Colin Henry | fgm2wg
Professor Williamson
DS 2002: Data Science Systems
7 May 2025

Data Project 2 Reflection Paper

For this project I chose the Open-Meteo API (ERA5 archive for historical data and 16-day forecast) as my live public source, and built a local CSV of daily max/min temperatures back to January 1, 2022. Open-Meteo's free, well-documented endpoints made extraction straightforward, and its built-in support for Fahrenheit units eliminated extra conversion steps. I focused on Charlottesville (latitude 38.0302, longitude -78.4769) so that the bot logic could remain location-fixed and I could concentrate on the ETL and date-parsing challenges. I chose this as my source because I knew that it was easy to extract data in a standard form and additionally the API does not require any key to use. It also could provide historical data, which was needed to create the static file, as well as a live-data forecasting API for future dates. It has a non-commercial limit of 10,000 daily calls as well which was completely sufficient for me, even when repeatedly calling it to test various aspects of my code. Compared with other weather APIs like OpenWeatherMap and NOAA's NWS API, it is much easier to use and has great customizability to get just the data you need without any annoying rate limits or signing up for an API key.

The most challenging part was interpreting user friendly date phrases. I had to distinguish "last Thursday" (the previous week) from "next Thursday" (the week after this one) and also handle bare weekdays in both past (when the user said "was Sunday") and future contexts. I kept finding lots of edge cases that wouldn't work with my current regex matching functions and then I would have to change them. However, everytime I changed the logic, it seemed to break something else that worked before and it took forever until I finally got all possible cases to work as expected. Supporting both MM/DD/YYYY and YYYY-MM-DD uniformly, without confusing US vs. ISO date formats, also took several iterations of regex and dateutil logic to get right. It was also challenging to try and make a nice looking site that had an aesthetic feel. I had to play around with the CSS and HTML code a bunch and look at other sites as references until I got a result that I was happy with. This style tuning was very time consuming and took many iterations of changing values to get the final product.

A key learning that I gained from this project is how important incremental ETL can be to large datasets. In my code, I added logic to check if the historical data csv was out of date and it would refresh it with new data by calling the historical API and making a new csv. However, I learned that I could instead just update the rows that would be missing data, rather than remake the entire csv. This meant that I could save a lot more time and space by just appending a few rows and not getting thousands of rows that I already had. Another discovery I made is that to save a session with flask you need a secret key, but you need to store this in a secure .env file so that others cannot access it. From this, I learned how to add a SECRET_KEY variable in a .env file and then get the value in my app.py code to securely retrieve it without exposing it to potential attackers. Finally, I learned about how to make my own Linux service to run a program without having a terminal open. I was able to create a service and give it my project directory and Python interpreter so that it could asynchronously run the program and I wouldn't have to worry about keeping my terminal open for the website to stay up.

With more time, I definitely would have pursued the extra credit opportunity to integrate Gemini AI for better responses as well as the Discord bot for another way to access the chat bot. Gemini AI would have been much better at handling the confusion around the differences in last weekday, this weekday, and next weekday. It could use its highly trained model to interpret these inputs and ultimately give a better response that is more tailored to what the user actually wants. Also, my current implementation only uses max and min temperature for each day for simplicity. However, it would be beneficial to expand this to work for hourly data as well and to include many other factors such as rainfall, humidity, UV index, etc. This would require handling more of the user input, however, like checking if they just want temperature or something like if it's going to rain. I only chose to store the data in a csv file, which worked fine, but it would probably be better to update this to some form of SQL database for more powerful search commands and ease of adding/removing new columns of data to the database. Finally, the last enhancement that would be good to add is to provide the ability to find weather in other locations than just Charlottesville. This would not work with the current csv, but likely porting it to a SQL database and restructuring it would allow this to work for different locations and ultimately being a lot more useful.