Book Rental Recommendation.

Course-end Project 1

DESCRIPTION

Book Rent is the largest online and offline book rental chain in India. They provide books of various genres, such as thrillers, mysteries, romances, and science fiction. The company charges a fixed rental fee for a book per month. Lately, the company has been losing its user base. The main reason for this is that users are not able to choose the right books for themselves. The company wants to solve this problem and increase its revenue and profit.

Project Objective:

You, as an ML expert, should focus on improving the user experience by personalizing it to the user's needs. You have to model a recommendation engine so that users get recommendations for books based on the behavior of similar users. This will ensure that users are renting the books based on their tastes and traits.

Note: You have to perform user-based collaborative filtering and item-based collaborative filtering.

Dataset description:

BX-Users: It contains the information of users.

- user_id - These have been anonymized and mapped to integers
- Location - Demographic data is provided
- Age - Demographic data is provided

If available, otherwise, these fields contain NULL-values.

BX-Books:

- isbn - Books are identified by their respective ISBNs. Invalid ISBNs have already been removed from the dataset.
- book_title
- book_author
- year_of_publication
- publisher

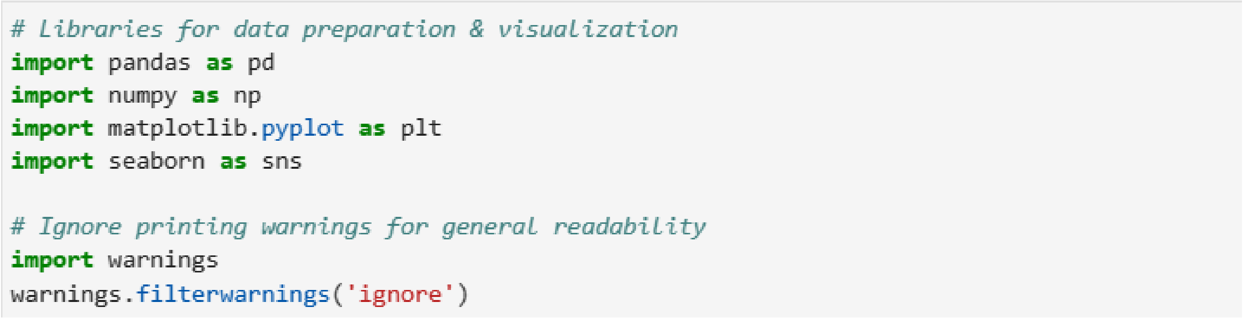
BX-Book-Ratings: Contains the book rating information.

- user_id
- isbn
- rating - Ratings (`Book-Rating`) are either explicit, expressed on a scale from 1–10 (higher values denoting higher appreciation), or implicit, expressed by 0.

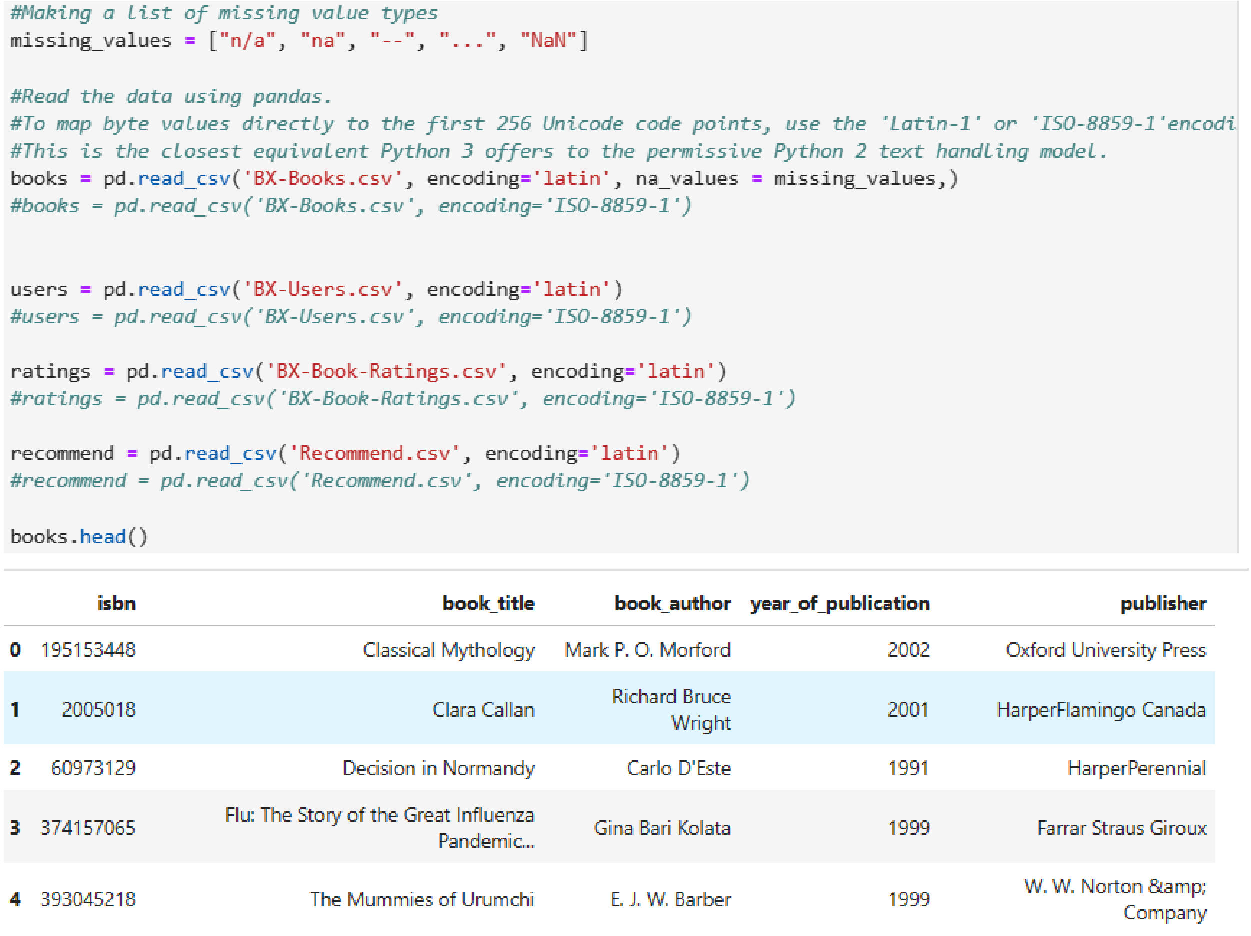
Note: Download the “BX-Book-Ratings.csv”, “BX-Books.csv”, “BX-Users.csv”, and “Recommend.csv” using the link given in the Book Rental Recommendation project problem statement.

Following operations should be performed:

- Read the books dataset and explore it

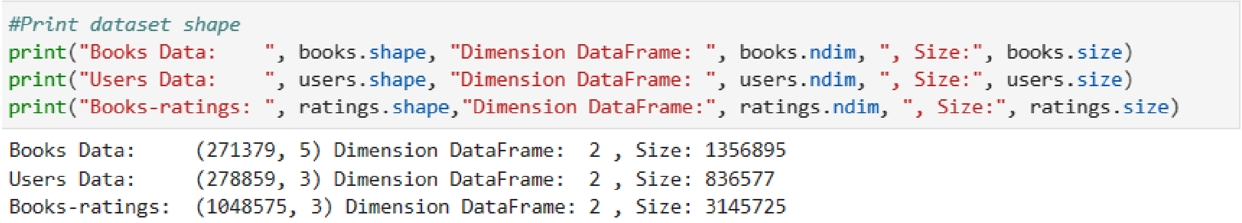


```python
# Libraries for data preparation & visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Ignore printing warnings for general readability
import warnings
warnings.filterwarnings('ignore')
```



```python
#Making a list of missing value types
missing_values = ["n/a", "na", "--", "...", "NaN"]

#Read the data using pandas.
#To map byte values directly to the first 256 Unicode code points, use the 'Latin-1' or 'ISO-8859-1'encodi
#This is the closest equivalent Python 3 offers to the permissive Python 2 text handling model.
books = pd.read_csv('BX-Books.csv', encoding='latin', na_values = missing_values,)
#books = pd.read_csv('BX-Books.csv', encoding='ISO-8859-1')


users = pd.read_csv('BX-Users.csv', encoding='latin')
#users = pd.read_csv('BX-Users.csv', encoding='ISO-8859-1')

ratings = pd.read_csv('BX-Book-Ratings.csv', encoding='latin')
#ratings = pd.read_csv('BX-Book-Ratings.csv', encoding='ISO-8859-1')

recommend = pd.read_csv('Recommend.csv', encoding='latin')
#recommend = pd.read_csv('Recommend.csv', encoding='ISO-8859-1')

books.head()
```

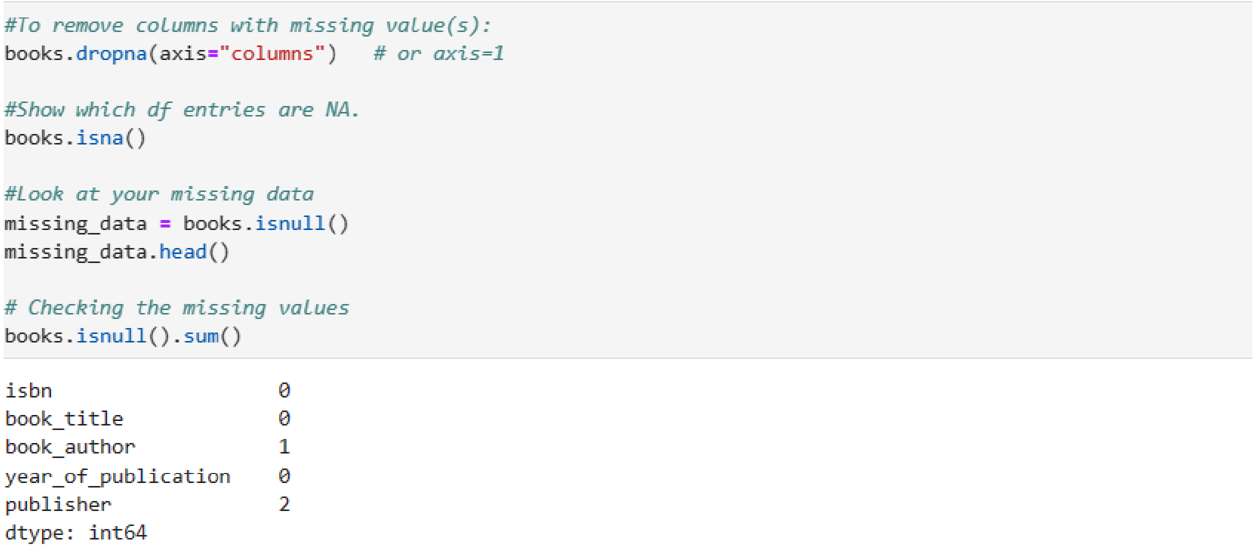
	isbn	book_title	book_author	year_of_publication	publisher
0	195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press
1	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
2	60973129	Decision in Normandy	Carlo D'Este	1991	HarperPerennial
3	374157065	Flu: The Story of the Great Influenza Pandemic...	Gina Bari Kolata	1999	Farrar Straus Giroux
4	393045218	The Mummies of Urumchi	E. J. W. Barber	1999	W. W. Norton &amp; Company



```python
#Print dataset shape
print("Books Data:    ", books.shape, "Dimension DataFrame: ", books.ndim, ", Size:", books.size)
print("Users Data:    ", users.shape, "Dimension DataFrame: ", users.ndim, ", Size:", users.size)
print("Books-ratings: ", ratings.shape,"Dimension DataFrame:", ratings.ndim, ", Size:", ratings.size)
```

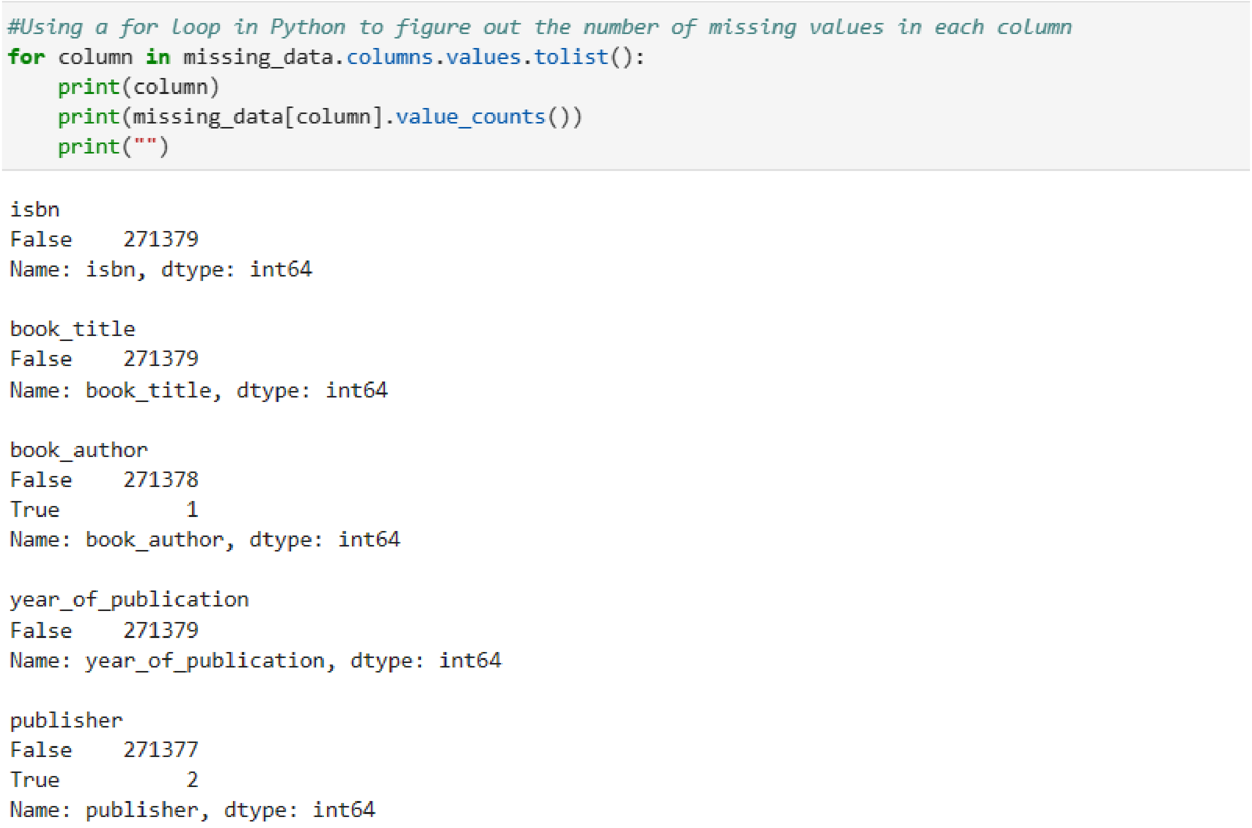
```
Books Data:     (271379, 5) Dimension DataFrame:  2 , Size: 1356895
Users Data:     (278859, 3) Dimension DataFrame:  2 , Size: 836577
Books-ratings:  (1048575, 3) Dimension DataFrame: 2 , Size: 3145725
```

- Clean up NaN values



```
#To remove columns with missing value(s):
books.dropna(axis="columns")   # or axis=1

#Show which df entries are NA.
books.isna()

#Look at your missing data
missing_data = books.isnull()
missing_data.head()

# Checking the missing values
books.isnull().sum()
```

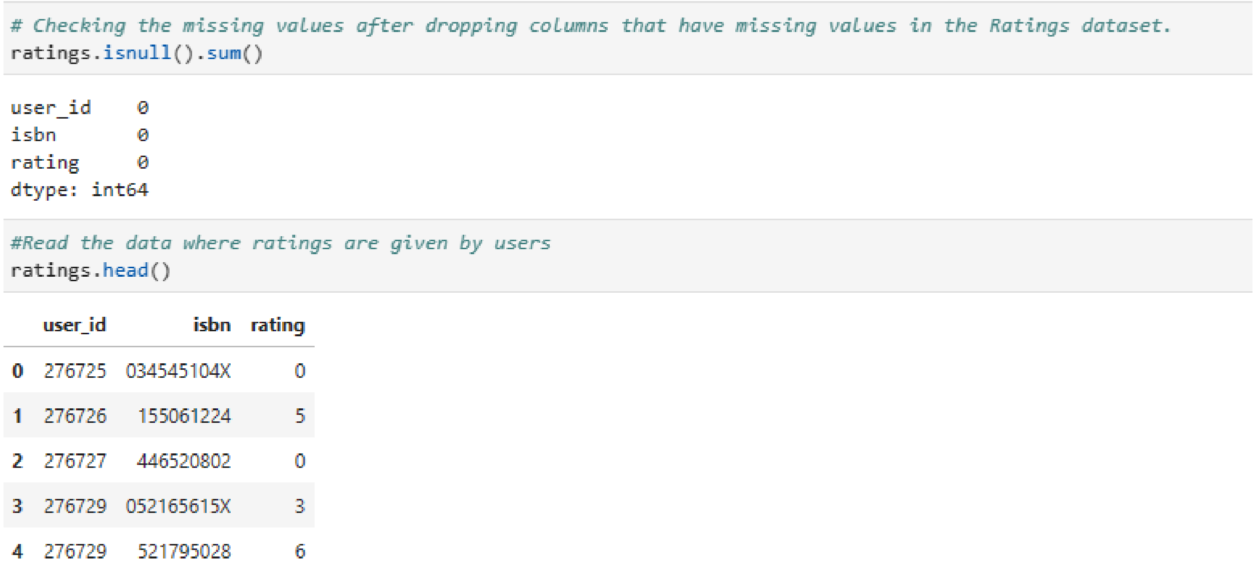
```
isbn                   0
book_title             0
book_author            1
year_of_publication    0
publisher              2
dtype: int64
```



```
#Using a for loop in Python to figure out the number of missing values in each column
for column in missing_data.columns.values.tolist():
    print(column)
    print(missing_data[column].value_counts())
    print("")
```

```
isbn
False    271379
Name: isbn, dtype: int64

book_title
False    271379
Name: book_title, dtype: int64

book_author
False    271378
True          1
Name: book_author, dtype: int64

year_of_publication
False    271379
Name: year_of_publication, dtype: int64

publisher
False    271377
True          2
Name: publisher, dtype: int64
```

- Read the data where ratings are given by users

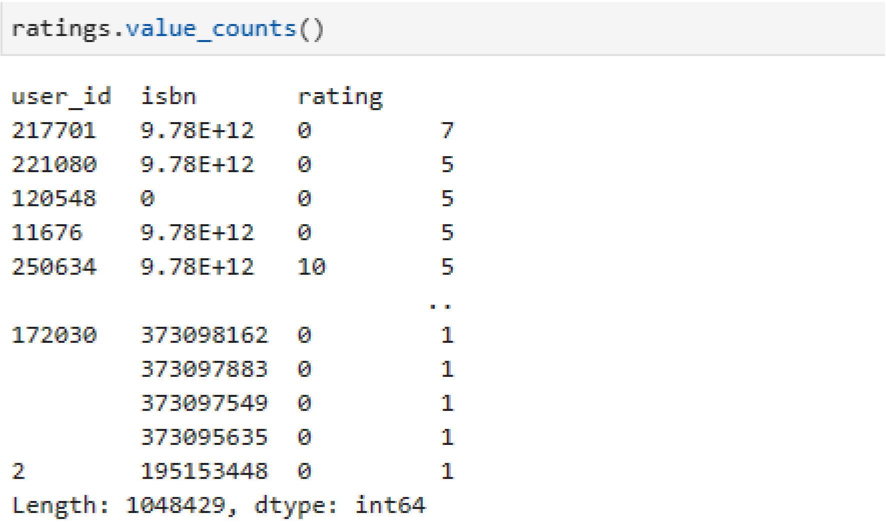


```
# Checking the missing values after dropping columns that have missing values in the Ratings dataset.
ratings.isnull().sum()
```

```
user_id    0
isbn       0
rating     0
dtype: int64
```

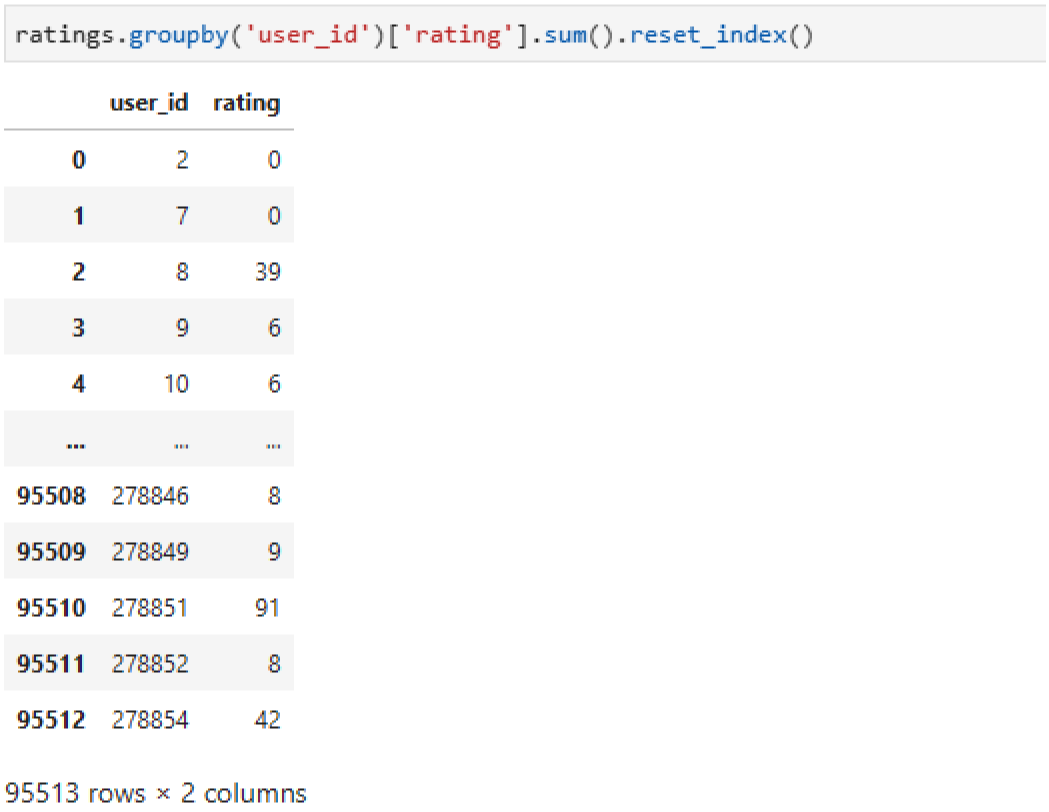
```
#Read the data where ratings are given by users
ratings.head()
```

	user_id	isbn	rating
0	276725	034545104X	0
1	276726	155061224	5
2	276727	446520802	0
3	276729	052165615X	3
4	276729	521795028	6



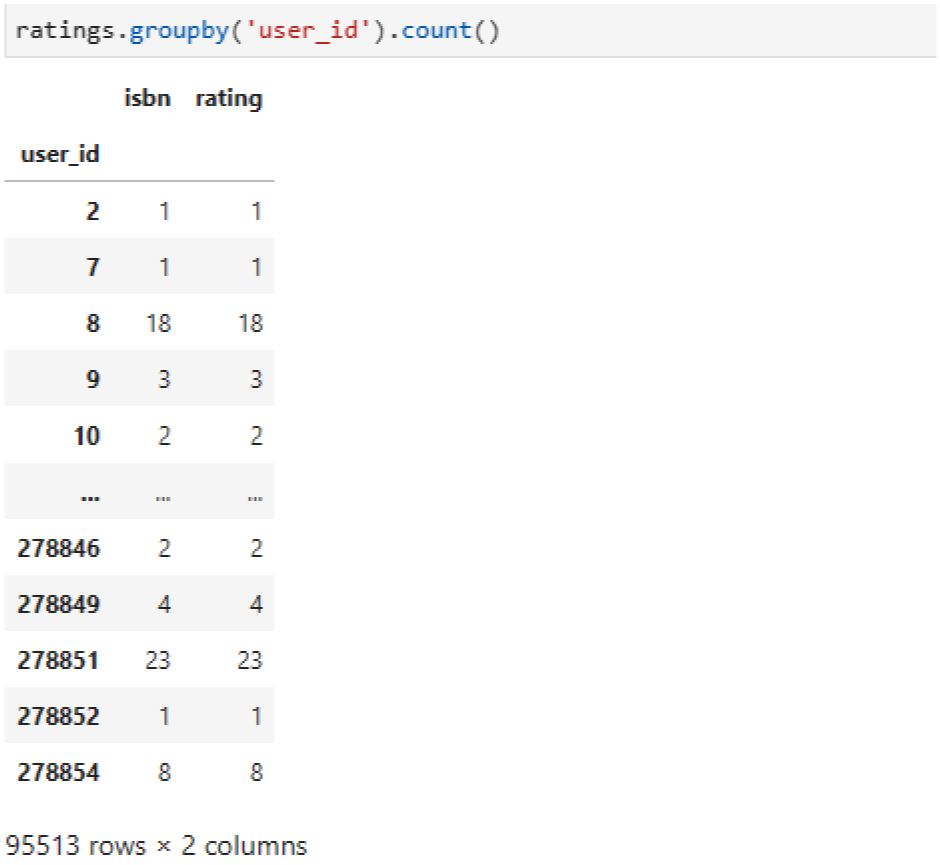
```
ratings.value_counts()
```

```
user_id  isbn       rating
217701   9.78E+12   0         7
221080   9.78E+12   0         5
120548   0          0         5
11676    9.78E+12   0         5
250634   9.78E+12   10        5
                             ..
172030   373098162  0         1
         373097883  0         1
         373097549  0         1
         373095635  0         1
2        195153448  0         1
Length: 1048429, dtype: int64
```



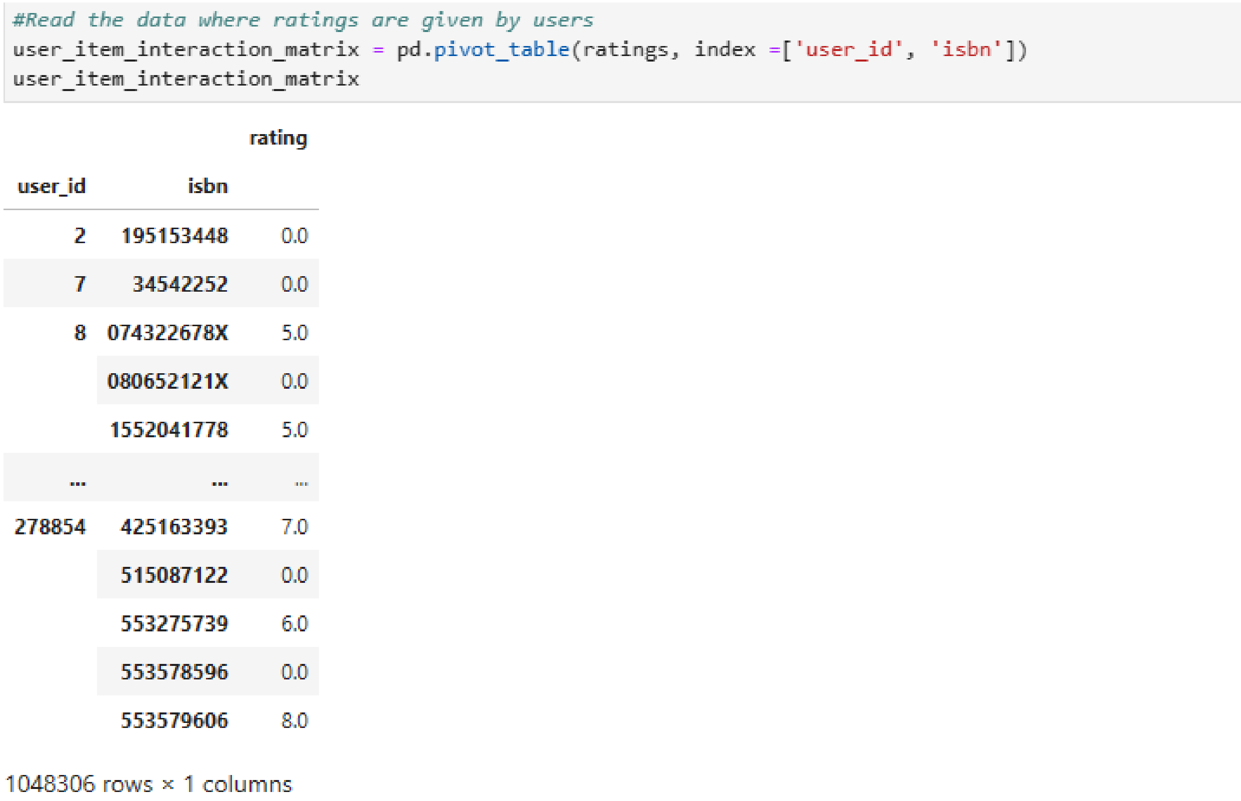
```
ratings.groupby('user_id')['rating'].sum().reset_index()
```

```
       user_id  rating
    0        2       0
    1        7       0
    2        8      39
    3        9       6
    4       10       6
  ...      ...     ...
95508   278846       8
95509   278849       9
95510   278851      91
95511   278852       8
95512   278854      42

95513 rows × 2 columns
```



```
ratings.groupby('user_id').count()
```

```
         isbn  rating
user_id
      2     1       1
      7     1       1
      8    18      18
      9     3       3
     10     2       2
    ...   ...     ...
 278846     2       2
 278849     4       4
 278851    23      23
 278852     1       1
 278854     8       8

95513 rows × 2 columns
```



```python
#Read the data where ratings are given by users
user_item_interaction_matrix = pd.pivot_table(ratings, index =['user_id', 'isbn'])
user_item_interaction_matrix
```

```
                     rating
user_id        isbn
      2   195153448     0.0
      7    34542252     0.0
      8  074322678X     5.0
         080652121X     0.0
         1552041778     5.0
    ...         ...     ...
 278854   425163393     7.0
          515087122     0.0
          553275739     6.0
          553578596     0.0
          553579606     8.0

1048306 rows × 1 columns
```

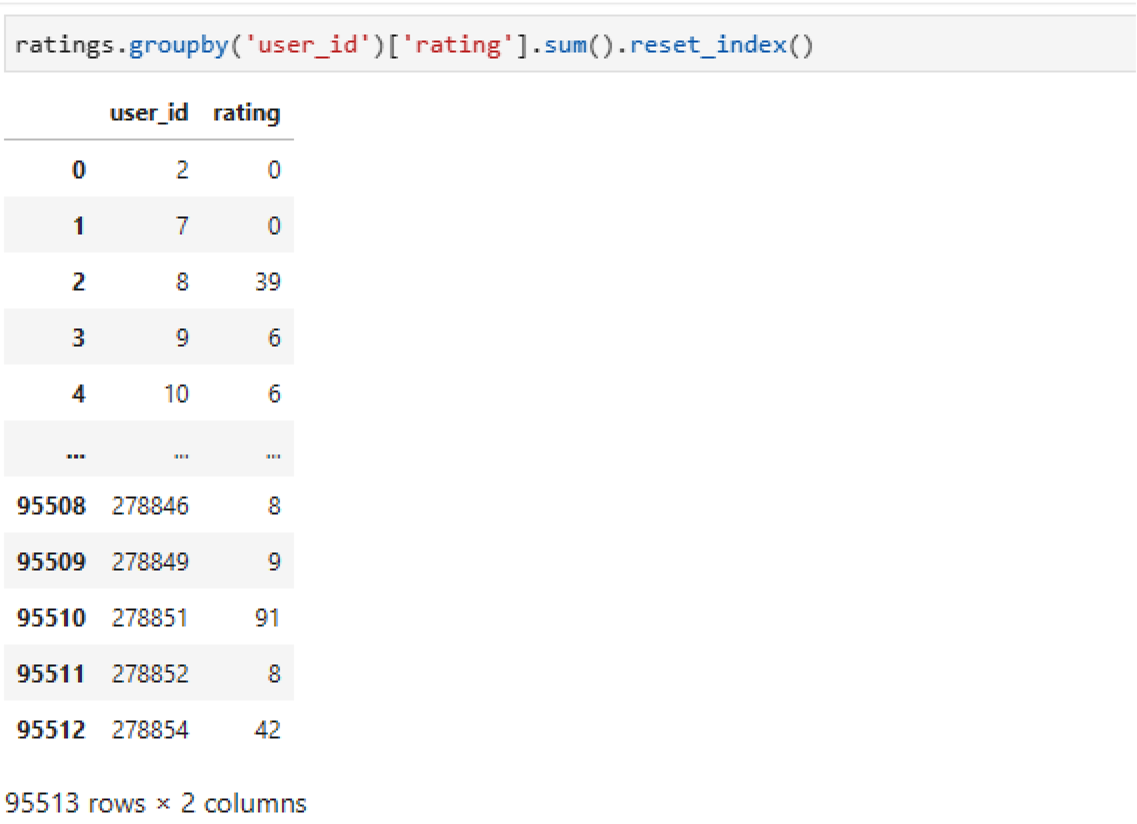
```python
# Check for duplicate values
print(f'Duplicate entries: {ratings.duplicated().sum()}')
```

```
Duplicate entries: 146
```

```python
#There were 146 duplicate records upon merging Ratings & Books which will be dropped

ratings.drop_duplicates(inplace=True)
ratings
```

	user_id	isbn	rating
0	276725	034545104X	0
1	276726	155061224	5
2	276727	446520802	0
3	276729	052165615X	3
4	276729	521795028	6
...	...	...	...
1048570	250764	451410777	0
1048571	250764	452264464	8
1048572	250764	048623715X	0
1048573	250764	486256588	0
1048574	250764	515069434	0

1048429 rows × 3 columns

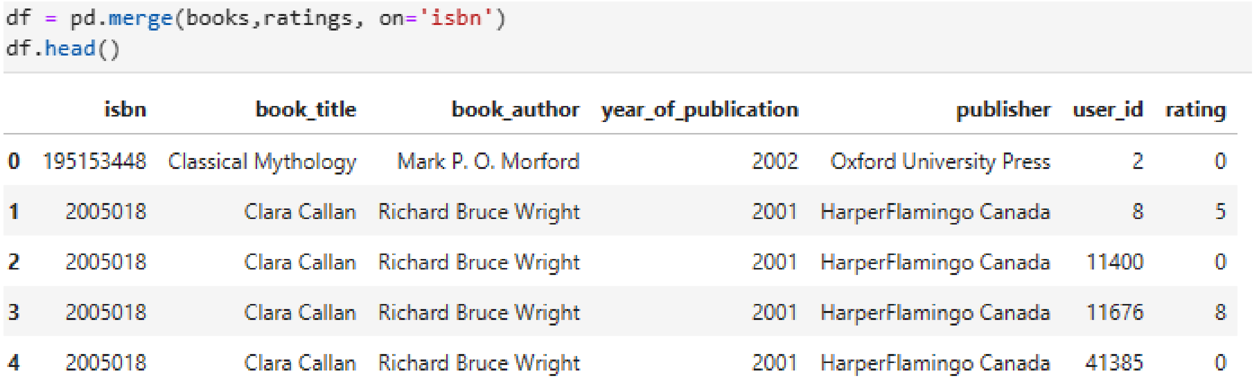


```python
ratings.groupby('user_id')['rating'].sum().reset_index()
```

	user_id	rating
0	2	0
1	7	0
2	8	39
3	9	6
4	10	6
...	...	...
95508	278846	8
95509	278849	9
95510	278851	91
95511	278852	8
95512	278854	42

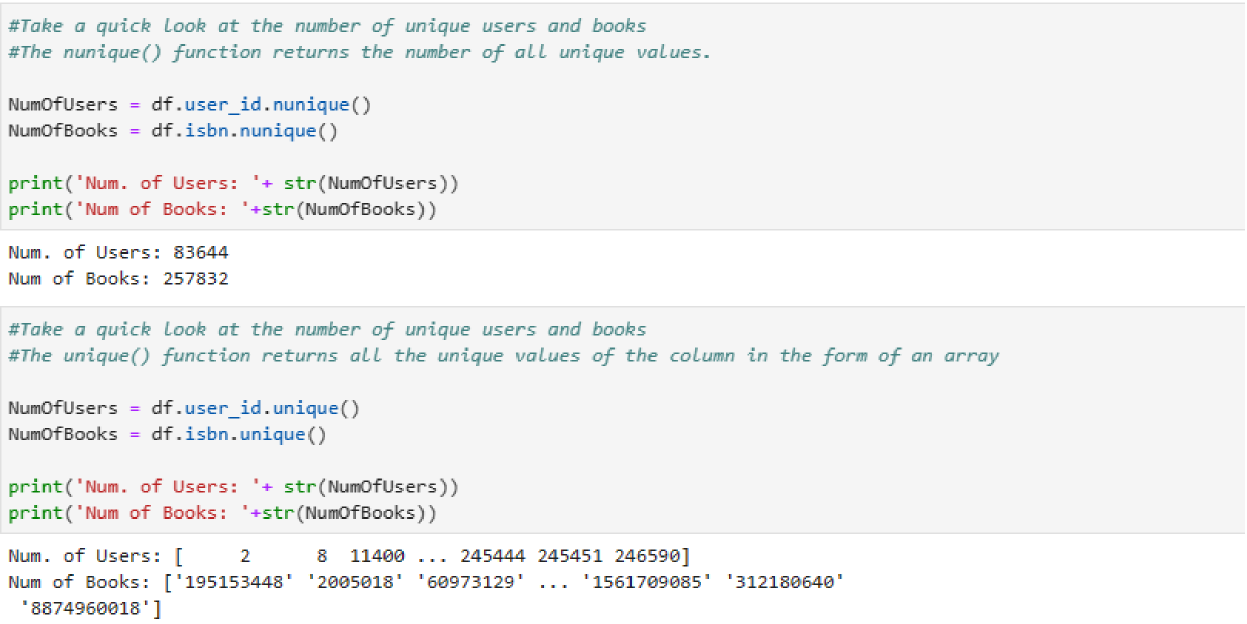
95513 rows × 2 columns

- Take a quick look at the number of unique users and books



```python
df = pd.merge(books,ratings, on='isbn')
df.head()
```

	isbn	book_title	book_author	year_of_publication	publisher	user_id	rating
0	195153448	Classical Mythology	Mark P. O. Morford	2002	Oxford University Press	2	0
1	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	8	5
2	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	11400	0
3	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	11676	8
4	2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada	41385	0



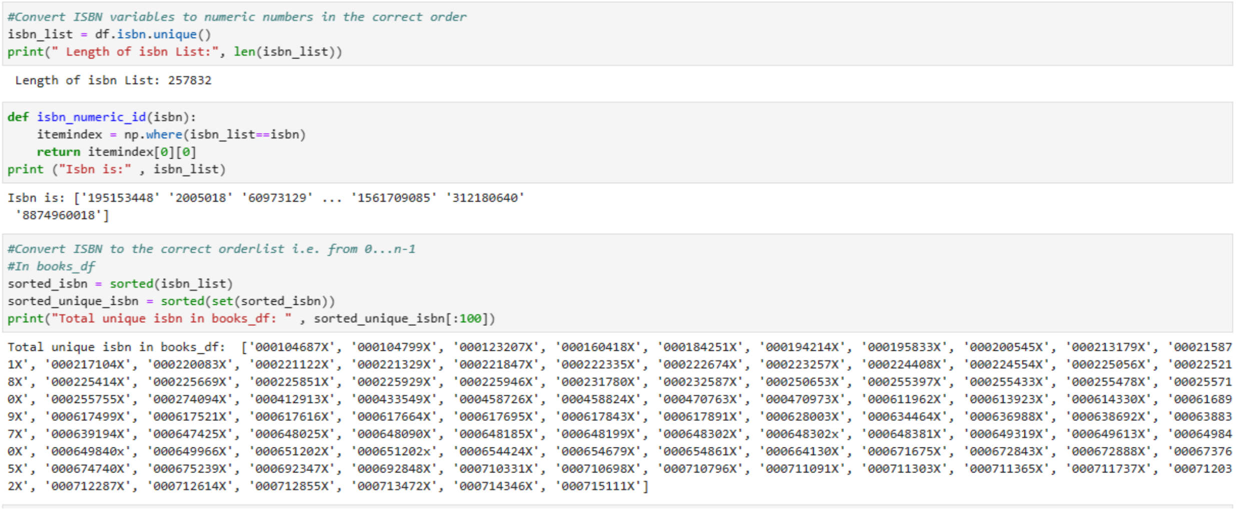
```
#Take a quick look at the number of unique users and books
#The nunique() function returns the number of all unique values.

NumOfUsers = df.user_id.nunique()
NumOfBooks = df.isbn.nunique()

print('Num. of Users: '+ str(NumOfUsers))
print('Num of Books: '+str(NumOfBooks))
```

```
Num. of Users: 83644
Num of Books: 257832
```

```
#Take a quick look at the number of unique users and books
#The unique() function returns all the unique values of the column in the form of an array

NumOfUsers = df.user_id.unique()
NumOfBooks = df.isbn.unique()

print('Num. of Users: '+ str(NumOfUsers))
print('Num of Books: '+str(NumOfBooks))
```

```
Num. of Users: [     2      8  11400 ... 245444 245451 246590]
Num of Books: ['195153448' '2005018' '60973129' ... '1561709085' '312180640'
 '8874960018']
```

- Convert ISBN variables to numeric numbers in the correct order



```
#Convert ISBN variables to numeric numbers in the correct order
isbn_list = df.isbn.unique()
print(" Length of isbn List:", len(isbn_list))
```

```
 Length of isbn List: 257832
```

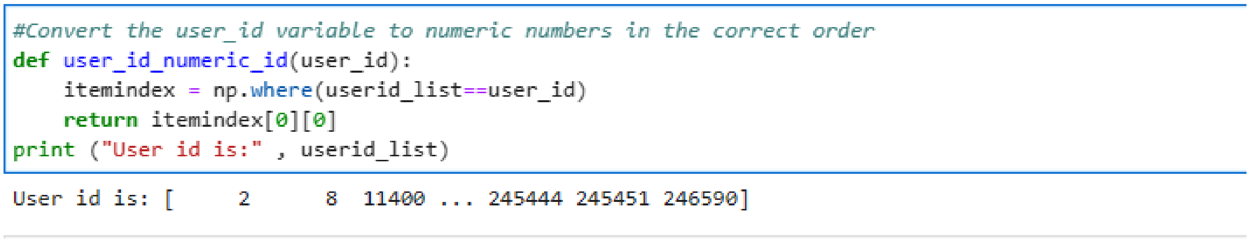
```
def isbn_numeric_id(isbn):
    itemindex = np.where(isbn_list==isbn)
    return itemindex[0][0]
print ("Isbn is:" , isbn_list)
```

```
Isbn is: ['195153448' '2005018' '60973129' ... '1561709085' '312180640'
 '8874960018']
```

```
#Convert ISBN to the correct orderlist i.e. from 0...n-1
#In books_df
sorted_isbn = sorted(isbn_list)
sorted_unique_isbn = sorted(set(sorted_isbn))
print("Total unique isbn in books_df: " , sorted_unique_isbn[:100])
```

```
Total unique isbn in books_df:  ['000104687X', '000104799X', '000123207X', '000160418X', '000184251X', '000194214X', '000195833X', '000200545X', '000213179X', '00021587
1X', '000217104X', '000220083X', '000221122X', '000221329X', '000221847X', '000222335X', '000222674X', '000223257X', '000224408X', '000224554X', '000225056X', '00022521
8X', '000225414X', '000225669X', '000225851X', '000225929X', '000225946X', '000231780X', '000232587X', '000250653X', '000255397X', '000255433X', '000255478X', '00025571
0X', '000255755X', '000274094X', '000412913X', '000433549X', '000458726X', '000458824X', '000470763X', '000470973X', '000611962X', '000613923X', '000614330X', '00061689
9X', '000617499X', '000617521X', '000617616X', '000617664X', '000617695X', '000617843X', '000617891X', '000628003X', '000634464X', '000636988X', '000638692X', '00063883
7X', '000639194X', '000647425X', '000648025X', '000648090X', '000648185X', '000648199X', '000648302X', '000648302x', '000648381X', '000649319X', '000649613X', '00064984
0X', '000649840x', '000649966X', '000651202X', '000651202x', '000654424X', '000654679X', '000654861X', '000664130X', '000671675X', '000672843X', '000672888X', '00067376
5X', '000674740X', '000675239X', '000692347X', '000692848X', '000710331X', '000710698X', '000710796X', '000711091X', '000711303X', '000711365X', '000711737X', '00071203
2X', '000712287X', '000712614X', '000712855X', '000713472X', '000714346X', '000715111X']
```

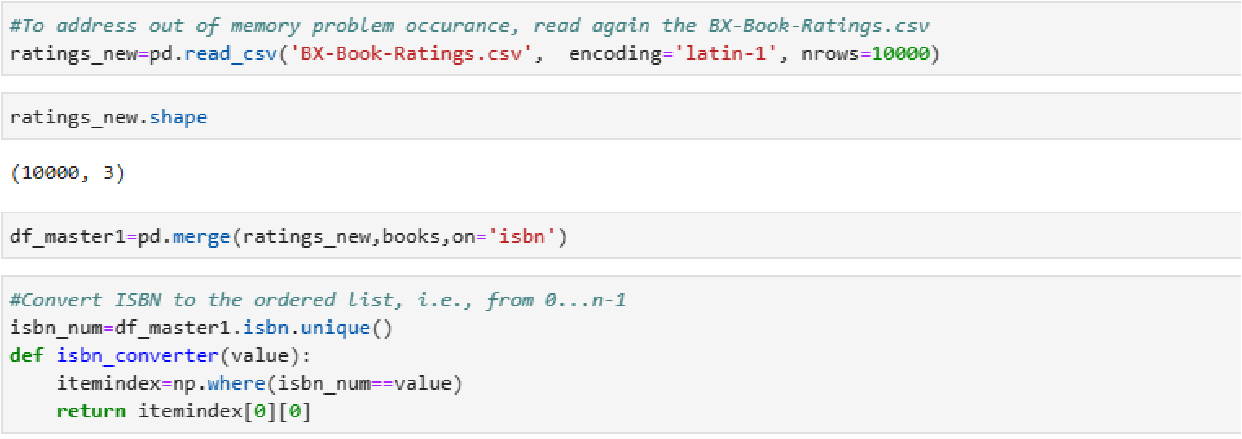
- Convert the user_id variable to numeric numbers in the correct order



```
#Convert the user_id variable to numeric numbers in the correct order
def user_id_numeric_id(user_id):
    itemindex = np.where(userid_list==user_id)
    return itemindex[0][0]
print ("User id is:" , userid_list)
```

```
User id is: [     2      8  11400 ... 245444 245451 246590]
```

- Convert both user_id and ISBN to the ordered list, i.e., from 0...n-1



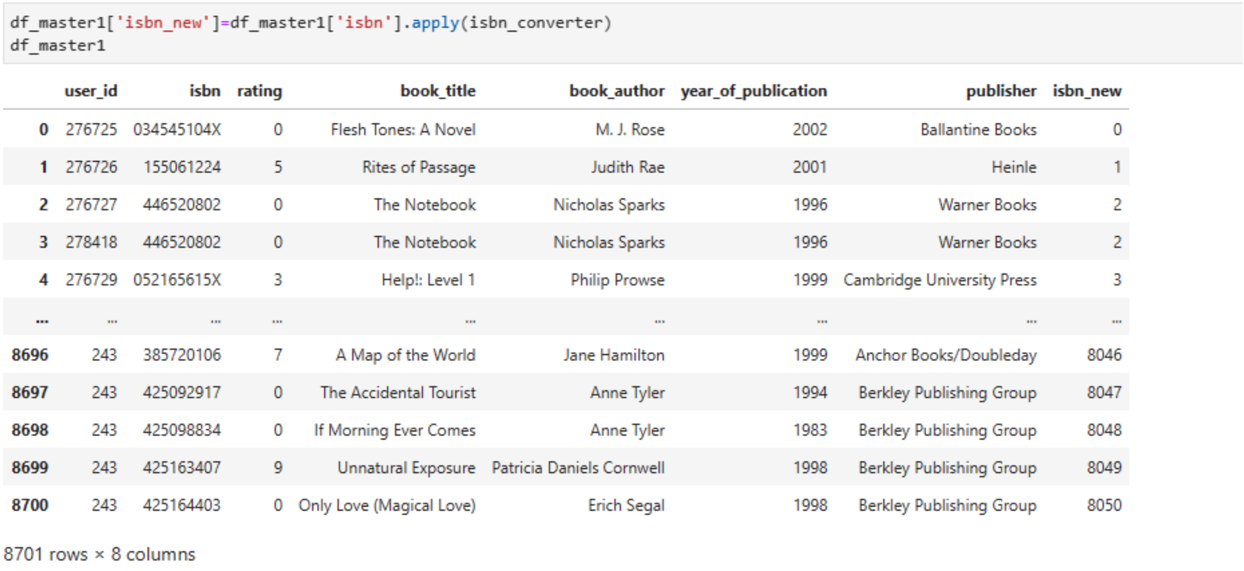
```
#To address out of memory problem occurance, read again the BX-Book-Ratings.csv
ratings_new=pd.read_csv('BX-Book-Ratings.csv',  encoding='latin-1', nrows=10000)
```

```
ratings_new.shape
```

```
(10000, 3)
```

```
df_master1=pd.merge(ratings_new,books,on='isbn')
```

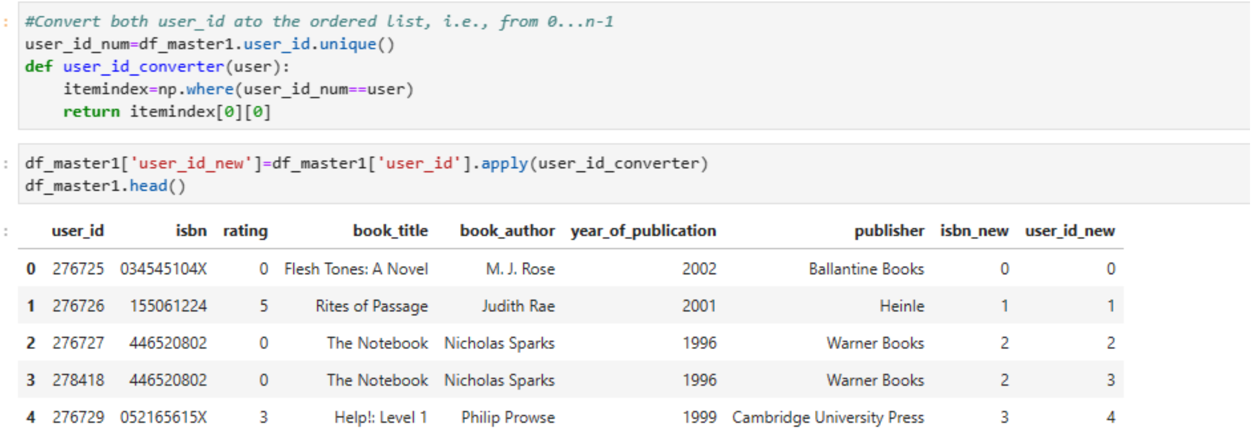
```
#Convert ISBN to the ordered list, i.e., from 0...n-1
isbn_num=df_master1.isbn.unique()
def isbn_converter(value):
    itemindex=np.where(isbn_num==value)
    return itemindex[0][0]
```



```
df_master1['isbn_new']=df_master1['isbn'].apply(isbn_converter)
df_master1
```

	user_id	isbn	rating	book_title	book_author	year_of_publication	publisher	isbn_new
0	276725	034545104X	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	0
1	276726	155061224	5	Rites of Passage	Judith Rae	2001	Heinle	1
2	276727	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books	2
3	278418	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books	2
4	276729	052165615X	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	3
...	...	...	...	...	...	...	...	...
8696	243	385720106	7	A Map of the World	Jane Hamilton	1999	Anchor Books/Doubleday	8046
8697	243	425092917	0	The Accidental Tourist	Anne Tyler	1994	Berkley Publishing Group	8047
8698	243	425098834	0	If Morning Ever Comes	Anne Tyler	1983	Berkley Publishing Group	8048
8699	243	425163407	9	Unnatural Exposure	Patricia Daniels Cornwell	1998	Berkley Publishing Group	8049
8700	243	425164403	0	Only Love (Magical Love)	Erich Segal	1998	Berkley Publishing Group	8050

8701 rows × 8 columns

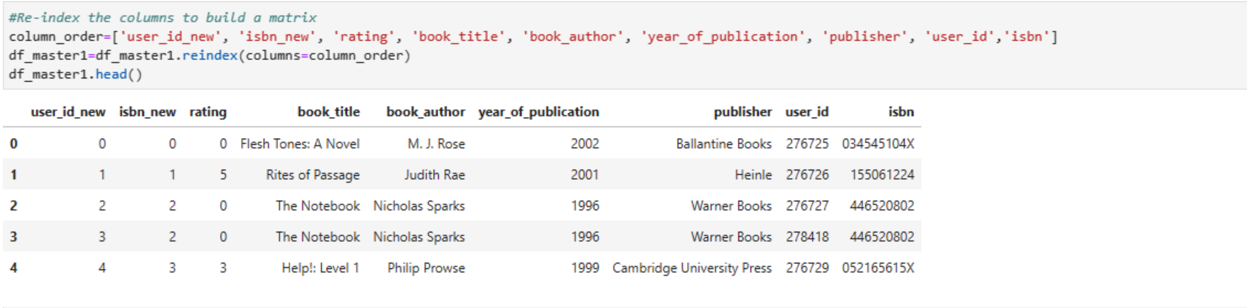


```
#Convert both user_id ato the ordered list, i.e., from 0...n-1
user_id_num=df_master1.user_id.unique()
def user_id_converter(user):
    itemindex=np.where(user_id_num==user)
    return itemindex[0][0]
```

```
df_master1['user_id_new']=df_master1['user_id'].apply(user_id_converter)
df_master1.head()
```

	user_id	isbn	rating	book_title	book_author	year_of_publication	publisher	isbn_new	user_id_new
0	276725	034545104X	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	0	0
1	276726	155061224	5	Rites of Passage	Judith Rae	2001	Heinle	1	1
2	276727	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books	2	2
3	278418	446520802	0	The Notebook	Nicholas Sparks	1996	Warner Books	2	3
4	276729	052165615X	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	3	4

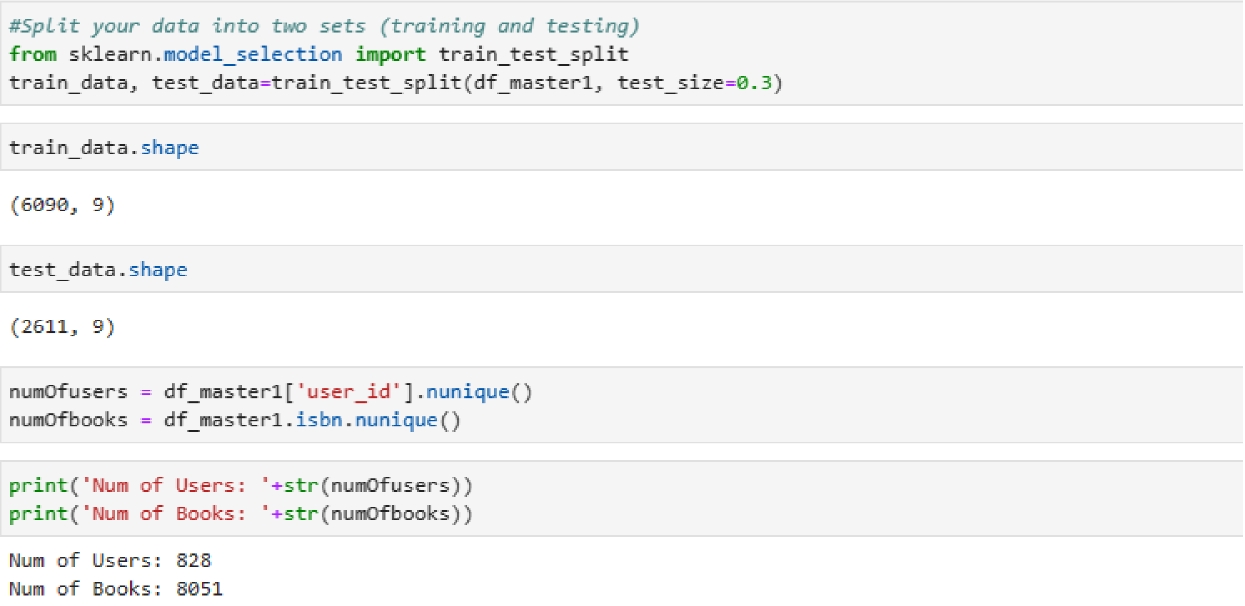
- Re-index the columns to build a matrix



```
#Re-index the columns to build a matrix
column_order=['user_id_new', 'isbn_new', 'rating', 'book_title', 'book_author', 'year_of_publication', 'publisher', 'user_id','isbn']
df_master1=df_master1.reindex(columns=column_order)
df_master1.head()
```

	user_id_new	isbn_new	rating	book_title	book_author	year_of_publication	publisher	user_id	isbn
0	0	0	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	276725	034545104X
1	1	1	5	Rites of Passage	Judith Rae	2001	Heinle	276726	155061224
2	2	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	276727	446520802
3	3	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	278418	446520802
4	4	3	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	276729	052165615X

- Split your data into two sets (training and testing)



```
#Split your data into two sets (training and testing)
from sklearn.model_selection import train_test_split
train_data, test_data=train_test_split(df_master1, test_size=0.3)
```

```
train_data.shape
```

```
(6090, 9)
```

```
test_data.shape
```

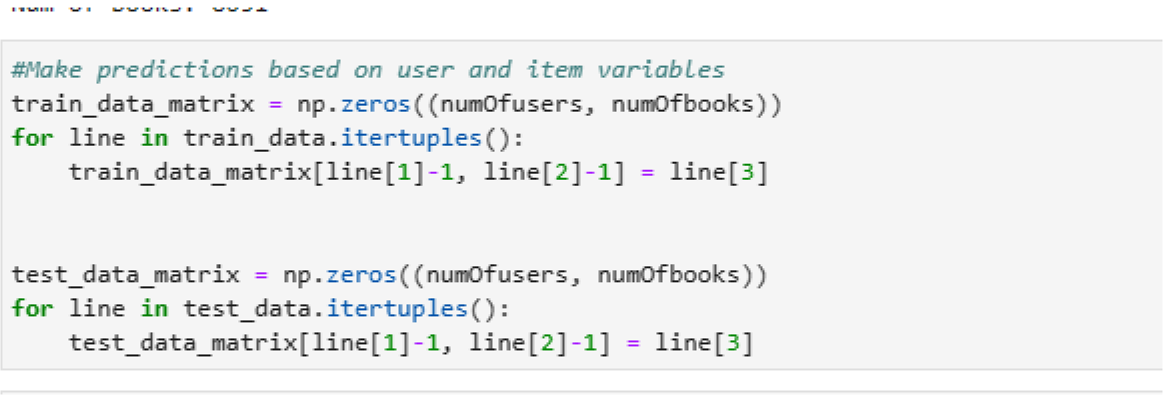
```
(2611, 9)
```

```
numOfusers = df_master1['user_id'].nunique()
numOfbooks = df_master1.isbn.nunique()
```

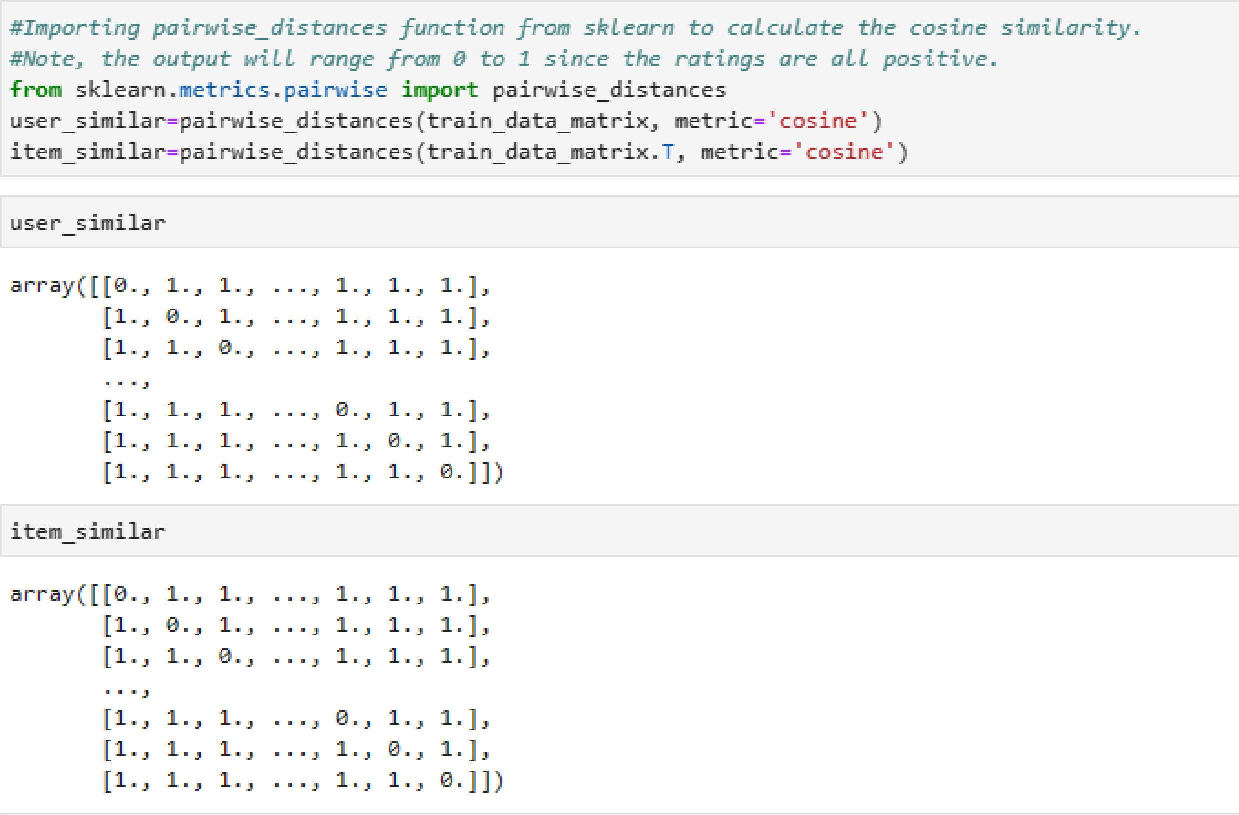
```
print('Num of Users: '+str(numOfusers))
print('Num of Books: '+str(numOfbooks))
```

```
Num of Users: 828
Num of Books: 8051
```

- Make predictions based on user and item variables



```
#Make predictions based on user and item variables
train_data_matrix = np.zeros((numOfusers, numOfbooks))
for line in train_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1] = line[3]


test_data_matrix = np.zeros((numOfusers, numOfbooks))
for line in test_data.itertuples():
    test_data_matrix[line[1]-1, line[2]-1] = line[3]
```



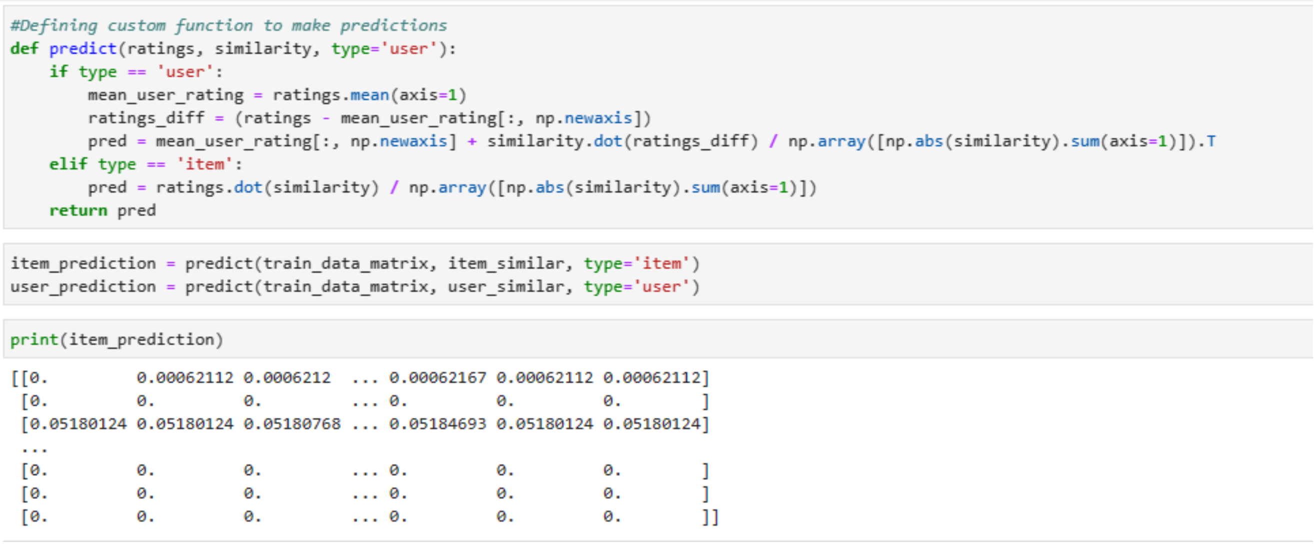
```
#Importing pairwise_distances function from sklearn to calculate the cosine similarity.
#Note, the output will range from 0 to 1 since the ratings are all positive.
from sklearn.metrics.pairwise import pairwise_distances
user_similar=pairwise_distances(train_data_matrix, metric='cosine')
item_similar=pairwise_distances(train_data_matrix.T, metric='cosine')
```

```
user_similar
```

```
array([[0., 1., 1., ..., 1., 1., 1.],
       [1., 0., 1., ..., 1., 1., 1.],
       [1., 1., 0., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 0., 1., 1.],
       [1., 1., 1., ..., 1., 0., 1.],
       [1., 1., 1., ..., 1., 1., 0.]])
```

```
item_similar
```

```
array([[0., 1., 1., ..., 1., 1., 1.],
       [1., 0., 1., ..., 1., 1., 1.],
       [1., 1., 0., ..., 1., 1., 1.],
       ...,
       [1., 1., 1., ..., 0., 1., 1.],
       [1., 1., 1., ..., 1., 0., 1.],
       [1., 1., 1., ..., 1., 1., 0.]])
```

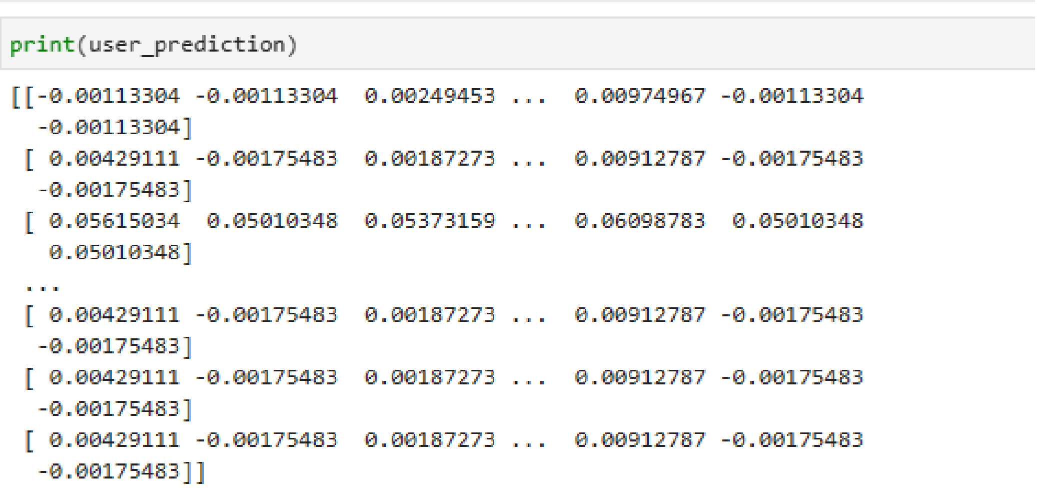


```
#Defining custom function to make predictions
def predict(ratings, similarity, type='user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)
        ratings_diff = (ratings - mean_user_rating[:, np.newaxis])
        pred = mean_user_rating[:, np.newaxis] + similarity.dot(ratings_diff) / np.array([np.abs(similarity).sum(axis=1)]).T
    elif type == 'item':
        pred = ratings.dot(similarity) / np.array([np.abs(similarity).sum(axis=1)])
    return pred
```

```
item_prediction = predict(train_data_matrix, item_similar, type='item')
user_prediction = predict(train_data_matrix, user_similar, type='user')
```

```
print(item_prediction)
```

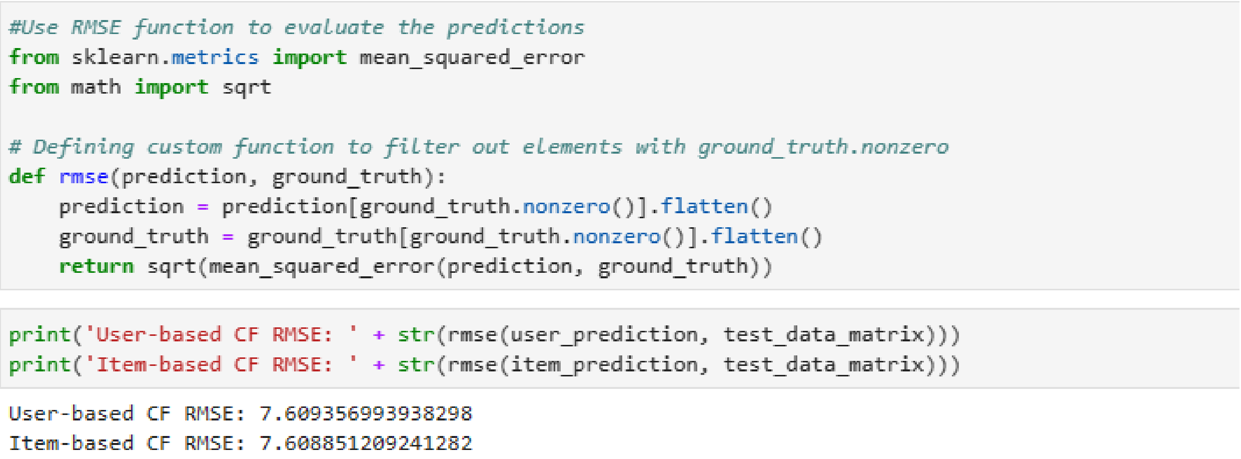
```
[[0.         0.00062112 0.0006212  ... 0.00062167 0.00062112 0.00062112]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.05180124 0.05180124 0.05180768 ... 0.05184693 0.05180124 0.05180124]
 ...
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```



```
print(user_prediction)
```

```
[[-0.00113304 -0.00113304  0.00249453 ...  0.00974967 -0.00113304
  -0.00113304]
 [ 0.00429111 -0.00175483  0.00187273 ...  0.00912787 -0.00175483
  -0.00175483]
 [ 0.05615034  0.05010348  0.05373159 ...  0.06098783  0.05010348
   0.05010348]
 ...
 [ 0.00429111 -0.00175483  0.00187273 ...  0.00912787 -0.00175483
  -0.00175483]
 [ 0.00429111 -0.00175483  0.00187273 ...  0.00912787 -0.00175483
  -0.00175483]
 [ 0.00429111 -0.00175483  0.00187273 ...  0.00912787 -0.00175483
  -0.00175483]]
```

- Use RMSE Root Mean Squared Error to evaluate the predictions



```
#Use RMSE function to evaluate the predictions
from sklearn.metrics import mean_squared_error
from math import sqrt

# Defining custom function to filter out elements with ground_truth.nonzero
def rmse(prediction, ground_truth):
    prediction = prediction[ground_truth.nonzero()].flatten()
    ground_truth = ground_truth[ground_truth.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, ground_truth))
```

```
print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
```

```
User-based CF RMSE: 7.609356993938298
Item-based CF RMSE: 7.608851209241282
```