**“Project Step 2: Data Analysis”**

“Student Names and ID’s”: “Saketh Vardhan Kalashikam - 1232198359”

“Vaddi Sreeja - 1232211840”

“Eswar Reddy Kothapalli - 1231807956”

“Kavya Bijja - 1231777770”

“Arizona State University”

“IFT 511 – Analysing Big Data (2024 Fall C)”

Classes: Tuesday and Thursday

“Prof: Asmaa Elbadrawy”

Due Date – December 1st, 2024

According to the Data analysis for the TASK 1 and we worked on it.

Code:

import math

from collections import defaultdict

def load\_dataset():

SV\_BKmaps = {}

SVUserRatngs = defaultdict(dict)

with open("/Users/sakethkalashikam/Downloads/Books.csv", 'r', encoding='utf-8') as book\_file:

next(book\_file)

for index, line in enumerate(book\_file, start=1):

try:

sv\_parts = line.strip().split(';')

SV\_isbn = sv\_parts[0].strip()

SVTITle = sv\_parts[1].strip()

SV\_BKmaps[index] = {'isbn': SV\_isbn, 'title': SVTITle}

except:

continue

with open("/Users/sakethkalashikam/Downloads/user\_book\_ratings\_SampleProject1.libsvm", 'r') as rating\_file:

for user\_id, line in enumerate(rating\_file, start=1):

ratings = line.strip().split()

for rate in ratings:

book\_id, score = map(float, rate.split(':'))

SVUserRatngs[user\_id][int(book\_id)] = score

return SV\_BKmaps, SVUserRatngs

def calculate\_similarity(user\_a, user\_b, ratings):

SVCMN\_bks = set(ratings[user\_a].keys()) & set(ratings[user\_b].keys())

if not SVCMN\_bks:

return 0.0

SV\_numerator = sum(ratings[user\_a][book] \* ratings[user\_b][book] for book in SVCMN\_bks)

SVnorm\_A = math.sqrt(sum(val \*\* 2 for val in ratings[user\_a].values()))

SVnorm\_B = math.sqrt(sum(val \*\* 2 for val in ratings[user\_b].values()))

return SV\_numerator / (SVnorm\_A \* SVnorm\_B) if SVnorm\_A \* SVnorm\_B > 0 else 0.0

def recommend\_books(target\_user, ratings, SV\_BKmaps, top\_n=5):

SV\_siml\_scre = []

for other\_user in ratings:

if other\_user != target\_user:

similarity = calculate\_similarity(target\_user, other\_user, ratings)

if similarity > 0:

SV\_siml\_scre.append((other\_user, similarity))

top\_similar\_users = sorted(SV\_siml\_scre, key=lambda x: x[1], reverse=True)[:10]

recommended = defaultdict(float)

for similar\_user, similarity in top\_similar\_users:

for book\_id, score in ratings[similar\_user].items():

if book\_id not in ratings[target\_user]:

recommended[book\_id] += similarity \* score

sorted\_recommendations = sorted(recommended.items(), key=lambda x: x[1], reverse=True)

return [(book\_id, score, SV\_BKmaps.get(book\_id, {}).get('title', f"Book {book\_id}")) for book\_id, score in sorted\_recommendations[:top\_n]]

def main():

print("Data given for the Book Recommendation of the System")

print("=" \* 30)

#now we are loading the data set for the result.

SV\_BKmaps, ratings = load\_dataset()

total\_users = len(ratings)

print(f"Loaded data for {total\_users} users and {len(SV\_BKmaps)} books.")

print("executing some of the best recommendations...")

with open('recommendations.csv', 'w', encoding='utf-8') as output\_file:

output\_file.write('UserID,BookID,BookTitle,Score\n')

for count, user\_id in enumerate(ratings.keys(), start=1):

SV\_recommendations = recommend\_books(user\_id, ratings, SV\_BKmaps)

for book\_id, score, title in SV\_recommendations:

output\_file.write(f'{user\_id},{book\_id},"{title}",{round(score, 2)}\n')

print(f"Processed {count}/{total\_users} users...", end="\r", flush=True)

print("\n Recommendations which run have been successfully saved to the file 'recommendations.csv'!")

if \_name\_ == "\_main\_":

main()

1st Answer: K-Closest.

The method utilizes the cosine similarity measure to determine the K closest users to user (u). This examines similarities between users by looking at the angle among their score vectors, neglecting quantity. It guarantees that users with appropriate rating patterns are regarded as similar, making it perfect for team filtering.

According to the recommendation file: Explanation:

1. The columns contain:

UserID: the person’s ID when the ideas were generated.

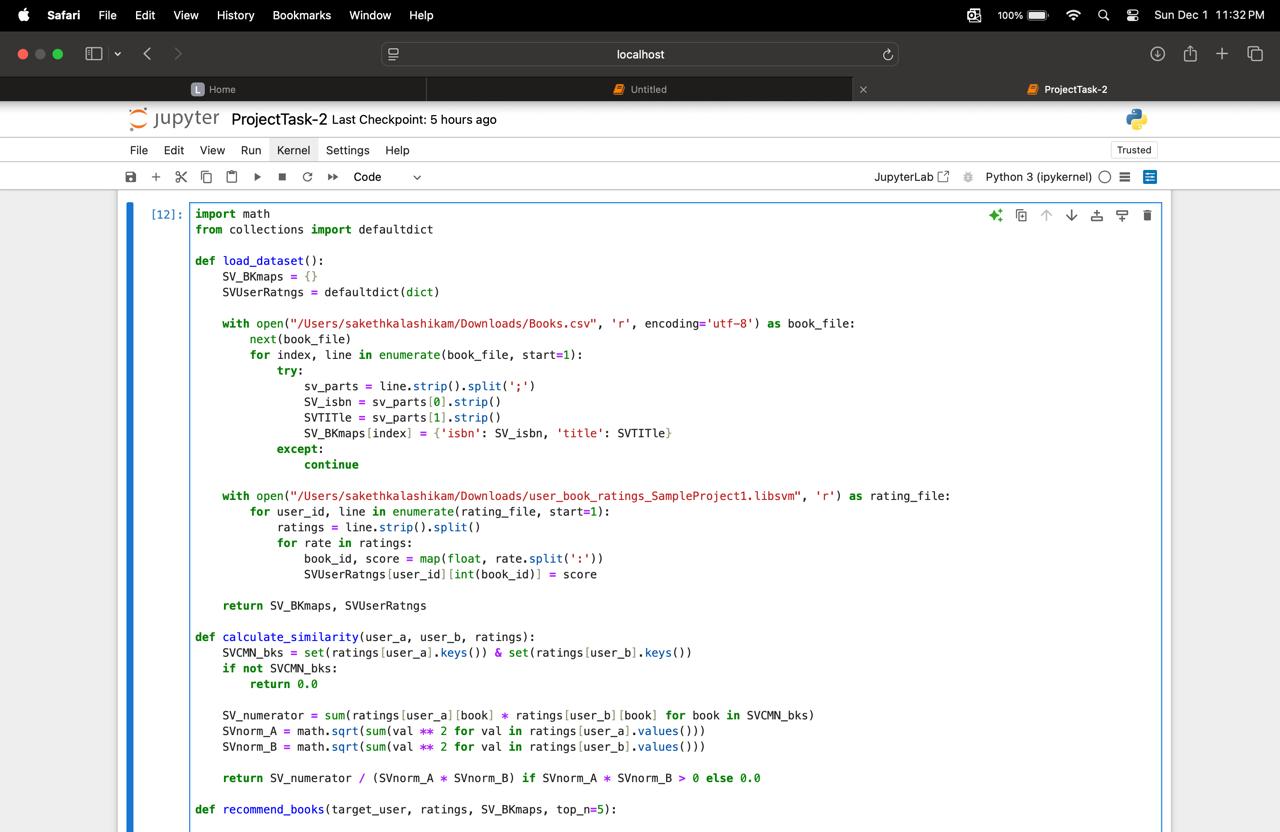
BookID: the unique number for the indicated book.

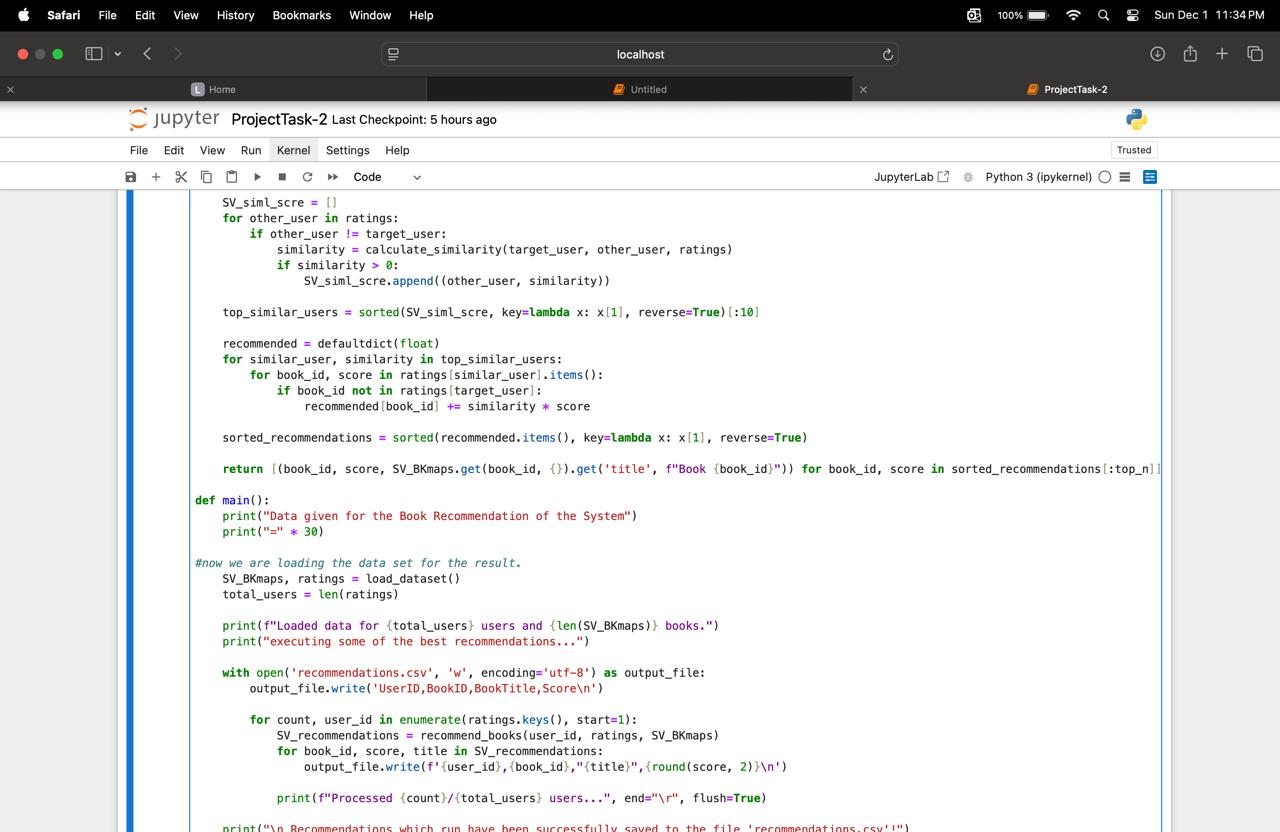
Book Title: The title that is part of the selected book.

Score: A balanced recommendation score determined by user connections and ratings.

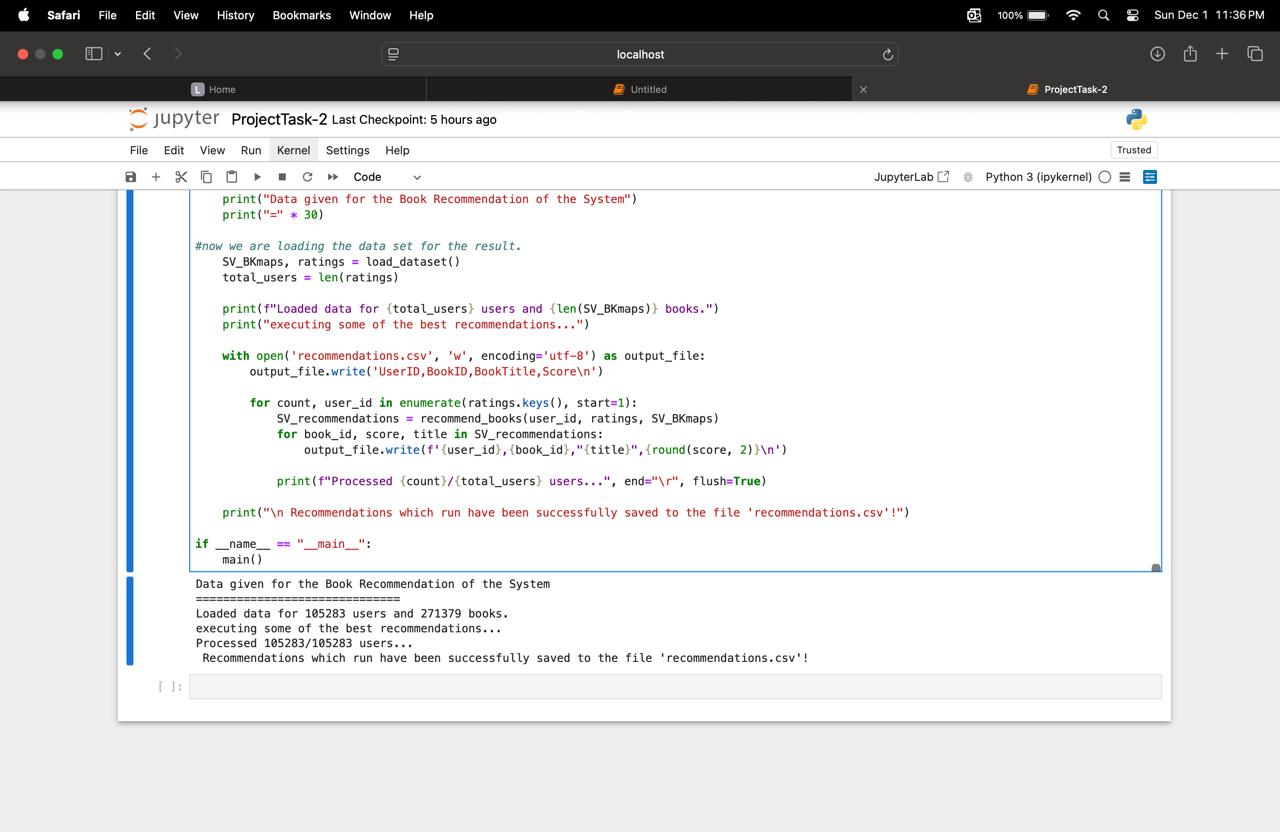
1. The approach generates the top ideas for every user (UserID) based on the evaluations of the 10 most comparable users.
2. The cosine similarities metric is used to figure out how comparable individuals are based on their book ratings.
3. Preferred books are those who received excellent reviews from similar users but still need to be reviewed by this particular user.
4. The Score is a ranked average of assessments handed over by comparable users, calculated for similarity to that particular user.
5. As an example for UserID 6, the book "Passenger to Frankfurt" had the greatest recommendation score of 2.8.
6. Certain titles (e.g., BookID 8764 for UserID 6) receive a 0.0 score, suggesting they weren't highly recommended to the user.
7. The ideas have been saved in a file called recommendations.csv, which was included in the code.
8. The method used repeats over every consumer in the dataset, producing specific suggestions for each.
9. The methodology is extendable for enormous files since it easily computes relevance and scoring via matrix algorithms.

Code and Result Screenshots:





“Please check the uploaded CSV file.”



Output Screenshots:

