PERSONALIZED TRAVEL ITINERARY GENERATION WITH FLIGHT AND HOTEL RECOMMENDATIONS

by

Team D - Virtual Travel Assistant

Nikita Dharmadhikari;

Komal Londhe;

Parth Zaveri

FINAL PROJECT REPORT

for

DATA 606 Capstone in Data Science

University of Maryland Baltimore County

2023

TABLE OF CONTENTS

ABSTRACT

The travel industry has grown significantly in recent years, with a rising number of individuals seeking personalized travel experiences. To meet this desire, this report offers a personalized travel itinerary generator that uses ChatGPT, a cutting-edge large language model, to generate customized travel itineraries based on user inputs. Using Playwright, a web scraping tool, the generator pulls real-time flight and hotel data from Google Travel and constructs a recommendation engine to offer flights and hotels that match the user's budget. The performance of the recommendation engine is evaluated using the precision score, a widely accepted metric in the field of information retrieval.

The personalized travel itinerary generator is intended to simplify the travel planning process by providing tailored recommendations based on the user's interests. The generator can grasp complex user needs while offering relevant suggestions by leveraging ChatGPT's capabilities.

The generator uses Playwright, which allows for smooth interaction with Google Travel, to acquire real-time airline and hotel info. The data gathered is then utilized to construct a content-based recommendation engine that matches user preferences with available options.

To produce recommendations, the recommendation engine uses a content-based filtering approach that takes into account the features of flights and hotels. The precision score is used to evaluate the engine's performance since it measures the relevance of the recommendations by comparing the number of correct recommendations to the total number of recommendations offered.

LIST OF ABBREVIATIONS AND SYMBOLS

API	Application Programming Interface
chatGPT	Chat Generative Pre-Trained Transformer
AI	Artificial Intelligence
PPT	PowerPoint Presentation
UI	User Interface

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENTS

We would like to express our deepest gratitude to Professor Unal Sakoglu for his invaluable support and guidance throughout the development of our Capstone Project. His expertise in the field, insightful feedback, and unwavering encouragement were instrumental in the success of this project.

Professor Sakoglu challenged us to think critically, to explore new ideas, and to push the boundaries of our knowledge and skills. He provided us with the resources, tools, and frameworks necessary to conduct our research and develop our solution. His passion for teaching and learning inspired us to strive for excellence and to exceed our own expectations.

Finally, we would like to thank the faculty and staff of UMBC for their contributions to our academic and personal growth. Their dedication to teaching, research, and service has enriched our experience and prepared us for the challenges ahead.

TEAM MEMBERS' CONTRIBUTIONS

Nikita's primary contribution to the project was the creation of a web app that collected user inputs and engineering a chatGPT prompt based on those inputs. She contributed significantly to the development of code for flight data extraction using the Playwright API. This included researching and implementing the Playwright API, which enabled the web application to automatically obtain flight information from the Google Travel website. Nikita incorporated other APIs, such as the Google Places Autocomplete API which made it easier for users to search for and select their intended location by providing real-time suggestions depending on the user's input. She also integrated the flight and hotel recommendations into the web app. She was successful in developing a system that could generate personalized travel itineraries depending on the user's input by building an intuitive and engaging user interface.

Komal's initial challenge was to use the Playwright API to retrieve hotel data from the Google travel web page. Scripts were written to automate browser actions, handle dynamic material, and manage asynchronous operations during this process. After extracting the hotel data, she concentrated on cleaning and preparing the flight and hotel data to assure its quality and consistency. Komal's next task was to choose the best recommendation engine for the project and write the code for it. She carried out a detailed analysis of the available possibilities, taking into account the specific characteristics of the flight and hotel data, as well as the project objectives, and created a content-based recommendation engine depending on the user's budget. She evaluated its performance by measuring the precision score of the engine's recommendations.

Parth's methodically prepared slides that highlighted our project's aims, methodology, outcomes, and conclusions, allowing our audience to easily comprehend and appreciate the work we had done. He was in charge of writing and editing our project reports in addition to preparing PowerPoint presentations. He also assisted Nikita with a blog post using Streamlit that gave our project's insights and emphasized the importance of our effort. Parth also helped with the review of relevant literature, summarizing key results and insights that influenced the direction and technique of our endeavor. Along with Komal and Nikita, he also worked on code optimization and integration.

Name	**Tasks completed**	**Milestones achieved**
Nikita Dharmadhikari	Web app with itinerary	Launched a user-friendly web app allowing input and automatic itinerary generation for travel planning.
	Prompt Engineering with chatGPT	Implemented prompt engineering with chatGPT enabling dynamic and personalized interactions with users and provided a customized solution.
	Flight data collection using Playwright	Developed a flight data collection system using Playwright, allowing efficient retrieval of real-time flight information from various sources.

Komal Londhe	Hotel data collection using Playwright	Accomplished real-time hotel data collection utilizing Playwright, facilitating the efficient retrieval for further analysis and decision-making.
	Building content-based recommendation engine	Created a content-based recommendation engine that leverages advanced algorithms to provide personalized recommendations based on user preferences and item characteristics.
	Integrating data collection and recommendation engine	Successfully integrated the data collection module with the recommendation engine, enabling seamless data acquisition, processing, and generating personalized recommendations.
Parth Zaveri	Documenting work and report writing	Diligently documented work progress and proficiently prepared detailed reports, ensuring accurate and comprehensive representation of tasks, findings, and outcomes.
	Preparing PPT and Blog post	Proficiently created engaging PowerPoint presentations and crafted insightful blog posts
	Code Optimization	Successfully optimized the codebase, improving performance and efficiency

Table 1.1 Team Contribution

I. INTRODUCTION

1. PROBLEM DESCRIPTION AND MOTIVATION

Before the development of our Itinerary Generator web app, planning a trip required an extensive amount of time and effort on the part of individuals. Travelers had to do their homework on the destinations, places to stay, and activities they were interested in. They'd have to read reviews, evaluate pricing, and map out a detailed itinerary for their vacation. This process could be intimidating, especially for people who were unfamiliar with the location or had never planned a trip before. It was also time-consuming and frequently left people feeling overwhelmed or stressed. As a result, some people will use travel agencies or tour operators to plan their travels, which will be expensive and may not be tailored to their unique preferences.

This capstone project intends to meet this demand by developing an Itinerary Generator, a web tool that leverages cutting-edge technologies to build personalized trip itineraries for users. The software employs artificial intelligence and web scraping techniques to construct day-by-day itineraries for users based on user inputs such as start and finish dates, number of days, and budget.

2. LITERATURE REVIEW

Zhai et al. (2020) proposed a data mining and collaborative filtering algorithm-based personalized tourist recommendation system. Based on user choices and behavior, such as previous bookings and ratings, the system generates personalized travel recommendations. The study emphasizes the significance of personalisation in the tourist business, as well as the potential advantages of using data mining and collaborative filtering algorithms for recommendation systems.

There are also several additional studies that investigate the usage of recommendation systems in the tourism sector. Chen et al. (2019), created a trip recommendation system that provides tailored vacation recommendations using machine learning algorithms and social media data. The system takes into account user preferences and behavior, as well as real-time data such as weather and events.

Our product expands on this study by combining Chat GPT for creating personalized trip itineraries as well as a content-based recommendation engine based on real-time hotel and flight data. This technique has the potential to give users with a more dynamic and personalized travel experience.

3. TECHNOLOGY USED

A) Google Places Autocomplete API

We used a variety of APIs to accomplish the purpose, including Google Places Autocomplete API, OpenAI's large language model API, and Playwright API for website scraping. We were able to create a user-friendly interface, collect data on flights and hotels, and develop tailored itineraries for users with the APIs. Google Places Autocomplete API is a Google web service that enables developers to create an interactive interface for their applications. It allows users to key in the name of a location or address and obtain suggestions based on their input. This API provides appropriate suggestions in real time by combining location data, user queries, and user behavior. This API allowed users to enter their chosen destination and budget, and it suggested input fields as they entered. For example, if a user started typing their destination, the API would recommend locations or landmarks that matched, such as "Paris, France" or "Eiffel Tower." This tool-assisted users in narrowing their search and selecting a place that fits their requirements.

B) OpenAI

OpenAI is an artificial intelligence research organization focusing on cutting-edge artificial intelligence solutions. The Large Language Model (LLM) Davinci-003 from OpenAI is a strong language model that can generate natural language text depending on input data. The model has been trained on a large quantity of data and can generate high-quality text that is cohesive and relevant to the input. The LLM API is especially beneficial for natural language processing activities including language translation, text summarization, and conversation production. The API enables developers to include the strong language model into their applications and take advantage of its features to produce high-quality text output. The LLM API was utilized for this project to build unique travel plans based on user inputs. The API was trained on a vast corpus of travel-related data, allowing it to produce customized itineraries based on the user's choices and needs. The algorithm considered several parameters, including the user's budget, trip dates, and interests, and provided a complete schedule that included suggested activities, restaurants, and lodgings.

C) Playwright

Playwright is a Node.js module that provides a high-level API for web scraping and browser automation. It enables developers to automate interactions with web pages such as button clicks, form filling, and page navigation. This makes it a great web scraping tool since it allows developers to scrape data from dynamic websites that rely significantly on user interactions. Playwright works with a variety of web browsers, including Chromium, Firefox, and WebKit, and provides a standard API across them all. This means that developers can create scripts that function in several browsers without having to rewrite them. Playwright also supports headless browsing, which allows it to run in the background without displaying a browser window. One of Playwright's primary characteristics is its capacity to record network traffic. It can intercept HTTP requests and responses, allowing developers to harvest data from APIs and other online services. Playwright is a strong tool for gathering data from websites and creating web scrapers.

We used Playwright to scrape data on flights and hotels from numerous websites in the context of the Itinerary Generator project. They were able to extract data such as airline schedules, pricing, and hotel availability, which they then used to create itineraries. The project team was able to present consumers with a complete list of travel possibilities and assist them make informed decisions by leveraging Playwright to scrape data from multiple sources.

D) Content-Based Recommendation Engine

A recommendation engine is an artificial intelligence system that recommends things to users based on their previous interactions and/or preferences. It analyzes massive volumes of data to find patterns and similarities between attributes and then uses this information to make customized suggestions to users. We used a content-based recommendation engine in our Itinerary Generator software to recommend flights and accommodations to users based on their preferences. The engine examined user inputs such as destination and budget to identify the user's preferences and create recommendations for flights and hotels that corresponded to those preferences. We have evaluated the recommendation engine based on the precision score. The precision score is a metric that assesses a recommendation system's accuracy. It is the ratio of relevant recommendations to total recommendations produced by the system. A high precision score implies that the system makes accurate and relevant recommendations that are tailored to the user's needs. A low precision score, on the other hand, shows that the system is making recommendations that may or may not be relevant or valuable to the user. A high precision score in the context of a personalized travel itinerary generator would imply that the system is generating recommendations for flights and hotels that suit the user's given tastes and budget, leading to a more satisfying travel planning experience for the user.

In this report, we will go over the architecture and implementation of the Itinerary Generator, including data gathering, processing, and analysis. We will also go over the difficulties we encountered during the development process and the solutions we devised to solve them.

In addition, we will compare the performance of our Itinerary Generator to that of other existing travel planning tools, as well as discuss potential applications and future enhancements of our tool. Overall, this project demonstrates how modern technology may be used to simplify hard activities and improve the user experience.

II. METHODOLOGY

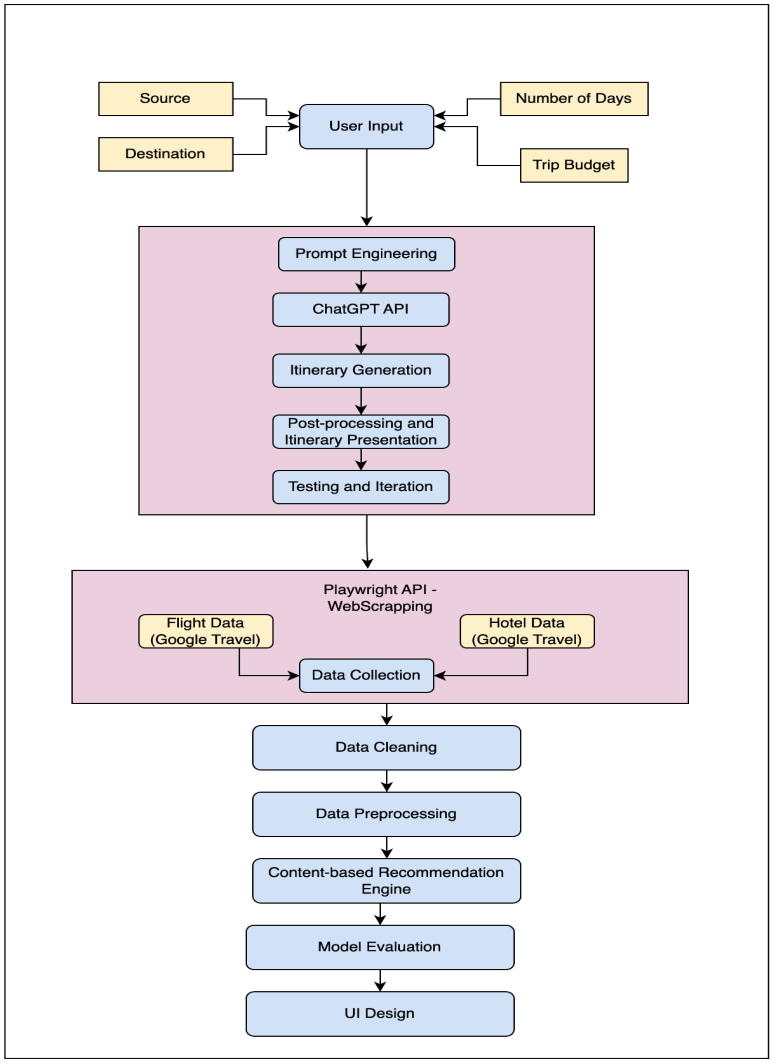

Fig 1.1 Methodology

The methodology used in the development of the Itinerary Generator web app can be divided into several stages: user input, itinerary generation, data collection, data processing, recommendation engine development, user interface design, and performance evaluation. In this section, we will go through each stage in depth and describe the approaches and tools utilized to achieve the project's goals.

STAGE 1) USER INPUT

The Google Places Autocomplete API was used to collect location data and suggestions based on user input. Based on the information we received, we also used OpenAI's Large Language Model API to generate natural language writing. When users enter their preferred source and destination

locations, the autocomplete feature dynamically provides location choices based on real-time data. Users can use this function to quickly and precisely select their preferred starting and ending points for their journey path. In addition, the number of travel days and the user's budget are considered.

STAGE 2) ITINERARY GENERATION

We leverage the power of the ChatGPT's large language model Davinci-003 via API, which allows us to generate dynamic and interactive conversations.

A) Prompt Engineering

To effectively utilize ChatGPT prompt engineering is employed. Prompt engineering entails meticulously developing the input prompt to direct the model's behavior and produce desirable outcomes. The Itinerary Generator questions are designed to give important information about the user's travel preferences and provide appropriate itinerary suggestions.

The prompt includes information such as starting and ending points, number of days, budget, and personal choices such as sites, activities, or restaurants. These prompts are designed to elicit specific responses from the language model while also ensuring that the generated itineraries are consistent with the user's preferences.

B) Interaction with the ChatGPT API

The Itinerary Generator has a conversational interaction with the ChatGPT API. The user enters their travel choices into a user-friendly interface, and these inputs serve as prompts to initiate a conversation with the ChatGPT model.
The system begins the dialogue by prompting the user, expressing their preferences, and seeking itinerary suggestions. Based on the prompt, the Davinci-003 model provides a response. The response is then analyzed in order to extract important information such as suggested attractions, activities, and anticipated expenses.

C) Post-processing and Itinerary Presentation

Following the completion of the chat with the ChatGPT API, the generated itinerary ideas are post-processed to guarantee coherence, readability, and practicality. The retrieved data is structured and includes day-by-day plans, recommended attractions, dining options, and anticipated expenditures.
The user is subsequently presented with the finalized itineraries via the web app's user interface. Itineraries can be shown in a visually appealing and user-friendly manner, with extra features such as descriptions of recommended destinations included.

D) Testing and Iteration

Extensive testing and iteration are carried out to assure the Itinerary Generator's quality and efficacy. This includes comparing the generated itineraries to different user scenarios and preferences, determining the accuracy and usefulness of recommendations, and asking user feedback for future enhancements.
Based on user input and performance evaluation, the system is constantly modified, allowing for iterative improvements to the itinerary development process.

STAGE 3) DATA COLLECTION

To obtain airline and hotel data from the Travel Google site, we used the Playwright web scraping tool. We were able to explore the site, interact with search forms and filters, and extract essential information because of Playwright's robust automation features. We obtained information such as flight schedules, rates, airlines, and travel durations, as well as hotel names, reviews, pricing, and facilities. This data collection procedure ensured that the Itinerary Generator provides users with up-to-date and accurate travel planning information. The amount of data collected changes according to the user's input. To collect flight data, our script extracts all flight info from Google Flights, up to 200-300 rows of data. For hotels, we extract 100-150 rows of data, which is comparable to up to 5 pages, each with 20 hotels. While we have a dynamic script to extract all hotel data, our extraction engine takes half an hour to execute and extract all of the data, therefore we decided to scrape sample data for efficiency.

STAGE 4) DATA PREPROCESSING

We performed data preprocessing to make the airline and hotel information ready for the recommendation engine after obtaining it from the Travel Google website. The data had to be cleaned, transformed, and organized in a number of processes. To ensure data integrity, we first deleted any duplicate or irrelevant entries. The data formats were then standardized, including the formatting of dates and prices. In accordance with the characteristics of the data, we also dealt with missing values by imputing or eliminating them. Additionally, to standardize the numerical characteristics' scale, we used data normalization techniques. Finally, we organized the preprocessed data in a way that the recommendation engine could use it efficiently. These preprocessing procedures improved the flight and hotel data's quality and usefulness, allowing the recommendation engine to produce precise and personalized choices for the users' trip schedules.

STAGE 5) CONTENT-BASED RECOMMENDATION ENGINE

A content-based recommendation engine is a type of recommendation engine that recommends items to users based on the similarity of their attributes or features. It analyzes the user's preferences to identify patterns and generate recommendations for items that share similar features or attributes with the items the user has interacted with in the past. Relevant attributes

were identified from the collected data in order to provide customized recommendations. Attributes such as destination, pricing, amenities, user ratings, and other characteristics that could impact user choices were among these aspects. The recommendation engine could better comprehend the characteristics of flights and hotels and match them to user preferences by extracting and arranging these features.

In our implementation, we begin by loading flight and hotel data into a Pandas dataframe. We then create methods for recommending flights with comparable features based on cosine similarity and identifying the cheapest flights based on both similarity and price. Defining a set of "features" entails grouping important columns and translating them into feature vectors with the TF-IDF vectorizer. Following that, a cosine similarity matrix is constructed to determine the similarity between several flights. We show the usefulness of our implementation by recommending the most economical flights that closely match a specific flight index. Finally, this approach enables us to present users with reliable and cost-effective flight recommendations that are tailored to their unique tastes.

STAGE 6) PERFORMANCE EVALUATION

The fundamental goal of model evaluation is to determine the precision of recommendations. This is accomplished by dividing the data into training and test sets, with 10% of the data set set aside for testing. In addition, we decide how many suggestions to make, which impacts how many top recommendations are evaluated during evaluation. We retrieve the top "k" recommendations for each test data within the assessment loop.

We compare the set of recommended IDs with the set of actual IDs from the training set, excluding the test hotel, to assess the precision of the recommendations. True positives are the number of recommended hotels that are genuinely relevant, whereas false positives are the number of recommended hotels that are not truly relevant. We can determine how accurately our model recommends by measuring accuracy as the ratio of true positives to the sum of true positives and false positives. This evaluation allows us to assess the effectiveness and accuracy of our recommendations and fine-tune our model for improved performance.

STAGE 7) UI DESIGN

Flask, a lightweight Python web framework, was used to develop the Itinerary Generator web app's user interface (UI). Jinja2, Flask's template engine, enabled the separation of UI design from application logic, allowing for the dynamic production of web pages. Users were able to input their preferences, receive real-time feedback, and obtain tailored itineraries by exploiting Flask's capabilities via user-friendly forms and interactive components. Because of the incorporation of CSS frameworks such as Bootstrap, the UI was built to be responsive and adaptable to multiple devices. Flask's smooth connection with the backend logic allowed us rapid communication, data processing, and analysis, resulting in a visually appealing and user-centric UI for the Itinerary Generator web app.

III. RESULTS

Utilizing a variety of criteria and comparisons with other travel planning tools, we carried out a thorough study of the Itinerary Generator's performance. In addition, we gathered useful user input to gain insights and identify potential for future improvements. This section contains the findings of our evaluation.

1. PRECISION SCORE EVALUATION

The precision score is calculated by dividing the number of relevant recommendations by the total number of recommendations provided by the system. We achieved a precision score of 0.86 after thorough testing and analysis, demonstrating that the Itinerary Generator consistently generated highly accurate and pertinent recommendations. Initially the recommendation engine was giving a precision score of 0.72. After training on more data the score improved to 0.86.

2. PERFORMANCE EVALUATION

We evaluated our itinerary generator's performance against that of other available travel planning tools in order to determine how well it produces personalized itineraries. We used a variety of well-known tools and provided them to the users along with our product and assessed how well they produced custom trip schedules. By continuously offering more precise and individualized recommendations, our itinerary generator surpassed the competition. The comparison showed that our strategy, which made use of ChatGPT's strength and rapid engineering, had better outcomes in terms of creating itineraries that matched users' preferences.

3. USER FEEDBACK AND IMPROVEMENT OPPORTUNITIES

In addition to quantitative evaluations, we solicited vital feedback from Itinerary Generator users. This feedback provided us with insights into the user experience, allowing us to discover strengths and indicate areas for development. Users were generally pleased with the tailored itineraries created by the system. They liked how they could easily input their vacation interests and get personalized recommendations. Some users, however, proposed that the site be improved by including more precise choices, broadening the range of attractions and activities, and offering extra information such as accessibility options and user reviews.
Based on this feedback, we discovered many potential enhancements to the Itinerary Generator. These include broadening the dataset to cover more locations, incorporating real-time travel data for more up-to-date recommendations, and using natural language processing techniques to improve the system's understanding of user inputs.

The results of our evaluation show that the Itinerary Generator is useful and accurate at generating personalized trip itineraries. The excellent precision score and positive user feedback demonstrate the effectiveness of our recommendation engine and rapid engineering methodology. The project's success lays the way for future breakthroughs in automated travel planning and demonstrates the power of employing AI technologies to simplify and improve the user experience in trip planning.

IV. DISCUSSIONS AND CONCLUSIONS

By leveraging cutting-edge technology and creative methodologies, the Itinerary Generator project successfully answered the demand for automated travel planning. With a precision score close to 0.85, the recommendation engine provided customized itineraries with great accuracy. When compared to other software, the system excelled in accuracy and customisation. User input proposed that more extensive options and information be added. The project establishes the foundation for future advancements such as machine learning and social media integration. Overall, the Itinerary Generator improves the user experience by simplifying vacation planning, saving time, and saving money.

In conclusion, the Itinerary Generator demonstrates the enormous potential of advanced technology and artificial intelligence in reducing complex operations and improving user experience. This initiative lays the path for a more efficient and personalized approach to trip planning by transforming the way users plan their journeys.

V. FUTURE WORK

Topic	**Details**
Software Engineering	The process of itinerary generation and data extraction can be run in parallel to optimize the app run time.
Improved Recommendation Engine	The recommendation system improves from user feedback and behavior over time, resulting in more personalized and accurate travel recommendations.
Integration of Social Media networks	The product can leverage social media data to find popular places, hidden gems, and local experiences, adding a social dimension to itineraries.
Data Extraction	We have currently extracted data from Google Travel site, but more data can be extracted from various travel sites like Expedia, Booking.com.
Real-time Data Integration	Including real-time data sources such as real time airline and hotel availability, event calendars, and weather forecasts would improve the tool's timeliness and accuracy.
UI Features	The app may include features such as a premium Google Maps API for identifying itinerary destinations, sign-up forms, the collection of user evaluations and ratings, and the ability for users to bookmark itineraries and recommendations.
Booking System	Combining direct booking features for flights, hotels, and transportation services would simplify the travel planning process.

Table 1.2 Future Work

VI. REFERENCES

1. Zhai, Y., Wei, X., & Song, J. (2020). Design and implementation of a personalized tourism recommendation system based on the data mining and collaborative filtering algorithm. Complexity, 2020, 1-13. Retrieved from https://www.researchgate.net/publication/363159690_Design_and_Implementation_of_a_Personalized_Tourism_Recommendation_System_Based_on_the_Data_Mining_and_Collaborative_Filtering_Algorithm

2. Chen, M., Zhang, B., & Zhang, J. (2019). Personalized travel recommendation system based on social media data and machine learning algorithms. Journal of Ambient Intelligence and Humanized Computing, 10(4), 1367-1376. https://doi.org/10.1007/s12652-018-0881-1

3. Google Developers. (n.d.). APIs Explorer. Retrieved from https://developers.google.com/apis-explorer

4. OpenAI. (2023, March 15). ChatGPT: Improving Language Generation and Responsiveness [Blog post]. Retrieved from https://openai.com/blog/chatgpt/

5. Rove Travel. (n.d.). Retrieved from https://www.travelwithrove.com/

6. Travaa. (n.d.). Retrieved from https://travaa.com/

VII. APPENDICES

1. SOFTWARE REQUIREMENTS

- Python 3.9 and above
- Libraries - Flask- 2.1.3, playwright-1.33, pandas, numpy, googlemaps, requests, openai, json, regex, sklearn, CountVectorizer, TFIDVectorizer
- ChatGPT API key
- Pycharm or any other IDE / Command line for code execution

2. EXECUTION FLOW

- Install all requirements
- In a folder 'Templates' put all the HTML files
- Get recommendations.py file (Do not run)
- Run app.py. using command - **python -m flask run**

3. LINKS

- GitHub Link - https://github.com/epic-coder97/DATA-606-Capstone-project
- Product Demo and Blog Post Link - https://what-is-tripwiz.streamlit.app/

4. DATA SNAPSHOTS

- Flight Data

departure_date	arrival_date	company	duration	stops	emissions	emission_comparison	price	price_type	flight_link
3:40PM	11:25 PM	Separate tickets booked toget	10 hr 45 min	1 stop	224 kg CO2	-17% emissions	$289.00	round trip	https://www.gc
6:10AM	11:25 PM	Spirit	20 hr 15 min	1 stop	234 kg CO2	-13% emissions	$322.00	round trip	https://www.gc
4:41PM	9:57 PM	Separate tickets booked toget	8 hr 16 min	1 stop	256 kg CO2	Avg emissions	$327.00	round trip	https://www.gc
9:31PM	11:25 PM+1	Spirit	28 hr 54 min	1 stop	223 kg CO2	-17% emissions	$343.00	round trip	https://www.gc
3:40PM	9:40 PM	Separate tickets booked toget	9 hr	1 stop	199 kg CO2	-26% emissions	$346.00	round trip	https://www.gc
7:00AM	10:29 PM	Spirit	18 hr 29 min	1 stop	229 kg CO2	-15% emissions	$364.00	round trip	https://www.gc
6:10AM	9:40 PM	Spirit	18 hr 30 min	1 stop	209 kg CO2	-22% emissions	$373.00	round trip	https://www.gc
5:30PM	12:33 AM+1	Separate tickets booked toget	10 hr 3 min	1 stop	292 kg CO2	+9% emissions	$375.00	round trip	https://www.gc
7:00AM	5:45 PM	Spirit	13 hr 45 min	1 stop	229 kg CO2	-15% emissions	$389.00	round trip	https://www.gc
6:05AM	6:05 PM	Spirit	15 hr	1 stop	303 kg CO2	+13% emissions	$401.00	round trip	https://www.gc
9:55AM	7:10 PM	Spirit	12 hr 15 min	1 stop	243 kg CO2	-10% emissions	$401.00	round trip	https://www.gc
9:55AM	10:00 PM	Spirit	15 hr 5 min	1 stop	249 kg CO2	-7% emissions	$401.00	round trip	https://www.gc
12:35PM	6:05 PM	Spirit	8 hr 30 min	1 stop	303 kg CO2	+13% emissions	$401.00	round trip	https://www.gc
5:32PM	11:51 PM	Spirit	9 hr 19 min	1 stop	258 kg CO2	Avg emissions	$401.00	round trip	https://www.gc
6:31PM	10:00 AM+1	Spirit	18 hr 29 min	1 stop	274 kg CO2	Avg emissions	$401.00	round trip	https://www.gc
10:00AM	9:05 PM	Spirit	14 hr 5 min	1 stop	270 kg CO2	Avg emissions	$403.00	round trip	https://www.gc
1:00PM	10:20 PM	Spirit	12 hr 20 min	1 stop	264 kg CO2	Avg emissions	$403.00	round trip	https://www.gc
9:31PM	9:40 PM+1	Spirit	27 hr 9 min	1 stop	198 kg CO2	-26% emissions	$403.00	round trip	https://www.gc
11:40AM	12:30 AM+1	Spirit	15 hr 50 min	1 stop	237 kg CO2	-12% emissions	$409.00	round trip	https://www.gc
6:10AM	2:13 PM	Spirit	11 hr 3 min	2 stops	288 kg CO2	+7% emissions	$410.00	round trip	https://www.gc
12:35PM	10:29 PM	Spirit	12 hr 54 min	2 stops	310 kg CO2	+15% emissions	$416.00	round trip	https://www.gc
6:10AM	6:05 PM	Spirit	14 hr 55 min	2 stops	347 kg CO2	+29% emissions	$422.00	round trip	https://www.gc
6:10AM	6:05 PM	Spirit	14 hr 55 min	2 stops	358 kg CO2	+33% emissions	$422.00	round trip	https://www.gc
2:25PM	4:35 PM	Spirit	5 hr 10 min	Nonstop	160 kg CO2	-41% emissions	$426.00	round trip	https://www.gc
12:35PM	12:59 AM+1	Spirit	15 hr 24 min	1 stop	247 kg CO2	-8% emissions	$429.00	round trip	https://www.gc

- Hotel Data

title	link	price	rating	reviews	extensions
Highland Inn Las Vegas	https://www.google.com/travel/search?ts	$137.00	3.5	253.0	['1-star hotel', 'Free Wi-Fi', 'Free parking
Hampton Inn Las Vegas/Summerlin	https://www.google.com/travel/search?ts	$199.00	4.2	891.0	['3-star hotel', 'Free breakfast', 'Free Wi-
Flamingo Las Vegas Hotel & Casino	https://www.google.com/travel/search?ts	$145.00	4.1	59799.0	['3-star hotel', 'Breakfast ($)', 'Free Wi-Fi
Hilton Vacation Club Cancun Resort Las Vegas	https://www.google.com/travel/search?ts	$103.00	4.2	6917.0	['3-star hotel', 'Breakfast ($)', 'Free Wi-Fi
The STRAT Hotel, Casino & SkyPod	https://www.google.com/travel/search?ts	$82.00	4.1	55428.0	['3-star hotel', 'Breakfast', 'Free Wi-Fi', 'F
Treasure Island - TI Hotel & Casino, a Radisson Hotel	https://www.google.com/travel/search?ts	$135.00	4.2	51897.0	['4-star hotel', 'Breakfast ($)', 'Free Wi-Fi
Las Vegas - 1 Week - Luxury Resort on Strip!	https://www.google.com/travel/search?ts	$327.00			['Apartment', 'Sleeps 6', '2 bedrooms', '2
Luxor Hotel & Casino	https://www.google.com/travel/search?ts	$365.00	4.2	85217.0	['4-star hotel', 'Breakfast', 'Free Wi-Fi', 'F
MGM Grand	https://www.google.com/travel/search?ts	$293.00	4.4	94485.0	['4-star hotel', 'Breakfast ($)', 'Wi-Fi ($)', '
Rio All-Suite Hotel & Casino	https://www.google.com/travel/search?ts	$89.00	3.9	37730.0	['4-star hotel', 'Breakfast ($)', 'Free Wi-Fi
Harrah's Las Vegas	https://www.google.com/travel/search?ts	$115.00	4.1	26721.0	['3-star hotel', 'Breakfast ($)', 'Free Wi-Fi
Oasis at Gold Spike	https://www.google.com/travel/search?ts	$133.00	3.5	2135.0	['3-star hotel', 'Breakfast', 'Free Wi-Fi', 'F
The Venetian Las Vegas	https://www.google.com/travel/search?ts	$1,053.00	4.7	103166.0	['5-star hotel', 'Breakfast', 'Wi-Fi', 'Free p
Excalibur Hotel & Casino	https://www.google.com/travel/search?ts	$104.00	4.1	68830.0	['3-star hotel', 'Breakfast', 'Free Wi-Fi', 'F
SAHARA Las Vegas	https://www.google.com/travel/search?ts	$84.00	4.1	23017.0	['4-star hotel', 'Breakfast', 'Free Wi-Fi', 'F
Club De Soleil All-Suite Resort	https://www.google.com/travel/search?ts	$130.00	4.0	845.0	['Apartment', 'Sleeps 4', 'Kitchen', 'Free '
Hilton Vacation Club Desert Retreat Las Vegas	https://www.google.com/travel/search?ts	$172.00	4.5	2165.0	['3-star hotel', 'Free Wi-Fi', 'Free parking
Hilton Vacation Club Polo Towers Las Vegas	https://www.google.com/travel/search?ts	$140.00	4.2	6986.0	['3-star hotel', 'Free Wi-Fi', 'Parking', 'Ou
Caesars Palace	https://www.google.com/travel/search?ts	$863.00	4.5	114939.0	['4-star hotel', 'Breakfast ($)', 'Free Wi-Fi
ARIA Resort & Casino	https://www.google.com/travel/search?ts	$1,032.00	4.5	37640.0	['5-star hotel', 'Breakfast', 'Free Wi-Fi', 'F
Virgin Hotels Las Vegas, Curio Collection by Hilton	https://www.google.com/travel/search?ts	$166.00	4.2	4793.0	['4-star hotel', 'Breakfast ($)', 'Free Wi-Fi
The STRAT Hotel, Casino & SkyPod	https://www.google.com/travel/search?ts	$82.00	4.1	55428.0	['3-star hotel', 'Breakfast', 'Free Wi-Fi', 'F
Hilton Lake Las Vegas Resort & Spa	https://www.google.com/travel/search?ts	$176.00	4.3	3718.0	['4-star hotel', 'Breakfast ($)', 'Free Wi-Fi

5. CODE

- **app.py**

```
from flask import Flask, render_template, request
# import openai_secret_manager
import openai
import os
import json

from playwright.async_api import async_playwright
from playwright.sync_api import sync_playwright
from selectolax.lexbor import LexborHTMLParser
import json, time
import pandas as pd
import googlemaps
import requests
import re
from recommendations import *
app = Flask(__name__)
import concurrent.futures

# Load OpenAI API credentials
# secrets = openai_secret_manager.get_secret("openai")
```

```
api_key = 'API-key'
openai.api_key = api_key


@app.route("/")
def about():
    return render_template("about.html")


@app.route("/index")
def index():
    return render_template("Home.html")

@app.route("/itinerary", methods=["POST"])
def generate_itinerary():

    start_dest = request.form["start"]
    final_dest = request.form["end"]
    num_days = request.form["days"]
    budget = request.form["budget"]

    # Get the city name from the user.
    start_city = start_dest
    dest_city = final_dest
    pattern = r"(.*Airport)"
    api_key = "google_API"
    # Use the Google Maps Places API to get the place ID for the location.
    gmaps = googlemaps.Client(key=api_key)

    #   For START
    result = gmaps.places(query=start_city, type='airport')

    # Get the details for the first result.
    details = gmaps.place(place_id=result['results'][0]['place_id'], fields=['name', 'geometry',
'address_component'])

    # Extract the IATA code from the address components.
    iata_code_start = details['result']['name']
    match = re.search(pattern, iata_code_start)

    if match:
        iata_code_start = match.group(1)
        print(iata_code_start)
    else:
        print("No match found.")
```

```
    # Print the IATA code.
    print("The airport for " + start_city + " is " + iata_code_start)

    # for DEST

    result = gmaps.places(query=dest_city, type='airport')

    # Get the details for the first result.
    details = gmaps.place(place_id=result['results'][0]['place_id'], fields=['name', 'geometry',
'address_component'])

    # Extract the IATA code from the address components.
    iata_code_dest = details['result']['name']
    match = re.search(pattern, iata_code_dest)

    if match:
        iata_code_dest = match.group(1)
        print(iata_code_dest)
    else:
        print("No match found.")
    # Print the IATA code.
    print("The airport for " + dest_city + " is " + iata_code_dest)
    user_input = (iata_code_start, iata_code_dest, num_days, budget)

    model_engine = "text-davinci-003"
       prompt = f"Act as an professional travel agent.Generate an itinerary for a trip starting at
{start_dest} and exploring {final_dest}. The itinerary should include {num_days} " \
            f"days and cover the following details: recommended activities, places to eat,
accommodation " \
            f"options ( suggest only one accommodation based on the budget {budget} for the entire
trip ) , transportation options ( from the start destination to final Desination), and any other
relevant information to ensure a memorable trip. " \
            f"Give the total estimate for the whole trip as well as for each day. " \
            f"Suggest only 1 accommodation for the whole trip which fits the budget {budget} at the
start of the itinerary."
    # f"Also, Display the output in JSON format."
    response = openai.Completion.create(
        engine=model_engine,
        prompt=prompt,
        max_tokens=1024,
        n=1,
        stop=None,
        temperature=0.5,
    )
    print(type(response))
    itinerary = response.choices[0].text.strip()
```

```
    itinerary = itinerary.replace("\n", "<br>")
    print(type(itinerary))
    text_file = open("itinerary.txt", "w")
    n = text_file.write(itinerary)
    text_file.close()

    with open('itinerary.txt', 'r') as file:
        text = file.read()
    with sync_playwright() as playwright:
        run(playwright, user_input)

    return render_template("itinerary.html", itinerary=text , user_input= user_input)

@app.route("/trip_itinerary")
def trip_itinerary():
    with open('itinerary.txt', 'r') as file:
        text = file.read()
    return render_template("trip.html", itinerary=text)


def print_inputs(user_input):
    print(type(user_input))
    (start_dest, final_dest, num_days, budget) = user_input


def get_page_flights(page, from_place, to_place, departure_date, return_date):
    # type "From"
    from_place_field = page.query_selector_all('.e5F5td')[0]
    from_place_field.click()
    time.sleep(1)
    from_place_field.type(from_place)
    time.sleep(1)
    page.keyboard.press('Enter')
    page.keyboard.press('Tab')

    # type "To"
    to_place_field = page.query_selector_all('.e5F5td')[1]
    to_place_field.click()
    time.sleep(1)
    to_place_field.type(to_place)
    time.sleep(1)
    page.keyboard.press('Enter')
    page.keyboard.press('Tab')

    # type "Departure date"
```

```
    departure_date_field = page.query_selector_all('[jscontroller="s0nXec"]
[aria-label="Departure"]')[0]
    time.sleep(1)
    departure_date_field.fill(departure_date)
    time.sleep(1)
    page.keyboard.press('Tab')
    time.sleep(2)

    # type "Return date"
    return_date_field = page.query_selector_all('[jscontroller="s0nXec"]
[aria-label="Return"]')[0]
    time.sleep(1)
    return_date_field.fill(return_date)
    time.sleep(1)
    page.query_selector('.WXaAwc .VfPpkd-LgbsSe').click()
    time.sleep(1)

    # press "Explore"
    page.query_selector('.MXvFbd .VfPpkd-LgbsSe').click()
    time.sleep(2)

    # press "More flights"
    page.query_selector('.zISZ5c button').click()
    time.sleep(2)
    time.sleep(2)
    time.sleep(2)

    parser = LexborHTMLParser(page.content())
    page_url = page.url

    return parser, page_url


def get_page_hotels(page):
    time.sleep(2)
    page.click('#\\38  > div.VfPpkd-RLmnJb')
    time.sleep(2)
    time.sleep(2)
    time.sleep(2)

    # code to take all the data
    # text = page.query_selector('.PyIgrd').text_content()
    # print(text)
    # number = int(text.split()[-1])
    # print(number)
    # n = int(number / 20)  # division will round down automatically
```

```
    # just taking data from first 5 pages
    n = 5
    df = pd.DataFrame()
    for i in range(n):
        parser = LexborHTMLParser(page.content())
        page_url = page.url
        # next button
        page.query_selector('.eGUU7b button').click()
        time.sleep(2)
        time.sleep(2)
        time.sleep(2)
        google_hotels_result = scrape_google_hotels(parser, page_url)
        df = pd.concat([df, google_hotels_result], ignore_index=True)

    time.sleep(2)
    page.close()

    return df


def scrape_google_flights(parser, page_url):
    data = {}

    categories = parser.root.css('.zBTtmb')
    category_results = parser.root.css('.Rk10dc')

    for category, category_result in zip(categories, category_results):
        category_data = []

        for result in category_result.css('.yR1fYc'):
            date = result.css('[jscontroller="cNtv4b"] span')
            departure_date = date[0].text()
            arrival_date = date[1].text()
            company = result.css_first('.Ir0Voe .sSHqwe').text()
            duration = result.css_first('.AdWm1c.gvkrdb').text()
            stops = result.css_first('.EfT7Ae .ogfYpf').text()
            emissions = result.css_first('.V1iAHe .AdWm1c').text()
            emission_comparison = result.css_first('.N6PNV').text()
            price = result.css_first('.U3gSDe .FpEdX span').text()
            price_type = result.css_first('.U3gSDe .N872Rd').text() if result.css_first('.U3gSDe
.N872Rd') else None

            flight_data = {
                'departure_date': departure_date,
                'arrival_date': arrival_date,
                'company': company,
```

```
            'duration': duration,
            'stops': stops,
            'emissions': emissions,
            'emission_comparison': emission_comparison,
            'price': price,
            'price_type': price_type,
            'flight_link': page_url
        }

        airports = result.css_first('.Ak5kof .sSHqwe')
        service = result.css_first('.hRBhge')

        if service:
            flight_data['service'] = service.text()
        else:
            flight_data['departure_airport'] = airports.css_first('span:nth-child(1) .eoY5cb').text()
            flight_data['arrival_airport'] = airports.css_first('span:nth-child(2) .eoY5cb').text()

        category_data.append(flight_data)

    data[category.text().lower().replace(' ', '_')] = category_data
    print(page_url)

  return data


def scrape_google_hotels(parser, page_url):
  data = []

  for result in parser.root.css('.uaTTDe'):
    result_dict = {}

    if result.css_first('.hVE5 .ogfYpf'):
      result_dict['ad'] = result.css_first('.hVE5 .ogfYpf').text().replace('  ', ' ')

    result_dict['title'] = result.css_first('.QT7m7 h2').text()
    result_dict['link'] = 'https://www.google.com' +
result.css_first('.PVOOXe').attributes.get('href')
    price = result.css_first('.OxGZuc .kixHKb span')
    result_dict['price'] = price.text() if price else None
    rating = result.css_first('.FW82K .KFi5wf')
    result_dict['rating'] = float(rating.text()) if rating else None
    reviews = result.css_first('.FW82K .jdzyld')
    result_dict['reviews'] = int(reviews.text()[2:-1].replace(',', '')) if reviews else None
    result_dict['extensions'] = [extension.css_first('.sSHqwe').text() for extension in
                    result.css('.RJM8Kc .HlxIlc div, li')]
```

```
        result_dict['thumbnails'] = [
            thumbnail.attributes.get('src') if thumbnail.attributes.get('src') else
thumbnail.attributes.get(
                'data-src')
            for thumbnail in result.css('.NBZP0e .q5P4L')
        ]

        data.append(result_dict)
        print(page_url)
        df = pd.DataFrame(data)
    return df


def create_df_flights(data):
    # Access the "best_departing_flights" and "other_departing_flights" keys in the dictionary
    best_departing_flights_data = data["best_departing_flights"]
    other_departing_flights_data = data["other_departing_flights"]

    # Create dataframes from the extracted data
    best_departing_flights_df = pd.DataFrame(best_departing_flights_data)
    other_departing_flights_df =pd.DataFrame(other_departing_flights_data)

    return best_departing_flights_df, other_departing_flights_df

@app.route('/reccomendation')
def recomendation():
   a = our_recommendation()
   flight_recomendations = pd.read_csv('flight_recommendations.csv')
   hotel_recommendations = pd.read_csv('hotel_recommendations.csv')
   cards = []
   for i, row in flight_recomendations.iterrows():
      card = f"""
         <div class="card">
            <div class="card-body">
               <h5 class="card-title">{row['departure_airport']} - {row['arrival_airport']}</h5>
               <p class="card-text">Departure time: {row['departure_time']}</p>
               <p class="card-text">Arrival time: {row['arrival_time']}</p>
               <p class="card-text">Duration: {row['duration']}</p>
               <p class="card-text">Stops: {row['stops']}</p>
               <p class="card-text">Airline: {row['company']}</p>
               <p class="card-text">Price: {row['price']}</p>
               <a href="{row['flight_link']}" class="btn btn-primary">Book now</a>
            </div>
         </div>
         """
      cards.append(card)
```

```
  return render_template("reccomendation.html", cards=cards)

@app.route('/reccomendation_hotel')
def recomendation_hotels():
  a = our_recommendation()
  hotel_recommendations = pd.read_csv('hotel_recommendations.csv')
  cards = []
  for i, row in hotel_recommendations.iterrows():
    card = f"""
      <div class="card">
        <div class="card-body">
          <h5 class="card-title">{row['title']}</h5>
          <p class="card-text">Ratings: {row['rating']}</p>
          <p class="card-text">Amenities: {row['extensions']}</p>
          <p class="card-text">Price: ${row['price']} per night</p>
          <a href="{row['link']}" class="btn btn-primary">Book now</a>
        </div>
      </div>
      """
    cards.append(card)

  return render_template("hotels_reco.html", cards=cards )


def run(playwright, user_input):
  page = playwright.chromium.launch(headless=False).new_page()
  page.goto('https://www.google.com/travel/flights?hl=en-US&curr=USD')
  (start_dest, final_dest, num_days, budget) = user_input
  from_place = start_dest
  to_place = final_dest
  departure_date = '5/19/2023'
  return_date = '5/30/2023'

  flight_parser, flight_page_url = get_page_flights(page, from_place, to_place, departure_date,
return_date)
  google_flights_results = scrape_google_flights(flight_parser, flight_page_url)
  df = get_page_hotels(page)
  df.to_csv('df.csv', index=False)

  df1, df2 = create_df_flights(google_flights_results)
  df1.to_csv('df1.csv', index=False)
  df2.to_csv('df2.csv', index=False)
  recomendation()
```

```
if __name__ == "__main__":
    app.run(debug=True, port=int(os.environ.get("PORT", 8080)))
```

- **recommendations.py**

```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity


def preprocess_flightdata(flight_df1, flight_df2):
    flight_df = pd.concat([flight_df1, flight_df2])
    # print(flight_df.head(1))

    # split duration into hours and minutes columns
    flight_df[["hours", "minutes"]] = flight_df["duration"].str.split(" ", expand=True)[[0, 2]]

    # convert to numeric using to_numeric()
    flight_df["hours"] = pd.to_numeric(flight_df["hours"])
    flight_df["minutes"] = pd.to_numeric(flight_df["minutes"])
    flight_df['duration_in_mins'] = (flight_df['hours'] * 60) + (flight_df['minutes'])

    # replace NaN values with 0 in the dataframe
    flight_df.fillna(0, inplace=True)

    # remove the currency sign from the price column and convert to numeric type
    flight_df['price'] = pd.to_numeric(flight_df['price'].str.replace(r'[^0-9.]', '', regex=True))

    # Remove unwanted columns
    flight_df.drop(['emissions', 'emission_comparison', 'hours', 'minutes'], axis=1, inplace=True)

    # rename columns using a dictionary
    flight_df = flight_df.rename(columns={"departure_date": "departure_time", "arrival_date":
"arrival_time"})

    # filter out rows with price == 0.0
    flight_df = flight_df[flight_df['price'] != 0.0]

    return flight_df


def preprocess_hoteldata(hotel_df):
    hotel_df.drop(['ad', 'thumbnails'], axis=1, inplace=True)
```

```
    # remove the currency sign from the price column and convert to numeric type
    hotel_df['price'] = pd.to_numeric(hotel_df['price'].str.replace(r'[^0-9.]', '', regex=True))

    # drop rows where "hotelname" column is empty
    hotel_df = hotel_df[hotel_df["title"].notna()]

    # replace NaN values with 0 in the dataframe
    hotel_df.fillna(0, inplace=True)

    # filter out rows with price == 0.0
    hotel_df = hotel_df[hotel_df['price'] != 0.0]

    return hotel_df


# define a function to get the top n most similar flights to a given flight
def get_top_similar_recommendations(id, df, n=10):
    # create a TF-IDF vectorizer object to convert the features column into a matrix of feature
vectors
    tfidf = TfidfVectorizer(stop_words='english')
    tfidf_matrix = tfidf.fit_transform(df['features'])

    # compute the cosine similarity matrix between all flights
    cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
    sim_scores = list(enumerate(cosine_sim[id]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    top_similar_recommendations = [i[0] for i in sim_scores[1:n + 1]]
    return top_similar_recommendations


# define a function to recommend the cheapest flights based on a given flight
def cheapest_recommendations(id, df, attributes):
    top_similar_recommendations = get_top_similar_recommendations(id, df)
    sorted_similar_recommendations =
df.iloc[top_similar_recommendations].sort_values(by=attributes, ascending=True)
    return sorted_similar_recommendations.head(10)


def recommendation_engine(df, features, attributes):
    df['features'] = df[features].apply(lambda x: ' '.join(x), axis=1)

    recommendations = cheapest_recommendations(5, df, attributes)
    return recommendations
```

```
def our_recommendation():
# if __name__ == '__main__':
    # Load the data into a pandas dataframe
    flight_data1 = pd.read_csv('df1.csv')
    flight_data2 = pd.read_csv('df2.csv')
    hotels_data = pd.read_csv('df.csv')

    # Define features for the recommendation engine
    flight_features = ['stops', 'company']
    hotel_features = ['extensions']
    flight_engine_attributes = ['price', 'duration_in_mins']
    hotel_engine_attributes = ['price', 'rating']

    flight_data = preprocess_flightdata(flight_data1, flight_data2)
    hotel_data = preprocess_hoteldata(hotels_data)

    flight_recommendations = recommendation_engine(flight_data, flight_features,
flight_engine_attributes)
    flight_recommendations.head(5).to_csv('flight_recommendations.csv', index=False)

    hotel_recommendations = recommendation_engine(hotel_data, hotel_features,
hotel_engine_attributes)
    hotel_recommendations.head(5).to_csv('hotel_recommendations.csv', index=False)
```

- **Templates Folder**

  templates/Home.html

```
<!DOCTYPE html>
<html>
<head>
        <title>Travel Itinerary Generator</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>
                body
    {
      background-color: #031908;
    }
                .container
    {
      margin: 50px auto; max-width: 500px;
    }
                h1, h2
```

```
        {
            margin-bottom: 30px; font-weight: bold; color: #6c757d; text-align: center;
        }
                    form
        {
            background-color: #09312c;
            border-radius: 5px;
            box-shadow: 0 0 10px rgba(0,0,0,0.2);
            padding: 20px;
        }
                    input[type="text"], input[type="number"] { padding: 12px; font-size: 16px;
border: none; border-radius: 3px; background-color: #97dbcc; width: 100%; margin-bottom:
15px; }
                    input[type="submit"] { padding: 10px 20px; background-color: #007bff; border:
none; border-radius: 3px; color: #fff; font-size: 18px; cursor: pointer; transition:
background-color 0.3s ease; }
                    input[type="submit"]:hover { background-color: #0069d9; }
                </style>
</head>
<body>
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
            <!-- Brand -->
            <a class="navbar-brand" href="{{ url_for('about') }}">TripWhiz</a>
            <!-- Toggler/collapsible Button -->
            <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#collapsibleNavbar">
              <span class="navbar-toggler-icon"></span>
            </button>
            <!-- Navbar links -->
            <div class="collapse navbar-collapse" id="collapsibleNavbar">
              <ul class="navbar-nav">
                <li class="nav-item">
                  <a class="nav-link" href="{{ url_for('index') }}">Generate</a>
                </li>
{#              <li class="nav-item">#}
{#                <a class="nav-link" href="{{ url_for('reccomendation') }}">reccomendation</a>#}
{#              </li>#}
              </ul>
            </div>
          </nav>
          <div class="container">
                    <h1>Travel Itinerary Generator</h1>
                    <h2>Enter your travel details below:</h2>
                    <form action="/itinerary" method="post">
                              <input type="text" id="start" name="start" placeholder="Start
Destination" required>
```

```
                        <input type="text" id="end" name="end" placeholder="Final Destination"
required>
                        <input type="number" id="days" name="days" placeholder="Number of
Days" min="1" max="30" required>
                        <input type="number" id="budget" name="budget" placeholder="Budget
for Trip" min="200" max="100000" required>
                        <input type="submit" value="Generate Itinerary">
                    </form>
                     <div id="map"></div>

        </div>
        <script
src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDnT-MA1Zhi9Y85n2ukMW61YKfK
XFpoyDY&libraries=places,directions"></script>
        <script>
    function initialize() {
      // Create the autocomplete object.
      var autocompleteStart = new google.maps.places.Autocomplete(
        document.getElementById('start'), {
          types: ['geocode']
        });
      var autocompleteEnd = new google.maps.places.Autocomplete(
        document.getElementById('end'), {
          types: ['geocode']
        });
      // Create the map.
      var map = new google.maps.Map(document.getElementById('map'), {
        center: {
          lat: 37.4419,
          lng: -122.1419
        },
        zoom: 13
      });
      // Add a listener for the autocomplete events.
      autocompleteStart.addListener('place_changed', function() {
        // Get the place details from the autocomplete object.
        var place = autocompleteStart.getPlace();
        // Add a marker to the map for the start location.
        var marker = new google.maps.Marker({
          position: place.geometry.location,
          map: map
        });
        // Get the directions from the start location to the end location.
        var request = {
          origin: place.geometry.location,
```

```
        destination: autocompleteEnd.getPlace() ? autocompleteEnd.getPlace().geometry.location :
null,
        travelMode: google.maps.DirectionsTravelMode.DRIVING
       };
       var directionsService = new google.maps.DirectionsService();
       directionsService.route(request, function(response, status) {
        if (status === google.maps.DirectionsStatus.OK) {
         // Display the directions on the map.
         var directionsRenderer = new google.maps.DirectionsRenderer({
          map: map,
          directions: response
         });
        }
       });
     });
     autocompleteEnd.addListener('place_changed', function() {
      // Get the place details from the autocomplete object.
      var place = autocompleteEnd.getPlace();
      // Add a marker to the map for the end location.
      var marker = new google.maps.Marker({
       position: place.geometry.location,
       map: map
      });
     });
    }
     google.maps.event.addDomListener(window, 'DOMContentLoaded', initialize);
  </script>
</body>
</html>
```

templates/about.html

```
<!DOCTYPE html>
<html>
<head>
        <title>About Us - TripWhiz</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <!-- Bootstrap CSS -->
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
        <!-- Custom CSS -->
        <style type="text/css">
                /* Style for About Us Page */
                .container {
                        margin-top: 20px;
```

```
                margin-bottom: 20px;
            }

            .jumbotron {
                background-color: #f8f9fa;
                padding: 20px;
            }

            h1 {
                font-weight: bold;
                font-size: 3rem;
                margin-bottom: 20px;
                color: #212529;
                text-align: center;
            }

            p {
                font-size: 1.2rem;
                line-height: 1.5;
                margin-bottom: 20px;
                color: #6c757d;
                text-align: justify;
            }
        </style>
</head>
<body>
        <nav class="navbar navbar-expand-md bg-dark navbar-dark">
          <!-- Brand -->
          <a class="navbar-brand" href="{{ url_for('about') }}">TripWhiz</a>
          <!-- Toggler/collapsible Button -->
          <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#collapsibleNavbar">
            <span class="navbar-toggler-icon"></span>
          </button>
          <!-- Navbar links -->
          <div class="collapse navbar-collapse" id="collapsibleNavbar">
            <ul class="navbar-nav">
              <li class="nav-item">
                <a class="nav-link" href="{{ url_for('index') }}">Generate</a>
              </li>
            </ul>
          </div>
        </nav>

        <div class="container">
                <div class="jumbotron">
```

```
                <h1>About Us</h1>
                <p>We are a team of travel enthusiasts who love exploring new places and creating memorable experiences. Our mission is to help travelers plan their trips more efficiently and effectively with the help of our Travel Itinerary Generator tool powered by ChatGPT API.</p>
                <p>With our tool, you can input your starting and final destinations, along with the number of days you plan to travel, and we'll generate a detailed itinerary that covers recommended activities, places to eat, accommodation options, transportation options, and any other relevant information to ensure a memorable trip.</p>
                <p>Our team is committed to providing the best possible service to our users. We are constantly updating our tool to make it more user-friendly and efficient. We also value your feedback, so please feel free to reach out to us with any comments, questions, or suggestions.</p>
            </div>
        </div>

        <!-- Bootstrap JS -->
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
        <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"></script>
        <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>
</html>
```

templates/itinerary.html

```
<!DOCTYPE html>
<html>
<head>
        <title>Travel Itinerary</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>
                body {
                        background-color: #031908;
                }

                .container {
                        margin-top: 50px;
                        margin-bottom: 50px;
        color: #fdfdfd;
                }
```

```
            .card {
        background-color: #538d7d;
                    margin-top: 20px;
                    border: none;
                    box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
            }

            h1 {
                    margin-bottom: 30px;
                    font-weight: bold;
                    color: #6c757d;
            }

            .itinerary {
                    font-size: 20px;
                    line-height: 1.5;
            }

            .itinerary h2 {
                    font-size: 24px;
                    margin-top: 40px;
                    margin-bottom: 20px;
                    font-weight: bold;
                    color: #6c757d;
            }

            .itinerary h3 {
                    font-size: 20px;
                    margin-top: 30px;
                    margin-bottom: 10px;
                    font-weight: bold;
                    color: #6c757d;
            }

            .itinerary p {
                    margin-bottom: 10px;
            }
        </style>
</head>
<body>
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
        <!-- Brand -->
        <a class="navbar-brand" href="{{ url_for('about') }}">TripWhiz</a>
        <!-- Toggler/collapsible Button -->
        <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#collapsibleNavbar">
```

```
            <span class="navbar-toggler-icon"></span>
          </button>
          <!-- Navbar links -->
          <div class="collapse navbar-collapse" id="collapsibleNavbar">
            <ul class="navbar-nav">
              <li class="nav-item">
                <a class="nav-link" href="{{ url_for('recomendation') }}">Flights</a>
              </li><li class="nav-item">
                <a class="nav-link" href="{{ url_for('recomendation_hotels') }}">Hotels</a>
              </li>
            </ul>
          </div>
        </nav>
        <div class="container">
      <h1>Travel Itinerary</h1>
      {{ itinerary | safe }}

   </div>
</body>
</html>

templates/hotels_reco.html
<!DOCTYPE html>
<html lang="en">
<head>
   <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
</head>
<body>
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
          <!-- Brand -->
          <a class="navbar-brand" href="{{ url_for('about') }}">TripWhiz</a>
          <!-- Toggler/collapsible Button -->
          <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#collapsibleNavbar">
            <span class="navbar-toggler-icon"></span>
          </button>
          <!-- Navbar links -->
          <div class="collapse navbar-collapse" id="collapsibleNavbar">
            <ul class="navbar-nav">
              <li class="nav-item">
                <a class="nav-link" href="{{ url_for('index') }}">Generate</a>
              </li>
        </li>
        <li class="nav-item">
```

```
              <a class="nav-link" href="{{ url_for('trip_itinerary') }}">Itinerary</a>
            </li>
          <li class="nav-item">
              <a class="nav-link" href="{{ url_for('recomendation') }}">Flights</a>
            </li><li class="nav-item">
              <a class="nav-link" href="{{ url_for('recomendation_hotels') }}">Hotels</a>
            </li>
           </ul>
          </div>
         </nav>
   <div class="container">

   <h2> Hotels Recommendations</h2>
      {% for card in cards %}
      <div class="row">
         <div class="col-md-6 offset-md-3">
            {{ card | safe }}
         </div>
      </div>
      {% endfor %}
   </div>

</body>
</html>
```

templates/reccomendation.html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
</head>
<body>
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
          <!-- Brand -->
          <a class="navbar-brand" href="{{ url_for('about') }}">TripWhiz</a>
          <!-- Toggler/collapsible Button -->
          <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#collapsibleNavbar">
            <span class="navbar-toggler-icon"></span>
          </button>
          <!-- Navbar links -->
          <div class="collapse navbar-collapse" id="collapsibleNavbar">
            <ul class="navbar-nav">
```

```
            <li class="nav-item">
              <a class="nav-link" href="{{ url_for('index') }}">Generate</a>
            </li>
          <li class="nav-item">
              <a class="nav-link" href="{{ url_for('trip_itinerary') }}">Itinerary</a>
            </li>
          </li>
          <li class="nav-item">
              <a class="nav-link" href="{{ url_for('recomendation') }}">Flights</a>
            </li>
          <li class="nav-item">
              <a class="nav-link" href="{{ url_for('recomendation_hotels') }}">Hotels</a>
            </li>
           </ul>
         </div>
        </nav>
   <div class="container">

   <h2> Flight Recommendations</h2>
      {% for card in cards %}
      <div class="row">
        <div class="col-md-6 offset-md-3">
          {{ card | safe }}
        </div>
      </div>
      {% endfor %}
   </div>

</body>
</html>
```

templates/trip.html

```
<!DOCTYPE html>
<html>
<head>
        <title>Travel Itinerary</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">
        <style>
                body {
                        background-color: #031908;
                }

                .container {
```

```
                margin-top: 50px;
                margin-bottom: 50px;
        color: #fdfdfd;
        }

        .card {
        background-color: #538d7d;
                margin-top: 20px;
                border: none;
                box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
        }

        h1 {
                margin-bottom: 30px;
                font-weight: bold;
                color: #6c757d;
        }

        .itinerary {
                font-size: 20px;
                line-height: 1.5;
        }

        .itinerary h2 {
                font-size: 24px;
                margin-top: 40px;
                margin-bottom: 20px;
                font-weight: bold;
                color: #6c757d;
        }

        .itinerary h3 {
                font-size: 20px;
                margin-top: 30px;
                margin-bottom: 10px;
                font-weight: bold;
                color: #6c757d;
        }

        .itinerary p {
                margin-bottom: 10px;
        }
    </style>
</head>
<body>
<nav class="navbar navbar-expand-md bg-dark navbar-dark">
```

```
            <!-- Brand -->
            <a class="navbar-brand" href="{{ url_for('about') }}">TripWhiz</a>
            <!-- Toggler/collapsible Button -->
            <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#collapsibleNavbar">
              <span class="navbar-toggler-icon"></span>
            </button>
            <!-- Navbar links -->
            <div class="collapse navbar-collapse" id="collapsibleNavbar">
              <ul class="navbar-nav">
                <li class="nav-item">
                  <a class="nav-link" href="{{ url_for('recomendation') }}">Flights</a>
                </li><li class="nav-item">
                  <a class="nav-link" href="{{ url_for('recomendation_hotels') }}">Hotels</a>
                </li>
              </ul>
            </div>
          </nav>
          <div class="container">
        <h1>Travel Itinerary</h1>
        {{ itinerary | safe }}

    </div>
</body>
</html>
```