University of Essex

School of Mathematics, Statistics and Actuarial Science

MA981 DISSERTATION

# Movie Recommendation Systems

**2212197**

Supervisor: **Danilo Petti**

November 24, 2023
Colchester

**Acknowledgment**

I would like to express my sincere gratitude to **Professor Danilo Petti** for his valuable guidance, support and mentorship throughout the completion of this project. His expertise, insightful feedback, and commitment to academic excellence have been crucial in shaping the content and structure of this document.

**Professor Petti's** insights into my research topic were invaluable, and he always took the time to answer my questions and provide feedback on my work. His attention to detail and his commitment to quality were invaluable in ensuring that my work was presented in the best possible light.

I am truly grateful for the opportunity to have worked with Professor Petti, and I am confident that his mentorship will have a lasting impact on my future career.

## Abstract

In our fast-paced lives, entertainment serves as a crucial element to rejuvenate our spirits and energy. It is significant for people to have fresh mind while working to get a productive result. Movies, music, sports are some of the activities which humans engage themselves to get relaxation. Among them movies play a prominent role as it can be watched whenever, wherever irrespective of time and place. But finding a perfect movie to watch is again a complicated and time-consuming task. This can be dealt by using a recommendation system. These recommendation systems use various filtering techniques to recommend movies to the user, such as demographic filtering, content-based filtering, collaborative based filtering(SVD) and hybrid filtering. These filtering techniques are based on the popularity, overview, metadata, user ratings. Cosine similarity is used to compute the similarity between the items to provide recommendations. By merging the strengths of both collaborative and content-based approaches and mitigating their weaknesses, the hybrid method gives a better approach for providing movie recommendations.

**Key Words:** Demographic Filtering, Content-Based Filtering, Collaborative Filtering, Hybrid Filtering, SVD , Cosine Similarity.

# Contents

# List of Figures

# List of Tables

CHAPTER 1

# Introduction

In today's digital era, people engage in day-to-day routine activities. Individuals strive hard to meet their daily necessities; elderly people work for significant hours just to feed their families, while younger people focus mostly on enhancing their skills to build their careers. In such a way, knowingly and unknowingly, humans become a part and parcel of the daily life routine loop, engaging in similar kinds of activities throughout most of the week.

According to a survey, it is said that if a person engages in similar activities without any sort of refreshment, they might end up in depression. To seek refreshment and relaxation, people allocate a certain amount of time for entertainment. People's preferences for entertainment differ from person to person; some find relaxation in reading a book, playing games, watching movies, or using social media. Among these, the most popular means of engagement is watching movies.

In a world ruled by digital technology and the internet, accessing movies over the internet has become easy. In contrast, in the olden days, people used to go physically to places where opera, skits, plays, or dramas were hosted. Whereas, today's technology allows people to have seamless movie access. However, this convenience comes with a challenge - the overwhelming volume of movie data. Among this huge data, finding a user-preferable movie is a burdensome task. To solve this issue, recommendation systems are used.

Recommendation systems are among the most popularly used machine learning algorithms that suggest items to users based on various factors. These systems find application in diverse domains such as movies, music, books, and social media platforms like Facebook, Instagram, and YouTube. It has been observed that more than two-thirds of the population prefer receiving recommendations due to the numerous perks associated with them.These systems leverage advanced algorithms to analyze user preferences, suggesting movies that align with individual tastes and interests. These responses are tailored to have an enjoyable viewing experience to user.[10]

## 1.1 Evolution of recommendation systems

In the realm of digital information overload, both users and content providers often struggle with the sheer volume of data available. Users often find difficulty in accessing interested information from huge data; this is where the challenge for content providers arises. The main challenge for content providers is to provide personalized suggestions tailored to the user's interests. To address this challenge, the development of recommendation systems becomes crucial. These systems serve as a bridge between users and content providers.

Recommendations system is basically an information filtering algorithm, that suggests or recommends the best matching information/item to the user. This algorithm creates a list of items that user might find interesting by skimming through all possible matching. These recommendations will basically work on the profile, browsing history, people with similar traits etc. Recommendation systems have a wide range of applications in various domains, such as e-commerce, content streaming, social media and many more. By developing the recommendation systems, one can provide personalized suggestions to the users/customers which in turn enhances the user engagement in that particular website and leading to an increase in the profits for the company. This recommendation system not only helps the providers by enhancing user engagement but also helps the user by providing suitable information and saving time. In a world where individuals are consistently occupied with a finite number of tasks within a limited timeframe, these recommendation systems play a pivotal role in helping users make informed choices effortlessly, ultimately contributing to time efficiency. There are various ways to recommend movies to a user. The following are the some popular filtering methods of recommending movies to user based on the techniques and data sources. [11]

- Content-Based Filtering
- Collaborative Filtering
- Hybrid Filtering
- Demographic Filtering

This project centers on the implementation of a recommendation system within the domain of movies. It is widely acknowledged that movies play a pivotal role in the entertainment industry. During leisure time, individuals frequently prioritize watching movies over other activities. According to recent surveys, a significant number of people express a preference for watching movies and hosting movie nights during weekends.

The movie domain boasts an extensive collection of films available on the internet[13]. Manually selecting a movie from this vast list can be a time-consuming and cumbersome task, potentially diminishing the viewer's interest. To streamline and enhance user flexibility, recommendation systems are favored for suggesting movies. These systems propose movies to users based on their viewing history, the popularity of movies in their region/country, or movies enjoyed by users with similar preferences.

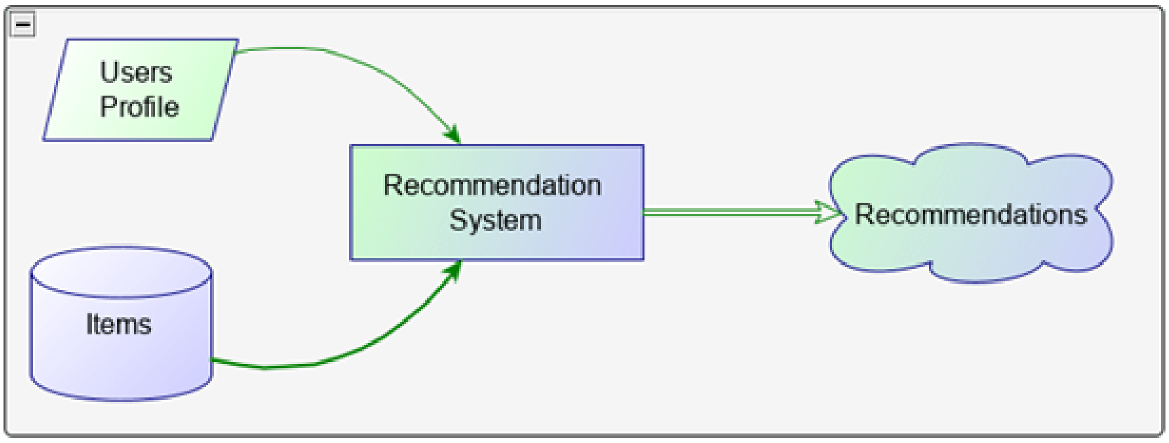

Figure 1.1: A Block diagram of recommendation systems working
Working of Recommendation systems

The above diagram1.1 shows the working of the recommendation systems[49]. From the diagram it can be interpreted that the user profile and items play a prominent role in providing the recommendations. In users profile component stores the crucial information about the user's browsing history, ratings, reviews and demographics. This data is utilized by the recommendation systems to generate the personalized suggestions. On the other hand, the item component provides a list of items that the recommendation systems can recommend, this includes the genres, overview, cast, crew etc. The recommendation system includes the machine learning models which are trained to analyse the user's profile and other data[50], such as the popularity of the items and the ratings of the other users, to generate the recommendations.

## 1.2 Benefits of using recommendation systems

The integration of recommendation systems benefits both the user and the application by enhancing the user experience and increase in the user engagement in a variety of applications. The following are some of the benefits of using the recommendation systems. [14]

- **Personalization:** Recommendation systems excel in tailoring suggestions to individual users, leveraging insights from their browsing history and preferences[53]. This not only helps user to discover new items but also it saves time and effort in seeking an item.

- **Engagement:** Personalized recommendations contribute to prolonged user engagement, resulting in extended session times and heightened user retention[56]. This prolonged engagement, in turn, plays a crucial role in boosting the overall revenue of the company.

- **Discovery:** Recommendation systems can aid in discovering new items that users may struggle to find manually.By suggesting items similar to those already interacted[55] with and those trending among users with similar interests, the system enhances the overall discovery experience.

- **Relevance:** Recommendation systems uses the machine learning algorithms to generate relevant recommendations, even for the users with broad and niche interests.[15] The machine learning algorithms helps in analysing the data and deriving the relevant pattern from data.

## 1.3 Objective of the Project

Around 800-900 movies are released annually, finding a preferred movie from this huge dataset manually is really a challenge.This challenge can be dealt by using the simple AI algorithm based movie recommendation system. The main objective of this project is to provide personalized movie recommendations to users[51]. This system suggests movies to users based on a variety of factors, including individual traits, viewing history, feedback received from users with similar tastes and , overview , metadata , popularity of movie . This project leverages three datasets-movies, credits, and ratings. The initial phase involves preprocessing these datasets to address null values, ensure data formatting, and extract relevant features. Subsequently, the preprocessed data is utilized to train four distinct machine learning filtering techniques- Demographic filtering, content-based filtering, collaborative filtering, and a hybrid approach. The goal of these filtering techniques is to provide the tailored personalized movie recommendations to the user. It considers user's previous watch history, likes, ratings etc while tailoring the suggestions[54].

## 1.4 Outline of the project

The initial chapter of this project provides an insight into the overview and evolution of movie recommendation systems. Following this, Chapter 2 delves into a comprehensive review of existing recommendation systems, offering insights into the current landscape and understanding the methodologies employed in the field.Chapter 3 focuses on two key aspects. Firstly, it provides a detailed description of the dataset utilized in the project. Secondly, it covers the implementation of various filtering algorithms. Prior to implementation, an exploratory data analysis is conducted to familiarize oneself with the dataset. Chapter 4 presents and discusses the results obtained from the implemented algorithms.Lastly, Chapter 5 encapsulates the project with a conclusion, summarizing key findings, and suggesting potential avenues for future research and development within the domain of movie recommendation systems. This well-structured approach ensures a logical flow, covering literature review, data handling, algorithm implementation, results, and concluding with insights for future exploration.

Chapter 2

# Literature Review

The field of recommendation systems has seen significant advancements in recent years due to the growing need for personalized content delivery in various domains. Among these, movie recommendation systems play a crucial role in facilitating user engagement and satisfaction.

The paper presented by Agrawal and Jain [1] introduces a novel approach of combining the content-based filtering, collaborative filtering, support vector machine and genetic algorithm to introduce the hybrid filtering. Their findings suggest a notable improvement in accuracy, quality and scalability compared to pure approaches. Using hybrid filtering, they tried to overcome the limitation of individual approach and enhance the accuracy.

Zan Wang, Xue Yu[2] approached hybrid model-based Collaborative Filtering technique using the K-means coupled with the genetic algorithms. The dimensions of the data are reduced using the principal component analysis, this in turn reduced the computational complexity and better performance. Clustering of users like-minded is done to suggest the movies to similar type of users.

The paper proposed by the Meenu Gupta, Aditya Thakkar [3]uses a notable approach to enhance the performance of the recommendation systems using the collaborative filtering with KNN algorithm. They used the item-based collaborative filtering, due its ability to analyse user interactions and provide better predictions. Gupta and Thakkar used KNN algorithm along with cosine similarity to generate movie suggestions. On comparative analysis, it can be said that the item-based collaborative filtering gave a significant precision and F1 compared to content-based approach.

George Lekakos, Petro Carvelas [4] has used various techniques to find the better performing model. The authors followed a systematic approach in developing recommendation systems from content-based to collaborative filtering to hybrid filtering. Used MAE to compare the methods performance mathematically. On comparing the filtering models, hybrid filtering gave the better performance by enhancing the accuracy and coverage and reducing the execution time.

A novel approach of using the SVD (singular Value Decomposition) is used along with demographic felting to enhance the performance of Collaborative filtering by Vozalis and Mrgaritis[5]. The authors tested four variants such as SVD on just demographics, just ratings, both or neither on movielens dataset. The user-based CF, results shows that all the SVD variants perform similar baseline CF without much improvement, Whereas, the item-based CF, application of SVD to ratings and demographic matrices improves accuracy substantially, with SVD on both matrices performing best and reducing error by 6.6% compared to baseline item-based CF.

Zhang et al.[6] present a novel collaborative filtering approach called Weighted KM-Slope-VU for movie recommendations. They introduce the concept of virtual users to represent clusters of users, utilizing K-means clustering based on profile attributes. The proposed Weighted KM-Slope-VU algorithm outperforms basic KM-Slope-VU, achieving comparable accuracy to matrix factorization methods like SVD. The authors further deploy their recommendation system, MovieWatch, in a live environment, collecting user feedback for evaluation. The real-life experiment suggests the system's effectiveness, although some challenges related to demographic differences and outdated movie data are acknowledged.

The paper published by Monjoy kumar, Ankur Singh [7] introduces MOVREC, a movie recommend-er system based on collaborative filtering. The system allows users to select preferences, and recommendations are generated by analyzing user choices, sorting movies based on ratings, and employing the K-means algorithm. Challenges addressed include user-friendliness, diverse movie databases, and attribute weighting. The proposed algorithm involves pre-filtering, weighted attributes, and K-means clustering. The study emphasizes the importance of user ratings and votes, with a survey supporting the weight assignment. Challenges include system satisfaction and geographical diversity.

The paper, authored by Wu, Garg and Bhandary, [8]introduces a collaborative filtering technique: user-based and item-based filtering methods. The authors leverage the Mahout library, built on the Hadoop platform, for implementation, employing algorithms such as loglikelihood similarity and Pearson coefficient. The system quantitatively evaluated, demonstrating accurate predictions, and potential issues like new user problem and item cold start are acknowledged with proposed resolutions.

CHAPTER 3

# Methodology

## 3.1 Machine learning

Machine learning is a subset of the artificial intelligence (AI) [34] that focuses on developing algorithms using the hidden patterns of the datasets [32]. The primary objective of machine learning models is to automate tasks traditionally performed manually by humans[35]. These models are inspired by the human brain, these models learn from the data provide just as like the human brain learns from experience[33]. These models don't need to be explicitly programmed. Machine learning models are used to solve the complex problems[39], these algorithms are built on the basis of the mathematical equations which suits the hidden pattern in the provided data.

The block diagram 3.1 illustrates the working of the machine learning model[20]. The majority of available data is often in its raw form. However, this raw data is unsuitable for training the machine learning model, so this data has to be cleaned and processed to make it compatible for training the machine learning model. The data pre-processing involves cleaning of the data such as, removing null values, formatting the data, scaling the data and removing noise. This processed data is then fed into the machine learning algorithm. As there are many different types of machine learning algorithms are present, choosing a particular algorithm is an iterative process.The effectiveness of a model is assessed based on metrics such as accuracy, precision, recall, etc. Subsequently, the chosen model is deployed into the application, allowing users to

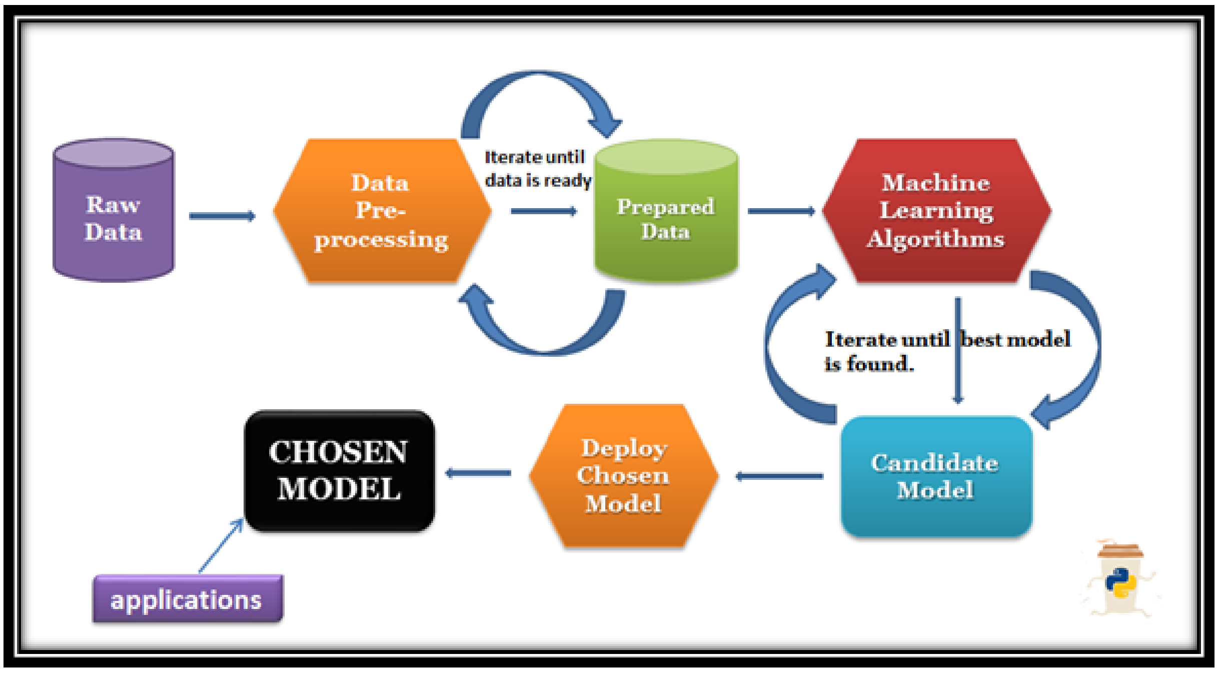

Figure 3.1: Block diagram of machine learning process
Working of Machine Learning Model

make informed decisions and predictions.

**Classification of Machine learning Algorithms:**

Machine learning algorithms are categorized into various types based on their learning style and the nature of the training data. Following are the primary types of machine learning algorithms.[36]

1. Supervised learning
2. Unsupervised learning
3. Reinforcement learning

**Supervised Learning:** As the name suggests, these machine learning models require supervision to learn (*labelled data*). These algorithms are trained on datasets where the desired outcomes are already known. [37]The primary goal of this model is to map the input data to the output data. The performance of the model is evaluated using test data to ensure it predicts the right outcomes. The main application of these models can be seen in regression and classifications problems.

The figure 3.2 shows the pictorial representation of the supervised machine learning model. The labeled data of apples and cupcakes is used to train the model [21]. When new unlabeled data (an image ) is given to the machine learning model,the model labels it as a cupcake based on the features such as size, shape.

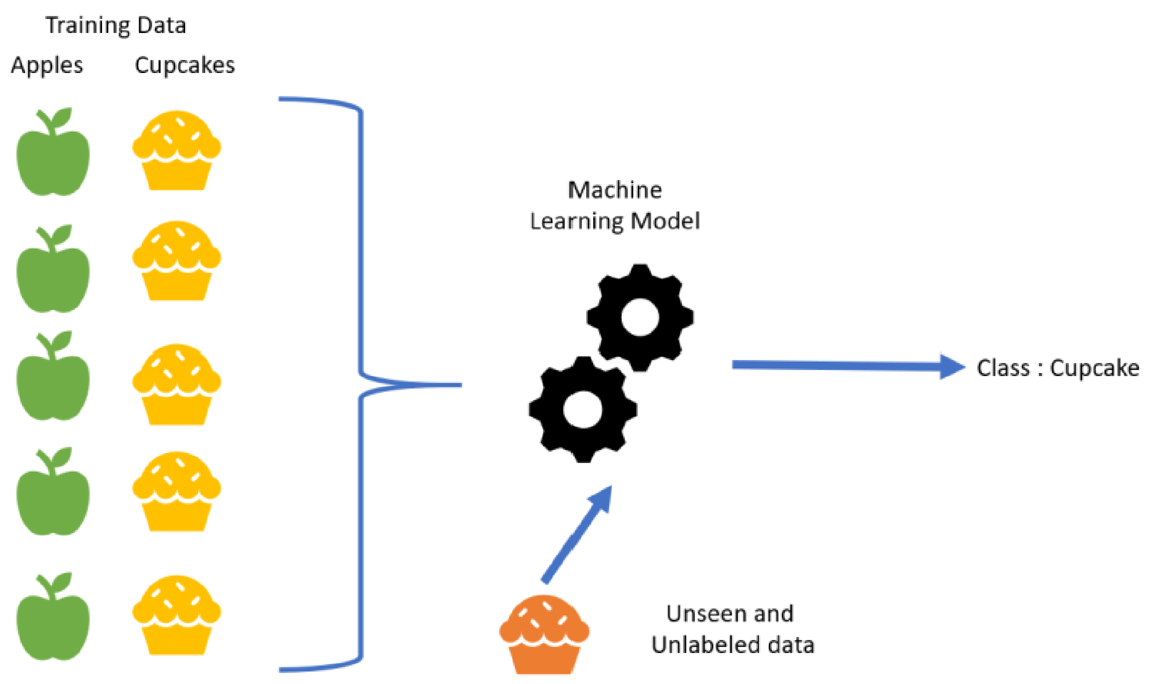

Figure 3.2: Machine learning trained using the labelled dataset
Supervised Model

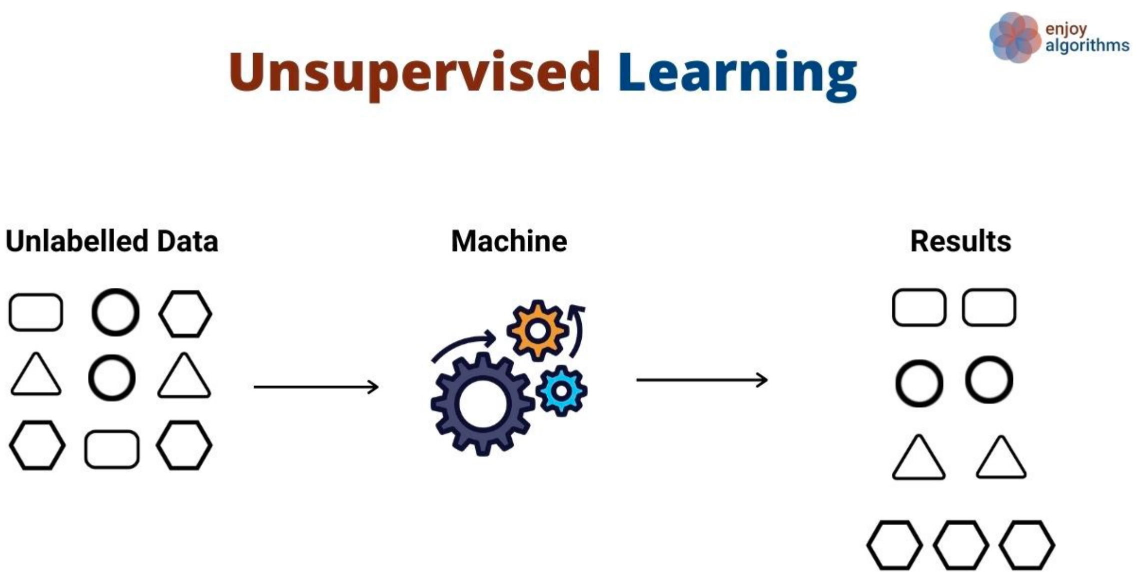

Figure 3.3: Machine learning model trained using the unlabelled data
Unsupervised Model

**Unsupervised learning:** As the name suggests, this model doesn't require any supervision to learn, it learns by analysing patterns in the dataset. Unsupervised machine learning models are trained on the unlabeled data, which means data without predefined outputs or labels. [22]The main application of these models can be observed in clustering and association problems.

The image 3.3 shows the pictorial representation of the unsupervised machine learning model. The unlabelled data of polygons is fed to the machine learning model, and based on the shape and features , the model groups the data without any labels.

**Reinforcement learning:** This model learns through the trial-and-error method[57]. The agent performs a task by interacting with the environment and learns with the help of the feedback it receives, presented as rewards or penalties [38]. Reinforcement learning is particularly useful in situations where the best decision-making strategy isn't known beforehand and needs to be discovered through trial and error. It is used in various domains such as, gaming, robotics, autonomous systems, recommendation systems and more[58].

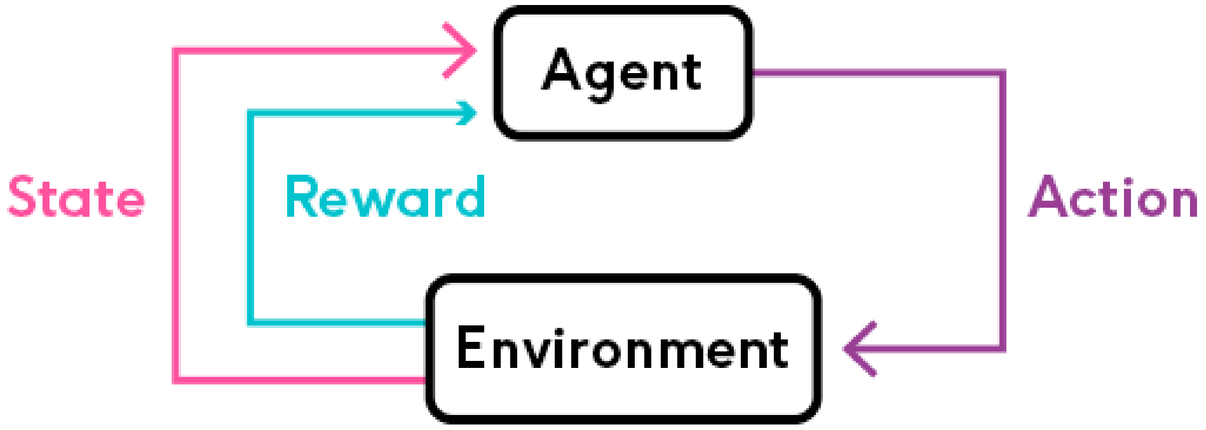

Figure 3.4: Working of Reinforcement learning model
RML

The above image 3.4 illustrates the pictorial representation of the reinforcement machine learning model. [23]The agent observes the current state of the environment and takes the selected action on the environment. Upon receiving the selected action, the environment sends the reward and current state to the agent, allowing the agent to adjust the selected action based on the feedback from the environment.

Recommendation systems is one of the applications of reinforcement learning as the model uses the user's previous data (watch history, ratings, likes, dislikes, etc) to learn and make decisions[59]. It tracks how often the user watches the recommended movie, and based on this feedback, the model tailors the future suggestions. Content-based filtering, Collaborative filtering are the two major filtering techniques used for recommending movies to users.

## 3.2 Dataset Discription

The following are the three different datasets used in developing a movie recommendation systems using four filtering techniques.

1. Movies dataset
2. Credits dataset
3. Ratings Dataset

**Description of Movies dataset:**

The movies dataset comprises 4803 records distributed across 19 columns. Among these columns, there are seven numeric columns, seven string columns (with one being categorical), and five columns in JSON format.The table 3.1 describes the movie dataset atributes

Table 3.1: Movie Dataset Attributes

| **Attribute** | **Description** |
|---|---|
| Budget | Budget allocated for producing the movie. |
| Genres | Categorizes movies based on their content, style, and themes (e.g., Action, Drama, Comedy). |
| Homepage | A link to the official movie homepage. |
| Id | Unique identifier for each movie in the dataset. |
| Keywords | Words or phrases capturing the essence of the movie. |
| Original_language | Language in which the movie was originally produced. |
| Original_title | Movie title in the original language. |
| Overview | Brief description or summary of the movie's plot. |
| Popularity | Measure of the movie's public recognition or discussion. |
| Production_companies | List of production companies involved in making the movie. |

| | |
|---|---|
| Production_countries | Countries where the movie was produced. |
| Release_data | Date of the movie's release. |
| Revenue | Total income generated by the movie (e.g., box office collections, home video sales). |
| Runtime | Duration of the movie. |
| Status | Current status of the movie (e.g., Released, Rumoured, Postproduction). |
| Tagline | Short and memorable phrase associated with the movie for marketing. |
| Title | Movie title in standardized format. |
| Vote_average | Average rating given to the movie by viewers or critics. |
| Vote_count | Total count of votes or ratings received by the movie. |

**Discription of Credits dataset** This dataset comprises of 4803 records distributed across 4 columns. Among these, one column is numeric, another column is a string type, while the remaining two columns are in JSON array format.The table 3.2 describes the credits dataset attributes.

Table 3.2: Credit Dataset Attributes

| Attribute | Description |
|---|---|
| Movie_id | Unique identifier for each movie in the dataset |
| Title | The title of the movie |
| Cast | Information about the actors and their roles in the movie. It includes details like cast_id, character name, gender, and actor names. |
| Crew | Information about the production crew members involved in making the movie. This can encompass a wide range of roles such as director, producer, cinematographer, etc. |

**Description of Ratings Dataset** The Ratings dataset consists of around 100k records distributed across four numeric columns.The table 3.3 describes the attributes of the rating dataset.

Table 3.3: Ratings Dataset Attributes

| Attribute | Description |
|---|---|
| userId | Unique identifier for users who provided ratings. |
| movieId | Unique identifier for movies being rated. |
| rating | The rating given by a particular user to a specific movie. Ratings are typically numerical values given on a scale (e.g., from 0.5 to 5) to indicate the user′s opinion or preference for the movie. |
| timestamp | Indicates the time when the rating was recorded, often represented in Unix time or a similar timestamp format, showing the number of seconds or milliseconds since a specific reference date. |

## 3.3 Implementation

Following are the steps involved in developing the movie recommendation system.

1. Data Integration and Pre-Processing:
2. Exploratory Data Analysis
3. Demographic based filtering
4. Content-based filtering
5. Collaborative based filtering
6. Hybrid filtering

### 3.3.1 Data Integration and Pre-Processing

In Data Pre-Processing, the primary focus lies in tasks such as data cleaning , data formatting, removing missing values, nullifying the errors etc. Since various methodologies are employed in developing a movie recommendation system, it is challenging to find all the required data in single dataset. Therefore, the construction of a recommendation system involves the utilization of three distinct datasets.

The ′movies′ and ′credits′ datasets underwent a seamless integration process, leveraging the unique ′movie _id′ as the key identifier. This merging operation ensured a cohesive dataset, consolidating essential information for subsequent analysis.

- **Data Pre-Processing**

  During the initial phase of data preparation, observations with missing values in the ′overview′ column are systematically removed. This strategic approach streamlined the dataset, enhancing the quality and reliability of the subsequent analyses.

- **Feature Selection and Dimensionality Reduction**

  To optimize the performance of the model and mitigate potential overfitting, a targeted feature extraction strategy was employed. Specifically, the columns ′homepage′, ′tagline′, and ′runtime′ are deliberately excluded from the dataset. This deliberate reduction in dimensionality was undertaken with a careful consideration of the overarching objective: to refine the dataset for more effective modeling.

- **Data Formatting and Extraction**

  In the dataset provided, several columns, namely ′keywords′, ′cast′, ′crew′, and ′genres′, contain information in JSON format. To facilitate analysis, these columns are processed to convert the JSON strings into Python lists.This transformation facilitates easier handling and extraction of information during subsequent analyses.

  – **Top Cast, and Crew**

    From the ′cast′ column, the top three actors for each film are identified. Additionally, the director of each movie was determined by extracting information from the ′crew′ column.

  – **All Genres, Keywords**

    All available genres are compiled into a comprehensive list, offering a clear overview of the various genres associated with each movie. From the ′keywords′ column, top three most frequently used keywords associated with each movie are extracted.

The formatted data now presents a structured and easily accessible representation of these key attributes, facilitating further analysis and visualization.

### 3.3.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) serves as the fundamental purpose of comprehensively understanding and interacting with a dataset before delving into more advanced analyses. It is an approach to analyzing data sets to summarize their main characteristics, often employing statistical graphics and other data visualization methods. The primary goal of EDA is to understand the data, identify patterns, detect outliers and anomalies. EDA not only aids in understanding the structure and patterns inherent in the data but also facilitates a familiarity with the dataset's content, thereby establishing a robust foundation for subsequent implementation.

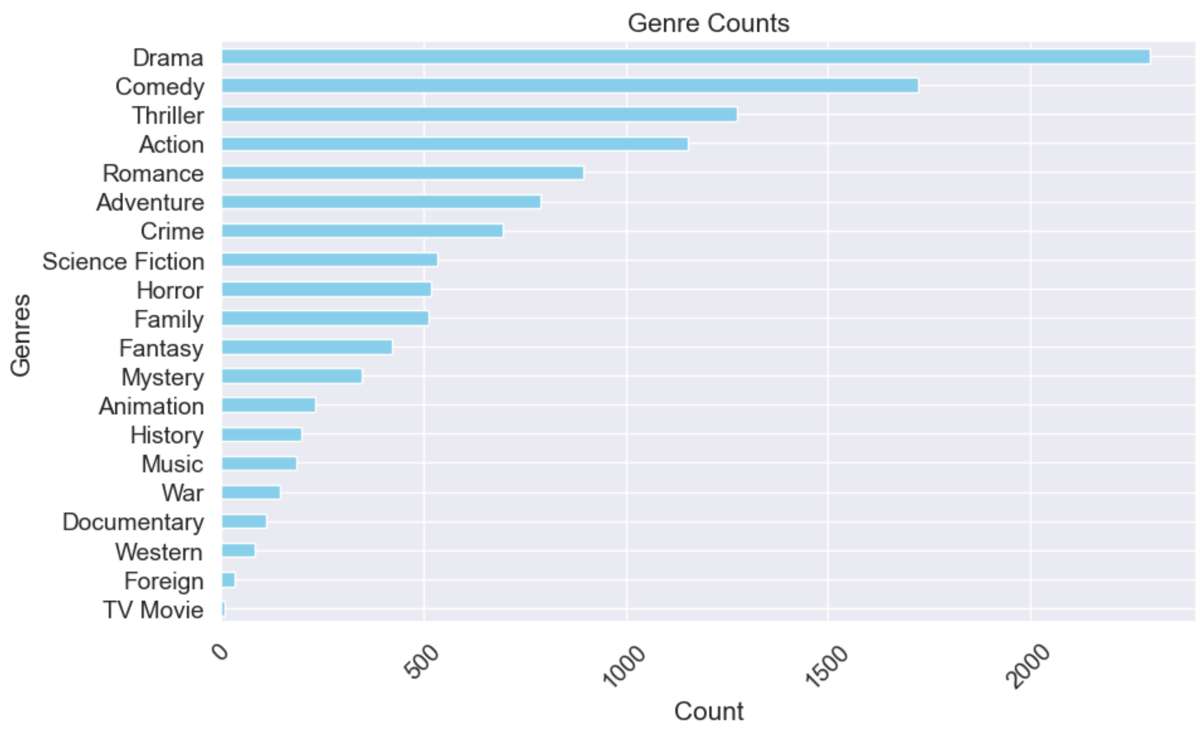

Figure 3.5: Distribution of Movie genres by count

The above graph 3.5 provides the pictorial representation of the distribution of movies across various genres in the dataset. "Drama" emerges as the most prevalent genre, followed closely by "Comedy", suggesting that there is a strong interest in movies that delve into human experiences and humour. Followed by, "Triller" and "Action", these genres are known for their excitement, suspense, and adrenaline-pumping sequences, which attracts substantial viewership interested in action-packed and suspenseful storytelling. Furthermore, there are significant number of "Romance"," Adventure"," Crime" genres movies, indicating that people are much interested in the exploration of love, thrilling and intricate criminal narratives. It can be said that there is similar viewer preference for the "Science Fiction"," Horror" and "Family" genres, suggesting a diverse yet substantial audience inclination toward the wonders of futuristic worlds, the thrill of fear, and the appeal of family-friendly content. Genres such as "Animation," "History," "War," and "Documentary" have lower counts, indicating a possible preference among younger audiences for animated movies and a comparatively lesser interest in historical, war, and documentary genres. Conversely, genres like "Foreign" and "Western" exhibit significantly lower counts, suggesting that the audience may be less inclined to watch films not in their native language. The lower popularity of the "TV Movie" can be attributed to its association with the small-scale productions often intended for television.

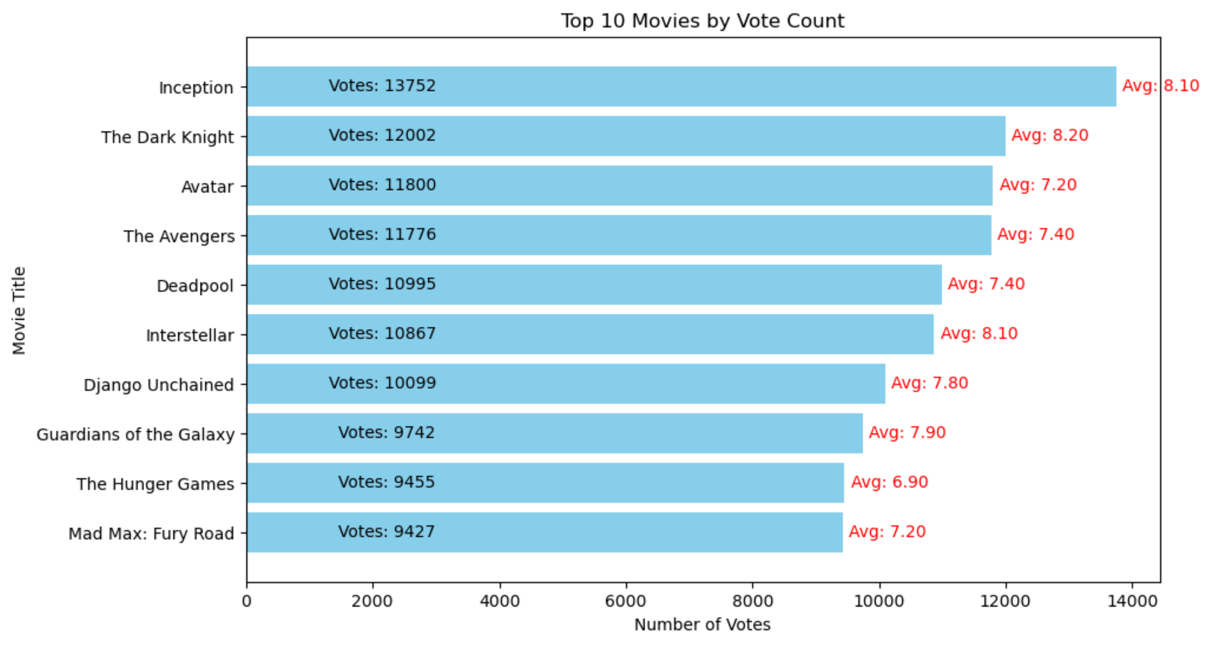

Figure 3.6: Ranking of top 20 movies by Vote Count

The bar plot 3.6 above provides a visual representation of the top 10 movies sorted by vote count alongside their vote averages. "Inception" emerges as a standout, boasting the second-highest vote count and an exceptional average rating of 8.1.This suggests the widespread appreciation and a strong resonance with viewers. Conversely, movies like "The Dark Knight" and "Interstellar" are slightly lower in terms of vote count compared to "Inception", but maintain exceptionally high average ratings of 8.2 and 7.2, respectively, reflecting consistent critical acclaim.

The disparities between popular appeal and critical acclaim are evident. Films like "Avatar" and "The Avengers" have high vote counts, reflecting their widespread popularity, while others like "The Dark Knight" and "Interstellar" maintain high critical ratings, suggesting a strong appreciation among viewers and critics. The differing ratings might stem from various factors such as storytelling, genres, and audience expectations. This combination signifies not only broad popularity but also widespread acclaim from viewers. These comparisons illuminate the intricate interplay between vote count and vote average, providing valuable insights into the film's levels of popularity and audience satisfaction. This information is valuable for filmmakers and industry professionals, offering guidance on audience engagement and content creation strategies.

Table 3.4: Count of movies categorised by status

| Status | Count |
|---|---:|
| Released | 4795 |
| Rumored | 5 |
| Post Production | 3 |

The table 3.4 shows the categorical distribution of data in the "Status" column of the movie dataset offers a valuable snapshot of the current status of movies within the dataset. From the table it can be interpreted that most of the movies are in "Released" status, this signifies that these movies have successfully completed production and readily available for the public viewing. In contrast, only around 5 movies are under the "Rumored" status, indicating that these films are most likely in the early stage of development, without any official report supporting their existence. Lastly, there are 3 movies under the "Post Production" category, indicating that these movies have completed principal photography and are in the final stage of production. This stage typically involves critical activities such as editing, visual effects, sound design, and other finishing touches before the movie is ready for public viewing. This breakdown provides a meaningful insight into the distribution of movies across various stages of production and release, offering a comprehensive view of the dataset's content landscape.

Table 3.5: Comparison of production companies

(a) Top 10 Production companies based on Number of movies produced

| production_companies | num_movies | avg_popularity |
|---|---|---|
| Warner Bros. | 319 | 36.247154 |
| Universal Pictures | 311 | 31.468230 |
| Paramount Pictures | 285 | 30.385787 |
| Twentieth Century Fox Film Corporation | 222 | 33.850898 |
| Columbia Pictures | 201 | 31.421507 |
| New Line Cinema | 165 | 27.761532 |
| Metro-Goldwyn-Mayer (MGM) | 122 | 20.150201 |
| Touchstone Pictures | 118 | 20.441030 |
| Walt Disney Pictures | 114 | 42.951517 |
| Relativity Media | 102 | 31.683988 |

(b) Top 10 Production companies based on average popularity

| production_companies | num_movies | avg_popularity |
|---|---|---|
| The Donners' Company | 1 | 514.569956 |
| Bulletproof Cupid | 1 | 481.098624 |
| Kinberg Genre | 2 | 326.920999 |
| Illumination Entertainment | 5 | 234.920042 |
| Deluxe Digital Studios | 1 | 198.372395 |
| Vita-Ray Dutch Productions (III) | 1 | 198.372395 |
| Syncopy | 8 | 192.417187 |
| Lynda Obst Productions | 4 | 192.226362 |
| Cruel & Unusual Films | 1 | 155.790452 |
| Atman Entertainment | 1 | 146.757391 |

The two tables 3.5a and 3.5b present distinct profiles of production companies, each adhering to its unique strategic approach to filmmaking. In the table 3.5a, companies like

Warner Bros., Universal Pictures, and Paramount Pictures stand out for their extensive filmography, collectively producing hundreds of movies. By tabulated analysis, it can be analysed that even though these production companies have produced hundreds of movies, their average popularity is moderate,indicating a broader audience reach without consistently high viewer ratings.

Conversely, in the second table 3.5b , production companies like "The Donners' Company" and "Bulletproof Cupid" have opted for a more selective approach, by producing a limited number of movies. However, it is noteworthy that even though they have produced a smaller number of movies but the average popularity score gained by these production companies is high, indicating a dedicated and enthusiastic viewer base. This suggests that a focused, quality-over-quantity strategy can lead to exceptionally well-received films. Possible factors influencing this difference in strategy may include resource allocation, genre specialization, or a deliberate emphasis on producing high-impact, memorable content. Ultimately, both approaches have their merits, and the choice likely hinges on a company's specific goals and resources in the competitive film industry.

Table 3.6: Top 20 production countries

| num_movies | country |
|---:|---|
| 636 | United Kingdom |
| 324 | Germany |
| 306 | France |
| 261 | Canada |
| 110 | Australia |
| 72 | Italy |
| 71 | Spain |
| 59 | China |
| 58 | Japan |
| 54 | India |
| 48 | Hong Kong |
| 37 | Ireland |
| 30 | Mexico |
| 28 | New Zealand |
| 25 | Belgium |
| 24 | Czech Republic |
| 20 | South Africa |
| 20 | Denmark |
| 19 | Switzerland |
| 19 | Sweden |

Certainly! Here's a simplified and professional rephrasing of the analysis:

The table 3.6 illustrates the comparative film production of different countries. The

United States stands out as the leader in movie production, showcasing its prominent position in the global film industry. Following closely is the United Kingdom, highlighting its strong presence in cinema. Germany and France also demonstrate robust film industries, contributing significantly to the global landscape. Canada, Australia, and Italy make substantial contributions, each adding a noteworthy number of films to the industry. Spain and China, while well-represented, show China's emerging prominence in global cinema. India, known for Bollywood, maintains a prolific film output, influencing international trends. Hong Kong and Japan, with their unique storytelling styles, contribute significantly. This analysis underlines the diverse contributions of various countries to the film industry, each bringing its cultural perspective. This diversity enriches global storytelling, making the film industry an exciting area for further exploration.

Table 3.7: Count of movies by original languages

| original_language | Count |
|---|---|
| English(en) | 4505 |
| French(fr) | 70 |
| Spanish(es) | 32 |
| Chinese(zh) | 27 |
| German(de) | 27 |

The table 3.7 provides a comprehensive overview of movies categorized by their original language. Notably, English takes the lead by a significant margin, encompassing a substantial 4505 movies. This dominance underscores the widespread global influence of English-language cinema and its accessibility to diverse audiences. Following this, French, Spanish, Chinese, and German contribute to the dataset with 70, 32, 27, and 27 movies respectively. French-language films maintain a significant presence, indicative of the enduring cultural impact of French cinema. Similarly, Spanish-language movies, though fewer in number, highlight the continued popularity and global reach of Spanish-language cinema. Chinese-language films, predominantly in Mandarin, and German-language films are also well-represented, emphasizing their influence and appeal on an international scale. This breakdown underscores the diverse linguistic landscape of cinema, providing audiences with a broad spectrum of cultural and storytelling experiences. It highlights the vital role that language plays in shaping the global film industry, enabling a multitude of voices and narratives to resonate with audiences worldwide.

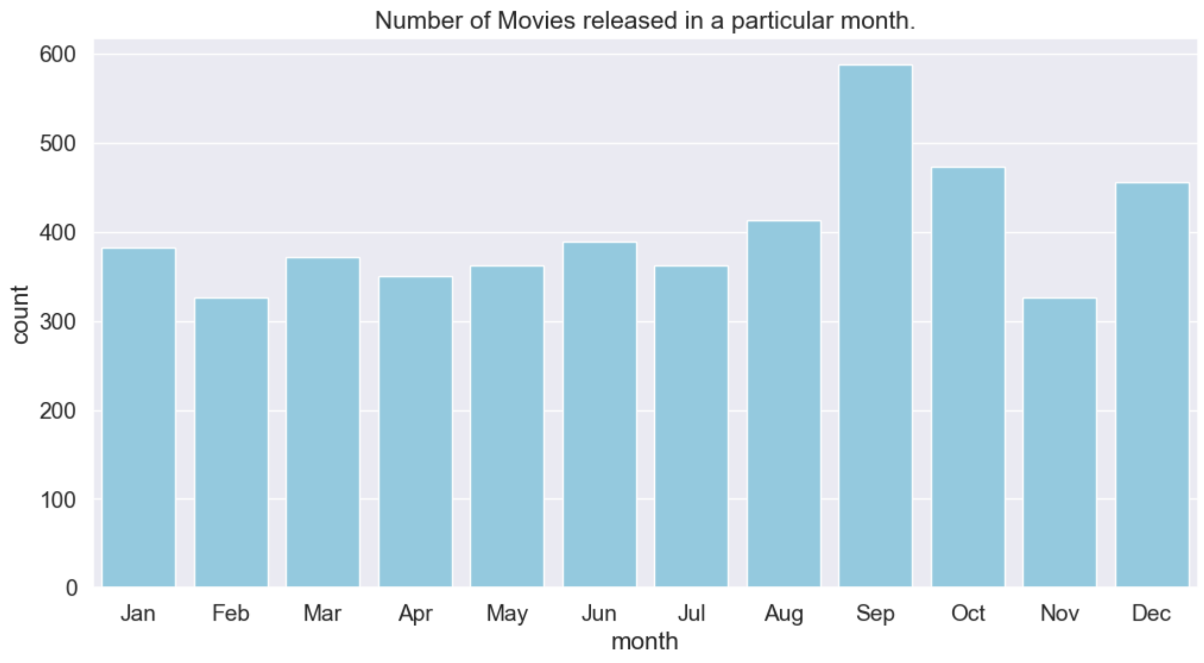

Figure 3.7: Movie count per each month

From the above graph 3.7, it can be depicted that there are significant number of movies released every month. However, notably, during the month of September, October and December, the count of movies released is notably higher. This surge could be attributed to the holiday season, a period when families often find more free time to engage themselves in watching new movies. The release of movies is influenced by various factors, including audience trends and seasonal preference for specific periods or holidays.The variation in the number of releases each month might be attributed to movie studios strategically planning their releases based on anticipated viewership or to avoid competition from other new movies. The relatively lower count in November and February might signify strategic lulls, potentially avoiding competitive release windows or focusing on more quality-driven releases during these periods.This observation adds a layer of complexity to the understanding of the movie release calendar, suggesting that industry dynamics and planning play a crucial role in determining when movies are brought to the audience.

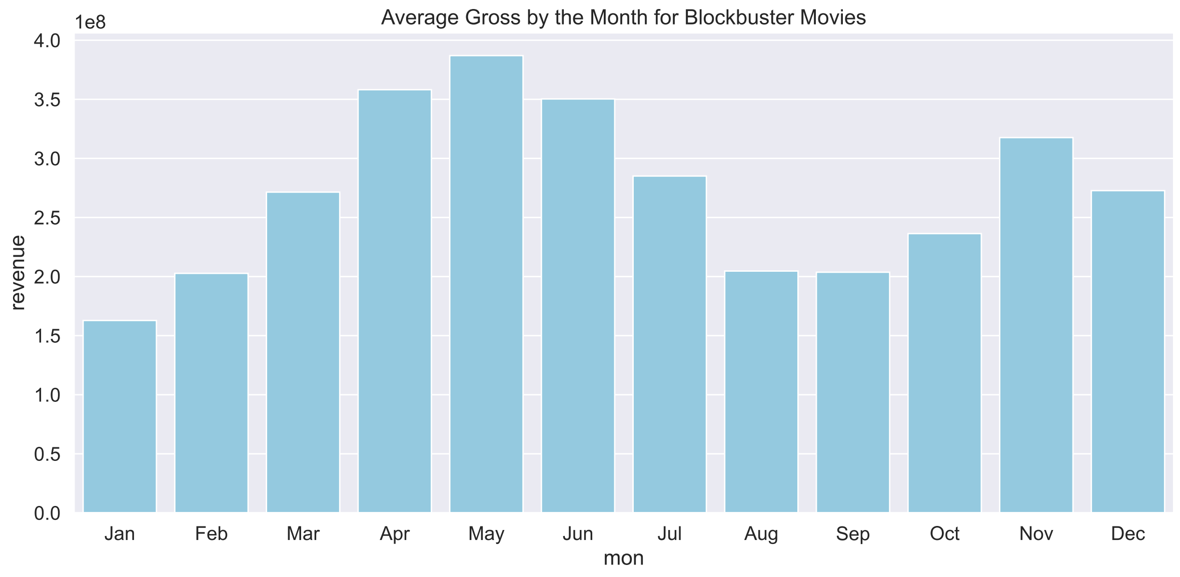

Figure 3.8: Average gross revenue of movies released across different months

The bar plot 3.8 indicates the average gross revenue for the blockbuster movies across different release months, providing a valuable insights into the strategic planning and revenue potential within the film industry. It is evident that certain months exhibit higher average gross revenues, indicating optimal release windows for maximizing profits. It can be said that movies released during the months May, June and July tend to generate higher average revenue compared to those released earlier or later in the year. One possible reason for this could be the release timing coinciding with peak movie-going seasons like summer vacations, leading to increased audience turnout and thus higher revenue.These months are often favored for big blockbuster releases due to more people being available to watch movies during this time. It is noteworthy that even though fewer movies were released during the month of November, these achieved substantial profits, possibly due to a reduced number of options available prior to the holiday season. Conversely, January and February, while starting of the year on a positive note , exhibit a slightly lower average revenue, this might be due to post-holiday consideration.This comparative analysis underscores the importance of strategic release planning, considering audience preferences and seasonal trends to optimize revenue potential in the film industry.

The table 3.8 showcases top 20 prolific actors along with the count of movies in which they have appeared. "Robert De Niro" leads the list with an impressive 46 movie credits,

Table 3.8: Actors with most appearances

| cast | No. of Movies Acted | cast | No. of Movies Acted |
|---|---|---|---|
| Robert De Niro | 46 | Tom Cruise | 28 |
| Bruce Willis | 35 | Mark Wahlberg | 28 |
| Samuel L. Jackson | 35 | Ben Stiller | 28 |
| Matt Damon | 35 | Eddie Murphy | 28 |
| Nicolas Cage | 34 | John Travolta | 27 |
| Johnny Depp | 32 | Julia Roberts | 27 |
| Brad Pitt | 30 | Harrison Ford | 27 |
| Tom Hanks | 29 | George Clooney | 27 |
| Morgan Freeman | 29 | Owen Wilson | 26 |
| Denzel Washington | 29 | Ewan McGregor | 26 |

showcasing his extensive and diverse body of work in the film industry. Following closely are "Bruce Willis","Matt Damon" and "Samuel L. Jackson", each with 35 movie credits, indicating their significant contributions to cinema. The table highlights a roster of accomplished actors, including , Nicolas Cage, Johnny Depp, Brad Pitt , all of whom have amassed a substantial number of movie credits, reflecting their enduring presence in the industry. This list represents a selection of highly sought-after actors known for their versatility and skill in bringing characters to life on screen. Their extensive filmographies underscore their impact and influence within the realm of cinema, showcasing the enduring legacy of these accomplished actors.

The table 3.9 enumerates accomplished film directors alongside the count of movies they have directed, offering insight into their prolificacy and influence within the film industry. "Steven Spielberg" stands out as the most prolific director in this list, having directed an impressive 27 movies. This extensive filmography is a testament to Spielberg's enduring impact on cinema and his versatility across various genres. Following closely are "Martin Scorsese" and "Woody Allen", each with 21 and 22 directed films, respectively. These directors have made significant contributions to the art of filmmaking, known for their distinct styles and storytelling . Noteworthy names like "Clint Eastwood", "Ridley Scott", and "Spike Lee" also feature prominently on the list,

Table 3.9: Directors with most Movies

| Director | Movies Directed |
|---|---|
| Steven Spielberg | 27 |
| Woody Allen | 22 |
| Martin Scorsese | 21 |
| Clint Eastwood | 20 |
| Robert Rodriguez | 17 |
| Spike Lee | 16 |
| Ridley Scott | 16 |
| Steven Soderbergh | 15 |
| Renny Harlin | 15 |
| Oliver Stone | 14 |
| Tim Burton | 14 |
| Ron Howard | 13 |
| Joel Schumacher | 13 |
| Barry Levinson | 13 |
| Robert Zemeckis | 13 |
| Francis Ford Coppola | 12 |
| Tony Scott | 12 |
| Michael Bay | 12 |
| Joel Coen | 12 |
| Brian De Palma | 12 |

all having directed a substantial number of films, reflecting their enduring presence and influence in the industry. This table highlights a group of highly respected directors known for their prolificacy and artistry, collectively shaping the landscape of modern cinema.This simply shows how much these directors have influenced and added to the diversity of the film industry.

### 3.3.3 Demographic based filtering

Demographic-based filtering in movie recommendation systems uses information like age, gender, and location to suggest movies that are popular within specific groups. It recommends films similar to those favored by people of a particular gender, age range, or region. This means that users with similar demographics receive comparable movie suggestions.However, this method has its limitations. It assumes that people with similar demographics share similar tastes, which might not always be true. It doesn't consider individual preferences when suggesting movies. Demographic-based filtering mainly suggests popular movies within specific demographics [40].

This model sorts the movies based on the rating metric and it gives the top movies from the sorted list. Using the average rating alone may not yield accurate results. [17] For example, a movie with a rating of 8.3 but only 5 user ratings might not be as reliable as a movie with a rating of 7.4 but rated by 30 users. Hence, the sorting is done on the weighted rating metric. It can be defined as

$$\text{Weighted rating } (WR) = (\frac{v}{v+w}.r) + (\frac{m}{v+m}.c) \tag{3.1}$$

Where

$v$ - Total number of received for the movie

$m$ - Minimum number of votes required

$r$ - average rating of the movie

$c$ - mean of votes across the complete report

$v$ (vote _count) and $r$(vote _average) are provided in the dataset

$c$ and $m$ need to be calculated

$c$ is defined as the mean of the vote _average ( average rating)

$m$ value is computed at the 90% quantile of vote _count

The mean rating for all the movies in the dataset is 6 on a scale of 10, suggesting that most movies received relatively positive ratings. The value of $'m'$ which serves as a filtering metric, is approximately 1838.4, representing the 90th percentile of the vote count. This means that only movies with a vote count higher than 1838.4 are taken into account when calculating the Weighted Rating metric. Essentially, this approach focuses on movies with a substantial number of votes, ensuring that the weighted rating is based on a reliable and significant audience response rather than considering movies with minimal engagement

**Limitations of Demographic-based Filtering:**

- ***Lack of personalization:*** Demographic based filtering recommends items based only on the overall popularity, it doesn't consider the preferences of each user[60].

- ***Vulnerable to manipulation:*** These recommendation system can be susceptible to manipulation, as actions like fake reviews or artificially increasing the popularity of an item can distort the recommendations.

- ***Quality vs Popularity:*** Popularity doesn't necessarily correlate with the quality or relevance of an item to individual user. Users may prefer high-quality or personalized recommendations over popular ones[61].

### 3.3.4 Content-Based Filtering:

Content based filtering tries to suggest items to the user based on the user's profile. It uses the contents of the dataset to filter the movies, such as keywords, genres, cast, crew and overview (description of the movie). [24]This filtering method creates two profiles [41],

1. User profile
2. Item profile

**User profile:** It represents the user preferences. It is built on the user's previous watch history, likes, dislikes, ratings and engagement[63].

**Item Profile:** Each item is defined by a set of features or attributes. These features may include genres, actors, directors, and overview[63].

In this project two types of content-based recommendation systems are used.

**1. Description based recommendation:** This approach recommends items based on the textual description or overview of the items. It leverages natural language processing and similarity measures to suggest items with similar descriptions.

**2. Metadata based recommendation:** This approach recommends items based on metadata such as genres, actors, directors, and other features associated with the items. It focuses on the inherent characteristics of the items rather than their textual descriptions.

Following are the steps involved in developing a content based recommendations.

1. Converting texts to numeric
2. Calculate similarities

**Converting texts to numeric:**

In the given dataset, the data is in the form of text. To develop a recommendation system, the contents of the dataset need to be vectorized, i.e., converted from text to numeric format for mathematical computations. Various methods can be employed for this vectorization process, such as:

1. TF-IDF vector
2. Count vector

- **TF-IDF Vectorization**

  TF-IDF stands for Term frequency- Inverse Document Frequency, is a numerical statistic used in natural language processing and information retrieval to evaluate the importance of a term (word or phrase) within a document or a collection of documents (corpus)[25].

- **Term Frequency (TF):** TF measures how frequently a term appears in the document. [19]It is mathematically define as the number of times a term appears in a document divided by the total number of terms in the document. TF considers that the more a term appears in the document the more important the term is.

$$\text{TF}(t,d) = \frac{\text{Number of times term } t \text{ appears in document } d}{\text{Total number of terms in the document } d} \tag{3.2}$$

- **Inverse Document Frequency (IDF):** IDF measures the importance of the term in the entire corpus. It is calculated as a logarithm of the total number of documents in the corpus divided by the number of documents that contain the term. The IDF value is higher for terms that are rare across the corpus and lower for terms that are common.[42]

$$\text{IDF}(t) = \log_e \left( \frac{\text{Total number of documents}}{\text{Number of documents with term } t \text{ in it}} \right) \tag{3.3}$$

- **TF-IDF Score:** The TF-IDF score for a term in a document combines the TF and IDF components to assess the importance of the term within that document and corpus. It is computed by multiplying the TF and IDF values for the term.

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) * \text{IDF}(t) \tag{3.4}$$

The TF-IDF score is higher for terms that are frequent with in a specific document and rare across the entire corpus. This means that terms that are unique to a particular document or highly discriminative are given more weight, while common terms that appear in many documents are down weighted.

- **Count vectorization**

It is the process of converting the text into numeric representation by counting the frequency of each word in the document[26]. It is also known as the "term frequency vectors".This process involves following steps:

– **Tokenization:** The first step is tokenization,where the text document is divided into individual words or terms. This process of involves breaking down the document into its constituent elements.

– **Counting Terms:** After tokenization, each term in the document is counted and stored in the corresponding position in the count vector. The count reflects how many times each term appears in the document.This results the frequency of each unique term.

– **Vectorization:** It is a numerical array, where each element corresponds to a unique term from the document and the value in each element represents the count of that term in that document.Vectorization representation provides a concise and quantitative description of the document's content.

These vectors consider the presences of the word and the frequency of that word in a document. The count vectors cannot consider the relative importance of terms, which is considered in TF-IDF vectorization.

In this report the TF-IDF vectorizer is used for the overview-based content filtering, and the count vector is used for metadata content-based filtering. This is because the TF-IDF excels in evaluating the importance of terms within textual description, whereas the count vectorization is suitable for the structured and categorical attributes where the focus is to evaluate the presences or absence of the term rather than the weighted importance.

Further converting the text into vectors, a similarity matrix is built on this data to generate recommendations.

**Computing similarity matrix:** Computing similarity matrix involves evaluating the similarity or closeness between pairs of items in a dataset and organizing the results into a matrix format. This matrix typically represents the pairwise similarity between items.

**Vector space:** This vector space is used to compute the similarity between items, between user and item, and between user[62]. The vector space is an n-dimensional space where dimensions are based on the number of terms in the document as extracted in the vectorization.[16] Each point in a vector space is represented by an n-dimensional vector. The similarity between these items is computed either using angle or the distance between these vectors. There are various methods to compute the similarity between two vectors, but cosine similarity is most used as it uses the angle between the two vectors to measure the similarity.

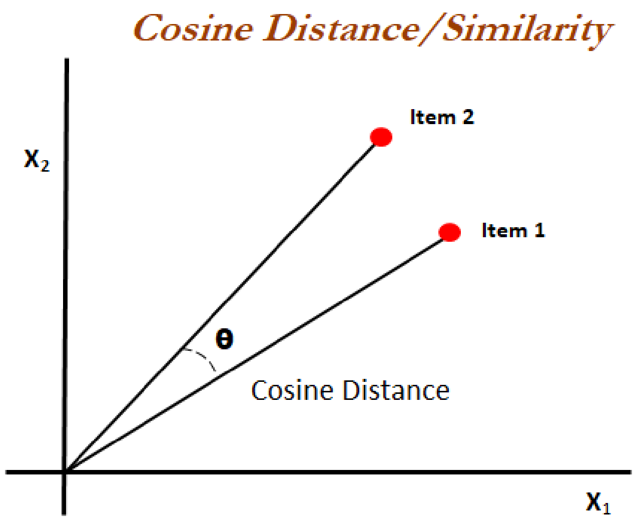

Figure 3.9: Cosine Similarity

Cosine Similarity

**Cosine Similarity:** It is a metric used to measure the similarity between two vectors in a multi-dimensional space. It computes the cosine angle between the two vectors, which measures the directional similarity and likeliness between the features of two vectors. The figure 3.9 shows the pictorial representation of the cosine similarity evaluation, in a vector space two items are mapped and the cosine similarity is computed by evaluating the angle between the two vectors. [27]

$$\text{Cosine similarity}(A, B) = \frac{A.B}{\|A\|\|B\|} \tag{3.5}$$

Where A.B represents the dot product between the two vectors A and B, $\|A\|$ and $\|B\|$ represents the Euclidean norms of A and B.

$$\|A\| = \sqrt{\sum_{i=1}^{n} A_i^2} \tag{3.6}$$

$$\|B\| = \sqrt{\sum_{i=1}^{n} B_i^2} \tag{3.7}$$

$$A.B = \sum_{i=1}^{n} A_i B_i \tag{3.8}$$

The cosine similarity score ranges from -1 to 1, -1 indicating perfect dissimilarity, 1 indicates perfect similarity and 0 indicates no similarity[64]. The higher the cosine score the greater the similarity between vectors.

$$\text{Similarity}(I_i, I_j) = \frac{\sum_{u \in U_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{u \in U_{ij}} r_{ui}^2} \cdot \sqrt{\sum_{u \in U_{ij}} r_{uj}^2}} \tag{3.9}$$

$I_i$ and $I_j$ are vectors representing the preferences of users for items $i$ and $j,$ respectively.

$r_{ui}$ and $r_{uj}$ are the ratings or interactions of user $u$ with items $i$ and $j$, respectively.

$\sum_{u \in U_{ij}}$ is over all users who have interacted with both items $i$ and $j$

The formula calculates the cosine similarity between the vectors $I_i$ and $I_j$ indicating how similar the preferences of users are between items $i$ and $j$

**Developing an Overview content-based recommendation systems:**

The following are the steps involved in developing an overview content based recommendation system.

**1. Tokenization:** It is process of breaking down text into individual words or tokens. Tokens refers to meaningful words or units of text. In the given dataset ,the 'overview' column is in the form of string (text). Tokenization enables the conversion of textual descriptions into a structured, analysable data.

**2. Stop Words:** Removing stop words is the most significant step in the Natural language processing and text analysis. Stop words are a set of commonly used words, such as 'a', 'an', 'the', 'is','of' etc. that occur more frequently in the language but don't carry any significant meaning. Thus, it is important to remove stop words, as it aids in noise reduction, reduced dimensionality and improved efficiency.

**3. Stemming:** It is a process of reducing words to their base or root form. The primary purpose of stemming is to simplify and standardize word variations, such as different verb tenses, plurals, or other variations, into a common form. For stemming, the nltk package is installed where the PorterStemmer is loaded.

**4. Vectorization:** After stemming the 'overview' column, TF-IDF vectorization is used to convert the text into vectors mapped in an n-dimensional vector space. The TF-IDF vectorizer is extracted from the sklearn.feature_extraction.text library.

$$\text{TF-IDF Vector}_d = [\text{TF-IDF}_{t_1,d}, \text{TF-IDF}_{t_2,d}, \ldots, \text{TF-IDF}_{t_n,d}] \tag{3.10}$$

The TF-IDF vector for a document is a vector where each element corresponds to the TF-IDF score of a term in the document.

**5. Cosine Similarity:** Cosine similarity is applied on these vectors to evaluate the similarity between the vectors in the vector space. To apply cosine similarity linear _kernel is imported from sklearn.metrics.pairwise

**6.Splitting of dataset**:The given dataset is sampled randomly into training and testing in the ratio of 80:20.

**7. Developing a description-based recommendation method:** A function is defined that takes the title (the title of the movie on which the movies are suggested) and similarity matrix as its arguments. This function takes the title input, finds its index in the given dataset, and further computes the cosine similarity between this movie and all the other movies in the given dataset. It returns the top 10 movies that are similar to the given title.

**8. Evaluating the model performance:** The filtering model is trained using the train dataset and is evaluated using the test dataset. RMSE and MAE are the two evaluation metrics used for determining the model performance

**Developing a Metadata content-based recommendation systems:**

The following are the steps involved in developing a metadata content-based recommendation system.

**1. Combining the metadata columns:** The columns cast(top 3 casts), crew(director), keywords(top 3 keywords) and genres are joined to form a single string. While extracting the cast and crew, the space between the names is removed because during similarity metric evaluation, if space is present between the names, the model considers it as two different names. It is necessary to remove the space between names.

**2. Vectorization:** After combining the data of four columns, this data is converted into vectors mapped in a vector space. The count vectorizer is used to convert the text into vectors for the metadata, as only the presence is significant rather than the weightage of the term in document.

**3. Similarity:** Cosine similarity matrix is applied to these vectors to compute the similarity between the metadata of movies.

**4. Developing a metadata content-based recommendation function:** A function is defined which takes title and similarity as input. This return the top 10 movies that match the metadata of the given movie title.

**5. Evaluating the model performance:** The train dataset is used to train this metadata based filtering model. RMSE and MAE are used to evaluate the model performance using the test dataset.

**Limitations of content-based movie recommendation systems:**

- ***Cold start problem:*** Content-based filtering faces challenges when dealing with new user, who doesn't have any watch history or prior preferences.[48]

- ***Limited Diversity:*** This recommendation system tends to recommend items that are similar to those the user has already watched, it doesn't show any diverse movies which doesn't match with user watch history.[47]

- ***Scalability Issues:*** As the number of items in the system grows, the scalability of content-baased filtering can become an issue. Extracting and processing features for a large number of items can be computationally expensive.[47]

### 3.3.5 Collaborative based recommendation systems:

Collaborative filtering is a technique used in the recommendation systems to predict user interests or preferences for items based on the behaviour and preferences of a group of users. It relies on the idea that users who have interacted with or rated items similarly in the past are likely to have similar tastes. It doesn't consider personal interests and doesn't provide suggestions based on genres, cast, crew, overview. It receives feed back from the user in two ways:[12]

1. Implicit feedback
2. Explicit feedback

**Implicit feedback:** Implicit feedback is more indirect and is derived from the user behavior. It includes actions like ratings, watch history, engagement with a particular movie.

**Explicit Feedback:** Explicit feedback is direct and explicit from the user, such as ratings, reviews, preferences like (thumbs up or down). User rates a movie on a scale of 1 to 5, where 1 indicates less likelihood and 5 indicates high satisfaction.[43]

There are mainly two types of collaborative based recommendation systems:

1. User based filtering

2. Item based filtering

**User based filtering:** User-based filtering is a technique used in recommendation systems to generate personalized item recommendations for a user based on the preferences and behavior of similar user[65]. Two users are said to be similar if they are interested in similar items/movies. The similarity between users is computed using the cosine similarity. Following is the example to illustrate the working of User based filtering technique. The rows indicates the users and columns indicates the movies.

Mathematically user-based filtering can be expressed as[67]:

$$S(u,i) = \overline{r_u} + \frac{\sum\limits_{v \in U} (r_{vi} - \overline{r_v}) \cdot w_{uv}}{\sum\limits_{v \in U} w_{uv}} \tag{3.11}$$

where $w_{uv}$ is cosine similarity

$$w_{uv} = \frac{\sum\limits_{i \in I} (r_{ui} - \overline{r_u})(r_{vi} - \overline{r_v})}{\sqrt{\sum (r_{vi} - \overline{r_v})^2} \sqrt{\sum (r_{vi} - \overline{r_v})^2}} \tag{3.12}$$

where

$S(u,i)$ is the estimated rating of user u given to item i

$\overline{r_u}$ is the user u average rating

$r_{vi}$ is the user v rating on item i

$\overline{r_v}$ is the user v average rating

$r_{ui}$ is the user u rating on item i

Table 3.10: User based filtering

| Users/movies | M1 | M2 | M3 | M4 |
|---|---|---|---|---|
| A | liked | | liked | recommended |
| B | | liked | | |
| C | liked | | liked | liked |

The table 3.10 is the example of user-based Collaborative filtering.From the table it can be interpreted that, the users A and C are alike as they have liked similar movies ( C might be the neighbor of A), thus this algorithm suggests the movies watched by C to A, thus algorithm suggests M3 (movie 3) to the user A.

for m users and n items, the big O notation for evaluating the pairwise cosine similarities is $O(m^2n)$ [67]

**Item Based filtering:** Item-Based Filtering is a method in recommendation systems that gives users personalized suggestions based on item similarities. It assumes that users will probably like items similar to those they enjoyed before. The idea is that if two items are similar in terms of user interactions or characteristics, they are likely to interest the same user. Cosine similarity is used to measure how alike these items are.[44]

Table 3.11: Item based filtering

| Users/movies | M1 | M2 | M3 |
|---|---|---|---|
| A | liked | | liked |
| B | liked | liked | liked |
| C | liked | | recommended |

Table 3.11 is an example of item-based collaborative filtering approach. It can be interpreted that movie 1 and movie 3 are similar,i.e users who likes movie 1 also liked movie 3. If user C likes movie 1 then he might also like movie 3, so this item-based filtering model suggests movie 3 to the user.

$$\text{rating}(U, I_i) = \frac{\sum \text{rating}(U, I_j) \times \text{similarity}_{i,j}}{\sum \text{similarity}_{i,j}} \tag{3.13}$$

where rating$(U, I_i)$ is the rating of user u given rating to item i similarity$_{i,j}$ is the similarity between the item i and item j [66]

Item-based filtering does not require detailed information about users; instead, it mainly focuses on item similarity. It is suitable for scenarios with a large number of items and relatively stable users. However, there are significant limitations to item-based filtering, including:[45]

**Limitations of item-based filtering:**

- ***Cold-Start Problem:***
  - This occurs for new items or items with few interactions.[69]
  - Recommending items with little to no historical data can be challenging.
- ***Scalability:***
  - As the number of items increases, the computation of item similarity becomes more resource-intensive.[68]
- ***Sparsity:***
  - In a user-item interaction matrix, the data is often sparse, with many missing values.
  - Sparse data can affect the accuracy and performance of item-based filtering.[69]

To address these limitations, matrix factorization techniques like Singular Value Decomposition (SVD) can be employed. SVD helps in decomposing the user-item interaction matrix into lower-dimensional matrices, capturing latent factors that contribute to user preferences. This approach enhances the model's ability to make accurate recommendations, even in scenarios with sparse data or new items.

**Matrix Factorization:**

Matrix factorization is a collaborative filtering technique used to decompose a user-item interaction matrix into lower-dimensional matrices[70]. This process aims to better understand the underlying patterns and relationships between users and items. Matrix factorization is widely applied in recommendation systems to enhance prediction accuracy and generate more personalized recommendations.

Consider the table below, where rows represent users and columns represent movies. It's important to note that not every user rates every movie, and vice versa (not every movie is watched by every user), leading to sparsity in the matrix. In matrix factorization, null values in the matrix are typically filled with the value 0. This factorization helps reveal latent factors that contribute to user preferences and item characteristics, enabling the system to make more accurate predictions and recommendations.

Table 3.12: User Ratings for Movies

| | **Movie 1** | **Movie 2** | **Movie 3** |
|---|---|---|---|
| **User 1** | 5 | 0 | 3 |
| **User 2** | 0 | 4 | 0 |
| **User 3** | 2 | 0 | 1 |

In the matrix above 3.12, the zeros represent missing ratings or interactions. Matrix factorization helps fill these gaps by decomposing the original matrix into matrices that capture latent features[18], making it possible to estimate missing values more accurately.

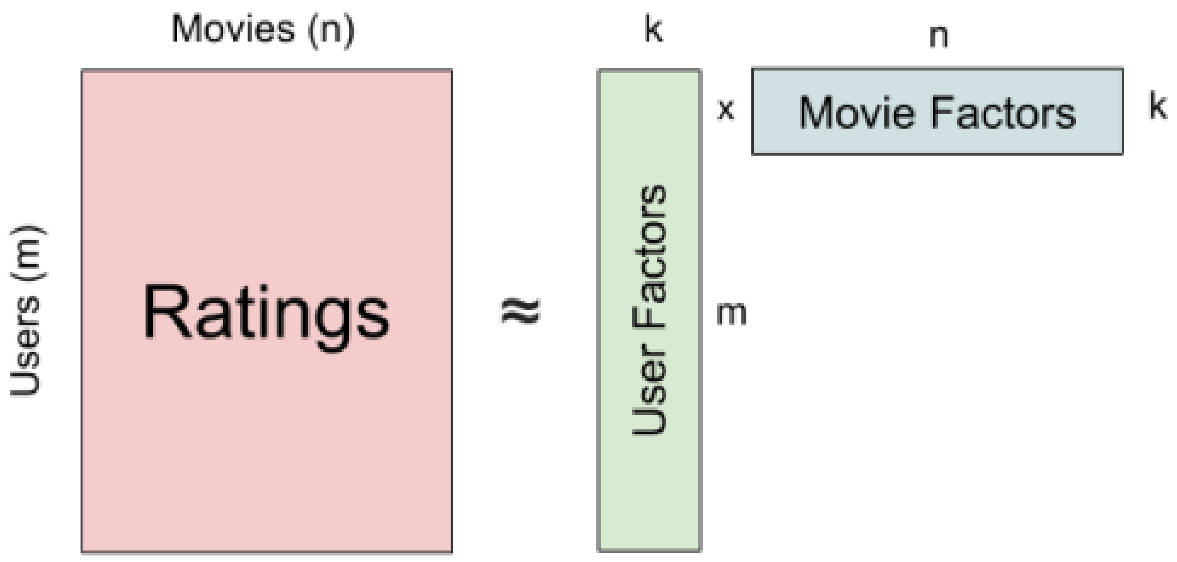

Figure 3.10: Matrix Factorisation

Matrix Factorization

$$\text{Rating Matrix } (R) = \text{User Matrix } (U) \times \text{Movie Feature Matrix } (M)$$

The figure 3.10 is the pictorial representation of the matrix factorization. Rating matrix(R) is constructing using the user matrix(U) and the movie feature matrix(M).

Each matrix containing k latent factors. Latent factors refer to features such as genres, plot, actors, and directors. The rating matrix is obtained by taking the dot product of these two matrices. Let the size of the user matrix be m, the size of the movie matrix be n, and the dimensions of the rating matrix be mxn, where k is always less than the size of both m and n. These latent factors serve as vector components, with each component carrying a significant weight. It is evident that the dimensionality of the matrix has increased. One method to reduce dimensionality without losing data is by performing principal component analysis. There are several models for dimensionality reduction, including:

1. Singular Value Decomposition
2. Probability Matrix Factorization
3. Non-Negative Matrix Factorization

**Singular value decomposition:**

Singular Value Decomposition (SVD) is a technique used in recommendation systems to provide personalized recommendations to users. It helps in reducing the dimensionality of a matrix by decomposing it into three separate matrices. SVD is particularly valuable for extracting underlying patterns or features from complex data.[46] It aids in identifying latent factors that represent user preferences and item characteristics. By decomposing the user-item interaction matrix, it enables the generation of personalized recommendations

Mathematically it can be represented as

$$A = U \sum V^t \tag{3.14}$$

Where

A - matrix of size mxn

U - nxm orthogonal matrix containing the left singular vectors

$\sum$ - mxn diagonal matrix containing the singular values in decreasing order

V - nxn orthogonal matrix containing the right singular vectors

$V^t$ - denotes the transpose of matrix V

The singular values in matrix $\sum$ are non-negative and represents the importance of each latent factor. By selecting top j singular values and their corresponding vectors, the dimensionality of the matrices can be reduced, enabling the approximation of the original matrix( R). A is the approximation matrix of R , the difference between the A and R is the error which can be calculated by RMSE and MAE.

**Theorem 3.1.** *Singular Value Decomposition (SVD):[31]*

*A matrix A of dimensions $m \times n$ can be expressed in a decomposed form as follows:*

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T = U\tilde{S}V^T, \quad \tilde{S} := \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix}$$

*where U and V are orthogonal matrices of size $m \times n$ and $n \times m$, where as $S$ is a diagonal matrix represented as*

$$S = diag(\sigma_1, \ldots, \sigma_r)$$

*where, the values in diagonal matrix $\sigma_1 \geq \ldots \geq \sigma_r > 0$ are unique, and are called singular values of matrix A.r is the rank of matrix A and $r \leq \min(m, n)$. $u_i$ for $i = 1, \ldots, r$ and $v_i$ for $i = 1, \ldots, r$ denotes the first r columns of matrix U and V respectively, these are called left and right singular vectors of A respectively, and satisfy:*

$$Av_i = \sigma_i u_i, \quad u_i^T A = \sigma_i v_i, \quad i = 1, \ldots, r$$

***Proof***

*The given matrix A is a real and symmetric matrix , denoted as $A^T A$. By spectral theorem, it can be said that the eigenvalue decomposition is $A^T A = V \Lambda V^T$, where V is an real value orthogonal matrix of size $n \times n$ and $\lambda$ be a diagonal matrix represented as t $\Lambda = diag(\lambda_1, \ldots, \lambda_r, 0, \ldots, 0)$.*

*The values of $\lambda_j$'s are non-negative ,as $A^T A$ is a semi-definite positive matrix where the singular values can be expressed as $\sigma_j = \sqrt{\lambda_j}$ for $j = 1, \ldots, r$.*

*Remark* 3.2. when $j > r$, $Av_j = 0$ since $|Av_j|_2^2 = v_j^T A^T A v_j = \lambda_j v_j^T v_j = 0$

*construction of $m \times m$ orthogonal matrix U is as follows:*

$$u_i = \frac{1}{\sigma_i} A v_i, \quad i = 1, \ldots, r$$

*since $v_j$'s are eigenvectors of $A^T A$ , these $m$ vectors are orthogonal to each other and of unit norm.To form an orthogonal matrix $U = (u_1, \ldots, u_m)$,these vectors are represented as $u_{r+1}, \ldots, u_m$ ,by the Gram-Schmidt orthogonalization procedure,* *The above formula shows that* $U^T AV = \tilde{S} = diag(\sigma_1, \ldots, \sigma_r, 0, \ldots, 0)$

$$(U^T AV)_{ij} = u_i^T Av_j = \begin{cases} \sigma_j u_i^T u_j & \textit{if } j \leq r \\ 0 & \textit{otherwise} \end{cases}$$

*Thus, it is proved that* $U^T AV = \tilde{S}$

**Implementation of the collaborative filtering:**

Following are the steps involved in implementing the svd algorithm on the collaborative filtering technique.

**1. Load and clean the data:** as the previous dataset doesn't give any sort of the user information and their respective rating, a new dataset of ratings is loaded. Further, loading the data is cleaned, removal of null values and noise. The dataset contains user_ID , movie id, ratings columns.

**2. Splitting the dataset:** The given dataset is divided into two parts testing and training dataset in the ratio 25:75, using the train_test _split method from surprise library model_selection . The data is randomly sampled into test dataset(25%) and train dataset(75%).

**3. Algorithm:** SVD algorithm is imported from the surprise library. Singular value decomposition algorithm is widely used technique in collaborative filtering. It decomposes the user-item interaction matrix into lower-dimensional matrices, enabling us to make personalized recommendations based on these embedding.

The SVD algorithm is applied to the user-item matrix, the sparse rating ($\hat{r}_{ui}$) of user $u$ to movie item $i$ is given by :

$$\hat{r}_{ui} = b_{ui} + \mathbf{p}_u^T \mathbf{q}_i \tag{3.15}$$

where $\mathbf{p}_u \in \mathbb{R}^k$ is the user's latent factor vector. and

$b_{ui}$ reprsents the baseline estimate for an unknowing rating $r_{ui}$ of user $u$ to movie $i$, define as

$$\hat{b}_{ui} = \mu + b_u + b_i \tag{3.16}$$

The computation complexity of the SVD algorithm is assessed in batches, equating to $O(m^2 n + n^3)$ (where m and n are the row and column sizes of the matrix).Since SVD necessitates processing all data simultaneously, it encounters challenges not only with large datasets but also with computationally expensive operations when incorporating new users or items into the system.

**4. Training the model:** The model is trained over the train dataset. This process involves optimizing the internal parameters of the model to best fit the observed user-item interactions. Training is crucial for the model to make accurate predictions.

**5. Similarity Computation:** The similarity between the users interaction or between the items is computed using the cosine similarity.

$$\text{Similarity}(U_u, I_i) = \frac{U_u \cdot I_i}{\|U_u\| \cdot \|I_i\|} \tag{3.17}$$

where, $U_u \cdot I_i$ denotes the dot product of vectors $U_u$ and $I_i$,

$\|U_u\|$ and $\|I_i\|$ represent the Euclidean norms of the vectors $U_u$ and $I_i$ respectively.

**6. Predictions:** The model is trained, it is used to predict for the items in the test dataset. These predictions are based on the patterns learned during the training. This step simulates how well the model would perform in real-world scenario, where actual ratings are not known in advance.

$$r_{ui} = \frac{\sum_{j \in N(u)} \text{sim}(i,j) r_{uj}}{\sum_{j \in N(u)} \text{sim}(i,j)} \tag{3.18}$$

where $N(u)$ is teh set of items that user $u$ has rated, $sim(i,j)$ is the simlarity score between items $i$ and $j$, and $r_{uj}$ is the rating of item $j$ by user $u$

**7. Evaluating Model Performance :** Root mean square error and Mean Absolute Error, is used to evaluate the model performance. This metric measures the average maginitude of predictions errors. A lower RMSE, MAE indicates better predictive accuracy, meaning our model provides more accurate recommendations.

### 3.3.6 Hybrid Filtering Recommendation systems:

Hybrid recommendation filtering combines multiple recommendation techniques to improve the accuracy and effectiveness of recommendations. It leverages the strengths of different approaches[52], such as collaborative filtering, content-based filtering to create a more comprehensive and personalized recommendation system.[28]

The main aim of the hybrid filtering is to overcome the limitations of individual methods. For instance, collaborative filtering struggle with the cold-start problem(when a new user or item has little or no interaction history), while the content-based filtering might not capture the users preferences or trends effectively. By integrating these techniques, hybrid filtering aims to provide more accurate, diverse, and contextually relevant recommendations.

**Implementation of hybrid recommendation system:**

Following are the steps involved in implementing the hybrid filtering movie recommendation systems:

**1. Data Preparation:** The dataset was processed and divided into training and testing sub datasets randomly in a ration of 80:20.

**2. Collaborative Filtering (Singular Value Decomposition - SVD):** SVD algorithm is used to model the user-item interactions to predict user preferences. This model is trained by using the training dataset.

**3. Content-Based Filtering (TF-IDF and Cosine Similarity):** TF-IDF vectorizer is employed to preprocess movie description and extract features into vectors. Computes the cosine similarity between movies based on their content features to measure similarities between movies.

**4. Hybrid Model Integration:** Collaborative and content-based models are integrated to develop a hybrid model. It generates recommendations by integrating predictions from both the models using weighted averaging strategy.The code identifies the top similar movies based on content similarity and predicts ratings using collaborative filtering.

**5. Model Evaluation:**Evaluates the hybrid model's performance using evaluation metrics such as RMSE (Root Mean Square Error) and MAE (Mean Absolute Error) on the test dataset.Conducts comparative analysis against individual collaborative and content-based models to assess the hybrid model's superiority.

**Limitations of Hybrid based recommendation systems**

- **Complexity:** Hybrid recommendation systems are complex to design, implement and maintain compared to individual recommendation approaches.

- **Dependency on Component Models:** The overall performance of the hybrid model depends on the content and collaborative. If one component model doesn't perform well, then the overall hybrid performance decreases.

## 3.4 Evaluation Metrics

Evaluation metrics play a significant role in assessing the performance and effectiveness of recommendation systems. Following are the various techniques used to evaluate a model performance.

- **Root Mean Square Error(RMSE):** RMSE is a commonly used metric in evaluating the performance of the recommendation systems. It measures the average magnitude of the differences between predicted and actual values. The name itself says that, it computes the square root of the mean of squared differences.[29]

  The following formula is used to compute the RMSE:

  $$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{3.19}$$

  where

  $n$- represents the number of observations

  $y_i$- denotes the actual values

  $\hat{y}_i$ - represents the predicted values

  **Steps to evaluate RMSE:**

  1. calculate the difference between each predicted and actual value.

  2. Square each of these differences to eliminate the effects of negative errors.

  3. Find the average(mean) of these squared differences.

  4. Take the square root of the average to obtain the final RMSE value

  The lower the RMSE value the better the performance of the model.

- **Mean Absolute Error(MAE):**MAE is a metric used to evaluate the accuracy of a predictive model by measuring the average absolute differences between the actual and predicted values.[30]

  The following formula is used to compute the MAE:

  $$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{3.20}$$

where

$n$- represents the number of observations

$y_i$- denotes the actual values

$\hat{y}_i$ - represents the predicted values

**Steps to evaluate MAE:**

1. calculate the absolute difference between each predicted and actual value.

2. Find the average(mean) of these absolute differences.

MAE provides a straightforward measure of the average magnitude of errors between predicted and actual values, with the errors being expressed as absolute differences. A lower MAE indicates better model performance, as it signifies smaller average prediction errors.

CHAPTER 4

# Results

Following are the results yielded by implementing various filtering techniques.

**Demographic Filtering:**

The analysis of demographic filtering based on rating and popularity is presented through two graphs, each offering insights into the trending movies using different metrics.

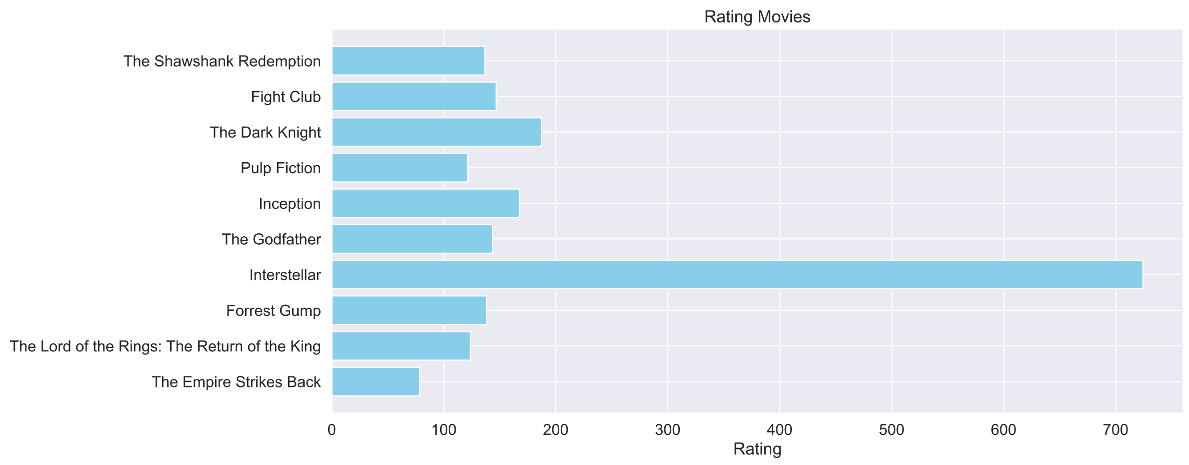

Figure 4.1: Trending movies based on Rating

Figure 4.1 showcases the top 10 movies based on the Weighted rating score metric, a composite measure likely considering factors such as vote count and vote average. This approach takes into account both the quantity and quality of votes, revealing that a high movie rating does not necessarily translate to a high score. The score metric offers a nuanced evaluation, reflecting a more comprehensive assessment of a movie's appeal. Movies like "The Shawshank Redemption", "Fight Club" and "The Dark Knight" emerge as top-ranking movies, indicating a blend of exceptional ratings and significant audience participation.

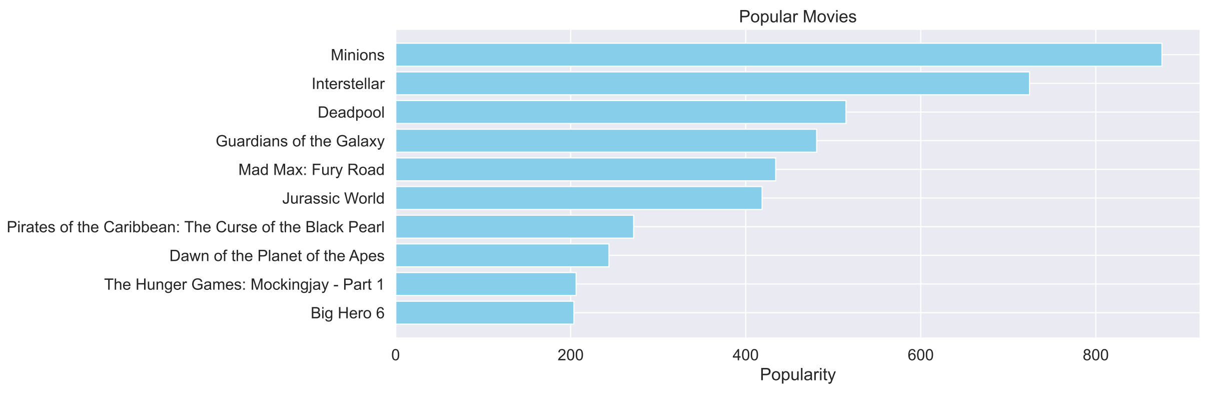

Figure 4.2: Trending movies based on popularity

On the other hand,Figure 4.2 displays top 10 movies sorted by popularity irrespective of the score. Popularity metric can consider combination of factors, such as, critical acclaim, box office success, and overall cultural impact. This approach emphasises movies that have gained widespread attention and popularity, potentially influenced by marketing efforts, franchise presence, or other external factors. Unlike the score metric, popularity may not exclusively rely on vote count and vote average, contributing to the differences observed in the two graphs.These distinct metrics provide complementary perspectives on movie trends, considering both audience engagement and broader cultural influence.Movies "Minions", "Interstellar" and "Deadpool" have gained immense attention, possibly driven by various factors such as marketing campaigns, widespread interest, or broad audience appeal.

These recommendations are general and based on the demographics of the user, but they do not account for individual interests or tastes. To provide more precise suggestions tailored to a particular user's history, content-based filtering is employed.

**Content-based Filtering:**

The tables( 4.1a and 4.1b) shows the suggestions generated by the content-based filtering. By passing the movie title and the similarity, the function generates movie suggestions based on the overview of the movie. It doesnot take into account the metadata, it suggests solely on the plot similarity of the movies. It illustrates that, for the input 'The Dark Knight Rises' and cosine similarity, the model suggests the similar movie plot items such as , 'The Dark Knight', 'Batman Returns', 'Batman Forever' and soon. Whereas, for the input "Avatar" and cosine similarity, the model suggests the similar movie plot items such as , "Apollo 18", "The Inhabited Island", "The Matrix" and soon.

Table 4.1: Content-based Movie Recommendations

(a) Movie Recommendations similar to the plot of "The Dark Knight Rises"

| Movie ID | Movie Title |
|---|---|
| 155 | The Dark Knight |
| 364 | Batman Returns |
| 414 | Batman Forever |
| 268 | Batman |
| 18777 | Slow Burn |
| 142061 | Batman: The Dark Knight Returns, Part 2 |
| 415 | Batman & Robin |
| 272 | Batman Begins |
| 820 | JFK |
| 209112 | Batman v Superman: Dawn of Justice |

(b) Movie Recommendations similar to the plot of "Avatar"

| Movie ID | Movie Title |
|---|---|
| 50357 | Apollo 18 |
| 16911 | The Inhabited Island |
| 603 | The Matrix |
| 9567 | Tears of the Sun |
| 10384 | Supernova |
| 11692 | The Adventures of Pluto Nash |
| 228326 | The Book of Life |
| 67911 | Bucky Larson: Born to Be a Star |
| 50456 | Hanna |
| 11260 | Meet Dave |

The table 4.2 shows the content based movie recommendations generated on the metadata (director, genres, cast, keywords). The function content _recommendations takes the title of the movie and the similarity as the input. In metadata based, count vectorizer is used instead of tf-idf. By passing the movie name 'The Dark Knight Rises' the model suggests similar metadata movies. It is evident that even though the title of the two methods is same but the output is different because the suggestions generated by metadata and plot are completely different.

Table 4.2: Movie Recommendations similar to the metadata of "The Dark Knight Rises"

| Movie ID | Movie Title |
|---|---|
| 155 | The Dark Knight |
| 272 | Batman Begins |
| 378237 | Amidst the Devil's Wings |
| 1124 | The Prestige |
| 2088 | Romeo Is Bleeding |
| 25941 | Harry Brown |
| 7873 | Harsh Times |
| 22314 | In Too Deep |
| 22907 | Takers |
| 41283 | Faster |

**Evaluation:**

The plot-based recommendation system demonstrates an RMSE of 1.26 and an MAE of 0.914. Conversely, the metadata-based recommendation system yields an RMSE of 1.24 and an MAE of 0.89. The similarity in error rates between both filtering techniques suggests potential scalability and sparsity issues within the systems. This parallelism in errors might point towards underlying challenges in handling the volume and distribution of data across both methods.

**Collaborative filtering:**

The collaborative movie recommendation system, employing SVD matrix factorization, offers personalized movie suggestions tailored to individual users. The table 4.3 showcases the recommendations for different users, each identified by a unique ID. Within each user's recommendations, a collection of movie IDs and corresponding titles is presented. The recommendation system suggested different movies to different users , implies a tailored approach, customizing suggestions according to user's preferences or historical ratings.This personalized nature enhances user experience by offering a diverse range of movie choices. Moreover, the inclusion of widely popular movies such as "Forrest Gump" and "Saving Private Ryan" across multiple users indicates their general appeal or acclaim within the system.

The presence of different movies for each user suggests that the system effectively leverages collaborative filtering, identifying similarities between user behaviors or preferences to offer relevant recommendations. These recommendations are instrumental in enhancing user engagement and satisfaction by providing a mix of familiar favorites and potentially new, yet fitting, movie options.

Table 4.3: Collaborative Movie Recommendations similar

| User ID | Movie ID | Movie Title |
|---|---|---|
| 17148 | 356 | Forrest Gump (1994) |
| | 2023 | Godfather: Part III, The (1990) |
| | 4022 | Cast Away (2000) |
| | 2496 | Blast from the Past (1999) |
| | 2907 | Superstar (1999) |
| | 4639 | America's Sweethearts (2001) |
| | 2003 | Gremlins (1984) |
| | 3977 | Charlie's Angels (2000) |
| | 2491 | Simply Irresistible (1999) |
| 144071 | 1721 | Titanic (1997) |
| | 30816 | Phantom of the Opera, The (2004) |
| | 74789 | Alice in Wonderland (2010) |
| | 2424 | You've Got Mail (1998) |
| | 8529 | Terminal, The (2004) |
| | 5989 | Catch Me If You Can (2002) |
| | 4370 | A.I. Artificial Intelligence (2001) |
| | 2671 | Notting Hill (1999) |
| | 99149 | MisÃ©rables, Les (2012) |
| | 103235 | Best Offer, The (Migliore offerta, La) (2013) |
| 163992 | 587 | Ghost (1990) |
| | 356 | Forrest Gump (1994) |
| | 339 | While You Were Sleeping (1995) |
| | 588 | Aladdin (1992) |
| | 150 | Apollo 13 (1995) |
| | 586 | Home Alone (1990) |

| User ID | Movie ID | Movie Title |
|---|---|---|
| | 1961 | Rain Man (1988) |
| | 1097 | E.T. the Extra-Terrestrial (1982) |
| | 2028 | Saving Private Ryan (1998) |
| | 2797 | Big (1988) |
| 161584 | 1213 | Goodfellas (1990) |
| | 2028 | Saving Private Ryan (1998) |
| | 1961 | Rain Man (1988) |
| | 1219 | Psycho (1960) |
| | 198 | Strange Days (1995) |
| | 1610 | Hunt for Red October, The (1990) |
| | 1242 | Glory (1989) |
| | 1590 | Event Horizon (1997) |
| | 1396 | Sneakers (1992) |
| | 329 | Star Trek: Generations (1994) |
| 188478 | 72998 | Avatar (2009) |
| | 5952 | Lord of the Rings: The Two Towers, The (2002) |
| | 3578 | Gladiator (2000) |
| | 3753 | Patriot, The (2000) |
| | 4993 | Lord of the Rings: The Fellowship of the Ring, The (2001) |
| | 733 | Rock, The (1996) |
| | 356 | Forrest Gump (1994) |
| | 2028 | Saving Private Ryan (1998) |
| | 51662 | 300 (2007) |
| | 59315 | Iron Man (2008) |

**Evaluation:**

The model's accuracy assessment relied on two key metrics: RMSE (Root Mean Square Error) and MAE (Mean Absolute Error). The evaluation was conducted over the test dataset, comparing estimated ratings against actual ratings. The computed values revealed an RMSE of 0.9024 and an MAE of 0.6928. These results signify an exemplary performance of the model in predicting ratings, demonstrating its robustness in accurately estimating user preferences.

**Hybrid filtering:**

The output for the same movie title but different user IDs is different because the recommendations are personalized based on each user's viewing history and references. In a hybrid recommendation system, the recommendations are a combination of different methods, such as collaborative and content-based filtering. When different user IDs are provided, then the system considers that user's history and preferences to generate recommendations. So, even if the same movie title is used as input, the recommendations may differ due to varying tastes and preferences among different users.

Table 4.4: Personalised movie recommendations for user 1

| **title** | **movieId** | **vote_average** | **vote_count** | **estimated_rating** |
|---|---:|---:|---:|---:|
| Scarface | 111 | 8.0 | 2948 | 3.172288 |
| Apocalypse Now | 28 | 8.0 | 2055 | 3.168075 |
| The Talented Mr. Ripley | 1213 | 7.0 | 767 | 3.100138 |
| The Cotton Club | 2148 | 6.6 | 68 | 2.719195 |
| The Son of No One | 74536 | 4.8 | 92 | 2.714431 |
| Amidst the Devil's Wings | 378237 | 0.0 | 0 | 2.714431 |
| The Rainmaker | 11975 | 6.7 | 235 | 2.714431 |
| Donnie Brasco | 9366 | 7.4 | 1147 | 2.714431 |
| The Godfather: Part II | 240 | 8.3 | 3338 | 2.520758 |
| The Godfather: Part III | 242 | 7.1 | 1546 | 2.478803 |

Table 4.5: Personalised movie recommendations for user 2

| **title** | **movieId** | **vote_average** | **vote_count** | **estimated_rating** |
|---|---:|---:|---:|---:|
| Scarface | 111 | 8.0 | 2948 | 4.207079 |
| Apocalypse Now | 28 | 8.0 | 2055 | 3.986233 |
| The Talented Mr. Ripley | 1213 | 7.0 | 767 | 3.808576 |
| The Godfather: Part III | 242 | 7.1 | 1546 | 3.723983 |
| The Son of No One | 74536 | 4.8 | 92 | 3.434960 |
| Amidst the Devil's Wings | 378237 | 0.0 | 0 | 3.434960 |

| | | | | |
|---|---|---|---|---|
| The Rainmaker | 11975 | 6.7 | 235 | 3.434960 |
| Donnie Brasco | 9366 | 7.4 | 1147 | 3.434960 |
| The Cotton Club | 2148 | 6.6 | 68 | 3.201132 |
| The Godfather: Part II | 240 | 8.3 | 3338 | 3.045971 |

From the tables (4.4 and 4.5), it is evident that the user 1 and user 2 may have different viewing histories, even for the same movie title "The GodFather", which could lead to variations in the recommendations provided. This personalization is a key factor of recommendation systems, as it aims to tailor suggestions to individual user's tastes. For given user hybrid recommendation systems gives top 10 (sorted on estimated ratings) response similar to the rating given by user to the given movie. From the above tables it can be said that the same movie "The Godfather" is given to two user's, but the models gives different suggestions, such as, for the user 1 the model gives suggestions in a estimated ratings range of 3.17 to 2.4 , whereas for the user 2 the model gives suggestions in a estimated ratings ranges from 4.2 to 3.4. These, recommendations suggest that, user 2 might have rated more than user 1.

**Evaluation:**

The evaluation of the hybrid recommendation system was executed using the test dataset to gauge its performance. The calculated RMSE and MAE values stood at 0.89 and 0.59 respectively, showcasing a substantial enhancement in model performance when compared to both the collaborative and content-based filtering approaches. This outcome underscores the effectiveness and superiority of the hybrid system in providing accurate and refined recommendations by leveraging combined techniques.

## 4.1 Comparing the evaluation metrics of models

From the above table 4.6, it is evident that the plot-based and metadata based content recommendation models show higher errors compared to other two models. This might be due to the limitations associated with each filtering approach, for instance, the plot-based approach might struggle because of its sole reliance on plot of the movie , where as the metadata based might get influenced by the sparse or incomplete data. On the other hand, the most widely used recommendation system approach collabortive filtering, shows a moderate errors 0.9024 and 0.6928 on evaluating over RMSE and

Table 4.6: RMSE and MAE values of Different Models

| **Model Name** | **RMSE** | **MAE** |
|---|---|---|
| Plot-Based Recommendation | 1.26 | 0.914 |
| Metadata-Based Recommendation | 1.24 | 0.879 |
| Collaborative Filtering | 0.9024 | 0.6928 |
| Hybrid Recommendation | 0.89 | 0.59 |

MAE respectively. Its reliance on user-item interactions limits its capability to capture nuanced user preferences.

However, the Hybrid Recommendation model stands out for its significantly lower RMSE and MAE. Integrating multiple techniques, it overcomes the limitations of singular approaches by merging collaborative and content-based methods, enhancing accuracy by considering both item metadata and user behavior.

CHAPTER 5

# Conclusion

The implementation and evaluation of various recommendation systems have provided valuable insights into their effectiveness and limitations. Each approach - demographic filtering, content-based filtering, collaborative filtering and hybrid filtering - offers distinct advantage and challenges in the context of movie recommendations.

In demographic filtering, movies are recommended based on general trends like high ratings or popularity. Content-based filtering, utilizing cosine similarity and TF-IDF vectorization on movie overviews, suggests movies similar to those a user has liked before, emphasizing content relevance. Collaborative filtering relies on user behavior and preferences, generating recommendations based on user interactions and item similarities, enhancing the personalized suggestions. The hybrid filtering model combines multiple methods, combining strengths from content and collaborative techniques to offer comprehensive recommendations. Notably, the content-based filtering used count vectorizer for metadata-based recommendations,as the count of each item is significant over the weighted count of item.

Different recommendation systems offer unique advantages and challenges. Demographic filtering relies on user attributes but might miss individual preferences. Content-based filtering considers movie details but struggles with new or less-known films. Collaborative filtering analyzes user behavior but faces issues with new users or items lacking data. Hybrid filtering combines these methods for more accurate recommendations, though its implementation can be complex.

Overall, hybrid filtering is promising, leveraging the strengths of demographic, content-based, and collaborative approaches. However, balancing these methods and refining the system with user feedback remains crucial for optimal performance and continuous enhancement over time.

## 5.1 Future Work

Exploring advanced content-based analysis through NLP and refining collaborative filtering with deep learning models are promising future directions. Optimizing hybrid filtering for better personalization and considering sequential recommendation models for temporal patterns could enhance system accuracy. Addressing ethical concerns and improving interactivity while ensuring scalability and gathering user feedback remain key focuses for ongoing development.

# Bibliography

[1] S. Agrawal and P. Jain, "An improved approach for movie recommendation system," 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2017, pp. 336-342, doi: 10.1109/I-SMAC.2017.8058367.

[2] Zan Wang, Xue Yu, Nan Feng, Zhenhua Wang, "An improved collaborative movie recommendation system using computational intelligence", Journal of Visual Languages & Computing, Volume 25, Issue 6, 2014, Pages 667-675, ISSN 1045-926X, `https://doi.org/10.1016/j.jvlc.2014.09.011.`

[3] M. Gupta, A. Thakkar, Aashish, V. Gupta and D. P. S. Rathore, "Movie Recommender System Using Collaborative Filtering," 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC), Coimbatore, India, 2020, pp. 415-420, doi: 10.1109/ICESC48915.2020.9155879.

[4] Lekakos, G., Caravelas, "P. A hybrid approach for movie recommendation". Multimed Tools Appl 36, 55-70 (2008). https://doi.org/10.1007/s11042-006-0082-7

[5] M.G. Vozalis, K.G. Margaritis, "Using SVD and demographic data for the enhancement of generalized Collaborative Filtering", Information Sciences, Volume 177, Issue 15, 2007, Pages 3017-3037, ISSN 0020-0255, https://doi.org/10.1016/j.ins.2007.02.036.

[6] J. Zhang, Y. Wang, Z. Yuan and Q. Jin, "Personalized real-time movie recommendation system: Practical prototype and evaluation," in Tsinghua Science and Technology, vol. 25, no. 2, pp. 180-191, April 2020, doi: 10.26599/TST.2018.9010118.

[7] Kumar, M., Yadav, D. K., Singh, A., & Gupta, V. K. (2015). "A movie recommender system: Movrec". International journal of computer applications, 124(3).

[8] Wu, C. S. M., Garg, D., & Bhandary, U. (2018, November). "Movie recommendation system using collaborative filtering". In 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS) (pp. 11-15). IEEE.

[9] Jake Tae, https://jaketae.github.io/study/svd/

[10] Khang Pham, https://medium.com/@khang.pham.exxact/what-are-recommendation-systems-6bb5036042db

[11] Shubham Kumar Agrawal, https://www.analyticsvidhya.com/blog/2021/07/recommendation-system-understanding-the-basic-concepts/

[12] F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh, "Recommendation systems: Principles, methods and evaluation", Egyptian Informatics Journal, Volume 16, Issue 3, 2015, Pages 261-273, ISSN 1110-8665, https://doi.org/10.1016/j.eij.2015.06.005. (https://www.sciencedirect.com/science/article/pii/S1110866515000341)

[13] Blueshift,"Evolution of Recommender Systems", https://medium.com/@blueshiftlabs/evolution-of-recommender-systems-a5cb1b0612fd, July,2017.

[14] Karolina Holewa,"Perks of recommendation systems in business",https://www.miquido.com/blog/perks-of-recommendation-systems-in-business/,2023

[15] Bushra Alhijawi, Arafat Awajan, and Salam Fraihat. 2022. "Survey on the Objectives of Recommender Systems": Measures, Solutions, Evaluation Methodology, and New Perspectives. ACM Comput. Surv. 55, 5, Article 93 (May 2023), 38 pages. https://doi.org/10.1145/3527449

[16] A. Javed, F. Rehman, N. Sarfraz, H. Sharif, R. Khan and A. M. Khan, "Movie Recommendation System with Sentimental Analysis Using Cosine Similarity Technique," 2022 3rd International Conference on Innovations in Computer Science & Software Engineering (ICONICS), Karachi, Pakistan, 2022, pp. 1-8, doi: 10.1109/ICONICS56716.2022.10100512.

[17] J. F. Mohammad and S. Urolagin, "Movie Recommender System Using Content-based and Collaborative Filtering," 2022 IEEE IAS Global Conference on Emerging Technologies (GlobConET), Arad, Romania, 2022, pp. 963-968, doi: 10.1109/GlobConET53749.2022.9872515.

[18] Cami, Bagher Rahimpour, Hamid Hassanpour, and Hoda Mashayekhi. "A content-based movie recommender system based on temporal user preferences." 2017 3rd Iranian conference on intelligent systems and signal processing (ICSPIS). IEEE, 2017.

[19] S. Sahu, R. Kumar, M. S. Pathan, J. Shafi, Y. Kumar and M. F. Ijaz, "Movie Popularity and Target Audience Prediction Using the Content-Based Recommender System," in IEEE Access, vol. 10, pp. 42044-42060, 2022, doi: 10.1109/ACCESS.2022.3168161.

[20] Shubham Sayon, "Introduction To Machine Learning And Its Applications", https://blog.finxter.com/introduction-to-machine-learning-and-its-applications/, December 19, 2020

[21] Canburak Tumer,"Machine Learning Basics with Examples -Part 2 Supervised Learning",https://medium.com/@canburaktumer/machine-learning-basics-with-examples-part-2-supervised-learning-e2b740ff014c,August,2018.

[22] Shubham Koli,"The 10 Best Machine Learning Algorithms for Data Science Beginners",https://medium.com/@MrBam44/the-10-best-machine-learning-algorithms-for-data-science-beginners-75c32b32a723,2022.

[23] sumit S Patil,"Reinforcement Learning", https://www.linkedin.com/pulse/reinforcement-learning-sumit-s-patil,2023

[24] Emma Grimaldi,"How to build a content-based movie recommender system with Natural Language Processing",https://towardsdatascience.com/how-to-build-from-scratch-a-content-based-movie-recommender-with-natural-language-processing-25ad400eb243,2018

[25] Unnikrishnan C S ,"How sklearn's Tfidfvectorizer Calculates tf-idf Values",https://www.analyticsvidhya.com/blog/2021/11/how-sklearns-tfidfvectorizer-calculates-tf-idf-values/,2021.

[26] Joshua Sung,"Natural Language Processing: Count Vectorization and Term Frequency -Inverse Document Frequency",https://medium.com/@joshsungasong/natural-language-processing-count-vectorization-and-term-frequency-inverse-document-frequency-49d2156552c1,2018.

[27] Varun,"Cosine similarity: How does it measure the similarity, Maths behind and usage in Python"https://towardsdatascience.com/cosine-similarity-how-does-it-measure-the-similarity-maths-behind-and-usage-in-python-50ad30aad7db,2020

[28] Mayur Badole, "A Comprehensive Guide on Recommendation Engines and Implementation",https://www.analyticsvidhya.com/blog/2022/03/a-comprehensive-guide-on-recommendation-engines-and-implementation/,2022.

[29] Akshita Chugh,"MAE, MSE, RMSE, Coefficient of Determination, Adjusted R Squared - Which Metric is Better?",https://medium.com/analytics-vidhya/mae-mse-rmse-coefficient-of-determination-adjusted-r-squared-which-metric-is-better-cd0326a5697e,2020.

[30] Padhma M,"A Comprehensive Introduction to Evaluating Regression Models",https://www.analyticsvidhya.com/blog/2021/10/evaluation-metric-for-regression-models/,2023

[31] Singular value decomposition (SVD) theorem, https://inst.eecs.berkeley.edu/~ee127/sp21/livebook/thm_svd.html, 2020

[32] "Machine Learning (ML) Defined: How It Works and Its Impact", https://www.tableau.com/learn/articles/define-machine-learning?cq_cmp=20663578515&cq_net=x&cq_plac=&gad=1&gclid=CjwKCAjw1t2pBhAFEiwA_-A-NOKAYPlmKlTgtapPfChn1Q0sOWzWqyy5LsF4MAUvsaB9KS0I6bLu3xoCd24QAvD_BwE&gclsrc=aw.ds

[33] "What is machine learning?",https://www.ibm.com/topics/machine-learning

[34] Sara Brown ,"Machine learning, explained",https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[35] Mohit Gupta,"What is Machine Learning?",https://www.geeksforgeeks.org/ml-machine-learning/

[36] Linnda Tucci,"What is machine learning and how does it work? In-depth guide",https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

[37] "Javatpoint-Machine Learning Algorithms" ,https://www.javatpoint.com/machine-learning-algorithms

[38] Simon Tavasoli,"Top 10 Machine Learning Algorithms For Beginners: Supervised, and More",https://www.simplilearn.com/10-algorithms-machine-learning-engineers-need-to-know-article 2023

[39] "What Is Machine Learning? A Definition.",https://www.expert.ai/blog/machine-learning-definition/

[40] Suman Mukherjee, " Movie Recommendation System",https://medium.com/@smn.mukherjee/movie-recommendation-system-d8eefce21edd

[41] Nathan Rosidi,"Step-by-Step Guide to Building Content-Based Filtering",https://www.stratascratch.com/blog/step-by-step-guide-to-building-content-based-filtering/

[42] zeynep beyza ayman ,"Recommendation Systems: Content-Based Filtering",https://medium.com/mlearning-ai/recommendation-systems-content-based-filtering-e19e3b0a309e

[43] "Collaborative Filtering In Recommender Systems: Learn All You Need To Know ",https://www.iteratorshq.com/blog/collaborative-filtering-in-recommender-systems/

[44] "How Does Collaborative Filtering Work in Recommender Systems?",https://www.turing.com/kb/collaborative-filtering-in-recommender-system#user-item-interaction-matrix

[45] Houtao Deng,"Recommender Systems in Practice", https://towardsdatascience.com/recommender-systems-in-practice-cef9033bb23a

[46] VAIBHAV KUMAR,"Singular Value Decomposition (SVD) & Its Application In Recommender System",https://analyticsindiamag.com/singular-value-decomposition-svd-application-recommender-system/

[47] The Upwork Team,"What Content-Based Filtering Is and Why You Should Use It",https://www.upwork.com/en-gb/resources/what-is-content-based-filtering

[48] Karolina Holewa,"We know what you like! Perks of recommendation systems in business",https://www.miquido.com/blog/perks-of-recommendation-systems-in-business/,2020

[49] Anwar, Taushif & Vijayasundaram, Uma & ., Shahajd. (2020). Book Recommendation for eLearning Using Collaborative Filtering and Sequential Pattern Mining. 1-6. 10.1109/ICDABI51230.2020.9325599.

[50] Badreesh Shetty,"An In-Depth Guide to How Recommender Systems Work", https://builtin.com/data-science/recommender-systems

[51] Melville, P., Sindhwani, V. (2011). Recommender Systems. In: Sammut, C., Webb, G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA.

[52] F.O. Isinkaye, Y.O. Folajimi, B.A. Ojokoh, Recommendation systems: Principles, methods and evaluation, Egyptian Informatics Journal, Volume 16, Issue 3, 2015, Pages 261-273, ISSN 1110-8665, (https://www.sciencedirect.com/science/article/pii/S1110866515000341)

[53] Pinakin Ariwala,"How Do Recommendation Engines Work? What are the Benefits?",https://marutitech.com/recommendation-engine-benefits/

[54] "nvidia-Recommendation system",https://www.nvidia.com/en-us/glossary/data-science/recommendation-system/#:~:text=Recommender%20systems%20are%20trained%20to,clicks%2C%20likes%2C%20and%20purchases.

[55] Maruti Techlabs, "Explained â Working and Advantages of a Recommendation Engine",https://medium.com/geekculture/explained-working-and-advantages-of-a-recommendation-engine-16cbff7796c

[56] Geoviz,"Recommendation System Advantages",https://geo-viz.com/blog/advantages-of-a-recommendation-system/,2014

[57] "Geeksforgeeks- Reinforcement learning", https://www.geeksforgeeks.org/what-is-reinforcement-learning/

[58] "Intellipaat-Applications of Reinforcement Learning",https://intellipaat.com/blog/applications-of-reinforcement-learning/

[59] "9 Real-Life Examples of Reinforcement Learning",https://onlinedegrees.scu.edu/media/blog/9-examples-of-reinforcement-learning

[60] Pratik Aher,"Balancing Act: Addressing Popularity Bias in Recommendation Systems", https://towardsdatascience.com/balancing-act-addressing-popularity-bias-in-recommendation-systems-db5448c6a2a4#:~:text=While%20popular%20items%20can%20still,metrics%20that%20reward%20overall%20popularity.

[61] Emre Yalcin, Alper Bilge, Evaluating unfairness of popularity bias in recommender systems: A comprehensive user-centric analysis, Information Processing Management, Volume 59, Issue 6, 2022, 103100, ISSN 0306-4573, https://www.sciencedirect.com/science/article/pii/S0306457322002011

[62] Shuvayan Das,"Beginners Guide to Content Based Recommender Systems",https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/#h-how-does-vector-space-model-works

[63] "Turing-A Guide to Content-Based Filtering In Recommender Systems", https://www.turing.com/kb/content-based-filtering-in-recommender-systems#content-based-filtering

[64] Fatih Karabiber,"Cosine Similarity",https://www.learndatasci.com/glossary/cosine-similarity/

[65] Mercy Moraa,"An overview of the algorithms behind Recommender Systems", https://www.linkedin.com/pulse/overview-algorithms-behind-recommender-systems-mercy-moraa/

[66] Upadhyaya, Prakash. (2023). Study of Mathematical Model for User-based Collaborative Filtering and Item-based Collaborative Filtering.

[67] "Memory Based Collaborative Filtering - User Based",https://medium.com/@corymaklin/memory-based-collaborative-filtering-user-based-42b2679c6fb5

[68] Ashmi Banerjee, "Drawbacks of Recommender Systems", https://medium.com/@ashmi_banerjee/drawbacks-of-recommender-systems-e6a596fc937e

[69] Madhukar, Mani. "Challenges & limitation in recommender systems." International Journal of Latest Trends in Engineering and Technology (IJLTET) 4.3 (2014): 138-142.

[70] Denise Chen, "Recommender System - Matrix Factorization", https://towardsdatascience.com/recommendation-system-matrix-factorization-d61978660b4b

APPENDIX A

# code

```
# importing required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import ast # it is used to convert the string into the list
import plotly
import plotly.offline as py
py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
from ast import literal_eval
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from nltk import PorterStemmer
from surprise import Reader, Dataset, SVD
from surprise import accuracy
from surprise.model_selection import train_test_split
from collections import defaultdict
from sklearn.metrics import mean_absolute_error,
    mean_squared_error

# Read csv files
credits_raw_dataset=pd.read_csv('C:/Users/sushm/OneDrive/
    Documents/dissertation/tmdb_5000_credits.csv')
movies_raw_dataset=pd.read_csv('C:/Users/sushm/OneDrive/
    Documents/dissertation/tmdb_5000_movies.csv')

pd.set_option('display.max_columns',None)  #display complete
    columns in the dataset
pd.set_option('display.max_rows',None)     # display complete
    rows in the dataset

credits_raw_dataset.head(4) # viewing the top 6 records of the
    credit dataset
movies_raw_dataset.head(4) # viewing the top 6 records of the
    movies dataset
credits_raw_dataset.info() # structure of the credits dataset
movies_raw_dataset.info()  # struucture of movies dataset

# Data pre-processing
credits_raw_dataset.isnull().sum()   # checking for null values
     in credits dataset
movies_raw_dataset.isnull().sum()  # checking for null values
    in movies dataset
credits_raw_dataset.columns=['id','title','cast','crew'] #
    renaming the columns of credits dataset for merging
movies_raw_dataset.drop(['title'],axis=1,inplace=True)  #
    droping the title column as the credits dataset already has
```

```
   title column
merged_movies_dataset=pd.merge(credits_raw_dataset,
   movies_raw_dataset,on='id')  # merging the credits and
   movies dataset on id
merged_movies_dataset.info()  # structure of the movies dataset
merged_movies_dataset.shape  # dimensions of the dataset (
   number of rows and number of columns)
merged_movies_dataset.isnull().sum()  # finding the null values
merged_movies_dataset.drop(['homepage','tagline','runtime'],
   axis=1,inplace=True)  # dropping the unwanted fetaures such
   as homepage, tagline, runtime
merged_movies_dataset['overview'].fillna('␣',inplace=True)  #
   replaing null values

# Data Formating
merged_movies_dataset.iloc[0].crew  # printing the complete
   crew data of 1st row
# extrcat the name of the crew where the job is director
''' Defing a function extrcat_director to extract the name of
   the director from the crew'''

def extract_director(column_name):
    list_name_director=[]
    for i in ast.literal_eval(column_name):
        if i['job']=='Director':
            list_name_director.append(i['name'])
    return list_name_director
merged_movies_dataset['crew']=merged_movies_dataset['crew'].
   apply(extract_director)  # extrcating the director
# extract the top 3 actors/cast members from the cast column
   and keywords from keyword column
```

```
''' Defining a function cast__keyword_list_names to extract the
     top three cast members from the cast column and three
   keywords from the keywords column'''
def extract_cast_keyword(column_name):
    cast__keyword_list_names=[]
    for i in ast.literal_eval(column_name):
        if len(cast__keyword_list_names)<3:
            cast__keyword_list_names.append(i['name'])

        else:
            break
    return cast__keyword_list_names
merged_movies_dataset['cast']=merged_movies_dataset['cast'].
   apply(extract_cast_keyword)    #extracting top three cast
   from the cast column
merged_movies_dataset['keywords']=merged_movies_dataset['
   keywords'].apply(extract_cast_keyword)  # listing frequently
    used three keywords of a movie

# extract genres
''' Defining a function formating data to list movie genres,
   production countries, companies'''
def formating_data(column_name):
    cast_list_names=[]
    for i in ast.literal_eval(column_name):
        cast_list_names.append(i['name'])

    return cast_list_names

merged_movies_dataset['genres']=merged_movies_dataset['genres'
   ].apply(formating_data)  # listing the genres of movie
merged_movies_dataset['production_companies']=
    merged_movies_dataset['production_companies'].apply(
    formating_data)  # listing names of companies involved in
    producing movies
merged_movies_dataset['production_countries']=
    merged_movies_dataset['production_countries'].apply(
    formating_data)   # listing the production countries
merged_movies_dataset.head(2)

# Exploratory Data Analysis
# Explode the genres lists into separate rows
merged_movies_dataset_exploded = merged_movies_dataset.explode(
    'genres')
# Get the count of movies per genre
movie_genres_count = merged_movies_dataset_exploded['genres'].
    value_counts()
movie_genres_count
movie_genres_count_sorted=movie_genres_count.sort_values(
    ascending=True)
# Plotting the bar graph
plt.figure(figsize=(10, 6))
movie_genres_count_sorted.plot(kind='barh', color='skyblue')
plt.title('Genre␣Counts')
plt.xlabel('Count')
plt.ylabel('Genres')
plt.xticks(rotation=45)  # Rotate x-axis labels for better
    visibility
plt.show()

# Select the top 20 movies based on vote_count
top_10_movies_vote_count = merged_movies_dataset.nlargest(10, '
    vote_count').sort_values('vote_count', ascending=True)
# Create the plot
```

```
plt.figure(figsize=(10, 6))
# Create the bar plot
plt.barh(top_10_movies_vote_count['title'],
   top_10_movies_vote_count['vote_count'], color='skyblue')
# Set the y-axis limits
plt.ylim([-1, 10])
# Add labels and title
plt.xlabel('Number␣of␣Votes')
plt.ylabel('Movie␣Title')
plt.title('Top␣20␣Movies␣by␣Vote␣Count')

# Add text labels for vote_count and vote_average
for i, (count, avg) in enumerate(zip(top_10_movies_vote_count['
   vote_count'], top_10_movies_vote_count['vote_average'])):
     plt.text(3000, i, f'Votes:␣{count}', ha='right', va='center
         ', color='black', fontsize=10)
     plt.text(count + 100, i, f'Avg:␣{avg:.2f}', ha='left', va='
         center', color='orange', fontsize=10)
# Show the plot
plt.show()
top_10_movies_vote_count[['title','vote_count','vote_average'
   ]].sort_values('vote_count',ascending=False)  # tabulating
   the top 20 movies ordered by vote count descending order
# Create a new DataFrame by exploding the 'production_companies
   ' column
merged_movies_dataset_exploded_companies =
   merged_movies_dataset.explode('production_companies')
# Group by production company and aggregate the number of
   movies and average popularity
movies_production_companies =
   merged_movies_dataset_exploded_companies.groupby('
   production_companies').agg(
    num_movies=('title', 'count'),
    avg_popularity=('popularity', 'mean')
).reset_index()

# Display the statistics in tabular format
movies_production_companies
movies_production_companies.sort_values('num_movies',ascending=
   False) # sorting the production companies by number of
   movies in an descending order
production_movies_no_of_mvs=movies_production_companies.
   nlargest(10,'num_movies')
production_movies_no_of_mvs          # printing top 10
   production companies by number of number produced
production_companies_average_popularity=
   movies_production_companies.nlargest(10,'avg_popularity')
production_companies_average_popularity # printing the top 10
   production companies by average popularity of produced
   movies
movie_status_counts = merged_movies_dataset['status'].
   value_counts().reset_index() # count the occurances of each
   unique value in the 'status' column
movie_status_counts.columns = ['Status', 'Count']  # rename the
    columns of the new dataframe
movie_status_counts

# stack the resulting series to create a multi-level index
   dataframe, resent teh index, dropping the level 1 of the
   index
stack_series = merged_movies_dataset.apply(lambda x: pd.Series(
   x['production_countries']),axis=1).stack().reset_index(level
   =1, drop=True)
stack_series.name = 'countries'  # renaming the resulting
```

```
    series to 'countries'

# drop the original 'production countries' column and join the
    dataframe with teh stacked countries series
movies_contries_dataframe = merged_movies_dataset.drop('
    production_countries', axis=1).join(stack_series)
movies_contries_dataframe = pd.DataFrame(
    movies_contries_dataframe['countries'].value_counts()) #
    create a new dataframe with teh count of movies for each
    country
movies_contries_dataframe['country'] =
    movies_contries_dataframe.index    # add a new column '
    country' with unique countries as teh index
movies_contries_dataframe.columns = ['num_movies', 'country']
      # rename teh columns
movies_contries_dataframe = movies_contries_dataframe.
    reset_index().drop('index', axis=1) # reset the index,
    dropping the existing index column
movies_contries_dataframe.head(20)  # printing the top 20 rows
    of teh resulting dataframe
movies_contries_dataframe = movies_contries_dataframe[
    movies_contries_dataframe['country'] != 'United␣States␣of␣
    America']
#create a cholopleth map
data = [ dict(
        type = 'choropleth',
        locations = movies_contries_dataframe['country'],
        locationmode = 'country␣names',
        z = movies_contries_dataframe['num_movies'],
        text = movies_contries_dataframe['country'],
        colorscale = 'Viridis',
        autocolorscale = False,
        reversescale = False,
        marker = dict(
            line = dict (
                color = 'rgb(180,180,180)',
                width = 0.5
            ) ),
        colorbar = dict(
            autotick = False,
            tickprefix = '',
            title = 'Production␣Countries'),
      ) ]

# define layout of the map
layout = dict(
    title = 'Production␣Countries␣for␣the␣Movies␣(Apart␣from␣US
       )',
    geo = dict(
        showframe = False,
        showcoastlines = False,
        projection = dict(
            type = 'Mercator'
        )
    )
)

# combine the data and layout to create the figures
fig = dict( data=data, layout=layout )
# plot the figures using plotly
py.iplot( fig, validate=False, filename='d3-world-map' )
# create a dataframe with the count of movies for each
   originial language
movie_language_dataframe=pd.DataFrame(merged_movies_dataset['
    original_language'].value_counts())
movie_language_dataframe.head()  # display the first few rows
    of the dataframe

# Define a custom function 'extract_month' to extract the month
     from a date string
# The function attempts to convert the month part of the date
    string to an integer and retrieve the corresponding month
    abbreviation
# If any error occurs during this process, it returns NaN (Not
    a Number)
months_year = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
     'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
def extract_month(date_obj):
    try:
        # Split the date string using '-' as the delimiter,
            convert the month part to an integer, and get the
            corresponding month abbreviation
        return months_year[int(str(date_obj).split('-')[1]) -
            1]
    except:
        # Return NaN if any error occurs during the process
        return np.nan

merged_movies_dataset['month'] = merged_movies_dataset['
    release_date'].apply(extract_month)  # extrcating teh month
    from teh relase date of the given dataset

# create a dataframe with count of movies per ecah month
movie_counts_month = merged_movies_dataset['month'].
    value_counts().reset_index()
movie_counts_month.columns = ['Month', 'Number␣of␣Movies␣
```

```
    Released'] # rename the columns
# sort the dataframe based on the number of movies released
    column
movie_counts_month = movie_counts_month.sort_values('Number␣of␣
    Movies␣Released')
print(movie_counts_month)
sns.set(font_scale=1.25)
pd.set_option('display.max_colwidth', 50)

# Create a bar plot using seaborn to visualize the number of
    movies released in each month
plt.figure(figsize=(12,6))
plt.title("Number␣of␣Movies␣released␣in␣a␣particular␣month.")
sns.countplot(x='month', data=merged_movies_dataset, order=
    months_year,color='skyblue')
# Create a DataFrame with average revenue for blockbuster
    movies (revenue > 1e8) grouped by month
movies_monthly_revenue = pd.DataFrame(merged_movies_dataset[
    merged_movies_dataset['revenue'] > 1e8].groupby('month')['
    revenue'].mean())
# Add a column for month to use in plotting
movies_monthly_revenue['mon'] = movies_monthly_revenue.index
# Plot a bar chart of average gross revenue by month for
    blockbuster movies
plt.figure(figsize=(12,6))
plt.title("Average␣Gross␣by␣the␣Month␣for␣Blockbuster␣Movies")
sns.barplot(x='mon', y='revenue', data=movies_monthly_revenue,
    order=months_year,color='skyblue')

# Extract and count the occurrences of movie cast members, then
      retrieve the top 20
movie_cast_dataframe_count=merged_movies_dataset['cast'].
```

```
    explode().value_counts().head(20)
movie_cast_dataframe_count  # printing frequenct occurances of
    top 20 cast
movie_cast_dataframe_count=movie_cast_dataframe_count.
    sort_values(ascending=True)
# Plotting the bar graph
plt.figure(figsize=(10, 6))
movie_cast_dataframe_count.plot(kind='barh', color='skyblue')
plt.title('Actors␣with␣most␣appearances')
plt.xlabel('count')
plt.ylabel('Actors')
plt.xticks(rotation=45)  # Rotate x-axis labels for better
    visibility
plt.show()

# Extract and count the occurrences of movie director, then
    retrieve the top 20
movie_crew_dataframe_count=merged_movies_dataset['crew'].
    explode().value_counts().head(20)
movie_crew_dataframe_count=movie_crew_dataframe_count.
    sort_values(ascending=True)
# Plotting the bar graph
plt.figure(figsize=(10, 6))
movie_crew_dataframe_count.plot(kind='barh', color='skyblue')
plt.title('Directors␣with␣most␣Movies')
plt.xlabel('count')
plt.ylabel('Directors')
plt.xticks(rotation=45)  # Rotate x-axis labels for better
    visibility
plt.show()

# Methodology
# Demographic Filtering
 #Filter the movies based on the vote count, keeping only those
     in the top 10% of the distribution
demographic_filtered_dataframe=merged_movies_dataset[
    merged_movies_dataset['vote_count']>merged_movies_dataset['
    vote_count'].quantile(q=0.9)]

# Define a function for calculating weighted ratings based on
    the IMDB formula
def weighted_rating(x):
      v=x['vote_count']
     m=merged_movies_dataset['vote_count'].quantile(q=0.9)
      r=x['vote_average']
      c=merged_movies_dataset['vote_average'].mean()
      return ((r*v)/(v+m))+((c*m)/(v+m))

# Apply the weighted_rating function to create a new column '
    WR_score' in the demographic_filtered_dataframe
demographic_filtered_dataframe['WR_score']=
    demographic_filtered_dataframe.apply(weighted_rating, axis
    =1)
# Sort the demographic_filtered_dataframe based on the '
    WR_score' column in descending order
demographic_filtered_dataframe_sorted_WR=
    demographic_filtered_dataframe.sort_values(by='WR_score',
    ascending=False)
# Display selected columns from the sorted DataFrame
demographic_filtered_dataframe_sorted_WR[['title', 'vote_count'
    , 'vote_average', 'popularity', 'WR_score']]
# Create a horizontal bar plot for the top 10 movies based on
    popularity
```

```
plt.figure(figsize=(20,7))
plt.barh(demographic_filtered_dataframe_sorted_WR['title'].head
   (10),demographic_filtered_dataframe_sorted_WR['popularity'].
   head(10), align='center', color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Rating")
plt.title("Rating␣Movies")
# Sort the merged_movies_dataset DataFrame based on the '
   popularity' column in descending order
demographic_filter_popular= merged_movies_dataset.sort_values('
   popularity', ascending=False)
# bar Plot the top 10 movies based on popularity
plt.figure(figsize=(15,5))
plt.barh(demographic_filter_popular['title'].head(10),
   demographic_filter_popular['popularity'].head(10), align='
   center', color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular␣Movies")

# Define a function that takes movie count as input an dgives
   high weight rated movies as output
def high_weight_rated_movies(movies_count):
    weighted_rating_sort = demographic_filtered_dataframe.
       sort_values('WR_score', ascending=False)
    return weighted_rating_sort[['id','title', 'vote_count', '
       vote_average','WR_score']].head(movies_count)
high_weight_rated_movies(10)

# Define a function that takes movie count as input an dgives
   high popular movies as output
def high_popular_movies(movies_count):
    movies_popular_sort = demographic_filtered_dataframe.
        sort_values('popularity', ascending=False)
    return movies_popular_sort[['id','title', 'vote_count', '
        vote_average', 'popularity']].head(movies_count)
high_popular_movies(10)

# Content-based filtering
# The lambda function splits each string in the 'overview'
    column into a list of words
merged_movies_dataset['overview']=merged_movies_dataset['
    overview'].apply(lambda x:x.split())
# The lambda function iterates over each element in the list
    and removes spaces between the words
merged_movies_dataset['genres']=merged_movies_dataset['genres'
    ].apply(lambda x: [i.replace(" ","") for i in x])
# The lambda function iterates over each element in the list
    and removes spaces between the words
merged_movies_dataset['keywords']=merged_movies_dataset['
    keywords'].apply(lambda x: [i.replace(" ","") for i in x])
merged_movies_dataset['cast']=merged_movies_dataset['cast'].
    apply(lambda x: [i.replace(" ","") for i in x])
merged_movies_dataset['crew']=merged_movies_dataset['crew'].
    apply(lambda x: [i.replace(" ","") for i in x])
# The lambda function checks if the element is a list, and if
    so, joins the list elements into a string with spaces in
    between
# If the element is not a list, it remains unchanged
merged_movies_dataset['overview'] = merged_movies_dataset['
    overview'].apply(lambda x: ' '.join(x) if isinstance(x, list
    ) else x)
# Create an instance of TfidfVectorizer with English stop words
tfidf_vectoriser=TfidfVectorizer(stop_words='english')
```

```
# Transform the 'overview' column of the merged_movies_dataset
    into a TF-IDF matrix
tfidf_vectoriser_matrix=tfidf_vectoriser.fit_transform(
    merged_movies_dataset['overview'])
print(tfidf_vectoriser_matrix.todense())    # Print the TF-IDF
    matrix in dense format
tfidf_vectoriser_matrix.todense().shape
# The lambda function converts each string in the 'overview'
    column to lowercase
merged_movies_dataset['overview']=merged_movies_dataset['
    overview'].apply(lambda x:x.lower())
# extracting the feature names generated by the TfidfVectorizer
tfidf_vectoriser.get_feature_names()
# create an instance of the porterstemmer from the NLTK library
porter_stemmer=PorterStemmer()

#define a function named 'stemming' that takes word as input
    and applies stemming using the provided stemmer
def stemming(word):
    stemmed_words=[]      # Create an empty list to store the
        stemmed words
    for i in word.split():
        stemmed_words.append(porter_stemmer.stem(i))     # Use
            the provided stemmer to stem each word and append it
             to the list
    return ' '.join(stemmed_words)           # Join the stemmed
        words into a string and return the result
merged_movies_dataset['overview']=merged_movies_dataset['
    overview'].apply(stemming)

# Create an instance of TfidfVectorizer with English stop words
tfidf_vectoriser=TfidfVectorizer(stop_words='english')
```

```
# Transform the 'overview' column of the merged_movies_dataset
    into a TF-IDF matrix
tfidf_vectoriser_matrix=tfidf_vectoriser.fit_transform(
    merged_movies_dataset['overview'])
print(tfidf_vectoriser_matrix.todense())    # Print the TF-IDF
    matrix in dense format
tfidf_vectoriser_matrix.todense().shape
# Calculate cosine similarity matrix using linear kernel on the
     TF-IDF matrix
cosine_similarity_plot = linear_kernel(tfidf_vectoriser_matrix,
     tfidf_vectoriser_matrix)
cosine_similarity_plot.shape  # Print the shape of the cosine
    similarity matrix
# Create a Pandas Series with movie indices, using movie titles
     as index
indices_movies_dataset=pd.Series(data=list(
    merged_movies_dataset.index), index= merged_movies_dataset['
    title'] )

# Function that takes in movie title and similarity matrix as
    input and outputs most similar movies
def get_content_plot_recommendations(title, similarity):
    # Get the index of the movie that matches the title
     index_movie_title = indices_movies_dataset[title]
     # Get the pairwsie similarity scores of all movies with the
          given movie title
     pairwise_similarity_score = list(enumerate(similarity[
         index_movie_title]))
     # Sort the movies based on the similarity scores in
         descending order
     pairwise_similarity_score.sort(key=lambda x: x[1], reverse=
         True)
     # Get the scores of the 10 most similar movies
     pairwise_similarity_score=pairwise_similarity_score[1:11]
     # Get the movie indices
     movie_indices=[]
     for (x,y) in pairwise_similarity_score:
         movie_indices.append(x)
     # Return the top 10 most similar movies
     movie_similar_title=[]
     for x in movie_indices:
         movie_similar_title.append(merged_movies_dataset.iloc[x
           ]['title'])
     return pd.Series(data=movie_similar_title, index=
         movie_indices)
merged_movies_dataset.head()

get_content_plot_recommendations('The␣Dark␣Knight␣Rises',
    cosine_similarity_plot)

def get_content_plot_recommendations(title, similarity):
     # Get the index of the movie that matches the title
     index_movie_title = indices_movies_dataset[title]
     # Get the pairwise similarity scores of all movies with the
          given movie title
     pairwise_similarity_score = list(enumerate(similarity[
         index_movie_title]))
     # Sort the movies based on the similarity scores in
         descending order
     pairwise_similarity_score.sort(key=lambda x: x[1], reverse=
         True)
     # Get the scores of the 10 most similar movies
     pairwise_similarity_score = pairwise_similarity_score[1:11]
     # Get the movie indices and titles
     movie_indices = []
     movie_titles = []
     for (x, y) in pairwise_similarity_score:
         movie_indices.append(merged_movies_dataset.iloc[x]['id'
           ]) # Movie ID
         movie_titles.append(merged_movies_dataset.iloc[x]['
           title']) # Movie Title
     # Create a DataFrame with movie indices and titles
     recommendations = pd.DataFrame({'Movie␣ID': movie_indices,
        'Movie␣Title': movie_titles})
     return recommendations


# Get content-based plot recommendations for the movie 'The
    Dark Knight Rises'
get_content_plot_recommendations('The␣Dark␣Knight␣Rises',
    cosine_similarity_plot)
# Get content-based plot recommendations for the movie 'The
    Dark Knight Rises'
get_content_plot_recommendations('The␣Dark␣Knight␣Rises',
    cosine_similarity_plot)
# Get content-based plot recommendations for the movie 'Avatar'
get_content_plot_recommendations('Avatar',
    cosine_similarity_plot)
# metadata content_based recommendation systems
merged_movies_dataset[['title', 'cast', 'crew', 'keywords', '
    genres']].head(3)

# Function that merges data from multiple columns into a single
     string
def merging_data_string(data):
     # Join the values in the 'keywords', 'crew', 'cast', and '
```

```
        genres' columns into a single string
    return '␣'.join(data['keywords']) + '␣' + '␣'.join(data['
        crew']) + '␣' +'␣'.join(data['cast']) + '␣' + '␣'.join(
        data['genres'])
# Apply the merging_data_string function to create a new column
    'merge_data_string' in the merged_movies_dataset
merged_movies_dataset['merge_data_string'] =
   merged_movies_dataset.apply(merging_data_string, axis=1)
# Create an instance of CountVectorizer with English stop words
count_vectoriser = CountVectorizer(stop_words='english')
# Transform the 'merge_data_soup' column of the
   merged_movies_dataset into a count matrix
count_vectoriser_matrix = count_vectoriser.fit_transform(
   merged_movies_dataset['merge_data_string'])

# Calculate cosine similarity matrix using linear kernel on the
    count vectorizer matrix
count_vectoriser_similarity = cosine_similarity(
   count_vectoriser_matrix, count_vectoriser_matrix)
# Get content-based recommendations for the movie 'The Dark
   Knight Rises' using count vectorizer similarity
get_content_plot_recommendations('The␣Dark␣Knight␣Rises',
   count_vectoriser_similarity)
# Evalution of content based filtering
# Split the dataset into training and testing
plot_train_data = merged_movies_dataset.sample(frac=0.8,
   random_state=42)
plot_test_data = merged_movies_dataset.drop(plot_train_data.
   index)
# Create a TF-IDF vectorizer for movie overviews
plot_tfidf_vectorizer = TfidfVectorizer(stop_words='english')
plot_tfidf_matrix = plot_tfidf_vectorizer.fit_transform(
    plot_train_data['overview'])
# Calculating cosine similarity between movie overviews in the
    training data
plot_cosine_sim = linear_kernel(plot_tfidf_matrix,
    plot_tfidf_matrix)

# Function to generate content-based recommendations
def generate_content_recommendations(test_data, cosine_sim,
    tfidf_vectorizer, tfidf_matrix, train_data):
    plot_mae_values = []        # Intiallizing an empty list of
        mae values
    plot_rmse_values = []       # Intiallizing an empty list of
        rmse_values
    for idx, row in test_data.iterrows():       # iterate
        through each row in the test data
        overview = row['overview']              # Get the movie
            overbiew from the current row
        # Calculate TF-IDF vector for the given item's genres
        plot_tfidf_query = tfidf_vectorizer.transform([overview
            ])            # transforminh the moview overview into
             TF-IDF vector
        # Calculate cosine similarity between the query and
            training data
        plot_cosine_similarities = linear_kernel(
            plot_tfidf_query, tfidf_matrix).flatten()
        #  selecting the most similar items based on cosine
            similarity
        plot_similar_items = plot_cosine_similarities.argsort()
            [-6:-1][::-1]
        #  Calculate the predicted vote_average as the average
            of vote_average of the similar items
        plot_similar_items_in_train = [item for item in
```

```
            plot_similar_items if item in train_data.index]
                    # filtering for similar items available
            in the training data
        plot_predicted_vote_average = train_data.loc[
            plot_similar_items_in_train, 'vote_average'].mean()
                    # Predicting the average vote of simialr
            items
        # Appending the predicted values for evaluation
        plot_mae_values.append(plot_predicted_vote_average)
        plot_rmse_values.append(plot_predicted_vote_average)
            # Calculating MAE and RMSE for evaluation
    plot_mae = mean_absolute_error(test_data['vote_average'],
        plot_mae_values)
    plot_rmse = mean_squared_error(test_data['vote_average'],
        plot_rmse_values, squared=False)
    return plot_mae, plot_rmse

# evaluating the performance of the plot-based content
    recommendation systems.
plot_mae_content, plot_rmse_content =
    generate_content_recommendations(plot_test_data,
    plot_cosine_sim, plot_tfidf_vectorizer, plot_tfidf_matrix,
    plot_train_data)
print(f'MAE␣for␣plot-based␣content-based␣recommendations:␣{
    plot_mae_content}')
print(f'RMSE␣for␣plot-based␣content-based␣recommendations:␣{
    plot_rmse_content}')

#EValution metrics for metadata content-based filtering
# Split the dataset into training and testing
metadata_train_data = merged_movies_dataset.sample(frac=0.8,
    random_state=42)
```

```
metadata_test_data = merged_movies_dataset.drop(
    metadata_train_data.index)
# Create a TF-IDF vectorizer for movie metadata
metadata_tfidf_vectorizer = TfidfVectorizer(stop_words='english
    ')
metadata_tfidf_matrix = metadata_tfidf_vectorizer.fit_transform
    (metadata_train_data['merge_data_string'])
# Calculate cosine similarity between movies based on metadata
metadata_cosine_sim = linear_kernel(metadata_tfidf_matrix,
    metadata_tfidf_matrix)
# Create a CountVectorizer for the 'merge_data_string' column
metadata_count_vectorizer_soup = CountVectorizer(stop_words='
    english')
metadata_count_matrix_soup = metadata_count_vectorizer_soup.
    fit_transform(merged_movies_dataset['merge_data_string'])
# Calculate cosine similarity between items based on the '
    merge_data_string' column
metadata_cosine_sim_soup = cosine_similarity(
    metadata_count_matrix_soup, metadata_count_matrix_soup)

# Function to generate metadata content-based recommendations
def generate_metadata_content_recommendations(test_data,
    cosine_sim, tfidf_vectorizer, tfidf_matrix, train_data):
      metadata_mae_values = []
      metadata_rmse_values = []
      for idx, row in test_data.iterrows():
          merge_data_string = row['merge_data_string']
          # Calculate TF-IDF vector for the given movie's
              merge_data_string
          metadata_tfidf_query = tfidf_vectorizer.transform([
              merge_data_string])
          # Calculate cosine similarity between the query and all
```

```
            movie's
        metadata_cosine_similarities = linear_kernel(
            metadata_tfidf_query, tfidf_matrix).flatten()
        # Get indices of top 5 similar metadata movies
        metadata_similar_items = metadata_cosine_similarities.
            argsort()[-6:-1][::-1]
        # Calculate the predicted vote_average as the average
            of vote_average of the similar items
        metadata_similar_items_in_train = [item for item in
            metadata_similar_items if item in train_data.index]
        metadata_predicted_vote_average = train_data.loc[
            metadata_similar_items_in_train, 'vote_average'].
            mean()
        # Append the predicted metadata vote_average to the
            list
        metadata_mae_values.append(
            metadata_predicted_vote_average)
        metadata_rmse_values.append(
            metadata_predicted_vote_average)
    #Calculate MAE and RMSE for evaluation
    metadata_mae = mean_absolute_error(test_data['vote_average'
        ], metadata_mae_values)
    metadata_rmse = mean_squared_error(test_data['vote_average'
        ], metadata_rmse_values, squared=False)
    return metadata_mae, metadata_rmse


# Example of generating content-based recommendations
metadata_mae_content, metadata_rmse_content =
    generate_content_recommendations(metadata_test_data,
    metadata_cosine_sim_soup, metadata_tfidf_vectorizer,
    metadata_tfidf_matrix, metadata_train_data)
```

```
print(f'MAE␣for␣metadata␣content-based␣recommendations:␣{
    metadata_mae_content}')
print(f'RMSE␣for␣metadata␣content-based␣recommendations:␣{
    metadata_rmse_content}')

# collaborative filtering
rtb = pd.read_csv('C:/Users/sushm/OneDrive/Documents/
    dissertation/ratings.csv')
movies_dataset = pd.read_csv('C:/Users/sushm/OneDrive/Documents
    /dissertation/new_movie.csv')
rt=pd.read_csv('C:/Users/sushm/OneDrive/Documents/dissertation/
    ratings_small.csv')
# Loading the rating dataset from a CSV file
# Displaying the first few rows of the dataset
rt.head()
movies_dataset.head()
rating_dataset = pd.merge(movies_dataset,rt,on='movieId').drop
    (['genres','timestamp'],axis=1)
print(rating_dataset.shape)
rating_dataset.head()
rating_dataset.shape
# Creating an instance of the Reader class
reader_instance = Reader()
# Loading the ratings data into Surprise Dataset
surprise_rating_data = Dataset.load_from_df(rating_dataset[['
    userId', 'movieId', 'rating']], Reader())
# sample random trainset and testset
# test set is made of 25% of the ratings.
training_dataset, testing_dataset = train_test_split(
    surprise_rating_data, test_size=.25)
# Initializing the SVD algorithm.
svd_algo = SVD()
```

```
# Train the algorithm on the trainset, and predict ratings for
   the testset
svd_algo.fit(training_dataset)
rating_predictions = svd_algo.test(testing_dataset)
# Calculating Root Mean Square Error (RMSE) to evaluate
   accuracy
accuracy.rmse(rating_predictions)
accuracy.mae(rating_predictions)
svd_algo.predict(1, 302)  # Predicting rating for user 1 and
   movie 302 using the trained SVD algorithm

# defining a collaborating filtering function to generate top n
    movie recommendations for each user based on ratings
   predictions
def n_collaborative_filtering_recomendations(rating_predictions
   , number_of_movies):
    #  Dictionary to store top N movie recommendations for each
        user
    top_n_movies_coll_filt = defaultdict(list) #convert list
       into defaultdict which accomodates empty key values pair
    # Loop through each predictions in the predicted rating
    for user_id, movie_id, true_rating,estimate_rating, _ in
       rating_predictions:
        top_n_movies_coll_filt[user_id].append((movie_id,
           estimate_rating))  # append movie id and estimated
           rating to each user's list
    # Sort movies for each user by estimated rating in
       descending order
    for user_id, user_ratings in top_n_movies_coll_filt.items()
       :
        user_ratings.sort(key = lambda x:x[1], reverse = True)
        top_n_movies_coll_filt[user_id] = user_ratings[:
            number_of_movies]    # Extracting only top n movies
            with highest estimated ratings
    return top_n_movies_coll_filt

# Define the number of movies to recommend for each user
number_of_movies = 10
# Get top n collaborative filtering movie recommendations for
   each user
top_n_movies_coll_filt =
   n_collaborative_filtering_recomendations(rating_predictions,
   number_of_movies)
top_n_movies_coll_filt
# recommend a list of movies to user
# Iterate through the top n movie recommendations for each user
for user_id, user_ratings in top_n_movies_coll_filt.items():
    # Print the user ID along with recommended movie IDs
    print(user_id, [movie_id for (movie_id, _) in user_ratings
       ])
# Mapping movieId to title for later use
movieId_to_title = dict(zip(rating_dataset['movieId'],
   rating_dataset['title']))

# Loop through each user's recommendations
for user_id, user_ratings in top_n_movies_coll_filt.items():
    col_recommended_movies = []      # initialize a list to
       store recommended movies for each user
    # Iterate through each movie recomemndation for the current
       user
    for movie_id, _ in user_ratings:
        movie_title = movieId_to_title.get(movie_id)    #
           retrieve movie title using the movie ID from the
           dictionary 'movieId_to_title'
        # checking for the existence of movie title if so , add
            the movie id and title to the list
        if movie_title:
            col_recommended_movies.append((movie_id,
               movie_title))
    # Print user ID and their recommended movies with titles
    print(f"User␣ID:␣{user_id}")
    for movie_id, movie_title in col_recommended_movies:
        print(f"Movie␣ID:␣{movie_id},␣Title:␣{movie_title}")
              # Display movie ID and title
    print("----------------------------------------")


# hybrid filtering
#sample full trainset
train_dataset = surprise_rating_data.build_full_trainset()
# This function generates hybrid movie recommendations for a
   given user based on movie title using collaborative and
   content-based filtering techniques.
# It calculates pairwise similarity scores between the given
   movie and other movies, sorts them, and retrieves the top
   similar movies.
# Then, it predicts the ratings for these movies using the SVD
   algorithm and returns a DataFrame with movie details and
   estimated ratings for the user.

def hybrid_movie_recommendations(user_id, movie_title):
    # retrive the index of the movie that matches the movie
        title in dataset
    movie_title_index = indices_movies_dataset[movie_title]
    # calculate the pairwsie similarity scores between the
        given movie and all other movies
```

```
pair_wise_similarity_scores = list(enumerate(
   count_vectoriser_similarity[movie_title_index]))
# Sort the movies based on the similarity scores
pair_wise_similarity_scores.sort(key=lambda x: x[1],
   reverse=True)
# Retrieve the top similar movies
pair_wise_similarity_scores=pair_wise_similarity_scores
   [1:11]
# extract the indices of similar movies
movie_indexes=[]
for (x,y) in pair_wise_similarity_scores:
     movie_indexes.append(x)
# initialize lists to store movie details
movie_title=[]
movie_id=[]
movie_vote_average=[]
movie_vote_count=[]
# retrive movie details based on the retrieved indexes
for x in movie_indexes:
     movie_title.append(merged_movies_dataset.iloc[x]['title
        '])
     movie_id.append(merged_movies_dataset.iloc[x]['id'])
     movie_vote_average.append(merged_movies_dataset.iloc[x
        ]['vote_average'])
     movie_vote_count.append(merged_movies_dataset.iloc[x]['
        vote_count'])
# Predict the ratings a user might give to these top 10
   most similar movies
predicted_rating=[]
for k in movie_id:
     predicted_rating.append(svd_algo.predict(user_id, k,
        r_ui=None).est)
```

```
    # Create a DataFrame with movie details and estimated
        ratings for the user
    return pd.DataFrame({'index': movie_indexes, 'title':
        movie_title, 'id':movie_id, 'vote_average':
        movie_vote_average, 'vote_count':movie_vote_count,'
        estimated_rating':predicted_rating}).set_index('index').
        sort_values(by='estimated_rating', ascending=False)

# Generating movie recommendations using a hybrid approach
    based on the movie 'Avatar' and user ID 1
hybrid_movie_recommendations(1,'Avatar')
# Generating movie recommendations using a hybrid approach
    based on the movie 'Avatar' and user ID 2
hybrid_movie_recommendations(2,'Avatar')

# Splitting the Surprise dataset into training and testing sets
trainset, testset = train_test_split(surprise_rating_data,
    test_size=0.2, random_state=42)
# Initializing the SVD algorithm.
svd_algo = SVD()
# Train the algorithm on the trainset
svd_algo.fit(trainset)
# Test the algorithm on the testset
predictions = svd_algo.test(testset)
train_dataset = merged_movies_dataset.sample(frac=0.8,
    random_state=42)
test_dataset = merged_movies_dataset.drop(train_dataset.index)
# Create a TF-IDF vectorizer for movie overviews
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix_content = tfidf_vectorizer.fit_transform(
    train_dataset['overview'])
# Get the true ratings from the test set
```

```
true_ratings = [pred.r_ui for pred in predictions]
# Get the predicted ratings from the test set
predicted_ratings = [pred.est for pred in predictions]
# Calculate MAE
mae_collaborative = mean_absolute_error(true_ratings ,
   predicted_ratings)
rmse_collaborative=mean_squared_error(true_ratings ,
   predicted_ratings)
# Calculate cosine similarity between movies based on overviews
cosine_sim_content = linear_kernel(tfidf_matrix_content ,
   tfidf_matrix_content)
# Function to generate content-based recommendations
def generate_content_recommendations(test_data , cosine_sim ,
   tfidf_vectorizer , tfidf_matrix , train_data):
    mae_values = []
    rmse_va=[]
    for idx, row in test_data.iterrows():
        # Get the movie overview for the test item
        overview = row['overview']
        # Calculate TF-IDF vector for the given item's overview
        tfidf_query = tfidf_vectorizer.transform([overview])
        # Calculate cosine similarity between the query and all
            items
        cosine_similarities = linear_kernel(tfidf_query ,
           tfidf_matrix).flatten()
        # Get indices of top 5 similar items
        similar_items = cosine_similarities.argsort()
           [-6:-1][::-1]
        # Calculate the predicted vote_average as the average
           of vote_average of the similar items
        similar_items_in_train = [item for item in
           similar_items if item in train_data.index]
        predicted_vote_average = train_data.loc[
           similar_items_in_train, 'vote_average'].mean()
        # Append the predicted vote_average to the list
        mae_values.append(predicted_vote_average)
        rmse_va.append(predicted_vote_average)
    # Calculate MAE
    mae = mean_absolute_error(test_data['vote_average'],
       mae_values)
    rmse=mean_squared_error(test_data['vote_average'], rmse_va,
        squared=False)
    return mae,rmse

# Evaluate Hybrid Model
mae_collaborative = mae_collaborative
mae_content,rmse_cont = generate_content_recommendations(
   test_dataset, cosine_sim_content, tfidf_vectorizer,
   tfidf_matrix_content, train_dataset)
# Assuming equal weights for collaborative and content-based
   recommendations
weight_collaborative = 0.9
weight_content = 0.9

# Calculate the weighted predictions for collaborative and
   content-based filtering
weighted_collaborative_predictions = [weight_collaborative *
   pred.est for pred in predictions]
weighted_content_predictions = [weight_content * mae_content] *
    len(predictions)
# Combine predictions using weighted average
hybrid_predictions = [a + b for a, b in zip(
   weighted_collaborative_predictions,
   weighted_content_predictions)]
```

```
testset_df = pd.DataFrame(testset, columns=['user_id', 'item_id
    ', 'rating'])
testset_df.reset_index(drop=True, inplace=True)
# Evaluate Hybrid Model
mae_hybrid = mean_absolute_error(testset_df['rating'],
    hybrid_predictions)
print(f'MAE␣for␣Hybrid␣Model:␣{mae_hybrid}')

# Calculate the weighted predictions for collaborative and
    content-based filtering
weighted_collaborative_predictions = [weight_collaborative *
    pred.est for pred in predictions]

weighted_content_predictions = [weight_content * rmse_cont] *
    len(predictions)
# Combine predictions using weighted average
hybrid_predictions = [a + b for a, b in zip(
    weighted_collaborative_predictions,
    weighted_content_predictions)]
# Evaluate Hybrid Model
mae_hybrid = mean_squared_error(testset_df['rating'],
    hybrid_predictions,squared=False)
print(f'MAE␣for␣Hybrid␣Model:␣{mae_hybrid}')
```