READILY

# A mini project report submitted for the partial fulfilment of the requirement of the award of the degree of

**MASTER OF COMPUTER APPLICATIONS**

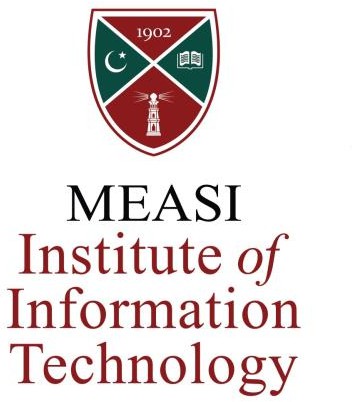
Submitted by

**FAMITHA M**

**(Reg. No: 812200342)**

Under the Guidance of

**DR. SARADHA S, M.Sc., M.Phil.,Ph.D**



**MEASI INSTITUTE OF INFORMATION TECHNOLOGY**

(Approved by AICTE & Affiliated to the University of Madras)

CHENNAI-600014

**BONAFIDE CERTIFICATE**

This is to certify that the Mini Project entitled

**READILY**

Being submitted to the University of Madras, Chennai

By

**FAMITHA M**

For the Partial fulfilment for the award of the degree of

**MASTER OF COMPUTER APPLICATIONS**

**PROJECT GUIDE HEAD OF THE DEPARTMENT**

**Dr. Saradha S, M.Sc., M. Phil., Ph.D Dr .Usha M, M.C.A., M.Tech., M.Phil., Ph.D.**

Assistant Professor Assistant Director & HOD

**Submitted for the Viva-voce Examination held on**

**Examiners**

**DATE :** **1.**

**PLACE :** **2.**

# ACKNOWLEDGEMENT

First of all, I must be thankful to the **Almighty** for showering blessing throughout my life. I would like to express my heartfelt thanks to my beloved parents who sacrifices their presence for my better life.

I wish to express my sincere gratitude to **Dr.Ahamed Mudassar Ali, M.E., Ph.D.,** Director, **MEASI Institute of Information Technology, Chennai** for his constant encouragement throughout my course of study.

A humble word of gratitude, affection and regard for my Assistant Director cum HoD , **Dr. USHA M, M.Tech., Ph.D.,** for her support and guidance.

I am thankful to my Project Guide **Dr. SARADHA S, M.Sc., M.Phil., Ph.D.** who has been helping me a lot for the completion of this Project. Her guidance and tips for life will always go a long way in the minds of all the students. Her guidance, encouragement and optimistic attitude has been very inspiring.

At this time, I thank all the staff of the Department of M.C.A for their valuable help and who directly or indirectly contributed to my project development.

I convey my gratitude to the lab administrators for considering the requirements for my project and satisfying them whenever I required them. It would have been highly impossible to carry forward the project without their assistance.

Due credits also go to my friends and well-wishers who uplifted and encouraged me to see the venture to its finish.

**FAMITHA M**

# ABSTRACT

Now-a-days, everyone depending on reviews by others in many things such as selecting a movie to watch, buying products, reading a book. Recommender systems are used for that purpose only. A recommender system is a kind of filtering system that predicts a user's rating of an item. Recommender systems recommend items to users by filtering through a large database of information using a ranked list of predicted ratings of items.

Book recommender system is a recommender system for ones who love books. This system takes important data such as genre, subject of the book from the Dataset to recommend similar books based on the user preferences. content-based filtering approach is used here.

The Python implementation utilizes libraries like NumPy and Pandas for data manipulation, and scikit-learn for machine learning algorithms. Web frameworks such as Flask or Django can be utilized to create a user-friendly interface for the book recommender system.

Through this book recommender system, users can discover new books tailored to their interests and reading habits, creating a more enjoyable and personalized reading experience.

**TABLE OF CONTENTS**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.NO** | **CONTENT** | | | **PAGE NO** |
| **1.** | **INTRODUCTION** | | | 1 |
| **2.** | **SYSTEM ANALYSIS** | | |  |
| 2.1 | Problem Definition | | 2 |
| 2.2 | Existing System | | 3 |
| 2.3 | Proposed System | | 5 |
| **3.** | **REQUIREMENT SPECIFICATION** | | |  |
| 3.1 | Hardware Specification | | 7 |
| 3.2 | Software Specification | | 7 |
| **4.** | **DESIGN AND IMPLEMENTATION** | | |  |
| 4.1 | Module Description | | 9 |
| **5.** | **TESTING** | | |  |
| 5.1 | | Introduction | 18 |
| 5.2 | | Types of testing | 18 |
| 5.3 | | Testing cases | 22 |
| 5.4 | | Testing Data | 22 |
| 5.5 | | Testing Report | 22 |
| **6.** | **CONCLUSION AND FUTURE ENHANCEMENT** | | | 23 |
| **7.** | **APPENDICES** | | |  |
| 7.1 | Sample Code | | 24 |
| 7.2 | Sample Screenshots | | 35 |
| 8. | **BIBLOGRAPHY** | | |  |
| 8.1 | Reference | | 45 |

|  |  |  |
| --- | --- | --- |
|  | **List of Figures** | **Page No** |
| 4.1 | Project Representation | 9 |
| 4.2 | Module 1: Front End | 11 |
| 4.3 | Module 2: Connectivity | 13 |
| 4.4 | Module 3: Dataset | 15 |
| 4.5 | Module 3: Backend | 16 |
|  | **List Of Tables** |  |
| 2.1 | Existing System | 3 |
| 2.2 | Proposed System | 5 |
| 5.1 | Test Case | 19 |

# 

# CHAPTER - 1

# INTRODUCTION

**1.1 ABOUT THE PROJECT:**

In this project, we present a 'Personalized Book Recommender System: Enhancing Your Reading Experience.' This system leverages content-based filtering techniques and utilizes Python libraries like NumPy and Pandas for data manipulation, along with scikit-learn for machine learning algorithms. With a user-friendly interface created using web frameworks such as Flask or Django, this book recommender system assists users in discovering new books tailored to their interests and reading habits, ultimately creating a more enjoyable and personalized reading experience.

# CHAPTER-2

**SYSTEM ANALYSIS**

**2.1 PROBLEM DEFINITION:**

In the digital age, where an abundance of books is readily available, finding the right book that suits an individual's preferences and interests can be a daunting task. A personalized book recommender system is a solution that leverages machine learning and data analysis to help users discover books that align with their unique tastes and needs. The primary objective of this project is to design and develop an intelligent book recommender system that offers tailored book recommendations to users, ultimately enhancing their reading experience.

Readily is a book recommender system which allows users to search books by book name and keywords. Each user has a Unique User Credentials

**2.2 EXISTING SYSTEM:**

Table 2.1 : Existing System

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S.N** | **Name** | **Front End** | **Backend** | **Source** | **Description** | **Drawback** |
| 1 | Brain L | HTML | SQL | <https://arxiv.org/abs/2302.00653> | Basic Book Recommender System that just shows the book names | Not Pleasant and it is not user friendly |
|  |
| 2 | Which  book | HTML/ CSS | JavaScript and SQL | <https://www.whichbook.net/> | A book Recommender system which shows, book name in a order with book covers | It does not allow the users to search manually |  |
| 3 | Gutnberg | HTML/ CSS | JavaScript and SQL | <https://www.gutenberg.org/> | A book Recommender system which shows, book name in a order with book covers | It does not show the relevant recommendation |  |

"Brain L," utilizes HTML for its front-end and SQL for its backend. It offers a basic book recommendation service that displays book names. However, it has limitations, such as not being very user-friendly and lacking an aesthetically pleasing design.

"Which book," employs HTML/CSS for its front-end and JavaScript and SQL for its backend. This recommender system arranges book names in a specific order along with book covers. One notable drawback is that it doesn't provide users with the option to manually search for books, potentially limiting user interaction.

"Gutenberg," also uses HTML/CSS for the front-end and JavaScript and SQL for the backend. It presents book names in a certain order along with book covers. However, it falls short in terms of offering relevant recommendations, which may hinder its effectiveness as a book recommender.

**2.3 PROPOSED SYSTEM:**

Table 2.2 : Proposed System

|  |  |
| --- | --- |
| **Name** | Readily |
| **Description** | A book recommender system using Machine Learning which recommends book based on user preferences such as book name and keyword |
| **Front End** | Python (Streamlit) and CSS |
| **Backend** | Python (Jupyter Notebook) |
| **Proposed Features** | 1.A pleasant user-friendly environment 2.Users can search manually 3.Recommends a relevant book, if the particular book cannot be recommended, this system recommends same genre books |

The "Readily" book recommender system is an exciting and promising project that leverages Machine Learning to offer users a dynamic and personalized book recommendation experience. It goes beyond the systems mentioned in the previous table by incorporating several noteworthy features. One of the standout attributes of "Readily" is its focus on creating a pleasant and user-friendly environment.

This is a significant positive comment about the project as it aims to ensure that users find the system engaging and enjoyable to use. With a user-friendly interface, individuals are more likely to have a satisfying interaction with the recommender, making it a practical choice for book enthusiasts. Additionally, the project addresses the limitation of manual search capabilities found in some other systems.

Users can manually search for books, allowing for a more tailored and interactive experience. This feature enhances user autonomy, making it easier for them to discover specific titles of interest. Furthermore, "Readily" aims to provide relevant book recommendations based on user preferences, such as book names and keywords.

This sets it apart from systems that may struggle to offer meaningful suggestions. The project's dedication to recommending books in the same genre when a specific title cannot be recommended demonstrates its commitment to providing valuable and contextually relevant suggestions.

In an effort to keep the recommendation content fresh and up-to-date, the project takes the innovative step of implementing a weekly update system. By offering the top 20 books every week, "Readily" ensures that users have access to the latest and most popular titles, keeping the recommender system both informative and timely.

# 

# CHAPTER – 3

# REQUIREMENT SPECIFICATION

**3.1 HARDWARE SPECIFICATION:**

|  |  |  |
| --- | --- | --- |
| * PROCESSOR | : | PROCESSOR WITH 1.7 – 2.4 GHz |
| * RAM | : | 8 GB |
| * SYSTEM TYPE | : | 64-BIT OPERATING SYSTEM, X-64 BASED PROCESSOR |

**3.2 SOFTWARE SPECIFICATION:**

* TEXT EDITOR (JUPITER NOTEBOOK).
* ANACONDA DISTRIBUTION PACKAGE (PYCHARM EDITOR) PYTHON LIBRARIES.
* LATEST VERSION OF ANY WEB BROWSER.

**SUBLIME TEXT:**

**CSS**

In our project, we're developing a web application using Python as the primary programming language for both the front-end and back-end components. The front-end is whsat users interact with directly, responsible for displaying content and handling user input. For this part, we might use web frameworks like Flask or Django, along with HTML templates and CSS for styling. Sublime Text serves as our code editor for writing and managing the front-end code.

**Python**

On the back-end, Python takes care of the server-side operations, including managing databases, handling user authentication, and processing requests made by the front-end. Here, we employ Python frameworks like Flask or Django to streamline the development process. Libraries such as SQLAlchemy are used for working with databases, and REST frameworks facilitate the creation of APIs.

PyCharm, an integrated development environment (IDE), is an excellent choice for managing the back-end part of the project. It provides features like code auto-completion, debugging tools, and version control integration, making back-end development more efficient.

In some cases, our project might involve data analysis or machine learning. For these data-related tasks, we utilize Jupyter Notebook, an interactive environment that allows us to work with data, visualize results, and create prototypes effectively.

To sum it up, our project leverages Python's versatility to create a comprehensive web application, and it's supported by various development tools and environments. Sublime Text, PyCharm, and Jupyter Notebook play essential roles in different aspects of the project, contributing to its efficiency and success.

# CHAPTER – 4

# SYSTEM DESIGN

**4.1. Project Representation**

****

Fig 4.1 : Project Representation

**Main Class:**

It has two private attributes: books and df, both of type DataFrame, which are used to store data.

It has a public method main(), indicating the program's entry point.

**CustomRecommender Class:**

it interacts with the DataFrame (df) to provide custom book recommendations.

It contains a private attribute df and a public method custom\_recommender (book\_title: str) for generating recommendations.

**ContentBasedRecommender Class:**

This class is involved in content-based book recommendations.

It has a public method content\_based\_recommender (book\_title: str, df: DataFrame) for generating recommendations based on book titles and the DataFrame.

**Relationships:**

The Main class has a composition relationship with the CustomRecommender class, indicating that the Main class contains an instance of the CustomRecommender class.

The CustomRecommender class, in turn, has a composition relationship with the ContentBasedRecommender class, suggesting that it contains an instance of the ContentBasedRecommender class.

All these classes are connected to the DataFrame class, which suggests that they interact with or use DataFrame objects.

Overall, this diagram shows a high-level view of the classes and their relationships in a program designed for book recommendations, where different classes are responsible for various aspects of the recommendation process.

The Main class serves as the program's entry point, while CustomRecommender and ContentBasedRecommender classes handle specific recommendation logic.

**Module 1**

**Front End Implementation**

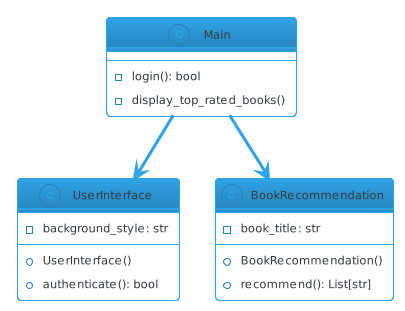


Fig 4.2 : Front End Implementation

**Main Class:**

Represents the central part of the program.

It contains the 'books' and 'df' attributes, which store DataFrames.

The 'main()' method is the program's entry point, responsible for executing the application.

This class manages user authentication and interaction with the recommendation system.

**UserInterface Class:**

Handles the user interface elements and styling.

Contains an authentication method, which is likely responsible for verifying and managing user logins.

**CustomRecommender Class:**

Specialized in generating custom book recommendations.

Utilizes the 'df' DataFrame to provide personalized book suggestions based on user input.

ContentBasedRecommender Class:

**A sub-component of the CustomRecommender.**

Focused on content-based recommendation by analyzing book titles, authors, publishers, and categories.

This class is essential for enhancing the recommendation system's effectiveness.

**Interactions:**

The 'Main' class connects to the 'CustomRecommender' and 'ContentBasedRecommender' classes, indicating that it relies on these components for its functionality.

The 'CustomRecommender' class, in turn, uses the 'ContentBasedRecommender' class to provide custom book recommendations.

Both the 'CustomRecommender' and 'ContentBasedRecommender' classes interact with the 'DataFrame' class, likely to access and manipulate data.

In essence, this UML diagram depicts a program that offers book recommendations to users. It uses data stored in DataFrames and applies custom recommendation logic, including content-based methods, to suggest books based on user input and preferences. The 'UserInterface' class manages the user experience and authentication.

**Module 2**

**Connectivity Implementation:**

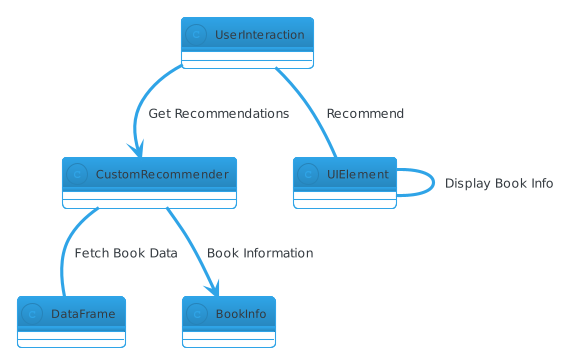


Fig 4.3 : Connectivity Implementation

**UserInteraction:**

Manages user interactions within the application.

It's responsible for initiating the book recommendation process when the user clicks the "Recommend" button.

**CustomRecommender:**

Handles the book recommendation logic.

This class selects and suggests books based on various criteria or algorithms.

**UIElement:**

An abstract class representing different elements in the user interface.

It encompasses elements such as buttons, images, and text displayed on the user interface.

These elements contribute to the overall user experience.

**DataFrame:**

Represents a data structure for organizing and processing tabular data.

In this context, it is likely used to manage and store book-related data, which is essential for the recommendation process.

**BookInfo:**

Represents detailed information about a book.

It includes attributes like cover image, summary, and rating, allowing for comprehensive book details to be displayed to the user.

These UML elements collectively form an application that handles user interactions, provides book recommendations, manages the user interface elements, and utilizes DataFrames to process and organize book data. The 'CustomRecommender' class plays a central role in determining book recommendations, while 'UIElement' represents various UI components. 'DataFrame' is the backbone for managing data, and 'BookInfo' offers detailed book-specific information for display.

**Module 3**

**Backend Implementation**

**Dataset**

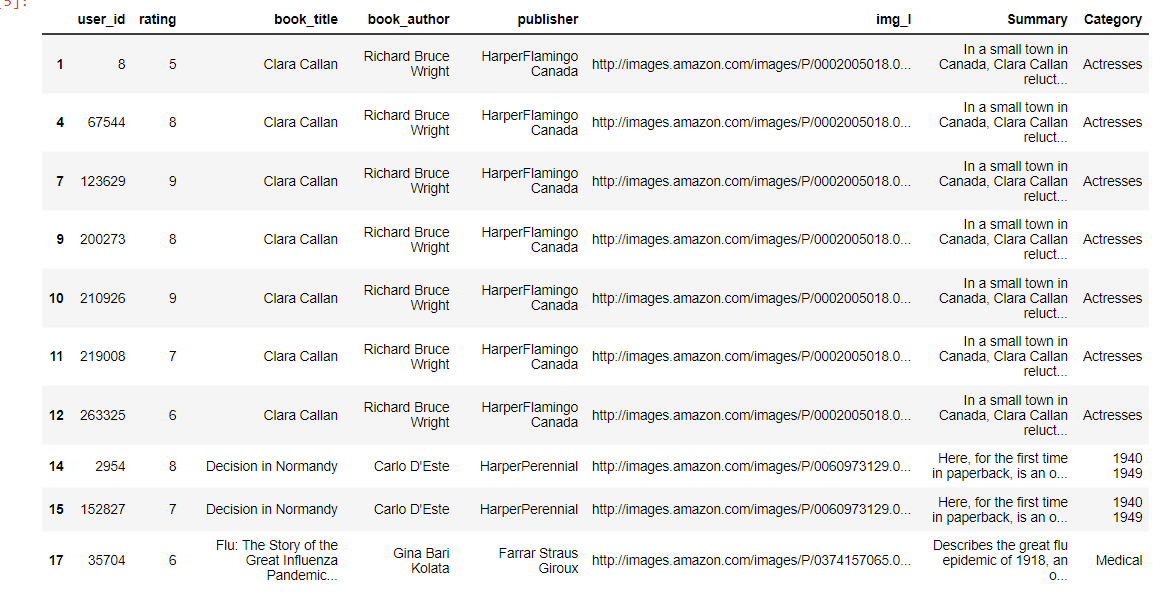
****

Fig 4.4 : Dataset

# 

The dataset contains 73073 Books. Every Book has a Unique ID used to fetch later on in

the recommender. It has fields like “book\_title”, ”Summary”, ”Category”, ”Book\_author”

which are used in the recommendation algorithm.

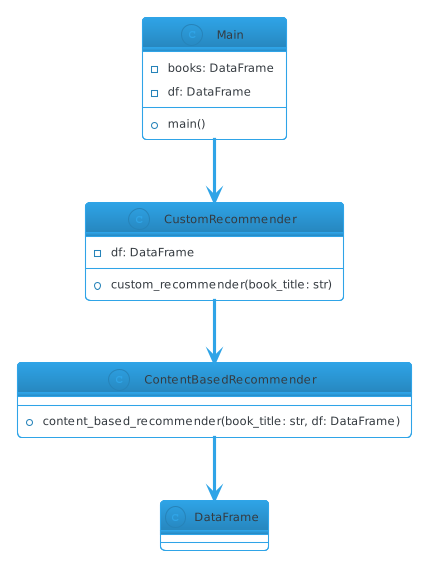


Fig 4.5 : Backend

**Main Class (Main):**

**Description:** The 'Main' class represents the central component of your program, serving as the main entry point.

**Attributes:**

'books' and 'df' are private attributes, both of which are instances of the 'DataFrame' class. These attributes likely hold data related to books, possibly including details such as titles, authors, summaries, and ratings.

**Methods:**

' main()': This method is the starting point of your program's execution. It orchestrates the entire application, managing user interactions, authentication, and book recommendations.

**CustomRecommender Class (CustomRecommender):**

Description: The 'CustomRecommender' class is responsible for providing custom book recommendations to users.

Attributes:

'df' is a private attribute, representing an instance of the 'DataFrame' class. It likely contains structured data related to books, which is essential for the recommendation process.

**Methods:**

'custom\_recommender(book\_title: str)': This method takes a 'book\_title' as a parameter, likely a user-selected book. It provides custom book recommendations using a combination of item-based and content-based approaches.

**ContentBasedRecommender Class (ContentBasedRecommender):**

Description: The 'ContentBasedRecommender' class specializes in content-based book recommendations.

Attributes: This class doesn't have any specific attributes, but it may utilize data from a 'DataFrame' for its operations.

Methods:'content\_based\_recommender(book\_title: str, df: DataFrame)': This method takes a 'book\_title' and a 'DataFrame' as parameters. It generates content-based book recommendations for the given 'book\_title' by analyzing the content, likely including attributes such as title, author, publisher, and category.

In summary, the 'Main' class acts as the program's core, managing data and user interactions. It employs the 'CustomRecommender' class to provide custom book recommendations, which, in turn, uses the 'ContentBasedRecommender' class to generate content-based suggestions. All these classes work together to offer a personalized book recommendation system.

# CHAPTER – 5

**TESTING**

**5.1 INTRODUCTION:**

During systems testing, the system is used experimentally to ensure that the software does not fail. In other words, we can say that it will run according to its specifications and in the way users expect. Special test data are input for processing and the results examined. A limited number of users may be allowed to use the system so that analyst can see whether they try to use it in unforeseen ways.

**5.2 TYPES OF TESTING:**

* Unit Testing
* System Testing
* Integration Testing
* Validation testing
* Output testing
* User acceptance testing

**Unit Testing:**

Unit testing focuses verification effort on the smallest unit of software design – the module. Using the detail design description as a guide, important control paths are tested to uncover errors within the boundary of the module. The relative complexity of tests and the errors detected as a result is limited by the constrained scope established for unit testing. The unit test is always white box oriented, and the step can be conducted in parallel for multiple modules.

Table 5.1 : Test Case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case Ref No** | | **TCT-001** | | |
|  | |  | | |
| Functionality | | Log in to the system | | |
|  | |  | | |
| Expected outcome | | The user should not login to member’s area and some error message follow | | |
|  | |  | | |
| **Step No** | | **Data Used** | | **Actual Outcome** |
| 1. | | Click on the log in button without entering username or password. | | An alert message came to enter Username |
| 2. | | Click on the log in button after entering some username leaving password field blank | | An alert message came to enter password |
| 3. | | Click on the log in button after entering some password but leaving username field blank | | An alert message came to enter username |
| 4. | | Click on the log in button after entering some wrong username but correct password | | A message displayed on Log in page about this |
| **Test Case Ref No** | **TCT-002** | | | |
| Functionality | Enter valid data for user registration | | | |
| Expected outcome | The user should not get register any record without filling all necessary fields and some error message follow: The user should not get registered again with same user id | | | |
|  |  | | | |
| **Step No** | **Data Used** | | **Actual Outcome** | |
| 1. | Click on the save button without entering valid details | | An alert message came to each details and focused on the respective fields | |
| 2. | Click on the submit button after entering a duplicate user id | | A message displayed about existence of such user | |

**System Testing:**

Software is only one element of a larger computer based system. Ultimately, software is incorporated with other system elements (e.g. new hardware, information), and a series of system integration and validation tests are conducted. Steps taken during software design and testing can greatly improve the probability of successful software integration in the larger system.

**Integration Testing:**

Integration testing is a systematic technique for construction the program structure while at the same time conduction test to uncover errors associated with interfacing. The objective is to take unit tested modules and build a program structure that has been dictated by design. Integration testing can be categorized into two types, namely top-down integration or bottom-up integration. Top-down integration is an incremental approach to the construction of program structure. The selection of an integration strategy depends upon software characteristic and, sometime project schedule. In general, a combined approach that uses the top-down strategy for the upper levels of the program structure, coupled with a bottom-up strategy for the subordinate levels, may be the best compromise.

**Validation Testing:**

This provides the final assurance that the software meets all functional, behavioral and performance requirements. The software is completely assembled as a package. Validation succeeds when the software functions in manner in which the user expects. Validation refers to a processor using software in a live environment in order to find errors. During the course of validating the system, failures may occur and sometimes the coding has to be changed according to the requirement. Thus, the feedback from the validation phase generally produces changes in the software. Once the application was made free of all logical and interface errors, inputting dummy data ensured that the software developed satisfied all the requirements of the user.

**Output Testing:**

After performing the validation testing, the next step is output testing of the proposed system since no system could be useful if it does not produce the required output generated or considered in two ways; one is on screen and another is printed format. The output format on the screen is found to be correct as the format was designed in the system design phase according to the user's needs. For the hard copy also, the output comes out as the specified requirements by the user hence output testing does not result in any correction in the system.

**User Acceptance Testing:**

User acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes whenever required. Preparation of test data plays a vital role in the system testing. After preparing the test data the system under study is tested using the test data. While testing the system by using test data errors are again uncovered and the corrections are also noted for future use.

**5.3 TESTING CASES:**

During test cases that are good at revealing the presence of faults is central to successful testing. The reason for this is that if there is a fault in the program, the program can still provide the certain inputs. Only for the set of inputs the faults that exercise the fault in the program will the output of the program devise from the expected behavior. Hence, it is fair to say that testing is as good as its test case. The number of test cases used to determine errors in the program should be minimum. There are two fundamental goals of a practical testing activity:

* Maximize the numbers of errors detected and
* Minimize the number of test cases

As these two goals are contradictory so the problem of selecting test cases is a complex one.

**5.3 TEST DATA:**

After preparing the test data the system under study was tested using test data. While testing the system by using test data, errors were again uncovered and corrected by using the above testing steps. Preparation of Test data plays a vital role in the system testing. Taking various types of test data does all the above testing.

**5.4 TESTING EXECUTION:**

Test data was prepared which were the acknowledgement details and the information regarding the various departments in the case. An already existing file was taken from the database and the data was fed into the new system. Various tests as mentioned above were carried out. Initially there were bugs and drawbacks for the user to complete the same process. Those bugs and drawbacks were noted down and modified later. Again, the same process was repeated three to four times.

# CHAPTER – 6

# CONCLUSION AND FUTURE ENHANCEMENT

**6.1 CONCLUSION:**

Creating a book recommender system with machine learning means using technology to suggest books that people are likely to enjoy. These recommendations are based on how people have interacted with books before and details about the books themselves. The goal is to help readers discover new books they might like.

The system works better when it has good information about readers and books. Choosing the right methods is important, and it's crucial to keep checking how well the system is doing and get feedback from users to make it even better. Overall, this kind of system can make it easier for people to find and enjoy books, which is great for readers and the book industry.

**6.2 FUTURE ENHANCEMENT:**

Transform your website into a Transform your website into a mobile app, offering users a more accessible way to discover books. Enhance the app by introducing user profiles for personalized recommendations and mobile app, offering users a more accessible way to discover books.

Enhance the app by introducing user profiles for personalized recommendations and tering to users on the move, and incorporate user reviews, encouraging a sense of community and facilitating book discussions among avid readers.

# CHAPTER-7

**APPENDICES**

**7.1 CODING**

Dataset

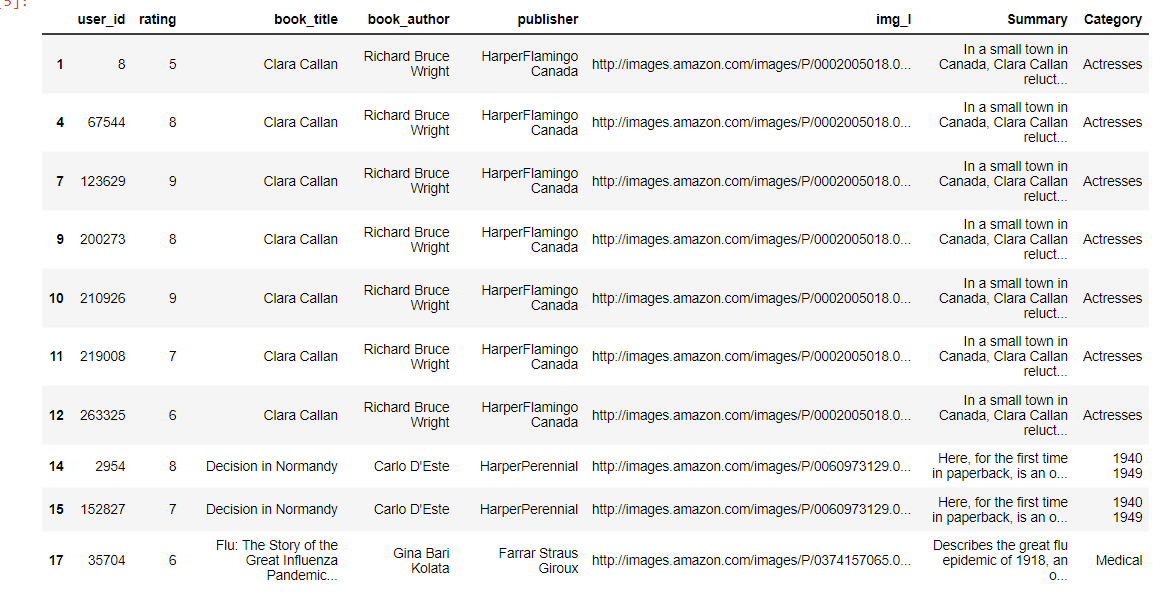


Fig 7.1 : Dataset

**Model.py**

import os

import re

import nltk

import requests

import warnings

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

%matplotlib inline

import io

import urllib

import traceback

from nltk.corpus import stopwords

nltk.download("stopwords")

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

from PIL import Image

warnings.filterwarnings('ignore')

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

books = pd.read\_csv('Preprocessed\_data.csv')

books.head(3)

df = books.copy()

df.dropna(inplace=True)

df.reset\_index(drop=True, inplace=True)

df.drop(columns = ['Unnamed: 0','location','isbn',

'img\_s','img\_m','city','age',

'state','Language','country',

'year\_of\_publication'],axis=1,inplace = True) #remove useless cols

df.drop(index=df[df['Category'] == '9'].index, inplace=True) #remove 9 in category

df.drop(index=df[df['rating'] == 0].index, inplace=True) #remove 0 in rating

df['Category'] = df['Category'].apply(lambda x: re.sub('[\W\_]+',' ',x).strip())

df.head(2)

def content\_based\_recommender(book\_title):

book\_title = str(book\_title)

if book\_title in df['book\_title'].values:

rating\_counts = pd.DataFrame(df['book\_title'].value\_counts())

rare\_books = rating\_counts[rating\_counts['book\_title'] <= 100].index

common\_books = df[~df['book\_title'].isin(rare\_books)]

if book\_title in rare\_books:

random = pd.Series(common\_books['book\_title'].unique()).sample(2).values

print('There are no recommendations for this book.')

print('Try: \n')

print('{}'.format(random[0]), '\n')

print('{}'.format(random[1]), '\n')

else:

common\_books = common\_books.drop\_duplicates(subset=['book\_title'])

common\_books.reset\_index(inplace=True)

common\_books['index'] = [i for i in range(common\_books.shape[0])]

target\_cols = ['book\_title', 'book\_author', 'publisher', 'Category']

common\_books['combined\_features'] = [' '.join(common\_books[target\_cols].iloc[i,].values) for i in range(common\_books[target\_cols].shape[0])]

cv = CountVectorizer()

count\_matrix = cv.fit\_transform(common\_books['combined\_features'])

cosine\_sim = cosine\_similarity(count\_matrix)

index = common\_books[common\_books['book\_title'] == book\_title]['index'].values[0]

sim\_scores = list(enumerate(cosine\_sim[index]))

sorted\_sim\_scores = sorted(sim\_scores, key=lambda x: x[1], reverse=True)[1:6]

recommended\_books = []

for i in range(len(sorted\_sim\_scores)):

book\_index = sorted\_sim\_scores[i][0]

if book\_index != index: # Exclude the input book

recommended\_books.append((common\_books.loc[book\_index, 'book\_title'], common\_books.loc[book\_index, 'img\_l']))

fig, axs = plt.subplots(1, 5, figsize=(18, 5))

fig.suptitle('You may also like these books', size=22)

for i in range(len(recommended\_books)):

book\_title, url = recommended\_books[i]

try:

opener = urllib.request.URLopener()

opener.addheader('User-Agent', 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36')

response = opener.open(url)

im = Image.open(io.BytesIO(response.read()))

axs[i].imshow(im)

axs[i].axis("off")

axs[i].set\_title(book\_title, y=-0.18, color="red", fontsize=18)

except Exception as e:

print(f"An error occurred: {e}")

plt.show()

else:

print('Can\'t find the book in the dataset. Please check the spelling.')

content\_based\_recommender('Wild Animus')

import pickle

with open('content\_based\_recommender.pkl', 'wb') as file:

pickle.dump(content\_based\_recommender, file)

with open('df.pkl', 'wb') as file:

pickle.dump(df, file)

def custom\_recommender(book\_title):

# ITEM-BASED

book\_title = str(book\_title)

if book\_title in df['book\_title'].values:

rating\_counts = pd.DataFrame(df['book\_title'].value\_counts())

rare\_books = rating\_counts[rating\_counts['book\_title'] <= 180].index

common\_books = df[~df['book\_title'].isin(rare\_books)]

if book\_title in rare\_books:

random = pd.Series(common\_books['book\_title'].unique()).sample(2).values

print('There are no recommendations for this book')

print('Try:\n')

print('{}'.format(random[0]), '\n')

print('{}'.format(random[1]), '\n')

else:

user\_book\_df = common\_books.pivot\_table(index=['user\_id'],

columns=['book\_title'], values='rating')

book = user\_book\_df[book\_title]

recom\_data = pd.DataFrame(user\_book\_df.corrwith(book). \

sort\_values(ascending=False)).reset\_index(drop=False)

if book\_title in [book for book in recom\_data['book\_title']]:

recom\_data = recom\_data.drop(recom\_data[recom\_data['book\_title'] == book\_title].index[0])

low\_rating = []

for i in recom\_data['book\_title']:

if df[df['book\_title'] == i]['rating'].mean() < 5:

low\_rating.append(i)

if recom\_data.shape[0] - len(low\_rating) > 5:

recom\_data = recom\_data[~recom\_data['book\_title'].isin(low\_rating)]

recom\_data = recom\_data[0:1]

recom\_data.columns = ['book\_title', 'corr']

recommended\_books = []

for i in recom\_data['book\_title']:

recommended\_books.append(i)

df\_new = df[~df['book\_title'].isin(recommended\_books)]

# CONTENT-BASED (Title, Author, Publisher, Category)

rating\_counts = pd.DataFrame(df\_new['book\_title'].value\_counts())

rare\_books = rating\_counts[rating\_counts['book\_title'] <= 100].index

common\_books = df\_new[~df\_new['book\_title'].isin(rare\_books)]

common\_books = common\_books.drop\_duplicates(subset=['book\_title'])

common\_books.reset\_index(inplace=True)

common\_books['index'] = [i for i in range(common\_books.shape[0])]

target\_cols = ['book\_title', 'book\_author', 'publisher', 'Category']

common\_books['combined\_features'] = [' '.join(common\_books[target\_cols].iloc[i, ].values) for i in

range(common\_books[target\_cols].shape[0])]

cv = CountVectorizer()

count\_matrix = cv.fit\_transform(common\_books['combined\_features'])

cosine\_sim = cosine\_similarity(count\_matrix)

index = common\_books[common\_books['book\_title'] == book\_title]['index'].values[0]

sim\_books = list(enumerate(cosine\_sim[index]))

sorted\_sim\_books = sorted(sim\_books, key=lambda x: x[1], reverse=True)[1:2]

books = []

for i in range(len(sorted\_sim\_books)):

books.append(common\_books[common\_books['index'] == sorted\_sim\_books[i][0]]['book\_title'].item())

for i in books:

recommended\_books.append(i)

df\_new = df\_new[~df\_new['book\_title'].isin(recommended\_books)]

# CONTENT-BASED (SUMMARY)

rating\_counts = pd.DataFrame(df\_new['book\_title'].value\_counts())

rare\_books = rating\_counts[rating\_counts['book\_title'] <= 100].index

common\_books = df\_new[~df\_new['book\_title'].isin(rare\_books)]

common\_books = common\_books.drop\_duplicates(subset=['book\_title'])

common\_books.reset\_index(inplace=True)

common\_books['index'] = [i for i in range(common\_books.shape[0])]

summary\_filtered = []

for i in common\_books['Summary']:

i = re.sub("[^a-zA-Z]", " ", i).lower()

i = nltk.word\_tokenize(i)

i = [word for word in i if not word in set(stopwords.words("english"))]

i = " ".join(i)

summary\_filtered.append(i)

common\_books['Summary'] = summary\_filtered

cv = CountVectorizer()

count\_matrix = cv.fit\_transform(common\_books['Summary'])

cosine\_sim = cosine\_similarity(count\_matrix)

index = common\_books[common\_books['book\_title'] == book\_title]['index'].values[0]

sim\_books = list(enumerate(cosine\_sim[index]))

sorted\_sim\_books2 = sorted(sim\_books, key=lambda x: x[1], reverse=True)[1:4]

sorted\_sim\_books = sorted\_sim\_books2[:2]

summary\_books = []

for i in range(len(sorted\_sim\_books)):

summary\_books.append(

common\_books[common\_books['index'] == sorted\_sim\_books[i][0]]['book\_title'].item())

for i in summary\_books:

recommended\_books.append(i)

df\_new = df\_new[~df\_new['book\_title'].isin(recommended\_books)]

# TOP RATED OF CATEGORY

category = common\_books[common\_books['book\_title'] == book\_title]['Category'].values[0]

top\_rated = common\_books[common\_books['Category'] == category].groupby('book\_title').agg(

{'rating': 'mean'}).reset\_index()

if top\_rated.shape[0] == 1:

recommended\_books.append(

common\_books[common\_books['index'] == sorted\_sim\_books2[2][0]]['book\_title'].item())

else:

top\_rated.drop(top\_rated[top\_rated['book\_title'] == book\_title].index[0], inplace=True)

top\_rated = top\_rated.sort\_values('rating', ascending=False).iloc[:1]['book\_title'].values[0]

recommended\_books.append(top\_rated)

fig, axs = plt.subplots(1, 5, figsize=(18, 5))

fig.suptitle('You may also like these books', size=22)

for i in range(len(recommended\_books)):

url = df.loc[df['book\_title'] == recommended\_books[i], 'img\_l'][:1].values[0]

try:

opener = urllib.request.URLopener()

opener.addheader('User-Agent',

'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.99 Safari/537.36')

im = Image.open(opener.open(url))

axs[i].imshow(im)

axs[i].axis("off")

axs[i].set\_title('Rating: {}'.format(round(df[df['book\_title'] == recommended\_books[i]]['rating'].mean(), 1)),

y=-0.18,

color="red",

fontsize=18)

except Exception as e:

print(f"Error opening image at URL: {url}")

print(traceback.format\_exc()) # Print traceback for debugging

plt.show() # Display the plot

else:

print("Can't find book in the dataset, please check spelling")

custom\_recommender("Tuesdays with Morrie: An Old Man, a Young Man, and Life's Greatest Lesson")

**App.py**

# Define the main function

import streamlit as st

import pandas as pd

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import urllib.request

import io

from PIL import Image

import pickle

import traceback

import re

import nltk

from nltk.corpus import stopwords

col1, col2, col3 = st.columns([1,3,1])

with col1:

st.write("")

with col2:

st.image("logo2.png")

with col3:

st.write("")

st.markdown(

"""

<style>

[data-testid="stAppViewContainer"]

{

background: linear-gradient( to right, #68abc0,#c6d9e6);

}

</style>

""",

unsafe\_allow\_html=True

)

# Load your preprocessed DataFrame from a Pickle file

with open('df.pkl', 'rb') as file:

df = pickle.load(file)

with open('df2.pkl', 'rb') as file:

df2 = pickle.load(file)

# Load your pre-trained content-based recommender function from a Pickle file

def custom\_recommender(book\_title):

# ITEM-BASED

book\_title = str(book\_title)

if book\_title in df['book\_title'].values:

rating\_counts = pd.DataFrame(df['book\_title'].value\_counts())

rare\_books = rating\_counts[rating\_counts['book\_title'] <= 180].index

common\_books = df[~df['book\_title'].isin(rare\_books)]

if book\_title in rare\_books:

random = pd.Series(common\_books['book\_title'].unique()).sample(2).values

st.write('There are no recommendations for this book')

st.write('Try:\n')

st.write('{}'.format(random[0]), '\n')

st.write('{}'.format(random[1]), '\n')

else:

user\_book\_df = common\_books.pivot\_table(index=['user\_id'],

columns=['book\_title'], values='rating')

book = user\_book\_df[book\_title]

recom\_data = pd.DataFrame(user\_book\_df.corrwith(book). \

sort\_values(ascending=False)).reset\_index(drop=False)

if book\_title in [book for book in recom\_data['book\_title']]:

recom\_data = recom\_data.drop(recom\_data[recom\_data['book\_title'] == book\_title].index[0])

low\_rating = []

for i in recom\_data['book\_title']:

if df[df['book\_title'] == i]['rating'].mean() < 5:

low\_rating.append(i)

if recom\_data.shape[0] - len(low\_rating) > 5:

recom\_data = recom\_data[~recom\_data['book\_title'].isin(low\_rating)]

recom\_data = recom\_data[0:1]

recom\_data.columns = ['book\_title', 'corr']

recommended\_books = []

for i in recom\_data['book\_title']:

recommended\_books.append(i)

df\_new = df[~df['book\_title'].isin(recommended\_books)]

# CONTENT-BASED (Title, Author, Publisher, Category)

rating\_counts = pd.DataFrame(df\_new['book\_title'].value\_counts())

rare\_books = rating\_counts[rating\_counts['book\_title'] <= 100].index

common\_books = df\_new[~df\_new['book\_title'].isin(rare\_books)]

common\_books = common\_books.drop\_duplicates(subset=['book\_title'])

common\_books.reset\_index(inplace=True)

common\_books['index'] = [i for i in range(common\_books.shape[0])]

target\_cols = ['book\_title', 'book\_author', 'publisher', 'Category']

common\_books['combined\_features'] = [' '.join(common\_books[target\_cols].iloc[i,].values) for i in

range(common\_books[target\_cols].shape[0])]

cv = CountVectorizer()

count\_matrix = cv.fit\_transform(common\_books['combined\_features'])

cosine\_sim = cosine\_similarity(count\_matrix)

index = common\_books[common\_books['book\_title'] == book\_title]['index'].values[0]

sim\_books = list(enumerate(cosine\_sim[index]))

sorted\_sim\_books = sorted(sim\_books, key=lambda x: x[1], reverse=True)[1:2]

books = []

for i in range(len(sorted\_sim\_books)):

books.append(common\_books[common\_books['index'] == sorted\_sim\_books[i][0]]['book\_title'].item())

for i in books:

recommended\_books.append(i)

df\_new = df\_new[~df\_new['book\_title'].isin(recommended\_books)]

# CONTENT-BASED (SUMMARY)

rating\_counts = pd.DataFrame(df\_new['book\_title'].value\_counts())

rare\_books = rating\_counts[rating\_counts['book\_title'] <= 100].index

common\_books = df\_new[~df\_new['book\_title'].isin(rare\_books)]

common\_books = common\_books.drop\_duplicates(subset=['book\_title'])

common\_books.reset\_index(inplace=True)

common\_books['index'] = [i for i in range(common\_books.shape[0])]

summary\_filtered = []

for i in common\_books['Summary']:

i = re.sub("[^a-zA-Z]", " ", i).lower()

i = nltk.word\_tokenize(i)

i = [word for word in i if not word in set(stopwords.words("english"))]

i = " ".join(i)

summary\_filtered.append(i)

common\_books['Summary'] = summary\_filtered

cv = CountVectorizer()

count\_matrix = cv.fit\_transform(common\_books['Summary'])

cosine\_sim = cosine\_similarity(count\_matrix)

index = common\_books[common\_books['book\_title'] == book\_title]['index'].values[0]

sim\_books = list(enumerate(cosine\_sim[index]))

sorted\_sim\_books2 = sorted(sim\_books, key=lambda x: x[1], reverse=True)[1:4]

sorted\_sim\_books = sorted\_sim\_books2[:2]

summary\_books = []

for i in range(len(sorted\_sim\_books)):

summary\_books.append(

common\_books[common\_books['index'] == sorted\_sim\_books[i][0]]['book\_title'].item())

for i in summary\_books:

recommended\_books.append(i)

df\_new = df\_new[~df\_new['book\_title'].isin(recommended\_books)]

# TOP RATED OF CATEGORY

category = common\_books[common\_books['book\_title'] == book\_title]['Category'].values[0]

top\_rated = common\_books[common\_books['Category'] == category].groupby('book\_title').agg(

{'rating': 'mean'}).reset\_index()

if top\_rated.shape[0] == 1:

recommended\_books.append(

common\_books[common\_books['index'] == sorted\_sim\_books2[2][0]]['book\_title'].item())

else:

top\_rated.drop(top\_rated[top\_rated['book\_title'] == book\_title].index[0], inplace=True)

top\_rated = top\_rated.sort\_values('rating', ascending=False).iloc[:1]['book\_title'].values[0]

recommended\_books.append(top\_rated)

recommended\_books.append(book\_title) # Add the original book for reference

return recommended\_books # Return the list of recommended books

else:

st.write("Can't find book in the dataset, please check spelling")

def login():

st.markdown("<h1 style='text-align: center;'> Login to Readily</h1>", unsafe\_allow\_html=True)

username = st.text\_input("Username")

password = st.text\_input("Password", type="password")

# Check if the username and password are correct

if username == "username" and password == "password":

st.success("Login successful!")

return True

else:

if st.button("Login"):

st.error("Invalid username or password. Please try again.")

return False

# Define a function to convert rating to stars

def rating\_to\_stars(rating):

full\_stars = int(rating)

empty\_stars = 10 - full\_stars

return f"{'✦' \* full\_stars}{'✧' \* empty\_stars}"

# Define the main function

def display\_top\_rated\_books():

st.title("Our Top Rated Books ")

# Sort the DataFrame by rating in descending order and get the top 20

top\_rated\_books = df.sort\_values(by='rating', ascending=False).drop\_duplicates(subset='book\_title').head(20)

# Display the top 20 books with their titles, ratings, summaries, and covers

for i, row in top\_rated\_books.iterrows():

book\_title = row['book\_title']

rating = row['rating']

img\_l = row['img\_l']

summary = row['Summary']

# Create a layout with two columns

col1, col2 = st.columns(2)

with col1:

# Display book cover image on the left

st.image(img\_l, caption=f"Book Cover: {book\_title}")

with col2:

# Display book title, rating, and summary on the right

st.write(f"Title: {book\_title}")

st.write(f"Rating: {rating\_to\_stars(rating)}", unsafe\_allow\_html=True)

st.write(f"Summary: {summary}")

# Define the main function

def main():

if login():

st.markdown("<h1 style='text-align: center'>Readily</h1>", unsafe\_allow\_html=True)

# Dropdown for selecting the page

page = st.selectbox("Select a page:", ["Book Recommendation", "Top Rated Books"])

if page == "Book Recommendation":

# Book recommendation page

book\_title = st.selectbox("Select a book title:", df2['books'], key='book\_title\_dropdown')

if st.button("Recommend"):

st.subheader("Recommended Books")

recommended\_books\_data = custom\_recommender(book\_title)

if recommended\_books\_data:

for recommended\_book in recommended\_books\_data[:5]:

# Display recommended books

book\_info = df[df['book\_title'] == recommended\_book]

img\_l = book\_info['img\_l'].values[0]

summary = book\_info['Summary'].values[0]

rating = book\_info['rating'].values[0]

# Create a layout with two columns

col1, col2 = st.columns(2)

with col1:

# Display book cover image on the left

st.image(img\_l, caption=f"Book Cover: {recommended\_book}")

with col2:

# Display book summary in the upper right

st.write(f"Summary: {summary}")

# Display book rating in star format below the summary

st.write(f"Rating: {rating\_to\_stars(rating)}", unsafe\_allow\_html=True)

elif page == "Top Rated Books":

# Top-rated books page

display\_top\_rated\_books()

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

main()

**7.2 SCREENSHOTS:**

**Login:**

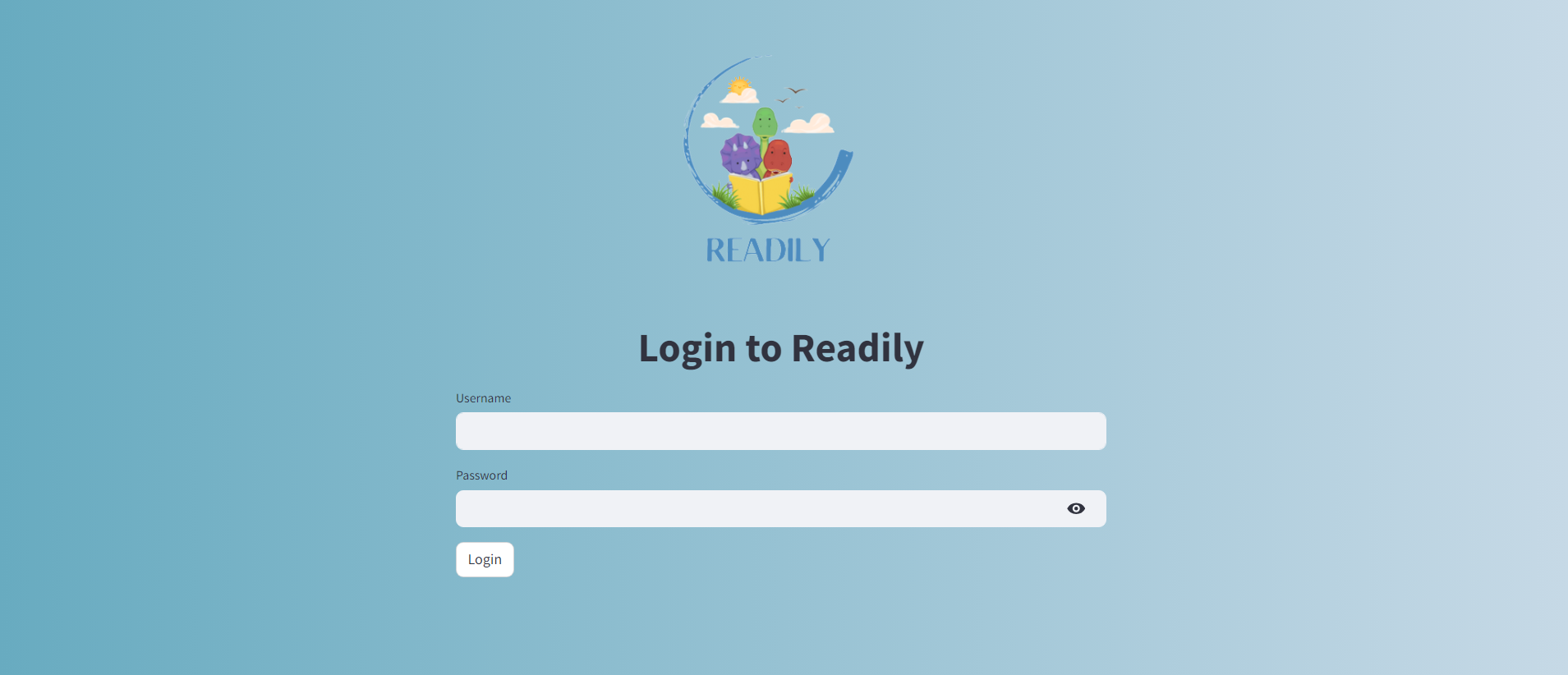


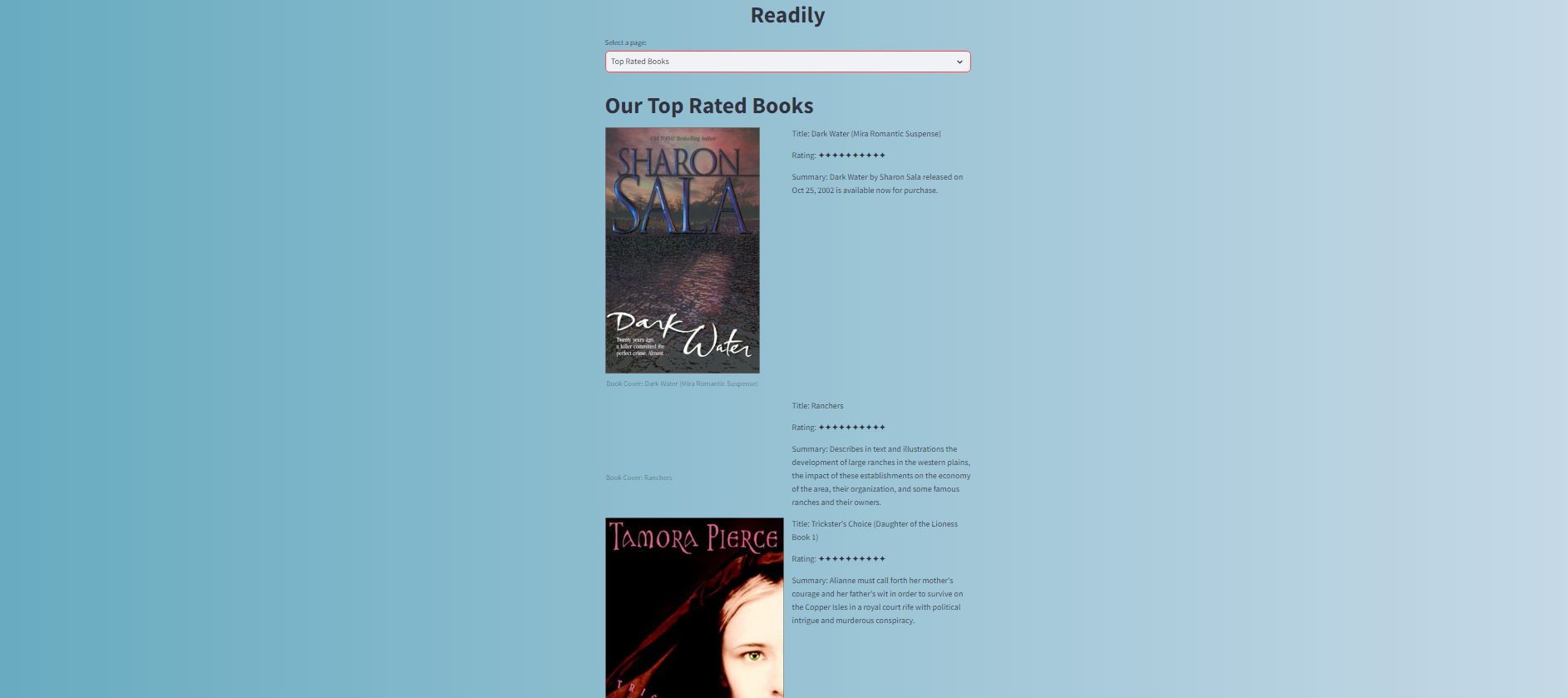
Fig 7.2 : Login

**Home**

****

Fig 7.3 : Home

# Top rated Books



# Fig 7.4 : Top Rated Books(1)

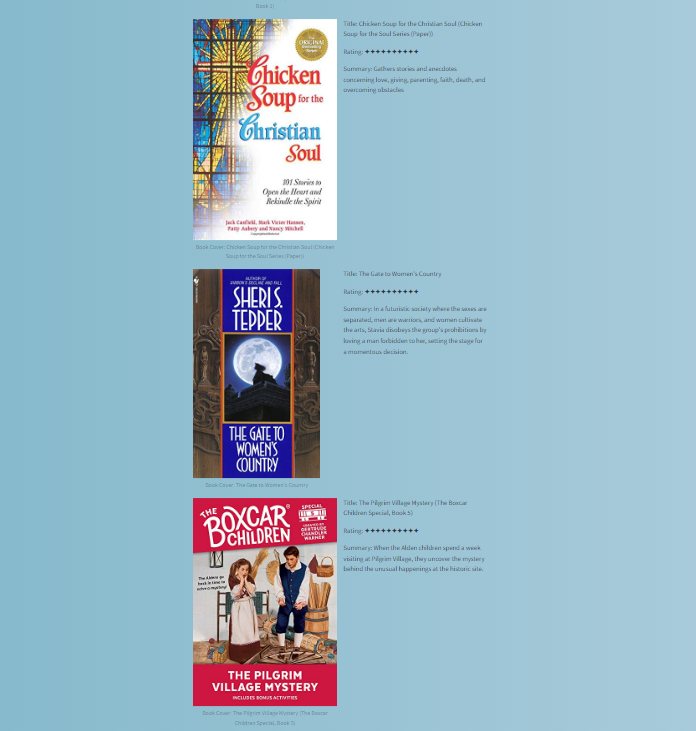


Fig 7.5 : Top Rated Books(2)

# 

Fig 7.6 : Top Rated Books(3)



Fig 7.7 : Top Rated Books(4)

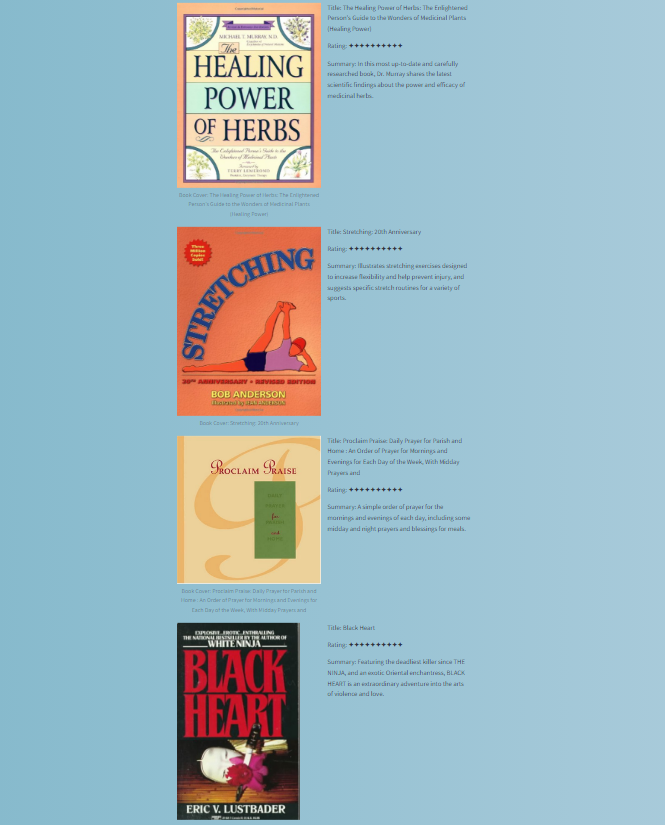
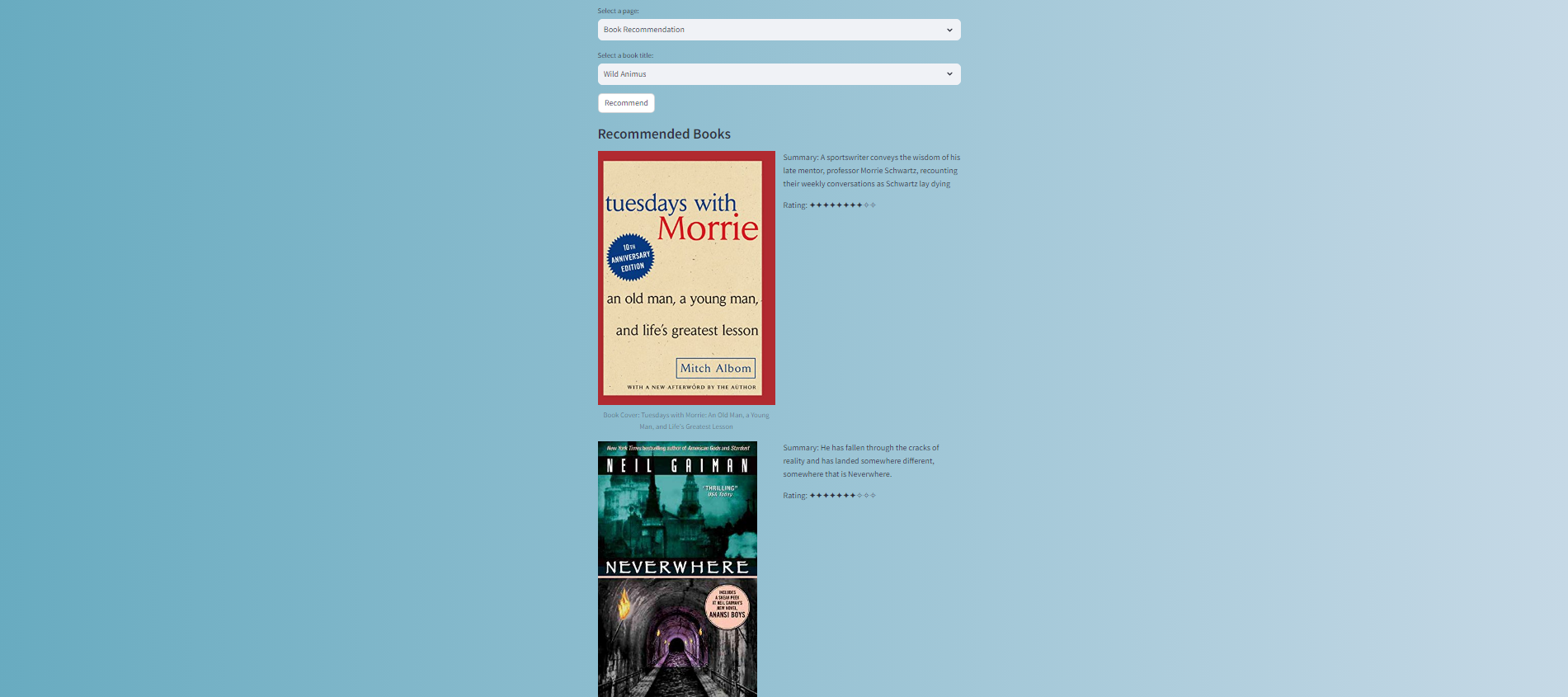


Fig 7.8 : Top Rated Books(5)



Fig 7.9 : Top Rated Books(6)

# Book Recommendation



# Fig 7.10 : Book Recommendation(1)

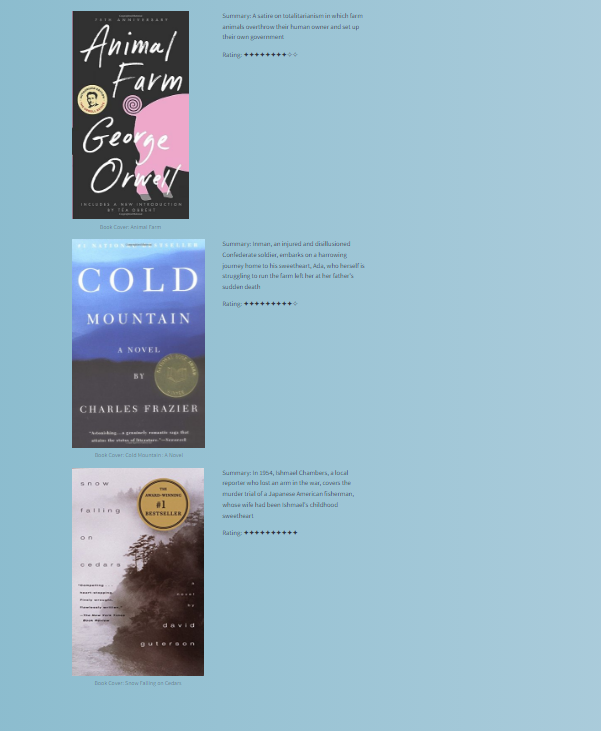


Fig 7.11 : Book Recommendation(2)

# CHAPTER-8

**BIBLOGRAPHY**

**8.1 REFERENCE:**

[1] "Python Crash Course" by Eric Matthes (Published in 2019]

[2] "CSS: The Definitive Guide" by Eric A. Meyer (Published in 2017)

[3] "Python for Data Analysis" by Wes McKinney (Published in 2017)  
[4] "Python for Everybody: Exploring Data in Python 3" by Charles Severance (Published in 2016)

[5] "Responsive Web Design with HTML5 and CSS3" by Ben Frain (Published in 2015)

[6] "CSS Secrets: Better Solutions to Everyday Web Design Problems" by Lea Verou (Published in 2015)

[7] "A Byte of Python" by C.H. Swaroop (Published in 2003)

[8] "Python Scripting for Computational Science" by Hans Petter Langtangen (Published in 2003)

[9] "Python Programming for the Absolute Beginner" by Michael Dawson(2003)

[10] "Python: The Complete Reference" by Martin C. Brown (Published in 2001)