**Final Project | The Hitchiker's Guide to Puerto Rico (DSML)**

**Welcome**

Welcome to your final project. This is the culmination of your learning journey during this bootcamp. Congratulations on making it until here!

In this project, you are going to use all the Data Science and Machine Learning skills you have acquired during the course of the last few weeks to build an interactive travel planner for the beautiful island of Puerto Rico. By the end of this project, you will present a working application that cooperates with a visitor to help them build a travel itinerary suitable to their personal preferences.

There is a minimal set of requirements — the Minimum Viable Product (MVP) — that you will need to implement to complete the project successfully, but the way your project will distinguish itself is by whatever features you add to this minimal skeleton. Over the course of the project statement, you will find several hints and suggestions of improvement over the MVP in collapsible sections like this:

Guide: A collapsible section

Remember that the contents of the collapsible sections are *just suggestions*; your own ideas for improvement of the guide make for a much more unique and memorable project and should be prioritized whenever feasible.

**Interface and minimal functionality**

The guide should be served through an application where the main interface with the user is presented as a chatbot that interacts with a prospective traveler and suggests visiting spots in the Island of Puerto Rico based on the interests of the user. It should be able to provide information on several Puerto Rico landmarks, answer questions about events, and help the user compile a list of points of interest to visit. The chatbot should always maintain a friendly and informative tone while remaining focused on helping the user compile a list of points of interest to visit. The end result of a successful interaction with an interested user should be a list of landmarks of interest for the user. The end result of a successful interaction with a non-interested user should be the termination of the conversation in a graceful mode.

Additional context

For additional implemented functionality above the MVP, the above parameters may be relaxed. For example:

1. you might want to consider an additional interface with a map of the island to showcase locations of landmarks, or where the user can select areas of interest. In a case like this, maybe your input requires more structure than simple text for a chatbot.
2. you might want to return the result as an itinerary segmented by days of travel, or printed on a map. In a case like this, maybe your output requires more structure than a simple list.

**Data**

You will find in the \data folder the following repositories of raw data:

1. \data\landmarks: contains raw .txt files for several Puerto Rico landmarks (extracted from [Wikipedia](https://en.wikipedia.org/wiki/List_of_Puerto_Rico_landmarks))
2. \data\municipalities: contains raw .txt files for several Puerto Rico municipalities (extracted from [Wikipedia](https://en.wikipedia.org/wiki/Municipalities_of_Puerto_Rico))
3. \data\news: contains raw .txt fies for several years worth of news from the Puerto Rican newspaper [El Mundo](https://www.elmundo.pr/) (extracted from [The Center for Research Libraries](https://gpa.eastview.com/crl/elmundo/?a=p&p=home&e=-------en-25--1--img-txIN----------))

Additional context

Consider developing a NER (Name Entity Recognition) engine to identify potential Entities (landmarks, municipalities, restaurants, etc.) mentioned in the news articles but present in our data Consider developing a Sentiment Analysis engine to enrich the metadata of the news articles as Positive or Negative about the landmark the article mentions. This can then be offered to the user as an optional filter.

**Extraction and summarization**

You will need to create a repository of structured information for the several Puerto Rico landmarks and municipalities. You can use an LLM or data scraping and HTML parsing techniques to retrieve, for each of the above *at least* the following information:

1. Location coordinates
2. A small summary of relevant information about the location

Store this information in an appropriately chosen format. Consider using a vector database for storing semantic information on the summaries. Keep in mind that you will be using the summary to match user preferences to potentially interesting locations, so prompt your summary extractions appropriately for this task.

Additional context

You are not limited to these datasources, nor to these datapoints. Maybe you want to additionally scrape information about museums, or stores, or restaurants. Maybe you want to add structured information like opening hours or access price, if available.

Consider resources like Zoomato, TripAdvisor, Booking, Skyscanner, etc. You can either scrape static information from their APIs or enrich the capabilities of your system later when we're discussing function calling.

**RAG system for perusing news articles**

The historic articles for El Mundo contain a wealth of information that may help your user narrow down sites of interest to visit based on their historical significance.

Devise an appropriate chunking strategy and create an appropriate RAG system that retrieves relevant documents to answer user queries and relate past events to user interests.

Notice that the documents are in Spanish, which means that you will have to:

* translate the documents
* use a multilingual system for your embeddings and retrieval (you can use a pre-trained model).

Additional context

You may also add structured information to each document, perhaps aligning each document with relevant landmarks and locations.

You may enrich this repository with information from other newspapers that you can find in the [Library of Congress](https://www.loc.gov/collections/chronicling-america/dynamic-list-of-titles/?searchType=advanced&st=table&sb=title_s_asc&location_state=puerto+rico). These datasets are image-based, so you will need to make use of a multi-modal model with vision capabilities.

Finally, you can of course train your own embedding model for this end.

**Functions for agentic utilization**

Create at least the following functions that your agent should be able to call:

* find\_weather\_forecast(date,location): function calls the [OpenWeather API](https://openweathermap.org/api) and retrieves the weather forecast location in date.
* rank\_appropriate\_locations(user\_prompt): function evaluates user-fed prompt and goes through RAG system to find appropriate locations to suggest as visitation destinations to the user. For example, if the user mentions liking the sun, beaches should be highly ranked. If they mention liking history, museums and locations with rich historical events should be highly ranked, etc.
* find\_info\_on\_location(user\_prompt, location): function evaluates request for information, finds relevant documents regarding location and returns information about location relevant to the user query.
* add\_location\_to\_visit\_list(list, location): function receives listof locations already selected to visit and adds location to that list.
* compute\_distance\_to\_list(list, new\_location): function that takes the locations already selected to visit and returns the distance of location to the closest location already in list.

Additional context

Here is where you can add most of the customized functionality to improve your system. Some ideas:

* you can allow your agent to plan the shortest circuit between locations to visit using a [scheduling API](https://docs.routific.com/reference/getting-started).
* you can allow your agent to suggest activities, restaurants, hotels, etc. that are close to the locations in the list of selected landmarks using one of the many [travel APIs](https://tripadvisor-content-api.readme.io/reference/overview).
* you can allow your agent to add a point to a map with the chosen locations so the user can visually understand and plan their trip.

-...

**Capabilities of the agentic system**

The system on your backend should be able to support the following interaction loop:

1. Ask the user for travel dates.
2. Ask the user for interests.
3. Suggest locations based on interests.
4. Answer questions from users regarding locations and events. Ask if the user wants to lock visiting specific locations.
5. Decide if location enjoyment is weather dependent. If so, check weather at location in travel date and issue a warning to reconfirm lock of location.
6. Add location to list of locked locations.
7. Ask if the user is done and either move to 3. or to 8.
8. Return the list of locked locations to the user.

Additional context

You may want to separate the system into an orchestrator that decides which functions to run next and a communicator just responsible for conveying information to the user.

If you have multiple disparate or competing functionalities (like balancing prices with itineraries, with days, with weather, etc.), you may want to consider using [a crew](https://www.crewai.com/open-source) to reach consensus.

**Fine-tuning the response agent**

You should select a tone of voice for your chatbot and fine-tune it to its intended function.

1. Utilize a powerful LLM, such as the latest models from Anthropic, OpenAI, Google, or Meta, to generate sample interactions that follow your preferred tone of voice. You can include few-shot examples during this generation. Also, make sure to include examples of interactions where the bot refuses to respond or re-steers the conversation back to its intended loop.
2. Use the sample interactions generated in 1. to tune a smaller LLM for these specific tasks. You can use full fine tuning, LoRA, or any other technique to make your weaker model more specialized.

**Evaluation process**

You should evaluate your system - in the end, you are a Data **Scientist**. Start in 1., and try to evaluate your system in every way possible, in an ascending order:

1. Evaluate your system, yourself, qualitatively - test your system thoroughly and frequently.
2. Create pairs of Questions-Answers that you consider the ground truth. A way to do this can be leveraging on an LLM application (ChatGPT, Claude, etc) to create 30 to 50 correct answers. After, consider using a metrics like [ROUGE](https://en.wikipedia.org/wiki/ROUGE_(metric)) to identify if your generated answer matches the correct answer.
3. *LLM-as-a-Judge*: the concept of using *LLM-as-a-Judge* is to have **another** LLM evaluating if the generated answer was correct, based on:
   * The user's question/input
   * The generated answer
   * The available information (documents, websources, etc)

This *LLM-as-a-Judge* should evaluate each answer with a quantitative score.

**Deployment and Logging**

**Deployment**

Your application should have an acceptable user interface that allows users to interact with the chatbot effectively. Consider using frameworks like Streamlit or Flask to provide a user-friendly starting point.

Additional context

**Logging**

You are required to implement a logging system within your application. This system should capture and store user interactions, such as input queries, timestamps, and any other relevant metadata. The logs should be structured in a way that allows for analysis and visualization.

Additionally, you should create a Tableau dashboard (or use another data visualization tool) to analyze the logs. The dashboard should provide the insights you see fit, but here are some suggestions:

* Frequently used words or phrases by users
* Common user intents or topics of interest
* Usage patterns over time
* Any other metrics that could help improve the application

You might need to use your tool quite a bit to have data to showcase as logs (~25 times).

**Final Remarks**

While deploying your application to a remote server is optional, doing so can enhance the accessibility and robustness of your project.

Enhancing the user interface to be more user-friendly can greatly improve the user experience and is a valuable addition to your project.

The logging and dashboard components will help you understand user behavior and are essential for iterating and improving your application over time.

**Project Plan**

**1. Project Overview and Objectives**

* **Project Title:** The Hitchhiker's Guide to Puerto Rico
* **Objective:** Develop a chatbot-based travel planner that interacts with users to suggest Puerto Rican landmarks, events, and itineraries based on personal interests, travel dates, and contextual factors (like weather).
* **Key Deliverables:**
  + An interactive chatbot with a friendly, informative tone.
  + A structured database of Puerto Rico landmarks, municipalities, and historical news.
  + A Retrieval-Augmented Generation (RAG) system to provide context-rich responses.
  + A suite of functions (weather lookup, ranking, distance computations, etc.) to support the planning process.
  + A logging system coupled with a dashboard (e.g., Tableau) to visualize user interactions.

**2. Data Acquisition and Processing**

**a. Data Sources**

* **Landmarks and Municipalities:**
  + Raw text files from /data/landmarks and /data/municipalities (extracted from Wikipedia).
* **News Articles:**
  + Historical articles from /data/news (extracted from El Mundo via The Center for Research Libraries).

**b. Extraction and Summarization**

* **Tasks:**
  + **Extract location coordinates** and key descriptive information from each text file.
  + Use an LLM or text scraping and HTML parsing techniques to generate concise summaries tailored to assist in matching user preferences.
  + Store each landmark’s summary and its metadata (coordinates, historical context, etc.) in a structured format.
* **Storage:**
  + Consider using a vector database for semantic searches, ensuring that the summaries can be matched to user interests effectively.

**c. Processing News Articles**

* **Translation and Chunking:**
  + Translate Spanish documents to English if needed.
  + Devise a chunking strategy to segment long articles into manageable pieces.
* **RAG System:**
  + Use pre-trained multilingual embedding models to index news chunks.
  + Retrieve relevant news based on user queries to provide historical context.

**3. System Architecture and Key Components**

**a. Chatbot Interface**

* **Functionality:**
  + Conduct a guided conversation with the user, beginning with travel dates and interests.
  + Suggest landmarks based on user preferences.
  + Answer questions regarding locations and events.
  + Manage conversation termination gracefully for non-interested users.
* **Design Considerations:**
  + Maintain a consistent, friendly tone.
  + Include fallback mechanisms where the chatbot can steer the conversation back on track.

**b. Core Functional Modules**

* **Weather Forecast Integration:**
  + find\_weather\_forecast(date, location): Calls the OpenWeather API to retrieve weather forecasts.
* **Location Ranking and Suggestion:**
  + rank\_appropriate\_locations(user\_prompt): Uses the RAG system to rank destinations based on user interests (e.g., beaches for sun lovers, museums for history buffs).
* **Information Retrieval:**
  + find\_info\_on\_location(user\_prompt, location): Retrieves detailed information on a location from the structured data or news documents.
* **Itinerary Management:**
  + add\_location\_to\_visit\_list(list, location): Adds a selected landmark to the travel plan.
  + compute\_distance\_to\_list(list, new\_location): Calculates the distance between a new location and the closest already selected landmark.

**c. Logging and Dashboard**

* **Logging System:**
  + Record user interactions (input queries, timestamps, chosen interests, locations, etc.).
  + Store logs in a structured format suitable for subsequent analysis.
* **Dashboard:**
  + Use Tableau or a similar visualization tool to analyze trends (frequently used words, common user intents, usage patterns over time).

**d. Deployment Considerations**

* **Backend:**
  + Python-based environment integrating APIs (e.g., OpenWeather) and custom functions.
* **Frontend:**
  + Deploy using web frameworks such as Flask or Streamlit for an accessible, user-friendly interface.
* **Optional Remote Deployment:**
  + Enhance accessibility and robustness by hosting on a remote server.

**4. Implementation Roadmap and Timeline**

**Phase 1: Setup and Data Ingestion (Week 1)**

* **Set Up Environment:**
  + Configure Python environment, install necessary libraries.
* **Data Extraction:**
  + Parse raw .txt files for landmarks, municipalities, and news.
* **Initial Summarization:**
  + Develop scripts to extract and generate summaries from raw data.

**Phase 2: Data Structuring and RAG System (Week 2)**

* **Database Creation:**
  + Organize extracted data into a structured format (e.g., vector database for semantic searches).
* **News Processing:**
  + Implement translation and chunking of news articles.
* **RAG System:**
  + Integrate multilingual embeddings and retrieval techniques to answer location-specific queries.

**Phase 3: Chatbot and Function Development (Week 3)**

* **Chatbot Interface:**
  + Design and implement the interactive conversation flow.
* **Function Integration:**
  + Develop and test core functions: weather forecast, ranking locations, retrieving detailed info, itinerary management.
* **Sample Interactions:**
  + Generate few-shot examples to fine-tune the chatbot’s conversational tone and edge-case handling.

**Phase 4: Logging, Evaluation, and Dashboard (Week 4)**

* **Logging System:**
  + Integrate logging of user interactions and build data storage.
* **Evaluation Framework:**
  + Create ground truth question-answer pairs and use metrics (e.g., ROUGE) along with an LLM-as-a-judge to assess responses.
* **Dashboard Development:**
  + Develop a Tableau (or similar) dashboard to visualize log data and usage patterns.

**Phase 5: Integration, Testing, and Deployment (Final Week)**

* **Integration:**
  + Combine all modules into a cohesive application.
* **Testing:**
  + Perform unit, integration, and user testing.
* **Deployment:**
  + Deploy the application locally or on a remote server.
* **Documentation and Final Presentation:**
  + Prepare documentation and finalize the presentation of the project.

**5. Risks and Mitigations**

* **Data Quality and Translation Issues:**
  + Mitigation: Use robust pre-trained models for text extraction and translation; manually verify summaries for critical landmarks.
* **API Limitations:**
  + Mitigation: Implement caching and fallback options for weather API calls.
* **Integration Complexity:**
  + Mitigation: Modularize components and conduct thorough testing at each integration step.
* **User Experience:**
  + Mitigation: Run iterative user tests to refine the chatbot’s dialogue and ensure a seamless interaction loop.

**6. Final Deliverables**

* **Interactive Chatbot Application:**
  + Fully functional travel planner that interacts with users to build personalized itineraries.
* **Structured Data Repositories:**
  + Indexed and searchable database of landmarks, municipalities, and news summaries.
* **RAG System:**
  + An effective retrieval mechanism for historical news and location-specific information.
* **Logging and Dashboard:**
  + A comprehensive system to log user interactions and visualize key metrics.
* **Documentation:**
  + Detailed technical and user documentation along with a final project presentation.