Book Rental Recommendation

```
# import libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import pairwise_distances
from sklearn.metrics import mean_squared_error
from math import sqrt

import warnings
warnings.filterwarnings('ignore')
```

Read datasets and explore them

```
# load datasets

user_df = pd.read_csv('BX-Users.csv', encoding='latin-1')
user_df.head()
```

```
  user_id                            Location   Age
0       1                 nyc, new york, usa   NaN
1       2           stockton, california, usa  18.0
2       3     moscow, yukon territory, russia   NaN
3       4           porto, v.n.gaia, portugal  17.0
4       5  farnborough, hants, united kingdom   NaN
```

```
user_df.shape
```

```
(278859, 3)
```

```
user_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278859 entries, 0 to 278858
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   user_id   278859 non-null  object
 1   Location  278858 non-null  object
 2   Age       168096 non-null  float64
dtypes: float64(1), object(2)
memory usage: 6.4+ MB
```

```
books_df = pd.read_csv('BX-Books.csv', encoding='latin-1')
books_df.head()
```

```
        isbn                                          book_title  \
0  195153448                                Classical Mythology
1    2005018                                       Clara Callan
2   60973129                               Decision in Normandy
3  374157065  Flu: The Story of the Great Influenza Pandemic...
4  393045218                              The Mummies of Urumchi

            book_author year_of_publication
publisher
0    Mark P. O. Morford               2002     Oxford University
Press
1  Richard Bruce Wright               2001       HarperFlamingo
Canada
2          Carlo D'Este               1991
HarperPerennial
3      Gina Bari Kolata               1999        Farrar Straus
Giroux
4       E. J. W. Barber               1999  W. W. Norton &amp;amp;
Company
```

```
books_df.shape
```

```
(271379, 5)
```

```
books_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271379 entries, 0 to 271378
Data columns (total 5 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   isbn                 271379 non-null  object
 1   book_title           271379 non-null  object
 2   book_author          271378 non-null  object
 3   year_of_publication  271379 non-null  object
 4   publisher            271377 non-null  object
dtypes: object(5)
memory usage: 10.4+ MB
```

```
ratings_df = pd.read_csv('BX-Book-Ratings.csv', encoding='latin-1',
nrows=30000)
ratings_df.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_df.shape
```

```
(30000, 3)
```

```
ratings_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30000 entries, 0 to 29999
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   user_id  30000 non-null  int64
 1   isbn     30000 non-null  object
 2   rating   30000 non-null  int64
dtypes: int64(2), object(1)
memory usage: 703.2+ KB
```

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

Lets check unique users and movies in rating data.

```
ratings_df['user_id'].nunique()
```

```
3445
```

```
ratings_df['isbn'].nunique()
```

```
23987
```

From this, we can say, not all movies got rating by all the users.

So we can clean up the NaN values in books and user dataframe and filter out the ratings based on that.

Clean Up NaN values

For user data, lets remove the user with no location details and for Age, we can fill with median value.

```
user_df = user_df.dropna(subset=['Location'], axis=0)
```

```
user_df['Age'] = user_df['Age'].replace(np.nan,
user_df['Age'].median())
```

```
user_df.isna().sum().sum()
```

```
0
```

For books data, lets drop the rows with any NaN value.

```
books_df.dropna(inplace=True)
```

```
# final shape

print(user_df.shape)
print(books_df.shape)
```

```
(278858, 3)
(271376, 5)
```

Still we have enough data to proceed.

Now lets combine rating and books datasets into one.

```
df = pd.merge(ratings_df, books_df, on='isbn')
df.head()
```

```
   user_id        isbn  rating            book_title      book_author
\
0   276725  034545104X       0  Flesh Tones: A Novel       M. J. Rose

1     2313  034545104X       5  Flesh Tones: A Novel       M. J. Rose

2     6543  034545104X       0  Flesh Tones: A Novel       M. J. Rose

3   276726   155061224       5      Rites of Passage       Judith Rae

4   276727   446520802       0          The Notebook  Nicholas Sparks


  year_of_publication         publisher
0                2002  Ballantine Books
1                2002  Ballantine Books
2                2002  Ballantine Books
3                2001            Heinle
4                1996      Warner Books
```

```
# get unique users and books

n_users = df.user_id.nunique()
n_books = df.isbn.nunique()

print('Num. of Users: '+ str(n_users))
print('Num of Books: '+str(n_books))
```

```
Num. of Users: 2979
Num of Books: 20044
```

Convert ibsn and user_id to numeric value in order

```
isbn_list = df.isbn.unique()
userid_list = df.user_id.unique()

def get_isbn_numeric_id(isbn):
    itemindex = np.where(isbn_list==isbn)
    return itemindex[0][0]
```

```
def get_user_id_numeric_id(user_id):
    itemindex = np.where(userid_list==user_id)
    return itemindex[0][0]
```

Now we can convert the ibsn and user id to numeric values in order of 0,1,2,3.. n

```
df['user_id_order'] = df['user_id'].apply(get_user_id_numeric_id)

df['isbn_id'] = df['isbn'].apply(get_isbn_numeric_id)
```

Re-index the columns to build a matrix

```
new_col_order = ['user_id_order', 'isbn_id', 'rating', 'book_title',
'book_author','year_of_publication','publisher','isbn','user_id']
df = df.reindex(columns= new_col_order)
df.head()
```

```
   user_id_order  isbn_id  rating            book_title
book_author  \
0              0        0       0  Flesh Tones: A Novel       M. J.
Rose
1              1        0       5  Flesh Tones: A Novel       M. J.
Rose
2              2        0       0  Flesh Tones: A Novel       M. J.
Rose
3              3        1       5      Rites of Passage       Judith
Rae
4              4        2       0          The Notebook  Nicholas
Sparks

  year_of_publication         publisher        isbn  user_id
0                2002  Ballantine Books  034545104X   276725
1                2002  Ballantine Books  034545104X     2313
2                2002  Ballantine Books  034545104X     6543
3                2001            Heinle   155061224   276726
4                1996      Warner Books   446520802   276727
```

Split your data into two sets (training and testing)

```
train_data, test_data = train_test_split(df, test_size=0.30)
```

Make predictions based on user and item variables

We can build a rating matrix and get the user and item similarity matrix from it.

Then based on it, we can predict ratings for user and movies.

```
# rating matrix for train and test

train_data_matrix = np.zeros((n_users, n_books))
for line in train_data.itertuples():
    train_data_matrix[line[1]-1, line[2]-1] = line[3]

test_data_matrix = np.zeros((n_users, n_books))
```

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

# Get user and item similarity

user_similarity = pairwise_distances(train_data_matrix,
metric='cosine')
item_similarity = pairwise_distances(train_data_matrix.T,
metric='cosine')

print(train_data_matrix.shape)
print(user_similarity.shape)
print(item_similarity.shape)
```

```
(2979, 20044)
(2979, 2979)
(20044, 20044)
```

```
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_sim_prediction = predict(train_data_matrix, item_similarity,
type='item')
user_sim_prediction = predict(train_data_matrix, user_similarity,
type='user')
```

Use RMSE to evaluate the predictions

```
def rmse(prediction, actual):
    prediction = prediction[actual.nonzero()].flatten()
    actual = actual[actual.nonzero()].flatten()
    return sqrt(mean_squared_error(prediction, actual))

print('User-based-sim CF RMSE: ' + str(rmse(user_sim_prediction,
test_data_matrix)))
print('Item-based-sim CF RMSE: ' + str(rmse(item_sim_prediction,
test_data_matrix)))
```

```
User-based-sim CF RMSE: 7.858512237437868
Item-based-sim CF RMSE: 7.858927804991649
```

We got similar result with both user and item based similarity.