Data Wrangling Report

The dataset that I was wrangling from is an API called Stattleship. It has a wide assortment of sports statistics from all the major American sports leagues including the NFL. Whether you want to query by season, by game, by team or by player, Stattleship has the info you need for any project like this. And one thing that made it more convenient was the ability to import the API as a package to use on my Jupyter Notebook to make querying data simpler than from a standard API.

For this project, I needed to get the game logs for each team so I could go through each game from the 2015-2016 season through the 2017-2018 season. In exploring the API, I found most things were labeled conveniently and there really was no missing data. When a team did not achieve one of the features, it gets a “0” (or “False” for Boolean features) instead of “NaN” making it simpler to deal with missing values since they do not need to be dropped or replaced. Also, while there are some outliers, they are standard for a normal NFL season and should still be included because of how much uncertainty there is in the league. In fact outliers are necessary in a project like this because I am trying to find what gives a team the best chance to win and those outliers can have a major impact on a team’s ability to win.

However, one of the biggest issues with using Stattleship is that the API limits how much it will display for a query. So, for example, if I try to query all the stats from one season, I will only get about 40 games worth of statistics when in fact there are many more games. This means that in order to get the game logs that I want, I needed to query many times since there are 32 teams with 16 games for each of the three seasons I am looking at. Clearly, I could not query individually like that so I needed to work on functions and For Loops to obtain the data.

After exploring different ways to query the data and what indexes are used to get certain information, I worked on a function that would help transfer the game names to game slugs. To do that, I made a list that split up the game’s name by spaces, several dictionaries for the key indexes that needed to be changed (including a tuple to make the season slug) and returned them all together. Later on, I would have to update the time dictionary as some game times were minimally different than expected as well as got rid of excess spaces that were in the names especially for games that occurred in January.

The next step was critical, making For Loops to loop through each team and make individual queries so I could get the names for each game. I made two separate lists: one had the three season slugs; the other had 32 team slugs (and thankfully, despite the Rams and Chargers moving, there slugs stayed consistent throughout each season). After that I used a For Loop that did a specialized zip on the seasons and teams to make them tuples that then made individual queries for each combination. These were all added to a list, which was then flattened and looped into another list in order to go through each index to get the game information from each team for each season. Lastly, I extracted the game names from this newer list into another new list. Now I had a list with all of the game names and the next step was getting the statistical information for each one.

After moving the game names to another list, I began creating a For Loop that would go through each game and grab the specific features I wanted. After making an empty list, I began looping through each game using the function I made earlier to transfer the names into slugs and query the data. I chose which features I wanted to include and created a Series to add them to, which would be concatenated to the list at the end of the loop. One issue that came up, however, was that the statistics are split up by the road and home team which means not only did I need to find the right list slicing for each side and use an If statement but also double up the features so each stat was included for the Road and Home team. Another issue was finding out the result of the game and which team won or lost since that was an important distinction. To do that, I defined another function (outside of the For Loop) that would give the names for the winners and losers as well as let me know which team won, the road team or home team. Also, in the rare cases that may arise, I was able to include if the teams tied. The function was then used inside the For Loop to get the winning and losing teams as well as the rest of the features to create a list, which was then transformed into a DataFrame.

Finally, because the For Loop goes through each game for each team, I needed to drop the duplicates in the DataFrame so each game is only listed once. Using the pandas' method "drop_duplicates," the DataFrame dropped all of the duplicate rows. I also filled the "NaNs" that occurred in most of the "Name of Tying Teams" column as well as the "Name of Winning Teams" and "Name of Losing Teams" columns for when a tie was achieved. None of this really impacts the data but I just wanted it to look cleaner. Now I had a DataFrame with clearly labeled statistics for each game for the last three seasons. To wrap it all up, I saved it to a CSV, with the delimiter being tabs, so that the data could easily be extracted moving forward.