Rays Data Project – Pitching Baseball Card

*Dalin Stone*

Design

Table

Description automatically generated with medium confidence

Explanation

I chose to design a card for pitchers, and I based it loosely off the #51 Mariners Randy Johnson pitchers’ card. This card is special in that it includes a length description of Johnson and his goliath career. From his stature to his stats, he proved to be an impressive pitcher and for that I reason I chose to stick to the basic statistics that are included on this card. My father-in-law’s collection of baseball cards have the authentic look and description, which influenced this cards design. As for statistics I elected to use mostly traditional stats, including: wins, losses, earned run average, games, innings pitched, hits, runs, walks, and strikeouts. Similarly, I elected to include the walks and hits per inning pitched stat to give a better understanding of the pitchers real value from the mound. Despite not being included in older cards (especially the Randy Johnson cards) the stat makes it quick to understand how well the pitcher did for that season when measured with (not against) the ERA.

Methodology

API

I started by exploring if there was an API wrapper available in Python, which there was. There was an available repo on Github and PyPi where someone had created a Python package that was a statsapi.mlb wrapper. This would make it 110% more easier and efficient to interact with the API. For the purpose of this project, I chose to do the calls myself using the requests package. The API documentation for the statsapi-mlb wrapper referenced some additional insight into the statsapi-mlb Open API documentation. I experimented by going to different end points, passing different parameters, modifying the URI string, converting to different data formats, and using different hydrate parameter options.

People

The people endpoint has everything I need for the basic player info. I chose to allow the user to interact with this end point through the use of inputting a player name string using the desired player’s full name. This goes through some validation using Regular Expression (RegEx) and then returns a JSON format of the desired players information. After getting the initial bulk response from the people endpoint, I use another function to parse out only the desired data values. This uses a while loop with a nested for loop and append structure. This type of structure works well, but is rather hard to interpret. For this reason, I chose to use simpler for loops for iterating through the different instances of awards or season stats in other functions. I chose to use a data value that wouldn’t be persisted, because it allows me to easily grab season stats from the API. This non-persisted data value is the mlbDebutDate year. I parse out the year using a split method and then pass that year value to the stats function, which then turns the string value into an int. Although it could be seen as needlessly ignoring garbage collection basics, I think this proved to be very helpful and efficient for getting the necessary data.

Draft

The draft data is being pulled from the person endpoint, using the draft hydration, which provides in depth enough metrics. Using the draft endpoint I was able to extract the headshot link, draft team, draft year, draft round, and draft pick. This information is also being extracted in a JSON format using a nested for loop structure to allow variability in the data gathering process.

Data Model

This data model sticks to a kind of JSON data format from the API responses, until it is necessary to move the data into a more human-readable and user-friendly format. The model is centered around dictionary objects and lists of dictionary objects where multiple iterations through the API or API response are required. This was partially required to handle the user input, but also helps in preserving data integrity throughout the transformation process. The end of the model converts the JSON-like dictionary data formats into Pandas DataFrames for ease of transfer into other formats. From the Pandas DataFrames formats we could write that data to a database, export it as a csv, write it to a parquet file, or transform it to a true JSON format. I chose to use csv formats for the sake of this project.

**Limitations**

In my project folder I chose to use ‘Zach Eflin’ (who last month signed to the Rays on a 3 year, $40-million contract. It’s nice to see that his childhood dreams finally came true and got to be a part of the team he wanted to join back in 2012) as the player. I have also tested this project with ‘Jason Adam’, another great pitcher.

The limitation of this project/model is in the ‘stats’ requests. If a player has two ‘teams’ for a single season, the players data can’t be pulled for that season and the for-loop structure throws a ‘KeyError’. This is most likely because Python dictionaries can’t have duplicate keys. When a request is made to the ‘stats’ endpoint duplicate ‘teams’ relative paths (dictionary keys) are returned and Python can’t decide which one to pick.

In order to remediate this issue, one solution would be to parse out the list of all the values and create a new list object of dictionaries. This would allow the first instance of the ‘teams’ key to be parsed into the list of dictionaries properly. Validation would need to be made to see if a ‘teams’ key already exists in a dict in the list object and if it does rename this second instance of ‘teams’ to ‘teams2’ (or similar) and then insert that unique dictionary object into the list object. This is one way of performing this type of technique.

For the purpose of completing this project, I elected to not do this. Consequently, if an attempt is made to create a ‘Shawn Armstrong’ player card, the attempt fails and the program doesn’t complete. (Shawn is really decorated in his awards and stats. It’s a shame his card can’t be created. Shawn has stats for both the Orioles and the Mariners in the 2019 season)

Further, if I wanted to automate this to create a player card each year, I could set this up using Prefect. Set a scheduler to run once a year at a specific day (day after the World Series), use docker to create a container that is running on an AWS EC2 instance, and use the linux bootup configuration program ‘supervisor’ to monitor and manage the server instance, ensuring this program is run on time.

If I wanted to create a card template based off of recent picks, that could be done and scheduled to occur 7 days (or however long it takes for the API to update) after the draft. A templated card could be created for the recently drafted player (assuming they have an mlb debut date in the same year) and then updated after the World Series with the populated data from that season.