Data Project 2 Reflection Paper

For this project, our team decided to build a functional data pipeline that could serve real-time NBA player information through a Discord bot. Our goal was to create a user-friendly and interactive tool that combined data engineering concepts like ETL with practical deployment in a cloud environment. This not only tested our ability to gather and clean data, but also our capacity to integrate and deliver it in a way that is accessible to non-technical users. Our GitHub repository, dataproject2 , includes documentation on implementation details and instructions for users to access and interact with the bot on Discord.

Dataset Selection

We approached dataset selection with a clear end goal in mind: efficient delivery of accurate NBA player information. To meet this need, we explored several public APIs before settling on the BallDontLie API, which offered structured and frequently updated NBA stats. To complement the API, we also used a CSV dataset sourced from Kaggle that contained a broad range of historical player data up to 2024. This combination allowed us to create a hybrid data cache, pulling from both static and dynamic sources, to give users a well-rounded and current view of a player's performance and profile.

When choosing data sources, our main priorities were accessibility, completeness, and update frequency. We also needed something that integrated well with our ETL design and wouldn't create unnecessary complexity. The BallDontLie API stood out for being intuitive and well-documented, while the CSV gave us a fallback layer of data in case the API was unavailable or slow.

Challenges and Obstacles

One of the biggest technical hurdles was improving the efficiency of our ETL pipeline. In the early stages, our pipeline ran in about eight minutes, which was way too long for a system designed to provide responses in real or near-real time. The issue stemmed from inefficient API calls that paused unnecessarily due to a sleep timer being triggered after every set of calls, even when it wasn't needed. We had to rethink our approach, optimize our transformations, and streamline data loading to reduce this processing time.

Another major challenge was making our system accessible to users outside our local machines. Deploying the ETL process and Discord bot on a Google Cloud Platform (GCP) virtual machine required us to deal with virtual environments, API permissions, and continuous deployment. Although this was frustrating at times, it gave us hands-on

experience with real-world issues like uptime reliability, credential management, and debugging cloud-based services.

Key Learnings and Discoveries

One of the most important things we learned was how to make an ETL pipeline publicly accessible through a cloud platform. Before this, most of our ETL experience was in local or sandbox environments. Hosting the bot and pipeline on GCP gave us a much better understanding of how cloud infrastructure supports automation and scalability. We also learned how to troubleshoot and adapt when integrating multiple tools and services, specifically, how to bridge the gap between a bot platform like Discord and backend data processes running in the cloud.

This project also helped us think more critically about performance optimization. In class projects, performance isn't always a concern, but in this case, it became central to the user experience. Making the system faster, leaner, and more responsive meant understanding where bottlenecks existed and how to remove them with minimal trade-offs.

Potential Enhancements

If we had more time, we would definitely improve the Discord bot's user experience, particularly its ability to understand a wider range of user input. Right now, it only responds when the player's name is typed at the beginning of the message, which isn't very flexible. A smarter approach would involve implementing basic natural language processing so that the bot could detect player names regardless of where they appear in the sentence.

We also discussed refining the data refresh logic. Currently, the system fetches data frequently, even if nothing has changed. Implementing smarter refresh intervals or caching strategies would make it more efficient and reduce unnecessary API calls. Another cool idea would be adding head-to-head player comparisons directly within Discord, or even pulling historical trends to show performance over time in a quick, visual format.

Final Thoughts

Overall, this project felt like a solid step forward in applying data pipeline concepts to a real-world scenario. It pushed us to think beyond just writing scripts, as we had to consider system design, performance, and the end-user experience. Building something interactive and publicly accessible made the work feel meaningful and gave us a better sense of how data engineering supports applications that people actually use.