DATA ANALYTICS ASSIGNMENT

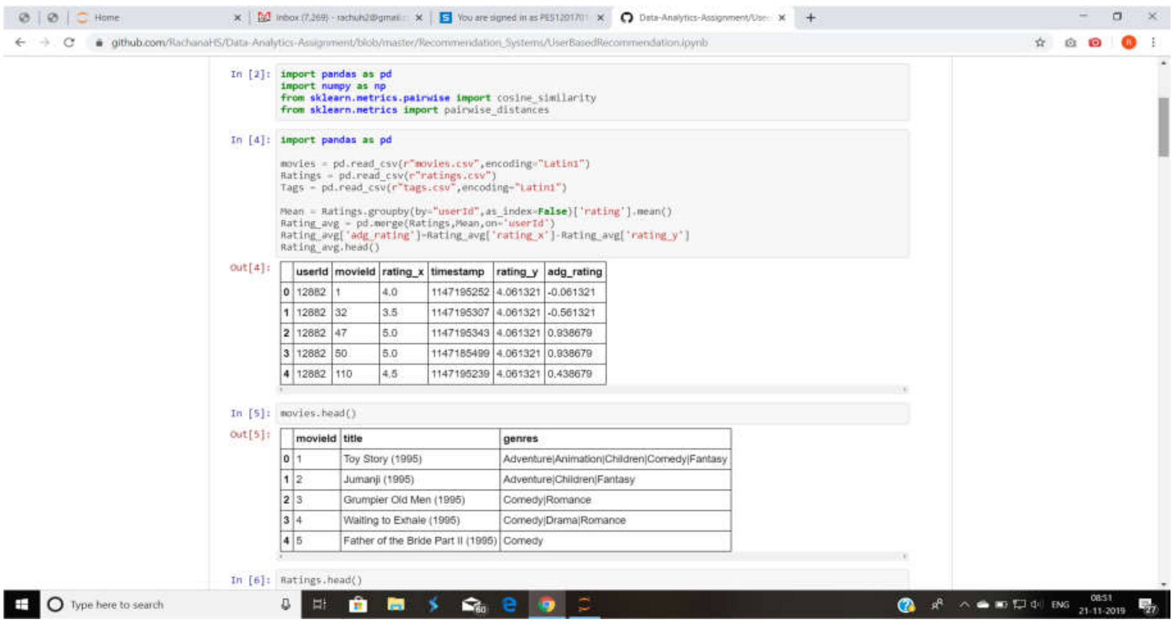
NIVEDITHA C U
PES1201701640
SECTION ‘C’

RACHANA H S
PES1201701726
SECTION ‘E’

PROBLEM STATEMENT:

IMPLEMENTATION OF RECOMMENDATION SYSYTEMS.

SCREENSHOTS



github.com/RachanaHS/Data-Analytics-Assignment/blob/master/Recommendation_Systems/UserBasedRecommendation.ipynb

```
In [2]: import pandas as pd
        import numpy as np
        from sklearn.metrics.pairwise import cosine_similarity
        from sklearn.metrics import pairwise_distances

In [4]: import pandas as pd

        movies = pd.read_csv(r"movies.csv",encoding="Latin1")
        Ratings = pd.read_csv(r"ratings.csv")
        Tags = pd.read_csv(r"tags.csv",encoding="Latin1")

        Mean = Ratings.groupby(by="userId",as_index=False)['rating'].mean()
        Rating_avg = pd.merge(Ratings,Mean,on='userId')
        Rating_avg['adg_rating']=Rating_avg['rating_x']-Rating_avg['rating_y']
        Rating_avg.head()
```

Out[4]:

	userId	movieId	rating_x	timestamp	rating_y	adg_rating
0	12882	1	4.0	1147195252	4.061321	-0.061321
1	12882	32	3.5	1147195307	4.061321	-0.561321
2	12882	47	5.0	1147195343	4.061321	0.938679
3	12882	50	5.0	1147185499	4.061321	0.938679
4	12882	110	4.5	1147195239	4.061321	0.438679

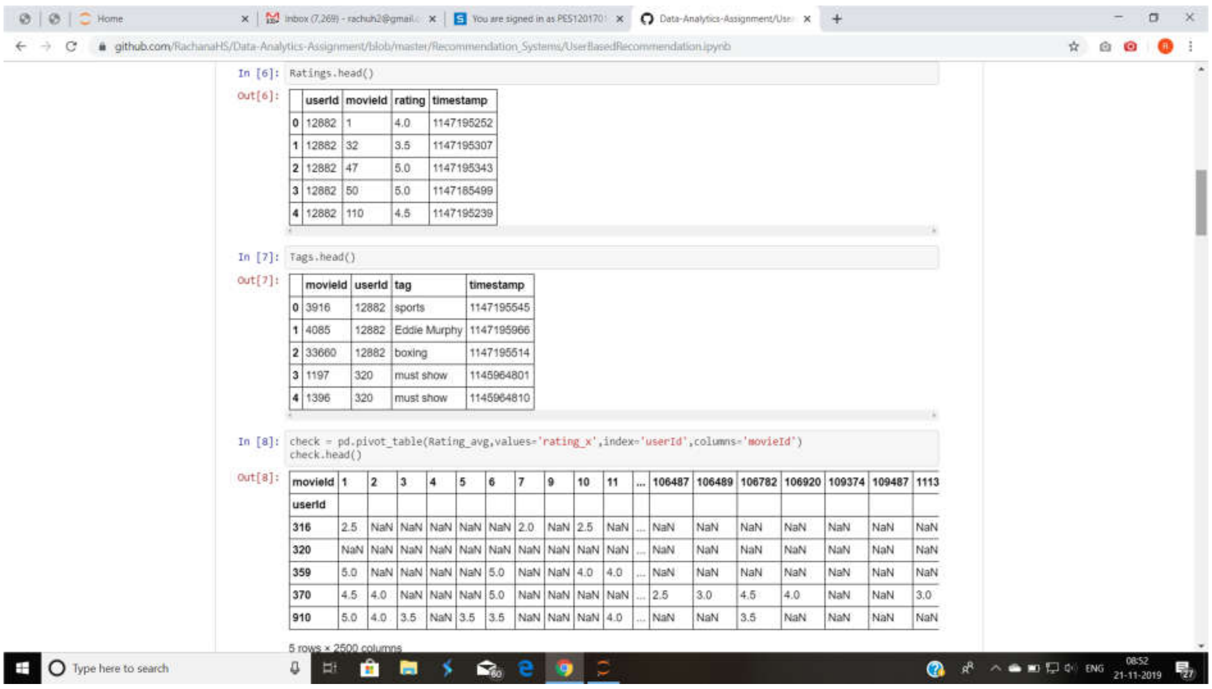
```
In [5]: movies.head()
```

Out[5]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

```
In [6]: Ratings.head()
```

Home
Inbox (7,269) - rachuh2@gmail
You are signed in as PES1201701
Data-Analytics-Assignment/User
Type here to search
ENG
08:51
21-11-2019



Home
Inbox (7,269) - rachuh2@gmail.c
You are signed in as PES120170
Data-Analytics-Assignment/User
github.com/RachanaHS/Data-Analytics-Assignment/blob/master/Recommendation_Systems/UserBasedRecommendation.ipynb
Type here to search
ENG
08:52
21-11-2019

```
In [6]: Ratings.head()
```

Out[6]:

	userId	movieId	rating	timestamp
0	12882	1	4.0	1147195252
1	12882	32	3.5	1147195307
2	12882	47	5.0	1147195343
3	12882	50	5.0	1147185499
4	12882	110	4.5	1147195239

```
In [7]: Tags.head()
```

Out[7]:

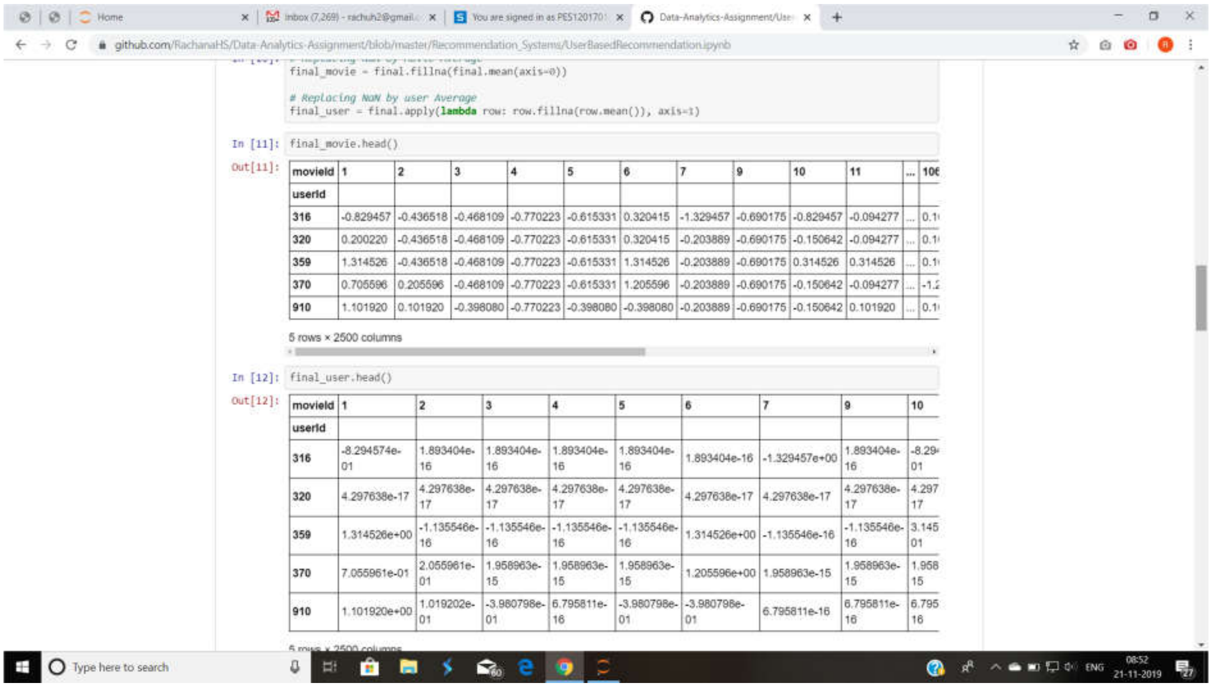
	movieId	userId	tag	timestamp
0	3916	12882	sports	1147195545
1	4085	12882	Eddie Murphy	1147195966
2	33660	12882	boxing	1147195514
3	1197	320	must show	1145964801
4	1396	320	must show	1145964810

```
In [8]: check = pd.pivot_table(Rating_avg,values='rating_x',index='userId',columns='movieId')
        check.head()
```

Out[8]:

movieId	1	2	3	4	5	6	7	9	10	11	...	106487	106489	106782	106920	109374	109487	1113
userId																		
316	2.5	NaN	NaN	NaN	NaN	NaN	2.0	NaN	2.5	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
320	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
359	5.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	4.0	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN
370	4.5	4.0	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	2.5	3.0	4.5	4.0	NaN	NaN	3.0
910	5.0	4.0	3.5	NaN	3.5	3.5	NaN	NaN	NaN	4.0	...	NaN	NaN	3.5	NaN	NaN	NaN	NaN

5 rows × 2500 columns



Home
Inbox (7,269) - rachuh2@gmail.c
You are signed in as PES120170
Data-Analytics-Assignment/User
github.com/RachanaHS/Data-Analytics-Assignment/blob/master/Recommendation_Systems/UserBasedRecommendation.ipynb
Type here to search
ENG
08:52
21-11-2019

```
        final_movie = final.fillna(final.mean(axis=0))

        # Replacing NaN by user Average
        final_user = final.apply(lambda row: row.fillna(row.mean()), axis=1)
```

```
In [11]: final_movie.head()
```

Out[11]:

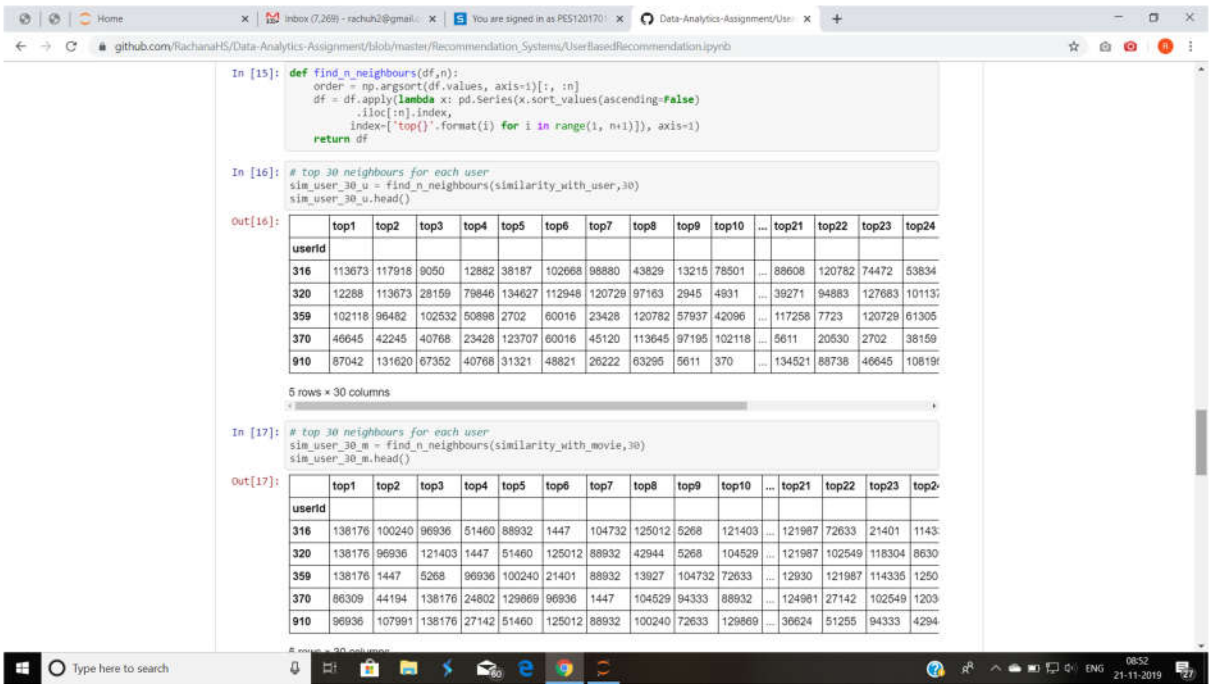
movieId	1	2	3	4	5	6	7	9	10	11	...	106
userId												
316	-0.829457	-0.436518	-0.468109	-0.770223	-0.615331	0.320415	-1.329457	-0.690175	-0.829457	-0.094277	...	0.1
320	0.200220	-0.436518	-0.468109	-0.770223	-0.615331	0.320415	-0.203889	-0.690175	-0.150642	-0.094277	...	0.1
359	1.314526	-0.436518	-0.468109	-0.770223	-0.615331	1.314526	-0.203889	-0.690175	0.314526	0.314526	...	0.1
370	0.705596	0.205596	-0.468109	-0.770223	-0.615331	1.205596	-0.203889	-0.690175	-0.150642	-0.094277	...	-1.2
910	1.101920	0.101920	-0.398080	-0.770223	-0.398080	-0.398080	-0.203889	-0.690175	-0.150642	0.101920	...	0.1

5 rows × 2500 columns

```
In [12]: final_user.head()
```

Out[12]:

movieId	1	2	3	4	5	6	7	9	10
userId									
316	-8.294574e-01	1.893404e-16	1.893404e-16	1.893404e-16	1.893404e-16	1.893404e-16	-1.329457e+00	1.893404e-16	-8.29 01
320	4.297638e-17	4.297638e-17	4.297638e-17	4.297638e-17	4.297638e-17	4.297638e-17	4.297638e-17	4.297638e-17	4.297 17
359	1.314526e+00	-1.135546e-16	-1.135546e-16	-1.135546e-16	-1.135546e-16	1.314526e+00	-1.135546e-16	-1.135546e-16	3.145 01
370	7.055961e-01	2.055961e-01	1.958963e-15	1.958963e-15	1.958963e-15	1.205596e+00	1.958963e-15	1.958963e-15	1.958 15
910	1.101920e+00	1.019202e-01	-3.980798e-01	6.795811e-16	-3.980798e-01	-3.980798e-01	6.795811e-16	6.795811e-16	6.795 16



In [15]:
```python
def find_n_neighbours(df,n):
    order = np.argsort(df.values, axis=1)[:, :n]
    df = df.apply(lambda x: pd.Series(x.sort_values(ascending=False)
           .iloc[:n].index,
          index=['top{}'.format(i) for i in range(1, n+1)]), axis=1)
    return df
```

In [16]:
```python
# top 30 neighbours for each user
sim_user_30_u = find_n_neighbours(similarity_with_user,30)
sim_user_30_u.head()
```

Out[16]:

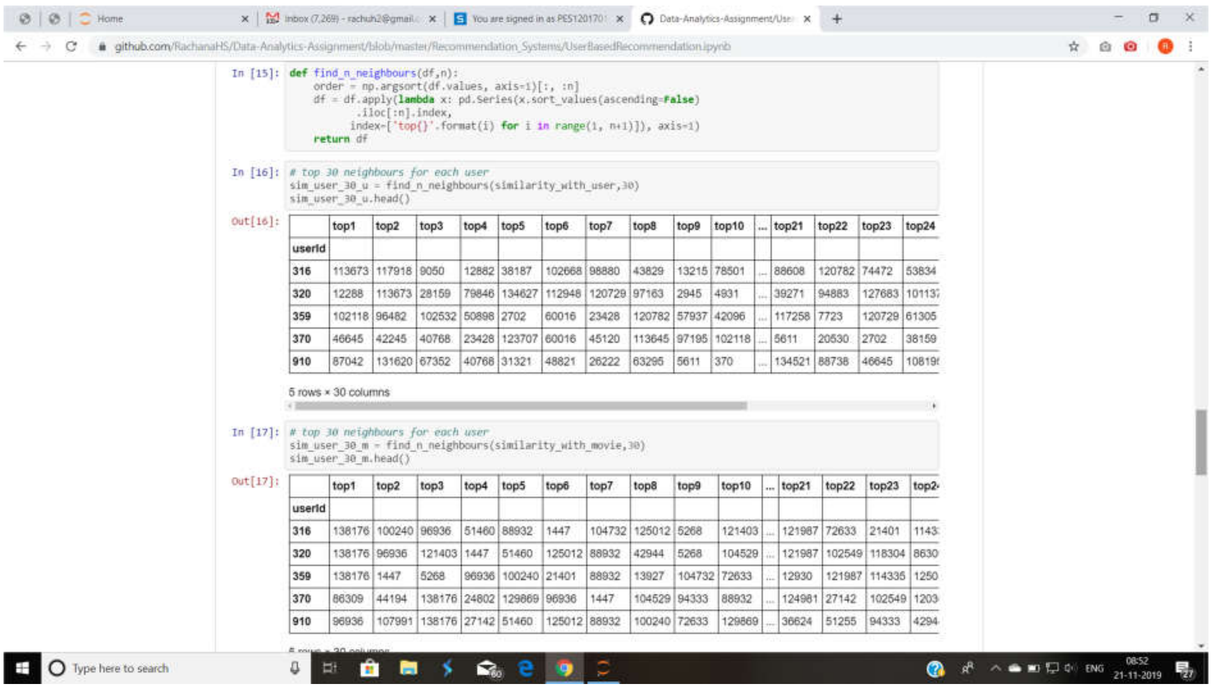
	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23	top24
userId															
316	113673	117918	9050	12882	38187	102668	98880	43829	13215	78501	...	88608	120782	74472	53834
320	12288	113673	28159	79846	134627	112948	120729	97163	2945	4931	...	39271	94883	127683	10113
359	102118	96482	102532	50898	2702	60016	23428	120782	57937	42096	...	117258	7723	120729	61305
370	46645	42245	40768	23428	123707	60016	45120	113645	97195	102118	...	5611	20530	2702	38159
910	87042	131620	67352	40768	31321	48821	26222	63295	5611	370	...	134521	88738	46645	10819

5 rows × 30 columns

In [17]:
```python
# top 30 neighbours for each user
sim_user_30_m = find_n_neighbours(similarity_with_movie,30)
sim_user_30_m.head()
```

Out[17]:

	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23	top2
userId															
316	138176	100240	96936	51460	88932	1447	104732	125012	5268	121403	...	121987	72633	21401	1143
320	138176	96936	121403	1447	51460	125012	88932	42944	5268	104529	...	121987	102549	118304	8630
359	138176	1447	5268	96936	100240	21401	88932	13927	104732	72633	...	12930	121987	114335	1250
370	86309	44194	138176	24802	129869	96936	1447	104529	94333	88932	...	124981	27142	102549	1203
910	96936	107991	138176	27142	51460	125012	88932	100240	72633	129869	...	36624	51255	94333	4294



In [15]:
```python
def find_n_neighbours(df,n):
    order = np.argsort(df.values, axis=1)[:, :n]
    df = df.apply(lambda x: pd.Series(x.sort_values(ascending=False)
           .iloc[:n].index,
          index=['top{}'.format(i) for i in range(1, n+1)]), axis=1)
    return df
```

In [16]:
```python
# top 30 neighbours for each user
sim_user_30_u = find_n_neighbours(similarity_with_user,30)
sim_user_30_u.head()
```

Out[16]:

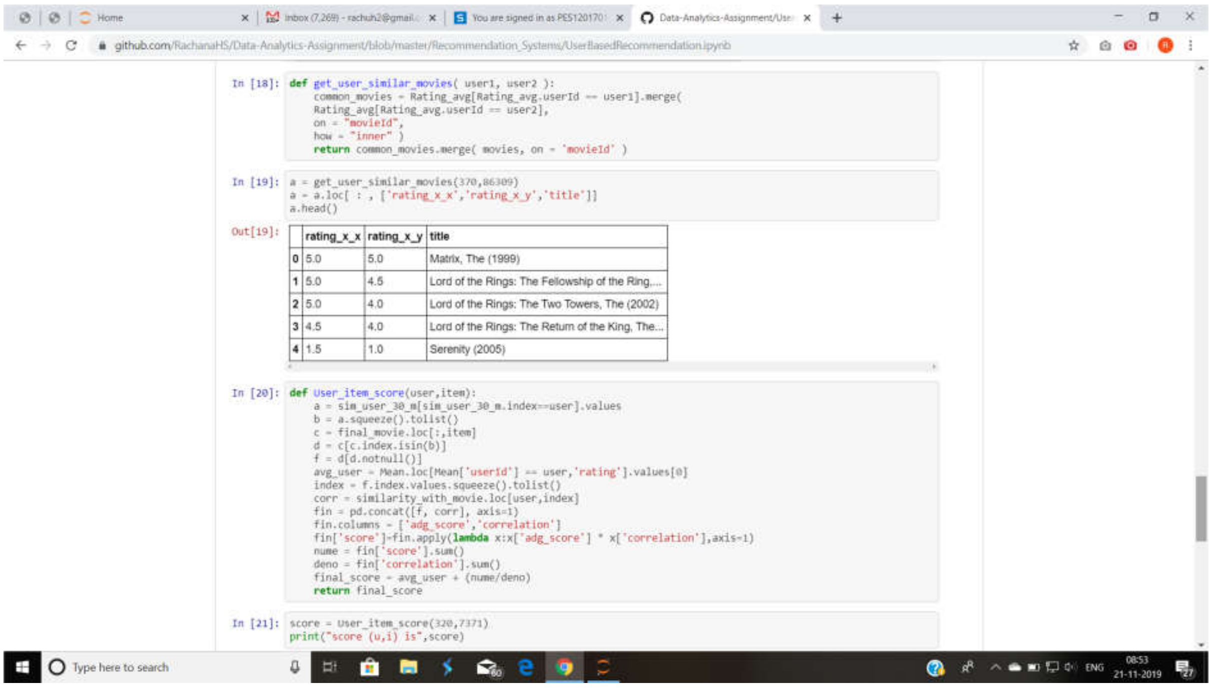
	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23	top24
userId															
316	113673	117918	9050	12882	38187	102668	98880	43829	13215	78501	...	88608	120782	74472	53834
320	12288	113673	28159	79846	134627	112948	120729	97163	2945	4931	...	39271	94883	127683	10113
359	102118	96482	102532	50898	2702	60016	23428	120782	57937	42096	...	117258	7723	120729	61305
370	46645	42245	40768	23428	123707	60016	45120	113645	97195	102118	...	5611	20530	2702	38159
910	87042	131620	67352	40768	31321	48821	26222	63295	5611	370	...	134521	88738	46645	10819

5 rows × 30 columns

In [17]:
```python
# top 30 neighbours for each user
sim_user_30_m = find_n_neighbours(similarity_with_movie,30)
sim_user_30_m.head()
```

Out[17]:

	top1	top2	top3	top4	top5	top6	top7	top8	top9	top10	...	top21	top22	top23	top2
userId															
316	138176	100240	96936	51460	88932	1447	104732	125012	5268	121403	...	121987	72633	21401	1143
320	138176	96936	121403	1447	51460	125012	88932	42944	5268	104529	...	121987	102549	118304	8630
359	138176	1447	5268	96936	100240	21401	88932	13927	104732	72633	...	12930	121987	114335	1250
370	86309	44194	138176	24802	129869	96936	1447	104529	94333	88932	...	124981	27142	102549	1203
910	96936	107991	138176	27142	51460	125012	88932	100240	72633	129869	...	36624	51255	94333	4294



```
In [18]: def get_user_similar_movies( user1, user2 ):
             common_movies = Rating_avg[Rating_avg.userId == user1].merge(
             Rating_avg[Rating_avg.userId == user2],
             on = "movieId",
             how = "inner" )
             return common_movies.merge( movies, on = 'movieId' )
```

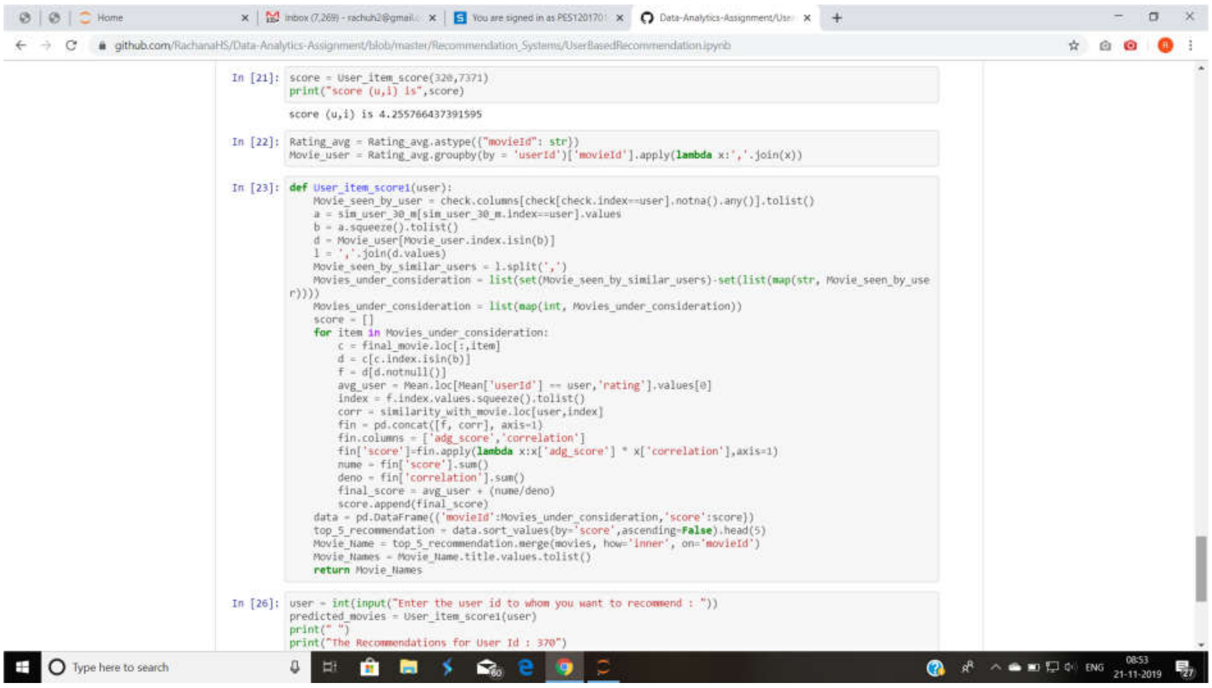
```
In [19]: a = get_user_similar_movies(370,86309)
         a = a.loc[ : , ['rating_x_x','rating_x_y','title']]
         a.head()
```

Out[19]:

	rating_x_x	rating_x_y	title
0	5.0	5.0	Matrix, The (1999)
1	5.0	4.5	Lord of the Rings: The Fellowship of the Ring,...
2	5.0	4.0	Lord of the Rings: The Two Towers, The (2002)
3	4.5	4.0	Lord of the Rings: The Return of the King, The...
4	1.5	1.0	Serenity (2005)

```
In [20]: def User_item_score(user,item):
             a = sim_user_30_m[sim_user_30_m.index==user].values
             b = a.squeeze().tolist()
             c = final_movie.loc[:,item]
             d = c[c.index.isin(b)]
             f = d[d.notnull()]
             avg_user = Mean.loc[Mean['userId'] == user,'rating'].values[0]
             index = f.index.values.squeeze().tolist()
             corr = similarity_with_movie.loc[user,index]
             fin = pd.concat([f, corr], axis=1)
             fin.columns = ['adg_score','correlation']
             fin['score']=fin.apply(lambda x:x['adg_score'] * x['correlation'],axis=1)
             nume = fin['score'].sum()
             deno = fin['correlation'].sum()
             final_score = avg_user + (nume/deno)
             return final_score
```

```
In [21]: score = User_item_score(320,7371)
         print("score (u,i) is",score)
```



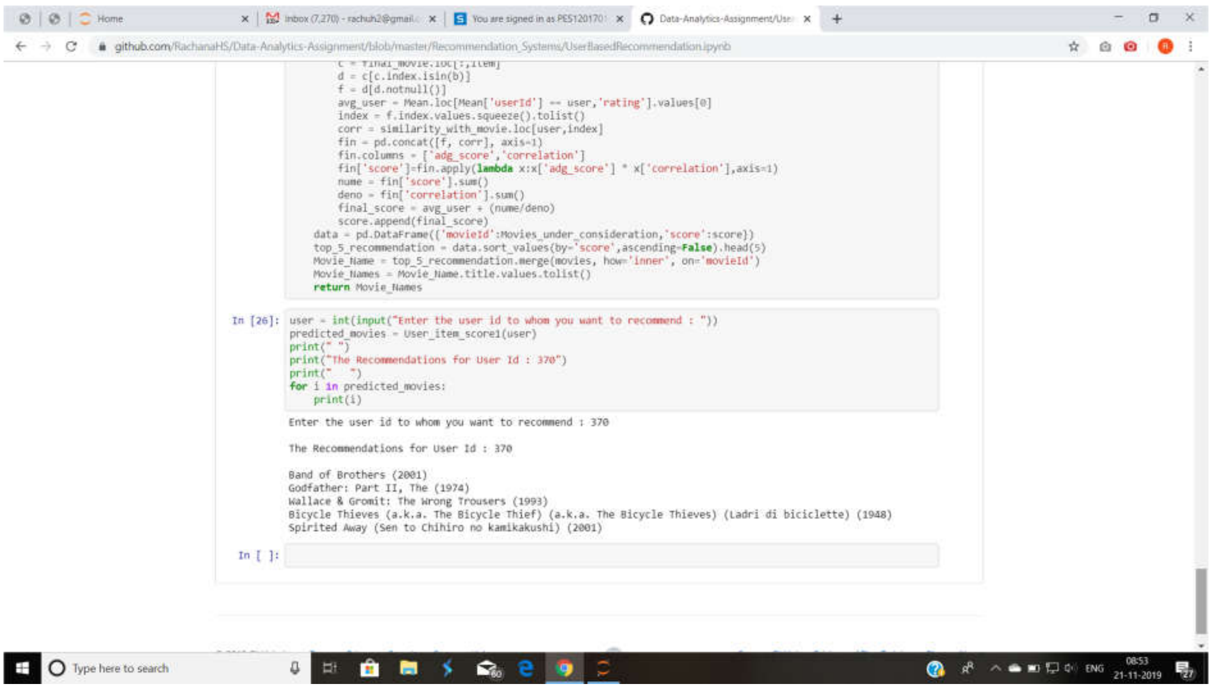
```
In [21]: score = User_item_score(320,7371)
         print("score (u,i) is",score)

         score (u,i) is 4.255766437391595
```

```
In [22]: Rating_avg = Rating_avg.astype({"movieId": str})
         Movie_user = Rating_avg.groupby(by = 'userId')['movieId'].apply(lambda x:','.join(x))
```

```
In [23]: def User_item_score1(user):
             Movie_seen_by_user = check.columns[check[check.index==user].notna().any()].tolist()
             a = sim_user_30_m[sim_user_30_m.index==user].values
             b = a.squeeze().tolist()
             d = Movie_user[Movie_user.index.isin(b)]
             l = ','.join(d.values)
             Movie_seen_by_similar_users = l.split(',')
             Movies_under_consideration = list(set(Movie_seen_by_similar_users)-set(list(map(str, Movie_seen_by_use
         r))))
             Movies_under_consideration = list(map(int, Movies_under_consideration))
             score = []
             for item in Movies_under_consideration:
                 c = final_movie.loc[:,item]
                 d = c[c.index.isin(b)]
                 f = d[d.notnull()]
                 avg_user = Mean.loc[Mean['userId'] == user,'rating'].values[0]
                 index = f.index.values.squeeze().tolist()
                 corr = similarity_with_movie.loc[user,index]
                 fin = pd.concat([f, corr], axis=1)
                 fin.columns = ['adg_score','correlation']
                 fin['score']=fin.apply(lambda x:x['adg_score'] * x['correlation'],axis=1)
                 nume = fin['score'].sum()
                 deno = fin['correlation'].sum()
                 final_score = avg_user + (nume/deno)
                 score.append(final_score)
             data = pd.DataFrame({'movieId':Movies_under_consideration,'score':score})
             top_5_recommendation = data.sort_values(by='score',ascending=False).head(5)
             Movie_Name = top_5_recommendation.merge(movies, how='inner', on='movieId')
             Movie_Names = Movie_Name.title.values.tolist()
             return Movie_Names
```

```
In [26]: user = int(input("Enter the user id to whom you want to recommend : "))
         predicted_movies = User_item_score1(user)
         print(" ")
         print("The Recommendations for User Id : 370")
```


Home
Inbox (7,270) - rachuh2@gmail.c
You are signed in as PES120170
Data-Analytics-Assignment/Use
github.com/RachanaHS/Data-Analytics-Assignment/blob/master/Recommendation_Systems/UserBasedRecommendation.ipynb
Type here to search
ENG
08:53
21-11-2019

```
            c = final_movie.loc[:,item]
            d = c[c.index.isin(b)]
            f = d[d.notnull()]
            avg_user = Mean.loc[Mean['userId'] == user,'rating'].values[0]
            index = f.index.values.squeeze().tolist()
            corr = similarity_with_movie.loc[user,index]
            fin = pd.concat([f, corr], axis=1)
            fin.columns = ['adg_score','correlation']
            fin['score']=fin.apply(lambda x:x['adg_score'] * x['correlation'],axis=1)
            nume = fin['score'].sum()
            deno = fin['correlation'].sum()
            final_score = avg_user + (nume/deno)
            score.append(final_score)
        data = pd.DataFrame({'movieId':Movies_under_consideration,'score':score})
        top_5_recommendation = data.sort_values(by='score',ascending=False).head(5)
        Movie_Name = top_5_recommendation.merge(movies, how='inner', on='movieId')
        Movie_Names = Movie_Name.title.values.tolist()
        return Movie_Names
```

```
In [26]: user = int(input("Enter the user id to whom you want to recommend : "))
         predicted_movies = User_item_score1(user)
         print(" ")
         print("The Recommendations for User Id : 370")
         print("   ")
         for i in predicted_movies:
             print(i)
```

```
Enter the user id to whom you want to recommend : 370

The Recommendations for User Id : 370

Band of Brothers (2001)
Godfather: Part II, The (1974)
Wallace & Gromit: The Wrong Trousers (1993)
Bicycle Thieves (a.k.a. The Bicycle Thief) (a.k.a. The Bicycle Thieves) (Ladri di biciclette) (1948)
Spirited Away (Sen to Chihiro no kamikakushi) (2001)
```

```
In [ ]:
```