Jessica Xiong (pqf6rd)

Lingyue Ji (gcs6zm)

Project Reflection: Recipe Chatbot with ETL and API Integration

For our final data science project, we created a recipe recommendation chatbot that runs on Discord and is powered by a Flask backend. The system allows users to search for recipes based on simple text commands like !recipe chicken, and it responds using either a cleaned local dataset or the Spoonacular API if no local match is found.

From the start, we wanted to create something fun and relatable, and food was a perfect choice. Instead of doing a serious or dry dataset, we thought a recipe chatbot would let us practice technical skills in a creative way. Everyone loves food, and we liked the idea of a system that could help people figure out what to cook based on ingredients or cravings.

The first step was downloading and inspecting the 13k-recipes.csv dataset. Our group wrote a Python script, etl_recipes.py, to clean the data: this involved removing nulls, extracting ingredients into a structured list, and saving it all into cleaned_recipes.csv. This part of the project helped us integrate the data with fewer errors. The raw dataset had a lot of inconsistencies, so we had to apply multiple cleaning steps before it could be used effectively in a chatbot.

To do the Flask part, we built a /chat route that takes in a user message and searches for matching recipes in the cleaned dataset. If nothing is found locally, the chatbot uses the Spoonacular API to return a recipe. Writing the logic to check both sources gave us a better understanding of fallback systems and error handling.

Adding the homepage route (/) was a nice touch because it made the project more accessible to users who just click the URL. We learned that Flask doesn't automatically serve a landing page, so customizing that gave us better insight into routing.

We deployed the final chatbot to Google Cloud using App Engine. Using GCP let us feel rewarding to access app from a real live URL. We also learned a lot about app.yaml configuration, runtime environments, and how requirements.txt is used in production.

One of our biggest challenges was getting the API integration to work smoothly. There were moments when Spoonacular returned no results, or we hit a key error from a missing field. We had to make the code more defensive and improve error checking. Another challenge was debugging deployment errors on GCP—especially around permissions. But now, we feel much more confident using both Flask and Google Cloud.

For the discord bot, we faced several technical challenges as well, such as getting the Discord bot to communicate with the Flask API correctly. We made a mistake that was sending POST requests to the root route (/) instead of /chat, resulting in 405 errors that required careful debugging of URLs and HTTP methods. We also ran into Discord's 2000-character message limit, which caused recipe responses to fail when too long—this was resolved by splitting lengthy messages into smaller chunks. Another persistent issue was improper rendering of newline characters (\n vs. \\n), which we traced using repr() and fixed by adjusting both the Flask API's response and the bot's formatting logic.

If we had more time, we would have added interactive features like step-by-step recipe walkthroughs using "next step" commands, improved the bot's visual appeal with Discord embeds or filtering based on dietary preferences. On the backend, we aimed to replace eval() with json.loads() in the Flask API for safer data parsing, add unit tests to ensure core functionality remains stable through future updates, and integrate natural language processing (NLP) to allow more flexible commands like “What can I make with tofu and rice?”.

This was our most comprehensive and rewarding project of the semester. It brought together everything we’ve learned—from data wrangling to APIs to cloud deployment—and turned it into a real, working product. Between ETL, routing, APIs, and deployment, we had to touch almost every stage of the data science workflow. Seeing our chatbot respond live on Discord, backed by both a structured dataset and real-time API calls, made the challenge worth it.