BIRMINGHAM CITY University

Academic Report:	Imdb Rating Prediction
Author:	Anupama Rai
Student ID:	24128432
Subject Code:	CMP4294
Date:	June 28, 2024
Wordcount:	1907

Table of contents

Contents

Table of Figures & Tables

Introduction

The film industry has long been a subject of extensive research and analysis, driven by the pursuit to understand what makes a movie successful in terms of critical reception, box office earnings, and audience approval (Bristi, Zaman and Sultana, 2019). The dataset under consideration, **IMDb Top 200 Movies** (*IMDB Movies Dataset*, 2021), encapsulates a wealth of information about some of the most critically acclaimed films listed on the Internet Movie Database (IMDb). This dataset includes various attributes such as IMDb rating, Meta score, number of votes, gross earnings, director, and genre, among others. By leveraging machine learning techniques, this study aims to predict the IMDb rating of a movie based on these attributes, thereby providing insights into the factors that influence a film's reception.

The dataset contains factors that are high likely to influence in IMDb rating of a certain movie such as; genre of the movie, director, number of votes from the audience, the Metacritic score and overall earnings. Based on the dataset, the central problem addressed in this analysis is to determine the correlation between several key factors—genre, director, number of votes, Meta score, and gross earnings—and the IMDb rating of a movie. Specifically, this study aims to deduce the IMDb rating of a movie based on these features. By constructing a predictive model, there has been tries to understand how each factor individually and collectively impacts the IMDb rating, providing insights behind the rating and ranking of the movies. The expected outcome of this analysis is a predictive model that accurately estimates a movie's IMDb rating based on its genre, director, number of votes, Meta score, and gross earnings. The model's performance will be evaluated using metrics such as Mean Squared Error (MSE) and R-squared (R^2) score.

The CSV file containing data of the top 200 movies ranked in IMDb is preprocessed creating binary features and numerical values from encoded genres and director's names allowing the code to use the categorical features in machine learning model. Random Forest Regressor model is then trained to predict a movie's IMDb rating based on the features. Model's accuracy and performance is evaluated using Mean Squared Error(MSE) and R-squared(R^2) metrics.

Literature Review

The entertainment industry is increasingly relying on data analysis to predict movie success. Machine learning models, particularly ensemble methods like Random Forests (Bristi, Zaman and Sultana, 2019), have been widely adopted for this purpose. These models typically consider factors like genre, director's track record, critical reception (Metacritic scores), and box office performance (Goyal and Urolagin, 2022). Despite this progress, there's still a need for a deeper understanding of how these factors influence IMDb ratings, especially regarding the complex relationship between box office success and audience appreciation.

The model this study is bringing forth is also trying to minimize the gap that is still prevalent in the industry, here's a general conceptual diagram of how the system works.

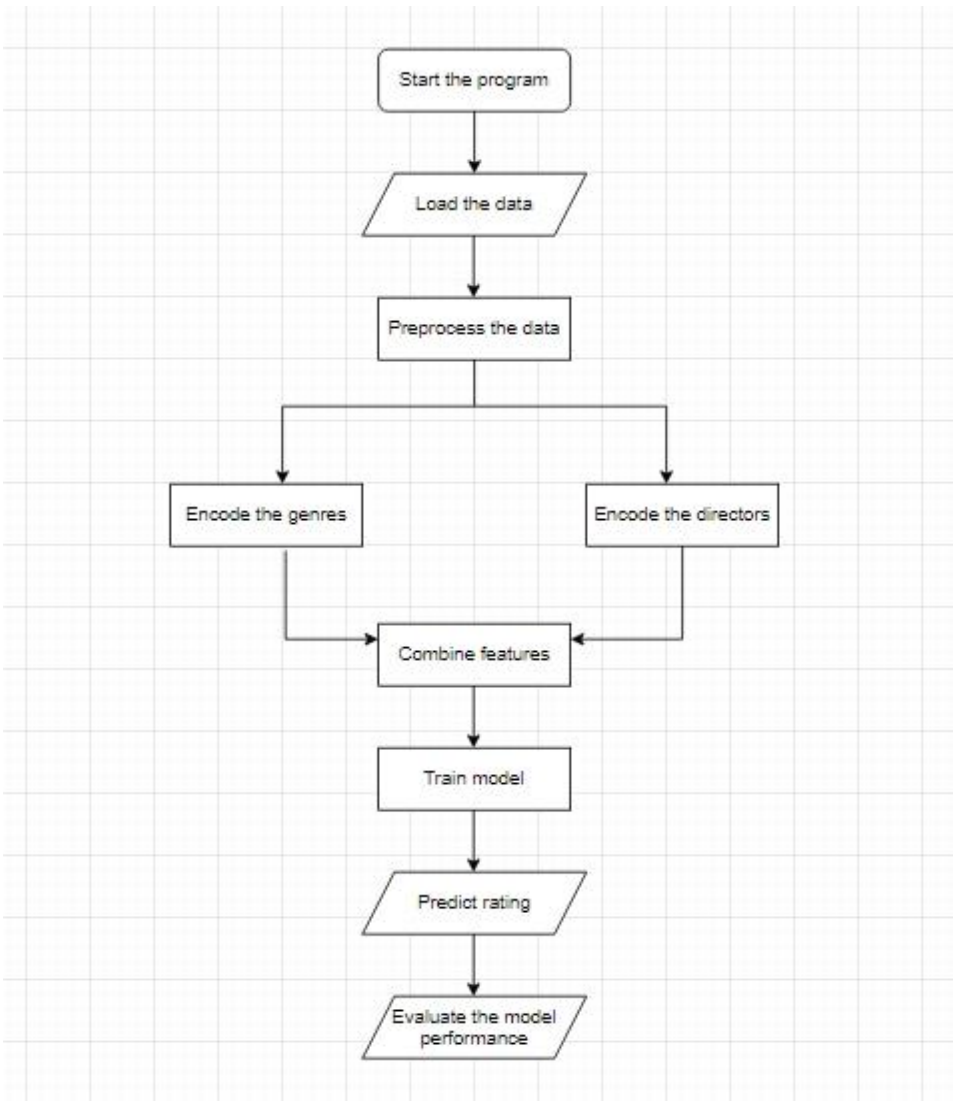


Figure 1: Model's steps

Data Overview

The dataset "imdb_top_200.csv" (*IMDB Movies Dataset*, 2021), presents a curated collection of the top 200 highest-rated films according to IMDb user ratings. This dataset serves as a valuable resource for exploring the factors that contribute to a film's success, particularly within the context of critically acclaimed and audience-beloved cinema. The CSV file consists of 200 rows and 8 columns.

The dataset encompasses a range of influential movie attributes, offering a multifaceted perspective on cinematic achievement. Each entry includes the following information:

- **Series title:** The title of the film, providing immediate identification.
- **Released year:** The year of release, contextualizing the film within a specific historical and cinematic landscape.
- **Genre:** The genre or genres of the film. The presence of multiple genres (separated by commas) indicates the potential for complex thematic explorations and hybrid approaches to storytelling.
- **IMDb rating:** The IMDb rating for the film, representing a direct measure of audience appreciation based on user-submitted reviews. This rating serves as a primary metric for evaluating a film's perceived quality and impact.
- **Meta score:** The Metacritic score for the film, offering a critical consensus score derived from reviews by professional critics. This score allows for an understanding of how a film is perceived by the critical community.
- **Director:** The name of the film's director, highlighting the significant role of individual creative vision in shaping a cinematic work.
- **No. of Votes:** The number of votes a film has received on IMDb. This serves as an indicator of the film's overall reach and popularity, demonstrating the breadth of its audience appeal.
- **Gross:** The film's box office gross earnings in US dollars. This provides a measure of the film's commercial success, reflecting its ability to attract viewers and generate revenue.

Among these features included in the data, features that are highly influential to the rating are the only ones extracted i.e. genre, director, gross, meta score and number of votes and the model is trained on the basis of those features, not all of the above provided in the dataset were used for development of this model.

Model Building Process & Approach

Approach

This program utilizes a predictive machine learning approach to estimate IMDb ratings for movies. Instead of simply describing the data (which is the domain of descriptive analysis), the program aims to build a model that can predict the IMDb rating of a movie based on specific input features.

The core of this approach is the use of a **Random Forest Regressor model**. This is a powerful predictive technique that combines multiple decision trees to make more accurate and robust predictions. By training this model on a dataset of the top 200 movies based on IMDb ratings, the program learns to identify patterns and relationships between movie features (like genre, director, Metacritic score, and box office gross) and the corresponding IMDb scores.

The process involves:

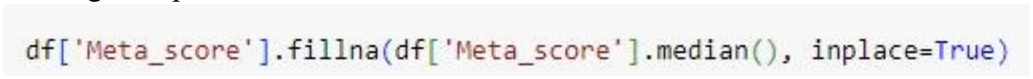
1. **Data Preprocessing:** The program prepares the data for use by the machine learning model. This involves:
   - **Handling Missing Values:** Addressing missing data in the 'Meta_score' and 'Gross' columns by filling them with the respective median values.
   - **Feature Encoding:** Converting categorical features (genres and directors) into numerical representations using one-hot encoding and label encoding, respectively.
2. **Model Training:** The program trains the Random Forest Regressor model using the preprocessed data. The model learns the relationships between features and IMDb ratings.
3. **Model Evaluation:** The program evaluates the trained model's performance using metrics like Mean Squared Error (MSE) and R-squared (R^2). This assessment determines how well the model can predict IMDb ratings for unseen data.
4. **Prediction:** The program provides a function to predict the IMDb rating for a new movie based on user input for its genre, director, Metacritic score, number of votes, and gross earnings.

A. Data Preparation

Data preprocessing is a crucial step in preparing the dataset for machine learning. It involves transforming the raw data into a format that is suitable for the model and enhances its accuracy(GeeksforGeeks, 2023). Here's a breakdown of the preprocessing steps of the model presented:

1. Handling missing values

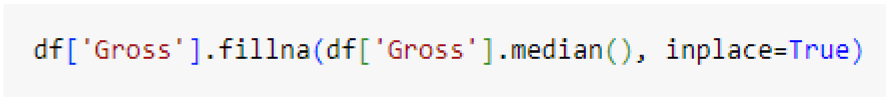
Meta_score: The code fills in missing values in the "Meta_score" column with the median value of the column. This ensures that the model doesn't encounter issues due to missing data points.



```
df['Meta_score'].fillna(df['Meta_score'].median(), inplace=True)
```

Figure 2: Meta score handling

Gross: The code similarly handles missing values in the "Gross" column by filling them with the median gross earnings.



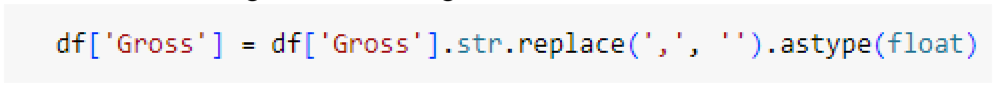
```
df['Gross'].fillna(df['Gross'].median(), inplace=True)
```

Figure 3: Gross values handling

2. Cleaning and Converting 'Gross' Column

Removing Commas: The code uses str.replace(',', '') to remove commas from the "Gross" column, ensuring that the values are purely numerical and can be converted to floating-point numbers.

Converting to Float: The astype(float) function then converts the cleaned "Gross" values into floating-point numbers, making them suitable for numerical calculations and machine learning model training.

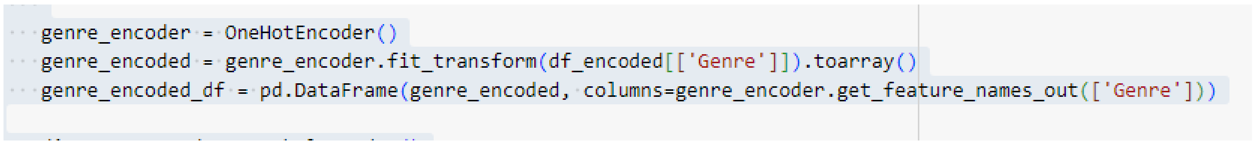


```
df['Gross'] = df['Gross'].str.replace(',', '').astype(float)
```

Figure 4: Cleaning and Converting

3. One-Hot Encoding Genres

Categorical to Binary: One-hot encoding is used to convert the categorical "Genre" column into a set of binary features, allowing the model to treat each genre as an independent variable. The OneHotEncoder is trained on the "Genre" column. It transforms each genre into a separate binary feature (0 or 1). The resulting encoded features are then added to the DataFrame as new columns.

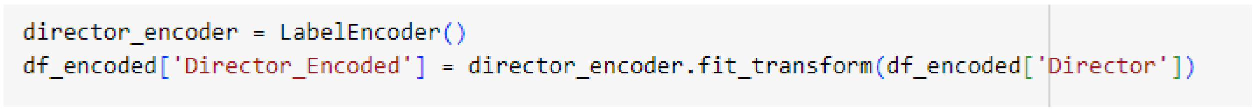


```
genre_encoder = OneHotEncoder()
genre_encoded = genre_encoder.fit_transform(df_encoded[['Genre']]).toarray()
genre_encoded_df = pd.DataFrame(genre_encoded, columns=genre_encoder.get_feature_names_out(['Genre']))
```

Figure 5: Genre encoding

4. Label Encoding Directors

   Categorical to Numerical: The code uses LabelEncoder to convert the "Director" column (which contains text values) into a set of numerical values. This enables the model to interpret directors as numerical features. It assigns a unique numerical value to each distinct director name.

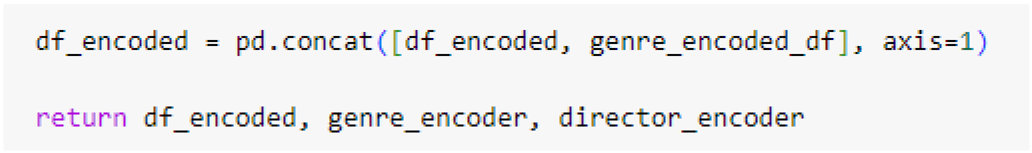


```
director_encoder = LabelEncoder()
df_encoded['Director_Encoded'] = director_encoder.fit_transform(df_encoded['Director'])
```

Figure 6: Director encoding

5. Combining Features

   Creating a Single Dataset: The one-hot encoded genre features and the label-encoded director feature are added to the original DataFrame, creating a single dataset with all the features the model needs.



```
df_encoded = pd.concat([df_encoded, genre_encoded_df], axis=1)

return df_encoded, genre_encoder, director_encoder
```

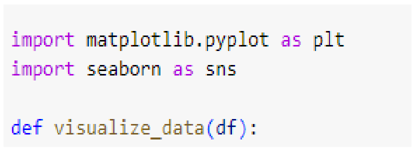
Figure 7: Combining features

B. Exploring Data Analysis

Visualization is the process of using graphical representations to explore, understand, and communicate patterns, trends, and insights within data (GeeksforGeeks, 2024). Visualizations are often more effective than simply looking at tables of numbers because they allow us to quickly grasp relationships and make connections.

1. Visualize Data Function

   This function is designed to create two visualizations: one for genre and one for directors. It takes a pandas DataFrame (`df`) as input, which is assumed to be the preprocessed DataFrame containing the features and IMDb ratings.

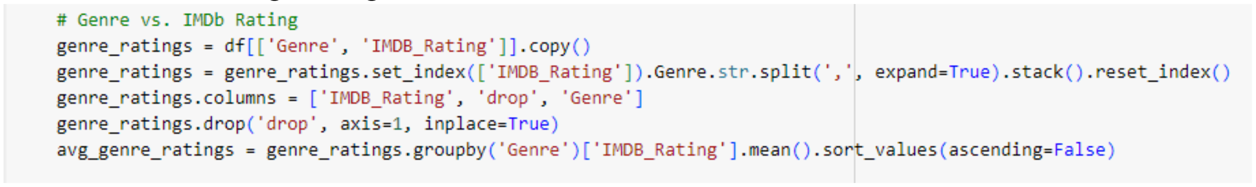


```
import matplotlib.pyplot as plt
import seaborn as sns

def visualize_data(df):
```

Figure 8: Import and visualize_data function

2. Genre vs. IMDb Rating

   The function first creates a new DataFrame called genre_ratings containing only the 'Genre' and 'IMDB_Rating' columns. To prepare the data for visualization, the code performs a series of manipulations. It sets the 'IMDB_Rating' as the index, splits the multiple genres listed in the 'Genre' column into separate columns, and reshapes the DataFrame so each row represents a single genre with its corresponding rating. It then drops an unnecessary 'drop' column created during the reshaping process. Finally, it calculates the average IMDb rating for each genre and sorts the genres in descending order based on these average ratings.

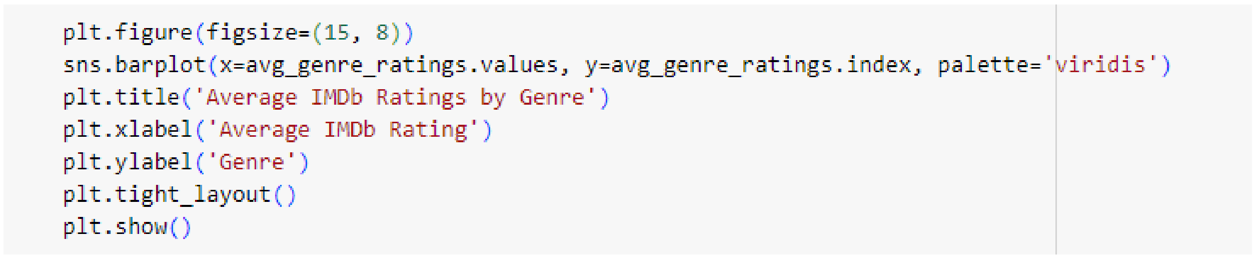


```python
# Genre vs. IMDb Rating
genre_ratings = df[['Genre', 'IMDB_Rating']].copy()
genre_ratings = genre_ratings.set_index(['IMDB_Rating']).Genre.str.split(',', expand=True).stack().reset_index()
genre_ratings.columns = ['IMDB_Rating', 'drop', 'Genre']
genre_ratings.drop('drop', axis=1, inplace=True)
avg_genre_ratings = genre_ratings.groupby('Genre')['IMDB_Rating'].mean().sort_values(ascending=False)
```

Figure 9: Genre vs. IMDb rating

3. Genre Bar Plotting

   Next, the code creates a bar plot using seaborn. It uses the average IMDb ratings as the x-axis values and the genre names as the y-axis values. A visually pleasing color palette ('viridis') is applied. The plot is then titled, labeled, and adjusted for spacing before being displayed.

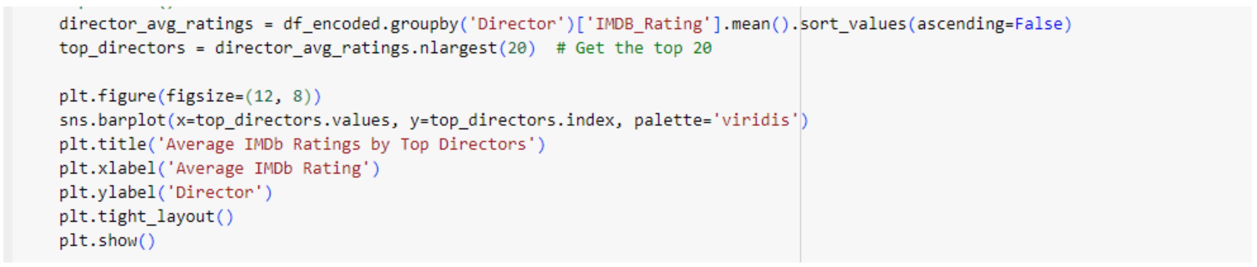


```python
plt.figure(figsize=(15, 8))
sns.barplot(x=avg_genre_ratings.values, y=avg_genre_ratings.index, palette='viridis')
plt.title('Average IMDb Ratings by Genre')
plt.xlabel('Average IMDb Rating')
plt.ylabel('Genre')
plt.tight_layout()
plt.show()
```

Figure 10: Genre bar graph

4. Director vs. IMDb Rating

   This function visualizes the relationship between director and IMDb rating. It again calculates the average IMDb ratings for each director, but this time, it selects only the top 20 directors with the highest average ratings. This ensures a more manageable and interpretable plot. It then uses this data to create a bar plot, mirroring the structure of the genre plot, with the average IMDb ratings on the x-axis and the director names on the y-axis.



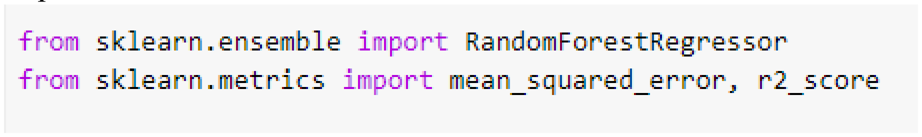
```
    director_avg_ratings = df_encoded.groupby('Director')['IMDB_Rating'].mean().sort_values(ascending=False)
    top_directors = director_avg_ratings.nlargest(20)  # Get the top 20

    plt.figure(figsize=(12, 8))
    sns.barplot(x=top_directors.values, y=top_directors.index, palette='viridis')
    plt.title('Average IMDb Ratings by Top Directors')
    plt.xlabel('Average IMDb Rating')
    plt.ylabel('Director')
    plt.tight_layout()
    plt.show()
```

Figure 11: Director vs. IMDb rating

C. Model Building

1. Import Necessary libraries

Random Forest Regressor model is imported from the sklearn.ensemble module. Also the mean squared error and r2 score functions for evaluating regression models are also imported.

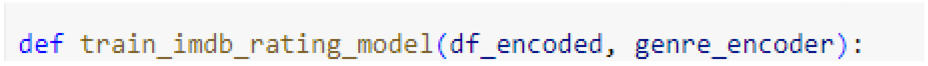


```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
```

Figure 12: Import libraries

2. 'Train_imdb_rating_model' Function

This function takes the preprocessed DataFrame and the genre encoder (genre_encoder) as input and trains a Random Forest model to predict IMDb ratings.

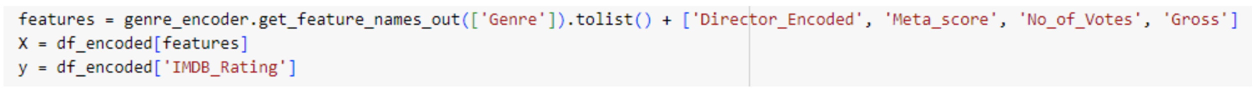


```
def train_imdb_rating_model(df_encoded, genre_encoder):
```

Figure 13: Train imdb function

3. Prepare features

'features' defines a list of feature names, combining the one-hot encoded genre features from genre encoder with the 'Director_Encoded', 'Meta_score', 'No_of_Votes', and 'Gross' columns. **'X = df_encoded[features]'** extracts the features from the DataFrame and stores them in a new DataFrame called X. **'y = df_encoded['IMDB_Rating']'** extracts the IMDb ratings from the DataFrame and stores them in a new series called Y.



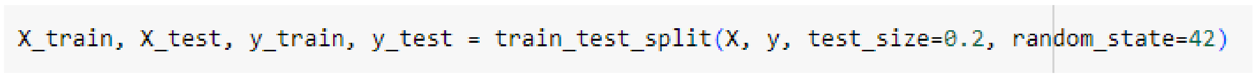
```
features = genre_encoder.get_feature_names_out(['Genre']).tolist() + ['Director_Encoded', 'Meta_score', 'No_of_Votes', 'Gross']
X = df_encoded[features]
y = df_encoded['IMDB_Rating']
```

Figure 14: Features and target preparation

4. Train-test Split

   This step splits the data into training and testing sets. This is crucial for evaluating the model's ability to generalize to unseen data. The function splits the data into:

   - X_train: Training data features.
   - X_test: Test data features.
   - y_train: Training data targets (IMDb ratings).
   - y_test: Test data targets.
   - **test_size=0.2:** Specifies that 20% of the data will be used for testing.
   - **random_state=42:** Sets a seed for the random number generator, ensuring consistent splitting across multiple runs.

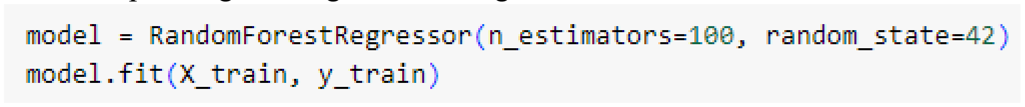


```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 15: Train-test and split

5. Create and Train the Random Forest Model

   An instance of the Random Forest Regressor is created with 100 trees and a random seed for reproductibility. Then the Random Forest model is trained on the training features and the corresponding training IMDb ratings.

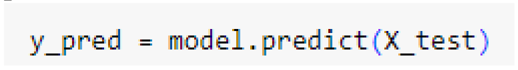


```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Figure 16: Create Random Forest model

6. Make Predictions

   The trained model is used to predict IMDb ratings for the test data features(x_test). The predictions are stored in a Series called 'y_pred'.

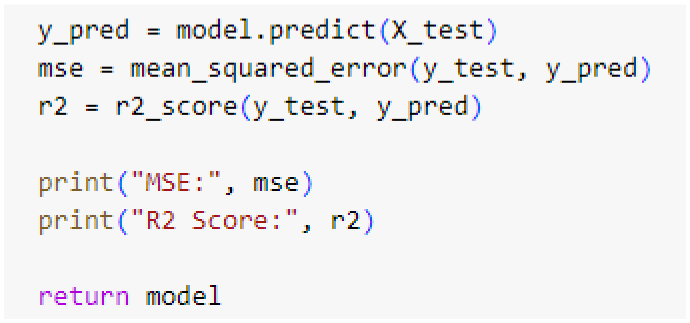


```
y_pred = model.predict(X_test)
```

Figure 17: make predictions

7. Evaluate the Model's Performance

   Mean squared error measures the average squared difference between the model's predictions and the actual IMDb ratings. Similarly, R-squared (R^2), which indicates how well the model explains the variation in the target variable (IMDb ratings).

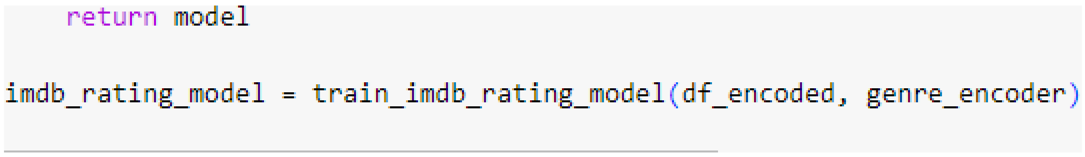


```
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("MSE:", mse)
print("R2 Score:", r2)

return model
```

Figure 18: Evaluate model's performance

8. Return and Save the Model

The 'train_imdb_rating_model' function returns the trained model, which allows user to use the model to make prediction on new data. Then the function is called and stored in a variable called 'imdb_rating_model'.



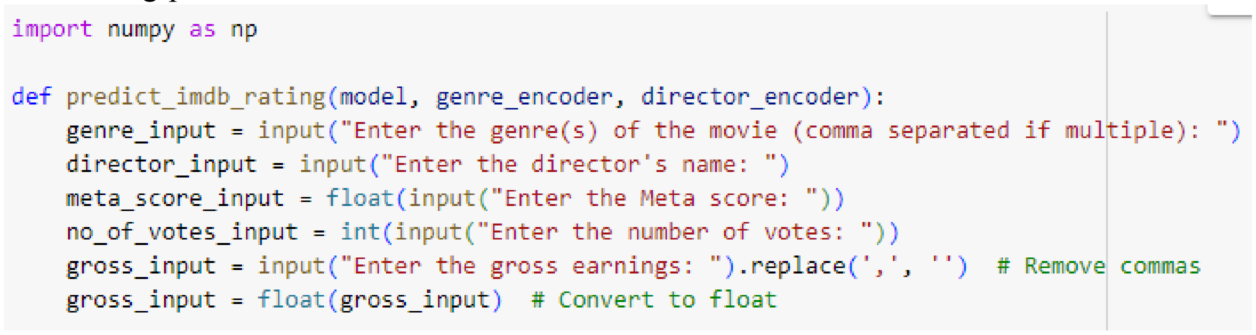
```
    return model

imdb_rating_model = train_imdb_rating_model(df_encoded, genre_encoder)
```

Figure 19: Save the model

D. Prediction

1. Get User Input

The code prompts the user to enter information about the movie they want to predict a rating for such as; genre, director, meta score, number of votes and gross earnings. The code handles basic input validation, first commas are removed from the gross earnings input to ensure it can be converted to a number, then gross earnings input is converted to a floating-point number.

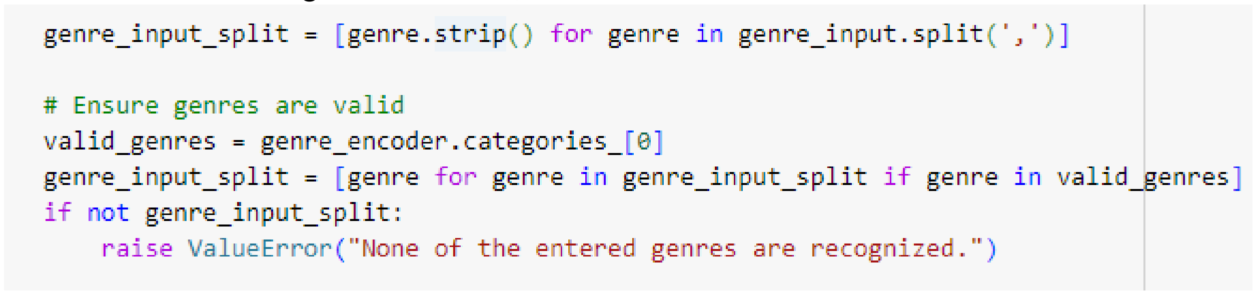


```
import numpy as np

def predict_imdb_rating(model, genre_encoder, director_encoder):
    genre_input = input("Enter the genre(s) of the movie (comma separated if multiple): ")
    director_input = input("Enter the director's name: ")
    meta_score_input = float(input("Enter the Meta score: "))
    no_of_votes_input = int(input("Enter the number of votes: "))
    gross_input = input("Enter the gross earnings: ").replace(',', '')  # Remove commas
    gross_input = float(gross_input)  # Convert to float
```

Figure 20: User input

2. Process Genre Input

   The user-provided genres are split into a list of individual genres, separating them if the user entered multiple genres separated by commas. The code checks if the entered genres are valid. It compares the user’s input against the list of genres the model was trained on. If no valid genres are found, a ValueError is raised.

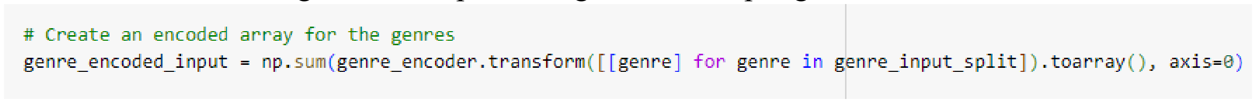


```
genre_input_split = [genre.strip() for genre in genre_input.split(',')]

# Ensure genres are valid
valid_genres = genre_encoder.categories_[0]
genre_input_split = [genre for genre in genre_input_split if genre in valid_genres]
if not genre_input_split:
    raise ValueError("None of the entered genres are recognized.")
```

Figure 21: Process genre input

3. Encode Genre Input

   The genre_encoder (which was trained during preprocessing) is used to one-hot encode the user's genre inputs. If multiple genres were entered, the encoded vectors are summed to create a single vector representing all of the input genres.

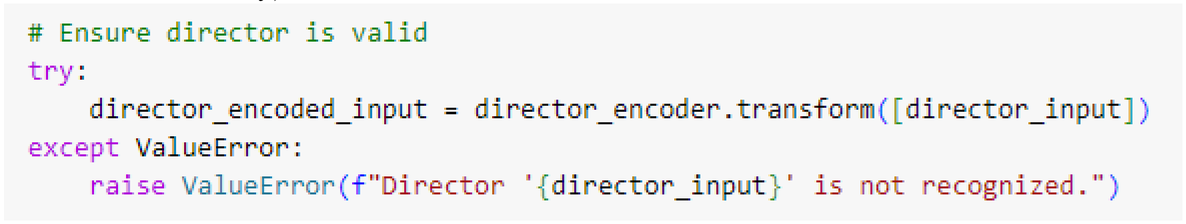


```
# Create an encoded array for the genres
genre_encoded_input = np.sum(genre_encoder.transform([[genre] for genre in genre_input_split]).toarray(), axis=0)
```

Figure 22: Encode genre input

4. Encode Director Input

   The code attempts to use the director_encoder (trained during preprocessing) to label-encode the director input. If the director name is not recognized (i.e., not in the encoder's vocabulary), a ValueError is raised.

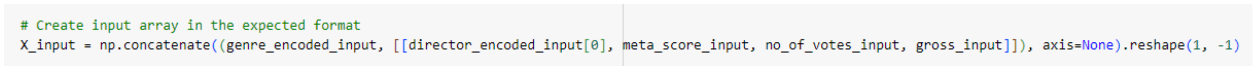


```
# Ensure director is valid
try:
    director_encoded_input = director_encoder.transform([director_input])
except ValueError:
    raise ValueError(f"Director '{director_input}' is not recognized.")
```

Figure 23: Encode director input

5. Create Input Array

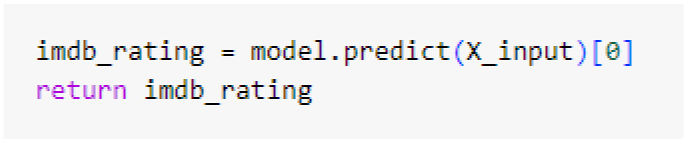
   First the input array (X_input) that the model will use to make a prediction is created. The encoded genres, director, Meta score, number of votes, and gross earnings are combined into a single array. The array is reshaped to match the expected shape of the data the model was trained on (in this case, a 2D array with one row and multiple columns).



```
# Create input array in the expected format
X_input = np.concatenate((genre_encoded_input, [[director_encoded_input[0], meta_score_input, no_of_votes_input, gross_input]]), axis=None).reshape(1, -1)
```

Figure 24: Create input array

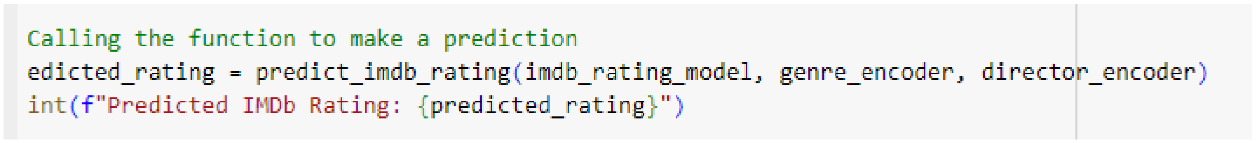
6. Make the Prediction
The trained Random Forest model (model) is used to make a prediction based on the constructed input array (X_input). Since the prediction is for a single movie (one row in the input array), the first element of the prediction array is extracted. Then the predicted IMDb rating is returned to the caller.



```
imdb_rating = model.predict(X_input)[0]
return imdb_rating
```

Figure 25: Make prediction

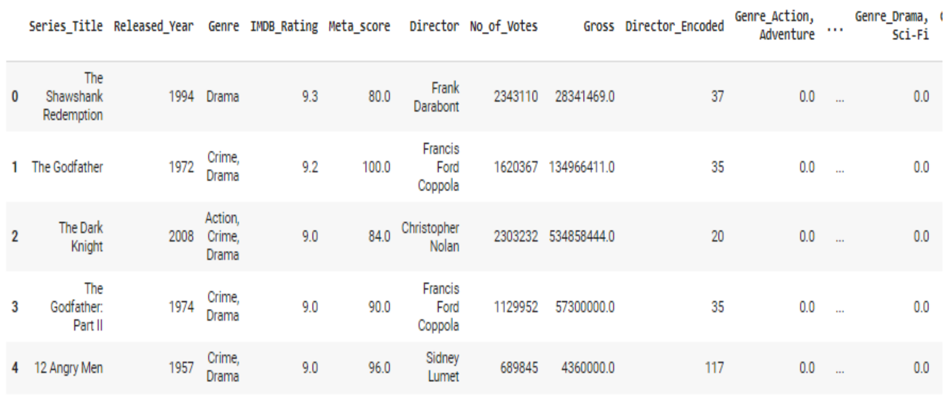
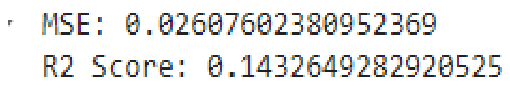
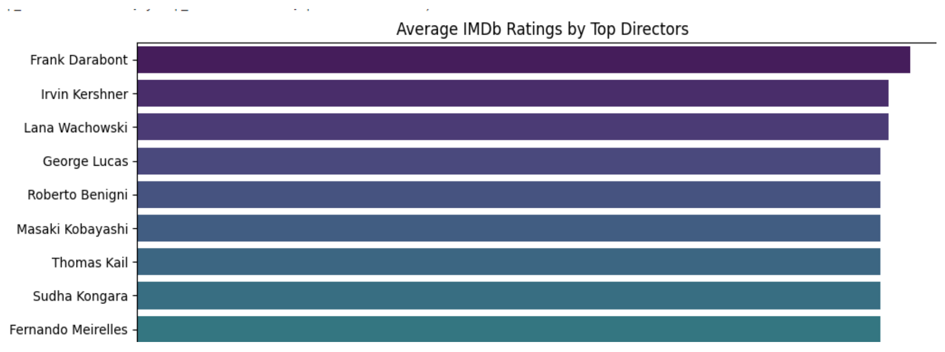
7. Print the Prediction
The predict_imdb_rating function is called, and the predicted rating is printed as output.



```
Calling the function to make a prediction
edicted_rating = predict_imdb_rating(imdb_rating_model, genre_encoder, director_encoder)
int(f"Predicted IMDb Rating: {predicted_rating}")
```

Figure 26: Print the prediction

Result

Program Features	Expected Result	Actual result	Functionality
Data preprocessing and loading	The missing values are handled well, required features are encoded and the data is loaded in a table.		working
Model training and accuracy checking	With the encoded features, model is trained and the model's accuracy is also checked.		working
Visualization of average IMDb ratings by genre and directors	The bar graph that depicts the ratings is shown.		working

	Series_Title	Released_Year	Genre	IMDB_Rating	Meta_score	Director	No_of_Votes	Gross	Director_Encoded	Genre_Action, Adventure	...	Genre_Drama, Sci-Fi
0	The Shawshank Redemption	1994	Drama	9.3	80.0	Frank Darabont	2343110	28341469.0	37	0.0	...	0.0
1	The Godfather	1972	Crime, Drama	9.2	100.0	Francis Ford Coppola	1620367	134966411.0	35	0.0	...	0.0
2	The Dark Knight	2008	Action, Crime, Drama	9.0	84.0	Christopher Nolan	2303232	534858444.0	20	0.0	...	0.0
3	The Godfather: Part II	1974	Crime, Drama	9.0	90.0	Francis Ford Coppola	1129952	57300000.0	35	0.0	...	0.0
4	12 Angry Men	1957	Crime, Drama	9.0	96.0	Sidney Lumet	689845	4360000.0	117	0.0	...	0.0

```
MSE: 0.02607602380952369
R2 Score: 0.1432649282920525
```

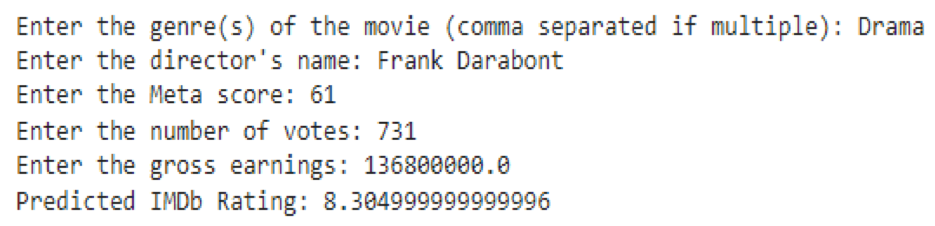
Predicts the IMDb rating of the movie	Based on the taken input, predicts the IMDb rating of the certain movie.	 `Enter the genre(s) of the movie (comma separated if multiple): Drama`
		`Enter the director's name: Frank Darabont`
		`Enter the Meta score: 61`
		`Enter the number of votes: 731`
		`Enter the gross earnings: 136800000.0`
		`Predicted IMDb Rating: 8.304999999999996`
			working

Table 1: Test case of output

Conclusion

This report investigated the feasibility of predicting IMDb ratings for movies using a predictive machine learning approach. The analysis focused on a dataset of the top 200 movies ranked on IMDb, exploring the influence of several key features: genre, director, Meta score, number of votes, and gross earnings.

Data preprocessing techniques, including handling missing values, one-hot encoding for genres, and label encoding for directors, were employed to prepare the dataset for model training. A Random Forest Regressor model was chosen for its ability to handle complex datasets and identify nonlinear relationships, demonstrating the predictive nature of the program.

The model's performance was evaluated using Mean Squared Error (MSE) and R-squared (R^2) metrics. While the model achieved a relatively low MSE, indicating accurate predictions in terms of numerical proximity, the R^2 score was also relatively low. This suggests that while the model can make accurate predictions on average, it may not fully capture the intricate factors that contribute to movie success and, as a result, might not always be reliable for predicting the exact IMDb rating of a movie.

This analysis highlights the challenges of accurately predicting subjective elements like movie ratings. Further research could focus on:

- **Expanding the dataset:** Incorporating a larger and more diverse dataset of movies could lead to more robust models.
- **Exploring additional features:** Including other potentially influential factors like cast, release date, and critical reviews from different platforms might enhance model accuracy.
- **Hyperparameter tuning:** Optimizing the hyperparameters of the Random Forest model could potentially improve its performance.
- **Comparing with other models:** Investigating other machine learning algorithms, such as neural networks or gradient boosting models, could provide valuable insights.

Despite the limitations observed, this study demonstrates the potential of machine learning for understanding and predicting movie ratings. By further refining techniques and incorporating richer datasets, researchers can continue to advance our understanding of the complex factors that contribute to a film's success.

References

Bristi, W.R., Zaman, Z. and Sultana, N. (2019) 'Predicting IMDb Rating of Movies by Machine Learning Techniques,' *Predicting IMDb Rating of Movies by Machine Learning Techniques* . https://doi.org/10.1109/icccnt45670.2019.8944604.

IMDB Movies Dataset (2021). https://www.kaggle.com/datasets/harshitshankhdhar/imdb-dataset-of-top-1000-movies-and-tv-shows.

Goyal, A. and Urolagin, S. (2022) 'Prediction of movie success on IMDB database using machine learning techniques,' in *Algorithms for intelligent systems*, pp. 273–288. https://doi.org/10.1007/978-981-16-6460-1_20.

GeeksforGeeks (2024) *What is Data Visualization and Why is It Important?* https://www.geeksforgeeks.org/data-visualization-and-its-importance/.

GeeksforGeeks (2023) *Data preprocessing in data mining*. https://www.geeksforgeeks.org/data-preprocessing-in-data-mining/.