NAME: SAHIR DEV Date: 10-05-2025

IMDB SCORE PREDICTION

END-TO-END DATA SCIENCE PROJECT

AGENDA:

- Introduction & Objectives
- Data Collection & Cleaning
- Exploratory Data Analysis (EDA)
- Feature Engineering
- Modeling & Evaluation
- Model Interpretability & Insights
- Conclusions

Introduction & Objectives

1. Problem Statement:

-Predict IMDb ratings to help understand movie success.

2. Objectives:

-Build and tune a robust predictive model.

-Identify key factors (features) influencing movie ratings.

3. Implications:

-Inform content strategy, marketing, or production decisions.

Data Collection & Cleaning

1. Data Source:

- Details about the IMDb dataset (movies, ratings, cast, etc.).

2. Data Cleaning Steps:

- Remove unneccesary columns.
- Handling missing values and outliers.
- Data type conversion.

3. Outcome:

- Cleaned dataset ready for analysis.

[22]:
```
df.isnull().sum()
```

[22]:
```
Poster_Link         0
Series_Title        0
Released_Year       0
Certificate       101
Runtime             0
Genre               0
IMDB_Rating         0
Overview            0
Meta_score        157
Director            0
Star1               0
Star2               0
Star3               0
Star4               0
No_of_Votes         0
Gross             169
dtype: int64
```

[23]:
```
## there are some missing values in the dataset.
```

[34]:
```
mod=df['Certificate'].mode()
df['Certificate']=df['Certificate'].fillna('U') # most certificates are of 'U'
```

[36]:
```
mean=df['Meta_score'].mean()
df['Meta_score']=df['Meta_score'].fillna(mean)
```

[37]:
```
df.Gross=df['Gross'].str.replace(',','')
```

[38]:
```
df.Gross=df.Gross.replace(np.nan,0)
df.Gross=df.Gross.astype(int)
```

[39]:
```
df.Gross=df.Gross.replace(0,df.Gross.mean())
```

CONTINUE:

There are some unnecessary columns so we should remove those columns

```
[42]: df= df.drop(columns=['Poster_Link','Series_Title','Overview'],axis=1)
```

```
[44]: df.Runtime= df.Runtime.str.extract('([^\s]+)')
      df["Runtime"] = df["Runtime"][~(df["Runtime"] == "min")]
      df["Runtime"] = df["Runtime"].astype(int)

[45]: df=df.drop(labels=np.where(df.Released_Year=='PG')[0][0],axis=0)

[46]: df.Released_Year= df.Released_Year.astype(int)
```

```
[55]: df.isnull().sum()

[55]: Released_Year    0
      Certificate      0
      Runtime          0
      Genre            0
      IMDB_Rating      0
      Meta_score       0
      Director         0
      Star1            0
      Star2            0
      Star3            0
      Star4            0
      No_of_Votes      0
      Gross            0
      dtype: int64
```

Exploratory Data Analysis (EDA)

1. Visualization Highlights:

- Distribution of IMDb Ratings, Meta Scores, etc.
-Distribution of every single Numeric columns(Features).
-Correlation and Heatmap for numeric columns(Features).
-Barplots for categorical columns.

2. Key Findings:

- Relationship between critical reception (Meta_score and No. of Votes) and IMDb ratings.
- Patterns in genre, runtime, and other variables.

[13]:
```
numeric_columns= df.select_dtypes(np.number).columns.tolist()
category_columns= ['Certificate','Genre','Director']
stars_col= ['Star1','Star2','Star3','Star4']
```

[14]:
```
df.shape
```

[14]:
```
(999, 13)
```

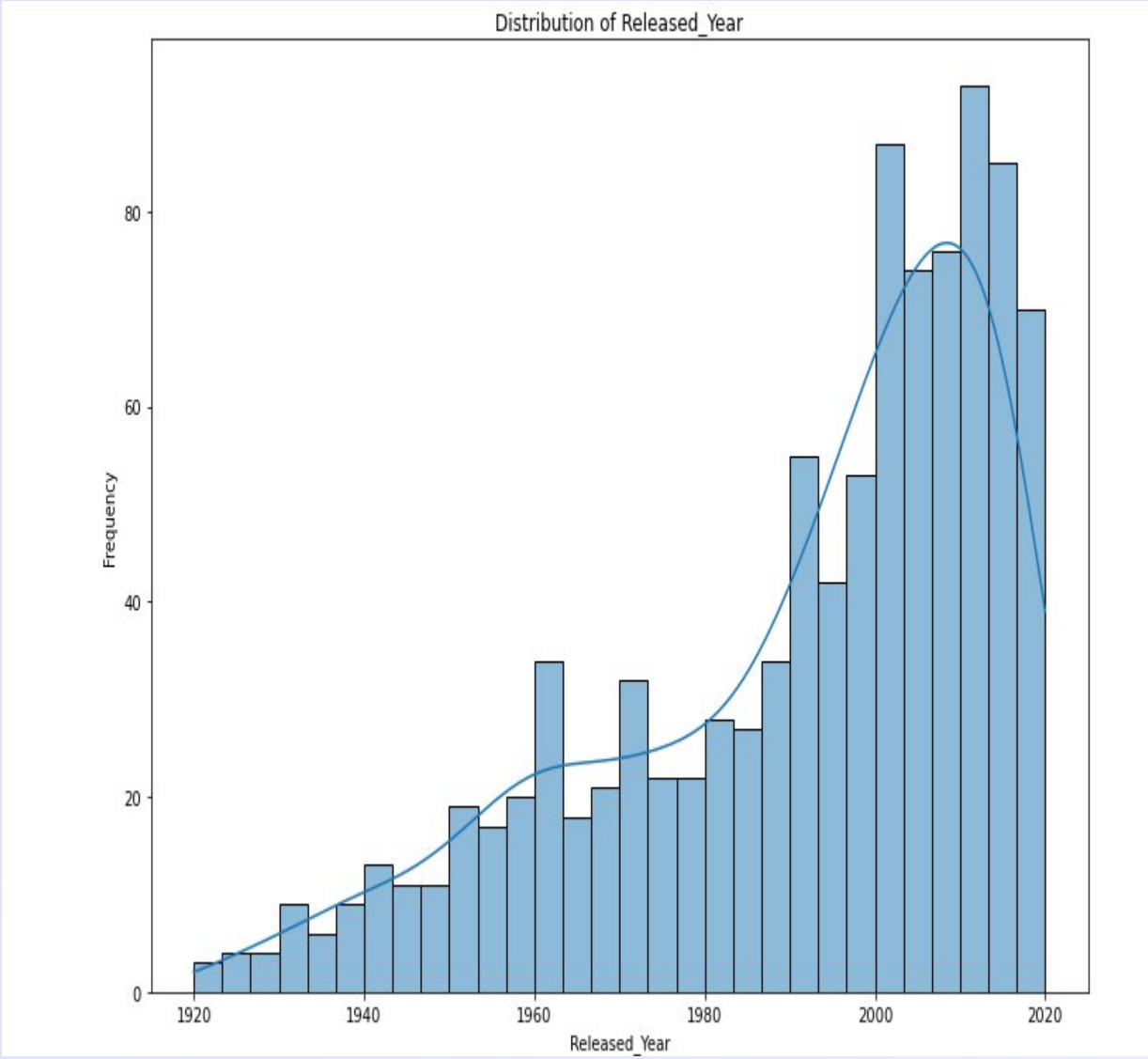
[16]:
```
# for every single numeric attribute distribution
```

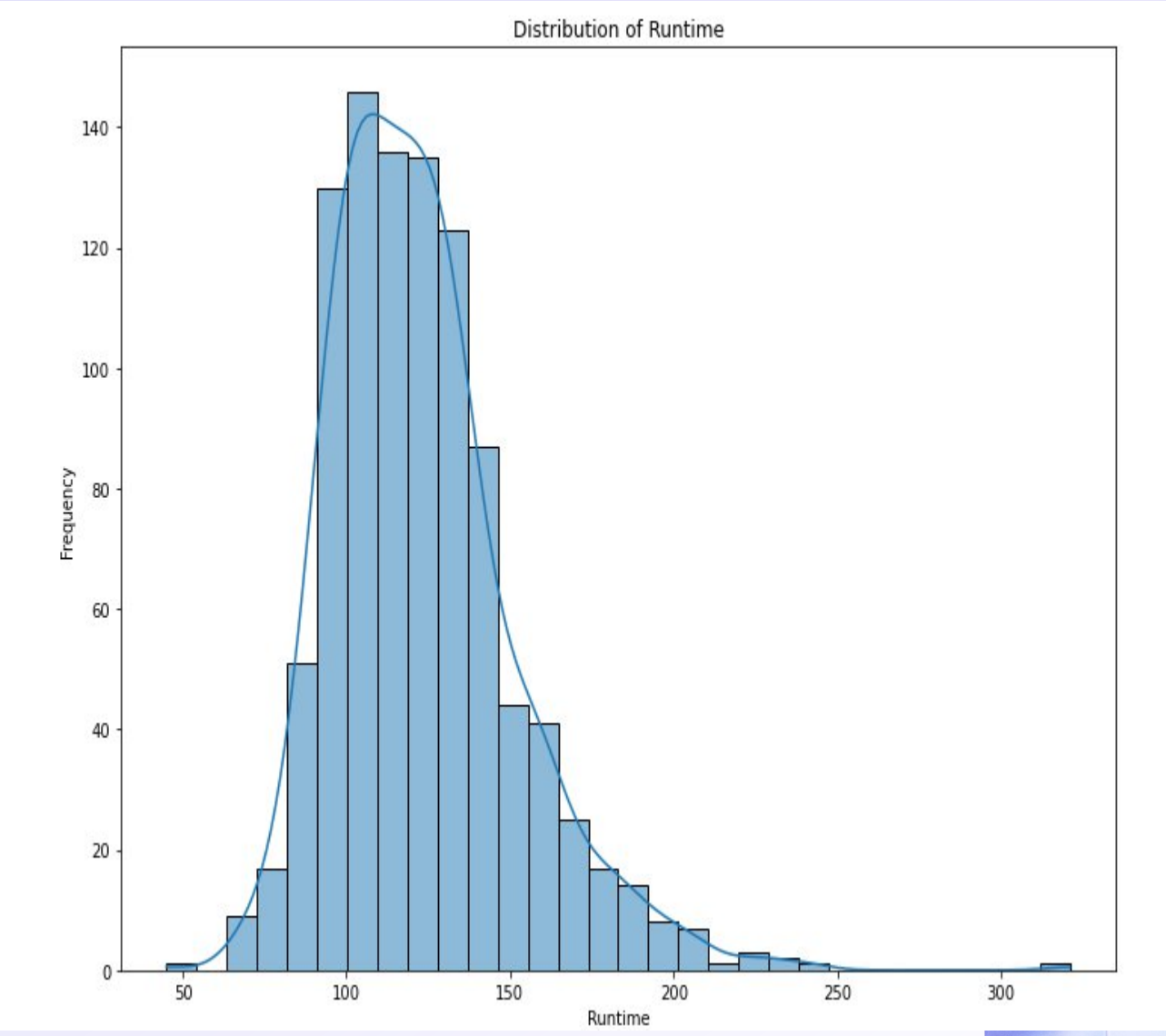
[19]:
```
for col in numeric_columns:
    plt.figure(figsize=(10,8))
    sns.histplot(df[col],kde=True,bins=30)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.savefig(f'{col}_Distribution.png')
    plt.tight_layout()
    plt.show()
```

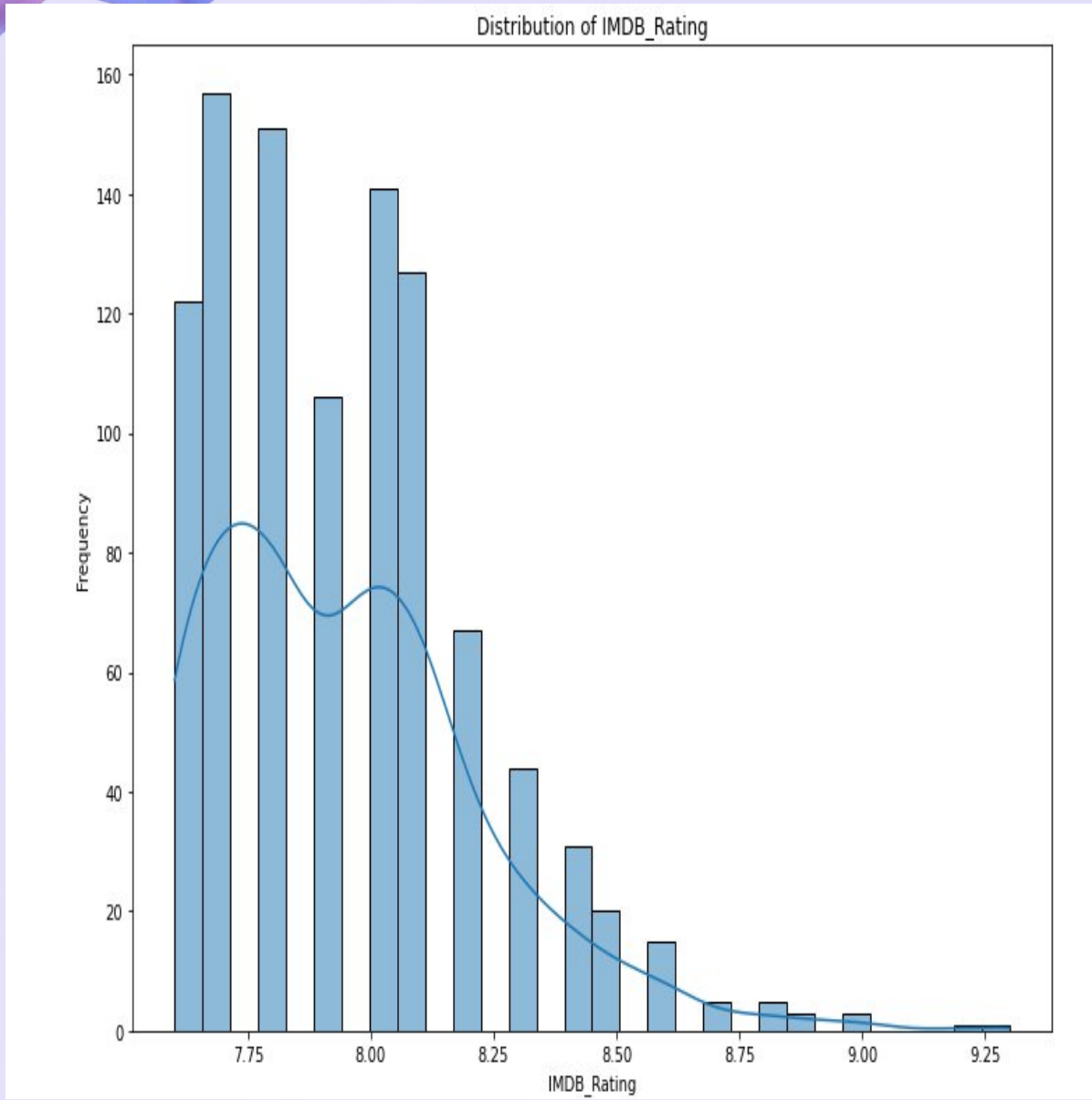
[62]:
```
# for categorical cols
```

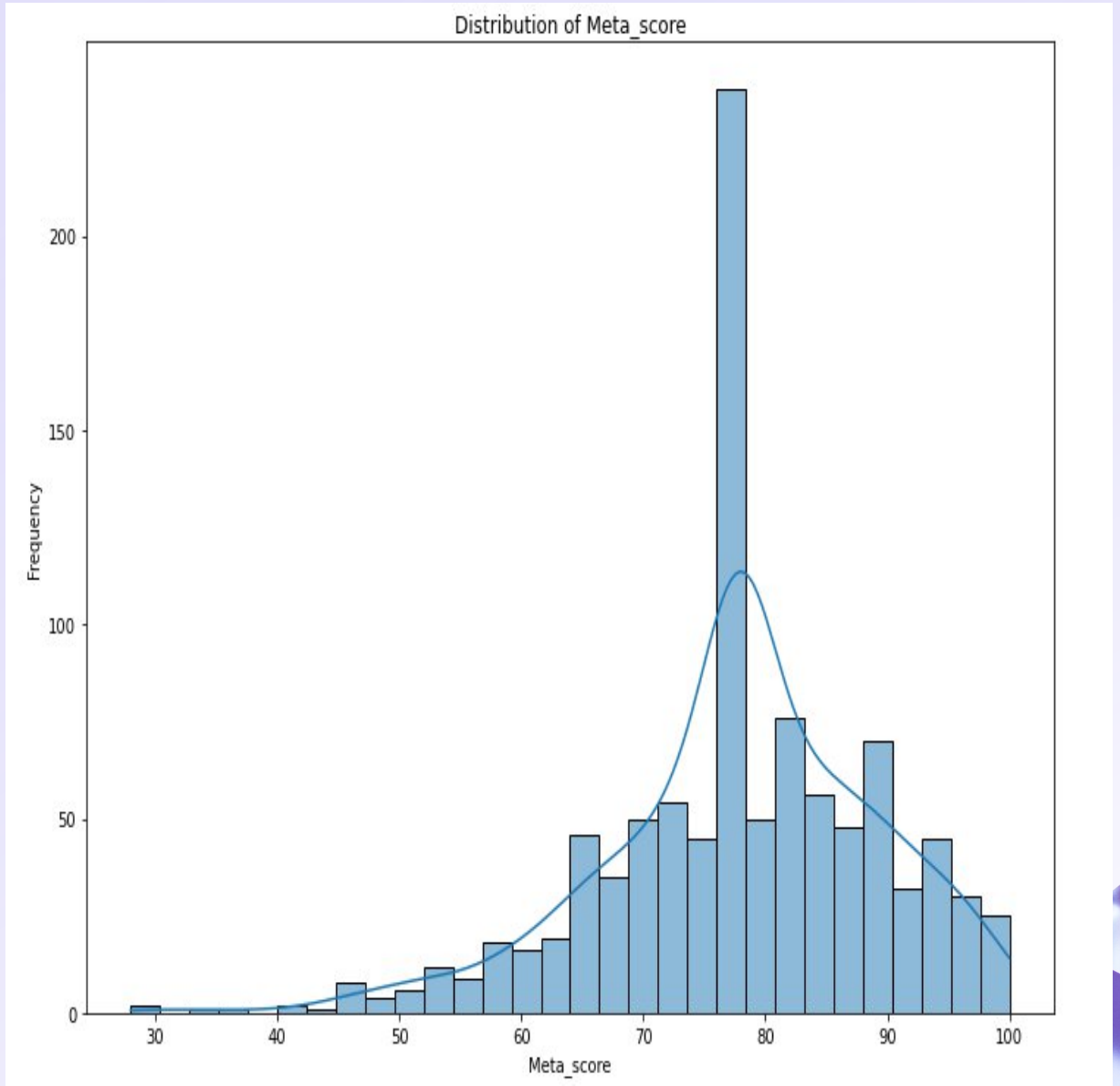
[26]:
```
for cat in category_columns:
    plt.figure(figsize=(12,8))
    dat= df[cat].value_counts().nlargest(10)
    sns.barplot(x=dat.index,y=dat.values,palette='Set2')
    plt.title(f"Top 10 {cat}")
    plt.xlabel(cat)
    plt.ylabel('Counts')
    plt.xticks(rotation=15)
    plt.savefig(f'{cat}_Countplot')
    plt.tight_layout()
    plt.show()
```

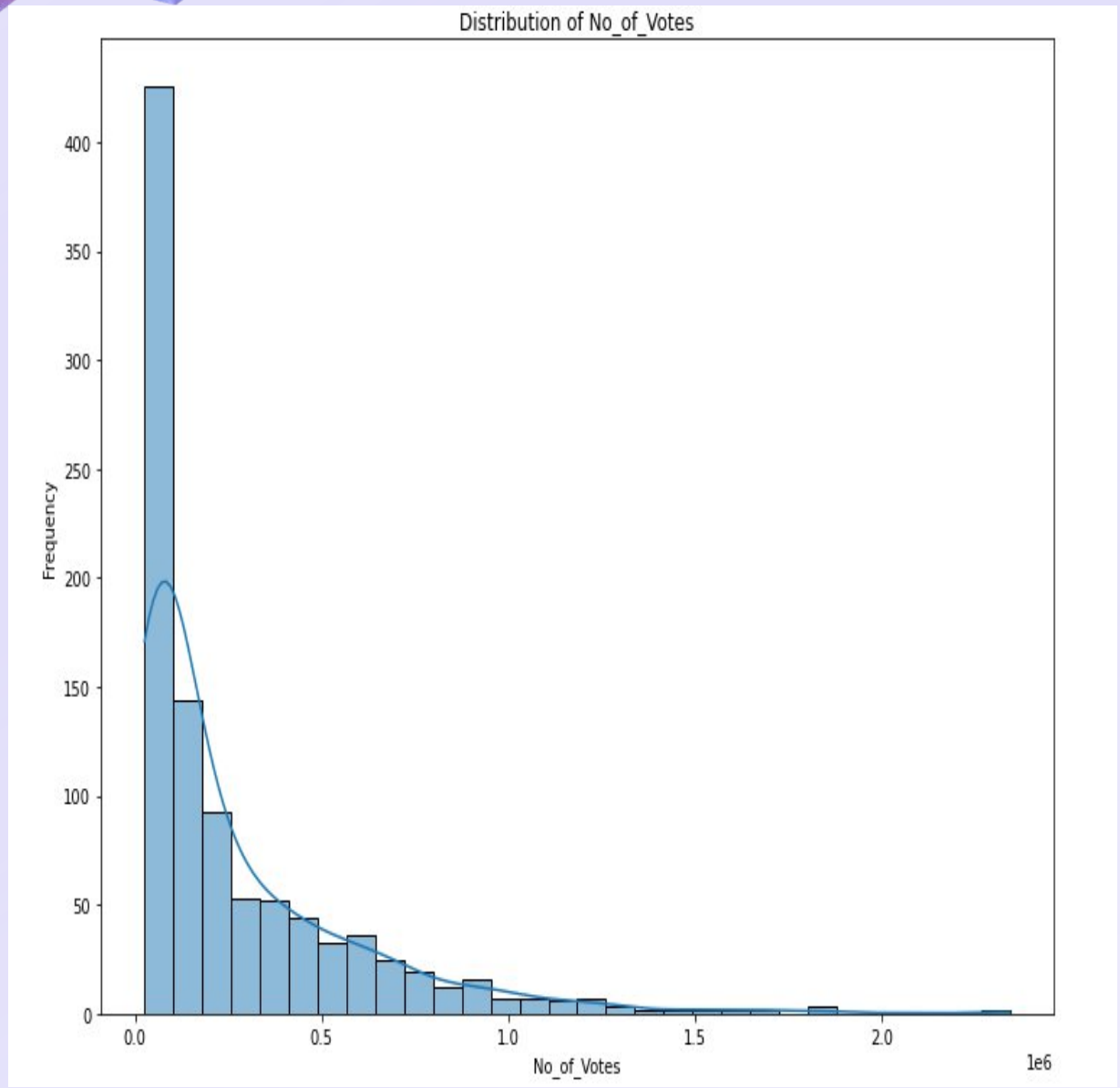
Distribution of every numeric attribute(Feature)

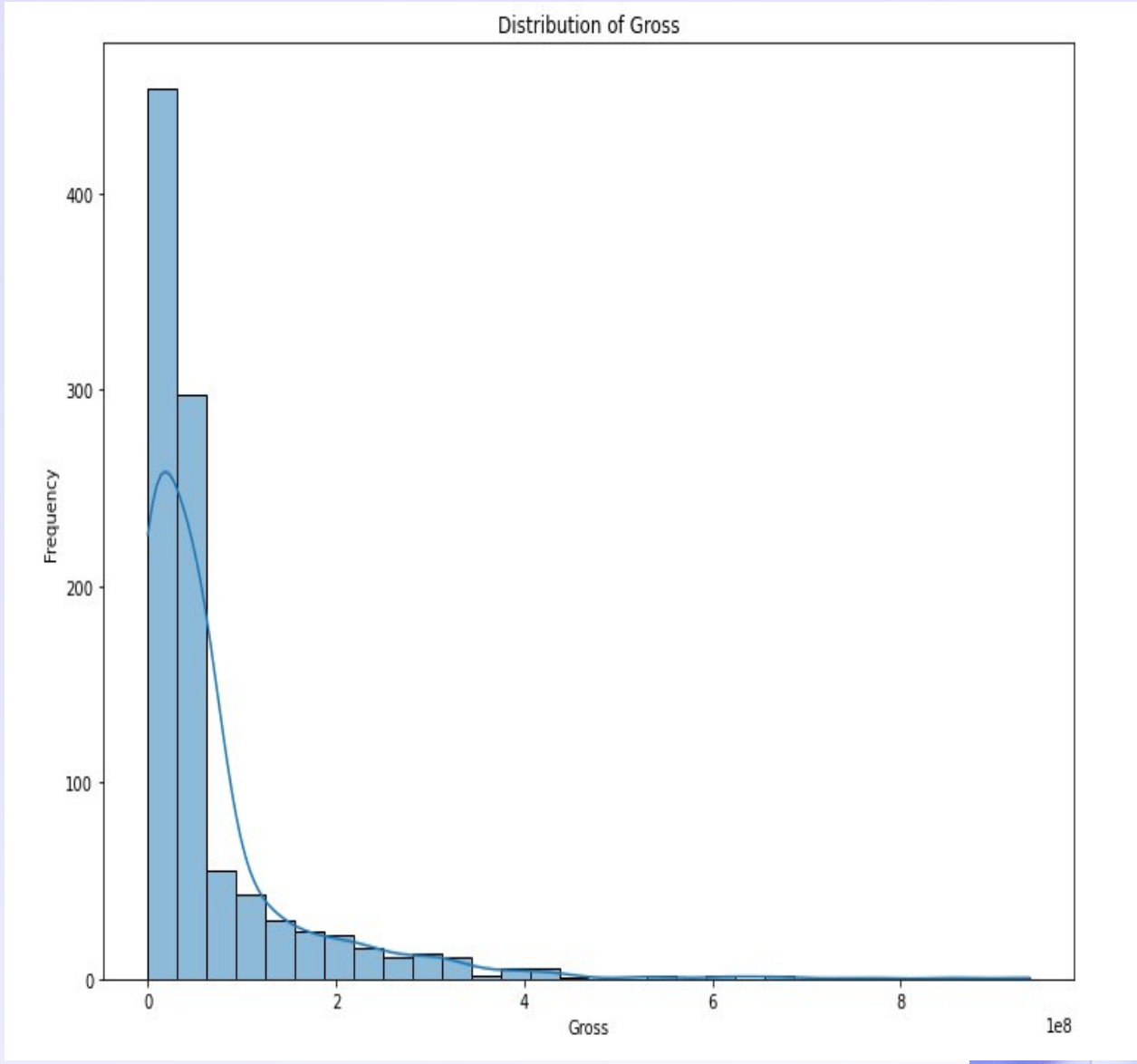





Distribution of IMDB_Rating
Frequency
IMDB_Rating
0
20
40
60
80
100
120
140
160
7.75
8.00
8.25
8.50
8.75
9.00
9.25

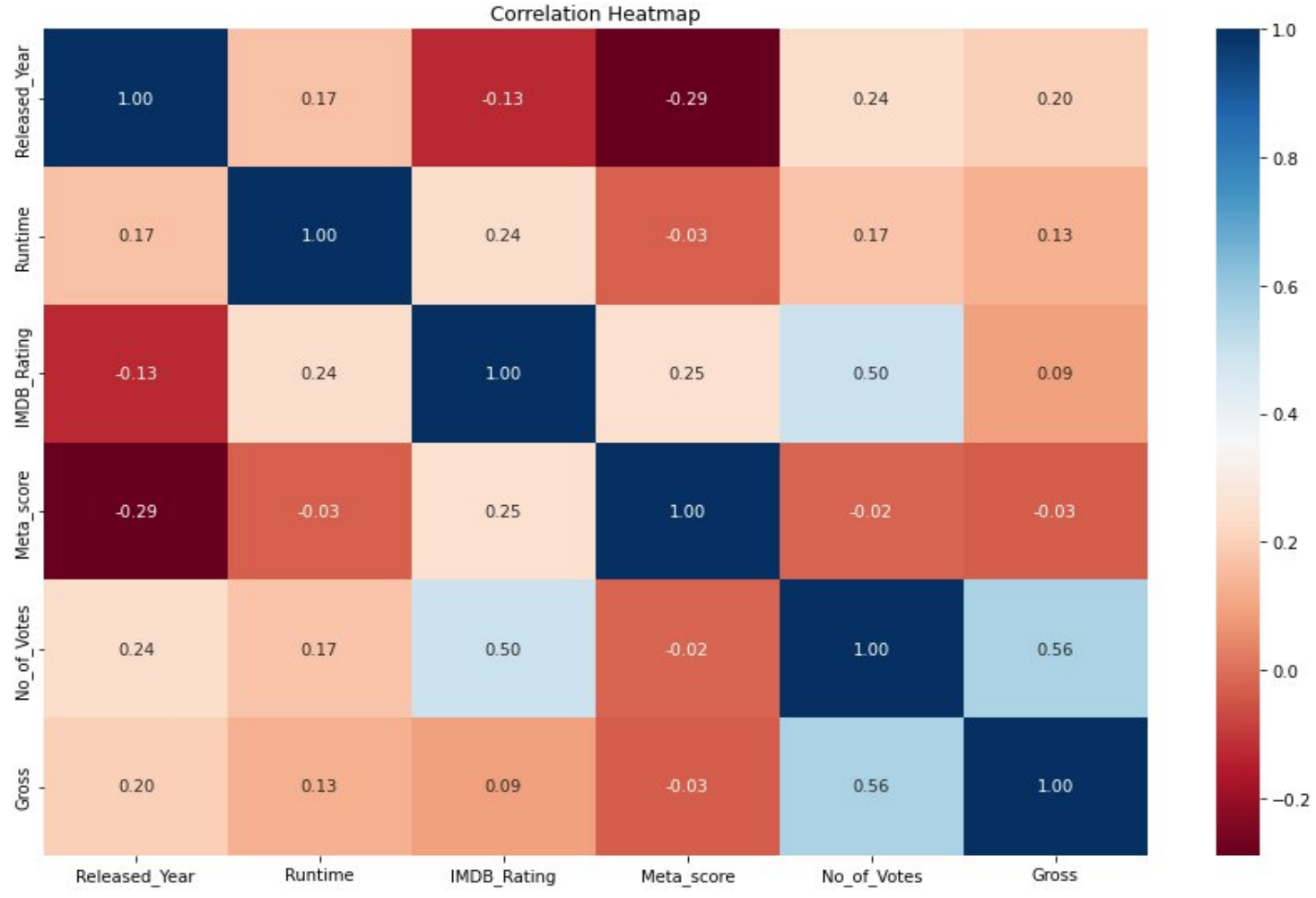

Distribution of Meta_score
Frequency
Meta_score
0
50
100
150
200
30
40
50
60
70
80
90
100


Distribution of No_of_Votes
Frequency
No_of_Votes
1e6
0
50
100
150
200
250
300
350
400
0.0
0.5
1.0
1.5
2.0

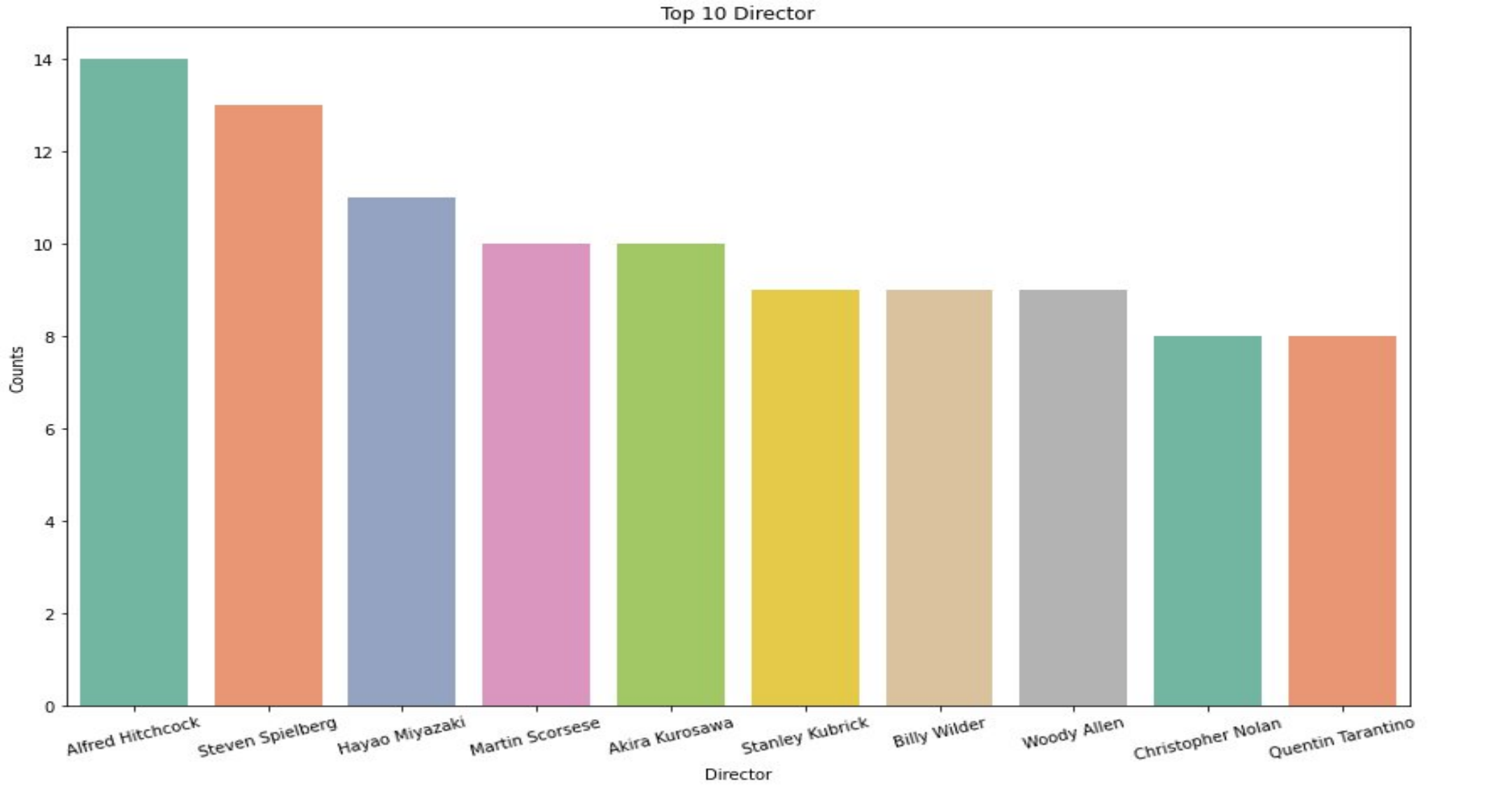

Distribution of Gross
Frequency
Gross
1e8
0
100
200
300
400
0
2
4
6
8

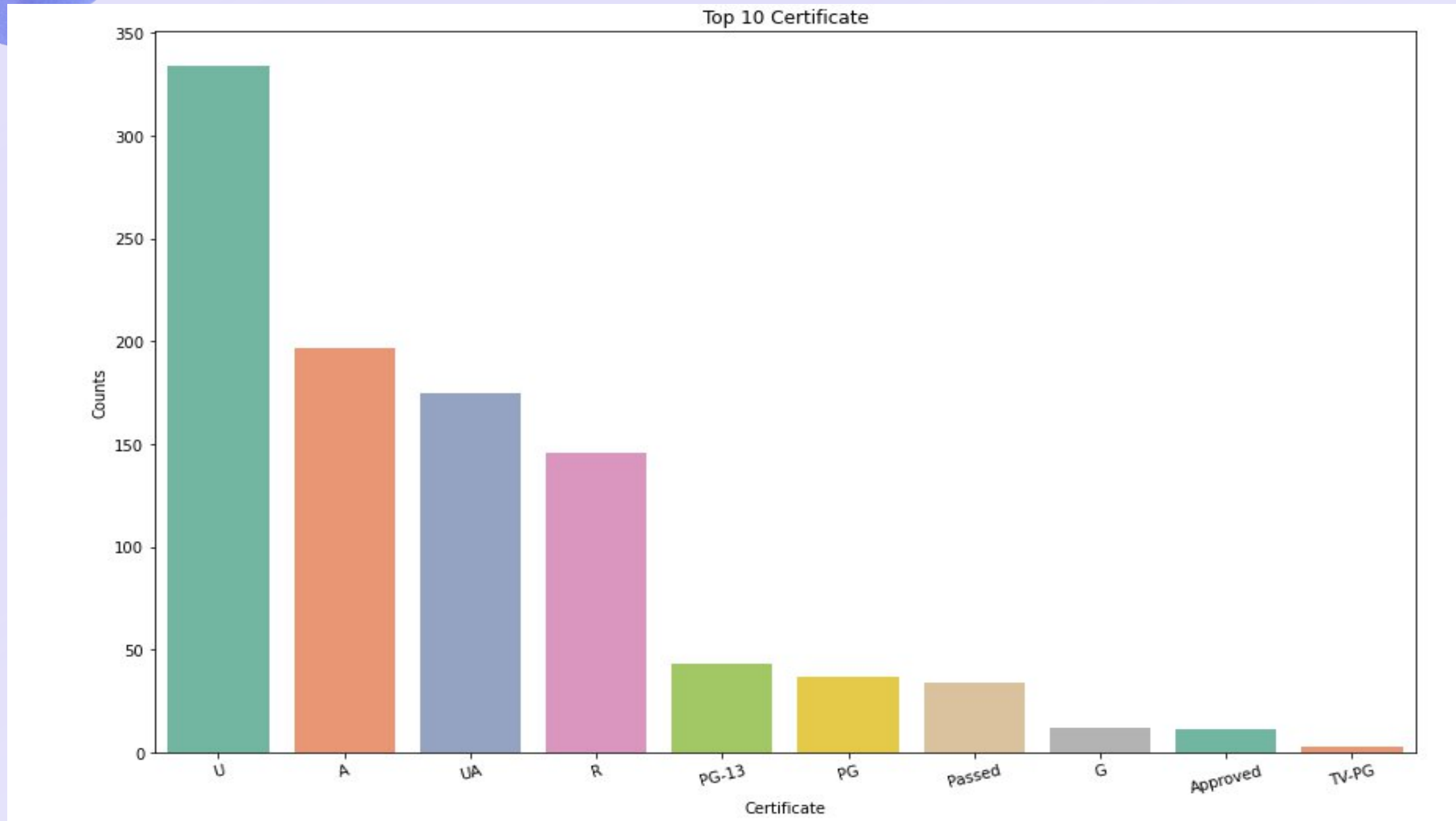
Correlation and heatmap for numerical columns.

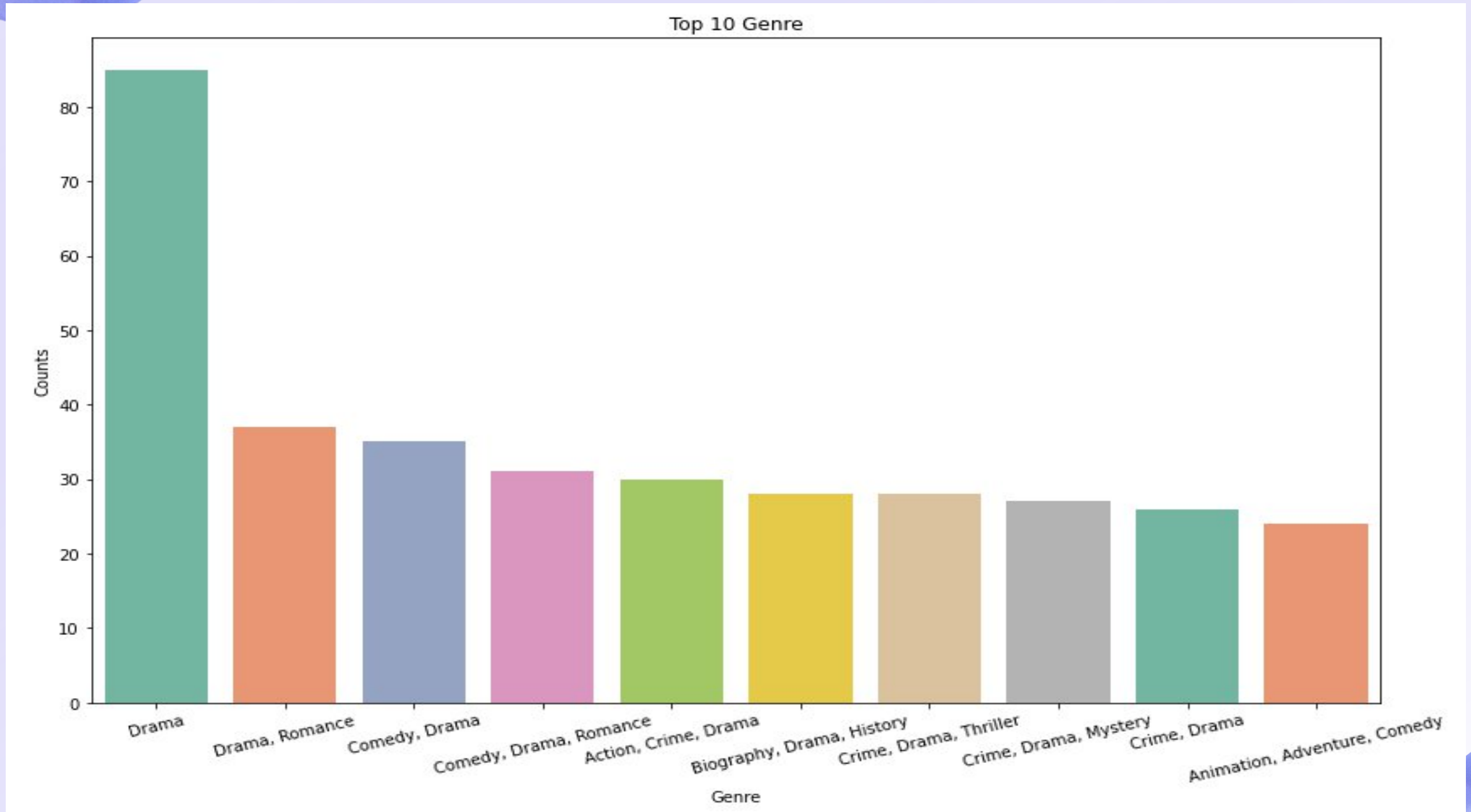
```
[23]: # for numerical cols correlation and heatmap

[25]: plt.figure(figsize=(12,8))
corr= df[numeric_columns].corr()-
sns.heatmap(corr, fmt="0.2f",cmap='RdBu',annot=True)
plt.title('Correlation Heatmap')
plt.savefig('heatmap.png')
plt.tight_layout()
plt.show()
```



Barplots for every categorical attribute(Feature)

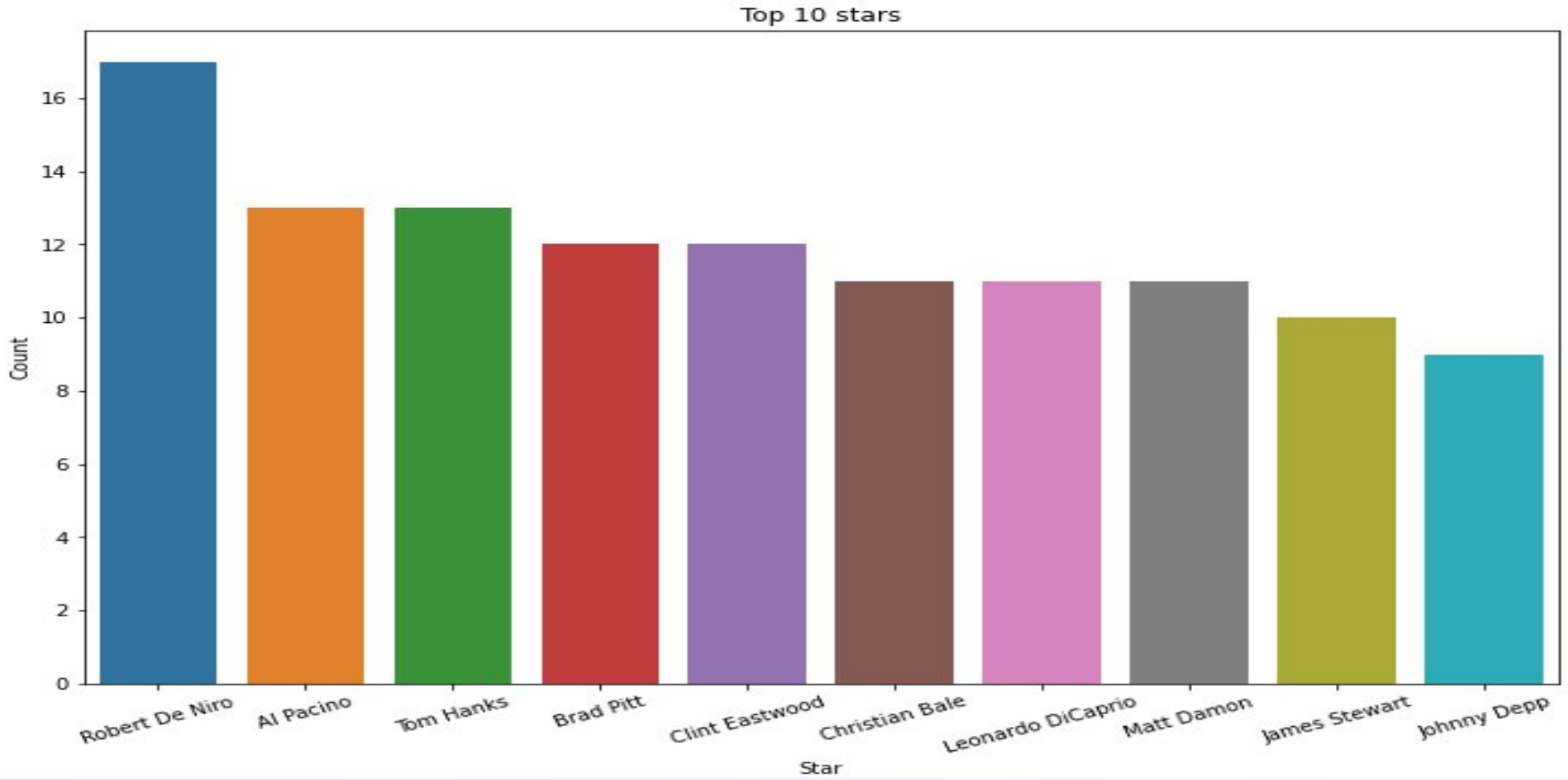



Top 10 Certificate
Counts
350
300
250
200
150
100
50
0
U
A
UA
R
PG-13
PG
Passed
G
Approved
TV-PG
Certificate


Top 10 Genre
Counts
0
10
20
30
40
50
60
70
80
Drama
Drama, Romance
Comedy, Drama
Comedy, Drama, Romance
Action, Crime, Drama
Biography, Drama, History
Crime, Drama, Thriller
Crime, Drama, Mystery
Crime, Drama
Animation, Adventure, Comedy
Genre

27]:
```
# combine the stars columns and plot the top 10 most frequent stars
# concatenate teh stars columns into one series
```

29]:
```
stars = pd.concat([df[col] for col in stars_col])
stars =stars.value_counts().nlargest(10)
plt.figure(figsize=(12,8))
sns.barplot(x=stars.index,y=stars.values)
plt.title('Top 10 stars')
plt.xlabel('Star')
plt.ylabel('Count')
plt.xticks(rotation=20)
plt.savefig('Top_Stars.png')
plt.show()
```



Feature Engineerining and Preprocessing

```
[8]: #     For high cardinality categorical columns, replace values that appear less than 'threshold' times with 'Other'.

•[85]: df['Certificate'] = df['Certificate'].astype('category')
       def reduce_cardinality(col, threshold=10):
           counts = col.value_counts()
           return col.apply(lambda x: x if counts[x] >= threshold else 'Other')
       for col in ['Director', 'Star1', 'Star2', 'Star3', 'Star4']:
           df[col] = reduce_cardinality(df[col], threshold=10)
           df[col] = df[col].astype('category')
```

The "cardinality problem" in data preprocessing refers to the challenges that arise when dealing with categorical features (columns) that have a high number of unique values (high cardinality). So we ave high cardinality problem in our categorical columns which can slow down our training.

Reduce the noise from rarely occuring categories by grouping them into a single category as “others”. By solving this problem, we prevent overfitting, make model much faster and robust, reduces dimensionality.

[35]:
```
df['Movie_Age'] = 2025 - df['Released_Year']
```

[36]:
```
from sklearn.preprocessing import MultiLabelBinarizer
```

[ ]:
```
#   Process the Genre column:
#   Split genres by a delimiter (e.g., '|').
#   Keep only the top_n most frequent genres, and replace others with 'Other'.
#   Create binary dummy (one-hot) features for the processed genres.
```

[37]:
```
def process_genres(df, col='Genre', delimiter=',', top_n=10):

#Split genres into lists
    df['Genre_list'] = df[col].apply(lambda x: [g.strip() for g in x.split(delimiter)] if isinstance(x, str) else [])
#Get overall frequency of each genre
    all_genres = pd.Series([genre for sublist in df['Genre_list'] for genre in sublist])
    top_genres = all_genres.value_counts().head(top_n).index.tolist()
#Replace non-top genres with 'Other'
    df['Genre_list_processed'] = df['Genre_list'].apply(
        lambda genres: [g if g in top_genres else 'Other' for g in genres]
    )
#Use MultiLabelBinarizer to create dummy variables
    mlb = MultiLabelBinarizer()
    genre_dummies = pd.DataFrame(mlb.fit_transform(df['Genre_list_processed']),
                                 columns=[f"Genre_{g}" for g in mlb.classes_],
                                 index=df.index)
#Append genre dummies and drop intermediate columns
    df = pd.concat([df, genre_dummies], axis=1)
    df.drop(columns=[col, 'Genre_list', 'Genre_list_processed'], inplace=True)
    return df
```

CONTINUE:

- Importing MultiLabelBinarizer, which helps convert a list of genres into binary (0 or 1) format — ideal for machine learning.
- One-Hot Encode with MultiLabelBinarizer.
- It transforms genre lists into one-hot encoded format.
- The resulting DataFrame has columns like Genre_Action, Genre_Drama, etc., each column has value of 0 or 1.
- Merge back to original dataframe.
- One-hot columns are added to the original DataFrame.
- Original genre columns (including intermediate lists) are dropped.
- Returns the cleaned DataFrame.
- Handles multilabel features (like genre) effiecently.
- Reduces noise by consolidating rare genres into 'Other'.

```
[39]: df = process_genres(df, col='Genre', delimiter=',', top_n=10)
```

```
[41]: df.head()
```

[41]:

	Released_Year	Certificate	Runtime	IMDB_Rating	Meta_score	Director	Star1	Star2	Star3	Star4	...	Genre_Adventure	Genre_Animation	Genre_Biography	Genre_Comedy	Genre_Crime	Genre_Drama	Genre_Mystery	Genre_Other	Genre_
0	1994	A	142	9.3	80.0	Other	Other	Other	Other	Other	...	0	0	0	0	0	1	0	0	
1	1972	A	175	9.2	100.0	Other	Other	Other	Other	Other	...	0	0	0	0	1	1	0	0	
2	2008	UA	152	9.0	84.0	Other	Other	Other	Other	Other	...	0	0	0	0	1	1	0	0	
3	1974	A	202	9.0	90.0	Other	Al Pacino	Other	Other	Other	...	0	0	0	0	1	1	0	0	
4	1957	U	96	9.0	96.0	Other	Other	Other	Other	Other	...	0	0	0	0	1	1	0	0	

5 rows × 24 columns

Baseline Modeling(LR), Advanced Modeling(RFR), HyperParameter Tuning And Evaluation.

[9]:
```
target= 'IMDB_Rating'
x= df.drop(columns=[target])
y= df[target]
x_train,x_test,y_train,y_test= train_test_split(x,y,random_state=42,test_size=0.2)
print(x_train.shape)
print(x_test.shape)
```

```
(799, 41)
(200, 41)
```

[10]:
```
lr= LinearRegression()
lr.fit(x_train,y_train)
y_pred_lin= lr.predict(x_test)
```

Evaluate Linear Regression RMSE AND R^2

[11]:
```
rmse_lin= np.sqrt(mean_squared_error(y_pred_lin,y_test))
r_score= r2_score(y_pred_lin,y_test)
```

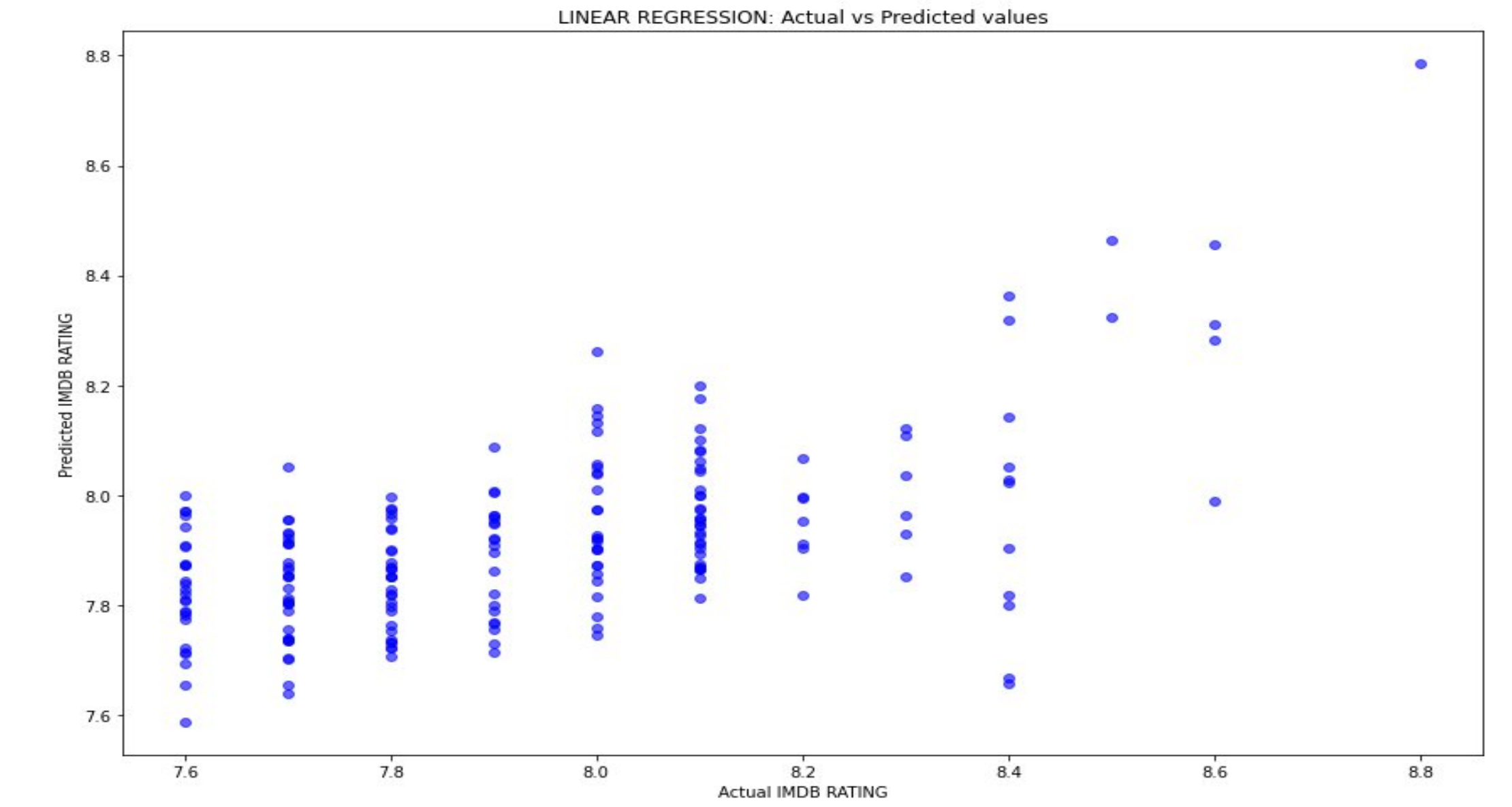
[12]:
```
print(round(rmse_lin,2),round(r_score,2),sep=" ")
```

```
0.21 -0.65
```

Plot Actual vs. Predicted for Linear Regression

[14]:
```
plt.figure(figsize=(12,8))
plt.scatter(y_test,y_pred_lin,color='blue',alpha=0.6)
plt.title('LINEAR REGRESSION: Actual vs Predicted values')
plt.xlabel('Actual IMDB RATING')
plt.ylabel('Predicted IMDB RATING')
plt.tight_layout()
plt.savefig('Preicted vs actual linear regression')
plt.show()
```

- We performed Linear Regression to predict IMDB Ratings using sciki-learn.
- This model struggles to capture meaningful relationships, suggesting that linear regression might not be suitable for this task without stronger feature engineering or using a more complex model.
- Performs worse that simply predicting the mean (-0.65)
- So we need advance model to solve this problem


LINEAR REGRESSION: Actual vs Predicted values
Predicted IMDB RATING
Actual IMDB RATING
8.8
8.6
8.4
8.2
8.0
7.8
7.6
7.6
7.8
8.0
8.2
8.4
8.6
8.8

Advance model: Random Forest Regression

```
[15]: rf= RandomForestRegressor()
```

```
[16]: rf.fit(x_train,y_train)
      y_pred_rf= rf.predict(x_test)
```

Evaluate the random forest performance

```
[17]: print(f"RMSE of rf Model: {np.sqrt(mean_squared_error(y_test,y_pred_rf))}")
```

```
RMSE of rf Model: 0.17727890455437711
```

```
[18]: print(f"r2_score of rf Model: {(r2_score(y_test,y_pred_rf))}")
```

```
r2_score of rf Model: 0.520002901870944
```

- We perfrom RandomForest Regression using scikit-learn ensemble.
- The model explains 52% of the variance in IMDb scores, which is a moderate fit.
- Overall, this model performs significantly better than a linear regression.
- RMSE indicates low prediction error with 0.17

Hyperparameter Tuning with GridSearchCV

Tuning Random Forest parameters

[24]:
```
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

grid_search = GridSearchCV(estimator=RandomForestRegressor(random_state=42),
                           param_grid=param_grid,
                           scoring='neg_root_mean_squared_error',
                           cv=5,
                           n_jobs=-1)
grid_search.fit(x_train, y_train)
print("Best parameters from GridSearchCV:", grid_search.best_params_)
```

```
Best parameters from GridSearchCV: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 200}
```

[ ]:

Evaluate the tuned model on test set

[25]:
```
best_rf = grid_search.best_estimator_
y_pred_best_rf = best_rf.predict(x_test)
rmse_best_rf = np.sqrt(mean_squared_error(y_test, y_pred_best_rf))
r2_best_rf = r2_score(y_test, y_pred_best_rf)
print("Tuned Random Forest RMSE:", rmse_best_rf)
print("Tuned Random Forest R²:", r2_best_rf)
```

```
Tuned Random Forest RMSE: 0.17554151332956397
Tuned Random Forest R²: 0.5293650568608862
```

Here with this model, we improved our prediction metrics a lot better than a linear regression model.

Visualization, Interpretation and Evaluation

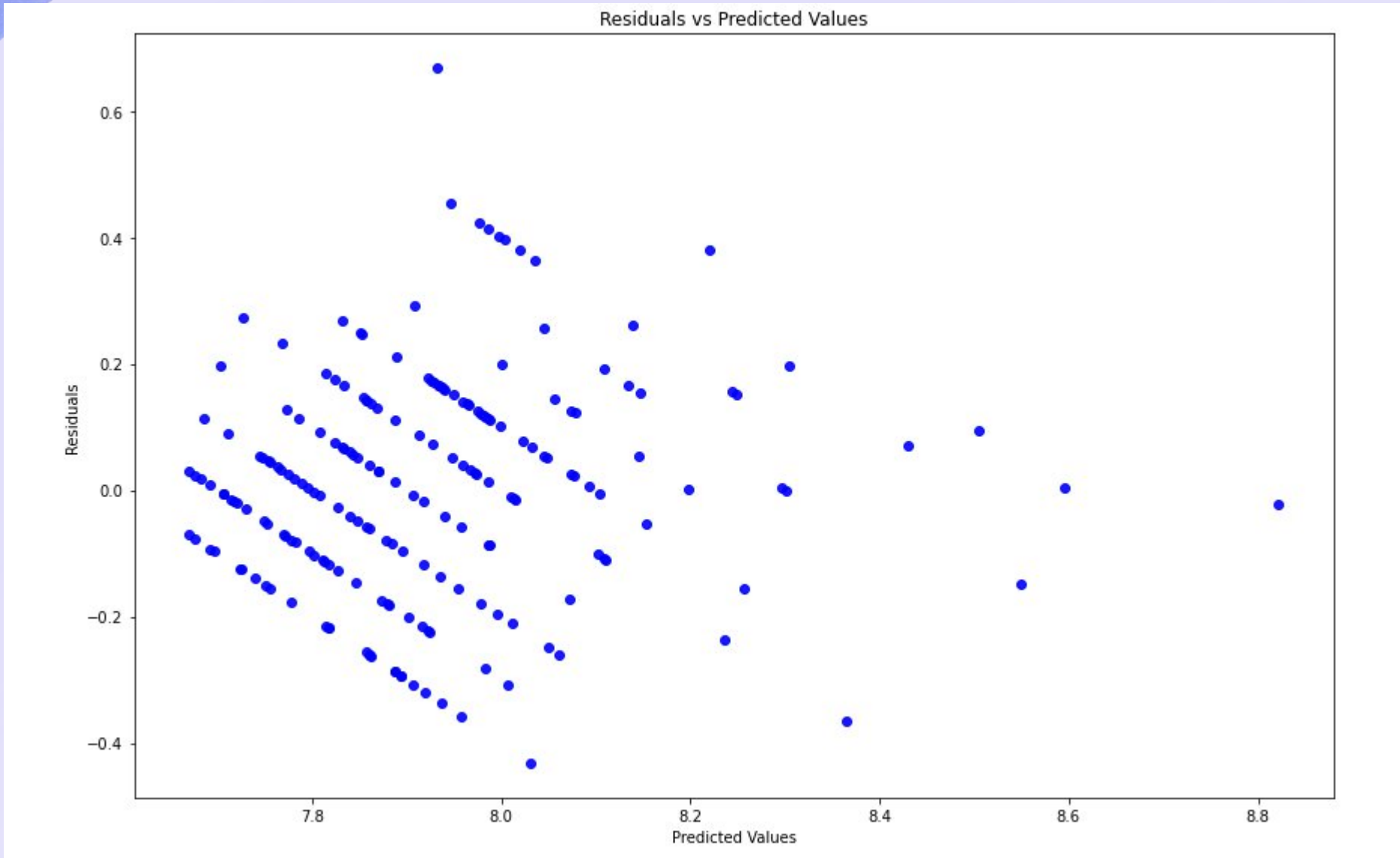
Residual Analysis

[26]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import scipy.stats as stats
```

[27]:
```
y_pred=best_rf.predict(x_test)
residuals= y_test.values-y_pred
```

Residuals vs Preidcted Values

[28]:
```
plt.figure(figsize=(12,8))
plt.scatter(x=y_pred,y=residuals,alpha=0.9, color='blue')
plt.title('Residuals vs Predicted Values')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.tight_layout()
plt.savefig('Residuals_VS_Predicted.png')
plt.show()
```



The residuals appear to be generally scattered around the horizontal line at zero (randomly), which is a positive sign. It suggests that, on average, the model does not systematically overpredict or underpredict IMDb ratings.

Cross-Validation Evaluation

```
[38]: from sklearn.model_selection import cross_val_score


cv_rmse = -cross_val_score(best_rf, x_train, y_train, cv=5, scoring='neg_root_mean_squared_error')
cv_r2 = cross_val_score(best_rf, x_train, y_train, cv=5, scoring='r2')

print("Cross-Validated RMSE: {:.3f} ± {:.3f}".format(np.mean(cv_rmse), np.std(cv_rmse)))
print("Cross-Validated R²: {:.3f} ± {:.3f}".format(np.mean(cv_r2), np.std(cv_r2)))
```

```
Cross-Validated RMSE: 0.194 ± 0.005
Cross-Validated R²: 0.500 ± 0.065
```

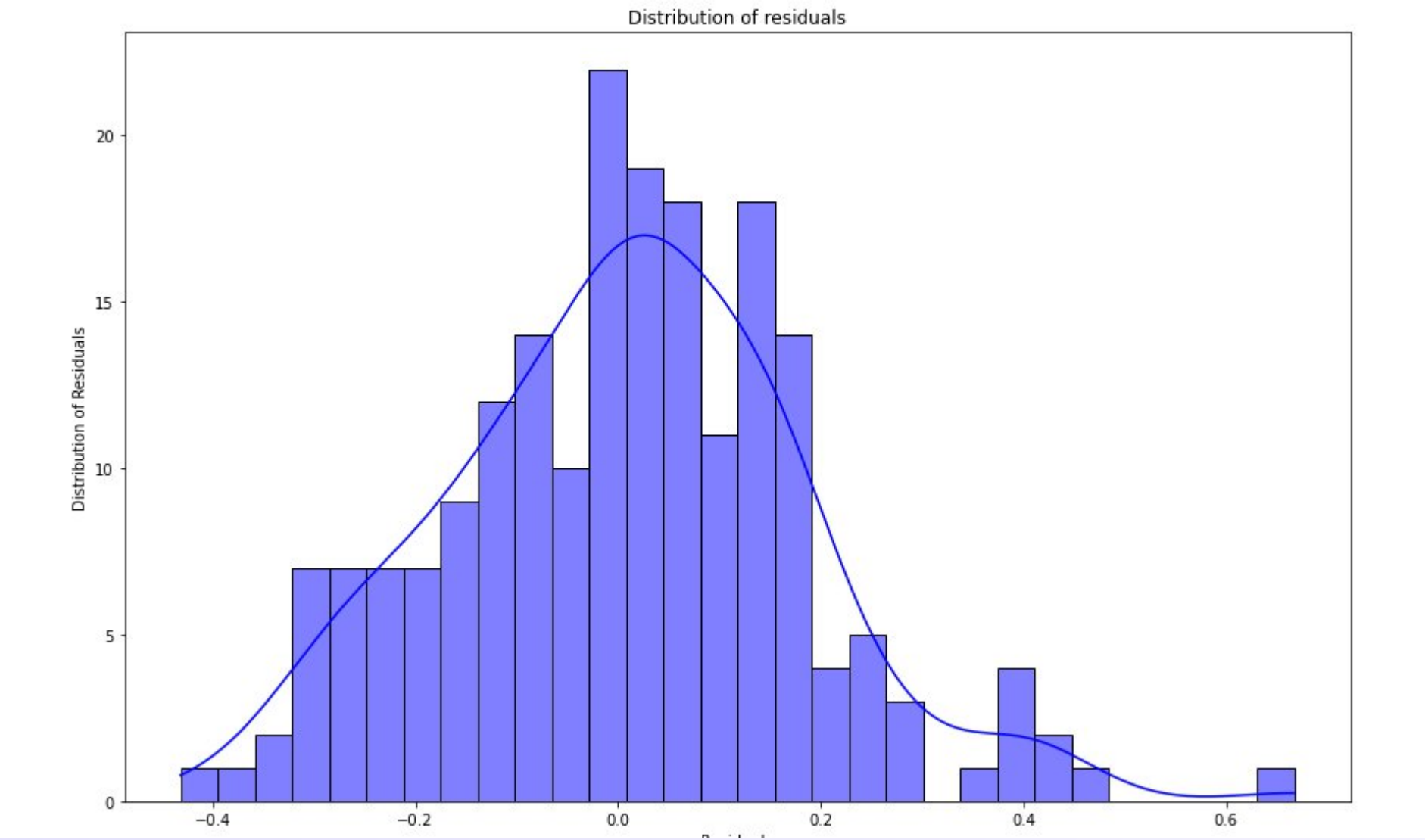
- These cross-validated metrics confirm that the tuned Random Forest is a solid and stable model for predicting IMDb ratings.
- Calculate cross-validated RMSE (using negative root mean squared error) and R².

Generate a QQ-Plot to assess normality of residuals

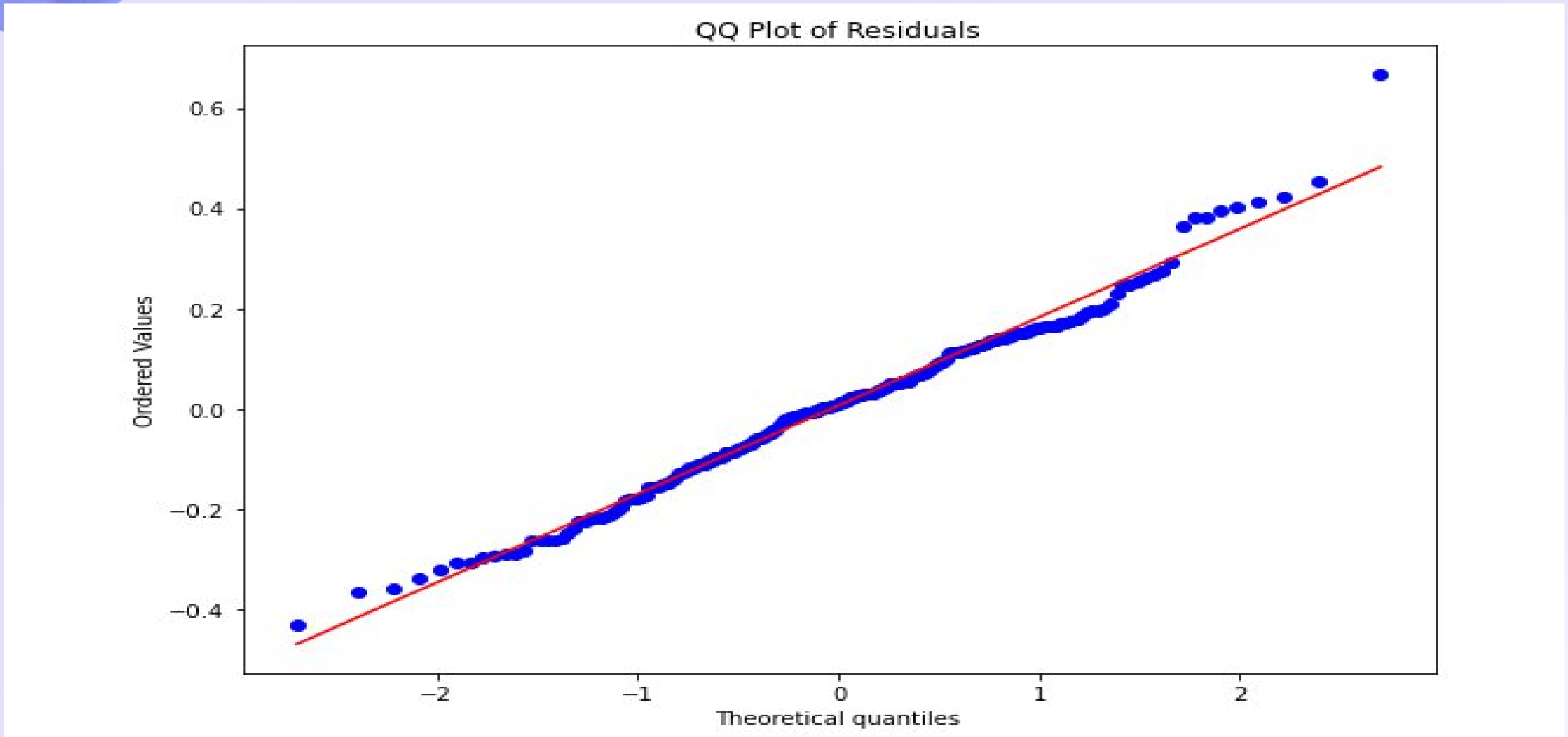
```
0]: plt.figure(figsize=(8, 6))
stats.probplot(residuals, dist="norm", plot=plt)
plt.title("QQ Plot of Residuals")
plt.savefig('QQ_Plot_of_Residuals.png')
plt.tight_layout()
plt.show()
```

Distribution of residuals

```
[29]: plt.figure(figsize=(12,8))
sns.histplot(residuals,bins=30,kde=True,color='blue')
plt.title('Distribution of residuals')
plt.xlabel('Residuals')
plt.ylabel('Distribution of Residuals')
plt.tight_layout()
plt.savefig('Distribution_of_Residuals.png')
plt.show()
```



Most residuals centered around zero which suggests that model doesn't have systematic Bias. Overall shape of the distribution looks bell-shaped which illustrate that residuals have normal distributions.

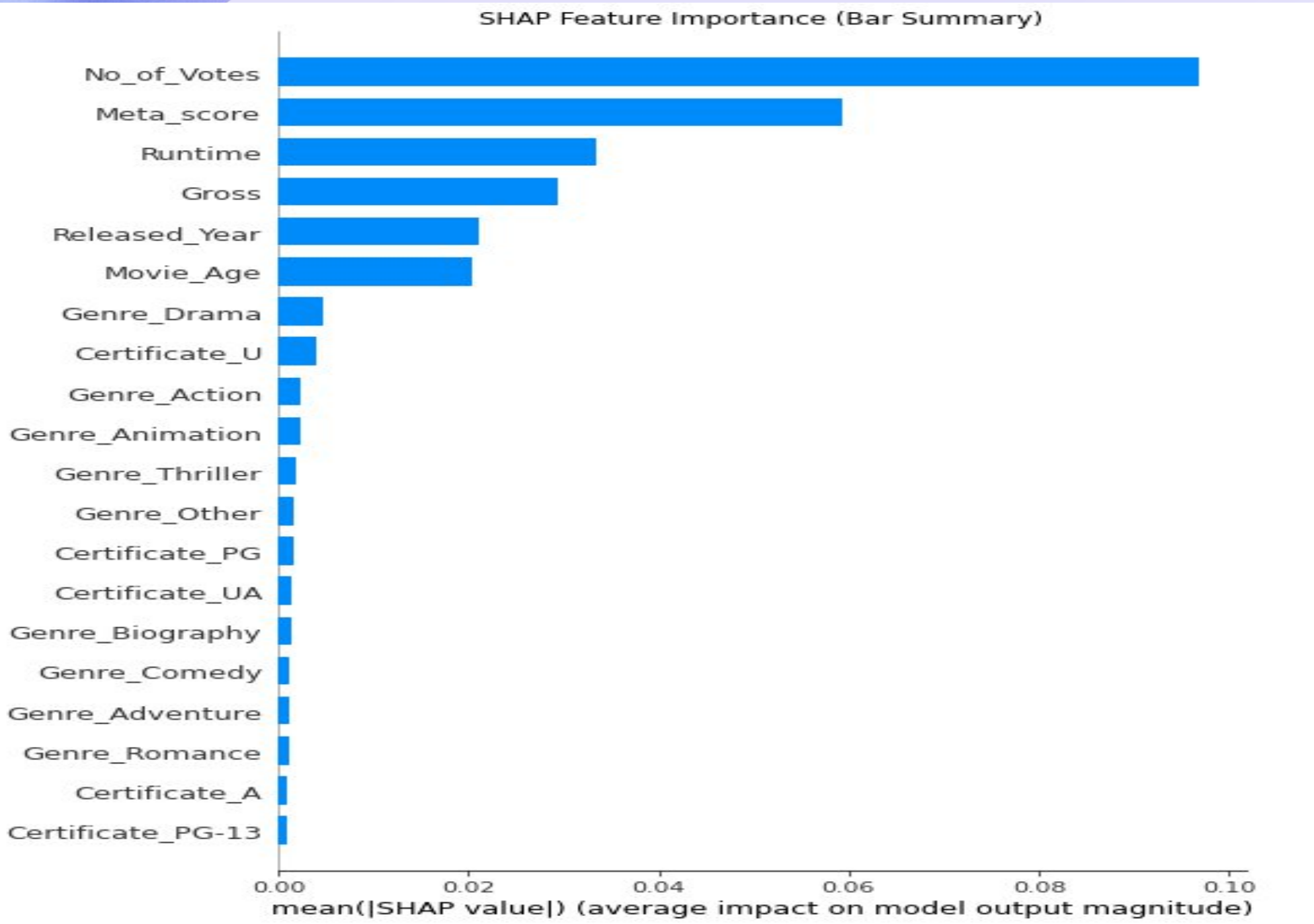


Majority of points align well with the line, the residuals are largely normal. This supports the assumption of normality in the errors for the model.

!pip install shap

```
33]: import shap
     explainer = shap.TreeExplainer(best_rf)

     shap_values = explainer.shap_values(x_test)

     shap.summary_plot(shap_values, x_test, plot_type="bar", show=False)
     plt.title("SHAP Feature Importance (Bar Summary)")
     plt.tight_layout()
     plt.savefig('BarSummary_featureImportance.png')
     plt.show()
```

- Create SHAP values using TreeExplainer (efficient for tree-based models).
- Compute SHAP values for a subset of the test set (you might use the entire X_test if it's not too large).
- Plot a bar summary to see global feature importance via SHAP.

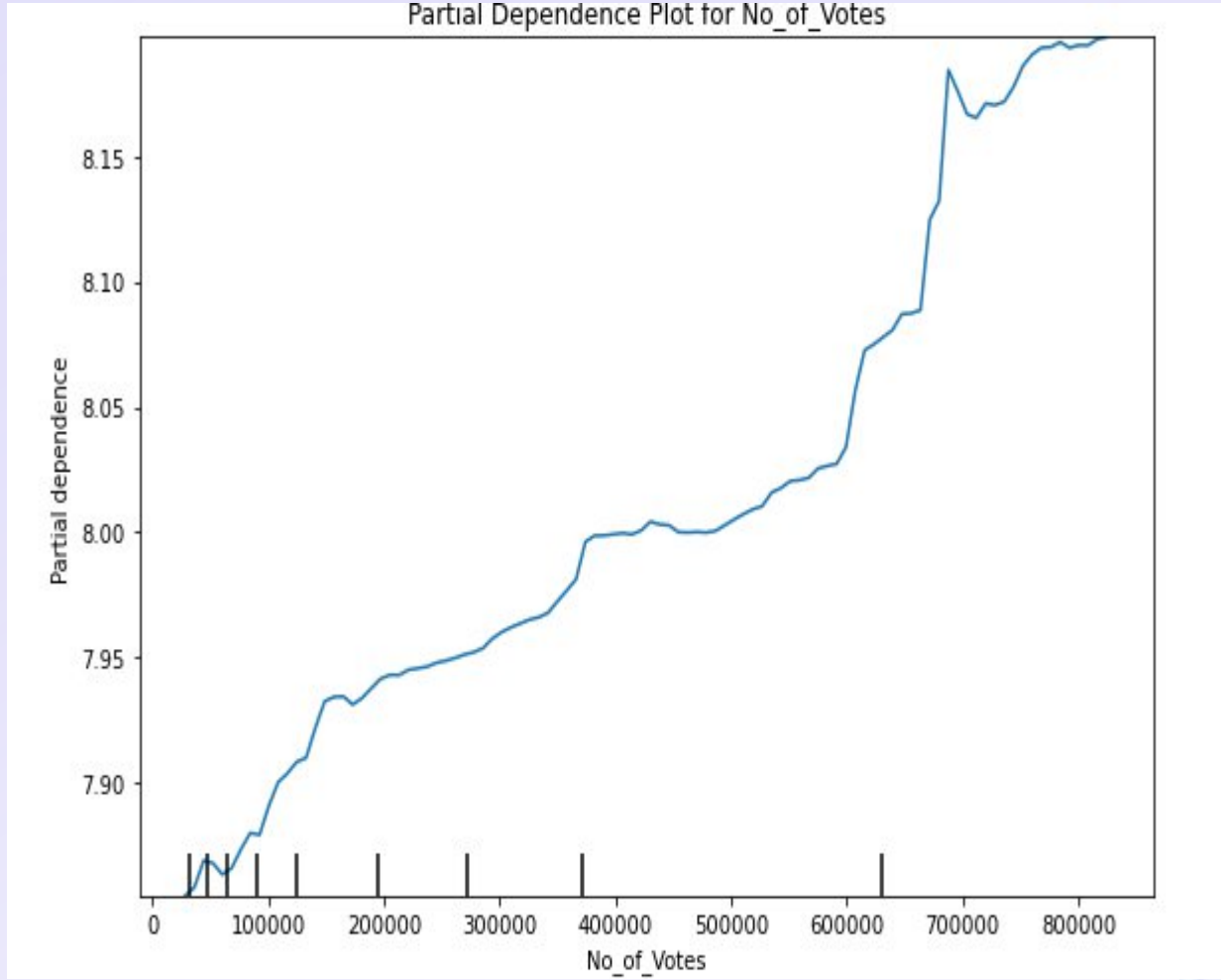


- No_of_Votes had the highest impact on model predictions—indicating that the number of votes a movie gets strongly correlates with its rating.
- Followed by Meta_score, Runtime, Gross, Released_Year, and Movie_Age.

Partial Dependence Plots (PDP)

```
[34]: from sklearn.inspection import PartialDependenceDisplay
top_feature = feature_names[indices[0]]
print("Top Feature for PDP:", top_feature)
fig, ax = plt.subplots(figsize=(8, 6))
PartialDependenceDisplay.from_estimator(best_rf, x_test, [top_feature], ax=ax)
plt.title("Partial Dependence Plot for " + top_feature)
plt.savefig('PDP_for_TopFeature.png')
plt.tight_layout()
plt.show()
```

```
Top Feature for PDP: No_of_Votes
```



- This code generates a Partial Dependence Plot (PDP) for the top feature: No_of_Votes—in a Random Forest model trained to predict IMDb scores
- The resulting plot suggests that as No_of_Votes increases, the model's predicted IMDb score also rises—indicating a positive relationship.

Conclusion

1. Predictive Performance:

After comprehensive cleaning and feature engineering, a tuned Random Forest Regressor cut error to RMSE ≈ 0.17 and explained ≈ 52 % of rating variance, a decisive improvement over the linear-regression baseline (R^2 = –0.65).

2. Key insights of IMDb Scores:

- Popularity and critical-consensus signals matter most:
- No_of_Votes (strongest effect)
- Meta_score
- Runtime, Gross, Released_Year / Movie_Age
- Genre and certificate tags contribute far less once high-cardinality noise is collapsed into “Other.”

3. Business Implications:

Studios or streaming platforms can prioritise titles with strong critic scores and sustained audience engagement (votes) to boost average ratings.

4. Model Robustness & Interpretability:

Residual diagnostics show no systematic bias, and SHAP + PDP visualisations make the model transparent enough for stakeholder trust and actionable what-ifs.

Continue

5. Limitations & Next Steps:

An R^2 of 0.52 leaves room for richer features—e.g. social-media sentiment, cast star-power, release-window effects.

Bottom line: A data-driven, interpretable workflow now predicts IMDb ratings with practical accuracy and clear levers for content, marketing, and investment teams to pull.