Problem Statement:

Generate the features from the dataset and use them to recommend the books accordingly to the users.

Recommendation systems:

are among the most popular applications of data science. They are used to predict the Rating or Preference that a user would give to an item.

Almost every major company has applied them in some form or the other: Amazon uses it to suggest products to customers, YouTube uses it to decide which video to play next on auto play, and Facebook uses it to recommend pages to like and people to follow.

In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users (items being movies to watch, text to read, products to buy, or anything else depending on industries). Recommender systems are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors. The main objective is to create a book recommendation system for users.

Import Libraries:

#Importing modules

import pandas as pd # Loading data

import numpy as np # manipulating data

import matplotlib.pyplot as plt # for visualization

import seaborn as sns # for visualization

import sys # providing access to system-specific parameters and functions.

import random # providing functionality for generating random numbers, shuffling

data, and making random selections

# This is to suppress the warning messages (if any) generated in our code

import warnings # to ignore warnings which

warnings.filterwarnings('ignore')

The Data:

The Book-Crossing dataset comprises 3 files.  
● Users :  
Contains the users. Note that user IDs (User-ID) have been anonymized and map to integers. Demographic data is provided (Location, Age) if available. Otherwise, these fields contain NULL values.  
● Books :  
Books are identified by their respective ISBN. Invalid ISBNs have already been removed from the dataset. Moreover, some content-based information is given (Book-Title, Book-Author, Year-Of-Publication, Publisher), obtained from Amazon Web Services. Note that in the case of several authors, only the first is provided. URLs linking to cover images are also given, appearing in three different flavors (Image-URL-S, Image-URL-M, Image-URL-L), i.e., small, medium, large. These URLs point to the Amazon website.  
● Ratings :  
Contains the book rating information. Ratings (Book-Rating) are either explicit, expressed on a scale from 1-10 (higher values denoting higher appreciation), or implicit, expressed by 0. by 0. by 0.

Loading data:

# user Data

users = pd.read\_csv('Users.csv',encoding='ISO-8859-1') # Load CSV file with ISO-8859-1 encoding

specifies the character encoding used to decode the text data

in the CSV file

users.head() # printing top head of users file

# Books Data

books= pd.read\_csv('Books.csv',encoding='ISO-8859-1')

books.head()

# Rating Data

ratings= pd.read\_csv('Ratings.csv',encoding='ISO-8859-1')

ratings.head()

EDA:

Dimension of dataset

# dimension of dataset

print(f'''\t  Book\_df shape is {books.shape}

          Ratings\_df shape is {ratings.shape}

          Users\_df shape is {users.shape}''')

1. Users\_Dataset

# This function calculates and displays missing values in a Pandas DataFrame.

def missing\_values(df):

    mis\_val = df.isnull().sum() # - identifies missing values (NaN) in the DataFrame.

calculates the total number of missing values for each column.

    mis\_val\_percent = round(df.isnull().mean().mul(100),2) # calculates the percentage of missing values

for each column.

.mul(100) converts the percentage to a whole number.

round(..., 2) rounds the percentage to two decimal places.

    mz\_table = pd.concat([mis\_val,mis\_val\_percent],axis=1) # - Combines missing value count and

percentage into a single table.

    mz\_table = mz\_table.rename(

    columns = {df.index.name:'col\_name',0:'Missing Values',1:'% of Total Values'}) # Renames columns for

clarity.

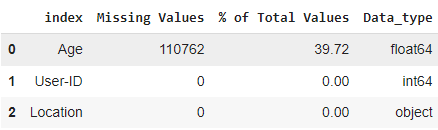
    mz\_table['Data\_type'] = df.dtypes # Adds data type of each column.

    mz\_table = mz\_table.sort\_values('% of Total Values',ascending=False) # Sorts table by percentage of

missing values in descending order.

    return mz\_table.reset\_index() # Returns summary table with index reset.

missing\_values(users) # calling define function for users table for missing values



# Age have around 39% missing values.

# Age Distribution

# This code generates a histogram to visualize the age distribution of users.

users.Age.hist(bins=[0, 10, 20, 30, 40, 50, 100])

# - users.Age selects the Age column from the users DataFrame.

- .hist() creates a histogram.

- bins=[0, 10, 20, 30, 40, 50, 100] specifies the age ranges (bins) for the histogram:

- 0-9 years

- 10-19 years

- 20-29 years

- 30-39 years

- 40-49 years

- 50-99 years

plt.title('Age Distribution\n')

plt.xlabel('Age')

plt.ylabel('Count')

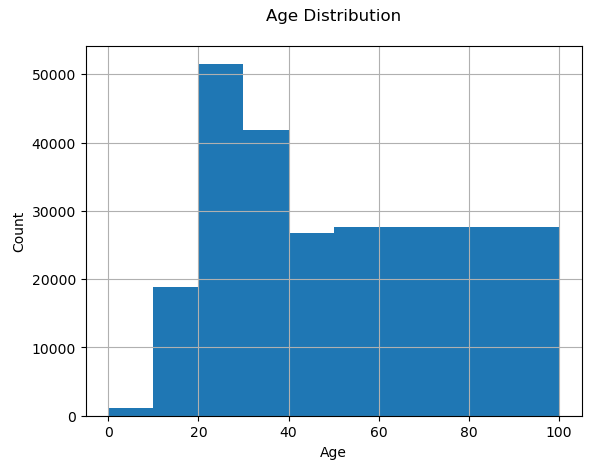
# - plt.title() sets the title of the plot.

# - plt.xlabel() sets the x-axis label.

# - plt.ylabel() sets the y-axis label.

plt.show()

# - Displays the plot.



# The most active users are among those in their 20–30s.

Let's check for outliers in age column

# This code creates a boxplot to visualize the distribution of ages in the users dataset and identify potential outliers.

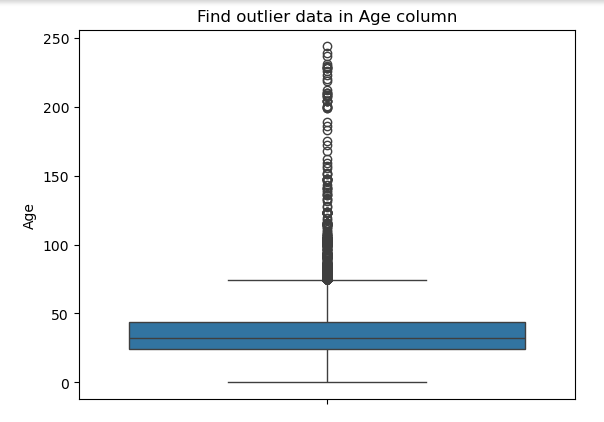
sns.boxplot(y='Age', data=users)

# - sns.boxplot() creates a boxplot using Seaborn.

- y='Age' specifies the column to plot (Age).

- data=users provides the dataset (users).

plt.title('Find outlier data in Age column') # - Sets the title of the plot.



print(sorted(users.Age.unique())) # finding how many values

Age : 244

# We have Outlier data in Age

Let's treat outliers in users age

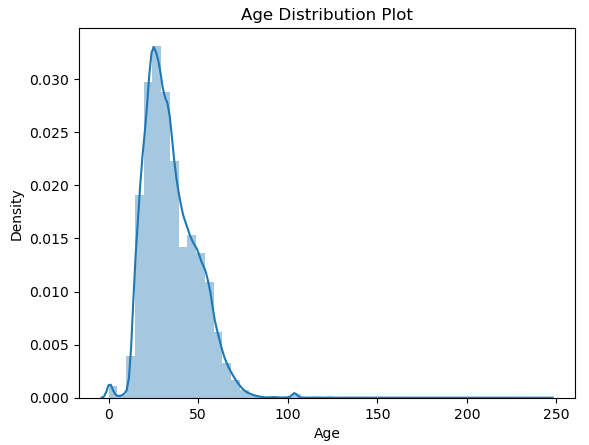
# This code creates a distribution plot (histogram with kernel density estimate) to visualize the age distribution of users.

sns.distplot(users.Age)

# - sns.distplot() creates a distribution plot using Seaborn.

- users.Age selects the Age column from the users DataFrame.

plt.title('Age Distribution Plot') # giving title



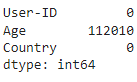
# Age value's below 5 and above 100 do not make much sense for our

# book rating case...hence replacing these by NaNs

# outlier data became NaN

users.loc[(users.Age > 100) | (users.Age < 5), 'Age'] = np.nan

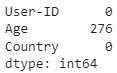
users.isna().sum()



#Age has positive Skewness (right tail) so we can use median to fill Nan values, but #for this we don't like to fill Nan value just for one range of age. To handle this #we'll use country column to fill Nan.

users['Age'] = users['Age'].fillna(users.groupby(Location)['Age'].transform('median'))

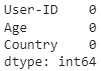
users.isna().sum()



# Still we have 276 Nan values let’s fill then with mean

users['Age'].fillna(users.Age.mean(),inplace=True)

users.isna().sum()



# Let's find our unique value in Location column

users.Location.unique()

users.Location.nunique()

57339

# 57339 unique value it’s really hard to understand

# So, lets create column Country

for i in users:

    users['Country']=users.Location.str.extract(r'\,+\s?(\w\*\s?\w\*)\"\*$')

( ‘ ‘ ‘

1. for i in users: - This loop iterates over each row in the 'users' DataFrame.

1. users['Country']=... - This assigns the extracted country values to a new column called 'Country'.

1. users.Location.str.extract(...) - This extracts the country from the 'Location' column.

1. r'\,+\s?(\w\*\s?\w\*)\"\*$' - This is the regular expression pattern used for extraction.

Pattern explanation:

- \, matches a comma.

- + matches one or more occurrences.

- \s? matches an optional whitespace.

- (\w\*\s?\w\*) captures one or more word characters (letters, numbers, underscores), optionally

followed by whitespace and more word characters.

- \"\* matches zero or more double quotes.

- $ asserts the end of the string.

‘ ‘ ‘)

users.Country.nunique()

529

#drop location column

users.drop('Location',axis=1,inplace=True)

users.head(2)

users.isnull().sum()

users['Country']=users['Country'].astype('str')

a=list(users.Country.unique())

a=set(a)

a=list(a)

a = [x for x in a if x is not None]

a.sort()

print(a)

# Some data has Misspellings Let's correct it.

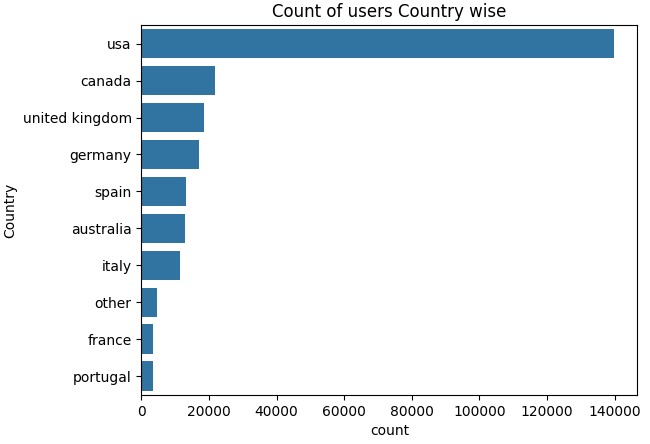
users['Country'].replace(['','01776','02458','19104','23232','30064','85021','87510','alachua','america','austria','autralia','cananda','geermany','italia','united kindgonm','united sates','united staes','united state','united states','us'],

                           ['other','usa','usa','usa','usa','usa','usa','usa','usa','usa','australia','australia','canada','germany','italy','united kingdom','usa','usa','usa','usa','usa'],inplace=True)

plt.figure(figsize=(15,7))

sns.countplot(y='Country',data=users,order=pd.value\_counts(users['Country']).iloc[:10].index)

plt.title('Count of users Country wise')



# Most number of users are from USA

# 2. Books\_Dataset

books.head(2)

#dropping last three columns containing image URLs which will not be required for analysis

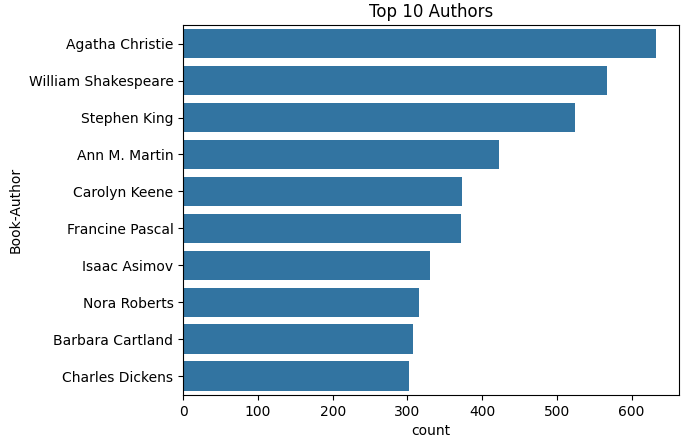
books.drop(['Image-URL-S', 'Image-URL-M', 'Image-URL-L'],axis=1,inplace=True)

# Top 10 Authors which have written the most books.

#plt.figure(figsize=(15,7))

sns.countplot(y='Book-Author',data=books,order=pd.value\_counts(books['Book-Author']).iloc[:10].index)

plt.title('Top 10 Authors')

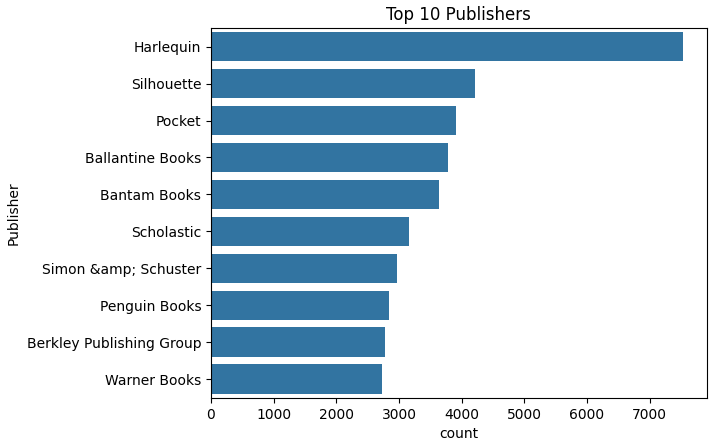


# Top 10 Publisher which have published the most books.

plt.figure(figsize=(15,7))

sns.countplot(y='Publisher',data=books,order=pd.value\_counts(books['Publisher']).iloc[:10].index)

plt.title('Top 10 Publishers')



books['Year-Of-Publication']=books['Year-Of-Publication'].astype('str')

a=list(books['Year-Of-Publication'].unique())

a=set(a)

a=list(a)

a = [x for x in a if x is not None]

a.sort()

print(a)

#investigating the rows having 'DK Publishing Inc' as yearOfPublication

books.loc[books['Year-Of-Publication'] == 'DK Publishing Inc',:]

# As it can be seen from above that there are some incorrect entries in Year-Of-Publication field. It looks like Publisher names 'DK Publishing Inc' and 'Gallimard' have been incorrectly loaded as Year-Of-Publication in dataset due to some errors in csv file

#From above, it is seen that bookAuthor is incorrectly loaded with bookTitle, hence making required corrections

#ISBN '0789466953'

books.loc[books.ISBN == '0789466953','Year-Of-Publication'] = 2000

books.loc[books.ISBN == '0789466953','Book-Author'] = "James Buckley"

books.loc[books.ISBN == '0789466953','Publisher'] = "DK Publishing Inc"

books.loc[books.ISBN == '0789466953','Book-Title'] = "DK Readers: Creating the X-Men, How Comic Books Come to Life (Level 4: Proficient Readers)"

#ISBN '078946697X'

books.loc[books.ISBN == '078946697X','Year-Of-Publication'] = 2000

books.loc[books.ISBN == '078946697X','Book-Author'] = "Michael Teitelbaum"

books.loc[books.ISBN == '078946697X','Publisher'] = "DK Publishing Inc"

books.loc[books.ISBN == '078946697X','Book-Title'] = "DK Readers: Creating the X-Men, How It All Began (Level 4: Proficient Readers)"

#rechecking

books.loc[(books.ISBN == '0789466953') | (books.ISBN == '078946697X'),:]

#corrections done

#investigating the rows having 'Gallimard' as yearOfPublication

books.loc[books['Year-Of-Publication'] == 'Gallimard',:]

#making required corrections as above, keeping other fields intact

books.loc[books.ISBN == '2070426769','Year-Of-Publication'] = 2003

books.loc[books.ISBN == '2070426769','Book-Author'] = "Jean-Marie Gustave Le ClÃ?Â©zio"

books.loc[books.ISBN == '2070426769','Publisher'] = "Gallimard"

books.loc[books.ISBN == '2070426769','Book-Title'] = "Peuple du ciel, suivi de 'Les Bergers"

books.loc[books.ISBN == '2070426769',:]

books['Year-Of-Publication']=pd.to\_numeric(books['Year-Of-Publication'], errors='coerce')

print(sorted(books['Year-Of-Publication'].unique()))

#Now it can be seen that yearOfPublication has all values as integers

# The value 0 for Year-Of\_Publication is invalid and as this dataset was published in 2004, We have assumed that the years after 2006 to be invalid and setting invalid years as NaN

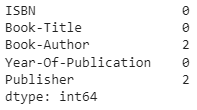
Reference of the fact: [http://www2.informatik.uni-freiburg.de/~cziegler/BX/](https://www.google.com/url?q=http%3A%2F%2Fwww2.informatik.uni-freiburg.de%2F%7Ecziegler%2FBX%2F)

books.loc[(books['Year-Of-Publication'] > 2006) | (books['Year-Of-Publication'] == 0),'Year-Of-Publication'] = np.NAN

#replacing NaNs with median value of Year-Of-Publication

books['Year-Of-Publication'].fillna(round(books['Year-Of-Publication'].median()), inplace=True)

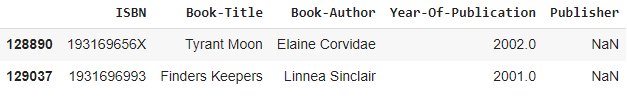
books.isna().sum()



#exploring 'publisher' column

books.loc[books.Publisher.isnull(),:]

#two NaNs

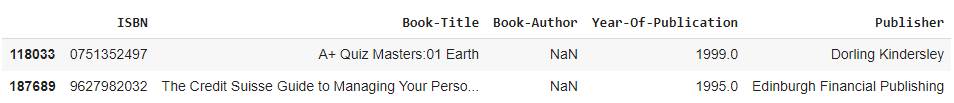


#Filling Nan of Publisher with others

books.Publisher.fillna('other',inplace=True)

#exploring 'Book-Author' column

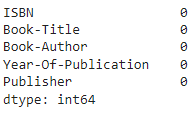
books.loc[books['Book-Author'].isnull(),:]



#Filling Nan of Book-Author with others

books['Book-Author'].fillna('other',inplace=True)

books.isna().sum()



# 3. Ratings\_Dataset

ratings.head(2)

# Ratings dataset should have books only which exist in our books dataset

ratings\_new = ratings[ratings.ISBN.isin(books.ISBN)]

ratings.shape,ratings\_new.shape

((1149780, 3), (1031136, 3))

# It can be seen that many rows having book ISBN not part of books dataset got dropped off

# Ratings dataset should have ratings from users which exist in users dataset.

print("Shape of dataset before dropping",ratings\_new.shape)

ratings\_new = ratings\_new[ratings\_new['User-ID'].isin(users['User-ID'])]

print("shape of dataset after dropping",ratings\_new.shape)

# same before and after dropping it can be seen that no user was there in ratings dataset.

# Let's see how the ratings are distributed

plt.rc("font", size=15)

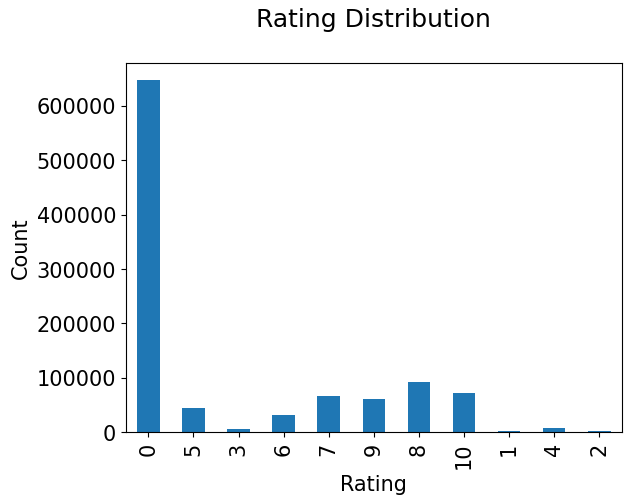
ratings\_new['Book-Rating'].value\_counts(sort=False).plot(kind='bar')

plt.title('Rating Distribution\n')

plt.xlabel('Rating')

plt.ylabel('Count')

plt.show()



# The ratings are very unevenly distributed, and the vast majority of ratings are 0 .As quoted in the description of the dataset - BX-Book-Ratings contains the book rating information. Ratings are either explicit, expressed on a scale from 1-10 higher values denoting higher appreciation, or implicit, expressed by 0.Hence segragating implicit and explict ratings datasets

#Hence segragating implicit and explict ratings datasets

ratings\_explicit = ratings\_new[ratings\_new['Book-Rating'] != 0]

ratings\_implicit = ratings\_new[ratings\_new['Book-Rating'] == 0]

print('ratings\_explicit dataset shape',ratings\_explicit.shape)

print('ratings\_implicit dataset',ratings\_implicit.shape)

* ratings\_explicit dataset shape (383842, 3)
* ratings\_implicit dataset (647294, 3)

plt.style.use('fivethirtyeight')

plt.figure(figsize=(12, 8))

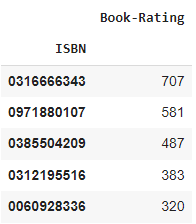
sns.countplot(data=ratings\_explicit , x='Book-Rating', palette='rocket\_r')

# It can be observe that higher ratings are more common amongst users and rating 8 has been rated highest number of times

# Let's find the top 5 books which are rated by most number of users.

rating\_count = pd.DataFrame(ratings\_explicit.groupby('ISBN')['Book-Rating'].count())

rating\_count.sort\_values('Book-Rating', ascending=False).head()

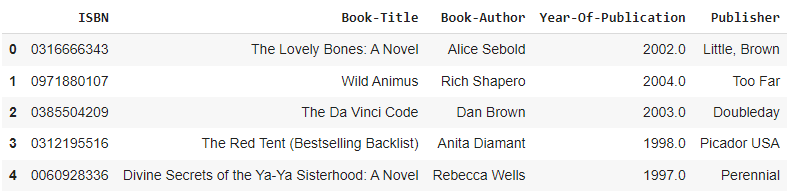


# The book with ISBN '0316666343' received the most rating counts. Let’s find out what book it is, and what books are in the top 5.

most\_rated\_books = pd.DataFrame(['0316666343', '0971880107', '0385504209', '0312195516', '0060928336'], index=np.arange(5), columns = ['ISBN'])

most\_rated\_books\_summary = pd.merge(most\_rated\_books, books, on='ISBN')

most\_rated\_books\_summary



# The book that received the most rating counts in this data set is Rich Shapero’s “Wild Animus”. And there is something in common among these five books that received the most rating counts — they are all novels. So it is conclusive that novels are popular and likely receive more ratings.

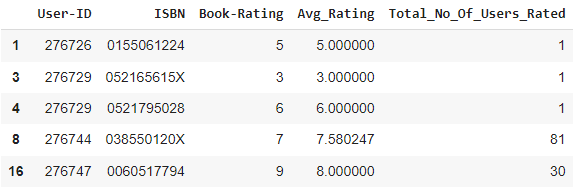
# Create column Rating average

ratings\_explicit['Avg\_Rating']=ratings\_explicit.groupby('ISBN')['Book-Rating'].transform('mean')

# Create column Rating sum

ratings\_explicit['Total\_No\_Of\_Users\_Rated']=ratings\_explicit.groupby('ISBN')['Book-Rating'].transform('count')

ratings\_explicit.head()



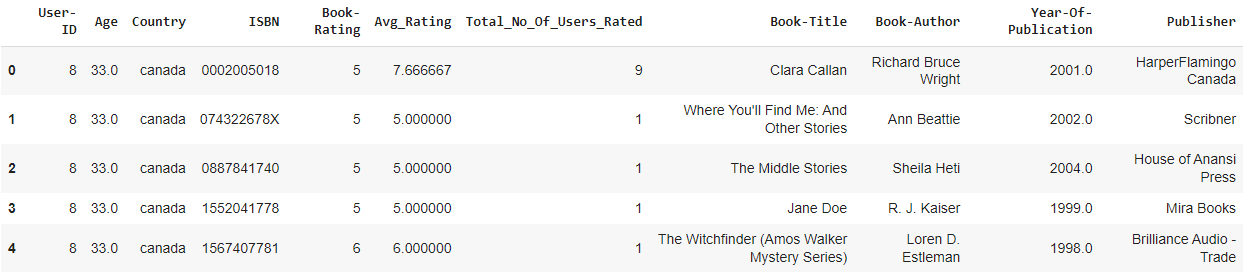
# Merging All Dataset.

Final\_Dataset=users.copy()

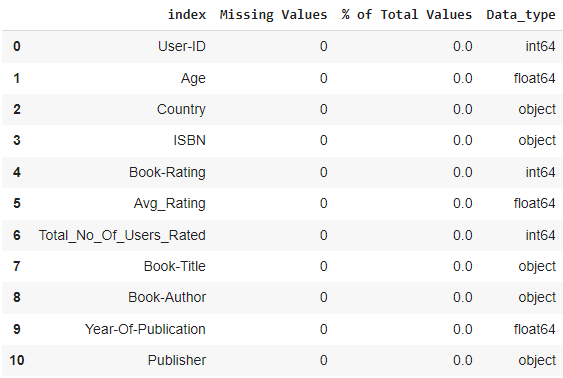
Final\_Dataset=pd.merge(Final\_Dataset,ratings\_explicit,on='User-ID')

Final\_Dataset=pd.merge(Final\_Dataset,books,on='ISBN')

Final\_Dataset.head()



missing\_values(Final\_Dataset)



Popularity Based Filtering

#### As the name suggests Popularity based recommendation system works with the trend. It basically uses the items which are in trend right now. For example, if any book which is usually bought by every new user then there are chances that it may suggest that book to the user who just signed up.  
Book weighted avg formula:  
**Weighted Rating(WR)=[vR/(v+m)]+[mC/(v+m)]**  
where,  
**v** is the number of votes for the books;  
**m** is the minimum votes required to be listed in the chart;  
**R** is the average rating of the book; and  
**C** is the mean vote across the whole report.  
Now we find the values of v,m,R,C.

# This code calculates the average rating and filters top-rated books based on the number of user ratings.

C= Final\_Dataset['Avg\_Rating'].mean()

m= Final\_Dataset['Total\_No\_Of\_Users\_Rated'].quantile(0.90) # calculates the 90th percentile (i.e., top

10%) of total ratings.

Top\_Books = Final\_Dataset.loc[Final\_Dataset['Total\_No\_Of\_Users\_Rated'] >= m] # filters books with total

ratings above the 90th percentile.

print(f'C={C} , m={m}')

Top\_Books.shape

#C=7.626700569505161 , m=64.0

#(38570, 11)

##### Here we used 90th percentile as our cutoff. In other words, for a book to feature in the charts, it must have more votes than at least 90% of the books in the list. We see that there are 38570 books which qualify to be in this list. Now, we need to calculate our metric for each qualified book. To do this, we will define a function, weighted\_rating() and define a new feature score, of which we’ll calculate the value by applying this function to our DataFrame of qualified books:

# This function calculates a weighted rating for a book based on its average rating and number of user ratings.

def weighted\_rating(x, m=m, C=C):

    v = x['Total\_No\_Of\_Users\_Rated']

    R = x['Avg\_Rating']

    return (v/(v+m) \* R) + (m/(m+v) \* C)

# The weighted rating formula is based on the Bayesian weighted average:

‘ ‘ ‘ How it works:

1. For books with many ratings (v >> m), the first term dominates, and the weighted rating approaches the book's average rating (R).

2. For books with few ratings (v << m), the second term dominates, and the weighted rating approaches the overall mean rating (C).

3. For books with moderate ratings (v ≈ m), the weighted rating balances between the book's average rating and the overall mean rating. ’ ‘ ‘

Top\_Books['Score'] = Top\_Books.apply(weighted\_rating,axis=1)

#Keeping only one entry of each book

Top\_Books=Top\_Books.sort\_values('Score', ascending=False).drop\_duplicates('ISBN').sort\_index()

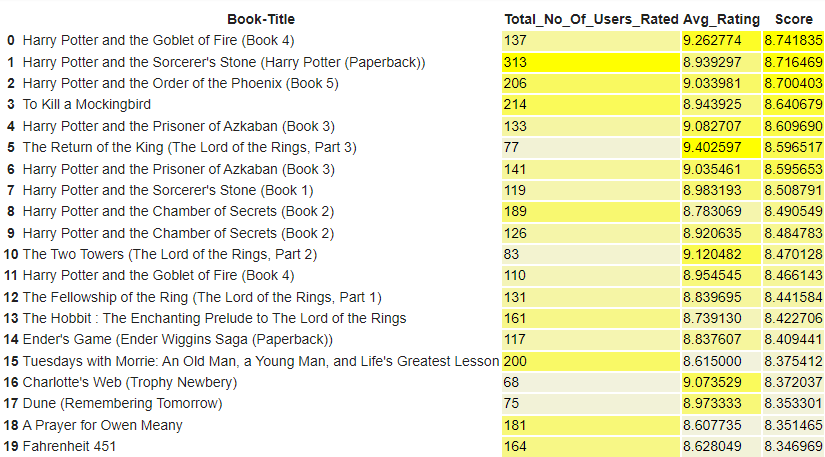
cm=sns.light\_palette('yellow',as\_cmap=True)

#Sorting books based on score calculated above

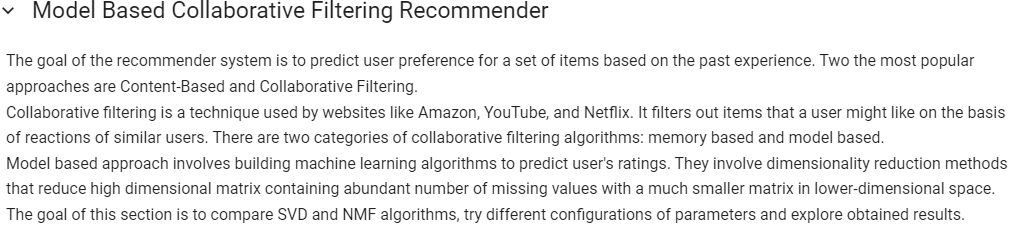
Top\_Books = Top\_Books.sort\_values('Score', ascending=False)

#Printing the top 20 books

Top\_Books[['Book-Title', 'Total\_No\_Of\_Users\_Rated', 'Avg\_Rating', 'Score']].reset\_index(drop=True).head(20).style.background\_gradient(cmap=cm)



#The Popularity based recommender provide a general chart of recommended books to all the users. #They are not sensitive to the interests and tastes of a particular user.



import scipy

import math

import sklearn

from nltk.corpus import stopwords

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from scipy.sparse.linalg import svds

import matplotlib.pyplot as plt

1. scipy: Scientific Computing Library for Python

- Provides functions for scientific and engineering applications

- Includes modules for linear algebra, optimization, signal processing, and more

2. math: Built-in Python Math Library

- Provides mathematical functions (e.g., sin, cos, exp)

3. sklearn: Scikit-learn Library

- Machine learning library for Python

- Includes algorithms for classification, regression, clustering, and more

4. nltk: Natural Language Toolkit Library

- Library for natural language processing tasks

- Includes tools for text processing, tokenization, and corpora management

5. matplotlib: Data Visualization Library

- Provides functions for creating plots and charts

# Filter: users with at least 3 ratings

# Number of records: 327271

# This code filters a dataset of book ratings to include only the top-rated books based on a specified threshold.

book\_ratings\_threshold\_perc = 0.1

book\_ratings\_threshold = len(df\_ratings\_top['isbn'].unique()) \* book\_ratings\_threshold\_perc

filter\_books\_list = df\_ratings\_top['isbn'].value\_counts().head(int(book\_ratings\_threshold)).index.to\_list()

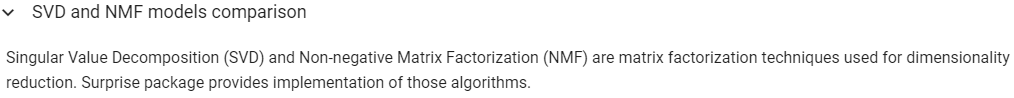
df\_ratings\_top = df\_ratings\_top[df\_ratings\_top['isbn'].isin(filter\_books\_list)]

print('Filter: top %d%% most frequently rated books\nNumber of records: %d' % (book\_ratings\_threshold\_perc\*100, len(df\_ratings\_top))) # - Filter the dataset to include only the

selected ISBNs

# Filter: top 10% most frequently rated books

# Number of records: 160787



from surprise import Dataset, Reader

from surprise import SVD, NMF

from surprise.model\_selection import cross\_validate, train\_test\_split, GridSearchCV

df=df\_ratings\_top.copy()

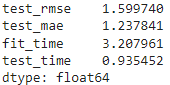
reader = Reader(rating\_scale=(1, 10))

data = Dataset.load\_from\_df(df[['user\_id', 'isbn', 'book\_rating']], reader)

model\_svd = SVD()

cv\_results\_svd = cross\_validate(model\_svd, data, cv=3)

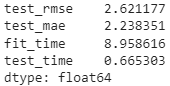
pd.DataFrame(cv\_results\_svd).mean()

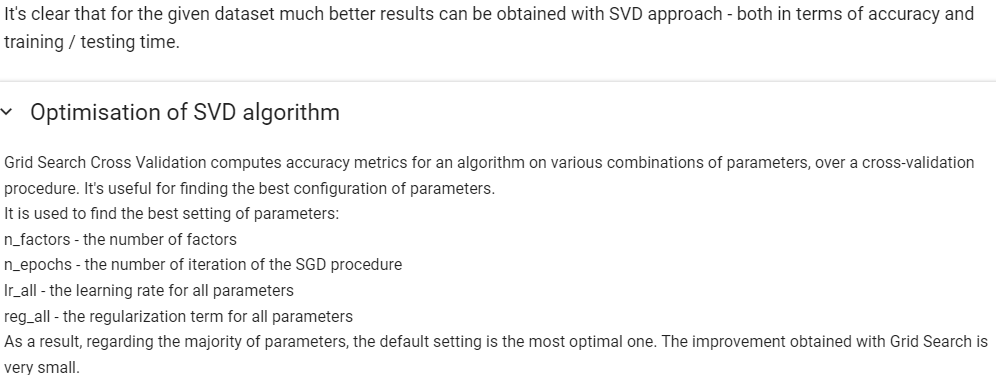


model\_nmf = NMF()

cv\_results\_nmf = cross\_validate(model\_nmf, data, cv=3)

pd.DataFrame(cv\_results\_nmf).mean()





# This code defines a parameter grid for hyperparameter tuning in a machine learning model, specifically for a recommender system using matrix factorization.

param\_grid = {'n\_factors': [80,100],

              'n\_epochs': [5, 20],

              'lr\_all': [0.002, 0.005],

              'reg\_all': [0.2, 0.4]}

# This code performs hyperparameter tuning for a Singular Value Decomposition (SVD)-based recommender system using Grid Search Cross-Validation.

gs = GridSearchCV(SVD, param\_grid, measures=['rmse', 'mae'], cv=3)

gs.fit(data)

1. estimator: The model to tune (SVD).

2. param\_grid: Hyperparameter combinations to try.

3. scoring: Evaluation metrics ('rmse' and 'mae').

4. cv: Number of folds for cross-validation (3).

print(gs.best\_score['rmse'])

print(gs.best\_params['rmse'])

# 1.594457622626435

# {'n\_factors': 80, 'n\_epochs': 20, 'lr\_all': 0.005, 'reg\_all': 0.2}



trainset, testset = train\_test\_split(data, test\_size=0.2)

model = SVD(n\_factors=80, n\_epochs=20, lr\_all=0.005, reg\_all=0.2)

model.fit(trainset)

predictions = model.test(testset)

df\_pred = pd.DataFrame(predictions, columns=['user\_id', 'isbn', 'actual\_rating', 'pred\_rating', 'details'])

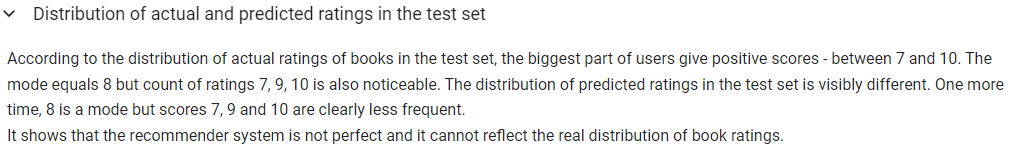
df\_pred['impossible'] = df\_pred['details'].apply(lambda x: x['was\_impossible'])

df\_pred['pred\_rating\_round'] = df\_pred['pred\_rating'].round()

df\_pred['abs\_err'] = abs(df\_pred['pred\_rating'] - df\_pred['actual\_rating'])

df\_pred.drop(['details'], axis=1, inplace=True)

df\_pred.sample(5)



palette = sns.color\_palette("RdBu", 10)

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(14, 4))

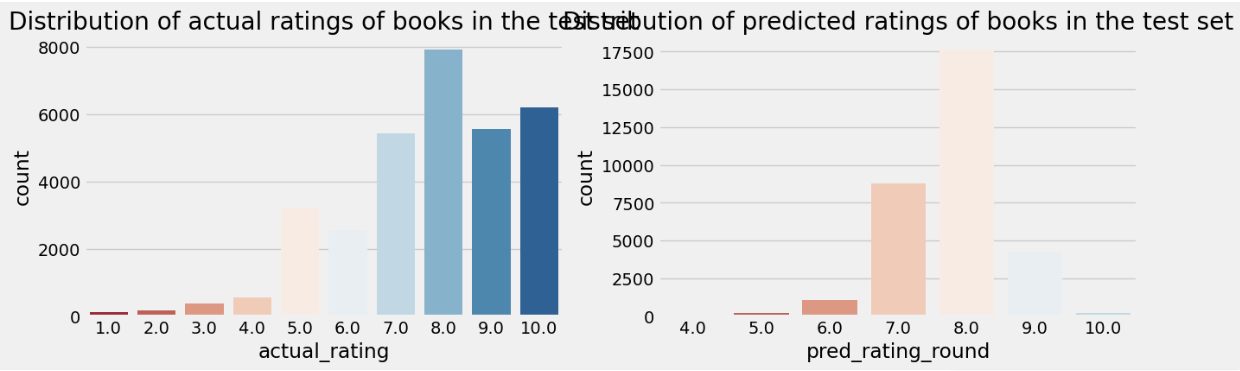
sns.countplot(x='actual\_rating', data=df\_pred, palette=palette, ax=ax1)

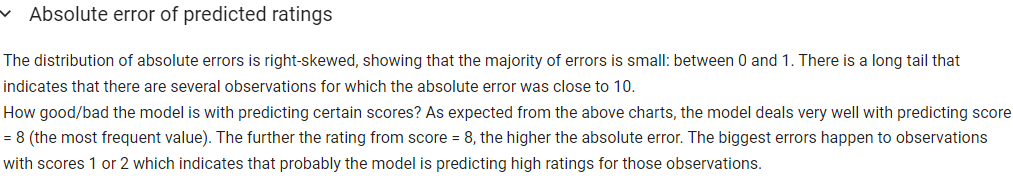
ax1.set\_title('Distribution of actual ratings of books in the test set')

sns.countplot(x='pred\_rating\_round', data=df\_pred, palette=palette, ax=ax2)

ax2.set\_title('Distribution of predicted ratings of books in the test set')

plt.show()





df\_pred\_err = df\_pred.groupby('actual\_rating')['abs\_err'].mean().reset\_index()

fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(14, 4))

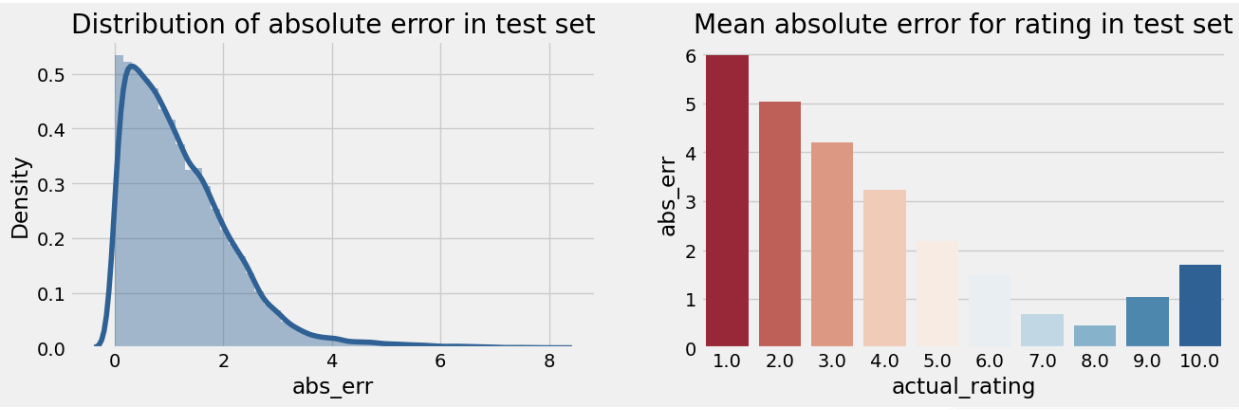
sns.distplot(df\_pred['abs\_err'], color='#2f6194', ax=ax1)

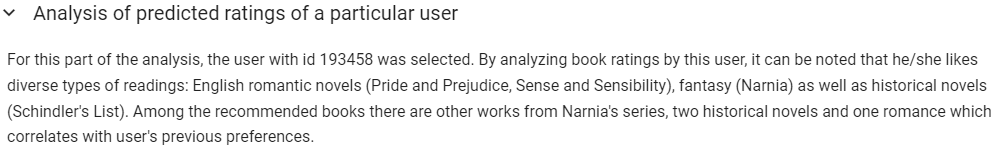
ax1.set\_title('Distribution of absolute error in test set')

sns.barplot(x='actual\_rating', y='abs\_err', data=df\_pred\_err, palette=palette, ax=ax2)

ax2.set\_title('Mean absolute error for rating in test set')

plt.show()





df\_books = books.copy()

df\_books.rename(columns = {'ISBN':'isbn' ,'Book-Title':'book\_title'},inplace=True)

df\_ext = df.merge(df\_books[['isbn', 'book\_title']], on='isbn', how='left')

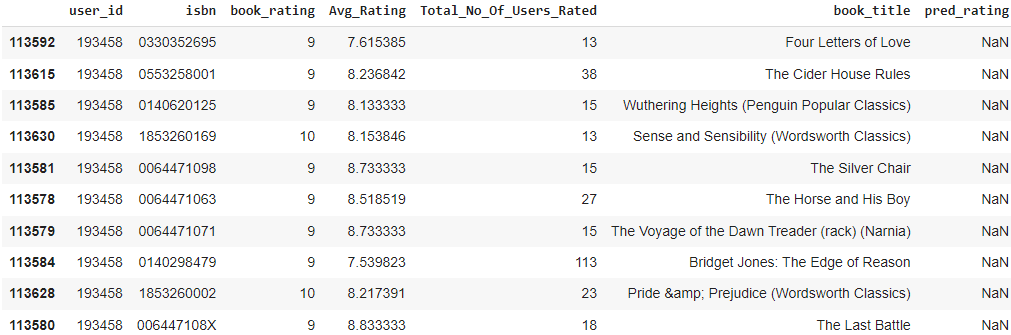
df\_ext = df\_ext.merge(df\_pred[['isbn', 'user\_id', 'pred\_rating']], on=['isbn', 'user\_id'], how='left')



selected\_user\_id = 193458

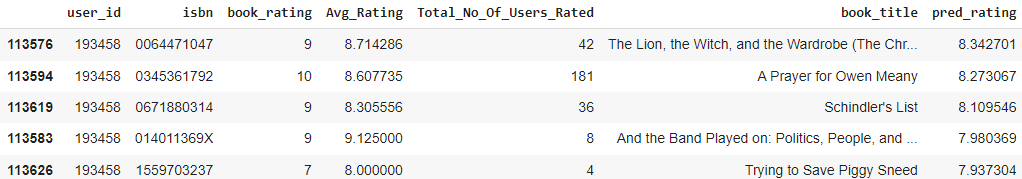
df\_user = df\_ext[df\_ext['user\_id']==selected\_user\_id]

df\_user[(df\_user['pred\_rating'].isna())&(df\_user['book\_rating']>=9)].sample(10)



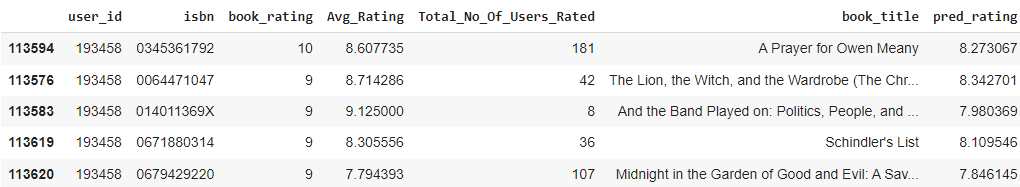


df\_user[df\_user['pred\_rating'].notna()].sort\_values('pred\_rating', ascending=False).head(5)





df\_user[df\_user['pred\_rating'].notna()].sort\_values('book\_rating', ascending=False).head(5)





from sklearn.neighbors import NearestNeighbors

from scipy.spatial.distance import correlation

from sklearn.metrics.pairwise import pairwise\_distances

import ipywidgets as widgets

from IPython.display import display, clear\_output

from contextlib import contextmanager

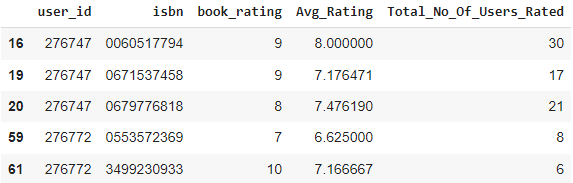
import numpy as np

import os, sys

import re

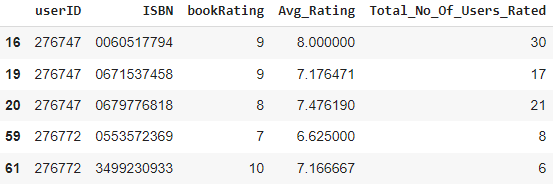
from scipy.sparse import csr\_matrix

df\_ratings\_top.head()



df\_ratings\_top.rename(columns={'user\_id':'userID' ,'isbn':'ISBN','book\_rating':'bookRating'},inplace=True)

df\_ratings\_top.head()





#Generating ratings matrix from explicit ratings table

ratings\_matrix = df\_ratings\_top.pivot(index='userID', columns='ISBN', values='bookRating')

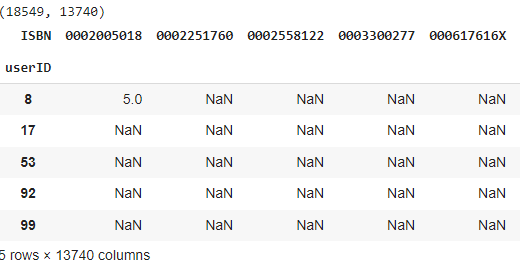
userID = ratings\_matrix.index

ISBN = ratings\_matrix.columns

print(ratings\_matrix.shape)

ratings\_matrix.head()

#Notice that most of the values are NaN (undefined) implying absence of ratings



n\_users = ratings\_matrix.shape[0] #considering only those users who gave explicit ratings

n\_books = ratings\_matrix.shape[1]

print (n\_users, n\_books)

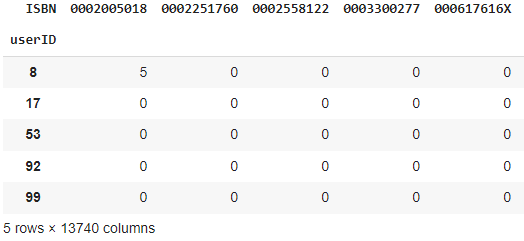
#18549 13740

ratings\_matrix.fillna(0, inplace = True)

ratings\_matrix = ratings\_matrix.astype(np.int32)

#checking first few rows

ratings\_matrix.head(5)



sparsity = 1.0-len(ratings\_explicit)/float(ratings\_explicit.shape[0]\*n\_books)

print ('The sparsity level of Book Crossing dataset is ' +  str(sparsity\*100) + ' %')

# The sparsity level of Book Crossing dataset is 99.99272197962155 %

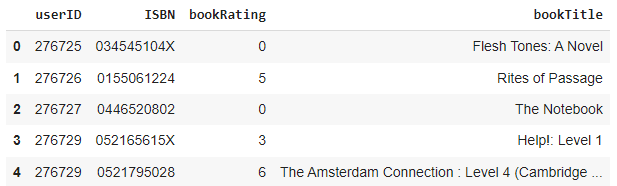
combine\_book\_rating = pd.merge(ratings, books, on = 'ISBN')

columns = ['Book-Author','Year-Of-Publication', 'Publisher']

combine\_book\_rating = combine\_book\_rating.drop(columns, axis = 1)

combine\_book\_rating.rename(columns={'User-ID':'userID','Book-Title':'bookTitle','Book-Rating':'bookRating'},inplace=True)

combine\_book\_rating.head()



combine\_book\_rating = combine\_book\_rating.dropna(axis = 0, subset = ['bookTitle'])

book\_ratingcount = (combine\_book\_rating.

                    groupby(by = ['bookTitle',])['bookRating'].

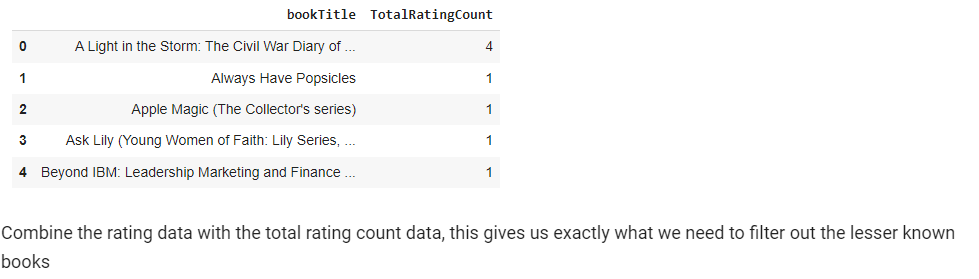
                    count().

                    reset\_index().

                    rename(columns = {'bookRating':'TotalRatingCount'})

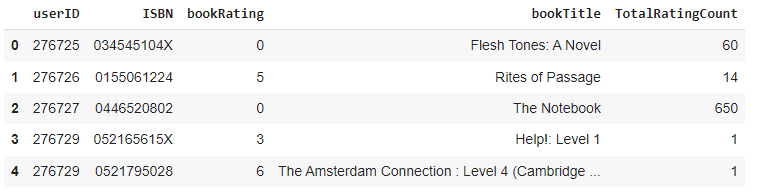
                    [['bookTitle','TotalRatingCount']])

book\_ratingcount.head()



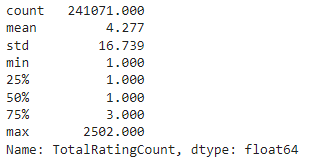
rating\_with\_totalratingcount = combine\_book\_rating.merge(book\_ratingcount, left\_on = 'bookTitle', right\_on = 'bookTitle', how = 'inner' )

rating\_with\_totalratingcount.head()



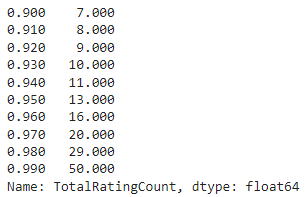
pd.set\_option('display.float\_format', lambda x: '%.3f' % x)

print(book\_ratingcount['TotalRatingCount'].describe())





print(book\_ratingcount['TotalRatingCount'].quantile(np.arange(.9,1,.01)))

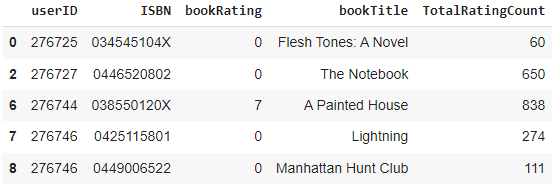




popularity\_threshold = 50

rating\_popular\_book = rating\_with\_totalratingcount.query('TotalRatingCount >= @popularity\_threshold')

rating\_popular\_book.head()



if not rating\_popular\_book[rating\_popular\_book.duplicated(['userID', 'bookTitle'])].empty:

    initial\_rows = rating\_popular\_book.shape[0]

    print('Initial dataframe shape {0}'.format(rating\_popular\_book.shape))

    rating\_popular\_book = rating\_popular\_book.drop\_duplicates(['userID', 'bookTitle'])

    current\_rows = rating\_popular\_book.shape[0]

    print('New dataframe shape {0}'.format(rating\_popular\_book.shape))

    print('Removed {0} rows'.format(initial\_rows - current\_rows))



us\_canada\_user\_rating\_pivot = rating\_popular\_book.pivot(index = 'bookTitle',columns = 'userID', values = 'bookRating').fillna(0)

us\_canada\_user\_rating\_matrix = csr\_matrix(us\_canada\_user\_rating\_pivot.values)



from sklearn.neighbors import NearestNeighbors

model\_knn = NearestNeighbors(metric = 'cosine', algorithm = 'brute')

model\_knn.fit(us\_canada\_user\_rating\_matrix)



query\_index = np.random.choice(us\_canada\_user\_rating\_pivot.shape[0])

distances, indices = model\_knn.kneighbors(us\_canada\_user\_rating\_pivot.iloc[query\_index, :].values.reshape((1, -1)), n\_neighbors = 6)

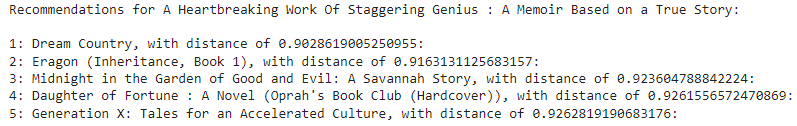
for i in range(0, len(distances.flatten())):

    if i == 0:

        print('Recommendations for {0}:\n'.format(us\_canada\_user\_rating\_pivot.index[query\_index]))

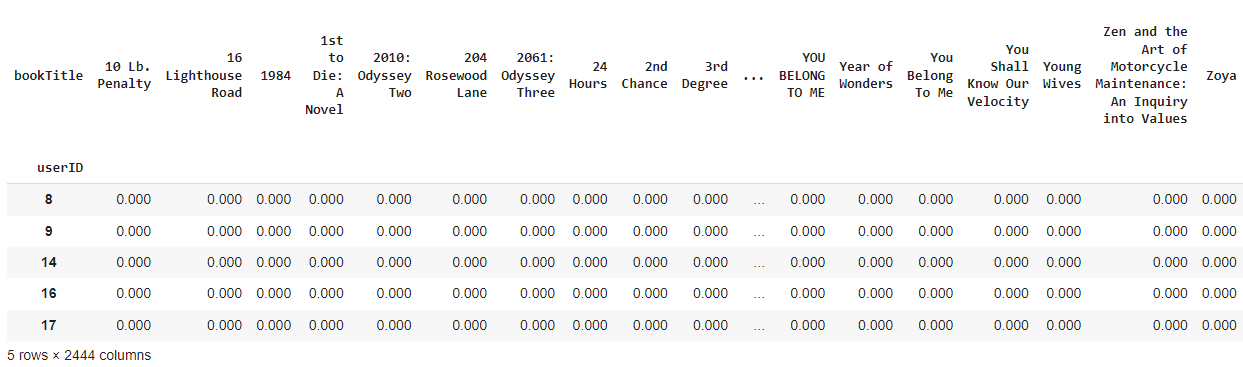
    else:

        print('{0}: {1}, with distance of {2}:'.format(i, us\_canada\_user\_rating\_pivot.index[indices.flatten()[i]], distances.flatten()[i]))



us\_canada\_user\_rating\_pivot2 = rating\_popular\_book.pivot(index = 'userID', columns = 'bookTitle', values = 'bookRating').fillna(0)

us\_canada\_user\_rating\_pivot2.head()



us\_canada\_user\_rating\_pivot2.shape

# (47994, 2444)

X = us\_canada\_user\_rating\_pivot2.values.T

X.shape

# (2444, 47994)

import sklearn

from sklearn.decomposition import TruncatedSVD

SVD = TruncatedSVD(n\_components=12, random\_state=17)

matrix = SVD.fit\_transform(X)

matrix.shape

# (2444, 12)

corr = np.corrcoef(matrix)

corr.shape

# (2444, 2444)



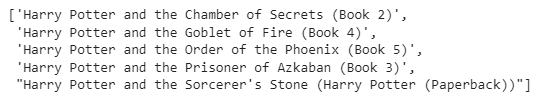
us\_canada\_book\_title = us\_canada\_user\_rating\_pivot2.columns

us\_canada\_book\_list = list(us\_canada\_book\_title)

coffey\_hands = us\_canada\_book\_list.index("Harry Potter and the Sorcerer's Stone (Book 1)")

corr\_coffey\_hands  = corr[coffey\_hands]

list(us\_canada\_book\_title[(corr\_coffey\_hands<1.0) & (corr\_coffey\_hands>0.9)])

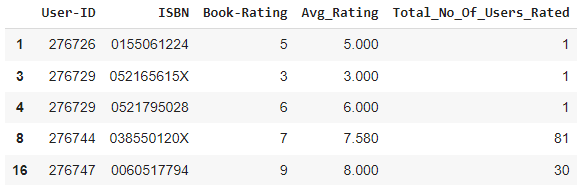




ratings\_explicit.head()

ratings\_explicit.rename(columns={'user\_id':'User-ID','isbn':'ISBN','book\_rating':'Book-Rating'},inplace=True)

ratings\_explicit.head()



users\_interactions\_count\_df = ratings\_explicit.groupby(['ISBN', 'User-ID']).size().groupby('User-ID').size()

print('# of users: %d' % len(users\_interactions\_count\_df))

users\_with\_enough\_interactions\_df = users\_interactions\_count\_df[users\_interactions\_count\_df >= 100].reset\_index()[['User-ID']]

print('# of users with at least 5 interactions: %d' % len(users\_with\_enough\_interactions\_df))



print('# of interactions: %d' % len(ratings\_explicit))

interactions\_from\_selected\_users\_df = ratings\_explicit.merge(users\_with\_enough\_interactions\_df,

               how = 'right',

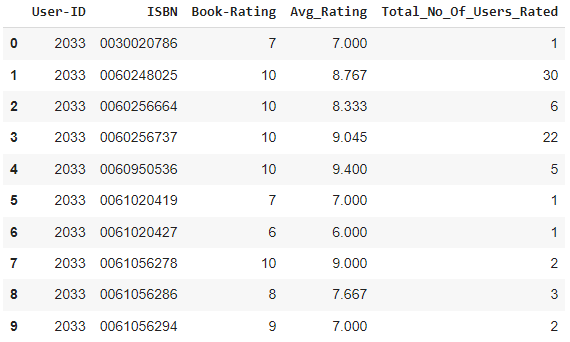
               left\_on = 'User-ID',

               right\_on = 'User-ID')

print('# of interactions from users with at least 5 interactions: %d' % len(interactions\_from\_selected\_users\_df))



interactions\_from\_selected\_users\_df.head(10)



import math

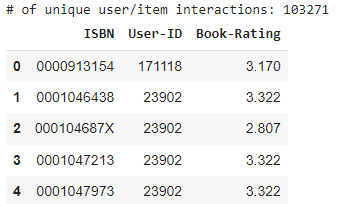
def smooth\_user\_preference(x):

    return math.log(1+x, 2)

interactions\_full\_df = interactions\_from\_selected\_users\_df.groupby(['ISBN', 'User-ID'])['Book-Rating'].sum().apply(smooth\_user\_preference).reset\_index()

print('# of unique user/item interactions: %d' % len(interactions\_full\_df))

interactions\_full\_df.head()



from sklearn.model\_selection import train\_test\_split

interactions\_train\_df, interactions\_test\_df = train\_test\_split(interactions\_full\_df,

                                   stratify=interactions\_full\_df['User-ID'],

                                   test\_size=0.20,

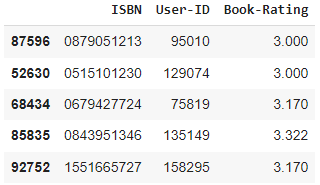
                                   random\_state=42)

print('# interactions on Train set: %d' % len(interactions\_train\_df))

print('# interactions on Test set: %d' % len(interactions\_test\_df))



interactions\_test\_df.head()



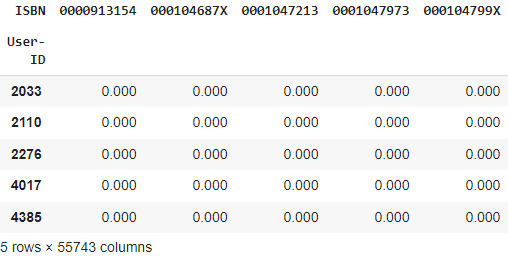
#Creating a sparse pivot table with users in rows and items in columns

users\_items\_pivot\_matrix\_df = interactions\_train\_df.pivot(index='User-ID',

                                                          columns='ISBN',

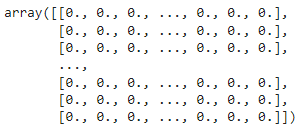
                                                          values='Book-Rating').fillna(0)

users\_items\_pivot\_matrix\_df.head()



users\_items\_pivot\_matrix = users\_items\_pivot\_matrix\_df.values

users\_items\_pivot\_matrix[:10]



users\_ids = list(users\_items\_pivot\_matrix\_df.index)

users\_ids[:10]

[2033, 2110, 2276, 4017, 4385, 5582, 6242, 6251, 6543, 6575]

from sklearn.metrics.pairwise import cosine\_similarity

from scipy.sparse.linalg import svds

# The number of factors to factor the user-item matrix.

NUMBER\_OF\_FACTORS\_MF = 15

#Performs matrix factorization of the original user item matrix

U, sigma, Vt = svds(users\_items\_pivot\_matrix, k = NUMBER\_OF\_FACTORS\_MF)

users\_items\_pivot\_matrix.shape

# (449, 55743)

U.shape

# (449, 15)

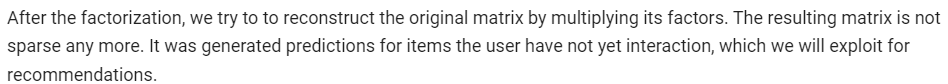
sigma = np.diag(sigma)

sigma.shape

# (15, 15)

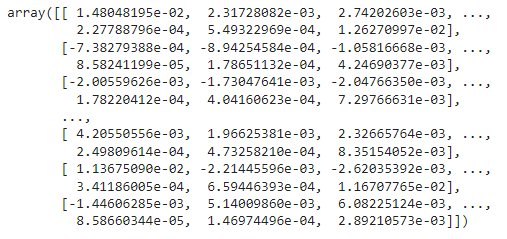
Vt.shape

# (15, 55743)



all\_user\_predicted\_ratings = np.dot(np.dot(U, sigma), Vt)

all\_user\_predicted\_ratings



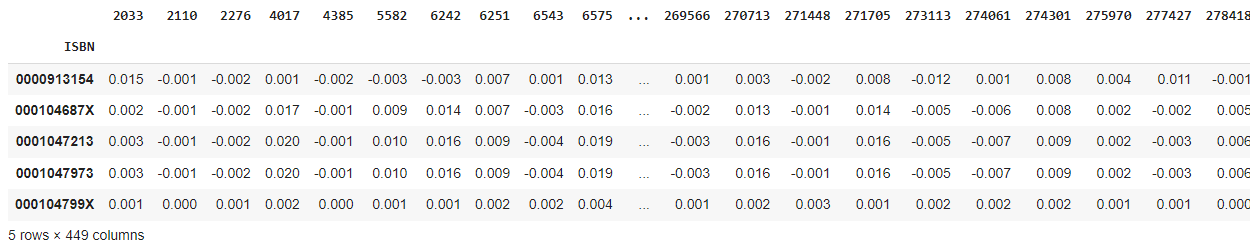
all\_user\_predicted\_ratings.shape

# (449, 55743)

#Converting the reconstructed matrix back to a Pandas dataframe

cf\_preds\_df = pd.DataFrame(all\_user\_predicted\_ratings, columns = users\_items\_pivot\_matrix\_df.columns, index=users\_ids).transpose()

cf\_preds\_df.head()

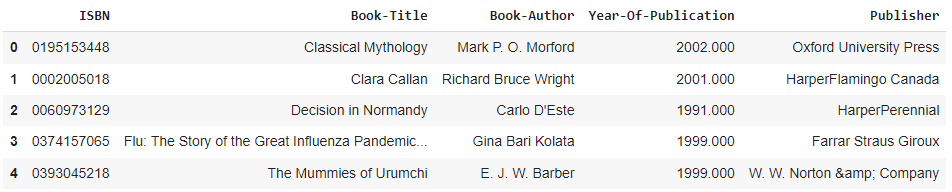


len(cf\_preds\_df.columns)

# 449

global books

books.head()



class CFRecommender:

    MODEL\_NAME = 'Collaborative Filtering'

    def \_\_init\_\_(self, cf\_predictions\_df):

        self.cf\_predictions\_df = cf\_predictions\_df

    def get\_model\_name(self):

        return self.MODEL\_NAME

    def recommend\_items(self, user\_id, items\_to\_ignore=[], topn=10):

        # Get and sort the user's predictions

        sorted\_user\_predictions = self.cf\_predictions\_df[user\_id].sort\_values(ascending=False).reset\_index().rename(columns={user\_id: 'recStrength'})

        # Recommend the highest predicted rating content that the user hasn't seen yet.

        recommendations\_df = sorted\_user\_predictions[~sorted\_user\_predictions['ISBN'].isin(items\_to\_ignore)].sort\_values('recStrength', ascending = False).head(topn)

        recommendations\_df=recommendations\_df.merge(books,on='ISBN',how='inner')

        recommendations\_df=recommendations\_df[['ISBN','Book-Title','recStrength']]

        return recommendations\_df

cf\_recommender\_model = CFRecommender(cf\_preds\_df)

#Indexing by personId to speed up the searches during evaluation

interactions\_full\_indexed\_df = interactions\_full\_df.set\_index('User-ID')

interactions\_train\_indexed\_df = interactions\_train\_df.set\_index('User-ID')

interactions\_test\_indexed\_df = interactions\_test\_df.set\_index('User-ID')

def get\_items\_interacted(UserID, interactions\_df):

    interacted\_items = interactions\_df.loc[UserID]['ISBN']

    return set(interacted\_items if type(interacted\_items) == pd.Series else [interacted\_items])

class ModelRecommender:

    # Function for getting the set of items which a user has not interacted with

    def get\_not\_interacted\_items\_sample(self, UserID, sample\_size, seed=42):

        interacted\_items = get\_items\_interacted(UserID, interactions\_full\_indexed\_df)

        all\_items = set(ratings\_explicit['ISBN'])

        non\_interacted\_items = all\_items - interacted\_items

        random.seed(seed)

        non\_interacted\_items\_sample = random.sample(non\_interacted\_items, sample\_size)

        return set(non\_interacted\_items\_sample)

    # Function to verify whether a particular item\_id was present in the set of top N recommended items

    def \_verify\_hit\_top\_n(self, item\_id, recommended\_items, topn):

            try:

                index = next(i for i, c in enumerate(recommended\_items) if c == item\_id)

            except:

                index = -1

            hit = int(index in range(0, topn))

            return hit, index

    # Function to evaluate the performance of model for each user

    def evaluate\_model\_for\_user(self, model, person\_id):

        # Getting the items in test set

        interacted\_values\_testset = interactions\_test\_indexed\_df.loc[person\_id]

        if type(interacted\_values\_testset['ISBN']) == pd.Series:

            person\_interacted\_items\_testset = set(interacted\_values\_testset['ISBN'])

        else:

            person\_interacted\_items\_testset = set([int(interacted\_values\_testset['ISBN'])])

        interacted\_items\_count\_testset = len(person\_interacted\_items\_testset)

 # Getting a ranked recommendation list from the model for a given user

        person\_recs\_df = model.recommend\_items(person\_id, items\_to\_ignore=get\_items\_interacted(person\_id, interactions\_train\_indexed\_df),topn=10000000000)

        print('Recommendation for User-ID = ',person\_id)

        print(person\_recs\_df.head(10))

        # Function to evaluate the performance of model at overall level

    def recommend\_book(self, model ,userid):

        person\_metrics = self.evaluate\_model\_for\_user(model, userid)

        return

model\_recommender = ModelRecommender()

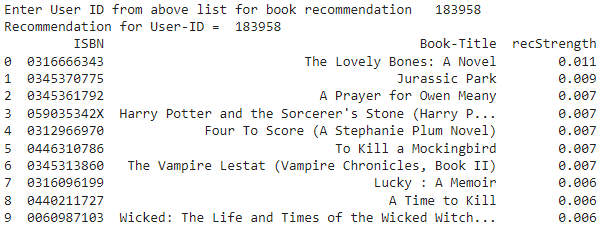


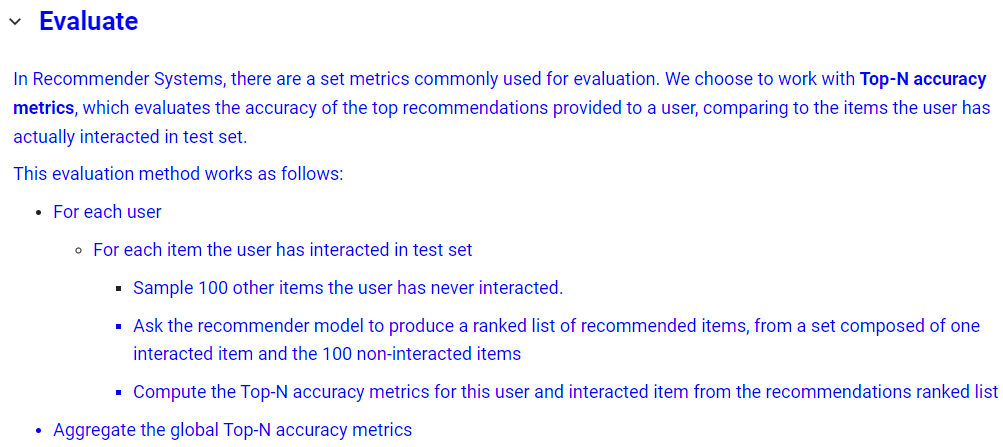
print(list(interactions\_full\_indexed\_df.index.values))



user=int(input("Enter User ID from above list for book recommendation  "))

model\_recommender.recommend\_book(cf\_recommender\_model,user)





#Top-N accuracy metrics consts

EVAL\_RANDOM\_SAMPLE\_NON\_INTERACTED\_ITEMS = 100

class ModelEvaluator:

    # Function for getting the set of items which a user has not interacted with

    def get\_not\_interacted\_items\_sample(self, UserID, sample\_size, seed=42):

        interacted\_items = get\_items\_interacted(UserID, interactions\_full\_indexed\_df)

        all\_items = set(ratings\_explicit['ISBN'])

        non\_interacted\_items = all\_items - interacted\_items

        random.seed(seed)

        # Convert the set to a list before sampling

        non\_interacted\_items\_sample = random.sample(list(non\_interacted\_items), sample\_size)

        return set(non\_interacted\_items\_sample)

    # Function to verify whether a particular item\_id was present in the set of top N recommended items

    def \_verify\_hit\_top\_n(self, item\_id, recommended\_items, topn):

            try:

                index = next(i for i, c in enumerate(recommended\_items) if c == item\_id)

            except:

                index = -1

            hit = int(index in range(0, topn))

            return hit, index

    # Function to evaluate the performance of model for each user

    def evaluate\_model\_for\_user(self, model, person\_id):

        # Getting the items in test set

        interacted\_values\_testset = interactions\_test\_indexed\_df.loc[person\_id]

        if type(interacted\_values\_testset['ISBN']) == pd.Series:

            person\_interacted\_items\_testset = set(interacted\_values\_testset['ISBN'])

        else:

            person\_interacted\_items\_testset = set([int(interacted\_values\_testset['ISBN'])])

        interacted\_items\_count\_testset = len(person\_interacted\_items\_testset)

        # Getting a ranked recommendation list from the model for a given user

        person\_recs\_df = model.recommend\_items(person\_id, items\_to\_ignore=get\_items\_interacted(person\_id, interactions\_train\_indexed\_df),topn=10000000000)

        hits\_at\_5\_count = 0

        hits\_at\_10\_count = 0

        # For each item the user has interacted in test set

        for item\_id in person\_interacted\_items\_testset:

            # Getting a random sample of 100 items the user has not interacted with

            non\_interacted\_items\_sample = self.get\_not\_interacted\_items\_sample(person\_id, sample\_size=EVAL\_RANDOM\_SAMPLE\_NON\_INTERACTED\_ITEMS, seed=item\_id)

            # Combine the current interacted item with the 100 random items and convert to list

            items\_to\_filter\_recs = list(non\_interacted\_items\_sample.union(set([item\_id])))

            # Filtering only recommendations that are either the interacted item or from a random sample of 100 non-interacted items

            valid\_recs\_df = person\_recs\_df[person\_recs\_df['ISBN'].isin(items\_to\_filter\_recs)]

            valid\_recs = valid\_recs\_df['ISBN'].values

            # Verifying if the current interacted item is among the Top-N recommended items

            hit\_at\_5, index\_at\_5 = self.\_verify\_hit\_top\_n(item\_id, valid\_recs, 5)

            hits\_at\_5\_count += hit\_at\_5

            hit\_at\_10, index\_at\_10 = self.\_verify\_hit\_top\_n(item\_id, valid\_recs, 10)

            hits\_at\_10\_count += hit\_at\_10

        # Recall is the rate of the interacted items that are ranked among the Top-N recommended items

        recall\_at\_5 = hits\_at\_5\_count / float(interacted\_items\_count\_testset)

        recall\_at\_10 = hits\_at\_10\_count / float(interacted\_items\_count\_testset)

        person\_metrics = {'hits@5\_count':hits\_at\_5\_count,

                          'hits@10\_count':hits\_at\_10\_count,

                          'interacted\_count': interacted\_items\_count\_testset,

                          'recall@5': recall\_at\_5,

                          'recall@10': recall\_at\_10}

        return person\_metrics

    # Function to evaluate the performance of model at overall level

    def evaluate\_model(self, model):

        people\_metrics = []

        for idx, person\_id in enumerate(list(interactions\_test\_indexed\_df.index.unique().values)):

            person\_metrics = self.evaluate\_model\_for\_user(model, person\_id)

            person\_metrics['User-ID'] = person\_id

            people\_metrics.append(person\_metrics)

        print('%d users processed' % idx)

        detailed\_results\_df = pd.DataFrame(people\_metrics).sort\_values('interacted\_count', ascending=False)

        global\_recall\_at\_5 = detailed\_results\_df['hits@5\_count'].sum() / float(detailed\_results\_df['interacted\_count'].sum())

        global\_recall\_at\_10 = detailed\_results\_df['hits@10\_count'].sum() / float(detailed\_results\_df['interacted\_count'].sum())

        global\_metrics = {'modelName': model.get\_model\_name(),

                          'recall@5': global\_recall\_at\_5,

                          'recall@10': global\_recall\_at\_10}

        return global\_metrics, detailed\_results\_df

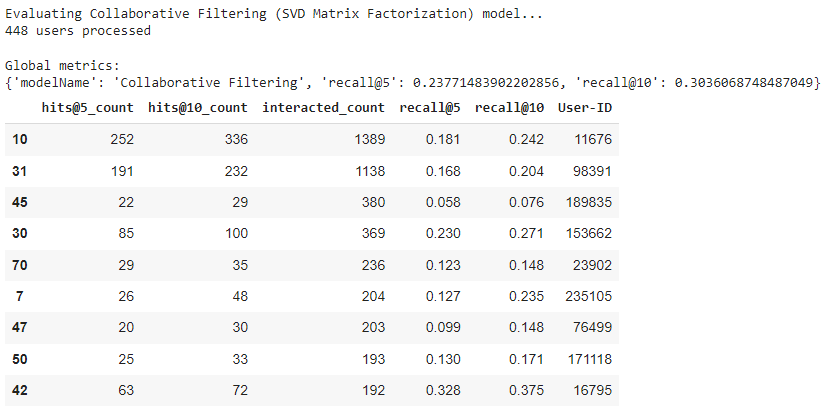
model\_evaluator = ModelEvaluator()

print('Evaluating Collaborative Filtering (SVD Matrix Factorization) model...')

cf\_global\_metrics, cf\_detailed\_results\_df = model\_evaluator.evaluate\_model(cf\_recommender\_model)

print('\nGlobal metrics:\n%s' % cf\_global\_metrics)

cf\_detailed\_results\_df.head(10)



**Conclusion**

* In EDA, the Top-10 most rated books were essentially **novels**. Books like **The Lovely Bone** and **The Secret Life of Bees** were very well perceived.
* Majority of the readers were of the **age bracket 20-35** and most of them came from North American and European countries namely **USA, Canada, UK, Germany and Spain**.
* If we look at the ratings distribution, **most of the books have high ratings** with maximum books being rated 8. Ratings below 5 are few in number.
* Author with the most books was **Agatha Christie, William Shakespeare and Stephen King**.
* For modelling, it was observed that for **model based** collaborative filtering SVD technique worked way better than NMF with lower Mean Absolute Error (MAE).
* Amongst the memory-based approach, **item-item CF performed better** than **user-user CF** because of lower computation.