# RECOMMENDER SYSTEM

## " People don't know what they want until you show it to them."

-Steve Jobs

# INTRODUCTION

## AIM

To Recommend a product to a user that the user is more likely to be intrested in .

The basic idea of a recommender system is to help users to discover new and relevant items based on their past experience. Recommender systems suggest items to users such as books,music,movies ,products etc.

Our goal of the project is to recommend a new product to a customer analyzing his/her past experience. We have taken Amazon dataset from data.gov. In the dataset we have user information like User name and a UserID. Also Product information like Product name,ProductId and Product description. We also have a rating to each product given by the user for the products that they have used.

## Audience

Audience of the project are the online customers

## Data Set

We have used amazon data set from data.gov

## Data Dictionary

Data Set has 68,015 rows and 12 columns Columns are .

'Id', 'ProductId', 'Product', 'UserId', 'ProfileName', 'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time', 'Count_Userid', 'Summary', 'Text'

## Models Used

Below models are used for recommendation

1)Popularity Based Recommender

2)Item based Recommendaion

3)User Based Recommendation

# MODELS

## Importing data

In [25]:

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


os.chdir("C:\\Users\\Ramya Shetty\\Desktop\\Rutgers\\IntroToDS\\FinalProject") #Setting data directory
data=pd.read_csv("FinalDataSetV1.csv") #importing the file
data.columns
```

```
C:\Users\Ramya Shetty\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:3020: DtypeWarnin
g: Columns (2) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[25]:

```
Index(['Id', 'ProductId', 'Product', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time',
       'Count_Userid', 'Summary', 'Text'],
      dtype='object')
```

In [132]:

```
#Splitting data into train and test for evaluationg the model
from sklearn.model_selection import train_test_split
train_data, test_data = train_test_split(data, test_size = 0.20, random_state=0)
```

## Popularity based recommendation system

Popularity based recommendation system works with the trend. Recommends products based on the number of count of reviews. This helps in recommendiong a new user or a user who has never reviewd any product before.

For example, if any product which is usually bought by every new user, then there are chances that item can be bought by the new user who just signed up.

In [133]:

```
#Popularity  trend based on the count of reviews

Popular_Product = data.groupby('Product')['Score'].count().sort_values(ascending=False).reset_index()
Popular_Product.head(5)


print("Based on our data the most popular products are ")
print( Popular_Product.head(5))
```

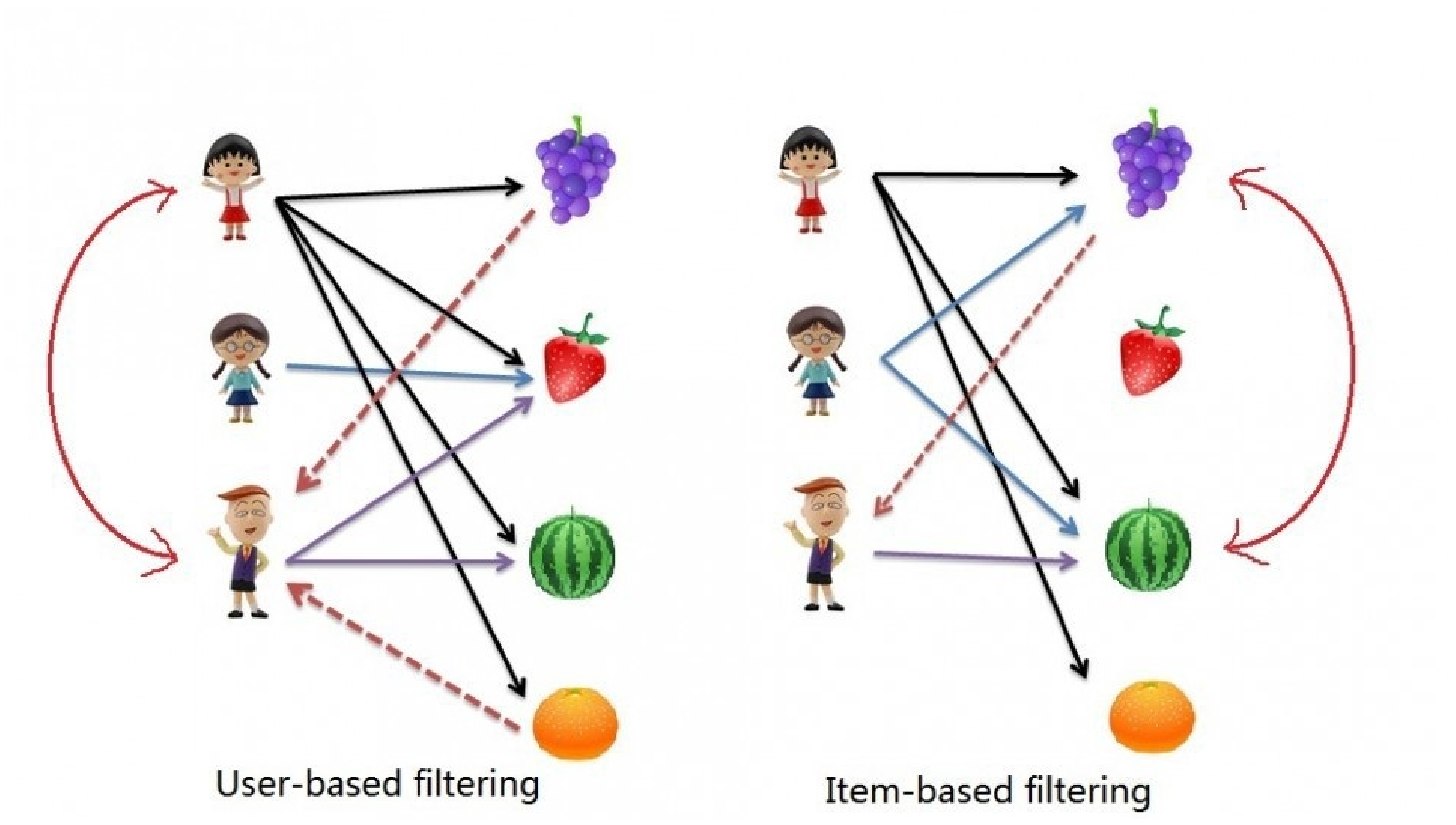
```
Based on our data the most popular products are 
                   Product  Score
0      Computers & Tablets    254
1                   Laptop    224
2                Tech Toys    201
3   E-Readers & Accessories    123
4     Household & Baby Care    122
```

## Collabrative Filtering

Collaborative Filtering (CF) is a mean of recommendation based on users' past behavior.There are two categories of CF: Item-Item Collaborative filtering. User-User Collaborative filtering.

Item based Collaborative filtering - Item to Item based collaborative filtering is based on similarity between items calculated using ratings of the user.

User based Collaborative filtering - User to User based collaborative filtering is based on similarity between users calculated using ratings of the user.



### Item based collabrative filtering

Item based Collaborative filtering - Item to Item based collaborative filtering is based on similarity between items calculated using ratings of the user.

Similarity is calculated based on the Pearson R Correlation. Correlation ranges from -1 to +1.

#### Code

In [26]:

```
df = data[['UserId','ProductId','Score']]
df = df.head(12000)

R_df = df.pivot_table(index = ['UserId'], columns =['ProductId'], values = 'Score')

def pearson(p1,p2):
    p1_c = p1 - p1.mean()
    p2_c = p2 - p2.mean()
    return np.sum(p1_c * p2_c)/ np.sqrt(np.sum(p1_c ** 2) * np.sum(p2_c ** 2))


def get_rec(ProductId,R_df, num):
    review = []
    for index in R_df.columns:
        if index == ProductId:
            continue

        cor = pearson(R_df[ProductId], R_df[index])
        if np.isnan(cor):
            continue
        else:
            review.append((index, cor))
    review.sort(key = lambda tup: tup[1], reverse = True)
    return review[:num]

recs = get_rec('B001GVISJM',R_df,5)
```

```
C:\Users\Ramya Shetty\Anaconda3\lib\site-packages\ipykernel_launcher.py:9: RuntimeWarning: invalid v
alue encountered in double_scalars
  if __name__ == '__main__':
```

#### Recommended Products

In [27]:

```
recs
```

Out[27]:

```
[('B000GZY9HY', 0.2886751345948126),
 ('B001CWZXIY', 0.16666666666666652),
 ('B0029JW9JU', 0.12961896184130303),
 ('B0040PYX90', 0.11785113019775781),
 ('B004MO6NI8', 0.0983146344786589)]
```

## Matrix Factorisation

Matrix Factorisation - Matrix factorization is a class of collaborative filtering algorithms used in recommender systems. Matrix factorization algorithms work by decomposing the user-item interaction matrix.

In [28]:

```
R_df = data.pivot_table(index = ['UserId'], columns =['ProductId'], values = 'Score').fillna(0)
R_df.head(10)
```

Out[28]:

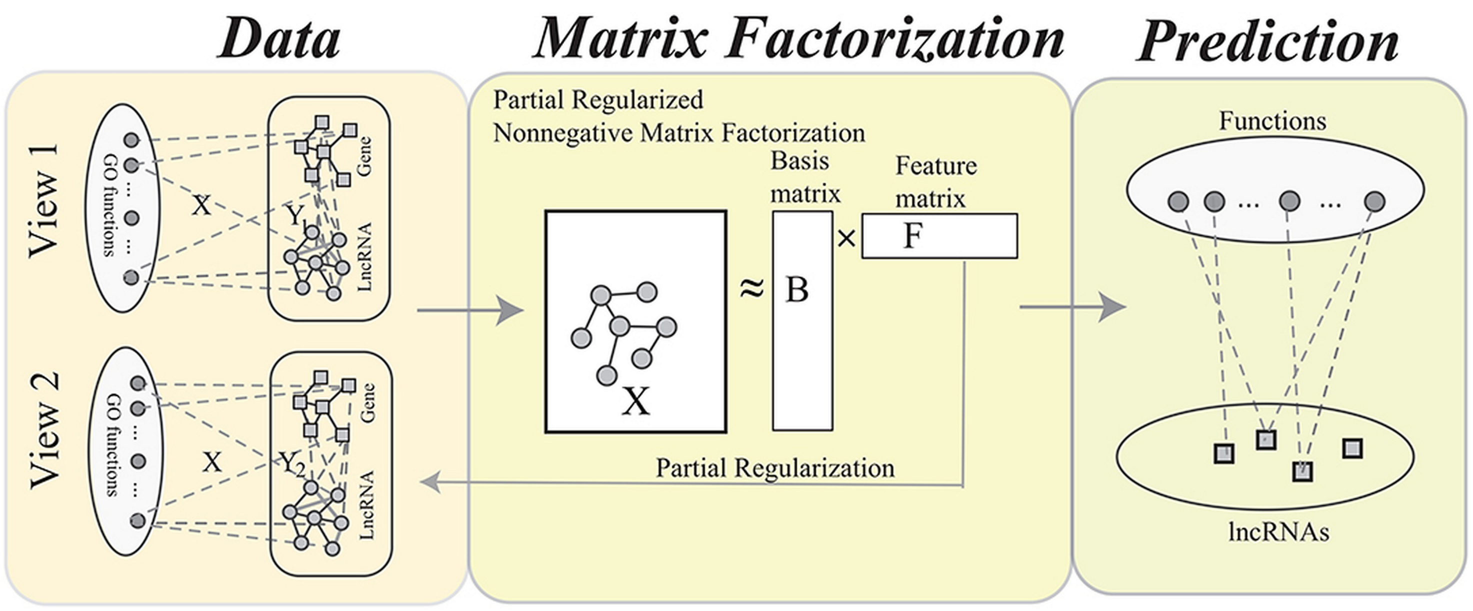
| ProductId | 7310172001 | 7310172101 | 7800648702 | B00004CI84 | B00004CXX9 | B00004RAMY | B00004RBDU | B00004RBDZ | B0000 |
|---|---|---|---|---|---|---|---|---|---|
| UserId | | | | | | | | | |
| A100WO06OQR8BQ | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | |
| A101P2KHWCU0G6 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| A1051DBTLWP5A2 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| A106ZCP7RSXMRU | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| A1080SE9X3ECK0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| A10AFVU66A79Y1 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| A10DRHWKS6NQUX | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| A10G136JEISLVR | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| A10H24TDLK2VDP | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |
| A10LIGIT9EGCM9 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | |

10 rows × 19488 columns

The Userid and the Productid are formed into a pivot table with ratings as values. The NaN values are converted into 0. The R_df matrix has the rating for given UserId and ProductId. Many values are missing in the dataset with few values available. In order to predict the missing values in the R_df matrix, We start with multiplication of two random matrices X and Y (for example). The product of X and Y will be equal to R_df. Random values are taken for X and Y and the error is calculated with the existing values in R_df. This process is repeated until the error is minimum and R_df matrix is filled. This is done using Single Value Decomposition.

**SVD**

Matrix factorization is the breaking down of one matrix into a product of multiple matrices.Singular value decomposition is particularly useful for making recommendations. SVD is an algorithm that decomposes a matrix X(Rating Matrix) into the best lower rank (i.e. smaller/simpler) approximation of the original matrix X. X is user ratings matrix, F is the user “features” matrix, Σ is the diagonal matrix of singular values (essentially weights), and B is the product “features” matrix. Using svd Matrix factorization we find 2(ie B and F) matrices whose product is the original matrix which is a sparse matrix with prediction of missing score for each user for defined product with best low-rank approximation



**Code**

In [29]:

```
import pandas as pd
import numpy as np

number_of__users = data.groupby("UserId", as_index=False).count()  #Gives the number of Users
#recommending for one user
number_of__users.loc[number_of__users['UserId'] == 'A10ZBR6O8S8OCY'] #To check numbers of products custome has re
viewd
number_of_Product= data.groupby("ProductId", as_index=False).mean() # Gives the count of products
```

In [30]:

```
userid_data = number_of__users[['UserId']] #count  all userid present in the datset
userid_data.head(10)
```

Out[30]:

| | UserId |
|---|---|
| 0 | A100WO06OQR8BQ |
| 1 | A101P2KHWCU0G6 |
| 2 | A1051DBTLWP5A2 |
| 3 | A106ZCP7RSXMRU |
| 4 | A1080SE9X3ECK0 |
| 5 | A10AFVU66A79Y1 |
| 6 | A10DRHWKS6NQUX |
| 7 | A10G136JEISLVR |
| 8 | A10H24TDLK2VDP |
| 9 | A10LIGIT9EGCM9 |

In [31]:

```
productid_data = number_of_Product[['ProductId']] #count  all products present in the datset
productid_data.head(10)
```

Out[31]:

| | ProductId |
|---|---|
| **0** | 7310172001 |
| **1** | 7310172101 |
| **2** | 7800648702 |
| **3** | B00004CI84 |
| **4** | B00004CXX9 |
| **5** | B00004RAMY |
| **6** | B00004RBDU |
| **7** | B00004RBDZ |
| **8** | B00004RYGX |
| **9** | B00004S1C6 |

In [32]:

```
data.columns
```

Out[32]:

```
Index(['Id', 'ProductId', 'Product', 'UserId', 'ProfileName',
       'HelpfulnessNumerator', 'HelpfulnessDenominator', 'Score', 'Time',
       'Count_Userid', 'Summary', 'Text'],
      dtype='object')
```

In [33]:

```
subset_data=data.head(5000) #create a subset of dataset
#create the pivot table using id, productId and rating score
Rating_pivot_table=subset_data.pivot(index="Id",columns="ProductId",values="Score").fillna(0)
```

In [35]:

```
#printing the ratings pivot table
Rating_pivot_table.head(10)
```

Out[35]:

| ProductId | B000084EK4 | B000084EK5 | B000084EK6 | B000084EK7 | B000084EK8 | B000084EK9 | B000084EKA | B000084EKB | B000084EKC |
|---|---|---|---|---|---|---|---|---|---|
| **Id** | | | | | | | | | |
| **14** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **15** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **19** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **45** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **47** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **74** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **110** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **142** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **144** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **145** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

10 rows × 1453 columns

In [36]:

```
#converting ratings pivot table to matrix
#this is ratings matrix with missing values
Ratings_data_matrix = Rating_pivot_table.as_matrix()
Ratings_data_matrix[:10]
```

```
C:\Users\Ramya Shetty\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: Method .as
_matrix will be removed in a future version. Use .values instead.
```

Out[36]:

```
array([[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.]])
```

In [37]:

```
#creating list of rating matrix ids
Ratings_data_ids = list(Rating_pivot_table.index)
Ratings_data_ids[:10]
```

Out[37]:

```
[14, 15, 19, 45, 47, 74, 110, 142, 144, 145]
```

In [38]:

```
#defining number of factors to factor the user-item matrix.
#k= how many latent factors
NUMBER_OF_FACTORS_MF = 15
```

In [39]:

```
#Performs matrix factorization of the original user item matrix
from scipy.sparse.linalg import svds
#U  and VT(transposed matrix for product) are orthogonal, and represent different things.
#U represents how much users “like” each feature and
#VT represents how relevant each feature is to each product.
#we need to get the lower rank approximation, svd generates matrix and it is
calulated using the common optimal minimium score
U, sigma, Vt = svds(Rating_pivot_table, k = NUMBER_OF_FACTORS_MF)
U.shape
Vt.shape
#converting diagonal matrix form
sigma = np.diag(sigma)
sigma.shape
```

Out[39]:

```
(15, 15)
```

In [40]:

```
#Here we are reconstruct the original matrix by multiplying its factors the resulting matrix will not sparse any
more.
It was generated predictions for items the user have not yet interaction, which we will exploit for recommendatio
ns.
user_product_ratings = np.dot(np.dot(U, sigma), Vt)
user_product_ratings
```

Out[40]:

```
array([[-2.35884577e-33, -6.80819130e-34,  4.42751109e-34, ...,
        -1.96172865e-33,  3.09894687e-34, -1.94377822e-33],
       [-2.94855721e-33, -8.51023913e-34,  5.53438886e-34, ...,
        -2.45216081e-33,  3.87368358e-34, -2.42972278e-33],
       [-2.94855721e-33, -8.51023913e-34,  5.53438886e-34, ...,
        -2.45216081e-33,  3.87368358e-34, -2.42972278e-33],
       ...,
       [ 2.45089090e-32, -3.82862621e-34,  1.69541877e-33, ...,
        -2.45913981e-32,  3.95594804e-33, -2.35547651e-32],
       [ 2.45089090e-32, -3.82862621e-34,  1.69541877e-33, ...,
        -2.45913981e-32,  3.95594804e-33, -2.35547651e-32],
       [ 2.45089090e-32, -3.82862621e-34,  1.69541877e-33, ...,
        -2.45913981e-32,  3.95594804e-33, -2.35547651e-32]])
```

In [41]:

```
#Converting the reconstructed matrix back to a Pandas dataframe
ratings_sparse_matrix = pd.DataFrame(user_product_ratings, index=Ratings_data_ids ,columns = Rating_pivot_table.c
olumns)
#printing the matrix
ratings_sparse_matrix.head(10)
```

Out[41]:

| ProductId | B000084EK4 | B000084EK5 | B000084EK6 | B000084EK7 | B000084EK8 | B000084EK9 | B000084EKA | B000084EKB | B000084EKC | B0 |
|---|---|---|---|---|---|---|---|---|---|---|
| 14 | -2.358846e-33 | -6.808191e-34 | 4.427511e-34 | 1.096531e-33 | -3.332965e-33 | -5.243882e-34 | 2.530746e-33 | -2.393914e-33 | -6.451799e-34 | 5 |
| 15 | -2.948557e-33 | -8.510239e-34 | 5.534389e-34 | 1.370664e-33 | -4.166206e-33 | -6.554853e-34 | 3.163432e-33 | -2.992393e-33 | -8.064748e-34 | 6 |
| 19 | -2.948557e-33 | -8.510239e-34 | 5.534389e-34 | 1.370664e-33 | -4.166206e-33 | -6.554853e-34 | 3.163432e-33 | -2.992393e-33 | -8.064748e-34 | 6 |
| 45 | -3.799470e-33 | -4.187070e-34 | 2.619470e-34 | 1.473333e-33 | -5.241589e-33 | -1.030010e-33 | 3.369018e-33 | -3.185894e-33 | -4.917402e-34 | 9 |
| 47 | -3.799470e-33 | -4.187070e-34 | 2.619470e-34 | 1.473333e-33 | -5.241589e-33 | -1.030010e-33 | 3.369018e-33 | -3.185894e-33 | -4.917402e-34 | 9 |
| 74 | 4.472965e-34 | -7.710534e-35 | 6.094317e-35 | -9.092955e-35 | 6.503754e-34 | 1.783527e-34 | -1.842998e-34 | 1.664961e-34 | -5.533384e-35 | -1 |
| 110 | -8.619615e-36 | 9.365916e-34 | -5.882036e-34 | -2.594757e-34 | 8.468827e-35 | -9.507246e-35 | -6.390608e-34 | 6.211372e-34 | 8.258862e-34 | 6 |
| 142 | 6.840143e-34 | 1.191632e-34 | -7.612280e-35 | -2.833070e-34 | 9.435404e-34 | 1.802509e-34 | -6.506197e-34 | 6.151642e-34 | 1.314160e-34 | -1 |
| 144 | 8.550179e-34 | 1.489540e-34 | -9.515350e-35 | -3.541337e-34 | 1.179426e-33 | 2.253136e-34 | -8.132746e-34 | 7.689552e-34 | 1.642700e-34 | -2 |
| 145 | 5.130107e-34 | 8.937238e-35 | -5.709210e-35 | -2.124802e-34 | 7.076553e-34 | 1.351882e-34 | -4.879648e-34 | 4.613731e-34 | 9.856199e-35 | -1 |

10 rows × 1453 columns

In [42]:

```
#merging the user id in constructed matrix
ratings_sparse_matrix['UserId'] = userid_data
#inserted the user id as index
ratings_sparse_matrix.set_index('UserId', inplace=True)
#printing the ratings sparse matrix
ratings_sparse_matrix
```

Out[42]:

| ProductId | B000084EK4 | B000084EK5 | B000084EK6 | B000084EK7 | B000084EK8 | B000084EK9 | B000084EKA | B000084EKB | B000 |
|---|---|---|---|---|---|---|---|---|---|
| UserId | | | | | | | | | |
| A10ZBR6O8S8OCY | -2.358846e-33 | -6.808191e-34 | 4.427511e-34 | 1.096531e-33 | -3.332965e-33 | -5.243882e-34 | 2.530746e-33 | -2.393914e-33 | -6.4 |
| A110RS7QL8Q1UX | -2.948557e-33 | -8.510239e-34 | 5.534389e-34 | 1.370664e-33 | -4.166206e-33 | -6.554853e-34 | 3.163432e-33 | -2.992393e-33 | -8.0 |
| A11LNY2OLQSUSV | -2.948557e-33 | -8.510239e-34 | 5.534389e-34 | 1.370664e-33 | -4.166206e-33 | -6.554853e-34 | 3.163432e-33 | -2.992393e-33 | -8.0 |
| A13MKSASQ6YWL7 | -3.799470e-33 | -4.187070e-34 | 2.619470e-34 | 1.473333e-33 | -5.241589e-33 | -1.030010e-33 | 3.369018e-33 | -3.185894e-33 | -4.9 |
| A13QLFTU0I8PSZ | -3.799470e-33 | -4.187070e-34 | 2.619470e-34 | 1.473333e-33 | -5.241589e-33 | -1.030010e-33 | 3.369018e-33 | -3.185894e-33 | -4.9 |
| A16AXQ11SZA8SQ | 4.472965e-34 | -7.710534e-35 | 6.094317e-35 | -9.092955e-35 | 6.503754e-34 | 1.783527e-34 | -1.842998e-34 | 1.664961e-34 | -5.5 |
| A18NJ23KZ9LBOV | -8.619615e-36 | 9.365916e-34 | -5.882036e-34 | -2.594757e-34 | 8.468827e-35 | -9.507246e-35 | -6.390608e-34 | 6.211372e-34 | 8.2 |
| A1B05INWIDZ74O | 6.840143e-34 | 1.191632e-34 | -7.612280e-35 | -2.833070e-34 | 9.435404e-34 | 1.802509e-34 | -6.506197e-34 | 6.151642e-34 | 1.3 |
| A1B6O7SAIYG2N0 | 8.550179e-34 | 1.489540e-34 | -9.515350e-35 | -3.541337e-34 | 1.179426e-33 | 2.253136e-34 | -8.132746e-34 | 7.689552e-34 | 1.6 |
| A1B70ZEWQ6UH1A | 5.130107e-34 | 8.937238e-35 | -5.709210e-35 | -2.124802e-34 | 7.076553e-34 | 1.351882e-34 | -4.879648e-34 | 4.613731e-34 | 9.8 |
| A1C217T6VWIE3 | -5.639776e-35 | -8.647312e-34 | 5.418236e-34 | 2.614189e-34 | -1.753097e-34 | 6.619469e-35 | 6.369768e-34 | -6.180755e-34 | -7.6 |
| A1DC1O4VX6AHPP | -8.639684e-34 | 8.919959e-35 | -4.514843e-35 | 3.029278e-34 | -1.206894e-33 | -2.287506e-34 | 6.787709e-34 | -6.494361e-34 | 5.4 |
| | -6.911747e- | 7.135967e- | -3.611874e- | 2.423422e- | -9.655155e- | -1.830005e- | 5.430167e- | -5.195488e- | 4.3 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **A1DH3QDZX8Z7GO** | 34 | 35 | 35 | 34 | 34 | 34 | 34 | 34 | |
| **A1DIFL0333QPEB** | 1.376547e-33 | 3.964903e-34 | -2.426835e-34 | -1.006200e-33 | 1.873933e-33 | 1.195440e-34 | -2.301568e-33 | 2.218063e-33 | 3.4 |
| **A1ED3ABQD627TY** | 2.855310e-35 | 8.306026e-34 | -5.193880e-34 | -2.387603e-34 | 1.367856e-34 | -6.699022e-35 | -5.797656e-34 | 5.589093e-34 | 7.3 |
| **A1EPP56A17R62Q** | -3.359314e-35 | 1.752795e-33 | -1.100672e-33 | -4.787754e-34 | 1.304714e-34 | -1.758439e-34 | -1.160690e-33 | 1.122818e-33 | 1.5 |
| **A1GWJQLX1LSZF0** | 1.255430e-33 | 6.499159e-35 | -4.383654e-35 | -3.813697e-34 | 1.764580e-33 | 1.800168e-34 | -8.517146e-34 | 8.128945e-34 | 4.1 |
| **A1HOXKR7OKJ1X1** | -1.701377e-37 | -7.762573e-36 | 4.859443e-36 | 2.337732e-36 | -9.827892e-37 | 7.635448e-37 | 5.684999e-36 | -5.459903e-36 | -6.8 |
| **A1I9PCPMT3GSKG** | 2.996800e-35 | 5.562062e-34 | -3.540601e-34 | -1.605316e-34 | 1.052643e-34 | -4.661265e-35 | -3.964916e-34 | 3.865166e-34 | 4.9 |
| **A1IHK8GL1ESOND** | 8.693688e-37 | 1.640179e-34 | -1.043831e-34 | -4.370370e-35 | 1.805032e-35 | -1.852478e-35 | -1.076580e-34 | 1.044197e-34 | 1.4 |
| **A1KE23JJ3YG4DT** | 2.000449e-35 | -9.023257e-35 | 5.613398e-35 | 1.371080e-35 | 1.869349e-35 | 1.207162e-35 | 3.552296e-35 | -3.439559e-35 | -7.9 |
| **A1KPQQXV7G8CLK** | 1.000225e-34 | -4.511629e-34 | 2.806699e-34 | 6.855401e-35 | 9.346743e-35 | 6.035808e-35 | 1.776148e-34 | -1.719780e-34 | -3.9 |
| **A1LJR5IS0B6ADX** | 2.600989e-34 | 1.123768e-34 | -7.102215e-35 | -2.162623e-34 | 3.622086e-34 | 2.109159e-35 | -4.969127e-34 | 4.803590e-34 | 9.8 |
| **A1LP1PGQLL9B5F** | 2.600989e-34 | 1.123768e-34 | -7.102215e-35 | -2.162623e-34 | 3.622086e-34 | 2.109159e-35 | -4.969127e-34 | 4.803590e-34 | 9.8 |
| **A1MC6BFHWY6WC3** | 3.445116e-34 | -1.322438e-33 | 8.268448e-34 | 1.731423e-34 | 3.696696e-34 | 2.113191e-34 | 4.672845e-34 | -4.572861e-34 | -1.1 |
| **A1O229NVVDJUX2** | 6.890231e-35 | -2.644876e-34 | 1.653690e-34 | 3.462846e-35 | 7.393392e-35 | 4.226383e-35 | 9.345690e-35 | -9.145723e-35 | -2.3 |
| **A1ODKP5Y7DPMM1** | -3.120558e-35 | -1.368267e-33 | 8.681136e-34 | 3.559962e-34 | -2.087123e-34 | 1.400869e-34 | 8.755237e-34 | -8.453034e-34 | -1.2 |
| **A1S52XH2CS4Y26** | -2.289482e-33 | -6.867974e-34 | 4.182172e-34 | 1.725495e-33 | -3.107528e-33 | -1.762796e-34 | 3.954852e-33 | -3.823583e-33 | -5.9 |
| **A1SYLII0808HD6** | -1.276563e-33 | 1.451335e-34 | -8.222621e-35 | 4.613603e-34 | -1.786685e-33 | -3.292096e-34 | 1.029073e-33 | -9.794122e-34 | 1.0 |
| **A1T1YSCDW0PD25** | -1.021250e-33 | 1.161068e-34 | -6.578097e-35 | 3.690882e-34 | -1.429348e-33 | -2.633677e-34 | 8.232588e-34 | -7.835298e-34 | 8.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 1.960713e-32 | -3.062901e-34 | 1.356335e-33 | 8.779502e-33 | 2.927535e-32 | -9.772935e-34 | 2.389778e-32 | -2.492226e-32 | -2.2 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 1.960713e-32 | -3.062901e-34 | 1.356335e-33 | 8.779502e-33 | 2.927535e-32 | -9.772935e-34 | 2.389778e-32 | -2.492226e-32 | -2.2 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 1.960713e-32 | -3.062901e-34 | 1.356335e-33 | 8.779502e-33 | 2.927535e-32 | -9.772935e-34 | 2.389778e-32 | -2.492226e-32 | -2.2 |
| **NaN** | 1.960713e-32 | -3.062901e-34 | 1.356335e-33 | 8.779502e-33 | 2.927535e-32 | -9.772935e-34 | 2.389778e-32 | -2.492226e-32 | -2.2 |
| **NaN** | 1.470535e-32 | -2.297176e-34 | 1.017251e-33 | 6.584626e-33 | 2.195652e-32 | -7.329701e-34 | 1.792333e-32 | -1.869169e-32 | -1.6 |

|  |  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
| **NaN** | 1.470535e-32 | -2.297176e-34 | 1.017251e-33 | 6.584626e-33 | 2.195652e-32 | -7.329701e-34 | 1.792333e-32 | -1.869169e-32 | -1.6 |
| **NaN** | 1.470535e-32 | -2.297176e-34 | 1.017251e-33 | 6.584626e-33 | 2.195652e-32 | -7.329701e-34 | 1.792333e-32 | -1.869169e-32 | -1.6 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 1.470535e-32 | -2.297176e-34 | 1.017251e-33 | 6.584626e-33 | 2.195652e-32 | -7.329701e-34 | 1.792333e-32 | -1.869169e-32 | -1.6 |
| **NaN** | 9.803564e-33 | -1.531450e-34 | 6.781675e-34 | 4.389751e-33 | 1.463768e-32 | -4.886468e-34 | 1.194889e-32 | -1.246113e-32 | -1.1 |
| **NaN** | 1.470535e-32 | -2.297176e-34 | 1.017251e-33 | 6.584626e-33 | 2.195652e-32 | -7.329701e-34 | 1.792333e-32 | -1.869169e-32 | -1.6 |
| **NaN** | 9.803564e-33 | -1.531450e-34 | 6.781675e-34 | 4.389751e-33 | 1.463768e-32 | -4.886468e-34 | 1.194889e-32 | -1.246113e-32 | -1.1 |
| **NaN** | 4.901782e-33 | -7.657252e-35 | 3.390838e-34 | 2.194875e-33 | 7.318838e-33 | -2.443234e-34 | 5.974444e-33 | -6.230564e-33 | -5.6 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |
| **NaN** | 2.450891e-32 | -3.828626e-34 | 1.695419e-33 | 1.097438e-32 | 3.659419e-32 | -1.221617e-33 | 2.987222e-32 | -3.115282e-32 | -2.8 |

In [43]:

```python
#function to recommend product
def product_recommendation(recommend, products, ratings, recommendations_number=5):

    # Get and sort the user's predictions
    user_predictions = recommend.loc['A10ZBR6O8S8OCY'].sort_values(ascending=False)

    # Get the user's data and merge in the movie information.
    user_id = ratings[ratings.UserId == 'A10ZBR6O8S8OCY']
    merge_data = (user_id.merge(products, how = 'left', left_on = 'ProductId', right_on = 'ProductId').
                     sort_values(['Score'], ascending=False)
                 )

    print ('User {0} has already rated {1} items.'.format('A10ZBR6O8S8OCY', merge_data.shape[0]))
    print ('Recommending the highest {0} predicted ratings for items not already rated.'.format(recommendations_n
umber))

    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    product_recommendations = (products[~products['ProductId'].isin(merge_data['ProductId'])].
         merge(pd.DataFrame(user_predictions).reset_index(), how = 'left',
               left_on = 'ProductId',
               right_on = 'ProductId').
         rename(columns = {'A10ZBR6O8S8OCY': 'Predictions'}).
         sort_values('Predictions', ascending = False).
                      iloc[:recommendations_number, :-1]
                     )
    print(product_recommendations)
    return user_predictions
```

**Top Recommendation**

In [164]:

```python
Top_Recommendation
```

Out[164]:

|  | ProductId | Product |
|---|---|---|
| **8062** | B001BDDTB2 | Laptop |
| **17219** | B004YV80OE | Health Care |
| **7135** | B0013NUGDE | Body Wash |
| **18415** | B006J4MAIQ | Kindle E-readers |
| **12572** | B002LANN56 | B002LANN56 |

## CONCLUSION

Based on the above models

Popularity based recommendation system

Advantage New user can identify the trending product Disadvantage Can not provide Personalized Recommendations as they dont use user data into account

Item based collabrative filtering

Advantage

Disadvantage

Matrix Factorisation

Advantage instead of having a high dimensional matrix containing abundant number of missing values we will be dealing with a much smaller matrix in lower-dimensional space. There are several advantages with this paradigm. It handles the sparsity of the original matrix better than memory based ones. Also comparing similarity on the resulting matrix is much more scalable especially in dealing with large sparse datasets.

The model doesn't need any data about other users, since the recommendations are specific to this user Disadvantage