", df.duplicated().sum())
```

Output:

Missing Values:

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

Summary of the statistics of numerical columns:

	Year	Rating
count	14981.000000	7919.000000
mean	-1987.012215	5.841621
std	25.416689	1.381777
min	-2022.000000	1.100000
25%	-2009.000000	4.900000
50%	-1991.000000	6.000000
75%	-1968.000000	6.800000
max	-1913.000000	10.000000

Dimensions of the Data Frame:

(15509, 10)

Number of duplicate rows: 6

Data Cleaning:

```
df.dropna(inplace=True)                                    # Drop missing
values
df.drop_duplicates(inplace=True)
```

Handling Missing Values and Data Types:

```
df.fillna(df.mean(numeric_only=True), inplace=True)
combined_actors = pd.concat([df['Actor 1'], df['Actor 2'], df['Actor 3']])
combined_actors = combined_actors.dropna()
df.dropna(subset=['Genre', 'Director'], inplace=True)
df['Year'] = df['Year'].astype(int)
df['Duration'] = pd.to_numeric(df['Duration'], errors='coerce')
```

Step 2: Exploratory Data Analysis (EDA)

#Year with the Best Rating

```
best_year = df.groupby('Year')['Rating'].mean().idxmax()
best_year_rating = df.groupby('Year')['Rating'].mean().max()
print(f"Year with the best rating: {best_year} with an average rating of
{best_year_rating}")
```

output:

Year with the best rating: -1952 with an average rating of 7.2125

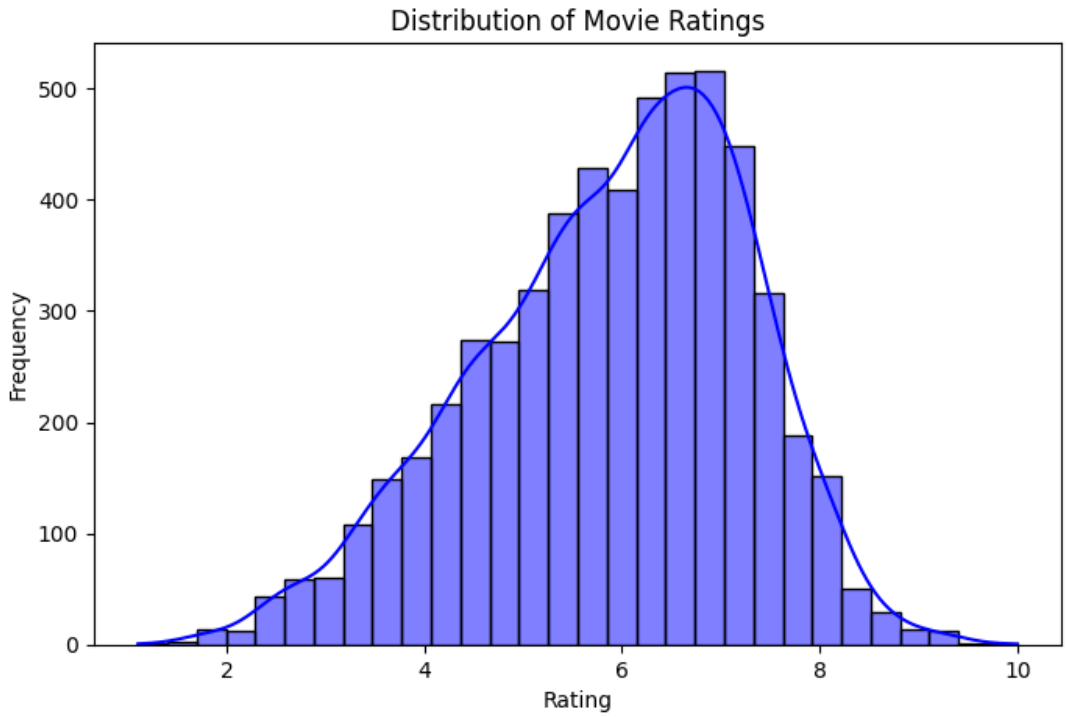
 # Correlation Between Duration and Rating

```
correlation = df['Duration'].corr(df['Rating'])
print(f"Correlation between Duration and Rating: {correlation}")
```

Output: Correlation between Duration and Rating: nan

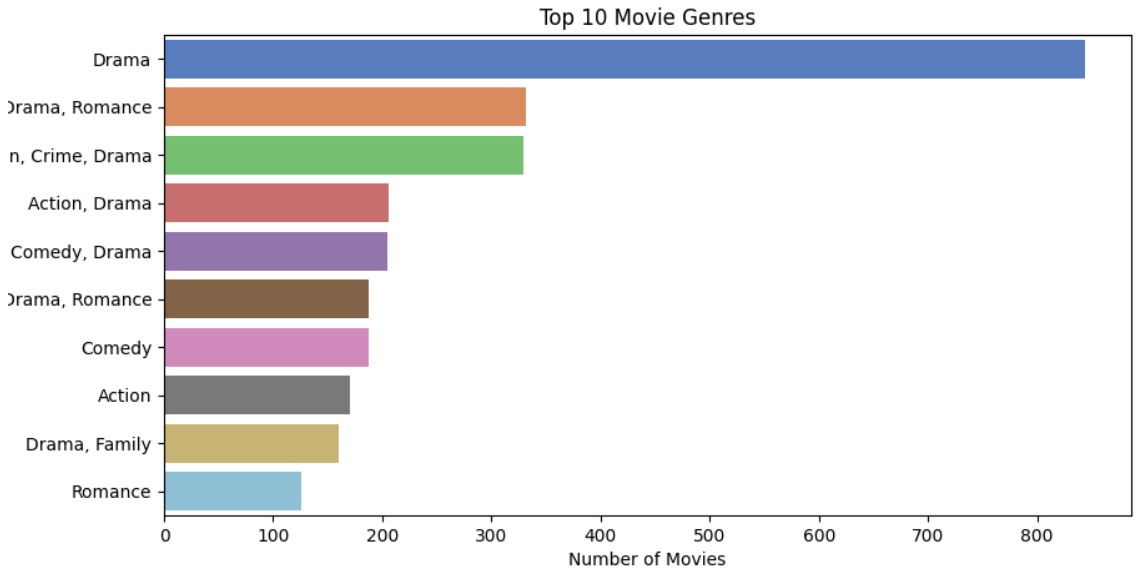
 # Distribution of Movie Ratings

```
plt.figure(figsize=(8, 5))
sns.histplot(df['Rating'], bins=30, kde=True, color='blue')
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Frequency')
plt.show()
```



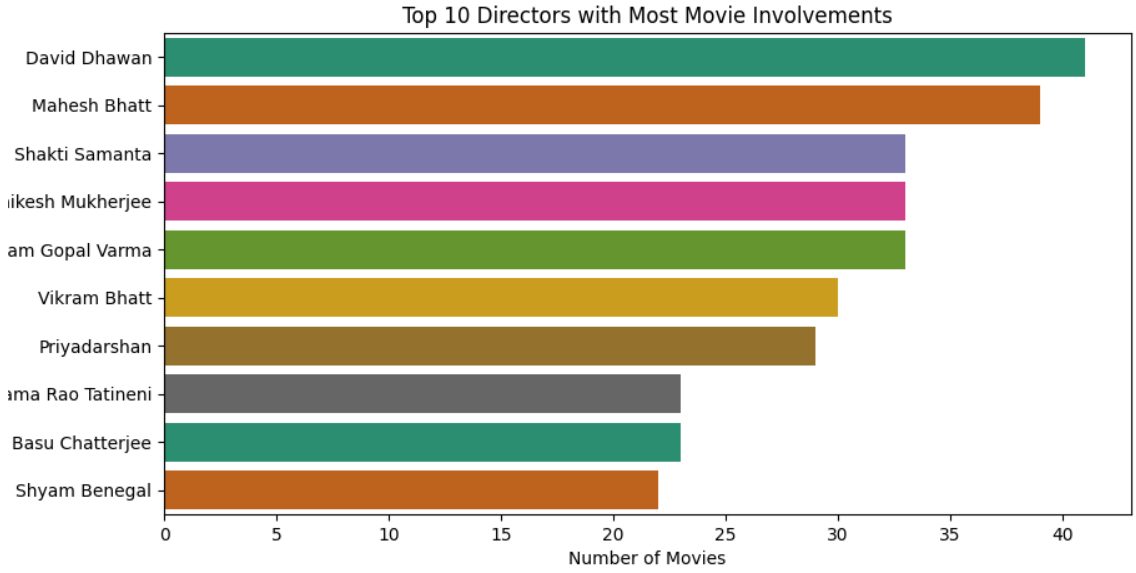
 # Top 10 Movie Genres

```
top_10_genres = df['Genre'].value_counts(ascending=False).head(10)
plt.figure(figsize=(10,5))
sns.barplot(x=top_10_genres.values, y=top_10_genres.index,  palette='muted')
plt.title('Top 10 Movie Genres')
plt.xlabel('Number of Movies')
plt.ylabel('Genre')
plt.show()
```



Top 10 Directors with Most Movie Involvements

```
top_10_directors = df['Director'].value_counts(ascending=False).head(10)
plt.figure(figsize=(10, 5))
sns.barplot(x=top_10_directors.values,
y=top_10_directors.index,  palette='Dark2')
plt.title('Top 10 Directors with Most Movie Involvements')
plt.xlabel('Number of Movies')
plt.ylabel('Director')
plt.show()
```



 # Data Analysis on Actors

```
# Count the frequency of each actor
top_10_actors = combined_actors.value_counts().head(10)
# Display the top 10 actors
print("Top 10 Actors:",top_10_actors)
```

Output:

Top 10 Actors:

Mithun Chakraborty 160

Amitabh Bachchan 148

Dharmendra 146

Ashok Kumar 124

Akshay Kumar 120

Shashi Kapoor 117

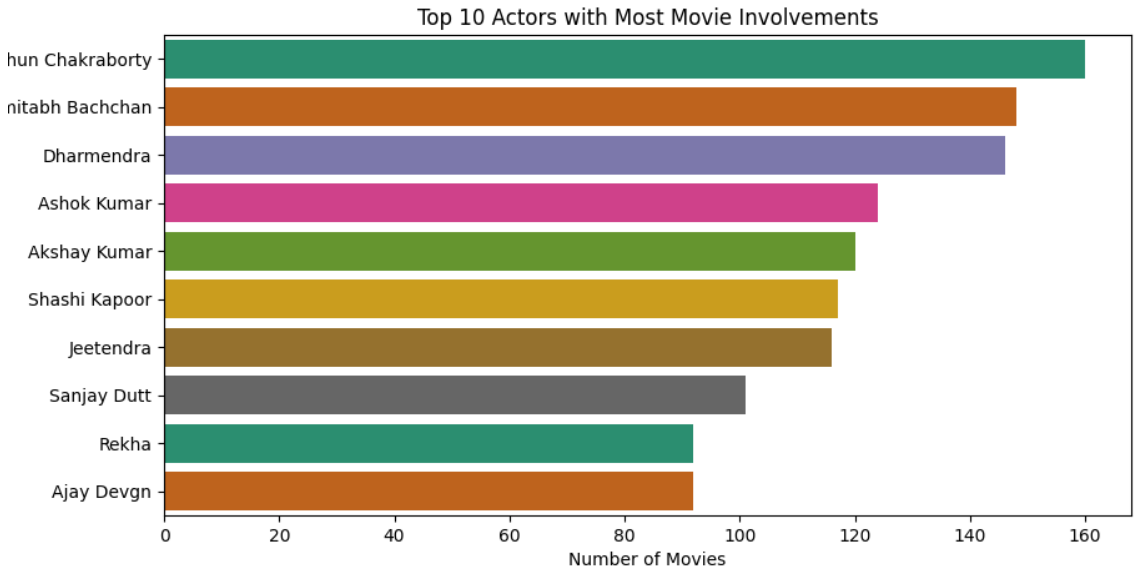
Jeetendra 116

Sanjay Dutt 101

Rekha 92

Ajay Devgn 92

```
# Plot the top 10 actors
plt.figure(figsize=(10, 5))
sns.barplot(x=top_10_actors.values, y=top_10_actors.index, palette='Dark2')
plt.title('Top 10 Actors with Most Movie Involvements')
plt.xlabel('Number of Movies')
plt.ylabel('Actor')
plt.show()
plt.tight_layout()
```



 # Movie Ratings and Votes

```
#Top 10 Movies by Rating (Overall and Per Year)
# Overall
top_10_movies = df.nlargest(10, 'Rating')[['Name', 'Rating', 'Year']]
print("Top 10 Movies Overall:",top_10_movies)
# Per Year
# Get the top 10 movies per year based on Rating
top_movies_per_year = df.groupby('Year').apply(lambda x: x.nlargest(10,
'Rating'))
top_movies_per_year = top_movies_per_year.reset_index(drop=True)
print("Top Movies Per Year:",top_movies_per_year[['Name', 'Rating', 'Year']])
```

Output: Top 10 Movies Overall:

	Name	Rating	Year
8339	Love Qubool Hai	10.0	-2020
5410	Half Songs	9.7	-2021
5077	Gho Gho Rani	9.4	-201
6852	June	9.4	-2021
14222	The Reluctant Crime	9.4	-2020

1314 Ashok Vatika 9.3 -2018

1729 Baikunth 9.3 -2021

5125 God of gods 9.3 -2019

8344 Love Sorries 9.3 -2021

11843 Refl3ct 9.3 -2021

Top Movies Per Year:

	Name	Rating	Year
0	**Half Songs**	**9.7**	**-2021**
1	**June**	**9.4**	**-2021**
2	**Baikunth**	**9.3**	**-2021**
3	**Love Sorries**	**9.3**	**-2021**
4	**Refl3ct**	**9.3**	**-2021**
..	**...**	**...**	**...**
812	**Piya Pyare**	**2.7**	**-1934**
813	**Fate**	**6.2**	**-1933**
814	**Indrasabha**	**6.0**	**-1932**
815	**The Light of the World**	**6.2**	**-1931**
816	**Draupadi**	**5.3**	**-1931**

```
# Number of Popular Movies Released Each Year
popular_movies_per_year = df[df['Rating'] > 7.5].groupby('Year')['Name'].count()
print("Number of Popular Movies Released Each Year:",popular_movies_per_year)
popular_movies_per_year.plot(kind='bar', title='Popular Movies Released Each
Year', figsize=(12, 6))
plt.ylabel('Number of Popular Movies')
plt.show()
```

Output: Number of Popular Movies Released Each Year

Year

-2021 23

-2020 28

-2019 51

-2018 37

-2017 28

..

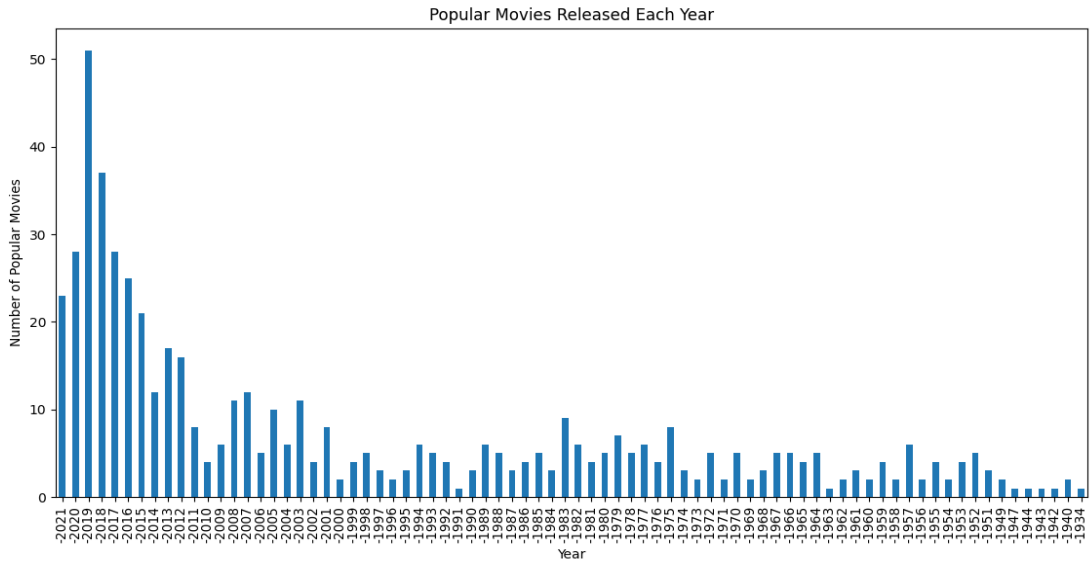
-1944 1

-1943 1

-1942 1

-1940 2

-1934 1



 # Most Popular Movies by Votes

```
# Most Votes Overall
# Convert 'Votes' to numeric and handle non-numeric values
```

```
df['Votes'] = pd.to_numeric(df['Votes'], errors='coerce')
df.dropna(subset=['Votes'], inplace=True)
print("datatype:",df['Votes'].dtype)
most_votes_movies = df.nlargest(10, 'Votes')[['Name', 'Votes', 'Rating']]
print("Movies with Most Votes Overall:", most_votes_movies)
```

Output:

datatype: float64

Movies with Most Votes Overall:

	Name	Votes	Rating
12569	Satyam Shivam Sundaram: Love Sublime	999.0	7.1
5663	Hera Pheri	998.0	6.8
7930	Kurukshetra	996.0	6.0
3961	Dishkiyaoon	986.0	5.2
6241	Ittefaq	985.0	7.4
7083	Kadvi Hawa	985.0	8.1
9822	Naam	984.0	7.5
12	100 Days	983.0	6.5
2612	Bumboo	982.0	6.0
2453	Blood Money	981.0	5.0

```
# Most Votes Per Year
most_votes_per_year = df.groupby('Year').apply(lambda x: x.nlargest(1,'Votes'))
most_votes_per_year = most_votes_per_year.reset_index(drop=True)
print("Movies with Most Votes Per Year:",most_votes_per_year[['Name', 'Votes',
'Year']])
```

Output: Movies with Most Votes Per Year:

	Name	Votes	Year

0	Ammaa Ki Boli	871.0	-2021
1	Harami	957.0	-2020
2	Chhappad Phaad Ke	954.0	-2019
3	Nanu Ki Jaanu	954.0	-2018
4	Kadvi Hawa	985.0	-2017
..	...	...	...
86	Inquilab	38.0	-1935
87	Piya Pyare	11.0	-1934
88	Fate	12.0	-1933
89	Indrasabha	12.0	-1932
90	The Light of the World	112.0	-1931

```
# High-Performing Movies Based on Rating
high_rated_movies = df[df['Rating'] >= 8]
print("High-Performing Movies (Rating >= 8):")
print(high_rated_movies[['Name', 'Rating', 'Year']])
```

Output: High-Performing Movies (Rating >= 8):

	Name	Rating	Year
137	A Billion Colour Story	8.1	-2016
151	A Gift of Love: Sifar	8.0	-2019
176	AA BB KK	8.0	-2018
944	Ammaa Ki Boli	8.1	-2021
1314	Ashok Vatika	9.3	-2018
...	...	...	...
14898	Viraat	8.9	-2021

15071 Win Marathon 9.0 -2018

15116 Writing with Fire 8.1 -2021

15288 Yeh Suhaagraat Impossible 8.6 -2019

15470 Zindagi Ek Safar 8.2 -1988

```
# Director Directed the Most Movies?
most_movies_director = df['Director'].value_counts().idxmax()
most_movies_count = df['Director'].value_counts().max()
print(f"Director with the most movies: {most_movies_director}
({most_movies_count} movies)")

# Actor Starred in the Most Movies?
df['combined_actors'] = df[['Actor 1', 'Actor 2', 'Actor 3']].apply(
    lambda row: ', '.join(row.dropna()), axis=1)
combined_actors = df['combined_actors'].str.split(',').explode()
most_movies_actor = combined_actors.value_counts().idxmax()
most_movies_actor_count = combined_actors.value_counts().max()
print(f"Actor with the most movies: {most_movies_actor}
({most_movies_actor_count} movies)")
```

Output:

Director with the most movies: Shakti Samanta (31 movies)

Actor with the most movies: Mithun Chakraborty (87 movies)

 # Other Trends and Insights

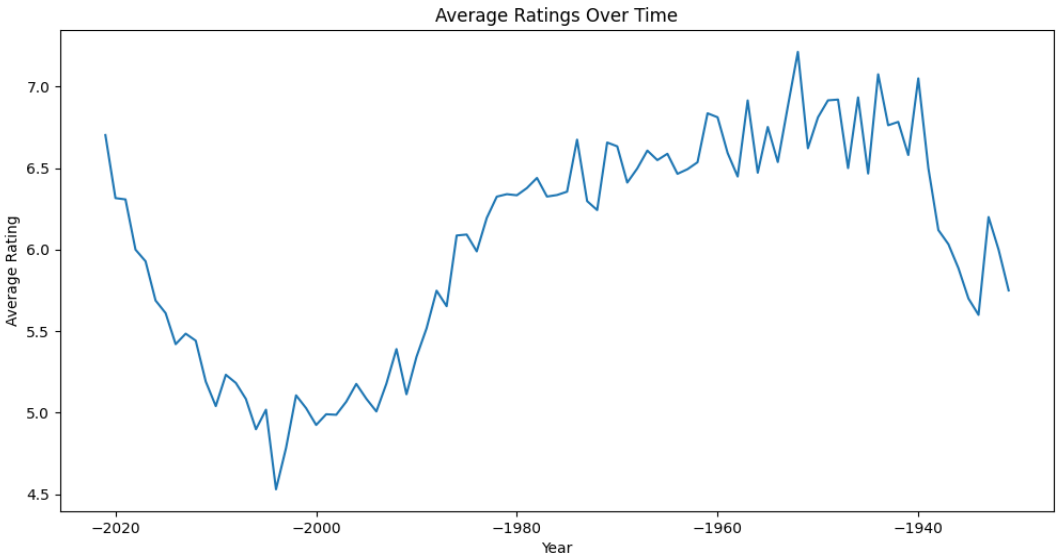
```
# Genre Analysis
genre_ratings = df.groupby('Genre')['Rating'].mean().sort_values(ascending=False)
print("Average Ratings by Genre:",genre_ratings)

# Trend in Ratings Over Time
ratings_trend = df.groupby('Year')['Rating'].mean()
ratings_trend.plot(title='Average Ratings Over Time', figsize=(12, 6))
plt.ylabel('Average Rating')
plt.show()
```

Output: Average Ratings by Genre

Genre

History, Romance	9.4
Documentary, Family, History	9.3
Documentary, Music	8.9
Biography, Crime, Drama	8.9
Animation, Comedy, Family	8.3
...	
Adventure, Drama, Sci-Fi	3.3
Comedy, Horror, Musical	2.7
Action, Crime, Sci-Fi	2.7
Comedy, Drama, Sport	2.6
Family, Music, Romance	2.6



Step 3: Model Building

```
#Future Predictions (Using Linear Regression)
X = ratings_trend.index.values.reshape(-1, 1)  # Years
y = ratings_trend.values
model = LinearRegression()
```

```
model.fit(X, y)

#Model Building for Rating Prediction
X = df[['Year', 'Duration', 'Votes', 'Genre', 'Director', 'Actor 1', 'Actor 2',
'Actor 3']]
y = df['Rating']
print("missing values:",df['Duration'].isnull().sum())

df['Duration'] = df['Duration'].fillna(0)
df['Duration'] = pd.to_numeric(df['Duration'], errors='coerce')
print("after cleaning duplicates:",df['Duration'].isnull().sum())
```

Output:

missing values: 4291

after cleaning duplicates: 0

Step 4: Preprocessing the Data

```
# Categorical variables (e.g., Genre, Director, Actor columns) need encoding
categorical_features = ['Genre', 'Director', 'Actor 1', 'Actor 2', 'Actor 3']
numeric_features = ['Year', 'Duration', 'Votes']
# Create the column transformer to handle different feature types
preprocessor = ColumnTransformer(
    transformers=[
        ('num', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='mean')),  # Only apply to numeric
features
            ('scaler', StandardScaler())  # Standardize numeric features
        ]), numeric_features),

        ('cat', Pipeline(steps=[
            ('imputer', SimpleImputer(strategy='most_frequent')),
            ('encoder', OneHotEncoder(handle_unknown='ignore'))
        ]), categorical_features)
    ])
```

Step 5: Train-Test Split

```
X = df[['Year', 'Duration', 'Votes', 'Genre', 'Director', 'Actor 1', 'Actor 2',
'Actor 3']]  # Features
```

```
y = df['Rating']  # Target variable (Rating)

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

print(f"Training set size: {X_train.shape[0]} samples")
print(f"Testing set size: {X_test.shape[0]} samples")
```

Output :

Training set size: 3432 samples

Testing set size: 859 samples

Step 6: Model Pipeline and Training

```
Creating a Pipeline with Preprocessing and Model
pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('model', RandomForestRegressor(n_estimators=100, random_state=42))
])
# Train the model
pipeline.fit(X_train, y_train)
#Make predictions on the test set
y_pred = pipeline.predict(X_test)
# Calculate mean squared error (MSE) and mean absolute error (MAE)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.2f}")
print(f"Mean Absolute Error: {mae:.2f}")
```

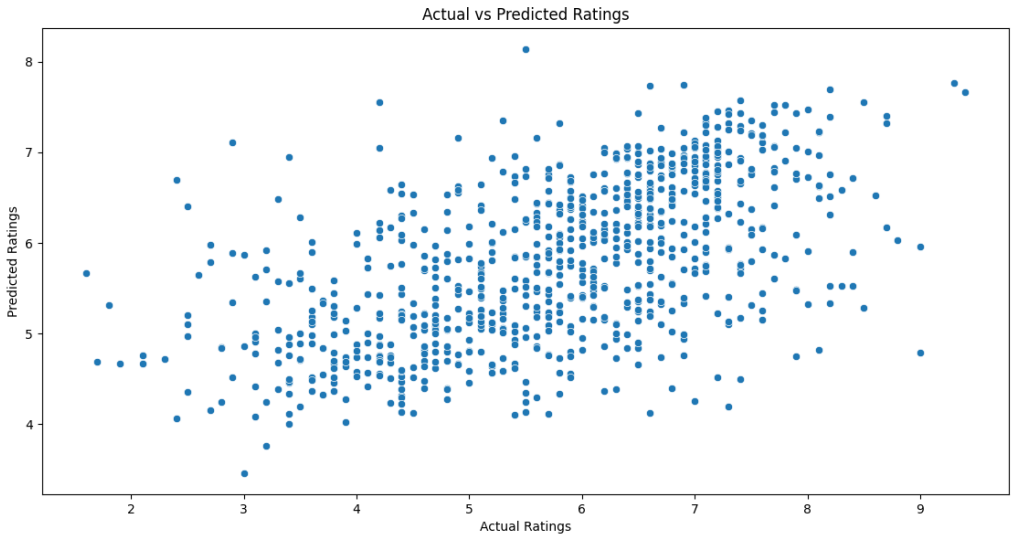
Output:

Mean Squared Error: 1.34

Mean Absolute Error: 0.86

 # Actual vs Predicted Ratings

```
# Visualize Actual vs Predicted Ratings
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.title('Actual vs Predicted Ratings')
plt.xlabel('Actual Ratings')
plt.ylabel('Predicted Ratings')
plt.show()
```



Conclusion:

The analysis revealed that genres with high ratings and directors with strong performance trends. Movies with more votes often had better ratings. The Random Forest model showed reasonable accuracy, though some deviations were due to outliers or feature limitations. Future improvements could include adding audience demographics and production budgets, and exploring advanced techniques like deep learning for better accuracy. This case study demonstrates the potential of data analytics and machine learning in making predictions from movie datasets.