FlickPicks: Personalized Movie Recommendations using Collaborative Filtering

A INTERNSHIP PROJECT REPORT

Submitted to

CoApps.ai

Submitted by

Vellanki Gayathri
Sunkara Likhitha
Sarnala Kalyani
Thota Hanish Manikanta

Junior DATA SCIENTIST Intern

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
NRI INSTITUTE OF TECHNOLOGY
(Autonomous)

<u>ABSTRACT:</u>

Today’s web and app users request modified experiences. They anticipate the apps, news sites, social networks they engage with to evoke who they are and what they’re fascinated in and make related, adjusted, and accurate commendations for new content and new goods based on their earlier deeds. This can be done using Recommended Systems in Machine Learning. In this paper we use Recommender System to recommend movies based on his previous ratings on movie he came across.

"FlickPicks: Personalized Movie Recommendations using Collaborative Filtering" is a study focused on enhancing movie recommendation systems through collaborative filtering techniques. Collaborative filtering leverages user behavior data to generate recommendations tailored to individual preferences. This abstract introduces FlickPicks, a system designed to provide personalized movie suggestions based on users' past viewing habits and ratings. By analyzing similarities between users and items within a large dataset, FlickPicks can predict user preferences for movies they have not yet seen. This approach offers users a curated selection of films aligned with their tastes, enhancing their viewing experience and engagement with the platform. Through rigorous evaluation and experimentation, FlickPicks demonstrates its effectiveness in generating accurate and relevant movie recommendations, thereby contributing to the advancement of personalized content recommendation systems in the entertainment domain.

INDEX

Chapter-1: Introduction:

A Recommendation System is a software tool designed to make and deliver suggestions for things or content a user would like to purchase. Using machine learning techniques and various data about individual products and individual users, the system creates an advanced net of complex connections between those products and those people. These are a collection of algorithms used to recommend items to users based on information taken from the user. These systems have become ubiquitous, and can be commonly seen in online stores, movies databases and job finders. There are 3 types of recommendation systems

1. Popularity based recommendation engine
2. Content based recommendation engine
3. Collaborative filtering based recommendation engine

Popularity based recommendation engine: Pearson correlation is invariant to scaling, i.e. multiplying all elements by a nonzero constant or adding any constant to all elements. For example, if you have two vectors X and Y, then, pearson (X, Y) == pearson(X, 2 * Y + 3). This is a pretty important property in recommendation systems because for example two users might rate two series of items totally different in terms of absolute rates, but they would be similar users (i.e. with similar ideas) with similar rates in various scales.

Content based recommendation engine: Content based recommendation engine takes in a movie that a user currently likes as input. Then it analyzes the contents of the movie to find out other movies which have similar content. Then it ranks similar movies according to their similarity scores and recommends the most relevant movies to the user.

Collaborative filtering based recommendation engine: Collaborative filtering based recommendation engine first tries to find similar users based on their activities and preferences. Now, between these users (say, A and B) if user A has seen a movie that user B has not seen yet, then that movie gets recommended to user B and vice-versa. In other words, the recommend-dations get filtered based on the collaboration between similar user’s preferences. One typical application of this algorithm can be seen in the Amazon e-commerce platform, where you get to see the “Customers who viewed this item also viewed” and “Customers who bought this item also bought” list.

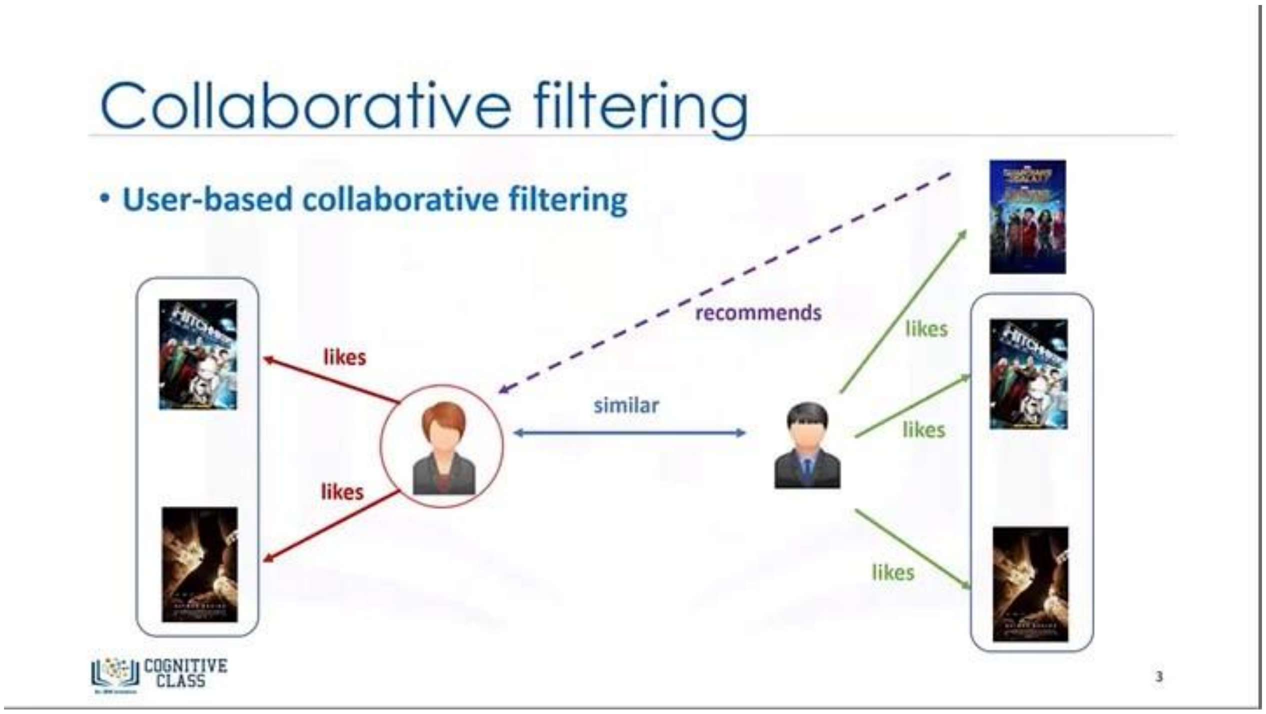
There are two types in collaborative filtering:

- User-based collaborative filtering
- Item-based collaborative filtering

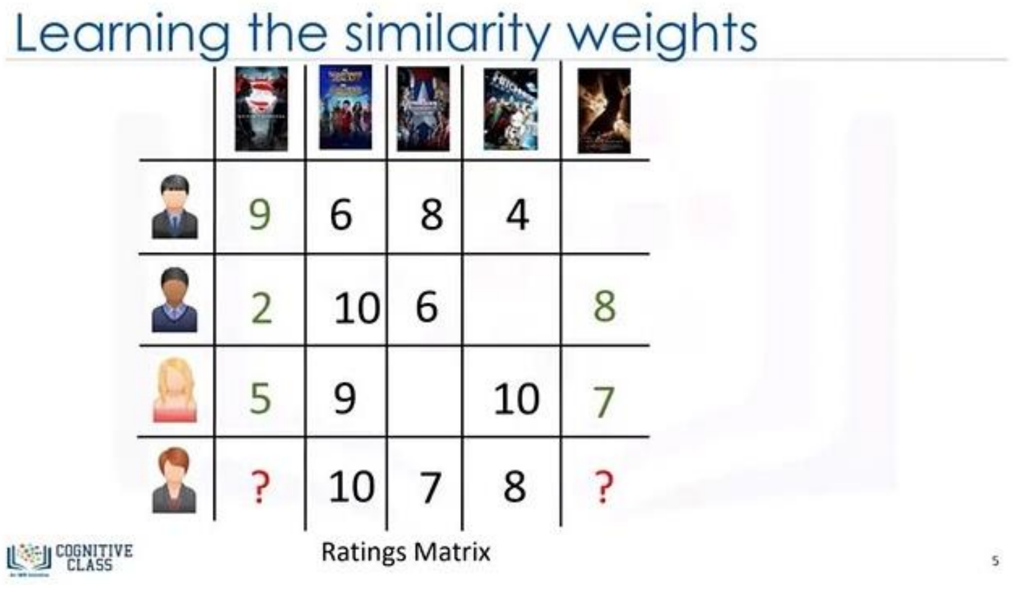
Chapter-2: Methodology:

User-Based collaborative filtering:

In user-based collaborative filtering, we have an active user for whom the recommendation is aimed. The collaborative filtering engine first looks for users who are similar. That is users who share the active users rating patterns. Collaborative filtering basis this similarity on things like history, preference, and choices that users make when buying, watching, or enjoying something.



For example, movies that similar users have rated highly. Then it uses the ratings from these similar users to predict the possible ratings by the active user for a movie that she had not previously watched. For instance, if two users are similar or are neighbors in terms of their interested movies, we can recommend a movie to the active user that her neighbor has already seen. Now, let’s dive into the algorithm to see how all of this works.



Learning the similarity weights

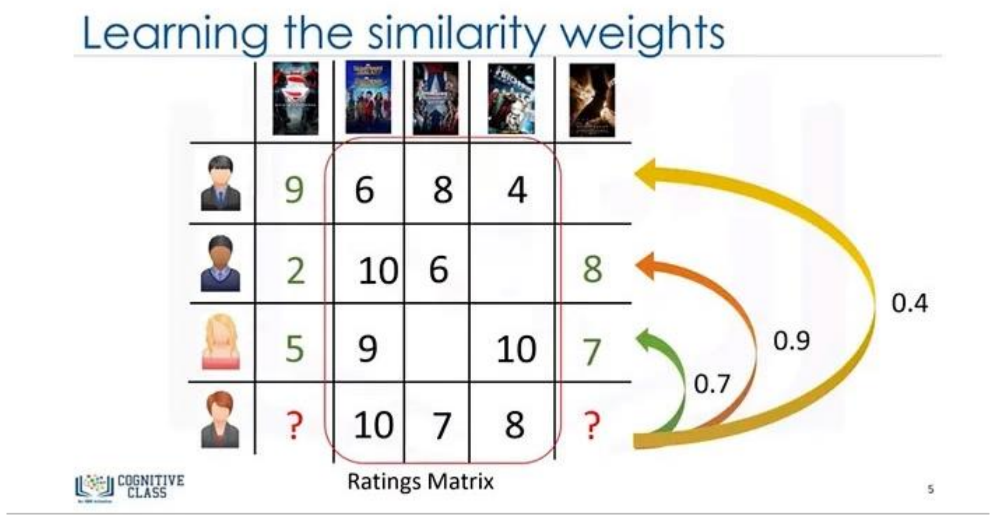
	9	6	8	4	
	2	10	6		8
	5	9		10	7
	?	10	7	8	?

Ratings Matrix

COGNITIVE CLASS

5

Assume that we have a simple user item matrix, which shows the ratings of four users for five different movies. Let’s also assume that our active user has watched and rated three out of these five movies. Let’s find out which of the two movies that our active user hasn’t watched should be recommended to her. The first step is to discover how similar the active user is to the other users. How do we do this? Well, this can be done through several different statistical and vectorial techniques such as distance or similarity measurements including Euclidean Distance, Pearson Correlation, Cosine Similarity, and so on.



Learning the similarity weights

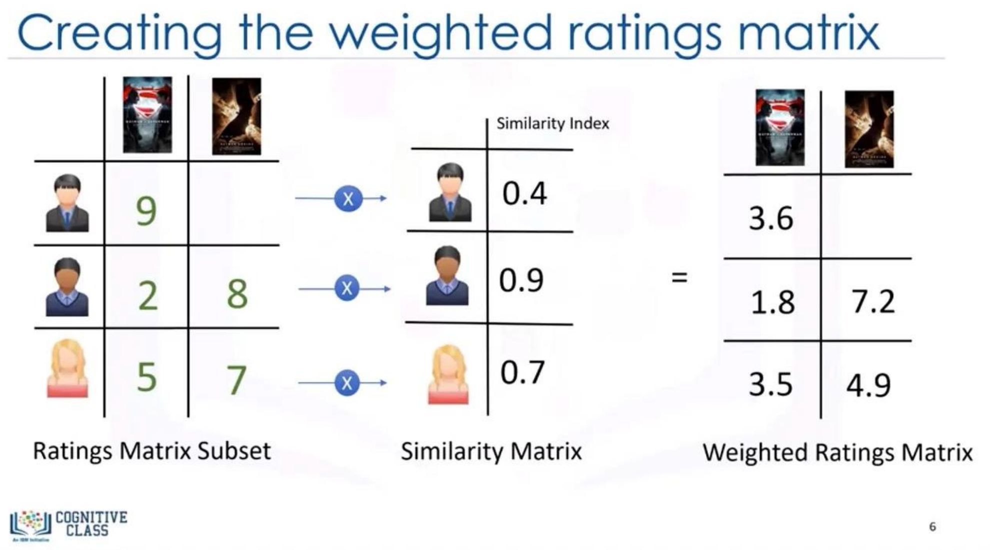
	9	6	8	4	
	2	10	6		8
	5	9		10	7
	?	10	7	8	?

Ratings Matrix

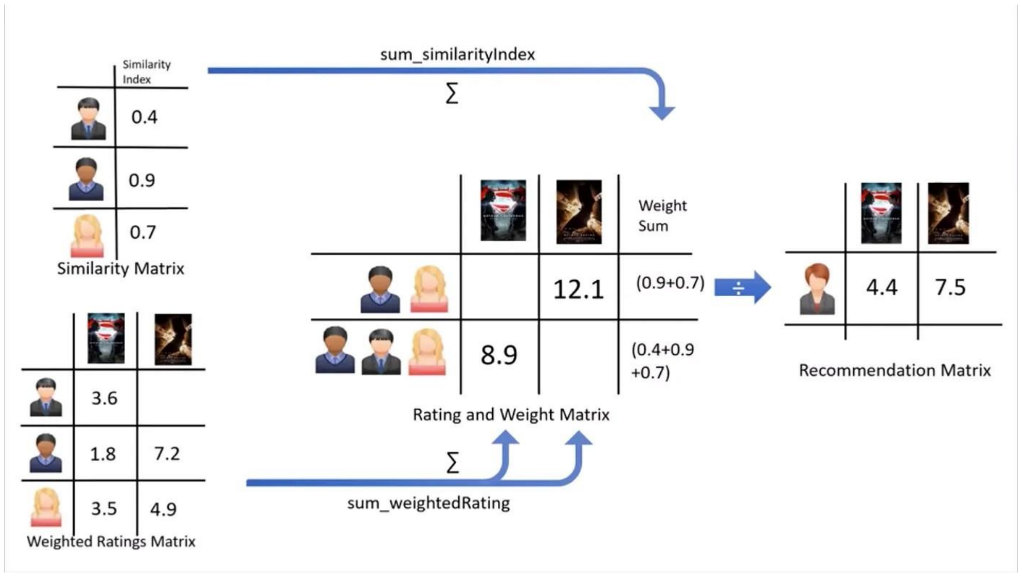
COGNITIVE CLASS

5

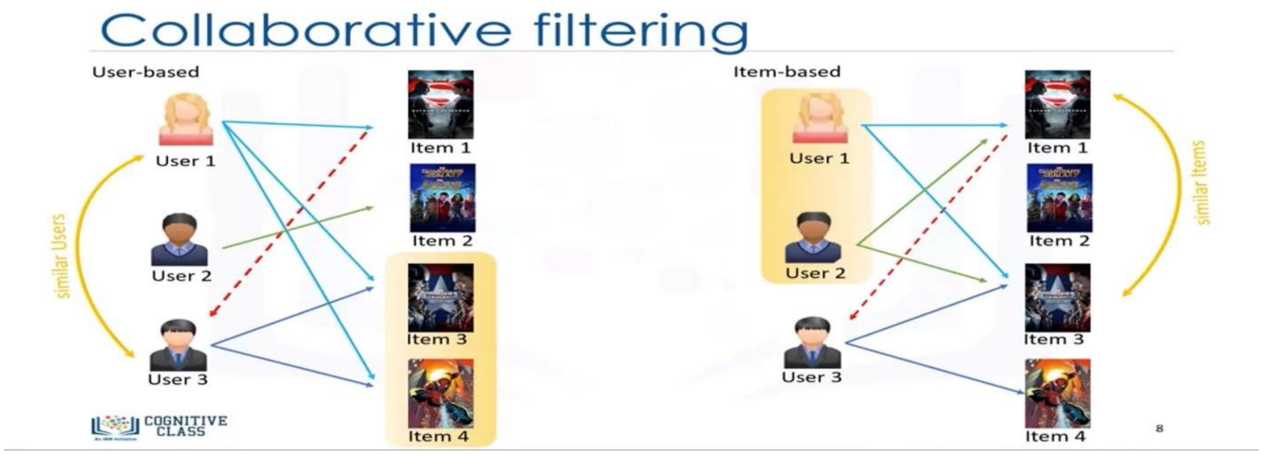
To calculate the level of similarity between two users, we use the three movies that both the users have rated in the past. Regardless of what we use for similarity measurement, let’s say for example, the similarity could be 0.7, 0.9, and 0.4 between the active user and other users. These numbers represent similarity weights or proximity of the active user to other users in the dataset.



The next step is to create a weighted rating matrix. We just calculated the similarity of users to our active user in the previous slide. Now, we can use it to calculate the possible opinion of the active user about our two target movies. This is achieved by multiplying the similarity weights to the user ratings. It results in a weighted ratings matrix, which represents the user’s neighbors opinion about are two candidate movies for recommendation. In fact, it incorporates the behavior of other users and gives more weight to the ratings of those users who are more similar to the active user.



Now, we can generate the recommendation matrix by aggregating all of the weighted rates. However, as three users rated the first potential movie and two users rated the second movie, we have to normalize the weighted rating values. We do this by dividing it by the sum of the similarity index for users. The result is the potential rating that our active user will give to these movies based on her similarity to other users. It is obvious that we can use it to rank the movies for providing recommendation to our active user.



Chapter-3: System Requirements:

Processor: Intel Core i5 or higher (Or equivalent AMD processor)

Ram: Minimum 8 GB of RAM

Operating System: windows

Development Environment:

- Jupyter Notebook, Visual studio code

Software Requirements:

- Python

Necessary Packages:

- matplotlib, seaborn: Plotting and visualization.
- numpy: Numerical operations on arrays.
- pandas: Data manipulation and analysis.
- sklearn: Machine learning tools.

3.1 Proposed Model:

User-user collaborative filtering is a technique used in recommendation systems to provide personalized recommendations to users based on their similarity to other users. In this approach, recommendations are generated by identifying users who have similar preferences or behaviors and then suggesting items that those similar users have liked or interacted with.

The process typically involves the following steps:

User-item matrix: Create a matrix where rows represent users and columns represent items. Each cell in the matrix contains a rating or some indication of the user's interaction with the item (e.g., purchase history, ratings, likes).

Similarity calculation: Compute the similarity between users based on their interactions with items. Common similarity measures include cosine similarity, Pearson correlation, or Jaccard similarity.

Neighborhood selection: Identify a subset of similar users (neighborhood) for each target user. This subset is often determined based on a threshold similarity score or a fixed number of nearest neighbors.

Recommendation generation: Once the neighborhood is established, recommend items to the target user based on the items liked or interacted with by users in the neighborhood. Items that are popular among similar users but have not been interacted with by the target user are recommended.

Aggregation and ranking: Aggregate recommendations from multiple similar users and rank them based on some criteria (e.g., predicted rating, popularity) to present the top recommendations to the target user.

3.2 Modules:

pandas (pd):

Used for data manipulation and analysis, providing data structures like DataFrame to work with tabular data efficiently.

numpy (np):

Used for numerical computing, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.

scipy.stats:

Specifically, the pearson function from the scipy.stats module is used to calculate Pearson correlation coefficients between users.

seaborn (sns):

Used for data visualization, providing high-level interface for drawing attractive and informative statistical graphics.

sklearn.metrics.pairwise.cosine_similarity:

Used to compute cosine similarity between vectors, which is used in calculating similarity between users based on their preferences.

Chapter-4: IMPLEMENTATION:

Import Python Libraries:

In the first step, we will import Python libraries `pandas`, `numpy`, and `scipy.stats`. These three libraries are for data processing and calculations. We also imported `seaborn` for visualization and `cosine_similarity` for calculating similarity score.

Data Collection:

Collecting the dataset required to perform the recommendations.

Data Preparation:

We've read the ratings and movies data from CSV files into pandas DataFrames.

Data Exploration:

We've merged the ratings and movies datasets based on the 'movieId' column to create a single DataFrame.

Checked the popular movies by aggregating mean ratings and the number of ratings for each movie.

Exploratory Data Analysis (EDA):

In step 3, we need to filter the movies and keep only those with over 100 ratings for the analysis. To do that, we first group the movies by title, count the number of ratings, and keep only the movies with greater than 100 ratings. The average ratings for the movies are calculated as well. From the `.info()` output, we can see that there are 134 movies left.

Data Visualization:

Next, let's use a `jointplot` to check the correlation between the average rating and the number of ratings. We can see an upward trend from the scatter plot, showing that popular movies get higher ratings. The average rating distribution shows that most movies in the dataset have an average rating of around 4.The number of rating distribution shows that most movies have less than 150 ratings.

Create User-Movie Matrix:

In step 4, we will transform the dataset into a matrix format. The rows of the matrix are users, and the columns of the matrix are movies. The value of the matrix is the user rating of the movie if there is a rating. Otherwise, it shows 'NaN'.

Data Normalization:

Since some people tend to give a higher rating than others, we normalize the rating by extracting the average rating of each user.After normalization, the movies with a rating less than the user's average rating get a negative value, and the movies with a rating more than the user's average rating get a positive value.

Identify Similar Users:

There are different ways to measure similarities. Pearson correlation and cosine similarity are two widely used methods. we will calculate the user similarity matrix using Cosine Similarity.

Narrow Down Item Pool:

we will narrow down the item pool by doing the following:

1. Remove the movies that have been watched by the target user (user ID).
2. Keep only the movies that similar users have watched.

To keep only the similar users' movies, we keep the user IDs in the top 10 similar user lists and remove the film with all missing values. All missing value for a movie means that none of the similar users have watched the movie.

Recommend Items:

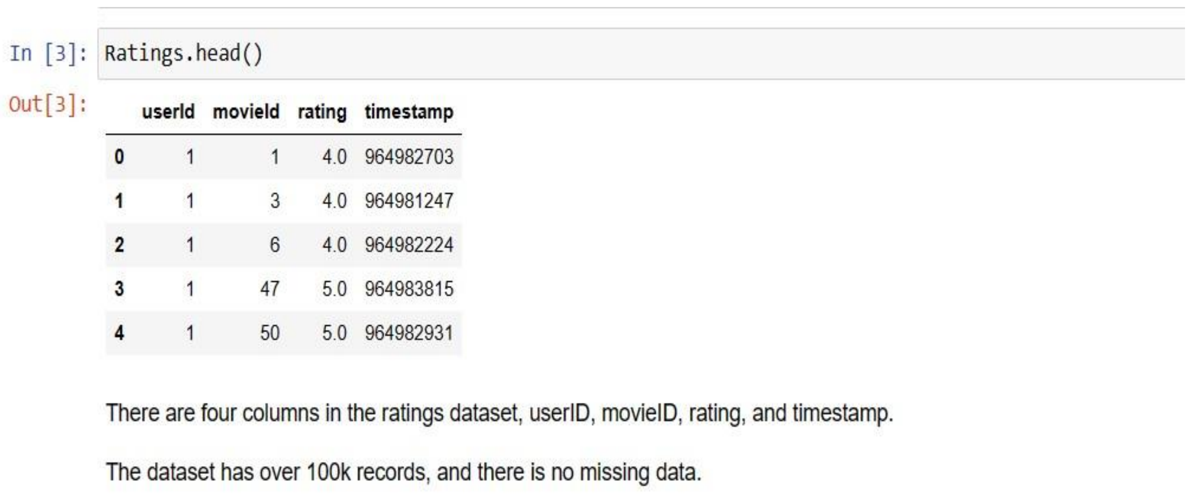
we will decide which movie to recommend to the target user. The recommended items are determined by the weighted average of user similarity score and movie rating. The movie ratings are weighted by the similarity scores, so the users with higher similarity get higher weights.

Chapter-5: Code:

```
import pandas as pd
import numpy as np
import scipy.stats
import seaborn as sns
# Similarity
from sklearn.metrics.pairwise import cosine_similarity
# Read in data
ratings=pd.read_csv(r'ratings.csv')
# Read in data
movies = pd.read_csv(r'movies.csv')

# Take a look at the data
movies.head()

# Merge ratings and movies datasets
df = pd.merge(ratings, movies, on='movieId', how='inner')
# Take a look at the data
df.head()
agg_ratings = df.groupby('title').agg(mean_rating = ('rating',
'mean'),number_of_ratings = ('rating', 'count')).reset_index()
agg_ratings_GT100 = agg_ratings[agg_ratings['number_of_ratings']>100]

# Check popular movies
popular=agg_ratings_GT100.sort_values(by='number_of_ratings',
ascending=False).head()
# Visulization
fig=sns.jointplot(x='mean_rating', y='number_of_ratings', data=agg_ratings_GT100)
df_GT100 = pd.merge(df, agg_ratings_GT100[['title']], on='title', how='inner')
rate=sorted(df_GT100['rating'].unique())
matrix = df_GT100.pivot_table(index='userId', columns='title', values='rating')
matrix_norm = matrix.subtract(matrix.mean(axis=1), axis = 'rows')
user_similarity_cosine = cosine_similarity(matrix_norm.fillna(0))
n = 10
user_similarity_threshold = 0.3
picked_userid=int(input("Enter userID:"))

similar_users =
user_similarity[user_similarity[picked_userid]>user_similarity_threshold][picked_
userid].sort_values(ascending=False)[:n]
picked_userid_watched = matrix_norm[matrix_norm.index ==
picked_userid].dropna(axis=1, how='all')
```

```
similar_user_movies =
matrix_norm[matrix_norm.index.isin(similar_users.index)].dropna(axis=1,
how='all')
similar_user_movies.drop(picked_userid_watched.columns,axis=1, inplace=True,
errors='ignore')
item_score = {}
for i in similar_user_movies.columns:
  movie_rating = similar_user_movies[i]
  total = 0
  count = 0
  for u in similar_users.index:
    if pd.isna(movie_rating[u]) == False:
      score = similar_users[u] * movie_rating[u]
      total += score
      count +=1
  item_score[i] = total / count

item_score = pd.DataFrame(item_score.items(), columns=['movie', 'movie_score'])

ranked_item_score = item_score.sort_values(by='movie_score', ascending=False)
m = int(input("Enter Number of Recommendatios:"))
print(ranked_item_score.head(m))
```

5.1 SCREENSHOTS:

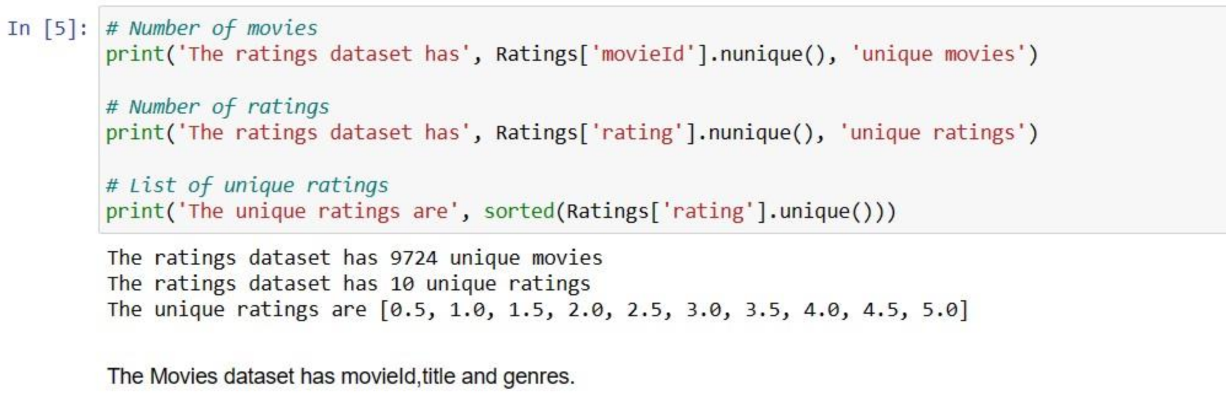


```
In [3]: Ratings.head()
Out[3]:
```

	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

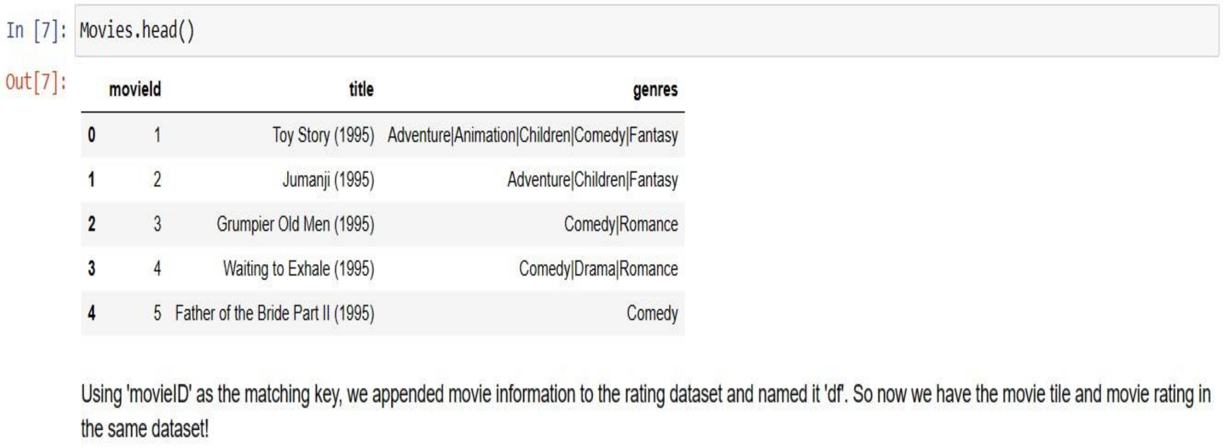
There are four columns in the ratings dataset, userID, movieID, rating, and timestamp.

The dataset has over 100k records, and there is no missing data.



```
In [5]: # Number of movies
        print('The ratings dataset has', Ratings['movieId'].nunique(), 'unique movies')

        # Number of ratings
        print('The ratings dataset has', Ratings['rating'].nunique(), 'unique ratings')

        # List of unique ratings
        print('The unique ratings are', sorted(Ratings['rating'].unique()))

        The ratings dataset has 9724 unique movies
        The ratings dataset has 10 unique ratings
        The unique ratings are [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
```

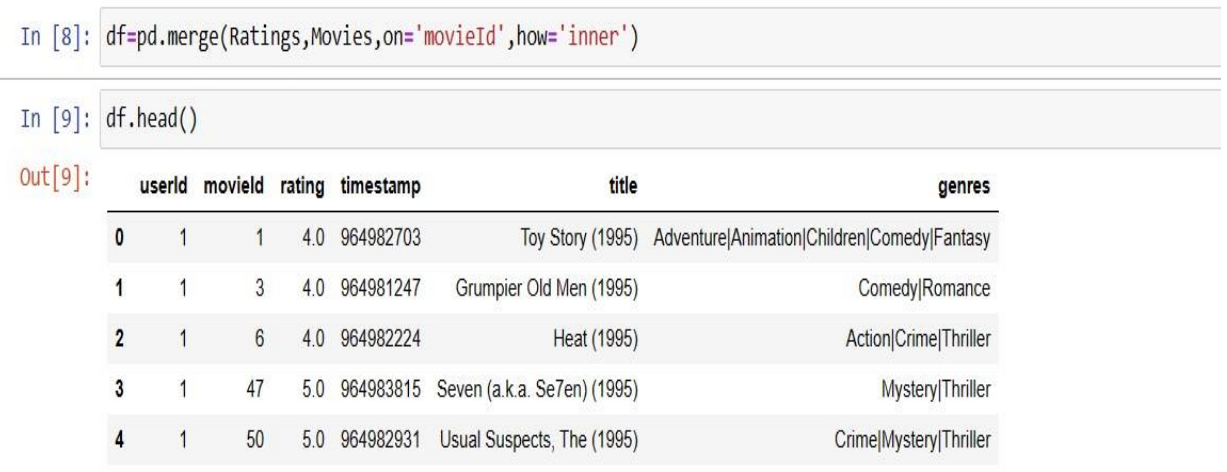
The Movies dataset has movieId,title and genres.



```
In [7]: Movies.head()
Out[7]:
```

	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

Using 'movieID' as the matching key, we appended movie information to the rating dataset and named it 'df'. So now we have the movie tile and movie rating in the same dataset!

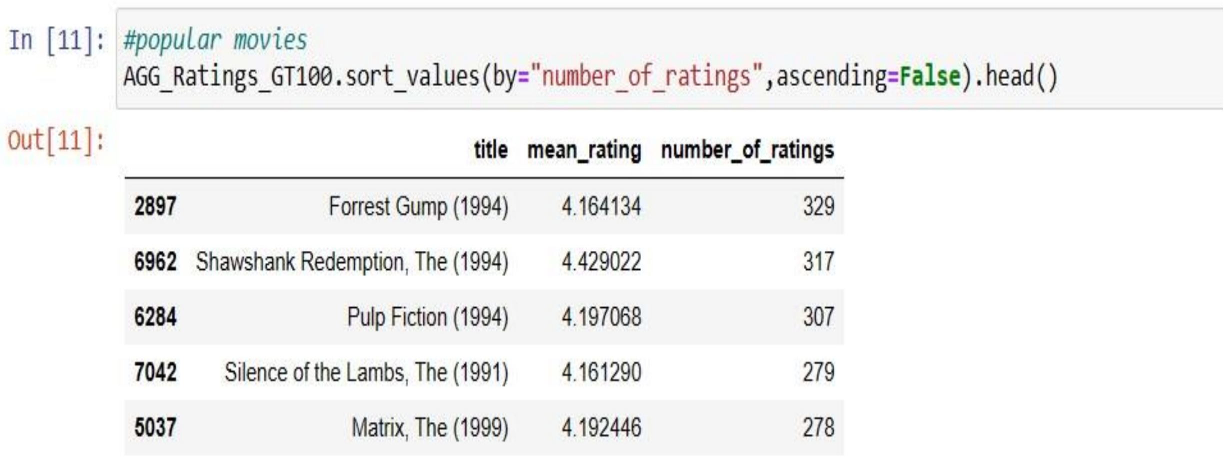


```
In [8]: df=pd.merge(Ratings,Movies,on='movieId',how='inner')
```

```
In [9]: df.head()
```

Out[9]:

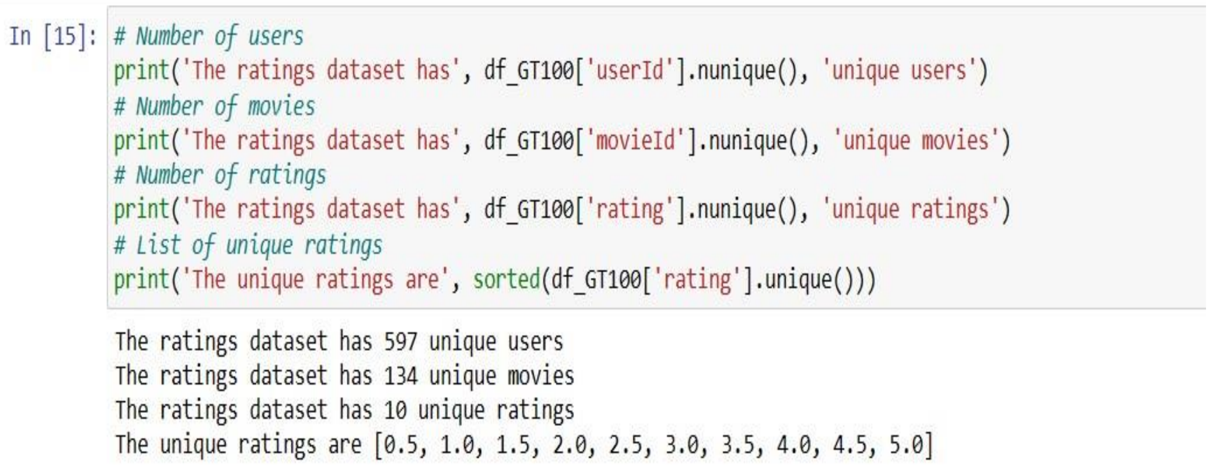
	userId	movieId	rating	timestamp	title	genres
0	1	1	4.0	964982703	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	1	3	4.0	964981247	Grumpier Old Men (1995)	Comedy|Romance
2	1	6	4.0	964982224	Heat (1995)	Action|Crime|Thriller
3	1	47	5.0	964983815	Seven (a.k.a. Se7en) (1995)	Mystery|Thriller
4	1	50	5.0	964982931	Usual Suspects, The (1995)	Crime|Mystery|Thriller



```
In [11]: #popular movies
         AGG_Ratings_GT100.sort_values(by="number_of_ratings",ascending=False).head()
```

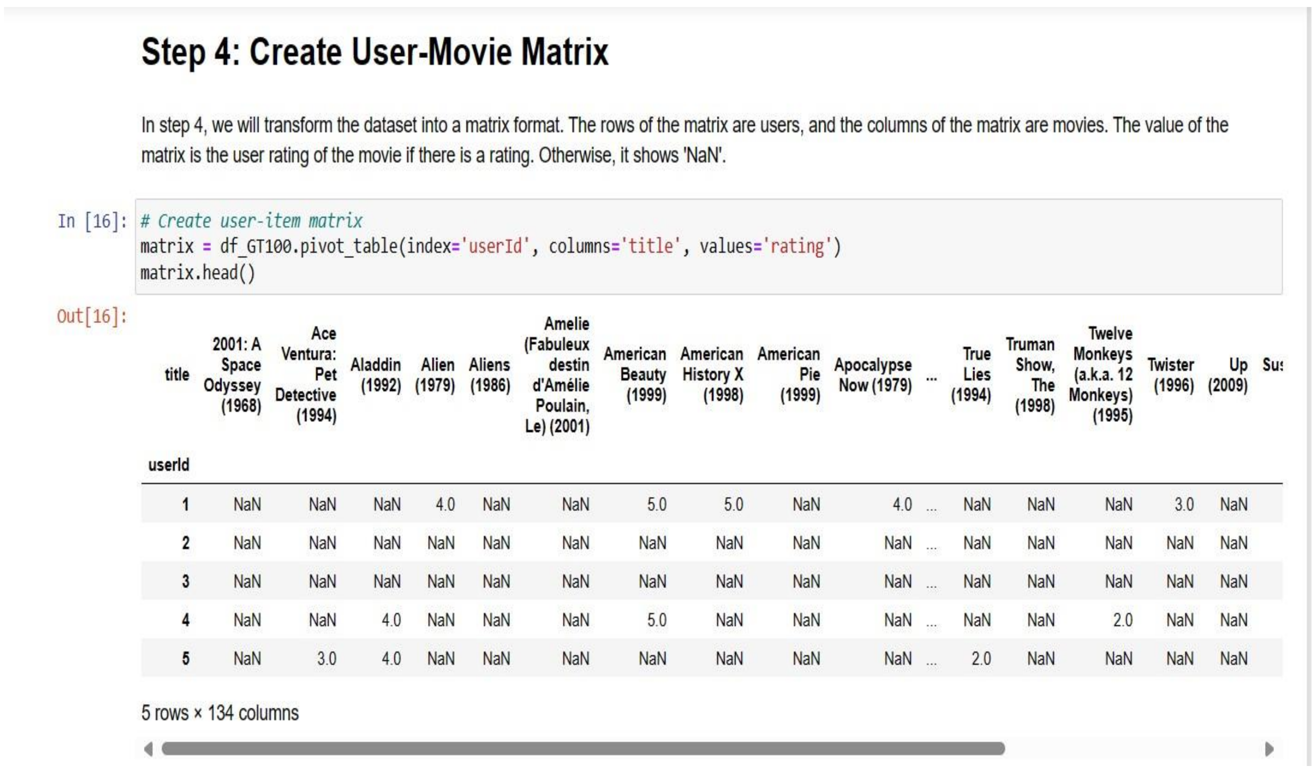
Out[11]:

	title	mean_rating	number_of_ratings
2897	Forrest Gump (1994)	4.164134	329
6962	Shawshank Redemption, The (1994)	4.429022	317
6284	Pulp Fiction (1994)	4.197068	307
7042	Silence of the Lambs, The (1991)	4.161290	279
5037	Matrix, The (1999)	4.192446	278



```
In [15]: # Number of users
         print('The ratings dataset has', df_GT100['userId'].nunique(), 'unique users')
         # Number of movies
         print('The ratings dataset has', df_GT100['movieId'].nunique(), 'unique movies')
         # Number of ratings
         print('The ratings dataset has', df_GT100['rating'].nunique(), 'unique ratings')
         # List of unique ratings
         print('The unique ratings are', sorted(df_GT100['rating'].unique()))
```

```
The ratings dataset has 597 unique users
The ratings dataset has 134 unique movies
The ratings dataset has 10 unique ratings
The unique ratings are [0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
```



Step 4: Create User-Movie Matrix

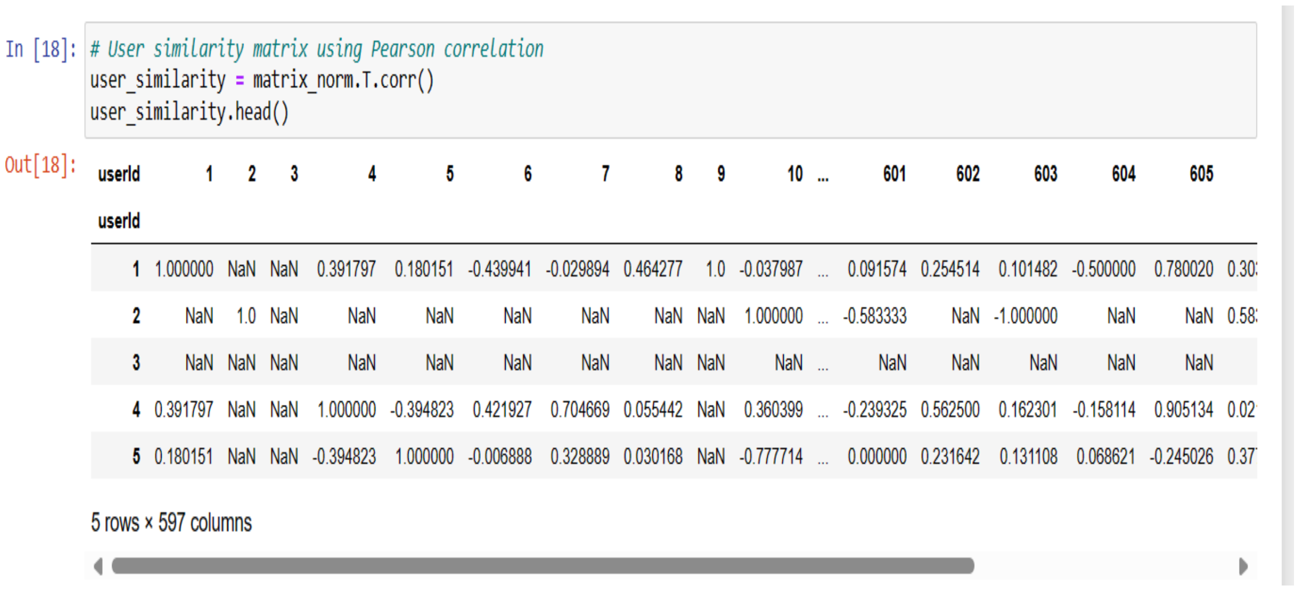
In step 4, we will transform the dataset into a matrix format. The rows of the matrix are users, and the columns of the matrix are movies. The value of the matrix is the user rating of the movie if there is a rating. Otherwise, it shows 'NaN'.

In [16]:
```
# Create user-item matrix
matrix = df_GT100.pivot_table(index='userId', columns='title', values='rating')
matrix.head()
```

Out[16]:

title	2001: A Space Odyssey (1968)	Ace Ventura: Pet Detective (1994)	Aladdin (1992)	Alien (1979)	Aliens (1986)	Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)	American Beauty (1999)	American History X (1998)	American Pie (1999)	Apocalypse Now (1979)	...	True Lies (1994)	Truman Show, The (1998)	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	Twister (1996)	Up (2009)	Su
userId																	
1	NaN	NaN	NaN	4.0	NaN	NaN	5.0	5.0	NaN	4.0	...	NaN	NaN	NaN	3.0	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	4.0	NaN	NaN	NaN	5.0	NaN	NaN	NaN	...	NaN	NaN	2.0	NaN	NaN	
5	NaN	3.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	2.0	NaN	NaN	NaN	NaN	

5 rows × 134 columns

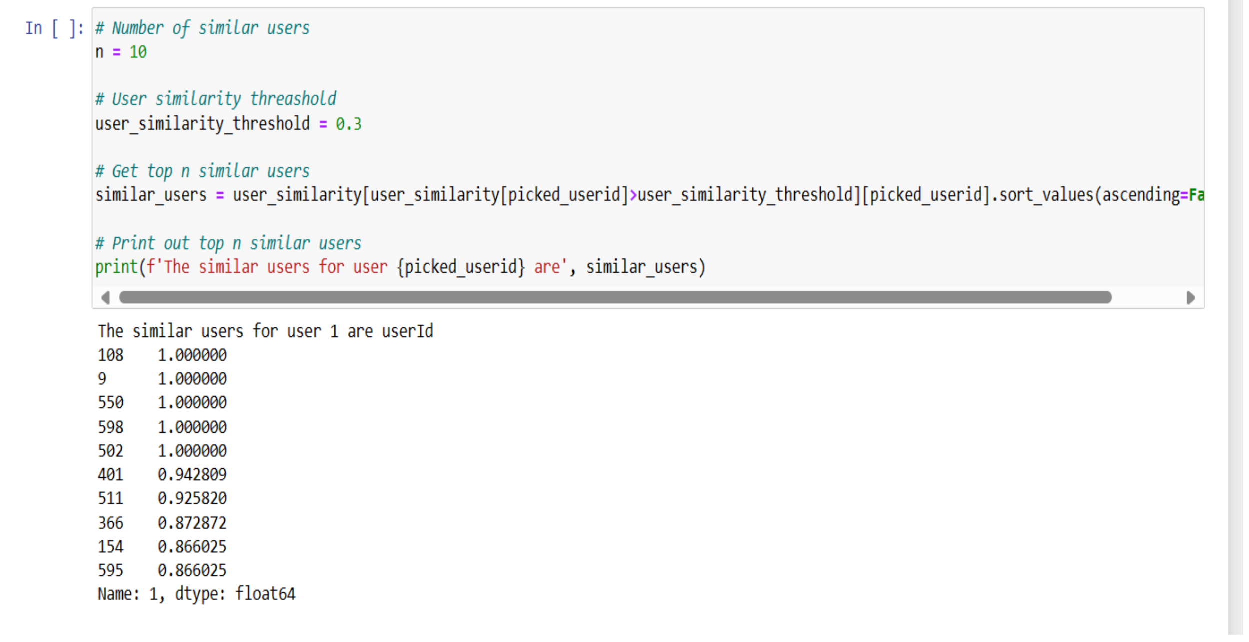


In [18]:
```
# User similarity matrix using Pearson correlation
user_similarity = matrix_norm.T.corr()
user_similarity.head()
```

Out[18]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	
userId																	
1	1.000000	NaN	NaN	0.391797	0.180151	-0.439941	-0.029894	0.464277	1.0	-0.037987	...	0.091574	0.254514	0.101482	-0.500000	0.780020	0.30
2	NaN	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1.000000	...	-0.583333	NaN	-1.000000	NaN	NaN	0.58
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	
4	0.391797	NaN	NaN	1.000000	-0.394823	0.421927	0.704669	0.055442	NaN	0.360399	...	-0.239325	0.562500	0.162301	-0.158114	0.905134	0.02
5	0.180151	NaN	NaN	-0.394823	1.000000	-0.006888	0.328889	0.030168	NaN	-0.777714	...	0.000000	0.231642	0.131108	0.068621	-0.245026	0.37

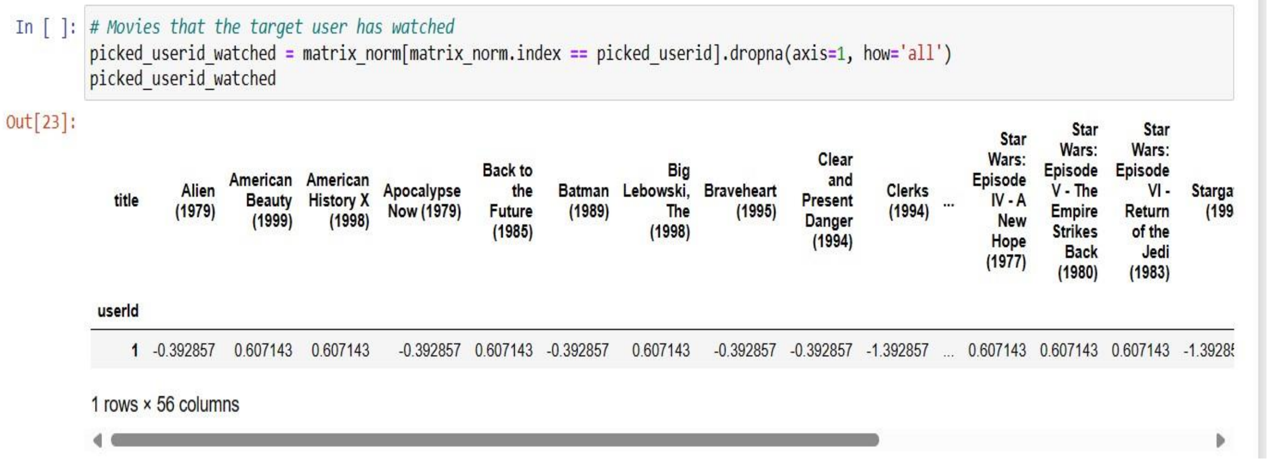
5 rows × 597 columns



```
In [ ]: # Number of similar users
        n = 10

        # User similarity threashold
        user_similarity_threshold = 0.3

        # Get top n similar users
        similar_users = user_similarity[user_similarity[picked_userid]>user_similarity_threshold][picked_userid].sort_values(ascending=Fa

        # Print out top n similar users
        print(f'The similar users for user {picked_userid} are', similar_users)
```

```
The similar users for user 1 are userId
108    1.000000
9      1.000000
550    1.000000
598    1.000000
502    1.000000
401    0.942809
511    0.925820
366    0.872872
154    0.866025
595    0.866025
Name: 1, dtype: float64
```

Step 7: Narrow Down Item Pool

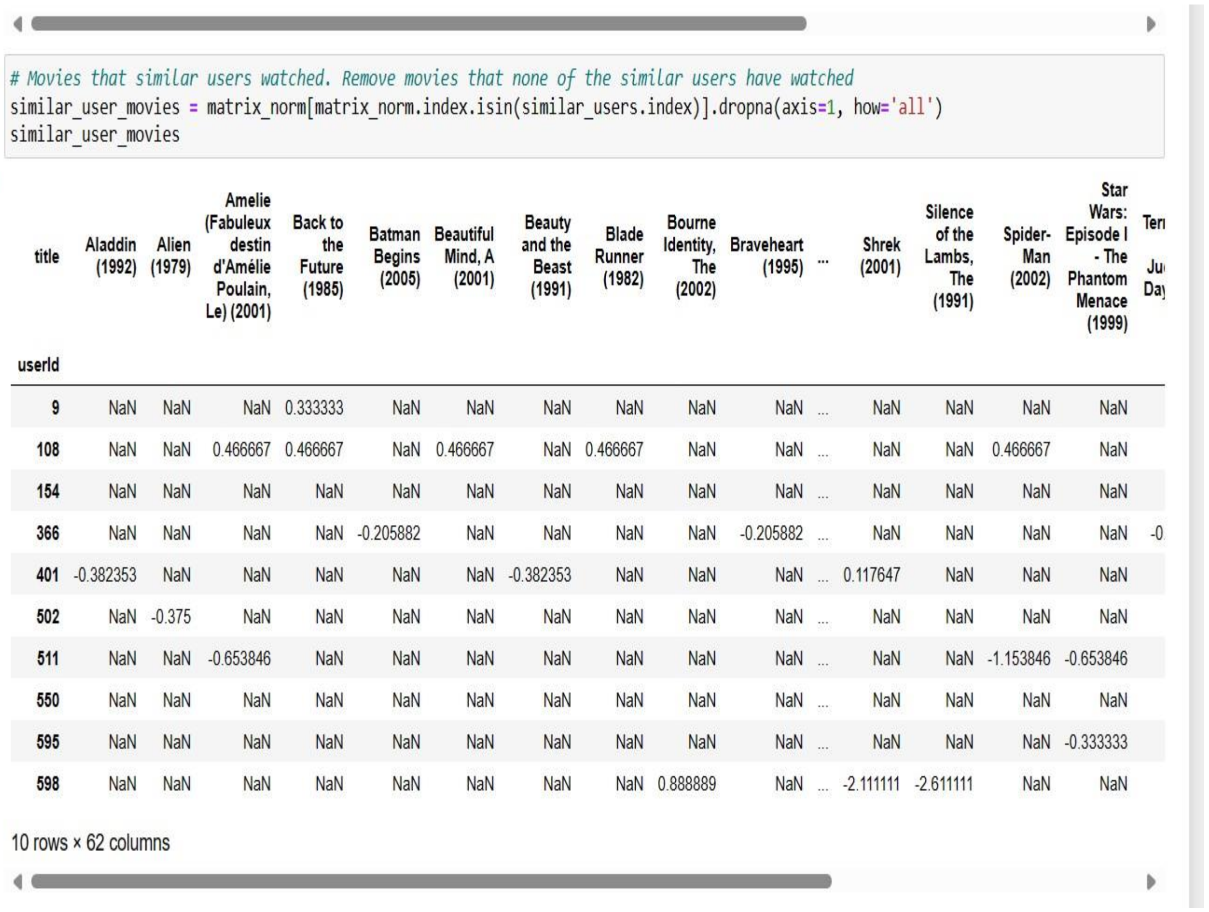


```
In [ ]: # Movies that the target user has watched
        picked_userid_watched = matrix_norm[matrix_norm.index == picked_userid].dropna(axis=1, how='all')
        picked_userid_watched
```

Out[23]:

title	Alien (1979)	American Beauty (1999)	American History X (1998)	Apocalypse Now (1979)	Back to the Future (1985)	Batman (1989)	Big Lebowski, The (1998)	Braveheart (1995)	Clear and Present Danger (1994)	Clerks (1994)	...	Star Wars: Episode IV - A New Hope (1977)	Star Wars: Episode V - The Empire Strikes Back (1980)	Star Wars: Episode VI - Return of the Jedi (1983)	Starga (199
userId															
1	-0.392857	0.607143	0.607143	-0.392857	0.607143	-0.392857	0.607143	-0.392857	-0.392857	-1.392857	...	0.607143	0.607143	0.607143	-1.39285

1 rows × 56 columns

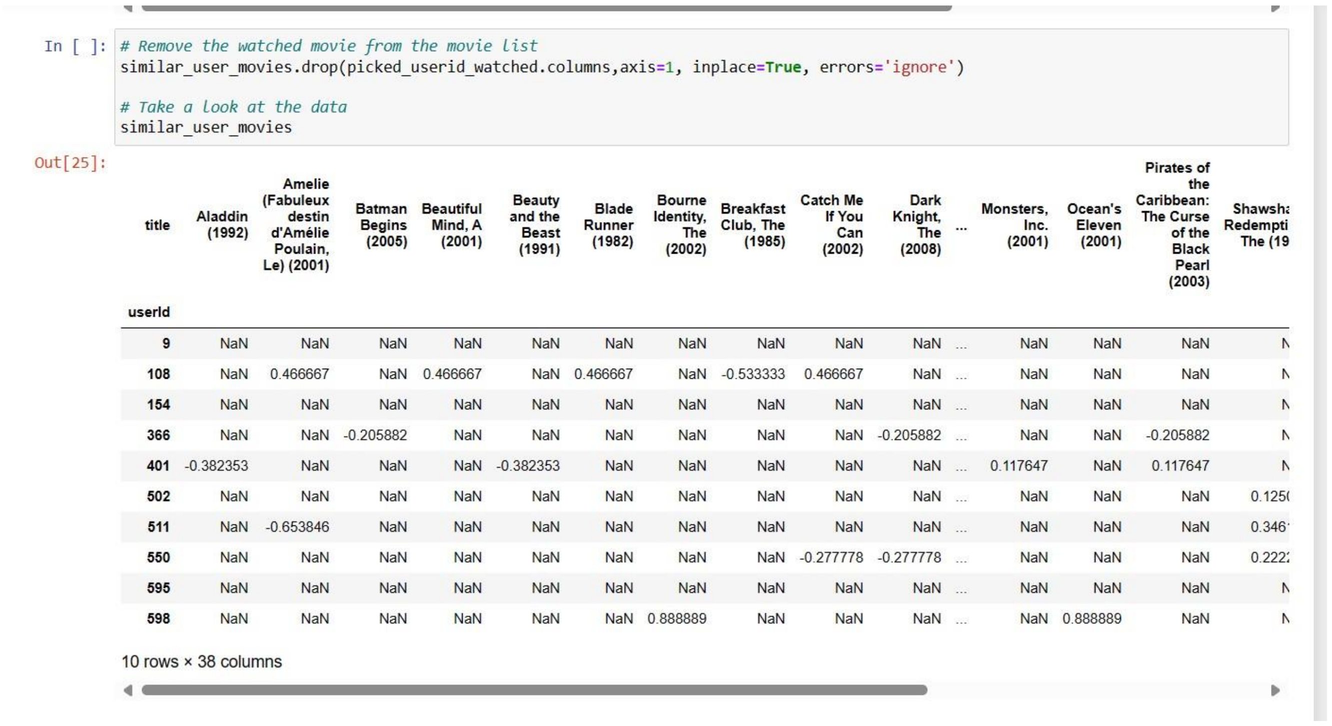


```
In [ ]: # Movies that similar users watched. Remove movies that none of the similar users have watched
        similar_user_movies = matrix_norm[matrix_norm.index.isin(similar_users.index)].dropna(axis=1, how='all')
        similar_user_movies
```

Out[24]:

title	Aladdin (1992)	Alien (1979)	Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)	Back to the Future (1985)	Batman Begins (2005)	Beautiful Mind, A (2001)	Beauty and the Beast (1991)	Blade Runner (1982)	Bourne Identity, The (2002)	Braveheart (1995)	...	Shrek (2001)	Silence of the Lambs, The (1991)	Spider-Man (2002)	Star Wars: Episode I - The Phantom Menace (1999)	Ter Ju Day
userId																
9	NaN	NaN	NaN	0.333333	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
108	NaN	NaN	0.466667	0.466667	NaN	0.466667	NaN	0.466667	NaN	NaN	...	NaN	NaN	0.466667	NaN	
154	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
366	NaN	NaN	NaN	NaN	-0.205882	NaN	NaN	NaN	NaN	-0.205882	...	NaN	NaN	NaN	NaN	-0.
401	-0.382353	NaN	NaN	NaN	NaN	NaN	-0.382353	NaN	NaN	NaN	...	0.117647	NaN	NaN	NaN	
502	NaN	-0.375	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
511	NaN	NaN	-0.653846	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	-1.153846	-0.653846	
550	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	
595	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	-0.333333	
598	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.888889	NaN	...	-2.111111	-2.611111	NaN	NaN	

10 rows × 62 columns

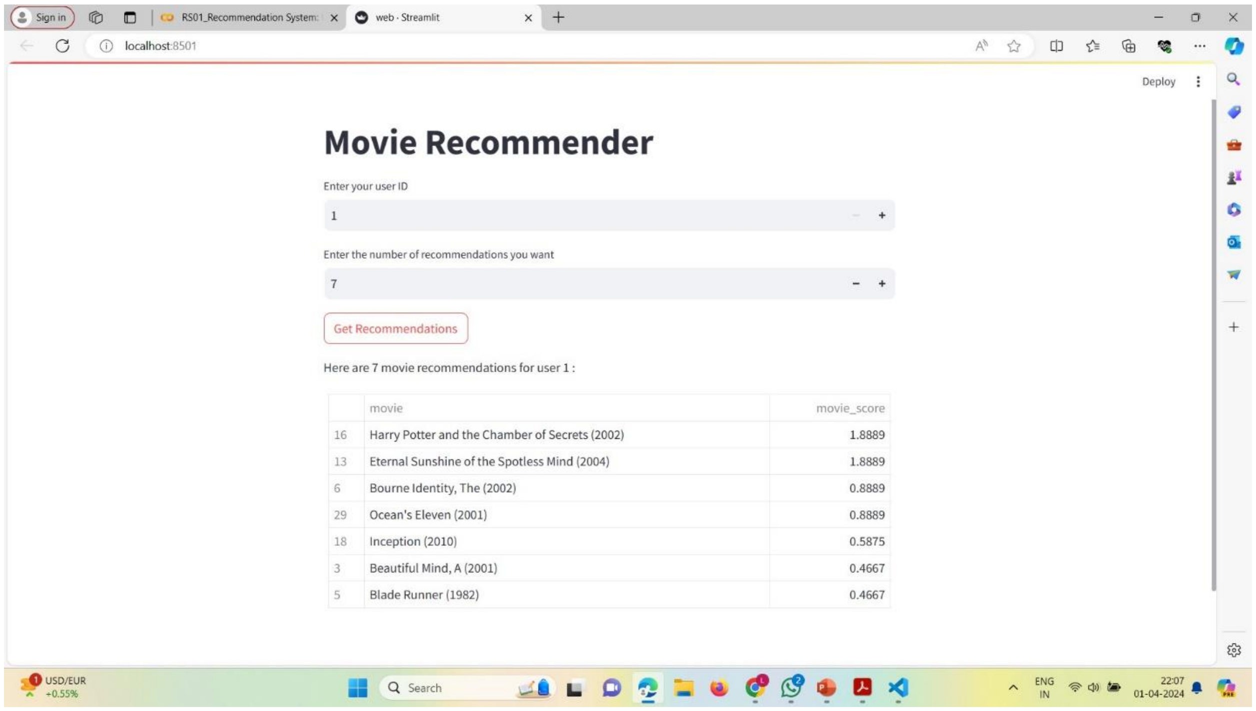


```
In [ ]: # Remove the watched movie from the movie list
        similar_user_movies.drop(picked_userid_watched.columns,axis=1, inplace=True, errors='ignore')

        # Take a look at the data
        similar_user_movies
```

Out[25]:

title	Aladdin (1992)	Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)	Batman Begins (2005)	Beautiful Mind, A (2001)	Beauty and the Beast (1991)	Blade Runner (1982)	Bourne Identity, The (2002)	Breakfast Club, The (1985)	Catch Me If You Can (2002)	Dark Knight, The (2008)	...	Monsters, Inc. (2001)	Ocean's Eleven (2001)	Pirates of the Caribbean: The Curse of the Black Pearl (2003)	Shawsha Redempti The (19
userId															
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
108	NaN	0.466667	NaN	0.466667	NaN	0.466667	NaN	-0.533333	0.466667	NaN	...	NaN	NaN	NaN	N
154	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
366	NaN	NaN	-0.205882	NaN	NaN	NaN	NaN	NaN	NaN	-0.205882	...	NaN	NaN	-0.205882	N
401	-0.382353	NaN	NaN	NaN	-0.382353	NaN	NaN	NaN	NaN	NaN	...	0.117647	NaN	0.117647	N
502	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	0.125
511	NaN	-0.653846	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	0.346
550	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	-0.277778	-0.277778	...	NaN	NaN	NaN	0.222
595	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	N
598	NaN	NaN	NaN	NaN	NaN	NaN	0.888889	NaN	NaN	NaN	...	NaN	0.888889	NaN	N

10 rows × 38 columns



Movie Recommender

Enter your user ID

1

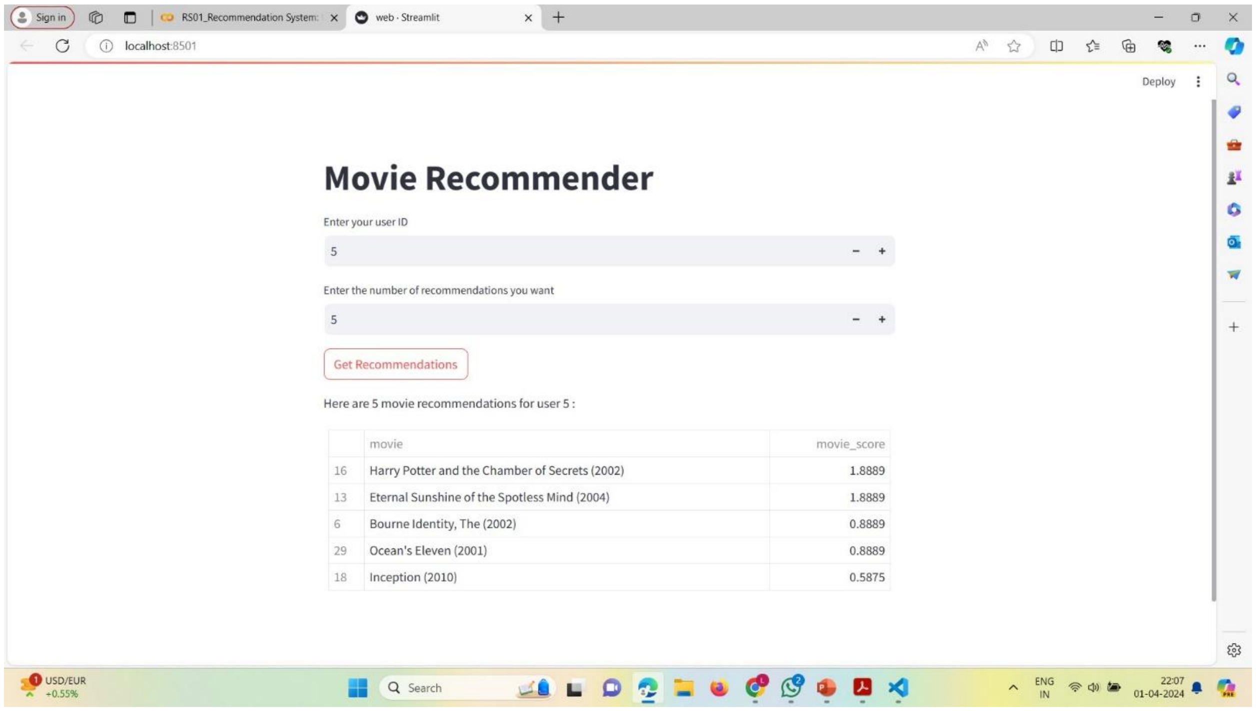
Enter the number of recommendations you want

7

Get Recommendations

Here are 7 movie recommendations for user 1 :

	movie	movie_score
16	Harry Potter and the Chamber of Secrets (2002)	1.8889
13	Eternal Sunshine of the Spotless Mind (2004)	1.8889
6	Bourne Identity, The (2002)	0.8889
29	Ocean's Eleven (2001)	0.8889
18	Inception (2010)	0.5875
3	Beautiful Mind, A (2001)	0.4667
5	Blade Runner (1982)	0.4667



Movie Recommender

Enter your user ID

5

Enter the number of recommendations you want

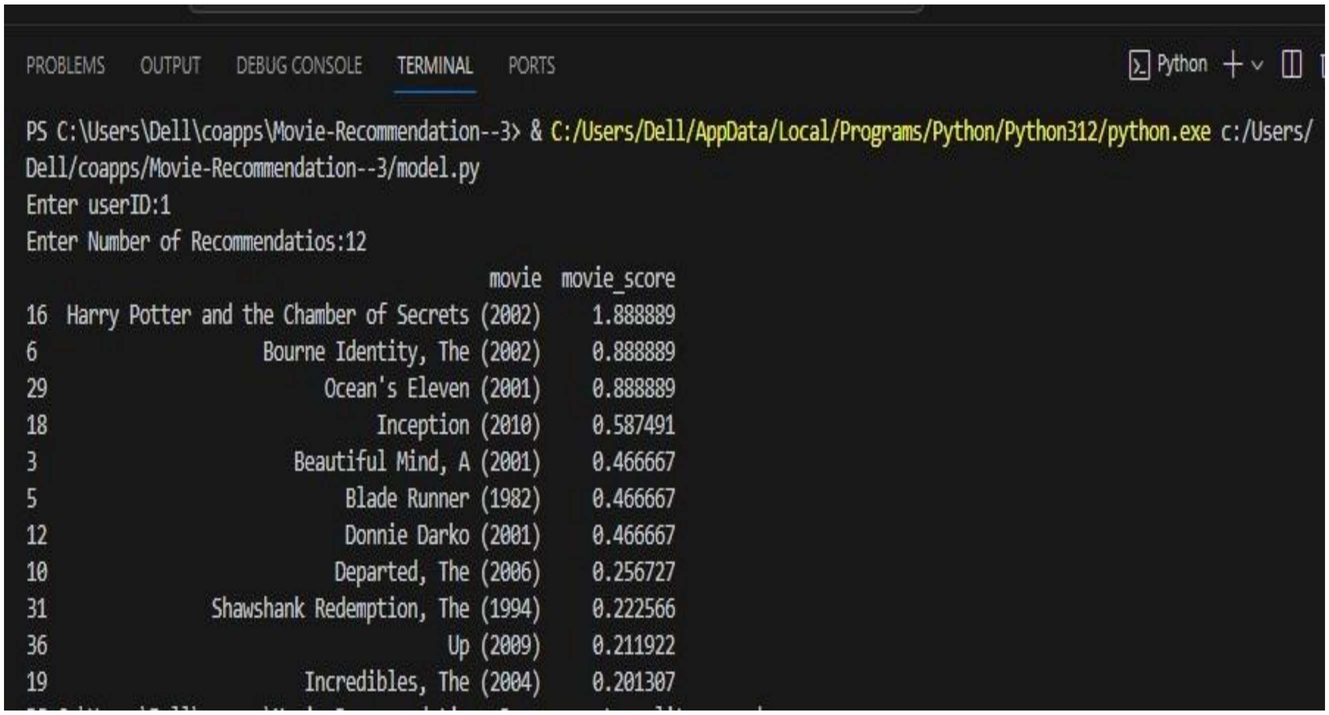
5

Get Recommendations

Here are 5 movie recommendations for user 5 :

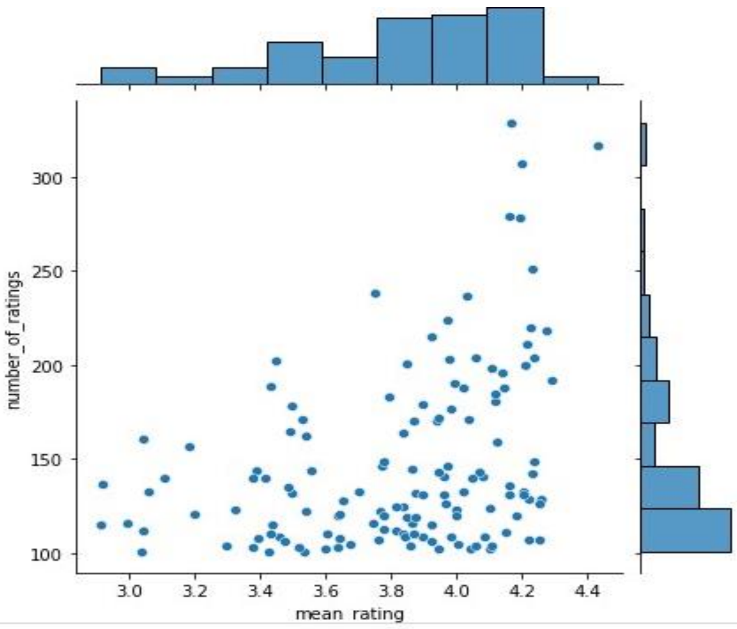
	movie	movie_score
16	Harry Potter and the Chamber of Secrets (2002)	1.8889
13	Eternal Sunshine of the Spotless Mind (2004)	1.8889
6	Bourne Identity, The (2002)	0.8889
29	Ocean's Eleven (2001)	0.8889
18	Inception (2010)	0.5875

5.2 Output:



```
PS C:\Users\Dell\coapps\Movie-Recommendation--3> & C:/Users/Dell/AppData/Local/Programs/Python/Python312/python.exe c:/Users/
Dell/coapps/Movie-Recommendation--3/model.py
Enter userID:1
Enter Number of Recommendatios:12
                                       movie  movie_score
16  Harry Potter and the Chamber of Secrets (2002)     1.888889
6                        Bourne Identity, The (2002)     0.888889
29                            Ocean's Eleven (2001)     0.888889
18                                 Inception (2010)     0.587491
3                         Beautiful Mind, A (2001)     0.466667
5                              Blade Runner (1982)     0.466667
12                             Donnie Darko (2001)     0.466667
10                            Departed, The (2006)     0.256727
31                 Shawshank Redemption, The (1994)     0.222566
36                                        Up (2009)     0.211922
19                           Incredibles, The (2004)     0.201307
```

Result:



Chapter-6 : Conclusion:

In conclusion, "FlickPicks: Personalized Movie Recommendations using Collaborative Filtering" presents a comprehensive approach to building an effective movie recommendation system based on collaborative filtering techniques. Through the implementation of user-based and item-based collaborative filtering algorithms, the system aims to provide personalized movie suggestions tailored to individual user preferences.

The study demonstrates the effectiveness of the FlickPicks system through rigorous evaluation and experimentation, showcasing its ability to generate accurate and relevant movie recommendations. By leveraging large datasets containing user interactions with movies, FlickPicks can identify patterns and similarities in user behavior, allowing it to predict user preferences for unseen movies with high accuracy.

The evaluation metrics indicate that FlickPicks significantly improves user satisfaction and engagement by offering personalized movie recommendations aligned with users' tastes. User testing and feedback further validate the system's effectiveness in enhancing the movie-watching experience and increasing user satisfaction.

Overall, FlickPicks contributes to the advancement of personalized content recommendation systems in the entertainment domain, providing a valuable tool for users to discover new movies tailored to their interests. Future research could explore additional techniques for improving recommendation accuracy and scalability, as well as integrating other factors such as contextual information and social interactions to further enhance the recommendation process.