Reflection Report – DS2002 Project 2

This project was a valuable experience in combining static datasets with real-time data to build an interactive chatbot using Flask. We worked with a cleaned NBA player dataset and integrated live information from the BallDon'tLie API, which gave us a better understanding of building lightweight APIs and managing data pipelines. Like our first project, there were a mix of tasks that went smoothly and others that were more complicated than we expected.

One of the main challenges was setting up and managing the interaction between our local data and the API. We had to make sure the chatbot could handle both static responses (like player weight or college counts) and live API calls (like jersey number or draft info). What made it tricky was designing the logic that could correctly detect what the user was asking for and then choosing which data source to pull from. There were also a few issues with API errors or inconsistent name formatting between sources, which required some debugging and logic adjustments. We learned that even small things—like spacing in player names or typos in college listings—could break an otherwise working function.

Deploying the app to Google Cloud also added a new layer of complexity we hadn't dealt with in earlier assignments. Getting the app to run locally was fine, but figuring out how to containerize it with Docker and configure App Engine to handle environment variables (like the API key) took a lot of trial and error. We had to go through documentation, troubleshoot port errors, and restructure the repo a few times to match the platform's expectations. These were all useful skills to practice, but definitely harder than we originally expected.

On the flip side, some parts of the project were surprisingly smooth. Flask made it easy to route API requests and return JSON responses, and pandas was once again really helpful for doing quick calculations from our cleaned CSV. Writing the ETL script was relatively easy too since the original dataset was already in decent shape. It was mostly about standardizing formats like height and date of birth and filtering out rows with missing info. Even the basic chatbot structure came together quickly once we decided on a simple interface using `curl` to simulate user questions.

Another interesting part of the process was testing and debugging our chatbot responses. We had to think about different ways users might phrase questions and make sure the chatbot could still respond sensibly. We focused on getting at least three core features working reliably: querying the heaviest player, identifying the college with the most players, and pulling live profile info. Once those were solid, we made sure the fallback responses were helpful in guiding users on what to ask.

This project helped us realize how powerful even a lightweight chatbot can be when connected to multiple data sources. In future data science workflows, tools like this could help non-technical users explore datasets without needing to write queries or use dashboards. You could easily adapt the same structure to support questions about financial data, product catalogs, or academic datasets, just by swapping out the data and adjusting the intent logic. We also saw how important it is to plan ahead when dealing with real-time data—especially when APIs aren't perfectly reliable or complete.

Overall, this project pulled together everything we've been learning: data cleaning, API integration, web development, and cloud deployment. It pushed us to solve new problems but still gave us a solid sense of accomplishment at each stage. Working on a topic like NBA data also made it more fun and engaging as a group. While there were definitely moments of frustration (especially around deployment), it was rewarding to see the chatbot working end-to-end and handling real user queries with real-time data.