Project Step 2: Task 1

Group Number: 9
Team members: Nikita Kumari, Nishu Singh, Jal Dharmendrabhai Patel, Karthikeya Battula

TASK 1: Building a Recommender System

Introduction:
Recommender systems are crucial in modern digital ecosystems, helping users navigate vast amounts of information by suggesting products or content based on past behaviors and preferences. In this project, a personalized book recommendation system was developed using the Book Crossing dataset in combination with a user-based collaborative filtering approach. The objective was to identify users with similar reading habits and recommend books by analyzing their historical rating patterns.

Methodology:
In this project, user-based collaborative filtering was the main methodology. Based on the past ratings, it was determined that the ten most comparable users for every user in the dataset were determined. It subsequently gathered the books rated by those comparable individuals. It produced specific recommendations by calculating a weighted ratings average, which calculated weights using the users' similarity scores. It utilized the cosine similarity metric to assess similarity since it is popular and can capture the proximity of rating structures of any amplitude, since it calculates the cosine of the angle that connects two non-zero vectors.

The Book Crossing dataset, which was acquired from Kaggle, was utilized for this research [1]. Its three primary files are Books.csv, which contains details about various books, like titles, descriptions, and authors; Users.csv, which provides user IDs and demographic data, including age; and Ratings.csv, which has user ratings for books on a rating system of 0 to 10 [1]. The dataset has over 1 million ratings, more than 270,000 books, and more than 270,000 users [1].

Rating and book metadata were merged during preprocessing to generate a user-book interaction matrix. To preserve the caliber of the suggestions, **books that lacked titles,** such as those labeled "Untitled," were eliminated. Recommendations derived from these books would prove worthless for consumers and decrease the system's perceived superiority. Additionally, ratings with a zero score were eliminated because they frequently failed to offer insightful commentary. If a recommended book had been invalid or missing, it was replaced with the next most suitable suggestion. Less than five recommendations were given to the user when there was no suitable replacement.

After identifying the top ten similar users, all of the books were gathered to generate suggestions. The similarity scores were then used as weights to determine each book's average score. The novels that were recommended were ones that the initial user had never read before. For the final recommendation, the top five novels were chosen with the highest weighted average scores of the preceding list.

Code:

```
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import lil_matrix, csr_matrix
from tqdm import tqdm

# Load ratings.libsvm and build user-book matrix
libsvm_file = "ratings.libsvm"
num_users, max_book_id = 0, 0

with open(libsvm_file, 'r') as file:
    for line in file:
        parts = line.strip().split()[1:]
        for part in parts:
            book_id, _ = part.split(":")
            max_book_id = max(max_book_id, int(book_id))
        num_users += 1

rating_matrix = lil_matrix((num_users, max_book_id + 1))
with open(libsvm_file, 'r') as file:
    for i, line in enumerate(file):
        parts = line.strip().split()[1:]
        for part in parts:
            book_id, rating = part.split(":")
            rating_matrix[i, int(book_id)] = float(rating)

rating_matrix = rating_matrix.tocsr()

# Load Books.csv and build mappings
books_df = pd.read_csv("Books.csv", sep=';', low_memory=False)
books_df.columns = books_df.columns.str.strip()

title_col = [col for col in books_df.columns if "title" in col.lower()]
```

```
if not title_col:
    raise ValueError("No title column found.")
title_col = title_col[0]

isbn_to_title = dict(zip(books_df['ISBN'], books_df[title_col]))
book_id_to_isbn = {idx + 1: isbn for idx, isbn in enumerate(books_df['ISBN'])}

# Compute cosine similarity as similarity measure
similarity_matrix = cosine_similarity(rating_matrix)

# Generate recommendations
recommendations = []

for user_idx in tqdm(range(num_users), desc="Generating Recommendations", miniters=500):
    user_ratings = rating_matrix[user_idx]
    user_sim = similarity_matrix[user_idx]
    top_users = np.argsort(user_sim)[-11:-1][::-1]

    neighbor_books = set()
    for neighbor_idx in top_users:
        neighbor_books.update(rating_matrix[neighbor_idx].nonzero()[1])

    user_books = set(user_ratings.nonzero()[1])
    recommendable_books = neighbor_books - user_books

    estimated_ratings = {}
    for book_idx in recommendable_books:
        weighted_sum = 0
        sim_sum = 0
        for neighbor_idx in top_users:
            rating = rating_matrix[neighbor_idx, book_idx]
            if rating > 0:
                weighted_sum += user_sim[neighbor_idx] * rating
                sim_sum += user_sim[neighbor_idx]
        if sim_sum > 0:
            estimated_ratings[book_idx] = weighted_sum / sim_sum

    top_books = sorted(estimated_ratings.items(), key=lambda x: -x[1])

    selected_books = []
```

```
    for book_idx, score in top_books:
        isbn = book_id_to_isbn.get(book_idx, None)
        title = isbn_to_title.get(isbn, "Unknown Title")
        if title != "Unknown Title":
            selected_books.append((user_idx, book_idx, title, round(score, 2)))
        if len(selected_books) == 5:
            break

    recommendations.extend(selected_books)


## saving to csv with sort by user ID
recommendations_df = pd.DataFrame(recommendations, columns=["User_ID", "Book_ID",
"Book_Title", "Recommendation_Score"])
recommendations_df["User_ID"] = recommendations_df["User_ID"] + 1
recommendations_df = recommendations_df.sort_values(by=["User_ID",
"Recommendation_Score"], ascending=[True, False])
recommendations_df.to_csv("final_recommendations.csv", index=False)

import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
sns.histplot(df['Recommendation_Score'], bins=20, kde=True)
plt.xlabel('Predicted Recommendation Score')
plt.ylabel('Frequency')
plt.title('Distribution of Recommendation Scores')
plt.grid(True)
plt.tight_layout()
plt.show()

reco_counts = recommendations_per_user.value_counts()
labels = [f'{i} Books' for i in reco_counts.index]
sizes = reco_counts.values

plt.figure(figsize=(7,7))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140,
wedgeprops={'edgecolor':'white'})
plt.title('Percentage of Users by Number of Recommendations')
plt.tight_layout()
```

```
plt.show()

plt.figure(figsize=(10,6))
top_books.sort_values().plot(kind='barh')
plt.xlabel('Number of Recommendations')
plt.ylabel('Book ID')
plt.title('Top 10 Most Recommended Books')
plt.grid(axis='x')
plt.tight_layout()
plt.show()
```

Result Analysis:

Observations and Explanations:

The system processed a total of **105,283** users, generating book recommendations based on their historical ratings. Despite this, the final output contained approximately **203,165** recommendation records, which is lower than the expected number if every user had received five suggestions. This discrepancy arose due to several users receiving fewer than five recommendations, a result of sparse rating histories or missing book metadata, with the system naturally avoiding recommending books with incomplete information.

Analyzing the recommendation scores revealed that most predicted scores were concentrated between 9 and 10, demonstrating high system confidence and strong alignment among user preferences. Only a small fraction of scores fell below this range, indicating that the system primarily generated suggestions that it considered highly relevant.

Looking at the distribution of recommendations, around **82.1%** of users received five book suggestions. A smaller portion of users, specifically those with limited activity or less overlap with their neighbors, received only one to four recommendations. This behavior reflects the sparsity challenge commonly faced in collaborative filtering systems.

Further examination showed that certain books were recommended significantly more often than others, with Book IDs such as **65551**, **32788**, **98350**, and **32828** appearing repeatedly. This indicates the presence of highly popular titles that appealed to a broad section of the user base.

Overall, the results demonstrate that the user-based collaborative filtering model performed effectively for the majority of users, successfully balancing recommendation quality with the realities of sparse and incomplete data.

Limitations:

The recommendation system exhibited several limitations. Users with few past ratings often received fewer recommendations, highlighting the cold-start problem inherent to user-based collaborative filtering methods. Additionally, the concentration of predicted scores near the upper bound may suggest reduced diversity in recommendations, which could result in repetitive suggestions over time. The system's ability to diversify based on genre or author was also limited by the lack of detailed structured metadata. Furthermore, because there was no separate test set with known ground truth, it was not possible to compute formal evaluation metrics such as precision, recall, or F1 score.

Future Work:

Future work could focus on addressing sparsity by incorporating item-based collaborative filtering, which relies on similarities between books rather than users. Building a hybrid system that combines collaborative filtering with content-based techniques using book genres, authors, and descriptions would likely improve recommendation diversity and relevance. Matrix factorization methods like Singular Value Decomposition (SVD) could be explored to predict missing ratings more accurately and address the data sparsity issue [2]. Improving metadata quality and cleaning invalid or inactive user accounts could further enhance the effectiveness of the system.

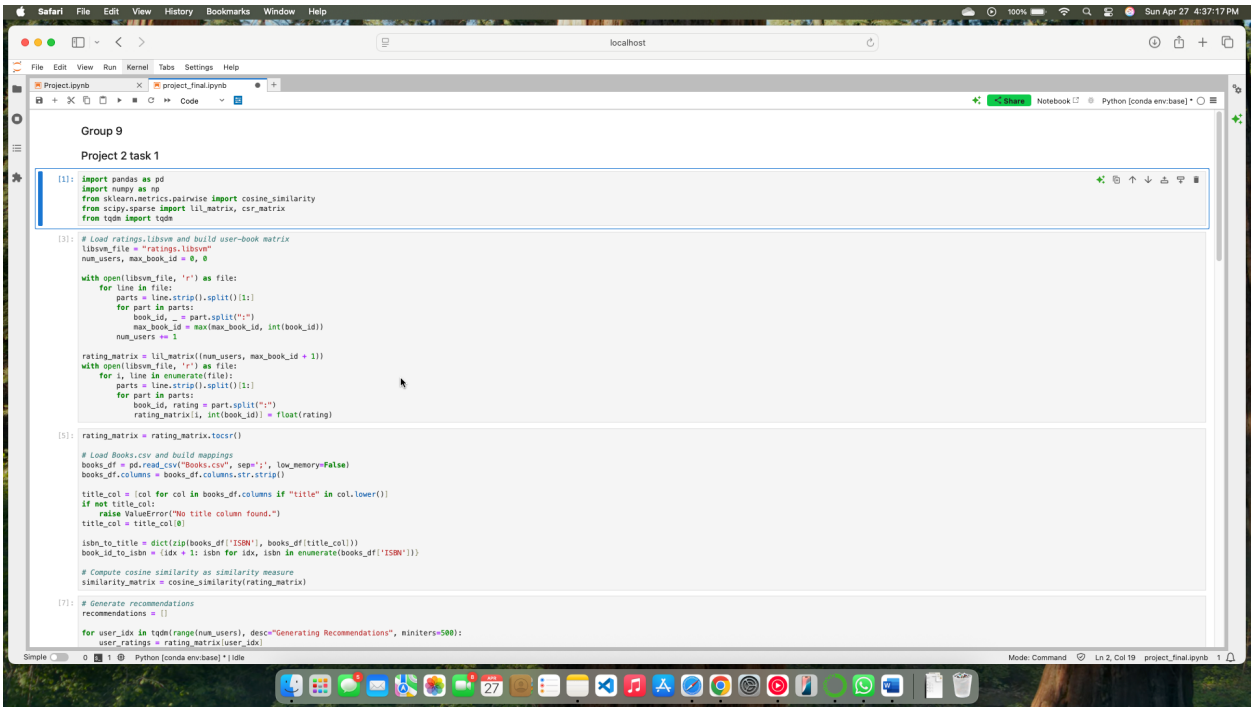
Conclusion:

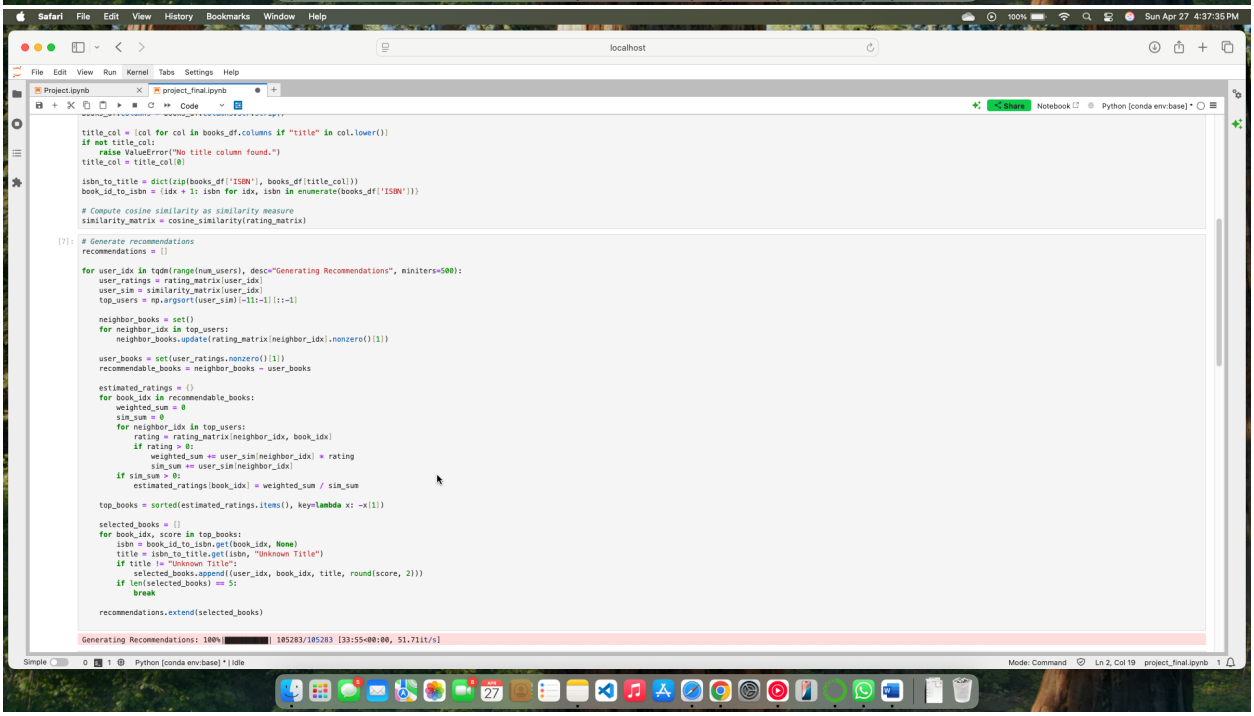
This project successfully developed a personalized book recommender system using user-based collaborative filtering on the Book Crossing dataset. Despite challenges such as sparse data, missing metadata, and inherent popularity biases, the system was able to produce meaningful recommendations for the majority of users. The project highlighted both the strengths and limitations of collaborative filtering in real-world settings. Looking ahead, hybrid recommendation models and improved data management techniques could be explored to further improve the system's performance and recommendation diversity.

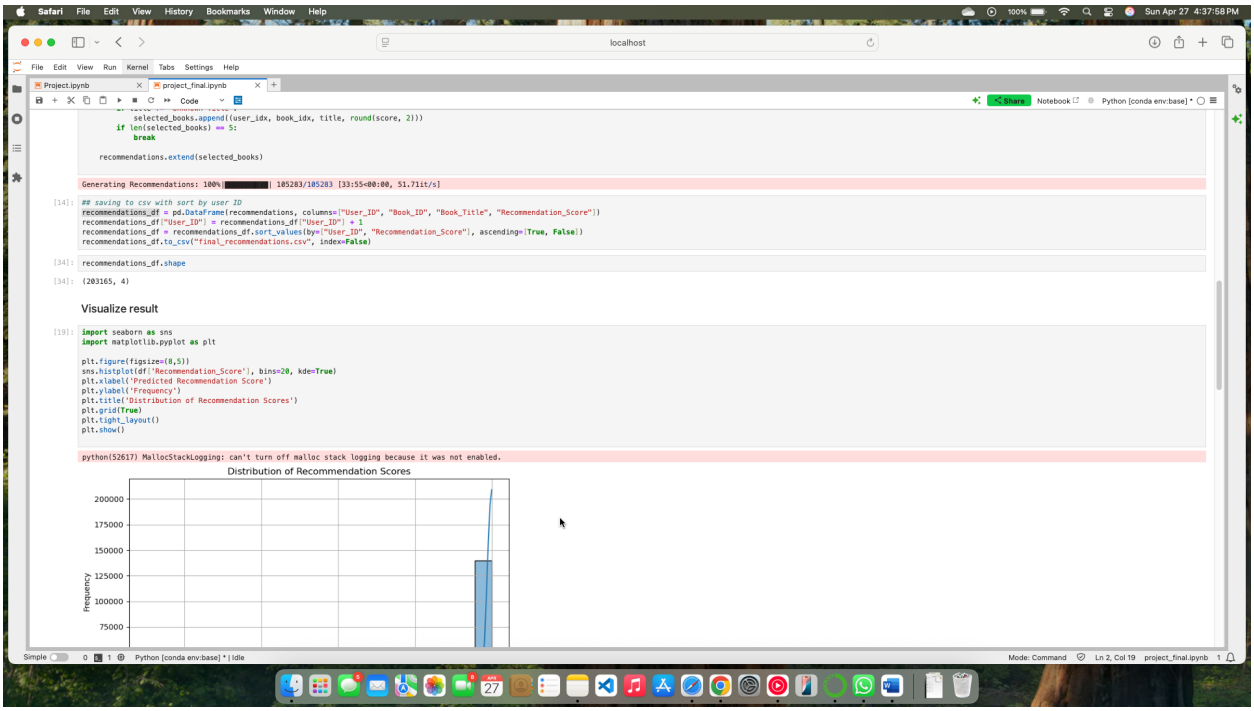
References:

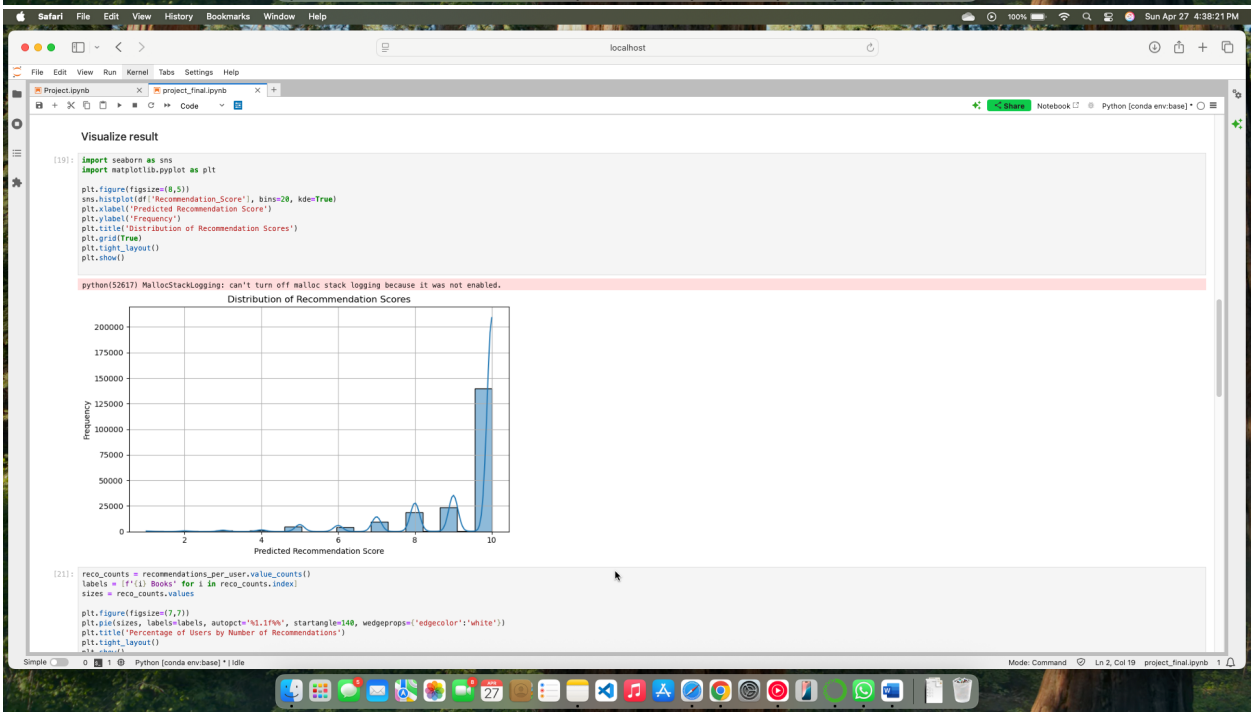
1. somnambWl. (2019, September 7). *Book-crossing dataset*. Kaggle. https://www.kaggle.com/datasets/somnambwl/bookcrossing-dataset?resource=download
2. GeeksforGeeks. (2022, October 17). *Singular Value Decomposition (SVD)*. GeeksforGeeks. https://www.geeksforgeeks.org/singular-value-decomposition-svd/

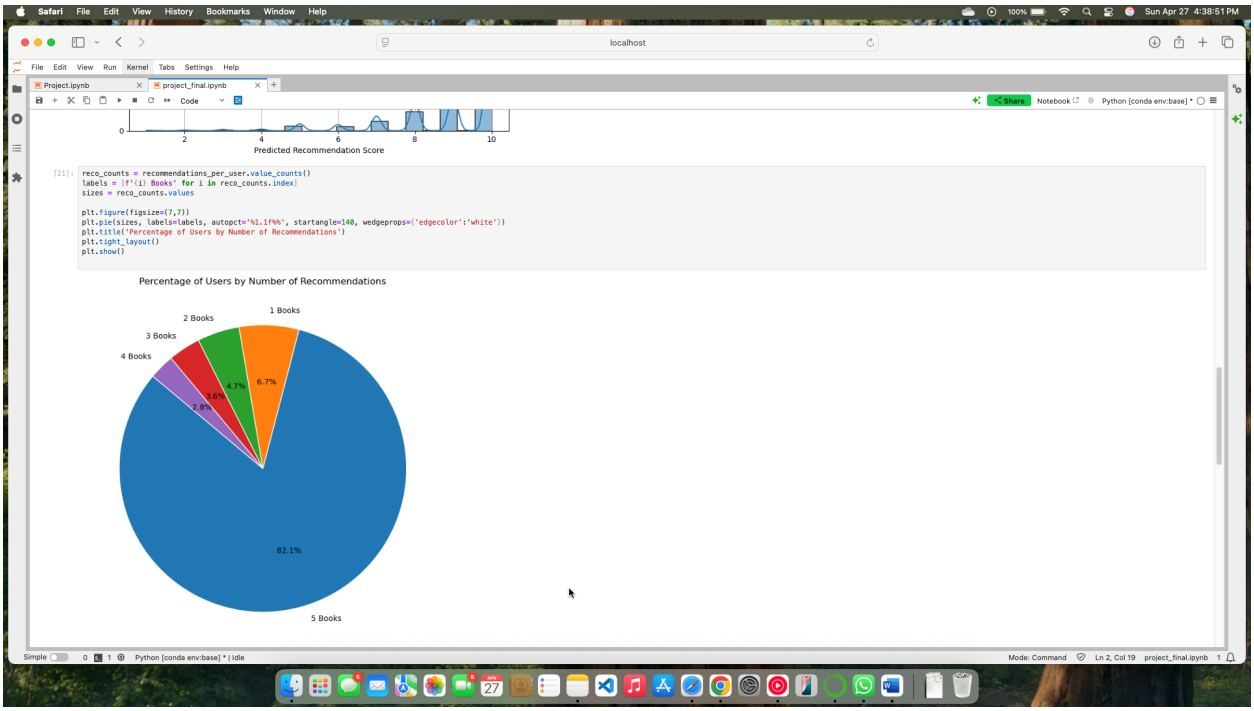
Screenshots:

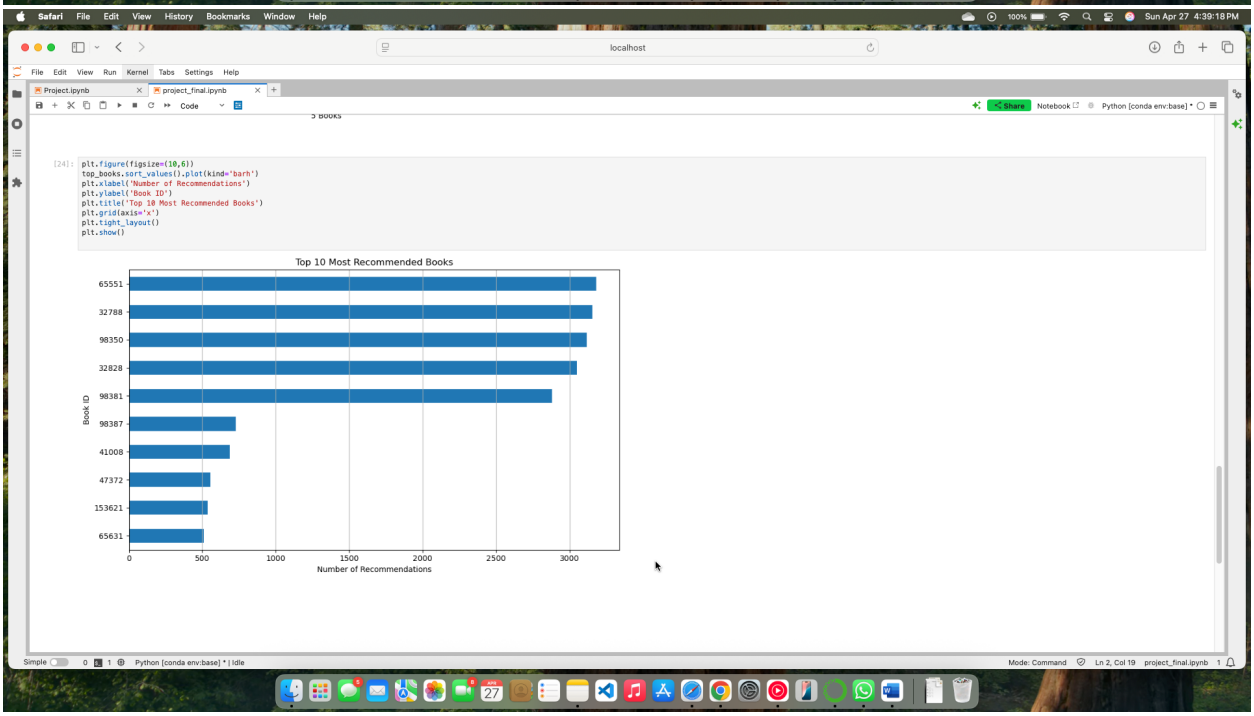

Group 9
Project 2 task 1
[1]:
import pandas as pd
import numpy as np
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import lil_matrix, csr_matrix
from tqdm import tqdm
[3]:
# Load ratings.libsvm and build user-book matrix
libsvm_file = "ratings.libsvm"
num_users, max_book_id = 0, 0
with open(libsvm_file, 'r') as file:
for line in file:
parts = line.strip().split()[1:]
for part in parts:
book_id, _ = part.split(":")
max_book_id = max(max_book_id, int(book_id))
num_users += 1
rating_matrix = lil_matrix((num_users, max_book_id + 1))
with open(libsvm_file, 'r') as file:
for i, line in enumerate(file):
parts = line.strip().split()[1:]
for part in parts:
book_id, rating = part.split(":")
rating_matrix[i, int(book_id)] = float(rating)
[5]:
rating_matrix = rating_matrix.tocsr()
# Load Books.csv and build mappings
books_df = pd.read_csv("Books.csv", sep=';', low_memory=False)
books_df.columns = books_df.columns.str.strip()
title_col = [col for col in books_df.columns if "title" in col.lower()]
if not title_col:
raise ValueError("No title column found.")
title_col = title_col[0]
isbn_to_title = dict(zip(books_df['ISBN'], books_df[title_col]))
book_id_to_isbn = {idx + 1: isbn for idx, isbn in enumerate(books_df['ISBN'])}
# Compute cosine similarity as similarity measure
similarity_matrix = cosine_similarity(rating_matrix)
[7]:
# Generate recommendations
recommendations = []
for user_idx in tqdm(range(num_users), desc="Generating Recommendations", miniters=500):
user_ratings = rating_matrix[user_idx]

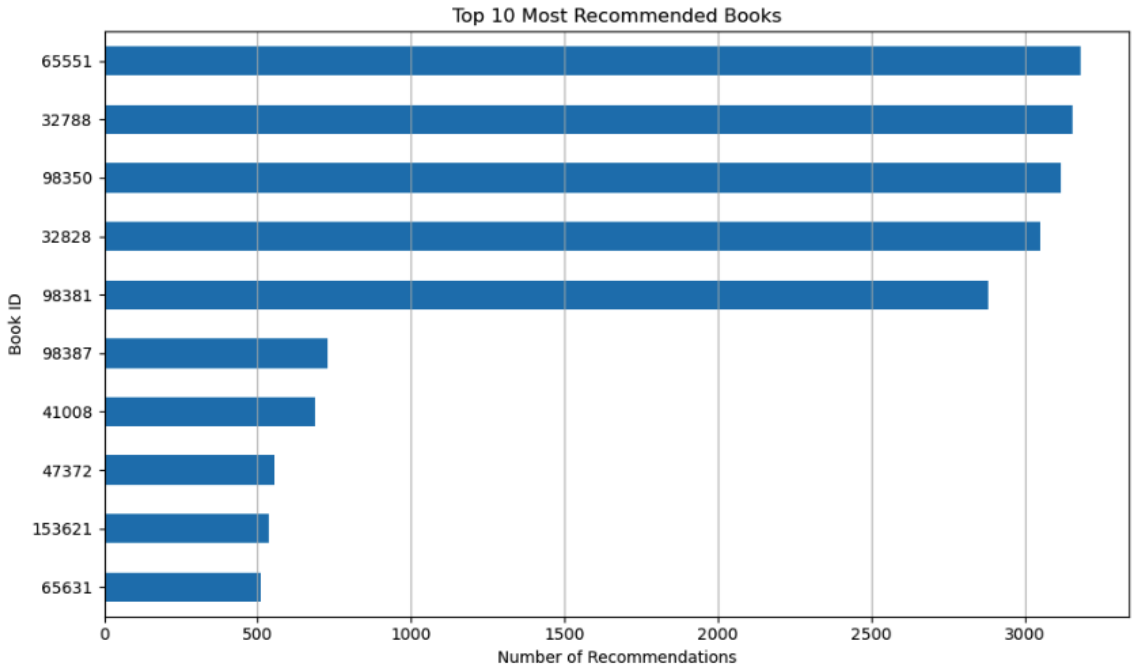

title_col = [col for col in books_df.columns if "title" in col.lower()]
if not title_col:
raise ValueError("No title column found.")
title_col = title_col[0]
isbn_to_title = dict(zip(books_df['ISBN'], books_df[title_col]))
book_id_to_isbn = {idx + 1: isbn for idx, isbn in enumerate(books_df['ISBN'])}
# Compute cosine similarity as similarity measure
similarity_matrix = cosine_similarity(rating_matrix)
[7]:
# Generate recommendations
recommendations = []
for user_idx in tqdm(range(num_users), desc="Generating Recommendations", miniters=500):
user_ratings = rating_matrix[user_idx]
user_sim = similarity_matrix[user_idx]
top_users = np.argsort(user_sim)[-11:-1][::-1]
neighbor_books = set()
for neighbor_idx in top_users:
neighbor_books.update(rating_matrix[neighbor_idx].nonzero()[1])
user_books = set(user_ratings.nonzero()[1])
recommendable_books = neighbor_books - user_books
estimated_ratings = {}
for book_idx in recommendable_books:
weighted_sum = 0
sim_sum = 0
for neighbor_idx in top_users:
rating = rating_matrix[neighbor_idx, book_idx]
if rating > 0:
weighted_sum += user_sim[neighbor_idx] * rating
sim_sum += user_sim[neighbor_idx]
if sim_sum > 0:
estimated_ratings[book_idx] = weighted_sum / sim_sum
top_books = sorted(estimated_ratings.items(), key=lambda x: -x[1])
selected_books = []
for book_idx, score in top_books:
isbn = book_id_to_isbn.get(book_idx, None)
title = isbn_to_title.get(isbn, "Unknown Title")
if title != "Unknown Title":
selected_books.append((user_idx, book_idx, title, round(score, 2)))
if len(selected_books) == 5:
break
recommendations.extend(selected_books)
Generating Recommendations: 100%| 105283/105283 [33:55<00:00, 51.71it/s]

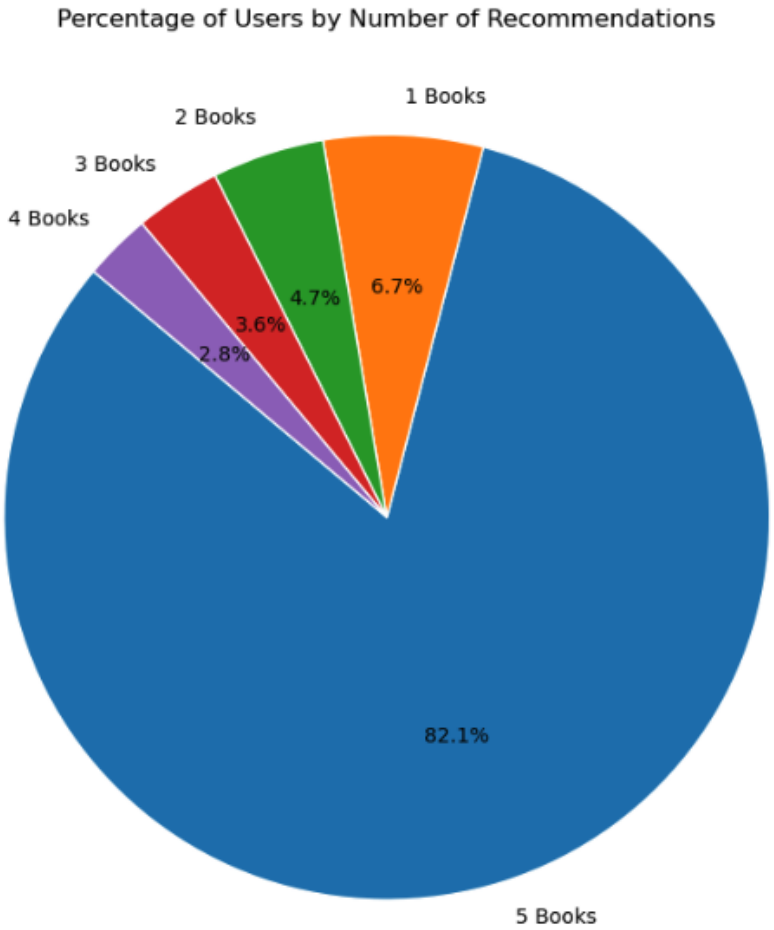

Safari File Edit View History Bookmarks Window Help
Sun Apr 27 4:37:58 PM
localhost
File Edit View Run Kernel Tabs Settings Help
Project.ipynb
project_final.ipynb
Code
Share
Notebook
Python [conda env:base]
selected_books.append((user_idx, book_idx, title, round(score, 2)))
if len(selected_books) == 5:
break
recommendations.extend(selected_books)
Generating Recommendations: 100%| | 105283/105283 [33:55<00:00, 51.71it/s]
saving to csv with sort by user ID
recommendations_df = pd.DataFrame(recommendations, columns=["User_ID", "Book_ID", "Book_Title", "Recommendation_Score"])
recommendations_df["User_ID"] = recommendations_df["User_ID"] + 1
recommendations_df = recommendations_df.sort_values(by=["User_ID", "Recommendation_Score"], ascending=[True, False])
recommendations_df.to_csv("final_recommendations.csv", index=False)
recommendations_df.shape
(203165, 4)
Visualize result
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5))
sns.histplot(df['Recommendation_Score'], bins=20, kde=True)
plt.xlabel('Predicted Recommendation Score')
plt.ylabel('Frequency')
plt.title('Distribution of Recommendation Scores')
plt.grid(True)
plt.tight_layout()
plt.show()
python(52617) MallocStackLogging: can't turn off malloc stack logging because it was not enabled.
Distribution of Recommendation Scores
Frequency
200000
175000
150000
125000
100000
75000
Simple
Python [conda env:base] | Idle
Mode: Command
Ln 2, Col 19
project_final.ipynb

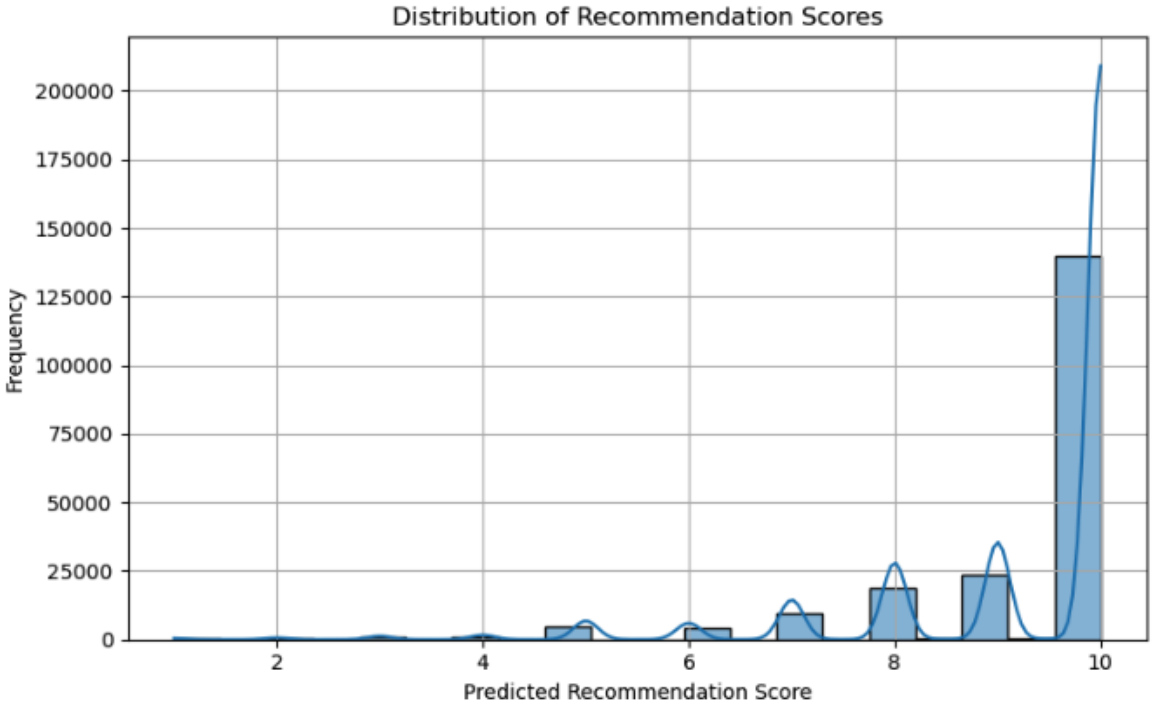

Safari File Edit View History Bookmarks Window Help
Sun Apr 27 4:38:21 PM
localhost
File Edit View Run Kernel Tabs Settings Help
Project.ipynb
project_final.ipynb
Code
Share
Notebook
Python [conda env:base]
Visualize result
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(8,5))
sns.histplot(df['Recommendation_Score'], bins=20, kde=True)
plt.xlabel('Predicted Recommendation Score')
plt.ylabel('Frequency')
plt.title('Distribution of Recommendation Scores')
plt.grid(True)
plt.tight_layout()
plt.show()
python(52617) MallocStackLogging: can't turn off malloc stack logging because it was not enabled.
Distribution of Recommendation Scores
Frequency
200000
175000
150000
125000
100000
75000
50000
25000
0
2
4
6
8
10
Predicted Recommendation Score
reco_counts = recommendations_per_user.value_counts()
labels = [f'{i} Books' for i in reco_counts.index]
sizes = reco_counts.values
plt.figure(figsize=(7,7))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, wedgeprops={'edgecolor':'white'})
plt.title('Percentage of Users by Number of Recommendations')
plt.tight_layout()
Simple
Python [conda env:base] | Idle
Mode: Command
Ln 2, Col 19
project_final.ipynb


Predicted Recommendation Score
reco_counts = recommendations_per_user.value_counts()
labels = [f'{i} Books' for i in reco_counts.index]
sizes = reco_counts.values
plt.figure(figsize=(7,7))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, wedgeprops={'edgecolor':'white'})
plt.title('Percentage of Users by Number of Recommendations')
plt.tight_layout()
plt.show()
Percentage of Users by Number of Recommendations
1 Books
2 Books
3 Books
4 Books
5 Books
6.7%
4.7%
3.6%
2.8%
82.1%


plt.figure(figsize=(10,6))
top_books.sort_values().plot(kind='barh')
plt.xlabel('Number of Recommendations')
plt.ylabel('Book ID')
plt.title('Top 10 Most Recommended Books')
plt.grid(axis='x')
plt.tight_layout()
plt.show()
Top 10 Most Recommended Books
65551
32788
98350
32828
98381
98387
41008
47372
153621
65631
Book ID
Number of Recommendations


Top 10 Most Recommended Books
65551
32788
98350
32828
98381
98387
41008
47372
153621
65631
Book ID
0
500
1000
1500
2000
2500
3000
Number of Recommendations


Percentage of Users by Number of Recommendations
1 Books
2 Books
3 Books
4 Books
5 Books
6.7%
4.7%
3.6%
2.8%
82.1%


Distribution of Recommendation Scores
Frequency
200000
175000
150000
125000
100000
75000
50000
25000
0
2
4
6
8
10
Predicted Recommendation Score