**Final Project - Book Recommendation Systems**

**Seung Pang (sp4232) | Jun Yong Song (jys358) | Hogyeong Kim (hk3337)**

# **Problem and Hypothesis**

This project focuses on modeling a book recommendation system for users on a certain website. Modeling is done by utilizing all three recommender system approaches discussed in the lecture: content-based, collaborative filtering, and latent factor-based. The challenge of implementing this book recommendation system is having no dataset on book genre and book description. However, given just Book-Title data we aimed to model an effective book recommendation system using the three approaches.

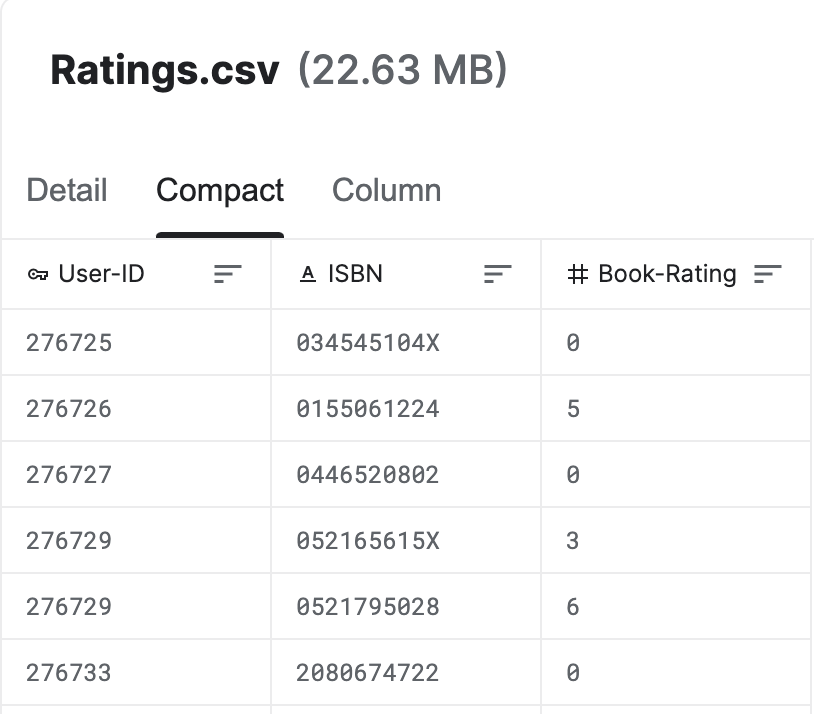
# **Dataset**

# <https://www.kaggle.com/arashnic/book-recommendation-dataset>

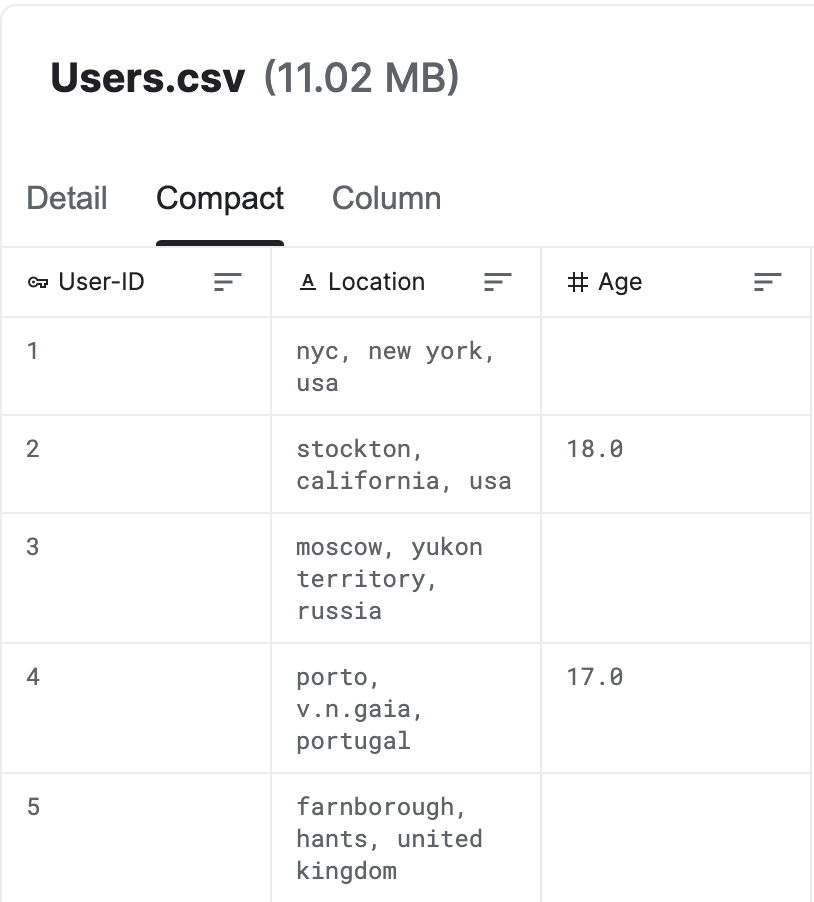
# The book recommendation dataset Books.csv, Ratings.csv, and Users.csv above were obtained from Kaggle.

# 

Books are uniquely identified by their ISBN. According to the owner, invalid ISBNs have already been removed from the dataset. There are a total of eight columns and contain information obtained from Amazon Web Services including Book-Title, Book-Author, Year-Of-Publication, Publisher, and Image-URLs linking to cover images. In the case of several authors, only the first is provided.



Ratings.csv dataset contains a total of three columns, User-ID, ISBN, and Book-Rating, and book ratings are represented on a scale of 1 (the lowest) to 10 (the highest).



Users.csv dataset contains general information about users and has a total of three columns, User-ID, Location, and Age. Age is NULL if not provided by the user.

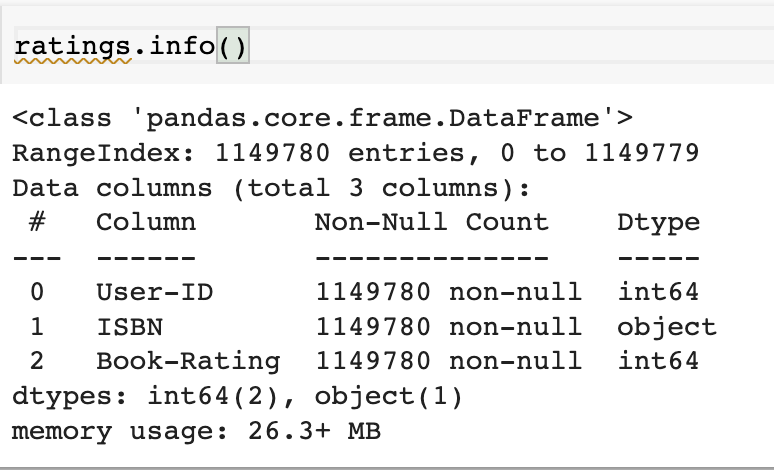
# 

# 

There are a total of 271,360 books, 278,858 users, and 1,149,780 ratings in the given dataset as shown below.

# 

# 

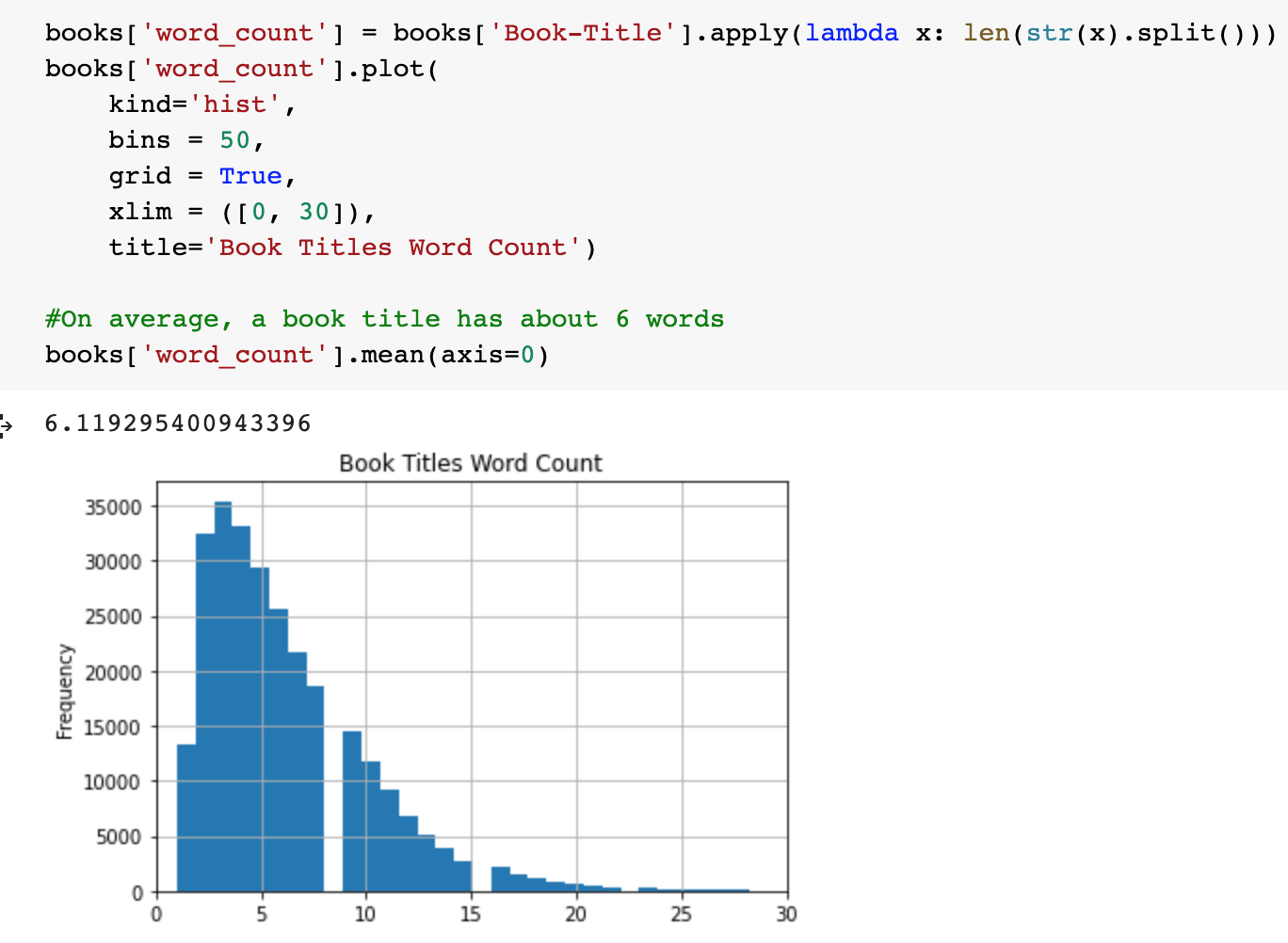


# **Models**

# **1. Contend-Based**

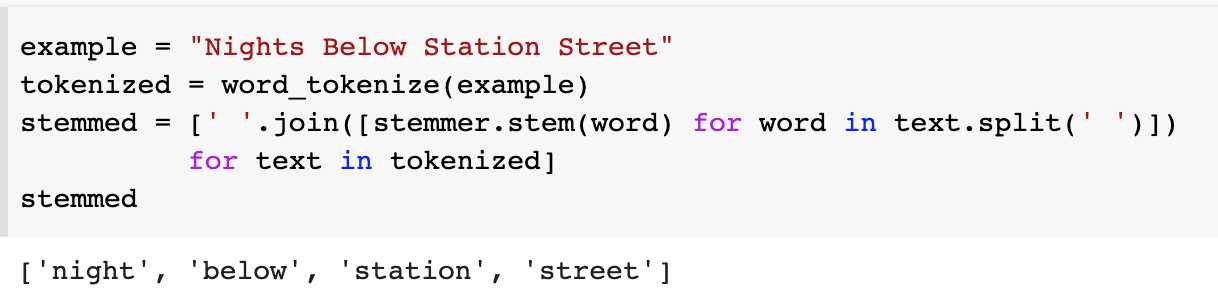
***Data Exploration***

On average, book titles in this dataset contain around 6 words.



***Text Pre-Processing using Stemming and Lemmatization***

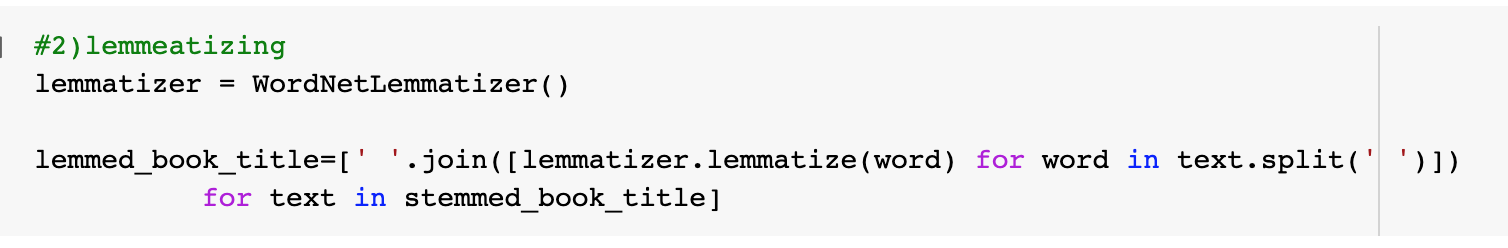
Using the **Python Natural Language Tool Kit (nltk) package**, Text Normalization techniques are applied on Book-Title. According to [DataCamp](https://www.datacamp.com/community/tutorials/stemming-lemmatization-python), Text Normalization techniques prepare text, words, and documents for further processing.



**Stemming** changes derived words to the root forms. **Stemming** is performed by removing the suffixes or prefixes used with a word.

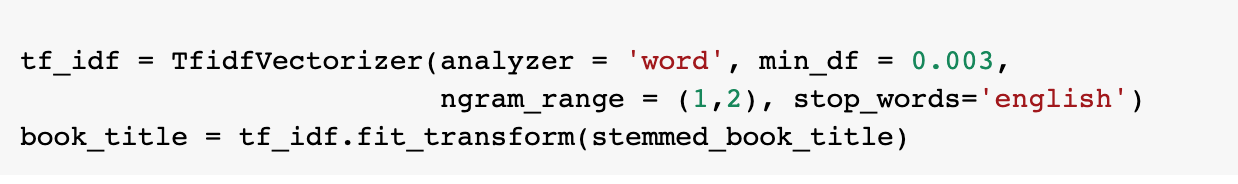
# 

According to [GeeksforGeeks](https://www.geeksforgeeks.org/python-lemmatization-with-nltk/), **Lemmatization** is similar to stemming but it brings context to the words.



**Stemming and Lemmatization** are applied to book titles to prepare the text data.

Using **Scikit-Learn’s built-in TfIdfVectorizer class**, we remove English stop words like “the”, and “an” which do not add any meaningful information about the book.



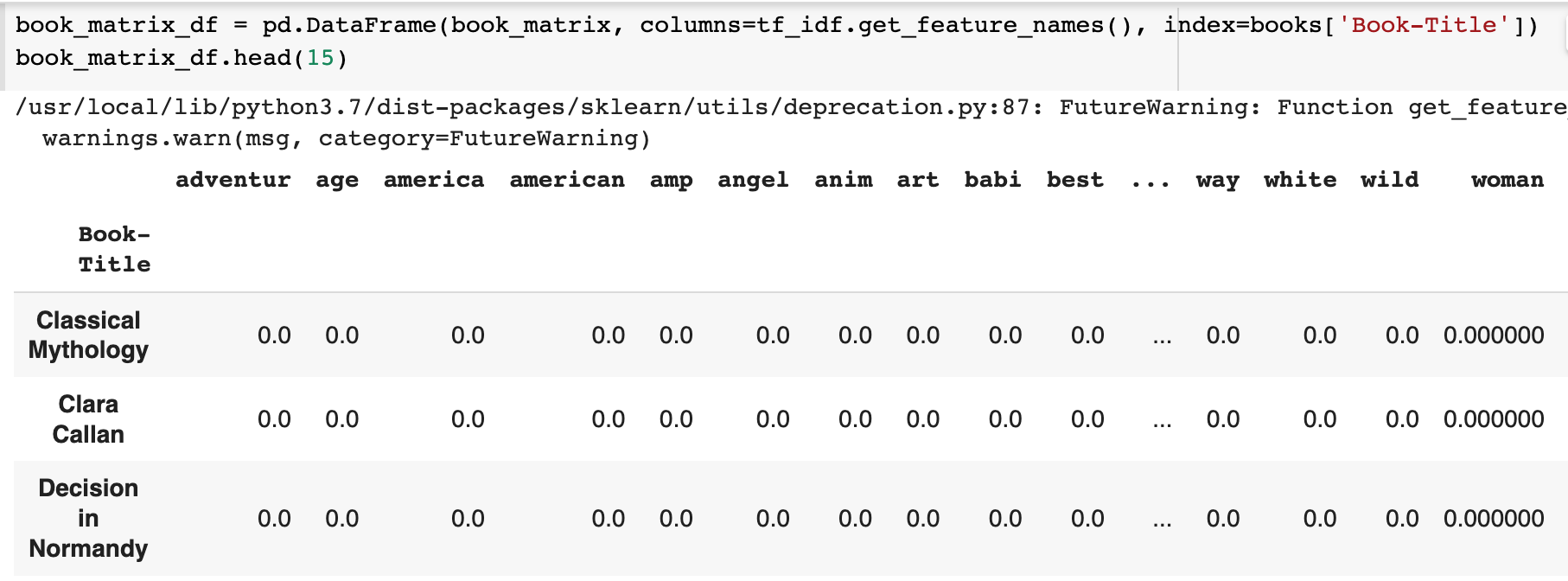
**Min\_df** is for removing words that appear too infrequently. **Min\_df** of 0.003 given above removes terms that appear in less than 0.3% of the documents.

If **min\_df** value is increased to 1, the book\_title matrix increases to (271,360 books, 569,779 terms). With **min\_df** value of 0.003, the book\_title matrix is (271,360 books, 164 terms).

The following is the shape of the **TF-IDF** matrix, book\_title. The matrix has 271,360 rows of book titles and 164 columns of pre-processed keywords.

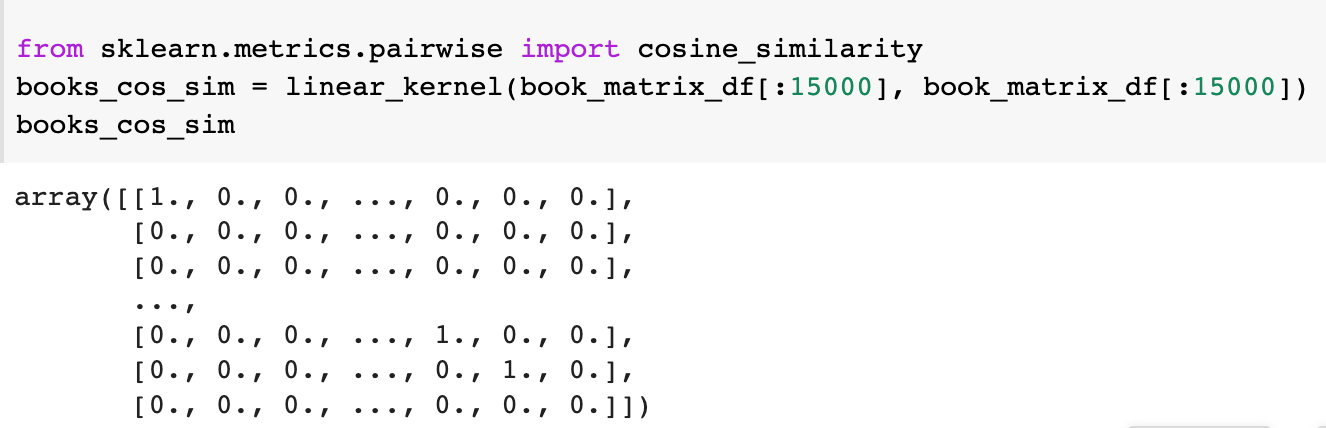


And **Book\_matrix\_df** is created based on Book-Title as rows, and pre-processed tokenized keywords as columns.



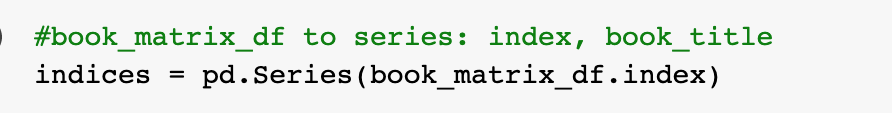
***Measuring Similarity***

Cosine similarity is a metric used to find the similarity of texts by measuring the cosine angle between two matrices.

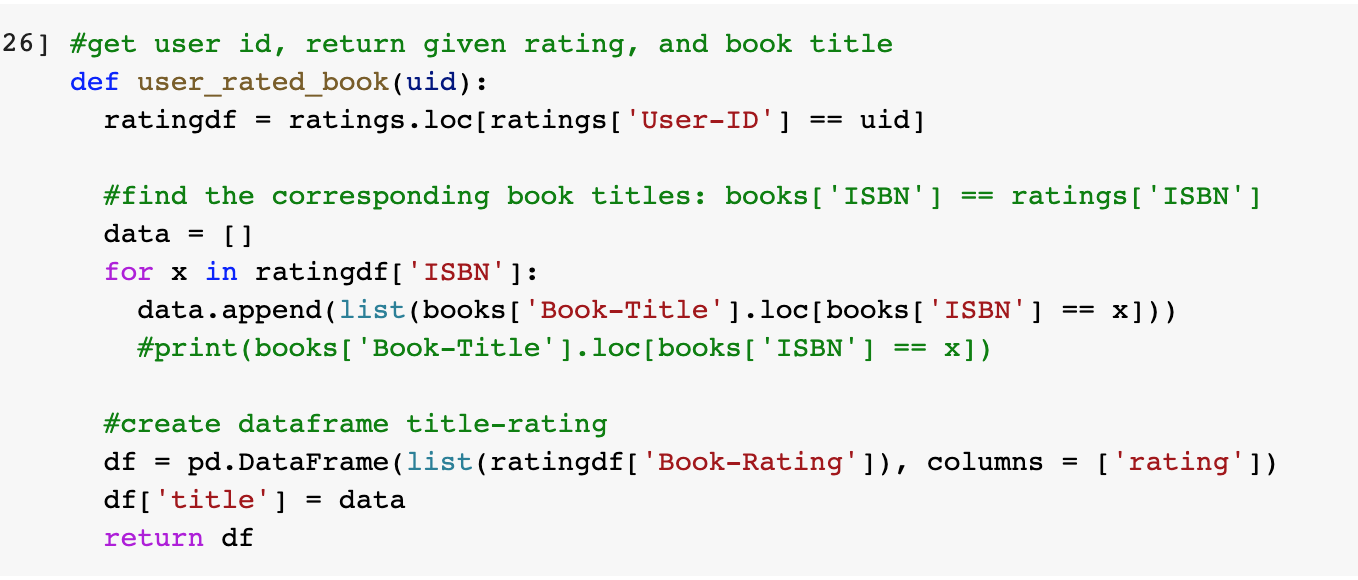
Using cosine similarity, we compare **Book\_matrix\_df** to find how similar book titles are in a range of 0, being the least similar, to 1, being the most similar. 

***Recommendation***

Using Pandas Series, book\_matrix\_df is rearranged based on index and book title.



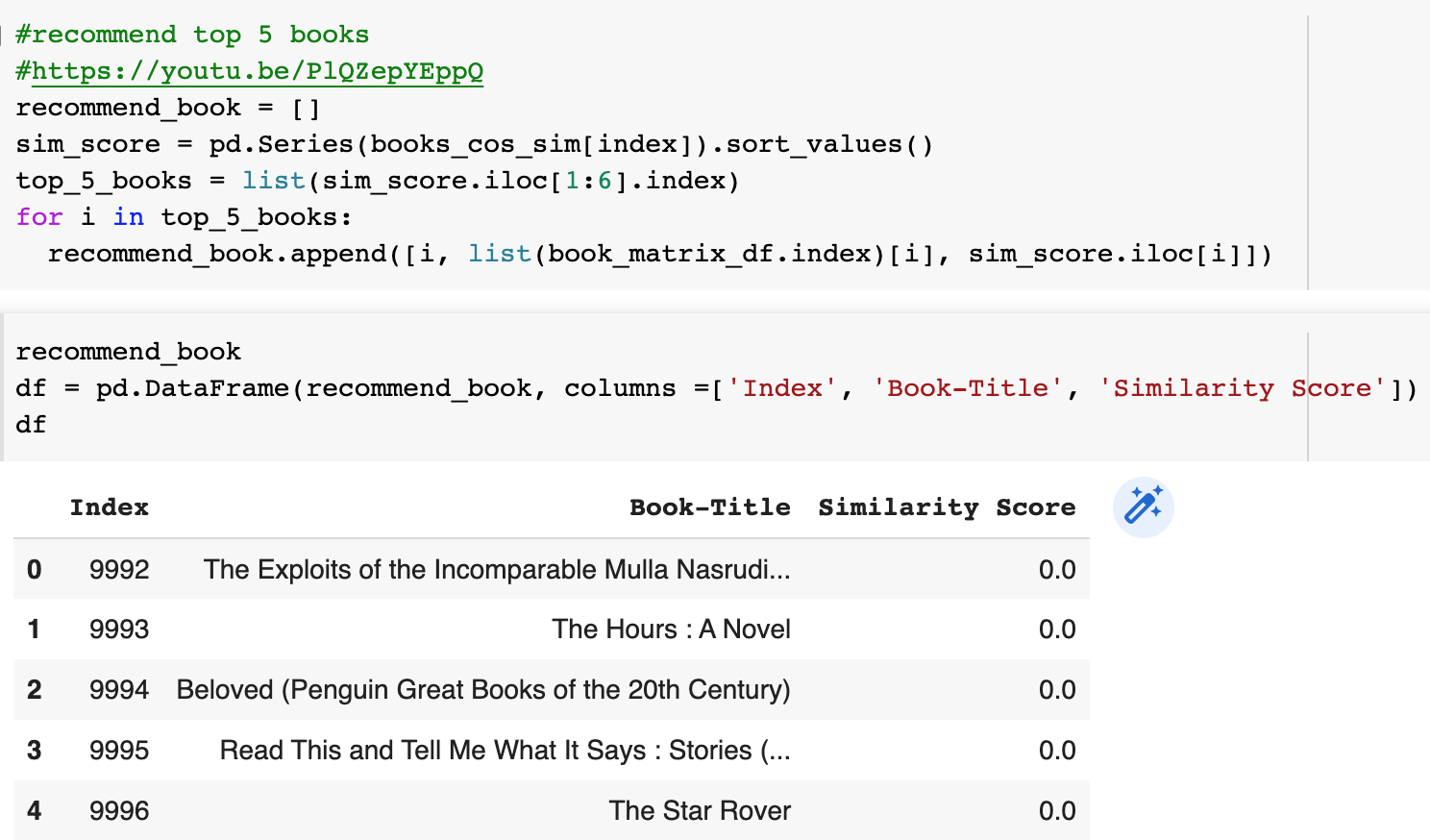
Book-Title is sorted based on the similarity scores and the top 5 books are recommended given a book title. The book title is selected from a given user-id and the first rated book is selected from a list of options.





Because similarity scores are measured solely based on the similarity of keywords and book titles do not directly explain the genre and description of the books, the similarity scores are assumed to be very low since book title, on average, has about six words.

With the book genre and book description data, the content-based model can be implemented much better and accurately.



# **2. Item-Based Collaborative Filtering**

**Data Merging**

Since “books” and “ratings” datasets have corresponding columns as “ISBN”, we merged those datasets first. Then we merged the new dataset with “users” dataset with corresponding value “User-ID”.

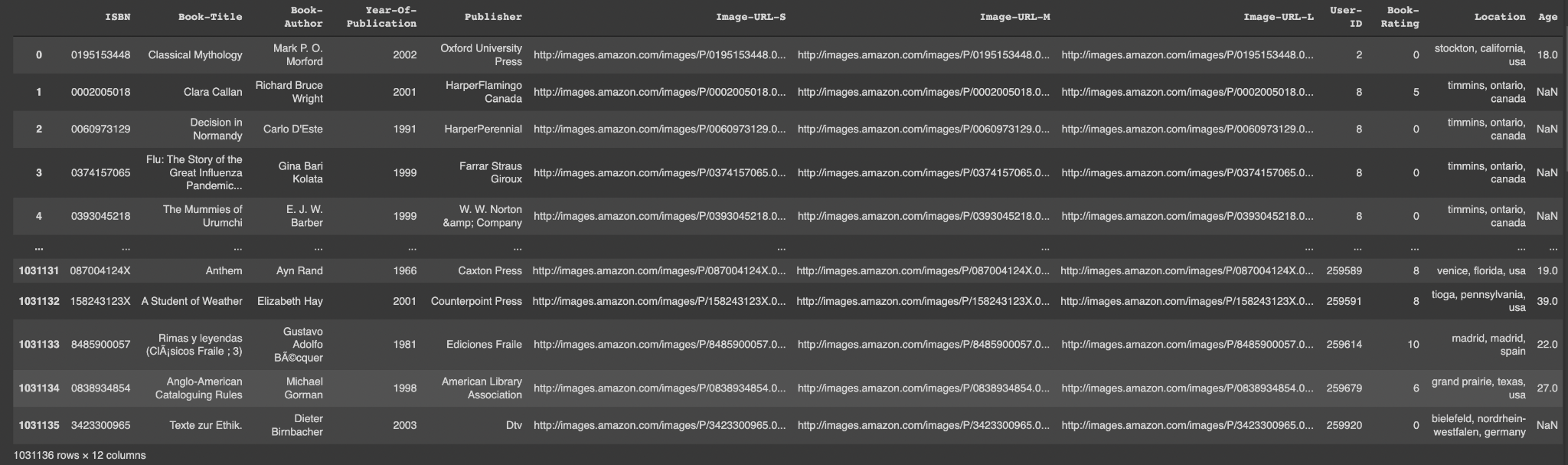
# Merge and transform datasets

books\_ratings = pd.merge(books,ratings,on="ISBN")

books\_ratings\_users = pd.merge(books\_ratings,users, on ="User-ID")

newData = books\_ratings\_users

newData



**Clearing Data**

Since, we only need to know the title of books, user-id, and books ratings, columns like ISBN, Book-Author, Year-of-Publication, Publisher, Image-URL-S, Image-URL-M, Image-URL-L, Location, and Age were removed from the dataset.

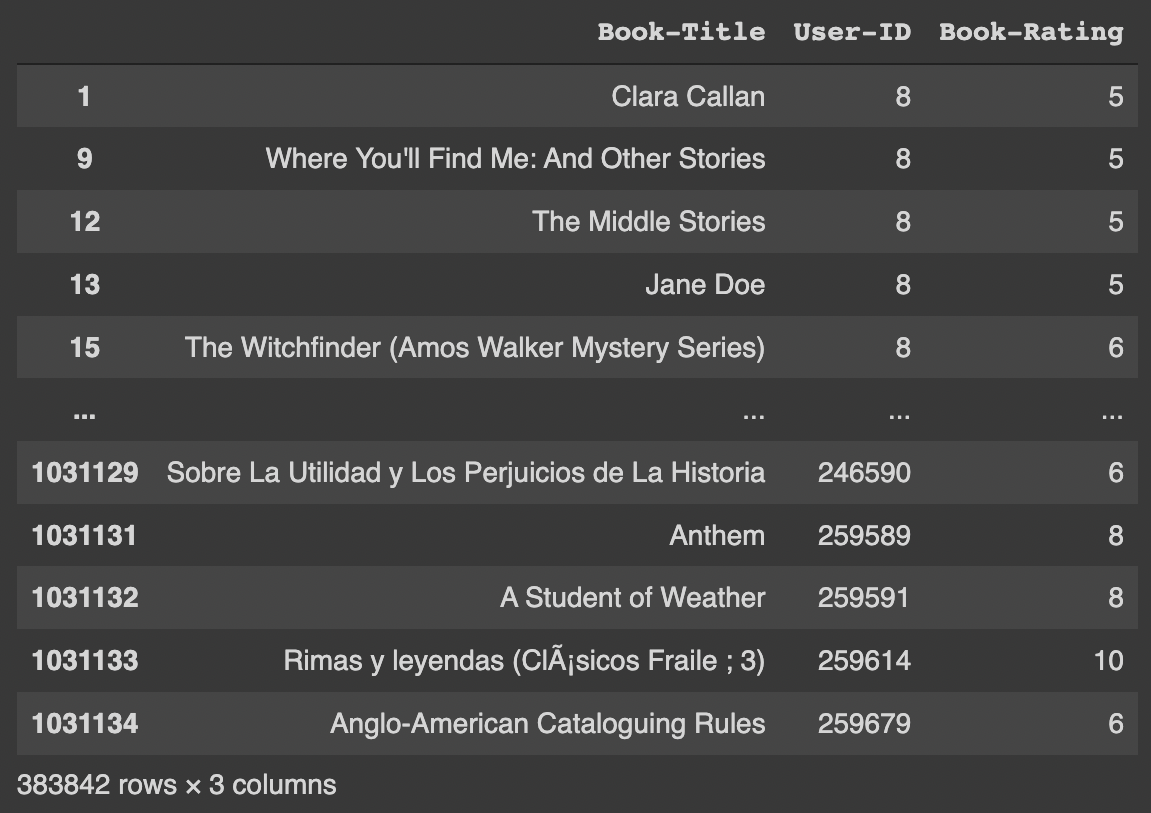
Also, in the dataset, there are book-ratings with value 0. However, in this dataset, 0 rating means no ratings. Therefore, we decided to remove the ratings with value 0.

# Remove unnecessary Columns (ex: ISBN, Book-Author, Year-Of-Publication, Publisher Image-URL-S, Image-URL-M, Image-URL-L, Location, Age)

newData = newData.drop(['ISBN','Book-Author', 'Year-Of-Publication','Publisher','Image-URL-S','Image-URL-M','Image-URL-L','Location','Age'], axis = 1)

# Remove 0 ratings representing no ratings.

newData = newData[newData['Book-Rating'] > 0]



**Compressing Large Data and Computing Validation**

Since the datasets we are using are very huge in size, there might be a runtime-error when computing the recommendations. In order to avoid those troubles, we tried to compress and only include the data with a certain number of ratings. In our case, we chose to make thresholds with 200 ratings, 100 ratings, and 50 ratings.

Then, we computed the validations using “Root-mean-square error” (RMSE).

The Kth-Nearest-Neighborhood algorithm with centered cosine similarity was used.

For ratings over 50, we got RMSE of 1.186.

For ratings over 100, we got RMSE of 1.0892

For ratings over 150, we got RMSE of 0.9430

Since the dataset with ratings over 150 computed the most valid RMSE value, we decided to make the recommendations out of this dataset.

**num\_ratings = pd.DataFrame(newData['Book-Title'].value\_counts())**

**invalid\_data\_1 = num\_ratings[num\_ratings['Book-Title'] <= 200].index**

**invalid\_data\_2 = num\_ratings[num\_ratings['Book-Title'] <= 100].index**

**invalid\_data\_3 = num\_ratings[num\_ratings['Book-Title'] <= 50].index**

**valid\_data\_1 = newData[~newData["Book-Title"].isin(invalid\_data\_1)]**

**valid\_data\_2 = newData[~newData["Book-Title"].isin(invalid\_data\_2)]**

**valid\_data\_3 = newData[~newData["Book-Title"].isin(invalid\_data\_3)]**

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

**trainset\_1 = Dataset.load\_from\_df(valid\_data\_1[['User-ID','Book-Title','Book-Rating']], reader).build\_full\_trainset()**

**trainset\_2 = Dataset.load\_from\_df(valid\_data\_2[['User-ID','Book-Title','Book-Rating']], reader).build\_full\_trainset()**

**trainset\_3 = Dataset.load\_from\_df(valid\_data\_3[['User-ID','Book-Title','Book-Rating']], reader).build\_full\_trainset()**

**sim\_options = {'name': 'cosine',**

**'user\_based': False # compute similarities between items**

**}**

**algo = KNNBasic(sim\_options=sim\_options)**

**algo.fit(trainset\_1)**

**trainset\_test\_1 = trainset\_1.build\_testset()**

**trainset\_predictions\_1 = algo.test(trainset\_test\_1)**

**algo.fit(trainset\_2)**

**trainset\_test\_2 = trainset\_2.build\_testset()**

**trainset\_predictions\_2 = algo.test(trainset\_test\_2)**

**algo.fit(trainset\_3)**

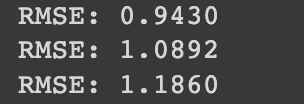
**trainset\_test\_3 = trainset\_3.build\_testset()**

**trainset\_predictions\_3 = algo.test(trainset\_test\_3)**

**accuracy.rmse(trainset\_predictions\_1, verbose= True)**

**accuracy.rmse(trainset\_predictions\_2, verbose= True)**

**accuracy.rmse(trainset\_predictions\_3, verbose= True)**

****

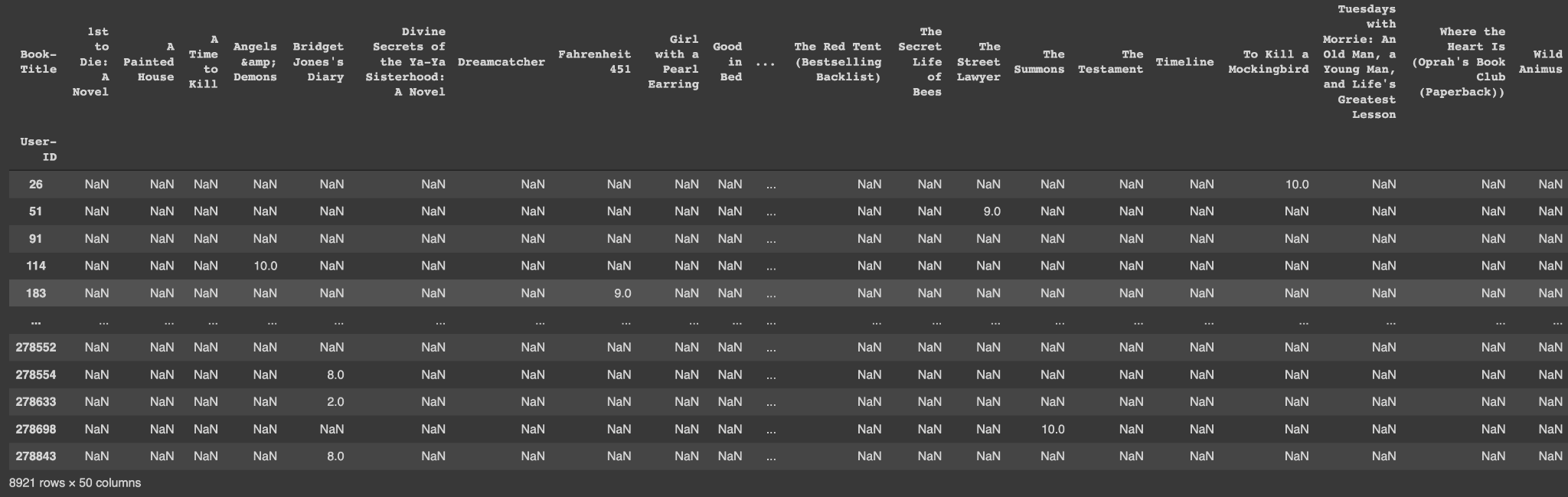
**Creating User-Item Matrix**

There are many functions to compute the similarity. Jaccard similarity, cosine similarity, and pearson correlation (centered similarity) are most well-known methods. However, pearson correlation computes the best outcome because it treats missing ratings as average ratings.In order to use pearson correlation to compute the similarity, we need a user-item matrix.

User-Id was used as a row, Book-title was used as a column, and book-ratings were used as a value.

**ui\_matrix = valid\_data\_1.pivot\_table(index = 'User-ID', columns = 'Book-Title', values = 'Book-Rating')**

**ui\_matrix**

****

**Get Recommendations**

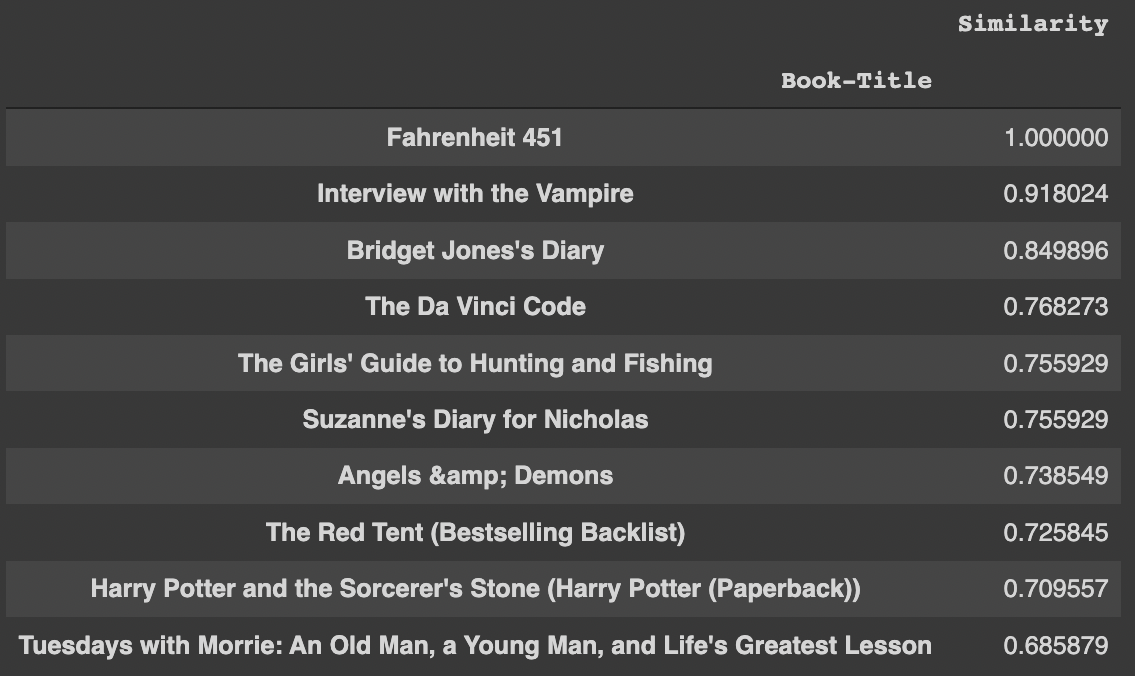
Using pearson correlation, we computed the similarity of the chosen book. In our case, we chose “Fahrenheit 451”. Similarities are from -1 to 1. The top 10 books with similarity values close to 1 are recommended. The book with low similarity value represents that the book is not similar to the chosen book.

**chosen\_item = ui\_matrix["Fahrenheit 451"]**

**find\_similarity = ui\_matrix.corrwith(chosen\_item, method="pearson")**

**recommendation = pd.DataFrame(find\_similarity, columns = ['Similarity'])**

**recommendation.sort\_values(by = 'Similarity', ascending=False).head(10)**

****

Referenced: <https://www.kaggle.com/code/mehmetcanyldrm/item-based-book-recommendation-engine>

# **3. Latent Factor-Based**

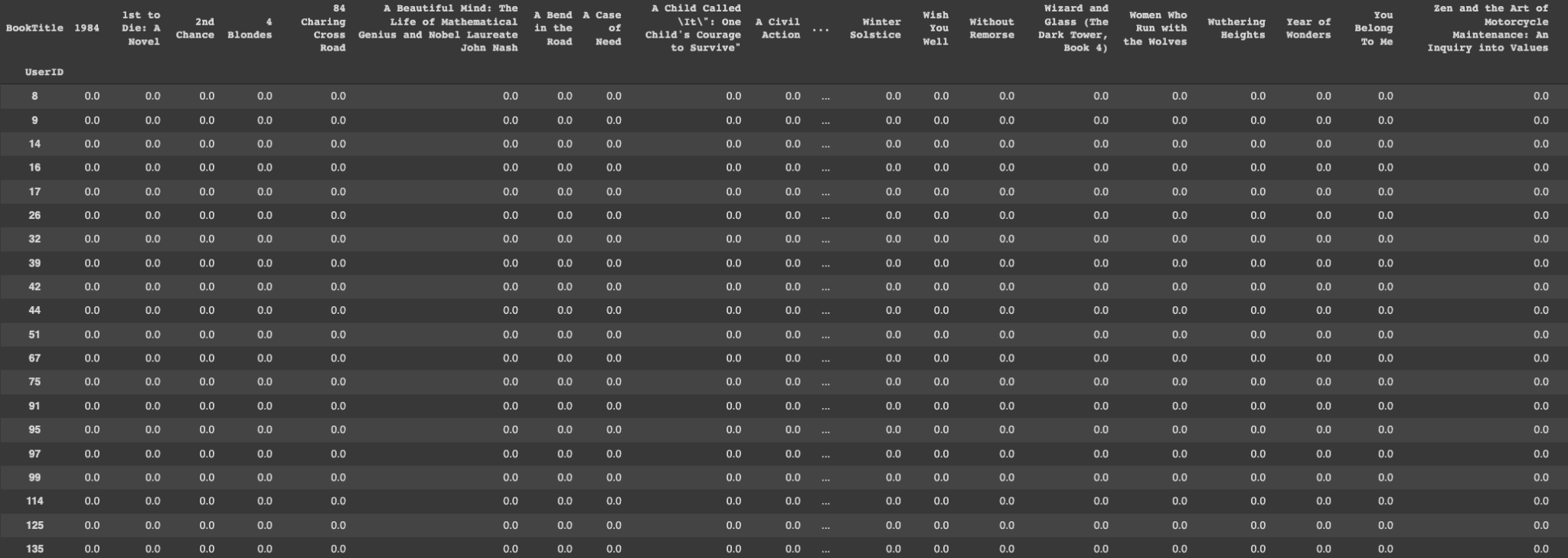
As in the Collaborative Filtering model, this model also dropped books that has less than 120 ratings.

1. Fill n/a values with 0

user\_book\_rating = ratings\_reduced.pivot\_table(index='UserID', columns = 'BookTitle', values = 'BookRating').fillna(0)

print(user\_book\_rating.shape)

user\_book\_rating.head(20)



This is the matrix that has user as an index and book titles as columns, and values for book ratings. Books that is not rated by certain users are filled with 0s.

1. Make a matrix so that our system can predict hidden (but known) ratings <https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.coo_matrix.html>

from scipy.sparse import coo\_matrix

R = coo\_matrix(user\_book\_rating.values)

print("R shape: ", R.shape)

R shape: (35487, 693)

1. Now, we have to do SVD for the matrix R. However, since there are missing entries, we cannot perform SVD.

Therefore, we randomly initialize matrix P and Q

Q’s number of rows should be the same as the number of books

P’s number of cols should be the same as the number of users

According to the following source (<https://towardsdatascience.com/introduction-to-latent-matrix-factorization-recommender-systems-8dfc63b94875>), the number of factors should be 10-250. For faster calculation, let’s make the number of factors as 10.

# Now, we have to do SVD from the matrix R. However, since there are missing entries, we cannot perform SVD.

# Randomly initialize Matrix P and Q (https://towardsdatascience.com/introduction-to-latent-matrix-factorization-recommender-systems-8dfc63b94875)

# Q's row should be same as the number of books

# P's cols should be same as the number of users

# number of factors is 10 - 250. we will choose 10

num\_factor = 10

P = np.random.normal(0,0.1, size=(R.shape[0], num\_factor))

Q = np.random.normal(0,0.1, size = (num\_factor, R.shape[1]))

1. Let’s calculate the error between the real ratings and random-ratings made from randomly created P and Q.

from numpy.linalg import norm

ratings = R.data

rows = R.row

cols = R.col

error = 0

print(ratings)

for rating\_index in range(len(ratings)):

rating = ratings[rating\_index]

user = rows[rating\_index]

book = cols[rating\_index]

if rating > 0:

# real rating - estimated rating

real\_minus\_est = rating-np.dot(P[user,:],Q[:,book])

error += real\_minus\_est\*\*2

print("error: ", error)

rmse = np.sqrt(error/len(R.data))

print("RMSE:", rmse)

error: 4189403.8543359973

RMSE: 7.962226443843482

1. Use Stochastic Gradient Descent to reduce the error and get the predicted ratings. <https://albertauyeung.github.io/2017/04/23/python-matrix-factorization.html/>

We have included regularization in order to avoid overfitting on training dataset created with coo\_matrix

# Stochastic Gradient Descent

# Now, using Stochastic Gradient Descent, we will try to reduce the error

# https://albertauyeung.github.io/2017/04/23/python-matrix-factorization.html/

steps = 100

lamb = 0.01 # regularization parameter

learning\_rate = 0.005

for step in range(steps):

if step%100 == 0: print("step: ", step)

for rating\_index in range(len(ratings)):

rating = R.data[rating\_index]

user = rows[rating\_index]

book = cols[rating\_index]

if rating > 0:

real\_minus\_est = rating - np.dot(P[user,:], Q[:,book])

# update user and book latent factor matrices

P[user,:] += learning\_rate\*(real\_minus\_est\*Q[:,book] - lamb\*P[user,:])

Q[:,book] += learning\_rate\*(real\_minus\_est\*P[user,:] - lamb\*Q[:,book])

# calc error after SGD

error = 0

for rating\_index in range(len(ratings)):

rating = ratings[rating\_index]

user = rows[rating\_index]

book = cols[rating\_index]

if rating > 0:

# real rating - estimated rating

real\_minus\_est = rating-np.dot(P[user,:],Q[:,book])

error += real\_minus\_est\*\*2

print("error: ", error)

rmse = np.sqrt(error/len(R.data))

print("RMSE:", rmse)

error: 12550.725010220038

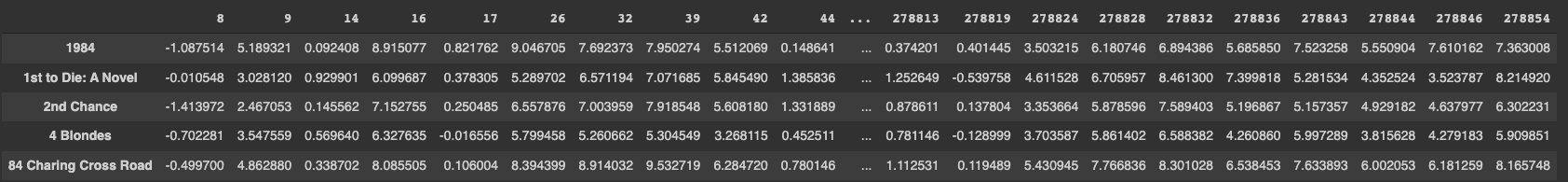
RMSE: 0.4358056112863486

predicted\_ratings = np.matmul(P, Q)

pred\_rat = pd.DataFrame(predicted\_ratings, columns = books\_in\_ratings, index = users\_in\_ratings)

pred\_rat = pred\_rat.transpose()

pred\_rat.head()



Now, we have made the matrix that predicts certain user’s ratings on certain books. Columns are users, and rows are book titles.

1. Put in user ID, and get the top 10 recommended books

#find user id 87's top 10 recommended books

top\_books = pred\_rat[32].sort\_values(ascending=False)[:10]

rec\_book\_names = []

rank = 1

for title in top\_books.keys():

print(rank, title)

print("-------------")

rank += 1

1 Seabiscuit

-------------

2 The Secret Garden

-------------

3 84 Charing Cross Road

-------------

4 To Kill a Mockingbird

-------------

5 East of Eden (Oprah's Book Club)

-------------

6 A Prayer for Owen Meany

-------------

7 The Princess Bride: S Morgenstern's Classic Tale of True Love and High Adventure

-------------

8 Holes (Yearling Newbery)

-------------

9 The Mists of Avalon

-------------

10 The Amber Spyglass (His Dark Materials, Book 3)

-------------

# 

# **Conclusion**

We aimed to implement a book recommendation system utilizing all three recommender system approaches discussed in the lecture: content-based, collaborative filtering, and latent factor-based.

The challenge of implementing this book recommendation system is having no dataset on book genre and book description. However, given just Book-Title data we aimed to model an effective book recommendation system using the three approaches.

The collaborative filtering approach achieved the RMSE score of 0.943. And the latent-based approach achieved the RMSE score of 7.96 and 0.43 after SGD.

# 

# **Business Applications**

**How your analyses would be implemented in a live system. (ie. A personal recommendation system or a tweetbot).**

Just like other recommendation systems, this model can also be used in e-commerce companies such as Amazon. Recommendation systems with high accuracy can be led to higher customer satisfaction, which will make those customers to use certain e-commerce again.

**When would your model learn new parameters?**

Since the datasets are massive, in order to change the output of the recommendations, we need to gain new massive data. If there’s not sufficient data, the output won’t change. Also, it is a waste of cost to update parameters every time there is new data.

**Describe in detail the pipeline from data ingestion to the end-user experience.**

To maximize the usage of these recommendation systems, a large amount of data is required. Therefore, companies and businesses should encourage users to rate their books as much as possible. They can consider giving compensation (such as gifts or discount coupons) if users rate. As more and more data is accumulated, the model will become more accurate, which will lead to a better user experience.