```
# @title 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
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression

# @title Load the Dataset
df = pd.read_csv("IMDb Movies India.csv", encoding='latin1')
df.head()
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 15509,\n  \"fields\":
[\n    {\n      \"column\": \"Name\",\n      \"properties\": {\n
\"dtype\": \"string\",\n        \"num_unique_values\": 13838,\n
\"samples\": [\n          \"Tumsa Nahin Dekha\",\n
\"Thanedaar\",\n          \"Farzande Hind\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Year\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 102,\n
\"samples\": [\n          \"(1974)\",\n          \"(1963)\",\n
\"(1971)\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Duration\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 182,\n
\"samples\": [\n          \"168 min\",\n          \"99 min\",\n
\"179 min\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Genre\",\n      \"properties\": {\n        \"dtype\": \"category\",\
n        \"num_unique_values\": 485,\n        \"samples\": [\n
\"Family, Mystery\",\n          \"Comedy, Fantasy, Musical\",\n
\"Action, Adventure, Crime\"\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Rating\",\n      \"properties\": {\n        \"dtype\":
\"number\",\n        \"std\": 1.3817771548659543,\n        \"min\":
1.1,\n        \"max\": 10.0,\n        \"num_unique_values\": 84,\n
\"samples\": [\n          1.4,\n          7.0,\n          2.9\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"Votes\",\n      \"properties\":
{\n        \"dtype\": \"category\",\n        \"num_unique_values\":
2034,\n        \"samples\": [\n          \"837\",\n          \"101\",\
n          \"2,566\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Director\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 5938,\n
\"samples\": [\n          \"Param Hans Chitra\",\n
```

\"Vickram\",\n   \"Zaigham Ali Syed\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Actor 1\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
4718,\n   \"samples\": [\n   \"Abhishek Bachchan\",\n
\"Giri Babu\",\n   \"Ajay Bafna\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Actor 2\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
4891,\n   \"samples\": [\n   \"Bipasha Basu\",\n
\"Mukul Dev\",\n   \"Arjun Kapoor\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Actor 3\",\n   \"properties\":
{\n   \"dtype\": \"category\",\n   \"num_unique_values\":
4820,\n   \"samples\": [\n   \"Damandeep Singh\",\n
\"Kamini Kaushal\",\n   \"Babu\"\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   }\n]\n}","type":"dataframe","variable_name":"df"}

```
# @title Dataset Understanding
df.info()
```

```
<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
```

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

```
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
```

```
# @title Data Cleaning
df = df[df['Rating'].notna()]

df['Year'] = df['Year'].str.replace('(', '').str.replace(')',
'').astype(int)

df['Duration'] = df['Duration'].str.replace(' min', '').astype(float)

df['Votes'] = df['Votes'].str.replace(',', '').astype(float)
```

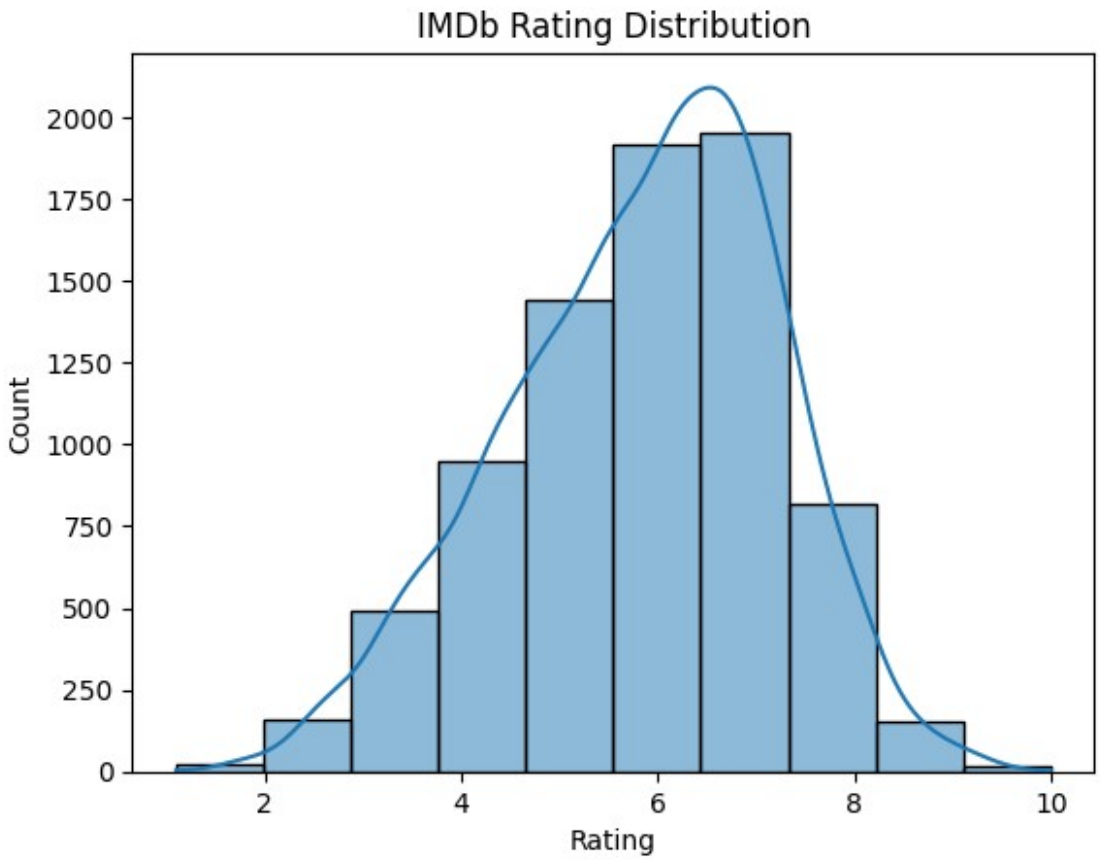
```
# @title Handle Missing Values
df['Genre'].fillna(df['Genre'].mode()[0], inplace=True)
df['Director'].fillna(df['Director'].mode()[0], inplace=True)
df['Actor 1'].fillna(df['Actor 1'].mode()[0], inplace=True)
df['Actor 2'].fillna(df['Actor 2'].mode()[0], inplace=True)
df['Actor 3'].fillna(df['Actor 3'].mode()[0], inplace=True)
df['Duration'].fillna(df['Duration'].median(), inplace=True)
df['Votes'].fillna(df['Votes'].median(), inplace=True)
```

```
/tmp/ipython-input-1461441439.py:1: 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)
/tmp/ipython-input-1461441439.py:2: 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['Director'].fillna(df['Director'].mode()[0], inplace=True)
```

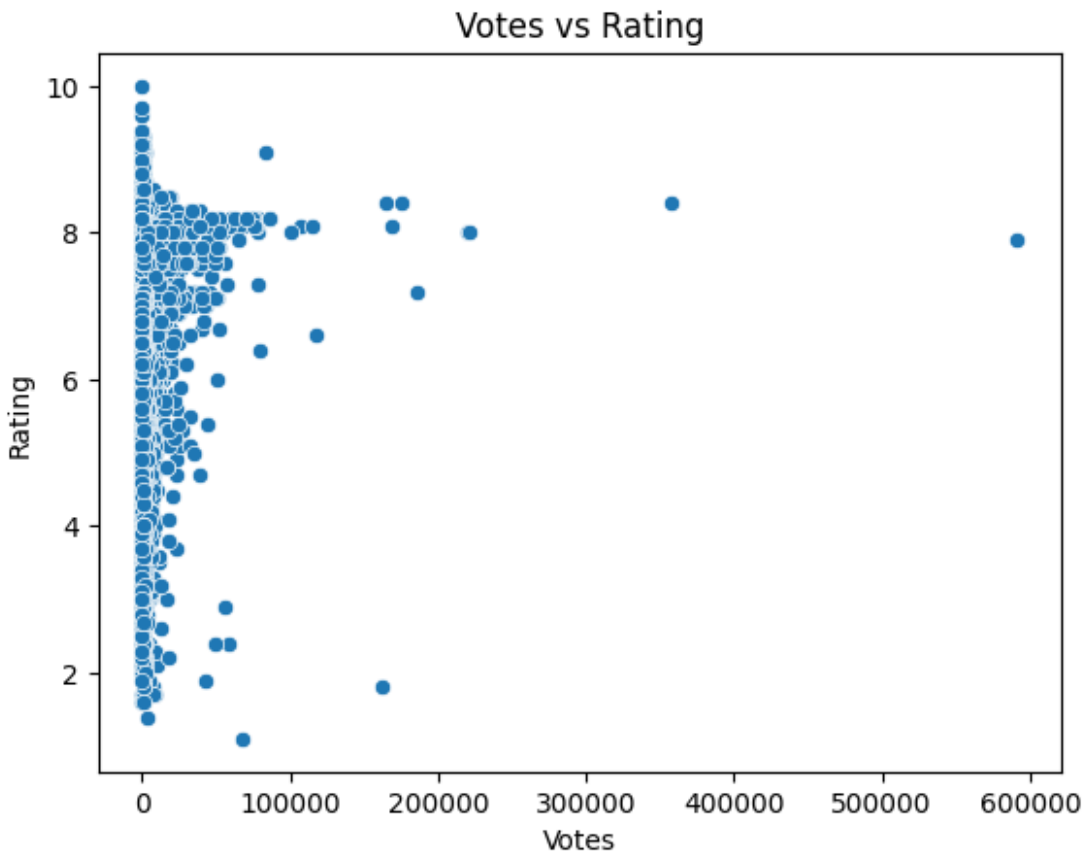
```
/tmp/ipython-input-1461441439.py:3: FutureWarning: A value is trying
to be set on a copy of a DataFrame or Series through chained
assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting 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['Actor 1'].fillna(df['Actor 1'].mode()[0], inplace=True)
/tmp/ipython-input-1461441439.py:4: 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['Actor 2'].fillna(df['Actor 2'].mode()[0], inplace=True)
/tmp/ipython-input-1461441439.py:5: 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['Actor 3'].fillna(df['Actor 3'].mode()[0], inplace=True)
/tmp/ipython-input-1461441439.py:6: 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['Duration'].fillna(df['Duration'].median(), inplace=True)
/tmp/ipython-input-1461441439.py:7: 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['Votes'].fillna(df['Votes'].median(), inplace=True)
```

```
# @title Exploratory Data Analysis (EDA)
sns.histplot(df['Rating'], bins=10, kde=True)
plt.title("IMDb Rating Distribution")
plt.show()
```




```
sns.scatterplot(x='Votes', y='Rating', data=df)
plt.title("Votes vs Rating")
plt.show()
```




```
# @title Encode Categorical Columns
le = LabelEncoder()

df['Genre'] = le.fit_transform(df['Genre'])
df['Director'] = le.fit_transform(df['Director'])
df['Actor 1'] = le.fit_transform(df['Actor 1'])
df['Actor 2'] = le.fit_transform(df['Actor 2'])
df['Actor 3'] = le.fit_transform(df['Actor 3'])

# @title Feature Selection
X = df[['Year', 'Duration', 'Votes', 'Genre', 'Director', 'Actor 1',
'Actor 2', 'Actor 3']]
y = df['Rating']

# @title Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# @title Build Regression Model
model = LinearRegression()
model.fit(X_train, y_train)

LinearRegression()
```

```python
# @title Prediction
y_pred = model.predict(X_test)

# @title Model Evaluation
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 1.6793329538213715
```

```python
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
R2 Score: 0.09671767033692713
```

```python
# @title conclusion
print("Movie Rating Prediction model built successfully.")
print("Votes, duration, and genre have strong influence on ratings.")
```

```
Movie Rating Prediction model built successfully.
Votes, duration, and genre have strong influence on ratings.
```