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

Problem Statement- 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.

RATINGS FILE DESCRIPTION

```
================================================================================
```

All ratings are contained in the file "ratings.dat" and are in the following format:

UserID::MovieID::Rating::Timestamp

UserIDs range between 1 and 6040

MovieIDs range between 1 and 3952

Ratings are made on a 5-star scale (whole-star ratings only)

Timestamp is represented in seconds

Each user has at least 20 ratings

USERS FILE DESCRIPTION

```
================================================================================
```

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

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

All demographic information is provided voluntarily by the users and is not checked for accuracy. Only users who have provided some demographic information are included in this data set.

Gender is denoted by a "M" for male and "F" for female

Age is chosen from the following ranges:

1: "Under 18"

18: "18-24"

25: "25-34"

35: "35-44"

45: "45-49"

50: "50-55"

56: "56+"

Occupation is chosen from the following choices:

0: "other" or not specified

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"

MOVIES FILE DESCRIPTION

==

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

MovieID::Title::Genres

Titles are identical to titles provided by the IMDB (including year of release)

Genres are pipe-separated and are selected from the following genres:

Action

Adventure

Animation

Children's

Comedy

Crime

Documentary

Drama

Fantasy

Film-Noir

Horror

Musical

Mystery

Romance

Sci-Fi

Thriller

War

Western

In [2]:
```
user = pd.read_csv("zee-users.dat", header= None,  names=["Data"] )
user
```

Out[2]:

	Data
0	UserID::Gender::Age::Occupation::Zip-code
1	1::F::1::10::48067
2	2::M::56::16::70072
3	3::M::25::15::55117
4	4::M::45::7::02460
...	...
6036	6036::F::25::15::32603
6037	6037::F::45::1::76006
6038	6038::F::56::1::14706
6039	6039::F::45::0::01060
6040	6040::M::25::6::11106

6041 rows × 1 columns

In [3]:
```
user[["UserID", "Gender", "Age", "Occupation", "Zip-code"]]= user["Data"].str.split(":
```

In [4]:
```
user
```

Out[4]:

	Data	UserID	Gender	Age	Occupation	Zip-code
0	UserID::Gender::Age::Occupation::Zip-code	UserID	Gender	Age	Occupation	Zip-code
1	1::F::1::10::48067	1	F	1	10	48067
2	2::M::56::16::70072	2	M	56	16	70072
3	3::M::25::15::55117	3	M	25	15	55117
4	4::M::45::7::02460	4	M	45	7	02460
...	...	...	...	...	...	...
6036	6036::F::25::15::32603	6036	F	25	15	32603
6037	6037::F::45::1::76006	6037	F	45	1	76006
6038	6038::F::56::1::14706	6038	F	56	1	14706
6039	6039::F::45::0::01060	6039	F	45	0	01060
6040	6040::M::25::6::11106	6040	M	25	6	11106

6041 rows × 6 columns

In [5]:
```
user.shape
```

Out[5]:
```
(6041, 6)
```

In [6]:
```
user = user.drop(["Data"], axis= 1)
```

In [7]:
```
user
```

Out[7]:

	UserID	Gender	Age	Occupation	Zip-code
0	UserID	Gender	Age	Occupation	Zip-code
1	1	F	1	10	48067
2	2	M	56	16	70072
3	3	M	25	15	55117
4	4	M	45	7	02460
...	...	...	...	...	...
6036	6036	F	25	15	32603
6037	6037	F	45	1	76006
6038	6038	F	56	1	14706
6039	6039	F	45	0	01060
6040	6040	M	25	6	11106

6041 rows × 5 columns

In [8]:
```
# Drop the first row
user = user.drop(0).reset_index(drop=True)
```

In [9]:
```
user.shape
```

Out[9]:
```
(6040, 5)
```

In [10]:
```
user
```

Out[10]:

	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 [11]:
```
user.dtypes
```

Out[11]:
```
UserID        object
Gender        object
Age           object
Occupation    object
Zip-code      object
dtype: object
```

Covering the Datatype of User DF

In [12]:
```
# Convert data types for user
user['UserID'] = user['UserID'].astype(int)
user['Age'] = user['Age'].astype(int)
user['Occupation'] = user['Occupation'].astype(int)
```

In [13]:
```
user.dtypes
```

Out[13]:
```
UserID         int32
Gender        object
Age            int32
Occupation     int32
Zip-code      object
dtype: object
```

In [14]:
```
rating = pd.read_csv("zee-ratings.dat",  delimiter='::', engine='python', header= None
```

In [15]:
```
rating
```

Out[15]:

	UserID	MovieID	Rating	Timestamp
0	UserID	MovieID	Rating	Timestamp
1	1	1193	5	978300760
2	1	661	3	978302109
3	1	914	3	978301968
4	1	3408	4	978300275
...	...	...	...	...
1000205	6040	1091	1	956716541
1000206	6040	1094	5	956704887
1000207	6040	562	5	956704746
1000208	6040	1096	4	956715648
1000209	6040	1097	4	956715569

1000210 rows × 4 columns

In [16]:

```
# Drop the first row
rating = rating.drop(0).reset_index(drop=True)
```

In [17]:

```
rating.shape
```

Out[17]:

```
(1000209, 4)
```

In [18]:

```
rating
```

Out[18]:

	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 [19]:
```
rating.dtypes
```

Out[19]:
```
UserID       object
MovieID      object
Rating       object
Timestamp    object
dtype: object
```

Coverting Datatypes of Rating DF

In [20]:
```
# Convert data types for rating
rating['UserID'] = rating['UserID'].astype(int)
rating['MovieID'] = rating['MovieID'].astype(int)
rating['Rating'] = rating['Rating'].astype(int)
rating['Timestamp'] = pd.to_datetime(rating['Timestamp'], unit='s')
```

In [21]:
```
rating.dtypes
```

Out[21]:
```
UserID                int32
MovieID               int32
Rating                int32
Timestamp    datetime64[ns]
dtype: object
```

In [22]:
```
# movie = pd.read_csv("zee-movies.dat", encoding='ISO-8859-1', on_bad_lines='skip',  d
movies = pd.read_csv("zee-movies.dat", skiprows=1, encoding = 'ISO-8859-1', delimiter=
```

In [23]:
```
movies
```

Out[23]:

	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

In [24]:
```
movies.shape
```

Out[24]:
```
(3883, 3)
```

In [25]:
```
Genre_Split= movies['Genres'].str.split('|')
```

In [26]:
```
movies['Genres']
```

Out[26]:
```
0         Animation|Children's|Comedy
1        Adventure|Children's|Fantasy
2                      Comedy|Romance
3                        Comedy|Drama
4                              Comedy
                     ...
3878                           Comedy
3879                            Drama
3880                            Drama
3881                            Drama
3882                   Drama|Thriller
Name: Genres, Length: 3883, dtype: object
```

In [27]:
```
Genre_Split
```

Out[27]:
```
0         [Animation, Children's, Comedy]
1        [Adventure, Children's, Fantasy]
2                       [Comedy, Romance]
3                         [Comedy, Drama]
4                                [Comedy]
                     ...
3878                             [Comedy]
3879                              [Drama]
3880                              [Drama]
3881                              [Drama]
3882                    [Drama, Thriller]
Name: Genres, Length: 3883, dtype: object
```

In [28]:
```
movies
```

Out[28]:

	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

In [29]:
```
#movies_splitted = movies.explode('Genres', ignore_index= True)
```

In [ ]:

In [30]:
```
movies.shape
```

Out[30]:
```
(3883, 3)
```

In [31]:
```
movies.dtypes
```

Out[31]:
```
MovieID     int64
Title      object
Genres     object
dtype: object
```

Checking for Duplicated Rows of all DF

In [32]:
```
#checking for duplicated rows in user
user.duplicated().sum()
```

Out[32]:
```
0
```

In [33]:
```
#checking for duplicated rows in rating
rating.duplicated().sum()
```

Out[33]:
```
0
```

In [34]:
```
#checking for duplicated rows in movie
movies.duplicated().sum()
```

Out[34]:
```
0
```

Checking for Null Values in all DF

In [35]:
```
#Checking for null values in user
user.isnull().sum()
```

Out[35]:
```
UserID        0
Gender        0
Age           0
Occupation    0
Zip-code      0
dtype: int64
```

In [36]:
```
#Checking for null values in rating
rating.isnull().sum()
```

Out[36]:
```
UserID       0
MovieID      0
Rating       0
Timestamp    0
dtype: int64
```

In [37]:
```
#Checking for null values in movie
movies.isnull().sum()
```

Out[37]:
```
MovieID    0
Title      0
Genres     0
dtype: int64
```

Exploratory Data Analysis

In [38]:
```
user.describe(include='all')
```

Out[38]:

	UserID	Gender	Age	Occupation	Zip-code
count	6040.000000	6040	6040.000000	6040.000000	6040
unique	NaN	2	NaN	NaN	3439
top	NaN	M	NaN	NaN	48104
freq	NaN	4331	NaN	NaN	19
mean	3020.500000	NaN	30.639238	8.146854	NaN
std	1743.742145	NaN	12.895962	6.329511	NaN
min	1.000000	NaN	1.000000	0.000000	NaN
25%	1510.750000	NaN	25.000000	3.000000	NaN
50%	3020.500000	NaN	25.000000	7.000000	NaN
75%	4530.250000	NaN	35.000000	14.000000	NaN
max	6040.000000	NaN	56.000000	20.000000	NaN

In [39]:
```
rating.describe()
```

Out[39]:

	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 [40]:
```
movies.describe(include='all')
```

Out[40]:

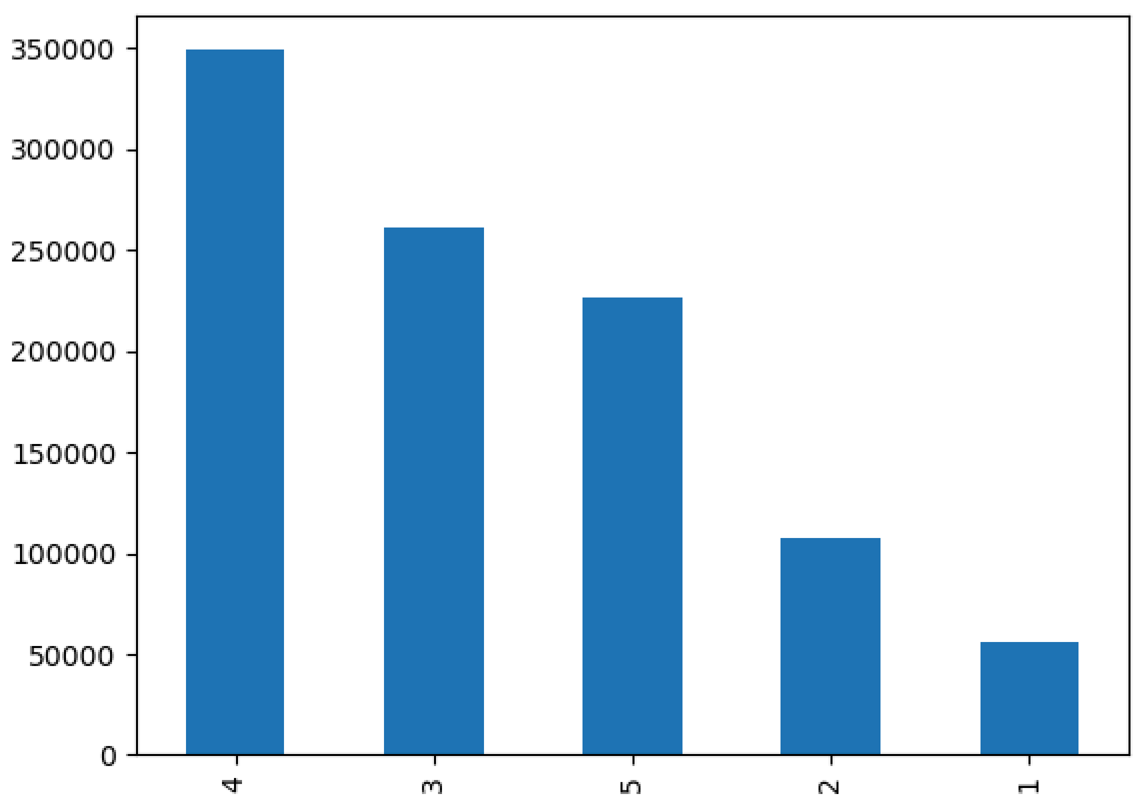
	MovieID	Title	Genres
count	3883.000000	3883	3883
unique	NaN	3883	301
top	NaN	Toy Story (1995)	Drama
freq	NaN	1	843
mean	1986.049446	NaN	NaN
std	1146.778349	NaN	NaN
min	1.000000	NaN	NaN
25%	982.500000	NaN	NaN
50%	2010.000000	NaN	NaN
75%	2980.500000	NaN	NaN
max	3952.000000	NaN	NaN

Distribution of Rating

In [41]:

```
rating['Rating'].value_counts().plot(kind='bar')
```

Out[41]: <Axes: >



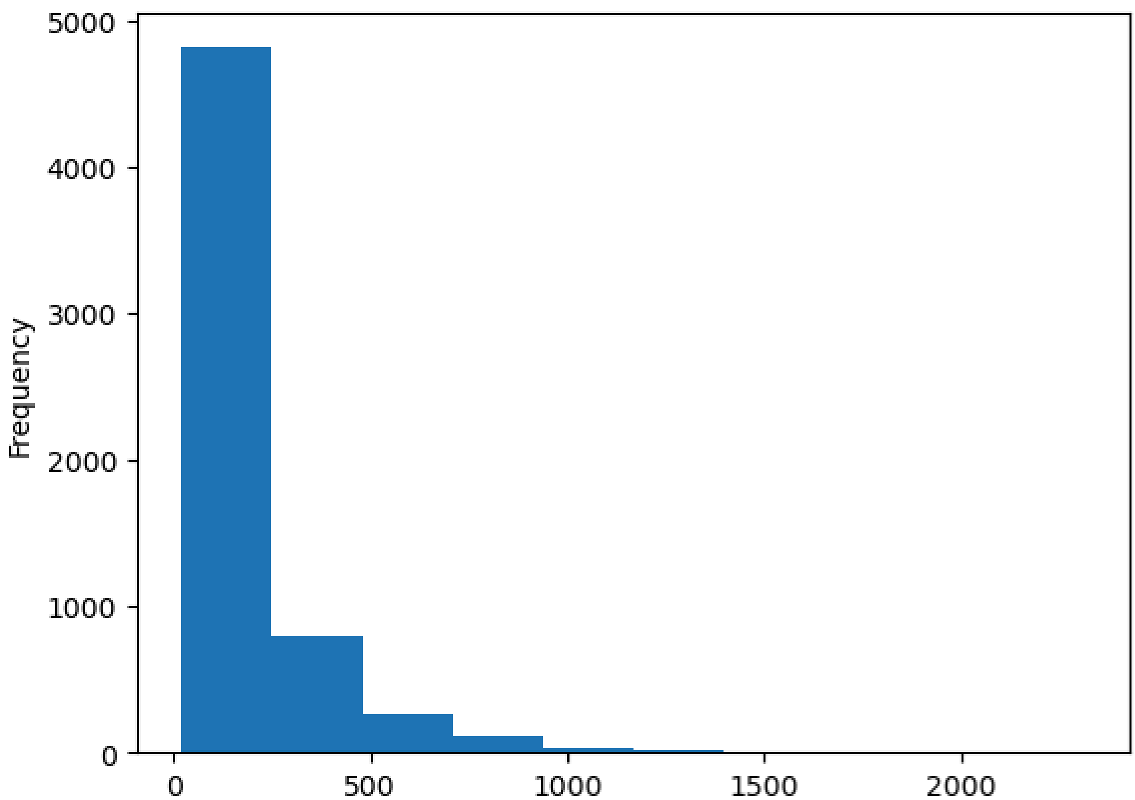
Number of Rating per user

In [42]:
```
rating['UserID'].value_counts()
```

Out[42]:
```
4169    2314
1680    1850
4277    1743
1941    1595
1181    1521
        ...
5725      20
3407      20
1664      20
4419      20
3021      20
Name: UserID, Length: 6040, dtype: int64
```

In [43]:
```
rating['UserID'].value_counts().plot(kind='hist', bins=10)
```

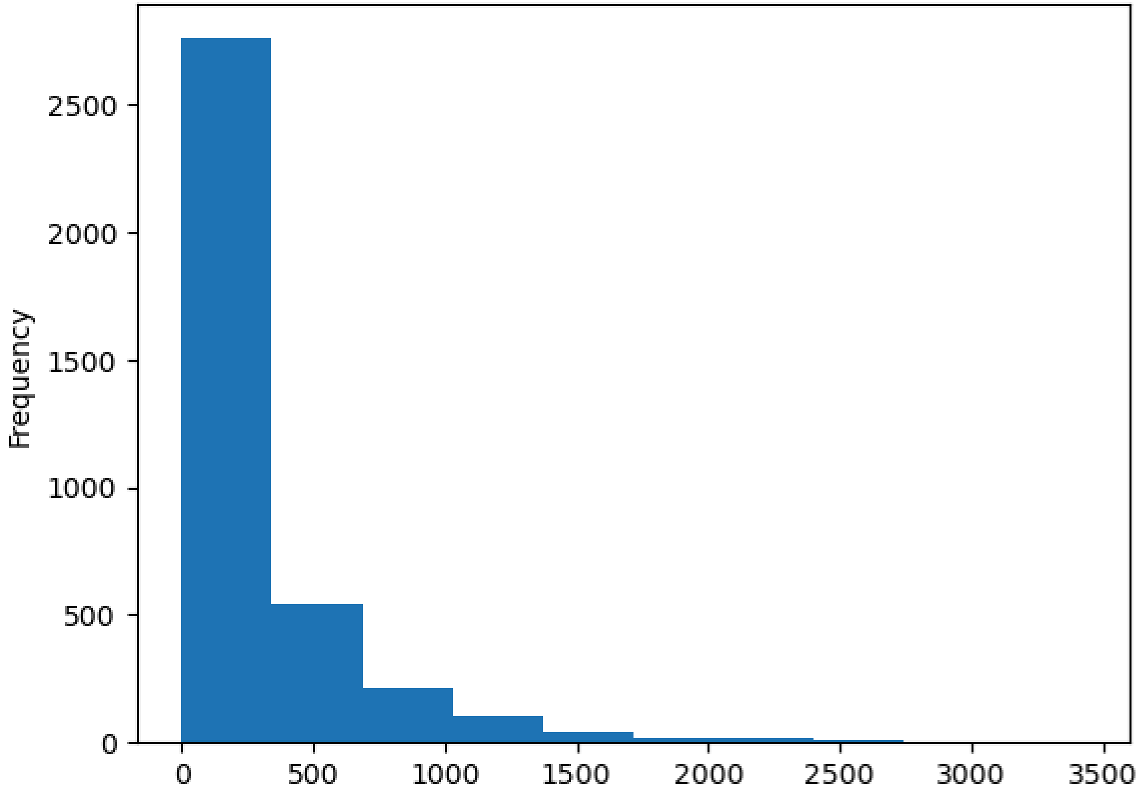
Out[43]:
```
<Axes: ylabel='Frequency'>
```



Number of ratings per movie:

In [44]:
```
rating['MovieID'].value_counts().plot(kind='hist', bins=10)
```

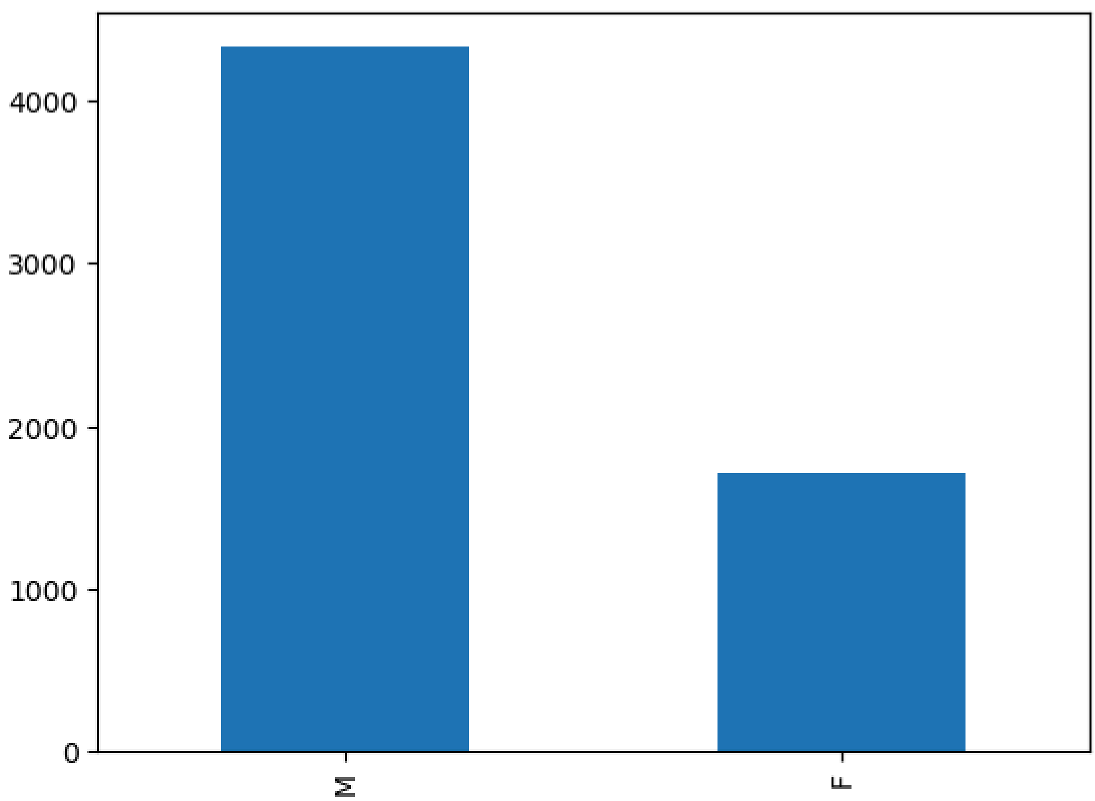
Out[44]:
```
<Axes: ylabel='Frequency'>
```



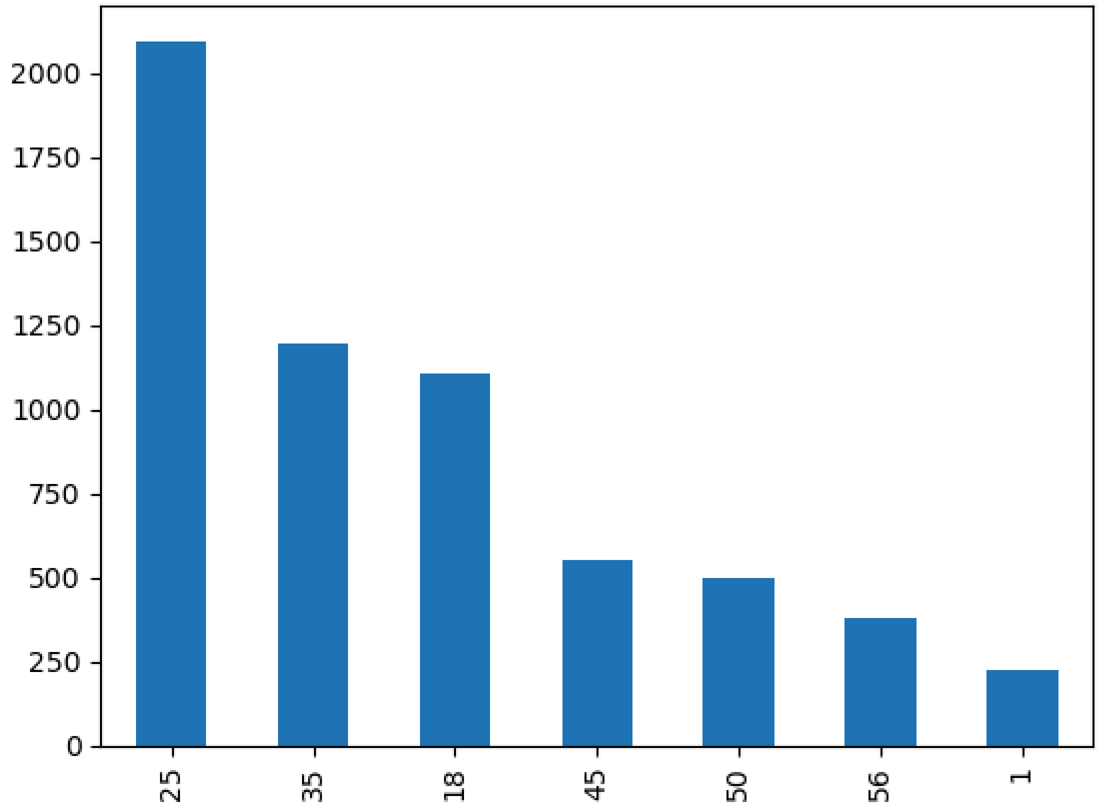
Explore User Demographic

In [45]:
```
user['Gender'].value_counts().plot(kind='bar')
```

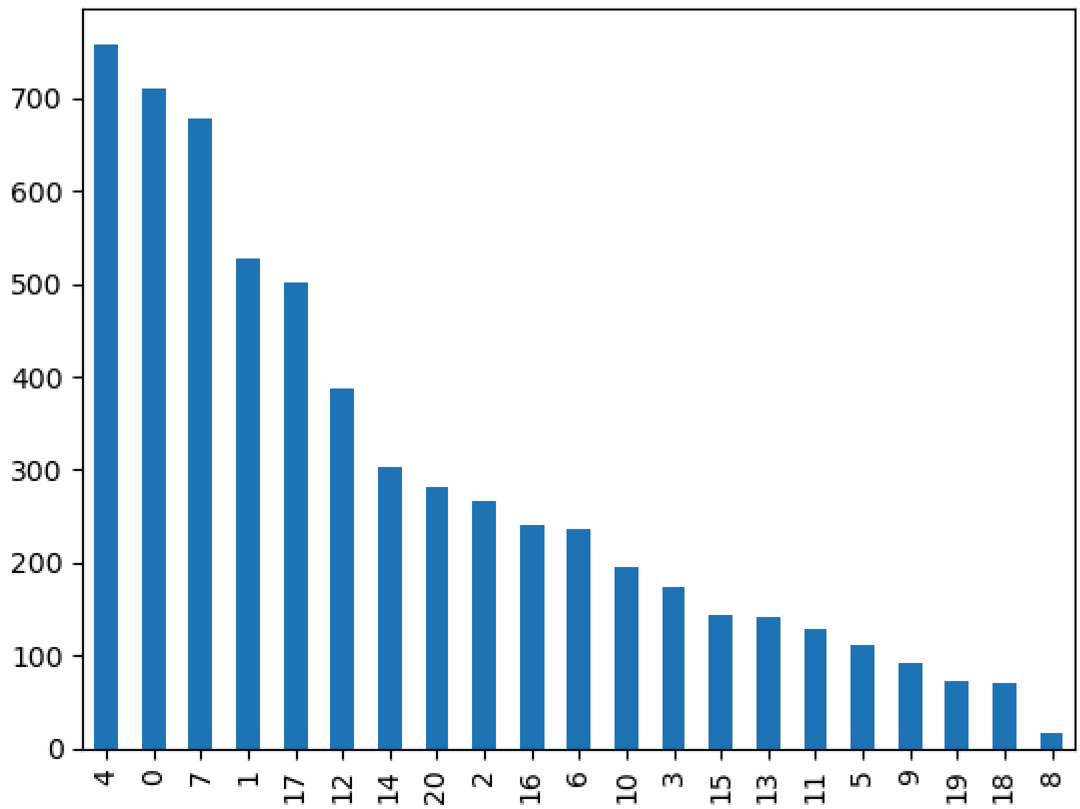
Out[45]: <Axes: >



In [46]:
```
user['Age'].value_counts().plot(kind='bar')
```
Out[46]: <Axes: >



In [47]:
```
user['Occupation'].value_counts().plot(kind='bar')
```
Out[47]: <Axes: >

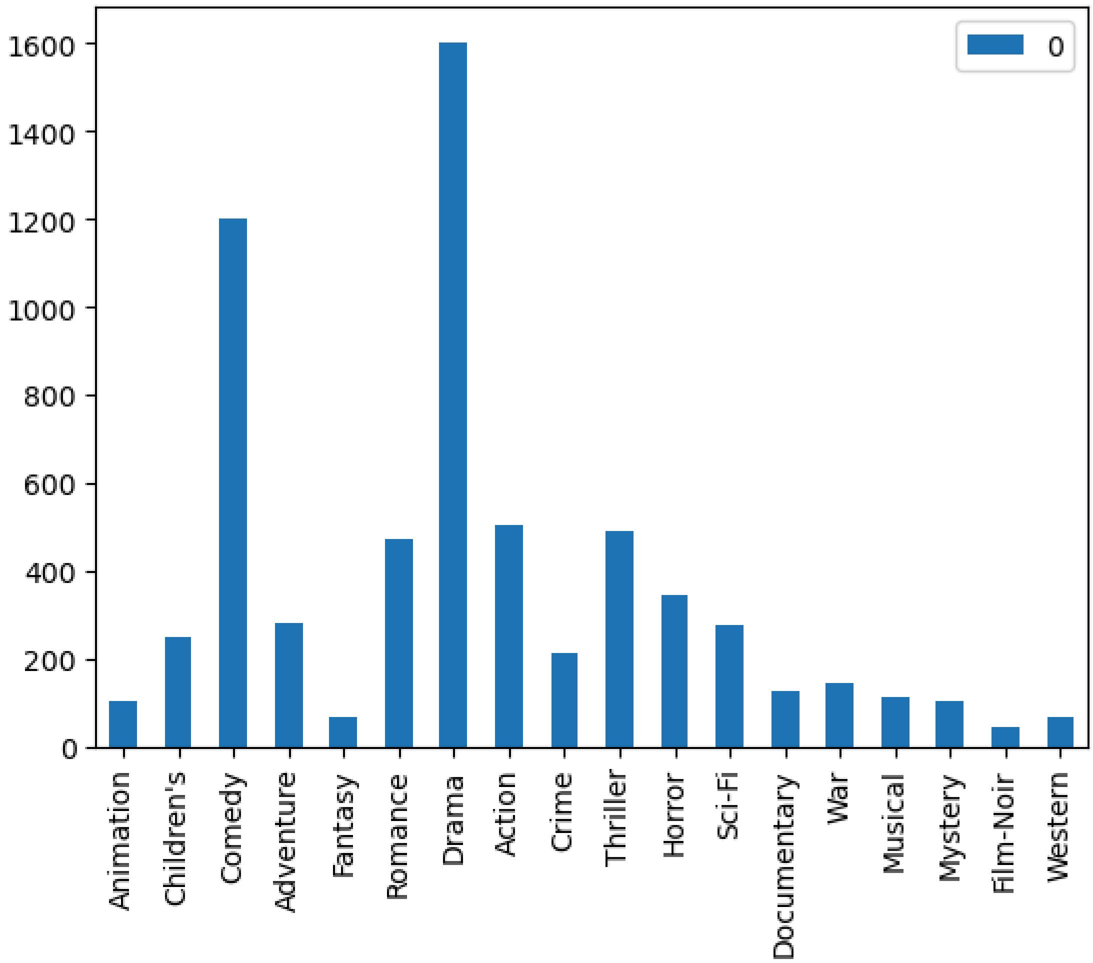


Most Popular Genre

In [48]:
```
from collections import Counter

genres = movies['Genres'].str.split('|').tolist()
genres = [genre for sublist in genres for genre in sublist]
genre_counts = Counter(genres)
pd.DataFrame.from_dict(genre_counts, orient='index').plot(kind='bar')
```

Out[48]: <Axes: >



In [49]:
```python
gender_age_mean = user.groupby(["Gender"])["Age"].mean()
```

In [50]:
```python
gender_age_mean
```

Out[50]:
```
Gender
F    30.859567
M    30.552297
Name: Age, dtype: float64
```

In [51]:
```python
# Group by Gender and Age and count the number of entries
gender_age_distribution = user.groupby(['Gender', 'Age']).size().unstack()

# Display the result
print(gender_age_distribution)
```

```
Age       1   18    25   35   45   50   56
Gender
F        78  298   558  338  189  146  102
M       144  805  1538  855  361  350  278
```

Data Transformation

In [52]:
```python
# Encode categorical data
from sklearn.preprocessing import LabelEncoder
```

```
le_gender = LabelEncoder()
user['Gender'] = le_gender.fit_transform(user['Gender'])
```

In [53]:
```
user['Gender']
```

Out[53]:
```
0       0
1       1
2       1
3       1
4       1
       ..
6035    0
6036    0
6037    0
6038    0
6039    1
Name: Gender, Length: 6040, dtype: int32
```

Create User-Item Matrix

In [54]:
```
user_item_matrix = rating.pivot(index='UserID', columns='MovieID', values='Rating').fi
```

In [55]:
```
user_item_matrix
```

Out[55]:

MovieID	1	2	3	4	5	6	7	8	9	10	...	3943	3944	3945	3946	3947	3948	394
UserID																		
1	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	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	0.0	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	0.0	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	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6036	0.0	0.0	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
6037	0.0	0.0	0.0	0.0	0.0	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	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
6040	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

6040 rows × 3706 columns

Feature Engineering

In [56]:
```
rating.columns
```

Out[56]:
```
Index(['UserID', 'MovieID', 'Rating', 'Timestamp'], dtype='object')
```

In [57]:
```
user['AverageRating'] = rating.groupby('UserID')['Rating'].mean()
user['AverageRating']
```

Out[57]:
```
0            NaN
1       4.188679
2       3.713178
3       3.901961
4       4.190476
          ...
6035    2.610714
6036    3.302928
6037    3.717822
6038    3.800000
6039    3.878049
Name: AverageRating, Length: 6040, dtype: float64
```

In [58]:
```
user['RatingCount'] = rating.groupby('UserID')['Rating'].count()
user['RatingCount']
```

Out[58]:
```
0         NaN
1        53.0
2       129.0
3        51.0
4        21.0
        ...
6035    280.0
6036    888.0
6037    202.0
6038     20.0
6039    123.0
Name: RatingCount, Length: 6040, dtype: float64
```

In [59]:
```
movies['AverageRating'] = rating.groupby('MovieID')['Rating'].mean()
movies['AverageRating']
```

Out[59]:
```
0            NaN
1       4.146846
2       3.201141
3       3.016736
4       2.729412
          ...
3878    2.833333
3879    2.784722
3880    3.500000
3881    5.000000
3882    3.273504
Name: AverageRating, Length: 3883, dtype: float64
```

In [60]:
```
movies['RatingCount'] = rating.groupby('MovieID')['Rating'].count()
movies['RatingCount']
```

Out[60]:
```
0          NaN
1       2077.0
2        701.0
3        478.0
4        170.0
         ...
3878      12.0
3879     144.0
3880      18.0
3881       1.0
3882     234.0
Name: RatingCount, Length: 3883, dtype: float64
```

In [61]:
```
rating['AverageRating'] = rating.groupby('MovieID')['Rating'].mean()
rating["AverageRating"]
```

Out[61]:
```
0               NaN
1          4.146846
2          3.201141
3          3.016736
4          2.729412
             ...
1000204         NaN
1000205         NaN
1000206         NaN
1000207         NaN
1000208         NaN
Name: AverageRating, Length: 1000209, dtype: float64
```

Handling Sparse Data

Sparsity is a measure of how many elements in a matrix are zero compared to the total number of elements. Sparsity=1− (Total Number of Possible Ratings/ Number of Actual Ratings)

Method 1 (User-Item Matrix): This method might be more intuitive for those who prefer working with matrices, especially if they are familiar with collaborative filtering techniques that rely on matrix representations.

In [62]:
```
# Count the total number of possible ratings
num_users = rating['UserID'].nunique()
num_movies = rating['MovieID'].nunique()
total_possible_ratings = num_users * num_movies
total_possible_ratings
```

Out[62]:
```
22384240
```

In [63]:
```
num_users
```

Out[63]:
```
6040
```

In [64]:
```
num_movies
```

Out[64]:
```
3706
```

In [65]:
```
# Count the number of actual ratings
num_actual_ratings = len(rating)
```

```
num_actual_ratings
```

Out[65]:

```
1000209
```

In [66]:

```
# Calculate sparsity
sparsity = 1 - (num_actual_ratings / total_possible_ratings)
print(f'Sparsity: {sparsity:.4f}')
```

```
Sparsity: 0.9553
```

95.5% of the values are filled

Method 2 (Raw Rating Data): This is simpler and faster to implement as it does not require the creation of a user-item matrix. It directly calculates sparsity from the raw data.

In [67]:

```
len(user)
```

Out[67]:

```
6040
```

In [68]:

```
len(movies)
```

Out[68]:

```
3883
```

In [69]:

```
# Evaluate sparsity using raw data
sparsity = 1.0 - (len(rating) / (len(user) * len(movies)))
print(f'Sparsity: {sparsity}')
```

```
Sparsity: 0.9573532020200125
```

The output Sparsity: 0.9573532020200125 indicates that approximately 95.74% of the entries in the user-item rating matrix are missing, meaning only about 4.26% of the possible ratings are actually provided.

method 1 is preferred where sparcity = 95.5%

In [70]:

```
Provided_value = (user_item_matrix > 0).sum().sum() / (user_item_matrix.shape[0] * use
Provided_value
```

Out[70]:

```
0.044683625622312845
```

Model Building

Collaborative Filtering with Pearson Correlation

In [71]:

```
user_item_matrix
```

Out[71]:

MovieID	1	2	3	4	5	6	7	8	9	10	...	3943	3944	3945	3946	3947	3948	394
UserID																		
1	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	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	0.0	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	0.0	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	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6036	0.0	0.0	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
6037	0.0	0.0	0.0	0.0	0.0	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	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
6040	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

6040 rows × 3706 columns

In [72]:

```
# Calculate Pearson Correlation between movies
subset_size = 500
movie_subset = user_item_matrix.columns[:subset_size]
movie_subset
```

Out[72]:

```
Int64Index([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
            ...
            504, 505, 506, 507, 508, 509, 510, 511, 512, 513],
           dtype='int64', name='MovieID', length=500)
```

In [73]:

```
movie_subset.shape
```

Out[73]:

```
(500,)
```

In [74]:

```
user_item_matrix[movie_subset]
```

Out[74]:

MovieID	1	2	3	4	5	6	7	8	9	10	...	504	505	506	507	508	509	510	511
UserID																			
1	5.0	0.0	0.0	0.0	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	0.0	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	0.0	0.0	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	0.0	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	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6036	0.0	0.0	0.0	2.0	0.0	3.0	0.0	0.0	0.0	0.0	...	3.0	0.0	0.0	0.0	3.0	2.0	0.0	0.0
6037	0.0	0.0	0.0	0.0	0.0	0.0	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	0.0	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	0.0	0.0	0.0	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	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	4.0	0.0	0.0	0.0

6040 rows × 500 columns

In [75]:
```
user_item_matrix[movie_subset].shape
```
Out[75]:
```
(6040, 500)
```

In [76]:
```
user_item_subset = user_item_matrix[movie_subset]
```

In [77]:
```
user_item_subset
```

Out[77]:

MovieID	1	2	3	4	5	6	7	8	9	10	...	504	505	506	507	508	509	510	511
UserID																			
1	5.0	0.0	0.0	0.0	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	0.0	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	0.0	0.0	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	0.0	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	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	0.0	4.0	0.0	0.0	4.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6036	0.0	0.0	0.0	2.0	0.0	3.0	0.0	0.0	0.0	0.0	...	3.0	0.0	0.0	0.0	3.0	2.0	0.0	0.0
6037	0.0	0.0	0.0	0.0	0.0	0.0	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	0.0	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	0.0	0.0	0.0	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	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	4.0	0.0	0.0	0.0

6040 rows × 500 columns

In [78]:

```
user_item_subset.shape
```

Out[78]:

```
(6040, 500)
```

In [79]:

```
# Compute Pearson Correlation for the subset of movies
movie_correlation_subset = user_item_subset.corr(method='pearson')
print(movie_correlation_subset)
```

```
MovieID       1         2         3         4         5         6         7    \
MovieID
1        1.000000  0.262649  0.146536  0.109375  0.170156  0.168087  0.189333
2        0.262649  1.000000  0.169586  0.111616  0.196561  0.137334  0.193658
3        0.146536  0.169586  1.000000  0.158659  0.268062  0.095834  0.238816
4        0.109375  0.111616  0.158659  1.000000  0.247529  0.071081  0.187311
5        0.170156  0.196561  0.268062  0.247529  1.000000  0.075012  0.264749
...           ...       ...       ...       ...       ...       ...       ...
509      0.149418  0.116426  0.110253  0.170077  0.083304  0.120363  0.204865
510      0.077278  0.076719  0.058421  0.168036  0.107638  0.051925  0.059553
511      0.080490  0.103092  0.127567  0.123559  0.093990  0.220918  0.079641
512      0.069834  0.160942  0.060295  0.043035  0.047113  0.065750  0.056838
513      0.109867  0.110627  0.087076  0.049162  0.145269  0.071254  0.118206

MovieID       8         9         10   ...       504       505       506  \
MovieID                                ...
1        0.082963  0.045705  0.215653  ...  0.082056  0.096585  0.084476
2        0.173878  0.126871  0.302042  ...  0.143829  0.132751  0.054171
3        0.068058  0.100622  0.164252  ...  0.117686  0.168127  0.033802
4        0.034228  0.042133  0.082035  ...  0.053294  0.118711  0.150781
5        0.076378  0.116465  0.176002  ...  0.073087  0.204517  0.055823
...           ...       ...       ...  ...       ...       ...       ...
509      0.025714  0.015006  0.079755  ...  0.030656  0.061548  0.279639
510      0.033634  0.097166  0.076089  ...  0.032127  0.094643  0.126304
511      0.047101  0.241162  0.206189  ...  0.232623  0.081457  0.003130
512      0.056551  0.048121  0.103457  ...  0.203123  0.096846  0.031003
513      0.066657  0.073168  0.118230  ...  0.106313  0.132301  0.077414

MovieID       507       508       509       510       511       512       513
MovieID
1        0.144312  0.182004  0.149418  0.077278  0.080490  0.069834  0.109867
2        0.161690  0.171937  0.116426  0.076719  0.103092  0.160942  0.110627
3        0.118375  0.130343  0.110253  0.058421  0.127567  0.060295  0.087076
4        0.090599  0.199829  0.170077  0.168036  0.123559  0.043035  0.049162
5        0.106318  0.149238  0.083304  0.107638  0.093990  0.047113  0.145269
...           ...       ...       ...       ...       ...       ...       ...
509      0.140857  0.242534  1.000000  0.060742  0.019447  0.033509  0.086185
510      0.079529  0.144560  0.060742  1.000000  0.101591  0.049203  0.052940
511      0.190436  0.084706  0.019447  0.101591  1.000000  0.064946  0.041915
512      0.066209  0.040578  0.033509  0.049203  0.064946  1.000000  0.091932
513      0.086019  0.052198  0.086185  0.052940  0.041915  0.091932  1.000000

[500 rows x 500 columns]
```

In [80]:

```python
# Define the recommendation function
def recommend_movies_pearson(user_id, user_item_matrix, movie_correlation_subset, n_re
    user_ratings = user_item_matrix.loc[user_id]
    rated_movies = user_ratings[user_ratings > 0].index

    if rated_movies.empty:
        return pd.Series(dtype='float64')

    # Calculate weighted ratings based on Pearson Correlation
    recommendations = pd.Series(dtype='float64')
    for movie in rated_movies:
        if movie in movie_correlation_subset.columns:
            similar_movies = movie_correlation_subset[movie].drop(movie, errors='ignor
            similar_movies = similar_movies[similar_movies > 0]
            if not similar_movies.empty:
                for similar_movie, correlation in similar_movies.items():
```

```
                    if similar_movie not in rated_movies and similar_movie in user_ite
                        recommendations[similar_movie] = recommendations.get(similar_m

    recommendations = recommendations.sort_values(ascending=False)
    return recommendations.head(n_recommendations)
```

Additional Analysis and Insights

In [81]:
```
user.head()
```

Out[81]:

	UserID	Gender	Age	Occupation	Zip-code	AverageRating	RatingCount
0	1	0	1	10	48067	NaN	NaN
1	2	1	56	16	70072	4.188679	53.0
2	3	1	25	15	55117	3.713178	129.0
3	4	1	45	7	02460	3.901961	51.0
4	5	1	25	20	55455	4.190476	21.0

In [82]:
```
rating.head()
```

Out[82]:

	UserID	MovieID	Rating	Timestamp	AverageRating
0	1	1193	5	2000-12-31 22:12:40	NaN
1	1	661	3	2000-12-31 22:35:09	4.146846
2	1	914	3	2000-12-31 22:32:48	3.201141
3	1	3408	4	2000-12-31 22:04:35	3.016736
4	1	2355	5	2001-01-06 23:38:11	2.729412

In [83]:
```
movies.head()
```

Out[83]:

	MovieID	Title	Genres	AverageRating	RatingCount
0	1	Toy Story (1995)	Animation|Children's|Comedy	NaN	NaN
1	2	Jumanji (1995)	Adventure|Children's|Fantasy	4.146846	2077.0
2	3	Grumpier Old Men (1995)	Comedy|Romance	3.201141	701.0
3	4	Waiting to Exhale (1995)	Comedy|Drama	3.016736	478.0
4	5	Father of the Bride Part II (1995)	Comedy	2.729412	170.0

In [84]:
```
merged_user_rating = user.merge(rating, on='UserID')
merged_user_rating
```

Out[84]:

	UserID	Gender	Age	Occupation	Zip-code	AverageRating_x	RatingCount	MovieID	Rating
0	1	0	1	10	48067	NaN	NaN	1193	5
1	1	0	1	10	48067	NaN	NaN	661	3
2	1	0	1	10	48067	NaN	NaN	914	3
3	1	0	1	10	48067	NaN	NaN	3408	4
4	1	0	1	10	48067	NaN	NaN	2355	5
...	...	...	...	...	...	...	...	...	...
1000204	6040	1	25	6	11106	3.878049	123.0	1091	1
1000205	6040	1	25	6	11106	3.878049	123.0	1094	5
1000206	6040	1	25	6	11106	3.878049	123.0	562	5
1000207	6040	1	25	6	11106	3.878049	123.0	1096	4
1000208	6040	1	25	6	11106	3.878049	123.0	1097	4

1000209 rows × 11 columns

In [85]:

```
# Most of the users in our dataset who’ve rated the movies are Male.
# 1 for Men
# 0 for women
gender_group_count = user.merge(rating, on="UserID")["Gender"].value_counts(normalize
gender_group_count
```

Out[85]:

```
1    0.753611
0    0.246389
Name: Gender, dtype: float64
```

In [86]:

```
#Users of which age group have watched and rated the most number of movies?

age_group_counts = rating.merge(user, on='UserID')['Age'].value_counts()
print("Age group with the most ratings:", age_group_counts.idxmax())
```

```
Age group with the most ratings: 25
```

In [87]:

```
# Users belonging to which profession have watched and rated the most movies?
occupation_counts = rating.merge(user, on= "UserID")["Occupation"].value_counts()
print("Top 5 Occupation with the most ratings and there counts:", occupation_counts[:5
```

```
Top 5 Occupation with the most ratings and there counts: 4     131032
0     130499
7     105425
1      85351
17     72816
Name: Occupation, dtype: int64
```

In [88]:
```
print("Occupation with the most ratings:", occupation_counts.idxmax())
```

```
Occupation with the most ratings: 4
```

In [89]:
```
# The movie with the maximum no. of ratings is
rating["MovieID"].value_counts()
```

Out[89]:
```
2858    3428
260     2991
1196    2990
1210    2883
480     2672
        ...
3458       1
2226       1
1815       1
398        1
2909       1
Name: MovieID, Length: 3706, dtype: int64
```

In [90]:
```
movie_details = movies[movies["MovieID"] == 2858]
movie_details
```

Out[90]:

	MovieID	Title	Genres	AverageRating	RatingCount
2789	2858	American Beauty (1999)	Comedy|Drama	2.993939	165.0

In [91]:
```
rating_details = rating[rating["MovieID"] == 2858]
rating_details
```

Out[91]:

	UserID	MovieID	Rating	Timestamp	AverageRating
105	2	2858	4	2000-12-31 21:33:54	3.232558
202	3	2858	4	2000-12-31 21:10:39	2.500000
299	5	2858	4	2000-12-31 05:43:10	3.664336
471	6	2858	1	2000-12-31 04:26:49	3.631052
585	8	2858	5	2000-12-31 02:30:17	2.906699
...	...	...	...	...	...
996998	6019	2858	5	2000-04-26 14:49:12	NaN
997895	6027	2858	3	2000-04-26 05:25:36	NaN
998845	6036	2858	5	2000-04-26 00:37:33	NaN
999571	6037	2858	4	2000-04-26 00:33:35	NaN
999938	6040	2858	4	2000-04-25 23:14:35	NaN

3428 rows × 5 columns

In [92]:

```
age_group_counts = user.merge(rating, on='UserID')['Age'].value_counts()
age_group_counts
```

Out[92]:

```
25    395556
35    199003
18    183536
45     83633
50     72490
56     38780
1      27211
Name: Age, dtype: int64
```

In [93]:

```
# counting the number of ratings each movie title has received
title_counts = movies.merge(rating, on='MovieID')['Title'].value_counts()
title_counts
```

Out[93]:

```
American Beauty (1999)                                   3428
Star Wars: Episode IV - A New Hope (1977)                2991
Star Wars: Episode V - The Empire Strikes Back (1980)    2990
Star Wars: Episode VI - Return of the Jedi (1983)        2883
Jurassic Park (1993)                                     2672
                                                         ...
Kestrel's Eye (Falkens öga) (1998)                          1
Last of the High Kings, The (a.k.a. Summer Fling) (1996)    1
Condition Red (1995)                                        1
Beauty (1998)                                               1
Soft Toilet Seats (1999)                                    1
Name: Title, Length: 3706, dtype: int64
```

In [94]:

```
# Count the Occurrences of Each Rating for Each Movie Title
merged_df = movies.merge(rating, on='MovieID')
title_rating_count = merged_df.groupby(['Title', 'Rating']).size().unstack(fill_value=
title_rating_count
```

Out[94]:

Rating	1	2	3	4	5
Title					
$1,000,000 Duck (1971)	3	8	15	7	4
'Night Mother (1986)	4	10	25	18	13
'Til There Was You (1997)	5	20	15	10	2
'burbs, The (1989)	36	69	107	68	23
...And Justice for All (1979)	2	12	65	82	38
...	...	...	...	...	...
Zed & Two Noughts, A (1985)	2	3	8	13	3
Zero Effect (1998)	7	32	72	108	82
Zero Kelvin (Kjærlighetens kjøtere) (1995)	0	0	1	1	0
Zeus and Roxanne (1997)	5	6	8	3	1
eXistenZ (1999)	43	61	109	142	55

3706 rows × 5 columns

In [95]:

```
# Get the counts of rating 5 for each movie
rating_5_counts = title_rating_count[5]
rating_5_counts
```

Out[95]:

```
Title
$1,000,000 Duck (1971)                         4
'Night Mother (1986)                          13
'Til There Was You (1997)                      2
'burbs, The (1989)                            23
...And Justice for All (1979)                 38
                                              ..
Zed & Two Noughts, A (1985)                    3
Zero Effect (1998)                            82
Zero Kelvin (Kjærlighetens kjøtere) (1995)     0
Zeus and Roxanne (1997)                        1
eXistenZ (1999)                               55
Name: 5, Length: 3706, dtype: int64
```

In [96]:

```
max_ratings = title_rating_count.max(axis=1)
max_ratings
```

Out[96]:

```
Title
$1,000,000 Duck (1971)                         15
'Night Mother (1986)                           25
'Til There Was You (1997)                      20
'burbs, The (1989)                            107
...And Justice for All (1979)                  82
                                             ...
Zed & Two Noughts, A (1985)                    13
Zero Effect (1998)                            108
Zero Kelvin (Kjærlighetens kjøtere) (1995)      1
Zeus and Roxanne (1997)                         8
eXistenZ (1999)                               142
Length: 3706, dtype: int64
```

In [97]:
```
max_ratings = title_rating_count.max(axis=0)
max_ratings
```

Out[97]:
```
Rating
1     314
2     324
3     683
4    1122
5    1963
dtype: int64
```

In [98]:
```
# Movies count as per rating
# Max Number of movies received 4*
rating.groupby("Rating")["MovieID"].count()
```

Out[98]:
```
Rating
1     56174
2    107557
3    261197
4    348971
5    226310
Name: MovieID, dtype: int64
```

In [99]:
```
movie_counts = rating['MovieID'].value_counts()
max_rated_movie_id = movie_counts.idxmax()
max_rated_movie_title = movies[movies['MovieID'] == max_rated_movie_id]['Title'].value
print("Movie with the most ratings:", max_rated_movie_title)
```

```
Movie with the most ratings: American Beauty (1999)
```

In [100…
```
# Movies count as per Genres
movies_count_Genre= movies.groupby("Genres")["Title"].count()
movies_count_Genre
```

Out[100]:
```
Genres
Action                                          65
Action|Adventure                                25
Action|Adventure|Animation                       1
Action|Adventure|Animation|Children's|Fantasy    1
Action|Adventure|Animation|Horror|Sci-Fi         1
                                                ...
Sci-Fi|Thriller|War                              1
Sci-Fi|War                                       1
Thriller                                       101
War                                             12
Western                                         33
Name: Title, Length: 301, dtype: int64
```

In [101…
```
movies_count_Genre.max()
```

Out[101]:
```
843
```

In [102…
```
movies_count_Genre= movies.groupby("Genres")["Title"].sum()
movies_count_Genre
```

Out[102]:
```
Genres
Action                                          Sudden Death (1995)Money Train (199
5)Fair Game...
Action|Adventure                                Mortal Kombat (1995)Waterworld (199
5)Good Man ...
Action|Adventure|Animation                          Princess Mononoke, The (Mononoke
Hime) (1997)
Action|Adventure|Animation|Children's|Fantasy                              Pagemaste
r, The (1994)
Action|Adventure|Animation|Horror|Sci-Fi                                       Heavy
Metal (1981)
                                                                          ...
Sci-Fi|Thriller|War
Them! (1954)
Sci-Fi|War                                      Dr. Strangelove or: How I Learned to
Stop Worr...
Thriller                                        Four Rooms (1995)Assassins (1995)Unf
orgettable...
War                                             Land and Freedom (Tierra y libertad)
(1995)Und...
Western                                         Wild Bill (1995)Wyatt Earp (1994)Bad
Girls (19...
Name: Title, Length: 301, dtype: object
```

Regex Pattern Explanation r: This indicates a raw string in Python. A raw string treats backslashes () as literal characters, which is helpful when writing regex patterns. (: This matches an opening parenthesis (. The backslash \ is used to escape the parenthesis, indicating that we are looking for the literal character (. (\d{4}): This is a capturing group that matches exactly four digits. \d: This matches any digit (equivalent to [0-9]). {4}: This specifies that we are looking for exactly four of the preceding element (digits in this case).): This matches a closing parenthesis). The backslash \ is used to escape the parenthesis, indicating that we are looking for the literal character).

In [103…
```python
# Most of the movies present in our dataset were released in which decade?

movies['Year'] = movies['Title'].str.extract(r'\((\d{4})\)').astype(int)
movies['Year']
```

Out[103]:
```
0       1995
1       1995
2       1995
3       1995
4       1995
        ...
3878    2000
3879    2000
3880    2000
3881    2000
3882    2000
Name: Year, Length: 3883, dtype: int32
```

In [104…
```python
# Group the movies by decade and count the number of movies in each decade
movies['Decade'] = (movies['Year'] // 10) * 10
decade_counts = movies['Decade'].value_counts().sort_index()
decade_counts
```

Out[104]:
```
1910       3
1920      34
1930      77
1940     126
1950     168
1960     191
1970     247
1980     598
1990    2283
2000     156
Name: Decade, dtype: int64
```

In [105…
```
# Determine the decade with the most movies
most_common_decade = decade_counts.idxmax()
print(f"The decade with the most movies is the {most_common_decade}s.")
```

```
The decade with the most movies is the 1990s.
```

Collaborative Filtering with Cosine Similarity

In [106…
```
user_item_matrix
```

Out[106]:

MovieID UserID	1	2	3	4	5	6	7	8	9	10	...	3943	3944	3945	3946	3947	3948	394
1	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
2	0.0	0.0	0.0	0.0	0.0	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	0.0	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	0.0	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	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
6036	0.0	0.0	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
6037	0.0	0.0	0.0	0.0	0.0	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	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
6040	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

6040 rows × 3706 columns

User-User Based Cosine Similarity

In [107…
```
from sklearn.metrics.pairwise import cosine_similarity

# Calculate cosine similarity between users
```

```
user_similarity = cosine_similarity(user_item_matrix)
user_similarity_df = pd.DataFrame(user_similarity, index=user_item_matrix.index, colum
```

In [108…

```
user_similarity_df
```

Out[108]:

UserID	1	2	3	4	5	6	7	8	9	
UserID										
1	1.000000	0.096382	0.120610	0.132455	0.090158	0.179222	0.059678	0.138241	0.226148	0.25
2	0.096382	1.000000	0.151479	0.171176	0.114394	0.100865	0.305787	0.203337	0.190198	0.22
3	0.120610	0.151479	1.000000	0.151227	0.062907	0.074603	0.138332	0.077656	0.126457	0.21
4	0.132455	0.171176	0.151227	1.000000	0.045094	0.013529	0.130339	0.100856	0.093651	0.12
5	0.090158	0.114394	0.062907	0.045094	1.000000	0.047449	0.126257	0.220817	0.261330	0.11
...	...	...	...	...	...	...	...	...	...	
6036	0.186329	0.228241	0.143264	0.170583	0.293365	0.093583	0.122441	0.227400	0.239607	0.33
6037	0.135979	0.206274	0.107744	0.127464	0.172686	0.065788	0.111673	0.144395	0.225055	0.24
6038	0.000000	0.066118	0.120234	0.062907	0.020459	0.065711	0.000000	0.019242	0.093470	0.11
6039	0.174604	0.066457	0.094675	0.064634	0.027689	0.167303	0.014977	0.044660	0.046434	0.29
6040	0.133590	0.218276	0.133144	0.137968	0.241437	0.083436	0.080680	0.148123	0.215819	0.25

6040 rows × 6040 columns

Generate recommendations (based on user similarity)

In [109…

```
import numpy as np

# Function to recommend items for a given user
def recommend_items(user_id, user_item_matrix, user_similarity_df, top_n=2):
    similar_users = user_similarity_df[user_id].sort_values(ascending=False).index[1:]
    similar_users_ratings = user_item_matrix.loc[similar_users]

    # Compute the weighted average of ratings for each item
    weighted_ratings = similar_users_ratings.T.dot(user_similarity_df[user_id].loc[sim
    weighted_ratings = weighted_ratings / np.array([np.abs(user_similarity_df[user_id]

    # Remove items already rated by the user
    user_ratings = user_item_matrix.loc[user_id]
    weighted_ratings = weighted_ratings[user_ratings[user_ratings == 0].index]

    # Recommend the top N items
    recommendations = weighted_ratings.sort_values(ascending=False).head(top_n)

    return recommendations

# Get recommendations for a specific user (e.g., user_id=1)
```

```
recommendations = recommend_items(user_id=1, user_item_matrix=user_item_matrix, user_s

print(recommendations)
```

```
MovieID
2858    2.660032
1196    2.566771
dtype: float64
```

Item-Item Based Cosine Similarty

In [110…

```
# Transpose the user-item matrix to create an item-user matrix
item_user_matrix = user_item_matrix.T
item_user_matrix
```

Out[110]:

UserID	1	2	3	4	5	6	7	8	9	10	...	6031	6032	6033	6034	6035	6036	603
MovieID																		
1	5.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	5.0	5.0	...	0.0	4.0	0.0	0.0	4.0	0.0	0
2	0.0	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
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	1.0	0.0	0
4	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	2.0	2.0	0
5	0.0	0.0	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
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
3948	0.0	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
3949	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
3950	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
3951	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0
3952	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0

3706 rows × 6040 columns

In [111…

```
# Compute the cosine similarity between items
item_similarity = cosine_similarity(item_user_matrix)
item_similarity
```

Out[111]:
```
array([[1.        , 0.39034871, 0.26794263, ..., 0.09347942, 0.04282933,
        0.18269056],
       [0.39034871, 1.        , 0.24094645, ..., 0.08701306, 0.02606255,
        0.12218461],
       [0.26794263, 0.24094645, 1.        , ..., 0.0622576 , 0.01007255,
        0.097786  ],
       ...,
       [0.09347942, 0.08701306, 0.0622576 , ..., 1.        , 0.20280851,
        0.2346385 ],
       [0.04282933, 0.02606255, 0.01007255, ..., 0.20280851, 1.        ,
        0.19297221],
       [0.18269056, 0.12218461, 0.097786  , ..., 0.2346385 , 0.19297221,
        1.        ]])
```

In [112…
```
# Convert the similarity matrix to a DataFrame for better readability
item_similarity_df = pd.DataFrame(item_similarity, index=item_user_matrix.index, colum
item_similarity_df
```

Out[112]:

MovieID	1	2	3	4	5	6	7	8	9	
MovieID										
1	1.000000	0.390349	0.267943	0.178789	0.256569	0.347373	0.301490	0.125709	0.106620	0.3
2	0.390349	1.000000	0.240946	0.155457	0.249970	0.244827	0.262772	0.196521	0.158469	0.3
3	0.267943	0.240946	1.000000	0.192788	0.308290	0.187020	0.292230	0.092122	0.128378	0.2
4	0.178789	0.155457	0.192788	1.000000	0.271990	0.125170	0.220024	0.049554	0.060334	0.1
5	0.256569	0.249970	0.308290	0.271990	1.000000	0.148114	0.305107	0.095512	0.138392	0.2
...	...	...	...	...	...	...	...	...	...	
3948	0.309676	0.213650	0.190575	0.118902	0.174554	0.236447	0.191689	0.090387	0.092347	0.2
3949	0.186633	0.140781	0.104837	0.096318	0.092403	0.201419	0.117660	0.080523	0.099554	0.1
3950	0.093479	0.087013	0.062258	0.022588	0.051633	0.115331	0.059262	0.084976	0.004956	0.0
3951	0.042829	0.026063	0.010073	0.024769	0.010750	0.029136	0.036102	0.072141	0.000000	0.0
3952	0.182691	0.122185	0.097786	0.095154	0.112835	0.222836	0.138879	0.045523	0.057881	0.1

3706 rows × 3706 columns

Generate recommendations (based on item similarity)

In [113…
```
# Function to recommend items for a given user based on item similarity
def recommend_items_based_on_similarity(user_id, user_item_matrix, item_similarity_df,
    user_ratings = user_item_matrix.loc[user_id]
    rated_items = user_ratings[user_ratings > 0].index

    # Compute the weighted average of similarities for each item
    similarity_scores = item_similarity_df[rated_items].mean(axis=1)
```

```
    # Remove items already rated by the user
    similarity_scores = similarity_scores.drop(rated_items)

    # Recommend the top N items
    recommendations = similarity_scores.sort_values(ascending=False).head(top_n)

    return recommendations

# Get recommendations for a specific user (e.g., user_id=1)
recommendations = recommend_items_based_on_similarity(user_id=1, user_item_matrix=user

print(recommendations)
```

```
MovieID
1196    0.402209
364     0.390736
dtype: float64
```

Matrix Factorization

In [114…

```
!pip install cmfrec
```

```
Requirement already satisfied: cmfrec in c:\users\harsh\anaconda\lib\site-packages
(3.5.1.post8)
Requirement already satisfied: cython in c:\users\harsh\anaconda\lib\site-packages (f
rom cmfrec) (3.0.10)
Requirement already satisfied: numpy>=1.25 in c:\users\harsh\anaconda\lib\site-packag
es (from cmfrec) (1.26.4)
Requirement already satisfied: scipy in c:\users\harsh\anaconda\lib\site-packages (fr
om cmfrec) (1.10.1)
Requirement already satisfied: pandas in c:\users\harsh\anaconda\lib\site-packages (f
rom cmfrec) (1.5.3)
Requirement already satisfied: findblas in c:\users\harsh\anaconda\lib\site-packages
(from cmfrec) (0.1.26.post1)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\harsh\anaconda\lib
\site-packages (from pandas->cmfrec) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\harsh\anaconda\lib\site-packa
ges (from pandas->cmfrec) (2022.7)
Requirement already satisfied: six>=1.5 in c:\users\harsh\anaconda\lib\site-packages
(from python-dateutil>=2.8.1->pandas->cmfrec) (1.16.0)
```

In [115…

```
rm_raw = rating[['UserID', 'MovieID', 'Rating']].copy()
rm_raw.columns = ['UserId', 'ItemId', 'Rating']  # Lib requires specific column names
rm_raw.head(2)
```

Out[115]:

	UserId	ItemId	Rating
0	1	1193	5
1	1	661	3

In [116…

```
from cmfrec import CMF
```

In [117…

```
# Create a matrix factorization model
model = CMF(method="als", k=2, lambda_=0.1, user_bias=False, item_bias=False, verbose=
```

In [118…

```
# Fit the model
model.fit(rm_raw)
```

Out[118]:

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

In [119…

```
# Shape of User Matrix and Item matrix generated by CMF
model.A_.shape, model.B_.shape
```

Out[119]:

```
((6040, 2), (3706, 2))
```

In [124…

```
# Calculate RMSE
rm__ = np.dot(model.A_, model.B_.T) + model.glob_mean_
rmse_value = mse(user_item_matrix.values[user_item_matrix > 0], rm__[user_item_matrix
print(f"RMSE: {rmse_value}")
```

```
RMSE: 1.3043536471783002
```

In [125…

```
# Calculate MAPE
actuals = user_item_matrix.values[user_item_matrix > 0]
predictions = rm__[user_item_matrix > 0]
mape_value = np.mean(np.abs((actuals - predictions) / actuals)) * 100
print(f"MAPE: {mape_value}")
```

```
MAPE: 37.65643733664071
```

In [126…

```
#This is the mean (average) of all the ratings
#global mean rating calculated by the matrix factorization model
rm_raw.Rating.mean(), model.glob_mean_
```

Out[126]:

```
(3.581564453029317, 3.581564426422119)
```

In [121…

```
from sklearn.metrics import mean_squared_error as mse
```

In [122…

```
rm__ = np.dot(model.A_, model.B_.T) + model.glob_mean_
mse(user_item_matrix.values[user_item_matrix > 0], rm__[user_item_matrix > 0])**0.5
```

Out[122]:

```
1.3043536471783002
```

In [123…

```
# Making predictions
user_id = 1  # Example user ID
item_id = 1  # Example item ID

# Predict the rating for a specific user and item
predicted_rating = model.predict(user=user_id, item=item_id)
print(f"Predicted rating for user {user_id} and item {item_id}: {predicted_rating}")
```

```
Predicted rating for user 1 and item 1: 4.237185716629028
```

Thank-You