# Recommendation Engine Netflix

The Netflix movie recommendation engine project stands as a noteworthy example of how machine learning, a core component of data science, extends its impact beyond entertainment into the telecom domain. By tailoring content suggestions based on individual preferences, viewing history, and behavior, the project significantly elevates user experience. This personalized approach holds considerable promise for telecom industries, where analogous algorithms can be leveraged to recommend customized services and content aligned with users' communication patterns and preferences, thereby enriching their overall interaction with telecom services.

Furthermore, the recommendation engine's pivotal role in enhancing user engagement has implications for both the entertainment and telecom sectors. Netflix's ability to provide relevant and enticing suggestions keeps users actively involved in the platform, leading to prolonged usage periods. This engagement-centric strategy is transferable to the telecom domain, where personalized recommendations for communication plans or additional features can enhance user satisfaction, fostering extended subscriptions and bolstering customer loyalty.

In essence, the integration of machine learning and big data processing, as demonstrated by Netflix, provides valuable insights for telecom companies aiming to elevate user experiences and achieve sustained business success through the strategic application of data science methodologies.

## Business Problem:

The business problem for the Netflix Recommendation Engine project centered around the challenge of enhancing user engagement and satisfaction in a vast content library. With a plethora of movies and TV shows available, users faced difficulty in discovering content aligned with their preferences, leading to decision fatigue and potential disengagement. The implementation of a recommendation engine aimed to address this issue by leveraging machine learning algorithms to analyze user data, offering personalized content suggestions. This solution sought to significantly improve the user experience by providing tailored recommendations, ultimately increasing user satisfaction, engagement, and retention for Netflix in the competitive streaming industry.

## Business Requirements

The business requirements for the Netflix Recommendation Engine project involve the development of a robust recommendation system that employs advanced machine learning algorithms to analyze user data, including viewing history, preferences, and behavior. The system should generate personalized content suggestions for users, enhancing their content discovery experience on the platform.

The primary goal is to increase user engagement, satisfaction, and retention by providing tailored recommendations that align with individual tastes. Additionally, the recommendation engine should be seamlessly integrated into the user interface, supporting real-time suggestions and facilitating continuous improvement through a feedback loop based on user interactions. The success of the project is contingent on the system's ability to adapt to changing user preferences, thereby contributing to the overall business growth and dominance of Netflix in the streaming industry.

## Objectives

- Building AI Engine where users will get best choice movies /series as per their past experience on movies to reduce the search time.

- User Personalization:
  Tailor recommendations based on individual user preferences, viewing history, and ratings to enhance the overall user experience.

- Content Similarity:
  Develop algorithms to analyze the content's characteristics, such as genre, theme, and style, to suggest similar movies or shows that align with users' interests.

- Diversity in Recommendations:
  Ensure a diverse range of recommendations to introduce users to new genres and content, promoting exploration and engagement.

- Real-time Updates:
  Implement mechanisms for real-time updates of recommendations, taking into account recent user interactions and new content additions to the platform.

- Multi-Modal Data Integration:
  Utilize a combination of user behavior, demographic information, and explicit feedback to create a comprehensive model for accurate and dynamic recommendations.

- Scalability:
  Design the recommendation engine to handle a large user base and a vast content library efficiently, ensuring scalability and optimal performance.

- Exploration-Exploitation Balance:
  Strike a balance between recommending popular content to cater to user preferences and introducing less-known but potentially interesting content to encourage diversity.

- Adaptability:
  Incorporate machine learning models that can adapt to changing user preferences over time, providing personalized recommendations that evolve with the user's taste.

- Transparency and Explainability:
  Ensure transparency in the recommendation process by incorporating explainable AI techniques, helping users understand why certain recommendations are made.

## Solution Approach: ML – Recommendations

### 1. K-Nearest Neighbors (KNN):

Use Case: Collaborative filtering based on user-item interactions.

Implementation: Build a KNN model to find similar users or items based on their ratings and make recommendations.

### 2. Non-Negative Matrix Factorization (NMF):

Use Case: Matrix factorization technique for collaborative filtering.

Implementation: Decompose the user-item interaction matrix into low-rank matrices to identify latent factors.

### 3.Decision Trees Random Forest:

Use Case: Incorporating content-based or hybrid recommendation approaches.

Implementation: Create decision tree models or ensemble methods like Random Forest using movie features (genres, actors, directors) to make recommendations.

### 4. Random Forest:

Use Case: analyze user viewing patterns, enhance content suggestions, and improve personalized recommendations.

Implementation: Utilize the scikit-learn library in Python to build a Random Forest model that processes user behavior data, such as viewing history and preferences, to predict and recommend movies or shows tailored to individual user tastes.

### 4.Naive Bayes:

Use Case: Use for classification or probabilistic recommendation.

Implementation: Apply Naive Bayes to predict user preferences based on movie features or user behavior.

This architecture provides a structured approach to integrating various ML algorithms into a Netflix recommendation system, ensuring a diverse range of approaches for generating recommendations based on user preferences and movie features.

## Scope:

The scope of the Recommendation System project in the telecom domain is to enhance user experience by implementing advanced recommendation algorithms based on both user and item-based filtering methodologies. The system aims to personalize content recommendations, such as mobile plans, value-added services, and promotions, to cater to individual user preferences. Leveraging collaborative filtering, the system will analyze user behavior and preferences, recommending products and services that align with their usage patterns. Additionally, content-based filtering will be employed to suggest offerings based on the intrinsic characteristics of telecom services, ensuring a diverse range of recommendations. The project scope includes the development of a scalable and real-time recommendation engine to adapt to changing user preferences dynamically. Integration of machine learning models like matrix factorization and deep learning will be explored to enhance recommendation accuracy. The telecom recommendation system will undergo rigorous A/B testing to evaluate the effectiveness of different algorithms, ensuring continuous improvement and a seamless user experience. The project also involves incorporating explainable AI techniques to enhance user trust and transparency in the recommendation process. Ultimately, the system aims to not only optimize telecom service recommendations but also contribute to increased user satisfaction and engagement.

## Data Sources & Understanding :

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset .

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI

repository, etc.In this project we have used .csv,excel data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques. Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

## Data Preperation :

**Importing the libraries**

```python
# Import Necessary Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

✓ 2.2s

```python
# Reading the dataset
data = pd.read_excel(r'C:\Users\aa912\OneDrive\Desktop\Netflix Movie Recommendation System\Netflix Movie Recommendation System\Netflix Movie Recommendation System\Netflix Movie Recommenda
data.head(5)
```

✓ 6.2s Python

| | Title | Genre | Tags | Languages | Series or Movie | Hidden Gem Score | Country Availability | Runtime | Director | Writer | Actors | View Rating | IMDb Score | Rotten Tomatoes Score | Meta |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Lets Fight Ghost | Crime, Drama, Fantasy, Horror, Romance | Comedy Programmes,Romantic TV Comedies,Horror ... | Swedish, Spanish | Series | 4.3 | Thailand | < 30 minutes | Tomas Alfredson | John Ajvide Lindqvist | Lina Leandersson, Kåre Hedebrant, Per Ragnar, ... | R | 7.9 | 98.0 | |
| 1 | HOW TO BUILD A GIRL | Comedy | Dramas,Comedies,Films Based on Books,British | English | Movie | 7.0 | Canada | 1-2 hour | Coky Giedroyc | Caitlin Moran | Cleo, Paddy Considine, Beanie Feldstein, Dónal... | R | 5.8 | 79.0 | |
| 2 | The Con-Heartist | Comedy, Romance | Romantic Comedies,Comedies,Romantic Films,Thai... | Thai | Movie | 8.6 | Thailand | > 2 hrs | Mez Tharatorn | Pattaranad Bhiboonsawade, Mez Tharatorn, Thods... | Kathaleeya McIntosh, Nadech Kugimiya, Pimchano... | NaN | 7.4 | NaN | |
| 3 | Gleboka woda | Drama | TV Dramas,Polish TV Shows,Social Issue TV Dramas | Polish | Series | 8.7 | Poland | < 30 minutes | NaN | NaN | Katarzyna Maciag, Piotr Nowak, Marcin Dorocins... | NaN | 7.5 | NaN | |
| 4 | Only a Mother | Drama | Social Issue Dramas,Dramas,Movies Based on Boo... | Swedish | Movie | 8.3 | Lithuania,Poland,France,Italy,Spain,Greece,Bel... | 1-2 hour | Alf Sjöberg | Ivar Lo-Johansson | Hugo Björne, Eva Dahlbeck, Ulf Palme, Ragnar F... | NaN | 6.7 | NaN | |

```
ix Movie Recommendation System\Netflix Movie Recommendation System\Netflix Movie Recommendation System\Netflix Movie Recommendation System\Datasets\Netflix Dataset Latest 2021.xlsx')
```

✓ 6.2s Python

| Netflix Link | IMDb Link | Summary | IMDb Votes | Image | Poster | TMDb Trailer | Trailer Site |
|---|---|---|---|---|---|---|---|
| https://www.netflix.com/watch/81415947 | https://www.imdb.com/title/tt1139797 | A med student with a supernatural gift tries t... | 205926.0 | https://occ-0-4708-64.1.nflxso.net/dnm/api/v6/... | https://m.media-amazon.com/images/M/MV5BOWM4NT... | https://www.youtube.com/watch?v=LqB6XJix-dM | YouTube |
| https://www.netflix.com/watch/81041267 | https://www.imdb.com/title/tt4193072 | When nerdy Johanna moves to London, things get... | 2838.0 | https://occ-0-1081-999.1.nflxso.net/dnm/api/v6... | https://m.media-amazon.com/images/M/MV5BZGUyN2... | https://www.youtube.com/watch?v=elbcxPy4okQ | YouTube |
| https://www.netflix.com/watch/81306155 | https://www.imdb.com/title/tt13393728 | After her ex-boyfriend cons her out of a large... | 131.0 | https://occ-0-2188-64.1.nflxso.net/dnm/api/v6/... | https://m.media-amazon.com/images/M/MV5BODAzOG... | https://www.youtube.com/watch?v=md3CmFLGK6Y | YouTube |
| https://www.netflix.com/watch/81307527 | https://www.imdb.com/title/tt2300049 | A group of social welfare workers led by their... | 47.0 | https://occ-0-2508-2706.1.nflxso.net/dnm/api/v... | https://m.media-amazon.com/images/M/MV5BMTc0Nz... | https://www.youtube.com/watch?v=5kyF2vy63r0 | YouTube |
| https://www.netflix.com/watch/81382068 | https://www.imdb.com/title/tt0041155 | An unhappily married farm worker struggling to... | 88.0 | https://occ-0-2851-41.1.nflxso.net/dnm/api/v6/... | https://m.media-amazon.com/images/M/MV5BMjVmMz... | https://www.youtube.com/watch?v=H0itWKFwMpQ | YouTube |

## Displaying All the Features Of the Dataset

```
# Displaying all the features of the dataset
data.columns
```

✓ 0.0s

```
Index(['Title', 'Genre', 'Tags', 'Languages', 'Series or Movie',
       'Hidden Gem Score', 'Country Availability', 'Runtime', 'Director',
       'Writer', 'Actors', 'View Rating', 'IMDb Score',
       'Rotten Tomatoes Score', 'Metacritic Score', 'Awards Received',
       'Awards Nominated For', 'Boxoffice', 'Release Date',
       'Netflix Release Date', 'Production House', 'Netflix Link', 'IMDb Link',
       'Summary', 'IMDb Votes', 'Image', 'Poster', 'TMDb Trailer',
       'Trailer Site'],
      dtype='object')
```

```
# Removing the Unnecessary columns from the dataset.
Unneccessary_Columns = ['Runtime','Boxoffice','TMDb Trailer','Trailer Site','Netflix Release Date','Production House','Netflix Link','Image', 'Poster', 'Awards Nominated For','IMDb Link',
data.filtered_Columns = data.drop(columns=Unneccessary_Columns)
data.filtered_Columns.head(10)
```

✓ 0.0s Python

```
C:\Users\aa912\AppData\Local\Temp\ipykernel_4924\844286032.py:3: UserWarning: Pandas doesn't allow columns to be created via a new attribute name - see https://pandas.pydata.org/pandas-docs/s
  data.filtered_Columns = data.drop(columns=Unneccessary_Columns)
```

| | Title | Genre | Tags | Languages | Series or Movie | Hidden Gem Score | Country Availability | Director | Writer | Actors | IMDb Score | Rotten Tomatoes Score | Metacritic Score | Summ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Lets Fight Ghost | Crime, Drama, Fantasy, Horror, Romance | Comedy Programmes,Romantic TV Comedies,Horror ... | Swedish, Spanish | Series | 4.3 | Thailand | Tomas Alfredson | John Ajvide Lindqvist | Lina Leandersson, Kåre Hedebrant, Per Ragnar, ... | 7.9 | 98.0 | 82.0 | A student superna gift trie |
| 1 | HOW TO BUILD A GIRL | Comedy | Dramas,Comedies,Films Based on Books,British | English | Movie | 7.0 | Canada | Coky Giedroyc | Caitlin Moran | Cleo, Paddy Considine, Beanie Feldstein, Dónal... | 5.8 | 79.0 | 69.0 | When n Joh mov Lon things |
| 2 | The Con-Heartist | Comedy, Romance | Romantic Comedies,Comedies,Romantic Films,Thai... | Thai | Movie | 8.6 | Thailand | Mez Tharatorn | Pattaranad Bhiboonsawade, Mez Tharatorn, Thods... | Kathaleeya McIntosh, Nadech Kugimiya, Pimchano... | 7.4 | NaN | NaN | Afte boyfr cons her of a lar |
| 3 | Gleboka woda | Drama | TV Dramas,Polish TV Shows,Social Issue TV Dramas | Polish | Series | 8.7 | Poland | NaN | NaN | Katarzyna Maciag, Piotr Nowak, Marcin Dorocins... | 7.5 | NaN | NaN | A grou s we worker by th |

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | Only a Mother | Drama | Social Issue Dramas,Dramas,Movies Based on Boo... | Swedish | Movie | 8.3 | Lithuania,Poland,France,Italy,Spain,Greece,Bel... | Alf Sjöberg | Ivar Lo-Johansson | Hugo Björne, Eva Dahlbeck, Ulf Palme, Ragnar F... | 6.7 | NaN | NaN |
| 5 | Snowroller | Comedy | Sports Movies,Sports Comedies,Comedies,Swedish... | Swedish, English, German, Norwegian | Movie | 5.3 | Lithuania,Poland,France,Italy,Spain,Greece,Cze... | Lasse Åberg | Lasse Åberg, Bo Jonsson | Lasse Åberg, Cecilia Walton, Eva Millberg, Jon... | 6.6 | NaN | NaN |
| 6 | The Invisible | Crime, Drama, Fantasy, Mystery, Thriller | Thriller Movies,Movies Based on Books,Supernat... | English | Movie | 2.0 | Lithuania,Poland,France,Italy,Spain,Greece,Cze... | David S. Goyer | Mats Wahl, Mick Davis, Christine Roum | Marcia Gay Harden, Margarita Levieva, Chris Ma... | 6.2 | 20.0 | 36.0 |
| 7 | The Simple Minded Murderer | Drama | Social Issue Dramas,Dramas,Movies Based on Boo... | Scanian, Swedish | Movie | 7.8 | Lithuania,Poland,France,Italy,Spain,Greece,Cze... | Hans Alfredson | Hans Alfredson | Maria Johansson, Hans Alfredson, Stellan Skars... | 7.6 | 92.0 | NaN |
| 8 | To Kill a Child | Short, Drama | Dramas,Swedish Movies | Spanish | Movie | 8.8 | Lithuania,Poland,France,Italy,Spain,Greece,Cze... | José Esteban Alenda, César Esteban Alenda | Victoria Ruiz, José Esteban Alenda, César Este... | Cristina Marcos, Manolo Solo, Roger Príncep, R... | 7.7 | NaN | NaN |
| 9 | Joker | Crime, Drama, Thriller | Dark Comedies,Crime Comedies,Dramas,Comedies,C... | English | Movie | 3.5 | Lithuania,Poland,France,Italy,Spain,Greece,Bel... | Todd Phillips | Bob Kane, Jerry Robinson, Bill Finger, Todd Ph... | Joaquin Phoenix, Robert De Niro, Zazie Beetz, ... | 8.4 | 68.0 | 59.0 |

```python
# Checking the Actual Size of the Dataset
data.shape
```
✓ 0.0s

```
(9425, 29)
```

```python
# Assigning the filtered columns to the data
data = data.filtered_Columns
```
✓ 0.0s

```python
# Checking the size of the dataset, after removing the unnecessary columns from the dataset
data.shape
```
✓ 0.0s

```
(9425, 15)
```

```python
# Checking the dtypes info from the dataset
data.info()
```
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9425 entries, 0 to 9424
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Title                   9425 non-null   object 
 1   Genre                   9400 non-null   object 
 2   Tags                    9389 non-null   object 
 3   Languages               9255 non-null   object 
 4   Series or Movie         9425 non-null   object 
 5   Hidden Gem Score        9415 non-null   float64
 6   Country Availability    9414 non-null   object 
 7   Director                7120 non-null   object 
 8   Writer                  7615 non-null   object 
 9   Actors                  9314 non-null   object 
 10  IMDb Score              9417 non-null   float64
 11  Rotten Tomatoes Score   5445 non-null   float64
 12  Metacritic Score        4082 non-null   float64
 13  Summary                 9420 non-null   object 
 14  IMDb Votes              9415 non-null   float64
dtypes: float64(5), object(10)
memory usage: 1.1+ MB
```

```python
# Checking the Null Values
data.isnull().any()
```

✓ 0.0s

```
Title                    False
Genre                     True
Tags                      True
Languages                 True
Series or Movie          False
Hidden Gem Score          True
Country Availability      True
Director                  True
Writer                    True
Actors                    True
IMDb Score                True
Rotten Tomatoes Score     True
Metacritic Score          True
Summary                   True
IMDb Votes                True
dtype: bool
```

```python
# Checking the Count of Null Values
data.isnull().sum()
```

✓ 0.0s

```
Title                       0
Genre                      25
Tags                       36
Languages                 170
Series or Movie             0
Hidden Gem Score           10
Country Availability       11
Director                 2305
Writer                   1810
Actors                    111
IMDb Score                  8
Rotten Tomatoes Score    3980
Metacritic Score         5343
Summary                     5
IMDb Votes                 10
dtype: int64
```

```python
#Filling all Nan values with mode of categorical feature
data["Genre"].fillna(data["Genre"].mode()[0],inplace=True)
data["Tags"].fillna(data["Tags"].mode()[0],inplace=True)
data["Languages"].fillna(data["Languages"].mode()[0],inplace=True)
data["Director"].fillna(data["Director"].mode()[0],inplace=True)
data["Writer"].fillna(data["Writer"].mode()[0],inplace=True)
data["Actors"].fillna(data["Actors"].mode()[0],inplace=True)
data["Summary"].fillna(data["Summary"].mode()[0],inplace=True)
data["Country Availability"].fillna(data["Country Availability"].mode()[0],inplace=True)

#Filling all Nan values with median of Numerical feature
data["IMDb Score"].fillna(data["IMDb Score"].median(),inplace=True)
data["Metacritic Score"].fillna(data["Metacritic Score"].median(),inplace=True)
data["Rotten Tomatoes Score"].fillna(data["Rotten Tomatoes Score"].median(),inplace=True)
data["Hidden Gem Score"].fillna(data["Hidden Gem Score"].median(),inplace=True)
data["Hidden Gem Score"].fillna(data["Hidden Gem Score"].median(),inplace=True)
```

✓ 0.0s

```python
data.head(15)
```

✓ 0.0s

| | Title | Genre | Tags | Languages | Series or Movie | Hidden Gem Score | Country Availability | Director | Writer | Actors | IMDb Score | Rotten Tomatoes Score | Metacritic Score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Lets Fight Ghost | Crime, Drama, Fantasy, Horror, Romance | Comedy Programmes,Romantic TV Comedies,Horror ... | Swedish, Spanish | Series | 4.3 | Thailand | Tomas Alfredson | John Ajvide Lindqvist | Lina Leandersson, Kåre Hedebrant, Per Ragnar, ... | 7.9 | 98.0 | 82.0 |
| 1 | HOW TO BUILD A GIRL | Comedy | Dramas,Comedies,Films Based on Books,British | English | Movie | 7.0 | Canada | Coky Giedroyc | Caitlin Moran | Cleo, Paddy Considine, Beanie Feldstein, Dónal... | 5.8 | 79.0 | 69.0 |
| 2 | The Con-Heartist | Comedy, Romance | Romantic Comedies,Comedies,Romantic Films,Thai... | Thai | Movie | 8.6 | Thailand | Mez Tharatorn | Pattaranad Bhiboonsawade, Mez Tharatorn, Thods... | Kathaleeya McIntosh, Nadech Kugimiya, Pimchano... | 7.4 | 70.0 | 59.0 |
| 3 | Gleboka woda | Drama | TV Dramas,Polish TV Shows,Social Issue TV Dramas | Polish | Series | 8.7 | Poland | Steven Spielberg | Fujio F. Fujiko | Katarzyna Maciag, Piotr Nowak, Marcin Dorocins... | 7.5 | 70.0 | 59.0 |
| 4 | Only a Mother | Drama | Social Issue Dramas,Dramas,Movies Based on Boo... | Swedish | Movie | 8.3 | Lithuania,Poland,France,Italy,Spain,Greece,Bel... | Alf Sjöberg | Ivar Lo-Johansson | Hugo Björne, Eva Dahlbeck, Ulf Palme, Ragnar F... | 6.7 | 70.0 | 59.0 |
| 5 | Snowroller | Comedy | Sports Movies,Sports Comedies,Comedies,Swedish... | Swedish, English, German, Norwegian | Movie | 5.3 | Lithuania,Poland,France,Italy,Spain,Greece,Cze... | Lasse Åberg | Lasse Åberg, Bo Jonsson | Lasse Åberg, Cecilia Walton, Eva Millberg, | 6.6 | 70.0 | 59.0 |

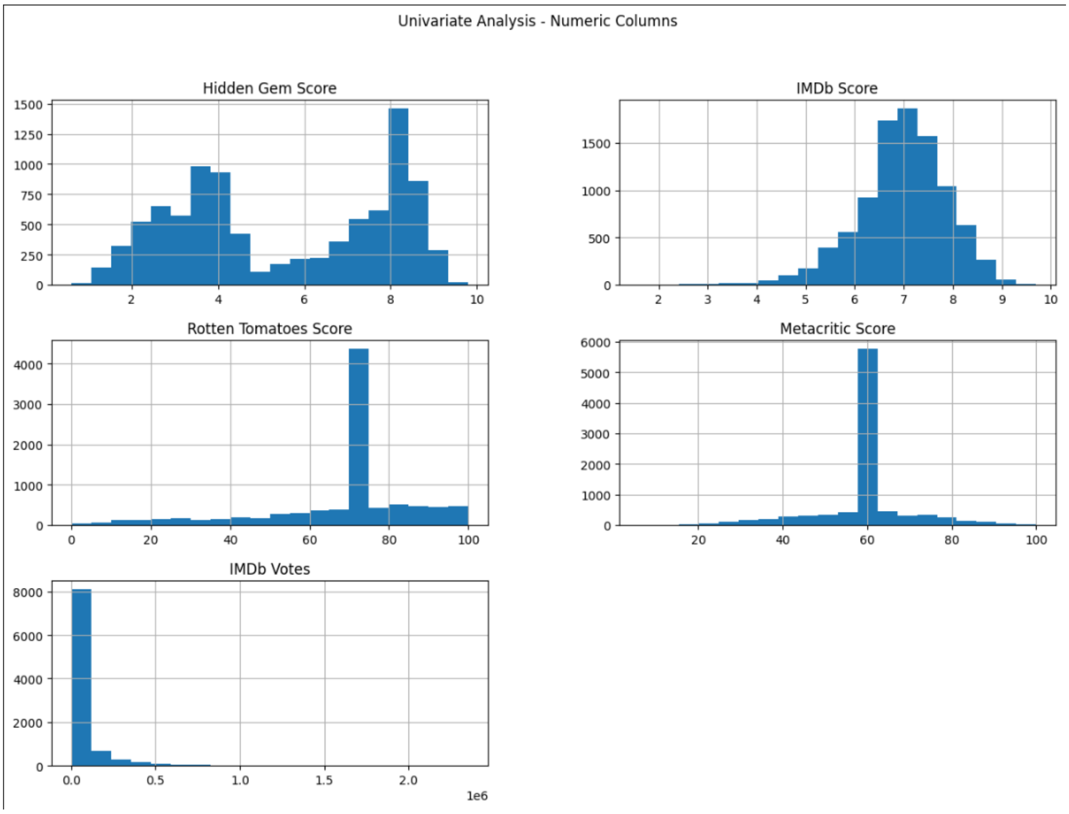
| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9 | Joker | Crime, Drama, Thriller | Dark Comedies,Crime Comedies,Dramas,Comedies,C... | English | Movie | 3.5 | Lithuania,Poland,France,Italy,Spain,Greece,Bel... | Todd Phillips | Bob Kane, Jerry Robinson, Bill Finger, Todd Ph... | Joaquin Phoenix, Robert De Niro, Zazie Beetz, ... | 8.4 | 68.0 | 59.0 |
| 10 | I | Action, Adventure, Fantasy, Sci-Fi | Dramas,Swedish Movies | English, Sanskrit | Movie | 2.8 | Lithuania,Poland,France,Italy,Spain,Greece,Cze... | George Lucas | George Lucas | Ewan McGregor, Natalie Portman, Jake Lloyd, Li... | 6.5 | 52.0 | 51.0 |
| 11 | Harrys Daughters | Adventure, Drama, Fantasy, Mystery | Dramas,Swedish Movies | English | Movie | 4.4 | Lithuania,Poland,France,Italy,Spain,Greece,Cze... | David Yates | Steve Kloves, J.K. Rowling | Daniel Radcliffe, Ralph Fiennes, Alan Rickman,... | 8.1 | 96.0 | 85.0 |
| 12 | Gyllene Tider | Music | Music & Musicals,Swedish Movies,Music & Concer... | Swedish | Movie | 8.8 | Lithuania,Poland,France,Italy,Spain,Greece,Cze... | Lasse Hallström | Fujio F. Fujiko | Anders Herrlin, Per Gessle, Micke Andersson, G... | 7.7 | 70.0 | 59.0 |
| 13 | Girls und Panzer das Finale | Animation, Action, Comedy | Drama Anime,Action & Adventure,Action Anime,An... | Japanese | Series | 8.5 | Japan | Tsutomu Mizushima | Reiko Yoshida | Ikumi Nakagami, Mai Fuchigami, Mami Ozaki, Ai ... | 7.3 | 70.0 | 59.0 |
| 14 | The Coroner | Crime, Drama | Mystery Programmes,Drama Programmes,Crime TV D... | English | Series | 7.8 | Canada | Steven Spielberg | Sally Abbott | Matt Bardock, Oliver Gomm, Claire Goose, Beati... | 7.0 | 70.0 | 59.0 |

## Data Visualization:

# Data Visualization

Univariate Analysis:

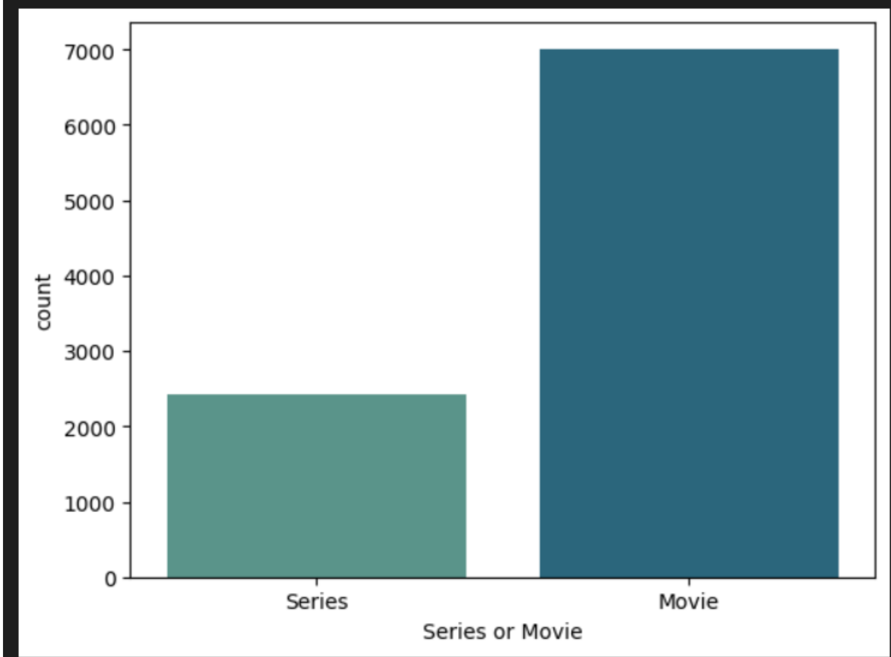
```
import matplotlib.pyplot as plt
numeric_columns = ['Hidden Gem Score', 'IMDb Score', 'Rotten Tomatoes Score', 'Metacritic Score', 'IMDb Votes']
data[numeric_columns].hist(bins=20, figsize=(15, 10))
plt.suptitle('Univariate Analysis - Numeric Columns')
plt.show()
```




```python
sns.countplot(x ='Series or Movie', data = data, palette="crest")
plt.show()
```

✓ 0.1s

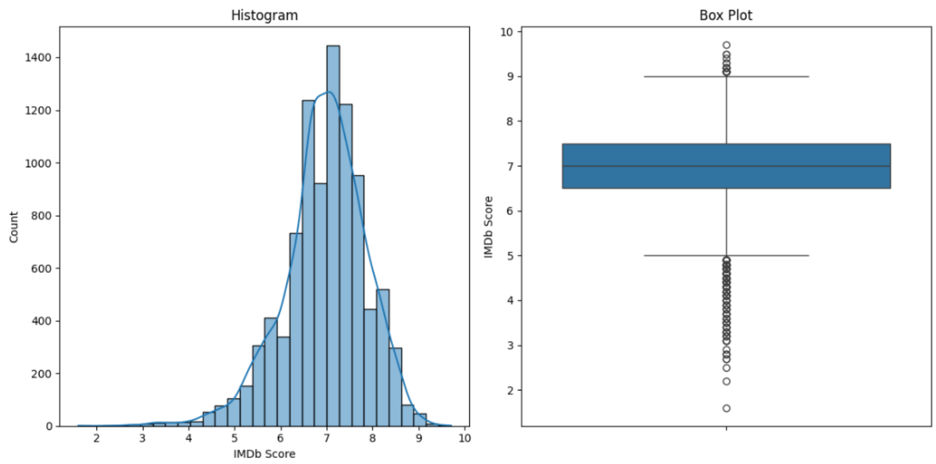
```
C:\Users\aa912\AppData\Local\Temp\ipykernel_4924\869491981.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x`

  sns.countplot(x ='Series or Movie', data = data, palette="crest")
```



Bivariate Analysis:

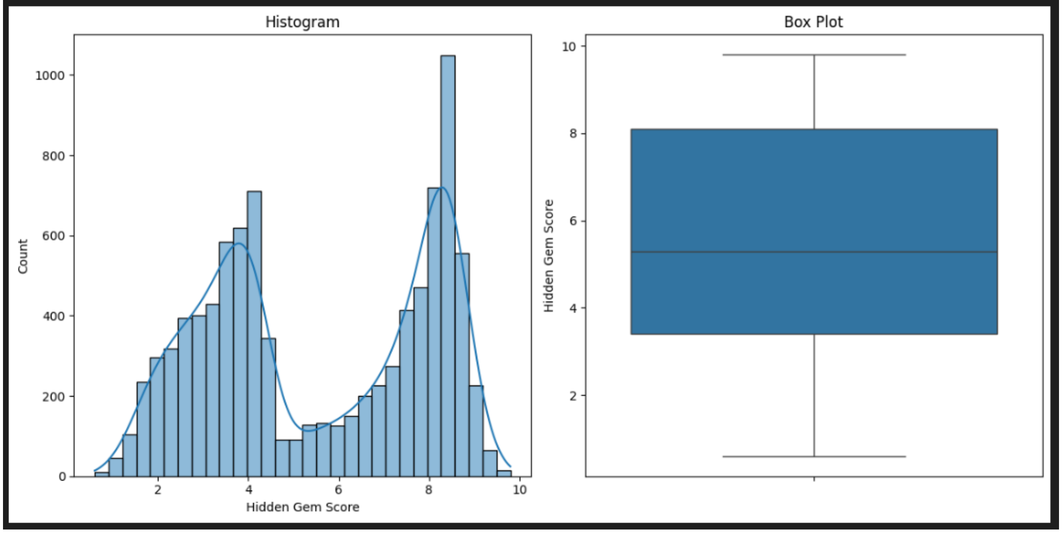
```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.histplot(data['IMDb Score'], kde=True, bins=30)
plt.title('Histogram')

plt.subplot(1, 2, 2)
sns.boxplot(data=data, y='IMDb Score')
plt.title('Box Plot')

plt.tight_layout()
plt.show()
```

[19] ✓ 0.3s



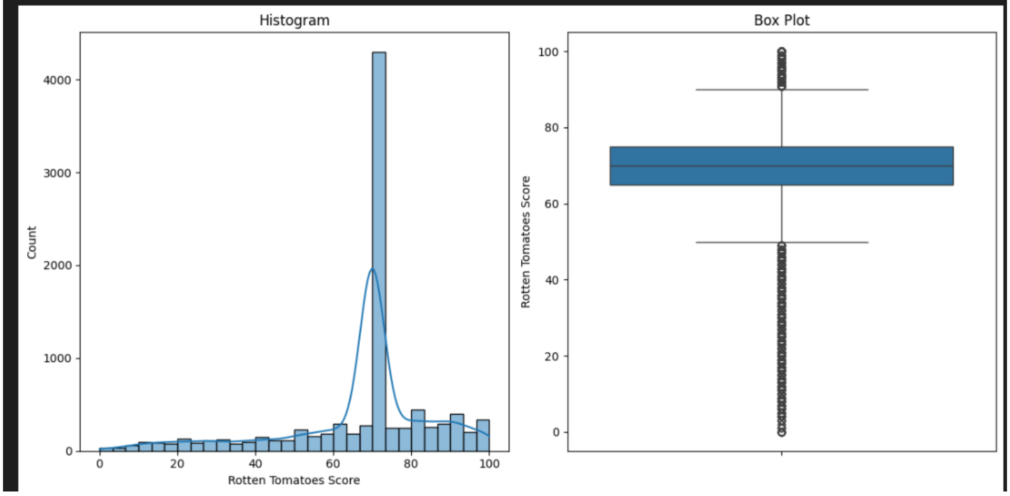

```
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.histplot(data['Hidden Gem Score'], kde=True, bins=30)
plt.title('Histogram')

plt.subplot(1, 2, 2)
sns.boxplot(data=data, y='Hidden Gem Score')
plt.title('Box Plot')

plt.tight_layout()
plt.show()
```

✓ 0.3s Python



```python
plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
sns.histplot(data['Rotten Tomatoes Score'], kde=True, bins=30)
plt.title('Histogram')

plt.subplot(1, 2, 2)
sns.boxplot(data=data, y='Rotten Tomatoes Score')
plt.title('Box Plot')

plt.tight_layout()
plt.show()
```

✓ 0.4s Python

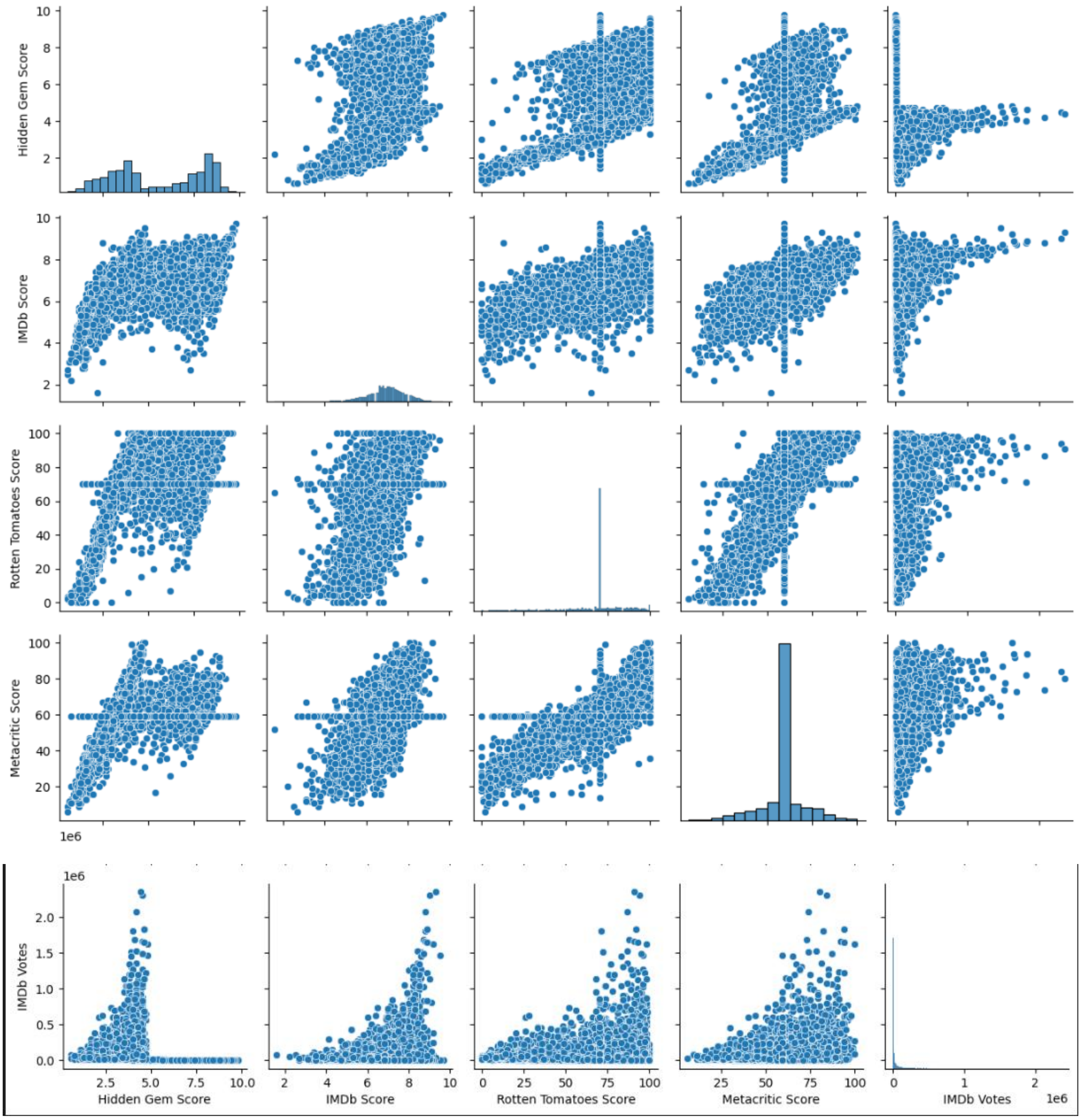



Multi Variate Analysis

```python
sns.pairplot(data)
```

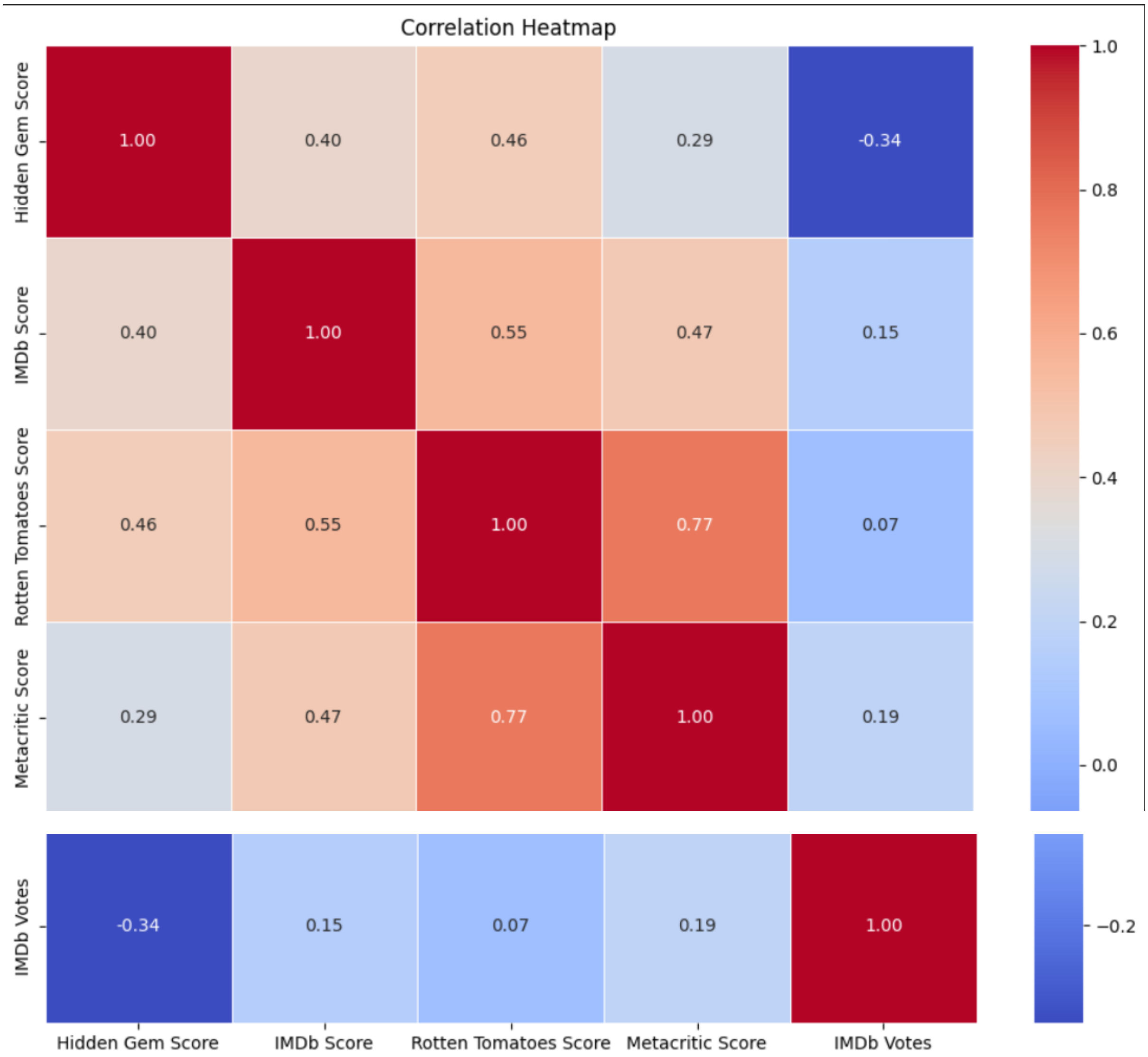
✓ 5.9s

```
<seaborn.axisgrid.PairGrid at 0x1cfa4f57a10>
```




```
correlation_matrix=data.corr(numeric_only=True)
```
✓ 0.0s

```
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```
✓ 0.2s


Correlation Heatmap
Hidden Gem Score
IMDb Score
Rotten Tomatoes Score
Metacritic Score
IMDb Votes
1.00
0.40
0.46
0.29
-0.34
0.40
1.00
0.55
0.47
0.15
0.46
0.55
1.00
0.77
0.07
0.29
0.47
0.77
1.00
0.19
-0.34
0.15
0.07
0.19
1.00
Hidden Gem Score
IMDb Score
Rotten Tomatoes Score
Metacritic Score
IMDb Votes
1.0
0.8
0.6
0.4
0.2
0.0
−0.2

# Statistical Analysis

```python
data.describe()
```

[25] ✓ 0.0s Python

...

| | Hidden Gem Score | IMDb Score | Rotten Tomatoes Score | Metacritic Score | IMDb Votes |
|---|---|---|---|---|---|
| count | 9425.000000 | 9425.000000 | 9425.000000 | 9425.000000 | 9.415000e+03 |
| mean | 5.540477 | 6.955554 | 66.933050 | 58.616021 | 6.014725e+04 |
| std | 2.446176 | 0.899300 | 19.384208 | 11.289801 | 1.463837e+05 |
| min | 0.600000 | 1.600000 | 0.000000 | 6.000000 | 5.000000e+00 |
| 25% | 3.400000 | 6.500000 | 65.000000 | 59.000000 | 9.695000e+02 |
| 50% | 5.300000 | 7.000000 | 70.000000 | 59.000000 | 6.602000e+03 |
| 75% | 8.100000 | 7.500000 | 75.000000 | 59.000000 | 5.098700e+04 |
| max | 9.800000 | 9.700000 | 100.000000 | 100.000000 | 2.354197e+06 |

```python
# Combining the tags column and Summary column as target column
data['Tags__Summary']= data['Tags'] + data['Summary']
```

[26] ✓ 0.0s Python

```python
# Taking only necessary data
data = data[['Title','Genre','Languages','Series or Movie','Actors','IMDb Score','Tags__Summary']]
data
```

[27] ✓ 0.0s Python


```python
# Taking only necessary data
data = data[['Title','Genre','Languages','Series or Movie','Actors','IMDb Score','Tags__Summary']]
data
```

✓ 0.0s Python


| | Title | Genre | Languages | Series or Movie | Actors | IMDb Score | Tags__Summary |
|---|---|---|---|---|---|---|---|
| 0 | Lets Fight Ghost | Crime, Drama, Fantasy, Horror, Romance | Swedish, Spanish | Series | Lina Leandersson, Kåre Hedebrant, Per Ragnar, ... | 7.9 | Comedy Programmes,Romantic TV Comedies,Horror ... |
| 1 | HOW TO BUILD A GIRL | Comedy | English | Movie | Cleo, Paddy Considine, Beanie Feldstein, Dónal... | 5.8 | Dramas,Comedies,Films Based on Books,BritishWh... |
| 2 | The Con-Heartist | Comedy, Romance | Thai | Movie | Kathaleeya McIntosh, Nadech Kugimiya, Pimchano... | 7.4 | Romantic Comedies,Comedies,Romantic Films,Thai... |
| 3 | Gleboka woda | Drama | Polish | Series | Katarzyna Maciag, Piotr Nowak, Marcin Dorocins... | 7.5 | TV Dramas,Polish TV Shows,Social Issue TV Dram... |
| 4 | Only a Mother | Drama | Swedish | Movie | Hugo Björne, Eva Dahlbeck, Ulf Palme, Ragnar F... | 6.7 | Social Issue Dramas,Dramas,Movies Based on Boo... |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 9420 | 13 Going on 30 | Comedy, Fantasy, Romance | English, Portuguese | Movie | Andy Serkis, Jennifer Garner, Mark Ruffalo, Ju... | 6.2 | Romantic Comedies,Comedies,Romantic Films,Roma... |
| 9421 | LIFE 2.0 | Documentary | English | Movie | Teasa Copprue | 6.2 | Social & Cultural Documentaries,Biographical D... |
| 9422 | Brand New Day | Documentary, Music | English | Movie | Ryuichi Sakamoto, Clem Burke, Annie Lennox, Pa... | 7.3 | Australian Comedies,Romantic Comedies,Australi... |
| 9423 | Daniel Arends: Blessuretijd | Comedy | Dutch | Movie | Daniël Arends | 7.8 | Stand-up Comedy,International Movies,ComediesI... |
| 9424 | DreamWorks Happy Holidays from Madagascar | Animation, Comedy, Family | English | Series | Jung Hyun Kim | 6.8 | TV Comedies,Kids TV,Animal Tales,TV Cartoons,T... |

9425 rows × 7 columns

## Model Selection and Model Building

```
from sklearn.feature_extraction.text import CountVectorizer
```
✓ 0.2s

```
cv = CountVectorizer(max_features=9425,stop_words='english')
```
✓ 0.0s

```
cv
```
✓ 0.0s

```
CountVectorizer
CountVectorizer(max_features=9425, stop_words='english')
```

```
vector = cv.fit_transform(data['Tags__Summary'].values.astype('U')).toarray()
```
✓ 0.5s

```
vector.shape
```
✓ 0.0s

```
(9425, 9425)
```

## Cosine Similarity

```
from sklearn.metrics.pairwise import cosine_similarity
similarity = cosine_similarity(vector)
def recommend(movie):
    # Find the index of the movie in the dataset
    index = data[data['Title'] == movie].index[0]

    # Calculate cosine similarity for the given movie
    distance = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda vector: vector[1])

    # Display recommended movies
    print(f"Recommendations for {movie}:")
    for i in distance[0:5]:
        print(data.iloc[i[0]]['Title'])

# Example usage
recommend("Handle Me With Care")
```
✓ 10.6s

```
Recommendations for Handle Me With Care:
Handle Me With Care
Brother Of The Year
A Gift
Tortilla Soup
The Teachers Diary
```

# K Nearest Neighbours

```python
from sklearn.neighbors import NearestNeighbors

def recommend(movies, model, data, similarity):
    # Find the index of the movie in the dataset
    index = data[data['Title'] == movies].index[0]

    # Get the indices and distances of the most similar movies using the KNN model
    distances, indices = model.kneighbors([similarity[index]])

    # Display recommended movies
    print(f"Recommendations for {movies}:")
    for i in indices[0][1:6]:  # Exclude the movie itself (first element)
        print(data.iloc[i]['Title'])

# Use KNN model for recommendations
knn_model = NearestNeighbors(n_neighbors=6, metric='cosine')
knn_model.fit(similarity)

# Example usage
recommend('Brand New Day', knn_model, data, similarity)
```

✓ 1.1s

```
Recommendations for Brand New Day:
The Con-Heartist
The Invention of Lying
Intolerable Cruelty
Shes Out of My League
40 Days and 40 Nights
```

## TF-IDF (Term Frequency-Inverse Document Frequency)

```python
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(data['Tags__Summary'].fillna(''))

# Use Nearest Neighbors with cosine similarity for content-based recommendations
knn_model_content = NearestNeighbors(n_neighbors=6, metric='cosine')
knn_model_content.fit(tfidf_matrix)

def recommend_content_based(movie, model, data, vectorizer):
    # Find the index of the movie in the dataset
    index = data[data['Title'] == movie].index[0]

    # Transform the movie description using the TF-IDF vectorizer
    movie_tfidf = vectorizer.transform([data.iloc[index]['Tags__Summary']])

    # Get the indices and distances of the most similar movies using the KNN model
    distances, indices = model.kneighbors(movie_tfidf)

    # Display recommended movies
    print(f"Content-Based Recommendations for {movie}:")
    for i in indices[0][1:6]:  # Exclude the movie itself (first element)
        print(data.iloc[i]['Title'])

# Example usage
recommend_content_based('Brand New Day', knn_model_content, data, tfidf_vectorizer)
```

✓ 0.3s

```
Content-Based Recommendations for Brand New Day:
Love, Rosie
Top End Wedding
200 Pounds Beauty
Easy A
Tanu Weds Manu
```

# Non-Negative Matrix Factorization (NMF)

```python
from sklearn.decomposition import NMF
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(data['Tags__Summary'].fillna(''))

nmf_model = NMF(n_components=4, random_state=42)  # You can adjust the number of components
nmf_matrix = nmf_model.fit_transform(tfidf_matrix)

# Function to get movie recommendations based on NMF
def recommend_movies_nmf(movie_title, model, data, vectorizer, num_recommendations=5):
    # Find the index of the movie in the dataset
    idx = data[data['Title'] == movie_title].index[0]

    # Transform the movie description using the TF-IDF vectorizer
    movie_tfidf = vectorizer.transform([data.iloc[idx]['Tags__Summary']])

    # Project the movie onto the latent feature space using NMF
    movie_representation = model.transform(movie_tfidf)

    # Calculate the cosine similarity between the movie and all other movies in the latent space
    similarities = nmf_matrix.dot(movie_representation.T)

    # Get the indices of the most similar movies
    movie_indices = similarities.argsort(axis=0)[:-num_recommendations-1:-1]

    # Return the top 5 most similar movies
    return data['Title'].iloc[movie_indices.flatten()]

movie_to_recommend_nmf = 'Brand New Day'
recommended_movies_nmf = recommend_movies_nmf(movie_to_recommend_nmf, nmf_model, data, tfidf_vectorizer)
print(f"Recommendations for {movie_to_recommend_nmf}:")
for title in recommended_movies_nmf:
    print(title)
```

```
Recommendations for Brand New Day:
Tanu Weds Manu
Bareilly Ki Barfi
Howards End
Out of Africa
Silver Linings Playbook
```

# Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dt_model_content = DecisionTreeClassifier(criterion='entropy', max_depth=5)
dt_model_content.fit(tfidf_matrix, data['Genre'].fillna(''))

def recommend_content_based_dt(movie, model, data, vectorizer):
    # Find the index of the movie in the dataset
    index = data[data['Title'] == movie].index[0]

    # Transform the movie description using the TF-IDF vectorizer
    movie_tfidf = vectorizer.transform([data.iloc[index]['Tags__Summary']])

    # Predict the genre of the movie using the decision tree model
    predicted_genre = model.predict(movie_tfidf)[0]

    # Display recommended movies based on the predicted genre
    print(f"Content-Based Recommendations for {movie} (Decision Tree):")
    recommended_movies = data[data['Genre'] == predicted_genre]
    for i in range(min(5, len(recommended_movies))):
        print(recommended_movies.iloc[i]['Title'])

# Example usage
recommend_content_based_dt('Brand New Day', dt_model_content, data, tfidf_vectorizer)
```

✓ 18.1s

```
Content-Based Recommendations for Brand New Day (Decision Tree):
The Con-Heartist
Alice
Erotikon
The House Arrest of Us
So Im a Spider, So What?
```

# Random Forest

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(data['Tags__Summary'].fillna(''))

rf_model_content = RandomForestClassifier(n_estimators=10, max_depth=5)
rf_model_content.fit(tfidf_matrix, data['Genre'].fillna(''))

def recommend_content_based_rf(movie, model, data, vectorizer):
    # Find the index of the movie in the dataset
    index = data[data['Title'] == movie].index[0]

    # Transform the movie description using the TF-IDF vectorizer
    movie_tfidf = vectorizer.transform([data.iloc[index]['Tags__Summary']])

    # Predict the genre of the movie using the Random Forest model
    predicted_genre = model.predict(movie_tfidf)[0]

    # Display recommended movies based on the predicted genre
    print(f"Content-Based Recommendations for {movie} (Random Forest):")
    recommended_movies = data[data['Genre'] == predicted_genre]
    for i in range(min(5, len(recommended_movies))):
        print(recommended_movies.iloc[i]['Title'])

# Example usage
recommend_content_based_rf('Brand New Day', rf_model_content, data, tfidf_vectorizer)
```

✓ 0.6s

```
Content-Based Recommendations for Brand New Day (Random Forest):
Gleboka woda
Only a Mother
The Simple Minded Murderer
Repast
When a Woman Ascends the Stairs
```

# Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB

# Create a Naive Bayes classifier
nb_model_content = MultinomialNB()

# Fit the Naive Bayes classifier on the TF-IDF matrix and genre labels
nb_model_content.fit(tfidf_matrix, data['Genre'].fillna(''))

def recommend_content_based_nb(movie, model, data, vectorizer):
    # Find the index of the movie in the dataset
    index = data[data['Title'] == movie].index[0]

    # Transform the movie description using the TF-IDF vectorizer
    movie_tfidf = vectorizer.transform([data.iloc[index]['Tags__Summary']])

    # Predict the genre of the movie using the Naive Bayes classifier
    predicted_genre = model.predict(movie_tfidf)[0]

    # Display recommended movies based on the predicted genre
    print(f"Content-Based Recommendations for {movie} (Naive Bayes):")
    recommended_movies = data[data['Genre'] == predicted_genre]
    for i in range(min(5, len(recommended_movies))):
        print(recommended_movies.iloc[i]['Title'])

# Example usage
recommend_content_based_nb('Brand New Day', nb_model_content, data, tfidf_vectorizer)
```

```
Content-Based Recommendations for Brand New Day (Naive Bayes):
Gleboka woda
Only a Mother
The Simple Minded Murderer
Repast
When a Woman Ascends the Stairs
```

## Cross Validation

```
from sklearn.model_selection import cross_val_score
import numpy as np

# Create a Naive Bayes classifier
nb_model_content = MultinomialNB()

# Define the feature matrix (X) and target variable (y)
X = tfidf_matrix
y = data['Genre'].fillna('')

# Perform 5-fold cross-validation
cv_scores = cross_val_score(nb_model_content, X, y, cv=5)

# Display the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))
```

✓ 12.3s

```
C:\Users\aa912\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p
  warnings.warn(
Cross-validation scores: [0.13156499 0.13740053 0.15066313 0.14907162 0.15278515]
Mean CV Score: 0.1442970822281167
```

```
from sklearn.model_selection import cross_val_score
import numpy as np

# Create a Random Forest classifier
rf_model_content = RandomForestClassifier(n_estimators=10, max_depth=5)

# Define the feature matrix (X) and target variable (y)
X = tfidf_matrix
y = data['Genre'].fillna('')

# Perform 5-fold cross-validation
cv_scores = cross_val_score(rf_model_content, X, y, cv=5)

# Display the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))
```

✓ 2.2s

```
C:\Users\aa912\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kf
  warnings.warn(
Cross-validation scores: [0.11777188 0.11034483 0.11299735 0.11458886 0.11618037]
Mean CV Score: 0.1143766578249337
```

```python
from sklearn.model_selection import cross_val_score
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(data['Tags__Summary'].fillna(''))

# Create a Decision Tree classifier
dt_model_content = DecisionTreeClassifier(criterion='entropy', max_depth=5)

# Define the feature matrix (X) and target variable (y)
X = tfidf_matrix
y = data['Genre'].fillna('')

# Perform 5-fold cross-validation
cv_scores = cross_val_score(dt_model_content, X, y, cv=5)

# Display the cross-validation scores
print("Cross-validation scores:", cv_scores)
print("Mean CV Score:", np.mean(cv_scores))
```

✓ 1m 2.5s

```
C:\Users\aa912\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra
  warnings.warn(
Cross-validation scores: [0.19893899 0.21697613 0.23501326 0.25676393 0.24456233]
Mean CV Score: 0.23045092838196285
```

## HyperParameter Tuning

```python
from sklearn.model_selection import GridSearchCV, cross_val_score
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(data['Tags__Summary'].fillna(''))

# Create a Decision Tree classifier
dt_model_content = DecisionTreeClassifier(random_state=42)

# Define the hyperparameter grid to search
param_grid = {
    'max_depth': [5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}  # Adjust the hyperparameter ranges as needed

# Use GridSearchCV to find the best combination of hyperparameters
grid_search = GridSearchCV(dt_model_content, param_grid, cv=5, scoring='accuracy', verbose=1, n_jobs=-1)
grid_search.fit(tfidf_matrix, data['Genre'].fillna(''))

# Display the best hyperparameters and accuracy
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)

# Perform cross-validation with the best hyperparameters
best_dt_model = DecisionTreeClassifier(**grid_search.best_params_)
cv_scores = cross_val_score(best_dt_model, tfidf_matrix, data['Genre'].fillna(''), cv=5)
print("Cross-validation scores with best hyperparameters:", cv_scores)
print("Mean CV Score with best hyperparameters:", np.mean(cv_scores))
```

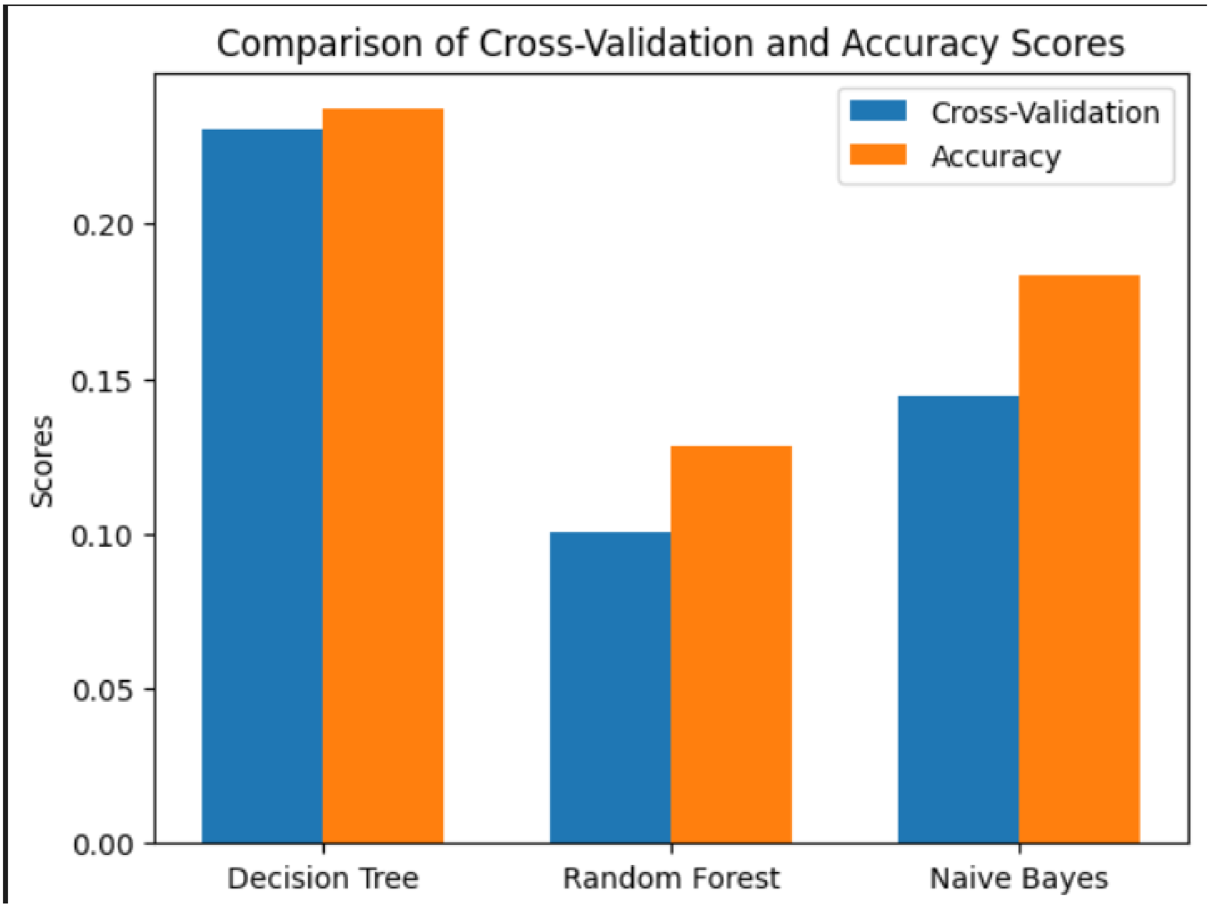
✓ 2m 18.0s

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
C:\Users\aa912\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packa
  warnings.warn(
Best Hyperparameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 10}
Best Accuracy: 0.24785145888594165
C:\Users\aa912\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packa
  warnings.warn(
Cross-validation scores with best hyperparameters: [0.21061008 0.22175066 0.26259947 0.27480106 0.26737401]
Mean CV Score with best hyperparameters: 0.24742705570291776
```

## Graph On Model Accuracy Comparison

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(data['Tags__Summary'].fillna(''))

# Create classifiers
classifiers = {
    'Decision Tree': DecisionTreeClassifier(criterion='entropy', max_depth=5),
    'Random Forest': RandomForestClassifier(n_estimators=10, max_depth=5),
    'Naive Bayes': MultinomialNB(),
}

# Initialize lists to store cross-validation and accuracy scores
cv_scores = []
accuracy_scores = []

# Perform cross-validation and accuracy score calculation
for name, clf in classifiers.items():
    # Cross-validation
    cv_score = cross_val_score(clf, tfidf_matrix, data['Genre'].fillna(''), cv=5)
    cv_scores.append(np.mean(cv_score))

    # Accuracy score
    clf.fit(tfidf_matrix, data['Genre'].fillna(''))
    accuracy = clf.score(tfidf_matrix, data['Genre'].fillna(''))
    accuracy_scores.append(accuracy)

# Plotting
labels = list(classifiers.keys())
x = np.arange(len(labels))
width = 0.35
```

```python
fig, ax = plt.subplots()
bar1 = ax.bar(x - width/2, cv_scores, width, label='Cross-Validation')
bar2 = ax.bar(x + width/2, accuracy_scores, width, label='Accuracy')

ax.set_ylabel('Scores')
ax.set_title('Comparison of Cross-Validation and Accuracy Scores')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()

plt.show()
```




## Best Finalized Model

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction.text import TfidfVectorizer

# Assuming 'data' is your DataFrame
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(data['Tags__Summary'].fillna(''))

# Create a Decision Tree classifier with the best hyperparameters
best_dt_model = DecisionTreeClassifier(max_depth=15, min_samples_split=2, min_samples_leaf=1, random_state=42)
best_dt_model.fit(tfidf_matrix, data['Genre'].fillna(''))

def recommend_movies_dt(movie_title, model, data, vectorizer, num_recommendations=5):
    # Find the index of the movie in the dataset
    idx = data[data['Title'] == movie_title].index[0]

    # Transform the movie description using the TF-IDF vectorizer
    movie_tfidf = vectorizer.transform([data.iloc[idx]['Tags__Summary']])

    # Predict the genre of the movie using the Decision Tree model
    predicted_genre = model.predict(movie_tfidf)[0]

    # Filter movies with the predicted genre
    recommended_movies = data[data['Genre'] == predicted_genre]

    # Exclude the input movie from the recommendations
    recommended_movies = recommended_movies[recommended_movies['Title'] != movie_title]

    # Get the top N recommended movie titles
    top_recommendations = recommended_movies.head(num_recommendations)['Title']
    return top_recommendations

# Example usage
movie_to_recommend_dt = 'Brand New Day'
recommended_movies_dt = recommend_movies_dt(movie_to_recommend_dt, best_dt_model, data, tfidf_vectorizer)
print(f"Recommendations for {movie_to_recommend_dt} using Decision Tree:")
print(recommended_movies_dt)
```

```
Recommendations for Brand New Day using Decision Tree:
2               The Con-Heartist
36                         Alice
139                     Erotikon
158        The House Arrest of Us
206      So Im a Spider, So What?
Name: Title, dtype: object
```

```python
import pickle

# Save the best Decision Tree model to a file
with open('best_decision_tree_model.pkl', 'wb') as model_file
    pickle.dump(best_dt_model, model_file)

print("Model saved to best_decision_tree_model.pkl")
```

✓ 0.0s

```
Model saved to best_decision_tree_model.pkl
```

```python
import pickle

with open('tfidf_vectorizer.pkl', 'wb') as vectorizer_file:
    pickle.dump(tfidf_vectorizer, vectorizer_file)
```

```python
import pandas as pd
import pickle
data = pd.read_excel(r'C:\Users\aa912\OneDrive\Desktop\Netflix Movie Recommendation System\Netflix Movie Recommendation System\Netflix Movie Recommendation Sys
# Save the DataFrame as a pickle file
with open('data.pkl', 'wb') as f:
    pickle.dump(data, f)
```

```python
import pandas as pd
import os

# Save the DataFrame as an Excel file
filename = 'MovieRecommendations.xlsx'
data.to_excel(filename, index=False)

# Download the Excel file
if os.name == 'nt':  # Windows
    os.startfile(filename)
elif os.name == 'posix':  # macOS or Linux
    os.system(f'open {filename}')
else:  # Other OS
    print(f'Please download the file manually: {filename}')
```
✓ 9.9s

```python
import pandas as pd
import pickle
data = pd.read_excel(r'C:\Users\aa912\OneDrive\Desktop\Netflix Movie Recommendation System\Netflix Movie Recommendation System\Netflix Movie Recommendation Sys
# Save the DataFrame as a pickle file
with open('data1.pkl', 'wb') as f:
    pickle.dump(data, f)
```

**Flask Integration**

```
project_directory
  static\css
    style.css
  templates
    index.html
    recommendations.html
  app_with_recommendatio.py
  best_decision_tree_model.pkl
  data1.pkl
  tfidf_vectorizer.pkl
4.2.0
best_decision_tree_model.pkl
data.pkl
data1.pkl
```

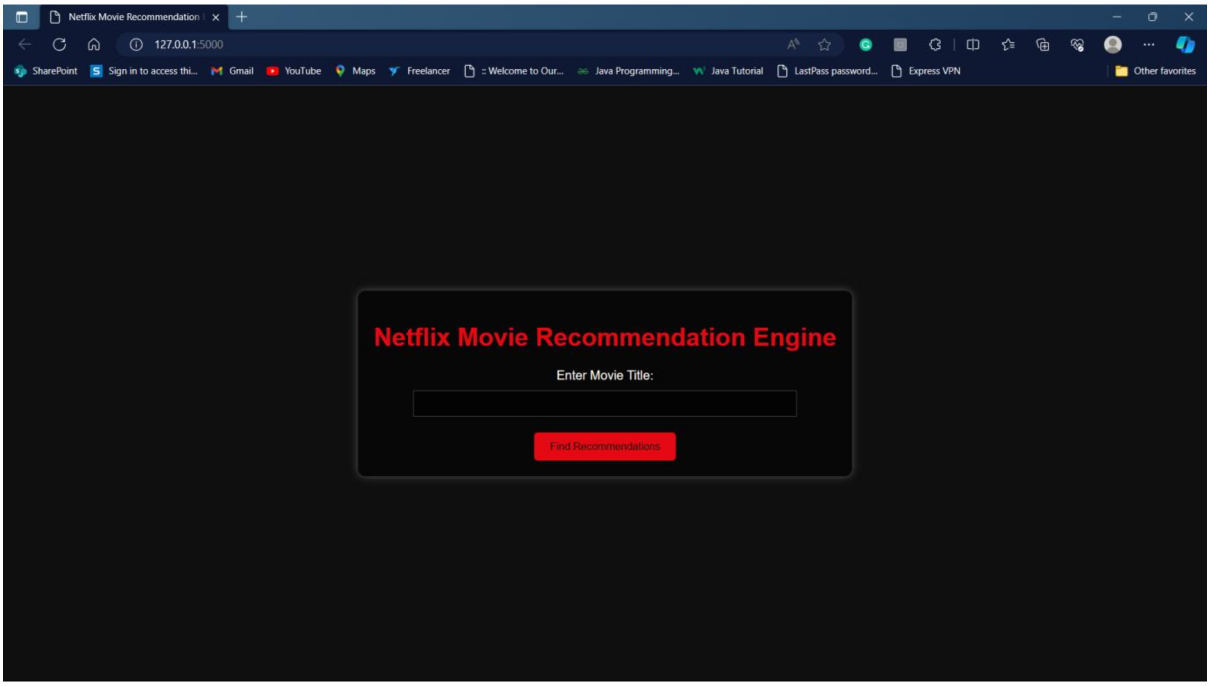
```python
# app_with_recommendation.py

from flask import Flask, render_template, request
import pickle
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_extraction.text import TfidfVectorizer

app = Flask(__name__)

# Load the data, trained Decision Tree model, and TF-IDF vectorizer
with open(r'C:\Users\aa912\OneDrive\Desktop\Netflix Movie Recommendation System\Netflix Movie Recommendation System\
    data = pickle.load(f)

with open(r'C:\Users\aa912\OneDrive\Desktop\Netflix Movie Recommendation System\Netflix Movie Recommendation System\
    best_dt_model = pickle.load(f)

with open(r'C:\Users\aa912\OneDrive\Desktop\Netflix Movie Recommendation System\Netflix Movie Recommendation System\
    tfidf_vectorizer = pickle.load(f)


def recommend_movies_dt(movie_title, model, data, vectorizer, num_recommendations=5):
    # Find the index of the movie in the dataset
    idx = data[data['Title'] == movie_title].index[0]

    # Transform the movie description using the TF-IDF vectorizer
    movie_tfidf = vectorizer.transform([data.iloc[idx]['Tags__Summary']])

    # Predict the genre of the movie using the Decision Tree model
    predicted_genre = model.predict(movie_tfidf)[0]

    # Filter movies with the predicted genre
    recommended_movies = data[data['Genre'] == predicted_genre]

    # Exclude the input movie from the recommendations
    recommended_movies = recommended_movies[recommended_movies['Title'] != movie_title]

    # Get the top N recommended movie titles
    top_recommendations = recommended_movies.head(num_recommendations)['Title']
```

```python
    return top_recommendations


@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        movie_title = request.form['movie_title']
        if movie_title not in data['Title'].values:
            # Movie title not found
            error_message = f"Movie title '{movie_title}' not found in the dataset"
            return render_template('index.html', error_message=error_message)
        else:
            recommended_movies = recommend_movies_dt(
                movie_title, best_dt_model, data, tfidf_vectorizer)
            return render_template('recommendations.html', movie_title=movie_title, recommended_movies=recommended_movies)

    return render_template('index.html')


if __name__ == '__main__':
    app.run(debug=True)
```

```
C:\Users\aa912\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Pytho
ickle estimator TfidfTransformer from version 1.3.0 when using version 1.3.2. This might lead to breaking code or invali
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
C:\Users\aa912\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Pytho
ickle estimator TfidfVectorizer from version 1.3.0 when using version 1.3.2. This might lead to breaking code or invalid
https://scikit-learn.org/stable/model_persistence.html#security-maintainability-limitations
  warnings.warn(
 * Serving Flask app 'app_with_recommendatio'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

These are screenshots of the Netflix Movie Recommendation Engine, which is a machine learning project that recommends movies to users based on their viewing history. The background is black with a little bit of red and white in the center. The text "Netflix Movie Recommendation Engine" is written in white. There is a text field with the prompt "Enter Movie Title". Below the text field, there is a red button that reads "Find Recommendation". We can use the buttons to navigate through the engine and get recommendations to watch

## Home Page




The Netflix recommendation system takes into account a lot of factors, such as:

1. Your interactions with the service (like viewing history and how you rated other titles).

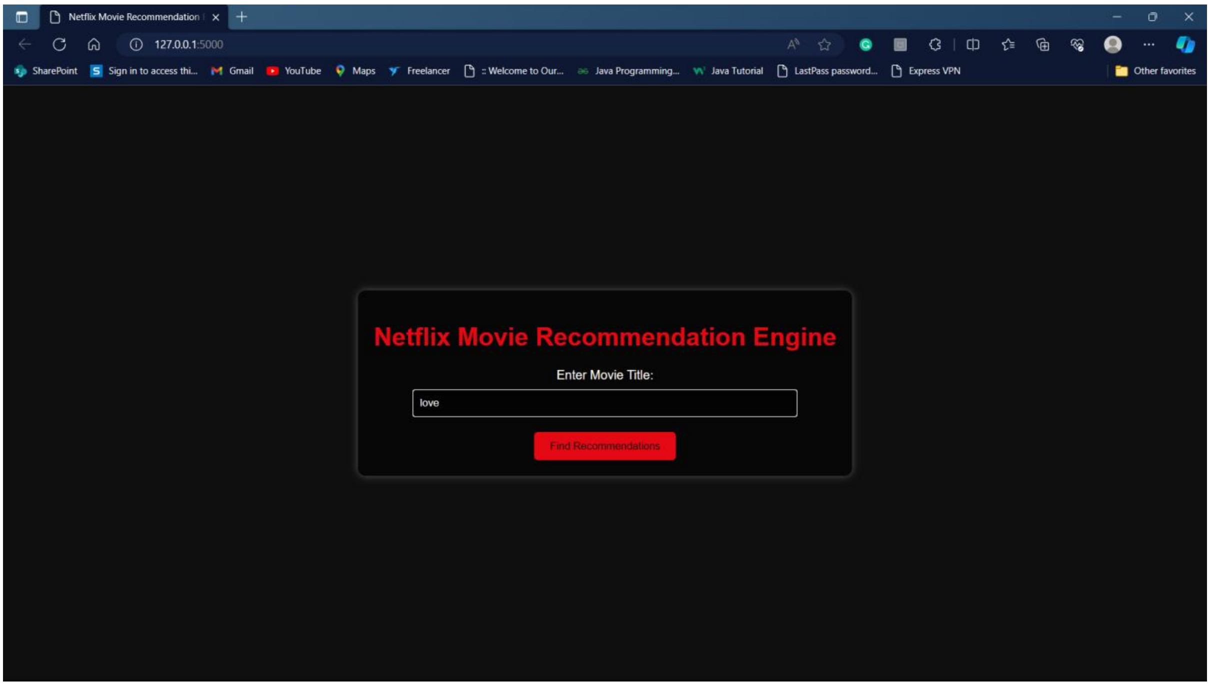
2. Other members with similar tastes and preferences.

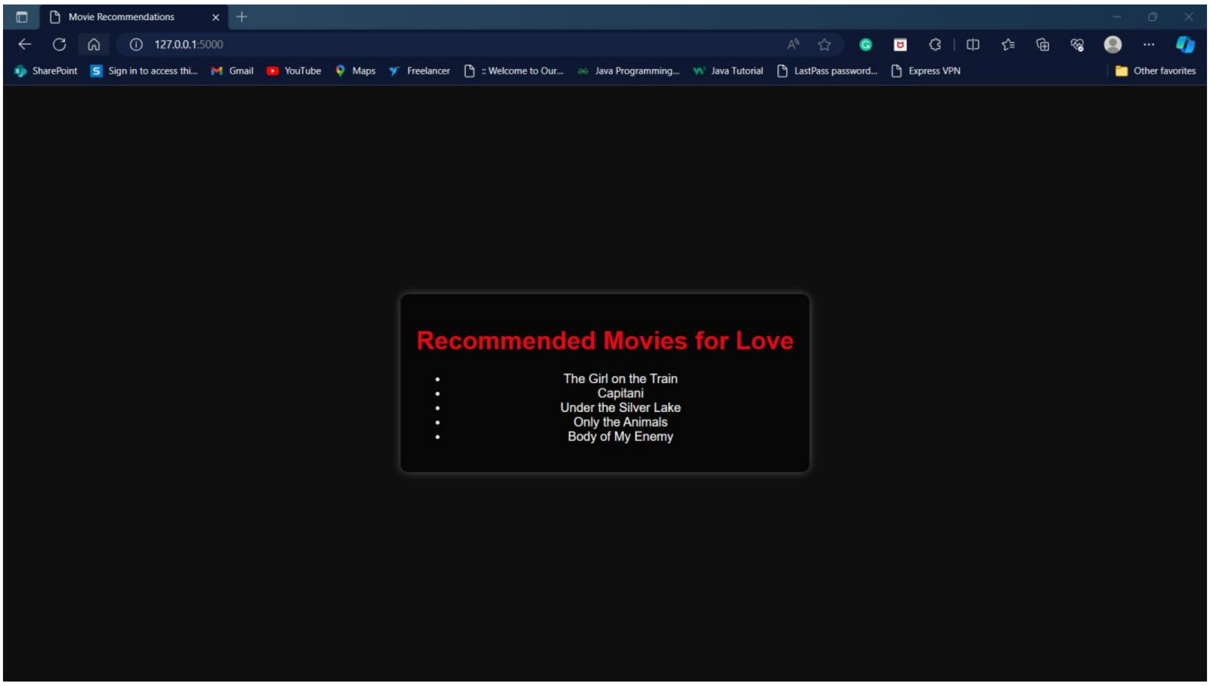
3. Information about the titles, such as their genre, categories, actors, release year, etc.

4. The time of day you watch.

5. The devices you are watching Netflix on.

6. How long you watch.




The system segments viewers into different taste groups and dictates recommendations based on the taste group a viewer falls into . With over 5000 TV shows and movies in the catalogue, it is actually impossible for a viewer to find movies they like to watch on their own. Netflix's recommendation engine automates this search process for its users .

**Category As Iron Man**