In [58]:
```
#DESCRIPTION

#The dataset provided contains movie reviews given by Amazon customers.
 Reviews were given between May 1996 and July 2014.

#Data Dictionary
#UserID – 4848 customers who provided a rating for each movie
#Movie 1 to Movie 206 – 206 movies for which ratings are provided by 48
48 distinct users

#Data Considerations
#- All the users have not watched all the movies and therefore, all mov
ies are not rated. These missing values are represented by NA.
#- Ratings are on a scale of -1 to 10 where -1 is the least rating and
 10 is the best.

#Analysis Task
#- Exploratory Data Analysis:

#Which movies have maximum views/ratings?
#What is the average rating for each movie? Define the top 5 movies wit
h the maximum ratings.
#Define the top 5 movies with the least audience.
#- Recommendation Model: Some of the movies hadn't been watched and the
refore, are not rated by the users. Netflix would like to take this as
 an opportunity and build a machine learning recommendation algorithm w
hich provides the ratings for each of the users.

#Divide the data into training and test data
#Build a recommendation model on training data
#Make predictions on the test data
```

In [59]:
```
# Data manipulation
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
```

```
# Visualization
import matplotlib.pyplot as plt
import seaborn as sns
# Set a few plotting defaults
%matplotlib inline
```

In [60]:
```
movie = pd.read_csv('Amazon - Movies and TV Ratings.csv')
movie.head()
```

Out[60]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie
0	A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	AH3QC2PC1VTGP	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	Na
2	A3LKP6WPMP9UKX	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	Na
3	AVIY68KEPQ5ZD	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	Na
4	A1CV1WROP5KTTW	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	Na

5 rows × 207 columns

In [61]:
```
movie.describe()
```

Out[61]:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	.
count	1.0	1.0	1.0	2.0	29.000000	1.0	1.0	1.0	1.0	1.0	.
mean	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	.
std	NaN	NaN	NaN	0.0	1.496301	NaN	NaN	NaN	NaN	NaN	.
min	5.0	5.0	2.0	5.0	1.000000	4.0	5.0	5.0	5.0	5.0	.
25%	5.0	5.0	2.0	5.0	4.000000	4.0	5.0	5.0	5.0	5.0	.
50%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	.
75%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	.
max	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	.

8 rows × 206 columns

In [62]:
```
movie.describe().transpose()
```

Out[62]:

	count	mean	std	min	25%	50%	75%	max
Movie1	1.0	5.000000	NaN	5.0	5.00	5.0	5.0	5.0
Movie2	1.0	5.000000	NaN	5.0	5.00	5.0	5.0	5.0
Movie3	1.0	2.000000	NaN	2.0	2.00	2.0	2.0	2.0
Movie4	2.0	5.000000	0.000000	5.0	5.00	5.0	5.0	5.0
Movie5	29.0	4.103448	1.496301	1.0	4.00	5.0	5.0	5.0
...	...	...	...	...	...	...	...	...
Movie202	6.0	4.333333	1.632993	1.0	5.00	5.0	5.0	5.0
Movie203	1.0	3.000000	NaN	3.0	3.00	3.0	3.0	3.0
Movie204	8.0	4.375000	1.407886	1.0	4.75	5.0	5.0	5.0
Movie205	35.0	4.628571	0.910259	1.0	5.00	5.0	5.0	5.0
Movie206	13.0	4.923077	0.277350	4.0	5.00	5.0	5.0	5.0

206 rows × 8 columns

In [63]:
```
movie.describe().transpose()['count']
```

Out[63]:
```
Movie1       1.0
Movie2       1.0
Movie3       1.0
Movie4       2.0
Movie5      29.0
            ...
Movie202     6.0
Movie203     1.0
Movie204     8.0
Movie205    35.0
```

```
Movie206    13.0
Name: count, Length: 206, dtype: float64
```

In [64]:
```
movie.describe().transpose()['count'].sort_values(ascending=False)
```

Out[64]:
```
Movie127    2313.0
Movie140     578.0
Movie16      320.0
Movie103     272.0
Movie29      243.0
             ...
Movie68        1.0
Movie69        1.0
Movie145       1.0
Movie71        1.0
Movie1         1.0
Name: count, Length: 206, dtype: float64
```

In [66]:
```
movie.describe().transpose()['mean']
```

Out[66]:
```
Movie1      5.000000
Movie2      5.000000
Movie3      2.000000
Movie4      5.000000
Movie5      4.103448
              ...
Movie202    4.333333
Movie203    3.000000
Movie204    4.375000
Movie205    4.628571
Movie206    4.923077
Name: mean, Length: 206, dtype: float64
```

In [68]:
```
movie[movie.columns[1:207]].sum().sort_values(ascending=False)
```

Out[68]:
```
Movie127    9511.0
Movie140    2794.0
Movie16     1446.0
Movie103    1241.0
```

```
Movie29     1168.0
              ...
Movie45        1.0
Movie60        1.0
Movie58        1.0
Movie154       1.0
Movie67        1.0
Length: 206, dtype: float64
```

In [69]:
```
movie.describe().transpose()
```

Out[69]:

	count	mean	std	min	25%	50%	75%	max
Movie1	1.0	5.000000	NaN	5.0	5.00	5.0	5.0	5.0
Movie2	1.0	5.000000	NaN	5.0	5.00	5.0	5.0	5.0
Movie3	1.0	2.000000	NaN	2.0	2.00	2.0	2.0	2.0
Movie4	2.0	5.000000	0.000000	5.0	5.00	5.0	5.0	5.0
Movie5	29.0	4.103448	1.496301	1.0	4.00	5.0	5.0	5.0
...	...	...	...	...	...	...	...	...
Movie202	6.0	4.333333	1.632993	1.0	5.00	5.0	5.0	5.0
Movie203	1.0	3.000000	NaN	3.0	3.00	3.0	3.0	3.0
Movie204	8.0	4.375000	1.407886	1.0	4.75	5.0	5.0	5.0
Movie205	35.0	4.628571	0.910259	1.0	5.00	5.0	5.0	5.0
Movie206	13.0	4.923077	0.277350	4.0	5.00	5.0	5.0	5.0

206 rows × 8 columns

In [70]:
```
# finding which movies have maximum views/ratings?
movie.describe().T["count"].sort_values(ascending = False).head()
```

Out[70]:
```
Movie127    2313.0
Movie140     578.0
Movie16      320.0
```

```
Movie103     272.0
Movie29      243.0
Name: count, dtype: float64
```

In [12]:
```
movie.head()
```

Out[12]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie
0	A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	NaN	Na
1	AH3QC2PC1VTGP	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	Na
2	A3LKP6WPMP9UKX	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	Na
3	AVIY68KEPQ5ZD	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	Na
4	A1CV1WROP5KTTW	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	Na

5 rows × 207 columns

In [72]:
```
# finding what is the average rating for each movie? Define the top 5 m
ovies with the maximum ratings.
movie.drop('user_id',axis=1).mean()
```

Out[72]:
```
Movie1      5.000000
Movie2      5.000000
Movie3      2.000000
Movie4      5.000000
Movie5      4.103448
               ...
Movie202    4.333333
Movie203    3.000000
Movie204    4.375000
Movie205    4.628571
Movie206    4.923077
Length: 206, dtype: float64
```

In [73]:
```
# finding top 5 movies with max total ratings
movie.drop('user_id',axis=1).sum()
```

Out[73]:
```
Movie1        5.0
Movie2        5.0
Movie3        2.0
Movie4       10.0
Movie5      119.0
            ...
Movie202     26.0
Movie203      3.0
Movie204     35.0
Movie205    162.0
Movie206     64.0
Length: 206, dtype: float64
```

In [74]:
```
movie.drop('user_id',axis=1).sum().sort_values(ascending=False).head()
```

Out[74]:
```
Movie127    9511.0
Movie140    2794.0
Movie16     1446.0
Movie103    1241.0
Movie29     1168.0
dtype: float64
```

In [75]:
```
# finding top 5 movies with max average ratings
movie.drop('user_id',axis=1).mean().sort_values(ascending=False).head()
```

Out[75]:
```
Movie1      5.0
Movie55     5.0
Movie131    5.0
Movie132    5.0
Movie133    5.0
dtype: float64
```

In [76]:
```
# finding the top 5 movies with the least audience
movie.drop('user_id',axis=1).isna().sum().sort_values(ascending=False).
head()
```

Out[76]:
```
Movie1      4847
Movie154    4847
Movie67     4847
```

```
Movie66     4847
Movie13     4847
dtype: int64
```

In [77]:
```
movie.drop('user_id',axis=1).fillna(movie.mean(axis=0)).min().head()
```

Out[77]:
```
Movie1    5.0
Movie2    5.0
Movie3    2.0
Movie4    5.0
Movie5    1.0
dtype: float64
```

In [78]:
```
movie_min=movie.drop('user_id',axis=1).fillna(movie.mean(axis=0)).min()
.to_frame('Min_Ratings')
```

In [20]:
```
movie_min
```

Out[20]:

	Min_Ratings
Movie1	5.0
Movie2	5.0
Movie3	2.0
Movie4	5.0
Movie5	1.0
...	...
Movie202	1.0
Movie203	3.0
Movie204	1.0
Movie205	1.0
Movie206	4.0

206 rows × 1 columns

In [79]:
```
movie_min_least = movie_min[movie_min.Min_Ratings <= 1]
```

In [80]:
```
movie_min_least
```

Out[80]:

	Min_Ratings
Movie5	1.0
Movie16	1.0
Movie26	1.0
Movie29	1.0
Movie45	1.0
Movie52	1.0
Movie53	1.0
Movie58	1.0
Movie60	1.0
Movie62	1.0
Movie67	1.0
Movie69	1.0
Movie81	1.0
Movie86	1.0
Movie89	1.0
Movie90	1.0
Movie91	1.0
Movie95	1.0
Movie103	1.0
Movie107	1.0

	Min_Ratings
Movie108	1.0
Movie111	1.0
Movie127	1.0
Movie138	1.0
Movie140	1.0
Movie144	1.0
Movie154	1.0
Movie158	1.0
Movie197	1.0
Movie202	1.0
Movie204	1.0
Movie205	1.0

In [81]:
```
movie.drop('user_id',axis=1).isna().sum().sort_values(ascending=False).
head()
```

Out[81]:
```
Movie1      4847
Movie154    4847
Movie67     4847
Movie66     4847
Movie13     4847
dtype: int64
```

In [88]:
```
#- Recommendation Model: Some of the movies hadn't been watched and the
refore, are not rated by the users. Netflix would like to take this as
 an opportunity and build a machine learning recommendation algorithm w
hich provides the ratings for each of the users.

#Divide the data into training and test data
#Build a recommendation model on training data
#Make predictions on the test data
```

```
movie_final = movie.fillna(0)
movie_final.set_index('user_id',inplace=True)
#df_user_moviesratings_and_views_final
from sklearn.model_selection import train_test_split
movie_final_train,movie_final_test= train_test_split(movie_final,test_s
ize=0.25,random_state=42)
```

In [89]:
```
#Shape of train and test set
print(movie_final_train.shape)
print(movie_final_test.shape)
```

```
(3636, 206)
(1212, 206)
```

In [90]:
```
import numpy as np
matrix_training = np.array(movie_final_train)
matrix_testing = np.array(movie_final_test)
```

In [91]:
```
from sklearn.metrics.pairwise import pairwise_distances
user_similarity_training = pairwise_distances(matrix_training, metric=
'cosine')
user_similarity_testing = pairwise_distances(matrix_testing, metric='co
sine')
user_similarity_training
```

Out[91]:
```
array([[0., 1., 1., ..., 1., 1., 1.],
       [1., 0., 0., ..., 0., 0., 1.],
       [1., 0., 0., ..., 0., 0., 1.],
       ...,
       [1., 0., 0., ..., 0., 0., 1.],
       [1., 0., 0., ..., 0., 0., 1.],
       [1., 1., 1., ..., 1., 1., 0.]])
```

In [92]:
```
def make_prediction(rating_matrix, similarity, type='user'):
    mean_user_rating = rating_matrix.mean(axis=1)
    rating_difference = (rating_matrix - mean_user_rating[:, np.newaxis
])
    pred = mean_user_rating[:, np.newaxis] + similarity.dot(rating_diff
```

```
erence) / np.array([np.abs(similarity).sum(axis=1)]).T
    return pred
```

In [93]:
```
predict_train_set = make_prediction(matrix_training,user_similarity_tra
ining,type='user')
predict_train_set
```

Out[93]:
```
array([[0.00417715, 0.00417715, 0.00324141, ..., 0.01322262, 0.0375518
3,
        0.01634175],
       [0.00334392, 0.00334392, 0.00174392, ..., 0.01881062, 0.0604107
1,
        0.02414396],
       [0.00334392, 0.00334392, 0.00174392, ..., 0.01881062, 0.0604107
1,
        0.02414396],
       ...,
       [0.00334392, 0.00334392, 0.00174392, ..., 0.01881062, 0.0604107
1,
        0.02414396],
       [0.00334392, 0.00334392, 0.00174392, ..., 0.01881062, 0.0604107
1,
        0.02414396],
       [0.0038696 , 0.0038696 , 0.00302186, ..., 0.01206441, 0.0341056
1,
        0.0148902 ]])
```

In [ ]:
```

```