Problem Statement : To Build a personalized movie recommendations based on ratings given by a user and other users similar to them

In [ ]:

```
Actionable Insights -
1. There are more Male users than Female users
2. Age group between 25-35 watches a lot of movies
3. College/grad students watch more movies
4. Highest number of movies are Drama, Comedy and Action
5. There are 3841 unique movies
6. From the year 1992 to 2000 there were more number of movies releases
7. Rating 3 and 4 has the highest count of rating
8. User ID 283, 2339 and 3324 has the highest average rating
9. There are total of 3439 unique zip codes
10. Movie ID that received the most ratings are- 1580,1197 and 1647
11. USer ID 4169, 1680 and 4277 has provided most number of rating

Recommendations :

1. Top 5 movie recommendations based on genres for 'Liar Liar' is shown in cell number 59
2. Top 5 similar movies based on pearson corr (rating) is shown in cell number 63
3.  Top 5 movies based on cosine similarity (genres) is shown in cell 71
4. Top 5 users based on cosine similarity (gender, age, Occupation) is shown in cell 84
5. Top 5 movies based on cosine similarity (rating) is shown in cell 87
6. Top 5 user id based on cosine similarity (rating) is shown in cell 91
7. Top 5 movies based on cosine similarity (Genres + average rating of the movie) using KNN is shown in cell 111
8.Top 5 movies based on cosine similarity (rating) using KNN is shown in cell 117
9. Top 5 movies based on matrix factorization  with d=4 is shown in cell 126
10. MAPE and RMSE error for the model is 0.3460 and 1.165 respectively
11. Top 5 movies based on embeddings for item-item similarity is shown in cell  135
12. Top 5 user ID based on embeddings for item-item similarity is shown in cell  141
13. Plot for Matric factorization for d=2 for user and movie is shown in cell 148 and 149. Movie average rating are different in each quad
14. PCA visualization (d=2) for the movies and user id does not show any results when compared with matrix factorization
15. Top 10 movies based on User-based approach is shown in cell 185
```

In [2]:

```
import pandas as pd
import numpy as np
import datetime as dt
import regex as re
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from scipy.spatial.distance import cosine
from sklearn.metrics import pairwise_distances
from scipy.sparse import csr_matrix
from sklearn.metrics import mean_squared_error as mse
from sklearn.preprocessing import OneHotEncoder
from sklearn.metrics import mean_absolute_percentage_error as mape
from sklearn.metrics import mean_squared_error as mse
from math import sqrt
from cmfrec import CMF
import random
from functools import reduce
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings("ignore")
from sklearn.neighbors import NearestNeighbors
from cmfrec import CMF
```

Importing Data

In [3]:

```
mov=pd.read_fwf('zee-movies.dat',encoding='ISO-8859-1')
rat=pd.read_fwf('zee-ratings.dat',encoding='ISO-8859-1')
use=pd.read_fwf('zee-users.dat',encoding='ISO-8859-1')
```

Regex expression for extracting year and Title from movie

In [4]:

```
def expression(x):
    mystring='\(\d\d\d\d\)'
    if re.search('(\d\d\d\d)', x) is None:
        return 0
    ans=re.search('(\d\d\d\d)', x).group(0)
    return ans
def expression1(x):
    mystring='\(\d\d\d\d\)'
    x=re.sub(mystring,'',x)
    return x
```

In [5]:

```
mov.head(5)
```

Out[5]:

	Movie ID::Title::Genres	Unnamed: 1	Unnamed: 2
0	1::Toy Story (1995)::Animation|Children's|Comedy	NaN	NaN
1	2::Jumanji (1995)::Adventure|Children's|Fantasy	NaN	NaN
2	3::Grumpier Old Men (1995)::Comedy|Romance	NaN	NaN
3	4::Waiting to Exhale (1995)::Comedy|Drama	NaN	NaN
4	5::Father of the Bride Part II (1995)::Comedy	NaN	NaN

In [6]:

```
rat.head(5)
```

Out[6]:

	UserID::MovieID::Rating::Timestamp
0	1::1193::5::978300760
1	1::661::3::978302109
2	1::914::3::978301968
3	1::3408::4::978300275
4	1::2355::5::978824291

In [7]:

```
use.head(5)
```

Out[7]:

	UserID::Gender::Age::Occupation::Zip-code
0	1::F::1::10::48067
1	2::M::56::16::70072
2	3::M::25::15::55117
3	4::M::45::7::02460
4	5::M::25::20::55455

Formatting the Data, Data Cleaning, and Feature Engineering

In [8]:

```
mov.drop(['Unnamed: 1','Unnamed: 2'],axis=1,inplace=True)
mov['list']=mov['Movie ID::Title::Genres'].apply(lambda x: x.split('::'))
mov=pd.DataFrame(mov['list'].to_list(),columns=['Movie ID', 'Title','Genres'])
mov['Title_']=mov['Title'].apply(lambda x:expression1(x).strip())
mov['Release_year_']=mov['Title'].apply(lambda x:expression(x)).astype('int')
mov['Release_year_']=mov['Release_year_'].replace(to_replace=0, value=mov['Release_year_'].median())
mov['Genres']=mov['Genres'].fillna(value=mov['Genres'].value_counts().reset_index()['index'][0])
mov.drop('Title',axis=1,inplace=True)
mov.columns=['Movie ID', 'Genres', 'Title', 'Release_year']
mov['Genres']=mov['Genres'].apply(lambda x: x.split('|'))
mov=mov.explode('Genres')
```

In [9]:

```
rat['list']=rat['UserID::MovieID::Rating::Timestamp'].apply(lambda x: x.split('::'))
rat=pd.DataFrame(rat['list'].to_list(),columns=['UserID', 'MovieID','Rating','Timestamp'])
rat['Ratings_Time']=pd.to_datetime(rat['Timestamp'],unit='s').dt.time
rat['Ratings_Date']=pd.to_datetime(rat['Timestamp'],unit='s').dt.date
rat.drop('Timestamp',axis=1,inplace=True)
```

In [10]:

```
use['list']=use['UserID::Gender::Age::Occupation::Zip-code'].apply(lambda x: x.split('::'))
use=pd.DataFrame(use['list'].to_list(),columns=['UserID', 'Gender','Age','Occupation','Zip-code'])
age={'1':"Under 18",'18':"18-24",'25':"25-34",'35':"35-44",'45':"45-49",'50':"50-55",'56':"56+"}
occ={'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"}
use['Age']=use['Age'].map(age)
use['Occupation']=use['Occupation'].map(occ)
```

In [11]:

```
mov.info()
# No null values in this data frame
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6366 entries, 0 to 3882
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   Movie ID      6366 non-null   object
 1   Genres        6366 non-null   object
 2   Title         6366 non-null   object
 3   Release_year  6366 non-null   int32 
dtypes: int32(1), object(3)
memory usage: 223.8+ KB
```

In [12]:

```
rat.info()
# No null values in this data frame
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 5 columns):
 #   Column        Non-Null Count    Dtype 
---  ------        --------------    ----- 
 0   UserID        1000209 non-null  object
 1   MovieID       1000209 non-null  object
 2   Rating        1000209 non-null  object
 3   Ratings_Time  1000209 non-null  object
 4   Ratings_Date  1000209 non-null  object
dtypes: object(5)
memory usage: 38.2+ MB
```

In [13]:

```
use.info()
# No null values in this data frame
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   UserID      6040 non-null   object
 1   Gender      6040 non-null   object
 2   Age         6040 non-null   object
 3   Occupation  6040 non-null   object
 4   Zip-code    6040 non-null   object
dtypes: object(5)
memory usage: 236.1+ KB
```

Converting each data types

In [14]:

```
mov['Movie ID']=mov['Movie ID'].astype(int)
rat['MovieID']=rat['MovieID'].astype(int)
rat['UserID']=rat['UserID'].astype(int)
rat['Rating']=rat['Rating'].astype(int)
use['UserID']=use['UserID'].astype(int)
```

In [15]:

```
mov.isna().sum()
# No Null values in Movies dataframe
```

Out[15]:

```
Movie ID        0
Genres          0
Title           0
Release_year    0
dtype: int64
```

In [16]:

```
rat.isna().sum()
# No Null values in ratings dataframe
```

Out[16]:

```
UserID          0
MovieID         0
Rating          0
Ratings_Time    0
Ratings_Date    0
dtype: int64
```

In [17]:

```
use.isna().sum()
# No Null values in users dataframe
```

Out[17]:

```
UserID        0
Gender        0
Age           0
Occupation    0
Zip-code      0
dtype: int64
```

In [18]:

```
mov.describe()
# There are few outliers in release year
```

Out[18]:

	Movie ID	Release_year
count	6366.000000	6366.000000
mean	1949.934025	1989.642004
std	1136.665617	149.042331
min	1.000000	1600.000000
25%	976.000000	1983.000000
50%	1961.500000	1994.000000
75%	2910.750000	1997.000000
max	3952.000000	9000.000000

In [19]:

```
mov.describe(include=object)
# Genres Drama are the most
```

Out[19]:

	Genres	Title
count	6366	6366
unique	63	3841
top	Drama	Mummy, The
freq	1601	7

In [20]:

```
rat.describe()
# Mean and median rating are close. There are no much outliers in rating
```

Out[20]:

	UserID	MovieID	Rating
count	1.000209e+06	1.000209e+06	1.000209e+06
mean	3.024512e+03	1.865540e+03	3.581564e+00
std	1.728413e+03	1.096041e+03	1.117102e+00
min	1.000000e+00	1.000000e+00	1.000000e+00
25%	1.506000e+03	1.030000e+03	3.000000e+00
50%	3.070000e+03	1.835000e+03	4.000000e+00
75%	4.476000e+03	2.770000e+03	4.000000e+00
max	6.040000e+03	3.952000e+03	5.000000e+00

In [21]:

```
mov['Release_year']=mov['Release_year'].clip(1900,2010)
# Cliping release year
```

In [22]:

```
rat.describe(include=object)
```

Out[22]:

	Ratings_Time	Ratings_Date
count	1000209	1000209
unique	81910	1040
top	03:30:29	2000-11-20
freq	57	63654

In [23]:

```
use.describe()
```

Out[23]:

	UserID
count	6040.000000
mean	3020.500000
std	1743.742145
min	1.000000
25%	1510.750000
50%	3020.500000
75%	4530.250000
max	6040.000000

In [24]:

```
use.describe(include=object)
# There are more number of Male users
# Age group between 25-34 watch a lot of movies
# college/grad student tend to watch more movies
```

Out[24]:

	Gender	Age	Occupation	Zip-code
count	6040	6040	6040	6040
unique	2	7	21	3439
top	M	25-34	college/grad student	48104
freq	4331	2096	759	19

EDA

In [25]:

```
G={'':'Comedy',
 'A':'Action',
 'Acti':'Action',
 'Action':'Action',
 'Adv':'Adventure',
 'Advent':'Adventure',
 'Adventu':'Adventure',
 'Adventur':'Adventure',
 'Adventure':'Adventure',
 'Animati':'Adventure',
 'Animation':'Adventure',
 'Chi':'Children\'s',
 'Chil':'Children\'s',
 'Childr':'Children\'s',
 'Childre':'Children\'s',
 'Children':'Children\'s',
 "Children'":'Children\'s',
 "Children's":'Children\'s',
 'Com':'Comedy',
 'Come':'Comedy',
 'Comed':'Comedy',
 'Comedy':'Comedy',
 'Crime':'Comedy',
 'D':'Documentary',
 'Docu':'Documentary',
 'Documen':'Documentary',
 'Document':'Documentary',
 'Documenta':'Documentary',
 'Documentary':'Documentary',
 'Dr':'Drama',
 'Dram':'Drama',
 'Drama':'Drama',
 'F':'Fantasy',
 'Fant':'Fantasy',
 'Fantas':'Fantasy',
 'Fantasy':'Fantasy',
 'Film-Noir':'Film-Noir',
 'Horr':'Horror',
 'Horro':'Horror',
 'Horror':'Horror',
 'Music':'Musical',
 'Musical':'Musical',
 'Mystery':'Mystery',
 'R':'Romance',
 'Ro':'Romance',
 'Rom':'Romance',
 'Roma':'Romance',
 'Roman':'Romance',
 'Romance':'Romance',
 'S':'Sci-Fi',
 'Sci':'Sci-Fi',
 'Sci-':'Sci-Fi',
 'Sci-F':'Sci-Fi',
 'Sci-Fi':'Sci-Fi',
 'Th':'Thriller',
 'Thri':'Thriller',
 'Thrille':'Thriller',
 'Thriller':'Thriller',
 'Wa':'War',
 'War':'War',
 'We':'Western',
 'Wester':'Western',
 'Western':'Western'}
```

In [26]:

```
mov['Genres']=mov['Genres'].map(G)
# Mapping genres with correct values
```

In [27]:

```
mov['Genres'].value_counts()
# Count of movies in each Genres
# Highest is for Drama, Comedy and Action movies
```

Out[27]:

```
Drama          1607
Comedy         1407
Action          503
Thriller        488
Romance         462
Adventure       386
Horror          340
Sci-Fi          265
Children's      249
War             139
Documentary     127
Musical         113
Mystery         105
Western          68
Fantasy          63
Film-Noir        44
Name: Genres, dtype: int64
```

In [28]:

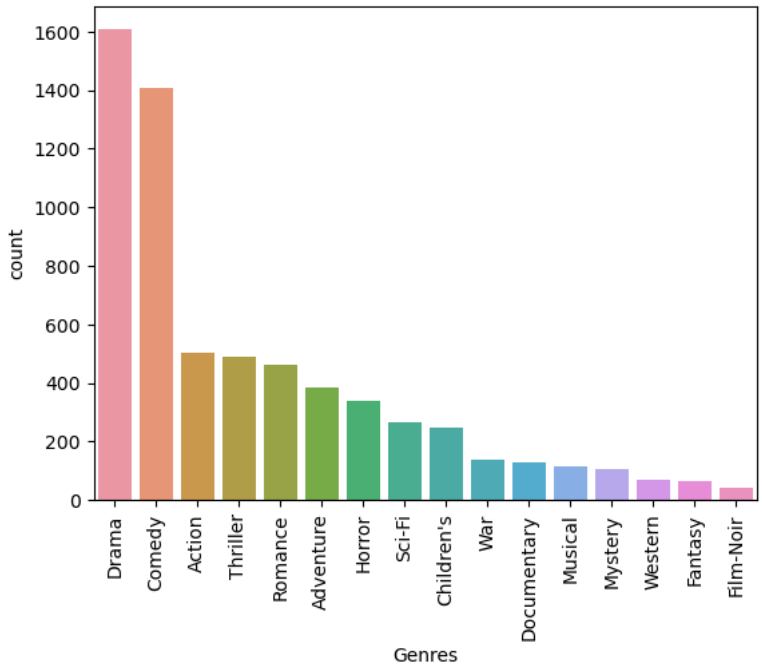
```
mov['Title'].nunique()
# There are total of 3841 unique movies
```

Out[28]:

```
3841
```

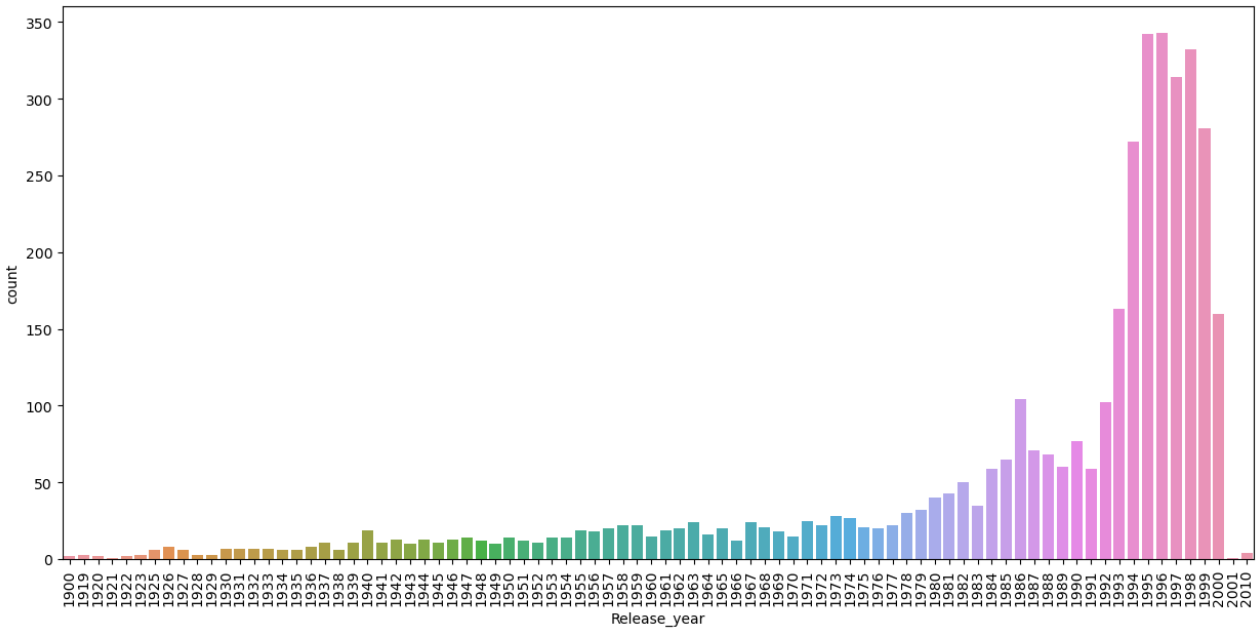
In [29]:

```
sns.countplot(data=mov, x='Genres',order=mov['Genres'].value_counts().index)
plt.xticks(rotation = 90)
plt.show()
# Bar plot below shows the count of movies in each Genres
```



In [30]:

```
plt.figure(figsize=(15,7))
sns.countplot(data=mov[['Title','Release_year']].drop_duplicates(), x='Release_year')
plt.xticks(rotation = 90)
plt.show()
# Plot below shows count of movies released in each year
# From year 1992 to 2000 there were more number of movie releases
```



In [31]:

```
rat['Rating']=rat['Rating'].astype(int)
rat['UserID']=rat['UserID'].astype(int)
```

In [32]:

```
rat
```

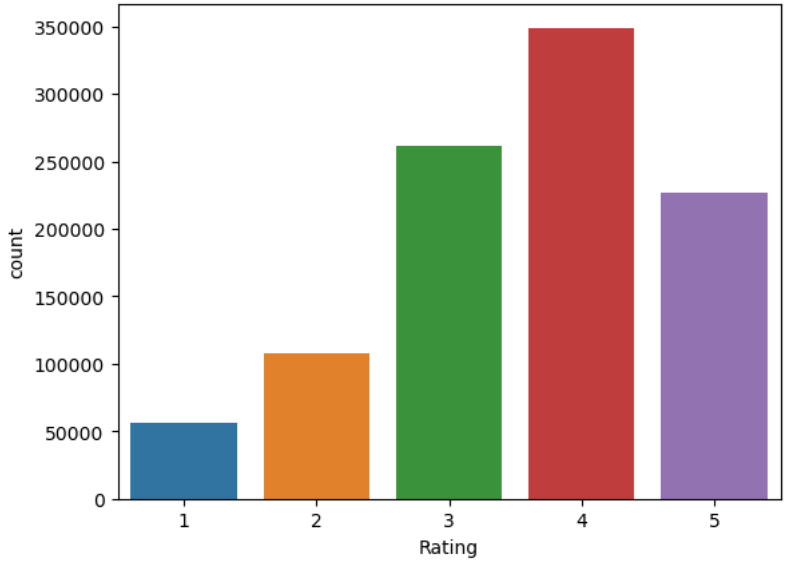
Out[32]:

	UserID	MovieID	Rating	Ratings_Time	Ratings_Date
0	1	1193	5	22:12:40	2000-12-31
1	1	661	3	22:35:09	2000-12-31
2	1	914	3	22:32:48	2000-12-31
3	1	3408	4	22:04:35	2000-12-31
4	1	2355	5	23:38:11	2001-01-06
...	...	...	...	...	...
1000204	6040	1091	1	02:35:41	2000-04-26
1000205	6040	1094	5	23:21:27	2000-04-25
1000206	6040	562	5	23:19:06	2000-04-25
1000207	6040	1096	4	02:20:48	2000-04-26
1000208	6040	1097	4	02:19:29	2000-04-26

1000209 rows × 5 columns

In [33]:

```
sns.countplot(data=rat, x='Rating')
plt.show()
# Bar plot for count of rating
# Rating 3 and 4 has the highest count of rating
```



In [34]:

```
rat.groupby('UserID')['Rating'].mean().reset_index().sort_values(by='Rating',ascending=False)
# Table below shows the average rating of each user
```

Out[34]:

	UserID	Rating
282	283	4.962963
2338	2339	4.956522
3323	3324	4.904762
3901	3902	4.890909
445	446	4.843137
...	...	...
5849	5850	1.844828
4538	4539	1.815126
2743	2744	1.304348
4485	4486	1.058824
3597	3598	1.015385

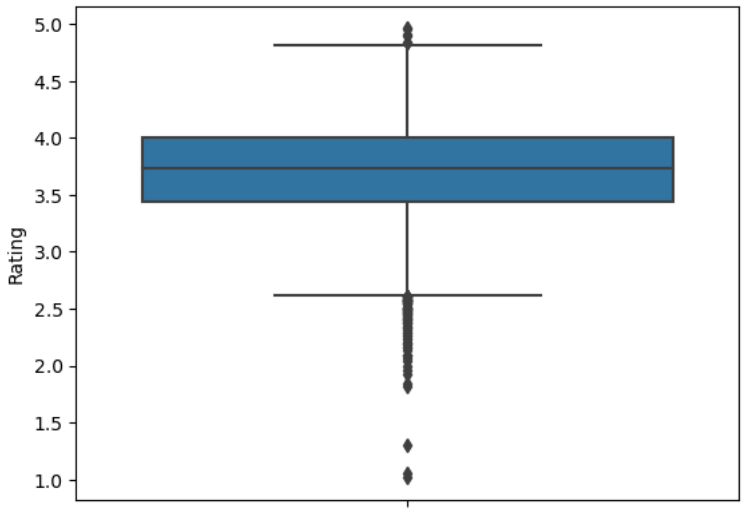
6040 rows × 2 columns

In [35]:

```
sns.boxplot(data=rat.groupby('UserID')['Rating'].mean().reset_index(), y='Rating')
# Boxplot for ratings
```

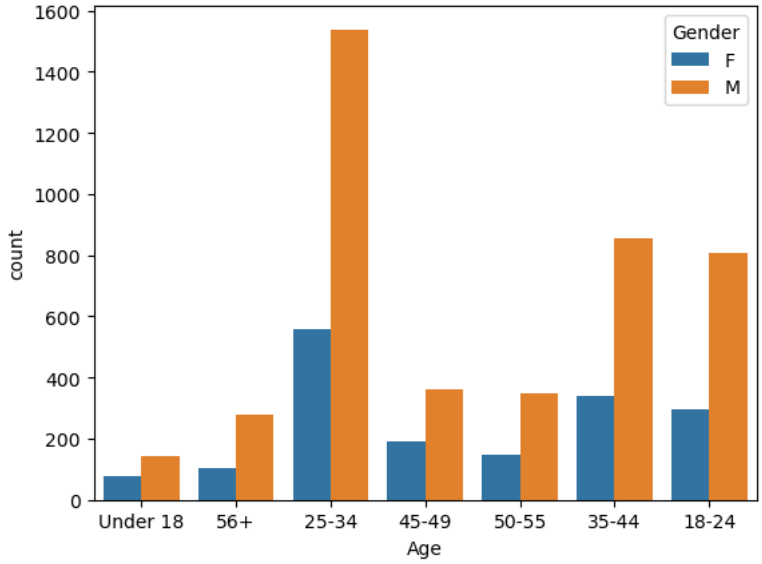
Out[35]:

```
<AxesSubplot:ylabel='Rating'>
```



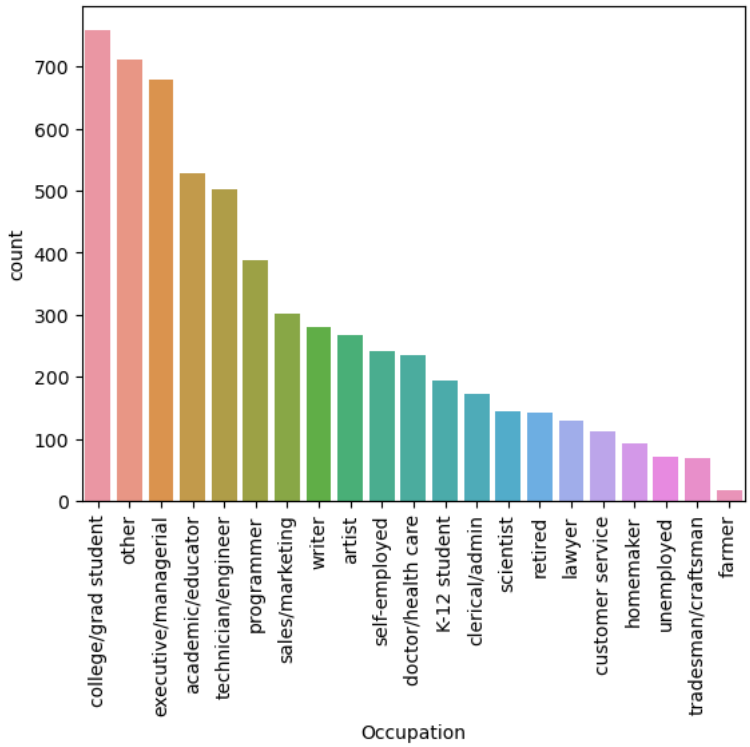
In [36]:

```
sns.countplot(data=use, x='Age',hue='Gender')
plt.show()
# In each are group there are almmost double the number of Male than Female
```



In [37]:

```
sns.countplot(data=use, x='Occupation',order=use['Occupation'].value_counts().index)
plt.xticks(rotation = 90)
plt.show()
# Top 4 occupations users those who watch most movies are
# College/grad student, other, executive and academic
```



In [38]:

```
use['Zip-code'].nunique()
# There are a total of 3439
```

Out[38]:

```
3439
```

Group the data according to the average rating and no. of ratings

In [39]:

```
mer1=rat.merge(use, how='left', left_on='UserID', right_on='UserID')
mer2=mov.merge(mer1, how='left', left_on='Movie ID', right_on='MovieID')
mer2.dropna(inplace=True)
```

In [40]:

```
mer2
```

Out[40]:

	Movie ID	Genres	Title	Release_year	UserID	MovieID	Rating	Ratings_Time	Ratings_Date	Gender	Age	Occupation	Zip-code
0	1	Adventure	Toy Story	1995	1.0	1.0	5.0	23:37:48	2001-01-06	F	Under 18	K-12 student	48067
1	1	Adventure	Toy Story	1995	6.0	1.0	4.0	04:30:08	2000-12-31	F	50-55	homemaker	55117
2	1	Adventure	Toy Story	1995	8.0	1.0	4.0	03:31:36	2000-12-31	M	25-34	programmer	11413
3	1	Adventure	Toy Story	1995	9.0	1.0	5.0	01:25:52	2000-12-31	M	25-34	technician/engineer	61614
4	1	Adventure	Toy Story	1995	10.0	1.0	5.0	01:34:34	2000-12-31	F	35-44	academic/educator	95370
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2064306	3952	Thriller	Contender, The	2000	5812.0	3952.0	4.0	07:34:59	2001-06-09	F	25-34	executive/managerial	92120
2064307	3952	Thriller	Contender, The	2000	5831.0	3952.0	3.0	14:52:05	2001-04-02	M	25-34	academic/educator	92120
2064308	3952	Thriller	Contender, The	2000	5837.0	3952.0	4.0	20:04:16	2002-01-24	M	25-34	executive/managerial	60607
2064309	3952	Thriller	Contender, The	2000	5927.0	3952.0	1.0	21:15:37	2001-01-18	M	35-44	sales/marketing	10003
2064310	3952	Thriller	Contender, The	2000	5998.0	3952.0	4.0	16:30:44	2001-09-29	M	18-24	college/grad student	61820

2064096 rows × 13 columns

In [41]:

```
A=mer2[['Title','Genres']].drop_duplicates(keep='first').reset_index()
```

In [42]:

```
A
```

Out[42]:

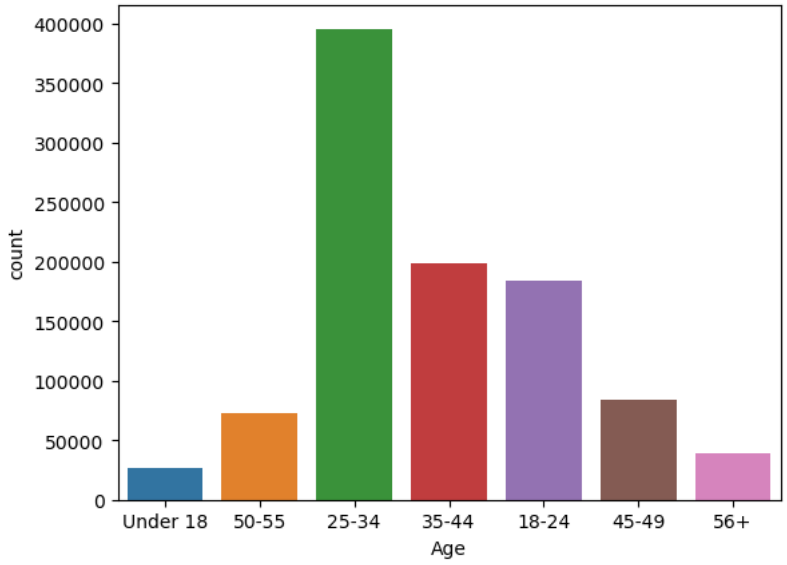
	index	Title	Genres
0	0	Toy Story	Adventure
1	2077	Toy Story	Children's
2	4154	Toy Story	Comedy
3	6231	Jumanji	Adventure
4	6932	Jumanji	Children's
...	...	...	...
6045	2063137	Requiem for a Dream	Drama
6046	2063441	Tigerland	Drama
6047	2063495	Two Family House	Drama
6048	2063535	Contender, The	Drama
6049	2063923	Contender, The	Thriller

6050 rows × 3 columns

Q. 1.Users of which age group have watched and rated the most number of movies?

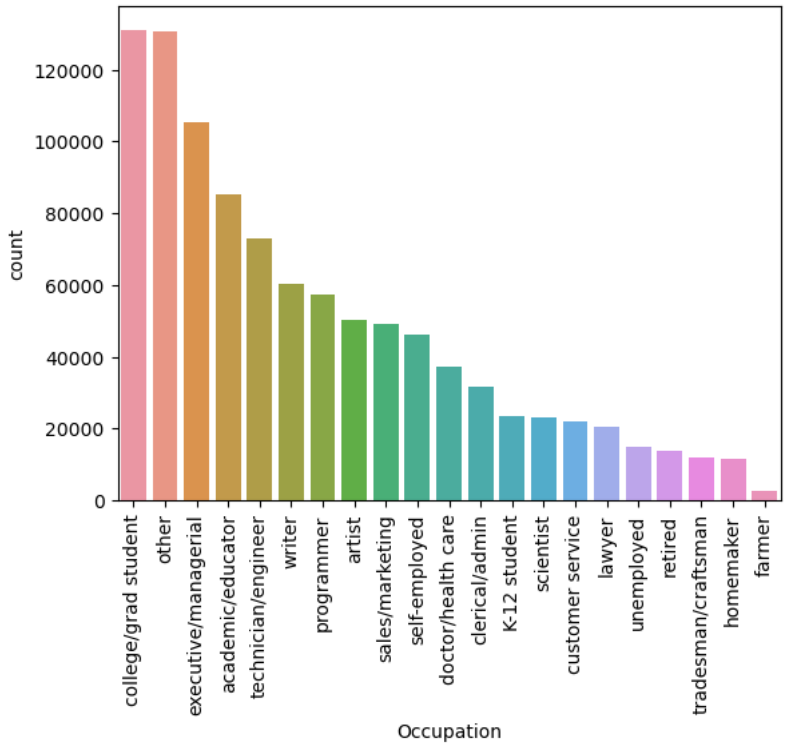
In [43]:

```
sns.countplot(data=mer2[['UserID','Age','MovieID']].drop_duplicates(), x='Age')
plt.show()
# Age group 25-34 have highest count who watch movies
```



In [44]:

```
sns.countplot(data=mer2[['UserID','Occupation','MovieID']].drop_duplicates(), x='Occupation',
              order=mer2[['UserID','Occupation','MovieID']].drop_duplicates()['Occupation'].value_counts().index)
plt.xticks(rotation=90)
plt.show()
# Top 4 occupations users those who watch most movies are
# College/grad student, other, executive and academic
```



In [45]:

```
mer2[['UserID','Occupation','MovieID']].drop_duplicates()['Occupation'].value_counts()
# Count of number of unique occupations
```

Out[45]:

```
college/grad student    131032
other                   130499
executive/managerial    105425
academic/educator        85351
technician/engineer      72816
writer                   60397
programmer               57214
artist                   50068
sales/marketing          49109
self-employed            46021
doctor/health care       37205
clerical/admin           31623
K-12 student             23290
scientist                22951
customer service         21850
lawyer                   20563
unemployed               14904
retired                  13754
tradesman/craftsman      12086
homemaker                11345
farmer                    2706
Name: Occupation, dtype: int64
```

In [46]:

```
mer2[['UserID','Gender','MovieID']].drop_duplicates()['Gender'].value_counts()
# Count of Male and Female
```

Out[46]:

```
M    753769
F    246440
Name: Gender, dtype: int64
```

In [47]:

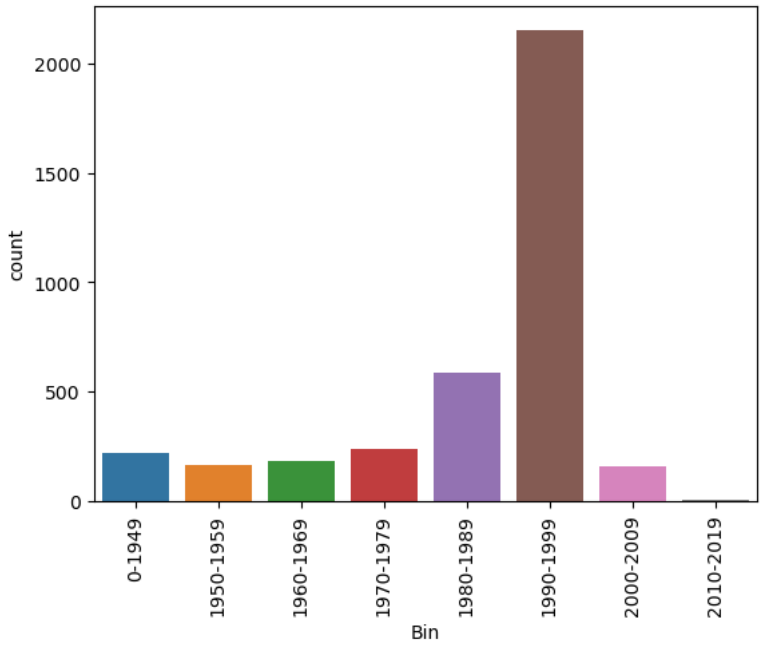
```
B=mer2[['Release_year','MovieID']].drop_duplicates()
```

In [48]:

```
cut_labels_4 = ['0-1949', '1950-1959','1960-1969','1970-1979','1980-1989','1990-1999','2000-2009','2010-2019']
cut_bins = [0, 1949, 1959, 1969, 1979,1989,1999,2009,2019]
B['Bin'] = pd.cut(B['Release_year'], bins=cut_bins, labels=cut_labels_4)
```

In [49]:

```
sns.countplot(data=B, x='Bin')
plt.xticks(rotation=90)
plt.show()
# Most movies was released in 90's followed by 80's
```



In [50]:

```
mer2[['Movie ID','UserID','Rating']].groupby('Movie ID')['Rating'].count().sort_values(ascending=False)
# Count of the rating received for each movie
```

Out[50]:

```
Movie ID
1580    10152
1197     9272
1617     9152
1097     9076
260      8973
        ...
576         1
872         1
3237        1
2619        1
730         1
Name: Rating, Length: 3706, dtype: int64
```

Grouping the data in terms of Average Rating and No. of Ratings given

In [51]:

```
gro=mer2[['UserID','MovieID','Rating']].drop_duplicates()
```

In [52]:

```
gro.groupby('UserID')['Rating'].mean().reset_index().sort_values(by='Rating',ascending=False).head(20)
# Each user's average rating shown below in descending order
```

Out[52]:

	UserID	Rating
282	283.0	4.962963
2338	2339.0	4.956522
3323	3324.0	4.904762
3901	3902.0	4.890909
445	446.0	4.843137
446	447.0	4.837838
4648	4649.0	4.818182
4633	4634.0	4.813725
1130	1131.0	4.796117
4924	4925.0	4.761905
4754	4755.0	4.760000
5068	5069.0	4.760000
4800	4801.0	4.734694
681	682.0	4.733333
1669	1670.0	4.714286
90	91.0	4.704545
5767	5768.0	4.702703
3460	3461.0	4.702703
5983	5984.0	4.700000
1020	1021.0	4.694656

In [53]:

```
gro.groupby('UserID')['Rating'].count().reset_index().sort_values(by='Rating',ascending=False).head(20)
# Number of rating each user has provided is shown below
```

Out[53]:

	UserID	Rating
4168	4169.0	2314
1679	1680.0	1850
4276	4277.0	1743
1940	1941.0	1595
1180	1181.0	1521
888	889.0	1518
3617	3618.0	1344
2062	2063.0	1323
1149	1150.0	1302
1014	1015.0	1286
5794	5795.0	1277
4343	4344.0	1271
1979	1980.0	1260
2908	2909.0	1258
1448	1449.0	1243
4509	4510.0	1240
423	424.0	1226
4226	4227.0	1222
5830	5831.0	1220
3840	3841.0	1216

In [54]:

```
fro=mer2[['UserID','Title','Rating']].drop_duplicates().drop('UserID',axis=1)
```

In [55]:

```
mov_avg=fro.groupby('Title')['Rating'].agg(['mean','count']).sort_values(by='mean',ascending=False)
```

In [56]:

```
mov_avg[mov_avg['count']>200]
# Average rating for each movie where the count of each movie is greater than 200
```

Out[56]:

	mean	count
Title		
Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (195	4.560510	628
Shawshank Redemption, The	4.554558	2227
Godfather, The	4.524966	2223
Close Shave, A	4.520548	657
Usual Suspects, The	4.517106	1783
...	...	...
Superman IV: The Quest for Peace	1.888554	332
Super Mario Bros.	1.874286	350
Speed 2: Cruise Control	1.871935	367
Jaws 3-D	1.852381	210
Battlefield Earth	1.611111	342

1410 rows × 2 columns

Build a Recommender System based on Pearson Correlation

1. Recommend 5 similar movies based on Pearson Correlation (genres)

In [58]:

```
piv=pd.pivot_table(A,columns='Genres',index='Title')
piv[piv >= 1] = 1
matrix=piv.transpose().reset_index().drop(['level_0','Genres'],axis=1)
matrix=matrix.fillna(0)
matrix1=matrix.corr(method ='pearson')
movie_name=input('Please enter a valid movie name ')
```

```
Please enter a valid movie name Liar Liar
```

In [59]:

```
matrix1[movie_name].sort_values(ascending=False).reset_index().iloc[:6][1:]
# Top 5 movie recommendation for 'Liar Liar' movie
```

Out[59]:

	Title	Liar Liar
1	Cecil B. Demented	1.0
2	Mr. Magoo	1.0
3	Mr. & Mrs. Smith	1.0
4	Mouth to Mouth (Boca a boca)	1.0
5	Mother	1.0

2. Recommend 5 similar movies based on Pearson Correlation (rating)

In [60]:

```
per_cor=mer2[['Title','Rating','UserID']].drop_duplicates(keep='first')
per_cor_df=pd.pivot_table(per_cor,values='Rating',columns='Title',index='UserID')
per_cor_df.fillna(0,inplace=True)
person_cor=per_cor_df.corr(method ='pearson')
```

In [61]:

```
person_cor
# Pearson Correlation Matrix
```

Out[61]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13th Warrior, The	...	Young Poisoner's Handbook, The	Young Sherlock Holmes	Yo Innc
Title														
$1,000,000 Duck	1.000000	0.065338	0.030805	0.065478	0.048708	-0.001319	0.036612	0.175724	0.075665	0.036112	...	0.030758	0.060574	-0.00
'Night Mother	0.065338	1.000000	0.107374	0.096778	0.144480	-0.001834	0.046122	0.109631	0.083988	0.013045	...	0.042061	0.065333	0.06
'Til There Was You	0.030805	0.107374	1.000000	0.082706	0.051965	0.079011	0.105672	0.105814	0.054821	0.040220	...	0.019685	0.043294	-0.00
'burbs, The	0.065478	0.096778	0.082706	1.000000	0.110790	-0.003821	0.133881	0.195062	0.113112	0.139186	...	0.092597	0.165666	0.01
...And Justice for All	0.048708	0.144480	0.051965	0.110790	1.000000	-0.003203	0.018614	0.128837	0.160556	0.068781	...	0.071819	0.115402	0.06
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
Zed & Two Noughts, A	0.040590	0.085001	0.016935	0.042862	0.075720	-0.001186	-0.009530	0.021234	0.019564	0.020547	...	0.040208	0.059839	0.06
Zero Effect	0.024165	0.054343	0.062262	0.121617	0.075804	-0.003931	0.114230	0.092254	0.070705	0.092476	...	0.159991	0.124075	0.01
Zero Kelvin (Kjærlighetens kjøtere)	-0.001332	-0.001852	-0.001571	-0.003858	0.072283	-0.000322	-0.006235	0.028734	0.032880	0.030558	...	0.046727	0.043840	-0.00
Zeus and Roxanne	0.116574	-0.005825	0.042842	0.022249	-0.010171	-0.001011	0.042612	0.075291	0.043186	0.016848	...	-0.006375	0.034016	-0.00
eXistenZ	0.009243	0.054699	0.043483	0.062471	0.071000	0.036301	0.089419	0.041839	0.023730	0.163531	...	0.091692	0.128434	0.01

3664 rows × 3664 columns

In [62]:

```
movie_name=input('Please enter a valid movie name ')
```

```
Please enter a valid movie name Liar Liar
```

In [63]:

```
person_cor[movie_name].sort_values(ascending=False).reset_index().iloc[:6][1:]
```

Out[63]:

	Title	Liar Liar
1	Mrs. Doubtfire	0.499927
2	Dumb & Dumber	0.459601
3	Ace Ventura: Pet Detective	0.458654
4	Home Alone	0.455967
5	Wedding Singer, The	0.429222

3. Cosine Similarity based on movie genres

In [64]:

```
matrix2=matrix.transpose()
```

In [65]:

```
matrix2
# Dataframe below shown has movie and its genres marked 1
```

Out[65]:

Title	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$1,000,000 Duck	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
'Night Mother	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
'Til There Was You	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
'burbs, The	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...And Justice for All	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Zed & Two Noughts, A	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Zero Effect	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
Zero Kelvin (Kjærlighetens kjøtere)	1.0	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
Zeus and Roxanne	0.0	0.0	1.0	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
eXistenZ	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0

3664 rows × 16 columns

CSR matrix for movie genres

In [66]:

```
csr_mat=csr_matrix(matrix2)
```

In [67]:

```
print(csr_mat)
# Sparse matrix for genres
```

```
  (0, 2)        1.0
  (0, 3)        1.0
  (1, 5)        1.0
  (2, 5)        1.0
  (2, 11)       1.0
  (3, 3)        1.0
  (4, 5)        1.0
  (4, 13)       1.0
  (5, 11)       1.0
  (6, 3)        1.0
  (6, 11)       1.0
  (7, 1)        1.0
  (7, 2)        1.0
  (7, 3)        1.0
  (8, 5)        1.0
  (9, 0)        1.0
  (9, 8)        1.0
  (9, 13)       1.0
  (10, 5)       1.0
  (11, 3)       1.0
  (12, 3)       1.0
  (13, 1)       1.0
  (13, 2)       1.0
  (13, 6)       1.0
  (14, 3)       1.0
  :     :
  (3650, 3)     1.0
  (3651, 3)     1.0
  (3651, 8)     1.0
  (3652, 0)     1.0
  (3652, 3)     1.0
  (3652, 15)    1.0
  (3653, 0)     1.0
  (3653, 3)     1.0
  (3653, 15)    1.0
  (3654, 3)     1.0
  (3655, 0)     1.0
  (3655, 1)     1.0
  (3655, 10)    1.0
  (3656, 3)     1.0
  (3656, 13)    1.0
  (3657, 5)     1.0
  (3658, 15)    1.0
  (3659, 5)     1.0
  (3660, 3)     1.0
  (3660, 13)    1.0
  (3661, 0)     1.0
  (3662, 2)     1.0
  (3663, 0)     1.0
  (3663, 12)    1.0
  (3663, 13)    1.0
```

In [68]:

```
new_cos=cosine_similarity(matrix2,matrix2)
new_cos=pd.DataFrame(new_cos)
new_cos.index=matrix2.index
new_cos.columns=matrix2.index
```

In [69]:

```
new_cos
# Cosine Similarity for each movies based on genres
```

Out[69]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13th Warrior, The	...	Young Poisoner's Handbook, The	Young Sherlock Holmes	Young and Innocent
Title														
$1,000,000 Duck	1.000000	0.000000	0.000000	0.707107	0.000000	0.000000	0.500000	0.816497	0.000000	0.000000	...	0.707107	0.000000	0.500000
'Night Mother	0.000000	1.000000	0.707107	0.000000	0.707107	0.000000	0.000000	0.000000	1.000000	0.000000	...	0.000000	0.000000	0.000000
'Til There Was You	0.000000	0.707107	1.000000	0.000000	0.500000	0.707107	0.500000	0.000000	0.707107	0.000000	...	0.000000	0.000000	0.000000
'burbs, The	0.707107	0.000000	0.000000	1.000000	0.000000	0.000000	0.707107	0.577350	0.000000	0.000000	...	1.000000	0.000000	0.707107
...And Justice for All	0.000000	0.707107	0.500000	0.000000	1.000000	0.000000	0.000000	0.000000	0.707107	0.408248	...	0.000000	0.000000	0.500000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Zed & Two Noughts, A	0.000000	1.000000	0.707107	0.000000	0.707107	0.000000	0.000000	0.000000	1.000000	0.000000	...	0.000000	0.000000	0.000000
Zero Effect	0.500000	0.000000	0.000000	0.707107	0.500000	0.000000	0.500000	0.408248	0.000000	0.408248	...	0.707107	0.000000	1.000000
Zero Kelvin (Kjærlighetens kjøtere)	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.577350	...	0.000000	0.577350	0.000000
Zeus and Roxanne	0.707107	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.577350	0.000000	0.000000	...	0.000000	0.000000	0.000000
eXistenZ	0.000000	0.000000	0.000000	0.000000	0.408248	0.000000	0.000000	0.000000	0.000000	0.666667	...	0.000000	0.333333	0.408248

3664 rows × 3664 columns

In [70]:

```
movie_name=input('Please enter a valid movie name ')
```

```
Please enter a valid movie name Liar Liar
```

In [71]:

```
new_cos.loc[movie_name].sort_values(ascending=False).reset_index()[:6][1:]
```

Out[71]:

	Title	Liar Liar
1	Man of the House	1.0
2	Me, Myself and Irene	1.0
3	Me Myself I	1.0
4	Maybe, Maybe Not (Bewegte Mann, Der)	1.0
5	Max Dugan Returns	1.0

4. Cosine Similarity based on user (Gender,Age, Occupation)

In [72]:

```
use1=use.set_index('UserID')
use1.drop('Zip-code',axis=1,inplace=True)
```

In [73]:

```
use1
# Each user has gender, age and Occupation
```

Out[73]:

UserID	Gender	Age	Occupation
1	F	Under 18	K-12 student
2	M	56+	self-employed
3	M	25-34	scientist
4	M	45-49	executive/managerial
5	M	25-34	writer
...	...	...	...
6036	F	25-34	scientist
6037	F	45-49	academic/educator
6038	F	56+	academic/educator
6039	F	45-49	other
6040	M	25-34	doctor/health care

6040 rows × 3 columns

In [74]:

```
use2=OneHotEncoder( drop='first' ).fit_transform(use1)
# One hot encoding all the columns
```

In [75]:

```
use2=use2.todense()
# Converting matrix to dense
```

CSR matrix for user (Gender,Age, Occupation)

In [76]:

```
csr_mat=csr_matrix(use2)
# CSR matrix
```

In [77]:

```
print(csr_mat)
# Sparse matrix of users
```

```
  (0, 6)        1.0
  (1, 0)        1.0
  (1, 5)        1.0
  (1, 22)       1.0
  (2, 0)        1.0
  (2, 1)        1.0
  (2, 21)       1.0
  (3, 0)        1.0
  (3, 3)        1.0
  (3, 13)       1.0
  (4, 0)        1.0
  (4, 1)        1.0
  (4, 26)       1.0
  (5, 4)        1.0
  (5, 15)       1.0
  (6, 0)        1.0
  (6, 2)        1.0
  (6, 7)        1.0
  (7, 0)        1.0
  (7, 1)        1.0
  (7, 18)       1.0
  (8, 0)        1.0
  (8, 1)        1.0
  (8, 23)       1.0
  (9, 2)        1.0
  :     :
  (6029, 1)     1.0
  (6029, 23)    1.0
  (6030, 17)    1.0
  (6031, 0)     1.0
  (6031, 3)     1.0
  (6031, 13)    1.0
  (6032, 0)     1.0
  (6032, 4)     1.0
  (6032, 19)    1.0
  (6033, 0)     1.0
  (6033, 1)     1.0
  (6033, 20)    1.0
  (6034, 1)     1.0
  (6034, 7)     1.0
  (6035, 1)     1.0
  (6035, 21)    1.0
  (6036, 3)     1.0
  (6036, 7)     1.0
  (6037, 5)     1.0
  (6037, 7)     1.0
  (6038, 3)     1.0
  (6038, 17)    1.0
  (6039, 0)     1.0
  (6039, 1)     1.0
  (6039, 12)    1.0
```

In [78]:

```
use2=pd.DataFrame(use2,index=use1.index)
```

In [79]:

```
use2
```

Out[79]:

UserID	0	1	2	3	4	5	6	7	8	9	...	17	18	19	20	21	22	23	24	25	26
1	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
4	1.0	0.0	0.0	1.0	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	1.0	1.0	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	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6036	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
6037	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6038	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6039	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6040	1.0	1.0	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

6040 rows × 27 columns

In [80]:

```
new_cos=cosine_similarity(use2,use2)
new_cos=pd.DataFrame(new_cos)
new_cos.index=use2.index
new_cos.columns=use2.index
```

In [81]:

```
new_cos
# Similarity matrix
```

Out[81]:

UserID	1	2	3	4	5	6	7	8	9	10	...	6031	6032	6033	6034	6035	603
UserID																	
1	1.0	0.000000	0.000000	0.000000	0.000000	0.0	0.000000	0.000000	0.000000	0.0	...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
2	0.0	1.000000	0.333333	0.333333	0.333333	0.0	0.333333	0.333333	0.333333	0.0	...	0.000000	0.333333	0.333333	0.333333	0.000000	0.000000
3	0.0	0.333333	1.000000	0.333333	0.666667	0.0	0.333333	0.666667	0.666667	0.0	...	0.000000	0.333333	0.333333	0.666667	0.408248	0.816497
4	0.0	0.333333	0.333333	1.000000	0.333333	0.0	0.333333	0.333333	0.333333	0.0	...	0.000000	1.000000	0.333333	0.333333	0.000000	0.000000
5	0.0	0.333333	0.666667	0.333333	1.000000	0.0	0.333333	0.666667	0.666667	0.0	...	0.000000	0.333333	0.333333	0.666667	0.408248	0.408248
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
6036	0.0	0.000000	0.816497	0.000000	0.408248	0.0	0.000000	0.408248	0.408248	0.0	...	0.000000	0.000000	0.000000	0.408248	0.500000	1.000000
6037	0.0	0.000000	0.000000	0.408248	0.000000	0.0	0.408248	0.000000	0.000000	0.5	...	0.000000	0.408248	0.000000	0.000000	0.500000	0.000000
6038	0.0	0.408248	0.000000	0.000000	0.000000	0.0	0.408248	0.000000	0.000000	0.5	...	0.000000	0.000000	0.000000	0.000000	0.500000	0.000000
6039	0.0	0.000000	0.000000	0.408248	0.000000	0.0	0.000000	0.000000	0.000000	0.0	...	0.707107	0.408248	0.000000	0.000000	0.000000	0.000000
6040	0.0	0.333333	0.666667	0.333333	0.666667	0.0	0.333333	0.666667	0.666667	0.0	...	0.000000	0.333333	0.333333	0.666667	0.408248	0.408248

6040 rows × 6040 columns

In [82]:

```
user_id=int(input('Please enter a valid user id '))
```

```
Please enter a valid user id 325
```

In [83]:

```
B=new_cos.loc[user_id].sort_values(ascending=False).reset_index()[:6][1:]
```

In [84]:

```
B
# Top 5 users similar to uder id entered
```

Out[84]:

	UserID	325
1	3039	1.0
2	5340	1.0
3	325	1.0
4	3171	1.0
5	2482	1.0

5. Cosine Similarity based on movie rating for movie

In [85]:

```
movie=per_cor_df.reset_index().drop('UserID',axis=1)
movie.fillna(0,inplace=True)
movie=movie.transpose()
movie_arr=movie.to_numpy()
movie_array=1-pairwise_distances(movie_arr, metric="cosine")
new_cos=pd.DataFrame(movie_array)
new_cos.index=movie.index
new_cos.columns=movie.index
```

In [86]:

```
movie_name=input('Please enter a valid movie name ')
```

```
Please enter a valid movie name Liar Liar
```

In [87]:

```
new_cos.loc[movie_name].sort_values(ascending=False).reset_index()[:6][1:]
# Top 5 movies based on movie name entered based on Cosine similarity
```

Out[87]:

	Title	Liar Liar
1	Mrs. Doubtfire	0.557067
2	Ace Ventura: Pet Detective	0.516861
3	Dumb & Dumber	0.512585
4	Home Alone	0.511204
5	Wayne's World	0.499368

In [88]:

```
new_cos
# Cosine similarity for each movie
```

Out[88]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13th Warrior, The	...	Young Poisoner's Handbook, The	Young Sherlock Holmes	Young and Innocent
Title														
$1,000,000 Duck	1.000000	0.072357	0.037011	0.079291	0.060838	0.000000	0.058619	0.189843	0.094785	0.058418	...	0.038725	0.076474	0.000000
'Night Mother	0.072357	1.000000	0.115290	0.115545	0.159526	0.000000	0.076798	0.137135	0.111413	0.046135	...	0.053010	0.087828	0.063758
'Til There Was You	0.037011	0.115290	1.000000	0.098756	0.066301	0.080250	0.127895	0.128523	0.079115	0.066598	...	0.029200	0.062893	0.000000
'burbs, The	0.079291	0.115545	0.098756	1.000000	0.143620	0.000000	0.192191	0.250140	0.170719	0.197808	...	0.113386	0.207897	0.019962
...And Justice for All	0.060838	0.159526	0.066301	0.143620	1.000000	0.000000	0.075093	0.178928	0.205486	0.122431	...	0.089998	0.153006	0.067009
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Zed & Two Noughts, A	0.045280	0.091150	0.022594	0.055704	0.086080	0.000000	0.012702	0.042295	0.039344	0.041324	...	0.047282	0.073996	0.070409
Zero Effect	0.039395	0.074787	0.079261	0.161174	0.110867	0.000000	0.175771	0.157313	0.133061	0.156505	...	0.179315	0.169677	0.021362
Zero Kelvin (Kjærlighetens kjøtere)	0.000000	0.000000	0.000000	0.000000	0.074317	0.000000	0.000000	0.033120	0.036867	0.034797	...	0.048440	0.046892	0.000000
Zeus and Roxanne	0.120242	0.000000	0.047526	0.033567	0.000000	0.000000	0.058708	0.089840	0.058692	0.034623	...	0.000000	0.046658	0.000000
eXistenZ	0.027003	0.077807	0.063284	0.110525	0.111040	0.039561	0.162060	0.120762	0.098731	0.230799	...	0.115734	0.180174	0.024437

3664 rows × 3664 columns

6.Cosine Similarity based on movie rating for user id

In [89]:

```
use_mat=per_cor_df.transpose()
use_mat=use_mat.reset_index().drop('Title',axis=1)
use_mat.fillna(0,inplace=True)
use_mat=use_mat.transpose()
use_arr=use_mat.to_numpy()
use_array=1-pairwise_distances(use_arr, metric="cosine")
new_cos_u=pd.DataFrame(use_array)
new_cos_u.index=use_mat.index
new_cos_u.columns=use_mat.index
```

In [90]:

```
user_id=int(input('Please enter a valid user name '))
```

```
Please enter a valid user name 325
```

In [91]:

```
new_cos_u.loc[user_id].sort_values(ascending=False).reset_index()[:6][1:]
# Top 5 users similar to user id entered
```

Out[91]:

	UserID	325.0
1	3766.0	0.356665
2	4336.0	0.293017
3	4181.0	0.283445
4	2418.0	0.281295
5	2390.0	0.273641

In [92]:

```
new_cos_u
# Cosine similarity for each user id
```

Out[92]:

UserID	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	...	6031.0	6032.0	6033.0	6034.0	
UserID																
1.0	1.000000	0.096382	0.120610	0.132455	0.090158	0.179222	0.059678	0.138241	0.226148	0.254736	...	0.170588	0.082006	0.090961	0.033663	0
2.0	0.096382	1.000000	0.151479	0.171176	0.114394	0.100865	0.305787	0.211120	0.190198	0.227745	...	0.112503	0.091222	0.268565	0.014286	0.
3.0	0.120610	0.151479	1.000000	0.151227	0.062907	0.074603	0.138332	0.077656	0.126457	0.214487	...	0.092960	0.125864	0.161507	0.000000	0.
4.0	0.132455	0.171176	0.151227	1.000000	0.045094	0.013529	0.130339	0.100856	0.093651	0.121208	...	0.163629	0.093041	0.382803	0.000000	0.
5.0	0.090158	0.114394	0.062907	0.045094	1.000000	0.047449	0.126257	0.220817	0.261330	0.117508	...	0.100652	0.035732	0.065278	0.054151	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6036.0	0.186957	0.229011	0.143747	0.171158	0.294354	0.093899	0.122854	0.232304	0.240415	0.344021	...	0.131737	0.210550	0.193417	0.103780	0.
6037.0	0.136233	0.206660	0.107945	0.127703	0.173009	0.065911	0.111882	0.152506	0.226883	0.251853	...	0.142575	0.276651	0.130228	0.118971	0.
6038.0	0.000000	0.066118	0.120234	0.062907	0.020459	0.065711	0.000000	0.019242	0.093470	0.114232	...	0.108837	0.106897	0.040689	0.000000	0.
6039.0	0.174604	0.066457	0.094675	0.064634	0.027689	0.167303	0.014977	0.044660	0.046434	0.297386	...	0.118776	0.250994	0.053750	0.102168	0.
6040.0	0.133590	0.218276	0.133144	0.137968	0.241437	0.083436	0.080680	0.148123	0.215819	0.256789	...	0.154574	0.291988	0.115540	0.118840	0.

6040 rows × 6040 columns

CSR matrix for Movie

In [93]:

```
movie=per_cor_df.reset_index().drop('UserID',axis=1)
movie.fillna(0,inplace=True)
movie=movie.transpose()
movie_arr=movie.to_numpy()
csr_mat=csr_matrix(movie_arr)
```

In [94]:

```
print(csr_mat)
# Sparse matrix for movie
```

```
  (0, 215)      2.0
  (0, 493)      5.0
  (0, 713)      4.0
  (0, 868)      1.0
  (0, 1033)     3.0
  (0, 1110)     5.0
  (0, 1140)     4.0
  (0, 1555)     3.0
  (0, 1634)     2.0
  (0, 1644)     5.0
  (0, 1679)     3.0
  (0, 1708)     5.0
  (0, 1747)     1.0
  (0, 1940)     4.0
  (0, 1979)     3.0
  (0, 1997)     3.0
  (0, 2029)     2.0
  (0, 2155)     3.0
  (0, 2254)     3.0
  (0, 2461)     2.0
  (0, 3270)     3.0
  (0, 3291)     2.0
  (0, 3299)     4.0
  (0, 3390)     2.0
  (0, 3664)     4.0
  :     :
  (3663, 5663)  4.0
  (3663, 5674)  3.0
  (3663, 5683)  5.0
  (3663, 5684)  3.0
  (3663, 5701)  4.0
  (3663, 5722)  3.0
  (3663, 5729)  1.0
  (3663, 5748)  4.0
  (3663, 5754)  2.0
  (3663, 5766)  2.0
  (3663, 5787)  4.0
  (3663, 5791)  3.0
  (3663, 5794)  1.0
  (3663, 5796)  3.0
  (3663, 5824)  3.0
  (3663, 5868)  4.0
  (3663, 5885)  4.0
  (3663, 5915)  2.0
  (3663, 5939)  4.0
  (3663, 5952)  4.0
  (3663, 5960)  4.0
  (3663, 5963)  5.0
  (3663, 6000)  4.0
  (3663, 6015)  4.0
  (3663, 6035)  2.0
```

In [95]:

```
similarities = cosine_similarity(csr_mat)
print('pairwise dense output:\n {}\n'.format(similarities))
```

```
pairwise dense output:
 [[1.         0.07235746 0.03701053 ... 0.         0.12024178 0.02700277]
 [0.07235746 1.         0.11528952 ... 0.         0.         0.07780705]
 [0.03701053 0.11528952 1.         ... 0.         0.04752635 0.0632837 ]
 ...
 [0.         0.         0.         ... 1.         0.         0.04564448]
 [0.12024178 0.         0.04752635 ... 0.         1.         0.04433508]
 [0.02700277 0.07780705 0.0632837  ... 0.04564448 0.04433508 1.        ]]
```

In [96]:

```
new_cos=pd.DataFrame(similarities)
new_cos.index=movie.index
new_cos.columns=movie.index
```

In [97]:

```
movie_name=input('Please enter a valid movie name ')
```

```
Please enter a valid movie name Liar Liar
```

In [98]:

```
new_cos.loc[movie_name].sort_values(ascending=False).reset_index()[:6][1:]
# Top 5 movies similar to 'Liar Liar'
```

Out[98]:

	Title	Liar Liar
1	Mrs. Doubtfire	0.557067
2	Ace Ventura: Pet Detective	0.516861
3	Dumb & Dumber	0.512585
4	Home Alone	0.511204
5	Wayne's World	0.499368

In [99]:

```
new_cos
# Cosine Similarity for movies
```

Out[99]:

Title	$1,000,000 Duck	'Night Mother	'Til There Was You	'burbs, The	...And Justice for All	1-900	10 Things I Hate About You	101 Dalmatians	12 Angry Men	13th Warrior, The	...	Young Poisoner's Handbook, The	Young Sherlock Holmes	Young and Innocent
Title														
$1,000,000 Duck	1.000000	0.072357	0.037011	0.079291	0.060838	0.000000	0.058619	0.189843	0.094785	0.058418	...	0.038725	0.076474	0.000000
'Night Mother	0.072357	1.000000	0.115290	0.115545	0.159526	0.000000	0.076798	0.137135	0.111413	0.046135	...	0.053010	0.087828	0.063758
'Til There Was You	0.037011	0.115290	1.000000	0.098756	0.066301	0.080250	0.127895	0.128523	0.079115	0.066598	...	0.029200	0.062893	0.000000
'burbs, The	0.079291	0.115545	0.098756	1.000000	0.143620	0.000000	0.192191	0.250140	0.170719	0.197808	...	0.113386	0.207897	0.019962
...And Justice for All	0.060838	0.159526	0.066301	0.143620	1.000000	0.000000	0.075093	0.178928	0.205486	0.122431	...	0.089998	0.153006	0.067009
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Zed & Two Noughts, A	0.045280	0.091150	0.022594	0.055704	0.086080	0.000000	0.012702	0.042295	0.039344	0.041324	...	0.047282	0.073996	0.070409
Zero Effect	0.039395	0.074787	0.079261	0.161174	0.110867	0.000000	0.175771	0.157313	0.133061	0.156505	...	0.179315	0.169677	0.021362
Zero Kelvin (Kjærlighetens kjøtere)	0.000000	0.000000	0.000000	0.000000	0.074317	0.000000	0.000000	0.033120	0.036867	0.034797	...	0.048440	0.046892	0.000000
Zeus and Roxanne	0.120242	0.000000	0.047526	0.033567	0.000000	0.000000	0.058708	0.089840	0.058692	0.034623	...	0.000000	0.046658	0.000000
eXistenZ	0.027003	0.077807	0.063284	0.110525	0.111040	0.039561	0.162060	0.120762	0.098731	0.230799	...	0.115734	0.180174	0.024437

3664 rows × 3664 columns

CSR matrix for User ID

In [100]:

```
use_mat=per_cor_df.transpose()
use_mat=use_mat.reset_index().drop('Title',axis=1)
use_mat.fillna(0,inplace=True)
use_mat=use_mat.transpose()
use_arr=use_mat.to_numpy()
csr_mat=csr_matrix(use_arr)
```

In [101]:

```
print(csr_mat)
```

```
  (0, 83)       4.0
  (0, 88)       4.0
  (0, 187)      4.0
  (0, 194)      5.0
  (0, 238)      5.0
  (0, 257)      5.0
  (0, 275)      4.0
  (0, 322)      5.0
  (0, 344)      5.0
  (0, 374)      4.0
  (0, 555)      5.0
  (0, 668)      5.0
  (0, 676)      5.0
  (0, 710)      3.0
  (0, 867)      4.0
  (0, 983)      4.0
  (0, 997)      5.0
  (0, 1002)     4.0
  (0, 1048)     4.0
  (0, 1116)     4.0
  (0, 1141)     4.0
  (0, 1322)     4.0
  (0, 1327)     4.0
  (0, 1513)     4.0
  (0, 1596)     4.0
  :     :
  (6039, 3418)  4.0
  (6039, 3436)  4.0
  (6039, 3439)  4.0
  (6039, 3464)  4.0
  (6039, 3471)  4.0
  (6039, 3477)  3.0
  (6039, 3478)  4.0
  (6039, 3486)  2.0
  (6039, 3500)  3.0
  (6039, 3507)  3.0
  (6039, 3510)  4.0
  (6039, 3533)  1.0
  (6039, 3534)  1.0
  (6039, 3537)  5.0
  (6039, 3542)  4.0
  (6039, 3555)  5.0
  (6039, 3567)  5.0
  (6039, 3574)  1.0
  (6039, 3600)  5.0
  (6039, 3614)  4.0
  (6039, 3615)  5.0
  (6039, 3618)  5.0
  (6039, 3621)  4.0
  (6039, 3643)  5.0
  (6039, 3651)  4.0
```

In [102]:

```
similarities = cosine_similarity(csr_mat)
print('pairwise dense output:\n {}\n'.format(similarities))
```

```
pairwise dense output:
 [[1.         0.09638153 0.12060981 ... 0.         0.17460369 0.13359025]
 [0.09638153 1.         0.1514786  ... 0.06611767 0.0664575  0.21827563]
 [0.12060981 0.1514786  1.         ... 0.12023352 0.09467506 0.13314404]
 ...
 [0.         0.06611767 0.12023352 ... 1.         0.16171426 0.09930008]
 [0.17460369 0.0664575  0.09467506 ... 0.16171426 1.         0.22833237]
 [0.13359025 0.21827563 0.13314404 ... 0.09930008 0.22833237 1.        ]]
```

In [103]:

```
new_cos_u=pd.DataFrame(similarities)
new_cos_u.index=use_mat.index
new_cos_u.columns=use_mat.index
```

In [104]:

```
new_cos_u
# Cosine Similarity for user ID
```

Out[104]:

UserID	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0	...	6031.0	6032.0	6033.0	6034.0	
UserID																
1.0	1.000000	0.096382	0.120610	0.132455	0.090158	0.179222	0.059678	0.138241	0.226148	0.254736	...	0.170588	0.082006	0.090961	0.033663	0
2.0	0.096382	1.000000	0.151479	0.171176	0.114394	0.100865	0.305787	0.211120	0.190198	0.227745	...	0.112503	0.091222	0.268565	0.014286	0.
3.0	0.120610	0.151479	1.000000	0.151227	0.062907	0.074603	0.138332	0.077656	0.126457	0.214487	...	0.092960	0.125864	0.161507	0.000000	0.
4.0	0.132455	0.171176	0.151227	1.000000	0.045094	0.013529	0.130339	0.100856	0.093651	0.121208	...	0.163629	0.093041	0.382803	0.000000	0.
5.0	0.090158	0.114394	0.062907	0.045094	1.000000	0.047449	0.126257	0.220817	0.261330	0.117508	...	0.100652	0.035732	0.065278	0.054151	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6036.0	0.186957	0.229011	0.143747	0.171158	0.294354	0.093899	0.122854	0.232304	0.240415	0.344021	...	0.131737	0.210550	0.193417	0.103780	0.
6037.0	0.136233	0.206660	0.107945	0.127703	0.173009	0.065911	0.111882	0.152506	0.226883	0.251853	...	0.142575	0.276651	0.130228	0.118971	0.
6038.0	0.000000	0.066118	0.120234	0.062907	0.020459	0.065711	0.000000	0.019242	0.093470	0.114232	...	0.108837	0.106897	0.040689	0.000000	0.
6039.0	0.174604	0.066457	0.094675	0.064634	0.027689	0.167303	0.014977	0.044660	0.046434	0.297386	...	0.118776	0.250994	0.053750	0.102168	0.
6040.0	0.133590	0.218276	0.133144	0.137968	0.241437	0.083436	0.080680	0.148123	0.215819	0.256789	...	0.154574	0.291988	0.115540	0.118840	0.

6040 rows × 6040 columns

In [105]:

```
user_id=int(input('Please enter a valid user name '))
```

```
Please enter a valid user name 325
```

In [106]:

```
new_cos_u.loc[user_id].sort_values(ascending=False).reset_index()[:6][1:]
# Top 5 user ID for user id 50
```

Out[106]:

	UserID	325.0
1	3766.0	0.356665
2	4336.0	0.293017
3	4181.0	0.283445
4	2418.0	0.281295
5	2390.0	0.273641

7.Take a movie name as user input and use KNN algorithm to recommend 5 similar movies based on Cosine Similarity. (With Genres)

In [107]:

```
mat1=rat.groupby('MovieID')['Rating'].mean().reset_index()
mat2=mov[['Movie ID','Title']].drop_duplicates()
mat3=mat1.merge(mat2,how='left',left_on='MovieID',right_on='Movie ID')[['Title','Rating']].drop_duplicates()
matrix4=matrix2.reset_index()
mat5=mat3.merge(matrix4).set_index('Title')
mat5=mat5.astype(float)
mov1=mat5
# Took all the genres and mean rating for all the movies
```

In [108]:

```
mat5
# Added average rating for each movie as a column
```

Out[108]:

Title	Rating	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Toy Story	4.146846	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Jumanji	3.201141	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Grumpier Old Men	3.016736	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
Waiting to Exhale	2.729412	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Father of the Bride Part II	3.006757	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Meet the Parents	3.635731	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Requiem for a Dream	4.115132	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Tigerland	3.666667	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Two Family House	3.900000	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Contender, The	3.780928	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

3706 rows × 17 columns

In [109]:

```
model=NearestNeighbors(metric='cosine',algorithm='brute',n_neighbors=20)
model.fit(mat5)
```

Out[109]:

```
NearestNeighbors(algorithm='brute', metric='cosine', n_neighbors=20)
```

In [110]:

```
movie_name=input('Please enter a valid movie name ')
```

```
Please enter a valid movie name Liar Liar
```

In [111]:

```
dis,ind=model.kneighbors([mat5.loc[movie_name]],n_neighbors=6)
mov1.reset_index().loc[ind[0]]['Title'][1:].reset_index()
# Top 5 movies similar to movie name entered
```

Out[111]:

	index	Title
0	2075	Cabinet of Dr. Ramirez, The
1	817	Celestial Clockwork
2	2920	Broadway Damage
3	2360	Six Ways to Sunday
4	1388	Liar Liar

8.Take a movie name as user input and use KNN algorithm to recommend 5 similar movies based on Cosine Similarity. (With ratings)

In [112]:

```
movie=per_cor_df.reset_index().drop('UserID',axis=1)
movie.fillna(0,inplace=True)
movie=movie.transpose()
```

In [113]:

```
movie
```

Out[113]:

Title	0	1	2	3	4	5	6	7	8	9	...	6030	6031	6032	6033	6034	6035	6036	6037	6038	6039
$1,000,000 Duck	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
'Night Mother	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	3.0	0.0	0.0	0.0	0.0
'Til There Was You	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
'burbs, The	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
...And Justice for All	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Zed & Two Noughts, A	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
Zero Effect	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
Zero Kelvin (Kjærlighetens kjøtere)	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
Zeus and Roxanne	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
eXistenZ	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	2.0	0.0	0.0	0.0	0.0

3664 rows × 6040 columns

In [114]:

```
mov_1=movie
```

In [115]:

```
model=NearestNeighbors(metric='cosine',algorithm='brute',n_neighbors=20)
model.fit(movie)
```

Out[115]:

```
NearestNeighbors(algorithm='brute', metric='cosine', n_neighbors=20)
```

In [116]:

```
movie_name=input('Please enter a valid movie name ')
```

```
Please enter a valid movie name Liar Liar
```

In [117]:

```
dis,ind=model.kneighbors([movie.loc[movie_name]],n_neighbors=6)
mov_1.reset_index().loc[ind[0]]['Title'][1:].reset_index()
# Top 5 movies similar to movie name entered
```

Out[117]:

	index	Title
0	2221	Mrs. Doubtfire
1	43	Ace Ventura: Pet Detective
2	996	Dumb & Dumber
3	1543	Home Alone
4	3526	Wayne's World

Collaborative Rec Sys - Matrix Factorisation (cmfrec)

In [118]:

```
mer2
```

Out[118]:

	Movie ID	Genres	Title	Release_year	UserID	MovieID	Rating	Ratings_Time	Ratings_Date	Gender	Age	Occupation	Zip-code
0	1	Adventure	Toy Story	1995	1.0	1.0	5.0	23:37:48	2001-01-06	F	Under 18	K-12 student	48067
1	1	Adventure	Toy Story	1995	6.0	1.0	4.0	04:30:08	2000-12-31	F	50-55	homemaker	55117
2	1	Adventure	Toy Story	1995	8.0	1.0	4.0	03:31:36	2000-12-31	M	25-34	programmer	11413
3	1	Adventure	Toy Story	1995	9.0	1.0	5.0	01:25:52	2000-12-31	M	25-34	technician/engineer	61614
4	1	Adventure	Toy Story	1995	10.0	1.0	5.0	01:34:34	2000-12-31	F	35-44	academic/educator	95370
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2064306	3952	Thriller	Contender, The	2000	5812.0	3952.0	4.0	07:34:59	2001-06-09	F	25-34	executive/managerial	92120
2064307	3952	Thriller	Contender, The	2000	5831.0	3952.0	3.0	14:52:05	2001-04-02	M	25-34	academic/educator	92120
2064308	3952	Thriller	Contender, The	2000	5837.0	3952.0	4.0	20:04:16	2002-01-24	M	25-34	executive/managerial	60607
2064309	3952	Thriller	Contender, The	2000	5927.0	3952.0	1.0	21:15:37	2001-01-18	M	35-44	sales/marketing	10003
2064310	3952	Thriller	Contender, The	2000	5998.0	3952.0	4.0	16:30:44	2001-09-29	M	18-24	college/grad student	61820

2064096 rows × 13 columns

In [119]:

```
mat=mer2[['UserID','Movie ID','Rating']]
mat.drop_duplicates(inplace=True)
mat.isna().sum()
```

Out[119]:

```
UserID      0
Movie ID    0
Rating      0
dtype: int64
```

In [120]:

```
mat.columns = ['UserId', 'ItemId', 'Rating']
```

9.cmfrec library to run matrix factorization. (results with d=4).

In [121]:

```
model = CMF(k=4, lambda_=0.2, user_bias=False, item_bias=False, verbose=False)
model.fit(mat)
```

Out[121]:

```
Collective matrix factorization model
(explicit-feedback variant)
```

In [122]:

```
model.A_.shape
# User Matrix
```

Out[122]:

```
(6040, 4)
```

In [123]:

```
model.B_.shape
# Movies matrix
```

Out[123]:

```
(3706, 4)
```

In [124]:

```
top_items = model.topN(user=4, n=10)
```

In [125]:

```
top_items
# Top 10 movies recommendation for user ID 4
```

Out[125]:

```
array([3022, 2019, 1207,  912,  670,  923,  922,  904, 1212,   53])
```

In [126]:

```
mov[mov['Movie ID'].isin(list(top_items))]
# Below is the list of movies that should be recommended for user ID=4
```

Out[126]:

	Movie ID	Genres	Title	Release_year
52	53	Drama	Lamerica	1994
664	670	Drama	World of Apu, The (Apur Sansar)	1959
892	904	Mystery	Rear Window	1954
892	904	Thriller	Rear Window	1954
900	912	Drama	Casablanca	1942
900	912	Romance	Casablanca	1942
900	912	War	Casablanca	1942
910	922	Film-Noir	Sunset Blvd. (a.k.a. Sunset Boulevard)	1950
911	923	Drama	Citizen Kane	1941
1189	1207	Drama	To Kill a Mockingbird	1962
1194	1212	Mystery	Third Man, The	1949
1194	1212	Thriller	Third Man, The	1949
1950	2019	Drama	Seven Samurai (The Magnificent Seven) (Shichin...	1994
2953	3022	Comedy	General, The	1927

In [127]:

```
rm = mat.pivot(index = 'UserId', columns ='ItemId', values = 'Rating').fillna(0)
rm=rm.astype(int)
```

MAPE error

In [128]:

```
rm__ = np.dot(model.A_, model.B_.T) + model.glob_mean_
mape(rm.values[rm > 0], rm__[rm > 0])
```

Out[128]:

```
0.3460533181914965
```

RMSE error

In [129]:

```
sqrt(mse(rm.values[rm > 0], rm__[rm > 0]))
```

Out[129]:

```
1.1650192822398189
```

10. Embeddings for item-item similarity

In [130]:

```
i_i=pd.DataFrame(model.B_)
i_i=i_i.set_index(mat['ItemId'].unique())
```

In [131]:

```
similarities = cosine_similarity(i_i)
print('pairwise dense output:\n {}\n'.format(similarities))
```

```
pairwise dense output:
 [[ 1.0000001   0.31631672 -0.0144735  ...  0.8373657   0.8300854
   0.92047894]
 [ 0.31631672  0.99999994  0.91461974 ...  0.621353    0.341515
   0.511277  ]
 [-0.0144735   0.91461974  1.         ...  0.26037967 -0.01575968
   0.23757568]
 ...
 [ 0.8373657   0.621353    0.26037967 ...  0.9999999   0.9142154
   0.8605141 ]
 [ 0.8300854   0.341515   -0.01575968 ...  0.9142154   1.
   0.8754104 ]
 [ 0.92047894  0.511277    0.23757568 ...  0.8605141   0.8754104
   0.99999994]]
```

In [132]:

```
similarities=pd.DataFrame(similarities)
similarities.index=i_i.index
similarities.columns=i_i.index
```

In [133]:

```
similarities
```

Out[133]:

	1	2	3	4	5	6	7	8	9	10	...	3943	3944	3945	39
1	1.000000	0.316317	-0.014474	-0.087672	0.107694	0.636013	0.366739	0.134621	-0.259054	0.516143	...	-0.004426	0.026651	-0.615256	-0.5263
2	0.316317	1.000000	0.914620	0.813512	0.954992	0.163574	0.953152	0.949543	0.780840	0.768004	...	0.606345	0.811515	0.459282	0.6361
3	-0.014474	0.914620	1.000000	0.748819	0.956172	0.052902	0.887822	0.861588	0.951130	0.743817	...	0.505311	0.779884	0.583331	0.8079
4	-0.087672	0.813512	0.748819	1.000000	0.827040	-0.297670	0.689215	0.949278	0.701644	0.268749	...	0.759319	0.804295	0.838398	0.8330
5	0.107694	0.954992	0.956172	0.827040	1.000000	-0.074584	0.957788	0.913744	0.836171	0.674375	...	0.465447	0.710672	0.587705	0.7653
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3948	0.522270	0.481933	0.480474	-0.065563	0.342486	0.861306	0.485061	0.242747	0.423240	0.919810	...	0.151762	0.398220	-0.366131	-0.0573
3949	0.232141	-0.237314	-0.342171	-0.302888	-0.484073	0.721827	-0.446119	-0.209407	-0.213338	0.032708	...	0.384769	0.204603	-0.385405	-0.4199
3950	0.837366	0.621353	0.260380	0.422911	0.400643	0.458772	0.546077	0.574249	0.051362	0.496932	...	0.496513	0.471946	-0.123324	-0.0978
3951	0.830085	0.341515	-0.015760	0.140845	0.058678	0.668494	0.216560	0.302246	-0.151249	0.376487	...	0.472975	0.367025	-0.347472	-0.3526
3952	0.920479	0.511277	0.237576	0.067080	0.271575	0.813040	0.484297	0.335238	0.055439	0.734984	...	0.275446	0.353805	-0.462523	-0.3054

3706 rows × 3706 columns

In [134]:

```
movie_id=int(input('Please enter a valid movie id '))
```

```
Please enter a valid movie id 36
```

In [135]:

```
similarities.loc[movie_id].sort_values(ascending=False).reset_index()[:6][1:]
# Top 5 movie ID's based on movie ID
```

Out[135]:

	index	36
1	58	0.999591
2	1944	0.997341
3	3246	0.997023
4	1946	0.996898
5	2202	0.995877

11. Embeddings for user_user similarity

In [136]:

```
u_u=pd.DataFrame(model.A_)
u_u=u_u.set_index(mat['UserId'].unique())
```

In [137]:

```
similarities = cosine_similarity(u_u)
print('pairwise dense output:\n {}\n'.format(similarities))
```

```
pairwise dense output:
 [[ 0.99999994 -0.01194302  0.32269195 ... -0.23823811  0.91291386
   0.27430463]
 [-0.01194302  1.         -0.50183123 ...  0.01506824 -0.41183046
   0.6323067 ]
 [ 0.32269195 -0.50183123  1.0000001  ...  0.27909845  0.5104769
   0.34769085]
 ...
 [-0.23823811  0.01506824  0.27909845 ...  1.         -0.27668148
   0.34630096]
 [ 0.91291386 -0.41183046  0.5104769  ... -0.27668148  1.0000001
   0.00293577]
 [ 0.27430463  0.6323067   0.34769085 ...  0.34630096  0.00293577
   1.0000001 ]]
```

In [138]:

```
similarities=pd.DataFrame(similarities)
similarities.index=u_u.index
similarities.columns=u_u.index
```

In [139]:

```
similarities
```

Out[139]:

	1.0	6.0	8.0	9.0	10.0	18.0	19.0	21.0	23.0	26.0	...	3542.0	4419.0	2697.0	4
1.0	1.000000	-0.011943	0.322692	-0.258899	0.782090	0.365167	0.004121	-0.619259	-0.480230	-0.559647	...	-0.099039	-0.539150	0.713966	0.42
6.0	-0.011943	1.000000	-0.501831	0.047151	0.593504	0.304282	0.060235	-0.633294	0.093255	0.709567	...	0.992025	0.328189	0.670248	0.63
8.0	0.322692	-0.501831	1.000000	0.670014	-0.019452	0.554809	0.732278	0.482958	0.436151	-0.185899	...	-0.574160	0.025058	-0.224698	0.32
9.0	-0.258899	0.047151	0.670014	1.000000	-0.149923	0.600942	0.929218	0.565227	0.907484	0.593571	...	0.007441	0.617134	-0.290258	0.53
10.0	0.782090	0.593504	-0.019452	-0.149923	1.000000	0.571742	0.023751	-0.848123	-0.352448	-0.013527	...	0.528261	-0.309890	0.948496	0.72
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3235.0	0.728676	-0.219404	0.861291	0.429774	0.454529	0.641146	0.625828	-0.026720	0.169491	-0.271564	...	-0.325109	-0.125198	0.289698	0.59
3298.0	0.814852	-0.544395	0.472791	-0.313190	0.345115	0.183623	-0.163240	-0.184653	-0.572927	-0.882326	...	-0.591850	-0.778227	0.209623	-0.06
5145.0	-0.238238	0.015068	0.279098	0.450678	-0.030812	0.662166	0.149482	0.379530	0.148287	0.252010	...	0.066987	-0.215585	-0.305666	0.11
768.0	0.912914	-0.411830	0.510477	-0.230778	0.461650	0.188868	0.022189	-0.302453	-0.430098	-0.777068	...	-0.493948	-0.565300	0.383395	0.14
5727.0	0.274305	0.632307	0.347691	0.641171	0.637662	0.867332	0.685151	-0.249878	0.452257	0.590515	...	0.565881	0.315081	0.514626	0.96

6040 rows × 6040 columns

In [140]:

```
user_id=int(input('Please enter a valid user id '))
```

```
Please enter a valid user id 325
```

In [141]:

```
similarities.loc[user_id].sort_values(ascending=False).reset_index()[:6][1:]
# Top 5 user id recommendation
```

Out[141]:

	index	325.0
1	3254.0	0.999456
2	1553.0	0.995471
3	4048.0	0.994850
4	5081.0	0.993550
5	2984.0	0.991189

12.Get d=2 embeddings, and plot the results. Write down your analysis from this visualisation. (Compare with other visualization techniques)

In [142]:

```
model = CMF(k=2, lambda_=0.1, user_bias=False, item_bias=False, verbose=False)
model.fit(mat)
```

Out[142]:

```
Collective matrix factorization model
(explicit-feedback variant)
```

In [143]:

```
A=model.A_
A=pd.DataFrame(A)
```

In [144]:

```
model.A_.shape
# User Matrix
```

Out[144]:

```
(6040, 2)
```

In [145]:

```
B=model.B_
B=pd.DataFrame(B)
```

In [146]:

```
model.B_.shape
# Movie matrix
```

Out[146]:

```
(3706, 2)
```

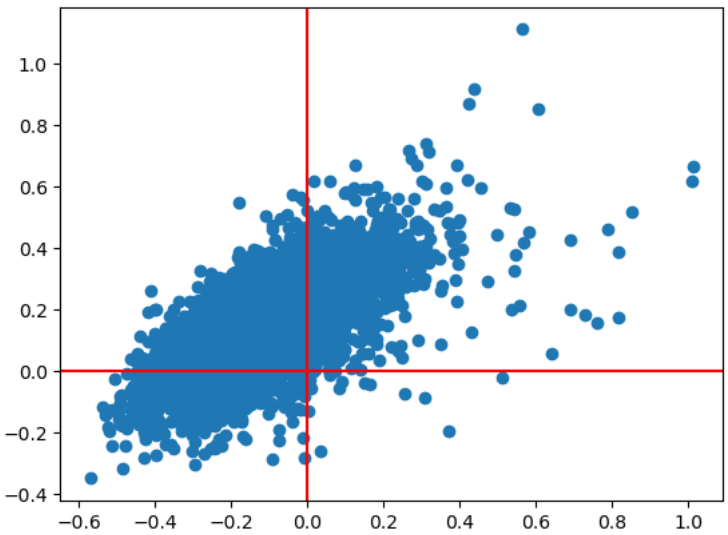
User_ID in the same quadrant are similar to each other. We can separate Users into 4 groups

In [147]:

```
plt.scatter(A[0],A[1])
plt.axhline(y = 0, color = 'r', linestyle = '-')
plt.axvline(x = 0, color = 'r', linestyle = '-')
```

Out[147]:

```
<matplotlib.lines.Line2D at 0x1d88596b5b0>
```



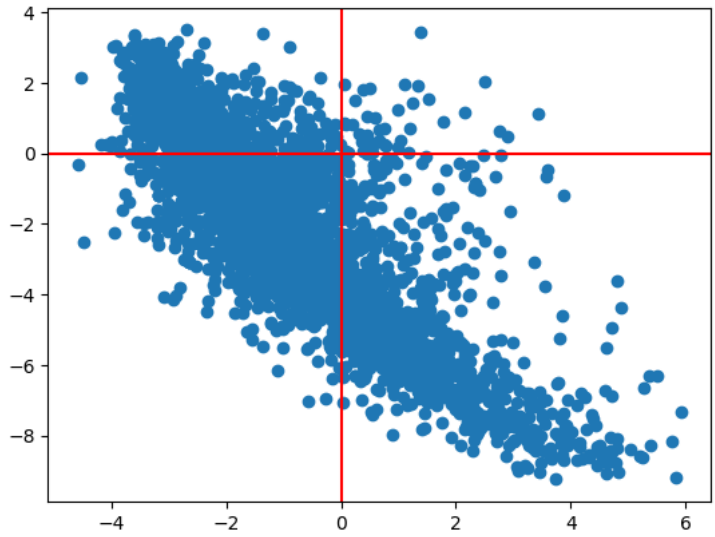
Movie_ID in the same quadrant are similar to each other. We can separate Users into 4 groups

In [148]:

```
plt.scatter(B[0],B[1])
plt.axhline(y = 0, color = 'r', linestyle = '-')
plt.axvline(x = 0, color = 'r', linestyle = '-')
```

Out[148]:

```
<matplotlib.lines.Line2D at 0x1d88576ff70>
```



Splitting Movies into 4 quadrants

In [149]:

```
B[2]=np.nan
```

In [150]:

```
B
```

Out[150]:

	0	1	2
0	-3.291066	1.268009	NaN
1	-0.746644	-3.420394	NaN
2	-0.105144	-4.061369	NaN
3	0.527322	-4.495835	NaN
4	-0.298230	-4.402603	NaN
...	...	...	...
3701	-1.731441	-1.183783	NaN
3702	-2.699752	1.553539	NaN
3703	-2.127834	-0.648497	NaN
3704	-3.408318	0.643376	NaN
3705	-2.077575	-0.161609	NaN

3706 rows × 3 columns

In [151]:

```
for i in range(len(np.array(B))):
    if np.array(B)[i][0]<0 and np.array(B)[i][1]<0:
        B[2][i]=3
    elif np.array(B)[i][0]>0 and np.array(B)[i][1]>0:
        B[2][i]=1
    elif np.array(B)[i][0]<0 and np.array(B)[i][1]>0:
        B[2][i]=2
    else:
        B[2][i]=4
```

In [152]:

```
B[2].value_counts()
```

Out[152]:

```
3.0    1808
4.0    1018
2.0     840
1.0      40
Name: 2, dtype: int64
```

In [153]:

```
B.index=mat['ItemId'].drop_duplicates()
B=B.reset_index()
C=mer2[['UserID','MovieID','Rating']].drop_duplicates().groupby('MovieID')['Rating'].mean().reset_index()
```

In [154]:

```
C
```

Out[154]:

	MovieID	Rating
0	1.0	4.146846
1	2.0	3.201141
2	3.0	3.016736
3	4.0	2.729412
4	5.0	3.006757
...	...	...
3701	3948.0	3.635731
3702	3949.0	4.115132
3703	3950.0	3.666667
3704	3951.0	3.900000
3705	3952.0	3.780928

3706 rows × 2 columns

In [155]:

```
D=B.merge(C,how='left',left_on='ItemId', right_on='MovieID')
D.groupby(2)['Rating'].agg(['mean']).sort_values('mean',ascending=False)
# Mean rating of movies in each quadrant
```

Out[155]:

	mean
2	
2.0	4.004031
1.0	3.531194
3.0	3.288539
4.0	2.507881

We can conclude that quadrant 2 has highest rating movies and quadrant 4 has low rating movies.

We can recommend similar movies to people is each quadrant

PCA Visualisation

In [156]:

```
AA=mat.pivot(index='UserId',columns='ItemId',values='Rating').fillna(0)
pca = PCA(n_components = 2)
A_user=pd.DataFrame(pca.fit_transform(AA))
B_movie=pd.DataFrame(pca.fit_transform(AA.T))
```

In [157]:

```
A_user.shape
```

Out[157]:

```
(6040, 2)
```

In [158]:

```
B_movie.shape
```

Out[158]:

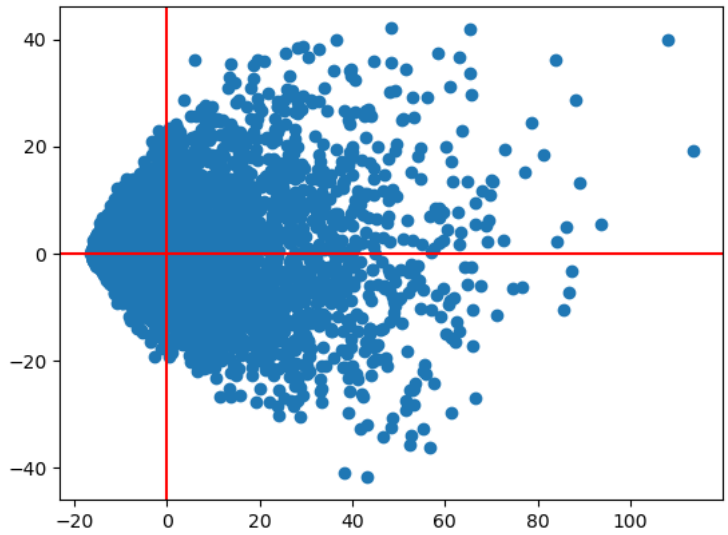
```
(3706, 2)
```

In [159]:

```
plt.scatter(A_user[0],A_user[1])
plt.axhline(y = 0, color = 'r', linestyle = '-')
plt.axvline(x = 0, color = 'r', linestyle = '-')
```

Out[159]:

```
<matplotlib.lines.Line2D at 0x1d8855428e0>
```

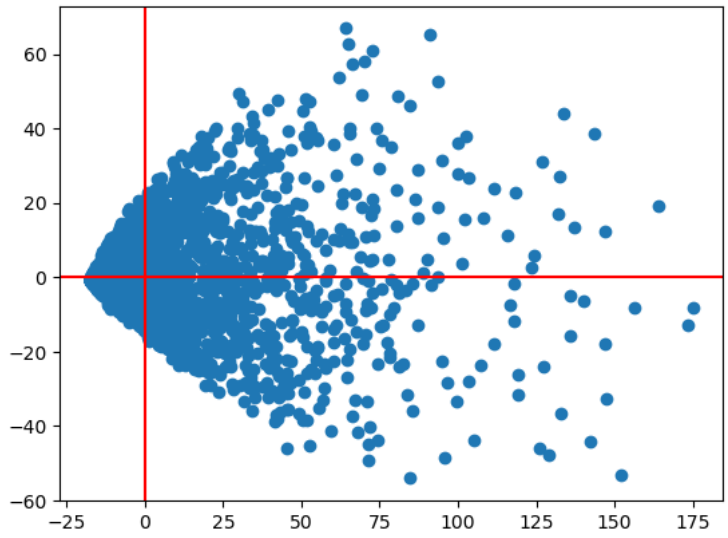


In [160]:

```
plt.scatter(B_movie[0],B_movie[1])
plt.axhline(y = 0, color = 'r', linestyle = '-')
plt.axvline(x = 0, color = 'r', linestyle = '-')
```

Out[160]:

```
<matplotlib.lines.Line2D at 0x1d885377b80>
```



Mean rating based on PCA for each quadrant

In [161]:

```
B_movie[2]=np.nan
B_movie
```

Out[161]:

	0	1	2
0	117.751348	-11.908854	NaN
1	25.481185	-20.568968	NaN
2	7.231473	-11.977140	NaN
3	-8.521629	-2.840425	NaN
4	-0.421377	-9.938390	NaN
...	...	...	...
3701	29.904978	-8.145984	NaN
3702	5.629524	3.205274	NaN
3703	-13.167280	0.583740	NaN
3704	-15.761895	0.849146	NaN
3705	6.572580	2.917552	NaN

3706 rows × 3 columns

In [162]:

```
for i in range(len(np.array(B_movie))):
    if np.array(B_movie)[i][0]<0 and np.array(B_movie)[i][1]<0:
        B_movie[2][i]=3
    elif np.array(B_movie)[i][0]>0 and np.array(B_movie)[i][1]>0:
        B_movie[2][i]=1
    elif np.array(B_movie)[i][0]<0 and np.array(B_movie)[i][1]>0:
        B_movie[2][i]=2
    else:
        B_movie[2][i]=4
```

In [163]:

```
B_movie
```

Out[163]:

	0	1	2
0	117.751348	-11.908854	4.0
1	25.481185	-20.568968	4.0
2	7.231473	-11.977140	4.0
3	-8.521629	-2.840425	3.0
4	-0.421377	-9.938390	3.0
...	...	...	...
3701	29.904978	-8.145984	4.0
3702	5.629524	3.205274	1.0
3703	-13.167280	0.583740	2.0
3704	-15.761895	0.849146	2.0
3705	6.572580	2.917552	1.0

3706 rows × 3 columns

In [164]:

```
B_movie[2].value_counts()
```

Out[164]:

```
3.0    1609
2.0     947
4.0     618
1.0     532
Name: 2, dtype: int64
```

In [165]:

```
B_movie.index=mat['ItemId'].drop_duplicates()
B_movie=B_movie.reset_index()
D=B_movie.merge(C,how='left',left_on='ItemId', right_on='MovieID')
D.groupby(2)['Rating'].agg(['mean']).sort_values('mean',ascending=False)
```

Out[165]:

2	mean
1.0	3.905960
2.0	3.571202
4.0	3.377570
3.0	2.769482

13. Follow the User-based approach

In [166]:

```
set_movies=list(set(list(mov['Title'])))
user_rex_rat=random.sample(set_movies,10)
user_rex_rat
# Random 10 movie
```

Out[166]:

```
['Holy Smoke',
 'Shanghai Surprise',
 'Wild Bunch, The',
 'Muppet Treasure Island',
 'Poetic Justice',
 'Light Years',
 'Jeanne and the Perfect Guy (Jeanne et le garçon formidable) (1998',
 'Mummy, The',
 'Mission: Impossible 2',
 'Only You']
```

In [167]:

```
rating_dict=dict()
for i in user_rex_rat:
    print('Please provide your rating from scale of 1-5 for {} : '.format(i))
    j=int(input('Enter rating'))
    rating_dict[i]=j
# Ratings provided by the user for 10 movies
```

```
Please provide your rating from scale of 1-5 for Holy Smoke :
Enter rating3
Please provide your rating from scale of 1-5 for Shanghai Surprise :
Enter rating2
Please provide your rating from scale of 1-5 for Wild Bunch, The :
Enter rating5
Please provide your rating from scale of 1-5 for Muppet Treasure Island :
Enter rating4
Please provide your rating from scale of 1-5 for Poetic Justice :
Enter rating1
Please provide your rating from scale of 1-5 for Light Years :
Enter rating2
Please provide your rating from scale of 1-5 for Jeanne and the Perfect Guy (Jeanne et le garçon formidable) (1998 :
Enter rating5
Please provide your rating from scale of 1-5 for Mummy, The :
Enter rating3
Please provide your rating from scale of 1-5 for Mission: Impossible 2 :
Enter rating4
Please provide your rating from scale of 1-5 for Only You :
Enter rating2
```

In [168]:

```
rating_dict
```

Out[168]:

```
{'Holy Smoke': 3,
 'Shanghai Surprise': 2,
 'Wild Bunch, The': 5,
 'Muppet Treasure Island': 4,
 'Poetic Justice': 1,
 'Light Years': 2,
 'Jeanne and the Perfect Guy (Jeanne et le garçon formidable) (1998': 5,
 'Mummy, The': 3,
 'Mission: Impossible 2': 4,
 'Only You': 2}
```

Users who've watched the same movies as the new user.

In [169]:

```
list_users_watch=[]
for i in user_rex_rat:
    list_users_watch.extend(list(mer2[mer2['Title']==i]['UserID'].values))
    #print(mer2[mer2['Title']==i]['UserID'])
list_users_watch=list(set(list_users_watch))
```

In [170]:

```
len(list_users_watch)
```

Out[170]:

```
2280
```

Sorting the old users by the count of most movies in common with the new user

Top 100 users

In [171]:

```
user_com=mer2[mer2['UserID'].isin(list_users_watch)]
user_list=user_com[['UserID','Title']].drop_duplicates().groupby('UserID')['Title'].apply(list).reset_index()
user_list['new_user']=0
for i in range(len(user_list)):
    user_list['new_user'][i]=reduce(lambda x, y: x+user_list['Title'][i].count(y), set(user_rex_rat),0)
user_list.sort_values('new_user',ascending=False,inplace=True)
top_100_user_id=user_list['UserID'][:100]
# Sort the old users by the count of most movies in common with the new user
```

In [172]:

```
top_100_user_id.values
```

Out[172]:

```
array([4169., 1941., 5795., 4277., 4725., 1449., 1051., 2063., 1447.,
       5530., 4808., 1880.,  869., 2225., 5643., 4064., 1340., 1333.,
       3829., 4579., 5880., 4227., 5878., 1285.,  216., 2116., 1354.,
       2907., 5367., 4083., 4673.,  550., 1181., 5605.,  881., 1647.,
       1203.,  889.,  531.,   48., 3934.,  752.,  329., 1033., 5268.,
       2996., 2810., 2015.,  343., 2012., 5277., 3261.,  352., 5282.,
        549., 3191., 4523.,  770., 5333.,  566.,  676., 4213., 2042.,
       3507., 4088., 4510., 4508., 2054., 1490., 5250., 4085., 3259.,
       3768., 3471., 5359., 4591., 1748.,  424., 5306., 1912., 3841.,
       5634., 2592., 1501.,  509., 4635., 1842., 1448., 4115., 1473.,
        854.,  524., 2980., 1778., 1483., 2946., 2934., 1958., 1524.,
       5011.])
```

In [173]:

```
all_user=per_cor_df.iloc[top_100_user_id.values]
newuser=pd.DataFrame(rating_dict.items())
newuser=newuser.set_index(0).transpose()
```

In [174]:

```
newuser
```

Out[174]:

	Holy Smoke	Shanghai Surprise	Wild Bunch, The	Muppet Treasure Island	Poetic Justice	Light Years	Jeanne and the Perfect Guy (Jeanne et le garçon formidable) (1998	Mummy, The	Mission: Impossible 2	Only You
1	3	2	5	4	1	2	5	3	4	2

In [175]:

```
newuser=newuser.T
newuser.columns=[7000]
```

In [176]:

```
newuser
```

Out[176]:

	7000
0	
Holy Smoke	3
Shanghai Surprise	2
Wild Bunch, The	5
Muppet Treasure Island	4
Poetic Justice	1
Light Years	2
Jeanne and the Perfect Guy (Jeanne et le garçon formidable) (1998	5
Mummy, The	3
Mission: Impossible 2	4
Only You	2

Calculated a Similarity Score for each user using the Pearson Correlation function.

In [177]:

```
all_user=all_user.T
all_user
```

Out[177]:

UserID	4170.0	1942.0	5796.0	4278.0	4726.0	1450.0	1052.0	2064.0	1448.0	5531.0	...	855.0	525.0	2981.0	1779.0	1484.0	2947.0	2935.0	19
Title																			
$1,000,000 Duck	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	
'Night Mother	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
'Til There Was You	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	
'burbs, The	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	
...And Justice for All	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	4.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
Zed & Two Noughts, A	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	
Zero Effect	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	5.0	0.0	0.0	
Zero Kelvin (Kjærlighetens kjøtere)	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	
Zeus and Roxanne	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	
eXistenZ	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	5.0	2.0	0.0	0.0	

3664 rows × 100 columns

Get the top 10 users with the highest similarity indices,

all the movies for these users, and add Weighted movie Ratings by Multiplying the Rating to the Similarity Index.

In [178]:

```
full_mat=all_user.join(newuser)
full_mat=full_mat.fillna(0)
top_10=full_mat.corr(method ='pearson').loc[7000].sort_values(ascending=False).reset_index()[1:11]
top_10.columns=['UserID','Sim_ind']
```

In [179]:

```
top_10
```

Out[179]:

	UserID	Sim_ind
1	5012.0	0.147647
2	5360.0	0.063995
3	1341.0	0.063642
4	330.0	0.054490
5	4511.0	0.050019
6	4278.0	0.042573
7	4214.0	0.041200
8	1286.0	0.040985
9	1779.0	0.037592
10	5644.0	0.036906

In [180]:

```
rat1=rat
rat1.drop(['Ratings_Time','Ratings_Date'],axis=1,inplace=True)
topUsersRating=top_10.merge(rat1, left_on='UserID', right_on='UserID', how='inner')
topUsersRating['Rating']=topUsersRating['Rating'].astype(float)
topUsersRating['weightedRating'] = topUsersRating['Sim_ind']*topUsersRating['Rating']
```

In [181]:

```
topUsersRating
```

Out[181]:

	UserID	Sim_ind	MovieID	Rating	weightedRating
0	5012.0	0.147647	599	5.0	0.738236
1	5012.0	0.147647	3037	3.0	0.442942
2	5012.0	0.147647	3494	3.0	0.442942
3	5012.0	0.147647	2527	2.0	0.295294
4	5012.0	0.147647	3508	4.0	0.590589
...	...	...	...	...	...
728	5644.0	0.036906	500	5.0	0.184530
729	5644.0	0.036906	527	5.0	0.184530
730	5644.0	0.036906	538	3.0	0.110718
731	5644.0	0.036906	2018	3.0	0.110718
732	5644.0	0.036906	2028	3.0	0.110718

733 rows × 5 columns

In [182]:

```
tempTopUsersRating = topUsersRating.groupby('MovieID').sum()[['Sim_ind','weightedRating']]
tempTopUsersRating.columns = ['sum_similarityIndex','sum_weightedRating']
tempTopUsersRating
```

Out[182]:

MovieID	sum_similarityIndex	sum_weightedRating
1	0.074498	0.260401
3	0.037592	0.112777
10	0.037592	0.112777
11	0.036906	0.110718
12	0.063642	0.318208
...	...	...
3910	0.037592	0.150370
3919	0.063642	0.063642
3940	0.063642	0.063642
3942	0.063642	0.127283
3948	0.054490	0.163471

543 rows × 2 columns

In [183]:

```
recommendation_df = pd.DataFrame()
#Now we take the weighted average
recommendation_df['weighted average recommendation score'] = tempTopUsersRating['sum_weightedRating']/tempTopUsersRating['sum_similarityIr
recommendation_df['movieId'] = tempTopUsersRating.index
recommendation_df
```

Out[183]:

MovieID	weighted average recommendation score	movieId
1	3.495393	1
3	3.000000	3
10	3.000000	10
11	3.000000	11
12	5.000000	12
...	...	...
3910	4.000000	3910
3919	1.000000	3919
3940	1.000000	3940
3942	2.000000	3942
3948	3.000000	3948

543 rows × 2 columns

Calculated the average recommendation score by dividing the Weighted Rating by the Similarity Index and select movies with the highest score i.e., 5.

In [184]:

```
recommendation_df = recommendation_df.sort_values(by='weighted average recommendation score', ascending=False)
recommendation_df.head(10)
```

Out[184]:

MovieID	weighted average recommendation score	movieId
2723	5.0	2723
2107	5.0	2107
3029	5.0	3029
3030	5.0	3030
1094	5.0	1094
1097	5.0	1097
1101	5.0	1101
2115	5.0	2115
1196	5.0	1196
954	5.0	954

Recommending 10 movies based on the ratings given by old users who are similar to the new user.

In [185]:

```
mov.loc[mov['Movie ID'].isin(recommendation_df.head(10)['movieId'].tolist())][['Title','Movie ID']].drop_duplicates().reset_index()
```

Out[185]:

	index	Title	Movie ID
0	942	Mr. Smith Goes to Washington	954
1	1078	Crying Game, The	1094
2	1081	E.T. the Extra-Terrestrial	1097
3	1085	Top Gun	1101
4	1178	Star Wars: Episode V - The Empire Strikes Back	1196
5	2038	Halloween: H20	2107
6	2046	Indiana Jones and the Temple of Doom	2115
7	2654	Mystery Men	2723
8	2960	Nighthawks	3029
9	2961	Yojimbo	3030

In [ ]:

```

```

Questionnaire:

1.Users of which age group have watched and rated the most number of movies?

Ans- 25-34

2.Users belonging to which profession have watched and rated the most movies?

Ans- college/grad student

3. Most of the users in our dataset who've rated the movies are Male. (T/F)

Ans- True

4. Most of the movies present in our dataset were released in which decade? 70s b. 90s c. 50s d.80s

Ans- 90's

5. The movie with maximum no. of ratings is ___.

Ans- Movie ID 1580, Name 'Men in Black'

6. Name the top 3 movies similar to 'Liar Liar' on the item-based approach.

Ans-

1.Man of the House

2.Me, Myself and Irene

3.Me Myself I

4.Maybe, Maybe Not (Bewegte Mann, Der)

5.Max Dugan Returns

7. On the basis of approach, Collaborative Filtering methods can be classified into _-based and _-based.

Ans- Item-based and User-based

8. Pearson Correlation ranges between _ to _ whereas, Cosine Similarity belongs to the interval between _ to _.

Ans- Pearson Correlation ranges between -1 to 1 whereas, Cosine Similarity belongs to the interval between 0 to 1.

9. Mention the RMSE and MAPE that you got while evaluating the Matrix Factorization model.

Ans-

MAPE error = 0.3460533181914965

RMSE error = 1.1650192822398189

10. Give the sparse 'row' matrix representation for the following dense matrix -

[[1 0] [3 7]]

In [186]:

```
matrix_given=np.array([[1, 0],
[3, 7]])
```

In [187]:

```
A=csr_matrix(matrix_given)
```

In [188]:

```
print(A)
```

```
  (0, 0)        1
  (1, 0)        3
  (1, 1)        7
```

In [189]:

```
print(A.todense())
```

```
[[1 0]
 [3 7]]
```