In [2]:
```
import numpy as np
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

In [78]:
```
df_user = pd.read_csv('BX-Users.csv', encoding = 'latin-1')
```

In [79]:
```
df_user.head()
```

Out[79]:

	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

Check for null values

In [80]:
```
df_user.isnull.any()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [80], in <cell line: 1>()
----> 1 df_user.isnull.any()

AttributeError: 'function' object has no attribute 'any'
```

In [81]:
```
df_user.isnull.sum()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [81], in <cell line: 1>()
----> 1 df_user.isnull.sum()

AttributeError: 'function' object has no attribute 'sum'
```

Dropping null values

In [8]:
```
df_user1 = df_user.dropna()
```

In [9]:
```
df_user1.isnull().any()
```

Out[9]:
```
user_id     False
Location    False
Age         False
dtype: bool
```

In [19]:
```
df_books = pd.read_csv('BX-Books.csv' , encoding = 'latin-1')
```

In [20]:
```
df_books.head()
```

Out[20]:

	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 & Company

In [25]:

```
#Read only 1st 10000 rows otherwise there could be chance of out of memory error co
df_bookrating = pd.read_csv('BX-Book-Ratings.csv' , encoding = 'latin-1',nrows=100(
```

In [26]:

```
df_bookrating.head()
```

Out[26]:

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

In [27]:

```
#it shows basic statistical details like percentile, mean and SD
df_bookrating.describe()
```

Out[27]:

	user_id	rating
count	10000.000000	10000.000000
mean	265844.379600	1.974700
std	56937.189618	3.424884
min	2.000000	0.000000
25%	277478.000000	0.000000
50%	278418.000000	0.000000
75%	278418.000000	4.000000
max	278854.000000	10.000000

Merge the DataFrames

```
 * For all practical purposes, user data is not required so ignore
dataframe df_user
 * Merge 2 Dataframe df_bookrating and df_books
```

In [28]:

```
#Inner join between 2 dataframes, Common b/w both the data frames are isbn value.
df = pd.merge(df_bookrating,df_books, on= 'isbn')
```

```
df.head()
```

Out[28]:

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

In [29]:

```
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: 828
Num of books: 8051
```

Convert ISBN to Numeric Number in order

In [30]:

```
#International Standard Book Number(ISBN)
isbn_list = df.isbn.unique()
```

In [31]:

```
print("Length of isbn list:" ,len(isbn_list))
```

```
Length of isbn list: 8051
```

In [32]:

```
isbn_list
```

Out[32]:

```
array(['034545104X', '155061224', '446520802', ..., '425098834',
       '425163407', '425164403'], dtype=object)
```

In [35]:

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

Do the same for the user_id, convert it into numeric and in order

In [36]:

```
userid_list = df.user_id.unique()
print('Length of user_id list:', len(isbn_list))

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

```
Length of user_id list: 8051
```

Converting both user_id and isbn to ordered list i.e., from 0...n-1

In [37]:
```
df['user_id_order'] = df['user_id'].apply(get_user_id_numeric)
```

In [38]:
```
df['isbn_id'] = df['isbn'].apply(get_isbn_numeric_id)
df.head()
```

Out[38]:

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

Re-index columns to build matrix

In [39]:
```
new_col_order = ['user_id_order','isbn_id','rating','book_title','book_author','yea
df = df.reindex(columns=new_col_order)
df.head()
```

Out[39]:

	user_id_order	isbn_id	rating	book_title	book_author	year_of_publication	publisher	
0	0	0	0	Flesh Tones: A Novel	M. J. Rose	2002	Ballantine Books	034545
1	1	1	5	Rites of Passage	Judith Rae	2001	Heinle	15506
2	2	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	44652
3	3	2	0	The Notebook	Nicholas Sparks	1996	Warner Books	44652
4	4	3	3	Help!: Level 1	Philip Prowse	1999	Cambridge University Press	052165

Train Test Split

Recommendation Systems are difficult to evaluate, but you will still learn how to evaluate them. In order to do this, you'll split your data into two sets. However, you won't do your classic X_train,X_test,y_train,y_test split. Instead, you can actually just segement the data into two sets of data:

In [41]:
```
from sklearn.model_selection import train_test_split
```

```
train_data, test_data = train_test_split(df,test_size=0.3)
```

Approach: You Will Use Memory-Based Collaborative Filtering Memory-Based Collaborative Filtering approaches can be divided into two main sections: user-item filtering and item-item filtering.

A user-item filtering will take a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked.

In contrast, item-item filtering will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items as input and outputs other items as recommendations.

Item-Item Collaborative Filtering: “Users who liked this item also liked …” User-Item Collaborative Filtering: “Users who are similar to you also liked …” In both cases, you create a user-book matrix which is built from the entire dataset.

Since you have split the data into testing and training, you will need to create two [828 x 8051] matrices (all users by all books). This is going to be a very large matrix

The training matrix contains 70% of the ratings and the testing matrix contains 30% of the ratings.

In [50]:
```
#Create two user_book matrices, one for training and another for testing
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]   #Update the ratings for eacl

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]
```

In [51]:
```
train_data_matrix
```

Out[51]:
```
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.]])
```

- You can use the **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.

In [52]:
```
from sklearn.metrics.pairwise import pairwise_distances
user_similarity = pairwise_distances(train_data_matrix,metric='cosine')
item_similarity = pairwise_distances(train_data_matrix.T,metric='cosine')
```

In [55]:
```
user_similarity
```

Out[55]:
```
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.]])
```

In [59]:
```
def predict(ratings, similarity, type= 'user'):
    if type == 'user':
        mean_user_rating = ratings.mean(axis=1)  #axis=1 is for coloumnwise
        ratings_diff = (ratings-mean_user_rating[:,np.newaxis])
        #you can use np.newaxis so that mean_user_rating has same format as rating
        pred = mean_user_rating[:,np.newaxis] + similarity.dot(ratings_diff)/np.ar

    elif type == 'item':
        pred = ratings.dot(similarity)/np.array([np.abs(similarity).sum(axis=1)])

    return pred
```

In [62]:
```
item_prediction = predict(train_data_matrix, item_similarity, type= 'item')
user_prediction = predict(train_data_matrix, user_similarity ,type= 'user')
```

In [63]:
```
item_prediction
```

Out[63]:
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.0536646 , 0.0536646 , 0.05367126, ..., 0.0536646 , 0.0536646 ,
        0.0536646 ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

Evaluation

There are many evaluation metrics, but one of the most popular metric used to evaluate accuracy of predicted ratings is Root Mean Squared Error (RMSE).

In [66]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
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))
```

In [69]:
```
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: 7.7149018877414735
Item-based CF RMSE: 7.7142662490421925
```

In [68]:
```
user_prediction[0].max()
```

Out[68]:
```
0.0393262299875776
```

In [74]:
```
np.where(user_prediction[0] == user_prediction[0].max())
```

Out[74]:
```
(array([652], dtype=int64),)
```

Both the approach yield almost same result