PGP AI & ML - Cohort 5 - Tamal Acharya

In [ ]:

```
# 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 [ ]:

In [3]:

```
# Read in data
movie = pd.read_csv('C:/Users/Tamal/Downloads/1567507480_amazonmoviesandtvratings/Amazon - Movies and TV Ratings.csv')
movie.head()
```

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

5 rows × 207 columns

In [4]:

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

```
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 [5]:

```
movie.describe().transpose()['mean']
```

```
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 [6]:

```
movie[movie.columns[1:207]].sum().sort_values(ascending=False)[:5]
```

```
Movie127    9511.0
Movie140    2794.0
Movie16     1446.0
Movie103    1241.0
Movie29     1168.0
dtype: float64
```

Q1. Which movies have maximum views/ratings?

In [8]:

```
movie.describe().T
```

	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 [11]:

```
movie.describe().T["count"].sort_values(ascending = False).head()
```

```
Movie127    2313.0
Movie140     578.0
Movie16      320.0
Movie103     272.0
Movie29      243.0
Name: count, dtype: float64
```

From the above table it is clear that “Movie 127” is having maximum views (2313 views)

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

In [22]:

```
movie.drop('user_id',axis=1).mean()
```

```
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
```

From the above table, the average ratings of each movie is given and it is between 2 to 5

In [20]:

```
#Top 5 movies with max total ratings
movie.drop('user_id',axis=1).sum().sort_values(ascending=False).head()
```

```
Movie127    9511.0
Movie140    2794.0
Movie16     1446.0
Movie103    1241.0
Movie29     1168.0
dtype: float64
```

In [26]:

```
#Top 5 movies with max average ratings
movie.drop('user_id',axis=1).mean().sort_values(ascending=False).head()
```

```
Movie1      5.0
Movie55     5.0
Movie131    5.0
Movie132    5.0
Movie133    5.0
dtype: float64
```

From the above table it is clear that “Movie 127” is having maximum rating (9511)

Q3. Define the top 5 movies with the least audience.

In [24]:

```
#Count the NaN fields
movie.drop('user_id',axis=1).isna().sum().sort_values(ascending=False).head()
```

```
Movie1      4847
Movie154    4847
Movie67     4847
Movie66     4847
Movie13     4847
dtype: int64
```

In [30]:

```
# Top 5 Movies with least audience
movie.drop('user_id',axis=1).fillna(movie.mean(axis=0)).min().head()
```

```
Movie1    5.0
Movie2    5.0
Movie3    2.0
Movie4    5.0
Movie5    1.0
dtype: float64
```

In [34]:

```
movie_min=movie.drop('user_id',axis=1).fillna(movie.mean(axis=0)).min().to_frame('Min_Ratings')
movie_min_least = movie_min[movie_min.Min_Ratings <= 1]
#dmin=dmin.sort_values(by=[Index],ascending = True)
movie_min_least.head()
```

	Min_Ratings
Movie5	1.0
Movie16	1.0
Movie26	1.0
Movie29	1.0
Movie45	1.0

In [35]:

```
#Count the NaN fields
movie.drop('user_id',axis=1).isna().sum().sort_values(ascending=False).head()
```

```
Movie1      4847
Movie154    4847
Movie67     4847
Movie66     4847
Movie13     4847
dtype: int64
```

From the above two tables it is clear the top 5 movies with 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.

Q4. Divide the data into training and test data

Q5. Build a recommendation model on training data

Q6. Make predictions on the test data

In [ ]:

```
#Q4. Divide the data into training and test data
```

In [42]:

```
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_size=0.25,random_state=42)
```

In [37]:

```
#Shape of train and test set
print(movie_final_train.shape)
print(movie_final_test.shape)
```

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

In [ ]:

```
#Q5. Build a recommendation model on training data
```

In [38]:

```
import numpy as np
matrix_training = np.array(movie_final_train)
matrix_testing = np.array(movie_final_test)
from sklearn.metrics.pairwise import pairwise_distances
user_similarity_training = pairwise_distances(matrix_training, metric='cosine')
user_similarity_testing = pairwise_distances(matrix_testing, metric='cosine')
user_similarity_training
```

```
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 [ ]:

```
#Q6. Make predictions on the test data
```

In [40]:

```
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_difference) / np.array([np.abs(similar
ity).sum(axis=1)]).T
    return pred

predict_train_set = make_prediction(matrix_training,user_similarity_training,type='user')
predict_train_set
```

```
array([[0.00417715, 0.00417715, 0.00324141, ..., 0.01322262, 0.03755183,
        0.01634175],
       [0.00334392, 0.00334392, 0.00174392, ..., 0.01881062, 0.06041071,
        0.02414396],
       [0.00334392, 0.00334392, 0.00174392, ..., 0.01881062, 0.06041071,
        0.02414396],
       ...,
       [0.00334392, 0.00334392, 0.00174392, ..., 0.01881062, 0.06041071,
        0.02414396],
       [0.00334392, 0.00334392, 0.00174392, ..., 0.01881062, 0.06041071,
        0.02414396],
       [0.0038696 , 0.0038696 , 0.00302186, ..., 0.01206441, 0.03410561,
        0.0148902 ]])
```

In [41]:

```
predict_test_set = make_prediction(matrix_testing,user_similarity_testing,type='user')
predict_test_set
```

```
array([[ 0.00240065,  0.00240065,  0.00240065, ...,  0.00333796,
         0.04926633,  0.02114692],
       [-0.01869911, -0.01869911, -0.01869911, ..., -0.01720574,
         0.05596953,  0.01116835],
       [ 0.00233722,  0.00233722,  0.00233722, ...,  0.00317997,
         0.04447472,  0.01919222],
       ...,
       [-0.01869911, -0.01869911, -0.01869911, ..., -0.01720574,
         0.05596953,  0.01116835],
       [ 0.00071837,  0.00071837,  0.00071837, ...,  0.00221174,
         0.075387  ,  0.03058582],
       [ 0.00071837,  0.00071837,  0.00071837, ...,  0.00221174,
         0.075387  ,  0.03058582]])
```

In [ ]:

In [ ]:

In [ ]: