AUGUST 5, 2021

FINAL PROJECT-WRITE UP

Movies Recommender System

ANWESH PRAHARAJ
STA 6704

Table of Contents

1. Introduction

Here in this project recommended system build to give move recommendation to users. I have used Movie-lens dataset for analysis and prediction. Both collaborative and content-based filtering technics are used in this project.

2. Dataset

I have used Movie-lens dataset for analysis purpose. There are lot of data available but in our case I used u.data and u.item .

U.data dataset contain user_id , movie_id and the rating provided by each user to a movie. U.item dataset contain movie_id , movie_title and all genre in individual columns.

```
# Reading ratings file
ratings_df = pd.read_csv('C:/Users/anwes/OneDrive/Documents/UCF_Summer_2021/STS/Final_Project/ml-100k/u.data',
                     sep='\t', header=None , encoding='latin-1',names=['user_id','movie_id','rating','timestamp'])

# Reading movies file
movies_df = pd.read_csv('C:/Users/anwes/OneDrive/Documents/UCF_Summer_2021/STS/Final_Project/ml-100k/u.item'
                    , sep='|', encoding='latin-1',header=None,names=['movie_id','movie_title','release_date','video_release_
             'IMDb_URL','genre_unknown','genre_Action','genre_Adventure',
             'genre_Animation','genre_Childrens','genre_Comedy','genre_Crime',
             'genre_Documentary','genre_Drama','genre_Fantasy','genre_FilmNoir',
             'genre_Horror','genre_Musical','genre_Mystery','genre_Romance',
             'genre_SciFi','genre_Thriller','genre_War','genre_Western'])

# Reading movie file with data preparation
data_movies = pd.read_csv('C:/Users/anwes/OneDrive/Documents/UCF_Summer_2021/STS/Final_Project/ml-100k/data_movies.csv', enco
```

```
ratings_df
```

	user_id	movie_id	rating	timestamp
0	196	242	3	881250949
1	186	302	3	891717742
2	22	377	1	878887116
3	244	51	2	880606923
4	166	346	1	886397596
...	...	...	...	...
99995	880	476	3	880175444
99996	716	204	5	879795543
99997	276	1090	1	874795795
99998	13	225	2	882399156
99999	12	203	3	879959583

100000 rows × 4 columns

```
movies_df
```

	movie_id	movie_title	release_date	video_release_date	IMDb_URL	genre_unknown	genre_Action	genre_Adventure	genre_Animation
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0	0	0	1
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0	1	1	0
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0	0	0	0
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0	1	0	0
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0	0	0	0
...	...	...	...	...	...	...	...	...	...
1677	1678	Mat' i syn (1997)	06-Feb-1998	NaN	http://us.imdb.com/M/title-exact?Mat%27+i+syn+...	0	0	0	0
1678	1679	B. Monkey (1998)	06-Feb-1998	NaN	http://us.imdb.com/M/title-exact?B%2E+Monkey+(...	0	0	0	0
1679	1680	Sliding Doors (1998)	01-Jan-1998	NaN	http://us.imdb.com/Title?Sliding+Doors+(1998)	0	0	0	0
1680	1681	You So Crazy (1994)	01-Jan-1994	NaN	http://us.imdb.com/M/title-exact?You%20So%20Cr...	0	0	0	0
1681	1682	Scream of Stone (Schrei aus Stein) (1991)	08-Mar-1996	NaN	http://us.imdb.com/M/title-exact?Schrei%20aus%...	0	0	0	0

1682 rows × 24 columns

For our analysis we have converted all the genre to a single column with pipe delimated values.

```
data_movies
```

	movie_id	movie_title	genre
0	1	Toy Story (1995)	Animation | Childrens | Comedy
1	2	GoldenEye (1995)	Action | Adventure | Thriller
2	3	Four Rooms (1995)	Thriller
3	4	Get Shorty (1995)	Action | Comedy | Drama
4	5	Copycat (1995)	Crime | Drama | Thriller
...	...	...	...
1677	1678	Mat' i syn (1997)	Drama
1678	1679	B. Monkey (1998)	Romance | Thriller
1679	1680	Sliding Doors (1998)	Drama | Romance
1680	1681	You So Crazy (1994)	Comedy
1681	1682	Scream of Stone (Schrei aus Stein) (1991)	Drama

1682 rows × 3 columns

3. Statistical Analysis

3.1. Shape of Dataset

There are 100000 rows and 4 columns for rating dataset whereas movie has 1682 rows and 3 column.

```
print(" Rating dataset: ( rows, columns) = ",ratings_df.shape)
print(" Movie dataset: ( rows, columns) = ",data_movies.shape)
```

```
 Rating dataset: ( rows, columns) =  (100000, 4)
 Movie dataset: ( rows, columns) =  (1682, 3)
```

3.2. Descriptive statistics

In rating data frame all the attributes are numeric whereas movie dataset only Movie_id is numeric.

```
ratings_df.describe(include='all')
```

	user_id	movie_id	rating	timestamp
count	100000.00000	100000.000000	100000.000000	1.000000e+05
mean	462.48475	425.530130	3.529860	8.835289e+08
std	266.61442	330.798356	1.125674	5.343856e+06
min	1.00000	1.000000	1.000000	8.747247e+08
25%	254.00000	175.000000	3.000000	8.794487e+08
50%	447.00000	322.000000	4.000000	8.828269e+08
75%	682.00000	631.000000	4.000000	8.882600e+08
max	943.00000	1682.000000	5.000000	8.932866e+08

```
data_movies.describe(include='all')
```

	movie_id	movie_title	genre
count	1682.000000	1682	1682
unique	NaN	1664	216
top	NaN	Chairman of the Board (1998)	Drama
freq	NaN	2	376
mean	841.500000	NaN	NaN
std	485.695893	NaN	NaN
min	1.000000	NaN	NaN
25%	421.250000	NaN	NaN
50%	841.500000	NaN	NaN
75%	1261.750000	NaN	NaN
max	1682.000000	NaN	NaN

3.3. Missing values

Both the dataset has no missing values.

```
ratings_df.isnull().any()
```

```
user_id      False
movie_id     False
rating       False
timestamp    False
dtype: bool
```

```
data_movies.isnull().any()
```

```
movie_id       False
movie_title    False
genre          False
dtype: bool
```

3.4. Duplicate Entry

Below movies are duplicate because they have same genre.

```
df_tmp=data_movies[data_movies.duplicated(subset = 'movie_title', keep = False)]
lis1=[]
lis2=[]
for j in range(32):
    x=0
    for i in range(31):
        if df_tmp.iloc[j,1]== df_tmp.iloc[i+1,1]:
            if df_tmp.iloc[j,2]== df_tmp.iloc[i+1,2]:
                x+=1
                if x>1:
#                 print(j, i)
                    lis1.append(df_tmp.iloc[j,1])
                else:
                    lis2.append(df_tmp.iloc[j,1])

print('List of dublicate movies with same genres {}'.format(list(set(lis1))))
print('')
print('List of dublicate movies with different genres {}'.format(list(set(lis2)-set(lis1))))
```

```
List of dublicate movies with same genres ['That Darn Cat! (1997)', 'Hugo Pool (1997)', 'Hurricane Streets (1998)', 'Nightwatch (1997)', 'Ice Storm, The (1997)', 'Kull the Conqueror (1997)', 'Desperate Measures (1998)', "Ulee's Gold (1997)", 'Money Talks (1997)', 'Deceiver (1997)', 'Designated Mourner, The (1997)', 'Fly Away Home (1996)', 'Body Snatchers (1993)']

List of dublicate movies with different genres ['Substance of Fire, The (1996)', 'Chairman of the Board (1998)', 'Sliding Doors (1998)', 'Butcher Boy, The (1998)', 'Chasing Amy (1997)']
```

Deleting the duplicate movie information.

```
data_movies.drop_duplicates(subset='movie_title', inplace = True, keep= 'first')
```

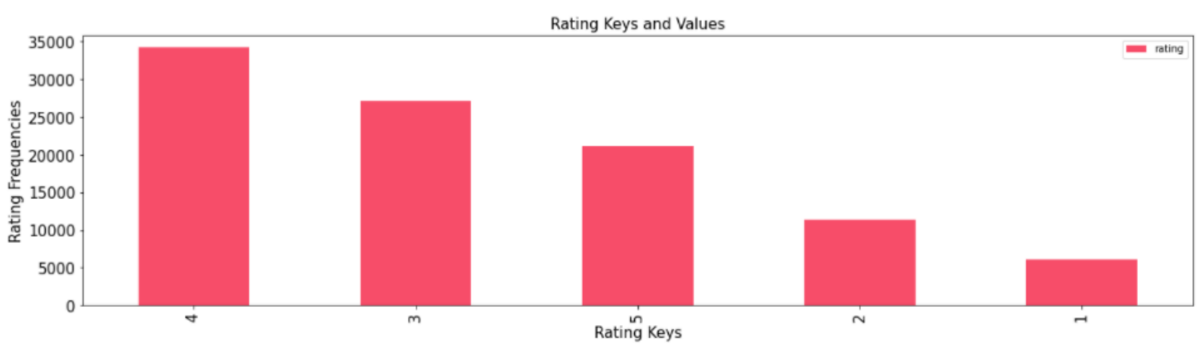
```
data_movies[data_movies.duplicated(subset = 'movie_title', keep = False)]
```

]:

movie_id	movie_title	genre

4. Exploratory Data Analysis

- Finding Rating and reference. Below is the bar plot . Most user rated movie as rating 4.



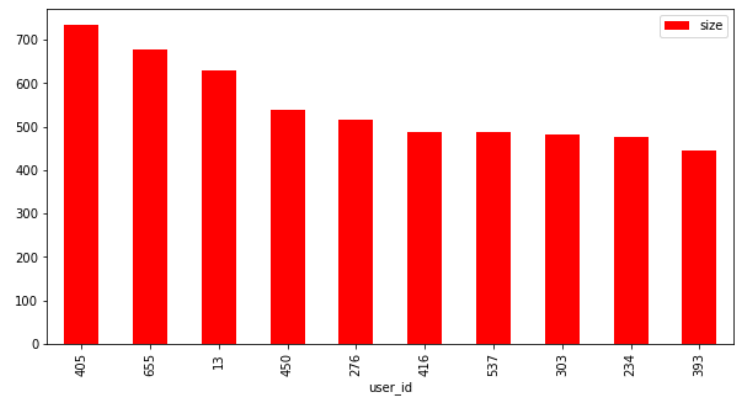
- Top users who have rated most of the movies.

```
merge_ratings_movies = pd.merge(data_movies, ratings_df, on='movie_id', how='inner')
merge_ratings_movies = merge_ratings_movies.drop('timestamp', axis=1)
ratings_grouped_by_users = merge_ratings_movies.groupby('user_id').agg([np.size, np.mean])
ratings_grouped_by_users = ratings_grouped_by_users.drop('movie_id', axis = 1)
```

```
ratings_grouped_by_users_df = pd.DataFrame(ratings_grouped_by_users['rating']['size'].sort_values(ascending=False).head(10))
```

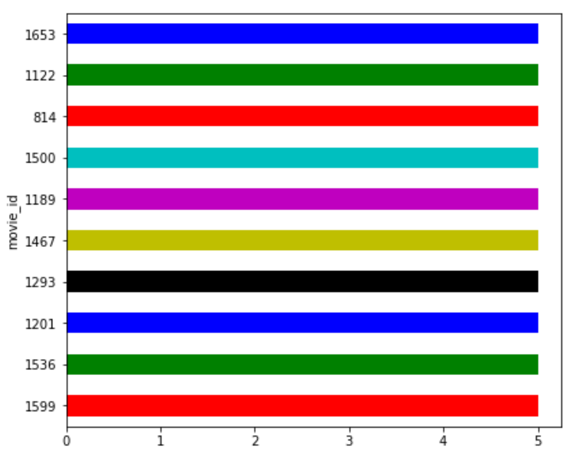
```
ratings_grouped_by_users_df.plot(kind="bar",figsize = (10,5), color = ['r', 'g', 'b', 'k', 'y', 'm', 'c'])
```

```
5]: <AxesSubplot:xlabel='user_id'>
```



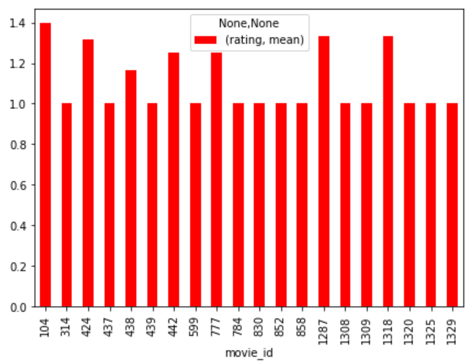
- Movie with high average rating

```
ratings_movies.groupby('movie_id').agg([np.mean], np.size)
s_grouped_by_movies.drop('user_id', axis=1)
]['mean'].sort_values(ascending=False).head(10).plot(kind='barh', figsize=(7,6), color = ['r', 'g', 'b', 'k', 'y', 'm', 'c'])
```



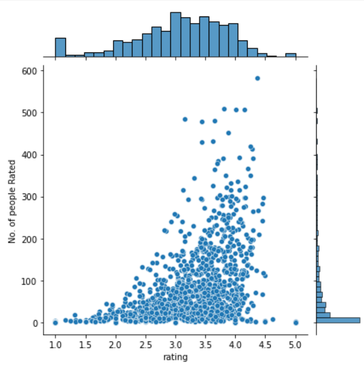
- Movies with low average rating

```
low_rated_movies_filter = ratings_grouped_by_movies['rating']['mean']< 1.5
low_rated_movies = ratings_grouped_by_movies[low_rated_movies_filter]
low_rated_movies.head(20).plot(kind='bar', figsize=(7,5),  color = ['r', 'g', 'b', 'k', 'y', 'm', 'c']);
```



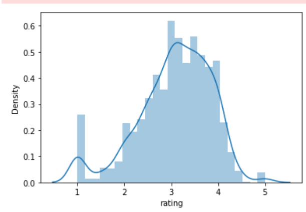
- Most rated users range from density plots. Most user rated between 3 and 4.

```
sns.jointplot(x=New_data['rating'],y=New_data['No. of people Rated']);
```




```
sns.distplot(New_data['rating']);
```

```
C:\Users\anwes\anaconda3\lib\site-packages\seaborn\distributions.py:2551
and will be removed in a future version. Please adapt your code to use e
r flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

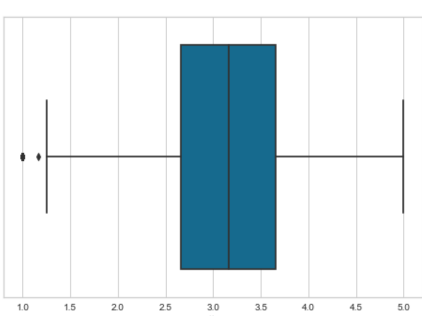



```
sns.boxplot(New_data['rating'], orient='v')
```

```
C:\Users\anwes\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWa
d arg: x. From version 0.12, the only valid positional argument will be `data`
cit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\anwes\anaconda3\lib\site-packages\seaborn\_core.py:1303: UserWarning:
ecified.
  warnings.warn(single_var_warning.format("Vertical", "x"))
```

```
]: <AxesSubplot:xlabel='rating'>
```

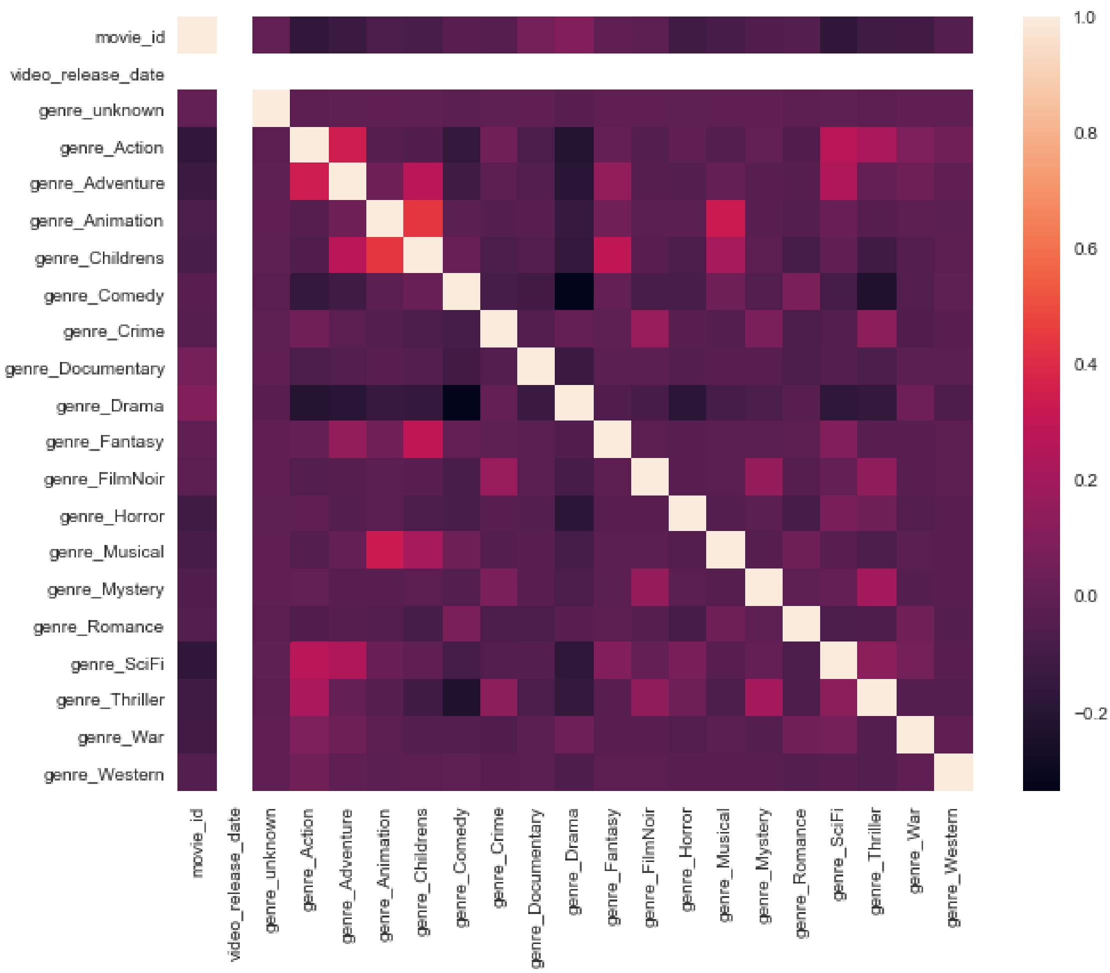
75% movies have rating less around 3.2



- Co-relation between all genre.

```
plt.figure(figsize = (10,8))
sns.heatmap(movies_df.corr(), annot=False)
```

5]: <AxesSubplot:>



5. Content based filtering

It simply helps you by identifying movies that are similar to the movies you like. Content-based recommendation systems are limited because they do not contain other user data. And it doesn't help a user discover their potential tastes. For example, let's say that user U1 and user U2 like Adventure movies. User A also likes drama movies, but since you don't have that knowledge, you keep offering Adventure movies. Eventually, this method eliminates other options that user U2 potentially might like. For this we need to have a minimal understanding of the users' preferences, so that we can then recommend new items with similar tags/keywords to those specified (or inferred) by the user

```
genre_popular = (data_movies.genre.str.split('|')
                     .explode()
                     .value_counts()
                     .sort_values(ascending=False))
genre_popular.head(10)
```

```
: Drama        370
  Action       217
  Comedy       216
  Comedy       209
   Thriller    207
   Romance     179
  Drama        153
   Drama       129
   War          65
   Drama        64
  Name: genre, dtype: int64
```

Using Wordcloud to find the frequency of each word of genre. This will help us to identify to take less frequent use word into consideration.

```
genre_wc = WordCloud(width=1000,height=400,background_color='white')
genre_wc.generate_from_frequencies(genre_popular.to_dict())
plt.figure(figsize=(16, 8))
plt.imshow(genre_wc, interpolation="bilinear")
plt.axis('off')
```

```
3]: (-0.5, 999.5, 399.5, -0.5)
```

From above visualization, the most frequent genres are Drama, Comedy and Action. less frequent genres are Western, Fantasy , Sci-Fi. For our recommendation system we need to consider genres with less frequency

As an example let's consider a user who wants to find a movie similar to "The Good, the Bad and the Ugly", which is a mixture of Western, Action and Adventure. We will consider Western, since there will be many Action or Adventure movies, which are not Western, which could lead to recommending many none Western movies.

Next we have to find similarity between the vector generated in previous step. The commonly used proximity measure algorithm is cosine similarity

The lower the angle between two vectors, the higher the cosine will be, hence yielding a higher similarity factor

```
# Define a TF-IDF Vectorizer Object.
tfidf_movies_genres = TfidfVectorizer(token_pattern = '[a-zA-Z0-9\-]+')

#Replace NaN with an empty string
data_movies['genre'] = data_movies['genre'].replace(to_replace="(no genres listed)", value="")

#Construct the required TF-IDF matrix by fitting and transforming the data
movies_genres_matrix = tfidf_movies_genres.fit_transform(data_movies['genre'])

cosine_sim_movies = linear_kernel(movies_genres_matrix, movies_genres_matrix)
#cosine_sim_test = cosine_similarity(movies_genres_matrix)
```

```
movies_genres_matrix
```

```
]: <1664x19 sparse matrix of type '<class 'numpy.float64'>'
        with 2863 stored elements in Compressed Sparse Row format>
```

```
cosine_sim_movies
```

```
]: array([[1.        , 0.        , 0.        , ..., 0.        , 0.34901009,
         0.        ],
        [0.        , 1.        , 0.53681382, ..., 0.37852635, 0.        ,
         0.        ],
        [0.        , 0.53681382, 1.        , ..., 0.70513526, 0.        ,
         0.        ],
        ...,
        [0.        , 0.37852635, 0.70513526, ..., 1.        , 0.        ,
         0.        ],
        [0.34901009, 0.        , 0.        , ..., 0.        , 1.        ,
         0.        ],
        [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
         1.        ]])
```

This function will recommended other movies similar to "birdcage, The (1996)"

```
CB_recommendations_on_genres("Birdcage, The (1996)", data_movies )
```

```
]: 24                                Birdcage, The (1996)
   25                       Brothers McMullen, The (1995)
   39     To Wong Foo, Thanks for Everything! Julie Newm...
   40                                Billy Madison (1995)
   41                                       Clerks (1994)
   Name: movie_title, dtype: object
```

Below function will give all the movies when we pass a userid which he has not yet watched but recommended.

```
CB_recommendation_on_content(20)
```

```
]: {'101 Dalmatians (1996)',
 'Absolute Power (1997)',
 'Adventures of Robin Hood, The (1938)',
 'Akira (1988)',
 'Aladdin and the King of Thieves (1996)',
 'Alice in Wonderland (1951)',
 'All Dogs Go to Heaven 2 (1996)',
 'Angels and Insects (1995)',
 "Antonia's Line (1995)",
 'Aristocats, The (1970)',
 'Backbeat (1993)',
 'Bad Taste (1987)',
 'Bananas (1971)',
 'Beavis and Butt-head Do America (1996)',
 'Belle de jour (1967)',
 'Blues Brothers 2000 (1998)',
 'Boot, Das (1981)',
 'Braindead (1992)',
 "Breakfast at Tiffany's (1961)",
```

6. Collaborative filtering

Collaborative filtering uses various techniques to check people with similar interests and make recommendations based on shared interests.

Below are steps followed by collaborative filtering:

i) User Rating: A user rates movies to express the liking. Algo treats the ratings as an approximate representation of the user's interest in movies

ii) Similar User: Then it matches this user's ratings with other users' ratings and finds the people with the most similar ratings

iii) Movie Recommendation: The system recommends items that the similar users have rated highly but not yet being rated by this user

Types of collaborative filtering techniques

i) Memory based

A memory-based system uses users' rating data to compute the similarity between users or Movie.

ii) User-Item Filtering

Step 1: Look for user who share the same rating patterns with the given user

Step 2: Use the ratings from the user found in step 1 to calculate a prediction of a rating by the given user on a movie

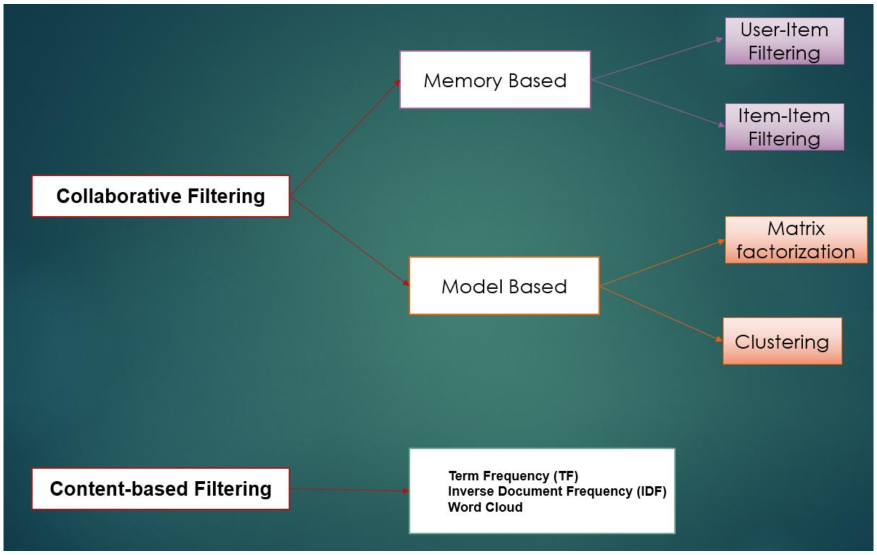
iii) Item-Item Filtering

Step 1: Build an item-item matrix of the rating relationships between pairs of items

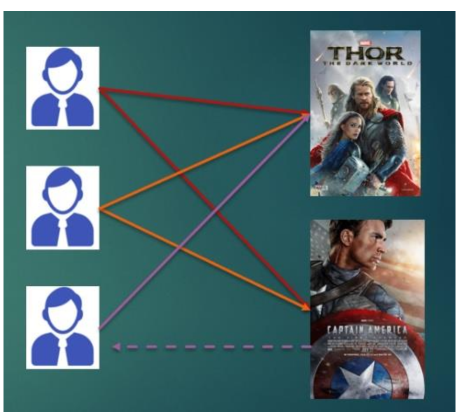
Step 2: Predict the rating of the current user on a product by examining the matrix and matching that user's rating data

* Model based

we develop models using different machine learning algorithms to predict users' unrated items There are many model-based collaborative filtering algorithms such as Matrix factorization algorithms.



6.1. Item-Item filtering



• Similarity between M1 and M2 is based on how many common users liked both.

• If similarity is high, then we can recommend M1 to user who did not watch it

Created the matrix of user and Movie for ratings

```
ratings_matrix_items = df_movies_ratings.pivot_table(index=['movie_id'],columns=['user_id'],values='rating').reset_index(
ratings_matrix_items.fillna( 0, inplace = True )
ratings_matrix_items.shape
```

0]: (1682, 943)

```
ratings_matrix_items
```

1]:

user_id	1	2	3	4	5	6	7	8	9	10	...	934	935	936	937	938	939	940	941	942	943
0	5.0	4.0	0.0	0.0	4.0	4.0	0.0	0.0	0.0	4.0	...	2.0	3.0	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0
1	3.0	0.0	0.0	0.0	3.0	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	5.0
2	4.0	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
3	3.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	4.0	...	5.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0
4	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	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1677	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	0.0	0.0	0.0	0.0	0.0
1678	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	0.0	0.0	0.0	0.0	0.0
1679	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	0.0	0.0	0.0	0.0	0.0
1680	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	0.0	0.0	0.0	0.0	0.0
1681	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	0.0	0.0	0.0	0.0	0.0

1682 rows × 943 columns

Convert the matrix into a pairwise cosine distance matrix where the diagonals are 0 and symmetric by diagonal.

```
movie_similarity = 1 - pairwise_distances( ratings_matrix_items.to_numpy(), metric="cosine" )
np.fill_diagonal( movie_similarity, 0 ) #Filling diagonals with 0s for future use when sorting is done
ratings_matrix_items = pd.DataFrame( movie_similarity )
ratings_matrix_items
```

:

	0	1	2	3	4	5	6	7	8	9	...	1672	1673	1674	1675	1676
0	0.000000	0.402382	0.330245	0.454938	0.286714	0.116344	0.620979	0.481114	0.496288	0.273935	...	0.035387	0.0	0.000000	0.000000	0.035387
1	0.402382	0.000000	0.273069	0.502571	0.318836	0.083563	0.383403	0.337002	0.255252	0.171082	...	0.000000	0.0	0.000000	0.000000	0.000000
2	0.330245	0.273069	0.000000	0.324866	0.212957	0.106722	0.372921	0.200794	0.273669	0.158104	...	0.000000	0.0	0.000000	0.000000	0.032292
3	0.454938	0.502571	0.324866	0.000000	0.334239	0.090308	0.489283	0.490236	0.419044	0.252561	...	0.000000	0.0	0.094022	0.094022	0.037609
4	0.286714	0.318836	0.212957	0.334239	0.000000	0.037299	0.334769	0.259161	0.272448	0.055453	...	0.000000	0.0	0.000000	0.000000	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1677	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000
1678	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000
1679	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000
1680	0.047183	0.078299	0.000000	0.056413	0.000000	0.000000	0.051498	0.082033	0.057360	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000
1681	0.047183	0.078299	0.096875	0.075218	0.094211	0.000000	0.051498	0.000000	0.071700	0.000000	...	0.000000	0.0	0.000000	0.000000	0.000000

1682 rows × 1682 columns

```
user_id=50
print("Recommended movies,:\n",movieIdToTitle(recommendedMoviesAsperItemSimilarity(user_id)))
#recommendedMoviesAsperItemSimilarity(1)
```

```
Recommended movies,:
 [14    Mr. Holland's Opus (1995)
Name: movie_title, dtype: object, 14    Mr. Holland's Opus (1995)
Name: movie_title, dtype: object, 3    Get Shorty (1995)
Name: movie_title, dtype: object, 3    Get Shorty (1995)
Name: movie_title, dtype: object, 14    Mr. Holland's Opus (1995)
Name: movie_title, dtype: object, 14    Mr. Holland's Opus (1995)
Name: movie_title, dtype: object, 3    Get Shorty (1995)
Name: movie_title, dtype: object, 3    Get Shorty (1995)
Name: movie_title, dtype: object, 14    Mr. Holland's Opus (1995)
Name: movie_title, dtype: object]
```

We pass the user Id to the function and it will give movies as recommended items.

6.2. User-item filtering

Below is one of the use case we can think off.

- There are two users U1 and U2
- U1 watched two movie M1 and M2
- U2 watched one movie M1
- This method computes similarity of two users and find
- any common watched movie.

Here we have to find user similarity.

```
ratings_matrix_users = df_movies_ratings.pivot_table(index=['user_id'],columns=['movie_id'],values='rating').reset_index(drop
ratings_matrix_users.fillna( 0, inplace = True )
movie_similarity = 1 - pairwise_distances( ratings_matrix_users.to_numpy(), metric="cosine" )
np.fill_diagonal( movie_similarity, 0 ) #Filling diagonals with 0s for future use when sorting is done
ratings_matrix_users = pd.DataFrame( movie_similarity )
ratings_matrix_users
```

	0	1	2	3	4	5	6	7	8	9	...	933	934	935	936	9
0	0.000000	0.166931	0.047460	0.064358	0.378475	0.430239	0.440367	0.319072	0.078138	0.376544	...	0.369527	0.119482	0.274876	0.189705	0.197
1	0.166931	0.000000	0.110591	0.178121	0.072979	0.245843	0.107328	0.103344	0.161048	0.159862	...	0.156986	0.307942	0.358789	0.424046	0.319
2	0.047460	0.110591	0.000000	0.344151	0.021245	0.072415	0.066137	0.083060	0.061040	0.065151	...	0.031875	0.042753	0.163829	0.069038	0.124
3	0.064358	0.178121	0.344151	0.000000	0.031804	0.068044	0.091230	0.188060	0.101284	0.060859	...	0.052107	0.036784	0.133115	0.193471	0.146
4	0.378475	0.072979	0.021245	0.031804	0.000000	0.237286	0.373600	0.248930	0.056847	0.201427	...	0.338794	0.080580	0.094924	0.079779	0.148
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
938	0.118095	0.228583	0.026271	0.030138	0.071459	0.111852	0.107027	0.095898	0.039852	0.071460	...	0.066039	0.431154	0.258021	0.226449	0.432
939	0.314072	0.226790	0.161890	0.196858	0.239955	0.352449	0.329925	0.246883	0.120495	0.342961	...	0.327153	0.107024	0.187536	0.181317	0.175
940	0.148617	0.161485	0.101243	0.152041	0.139595	0.144446	0.059993	0.146145	0.143245	0.090305	...	0.046952	0.203301	0.288318	0.234211	0.313
941	0.179508	0.172268	0.133416	0.170086	0.152497	0.317328	0.282003	0.175322	0.092497	0.212330	...	0.226440	0.073513	0.089588	0.129554	0.099
942	0.398175	0.105798	0.026556	0.058752	0.313941	0.276042	0.394364	0.299809	0.075617	0.221860	...	0.263791	0.210763	0.143253	0.077793	0.202

943 rows × 943 columns

Here is the mapping of two similar user.

```
similar_user_series= ratings_matrix_users.idxmax(axis=1)
df_similar_user= similar_user_series.to_frame()
```

```
df_similar_user.columns=['similarUser']
```

```
df_similar_user
```

	similarUser
0	915
1	700
2	862
3	749
4	306
...	...
938	717
939	912
940	688
941	453
942	681

943 rows × 1 columns

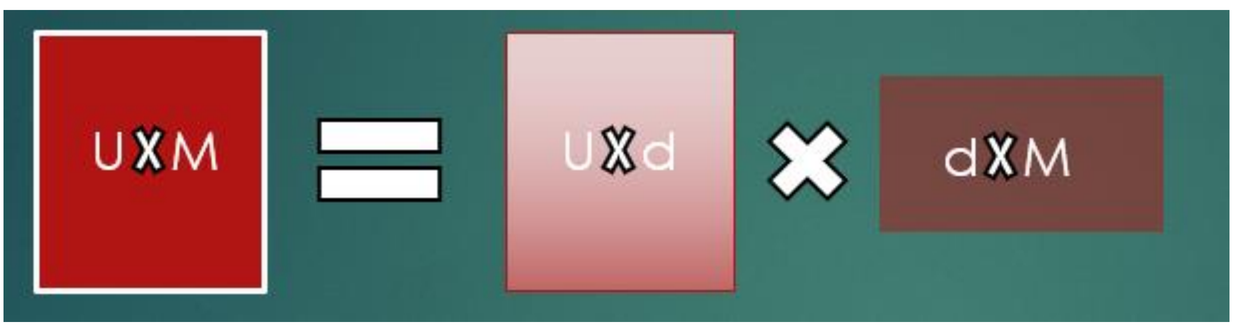
Below function will output the recommended movies if we pass the user id.

```
user_id=50
recommend_movies= movieIdToTitle(getRecommendedMoviesAsperUserSimilarity(user_id))
print("Movies you should watch are:\n")
print(recommend_movies)
```

```
Movies you should watch are:

[267    Chasing Amy (1997)
Name: movie_title, dtype: object, 344    Deconstructing Harry (1997)
Name: movie_title, dtype: object, 690    Dark City (1998)
Name: movie_title, dtype: object, 301    L.A. Confidential (1997)
Name: movie_title, dtype: object, 285    English Patient, The (1996)
Name: movie_title, dtype: object, 749    Amistad (1997)
Name: movie_title, dtype: object, 312    Titanic (1997)
Name: movie_title, dtype: object, 314    Apt Pupil (1998)
Name: movie_title, dtype: object, 333    U Turn (1997)
Name: movie_title, dtype: object]
```

6.3. Matrix factorization

Matrix factorization is used to dimension reduction technique. In our user-movie matrix there are lot of users have not provided any rtings so these are empty cells. To avoid this we have to perform Dimension reduction where Matrix (U-M) decomposed into Matric (U-D) where each row are users and Matrix (d-M) where each columns are movies. After that used dot product to get the final matrix having no non-empty cell.




```
n_users = ratings_df.user_id.unique().shape[0]
n_movies = ratings_df.movie_id.unique().shape[0]
```

```
Ratings = ratings_df.pivot(index = 'user_id', columns ='movie_id', values = 'rating').fillna(0)
Ratings.head()
```

movie_id user_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	1676	1677	1678	1679	1680	1681	1682
1	5.0	3.0	4.0	3.0	3.0	5.0	4.0	1.0	5.0	3.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	...	0.0	0.0	0.0	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	0.0	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
4	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	0.0	0.0	0.0	0.0	0.0
5	4.0	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	0.0	0.0	0.0	0.0

5 rows × 1682 columns

```
U, sigma, Vt = svds(Ratings_demeaned, k = 50)
```

```
print('Size of sigma: ' , sigma.size)
```

```
Size of sigma:  50
```

```
sigma = np.diag(sigma)
```

```
print('Shape of U: ', U.shape)
print('Shape of Vt: ', Vt.shape)
```

```
Shape of U:  (943, 50)
Shape of Vt:  (50, 1682)
```

```
U
```

```
]: array([[ 0.13944814,  0.08802883, -0.11959544, ...,  0.00473136,
          0.0043773 , -0.06653149],
        [ 0.02104373,  0.03419113, -0.00195072, ..., -0.05392487,
         -0.04620115, -0.01309312],
        [-0.01493341,  0.00562006,  0.02046879, ..., -0.02317463,
         -0.02481712, -0.00320484],
        ...,
        [ 0.01119652, -0.00552595, -0.00069347, ..., -0.00746046,
         -0.02554262, -0.0082399 ],
        [ 0.05682848,  0.00132044, -0.08516041, ..., -0.02379019,
          0.00759561, -0.02504761],
        [ 0.00657694,  0.02726909, -0.06758361, ...,  0.05701743,
         -0.01320454, -0.04472769]])
```

```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
```

```
print('All user predicted rating : ', all_user_predicted_ratings.shape)
```

```
All user predicted rating :  (943, 1682)
```

Below is the final matrix.

```
preds = pd.DataFrame(all_user_predicted_ratings, columns = Ratings.columns)
preds
```

]:

movie_id	1	2	3	4	5	6	7	8	9	10	...	1673	1674	1675	
0	6.488436	2.959503	1.634987	3.024467	1.656526	1.659506	3.630469	0.240669	1.791518	3.347816	...	0.011976	-0.092017	-0.074553	-0
1	2.347262	0.129689	-0.098917	0.328828	0.159517	0.481361	0.213002	0.097908	1.892100	0.671000	...	0.003943	-0.026939	-0.035460	-0
2	0.291905	-0.263830	-0.151454	-0.179289	0.013462	-0.088309	-0.057624	0.568764	-0.018506	0.280742	...	-0.028964	-0.031622	0.045513	0
3	0.366410	-0.443535	0.041151	-0.007616	0.055373	-0.080352	0.299015	-0.010882	-0.160888	-0.118834	...	0.020069	0.015981	-0.000182	0
4	4.263488	1.937122	0.052529	1.049350	0.652765	0.002836	1.730461	0.870584	0.341027	0.569055	...	0.019973	-0.053521	-0.017242	-0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
938	1.601615	-0.110491	-0.198045	-0.229476	0.345397	0.152378	-0.133373	1.073894	2.993480	-0.240829	...	0.033564	0.014452	0.067121	0
939	0.585532	-0.355471	-0.186924	2.170066	0.457680	0.013850	3.113494	2.612028	2.554361	-0.451883	...	-0.016607	0.003067	-0.021431	-0
940	3.118558	-0.041062	0.546047	-0.060874	-0.169393	0.015739	2.338824	0.417505	0.679524	-0.015267	...	-0.009333	-0.006661	-0.040438	-0
941	0.943730	0.599492	0.486034	-0.363920	0.465666	0.173843	-0.276099	1.390914	-0.509617	-0.751110	...	0.010092	0.028925	0.033764	0
942	1.359590	2.856329	1.770723	1.820281	1.066240	0.314059	1.291571	0.047941	1.869433	-0.549563	...	0.000092	-0.115652	-0.100940	-0

943 rows × 1682 columns

Below function will give output of movies that user have not watch bit may be interested in watching it and rating.

```
already_rated, predictions = recommend_movies(preds, 20, data_movies, ratings_df, 20)
```

```
User 20 has already rated 48 movies.
Recommending highest 20 predicted ratings movies not already rated.
```

```
already_rated.head(20)
```

36]:

	user_id	movie_id	rating	timestamp	movie_title	release_date	similarity
21	20	87	5	879669746	Searching for Bobby Fischer (1993)	Drama	0.352938
38	20	496	5	879669244	It's a Wonderful Life (1946)	Drama	0.385319
35	20	148	5	879668713	Ghost and the Darkness, The (1996)	Action | Adventure	0.342833
42	20	22	5	879669339	Braveheart (1995)	Action | Drama | War	0.413189
17	20	252	4	879669697	Lost World: Jurassic Park, The (1997)	Action | Adventure | SciFi | Thriller	0.301736
40	20	633	4	879668979	Christmas Carol, A (1938)	Drama	0.224437
37	20	274	4	879668248	Sabrina (1995)	Comedy | Romance	0.362297
31	20	174	4	879669087	Raiders of the Lost Ark (1981)	Action | Adventure	0.438706
30	20	210	4	879669065	Indiana Jones and the Last Crusade (1989)	Action | Adventure	0.372300
29	20	934	4	879668783	Preacher's Wife, The (1996)	Drama	0.222122
44	20	243	4	879667799	Jungle2Jungle (1997)	Childrens | Comedy	0.143937

For evaluation purpose used SVD (Single vector decompose) to predict the matrix and find out RMSE value. Here RMSE value we got around 0.94 which are good predictive model.

```
# Load Reader library
reader = Reader()
svd = SVD()
# Load ratings dataset with Dataset library
data = Dataset.load_from_df(ratings_df[['user_id', 'movie_id', 'rating']], reader)

# Split the dataset for 5-fold evaluation
#data.split(n_folds=5)
cross_validate(SVD(), data, measures=['RMSE', 'MAE'], cv=5, verbose=True)
```

```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9425  0.9377  0.9297  0.9339  0.9321  0.9352  0.0045
MAE (testset)     0.7424  0.7371  0.7337  0.7394  0.7348  0.7375  0.0031
Fit time          11.31   10.56   12.37   13.70   11.87   11.96   1.06
Test time         1.07    0.16    0.38    0.36    0.31    0.46    0.32
```

3]:

```
{'test_rmse': array([0.94250725, 0.93774227, 0.92966634, 0.93394736, 0.93205719]),
 'test_mae': array([0.74243662, 0.73711627, 0.73374884, 0.73939306, 0.73480535]),
 'fit_time': (11.311083555221558,
  10.562217235565186,
  12.37008285522461,
  13.70439863204956,
  11.867352485656738),
 'test_time': (1.074561357498169,
  0.16156864166259766,
  0.3809826374053955,
  0.3620333671569824,
  0.31027936935424805)}
```

Below are the predict result of user 20 for movie 10 and 194. The prediction is giving us the may be rating of the user.

```
]: svd.predict(20, 10)
```

```
[260]: Prediction(uid=20, iid=10, r_ui=None, est=3.7539161946691997, details={'was_impossible': False})
```

For movie with ID 10, I get an estimated prediction of 3.7. The recommender system works purely on the basis of an assigned movie ID and t
ratings based on how the other users have predicted the movie.

```
.]: svd.predict(20, 194)
```

```
[261]: Prediction(uid=20, iid=194, r_ui=None, est=3.954585087195383, details={'was_impossible': False})
```

6.4. Clustering

Clustering is methodology to find common pattern in user or movies which can club them in one cluster. We prepared a sparse matrix before applying K mean clustering algorithm.

```
sparseMatrix, feature_names = prepSparseMatrix(users_movies_list)
```

```
df_sparseMatrix = pd.DataFrame(sparseMatrix, index = users, columns = feature_names)
df_sparseMatrix
```

7]:

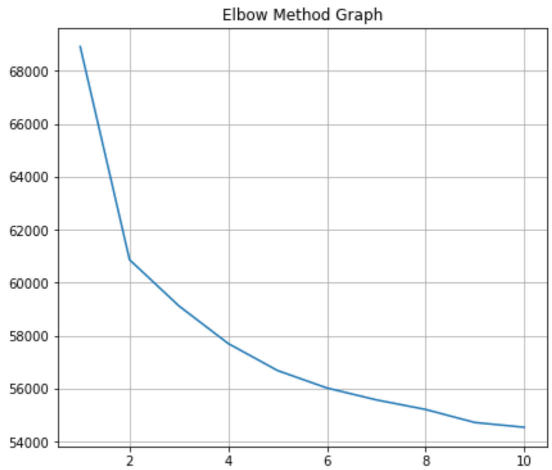
	1	10	100	1000	1001	1002	1003	1004	1005	1006	...	990	991	992	993	994	995	996	997	998	999
1	1	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	1	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	1	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
939	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
940	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
941	1	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	0
942	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
943	0	0	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

943 rows × 1574 columns

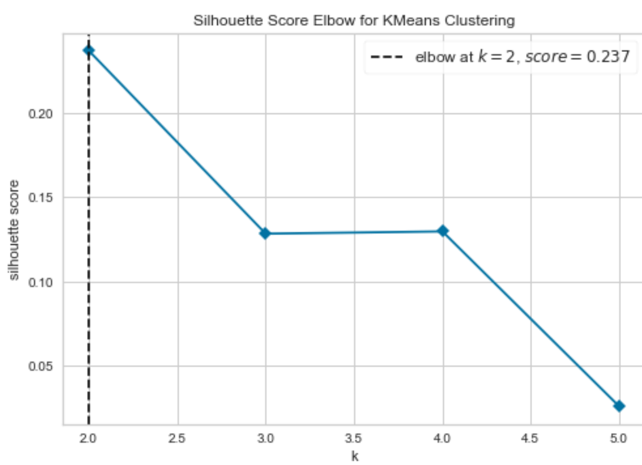
Use K mean clustering finding out best K value using Elbow and Silhouette graph.

```python
elbow_method = elbowMethod(sparseMatrix)
```

```python
elbow_method.run(1, 10)
elbow_method.showPlot(boundary = 10)
```




```python
from yellowbrick.cluster import KElbowVisualizer
model = KMeans(random_state=123)
# Instantiate the KElbowVisualizer with the number of clusters and the metric
visualizer = KElbowVisualizer(model, k=(2,6), metric='silhouette', timings=False)
# Fit the data and visualize
visualizer.fit(sparseMatrix)
visualizer.poof()
```



As per above curve the ideal K value should be 2. So in below Clustering using K as 2.

```
kmeans = KMeans(n_clusters=2, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 123)
clusters = kmeans.fit_predict(sparseMatrix)
```

```
print(kmeans.cluster_centers_)
```

```
[[ 3.28244275e-01  5.19083969e-02  4.06106870e-01 ...  1.47451495e-17
  -2.08166817e-17  1.30104261e-17]
 [ 7.01388889e-01  1.59722222e-01  7.29166667e-01 ...  2.43055556e-02
   2.77777778e-02  1.73611111e-02]]
```

```
print(kmeans.labels_)
```

```
[1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 1 0 1 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0
 0 0 0 0 0 0 0 1 1 0 1 0 1 0 0 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0
 0 0 1 1 0 0 0 0 1 0 0 0 1 0 1 1 0 0 0 0 0 1 0 0 1 0 0 1 1 1 1 0 0 0 0 1 1
 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 0 0 1 1 0 1 0 0 0
 0 0 1 1 0 0 0 1 1 1 0 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 1 0 0 0 1 1 1 1 0 1 1
 1 1 1 0 1 0 1 0 1 0 1 1 0 0 1 1 1 1 0 0 0 1 0 1 1 0 0 0 1 1 1 1 0 1 0 1 0
 1 0 0 0 0 1 0 0 1 1 1 1 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
 0 0 1 1 0 0 0 1 1 1 0 0 0 0 1 0 1 0 1 0 1 0 1 1 0 0 0 1 1 0 1 0 0 0 1 1 1
 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 1 0 0 0 0 1 0 0
 0 0 1 0 0 1 0 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 0 0 0 0 1 0 0
 0 0 1 0 0 1 1 0 0 0 0 1 0 1 0 1 1 0 1 0 0 1 1 1 1 0 1 0 0 0 0 0 1 0 0 0 0
 0 0 1 0 0 1 0 0 1 0 0 0 0 1 1 0 1 1 1 1 0 0 1 1 1 0 1 0 0 0 0 0 1 0 0 1 0
 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1
 1 0 0 0 0 0 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 1 0 1 0 1
 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0
 0 1 0 1 0 1 0 1 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
 1 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0 1 1 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 1 1 0 1 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 0 1 0 0 1 0 0 1 1 1 0 0 0
 0 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 1 0
 1 1 0 1 0 0 0 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 1 0 1 0 0 1 0 0 1 0 0 1 0 0 0
 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1]
```

Here all the users are cluster into two groups (0 and 1) . below are the number of users per group.

```
for i in range(2):
    len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
    print('Users in Cluster ' + str(i) + ' -> ', len_users)
```

```
Users in Cluster 0 ->  655
Users in Cluster 1 ->  288
```

Case 1: Considering a subset of users having most rated genre.

Here we are considering two movies action and horror and we are trying to find out the cluster which shows the likes of each movie.

Calculating average rating per user for Action and horror movie.

```
genre_ratings = get_genre_ratings(ratings_df, data_movies, ['Action', 'Horror'], ['avg_action_rating', 'avg_horror_rating'])
genre_ratings.head()
```

9]:

	avg_action_rating	avg_horror_rating
1	3.33	3.46
2	3.80	3.00
3	2.79	2.40
4	3.88	4.00
5	3.14	2.54

We are only considering rating range from 3 to 5 that is why called this data set as biased dataset.

```
genre_rating_dataset(genre_ratings, score_limit_1, score_limit_2):
d_dataset =    genre_ratings[((genre_ratings['avg_action_rating'] < score_limit_1 - 0.2) & (genre_ratings['avg_horror_rating'
d_dataset = pd.concat([biased_dataset[:300], genre_ratings[:2]])
d_dataset = pd.DataFrame(biased_dataset.to_records())
n biased_dataset
```

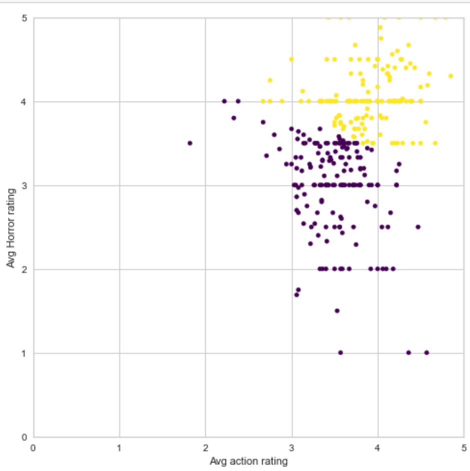
```
biased_dataset = bias_genre_rating_dataset(genre_ratings, 5, 3)
```

```
print( "Number of records: ", len(biased_dataset))
biased_dataset.head()
```

```
Number of records:  302
```

Below are the clusters with 2 K value

```
# Let's turn our dataset into a list
X = biased_dataset[['avg_action_rating','avg_horror_rating']].values
# Import KMeans
from sklearn.cluster import KMeans
# Create an instance of KMeans to find two clusters
kmeans_1 = KMeans(n_clusters=2)
# Use fit_predict to cluster the dataset
predictions = kmeans_1.fit_predict(X)
# Defining the cluster plotting function

draw_clusters(biased_dataset, predictions)
```

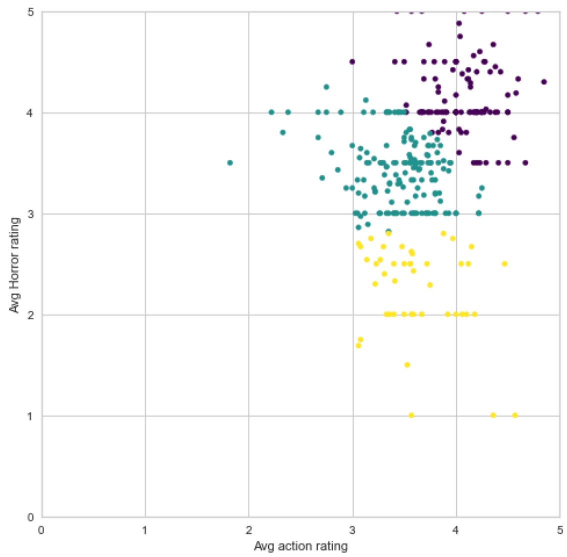


User who like Horror but not Action are in Yellow group

User who like action but not horror are in purple.

Below are the clusters with 3 K value

```python
# Import KMeans
from sklearn.cluster import KMeans
# Create an instance of KMeans to find two clusters
kmeans_2 = KMeans(n_clusters=3)
# Use fit_predict to cluster the dataset
predictions2 = kmeans_2.fit_predict(X)
# Defining the cluster plotting function

draw_clusters(biased_dataset, predictions2)
```



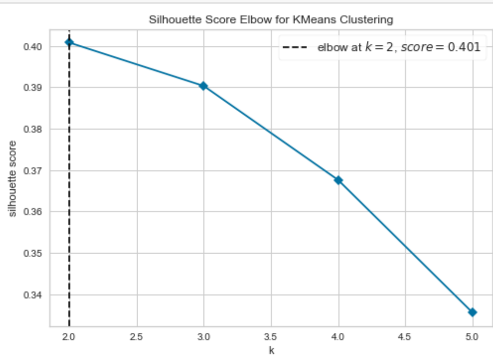
User who like Action movie but not Horror movies are in yellow cluster

User who like Horror and Action are in purple cluster

The green one are quite not sure.

To find best K value used Elbow method .

```python
from yellowbrick.cluster import KElbowVisualizer
model = KMeans(random_state=123)
# Instantiate the KElbowVisualizer with the number of clusters and the metric
visualizer = KElbowVisualizer(model, k=(2,6), metric='silhouette', timings=False)
# Fit the data and visualize
visualizer.fit(X)
visualizer.poof()
```



As per this plot K = 2 is good

Case2: user rated most in a cluster.

We have lot of data with 0 rating so need to re arrange our sparse matrix so that dense will be at beginning of matrix. For this we are using 100 movies and 40 users.

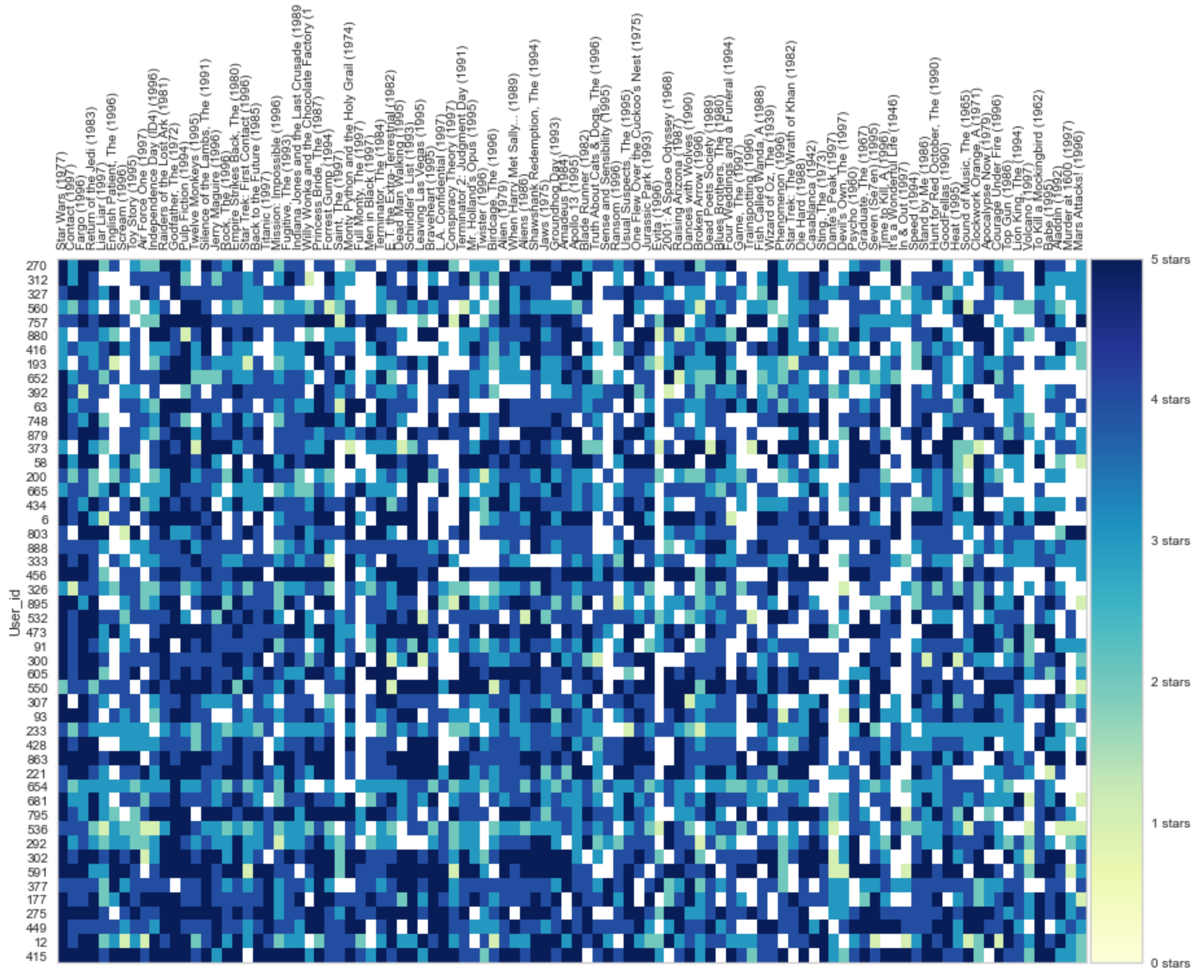
```
n_movies = 100
n_users = 50
most_rated_movies_users_selection = sort_by_rating_density(user_movie_ratings, n_movies, n_users)

most_rated_movies_users_selection
```

7]:

movie_title	Star Wars (1977)	Contact (1997)	Fargo (1996)	Return of the Jedi (1983)	Liar Liar (1997)	English Patient, The (1996)	Scream (1996)	Toy Story (1995)	Air Force One (1997)	Independence Day (ID4) (1996)	...	Apocalypse Now (1979)	Courage Under Fire (1996)	Top Gun (1986)	Lion King, The (1994)	Volcano (1997)	Moc
415	5.0	5.0	5.0	5.0	4.0	5.0	5.0	5.0	4.0	5.0	...	NaN	5.0	4.0	4.0	2.0	
12	5.0	4.0	5.0	5.0	2.0	3.0	1.0	3.0	1.0	5.0	...	5.0	1.0	5.0	4.0	3.0	
449	5.0	4.0	4.0	4.0	4.0	4.0	3.0	4.0	4.0	3.0	...	4.0	4.0	5.0	3.0	NaN	
275	5.0	5.0	5.0	5.0	4.0	NaN	4.0	5.0	4.0	4.0	...	5.0	4.0	3.0	4.0	3.0	
177	5.0	4.0	4.0	5.0	2.0	3.0	5.0	4.0	5.0	5.0	...	3.0	4.0	5.0	4.0	3.0	
377	4.0	4.0	4.0	4.0	2.0	5.0	3.0	4.0	4.0	4.0	...	3.0	3.0	4.0	4.0	NaN	
591	5.0	5.0	5.0	3.0	3.0	5.0	5.0	4.0	NaN	4.0	...	5.0	4.0	NaN	4.0	2.0	
302	5.0	4.0	5.0	5.0	4.0	5.0	4.0	5.0	1.0	3.0	...	NaN	NaN	5.0	3.0	1.0	
292	5.0	3.0	4.0	3.0	2.0	3.0	3.0	2.0	2.0	3.0	...	5.0	3.0	2.0	4.0	2.0	
536	4.0	4.0	4.0	2.0	1.0	3.0	2.0	2.0	1.0	1.0	...	4.0	3.0	NaN	NaN	1.0	
795	5.0	4.0	3.0	5.0	3.0	2.0	NaN	2.0	4.0	5.0	...	2.0	NaN	5.0	4.0	NaN	
681	5.0	3.0	3.0	5.0	3.0	NaN	4.0	4.0	2.0	4.0	...	3.0	3.0	3.0	5.0	1.0	
654	4.0	2.0	3.0	3.0	3.0	3.0	3.0	2.0	3.0	3.0	...	NaN	3.0	2.0	NaN	NaN	
221	4.0	5.0	5.0	4.0	3.0	NaN	4.0	4.0	5.0	3.0	...	3.0	3.0	4.0	4.0	3.0	
863	5.0	5.0	5.0	5.0	4.0	5.0	5.0	5.0	NaN	4.0	...	NaN	5.0	4.0	3.0	4.0	
428	5.0	4.0	5.0	5.0	NaN	NaN	3.0	3.0	3.0	3.0	...	5.0	NaN	3.0	3.0	NaN	
233	4.0	2.0	4.0	3.0	3.0	3.0	3.0	3.0	3.0	NaN	...	3.0	3.0	3.0	3.0	NaN	
93	5.0	5.0	5.0	4.0	NaN	4.0	3.0	4.0	NaN	2.0	...	5.0	4.0	3.0	4.0	NaN	
307	5.0	NaN	5.0	4.0	3.0	NaN	4.0	4.0	NaN	3.0	...	5.0	3.0	3.0	4.0	3.0	
550	2.0	4.0	4.0	2.0	4.0	4.0	4.0	NaN	4.0	5.0	...	5.0	5.0	5.0	4.0	NaN	
605	5.0	4.0	5.0	5.0	2.0	NaN	4.0	5.0	NaN	4.0	...	4.0	4.0	4.0	5.0	3.0	
300	5.0	4.0	5.0	5.0	4.0	NaN	4.0	4.0	4.0	4.0	...	3.0	NaN	3.0	4.0	2.0	

Below visualization will tell the rating density of each user for a movie in a specific cluster. If the density are same for a vertical line then those users are similar in nature.

```python
from mpl_toolkits.axes_grid1 import make_axes_locatable
draw_movies_heatmap(most_rated_movies_users_selection)
```



Dark blue shows rating 5 and light shows rating tends towards 0.

7. Conclusion

In this project I want to compare the advantages and disadvantages of collaborative and content base filtering techniques and different scenarios of recommending movies to user. I feel there are future work required such as using Bayesian network or Hybrid model like including Content and collaborative model using deep learning can give us better result.

8. Helper-functions

```
Calculates top 2 movies to recommend based on given movie titles genres.
:param title: title of movie to be taken for base of recommendation
:param cosine_sim_movies: cosine similarity between movies
:return: Titles of movies recommended to user
```

```python
def CB_recommendations_on_genres(title, df):
    data_frame = df
    # Get the index of the movie that matches the title
    idx_movie = data_frame.loc[data_frame['movie_title'].isin([title])]
    idx_movie = idx_movie.index
    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores_movies = list(enumerate(cosine_sim_movies[idx_movie][0]))
    # Sort the movies based on the similarity scores
    sim_scores_movies = sorted(sim_scores_movies, key=lambda x: x[1], reverse=True)
    # Get the scores of the 10 most similar movies
    sim_scores_movies = sim_scores_movies[1:6]
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores_movies]
    # Return the top 2 most similar movies
    return data_frame['movie_title'].iloc[movie_indices]
```

```
Calculates top movies to be recommended to user based on movie user has watched.
:param userId: userid of user
:return: Titles of movies recommended to user
```

```python
def CB_recommendation_on_content(userId):
    recommended_movie_list = []
    movie_list = []
    df_rating_filtered = ratings_df[ratings_df["user_id"]== userId]
    for key, row in df_rating_filtered.iterrows():
        movie_list.append((data_movies["movie_title"][row["movie_id"]==data_movies["movie_id"]]).values)
    for index, movie in enumerate(movie_list):
        for key, movie_recommended in recommendations_on_genres(movie[0],data_movies).iteritems():
            recommended_movie_list.append(movie_recommended)
    for movie_title in recommended_movie_list:
        if movie_title in movie_list:
            recommended_movie_list.remove(movie_title)
    return set(recommended_movie_list)
```

```python
def get_movie_label(movie_id):
    classifier = KNeighborsClassifier(n_neighbors=5)
    x= movies_genres_matrix
    y = data_movies.iloc[:,-1]
    classifier.fit(x, y)
    y_pred = classifier.predict(movies_genres_matrix[movie_id])
    return y_pred
```

```python
def item_similarity(movieName):
    """
    recomendates similar movies
   :param data: name of the movie
   """
    try:
        #user_inp=input('Enter the reference movie title based on which recommendations are to be made: ')
        user_inp=movieName
        inp=data_movies[data_movies['movie_title']==user_inp].index.tolist()
        inp=inp[0]

        data_movies['similarity'] = ratings_matrix_items.iloc[inp]
        data_movies.columns = ['movie_id', 'movie_title', 'release_date','similarity']
        #print(data_movies)
    except:
        print("Sorry, the movie is not in the database!")
```

```python
def recommendedMoviesAsperItemSimilarity(user_id):
    """
     Recommending movie which user hasn't watched as per Item Similarity
    :param user_id: user_id to whom movie needs to be recommended
    :return: movieIds to user
    """
    user_movie= df_movies_ratings[(df_movies_ratings.user_id==user_id) & (df_movies_ratings.rating.isin([5,4.5]))][['movie_ti
    user_movie=user_movie.iloc[0,0]
    #print(user_movie)
    item_similarity(user_movie)
    sorted_movies_as_per_userChoice=data_movies.sort_values(['similarity'], ascending = False )
    sorted_movies_as_per_userChoice=sorted_movies_as_per_userChoice[sorted_movies_as_per_userChoice['similarity'] >=0.4]['mov
    recommended_movies=list()
    df_recommended_item=pd.DataFrame()
    user2Movies= ratings_df[ratings_df['user_id']== user_id]['movie_id']
    for movieId in sorted_movies_as_per_userChoice:
            if movieId not in user2Movies:
                df_new= ratings_df[(ratings_df.movie_id==movieId)]
                df_recommended_item=pd.concat([df_recommended_item,df_new])
            best10=df_recommended_item.sort_values(["rating"], ascending = False )[1:10]
    return best10['movie_id']
```

```python
def movieIdToTitle(listMovieIDs):
    """
     Converting movieId to titles
    :param user_id: List of movies
    :return: movie titles
    """
    movie_titles= list()
    for id in listMovieIDs:
        movie_titles.append(data_movies[data_movies['movie_id']==id]['movie_title'])
    return movie_titles
```

```python
movieId_recommended=list()
def getRecommendedMoviesAsperUserSimilarity(userId):
    """
     Recommending movies which user hasn't watched as per User Similarity
    :param user_id: user_id to whom movie needs to be recommended
    :return: movieIds to user
    """
    user2Movies= ratings_df[ratings_df['user_id']== userId]['movie_id']
    sim_user=df_similar_user.iloc[0,0]
    df_recommended=pd.DataFrame(columns=['movieId','title','genres','user_id','rating','timestamp'])
    for movieId in ratings_df[ratings_df['user_id']== sim_user]['movie_id']:
        if movieId not in user2Movies:
            df_new= df_movies_ratings[(df_movies_ratings.user_id==sim_user) & (df_movies_ratings.movie_id==movieId)]
            df_recommended=pd.concat([df_recommended,df_new])
        best10=df_recommended.sort_values(['rating'], ascending = False )[1:10]
    return best10['movie_id']
```

```python
def recommend_movies(predictions, userID, movies, original_ratings, num_recommendations):
    """
    Implementation of SVD by hand
    :param predictions : The SVD reconstructed matrix,
    userID : UserId for which you want to predict the top rated movies,
    movies : Matrix with movie data, original_ratings : Original Rating matrix,
    num_recommendations : num of recos to be returned
    :return: num_recommendations top movies
    """
    # Get and sort the user's predictions
    user_row_number = userID - 1 # User ID starts at 1, not 0
    sorted_user_predictions = predictions.iloc[user_row_number].sort_values(ascending=False) # User ID starts at 1

    # Get the user's data and merge in the movie information.
    user_data = original_ratings[original_ratings.user_id == (userID)]
    user_full = (user_data.merge(movies, how = 'left', left_on = 'movie_id', right_on = 'movie_id').
                     sort_values(['rating'], ascending=False)
                 )

    print('User {0} has already rated {1} movies.'.format(userID, user_full.shape[0]))
    print('Recommending highest {0} predicted ratings movies not already rated.'.format(num_recommendations))

    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    recommendations = (movies[~movies['movie_id'].isin(user_full['movie_id'])].
         merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',
               left_on = 'movie_id',
               right_on = 'movie_id').
         rename(columns = {user_row_number: 'Predictions'}).
         sort_values('Predictions', ascending = False).
                       iloc[:num_recommendations, :-1]
                      )

    return user_full, recommendations
```

```python
def prepSparseMatrix(list_of_str):
    # list_of_str = A list, which contain strings of users favourite movies separate by comma ",".
    # It will return us sparse matrix and feature names on which sparse matrix is defined
    # i.e. name of movies in the same order as the column of sparse matrix
    cv = CountVectorizer(token_pattern = r'[^\,\ ]+', lowercase = False)
    sparseMatrix = cv.fit_transform(list_of_str)
    return sparseMatrix.toarray(), cv.get_feature_names()
```

```python
class elbowMethod():
    def __init__(self, sparseMatrix):
        self.sparseMatrix = sparseMatrix
        self.wcss = list()
        self.differences = list()
    def run(self, init, upto, max_iterations = 300):
        for i in range(init, upto + 1):
            kmeans = KMeans(n_clusters=i, init = 'k-means++', max_iter = max_iterations, n_init = 10, random_state = 0)
            kmeans.fit(sparseMatrix)
            self.wcss.append(kmeans.inertia_)
        self.differences = list()
        for i in range(len(self.wcss)-1):
            self.differences.append(self.wcss[i] - self.wcss[i+1])
    def showPlot(self, boundary = 500, upto_cluster = None):
        if upto_cluster is None:
            WCSS = self.wcss
            DIFF = self.differences
        else:
            WCSS = self.wcss[:upto_cluster]
            DIFF = self.differences[:upto_cluster - 1]
        plt.figure(figsize=(15, 6))
        plt.subplot(121).set_title('Elbow Method Graph')
        plt.plot(range(1, len(WCSS) + 1), WCSS)
        plt.grid(b = True)
        plt.subplot(122).set_title('Differences in Each Two Consective Clusters')
        len_differences = len(DIFF)
        X_differences = range(1, len_differences + 1)
        plt.plot(X_differences, DIFF)
        plt.plot(X_differences, np.ones(len_differences)*boundary, 'r')
        plt.plot(X_differences, np.ones(len_differences)*(-boundary), 'r')
        plt.grid()
        plt.show()
```

```python
def clustersMovies(users_cluster, users_data):
    clusters = list(users_cluster['Cluster'])
    each_cluster_movies = list()
    for i in range(len(np.unique(clusters))):
        users_list = list(users_cluster[users_cluster['Cluster'] == i]['user_id'])
        users_movies_list = list()
        for user in users_list:
            users_movies_list.extend(list(users_data[users_data['user_id'] == user]['movie_id']))
        users_movies_counts = list()
        users_movies_counts.extend([[movie, users_movies_list.count(movie)] for movie in np.unique(users_movies_list)])
        each_cluster_movies.append(pd.DataFrame(users_movies_counts, columns=['movie_id', 'Count']).sort_values(by = ['Count
    return each_cluster_movies
cluster_movies = clustersMovies(users_cluster, users_fav_movies)
```

```python
def get_genre_ratings(ratings, movies, genres, column_names):
    genre_ratings = pd.DataFrame()
    for genre in genres:
        genre_movies = data_movies[data_movies['genre'].str.contains(genre) ]
        avg_genre_votes_per_user = ratings_df[ratings_df['movie_id'].isin(genre_movies['movie_id'])].loc[:, ['user_id', 'rati

        genre_ratings = pd.concat([genre_ratings, avg_genre_votes_per_user], axis=1)

    genre_ratings.columns = column_names
    return genre_ratings
```

```python
def draw_clusters(biased_dataset, predictions, cmap='viridis'):
    fig = plt.figure(figsize=(8,8))
    ax = fig.add_subplot(111)
    plt.xlim(0, 5)
    plt.ylim(0, 5)
    ax.set_xlabel('Avg action rating')
    ax.set_ylabel('Avg Horror rating')
    clustered = pd.concat([biased_dataset.reset_index(), pd.DataFrame({'group':predictions})], axis=1)
    plt.scatter(clustered['avg_action_rating'], clustered['avg_horror_rating'], c=clustered['group'], s=20, cmap=cmap)
# Plot
```

```python
def get_most_rated_movies(user_movie_ratings, max_number_of_movies):
    # 1- Count
    user_movie_ratings = user_movie_ratings.append(user_movie_ratings.count(), ignore_index=True)
    # 2- sort
    user_movie_ratings_sorted = user_movie_ratings.sort_values(len(user_movie_ratings)-1, axis=1, ascending=False)
    user_movie_ratings_sorted = user_movie_ratings_sorted.drop(user_movie_ratings_sorted.tail(1).index)
    # 3- slice
    most_rated_movies = user_movie_ratings_sorted.iloc[:, :max_number_of_movies]
    return most_rated_movies
```

```python
def get_users_who_rate_the_most(most_rated_movies, max_number_of_movies):
    # Get most voting users
    # 1- Count
    most_rated_movies['counts'] = pd.Series(most_rated_movies.count(axis=1))
    # 2- Sort
    most_rated_movies_users = most_rated_movies.sort_values('counts', ascending=False)
    # 3- Slice
    most_rated_movies_users_selection = most_rated_movies_users.iloc[:max_number_of_movies, :]
    most_rated_movies_users_selection = most_rated_movies_users_selection.drop(['counts'], axis=1)

    return most_rated_movies_users_selection
```

```python
def sort_by_rating_density(user_movie_ratings, n_movies, n_users):
    most_rated_movies = get_most_rated_movies(user_movie_ratings, n_movies)
    most_rated_movies = get_users_who_rate_the_most(most_rated_movies, n_users)
    return most_rated_movies
```

```python
def draw_movies_heatmap(most_rated_movies_users_selection, axis_labels=True):
    fig = plt.figure(figsize=(15,10))
    ax = plt.gca()

    heatmap = ax.imshow(most_rated_movies_users_selection,  interpolation='nearest', vmin=0, vmax=5, aspect='auto', cmap =

    if axis_labels:
        ax.set_yticks(np.arange(most_rated_movies_users_selection.shape[0]) , minor=False)
        ax.set_xticks(np.arange(most_rated_movies_users_selection.shape[1]) , minor=False)
        ax.invert_yaxis()
        ax.xaxis.tick_top()
        labels = most_rated_movies_users_selection.columns.str[:40]
        ax.set_xticklabels(labels, minor=False)
        ax.set_yticklabels(most_rated_movies_users_selection.index, minor=False)
        plt.setp(ax.get_xticklabels(), rotation=90)
    else:
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)

    ax.grid(False)
    ax.set_ylabel('User_id')
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="5%", pad=0.05)
    cbar = fig.colorbar(heatmap, ticks=[5, 4, 3, 2, 1, 0], cax=cax)
    cbar.ax.set_yticklabels(['5 stars', '4 stars','3 stars','2 stars','1 stars','0 stars'])
    plt.show()
```