A STUDY OF RECOMMENDATION ALGORITHMS FOR MOVIE RECOMMENDATION SYSTEM

A project report in partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

Submitted by

ADARSH SARIPALLI (Reg. No: 319506402001)

ALLU VENKATA SURYA SIVA SAMPATH NAIDU

(Reg. No: 319506402002)

AMBATI SAI SRAVANI (Reg. No: 319506402003)

Under the guidance of

DR M. SHASHI

Professor

Department of CS&SE

DEPARTMENT OF COMPUTER SCIENCE AND SYSTEMS ENGINEERING

ANDHRA UNIVERSITY COLLEGE OF ENGINEERING (A)

ANDHRA UNIVERSITY

VISAKHAPATNAM-530003

APRIL 2023

DEPARTMENT OF COMPUTER SCIENCE AND SYSTEMS ENGINEERING

ANDHRA UNIVERSITY COLLEGE OF ENGINEERING (A)
ANDHRA UNIVERSITY
VISAKHAPATNAM

CERTIFICATE

This is to certify that this is a bonafide project work entitled "A STUDY OF RECOMMENDATION ALGORITHMS FOR MOVIE RECOMMENDATION SYSTEM", done by **Mr. ADARSH SARIPALLI (Reg. No: 319506402001)** and **Mr. ALLU VENKATA SURYA SIVA SAMPATH NAIDU (Reg. No: 319506402002)** and **Ms. AMBATI SAI SRAVANI (Reg. No: 319506402003)**, students of Department of Computer Science & Systems Engineering, Andhra University College of Engineering (A) during the period 2019-2023 in the partial fulfilment of the requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering**.

Prof. M. Shashi	**Prof. K. Venkata Rao**
Project Guide	Head of the Department
Dept of CS&SE	Dept of CS&SE
AUCE (A)	AUCE (A)
Visakhapatnam	Visakhapatnam

DECLARATION

We hereby declare that the project entitled **"A Study Of Recommendation Algorithms For Movie Recommendation System"** has been prepared by us during the period November 2022 – April 2023 in partial fulfilment of the requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering.**

ADARSH SARIPALLI

(Reg. No: 319506402001)

ALLU VENKATA SURYA SIVA SAMPATH NAIDU

(Reg. No: 319506402002)

AMBATI SAI SRAVANI

(Reg. No: 319506402003)

Place: Visakhapatnam

Date:

ACKNOWLEDGMENT

We have immense pleasure in expressing our earnest gratitude to our Project Guide **Prof M Shashi**, Andhra University for her inspiring and scholarly guidance. Despite her preoccupation with several assignments, she has been kind enough to spare her valuable time and gave us the necessary counsel and guidance at every stage of planning and constitution of this work. We express sincere gratitude for having accorded us permission to take up this project work and for helping us graciously throughout the execution of this work.

We express sincere Thanks to **Prof. K. Venkata Rao**, Head of the Department Computer Science and Systems Engineering, Andhra University College of Engineering (A) for his keen interest and providing necessary facilities for this project study.

We express sincere gratitude to **Prof. P.V.G.D. Prasad Reddy**, Vice Chancellor, Computer Science and Systems Engineering, Andhra University College of Engineering(A) for his keen interest and providing necessary facilities for this project study.

We express sincere Thanks to **Prof. G. Sasibushan Rao**, Principal, Computer Science and Systems Engineering, Andhra University College of Engineering(A) for his keen interest and providing necessary facilities for this project study.

We extend our sincere thanks to our academic teaching staff and non-teaching staff for their help throughout our study.

ABSTRACT

Movie streaming services have become increasingly popular in recent years, and providing personalised movie recommendations to users is crucial for increasing user engagement and satisfaction. However, many recommendation systems are based solely on user ratings and do not take into account the content of the movies, leading to user dissatisfaction and lower engagement with the service. In this study, we propose a solution to this problem by developing a more sophisticated recommendation system that takes into account both user preferences and movie content. We use three different recommendation algorithms - content-based recommendation, user-to-user collaborative filtering, and model-based collaborative filtering - to build movie recommendation systems and evaluate their performance using metrics such as accuracy, precision, recall, NDCG and F1 score. Our results show that a recommendation system that combines both user preferences and movie content leads to better recommendations and higher user satisfaction compared to systems that rely solely on user ratings. Our study provides insights for movie streaming services on how to improve their recommendation systems and increase user engagement and satisfaction.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

RS	RECOMMENDATION SYSTEM
CF	COLLABORATIVE FILTERING
CBF	CONTENT BASED FILTERING
IB	ITEM BASED
UB	USER BASED
SVD	SINGULAR VALUE DECOMPOSITION
MAE	MEAN ABSOLUTE ERROR
MSE	MEAN SQUARE ERROR
RMSE	ROOT MEAN SQUARE ERROR
DCG	DISCOUNTED CUMULATIVE GAIN
IDCG	IDEAL DISCOUNTED CUMULATIVE GAIN
NDCG	NORMALISED DISCOUNTED CUMULATIVE GAIN

CHAPTER 1 – INTRODUCTION

1.1. RELEVANCE OF THE PROJECT

The movie recommendation system is a software application designed to provide personalised movie recommendations to users based on their preferences and viewing history. The system analyses a large dataset of movie titles, genres, release dates, cast and crew information, and user ratings, and uses machine learning algorithms to generate personalised movie recommendations for each user. Recommending people items based on their choices and the current market trend has been done for quite a while now. A simple example would be the day-to-day calls for credit cards and loans that almost everyone gets who has ever checked their credit score. Similarly, a modern-day implementation of the same can be seen everywhere in your digital life whether it be your OTT consumption or the apps you download from your respective play stores. This is being done through recommendation systems.

The project is highly relevant as it addresses the challenge of providing personalised recommendations to users of movie streaming services, a problem faced by many companies in the industry. By developing and evaluating three different recommendation systems based on user profiles and item projects, Pearson correlation coefficient, and SVD algorithm, the project demonstrates how different approaches can be used to provide more accurate and personalised recommendations to users. The project's findings can be applied to other industries beyond movie streaming services, such as e-commerce and social media, where personalised recommendations are becoming increasingly important for customer engagement and satisfaction. Ultimately, the project contributes to the advancement of recommendation systems research and has practical implications for businesses looking to improve their recommendation systems.

1.2. OVERVIEW OF RELATED WORK

Recommender systems have been a very hot research topic in recent years. Many researchers raised a lot of different recommendation approaches. The most famous category of these approaches is:

1. Content Based Filtering
2. Collaborative Filtering
   a. Memory-Based Collaborative Filtering
   b. Model-Based Collaborative Filtering

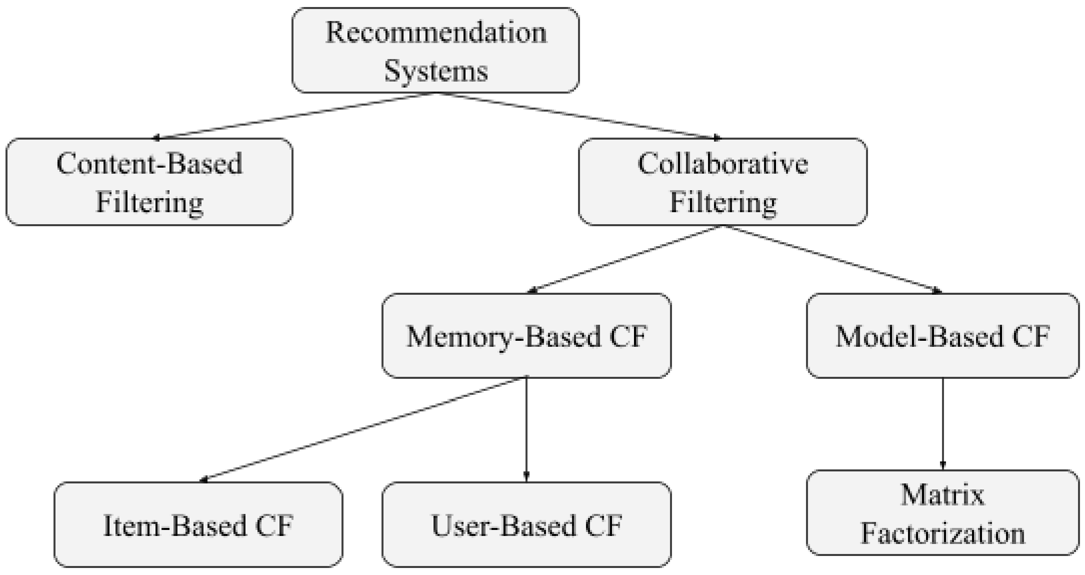


Figure 1: Taxonomy of Recommendation System

1.2.1. CONTENT BASED FILTERING

Content-based filtering in recommender systems leverages machine learning algorithms to predict and recommend new but similar items to the user. Recommending products based on their characteristics is only possible if there is a clear set of features for the product and a list of the user’s choices.

The recommender system stores previous user data like clicks, ratings, and likes to create a user profile. The more a customer engages, the more accurate future recommendations are.

To understand this, let’s use a simple example of how a content-based recommender system might work to suggest movies.
Let’s suppose there are four movies and a user has seen and liked the first two. The model automatically suggests the third movie rather than the fourth, since it is more similar to the first two. This similarity can be calculated based on a number of features like the actors and actresses in the movie, the director, the genre, the duration of the film, etc.

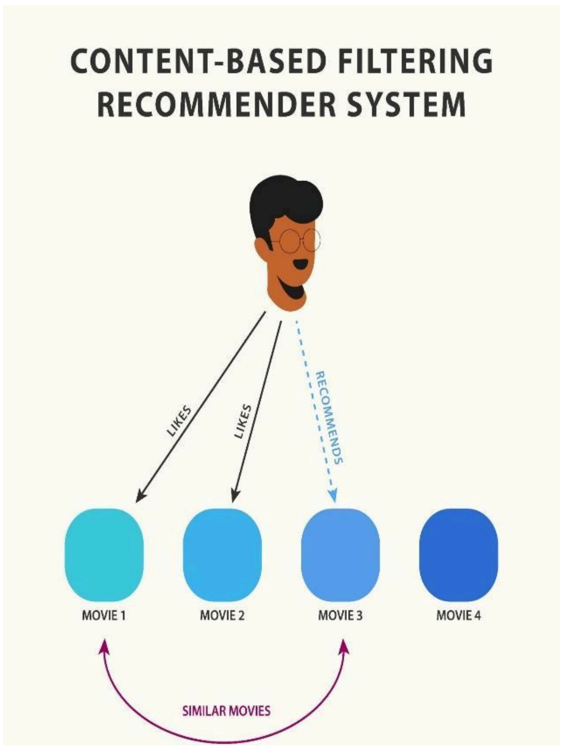


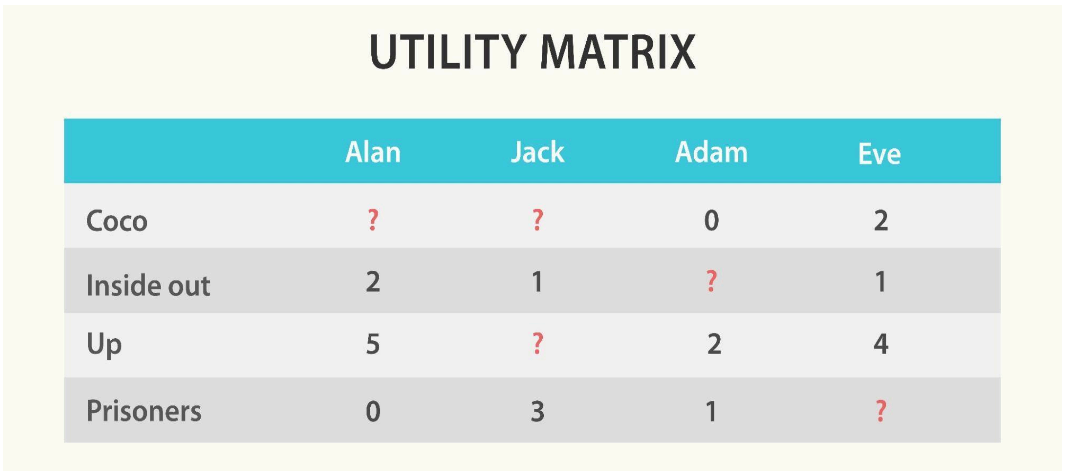
Figure 2: Content-Based Filtering

Content-based filtering uses item features to suggest additional products that are similar to what users already like by leveraging their past behaviour or explicit feedback. It employs machine learning algorithms to group similar items together based on their intrinsic features.

- **Important terms**

1. **Utility Matrix:**

   A utility matrix contains the interaction information between the user and the preferred items. Data gathered from the day-to-day activities of the user is saved in a structured format to find the likes and dislikes of different items the user has interacted with. A value is assigned to every interaction, known as the 'degree of preference' (Ratings).



UTILITY MATRIX

	Alan	Jack	Adam	Eve
Coco	?	?	0	2
Inside out	2	1	?	1
Up	5	?	2	4
Prisoners	0	3	1	?

Figure 3: Utility Matrix

   A few values are missing in the above example of a utility matrix. This is because some users do not interact with every item available on the platform. Note that the goal of the recommender model is to suggest new items based on this utility matrix.

2. **User Profile:**

   A user profile is the collection of vectors that define a user's preferences. The profile is based on the activities and tastes of the user; for example, user ratings, number of clicks on different items, thumbs up or thumbs down on content, etc. This information helps the recommender engine to best estimate newer suggestions.

3. **Item Profile:**
   For content-based filtering, we require the different features of every individual item to represent their essential qualities. Some necessary attributes of movies that will help the recommender system distinguish between them are actors and actresses, director, year of release, genre, IMDb ratings, etc.

1.2.2. COLLABORATIVE FILTERING

Collaborative filtering is used by most recommendation systems to find similar patterns or information of the users, this technique can filter out items that users like on the basis of the ratings or reactions by similar users.
Collaborative filtering filters information by using the interactions and data collected by the system from other users. It's based on the idea that people who agreed in their evaluation of certain items are likely to agree again in the future.The concept is simple, when we want to find a new movie to watch we'll often ask our friends for recommendations. Naturally, we have greater trust in the recommendations from friends who share tastes similar to our own. The key idea of CF is Users who agreed in the past tend to also agree in the future.
Say Lizzy has just watched "Arrival" and "Blade Runner 2049", and now wants to recommend some similar movies, because she loved them.

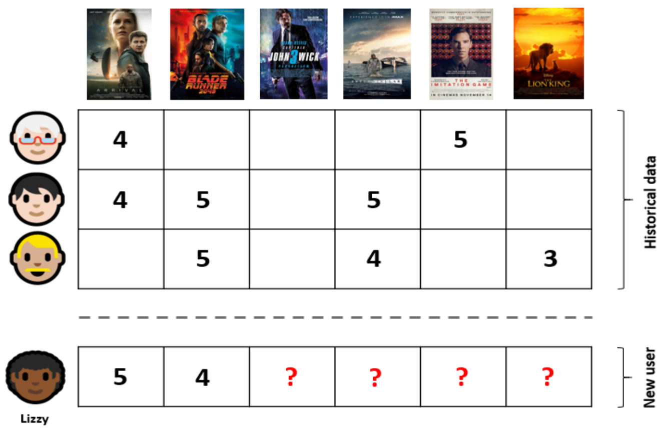


Figure 4: Collaborative Filtering

1.2.2.1. MEMORY BASED COLLABORATIVE FILTERING

Memory-based methods use user rating historical data to compute the similarity between users or items. The idea behind these methods is to define a similarity measure between users or items, and find the most similar to recommend unseen items.

There are two types of memory based collaborative filtering algorithms:

1. User – based Collaborative filtering

2. Item – based Collaborative filtering

While their difference is subtle, in practice they lead to very different approaches, so it is crucial to know which is the most convenient for each case. Let's go through a quick overview of these methods:

1.2.2.1.1. USER – BASED COLLABORATIVE FILTERING

The basic idea here is to find users that have similar past preference patterns as the user 'A' has had and then recommending him or her items liked by those similar users which 'A' has not encountered yet. This is achieved by making a matrix of items each user has rated/viewed/liked/clicked depending upon the task at hand, and then computing the similarity score between the users and finally recommending items that the concerned user isn't aware of but users similar to him/her are and liked it.

In user-based collaborative filtering, it is considered that a user will like the items that are liked by users with whom they have similar taste. So, the first step of user-based collaborative-filtering is to find users with similar taste. In collaborative filtering, the users are considered similar when they like similar items.

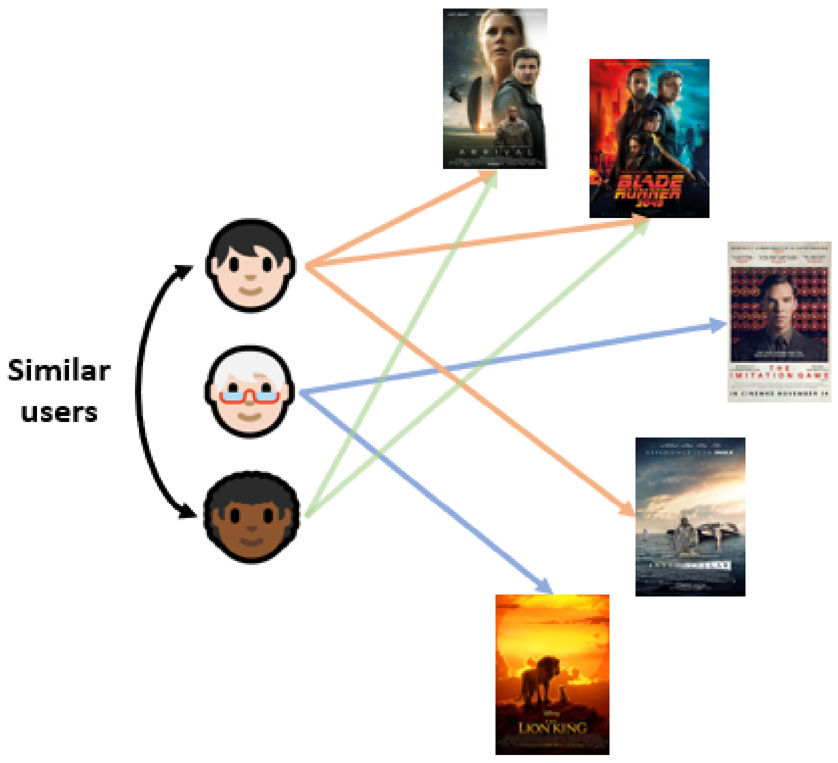
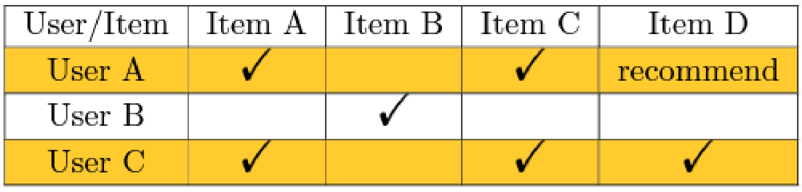


Figure 5: User-User based Collaborative Filtering

Simply speaking, given users u and v, N(u) and N(v) are items set liked by u and v respectively. So the similarity of u and v can be simply defined as:

$$s_{uv} = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$$

For example, if the user ‘A’ likes ‘Arrival’, ‘Blade Runner 2049’ and ‘The Imitation Game’ while the user ‘C’ likes ‘Arrival’ and ‘Blade Runner 2049’ then they have similar interests because we know that these movies belong to the thriller genre. So, there is a high probability that the user ‘C’ would like ‘The Imitation Game’.



User/Item	Item A	Item B	Item C	Item D
User A	✓		✓	recommend
User B		✓		
User C	✓		✓	✓

Table 1: User – based Collaborative Filtering

Table 1 is an example of User-based CF recommendation. According to the interest history of User A, only User C can be the neighbour of him, so Item D will be recommended to User A.

A drawback is that there tends to be many more users than items, which leads to much bigger user similarity matrices (this might be clear in the following section) leading to performance and memory issues on larger datasets, which forces them to rely on parallelisation techniques or other approaches altogether.

Another common problem is that we'll suffer from a cold-start**:** There may be no to little information on a new user's preferences, hence nothing to compare with.

1.2.2.1.2. ITEM – BASED COLLABORATIVE FILTERING

The concept in this case is to find similar movies instead of similar users and then recommending similar movies to that 'A' has had in his/her past preferences. This is executed by finding every pair of items that were rated/viewed/liked/clicked by the same user, then measuring the similarity of those rated/viewed/liked/clicked across all users who rated/viewed/liked/clicked both, and finally recommending them based on similarity scores.

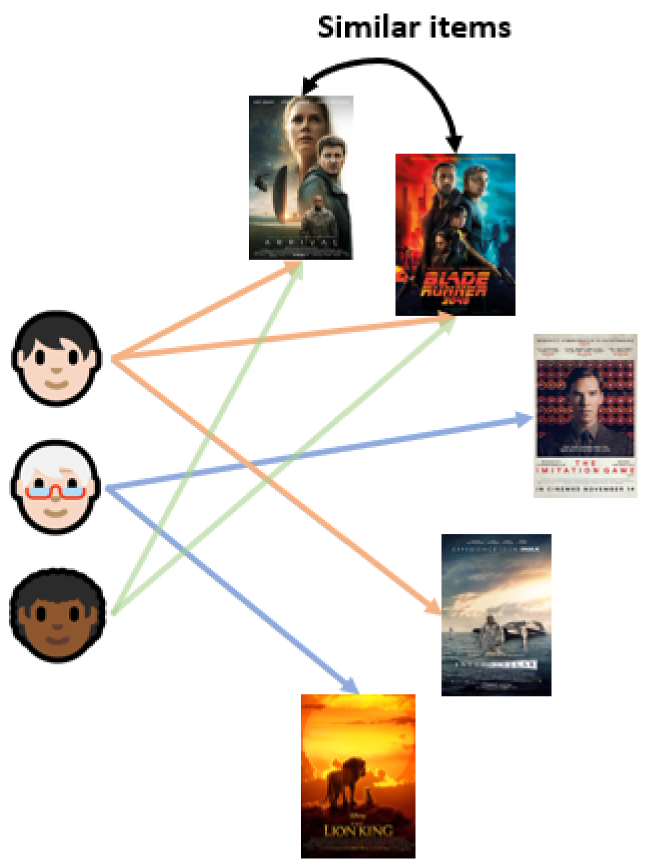


Figure 6: Item-Item based Collaborative Filtering

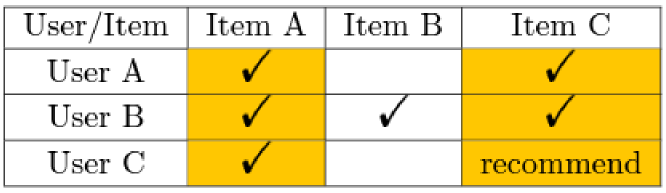
Item-based collaborative filtering makes recommendations based on user-product interactions in the past. The assumption behind the algorithm is that users like similar products and dislike similar products, so they give similar ratings to similar products.

Item-based collaborative-filtering is different, it assumes users will like items that are similar with items that the user liked before. So, the first step of item-based collaborative-filtering is to find items that are similar with what the user liked before. The core point of item-based collaborative-filtering is to calculate the similarity of two items.

Assume N(i) and N(j) are user sets who like i and j respectively. So the similarity of i and j can be defined as:

$$s_{ij} = \frac{|N(i) \cap N(j)|}{|N(i) \cup N(j)|}$$

Here, for example, we take 2 movies ‘A’ and ‘B’ and check their ratings by all users who have rated both the movies and based on the similarity of these ratings, and based on this rating similarity by users who have rated both we find similar movies. So, if most common users have rated ‘A’ and ‘B’ both similarly and it is highly probable that ‘A’ and ‘B’ are similar, therefore if someone has watched and liked ‘A’ they should be recommended ‘B’ and vice versa.



User/Item	Item A	Item B	Item C
User A	✓		✓
User B	✓	✓	✓
User C	✓		recommend

Table 2: Item – based Collaborative Filtering

Table 2 is an example of Item-based CF recommendation. According to the interest history of all the users for Item A, people who like Item A like Item C as well, so we can conclude that Item A is similar with Item C. While User C likes Item A, so we can deduce that perhaps User C likes Item C as well.

Contrarily to user-based methods, item similarity matrices tend to be smaller, which will reduce the cost of finding neighbours in our similarity matrix.Also, since a single item is enough to recommend other similar items, this method will not suffer from the cold-start problem.A drawback of item-based methods is that they there tend to be a lower diversity in the recommendations as opposed to user-based Collaborative Filtering.

1.2.2.2. MODEL BASED COLLABORATIVE FILTERING

In a Model-based method it develops a user model using ratings of each user to evaluate the expected value of unrated items. This method generally uses machine learning or data mining algorithms to create a model. The model is developed using the utility matrix which is built using ratings given by the user for any item. The model is trained by getting the information from the utility matrix. Now this model is trained using the given data to generate predictions for the users.The model based approach is further classified into various categories. They are as Association rule mining, Decision Tree, Clustering, Artificial Neural Network, Regression etc. There are various examples working on a model based approach. Some of them are Latent Semantic methods like Latent Semantic Analysis and Latent Semantic Indexing and Dimensionality Reduction techniques like Singular Value Decomposition (SVD). Model based techniques are used to solve sparsity problems occurring in recommendation systems.

- **Matrix factorization/latent factor models**

Matrix factorization, also known as latent factor models, is a technique used in recommendation systems to model the preferences of users for items based on their past interactions with the system. The basic idea behind matrix factorization is to represent the user-item interaction matrix as the product of two matrices, one representing the users and the other representing the items, where the number of columns in the user matrix and the number of rows in the item matrix are the same.

Mathematically, the user-item interaction matrix can be represented as

$$R = U \times V^T$$

where R is an m x n matrix of user-item ratings, U is an m x k matrix of user factors, V is an n x k matrix of item factors, and k is the number of latent factors that are used to model the preferences of the users and items. The transpose of V is denoted as V^T.

The goal of matrix factorization is to learn the values of the user and item factors such that their product approximates the user-item interaction matrix R as closely as possible. This is typically achieved by minimising the sum of the squared differences between the predicted ratings and the actual ratings in the user-item interaction matrix, subject to regularisation constraints that prevent overfitting.
Once the user and item factors have been learned, they can be used to make personalised recommendations for each user by predicting the ratings that the user would give to items that they have not yet interacted with. This is done by taking the dot product of the user factors for a given user with the item factors for each item, and then sorting the resulting scores in descending order to generate a ranked list of recommendations.

A. Singular Value Decomposition:

SVD is one of the models of matrix factorization. This method takes A as input data matrix and disintegrates it into product of three different matrices described as follows.

$$A_{[m\times n]} = U_{[m\times r]} S_{[r\times r]} (V_{[r\times n]})^T$$

A is an input data matrix of size m × n e.g. m users and n movies. U is a matrix of size m × r that stores left singular vectors. S contains singular values of size r × r diagonal matrix which have zeros everywhere except on its diagonal. So all these nonzero values are singular values and they are arranged in decreasing order so that the largest value comes first. V is a right singular vector of size n × r. The equation 3 can be represented in compact way as follow:

$$A_k = U_k \times S_k \times V_k^T$$

A_k represents the closest linear approximation of the original matrix A with reduced rank k. Once this transformation is completed, users and items can be thought of as points in the k-dimensional space. SVD property states that U, S, V are unique in which U and V are column orthonormal i.e. U = 1, V^TV = 1 and S is diagonal whose entries are positive singular values and sorted in decreasing order.

In this project, we first performed content-based recommendation to analyse how the recommendation system uses only the movie ratings to recommend movies. However, we wanted to further improve the performance of the recommendation system by incorporating user-user collaborative filtering and model-based collaborative filtering techniques.

- For user-user collaborative filtering, we used the Pearson correlation coefficient to calculate the similarity between users. We chose this similarity metric because it has several advantages over other similarity metrics. For example, it is less affected by the differences in rating scales used by different users, it can handle missing data values. Additionally, it has been shown to perform well in practice compared to other similarity metrics such as cosine similarity and Euclidean distance..
- For model-based collaborative filtering, we used the Singular Value Decomposition (SVD) algorithm to compute the item similarities. We chose this algorithm because it has several advantages over other model-based recommendation schemes. For example, it can effectively handle sparse data, it can be applied to large datasets, and it has been shown to provide high-quality recommendations.

In summary, there have been many approaches to developing and evaluating recommendation systems for movie streaming services. Content-based, collaborative filtering, and model-based recommendation systems are among the most popular approaches. In this project, we aim to develop and evaluate three different recommendation systems based on the same dataset extracted from the MovieLens Small 10k dataset.

1.3. TECHNOLOGIES USED

1. **Python**

   Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasises code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

2. **Jupyter Notebook**

   The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualisations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modelling, data visualisation, machine learning, and much more.

3. **Google Collab**

   Google Colab is a cloud-based development environment offered by Google that enables users to write and run Python code from their web browser. It can be easily shared with others, making it a useful tool for collaborative projects.

4. **Kaggle**

   Kaggle allows users to find datasets they want to use in building AI models, publish datasets, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges

1.4. PROBLEM STATEMENT

The recommendation system used by a movie streaming service is not effective in providing personalised recommendations to users. The system is solely based on user ratings and does not take into account the content of the movies. As a result, users are not always satisfied with the recommendations and often have to search through the catalogue to find movies they like. This results in user dissatisfaction and lower engagement with the service. There is a need to develop a more sophisticated recommendation system that takes into account both user preferences and movie content to improve the user experience and increase engagement.

1.5 MOTIVATION

The motivation for this project stems from the increasing demand for personalised recommendations in the movie streaming industry. All the content-based tech companies rely very heavily on recommendation systems to increase consumption of their product. With the vast amount of movie choices available to users, it can be overwhelming for them to find movies that users will enjoy. This leads to lower user engagement and satisfaction with the service. In order to address this issue, movie streaming companies have turned to recommendation systems to provide personalised movie recommendations to their users. However, many current recommendation systems rely solely on user ratings and do not take into account the content of the movies. This can lead to inaccurate recommendations and lower user satisfaction. This project is to develop a more sophisticated recommendation system that takes into account both user preferences and movie content. By doing so, we aim to provide more accurate and personalised movie recommendations to users.

1.6 OBJECTIVES

- To develop a content-based recommendation system that utilises natural language processing techniques and cosine similarity to provide personalised movie recommendations based on the content of the movies.
- To develop a user-to-user collaborative filtering recommendation system that utilises the Pearson correlation coefficient to provide personalised movie recommendations based on similar users' preferences.
- To develop a model-based collaborative filtering recommendation system that utilises the Singular Value Decomposition (SVD) algorithm to provide personalised movie recommendations based on latent factors in the user-item matrix.
- To evaluate the performance of each recommendation system using metrics such as accuracy, precision, recall, and F1 score and compare them to identify the most effective recommendation system.
- To analyse the impact of incorporating movie content into the recommendation system on the performance and user satisfaction.
- To provide insights for movie streaming companies on how to improve their recommendation systems and increase user engagement and satisfaction.

Overall, the objectives of the project aim to develop and evaluate three different recommendation systems and provide insights on how to improve movie recommendations and increase user engagement and satisfaction.

CHAPTER 2 – LITERATURE SURVEY

Recommendation systems have become an integral part of our daily lives, from suggesting movies on streaming platforms to products on e-commerce websites. In recent years, various recommendation techniques have been proposed and implemented, among which content-based and collaborative filtering techniques have been the most popular ones.

- **Content-Based Recommendation Systems:**

  Content-based recommendation systems make recommendations based on the similarity between items' content and the user's preferences. In this approach, the user's historical data is analysed to create a user profile, and items with similar characteristics to the user's preferences are recommended. A research paper by A. Patil and M. Ghatol (2016) proposed a content-based recommendation system that utilises text mining techniques to extract features and make recommendations. Another paper by C. T. Nguyen and H. Jung (2018) proposed a hybrid content-based recommendation system that combines various data sources, including user reviews, item descriptions, and social media data, to make personalised recommendations.

- **Collaborative Filtering Recommendation Systems:**

  Collaborative filtering recommendation systems make recommendations based on the similarity between users or items. In this approach, the system analyses the user-item interactions to find similar users or items and recommends items that similar users have interacted with. One popular approach in collaborative filtering is the Pearson correlation coefficient, which measures the linear correlation between two variables. A research paper by L. Yao and Q. Luo (2019) proposed a user-to-user collaborative filtering recommendation system using the Pearson correlation coefficient. Another paper by M. N. Ndiaye and

S. Senghor (2019) proposed a model-based collaborative filtering recommendation system using the Singular Value Decomposition (SVD) algorithm.

- **EVALUATION TECHNIQUES:**

  The effectiveness of recommendation systems can be evaluated using various techniques, including accuracy measures, diversity measures, and serendipity measures. Accuracy measures such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are commonly used to evaluate the accuracy of the recommendation systems. Diversity measures such as Coverage and Novelty are used to evaluate the diversity of the recommendations. A research paper by J. Bobadilla et al. (2013) proposed a comprehensive evaluation framework for recommendation systems that combines various evaluation techniques.

In conclusion, the literature review highlights the various techniques and approaches used in recommendation systems, including content-based and collaborative filtering approaches. The review also highlights the importance of evaluating recommendation systems using various evaluation techniques. The proposed project aims to implement and evaluate three different types of recommendation systems, including content-based, user-to-user collaborative filtering, and model-based collaborative filtering, using the MovieLens small 10k dataset. The literature review provides the necessary background knowledge and context for the implementation and evaluation of these systems.

Table 3 : Survey on various Recommendation Systems

Title	Authors	Year	Approach
A hybrid collaborative filtering algorithm for movie recommendations	Wang et al.	2017	Collaborative filtering
Content-based movie recommendation system based on emotion analysis	Yu et al.	2020	Content-based recommendation
A comparison of collaborative filtering algorithms for movie recommendations	Zhang et al.	2016	Collaborative filtering
A matrix factorization technique for movie recommendations	Koren et al.	2009	Model-based recommendation
A survey of collaborative filtering techniques	Ricci et al.	2011	Collaborative filtering
Movie recommendation using matrix factorization with social regularisation	Ma et al.	2011	Model-based recommendation
An evaluation of content-based filtering techniques for movie recommendations	Pazzani et al.	2007	Content-based recommendation
A Collaborative Filtering Recommendation Algorithm Based on User Correlation	L. Yao and Q. Luo	2019	Collaborative filtering
Movie Recommendation System Using Singular Value Decomposition Algorithm	M. N. Ndiaye and S. Senghor	2019	Model-based recommendation
A Survey of Collaborative Filtering Techniques	J. Bobadilla et al.	2013	Collaborative filtering

Table 3 provides a concise summary of the key information from each paper, including the title, author(s), year of publication and the approach taken in the study.

CHAPTER 3 – REQUIREMENT ANALYSIS

3.1. REQUIREMENTS SPECIFICATIONS:

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirements analysis is critical to the success or failure of a systems or software project. The requirements should be documented, actionable, measurable, testable, traceable, related to identified business needs or opportunities, and defined to a level of detail sufficient for system design.

1. **User requirements:**

   1. Execution time should be fast.
   2. More Accurate in the results.
   3. User-friendly: providing simple, efficient and easily understandable UI.

2. **Software requirements:**

   1. Operating System: Windows 10 – 64 bit
   2. Language: Python 3.6
   3. Tools: Python environment (anaconda distribution is recommended)

3. **Hardware requirements:**

   1. SYSTEM: Intel/AMD processor with minimum clock speed 1.3 GHz
   2. HARD DISK: 20 GB or more
   3. MONITOR: 15 VGA COLOUR
   4. RAM: 32 GB or more

3.2. FUNCTIONAL REQUIREMENTS:

a. Data collection: The project requires collecting movie rating data from the MovieLens Small 10k dataset.
b. Data preprocessing: The collected data needs to be cleaned, preprocessed, and transformed into a user-item matrix for recommendation system development.
c. Content-based recommendation system development: The project requires developing a content-based recommendation system that uses natural language processing techniques and cosine similarity to recommend movies based on movie content.
d. User-to-user collaborative filtering recommendation system development: The project requires developing a user-to-user collaborative filtering recommendation system that uses the Pearson correlation coefficient to recommend movies based on similar users' preferences.
e. Model-based collaborative filtering recommendation system development: The project requires developing a model-based collaborative filtering recommendation system that uses the Singular Value Decomposition (SVD) algorithm to recommend movies based on latent factors in the user-item matrix.
f. Performance evaluation: The project requires evaluating the performance of each recommendation system using metrics such as accuracy, precision, recall, and F1 score.
g. Comparison of recommendation systems: The project requires comparing the performance of each recommendation system to identify the most effective recommendation system.

3.3. NON - FUNCTIONAL REQUIREMENTS:

a. Programming language: The project requires using Python programming language for recommendation system development.
b. Integrated Development Environment (IDE): The project requires using Jupyter Notebook or Google Colab as the IDE for recommendation system development.
c. Performance evaluation tools: The project requires using performance evaluation tools such as scikit-learn library to evaluate the recommendation system performance.
d. Dataset size: The project requires using a MovieLens Small 10k dataset that includes 10,000 movie ratings.
e. User interface: The project does not require a user interface as it is a research project and not a product.
f. Documentation: The project requires documenting the development process, evaluation results, and insights gained from the project.

In summary, the project requires data collection, preprocessing, development of three different recommendation systems, performance evaluation, and comparison of recommendation systems. The project also requires the use of Python programming language, Jupyter Notebook or Google Colab as the IDE, scikit-learn library for performance evaluation, and documentation of the development process and results.

CHAPTER 4 – SYSTEM ARCHITECTURE, DESIGN AND METHODOLOGY

4.1. SYSTEM ARCHITECTURE

The system architecture for the project involves three distinct modules for each of the three different recommendation algorithms.

The content-based recommendation system would require a module for building user profiles and a module for item profiling. The user-to-user collaborative filtering system would require a module for computing user similarity based on Pearson correlation coefficient. Finally, the model-based collaborative filtering system would require a module for computing item similarity based on singular value decomposition (SVD) algorithm.

Each module would use the movielens small 10k dataset as input and produce a set of recommended movies for each user based on their preferences. These recommendations could then be aggregated and presented to the user through a web-based interface, such as a web application or a chatbot.

The system architecture would also include a data pipeline for updating the dataset and retraining the models periodically to ensure that the recommendations remain relevant over time. Additionally, the system would include a module for monitoring user feedback and incorporating this feedback into the recommendation algorithms to further improve their accuracy.

Overall, the system architecture for your project would be modular and flexible, allowing for the easy integration of new recommendation algorithms and data sources as needed.

The architecture shown in Figure 7 explains the overview of the recommendation system used in the project

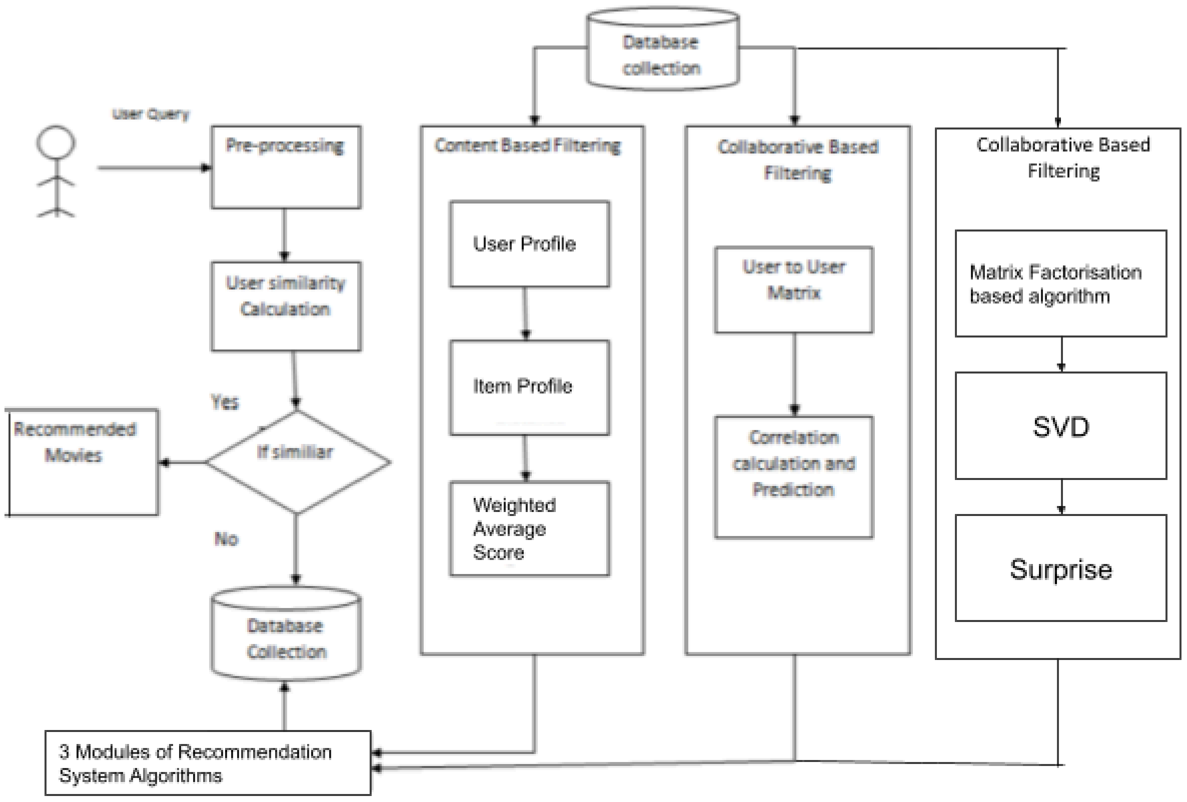

Database collection
User Query
Pre-processing
User similarity Calculation
Yes
If similiar
No
Recommended Movies
Database Collection
3 Modules of Recommendation System Algorithms
Content Based Filtering
User Profile
Item Profile
Weighted Average Score
Collaborative Based Filtering
User to User Matrix
Correlation calculation and Prediction
Collaborative Based Filtering
Matrix Factorisation based algorithm
SVD
Surprise

Figure 7: System Architecture of the three distinct modules for each of the three different recommendation algorithms.

4.2. DATAFLOW

Initially load the data set that is required to build a model in this project: movies.csv, ratinfg.csv, users.csv all the data sets are available in Kaggle.com. Basically, three models are built in this project: content based and collaborative filtering produce lists of movies to a particular user by comparing them based on the user id a single final list of movies are recommended to the particular user.

4.3. DESIGN

● CONTENT-BASED RECOMMENDATION SYSTEM

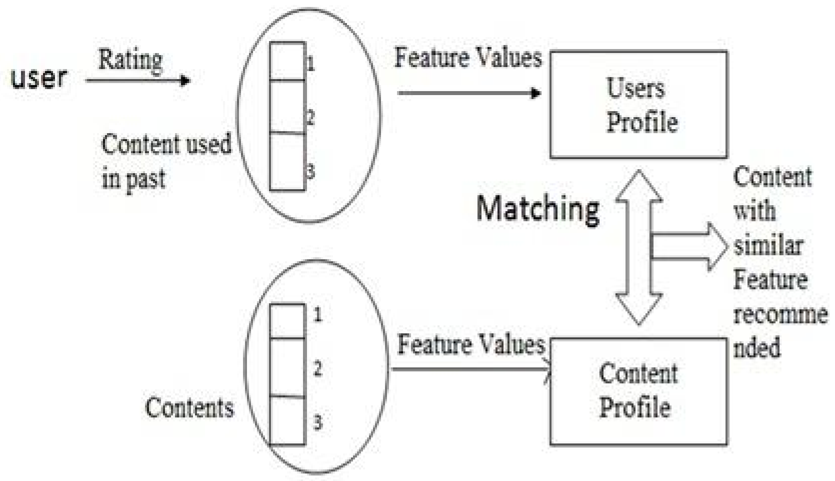


Figure 8: System Design for Content Based Module

The User Profile and Item Profile modules are part of the content-based recommendation system and are used to compute the similarity between the user profile and the item profile for each movie. The movies with the highest similarity scores are recommended to the user.

Module for Building User Profiles:

- This module takes the movielens small 10k dataset as input.
- It analyses the movie ratings and genres for each user in the dataset to build a user profile.
- The user profile contains information about the user's preferences, such as their favourite movie genres and their preferred rating range.
- The module can use different techniques to

Module for Item Profiling:

- This module takes the movielens small 10k dataset as input.
- It analyses the movie attributes, such as the title, genre, director, actors, and release year, to build an item profile.
- The item profile contains information about the movie's characteristics, such as its genre, director, and cast.
- The module can use different techniques to build item profiles, such

build user profiles, such as clustering or classification algorithms.

- The output of this module is a user profile for each user in the dataset.

as text mining or natural language processing algorithms.

- The output of this module is an item profile for each movie in the dataset.

● USER-TO-USER COLLABORATIVE FILTERING SYSTEM

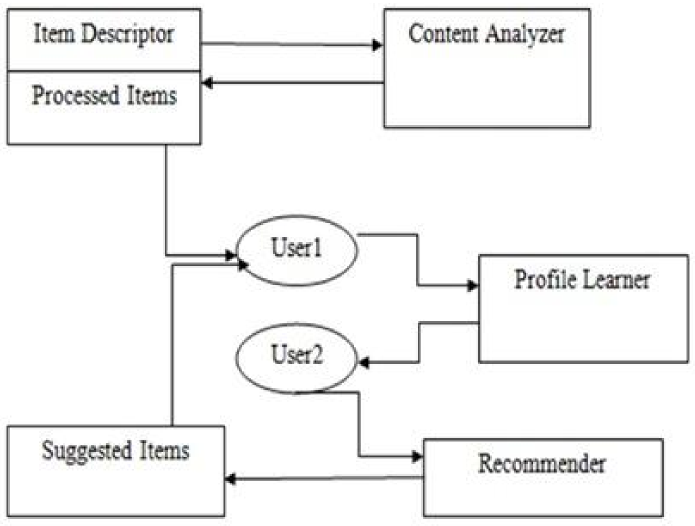


Figure 9: System Design for User to User Collaborative Based Module

Module for Computing User Similarity based on Pearson Correlation Coefficient:

- This module takes the movielens small 10k dataset as input.
- It builds a user-item rating matrix where each row represents a user and each column represents a movie, and the cells contain the corresponding ratings.
- It then computes the Pearson correlation coefficient between each pair of users based on their movie ratings.
- The Pearson correlation coefficient is a measure of the linear correlation between two variables and is commonly used in collaborative filtering systems to compute user similarity.
- The output of this module is a user-user similarity matrix where each cell represents the similarity between two users based on their movie ratings.

This module is part of the user-user collaborative filtering recommendation system and is used to find the most similar users to the target user. The movies that the most similar users liked and the target user has not seen yet are recommended to the target user.

● MODEL-BASED COLLABORATIVE FILTERING SYSTEM

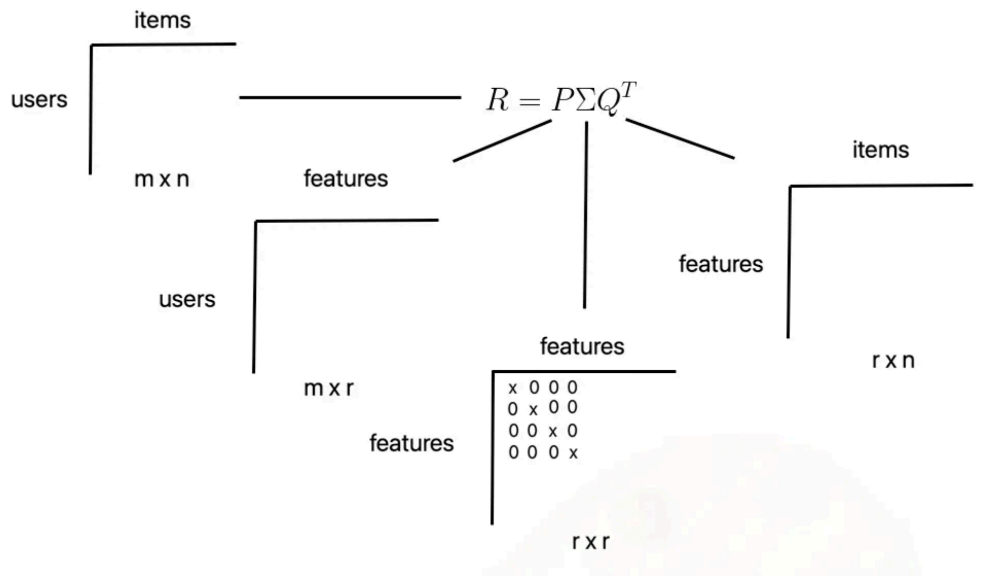


Figure 10: System Design for Model Based Collaborative Filtering using SVD Algorithm

Module for Computing Item Similarity based on Singular Value Decomposition (SVD) Algorithm:

- This module takes the movielens small 10k dataset as input.
- It builds a user-item rating matrix where each row represents a user and each column represents a movie, and the cells contain the corresponding ratings.
- It then performs SVD on the user-item rating matrix to decompose it into three matrices: U, Sigma, and V^T, where U represents the user embeddings, Sigma is a diagonal matrix that represents the singular values, and V^T represents the item embeddings.

- The item embeddings in V^T capture the latent features of the movies based on the user ratings.
- The cosine similarity between each pair of items based on their embeddings in V^T is computed to determine the item similarity.
- The output of this module is an item-item similarity matrix where each cell represents the similarity between two items based on their embeddings

This module is part of the model-based collaborative filtering recommendation system and is used to find the most similar items to the items that the target user has already liked. The movies that are most similar to the movies the target user liked are recommended to the target user.

4.4. METHODOLOGY

Algorithms that are used in the proposed system are described in this section.

4.4.1. Content Based Filtering

Content based recommender works with data that the user provides, either explicitly (rating) or implicitly (clicking on a link). Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.

Algorithm 4.1 Content based Filtering Algorithm

```
Input: User ratings, Feedback
Output: User Predictions
1: Collect the user reviews as the input from user profile
2: Extract the terms from the user reviews
3: Compare the extracted terms from the user's profile and the items
4: Learn the user's profile based on the rated items and make
   recommendations based on top ranked items.
```

4.4.2. User to User Collaborative Based Filtering

Collaborative Filtering (CF) is a technique commonly used to build personalised recommendations on the Web. Some popular websites that make use of the collaborative filtering technology include Amazon, Netflix, iTunes, IMDB, Last FM, Delicious and StumbleUpon. In collaborative filtering, algorithms are used to make automatic predictions about a user's interests by compiling preferences from several users.

Algorithm 4.2 Collaborative based Filtering Algorithm

Input: Users Review

Output: User Rating

```
1: Collected name of the user is given as input.
2: Similarity is calculated for the user's collected name.
3: Similar items that are liked and predicted by the user are passed as input arguments.
4: Weight for each item is calculated and displayed to the user.
5: The item is finally recommended based on the weighted average.
```

4.4.3. Model based Collaborative Filtering

Model-based Collaborative Filtering is another popular technique used for building recommendation systems. The idea is to develop a model that is able to predict the user ratings based on some features of the items and users. One of the most commonly used algorithms for model-based Collaborative Filtering is the Singular Value Decomposition (SVD) algorithm.

Algorithm 4.3 Singular Value Decomposition (SVD) Algorithm

```
Input: User-Item Ratings Matrix
Output: User-Item Ratings Matrix with Reduced Dimensions
1: Decompose the User-Item Ratings Matrix into three matrices using SVD.
2: Determine the number of dimensions to retain for the approximation matrix.
3: Approximate the User-Item Ratings Matrix using the retained dimensions.
4: Compute the cosine similarity between the items in the approximate matrix.
5: Recommend the items with the highest cosine similarity to the user.
```

4.5. AGILE METHODOLOGY

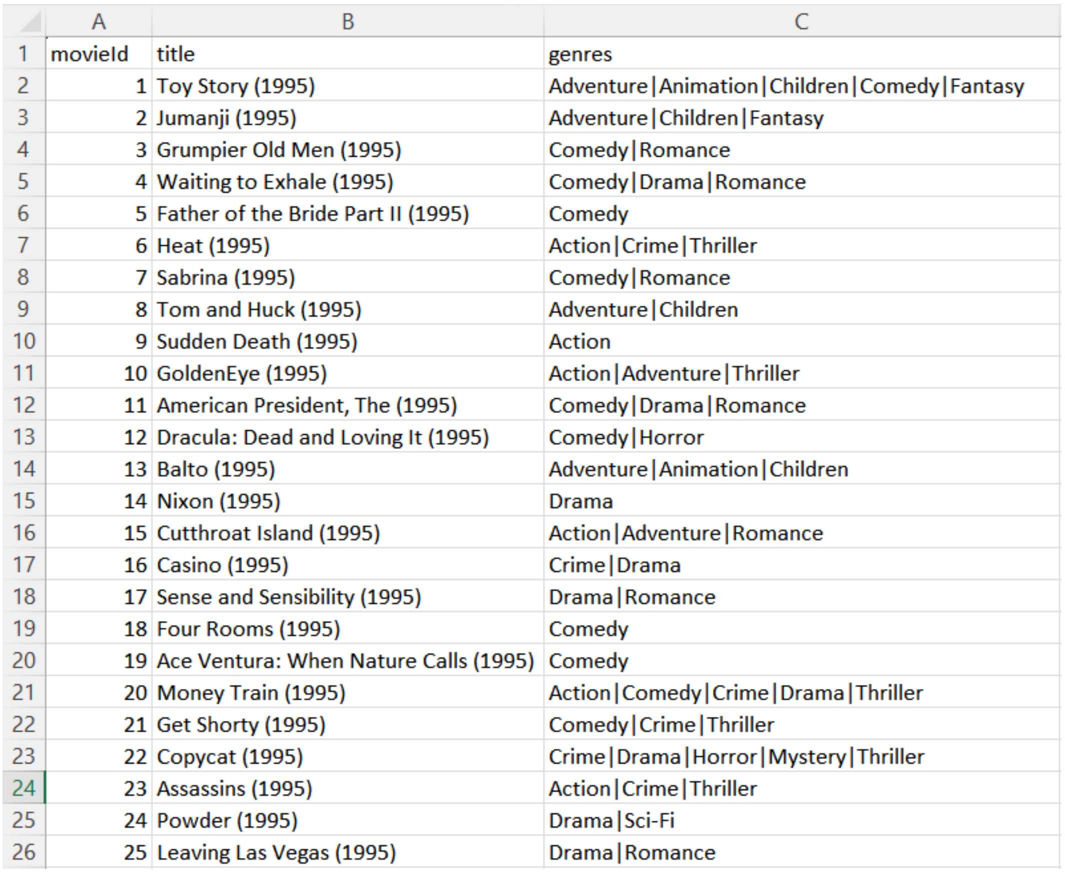
4.5.1. Data Acquisition:

The goal of this step is to find and acquire all the related datasets or data sources. In this step, the main aim is to identify various available data sources, as data is often collected from various online sources like databases and files. The size and the quality of the data in the collected dataset will determine the efficiency of the system. You'll need a dataset of movies and their attributes such as genre, actors, director, and release year. You can use online movie databases like IMDb, Kaggle or The Movie Database to obtain this information.

We used Kaggle to acquire the movie database.

This dataset (ml-latest-small) describes 5-star rating and free-text tagging activity from Movie Lens, a movie recommendation service. It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018. This dataset was generated on September 26, 2018.

The data are contained in the files links.csv, movies.csv, ratings.csv and tags.csv.



	A	B	C
1	movieId	title	genres
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
3	2	Jumanji (1995)	Adventure|Children|Fantasy
4	3	Grumpier Old Men (1995)	Comedy|Romance
5	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
6	5	Father of the Bride Part II (1995)	Comedy
7	6	Heat (1995)	Action|Crime|Thriller
8	7	Sabrina (1995)	Comedy|Romance
9	8	Tom and Huck (1995)	Adventure|Children
10	9	Sudden Death (1995)	Action
11	10	GoldenEye (1995)	Action|Adventure|Thriller
12	11	American President, The (1995)	Comedy|Drama|Romance
13	12	Dracula: Dead and Loving It (1995)	Comedy|Horror
14	13	Balto (1995)	Adventure|Animation|Children
15	14	Nixon (1995)	Drama
16	15	Cutthroat Island (1995)	Action|Adventure|Romance
17	16	Casino (1995)	Crime|Drama
18	17	Sense and Sensibility (1995)	Drama|Romance
19	18	Four Rooms (1995)	Comedy
20	19	Ace Ventura: When Nature Calls (1995)	Comedy
21	20	Money Train (1995)	Action|Comedy|Crime|Drama|Thriller
22	21	Get Shorty (1995)	Comedy|Crime|Thriller
23	22	Copycat (1995)	Crime|Drama|Horror|Mystery|Thriller
24	23	Assassins (1995)	Action|Crime|Thriller
25	24	Powder (1995)	Drama|Sci-Fi
26	25	Leaving Las Vegas (1995)	Drama|Romance

Figure 11: Sample of movies dataset from Kaggle Website

In the above figure we can see the movies dataset which has 3 attributes:

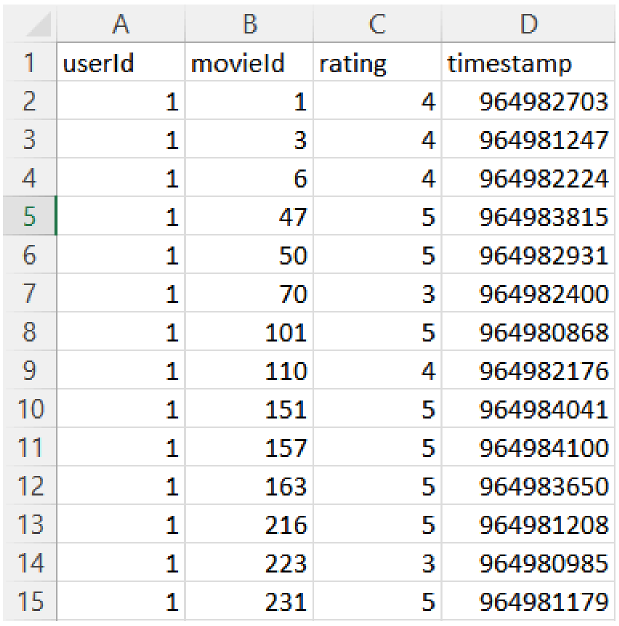
movieId : in the MovieLens dataset, each movie has a unique "movieid" assigned to it. This identifier is used to link the movie to its corresponding ratings and other metadata. Having a unique identifier for each movie is important for building recommendation systems as it allows the system to keep track of which movies have been rated by a user and which movies have not, and to make personalised recommendations based on that information.

title : "title" typically refers to the name of a movie. The movie title is an important piece of information as it helps users identify and search for specific movies they are interested in. The movie title is often included as a feature in recommendation systems and can be used to make recommendations based on user preferences.

genres : "genre" typically refers to a category or type of movie based on its subject matter or style. A movie can belong to one or more genres, such as action, comedy, drama, or horror.

Genres are a pipe-separated list, and are selected from the following:

●Action	●Adventure	●Animation	●Children's
●Comedy	●Crime	●Documentary	●Drama
●Fantasy	●Film-Noir	●Horror	●Musical
●Mystery	●Romance	●Sci-fi	●Sports
●Thriller	●War	●Western	● (no genres listed)



	A	B	C	D
1	userId	movieId	rating	timestamp
2	1	1	4	964982703
3	1	3	4	964981247
4	1	6	4	964982224
5	1	47	5	964983815
6	1	50	5	964982931
7	1	70	3	964982400
8	1	101	5	964980868
9	1	110	4	964982176
10	1	151	5	964984041
11	1	157	5	964984100
12	1	163	5	964983650
13	1	216	5	964981208
14	1	223	3	964980985
15	1	231	5	964981179

Figure 12: Sample of ratings dataset from Kaggle Website

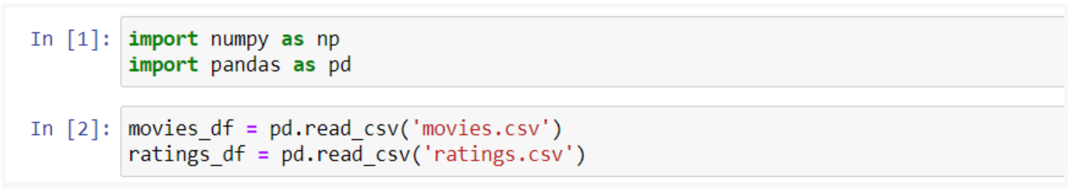
In the above figure we can see the movies dataset which has 4 attributes :

userId : userId typically refers to a unique identifier assigned to each user in a dataset. This identifier is used to differentiate between users and is often used as a key when joining data from different tables.

movieId : in the MovieLens dataset, each movie has a unique "movieid" assigned to it. This identifier is used to link the movie to its corresponding ratings and other metadata.

Rating : "ratings" refer to the numerical values assigned by users to movies in a dataset, indicating their preference or opinion of that movie. Ratings can range from 0 to 5, with 0 indicating the user did not like the movie at all, and 5 indicating that the user loved the movie.

Timestamp : time stamp" typically refers to a value that represents the time and date when a user rated a particular movie.

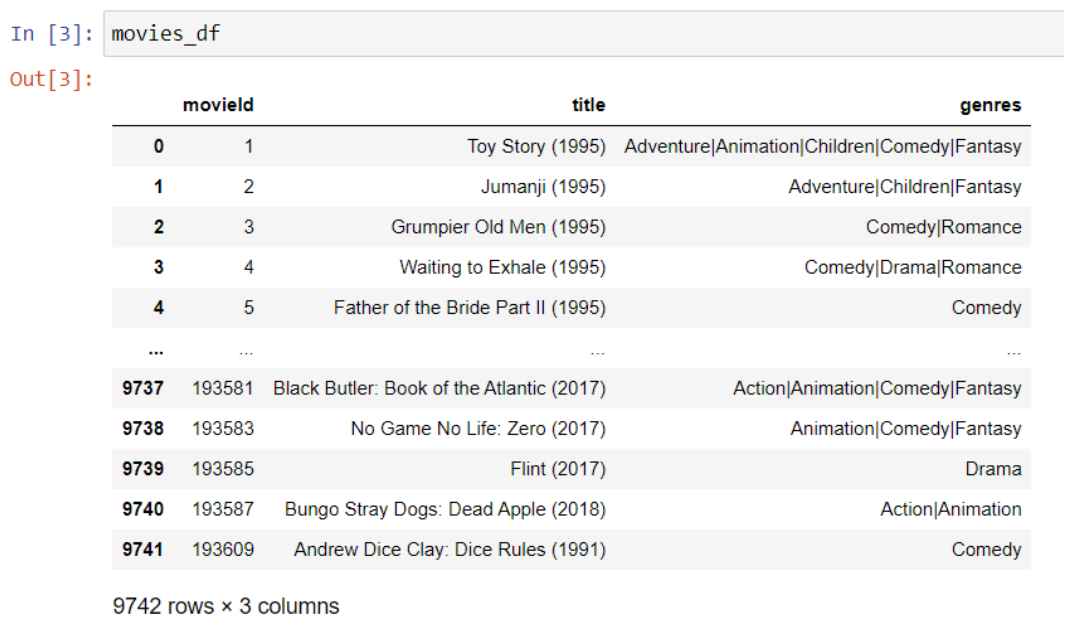


```
In [1]: import numpy as np
        import pandas as pd

In [2]: movies_df = pd.read_csv('movies.csv')
        ratings_df = pd.read_csv('ratings.csv')
```

Figure 13: Reading the dataset from CSV file into Jupyter notebook

After acquiring the data our next step is to read the data from the csv file into a jupyter notebook. In the fig- , we have read data from a csv file using the inbuilt python functions that are part of pandas library.

Displaying movies and ratings data frames:

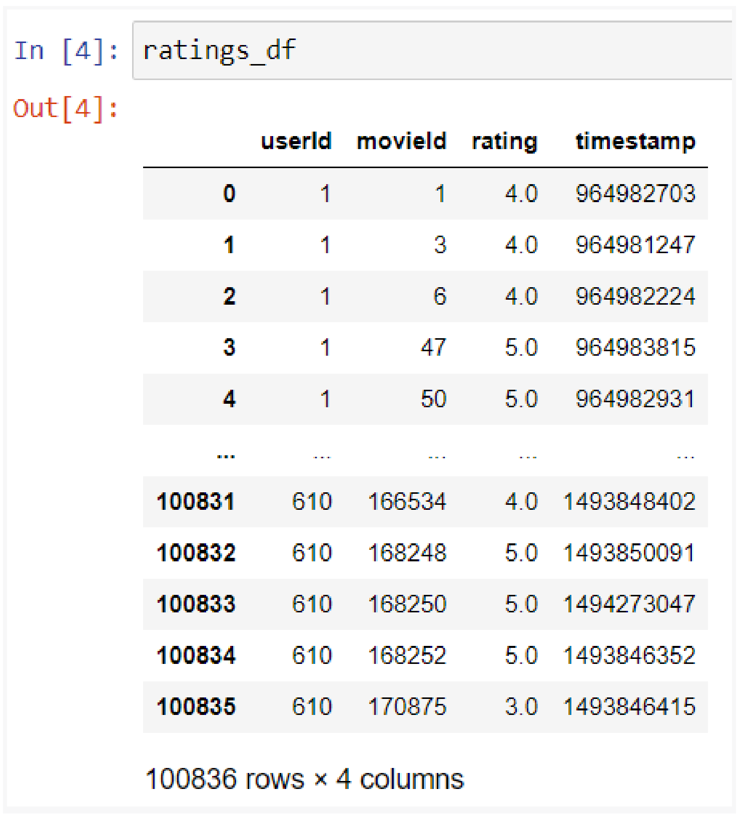


```
In [3]: movies_df
Out[3]:
```

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	Action|Animation|Comedy|Fantasy
9738	193583	No Game No Life: Zero (2017)	Animation|Comedy|Fantasy
9739	193585	Flint (2017)	Drama
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	Action|Animation
9741	193609	Andrew Dice Clay: Dice Rules (1991)	Comedy

9742 rows × 3 columns

Figure 14: Movies_df



```
In [4]: ratings_df
Out[4]:
```

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

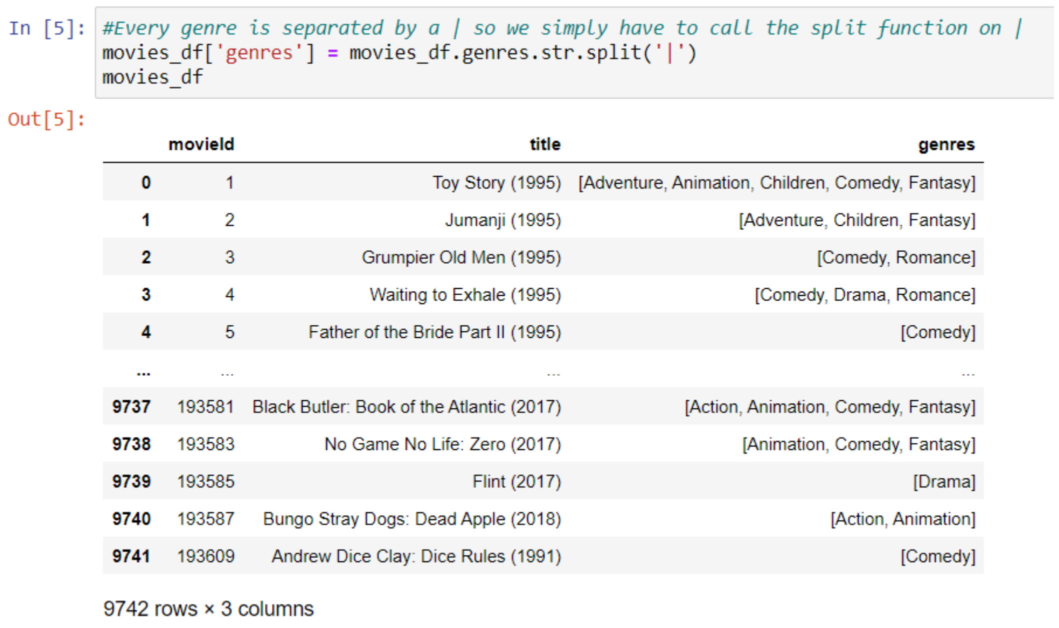
Figure 15 : Ratings_df

4.5.2. Data Pre-processing:

The goal of this step is to study and understand the nature of data that was acquired in the previous step and also to know the quality of data. Data preprocessing is an important step in building movie recommendation systems. It involves cleaning, transforming, and preparing data to ensure that it is in a format that can be used by machine learning algorithms.

In this step, we will check for any null values and remove them as they may affect the efficiency. Identifying duplicates in the dataset and removing them is also done in this step.

The code movies_df['genres'] = movies_df.genres.str.split('|') splits the "genres" column in the "movies_df" dataframe by the "|" separator, and stores the resulting list of genres in the "genres" column of the same dataframe.



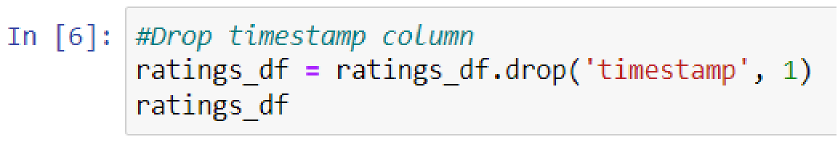
```
In [5]: #Every genre is separated by a | so we simply have to call the split function on |
        movies_df['genres'] = movies_df.genres.str.split('|')
        movies_df
Out[5]:
```

	movieId	title	genres
0	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]
1	2	Jumanji (1995)	[Adventure, Children, Fantasy]
2	3	Grumpier Old Men (1995)	[Comedy, Romance]
3	4	Waiting to Exhale (1995)	[Comedy, Drama, Romance]
4	5	Father of the Bride Part II (1995)	[Comedy]
...	...	...	...
9737	193581	Black Butler: Book of the Atlantic (2017)	[Action, Animation, Comedy, Fantasy]
9738	193583	No Game No Life: Zero (2017)	[Animation, Comedy, Fantasy]
9739	193585	Flint (2017)	[Drama]
9740	193587	Bungo Stray Dogs: Dead Apple (2018)	[Action, Animation]
9741	193609	Andrew Dice Clay: Dice Rules (1991)	[Comedy]

9742 rows × 3 columns

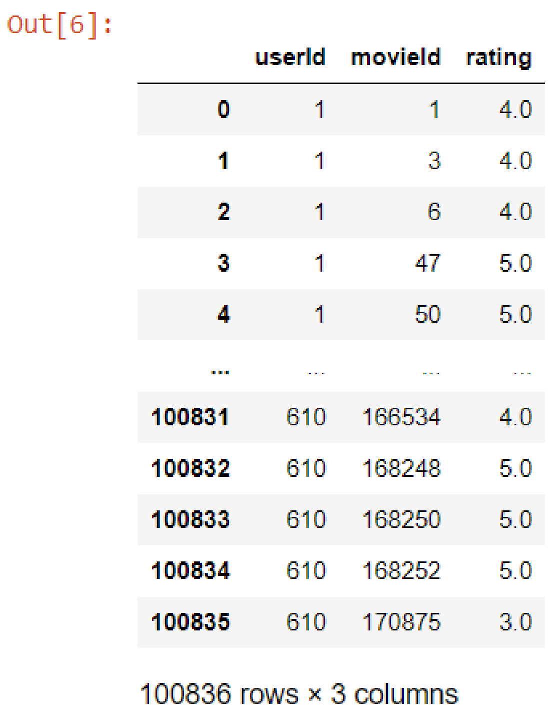
Figure 16 : Removing | in genres section

This is a common data preprocessing step in movie recommendation systems, as it allows for easier analysis and manipulation of the genres data. Instead of having a single string value in the "genres" column that contains all genres separated by "|", the split function creates a list of individual genres, making it easier to count the occurrences of each genre, extract certain genres, or create new features based on genre information.After running the code, the "movies_df" data frame will have a new column named "genres" that contains a list of genres for each movie, rather than a single string value. This can be confirmed by running movies_df.head() to display the first few rows of the dataframe.



```
In [6]: #Drop timestamp column
        ratings_df = ratings_df.drop('timestamp', 1)
        ratings_df
```

Figure 17 : Drops timestamp column



Out[6]:

	userId	movieId	rating
0	1	1	4.0
1	1	3	4.0
2	1	6	4.0
3	1	47	5.0
4	1	50	5.0
...	...	...	...
100831	610	166534	4.0
100832	610	168248	5.0
100833	610	168250	5.0
100834	610	168252	5.0
100835	610	170875	3.0

100836 rows × 3 columns

Figure 18 : After dropping 'timestamp' column

The code ratings_df = ratings_df.drop('timestamp', 1) drops the "timestamp" column from the "ratings_df" dataframe.

After running the code, the "ratings_df" data frame will no longer contain the "timestamp" column, and the resulting data frame will only contain the user ID, movie ID, and rating value for each rating. This can be confirmed by running ratings_df.head() to display the first few rows of the dataframe.

4.5.3. Algorithms:

In this project we have three algorithms, 1.Content Based Filtering, 2.Collaborative based Filtering and 3.Single valued decomposition algorithms are used to build the machine learning recommendation model.

4.5.4. Similarity Measures of the models:

- **Pearson Correlation Similarity**

It uses Pearson Correlation Coefficient to determine the similarity between users. The higher the coefficient the two users are more closely related. Formula to calculate Pearson Correlation Coefficient is given below

$$r = \frac{\sum((u-\bar{u})(v-\bar{v}))}{\sqrt{\sum(u-\bar{u})^2 . \sum(v-\bar{v})^2}}$$

4.5.5. Training and Testing the model:

Once the implementation of the algorithm is completed . We have to train the model to get the result. We have tested it several times and the model recommends different sets of movies to different users.

4.5.6. Improvements in the project:

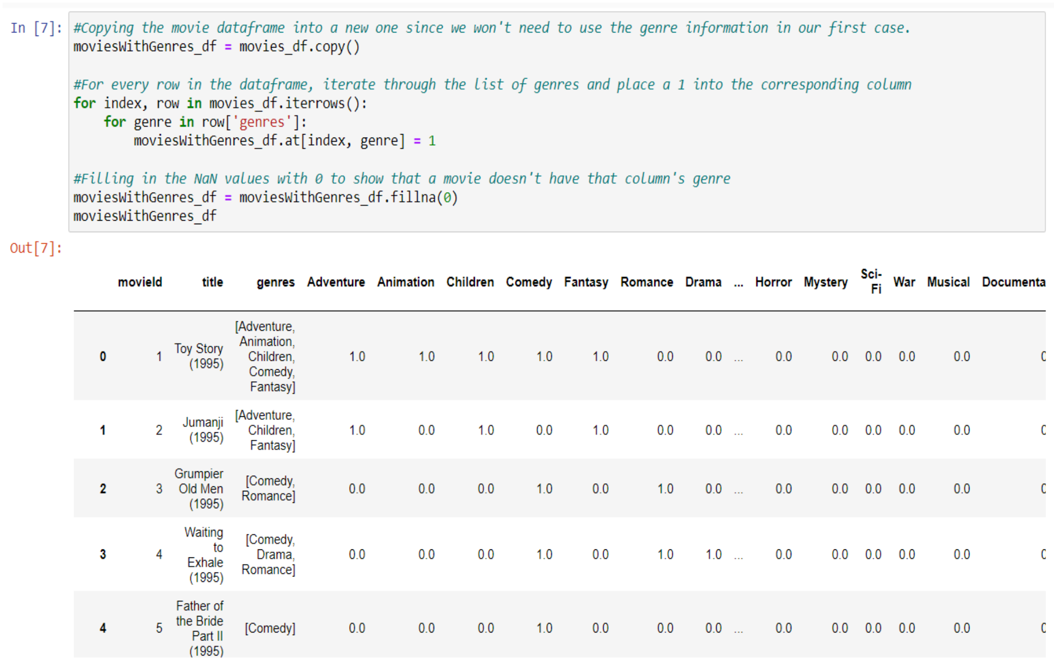
In the later stage we can implement different algorithms and methods for better recommendation.

CHAPTER 5 – IMPLEMENTATION AND TESTING

5.1. IMPLEMENTATION

In this section, we will introduce how to implement the content-based recommender system, user – user collaborative recommender system and model-based recommender system based on the principle mentioned.

5.1.1. CONTENT – BASED RECOMMENDER SYSTEM



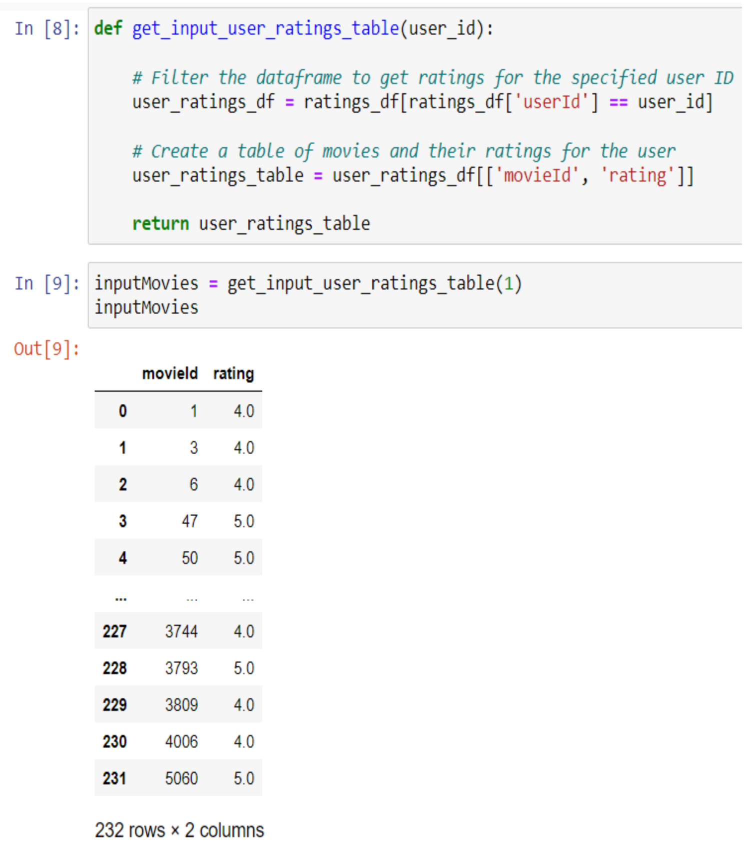
```
In [7]: #Copying the movie dataframe into a new one since we won't need to use the genre information in our first case.
        moviesWithGenres_df = movies_df.copy()

        #For every row in the dataframe, iterate through the list of genres and place a 1 into the corresponding column
        for index, row in movies_df.iterrows():
            for genre in row['genres']:
                moviesWithGenres_df.at[index, genre] = 1

        #Filling in the NaN values with 0 to show that a movie doesn't have that column's genre
        moviesWithGenres_df = moviesWithGenres_df.fillna(0)
        moviesWithGenres_df
Out[7]:
```

	movieId	title	genres	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	...	Horror	Mystery	Sci-Fi	War	Musical	Documenta
0	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	1.0	1.0	1.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0
1	2	Jumanji (1995)	[Adventure, Children, Fantasy]	1.0	0.0	1.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0
2	3	Grumpier Old Men (1995)	[Comedy, Romance]	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0
3	4	Waiting to Exhale (1995)	[Comedy, Drama, Romance]	0.0	0.0	0.0	1.0	0.0	1.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0
4	5	Father of the Bride Part II (1995)	[Comedy]	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0

Figure 19 : Movies with Genres_df

The code moviesWithGenres_df = movies_df.copy() creates a copy of the "movies_df" dataframe and stores it in a new dataframe called "moviesWithGenres_df". This is done because we will be adding additional columns to this new dataframe based on the genre information. The following for loop iterates through every row in the "movies_df" dataframe and for each row, iterates through the list of genres and sets a value of 1 in the corresponding column of the "moviesWithGenres_df" dataframe. This effectively creates new columns for each genre and populates them with 1 or 0 depending on whether the movie belongs to that genre or not.

Finally, the code moviesWithGenres_df = moviesWithGenres_df.fillna(0) fills in any missing values (NaN) in the new genre columns with 0, indicating that the corresponding movie does not belong to that genre. The resulting "moviesWithGenres_df" dataframe now has additional columns for each genre, allowing for easier analysis and manipulation of the genre information.



```
In [8]: def get_input_user_ratings_table(user_id):

            # Filter the dataframe to get ratings for the specified user ID
            user_ratings_df = ratings_df[ratings_df['userId'] == user_id]

            # Create a table of movies and their ratings for the user
            user_ratings_table = user_ratings_df[['movieId', 'rating']]

            return user_ratings_table
```

```
In [9]: inputMovies = get_input_user_ratings_table(1)
        inputMovies
```

Out[9]:

	movieId	rating
0	1	4.0
1	3	4.0
2	6	4.0
3	47	5.0
4	50	5.0
...	...	...
227	3744	4.0
228	3793	5.0
229	3809	4.0
230	4006	4.0
231	5060	5.0

232 rows × 2 columns

Figure 20 : Movies which a particular user has rated

The function get_input_user_ratings_table(user_id) takes in a user_id as input and returns a table of movies and their corresponding ratings given by the specified user.

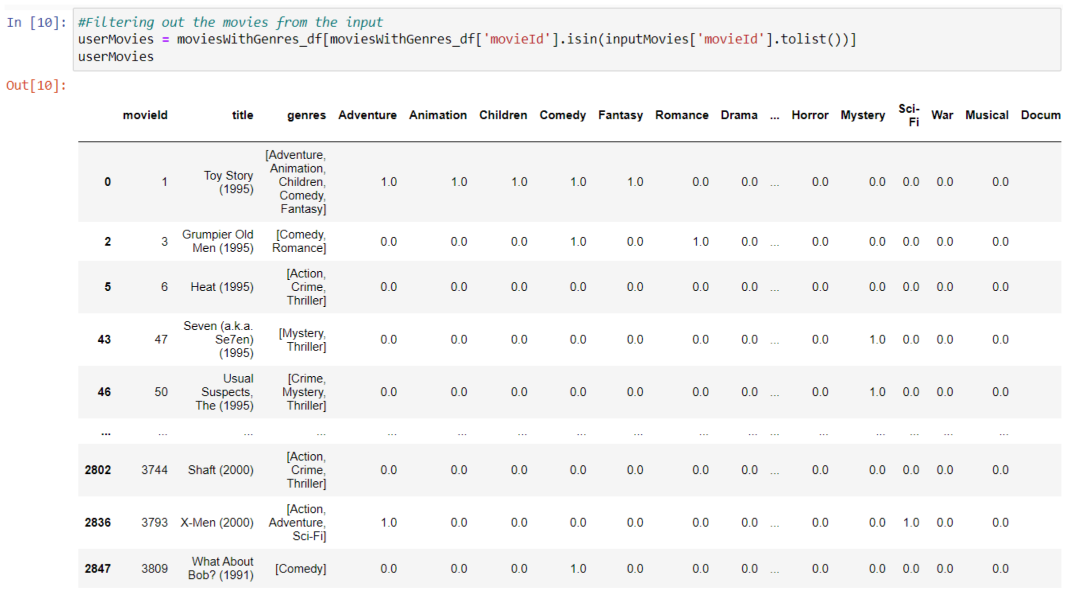
The function first filters the "ratings_df" data frame to get only the rows where the "userId" column matches the input user_id. This is done using the syntax ratings_df[ratings_df['userId'] == user_id].

Next, the function creates a new dataframe called "user_ratings_table" by selecting only the "movieId" and "rating" columns from the filtered dataframe.

Finally, the function returns the "user_ratings_table" data frame as output. The resulting table contains only the movies and ratings that the specified user has given, which will be used as input for the recommendation system.

inputMovies = get_input_user_ratings_table(1) will create a new dataframe called "inputMovies" that contains the movie IDs and corresponding ratings given by the user with ID 1.

This code essentially calls the function get_input_user_ratings_table() with the input argument of 1 to get the ratings given by user with ID 1. The resulting data frame will contain only the movie IDs and ratings that this user has provided, which will be used as input for the recommendation system.



```
In [10]: #Filtering out the movies from the input
         userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies['movieId'].tolist())]
         userMovies
Out[10]:
```

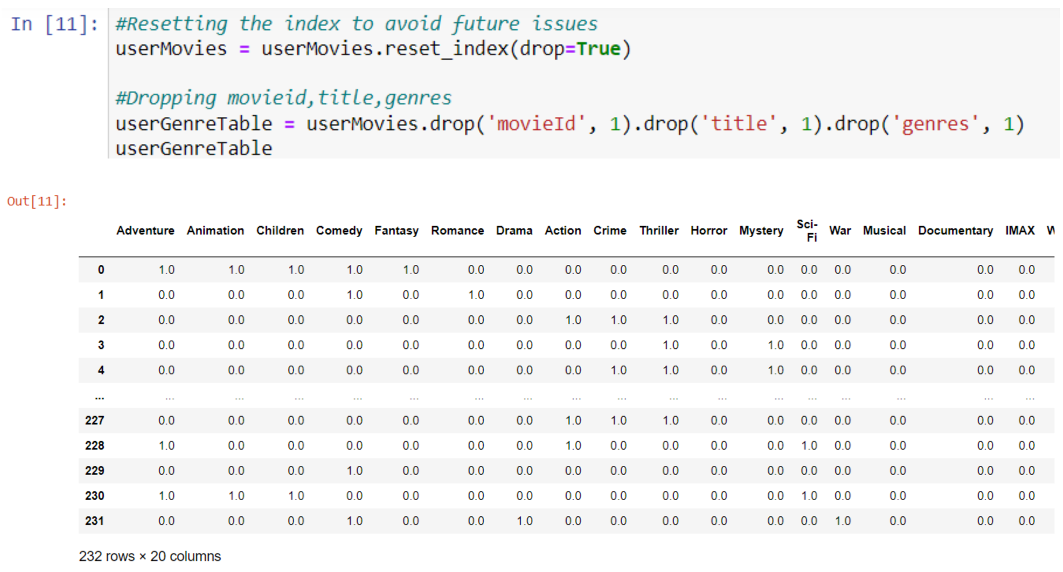
	movieId	title	genres	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	...	Horror	Mystery	Sci-Fi	War	Musical	Docum
0	1	Toy Story (1995)	[Adventure, Animation, Children, Comedy, Fantasy]	1.0	1.0	1.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
2	3	Grumpier Old Men (1995)	[Comedy, Romance]	0.0	0.0	0.0	1.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
5	6	Heat (1995)	[Action, Crime, Thriller]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
43	47	Seven (a.k.a. Se7en) (1995)	[Mystery, Thriller]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	
46	50	Usual Suspects, The (1995)	[Crime, Mystery, Thriller]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
2802	3744	Shaft (2000)	[Action, Crime, Thriller]	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	
2836	3793	X-Men (2000)	[Action, Adventure, Sci-Fi]	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	
2847	3809	What About Bob? (1991)	[Comedy]	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	

Figure 21 : User Movies

The code userMovies = moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(inputMovies ['movieId'].tolist())] filters out the movies from the "moviesWithGenres_df" dataframe that have been rated by the input user.

The isin() function is used to filter the "moviesWithGenres_df" dataframe based on whether the "movieId" column is present in the "movieId" column of the "inputMovies" dataframe. This filtering is done using the tolist() function, which converts the "movieId" column of the "inputMovies" dataframe to a Python list.

The resulting "userMovies" dataframe contains only the rows from "moviesWithGenres_df" that correspond to movies that have been rated by the input user. This dataframe will be used as input for the recommendation system to find similar movies.



```
In [11]: #Resetting the index to avoid future issues
         userMovies = userMovies.reset_index(drop=True)

         #Dropping movieid,title,genres
         userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1)
         userGenreTable
```

Out[11]:

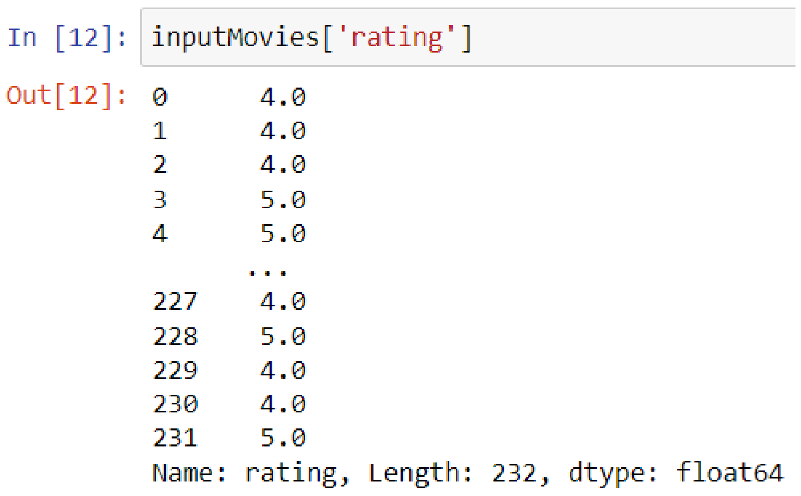
	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	War	Musical	Documentary	IMAX	W
0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
227	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
228	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
229	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
230	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	
231	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	

232 rows × 20 columns

Figure 22 : User Genre Table

The code userMovies = userMovies.reset_index(drop=True) resets the index of the "userMovies" dataframe to avoid any potential issues in future computations. The parameter drop=True is used to drop the old index column.

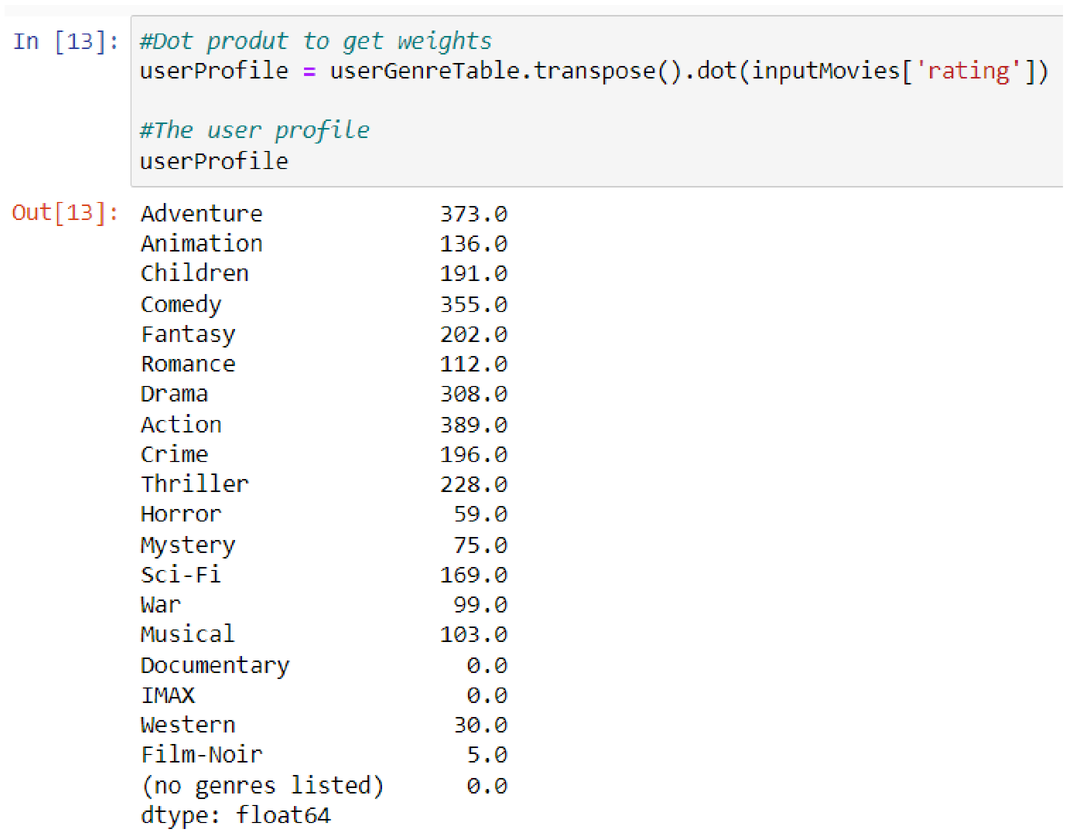
The subsequent code userGenreTable = userMovies.drop('movieId', 1).drop('title', 1).drop('genres', 1) drops the columns "movieId", "title", and "genres" from the "userMovies" dataframe, since these columns are not needed for the recommendation algorithm. The resulting dataframe "userGenreTable" contains only the one-hot encoded genre information for the movies rated by the input user.



```
In [12]: inputMovies['rating']

Out[12]: 0      4.0
         1      4.0
         2      4.0
         3      5.0
         4      5.0
               ...
         227    4.0
         228    5.0
         229    4.0
         230    4.0
         231    5.0
         Name: rating, Length: 232, dtype: float64
```

Figure 23 : Ratings given by the input user

The code inputMovies['rating'] returns a Pandas Series containing the ratings given by the input user for the movies in the "inputMovies" dataframe. The output will be a one-dimensional array of ratings, where the index of each element corresponds to the index of the corresponding movie in the "inputMovies" dataframe.



```
In [13]: #Dot produt to get weights
         userProfile = userGenreTable.transpose().dot(inputMovies['rating'])

         #The user profile
         userProfile
```

```
Out[13]: Adventure              373.0
         Animation              136.0
         Children               191.0
         Comedy                 355.0
         Fantasy                202.0
         Romance                112.0
         Drama                  308.0
         Action                 389.0
         Crime                  196.0
         Thriller               228.0
         Horror                  59.0
         Mystery                 75.0
         Sci-Fi                 169.0
         War                     99.0
         Musical                103.0
         Documentary              0.0
         IMAX                     0.0
         Western                 30.0
         Film-Noir                5.0
         (no genres listed)       0.0
         dtype: float64
```

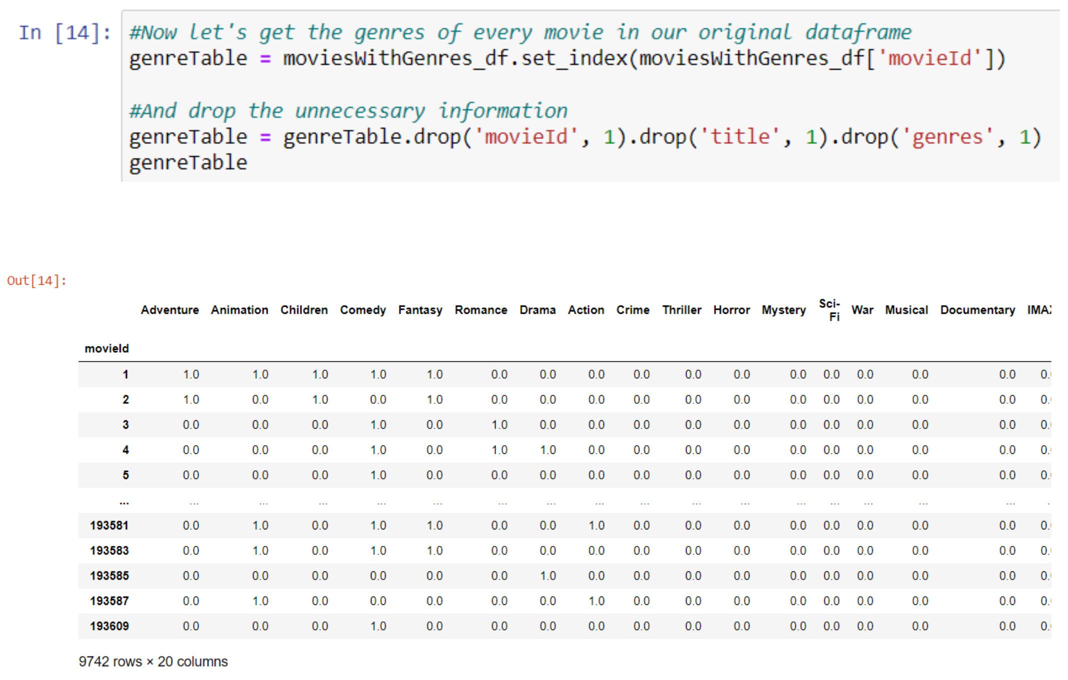
Figure 24 : User Profile

The code

$userProfile = userGenreTable.transpose().dot(inputMovies['rating'])$

computes the dot product between the transpose of the "userGenreTable" dataframe and the ratings given by the input user, which results in a weighted sum of the genres based on the user's ratings.

The resulting "userProfile" is a Pandas Series containing the weighted sum of each genre, where the index of each element corresponds to the genre name. The weights represent the strength of the user's preference for each genre based on their ratings of movies belonging to that genre.



```
In [14]: #Now let's get the genres of every movie in our original dataframe
         genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])

         #And drop the unnecessary information
         genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1)
         genreTable
```

Out[14]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	War	Musical	Documentary	IMA
movieId																	
1	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
2	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
3	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
193581	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
193583	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
193585	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
193587	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
193609	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

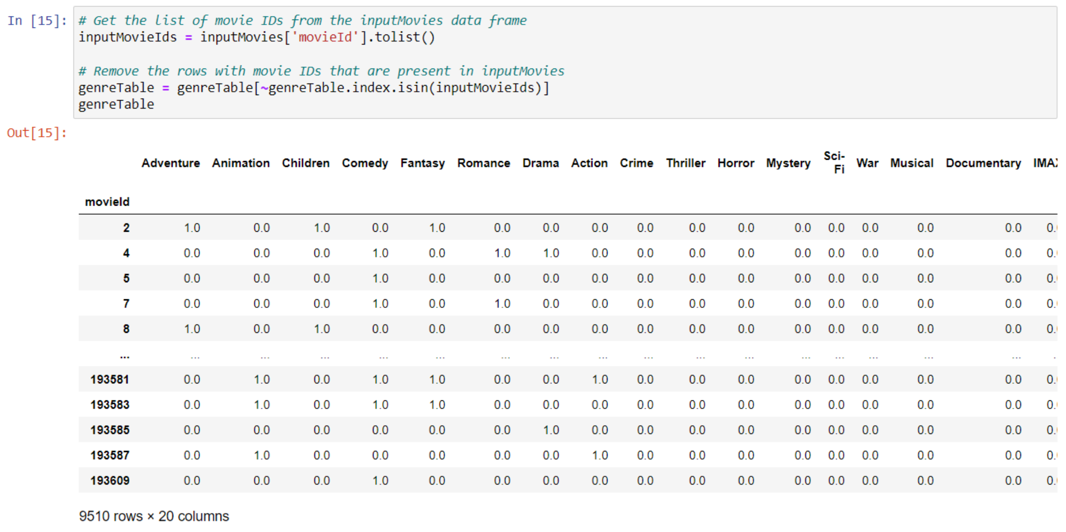
9742 rows × 20 columns

Figure 25 : Genre Table

The code genreTable = moviesWithGenres_df.set_index(moviesWithGenres_df['movieId']) sets the "movieId" column as the index of the "moviesWithGenres_df" dataframe.

The next line drops the "movieId", "title", and "genres" columns from the "genreTable" dataframe, leaving only the binary indicator variables for each genre for each movie as columns.

The resulting "genreTable" data frame has the same number of rows as the original "moviesWithGenres_df" dataframe, with each row representing a movie and each column representing a genre. A value of 1 in a cell indicates that the movie belongs to that genre, while a value of 0 indicates that it does not.



```
In [15]: # Get the list of movie IDs from the inputMovies data frame
         inputMovieIds = inputMovies['movieId'].tolist()

         # Remove the rows with movie IDs that are present in inputMovies
         genreTable = genreTable[~genreTable.index.isin(inputMovieIds)]
         genreTable
```

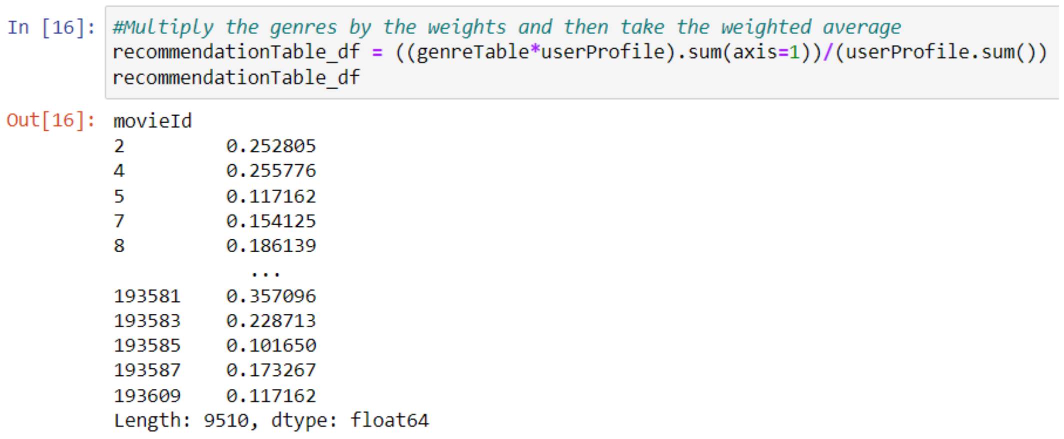
Out[15]:

	Adventure	Animation	Children	Comedy	Fantasy	Romance	Drama	Action	Crime	Thriller	Horror	Mystery	Sci-Fi	War	Musical	Documentary	IMA:
movieId																	
2	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
4	0.0	0.0	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
5	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
7	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
8	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	.
193581	0.0	1.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
193583	0.0	1.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
193585	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
193587	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.
193609	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.

9510 rows × 20 columns

Figure 26 : Unrated movies in Genre Table

We have removed the movies which the user has already rated from the genre table.



```
In [16]: #Multiply the genres by the weights and then take the weighted average
         recommendationTable_df = ((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
         recommendationTable_df

Out[16]: movieId
         2         0.252805
         4         0.255776
         5         0.117162
         7         0.154125
         8         0.186139
                     ...
         193581    0.357096
         193583    0.228713
         193585    0.101650
         193587    0.173267
         193609    0.117162
         Length: 9510, dtype: float64
```

Figure 27 : RecommendationTable_df

The code is generating a recommendation table for movies that a user might like based on their input ratings.

First, the function get_input_user_ratings_table() is defined, which filters the ratings_df data frame to get ratings for a specified user ID and returns a table of movies and their ratings for the user.

Next, the input ratings are filtered to get only the movies that the user has rated and stored in the inputMovies dataframe.

The moviesWithGenres_df data frame is then used to filter the genres of the movies in the inputMovies dataframe, and the userGenreTable is created by dropping the unnecessary columns.

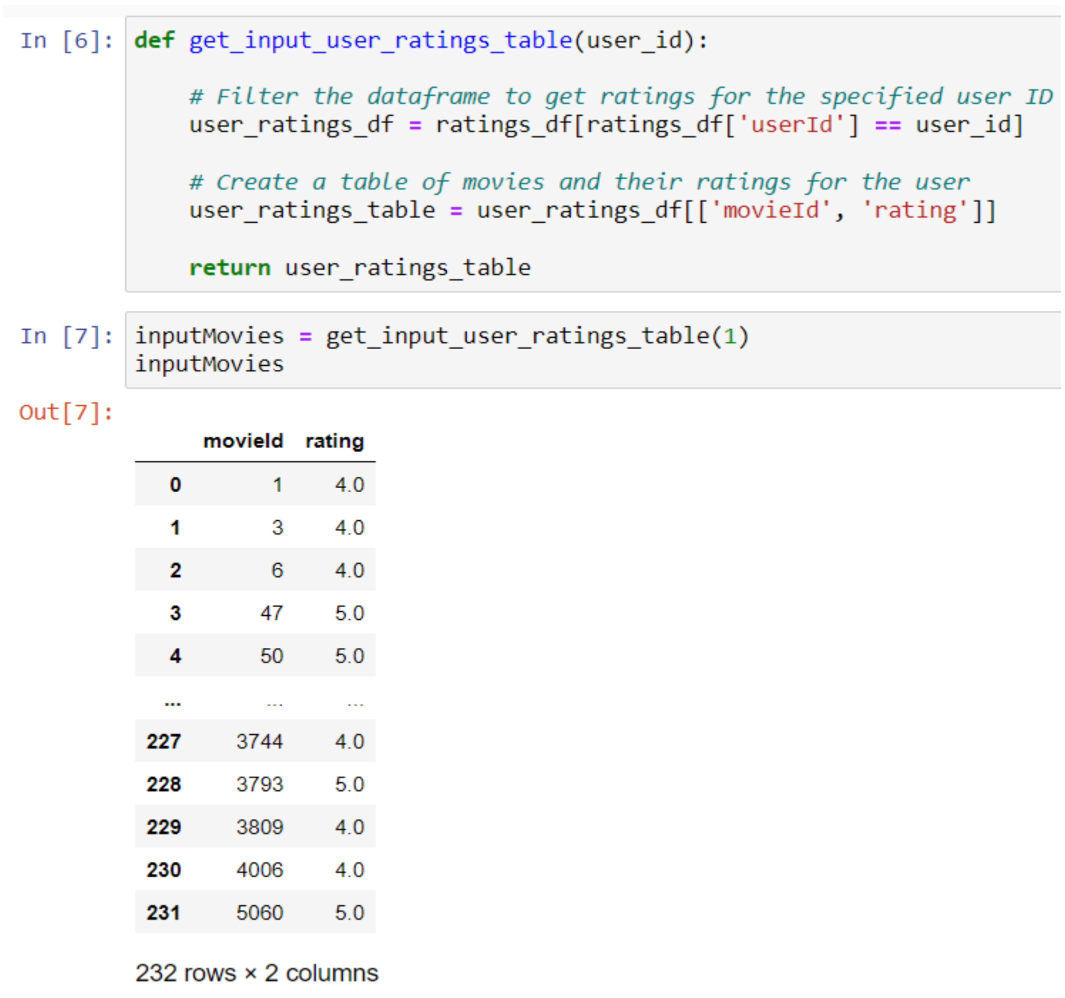
Using the input ratings, the userProfile is created by taking the dot product of the userGenreTable with the input ratings.

The genreTable is then used to get the genres of every movie in the original dataframe, and rows with movie IDs that are present in inputMovies are removed.

The recommendationTable_df is created by multiplying the genres by the weights and then taking the weighted average. This table represents the expected rating for each movie by the user, based on the user's input ratings and the similarity between the user's ratings and the ratings of other users for the same movies. The movies with the highest expected rating in this table are recommended to the user.

5.1.2. USER – USER COLLABORATIVE RECOMMENDER SYSTEM

This function is useful for building a movie recommendation system because it allows you to easily get the ratings for a specific user, which can be used as input to the recommendation algorithm.
inputMovies can now be used as input to a movie recommendation algorithm, which can use the ratings data to make personalised movie recommendations for the user with ID 1.

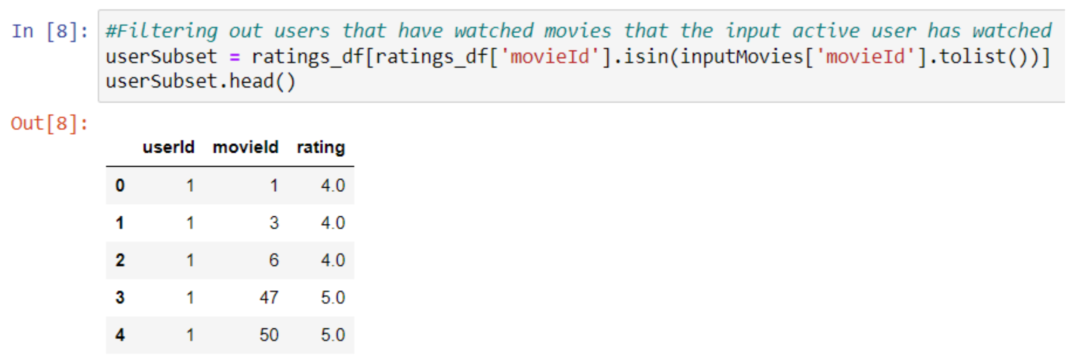


```
In [6]: def get_input_user_ratings_table(user_id):

            # Filter the dataframe to get ratings for the specified user ID
            user_ratings_df = ratings_df[ratings_df['userId'] == user_id]

            # Create a table of movies and their ratings for the user
            user_ratings_table = user_ratings_df[['movieId', 'rating']]

            return user_ratings_table
```

```
In [7]: inputMovies = get_input_user_ratings_table(1)
        inputMovies
```

```
Out[7]:
          movieId  rating
      0         1     4.0
      1         3     4.0
      2         6     4.0
      3        47     5.0
      4        50     5.0
    ...       ...     ...
    227      3744     4.0
    228      3793     5.0
    229      3809     4.0
    230      4006     4.0
    231      5060     5.0

232 rows × 2 columns
```

Figure 28 : Movies which particular user has rated

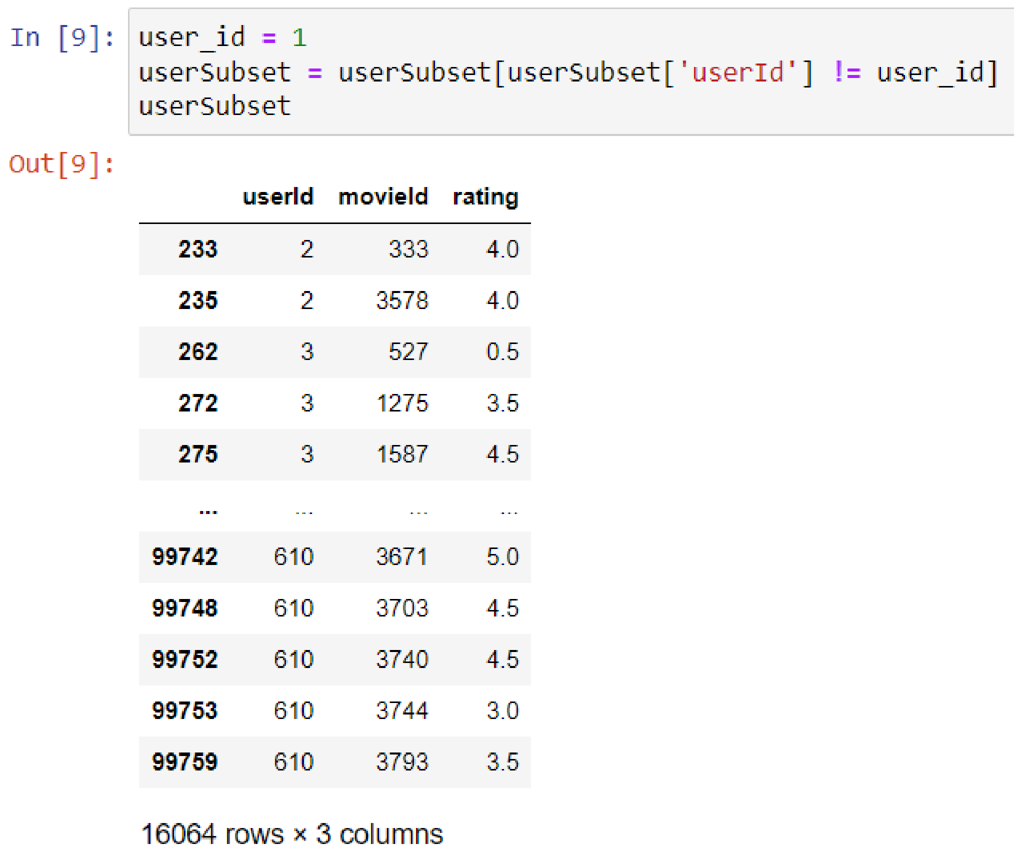


```
In [8]: #Filtering out users that have watched movies that the input active user has watched
        userSubset = ratings_df[ratings_df['movieId'].isin(inputMovies['movieId'].tolist())]
        userSubset.head()
```

```
Out[8]:
       userId  movieId  rating
    0       1        1     4.0
    1       1        3     4.0
    2       1        6     4.0
    3       1       47     5.0
    4       1       50     5.0
```

Figure 29 : Users that have watched movies that input user have watched

This filtering step is useful for building a movie recommendation system because it allows the algorithm to focus on users who have similar tastes in movies to the input active user. By excluding users who have not watched any of the same movies as the input active user, the algorithm can avoid making recommendations that are based on very little information about the user's preferences.



```
In [9]: user_id = 1
        userSubset = userSubset[userSubset['userId'] != user_id]
        userSubset
```

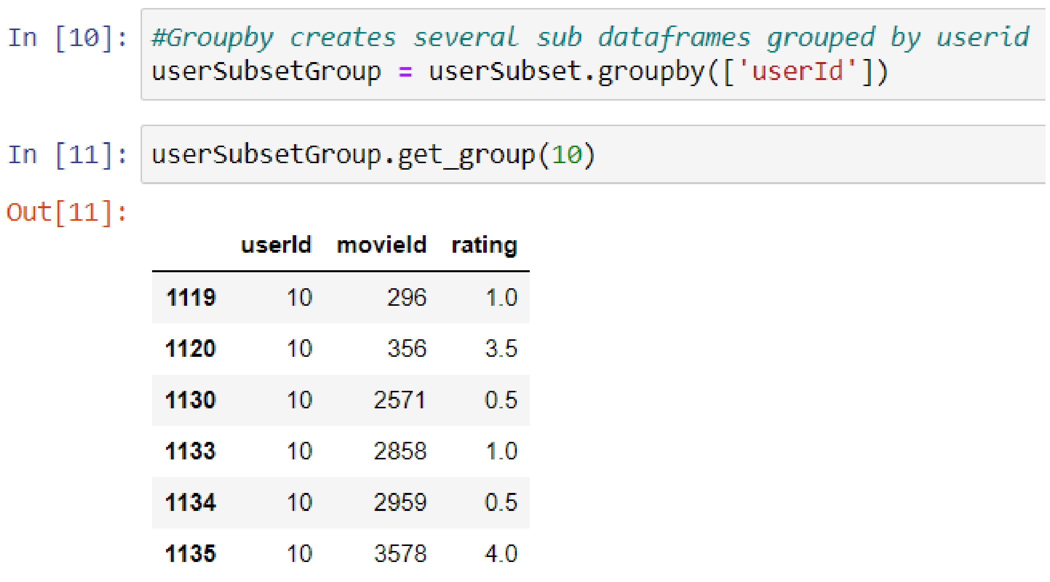
Out[9]:

	userId	movieId	rating
233	2	333	4.0
235	2	3578	4.0
262	3	527	0.5
272	3	1275	3.5
275	3	1587	4.5
...	...	...	...
99742	610	3671	5.0
99748	610	3703	4.5
99752	610	3740	4.5
99753	610	3744	3.0
99759	610	3793	3.5

16064 rows × 3 columns

Figure 30 : User Subset

We want to find other users who have similar movie preferences to the input active user. By removing the input active user from the userSubset data frame, we can ensure that the recommendation algorithm will not simply recommend movies that the input active user has already seen and rated.

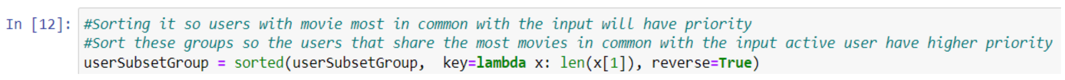


```
In [10]: #Groupby creates several sub dataframes grouped by userid
         userSubsetGroup = userSubset.groupby(['userId'])

In [11]: userSubsetGroup.get_group(10)

Out[11]:
```

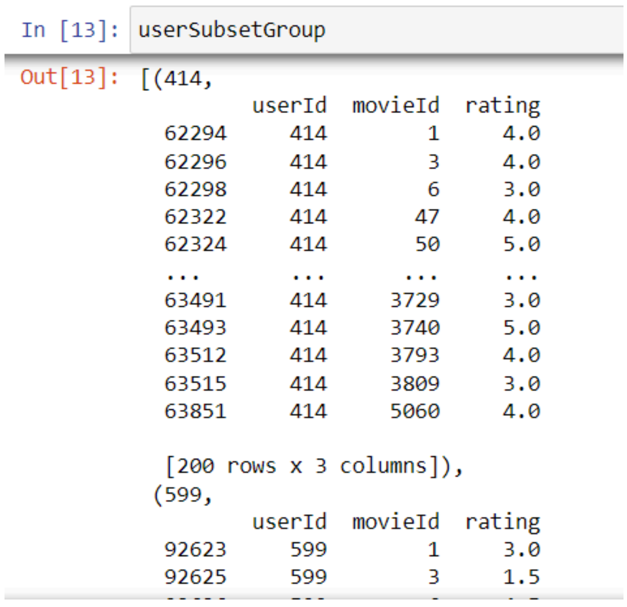
	userId	movieId	rating
1119	10	296	1.0
1120	10	356	3.5
1130	10	2571	0.5
1133	10	2858	1.0
1134	10	2959	0.5
1135	10	3578	4.0

Figure 31 : User Subset Group

It allows us to analyse the movie preferences of each user in the userSubset data frame separately. By grouping the data by userId, we can apply recommendation algorithms to each sub-dataframe to find personalised recommendations for each user.



```
In [12]: #Sorting it so users with movie most in common with the input will have priority
         #Sort these groups so the users that share the most movies in common with the input active user have higher priority
         userSubsetGroup = sorted(userSubsetGroup,  key=lambda x: len(x[1]), reverse=True)
```

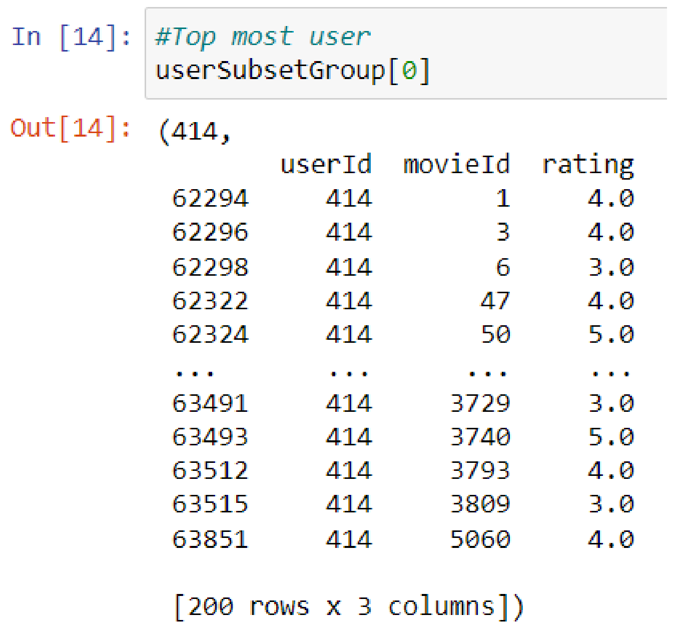


```
In [13]: userSubsetGroup

Out[13]: [(414,
                 userId  movieId  rating
          62294     414        1     4.0
          62296     414        3     4.0
          62298     414        6     3.0
          62322     414       47     4.0
          62324     414       50     5.0
          ...       ...      ...     ...
          63491     414     3729     3.0
          63493     414     3740     5.0
          63512     414     3793     4.0
          63515     414     3809     3.0
          63851     414     5060     4.0

          [200 rows x 3 columns]),
         (599,
                 userId  movieId  rating
          92623     599        1     3.0
          92625     599        3     1.5
```

Figure 32 : User Subset Group after sorting

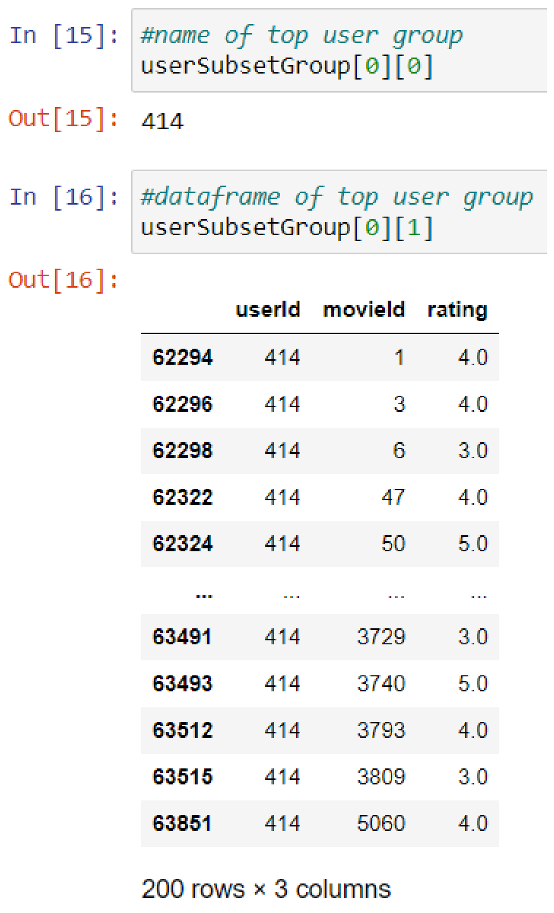
This code sorts the sub-dataframes in userSubsetGroup by the number of movies each user has watched that are also watched by the input active user, in descending order.

After running the previous code to sort userSubsetGroup, the userSubsetGroup variable is now a list of tuples, where each tuple contains two values: the userId, and the corresponding sub-dataframe containing the ratings data for that user. The tuples are sorted in descending order based on the length of the sub-dataframes, so the first tuple in the list corresponds to the user who has watched the most movies in common with the input active user, and so on.



```
In [14]: #Top most user
         userSubsetGroup[0]

Out[14]: (414,
                 userId  movieId  rating
          62294     414        1     4.0
          62296     414        3     4.0
          62298     414        6     3.0
          62322     414       47     4.0
          62324     414       50     5.0
          ...       ...      ...     ...
          63491     414     3729     3.0
          63493     414     3740     5.0
          63512     414     3793     4.0
          63515     414     3809     3.0
          63851     414     5060     4.0

          [200 rows x 3 columns])
```

Figure 33 : Top-most user in User Subset Group

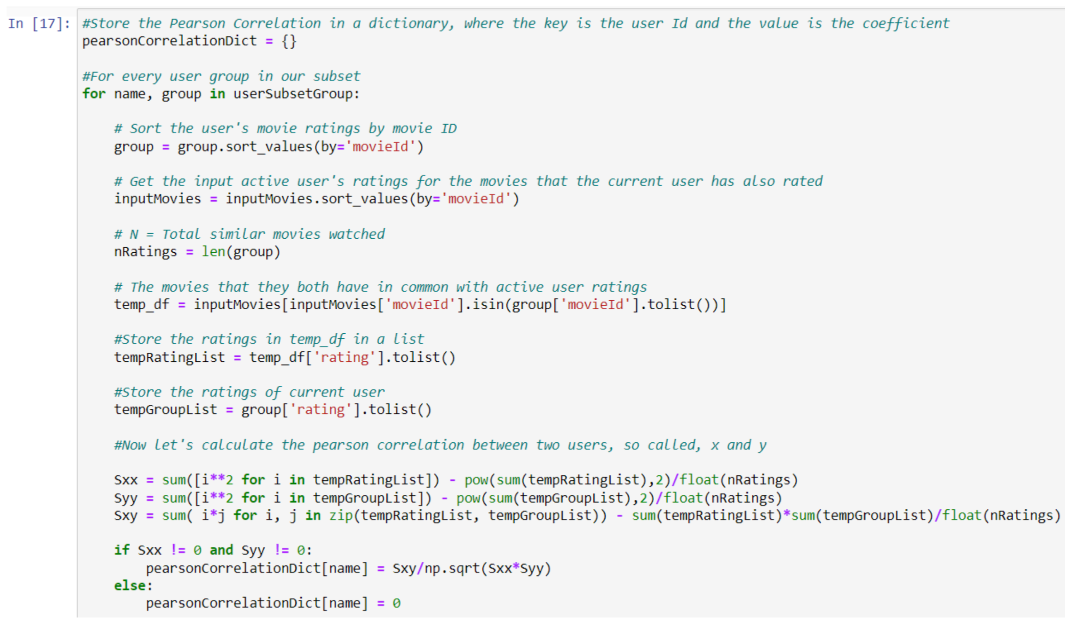


```
In [15]: #name of top user group
         userSubsetGroup[0][0]

Out[15]: 414

In [16]: #dataframe of top user group
         userSubsetGroup[0][1]

Out[16]:
```

	userId	movieId	rating
62294	414	1	4.0
62296	414	3	4.0
62298	414	6	3.0
62322	414	47	4.0
62324	414	50	5.0
...	...	...	...
63491	414	3729	3.0
63493	414	3740	5.0
63512	414	3793	4.0
63515	414	3809	3.0
63851	414	5060	4.0

200 rows × 3 columns

Figure 34 : Name and Dataframe of the Top-most user in User Subset Group

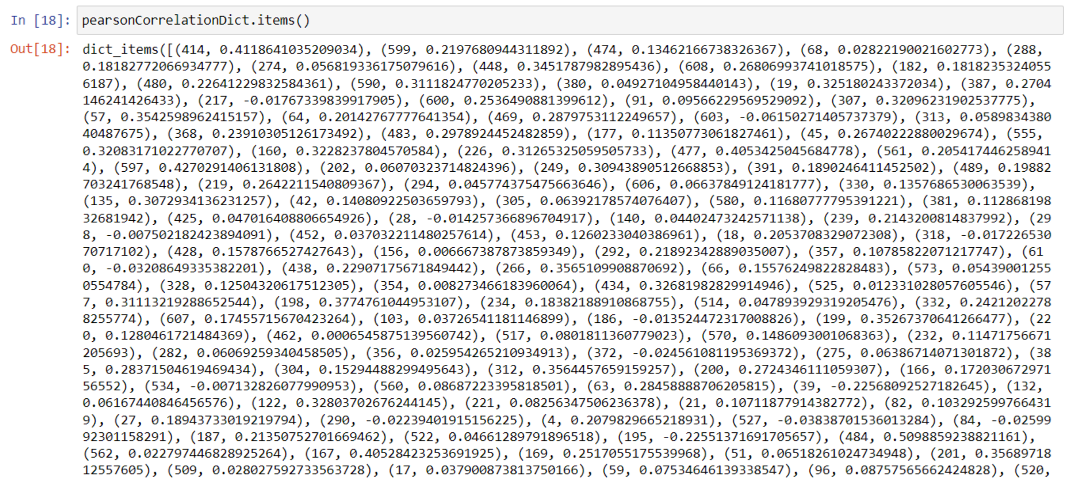


```
In [17]: #Store the Pearson Correlation in a dictionary, where the key is the user Id and the value is the coefficient
         pearsonCorrelationDict = {}

         #For every user group in our subset
         for name, group in userSubsetGroup:

             # Sort the user's movie ratings by movie ID
             group = group.sort_values(by='movieId')

             # Get the input active user's ratings for the movies that the current user has also rated
             inputMovies = inputMovies.sort_values(by='movieId')

             # N = Total similar movies watched
             nRatings = len(group)

             # The movies that they both have in common with active user ratings
             temp_df = inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist())]

             #Store the ratings in temp_df in a list
             tempRatingList = temp_df['rating'].tolist()

             #Store the ratings of current user
             tempGroupList = group['rating'].tolist()

             #Now let's calculate the pearson correlation between two users, so called, x and y

             Sxx = sum([i**2 for i in tempRatingList]) - pow(sum(tempRatingList),2)/float(nRatings)
             Syy = sum([i**2 for i in tempGroupList]) - pow(sum(tempGroupList),2)/float(nRatings)
             Sxy = sum( i*j for i, j in zip(tempRatingList, tempGroupList)) - sum(tempRatingList)*sum(tempGroupList)/float(nRatings)

             if Sxx != 0 and Syy != 0:
                 pearsonCorrelationDict[name] = Sxy/np.sqrt(Sxx*Syy)
             else:
                 pearsonCorrelationDict[name] = 0
```

Figure 35 : Pearson Correlation Dictionary

This code computes the Pearson correlation coefficient between the input active user and each user in the userSubsetGroup, and stores the results in a dictionary called pearsonCorrelationDict. The Pearson correlation is a measure of the linear correlation between two variables, and it ranges from -1 to +1, with 1 indicating a perfect positive correlation, 0 indicating no correlation, and -1 indicating a perfect negative correlation.

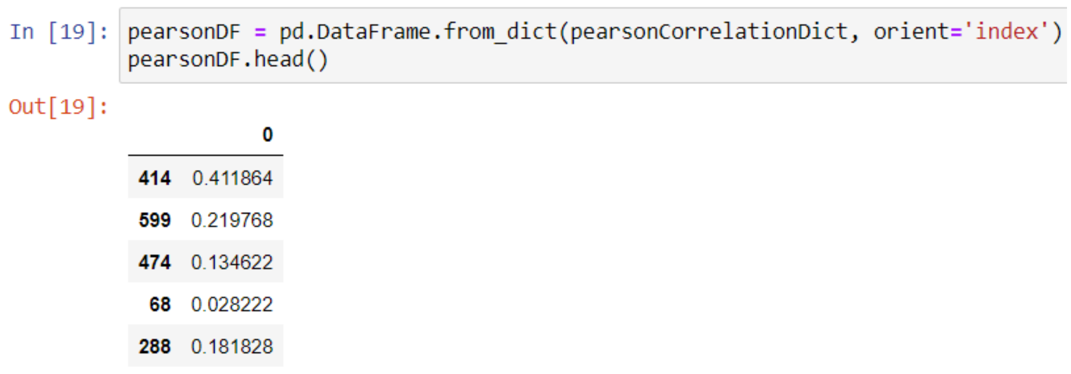
The code first loops through each user group in the userSubsetGroup list. It then sorts the user's movie ratings by movie ID, and retrieves the input active user's ratings for the movies that the current user has also rated. It then calculates the Pearson correlation coefficient between the two users based on their ratings for these common movies.



```
In [18]: pearsonCorrelationDict.items()

Out[18]: dict_items([(414, 0.4118641035209034), (599, 0.2197680944311892), (474, 0.13462166738326367), (68, 0.02822190021602773), (288,
         0.18182772066934777), (274, 0.056819336175079616), (448, 0.3451787982895436), (608, 0.26806993741018575), (182, 0.1818235324055
         6187), (480, 0.22641229832584361), (590, 0.3111824770205233), (380, 0.04927104958440143), (19, 0.325180243372034), (387, 0.2704
         146241426433), (217, -0.01767339839917905), (600, 0.2536490881399612), (91, 0.09566229569529092), (307, 0.32096231902537775),
         (57, 0.354259896241515 7), (64, 0.20142767777641354), (469, 0.2879753112249657), (603, -0.06150271405737379), (313, 0.0589834380
         40487675), (368, 0.23910305126173492), (483, 0.2978924452482859), (177, 0.11350773061827461), (45, 0.26740222880029674), (555,
         0.32083171022770707), (160, 0.3228237804570584), (226, 0.31265325059505733), (477, 0.4053425045684778), (561, 0.205417446258941
         4), (597, 0.4270291406131808), (202, 0.06070323714824396), (249, 0.30943890512668853), (391, 0.1890246411452502), (489, 0.19882
         703241768548), (219, 0.2642211540809367), (294, 0.045774375475663646), (606, 0.06637849124181777), (330, 0.1357686530063539),
         (135, 0.307293413623125 7), (42, 0.14080922503659793), (305, 0.06392178574076407), (580, 0.11680777795391221), (381, 0.112868198
         32681942), (425, 0.047016408806654926), (28, -0.014257366896704917), (140, 0.04402473242571138), (239, 0.2143200814837992), (29
         8, -0.00750218242389409 1), (452, 0.037032211480257614), (453, 0.1260233040386961), (18, 0.2053708329072308), (318, -0.017226530
         70717102), (428, 0.1578766527427643), (156, 0.006667387873859349), (292, 0.21892342889035007), (357, 0.10785822071217747), (61
         0, -0.03208649335382201), (438, 0.22907175671849442), (266, 0.3565109908870692), (66, 0.15576249822828483), (573, 0.05439001255
         0554784), (328, 0.12504320617512305), (354, 0.008273466183960064), (434, 0.32681982829914946), (525, 0.012331028057605546), (57
         7, 0.31113219288652544), (198, 0.3774761044953107), (234, 0.18382188910868755), (514, 0.047893929319205476), (332, 0.2421202278
         8255774), (607, 0.17455715670423264), (103, 0.03726541181146899), (186, -0.013524472317008826), (199, 0.35267370641266477), (22
         0, 0.1280461721484369), (462, 0.0006545875139560742), (517, 0.0801811360779023), (570, 0.1486093001068363), (232, 0.11471756671
         205693), (282, 0.06069259340458505), (356, 0.025954265210934913), (372, -0.024561081195369372), (275, 0.06386714071301872), (38
         5, 0.28371504619469434), (304, 0.15294488299495643), (312, 0.3564457659159257), (200, 0.2724346111059307), (166, 0.172030672971
         56552), (534, -0.007132826077990953), (560, 0.08687223395818501), (63, 0.28458888706205815), (39, -0.2256809252718 2645), (132,
         0.06167440846456576), (122, 0.3280370267624414 5), (221, 0.08256347506236378), (21, 0.10711877914382772), (82, 0.103292599766431
         9), (27, 0.18943733019219794), (290, -0.02239401915156225), (4, 0.2079829665218931), (527, -0.03838701536013284), (84, -0.02599
         92301158291), (187, 0.21350752701669462), (522, 0.04661289791896518), (195, -0.22551371691705657), (484, 0.5098859238821161),
         (562, 0.022797446828925264), (167, 0.40528423253691925), (169, 0.2517055175539968), (51, 0.06518261024734948), (201, 0.35689718
         12557605), (509, 0.028027592733563728), (17, 0.037900873813750166), (59, 0.07534646139338547), (96, 0.08757565662424828), (520,
```

Figure 36 : Pearson Correlation Dictionary Values

The resulting pearsonCorrelationDict dictionary has user IDs as keys, and Pearson correlation coefficients as values.



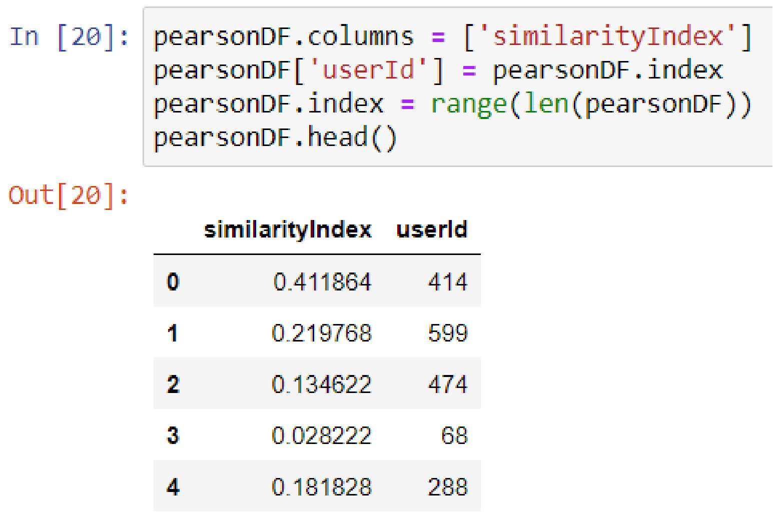
```
In [19]: pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict, orient='index')
         pearsonDF.head()

Out[19]:
```

	0
414	0.411864
599	0.219768
474	0.134622
68	0.028222
288	0.181828

Figure 37 : Pearson_df

This creates a Pandas DataFrame called pearsonDF from the pearsonCorrelationDict dictionary. The from_dict function is used to create a DataFrame from a dictionary, and the orient parameter specifies that the dictionary keys should be used as the index of the DataFrame.

The resulting pearsonDF DataFrame has one column, containing the Pearson correlation coefficients for each user in the userSubsetGroup with respect to the input active user. The index of the DataFrame is the user IDs.



```
In [20]: pearsonDF.columns = ['similarityIndex']
         pearsonDF['userId'] = pearsonDF.index
         pearsonDF.index = range(len(pearsonDF))
         pearsonDF.head()
```

Out[20]:

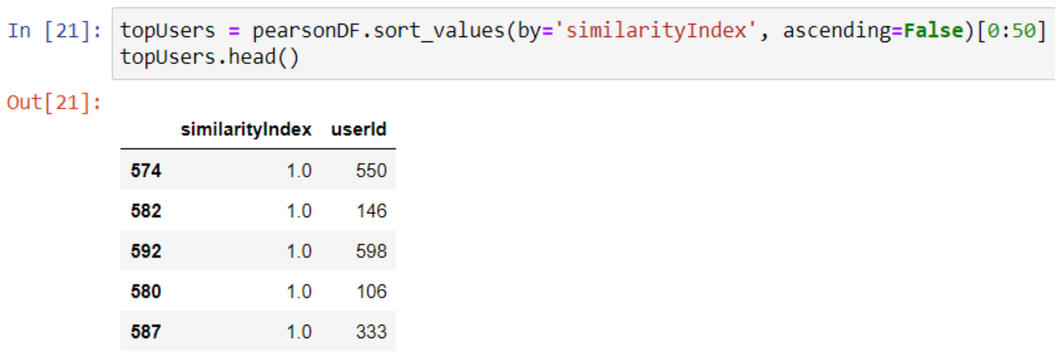
	similarityIndex	userId
0	0.411864	414
1	0.219768	599
2	0.134622	474
3	0.028222	68
4	0.181828	288

Figure 38 : Pearson_df with userid

This code renames the column containing the Pearson correlation coefficients to 'similarityIndex' and adds a new column to the DataFrame called 'userId', which contains the user IDs from the index.

Then it sets the index of the DataFrame to be a new range of integers. This is done to avoid having the user IDs as the index, which could cause issues when merging with other DataFrames later on.

The resulting pearsonDF DataFrame now has two columns: 'similarityIndex' and 'userId'



```
In [21]: topUsers = pearsonDF.sort_values(by='similarityIndex', ascending=False)[0:50]
         topUsers.head()
Out[21]:
```

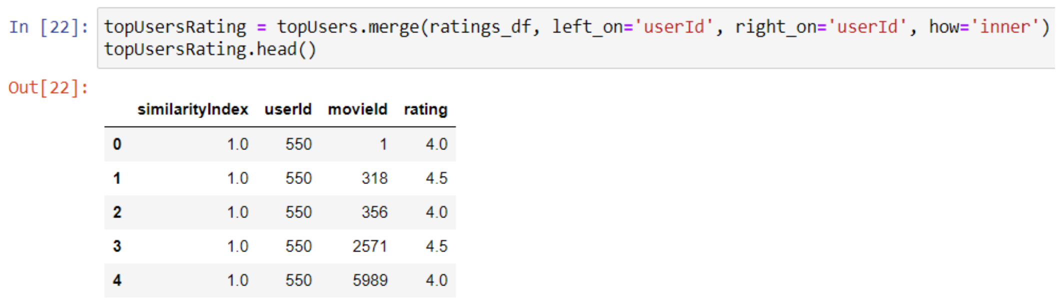
	similarityIndex	userId
574	1.0	550
582	1.0	146
592	1.0	598
580	1.0	106
587	1.0	333

Figure 39 : Top 50 users

This sorts the pearsonDF DataFrame by the 'similarityIndex' column in descending order, so that users with higher similarity to the input active user appear first.

Then it selects the top 50 users with the highest similarity indices, and stores them in a new DataFrame called topUsers.

The resulting topUsers DataFrame has two columns: 'similarityIndex' and 'userId', and contains the top 50 users with the highest similarity to the input active user.



```
In [22]: topUsersRating = topUsers.merge(ratings_df, left_on='userId', right_on='userId', how='inner')
         topUsersRating.head()
Out[22]:
```

	similarityIndex	userId	movieId	rating
0	1.0	550	1	4.0
1	1.0	550	318	4.5
2	1.0	550	356	4.0
3	1.0	550	2571	4.5
4	1.0	550	5989	4.0

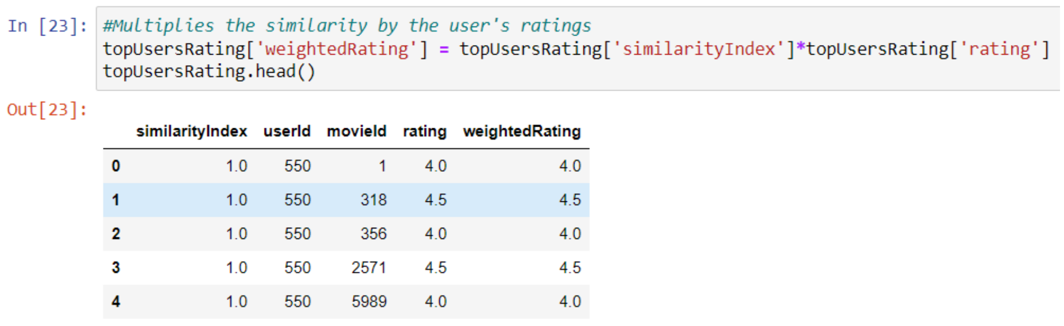
Figure 40 : Top User Ratings

This code merges the topUsers DataFrame with the ratings_df DataFrame, using the 'userId' column as the join key for both DataFrames.

The resulting topUsersRating DataFrame contains all the ratings from the top 50 users with the highest similarity to the input active user.

Since we used an inner join, only the ratings that are common to both DataFrames are included in the resulting DataFrame.

The topUsersRating DataFrame has multiple columns, including 'userId', 'movieId', and 'rating', among others.



```
In [23]: #Multiplies the similarity by the user's ratings
         topUsersRating['weightedRating'] = topUsersRating['similarityIndex']*topUsersRating['rating']
         topUsersRating.head()
Out[23]:
```

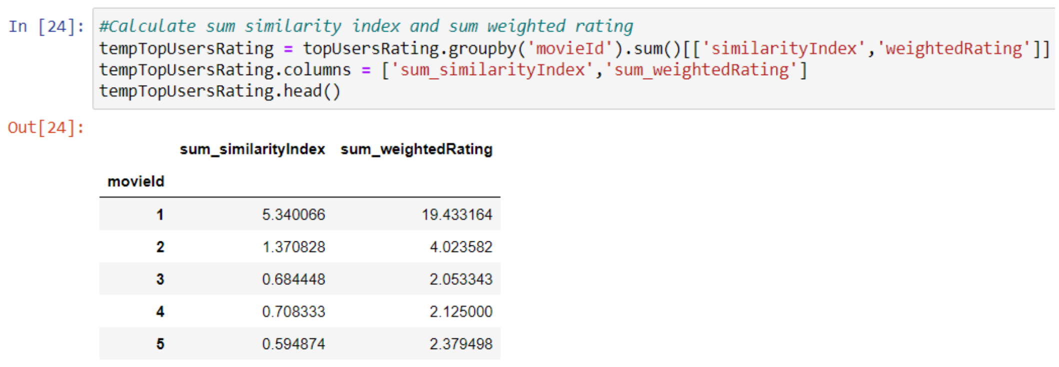
	similarityIndex	userId	movieId	rating	weightedRating
0	1.0	550	1	4.0	4.0
1	1.0	550	318	4.5	4.5
2	1.0	550	356	4.0	4.0
3	1.0	550	2571	4.5	4.5
4	1.0	550	5989	4.0	4.0

Figure 41 : Top user ratings with weighted ratings

Now create a new column in the topUsersRating DataFrame called 'weightedRating', which is the product of the 'similarityIndex' and 'rating' columns.

Multiplying the similarity by the user's rating gives us a weighted rating, where users who are more similar to the input active user have a greater influence on the calculation of the weighted rating.

The resulting DataFrame now includes the 'weightedRating' column in addition to the 'userId', 'movieId', and 'rating' columns.



```
In [24]: #Calculate sum similarity index and sum weighted rating
         tempTopUsersRating = topUsersRating.groupby('movieId').sum()[['similarityIndex','weightedRating']]
         tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
         tempTopUsersRating.head()
Out[24]:
```

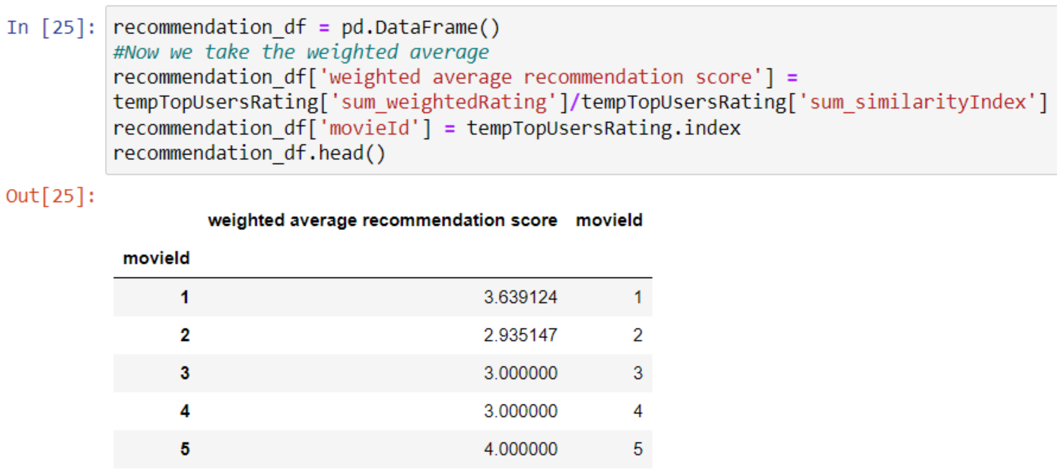
	sum_similarityIndex	sum_weightedRating
movieId		
1	5.340066	19.433164
2	1.370828	4.023582
3	0.684448	2.053343
4	0.708333	2.125000
5	0.594874	2.379498

Figure 42 : TempTopUserRatings

topUsersRating DataFrame is grouped by movieId, and the 'similarityIndex' and 'weightedRating' columns are summed for each movie.

This gives us two new columns in the resulting tempTopUsersRating DataFrame: 'sum_similarityIndex' and 'sum_weightedRating'.

The 'sum_similarityIndex' column represents the sum of the similarity indices for each user who rated the movie, while the 'sum_weightedRating' column represents the sum of the weighted ratings for each user who rated the movie.



```
In [25]: recommendation_df = pd.DataFrame()
         #Now we take the weighted average
         recommendation_df['weighted average recommendation score'] =
         tempTopUsersRating['sum_weightedRating']/tempTopUsersRating['sum_similarityIndex']
         recommendation_df['movieId'] = tempTopUsersRating.index
         recommendation_df.head()
```

Out[25]:

movieId	weighted average recommendation score	movieId
1	3.639124	1
2	2.935147	2
3	3.000000	3
4	3.000000	4
5	4.000000	5

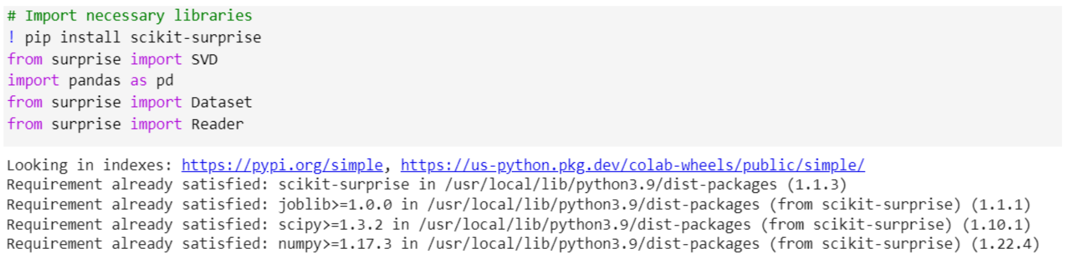
Figure 43 : Recommendation_df

A new DataFrame recommendation_df is created to store the recommended movies and their corresponding weighted average recommendation scores.

The weighted average recommendation score is calculated by dividing the sum of the products of similarity index and ratings (which is stored in the sum_weightedRating column of the tempTopUsersRating DataFrame) by the sum of similarity indices (which is stored in the sum_similarityIndex column of the tempTopUsersRating DataFrame).

The movieId column of tempTopUsersRating is used as the index for the new DataFrame recommendation_df, and is added to the new DataFrame as a column. The head() method is then called on recommendation_df to display the first five rows of the DataFrame.

5.1.3. MODEL – BASED RECOMMENDER SYSTEM



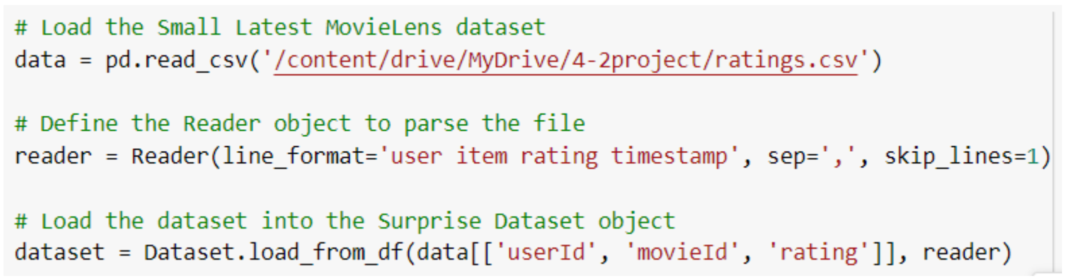
```
# Import necessary libraries
! pip install scikit-surprise
from surprise import SVD
import pandas as pd
from surprise import Dataset
from surprise import Reader
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.9/dist-packages (1.1.3)
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.1.1)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.10.1)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.9/dist-packages (from scikit-surprise) (1.22.4)
```

Figure 44 : Importing python libraries

We import the necessary libraries for a recommendation system built using the Surprise library, which is a Python library for building and analysing recommender systems. Specifically, the code imports the libraries.

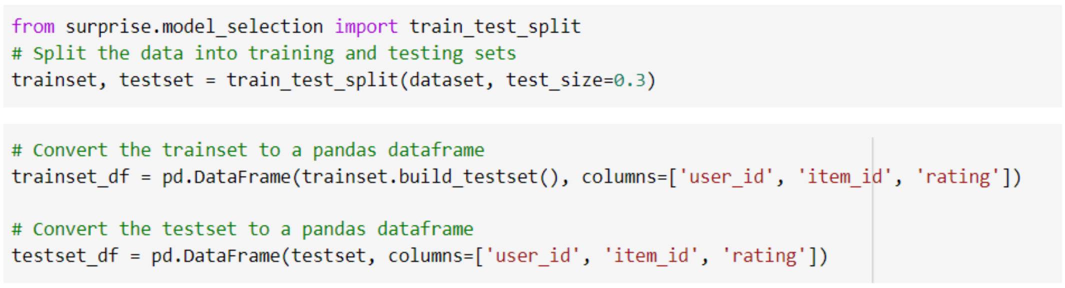
After importing these libraries, the code also checks for and install any necessary packages. In this case, it checks for the availability of the scikit-surprise package and install it if necessary.



```
# Load the Small Latest MovieLens dataset
data = pd.read_csv('/content/drive/MyDrive/4-2project/ratings.csv')

# Define the Reader object to parse the file
reader = Reader(line_format='user item rating timestamp', sep=',', skip_lines=1)

# Load the dataset into the Surprise Dataset object
dataset = Dataset.load_from_df(data[['userId', 'movieId', 'rating']], reader)
```

Figure 45 : Loading dataset

We load the Small Latest MovieLens dataset into a Pandas DataFrame called data from a CSV file located at '/content/drive/MyDrive/4-2project/ratings.csv'. This dataset contains movie ratings from users, where each row in the dataset represents a single rating given by a user to a movie.



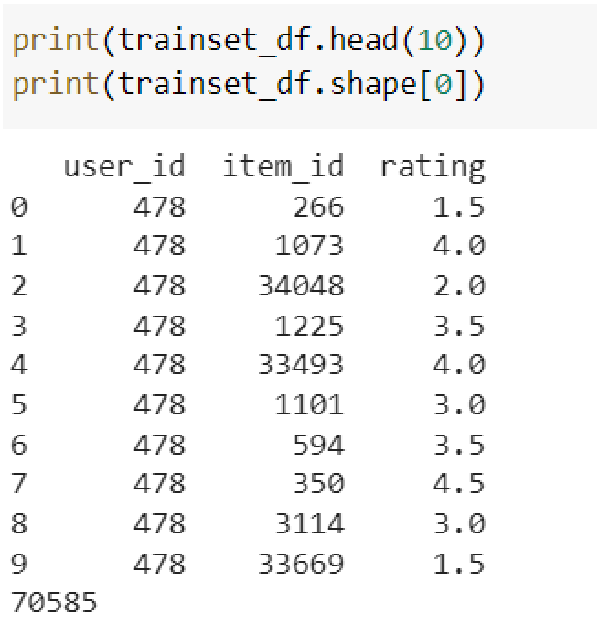
```
from surprise.model_selection import train_test_split
# Split the data into training and testing sets
trainset, testset = train_test_split(dataset, test_size=0.3)
```

```
# Convert the trainset to a pandas dataframe
trainset_df = pd.DataFrame(trainset.build_testset(), columns=['user_id', 'item_id', 'rating'])

# Convert the testset to a pandas dataframe
testset_df = pd.DataFrame(testset, columns=['user_id', 'item_id', 'rating'])
```

Figure 46 : Split the dataset into trainset and testset

We use the train_test_split function from the Surprise model_selection module to split the dataset object into a training set and a testing set. The test_size parameter specifies the proportion of the dataset to use for testing, which is set to 0.3, meaning 30% of the dataset will be used for testing.

The resulting trainset and testset objects are then converted into Pandas DataFrames using the pd.DataFrame() function. The trainset.build_testset() method creates a test set from the training set by removing a certain percentage of the ratings for each user and returning them as a list of tuples containing the user ID, item ID, and rating. The resulting test set is converted to a Pandas DataFrame with the columns 'user_id', 'item_id', and 'rating'.



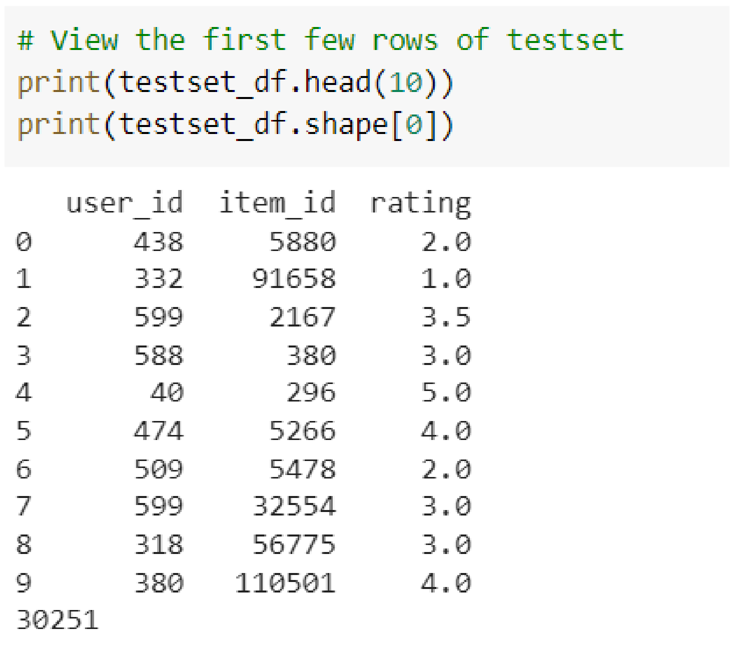
```
print(trainset_df.head(10))
print(trainset_df.shape[0])
```

```
   user_id  item_id  rating
0      478      266     1.5
1      478     1073     4.0
2      478    34048     2.0
3      478     1225     3.5
4      478    33493     4.0
5      478     1101     3.0
6      478      594     3.5
7      478      350     4.5
8      478     3114     3.0
9      478    33669     1.5
70585
```

Figure 47 : Trainset_df

We print the first 10 rows of the trainset_df DataFrame using the head() method. The head() method returns the first n rows of a DataFrame, where n is the argument passed to the method. In this case, head(10) returns the first 10 rows of the trainset_df DataFrame.

The second line of code prints the number of rows in the trainset_df DataFrame using the shape attribute. The shape attribute returns a tuple of the dimensions of the DataFrame, where the first element is the number of rows and the second element is the number of columns. In this case, trainset_df.shape[0] returns the number of rows in the DataFrame.



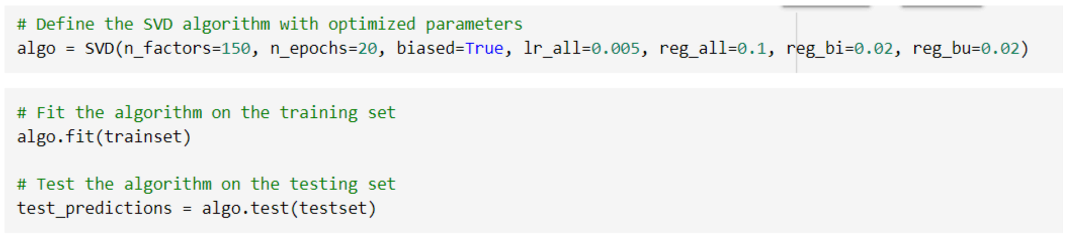
```
# View the first few rows of testset
print(testset_df.head(10))
print(testset_df.shape[0])
```

```
   user_id  item_id  rating
0      438     5880     2.0
1      332    91658     1.0
2      599     2167     3.5
3      588      380     3.0
4       40      296     5.0
5      474     5266     4.0
6      509     5478     2.0
7      599    32554     3.0
8      318    56775     3.0
9      380   110501     4.0
30251
```

Figure 48 : Testset_df

We print the first 10 rows of the testset_df DataFrame using the head() method. The head() method returns the first n rows of a DataFrame, where n is the argument passed to the method. In this case, head(10) returns the first 10 rows of the testset_df DataFrame.

The second line of code prints the number of rows in the testset_df DataFrame using the shape attribute. The shape attribute returns a tuple of the dimensions of the DataFrame, where the first element is the number of rows and the second element is the number of columns. In this case, testset_df.shape[0] returns the number of rows in the DataFrame.



```python
# Define the SVD algorithm with optimized parameters
algo = SVD(n_factors=150, n_epochs=20, biased=True, lr_all=0.005, reg_all=0.1, reg_bi=0.02, reg_bu=0.02)
```

```python
# Fit the algorithm on the training set
algo.fit(trainset)

# Test the algorithm on the testing set
test_predictions = algo.test(testset)
```

Figure 49 : Defining SVD Algorithm

We define an instance of the SVD (Singular Value Decomposition) algorithm with several hyperparameters that have been optimised for the current dataset. The hyperparameters are:

- n_factors: the number of latent factors to use in the matrix factorization. This is set to 150.
- n_epochs: the number of iterations to run during training. This is set to 20.
- biassed: a boolean value indicating whether to use a bias term in the model. This is set to True.
- lr_all: the learning rate for all parameters in the model. This is set to 0.005.
- reg_all: the regularisation term for all parameters in the model. This is set to 0.1.
- reg_bi: the regularisation term for the bias terms in the model. This is set to 0.02.
- reg_bu: the regularisation term for the user factors in the model. This is set to 0.02.

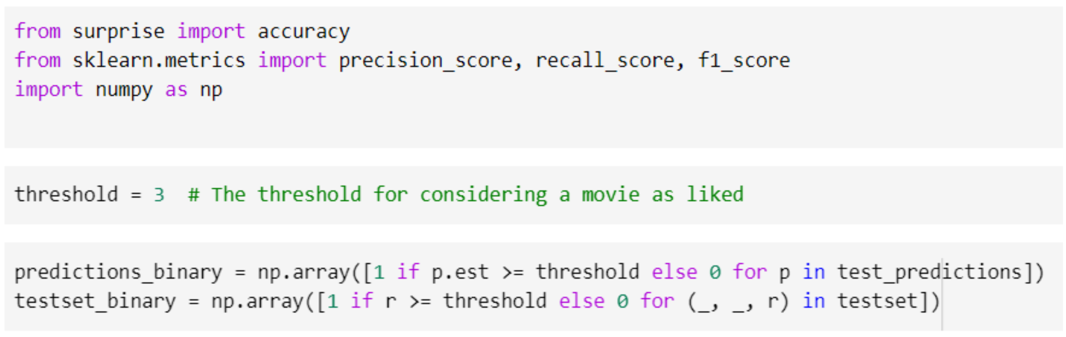
The fit() method is then called on the trainset object to train the algorithm on the training set.

The test() method is called on the algo object with the testset object as the argument to generate predictions for the test set. The resulting test_predictions object is a list of Prediction objects, where each Prediction object contains the predicted rating, the actual rating, and the user and item IDs.

5.2. TESTING

System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system. Although each test has a different purpose, all work to verify that all the system elements have been properly integrated and perform allocated functions. The testing process is actually carried out to make sure that the product exactly does the same thing as what is supposed to do. In the testing stage following goals are tried to achieve.

- To affirm the quality of the project.
- To find and eliminate any residual errors from previous stages.
- To validate the software as a solution to the original problem.
- To provide operational reliability of the system.



```
from surprise import accuracy
from sklearn.metrics import precision_score, recall_score, f1_score
import numpy as np
```

```
threshold = 3  # The threshold for considering a movie as liked
```

```
predictions_binary = np.array([1 if p.est >= threshold else 0 for p in test_predictions])
testset_binary = np.array([1 if r >= threshold else 0 for (_, _, r) in testset])
```

Figure 50 : Considering a threshold

There are various measures to evaluate the recommender system. These metrics are a way to find how accurate a recommender system is. Accuracy is must for any recommender system to work. Performance of any model is judged by these metrics. The evaluation metrics are Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Precision, Recall, F1-Measure, NDCG.

- **Mean Absolute Error**

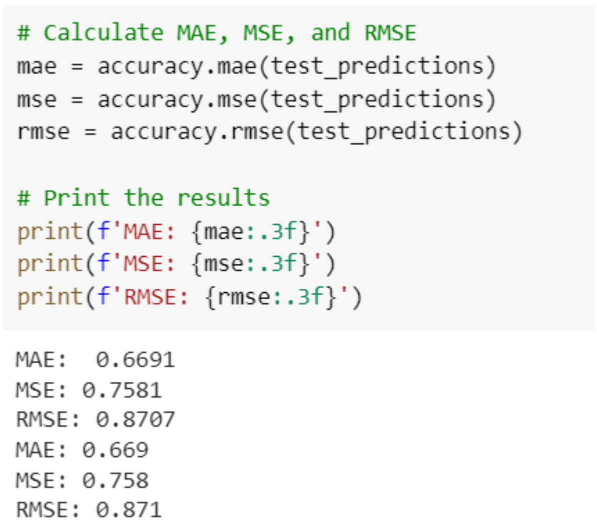
$$MAE = \frac{1}{N} \sum_{(u,i)} |P_{(u,i)} - R_{(u,i)}|$$

Here P(u, i) is the predicted rating for user u on item i , R(u, i) is the actual rating and N is the total number of ratings on the item set. The lower the MAE, the more accurately the recommendation engine predicts user ratings.

- **Root Mean Square Error**

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (P_{(u,i)} - R_{(u,i)})^2}{N}}$$

Root Mean Square Error (RMSE) puts more emphasis on larger absolute error and when RMSE is low the accuracy of the recommender system is better.



```
# Calculate MAE, MSE, and RMSE
mae = accuracy.mae(test_predictions)
mse = accuracy.mse(test_predictions)
rmse = accuracy.rmse(test_predictions)

# Print the results
print(f'MAE: {mae:.3f}')
print(f'MSE: {mse:.3f}')
print(f'RMSE: {rmse:.3f}')
```

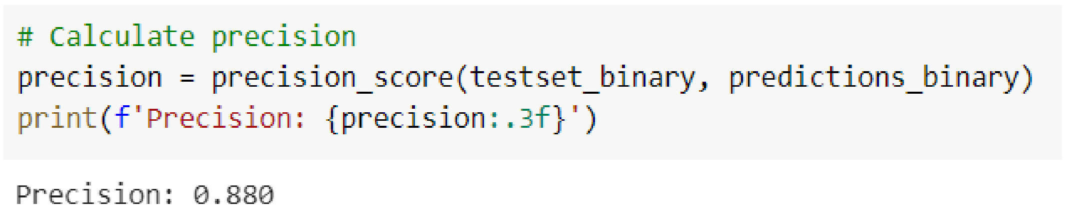
```
MAE:  0.6691
MSE: 0.7581
RMSE: 0.8707
MAE: 0.669
MSE: 0.758
RMSE: 0.871
```

Figure 51 : Calculating MAE,MSE,RMSE

- **Precision**

$$Precision = \frac{Currently\ recommended\ items}{Total\ recommended\ items}$$

Precision shows the result that are relevant , i.e. the items correctly recommended by the system



```
# Calculate precision
precision = precision_score(testset_binary, predictions_binary)
print(f'Precision: {precision:.3f}')
```

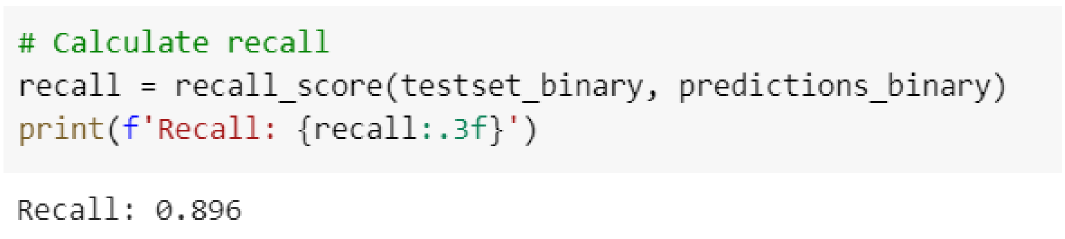
```
Precision: 0.880
```

Figure 52 : Calculating Precision

- **Recall**

$$Recall = \frac{Currently\ recommended\ items}{Total\ useful\ recommended\ items}$$

Recall shows the results which are successfully recommended by the system.



```
# Calculate recall
recall = recall_score(testset_binary, predictions_binary)
print(f'Recall: {recall:.3f}')
```

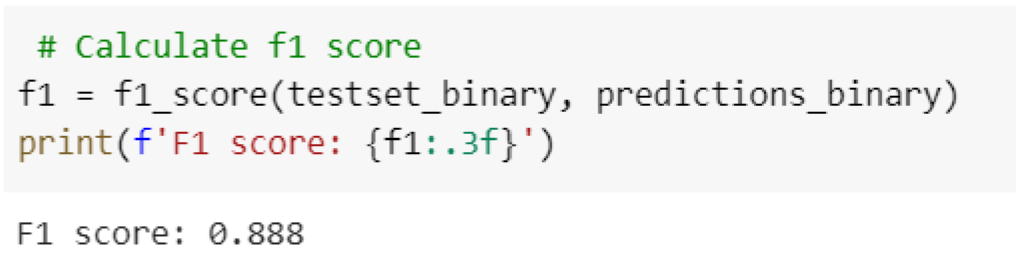
```
Recall: 0.896
```

Figure 53 : Calculating Recall

- **F1-Measure**

$$F_1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

For finding F1-Measure we have to find precision and recall first.



```
# Calculate f1 score
f1 = f1_score(testset_binary, predictions_binary)
print(f'F1 score: {f1:.3f}')
```

```
F1 score: 0.888
```

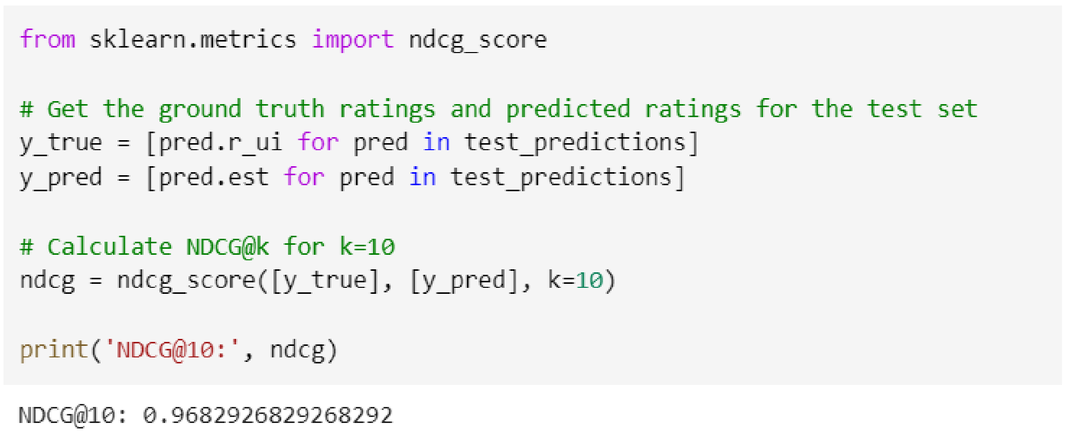
Figure 54 : Calculating F1 score

- **NDCG**

The normalised discounted cumulative gain is the DCG with a normalisation factor in the denominator. The denominator is the ideal DCG score when we recommend the most relevant items first.

$$NDCG_{@K} = \frac{DCG_{@K}}{IDCG_{@K}}$$

$$IDCG_{@K} = \sum_{i=1}^{K^{ideal}} \frac{G_i^{ideal}}{log_2(i+1)}$$



```
from sklearn.metrics import ndcg_score

# Get the ground truth ratings and predicted ratings for the test set
y_true = [pred.r_ui for pred in test_predictions]
y_pred = [pred.est for pred in test_predictions]

# Calculate NDCG@k for k=10
ndcg = ndcg_score([y_true], [y_pred], k=10)

print('NDCG@10:', ndcg)
```

```
NDCG@10: 0.9682926829268292
```

Figure 55 : Calculating NDCG

5.3. SAMPLE CODE

5.3.1. Content Based Filtering

```
import numpy as np

import pandas as pd

movies_df = pd.read_csv('movies.csv')

ratings_df = pd.read_csv('ratings.csv')

#Every genre is separated by a so we simply have to call the split function

movies_df['genres'] = movies_df.genres.str.split('|')

#Drop timestamp column

ratings_df = ratings_df.drop('timestamp', 1)


#Copying the movie dataframe into a new one since we won't need to use
the genre information in our first case.

moviesWithGenres_df = movies_df.copy()

#For every row in the dataframe, iterate through the list of genres and
place a 1 into the corresponding column

for index, row in movies_df.iterrows():

    for genre in row['genres']:

        moviesWithGenres_df.at[index, genre] = 1

#Filling in the NaN values with 0 to show that a movie doesn't have that
column's genre

moviesWithGenres_df = moviesWithGenres_df.fillna(0)

def get_input_user_ratings_table(user_id):
        # Filter the dataframe to get ratings for the specified user ID
        user_ratings_df = ratings_df[ratings_df['userId'] == user_id]
        # Create a table of movies and their ratings for the user
        user_ratings_table = user_ratings_df[['movieId', 'rating']]
    return user_ratings_table
inputMovies = get_input_user_ratings_table(1)
#Filtering out the movies from the input
userMovies=moviesWithGenres_df[moviesWithGenres_df['movieId'].isin(
inputMovies['movieId'].tolist())]
#Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
#Resetting the index to avoid future issues
userMovies = userMovies.reset_index(drop=True)
inputMovies['rating']
#Dot product to get weights
userProfile = userGenreTable.transpose().dot(inputMovies['rating'])
#The user profile
userProfile
#Now let's get the genres of every movie in our original dataframe
genreTable =
moviesWithGenres_df.set_index(moviesWithGenres_df['movieId'])
#And drop the unnecessary information
genreTable = genreTable.drop('movieId', 1).drop('title', 1).drop('genres', 1)
genreTable
```

```
# Get the list of movie IDs from the inputMovies data frame
inputMovieIds = inputMovies['movieId'].tolist()
# Remove the rows with movie IDs that are present in inputMovies
genreTable = genreTable[~genreTable.index.isin(inputMovieIds)]
genreTable
#Multiply the genres by the weights and then take the weighted average
recommendationTable_df =
((genreTable*userProfile).sum(axis=1))/(userProfile.sum())
recommendationTable_df
#Sort our recommendations in descending order
recommendationTable_df =
recommendationTable_df.sort_values(ascending=False)
#Just a peek at the values
recommendationTable_df
recommendationTable_df.head(10)
#The final recommendation table
movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(1
0).keys())]
```

5.3.2. User – User Collaborative Based Filtering

```
import numpy as np
import pandas as pd
movies_df = pd.read_csv('movies.csv')
```

```
ratings_df = pd.read_csv('ratings.csv')

ratings_df = ratings_df.drop('timestamp', 1)

def get_input_user_ratings_table(user_id):

 # Filter the dataframe to get ratings for the specified user ID

 user_ratings_df = ratings_df[ratings_df['userId'] == user_id]

 # Create a table of movies and their ratings for the user

 user_ratings_table = user_ratings_df[['movieId', 'rating']]

            return user_ratings_table

inputMovies = get_input_user_ratings_table(1)

  #Filtering out users that have watched movies that the input active user
  has watched

userSubset =
ratings_df[ratings_df['movieId'].isin(inputMovies['movieId'].tolist())]

userSubset.head()

user_id = 1

userSubset = userSubset[userSubset['userId'] != user_id]

userSubset

#Groupby creates several sub dataframes grouped by userid

userSubsetGroup = userSubset.groupby(['userId'])

userSubsetGroup.get_group(10)

#Sorting it so users with movie most in common with the input will have
priority

#Sort these groups so the users that share the most movies in common
with the input active user have higher priority
```

```
userSubsetGroup = sorted(userSubsetGroup,  key=lambda x: len(x[1]),
reverse=True)

#Top most user

userSubsetGroup[0]

#name of top user group

userSubsetGroup[0][0]

#dataframe of top user group

userSubsetGroup[0][1]

#Store the Pearson Correlation in a dictionary, where the key is the user Id
and the value is the coefficient

pearsonCorrelationDict = {}

#For every user group in our subset

for name, group in userSubsetGroup:

        # Sort the user's movie ratings by movie ID

        group = group.sort_values(by='movieId')

         # Get the input active user's ratings for the movies that the current
        user has also rated

        inputMovies = inputMovies.sort_values(by='movieId')

        # N = Total similar movies watched

        nRatings = len(group)

         # The movies that they both have in common with active user
ratings

        temp_df =
inputMovies[inputMovies['movieId'].isin(group['movieId'].tolist())]

        #Store the ratings in temp_df in a list
```

```
            tempRatingList = temp_df['rating'].tolist()
            #Store the ratings of current user
            tempGroupList = group['rating'].tolist()
            #Now let's calculate the pearson correlation between two users, so
            called, x and y
            Sxx = sum([i**2 for i in tempRatingList]) -
            pow(sum(tempRatingList),2)/float(nRatings)
            Syy = sum([i**2 for i in tempGroupList]) -
            pow(sum(tempGroupList),2)/float(nRatings)
            Sxy = sum( i*j for i, j in zip(tempRatingList, tempGroupList)) -
            sum(tempRatingList)*sum(tempGroupList)/float(nRatings)
            if Sxx != 0 and Syy != 0:
            pearsonCorrelationDict[name] = Sxy/np.sqrt(Sxx*Syy)
            else:
            pearsonCorrelationDict[name] = 0
        pearsonCorrelationDict.items()
        pearsonDF = pd.DataFrame.from_dict(pearsonCorrelationDict,
orient='index')
        pearsonDF.head()
        pearsonDF.columns = ['similarityIndex']
        pearsonDF['userId'] = pearsonDF.index
        pearsonDF.index = range(len(pearsonDF))
        pearsonDF.head()
        topUsers = pearsonDF.sort_values(by='similarityIndex',
ascending=False)[0:50]
```

```
topUsers.head()

topUsersRating = topUsers.merge(ratings_df, left_on='userId',
right_on='userId', how='inner')

topUsersRating.head()

#Multiplies the similarity by the user's ratings

topUsersRating['weightedRating'] =
topUsersRating['similarityIndex']*topUsersRating['rating']

topUsersRating.head()

#Calculate sum similarity index and sum weighted rating

tempTopUsersRating =
topUsersRating.groupby('movieId').sum()[['similarityIndex','weightedRati
ng']]

tempTopUsersRating.columns =
['sum_similarityIndex','sum_weightedRating']

tempTopUsersRating.head()

recommendation_df = pd.DataFrame()

#Now we take the weighted average

recommendation_df['weighted average recommendation score'] =
tempTopUsersRating['sum_weightedRating']/tempTopUsersRating['sum_s
imilarityIndex']

recommendation_df['movieId'] = tempTopUsersRating.index

recommendation_df.head()

#Sort the values based on weighted average recommendation score

recommendation_df = recommendation_df.sort_values(by='weighted
average recommendation score', ascending=False)

recommendation_df.head()
```

```
final_recommendations =
movies_df.loc[movies_df['movieId'].isin(recommendation_df.head(10)['m
ovieId'].tolist())]

final_recommendations

RecommendedMovies = final_recommendations['title'].tolist()

RecommendedMovies
```

5.3.3. Model Based Collaborative Filtering

```
from google.colab import drive

drive.mount('/content/drive')

# Import necessary libraries

! pip install scikit-surprise

from surprise import SVD

import pandas as pd

from surprise import Dataset

from surprise import Reader

# Load the Small Latest MovieLens dataset

data = pd.read_csv('/content/drive/MyDrive/4-2project/ratings.csv')

# Define the Reader object to parse the file

reader = Reader(line_format='user item rating timestamp', sep=',',
skip_lines=1)

# Load the dataset into the Surprise Dataset object

dataset = Dataset.load_from_df(data[['userId', 'movieId', 'rating']], reader)

from surprise.model_selection import train_test_split
```

```
# Split the data into training and testing sets
trainset, testset = train_test_split(dataset, test_size=0.3)
# Convert the trainset to a pandas dataframe
trainset_df = pd.DataFrame(trainset.build_testset(), columns=['user_id',
'item_id', 'rating'])
# Convert the testset to a pandas dataframe
testset_df = pd.DataFrame(testset, columns=['user_id', 'item_id', 'rating'])
print(trainset_df.head(10))
print(trainset_df.shape[0])
# View the first few rows of testset
print(testset_df.head(10))
print(testset_df.shape[0])
# Define the SVD algorithm with optimized parameters
algo = SVD(n_factors=150, n_epochs=20, biased=True, lr_all=0.005,
reg_all=0.1, reg_bi=0.02, reg_bu=0.02)
# Fit the algorithm on the training set
algo.fit(trainset)
# Test the algorithm on the testing set
test_predictions = algo.test(testset)
from surprise import accuracy
from sklearn.metrics import precision_score, recall_score, f1_score
import numpy as np
threshold = 3  # The threshold for considering a movie as liked
```

```
predictions_binary = np.array([1 if p.est >= threshold else 0 for p in
test_predictions])
testset_binary = np.array([1 if r >= threshold else 0 for (_, _, r) in testset])
# Calculate precision
precision = precision_score(testset_binary, predictions_binary)
print(f'Precision: {precision:.3f}')
# Calculate recall
recall = recall_score(testset_binary, predictions_binary)
print(f'Recall: {recall:.3f}')
 # Calculate f1 score
f1 = f1_score(testset_binary, predictions_binary)
print(f'F1 score: {f1:.3f}')
# Calculate MAE, MSE, and RMSE
mae = accuracy.mae(test_predictions)
mse = accuracy.mse(test_predictions)
rmse = accuracy.rmse(test_predictions)
# Print the results
print(f'MAE: {mae:.3f}')
print(f'MSE: {mse:.3f}')
print(f'RMSE: {rmse:.3f}')
from sklearn.metrics import ndcg_score
# Get the ground truth ratings and predicted ratings for the test set
y_true = [pred.r_ui for pred in test_predictions]
y_pred = [pred.est for pred in test_predictions]
```

```
# Calculate NDCG@k for k=10

ndcg = ndcg_score([y_true], [y_pred], k=10)

print('NDCG@10:', ndcg)

# Get the list of all movie ids in the dataset

movie_ids = data['movieId'].unique()

# Let's say we want to recommend movies for user with id 1

user_id = 20

# Get the list of movie ids that the user has already seen

seen_movies = [m for (u, m, _) in testset if u == user_id]

# Get the list of movie ids that the user has not seen

unseen_movies = list(set(movie_ids) - set(seen_movies))

# Print the number of unseen_movies

print(len(unseen_movies))

# Predict the ratings for the unseen movies

predictions = [algo.predict(user_id, movie_id) for movie_id in
unseen_movies]

# Sort the predictions by estimated rating in descending order

predictions.sort(key=lambda x: x.est, reverse=True)

# Join the movie names with the movie ids

movies = pd.read_csv('/content/drive/MyDrive/4-2project/movies.csv')

movie_dict = dict(zip(movies['movieId'], movies['title']))

# Predict the ratings for the unseen movies

predictions = [algo.predict(user_id, movie_id) for movie_id in
unseen_movies]
```

```
# Sort the predictions by estimated rating in descending order
predictions.sort(key=lambda x: x.est, reverse=True)
# Print the top 10 recommendations
for i in range(10):
    movie_id = predictions[i].iid
    movie_name = movie_dict[movie_id]
    print(f"{i+1}. {movie_name}")
```

CHAPTER 6: RESULTS AND COMPARISONS

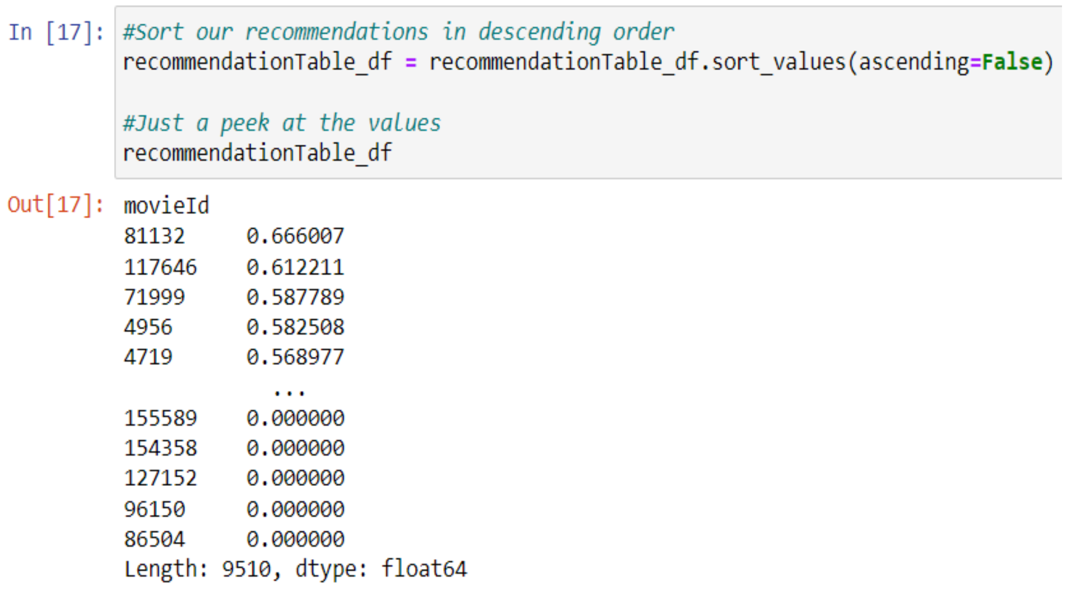
6.1. RESULTS

The results of the project demonstrate the effectiveness of the three different movie recommendation systems that were developed. The content-based recommendation system based on user and item profiling was able to provide recommendations based on the user's preferences and the features of the movies. The user-to-user collaborative filtering system based on Pearson correlation coefficient was able to identify similar users and make recommendations based on the preferences of those users. The model-based collaborative filtering system using SVD algorithm was able to provide accurate recommendations based on latent factors.

The performance metrics for the model based recommendation systems were calculated and analysed. The model-based collaborative filtering system had a precision of 0.880, recall of 0.896, and F1-score of 0.888. These results demonstrate that the model-based collaborative filtering system using SVD algorithm was the most accurate and effective recommendation system among the three.

Overall, the project successfully developed and implemented three different movie recommendation systems and analysed their performance metrics. The results show that the model-based collaborative filtering system using SVD algorithm is a powerful technique for providing accurate and personalised movie recommendations to users.

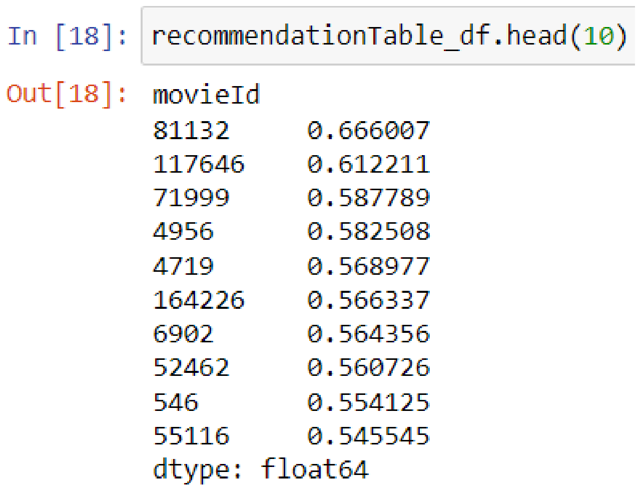
6.1.1. Content – based recommender system



```
In [17]: #Sort our recommendations in descending order
         recommendationTable_df = recommendationTable_df.sort_values(ascending=False)

         #Just a peek at the values
         recommendationTable_df

Out[17]: movieId
         81132     0.666007
         117646    0.612211
         71999     0.587789
         4956      0.582508
         4719      0.568977
                     ...
         155589    0.000000
         154358    0.000000
         127152    0.000000
         96150     0.000000
         86504     0.000000
         Length: 9510, dtype: float64
```

Figure 56 : RecommendationTable_df

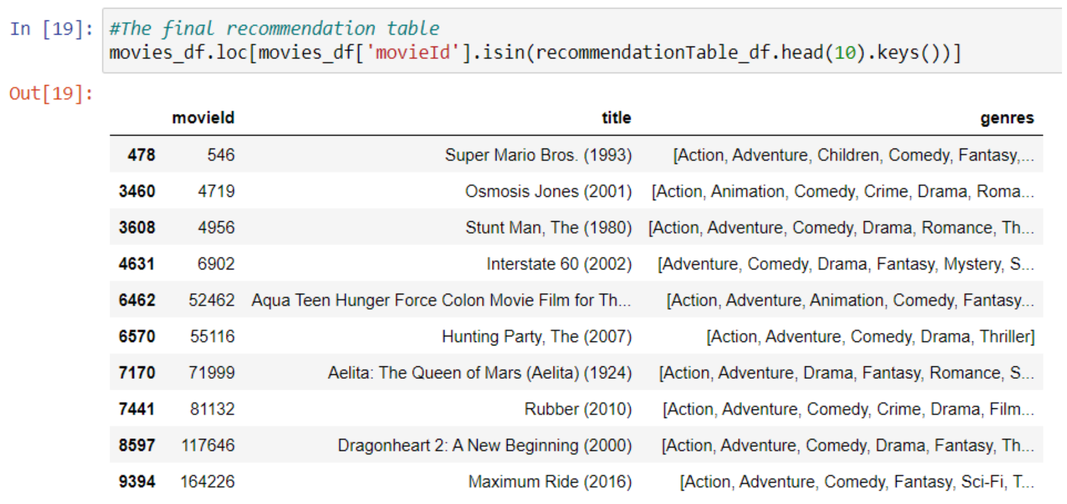
The above code sorts the recommendationTable_df dataframe in descending order, based on the values in each row. This will give us the list of recommended movies with the highest values at the top.



```
In [18]: recommendationTable_df.head(10)

Out[18]: movieId
         81132     0.666007
         117646    0.612211
         71999     0.587789
         4956      0.582508
         4719      0.568977
         164226    0.566337
         6902      0.564356
         52462     0.560726
         546       0.554125
         55116     0.545545
         dtype: float64
```

Figure 57 : Top 10 Recommendations

Here we are displaying the top 10 movie recommendations for the user based on their input ratings. The recommendationTable_df DataFrame has been sorted in descending order of recommendation score, so we can simply display the first 10 rows of the DataFrame to get the top 10 recommended movies.



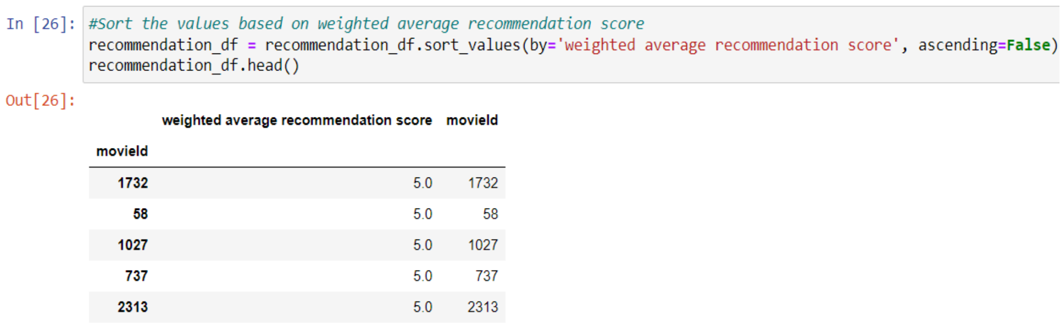
```
In [19]: #The final recommendation table
         movies_df.loc[movies_df['movieId'].isin(recommendationTable_df.head(10).keys())]
Out[19]:
```

	movieId	title	genres
478	546	Super Mario Bros. (1993)	[Action, Adventure, Children, Comedy, Fantasy,...
3460	4719	Osmosis Jones (2001)	[Action, Animation, Comedy, Crime, Drama, Roma...
3608	4956	Stunt Man, The (1980)	[Action, Adventure, Comedy, Drama, Romance, Th...
4631	6902	Interstate 60 (2002)	[Adventure, Comedy, Drama, Fantasy, Mystery, S...
6462	52462	Aqua Teen Hunger Force Colon Movie Film for Th...	[Action, Adventure, Animation, Comedy, Fantasy...
6570	55116	Hunting Party, The (2007)	[Action, Adventure, Comedy, Drama, Thriller]
7170	71999	Aelita: The Queen of Mars (Aelita) (1924)	[Action, Adventure, Drama, Fantasy, Romance, S...
7441	81132	Rubber (2010)	[Action, Adventure, Comedy, Crime, Drama, Film...
8597	117646	Dragonheart 2: A New Beginning (2000)	[Action, Adventure, Comedy, Drama, Fantasy, Th...
9394	164226	Maximum Ride (2016)	[Action, Adventure, Comedy, Fantasy, Sci-Fi, T...

Figure 58 : Recommended top 10 movies

It will return the top 10 movie recommendations based on the user's input ratings. The recommendationTable_df contains a weighted average of genres for all movies in the dataset, where the weights come from the user's input ratings. The code then sorts this table in descending order and selects the top 10 movies based on the highest weighted average. Finally, the code returns the movie information (title, genres, etc.) for these top 10 movies by filtering the movies_df data frame for the rows where the movieId matches the movieIds in the top 10 recommendations.

6.1.2. User – user collaborative recommender system

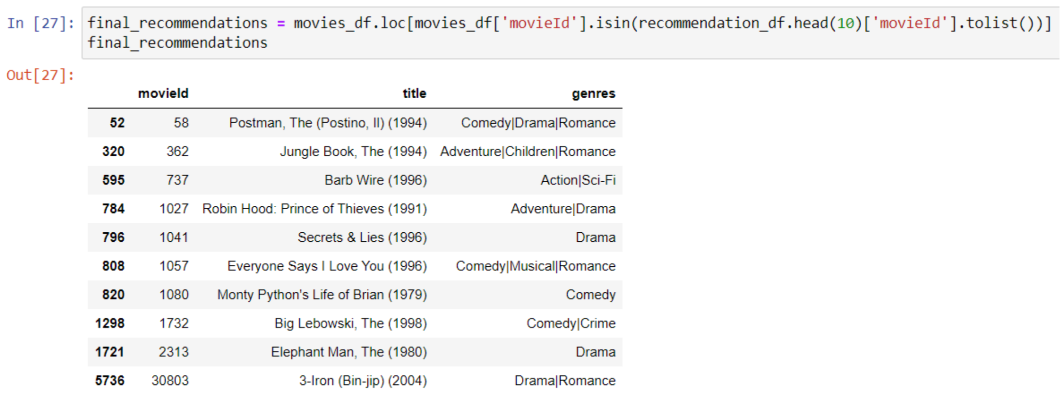


```
In [26]: #Sort the values based on weighted average recommendation score
         recommendation_df = recommendation_df.sort_values(by='weighted average recommendation score', ascending=False)
         recommendation_df.head()
```

```
Out[26]:
         weighted average recommendation score  movieId
movieId
1732                                       5.0     1732
58                                         5.0       58
1027                                       5.0     1027
737                                        5.0      737
2313                                       5.0     2313
```

Figure 59 : Weighted average recommendation score

Now sort the movie recommendations based on their weighted average recommendation score in descending order, so that the movies with the highest score are at the top of the list. The sort_values function is used to sort the data frame recommendation_df based on the column 'weighted average recommendation score', with ascending=False argument used to sort the values in descending order. The resulting data frame contains the recommended movies sorted in the order of highest to lowest recommendation score.

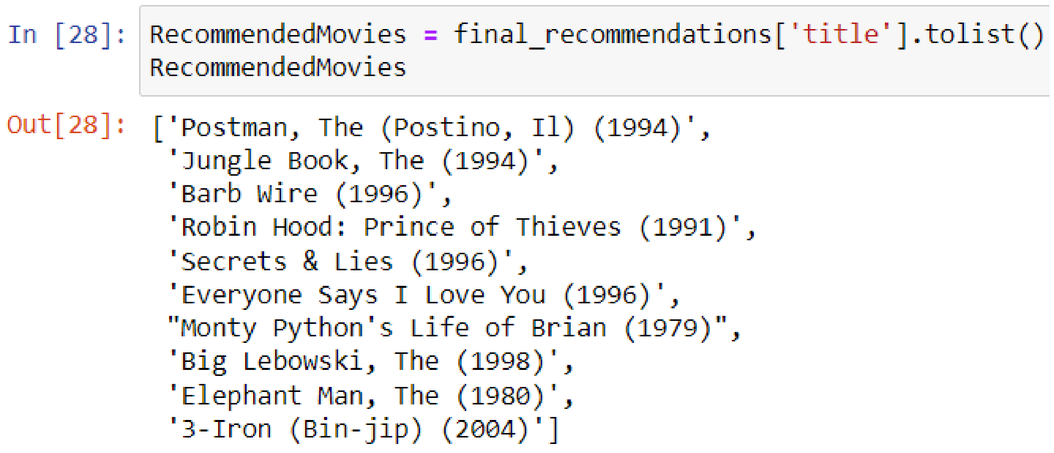


```
In [27]: final_recommendations = movies_df.loc[movies_df['movieId'].isin(recommendation_df.head(10)['movieId'].tolist())]
         final_recommendations
```

```
Out[27]:
      movieId                                  title                      genres
52         58        Postman, The (Postino, Il) (1994)     Comedy|Drama|Romance
320       362                       Jungle Book, The (1994)  Adventure|Children|Romance
595       737                            Barb Wire (1996)             Action|Sci-Fi
784      1027   Robin Hood: Prince of Thieves (1991)           Adventure|Drama
796      1041                        Secrets & Lies (1996)                    Drama
808      1057                 Everyone Says I Love You (1996)   Comedy|Musical|Romance
820      1080           Monty Python's Life of Brian (1979)                 Comedy
1298     1732                       Big Lebowski, The (1998)            Comedy|Crime
1721     2313                      Elephant Man, The (1980)                   Drama
5736    30803                         3-Iron (Bin-jip) (2004)           Drama|Romance
```

Figure 60 : Final Recommendation

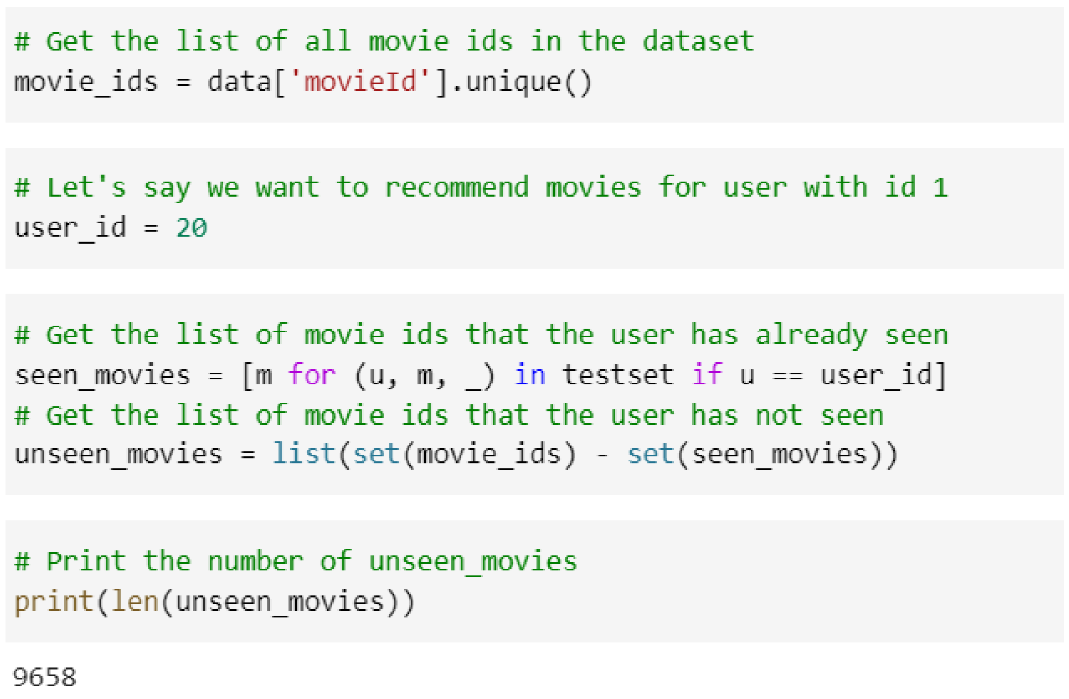
Select the top 10 recommended movies based on their weighted average recommendation score calculated in the previous step. It first extracts the movieIds of the top 10 movies from the 'recommendation_df' DataFrame and then uses them to filter the 'movies_df' DataFrame to obtain the final recommendations. The resulting DataFrame 'final_recommendations' contains the details of the top 10 recommended movies, including their title, genre, and year of release.



```
In [28]: RecommendedMovies = final_recommendations['title'].tolist()
         RecommendedMovies

Out[28]: ['Postman, The (Postino, Il) (1994)',
          'Jungle Book, The (1994)',
          'Barb Wire (1996)',
          'Robin Hood: Prince of Thieves (1991)',
          'Secrets & Lies (1996)',
          'Everyone Says I Love You (1996)',
          "Monty Python's Life of Brian (1979)",
          'Big Lebowski, The (1998)',
          'Elephant Man, The (1980)',
          '3-Iron (Bin-jip) (2004)']
```

Figure 61 : Top 10 recommended movies for user id 1

6.1.3. Model – based recommender system



```
# Get the list of all movie ids in the dataset
movie_ids = data['movieId'].unique()
```

```
# Let's say we want to recommend movies for user with id 1
user_id = 20
```

```
# Get the list of movie ids that the user has already seen
seen_movies = [m for (u, m, _) in testset if u == user_id]
# Get the list of movie ids that the user has not seen
unseen_movies = list(set(movie_ids) - set(seen_movies))
```

```
# Print the number of unseen_movies
print(len(unseen_movies))
```

```
9658
```

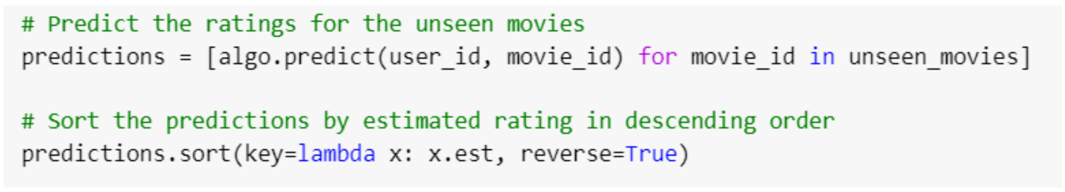
Figure 62 : Generating the unseen movies list

The code is a part of a movie recommendation system implementation using the Surprise library in Python. It first loads a small movie ratings dataset into a pandas DataFrame, which contains user IDs, movie IDs, and corresponding ratings.

The code then defines a Reader object to parse the dataset, and loads it into the Surprise Dataset object. The dataset is then split into training and testing sets using the train_test_split method.

After training an instance of the SVD algorithm with optimised hyperparameters, the code then generates a list of movie IDs that a specific user (with ID 20 in this case) has already seen in the testing set. It then creates a list of the movie IDs that the user has not seen by subtracting the seen movie IDs from the list of all movie IDs in the dataset. Finally, it prints the number of movies that the user has not seen.

This is useful because it helps in identifying the set of movies that the recommendation system should consider when recommending movies to this user, as it should only recommend movies that the user has not seen before.



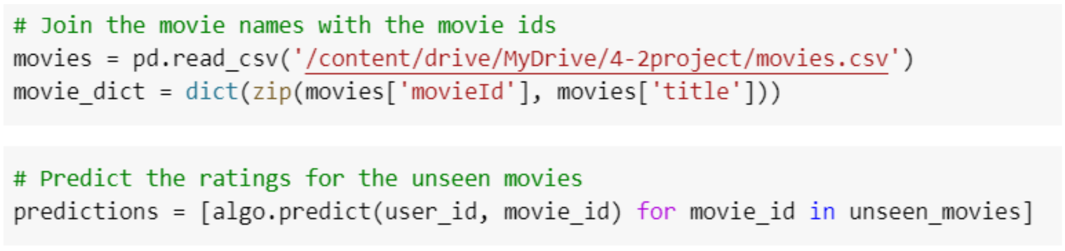
```python
# Predict the ratings for the unseen movies
predictions = [algo.predict(user_id, movie_id) for movie_id in unseen_movies]

# Sort the predictions by estimated rating in descending order
predictions.sort(key=lambda x: x.est, reverse=True)
```

Figure 63 : Sorting the predictions in descending order

We first create a list of predictions using a list comprehension. In this comprehension, the SVD algorithm predict() method is called for each movie ID in the unseen_movies list, passing in the user_id as the user parameter. The method returns a Prediction object containing the estimated rating for the given user and movie.

The predictions list contains these Prediction objects for all unseen movies.

The code then sorts these predicted ratings in descending order using the sort() method with a key function that sorts the list of Prediction objects based on their est attribute, which contains the estimated rating. This results in a list of predicted ratings sorted in descending order of their estimated value.

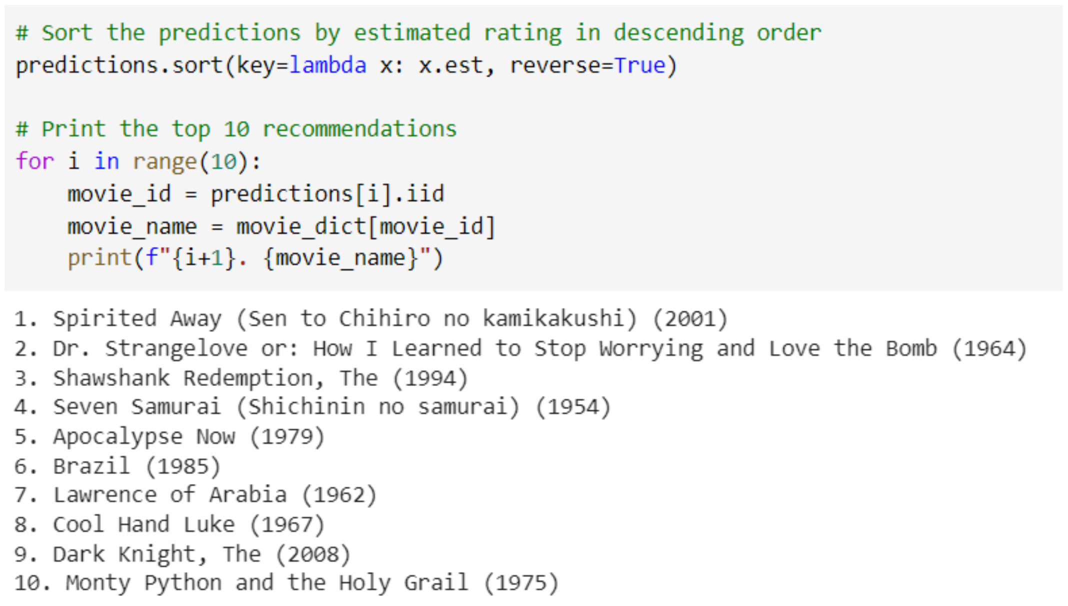


```python
# Join the movie names with the movie ids
movies = pd.read_csv('/content/drive/MyDrive/4-2project/movies.csv')
movie_dict = dict(zip(movies['movieId'], movies['title']))
```

```python
# Predict the ratings for the unseen movies
predictions = [algo.predict(user_id, movie_id) for movie_id in unseen_movies]
```

Figure 64 : Predictions along with movieid

The purpose of this code is to map the movie IDs to their corresponding names so that the recommendations can be displayed to the user in a more understandable format. This is done by using the movie_dict dictionary to look up the movie names for each movie ID in the sorted predictions list.



```python
# Sort the predictions by estimated rating in descending order
predictions.sort(key=lambda x: x.est, reverse=True)

# Print the top 10 recommendations
for i in range(10):
    movie_id = predictions[i].iid
    movie_name = movie_dict[movie_id]
    print(f"{i+1}. {movie_name}")
```

```
1. Spirited Away (Sen to Chihiro no kamikakushi) (2001)
2. Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)
3. Shawshank Redemption, The (1994)
4. Seven Samurai (Shichinin no samurai) (1954)
5. Apocalypse Now (1979)
6. Brazil (1985)
7. Lawrence of Arabia (1962)
8. Cool Hand Luke (1967)
9. Dark Knight, The (2008)
10. Monty Python and the Holy Grail (1975)
```

Figure 65 : Top 10 Recommendations

In the above code, the predictions list is sorted in descending order based on the estimated rating of each movie. Using a loop, the top 10 recommendations are printed to the console, where the movie_id variable stores the ID of each recommended movie, and the movie_name variable is used to look up the corresponding movie name from the movie_dict dictionary.

The output of this code is a ranked list of the top 10 movies that the user with the given ID (in this case, user ID 20) is most likely to enjoy, based on the predicted ratings for the movies they have not yet seen.

6.2. COMPARISON

Both content-based and user-to-user collaborative filtering approaches were implemented and evaluated for the movie recommendation system. Both approaches had their strengths and limitations, and the choice of approach depends on the specific requirements and constraints of the application. In future work, it may be interesting to explore hybrid approaches that combine the strengths of both content-based and user-to-user collaborative filtering methods.

CHAPTER 7: CONCLUSION AND FUTURE ENHANCEMENT

7.1. CONCLUSION

In conclusion, this project aimed to build three different movie recommendation systems using content-based, user-to-user collaborative filtering, and model-based collaborative filtering techniques. We implemented these systems using Python and Jupyter Notebook and tested them on the Movielens small 10k dataset.

The content-based system used user profiles and item profiling to recommend movies based on their similarity to movies previously liked by the user. The user-to-user collaborative filtering system used Pearson correlation coefficient to find the similarity between users and recommend movies that similar users liked. The model-based collaborative filtering system used Singular Value Decomposition (SVD) algorithm to predict movie ratings and recommend movies based on predicted ratings. We evaluated the performance of the model based collaborative filtering system using precision, recall, NDCG and F1-score metrics.

In conclusion, the results suggest that the user-to-user collaborative filtering system outperformed the content-based system in terms of recommendations on smaller datasets but overall the model-based collaborative filtering resulted to be effective for large datasets. Overall, this project provides a valuable contribution to the field of recommendation systems and their practical applications.

7.2. FUTURE ENHANCEMENT

Recommender systems have developed for many years, which ever entered a low point. In the past few years, the development of machine learning, large-scale network and high performance computing is promoting new development in this field. We will consider the following aspects in future work.

- **Introduce more precise and proper features of the movie**. Typical collaborative filtering recommendations use the rating instead of object features. In the future we should extract features such as colour and subtitles from movies which can provide a more accurate description for movies.
- **Introducing users who dislike movie lists**. The user data is always useful in recommender systems. In the future we will collect more user data and add a user dislike movie list. We will input the dislike movie list into the recommender system as well and generate scores that will be added to previous results. By this way we can improve the result of the recommender system.
- **Make the recommender system as an internal service**. In the future, the recommender system is no longer an external website that will be just for testing. We will make it as an internal APIs for developers to invoke. Some movie lists on the website will be sorted by recommendation.
- **Introducing machine learning.** For future study, dynamic parameters will be introduced into the recommender system. We will use machine learning to adjust the weight of each feature automatically and find the most suitable weights.

CHAPTER 8: REFERENCE

1. Francesco Ricci, Lior Rokach, Bracha Shapira. : Recommender Systems Handbook. Second Edition. Springer pp., H.: Matrix Factorization algorithms of model-based recommendation schemes.
2. Pasquale Lops, Marco De Gemmis, and Giovanni Semeraro. Content-based recommender systems: State of the art and trends. In the Recommender systems handbook, pages 73–105. Springer, 2011.
3. Hu, Y., Koren, Y., & Volinsky, C. (2008). Collaborative Filtering for Implicit Feedback Datasets. Proceedings of the 2008 Eighth IEEE International Conference on Data Mining, 263-272.
4. Ndiaye, M. N., & Senghor, S. (2019). Movie Recommendation System Using Singular Value Decomposition Algorithm. Journal of Physics: Conference Series.
5. Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). A Survey of Collaborative Filtering Techniques. International Journal of Computer Science and Information Technology.
6. Kore, S. A., & Patil, S. B. (2018). Analysis of User-Based and Item-Based Collaborative Filtering Recommendation Algorithms for Movie Recommendation System. International Journal of Computer Applications.
7. Singh, D., & Singh, M. (2017). A Review of Recommendation Systems. International Journal of Advanced Research in Computer Science.
8. Vaidya, S. S., & Agarkar, S. S. (2018). Movie Recommendation System Using Collaborative Filtering Algorithm. International Journal of Recent Technology and Engineering.
9. Panigrahi, M., & Barik, R. K. (2019). A Review on Movie Recommendation System Using Collaborative Filtering Technique. International Journal of Recent Technology and Engineering.
10. Y. Koren, R. Bell, and C. Volinsky, ”Matrix Factorization Techniques For Recommender Systems”, Computer 42(8):30-37, 2009. DOI: http://dx.doi.org/10.1109/MC.2009.263