Building 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.

In [96]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

from sklearn.model_selection import train_test_split

from sklearn.metrics.pairwise import cosine_similarity
from sklearn.neighbors import NearestNeighbors

from surprise import Reader, Dataset, SVD, SVDpp, NMF, KNNBaseline, KNNBasic, KNNWithM

from surprise.model_selection import cross_validate
```

In [5]:
```
# !pip install scikit-surprise
```

USERS FILE DESCRIPTION

==

User information is in the file "users.dat" and is in the following format:

UserID::Gender::Age::Occupation::Zip-code

MOVIES FILE DESCRIPTION

==

Movie information is in the file "movies.dat" and is in the following format:

MovieID::Title::Genres

RATINGS FILE DESCRIPTION

==

Rating information is in the file "ratings.dat" and is in the following format:

UserID::MovieID::Rating::Timestamp

In [7]:
```python
df_users = pd.read_csv('zee-users.dat', delimiter='::', encoding='latin-1')
df_movies = pd.read_csv('zee-movies.dat', delimiter='::', encoding='latin-1')
df_ratings = pd.read_csv('zee-ratings.dat', delimiter='::', encoding='latin-1')
```

In [8]:
```python
df_movies.rename(columns={'Movie ID':'MovieID'},inplace=True)  # replacing MovieID to
```

In [9]:
```python
df_users.head()
```

Out[9]:

	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

In [10]:
```python
df_users.shape
```

Out[10]:
```
(6040, 5)
```

In [11]:
```python
df_users.info()
```

```
<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   int64
 1   Gender      6040 non-null   object
 2   Age         6040 non-null   int64
 3   Occupation  6040 non-null   int64
 4   Zip-code    6040 non-null   object
dtypes: int64(3), object(2)
memory usage: 236.1+ KB
```

In [ ]:
```python

```

Replacing the encoded values of age and occupation with original values for better understanding

In [12]:
```python
age_categories = {1: "Under 18",18: "18-24",25: "25-34",35: "35-44",45: "45-49",50: "5
occupation_categories = {0: "other",1: "academic/educator",2: "artist",3: "clerical/ac
```

In [13]:
```python
df_users['Age Categories'] = df_users['Age'].replace(age_categories)
df_users['Occupation Category'] = df_users['Occupation'].replace(occupation_categories
```

In [14]:
```python
df_users.head()
```

Out[14]:

	UserID	Gender	Age	Occupation	Zip-code	Age Categories	Occupation Category
0	1	F	1	10	48067	Under 18	K-12 student
1	2	M	56	16	70072	56+	self-employed
2	3	M	25	15	55117	25-34	scientist
3	4	M	45	7	02460	45-49	executive/managerial
4	5	M	25	20	55455	25-34	writer

In [14]:

In [15]:
```
df_movies.head()
```

Out[15]:

	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

In [16]:
```
df_movies.shape
```

Out[16]:
```
(3883, 3)
```

In [17]:
```
df_movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   MovieID  3883 non-null   int64 
 1   Title    3883 non-null   object
 2   Genres   3883 non-null   object
dtypes: int64(1), object(2)
memory usage: 91.1+ KB
```

In [18]:
```
genres_list = list(set([l2 for l1 in df_movies.Genres.str.split('|').to_list() for l2
```

In [19]:
```
genres_list
```

Out[19]:
```
['Thriller',
 'Mystery',
 'Romance',
 'Film-Noir',
 'Horror',
 'Documentary',
 'Drama',
 'Western',
 'Adventure',
 "Children's",
 'Action',
 'War',
 'Sci-Fi',
 'Crime',
 'Musical',
 'Fantasy',
 'Animation',
 'Comedy']
```

In [1]:
```
# creating seperate columns for genres
def genre_flag(x):
    x = x['Genres']
    x_list = x.split('|')
    return [1 if genre in x_list else 0 for genre in genres_list]
```

In [21]:
```
temp_df = df_movies.apply(genre_flag,axis=1,result_type='expand')
```

In [22]:
```
for i,genre in enumerate(genres_list):
    df_movies[genre+'_G']=temp_df[i]
```

In [22]:
```

```

In [23]:
```
df_movies['Release year'] = df_movies['Title'].str[-5:-1].apply(int)  # extracting rel
```

In [24]:
```
df_movies.head()
```

Out[24]:

	MovieID	Title	Genres	Thriller_G	Mystery_G	Romance_G	Film-Noir_G	Horror
0	1	Toy Story (1995)	Animation|Children's|Comedy	0	0	0	0	
1	2	Jumanji (1995)	Adventure|Children's|Fantasy	0	0	0	0	
2	3	Grumpier Old Men (1995)	Comedy|Romance	0	0	1	0	
3	4	Waiting to Exhale (1995)	Comedy|Drama	0	0	0	0	
4	5	Father of the Bride Part II (1995)	Comedy	0	0	0	0	

5 rows × 22 columns

In [24]:

In [25]:

```
df_ratings.head()
```

Out[25]:

	UserID	MovieID	Rating	Timestamp
0	1	1193	5.0	978300760.0
1	1	661	3.0	978302109.0
2	1	914	3.0	978301968.0
3	1	3408	4.0	978300275.0
4	1	2355	5.0	978824291.0

In [26]:

```
df_ratings.shape
```

Out[26]:

```
(44714, 4)
```

In [27]:

```
df_ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44714 entries, 0 to 44713
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   UserID     44714 non-null  int64  
 1   MovieID    44714 non-null  int64  
 2   Rating     44713 non-null  float64
 3   Timestamp  44713 non-null  float64
dtypes: float64(2), int64(2)
memory usage: 1.4 MB
```

In [28]:
```
df_ratings['datetime'] = pd.to_datetime(df_ratings.Timestamp,unit='s')
```

In [29]:
```
df_ratings['year'] = df_ratings['datetime'].dt.year
df_ratings['hour'] = df_ratings['datetime'].dt.hour
df_ratings['Day of Week'] = df_ratings['datetime'].dt.day_name()
```

In [30]:
```
df_ratings.head()
```

Out[30]:

	UserID	MovieID	Rating	Timestamp	datetime	year	hour	Day of Week
0	1	1193	5.0	978300760.0	2000-12-31 22:12:40	2000.0	22.0	Sunday
1	1	661	3.0	978302109.0	2000-12-31 22:35:09	2000.0	22.0	Sunday
2	1	914	3.0	978301968.0	2000-12-31 22:32:48	2000.0	22.0	Sunday
3	1	3408	4.0	978300275.0	2000-12-31 22:04:35	2000.0	22.0	Sunday
4	1	2355	5.0	978824291.0	2001-01-06 23:38:11	2001.0	23.0	Saturday

In [30]:
```

```

Data Analysis and Visualization

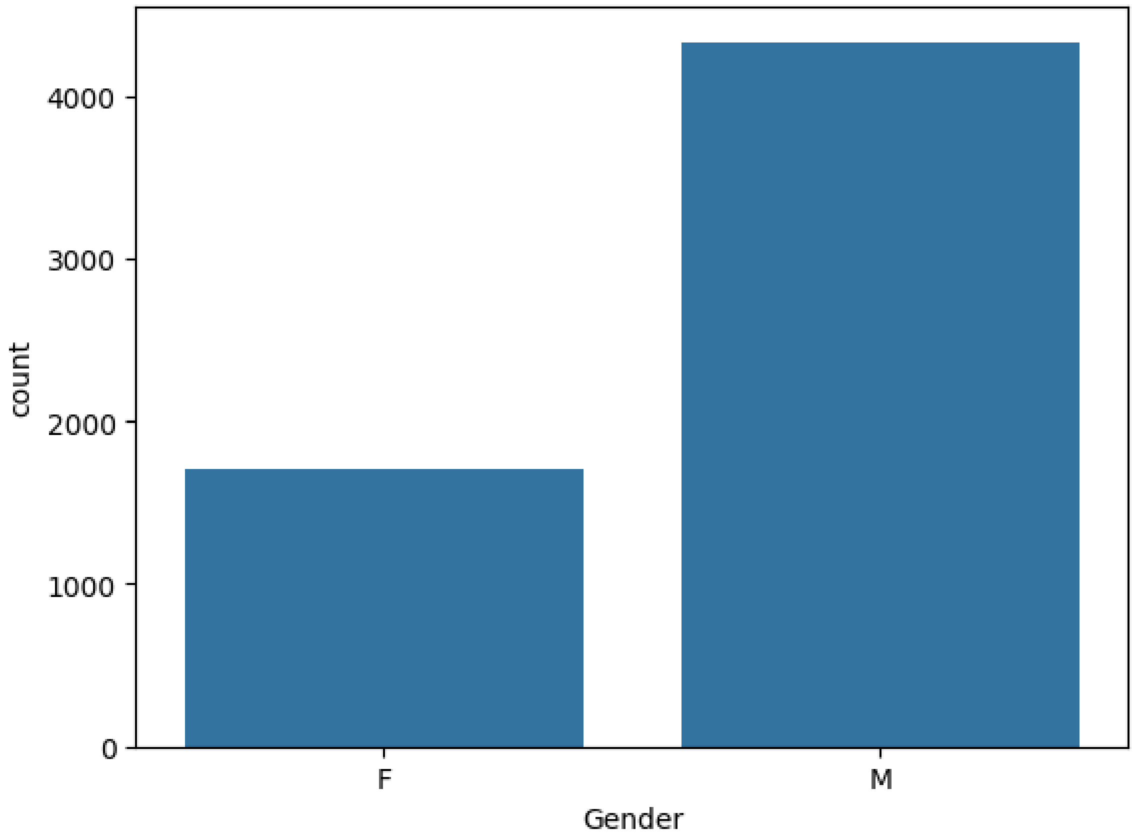
In [31]:
```
df_users.head()
```

Out[31]:

	UserID	Gender	Age	Occupation	Zip-code	Age Categories	Occupation Category
0	1	F	1	10	48067	Under 18	K-12 student
1	2	M	56	16	70072	56+	self-employed
2	3	M	25	15	55117	25-34	scientist
3	4	M	45	7	02460	45-49	executive/managerial
4	5	M	25	20	55455	25-34	writer

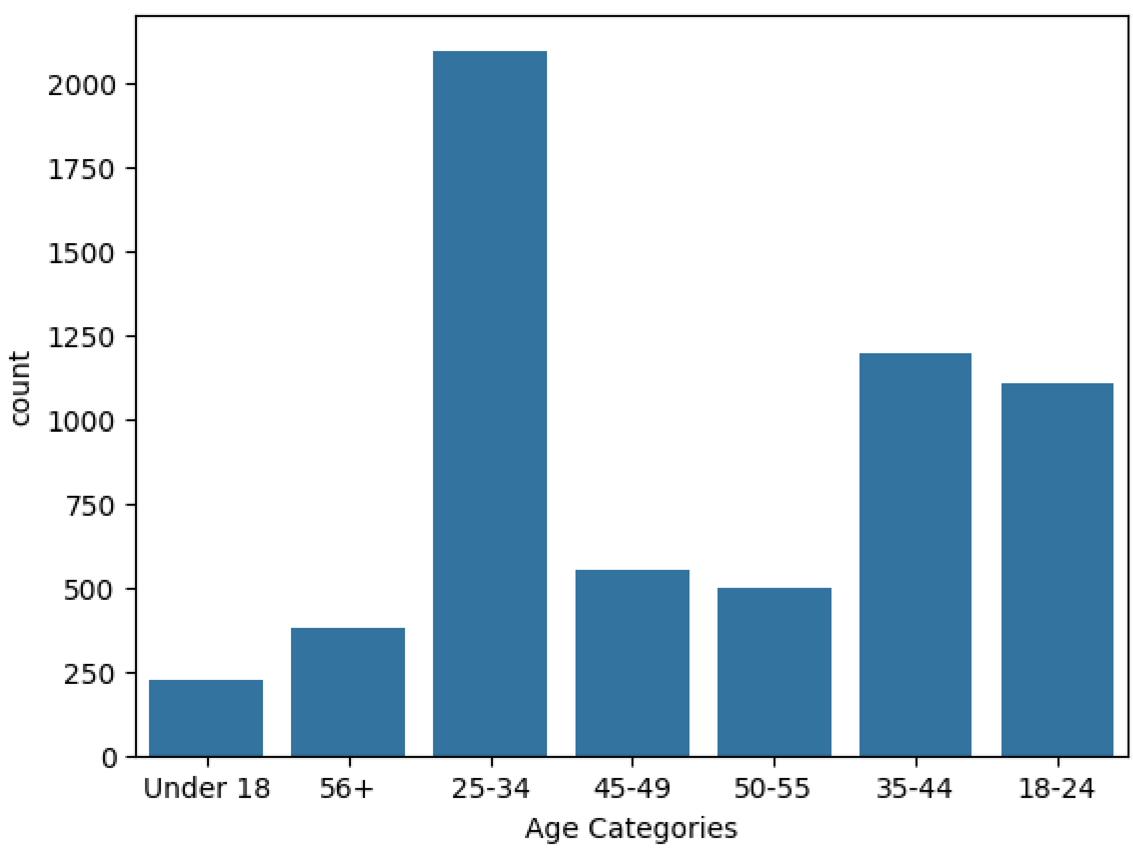
In [32]:
```
sns.countplot(data=df_users,x='Gender')
```

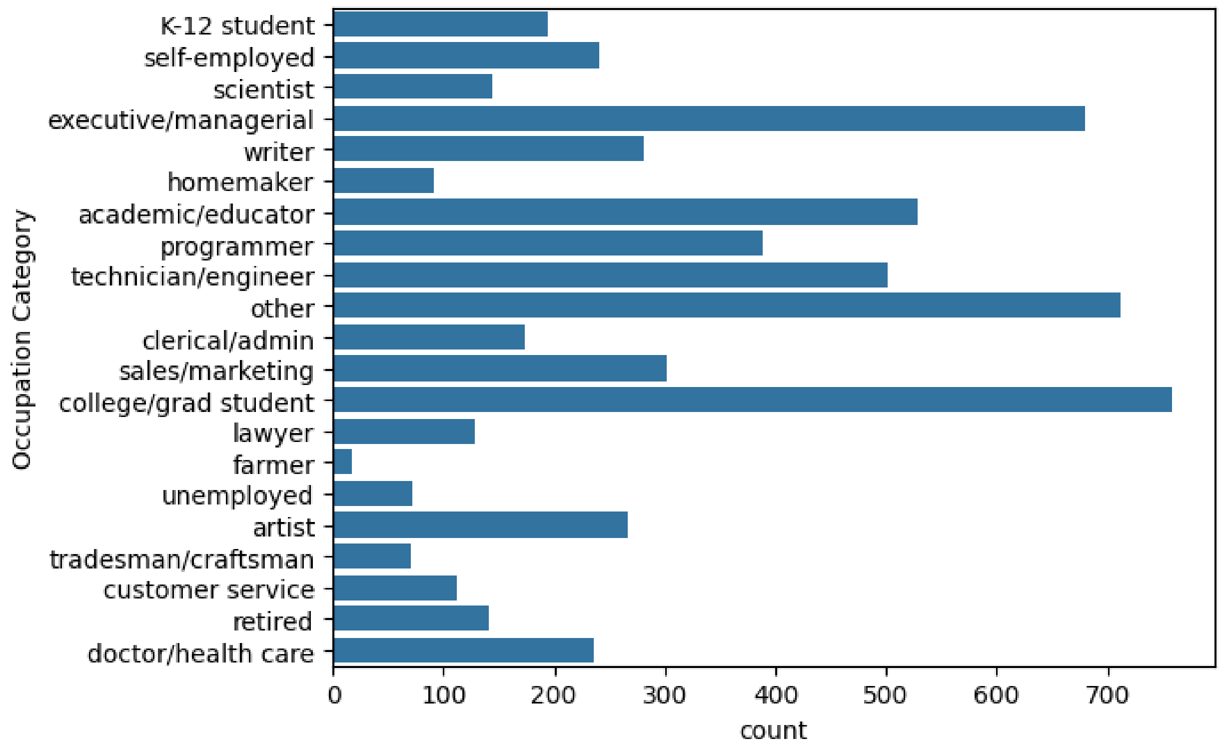
Out[32]:
```
<Axes: xlabel='Gender', ylabel='count'>
```



In [33]:
```
sns.countplot(data=df_users,x='Age Categories')
```

Out[33]: <Axes: xlabel='Age Categories', ylabel='count'>



In [34]:
```
sns.countplot(data=df_users,y='Occupation Category')
```
Out[34]:
```
<Axes: xlabel='count', ylabel='Occupation Category'>
```


In [34]:

In [35]:
```
df_movies.head()
```
Out[35]:

	MovieID	Title	Genres	Thriller_G	Mystery_G	Romance_G	Film-Noir_G	Horror
0	1	Toy Story (1995)	Animation|Children's|Comedy	0	0	0	0	
1	2	Jumanji (1995)	Adventure|Children's|Fantasy	0	0	0	0	
2	3	Grumpier Old Men (1995)	Comedy|Romance	0	0	1	0	
3	4	Waiting to Exhale (1995)	Comedy|Drama	0	0	0	0	
4	5	Father of the Bride Part II (1995)	Comedy	0	0	0	0	

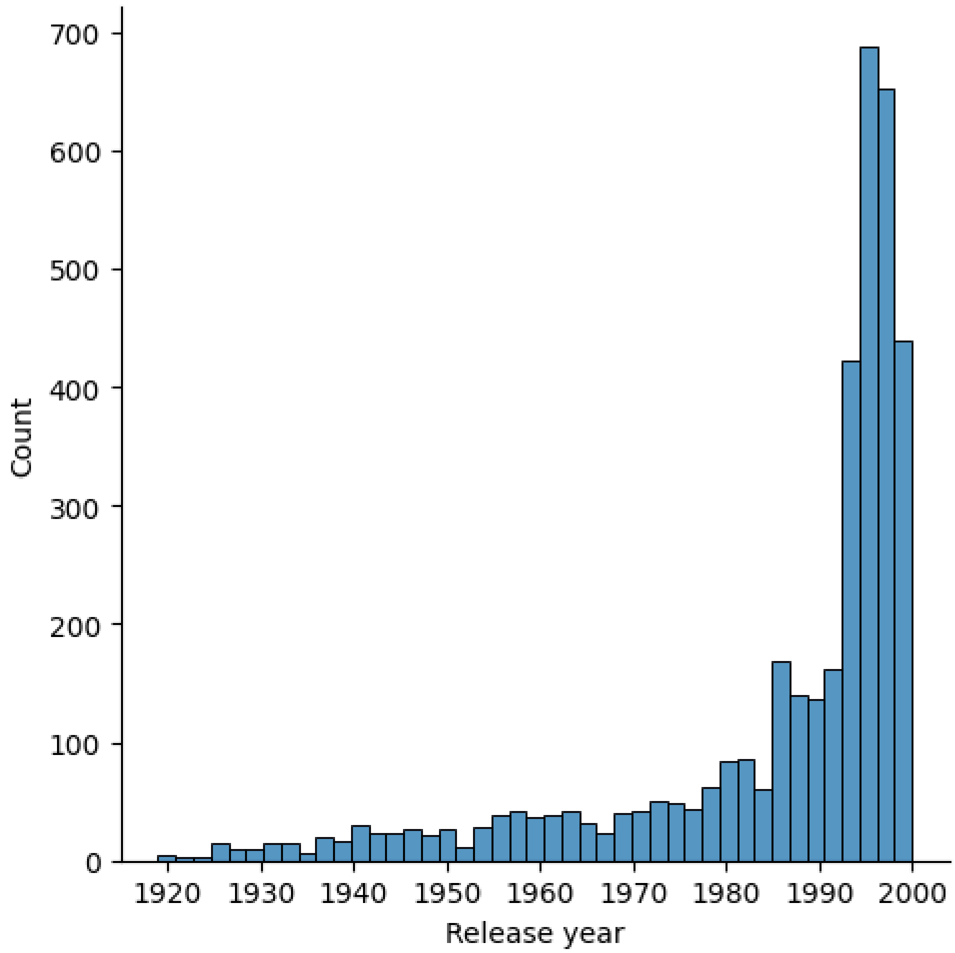
5 rows × 22 columns

```
In [ ]:
```

The release of movies are growing exponentially in Zee OTT

```
In [36]: sns.displot(data=df_movies,x='Release year')
```

```
Out[36]: <seaborn.axisgrid.FacetGrid at 0x7c6fb3f53d00>
```




```
In [36]:
```

```
In [37]: df_movies_genre = df_movies.melt(id_vars='Title',
             value_vars=[col for col in df_movies.columns if '_G' in col],
             var_name='Genre',
             value_name='Genre flag',)
```

```
In [38]: df_movies_genre = df_movies_genre[df_movies_genre['Genre flag']==1].reset_index()
```

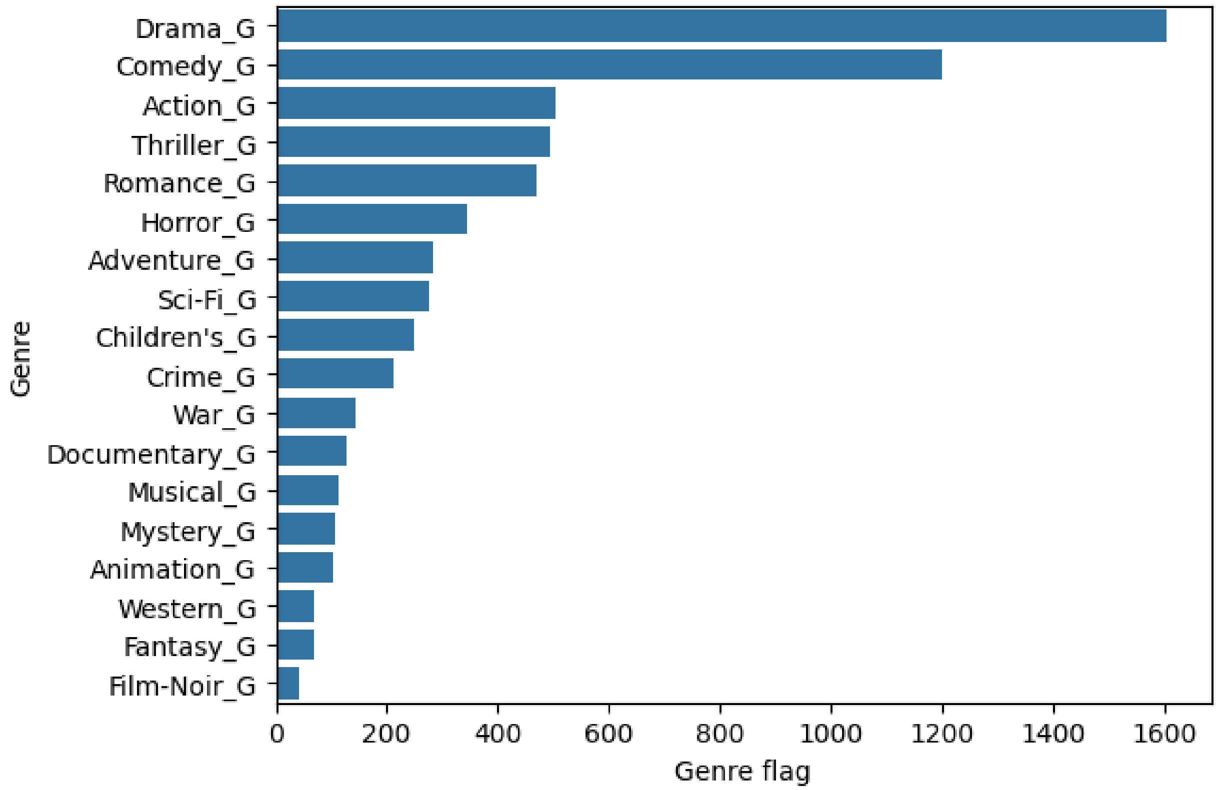
```
In [39]: df_movies_genre = df_movies_genre.groupby(['Genre'])['Genre flag'].count().sort_values
```

```
In [ ]:
```

Drama, Codedy are the most common Genres of the movies created in Zee

```
In [40]: sns.barplot(data=df_movies_genre,y='Genre',x='Genre flag')
```

Out[40]: <Axes: xlabel='Genre flag', ylabel='Genre'>



In [40]:

In [40]:

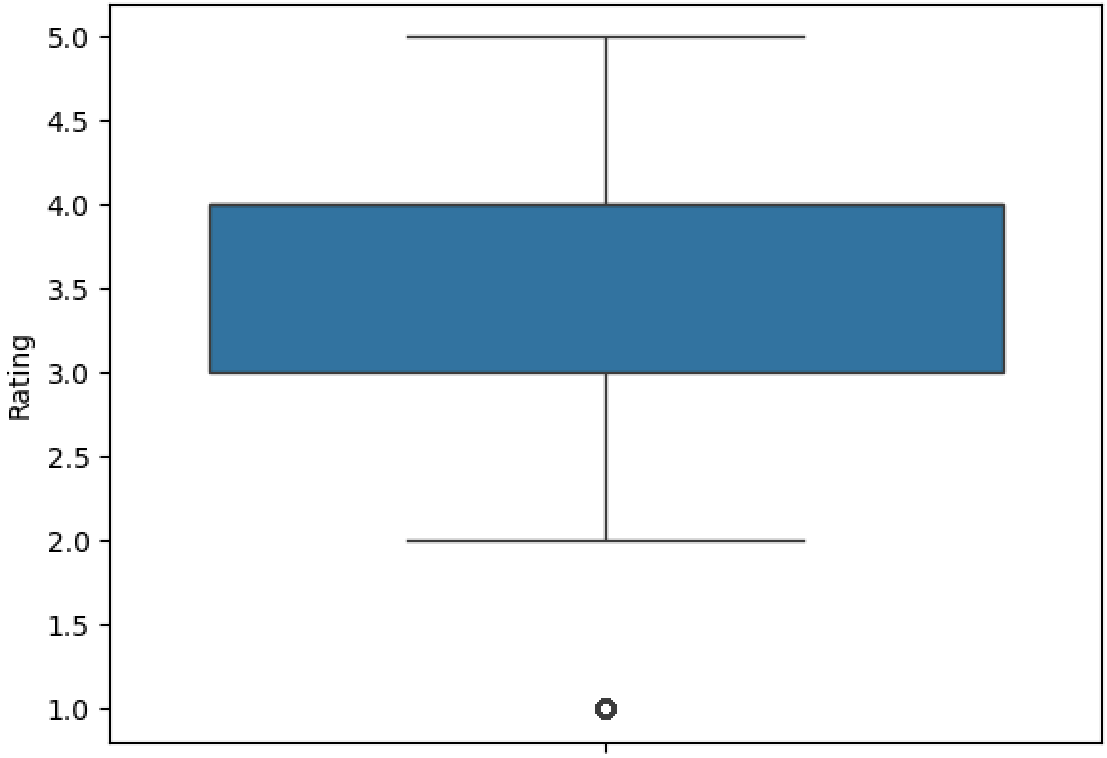
In [41]:
```
df_ratings.head()
```

Out[41]:

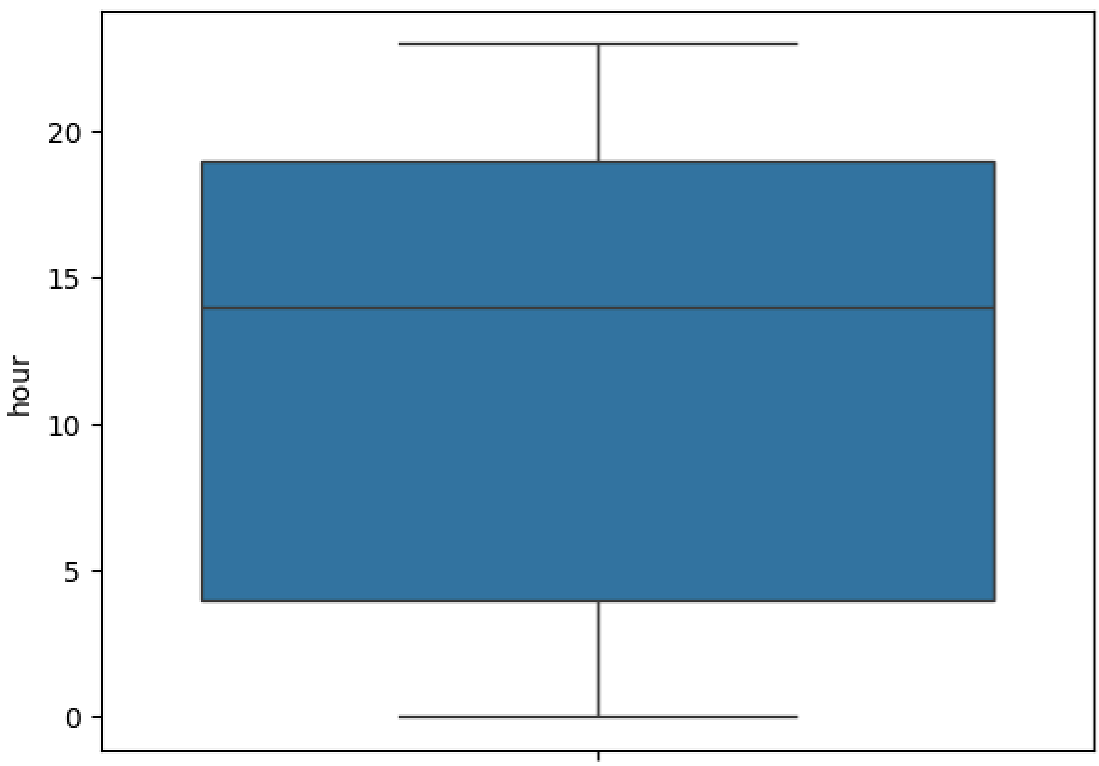
	UserID	MovieID	Rating	Timestamp	datetime	year	hour	Day of Week
0	1	1193	5.0	978300760.0	2000-12-31 22:12:40	2000.0	22.0	Sunday
1	1	661	3.0	978302109.0	2000-12-31 22:35:09	2000.0	22.0	Sunday
2	1	914	3.0	978301968.0	2000-12-31 22:32:48	2000.0	22.0	Sunday
3	1	3408	4.0	978300275.0	2000-12-31 22:04:35	2000.0	22.0	Sunday
4	1	2355	5.0	978824291.0	2001-01-06 23:38:11	2001.0	23.0	Saturday

In [42]:
```
sns.boxplot(df_ratings['Rating'])
```

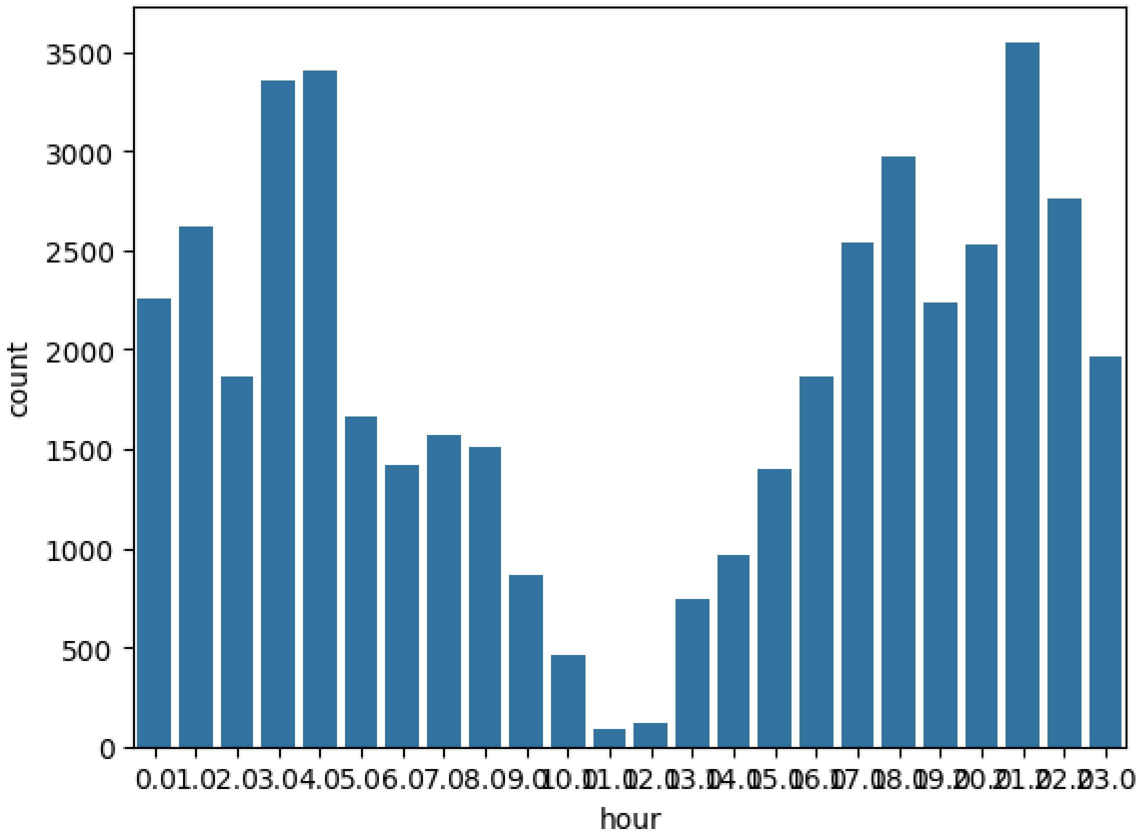
Out[42]: <Axes: ylabel='Rating'>



In [43]:
```
sns.boxplot(df_ratings['hour'])
```
Out[43]:
```
<Axes: ylabel='hour'>
```



In [44]:
```
sns.countplot(data = df_ratings, x='hour')
```
Out[44]:
```
<Axes: xlabel='hour', ylabel='count'>
```



If you observe carefully user watch movies that forms a bi-normal distribution starting afternoon 1pm and dropping at 12am midnight and aagin peaking early in the moring

In [ ]:

In [45]:
```
df_users.head()
```

Out[45]:

	UserID	Gender	Age	Occupation	Zip-code	Age Categories	Occupation Category
0	1	F	1	10	48067	Under 18	K-12 student
1	2	M	56	16	70072	56+	self-employed
2	3	M	25	15	55117	25-34	scientist
3	4	M	45	7	02460	45-49	executive/managerial
4	5	M	25	20	55455	25-34	writer

In [46]:
```
df_ratings = df_ratings.merge(df_users[['UserID','Age Categories']],on='UserID')
```

In [47]:
```
df_ratings.head()
```

Out[47]:

	UserID	MovieID	Rating	Timestamp	datetime	year	hour	Day of Week	Age Categories
0	1	1193	5.0	978300760.0	2000-12-31 22:12:40	2000.0	22.0	Sunday	Under 18
1	1	661	3.0	978302109.0	2000-12-31 22:35:09	2000.0	22.0	Sunday	Under 18
2	1	914	3.0	978301968.0	2000-12-31 22:32:48	2000.0	22.0	Sunday	Under 18
3	1	3408	4.0	978300275.0	2000-12-31 22:04:35	2000.0	22.0	Sunday	Under 18
4	1	2355	5.0	978824291.0	2001-01-06 23:38:11	2001.0	23.0	Saturday	Under 18

In [48]:

```
df_ratings.groupby('Age Categories')['Rating'].mean().apply(round,args={2})
```

Out[48]:

```
Age Categories
18-24       3.40
25-34       3.68
35-44       3.61
45-49       3.70
50-55       3.77
56+         3.57
Under 18    3.78
Name: Rating, dtype: float64
```

In [49]:

```
df_ratings.groupby('Age Categories')['Rating'].count()
```

Out[49]:

```
Age Categories
18-24       11398
25-34       15915
35-44        7501
45-49        5417
50-55        2293
56+          1211
Under 18      978
Name: Rating, dtype: int64
```

We can still caclulate

Distrubution between no of users who rated vs not rated \ Distrubution of different age and occupation categories who rated high or low \ Gender distribution across age categories and their rating behaviour \ Which age category people tend to watch older movies and newer movies Age categories people who are willing to rate the movies vs who arent willing to rate movies etc,. \

In [49]:

```
```

In [49]:

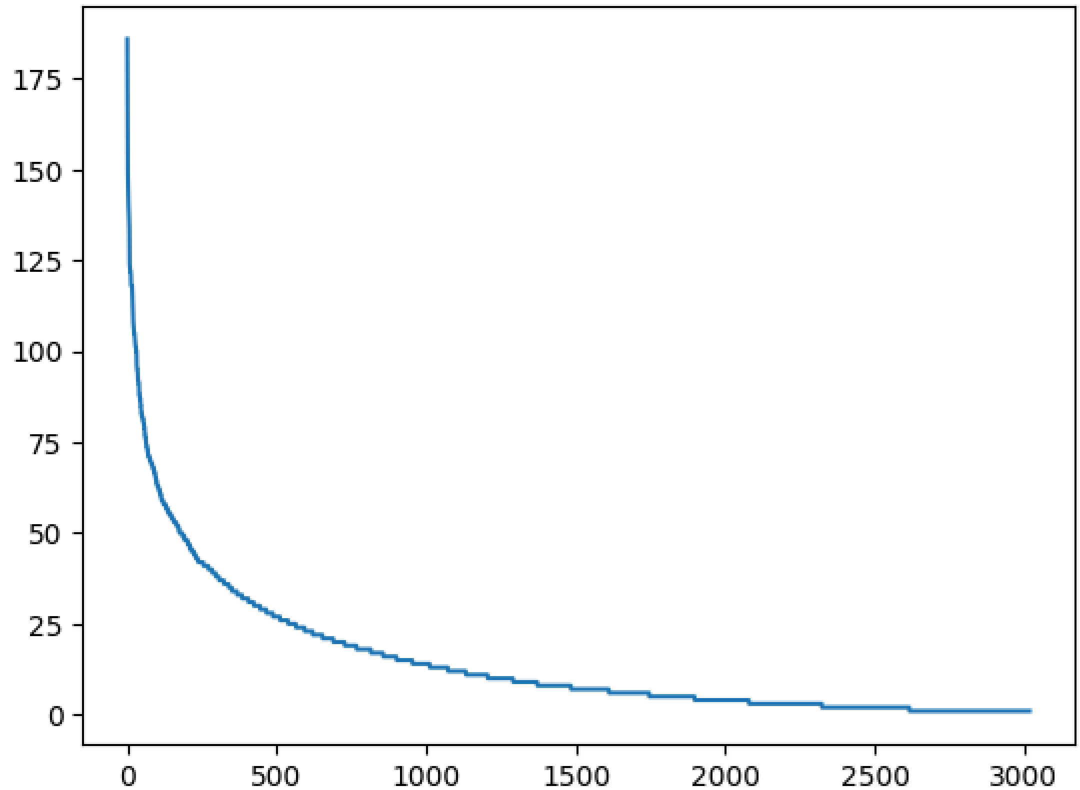
```
```

No of users in X and No of Ratings in Y, to find what is the minimum ratings to pick for our training model, From the obserbed below 2 graphs, we

approximately take top 1500 rated movies and 1500 rated users

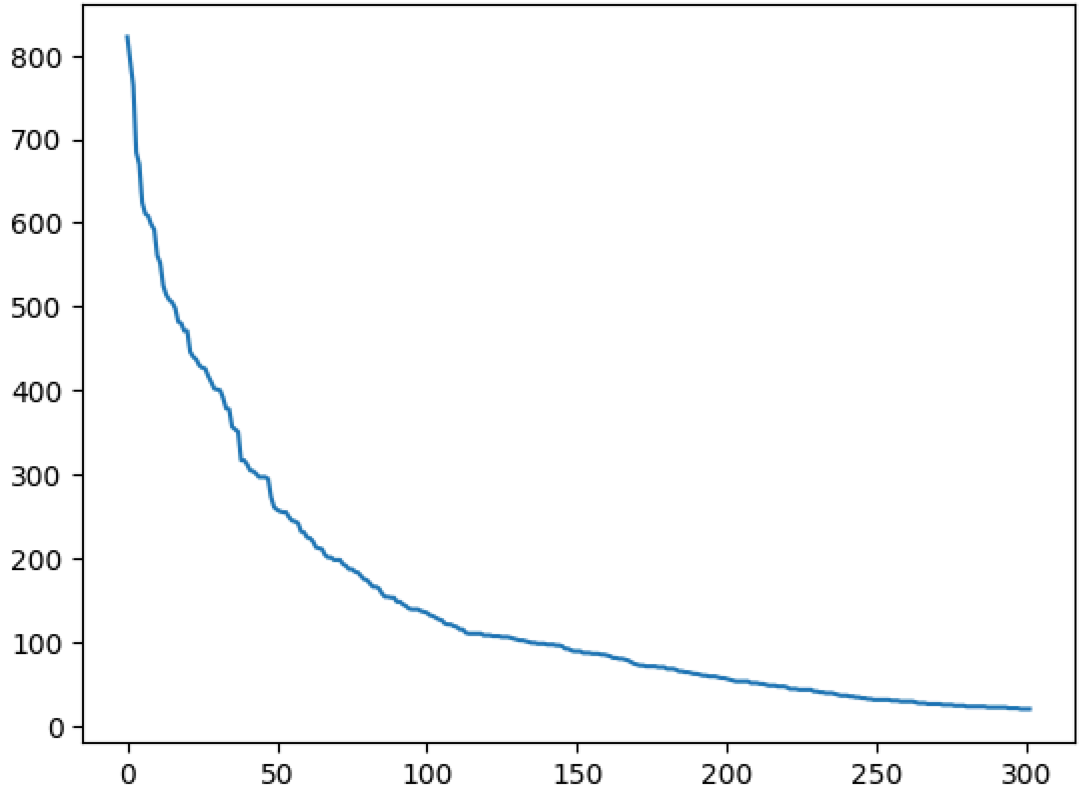
In [50]:
```
df_ratings.groupby('MovieID')['Rating'].count().sort_values(ascending=False).reset_inc
```

Out[50]:
```
<Axes: >
```



In [51]:
```
df_ratings.groupby('UserID')['Rating'].count().sort_values(ascending=False).reset_inde
```

Out[51]:
```
<Axes: >
```



In [52]:
```
df_movies_x = df_movies.merge(df_ratings.groupby('MovieID')['Rating'].count().reset_ir
```

In [53]:
```
df_movies_1500 = df_movies_x.loc[sorted(df_movies_x['Rating'].sort_values(ascending=Fa
```

In [56]:
```
# df_users_x.drop(columns=['Rating_x'],inplace=True)
# df_users_x.rename(columns={'Rating_y':'Rating'},inplace=True)
```

In [57]:
```
df_users_x = df_users.merge(df_ratings.groupby('UserID')['Rating'].count().reset_index
```

In [58]:
```
df_users_1500 = df_users_x.loc[sorted(df_users_x['Rating'].sort_values(ascending=False
```

In [59]:
```
df_movies_1500.shape,df_users_1500.shape
```

Out[59]:
```
((1500, 23), (302, 8))
```

In [59]:
```
```

In [60]:
```
df_ratings_1500=df_ratings.merge(df_users[['UserID']],on='UserID').merge(df_movies[['M
```

In [61]:
```
df_ratings_1500.head()
```

Out[61]:

	UserID	MovieID	Rating	Timestamp	datetime	year	hour	Day of Week	Age Categories	Title
0	1	1193	5.0	978300760.0	2000-12-31 22:12:40	2000.0	22.0	Sunday	Under 18	One Flew Over the Cuckoo's Nest (1975)
1	2	1193	5.0	978298413.0	2000-12-31 21:33:33	2000.0	21.0	Sunday	56+	One Flew Over the Cuckoo's Nest (1975)
2	12	1193	4.0	978220179.0	2000-12-30 23:49:39	2000.0	23.0	Saturday	25-34	One Flew Over the Cuckoo's Nest (1975)
3	15	1193	4.0	978199279.0	2000-12-30 18:01:19	2000.0	18.0	Saturday	25-34	One Flew Over the Cuckoo's Nest (1975)
4	17	1193	5.0	978158471.0	2000-12-30 06:41:11	2000.0	6.0	Saturday	50-55	One Flew Over the Cuckoo's Nest (1975)

In [61]:

In [62]:
```
matrix = df_ratings_1500.pivot(index='UserID', columns='Title', values='Rating')
```

In [63]:
```
matrix
```

Out[63]:

Title	$1,000,000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	'burbs, The (1989)	...And Justice for All (1979)	10 Things I Hate About You (1999)	101 Dalmatians (1961)	101 Dalmatians (1996)	12 Angry Men (1957)	13t Warrio Th (199
UserID										
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	
298	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
299	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	Na
300	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
301	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	Na
302	NaN	NaN	NaN	4.0	NaN	5.0	NaN	NaN	NaN	2.

302 rows × 3020 columns

In [66]:
```
round(matrix.isna().sum().sum()/matrix.notna().sum().sum())  # sparse, for every value
```

Out[66]:
```
19
```

In [67]:
```
matrix.shape
```

Out[67]:
```
(302, 3020)
```

In [ ]:

Helper functions for predicting User-User-matrix and Item-Item-matrix

In [68]:
```
def fetch_items_from_user_user_matrix(matrix,similarity_matrix,picked_userid,no_simila
    user_similarity_corr = similarity_matrix.copy().drop(index=picked_userid)

    # Get top n similar users
    similar_users = user_similarity_corr[user_similarity_corr[picked_userid]>user_simi

    # Movies that the target user has watched
    picked_userid_watched = matrix_train[matrix_train.index == picked_userid].dropna(a

    # Movies that similar users watched. Remove movies that none of the similar users
    similar_user_movies = matrix_train[matrix_train.index.isin(similar_users.index)].c
```

```python
    similar_user_movies.drop(picked_userid_watched.columns,axis=1, inplace=True, error

    # A dictionary to store item scores
    item_score = {}

    # Loop through items
    for i in similar_user_movies.columns:
      # Get the ratings for movie i
      movie_rating = similar_user_movies[i]
      # Create a variable to store the score
      total = 0
      # Create a variable to store the number of scores
      count = 0
      # Loop through similar users
      for u in similar_users.index:
        # If the movie has rating
        if pd.isna(movie_rating[u]) == False:
          # Score is the sum of user similarity score multiply by the movie rating
          score = similar_users[u] * movie_rating[u]
          # Add the score to the total score for the movie so far
          total += score
          # Add 1 to the count
          count +=1
      # Get the average score for the item
      item_score[i] = total / count

    # Convert dictionary to pandas dataframe
    item_score = pd.DataFrame(item_score.items(), columns=['movie', 'movie_score'])

    # Sort the movies by score
    ranked_item_score = item_score.sort_values(by='movie_score', ascending=False)

    return ranked_item_score.head(top_n_movies)
```

In [69]:
```python
def fetch_items_from_item_item_matrix(matrix,item_similarity, picked_userid, picked_mo
    picked_userid_watched = pd.DataFrame(matrix.T[picked_userid].dropna(axis=0, how='a
                          .sort_values(ascending=False))\
                          .reset_index()\
                          .rename(columns={picked_userid:'rating'})
    # Similarity score of the movie American Pie with all the other movies
    picked_movie_similarity_score = item_similarity[[picked_movie]].reset_index().rena

    picked_userid_watched_similarity = pd.merge(left=picked_userid_watched,
                                                right=picked_movie_similarity_score,
                                                on='Title',
                                                how='inner')\
                                        .sort_values('similarity_score', ascending=Fa
    predicted_rating = round(np.average(picked_userid_watched_similarity['rating'],
                                 weights=picked_userid_watched_similarity['similari

    print(f"The predicted rating for '{picked_movie}' by user '{picked_userid}' is '{p
```

In [69]:
```python

```

In [69]:
```python

```

In [69]:
```python

```

In [69]:

In [69]:

In [70]:
```
matrix_train,matrix_test = train_test_split(matrix,test_size=.15,random_state=42)
```

In [71]:
```
matrix_train.shape,matrix_test.shape
```

Out[71]:
```
((256, 3020), (46, 3020))
```

In [72]:
```
matrix_train.head()
```

Out[72]:

Title	$1,000,000 Duck (1971)	'Night Mother (1986)	'Til There Was You (1997)	'burbs, The (1989)	...And Justice for All (1979)	10 Things I Hate About You (1999)	101 Dalmatians (1961)	101 Dalmatians (1996)	12 Angry Men (1957)	13t Warrio Th (199
UserID										
281	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
79	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
292	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
233	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
220	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

5 rows × 3020 columns

In [72]:

User similarity using User-User Pearson correlation matrix

In [73]:
```
user_similarity_corr = matrix_train.T.corr()
```

In [74]:
```
user_similarity_corr.head()
```

Out[74]:

UserID	281	79	292	233	220	256	64	83	237
UserID									
281	1.000000	1.000000	0.324617	0.500000	-0.187500	0.612372	-0.166667	-0.514330	-0.259860
79	1.000000	1.000000	1.000000	NaN	0.528085	0.944911	NaN	-0.322749	NaN
292	0.324617	1.000000	1.000000	NaN	0.650791	0.481218	0.936586	0.023837	0.089061
233	0.500000	NaN	NaN	1.000000	0.333333	-0.102062	1.000000	NaN	0.021148
220	-0.187500	0.528085	0.650791	0.333333	1.000000	0.583632	0.555556	0.032556	0.235213

5 rows × 256 columns

Top 10 movies recommended for user id 220 based on pearson correlation matrix

In [78]:
```
fetch_items_from_user_user_matrix(matrix_train,user_similarity_corr,220)
```

Out[78]:

	movie	movie_score
1006	Swamp Thing (1982)	5.0
498	Hidden, The (1987)	5.0
1035	Thing From Another World, The (1951)	5.0
352	Eyes Without a Face (1959)	5.0
908	Serial Mom (1994)	5.0
784	Phantasm (1979)	5.0
702	Mummy, The (1932)	5.0
250	Cronos (1992)	5.0
495	Help! (1965)	5.0
836	Raven, The (1963)	5.0

In [75]:
```
```

User similarity using User-User Cosine similarity matrix

In [76]:
```
user_similarity_cosine = cosine_similarity(matrix_train.fillna(0))
```

In [77]:
```
user_similarity_cosine = pd.DataFrame(data=user_similarity_cosine,columns=matrix_trair
```

Top 10 movies recommended for user id 5542 based on cosine similarity matrix

In [ ]:
```
fetch_items_from_user_user_matrix(matrix_train,user_similarity_cosine,5542)
```

Out[ ]:

	movie	movie_score
439	Grand Day Out, A (1992)	1.720979
700	My Best Fiend (Mein liebster Feind) (1999)	1.720979
753	Once Upon a Time... When We Were Colored (1995)	1.668122
577	Kundun (1997)	1.668122
102	Beautiful Thing (1996)	1.668122
877	Romeo and Juliet (1968)	1.664460
151	Bound (1996)	1.635513
383	Frances (1982)	1.635513
220	City of Angels (1998)	1.635513
424	Gods and Monsters (1998)	1.635513

In [ ]:

Item similarity using Item-Item Pearson correlation matrix

In [ ]:
```
item_similarity_corr = matrix_train.corr()
```

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

Out[ ]:

Title	$1,000,000 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)	Dal
Title								
$1,000,000 Duck (1971)	1.000000e+00	5.222330e-01	NaN	9.410021e-17	0.422577	NaN	-0.455383	(
'Night Mother (1986)	5.222330e-01	1.000000e+00	-6.966594e-17	2.174729e-01	0.235483	NaN	0.260513	(
'Til There Was You (1997)	NaN	-6.966594e-17	1.000000e+00	8.503904e-01	0.719676	NaN	0.202885	(
'burbs, The (1989)	9.410021e-17	2.174729e-01	8.503904e-01	1.000000e+00	0.179745	NaN	0.157364	(
...And Justice for All (1979)	4.225771e-01	2.354830e-01	7.196763e-01	1.797450e-01	1.000000	NaN	0.254492	(

5 rows × 3706 columns

In [ ]:

```
picked_userid = 5542
picked_movie = 'American Pie (1999)'
```

In [ ]:

```
fetch_items_from_item_item_matrix(matrix_train,item_similarity_corr, picked_userid, pi
```

```
The predicted rating for 'American Pie (1999)' by user '5542' is '4.629108'
```

In [ ]:

Item similarity using Item-Item Cosine similairty matrix

In [ ]:

```
item_similarity_cosine = cosine_similarity(matrix_train.T.fillna(0))
```

In [ ]:

```
item_similarity_cosine = pd.DataFrame(data=item_similarity_cosine,columns=matrix_trair
```

In [ ]:

```
item_similarity_cosine.head()
```

Out[ ]:

Title	$1,000,000 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)	Dalm… (
Title									
$1,000,000 Duck (1971)	1.000000	0.077175	0.040719	0.089714	0.066385	0.000000	0.058165	0.196912	0.1
'Night Mother (1986)	0.077175	1.000000	0.121769	0.108498	0.160112	0.000000	0.083533	0.151401	0.1
'Til There Was You (1997)	0.040719	0.121769	1.000000	0.099920	0.074803	0.085592	0.118682	0.123273	0.1
'burbs, The (1989)	0.089714	0.108498	0.099920	1.000000	0.146753	0.000000	0.184452	0.234000	0.1
...And Justice for All (1979)	0.066385	0.160112	0.074803	0.146753	1.000000	0.000000	0.077838	0.190840	0.1

5 rows × 3706 columns

In [ ]:
```
fetch_items_from_item_item_matrix(matrix_train,item_similarity_cosine, picked_userid,
```
```
The predicted rating for 'American Pie (1999)' by user '5542' is '4.633477'
```

In [ ]:

In [ ]:

In [ ]:
```
knn = NearestNeighbors(n_neighbors=5, algorithm='ball_tree').fit(matrix_train.T.fillna
```

In [ ]:
```
distances, indices = knn.kneighbors([matrix_train.iloc[:,50].fillna(0)])
```

In [ ]:
```
distances
```
Out[ ]:
```
array([[0.        , 8.36660027, 8.71779789, 8.71779789, 8.71779789]])
```

In [ ]:
```
indices
```
Out[ ]:
```
array([[  50, 2150,  398,  256,  540]], dtype=int64)
```

In [ ]:
```
matrix_train.iloc[:,indices[0]].columns
```
Out[ ]:
```
Index(['Actor's Revenge, An (Yukinojo Henge) (1963)',
       'Metisse (Café au Lait) (1993)', 'Billy's Holiday (1995)',
       'Back Stage (2000)', 'Broken Vessels (1998)'],
      dtype='object', name='Title')
```

The nearest movie for "Actor's Revenge, An (Yukinojo Henge) (1963)" are :

'Metisse (Café au Lait) (1993)', \ 'Billy's Holiday (1995)', \ 'Back Stage (2000)', \ 'Broken Vessels (1998)'

In [82]:

```

```

Matrix factorization

In [79]:

```
reader = Reader(rating_scale=(0, 5))
```

In [85]:

```
data = Dataset.load_from_df(df_ratings[['UserID', 'MovieID', 'Rating']], reader)
```

In [98]:

```
benchmark = []
# Iterate over all algorithms
for algorithm in [SVD(), SVDpp(), NMF(), KNNBaseline(), KNNBasic(), KNNWithMeans(), KN
    # Perform cross validation
    results = cross_validate(algorithm, data, measures=['RMSE','MAE'], cv=3, verbose=F

    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    tmp = pd.concat([tmp,pd.Series([str(algorithm).split(' ')[0].split('.')[-1]], inde
    benchmark.append(tmp)
```

In [99]:

```
pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
```

Out[99]:

Algorithm	test_rmse	test_mae	fit_time	test_time
KNNBasic	1.063808	0.834582	0.032180	0.852045
KNNWithMeans	1.212207	0.912728	0.054488	1.168667
KNNWithZScore	1.212491	0.913058	0.063367	0.790620
NMF	1.784164	1.388071	0.841373	0.119940
SVD	1.786608	1.391861	0.670844	0.150759
KNNBaseline	1.788655	1.394545	0.096315	0.867506
SVDpp	1.792795	1.395840	13.431853	4.167411

Here we trained and tested our models on Matrix Factorization concept using Surprise library, above table shows with different techniques of Matrix Factorization we brought different results, here if you observe we got the best results i.e low RMSE and MAE for KNNBasic algorithm.

In [ ]:

In [99]:

In [ ]:

Questionnaire:

25-34 age group all the most rated group

In [105…

```
rated_users = df_users['UserID'].isin(df_ratings['UserID'])
df_users[rated_users].groupby('Age Categories')['UserID'].count().sort_values(ascendir
```

Out[105]:

```
Age Categories
25-34       99
18-24       72
35-44       54
45-49       36
50-55       18
56+         13
Under 18    10
Name: UserID, dtype: int64
```

In [101…

college/grad student are the most movie rated user category, 42 people rated from this category

In [103…

```
df_users[rated_users].groupby('Occupation Category')['UserID'].count().sort_values(asc
```

Out[103]:

```
Occupation Category
college/grad student    42
other                   31
academic/educator       30
executive/managerial    30
technician/engineer     29
programmer              24
self-employed           16
writer                  14
sales/marketing         11
K-12 student            11
clerical/admin          11
artist                  10
lawyer                   8
homemaker                7
scientist                6
retired                  5
unemployed               5
customer service         4
tradesman/craftsman      4
doctor/health care       3
farmer                   1
Name: UserID, dtype: int64
```

In [ ]:

Males rated most of the users

In [104...
```
df_users[rated_users].groupby('Gender')['UserID'].count().sort_values(ascending=False)
```

Out[104]:
```
Gender
M    213
F     89
Name: UserID, dtype: int64
```

In [ ]:

2000's has the most movie releases

In [116...
```
df_movies['Release decade'] = (round(df_movies['Release year']/10)*10).apply(int)
```

In [117...
```
df_movies.groupby(['Release decade'])['MovieID'].count().sort_values(ascending=False)
```

Out[117]:
```
Release decade
2000    1778
1990     965
1980     420
1960     217
1970     197
1940     119
1950     115
1930      55
1920      17
Name: MovieID, dtype: int64
```

In [ ]:

American Beauty (1999) is the most rated movie

In [123...
```
df_ratings = df_ratings.merge(df_movies[['MovieID','Title']],on='MovieID')
```

In [124...
```
df_ratings.groupby(['Title'])['Rating'].count().sort_values(ascending=False)
```

Out[124]:
```
Title
American Beauty (1999)                                    186
Jurassic Park (1993)                                      157
Star Wars: Episode V - The Empire Strikes Back (1980)     149
Saving Private Ryan (1998)                                143
Star Wars: Episode VI - Return of the Jedi (1983)         141
                                                         ...
Funhouse, The (1981)                                        1
Naked Man, The (1998)                                       1
G. I. Blues (1960)                                          1
Gate of Heavenly Peace, The (1995)                          1
$1,000,000 Duck (1971)                                      1
Name: Rating, Length: 3020, dtype: int64
```

In [ ]:

Top 3 movies similar to ‘Liar Liar’ on the item-based approach are

'Cable Guy, The (1996)', \ 'Home Alone 2: Lost in New York (1992)', \ 'Brady Bunch Movie, The (1995)'

In [ ]:

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

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

1.9 and 1.4 are the RMSE and MAPE that we got while evaluating the Matrix Factorization model

In [ ]:

The sparse 'row' matrix representation would be:

data = [1, 3, 7] indices = [0, 0, 1] indptr = [0, 1, 3]

In [ ]: