PROJECT

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 [ ]:
```
import numpy as np
import pandas as pd
import re
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import surprise
```

In [2]:
```
df = pd.read_csv('Amazon - Movies and TV Ratings.csv')
```

In [3]:
```
df.head()
```

Out[3]:

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

5 rows × 207 columns

In [4]:
```
df.shape
```

Out[4]:
```
(4848, 207)
```

In [7]:
```
df_org = df.copy()
```

In [8]:
```
1 df.describe().transpose()
```

Out[8]:

	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

Q 1 - Which movies have maximum views/ratings?

In [22]:
```
1 #Movies with highest views
2 count=df.describe().transpose()['count']
```

In [23]:
```
1 count.sort_values(ascending=False)
```

Out[23]:
```
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
```

Movie127 is highest views 2313

In [30]:
```
1 #Movie with highest ratings
2 df.drop('user_id',axis=1).sum().sort_values(ascending=False)
```

Out[30]:
```
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
```

Movie127 is highest ratings 9511

Q2- What is the average rating for each movie? Define the top 5 movies with the maximum ratings

In [35]:
```
avg_ratings=df.drop('user_id',axis=1).mean().sort_values(ascending=False)
```

In [37]:
```
avg_ratings[:5]
```

Out[37]:
```
Movie1      5.0
Movie55     5.0
Movie131    5.0
Movie132    5.0
Movie133    5.0
dtype: float64
```

These are the top 5 Movies with maximum ratings Movie1=5, Movie55=5, Movie131=5, Movie132=5, Movie133=5.

Q3 - Define the top 5 movies with the least audience

In [39]:
```
count.sort_values(ascending=True)[:5]
```

Out[39]:
```
Movie1      1.0
Movie71     1.0
Movie145    1.0
Movie69     1.0
Movie68     1.0
Name: count, dtype: float64
```

These are the top 5 movies with least audience

Q4 - Recommendation Model

1)Divide the data into training and test data

2)Build a recommendation model on training data

3)Make predictions on the test data

In [47]:
```
from surprise import Reader
from surprise import accuracy
from surprise import Dataset
from surprise.model_selection import train_test_split
from surprise import SVD
from surprise.model_selection import cross_validate
```

In [48]:
```
df_melt = df.melt(id_vars = df.columns[0],value_vars=df.columns[1:],var_name="Movies",value_name
```

In [49]:
```
df_melt
```

Out[49]:

	user_id	Movies	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 [50]:
```
rd = Reader()
data = Dataset.load_from_df(df_melt.fillna(0),reader=rd)
data
```

Out[50]:
```
<surprise.dataset.DatasetAutoFolds at 0x1aa9c3a3ee0>
```

In [51]:
```
trainset, testset = train_test_split(data,test_size=0.25)
```

In [53]:
```
svd = SVD()
svd.fit(trainset)
```

Out[53]:
```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x1aa9be60670>
```

In [54]:
```
pred = svd.test(testset)
```

In [55]:
```
accuracy.rmse(pred)
```

```
RMSE: 1.0262
```

Out[55]:
```
1.026169304037489
```

In [56]:
```
accuracy.mae(pred)
```

```
MAE:  1.0121
```

Out[56]:
```
1.012100664159146
```

In [57]:
```
cross_validate(svd, 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)    1.0264  1.0256  1.0266  1.0262  0.0004
MAE (testset)     1.0121  1.0119  1.0123  1.0121  0.0002
Fit time          92.18   98.22   95.99   95.46   2.49
Test time         6.89    7.72    7.88    7.50    0.43
```

Out[57]:
```
{'test_rmse': array([1.02640404, 1.02562186, 1.02663572]),
 'test_mae': array([1.01211636, 1.01185292, 1.01226253]),
 'fit_time': (92.1798300743103, 98.21732902526855, 95.98875212669373),
 'test_time': (6.892607688903809, 7.716824531555176, 7.882725477218628)}
```

In [58]:
```
def repeat(ml_type,dframe):
    rd = Reader()
    data = Dataset.load_from_df(dframe,reader=rd)
    print(cross_validate(ml_type, data, measures = ['RMSE', 'MAE'], cv = 3, verbose = True))
    print("--"*15)
    usr_id = 'A3R5OBKS7OM2IR'
    mv = 'Movie1'
    r_u = 5.0
    print(ml_type.predict(usr_id,mv,r_ui = r_u,verbose=True))
    print("--"*15)

```

In [74]:
```
repeat(SVD(),df_melt.fillna(df_melt['Rating'].mean()))
```

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0864  0.0867  0.0848  0.0860  0.0008
MAE (testset)     0.0098  0.0099  0.0095  0.0097  0.0001
Fit time          54.06   54.93   54.19   54.39   0.38
Test time         47.32   4.84    4.01    18.72   20.22
{'test_rmse': array([0.08639582, 0.08669061, 0.08481394]), 'test_mae': array([0.00975858, 0.0098920
9, 0.00953199]), 'fit_time': (54.06094002723694, 54.925249338150024, 54.188737869262695), 'test_tim
e': (47.31636881828308, 4.838199615478516, 4.006798982620239)}
------------------------------
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 4.40   {'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 4.40   {'was_impossible': False}
------------------------------
```

In [75]:
```
repeat(SVD(),df_melt.fillna(df_melt['Rating'].median()))
```

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

                  Fold 1  Fold 2  Fold 3  Mean    Std
RMSE (testset)    0.0944  0.0926  0.0905  0.0925  0.0016
MAE (testset)     0.0072  0.0072  0.0069  0.0071  0.0001
Fit time          54.15   54.50   54.46   54.37   0.15
Test time         4.03    3.84    3.84    3.91    0.09
{'test_rmse': array([0.09440161, 0.09255606, 0.0904952 ]), 'test_mae': array([0.00718722, 0.0071571
1, 0.00692521]), 'fit_time': (54.15489196777344, 54.49550271034241, 54.45726823806763), 'test_tim
e': (4.0323145389556885, 3.842820644378662, 3.842846393585205)}
------------------------------
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 5.00   {'was_impossible': False}
user: A3R5OBKS7OM2IR item: Movie1     r_ui = 5.00   est = 5.00   {'was_impossible': False}
------------------------------
```

In [ ]:
```

```