Introduction to Big Data

The Islanders

Avlonitis Ektoras

Lazarević Miloš

Vavakas Alexandros

Table of Contents

Project Overview

A recommender system is a type of filtering system which is used to provide suggestions for items that are most applicable to the end user. These suggestions are meant to guide the user on what would be the best product to buy or the best music to listen to, amongst other topics, according to what other similar users are buying or listening to. This is especially useful when the data set is too large for a common observer.[1] Such a data set can be a book database/collection, which is what we will be covering in our Big Data project.

In this document, we have created a walkthrough of the methods and technologies used, as well as our approach to building the recommender system. The technologies and methods will be outlined, as well as a review of the limitations of our approach.

[1] "Recommender System." Wikipedia, Wikimedia Foundation, 6 Dec. 2022, https://en.wikipedia.org/wiki/Recommender_system.

List of Libraries Used

Q1 – Data Description, Outlier Detection:

- Pandas
- Stats from Scipy
- Numpy

Q2 – Recommender System:

- Pandas
- Numpy
- MatPlotLib
- Cosine_similarity from sklearn

Collaborative Filtering

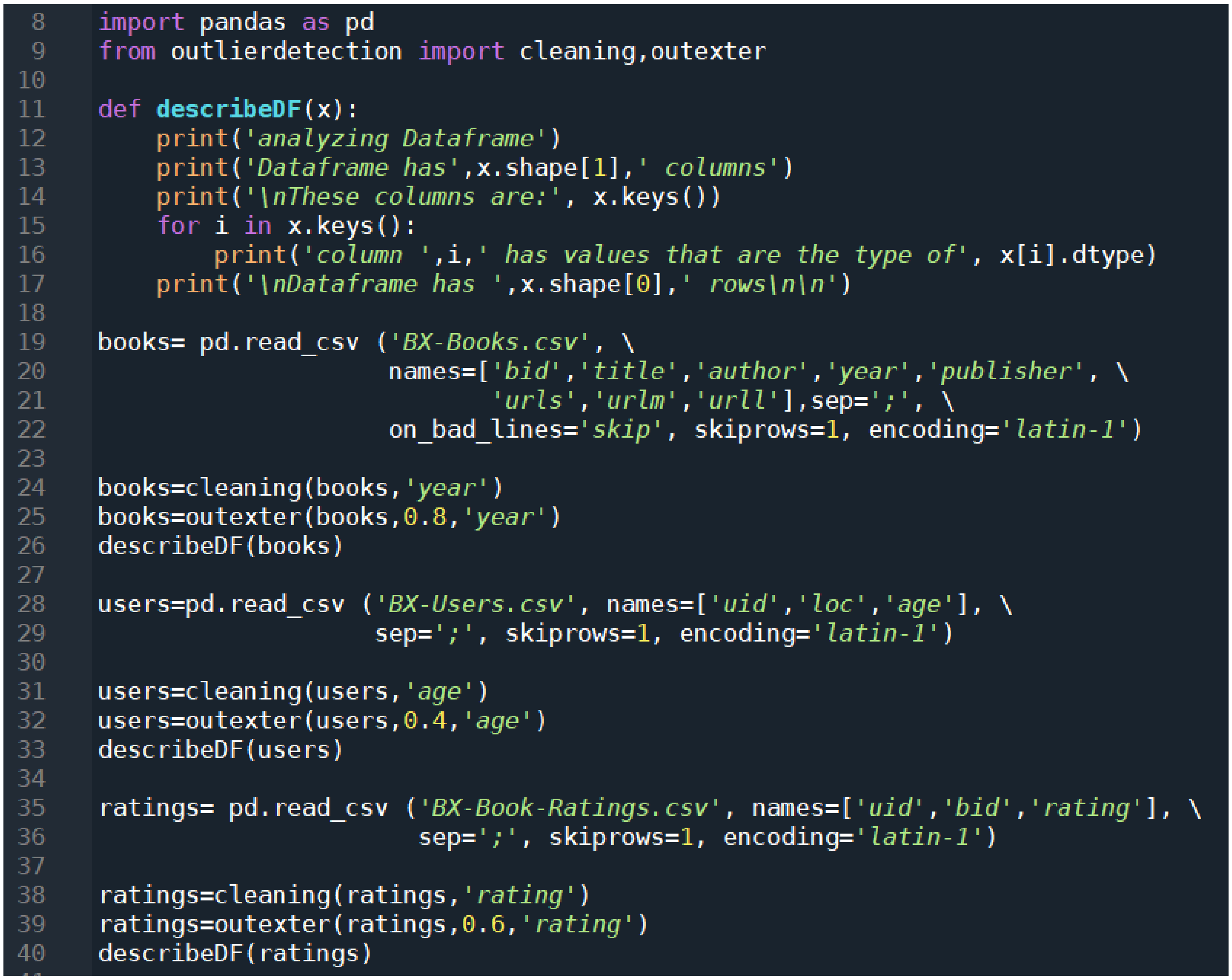
Taking collaborative filtering into perspective for our project, this technique uses similarities between users and items together at the same time to provide a recommendation to the end user, which is the foundation of our recommender system. It does this by considering the interests of neighboring users to filter out and evaluate the information before the recommendation, taking the similarity into account before the final output.[2] In our book recommender system, we have both explicit and implicit categories to use, where implicit concerns whether the user read the book or not, and explicit concerns the rating the user gave a book. Both categories can be satisfied with the user ratings of books, as a rating higher than zero would imply the user had some interest in the book, while higher number ratings would imply that the user liked the book enough to satisfy the recommendation criteria.

[2] "Collaborative Filtering &nbsp;|&nbsp; Machine Learning &nbsp;|&nbsp; Google Developers." Google, Google, https://developers.google.com/machine-learning/recommendation/collaborative/basics.

Understanding the Data

The data set consists of three CSV files: Books, Users and Ratings.[3] Each file contains over 200,000 rows of data. At this point, our goal is to use this correlating data to find the popularity of each author and book, and the book reading activity of each user according to their age range.

To prepare the data set, we read each file as a CSV using pandas, after which we create the "**describeDF**" function to describe the data set.

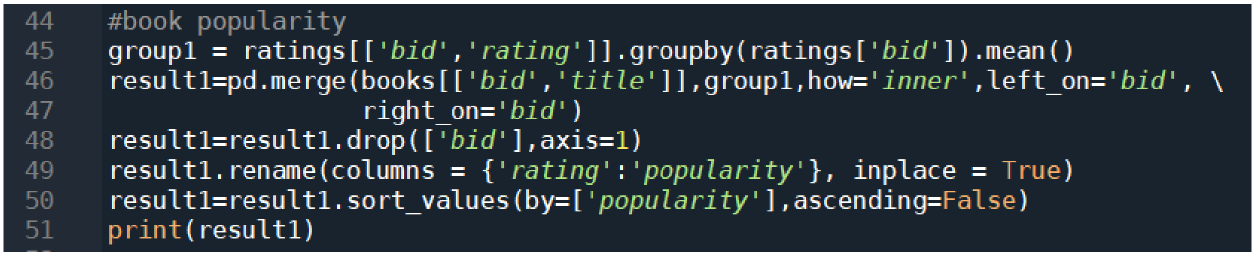


```python
import pandas as pd
from outlierdetection import cleaning,outexter

def describeDF(x):
    print('analyzing Dataframe')
    print('Dataframe has',x.shape[1],' columns')
    print('\nThese columns are:', x.keys())
    for i in x.keys():
        print('column ',i,' has values that are the type of', x[i].dtype)
    print('\nDataframe has ',x.shape[0],' rows\n\n')

books= pd.read_csv ('BX-Books.csv', \
                    names=['bid','title','author','year','publisher', \
                           'urls','urlm','urll'],sep=';', \
                    on_bad_lines='skip', skiprows=1, encoding='latin-1')

books=cleaning(books,'year')
books=outexter(books,0.8,'year')
describeDF(books)

users=pd.read_csv ('BX-Users.csv', names=['uid','loc','age'], \
                   sep=';', skiprows=1, encoding='latin-1')

users=cleaning(users,'age')
users=outexter(users,0.4,'age')
describeDF(users)

ratings= pd.read_csv ('BX-Book-Ratings.csv', names=['uid','bid','rating'], \
                      sep=';', skiprows=1, encoding='latin-1')

ratings=cleaning(ratings,'rating')
ratings=outexter(ratings,0.6,'rating')
describeDF(ratings)
```

To process outliers in our files, we made functions "**outexter**" and "**cleaning**" to handle them in a separate python file (outlierdetection.py) which we then import into the current file (Explained in the next section).

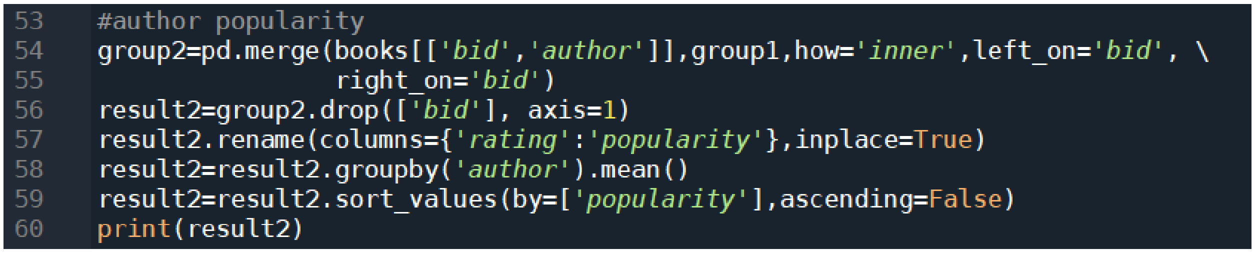
[3] "Book-Crossing Dataset ... Mined by Cai-Nicolas Ziegler, DBIS Freiburg." Book-Crossing Dataset, http://www2.informatik.uni-freiburg.de/~cziegler/BX/.

Then, to find book popularity, we merge the books and book ratings CSV files by book ID (*bid*) using an inner join, after which we drop the unnecessary information and rename the ratings column to popularity to make it more presentable:



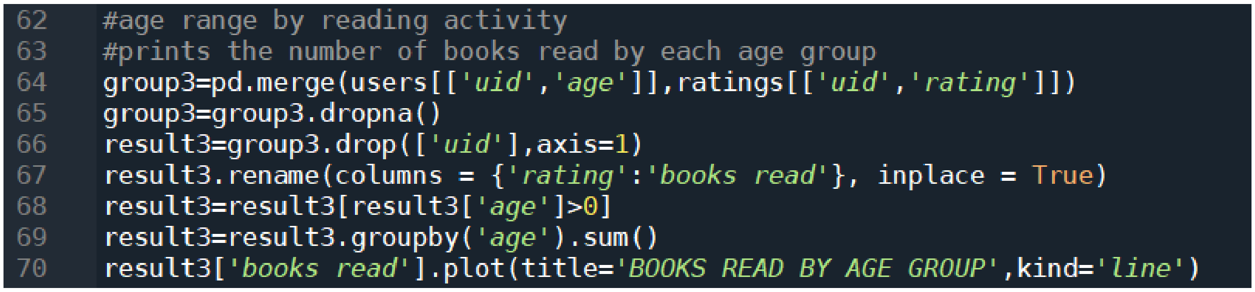
```
#book popularity
group1 = ratings[['bid','rating']].groupby(ratings['bid']).mean()
result1=pd.merge(books[['bid','title']],group1,how='inner',left_on='bid', \
                 right_on='bid')
result1=result1.drop(['bid'],axis=1)
result1.rename(columns = {'rating':'popularity'}, inplace = True)
result1=result1.sort_values(by=['popularity'],ascending=False)
print(result1)
```

To find author popularity, the procedure remains the same, we merge the book ID using inner join. The book ID column is dropped later as we have no further use of it for this example, written as below:



```
#author popularity
group2=pd.merge(books[['bid','author']],group1,how='inner',left_on='bid', \
                right_on='bid')
result2=group2.drop(['bid'], axis=1)
result2.rename(columns={'rating':'popularity'},inplace=True)
result2=result2.groupby('author').mean()
result2=result2.sort_values(by=['popularity'],ascending=False)
print(result2)
```

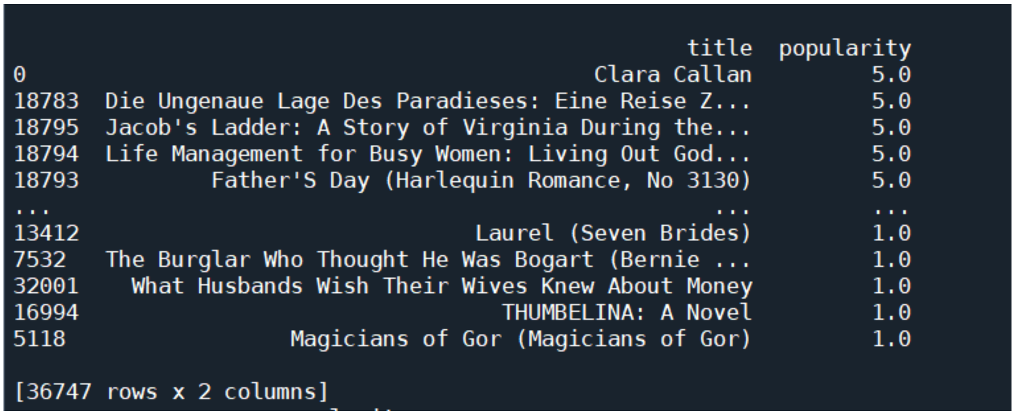
Finally, for reading activity by age range, we merge the users and the ratings tables using user ID (*uid*). The **dropna()** function is then used to discard all NaN values, as we do not need them at this time. We removed all zero age values from the sort as they do not add any value to the rating.



```
#age range by reading activity
#prints the number of books read by each age group
group3=pd.merge(users[['uid','age']],ratings[['uid','rating']])
group3=group3.dropna()
result3=group3.drop(['uid'],axis=1)
result3.rename(columns = {'rating':'books read'}, inplace = True)
result3=result3[result3['age']>0]
result3=result3.groupby('age').sum()
result3['books read'].plot(title='BOOKS READ BY AGE GROUP',kind='line')
```

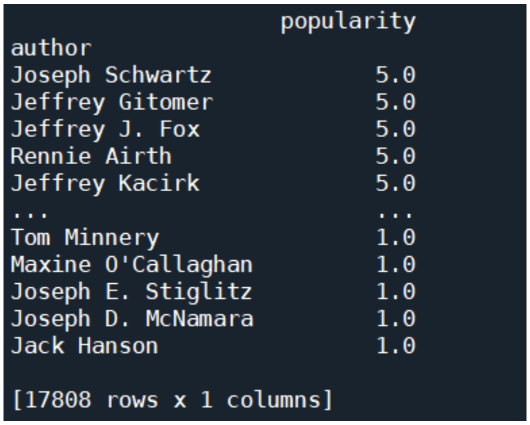
Lastly, the results of these merges are shown below:

Book Popularity



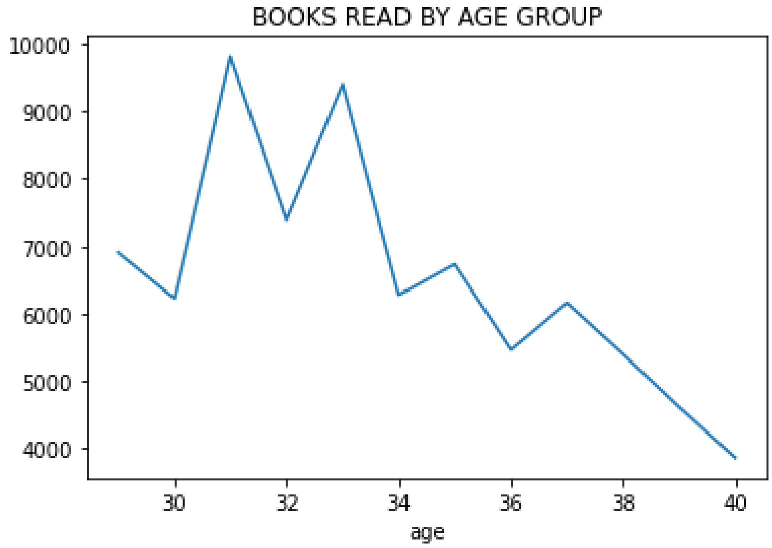
```
                                                  title  popularity
0                                           Clara Callan         5.0
18783  Die Ungenaue Lage Des Paradieses: Eine Reise Z...         5.0
18795  Jacob's Ladder: A Story of Virginia During the...         5.0
18794  Life Management for Busy Women: Living Out God...         5.0
18793          Father'S Day (Harlequin Romance, No 3130)         5.0
...                                                 ...          ...
13412                               Laurel (Seven Brides)         1.0
7532   The Burglar Who Thought He Was Bogart (Bernie ...         1.0
32001    What Husbands Wish Their Wives Knew About Money         1.0
16994                                THUMBELINA: A Novel         1.0
5118                  Magicians of Gor (Magicians of Gor)         1.0

[36747 rows x 2 columns]
```

Author Popularity



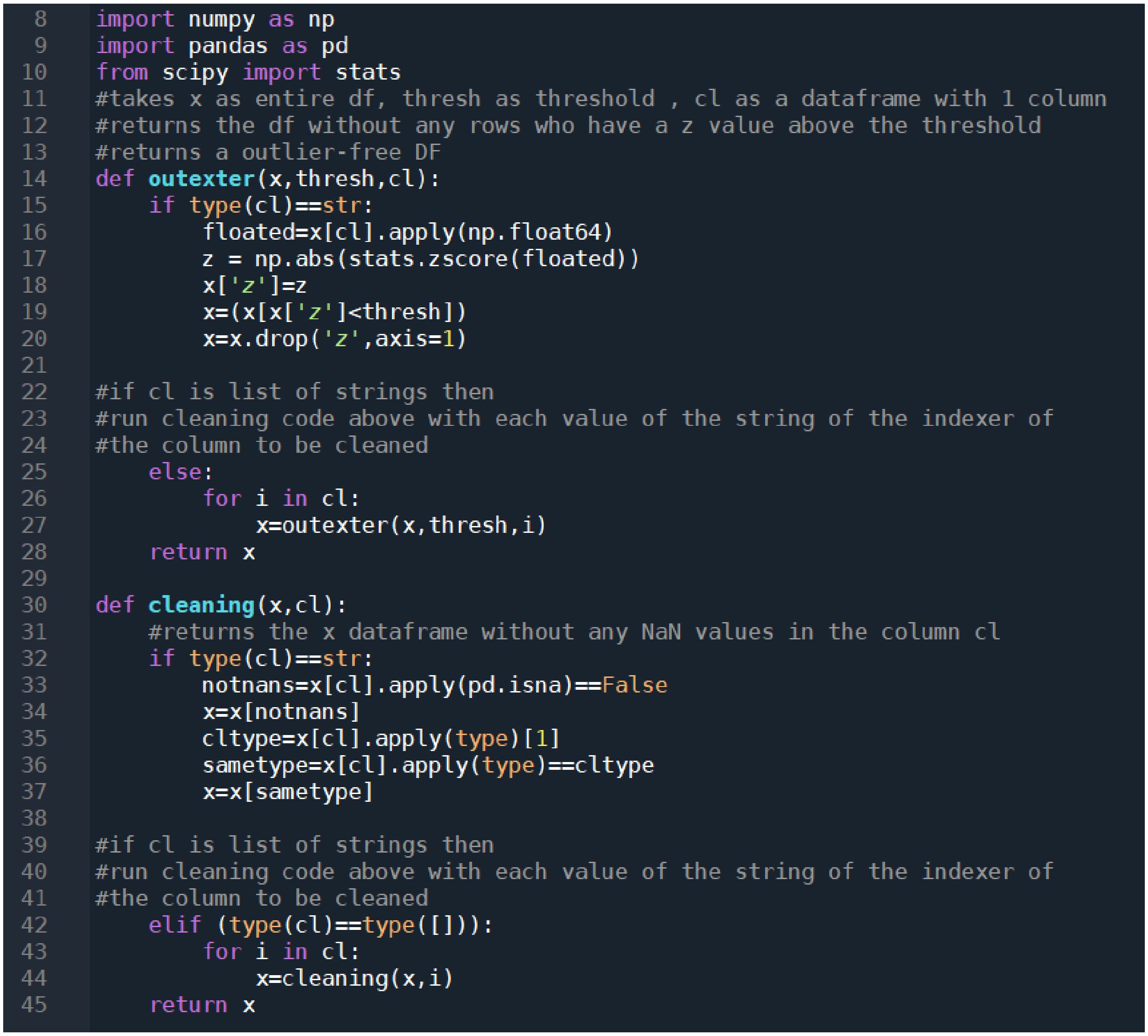
```
                    popularity
author
Joseph Schwartz            5.0
Jeffrey Gitomer            5.0
Jeffrey J. Fox             5.0
Rennie Airth               5.0
Jeffrey Kacirk             5.0
...                        ...
Tom Minnery                1.0
Maxine O'Callaghan         1.0
Joseph E. Stiglitz         1.0
Joseph D. McNamara         1.0
Jack Hanson                1.0

[17808 rows x 1 columns]
```

Age Range by Reading Activity

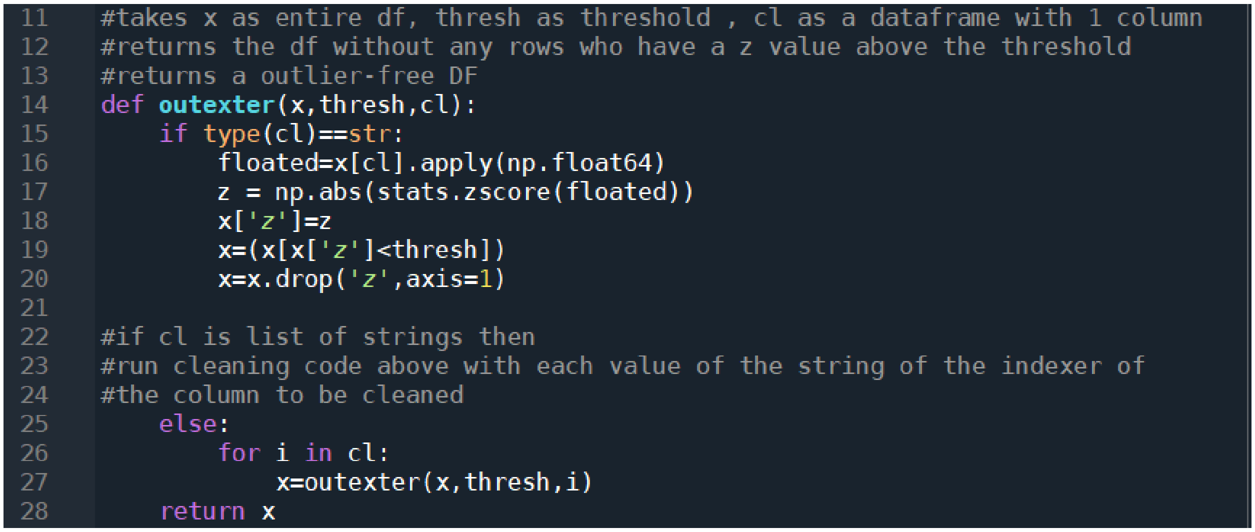


Handling Outliers

In our project, outlier data can usually be represented as unlikely or impossible values, which are not necessarily erroneous, however they can be disruptive during processing or decrease the accuracy of collaborative filtering in our recommendation system. By simply removing or readjusting the outlier data manually in the CSV files we can effectively avoid this issue, however we believe it is not an appropriate solution for our project as this method would suggest that our recommender system is unable to filter out this information. We did not want to limit the functionality of our project by doing so, therefore, we created the python file "**outlierdetection.py**" which holds the functions "**outexter**" and "**cleaning**" to handle these outliers.



```python
import numpy as np
import pandas as pd
from scipy import stats
#takes x as entire df, thresh as threshold , cl as a dataframe with 1 column
#returns the df without any rows who have a z value above the threshold
#returns a outlier-free DF
def outexter(x,thresh,cl):
    if type(cl)==str:
        floated=x[cl].apply(np.float64)
        z = np.abs(stats.zscore(floated))
        x['z']=z
        x=(x[x['z']<thresh])
        x=x.drop('z',axis=1)

#if cl is list of strings then
#run cleaning code above with each value of the string of the indexer of
#the column to be cleaned
    else:
        for i in cl:
            x=outexter(x,thresh,i)
    return x

def cleaning(x,cl):
    #returns the x dataframe without any NaN values in the column cl
    if type(cl)==str:
        notnans=x[cl].apply(pd.isna)==False
        x=x[notnans]
        cltype=x[cl].apply(type)[1]
        sametype=x[cl].apply(type)==cltype
        x=x[sametype]

#if cl is list of strings then
#run cleaning code above with each value of the string of the indexer of
#the column to be cleaned
    elif (type(cl)==type([])):
        for i in cl:
            x=cleaning(x,i)
    return x
```



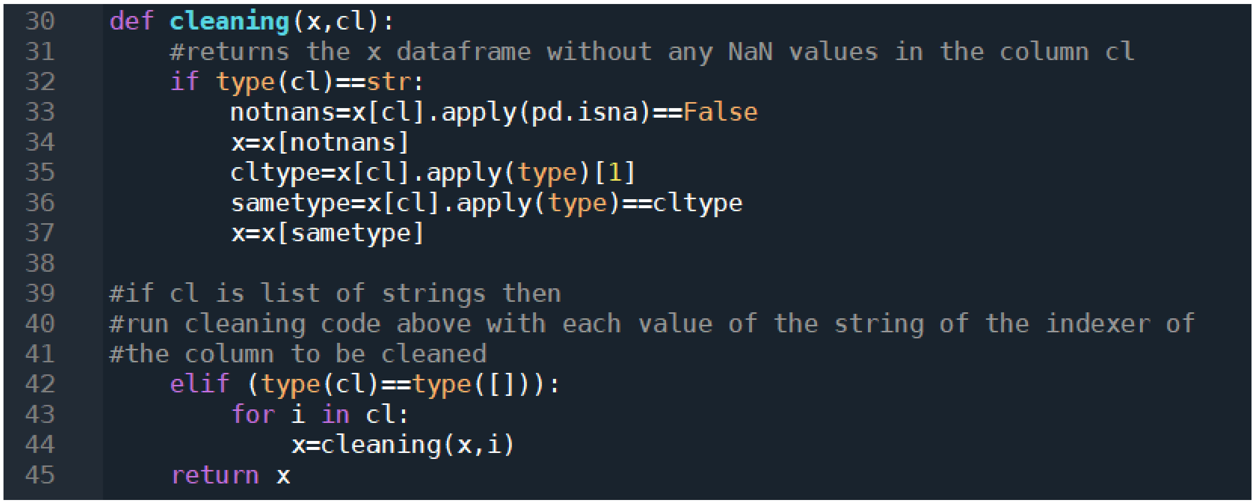
```
#takes x as entire df, thresh as threshold , cl as a dataframe with 1 column
#returns the df without any rows who have a z value above the threshold
#returns a outlier-free DF
def outexter(x,thresh,cl):
    if type(cl)==str:
        floated=x[cl].apply(np.float64)
        z = np.abs(stats.zscore(floated))
        x['z']=z
        x=(x[x['z']<thresh])
        x=x.drop('z',axis=1)

#if cl is list of strings then
#run cleaning code above with each value of the string of the indexer of
#the column to be cleaned
    else:
        for i in cl:
            x=outexter(x,thresh,i)
    return x
```

The “**outexter**” function takes three arguments:

1) X – The entire subject data frame.
2) Thresh – The assigned threshold.
3) CL – One or more columns from a data frame.

The method of calculation is the Z-Score. Also referred to as the standard score, it is the number of standard deviations by which the target data point is above or below the mean value of the population.[4] In our outlier detection function “**outexter**”, the column values to be filtered are converted to z-values. Then, we use the “**thresh**” variable to compute the data and then once the computing is finished, we drop all values whose corresponding z-values are higher than this threshold, removing the outliers. The rest of the data is returned to the main file. To simplify the calculation process we use the **stats** library from **scipy** in the function. From this library, the **zscore** function takes the preprocessed “**tested**” array-like object and returns the z-scores, standardized by mean and standard deviation of the input array.

[4] “Standard Score.” Wikipedia, Wikimedia Foundation, 17 Dec. 2022, https://en.wikipedia.org/wiki/Standard_score.



```
def cleaning(x,cl):
    #returns the x dataframe without any NaN values in the column cl
    if type(cl)==str:
        notnans=x[cl].apply(pd.isna)==False
        x=x[notnans]
        cltype=x[cl].apply(type)[1]
        sametype=x[cl].apply(type)==cltype
        x=x[sametype]

#if cl is list of strings then
#run cleaning code above with each value of the string of the indexer of
#the column to be cleaned
    elif (type(cl)==type([])):
        for i in cl:
            x=cleaning(x,i)
    return x
```

Similarly, the “**cleaning**” function takes the **X** and **CL** as arguments. Its purpose is to eliminate NaN values and incorrect data in the data frame and returns the cleaned data frame as output.

With all the above mentioned, we have successfully described and displayed the data set, removing any inconsistencies, outliers, and errors which could be detrimental when visualizing the data. We can now proceed to the next stage of this project, which is the recommender system.

Recommender System

We are asked to build a system that recommends books to the users. The recommender system will be based on a method called: collaborative filtering:

Collaborative filtering's idea essentially is that the users that are most like the active user will be used to suggest a book which the user has not seen before. The build-up of the system is based on a series of steps that are going to be discussed below:

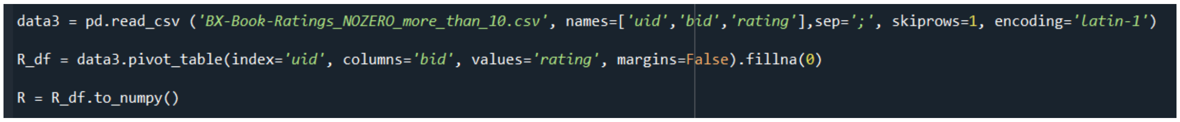
Before we begin, we examine the quantity of data we are going to use for our code. We used the whole "BX-Users.csv" and "BX-Books.csv". Firstly, we took a sample of 1% of the csv with the ratings (***BX-Book-Ratings.csv***). After we got our desirable results, we tried increasing the sampling size. We ended up with 10% of the csv. We tried computing it with the whole data, but there were some errors when we were creating the pivot tables because we reached the maximum memory that it can store.

Find Similarities

1. We need to find the similarity between all of pairs users based on the books they have read and store them in a csv file.

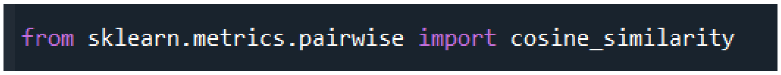
Firstly, we need to access the data that we care about, for our similarities to have actual meaning. What we mean by that is, that in the csv file the zero ratings will be removed, because they are not a realistic rating for the users. The zero ratings may be an indication that the user has accessed the book but has not rated it. Either way, if these ratings are not removed, the similarities will not be accurate. Also, to enhance our recommender system, we are keeping the users who have rated more than ten books. The reason why we do that, is to create a more reliable system with users that are more used to the process and thus their rating may be of higher significance.

After these processes are done, we import the processed csv, and convert it to a panda data frame, and that to a pivot table.

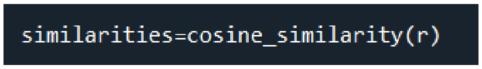


```
data3 = pd.read_csv ('BX-Book-Ratings_NOZERO_more_than_10.csv', names=['uid','bid','rating'],sep=';', skiprows=1, encoding='latin-1')

R_df = data3.pivot_table(index='uid', columns='bid', values='rating', margins=False).fillna(0)

R = R_df.to_numpy()
```

We use the ***sklearn.metrics.pairwise*** library to import the ***cosine_similarity*** function. With this function we convert the pivot table to another data frame representing the cosine similarities between each user. We store that data frame to a *csv* called ***user-pairs-books.data***.



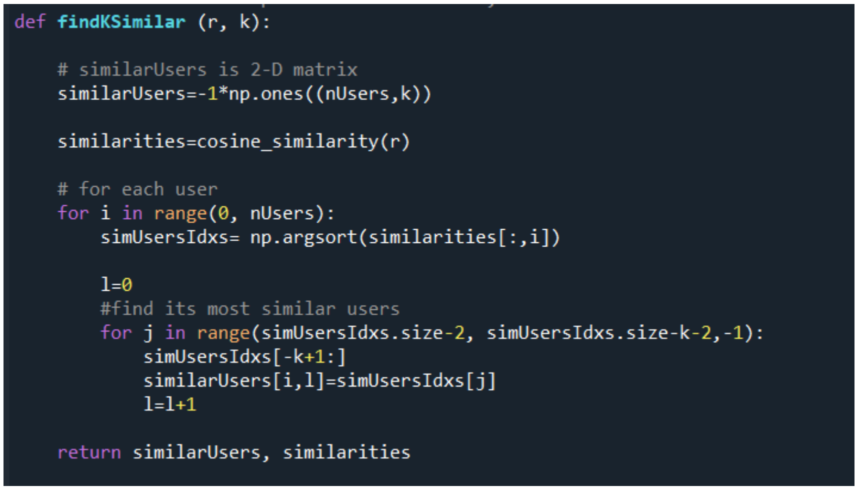
```
from sklearn.metrics.pairwise import cosine_similarity
```



```
similarities=cosine_similarity(r)
```

2. We need to find the k-nearest neighbors for each user based on reading similarity, and thus based on the similarity we calculated on the previous question.

At first, we choose the number of neighbors k we would like to find. After that, we find through the similarities we calculated, with a method called sorted indexing (function ***argsort***), the most similar users' indexes. Then we create a list with the k- nearest users for each one. We convert the list to a data frame, which we then turn it to a ***JSON*** file called ***neighbors-k-books.data***.



```
def findKSimilar (r, k):

    # similarUsers is 2-D matrix
    similarUsers=-1*np.ones((nUsers,k))

    similarities=cosine_similarity(r)

    # for each user
    for i in range(0, nUsers):
        simUsersIdxs= np.argsort(similarities[:,i])

        l=0
        #find its most similar users
        for j in range(simUsersIdxs.size-2, simUsersIdxs.size-k-2,-1):
            simUsersIdxs[-k+1:]
            similarUsers[i,l]=simUsersIdxs[j]
            l=l+1

    return similarUsers, similarities
```

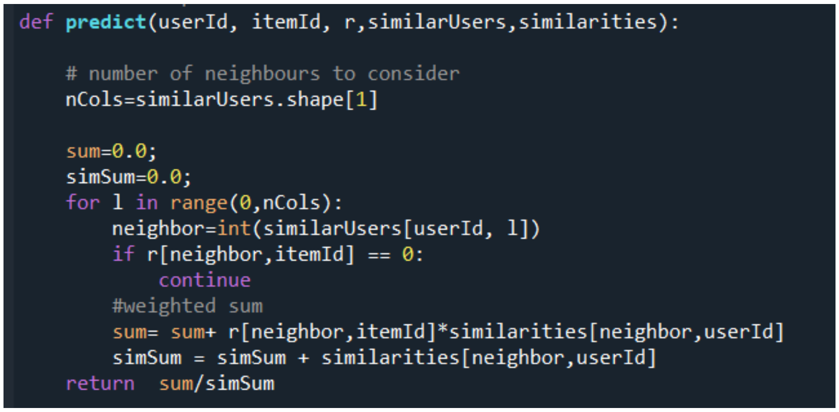
Trying different options of k, we choose the ones with smaller RMSE which will be calculated afterwards.

Recommend

We compute the recommendation according to the collaborative filtering. The core idea is the neighbors vote according to their similarity to the user.

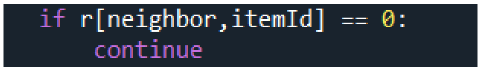
Our goal is to predict for the active user the rating of each item, and afterwards recommend the best predictions. After we have determined a list of users like the active user, we calculate the rating which that user would give to a certain item. We can predict that a user's rating R_U for an item will be close to the average of the ratings given to the item by the top k users most like the active one. However, we consider the weighted approach, where the rating of the most similar user matters more than the second most similar user and so on. We multiply each rating by a similarity factor S_U, to add weights to the ratings. We calculate the weighted average R_U using the formula:

$$R_U = \frac{(\sum_{u=1}^{n} R_U * S_U)}{\sum_{u=1}^{n} S_U}$$



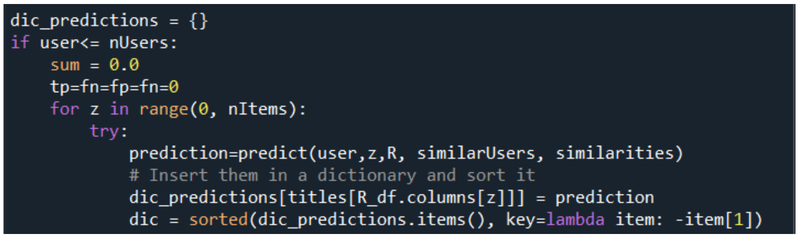
```
def predict(userId, itemId, r,similarUsers,similarities):

    # number of neighbours to consider
    nCols=similarUsers.shape[1]

    sum=0.0;
    simSum=0.0;
    for l in range(0,nCols):
        neighbor=int(similarUsers[userId, l])
        if r[neighbor,itemId] == 0:
            continue
        #weighted sum
        sum= sum+ r[neighbor,itemId]*similarities[neighbor,userId]
        simSum = simSum + similarities[neighbor,userId]
    return  sum/simSum
```

When we compute that function in python, we face a problem. For the active user, some of the k-nearest neighbors may have not rated the item we choose. Therefore, when we increase the number of neighbors, the top ratings tend to decrease because we add null ratings of the most similar user to the active one for the item. Hence, we add an if statement inside the function to check if the neighbors have indeed rated the item. If they have not, we trust the ratings of the remainder neighbors.



```
  if r[neighbor,itemId] == 0:
      continue
```

Using the similarity table and similar users from the previous question, we can predict the rating of a user for an item. Iterating through the items for the active user, we can append the ratings in a dictionary with key the title of the movie and value the prediction rating. After we sort the dictionary based on the ratings, we display the highest rated ones. In order to insert the titles of the books we imported the *BX-Books.csv* file, in which we drop most of the columns, to decrease the space of the books data frame.

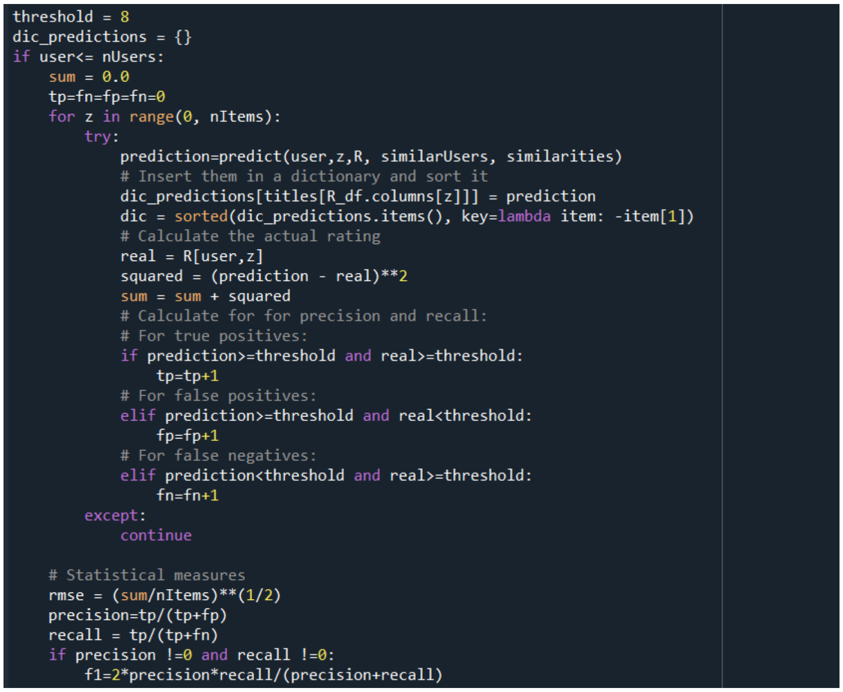


```
dic_predictions = {}
if user<= nUsers:
    sum = 0.0
    tp=fn=fp=fn=0
    for z in range(0, nItems):
        try:
            prediction=predict(user,z,R, similarUsers, similarities)
            # Insert them in a dictionary and sort it
            dic_predictions[titles[R_df.columns[z]]] = prediction
            dic = sorted(dic_predictions.items(), key=lambda item: -item[1])
```

Evaluate the recommendation

We use the following statistical measures to evaluate the system: Root mean squared error (RMSE), Precision, Recall, F1 for different neighborhood sizes.

For the active user, we iterate through every item/book.



```python
threshold = 8
dic_predictions = {}
if user<= nUsers:
    sum = 0.0
    tp=fn=fp=fn=0
    for z in range(0, nItems):
        try:
            prediction=predict(user,z,R, similarUsers, similarities)
            # Insert them in a dictionary and sort it
            dic_predictions[titles[R_df.columns[z]]] = prediction
            dic = sorted(dic_predictions.items(), key=lambda item: -item[1])
            # Calculate the actual rating
            real = R[user,z]
            squared = (prediction - real)**2
            sum = sum + squared
            # Calculate for for precision and recall:
            # For true positives:
            if prediction>=threshold and real>=threshold:
                tp=tp+1
            # For false positives:
            elif prediction>=threshold and real<threshold:
                fp=fp+1
            # For false negatives:
            elif prediction<threshold and real>=threshold:
                fn=fn+1
        except:
            continue

    # Statistical measures
    rmse = (sum/nItems)**(1/2)
    precision=tp/(tp+fp)
    recall = tp/(tp+fn)
    if precision !=0 and recall !=0:
        f1=2*precision*recall/(precision+recall)
```

For the RMSE:

We calculate the prediction of the user for the item. Then we calculate the difference between the prediction and the real rating and square it. We use the formula below:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(prediction_i - real_i)^2}{N}}$$

For Precision and Recall:

We calculate for the active user all the true positives, false positives, and false negatives. In order to compute those, we decide a threshold to determine the “acceptable” or “good” from the “not acceptable” or “bad” ratings. By changing the threshold, we observe rapid changes to the measures.

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

$$recall = \frac{true\ positives}{true\ positives + false\ negatives}$$

For F1:

We calculate it using the formula:

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

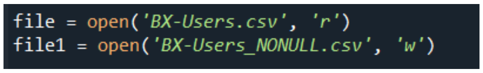
Improving the recommendation-algorithms

One way to improve the recommendation is adding a new statistical parameter called Pearson correlation coefficient (r). This parameter measures the correlation between two sets of data, in this case the prediction and the real rating.

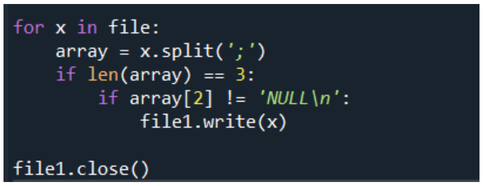
Improving the recommendation-more features

We tried improving the recommendation by adding some demographic features. We used the "Age" column from the *"BX-Users.csv"* to calculate the similarities between the users based on their age.

Before we did that, we made sure that the csv was cleaned and organized, by choosing all the non-NULL values from the "Age" column and store them in a new csv called *"BX-Users_NONULL.csv"*.

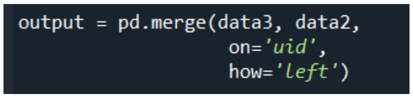


```
file = open('BX-Users.csv', 'r')
file1 = open('BX-Users_NONULL.csv', 'w')
```

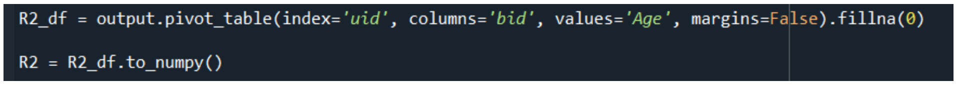


```
for x in file:
    array = x.split(';')
    if len(array) == 3:
        if array[2] != 'NULL\n':
            file1.write(x)

file1.close()
```

We merged the two csv's to create a new pivot table with the age of the users as values.

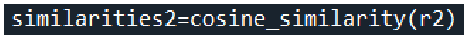


```
output = pd.merge(data3, data2,
                   on='uid',
                   how='left')
```



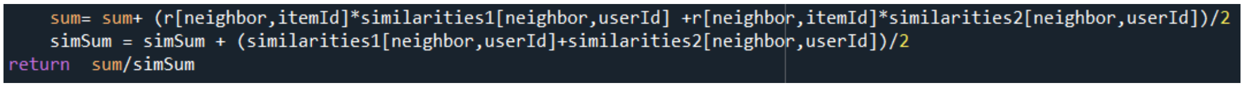
```
R2_df = output.pivot_table(index='uid', columns='bid', values='Age', margins=False).fillna(0)

R2 = R2_df.to_numpy()
```

We add a new input *r2* to the function that calculates the cosine similarities (**find_similar**), to calculate the similarities of the users also based on their age.



```
similarities2=cosine_similarity(r2)
```

Later, when we compute the predictions, we calculate the average of the two similarities: the one based on the ratings and the one based on the age. For that part, we use the most similar neighbors that were found based on the ratings data frame, because this is our main source for the results.



```
    sum= sum+ (r[neighbor,itemId]*similarities1[neighbor,userId] +r[neighbor,itemId]*similarities2[neighbor,userId])/2
    simSum = simSum + (similarities1[neighbor,userId]+similarities2[neighbor,userId])/2
return  sum/simSum
```

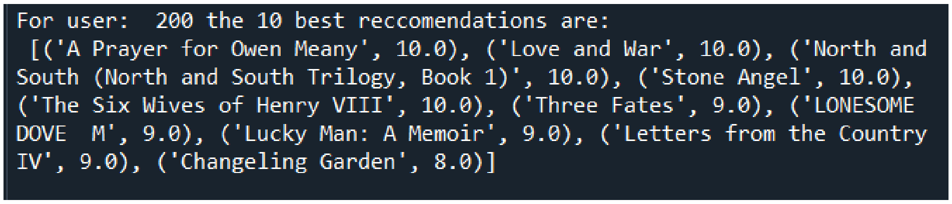
We add this part inside the **predict** function, and then we output the outcome to use it for the evaluation and the display of the results.

Results

For example, the 200th user, who is user: “9381”, we get the results below:

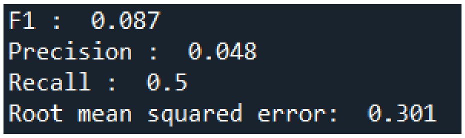
For number of neighbors k = 3, and a threshold = 8:

The top 10 recommendations are:

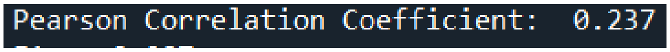


```
For user:  200 the 10 best reccomendations are:
 [('A Prayer for Owen Meany', 10.0), ('Love and War', 10.0), ('North and
South (North and South Trilogy, Book 1)', 10.0), ('Stone Angel', 10.0),
('The Six Wives of Henry VIII', 10.0), ('Three Fates', 9.0), ('LONESOME
DOVE  M', 9.0), ('Lucky Man: A Memoir', 9.0), ('Letters from the Country
IV', 9.0), ('Changeling Garden', 8.0)]
```

The statistical measures are:



```
F1 :  0.087
Precision :  0.048
Recall :  0.5
Root mean squared error:  0.301
```



```
Pearson Correlation Coefficient:  0.237
```

Limitations

1) Currently, for neighboring users with no ratings or ratings of zero, the recommender system results get heavily influenced and produce a lower overall score of similarity to the end user.

2) As we wanted to build our recommender system without tampering with the data set, the cleaned data from the first part of this project is not included in the recommender system. This way, we successfully make recommendations to the end user, however it is likely that for some users the recommender system will be less accurate due to the outliers and incorrect data points present in the original data set.

3) The Q1 and Q2 segments are not connected to each other and run independently. While this may be optimal in a developer environment, it is by no means suitable for production/consumer use.

4) The project lacks a user interface. All the results are printed in the kernel and thus cannot be used by someone other than a developer.

5) The recommender system is specifically customized for the books data set. It cannot work for any other data sets, making it very limited in scope.

Future Improvements

1) By changing the number of users taken into consideration during the calculations of the recommender system, we will likely get better results.

2) Implementing a user interface and perhaps allowing input of new users/books we could drastically increase the functionality of our project.

3) Exploring ways to write code so that our recommender system accepts other data sets.

4) Storing information in a database instead of locally will help alleviate the size, as well as give the recommendation system back-end functionality

5) Including all the above, connecting Q1 and Q2 would give us the ability to transform this project into an application.

Conclusion

To conclude with, while our recommender system is functional and completes the task of this project, carrying out the data description, recommendation and evaluation, there is still much more room for improvement which we would like to implement in the future. That being said, this concludes our Big Data recommender system report.

References

1) “Recommender System.” Wikipedia, Wikimedia Foundation, 6 Dec. 2022, https://en.wikipedia.org/wiki/Recommender_system.

2) “Collaborative Filtering &nbsp;|&nbsp; Machine Learning &nbsp;|&nbsp; Google Developers.” Google, Google, https://developers.google.com/machine-learning/recommendation/collaborative/basics.

3) “Book-Crossing Dataset ... Mined by Cai-Nicolas Ziegler, DBIS Freiburg.” Book-Crossing Dataset, http://www2.informatik.uni-freiburg.de/~cziegler/BX/.

4) “Standard Score.” Wikipedia, Wikimedia Foundation, 17 Dec. 2022, https://en.wikipedia.org/wiki/Standard_score.