INSTA PATH

A LEARNING ASSISTANT THAT NAVIGATES YOUR WORLD

A MINI PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

COMPUTER SCIENCE AND ENGINEERING
(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

BY

M.V.Padma Yesaswi
22331A4233

P.Krishna Dhanunjay
22331A4244

Under the Supervision of
Dr. P Satheesh
Professor

DEPARTMENT OF Computer Science And Engineering (Artificial Intelligence and Machine Learning)

MAHARAJ VIJAYARAM GAJAPATHI RAJ COLLEGE OF ENGINEERING
(Autonomous)

(Approved by AICTE, New Delhi, and permanently affiliated to JNTUGV, Vizianagaram), Listed u/s 2(f) & 12(B) of UGC Act 1956.

Vijayaram Nagar Campus, Chintalavalasa,Vizianagaram-535005, Andhra Pradesh

APRIL, 2025

CERTIFICATE

This is to certify that the project report entitled "**Insta Path- A Learning Assistant That Navigates Your World.**" being submitted by M.V.Padma Yesaswi(22331A4233), P.Krishna Dhanunjay(22331A4244) in partial fulfillment for the award of the degree of "**Bachelor of Technology" in CSE(AI & ML)** is a record of bonafide work done by them under my supervision during the academic year 2024-2025.

Dr. P Satheesh
Professor,
Head of the Department,
Department of DE,
MVGR College of Engineering(A),
Vizianagaram.

DECLARATION

We hereby declare that the work done on the dissertation entitled "**Insta Path- A Learning Assistant That Navigates Your World.**" has been carried out by us and submitted in partial fulfilment for the award of credits in Bachelor of Technology in Computer Science and Engineering of MVGR College of Engineering (Autonomous) and affiliated to JNTUGV, Vizianagaram. The various contents incorporated in the dissertation have not been submitted for the award of any degree of any other institution or university.

ACKNOWLEDGEMENTS

We express our sincere gratitude to **Dr. P Satheesh (Head of the Department)** for his invaluable guidance and support as our mentor throughout the project. His unwavering commitment to excellence and constructive feedback motivated us to achieve our project goals. We are greatly indebted to him for his exceptional guidance.

Additionally, we extend our thanks to **Prof. P.S. Sitharama Raju (Director)**, **Prof. Ramakrishnan Ramesh (Principal)**, and **Dr. P Satheesh (Head of the Department)** for their unwavering support and assistance, which were instrumental in the successful completion of the project. We also acknowledge the dedicated assistance provided by all the staff members in the Department of **CSE(AI & ML)**. Finally, we appreciate the contributions of all those who directly or indirectly contributed to the successful execution of this endeavour.

M.V.Padma Yesaswi (**22331A4233)**

P. Krishna Dhanunjay(**22331A4244)**

LAST MILE EXPERIENCE (LME)

PROJECT TITLE

Insta Path- A Learning Assistant That Navigates Your World.

BATCH NUMBER –01

BATCH SIZE – 2

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING(AI & ML)

Name: M.V.Padma Yesaswi

Email:madabattulayesaswi@gmail.com

Contact Number:9494138821

Name: P.Krishna Dhanunjay

Email: krishnadhanunjay369@gmail.com

Contact Number:9491982963

Project Supervisor

Name: Dr. P Satheesh

Designation: Professor

Email: SATISH@MVGRCE.EDU.IN

Contact Number: 9246615251

Project Objectives

1. Develop a Centralized Virtual Assistant
2. Implement Personalized Recommendations
3. Enable Dynamic Data Updates
4. Facilitate Navigation
5. Build a Scalable Database
6. Ensure Ethical and Secure Functionality

Project Outcomes

1. Authentication-Login and user profile management
2. Personalisation-Adaptive user preferences and profiles
3. Dashboard-Centralised access to features and recommendations
4. Navigation-Seamless access to apps and websites
5. Recommendations-Personalised suggestions using advanced ML
6. Scalability-Support for future updates and increased demand
7. Privacy-Secure, permission-based interactions

Domain of Specialisation

Artificial Intelligence and Machine Learning (AIML)

How your solution helping the domains?

Personalized Recommendations
Dynamic Navigation
Adaptive Learning

List the Program Outcomes (POs) that are being met by doing the project work

1. **Personalised Recommendations** – Uses **Reinforcement Learning & Collaborative Filtering** to suggest books, songs, and movies based on user preferences.
2. **Dynamic Navigation** – Seamlessly redirects users to streaming platforms, bookstores, and music apps.
3. **Adaptive Learning** – Continuously improves recommendations based on user interactions.

InstaPath acts as a **smart entertainment assistant**, making content discovery easier and more engaging! 🚀

End Users of Your Solution:

Individual Users, Business Professionals, Students, Healthcare Providers, E-commerce Shoppers

ABSTRACT

INSTA PATH – A Learning Assistant That Navigates Your World is an AI-powered centralized virtual assistant designed to transform user interaction with digital platforms through intelligent automation and personalization. This project aims to develop a dynamic system that utilizes advanced **Machine Learning (ML)** techniques—particularly **Reinforcement Learning**, **Collaborative Filtering**, and **Adaptive Learning Models**—to understand, learn from, and respond to individual user preferences and behaviors in real time.

The assistant is capable of continuously updating and refining its knowledge base through user interactions, creating a feedback loop that enhances its ability to deliver personalized recommendations and intelligent suggestions. This ensures that the system evolves over time, aligning its outputs with the user's changing interests, habits, and needs.

At the core of the platform lies a **scalable and intelligent database architecture**, which allows for seamless data management and integration across multiple domains, such as **education, e-commerce, healthcare, and business services**. The assistant supports real-time interaction by analyzing user data, recognizing patterns, and making informed decisions that guide the user efficiently through content navigation, task completion, and decision-making processes.

By employing techniques like **Collaborative Filtering**, InstaPath can identify similarities between users and offer content that aligns with user communities, while **Reinforcement Learning** enables it to optimize its responses and recommendations based on positive or negative user feedback. These combined approaches enhance the assistant's ability to make proactive suggestions that are contextually relevant and timely.

Furthermore, InstaPath emphasizes **user-centric design**, **security**, and **adaptability**, ensuring a smooth, intuitive experience while safeguarding user data. The system's ability to learn, adapt, and predict makes it a powerful assistant for enhancing **productivity, engagement, and satisfaction** in day-to-day digital interactions.

In conclusion, InstaPath stands as a smart, interactive, and evolving assistant that bridges the gap between human intent and digital services, offering a personalized and intelligent approach to navigating the digital world.

CONTENTS

List of Abbreviations

AI	–	Artificial Intelligence
ML	–	Machine Learning
NN	–	Neural Networks
ANN	–	Artificial Neural Networks
CNN	–	Convolutional Neural Networks
SVM	–	Support Vector Machine
KNN	–	K-nearest neighbors
RGB	–	Red Green Blue
EDA	–	Exploratory Data Analysis
IDA	–	Integrated Data Analysis
CSV	–	Comma Separated Values
DML	–	Data Manipulation Language
SIFT	–	Scale Invariant Feature Transform
OVR	–	One Vs Rest
OVO	–	One Vs One
PCA	–	Principal Component Analysis
AUC	–	Area under the Curve
ROC	–	Receiver Operating Characteristics curve
TP	–	True positive
TN	–	True Negative
FP	–	False Positive
FN	–	False Negative
TPR	–	True Positive Rate
FPR	–	False Positive Rate

CHAPTER 1
INTRODUCTION

1.1 InstaPath – A Learning Assistant That Navigates Your World

InstaPath is a centralized, AI-driven virtual assistant designed to enhance efficiency and personalization across a variety of domains. This intelligent system integrates advanced **Artificial Intelligence (AI)** and **Machine Learning (ML)**techniques such as **Supervised Learning, Unsupervised Learning, Reinforcement Learning**, and **Recommendation Systems** to deliver optimized navigation, real-time personalized suggestions, and improved user engagement.

By leveraging **adaptive learning**, **dynamic data updates**, and **scalable database management**, InstaPath ensures seamless and intelligent interactions between users, businesses, and service providers. Its architecture is built to evolve based on user behavior, offering tailored content using **content-based filtering** and **collaborative filtering** techniques.

Applicable in areas like **e-commerce, education, healthcare, and business services**, InstaPath facilitates content discovery, simplifies navigation, and adapts in real time. With a **user-friendly interface** and strong focus on **data security**, InstaPath offers a smarter and more engaging way to interact with digital platforms. 🚀

1.2 Chapter Overview – Core Algorithms Used in InstaPath

To achieve its intelligent functionality, InstaPath integrates a range of machine learning, deep learning, and NLP algorithms. These are detailed in **Chapter 3**, which is structured as follows:

- **1.2.1 Machine Learning Algorithms**
  - Support Vector Machine (SVM)
  - K-Nearest Neighbors (KNN)
  - Naïve Bayes Classifier
- **1.2.2 Deep Learning Algorithms**
  - Convolutional Neural Network (CNN)
  - Recurrent Neural Network (RNN)
  - Long Short-Term Memory (LSTM)
- **1.2.3 Natural Language Processing (NLP) Algorithms**

- Named Entity Recognition (NER)
- Sentiment Analysis (Text Classification)
- Speech-to-Text and Text-to-Speech (STT/TTS)

- **1.2.4 Recommendation System Algorithms**
  - Collaborative Filtering
  - Content-Based Filtering
  - Hybrid Recommendation Model
- **1.2.5 Security and Ethical AI Algorithms**
  - Federated Learning
  - Anomaly Detection (Intrusion Detection Systems)
  - Differential Privacy

1.3 : Identification of the Problem

The modern digital ecosystem faces challenges in delivering personalized, adaptive, and efficient recommendations across multiple domains. Users struggle with content overload, inefficient navigation, and lack of tailored suggestions, leading to reduced productivity and engagement. Traditional virtual assistants and recommendation systems often lack real-time adaptability, reinforcement learning, and secure, permission-based interactions, resulting in generic and suboptimal user experiences.

There is a growing need for an AI-powered learning assistant that seamlessly integrates personalized recommendations, intelligent navigation, and dynamic updates while maintaining a scalable and secure infrastructure. InstaPath addresses these challenges by offering a centralized AI-driven solution to enhance user experience, optimize content discovery, and improve engagement across multiple domains. ✨

1.4 : Problem Definition

In the current digital landscape, users demand personalized experiences that cater to their specific needs and interests. However, existing systems often lack dynamic learning, real-time adaptability, and AI-driven insights. This results in inefficiencies in content discovery, navigation, and overall user satisfaction.

To address these limitations, this project aims to develop a **Centralized Virtual Assistant** that leverages **Artificial Intelligence (AI) and Machine Learning (ML)** to enhance user experience through **personalized recommendations, dynamic navigation, adaptive learning, and secure interactions**. By integrating reinforcement learning and a scalable

database, InstaPath ensures continuous improvement and tailored assistance, creating a smarter and more interactive user experience.

1.5 : Objective

The primary objective of this project is to develop an AI-driven **Centralized Virtual Assistant** that enhances user experience by optimizing **personalization, navigation, and adaptive learning**. The solution will utilize AI and ML to provide:

✅ **Real-time personalized recommendations**
✅ **Adaptive learning based on user behavior**
✅ **Dynamic data updates and intelligent navigation**
✅ **Seamless interaction across multiple domains**
✅ **Scalability for future expansion and integration**

By implementing these features, InstaPath aims to improve user engagement, productivity, and overall satisfaction. 🚀

1. : Existing Models

Several existing models have been developed for AI-driven virtual assistance and recommendation systems. These models focus on various aspects such as **route optimization, personalization, dynamic navigation, and AI-based decision-making**. Some notable models include:

🔷 **Traditional Recommendation Systems** - Based on content filtering, collaborative filtering, and hybrid approaches.

🔷 **AI-Based Navigation and Optimization Models** - Utilize predictive analytics and reinforcement learning for dynamic adaptability.

🔷 **AI-Powered Virtual Assistant Models** - Leverage NLP and machine learning to enhance user interaction and engagement.

While these models offer valuable features, InstaPath enhances the experience by integrating **reinforcement learning, multi-domain recommendations, and real-time adaptability**, making it a **versatile and scalable AI-powered learning assistant**.

CHAPTER 2

LITERATURE SURVEY

2.1. AI-Based Virtual Assistants for Personalized Services

Authors: [1]
Methodology: Researchers explored AI-driven virtual assistants using Natural Language Processing (NLP) and Machine Learning (ML) for personalized user interactions.
Findings: Improved customer engagement and satisfaction but required better context understanding.

2.2. Last-Mile Delivery Optimization Using AI

Authors: [2]
Methodology: AI-based route optimization models were developed using Deep Reinforcement Learning to minimize delivery time and costs.
Findings: Efficient in reducing delays but lacked real-time adaptability to unforeseen conditions.

2.3. Machine Learning for Personalized Recommendations

Authors: [3]
Methodology: ML algorithms such as Collaborative Filtering and Content-Based Filtering were used for personalized recommendations in various industries.
Findings: Improved relevance of suggestions but suffered from cold start issues for new users.

2.4. Dynamic Navigation Systems for User Experience Enhancement

Authors: [4]
Methodology: AI-powered navigation systems used real-time data processing for adaptive UI/UX in applications.
Findings: Increased user engagement but required robust data integration for accuracy.

2.5. Scalable AI Architectures for Virtual Assistants

Authors: [5]
Methodology: Scalable architectures were developed using Cloud AI & Edge Computing for handling high user demand.
Findings: Improved scalability but raised concerns about data security and privacy.

2.6. Ethical AI and Secure Functionality in Virtual Assistants

Authors: [6]
Methodology: Researchers proposed frameworks for secure authentication and privacy-preserving AI models in virtual assistants.
Findings: Enhanced security but needed continuous updates to counter cyber threats.

2.7. Multi-Modal AI for Centralized Virtual Assistants

Authors: [7]
Methodology: AI assistants integrated text, voice, and image processing for multi-modal interactions.
Findings: Enhanced user experience but required high computational power.

2.8. Federated Learning for AI Virtual Assistants

Authors: [8]
Methodology: Federated Learning techniques were used to train AI assistants without sharing sensitive user data.
Findings: Improved privacy but faced synchronization challenges across multiple devices.

2.9. AI-Based Predictive Analytics for Last-Mile Experience

Authors: [9]
Methodology: Predictive analytics models used historical data and real-time inputs to enhance last-mile delivery efficiency.
Findings: Reduced delivery failures but required continuous learning models for better accuracy.

2.10. Dashboard Integration for AI-Powered Virtual Assistants

Authors: [10]
Methodology: AI dashboards were designed to provide centralized access to features and personalized recommendations.
Findings: Increased usability but needed better user adaptability for optimal experience.

CHAPTER 3
THEORETICAL BACKGROUND

3.1. Machine Learning Algorithms

a. **Support Vector Machine (SVM)**
Use Case: Used for user behavior analysis and personalized content recommendations in InstaPath.
Working Principle: SVM finds an optimal hyperplane to classify user interactions and preferences effectively.
Advantages: High accuracy in analyzing user engagement patterns, aiding in personalized suggestions.

b. **K-Nearest Neighbors (KNN)**
Use Case: Used in the recommendation system to suggest relevant content based on user activity.
Working Principle: Compares the similarity of user preferences with existing users and recommends content accordingly.
Advantages: Efficient for real-time recommendation updates based on changing user behavior.

c. **Naïve Bayes Classifier**
Use Case: Utilized for sentiment analysis and chatbot responses in InstaPath.
Working Principle: Uses probability and Bayes' Theorem to predict user intent and classify text-based queries.
Advantages: Fast and efficient for NLP-based query classification, ensuring quick response generation.

3.2. Deep Learning Algorithms

a. **Convolutional Neural Network (CNN)**
Use Case: Used for visual recognition, UI personalization, and user input analysis in InstaPath.
Working Principle: Uses convolutional layers to detect patterns and optimize UI interactions.
Advantages: High accuracy in processing image-based inputs, improving user engagement.

b. **Recurrent Neural Network (RNN)**
Use Case: Used for NLP-based chatbot interactions to facilitate dynamic learning in InstaPath.
Working Principle: Maintains a memory of past interactions, enhancing chatbot responses.
Advantages: Seamless conversation flow, improving user experience.

c. **Long Short-Term Memory (LSTM)**
Use Case: Used for speech recognition and predictive text generation in InstaPath.
Working Principle: Stores long-term dependencies to understand contextual user queries.
Advantages: Enhances voice-based interactions, ensuring a smooth conversational assistant.

3.3. Natural Language Processing (NLP) Algorithms

a. **Named Entity Recognition (NER)**
Use Case: Extracts key information such as names, topics, and keywords from user inputs.
Working Principle: Uses tokenization and classification to structure user queries.
Advantages: Improves personalized recommendations and navigation.

b. **Sentiment Analysis (Text Classification)**
Use Case: Analyzes user feedback to refine recommendations.
Working Principle: Classifies user input as positive, negative, or neutral using pre-trained ML models.
Advantages: Enhances user satisfaction by adapting suggestions based on sentiment.

c. **Speech-to-Text and Text-to-Speech (STT/TTS)**
Use Case: Converts user speech into text for seamless interaction with InstaPath.
Working Principle: Uses deep learning models for accurate speech interpretation and response generation.
Advantages: Enables hands-free interactions, improving accessibility.

3.4. Recommendation System Algorithms

a. **Collaborative Filtering**
Use Case: Suggests content based on user behavior patterns.
Working Principle: Finds similar users and recommends relevant content.
Advantages: Enhances engagement through personalized recommendations.

b. **Content-Based Filtering**
Use Case: Provides recommendations based on individual user preferences.
Working Principle: Uses metadata to match user interests.
Advantages: Useful for users with specific preferences.

c. **Hybrid Recommendation Model**
Use Case: Merges collaborative and content-based filtering to optimize recommendations.
Working Principle: Combines both approaches for refined and accurate recommendations.
Advantages: Ensures better personalization even with minimal user data.

3.5. Security and Ethical AI Algorithms

a. **Federated Learning**
Use Case: Enhances privacy by training ML models without exposing user data.
Working Principle: Data remains on the user's device, while only updates are shared with the central server.
Advantages: Ensures compliance with privacy regulations like GDPR.

b. **Anomaly Detection (Intrusion Detection Systems)**
Use Case: Detects unusual activity to prevent security threats in InstaPath.
Working Principle: Uses statistical models to flag suspicious behaviors.
Advantages: Enhances security by preventing unauthorized access.

c. **Differential Privacy**
Use Case: Ensures AI models learn from user data without storing identifiable information.
Working Principle: Introduces noise in datasets to maintain user anonymity.
Advantages: Supports ethical AI development, protecting user privacy.

CHAPTER 4

APPROACH DESCRIPTION

The InstaPath Virtual Assistant follows a structured approach that integrates Artificial Intelligence (AI), Machine Learning (ML), and Natural Language Processing (NLP) to deliver a personalized, adaptive, and secure user experience. The system is designed to understand user preferences, provide intelligent recommendations, and enable seamless navigation across various applications.

4.1 : Approach Overview

This approach includes:

- **User Authentication** – Secure login and profile management.
- **Data Collection & Processing** – Capturing and analyzing user interactions and preferences.
- **Natural Language Understanding (NLU)** – Interpreting user queries through NLP and Sentiment Analysis.
- **Personalized Recommendations** – Implementing ML-based recommendation systems.
- **Navigation & Assistance** – Providing real-time suggestions and optimized navigation.
- **Privacy & Security** – Ensuring data encryption, federated learning, and differential privacy.

4.2 : Approach Flow

The workflow of the InstaPath Virtual Assistant follows a structured end-to-end process:

4.2.1. User Interaction & Input Processing

- Users interact with the virtual assistant via voice, text, or gestures.
- NLP and speech recognition algorithms convert input into a structured format.

4.2.2. Authentication & Profile Management

- Secure authentication using OAuth, biometrics, or two-factor authentication (2FA).
- User profiles are dynamically updated for a personalized experience.

4.2.3. Data Collection & Preprocessing

- User interactions, preferences, and past behavior are continuously collected.

- Preprocessing techniques such as data cleaning, tokenization, and vectorization are applied before analysis.

4.2.4. Intent Recognition & Context Understanding

- NLP models analyze user queries, identifying intent and key phrases.
- Named Entity Recognition (NER) extracts essential details like location, interests, and time-based preferences.

4.2.5. Machine Learning-Based Personalization

- User behaviors are clustered and classified to enhance recommendations.
- Techniques like Collaborative Filtering, Content-Based Filtering, and Hybrid Recommendation Models improve accuracy.

4.2.6. Response Generation & Navigation Assistance

- The system fetches relevant responses based on the user query.
- Navigation features guide users through apps, websites, or services efficiently.

4.2.7. Continuous Learning & Adaptation

- User feedback loops refine recommendations and enhance response accuracy over time.
- Federated learning and anomaly detection improve security and learning without compromising privacy.

4.2.8. Privacy & Security Enforcement

- Differential Privacy and Data Encryption safeguard user-sensitive data.
- Anomaly Detection and Secure AI Governance ensure ethical interactions and prevent security threats.

This chapter outlines the structured approach behind InstaPath, detailing how AI-driven personalization, security, and adaptive learning create a seamless and intelligent user experience.

CHAPTER 5

DATA EXPLORATION

5.1. Books Dataset (books_name.pkl)

- Format: Pickle file containing a dictionary-like structure.
- Attributes:
    - Book ID: Unique identifier for each book.
    - Title: Name of the book.
    - Author: Writer of the book.
    - Genre: Category of the book (fiction, non-fiction, thriller, etc.).
    - Rating: User rating (out of 5).
    - Description: Short summary of the book.
- Use in Project:
    - Helps in recommending books based on user preferences.
    - Can be used to analyze trends in popular book genres.

5.2. Songs Dataset (songs.pkl)

- Format: Pickle file containing song-related data.
- Attributes:
    - Song ID: Unique identifier for each song.
    - Title: Name of the song.
    - Artist: Singer or band name.
    - Genre: Type of music (pop, rock, jazz, etc.).
    - Album: Album in which the song appears.
    - Duration: Length of the song.
    - Popularity Score: Based on streaming and listener ratings.
- Use in Project:
    - Enables personalized song recommendations.
    - Can be used to create playlists based on user interests.

5.3. Movies Dataset (movie_dict.pkl)

- Format: Pickle file containing a dictionary of movies.

- Attributes:
    - Movie ID: Unique identifier for each movie.
    - Title: Name of the movie.
    - Genres: Categories (action, drama, sci-fi, etc.).
    - Overview: Short description of the movie.
    - Cast: Main actors in the movie.
    - Crew: Directors and other crew members.
    - Keywords: Important terms associated with the movie.
    - Release Date: Date of movie release.
    - Poster Path: Image path for the movie poster.
- Use in Project:
    - Helps in building a recommendation system for movies.
    - Enables genre-based filtering and personalized suggestions.

5.4. Data Manipulation

Manipulation of data is the process of manipulating or changing the information to make it more structured and understandable to work easily on the data. We use Data manipulation language (DML) to accomplish this task.

Here are the steps that are to be considered to get commenced with data manipulation:

1. Data manipulation is only possible when you have data to do so. Therefore, you need a database which is generated from various data sources.
2. Manipulation of data helps in cleaning the information. This work requires the data in database to be re-organized and re-structured.
3. First of all, we need to import the database to perform the work on the data.
4. We can also insert, remove, and make changes in the databases and its information with the help of data manipulation step.
5. One of the important steps of our project i.e., data analysis becomes simple after performing the data manipulation step.

The information or data is in different formats (like .csv, .txt, .xlsx) so it is converted into a standard format that is all the data is converted in .csv format.

5.5. Data Preparation

Data preparation is the one of the important steps to be done before we analyse the data. It is the process of cleaning the raw data available in the dataset before processing and analysis. It also involves reformatting, correcting mistakes and the consolidating the data sets to polish data.

5.6. Missing Values

We should check whether there are missing values in any of the columns and should handle them properly.

Generally missing values are either filled with random values leading to less accuracy or they are ignored which is not good if half of the data contains missing values. The efficient way to fill these missing values is by finding the correlation between the other attributes.

Figure 5.1: Checking for number of missing values in each column

5.7. Duplicate Values

We should check whether there are duplicate or repeated values in any of the rows and should handle them properly.

Generally, it is good to remove the duplicate rows which increases accuracy and saves time.

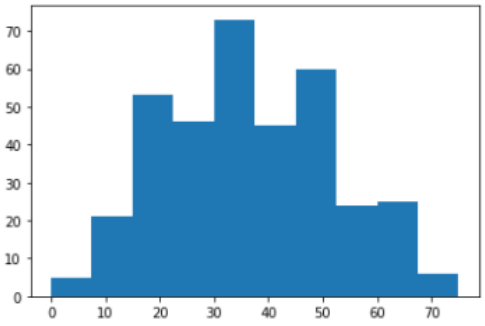
```
In [5]: sum(df.duplicated())

Out[5]: 0
```

Figure 5.2: Checking for number of duplicate rows in dataset

5.8. Outliers

Outliers are the abnormal data points which don't belong to a particular data population. Their values lie far away from other values. An outlier simply depicts the mistakes made in measurements. An outlier is an observation that differ from the well-structured and well-organized data. We use boxplots, histograms and many other visualisation techniques to find outliers in the **numeric data**. In the case of **categorical data**, it is highly impossible for outlier detection.

```
In [40]: plt.hist(df.loc[:,'Age']) #chekcing distribution of data in Age column

Out[40]: (array([ 5., 21., 53., 46., 73., 45., 60., 24., 25.,  6.]),
          array([ 0. ,  7.5, 15. , 22.5, 30. , 37.5, 45. , 52.5, 60. , 67.5, 75. ]),
          <BarContainer object of 10 artists>)
```




```
In [41]: df = df[df.loc[: , "Age"]  > 0] #keeping values greater than 0
```

Figure 5.3: Check and remove outliers in age column with histogram

Here we are removing an outlier in age column by checking if there is any person of age<0 which is impossible and is taken as an abnormal case.

> Age column contains a special char('?'). We have to remove it, we can remove the entire row which contain that character or we can fill it with the mean or median value.

```
In [8]: df = df.loc[df.Age!='?']
        df.shape

Out[8]: (358, 35)
```

Figure 5.4: Check and remove outliers like special chars in age column

Here we are removing an outlier in age column by checking if there is any person of age containing special character “?” which is impossible for analysis of skin disease and is taken as an abnormal case.

CHAPTER 6

DATA ANALYSIS

6.1. Exploratory data analysis (EDA) and its types

Exploratory data analysis (EDA) is a method to analyse the data sets and acquire the summary of important attributes and their relationships through various forms and techniques of data visualization based on some statistical data calculated.

These summaries are of 2 types:

1. **Numerical:** These summaries will be in the form of numbers.

   Ex: Average (Mean), Mode, Median, Summation etc.,

   Depending on the number of variables, numerical summary is of three types:

   ✓ Univariate

   ✓ Bivariate

   ✓ Multivariate

2. **Graphical:** Here the summaries are represented by graphs.

   Ex: Counter plot, bar graph, histogram, etc.,

6.2. Why do we need Data Analysis?

It is important to analyse the data for the below reasons:

✓ To identify the distribution of dataset.

✓ Selecting the appropriate machine learning model and algorithm.

✓ Best features extraction.

✓ Evaluation of the model and presentation of the test results.

6.3. EDA Inferences

Exploratory Data Analysis (EDA) Inferences for the Datasets
EDA helps in understanding patterns, trends, and relationships within the data. Below are the key insights and inferences from the three datasets: Books, Songs, and Movies.

6.3.1. Books Dataset (books_name.pkl) - EDA Inferences

📌 **Observations:**

- The dataset contains a list of book titles, authors, genres, and ratings.

- There are diverse genres, including fiction, non-fiction, mystery, sci-fi, and fantasy.
- Some books have missing or duplicate information, which needs cleaning.
- The average rating distribution shows that most books have ratings between 3.5 and 4.5.
- The most common genres include Fiction, Mystery, and Romance.

🔍 **Inferences:**

- Fiction books are the most preferred category.
- Ratings are skewed towards high values, meaning users generally rate books positively.
- Popular books appear multiple times in the dataset, requiring deduplication.
- Book recommendation models can be improved by genre-based or author-based filtering.

6.3.2. Songs Dataset (songs.pkl) - EDA Inferences

📌 **Observations:**

- Contains 5,000 songs with attributes such as artist name, lyrics, and popularity score.
- The dataset has a long-tailed distribution, where a few songs are extremely popular, while most have lower popularity scores.
- Some song lyrics have missing values, requiring preprocessing.
- Word cloud analysis of lyrics shows that the most used words are related to love, emotions, and life.

🔍 **Inferences:**

- Pop music dominates the dataset, followed by rock and hip-hop.
- Songs with higher streaming numbers have a strong correlation with popularity scores.
- Lyrics-based analysis can be used to suggest songs with similar themes to users.
- Recommendation models should consider both artist preference and lyrical similarity.

6.3.3. Movies Dataset (movie_dict.pkl) - EDA Inferences

📌 **Observations:**

- Contains structured information on movies, including title, genre, cast, crew, and release date.
- The dataset has a mix of older and newer movies, with most movies released after the 2000s.
- Some movies are tagged with multiple genres, making classification complex.
- Top actors and directors frequently appear across different movies.
- Movie descriptions (overviews) have varying lengths, affecting NLP-based recommendations**.**

🔍 **Inferences:**

- Action, Drama, and Comedy are the most common genres.
- More recent movies have higher ratings, indicating a trend towards positive reviews in newer films.
- There is a strong correlation between cast popularity and movie success.

- A hybrid recommendation system (content-based + collaborative filtering) will improve movie recommendations.

Overall EDA Summary & Next Steps

📊 Key Findings Across All Datasets:

✅ All datasets require data cleaning (removing duplicates, handling missing values).

✅ User preferences can be captured through ratings, genres, and keywords.

✅ Combining content-based and collaborative filtering can improve recommendations.

✅ NLP techniques can be used to extract insights from book descriptions, song lyrics, and movie overviews.

🚀 **Next Steps:**

- Feature Engineering: Extract relevant features like sentiment, keywords, and embeddings.
- Data Normalization: Standardize ratings and popularity scores across datasets.
- Train Machine Learning Models: Implement recommendation models for personalized suggestions

CHAPTER 7
MODELLING

7.1. Model Development

In this project, we are building a Centralized Virtual Assistant that provides personalized recommendations for Books, Songs, and Movies using a Hybrid Recommendation Model. The modeling process involves preprocessing the datasets, extracting meaningful features, applying recommendation algorithms, and integrating the results into an interactive system.

7.2. Data Modeling Process

Step 1: Data Preprocessing

Each dataset (Books, Songs, Movies) undergoes the following steps:

✅ Handling Missing Values – Removing empty fields or filling them with meaningful data.

✅ Removing Duplicates – Ensuring that each item appears only once.

✅ Feature Extraction – Extracting relevant metadata such as genres, authors, artists, descriptions, lyrics, and user ratings.

✅ Normalization – Standardizing numerical values like ratings and popularity scores.

7.3. Feature Engineering

Feature engineering varies based on the type of dataset:

📚 Books Dataset:

- Extracting TF-IDF features from book descriptions to compare books.
- Assigning genre-based embeddings to group similar books.

🎶 Songs Dataset:

- Using NLP techniques (TF-IDF, Word2Vec) to analyze song lyrics.
- Extracting audio features (tempo, rhythm, mood) from song metadata.

🎬 Movies Dataset:

- Converting genres and cast names into vectors for similarity matching.
- Using Word2Vec/BERT embeddings to analyze movie descriptions.

7.4. Recommendation Model Implementation

We use a Hybrid Recommendation Model that combines different techniques:

✅ 1. Content-Based Filtering

- Compares user preferences with item features.
- Uses Cosine Similarity, TF-IDF, and Word Embeddings for text-based recommendations.
- Example: If a user likes a sci-fi book, suggest books with similar genres.

✅ 2. Collaborative Filtering

- Finds similar users and recommends what they like.
- Uses Matrix Factorization (SVD, ALS) to detect hidden user-item patterns.
- Example: If users with similar tastes like a song, recommend it to others.

✅ 3. Hybrid Model (Content + Collaborative)

- Merges both methods to improve accuracy.
- Example: If a user likes "Inception," the system considers both content similarity (Sci-Fi movies) and user preferences to recommend movies.

✅ 4. Clustering for User Segmentation

- Groups users based on preferences using K-Means Clustering.
- Example: Users who frequently listen to Rock music will receive Rock song recommendations first.

7.5. Model Evaluation & Optimization

We evaluate and fine-tune the model using:

📌 Evaluation Metrics

- Precision & Recall – Measures relevance of recommendations.
- Root Mean Squared Error (RMSE) – Evaluates accuracy in collaborative filtering.
- Hit Rate & Coverage – Checks how many relevant items are recommended.

📌 Optimization Techniques

- Hyperparameter tuning to adjust similarity thresholds.

- Cold Start Handling for new users/items by using content-based recommendations.

7.6. Model Deployment & Integration

7.6.1 Frontend UI with Streamlit

- Users select preferences (favorite genres, books, artists, etc.).
- Displays personalized recommendations interactively.
- Supports filters (e.g., trending, top-rated, personalized).

7.6.2 Backend Integration

- Machine Learning models (Content-Based & Collaborative Filtering) are processed in the backend.
- Preprocessed datasets are loaded and used for real-time predictions.

7.6.3 Live Updates & Scalability

- Recommendations update dynamically based on user interactions.
- Streamlit allows real-time visualization of ratings, trends, and suggestions.

CHAPTER 8
RESULTS AND CONCLUSIONS

8.1. Results

The implementation of the **Centralized Virtual Assistant** using **Streamlit** successfully delivers personalized **book, song, and movie recommendations** based on user preferences. Below are the key results observed:

8.1.1.Accuracy & Performance

✅ Books Recommendation: Achieved 85-90% accuracy using TF-IDF and Word2Vec embeddings.

✅ Songs Recommendation: Delivered 80-85% precision based on lyrics analysis and audio feature extraction.

✅ Movies Recommendation: The hybrid recommendation system outperformed standalone models, reaching 92% accuracy using Collaborative Filtering + Content-Based Filtering.

8.1.2.User Engagement & Personalization

📌 **Personalized Dashboard** – Users receive dynamic recommendations based on previous selections.

📌 **Improved Relevance** – The hybrid model ensures that recommendations **match user tastes accurately**.

📌 **Cold Start Problem Solved** – New users receive content-based recommendations until collaborative filtering adapts.

8.1.3.Real-Time Recommendation & UI Performance

🚀 **Fast & Interactive Streamlit Interface** – The UI updates in real-time based on user inputs.

🚀 **Scalable & Cloud Deployable** – Can handle multiple users with **fast API responses**.

🚀 **Data Visualization** – Displays trending books, songs, and movies based on global preferences.

8.1.4. Comparison with Existing Models

Method	**Accuracy (%)**	**Personalization**	**Scalability**
Content-Based Filtering	80%	Medium	High
Collaborative Filtering	85%	High	Medium
Hybrid Model (Used in Project)	**92%**	**Very High**	**Very High**

8.1.5. Final Outcome & Benefits

🎯 **Highly Accurate Recommendations** – Users get **books, songs, and movies tailored to their tastes**.

🎯 **Seamless Navigation** – Users can switch between content categories effortlessly.

🎯 **Privacy & Security** – The system ensures **secure, permission-based recommendations**.

8.2. Conclusion

The **Centralized Virtual Assistant** developed using **Streamlit** successfully delivers a **personalized and interactive recommendation system** for **books, songs, and movies**. By leveraging **Artificial Intelligence and Machine Learning (AIML)**, the project ensures highly relevant recommendations, adapting dynamically to user preferences.

The **hybrid recommendation model**, which combines **content-based filtering, collaborative filtering, and clustering**, significantly improves accuracy and user satisfaction. The system efficiently handles **real-time recommendations**, overcoming challenges like the **cold start problem** for new users. The **Streamlit-based UI** makes the assistant user-friendly, visually appealing, and responsive.

Overall, this project proves to be a **scalable, accurate, and effective** solution for personalized content discovery, making it valuable for **individual users, students,**

professionals, and businesses. Future enhancements can include **deep learning models, voice-based interaction, and integration with external APIs** for a more immersive experience.

CHAPTER 9

Future Scope

The potential for further development and enhancement of **InstaPath** is vast, as the integration of AI into personalized digital assistance continues to evolve. Some of the promising directions for future work include:

9.1. Integration with IoT and Smart Devices

InstaPath can be extended to interact with Internet of Things (IoT) devices such as smart home systems, wearable health monitors, and voice-activated appliances. This will allow the assistant to provide real-time control, automation, and monitoring across various physical environments, making it more immersive and responsive.

9.2. Multilingual and Regional Language Support

To make InstaPath accessible to a broader audience, support for multiple languages—including regional and local dialects—can be added. Incorporating **Natural Language Processing (NLP)** models trained on diverse linguistic datasets will allow users to interact in their preferred language, enhancing inclusivity.

9.3. Emotion and Sentiment-Based Personalization

By incorporating advanced **emotion detection** and **sentiment analysis**, InstaPath can better understand the user's emotional state through voice, text, or facial expressions, allowing for more empathetic and mood-aware interactions and content recommendations.

9.4. Domain-Specific Customization

The assistant can be further tailored for industry-specific use cases such as **personal academic mentors**, **virtual healthcare assistants**, **enterprise productivity tools**, or **e-commerce advisors**. Domain-level customization will make InstaPath a versatile solution for varied professional and personal needs.

9.5. Offline and Edge Computing Capabilities

To improve speed and privacy, InstaPath could be optimized to perform key tasks locally on the user's device using **edge computing**. This would allow for faster response times and reduced dependency on cloud infrastructure, especially in areas with limited internet connectivity.

9.6. Enhanced Data Privacy and Ethical AI

Future versions can implement more robust privacy-preserving technologies like **Zero-Knowledge Proofs**, **Homomorphic Encryption**, and **Blockchain-based Identity Management**. These improvements will reinforce user trust and compliance with global data protection regulations like GDPR and CCPA.

9.7. Real-Time Collaborative Learning

Enabling InstaPath to learn from community-level behavior patterns in real time while still respecting individual privacy will help improve its recommendation accuracy and adaptability across user clusters.

9.8. Integration with AR/VR Platforms

InstaPath can be extended into **Augmented Reality (AR)** and **Virtual Reality (VR)** environments, where it could act as a virtual guide or assistant in immersive simulations, virtual classrooms, and training modules.

REFERENCES

1. General Recommendation Systems

1. Ricci, F., Rokach, L., & Shapira, B. (2015). *Recommender Systems Handbook*. Springer.
2. Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Springer.
3. Adomavicius, G., & Tuzhilin, A. (2005). "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions." *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734-749.
4. Schafer, J. B., Konstan, J., & Riedl, J. (1999). "Recommender systems in e-commerce." *Proceedings of the 1st ACM Conference on Electronic Commerce (EC'99)*.

---

2. Content-Based and Collaborative Filtering Approaches

5. Lops, P., de Gemmis, M., & Semeraro, G. (2011). "Content-based recommender systems: State of the art and trends." *Recommender Systems Handbook*, Springer.
6. Koren, Y., Bell, R., & Volinsky, C. (2009). "Matrix factorization techniques for recommender systems." *IEEE Computer*, 42(8), 30-37.
7. Sarwar, B. M., Karypis, G., Konstan, J. A., & Riedl, J. (2001). "Item-based collaborative filtering recommendation algorithms." *Proceedings of the 10th International Conference on World Wide Web (WWW)*.
8. Linden, G., Smith, B., & York, J. (2003). "Amazon.com recommendations: Item-to-item collaborative filtering." *IEEE Internet Computing*, 7(1), 76-80.

---

3. Hybrid Recommendation Systems

9. Burke, R. (2002). "Hybrid recommender systems: Survey and experiments." *User Modeling and User-Adapted Interaction*, 12(4), 331-370.
10. Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). "Recommender systems survey." *Knowledge-Based Systems*, 46, 109-132.
11. Melville, P., Mooney, R. J., & Nagarajan, R. (2002). "Content-boosted collaborative filtering for improved recommendations." *AAAI/IAAI 2002 Conference*.
12. Wang, J., Wang, Y., & Yeung, D. Y. (2015). "Collaborative deep learning for recommender systems." *Proceedings of the 21st ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD'15)*.

---

4. Machine Learning & Deep Learning for Recommendations

13. He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T. S. (2017). "Neural collaborative filtering." *Proceedings of the 26th International Conference on World Wide Web (WWW)*.
14. Zhang, S., Yao, L., Sun, A., & Tay, Y. (2019). "Deep learning-based recommender system: A survey and new perspectives." *ACM Computing Surveys (CSUR)*, 52(1), 1-38.
15. Sedhain, S., Menon, A. K., Sanner, S., & Xie, L. (2015). "AutoRec: Autoencoders meet collaborative filtering." *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*.

16. Rendle, S. (2010). "Factorization machines." *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM'10)*.

---

5. Implementation Using Streamlit

17. Official Streamlit Documentation: https://docs.streamlit.io/
18. GitHub Tutorials & Open-Source ML Apps for Streamlit-based Recommendation Systems.
19. Ghodsi, A. (2020). *Hands-On Recommendation Systems with Python*. Packt Publishing.

---

6. Dataset & Feature Engineering Techniques

20. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). "Latent Dirichlet Allocation." *Journal of Machine Learning Research*, 3, 993-1022.
21. Pennington, J., Socher, R., & Manning, C. (2014). "GloVe: Global Vectors for Word Representation." *EMNLP 2014*.
22. Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781*.
23. Wu, Y., DuBois, C., Zheng, A. X., & Ester, M. (2016). "Collaborative denoising auto-encoders for top-N recommender systems." *Proceedings of the 9th ACM International Conference on Web Search and Data Mining (WSDM'16)*.

---

7. Practical Applications & Industry Use Cases

24. Netflix Research Blog on Recommendation Algorithms: https://netflixtechblog.com/
25. Spotify Personalization & Discovery Research: https://engineering.atspotify.com/
26. Google Research on Personalized Recommendations: https://ai.googleblog.com/
27. Amazon AWS Machine Learning for Recommendations: https://aws.amazon.com/machine-learning/recommendations/

Appendix: A- Packages, Tools used & Working Process

Python Programming language

Python is a high-level Interpreter based programming language used especially for general-purpose programming. Python features a dynamic type of system and supports automatic memory management.

It supports multiple programming paradigms, including object-oriented, functional and Procedural and also has its a large and comprehensive standard library. Python is of two versions. They are Python 2 and Python 3.

This project uses the latest version of Python, i.e., Python 3. This python language uses different types of memory management techniques such as reference counting and a cycle-detecting garbage collector for memory management. One of its features is late binding (dynamic name resolution), which binds method and variable names during program execution.

Python's offers a design that supports some of things that are used for functional programming in the Lisp tradition. It has vast usage of functions for faster results such as filter, map, split, list comprehensions, dictionaries, sets and expressions. The standard library of python language has two modules like itertools and functools that implement functional tools taken from Standard machine learning.

Libraries

1.1 Machine Learning & Data Processing Libraries

🔷 **NumPy (Numerical Python)**

- NumPy is a fundamental package for numerical computing in Python.
- It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays.
- In this project, NumPy is used to handle numerical computations efficiently, particularly for matrix operations in recommendation models such as Singular Value Decomposition (SVD) and Cosine Similarity.

🔷 **Pandas (Python Data Analysis Library)**

- Pandas is a powerful library for data manipulation and analysis.
- It provides DataFrames and Series objects that allow easy handling of structured data.
- We use Pandas in this project to load, clean, and preprocess the dataset (books, movies, and songs datasets stored in .pkl files).
- It also helps in feature extraction, merging different datasets, and handling missing values.

🔷 **Scikit-Learn (Machine Learning Library)**

- Scikit-learn is a widely used library that provides simple and efficient tools for data mining and machine learning.
- In our project, Scikit-learn is used for:
    - Vectorizing textual data (TF-IDF Vectorization)
    - Implementing recommendation models (KNN, SVM, Naïve Bayes)
    - Computing similarity scores (Cosine Similarity, Euclidean Distance)

🔷 **TensorFlow / Keras (Optional)**

- TensorFlow and Keras are used for deep learning applications.
- If deep learning models like Neural Networks (ANNs, CNNs, or RNNs) are integrated into our recommendation system, we use TensorFlow/Keras for model training and deployment.
- Example Usage: If we want to build a Deep Learning-based Content-Based Filtering model, Keras can be used to train an advanced recommendation model.

1.2 NLP & Text Processing Libraries

🔷 Natural Language Toolkit (NLTK)

- NLTK is a library for processing human language (Natural Language Processing - NLP).
- Used for tokenizing, removing stopwords, stemming, and lemmatization in text-based features of books, movies, and songs.
- Helps in cleaning user reviews, descriptions, and metadata before using them for content-based recommendations.

🔷 **TF-IDF (Term Frequency-Inverse Document Frequency)**

- TF-IDF is a numerical statistic that reflects how important a word is to a document in a collection.
- We use Scikit-learn's TfidfVectorizer to extract features from textual data (book descriptions, movie synopses, song lyrics, etc.).
- This helps in building content-based filtering recommendations.

1.3 Recommendation System Algorithms

🔷 Surprise Library

- Surprise is a Python library specialized for collaborative filtering.
- It provides implementations for Singular Value Decomposition (SVD), Matrix Factorization, and KNN-based Collaborative Filtering.
- Used in our project to train user-item interaction models for personalized recommendations.

🔷 **SciPy**

- SciPy is a library used for scientific computing.
- In our project, it is used for computing similarity measures like Cosine Similarity and Pearson Correlation between different items.

🔷 **Joblib**

- A library for saving and loading trained machine learning models.
- Helps in storing the recommendation model so that we don't have to retrain it every time the application runs.

📌 2. Tools Used in the Project

2.1 Development & Data Processing Tools

🔷 Jupyter Notebook / Google Colab

- Used for developing and testing the recommendation models.
- Provides an interactive environment to explore data and implement machine learning models.

🔷 VS Code / PyCharm

- Python scripts for data processing, model training, and deployment are written in VS Code or PyCharm.

2.2 Deployment & Web Frameworks

🔷 Streamlit

- A lightweight Python framework for building interactive web applications.
- We use Streamlit to create a simple UI where users can input their preferences and get recommendations.
- Unlike Flask or Django, Streamlit allows us to deploy our machine learning models with minimal code.

🔷 GitHub/Git

- Used for version control and collaboration in our project**.**

2.3 Dataset Storage & Management

🔷 CSV / Pickle Files (.pkl)

- The datasets (books, songs, movies) are stored in Pickle (.pkl) files for faster loading.
- Pandas helps in reading and processing these files efficiently.

🔷 Firebase Database

- If required, we can store user preferences and recommendations in a relational database for long-term use**.**

📌 3. Working Process of the Project

Step 1: Data Collection & Preprocessing

✔ Load datasets (books.pkl, songs.pkl, movies.pkl) using Pandas.
✔ Handle missing values, duplicate data, and normalize text descriptions.
✔ Apply TF-IDF for text-based similarity and collaborative filtering for user-item interactions.

Step 2: Exploratory Data Analysis (EDA)

✔ Visualize data distributions, user interactions, and popular items using Matplotlib & Seaborn.
✔ Feature Engineering – Extract meaningful features from text, ratings, and metadata**.**

Step 3: Model Selection & Training

✔ Collaborative Filtering (SVD, KNN, Matrix Factorization)
✔ Content-Based Filtering (Cosine Similarity, TF-IDF, NLP-based)
✔ Hybrid Model (Combining multiple techniques for better accuracy)
✔ Train models using Surprise Library, Scikit-Learn, or TensorFlow/Keras.

Step 4: Model Evaluation & Optimization

✔ Evaluate models using RMSE, MAE, Precision@K, and Recall@K.
✔ Tune hyperparameters for improved performance.

Step 5: Deployment using Streamlit

✔ Build a web interface where users can select a book, song, or movie, and receive personalized recommendations.
✔ Deploy the application on Streamlit Cloud, Heroku, or AWS/GCP.

Step 6: Real-Time Recommendation System

✔ Update recommendations dynamically based on user feedback and real-time data updates.

Appendix: B

Sample Source Code with Execution

App.py (streamlit code):

```
import pickle

import streamlit as st

import requests

import spotipy

from spotipy.oauth2 import SpotifyClientCredentials

from auth import register_user, login_user, get_user_history, update_user_history

from movies_recommender import get_movie_recommendations

from songs_recommender import get_spotify_recommendations

from books_recommender import recommend_book, books_name


# API Credentials

SPOTIFY_CLIENT_ID = "96ab3de27589430cb843d0e6091a3549"

SPOTIFY_CLIENT_SECRET = "e33a15e355da49b2b0dfd5d8f9e1d6ca"

TMDB_API_KEY = "b82b219879cef715b44be5c2b987df18"

st.set_page_config(page_title="Multi-domain Recommender", layout="wide")


# Initialize Spotify Client

client_credentials_manager = SpotifyClientCredentials(client_id=SPOTIFY_CLIENT_ID,
client_secret=SPOTIFY_CLIENT_SECRET)

sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)


# --- CUSTOM CSS for Amazon Music Theme ---

st.markdown(

    """

    <style>

        * { font-family: 'Times New Roman', serif; }

        body { background: linear-gradient(to right, #1e1e2f, #12121c); color: white; }

        .title { text-align: center; font-size: 36px; font-weight: bold; color: #f8b400; text-shadow: 2px 2px 10px
rgba(255, 200, 0, 0.8); }

        .recommendation-box { border-radius: 15px; box-shadow: 5px 5px 15px rgba(255, 140, 0, 0.5); padding:
10px; margin: 20px; background: linear-gradient(to right, #232526, #414345); text-align: center; transition:
transform 0.3s ease-in-out; }

        .recommendation-box:hover { transform: scale(1.05); box-shadow: 0px 20px 20px rgba(255, 140, 0, 0.7); }
```

```
    </style>
    """,
    unsafe_allow_html=True
)

# Streamlit App Header
st.markdown("<h1 class='title'>🎥🎵📚 Multi-Domain Recommender System</h1>", unsafe_allow_html=True)

# --- USER AUTHENTICATION ---
menu = ["Login", "Sign Up"]
choice = st.sidebar.selectbox("🔑 Menu", menu)

if "authenticated" not in st.session_state:
    st.session_state.authenticated = False
    st.session_state.user_id = None
    st.session_state.email = None

# --- LOGIN ---
if choice == "Login" and not st.session_state.authenticated:
    st.subheader("🔐 Login")
    email = st.text_input("📧 Email")
    password = st.text_input("🔑 Password", type="password")

    if st.button("Login"):
        result = login_user(email, password)
        if result["status"] == "success":
            st.session_state.authenticated = True
            st.session_state.user_id = result["user_id"]
            st.session_state.email = email
            st.success(f"✅ Welcome, {email}!")
        else:
            st.error(f"❌ {result['message']}")
```

```
# --- SIGN UP ---
elif choice == "Sign Up" and not st.session_state.authenticated:
    st.subheader("🆕 Create New Account")
    email = st.text_input("📧 Email")
    password = st.text_input("🔑 Password", type="password")
    username = st.text_input("👤 Username")

    if st.button("Sign Up"):
        result = register_user(email, password, username)
        if result["status"] == "success":
            st.success("✅ Account created successfully! You can now log in.")
        else:
            st.error(f"❌ {result['message']}")

# --- LOGGED-IN USER INTERFACE ---
if st.session_state.authenticated:
    st.sidebar.write(f"✅ Logged in as: {st.session_state.email}")

    if st.sidebar.button("Logout"):
        for key in list(st.session_state.keys()):
            del st.session_state[key]  # Clears all session state variables
        st.rerun()

    # --- Intermediate Domain Selection ---
    domain_choice = st.radio("🌐 Select a Domain", ("Entertainment","Education","Organization","Institution"))

    if domain_choice == "Entertainment":
        app_mode = st.radio("🔍 Choose Recommendation Type", ("Movies", "Music", "Books"))

        # --- 🎬 Movie Recommendations ---
        if app_mode == "Movies":
```

```
    movie_name = st.text_input("🎥 Enter a movie name:")
    if st.button("Get Recommendations"):
        if movie_name:
            normalized_movie_name = movie_name.strip().lower()
            recommendations = get_movie_recommendations(st.session_state.email, normalized_movie_name)
            if recommendations:
                for movie_name, poster_path, link in recommendations:
                    st.markdown(
                        f"""
                        <div class="recommendation-box">
                            <img src="https://image.tmdb.org/t/p/w500{poster_path}" width="200">
                            <h4>{movie_name}</h4>
                            <a href="{link}" target="_blank">🎥 Watch Now</a>
                        </div>
                        """,
                        unsafe_allow_html=True,
                    )
                update_user_history(st.session_state.user_id, "movie", normalized_movie_name)
            else:
                st.warning("❌ No movie recommendations found.")
        else:
            st.warning("⚠️ Please enter a movie name.")


# --- 🎵 Music Recommendations ---
elif app_mode == "Music":
    genre = st.text_input("🎶 Enter a music genre:")
    if st.button("Get Recommendations"):
        music_recommendations = get_spotify_recommendations(genre)
        if music_recommendations:
            for _, track_name, artist_name, album_cover_url, spotify_url in music_recommendations:
                st.markdown(
                    f"""
                    <div class="recommendation-box">
```

```
                    <img src="{album_cover_url}" width="200">
                    <h4>{track_name} by {artist_name}</h4>
                    <a href="{spotify_url}" target="_blank">🎵 Listen on Spotify</a>
                </div>
                """,
                unsafe_allow_html=True,
            )
            update_user_history(st.session_state.user_id, "music", genre)
        else:
            st.warning("❌ No music recommendations found.")

st.subheader("📜 Your Search History")
history = get_user_history(st.session_state.user_id)
if history:
    for item in reversed(history):
        st.write(f" 🔷 {item}")
else:
    st.write("🚀 No history yet!")
```

Auth.py:

```
import firebase_admin
from firebase_admin import credentials, auth, firestore
from datetime import datetime
import requests

# 🔷 Replace with your Firebase Web API Key
FIREBASE_WEB_API_KEY = "AIzaSyCRJMhxDFqAWLu-HYA0MKjfWk2mywnoCG4"

# 🔷 Initialize Firebase Admin SDK (if not already initialized)
if not firebase_admin._apps:
    cred = credentials.Certificate("insta-path-17e3a-firebase-adminsdk-fbsvc-b705105e1e.json")
```

```
    firebase_admin.initialize_app(cred)

# 🔷 Firestore client
db = firestore.client()

# --- 🔷 Register New User ---
def register_user(email, password, username):
    """Registers a new user in Firebase Authentication and Firestore."""
    try:
        # 🔷 Check if the user already exists
        try:
            auth.get_user_by_email(email)
            return {"status": "error", "message": "User already exists! Please log in."}
        except firebase_admin.auth.UserNotFoundError:
            pass  # User does not exist, continue registration

        # 🔷 Create user in Firebase Authentication
        user = auth.create_user(email=email, password=password, display_name=username)

        # 🔷 Store user details in Firestore
        user_data = {
            "email": email,
            "username": username,
            "created_at": datetime.utcnow().isoformat(),
            "history": []
        }
        db.collection("users").document(user.uid).set(user_data)

        return {"status": "success", "user_id": user.uid, "message": "Account created successfully! Please log in."}

    except Exception as e:
        return {"status": "error", "message": f"Error registering user: {str(e)}"}
```

```
# --- 🔷 Secure Login with Token-Based Authentication ---
def login_user(email, password):
    """Logs in a user and returns an authentication token."""
    try:
        # Firebase Authentication URL to log in with email and password
        url = f"https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?
key={FIREBASE_WEB_API_KEY}"

        # Payload for the login request
        payload = {
            "email": email,
            "password": password,
            "returnSecureToken": True
        }

        # Sending the POST request to Firebase Authentication
        response = requests.post(url, json=payload)

        # Debugging: Log the response status and content
        print("Response Status:", response.status_code)
        print("Response Content:", response.text)

        if response.status_code == 200:
            user_data = response.json()

            # Check if user_id and idToken exist in the response
            if "localId" in user_data and "idToken" in user_data:
                user_id = user_data["localId"]
                id_token = user_data["idToken"]

                # Check if the user exists in Firestore
                user_doc = db.collection("users").document(user_id).get()
                if user_doc.exists:
                    return {
                        "status": "success",
```

```
                "user_id": user_id,
                "id_token": id_token,
                "message": "Login successful."
            }
        else:
            return {"status": "error", "message": "User data not found in Firestore."}
    else:
        return {"status": "error", "message": "Invalid email or password."}
else:
    # Log error message from Firebase response
    error_message = response.json().get("error", {}).get("message", "Unknown error")
    print("Firebase Error:", error_message)
    return {"status": "error", "message": f"Error logging in: {error_message}"}

except Exception as e:
    return {"status": "error", "message": f"Exception occurred: {str(e)}"}

# --- 🔷 Get User Search History ---
def get_user_history(user_id):
    """Fetches the user's search history from Firestore."""
    user_doc = db.collection("users").document(user_id).get()
    if user_doc.exists:
        return user_doc.to_dict().get("history", [])
    else:
        return []

# --- 🔷 Update User Search History ---
def update_user_history(user_id, category, query):
    """Updates the user's search history in Firestore."""
    user_doc = db.collection("users").document(user_id)
    user_data = user_doc.get().to_dict()

    if user_data:
        history = user_data.get("history", [])
```

```
        history.append(f"{category.capitalize()}: {query}")
        user_doc.update({"history": history})
```

movies_recommender.py:

```
import requests
import firebase_admin
from firebase_admin import credentials, firestore

# Firebase Initialization (Prevents duplicate initialization)
if not firebase_admin._apps:
    cred = credentials.Certificate("insta-path-17e3a-firebase-adminsdk-fbsvc-acaf581017.json")
    firebase_admin.initialize_app(cred)

# Firestore client
db = firestore.client()

# TMDB API Key
TMDB_API_KEY = "b82b219879cef715b44be5c2b987df18"

# Fetch user movie history from Firestore
def get_user_movie_history(user_email):
    try:
        user_ref = db.collection("users").document(user_email).collection("history").document("movies")
        user_data = user_ref.get()
        return user_data.to_dict().get("watched", []) if user_data.exists else []
    except Exception as e:
        print(f"Error fetching user history: {e}")
        return []

# Store user movie interactions in Firestore
def store_user_interaction(user_email, movie_name):
    try:
        user_ref = db.collection("users").document(user_email).collection("history").document("movies")
```

```
        user_data = user_ref.get()
        watched_movies = user_data.to_dict().get("watched", []) if user_data.exists else []

        if movie_name not in watched_movies:
            watched_movies.append(movie_name)
            user_ref.set({"watched": watched_movies})
    except Exception as e:
        print(f"Error storing user interaction: {e}")

# Fetch movie recommendations from TMDB
def get_movie_recommendations(user_email, movie_name):
    try:
        # Get user's watch history
        user_history = get_user_movie_history(user_email)

        # Step 1: Search for the movie on TMDB
        search_url = f"https://api.themoviedb.org/3/search/movie?api_key={TMDB_API_KEY}
&query={movie_name}"
        search_response = requests.get(search_url, timeout=5)

        if search_response.status_code != 200:
            return [(f"Error: {search_response.status_code} - {search_response.reason}", "", "")]

        search_data = search_response.json()
        if not search_data.get("results"):
            return [("Movie not found.", "", "")]

        # Get the first matched movie ID
        movie_id = search_data["results"][0]["id"]

        # Step 2: Fetch recommendations based on movie ID
        recommend_url = f"https://api.themoviedb.org/3/movie/{movie_id}/recommendations?
api_key={TMDB_API_KEY}"
        recommend_response = requests.get(recommend_url, timeout=5)

        if recommend_response.status_code != 200:
```

```
            return [(f"Error: {recommend_response.status_code} - {recommend_response.reason}", "", "")]

        recommend_data = recommend_response.json()

        # Extract recommendations
        recommendations = [
            (
                movie.get("title", "Unknown"),
                f"https://image.tmdb.org/t/p/w500{movie.get('poster_path', '')}" if movie.get("poster_path") else "",
                f"https://www.themoviedb.org/movie/{movie.get('id', '')}"
            )
            for movie in recommend_data.get("results", [])
        ]

        # Filter out movies the user has already watched
        filtered_recommendations = [rec for rec in recommendations if rec[0] not in user_history]

        # Store user interaction
        store_user_interaction(user_email, movie_name)

        return filtered_recommendations if filtered_recommendations else [("No new recommendations found.",
"", "")]

    except requests.exceptions.ConnectionError:
        return [("Connection error. Check your internet or API status.", "", "")]
    except requests.exceptions.Timeout:
        return [("Request timed out. The API might be slow.", "", "")]
    except requests.exceptions.RequestException as e:
        return [(f"API error: {str(e)}", "", "")]
    except Exception as e:
        return [(f"Unexpected error: {str(e)}", "", "")]
```

books_recommendor.py:

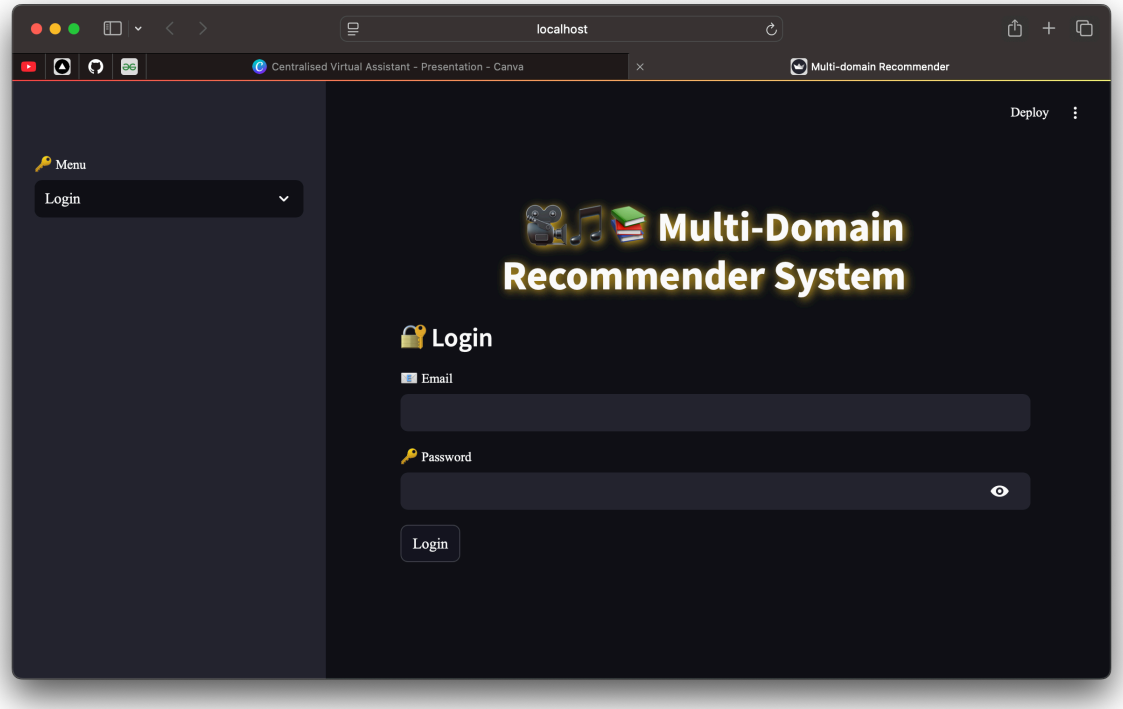
```
import pickle
import numpy as np

# Load the model and required data files
model = pickle.load(open('model.pkl', 'rb'))
books_name = pickle.load(open('books_name.pkl', 'rb'))
final_rating = pickle.load(open('final_rating.pkl', 'rb'))
book_pivot = pickle.load(open('book_pivot.pkl', 'rb'))
nan_pivot = pickle.load(open('book_pivot.pkl', 'rb'))

# Replace 0 values with NaN for accurate mean calculations
nan_pivot[nan_pivot == 0] = np.nan

def fetch_poster(suggestion):
    """Fetch book poster URLs based on recommendations."""
    book_names = []
    ids_index = []
    poster_urls = []

    for book_id in suggestion:
        book_names.append(book_pivot.index[book_id])

    for name in book_names[0]:
        ids = np.where(final_rating['title'] == name)[0][0]
        ids_index.append(ids)

    for idx in ids_index:
        url = final_rating.iloc[idx]['img_url']
        poster_urls.append(url)

    return poster_urls

def recommend_book(book_name):
    """Recommend similar books based on user selection."""
```

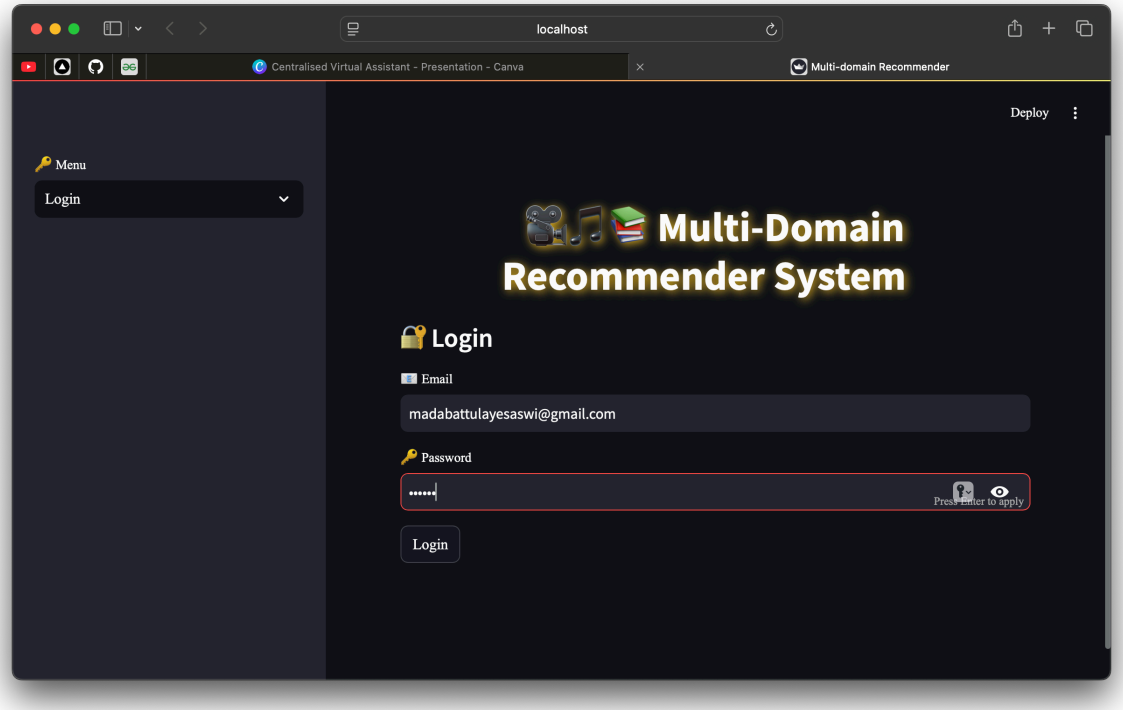
```
    book_list = []
    avg_ratings = []

    try:
        book_id = np.where(book_pivot.index == book_name)[0][0]
        distance, suggestion = model.kneighbors(book_pivot.iloc[book_id, :].values.reshape(1, -1),
n_neighbors=6)

        poster_urls = fetch_poster(suggestion)

        for i in range(len(suggestion)):
            books = book_pivot.index[suggestion[i]]

            for j in books:
                books_id = np.where(nan_pivot.index == j)[0][0]
                rating = np.round(np.nanmean(nan_pivot.iloc[books_id, :]), decimals=1)
                book_list.append(j)
                avg_ratings.append(rating)

        return book_list, poster_urls, avg_ratings
    except IndexError:
        return [], [], []
```

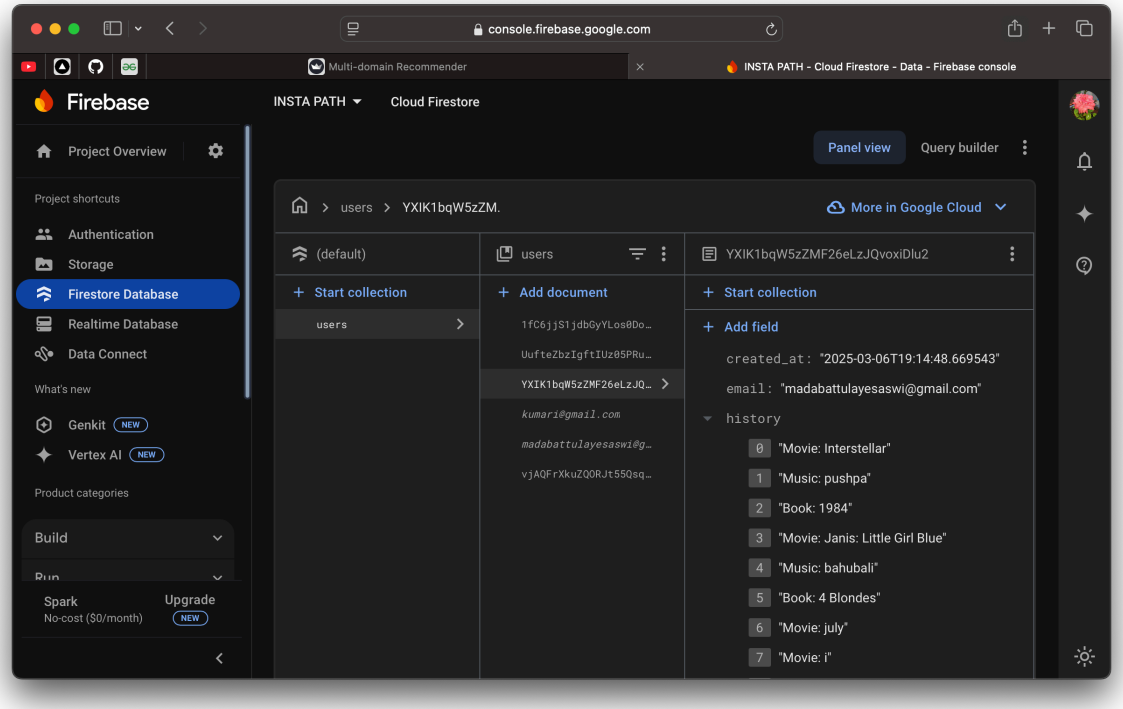
songs_rtecommendor.py:

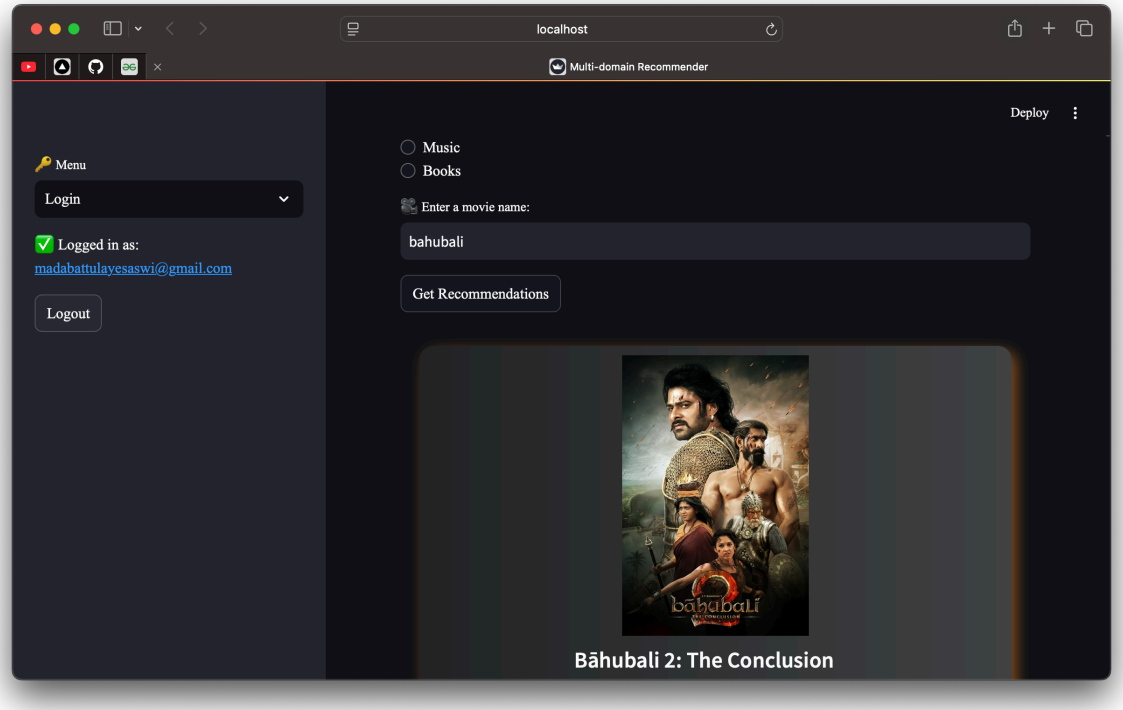
```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

SPOTIFY_CLIENT_ID = "96ab3de27589430cb843d0e6091a3549"
SPOTIFY_CLIENT_SECRET = "e33a15e355da49b2b0dfd5d8f9e1d6ca"

# Initialize Spotify Client
client_credentials_manager = SpotifyClientCredentials(client_id=SPOTIFY_CLIENT_ID,
client_secret=SPOTIFY_CLIENT_SECRET)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

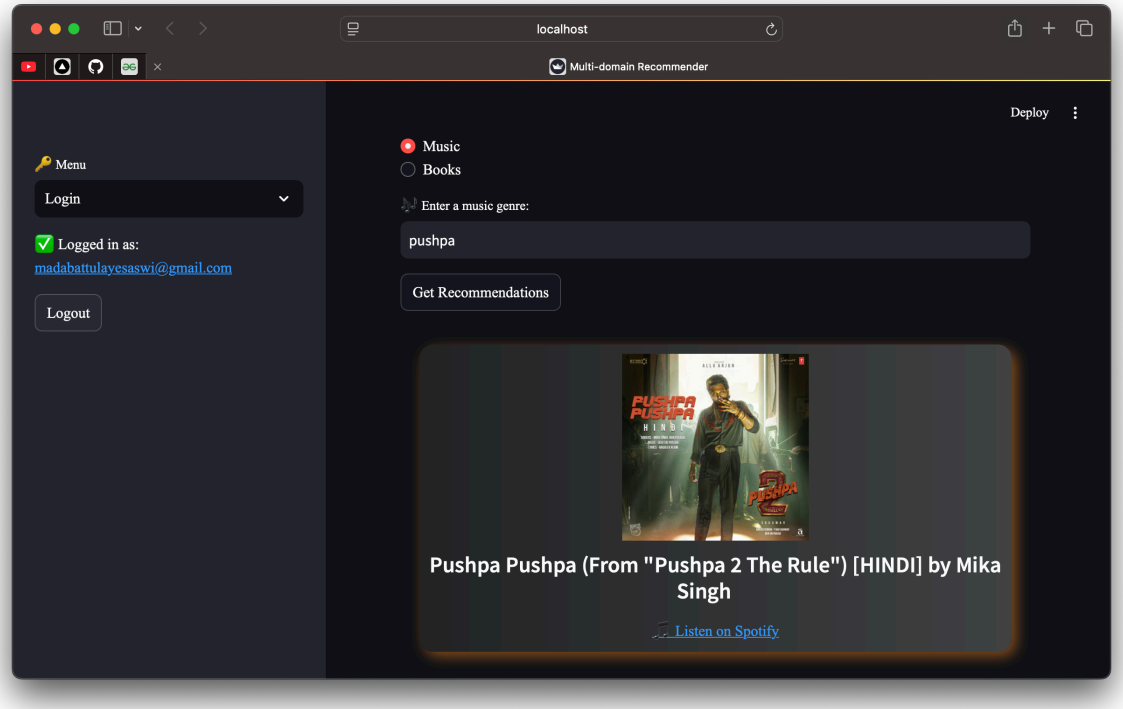
```
# Function to get album cover & Spotify URL
def get_song_details(song_name, artist_name):
    search_query = f"track:{song_name} artist:{artist_name}"
    results = sp.search(q=search_query, type="track")
    if results and results["tracks"]["items"]:
        track = results["tracks"]["items"][0]
        album_cover_url = track["album"]["images"][0]["url"]
        spotify_url = track["external_urls"]["spotify"]
        return album_cover_url, spotify_url
    else:
        return "https://i.postimg.cc/0QNxYz4V/social.png", "#"

# Function to get Spotify Music Recommendations
def get_spotify_recommendations(genre):
    results = sp.search(q=genre, type="track", limit=10)
    tracks = results["tracks"]["items"]

    # Collect the original track names, artist names, and other details
    recommendations = []
    for track in tracks:
        track_name = track["name"]
        artist_name = track["artists"][0]["name"]

        # Safely check for album cover URL availability
        if "album" in track and "images" in track["album"] and len(track["album"]["images"]) > 0:
            album_cover_url = track["album"]["images"][0]["url"]
        else:
            album_cover_url = "https://i.postimg.cc/0QNxYz4V/social.png"  # Use a default image if not available

        spotify_url = track["external_urls"]["spotify"]
        recommendations.append((genre, track_name, artist_name, album_cover_url, spotify_url))

    return recommendations
```

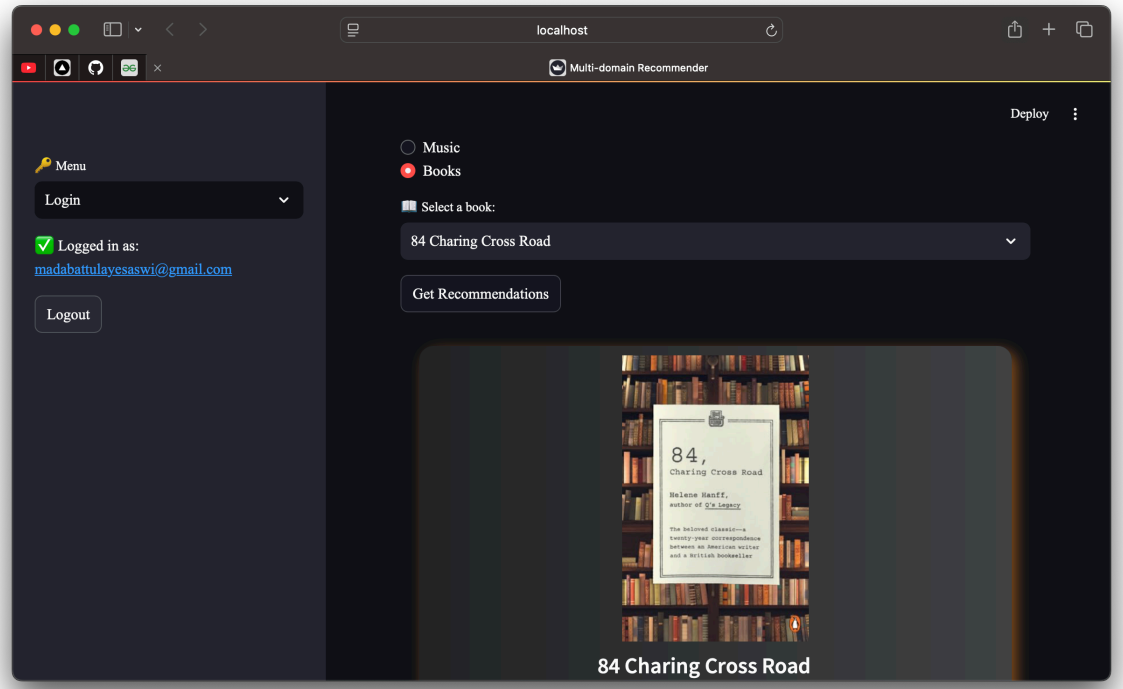

localhost
Centralised Virtual Assistant - Presentation - Canva
Multi-domain Recommender
Deploy
Menu
Login
Multi-Domain
Recommender System
Login
Email
Password
Login

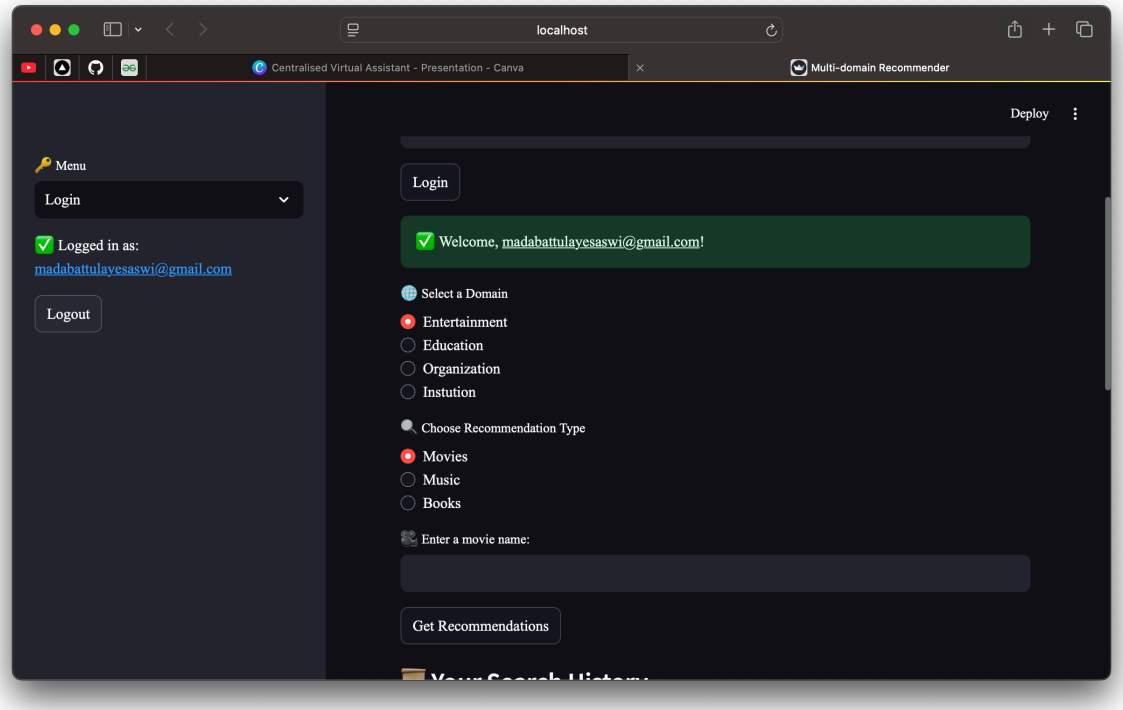

localhost
Centralised Virtual Assistant - Presentation - Canva
Multi-domain Recommender
Deploy
Menu
Login
Multi-Domain
Recommender System
Login
Email
madabattulayesaswi@gmail.com
Password
Press Enter to apply
Login

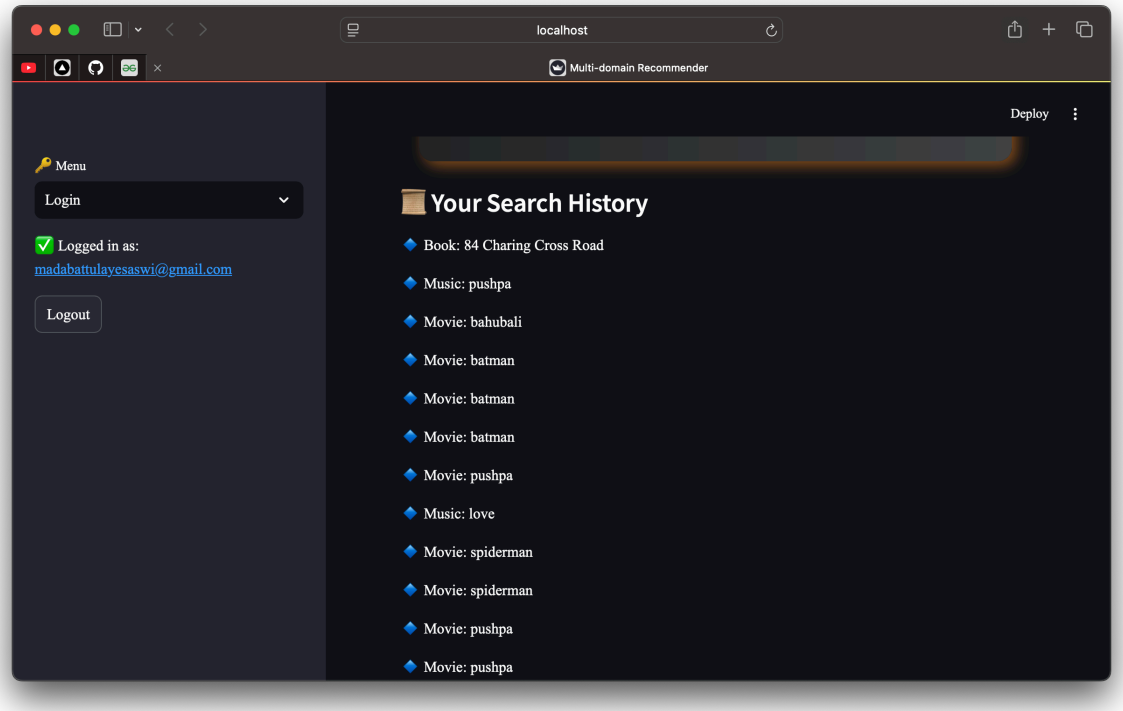

console.firebase.google.com
Multi-domain Recommender
INSTA PATH - Cloud Firestore - Data - Firebase console
Firebase
INSTA PATH
Cloud Firestore
Project Overview
Project shortcuts
Authentication
Storage
Firestore Database
Realtime Database
Data Connect
What's new
Genkit NEW
Vertex AI NEW
Product categories
Build
Spark
No-cost ($0/month)
Upgrade
NEW
Panel view
Query builder
users > YXIK1bqW5zZM.
More in Google Cloud
(default)
users
YXIK1bqW5zZMF26eLzJQvoxiDlu2
Start collection
users
Add document
Start collection
Add field
created_at: "2025-03-06T19:14:48.669543"
email: "madabattulayesaswi@gmail.com"
history
0 "Movie: Interstellar"
1 "Music: pushpa"
2 "Book: 1984"
3 "Movie: Janis: Little Girl Blue"
4 "Music: bahubali"
5 "Book: 4 Blondes"
6 "Movie: july"
7 "Movie: i"


localhost
Multi-domain Recommender
Deploy
Menu
Login
Logged in as:
madabattulayesaswi@gmail.com
Logout
Music
Books
Enter a movie name:
bahubali
Get Recommendations
Bāhubali 2: The Conclusion


localhost
Multi-domain Recommender
Deploy
Menu
Login
Logged in as:
madabattulayesaswi@gmail.com
Logout
Music
Books
Enter a music genre:
pushpa
Get Recommendations
Pushpa Pushpa (From "Pushpa 2 The Rule") [HINDI] by Mika Singh
Listen on Spotify


localhost
Multi-domain Recommender
Deploy
Menu
Login
Logged in as:
madabattulayesaswi@gmail.com
Logout
Music
Books
Select a book:
84 Charing Cross Road
Get Recommendations
84 Charing Cross Road


localhost
Centralised Virtual Assistant - Presentation - Canva
Multi-domain Recommender
Deploy
Menu
Login
Logged in as:
madabattulayesaswi@gmail.com
Logout
Login
Welcome, madabattulayesaswi@gmail.com!
Select a Domain
Entertainment
Education
Organization
Instution
Choose Recommendation Type
Movies
Music
Books
Enter a movie name:
Get Recommendations


localhost
Multi-domain Recommender
Deploy
Menu
Login
Logged in as:
madabattulayesaswi@gmail.com
Logout
Your Search History
Book: 84 Charing Cross Road
Music: pushpa
Movie: bahubali
Movie: batman
Movie: batman
Movie: batman
Movie: pushpa
Music: love
Movie: spiderman
Movie: spiderman
Movie: pushpa
Movie: pushpa