*r/nba Game Threads: An ETL Project*

*Overview*

Using Python, this ETL project pulled game data from each NBA game from the last regular season (2018-19) and its corresponding Game Thread on r/nba. The datasets were then combined and stored in a non-relational database on mongoDB Atlas. I faced a significant hurdle when parsing subreddit searches/requests due to the shifting nature of APIs and the dedication (or lack thereof) of Reddit itself to advanced searching.

*Part I: NBA-API*

The first step for this project was to pull a Box Score for each regular season NBA game. This would contain all the relevant data points—date, teams, matchup, winner/loser, game id, and other basketball metrics. Using this data, I would then parse r/nba for each corresponding Game Thread and pull data from it for storage and analysis.

The official API is simple enough to use via a simple *requests.get(url)* method; however, there does exist a Python wrapper, *NBA\_API*. This wrapper proved highly useful for pulling the entire *game log* for the season. The package contains a built-in *get\_df* function, which enabled me to construct a DataFrame of the entire season. However, the game log contained double the amount of games I needed. With 30 teams playing 82 games, there would be 2460 games played in the regular season. The problem is that each game, counted in that fashion, would be counted twice—once from each team’s perspective. To use the NBA DataFrame efficiently, I needed to combine the “duplicate” rows. Luckily, there is a function provided for that purpose in the NBA\_API documentation. Using that function, I was able to condense the DataFrame to 1230 rows, with each row containing all the requisite data for both teams.

*Part II: PRAW (Reddit API)*

This portion of the project proved this most difficult and involved *much* trial and error. As the *game log* pulled from the NBA API contained a date field, it seemed fairly obvious that it would be critical to the search process. I misguidedly began by looking to pull the entire subreddit and search for the individual threads in that bucket, which quickly led to the realization that Reddit limits that pull method by 1000 results or less. I needed to cast a deeper net for each individual thread.

The ultimate solution to this problem—and an imperfect one as I discovered—was to user the itertuples function on the DataFrame and construct a search based on each tuple from each row. This was largely successful, but due to the way Reddit constricts searches, did not work for all games. Reddit does *not* enable its users to limit searches based on a date range—even its regular web search. Based on some research I did at the beginning of the project, I originally expected to coerce the dates from the DataFrame into UNIX in order to construct each query—and I created the code to do this. However, this functionality ended a couple years ago, so that method no longer functions.

My solution to this problem was to dive into the Game Threads themselves and investigate any particular moderators/posters or information that could help construct particular queries. As I discovered, nearly all Game Thread titles contained the same string, which I could approximate using the data I extracted from the NBA:

*“GAME THREAD: Home\_Team (W/L) @ Away\_Team (W/L) – (Month Day, Year)”*

I decided to reconstruct each title in order to programmatically query the subreddit. There are some interesting takeaways from that process. First, coercing and searching strings can become very tricky, especially when the format of the month could vary per search (ie. “*November”* vs *“Nov”*). The same is true for city names. The *Los Angeles Clippers* and *Lakers* were formatted in various ways, so I decided to drop the city from the team name when performing searches.

The second is that APIs and queries are ephemeral. As I ran tests of larger and larger portions of the dataset, I realized that approximately 10% was not found. I inspected examples and, other than date/team names being written in various formats, I discovered that if a thread was created by an account that has been deleted, that thread will no longer be searchable on reddit itself. Given more time, I would most likely use Google’s search API or a similar alternative, as I had no problem searching for the threads outside Reddit. As currently constructed, the query loop caught roughly 90% of the games and uploaded to the database, and uploading the remainder—even without Reddit data—would not be a problem, given the flexible nature of the mongoDB.

As a member of r/nba, I am well-aware of the toxicity that consistently takes place. So, I thought it would be an interesting portion of this project to perform some sort of rudimentary analysis of the top comment. Detecting expletives seemed like a good starting point. There do exist some Python modules that perform this function, though quite a few merely hard-code words into a text file and test for those words. “Profanity\_check” aims to use machine learning—it relies on Sci-Kit Learn—in order to predict expletives and detects them as well. For each top comment, I checked for profanity and appended that value (1 or 0). This brought some levity to the project, while also making it more complex and perhaps useful later on in the course.

*Part III: MongoDB Upload*

Because of the nested nature of the database I intended to create, I decided to move forward with a mongoDB. I initially began the project with an intention of creating mySQL or SQLite, but when I decided to pull comments/titles—longer and unstructured data—mongoDB seemed like a better choice. Moreover, as outlined above, the extraction portion of the project definitely exceeded expectations in terms of time and difficulty and non-relational database seemed a good solution for the project.

I created a remote database via mongoDB Atlas, whitelisted my IP address, and used pyMongo for the database connection and upload. Importantly, I used the *replace\_one(upsert:True)* function to ensure that any duplicate uploads would not create more entries in the database itself. Were I to revise the project, I could update the data present in the database with more comments, expletive analysis, etc.

My ultimate methodology for uploading my documents to the DB was to structure the upload directly into the search loop. This most likely added to the compute-time, but I was concerned with IP blocks, so some added time seemed like a fair compromise. I used the NBA DataFrame columns as a template for the document dictionary and nested the r/nba data and comment data within.

*Conclusion*

Overall, I consider this ETL project a success, both in terms of outcome and learning. I will definitely revisit this project and revise the code to extract all missing Game Threads. Another next step will be to perform more complex analysis on the comments themselves—that seems to be where the meat of this project lies—and combine that analysis with game/player performance. Does Kevin Durant *really* read r/nba at halftime? I hope to answer that question with more patches of this script.