

## Mini Project

Name:-Anjali Vilas Abhang

Subject:- Artificial Intelligence

Project Name:- Recommendation System

Roll No:- 2337059

Batch:- C

---

Implementation:-

# 1. Import the libraries

In [19]:

```
#import the required libraries from
IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all" import
numpy as np import pandas as pd import math import json
import time import matplotlib.pyplot as plt import seaborn
as sns from sklearn.metrics.pairwise import
cosine_similarity from sklearn.model_selection import
train_test_split from sklearn.neighbors import
NearestNeighbors from sklearn.externals import joblib
import scipy.sparse from scipy.sparse import csr_matrix
from scipy.sparse.linalg import svds
import warnings; warnings.simplefilter('ignore') %matplotlib
inline
```

# 2. Load the dataset and add headers

In [20]:

```
# Import the dataset and give the column names columns=['userId',
'productId', 'ratings','timestamp']
electronics_df=pd.read_csv('ratings_Electronics.csv',names=columns)
```

In [21]: electronics\_df.head()

Out[21]:

**userId productId ratings timestamp 0 AKM1MP6P0OYPR**

**0132793040 5.0 1365811200**

1	A2CX7LUOHB2NDG	0321732944	5.0	1341100800
2	A2NWSAGRHCP8N5	0439886341	1.0	1367193600
3	A2WNBOD3WNDNKT	0439886341	3.0	1374451200
4	A1GI0U4ZRJA8WN	0439886341	1.0	1334707200

## Dropping the timestamp column

In [22]:

```
electronics_df.drop('timestamp',axis=1,inplace=True)
```

In [23]: electronics\_df.info()

```
<class 'pandas.core.frame.DataFrame'> RangeIndex:
7824482 entries, 0 to 7824481
Data columns (total 3 columns):
userId          object productId
object ratings   float64
dtypes: float64(1), object(2)
memory usage: 179.1+ MB
```

In [24]:

```
#Check the number of rows and columns
rows,columns=electronics_df.shape
print('Number of rows: ',rows) print('Number
of columns: ',columns)

Number of rows: 7824482
Number of columns: 3
```

In [25]:

```
#Check the datatypes electronics_df.dtypes
```

Out[25]:

```
userId          object productId          object ratings          float64 dtype:
object
```

In [26]:

```
#Taking subset of the dataset
electronics_df1=electronics_df.iloc[:50000,0:]
```

- Since the data is very big. Consider electronics\_df1 named dataframe with first 50000 rows and all columns from 0 of dataset.

In [27]: electronics\_df1.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 3 columns):
userId          50000 non-null object
productId       50000 non-null object
ratings         50000 non-null float64
```

```
dtypes: float64(1), object(2) memory
usage: 1.1+ MB
```

In [28]:

```
#Summary statistics of rating variable
electronics_dfl['ratings'].describe().transpose()
```

Out[28]:

```
count      50000.00000 mean          4.03524 std          1.35555 min
1.00000 25%          4.00000
50%          5.00000 75%
5.00000 max
5.00000
Name: ratings, dtype: float64
```

In [29]:

```
#Find the minimum and maximum ratings print('Minimum rating is:
%d' %(electronics_dfl.ratings.min())) print('Maximum rating is:
%d' %(electronics_dfl.ratings.max()))
```

```
Minimum rating is: 1
Maximum rating is: 5
```

- Rating are on the scale 1 to 5.

## Handling Missing values

In [30]:

```
#Check for missing values print('Number of
missing values across columns:
\n',electronics_df.isnull().sum())

Number of missing values across columns:
  userId      0
productId    0
ratings      0 dtype:
int64
```

- There are no missing records in the dataset.

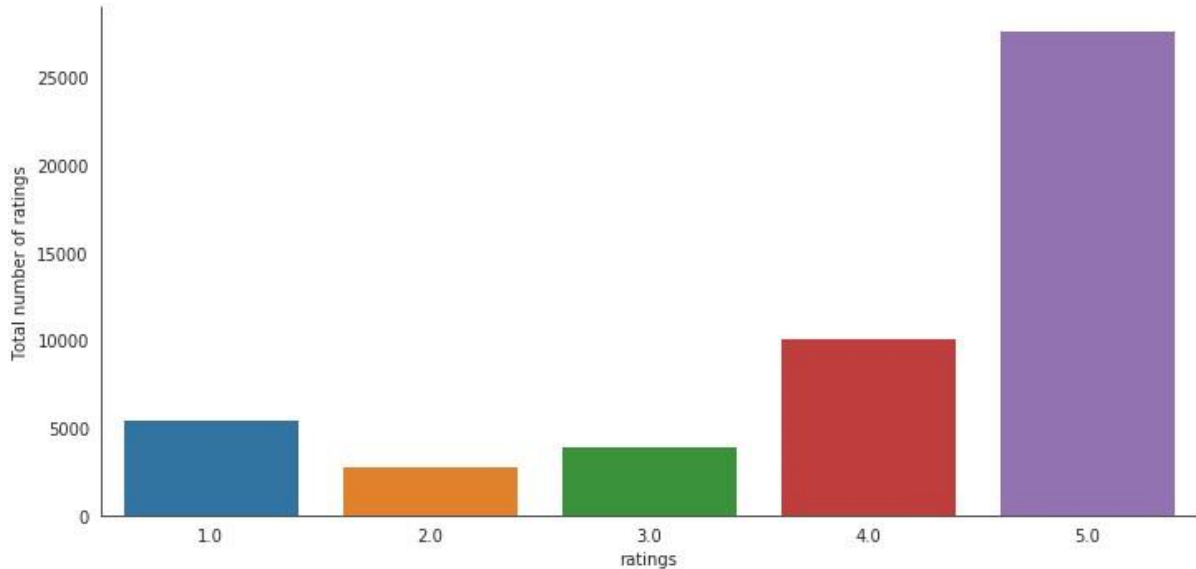
## Ratings

In [31]:

```
# Check the distribution of the rating with
sns.axes_style('white'):
    g = sns.factorplot("ratings", data=electronics_dfl,
aspect=2.0,kind='count')
    g.set_ylabels("Total number of ratings")
```

Out[31]:

```
<seaborn.axisgrid.FacetGrid at 0x247bbfdbcb18>
```



- We can see that more number of users have given the rating of 5.

## Users and products

In [34]:

```
# Number of unique user id in the data
print('Number of unique users in Raw data = ',
electronics_df1['userId'].nunique()) # Number of
unique product id in the data print('Number of
unique product in Raw data = ',
electronics_df1['productId'].nunique())

Number of unique users in Raw data = 46554
Number of unique product in Raw data = 3446
```

## 3. Taking the subset of dataset to make it less sparse/ denser.

In [35]:

```
#Check the top 10 users based on ratings
most_rated=electronics_df1.groupby('userId').size().sort_values(ascending=False)
print('Top 10 users based on ratings: \n',most_rated)

Top 10 users based on ratings:
  userId
A231WM2Z2JL0U3    37
AY8Q1X7G96HV5     31
ALUNVOQRXOZIA     20
A1NVD0TKNS1GT5    19
A243HY69GIAHFI    18
A1RPTVW5VEOSI     17
A1ISUNUWG0K02V    16
A1MJMYLRTZ76ZX    16
```

In [36]:

- electronics\_df1\_final has the users who have rated 25 or more items.

## ratings analysis in final dataset

In [37]:

```
#constructing the pivot table final_ratings_matrix =
electronics_dfl_final.pivot(index = 'userId', columns = 'productId',
values = 'ratings').fillna(0) final_ratings_matrix.head()
```

Out[37]:

	1BBBBBBBBBBBBBBBBBBB400000000000000000	
	000000000000000000pr	
	000000000000000000od.	
	500000000000000000uc.	
	90000000000444444444	
tI	.	
	9DJJJJJJJTTTTTTTTTTT	
d9M00001133DEEHHHHHHH		7
9968AEQUHHW7N22CCCD		7
	7M1C28ZS5BY52WYXYZE	7
us		
er		
Id		
A		
I		
S		
U		
N		

	0	0	0	0	0	3	0	5	5	.	0	0
	0 U	0.	0.	0.	0.	0.	0.	0.	0.	.	.	.
	.	.	.	.	.	.	.	.	.			
W	0						0		0	0	0	0
	0	0	0	0	0	0	0	0	.	0	0	0

G  
0  
K  
02  
V

A  
1  
M  
J  
M

	0	0	0	0	0	0	0	0	0	.	0	0
	0 Y	0.	0.	0.	0.	0.	0.	0.	0.	.	.	.
	.	.	.	.	.	.	.	.	.			
L	0						0		0	0	0	0
	0	0	0	0	0	0	0	0	.	0	0	0

R  
T  
Z7  
6Z  
X

A  
1 0 0. 0 0 0 0 0 0 0 0 0 . 0. 0 0 0. 0 0. 0. 0. 0. 0.  
N.0.....0..0.00000  
V000000000.000

D  
1BBBBBBBBBBBBBBBBBBBB40000000000000000000  
00000000000000000000pr  
00000000000000000000od .  
50000000000000000000uc .  
90000000004444444444tI .  
9DJJJJJJJTTTTTTTTTTd

9 M 0 0 0 0 1 1 3 3 D E E H H H H H H H  
9 9 6 8 A E Q U H H W 7 N 2 2 C C C D M  
7 M 1 C 2 8 Z S 5 B Y 5 2 W Y X Y Z E 6

us  
er  
Id

0T  
K  
N  
S1  
G  
T5

A  
1  
R  
P  
T  
0 5 0 0 0 0 0 0 0 0 . 0 0  
0 V 0. 0. 5. 0. 0. 0. 0. 0. . . .  
W 0 . . . . . 0 . 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 . 0 0 0

5  
V  
E  
O  
SI

A  
23  
1  
W  
0 0 0 0 0 0 5 0 0 . 5 5 4  
M 0. . . . . . . . . 0. . 0. 0. 0. 5. 0.  
2Z 0 . . . . . . . . 0 0 0 0 0  
0 0 0 0 0 0 0 0 0 . 0 0 0

2J  
L0  
U

### 3

5 rows × 186 columns

- It shows that it is a sparse matrix. So, many cells are filled with 0 values.

In [38]:

```
print('Shape of final_ratings_matrix: ', final_ratings_matrix.shape)
Shape of final_ratings_matrix: (11, 186)
```

- We can see that there are 7 products and 236 users.

In [39]:

```
#Calculating the density of the rating matrix
given_num_of_ratings = np.count_nonzero(final_ratings_matrix)
print('given_num_of_ratings = ', given_num_of_ratings)
possible_num_of_ratings = final_ratings_matrix.shape[0] *
final_ratings_matrix.shape[1] print('possible_num_of_ratings
= ', possible_num_of_ratings) density =
(given_num_of_ratings/possible_num_of_ratings) density *= 100
print ('density: {:.2f}%'.format(density))
given_num_of_ratings = 219 possible_num_of_ratings = 2046
density: 10.70%
```

- The density value of the matrix also shows that it is a sparse matrix.

## 4. Splitting the data

In [40]:

```
#Split the data randomly into train and test datasets into 70:30 ratio
train_data, test_data = train_test_split(electronics_dfl_final, test_size =
0.3, random_state=0) train_data.head()
```

Out[40]:

**userId productId ratings 17509 AY8Q1X7G96HV5**

B00000JSES 4.0

**11968** A243HY69GIAHFI B00000J3Q7 3.0

**35533** A1RPTVW5VEOSI B00003WGP5 5.0

**31480** A1NVD0TKNS1GT5 B00002JXFH 4.0

**13526** A23ZO1BVFFLGHO B00000J570 5.0



```
In [41]:
print('Shape of training data: ',train_data.shape) print('Shape of testing
data: ',test_data.shape)

Shape of training data:  (153, 3)
Shape of testing data:  (66, 3)
```

## 5. Building Popularity Recommender model

```
In [42]:
#Count of user_id for each unique product as recommendation score
train_data_grouped = train_data.groupby('productId').agg({'userId':
'count'}).reset_index()
train_data_grouped.rename(columns = {'userId': 'score'},inplace=True)
train_data_grouped.head(40)
```

Out[42]:

	productId	score
	1400599997	1
1	B00000DM9M	1
2	B00000J061	1
3	B00000J08C	1
4	B00000J1QZ	1
5	B00000J3HB	1
6	B00000J3II	2
7	B00000J3Q7	1
8	B00000J3T1	1
9	B00000J47A	1

**productId   score**

<b>10</b>	B00000J4ER	1
<b>11</b>	B00000J4FS	1
<b>12</b>	B00000J4O2	1
<b>13</b>	B00000J570	2
<b>14</b>	B00000J579	1
<b>15</b>	B00000JBIA	1
<b>16</b>	B00000JBJQ	2
<b>17</b>	B00000JBK6	1
<b>18</b>	B00000JBUI	1
<b>19</b>	B00000JDEI	1
<b>20</b>	B00000JFDW	1
<b>21</b>	B00000JFJA	1
<b>22</b>	B00000JFMW	1
<b>23</b>	B00000JHVP	1
<b>24</b>	B00000JI2C	1

	productId	score	
25	B00000JSES	1	
26	B00000JSEW	1	27 B00000JSGF 2
28	B00000K1SD	1	
29	B00000K390	1	
30	B00000K3RI	1	
31	B00000K3RO	1	
32	B00000K4KH	1	
33	B00001O2YP	1	
34	B00001OPK8	1	
35	B00001OWYM	1	
36	B00001P4ZH	1	
37	B00001QEMF	1	
38	B00001QHP5	1	
39	B00001RMCY	1	

In [43]:

```
#Sort the products on recommendation score
train_data_sort =
train_data_grouped.sort_values(['score', 'productId'], ascending =
[0,1])
```

**productId score**

```
#Generate a recommendation rank based upon score
train_data_sort['rank'] = train_data_sort['score'].rank(ascending=0,
method='first')
```

```
#Get the top 5 recommendations popularity_recommendations
= train_data_sort.head(5) popularity_recommendations
```

Out[43]:

		rank	
87	B00004RC2D	4	1.0
57	B00002SWHH	3	2.0
99	B00004SC3Y	3	3.0
100	B00004SCKA 3 4.0 6 B00000J3II 2 5.0		

In [44]:

```
# Use popularity based recommender model to make predictions
def recommend(user_id):
    user_recommendations = popularity_recommendations

    #Add user_id column for which the recommendations are being generated
    user_recommendations['userId'] = user_id

    #Bring user_id column to the front
    cols = user_recommendations.columns.tolist()
    cols = cols[-1:] + cols[:-1]
    user_recommendations = user_recommendations[cols]

    return user_recommendations
```

In [45]:

```
find_recom = [10,100,150] # This list is user choice.
for i in find_recom:
    print("The list of recommendations for the userId: %d\n" % (i))
    print(recommend(i))
    print("\n")

The list of recommendations for the userId: 10
  userId  productId  score  rank
87      10  B00004RC2D      4   1.0
57      10  B00002SWHH      3   2.0
```

**productId score**

99	10	B00004SC3Y	3	3.0
100	10	B00004SCKA	3	4.0
6	10	B00000J3II	2	5.0

The list of recommendations for the userId: 100

	userId	productId	score	rank
87	100	B00004RC2D	4	1.0
57	100	B00002SWHH	3	2.0
99	100	B00004SC3Y	3	3.0
100	100	B00004SCKA	3	4.0

```
6          100  B00000J3II          2    5.0
```

The list of recommendations for the userId: 150

	userId	productId	score	rank
87	150	B00004RC2D	4	1.0
57	150	B00002SWHH	3	2.0
99	150	B00004SC3Y	3	3.0
100	150	B00004SCKA	3	4.0
6	150	B00000J3II	2	5.0

- Since, it is a Popularity recommender model, so, all the three users are given the same recommendations. Here, we predict the products based on the popularity. It is not personalized to particular user. It is a non-personalized recommender system.

## 6. Building Collaborative Filtering recommender model.

```
In [46]:  
electronics_df_CF = pd.concat([train_data, test_data]).reset_index()  
electronics_df_CF.head()
```

Out[46]:

```
index  userId  productId  ratings  0 17509
```

```
AY8Q1X7G96HV5 B00000JSES 4.0
```

```
1      11968  A243HY69GIAHFI  B00000J3Q7  3.0
```

```
2      35533  A1RPTVW5VEOSI  B00003WGP5      5.0
```

```
3      31480  A1NVD0TKNS1GT5      B00002JXFH 4.0
```

```
4      13526  A23ZO1BVFFLGHO      B00000J570  5.0
```

## User Based Collaborative Filtering model

In [47]:

1BBBBBBBBBBBBBBBBBBBB40000000000000000000  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

pr

00000000000000000000od .

```
# Matrix with row per 'user' and column per 'item' pivot_df =  
electronics_df_CF.pivot(index = 'userId', columns = 'productId', values =  
'ratings').fillna(0) pivot_df.head()
```

Out[47]:

1BBBBBBBBBBBBBBBBBBBB400000000000000000000  
0 00000000000000000000pr  
00000000000000000000od .  
5 00000000000000000000uc .  
90000000000444444444tI .  
9DJJJJJJJTTTTTTTTTTd  
9 M 0 0 0 0 1 1 3 3 D E E H H H H H H  
9 9 6 8 A E Q U H H W 7 N 2 2 C C C D M  
7M 1 C 2 8 Z S 5 B Y 5 2 WY X Y Z E 6

us

er

Id

A  
1I  
S  
U  
N

0 0 0 0 0 3 0 5 5 . 0 0  
0 U 0. 0. 0. 0. 0. 0. 0. 0.. . .  
. . . . . . . . .  
W 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

G  
0  
K  
02  
V

A  
1  
M  
J  
M

0 0 0 0 0 0 0 0 0 . 0 0  
0 Y 0. 0. 0. 0. 0. 0. 0. 0.. . .  
. . . . . . . . .

L 0  
0  
R  
T  
Z7  
6Z  
X

A 0  
1 . 0. . . . . . . . . . 0. . . 0. . 0. 0. 0. 0. 0.  
N 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 V

B  
0  
0  
0

50000000000000000000uc .  
900000000004444444444tI .  
9DJJJJJJJJJTTTTTTTTTTTd  
9 M 0 0 0 0 1 1 3 3 D E E H H H H H H H  
9 9 6 8 A E Q U H H W 7 N 2 2 C C C D M  
7M 1 C 2 8 Z S 5 B Y 5 2 WY X Y Z E 6

us  
er  
Id

D  
0T  
K  
N  
S1  
G  
T5

A  
1  
R  
P  
T



[illegible]



1BBBBBBBBBBBBBBBBBB40000000000000000000  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

pr

00000000000000000000000000od .  
9000000000004444444444itl .  
9DJJJJJJJTTTTTTTTTTn

d  
9M00001133EEHHHHHHHd9968AEQUHH7N22CCCD  
Me

7 M 1 C 2 8 Z S 5 B 5 2 WY X Y Z E 6 x

us

er

Id

A  
II  
S  
U  
N

0 0 0 0 0 3 0 5 5 . 0 0 0  
U 0. 0. 0. 0. 0. 0.  
W 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

G  
0  
K  
02  
V

A  
1  
M  
J  
M

0 0 0 0 0 0 0 0 0 . 0 0 0  
Y 0. 0. 0. 0. 0. 0.  
L 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
R 0 0 0 0 0 0 0 0 . 0 0 0

R

**u  
s  
e  
r**

# ART

[illegible]

5 rows × 187 columns

In [50]:

```
pivot_df.set_index(['user_index'], inplace=True)
# Actual ratings given by users pivot_df.head()
```

Out[50]:

d u c  I   u s e r  - i	1	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	p	
	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	r	
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	o	
	0	0	0	0	0	0	0	0	0	0	0	0	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	.	
	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	.	
	9	D	J	J	J	J	J	J	J	J	T	T	T	T	T	T	T	T	T	T	T	t
	9	M	0	0	0	0	1	1	3	3	D	E	E	H	H	H	H	H	H	H	H	
	9	9	6	8	A	E	Q	U	H	H	W	7	N	2	2	C	C	C	D	M		
	7	M	1	C	2	8	Z	S	5	B	Y	5	2	W	Y	X	Y	Z	E	6		

n  
d  
e  
x

000003055.0000.0.0.0.0.0.0.0.0.0.0.0.0.0.....  
0  
0 0 0 0 0 0 0 0 0 0 . 0 0

0000000000.0000.0.0.0.0.0.0.0.0.0.0.0.1.....  
0  
0 0 0 0 0 0 0 0 0 0 . 0 0

0000000000.0000.0.0.0.0.0.0.0.0.0.0.0.2.....  
0  
0 0 0 0 0 0 0 0 0 0 . 0 0

0500000000.0000.0.5.0.0.0.0.0.0.0.0.3.....  
0  
0 0 0 0 0 0 0 0 0 0 0 0 0 . 0 0

1 BBBB BBBB BBBB BBBB BBBB p  
400000000000000000000000r  
000000000000000000000000o  
0 0

d .  
5 0  
u .  
9 0 0 0 0 0 0 0 0 0 0 4 4 4 4 4 4 4 4 4 4 4

c .  
9DJJJJJJJJTTTTTTTTTTTTt  
9 M 0 0 0 0 1 1 3 3 D E E H H H H H H H

I  
9968AEQUHHW7N22CCCDM  
d  
7 M 1 C 2 8 Z S 5 B Y 5 2 WY X Y Z E 6

u  
s  
e  
r  
-  
i

# **n d e x**

0 0 0 0 0 5 0 0 . 5 5 0 . 0 0 . 4 . 0 . 0 . 0 . 5 . 0 . 4 .....  
           0   0          0    0    0    0    0    0  
 0          0    0    0    0    0    0    0    0    .        0    0

5 rows  $\times$  186 columns

- As this is a sparse matrix we will use SVD.

## Singular Value Decomposition

In [51]:

```
# Singular Value Decomposition
U, sigma, Vt = svds(pivot_df, k = 10)
```

In [52]:

```
print('Left singular matrix: \n',U)
Left singular matrix:
[[ 2.65871898e-01  8.28788249e-01  1.14160900e-01 -2.15685919e-01
 -2.65928896e-04 -4.93636653e-02  1.66297884e-01 -3.12732051e-01
 1.53122256e-01  1.38465143e-01]
 [ 5.53711304e-02  1.21204397e-01 -8.17836609e-02  3.29682359e-01
 -1.47955175e-01 -8.18019500e-01 -2.13209931e-01  2.76257471e-01
 2.09061119e-01  1.01498819e-01]
 [ 5.46679471e-01 -1.37512688e-01  2.79596983e-02  1.45212122e-01
 -3.57798248e-02  1.89642626e-01 -6.60636300e-01 -2.07100247e-01
 3.59790632e-02  1.23044532e-01]
 [ 4.63168744e-02  5.39880164e-02 -8.32935752e-01  1.36292972e-01
```



```

1.12974956e-02  9.74799278e-02 -5.43326589e-02 -2.99913518e-01  -
1.00398248e-01  1.16155900e-01]
[-4.82857184e-02 -5.06969281e-03  6.10478781e-02  7.63553438e-02
 8.79528911e-02  1.33412578e-01  9.24837886e-02  2.48586367e-01
1.79064339e-01  9.26201600e-01]
[-1.33072419e-01 -3.30520700e-01 -8.58433108e-03 -6.82184355e-01
-3.10981905e-02 -4.06414738e-01 -1.22417282e-01 -4.14339191e-01
 4.04459226e-03  2.33218097e-01]
[-5.85058830e-01  2.82858362e-01  2.67444902e-01  1.03926107e-01
-7.39796949e-02  1.25896947e-01 -5.86529180e-01 -1.64067216e-01
9.37707797e-03  1.79755959e-02]
[-5.04793124e-01  1.13702295e-01 -3.57486565e-01  6.92659013e-02
 2.00732743e-02  4.29965080e-02  5.56583151e-04 -2.51788542e-02
9.00894193e-04  3.04631617e-02]
[ 1.37401523e-02  1.68164402e-03 -5.23514746e-02 -1.04867805e-01
-9.72327480e-01  1.54469490e-01  6.89319218e-02  9.60100045e-02
2.28758644e-02  4.64399549e-02]
[-7.77524293e-02 -2.36994718e-01  2.76662643e-01  5.39661989e-01
-1.21856053e-01 -7.39304558e-02  3.36795340e-01 -6.29018917e-01  -
1.54119324e-01  1.21495612e-01]
[-4.31106769e-02 -1.34978010e-01  5.75166098e-02 -1.26848095e-01
 4.30464817e-02  2.30094616e-01 -4.18979575e-02  1.64726898e-01
-9.30068553e-01  1.07164263e-01]]

```

In [53]:

```

print('Sigma: \n',sigma)
Sigma:
[15.65710264 16.05091576 17.43121071 17.74252629 18.18622003 18.83061496
19.14527912 19.94315139 22.99328895 28.83750492]

```

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

In [54]:

```

# Construct diagonal array in SVD
sigma = np.diag(sigma) print('Diagonal
matrix: \n',sigma)
Diagonal matrix:
[[15.65710264  0.          0.          0.          0.          0.
  0.          0.          0.          0.          ]
 [ 0.          16.05091576  0.          0.          0.          0.
  0.          0.          0.          0.          ]
 [ 0.          0.          17.43121071  0.          0.          0.
  0.          0.          0.          0.          ]
 [ 0.          0.          0.          17.74252629  0.          0.
  0.          0.          0.          0.          ]
 [ 0.          0.          0.          0.          18.18622003  0.
  0.          0.          0.          0.          ]
 [ 0.          0.          0.          0.          0.          18.83061496
  0.          0.          0.          0.          ]
 [ 0.          0.          0.          0.          0.          0.
 19.14527912  0.          0.          0.          ]
 [ 0.          0.          0.          0.          0.          0.
 0.          19.94315139  0.          0.          ]

```

```
[ 0.      0.      0.      0.      0.      0.
0.      0.      22.99328895 0.      ]
[ 0.      0.      0.      0.      0.      0.
0.      0.      0.      28.83750492]]
```

In [55]:

```
print('Right singular matrix: \n',Vt)
Right singular matrix:
[[-0.00496595 -0.01376713  0.01479101 ... -0.0110137 -0.01541975 -
0.0110137 ]
[-0.01476518 -0.04204683  0.01681774 ... -0.03363746 -0.00157925 -
0.03363746]
[ 0.01587168  0.01649817 -0.2389208 ...  0.01319853  0.01751108
0.01319853]
...
[-0.0315406  0.04129911 -0.07519211 ...  0.03303929  0.06232374
0.03303929]
[-0.0067028 -0.20224783 -0.02183208 ... -0.16179826  0.03893839 -
0.16179826]
[ 0.00421311  0.01858071  0.02013973 ...  0.01486457  0.16058976
0.01486457]]
```

In [56]:

```
#Predicted ratings all_user_predicted_ratings =
np.dot(np.dot(U, sigma), Vt)
# Convert predicted ratings to dataframe preds_df =
pd.DataFrame(all_user_predicted_ratings, columns =
pivot_df.columns) preds_df.head()
```

Out[56]:

```
1BBBBBBBBBBBBBBBBBBBBBp
400000000000000000000r
000000000000000000000o
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
d
5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
u
9 0 0 0 0 0 0 0 0 0 0 4 4 4 4 4 4 4 4 4
c
9DJJJJJJJTTTTTTTTTTt
9 M 0 0 0 0 1 1 3 3 D E E H H H H H H
I
9968AEQUHHW7N22CCCDM
d
7 M 1 C 2 8 Z S 5 B Y 5 2 WY X Y Z E 6

- - - -
- 0 2 0 4 4 - 0 - - - 0 0 0 0 0 0 0 0 0 0
0. . . . . 0. . 0. 0. 0. 0.
. 0 2 . . . 9 0 9 9 0 0 0 2 0 0 0 0
```

	0		00	0	.	2	1	2	2										
0	1		16	2	3	3	2	2	0	0	0	0							
	0 1	2	0	.	6	9	1	6											
	0	9			2	6	8	8	5	6				8	6	8		8	
5			0	0	6			.			5	8	2				5		
	0	8				9	5	2	2	3	5				0	0	0		0
	0	0	3	5	9	6	7	9											
	2	6	2	9	1	1	8	9	1		1		1		1				
7			2	1	6						4	4	5					4	
	3	4	9	4	6	6	7	4	8		4		8		8				
7			3	0	2														

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	1	0	0	0	0	0	
	0	0		1	0	0	0	0	0	0		0	0	0	0	0	0	0	0		
	1		1	1	0	0	0	1	0												
	1	<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>	
	<b>B</b>	<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>		<b>B</b>	<b>B</b> p								

	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	r
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	o
d	0	0	0	0	0	0	0	0	0	0		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	
u											.									
c	9	0	0	0	0	0	0	0	0	0	.	4	4	4	4	4	4	4	4	
											.									
	9	D	J	J	J	J	J	J	J	J	T	T	T	T	T	T	T	T	T	t
I	9	M	0	0	0	0	1	1	3	3		D	E	E	H	H	H	H	H	H
	9	9	6	8	A	E	Q	U	H	H		W	7	N	2	2	C	C	C	D
d																				
	7	M	1	C	2	8	Z	S	5	B		Y	5	2	W	Y	X	Y	Z	E
																				6

	0	0	1	0	0	0	0	1	0	0		1	1	3	3	0	0	0	0	3
	0	5	3	0	1	0	1	3	3	3		3	3	6	0	9	4	3	4	6
	2	1	0	5	0	3	9	6	1	1		0	6	7	0	3	1	0	1	7
6	5		0		1		4		3	0		7		7		7		5		7
	9		2		2	1		5		4		7		5		6		6		
	-			--		-		-		-		-								
	-			00		-		0		-		-		-		-				

0		0	0	0	0		0	0		0.	0.	0.		0.					
	0.	.					.			0.	.			0.	0.	0.		0.	
.			.	.	.	.		.	.			0	7	0			0		
	0	7					0			0	0				0	0	0		0
0			0	0	0	1		1	1	.		8	0	6				8	
	3	0					8				8	8			2	1	2		2
<b>2</b>			1	3		6	2	1		9	9	.			5	5		8	
				5															
	2	5					5			1	5				5	9	5		5
6			2	5	1	9		8	8	.			3	8	2			3	
	1	8					3				5	3				7	3	7	7
<b>3</b>			1	2		0	0	3		3		7		3		9		7	
	7	3					7			0	7					4	0	4	4
0			7	0	6	0		4	4				4	9	9			4	
	7	9					4			2	4					2	6	2	2
0			7	1	6	9		9	9										

							-			-									
0		4	0	0	0	0		0	0		-		-						
	0.		0		0.		4.		0.		0.		0.		0.		.		.
.		.		.	.	.	0.		0.		0.								
	0						.			0	.		2		0	0	0		0
0		2		0		0	0	1	2	2		0		0	0	0			
	3	0.	9	0	1	2	2	2	2										
1																			
<b>3</b>		5	9			0	9	7	8	1		8		8					
8		7		5		2	3	1	9	9		4		5		4			
	6						4		.	3	4		5		5	4	5		5
0		5	6	2	3	9		8	8		6		7				6		
	6						6			4	6		9		3	0	3		3
6						9	6	7	5	1		6		6		3		0	
							3												
7						3	2	3	9	4		0		4		4			
8						9	7	4	1	8		4		4		5		8	
							5												
							5			5									

0			0	0	0	0	4	0	0		4		-						
	0.																		
.		-	.	.	.	.	.	.	.	0.	.	4.	0.	3.	0.	0.	0.	4.	0.
	0																		
0		0	0	0	0	0	9	0	0		0	9	9	0	9	0	0	0	9
	0																		
0		.	0	0	0	1	8	2	2	.	1	8	8	9	9	0	0	0	8

```

4      4      0      4      8      2      5      8      6      6      .      0      8
      8      4      0      3      2      3      8      3
2
      3
1      9      3      7      8      9      5      5      5      .      9      5      5      6      8      4      5      4      5      4
      1
8      4      1      4      2      5      5      9      9      2      5      5
      3      4      5      8      5      5      5
4
5      6      4      2      4      6      3      4      4      7      3      3      5
      3      1      8      1      3      1
1BBBBBBBBBBBBBBBBBBBBBp
4000000000000000000000r
0000000000000000000000o
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
d
5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
u
9 0 0 0 0 0 0 0 0 0 0 0 4 4 4 4 4 4 4 4 4
c
9DJJJJJJJTTTTTTTTTTt
9 M 0 0 0 0 1 1 3 3 D E E H H H H H H H
I
9968AEQUHHW7N22CCCDM
d
7 M 1 C 2 8 Z S 5 B Y 5 2 WY X Y Z E 6

3
5

```

5 rows × 186 columns

In [57]:

```

# Recommend the items with the highest predicted ratings
def recommend_items(userID, pivot_df, preds_df,
num_recommendations):
    # index starts at 0
    user_idx = userID-1
    # Get and sort the user's ratings
    sorted_user_ratings =
pivot_df.iloc[user_idx].sort_values(ascending=False)
    #sorted_user_ratings      sorted_user_predictions
=
preds_df.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 Items'      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 items for user(user_id =
{}):\n'.format(userID))
print(temp.head(num_recommendations))
```

In [58]:

```
userID = 4 num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
Below are the recommended items for user(user_id = 4):
```

	user_ratings	user_predictions	Recommended
Items			
B00001P4XA	0.0	1.526746	
B00000JYLO	0.0	1.526746	
B00004T1WX	0.0	1.526746	
B00004T1WZ	0.0	1.526746	
B00000J4ER	0.0	1.526746	

In [59]:

```
userID = 6 num_recommendations
= 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
Below are the recommended items for user(user_id = 6):
```

	user_ratings	user_predictions	Recommended
Items			
B00004RERZ	0.0	0.038989	
B00000J061	0.0	0.038989	
B00003WGP5	0.0	0.038989	
B00004TH2W	0.0	0.038989	
B00004S9WQ	0.0	0.038989	

In [60]:

```
userID = 8 num_recommendations = 5
recommend_items(userID, pivot_df, preds_df, num_recommendations)
Below are the recommended items for user(user_id = 8):
```

	user_ratings	user_predictions	Recommended
Items			
B00000J061	0.0	1.526746	
B00003WGP5	0.0	1.526746	
B00004TH2W	0.0	1.526746	
B00004RERZ	0.0	1.526746	
B00004RIPE	0.0	1.526746	

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

## 7. Evaluation of Collaborative recommendation model

In [61]:

```
# Actual ratings given by the users final_ratings_matrix.head()
```

Out[61]:

```
1BBBBBBBBBBBBBBBBBBBB4000000000000000000000
0000000000000000000000pr
0000000000000000000000od .
5000000000000000000000uc .
9000000000004444444444tI .
9DJJJJJJJJJTTTTTTTTTTTd
9 M 0 0 0 0 1 1 3 3 D E E H H H H H H H
9 9 6 8 A E Q U H H W 7 N 2 2 C C C D M
7M 1 C 2 8 Z S 5 B Y 5 2 WY X Y Z E 6
```

```
us
er
Id

A
II
S
U
N
0 0 0 0 0 3 0 5 5 . 0 0
0U 0. 0. 0. 0. 0. 0. 0. 0.. . .
. . . . . . . . .
W 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
G
0
K
02
V

A
1
M
J
M
0 0 0 0 0 0 0 0 0 . 0 0
0Y 0. 0. 0. 0. 0. 0. 0. 0.. . .
. . . . . . . . .
L 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
```

R  
T  
Z7  
6Z  
X

A  
1 0 0. 0 0 0 0 0 0 0 0 . 0. 0 0 0. 0 0. 0. 0. 0. 0.  
N.0.....0..0.00000V0000000000.000

D  
1BBBBBBBBBBBBBBBBBBBB40000000000000000000  
0000000000000000000000pr  
0000000000000000000000od .  
5000000000000000000000uc .  
900000000004444444444tI .  
9DJJJJJJJJTTTTTTTTTTTd  
9 M 0 0 0 0 1 1 3 3 D E E H H H H H H H  
9 9 6 8 A E Q U H H W 7 N 2 2 C C C D M  
7M 1 C 2 8 Z S 5 B Y 5 2 WY X Y Z E 6

us  
er  
Id

0T  
K  
N  
S1  
G  
T5

A  
1  
R  
P  
T  
0 5 0 0 0 0 0 0 0 . 0 0  
0V 0. 0. 5. 0. 0. 0. 0. 0. 0.. . .  
. . . . . . . . .  
W 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



V  
E  
O  
SI

A  
23  
1  
W

	0	0	0	0	0	0	5	0	0	.	5	5
4 M	0.	0.	0.	0.	0.	0.	0.	5.	0.	.	.	.
	.	.	.	.	.	.	.	.	.			
2Z	0						0		0	0	0	0
	0	0	0	0	0	0	0	0	.	0	0	0

2J  
L0  
U  
3

5 rows × 186 columns

In [62]:

```
# Average ACTUAL rating for each item
final_ratings_matrix.mean().head()
```

Out[62]:

```
productId
1400599997    0.090909
B00000DM9M    0.454545
B00000J061    0.454545
B00000J08C    0.454545
B00000J0A2    0.363636 dtype:
float64
```

In [63]:

```
# Predicted ratings  preds_df.head()
```

Out[63]:

1	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	p
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	r
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	o
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
d	5	0	0	0	0	0	0	0	0	0	.	0	0	0	0	0	0	0	0	
u	9	0	0	0	0	0	0	0	0	0	.	4	4	4	4	4	4	4	4	
c	9	D	J	J	J	J	J	J	J	J	J	T	T	T	T	T	T	T	T	t
I	9	M	0	0	0	0	1	1	3	3		D	E	E	H	H	H	H	H	H

9968AEQUHHW7N22CCCDM

d

7 M 1 C 2 8 Z S 5 B Y 5 2 WY X Y Z E 6

- - - -  
- 02044-0---00000.0.0.0.  
0. . . . . 0. . 0. 0. 0. 0.  
. 0 2 . . . 0 2 0 0  
0 0 2 9 0 9 9 0 0 0 0 0 0  
1 00 0 . 2 1 2 2 0 0 0 0  
0 0 1 2 0 . 6 9 1 6 0  
0 9 2 6 8 8 5 6 8 6 8 8  
5 0 0 6 . 5 8 2 5  
0 8 9 5 2 2 3 5 0 0 0  
1 0 3 5 9 6 7 9 1 1 1 1  
2 6 2 9 1 1 8 9 4 4 5 4  
7 3 4 9 4 6 6 7 4 8 4 8 8  
7 3 0 2

- - - - -  
- 00 - 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 0 0 0 . 0 1 0 0  
0 1 0 0 0 0 0 0 0 0 0 0 0  
1 0 0 0 0 0 0 0 . 1 1 1  
1 0 1 1 1 1 0 0 0 0  
0 0 1 0 1 3 3 . 3 3 0 3  
5 3 3 3 3 6 0 4 3 4 4  
2 5 0 3 9 1 1 6 0 9 6  
1 0 6 0 6 1 0 1 1  
6 1 4 3 0 7 7 7 0 3 7  
5 0 7 5 7 2 9 2 2  
1 5 4 7 5 6 6

1BBBBBBBBBBBBBBBBBBp

40000000000000000000r

0 00000000000000000000o

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

d

.

u  
 c  
 I  
 d  
 7

5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4
9	D	J	J	J	J	J	J	T	T	T	T	T	T	T	T	T	T	T	T	T
9	M	0	0	0	0	1	1	3	3		D	E	E	H	H	H	H	H	H	H
9	9	6	8	A	E	Q	U	H	H	W	7	N	2	2	C	C	C	D	M	
7	M	1	C	2	8	Z	S	5	B		Y	5	2	W	Y	X	Y	Z	E	6

-			-	-	-	-	-													
-			00	-	0	-	-				-		-							
0			0	0	0	0		0	0			0.	0.	0.					0.	
	0.	.						.			0.	.				0.	0.	0.		0.
.			.	.	.	.		.	.				0	7	0				0	
	0	7					0				0	0				0	0	0		0
0			0	0	0	1		1	1	.			8	0	6				8	
	3	0					8				8	8				2	1	2		2
2			1	3		6		2	1		9		9		.		5	5		8
				5																
	2	5					5				1	5				5	9	5		5
6			2	5	1	9		8	8	.			3	8	2				3	
	1	8					3				5	3				7	3	7		7
3			1	2		0		0	3		3		7		3		9	7		
	7	3					7				0	7				4	0	4		4
0			7	0	6	0		4	4				4	9	9				4	
	7	9					4				2	4				2	6	2		2
0			7	1	6	9		9	9											

0		4	0	0	0	0		0	0		-		-						-	
	0.	0		0.		0		4.	0.		0.	0.	0.		.		.		.	
.		.		.		.		0.	0.		0.									
	0						.				0	.		2		0	0	0		0
0		2		0		0		0	1		2		2		0		0	0		
	3	0.	9	0	1	2	2	2	2											
1			1	3	7	2	3	1	1	9	7	9								
3			5		9		.		0		9		7		8		1		8	8
8				7	5		2		3		1		9		9		4		5	4
	6								4			.	3	4		5		5	4	5
0		5	6	2	3	9			8	8			6		7				6	
	6							6					4	6		9		3	0	3



productId			
1400599997	0.090909	0.088513	0
B00000DM9M	0.454545	0.449816	1
B00000J061	0.454545	0.558292	2
B00000J08C	0.454545	0.449816	3
B00000J0A2	0.363636	0.354053	4

```
In [66]:
RMSE = round((((rmse_df.Avg_actual_ratings - rmse_df.Avg_predicted_ratings)
** 2).mean() ** 0.5), 5)
print('\nRMSE SVD Model = {} \n'.format(RMSE))
RMSE SVD Model = 0.05854
```

## 8. Getting top - K ( K = 5) recommendations.

```
In [70]:
# Enter 'userID' and 'num_recommendations' for the user # userID
= 9 num_recommendations = 5 recommend_items(userID, pivot_df,
preds_df, num_recommendations) Below are the recommended items
for user(user_id = 9):
```

	user_ratings	user_predictions	Recommended
Items			
B00003WGP5	0.0	0.023351	
B00000J061	0.0	0.023351	
B00004TH2W	0.0	0.023351	
B00001QHP5	0.0	0.023351	
B00004S9WQ	0.0	0.023351	

## Summarising insights.

- The Popularity-based recommender system is a non-personalised recommender system and these are based on frequency counts, which may be not suitable to the user. We can see the difference above for the user id 4, 6 & 8, The Popularity based model has recommended the same set of 5 products to both but Collaborative Filtering based model has recommended entire different list based on the user past purchase history.

- Model-based Collaborative Filtering is a personalised recommender system, the recommendations are based on the past behavior of the user and it is not dependent on any additional information.