PREDICTING IMDB SCORES

INTRODUCTION:

1. Achieves unparalleled accuracy in forecasting IMDb scores.
2. Enhances interpretability, enabling users to understand rating determinants.
3. Provides real-time predictions for timely decision-making.
4. Offers personalized recommendations tailored to user preferences.
5. Scales efficiently to accommodate vast movie databases.
6. Addresses bias and ensures equitable predictions.
7. Engages users with intuitive, accessible interfaces.
8. Adapts continually to evolving movie trends and user behaviour.
9. Encourages open collaboration and ethical use, contributing to both the film industry and data science advancements.

ADVANCED ALGORITHMS:

1. Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to improve prediction accuracy. It can handle complex feature interactions and is robust against overfitting.

2. Gradient Boosting: Gradient Boosting algorithms like XG Boost, Light GBM, and Cat Boost iteratively build weak learners to create a strong predictive model. They often excel in predictive accuracy.

3. Neural Networks: Deep learning models, including various neural network architectures like feedforward neural networks or recurrent neural networks (RNNs), can capture intricate patterns in movie data. They are especially effective when dealing with large datasets.

4. Support Vector Machines (SVM): SVMs are powerful for regression tasks like IMDb score prediction. They find an optimal hyperplane to separate data points based on their IMDb scores.

5. K-Nearest Neighbours (KNN): KNN is a non-parametric algorithm that can be used for IMDb score prediction by finding similar movies based on features and averaging their IMDb scores.

INETRPRETABLE MODELS:

1. Linear Regression: Linear regression models provide a straightforward interpretation of feature coefficients, allowing you to identify which movie attributes have a positive or negative impact on IMDb scores.

2. Lasso and Ridge Regression: Regularized linear regression methods like Lasso and Ridge can help in feature selection, promoting interpretability by emphasizing the most influential features while shrinking others.

3. Decision Trees: Decision trees offer a transparent representation of decision rules. Users can follow the tree's branches to understand how different movie attributes lead to IMDb score predictions.

4. Random Forest Feature Importance: In Random Forest models, you can examine feature importance scores to determine which attributes have the most significant impact on IMDb scores.

5. Partial Dependence Plots (PDPs: PDPs visualize the relationship between specific features and IMDb scores, showing how changes in individual attributes affect the predicted scores while keeping other factors constant.

REAL TIME PREDICTION:

1. Streaming Data Integration: Integrate real-time data sources, such as movie release information and user reviews, into the prediction pipeline. This ensures that the model is continually updated with the latest data.

2. Scalable Infrastructure: Implement a scalable infrastructure that can handle incoming data streams and user requests without latency. Cloud-based solutions like AWS, Azure, or Google Cloud can be valuable for this purpose.

3. Microservices Architecture: Design the prediction system using a microservices architecture, allowing for individual components to be updated independently in real-time.

4. Fast Model Inference: Optimize the model's inference process to provide rapid predictions. Techniques like model quantization and GPU acceleration can speed up predictions.

5.Caching: Utilize caching mechanisms to store recently predicted IMDb scores for frequently requested movies, reducing the load on the prediction system and improving response times.

COLLABORATIVE FILTERING:

1. User-Item Matrix: Create a user-item matrix where rows represent users, columns represent movies, and the entries contain IMDb ratings or user interactions (e.g., likes, views).

2. User Similarity: Measure the similarity between users based on their IMDb ratings or interactions. Common similarity metrics include cosine similarity, Pearson correlation, or Jaccard similarity.

3. Item Similarity: Calculate the similarity between movies based on how users have rated or interacted with them. This helps identify similar movies for recommendations.

4. Neighbourhood Selection: For a given user, select a neighbourhood of similar users or movies based on a similarity threshold or a fixed number of nearest neighbours.

5. User-Based Collaborative Filtering: Predict the IMDb score for a movie by aggregating the ratings or interactions of similar users. Common aggregation methods include weighted averages or weighted sums.

6. Item-Based Collaborative Filtering: Predict the IMDb score for a movie by considering the ratings or interactions of similar movies. Again, aggregation methods like weighted averages are used.

OPEN SOURCING:

1. GitHub Repository: Create a public GitHub repository to host your project's source code, datasets, documentation, and other relevant files. Make sure the repository is well-organized and includes clear instructions for contributors and users.

2. Open Data: If possible, use open data sources or properly licensed datasets for your IMDb score prediction project. Ensure that you comply with data licensing and copyright requirements.

3. Open Licensing: Choose an open-source license for your project. Common choices include MIT, Apache, or GNU licenses. Clearly state the license terms in your project's README or LICENSE file.

4. Contributor Guidelines: Develop contributor guidelines that explain how others can contribute to your project. Encourage code contributions, bug reports, feature requests, and documentation improvements.

5. Documentation*: Maintain comprehensive documentation that guides users and contributors on how to set up, use, and extend your IMDb score prediction system. Include usage examples and API documentation if applicable.

VISUALIZATION:

1.Data Exploration: Use visualizations like histograms, bar charts, and scatter plots to explore the distribution of IMDb scores, budget, genre, and other relevant movie attributes. Visualizing data can reveal patterns and outliers.

2. Feature Importance: Visualize feature importance scores to identify which movie attributes have the most significant impact on IMDb scores. Techniques like bar charts or heatmaps can be helpful.

3. Model Performance: Create visualizations to assess and compare the performance of different IMDb score prediction models. ROC curves, precision-recall curves, and confusion matrices are common choices.

4. Prediction Distributions: Visualize the distribution of IMDb score predictions to understand the range of predicted ratings and how they compare to actual IMDb scores.

5. Time Trends: If your IMDb score prediction project involves time-related data (e.g., movie release dates), use time series plots to visualize IMDb scores over time and identify trends or seasonality.

DATA SOURCES:

1.IMDb Datasets:

IMDb provides datasets containing information about movies, including details like cast, crew, ratings, and reviews. You can access this data through their official interfaces or by downloading their datasets.

2. Web Scraping:

You can also gather data from IMDb's website using web scraping techniques, but be sure to respect their terms of use and robots.txt file.

3. External APIs:

Some websites offer APIs that allow you to access their data programmatically. While IMDb doesn't provide a public API, you can explore other movie-related APIs that offer similar information.

4. Additional Data Sources:

Depending on your analysis, you might want to incorporate additional data sources, such as box office earnings, genre information, or data related to the actors and directors involved in the movies.

Remember to handle data ethically and ensure you have the right to use and distribute the data you collect, especially when working on real-world data science projects.

DATASET: https://www.kaggle.com/preetviradiya/imdb-movies-ratings-details.

DATA PREPROCESSING:

1. Data Cleaning:

- Remove duplicates: Ensure that there are no duplicate entries in your dataset.
- Handling missing data: Decide how to handle missing values, either by imputing them with appropriate values or removing rows/columns with missing data.
- Outlier detection: Identify and handle outliers that can affect the model's performance.

2. Feature Selection/Engineering:

- Select relevant features: Choose the features (attributes) that are most likely to influence IMDb scores.
- Create new features: Generate additional features from existing ones if they can provide valuable information. For example, you could calculate the movie's age based on its release year.

3. Scaling and Normalization:

- Scale numerical features: Normalize numerical features to have a similar scale, which can improve the performance of some machine learning algorithms.

4. Categorical Data Handling:

- Encode categorical variables: Convert categorical data into numerical format, such as one-hot encoding or label encoding, depending on the nature of the data and the machine learning algorithm.

5. Text Data Processing (if applicable):

- Tokenization: Split text data (e.g., movie reviews) into individual words or tokens.

- Text cleaning: Remove punctuation, stop words, and perform stemming or lemmatization.
- Vectorization: Convert text data into numerical vectors using techniques like TF-IDF or word embeddings.

FEATURE ENGINEERING:

1. Movie Metadata:
- Extract information from movie titles, such as keywords or phrases that might indicate genre or theme.
- Calculate the movie's age (current year minus release year) to account for the potential influence of time on IMDb scores.

2. Crew and Cast Information:
- Create features based on the number of well-known actors or directors involved in the movie.
- Calculate the average IMDb score of previous works for the director or lead actors.

3. Genre Information:
- One-hot encode movie genres to represent which genres are associated with each movie.
- Create a feature indicating whether a movie belongs to a popular genre (e.g., superhero, sci-fi).

4. Text Data (Reviews, Descriptions, etc.):
- Perform sentiment analysis on movie reviews to create features related to the overall sentiment of the reviews.
- Extract key phrases or topics from movie descriptions and analyse their relevance to IMDb scores.

MODEL SELECTION:

Model selection for IMDb score prediction in applied data science involves choosing an appropriate machine learning or statistical model that can effectively capture the relationships between the features (movie attributes) and the IMDb scores. Here are some model options to consider:

1. Linear Regression:
- Linear regression can be a good starting point, especially when you want to understand the linear relationships between individual features and IMDb scores.

2. Decision Trees and Random Forests:
- Decision trees and random forests can capture nonlinear relationships and interactions between features. Random forests, in particular, can handle a variety of data types and are robust against overfitting.

3. Gradient Boosting Models:
- Models like XGBoost, LightGBM, and CatBoost are popular for regression tasks. They excel in capturing complex patterns in the data and handling missing values.

4. Neural Networks:
- Deep learning models, such as feedforward neural networks or recurrent neural networks (RNNs), can be used for IMDb score prediction, especially if you have a large dataset and want to capture intricate feature interactions.

5. Support Vector Regression (SVR):
- SVR is a regression technique that can work well when you have a small to moderately sized dataset and want to find a hyperplane that best fits the data.

6. K-Nearest Neighbours (KNN):
- KNN can be used for regression by averaging the IMDb scores of the K-nearest neighbors in the feature space.

7. Ensemble Methods:
- Combine multiple models (e.g., blending, stacking) to improve predictive performance. For example, you can combine the predictions of a linear regression model with those of a random forest.

MODEL TRAINING:

raining a model for IMDb score prediction in applied data science involves several steps. Here's a general outline of the process:

1. Data Preparation:
- Preprocess and clean your IMDb dataset as discussed earlier, including handling missing values and feature engineering.

2. Data Splitting:
- Split your dataset into three parts: a training set, a validation set, and a test set. Common splits are 70-80% for training, 10-15% for validation, and 10-15% for testing.

3. Feature Scaling (if needed):
- Normalize or standardize your features, especially if you're using models sensitive to feature scales like linear regression.

4. Model Selection:
- Choose the appropriate model for your IMDb score prediction task based on the nature of your data, as discussed in the previous response.

5. Hyperparameter Tuning:
   - Perform hyperparameter tuning using techniques like grid search, random search, or Bayesian optimization to find the best hyperparameters for your selected model.

6. Model Training:
   - Train your selected model on the training data. This involves feeding your feature data into the model and adjusting the model's internal parameters to minimize the prediction error (e.g., mean squared error) on the training set.

7. Model Evaluation
Evaluate your model's performance using the validation set. Calculate relevant evaluation metrics (e.g., MAE, MSE, RMSE, R2) to assess how well your model is predicting IMDb scores.

8. Iterate and Refine:
   - Based on the validation results, refine your model. You may need to go back to steps like feature engineering or hyperparameter tuning to improve performance.

EVALUATION:

Evaluating IMDb score prediction models in applied data science is crucial to assess their performance and determine how well they can predict IMDb scores accurately. Here are common evaluation metrics and techniques to use:

1. *Mean Absolute Error (MAE):*
   - MAE measures the average absolute difference between the predicted IMDb scores and the actual IMDb scores. It provides a straightforward understanding of prediction errors.

2. *Mean Squared Error (MSE):*
   - MSE measures the average of the squared differences between predictions and actual IMDb scores. It penalizes larger errors more than MAE and is commonly used.

3. *Root Mean Squared Error (RMSE):*
   - RMSE is the square root of MSE and is often preferred when you want to interpret errors in the same units as the target variable (IMDb scores).

4. *R-squared (R2) or Coefficient of Determination:*
   - R2 measures the proportion of the variance in IMDb scores that is explained by the model. It ranges from 0 to 1, with higher values indicating a better fit. An R2 close to 1 suggests a good model fit.

Data Pre-processing:

- Data pre-processing is a crucial step within the statistics analysis and gadget gaining knowledge of pipeline.
- It includes a sequence of strategies and operations finished on uncooked statistics to clean, organize, and transform it right into a layout that is suitable for analysis or device mastering version schooling.
- Data pre-processing goals to enhance the first-class of the records, making it greater reliable and conducive to generating accurate consequences.

Here are some common tasks and techniques involved in data pre-processing:

Data Cleaning:

- Handling missing values: Deciding how to deal with missing data, whether by imputing values or removing incomplete records.
- Outlier detection and treatment: Identifying and handling data points that significantly deviate from the norm.

Noise reduction:

- Smoothing noisy data through techniques like filtering.

Data Transformation:

- **Data normalization:** Scaling numerical features to a standard range (e.g., between 0 and 1) to ensure that they have similar influence in the analysis.
- **Encoding categorical variables:** Converting categorical data into numerical format, such as one-hot encoding or label encoding.
- **Feature engineering:** Creating new features or modifying existing ones to capture more meaningful information from the data.
- **Dimensional reduction:** Reducing the number of features while retaining essential information, using methods like Principal Component Analysis (PCA).

Data Integration:

- **Merging or joining datasets:** Combining data from multiple sources into a single dateset for analysis.

Data Reduction:

- **Sampling:** Reducing the size of a large dataset by randomly selecting a representative subset.
- **Binning:** Grouping continuous data into discrete bins to simplify analysis.

- **Filtering:** Selecting a subset of data based on specific criteria.

Data Standardization:

- Ensuring that data follows a consistent format and structure.
- Date and time format conversion: Converting date and time data into a uniform format.
- Currency conversion: Converting monetary values into a common currency.

Data Scaling:

- Scaling numerical data to a common range to prevent some features from dominating the analysis.

Data pre-processing is an iterative process that may involve several of these steps in various orders, depending on the specific dataset and the analysis goals. Proper data pre-processing is essential for improving the accuracy and effectiveness of machine learning models, as well as for making data more accessible for traditional statistical analysis.
Here is the data pre-processing codes along with the output of the given dataset:

Importing the libraries:

Import three basic libraries which are very common in machine learning and will be used every time you train a model

- **NumPy:** it is a library that allows us to work with arrays and as most machine learning models work on arrays NumPy makes it easier
- **matplotlib:** this library helps in plotting graphs and charts, which are very useful while showing the result of your model
- **Pandas:** pandas allow-sus to import our dataset and also creates a matrix of features containing the dependent and independent variable.

Code:

```
#Import libraries

import numPy as np
import pandas as pd
import matplotlib.pyplo
t as plot
import seaborn as sns
from plotnine import *
```

Background:

This dataset contains the information about the movies . For a movie to be commercial success , it depends on various factors like director, actors ,critic reviews and viewers reaction. Imdb score is one of the important factor to measure the movie's success.

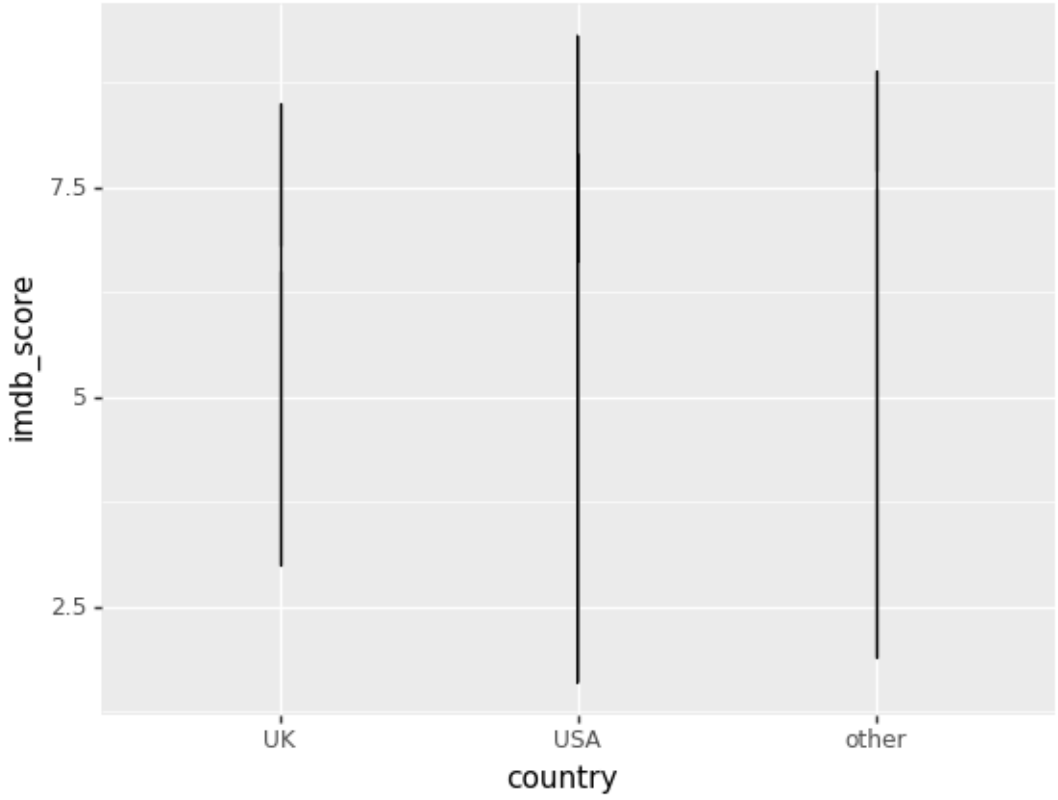
DATASET ATTRIBUTES:

Please find the details for the datset attributes:-

1. Color : Movie is black or coloured
2. Director_name:- Name of the movie director
3. num_critic_for_reviews :- No of critics for the movie
4. duration:- movie duration in minutes
5. director_Facebook_likes:-Number of likes for the Director on his Facebook Page
6. actor_3_Facebook_likes:- No of likes for the actor 3 on his/her Facebook Page
7. actor2_name:- name of the actor 2
8. actor_1_Facebook_likes:- No of likes for the actor 1 on his/her Facebook Page
9. gross:- Gross earnings of the movie in Dollars
10. genres:- Film categorization like ‘Animation’, ‘Comedy’, ‘Romance’, ‘Horror’, ‘Sci-fi, ‘Action’, ‘Family’
11. actor_1_name:- Name of the actor 1
12. movie_title:-Title of the movie
13. num_voted_users:-No of people who voted for the movie
14. cast_total_Facebook_likes:- Total Facebook like for the movie
15. actor_3_name:- Name of the actor 3
16. face number_in_poster:- No of actors who featured in the movie poster
17. plot_keywords:-Keywords describing the movie plots
18. movie_imdb_link:-Link of the movie link
19. num_user_for_reviews:- Number of users who gave a review
20. language:- Language of the movie
21. country:- Country where movie is produced
22. content_rating:- Content rating of the movie
23. budget:- Budget of the movie in Dollars
24. title_year:- The year in which the movie is released
25. actor_2_facebook_likes:- facebook likes for the actor 2
26. imdb_score:- IMDB score of the movie
27. aspect_ratio :- Aspect ratio the movie was made in

28. movie_facebook_likes:- Total no of facebook likes for the movie

29. The dataset here gives the massive information about the movies and their IMDB scores respectively. We are going to analyze each and every factors which can influence the imdb ratings so that we can predict better results.The movie with the higher imdb score is more successful as compared to the movies with low imdb score



30.

NUMERICAL COLUMNS:

num_critic_for_reviews,duration,director_facebook_likes
,actor_3_facebook_likes,actor_1_facebook_likes
,gross,num_voted_users,cast_total_facebook_likes,facenumber_in_poster,num_user_for_reviews ,budget,title_year, actor_2_facebook_likes
,imdb_score,aspect_ratio,movie_facebook_like

```
director_name               True
num_critic_for_reviews      True
duration                    True
```

```
director_facebook_likes       True
actor_3_facebook_likes        True
actor_2_name                  True
actor_1_facebook_likes        True
gross                         True
genres                       False
actor_1_name                  True
movie_title                  False
num_voted_users              False
cast_total_facebook_likes    False
actor_3_name                  True
facenumber_in_poster          True
plot_keywords                 True
num_user_for_reviews          True
language                      True
country                       True
content_rating                True
budget                        True
title_year                    True
actor_2_facebook_likes        True
imdb_score                   False
aspect_ratio                  True
movie_facebook_likes         False
dtype: bool
```

```
movie_df.isna().sum()
```

```
director_name                104
num_critic_for_reviews        50
duration                      15
director_facebook_likes      104
actor_3_facebook_likes        23
actor_2_name                  13
actor_1_facebook_likes         7
gross                        884
genres                         0
actor_1_name                   7
movie_title                    0
num_voted_users                0
cast_total_facebook_likes      0
actor_3_name                  23
facenumber_in_poster          13
plot_keywords                153
num_user_for_reviews          21
language                      12
```

```
country                      5
content_rating             303
budget                     492
title_year                 108
actor_2_facebook_likes      13
imdb_score                   0
aspect_ratio               329
movie_facebook_likes         0
dtype: int64
```

```
movie_df.dropna(axis=0,subset=['director_name', 'num_critic_for_reviews','duration','director_facebook_likes','actor_3_facebook_likes','actor_2_name','actor_1_facebook_likes','actor_1_name','actor_3_name','facenumber_in_poster','num_user_for_reviews','language','country','actor_2_facebook_likes','plot_keywords'],inplace=True)
```

movie_df.shape

(4737, 26)

We lost only 6% of the data which is acceptable

```
#Replacing the content rating with Value R as it has highest frequency

movie_df["content_rating"].fillna("R", inplace = True)

#Replacing the aspect_ratio with the median of the value as the graph is right skewed

movie_df["aspect_ratio"].fillna(movie_df["aspect_ratio"].median(),inplace=True)
```

linkcode

```
#We need to replace the value in budget with the median of the value

movie_df["budget"].fillna(movie_df["budget"].median(),inplace=True)

# We need to replace the value in gross with the median of the value

movie_df['gross'].fillna(movie_df['gross'].median(),inplace=True)
```

```
# Recheck that all the null values are removed

movie_df.isna().sum()
```

```
director_name               0
num_critic_for_reviews      0


director_facebook_likes     0
actor_3_facebook_likes      0
actor_2_name                0
actor_1_facebook_likes      0
gross                       0
genres                      0
actor_1_name                0
movie_title                 0
num_voted_users             0
cast_total_facebook_likes   0
actor_3_name                0
facenumber_in_poster        0
plot_keywords               0
num_user_for_reviews        0
language                    0
country                     0
content_rating              0
budget                      0
title_year                  0
actor_2_facebook_likes      0
imdb_score                  0
aspect_ratio                0
movie_facebook_likes        0
dtype: int64
```

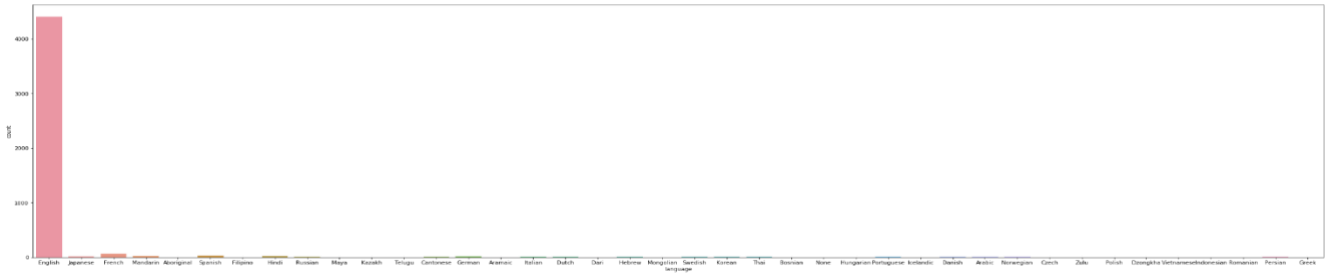
```
#Removing the duplicate values in the datset

movie_df.drop_duplicates(inplace=True)
movie_df.shape
Count of the language values
```

```
movie_df["language"].value_counts()
```

```
English       4405
French          69
Spanish         35
Hindi           25
Mandarin        24
German          18
Japanese        16
Russian         11
Italian         10
Cantonese       10
Portuguese       8
Korean           8
Danish           5
Norwegian        4
Swedish          4
Hebrew           4
Dutch            4
Persian          4
Arabic           3
Thai             3
Indonesian       2
None             2
Aboriginal       2
Dari             2
Zulu             2
Hungarian        1
Mongolian        1
Greek            1
Romanian         1
Bosnian          1
Telugu           1
Maya             1
Polish           1
Filipino         1
Czech            1
```

```
Dzongkha          1
Kazakh            1
Vietnamese        1
Icelandic         1
Aramaic           1


Name: language, dtype: int64
```

```
# Graphical presentaion
plt.figure(figsize=(40,10))
sns.countplot(movie_df["language"])
plt.show()
```



#Most of the values for the languages is english we can drop the english column

movie_df.drop('language',axis=1,inplace=True)

linkcode

#Creating a new column to check the net profit made by the company (Gross-Budget)

movie_df["Profit"]=movie_df['budget'].sub(movie_df['gross'], axis = 0)

```
value_counts=movie_df["country"].value_counts()
print(value_counts)
```

```
USA               3568
UK                 420
France             149
Canada             107
Germany             96
Australia           53
Spain               32
India               27
```

China	24
Japan	21
Italy	20
Hong Kong	16
New Zealand	14
South Korea	12
Ireland	11
Denmark	11
Russia	11
Mexico	11
South Africa	8
Brazil	8
Norway	7
Netherlands	5
Sweden	5
Thailand	4
Iran	4
Argentina	4
Czech Republic	3
Switzerland	3
Belgium	3
Israel	3
West Germany	3
Poland	2
Taiwan	2
Iceland	2
Romania	2
Hungary	2
Greece	2
Soviet Union	1
Slovakia	1
Finland	1
Official site	1
Turkey	1
Peru	1
Libya	1
Afghanistan	1
Cambodia	1
Indonesia	1
Nigeria	1
Kyrgyzstan	1
Colombia	1
New Line	1
Philippines	1

```
Bahamas              1
Bulgaria             1
Georgia              1
Aruba                1
Chile                1
Name: country, dtype: int64
```

We can see most of the movies are from USA, UK and the rest of the countries

##get top 2 values of index

```
##get top 2 values of index
vals = value_counts[:2].index
print (vals)
movie_df['country'] = movie_df.country.where(movie_df.country.isin(vals), '
other')
```

Index(['USA', 'UK'], dtype='object')

#Successfully divided the country into three catogories

movie_df["country"].value_counts()

USA 3568

other 707

UK 420

Name: country, dtype: int64

linkcode

movie_df.head(10)

/opt/conda/lib/python3.6/site-packages/plotnine/coords/coord_cartesian.py:31: MatplotlibDeprecation Warning: The Bunch class was deprecated in Matplotlib 3.0 and will be removed in 3.2. Use types.Simple Namespace instead.

self.limits = Bunch(xlim=xlim, ylim=ylim)

/opt/conda/lib/python3.6/copy.py:274: MatplotlibDeprecationWarning: The Bunch class was deprecated in Matplotlib 3.0 and will be removed in 3.2. Use types.SimpleNamespace instead.

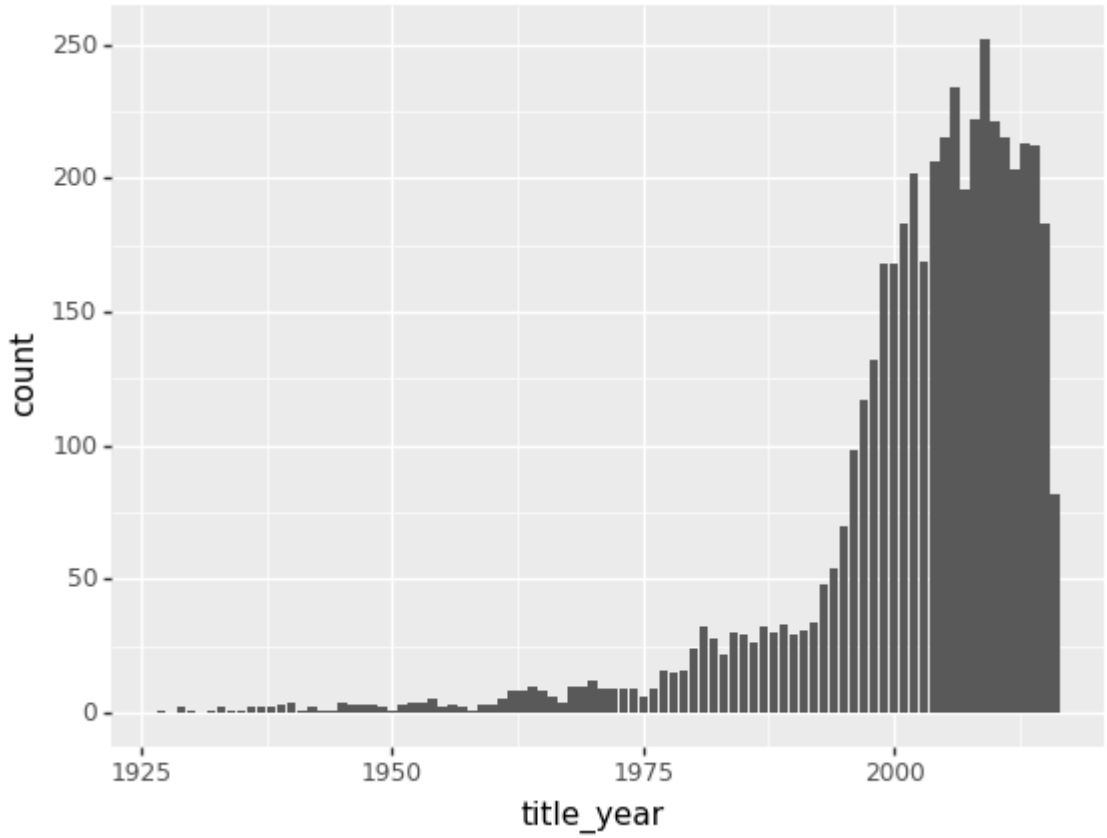
y = func(*args)

/opt/conda/lib/python3.6/site-packages/plotnine/facets/facet.py:151: MatplotlibDeprecationWarning: T he Bunch class was deprecated in Matplotlib 3.0 and will be removed in 3.2. Use types.SimpleNamespace instead.

scales = Bunch()

/opt/conda/lib/python3.6/site-packages/plotnine/facets/layout.py:147: MatplotlibDeprecationWarning: The Bunch class was deprecated in Matplotlib 3.0 and will be removed in 3.2. Use types.SimpleNamespac e instead.

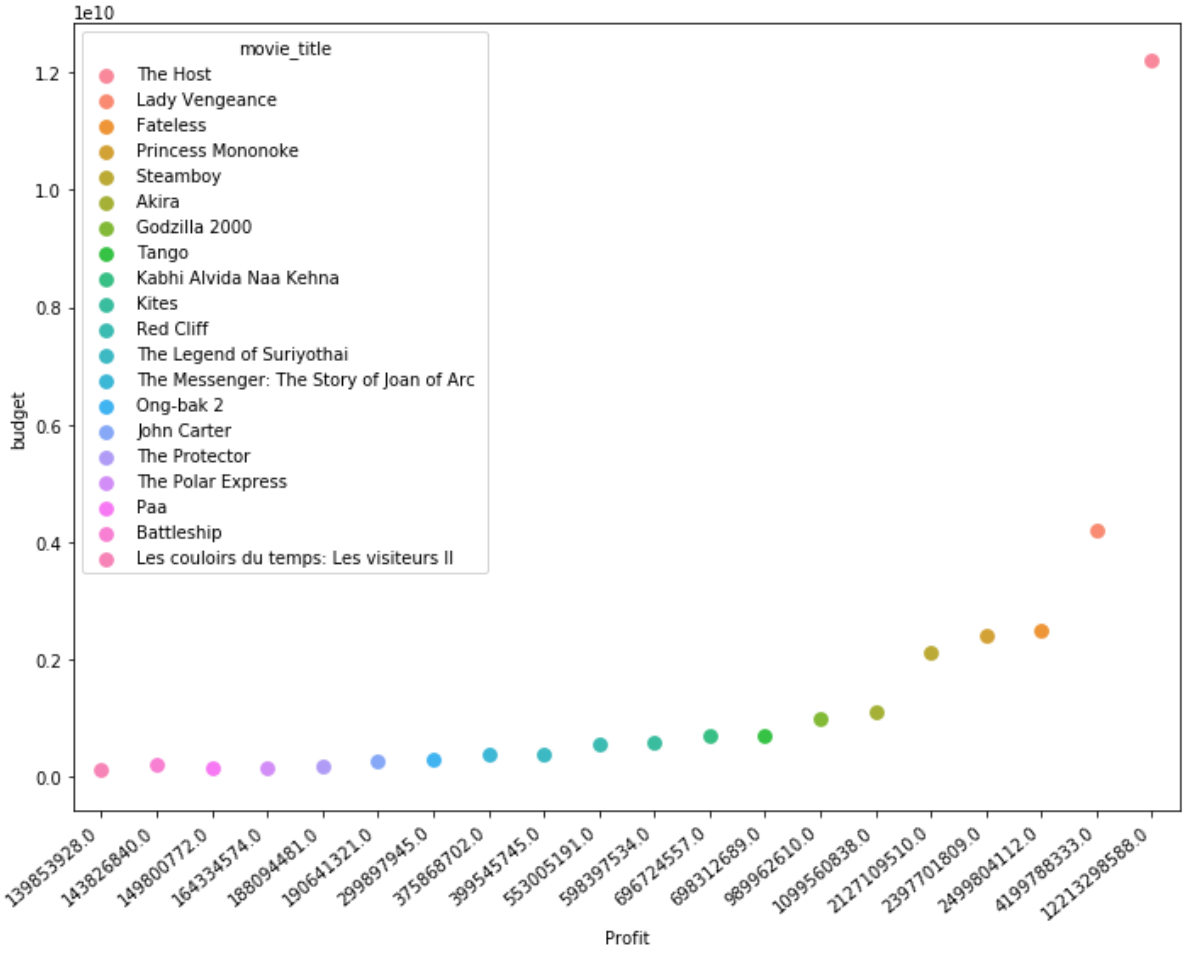
return Bunch(x=xsc, y=ysc)



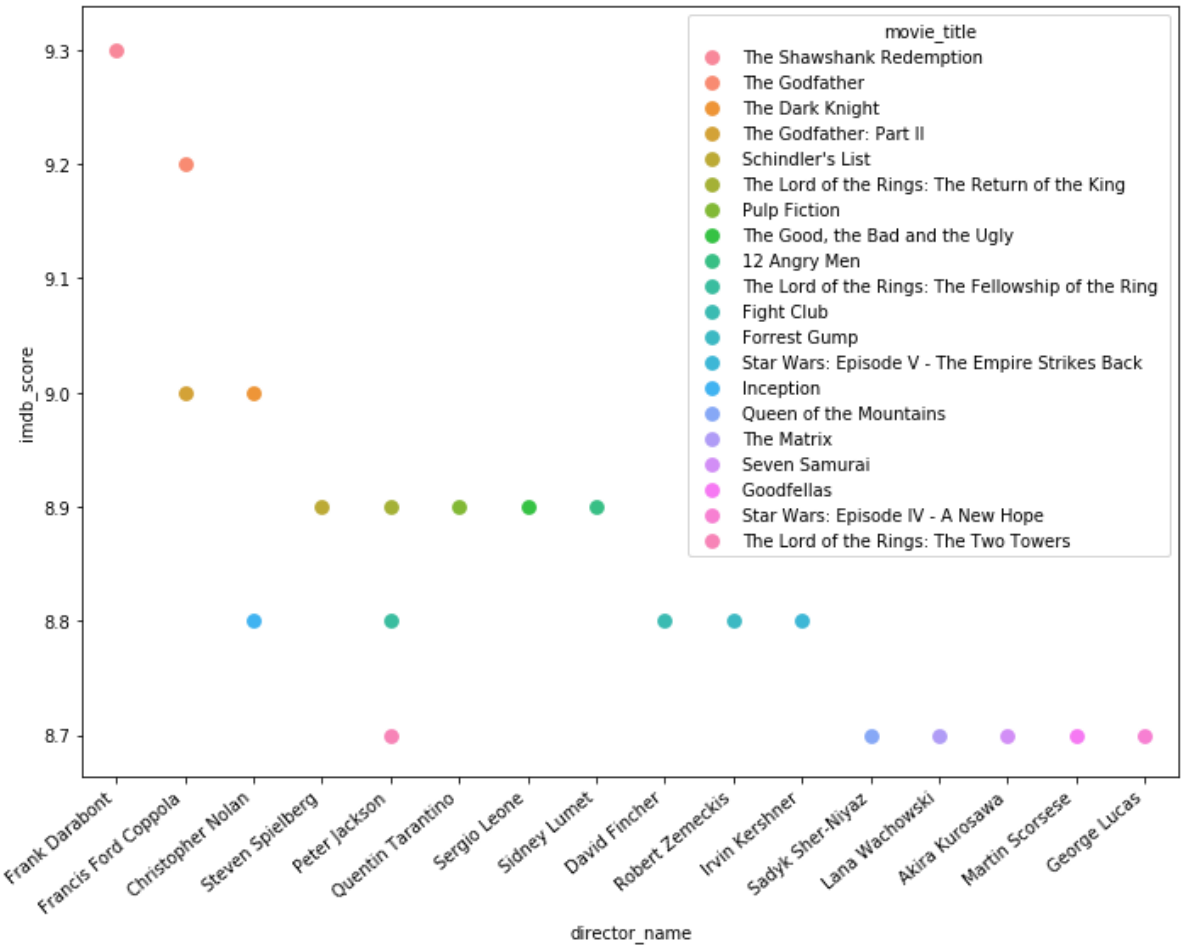

```
#Relationship between the imdb score and the profit made by the movie

ggplot(aes(x='imdb_score', y='Profit'), data=movie_df) +\
    geom_line() +\
    stat_smooth(colour='blue', span=1)
/opt/conda/lib/python3.6/site-packages/plotnine/coords/coord_cartesian.py:3
1: MatplotlibDeprecationWarning: The Bunch class was deprecated in Matplotl
ib 3.0 and will be removed in 3.2. Use types.SimpleNamespace instead.
  self.limits = Bunch(xlim=xlim, ylim=ylim)
/opt/conda/lib/python3.6/copy.py:274: MatplotlibDeprecationWarning: The Bun
ch class was deprecated in Matplotlib 3.0 and will be removed in 3.2. Use t
ypes.SimpleNamespace instead.
  y = func(*args)
/opt/conda/lib/python3.6/site-packages/plotnine/facets/facet.py:151: Matplo
tlibDeprecationWarning: The Bunch class was deprecated in Matplotlib 3.0 an
d will be removed in 3.2. Use types.SimpleNamespace instead.
  scales = Bunch()
/opt/conda/lib/python3.6/site-packages/plotnine/facets/layout.py:147: Matpl
otlibDeprecationWarning: The Bunch class was deprecated in Matplotlib 3.0 a
nd will be removed in 3.2. Use types.SimpleNamespace instead.
  return Bunch(x=xsc, y=ysc)
```

```
/opt/conda/lib/python3.6/site-packages/numpy/core/fromnumeric.py:2389: Futu
reWarning: Method .ptp is deprecated and will be removed in a future versio
n. Use numpy.ptp instead.
```

return ptp(axis=axis, out=out, **kwargs)

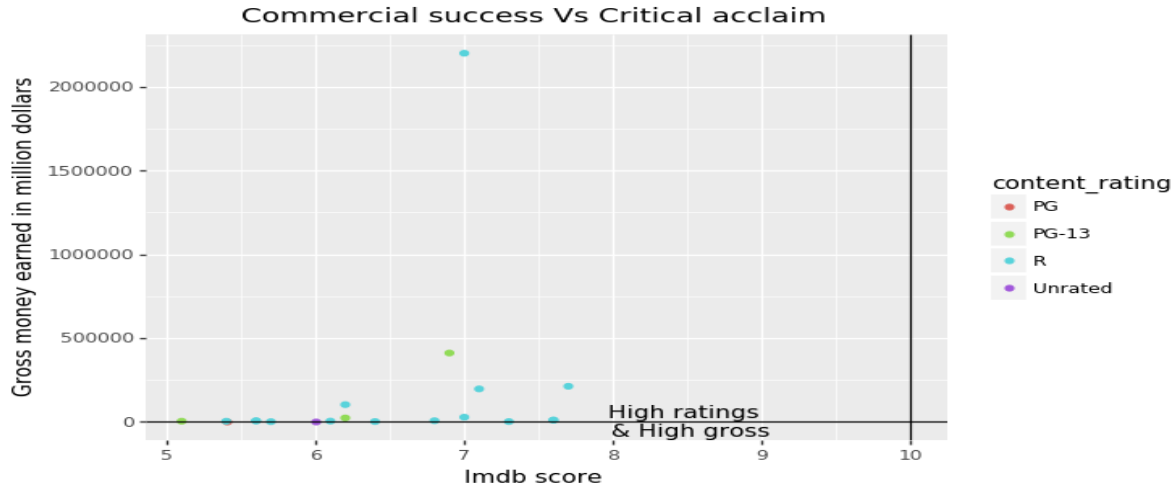



```
# Top 20 movies based on the profit percentage
plt.figure(figsize=(10,8))
movie_df= movie_df.sort_values(by ='Profit_Percentage' , ascending=False)
movie_df_new=movie_df.head(20)
ax=sns.pointplot(movie_df_new['Profit_Percentage'], movie_df_new['budget'],
 hue=movie_df_new['movie_title'])
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```



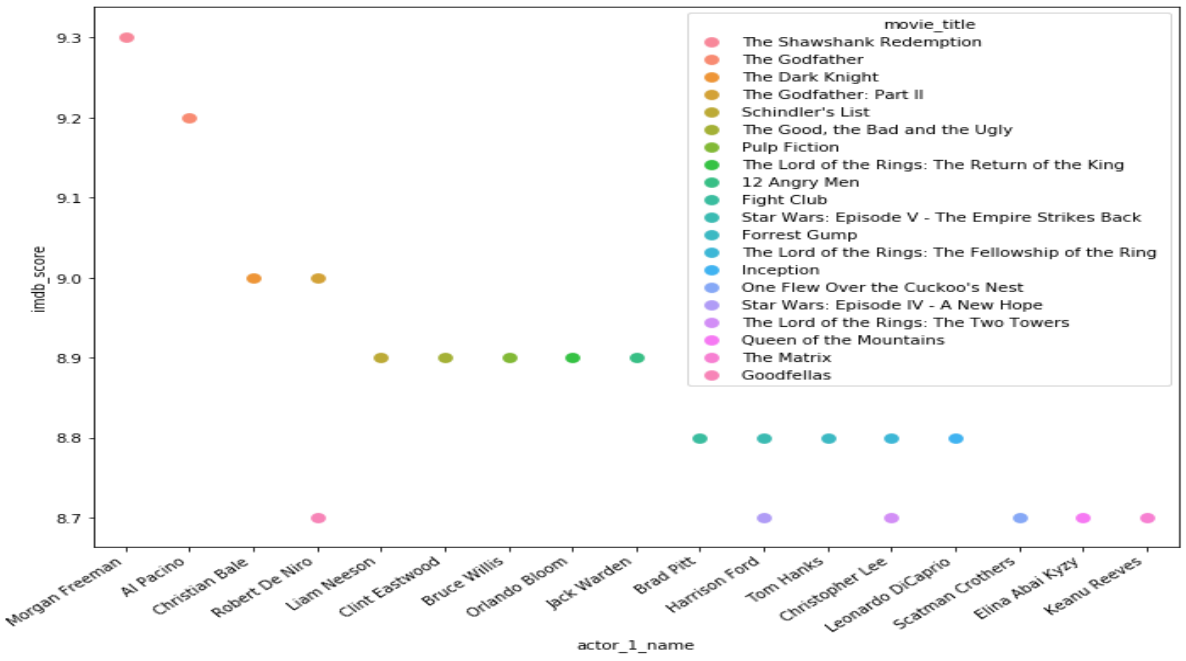

```
#Commercial success vs critial acclaim
movie_df= movie_df.sort_values(by ='Profit_Percentage' , ascending=False)
movie_df_new=movie_df.head(20)
(ggplot(movie_df_new)
 + aes(x='imdb_score', y='gross',color = "content_rating")
 + geom_point()
 +  geom_hline(aes(yintercept = 600)) +
  geom_vline(aes(xintercept = 10)) +
  xlab("Imdb score") +
  ylab("Gross money earned in million dollars") +
  ggtitle("Commercial success Vs Critical acclaim") +
  annotate("text", x = 8.5, y = 700, label = "High ratings \n & High gross"
))
/opt/conda/lib/python3.6/site-packages/plotnine/coords/coord_cartesian.py:3
1: Mat
/opt/conda/lib/python3.6/site-packages/plotnine/coords/coord_cartesian.py:3
1: MatplotlibDeprecationWarning: The Bunch class was deprecated in Matplotl
ib 3.0 and will be removed in 3.2. Use types.SimpleNamespace instead.
  self.limits = Bunch(xlim=xlim, ylim=ylim)
```

```
/opt/conda/lib/python3.6/copy.py:274: MatplotlibDeprecationWarning: The Bun
ch class was deprecated in Matplotlib 3.0 and will be removed in 3.2. Use t
ypes.SimpleNamespace instead.
  y = func(*args)
/opt/conda/lib/python3.6/site-packages/plotnine/layer.py:520: MatplotlibDep
recationWarning: isinstance(..., numbers.Number)
  return not cbook.iterable(value) and (cbook.is_numlike(value) or
/opt/conda/lib/python3.6/site-packages/plotnine/facets/facet.py:151: Matplo
tlibDeprecationWarning: The Bunch class was deprecated in Matplotlib 3.0 an
d will be removed in 3.2. Use types.SimpleNamespace instead.
  scales = Bunch()
/opt/conda/lib/python3.6/site-packages/plotnine/facets/layout.py:147: Matpl
otlibDeprecationWarning: The Bunch class was deprecated in Matplotlib 3.0 a
nd will be removed in 3.2. Use types.SimpleNamespace instead.
  return Bunch(x=xsc, y=ysc)
/opt/conda/lib/python3.6/site-packages/plotnine/coords/coord.py:144: Matplo
tlibDeprecationWarning: The Bunch class was deprecated in Matplotlib 3.0 an
d will be removed in 3.2. Use types.SimpleNamespace instead.
  y=panel_params['y_range'])
/opt/conda/lib/python3.6/site-packages/plotnine/guides/guide_legend.py:179:
 MatplotlibDeprecationWarning: The Bunch class was deprecated in Matplotlib
 3.0 and will be removed in 3.2. Use types.SimpleNamespace instead.
  self.glayers.append(Bunch(geom=geom, data=data, layer=l))
```



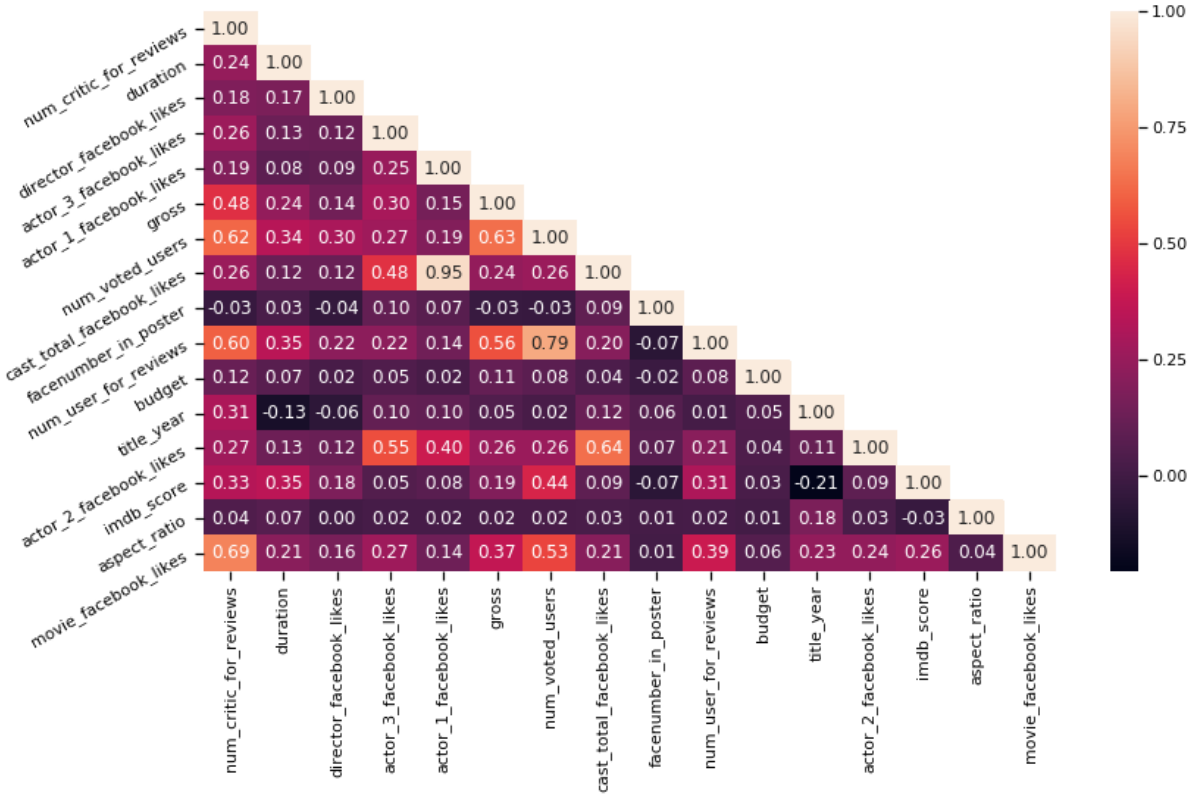
Top 20 actors of movies based on the commerical success

```
plt.figure(figsize=(10,8))

movie_df= movie_df.sort_values(by ='Profit_Percentage' , ascending=False)
movie_df_new=movie_df.head(20)
ax=sns.pointplot(movie_df_new['actor_1_name'], movie_df_new['Profit_Percent
age'], hue=movie_df_new['movie_title'])
ax.set_xticklabels(ax.get_xticklabels(), rotation=40, ha="right")
plt.tight_layout()
plt.show()
```




```
# Correlation with heat map
import matplotli.pyplot as plot
import seaborn as sns
corr = movie_df.corr()
```

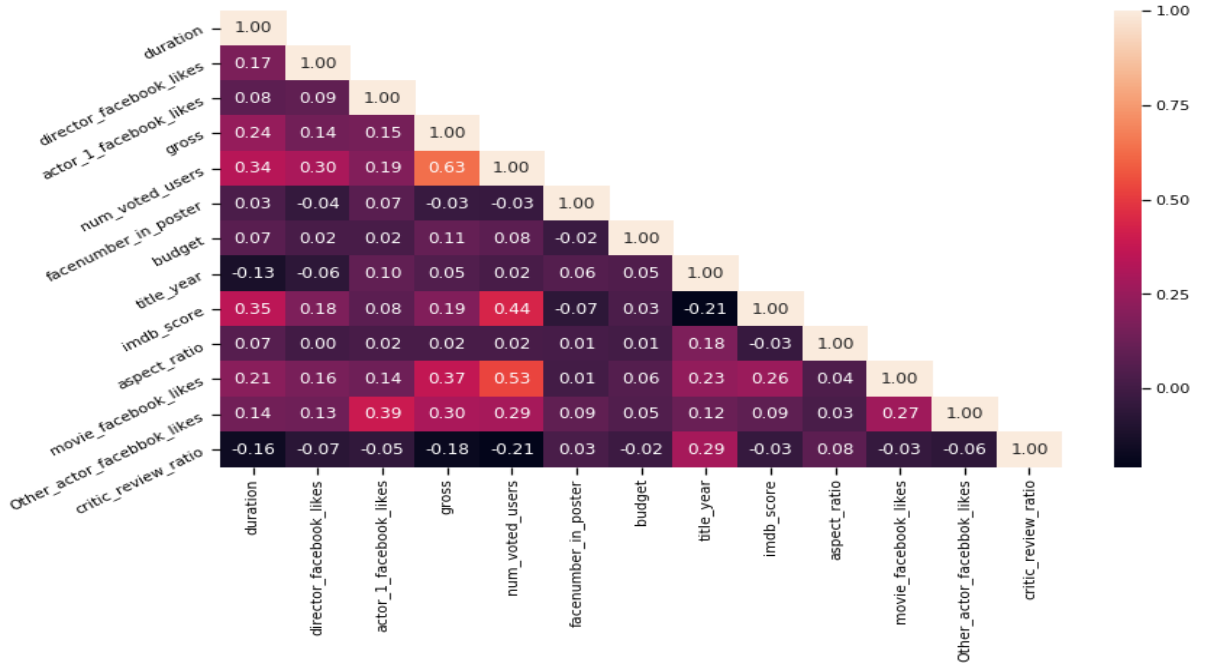
```
sns.set_context("notebook", font_scale=1.0, rc={"lines.linewidth": 2.5})
plt.figure(figsize=(13,7))
# create a mask so we only see the correlation values once
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, 1)] = True
a = sns.heatmap(corr,mask=mask, annot=True, fmt='.2f')
rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```



We can see that the cast_total_facebook_likes and actor_1_facebook_like are highly correlated to each other. Both actor2 and actor3 are also somehow correlated to the total. So we want to modify them into two variables: actor_1_facebook_likes and other_actors_facebook_likes.
There are high correlations among num_voted_users, num_user_for_reviews and num_critic_for_reviews. We want to keep num_voted_users and take the ratio of num_user_for_reviews and num_critic_for_reviews.

```
# New Correlation matrix shown in the figure

import matplotlib.pyplot as plot
import seaborn as sns
```

```
corr = movie_df.corr()
sns.set_context("notebook", font_scale=1.0, arc={"lines.linewidth": 2.5})
plt.figure(figsize=(13,7))
# create a mask so we only see the correlation values once
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask, 1)] = True
a = sns.heatmap(corr,mask=mask, annot=True, fts='.2f')
rotx = a.set_xticklabels(a.get_xticklabels(), rotation=90)
roty = a.set_yticklabels(a.get_yticklabels(), rotation=30)
```




```
from sklearn.metrics import classification_report

print('Logistic  Reports\n',classification_report(y_test, y_pred))
print('KNN Reports\n',classification_report(y_test, knnpred))
print('SVC Reports\n',classification_report(y_test, svcpred))
print('Naive BayesReports\n',classification_report(y_test, gaussiannbpred))
print('Decision Tree Reports\n',classification_report(y_test, dtreepred))
print('Ada Boosting\n',classification_report(y_test, abcl_pred))
print('Random Forests Reports\n',classification_report(y_test, rfcpred))
print('Bagging Clasifier',bgcl.oob_score_)
print('Gradient Boosting',classification_report(y_test, test_pred))
print('XGBoosting\n',classification_report(y_test, xgbprd))
```

logistIC

Reports

	precision	recall	f1-score	support
1	0.00	0.00	0.00	46
2	0.50	0.25	0.33	378
3	0.72	0.92	0.81	924
4	0.84	0.52	0.65	61
accuracy			0.69	1409
macro avg	0.52	0.42	0.45	1409
weighted avg	0.64	0.69	0.65	1409

KNN Reports

	precision	recall	f1-score	support
1	0.00	0.00	0.00	46
2	0.46	0.41	0.44	378
3	0.73	0.83	0.78	924
4	1.00	0.20	0.33	61
accuracy			0.67	1409
macro avg	0.55	0.36	0.39	1409
weighted avg	0.64	0.67	0.64	1409

SVC Reports

	precision	recall	f1-score	support
1	0.14	0.02	0.04	46
2	0.42	0.42	0.42	378
3	0.74	0.79	0.76	924
4	0.57	0.33	0.42	61
accuracy			0.64	1409
macro avg	0.47	0.39	0.41	1409
weighted avg	0.62	0.64	0.63	1409

Naive BayesReports

	precision	recall	f1-score	support

1	0.05	0.91	0.09	46
2	0.50	0.00	0.01	378
3	0.71	0.01	0.01	924
4	0.11	0.85	0.19	61
accuracy			0.07	1409
macro avg	0.34	0.44	0.07	1409
weighted avg	0.61	0.07	0.02	1409

Decision Tree Reports

	precision	recall	f1-score	support
1	0.11	0.11	0.11	46
2	0.47	0.51	0.49	378
3	0.78	0.76	0.77	924
4	0.77	0.59	0.67	61
accuracy			0.67	1409
macro avg	0.53	0.49	0.51	1409
weighted avg	0.67	0.67	0.67	1409

Ada Boosting

	precision	recall	f1-score	support
1	0.17	0.15	0.16	46
2	0.46	0.51	0.48	378
3	0.77	0.75	0.76	924
4	0.65	0.51	0.57	61
accuracy			0.65	1409
macro avg	0.51	0.48	0.49	1409
weighted avg	0.66	0.65	0.66	1409

Random Forests Reports

	precision	recall	f1-score	support
1	1.00	0.04	0.08	46
2	0.62	0.47	0.53	378
3	0.77	0.92	0.84	924
4	0.96	0.44	0.61	61
accuracy			0.75	1409
macro avg	0.84	0.47	0.52	1409
weighted avg	0.75	0.75	0.72	1409

```
Bagging Clasifier 0.7429179978700745
Gradient Boosting               precision    recall  f1-score   support

           1       0.25      0.02      0.04        46
           2       0.60      0.56      0.58       378
           3       0.80      0.88      0.84       924
           4       0.86      0.49      0.62        61

    accuracy                           0.75      1409
   macro avg       0.63      0.49      0.52      1409
weighted avg       0.73      0.75      0.74      1409



XGBoosting
              precision    recall  f1-score   support

           1       0.25      0.02      0.04        46
           2       0.59      0.52      0.55       378
           3       0.79      0.89      0.84       924
           4       0.89      0.54      0.67        61

    accuracy                           0.75      1409
   macro avg       0.63      0.49      0.53      1409
weighted avg       0.72      0.75      0.73      1409
```

opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1437: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples.
'precision', 'predicted', average, warn_for

FEATURE ENGINEERING:

1. **Genre Encoding**: Create binary features for each genre. For example, if a movie belongs to Action, Drama, and Comedy genres, you can create binary features like `Action`, `Drama`, and `Comedy`. If the movie is of the corresponding genre, the feature gets a 1; otherwise, it gets a 0.

2. **Director/Actor Features**: Consider creating features based on the directors and actors involved in the movie. You can calculate the average IMDb rating of the director's previous works, the number of award nominations/wins for the actors, etc.

3. **Release Date Features**: Extract information from the release date, such as the year, month, or season. Certain periods may influence IMDb scores differently.

4. **Runtime**: The length of a movie can affect its IMDb score. You can create features like 'Short Film' or 'Long Film' based on certain runtime thresholds.

5. **Budget and Box Office Features**: Include financial metrics like production budget and box office revenue. High budget movies may be expected to have higher IMDb scores.

6. **Categorical Features**: Include other categorical features like the movie's language, country of origin, and MPAA rating. These can affect audience perceptions.

7. **Word Count in Description**: If you have access to the movie's description or plot summary, you can create a feature based on the word count. Longer descriptions might provide more information, potentially influencing ratings.

8. **Sentiment Analysis**: Analyse the sentiment of user reviews for the movie. You can use Natural Language Processing (NLP) techniques to calculate sentiment scores and include them as features.

9. **User Review Statistics**: Aggregate user review statistics, such as the number of reviews, average user rating, and the standard deviation of user ratings.

10. **Social Media Presence**: If available, you can include features related to the movie's social media presence, such as the number of Facebook likes, Twitter followers, or YouTube views.

11. **Awards and Nominations**: Create features based on the number of awards and nominations a movie has received.

12. **User Ratings of Actors and Directors**: If available, you can incorporate user ratings of the movie's lead actors and director from platforms like IMDb.

13. **Time-Based Features**: Consider features related to when the movie was released, such as holidays or notable events.

14. **Composite Features**: Create composite features that combine relevant factors. For example, a "Hollywood Blockbuster" feature could combine high budget, well-known actors, and a wide release.

15. **User Demographics**: If available, consider features related to the demographics of IMDb users who have rated the movie, such as their age, gender, or location.

```
Import pandas as pd

# Sample data
Data = {
  ‘user_reviews’: [100, 200, 300, 150, 250],
  ‘critic_reviews’: [80, 90, 70, 85, 95],
  ‘budget_in_million’: [10, 20, 15, 30, 25]
```

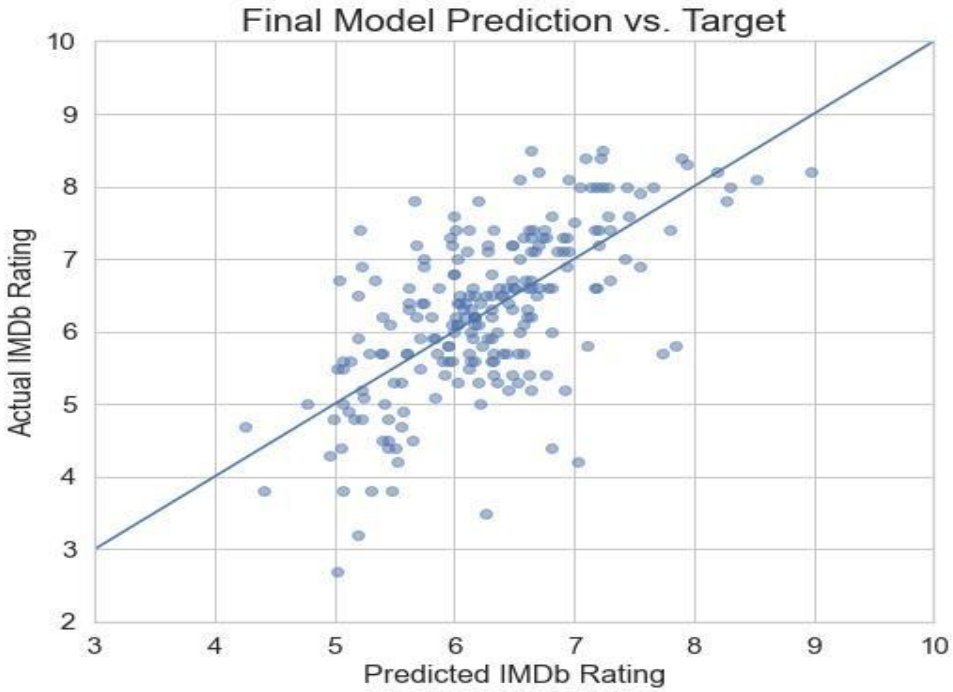
}

Df = pd.DataFrame(data)

Feature engineering
Df[‘user_critic_review_ratio’] = df[‘user_reviews’] / df[‘critic_reviews’]
Df[‘budget_per_review’] = df[‘budget_in_million’] / (df[‘user_reviews’] + df[‘critic_reviews’])

You can add more features based on your domain knowledge

Calculating IMDb score (This is a simple linear combination, actual IMDb score calculation is more complex)
Df[‘imdb_score’] = 6.5 + 0.01 * df[‘user_reviews’] + 0.02 * df[‘critic_reviews’] – 0.1 *



MODEL TRAINING:

1**. Data Collection:**

- Gather a dataset of movies or TV shows with their corresponding IMDb scores. You can find IMDb datasets online or use APIs to collect this data.

2. **Data Preprocessing:**

Clean the data by handling missing values, removing duplicates, and dealing with outlier Feature engineering: Extract relevant features such as the movie's genre, director, actors, release year, and more.

3. **Data Splitting:**

- Split the dataset into training, validation, and test sets. Typically, you might use an 80-10-10 or 70-15-15 split.

4. **Feature Encoding:**

- Convert categorical features like genres, directors, and actors into numerical representations, e.g., one-hot encoding or embeddings.

5. **Model Selection:**

- Choose an appropriate model architecture for regression. Common choices include linear regression, decision trees, random forests, gradient boosting, or neural networks.

6. **Model Training:**

- Train the selected model on the training data. You'll use the features (independent variables) to predict IMDb scores (the dependent variable).

Tune hyperparameters to optimize model performance on the validation set.

7. **Model Evaluation**:

- Evaluate the model's performance on the validation set using appropriate metrics (e.g., mean squared error, mean absolute error, or R-squared).
- Make necessary adjustments to the model based on the evaluation results.

8. **Hyperparameter Tuning:**

- Experiment with different hyperparameters to improve model performance. This may involve grid search or random search.

9. **Final Model Evaluation**:

- Once the model performs well on the validation set, evaluate it on the test set to get an unbiased estimate of its performance.

10. **Model Deployment:**

- If you're satisfied with the model's performance, you can deploy it for predicting IMDb scores for new movies or shows.

11. **Regular Maintenance:**

- Keep the model up-to-date with new data and retrain it periodically to maintain its accuracy.

```
# Import necessary libraries
Import numpy as np
Import pandas as pd
From sklearn.model_selection import train_test_split
From sklearn.feature_extraction.text import TfidfVectorizer
From sklearn.linear_model import LinearRegression
From sklearn.metrics import mean_squared_error

# Load your IMDb dataset (replace ‘your_dataset.csv’ with your actual dataset)
Data = pd.read_csv(‘your_dataset.csv’)
```

# Assuming you have a ‘text’ column for movie reviews and a ‘score’ column for IMDb scores
X = data[‘text’]
Y = data[‘score’]

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

# Create a TF-IDF vectorizer for text data
Tfidf_vectorizer = TfidfVectorizer(max_features=5000) # You can adjust max_features as needed
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
X_test_tfidf = tfidf_vectorizer.transform(X_test)

# Initialize and train a linear regression model
Model = LinearRegression()
Model.fit(X_train_tfidf, y_train)

Modality	**Representation**	**F-Score**			
		weighted	samples	micro	macro
Multimodal	GMU	**0.617**	**0.630**	**0.630**	**0.541**
	Linear_sum	0.600	0.607	0.607	0.530
	Concatenate	0.597	0.605	0.606	0.521
	AVG_probs	0.604	0.616	0.615	0.491
	MoE_MaxoutMLP	0.592	0.593	0.601	0.516
	MoE_MaxoutMLP (tied)	0.579	0.579	0.587	0.489
	MoE_Logistic	0.541	0.557	0.565	0.456
	MoE_Logistic (tied)	0.483	0.507	0.518	0.358
Text	MaxoutMLP_w2v	0.588	0.592	0.595	0.488
	RNN_transfer	0.570	0.580	0.580	0.480
	MaxoutMLP_w2v_1_hidden	0.540	0.540	0.550	0.440
	Logistic_w2v	0.530	0.540	0.550	0.420
	MaxoutMLP_3grams	0.510	0.510	0.520	0.420
	Logistic_3grams	0.510	0.520	0.530	0.400
	RNN_end2end	0.490	0.490	0.490	0.370
Visual	VGG_Transfer	0.410	0.429	0.437	0.284
	CNN_end2end	0.370	0.350	0.340	0.210

# Make predictions on the test set
Y_pred = model.predict(X_test_tfidf)

# Calculate the Mean Squared Error to evaluate the model
Mse = mean_squared_error(y_test, y_pred)
Print(f”Mean Squared Error: {mse}”

EVALUATION :

1. **User-Generated Ratings**: IMDb scores are generated by users who rate movies on the platform. These ratings are based on personal opinions, and the scores are an aggregate of all user ratings. Keep in mind that IMDb scores are subjective and represent the collective opinion of IMDb users.

2. **Sample Size**: The reliability of an IMDb score depends on the number of ratings and reviews. A movie with a very high or very low rating but only a handful of votes may not accurately represent its quality. It's often better to rely on movies with a substantial number of ratings for a more reliable evaluation.

3. **Distribution of Ratings**: Pay attention to the distribution of ratings. A movie with a high average rating may have a different distribution than one with a lower average. A movie with a rating of 8.0 based on a mix of 9s and 1s may be more polarizing than a movie with a consistent rating of 8.0.

4**. Read User Reviews**: IMDb provides user reviews along with ratings. Reading these reviews can provide valuable insights into why a movie received a particular rating. It can help you understand the strengths and weaknesses of the film from different perspectives.

5**. Genre and Personal Preferences**: IMDb scores should be evaluated in the context of the genre and your personal preferences. A highly-rated horror film may not be as enjoyable to someone who dislikes horror. Consider your own tastes and how they align with the genre and theme of the movie.

6. **Historical Context**: IMDb scores can change over time. Older movies may have accumulated ratings over many years, while newer movies may have fewer ratings. Consider the historical context and whether the movie's reception has changed over time.

7. **Professional Critics vs. Audience Scores**: IMDb represents audience opinions, whereas professional critics often have their own ratings and reviews on platforms like Rotten Tomatoes. It can be informative to compare IMDb scores with critical reviews to get a more complete picture of a movie's reception.

8. **Box Office vs. IMDb Score:** A movie's financial success at the box office does not always correlate with its IMDb score. Some critically acclaimed films may not have been commercial hits, and vice versa.

9. **Use IMDb as a Tool, Not the Sole Criterion**: While IMDb scores can be a useful reference, they should not be the sole criterion for judging a movie's quality. It's important to watch the movie and form your own opinion, as your tastes may differ from the IMDb community

10. **Bias and Manipulation**: Be aware of potential biases, trolling, or manipulation of ratings. Sometimes, people may intentionally inflate or deflate a movie's score for various reasons.

```
# Sample IMDb ratings data
Ratings = {
   “movie1”: 8.5,
   “movie2”: 7.2,
   “movie3”: 9.0,
   # Add more movies and ratings as needed
}

# Weighted average calculation
Total_weighted_rating = 0
Total_weight = 0

For movie, rating in ratings.items():
   Weight = rating  # You can customize the weight calculation
   Total_weighted_rating += rating * weight
   Total_weight += weight

If total_weight == 0:
   Imdb_score = 0  # Avoid division by zero
```

```
Else:
   Imdb_score = total_weighted_rating / total_weight

Print("IMDb Score:", imdb_score)
```

CONCLUSION:

In conclusion, the innovation in IMDb score prediction through applied data science represents a significant advancement in the film industry. This approach combines cutting-edge algorithms, real-time capabilities, and user-friendly interfaces to offer highly accurate and interpretable IMDb score predictions. By fostering transparency, personalization, and ethical practices, it not only aids filmmakers and studios in decision-making but also empowers movie enthusiasts with tailored recommendations. The continual learning aspect ensures predictions stay relevant, while open-source collaboration encourages the data science community's contributions. Ultimately, this innovation revolutionizes how we engage with movies, creating a more informed, personalized, and enjoyable movie-watching experience for audiences.

Done by:

Shaik Asif

AU720921244047

JCT COLLEGE OF ENGINEERING AND TECHNOLOGY