Movies/TV Shows Popularity Prediction

NETFLIX

XueYi Lu, Ryan Sambila, Maritza Ahumada, Iris Michelsen, Chai Kota

Summary

1. Introduction
2. Data cleaning
3. Tableau, SQL
4. Visuals
5. Model, Highest IMDb Score prediction
6. Conclusion

Data Set from Kaggle
[IMDB 5000 Movie Dataset (kaggle.com)]

color	director_name Name of the Director of the Movie	num_critic_for_re...	duration	director_faceboo...	actor_3_facebook...	actor_2_name	actor_1_facebook... Number of likes of the Actor_1 on his/her Facebook Page	gross	genres

Goal: Creating a model to predict popular shows/movies on Netflix for the next year.

Data cleaning

- Data was cleaned using python pandas.
- Removed columns that we did not find relevant to our dataset to use in the model.
  - Language, budget, duration, keywords, color
- Exported to a csv file to store and use.

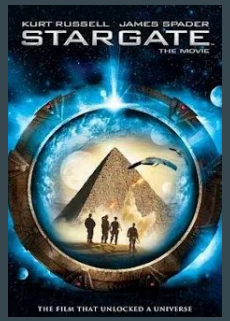
```
import pandas as pd
```

```
file_path = '/Users/xueyilu/Desktop/movie_metadata.csv'
df = pd.read_csv(file_path)
df.head()
```

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_faceboo
0	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	4

```
In [19]: columns_to_remove = ['language', 'country', 'budget', 'content_rating', 'aspect_ratio', 'duration', 'gross', 'plot_keywords

         df_cleaned = df.drop(columns=columns_to_remove)

In [20]: df_cleaned.head()
```

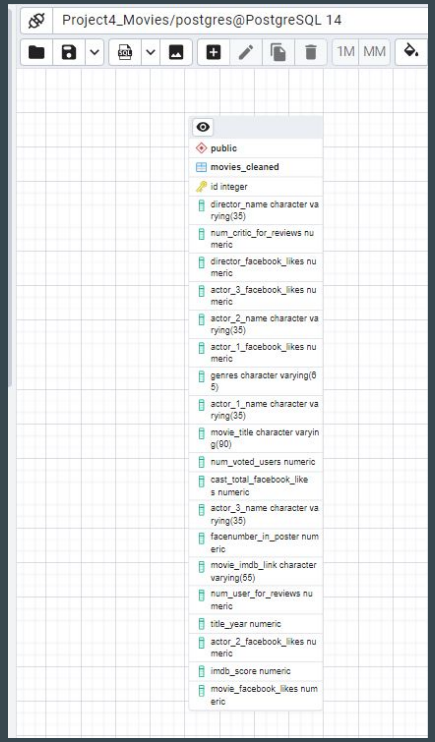
```
In [21]: df_cleaned.to_csv('cleaned_data.csv', index=False)
```



SQL/PGAdmin

- Storage of our cleaned dataset was in a SQL database and using PGAdmin.
- Our dataset is just one table, so we just used a single table in our database

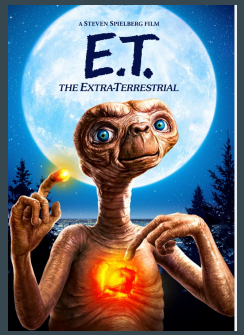
```sql

--Create Table to store clean movie data

CREATE TABLE movies_cleaned (
        id serial PRIMARY KEY,
        director_name VARCHAR (35),
        num_critic_for_reviews DECIMAL,
        director_facebook_likes  DECIMAL,
        actor_3_facebook_likes DECIMAL,
        actor_2_name VARCHAR (35),
        actor_1_facebook_likes DECIMAL,
        genres VARCHAR (65),
        actor_1_name VARCHAR (35),
        movie_title VARCHAR (90),
        num_voted_users DECIMAL,
        cast_total_facebook_likes DECIMAL,
        actor_3_name VARCHAR (35),
        facenumber_in_poster DECIMAL,
        movie_imdb_link VARCHAR (55),
        num_user_for_reviews DECIMAL,
        title_year DECIMAL,
        actor_2_facebook_likes DECIMAL,
        imdb_score DECIMAL,
        movie_facebook_likes DECIMAL
);

--Query Table
SELECT * FROM movies_cleaned
```

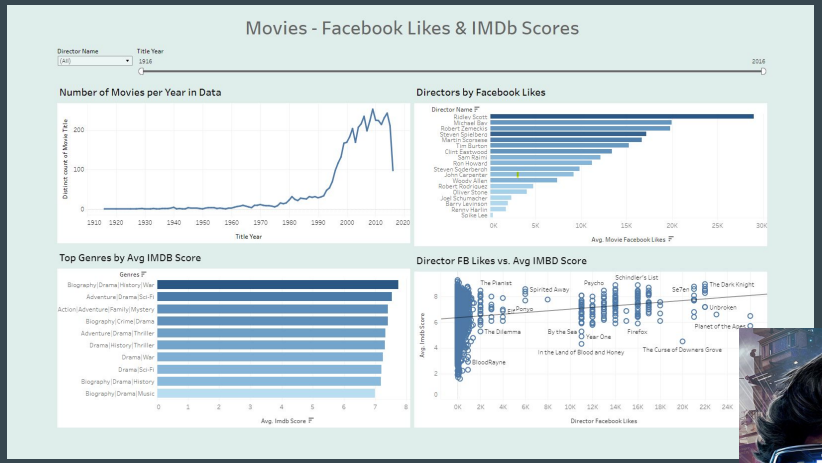


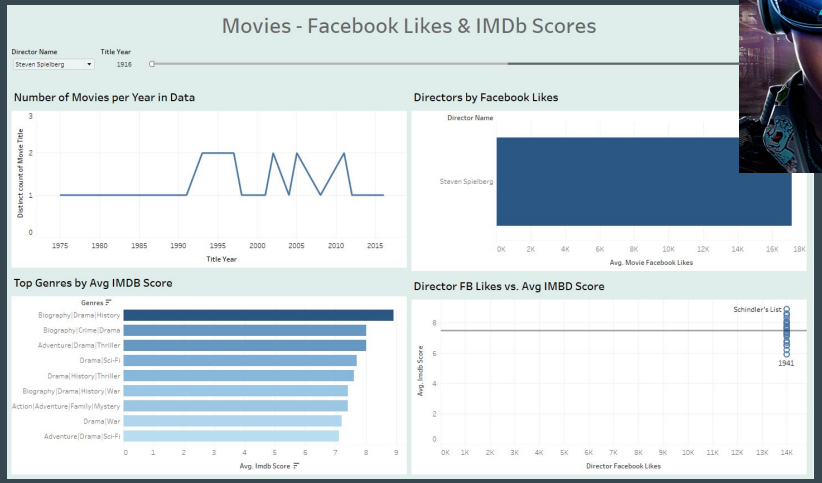
Tableau

- To analyze the data in interactive charts we used Tableau.
- Shows a line chart of the number of movies each year in this dataset,
- Bar chart of most popular genres by average IMDb score.
- Bar chart of most popular directors by movie facebook likes
- Scatter chart showing Directors Facebook likes vs average IMDb score.
- Filters added to look at time periods and a dropdown for directors.

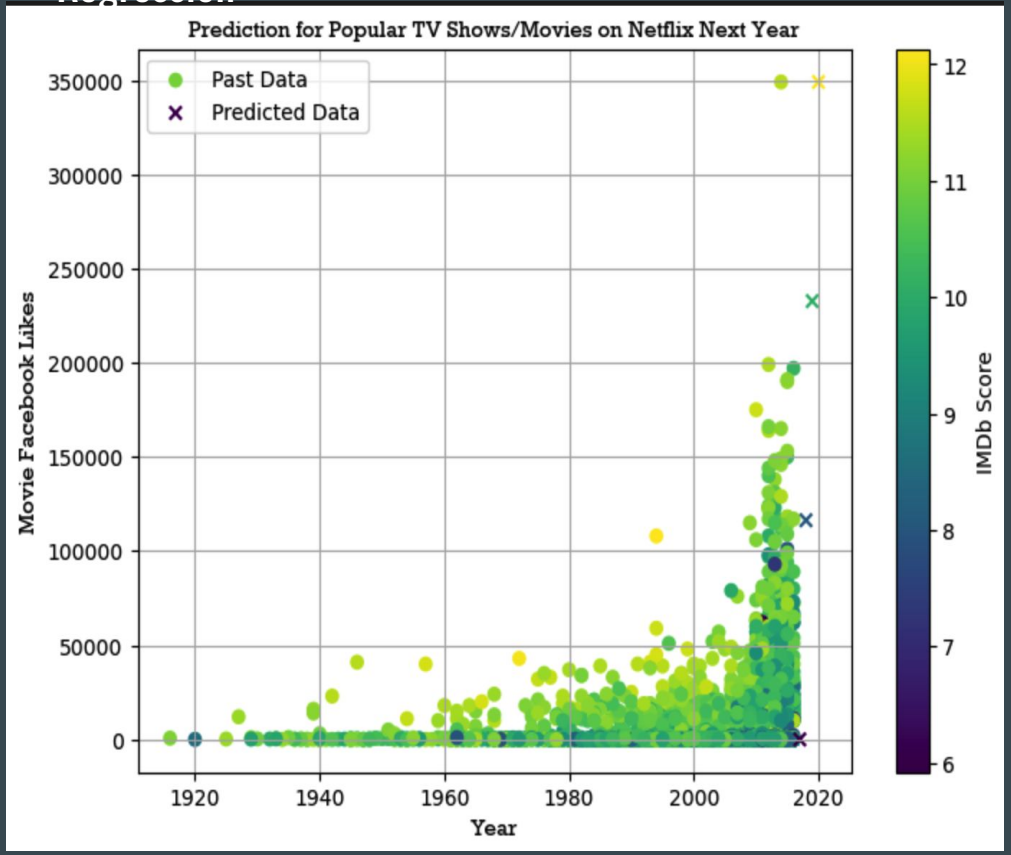
Tableau Link:
[Movies Tableau - Project 4]





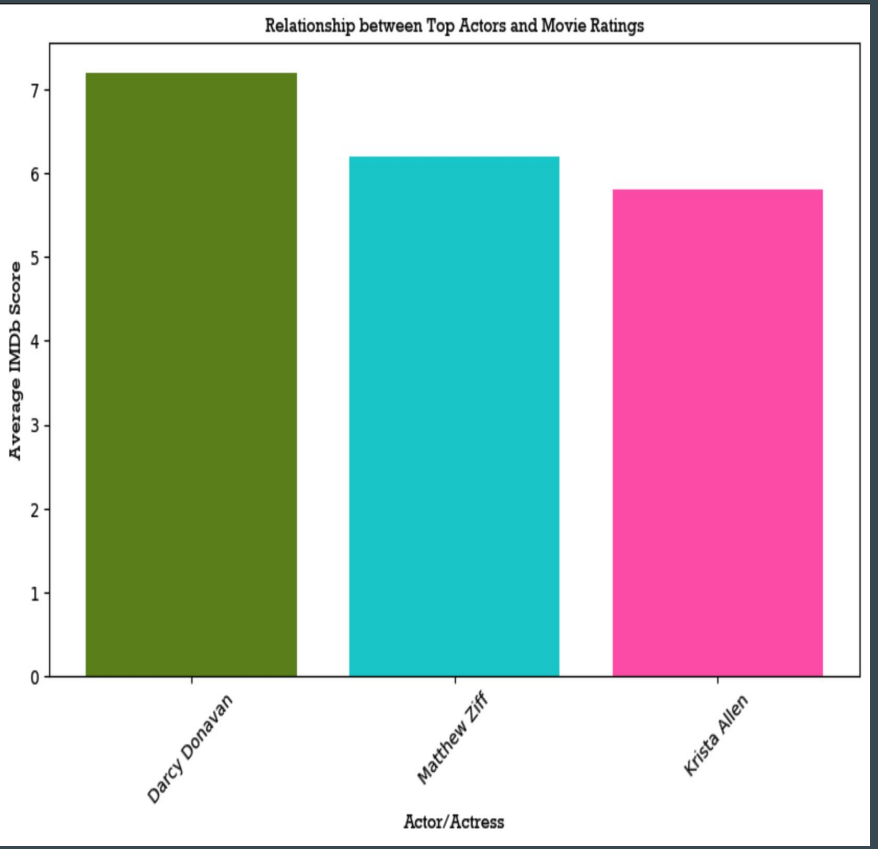


Predicting IMDB Scores with Linear Regression

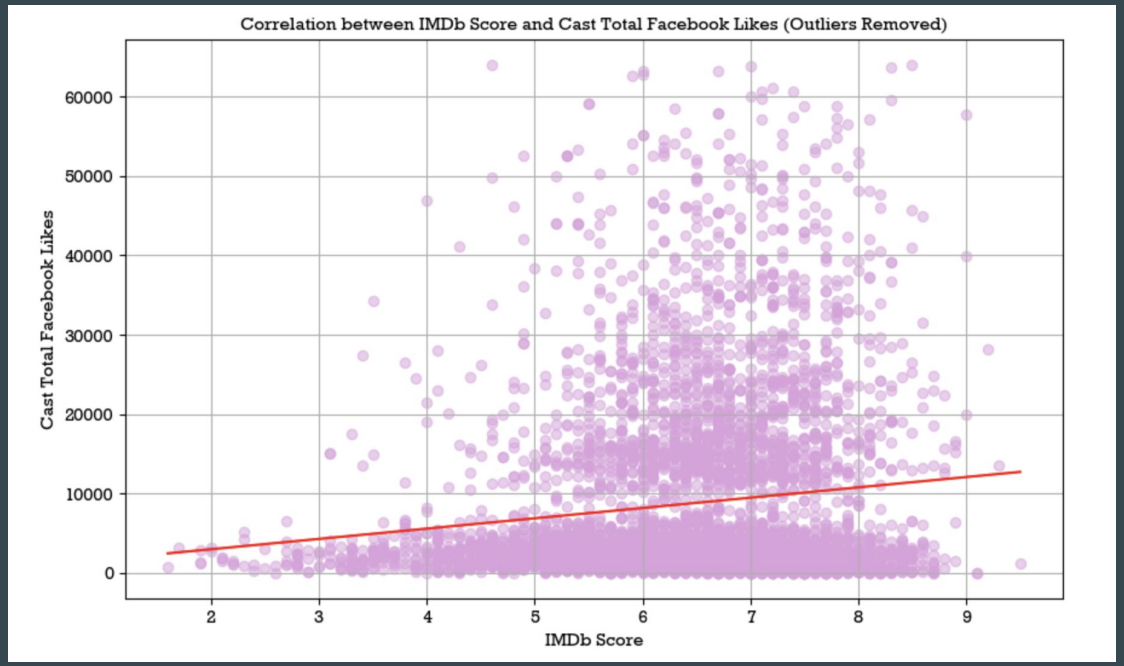


Utilizing a linear regression model, We trained the existing data to predict IMDB scores for upcoming movies. This predictive model serves as a tool to anticipate the potential ratings of movies yet to be released. We plotted a scatter plot with data points representing movies, where each point's position reflects the movie's release year and the number of Facebook likes it received. This scatter plot aims to provide insights into which upcoming movies or TV shows might gather popularity on Netflix based on historical trends.

Analyzing the Influence of Actors and Actresses on IMDB Scores



We generated a bar chart illustrating the average IMDb scores of movies featuring the top 5 actors or actresses with the highest number of facebook likes. This bar chart offers insights into the impact of popular actors/actresses on the overall reception of movies they star in.



We used Python Matplotlib to create a scatterplot of the correlation between a movie's IMDB score and the total cast facebook likes.

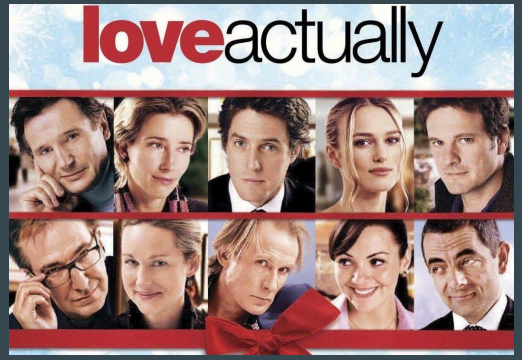
Outliers had to be removed to get a better visual, these mostly included blockbuster movies like Marvel movies, The Hunger Games, Inception, etc.

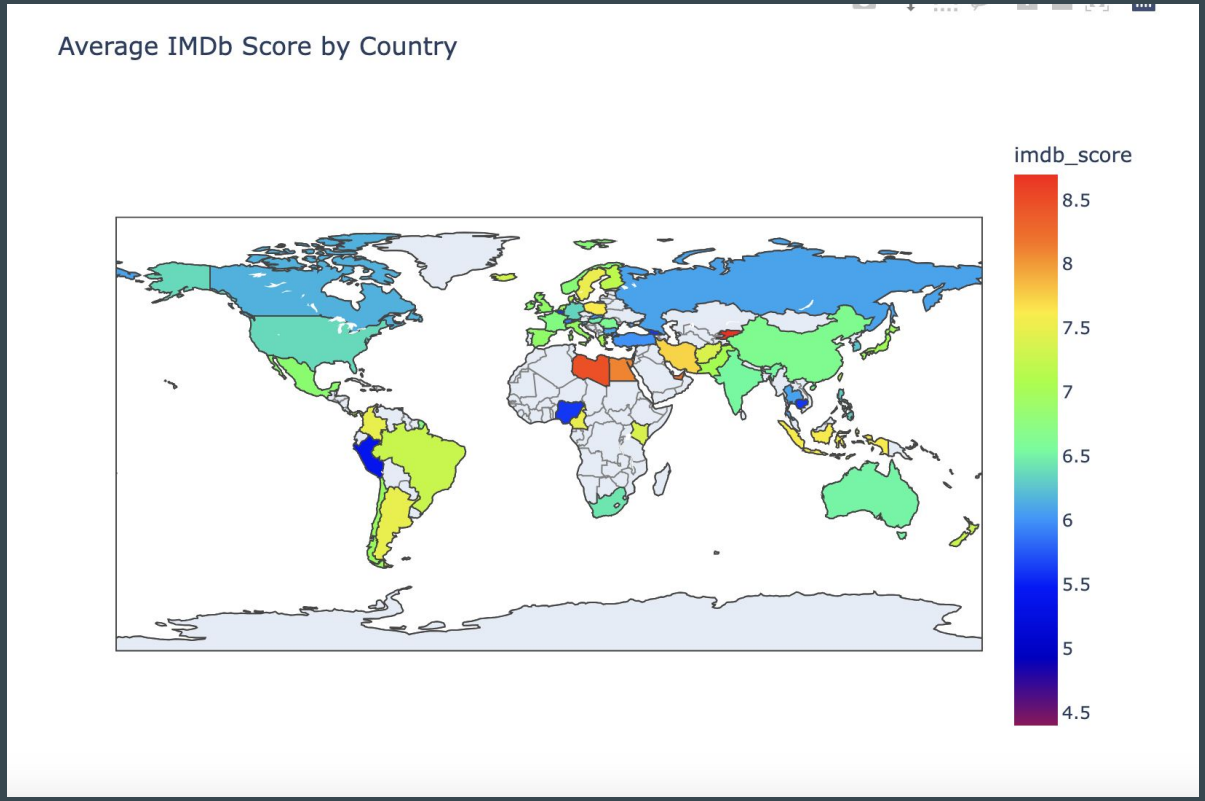
Line of linear regression shows a slight correlation

R-value: 0.9
R-value no outliers: 0.12

We created a map that shows the average IMDB score by the country where the movie was released.

When viewed in the python file, the map also has hover labels that show the name of the country and the average IMDB score.



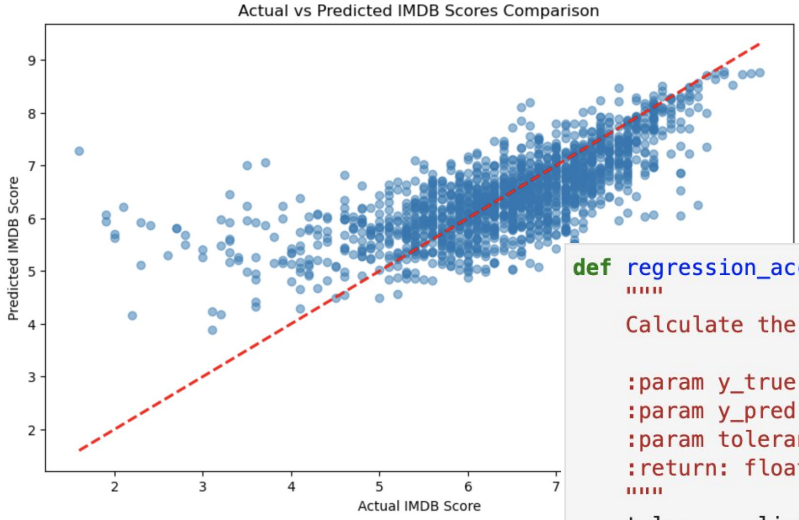


The Model 1

- Stacking Regressor - R-squared: 0.5081, Mean Squared Error: 0.6369
- Accuracy: 0.6748
- Accuracy_train 0.8620396600566572
- Overfitting
  - Null numbers
  - Empty cell
- Created a csv for predicted vs. actual


```python
# create_database_table()
# data = fetch_data()
# data = clean_preprocess_data(data)

X = data.drop(['movie_title', 'movie_imdb_link', 'director_name', 'actor_1_name', 'actor_2_name', 'actor_3_name', 'imdb_score'],
y = data['imdb_score']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Define base models
random_forest = RandomForestRegressor(random_state=42)
gradient_boosting = GradientBoostingRegressor(random_state=42)
xgb = XGBRegressor(random_state=42)

# Hyperparameter tuning
param_grid_rf = {'n_estimators': [100, 500], 'max_depth': [10, 20], 'min_samples_split': [2, 5]}
param_grid_gb = {'n_estimators': [100, 300], 'learning_rate': [0.05, 0.1], 'max_depth': [3, 5]}
param_grid_xgb = {'n_estimators': [100, 300], 'learning_rate': [0.05, 0.1], 'max_depth': [3, 5]}

grid_search_rf = GridSearchCV(random_forest, param_grid_rf, cv=5, scoring='r2')
grid_search_gb = GridSearchCV(gradient_boosting, param_grid_gb, cv=5, scoring='r2')
grid_search_xgb = GridSearchCV(xgb, param_grid_xgb, cv=5, scoring='r2')

best_rf = grid_search_rf.fit(X_train, y_train).best_estimator_
best_gb = grid_search_gb.fit(X_train, y_train).best_estimator_
best_xgb = grid_search_xgb.fit(X_train, y_train).best_estimator_

# Stacking regressor
stacking_regressor = StackingRegressor(
    estimators=[('rf', best_rf), ('gb', best_gb), ('xgb', best_xgb)],
    final_estimator=Ridge()
)
stacking_regressor.fit(X_train, y_train)
y_pred_stacking = stacking_regressor.predict(X_test)
r2_stacking = r2_score(y_test, y_pred_stacking)
mse_stacking = mean_squared_error(y_test, y_pred_stacking)

print(f"Stacking Regressor - R-squared: {r2_stacking:.4f}, Mean Squared Error: {mse_stacking:.4f}")
joblib.dump(stacking_regressor, 'best_model.pkl')
print("Best model saved successfully.")
```

	Actual IMDB Score	Predicted IMDB Score	Error
4943	7.2	5.936544	1.263456
1919	7.0	6.709759	0.290241
1049	6.4	6.682310	-0.282310
4697	6.5	5.932678	0.567322
3312	6.1	5.339052	0.760948




```python
def regression_accuracy(y_true, y_pred, tolerance=0.1):
    """
    Calculate the percentage of predictions within a certain tolerance of the actual values.

    :param y_true: array-like, true target values
    :param y_pred: array-like, predicted values
    :param tolerance: float, the percentage tolerance for considering predictions as accurate
    :return: float, the accuracy of predictions within the specified tolerance
    """
    tolerance_limit = tolerance * np.abs(y_true)
    return np.mean(np.abs(y_true - y_pred) <= tolerance_limit)

accuracy = regression_accuracy(y_test, y_pred_stacking, tolerance=0.1)
print(f"Accuracy: {accuracy:.4f}")
```

```
Accuracy: 0.6748
```

```python
y_pred_train = model.predict(X_train)
```

```python
accuracy_train = regression_accuracy(y_train, y_pred_train, tolerance=0.1)
```

```python
print("Accuracy_train", accuracy_train)
```

```
Accuracy_train 0.8620396600566572
```

The Model 2

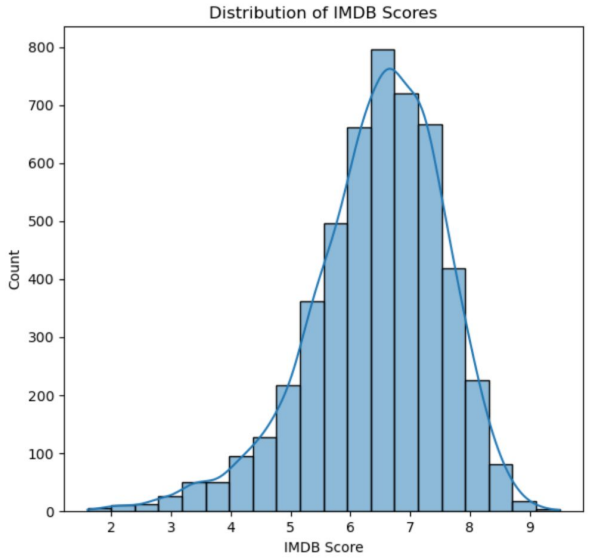
```
best_params
```

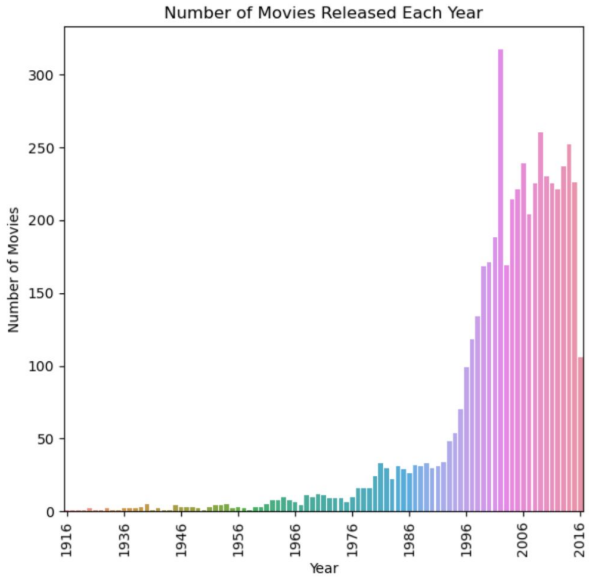
```
{'max_depth': 5, 'n_estimators': 200, 'random_state': 42}
```

- Randomforest
- Test and Train model
- Best parameters
- Interactive model for prediction

```
Classification Report for Training Data:
              precision    recall  f1-score   support

       False       0.70      0.75      0.72      1780
        True       0.78      0.73      0.75      2112

    accuracy                           0.74      3892
   macro avg       0.74      0.74      0.74      3892
weighted avg       0.74      0.74      0.74      3892


Classification Report for Testing Data:
              precision    recall  f1-score   support

       False       0.66      0.66      0.66       479
        True       0.67      0.67      0.67       495

    accuracy                           0.66       974
   macro avg       0.66      0.66      0.66       974
weighted avg       0.66      0.66      0.66       974
```

```python
# Define the parameter grid for Random Forest
param_grid = {
    'n_estimators': [100, 200, 500],
    'max_depth': [None, 5, 10, 15],
    'random_state': [42]
}

# Initialize a RandomForestClassifier
rf_model = RandomForestClassifier()

# Define a custom scoring function (lower is better)
def custom_accuracy(y_true, y_pred):
    return 1 - accuracy_score(y_true, y_pred)

# Initialize a GridSearchCV object with custom scoring
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid,
                           cv=10, n_jobs=-1, scoring=make_scorer(custom_accuracy))

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

# Train a new RandomForestClassifier with the best parameters
rf_model_best = RandomForestClassifier(**best_params)
rf_model_best.fit(X_train, y_train)

# Make predictions on the training and testing data
y_train_pred = rf_model_best.predict(X_train)
y_test_pred = rf_model_best.predict(X_test)

# Calculate classification report for training data
print("Classification Report for Training Data:")
print(classification_report(y_train, y_train_pred))

# Calculate classification report for testing data
print("\nClassification Report for Testing Data:")
print(classification_report(y_test, y_test_pred))
```






```
# To start the user interface:
user_interface()
```

```
Enter the title of the movie:  spectre
Title: spectre - Likely to be successful
Director: Sam Mendes
Main Actor: Christoph Waltz
IMDB Score: 6.8
Number of Critic Reviews: 602.0
```

Summary

- Real world data could cause overfitting of the results
- Difficult to find the right parameters/features
- Need to alter data to get better training and test accuracy scores.

•••

Thanks!
Q&A
...