

```
In [1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In [2]: df = pd.read_csv("IMDb Movies India.csv", encoding='latin-1')
df.head(10)
```

Out[2]:

	Name	Year	Duration	Genre	Rating	Votes	Director	Actor 1	Actor 2	Ac
0		NaN	NaN	Drama	NaN	NaN	J.S. Randhawa	Manmauji	Birbal	Raj E
1	#Gadhvi (He thought he was Gandhi)	(2019)	109 min	Drama	7.0	8	Gaurav Bakshi	Rasika Dugal	Vivek Ghamande	A J
2	#Homecoming	(2021)	90 min	Drama, Musical	NaN	NaN	Soumyajit Majumdar	Sayani Gupta	Plabita Borthakur	Ar
3	#Yaaram	(2019)	110 min	Comedy, Romance	4.4	35	Ovais Khan	Prateik	Ishita Raj	Sid K
4	...And Once Again	(2010)	105 min	Drama	NaN	NaN	Amol Palekar	Rajat Kapoor	Rituparna Sengupta	A
5	...Aur Pyaar Ho Gaya	(1997)	147 min	Comedy, Drama, Musical	4.7	827	Rahul Rawail	Bobby Deol	Aishwarya Rai Bachchan	Sh K
6	...Yahaan	(2005)	142 min	Drama, Romance, War	7.4	1,086	Shoojit Sircar	Jimmy Sheirgill	Minissha Lamba	Ya Sh
7	.in for Motion	(2008)	59 min	Documentary	NaN	NaN	Anirban Datta	NaN	NaN	E
8	? : A Question Mark	(2012)	82 min	Horror, Mystery, Thriller	5.6	326	Allyson Patel	Yash Dave	Muntazir Ahmad	
9	@Andheri	(2014)	116 min	Action, Crime, Thriller	4.0	11	Biju Bhaskar Nair	Augustine	Fathima Babu	

In [3]: 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
```

```
In [5]: df.isna().sum()
```

```
Out[5]: 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
```

```
In [6]: nulls = df[df.iloc[:, 1:9].isna().all(axis=1)]
nulls.count()
```

```
Out[6]: Name      8
Year      0
Duration  0
Genre      0
Rating     0
Votes      0
Director   0
Actor 1    0
Actor 2    0
Actor 3    0
dtype: int64
```

```
In [7]: df.dropna(subset=df.columns[1:9], how='all', inplace=True)
```

```
In [8]: df.drop_duplicates(subset = ['Name', 'Year'], keep = 'first', inplace=True)
```

```
In [9]: #removing rows with missing values in column 1,2,4,5 {excluding 'Genre'}
df.dropna(subset=df.columns[[1,2,4,5]], how='all', inplace=True)
```

```
In [10]: #Clean "year" and 'duration' columns
df['Year'] = df['Year'].str.replace(r'[()]', '', regex=True)
df['Duration'] = df['Duration'].str.replace(r' min', '', regex=True)
```

```
In [11]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 15083 entries, 1 to 15508
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Name        15083 non-null   object  
 1   Year         14958 non-null   object  
 2   Duration     7235 non-null   object  
 3   Genre        13244 non-null   object  
 4   Rating       7915 non-null   float64 
 5   Votes        7916 non-null   object  
 6   Director     14587 non-null   object  
 7   Actor 1      13584 non-null   object  
 8   Actor 2      12863 non-null   object  
 9   Actor 3      12148 non-null   object  
dtypes: float64(1), object(9)
memory usage: 1.3+ MB
```

```
In [12]: genres = df["Genre"].str.split(',', expand=True)
genres.head()
```

```
Out[12]:
```

	0	1	2
1	Drama	None	None
2	Drama	Musical	None
3	Comedy	Romance	None
4	Drama	None	None
5	Comedy	Drama	Musical

```
In [13]: df['Votes'] = df['Votes'].str.replace(',', '')
df['Votes'] = pd.to_numeric(df['Votes'], errors='coerce')
```

```
In [14]: df['Duration'] = pd.to_numeric(df['Duration'], errors='coerce')
```

```
In [15]: df['Year'] = pd.to_numeric(df['Year'], errors='coerce')
```

```
In [16]: df.dropna(subset=['Rating'], inplace=True)
```

```
In [17]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7915 entries, 1 to 15508
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Name        7915 non-null    object  
 1   Year         7915 non-null    float64 
 2   Duration     5849 non-null    float64 
 3   Genre        7814 non-null    object  
 4   Rating        7915 non-null    float64 
 5   Votes         7915 non-null    float64 
 6   Director      7910 non-null    object  
 7   Actor 1      7790 non-null    object  
 8   Actor 2      7715 non-null    object  
 9   Actor 3      7624 non-null    object  
dtypes: float64(4), object(6)
memory usage: 680.2+ KB
```

```
In [18]: df['Duration'].fillna(method='ffill', inplace=True)
```

C:\Users\poorv\AppData\Local\Temp\ipykernel_21812\2396974816.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['Duration'].fillna(method='ffill', inplace=True)
C:\Users\poorv\AppData\Local\Temp\ipykernel_21812\2396974816.py:1: FutureWarning: Series.fillna with 'method' is deprecated and will raise in a future version. Use obj.ffmpeg() or obj.bfill() instead.
df['Duration'].fillna(method='ffill', inplace=True)
```

```
In [19]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7915 entries, 1 to 15508
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Name        7915 non-null    object  
 1   Year         7915 non-null    float64 
 2   Duration     7915 non-null    float64 
 3   Genre        7814 non-null    object  
 4   Rating        7915 non-null    float64 
 5   Votes         7915 non-null    float64 
 6   Director      7910 non-null    object  
 7   Actor 1      7790 non-null    object  
 8   Actor 2      7715 non-null    object  
 9   Actor 3      7624 non-null    object  
dtypes: float64(4), object(6)
memory usage: 680.2+ KB
```

```
In [20]: #Randomly sampling non-missing values for 'director', 'Actor 1', "Actor 2", 'Actor 3'
non_missing_directors = df['Director'].dropna()
non_missing_actor1 = df["Actor 1"].dropna()
```

```
non_missing_actor2 = df["Actor 2"].dropna()
non_missing_actor3 = df["Actor 3"].dropna()
```

```
In [21]: df['Director'] = df['Director'].apply(lambda x: np.random.choice(non_missing_directors)
df['Actor 1'] = df['Actor 1'].apply(lambda x: np.random.choice(non_missing_actor1) if
df['Actor 2'] = df['Actor 2'].apply(lambda x: np.random.choice(non_missing_actor2) if
df['Actor 3'] = df['Actor 3'].apply(lambda x: np.random.choice(non_missing_actor3) if
```

```
In [22]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7915 entries, 1 to 15508
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
---  --          -----          ----- 
 0   Name        7915 non-null    object  
 1   Year         7915 non-null    float64 
 2   Duration     7915 non-null    float64 
 3   Genre        7814 non-null    object  
 4   Rating        7915 non-null    float64 
 5   Votes         7915 non-null    float64 
 6   Director      7915 non-null    object  
 7   Actor 1       7915 non-null    object  
 8   Actor 2       7915 non-null    object  
 9   Actor 3       7915 non-null    object  
dtypes: float64(4), object(6)
memory usage: 680.2+ KB
```

```
In [23]: df['Genre'].fillna(df['Genre'].mode()[0], inplace=True)
```

C:\Users\poorv\AppData\Local\Temp\ipykernel_21812\3969028657.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)
```

```
In [24]: df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Index: 7915 entries, 1 to 15508
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   Name        7915 non-null    object  
 1   Year         7915 non-null    float64 
 2   Duration     7915 non-null    float64 
 3   Genre        7915 non-null    object  
 4   Rating       7915 non-null    float64 
 5   Votes        7915 non-null    float64 
 6   Director     7915 non-null    object  
 7   Actor 1      7915 non-null    object  
 8   Actor 2      7915 non-null    object  
 9   Actor 3      7915 non-null    object  
dtypes: float64(4), object(6)
memory usage: 680.2+ KB

```

```
In [25]: top_10_movies = df.loc[df["Rating"].sort_values(ascending=False)[:10].index]
```

	Name	Year	Duration	Genre	Rating	Votes	Director	Actor 1	Actor 2
8339	Love Qubool Hai	2020.0	94.0	Drama, Romance	10.0	5.0	Saif Ali Sayeed	Ahaan Jha	Mahesh Narayan R.
5410	Half Songs	2021.0	79.0	Music, Romance	9.7	7.0	Sriram Raja	Raj Banerjee	Emon Chatterjee Pu
2563	Breed	2020.0	82.0	Drama	9.6	48.0	Bobby Kumar	Bobby Kumar	Ashfaq
11704	Ram-Path	2017.0	150.0	Documentary	9.4	5.0	Ashish Dubey	Ishan Jacob	Rishi Kapoor Gh
5077	Gho Gho Rani	2019.0	105.0	History, Romance	9.4	47.0	Munni Pankaj	Nishi Neha Mishra	Pankaj Kamal
6852	June	2021.0	93.0	Drama	9.4	18.0	Suhrud Godbole	Vaibhav Khisti	Nilesh Divekar
14222	The Reluctant Crime	2020.0	113.0	Drama	9.4	16.0	Arvind Pratap	Dharmendra Ahir	Awanish Kotnal
12673	Secrets of Sinauli	2021.0	56.0	Documentary, History	9.3	1373.0	Raghav Jairath	Manoj Bajpayee	R.S. Bhist
1729	Baikunth	2021.0	72.0	Family	9.3	29.0	Vishwa Bhanu	Vishwa Bhanu	Sangam Shukla
8344	Love Sorries	2021.0	101.0	Comedy, Drama, Romance	9.3	79.0	Gautam Joshi	Prashant Chaubey	Puneet Chouksey

```
In [26]: #Top rated movies of each year
df.groupby('Year').apply(pd.DataFrame.nlargest, n=1, columns=["Rating"])
```

```
C:\Users\poorv\AppData\Local\Temp\ipykernel_21812\4144860762.py:2: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.
  df.groupby('Year').apply(pd.DataFrame.nlargest, n=1, columns=["Rating"])
```

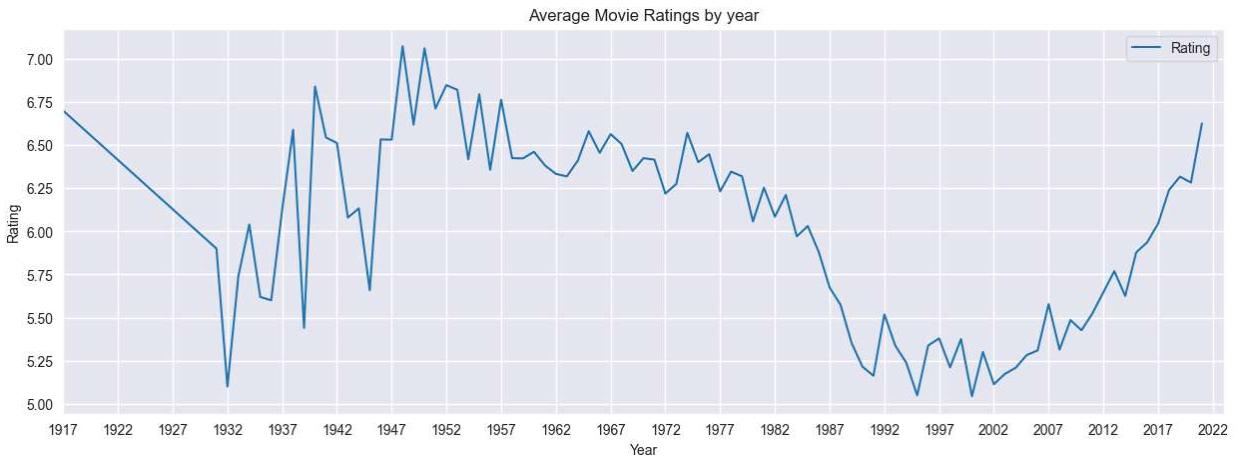
Out[26]:

		Name	Year	Duration	Genre	Rating	Votes	Director	Actor 1	Actor 2
Year										
1917.0	8146	Lanka Dahan	1917.0	107.0	Adventure, Drama, Fantasy	6.7	22.0	Dhundiraj Govind Phalke	Anna Salunke	Gang Sr
1931.0	7203	Kalidas	1931.0	143.0	Drama, History, Musical	6.2	12.0	H.M. Reddy	T.P. Rajalakshmi	Venkat
1932.0	2183	Bharati Mata	1932.0	170.0	Drama	6.1	12.0	Pesi Karani	Hadi Jams	
1933.0	851	Alif Laila	1933.0	120.0	Fantasy	7.2	37.0	Balwant Bhatt	Shanti Dave	B C
1934.0	9053	Mazdoor	1934.0	155.0	Drama	8.5	6.0	Mohan Dayaram Bhavnani	Bibbo Nayan	
...
2017.0	11704	Ram-Path	2017.0	150.0	Documentary	9.4	5.0	Ashish Dubey	Ishan Jacob	Ka
2018.0	1314	Ashok Vatika	2018.0	97.0	Drama	9.3	7.0	Rahul Mallick	Kunj Anand	S B
2019.0	5077	Gho Rani	2019.0	105.0	History, Romance	9.4	47.0	Munni Pankaj	Nishi Neha Mishra	P K
2020.0	8339	Love Qubool Hai	2020.0	94.0	Drama, Romance	10.0	5.0	Saif Ali Sayeed	Ahaan Jha	Ma Nar
2021.0	5410	Half Songs	2021.0	79.0	Music, Romance	9.7	7.0	Sriram Raja	Raj Banerjee	E Chatt

92 rows × 10 columns

In [27]:

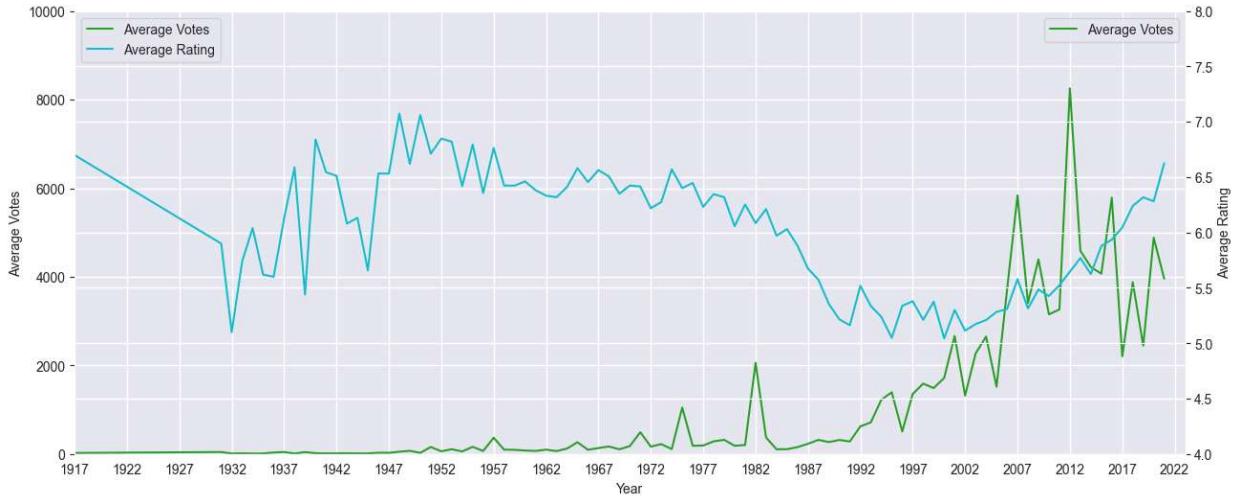
```
#Graph shows average rating for every year and according to it the year with best rating
sns.set_style("darkgrid")
df.groupby('Year')[['Rating']].mean().plot(figsize=(15,5))
plt.xlabel("Year")
plt.ylabel("Rating")
plt.title("Average Movie Ratings by year")
plt.xticks(np.arange(1917,2023,5))
plt.xlim(1917,2023)
plt.show()
```



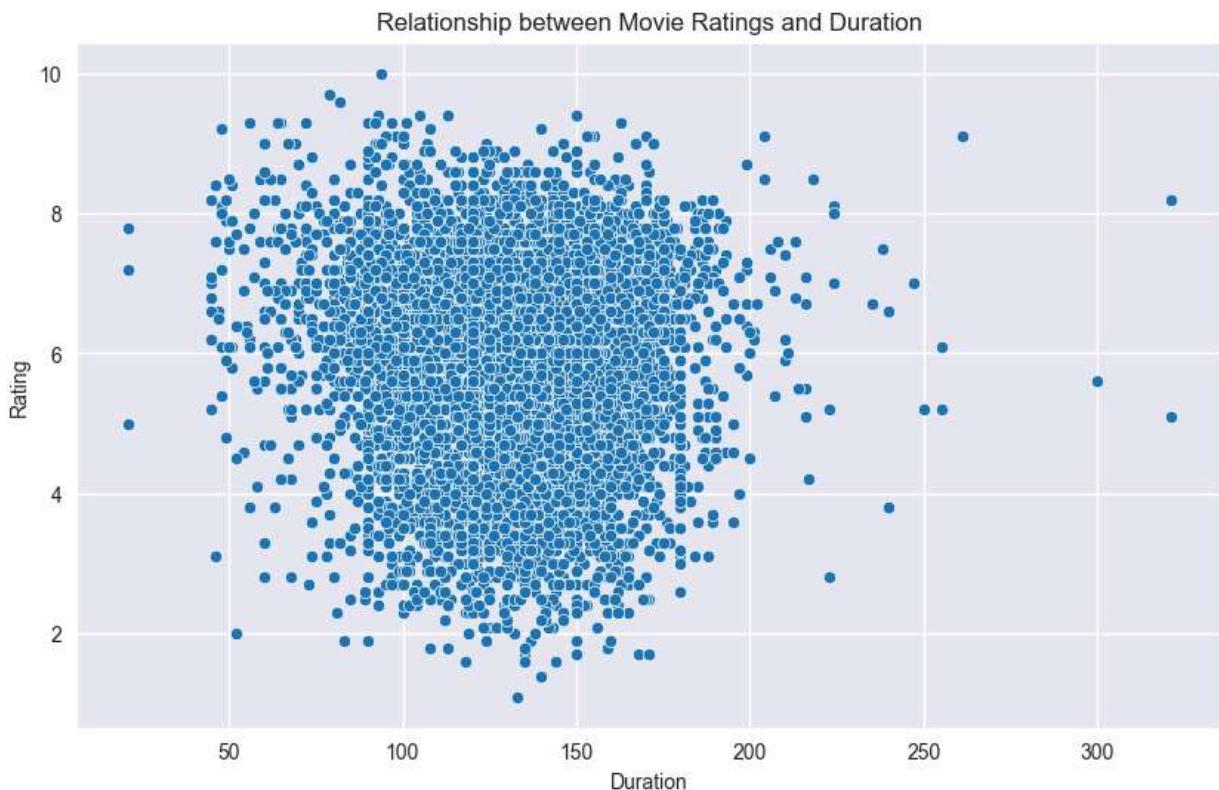
```
In [28]: #Average Votes VS Year
#Average Ratings VS Year
fig,ax1 = plt.subplots(figsize=(15,6))
sns.lineplot(data=df,x='Year',y='Votes', errorbar=None, ax=ax1, label='Average Votes',
ax1.set_xlabel('Year')
ax1.set_ylabel('Average Votes')
ax1.set_xlim(1917,2023)
ax1.set_ylim(0,10000)
ax1.set_xticks(np.arange(1917,2023,5))

ax2 = ax1.twinx()
sns.lineplot(data=df,x='Year',y='Rating', errorbar=None, ax=ax2, label='Average Rating')
ax2.set_ylabel('Average Rating')
ax2.set_ylim(4,8)
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, loc ='upper left')
plt.show()
```

```
C:\Users\poorv\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
    with pd.option_context('mode.use_inf_as_na', True):
C:\Users\poorv\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
    with pd.option_context('mode.use_inf_as_na', True):
C:\Users\poorv\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
    with pd.option_context('mode.use_inf_as_na', True):
C:\Users\poorv\anaconda3\lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: u
se_inf_as_na option is deprecated and will be removed in a future version. Convert in
f values to NaN before operating instead.
    with pd.option_context('mode.use_inf_as_na', True):
```



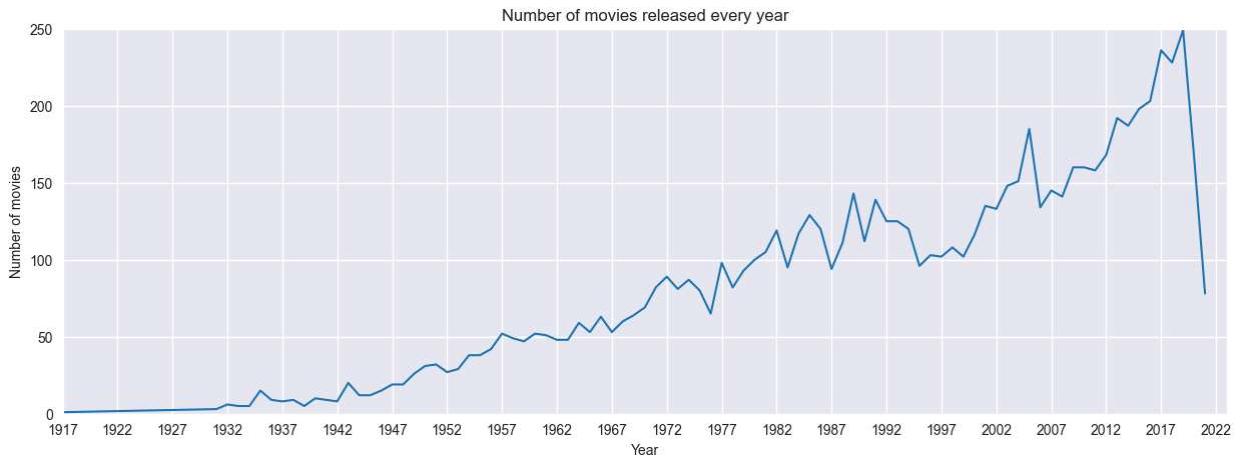
```
In [29]: plt.figure(figsize=(10,6))
sns.scatterplot(x='Duration', y='Rating', data=df)
plt.xlabel('Duration')
plt.ylabel('Rating')
plt.title('Relationship between Movie Ratings and Duration')
plt.show()
```



```
In [30]: #Most of the movies duration Lie between 50 min to 200 min and highest rated movie is
```

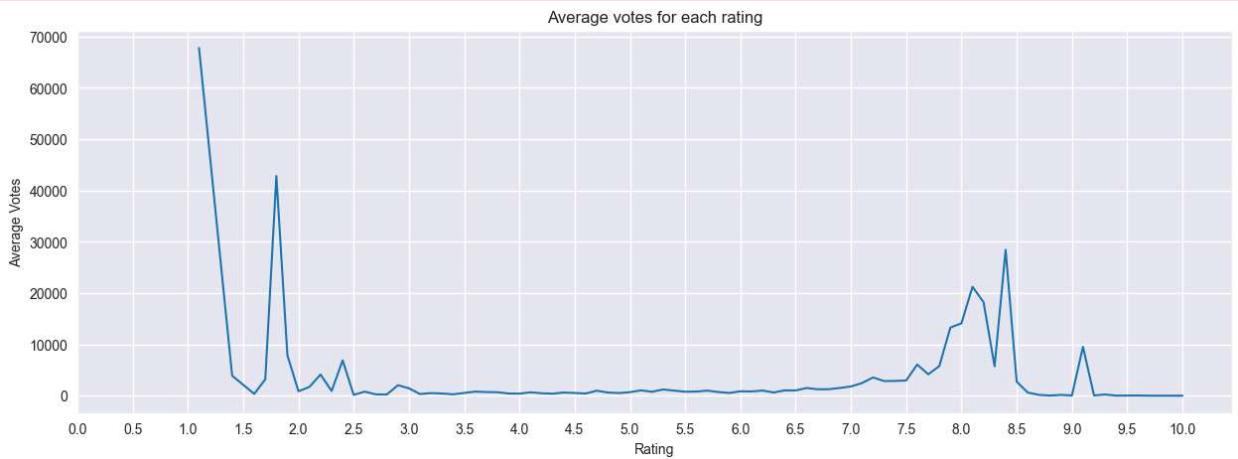
```
In [31]: sns.set_style('darkgrid')
df.groupby(['Year'])['Name'].count().plot(figsize=(15,5))
plt.xlabel('Year')
plt.ylabel('Number of movies')
plt.title('Number of movies released every year')
plt.ylim(0,250)
plt.xlim(1917, 2023)
```

```
plt.xticks(np.arange(1917, 2023, 5))
plt.show()
```



```
In [32]: sns.set_style('darkgrid')
plt.figure(figsize=(15,5))
sns.lineplot(data=df, x='Rating', y='Votes', errorbar=None)
plt.xlabel('Rating')
plt.ylabel('Average Votes')
plt.xticks(np.arange(0,10.5,0.5))
plt.title('Average votes for each rating')
plt.show()
```

C:\Users\poorv\anaconda3\lib\site-packages\seaborn_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
with pd.option_context('mode.use_inf_as_na', True):
C:\Users\poorv\anaconda3\lib\site-packages\seaborn_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
with pd.option_context('mode.use_inf_as_na', True):



```
In [33]: #above graph shows the average votes for each ratings and we can see the movie rated 9.5-1
```

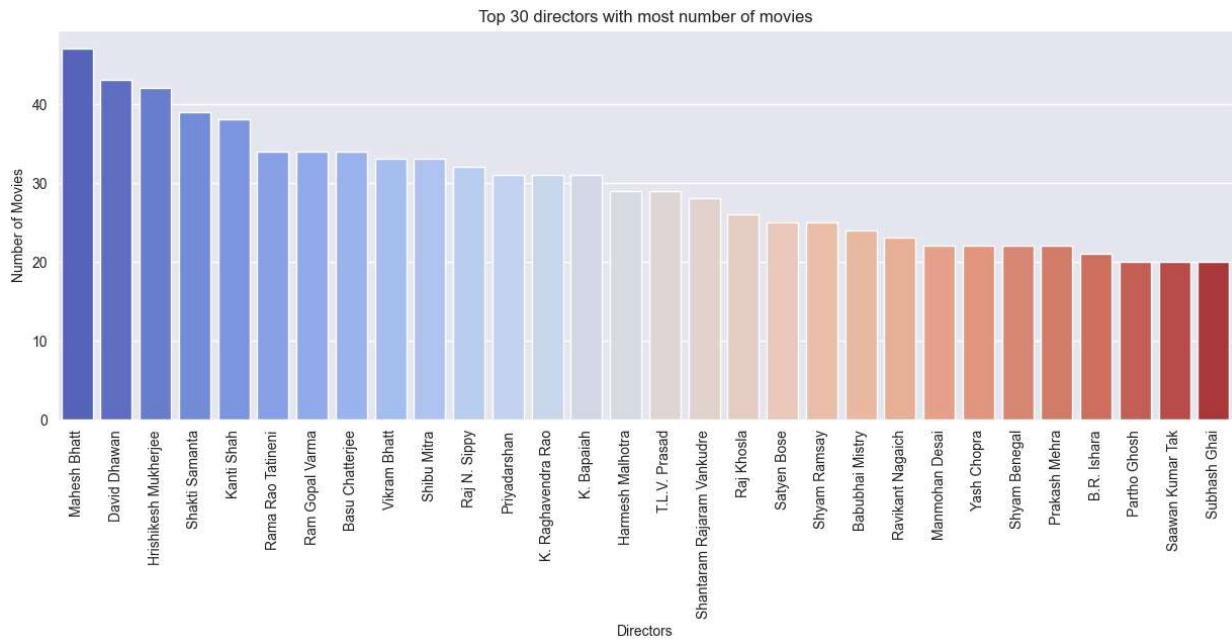
```
In [34]: directors = df.groupby('Director').agg({'Rating':[ 'mean', 'count']})
directors.columns = directors.columns.droplevel(0)
directors.reset_index(inplace=True)
directors.columns =['Director', 'Average Rating', 'Movie count']
directors['Average Rating'] = directors['Average Rating'].round(1)
directors.sort_values(by='Movie count', ascending=False, inplace=True)
directors.head()
```

Out[34]:

	Director	Average Rating	Movie count
1422	Mahesh Bhatt	5.4	47
629	David Dhawan	5.2	43
962	Hrishikesh Mukherjee	7.1	42
2562	Shakti Samanta	6.6	39
1233	Kanti Shah	5.0	38

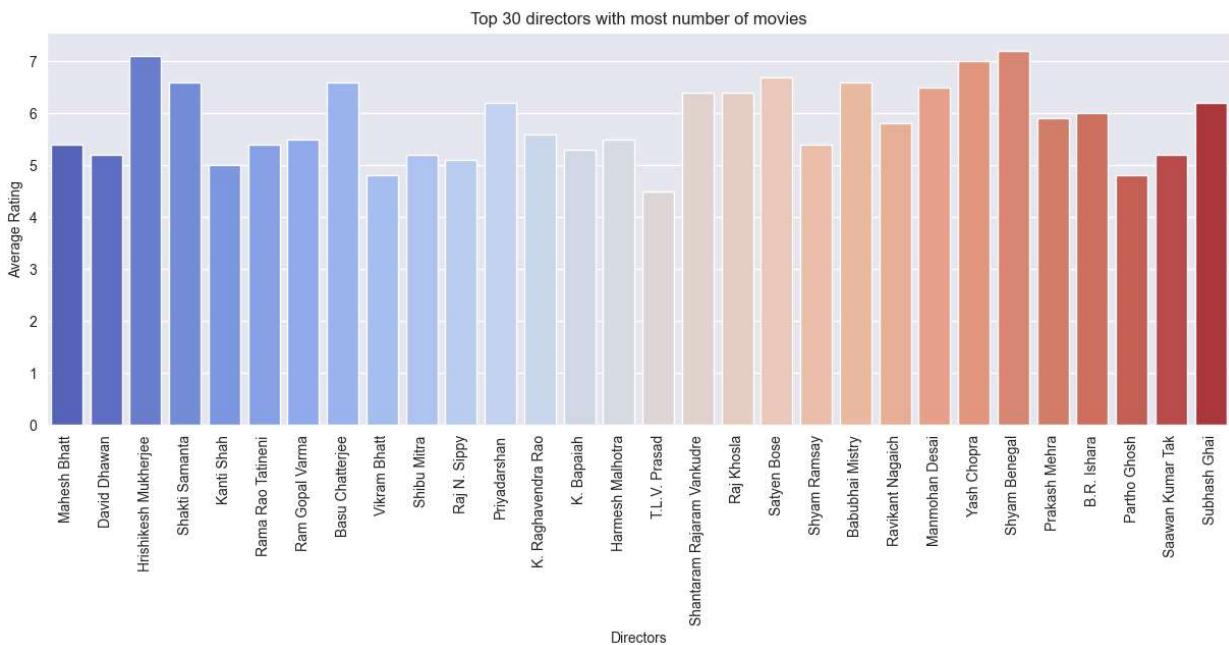
In [35]:

```
directors_dict = dict(zip(directors['Director'], directors['Average Rating']))
plt.figure(figsize=(15,5))
sns.set_style('darkgrid')
sns.barplot(data=directors.head(30), x='Director', y='Movie count', palette='coolwarm'
plt.xlabel('Directors')
plt.ylabel('Number of Movies')
plt.xticks(rotation=90)
plt.title('Top 30 directors with most number of movies')
plt.show()
```



In [36]:

```
plt.figure(figsize=(15,5))
sns.set_style('darkgrid')
sns.barplot(data=directors.head(30), x='Director', y='Average Rating', palette='coolwarm'
plt.xlabel('Directors')
plt.ylabel('Average Rating')
plt.xticks(rotation=90)
plt.title('Top 30 directors with most number of movies')
plt.show()
```



```
In [37]: df_melted = df.melt(id_vars='Rating', value_name='actor', var_name='role', value_vars=[],
actor_scores = df_melted.groupby('actor')['Rating'].agg(['mean', 'count'])
actor_scores.reset_index(inplace=True)
actor_scores.columns =['Actor', 'Average Score', 'Number of Movies']
actor_scores.sort_values('Number of Movies', ascending=False, inplace=True)
actor_scores['Average Score']=actor_scores['Average Score'].round(1)
actor_scores
```

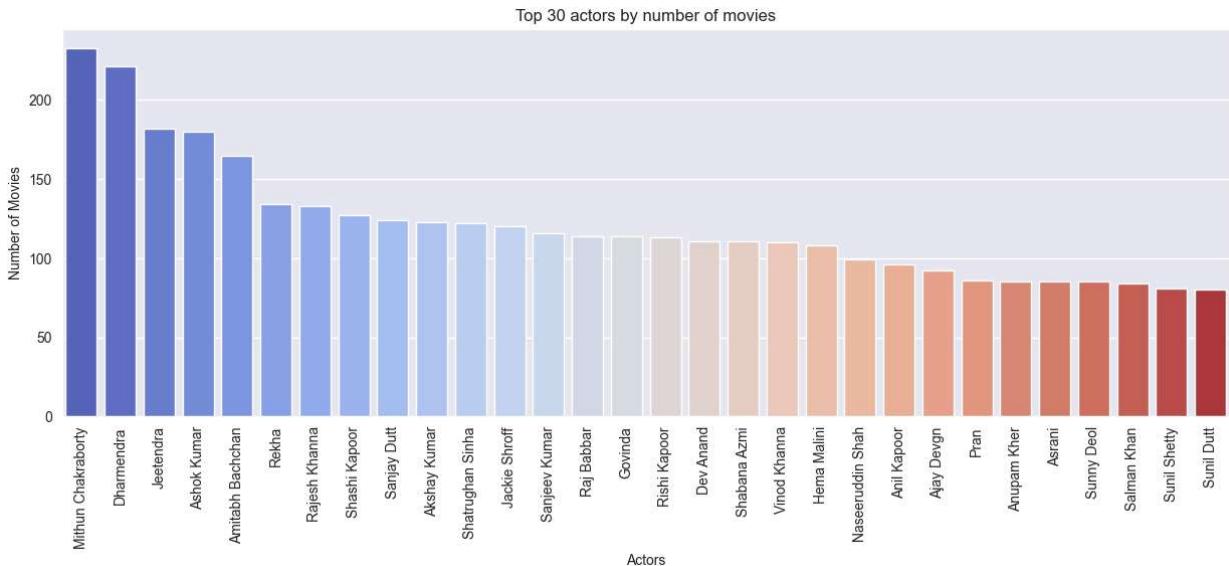
Out[37]:

	Actor	Average Score	Number of Movies
3094	Mithun Chakraborty	5.4	233
1441	Dharmendra	5.8	221
2164	Jeetendra	5.4	182
844	Ashok Kumar	6.4	180
440	Amitabh Bachchan	6.2	165
...
2456	Kevin Mukherji	7.0	1
2455	Kev Clinsc	5.3	1
2453	Ketan Kava	8.3	1
2451	Keshto Iqbal	4.0	1
3075	Mini Bansal	5.2	1

6151 rows × 3 columns

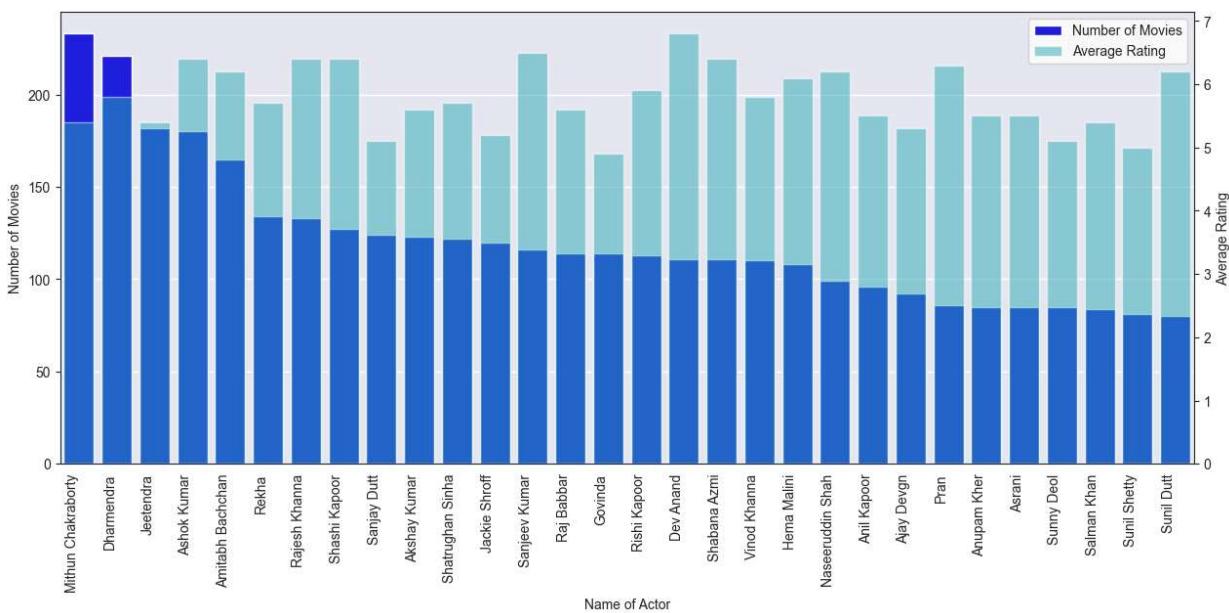
```
In [38]: actor_score_dict = dict(zip(actor_scores['Actor'], actor_scores['Average Score']))
plt.figure(figsize=(15,5))
sns.barplot(data=actor_scores[:30], x='Actor', y='Number of Movies', dodge=False, palette='viridis')
plt.xticks(rotation=90)
plt.xlabel('Actors')
plt.ylabel('Number of Movies')
```

```
plt.title('Top 30 actors by number of movies')
plt.show()
```



In [39]: *#Along with Average Ratings*

```
fig,ax1 = plt.subplots(figsize=(15,6))
sns.set_style('white')
sns.barplot(data=actor_scores[:30], x='Actor', y='Number of Movies',dodge=True, ax=ax1)
ax1.set(xlabel='Name of Actor', ylabel='Number of Movies')
ax1.set_xticklabels(ax1.get_xticklabels(), rotation=90, ha='right')
ax2 = ax1.twinx()
sns.barplot(data=actor_scores[:30],x='Actor',y='Average Score',dodge=True,ax=ax2,label=True)
ax2.set_ylabel('Average Rating')
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, loc ='upper right')
plt.show()
```



In [40]: *#Feature Engineering*

In [41]: *#Grouping the column with their average rating 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']=actor1_mean_rating

actor3_mean_rating = df.groupby('Actor 3')['Rating'].transform('mean')
df['Actor3_encoded']=actor1_mean_rating
```

In [42]: *#Keeping the predictor and target variable*
x = df[['Year', 'Votes', 'Duration', 'Genre_mean_rating', 'Director_encoded', 'Actor1_encoded', 'Actor2_encoded', 'Actor3_encoded']]
y = df['Rating']

In [43]: *#training and testing the dataset*
from sklearn.model_selection import train_test_split, cross_val_score
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

In [44]: from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score as score
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR

In [45]: def evaluate_model(y_true, y_pred, model_name):
 print('Model:', model_name)
 print('Accuracy ={:0.2f}%'.format(score(y_true, y_pred)*100))
 print('Mean Squared Error = {:0.2f}\n'.format(mean_squared_error(y_true, y_pred, s))
 return round(score(y_true, y_pred)*100,2)

In [46]: *#Building Machine Learning Models and training them using fit()*

In [47]: LR = LinearRegression()
LR.fit(X_train, y_train)

Out[47]: ▾ LinearRegression
LinearRegression()

In [48]: lr_pred = LR.predict(X_test)
lr_pred

Out[48]: array([3.02687432, 6.90713711, 5.07412023, ..., 8.35817032, 5.24290477, 4.83823549])

In [49]: RFR = RandomForestRegressor(n_estimators=100, random_state=1)
RFR.fit(X_train, y_train)

Out[49]: ▾ RandomForestRegressor
RandomForestRegressor(random_state=1)

```
In [50]: r_pred = RFR.predict(X_test)
r_pred
```

```
Out[50]: array([3.417, 7.021, 4.727, ..., 8.098, 5.639, 4.285])
```

```
In [51]: DTR = DecisionTreeRegressor(random_state=1)
DTR.fit(X_train, y_train)
```

```
Out[51]: ▾ DecisionTreeRegressor
```

```
DecisionTreeRegressor(random_state=1)
```

```
In [52]: dt_pred = DTR.predict(X_test)
dt_pred
```

```
Out[52]: array([3.3, 7.2, 4.7, ..., 8.1, 5.6, 5.1])
```

```
In [53]: GBR = GradientBoostingRegressor(n_estimators=100, random_state=60)
GBR.fit(X_train, y_train)
```

```
Out[53]: ▾ GradientBoostingRegressor
```

```
GradientBoostingRegressor(random_state=60)
```

```
In [54]: g_pred = GBR.predict(X_test)
g_pred
```

```
Out[54]: array([3.31672877, 7.01643421, 4.91543892, ..., 8.33709339, 5.36558714,
   4.72788478])
```

```
In [55]: KNR = KNeighborsRegressor(n_neighbors=5)
KNR.fit(X_train, y_train)
```

```
Out[55]: ▾ KNeighborsRegressor
```

```
KNeighborsRegressor()
```

```
In [56]: knn_pred = KNR.predict(X_test)
knn_pred
```

```
Out[56]: array([4.3 , 5.86, 6.22, ..., 7.06, 4.8 , 5.62])
```

```
In [57]: LRscore = evaluate_model(y_test, lr_pred, 'Linear Regression')
RFscore = evaluate_model(y_test, r_pred, 'Random Forest')
DTscore = evaluate_model(y_test, dt_pred, 'Decision Tree')
GBscore = evaluate_model(y_test, g_pred, 'Gradient Boosting')
KNNscore = evaluate_model(y_test, knn_pred, 'K Nearest Neighbors')
```

```
Model: Linear Regression  
Accuracy =67.51%  
Mean Squared Error = 0.77
```

```
Model: Random Forest  
Accuracy =72.62%  
Mean Squared Error = 0.70
```

```
Model: Decision Tree  
Accuracy =46.07%  
Mean Squared Error = 0.99
```

```
Model: Gradient Boosting  
Accuracy =72.70%  
Mean Squared Error = 0.70
```

```
Model: K Nearest Neighbors  
Accuracy =25.78%  
Mean Squared Error = 1.16
```

```
In [58]: #Random forest is best model
```

```
In [59]: x.head()
```

```
Out[59]:
```

	Year	Votes	Duration	Genre_mean_rating	Director_encoded	Actor1_encoded	Actor2_encoded
1	2019.0	8.0	109.0	6.323944	7.000000	6.850000	6.850000
3	2019.0	35.0	110.0	5.722500	4.400000	5.420000	5.420000
5	1997.0	827.0	147.0	6.224490	5.358824	4.788889	4.788889
6	2005.0	1086.0	142.0	6.820000	7.500000	5.465385	5.465385
8	2012.0	326.0	82.0	5.463636	5.600000	5.600000	5.600000

```
In [60]: y.head()
```

```
Out[60]:
```

1	7.0
3	4.4
5	4.7
6	7.4
8	5.6

```
Name: Rating, dtype: float64
```

```
In [61]: #Predicting value using 5th row
```

```
In [62]:
```

```
data = {  
    'Year': [1997],  
    'Votes': [827],  
    'Duration': [147],  
    'Genre_mean_rating': [6.224490],  
    'Director_encoded': [5.358824],  
    'Actor1_encoded': [4.788889],  
    'Actor2_encoded': [4.788889],  
    'Actor3_encoded': [4.788889]  
}
```

```
DF = pd.DataFrame(data)

pred_rating = RFR.predict(DF)
print("Predicted Rating: ", pred_rating[0])
```

Predicted Rating: 4.90299999999995

In []:

In []:

In []:

In []:

In []: