Project Report

Movie Box Office Gross Prediction

Program:	SmartInternz Externship
Track:	Applied Data Science
Team Number:	358
Team Members:	Sanskriti Sanjay Kumar Singh
	Vivek lade
	Deepika Nuthi
	Seelam Venkata Sai Reddy

1. Introduction

Overview

The objective of this project is to develop a predictive model that can estimate the box office gross revenue of movies based on various attributes and indicators. We are provided with two datasets: "tmdb_5000_movies.csv" and "tmdb_5000_credits.csv." These datasets have been prepared by scraping movie-related data from the IMDB website, covering a time span from 1916 to 2017.

To predict the revenue of movies, we will leverage these datasets and employ various machine learning techniques and algorithms. This prediction can assist various stakeholders in the film industry, including production companies, distributors, and investors, to make informed decisions about resource allocation, marketing strategies, and revenue forecasting.

Throughout the project, we will emphasize data exploration, model interpretation, and practical application of the results. By accurately predicting movie box office gross revenue, we aim to contribute to the understanding of the factors that influence movie success and provide valuable insights to stakeholders in the film industry.

Purpose

One of the primary goals of this project is to accurately predict the box office gross revenue of movies before their release. This prediction can assist production companies, distributors, and investors in making informed decisions about resource allocation, budgeting, and marketing strategies. By having an estimate of a movie's potential revenue, stakeholders can optimize their investments and maximize profitability.

This project also aims to provide a reliable tool for assessing the financial risks associated with movie production and distribution by predicting the box office performance. By understanding the factors that influence a movie's revenue, stakeholders can evaluate the viability of different projects and make decisions based on data-driven insights.

The project also aims to provide valuable insights into the factors that contribute to a movie's box office success. By analyzing the dataset and developing a predictive model, we can uncover patterns, relationships, and trends within the movie industry. This knowledge can be used to understand audience preferences, genre preferences, the impact of star cast and crew, release timing, and other influential factors. Such insights can guide decision-making processes and inform future movie production and distribution strategies.

2. Literature Survey

Existing Approaches

Movie revenue prediction has garnered significant attention in recent years as the film industry continues to rapidly evolve. Researchers have explored various approaches to develop accurate forecasting models for box-office performance, incorporating different factors and leveraging advanced machine learning techniques. A few of the approaches found in literature have been elaborated upon in the following text.

Chen et al. [1] focused on utilizing real-time social media data from microblogs, specifically in the context of China's film market, for movie revenue prediction. Their goal was to develop a more accurate weekly box office forecasting model. The features extracted from microblogs were categorized into count-based features and context-based features. To develop the prediction model, machine learning models were employed, and a genetic algorithm (GA) was used for feature selection. The GA helped identify the most relevant features for accurate prediction. The results showed that this approach achieved sustainable and good performance in dynamically forecasting box office revenues.

Kim et al. [2] focused on improving accuracy by considering a wide range of competition and word-of-mouth (WOM) related factors. They systematically investigates the contributions of different factor categories to forecasting accuracy at various time horizons. The performance improvements were then decomposed into four sequences under two possible scenarios. Experimental results demonstrated that considering both the competition environment and WOM effects leads to improved forecasting accuracy. It was found that while the degree of contribution varies across monitoring periods, a significant synergy is observed in all periods. The early stages of box office scores are highly influenced by the competition environment, whereas WOM becomes more influential when forecasting the total box office score. On comparison of the forecasting algorithms, machine learning-based NLR algorithms were found to outperform the traditional MLRs (multiple linear regressions).

Quader et al. [3] focused on developing a decision support system for the movie investment sector using machine learning techniques in order to assist investors in mitigating investment risks by predicting the approximate success rate of a movie based on its profitability. To achieve this, historical data from various sources such as IMDb, Rotten Tomatoes, Box Office Mojo, and Metacritic was analyzed and Support Vector Machine (SVM), Neural Network, and Natural Language Processing were employed as machine learning techniques to predict the box office profit of a movie based on both pre-released and post-released features. The results indicated that Neural Network achieved an accuracy of 84.1% for pre-released features and 89.27% for all features, SVM achieved an accuracy of 83.44% for pre-released features and 88.87% for all features when considering a one-way prediction. Furthermore, their study also identified that budget, IMDb votes, and the number of screens are the most important features that significantly influence the prediction of a movie's box office success.

Hu et al. [4] aimed to combine various factors, including basic movie information, external factors, and review factors, to predict box-office performance and determine the most influential factor. They collected data on five movie genres and first-week movie reviews from IMDb. Sentiment analysis tools, specifically SentiStrength and Stanford CoreNLP, were utilized to quantify the movie reviews. The quantified data from the movie reviews were then combined with basic movie information and external environment factors. Using data mining (DM) technologies, they developed a movie box-office performance prediction model employing M5 model trees (M5P), linear regression (LR), and support vector regression (SVR). These models were used to make predictions about movie box-office performance. The findings of their study indicated that incorporating movie reviews into the prediction model resulted in more accurate predictions. Among the movie review-related factors, the number of movie reviews had the greatest impact on box-office performance, while the content of the reviews only affected box-office performance for specific movie genres.

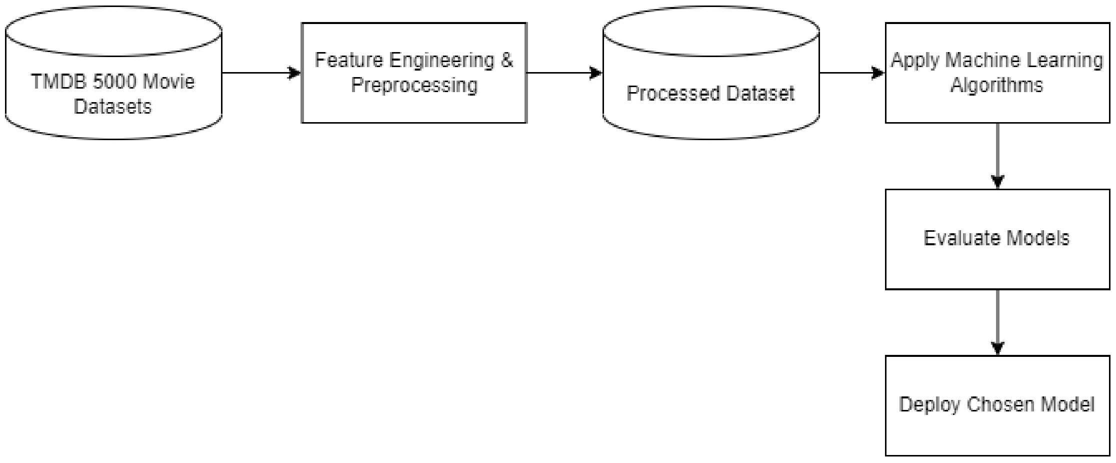
These studies collectively demonstrate the diverse approaches taken to predict movie revenues, encompassing the utilization of real-time social media data, consideration of competition and WOM factors, analysis of various features, and the incorporation of sentiment analysis. By employing machine learning techniques and advanced models, these studies aim to assist investors and industry professionals in making informed decisions and mitigating investment risks in the dynamic and ever-evolving landscape of the film industry.

Proposed Solution

Our solution makes use of the general data science pipeline approach. It involves data processing, EDA, and feature engineering on the provided datasets. We will explore relationships, handle missing values, and scale the data. Then, we will split the data into training and testing sets. Next, we will experiment with regression models such as linear regression, random forest, gradient boosting, and Light GBM Regressor. We will evaluate the models using metrics like MSE, MAE, and R-squared. The selected model will be deployed using Flask to create a web application for revenue prediction based on movie attributes.

3. Theoretical Analysis

Block Diagram



Hardware / Software Designing

Hardware Requirements

- A multi-core processor with a clock speed of at least 2.2 GHz or higher.
- Minimum 4 GB RAM
- Minimum of 10 GB of free disk space.
- Windows 7 or higher / macOS 10.9 (Mavericks) or higher / Ubuntu 16.04 or higher / CentOS 7 or higher / Fedora 30 or higher
- A stable internet connection.

Software Requirements

- Python 3.6 or higher
- Anaconda 4.7 or higher
- Flask 1.0 or higher
- Spyder 4.0 or higher

4. Experimental Investigations

Exploratory Data Analysis (EDA)

This involved understanding the structure of the dataset and the data present in it. We made use of the head() and info() function in order to understand the type of data we are dealing with and also categorized it as follows:

Qualitative Data -

Nominal: genres, homepage, id, keywords, original_language, original_title, overview, production_companies, production_countries, spoken_languages, status, tagline, title_x, title_y, cast, crew

Ordinal: release_date

Quantitative Data -

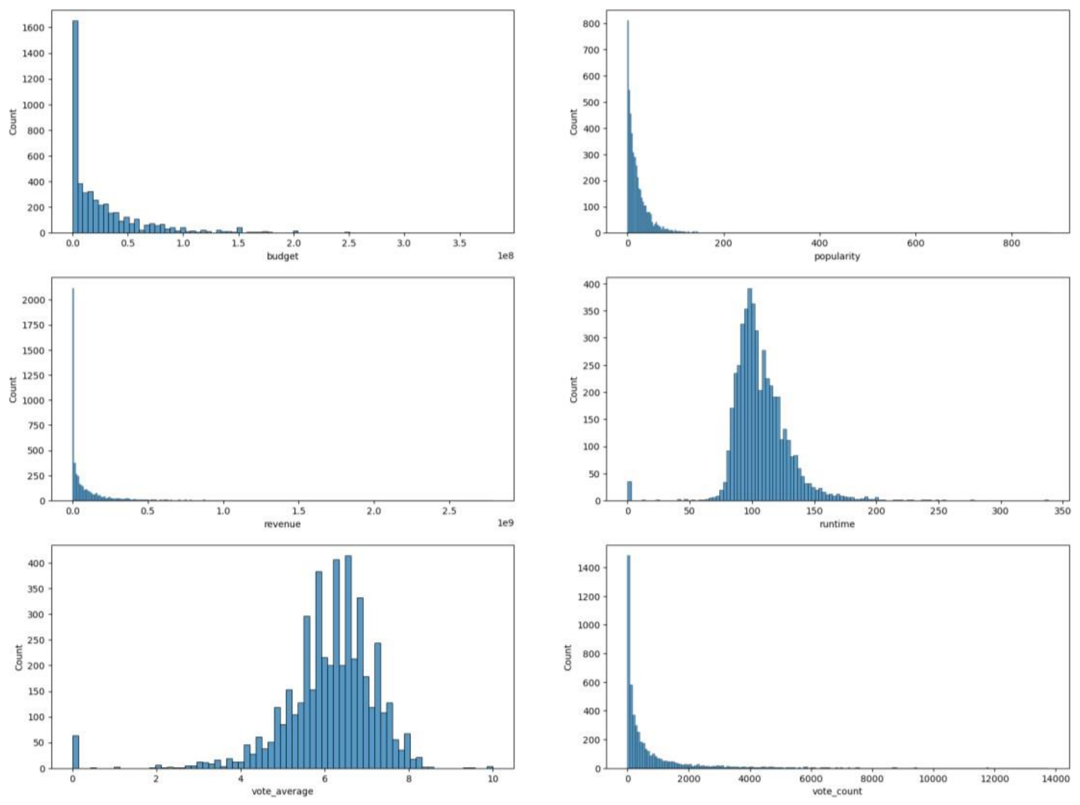
Discrete: vote_count

Continuous: budget, popularity, revenue, runtime, vote_average

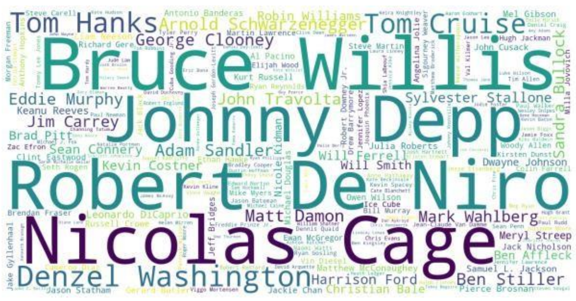
We also made use of describe() function to study the summary statistics of quantitative attributes. Also, since seven of the attributes consisted of json object data, we parsed the json objects in dictionarys using the json library.

Data Visualization

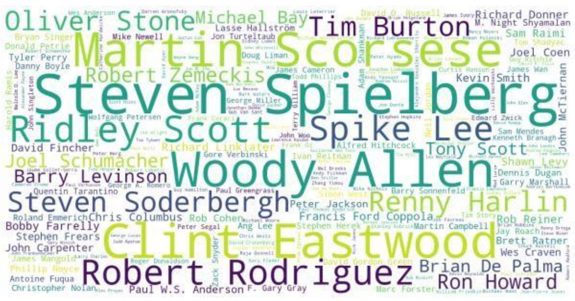
From the univariate analysis of the quantitative attributes using a histogram, it was observed that four of the attributes (budget, revenue, popularity, vote_count) were extremely skewed.



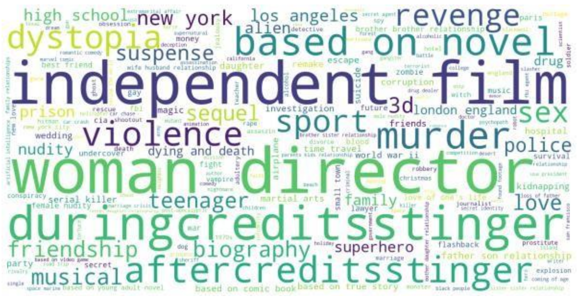
We also found some interesting facts from the word clouds created such as the most used keywords, the actors that play the most lead roles, the most used words in movie overviews and many more.



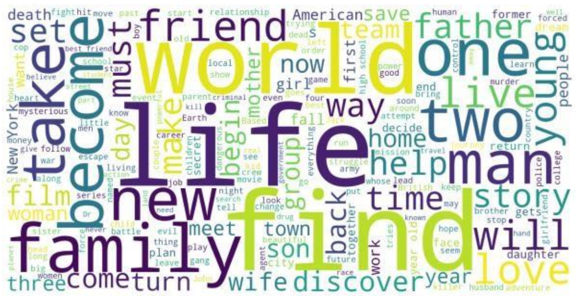
Actors who played lead in most movies



Directors who directed the most movies

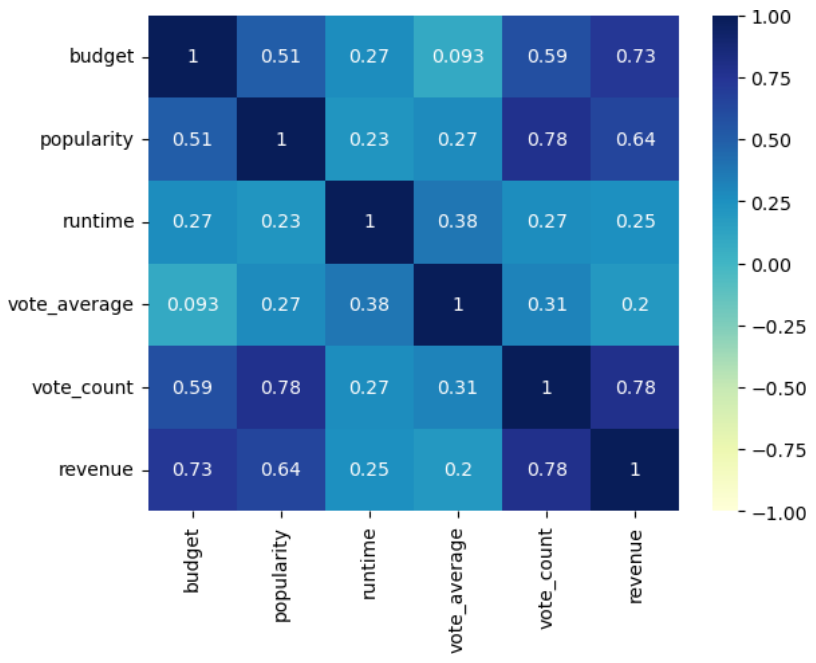


Most used keywords



Most used words in movie overviews

From bivariate analysis, it was found that budget, popularity and vote_count were highly correlated to our target variable revenue. The target attribute also seemed to have significant relationships with genre, homepage, original_language, status and release day of the week, year and month.

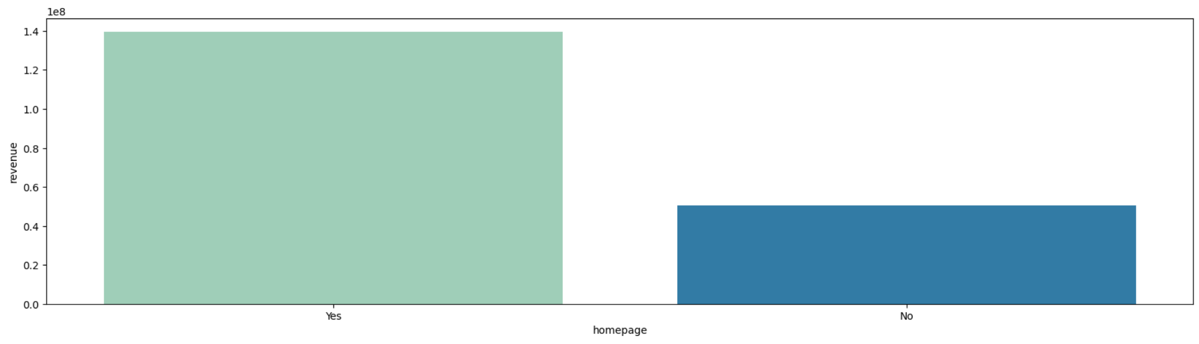


Dropping Unwanted Columns

Using the visualizations interpreted above, along with data type information, we dropped or modified attributes which had no relation to the target attribute. The release day, month and week seemed to have some effect on revenue as seen from the above bar charts. Hence we create three separate variables for the same and drop the attribute 'release_date'. The attributes 'id', 'original_title', 'title_x' and 'title_y' are used for the identification of the distinct movies that are present in the dataset and hence were of no use for modelling and hence were dropped. Textual data like 'overview', 'keywords' and 'tagline' were assumed to not likely be correlated to our target attribute 'revenue' and were dropped. The attribute 'cast' consisted of all the actors who have acted in a movie. Due to the huge variety of actors that have played roles (lead & supporting) in movies, it was assume that there is going to be no significant correlation between this attribute and our target variable (revenue) and hence we dropped it. The 'crew', 'production_companies' and 'production_countries' attributes were also dropped for similar reasoning.

Handling Missing Values

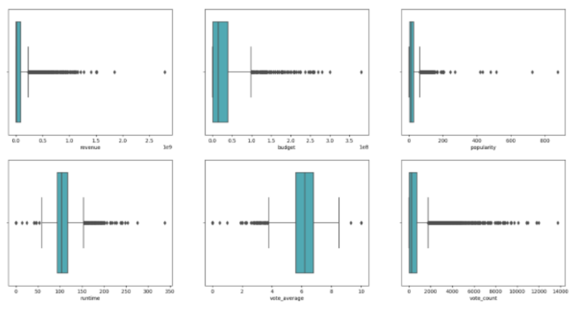
From the bar plot in the bivariate analysis, it was observed that the presence of homepage affects revenue, i.e., movies with homepage have a better revenue as compared to those that don't. Hence Let us just transform the attribute into having two values -> 1 (i.e. has homepage) and 0 (i.e. doesn't have homepage). This was done to remove the 3091 missing homepage values in the dataset.

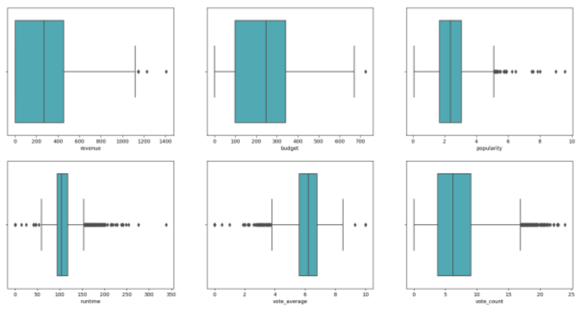


Since only 28 records, i.e. only 0.58% records, had missing genres, we went for deletion of these particular records because we will still had 99.42% of the records left for the modelling. Also since barely 1 record had a missing day, month and year and 2 have a missing runtime, we delete those.

Handling Outliers

From the boxplots, the visualization of the distribution of quantitative variables showed that all the quantitative variables consisted of a large number of outliers, also as mentioned in Section 4.2, budget, revenue, popularity, vote_count had extremely skewed distributions as according to histograms. Hence we made use of linear transformations (specifically cube root) in order to even out the distribution and get it closer to a normal distribution.

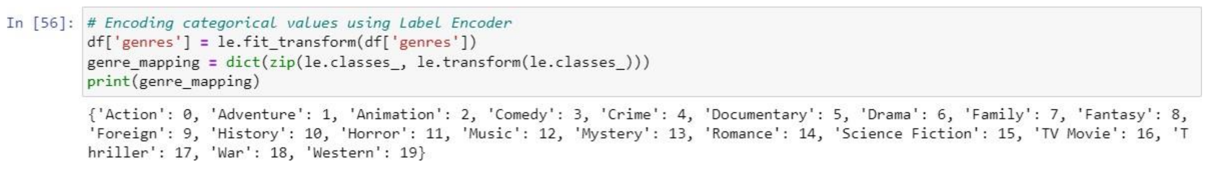




Before cube root transformation After cube root transformation

Encoding Categorical Attributes

After this, the categorical attributes were encoded with the help of label encoder and their mappings were displayed for later use during deployment.



```
In [56]: # Encoding categorical values using Label Encoder
         df['genres'] = le.fit_transform(df['genres'])
         genre_mapping = dict(zip(le.classes_, le.transform(le.classes_)))
         print(genre_mapping)

         {'Action': 0, 'Adventure': 1, 'Animation': 2, 'Comedy': 3, 'Crime': 4, 'Documentary': 5, 'Drama': 6, 'Family': 7, 'Fantasy': 8,
         'Foreign': 9, 'History': 10, 'Horror': 11, 'Music': 12, 'Mystery': 13, 'Romance': 14, 'Science Fiction': 15, 'TV Movie': 16, 'T
         hriller': 17, 'War': 18, 'Western': 19}
```

Splitting the Dataset into Dependent and Independent Variables

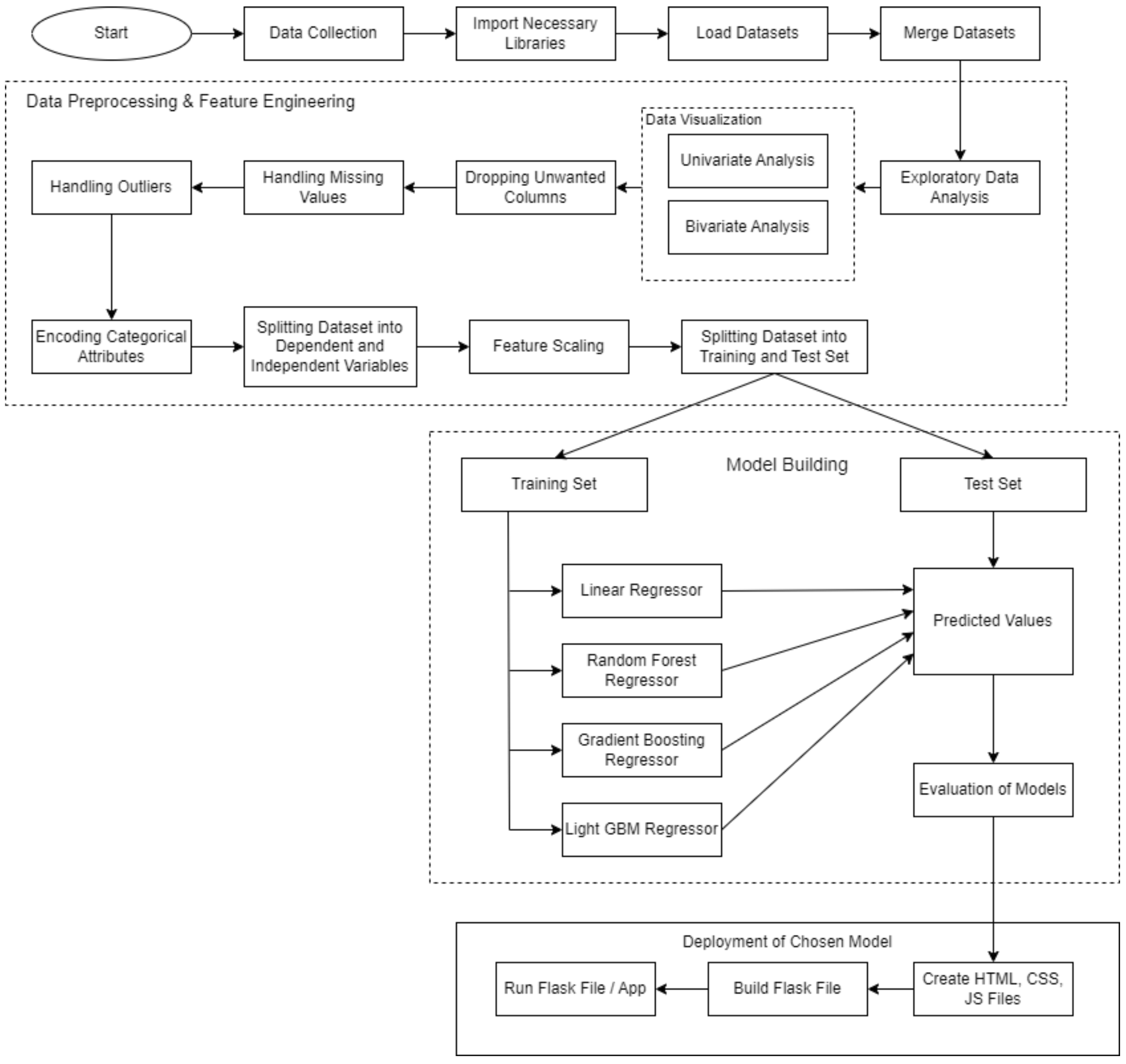
Here we split our dataset in two parts in the test-train split ratio of 75:25. Splitting the dataset into train and test subsets enables us to train the model on one portion of the data, evaluate its performance on unseen data, detect overfitting, and make informed decisions about model selection, hyperparameter tuning, and generalization ability. It helps ensure that the model performs well not only on the training data but also on new, unseen data.

Model Building

We employed several machine learning algorithms: linear regression, random forest, gradient boosting, and Light GBM regressor for model building. For each regressor, first an instance was initialized, then the model was fitted onto the train subset. Following this prediction was made on the test set and model performance was evaluated using the evaluation metrics: Mean Absolute Error (MAE), Mean Squared Error (MSE), R2 Score.

Finally the best performing model was saved for later, i.e. for deployment, using the pickle library.

5. Flowchart


Start
Data Collection
Import Necessary Libraries
Load Datasets
Merge Datasets
Data Preprocessing & Feature Engineering
Data Visualization
Univariate Analysis
Bivariate Analysis
Exploratory Data Analysis
Dropping Unwanted Columns
Handling Missing Values
Handling Outliers
Encoding Categorical Attributes
Splitting Dataset into Dependent and Independent Variables
Feature Scaling
Splitting Dataset into Training and Test Set
Model Building
Training Set
Test Set
Linear Regressor
Random Forest Regressor
Gradient Boosting Regressor
Light GBM Regressor
Predicted Values
Evaluation of Models
Deployment of Chosen Model
Create HTML, CSS, JS Files
Build Flask File
Run Flask File / App

6. Result

We have considered four metrics to estimate the accuracy of regressors – Mean Squared Error, Root Mean Squared Error, Mean Absolute Error and R2 score.

Mean Squared Error:

Mean Squared Error (MSE) measures the average squared difference between the predicted values and the actual values. MSE provides a quantitative measure of how well the model fits the data, with lower values indicating better performance.

The formula to calculate MSE is as follows:

$$MSE = (1/n) * \Sigma(y - \hat{y})^2$$

Root Mean Squared Error:

Root Mean Squared Error (RMSE) derived from the Mean Squared Error (MSE) and provides a more interpretable measure of the average difference between the predicted values and the actual values.

The RMSE is calculated by taking the square root of the MSE: RMSE = sqrt(MSE)

Mean Absolute Error:

Mean Absolute Error (MAE) measures the average absolute difference between the predicted values and the actual values. MAE provides a straightforward measure of the average prediction error without considering the direction of the errors.

The formula to calculate MAE is as follows:

$$MAE = (1/n) * \Sigma|y - \hat{y}|$$

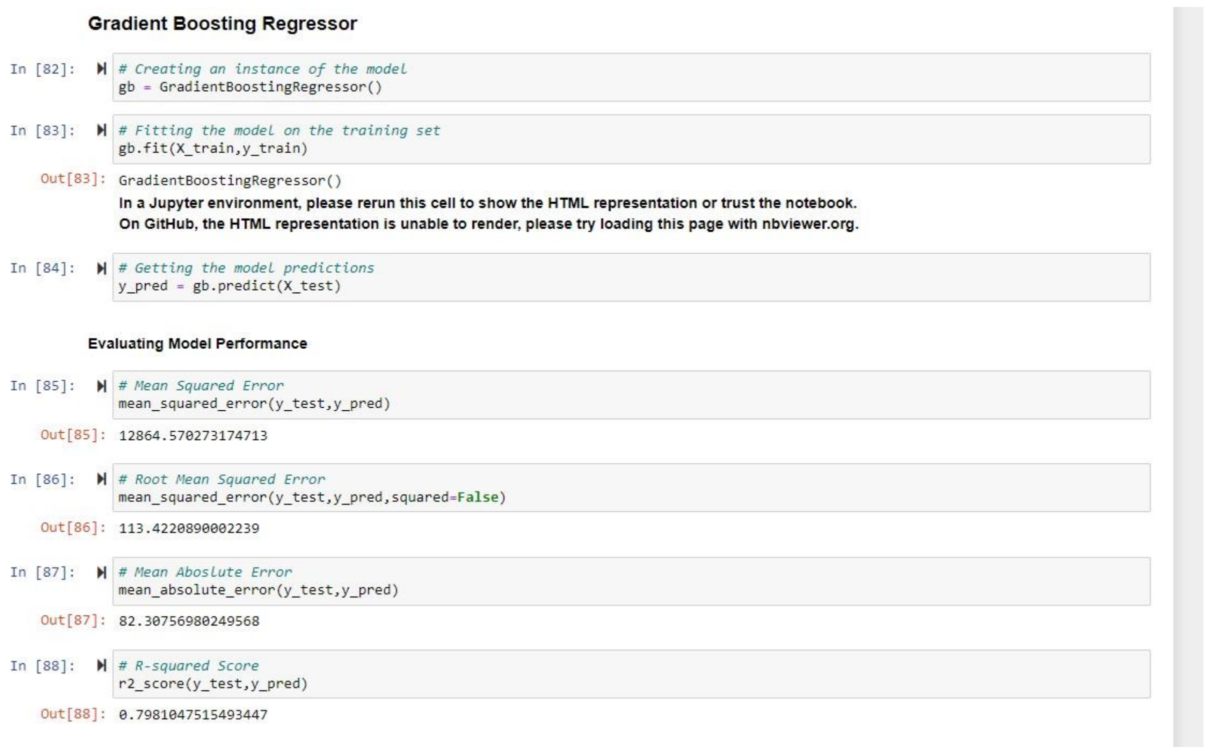
R^2 Score:

R-squared (R^2) score, is an evaluation metric commonly used for regression models. R^2 measures the proportion of the variance in the dependent variable that is explained by the independent variables in the model. It quantifies the goodness of fit of the model.

The formula to calculate R^2 is as follows:

$$R\text{^}2 = 1 - (SS\_res / SS\_tot)$$

Regressors	**MSE**	**RMSE**	**MAE**	**R2**
Linear Regressor	14349.028	119.787	89.563	0.775
Random Forest Regressor	13351.431	115.548	83.179	0.79
Gradient Boost Regressor	12859.039	113.397	82.268	0.798
Light GBM Regressor	13165.948	114.743	82.705	0.793



Gradient Boosting Regressor

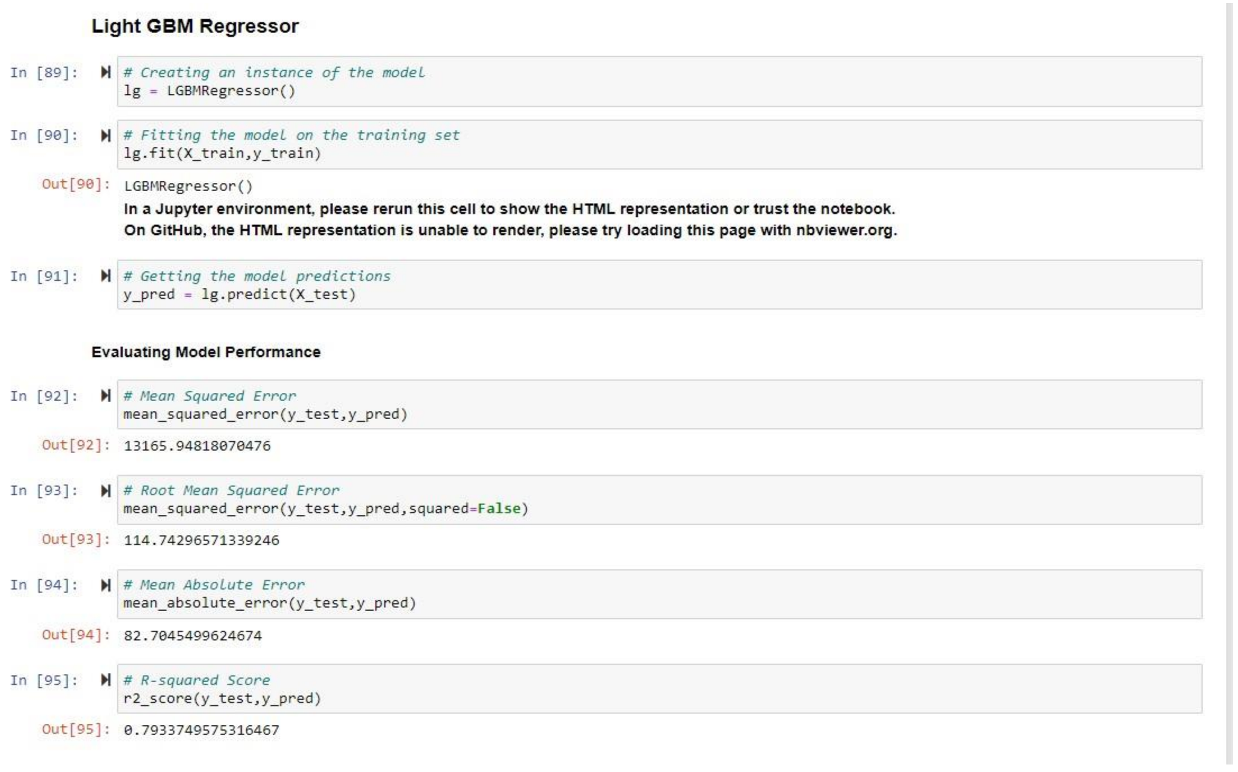
```
In [82]:  # Creating an instance of the model
          gb = GradientBoostingRegressor()

In [83]:  # Fitting the model on the training set
          gb.fit(X_train,y_train)

Out[83]:  GradientBoostingRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [84]:  # Getting the model predictions
          y_pred = gb.predict(X_test)
```

Evaluating Model Performance

```
In [85]:  # Mean Squared Error
          mean_squared_error(y_test,y_pred)

Out[85]:  12864.570273174713

In [86]:  # Root Mean Squared Error
          mean_squared_error(y_test,y_pred,squared=False)

Out[86]:  113.4220890002239

In [87]:  # Mean Abosolute Error
          mean_absolute_error(y_test,y_pred)

Out[87]:  82.30756980249568

In [88]:  # R-squared Score
          r2_score(y_test,y_pred)

Out[88]:  0.7981047515493447
```



Light GBM Regressor

```
In [89]:  # Creating an instance of the model
          lg = LGBMRegressor()

In [90]:  # Fitting the model on the training set
          lg.fit(X_train,y_train)

Out[90]:  LGBMRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [91]:  # Getting the model predictions
          y_pred = lg.predict(X_test)
```

Evaluating Model Performance

```
In [92]:  # Mean Squared Error
          mean_squared_error(y_test,y_pred)

Out[92]:  13165.94818070476

In [93]:  # Root Mean Squared Error
          mean_squared_error(y_test,y_pred,squared=False)

Out[93]:  114.74296571339246

In [94]:  # Mean Absolute Error
          mean_absolute_error(y_test,y_pred)

Out[94]:  82.7045499624674

In [95]:  # R-squared Score
          r2_score(y_test,y_pred)

Out[95]:  0.7933749575316467
```

Random Forest Regressor

```
In [75]:  # Creating an instance of the model
          rf = RandomForestRegressor()
```

```
In [76]:  # Fitting the model on the training set
          rf.fit(X_train,y_train)

Out[76]:  RandomForestRegressor()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [77]:  # Getting the model predictions
          y_pred = rf.predict(X_test)
```

Evaluating Model Performance

```
In [78]:  # Mean Squared Error
          mean_squared_error(y_test,y_pred)

Out[78]:  13323.691693709132
```

```
In [79]:  # Root Mean Squared Error
          mean_squared_error(y_test,y_pred,squared=False)

Out[79]:  115.42829676344155
```

```
In [80]:  # Mean Aboslute Error
          mean_absolute_error(y_test,y_pred)

Out[80]:  83.09463849479197
```

```
In [81]:  # R-squared Score
          r2_score(y_test,y_pred)

Out[81]:  0.7908993469925285
```

Linear Regressor

```
In [68]:  # Creating an instance of the model
          lr = LinearRegression()
```

```
In [69]:  # Fitting the model on the training set
          lr.fit(X_train,y_train)

Out[69]:  LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [70]:  # Getting the model predictions
          y_pred = lr.predict(X_test)
```

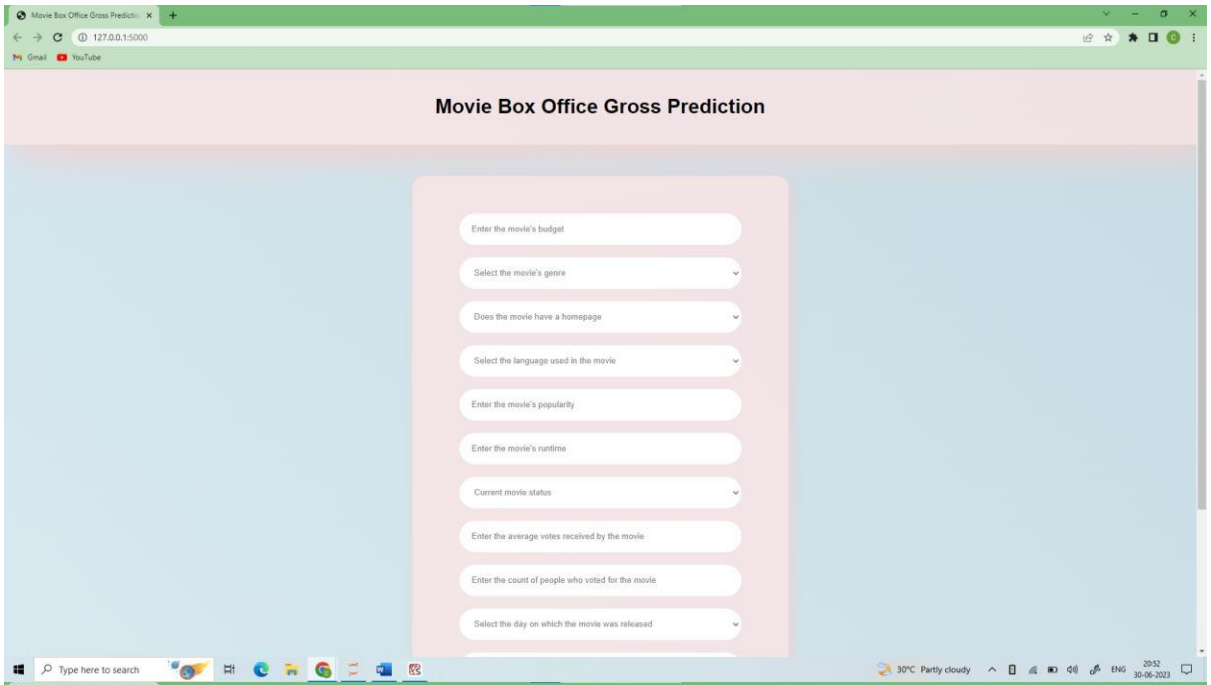
Evaluating Model Performance

```
In [71]:  # Mean Squared Error
          mean_squared_error(y_test,y_pred)

Out[71]:  14349.028200701816
```

```
In [72]:  # Root Mean Squared Error
          mean_squared_error(y_test,y_pred,squared=False)

Out[72]:  119.78742922653369
```

```
In [73]:  # Mean Aboslute Error
          mean_absolute_error(y_test,y_pred)

Out[73]:  89.56338652485672
```

```
In [74]:  # R-squared Score
          r2_score(y_test,y_pred)

Out[74]:  0.7748078208529827
```


Movie Box Office Gross Prediction

Enter the movie's budget

Select the movie's genre

Does the movie have a homepage

Select the language used in the movie

Enter the movie's popularity

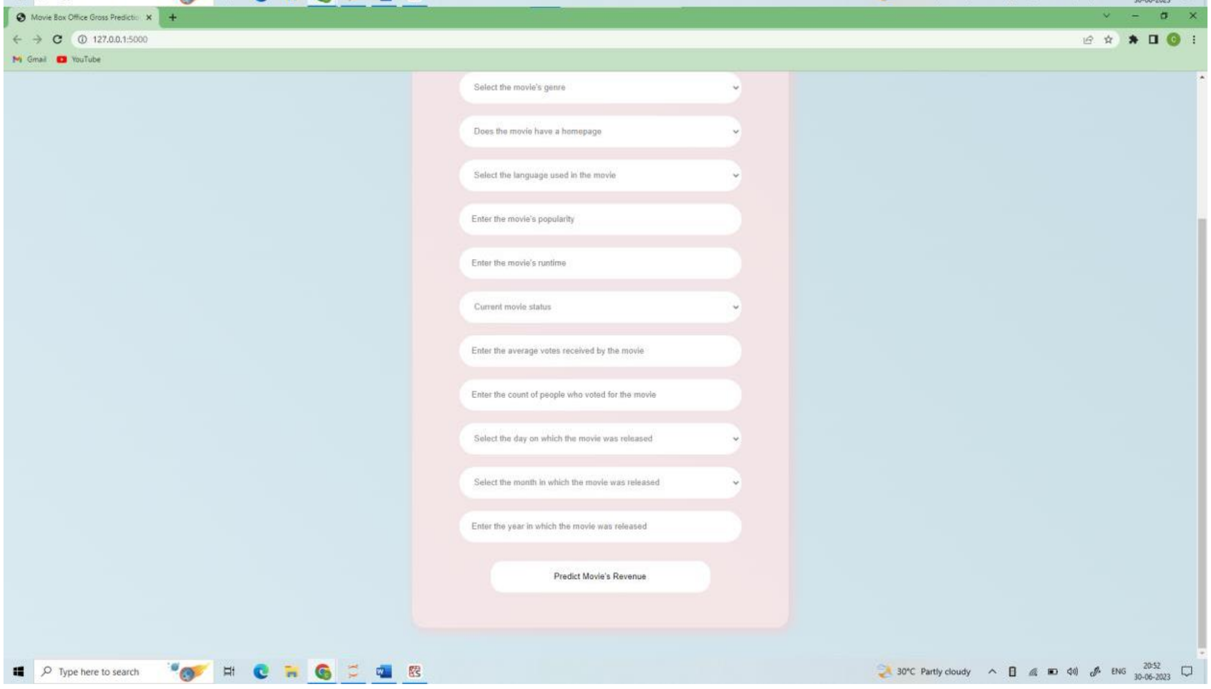
Enter the movie's runtime

Current movie status

Enter the average votes received by the movie

Enter the count of people who voted for the movie

Select the day on which the movie was released


Select the movie's genre

Does the movie have a homepage

Select the language used in the movie

Enter the movie's popularity

Enter the movie's runtime

Current movie status

Enter the average votes received by the movie

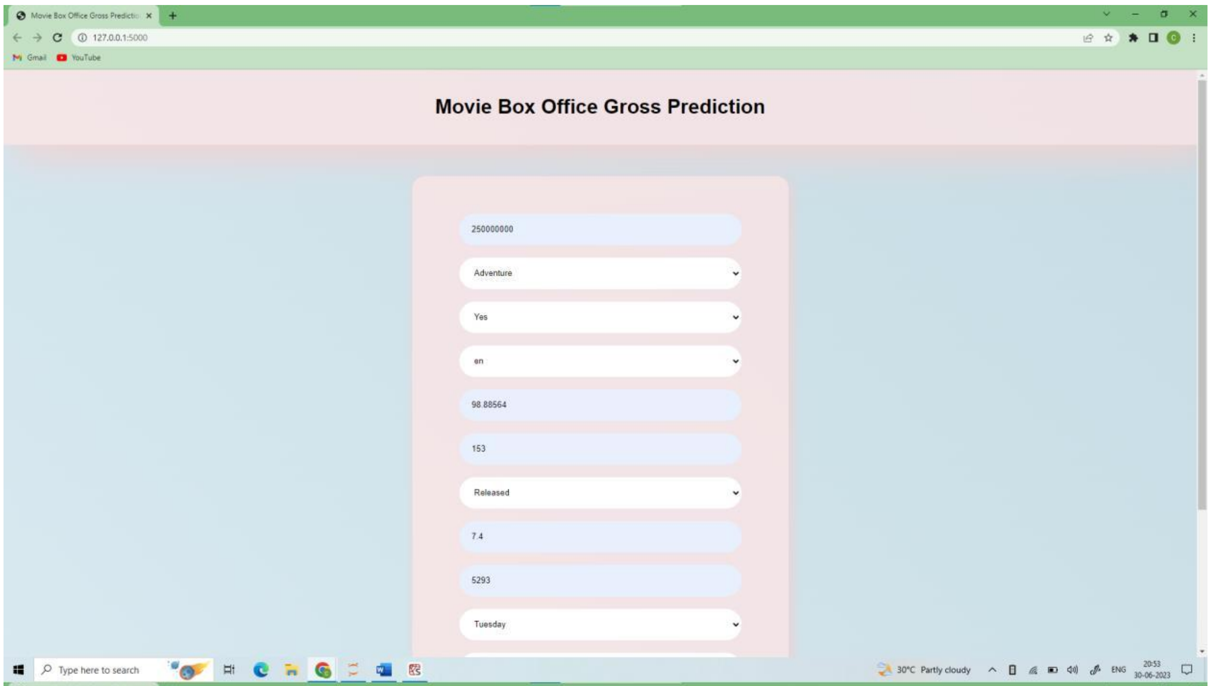
Enter the count of people who voted for the movie

Select the day on which the movie was released

Select the month in which the movie was released

Enter the year in which the movie was released

Predict Movie's Revenue



Movie Box Office Gross Prediction

250000000

Adventure

Yes

en

98.88564

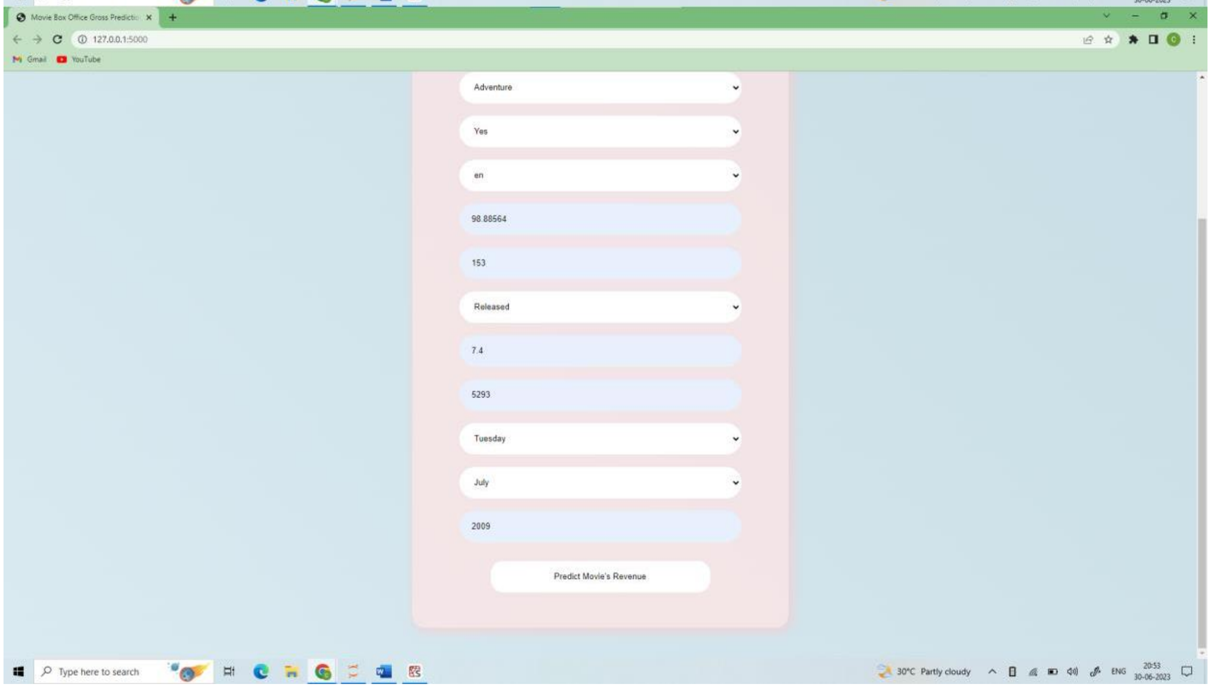
153

Released

7.4

5293

Tuesday



Adventure

Yes

en

98.88564

153

Released

7.4

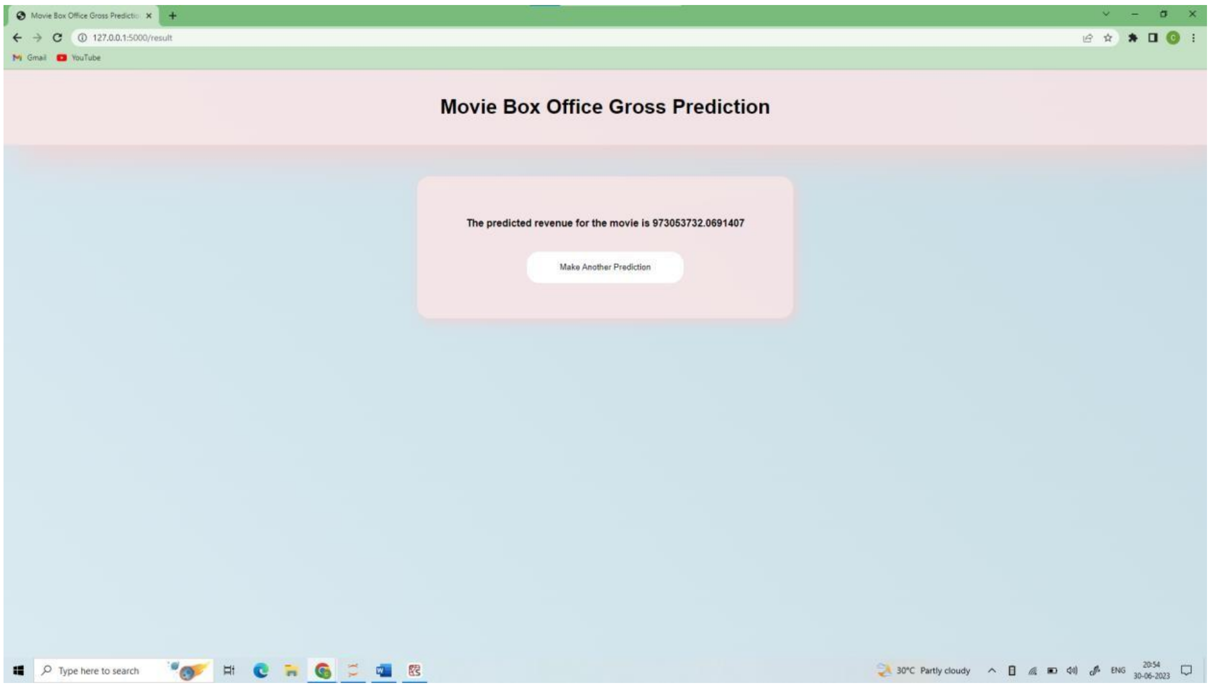
5293

Tuesday

July

2009

Predict Movie's Revenue


Movie Box Office Gross Prediction
127.0.0.1:5000/result
Movie Box Office Gross Prediction
Make Another Prediction

The predicted revenue for the movie is 973053732.0691407

7. Advantages & Limitations

Advantages of utilizing the model:

Financial Planning: Accurate box office gross forecasts aid distributors, financiers, and film studios in determining a film's prospective income and profitability. They are able to decide on distribution, marketing, and budgeting plans with knowledge.

Marketing Strategy: Box office forecasts aid in the creation of efficient marketing initiatives. Studios may adapt their promotional campaigns and distribute resources effectively by having a thorough understanding of the target audience and their preferences. It aids in expanding the film's audience and drawing in target audiences.

Industry Competition: Accurate box office forecasts enable studios to assess how well their films are doing in comparison to competitors. It aids in recognizing possible areas for improvement, recognizing market trends, and evaluating the success of comparable films. This information can help create a more vibrant and competitive industry.

Audience Engagement: Predictions can create excitement and apprehension among moviegoers. Accurate forecasts frequently garner media attention, which raises public awareness and interest in the film. Increased ticket sales may result from the enthusiasm it generates and the audience engagement it fosters.

Limitations of the model:

Uncertainty: Box office gross forecasting is unreliable to some extent and is dependent on a number of variables that can impact a film's performance. A movie's box office performance can be strongly impacted by factors like competition from other films, critical acclaim, marketing initiatives, and even unanticipated events. Sometimes predictions don't take these aspects into consideration correctly.

Complex Market Dynamics: The movie industry is impacted by a number of complicated market aspects, such as shifting consumer tastes, trends, and cultural considerations. It can be difficult and error-prone to analyze and comprehend these dynamics, which are necessary for predicting box office results.

Unforeseen External circumstances: Unforeseen external circumstances, such as economic downturns, natural disasters, or world events, might have an impact on predictions. These elements may alter audience behavior, influence consumer spending, and provide erratic results at the box office.

8. Applications

Investment and Financing Decisions: Predictions about the box office take for a film are important in making these decisions. These forecasts are used by production companies, studios, and investors to evaluate a project's financial sustainability and decide whether to support it. The prospective return on investment and discussions for licencing and distribution rights can both be guided by predictions.

Distribution Planning: Box office gross projections aid in distribution planning by assisting studios and distributors in determining how widely and in which regions to release a film. Predictions can show how popular a movie will be in various markets, enabling smart release plans and maximizing distribution efforts. By doing this, the movie's potential audience and earnings are maximized.

Evaluation of Performance: After a film is released, its performance is measured against projections of its box office take. Stakeholders can evaluate a film's success and spot any anomalies by comparing the expected gross with the actual box office numbers. This data makes it easier to evaluate a movie's marketing efficiency, audience response, and other performance-related criteria.

Industry Insights and Trends: Box office projections are a useful source of information about market trends and audience preferences when analyzed as a whole. Predictions can be monitored and analyzed over time to spot trends, new genres, or modifications in audience behavior. Professionals in the industry can use this information to make well-informed decisions on production, genre, and market placement.

9. Conclusion

In this study, we compared the performance of four regression models: Linear Regression, Random Forest Regression, Gradient Boost Regression and Light GBM Regression. After evaluating the models on various metrics, we found that Gradient Boost Regressor is the best performing model. It has the lowest mean squared, root mean squared and mean absolute error and the highest r2 score. The next best performing regressors are Light GBM, Random Forest and Linear Regressors in order.

10. Future Scope

Further investigation can be done to explore other regression models or apply advanced techniques such as regularization or ensemble methods to improve prediction accuracy. Also, future work could involve more thorough hyperparameter tuning, external validation on independent, larger and more diverse datasets. Future research direction would be to minimize the limitations.

11. Bibliography

[1] R. Chen, W. Xu, and X. Zhang, “Dynamic Box Office Forecasting Based on Microblog Data,” Filomat, vol. 30, no. 15, pp. 4111–4124, 2016.

[2] T. Kim, J. Hong, and P. Kang, “Box Office Forecasting considering Competitive Environment and Word-of-Mouth in Social Networks: A Case Study of Korean Film Market,” Computational Intelligence and Neuroscience, vol. 2017, p. 4315419, Jul. 2017, doi: 10.1155/2017/4315419.

[3] N. Quader, Md. O. Gani, D. Chaki, and Md. H. Ali, “A machine learning approach to predict movie box-office success,” in 2017 20th International Conference of Computer and Information Technology (ICCIT), 2017, pp. 1–7. doi: 10.1109/ICCITECHN.2017.8281839.

[4] Y.-H. Hu, W.-M. Shiau, S.-P. Shih, and C.-J. Chen, “Considering online consumer reviews to predict movie box-office performance between the years 2009 and 2014 in the US,” The Electronic Library, vol. 36, no. 6, pp. 1010–1026, Jan. 2018, doi: 10.1108/EL-02-2018-0040.

12. Appendix

Python code

```
# Importing the necessary packages and libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import json
import pickle
from scipy import stats
from wordcloud import WordCloud, STOPWORDS
from collections import Counter
from sklearn.preprocessing import LabelEncoder,RobustScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
# Loading the datasets
movies = pd.read_csv("tmdb_5000_movies.csv")
credits = pd.read_csv("tmdb_5000_credits.csv")
# Displaying the first 5 records of the movies dataset
movies.head()
# Displaying the shape of the movies dataset i.e. (no. of records, no. of attributes)
movies.shape
# Displaying the first 5 records of the credits dataset
credits.head()
# Displaying the shape of the credits dataset i.e. (no. of records, no. of attributes)
credits.shape
credits.rename(columns = {'movie_id':'id'}, inplace = True)
# Checking whether the above process has been done successfully
credits.columns
# Merging the two datasets based on the key element 'id'
df = movies.merge(credits, on = 'id')
# Displaying the first 5 records of the newly merged dataset
df.head()
# Displaying the shape of the dataset i.e. (no. of records, no. of attributes)
df.shape
# Displaying information about the dataset attributes
df.info()
# Changing the data type of the attribute 'release_date'
df['release_date'] = pd.to_datetime(df['release_date']).apply(lambda x: x.date())
# Displaying summary statistics for the numerical attributes
df.describe()
# Parsing json objects into dictionaries
json_columns = ['genres', 'keywords', 'production_countries', 'production_companies', 'spoken_languages', 'cast', 'crew']
for column in json_columns:
    df[column] = df[column].apply(json.loads)
# Visualizing the distribution of quantitative attributes using histograms
fig,ax = plt.subplots(3,2,figsize=(20,15))
fig.suptitle('Histograms for Continuous Attribute Analysis', fontsize = 20)

sns.histplot(ax = ax[0,0], x = 'budget', data = df)
sns.histplot(ax = ax[0,1], x = 'popularity', data = df)
sns.histplot(ax = ax[1,0], x = 'revenue', data = df)
sns.histplot(ax = ax[1,1], x = 'runtime', data = df)
sns.histplot(ax = ax[2,0], x = 'vote_average', data = df)
sns.histplot(ax = ax[2,1], x = 'vote_count', data = df); # semicolon added to remove unecessary output text
# Visualizing the counts of the genres of the movies
primary_genres = df['genres'].apply(lambda x: x[0]['name'] if len(x)!=0 else pd.NA)
primary_genres = primary_genres.dropna()
primary_genres = primary_genres.tolist()
primary_genres.sort()

all_genres = []
glst = df['genres'].apply(lambda x: [i['name'] for i in x])
for genre in glst:
    all_genres.extend(genre)
all_genres.sort()
fig,ax = plt.subplots(2,1,figsize=(20,10))
sns.countplot(ax = ax[0], x = primary_genres, palette = 'YlGnBu')
ax[0].bar_label(ax[0].containers[0])
ax[0].set_title('Count of movie genres where only first/primary genre is considered')
sns.countplot(ax = ax[1], x = all_genres, palette = 'YlGnBu')
ax[1].bar_label(ax[1].containers[0])
ax[1].set_title('Count of movie genres where all genres are considered');
# Since there's not much difference, let us just consider the first genre for each movie
df['genres'] = df['genres'].apply(lambda x: x[0]['name'] if len(x)!=0 else pd.NA)
# Getting the counts of the original language of movies
plt.figure(figsize = (20, 6))
ax = sns.countplot(x = df['original_language'], palette = 'YlGnBu')
ax.bar_label(ax.containers[0])
ax.set_title('Count of original language of movies');
# Getting the count of movies having different statuses
plt.figure(figsize = (20, 6))
ax = sns.countplot(x = df['status'], palette = 'YlGnBu')
ax.bar_label(ax.containers[0]);
# Getting insight on which actors played the role of main characters the most in movies
cast = df['cast'].apply(lambda x: x[0]['name'] if len(x)!=0 else pd.NA)
cast = cast.dropna()
d = Counter(cast)
wordcloud = WordCloud(width=1600, height=800, background_color ='white', collocations=False).generate_from_frequencies(d)
plt.figure(figsize = (20, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off");
# Getting insight on actors who have appeared the most in movies
all_cast = []
clst = df['cast'].apply(lambda x: [i['name'] for i in x])
for genre in clst:
    all_cast.extend(genre)
all_cast.sort()
d = Counter(all_cast)
wordcloud = WordCloud(width=1600, height=800, background_color ='white', collocations=False).generate_from_frequencies(d)
plt.figure(figsize = (20, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off");
```

```
def find_director(c):
    for m in c:
        if m['job'] == 'Director':
            return m['name']
# Getting insight on the directors who directed the most
amount of movies
directors = df['crew'].apply(lambda x: find_director(x))
directors = directors.dropna()
d = Counter(directors)
wordcloud = WordCloud(width=1600, height=800,
background_color ='white',
collocations=False).generate_from_frequencies(d)
plt.figure(figsize = (20, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off");
# Getting insight on the most used keywords across movies
using word cloud
keywords = []
klst = df['keywords'].apply(lambda x: [i['name'] for i in x])
for word in klst:
    keywords.extend(word)
d = Counter(keywords)
wordcloud = WordCloud(width=1600, height=800,
background_color ='white',
collocations=False).generate_from_frequencies(d)
plt.figure(figsize = (20, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off");
# Getting insight on the most used words used to describe
movies in overview
overviews = df['overview'].dropna()
words = ''
stopwords = set(STOPWORDS)
for val in overviews:
    val.lower()
    tokens = val.split()
    words += " ".join(tokens)+" "
wordcloud = WordCloud(width=1600, height=800,
background_color ='white', stopwords =
stopwords).generate(words)
plt.figure(figsize = (20, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off");
# Getting insight on the most used words in movie taglines
using word cloud
taglines = df['tagline'].dropna()
words = ''
stopwords = set(STOPWORDS)
for val in taglines:
    val.lower()
    tokens = val.split()
    words += " ".join(tokens)+" "
wordcloud = WordCloud(width=1600, height=800,
background_color ='white', stopwords =
stopwords).generate(words)
plt.figure(figsize = (20, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off");
# Getting insight on the companies which produced the most
movies
prodcoms = []
pclst = df['production_companies'].apply(lambda x:
[i['name'] for i in x])
for comp in pclst:
    prodcoms.extend(comp)
d = Counter(prodcoms)
wordcloud = WordCloud(width=1600, height=800,
background_color ='white',
collocations=False).generate_from_frequencies(d)
plt.figure(figsize = (20, 10), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off");
# Let us find the relationship between our target variable
'Revenue' and other possible quantitative predictor variables
using scatter plot
fig,ax = plt.subplots(3,2,figsize=(20,15))
fig.suptitle('Scatterplots for finding relationship between
Quantitative Attributes and Revenue', fontsize = 20)
sns.scatterplot(ax = ax[0,0], x = 'budget', y = 'revenue', data =
df)
sns.scatterplot(ax = ax[0,1], x = 'popularity', y = 'revenue',
data = df)
sns.scatterplot(ax = ax[1,0], x = 'runtime', y = 'revenue', data
= df)
sns.scatterplot(ax = ax[1,1], x = 'vote_average', y = 'revenue',
data = df)
sns.scatterplot(ax = ax[2,0], x = 'vote_count', y = 'revenue',
data = df)
fig.delaxes(ax[2,1])
# Finding the correlation between the quantitative attributes
present in the dataset using heatmap
sns.heatmap(df[['budget','popularity','runtime','vote_average'
,'vote_count','revenue']].corr(), annot = True, cmap =
'YlGnBu', vmin = -1, vmax = 1);
# Finding whether genre of the movie affects the average
revenue
plt.figure(figsize=(20,5))
sns.barplot(x = 'genres', y = 'revenue', data = df, palette =
'YlGnBu', errorbar = None);
# Finding whether presence of homepage affects average
movie revenue
has_home = df['homepage'].apply(lambda x: 'Yes' if
pd.notna(x) else 'No')
plt.figure(figsize=(20,5))
sns.barplot(x = has_home, y = df['revenue'], palette =
'YlGnBu', errorbar = None);
# Finding whether the original language of the movie affects
the revenue
plt.figure(figsize=(20,5))
sns.barplot(x = 'original_language', y = 'revenue', data = df,
palette = 'YlGnBu', errorbar = None);
# Finding whether status affects revenue of movies
plt.figure(figsize=(20,5))
sns.barplot(x = 'status', y = 'revenue', data = df, palette =
'YlGnBu', errorbar = None);
# Finding whether the release day of the week affects revenue
day = df['release_date'].dropna().apply(lambda x:
x.strftime('%A'))
plt.figure(figsize=(20,5))
sns.barplot(x = day, y = df['revenue'], palette = 'YlGnBu',
errorbar = None);
# Finding whether the release day of the month affects
revenue
day = df['release_date'].apply(lambda x: x.day)
day = day.sort_values()
plt.figure(figsize=(20,5))
sns.barplot(x = day, y = df['revenue'], palette = 'YlGnBu',
errorbar = None);
# Finding whether the release month affects revenue
month = df['release_date'].apply(lambda x: x.month)
month = month.sort_values()
plt.figure(figsize=(20,5))
sns.barplot(x = month, y = df['revenue'], palette = 'YlGnBu',
errorbar = None);
# Finding whether release year affects revenue
years = df['release_date'].apply(lambda x: x.year)
years = years.sort_values()
plt.figure(figsize=(20,5))
sns.barplot(x = years, y = df['revenue'], palette = 'YlGnBu',
errorbar = None)
plt.xticks(rotation = 90);
# Creating the above-mentioned attributes
```

```
df['day'] = df['release_date'].apply(lambda x: x.weekday()) #
0 -> Monday, 6 -> Sunday
df['month'] = df['release_date'].apply(lambda x: x.month)
df['year'] = df['release_date'].apply(lambda x: x.year)
# Dropping the above-mentioned column
df = df.drop(['release_date'], axis = 1)
# Dropping the above-mentioned columns
df = df.drop(['id', 'original_title', 'title_x','title_y'], axis = 1)
# Dropping the above-mentioned columns
df = df.drop(['overview', 'keywords', 'tagline'], axis = 1)
# Dropping the above-mentioned column
df = df.drop(['spoken_languages'], axis = 1)
# Dropping the above-mentioned column
df =
df.drop(['cast','crew','production_companies','production_co
untries'], axis = 1)
# Displaying the first 5 records of the data frame after
dropping unwanted columns
df.head()
# Checking whether the dataset has any missing values by
visualizing the missing values present in the dataset
sns.heatmap(df.isna(), yticklabels = False, cbar = False, cmap
= "YlGnBu_r");
# Count of the missing values present in the dataset
df.isnull().sum()
# From the bar plot in the bivariate analysis, we can see that
the presence of homepage affects revenue, i.e.,
#movies with homepage have a better revenue as compared
to those that don't. Hence Let us just transform the attribute
#into having two values -> 1 (i.e. has homepage) and 0 (i.e.
doesn't have homepage)
df['homepage'] = df['homepage'].apply(lambda x: 1 if
pd.notna(x) else 0)
# Since only 28 records, i.e. only 0.58% records, have missing
genres, let us just go for deletion of these particular records
# because we will still have 99.42% of the records left for the
modelling. Also since barely 1 record has a missing
day,month and year
# and 2 have a missing runtime, let us also delete those.
df.dropna(subset = ['genres','day','month','year','runtime'],
inplace = True)
# Checking whether all missing values have been removed
from the dataset
df.isnull().sum()
# Data frame after handling missing values
df.head()
# Using Boxplots to visualize outliers
fig,ax = plt.subplots(2,3,figsize=(20, 10))
fig.suptitle('Box Plots for Quantitative Attributes', fontsize =
20)

sns.boxplot(ax = ax[0,0], x = 'revenue', data = df, palette =
'YlGnBu')
sns.boxplot(ax = ax[0,1], x = 'budget', data = df, palette =
'YlGnBu')
sns.boxplot(ax = ax[0,2], x = 'popularity', data = df, palette =
'YlGnBu')

sns.boxplot(ax = ax[1,0], x = 'runtime', data = df, palette =
'YlGnBu')
sns.boxplot(ax = ax[1,1], x = 'vote_average', data = df, palette
= 'YlGnBu')
sns.boxplot(ax = ax[1,2], x = 'vote_count', data = df, palette =
'YlGnBu');
# Performing cube root transformation on 'revenue', 'budget',
'popularity' and 'vote_count'
df['revenue'] = np.cbrt(df['revenue'])
df['budget'] = np.cbrt(df['budget'])
df['popularity'] = np.cbrt(df['popularity'])
df['vote_count'] = np.cbrt(df['vote_count'])
# Using Boxplots to see status of outliers after
transformations
fig,ax = plt.subplots(2,3,figsize=(20, 10))

sns.boxplot(ax = ax[0,0], x = 'revenue', data = df, palette =
'YlGnBu')
sns.boxplot(ax = ax[0,1], x = 'budget', data = df, palette =
'YlGnBu')
sns.boxplot(ax = ax[0,2], x = 'popularity', data = df, palette =
'YlGnBu')

sns.boxplot(ax = ax[1,0], x = 'runtime', data = df, palette =
'YlGnBu')
sns.boxplot(ax = ax[1,1], x = 'vote_average', data = df, palette
= 'YlGnBu')
sns.boxplot(ax = ax[1,2], x = 'vote_count', data = df, palette =
'YlGnBu');
# Creating an instance of Label Encoder
le = LabelEncoder()
# Encoding categorical values using Label Encoder
df['genres'] = le.fit_transform(df['genres'])
genre_mapping = dict(zip(le.classes_,
le.transform(le.classes_)))
print(genre_mapping)
df['original_language'] =
le.fit_transform(df['original_language'])
lang_mapping = dict(zip(le.classes_,
le.transform(le.classes_)))
print(lang_mapping)
df['status'] = le.fit_transform(df['status'])
status_mapping = dict(zip(le.classes_,
le.transform(le.classes_)))
print(status_mapping)
# y -> dependent variable(target), X -> independent
variables(predictors)
y = df['revenue']
X = df.drop(['revenue'], axis = 1)
# Displaying the first five instances of the dependent variable
y.head()
# Displaying the first five records of the independent
variables
X.head()
# Creating an instance of Robust Scaler
scale = RobustScaler()
# Scaling the independent variables according to the inter-
quartile range
X_scaled = scale.fit_transform(X)
# Replacing the orginal values with the scaled values
X = pd.DataFrame(X_scaled, columns = X.columns)
# Checking whether the values have been successfully scaled
X.head()
# Saving the scaler object for later use during deployment
pickle.dump(scale,open("scaler.pkl","wb"))
# Splitting the dataset in a 75:25 train-test split ratio (i.e. 75%
records will belong to training set and rest 25% to the testing
set)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size = 0.25, random_state = 42)
# Creating an instance of the model
lr = LinearRegression()
# Fitting the model on the training set
lr.fit(X_train,y_train)
# Getting the model predictions
y_pred = lr.predict(X_test)
# Mean Squared Error
mean_squared_error(y_test,y_pred)
# Root Mean Squared Error
mean_squared_error(y_test,y_pred,squared=False)
# Mean Aboslute Error
mean_absolute_error(y_test,y_pred)
# R-squared Score
```

```
r2_score(y_test,y_pred)
# Creating an instance of the model
rf = RandomForestRegressor()
# Fitting the model on the training set
rf.fit(X_train,y_train)
# Getting the model predictions
y_pred = rf.predict(X_test)
# Mean Squared Error
mean_squared_error(y_test,y_pred)
# Root Mean Squared Error
mean_squared_error(y_test,y_pred,squared=False)
# Mean Aboslute Error
mean_absolute_error(y_test,y_pred)
# R-squared Score
r2_score(y_test,y_pred)
# Creating an instance of the model
gb = GradientBoostingRegressor()
# Fitting the model on the training set
gb.fit(X_train,y_train)
# Getting the model predictions
y_pred = gb.predict(X_test)
# Mean Squared Error
mean_squared_error(y_test,y_pred)
# Root Mean Squared Error
mean_squared_error(y_test,y_pred,squared=False)
# Mean Aboslute Error
mean_absolute_error(y_test,y_pred)
# R-squared Score
r2_score(y_test,y_pred)
# Creating an instance of the model
lg = LGBMRegressor()
# Fitting the model on the training set
lg.fit(X_train,y_train)
# Getting the model predictions
y_pred = lg.predict(X_test)
# Mean Squared Error
mean_squared_error(y_test,y_pred)
# Root Mean Squared Error
mean_squared_error(y_test,y_pred,squared=False)
# Mean Absolute Error
mean_absolute_error(y_test,y_pred)
# R-squared Score
r2_score(y_test,y_pred)
# Saving the model object for later use during deployment
pickle.dump(gb,open("model.pkl","wb"))
```

Flask code

```
from flask import Flask, render_template, request
import pickle
import numpy as np
app = Flask(_name_)
scale = pickle.load(open(r'scaler.pkl', 'rb'))
model = pickle.load(open(r'model.pkl', 'rb'))
@app.route("/")
def initial():
    return render_template("index.html")
@app.route('/result', methods = ['POST'])
def login():
    budg = request.form["budg"]
    genr = request.form["genr"]
    home = request.form["home"]
    lang = request.form["lang"]
    popu = request.form["popu"]
    runt = request.form["runt"]
    stat = request.form["stat"]
    avot = request.form["avot"]
    tvot = request.form["tvot"]
    wday = request.form["wday"]
    mont = request.form["mont"]
    year = request.form["year"]
    t = [[np.cbrt(float(budg)), float(genr), float(home),
float(lang), np.cbrt(float(popu)), float(runt), float(stat),
float(avot), np.cbrt(float(tvot)), float(wday), float(mont),
float(year)]]
    t = scale.transform(t)
    output = model.predict(t)
    output = np.power(output,3)
    return render_template("result.html", y = str(output[0]))
if___name___== '_main_':
app.run(debug = False)
```

HTML code

Index.html

```
<html>
    <head>
        <title>Movie Box Office Gross Prediction</title>
        <link rel="stylesheet" href="{{
url_for('static',filename='style.css')}}">
    </head>
    <body>
        <div>
            <div class="header">
                <div class="header-center">
                    <h1>Movie Box Office Gross Prediction</h1>
                </div>
            </div>

            <div class="box">
                <form action="/result" method="post">
                    <input type="text" placeholder="Enter the
movie's budget" name="budg" required><br>
                    <select name="genr" required>
                        <option value="" disabled selected
hidden>Select the movie's genre</option>
                        <option value="0">Action</option>
                        <option value="1">Adventure</option>
                        <option value="2">Animation</option>
                        <option value="3">Comedy</option>
                        <option value="4">Crime</option>
                        <option value="5">Documentary</option>
                        <option value="6">Drama</option>
                        <option value="7">Family</option>
                        <option value="8">Fantasy</option>
                        <option value="9">Foreign</option>
                        <option value="10">History</option>
                        <option value="11">Horror</option>
                        <option value="12">Music</option>
                        <option value="13">Mystery</option>
                        <option value="14">Romance</option>
                        <option value="15">Science
Fiction</option>
                        <option value="16">TV Movie</option>
                        <option value="17">Thriller</option>
                        <option value="18">War</option>
                        <option value="19">Western</option>
                    </select><br>
                    <select name="home" required>
```

```
          <option value="" disabled selected
hidden>Does the movie have a homepage</option>
          <option value="1">Yes</option>
          <option value="0">No</option>
        </select><br>
        <select name="lang" required>
          <option value="" disabled selected
hidden>Select the language used in the movie</option>
          <option value="0">af</option>
          <option value="1">ar</option>
          <option value="2">cn</option>
          <option value="3">cs</option>
          <option value="4">da</option>
          <option value="5">de</option>
          <option value="6">el</option>
          <option value="7">en</option>
          <option value="8">es</option>
          <option value="9">fa</option>
          <option value="10">fr</option>
          <option value="11">he</option>
          <option value="12">hi</option>
          <option value="13">hu</option>
          <option value="14">id</option>
          <option value="15">is</option>
          <option value="16">it</option>
          <option value="17">ja</option>
          <option value="18">ko</option>
          <option value="19">ky</option>
          <option value="20">nb</option>
          <option value="21">nl</option>
          <option value="22">no</option>
          <option value="23">pl</option>
          <option value="24">ps</option>
          <option value="25">pt</option>
          <option value="26">ro</option>
          <option value="27">ru</option>
          <option value="28">sl</option>
          <option value="29">sv</option>
          <option value="30">ta</option>
          <option value="31">te</option>
          <option value="32">th</option>
          <option value="33">tr</option>
          <option value="34">vi</option>
          <option value="35">xx</option>
          <option value="36">zh</option>
        </select><br>
        <input type="text" placeholder="Enter the
movie's popularity" name="popu" required> <br>
        <input type="text" placeholder="Enter the
movie's runtime" name="runt" required> <br>
        <select name="stat" required>
          <option value="" disabled selected
hidden>Current movie status</option>
          <option value="0">Post Production</option>
          <option value="1">Released</option>
          <option value="2">Rumored</option>
        </select><br>
        <input type="text" placeholder="Enter the
average votes received by the movie" name="avot" required>
<br>
        <input type="text" placeholder="Enter the count
of people who voted for the movie" name="tvot" required>
<br>
        <select name="wday" required>
          <option value="" disabled selected
hidden>Select the day on which the movie was
released</option>
          <option value="0">Monday</option>
          <option value="1">Tuesday</option>
          <option value="2">Wednesday</option>
          <option value="3">Thursday</option>
          <option value="4">Friday</option>
          <option value="5">Saturday</option>
          <option value="6">Sunday</option>
        </select><br>
         <select name="mont" required>
           <option value="" disabled selected
hidden>Select the month in which the movie was
released</option>
           <option value="0">January</option>
           <option value="1">February</option>
           <option value="2">March</option>
           <option value="3">April</option>
           <option value="4">May</option>
           <option value="5">June</option>
           <option value="6">July</option>
           <option value="7">August</option>
           <option value="8">September</option>
           <option value="9">October</option>
           <option value="10">November</option>
           <option value="11">December</option>
         </select><br>
        <input type="text" placeholder="Enter the year
in which the movie was released" name="year" required>
<br>
        <button type="submit" id="submit">Predict
Movie's Revenue</button>
      </form>
    </div>
  </div>
 </body>
</html>
```

Result.html

```
<html>
  <head>
    <title>Movie Box Office Gross Prediction</title>
    <link rel="stylesheet" href="{{
url_for('static',filename='style.css')}}">
  </head>
  <body>
    <div>
      <div class="header">
        <div class="header-center">
          <h1>Movie Box Office Gross Prediction</h1>
        </div>
      </div>
      <div class="box">
        <p>The predicted revenue for the movie is
{{y}}</p>
        <form action="/">
          <button type="submit" id="return">Make
Another Prediction</button>
        </form>
      </div>
    </div>
  </body>
</html>
```

CSS code

```css
.header {
    overflow: hidden;
    background-color: rgba(255, 230, 230, 0.75);
    padding: 20px 10px;
    box-shadow: 20px 20px 60px #F2D1D1,
        -20px -20px 60px #DAEAF1;
}

.header-center {
    text-align: center;
}

@media screen and (max-width: 500px) {
    .header-center {
        float: none;
    }
}

body {
    background-image: linear-gradient(45deg, #DAEAF1,
#C6DCE4);
    font-family: 'Poppins', sans-serif;
    margin: 0;
}

.box {
    border-radius: 20px;
    background-color: rgba(255, 230, 230, 0.75);
    box-shadow: 5px 5px 20px #F2D1D1;
    padding-top: 50px;
    margin: 50px auto;
    overflow: hidden;
    text-align: center;
    width: 600px;
}

input {
    border-radius: 50px;
    padding: 10px 20px;
    background-color: white;
    height: 50px;
    width: 450px;
    margin: 10px;
    border: none;
}

select {
    border-radius: 50px;
    padding: 10px 20px;
    background-color: white;
    height: 50px;
    width: 450px;
    margin: 10px;
    border: none;
}

select:not(:valid) {
    color: gray;
}

#submit          {
    border: none;
    background-color: white;
    cursor: pointer;
    border-radius: 20px;
    padding: 10px 20px;
    height: 50px;
    width: 350px;
    margin-top: 20px;
    margin-bottom:40px;
    border: none;
}

#return          {
    border: none;
    background-color: white;
    cursor: pointer;
    border-radius: 20px;
    padding: 10px 20px;
    height: 50px;
    width: 250px;
    margin-top: 20px;
    margin-bottom:40px;
    border: none;
}

p {
    font-weight: bold;
}
```