Reflection Paper

For our DS2002 Data Project 2, we created WeatherBot, a Flask-based chatbot that provides weather information based on user questions. This assistant uses two distinct data sources: real-time weather data from the WeatherAPI service and historical weather data for March 2016 stored locally in a cleaned CSV file. Our project involved designing and deploying a functional web-based assistant that can intelligently distinguish between user intents and respond using either the API or local dataset. We also hosted the chatbot on a publicly accessible Google Cloud Platform (GCP) virtual machine to meet the deployment requirement.

To make the chatbot engaging and informative, we decided to focus on weather as the topic. Weather is a universally relevant subject and provides a wide range of measurable variables such as temperature, humidity, wind speed, UV index, and precipitation, which are ideal for both real-time and historical comparisons. By providing data from 2016 with our CSV file and current weather data from the API, we can effectively compare weather today to how it was approximately 10 years ago.

For the live data, we selected WeatherAPI as it provided free API access with detailed, up-to-date weather metrics for a large number of global locations. It also had a relatively simple authentication process and user-friendly documentation, which made integration easier. For the local data source, we used a CSV dataset containing historical U.S. weather observations for March 2016 across different states. This dataset included important variables like temperature, wind direction, precipitation, and wind speed. We chose this file because it offered real-world, structured data that still required substantial cleaning, making it ideal for implementing a full ETL (Extract, Transform, Load) pipeline. We performed tasks such as renaming mismatched column headers, filtering data to include only March 2016 records, handling missing values, and ensuring consistency in state name formats. The resulting transformed dataset was then loaded into memory via Pandas and used as a reliable source for answering historical queries.

One of the main technical challenges we encountered was determining how to parse queries to accurately route them to either the API or the local dataset. To address this, we implemented simple keyword-based pattern matching using Python’s re module. For instance, if a user asked about “temperature in Texas in March 2016,” the bot would extract both the location and time period, determine that it related to historical data, and then query the local dataset accordingly. If the query instead asked for “current temperature in Texas,” it would route to the live API.

Building the ETL pipeline also presented several complications. The raw dataset had inconsistent column naming, such as “Data.Temperature.Avg Temp\_dataset,” which we had to clean and rename. We also encountered some rows with missing or malformed entries, which required additional filtering and validation to ensure the dataset would not cause runtime errors during chatbot operation.

Deployment on GCP was another key hurdle. Initially, we ran into issues with firewall settings and virtual machine configurations. Because we were using a CSV file and an API, we had to make sure all the appropriate files were uploaded onto GCP before attempting to have the chat run on the web. After reading documentation and troubleshooting port permissions, we were able to successfully launch our Flask app and make it publicly accessible. Ensuring the /chat route responded properly via curl and Postman helped us validate our deployment.

We also placed a strong emphasis on error handling. For example, if the CSV file failed to load, our app would display a warning and gracefully default to an empty dataset. Similarly, any failure to reach the WeatherAPI—due to a bad API key or network issue—would return an informative JSON error message to the user.

This project helped us grow significantly as developers by becoming more comfortable with Flask and how to design a clean website that can complete a task. We learned how to use Python’s requests library to communicate with external APIs and how to process those responses for relevant information. The ETL portion reinforced our understanding of how real-world data often needs extensive preprocessing. Despite using a dataset with only weather for 50 states, we saw plenty of data cleaning that needed to be performed before being able to utilize this dataset in our chatbot. We gained experience using pandas for data manipulation and learned how to structure our data pipeline so that the final file would be clean, efficient, and directly usable by the chatbot.

Given more time and resources, we would enhance WeatherBot by implementing several new features. First, we would introduce multi-day forecasts and weather trends, allowing users to compare historical and projected weather for any state. We would also like to expand support to include international cities in both historical and current queries, provided we can access global datasets. Another key improvement would be integrating a simple natural language processing model to better parse user input. This could help resolve confusion, such as when users refer to cities and states with overlapping names, or use phrasing not covered by our regex patterns. Additionally, we would integrate Gemini or another large language model to provide smarter, more conversational responses.

Overall, this project pushed us to combine theoretical knowledge with practical tools. It required creativity, debugging resilience, and attention to both backend logic and user experience. From integrating APIs and building a custom ETL pipeline, to deploying the app on a cloud platform, this experience provided a strong learning opportunity.