optional work

In [ ]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (12,8)
import warnings
warnings.filterwarnings("ignore")

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 10000)

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:
```
movies = pd.read_fwf("/content/drive/Othercomputers/My Laptop/Data Science Studies/GitHub_Desktop/BusinessCase_Data_Explo
ratings =pd.read_fwf("/content/drive/Othercomputers/My Laptop/Data Science Studies/GitHub_Desktop/BusinessCase_Data_Explo
users = pd.read_fwf("/content/drive/Othercomputers/My Laptop/Data Science Studies/GitHub_Desktop/BusinessCase_Data_Explor
```

In [ ]:
```
delimiter ="::"

users = users["UserID::Gender::Age::Occupation::Zip-code"].str.split(delimiter,expand = True)
users.columns = ["UserID","Gender","Age","Occupation","Zipcode"]

users["Age"].replace({"1": "Under 18","18": "18-24","25": "25-34",
                          "35": "35-44","45": "45-49","50": "50-55","56": "56+"},inplace=True)

users["Occupation"] = users["Occupation"].astype(int).replace({0: "other",1: "academic/educator",2: "artist",
                                                              3: "clerical/admin",4: "college/grad student",
                                            5: "customer service",6: "doctor/health care",7: "executive/managerial",
                                            8: "farmer" ,9: "homemaker",10: "K-12 student",11: "lawyer",
                                            12: "programmer",13: "retired",14: "sales/marketing",15: "scientist",
                                            16: "self-employed",17: "technician/engineer",
                                            18: "tradesman/craftsman",19: "unemployed",20: "writer"},
                                           )

delimiter ="::"

ratings = ratings["UserID::MovieID::Rating::Timestamp"].str.split(delimiter,expand = True)
ratings.columns = ["UserID","MovieID","Rating","Timestamp"]


movies.drop(["Unnamed: 1","Unnamed: 2"],axis = 1,inplace=True)



delimiter ="::"

movies = movies["Movie ID::Title::Genres"].str.split(delimiter,expand = True)
movies.columns = ["MovieID","Title","Genres"]


movies.shape,ratings.shape,users.shape
```

Out[11]:
```
((3883, 3), (1000209, 4), (6040, 5))
```

In [ ]:
```
movies["Release_year"] = movies["Title"].str.extract('^(.+)\s\((([0-9]*)\)$',expand = True)[1]
movies["Title"] = movies["Title"].str.split("(").apply(lambda x:x[0])
```

In [ ]:
```
from datetime import datetime
ratings["Watch_Hour"] =ratings["Timestamp"].apply(lambda x:datetime.fromtimestamp(int(x)).hour)
ratings.drop(["Timestamp"],axis = 1,inplace=True)
```

In [ ]:
```
movies.shape,ratings.shape,users.shape
```

Out[14]:
```
((3883, 4), (1000209, 4), (6040, 5))
```

In [ ]:
```
df = users.merge(movies.merge(ratings,on="MovieID",how="outer"),on="UserID",how="outer")
```

In [ ]:
```
(df.isna().sum())/len(df)  * 100
```

Out[9]:
```
UserID          0.017693
Gender          0.017693
Age             0.017693
Occupation      0.017693
Zipcode         0.017693
MovieID         0.000000
Title           0.000000
Genres          0.406443
Release_year    0.377854
Rating          0.017693
Watch_Hour      0.017693
dtype: float64
```

In [ ]:
```
data = df.copy()
data.dropna(inplace= True)
```

In [ ]:
```
data
```

Out[11]:

	UserID	Gender	Age	Occupation	Zipcode	MovieID	Title	Genres	Release_year	Rating	Watch_Hour
0	1	F	Under 18	K-12 student	48067	1	Toy Story	Animation|Children's|Comedy	1995	5	23.0
1	1	F	Under 18	K-12 student	48067	48	Pocahontas	Animation|Children's|Musical|Romance	1995	5	23.0
2	1	F	Under 18	K-12 student	48067	150	Apollo 13	Drama	1995	5	22.0
3	1	F	Under 18	K-12 student	48067	260	Star Wars: Episode IV - A New Hope	Action|Adventure|Fantas	1977	4	22.0
4	1	F	Under 18	K-12 student	48067	527	Schindler's List	Drama|War	1993	5	23.0
...	...	...	...	...	...	...	...	...	...	...	...
1000204	6040	M	25-34	doctor/health care	11106	3683	Blood Simple	Drama|Film-Noir	1984	4	8.0
1000205	6040	M	25-34	doctor/health care	11106	3703	Mad Max 2	Action|Sci-Fi	1981	4	23.0
1000206	6040	M	25-34	doctor/health care	11106	3735	Serpico	Crime|Drama	1973	4	8.0
1000207	6040	M	25-34	doctor/health care	11106	3751	Chicken Run	Animation|Children's|Comedy	2000	4	23.0
1000208	6040	M	25-34	doctor/health care	11106	3819	Tampopo	Comedy	1986	5	23.0

996144 rows × 11 columns

In [ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 996144 entries, 0 to 1000208
Data columns (total 11 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   UserID        996144 non-null  object 
 1   Gender        996144 non-null  object 
 2   Age           996144 non-null  object 
 3   Occupation    996144 non-null  object 
 4   Zipcode       996144 non-null  object 
 5   MovieID       996144 non-null  object 
 6   Title         996144 non-null  object 
 7   Genres        996144 non-null  object 
 8   Release_year  996144 non-null  object 
 9   Rating        996144 non-null  object 
 10  Watch_Hour    996144 non-null  float64
dtypes: float64(1), object(10)
memory usage: 91.2+ MB
```

In [ ]:
```
data.nunique()
```

Out[13]:
```
UserID          6040
Gender             2
Age                7
Occupation        21
Zipcode         3439
MovieID         3682
Title           3633
Genres           358
Release_year      81
Rating             5
Watch_Hour        24
dtype: int64
```

In [ ]:
```
# 6040 unique UserID
# 7 different age groups
# 21 occupations
# 3493 different locations of users
# 3682 unique movies
```

In [ ]:
```
# There are movies available in database , which were never been watched by any user before .
# Thats is the reason we have lots of NaN values in our final dataset.
```

In [ ]:
```
data.shape
```

Out[16]: (996144, 11)

In [ ]:
```
m = movies[["MovieID","Title","Genres"]]
```

In [ ]:
```
m["Genres"] = m["Genres"].str.split("|")
```

In [ ]:
```

m = m.explode("Genres")
m["Genres"] = m["Genres"].replace({"Horro":"Horror","Sci-":"Sci-Fi","Sci":"Sci-Fi","Sci-F":"Sci-Fi","Dr":"Drama",
                     "Documenta":"Documentary","Wester":"Western","Fant":"Fantasy","Chil":"Children's","R":"Romance","D":
                     "Animati":"Animation","Childr":"Children's","Childre":"Children's","Fantas":"Fantasy","Come":"Comedy
                     "Roma":"Romance","A":"Adventure","Children":"Children's","Adventu":"Adventure","Adv":"Adventure",
                      "Wa":"War","Thrille"  :"Thriller","Com":"Comedy","Comed":"Comedy","Acti":"Action",
                        "Advent":"Adventure","Adventur":"Adventure","Thri":"Thriller","Chi":"Children's","Ro":"Romance",
                        "F":"Fantasy","We":"Western","Documen":"Documentary",
                        "Music":"Musical","Children":"Children's" ,"Horr":"Horror","Children'":"Children's","Roman":"Roma
                     })
```

In [ ]:
```
m
```

Out[20]:

	MovieID	Title	Genres
0	1	Toy Story	Animation
0	1	Toy Story	Children's
0	1	Toy Story	Comedy
1	2	Jumanji	Adventure
1	2	Jumanji	Children's
...	...	...	...
3879	3949	Requiem for a Dream	Drama
3880	3950	Tigerland	Drama
3881	3951	Two Family House	Drama
3882	3952	Contender, The	Drama
3882	3952	Contender, The	Thriller

6366 rows × 3 columns

In [ ]:
```

m = pd.crosstab(m["MovieID"],       m["Genres"])
m = pd.DataFrame(np.where(m>=1,1,0),index = m.index,columns=m.columns)
```

In [ ]:
```
m
```

Out[22]:

Genres		Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sc
MovieID																
1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	
10	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
100	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
1000	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
1001	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
994	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
996	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	
997	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
998	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	
999	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	

3858 rows × 19 columns

In [ ]:
```
final_data = data.merge(m,on="MovieID",how="left").drop(["Genres"],axis = 1)
```

In [ ]:
```
path = '/content/drive/Othercomputers/My Laptop/Data Science Studies/final_data.csv '
```

In [ ]:
```
final_data.to_csv(path)
```

The movie with maximum no. of ratings is ___.

In [ ]:
```
final_data.groupby("Title")["Rating"].count().reset_index().sort_values(by="Rating",ascending=False).head(10)
```

Out[61]:

	Title	Rating
125	American Beauty	3428
3092	Star Wars: Episode IV - A New Hope	2991
3093	Star Wars: Episode V - The Empire Strikes Back	2990
3094	Star Wars: Episode VI - Return of the Jedi	2883
1756	Jurassic Park	2672
2837	Saving Private Ryan	2653
3231	Terminator 2: Judgment Day	2649
2070	Matrix, The	2590
255	Back to the Future	2583
2929	Silence of the Lambs, The	2578

In [ ]:
```
final_data.sample(2)
```

Out[67]:

	UserID	Gender	Age	Occupation	Zipcode	MovieID	Title	Release_year	Rating	Watch_Hour		Action	Adventure	Animation
826147	4980	M	25-34	academic/educator	55403	527	Schindler's List	1993	5	4.0	0	0	0	0
69505	469	M	35-44	doctor/health care	55122	3359	Breaking Away	1979	5	21.0	0	0	0	0

In [ ]:
```
m = movies[["MovieID","Title","Genres"]]
m["Genres"] = m["Genres"].str.split("|")
```

In [ ]:
```python
m = m.explode("Genres")
m["Genres"] = m["Genres"].replace({"Horro":"Horror",
                     "Sci-":"Sci-Fi","Sci":"Sci-Fi","Sci-F":"Sci-Fi",
                     "Dr":"Drama",
                     "Documenta":"Documentary",
                     "Wester":"Western",
                     "Fant":"Fantasy","Chil":"Children's","R":"Romance","D":"Drama","Rom":"Romance",
                     "Animati":"Animation",
                     "Childr":"Children's","Childre":"Children's",
                     "Fantas":"Fantasy","Come":"Comedy","Dram":"Drama","S":"Sci-Fi",
                     "Roma":"Romance","A":"Adventure","Children":"Children's",
                     "Adventu":"Adventure",
                      "Adv":"Adventure",
                      "Wa":"War",
                      "Thrille"  :"Thriller"     ,
                      "Com"    :"Comedy"    ,
                      "Comed"    :"Comedy",
                      "Acti"   :"Action",
                        "Advent"   :"Adventure",
                        "Adventur"      :"Adventure",
                        "Thri":"Thriller",
                        "Chi":"Children's",
                        "Ro":"Romance",
                        "F":"Fantasy",
                        "We":"Western",
                        "Documen":"Documentary"        ,
                        "Music":"Musical"          ,
                        "Children":"Children's"  ,
                        "Horr":"Horror"           ,
                     "Children'":"Children's",
                     "Roman":"Romance","Docu":"Documentary","Th":"Thriller","Document":"Documentary"
                     })
```

In [ ]:
```python
merged_data = ratings.merge(users,on="UserID",how="outer").merge(m,on="MovieID",how="outer")
```

In [ ]:
```python
merged_data.groupby("Genres")[["Title","UserID"]].nunique()
```

Out[139]:

Genres	Title	UserID
	8	1344
Action	497	6011
Adventure	281	5894
Animation	104	4794
Children's	244	5280
Comedy	1182	6031
Crime	209	5662
Documentary	124	2237
Drama	1569	6037
Fantasy	63	4617
Film-Noir	44	4150
Horror	332	5299
Musical	113	4752
Mystery	104	5129
Romance	461	5910
Sci-Fi	261	5809
Thriller	481	5989
War	139	5649
Western	68	4100

Number of Movie TItles been raten as per Genre by each type of use Occupation :

In [ ]:
```python
Occupation_genre_count = merged_data.groupby(["Occupation","Genres"])["Title"].nunique().sort_values(ascending=False).res
```

In [ ]:
```python
Occupation_genre_count.pivot(index="Occupation",columns="Genres",values="Title")
```

Out[135]:

Genres		Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	F
Occupation															
K-12 student	5	409	234	97	229	855	135	35	852	62	30	228	94	74	
academic/educator	6	458	262	96	231	1033	175	74	1239	62	40	282	104	94	
artist	6	457	260	99	224	1002	176	69	1203	60	39	284	107	93	
clerical/admin	6	440	245	98	212	969	169	56	1122	61	36	246	104	96	
college/grad student	7	464	266	103	237	1072	185	83	1294	61	41	297	107	98	
customer service	6	451	242	96	202	883	160	40	905	59	36	264	98	87	
doctor/health care	7	450	253	102	228	946	165	68	1137	60	41	285	111	89	
executive/managerial	6	466	269	101	234	1038	180	70	1243	62	40	301	106	94	
farmer	5	322	191	81	151	519	98	8	473	44	21	113	57	43	
homemaker	5	358	211	87	211	808	127	21	789	55	26	159	90	79	
lawyer	5	425	238	91	200	853	154	44	933	55	37	224	96	87	
other	7	472	273	103	240	1081	195	86	1330	63	43	317	108	97	
programmer	6	455	259	100	219	943	163	59	1089	60	36	264	98	94	
retired	3	366	213	63	152	750	139	41	925	45	34	177	89	86	
sales/marketing	6	459	251	97	217	970	175	59	1100	59	36	251	102	93	
scientist	6	424	230	86	176	804	142	42	971	54	31	194	86	87	
self-employed	7	459	258	97	217	994	179	71	1182	57	36	281	99	97	
technician/engineer	6	466	262	101	233	990	172	64	1146	62	38	295	104	96	
tradesman/craftsman	4	414	226	76	182	789	155	36	853	57	33	279	87	83	
unemployed	6	432	235	93	208	894	148	47	958	59	30	266	95	83	
writer	8	461	263	102	233	1053	182	80	1294	62	42	295	107	99	

In [ ]:
```python
pd.set_option("max_colwidth", None)
```

Co-occurance | Frequency Based Recommender System (Apriory)

In [ ]:
```python
frame = data.groupby(["UserID","Title"])["Rating"].mean().unstack().reset_index().fillna(0).set_index('UserID')
```

In [ ]:
```python
frame = (frame > 0).astype(int)
frame.shape
```

Out[30]: (6040, 3633)

In [ ]:
```python
from mlxtend.frequent_patterns import apriori
frequent_itemsets_plus = apriori(frame, min_support=0.2,
                                 use_colnames=True).sort_values('support', ascending=False).reset_index(drop=True)


frequent_itemsets_plus['length'] = frequent_itemsets_plus['itemsets'].apply(lambda x: len(x))
```

In [ ]:
```python

frequent_itemsets_plus.shape
```

Out[32]: (1081, 3)

In [ ]:
```python
from mlxtend.frequent_patterns import association_rules
rules = association_rules(frequent_itemsets_plus, metric ="lift", min_threshold = 0.8)
rules.shape
```

Out[33]: (4610, 9)

```
In [ ]: rules.groupby(["antecedents"])["lift"].max().reset_index().merge(rules,on=["antecedents","lift"])
```

20	(Star Wars: Episode VI - Return of the Jedi , Men in Black , Terminator 2: Judgment Day)	2.047691	(Matrix, The)	0.229470	0.428808	0.201490	0.878066	0.103091	4.684451
21	(Raiders of the Lost Ark , Back to the Future , Star Wars: Episode V - The Empire Strikes Back)	1.832652	(Star Wars: Episode IV - A New Hope)	0.230960	0.495199	0.209603	0.907527	0.095231	5.458898
22	(Raiders of the Lost Ark , Back to the Future , Star Wars: Episode IV - A New Hope)	1.879063	(Star Wars: Episode V - The Empire Strikes Back)	0.225331	0.495033	0.209603	0.930198	0.098056	7.234315
23	(Total Recall , Terminator, The)	2.090372	(Terminator 2: Judgment Day)	0.228808	0.438576	0.209768	0.916787	0.109419	6.746850
24	(Babe)	1.260843	(American Beauty)	0.289901	0.567550	0.207450	0.715591	0.042917	1.520523
25	(Aliens , Alien)	1.826383	(Star Wars: Episode IV - A New Hope)	0.232119	0.495199	0.209934	0.904422	0.094989	5.281578
26	(Star Wars: Episode VI - Return of the Jedi , Aliens , Star Wars: Episode IV - A New Hope)	1.941061	(Star Wars: Episode V - The Empire Strikes Back)	0.215894	0.495033	0.207450	0.960890	0.100576	12.911310
27	(Aliens , Star Wars: Episode IV - A [illegible]	[illegible]	(Star Wars: Episode VI - [illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

```
In [ ]: rules[rules["antecedents"] == rules.loc[4606]["antecedents"]].sort_values(by="lift",ascending=False).head(5)
```

Out[44]:

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
4606	(Fight Club)	(American Beauty)	0.240232	0.56755	0.2	0.832529	1.466884	0.063657	2.582245

Item-Item Similarity Based Rec System

Name the top 3 movies similar to ‘Liar Liar’ on the item-based approach.

```
In [ ]: movies[movies["Title"].str.contains("Liar Liar")]
```

Out[46]:

	MovieID	Title	Genres	Release_year
1455	1485	Liar Liar	Comedy	1997

```
In [ ]: m = movies[["MovieID","Title","Genres"]]

m = m.explode("Genres")
m["Genres"] = m["Genres"].replace({"Horro":"Horror",
                     "Sci-":"Sci-Fi","Sci":"Sci-Fi","Sci-F":"Sci-Fi",
                     "Dr":"Drama",
                     "Documenta":"Documentary",
                     "Wester":"Western",
                     "Fant":"Fantasy","Chil":"Children's","R":"Romance","D":"Drama","Rom":"Romance",
                     "Animati":"Animation",
                     "Childr":"Children's","Childre":"Children's",
                     "Fantas":"Fantasy","Come":"Comedy","Dram":"Drama","S":"Sci-Fi",
                     "Roma":"Romance","A":"Adventure","Children":"Children's",
                     "Adventu":"Adventure",
                      "Adv":"Adventure",
                      "Wa":"War",
                      "Thrille"  :"Thriller"     ,
                      "Com"        :"Comedy"    ,
                      "Comed"          :"Comedy",
                      "Acti"           :"Action",
                        "Advent"         :"Adventure",
                        "Adventur"       :"Adventure",
                        "Thri":"Thriller",
                        "Chi":"Children's",
                        "Ro":"Romance",
                        "F":"Fantasy",
                        "We":"Western",
                        "Documen":"Documentary"       ,
                        "Music":"Musical"           ,
                        "Children":"Children's" ,
                        "Horr":"Horror"            ,
                     "Children'":"Children's",
                     "Roman":"Romance","Docu":"Documentary","Th":"Thriller","Document":"Documentary"
                     })
m = pd.crosstab(m["MovieID"],m["Genres"])
m = pd.DataFrame(np.where(m>=1,1,0),index = m.index,columns=m.columns)
```

```
In [ ]: def Hamming_distance(x1,x2):
          return np.sum(abs(x1-x2))
```

In [ ]:
```
Ranks = []
Query = "1485"
for candidate in m.index:
  if candidate == Query:
    continue
  Ranks.append([Query,candidate,Hamming_distance(m.loc[Query],m.loc[candidate])])
```

In [ ]:
```
Ranks = pd.DataFrame(Ranks,columns=["Query","Candidate","Hamming_distance"])
Ranks = Ranks.merge(movies[['MovieID', 'Title']], left_on='Query', right_on='MovieID').rename(columns={'Title': 'query_ti
Ranks = Ranks.merge(movies[['MovieID', 'Title']], left_on='Candidate', right_on='MovieID').rename(columns={'Title': 'canc
Ranks = Ranks.sort_values(by=['Query', 'Hamming_distance'])
```

In [ ]:
```
Ranks.head()
```

Out[52]:

	Query	Candidate	Hamming_distance	query_tittle	candidate_tittle
4	1485	1001	0	Liar Liar	Associate, The
5	1485	1002	0	Liar Liar	Ed's Next Move
13	1485	101	0	Liar Liar	Bottle Rocket
24	1485	102	0	Liar Liar	Mr. Wrong
25	1485	1020	0	Liar Liar	Cool Runnings

Collaborative Filtering :

In [ ]:
```
user_movie_ratings = ratings.pivot(index ="UserID",
              columns = "MovieID",
              values ="Rating").fillna(0)
```

In [ ]:
```
user_movie_ratings.shape
```

Out[54]:
```
(6040, 3706)
```

In [ ]:
```
# 6040 users # 3706 movies
```

In [ ]:
```
ratings.shape, users.shape
```

Out[56]:
```
((1000209, 4), (6040, 5))
```

In [ ]:
```
rm_raw = ratings[['UserID', 'MovieID', 'Rating']].copy()
rm_raw.columns = ['UserId', 'ItemId', 'Rating']  # Lib requires specific column names
```

In [ ]:
```
rm_raw.Rating = rm_raw.Rating.astype(int)
rm_raw.UserId = rm_raw.UserId.astype(int)
rm_raw.ItemId = rm_raw.ItemId.astype(int)
```

In [ ]:
```
rm_raw.nunique()
```

Out[133]:
```
UserId    6040
ItemId    3706
Rating       5
dtype: int64
```

In [ ]:
```
# !pip install cmfrec
```

In [ ]:
```
from cmfrec import CMF

model = CMF(k=5, lambda_=0.1, user_bias=False, item_bias=False, verbose=False)
model.fit(rm_raw)
```

Out[111]:
```
Collective matrix factorization model
(explicit-feedback variant)
```

In [ ]:
```
rm_raw.shape,model.A_.shape,model.B_.shape
```

Out[112]:
```
((1000209, 3), (6040, 5), (3706, 5))
```

In [ ]:
```
model.A_.shape,model.B_.T.shape
```
Out[113]:
```
((6040, 5), (5, 3706))
```

In [ ]:
```
model.topN(user=8, n=10)
```
Out[126]:
```
array([2323,  296, 1136, 1089, 3421,  858, 1198, 3552,  260, 2776])
```

In [ ]:
```
movies_to_recommend = model.topN(user=1, n=10)
movies_to_recommend = movies_to_recommend[movies_to_recommend<3706]
movies_to_recommend

movies.MovieID = movies.MovieID.astype(int)
movies.loc[movies_to_recommend]
```
Out[143]:

	MovieID	Title	Genres	Release_year
1421	1446	Kolya	Comedy	1996
2099	2168	Dance with Me	Drama|Romance	1998
2776	2845	White Boys	Drama	1999
2211	2280	Clay Pigeons	Crime	1998
1022	1035	Sound of Music, The	Musical	1965
2018	2087	Peter Pan	Animation|Children's|Fantasy|Musical	1953
1951	2020	Dangerous Liaisons	Drama|Romance	1988
1028	1041	Secrets & Lies	Drama	1996
2035	2104	Tex	Drama	1982
595	599	Wild Bunch, The	Western	1969

In [ ]:
```
!pip install scikit-surprise
```
```
Looking in indexes: https://pypi.org/simple, (https://pypi.org/simple,) https://us-python.pkg.dev/colab-wheels/public/s
imple/ (https://us-python.pkg.dev/colab-wheels/public/simple/)
Collecting scikit-surprise
  Downloading scikit-surprise-1.1.3.tar.gz (771 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 772.0/772.0 KB 14.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: joblib>=1.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise) (1.2.0)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise) (1.21.6)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.8/dist-packages (from scikit-surprise) (1.7.3)
Building wheels for collected packages: scikit-surprise
  Building wheel for scikit-surprise (setup.py) ... done
  Created wheel for scikit-surprise: filename=scikit_surprise-1.1.3-cp38-cp38-linux_x86_64.whl size=3366465 sha256=a3d8
dd2a966666053cdeab5b9aa99d54786198cded3c182220c579d2074fab3b
  Stored in directory: /root/.cache/pip/wheels/af/db/86/2c18183a80ba05da35bf0fb7417aac5cddbd93bcb1b92fd3ea
Successfully built scikit-surprise
Installing collected packages: scikit-surprise
Successfully installed scikit-surprise-1.1.3
```

In [ ]:
```
final_data= pd.read_csv("/content/drive/Othercomputers/My Laptop/Data Science Studies/GitHub_Desktop/BusinessCase_Data_Ex

from surprise import KNNWithMeans
from surprise import Dataset
from surprise import accuracy
from surprise.model_selection import train_test_split
from surprise import Reader

## The Reader class is used to parse a file containing ratings.It orders the data in format of (userid,title,rating) and
reader = Reader(rating_scale=(0.5 , 5))
# The columns must correspond to user id, item id and ratings (in that order).
data = Dataset.load_from_df(final_data[["UserID","MovieID","Rating"]], reader) # Loading the data as per the format
```

In [ ]:
```
data
```
Out[26]:
```
<surprise.dataset.DatasetAutoFolds at 0x7f814324beb0>
```

In [ ]:
```
anti_set = data.build_full_trainset().build_anti_testset()
```

In [ ]:
```
trainset, testset = train_test_split(data, test_size=.15) # Splitting the data
```

User Based Collaborative Filtering :

In [ ]:
```
algo = KNNWithMeans(k = 50, sim_options={'name': 'cosine', 'user_based': True})

# K value represents the (max) number of neighbors to take into account for aggregation. Example for every item it gives
# There are many similarity options to calculate the similarity between the neighbors. Here, we have used the cosine simi
# when user_based = True then it performs user based collaborative filtering

algo.fit(trainset) #fitting the train dataset
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

Out[145]:
```
<surprise.prediction_algorithms.knns.KNNWithMeans at 0x7f2ed8392df0>
```

In [ ]:
```
# run the trained model against the testset
test_pred = algo.test(testset)
```

In [ ]:
```
test_pred[0]
```

Out[147]:
```
Prediction(uid='770', iid='2383', r_ui=1.0, est=0.8775986827947615, details={'actual_k': 50, 'was_impossible': False})
```

In [ ]:
```
# get RMSE on test set
print("User-based Model : Test Set")
accuracy.rmse(test_pred, verbose=True)
```

```
User-based Model : Test Set
RMSE: 0.9345
```

Out[148]:
```
0.9344753997438834
```

In [ ]:
```
accuracy.mae(test_pred, verbose=True)
```

```
MAE:  0.7433
```

Out[149]:
```
0.7432764090468391
```

In [ ]:
```
# we can query for specific predicions
uid = str(1)  # raw user id
iid = str(1)  # raw item id
```

In [ ]:
```
pred = algo.predict(uid, iid, verbose=True)
```

```
user: 1          item: 1          r_ui = None   est = 4.88   {'actual_k': 50, 'was_impossible': False}
```

In [ ]:
```
# anti_pre = algo.test(anti_set)
# pred_df = pd.DataFrame(anti_pre).merge(movies , left_on = ['iid'], right_on = ['MovieID'])
# pred_df = pd.DataFrame(pred_df).merge(users , left_on = ['uid'], right_on = ['UserID'])
```

Item Based Collaborative Filtering :

In [ ]:
```
# K value represents the (max) number of neighbors to take into account for aggregation. Example for every item it gives
# There are many similarity options to calculate the similarity between the neighbors . Here, we have used the cosine sim
# when user_based = False then it performs item based collaborative filtering

algo_i = KNNWithMeans(k=30, sim_options={'name': 'cosine', 'user_based': False})
algo_i.fit(trainset)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
```

Out[153]:
```
<surprise.prediction_algorithms.knns.KNNWithMeans at 0x7f2ed36d0be0>
```

In [ ]:
```
test_pred = algo_i.test(testset)
```

In [ ]:
```
test_pred[0]
```

Out[155]:
```
Prediction(uid='770', iid='2383', r_ui=1.0, est=1.1141077593830107, details={'actual_k': 30, 'was_impossible': False})
```

In [ ]:
```
# get RMSE on test set
print("Item-based Model : Test Set")
accuracy.rmse(test_pred, verbose=True)
```

```
Item-based Model : Test Set
RMSE: 0.8926
```

Out[156]:
```
0.8926413172784623
```

In [ ]:
```python
# we can query for specific predicions
uid = str(196)  # raw user id
iid = str(303)  # raw item id
```

In [ ]:
```python
pred = algo_i.predict(uid, iid, verbose=True)
```

```
user: 196        item: 303        r_ui = None   est = 2.90   {'actual_k': 29, 'was_impossible': False}
```

In [ ]:
```python
# final_data[final_data["MovieID"]=="984"]
```

In [ ]:
```python

tsr_inner_id = algo_i.trainset.to_inner_iid("1485") #Considering the movieId 1485 : Liar Liar

tsr_neighbors = algo_i.get_neighbors(tsr_inner_id, k=5) #Getting the 5 nearest neighbors for movieId 1
```

In [ ]:
```python

movies[movies.MovieID.isin([algo.trainset.to_raw_iid(inner_id)
                       for inner_id in tsr_neighbors])] #Displaying the 5 nearest neighbors to the Liar Liar
```

Out[161]:

	MovieID	Title	Genres	Release_year
127	129	Pie in the Sky	Comedy|Romance	1995
968	980	In the Line of Duty 2	Action	1987
1645	1692	Alien Escape	Horror|Sci-Fi	1995
2089	2158	Henry: Portrait of a Serial Killer, Part 2	Crime|Horror	1996
2500	2569	Among Giants	Drama|Romance	1998

Matrix Factorisation:

In [ ]:
```python
from surprise import SVD
from surprise.model_selection import cross_validate
```

In [ ]:
```python
svd = SVD() #Suprise library uses the SVD algorithm to perform the matrix factorisation where as other libraries uses ALS
cross_validate(svd, data, measures=['rmse','mae'], cv = 5 , return_train_measures=True,verbose=True)
##The dataset is divided into train and test and with 5 folds the rmse has been calculated
```

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8768  0.8736  0.8753  0.8726  0.8729  0.8742  0.0016
MAE (testset)     0.6885  0.6858  0.6883  0.6852  0.6862  0.6868  0.0014
RMSE (trainset)   0.6691  0.6702  0.6712  0.6692  0.6702  0.6700  0.0008
MAE (trainset)    0.5298  0.5303  0.5310  0.5297  0.5302  0.5302  0.0005
Fit time          13.61   14.77   14.17   14.88   14.20   14.33   0.46
Test time         2.63    1.64    4.00    3.98    1.65    2.78    1.05
```

Out[30]:
```
{'test_rmse': array([0.87679055, 0.87363147, 0.87525437, 0.87256188, 0.8728605 ]),
 'train_rmse': array([0.6691445 , 0.67017315, 0.67120119, 0.6692182 , 0.67018469]),
 'test_mae': array([0.68851652, 0.68582647, 0.68829271, 0.68518788, 0.68615884]),
 'train_mae': array([0.52975006, 0.53025507, 0.53101726, 0.52965381, 0.53016147]),
 'fit_time': (13.613670587539673,
  14.765673160552979,
  14.170423984527588,
  14.876917362213135,
  14.199043035507202),
 'test_time': (2.6307239532470703,
  1.639423131942749,
  3.9973740577697754,
  3.9794485569000244,
  1.650454044342041)}
```

In [ ]:
```python

import pandas as pd
final_data= pd.read_csv("/content/drive/Othercomputers/My Laptop/Data Science Studies/GitHub_Desktop/BusinessCase_Data_Ex
from surprise import Dataset

from surprise import SVD

from surprise import Reader

## The Reader class is used to parse a file containing ratings.It orders the data in format of (userid,title,rating) and
reader = Reader(rating_scale=(0.5 , 5))
# The columns must correspond to user id, item id and ratings (in that order).
data = Dataset.load_from_df(final_data[["UserID","MovieID","Rating"]], reader) # Loading the data as per the format
```

In [ ]:
```python
svd = SVD(n_factors =10)
trainset = data.build_full_trainset()
svd.fit(trainset) ##Fitting the trainset with the help of svd
```

Out[6]: <surprise.prediction_algorithms.matrix_factorization.SVD at 0x7fc169469d30>

In [ ]:
```python
svd.pu.shape , svd.qi.shape #pu gives the embeddings of Users and qi gives the embeddings of Items.
```

Out[7]: ((6040, 10), (3682, 10))

In [ ]:
```python
#Storing all the movie titles in items
items = movies['Title'].unique()
##Considering the user '662'
test = [[662, iid, 4] for iid in items]
##Finding the user predictions(ratings) for all the movies
predictions = svd.test(test)
pred = pd.DataFrame(predictions)
```

In [ ]:
```python
a = pred.sort_values(by='est', ascending=False) ##Sorting the values based on the estimated predictions
a[0:10] ##TOP 10
```

Out[16]:

	uid	iid	r_ui	est	details
0	662	Toy Story	4	3.385119	{'was_impossible': False}
2560	662	It Came from Hollywood	4	3.385119	{'was_impossible': False}
2548	662	House of Frankenstein	4	3.385119	{'was_impossible': False}
2549	662	Frankenstein	4	3.385119	{'was_impossible': False}
2550	662	Son of Frankenstein	4	3.385119	{'was_impossible': False}
2551	662	Ghost of Frankenstein, The	4	3.385119	{'was_impossible': False}
2552	662	Frankenstein Meets the Wolf Man	4	3.385119	{'was_impossible': False}
2553	662	Curse of Frankenstein, The	4	3.385119	{'was_impossible': False}
2554	662	Son of Dracula	4	3.385119	{'was_impossible': False}
2555	662	Wolf Man, The	4	3.385119	{'was_impossible': False}

In [ ]:
```python
testset = trainset.build_anti_testset()
predictions_svd = svd.test(testset) #Predicting for the test set
```

In [ ]:
```python
from surprise import accuracy
```

In [ ]:
```python
print('SVD - RMSE:', accuracy.rmse(predictions_svd, verbose=False))
print('SVD - MAE:', accuracy.mae(predictions_svd, verbose=False))
```

```
SVD - RMSE: 0.7034125702508484
SVD - MAE: 0.5440720263794291
```