AI Book Recommender: Get Book Recommendations using AI

SANTHOSH KUMAR

Date - 31th August 2023

ABSTRACT

Users can use book recommendation systems to search and select books from a number of options available on the web or elsewhere electronic sources. They give the user a little bit selection of products that fit the description, given a large group of objects and a description of the user needs. Our system will simply provide recommendations. Recommendations are based on previous user activity, such as selection of genre and rating. The choice of book of user is tracked and data is collected. These systems gain lot of interest. In the proposed system, we have a big problem: when the user buys book, we want to recommend some books that the user can enjoy. Buyers also have a great deal of options when it comes to recommending the best and most appropriate books for them. User development privacy while placing small and minor losses of accuracy recommendations. The proposed recommendation system will provide user's ability to view and search the publications and using the methods like basic recommender, content-based filtering and collaborative filtering.

1. Problem Statement

In the ever-expanding landscape of literature, selecting the right book to read has become a complex and overwhelming task for many readers. With countless genres, themes, and authors to choose from, there is a growing need for an intelligent and efficient book recommendation system that can assist users in discovering books that align with their preferences. This project aims to address this challenge by developing a book recommendation system powered by machine learning techniques.

The primary problem to be tackled is how to leverage available book-related data, user behavior, and preferences to create a personalized book recommendation engine. This system should consider factors such as genre affinity, author styles, and historical reading patterns to generate tailored recommendations that match each user's individual taste.

Moreover, the challenge lies in implementing a mechanism that dynamically adapts to users' evolving preferences over time and provides explanations for its suggestions, fostering a deeper understanding of the recommendations. To create an effective book recommendation system, the project needs to overcome the complexities of processing and analyzing vast amounts of textual data, constructing accurate user profiles, and designing algorithms that can strike a balance between diversity and relevance in recommendations.

The system must be intuitive, user-friendly, and capable of making recommendations that enhance the overall reading experience while catering to the diverse reading preferences of users. This project seeks to build a book recommendation system that not only simplifies the process of book selection but also fosters engagement and satisfaction among readers.

2. Market/Customer/Business Need Assessment

2.1 Market Demand

The book industry is undergoing a digital transformation, with readers increasingly relying on online platforms and e-commerce for book purchases. However, with the abundance of choices available, customers often face difficulties in selecting books that match their interests and preferences. This gap in the market presents a significant opportunity for a robust and accurate book recommendation system.

Market Need: Information Overload: As the number of books available grows, readers find it challenging to navigate through the vast selection and make informed choices. **Personalization**: Readers desire a personalized experience that caters to their unique tastes and preferences, helping them discover books that resonate with them. Time **Efficiency**: An efficient recommendation system can save users time by narrowing down options, leading to higher customer satisfaction.

2.2 Customer Preferences

Readers want to explore new books that align with their interests, expanding their reading horizons beyond familiar authors or genres. Customers seek guidance in finding books that match their mood, current interests, and reading goals. Readers are more likely to stay engaged with a platform that offers tailored book recommendations, fostering a sense of connection and trust.

2.3 Business Opportunities

An effective recommendation system enhances customer satisfaction, leading to higher retention rates and repeat business.Implementing a sophisticated book recommendation system can set a business apart in a competitive market, attracting more users and generating positive word-of-mouth. Businesses can leverage user data to gain insights into reading trends, preferences, and popular genres, informing inventory management and marketing strategies. Recommendation systems can be used strategically to cross-sell related books or upsell premium content, increasing revenue potential.

3. Target Specifications and Characterization

3.1 Target Specifications

Designing a successful Book Recommendation System powered by machine learning requires defining clear target specifications to guide its development. These specifications ensure that the system meets the needs of users and delivers accurate, personalized recommendations. Here are the target specifications

Personalization: Objective: Generate personalized book recommendations based on user preferences. Specifications: Develop algorithms that analyze user behavior, reading history, ratings, and interactions to tailor recommendations to individual tastes.

Accuracy and Diversity: Objective: Provide accurate recommendations while promoting diverse book options. Specifications: Utilize collaborative filtering, content-based analysis, or hybrid approaches to balance precise suggestions with introducing users to new genres and authors.

Real-time Updates: Objective: Offer timely suggestions by incorporating recent releases and trends. Specifications: Implement a mechanism that tracks and integrates new book releases and emerging literary trends into the recommendation pool.

User Interaction: Objective: Facilitate user engagement and feedback for refining recommendations. Specifications: Design an intuitive interface allowing users to rate, review, and mark favorite books, contributing to the system's learning process.

Explainability: Objective: Provide clear explanations for recommended book choices. Specifications: Incorporate a feature that highlights common themes, genres, or user preferences that led to a specific recommendation, enhancing user understanding.

Adaptability and Learning: Objective: Continuously learn from user interactions to enhance future recommendations. Specifications: Integrate machine learning models that dynamically adjust recommendations based on user feedback, evolving preferences, and changing trends.

Scalability and Performance: Objective: Ensure the system remains responsive and efficient as user data and activity grow. Specifications: Employ scalable algorithms and optimize data processing to maintain fast response times, even with a growing user base.

Privacy and Security: Objective: Protect user data and ensure privacy in the recommendation process. Specifications: Implement robust data encryption, anonymization techniques, and user-controlled data sharing to safeguard sensitive information.

3.2 Customer Characterization

Book Enthusiasts: Description: These are avid readers who are passionate about discovering new books and authors. Characteristics: They value personalized recommendations that align with their diverse reading interests and appreciate the convenience of discovering hidden gems.

Exploratory Readers: Description: Individuals who enjoy exploring different genres and are open to trying new authors. Characteristics: They are interested in recommendations that introduce them to genres they might not have considered, expanding their reading horizons.

Busy Professionals: Description: Professionals with limited leisure time seeking efficient ways to find books that resonate with them. Characteristics: They prefer accurate recommendations that match their preferences quickly, as they have less time for extensive research.

Casual Readers: Description: Individuals who read occasionally for leisure and seek recommendations for light and engaging reads. Characteristics: They appreciate suggestions for enjoyable and easy-to-read books, making their sporadic reading experiences more enjoyable.

Young Adult Audience: Description: A younger demographic interested in diverse genres, including YA fiction, fantasy, and contemporary literature. Characteristics: They look for recommendations that cater to their age group and interests, often relying on peer reviews and trending titles.

Book Club Members: Description: Members of book clubs who require suggestions for group reading choices. Characteristics: They seek recommendations that facilitate group discussions and encourage participants to explore various themes and genres.

Gift Shoppers: Description: Individuals seeking book recommendations as gifts for friends and family. Characteristics: They value recommendations that match the recipient's preferences, making the gift more meaningful.

Literary Enthusiasts: Description: Those interested in literary analysis, classics, and deeper thematic exploration. Characteristics: They appreciate recommendations that align with their interest in thought-provoking and intellectually stimulating literature.

4 External Search

- **Book Recommendation System**
- **AI based Book Recommender System with Hybrid Approach**
- **A Survey Report on Book Recommendation Techniques**
- **Enhanced Book Recommendation System using Collaborative Filtering**
- **BookBarn: Web Based Book Recommendation and E-Commerce System**

5 Benchmarking

Bookreads and Kindle likely have access to extensive user interaction data, given their large user bases. They can leverage this data to improve recommendation accuracy. My system focused on collecting and curating high-quality data to enhance your system performance. It too can emphasize personalized recommendations, utilizing user profiles, and considering individual user preferences for enhanced personalization. It need to be intuitive and visually appealing interface that enhances the user experience and encourage engagement. It will be adaptable by implementing real-time learning mechanisms that improve recommendations over time. Ensuring efficient performance and scalability as it grows in popularity and user engagement. The system helps even to understand the pain points guiding to enhance system features.

6 Applicable Patents

Patent 1 - E-book reading plan recommendation system and method

The first patent is all about acquiring personal information of a target user and a book name of an electronic book currently read by the target user; constructing a similar user group according to the personal information and the book title, and acquiring reading characteristics of other users in the similar user group; reading characteristics of other users in the same type of user group are obtained, a first recommended reading plan is generated, and the first recommended reading plan is recommended to a target user; obtaining historical reading information of a target user accessing a reading book, and counting the reading behavior characteristics of the target user according to the historical reading information; and adjusting the first recommended reading plan according to the reading behavior characteristics, generating a second recommended reading plan, and recommending the second recommended reading plan to the target user. All data are based on the basic information, the reading behavior information and the personalized information of the user without the setting of the user, and meet the personalized requirements of the user.

Patent 2 - A kind of books personalized recommendation method and system towards libraries of the universities

The second patent here discloses a kind of books personalized recommendation towards libraries of the universities, solve the problems, such as that mass data storage and inquiry, scalability and recommendation effect are poor in the existing book recommendation algorithm in libraries of the universities, its basic ideas is as follows: first by library reader and books etc. be used as node, construct graph model. Secondly, converting reader-books classification preference matrix for the operation log file of reader and reader's personal information matrix calculates similarity between reader together, and these operations are constructed with the information excavated as side and are associated with map. Secondly, association map and spectral clustering are combined, a kind of new books Personalization recommendation model is proposed, the class cluster distribution about reader is calculated. Finally, calculating Recommended Books list according to collaborative filtering in the corresponding class cluster of the reader when needing to carry out book recommendation.

7 Applicable Regulations

- General Data Protection Regulation (GDPR)
- California Consumer Privacy Act (CCPA)
- Consumer Protection
- Digital Service Regulations
- E-commerce Regulations
- Patents on ML Algorithms created
- Ensure algorithms are reviewed by the open source and academic communities

8 Applicable Constraints

- Requires high computational power
- Handling high-user volumes and user interactions
- Recommending books for new users who lack historical interaction data can be challenging

- Developing high quality system can be time consuming
- Limited data can lead to suboptimal recommendations

9 Business Model (Monetization Idea)

I want to use the opportunity to incorporate the system in bookstores and libraries in schools. This will help the children to inculcate reading habits and foster the curiosity of a student. Coming to small scale size, it is focused on developing a partnership with online bookstores and publishers. Students from age 10-20 need to have extensive reading to have a successful life. It purely depends on the interest of the genre they read. Some like fiction and others non-fiction. To reduce their time in searching for books, this will gather some information by collecting the genre of user and give a recommendation list of numbers 5, 10 or 20.

10 Concept Generation

The concept for the system came from relevancy. There are two types of recommender system-personalized and non-personalized system. The end results should have good relevancy and it should include different user interests(diverse). The recommendation system here focuses on eliminating the repetitive results(does not show the same results second time). Thus, the motive is to meet the expectations and likes of books of users.

11 Concept Development

Firstly we have to collect the data which can we done by webscrapping, collecting data from Kaggle (machine learning and data science community). We need to think about what we achieve by using data. The data will have a information like title,year,pages,description,genres,average_ratings,ratings_count,books_count and authors. After collecting the data, the next step is doing data cleaning which removes the column which are not important for model building and for the whole application. The EDA is done in by asking deep questions to know how the data performs with one another with checking the correlation of one variable with another variable. When it comes to model building, the three types of recommendation system is used in order to check how well the recommended list it shows and exploring the strengths and weaknesses of every types of recommendation system. Each type of recommendation model is dumped and used in streamlit on showing at frontend application.

12 Final Product Prototype and Schematic Diagram

The product is developed by integrating frontend with backend to show a recommendation of books

Backend(Exploratory Data Analysis and Model Building)

Firstly the dataset was originally scrapped from the Goodreads API and collected from github repository.

Dataset is imported and sufficient data cleaning is done like handling missing null values and handling duplicate values. This is done so that the product developed shows accurate results whereas lack of proper cleaning may lead to poor accuracy.

We need to understand when to use each model.

When we want to recommend a completly new user, we can use Simple Recommender. If the user want personalized recommendation, they can provide one book they liked. By applying a Content Based Filtering, instead of having to rate 30 books to start the recommendation engine, users can just pick only one book for Goodreads to provide good recommendations. We must pay attention to what content is chosen, whether only the author, the description etc.

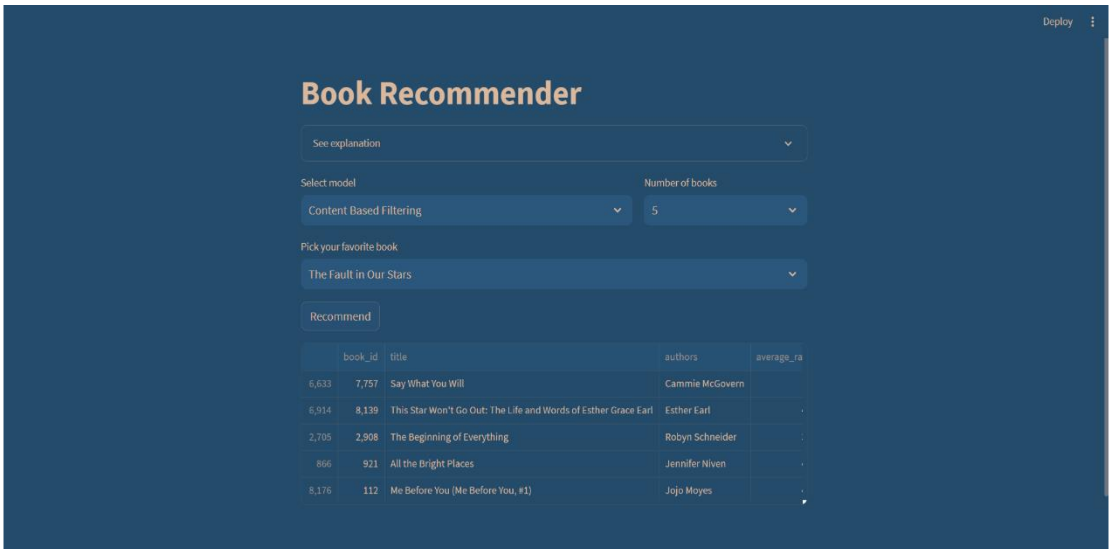
When they want even more diverse recommendation, we can use Collaborative Filtering. Here, we need other users data so that we can make the recommendation. For this method, I think SVD is suitable for making a recommendation system in Goodreads because of its lower RMSE, faster calculation and lower memory requirement unlike KNN. I think the RMSE value of 0.8 is still quite reasonable for a Goodreads rating.

Frontend:

The python module named Streamlit is used for frontend purpose. Streamlit is an open-source Python library that allows you to create interactive web applications for data science and machine learning projects with minimal effort. It's designed to turn data scripts into shareable web apps quickly.

The UI has a selection control where it asks for

- Which model to select
- Number of books
- Pick your favourite book (for Content and Collaborative Filtering)



Book Recommender

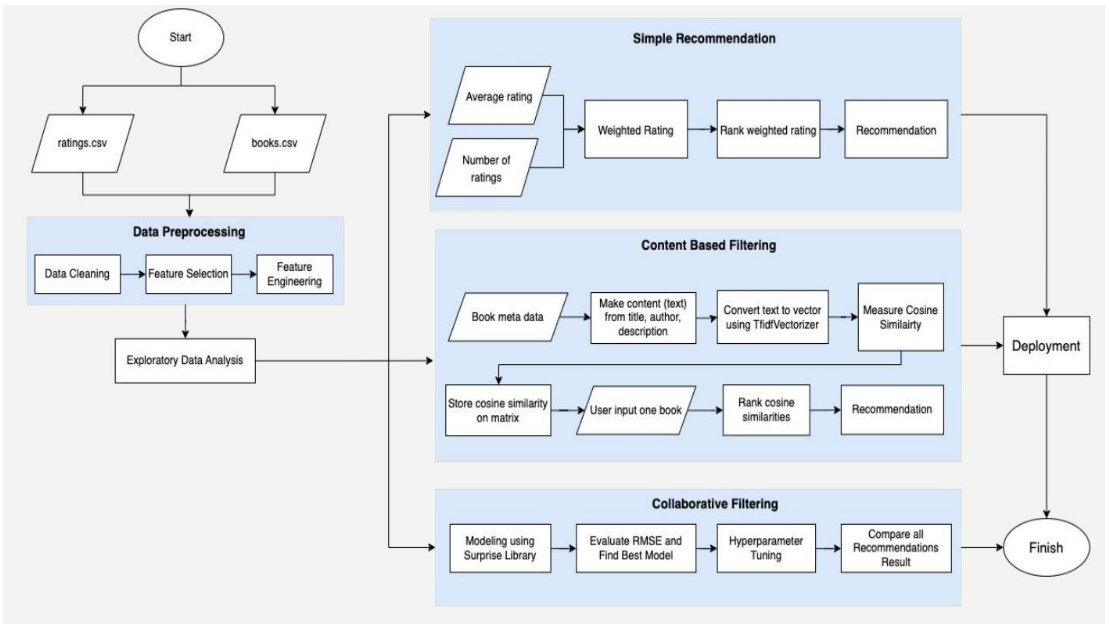
Select model: Content Based Filtering

Number of books: 5

Pick your favorite book: The Fault in Our Stars

	book_id	title	authors	average_ra
6,633	7,757	Say What You Will	Cammie McGovern	
6,914	8,139	This Star Won't Go Out: The Life and Words of Esther Grace Earl	Esther Earl	
2,705	2,908	The Beginning of Everything	Robyn Schneider	
866	921	All the Bright Places	Jennifer Niven	
8,176	112	Me Before You (Me Before You, #1)	Jojo Moyes	

Schematic Diagram :-



13 Product Details

How does it work:-

A book recommendation system works by analyzing user preferences and book characteristics to provide personalized suggestions to users. The system gathers data on books, including their titles, authors, genres, descriptions, and user interactions (such as ratings, reviews, and purchase history). This data forms the basis for generating recommendations. The collected data is cleaned, transformed, and organized into a suitable format for analysis. This step involves handling missing values, removing outliers, and standardizing data. The system creates user profiles based on their interactions with books. This includes the books they've rated, reviewed, or purchased. User profiles capture the user's preferences, interests, and behavior. For content-based recommendations, the system analyzes the content of the books and compares it to users' preferences. It identifies patterns in the features such as genre, author, and keywords that users have shown interest in. Books with similar content to those the user has liked are recommended. Collaborative filtering identifies patterns in user-item interactions. It groups users with similar preferences and recommends items (books) that similar users have liked. It also identifies items that are often liked by users who share a preference for a particular book. Matrix factorization techniques like Singular Value Decomposition (SVD) and SVD++ decompose the user-item interaction matrix into latent factors. These factors represent hidden features that influence user preferences and item characteristics. By predicting missing values in the matrix, the system generates recommendations. In k-NN, the system identifies the "k" most similar users to a target user based on their preferences. It then recommends items that these similar users have liked but the target user hasn't interacted. Many modern recommendation systems combine multiple techniques to overcome individual limitations. Hybrid approaches can leverage the strengths of content-based filtering and collaborative filtering to provide more accurate and diverse recommendations. The system evaluates recommendation quality using metrics like precision, recall, and user engagement. User feedback and interactions are collected to continuously improve the recommendations over time. As users provide more feedback and interact with the system, their preferences evolve. The recommendation system continually adapts and updates recommendations based on these changing preferences. Overall, a book recommendation system uses advanced algorithms and techniques to understand user behavior, book attributes, and patterns of preference, ultimately providing users with relevant and engaging book suggestions.

Data Source:-

This dataset was originally scrapped from the Goodreads API which can be even collected from Github repository.

Modeling:-

Simple Recommender

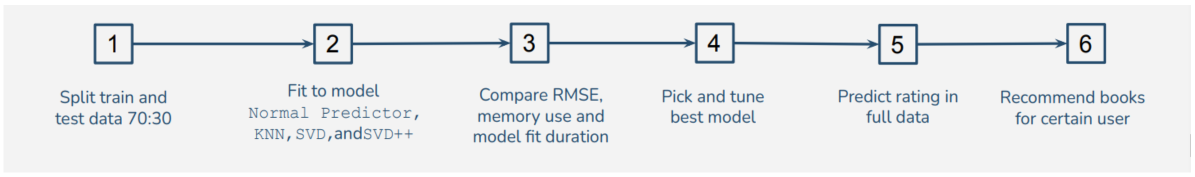
One of the easiest way to give recommendation is to rank the book based on rating (average_rating) or ratings_count. However in EDA, we need to make a weighted rating of average_rating and rating_count.

Content Based Recommendation

This approach makes recommendations to users based on the features or characteristics of the books. Using item metadata, the computer will assess how similar the books are to one another and then recommend the books that are most like the one the user loved. One of the way is using cosine similarity.

Collaborative Filtering Recommendation

This approach recommends books to users based on their prior reading habits and the preferences of other users.


1
Split train and test data 70:30
2
Fit to model Normal Predictor, KNN, SVD, and SVD++
3
Compare RMSE, memory use and model fit duration
4
Pick and tune best model
5
Predict rating in full data
6
Recommend books for certain user

Algorithms:-

Normal Predictor
KNN (K-Nearest Neighbor)
SVD (Singular Value Decomposition)
SVD++

Environment or Software used:-

Pycharm
Google Colab

Frameworks or Libraries used:-

Pandas
Numpy
Seaborn
Matplotlib
Scipy
Surprise
Sklearn (for linear_kernel and TfidfVectorizer)
Math

Team required to develop:

To develop a book recommendation system, you will need a team with the following roles and skills:

Project Manager: Responsible for overall project planning, scheduling, and coordination. They ensure that the project stays on track, deadlines are met, and resources are effectively managed.

Data Scientist / Machine Learning Engineer: This role will be focused on designing and implementing the recommendation algorithm. They need expertise in machine learning, data analysis, and data preprocessing. They might work with collaborative filtering, content-based filtering, or hybrid approaches to generate recommendations.

Software Engineers: Backend and frontend developers are crucial to build the technical infrastructure of the recommendation system. Backend developers will create the APIs, databases, and server architecture, while frontend developers will design the user interface through which users interact with the system.

Database Specialist: This role involves designing and maintaining the database where book information, user preferences, and historical data are stored. Knowledge of database management systems (e.g., SQL, NoSQL) is important.

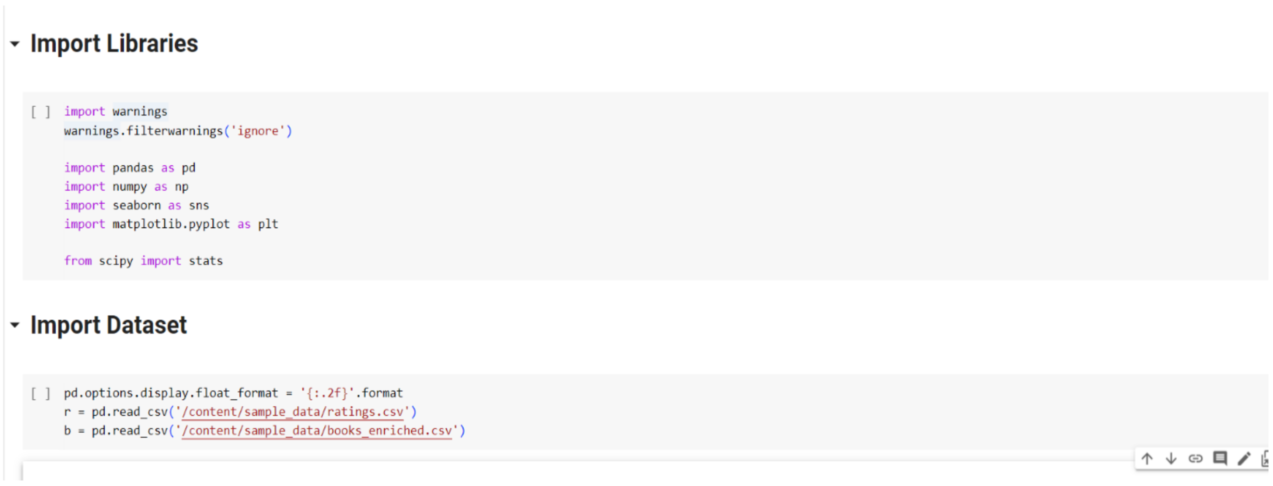
UI/UX Designer: Responsible for designing the user interface that presents book recommendations to users. A user-friendly and visually appealing design can significantly enhance user experience.

Quality Assurance (QA) Tester: QA testers will ensure that the system works as intended and is free of bugs or usability issues. They conduct thorough testing to identify and report any problems.

Data Engineer: Data engineers are responsible for the collection, integration, and preparation of data from various sources. They ensure that the data is in the right format for analysis and model training.

Code:-

ML Implentation:-

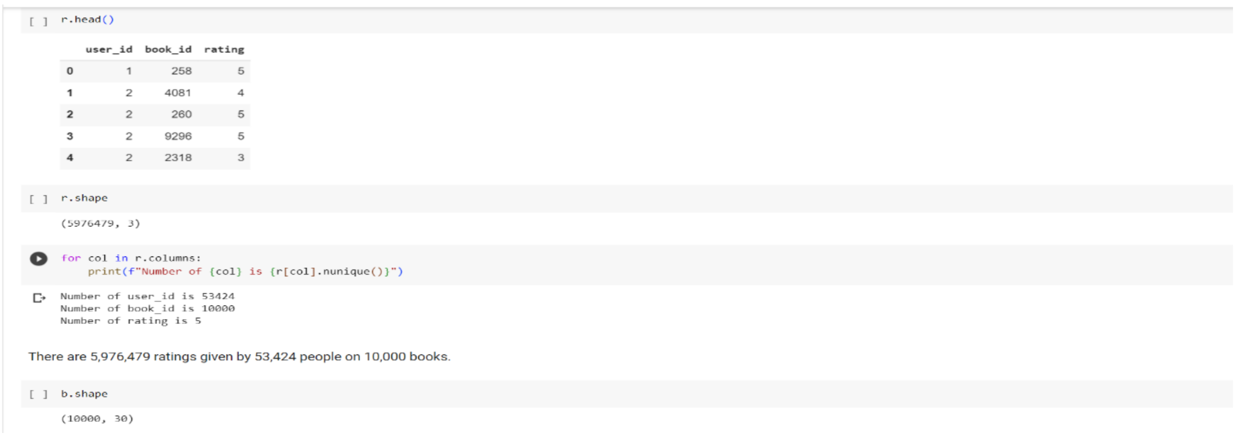


Import Libraries

```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from scipy import stats
```

Import Dataset

```
pd.options.display.float_format = '{:.2f}'.format
r = pd.read_csv('/content/sample_data/ratings.csv')
b = pd.read_csv('/content/sample_data/books_enriched.csv')
```



```
[ ] r.head()
```

	user_id	book_id	rating
0	1	258	5
1	2	4081	4
2	2	260	5
3	2	9296	5
4	2	2318	3

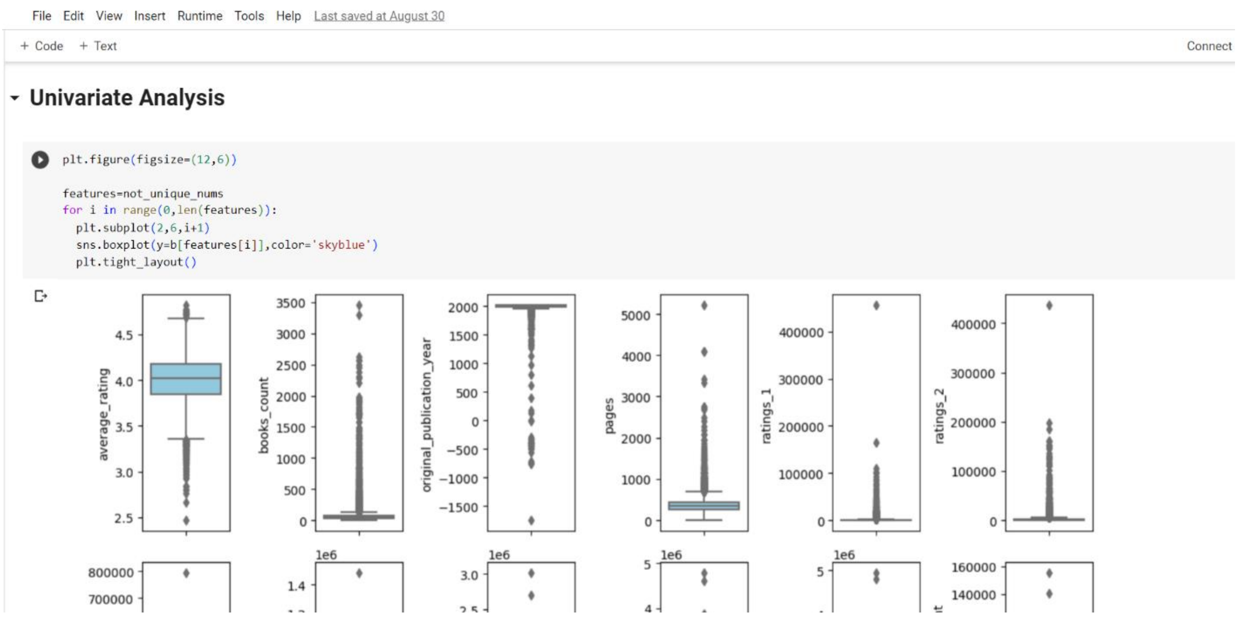
```
[ ] r.shape

(5976479, 3)
```

```
for col in r.columns:
    print(f"Number of {col} is {r[col].nunique()}")
```

```
Number of user_id is 53424
Number of book_id is 10000
Number of rating is 5
```

There are 5,976,479 ratings given by 53,424 people on 10,000 books.

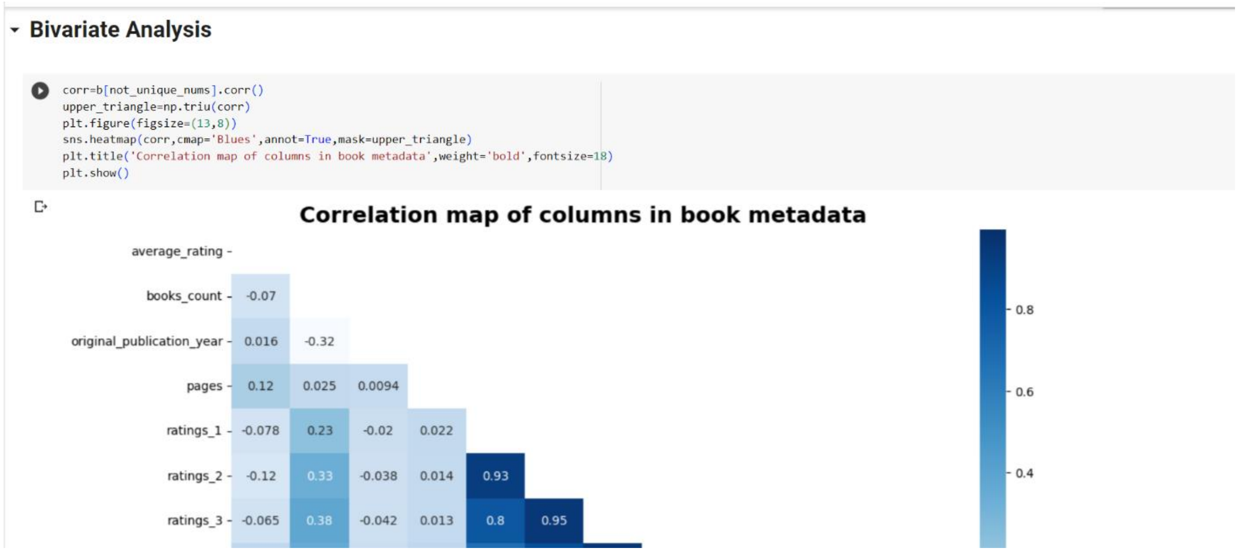
```
[ ] b.shape

(10000, 30)
```


File Edit View Insert Runtime Tools Help Last saved at August 30
+ Code + Text
Connect

Univariate Analysis

```
plt.figure(figsize=(12,6))

features=not_unique_nums
for i in range(0,len(features)):
  plt.subplot(2,6,i+1)
  sns.boxplot(y=b[features[i]],color='skyblue')
  plt.tight_layout()
```

average_rating
books_count
original_publication_year
pages
ratings_1
ratings_2
1e6

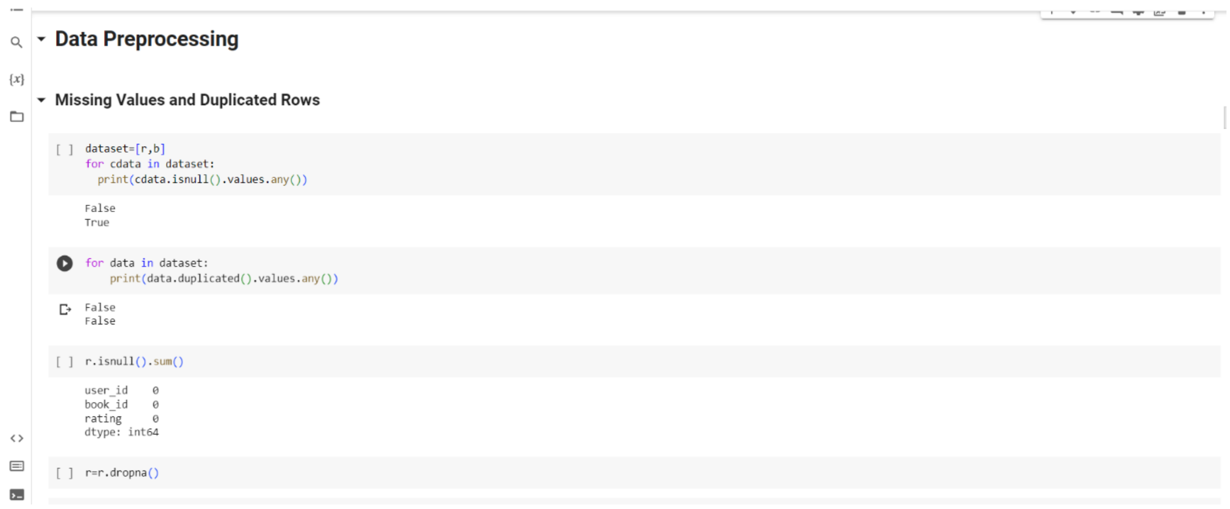


Bivariate Analysis

```
corr=b[not_unique_nums].corr()
upper_triangle=np.triu(corr)
plt.figure(figsize=(13,8))
sns.heatmap(corr,cmap='Blues',annot=True,mask=upper_triangle)
plt.title('Correlation map of columns in book metadata',weight='bold',fontsize=18)
plt.show()
```

Correlation map of columns in book metadata

average_rating						
books_count	-0.07					
original_publication_year	0.016	-0.32				
pages	0.12	0.025	0.0094			
ratings_1	-0.078	0.23	-0.02	0.022		
ratings_2	-0.12	0.33	-0.038	0.014	0.93	
ratings_3	-0.065	0.38	-0.042	0.013	0.8	0.95



Data Preprocessing

Missing Values and Duplicated Rows

```
dataset=[r,b]
for cdata in dataset:
  print(cdata.isnull().values.any())
```

```
False
True
```

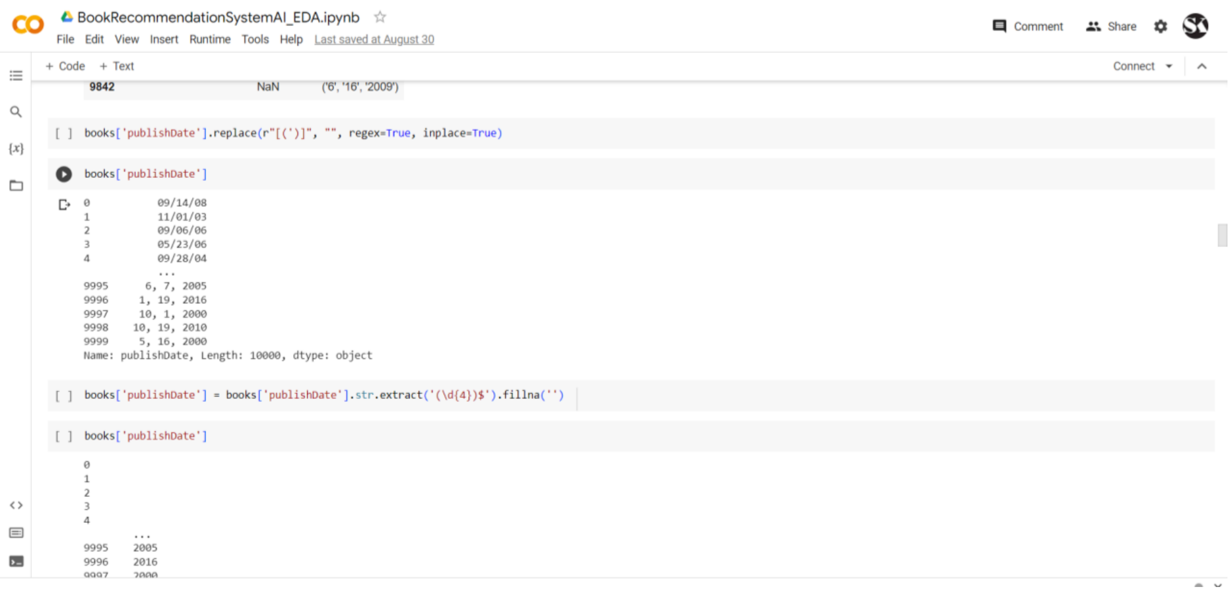
```
for data in dataset:
    print(data.duplicated().values.any())
```

```
False
False
```

```
r.isnull().sum()
```

```
user_id    0
book_id    0
rating     0
dtype: int64
```

```
r=r.dropna()
```



BookRecommendationSystemAI_EDA.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment Share
+ Code + Text
Connect

```
9842                NaN        ('6', '16', '2009')
```

```
books['publishDate'].replace(r"[(')]", "", regex=True, inplace=True)
```

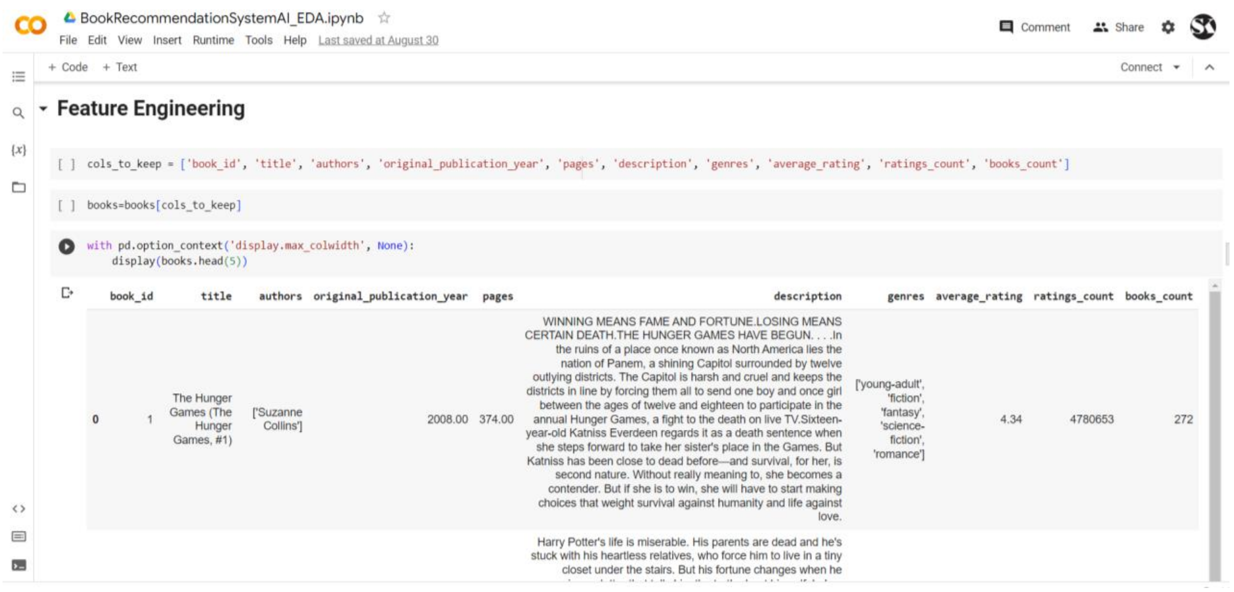
```
books['publishDate']
```

```
0           09/14/08
1           11/01/03
2           09/06/06
3           05/23/06
4           09/28/04
              ...
9995      6, 7, 2005
9996     1, 19, 2016
9997     10, 1, 2000
9998    10, 19, 2010
9999     5, 16, 2000
Name: publishDate, Length: 10000, dtype: object
```

```
books['publishDate'] = books['publishDate'].str.extract('(\d{4})$').fillna('')
```

```
books['publishDate']
```

```
0
1
2
3
4
        ...
9995    2005
9996    2016
9997    2000
```



BookRecommendationSystemAI_EDA.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment Share
+ Code + Text
Connect

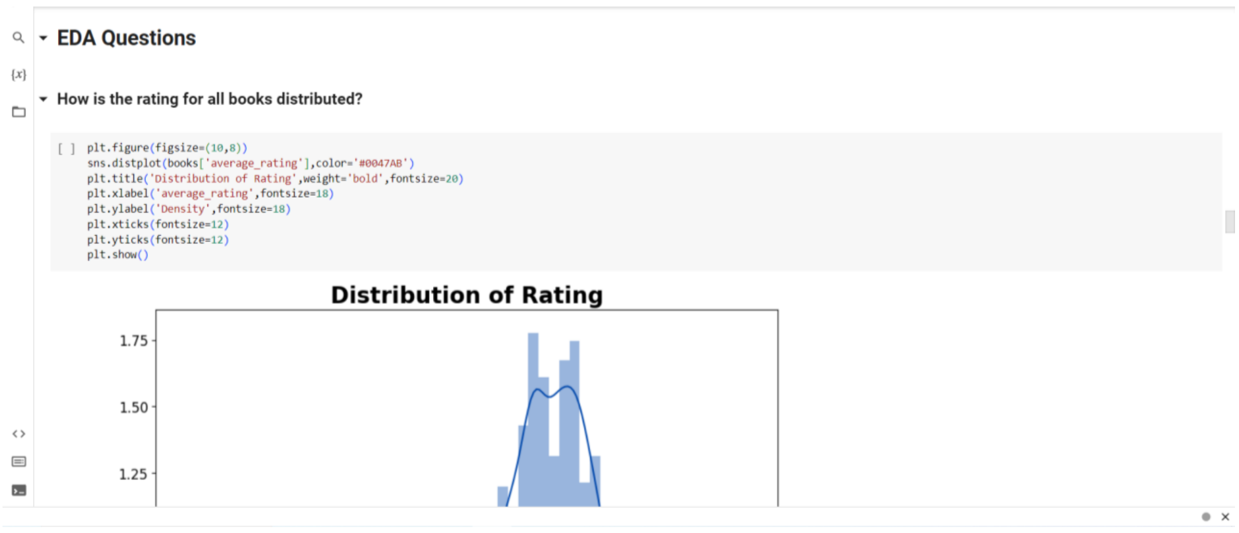
Feature Engineering

```
[ ] cols_to_keep = ['book_id', 'title', 'authors', 'original_publication_year', 'pages', 'description', 'genres', 'average_rating', 'ratings_count', 'books_count']
```

```
[ ] books=books[cols_to_keep]
```

```
with pd.option_context('display.max_colwidth', None):
    display(books.head(5))
```

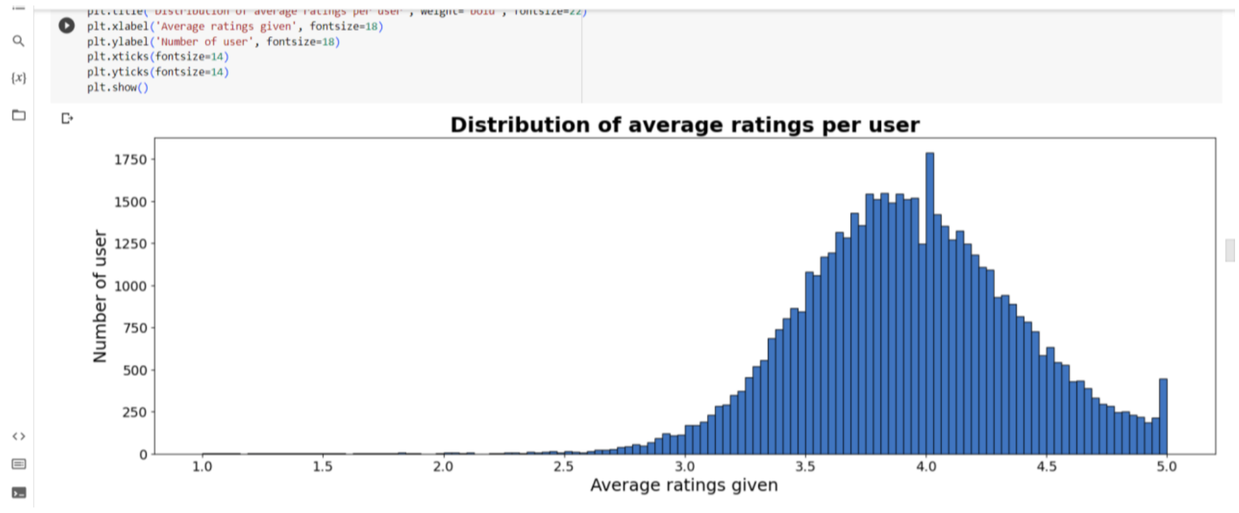
	book_id	title	authors	original_publication_year	pages	description	genres	average_rating	ratings_count	books_count
0	1	The Hunger Games (The Hunger Games, #1)	['Suzanne Collins']	2008.00	374.00	WINNING MEANS FAME AND FORTUNE.LOSING MEANS CERTAIN DEATH.THE HUNGER GAMES HAVE BEGUN. . . .In the ruins of a place once known as North America lies the nation of Panem, a shining Capitol surrounded by twelve outlying districts. The Capitol is harsh and cruel and keeps the districts in line by forcing them all to send one boy and once girl between the ages of twelve and eighteen to participate in the annual Hunger Games, a fight to the death on live TV.Sixteen-year-old Katniss Everdeen regards it as a death sentence when she steps forward to take her sister's place in the Games. But Katniss has been close to dead before—and survival, for her, is second nature. Without really meaning to, she becomes a contender. But if she is to win, she will have to start making choices that weight survival against humanity and life against love.	['young-adult', 'fiction', 'fantasy', 'science-fiction', 'romance']	4.34	4780653	272
						Harry Potter's life is miserable. His parents are dead and he's stuck with his heartless relatives, who force him to live in a tiny closet under the stairs. But his fortune changes when he				



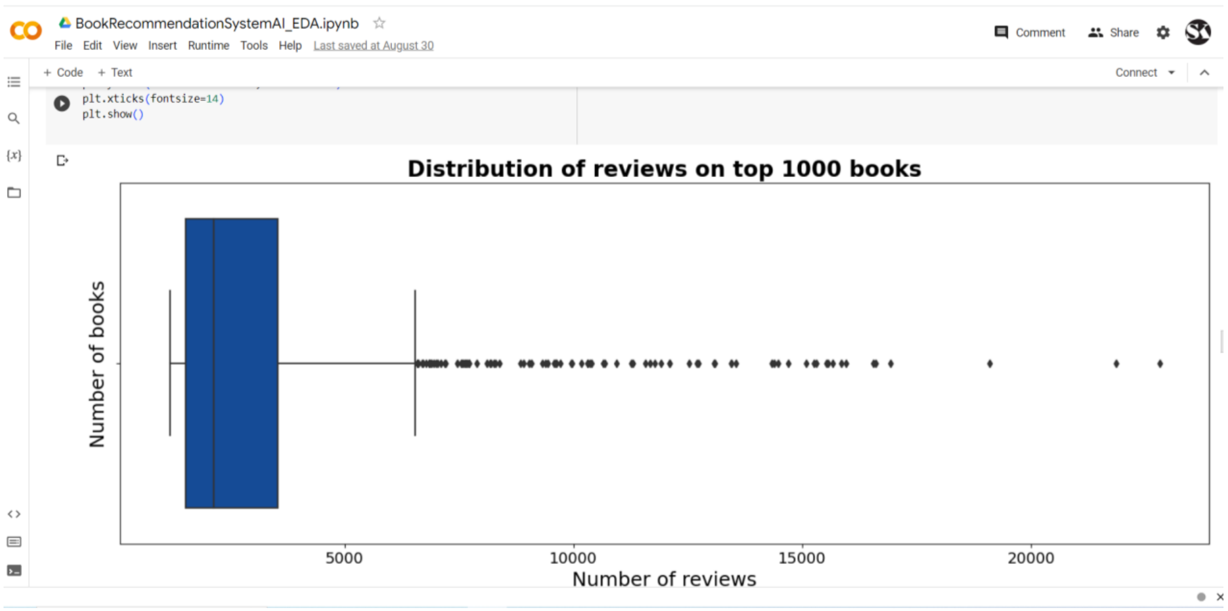
EDA Questions

How is the rating for all books distributed?

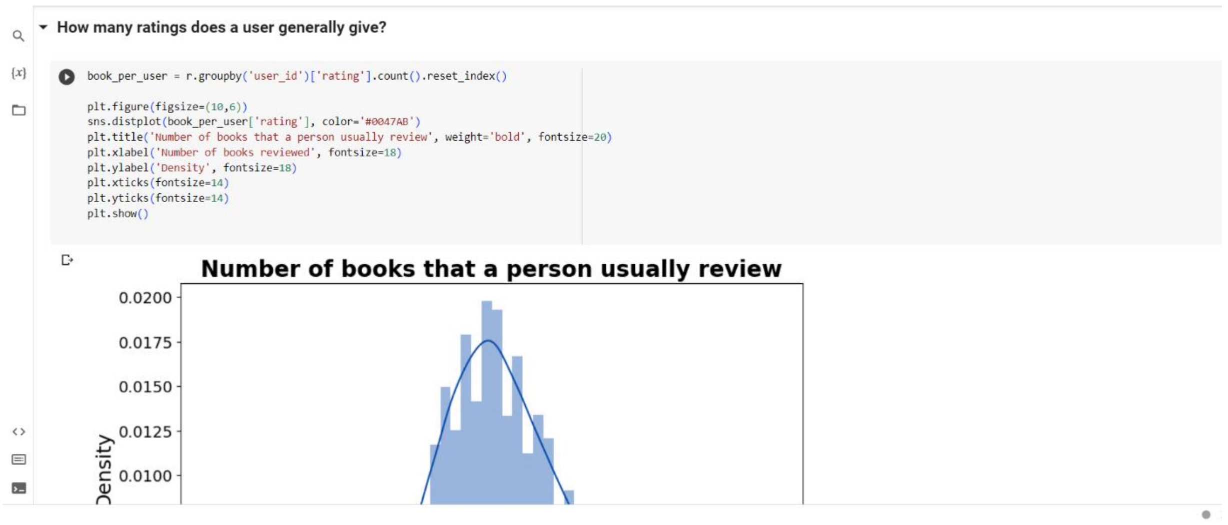
```
[ ] plt.figure(figsize=(10,8))
    sns.distplot(books['average_rating'],color='#0047AB')
    plt.title('Distribution of Rating',weight='bold',fontsize=20)
    plt.xlabel('average_rating',fontsize=18)
    plt.ylabel('Density',fontsize=18)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.show()
```



```
plt.title('Distribution of average ratings per user', weight='bold', fontsize=22)
plt.xlabel('Average ratings given', fontsize=18)
plt.ylabel('Number of user', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

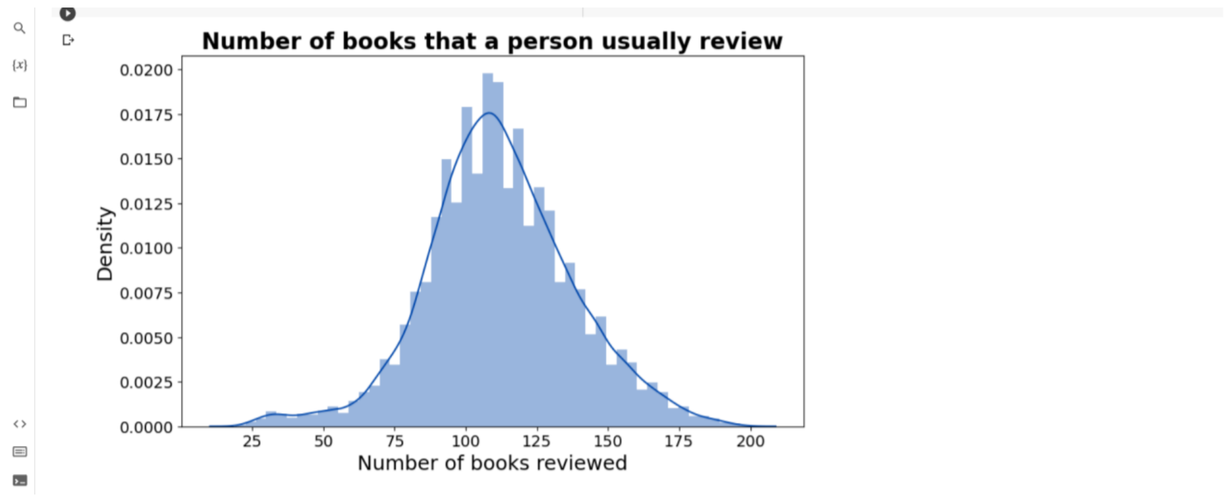

BookRecommendationSystemAI_EDA.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment
Share
+ Code + Text
Connect
Distribution of reviews on top 1000 books
Number of books
5000
10000
15000
20000
Number of reviews

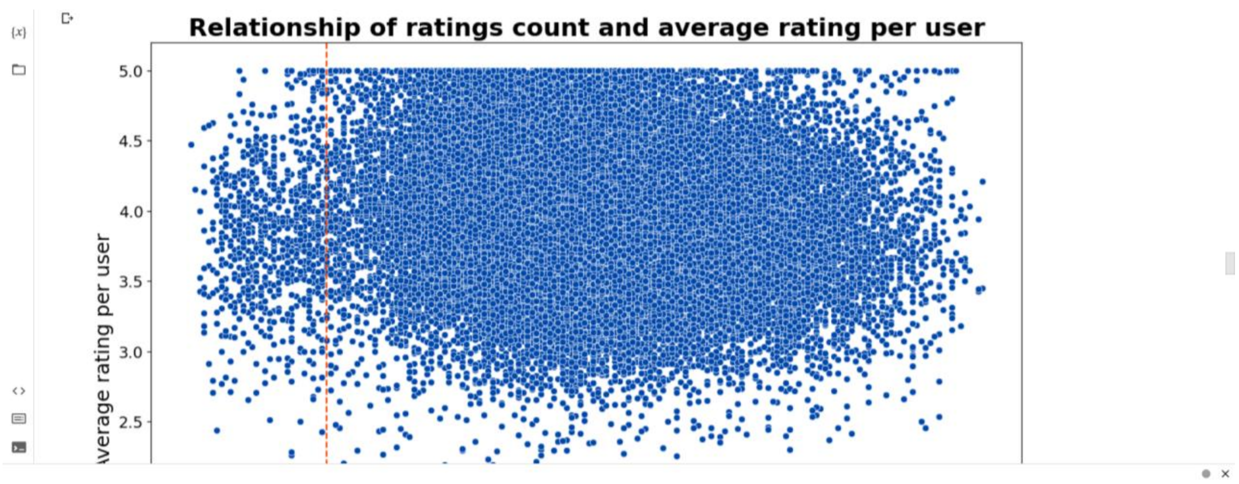
```
plt.xticks(fontsize=14)
plt.show()
```

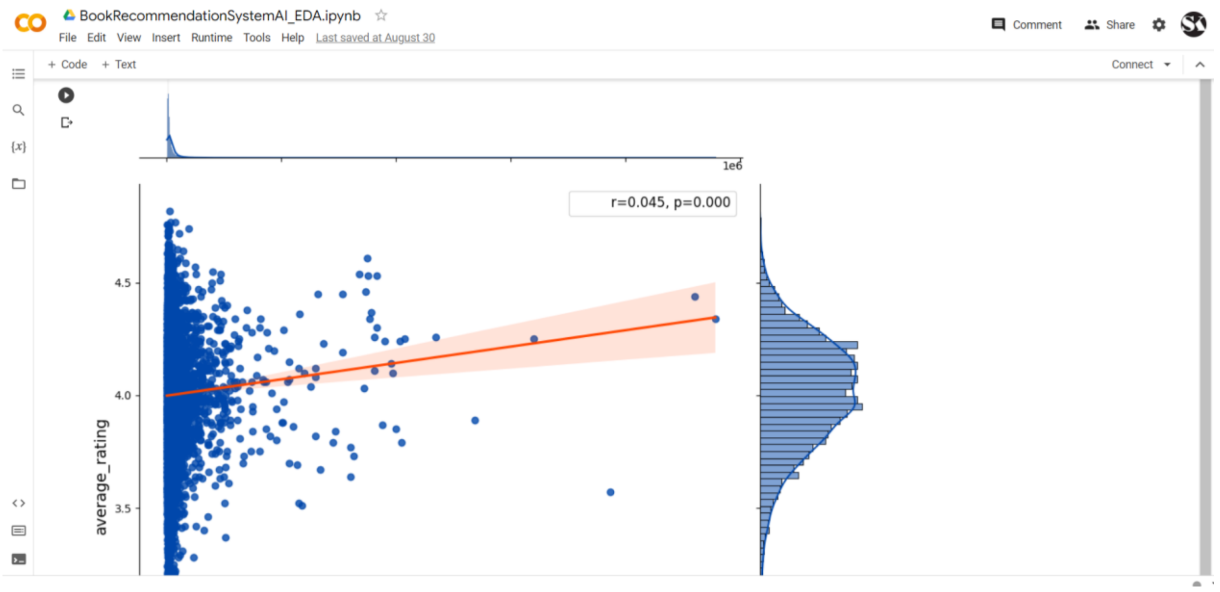


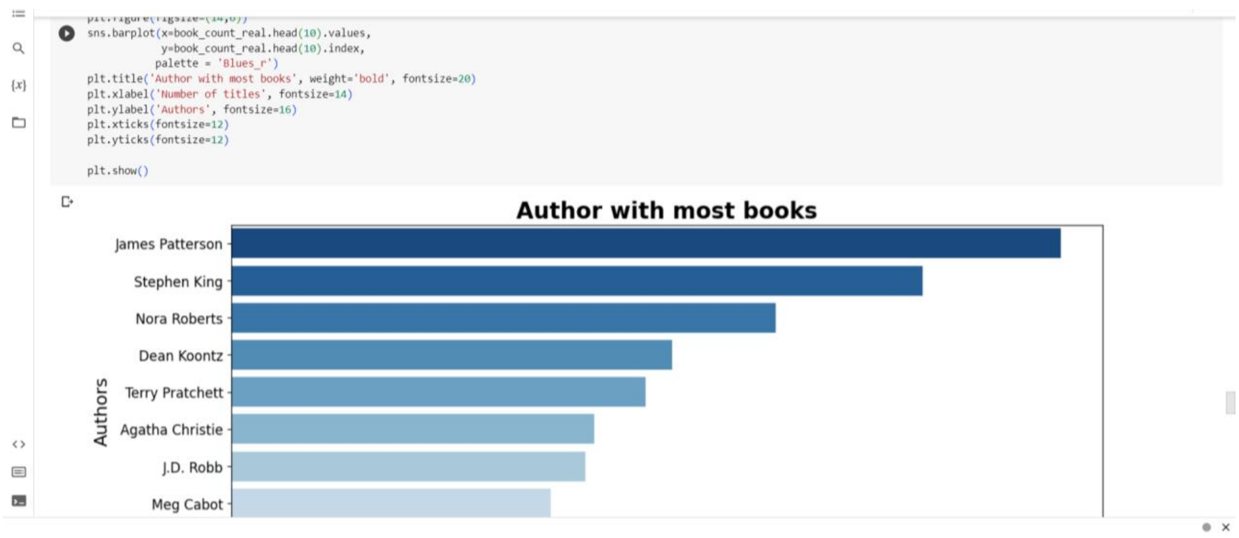
How many ratings does a user generally give?

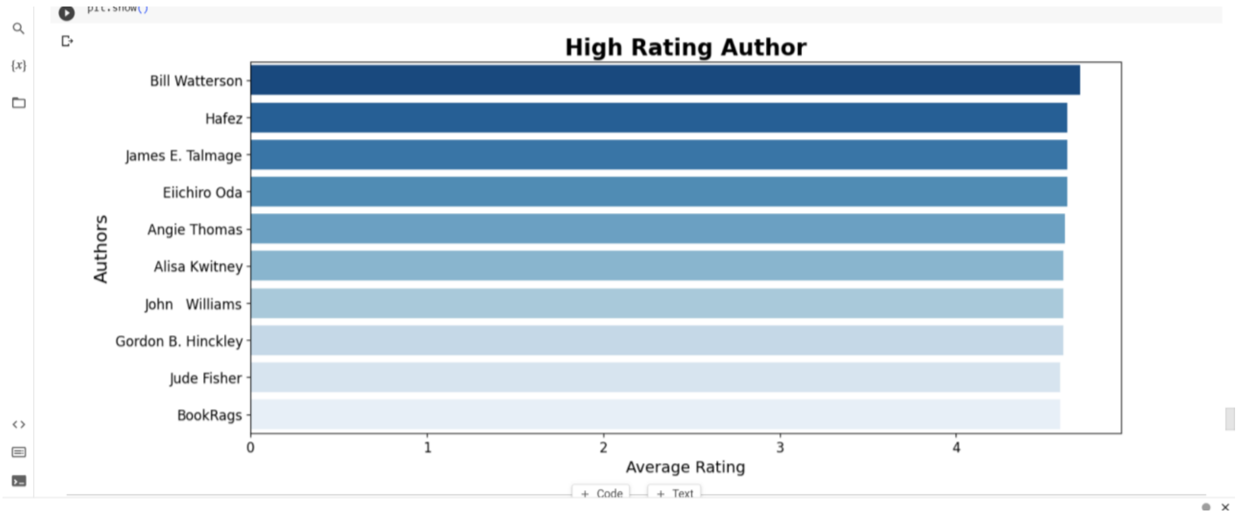
```
book_per_user = r.groupby('user_id')['rating'].count().reset_index()

plt.figure(figsize=(10,6))
sns.distplot(book_per_user['rating'], color='#0047AB')
plt.title('Number of books that a person usually review', weight='bold', fontsize=20)
plt.xlabel('Number of books reviewed', fontsize=18)
plt.ylabel('Density', fontsize=18)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```



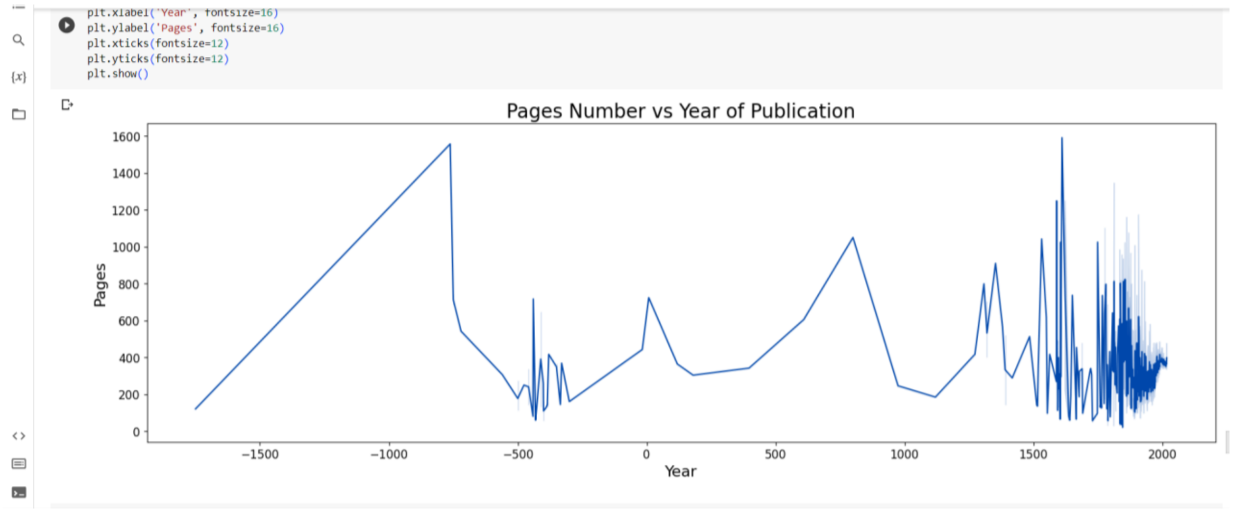



BookRecommendationSystemAI_EDA.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment
Share
+ Code + Text
Connect
1e6
r=0.045, p=0.000
4.5
4.0
3.5
average_rating

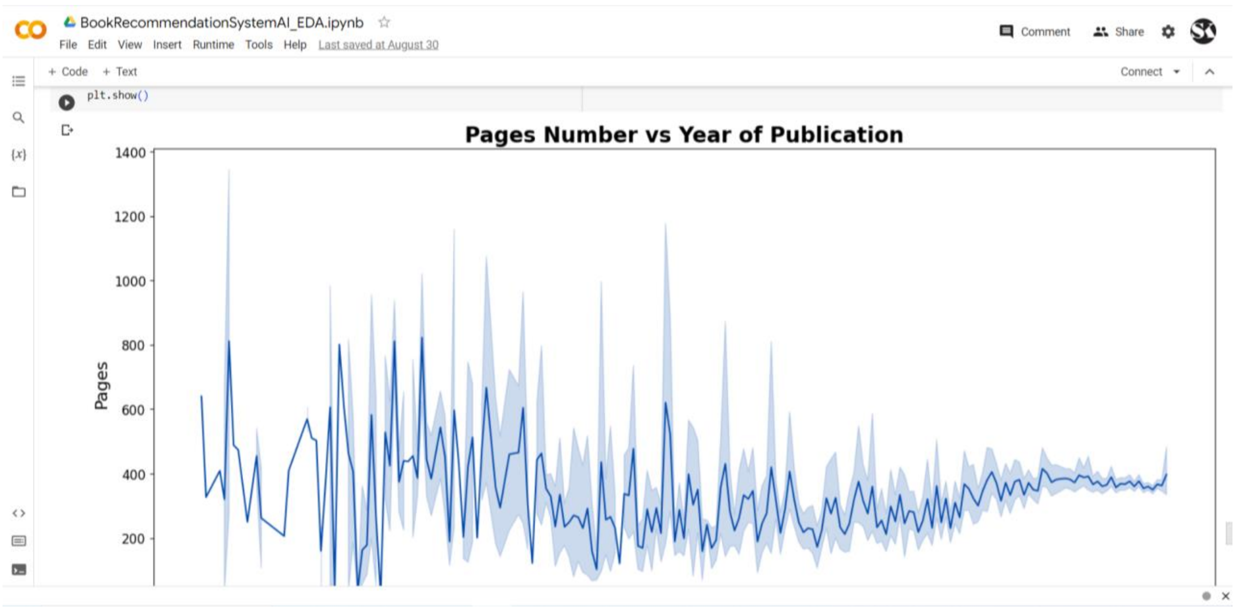

```
plt.figure(figsize=(14,6))
sns.barplot(x=book_count_real.head(10).values,
            y=book_count_real.head(10).index,
           palette = 'Blues_r')
plt.title('Author with most books', weight='bold', fontsize=20)
plt.xlabel('Number of titles', fontsize=14)
plt.ylabel('Authors', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```



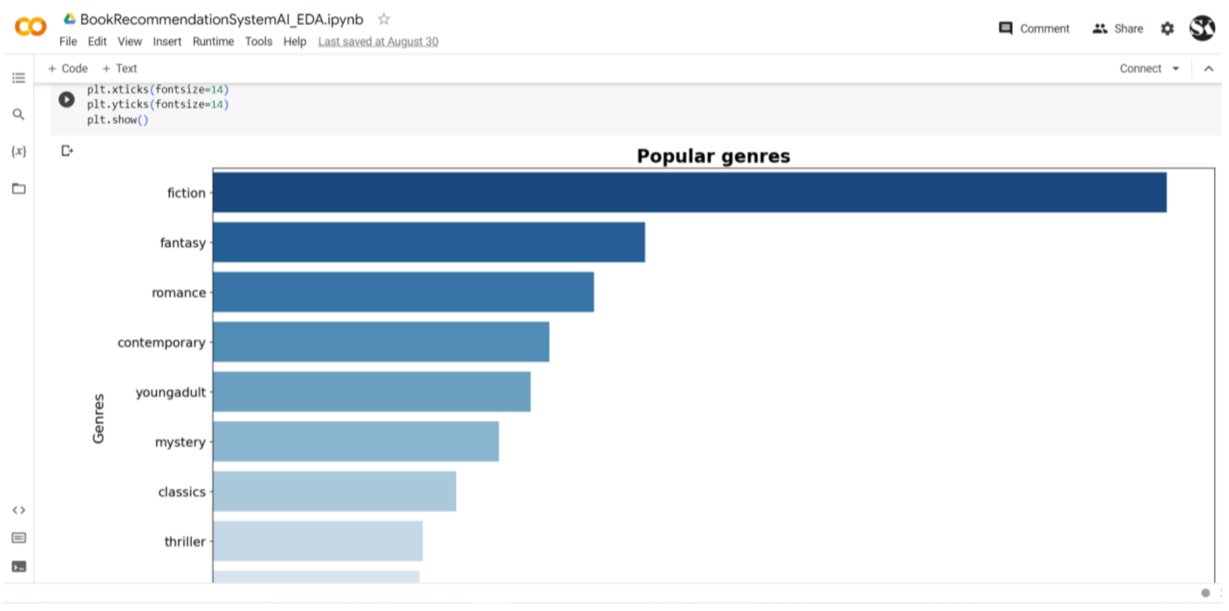
```
plt.show()
```



```
plt.xlabel('Year', fontsize=16)
plt.ylabel('Pages', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```

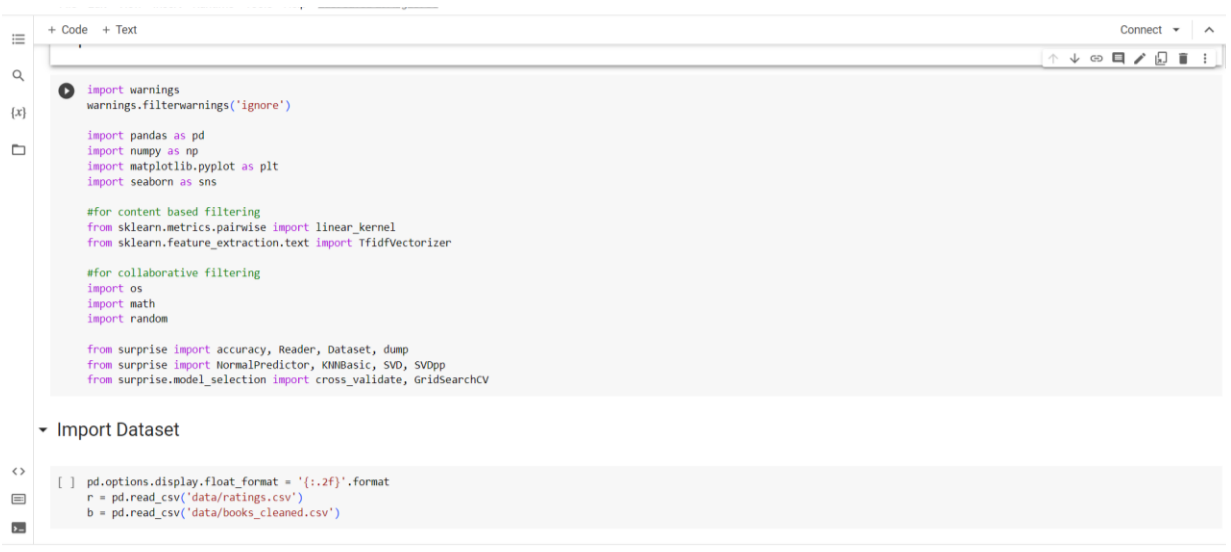


```
plt.show()
```



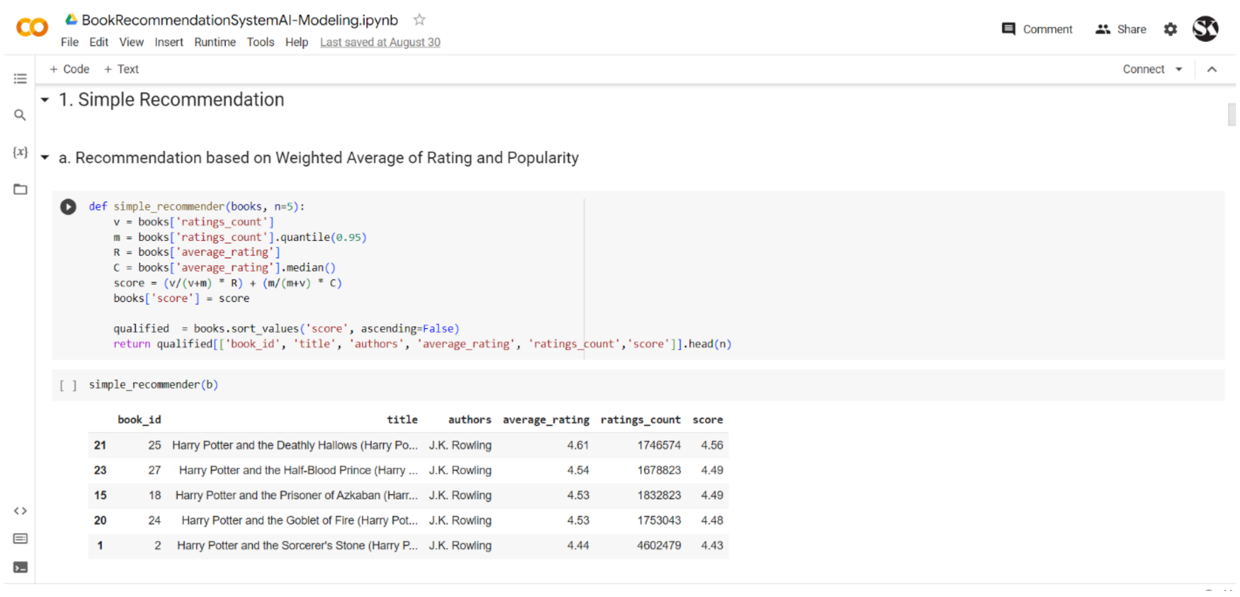
```
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.show()
```

Modelling:-



```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#for content based filtering
from sklearn.metrics.pairwise import linear_kernel
from sklearn.feature_extraction.text import TfidfVectorizer

#for collaborative filtering
import os
import math
import random

from surprise import accuracy, Reader, Dataset, dump
from surprise import NormalPredictor, KNNBasic, SVD, SVDpp
from surprise.model_selection import cross_validate, GridSearchCV
```

```
[ ] pd.options.display.float_format = '{:.2f}'.format
    r = pd.read_csv('data/ratings.csv')
    b = pd.read_csv('data/books_cleaned.csv')
```


BookRecommendationSystemAI-Modeling.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment Share
+ Code + Text
Connect

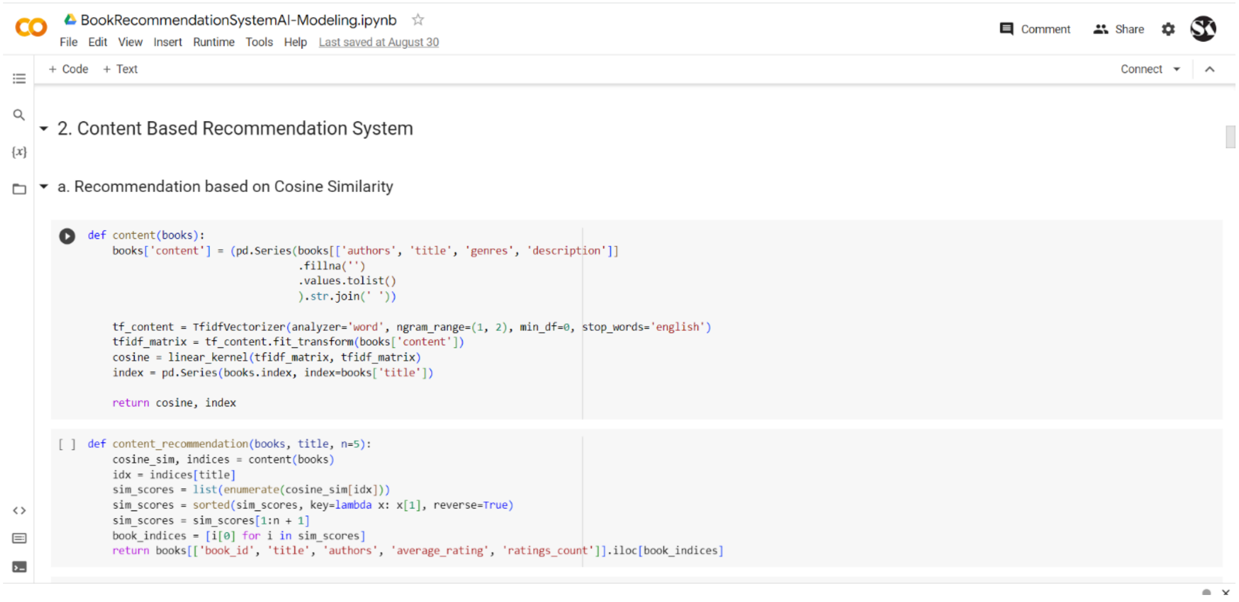
1. Simple Recommendation

a. Recommendation based on Weighted Average of Rating and Popularity

```
def simple_recommender(books, n=5):
    v = books['ratings_count']
    m = books['ratings_count'].quantile(0.95)
    R = books['average_rating']
    C = books['average_rating'].median()
    score = (v/(v+m) * R) + (m/(m+v) * C)
    books['score'] = score

    qualified  = books.sort_values('score', ascending=False)
    return qualified[['book_id', 'title', 'authors', 'average_rating', 'ratings_count','score']].head(n)
```

```
[ ] simple_recommender(b)
```

```
    book_id                                           title     authors  average_rating  ratings_count  score
21       25  Harry Potter and the Deathly Hallows (Harry Po...  J.K. Rowling            4.61        1746574   4.56
23       27   Harry Potter and the Half-Blood Prince (Harry ...  J.K. Rowling            4.54        1678823   4.49
15       18  Harry Potter and the Prisoner of Azkaban (Harr...  J.K. Rowling            4.53        1832823   4.49
20       24    Harry Potter and the Goblet of Fire (Harry Pot...  J.K. Rowling            4.53        1753043   4.48
1         2   Harry Potter and the Sorcerer's Stone (Harry P...  J.K. Rowling            4.44        4602479   4.43
```

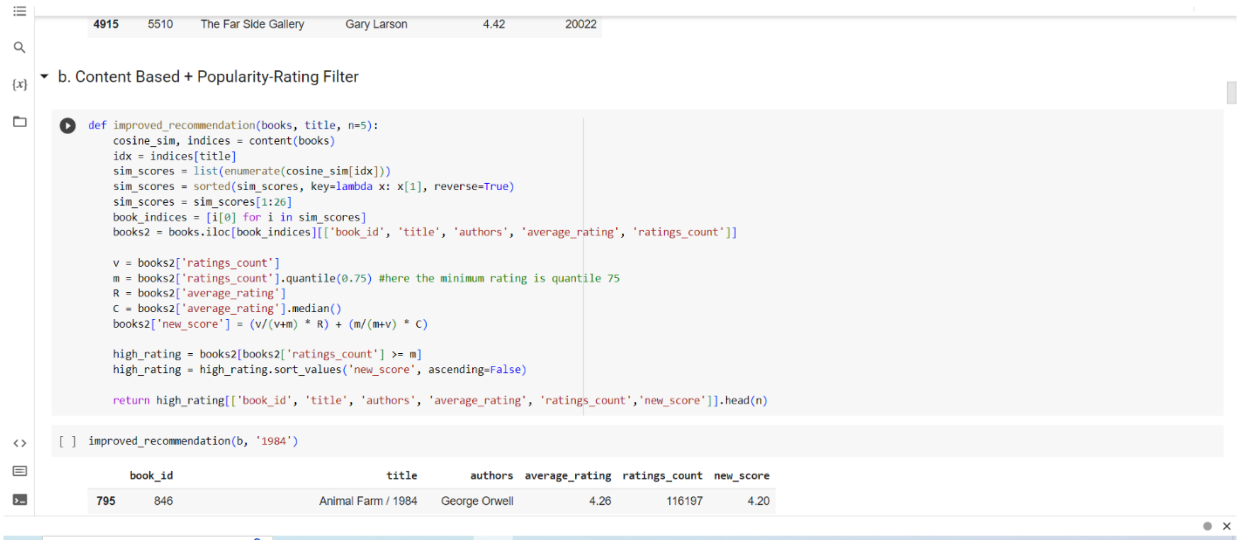

BookRecommendationSystemAI-Modeling.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment Share
+ Code + Text
Connect

2. Content Based Recommendation System

a. Recommendation based on Cosine Similarity

```
def content(books):
    books['content'] = (pd.Series(books[['authors', 'title', 'genres', 'description']]
                                  .fillna('')
                                  .values.tolist()
                                  ).str.join(' '))

    tf_content = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=0, stop_words='english')
    tfidf_matrix = tf_content.fit_transform(books['content'])
    cosine = linear_kernel(tfidf_matrix, tfidf_matrix)
    index = pd.Series(books.index, index=books['title'])

    return cosine, index
```

```
[ ] def content_recommendation(books, title, n=5):
    cosine_sim, indices = content(books)
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:n + 1]
    book_indices = [i[0] for i in sim_scores]
    return books[['book_id', 'title', 'authors', 'average_rating', 'ratings_count']].iloc[book_indices]
```



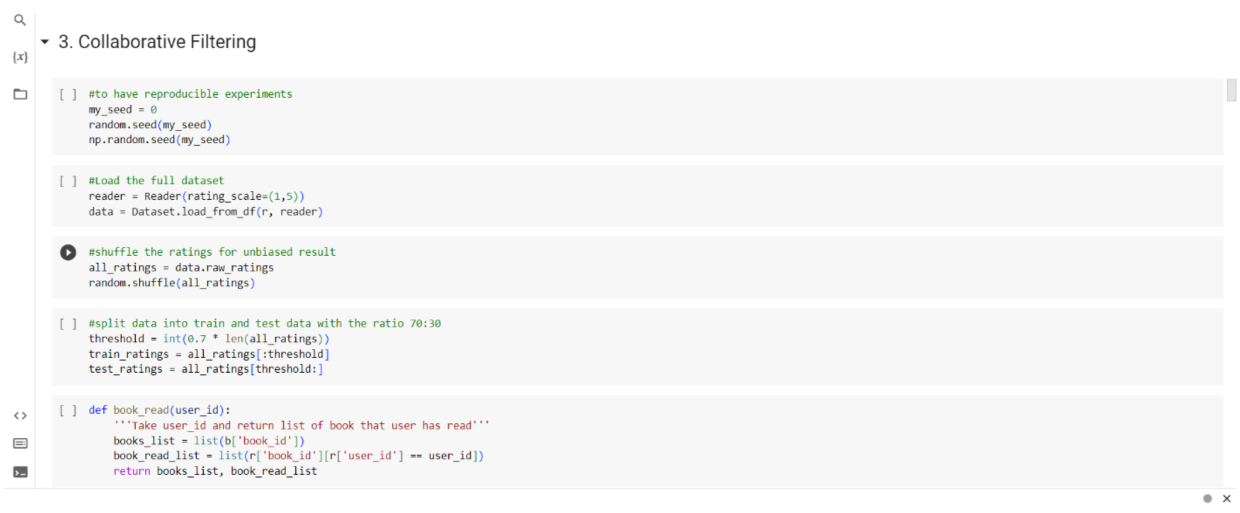
4915	5510	The Far Side Gallery	Gary Larson	4.42	20022

b. Content Based + Popularity-Rating Filter

```
def improved_recommendation(books, title, n=5):
    cosine_sim, indices = content(books)
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    book_indices = [i[0] for i in sim_scores]
    books2 = books.iloc[book_indices][['book_id', 'title', 'authors', 'average_rating', 'ratings_count']]

    v = books2['ratings_count']
    m = books2['ratings_count'].quantile(0.75) #here the minimum rating is quantile 75
    R = books2['average_rating']
    C = books2['average_rating'].median()
    books2['new_score'] = (v/(v+m) * R) + (m/(m+v) * C)

    high_rating = books2[books2['ratings_count'] >= m]
    high_rating = high_rating.sort_values('new_score', ascending=False)

    return high_rating[['book_id', 'title', 'authors', 'average_rating', 'ratings_count','new_score']].head(n)
```

```
improved_recommendation(b, '1984')
```

	book_id	title	authors	average_rating	ratings_count	new_score
795	846	Animal Farm / 1984	George Orwell	4.26	116197	4.20



3. Collaborative Filtering

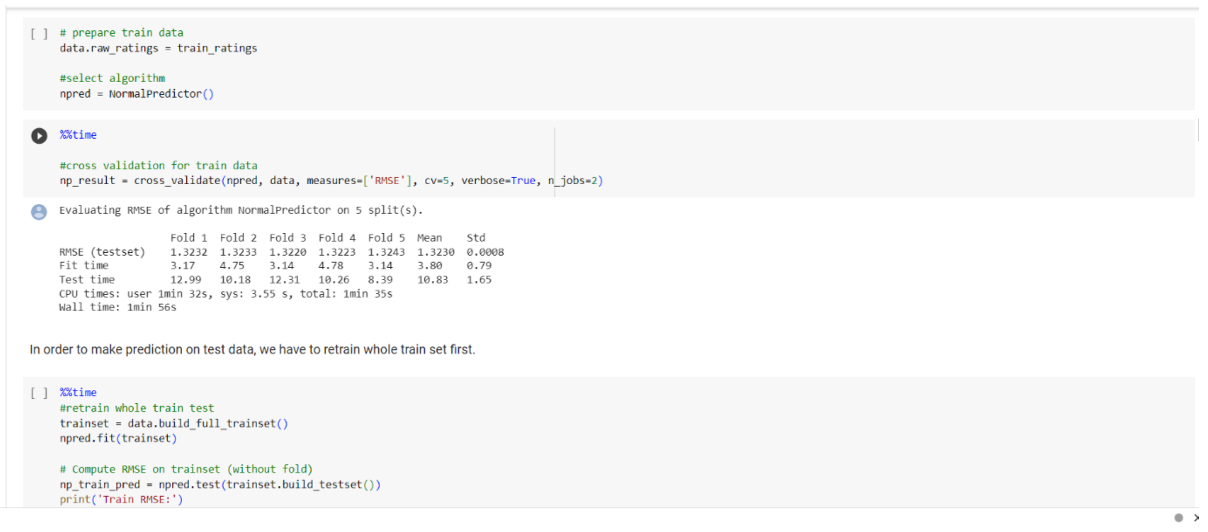
```
#to have reproducible experiments
my_seed = 0
random.seed(my_seed)
np.random.seed(my_seed)
```

```
#Load the full dataset
reader = Reader(rating_scale=(1,5))
data = Dataset.load_from_df(r, reader)
```

```
#shuffle the ratings for unbiased result
all_ratings = data.raw_ratings
random.shuffle(all_ratings)
```

```
#split data into train and test data with the ratio 70:30
threshold = int(0.7 * len(all_ratings))
train_ratings = all_ratings[:threshold]
test_ratings = all_ratings[threshold:]
```

```
def book_read(user_id):
    '''Take user_id and return list of book that user has read'''
    books_list = list(b['book_id'])
    book_read_list = list(r['book_id'][r['user_id'] == user_id])
    return books_list, book_read_list
```



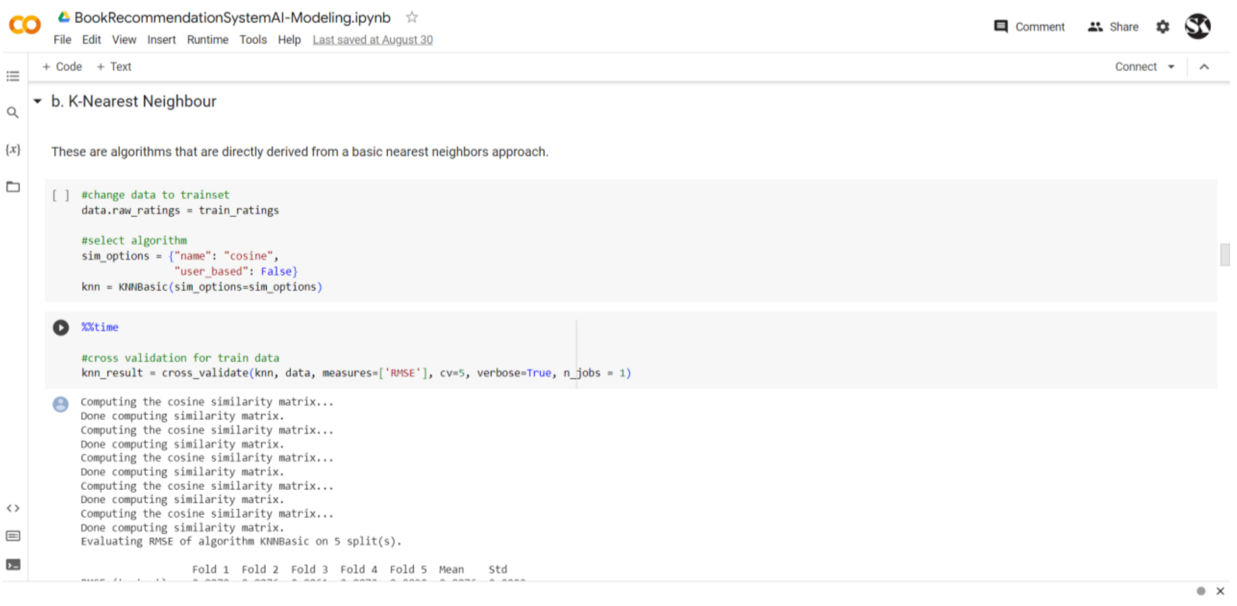
```
# prepare train data
data.raw_ratings = train_ratings

#select algorithm
npred = NormalPredictor()
```

```
%%time

#cross validation for train data
np_result = cross_validate(npred, data, measures=['RMSE'], cv=5, verbose=True, n_jobs=2)
```

```
Evaluating RMSE of algorithm NormalPredictor on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    1.3232  1.3233  1.3220  1.3223  1.3243  1.3230  0.0008
Fit time          3.17    4.75    3.14    4.78    3.14    3.80    0.79
Test time         12.99   10.18   12.31   10.26   8.39    10.83   1.65
CPU times: user 1min 32s, sys: 3.55 s, total: 1min 35s
Wall time: 1min 56s
```

In order to make prediction on test data, we have to retrain whole train set first.

```
%%time
#retrain whole train test
trainset = data.build_full_trainset()
npred.fit(trainset)

# Compute RMSE on trainset (without fold)
np_train_pred = npred.test(trainset.build_testset())
print('Train RMSE:')
```


BookRecommendationSystemAI-Modeling.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment Share
+ Code + Text
Connect

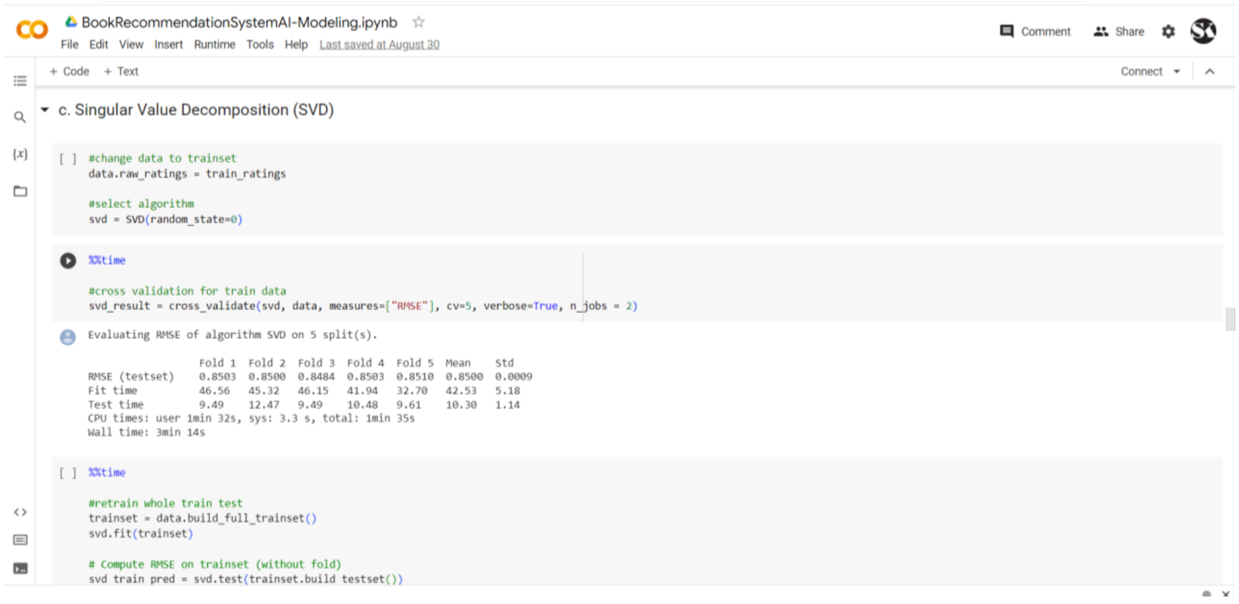
b. K-Nearest Neighbour

These are algorithms that are directly derived from a basic nearest neighbors approach.

```
#change data to trainset
data.raw_ratings = train_ratings

#select algorithm
sim_options = {"name": "cosine",
               "user_based": False}
knn = KNNBasic(sim_options=sim_options)
```

```
%%time

#cross validation for train data
knn_result = cross_validate(knn, data, measures=['RMSE'], cv=5, verbose=True, n_jobs = 1)
```

```
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Computing the cosine similarity matrix...
Done computing similarity matrix.
Evaluating RMSE of algorithm KNNBasic on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
```


BookRecommendationSystemAI-Modeling.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment Share
+ Code + Text
Connect

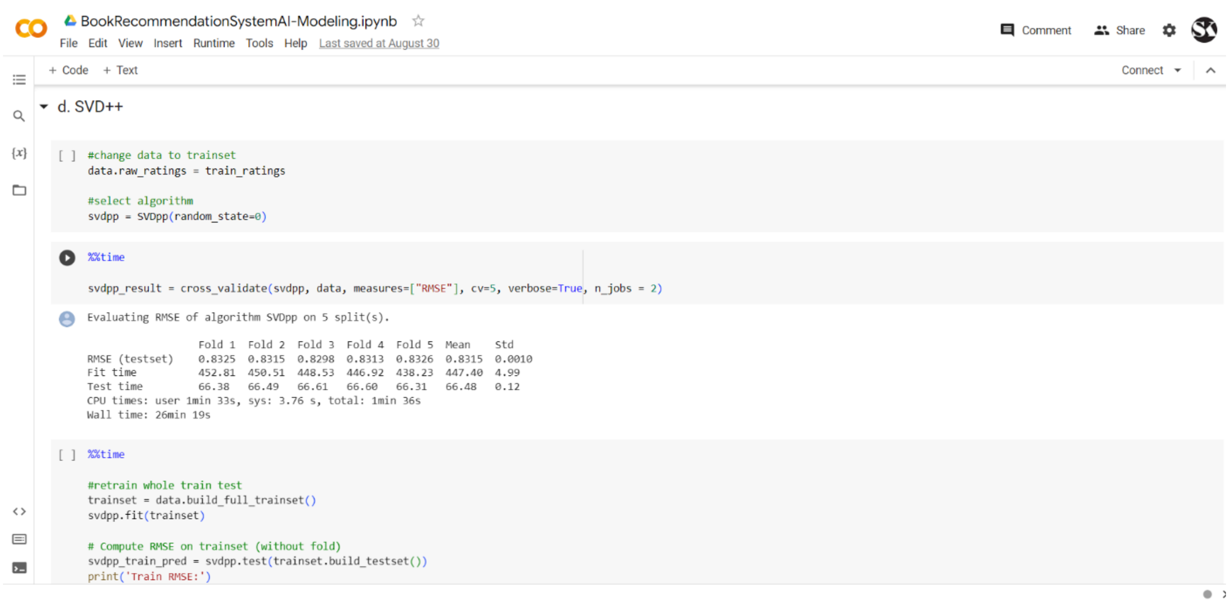
c. Singular Value Decomposition (SVD)

```
#change data to trainset
data.raw_ratings = train_ratings

#select algorithm
svd = SVD(random_state=0)
```

```
%%time

#cross validation for train data
svd_result = cross_validate(svd, data, measures=["RMSE"], cv=5, verbose=True, n_jobs = 2)
```

```
Evaluating RMSE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8503  0.8500  0.8484  0.8503  0.8510  0.8500  0.0009
Fit time          46.56   45.32   46.15   41.94   32.70   42.53   5.18
Test time         9.49    12.47   9.49    10.48   9.61    10.30   1.14
CPU times: user 1min 32s, sys: 3.3 s, total: 1min 35s
Wall time: 3min 14s
```

```
%%time

#retrain whole train test
trainset = data.build_full_trainset()
svd.fit(trainset)

# Compute RMSE on trainset (without fold)
svd train pred = svd.test(trainset.build testset())
```


BookRecommendationSystemAI-Modeling.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment Share
+ Code + Text
Connect

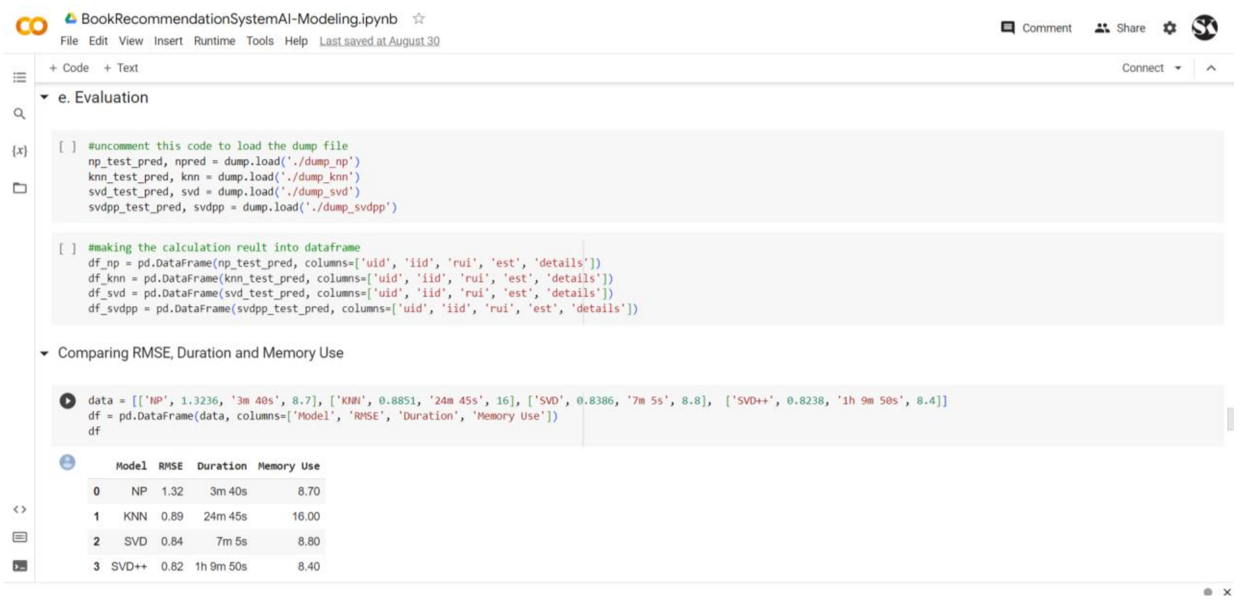
d. SVD++

```
#change data to trainset
data.raw_ratings = train_ratings

#select algorithm
svdpp = SVDpp(random_state=0)
```

```
%%time

svdpp_result = cross_validate(svdpp, data, measures=["RMSE"], cv=5, verbose=True, n_jobs = 2)
```

```
Evaluating RMSE of algorithm SVDpp on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8325  0.8315  0.8298  0.8313  0.8326  0.8315  0.0010
Fit time          452.81  450.51  448.53  446.92  438.23  447.40  4.99
Test time         66.38   66.49   66.61   66.60   66.31   66.48   0.12
CPU times: user 1min 33s, sys: 3.76 s, total: 1min 36s
Wall time: 26min 19s
```

```
%%time

#retrain whole train test
trainset = data.build_full_trainset()
svdpp.fit(trainset)

# Compute RMSE on trainset (without fold)
svdpp_train_pred = svdpp.test(trainset.build_testset())
print('Train RMSE:')
```



BookRecommendationSystemAI-Modeling.ipynb
File Edit View Insert Runtime Tools Help Last saved at August 30
Comment Share
+ Code + Text
Connect

e. Evaluation

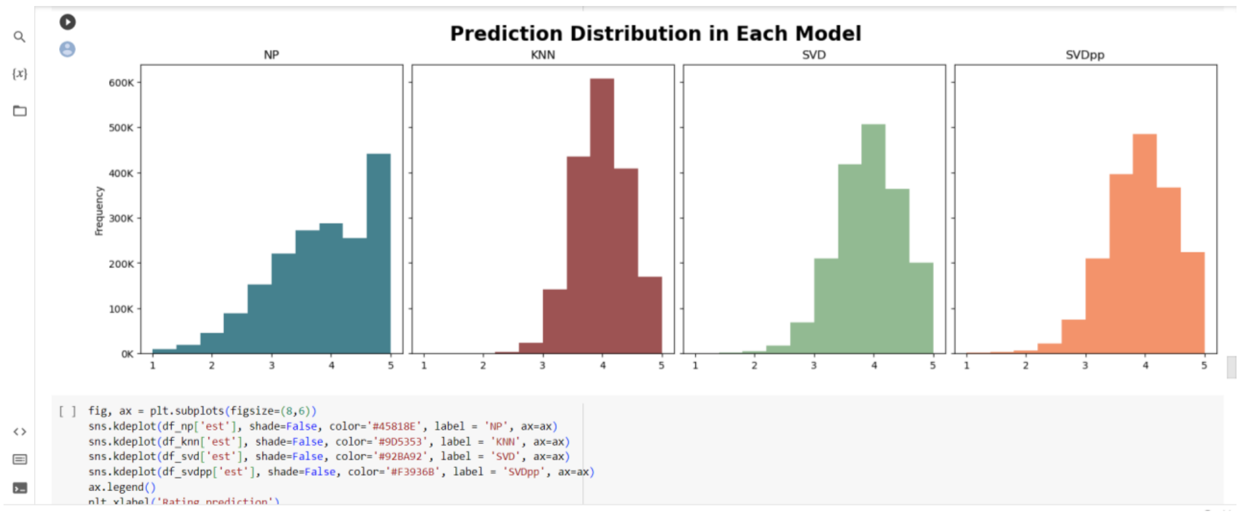
```
#uncomment this code to load the dump file
np_test_pred, npred = dump.load('./dump_np')
knn_test_pred, knn = dump.load('./dump_knn')
svd_test_pred, svd = dump.load('./dump_svd')
svdpp_test_pred, svdpp = dump.load('./dump_svdpp')
```

```
#making the calculation reult into dataframe
df_np = pd.DataFrame(np_test_pred, columns=['uid', 'iid', 'rui', 'est', 'details'])
df_knn = pd.DataFrame(knn_test_pred, columns=['uid', 'iid', 'rui', 'est', 'details'])
df_svd = pd.DataFrame(svd_test_pred, columns=['uid', 'iid', 'rui', 'est', 'details'])
df_svdpp = pd.DataFrame(svdpp_test_pred, columns=['uid', 'iid', 'rui', 'est', 'details'])
```

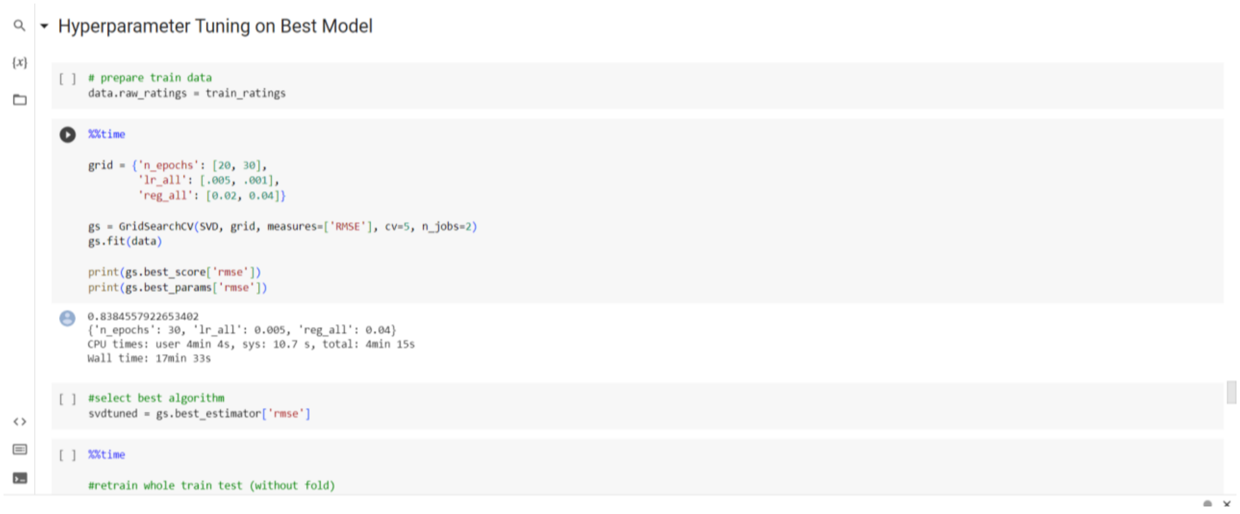
Comparing RMSE, Duration and Memory Use

```
data = [['NP', 1.3236, '3m 40s', 8.7], ['KNN', 0.8851, '24m 45s', 16], ['SVD', 0.8386, '7m 5s', 8.8],  ['SVD++', 0.8238, '1h 9m 50s', 8.4]]
df = pd.DataFrame(data, columns=['Model', 'RMSE', 'Duration', 'Memory Use'])
df
```

	Model	RMSE	Duration	Memory Use
0	NP	1.32	3m 40s	8.70
1	KNN	0.89	24m 45s	16.00
2	SVD	0.84	7m 5s	8.80
3	SVD++	0.82	1h 9m 50s	8.40



```
fig, ax = plt.subplots(figsize=(8,6))
sns.kdeplot(df_np['est'], shade=False, color='#45818E', label = 'NP', ax=ax)
sns.kdeplot(df_knn['est'], shade=False, color='#9D5353', label = 'KNN', ax=ax)
sns.kdeplot(df_svd['est'], shade=False, color='#92BA92', label = 'SVD', ax=ax)
sns.kdeplot(df_svdpp['est'], shade=False, color='#F3936B', label = 'SVDpp', ax=ax)
ax.legend()
plt.xlabel('Rating prediction')
```



Hyperparameter Tuning on Best Model

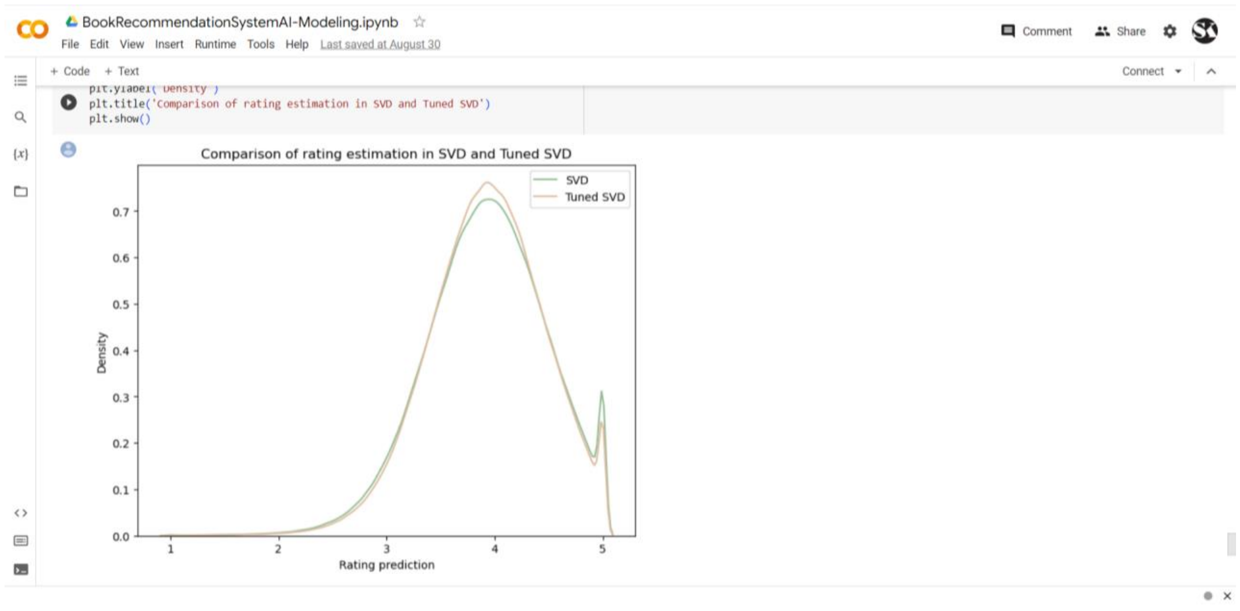
```
# prepare train data
data.raw_ratings = train_ratings
```

```
%%time

grid = {'n_epochs': [20, 30],
        'lr_all': [.005, .001],
        'reg_all': [0.02, 0.04]}

gs = GridSearchCV(SVD, grid, measures=['RMSE'], cv=5, n_jobs=2)
gs.fit(data)

print(gs.best_score['rmse'])
print(gs.best_params['rmse'])
```

```
0.8384557922653402
{'n_epochs': 30, 'lr_all': 0.005, 'reg_all': 0.04}
CPU times: user 4min 4s, sys: 10.7 s, total: 4min 15s
Wall time: 17min 33s
```

```
#select best algorithm
svdtuned = gs.best_estimator['rmse']
```

```
%%time

#retrain whole train test (without fold)
```

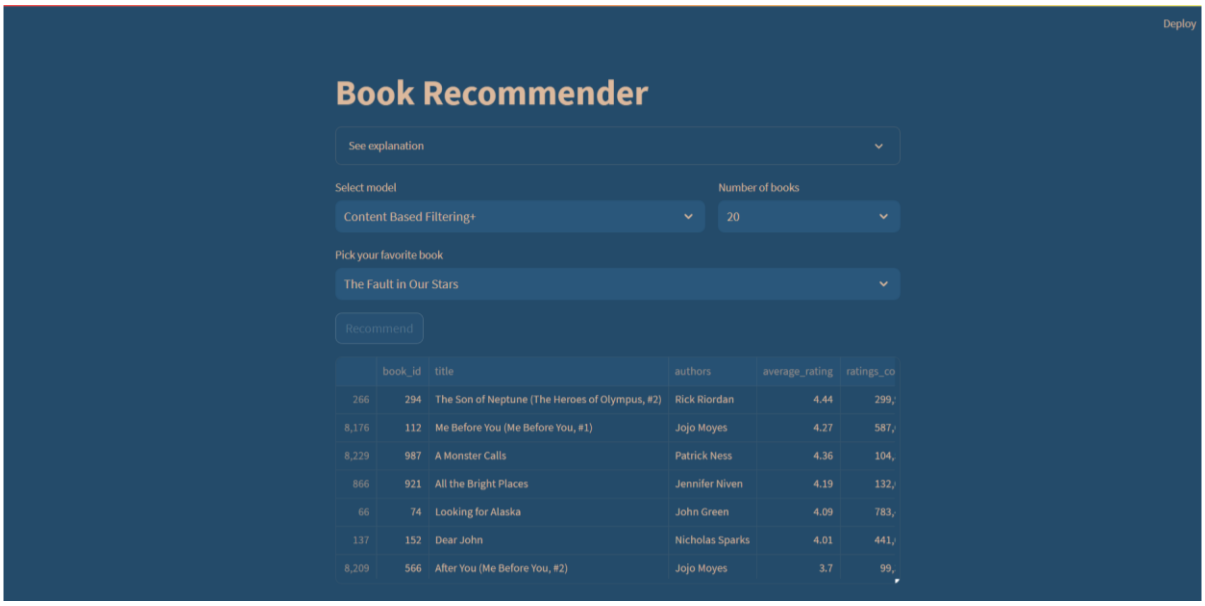


```
plt.ylabel('Density')
plt.title('Comparison of rating estimation in SVD and Tuned SVD')
plt.show()
```

Frontend Streamlit Code:-

```
import numpy as np
import streamlit as st
import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

# collaborative filtering works perfectly on local
# from surprise import Reader, Dataset, SVD


# data loading
@st.cache_data()
def read_book_data():
    return pd.read_csv('data/books_cleaned.csv')


# @st.cache()
# def read_ratings_data():
#     return pd.read_csv('data/ratings.csv')


@st.cache_data()
def content(books):
    books['content'] = (pd.Series(books[['authors', 'title', 'genres',
'description']]
                                  .fillna('')
                                  .values.tolist()
                                  ).str.join(' '))

    tf_content = TfidfVectorizer(analyzer='word', ngram_range=(1, 2),
min_df=0, stop_words='english')
    tfidf_matrix = tf_content.fit_transform(books['content'])
    cosine = linear_kernel(tfidf_matrix, tfidf_matrix)
    index = pd.Series(books.index, index=books['title'])

    return cosine, index


def simple_recommender(books, n=5):
    v = books['ratings_count']
    m = books['ratings_count'].quantile(0.95)
    R = books['average_rating']
    C = books['average_rating'].median()
    score = (v / (v + m) * R) + (m / (m + v) * C)
    books['score'] = score
    qualified = books.sort_values('score', ascending=False)
    return qualified[['book_id', 'title', 'authors', 'score']].head(n)


def content_recommendation(books, title, n=5):
    cosine_sim, indices = content(books)
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:n + 1]
    book_indices = [i[0] for i in sim_scores]
    return books[['book_id', 'title', 'authors', 'average_rating',
'ratings_count']].iloc[book_indices]


def improved_recommendation(books, title, n=5):
    cosine_sim, indices = content(books)
    idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    book_indices = [i[0] for i in sim_scores]
    books2 = books.iloc[book_indices][['book_id', 'title', 'authors',
'average_rating', 'ratings_count']]

    v = books2['ratings_count']
    m = books2['ratings_count'].quantile(0.75)  # here the minimum rating is
quantile 75
    R = books2['average_rating']
    C = books2['average_rating'].median()
    books2['weighted_rating'] = (v / (v + m) * R) + (m / (m + v) * C)

    high_rating = books2[books2['ratings_count'] >= m]
    high_rating = high_rating.sort_values('weighted_rating',
ascending=False)

    return high_rating[['book_id', 'title', 'authors', 'average_rating',
'ratings_count']].head(n)


# def book_read(books, ratings_data, user_id):
#     """Take user_id and return list of book that user has read"""
#     books_list = list(books['book_id'])
#     book_read_list = list(ratings_data['book_id'][ratings_data['user_id']
== user_id])
#     return books_list, book_read_list


# def get_recommendation_svd(books, ratings_data, user_id, n=5):
#     """Give n recommendation to user_id"""
#
#     all_books, user_books = book_read(books, ratings_data, user_id)
#     next_books = [book for book in all_books if book not in user_books]
#
#     reader = Reader(rating_scale=(1, 5))
#     data = Dataset.load_from_df(ratings_data, reader)
#     svd = SVD(random_state=0, n_epochs=30, lr_all=0.005, reg_all=0.04)
#     svd.fit(data.build_full_trainset())
#
#     if n <= len(next_books):
#         ratings = []
#         for book in next_books:
#             est = svd.predict(user_id, book).est
#             ratings.append((book, est))
#         ratings = sorted(ratings, key=lambda x: x[1], reverse=True)
#         book_ids = [id_ for id_, rate in ratings[:n]]
#         return books[books.book_id.isin(book_ids)][['book_id', 'title',
'authors', 'average_rating', 'ratings_count']]
#     else:
#         print('Please reduce your recommendation request')


# App declaration
def main():
    st.set_page_config(page_title="Book Recommender", page_icon="📘",
layout="centered", initial_sidebar_state="auto",
                       menu_items=None)

    # Header contents
    st.write('# Book Recommender')
    with st.expander("See explanation"):
        st.write("""
            In this book recommender, there are three models available.
            1. Simple Recommender
            This model offers generalized recommendations to every user
based on popularity and average rating of
            the book. This model does not provide user-specific
recommendations.

            2.  Content Based Filtering
            To personalise our recommendations, you need to pick your
favorite book. The cosine similarity between
            books are measured, and then the model will suggest books that
are most similar to a particular book that
            a user liked.

            3. Content Based Filtering+
            The mechanism to remove books with low ratings has been added on
top of the content based filtering.
            This model will return books that are similar to your input, are
popular and have high ratings.

            """)

    books = read_book_data().copy()

    # User input
    model, book_num = st.columns((2, 1))
    selected_model = model.selectbox('Select model',
                                     options=['Simple Recommender', 'Content
Based Filtering',
                                              'Content Based Filtering+'])
    selected_book_num = book_num.selectbox('Number of books',
                                           options=[5, 10, 15, 20, 25])

    if selected_model == 'Simple Recommender':
        if st.button('Recommend'):
            try:
                recs = simple_recommender(books=books,
                                          n=selected_book_num)
                st.write(recs)
            except:
                st.error('Oops!. I need to fix this algorithm.')

    # elif selected_model == 'Collaborative Filtering':
    #     ratings_data = read_ratings_data()
    #     # user_id_picked = st.selectbox('Select user_id to explore',
ratings_data["user_id"].unique(), 0)
    #     user_id_picked = st.number_input(label="User ID:", min_value=1,
max_value=60000)
    #     if st.button('Recommend'):
    #         if user_id_picked in ratings_data["user_id"].unique():
    #             with st.spinner('Getting recommendation for you...'):
    #                 recs = get_recommendation_svd(books=books,
    #
ratings_data=ratings_data,
    #                                                   user_id=user_id_picked,
    #                                                   n=selected_book_num)
    #             st.write(recs)
    #         else:
    #             st.write('You have entered an invalid User ID')

    else:
        options = np.concatenate(([''], books["title"].unique()))
        book_title = st.selectbox('Pick your favorite book', options, 0)

        if selected_model == 'Content Based Filtering':
            if st.button('Recommend'):
                if book_title == '':
                    st.write('Please pick a book or use Rating-Popularity
Model')
                    return
                try:
                    recs = content_recommendation(books=books,
                                                  title=book_title,
                                                  n=selected_book_num)
                    st.write(recs)
                except:
                    st.error('Oops! I need to fix this algorithm.')

        elif selected_model == 'Content Based Filtering+':
            if book_title == '':
                st.write('Please pick a book or use Simple Recommender')
                return
            if st.button('Recommend'):
                try:
                    recs = improved_recommendation(books=books,
                                                   title=book_title,
                                                   n=selected_book_num)
                    st.write(recs)
                except:
                    st.error('Oops! I need to fix this algorithm.')


if __name__ == '__main__':
    main()
```

Ouput:-



Select model: Content Based Filtering+

Number of books: 20

Pick your favorite book: The Fault in Our Stars

	book_id	title	authors	average_rating	ratings_co
266	294	The Son of Neptune (The Heroes of Olympus, #2)	Rick Riordan	4.44	299,
8,176	112	Me Before You (Me Before You, #1)	Jojo Moyes	4.27	587,
8,229	987	A Monster Calls	Patrick Ness	4.36	104,
866	921	All the Bright Places	Jennifer Niven	4.19	132,
66	74	Looking for Alaska	John Green	4.09	783,
137	152	Dear John	Nicholas Sparks	4.01	441,
8,209	566	After You (Me Before You, #2)	Jojo Moyes	3.7	99,

Github Link:-

https://github.com/Santhosh-1801/AI-Book-Recommender-Get-Book-Recommendations-using-AI

Conclusion:-

The model offers the simple recommendation. This model does not provide user-specific recommendations but suitable for new user (have no cold-start problem). Cold problem refers to a problem when the system is unable to draw any conclusions about users or objects because it has not yet acquired enough data. Recommendations based on title, authors, description, and genre using cosine similarity have been made. To provide a balance of book recommendations, an additional popularity-rating filter has been added. This method is suitable for people who are looking for books that are similar to their favorite books, but this system cannot capture tastes and provide recommendations across genres. By applying a content based model, instead of having to rate 30 books to start the recommendation engine, users can just pick one book they liked for Goodreads to provide good recommendations for new users, making the process easier. By using Surprise library, I tried building a recommender with 4 algorithms: Normal Predictor, KNN, SVD, and SVD++. SVD is suitable for making a recommendation system in Goodreads because of its lower RMSE, faster calculation and lower memory requirement unlike KNN. I think the RMSE value of 0.8 is still quite reasonable for a Goodreads rating.Notice that Collaborative filtering doesn’t require features about the items or users to be known. We can only use rating data. However, this recommendation system need prior data and suffers cold start problem, for example when new user or new book added to the list.