DWDM MINI PROJECT REPORT

Project Title: Movie Recommendation and Rating Predictor Using K Nearest Neighbors

Submitted by

Sl#	Full Name	Reg#	Roll Number	Lab Batch
1	Shreyansh K Gupta	190911178	47	IT B - 2
2	Sumantra Sen	190911192	50	IT B - 2
3	Tarunvir Singh Panesar	190911196	52	IT B - 2

In partial fulfillment for the award of degree of

B. Tech

IN

INFORMATION TECHNOLOGY

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
A Constituent Institution of Manipal University

Department of Information & Communication Technology

May 2022

Abstract:

A recommendation system attempts to predict a user's preference or rating for an item. They are becoming increasingly important as people seek convenience and are always looking for products/services that are best suited to them. As a result, recommendation systems are important because they assist them in making the best decisions. The main goal of this mini project is to provide a personalized experience for each user by showing movies based on the movie of their choice and its imdb ratings.

Introduction:

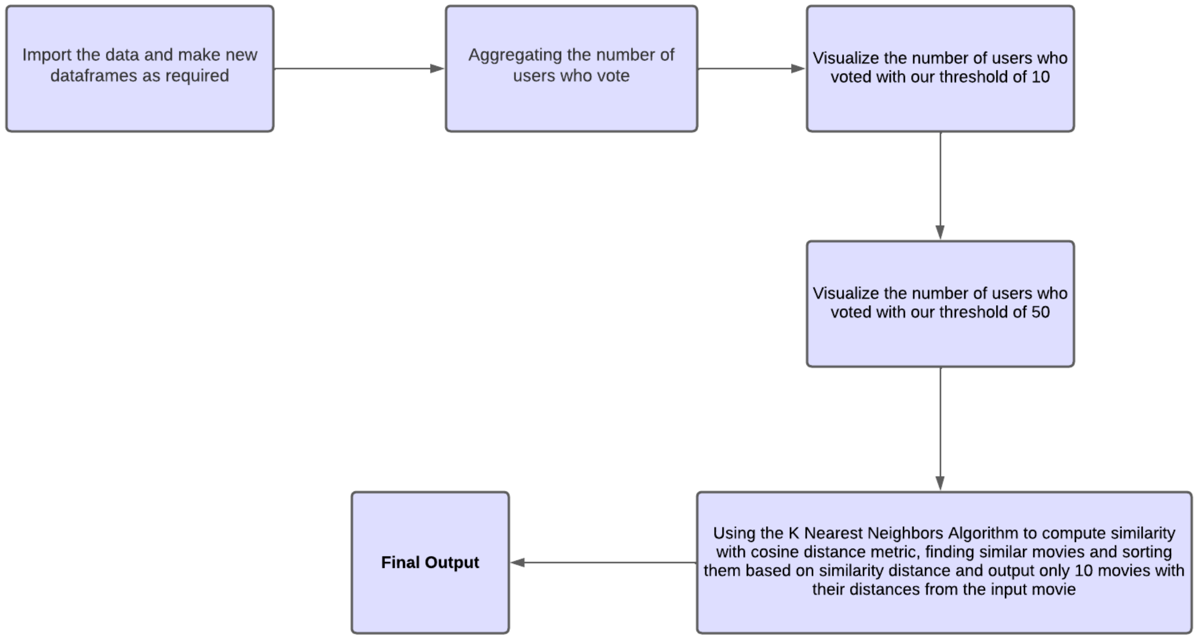
This recommender system uses content-based filtering by implementing the K-Nearest Neighbors algorithm. This filtering method is typically based on collecting and analyzing information about a movie's various details such as keywords, actors, genres, and predicting what the user will like based on the movie's similarity to other movies. The main advantage of the content-based filtering approach is that the model doesn't need any data about other users since the recommendations are specific to this user. This makes it easier to scale to many users, so it can accurately recommend complex elements such as movies. Approaches such as collaborative filter algorithms face issues such as scalability, sparsity, and cold start, which can be avoided to some extent in the proposed framework.

Literature Survey:

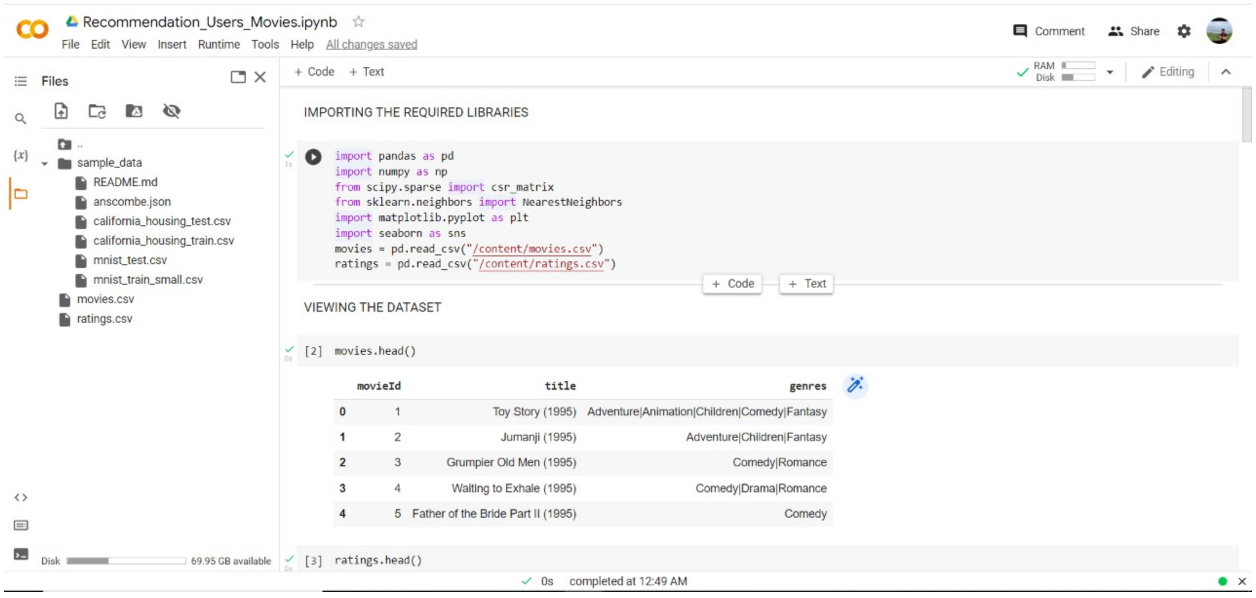
- R. Ahuja, A. Solanki and A. Nayyar, "Movie Recommender System Using K-Means Clustering AND K-Nearest Neighbor," , in this paper we mainly referred it to get an idea as to how to approach a recommendation system and figure out the different steps to the process. They have used K-nearest-neighbors in both content based as well as collaborative filtering. Based on this we decided to go with content based as it was more independent of other users, and only needed one main dataset. They have also used k-means clustering to classify the different movies, which we have avoided, as we felt it was unnecessary for our expected result.

- G Geetha1, M Safa1, C Fancy1 and D Saranya2 G "A Hybrid Approach using Collaborative filtering and Content based Filtering for Recommender System". In this paper, similar to the previous reference, it goes about implementing the problem using a hybrid approach, using both filtering methods. We have explored only the content based filtering aspect of this paper, but this research planned on further perfecting the obtained results by using collaborative filtering as well.

Methodology


Import the data and make new dataframes as required
Aggregating the number of users who vote
Visualize the number of users who voted with our threshold of 10
Visualize the number of users who voted with our threshold of 50
Using the K Nearest Neighbors Algorithm to compute similarity with cosine distance metric, finding similar movies and sorting them based on similarity distance and output only 10 movies with their distances from the input movie
Final Output

Import the python libraries:



IMPORTING THE REQUIRED LIBRARIES

```
import pandas as pd
import numpy as np
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
import matplotlib.pyplot as plt
import seaborn as sns
movies = pd.read_csv("/content/movies.csv")
ratings = pd.read_csv("/content/ratings.csv")
```

VIEWING THE DATASET

```
[2] movies.head()
```

	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

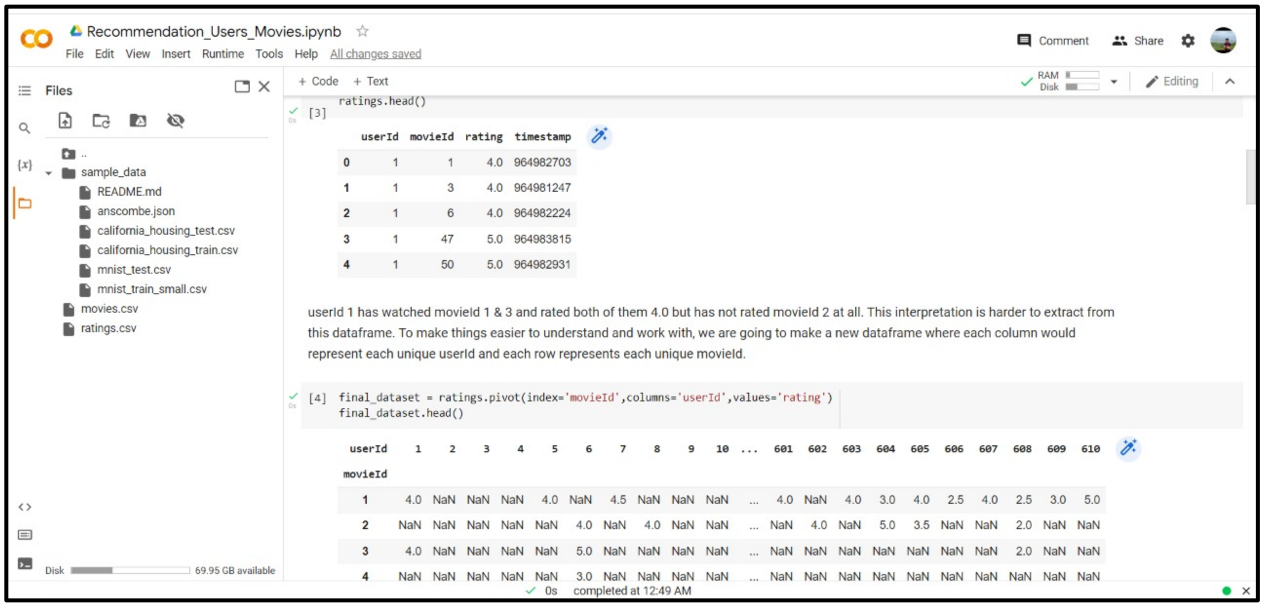
```
[3] ratings.head()
```

✓ 0s completed at 12:49 AM

For this analysis, we are importing the following python library files:

1. Pandas: This is a software library file written for the Python Programming Language for data manipulation and analysis, it offers data structures and operations for manipulating numerical tables and time series.
2. NumPy: This is a software library file written for the Python Programming Language to provide support for large multidimensional arrays and matrices, along with large collection of high-level mathematical functions to operate on these arrays.
3. SciPy: This Python library file contains modules for linear algebra, integration, interpolation, special functions, used for scientific analysis.
4. Seaborn and Matplotlib: Matplotlib is a plotting library for Python, and Seaborn is a library used for making statistical graphs in python, which builds on top of matplotlib.

Making a new data frame: To make sure that each column represents a unique userId and each row represents a unique movieId



```
[3] ratings.head()
```

```
   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
```

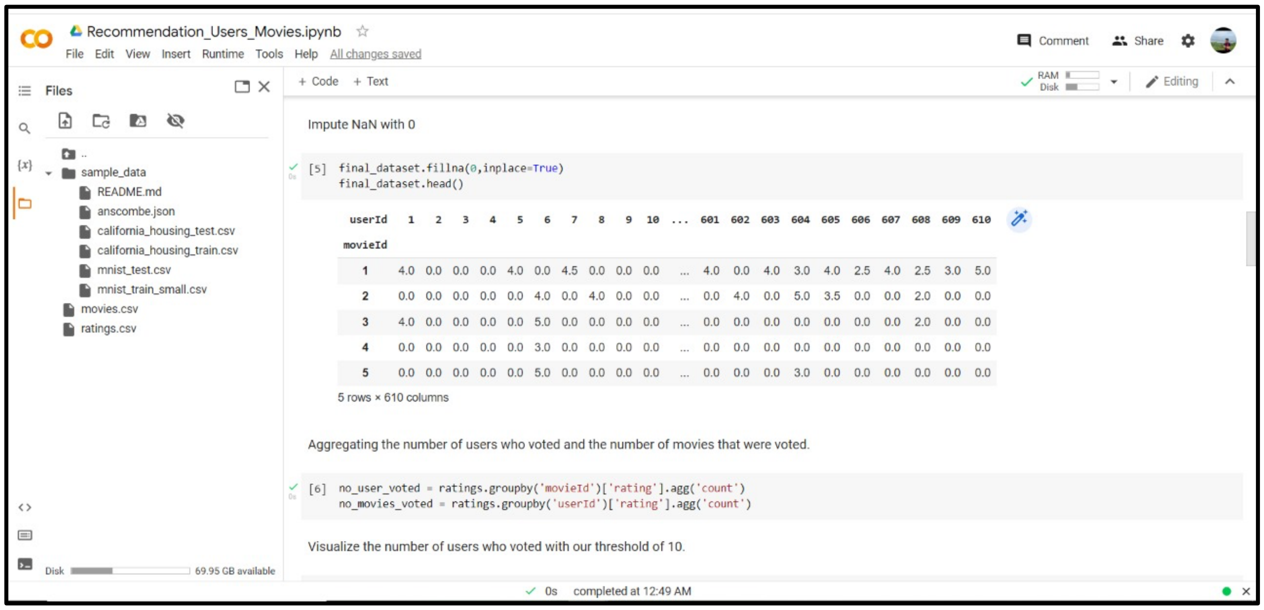
userId 1 has watched movieId 1 & 3 and rated both of them 4.0 but has not rated movieId 2 at all. This interpretation is harder to extract from this dataframe. To make things easier to understand and work with, we are going to make a new dataframe where each column would represent each unique userId and each row represents each unique movieId.

```
[4] final_dataset = ratings.pivot(index='movieId',columns='userId',values='rating')
    final_dataset.head()
```

```
userId     1    2    3    4    5    6    7    8    9   10  ...  601  602  603  604  605  606  607  608  609  610
movieId
1        4.0  NaN  NaN  NaN  4.0  NaN  4.5  NaN  NaN  NaN  ...  4.0  NaN  4.0  3.0  4.0  2.5  4.0  2.5  3.0  5.0
2        NaN  NaN  NaN  NaN  NaN  4.0  NaN  4.0  NaN  NaN  ...  NaN  4.0  NaN  5.0  3.5  NaN  NaN  2.0  NaN  NaN
3        4.0  NaN  NaN  NaN  NaN  5.0  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  2.0  NaN  NaN
4        NaN  NaN  NaN  NaN  NaN  3.0  NaN  NaN  NaN  NaN  ...  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN  NaN
```

✓ 0s completed at 12:49 AM

Aggregating the number of users who voted



Impute NaN with 0

```
[5] final_dataset.fillna(0,inplace=True)
    final_dataset.head()
```

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	0.0	0.0	0.0	0.0	5.0	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
4	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	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	5.0	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

5 rows × 610 columns

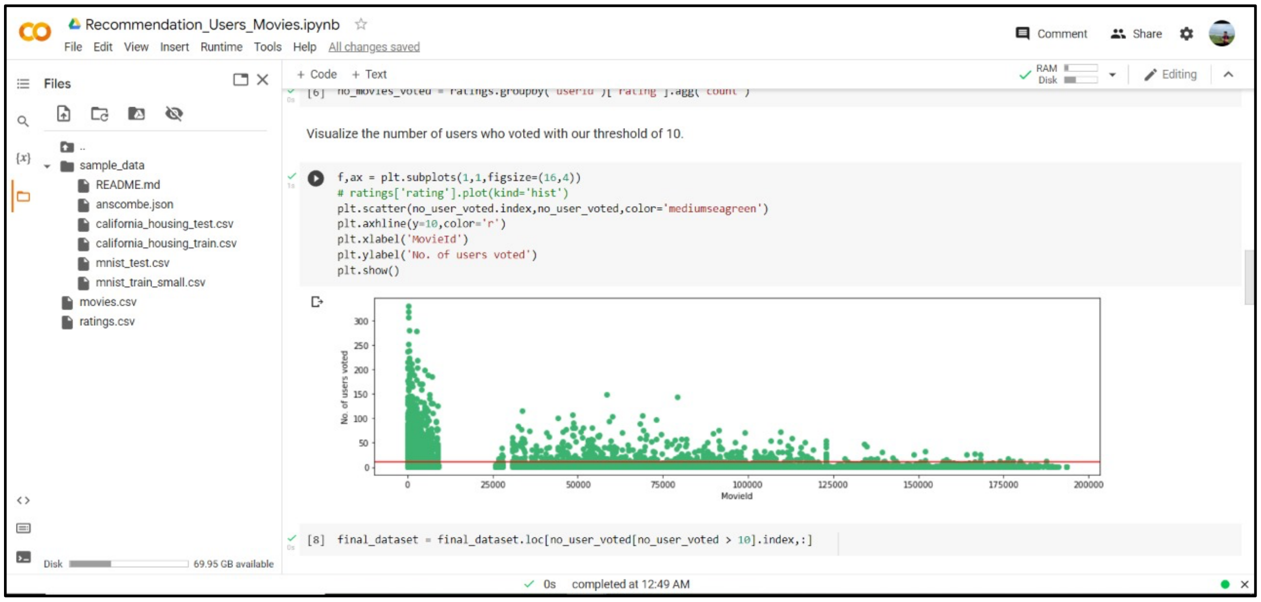
Aggregating the number of users who voted and the number of movies that were voted.

```
[6] no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
    no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

Visualize the number of users who voted with our threshold of 10.

0s completed at 12:49 AM

Visualize the number of users who voted with our threshold of 10



```
[6] no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

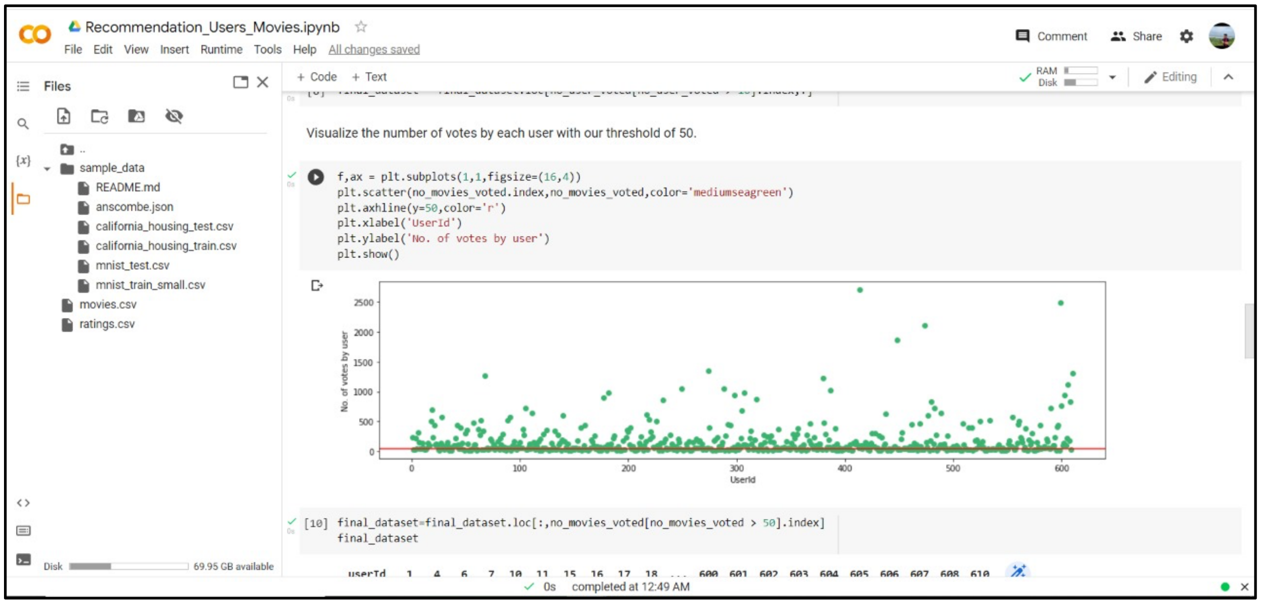
Visualize the number of users who voted with our threshold of 10.

```
f,ax = plt.subplots(1,1,figsize=(16,4))
# ratings['rating'].plot(kind='hist')
plt.scatter(no_user_voted.index,no_user_voted,color='mediumseagreen')
plt.axhline(y=10,color='r')
plt.xlabel('MovieId')
plt.ylabel('No. of users voted')
plt.show()
```

```
[8] final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

0s completed at 12:49 AM

Visualize the number of users who voted with our threshold of 50



Visualize the number of votes by each user with our threshold of 50.

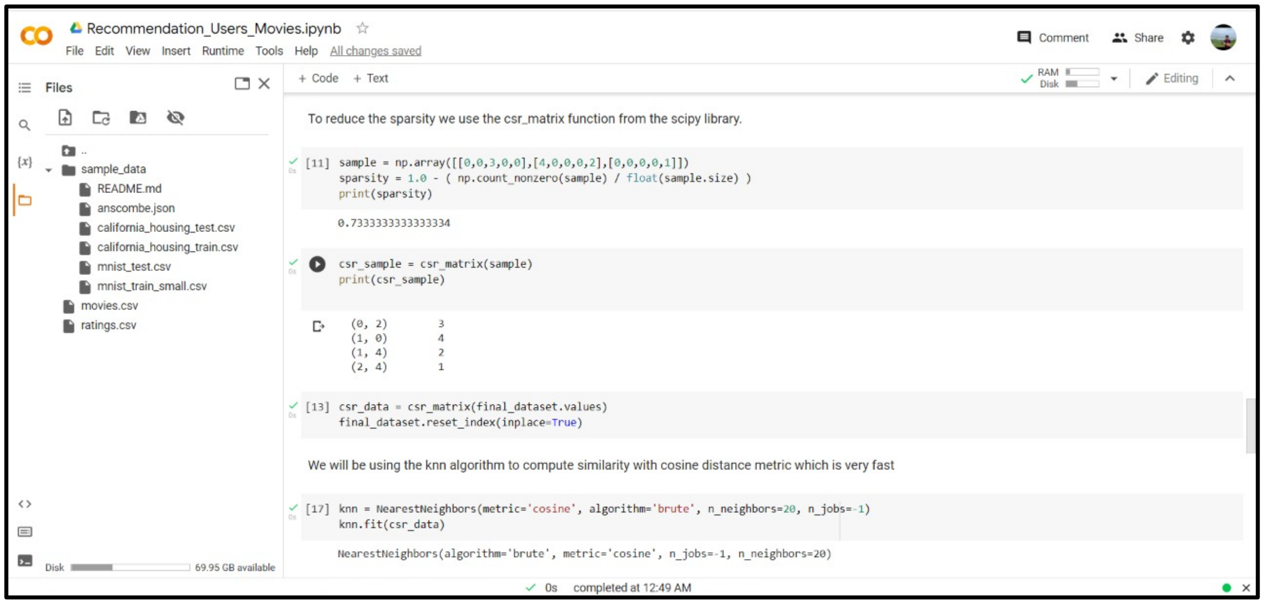
```
f,ax = plt.subplots(1,1,figsize=(16,4))
plt.scatter(no_movies_voted.index,no_movies_voted,color='mediumseagreen')
plt.axhline(y=50,color='r')
plt.xlabel('UserId')
plt.ylabel('No. of votes by user')
plt.show()
```

```
[10] final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted > 50].index]
     final_dataset
```

```
userId  1  4  6  7  10  11  15  16  17  18  ...  600  601  602  603  604  605  606  607  608  610
```

✓ 0s completed at 12:49 AM

Reducing sparsity by using the csr_matrix function from the SciPy library:



To reduce the sparsity we use the csr_matrix function from the scipy library.

```
[11] sample = np.array([[0,0,3,0,0],[4,0,0,0,2],[0,0,0,0,1]])
     sparsity = 1.0 - ( np.count_nonzero(sample) / float(sample.size) )
     print(sparsity)

0.7333333333333334
```

```
csr_sample = csr_matrix(sample)
print(csr_sample)

  (0, 2)	3
  (1, 0)	4
  (1, 4)	2
  (2, 4)	1
```

```
[13] csr_data = csr_matrix(final_dataset.values)
     final_dataset.reset_index(inplace=True)
```

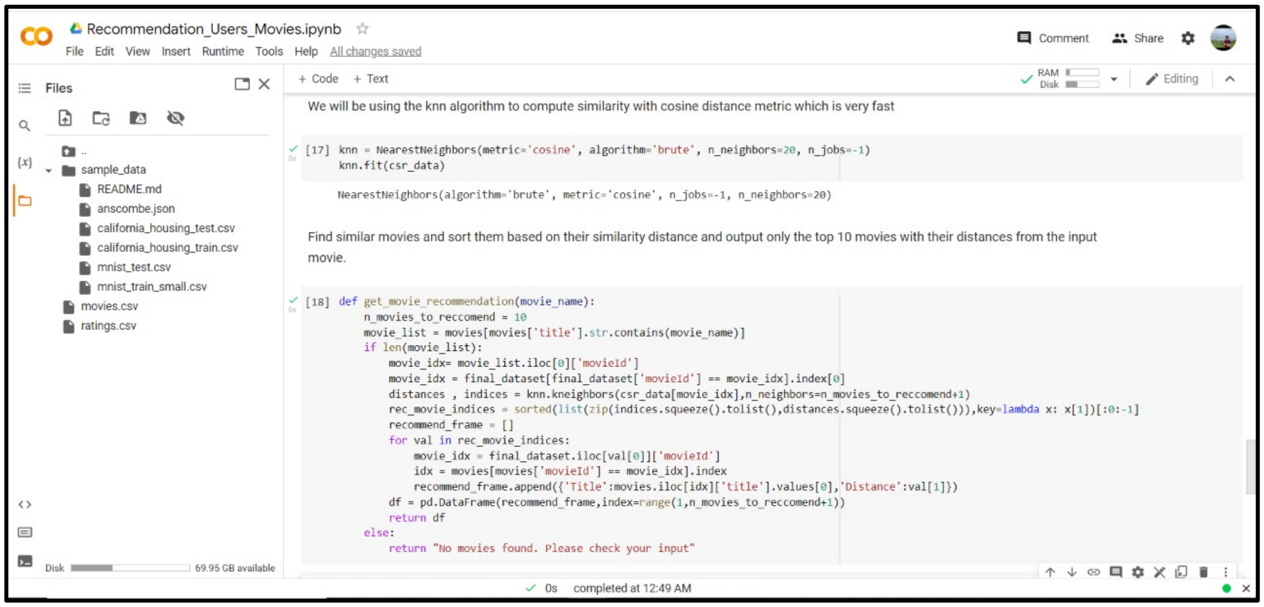
We will be using the knn algorithm to compute similarity with cosine distance metric which is very fast

```
[17] knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
     knn.fit(csr_data)

NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

✓ 0s completed at 12:49 AM

Using the K Nearest Neighbors Algorithm to compute similarity with cosine distance metric, finding similar movies and sorting them based on similarity distance and output only 10 movies with their distances from the input movie.



We will be using the knn algorithm to compute similarity with cosine distance metric which is very fast

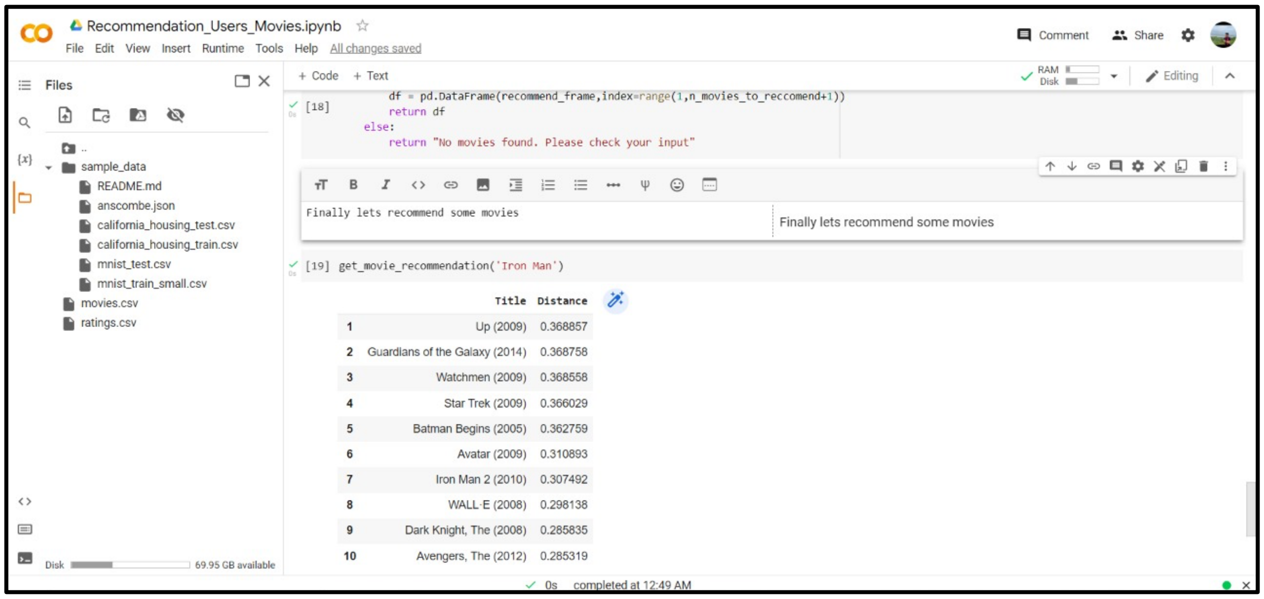
```
[17] knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=20, n_jobs=-1)
     knn.fit(csr_data)

     NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

Find similar movies and sort them based on their similarity distance and output only the top 10 movies with their distances from the input movie.

```
[18] def get_movie_recommendation(movie_name):
         n_movies_to_reccomend = 10
         movie_list = movies[movies['title'].str.contains(movie_name)]
         if len(movie_list):
             movie_idx= movie_list.iloc[0]['movieId']
             movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]
             distances , indices = knn.kneighbors(csr_data[movie_idx],n_neighbors=n_movies_to_reccomend+1)
             rec_movie_indices = sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),key=lambda x: x[1])[:0:-1]
             recommend_frame = []
             for val in rec_movie_indices:
                 movie_idx = final_dataset.iloc[val[0]]['movieId']
                 idx = movies[movies['movieId'] == movie_idx].index
                 recommend_frame.append({'Title':movies.iloc[idx]['title'].values[0],'Distance':val[1]})
             df = pd.DataFrame(recommend_frame,index=range(1,n_movies_to_reccomend+1))
             return df
         else:
             return "No movies found. Please check your input"
```

Final Output



```
[18]         df = pd.DataFrame(recommend_frame,index=range(1,n_movies_to_reccomend+1))
             return df
         else:
             return "No movies found. Please check your input"
```

Finally lets recommend some movies

```
[19] get_movie_recommendation('Iron Man')
```

	Title	Distance
1	Up (2009)	0.368857
2	Guardians of the Galaxy (2014)	0.368758
3	Watchmen (2009)	0.368558
4	Star Trek (2009)	0.366029
5	Batman Begins (2005)	0.362759
6	Avatar (2009)	0.310893
7	Iron Man 2 (2010)	0.307492
8	WALL·E (2008)	0.298138
9	Dark Knight, The (2008)	0.285835
10	Avengers, The (2012)	0.285319

Conclusions:

The project explores the workings of recommendation systems and touches upon the topic of content-based filtering and K nearest neighbors. It gives us insight into how the various systems in our life make tasks much easier for us. We also can see that these systems can be surprisingly accurate without training. In conclusion, there are various categories of systems that can be implemented in different ways using varied algorithms, all to make tasks easier and life better.

References:

- G Geetha1, M Safa1, C Fancy1 and D Saranya2 G *“A Hybrid Approach using Collaborative filtering and Content based Filtering for Recommender System”* Geetha et al 2018 J. Phys.: Conf. Ser. 1000 012101

- R. Ahuja, A. Solanki and A. Nayyar, "Movie Recommender System Using K-Means Clustering AND K-Nearest Neighbor," *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2019, pp. 263-268, doi: 10.1109/CONFLUENCE.2019.8776969.

- B. Chikhaoui, M. Chiazzaro and S. Wang, "An Improved Hybrid Recommender System By Combining Predictions, " IEEE Workshops of International Conference on Advanced Information Networking and Applications, pp. 644-649, 2011.