Development

Feature engineering:

Release Year Features:

Extract the year from the "Release Year" column.

Consider creating binary features to indicate the decade of release (e.g., 2000s, 2010s, 2020s).

Genre Features:

Create binary features for each unique genre. If a movie belongs to a particular genre, set the corresponding feature to 1; otherwise, set it to 0.

Director and Actor Features:

Calculate statistics for directors and actors who appear in multiple movies:

Average IMDb score of movies directed by a specific director.

Average IMDb score of movies featuring a particular actor.

Runtime Features:

Categorize runtime into buckets (e.g., short, medium, long).

Calculate the average IMDb score for each runtime category.

Country Features:

Create binary features for each unique country of origin. Indicate whether the movie is from a specific country.

Language Features:

Similar to the country features, create binary features for each unique language spoken in the movie.

Number of Awards Features:

Create a feature indicating the number of awards or nominations a movie has received.

Production Company Features:

Engineer features related to the production companies, such as the number of movies produced by a specific company and their average IMDb scores.

Rating Features:

Use a one-hot encoding for the movie rating (e.g., G, PG, PG-13, R, etc.).

Movie Duration Features:

Create features based on the movie's duration, such as the number of minutes or categories like "short," "medium," and "long."

User Reviews and Ratings:

If available, consider including user reviews and ratings from other platforms (e.g., Rotten Tomatoes, Metacritic, user reviews) as features.

Sentiment Analysis Features:

Perform sentiment analysis on user reviews to create features based on the overall sentiment of reviews.

Interaction Features:

Create interaction features that combine the effects of two or more existing features. For example, a feature that multiplies the number of user reviews by the average user rating.

Here's a simplified example of how you might perform some of this feature engineering in Python:

```
# Extracting year and creating decade features

data['Release_Year'] = data['Release Year'].str.extract(r'(\d{4})').astype(int)

data['Decade_2000s'] = (data['Release_Year'] >= 2000) & (data['Release_Year'] < 2010)

data['Decade_2010s'] = (data['Release_Year'] >= 2010) & (data['Release_Year'] < 2020)

data['Decade_2020s'] = (data['Release_Year'] >= 2020)


# Creating genre features
```

```
genres = data['Genre'].str.get_dummies(',')

data = pd.concat([data, genres], axis=1)


# Calculate average IMDb scores for directors

director_avg_scores = data.groupby('Director')['IMDb'].mean().reset_index()

director_avg_scores.columns = ['Director', 'Director_Avg_IMDb']

data = data.merge(director_avg_scores, on='Director', how='left')


# Calculate average IMDb scores for actors

actor_avg_scores = data.groupby('Actors')['IMDb'].mean().reset_index()

actor_avg_scores.columns = ['Actors', 'Actors_Avg_IMDb']

data = data.merge(actor_avg_scores, on='Actors', how='left']
```

This is just a starting point, and you can further customize your feature engineering based on the specific characteristics of your dataset and your modeling goals.

After feature engineering, you can proceed with model training and evaluation .

Model training:

Data Preprocessing and Feature Selection:

Preprocess the dataset by performing the feature engineering steps mentioned earlier.

Select the features you want to include in your model and define the target variable (IMDb scores).

```
import pandas as pd

# Load and preprocess your dataset (feature engineering)

# Define your features and target variable
```

Split the Data:

Split the dataset into a training set and a testing set. This allows you to train the model on one subset and evaluate it on another.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Choose a Model:

Choose a regression model to predict IMDb scores. For simplicity, let's use a Linear Regression model in this example. You can explore more advanced models if needed.

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
```

Train the chosen model using the training data.

```
model.fit(X_train, y_train)
```

Model Evaluation:

Evaluate the model using appropriate regression metrics, such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2).

```
from sklearn.metrics import mean_squared_error, r2_score
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
rmse = mse ** 0.5
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

Hyperparameter Tuning (Optional):

Depending on the model's performance, you may need to fine-tune hyperparameters to improve accuracy.

Model Interpretation:

Analyze the model's coefficients to understand which features have the most significant impact on IMDb scores.

Continuous Improvement:

As new data becomes available or as IMDb scores change over time, consider retraining and updating your model.

The code provided is a basic example using linear regression.

For more accurate predictions, you can experiment with other regression models

(e.g., Random Forest, Gradient Boosting, or neural networks) and explore more advanced feature engineering techniques.

Be sure to preprocess your data, handle missing values, and scale/normalize features as necessary.

Evaluation:

You can use regression evaluation metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2):

Data Preprocessing and Model Preparation:

Preprocess the dataset, perform feature engineering, and prepare the model, as mentioned in the previous responses.

Model Evaluation:

Once the model is trained and predictions are made, evaluate the model using various regression metrics.

```
from sklearn.metrics import mean_squared_error, r2_score

# Assuming you have trained the model and obtained predictions as y_pred

# Calculate the Mean Squared Error (MSE)

mse = mean_squared_error(y_test, y_pred)


# Calculate the Root Mean Squared Error (RMSE)

rmse = mse ** 0.5
```

```
# Calculate R-squared (R²)
r2 = r2_score(y_test, y_pred)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared (R²):", r2)
```

Visualization (Optional):

Create visualizations to help understand model performance.

For instance, you can plot the actual IMDb scores against predicted scores to visualize how well the model aligns with the actual values.

```
import matplotlib.pyplot as plt


# Visualize actual IMDb scores vs. predicted scores
plt.scatter(y_test, y_pred)
plt.xlabel("Actual IMDb Scores")
plt.ylabel("Predicted IMDb Scores")
plt.title("Actual vs. Predicted IMDb Scores")
plt.show()
```

Baseline Model Comparison (Optional):

Compare the model's performance against a baseline model. One simple baseline is predicting the mean IMDb score for all movies.

Model Interpretation:

Analyze the model's coefficients (for linear models) or feature importance (for tree-based models) to understand which features have the most significant impact on IMDb scores.

Continuous Improvement:

Continuously monitor and update your model as new data becomes available. IMDb scores can change over time due to new user reviews and ratings.

Reporting and Presentation:

Summarize the model's performance, findings, and any limitations in a report or presentation.

Remember that model evaluation is a crucial step, and the choice of metrics depends on your specific goals. Additionally, the quality of your data, feature engineering, and model training process will have a significant impact on the model's performance, so continuous improvement is essential.