Project Report

On

MOVIE RECOMMENDATION SYSTEM

Submitted by

Sadhu Muni Chithra

B Vijay

G Vamsi Krishna

Under the guidance of

Mrs P. Siva Lakshmi

Assistant Professor

Department of Computer Science and Engineering

Rajiv Gandhi University of Knowledge and Technologies R.K.Valley, Kadapa

Andhrapradesh

Rajiv Gandhi University of Knowledge Technologies

RK Valley, Kadapa (Dist), Andhra Pradesh, 516330

DECLARATION

We hereby declare that the report of the B.Tech Mini Project Work entitled "**MOVIE RECOMMENDATION SYSTEM**" which is being submitted to Rajiv Gandhi University of Knowledge Technologies, RK Valley, in partial fulfillment of the requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering, is a bonafide report of the work carried out by me. The material contained in this report has not been submitted to any university or institution for award of any degree.

Sadhu Muni Chithra – R170631

Bujjigalla Vijay- R170659

G Vamsi Krishna-R170656

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

RGUKT

(A.P.Government Act 18 of 2008)

RGUKT, RK VALLEY

Department of Computer Science and Engineering

CERTIFICATE FOR PROJECT COMPLETION

This is certify that the project entitled "**MOVIE RECOMMNENDATION SYSTEM**" submitted by **S Muni Chithra (R170631) B Vijay(R170659),G Vamsi Krishna(R170656)** under our guidance and supervision for the partial fulfillment for the degree Bachelor of Technology in Computer Science and Engineering during the academic year 2021-2022 at RGUKT, RK VALLEY. To the best of my knowledge, the results embodied in this dissertation work have not been submitted to any University or Institute for the award of any degree or diploma.

Project Internal Guide	**Head of the Department**
Mrs P.Siva Lakshmi	Mr.P.Harinadha
Project Coordinator,	
Assistant Professor,	HOD Of CSE,
RGUKT, RK Valley.	RGUKT, RK Valley.

Acknowledgement

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of the people who made it possible and whose constant guidance and encouragement crown all the efforts success. I am extremely grateful to our respected Director,Prof. **K.SANDHYA RANI f**or fostering an excellent academic climate in our institution. I also express my sincere gratitude to our respected Head of the Department **Mr P.HARINADHA f**or their encouragement, overall guidance in viewing this project a good asset and effort in bringing out this project. I would like to convey thanks to our guide at college **Mrs.P.Siva Lakshmi** for their guidance, encouragement, co-operation and kindness during the entire duration of the course and academics. My sincere thanks to all the members who helped me directly and indirectly in the completion of project work. I express my profound gratitude to all our friends and family members for their encouragement.

Index

Abstract

The main objective of this project is to suggest movie recommendation to users depending on variety of criteria. These systems estimate the most likely product that consumers will buy and that they will be interested in Netflix,Amazon,and other companies use recommend system to help their users fin te right product or movie for them.

Content Based: We recommend movies based on content of films using TFIDF Vectorizer and Cosine Similarity Matrix.

Item-based: We recommend movies based on rating of films using KNN algorithm.

INTRODUCTION

- A recommend system is a simple algorithm whose aim to provide the most relevant information to a user by discovering the patterns in a dataset. The algorithm rates the items and shows the user that would rate highly.

An example of recommendation in action is when we visit Netflix and you notice that some movies are being recommend. They are also used by Amazon, Spotify and Zomato.

TYPES OF MOVIE RECOMMENDATION SYSTEMS

Content Based Recommendation System:

This filtering is based on the description or some data provided for that product. The system finds the similarity between products based on its context or description. The user's previous history is taken into account to find similar products the user may like.

For example, if a user likes movies such as 'Mission Impossible' then we can recommend him the movies of 'Tom Cruise' or movies with the genre 'Action'.

- The user gets recommended the types of items they love.The user is satisfied by the type of recommendation.

New items can be recommended; just data for that item is required.

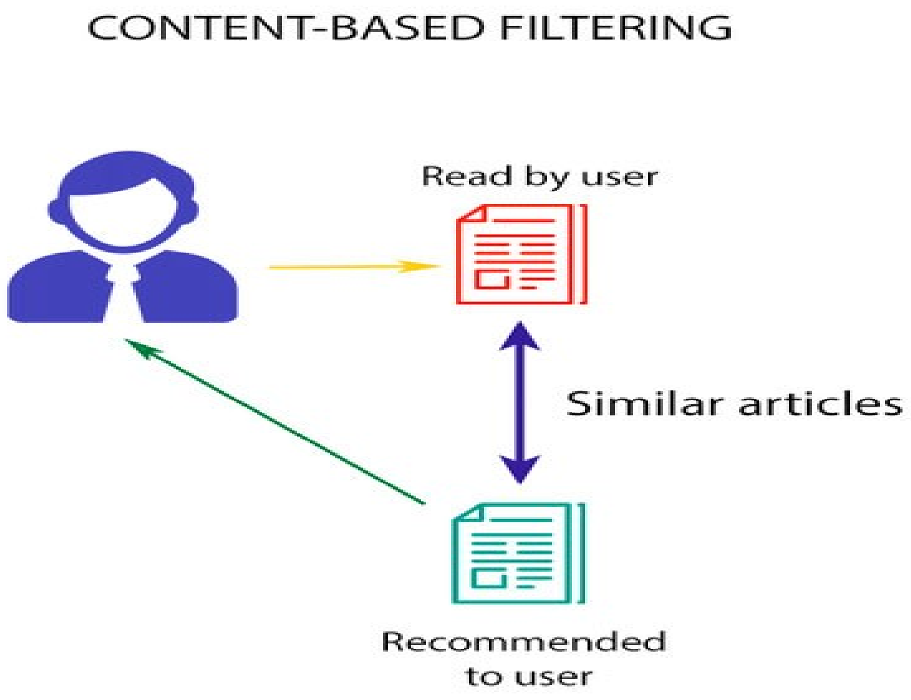


Fig1- Content-Based Filtering

Disdvantages:

- Different products do not get much explosure to the user.
- Businessess cannot be expanded as the user does not try different types of products

Collaborative Based Recommendation System:

The recommendations are done based on the user's behavior. History of the user plays an important role. For example, if the user 'A' likes 'Coldplay', ‘The Linkin Park’ and‘Britney Spears’ while the user ‘B’ likes ‘Coldplay’, ‘The Linkin Park’ and ‘Taylor Swift’ then they have similar interests.So,there is a huge probability that the user ‘A’ would like ‘Taylor Swift’ and the user ‘B’ would like ‘Britney Spears’. This is the way collaborative filtering is done.

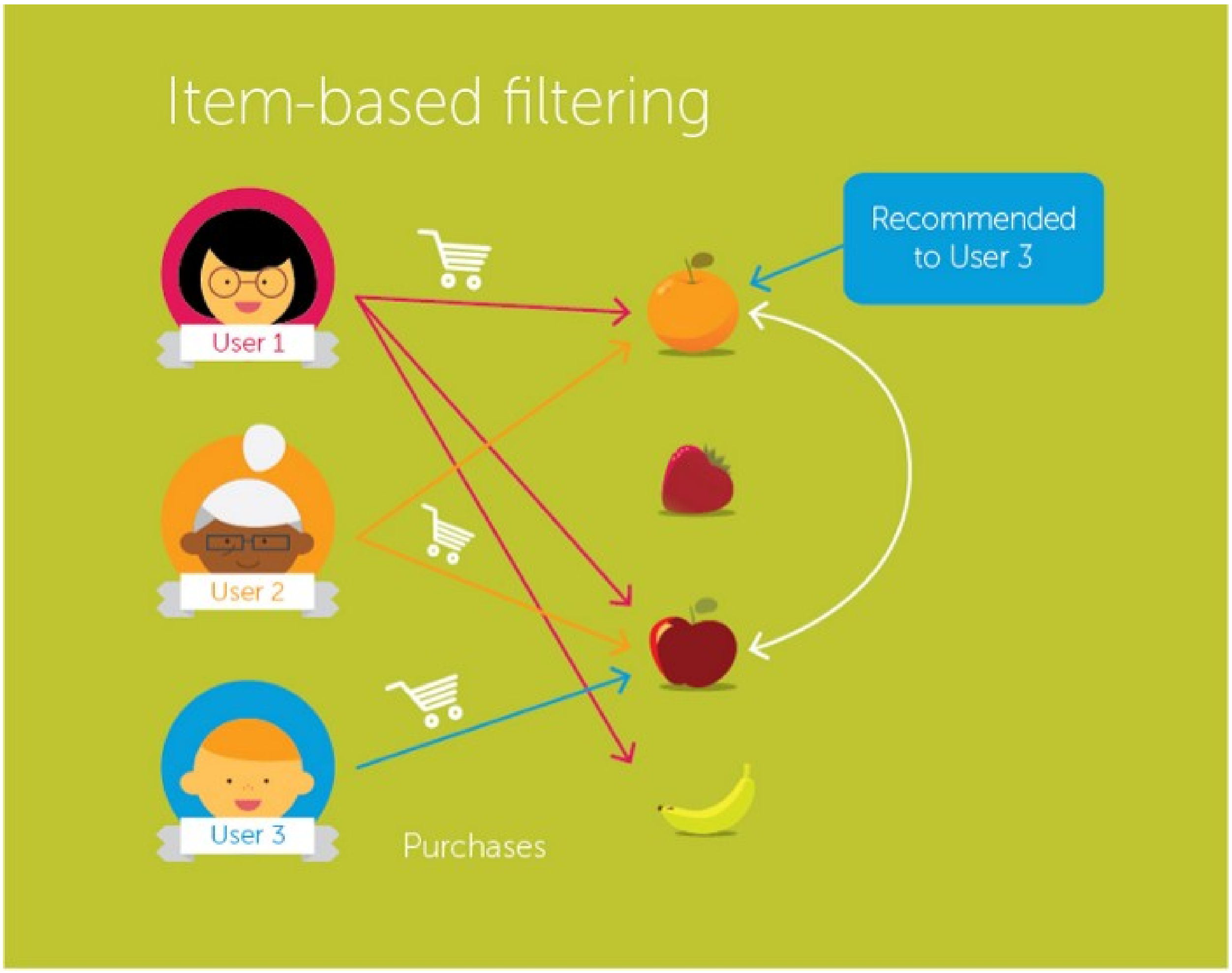


Fig2-Item-based Filtering

●Advantages:

--->New products can be introduced to the user.

---->Business can be expanded and can popularise new products.

●Dis-Advantages:

--->User's previous history is required or data for products is required based on the type of collaborative method used.

--->The new item cannot be recommended if no user has purchased or rated it

WORK FLOW

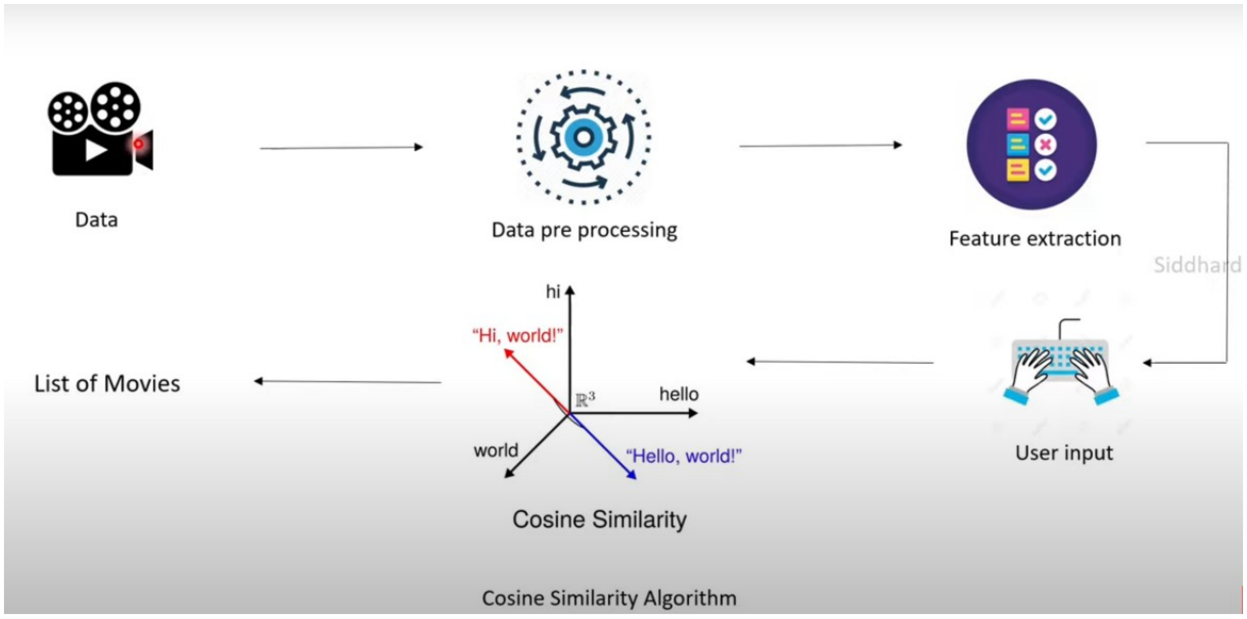


Fig-3 Work Flow

Data:

- First we need to collect the dataset that means details of the movie such as actor,genre,description of the movie

Data Pre-Processing :

- Once we collect the data we need to process the data.
- So we have to clean this data like there are any missing values in the data.
- That part will be update in this step.

Feature Extraction :

- All the details of the movie in textual format.we cannot use the data directly.

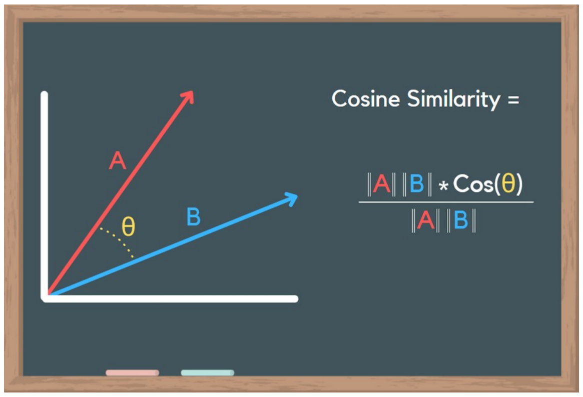
- So, we need to convert into numerical data in these step.

User Input :

- After that we will try to find which movies are similarity each other.
- Once we do that As user give the input
- For example: user give the name of their favourite movie based on this input we are going to suggest the movies they can watch.
- For these we are going to use cosine similarity algorithm.

TERMINOLOGIES

1.Cosine Similarity :



- We know that each movies is converting into kind of a vector
- We will try to find similarity between them using cosine similarity algorithms.
- The user give the movie name and try to compare that movie and we will just find which movies similar to the user movies.
- Then we will get list of movies,after that we suggest to the user.

2.TFIDF Vectorizer :

TF (Term Frequency):

- In the document, words are present so many times that is called term frequency.
- Term frequency is the number of times a given term or query apears within a search index.
- Since the ordering of terms is not significant, we can use a vector to describe the text in the bag of term models.
- For each specific term in the paper, there is an entry with the value being the term frequency.

To import Tfidf vector in python:

from sklearn.feature_extraction.text import TfidfVectorizer

- The weight of a term that occurs in a document is simply proportional to the term frequency.
- Formula to calculate Term Frequency:
- tf(d,f)=count of t in d / number of words in d Where , t is the given word d is the document

2.1.<u>Inverse Document Frequency</u>

- Inverse Document Frequency (IDF) is a weight indicating how commonly a word is used.
- The more frequent its usage across documents, the lower its score.
- The lower the score, the less important the word becomes.
- IDF is typically used to boost the scores of words that are unique to a document with the hope that you surface high information words that characterize your document and suppress words that don't carry much weight in a document.

- Traditionally IDF is computed as

$$idf(w, D) = log(\frac{N}{f(w, D)})$$

- N is the total number of documents in your text collection.
- DFt is the number of documents containing the term t.
- t is any word in your vocabulary.
- Tf-idf is one of the best metrics to determine how significant a term is to a text in a series or a corpus.
- tf-idf is a weighting system that assigns a weight to each word in a document based on its term frequency (tf) and the reciprocal document frequency (idf).
- The words with higher scores of weight are deemed to be more significant.
- ***Computation of tf-idf:***
- **tf-idf(t,d)=tf(t,d)*idf(t)**

3.KNN Algorithm :

K-nearest neighbors (KNN) algorithm uses ‘feature similarity’ to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps −

Step 1 − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 − For each point in the test data do the following −

- **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.

- **3.2** − Now, based on the distance value, sort them in ascending order.
- **3.3** − Next, it will choose the top K rows from the sorted array.
- **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 − End

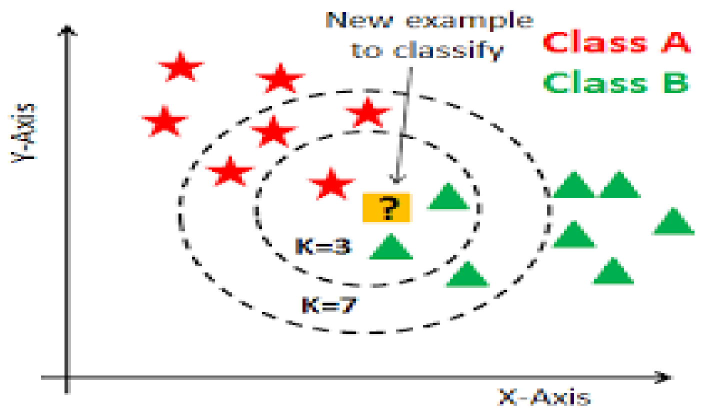


Fig-4 KNN Algorithm

Brief Description about Implementation

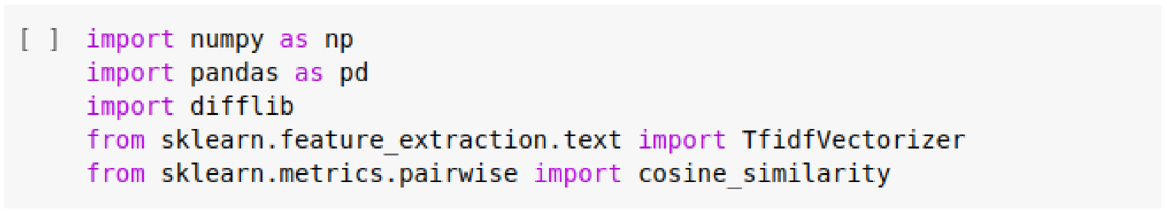
Content-Based

● **Implementation:**

- Loading the data from the csv file to a pandas dataframe
- selecting the relevant features for recommendation
- replacing the null values with null string
- combining all the 5 selected features
- converting the text data to feature vectors using Tfidvectorizer.
- Getting the similarity scores using cosine similarity

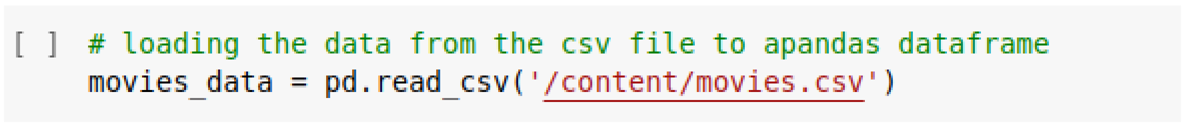
- getting the movie name from the user
- Creating a list with all the movie names given in the dataset
- finding the close match for the movie name given by the user
- finding the index of the movie with title
- Getting a list of similar movies
- Sorting the movies based on their similarity score
- print the name of similar movies based on the index.

Importing the dependencies

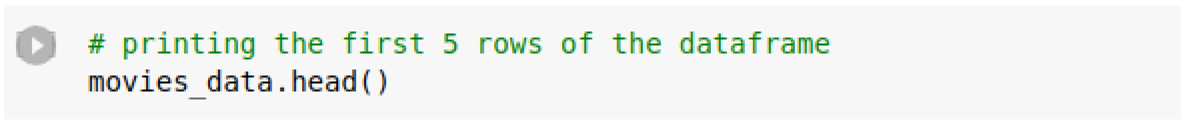


```
[ ] import numpy as np
    import pandas as pd
    import difflib
    from sklearn.feature_extraction.text import TfidfVectorizer
    from sklearn.metrics.pairwise import cosine_similarity
```

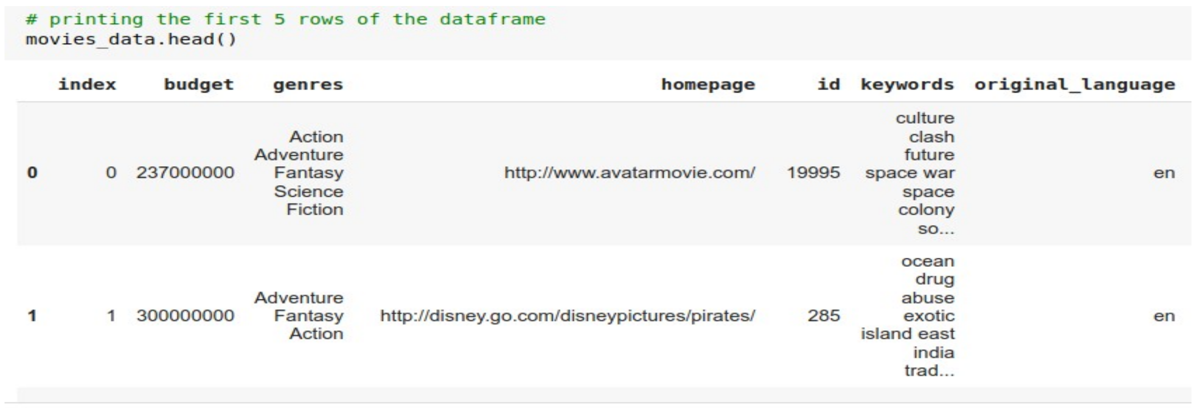
Data Collection and Pre-Processing



```
[ ] # loading the data from the csv file to apandas dataframe
    movies_data = pd.read_csv('/content/movies.csv')
```

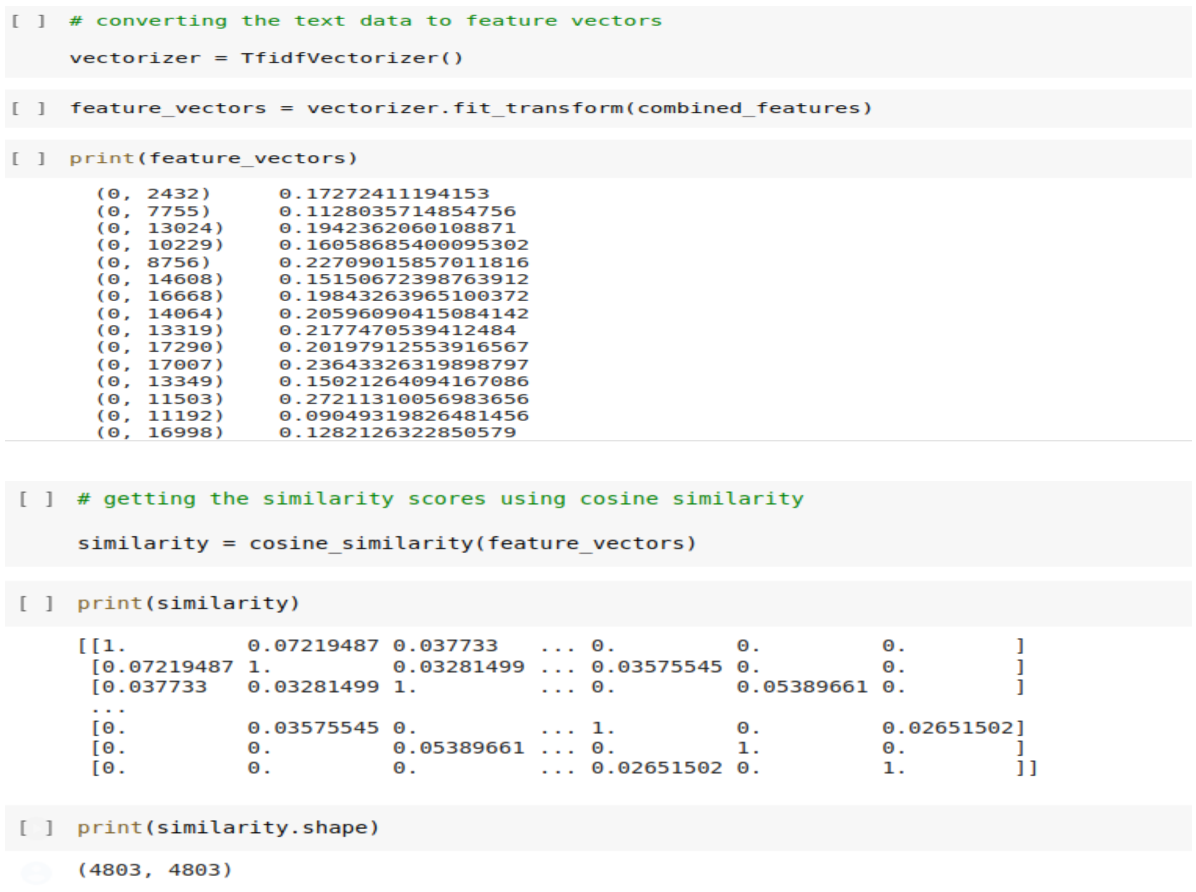


```
# printing the first 5 rows of the dataframe
movies_data.head()
```



```
# printing the first 5 rows of the dataframe
movies_data.head()
```

	index	budget	genres	homepage	id	keywords	original_language
0	0	237000000	Action Adventure Fantasy Science Fiction	http://www.avatarmovie.com/	19995	culture clash future space war space colony so...	en
1	1	300000000	Adventure Fantasy Action	http://disney.go.com/disneypictures/pirates/	285	ocean drug abuse exotic island east india trad...	en



```
[ ]  # converting the text data to feature vectors

     vectorizer = TfidfVectorizer()
```

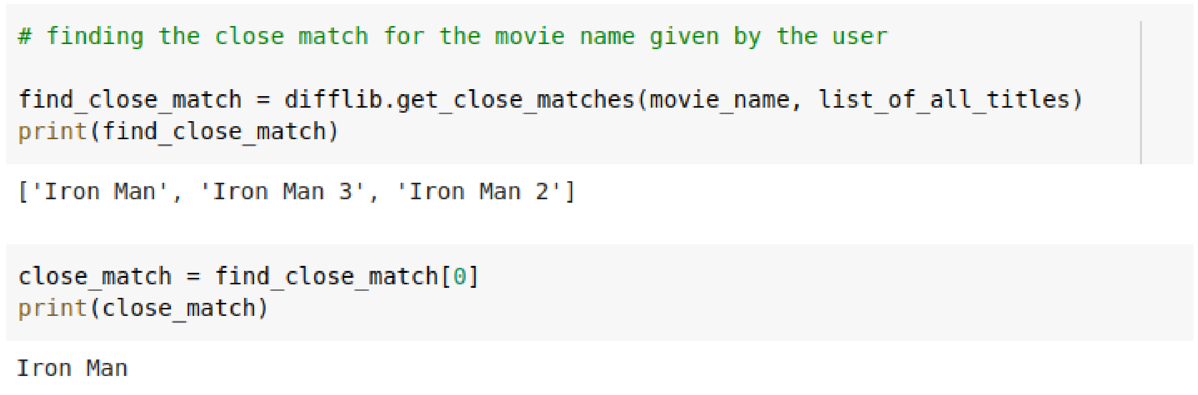
```
[ ]  feature_vectors = vectorizer.fit_transform(combined_features)
```

```
[ ]  print(feature_vectors)

       (0, 2432)      0.17272411194153
       (0, 7755)      0.1128035714854756
       (0, 13024)     0.1942362060108871
       (0, 10229)     0.16058685400095302
       (0, 8756)      0.22709015857011816
       (0, 14608)     0.15150672398763912
       (0, 16668)     0.19843263965100372
       (0, 14064)     0.20596090415084142
       (0, 13319)     0.2177470539412484
       (0, 17290)     0.20197912553916567
       (0, 17007)     0.23643326319898797
       (0, 13349)     0.15021264094167086
       (0, 11503)     0.27211310056983656
       (0, 11192)     0.09049319826481456
       (0, 16998)     0.1282126322850579
```

```
[ ]  # getting the similarity scores using cosine similarity

     similarity = cosine_similarity(feature_vectors)
```

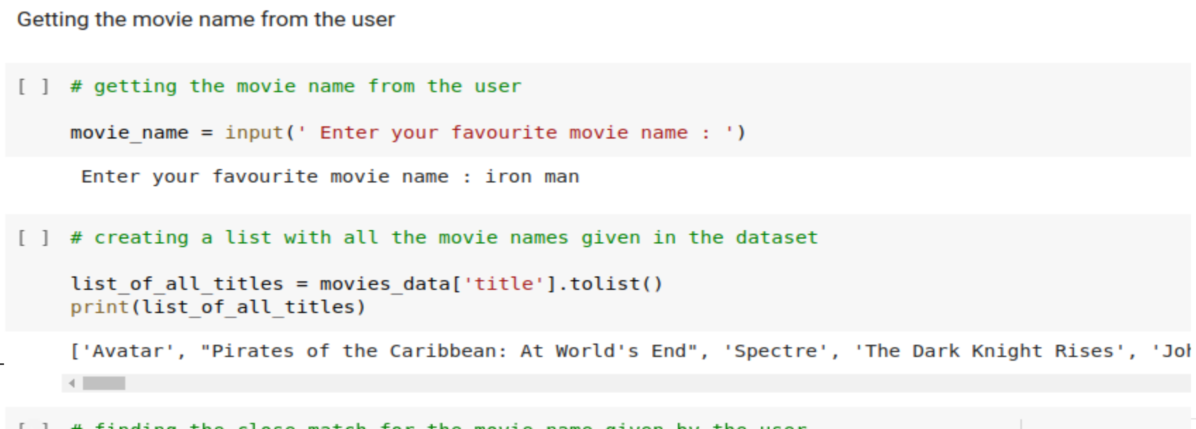
```
[ ]  print(similarity)

     [[1.         0.07219487 0.037733   ... 0.         0.         0.        ]
      [0.07219487 1.         0.03281499 ... 0.03575545 0.         0.        ]
      [0.037733   0.03281499 1.         ... 0.         0.05389661 0.        ]
      ...
      [0.         0.03575545 0.         ... 1.         0.         0.02651502]
      [0.         0.         0.05389661 ... 0.         1.         0.        ]
      [0.         0.         0.         ... 0.02651502 0.         1.        ]]
```

```
[ ]  print(similarity.shape)

     (4803, 4803)
```



```
# finding the close match for the movie name given by the user

find_close_match = difflib.get_close_matches(movie_name, list_of_all_titles)
print(find_close_match)

['Iron Man', 'Iron Man 3', 'Iron Man 2']
```

```
close_match = find_close_match[0]
print(close_match)

Iron Man
```

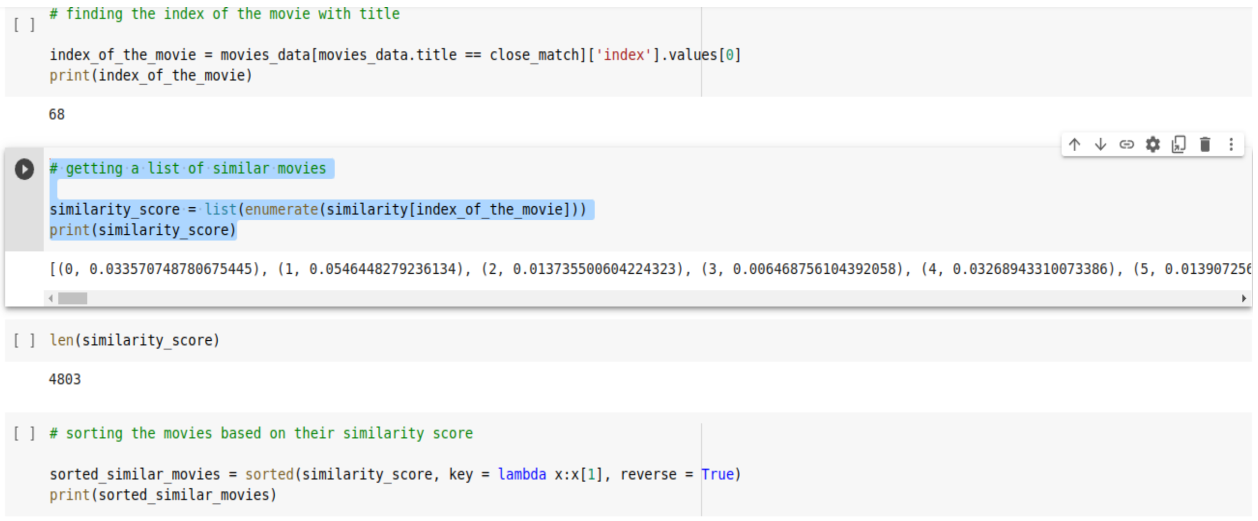


Getting the movie name from the user

```
[ ]  # getting the movie name from the user

     movie_name = input(' Enter your favourite movie name : ')

      Enter your favourite movie name : iron man
```

```
[ ]  # creating a list with all the movie names given in the dataset

     list_of_all_titles = movies_data['title'].tolist()
     print(list_of_all_titles)

     ['Avatar', "Pirates of the Caribbean: At World's End", 'Spectre', 'The Dark Knight Rises', 'Jol
```

```
[ ]  # finding the close match for the movie name given by the user
```



```
# finding the index of the movie with title

index_of_the_movie = movies_data[movies_data.title == close_match]['index'].values[0]
print(index_of_the_movie)
```

```
68
```

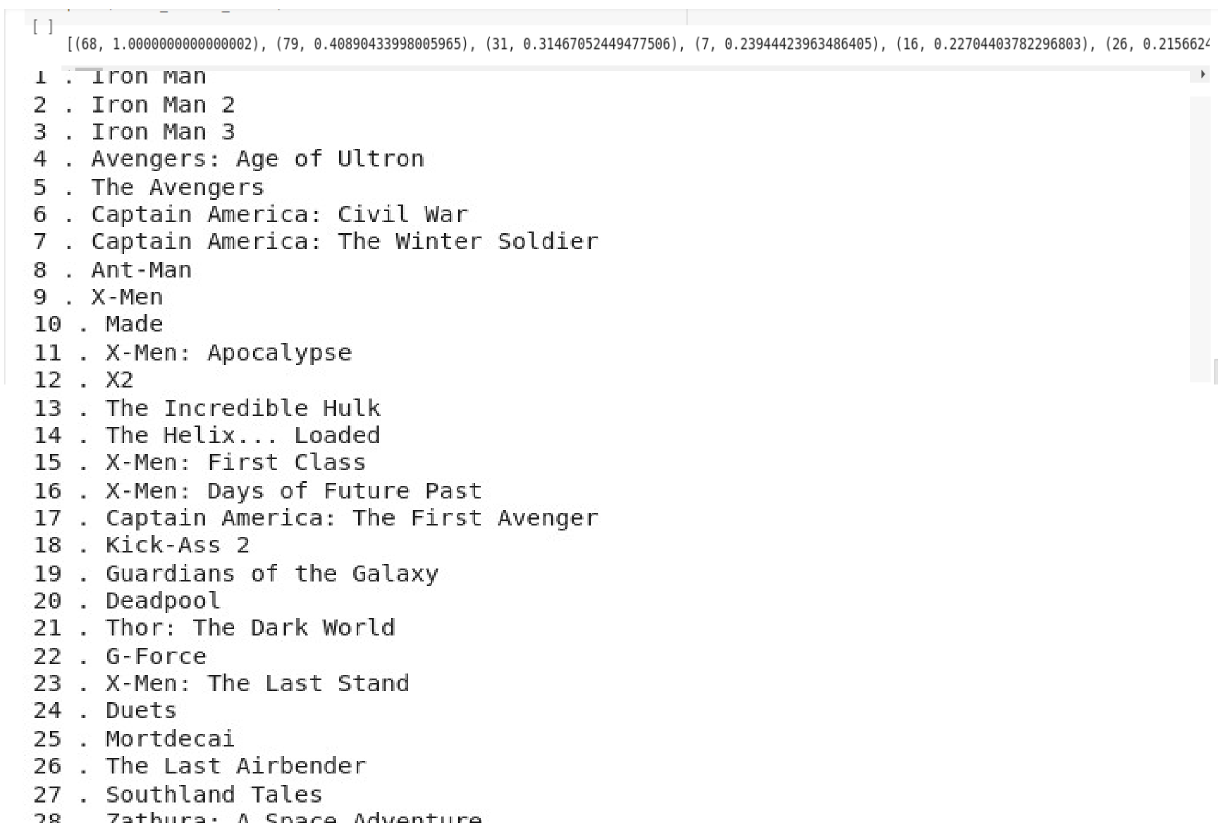
```
# getting a list of similar movies

similarity_score = list(enumerate(similarity[index_of_the_movie]))
print(similarity_score)
```

```
[(0, 0.033570748780675445), (1, 0.0546448279236134), (2, 0.013735500604224323), (3, 0.006468756104392058), (4, 0.03268943310073386), (5, 0.01390725
```

```
len(similarity_score)
```

```
4803
```

```
# sorting the movies based on their similarity score

sorted_similar_movies = sorted(similarity_score, key = lambda x:x[1], reverse = True)
print(sorted_similar_movies)
```



```
[(68, 1.0000000000000002), (79, 0.40890433998005965), (31, 0.31467052449477506), (7, 0.23944423963486405), (16, 0.22704403782296803), (26, 0.215662
```

```
1 . Iron Man
2 . Iron Man 2
3 . Iron Man 3
4 . Avengers: Age of Ultron
5 . The Avengers
6 . Captain America: Civil War
7 . Captain America: The Winter Soldier
8 . Ant-Man
9 . X-Men
10 . Made
11 . X-Men: Apocalypse
12 . X2
13 . The Incredible Hulk
14 . The Helix... Loaded
15 . X-Men: First Class
16 . X-Men: Days of Future Past
17 . Captain America: The First Avenger
18 . Kick-Ass 2
19 . Guardians of the Galaxy
20 . Deadpool
21 . Thor: The Dark World
22 . G-Force
23 . X-Men: The Last Stand
24 . Duets
25 . Mortdecai
26 . The Last Airbender
27 . Southland Tales
28 . Zathura: A Space Adventure
```

Item-Based:

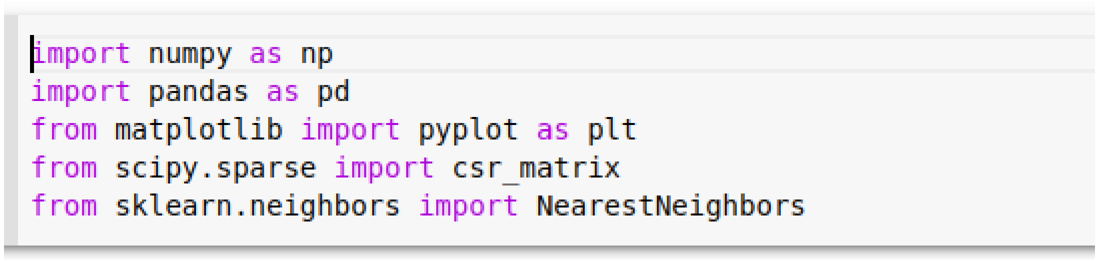
Dataset:

Movie dataset consists of movieId and geners.

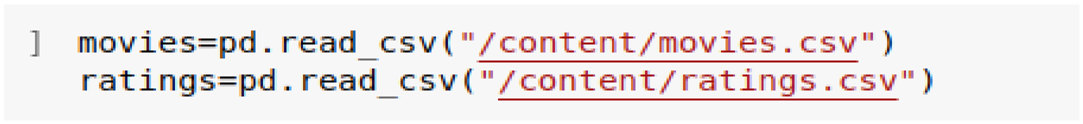
Ratings dataset consists of userId,movieId,Ratings

Implementation:

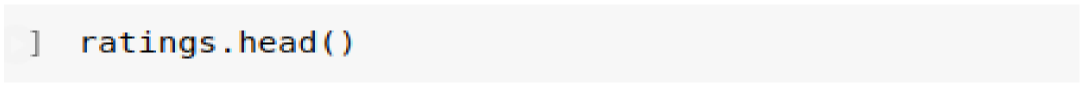
- Make a new dataframe where each column represent each unique userid and each row represents each unique movieId
- Impute NaN with 0 to make things understandable for the algorithm
- we will reduce the noise by adding some filters for the final dataset
- To qualify a movie,a minimum of 10 users should have voted a movie.
- To qualify a user,a minimum of 50 movies should have voted a user.
- To reduce the sparsity we use the csr_matrix fucntion from the scipy library.



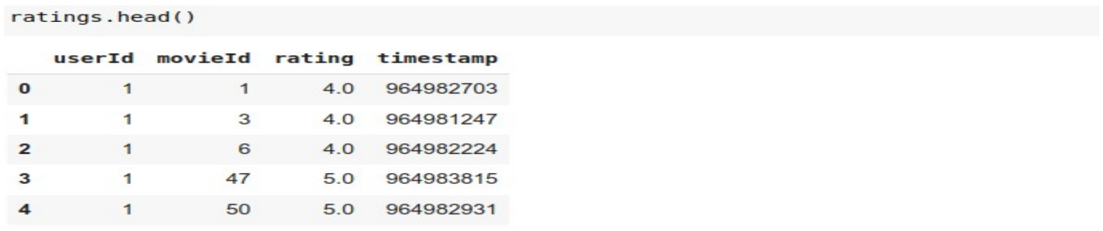
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors
```



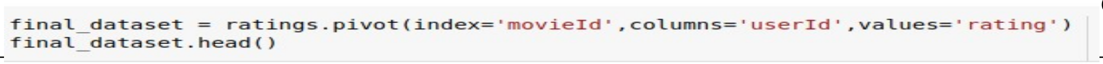
```
]  movies=pd.read_csv("/content/movies.csv")
   ratings=pd.read_csv("/content/ratings.csv")
```



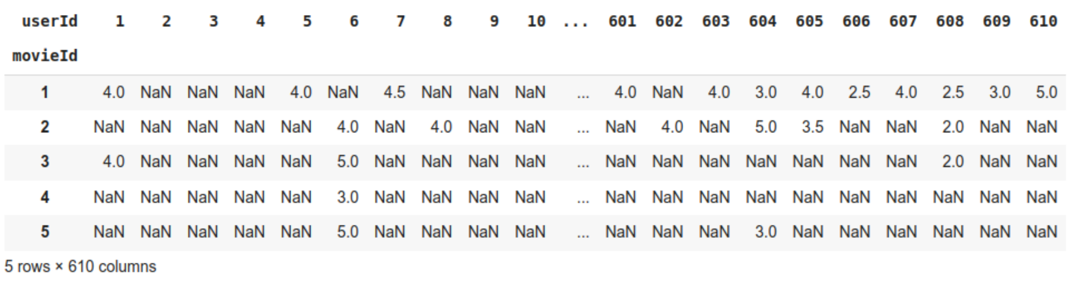
```
]  ratings.head()
```



```
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
```

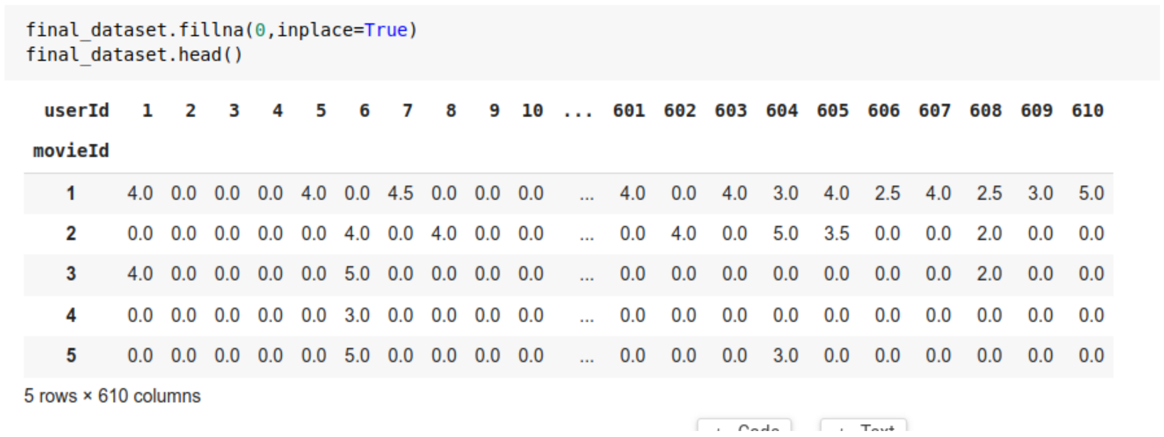


```
final_dataset = ratings.pivot(index='movieId',columns='userId',values='rating')
final_dataset.head()
```



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
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN

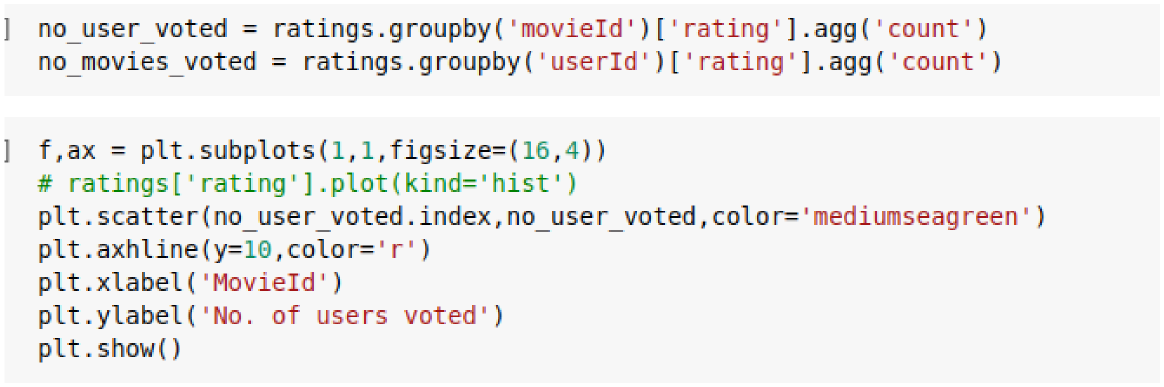
5 rows × 610 columns



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

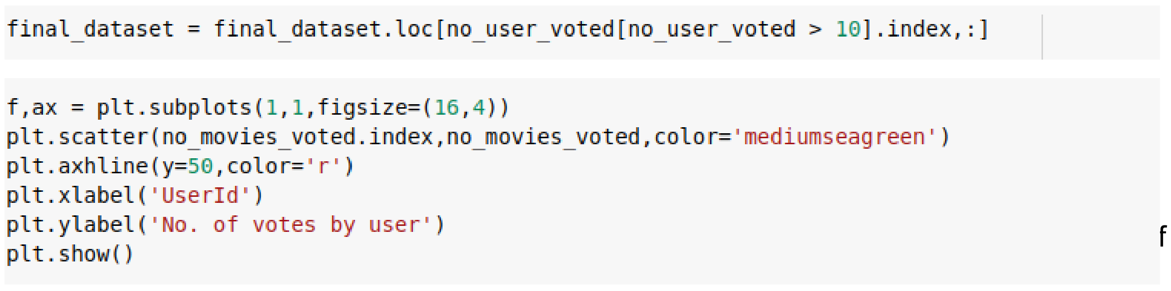
userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	0.0	0.0	0.0	0.0	5.0	0.0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



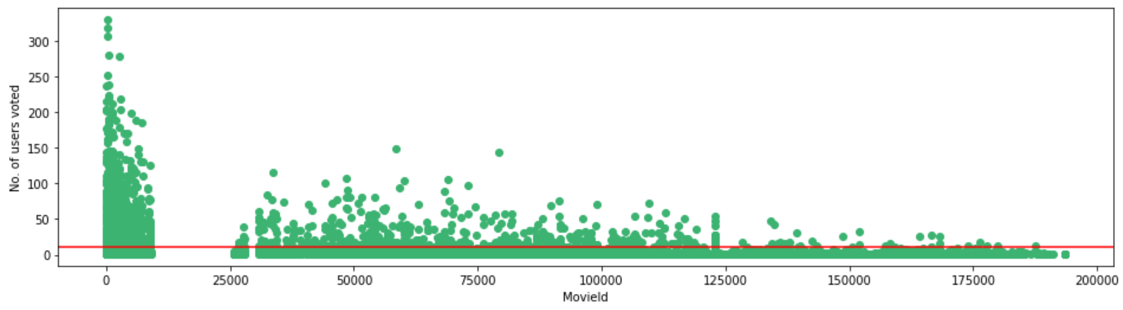
```
]  no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
   no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

```
]  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()
```



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

```
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()
```

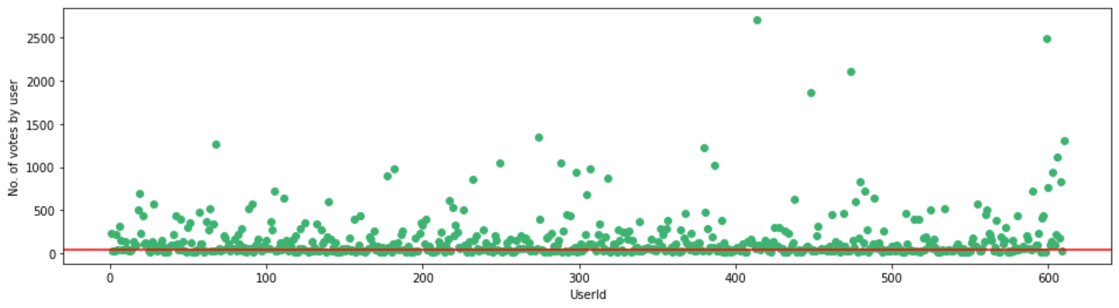


```
] final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

```
] 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')
```

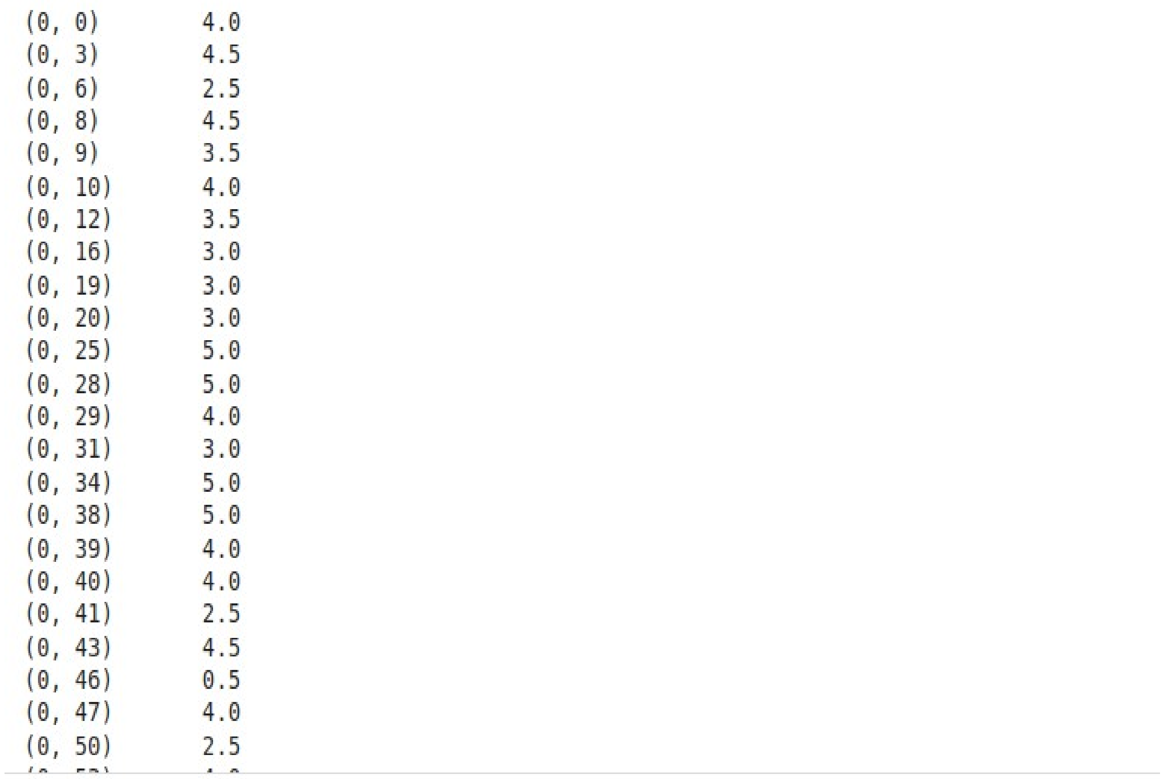
```
final_dataset = final_dataset.loc[no_user_voted[no_user_voted > 10].index,:]
```

```
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()
```

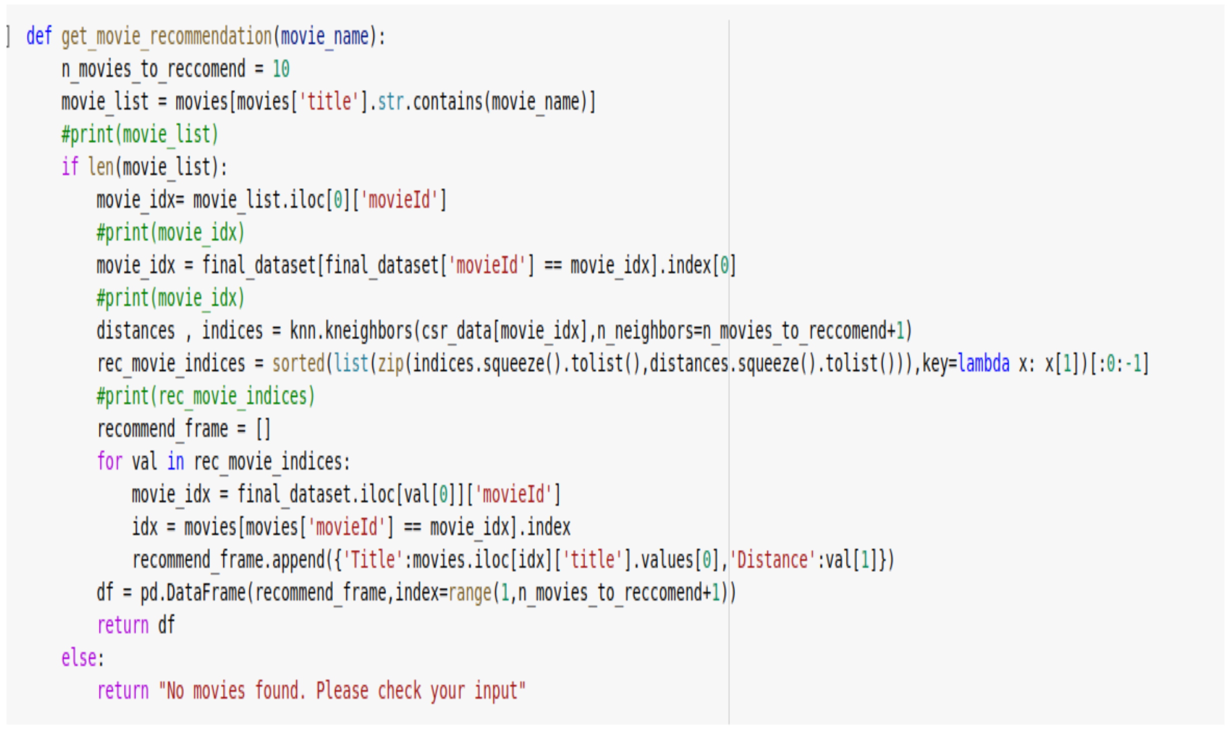


```
final_dataset=final_dataset.loc[:,no_movies_voted[no_movies_voted > 50].index]
final_dataset
```

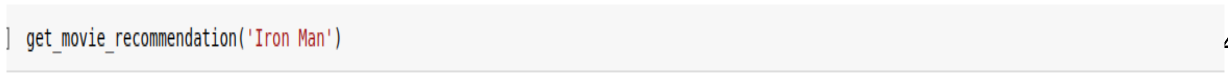
userId	1	4	6	7	10	11	15	16	17	18	...	600	601	602	603	604	605	606	607	608	610
movieId																					
1	4.0	0.0	0.0	4.5	0.0	0.0	2.5	0.0	4.5	3.5	...	2.5	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	5.0
2	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	...	4.0	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0
3	4.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
5	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0	0.0	4.0	...	0.0	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
174055	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	0.0	0.0	0.0	0.0	0.0	0.0
176371	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	0.0	0.0	0.0	0.0	0.0	0.0
177765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0



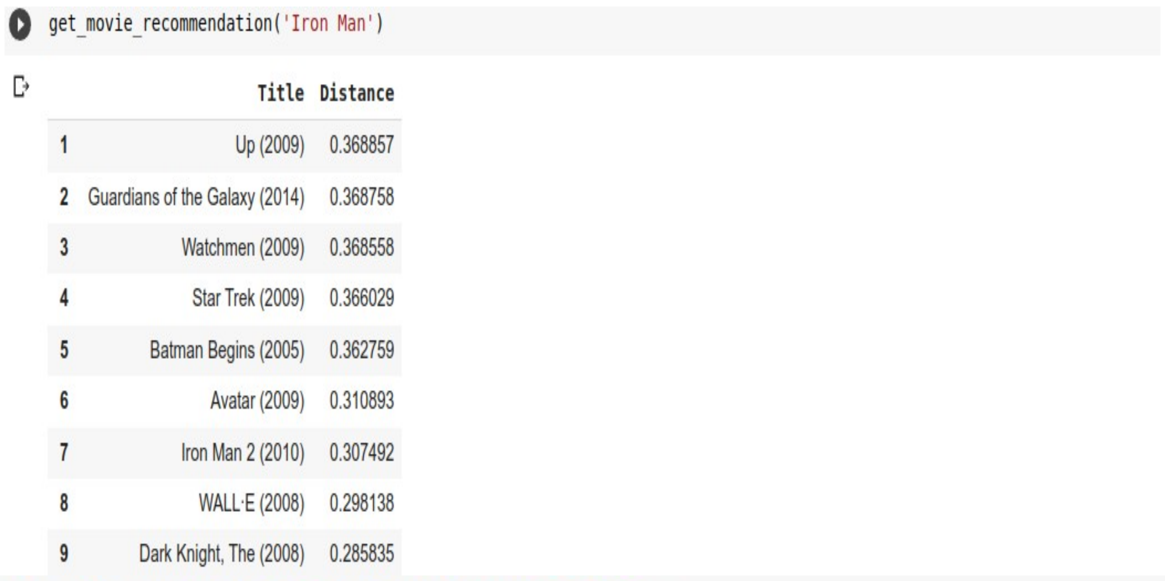
```
(0, 0)        4.0
(0, 3)        4.5
(0, 6)        2.5
(0, 8)        4.5
(0, 9)        3.5
(0, 10)       4.0
(0, 12)       3.5
(0, 16)       3.0
(0, 19)       3.0
(0, 20)       3.0
(0, 25)       5.0
(0, 28)       5.0
(0, 29)       4.0
(0, 31)       3.0
(0, 34)       5.0
(0, 38)       5.0
(0, 39)       4.0
(0, 40)       4.0
(0, 41)       2.5
(0, 43)       4.5
(0, 46)       0.5
(0, 47)       4.0
(0, 50)       2.5
```



```
] def get_movie_recommendation(movie_name):
    n_movies_to_reccomend = 10
    movie_list = movies[movies['title'].str.contains(movie_name)]
    #print(movie_list)
    if len(movie_list):
        movie_idx= movie_list.iloc[0]['movieId']
        #print(movie_idx)
        movie_idx = final_dataset[final_dataset['movieId'] == movie_idx].index[0]
        #print(movie_idx)
        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]
        #print(rec_movie_indices)
        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"
```

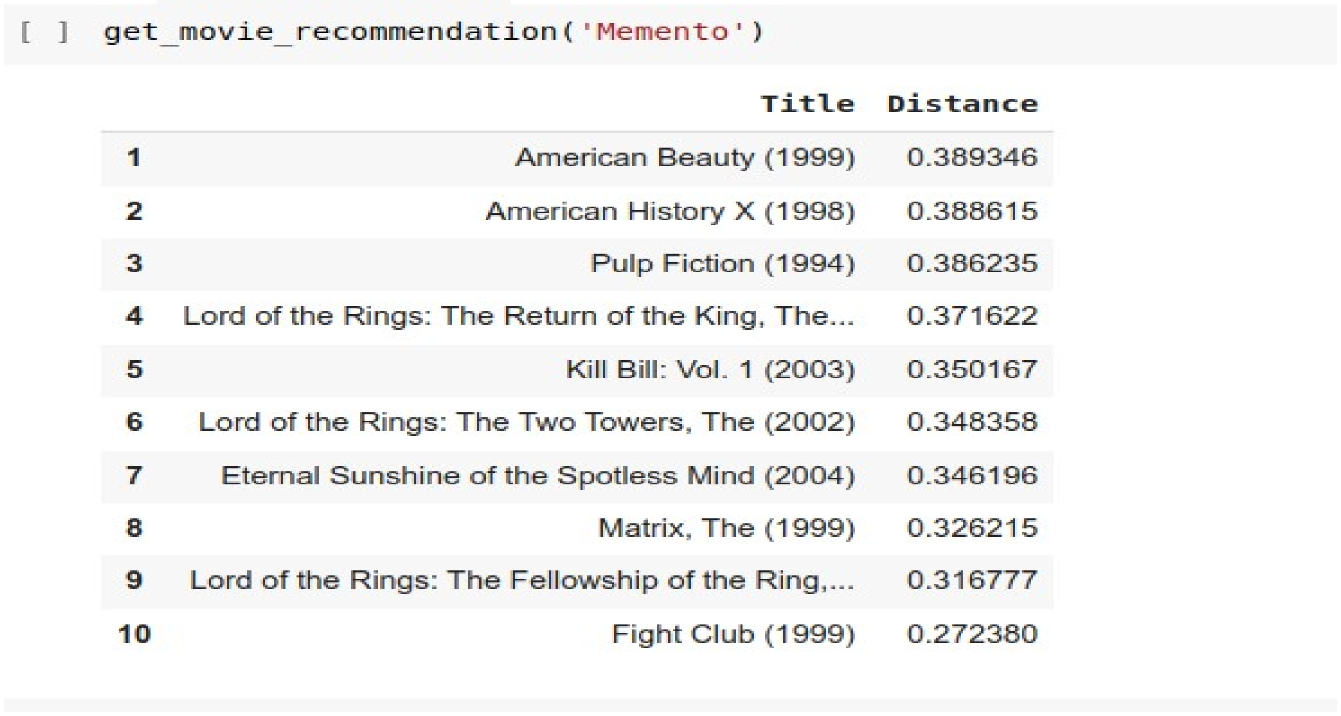


```
] get_movie_recommendation('Iron Man')
```



```
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



```
[ ]  get_movie_recommendation('Memento')
```

	Title	Distance
1	American Beauty (1999)	0.389346
2	American History X (1998)	0.388615
3	Pulp Fiction (1994)	0.386235
4	Lord of the Rings: The Return of the King, The...	0.371622
5	Kill Bill: Vol. 1 (2003)	0.350167
6	Lord of the Rings: The Two Towers, The (2002)	0.348358
7	Eternal Sunshine of the Spotless Mind (2004)	0.346196
8	Matrix, The (1999)	0.326215
9	Lord of the Rings: The Fellowship of the Ring,...	0.316777
10	Fight Club (1999)	0.272380

Results

Through this Movie Recommendation System,We achieved user loyalty by providing revelant content and maximising the time spent by user on our website or app.

Conclusion

In today's world,we can see that most people are using neflix and amazon Netflix,Hotstar, and many other ott platforms.In that, we can see that movies are being recommended to us based on our watch history. Not only these,but you can also observe while watching Instagram reels and youtube shorts.These videos are also recommended based on our watch history. This is where the recommendation System works.And that we had buitlt in this project.

Future Scope

This recommendation system can also be further customized to use is different environments such as e-commerce websites to recommend the products or goods to users which helps in increasing the customer engagement.

References

Reference some documents to get know about recommendation systems followed by Netflix,Youtube etc.

Took suggestions from guide to give best recommendation system to industry.

- https://www.w3schools.com
- https://www.arangodb.com/
- https://www.javatpoint.com/
- https://www.geeksforgeeks.org/