Building user-based recommendation model for Amazon

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 4848 distinct users

Data Considerations

- All the users have not watched all the movies and therefore, all movies 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 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

import the nessasary libraries:

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

load the dataset:

In [22]:
```
data = pd.read_csv('Amazon - Movies and TV Ratings.csv')
data.head(10)
```

Out[22]:

	user_id	Movie1	Movie2	Movie3	Movie4	Movie5	Movie6	Movie7	Movie8	Movie9	...	Movie197	Movie198	Movie199	Movie200	Movie201	Movie202	Movie203	Movie204	Movie205	Movie206
0	A3R5OBKS7OM2IR	5.0	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	AH3QC2PC1VTGP	NaN	NaN	2.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A3LKP6WPMP9UKX	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	AVIY68KEPQ5ZD	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	A1CV1WROP5KTTW	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	AP57WZ2X4G0AA	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	A3NMBJ2LCRCATT	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	A5Y15SAOMX6XA	NaN	NaN	NaN	NaN	2.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	A3P671HJ32TCSF	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	A3VCKTRD24BG7K	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

10 rows × 207 columns

In [63]:
```
data.shape
```

Out[63]:
```
(4848, 207)
```

Exploratory Data Analysis:

Task 1 - Which movies have maximum views/ratings?

In [23]:
```
# calculate the statistical information of all movies:
desc = data.describe().T
desc
```

Out[23]:

	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 [45]:
```
# The movie with the max. views/ratings:
desc2 = desc['count'].sort_values(ascending=False).to_frame()
pop = desc2[:1]
pop
```

Out[45]:

	count
Movie127	2313.0

Insight:

we can see from the output above, the Movie127 has the maximum views/ratings.

Task 2 - What is the average rating for each movie? Define the top 5 movies with the maximum ratings

In [43]:
```
data2 = data.drop('user_id', axis = 1)
rating_avr = data2.mean().sort_values(ascending=False).to_frame().rename(columns= {0:'Average Rating'} )
rating_avr.head(5)
```

Out[43]:

	Average Rating
Movie1	5.0
Movie66	5.0
Movie76	5.0
Movie75	5.0
Movie74	5.0

Insight:

we can see from the table above, that some "top" movies are not as popular as we know, the reason could be the number of ratings is too few. Then we should count the rating number of each movie and filter the ones with too few viewers.

In [51]:
```
rating_avr['count'] = desc2['count']
rating_avr.head(5)
```

Out[51]:

	Average Rating	count
Movie1	5.0	1.0
Movie66	5.0	1.0
Movie76	5.0	2.0
Movie75	5.0	1.0
Movie74	5.0	1.0

The table above has proved our presumption: a big part of the "top 25" has very few viewer. So now we need to identify a filter:

In [49]:
```
rating_avr['count'].describe()
```

Out[49]:
```
count     206.000000
mean       24.271845
std       168.937841
min         1.000000
25%         1.000000
50%         2.000000
75%         5.000000
max      2313.000000
Name: count, dtype: float64
```

From the information above we can see the difference between mean value(24.3) and 50%-quantile(2) is quite big. So we can try to set the threshold as 10 to filter the outliers out of the list.

In [53]:
```
rating_avr_filtered = rating_avr[rating_avr['count']>10]
rating_avr_filtered.head(5)
```

Out[53]:

	Average Rating	count
Movie206	4.923077	13.0
Movie162	4.866667	15.0
Movie140	4.833910	578.0
Movie184	4.823529	17.0
Movie158	4.818182	66.0

Task 3 - Define the top 5 movies with the least audience

In [60]:
```
rating_count = rating_avr['count'].sort_values(ascending=True).to_frame()
rating_count.head(5)
```

Out[60]:

	count
Movie1	1.0
Movie34	1.0
Movie35	1.0
Movie36	1.0
Movie37	1.0

Insight:

it could be more than 5 movies with only one audience, so we can list all movies with only 1 audience:

In [62]:
```
rating_count[rating_count['count']==1]
```

Out[62]:

	count
Movie1	1.0
Movie34	1.0
Movie35	1.0
Movie36	1.0
Movie37	1.0
...	...
Movie54	1.0
Movie84	1.0
Movie72	1.0
Movie77	1.0
Movie143	1.0

89 rows × 1 columns

From the information above we can see there are 89 movies with only one audience.

Recommendation Model:

import the nessasary libraries:

In [91]:
```
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import train_test_split
from sklearn.neighbors import NearestNeighbors
import scipy.sparse
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds
import warnings; warnings.simplefilter('ignore')
%matplotlib inline
```

Task 4 - Divide the data into training and test data

In [92]:
```
data_melt = data.melt(id_vars = data.columns[0], value_vars = data.columns[1:], var_name = "movies", value_name = "ratings")
data_melt
```

Out[92]:

	user_id	movies	ratings
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 [101...]:
```
dataset = data_melt.fillna(0)
dataset.head()
```

Out[101...]:

	user_id	movies	ratings
0	A3R5OBKS7OM2IR	Movie1	5.0
1	AH3QC2PC1VTGP	Movie1	0.0
2	A3LKP6WPMP9UKX	Movie1	0.0
3	AVIY68KEPQ5ZD	Movie1	0.0
4	A1CV1WROP5KTTW	Movie1	0.0

In [102...]:
```
ds_train, ds_test = train_test_split(dataset, test_size = 0.3, random_state=0)
print('Shape of training data: ',ds_train.shape)
print('Shape of testing data: ',ds_test.shape)
```

```
Shape of training data:  (699081, 3)
Shape of testing data:  (299607, 3)
```

Task 5 - Build a recommendation model on training data

At first we count the user_id for each unique movie as recommendation score:

In [113...]:
```
ds_cf = pd.concat([ds_train, ds_test]).reset_index()
ds_cf.head()
```

Out[113...]:

	index	user_id	movies	ratings
0	338560	A9Q5O3PAC51MV	Movie70	0.0
1	471992	A1EI65WJC85U68	Movie98	0.0
2	185811	A379SAP75SPDHD	Movie39	0.0
3	817860	A1JIPFV4OL520T	Movie169	0.0
4	251599	A11YEGV0NPRF3H	Movie52	0.0

Now we try to build a user-based collaborative filtering model:

In [116...]:
```
ds_pivot = ds_cf.pivot(index = 'user_id', columns ='movies', values = 'ratings')
ds_pivot.head()
```

Out[116...]:

movies	Movie1	Movie10	Movie100	Movie101	Movie102	Movie103	Movie104	Movie105	Movie106	Movie107	...	Movie90	Movie91	Movie92	Movie93	Movie94	Movie95	Movie96	Movie
user_id																			
A0047322388NOTO4N8SKD	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
A00473363TJ8YSZ3YAGG9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
A1004AX2J2HXGL	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
A100CQXJ6D44T9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
A100Z2S0880G9A	0.0	0.0	0.0	0.0	0.0	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 rows × 206 columns

In [118...]:
```
print('Shape of the pivot table: ', ds_pivot.shape)
```

```
Shape of the pivot table:  (4848, 206)
```

define user index from 0 to 4847:

In [120...]:
```
ds_pivot['user_index'] = np.arange(0, ds_pivot.shape[0], 1)
ds_pivot.head()
```

Out[120...]:

movies	Movie1	Movie10	Movie100	Movie101	Movie102	Movie103	Movie104	Movie105	Movie106	Movie107	...	Movie91	Movie92	Movie93	Movie94	Movie95	Movie96	Movie97	Movie
user_id																			
A0047322388NOTO4N8SKD	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
A00473363TJ8YSZ3YAGG9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
A1004AX2J2HXGL	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
A100CQXJ6D44T9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

movies	Movie1	Movie10	Movie100	Movie101	Movie102	Movie103	Movie104	Movie105	Movie106	Movie107	...	Movie91	Movie92	Movie93	Movie94	Movie95	Movie96	Movie97	Movie
user_id																			
A100Z2S0880G9A	0.0	0.0	0.0	0.0	0.0	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 rows × 207 columns

Actual ratings given by users:

In [121...
```
ds_pivot.set_index(['user_index'], inplace=True)
ds_pivot.head()
```

Out[121...

movies	Movie1	Movie10	Movie100	Movie101	Movie102	Movie103	Movie104	Movie105	Movie106	Movie107	...	Movie90	Movie91	Movie92	Movie93	Movie94	Movie95	Movie96	Movie97	Movie98	M
user_index																					
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	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	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	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	0.0	

5 rows × 206 columns

Insight:

As the table above shows, it is a sparse matrix. Therefore I decide to use SVD to build the model.

Singular Value Decomposition:

In [124...
```
U, sigma, Vt = svds(ds_pivot, k = 10)
```

In [125...
```
print('Left singular matrix: \n',U)
```

```
Left singular matrix:
 [[-1.41948014e-07  0.00000000e+00  3.70967613e-07 ...  1.50829005e-07
  -4.26840615e-02 -7.96633419e-05]
 [ 1.35316796e-05  5.13279606e-18 -3.46952163e-05 ... -8.11610274e-06
   1.14463559e-04 -1.91174114e-02]
 [ 3.38291990e-06  1.28319901e-18 -8.67380407e-06 ... -2.02902569e-06
   2.86158897e-05 -4.77935284e-03]
 ...
 [ 5.83560217e-17 -5.12079788e-17 -2.15057739e-18 ...  2.80729933e-18
   1.12638706e-18 -9.77882255e-19]
 [ 1.01487597e-05  3.84959704e-18 -2.60214122e-05 ... -6.08707706e-06
   8.58476690e-05 -1.43380585e-02]
 [ 6.76583981e-06  2.56639803e-18 -1.73476081e-05 ... -4.05805137e-06
   5.72317793e-05 -9.55870568e-03]]
```

In [126...
```
print('Sigma: \n',sigma)
```

```
Sigma:
 [ 34.72750225  39.54231658  41.16418185  47.74283509  53.94837751
  75.614235    77.65328486  82.08093229 117.1348772  209.22865684]
```

Insight:

As sigma is not a diagonal matrix we have to convert it into diagonal matrix.

In [127...
```
sigma = np.diag(sigma)
print('Diagonal matrix: \n',sigma)
```

```
Diagonal matrix:
 [[ 34.72750225   0.           0.           0.           0.
    0.           0.           0.           0.           0.        ]
 [  0.          39.54231658   0.           0.           0.
    0.           0.           0.           0.           0.        ]
 [  0.           0.          41.16418185   0.           0.
    0.           0.           0.           0.           0.        ]
 [  0.           0.           0.          47.74283509   0.
    0.           0.           0.           0.           0.        ]
 [  0.           0.           0.           0.          53.94837751
    0.           0.           0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
   75.614235     0.           0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
    0.          77.65328486   0.           0.           0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.          82.08093229   0.           0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.           0.         117.1348772    0.        ]
 [  0.           0.           0.           0.           0.
    0.           0.           0.           0.         209.22865684]]
```

In [128...
```
print('Right singular matrix: \n',Vt)
```

```
Right singular matrix:
 [[ 8.80742461e-18  4.12188543e-20 -5.14725821e-17 ...  4.29876196e-17
   1.17053737e-17 -2.86577494e-17]
 [-6.04778802e-18 -2.69716801e-20  3.28390103e-17 ... -4.90601570e-17
  -9.26786201e-18  2.06991882e-17]
 [-1.54140315e-17  3.01696910e-20 -5.19119135e-17 ...  1.04465392e-16
   8.13516843e-18 -6.54065377e-17]
 ...
 [ 2.47848977e-18  7.59626632e-21 -7.28740593e-18 ... -4.24420858e-18
   1.80783071e-18 -5.70492039e-20]
 [ 1.80479109e-17  4.48892113e-20 -5.02959782e-17 ... -3.18575876e-17
   1.35694779e-17  1.46018920e-17]
 [ 5.50504088e-17  1.18723560e-19 -1.28576337e-16 ... -1.06599787e-16
   3.57470449e-17  4.97935872e-17]]
```

Task 6 - Make predictions on the test data

Predicted ratings:

In [129...
```
user_predicted_ratings = np.dot(np.dot(U, sigma), Vt)
ds_pred = pd.DataFrame(user_predicted_ratings, columns = ds_pivot.columns)
ds_pred.head()
```

Out[129...

movies	Movie1	Movie10	Movie100	Movie101	Movie102	Movie103	Movie104	Movie105	Movie106	Movie107	...	Movie90	Movie91	Movie92	Movie93	Movie94	Movie95	Movie
0	-9.128838e-17	-2.272035e-19	2.545680e-16	1.817148e-16	6.498682e-07	1.093853e-06	4.345189e-12	-2.497853e-09	-2.506972e-09	-0.000003	...	5.399395e-07	2.204105e-07	1.330251e-06	1.288095e-16	3.514101e-17	0.000001	1.482448
1	-2.200700e-16	-4.753755e-19	5.150589e-16	5.005842e-16	-6.889186e-05	-2.419713e-06	-9.709430e-10	-3.222229e-07	-3.202302e-07	0.000316	...	-5.695961e-05	-2.481068e-06	7.615735e-06	2.828954e-16	8.567926e-17	-0.000137	3.616540

movies	Movie1	Movie10	Movie100	Movie101	Movie102	Movie103	Movie104	Movie105	Movie106	Movie107	...	Movie90	Movie91	Movie92	Movie93	Movie94	Movie95	Movie
2	-5.501750e-17	-1.188439e-19	1.287647e-16	1.251460e-16	-1.722296e-05	-6.049284e-07	-2.427357e-10	-8.055573e-08	-8.005755e-08	0.000079	...	-1.423990e-05	-6.202669e-07	1.903934e-06	7.072385e-17	2.141981e-17	-0.000034	9.041350
3	4.274205e-17	2.554430e-19	-3.106092e-16	1.845739e-16	1.932794e-06	7.188264e-08	-1.267691e-10	-3.464287e-08	-3.439380e-08	-0.000010	...	1.617642e-06	7.717334e-08	-2.259384e-07	-7.598565e-17	1.776231e-18	0.000004	-1.359844
4	1.147485e-17	3.105607e-20	-2.720744e-17	-1.874918e-17	6.549731e-05	-3.995040e-05	-4.219027e-08	-4.507963e-06	-4.425759e-06	-0.000409	...	5.393707e-05	-1.041088e-04	-1.538701e-04	-1.221829e-17	-3.362624e-18	0.000089	-1.927397

5 rows × 206 columns

Recommend the items with the highest predicted ratings:

In [136…
```
def recommend_items(user_index, ds_pivot, ds_pred, num_recommendations):
    user_idx = user_index
    # Get and sort the user's ratings
    sorted_user_ratings = ds_pivot.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_ratings
    sorted_user_predictions = ds_pred.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_predictions
    temp = pd.concat([sorted_user_ratings, sorted_user_predictions], axis=1)
    temp.index.name = 'Recommended Movie'
    temp.columns = ['user_ratings', 'user_predictions']
    temp = temp.loc[temp.user_ratings == 0]
    temp = temp.sort_values('user_predictions', ascending=False)
    print('\nBelow are the recommended Movies for user(user_index = {}):\n'.format(user_index))
    print(temp.head(num_recommendations))
```

In [137…
```
user_index = 4
num_recommendations = 5
recommend_items(user_index, ds_pivot, ds_pred, num_recommendations)
```

```
Below are the recommended Movies for user(user_index = 4):

                 user_ratings  user_predictions
Recommended Movie
Movie162                  0.0          0.019604
Movie86                   0.0          0.011711
Movie185                  0.0          0.000109
Movie163                  0.0          0.000104
Movie95                   0.0          0.000089
```

In [139…
```
user_index = 123
num_recommendations = 5
recommend_items(user_index, ds_pivot, ds_pred, num_recommendations)
```

```
Below are the recommended Movies for user(user_index = 123):

                 user_ratings  user_predictions
Recommended Movie
Movie202                  0.0          0.002864
Movie132                  0.0          0.002861
Movie188                  0.0          0.002838
Movie189                  0.0          0.002836
Movie190                  0.0          0.002835
```

In [140…
```
user_index = 2345
num_recommendations = 5
recommend_items(user_index, ds_pivot, ds_pred, num_recommendations)
```

```
Below are the recommended Movies for user(user_index = 2345):

                 user_ratings  user_predictions
Recommended Movie
Movie86                   0.0          0.000005
Movie95                   0.0          0.000004
Movie140                  0.0          0.000002
Movie102                  0.0          0.000002
Movie90                   0.0          0.000002
```

Insight:

as above shows, it is a Collaborative recommender model, so, all the three users are given different recommendations based on users past behaviour.

Model Evaluation:

Average actual ratings for each movie:

In [141…
```
ds_pivot.mean().head()
```

Out[141…
```
movies
Movie1      0.001031
Movie10     0.001031
Movie100    0.000825
Movie101    0.005157
Movie102    0.001650
dtype: float64
```

Average predicted ratings for each movie:

In [143…
```
ds_pred.mean().head()
```

Out[143…
```
movies
Movie1     -1.143966e-16
Movie10    -2.371565e-19
Movie100    2.546512e-16
Movie101    2.818410e-16
Movie102    7.905065e-04
dtype: float64
```

In [145…
```
ds_rmse = pd.concat([ds_pivot.mean(), ds_pred.mean()], axis=1)
ds_rmse.columns = ['Avg_actual_ratings', 'Avg_predicted_ratings']
print(ds_rmse.shape)
ds_rmse.head()
```

```
(206, 2)
```

Out[145…

movies	Avg_actual_ratings	Avg_predicted_ratings
Movie1	0.001031	-1.143966e-16
Movie10	0.001031	-2.371565e-19
Movie100	0.000825	2.546512e-16
Movie101	0.005157	2.818410e-16
Movie102	0.001650	7.905065e-04

In [146…
```
RMSE = round((((ds_rmse.Avg_actual_ratings - ds_rmse.Avg_predicted_ratings) ** 2).mean() ** 0.5), 5)
```

```
print('\nRMSE SVD Model = {} \n'.format(RMSE))
```

```
RMSE SVD Model = 0.00669
```

Getting top - K (K = 5) recommendations:

In [148…

```
# Enter 'user_index' and 'num_recommendations' for the user
user_index = 2333
num_recommendations = 5
recommend_items(user_index, ds_pivot, ds_pred, num_recommendations)
```

```
Below are the recommended Movies for user(user_index = 2333):

                  user_ratings  user_predictions
Recommended Movie
Movie202                   0.0          0.002292
Movie132                   0.0          0.002289
Movie188                   0.0          0.002270
Movie189                   0.0          0.002269
Movie190                   0.0          0.002268
```

Insight:

This user-based Collaborative Filtering model is a personalised recommender system, the recommendations are based on the past behavior of the selected users.