A black background with orange and blue text

Description automatically generated



**Mini Capstone Report  
AIL391m– AI1801**

**Collaborative Filtering-based Recommender System using K Nearest Neighbors**

**Mentor: Tran Ngoc Hoang**

**Can Tho, Nov 2024**

**Group 2: Mini Capstone**

| **Name of group member** | **Contribution (%)** |
| --- | --- |
| 1. Dao Anh Khoa (Leader) | 20% |
| 2. Huynh Anh Phuong | 20% |
| 3. Nguyen Van Phong | 20% |
| 4. Huynh Ngoc Nhu Quynh | 20% |
| 5. Tran Trung Nhan | 20% |

**Table of Contents**

[**I. Introduction 3**](#_1fob9te)

[**II. Mathematical background 3**](#_2et92p0)

[**1. User-based and Item-based 3**](#_3dy6vkm)

[**2. KNN in the realm of collaborative filtering 4**](#_1t3h5sf)

[**III. Implementations 7**](#_3rdcrjn)

[**1. Dataset – preprocessing: 7**](#_26in1rg)

[**2. Implementation with sklearn-surprise: 12**](#_lnxbz9)

[**3. Implementation with numpy, pandas and sklearn**](#_35nkun2)

**IV. Summary:****22**

# **I. Introduction**

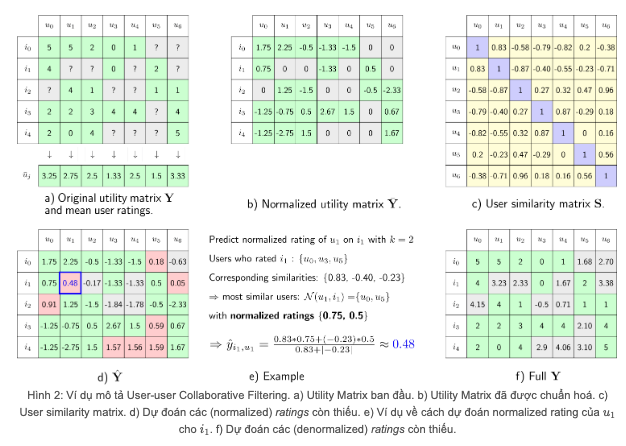
K-Nearest Neighbors (KNN) is a versatile machine learning algorithm that can be applied to various tasks, including recommendation systems. In the realm of collaborative filtering, KNN is employed to predict user preferences based on the ratings or behaviors of similar users. This technique leverages the principle of assuming that users with similar tastes will tend to rate items similarly.

There are two approaches to collaborative filtering recommendation systems: User-based and Item-based.

# **II. Mathematical background**

## **User-based and Item-based**

**User-based** collaborative filtering is based on the user similarity or neighborhood

****

**Item-based** collaborative filtering is based on similarity among items

A screenshot of a grid

Description automatically generated

## **KNN in the realm of collaborative filtering**

KNN in the cases of both User-based and Item-based are generally similar, let’s dive into the case of User-based.

**User-based:**

User-based collaborative filtering looks for users who are similar. This is analogous to a clustering problem, where we employed explicit user profiles to calculate user similarity. However, the user profiles may not be available, so how can we determine if two users are similar?

A black text on a white background

Description automatically generated

where:

* A⋅B is the dot product.
* ∥A∥ is the norm of vector A.
* ∥B∥ is the norm of vector B.







**User-item interaction matrix**

For most collaborative filtering-based recommender systems, the main dataset format is a 2-D matrix called the user-item interaction matrix. In the matrix, its row is labeled as the user id/index and column labeled to be the item id/index, and the element (i, j) represents the rating of user i to item j.

Below is a simple example of a user-item interaction matrix:A table with numbers and text

Description automatically generated with medium confidence

**KNN-based collaborative filtering**

As we can see from above, each row vector represents the rating history of a user and each column vector represents the users who rated the item. A user-item interaction matrix is usually very sparse as you can imagine one user very likely only interacts with a very small subset of items and one item is very likely to be interacted by a small subset of users.

Now to determine if two users are similar, we can simply calculate the similarities between their row vectors in the interaction matrix. Then based on the similarity measurements, we can find the k nearest neighbor as the similar users.

Item-based collaborative filtering works similarly, we just need to look at the user-item matrix vertically. Instead of finding similar users, we are trying to find similar items (courses). If two courses are enrolled by two groups of similar users, then we could consider the two items are similar and use the known ratings from the other users to predict the unknown ratings.

If we formulate the KNN based collaborative filtering, the predicted rating of user uu to item ii, ^r\_uir^\_ui is given by:

**A screenshot of a math problem

Description automatically generated**Let's illustrate how the equation works using a simple example. From the above figure, suppose we want to predict the rating of user6 to item Machine Learning Capstone course. After some similarity measurements, we found that k = 4 nearest neighbors: user2, user3, user4, user5 with similarities in array knn\_sims:

**import** numpy **as** np

**import** math

*# An example similarity array stores the similarity of user2, user3, user4, and user5 to user6*

knn\_sims **=** np**.**array([0.8, 0.92, 0.75, 0.83])

Also their rating on the Machine Learning Capstone course are:

*# 2.0 means audit and 3.0 means complete the course*

knn\_ratings **=** np**.**array([3.0, 3.0, 2.0, 3.0])

So the predicted rating of user6 to item Machine Learning Capstone course can be calculated as:

r\_u6\_ml **=**  np**.**dot(knn\_sims, knn\_ratings)**/** sum(knn\_sims)

r\_u6\_ml

If we already know the true rating to be 3.0, then we get a prediction error RMSE (Rooted Mean Squared Error) as:

true\_rating **=** 3.0

rmse **=** math**.**sqrt(true\_rating **-** r\_u6\_ml) **\*\*** 2

rmse

The predicted rating is around 2.7 (close to 3.0 with RMSE 0.22), which indicates that user6 is also likely to complete the course Machine Learning Capstone. As such, we may recommend it to user6 with high confidence.

# **III. Implementations**

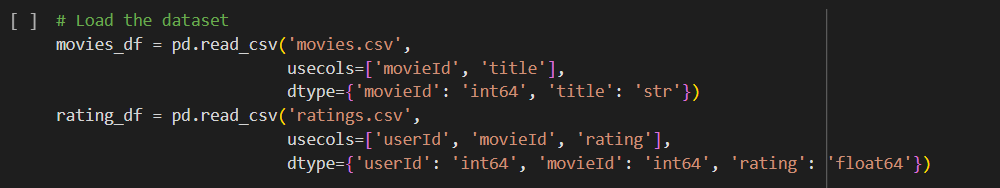
## **Dataset – preprocessing:**

We here use a movie recommendation system dataset available at:  
[*https://github.com/krishnaik06/Recommendation\_complete\_tutorial/tree/master/KNN%20Movie%20Recommendation*](https://github.com/krishnaik06/Recommendation_complete_tutorial/tree/master/KNN%20Movie%20Recommendation)

* 1. **Load the dataset:**

First, we clone the dataset from github:**

*Fig 1.1.1: Cloning data from github*

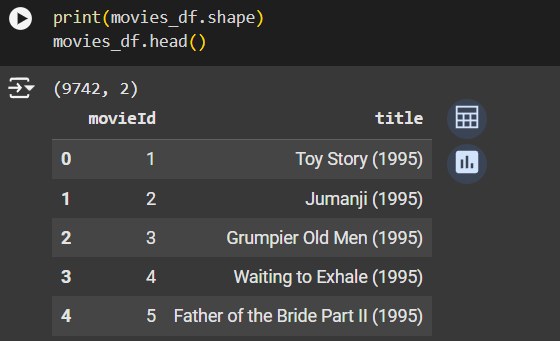
Then, we load the dataset into DataFrames using pandas:

*Fig 1.1.2: Loading movies\_df and rating\_df from movies.csv and ratings.csv*

**movies\_df contains:**

movieId: distinct ids for movies

title: titles of the movies

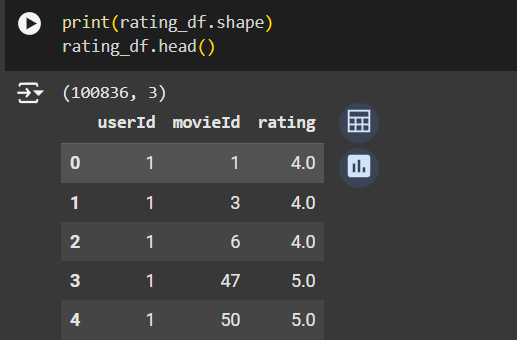
*  
Fig 1.1.3: movie\_ddf*

**rating\_df contains:**

userId: Indistinct ids for user corresponding to each rating

movieId: Ids of movies being rated

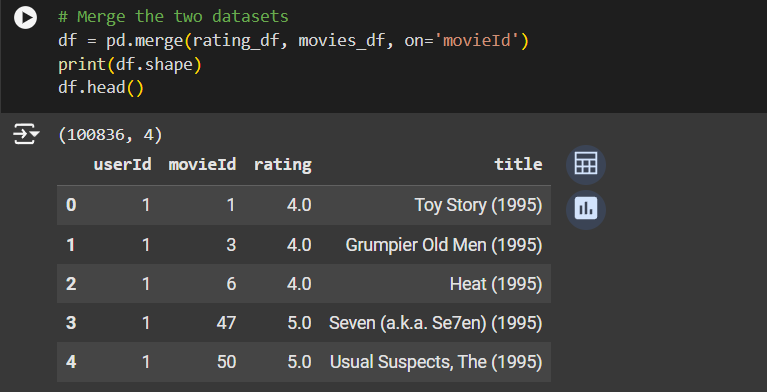
rating: rating for each movie by each user



*Fig 1.1.4: rating\_df*

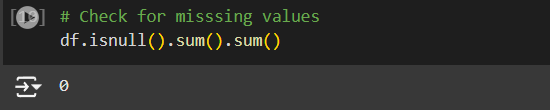
* 1. **Preprocessing:**

We need three columns: user, movie, rating; hence, we merge two df into one on ‘movieId’



*Fig 1.2.1: Merging rating\_df with movies\_df on movieId*

Now we check for NaN values. There is no NaN value.



*Fig 1.2.2: Check for NaN values*

**Semantic Issue Resolving:**

For a recommendation system to work semantically correctly, each

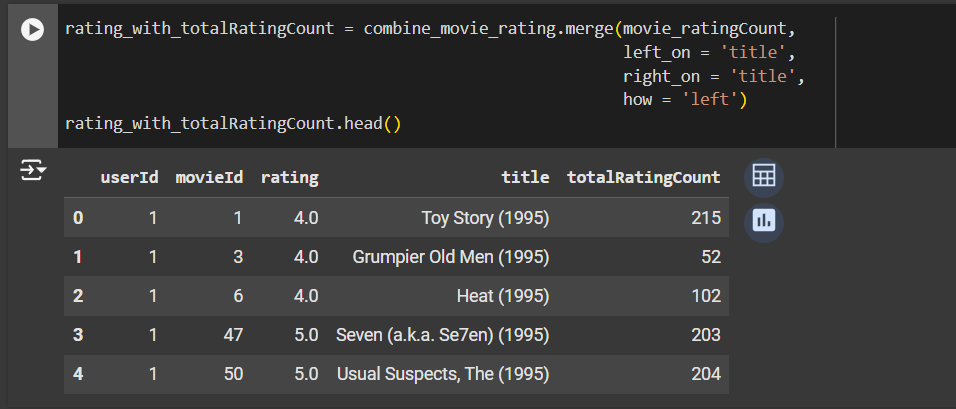
movie (item) needs to be rated by more than a certain number of users. Here we declare that number to be 50, meaning movies with lower number of ratings than 50 will be filtered out.

This step helps ensure that the data is clean and suitable for training the film production model. The missing value check step helps eliminate invalid data, while filtering out underrated movies to ensure the accuracy of the system's output.

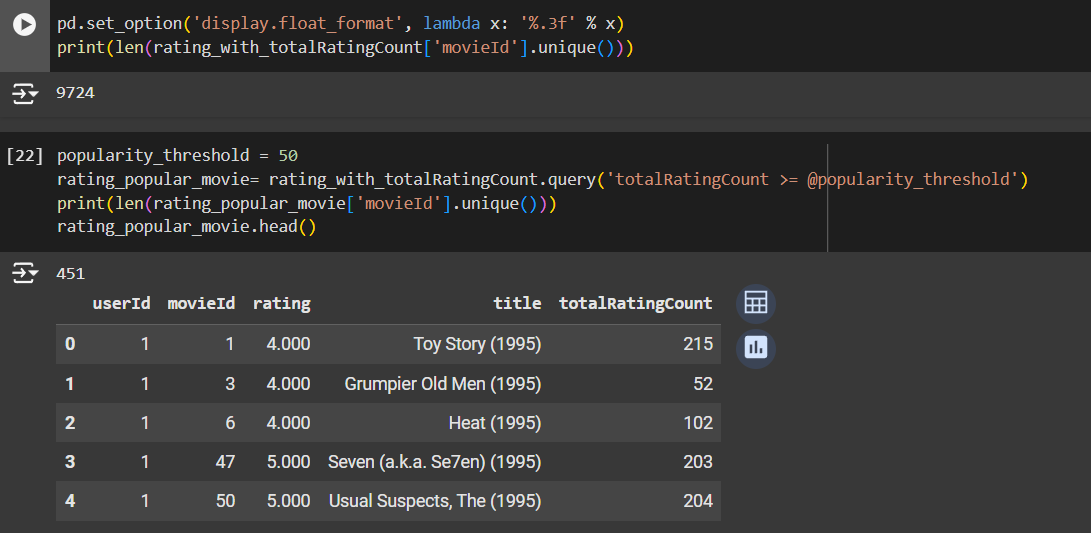
The system can prioritize recommending movies with a higher number of reviews, because the ratings provide less variance so it’s more likely to be credible and reliable.



*Fig 1.2.3: Total rating count of each movie*



*Fig 1.2.4: Adding the totalRatingCount to our combined data frame*



*Fig 1.2.5: Highlighting the number of movies with equal or more than 50 rating records.*

Only 451/9724 movies satisfy the condition.

*Caution:*

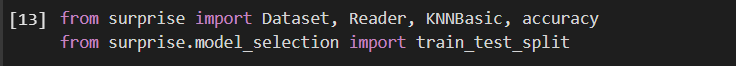
The minimum number of reviews threshold may vary depending on the requirements of the proposed system.

In addition to the number of reviews threshold, other criteria can be used to filter movies, such as release time, genre, etc.

## **Implementation with sklearn-surprise:**

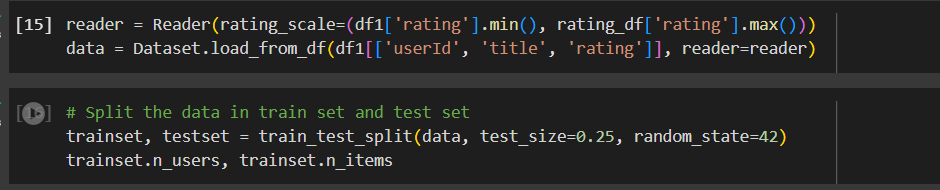
**Introduction**: scikit-surprise is a Python library designed for recommender systems. It provides algorithmic tips such as collaborative filtering, content-based tips, and hybrid algorithms. scikit-surprise is built to easily integrate with tools like scikit-learn and supports training and evaluating tip models with user-target data matrices (user-item matrices).

* 1. **Import modules, load the dataset:**

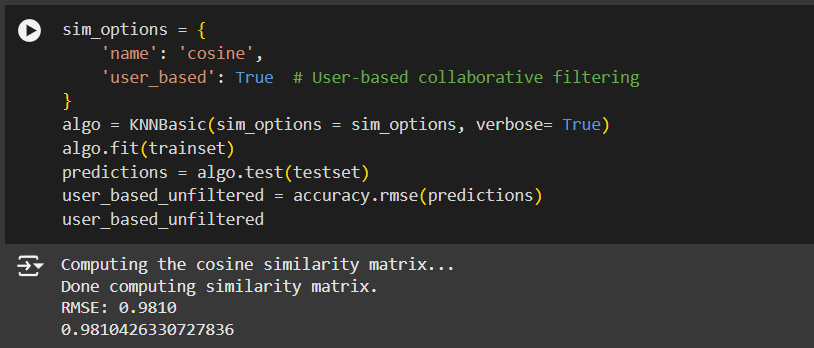


*Fig 2.1.1: Import modules*

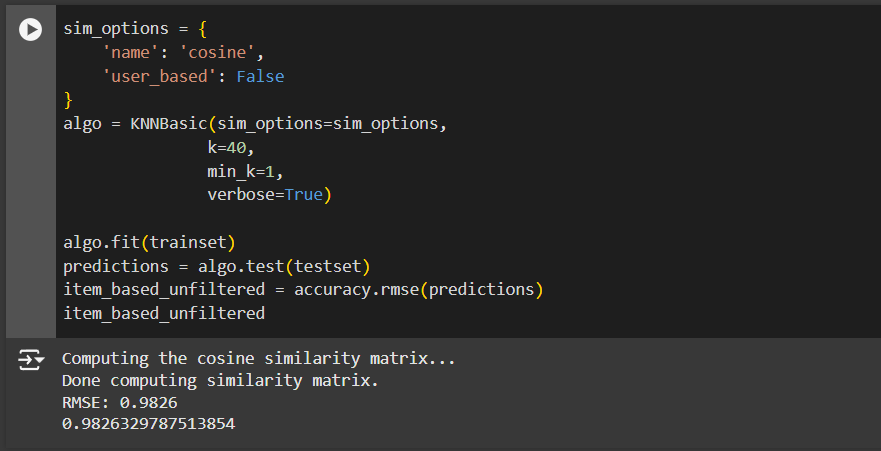
* 1. **Evaluation on the unfiltered dataset:**



*Fig 2.2.1: Initialize Reader and Dataset object and split trainset, testset*

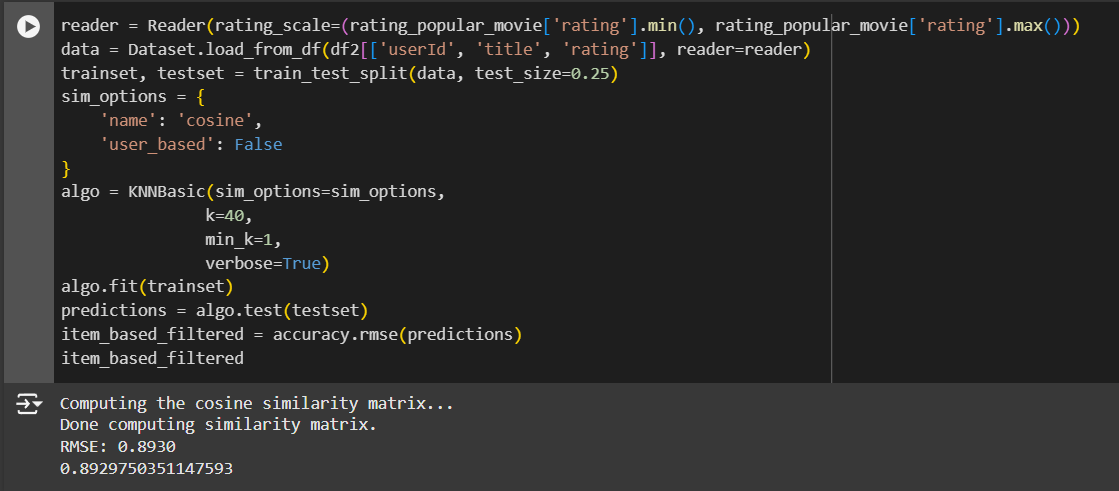


*Fig 2.2.2: User-based KNNBasic with k = 40, min\_k = 1 on unfiltered dataset*

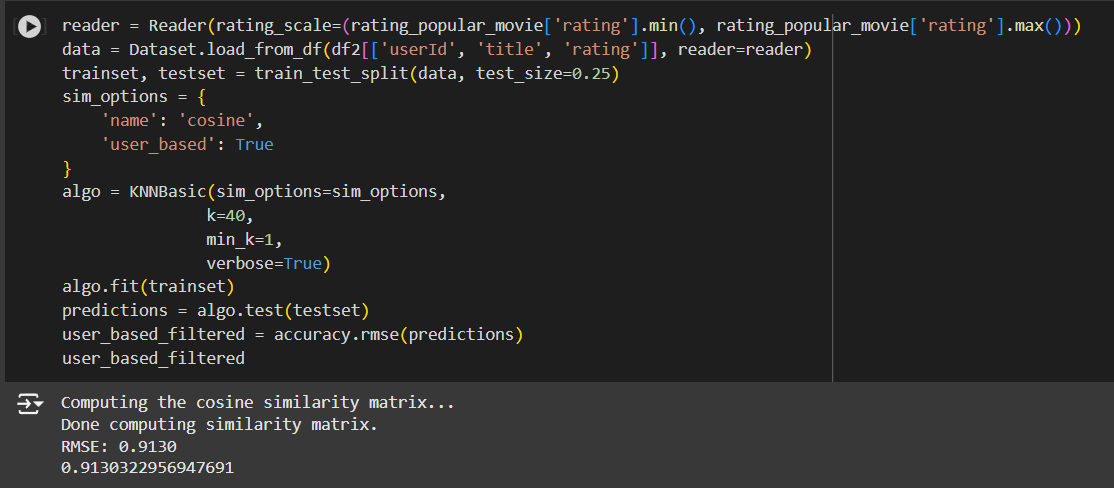


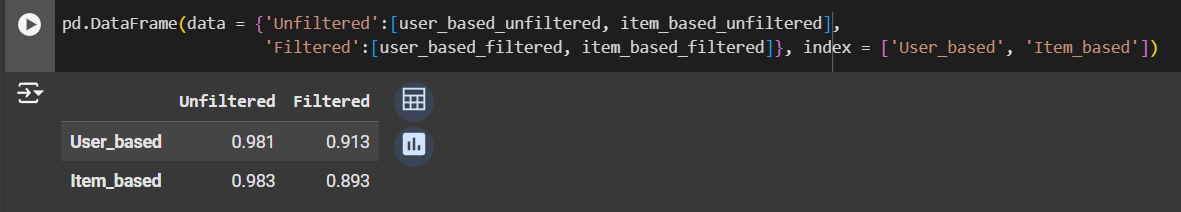
*Fig 2.2.3: Item-based KNNBasic() with k = 40, min\_k = 1 on unfiltered dataset*

* 1. **Evaluation on the filtered dataset:**



*Fig 2.3.1: User-based with k = 40, min\_k = 1 on the filtered dataset*



*Fig 2.3.2: Item-based with k = 40, min\_k = 1 on the filtered dataset*  
  
*Fig 2.3.3: Evaluation between user\_based and item\_based on unfiltered and filtered datasets*

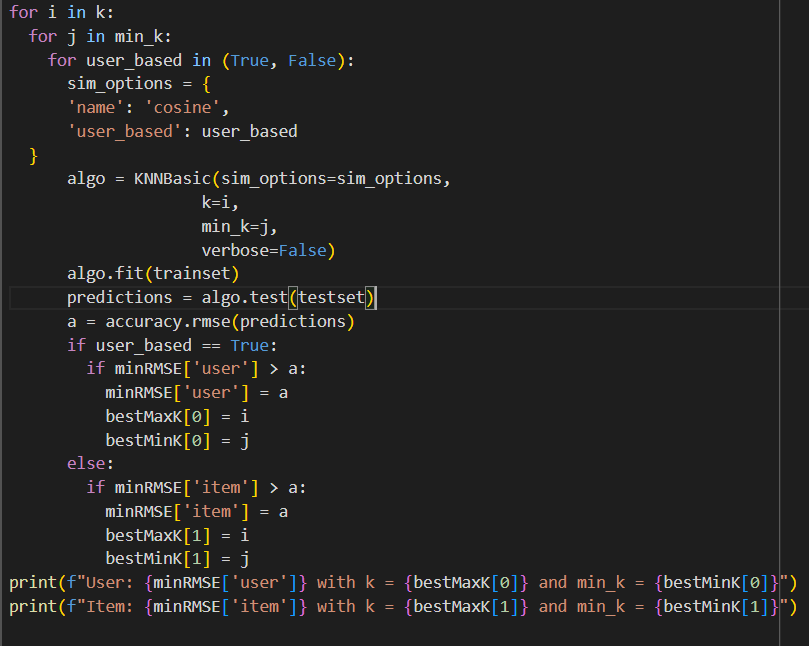
We can see that after filtered, the dataset provides more clarity for the model to work, hence the RMSE decreased by a small portion.

**Fine tuning hyperparameters:**

We iteratively run the model using different k, min\_k to find the best hyperparameters



*Fig 2.3.4: Sets of hyperparameters*



*Fig 2.3.5: Iteratively finding the best combination of hyperparameters*



*Fig 2.3.6: Combinations found*

## **Implementation with numpy, pandas and sklearn**

* 1. **User-based**

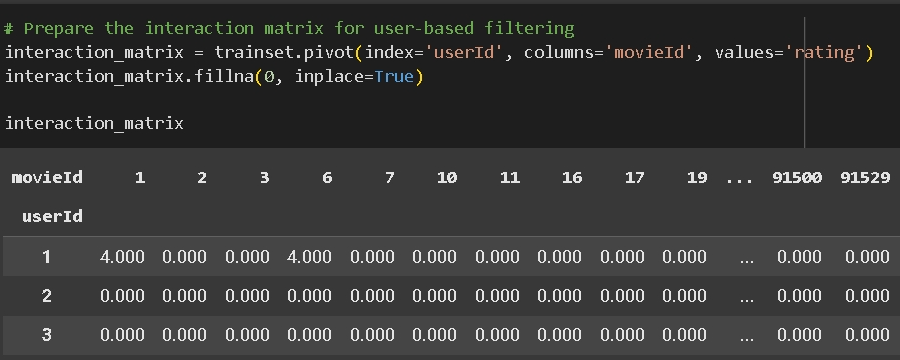
We use the filtered dataset rating\_popular\_movie above where movies that are rated by fewer than 50 users are removed, as they serve as noise. Take only necessary columns, name it df4.

*Fig 3.1.1: Use the filtered dataset*

**

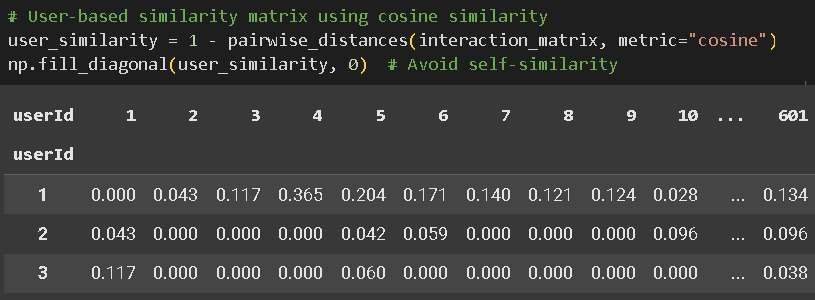
*Fig 3.1.2: Train test split*

* + 1. **Build interaction and similarity matrices**

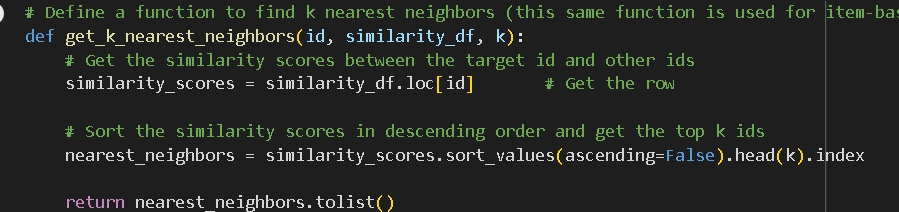
  
*Fig 3.1.1.1: User-Movie interaction matrix*

**User-item interaction matrix:** We use pivot method from pandas dataframe to make userId as index and movieID are columns, the entries are their ratings. We also fill NaN values with 0.

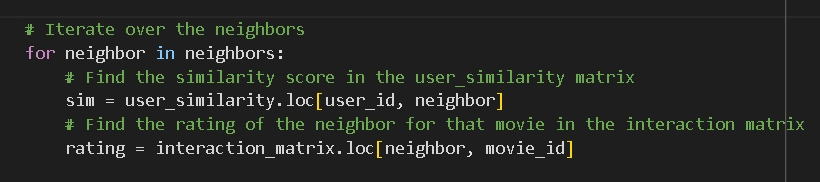
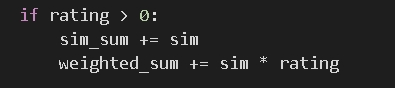
**User similarity matrix:** We use pairwise\_distances from sklearn.metrics, measured by cosine distance. Because we want to get the degree of similarity, we minus the distance from 1. Also, the diagonal entries represent the same user so we set it to 0 to prevent self-similarity.

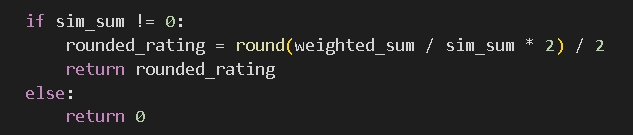
  
*Fig 3.1.1.2: User similarity matrix*

* + 1. **Estimate rating**

**User’s K-neighbor:** We first get the row of the target user from the similarity matrix.  
Then get the index of the largest k entries of that row. The index is the userId that is most similar.  


*Fig 3.1.2.1: Function to find K nearest neighbors*  
We then can use this function to find k most similar users of each user.

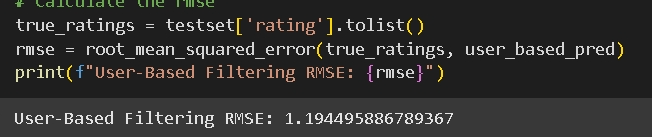
**Rating prediction:** For each neighbor, we get the similarity score, and their rating*Fig 3.1.2.2: Rating prediction user-based*

We then collect the similarity sum for denominator and weighted sum for numerator.*Fig 3.1.2.3: Weighted sum*We then round it to the nearest 0.5 (i,e 0.5, 1, 1.5, 2, 2.5 …) to fit it with the data. ****

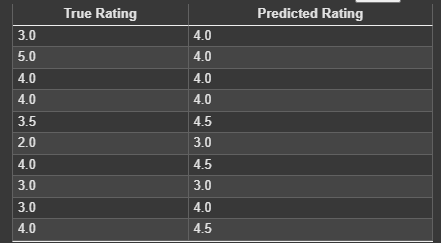
*Fig 3.1.2.4: Rounded prediction*

* + 1. **Evaluation**

Our RMSE is not too high.



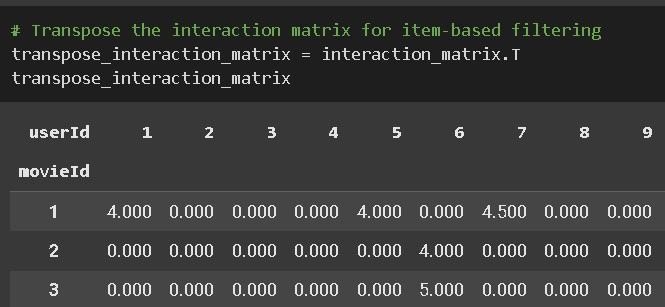
*Fig 3.1.3.1: User-based RMSE*



*Fig 3.1.3.2: Example of predicted rating vs true rating*

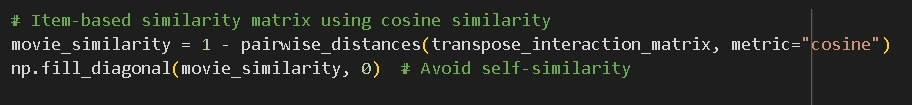
* 1. **Item-based:**
     1. **Item similarity matrix:**

**Item-user interaction matrix:** We need movieID as index and userId as columns. i.e the transpose of our previous User-item interaction matrix:

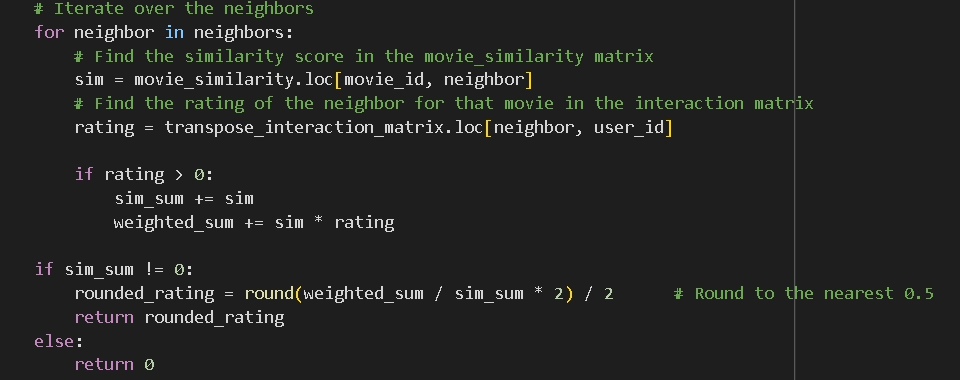


*Fig 3.2.1.1: Item-user interaction matrix*

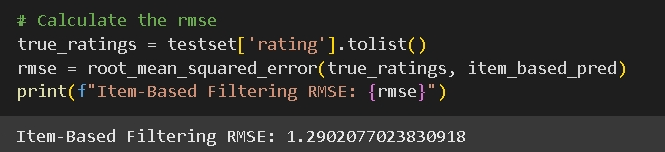
**Item similarity matrix:** Since the matrix does not change, we can apply the same process. We use pairwise\_distances from sklearn.metrics, measured by cosine distance. Because we want to get the degree of similarity, we minus the distance by 1. Also, the diagonal entries represent the same item so we set it to 0 to prevent self-similarity.

  
*Fig 3.2.1.2: Item similarity matrix*

**Estimate rating**  
We first get k-neighbors items that are the most similar to our query item (input).  
Then collect the similarity sum and its weighted sum to calculate the prediction rating.  
We also rounded it to the nearest 0.5 unit.

  
*Fig 3.2.2.1: Estimating rating of item-based*

* + 1. **Evaluation**

We split the whole datasets into test set and train set using sklearn.model\_selection.train\_test\_split. Our RMSE is similar:  


*Fig 3.2.3.1: RMSE error of item-based*

**IV. Summary:**

The model implemented by KNNBasic() and numpy, pandas both perform quite similarly on the same dataset. Given the dataset is filtered with rational conditions, the recommendations made introduce an engaging and intriguing user interaction.

On finishing this mini capstone project, we are capable of implementing a collaborative filtering recommendation system using KNN using high level pre-built libraries and low-level libraries such as numpy and pandas.