Project 3: Building user-based recommendation model for Amazon. Analysis Task

- Exploratory Data Analysis:

Which movies have maximum views/ratings? What is the average rating for each movie? Define the top 5 movies with the maximum ratings. Define the top 5 movies with the least audience.

- Recommendation Model: Some of the movies hadn't been watched and therefore, are not rated by the users. Netflix would like to take this as an opportunity and build a machine learning recommendation algorithm which 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 [19]:

```
import numpy as np
import pandas as pd
```

In [20]:

```
amazon= pd.read_csv('C:\\Users\\lenovo\\Desktop\\Amazon - Movies and TV Ratings.csv')
```

In [21]:

```
amazon_pd = pd.DataFrame(amazon)
```

In [22]:

```
amazon.head()
```

Out[22]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	...	Movie197	Movie198	Movie199	Movie20
0	A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
1	AH3QC2PC1VTGP	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
2	A3LKP6WPMP9UKX	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
3	AVIY68KEPQ5ZD	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
4	A1CV1WROP5KTTW	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na

5 rows × 207 columns

In [23]:

```
amazon.shape
```

Out[23]:

```
(4848, 207)
```

In [24]:

```
amazon.size
```

Out[24]:

```
1003536
```

In [25]:

```
amazon.describe()
```

Out[25]:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	...	Movie197	Movie198	Movie199	Movie200	M
count	1.0	1.0	1.0	2.0	29.000000	1.0	1.0	1.0	1.0	1.0	...	5.000000	2.0	1.0	8.000000	3
mean	5.0	5.0	2.0	5.0	4.103448	4.0	5.0	5.0	5.0	5.0	...	3.800000	5.0	5.0	4.625000	4
std	NaN	NaN	NaN	0.0	1.496301	NaN	NaN	NaN	NaN	NaN	...	1.643168	0.0	NaN	0.517549	
min	5.0	5.0	2.0	5.0	1.000000	4.0	5.0	5.0	5.0	5.0	...	1.000000	5.0	5.0	4.000000	3
25%	5.0	5.0	2.0	5.0	4.000000	4.0	5.0	5.0	5.0	5.0	...	4.000000	5.0	5.0	4.000000	4
50%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	4.000000	5.0	5.0	5.000000	5
75%	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	5.000000	5.0	5.0	5.000000	5
max	5.0	5.0	2.0	5.0	5.000000	4.0	5.0	5.0	5.0	5.0	...	5.000000	5.0	5.0	5.000000	5

8 rows × 206 columns

In [26]:

```
# Maximum number of views
amazon.describe().T["count"].sort_values(ascending = False)[0:6]
```

Out[26]:

```
Movie127     2313.0
```

```
Movie140     578.0
Movie16      320.0
Movie103     272.0
Movie29      243.0
Movie91      128.0
Name: count, dtype: float64
```

In [27]:

```
amazon.index
```

Out[27]:

```
RangeIndex(start=0, stop=4848, step=1)
```

In [28]:

```
amazon.columns
```

Out[28]:

```
Index(['user_id', 'Movie1', 'Movie2', 'Movie3', 'Movie4', 'Movie5', 'Movie6',
       'Movie7', 'Movie8', 'Movie9',
       ...
       'Movie197', 'Movie198', 'Movie199', 'Movie200', 'Movie201', 'Movie202',
       'Movie203', 'Movie204', 'Movie205', 'Movie206'],
      dtype='object', length=207)
```

In [29]:

```
Amazon_filtered = amazon.fillna(value=0)
Amazon_filtered
```

Out[29]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	...	Movie197	Movie198	Movie199	Mov
0	A3R5OBKS7OM2IR	5.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
1	AH3QC2PC1VTGP	0.0	0.0	2.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
2	A3LKP6WPMP9UKX	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
3	AVIY68KEPQ5ZD	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4	A1CV1WROP5KTTW	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4843	A1IMQ9WMFYKWH5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4844	A1KLIKPUF5E88I	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4845	A5HG6WFZLO10D	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4846	A3UU690TWXCG1X	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	
4847	AI4J762YI6S06	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	

4848 rows × 207 columns

In [30]:

```
Amazon_filtered1 = Amazon_filtered.drop(columns='user_id')
Amazon_filtered1.head()
```

Out[30]:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	...	Movie197	Movie198	Movie199	Movie200	Movie20
0	5.0	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.
1	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.
2	0.0	0.0	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.
3	0.0	0.0	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.
4	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.

5 rows × 206 columns

In [31]:

```
Amazon_filtered1.describe()
```

Out[31]:

	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	Movie10	...
count	4848.000000	4848.000000	4848.000000	4848.000000	4848.000000	4848.000000	4848.000000	4848.000000	4848.000000	4848.000000	...
mean	0.001031	0.001031	0.000413	0.002063	0.024546	0.000825	0.001031	0.001031	0.001031	0.001031	...
std	0.071811	0.071811	0.028724	0.101545	0.336268	0.057448	0.071811	0.071811	0.071811	0.071811	...

min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
25%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
50%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
75%	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...
max	5.000000	5.000000	2.000000	5.000000	5.000000	4.000000	5.000000	5.000000	5.000000	5.000000	...

8 rows × 206 columns

In [32]:

```
Amazon_max_views = Amazon_filtered1.sum()
Amazon_max_views
```

Out[32]:

```
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 [33]:

```
#Finding maximum sum of ratings
max(Amazon_max_views)
```

Out[33]:

```
9511.0
```

In [34]:

```
Amazon_max_views.head()
Amazon_max_views.tail()
```

Out[34]:

```
Movie202     26.0
Movie203      3.0
Movie204     35.0
Movie205    162.0
Movie206     64.0
dtype: float64
```

In [35]:

```
Amazon_max_views.index
```

Out[35]:

```
Index(['Movie1', 'Movie2', 'Movie3', 'Movie4', 'Movie5', 'Movie6', 'Movie7',
       'Movie8', 'Movie9', 'Movie10',
       ...
       'Movie197', 'Movie198', 'Movie199', 'Movie200', 'Movie201', 'Movie202',
       'Movie203', 'Movie204', 'Movie205', 'Movie206'],
      dtype='object', length=206)
```

In [36]:

```
#Finding which movie has maximum veiws\Ratings
max_views= Amazon_max_views.argmax()
max_views
```

Out[36]:

```
126
```

In [37]:

```
#checking whether that movie has max views/ratings or not
Amazon_max_views['Movie126']
```

Out[37]:

```
9.0
```

In [38]:

```
sum(Amazon_max_views)
sum(Amazon_max_views)
```

Out[38]: 21928.0

In [39]:

```
len(Amazon_max_views.index)
```

Out[39]: 206

In [40]:

```
#Average rating for each movie
Average_ratings_of_every_movie=sum(Amazon_max_views)/len(Amazon_max_views.index)
Average_ratings_of_every_movie
```

Out[40]: 106.44660194174757

In [41]:

```
Amazon_df = pd.DataFrame(Amazon_max_views)
Amazon_df.head()
```

Out[41]:

	0
Movie1	5.0
Movie2	5.0
Movie3	2.0
Movie4	10.0
Movie5	119.0

In [42]:

```
Amazon_df.columns=['rating']
```

In [43]:

```
Amazon_df.index
```

Out[43]:

```
Index(['Movie1', 'Movie2', 'Movie3', 'Movie4', 'Movie5', 'Movie6', 'Movie7',
       'Movie8', 'Movie9', 'Movie10',
       ...
       'Movie197', 'Movie198', 'Movie199', 'Movie200', 'Movie201', 'Movie202',
       'Movie203', 'Movie204', 'Movie205', 'Movie206'],
      dtype='object', length=206)
```

In [44]:

```
Amazon_df.tail()
```

Out[44]:

	rating
Movie202	26.0
Movie203	3.0
Movie204	35.0
Movie205	162.0
Movie206	64.0

In [45]:

```
#Top 5 movie ratings
Amazon_df.nlargest(5,'rating')
```

Out[45]:

	rating
Movie127	9511.0
Movie140	2794.0
Movie16	1446.0
Movie103	1241.0
Movie29	1168.0

In [46]:

```
#Top 5 movies having least audience
Amazon_df.nsmallest(5,'rating')
```

Out[46]:

	rating
Movie45	1.0
Movie58	1.0
Movie60	1.0
Movie67	1.0
Movie69	1.0

In [47]:

```
melt_df=amazon_pd.melt(id_vars= amazon.columns[0],value_vars=amazon.columns[1:],var_name='Movie',value_name='rati
```

In [48]:

```
melt_df
```

Out[48]:

	user_id	Movie	rating
0	A3R5OBKS7OM2IR	Movie1	5.0
1	AH3QC2PC1VTGP	Movie1	NaN
2	A3LKP6WPMP9UKX	Movie1	NaN
3	AVIY68KEPQ5ZD	Movie1	NaN
4	A1CV1WROP5KTTW	Movie1	NaN
...	...	...	...
998683	A1IMQ9WMFYKWH5	Movie206	5.0
998684	A1KLIKPUF5E88I	Movie206	5.0
998685	A5HG6WFZLO10D	Movie206	5.0
998686	A3UU690TWXCG1X	Movie206	5.0
998687	AI4J762YI6S06	Movie206	5.0

998688 rows × 3 columns

In [51]:

```
melt_df.shape
```

Out[51]:

```
(998688, 3)
```

In [52]:

```
melt_filtered = melt_df.fillna(0)
melt_filtered.shape
```

Out[52]:

```
(998688, 3)
```

In [99]:

```
import sklearn
```

In [107…

```
from sklearn.model_selection import train_test_split
```

In [108…

```
trainset, testset = train_test_split(amazon, test_size=0.25)
```

In [111…

```
import surprise
```

In [113…

```
from surprise import Reader
from surprise import Dataset
from surprise import SVD
from surprise.model_selection import train_test_split
```

In [115…

```
reader = Reader(rating_scale=(-1,10))
data = Dataset.load_from_df(melt_df.fillna(0), reader=reader)
```

In [116…

```
#Divide the data into training and test data
trainset, testset = train_test_split(data, test_size=0.25)
```

In [117…

```
algo = SVD()
```

In [118…

```
#Building a model
algo.fit(trainset)
```

Out[118…

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x145579c7910>
```

In [119…

```
#Make predictions on the test data
predict= algo.test(testset)
```

In [120…

```
from surprise.model_selection import cross_validate
```

In [122…

```
cross_validate(algo,data,measures=['RMSE','MAE'],cv=3,verbose=True)
```

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

                  Fold 1  Fold 2  Fold 3  Mean    Std     
RMSE (testset)    0.2863  0.2788  0.2813  0.2821  0.0031  
MAE (testset)     0.0431  0.0420  0.0426  0.0426  0.0005  
Fit time          57.64   58.25   69.61   61.83   5.51    
Test time         4.39    4.68    6.62    5.23    0.99    
```

Out[122…

```
{'test_rmse': array([0.28631854, 0.27878443, 0.28129053]),
 'test_mae': array([0.0431301 , 0.04200665, 0.04259365]),
 'fit_time': (57.63597893714905, 58.24941897392273, 69.61171960830688),
 'test_time': (4.391650915145874, 4.675920248031616, 6.62487006187439)}
```

In [123…

```
user_id='A1CV1WROP5KTTW'
Movie='Movie6'
rating='5'
algo.predict(user_id,Movie,r_ui=rating)
print(cross_validate(algo,data,measures=['RMSE','MAE'],cv=3,verbose=True))
```

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

                  Fold 1  Fold 2  Fold 3  Mean    Std     
RMSE (testset)    0.2758  0.2848  0.2858  0.2821  0.0045  
MAE (testset)     0.0426  0.0428  0.0429  0.0428  0.0002  
Fit time          57.45   56.91   53.07   55.81   1.95    
Test time         7.78    4.04    4.31    5.37    1.70    
{'test_rmse': array([0.27583837, 0.28482575, 0.28575805]), 'test_mae': array([0.04256259, 0.0427926 , 0.04293221]
), 'fit_time': (57.448200702667236, 56.912312746047974, 53.074296951293945), 'test_time': (7.778097629547119, 4.0
35364151000977, 4.307275772094727)}
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js