In [5]:
```python
import pandas as pd
import numpy as np
from datetime import datetime
```

In [6]:
```python
movies=pd.read_csv(r"C:\Users\kanwar\Downloads\movies.csv")
```

In [7]:
```python
ratings=pd.read_csv(r"C:\Users\kanwar\Downloads\ratings.csv")
users=pd.read_csv(r"C:\Users\kanwar\Downloads\users.csv")
```

In [5]:
```python
movies.shape
```

Out[5]:
```
(10329, 3)
```

In [6]:
```python
ratings.shape
```

Out[6]:
```
(105339, 4)
```

In [7]:
```python
ratings.columns
```

Out[7]:
```
Index(['userId', 'movieId', 'rating', 'timestamp'], dtype='object')
```

In [1]:
```python
## Popular movies are those which have more number of ratings
```

In [8]:
```python
movies_ls=ratings.movieId.value_counts()[:1000].index.to_list()
```

In [9]:
```python
movies=movies[movies.movieId.isin(movies_ls)]
movies.shape
```

Out[9]:
```
(1000, 3)
```

In [10]:
```python
ratings=ratings[ratings.movieId.isin(movies_ls)]
ratings.shape
```

Out[10]:
```
(63250, 4)
```

In [8]:
```python
m=movies.copy()
m
```

Out[8]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
10324	146684	Cosmic Scrat-tastrophe (2015)	Animation|Children|Comedy
10325	146878	Le Grand Restaurant (1966)	Comedy
10326	148238	A Very Murray Christmas (2015)	Comedy
10327	148626	The Big Short (2015)	Drama
10328	149532	Marco Polo: One Hundred Eyes (2015)	(no genres listed)

10329 rows × 3 columns

In [9]:
```python
## split geners into a list of genres for the movie
m.genres=m.genres.str.split('|')
m.head()
```

Out[9]:

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

In [10]:

```
m=m.explode('genres')
m.head()
```

Out[10]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure
0	1	Toy Story (1995)	Animation
0	1	Toy Story (1995)	Children
0	1	Toy Story (1995)	Comedy
0	1	Toy Story (1995)	Fantasy

In [11]:

```
m=m.pivot(index='movieId',columns='genres',values='title')
m.head()
```

Out[11]:

genres	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	IMAX	Music
movieId														
1	NaN	NaN	Toy Story (1995)	Toy Story (1995)	Toy Story (1995)	Toy Story (1995)	NaN	NaN	NaN	Toy Story (1995)	NaN	NaN	NaN	Na
2	NaN	NaN	Jumanji (1995)	NaN	Jumanji (1995)	NaN	NaN	NaN	NaN	Jumanji (1995)	NaN	NaN	NaN	Na
3	NaN	NaN	NaN	NaN	NaN	Grumpier Old Men (1995)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	NaN	NaN	NaN	NaN	NaN	Waiting to Exhale (1995)	NaN	NaN	Waiting to Exhale (1995)	NaN	NaN	NaN	NaN	Na
5	NaN	NaN	NaN	NaN	NaN	Father of the Bride Part II (1995)	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

In [3]:

```
## Movie Feature matrix:
```

In [12]:

```
m=~m.isna()
m=m.astype(int)
m.head()
```

Out[12]:

genres	(no genres listed)	Action	Adventure	Animation	Children	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	IMAX	Musica
movieId														
1	0	0	1	1	1	1	0	0	0	1	0	0	0	0
2	0	0	1	0	1	0	0	0	0	1	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	0	0	0	0
4	0	0	0	0	0	1	0	0	1	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0

In [16]:
```
m.shape
```
Out[16]: (1000, 19)

In [17]:
```
## movieid=1
m.iloc[1].values
```
Out[17]:
```
array([0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [18]:
```
## movieid=7
m.iloc[7].values
```
Out[18]:
```
array([0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0])
```

In [19]:
```
m.iloc[7].values !=m.iloc[1].values
```
Out[19]:
```
array([False,  True, False,  True,  True, False, False,  True,  True,
       False, False, False, False, False,  True, False, False, False,
       False])
```

In [20]:
```
def hamming_distance(a,b):
    return (a!=b).sum()
```

In [21]:
```
## hamming_distance check the disimilarity between the strings

hamming_distance(m.iloc[7].values,m.iloc[1].values)
```

Out[21]: np.int64(6)

In [4]:
```
## creating Movie similarity matrix
```

In [22]:
```
rank=[]
for query in m.index:
    for candidate in m.index:
        if candidate==query:
            continue
        rank.append([query,candidate,hamming_distance(m.loc[query],m.loc[candidate])])
```

In [23]:
```
ranks=pd.DataFrame(rank,columns=['query','candidate','rank'])
ranks.head()
```

Out[23]:

	query	candidate	rank
0	1	2	2
1	1	3	5
2	1	5	4
3	1	6	8
4	1	7	5

Content Based Recommendation

Recommend Movie basis Item Item similarity i.e Movie Movie similarities

This is content based recommendation because the movie genre is used to check similarities

In [24]:
```
ranks=ranks.merge(movies[['movieId','title']],left_on='query',right_on='movieId').rename(columns={'title':'query_title'}).drop
ranks=ranks.merge(movies[['movieId','title']],left_on='candidate',right_on='movieId').rename(columns={'title':'candidate_title
ranks=ranks.sort_values(by=['query','rank'])
ranks.head(10)
```

Out[24]:

	query	candidate	rank	query_title	candidate_title
541	1	2294	0	Toy Story (1995)	Antz (1998)
668	1	3114	0	Toy Story (1995)	Toy Story 2 (1999)
793	1	4886	0	Toy Story (1995)	Monsters, Inc. (2001)
186	1	673	1	Toy Story (1995)	Space Jam (1996)
552	1	2355	1	Toy Story (1995)	Bug's Life, A (1998)
767	1	4306	1	Toy Story (1995)	Shrek (2001)
806	1	5218	1	Toy Story (1995)	Ice Age (2002)
844	1	6377	1	Toy Story (1995)	Finding Nemo (2003)
981	1	78499	1	Toy Story (1995)	Toy Story 3 (2010)
0	1	2	2	Toy Story (1995)	Jumanji (1995)

Recommendation Problem using Linear Regression

In [25]:
```
users.head()
```

Out[25]:

	userId	age	time_spent_per_day
0	1	16	3.976315
1	2	24	1.891303
2	3	20	4.521478
3	4	23	2.095284
4	5	35	1.759860

In [26]:
```
ratings.head()
```

Out[26]:

	userId	movieId	rating	timestamp
0	1	16	4.0	1217897793
1	1	24	1.5	1217895807
2	1	32	4.0	1217896246
3	1	47	4.0	1217896556
4	1	50	4.0	1217896523

In [27]:
```
r=ratings.copy()
r['hour']=r['timestamp'].apply(lambda x: datetime.fromtimestamp(x).hour)
```

In [28]:
```
users=users.merge(r.groupby('userId').rating.mean().reset_index(),on='userId')
users=users.merge(r.groupby('userId').hour.mean().reset_index(),on='userId')
```

In [29]:
```
users.head()
```

Out[29]:

	userId	age	time_spent_per_day	rating	hour
0	1	16	3.976315	3.691589	5.616822
1	2	24	1.891303	3.923077	21.000000
2	3	20	4.521478	3.806452	14.370968
3	4	23	2.095284	4.159420	8.000000
4	5	35	1.759860	2.864865	0.513514

In [30]:

```
u=users.copy()
u=users.set_index('userId')
u.columns=['age','time_spent_per_day','u_avg_rating','hour']
u.head()
```

Out[30]:

	age	time_spent_per_day	u_avg_rating	hour
userId				
1	16	3.976315	3.691589	5.616822
2	24	1.891303	3.923077	21.000000
3	20	4.521478	3.806452	14.370968
4	23	2.095284	4.159420	8.000000
5	35	1.759860	2.864865	0.513514

In [31]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
u=pd.DataFrame(scaler.fit_transform(u),columns=u.columns,index=u.index)
u.head()
```

Out[31]:

userId	age	time_spent_per_day	u_avg_rating	hour
1	-1.470292	0.341073	-0.070485	-0.881573
2	-0.135616	-1.079947	0.428723	1.478222
3	-0.802954	0.712624	0.177219	0.461322
4	-0.302450	-0.940926	0.938401	-0.515991
5	1.699565	-1.169532	-1.853328	-1.664425

In [32]:

```
## get users' features and movies' features
data = ratings[['movieId', 'userId', 'rating']].copy()
data = data.merge(u.reset_index(), on='userId', how='right')
data = data.merge(m.reset_index(), on='movieId', how='right')
data.head()
```

Out[32]:

	movieId	userId	rating	age	time_spent_per_day	u_avg_rating	hour	Action	Adventure	Animation	...	Film-Noir	Horror	IMA
0	1	2	5.0	-0.135616	-1.079947	0.428723	1.478222	0	1	1	...	0	0	
1	1	5	4.0	1.699565	-1.169532	-1.853328	-1.664425	0	1	1	...	0	0	
2	1	8	5.0	0.364888	0.298545	0.163306	1.324821	0	1	1	...	0	0	
3	1	11	4.0	-1.303458	0.513712	-0.377008	0.557816	0	1	1	...	0	0	
4	1	14	4.0	-0.302450	1.251552	-0.375823	0.557816	0	1	1	...	0	0	

5 rows × 26 columns

In [33]:

```
data.drop(columns=['movieId','userId'],inplace=True)
data.head()
```

Out[33]:

	rating	age	time_spent_per_day	u_avg_rating	hour	Action	Adventure	Animation	Children	Comedy	...	Film-Noir	Horror	II
0	5.0	-0.135616	-1.079947	0.428723	1.478222	0	1	1	1	1	...	0	0	
1	4.0	1.699565	-1.169532	-1.853328	-1.664425	0	1	1	1	1	...	0	0	
2	5.0	0.364888	0.298545	0.163306	1.324821	0	1	1	1	1	...	0	0	
3	4.0	-1.303458	0.513712	-0.377008	0.557816	0	1	1	1	1	...	0	0	
4	4.0	-0.302450	1.251552	-0.375823	0.557816	0	1	1	1	1	...	0	0	

5 rows × 24 columns

In [34]:

```
y=data.rating
X=data.drop(columns=['rating'])
X.head()
```

Out[34]:

	age	time_spent_per_day	u_avg_rating	hour	Action	Adventure	Animation	Children	Comedy	Crime	...	Film-Noir	Horror	II
0	-0.135616	-1.079947	0.428723	1.478222	0	1	1	1	1	0	...	0	0	
1	1.699565	-1.169532	-1.853328	-1.664425	0	1	1	1	1	0	...	0	0	
2	0.364888	0.298545	0.163306	1.324821	0	1	1	1	1	0	...	0	0	
3	-1.303458	0.513712	-0.377008	0.557816	0	1	1	1	1	0	...	0	0	
4	-0.302450	1.251552	-0.375823	0.557816	0	1	1	1	1	0	...	0	0	

5 rows × 23 columns

In [35]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=0)
```

In [36]:
```
from sklearn.ensemble import GradientBoostingRegressor

model=GradientBoostingRegressor()
model.fit(X_train,y_train)
y_pred=model.predict(X_test)
```

In [37]:
```
y_test.iloc[4]
```

Out[37]:
```
np.float64(5.0)
```

In [38]:
```
y_pred[4]
```

Out[38]:
```
np.float64(3.9089975309481337)
```

In [39]:
```
from sklearn.metrics import mean_squared_error as mse
```

In [40]:
```
# on an average, our predictions are deviating by 0.89 units.
mse(y_test,y_pred)**0.5
```

Out[40]:
```
0.8915242712591698
```

Recommendation

Basis on the predicted rating a default can be set if rating is greater than threshold then recommend else not

In [41]:
```
u3m1 = pd.concat((u.loc[3], m.loc[1]))
u3m1
```

Out[41]:
```
age                 -0.802954
time_spent_per_day   0.712624
u_avg_rating         0.177219
hour                 0.461322
Action               0.000000
Adventure            1.000000
Animation            1.000000
Children             1.000000
Comedy               1.000000
Crime                0.000000
Documentary          0.000000
Drama                0.000000
Fantasy              1.000000
Film-Noir            0.000000
Horror               0.000000
IMAX                 0.000000
Musical              0.000000
Mystery              0.000000
Romance              0.000000
Sci-Fi               0.000000
Thriller             0.000000
War                  0.000000
Western              0.000000
dtype: float64
```

In [42]:
```
# should u3 watch m1 or not ?

model.predict(u3m1.values.reshape(1, -1))
```

```
C:\Users\kanwar\anaconda3\Lib\site-packages\sklearn\utils\validation.py:2739: UserWarning: X does not have valid feature names,
but GradientBoostingRegressor was fitted with feature names
  warnings.warn(
```

Out[42]:
```
array([3.81314863])
```

In [43]:
```
ratings.head()
```

Out[43]:

	userId	movieId	rating	timestamp
0	1	16	4.0	1217897793
1	1	24	1.5	1217895807
2	1	32	4.0	1217896246
3	1	47	4.0	1217896556
4	1	50	4.0	1217896523

In [44]:

```
ratings.shape
```

Out[44]:

```
(63250, 4)
```

Build interaction matrix:

In [13]:

```
rm=ratings.pivot(index='userId',columns='movieId',values='rating')
rm.head(100)
```

Out[13]:

movieId userId	1	2	3	4	5	6	7	8	9	10	...	144482	144656	144976	146344	146656	146684	146878	14
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	5.0	NaN	2.0	NaN	3.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	3.0	NaN	3.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
96	5.0	NaN	4.0	NaN	3.0	4.0	3.0	NaN	1.0	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
97	3.5	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
98	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
99	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
100	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

100 rows × 10325 columns

In [14]:
```
rm.shape
```

Out[14]:
```
(668, 10325)
```

In [15]:
```
(rm>0).sum().sum() ### number of combinations >0
```

Out[15]:
```
np.int64(105339)
```

In [57]:
```
(rm>0).sum().sum() /( rm.shape[0]*rm.shape[1])
```

Out[57]:
```
np.float64(0.09468562874251497)
```

In [16]:
```
## 668 users and 1000 movies
## to implement the code for matrix factorization lets take a part of this data

rm_small=rm.copy()
rm_small=rm_small[rm_small.columns[:100]]
rm_small=rm_small.head(100)
```

In [17]:
```
rm_small.shape
```

Out[17]:
```
(100, 100)
```

In [18]:
```
rm_small=rm_small.fillna(0)
```

In [19]:
```
rm_small
```

Out[19]:

movieId	1	2	3	4	5	6	7	8	9	10	...	100	101	102	103	104	105	107	108	110	111
userId																					
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	4.0	0.0
2	5.0	0.0	2.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	3.0	0.0	3.0	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
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0
5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
96	5.0	0.0	4.0	0.0	3.0	4.0	3.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	4.0	0.0	3.0	0.0	0.0	0.0
97	3.5	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
98	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0
99	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
100	3.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

100 rows × 100 columns

In [50]:

```
K = 2 # d: embeddings
P = np.random.normal(size=(rm_small.shape[0], K))
Q = np.random.normal(size=(rm_small.shape[1], K))


def matrix_factorization(R, P, Q, K, steps=100, alpha=0.01):
    Q = Q.T
    # P : (n,k)
    # Q : (k,m)
    for step in range(steps):

        for i in range(len(R)): # n
            for j in range(len(R[i])): # m
                if R[i][j] == 0:
```

```
                continue # skip to next rating

            eij = np.dot(P[i,:],Q[:,j]) - R[i][j]

            for k in range(K):
                P[i][k] = P[i][k] - alpha * (2 * eij * Q[k][j])
                Q[k][j] = Q[k][j] - alpha * (2 * eij * P[i][k])
    return P, Q.T
```

In [51]:
```
P_, Q_ =  matrix_factorization(rm_small.values.copy(), P.copy(), Q.copy(), 2)
```

In [22]:
```
P_.shape
```

Out[22]: (100, 2)

In [23]:
```
Q_.shape
```

Out[23]: (100, 2)

In [57]:
```
rm_ = np.dot(P_, Q_.T)
rm_.shape
```

Out[57]: (100, 100)

In [53]:
```
# u3, m17
np.dot(P_[3], Q_[17])
```

Out[53]: np.float64(2.8845625303390596)

In [54]:
```
rm_small.values[3, 17]
```

Out[54]: np.float64(0.0)

In [55]:
```
np.dot(P_[4], Q_[36]) ## predicted value
```

Out[55]: np.float64(3.610811089065432)

In [56]:
```
rm_small.values[4,36]
```

Out[56]: np.float64(0.0)

In [58]:
```
from sklearn.metrics import mean_squared_error as mse

# final evaluation- RMSE
mse( rm_small.values[rm_small>0] , rm_[rm_small>0] )**0.5
```

Out[58]: 0.6658166408682523

on an average the predict ratings are 0.6 units away from acutal

In [ ]: