Library Book Recommendation System

Abstract: -

Due to wide application of management system, information or data grows rapidly. On one hand, students have a large number of information resources. On the other hand, the time cost and difficulty of students in finding the proper information increases. To tackle the problems, book recommendation is one of the solutions for college libraries which possess huge volumes of books and reading-intensive users.

Recommender systems usually provide the student with a list of recommendations that they might prefer, or supply predictions on how much the student might prefer each item. Choosing what book to read next has always been a question for many. Even for students, deciding which textbook or reference book to read on a topic unknown to them is a big question.

In this report we try to present a model for a personalized recommendation system for books that uses hybrid recommendation approach which is combination of user-based collaborative filtering and KNN algorithm. The proposed recommendation system tries to learn the student's preferences, the rating of books provided by students and recommends the books to the student based on their preferences.

Introduction: -

Recommendation systems are software programs that help a student to find products according to their needs and interests by using the student's rating of each item and the student's preferences. A recommender system works as a helper in finding relevant and related items based either on their explicitly mentioned preferences or objective behaviours. This way, it is a big source of reducing information overload in finding relevant items in several domains including books.

In order to recommend items including books to the student. The book recommendation system would help the student to borrow a book, by recommending books based on collaborative filtering and k-nearest neighbour classification algorithm.

Problem statement:

- What is the Problem?

  Library Book Recommendation System

- Why is the Problem?

  Due to wide application of management system and information density, information data grows rapidly day by day. On one hand, student have a large number of information resources. On the other hand, the time cost and difficulty of student to finding the proper information increases. To tackle the problems, book recommendation is one of the solutions for college libraries which possess huge volumes of books and reading-intensive users.

- How is this Problem solved?

  A recommender system works as a helper in finding relevant and related items based either on their explicitly mentioned preferences or objective behaviours. This way, it is a big source of reducing information overload in finding relevant items in several domains including books.
  In order to recommend items including books to the student. The book recommendation system would help the student to borrow a book, by recommending books based on collaborative filtering and k-nearest neighbour classification algorithm.

Literature Survey

Recommender systems have become a vital research field since the emergence of the first paper on collaborative filtering in the mid- 1990s. In general, these systems are stated as the support systems which help users to find content, products, or services (such as books, movies, music, TV programs, and websites) by gathering and examining suggestions from other users, which means reviews from various establishments, and users. These systems are broadly classified into collaborative filtering (CF) and content-based filtering (CB). CF is an information filtering practice that is based on the user's evaluation of items or previous purchases records. However, this method has been known to expose two major issues that are sparsity problem and scalability problem. CB examines a set of items rated by an individual user and then uses the content of these items, as well as the provided ratings, to deduce a profile that can be used to recommend additional items of interest. However, the syntactic nature of Content based filtering to detect the similarity between items that share the same attributes or features causes overspecialized recommendations that only comprise very similar items to those the user already knows.

Tools used

- Anaconda

  It is a free and open distribution of the Python and R languages for scientific computing, that aims to simplify package management and deployment. It comes with more than 1,500 packages as well as the conda package and virtual environment manager. It also includes a GUI Anaconda Navigator, as a graphical alternative to the command line interface.

- PyCharm

  It is an IDE used for computer programming, specifically for Python language. It is developed by Czech company JetBrains. It supports wed development with Django as well as Data Science with Anaconda.

- VSCode

  It is an IDE developed by Microsoft. It is based on Electron, a framework which is used to deploy Node.js applications for the desktop running on the Bink layout engine.

- MS-Excel

  Microsoft Excel is a spreadsheet developed by Microsoft for Windows, macOS, Android and iOS. It features calculation, graphing tools, pivot tables, and a macro programming language called Visual Basic for Applications.

- Excel to CSV convertor

A website used for converting Excel file to csv.

Technology used

- Python

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

- Scikit-learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007.

  - KNN

    The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. However, it is more widely used in classification problems in the industry.

  - Cosine similarity

    Cosine similarity is the cosine of the angle between two *n*-dimensional vectors in an *n*-dimensional space. It is the dot product of the two vectors divided by the product of the two vectors' lengths (or magnitudes). Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The cosine similarity is advantageous because even if the two similar documents are far apart

by the Euclidean distance (due to the size of the document), chances are they may still be oriented closer together. The smaller the angle, higher the cosine similarity.

- SciPy

SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing. SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or MATLAB's.

- Matplotlib

  Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

  - Histogram

    A histogram is a graphical display of data using bars of different heights. In a histogram, each bar groups numbers into ranges. Taller bars show that more data falls in that range. A histogram displays the shape and spread of continuous sample data.

- Pandas

It is software library written for Python language for data manipulation and analysis. It offers data structures and operations for manipulating numeric tables and time series.

- Dataframes

Pandas allow importing data of various file formats such as csv, excel etc. Pandas allows various data manipulation operations such as group by, join, merge, melt, concatenation as well as data cleaning features such as filling, replacing or imputing null values.

- Read_csv

  read_csv is an important pandas' function to read csv files and do operations on it.

Hardware Requirements

- CPU: 2 x 64-bit 2.8 GHz 8.00 GT/s CPUs
- RAM: 8 GB (or 4 GB)
- Storage: 100 GB
- Internet access to download the files from Anaconda Cloud or a USB drive containing all of the files you need with alternate instructions for air gapped installations.

Issues or Challenges faced by Recommendation System:

1.Cold-start problem: It's difficult to give recommendations to new users as his profile is almost empty and he hasn't rated any items yet so his taste is unknown to the system. This is called the cold start problem. Items can also have a cold start when they are new in the system and haven't been rated before.

2. Scalability: With the growth of numbers of users and items, the system needs more resources for processing information and forming recommendations. Majority of resources is consumed with the purpose of determining users with similar tastes, and goods with similar descriptions.

3. Sparsity: In online shops that have a huge number of users and items there are almost always users that have rated just a few items. i.e. Students in general rate only a limited or no number of books. Using collaborative and other approaches recommender systems generally create neighborhoods of users using their profiles. If a user has evaluated just few items then it's pretty difficult to determine his taste and he/she could be related to the wrong neighborhood. Sparsity is the problem of lack of information.

Dataset Collection

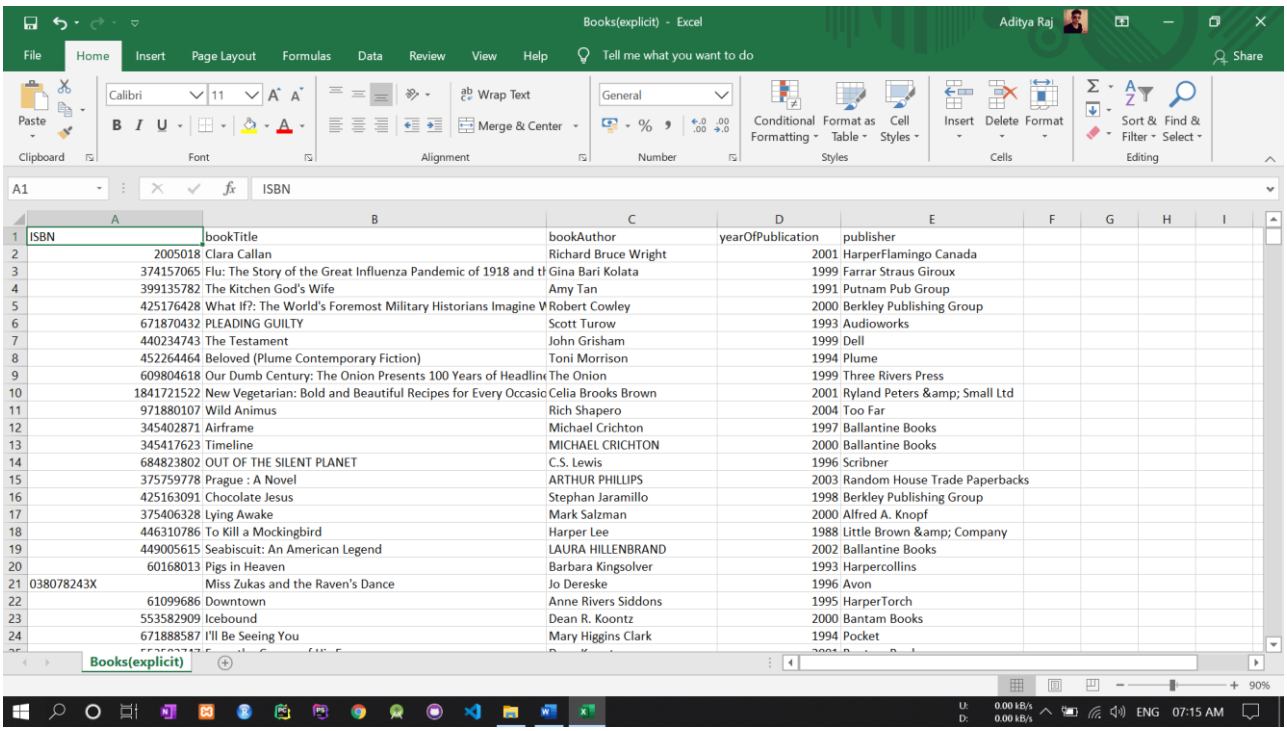
The dataset of books is collected from IIF (Institute for Informatik Freiburg). The dataset Contains 278,858 users (anonymized but with demographic information) providing 1,149,780 ratings (explicit / implicit) about 271,379 books.(http://www2.informatik.uni-freiburg.de/~cziegler/BX/)

The dataset comprises 3 tables. They are as follows:

BX-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.

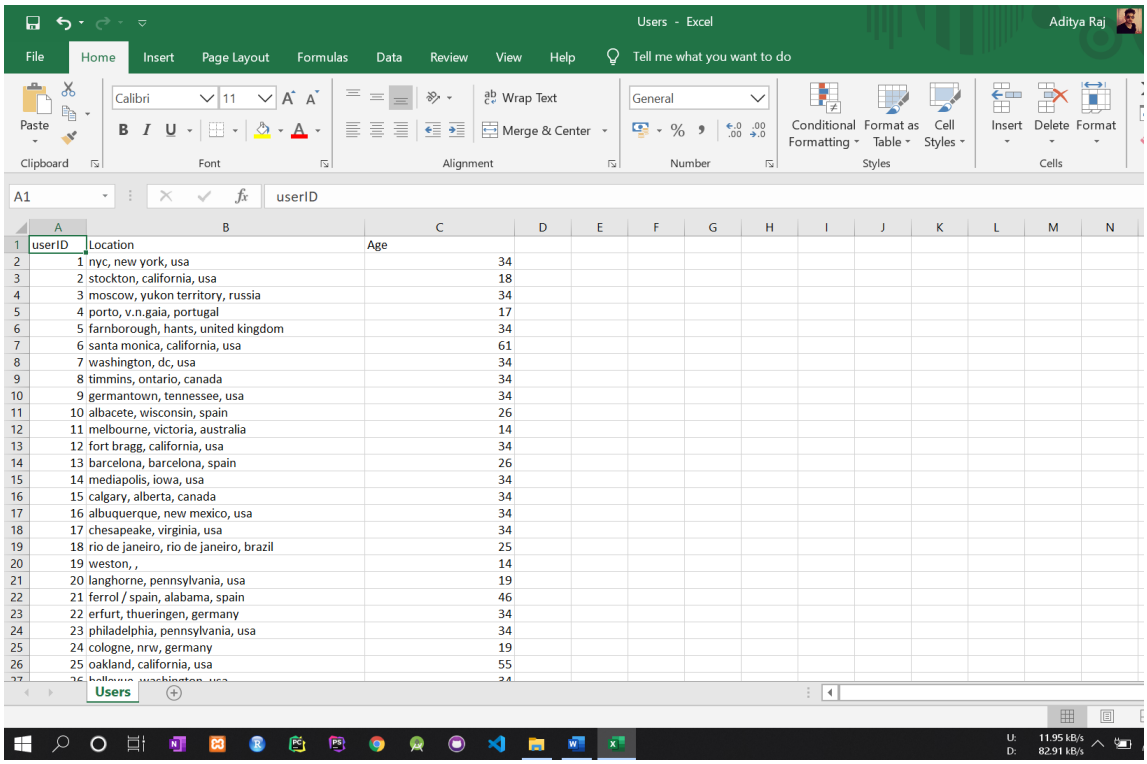
BX-Books: Books are identified by their respective ISBN. Invalid ISBNs have already been removed from the dataset. Moreover, some content-based information is given (`Book-Title`, `Book-Author`, `Year-Of-Publication`, `Publisher`).

BX-Book-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.



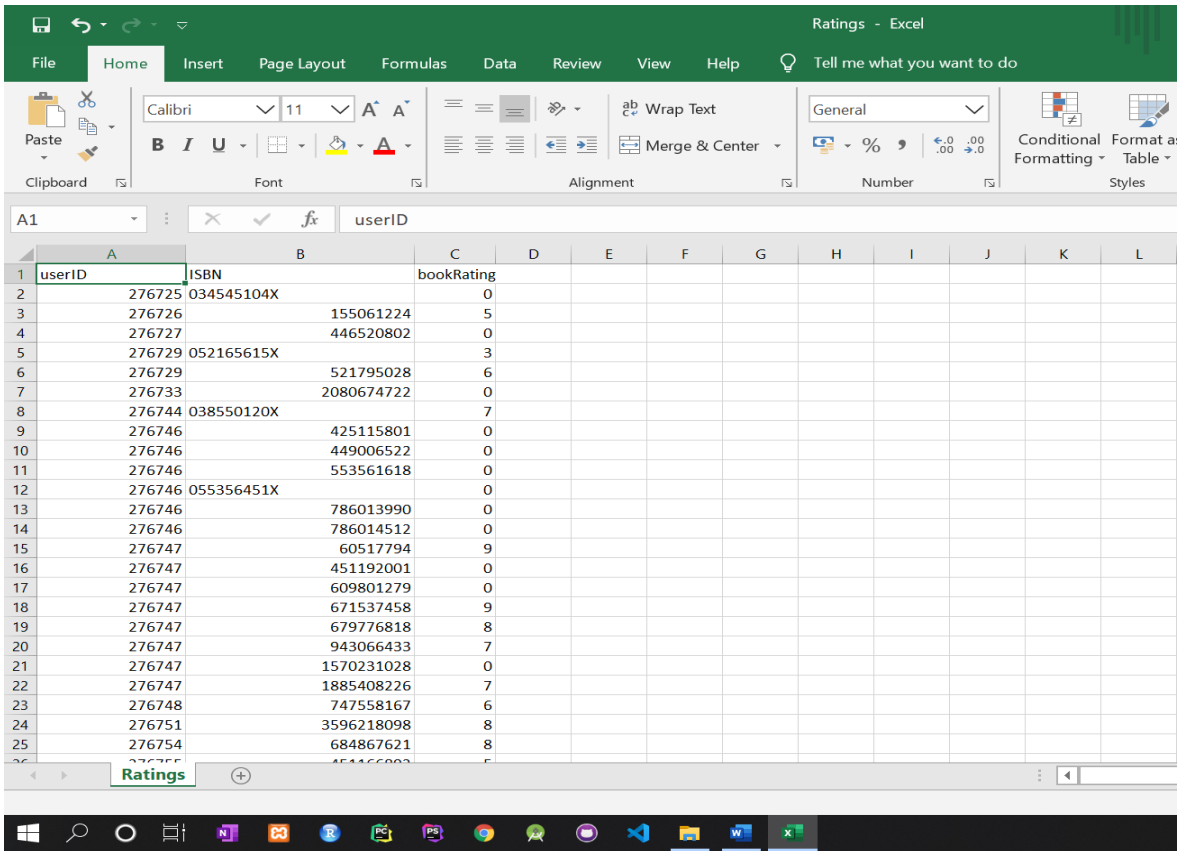
ISBN	bookTitle	bookAuthor	yearOfPublication	publisher
2005018	Clara Callan	Richard Bruce Wright	2001	HarperFlamingo Canada
374157065	Flu: The Story of the Great Influenza Pandemic of 1918 and th	Gina Bari Kolata	1999	Farrar Straus Giroux
399135782	The Kitchen God's Wife	Amy Tan	1991	Putnam Pub Group
425176428	What If?: The World's Foremost Military Historians Imagine W	Robert Cowley	2000	Berkley Publishing Group
671870432	PLEADING GUILTY	Scott Turow	1993	Audioworks
440234743	The Testament	John Grisham	1999	Dell
452264464	Beloved (Plume Contemporary Fiction)	Toni Morrison	1994	Plume
609804618	Our Dumb Century: The Onion Presents 100 Years of Headline	The Onion	1999	Three Rivers Press
1841721522	New Vegetarian: Bold and Beautiful Recipes for Every Occasio	Celia Brooks Brown	2001	Ryland Peters & Small Ltd
971880107	Wild Animus	Rich Shapero	2004	Too Far
345402871	Airframe	Michael Crichton	1997	Ballantine Books
345417623	Timeline	MICHAEL CRICHTON	2000	Ballantine Books
684823802	OUT OF THE SILENT PLANET	C.S. Lewis	1996	Scribner
375759778	Prague : A Novel	ARTHUR PHILLIPS	2003	Random House Trade Paperbacks
425163091	Chocolate Jesus	Stephan Jaramillo	1998	Berkley Publishing Group
375406328	Lying Awake	Mark Salzman	2000	Alfred A. Knopf
446310786	To Kill a Mockingbird	Harper Lee	1988	Little Brown & Company
449005615	Seabiscuit: An American Legend	LAURA HILLENBRAND	2002	Ballantine Books
60168013	Pigs in Heaven	Barbara Kingsolver	1993	Harpercollins
038078243X	Miss Zukas and the Raven's Dance	Jo Dereske	1996	Avon
61099686	Downtown	Anne Rivers Siddons	1995	HarperTorch
553582909	Icebound	Dean R. Koontz	2000	Bantam Books
671888587	I'll Be Seeing You	Mary Higgins Clark	1994	Pocket

Book dataset



userID	Location	Age
1	nyc, new york, usa	34
2	stockton, california, usa	18
3	moscow, yukon territory, russia	34
4	porto, v.n.gaia, portugal	17
5	farnborough, hants, united kingdom	34
6	santa monica, california, usa	61
7	washington, dc, usa	34
8	timmins, ontario, canada	34
9	germantown, tennessee, usa	34
10	albacete, wisconsin, spain	26
11	melbourne, victoria, australia	14
12	fort bragg, california, usa	34
13	barcelona, barcelona, spain	26
14	mediapolis, iowa, usa	34
15	calgary, alberta, canada	34
16	albuquerque, new mexico, usa	34
17	chesapeake, virginia, usa	34
18	rio de janeiro, rio de janeiro, brazil	25
19	weston, ,	14
20	langhorne, pennsylvania, usa	19
21	ferrol / spain, alabama, spain	46
22	erfurt, thueringen, germany	34
23	philadelphia, pennsylvania, usa	34
24	cologne, nrw, germany	19
25	oakland, california, usa	55

User dataset



userID	ISBN	bookRating
276725	034545104X	0
276726	155061224	5
276727	446520802	0
276729	052165615X	3
276729	521795028	6
276733	2080674722	0
276744	038550120X	7
276746	425115801	0
276746	449006522	0
276746	553561618	0
276746	055356451X	0
276746	786013990	0
276746	786014512	0
276747	60517794	9
276747	451192001	0
276747	609801279	0
276747	671537458	9
276747	679776818	8
276747	943066433	7
276747	1570231028	0
276747	1885408226	7
276748	747558167	6
276751	3596218098	8
276754	684867621	8

Rating dataset

Preliminaries:

This section summarizes related work and briefly defines the fundamental concept needed to facilitate the presentation of the proposed algorithm.

A. Related work:

Different models have been developed in order to generate book recommendation. Many approaches rely on collaborative filtering (CF) methods based on the main idea that people have similar preferences and interests, so similarities of users or books are calculated. Basically, each user gives rating scores for a list of items and these scores are used to predict the rating active user.

B. User-based collaborative filtering:

Collaborative Filtering algorithm is based on the main idea that people have similar preferences and interests. One user's behavior is compared with other user's behavior to and his/her nearest neighbors, and according to his/her neighbor's preferences or interest to predict his/her preferences or interest.

Suppose that $U = \{u_1, u_2, ..., u_m\}$ is a list of m users and $I = \{i_1, i_2, ..., i_n\}$ is a list of n items. Each user U_i gives rating scores for a list of items I_{ui}. The prediction problem is to predict the rating active user U_a will give to an item I_{ua} from the set of all items that U_a has not yet rated. The CF technique composes of 3 steps as follows: 1) users similarity calculation 2) top N nearest neighbors selection(knn) and 3) prediction.

1. SVD (Singular value decomposition)
2. k-nearest neighbour classification
3. Recommendation

- **SVD (Singular Value Decomposition)**

  SVD in the context of recommendation systems is used as a collaborative filtering (CF) algorithm. collaborative filtering is a method to predict a rating for a user item pair based on the history of ratings given by the user and given to the item. Most CF algorithms are based on user-item rating matrix where each row represents a user, each column an item. The entries of this matrix are ratings given by users to items.

  SVD is a matrix factorization technique that is usually used to reduce the number of features of a data set by reducing space dimensions from N to K where K < N. For the purpose of the

recommendation systems however, we are only interested in the matrix factorization part keeping same dimensionality. The matrix factorization is done on the user-item ratings matrix.

One way to handle the scalability and sparsity issue created by CF is to leverage a **latent factor model** to capture the similarity between users and items. Essentially, we want to turn the recommendation problem into an optimization problem. We can view it as how good we are in predicting the rating for items given a user.

One common metric is Root Mean Square Error (RMSE). The lower the RMSE, the better the performance. Since we do not know the rating for the unseen items, we will temporarily ignore them. Namely, we are only minimizing RMSE on the known entries in the utility matrix. To achieve minimal RMSE, Singular Value Decomposition (SVD) is adopted as shown in the below formula.

$$\underset{m \times n}{\hat{X}} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix} \approx \underset{m \times r}{U} \begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix} \underset{r \times r}{S} \begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix} \underset{r \times n}{V^{\top}} \begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}$$

Singular Matrix

X denotes the utility matrix, and U is a left singular matrix, representing the relationship between users and latent factors. S is a diagonal matrix describing the strength of each latent factor, while V transpose is a right singular matrix, indicating the similarity between items and latent factors. SVD decreases the dimension of the utility matrix by extracting its latent factors.

Essentially, we map each user and each item into a latent space with dimension r. Therefore, it helps us better understand the relationship between users and items as they become directly comparable. The below figure illustrates this idea.

- **k-nearest neighbour classification**

Our dataframe of book features is an extremely sparse matrix with a shape of 86580 rows x 1180 columns. We definitely don't want to feed the entire data with mostly 0s in float32 datatype to KNN. For more efficient calculation and less memory footprint, we need to transform the values of the dataframe into a **scipy sparse matrix**.

```
def __init__(self, n_neighbors=5):
    # calling super class __init__ method
    super().__init__()
    # assigning k  value = 5
    self.n_neighbors = n_neighbors
    # removing nan value
    self.ratings_mat = self.ratings_explicit.pivot(
        index="ISBN", columns="userID", values="bookRating").fillna(0)
    print(self.ratings_mat)
    '''
    Implementing kNN
    In numerical analysis and scientific computing, a sparse matrix or sparse array is a matrix in which
    most of the elements are zero.
    We convert our table to a 2D matrix, and fill the missing values with zeros
    (since we will calculate distances between rating vectors). We then transform the values(ratings)
    of the matrix dataframe into a scipy sparse matrix for more efficient calculations.
    Finding the Nearest Neighbors We use unsupervised algorithms with sklearn.neighbors.
    The algorithm we use to compute the nearest neighbors is "brute", and we specify "metric=cosine"
    algorithm will calculate the cosine similarity between rating vectors. Finally, we fit the model.
    '''
    self.uti_mat = csr_matrix(self.ratings_mat.values)
    print(self.uti_mat)
    print("sparse : ", self.uti_mat)
    # KNN Model Fitting
    # using cosine similarity
    '''Mathematically, it measures
    the cosine of the angle between two vectors projected in a multi-dimensional space
    Cosine similarity is a metric used to determine how
     similar the documents are irrespective of their size.'''
    self.model_knn = NearestNeighbors(metric='cosine', algorithm='brute')
    self.model_knn.fit(self.uti_mat)
```

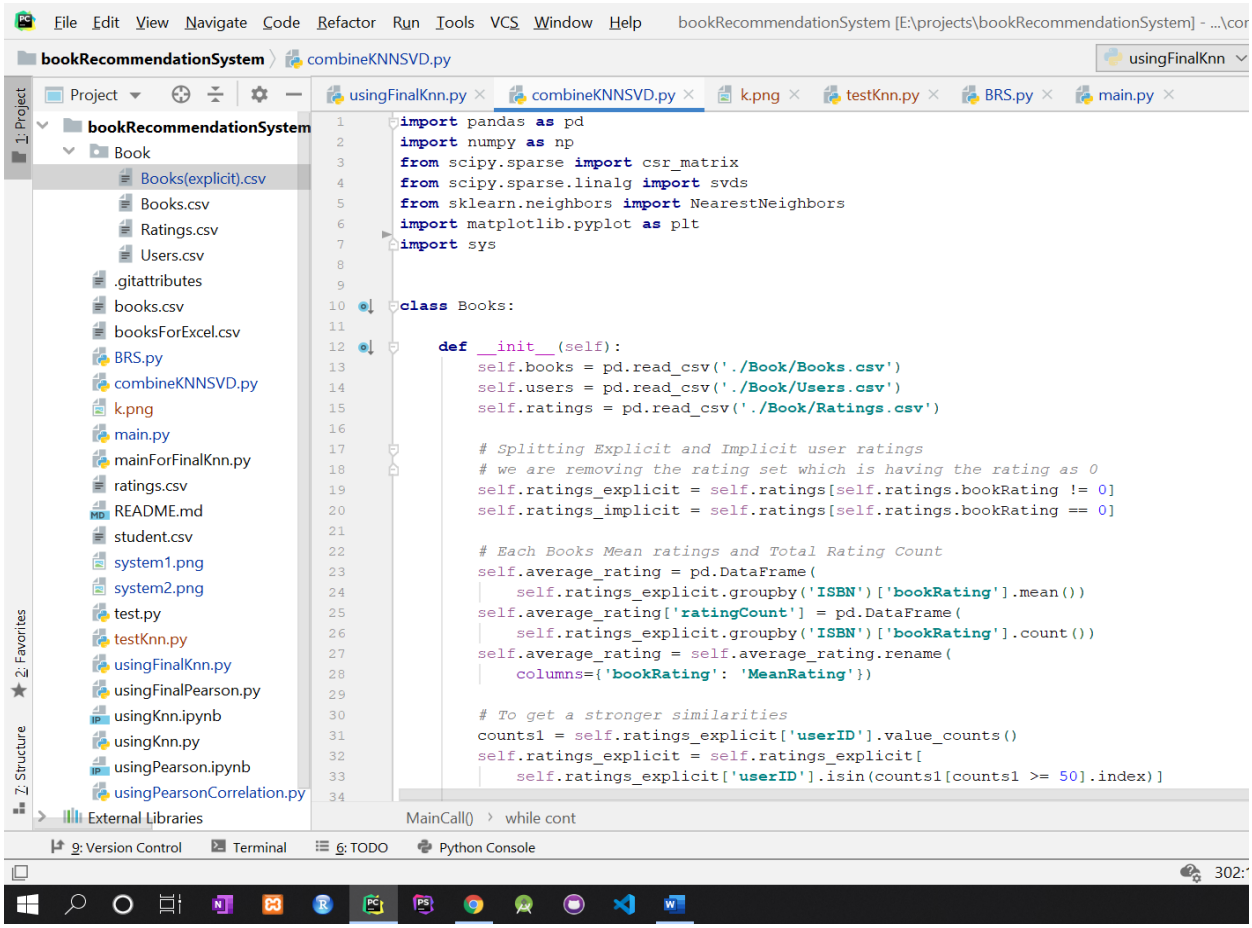
Now our training data has a very high dimensionality. KNN's performance will suffer from curse of dimensionality if it uses "Euclidean distance" in its objective function. Euclidean distance is unhelpful in high dimensions because all vectors are almost equidistant to the search query vector (target book's features).

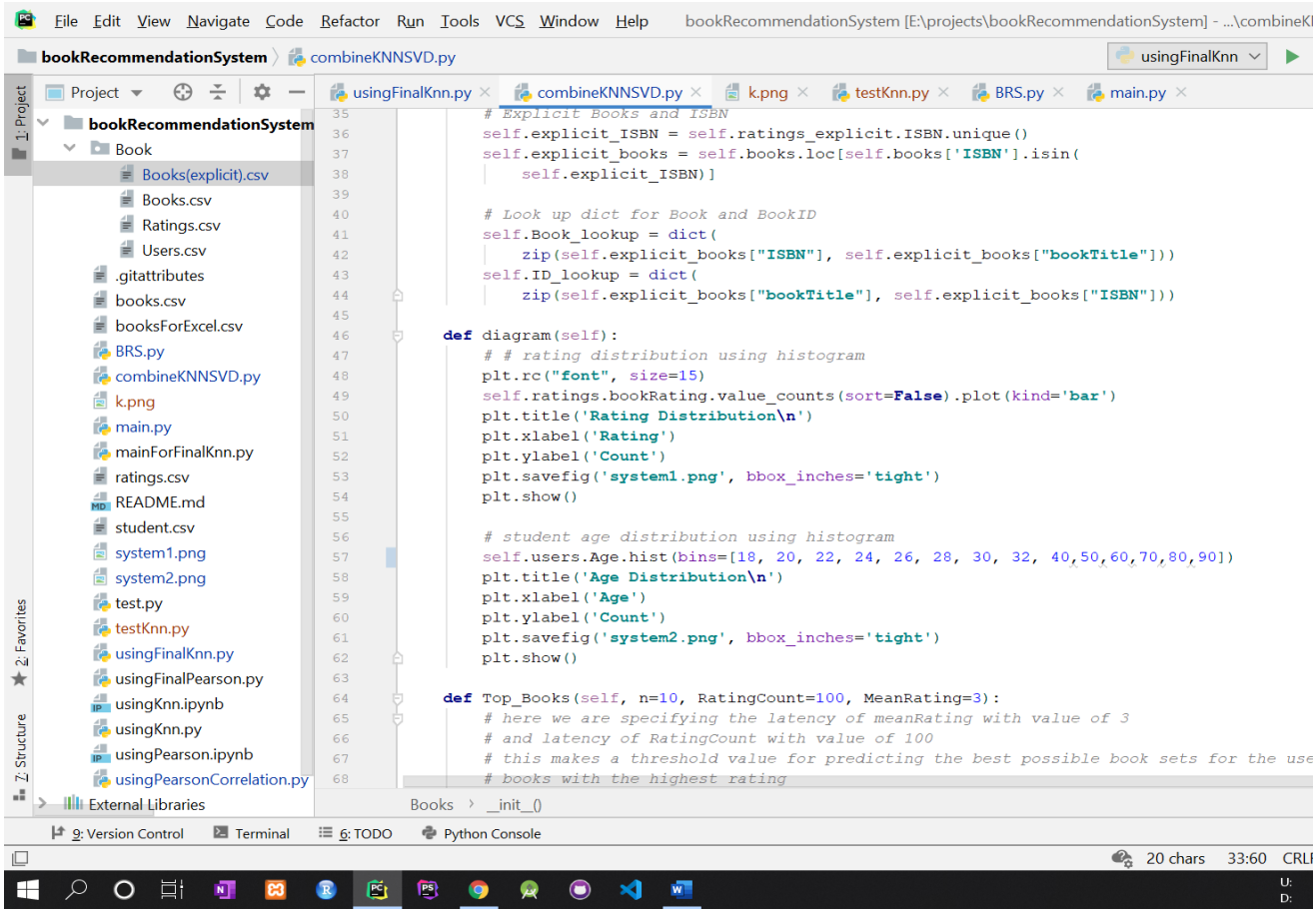
The k-nearest neighbors (KNN) algorithm doesn't make any assumptions on the underlying data distribution, but it relies on item feature similarity. When a KNN makes a prediction about a book, it will calculate the "distance" (distance metrics will be discussed later) between the target book and every other book in its database. It then ranks its distances and returns the top k nearest neighbor book as the most similar book recommendations.

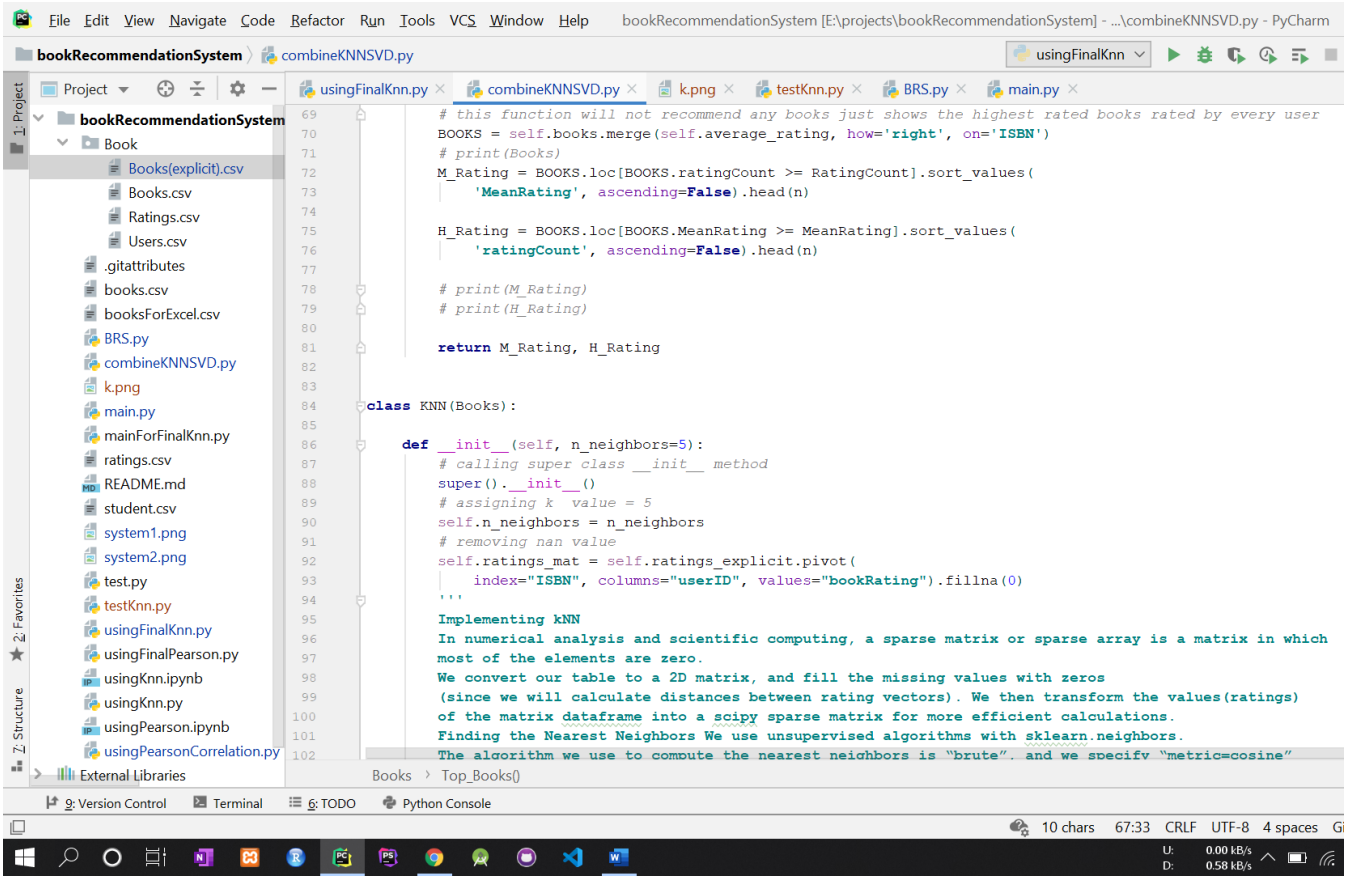
- **Making Recommendations**

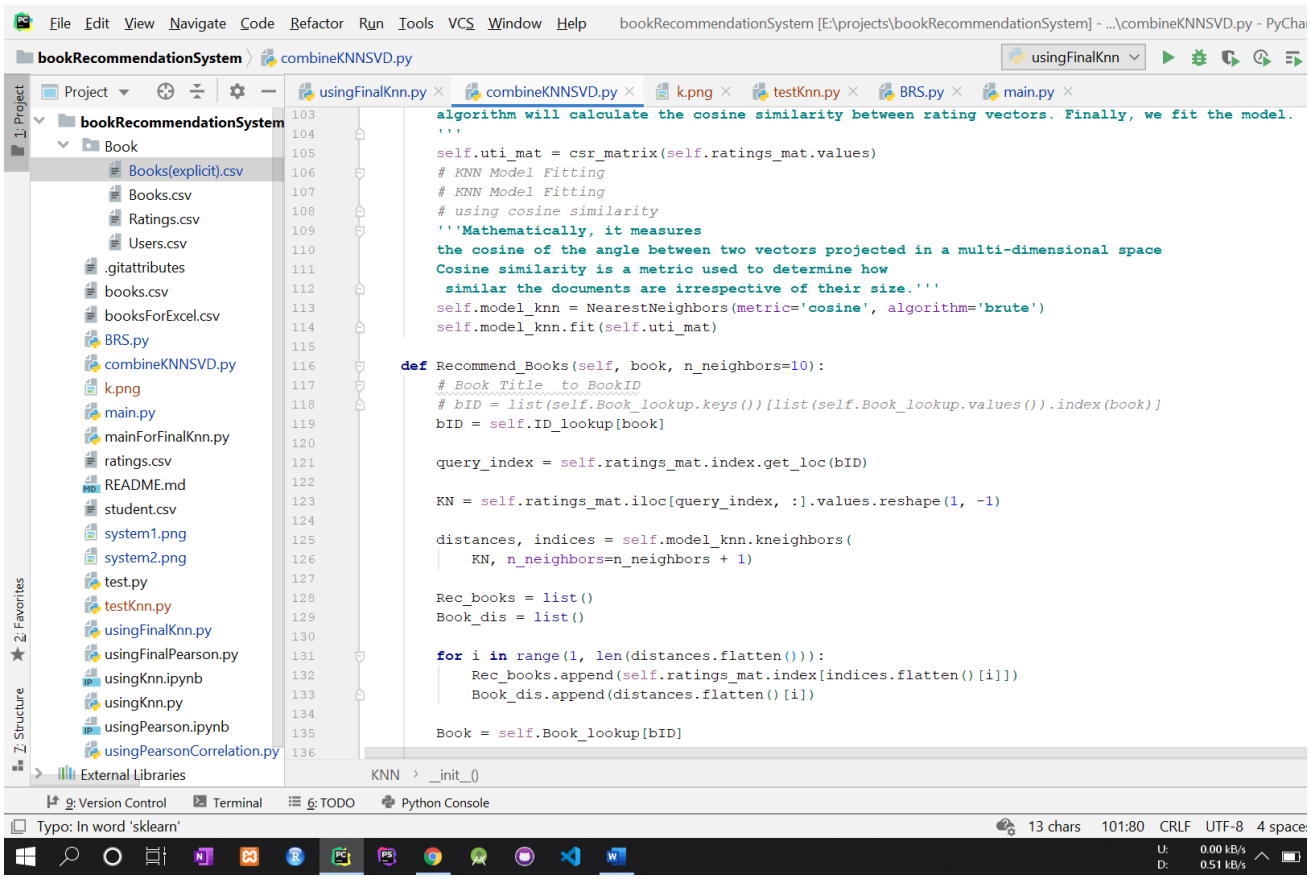
We've already fit the pre-processed dataset in our KNN model. Now we just need to take a book as input and recommend books based on the inference derived from the KNN.

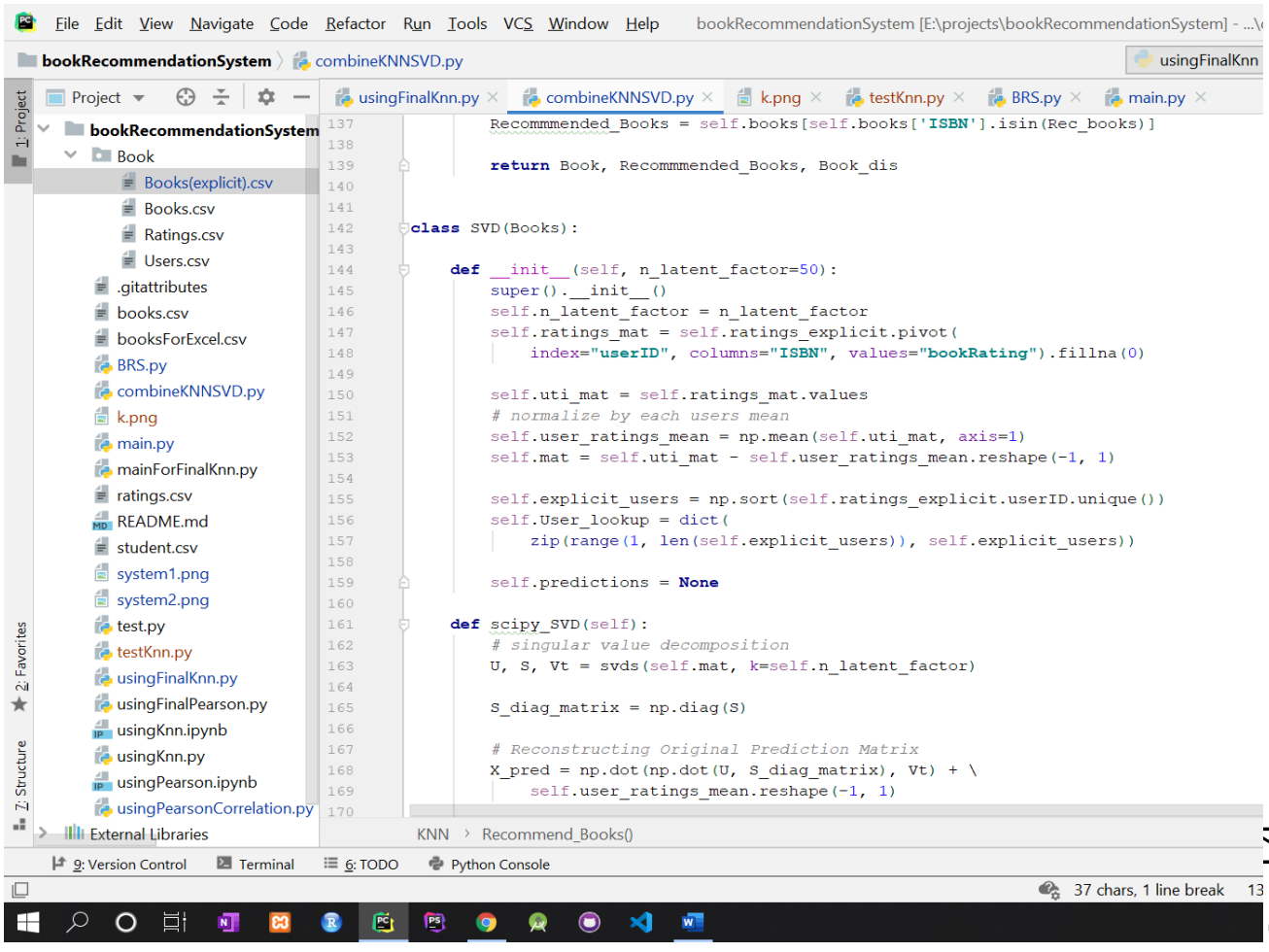
Code Screenshots:

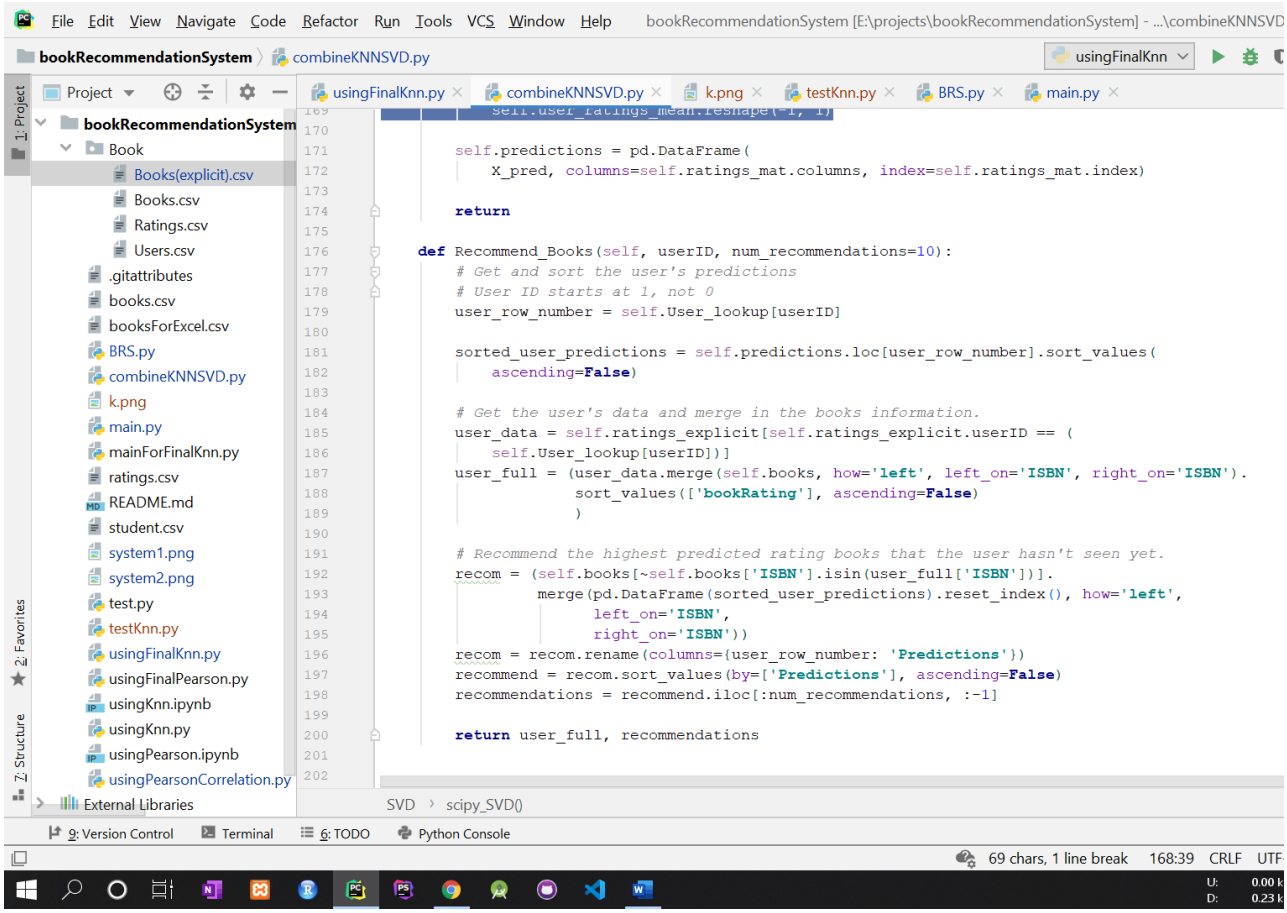



File Edit View Navigate Code Refactor Run Tools VCS Window Help
bookRecommendationSystem [E:\projects\bookRecommendationSystem] - ...\combineKNNSVD.py - PyCharm
bookRecommendationSystem
combineKNNSVD.py
usingFinalKnn
usingFinalKnn.py
combineKNNSVD.py
k.png
testKnn.py
BRS.py
main.py
Project
bookRecommendationSystem
Book
Books(explicit).csv
Books.csv
Ratings.csv
Users.csv
.gitattributes
books.csv
booksForExcel.csv
BRS.py
combineKNNSVD.py
k.png
main.py
mainForFinalKnn.py
ratings.csv
README.md
student.csv
system1.png
system2.png
test.py
testKnn.py
usingFinalKnn.py
usingFinalPearson.py
usingKnn.ipynb
usingKnn.py
usingPearson.ipynb
usingPearsonCorrelation.py
External Libraries
# Explicit Books and ISBN
self.explicit_ISBN = self.ratings_explicit.ISBN.unique()
self.explicit_books = self.books.loc[self.books['ISBN'].isin(
self.explicit_ISBN)]
# Look up dict for Book and BookID
self.Book_lookup = dict(
zip(self.explicit_books["ISBN"], self.explicit_books["bookTitle"]))
self.ID_lookup = dict(
zip(self.explicit_books["bookTitle"], self.explicit_books["ISBN"]))
def diagram(self):
# # rating distribution using histogram
plt.rc("font", size=15)
self.ratings.bookRating.value_counts(sort=False).plot(kind='bar')
plt.title('Rating Distribution\n')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.savefig('system1.png', bbox_inches='tight')
plt.show()
# student age distribution using histogram
self.users.Age.hist(bins=[18, 20, 22, 24, 26, 28, 30, 32, 40,50,60,70,80,90])
plt.title('Age Distribution\n')
plt.xlabel('Age')
plt.ylabel('Count')
plt.savefig('system2.png', bbox_inches='tight')
plt.show()
def Top_Books(self, n=10, RatingCount=100, MeanRating=3):
# here we are specifying the latency of meanRating with value of 3
# and latency of RatingCount with value of 100
# this makes a threshold value for predicting the best possible book sets for the use
# books with the highest rating
Books › __init__()
9: Version Control
Terminal
6: TODO
Python Console
20 chars
33:60
CRLF

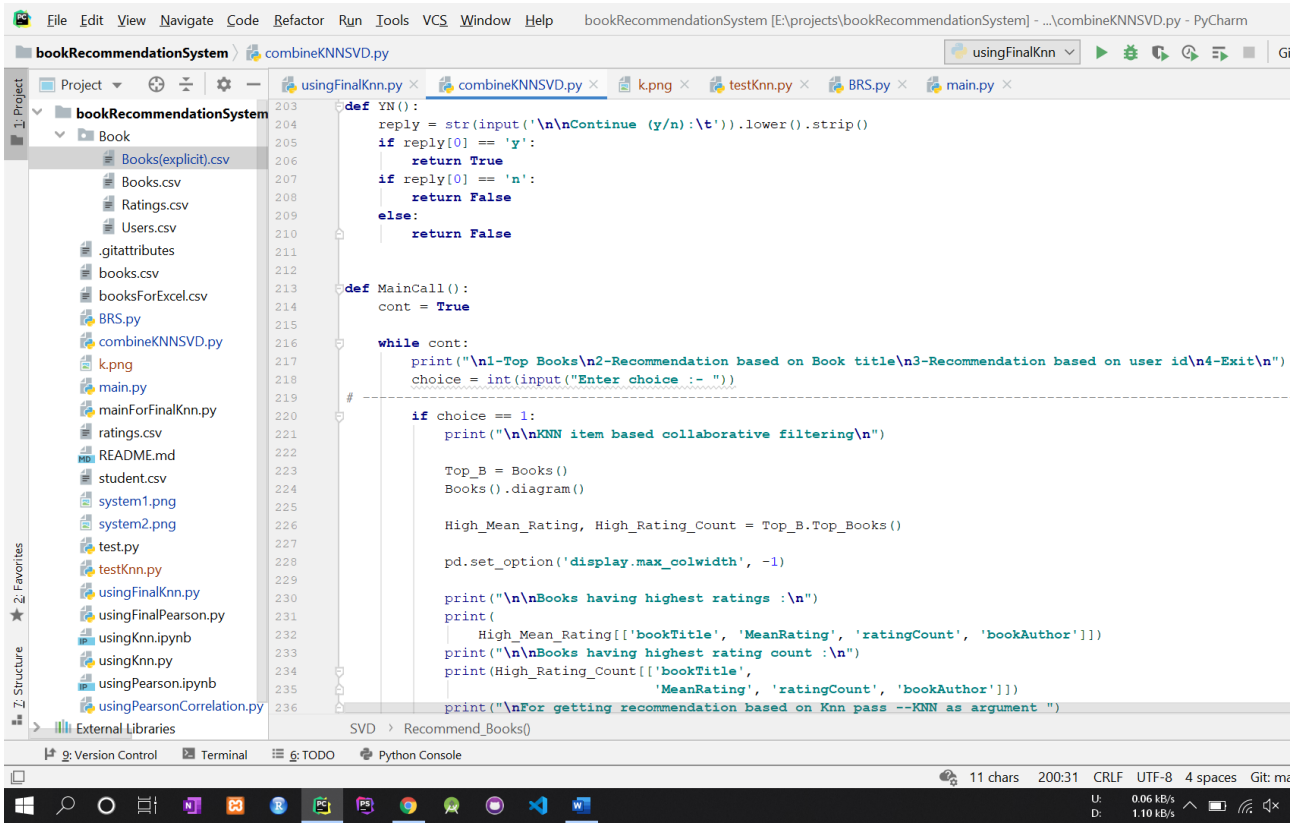

File Edit View Navigate Code Refactor Run Tools VCS Window Help
bookRecommendationSystem [E:\projects\bookRecommendationSystem] - ...\combineKNNSVD.py - PyCharm
bookRecommendationSystem
combineKNNSVD.py
usingFinalKnn
usingFinalKnn.py
combineKNNSVD.py
k.png
testKnn.py
BRS.py
main.py
Project
bookRecommendationSystem
Book
Books(explicit).csv
Books.csv
Ratings.csv
Users.csv
.gitattributes
books.csv
booksForExcel.csv
BRS.py
combineKNNSVD.py
k.png
main.py
mainForFinalKnn.py
ratings.csv
README.md
student.csv
system1.png
system2.png
test.py
testKnn.py
usingFinalKnn.py
usingFinalPearson.py
usingKnn.ipynb
usingKnn.py
usingPearson.ipynb
usingPearsonCorrelation.py
External Libraries
# this function will not recommend any books just shows the highest rated books rated by every user
BOOKS = self.books.merge(self.average_rating, how='right', on='ISBN')
# print(Books)
M_Rating = BOOKS.loc[BOOKS.ratingCount >= RatingCount].sort_values(
'MeanRating', ascending=False).head(n)
H_Rating = BOOKS.loc[BOOKS.MeanRating >= MeanRating].sort_values(
'ratingCount', ascending=False).head(n)
print(M_Rating)
print(H_Rating)
return M_Rating, H_Rating
class KNN(Books):
def __init__(self, n_neighbors=5):
# calling super class __init__ method
super().__init__()
# assigning k value = 5
self.n_neighbors = n_neighbors
# removing nan value
self.ratings_mat = self.ratings_explicit.pivot(
index="ISBN", columns="userID", values="bookRating").fillna(0)
'''
Implementing kNN
In numerical analysis and scientific computing, a sparse matrix or sparse array is a matrix in which
most of the elements are zero.
We convert our table to a 2D matrix, and fill the missing values with zeros
(since we will calculate distances between rating vectors). We then transform the values(ratings)
of the matrix dataframe into a scipy sparse matrix for more efficient calculations.
Finding the Nearest Neighbors We use unsupervised algorithms with sklearn.neighbors.
The algorithm we use to compute the nearest neighbors is "brute", and we specify "metric=cosine"
Books › Top_Books()
9: Version Control
Terminal
6: TODO
Python Console
10 chars
67:33
CRLF
UTF-8
4 spaces


algorithm will calculate the cosine similarity between rating vectors. Finally, we fit the model.
'''
self.uti_mat = csr_matrix(self.ratings_mat.values)
# KNN Model Fitting
# KNN Model Fitting
# using cosine similarity
'''Mathematically, it measures
the cosine of the angle between two vectors projected in a multi-dimensional space
Cosine similarity is a metric used to determine how
similar the documents are irrespective of their size.'''
self.model_knn = NearestNeighbors(metric='cosine', algorithm='brute')
self.model_knn.fit(self.uti_mat)
def Recommend_Books(self, book, n_neighbors=10):
# Book_Title__to_BookID
# bID = list(self.Book_lookup.keys())[list(self.Book_lookup.values()).index(book)]
bID = self.ID_lookup[book]
query_index = self.ratings_mat.index.get_loc(bID)
KN = self.ratings_mat.iloc[query_index, :].values.reshape(1, -1)
distances, indices = self.model_knn.kneighbors(
KN, n_neighbors=n_neighbors + 1)
Rec_books = list()
Book_dis = list()
for i in range(1, len(distances.flatten())):
Rec_books.append(self.ratings_mat.index[indices.flatten()[i]])
Book_dis.append(distances.flatten()[i])
Book = self.Book_lookup[bID]

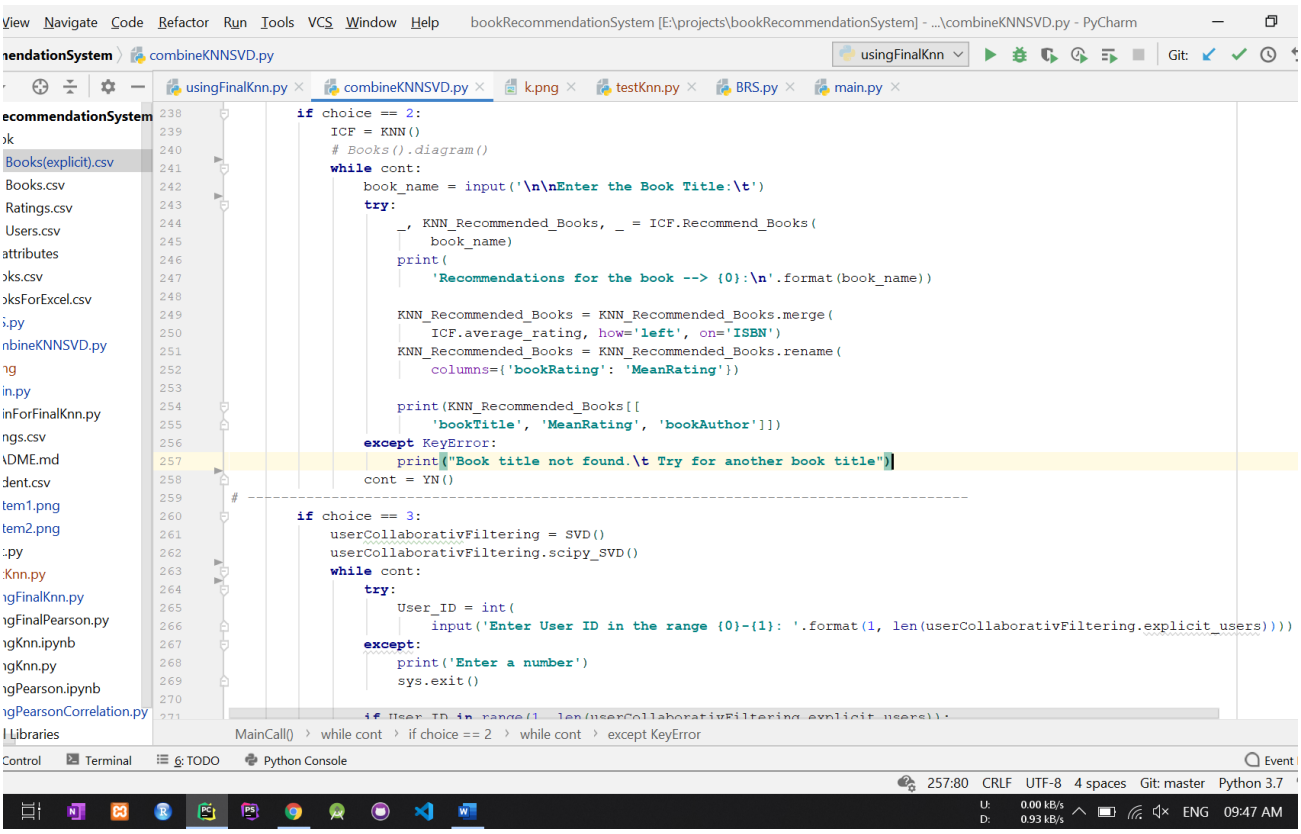

Recommended_Books = self.books[self.books['ISBN'].isin(Rec_books)]
return Book, Recommended_Books, Book_dis
class SVD(Books):
def __init__(self, n_latent_factor=50):
super().__init__()
self.n_latent_factor = n_latent_factor
self.ratings_mat = self.ratings_explicit.pivot(
index="userID", columns="ISBN", values="bookRating").fillna(0)
self.uti_mat = self.ratings_mat.values
# normalize by each users mean
self.user_ratings_mean = np.mean(self.uti_mat, axis=1)
self.mat = self.uti_mat - self.user_ratings_mean.reshape(-1, 1)
self.explicit_users = np.sort(self.ratings_explicit.userID.unique())
self.User_lookup = dict(
zip(range(1, len(self.explicit_users)), self.explicit_users))
self.predictions = None
def scipy_SVD(self):
# singular value decomposition
U, S, Vt = svds(self.mat, k=self.n_latent_factor)
S_diag_matrix = np.diag(S)
# Reconstructing Original Prediction Matrix
X_pred = np.dot(np.dot(U, S_diag_matrix), Vt) + \
self.user_ratings_mean.reshape(-1, 1)



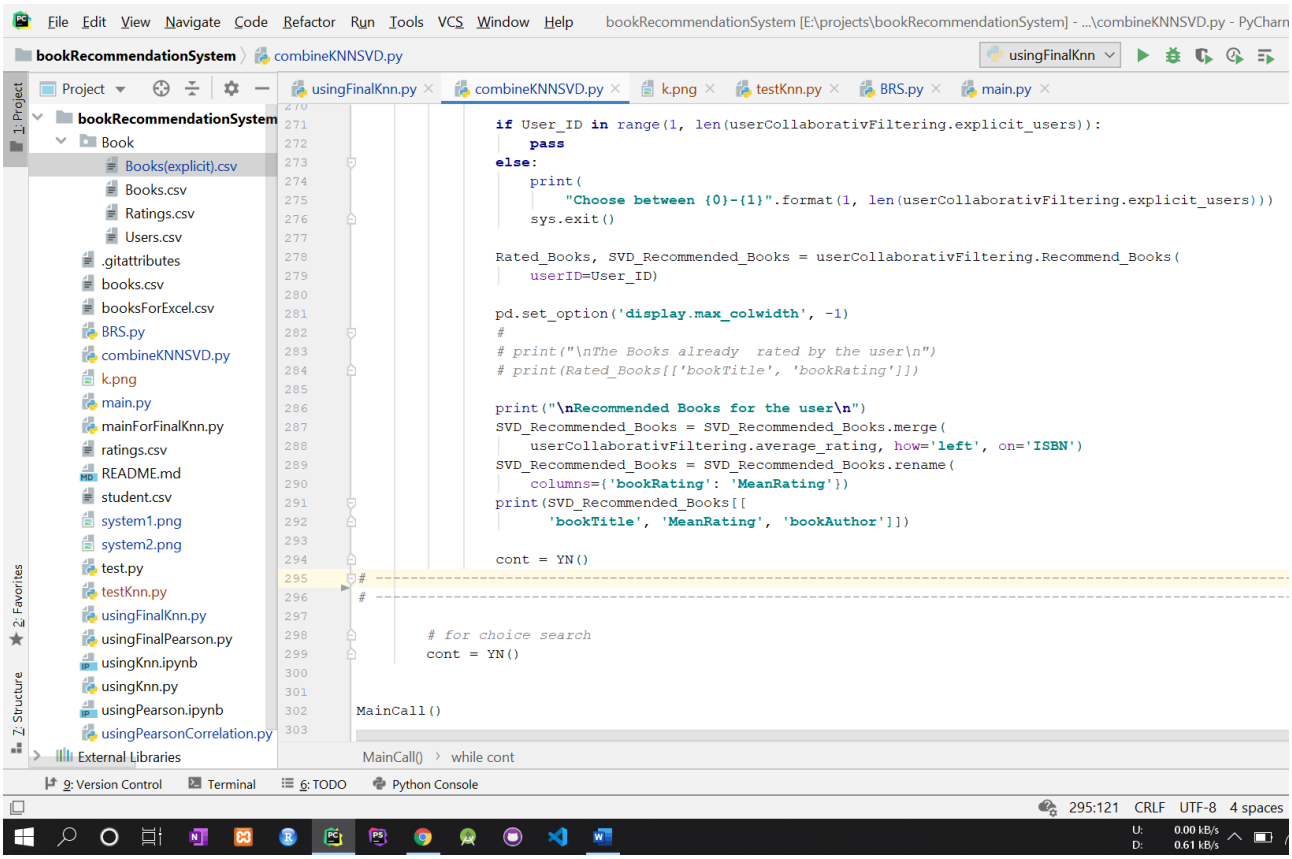
```
        self.predictions = pd.DataFrame(
            X_pred, columns=self.ratings_mat.columns, index=self.ratings_mat.index)

        return

    def Recommend_Books(self, userID, num_recommendations=10):
        # Get and sort the user's predictions
        # User ID starts at 1, not 0
        user_row_number = self.User_lookup[userID]

        sorted_user_predictions = self.predictions.loc[user_row_number].sort_values(
            ascending=False)

        # Get the user's data and merge in the books information.
        user_data = self.ratings_explicit[self.ratings_explicit.userID == (
            self.User_lookup[userID])]
        user_full = (user_data.merge(self.books, how='left', left_on='ISBN', right_on='ISBN').
                     sort_values(['bookRating'], ascending=False)
                     )

        # Recommend the highest predicted rating books that the user hasn't seen yet.
        recom = (self.books[~self.books['ISBN'].isin(user_full['ISBN'])].
                 merge(pd.DataFrame(sorted_user_predictions).reset_index(), how='left',
                       left_on='ISBN',
                       right_on='ISBN'))
        recom = recom.rename(columns={user_row_number: 'Predictions'})
        recommend = recom.sort_values(by=['Predictions'], ascending=False)
        recommendations = recommend.iloc[:num_recommendations, :-1]

        return user_full, recommendations
```



```
def YN():
    reply = str(input('\n\nContinue (y/n):\t')).lower().strip()
    if reply[0] == 'y':
        return True
    if reply[0] == 'n':
        return False
    else:
        return False


def MainCall():
    cont = True

    while cont:
        print("\n1-Top Books\n2-Recommendation based on Book title\n3-Recommendation based on user id\n4-Exit\n")
        choice = int(input("Enter choice :- "))
# ------------------------------------------------------------------------------------------------------------------------
        if choice == 1:
            print("\n\nKNN item based collaborative filtering\n")

            Top_B = Books()
            Books().diagram()

            High_Mean_Rating, High_Rating_Count = Top_B.Top_Books()

            pd.set_option('display.max_colwidth', -1)

            print("\n\nBooks having highest ratings :\n")
            print(
                High_Mean_Rating[['bookTitle', 'MeanRating', 'ratingCount', 'bookAuthor']])
            print("\n\nBooks having highest rating count :\n")
            print(High_Rating_Count[['bookTitle',
                                     'MeanRating', 'ratingCount', 'bookAuthor']])
            print("\nFor getting recommendation based on Knn pass --KNN as argument ")
```



```
if choice == 2:
    ICF = KNN()
    # Books().diagram()
    while cont:
        book_name = input('\n\nEnter the Book Title:\t')
        try:
            _, KNN_Recommended_Books, _ = ICF.Recommend_Books(
                book_name)
            print(
                'Recommendations for the book --> {0}:\n'.format(book_name))

            KNN_Recommended_Books = KNN_Recommended_Books.merge(
                ICF.average_rating, how='left', on='ISBN')
            KNN_Recommended_Books = KNN_Recommended_Books.rename(
                columns={'bookRating': 'MeanRating'})

            print(KNN_Recommended_Books[[
                'bookTitle', 'MeanRating', 'bookAuthor']])
        except KeyError:
            print("Book title not found.\t Try for another book title")
        cont = YN()
# ------------------------------------------------------------------------
if choice == 3:
    userCollaborativFiltering = SVD()
    userCollaborativFiltering.scipy_SVD()
    while cont:
        try:
            User_ID = int(
                input('Enter User ID in the range {0}-{1}: '.format(1, len(userCollaborativFiltering.explicit_users))))
        except:
            print('Enter a number')
            sys.exit()
```



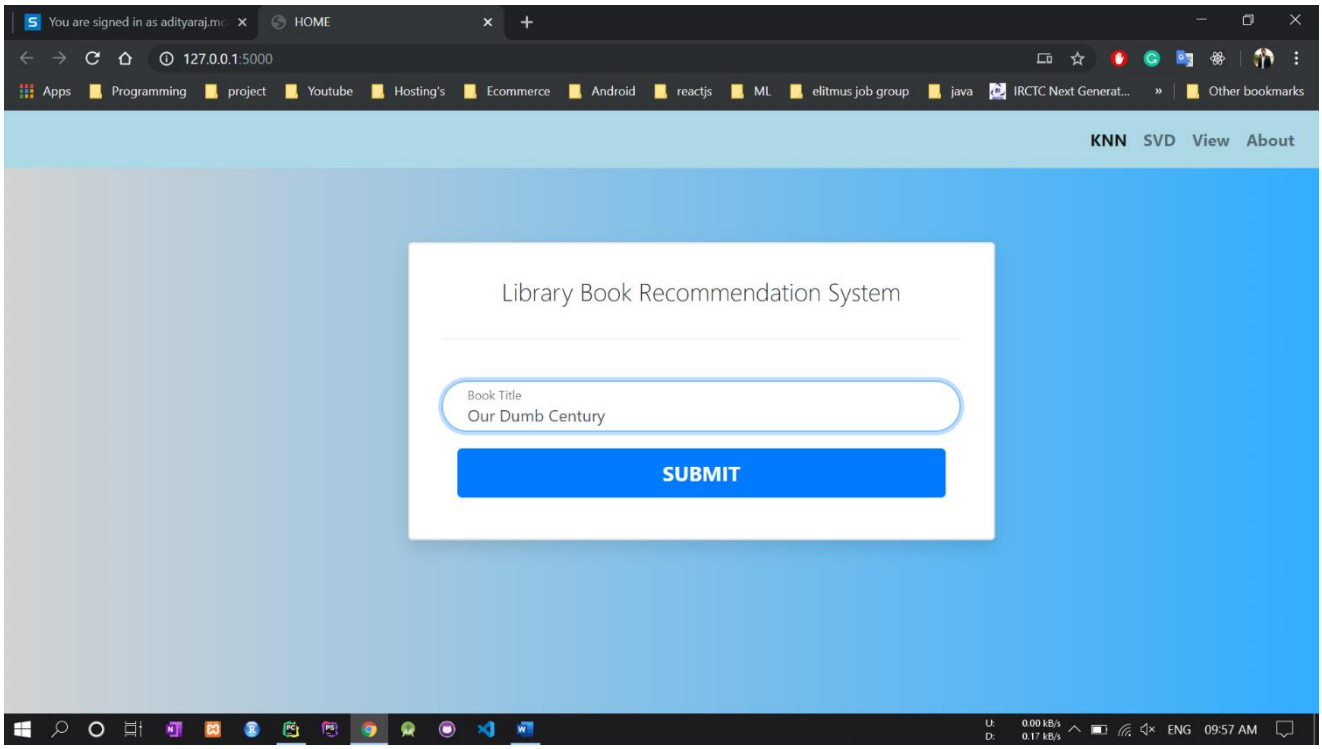
```
            if User_ID in range(1, len(userCollaborativFiltering.explicit_users)):
                pass
            else:
                print(
                    "Choose between {0}-{1}".format(1, len(userCollaborativFiltering.explicit_users)))
                sys.exit()

            Rated_Books, SVD_Recommended_Books = userCollaborativFiltering.Recommend_Books(
                userID=User_ID)

            pd.set_option('display.max_colwidth', -1)
            #
            # print("\nThe Books already  rated by the user\n")
            # print(Rated_Books[['bookTitle', 'bookRating']])

            print("\nRecommended Books for the user\n")
            SVD_Recommended_Books = SVD_Recommended_Books.merge(
                userCollaborativFiltering.average_rating, how='left', on='ISBN')
            SVD_Recommended_Books = SVD_Recommended_Books.rename(
                columns={'bookRating': 'MeanRating'})
            print(SVD_Recommended_Books[[
                'bookTitle', 'MeanRating', 'bookAuthor']])

            cont = YN()
# ------------------------------------------------------------------------
# ------------------------------------------------------------------------

        # for choice search
        cont = YN()


MainCall()
```

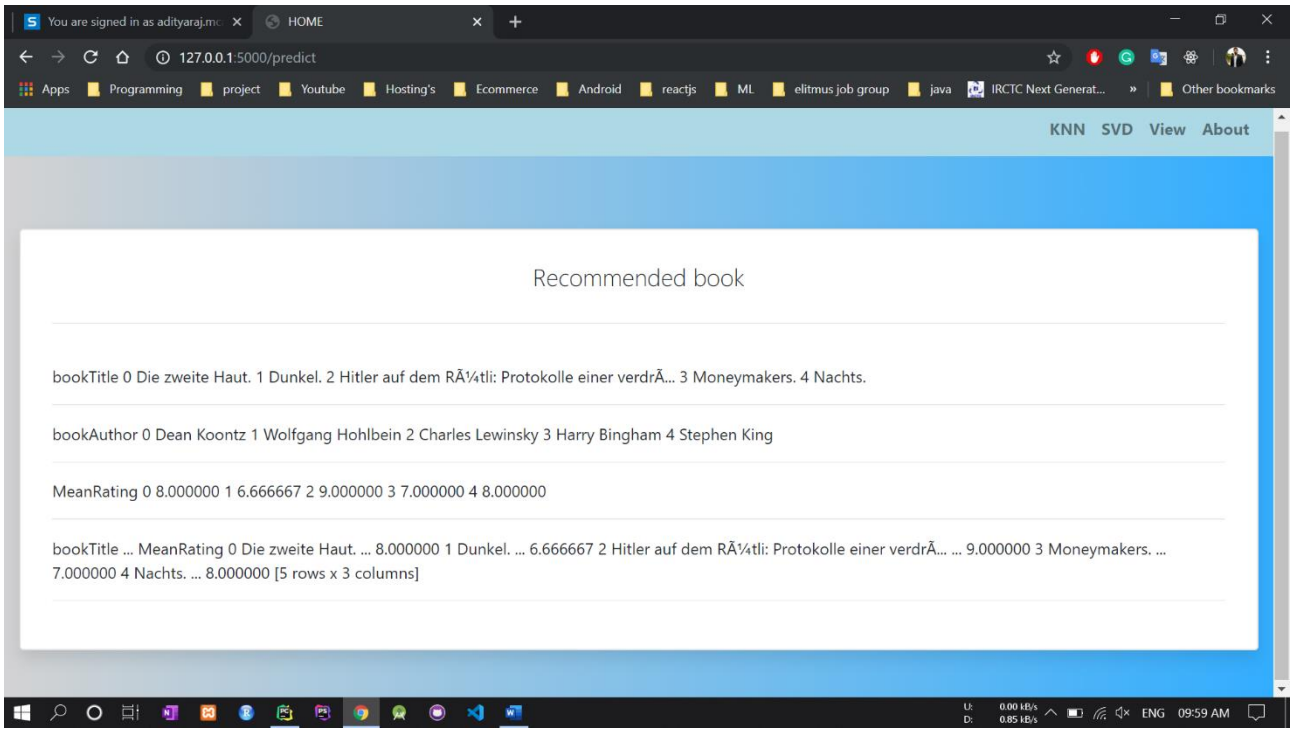
```
Python 3.7.5 (default, Oct 31 2019, 15:18:51) [MSC v.1916 64 bit (AMD64)]

Python 3.7.5 (default, Oct 31 2019, 15:18:51) [MSC v.1916 64 bit (AMD64)] on win32
In[2]: runfile('E:/projects/bookRecommendationSystem/combineKNNSVD.py',
 wdir='E:/projects/bookRecommendationSystem')

1-Top Books
2-Recommendation based on Book title
3-Recommendation based on user id
4-Exit

Enter choice :-
>?
```

```
Enter choice :- >? 2


Enter the Book Title:   >? The Testament
Recommendations for the book --> The Testament:

                                          bookTitle  ...          bookAuthor
0  Greek Myths: Gods, Heroes and Monsters : Their...  ...       Ellen Switzer
1                      What Did Mommy Do Before You?  ...         Abby Levine
2                The Heart Reader Of Franklin High  ...           Anonymous
3                            How Santa Got His Job  ...     Stephen Krensky
4  Officer Buckle and Gloria (Caldecott Medal Boo...  ...      Peggy Rathmann
5  Wood: Creation of the American Republic 1776-1...  ...            G S WOOD
6  Culture and Anarchy (Rethinking the Western Tr...  ...      Matthew Arnold
7           Random House Basic Dictionary of French  ...  Francesca Langbaum
8                           Immortal Love: Stories  ...         Sally Laird
9  Diablo II Official Strategy Guide (Official Gu...  ...         Bart Farkas

[10 rows x 3 columns]


Continue (y/n):
>?
```

```
1-Top Books
2-Recommendation based on Book title
3-Recommendation based on user id
4-Exit

Enter choice :- >? 3
Enter User ID in the range 1-1180: >? 121

Recommended Books for the user

                                                bookTitle  ...       bookAuthor
0  The Lovely Bones: A Novel                                ...  Alice Sebold
1  The Da Vinci Code                                        ...  Dan Brown
2  The Secret Life of Bees                                  ...  Sue Monk Kidd
3  Where the Heart Is (Oprah's Book Club (Paperback))       ...  Billie Letts
4  Divine Secrets of the Ya-Ya Sisterhood: A Novel          ...  Rebecca Wells
5  Snow Falling on Cedars                                   ...  David Guterson
6  Wicked: The Life and Times of the Wicked Witch of the West  ...  Gregory Maguire
7  To Kill a Mockingbird                                    ...  Harper Lee
8  Girl with a Pearl Earring                                ...  Tracy Chevalier
9  STONES FROM THE RIVER                                    ...  Ursula Hegi

[10 rows x 3 columns]
```

User Interface:




You are signed in as adityaraj.mc
HOME
127.0.0.1:5000/predict
Apps
Programming
project
Youtube
Hosting's
Ecommerce
Android
reactjs
ML
elitmus job group
java
IRCTC Next Generat...
Other bookmarks
KNN
SVD
View
About
Recommended book
bookTitle 0 Die zweite Haut. 1 Dunkel. 2 Hitler auf dem RÃ¼tli: Protokolle einer verdrÃ... 3 Moneymakers. 4 Nachts.
bookAuthor 0 Dean Koontz 1 Wolfgang Hohlbein 2 Charles Lewinsky 3 Harry Bingham 4 Stephen King
MeanRating 0 8.000000 1 6.666667 2 9.000000 3 7.000000 4 8.000000
bookTitle ... MeanRating 0 Die zweite Haut. ... 8.000000 1 Dunkel. ... 6.666667 2 Hitler auf dem RÃ¼tli: Protokolle einer verdrÃ... ... 9.000000 3 Moneymakers. ... 7.000000 4 Nachts. ... 8.000000 [5 rows x 3 columns]
ENG
09:59 AM

Conclusion

This system aims to provide personalized recommendation of books to the students. This system considers big data of books. The system makes use of collaborative filtering algorithm and knn classification algorithm to provides the user with recommendation list. The system tries to predict the ranking by considering the item's similarity as well as user's similarity so that a user can get recommendations of new books.

REFERENCES

[1] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," Advances in Artificial Intelligence archive, 2009

[2] L. Xin, E. Haihong, S. Junde, S. Meina, and T. Junjie, "Collaborative book recommendation based on readers' borrowing records," In Proceeding of the 2013 International Conference on Advanced Cloud ang Big Data, Dec. 2013, pp. 159-163.

[3] Y. Koren, R. Bell, and C. Volinsky, "Matrix Factoirzation techniques for recommendaer systems," Computer, vol. 42, Aug. 2009, pp. 42-49

[4] H. Polat and W. Du, "SVD-based collaborative filtering with privacy," ACM Symposium on Allied Computing, Mar. 2005, pp. 142-146