INNOMATICS RESEARCH LABS

INTERNSHIP FINAL PROJECT REPORT – TEAM 2

PROJECT TITLE: RECOMMENDATION SYSTEMS

PROJECT DOMAIN: ENTERTAINMENT

GUIDED BY:

- Kanav Bansal sir (Chief Data Scientist)
- Akhila mam (Mentor)

TEAM MEMBERS:

1. Arsavarthini V
2. Durgesh Bhargava
3. Pavan Yeluri
4. Ramakant
5. Santhosh M

INTRODUCTION:

- With the rise of YouTube, Amazon, Netflix and many other such web services, recommender systems have taken more and more place in our lives.
- From e-commerce (suggest to buyer’s articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys.
- Recommender systems are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors.

What is Recommendation System?

Recommender systems are programs which attempt to predict items (movies, music, books, news) that a user may be interested in, based on user's profile.

Why the Recommendation system?

- Benefits users in finding items of their interest.
- Help item providers in delivering their items to the right user.
- Identity products that are most relevant to users.
- Personalized content.
- Help websites to improve user engagement.

OBJECTIVE:

To build a various recommendation system that can recommend movies to user’s using information like Cast, Genre, Reviews, TMDB/IMDB ratings etc.

TECHNOLOGIES USED:

- Python Programming
- Machine Learning Algorithms - Like KNN, SVD
- NLP concepts like TF -IDF

LIBRARIES USED:

- Numpy
- Pandas
- Json
- Word Cloud
- Matplotlib
- Scikit Learn
- Surprise
- Opendatasets

DATASET OVERVIEW:

The dataset consists of metadata for all 45,000 movies listed in the Full MovieLens Dataset. The dataset consists of movies released on or before July 2017. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts and vote averages.

DATASET DESCRIPTION:

This dataset consists of the following files:

- **movies_metadata.csv:** The main Movies Metadata file. Contains information on 45,000 movies featured in the Full MovieLens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.
- **keywords.csv:** Contains the movie plot keywords for our MovieLens movies. Available in the form of a stringified JSON Object.
- **credits.csv:** Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.
- **links.csv:** The file that contains the TMDB and IMDB IDs of all the movies featured in the Full MovieLens dataset.
- **links_small.csv:** Contains the TMDB and IMDB IDs of a small subset of 9,000 movies of the Full Dataset.
- **ratings_small.csv:** The subset of 100,000 ratings from 700 users on 9,000 movies.

STEPS INVOLVED FOR THE PROJECT:

- Data Cleaning - Data Preprocessing.
- Exploratory Data Analysis.
- Data Visualization.
- Model Building.
  - Popularity Based Recommendation System.
  - Content Based Recommendation Systems.
  - Collaborative Based Recommendation Systems.
- Interpreting the predicted recommendations.

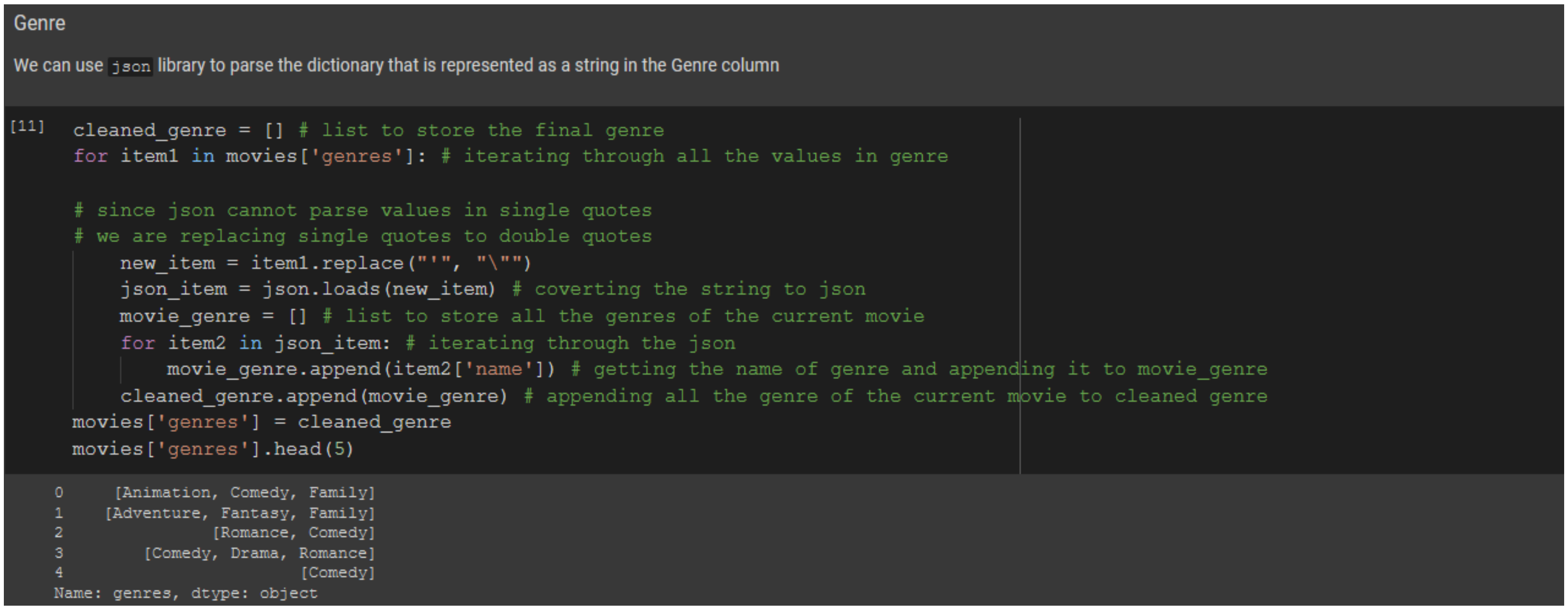
DATA CLEANING AND PREPROCESSING:

Since all the features present in the dataset are not necessary to build the recommendation system, we have used only the important features. The features used to build the recommendation systems are as follows

- Genre.
- Runtime.
- Language.
- Release Year.
- Vote Count.
- Vote Average.
- Weighted Rating. (Derived using Vote Count and Vote Average)

GENRE:

- All the data in the Genres feature is represented in the form of stringified list of dictionaries.
- Example: '[{'id': 16, 'name': 'Animation'}, {'id': 35, 'name': 'Comedy'},{'id': 10751, 'name': 'Family'}]'
- So, we had used the json library to convert it to a list of dictionaries and get all the genres for a movie.



Genre

We can use `json` library to parse the dictionary that is represented as a string in the Genre column

```
[11]  cleaned_genre = [] # list to store the final genre
      for item1 in movies['genres']: # iterating through all the values in genre

      # since json cannot parse values in single quotes
      # we are replacing single quotes to double quotes
          new_item = item1.replace("'", "\"")
          json_item = json.loads(new_item) # coverting the string to json
          movie_genre = [] # list to store all the genres of the current movie
          for item2 in json_item: # iterating through the json
              movie_genre.append(item2['name']) # getting the name of genre and appending it to movie_genre
          cleaned_genre.append(movie_genre) # appending all the genre of the current movie to cleaned genre
      movies['genres'] = cleaned_genre
      movies['genres'].head(5)
```

```
0     [Animation, Comedy, Family]
1    [Adventure, Fantasy, Family]
2               [Romance, Comedy]
3        [Comedy, Drama, Romance]
4                        [Comedy]
Name: genres, dtype: object
```

POPULARITY BASED RECOMMENDATION SYSTEM:

- It is a type of recommendation system which works on the principle of popularity and or anything which is in trend. These systems check about the product or movie which are in trend or are most popular among the users and directly recommend those.
- We had used the weighted ratings feature and genre of the movies to find movies that are popular with respect to each genre and recommend them to the user.

WEIGHTED RATING:

Weighted rating is a formula that provides a true 'Bayesian estimate', which takes into account the number of votes each title has received, minimum votes required to be on the list, and the mean vote for all titles:

weighted rating (WR) → $\left(\frac{v}{v+m}.R\right) + \left(\frac{m}{v+m}.C\right)$

Where:

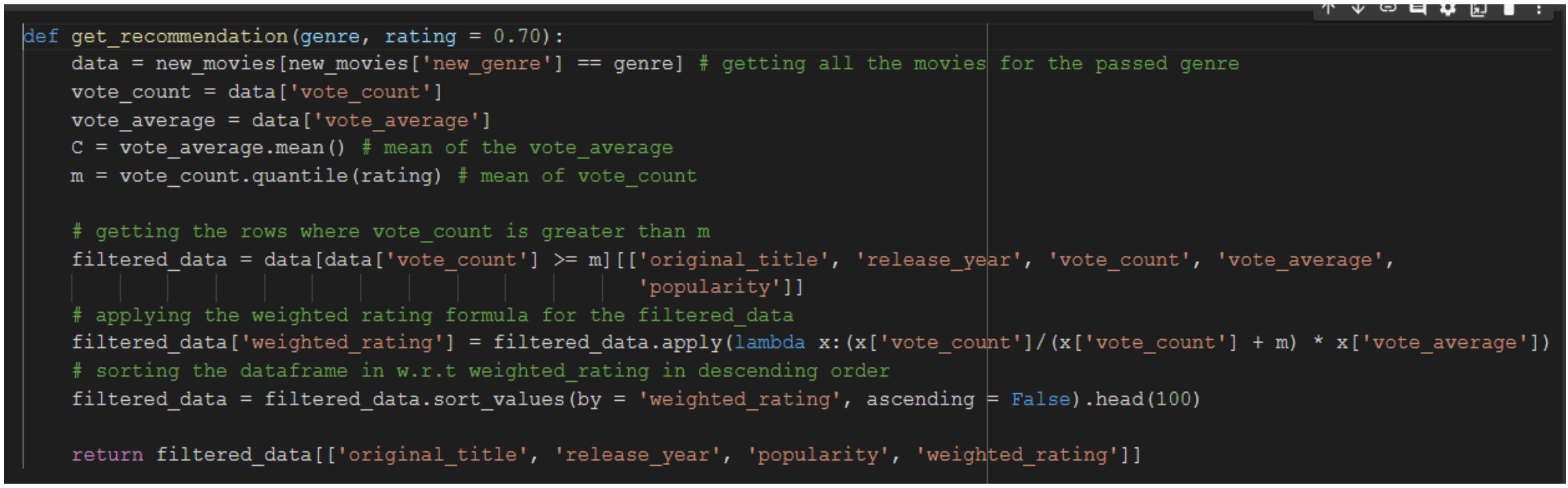
R = average for the movie (mean) = (rating)

v = number of votes for the movie = (votes)

m = minimum votes required to be listed in the Top Rated 250 list (95th percentile value of vote_count column)

C = the mean vote across the whole report (mean of vote_average column)

This is the function that is used to implement Popularity Based Recommendation.

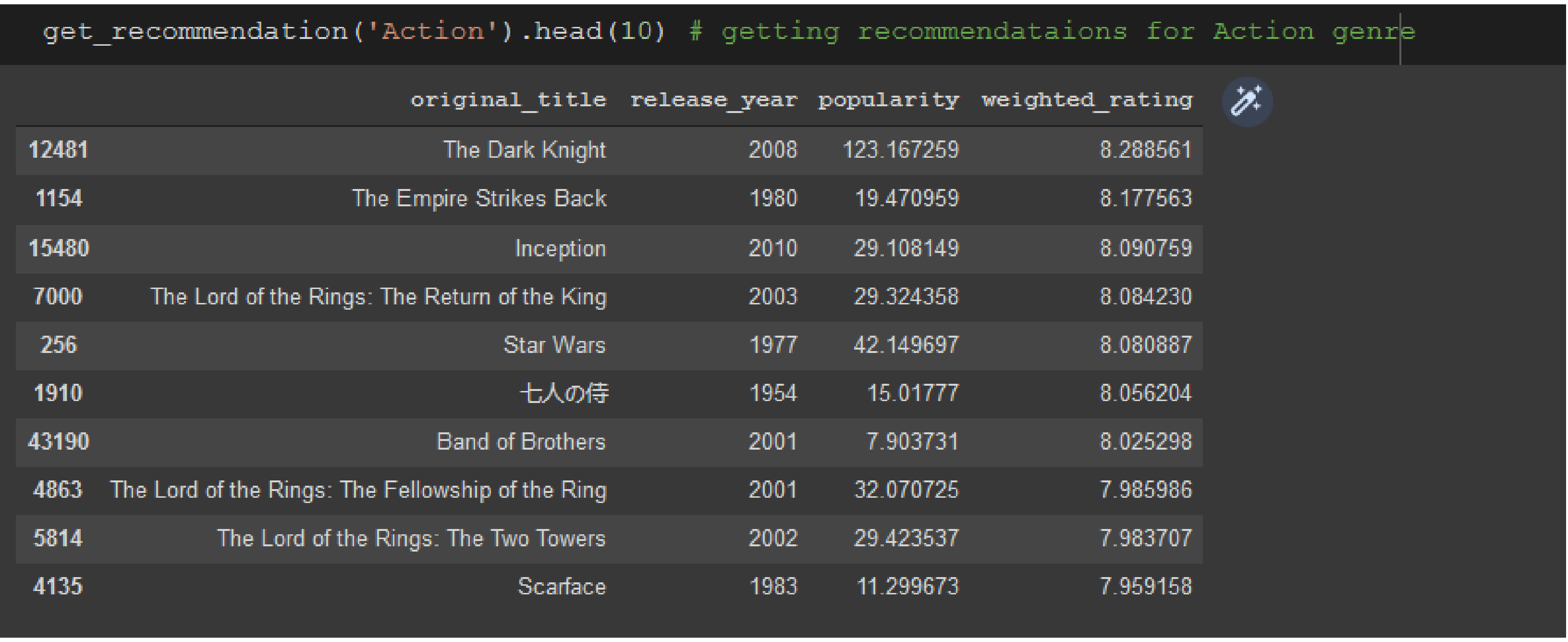


```python
def get_recommendation(genre, rating = 0.70):
    data = new_movies[new_movies['new_genre'] == genre] # getting all the movies for the passed genre
    vote_count = data['vote_count']
    vote_average = data['vote_average']
    C = vote_average.mean() # mean of the vote_average
    m = vote_count.quantile(rating) # mean of vote_count

    # getting the rows where vote_count is greater than m
    filtered_data = data[data['vote_count'] >= m][['original_title', 'release_year', 'vote_count', 'vote_average',
                                                    'popularity']]
    # applying the weighted rating formula for the filtered_data
    filtered_data['weighted_rating'] = filtered_data.apply(lambda x:(x['vote_count']/(x['vote_count'] + m) * x['vote_average'])
    # sorting the dataframe in w.r.t weighted_rating in descending order
    filtered_data = filtered_data.sort_values(by = 'weighted_rating', ascending = False).head(100)

    return filtered_data[['original_title', 'release_year', 'popularity', 'weighted_rating']]
```

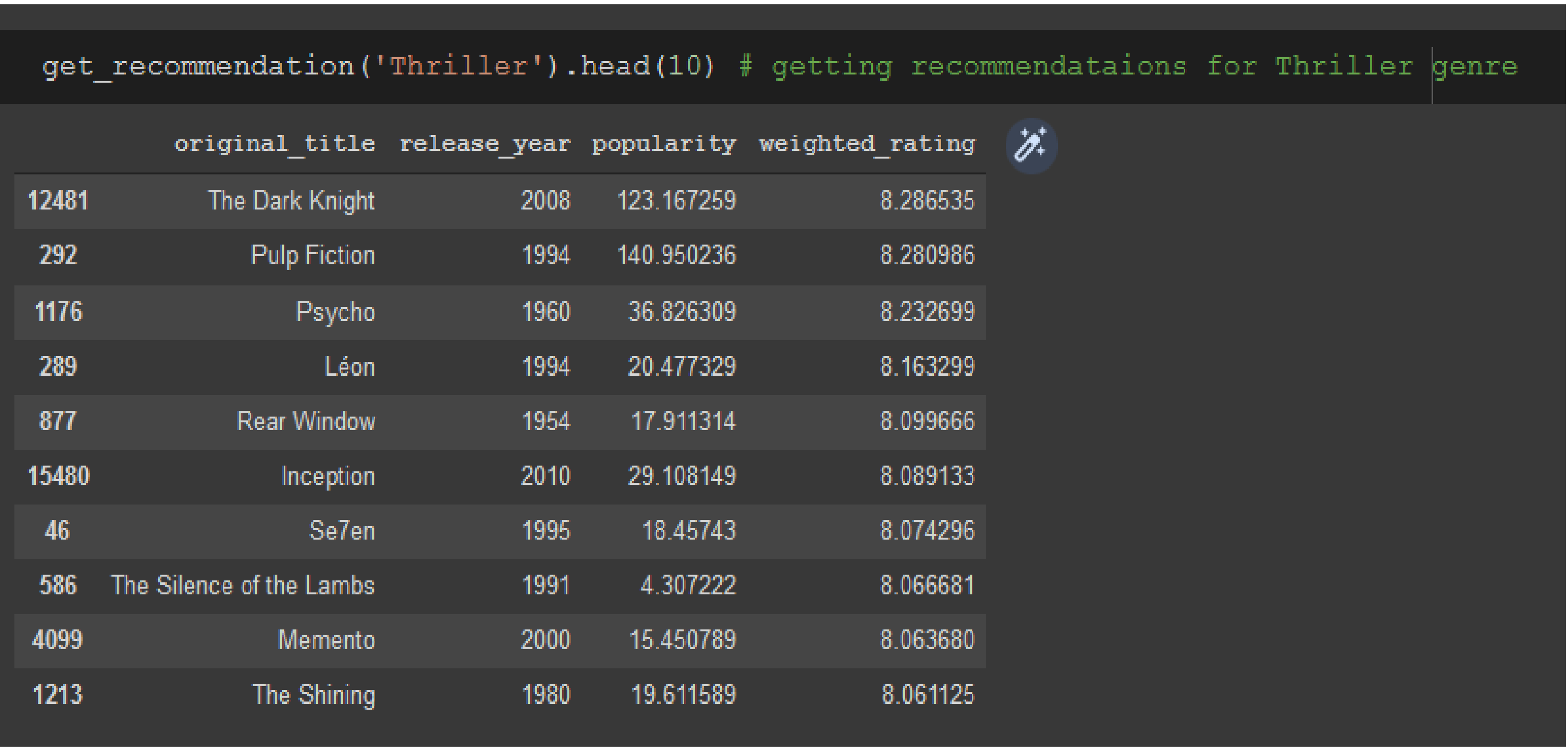
- Getting recommendation on **Action** genre using Popularity Based Recommendation.



```
get_recommendation('Action').head(10) # getting recommendataions for Action genre
```

	original_title	release_year	popularity	weighted_rating
12481	The Dark Knight	2008	123.167259	8.288561
1154	The Empire Strikes Back	1980	19.470959	8.177563
15480	Inception	2010	29.108149	8.090759
7000	The Lord of the Rings: The Return of the King	2003	29.324358	8.084230
256	Star Wars	1977	42.149697	8.080887
1910	七人の侍	1954	15.01777	8.056204
43190	Band of Brothers	2001	7.903731	8.025298
4863	The Lord of the Rings: The Fellowship of the Ring	2001	32.070725	7.985986
5814	The Lord of the Rings: The Two Towers	2002	29.423537	7.983707
4135	Scarface	1983	11.299673	7.959158

- Getting recommendation on **Thriller** genre using Popularity Based Recommendation.



```
get_recommendation('Thriller').head(10) # getting recommendataions for Thriller genre
```

	original_title	release_year	popularity	weighted_rating
12481	The Dark Knight	2008	123.167259	8.286535
292	Pulp Fiction	1994	140.950236	8.280986
1176	Psycho	1960	36.826309	8.232699
289	Léon	1994	20.477329	8.163299
877	Rear Window	1954	17.911314	8.099666
15480	Inception	2010	29.108149	8.089133
46	Se7en	1995	18.45743	8.074296
586	The Silence of the Lambs	1991	4.307222	8.066681
4099	Memento	2000	15.450789	8.063680
1213	The Shining	1980	19.611589	8.061125

CONTENT BASED RECOMMENDATION SYSTEM:

- In this type of recommendation system, relevant items are shown using the content of the previously searched items by the users.
- Here content refers to the attribute/tag of the product that the user like.
- In this type of system, products are tagged using certain keywords, then the system tries to understand what the user wants and it looks in its database and finally tries to recommend different products that the user wants.
- To get tags related to each movie we have created a new feature called "tagline" by concatenating the description and overview features of each movie.
- We have used the concepts of Tf-Idf vectorizer and cosine similarity to build the content-based recommendation systems.

TF-IDF VECTORIZER:

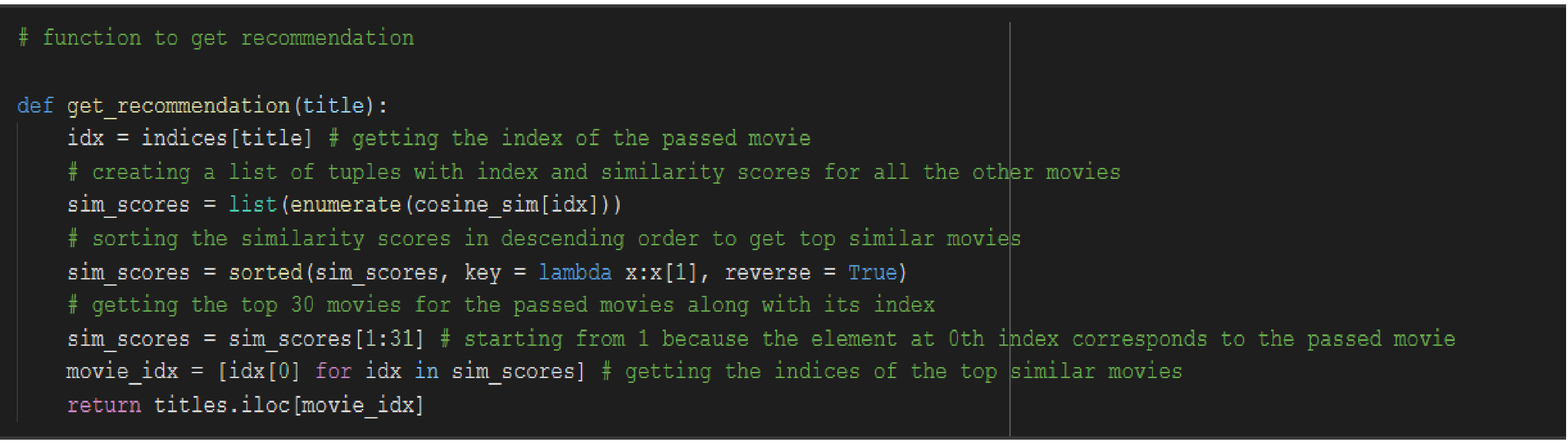
Tf-Idf stands for Term Frequency - Inverse Document Frequency**.** It is a 2-dimensional data matrix where each term denotes the relative frequency of a particular word in a particular document as compared to other documents. This is a widely used metric and is used in Text Mining and Information retrieval.

COSINE SIMILARITY:

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them.

$$cosine(x, y) = \frac{x . y^{\intercal}}{||x||.||y||}$$

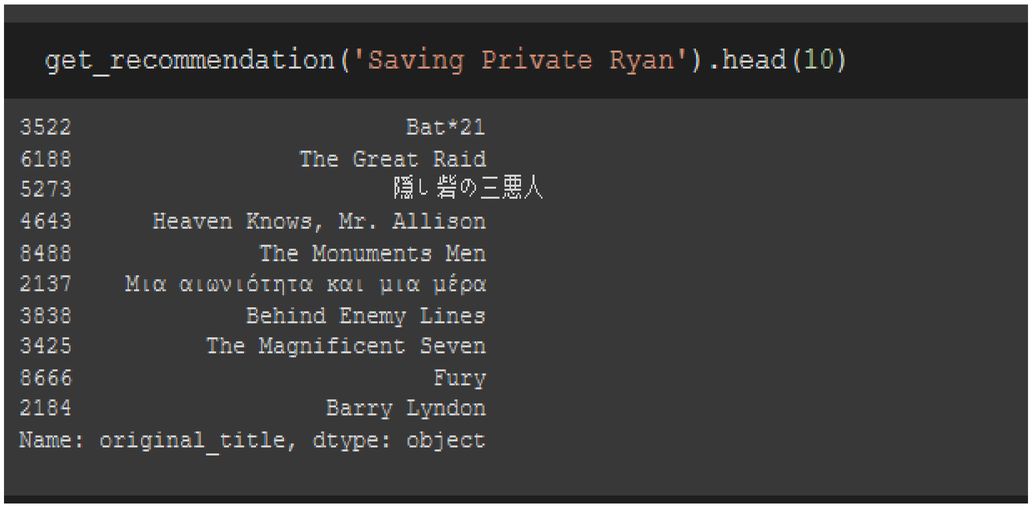
- Function to perform Content Based Recommendation



```python
# function to get recommendation

def get_recommendation(title):
    idx = indices[title] # getting the index of the passed movie
    # creating a list of tuples with index and similarity scores for all the other movies
    sim_scores = list(enumerate(cosine_sim[idx]))
    # sorting the similarity scores in descending order to get top similar movies
    sim_scores = sorted(sim_scores, key = lambda x:x[1], reverse = True)
    # getting the top 30 movies for the passed movies along with its index
    sim_scores = sim_scores[1:31] # starting from 1 because the element at 0th index corresponds to the passed movie
    movie_idx = [idx[0] for idx in sim_scores] # getting the indices of the top similar movies
    return titles.iloc[movie_idx]
```

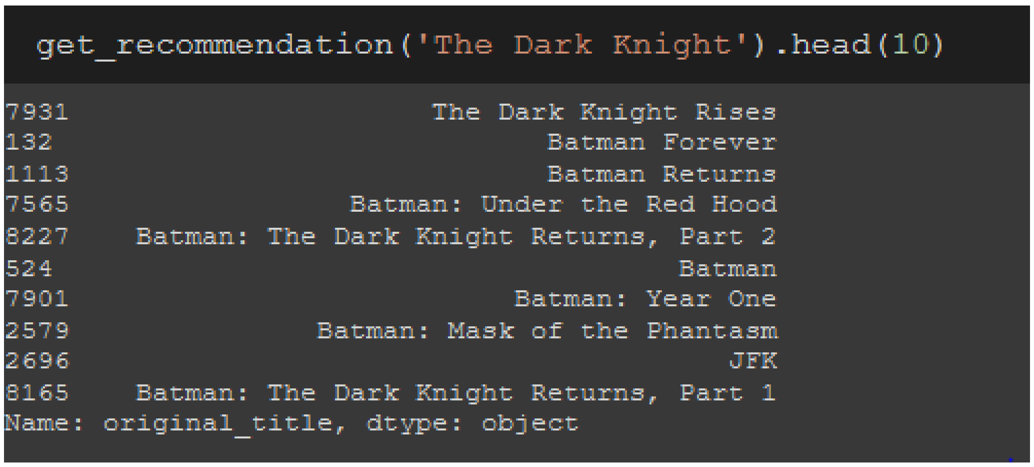
- Getting recommendations for Saving Private Ryan



```
get_recommendation('Saving Private Ryan').head(10)
```

```
3522                          Bat*21
6188                  The Great Raid
5273                          隠し砦の三悪人
4643        Heaven Knows, Mr. Allison
8488                The Monuments Men
2137      Μια αιωνιότητα και μια μέρα
3838              Behind Enemy Lines
3425           The Magnificent Seven
8666                            Fury
2184                    Barry Lyndon
Name: original_title, dtype: object
```

- Getting recommendations for The Dark Knight



```
get_recommendation('The Dark Knight').head(10)
```

```
7931                        The Dark Knight Rises
132                                Batman Forever
1113                               Batman Returns
7565                    Batman: Under the Red Hood
8227    Batman: The Dark Knight Returns, Part 2
524                                        Batman
7901                            Batman: Year One
2579                 Batman: Mask of the Phantasm
2696                                          JFK
8165    Batman: The Dark Knight Returns, Part 1
Name: original_title, dtype: object
```

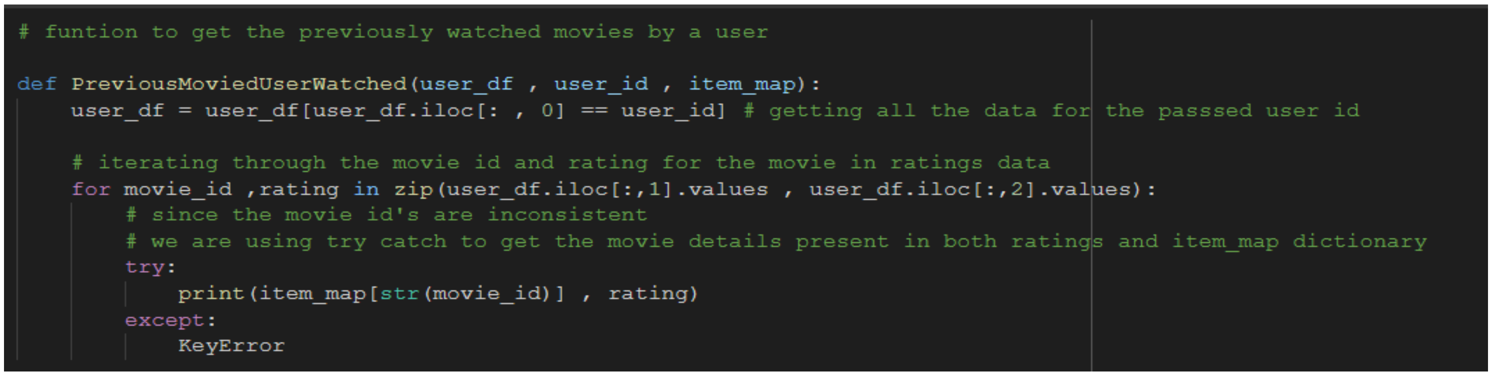
COLLABORATIVE BASED RECOMMENDATION SYSTEMS:

- Recommending the new items to users based on the interest and preference of other similar users is basically collaborative-based filtering.
- For example: - When we shop on Amazon it recommends new products saying *“Customer who brought this also brought”*.
- This overcomes the disadvantage of content-based filtering as it will use the user interaction instead of content from the items used by the users.
- For this, it only needs the historical performance of the users. Based on the historical data, with the assumption that user who has agreed in past tends to also agree in future.
- There are 2 types of collaborative based filtering.
    - User-Based Collaborative Filtering
    - Item-Based Collaborative Filtering

USER BASED COLLABORATIVE FILTERING:

- Rating of the item is done using the rating of neighboring users. In simple words, it is based on the notion of users’ similarity.
- Since collaborative filtering focuses on the interaction of the user, we need to get the movies that was watched by a user before making recommendations.
- So, we have written a function that gives the movies that were previously watched by the user.
- Using that we can interpret the recommendations that are given for a user.

- Function to get previously watched movies by the user.

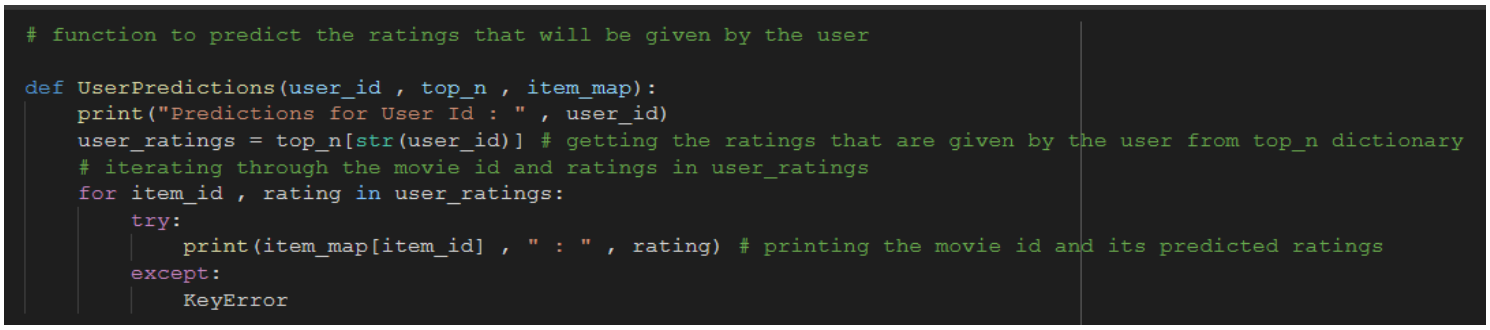


```
# funtion to get the previously watched movies by a user

def PreviousMoviedUserWatched(user_df , user_id , item_map):
    user_df = user_df[user_df.iloc[: , 0] == user_id] # getting all the data for the passsed user id

    # iterating through the movie id and rating for the movie in ratings data
    for movie_id ,rating in zip(user_df.iloc[:,1].values , user_df.iloc[:,2].values):
        # since the movie id's are inconsistent
        # we are using try catch to get the movie details present in both ratings and item_map dictionary
        try:
            print(item_map[str(movie_id)] , rating)
        except:
            KeyError
```

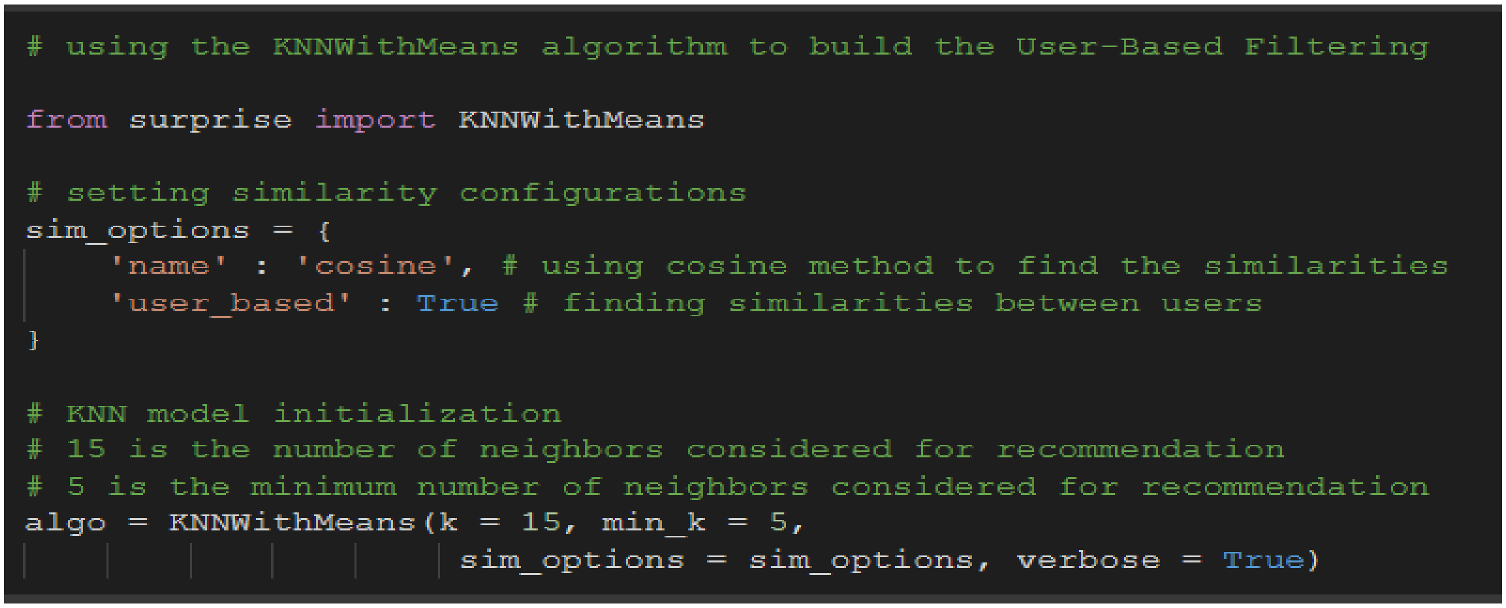
- Function to get the recommendation for the user.



```
# function to predict the ratings that will be given by the user

def UserPredictions(user_id , top_n , item_map):
    print("Predictions for User Id : " , user_id)
    user_ratings = top_n[str(user_id)] # getting the ratings that are given by the user from top_n dictionary
    # iterating through the movie id and ratings in user_ratings
    for item_id , rating in user_ratings:
        try:
            print(item_map[item_id] , " : " , rating) # printing the movie id and its predicted ratings
        except:
            KeyError
```

- The algorithm used for building the user-based recommendation system is KNNWithMeans.



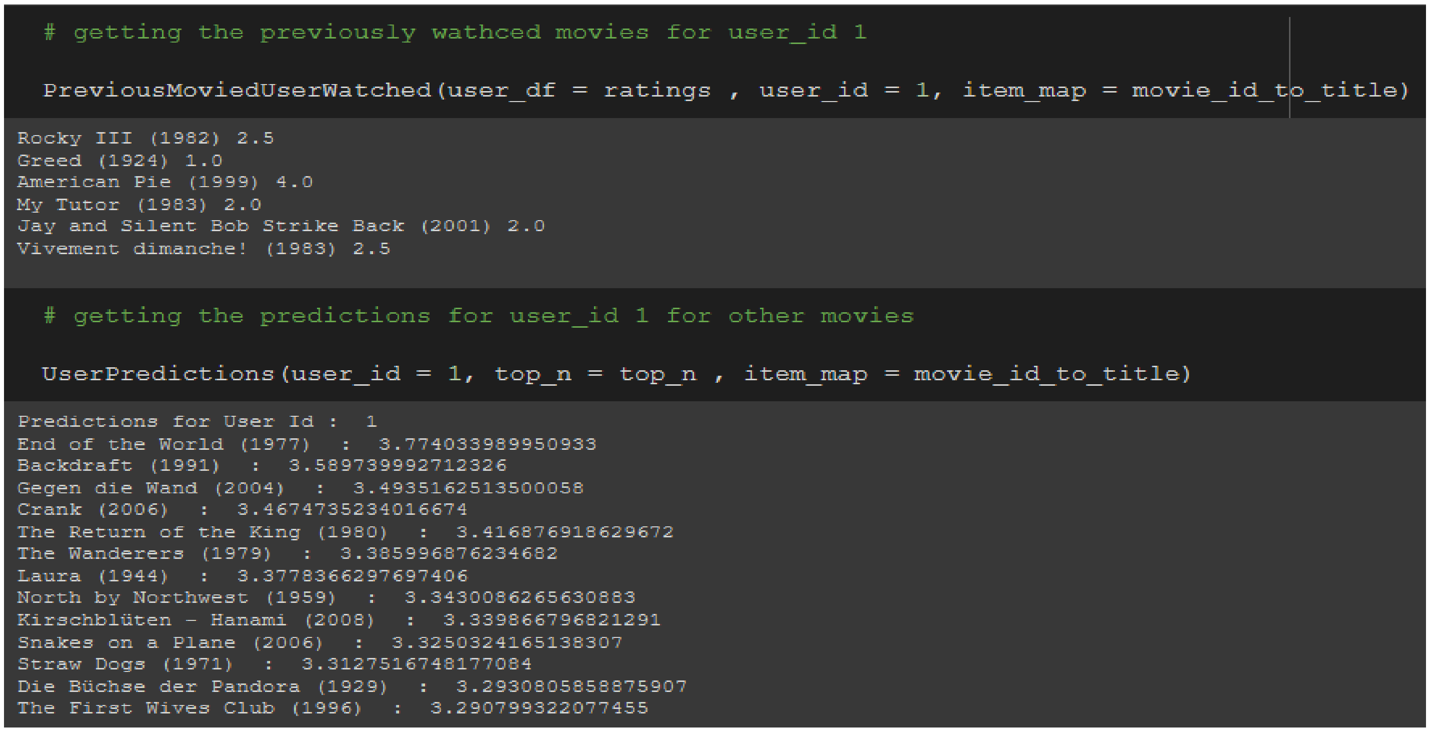
```
# using the KNNWithMeans algorithm to build the User-Based Filtering

from surprise import KNNWithMeans

# setting similarity configurations
sim_options = {
    'name' : 'cosine', # using cosine method to find the similarities
    'user_based' : True # finding similarities between users
}

# KNN model initialization
# 15 is the number of neighbors considered for recommendation
# 5 is the minimum number of neighbors considered for recommendation
algo = KNNWithMeans(k = 15, min_k = 5,
                    sim_options = sim_options, verbose = True)
```

- Here we can see the previously watched movies by user 1 and their recommendations.



```
# getting the previously wathced movies for user_id 1

PreviousMoviedUserWatched(user_df = ratings , user_id = 1, item_map = movie_id_to_title)
```

```
Rocky III (1982) 2.5
Greed (1924) 1.0
American Pie (1999) 4.0
My Tutor (1983) 2.0
Jay and Silent Bob Strike Back (2001) 2.0
Vivement dimanche! (1983) 2.5
```

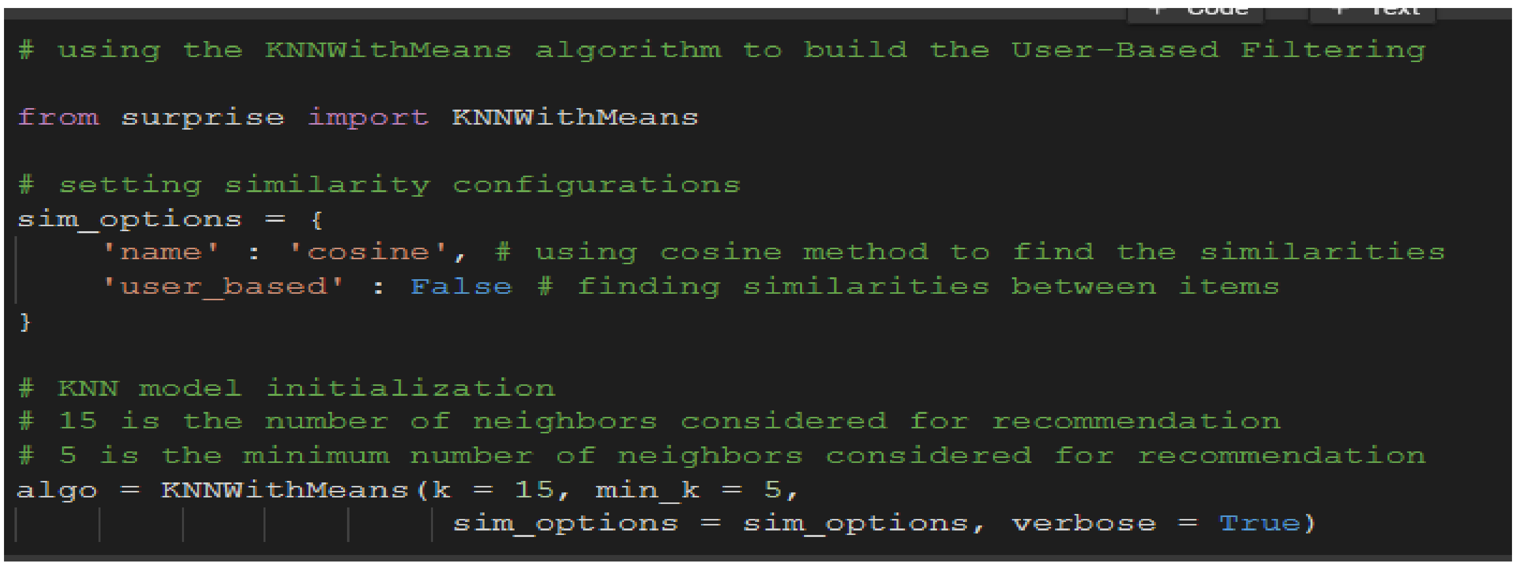
```
# getting the predictions for user_id 1 for other movies

UserPredictions(user_id = 1, top_n = top_n , item_map = movie_id_to_title)
```

```
Predictions for User Id :  1
End of the World (1977)  :  3.774033989950933
Backdraft (1991)  :  3.589739992712326
Gegen die Wand (2004)  :  3.4935162513500058
Crank (2006)  :  3.4674735234016674
The Return of the King (1980)  :  3.416876918629672
The Wanderers (1979)  :  3.385996876234682
Laura (1944)  :  3.3778366297697406
North by Northwest (1959)  :  3.3430086265630883
Kirschblüten - Hanami (2008)  :  3.339866796821291
Snakes on a Plane (2006)  :  3.3250324165138307
Straw Dogs (1971)  :  3.3127516748177084
Die Büchse der Pandora (1929)  :  3.2930805858875907
The First Wives Club (1996)  :  3.290799322077455
```

ITEM BASED RECOMMENDATIONS SYSTEM:

- The rating of the item is predicted using the user’s own rating on neighboring items, in simple words it is based on the notion of item similarity.
- Here also we have used the KNNWithMeans algorithm to implement item-based recommendation.



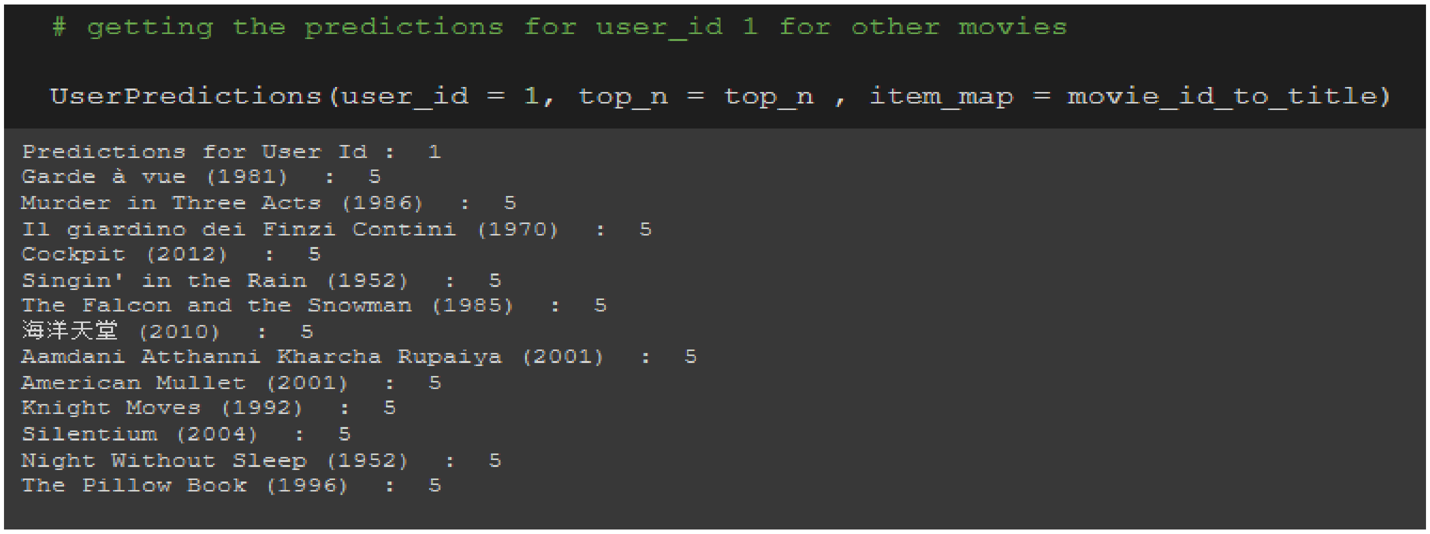
```
# using the KNNWithMeans algorithm to build the User-Based Filtering

from surprise import KNNWithMeans

# setting similarity configurations
sim_options = {
    'name' : 'cosine', # using cosine method to find the similarities
    'user_based' : False # finding similarities between items
}

# KNN model initialization
# 15 is the number of neighbors considered for recommendation
# 5 is the minimum number of neighbors considered for recommendation
algo = KNNWithMeans(k = 15, min_k = 5,
                    sim_options = sim_options, verbose = True)
```

- Getting predictions for user 1 based on item-based filtering.



```
# getting the predictions for user_id 1 for other movies

UserPredictions(user_id = 1, top_n = top_n , item_map = movie_id_to_title)
```

```
Predictions for User Id :  1
Garde à vue (1981)  :  5
Murder in Three Acts (1986)  :  5
Il giardino dei Finzi Contini (1970)  :  5
Cockpit (2012)  :  5
Singin' in the Rain (1952)  :  5
The Falcon and the Snowman (1985)  :  5
海洋天堂 (2010)  :  5
Aamdani Atthanni Kharcha Rupaiya (2001)  :  5
American Mullet (2001)  :  5
Knight Moves (1992)  :  5
Silentium (2004)  :  5
Night Without Sleep (1952)  :  5
The Pillow Book (1996)  :  5
```

FUTURE SCOPE:

- We will look into improving our model to deep reinforcement-based recommendation system. (https://ieeexplore.ieee.org/document/8350761/metrics#metrics)
- We can also develop the system in a way that the movie database is updated frequently so the system can also recommend new movies.
- We can also use the methodology used for recommendation for other entertainment apps like YouTube, Spotify etc.