Problem statement : In this business case we need to create a Recommender System to show personalized movie recommendations based on ratings given by a user and other users similar to them in order to improve user experience. Byy making use of the Ratings, Users and Movies dataset we need to build a recommender system which can recommend movies based on the various charateristics of users like moving rating by them, age, gender etc. Also, item to item(in our case movies) will also be used to find the right movies to recommend to the users.

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:
```
ratings = pd.read_csv('C:/DSML/Case study/Zee Recommender systems/zee-ratings.
```

In [3]:
```
users = pd.read_csv('C:/DSML/Case study/Zee Recommender systems/zee-users.csv'
```

In [4]:
```
movies = pd.read_csv('C:/DSML/Case study/Zee Recommender systems/zee-movies.cs
```

Define Problem Statement and Formatting the Data

Formatting the data files to bring them into a workable format

In [5]:
```
ratings.rename(columns = {'UserID::MovieID::Rating::Timestamp':'text'}, inplac
ratings[['UserID','MovieID','Rating','Timestamp']] = ratings.text.str.split(":
ratings.drop(['text'],axis=1,inplace=True)
```

In [6]:
```
ratings
```

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
...	...	...	...	...
1000204	6040	1091	1	956716541
1000205	6040	1094	5	956704887
1000206	6040	562	5	956704746
1000207	6040	1096	4	956715648
1000208	6040	1097	4	956715569

1000209 rows × 4 columns

In [7]:
```
users.rename(columns = {'UserID::Gender::Age::Occupation::Zip-code':'text'}, i
users[['UserID','Gender','Age','Occupation','Zip-code']] = users.text.str.spli
users.drop(['text'],axis=1,inplace=True)
```

In [8]:
```
users
```

Out[8]:

	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
...	...	...	...	...	...
6035	6036	F	25	15	32603
6036	6037	F	45	1	76006
6037	6038	F	56	1	14706
6038	6039	F	45	0	01060
6039	6040	M	25	6	11106

6040 rows × 5 columns

In [9]:
```
movies.rename(columns = {'MovieID::Title::Genres':'text'}, inplace = True)
movies["text1"].fillna("", inplace = True)
movies["text2"].fillna("", inplace = True)
movies['text3'] = movies['Movie ID::Title::Genres'] + '' + movies['text1'] + '
movies.drop(['Movie ID::Title::Genres','text1','text2'],axis=1,inplace=True)
movies[['MovieID','Title','Genres']] = movies.text3.str.split("::",expand=True
movies.drop(['text3'],axis=1,inplace=True)
```

In [10]:
```
movies
```

Out[10]:

	MovieID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
3878	3948	Meet the Parents (2000)	Comedy
3879	3949	Requiem for a Dream (2000)	Drama
3880	3950	Tigerland (2000)	Drama
3881	3951	Two Family House (2000)	Drama
3882	3952	Contender The (2000)	Drama|Thriller

3883 rows × 3 columns

Merging the data files and creating a single consolidated dataframe

In [11]:
```
ratings_movies = pd.merge(ratings, movies, on = "MovieID", how = "inner")
```

In [12]:
```
ratings_movies
```

Out[12]:

	UserID	MovieID	Rating	Timestamp	Title	Genres
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama
1	2	1193	5	978298413	One Flew Over the Cuckoo's Nest (1975)	Drama
2	12	1193	4	978220179	One Flew Over the Cuckoo's Nest (1975)	Drama
3	15	1193	4	978199279	One Flew Over the Cuckoo's Nest (1975)	Drama
4	17	1193	5	978158471	One Flew Over the Cuckoo's Nest (1975)	Drama
...	...	...	...	...	...	...
1000204	5949	2198	5	958846401	Modulations (1998)	Documentary
1000205	5675	2703	3	976029116	Broken Vessels (1998)	Drama
1000206	5780	2845	1	958153068	White Boys (1999)	Drama
1000207	5851	3607	5	957756608	One Little Indian (1973)	Comedy|Drama|Western
1000208	5938	2909	4	957273353	Five Wives Three Secretaries and Me (1998)	Documentary

1000209 rows × 6 columns

In [13]:
```
ratings_movies_users = pd.merge(ratings_movies, users, on = "UserID", how = "i
```

In [14]:
```
ratings_movies_users
```

Out[14]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama	F
1	1	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical	F
2	1	914	3	978301968	My Fair Lady (1964)	Musical|Romance	F
3	1	3408	4	978300275	Erin Brockovich (2000)	Drama	F
4	1	2355	5	978824291	Bug's Life A (1998)	Animation|Children's|Comedy	F
...	...	...	...	...	...	...	...
1000204	4211	3791	2	965319075	Footloose (1984)	Drama	M
1000205	4211	3806	3	965319138	MacKenna's Gold (1969)	Western	M
1000206	4211	3840	4	965319197	Pumpkinhead (1988)	Horror	M
1000207	4211	3766	2	965319138	Missing in Action (1984)	Action|War	M
1000208	4211	3834	2	965318885	Bronco Billy (1980)	Adventure|Drama|Romance	M

1000209 rows × 10 columns

Performing EDA, Data Cleaning, and Feature Engineering

Reviewing the shape and structure of the dataset

In [15]:
```
ratings_movies_users.shape
```

Out[15]:
```
(1000209, 10)
```

In [16]:
```
ratings_movies_users.columns
```

Out[16]:
```
Index(['UserID', 'MovieID', 'Rating', 'Timestamp', 'Title', 'Genres', 'Gende
r',
       'Age', 'Occupation', 'Zip-code'],
      dtype='object')
```

In [17]:
```
ratings_movies_users.dtypes
```

Out[17]:
```
UserID        object
MovieID       object
Rating        object
Timestamp     object
Title         object
Genres        object
Gender        object
Age           object
Occupation    object
Zip-code      object
dtype: object
```

Performing necessary type conversion and deriving new features

In [18]:
```
ratings_movies_users['UserID'] = ratings_movies_users['UserID'].astype('str')
ratings_movies_users['MovieID'] = ratings_movies_users['MovieID'].astype('str'
ratings_movies_users['Rating'] = ratings_movies_users['Rating'].astype('int')
ratings_movies_users['Title'] = ratings_movies_users['Title'].astype('str')
ratings_movies_users['Genres'] = ratings_movies_users['Genres'].astype('str')
ratings_movies_users['Age'] = ratings_movies_users['Age'].astype('int')
```

```
Splitting the Title column to derive 'Release year'
```

In [19]:
```
ratings_movies_users['Release year'] = ratings_movies_users['Title'].str[-5:-1
```

In [20]:
```
ratings_movies_users
```

Out[20]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama	F
1	1	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical	F
2	1	914	3	978301968	My Fair Lady (1964)	Musical|Romance	F
3	1	3408	4	978300275	Erin Brockovich (2000)	Drama	F
4	1	2355	5	978824291	Bug's Life A (1998)	Animation|Children's|Comedy	F
...	...	...	...	...	...	...	...
1000204	4211	3791	2	965319075	Footloose (1984)	Drama	M
1000205	4211	3806	3	965319138	MacKenna's Gold (1969)	Western	M
1000206	4211	3840	4	965319197	Pumpkinhead (1988)	Horror	M
1000207	4211	3766	2	965319138	Missing in Action (1984)	Action|War	M
1000208	4211	3834	2	965318885	Bronco Billy (1980)	Adventure|Drama|Romance	M

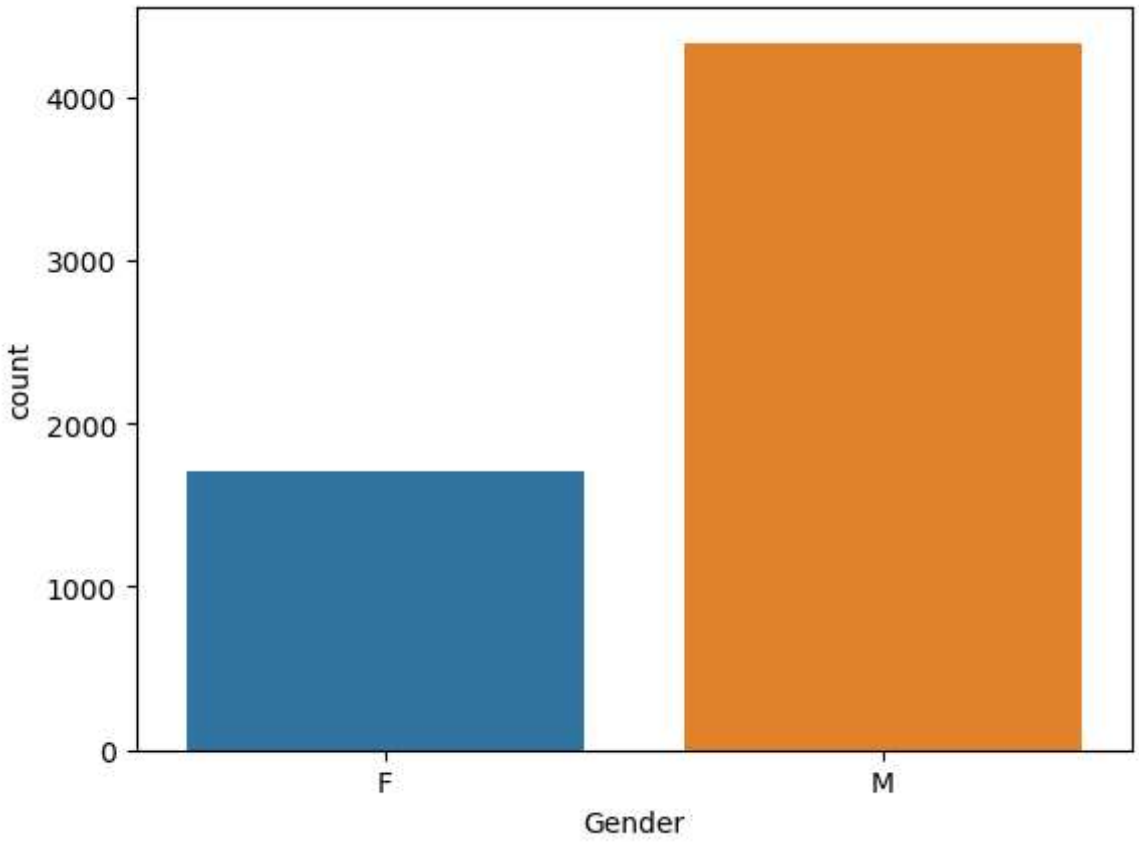
1000209 rows × 11 columns

Visualizing the data with respect to different categories to get a better understanding of the underlying distribution

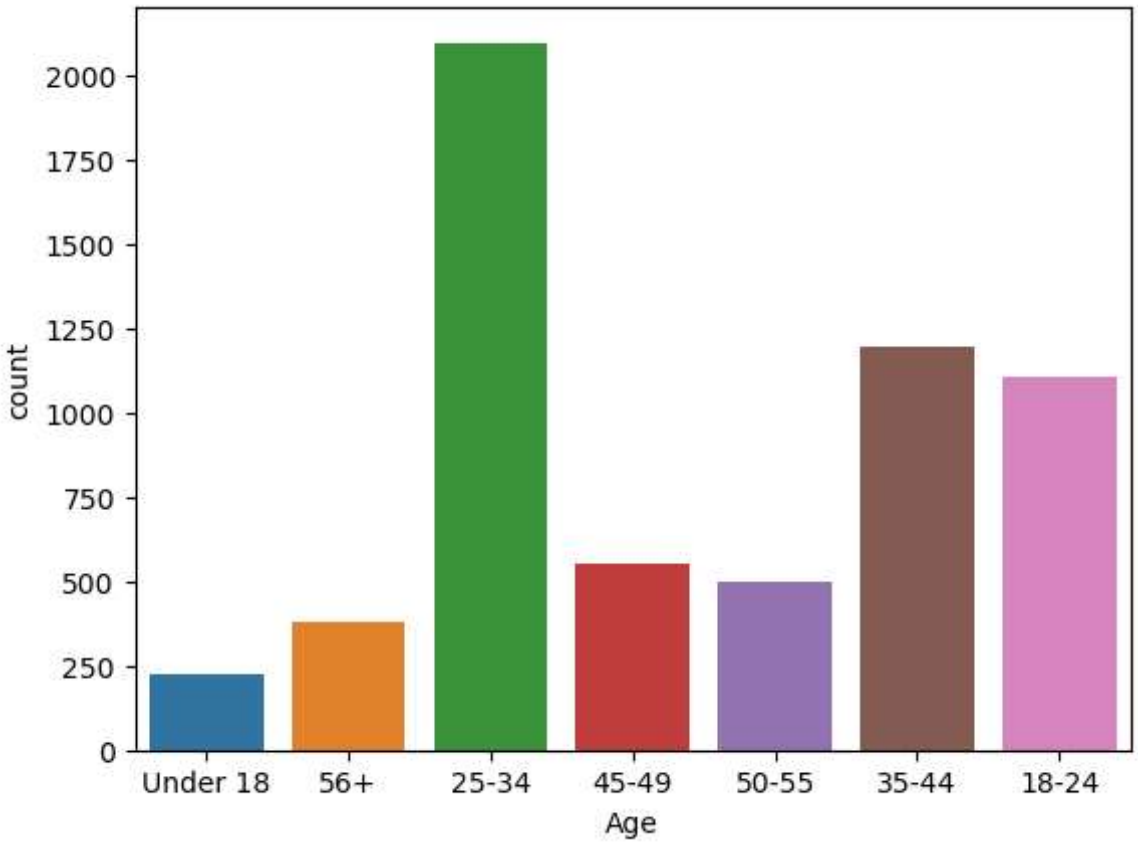
In [21]:
```
users['Age'] = users['Age'].astype('int')
conditions = [
    (users['Age'] == 1),
    (users['Age'] == 18),
    (users['Age'] == 25),
    (users['Age'] == 35),
    (users['Age'] == 45),
    (users['Age'] == 50),
    (users['Age'] == 56)
]

values = ['Under 18', '18-24', '25-34', '35-44', '45-49', '50-55', '56+']

users['Age'] = np.select(conditions, values)
```

In [22]:
```
users['Occupation'] = users['Occupation'].astype('int')
conditions = [
    (users['Occupation'] == 0),
    (users['Occupation'] == 1),
    (users['Occupation'] == 2),
    (users['Occupation'] == 3),
    (users['Occupation'] == 4),
    (users['Occupation'] == 5),
    (users['Occupation'] == 6),
    (users['Occupation'] == 7),
    (users['Occupation'] == 8),
    (users['Occupation'] == 9),
    (users['Occupation'] == 10),
    (users['Occupation'] == 11),
    (users['Occupation'] == 12),
    (users['Occupation'] == 13),
    (users['Occupation'] == 14),
    (users['Occupation'] == 15),
    (users['Occupation'] == 16),
    (users['Occupation'] == 17),
    (users['Occupation'] == 18),
    (users['Occupation'] == 19),
    (users['Occupation'] == 20)
]

values = ['other', 'academic/educator', 'artist', 'clerical/admin', 'college/g

users['Occupation'] = np.select(conditions, values)
```

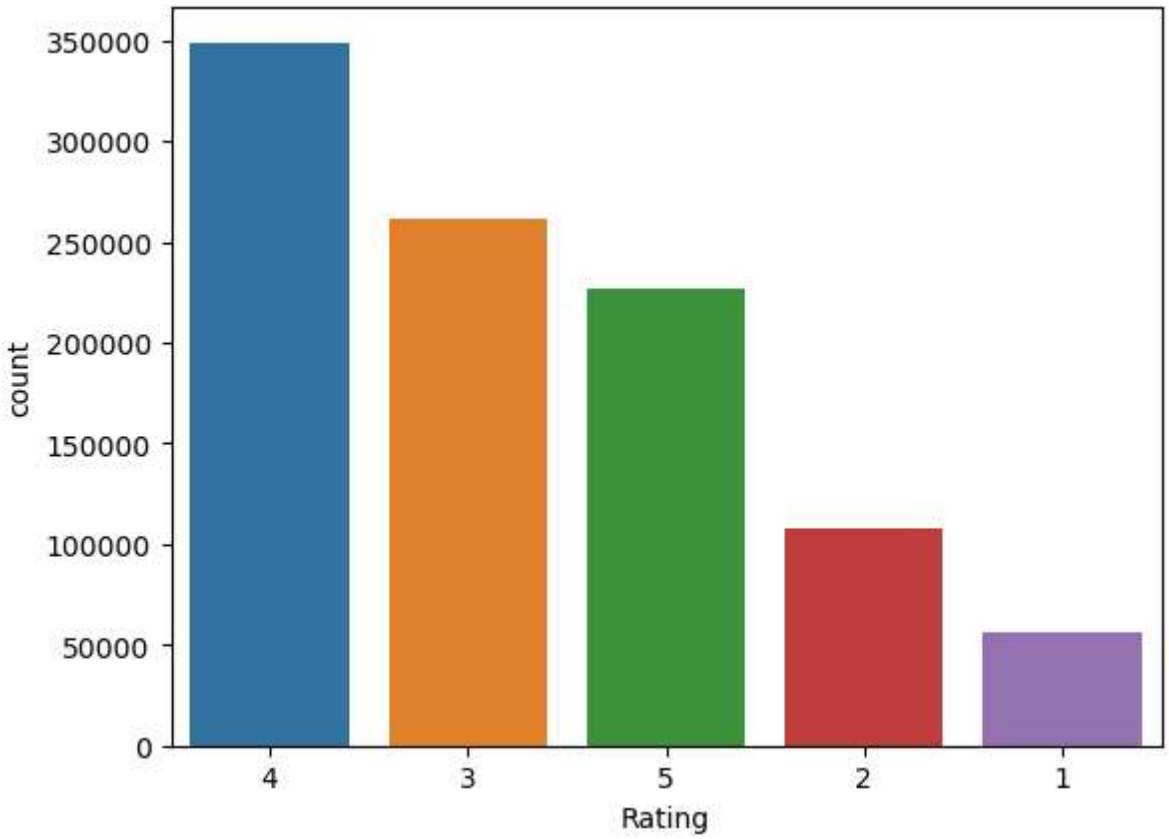
In [23]:
```
sns.countplot(x = 'Gender',data = users)
plt.show()
```



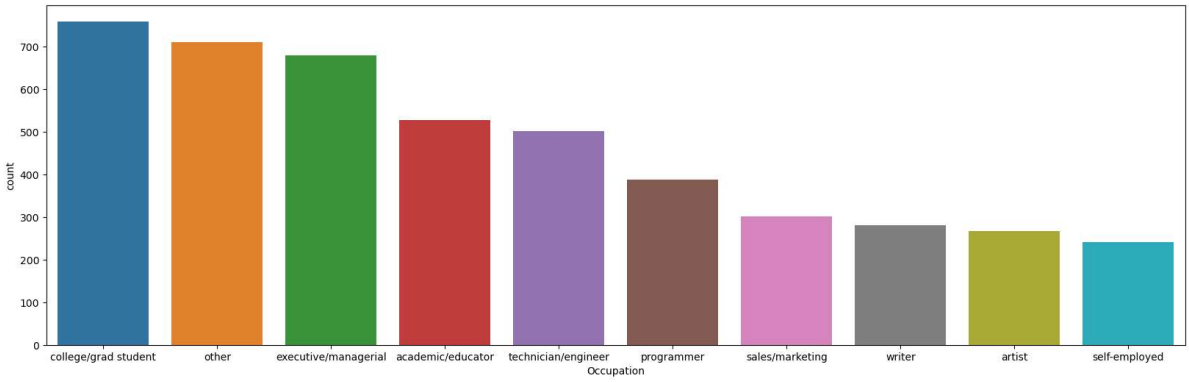
In [24]:
```
sns.countplot(x = 'Age',data = users)
plt.show()
```



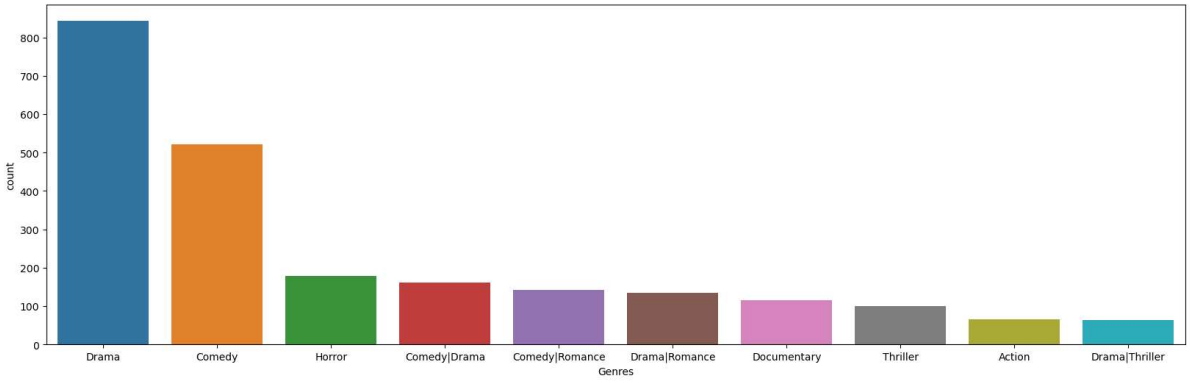
In [25]:
```
sns.countplot(x = 'Rating', data=ratings, order=ratings.value_counts(ratings['
plt.show()
```



In [26]:
```
plt.figure(figsize=(20,6))
sns.countplot(x = 'Occupation', data=users, order=users.value_counts(users['Oc
plt.show()
```



In [27]:
```
plt.figure(figsize=(20,6))
sns.countplot(x = 'Genres', data=movies, order=movies.value_counts(movies['Gen
plt.show()
```



Investigating the data for any inconsistency

In [28]:
```
ratings_movies_users.isna().sum()
```

Out[28]:
```
UserID          0
MovieID         0
Rating          0
Timestamp       0
Title           0
Genres          0
Gender          0
Age             0
Occupation      0
Zip-code        0
Release year    0
dtype: int64
```

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

```
First, let us find the average ratings and no. of ratings per movie
```

In [29]:
```
movie_avg_ratings = pd.DataFrame(ratings_movies_users.groupby(['MovieID','Titl
movie_avg_ratings.reset_index(inplace=True)
movie_avg_ratings
```

Out[29]:

	MovieID	Title	Rating
0	3233	Smashing Time (1967)	5.0
1	3656	Lured (1947)	5.0
2	3382	Song of Freedom (1936)	5.0
3	3607	One Little Indian (1973)	5.0
4	1830	Follow the Bitch (1998)	5.0
...	...	...	...
3701	3460	Hillbillys in a Haunted House (1967)	1.0
3702	3228	Wirey Spindell (1999)	1.0
3703	3493	Torso (Corpi Presentano Tracce di Violenza Car...	1.0
3704	3123	Spring Fever USA (a.k.a. Lauderdale) (1989)	1.0
3705	1430	Underworld (1997)	1.0

3706 rows × 3 columns

In [30]:
```
movie_no_of_ratings = pd.DataFrame(ratings_movies_users.groupby(['MovieID','Ti
movie_no_of_ratings.reset_index(inplace=True)
movie_no_of_ratings
```

Out[30]:

	MovieID	Title	Rating
0	2858	American Beauty (1999)	3428
1	260	Star Wars: Episode IV - A New Hope (1977)	2991
2	1196	Star Wars: Episode V - The Empire Strikes Back...	2990
3	1210	Star Wars: Episode VI - Return of the Jedi (1983)	2883
4	480	Jurassic Park (1993)	2672
...	...	...	...
3701	865	Small Faces (1995)	1
3702	641	Little Indian Big City (Un indien dans la vill...	1
3703	868	Death in Brunswick (1991)	1
3704	1714	Never Met Picasso (1996)	1
3705	1630	Lay of the Land The (1997)	1

3706 rows × 3 columns

First, let us find the average ratings and no. of ratings per user

In [31]:
```
user_avg_ratings = pd.DataFrame(ratings_movies_users.groupby('UserID')['Rating
user_avg_ratings.reset_index(inplace=True)
user_avg_ratings
```

Out[31]:

	UserID	Rating
0	283	4.962963
1	2339	4.956522
2	3324	4.904762
3	3902	4.890909
4	446	4.843137
...	...	...
6035	5850	1.844828
6036	4539	1.815126
6037	2744	1.304348
6038	4486	1.058824
6039	3598	1.015385

6040 rows × 2 columns

In [32]:
```
user_no_of_ratings = pd.DataFrame(ratings_movies_users.groupby('UserID')['Rati
user_no_of_ratings.reset_index(inplace=True)
user_no_of_ratings
```

Out[32]:

	UserID	Rating
0	4169	2314
1	1680	1850
2	4277	1743
3	1941	1595
4	1181	1521
...	...	...
6035	4383	20
6036	4365	20
6037	3633	20
6038	4332	20
6039	3787	20

6040 rows × 2 columns

Build a Recommender System based on Pearson Correlation

Creating a pivot table of movie titles & user id and imputing the NaN values

In [33]:
```
m = ratings_movies_users.copy()
data_table = pd.pivot_table(m,values='Rating',columns='Title',index='UserID')
data_table.fillna(0, inplace=True)
data_table.head()
```

Out[33]:

Title	$10000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	'burbs The (1989)	...And Justice for All (1979)	1-900 (1994)	10 Things I Hate About You (1999)	101 Dalmatians (1961)	101 Dalmatians (1996)	12 Angry Men (1957)
UserID										
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	3.0
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0
1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	0.0

5 rows × 3706 columns

Use the Item-based approach to create a simple recommender system that uses Pearson Correlation

In [34]:
```
def get_item_recommendations(df, item):
    return (df.corr()[item].sort_values(ascending=False))
```

In [35]:
```
movie_name = "10 Things I Hate About You (1999)"
recommendations = get_item_recommendations(data_table, movie_name)
recommendations.head(6)
```

Out[35]:
```
Title
10 Things I Hate About You (1999)    1.000000
She's All That (1999)                0.409002
Never Been Kissed (1999)             0.400310
American Pie (1999)                  0.368416
Cruel Intentions (1999)              0.348174
Big Daddy (1999)                     0.337655
Name: 10 Things I Hate About You (1999), dtype: float64
```

Build a Recommender System based on Cosine Similarity

Print the user similarity matrix and item similarity matrix

```
We have categorical values, many unique values and also non scaled data in
the 'users' data set, we need to perform some preprocessing in order to be
able to create the user-user similarity matrix using cosine similarity
```

In [36]:
```
users['Age'].value_counts()
```

Out[36]:
```
25-34       2096
35-44       1193
18-24       1103
45-49        550
50-55        496
56+          380
Under 18     222
Name: Age, dtype: int64
```

In [37]:
```
users['Occupation'].value_counts()
```

Out[37]:
```
college/grad student     759
other                    711
executive/managerial     679
academic/educator        528
technician/engineer      502
programmer               388
sales/marketing          302
writer                   281
artist                   267
self-employed            241
doctor/health care       236
K-12 student             195
clerical/admin           173
scientist                144
retired                  142
lawyer                   129
customer service         112
homemaker                 92
unemployed                72
tradesman/craftsman       70
farmer                    17
Name: Occupation, dtype: int64
```

In [38]:
```
users = users.merge(user_avg_ratings, on='UserID')
```

In [39]:
```
users.rename(columns = {'Rating':'rating_mean'}, inplace = True)
```

In [40]:
```
users = users.merge(user_no_of_ratings, on='UserID')
```

In [41]:
```
users.rename(columns = {'Rating':'no_of_ratings'}, inplace = True)
```

In [42]:
```
users.drop(['Zip-code'], axis=1, inplace=True)
```

In [43]:
```
users
```

Out[43]:

	UserID	Gender	Age	Occupation	rating_mean	no_of_ratings
0	1	F	Under 18	K-12 student	4.188679	53
1	2	M	56+	self-employed	3.713178	129
2	3	M	25-34	scientist	3.901961	51
3	4	M	45-49	executive/managerial	4.190476	21
4	5	M	25-34	writer	3.146465	198
...	...	...	...	...	...	...
6035	6036	F	25-34	scientist	3.302928	888
6036	6037	F	45-49	academic/educator	3.717822	202
6037	6038	F	56+	academic/educator	3.800000	20
6038	6039	F	45-49	other	3.878049	123
6039	6040	M	25-34	doctor/health care	3.577713	341

6040 rows × 6 columns

In [44]:
```
users_encoded = pd.get_dummies(users, columns=['Gender', 'Age', 'Occupation'])
```

In [45]:
```
users_encoded
```

Out[45]:

	UserID	rating_mean	no_of_ratings	Gender_F	Gender_M	Age_18-24	Age_25-34	Age_35-44	Ag
0	1	4.188679	53	1	0	0	0	0	
1	2	3.713178	129	0	1	0	0	0	
2	3	3.901961	51	0	1	0	1	0	
3	4	4.190476	21	0	1	0	0	0	
4	5	3.146465	198	0	1	0	1	0	
...	...	...	...	...	...	...	...	...	
6035	6036	3.302928	888	1	0	0	1	0	
6036	6037	3.717822	202	1	0	0	0	0	
6037	6038	3.800000	20	1	0	0	0	0	
6038	6039	3.878049	123	1	0	0	0	0	
6039	6040	3.577713	341	0	1	0	1	0	

6040 rows × 33 columns

In [46]:
```
from sklearn.preprocessing import StandardScaler

features = users_encoded.columns
scaler = StandardScaler()
users_encoded_scaled = scaler.fit_transform(users_encoded)
users_encoded_scaled = pd.DataFrame(users_encoded_scaled)
users_encoded_scaled.head()
```

Out[46]:

	0	1	2	3	4	5	6	7	8
0	-1.731764	1.131261	-0.584221	1.591927	-1.591927	-0.472668	-0.728999	-0.496117	-0.31651
1	-1.731191	0.024380	-0.189889	-0.628170	0.628170	-0.472668	-0.728999	-0.496117	-0.31651
2	-1.730617	0.463832	-0.594598	-0.628170	0.628170	-0.472668	1.371743	-0.496117	-0.31651
3	-1.730043	1.135444	-0.750255	-0.628170	0.628170	-0.472668	-0.728999	-0.496117	3.15940
4	-1.729470	-1.294827	0.168123	-0.628170	0.628170	-0.472668	1.371743	-0.496117	-0.31651

5 rows × 33 columns

In [47]:
```
users_encoded_scaled.columns = users_encoded.columns
```

In [48]:
```
users_encoded_scaled
```

Out[48]:

	UserID	rating_mean	no_of_ratings	Gender_F	Gender_M	Age_18-24	Age_25-34	Age_35-4
0	-1.731764	1.131261	-0.584221	1.591927	-1.591927	-0.472668	-0.728999	-0.49611
1	-1.731191	0.024380	-0.189889	-0.628170	0.628170	-0.472668	-0.728999	-0.49611
2	-1.730617	0.463832	-0.594598	-0.628170	0.628170	-0.472668	1.371743	-0.49611
3	-1.730043	1.135444	-0.750255	-0.628170	0.628170	-0.472668	-0.728999	-0.49611
4	-1.729470	-1.294827	0.168123	-0.628170	0.628170	-0.472668	1.371743	-0.49611
...	...	...	...	...	...	...	...	...
6035	1.729470	-0.930609	3.748241	1.591927	-1.591927	-0.472668	1.371743	-0.49611
6036	1.730043	0.035189	0.188877	1.591927	-1.591927	-0.472668	-0.728999	-0.49611
6037	1.730617	0.226486	-0.755444	1.591927	-1.591927	-0.472668	-0.728999	-0.49611
6038	1.731191	0.408169	-0.221020	1.591927	-1.591927	-0.472668	-0.728999	-0.49611
6039	1.731764	-0.290959	0.910089	-0.628170	0.628170	-0.472668	1.371743	-0.49611

6040 rows × 33 columns

User similarity matrix using cosine similarity

calculating cosine distance between users

In [49]:
```
from scipy.spatial import distance

def cosine_similarity(x,y):
    return distance.cosine(x,y)
```

In [50]:
```
user_cosine_ranks = []

# We will be using only first 100 users against all the users since there are
# computationally expensive
for query in users_encoded_scaled.index[:100]:
    for candidate in users_encoded_scaled.index:
        if candidate == query:
            continue
        user_cosine_ranks.append([query, candidate, cosine_similarity(users_en
```

In [51]:
```
users_cosine = pd.DataFrame(user_cosine_ranks, columns=['User1', 'User2', 'cos
```

In [52]:
```
users_cosine_pivot = pd.pivot_table(users_cosine, index='User1', columns='User
users_cosine_pivot.fillna(0, inplace=True)
```

In [53]:
```
users_cosine_pivot
```

Out[53]:

User2	0	1	2	3	4	5	6	7	8
User1									
0	0.000000	1.011996	1.006564	0.980891	1.063175	0.907933	0.974713	1.019029	1.029692
1	1.011996	0.000000	0.965441	0.941248	0.961436	1.011802	0.941534	0.952984	0.948521
2	1.006564	0.965441	0.000000	0.933672	0.875731	1.012026	0.929948	0.827364	0.813833
3	0.980891	0.941248	0.933672	0.000000	1.008819	1.001451	0.845192	0.919760	0.925428
4	1.063175	0.961436	0.875731	1.008819	0.000000	1.039413	1.028818	0.812297	0.777676
...	...	...	...	...	...	...	...	...	...
95	0.884617	0.383554	0.935097	1.047731	0.915377	0.896323	1.057900	0.912094	0.898733
96	0.863390	1.021815	1.013624	0.976244	1.100404	0.897490	0.777916	1.024626	1.047575
97	0.831271	1.033068	1.032998	0.603300	1.081186	0.856203	0.747531	1.067069	1.072021
98	0.038395	1.015759	1.029412	1.049927	0.999019	0.924392	1.065135	1.047377	1.039903
99	1.066017	0.948527	0.978741	0.997421	0.882077	1.037009	0.751842	0.977131	0.316463

100 rows × 6040 columns

Item similarity matrix using cosine similarity

In [54]:
```
movies_split = movies.copy()
movies_split['Genres'] = movies_split['Genres'].str.split('|')
movies_split = movies_split.explode('Genres')
movies_split = movies_split.pivot(index='MovieID', columns='Genres', values='T
movies_split = ~movies_split.isna()
movies_split = movies_split.astype(int)
movies_split
```

Out[54]:

Genres	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fanta
MovieID									
1	0	0	1	1	1	0	0	0	
10	1	1	0	0	0	0	0	0	
100	0	0	0	0	0	0	0	1	
1000	0	0	0	0	0	1	0	0	
1001	0	0	0	0	1	0	0	0	
...	...	...	...	...	...	...	...	...	
994	0	0	0	0	0	0	0	1	
996	1	0	0	0	0	0	0	1	
997	0	0	0	0	0	0	0	1	
998	1	0	0	0	0	1	0	0	
999	0	0	0	0	0	1	0	0	

3883 rows × 18 columns

In [55]:
```
movie_cosine_ranks = []

# We will be using only first 100 movies against all the users since there are
# computationally expensive
for query in movies_split.index[:100]:
    for candidate in movies_split.index:
        if candidate == query:
            continue
        movie_cosine_ranks.append([query, candidate, cosine_similarity(movies_
```

In [56]:
```
movies_cosine = pd.DataFrame(movie_cosine_ranks, columns=['Movie1', 'Movie2',
```

In [57]:
```
movies_cosine_pivot = pd.pivot_table(movies_cosine, index='Movie1', columns='M
movies_cosine_pivot.fillna(0, inplace=True)
```

In [58]:
```
movies_cosine_pivot
```

Out[58]:

Movie2	1	10	100	1000	1001	1002	1003	1004	1005	
Movie1										
1	0.00000	1.000000	1.000000	1.000000	0.42265	0.42265	1.000000	1.000000	0.183503	1
10	1.00000	0.000000	0.591752	1.000000	1.00000	1.00000	0.591752	0.183503	1.000000	1
100	1.00000	0.591752	0.000000	1.000000	1.00000	1.00000	0.000000	0.500000	1.000000	0
1000	1.00000	1.000000	1.000000	0.000000	1.00000	1.00000	1.000000	1.000000	1.000000	1
1001	0.42265	1.000000	1.000000	1.000000	0.00000	0.00000	1.000000	1.000000	0.292893	1
...	...	...	...	...	...	...	...	...	...	
1087	1.00000	1.000000	1.000000	1.000000	1.00000	1.00000	1.000000	1.000000	1.000000	1
1088	1.00000	1.000000	1.000000	1.000000	1.00000	1.00000	1.000000	1.000000	1.000000	1
1089	1.00000	0.591752	0.500000	0.292893	1.00000	1.00000	0.500000	0.500000	1.000000	1
109	0.42265	1.000000	1.000000	1.000000	0.00000	0.00000	1.000000	1.000000	0.292893	1
1090	1.00000	1.000000	0.500000	1.000000	1.00000	1.00000	0.500000	1.000000	1.000000	0

100 rows × 3883 columns

Use the Item-based approach to create a recommender system that uses Nearest Neighbors algorithm and Cosine Similarity

In [59]:
```
movies_split.to_numpy()
```

Out[59]:
```
array([[0, 0, 1, ..., 0, 0, 0],
       [1, 1, 0, ..., 1, 0, 0],
       [0, 0, 0, ..., 1, 0, 0],
       ...,
       [0, 0, 0, ..., 1, 0, 0],
       [1, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])
```

In [60]:
```
from sklearn.neighbors import NearestNeighbors
nbrs = NearestNeighbors(n_neighbors=5, metric='cosine').fit(movies_split)
distances, indices = nbrs.kneighbors(movies_split)
```

In [61]:
```
nn_movies = pd.DataFrame(list(zip(movies_split.index.to_list(), indices, dista
```

In [63]:
```
nn_movies = nn_movies.merge(movies, on='MovieID')
```

In [64]:
```
nn_movies
```

Out[64]:

	MovieID	Neighbours	Distances	Title	Genres
0	1	[3000, 1450, 1215, 72, 2845]	[0.0, 0.0, 0.0, 0.0, 0.0]	Toy Story (1995)	Animation|Children's|Comedy
1	10	[3004, 3874, 589, 3705, 3663]	[0.0, 0.0, 0.0, 0.0, 0.0]	GoldenEye (1995)	Action|Adventure|Thriller
2	100	[3480, 182, 1944, 3756, 2129]	[2.220446049250313e-16, 2.220446049250313e-16,...	City Hall (1996)	Drama|Thriller
3	1000	[343, 3574, 2575, 540, 788]	[0.0, 0.0, 0.0, 0.0, 0.0]	Curdled (1996)	Crime
4	1001	[2749, 2755, 1707, 2744, 1704]	[0.0, 0.0, 0.0, 0.0, 0.0]	Associate The (L'Associe) (1982)	Comedy
...	...	...	...	...	...
3878	994	[1268, 3119, 1267, 3127, 1270]	[0.0, 0.0, 0.0, 0.0, 0.0]	Big Night (1996)	Drama
3879	996	[3879, 418, 3132, 3347, 2646]	[0.0, 0.18350341907227397, 0.18350341907227397...	Last Man Standing (1996)	Action|Drama|Western
3880	997	[3480, 182, 1944, 3756, 2129]	[2.220446049250313e-16, 2.220446049250313e-16,...	Caught (1996)	Drama|Thriller
3881	998	[2460, 1398, 3034, 1988, 1345]	[2.220446049250313e-16, 2.220446049250313e-16,...	Set It Off (1996)	Action|Crime
3882	999	[343, 3574, 2575, 540, 788]	[0.0, 0.0, 0.0, 0.0, 0.0]	2 Days in the Valley (1996)	Crime

3883 rows × 5 columns

Build a Recommender System based on Matrix Factorization

Create a Recommender System using the Matrix Factorization method

In [65]:
```
ratings_mf = ratings.copy()
ratings_mf = pd.pivot_table(ratings_mf,values='Rating',columns='MovieID',index
ratings_mf.fillna(0, inplace=True)
```

In [66]:
```
ratings_mf.astype(int)
```

Out[66]:

MovieID	1	10	100	1000	1002	1003	1004	1005	1006	1007	...	99	990	991	992	993	99
UserID																	
1	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
10	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
100	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1000	5	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
1001	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
995	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
996	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
997	4	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
998	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	
999	0	0	0	0	0	2	3	0	0	0	...	0	0	0	0	0	

6040 rows × 3706 columns

In [67]:
```
from surprise import SVD, Dataset, Reader
from surprise.model_selection import cross_validate
```

In [68]:
```
reader = Reader(rating_scale=(1 , 5))
data = Dataset.load_from_df(ratings[['UserID', 'MovieID', 'Rating']], reader)
```

Evaluate the model in terms of the Root Mean Squared Error and Mean Absolute Percentage Error

In [69]:
```
svd = SVD(n_factors=4)
cross_validate(svd, data, measures=['rmse','mae'], cv = 3 , return_train_measu
```

```
Evaluating RMSE, MAE of algorithm SVD on 3 split(s).

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.8914  0.8902  0.8894  0.8903  0.0008
MAE (testset)     0.7040  0.7026  0.7010  0.7025  0.0013
RMSE (trainset)   0.8614  0.8626  0.8596  0.8612  0.0012
MAE (trainset)    0.6809  0.6813  0.6791  0.6804  0.0010
Fit time          4.82    5.00    5.01    4.94    0.08
Test time         4.14    3.68    4.22    4.01    0.24
```

Out[69]:
```
{'test_rmse': array([0.89139624, 0.89022089, 0.88942846]),
 'train_rmse': array([0.86135854, 0.86256779, 0.85955115]),
 'test_mae': array([0.70404845, 0.70262326, 0.70095994]),
 'train_mae': array([0.68086301, 0.68133346, 0.67909811]),
 'fit_time': (4.822999954223633, 4.997015714645386, 5.00758695602417),
 'test_time': (4.140318870544434, 3.684905529022217, 4.219529151916504)}
```

Use embeddings for visualization and similarity-based models

In [70]:
```
users_embedded = svd.pu
movies_embedded = svd.pu
```

User-User similarity

In [71]:
```
def get_cosine_ranks(embed,n, column):
    embed_ranks = []

    for query in range(n):
        for candidate in range(len(embed)):
            if candidate == query:
                continue
            embed_ranks.append([query+1, candidate+1, cosine_similarity(embed[
    return pd.DataFrame(embed_ranks, columns=column)
```

In [72]:
```
user_embed_df = get_cosine_ranks(users_embedded, 100, ['User1', 'User2', 'cosi
user_embed_df
```

Out[72]:

	User1	User2	cosine similarity
0	1	2	1.243223
1	1	3	0.425256
2	1	4	0.776194
3	1	5	1.447671
4	1	6	0.678467
...	...	...	...
603895	100	6036	0.780180
603896	100	6037	0.120388
603897	100	6038	0.973833
603898	100	6039	0.772710
603899	100	6040	1.014679

603900 rows × 3 columns

In [73]:
```
user_embed_cosine_pivot = pd.pivot_table(user_embed_df, index='User1', columns
user_embed_cosine_pivot.fillna(0, inplace=True)
user_embed_cosine_pivot
```

Out[73]:

User2	1	2	3	4	5	6	7	8	9
User1									
1	0.000000	1.243223	0.425256	0.776194	1.447671	0.678467	0.298492	0.446175	1.032254
2	1.243223	0.000000	0.588971	0.165648	1.694415	1.945919	0.708263	1.372365	1.450033
3	0.425256	0.588971	0.000000	0.453705	1.529286	1.175204	0.062642	1.287568	1.746503
4	0.776194	0.165648	0.453705	0.000000	1.911634	1.739960	0.564466	0.940996	1.188237
5	1.447671	1.694415	1.529286	1.911634	0.000000	0.314634	1.564409	1.309969	0.848545
...	...	...	...	...	...	...	...	...	...
96	0.950944	1.074501	0.776784	1.375584	0.855857	0.994787	0.560325	1.338191	1.652194
97	1.051060	1.464711	0.903754	1.431473	0.327072	0.306720	1.114174	1.411589	1.085126
98	1.060364	0.417358	0.642596	0.721350	1.441066	1.664080	0.490152	1.318077	1.681075
99	1.719246	1.103758	1.391312	1.323841	0.322494	0.795167	1.656891	1.535151	0.862269
100	0.414909	1.671436	1.307280	1.326170	1.005044	0.502266	1.046633	0.108674	0.400126

100 rows × 6040 columns

User-User similarity

In [74]:
```
movie_embed_df = get_cosine_ranks(movies_embedded, 100, ['Movie1', 'Movie2', '
movie_embed_df
```

Out[74]:

	Movie1	Movie2	cosine similarity
0	1	2	1.243223
1	1	3	0.425256
2	1	4	0.776194
3	1	5	1.447671
4	1	6	0.678467
...	...	...	...
603895	100	6036	0.780180
603896	100	6037	0.120388
603897	100	6038	0.973833
603898	100	6039	0.772710
603899	100	6040	1.014679

603900 rows × 3 columns

In [75]:
```
movie_embed_cosine_pivot = pd.pivot_table(movie_embed_df, index='Movie1', colu
movie_embed_cosine_pivot.fillna(0, inplace=True)
movie_embed_cosine_pivot
```

Out[75]:

Movie2	1	2	3	4	5	6	7	8	9
Movie1									
1	0.000000	1.243223	0.425256	0.776194	1.447671	0.678467	0.298492	0.446175	1.03225
2	1.243223	0.000000	0.588971	0.165648	1.694415	1.945919	0.708263	1.372365	1.45003
3	0.425256	0.588971	0.000000	0.453705	1.529286	1.175204	0.062642	1.287568	1.74650
4	0.776194	0.165648	0.453705	0.000000	1.911634	1.739960	0.564466	0.940996	1.18823
5	1.447671	1.694415	1.529286	1.911634	0.000000	0.314634	1.564409	1.309969	0.84854
...	...	...	...	...	...	...	...	...	..
96	0.950944	1.074501	0.776784	1.375584	0.855857	0.994787	0.560325	1.338191	1.65219
97	1.051060	1.464711	0.903754	1.431473	0.327072	0.306720	1.114174	1.411589	1.08512
98	1.060364	0.417358	0.642596	0.721350	1.441066	1.664080	0.490152	1.318077	1.68107
99	1.719246	1.103758	1.391312	1.323841	0.322494	0.795167	1.656891	1.535151	0.86226
100	0.414909	1.671436	1.307280	1.326170	1.005044	0.502266	1.046633	0.108674	0.40012

100 rows × 6040 columns

Getting embeddings for d=2 and plotting the results

In [76]:
```
train_set = data.build_full_trainset()
```

In [77]:
```
svd2 = SVD(n_factors = 2)
svd2.fit(train_set)
```

Out[77]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x173b7543940>
```

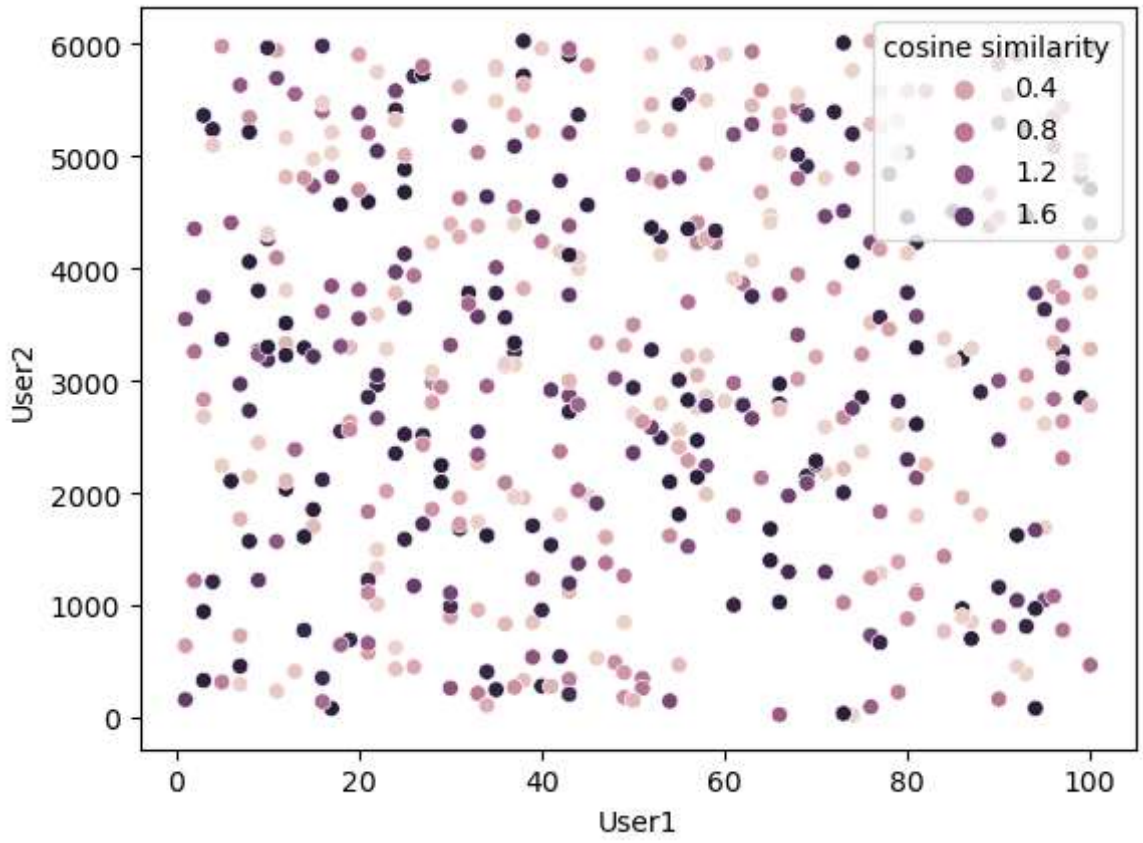
In [78]:
```
user_svd2_ranks = get_cosine_ranks(svd2.pu, 100, ['User1', 'User2', 'cosine si
user_svd2_ranks_sample = user_svd2_ranks.sample(500)
user_svd2_ranks_sample
```

Out[78]:

	User1	User2	cosine similarity
30132	5	5978	0.573356
241478	40	5959	0.083775
196817	33	3571	1.574845
132566	22	5749	0.004166
95982	16	5399	1.004008
...	...	...	...
334970	56	2827	1.981044
121890	21	1112	0.683363
435829	73	1023	0.725991
344008	57	5826	0.152927
52112	9	3802	1.946570

500 rows × 3 columns

In [79]:
```
sns.scatterplot(x = user_svd2_ranks_sample['User1'], y = user_svd2_ranks_sampl
plt.show()
```



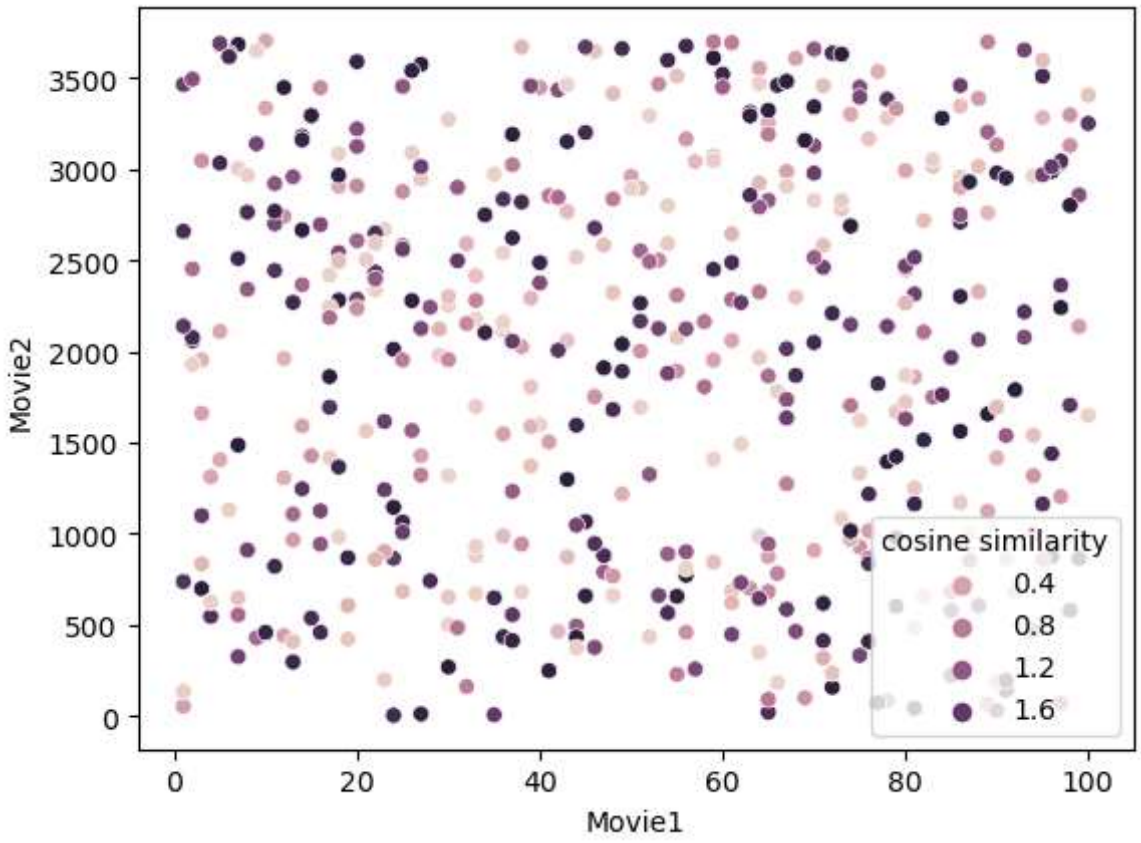
In [80]:
```
movie_svd2_ranks = get_cosine_ranks(svd2.qi, 100, ['Movie1', 'Movie2', 'cosine
movie_svd2_ranks_sample = movie_svd2_ranks.sample(500)
movie_svd2_ranks_sample
```

Out[80]:

	Movie1	Movie2	cosine similarity
65265	18	2282	1.960874
278889	76	1016	0.465366
94903	26	2280	1.961738
83125	23	1617	1.503325
287422	78	2139	1.463318
...	...	...	...
72629	20	2236	0.929479
197253	54	890	1.119412
258473	70	2830	0.012908
217937	59	3049	0.044304
72631	20	2238	0.595678

500 rows × 3 columns

In [81]:
```
sns.scatterplot(x = movie_svd2_ranks_sample['Movie1'], y = movie_svd2_ranks_sa
plt.show()
```



Questionnaire

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

In [83]:
```
user_ratings_ques = pd.merge(users, ratings, on = "UserID", how = "inner")
user_ratings_movies_ques = pd.merge(user_ratings_ques, movies, on = "MovieID",
user_ratings_movies_ques
```

Out[83]:

	UserID	Gender	Age	Occupation	rating_mean	no_of_ratings	MovieID	Ratin
0	1	F	Under 18	K-12 student	4.188679	53	1193	
1	2	M	56+	self-employed	3.713178	129	1193	
2	12	M	25-34	programmer	3.826087	23	1193	
3	15	M	25-34	executive/managerial	3.323383	201	1193	
4	17	M	50-55	academic/educator	4.075829	211	1193	
...	...	...	...	...	...	...	...	
1000204	5949	M	18-24	technician/engineer	3.497238	181	2198	
1000205	5675	M	35-44	sales/marketing	3.807167	586	2703	
1000206	5780	M	18-24	technician/engineer	3.741313	259	2845	
1000207	5851	F	18-24	writer	4.012346	162	3607	
1000208	5938	M	25-34	academic/educator	3.910526	190	2909	

1000209 rows × 11 columns

In [84]:
```
age_ratings_q1 = user_ratings_movies_ques.groupby(['Age']).agg({"Title":"nuniq
age_ratings_q1
```

Out[84]:

	Age	Title
1	25-34	3508
2	35-44	3447
0	18-24	3393
3	45-49	3288
4	50-55	3258
5	56+	2913
6	Under 18	2650

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

In [85]:
```
profession_ratings_q2 = user_ratings_movies_ques.groupby(['Occupation']).agg({
profession_ratings_q2
```

Out[85]:

	Occupation	Title
11	other	3448
4	college/grad student	3363
20	writer	3330
7	executive/managerial	3269
1	academic/educator	3218
2	artist	3145
16	self-employed	3123
17	technician/engineer	3087
6	doctor/health care	3011
14	sales/marketing	2953

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

In [86]:
```
gender_ratings_q3 = user_ratings_movies_ques.groupby(['Gender']).agg({"Title":
gender_ratings_q3
```

Out[86]:

	Gender	Title
1	M	3671
0	F	3481

The answer to above question is TRUE

4. Most of the movies present in our dataset were released in which decade?

a. 70s b. 90s c. 50s d. 80s

In [87]:
```
user_ratings_movies_ques['Release year'] = user_ratings_movies_ques['Title'].s
```

In [88]:
```
years = user_ratings_movies_ques['Release year'].unique().tolist()
print(min(years))
print(max(years))
```

```
1919
2000
```

In [89]:
```
decade_bins = [1910,1920,1930,1940,1950,1960,1970,1980,1990,2000]
decades_labels = ['1910-1920','1920-1930','1930-1940', '1940-1950','1950-1960'
user_ratings_movies_ques['Release decade'] = pd.cut(user_ratings_movies_ques['
```

In [90]:
```
release_decade_q4 = user_ratings_movies_ques.groupby(['Release decade']).agg({
release_decade_q4
```

Out[90]:

	Release decade	Title
8	1990-2000	2229
7	1980-1990	626
6	1970-1980	266
5	1960-1970	188
4	1950-1960	167
3	1940-1950	115
2	1930-1940	82
1	1920-1930	29
0	1910-1920	4

5. The movie with maximum no. of ratings is

In [91]:
```
max_ratings_q5 = user_ratings_movies_ques.groupby(['Title']).agg({"no_of_ratin
max_ratings_q5
```

Out[91]:

	Title	no_of_ratings
127	American Beauty (1999)	3428
3153	Star Wars: Episode IV - A New Hope (1977)	2991
3154	Star Wars: Episode V - The Empire Strikes Back...	2990
3155	Star Wars: Episode VI - Return of the Jedi (1983)	2883
1789	Jurassic Park (1993)	2672
2894	Saving Private Ryan (1998)	2653
3293	Terminator 2: Judgment Day (1991)	2649
2113	Matrix The (1999)	2590
258	Back to the Future (1985)	2583
2990	Silence of the Lambs The (1991)	2578

In [92]:
```
user_ratings_movies_ques['Title'] = user_ratings_movies_ques['Title'].str[:-6]
```

In [93]:
```
user_ratings_movies_ques
```

Out[93]:

	UserID	Gender	Age	Occupation	rating_mean	no_of_ratings	MovieID	Ratin
0	1	F	Under 18	K-12 student	4.188679	53	1193	
1	2	M	56+	self-employed	3.713178	129	1193	
2	12	M	25-34	programmer	3.826087	23	1193	
3	15	M	25-34	executive/managerial	3.323383	201	1193	
4	17	M	50-55	academic/educator	4.075829	211	1193	
...	...	...	...	...	...	...	...	
1000204	5949	M	18-24	technician/engineer	3.497238	181	2198	
1000205	5675	M	35-44	sales/marketing	3.807167	586	2703	
1000206	5780	M	18-24	technician/engineer	3.741313	259	2845	
1000207	5851	F	18-24	writer	4.012346	162	3607	
1000208	5938	M	25-34	academic/educator	3.910526	190	2909	

1000209 rows × 13 columns

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

Ans. On the basis of approach, Collaborative Filtering methods can be classified into 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.

RMSE (testset)	0.8914	0.8902	0.8894	0.8903	0.0008
MAE (testset)	0.7040	0.7026	0.7010	0.7025	0.0013
RMSE (trainset)	0.8614	0.8626	0.8596	0.8612	0.0012
MAE (trainset)	0.6809	0.6813	0.6791	0.6804	0.0010

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

[[1 0]
[3 7]]

In [94]:
```
#Ans.

from scipy.sparse import csr_matrix
array = np.array([[1,0],[3,7]])

arr_csr = csr_matrix(array).tocsr()
for ele in arr_csr:
    print(ele)
```

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

In [ ]: