Rain Lasch

Computational Reasoning 2: Representations of Data

Prof. Zietz

9 December 2021

For my final project, I want to answer the question of how Twitter performance affects NBA players' game and salary performance. I think this is an interesting question because social media has become an immense influence in people's lives, particularly among famous people and high earning individuals that may utilize social media to further improve their capital and promote their business interests. I also really love basketball and have seen interesting analyses in the past of different players' tweets and their game performance, and I really wanted to come up with a similar analysis as the idea is intriguing.

In order to conduct this analysis, I have chosen to use data from the Twitter API, the Balldontlie API, and CSV data on player salaries from Data.World that I found from an R blog. Using the Twitter API, I planned to collect NBA players' tweet data by their usernames. This API would give me timelines of users' tweets, so from these timelines, I just went ahead and figured out how to count the number of tweets for a given period of time by storing the tweets in a dictionary for each year. For the Balldontlie API, I needed to gather the player's ID from the players endpoint, use that ID to make requests for the player's different seasons as given by the user, and then store the desired data (points per game in this case) inside a dictionary. Then, in order to use this data for analysis with the CSV data, I simply turned the two dictionaries for tweets and season stats into pandas data frames. For the Data.World CSVs, I actually had to use two CSVs: one for players, which had names, player IDs, and game stats, and one for player salaries, which only used player IDs to identify players. So, since a parameter for my desired function would be “player_name,” I matched the player ID with the player name in the CSV, set that player ID to a variable that could be used to search through the salaries CSV, and then picked out the salary CSV rows I wanted for the specific player. All of this process is what I wanted to put into a function, but in order to properly do this for each different player and all the different ranges of user inputs, I implemented various checks throughout the function to make sure the proper data was being retrieved.

To capture the full detail of my coding process, I will do my best to fully explain what is going on throughout the file. I first load in all the different packages I will use, including time, which allows me to place waiting periods throughout the code to ensure that I don’t exceed the Balldontlie rate limit of 60 requests per minute, and MaxNLocator, which I found on StackOverflow and allows me to set graph tickers to integers only. Then, to handle the Twitter API’s OAuth2 Authentication process, I set variables for all the different consumer tokens, keys, and secrets that I received as part of my academic research project that I had to create for Twitter Developers. The most important of these variables is the Twitter bearer token, which is then used in our authorization headers. These headers are crucial and must be included in all Twitter requests, but they are also quite tricky to figure out as I had to find a solution for them on StackOverflow as well. Then, I got all the root URLs in order. Once these variables were instantiated, I could then go on to build my function.

For the function’s three parameters, I used “player_name,” which would be a string of the desired player, “years,” which would be a list of integers for years between 2011 and 2017, and “user_name,” which would be used to find Twitter details for the desired account. I decided to use years between 2011 and 2017 because 1) the players’ salary data only goes up to the season that ends in 2018 (2017-2018) and 2) because the Twitter API only allows Tweets to be retrieved after November 6th, 2010. However, this is still a fairly wide range for our analysis. So, with these parameters in mind, I go ahead and set variables for the root URLs that I will use as well as empty dictionaries for the player Balldontlie ID, season tweets, and points per game stats per season.

With these variables, the first task I have to complete is retrieving and storing the Balldontlie ID for the specific player. So, I use the players endpoint, and since this returns 38 pages of all the different players in the database, I loop through a range of 1 up to, but not including, 39, and for each loop, I check through the page of data, create a current name variable using the current data's "first_name" and "last_name" keys, check our desired "player_name" against the current name, and once the current name is our player name, we set the current name's "id" key value equal to our player name key in the player name ID dictionary. Once this value is retrieved and stored, we break the loop and move onto the next task, but, right after this is done, we check the player name ID dictionary, see if there is anything in there or not after the requests have been made, and if there isn't anything in the dictionary, we tell the user to try another name as the first name didn't work and stop the function with a return command. This is one of various error checks to make sure the user is entering the correct name that we want, for spelling is crucial (requests for "Lebron James" will yield nothing whereas requests for "LeBron James" do).

For the next task of retrieving Twitter and season average data, I first have to store the user's Twitter account ID using their username. Once I store that, it's onto requesting data from the Twitter and Balldontlie APIs. I first run a couple different checks: first, I check to see if the "years" parameter is a list or not and return the function if it isn't; next, I loop through the "years" list and check if the "years" list items are integers, and if they aren't, I return the function; and finally, I check if any of the years is before or during 2010 or after or during 2018 as these years do not have data, and if the wrong years are entered, I again return the function. After these checks, I initialize an empty list for each year in the season tweets dictionary, and these lists will simply store information about each tweet within each year. I set empty strings for start and end dates as I will have to set these to make Twitter requests for a range of time, and I start this process by making a list that each player's games' dates will go into. These game dates will then all be compared to find the minimum and maximum dates for the start and end date. So, I make a request for all of the player's game data, and I loop through the pages of these requests while adding each game's date to the list of dates. I also check if there is more than one page of data for this request, and if yes, I loop through the range of pages, make a request for each page, and through each page, I add each page's games' dates to the list of dates.

The date comparison process is a bit ridiculous, but it works and there seems to hardly be a good way around it. After all of the player's games' dates for a particular season have been added to the "dates" list, I begin by iterating through that list. This further constrains the date range to NBA seasons only as it'll only take into account dates where the player played, so this gives a better range of dates for a season rather than simply inputting 2016 or 2017 as a start or end year. With slice notation, I take each part of the strings that house the dates, turn those parts into integers, and compare those integers with one another to find conditions that will yield the maximum and minimum date. I just set the first date item as the start and end date, then I compare whether each new item has years, months, and days greater than the start date, and if yes, then that date becomes the new end date, so the end date by the end will be the date with year, month, and day integer values greater than any other date in the list. Similarly, I check if each date has year, month, and day integer values that are less than the initialized start date, so by the end, the start date will be the date with the lowest values for year, month, and day compared to all other dates in the list. This then yields proper start and end dates, but if players don't play for a particular season for whatever reason, whether it's injuries or retirement, some years will yield no start and end dates as players did not play for that season. So, I check if the start and end dates are still empty strings after this entire process of date comparison, and if they are still empty, then it's safe to assume the player didn't play that year. So, for these cases, the list for the particular season will only have a value of 0, and the length of this list, which will be used to measure how many tweets a player sent out in a year later on, will thus be 1. Later on in the analysis, if the number of tweets for a particular season equals 1, then we assume the player didn't play that year, so values of 1 in the data frame correspond to this assumption. However, there may still be points per game stats retrieved for the player, so injured or retirement years are just marked by years with number of tweets equaling 1.

Moving on, I check if the start and end dates are, indeed, dates at this point by checking the string length, and if they are, then I can begin making Twitter requests with those start and end dates creating a range of tweets to get a timeline of tweets for our desired range of time. I make a request to the Twitter API, and after doing so, I implement various checks to make sure I retrieve all possible Twitter data for the player. First, I check if the first key in the request has "error" in it, and if it does, I tell the user to input a different Twitter username and return the function. Otherwise, I move on and look through the page of request data. Next, I check if the key "next_token" is in the "meta" key of the request, and if this token is in there, then there are multiple pages of data to go through. With this in mind, if this condition is true, I add all the tweets from the first page of data to the season tweets dictionary, create a while loop that continues for as long as “next_token” is in the request meta, store the next page token for each request page, request a new page of tweets for each new page, and add all those new page’s tweets to the season tweets dictionary as well. This handles the pagination process with Twitter request data, but if there’s no “next_token” in the meta, then there’s only one page to look through. So, with an else statement I control for this, and I simply add all the tweets for the single page to the season tweets dictionary for the given season. I also check if the “result_count” value is 0, and if it is, then I simply append 0 to the year’s list in the season tweets dictionary. This will similarly create a value of 1 for number of tweets in seasons where the players did not tweet, but this value will simply tell us that players either didn’t tweet or were injured in the year. In the end, each year in the season tweets dictionary will have a list of the player’s tweets for that year as well as those tweets’ “created_at” dates. Moving on, for each year I will also run a request to the Balldontlie season averages endpoint to retrieve the points per game stat for each of the player’s desired seasons. This is simply retrieved by going through the one page of average statistics and setting the ‘pts’ key value equal to the points per game dictionary season key. After all this, I have the Twitter and player stats data ready to go, so I move onto working with the players and 1985-2018 salaries CSVs.

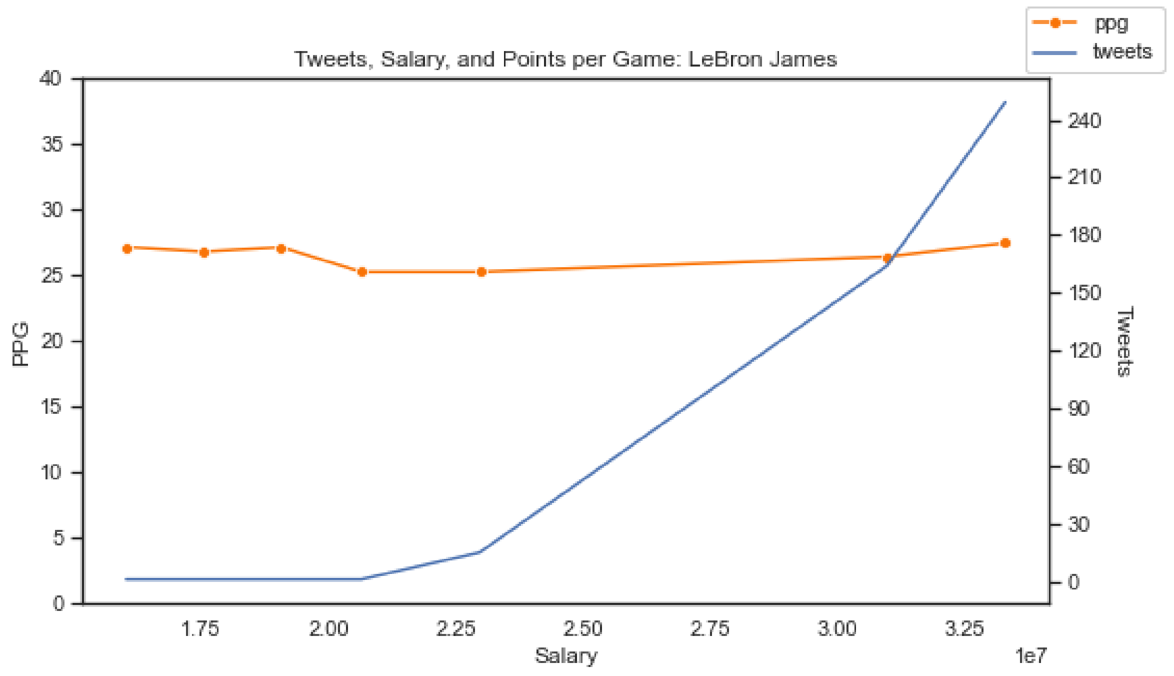
First, I simply load in the CSVs as pandas data frames. Next, I work on getting the player ID that I have to use in the salaries CSV from the players CSV, and to do this, I use loc to retrieve the row that has “player_name” equal to our “player_name” parameter, and I specify the “_id” value after this condition with a comma to retrieve that value. Then, with this loc, I use .items() to turn the “_id” value into an item I can use, and the code for this problem was also found on StackOverflow. After this, using the stored value “salary_id,” I query for rows in the salaries CSV that have “player_id == @salary_id” while also specifying for the other columns I want (team name, season start, ID, and salary). This gives me a data frame of all the salary data for the desired player, but now I also have to filter out the data that does not correspond to the desired years in the "years" parameter list. So, with help from Tian, I further filtered for years that I wanted using the "season_start" variable in the data frame as this variable has the year that would correspond to the seasons that we would want (if we matched to "season_end" on the other hand, it would not be what we wanted). To find the years I wanted, I used .loc again, but this time, I did .loc[player_salary['season_start'].isin(years)] as .isin would look at all the items in the "years" list and find rows that had "season_start" values that were also in the "years" list values. This finally gave me the part of the data frame from the salaries CSV that I wanted, but I had to further merge this data with the tweets and stats data so that I could chart and interact with the data altogether.

For merging all these pieces of data into one cohesive data frame, I got more help from Tian and learned a lot about merging data frames and also turning dictionaries into data frames. To make each dictionary into a data frame, I used pd.DataFrame(dictionary.items()) for each dictionary as the .items() method would put the various items of the dictionary into data frame columns. Then, for the sake of merging data frames, I named each of these dictionaries-turned-data-frames' year columns "season_start" so that I could merge all the data frames on a common column (remember that the salaries CSV has a "season_start" column that we want to use). In the season tweets data frame, I went about creating a column named "num_tweets" as well by applying a lambda function that took the length of the list in the newly created "tweets" column, and this application returned a column of integers that could now be interpreted as a player's number of tweets for a particular season. Finally, I merge the tweets and salaries data frames by using the .merge method, and I do this by using 'on = "season_start"' and 'how = "outer"' to match the data frames on the common column while also retaining all the data of the previous columns. This literally appends the second data frame to the outer edge (end) of the first one. To finish the frame off, I merge this new data frame with the points per game data frame as well, and now, I have a complete data frame to work with.

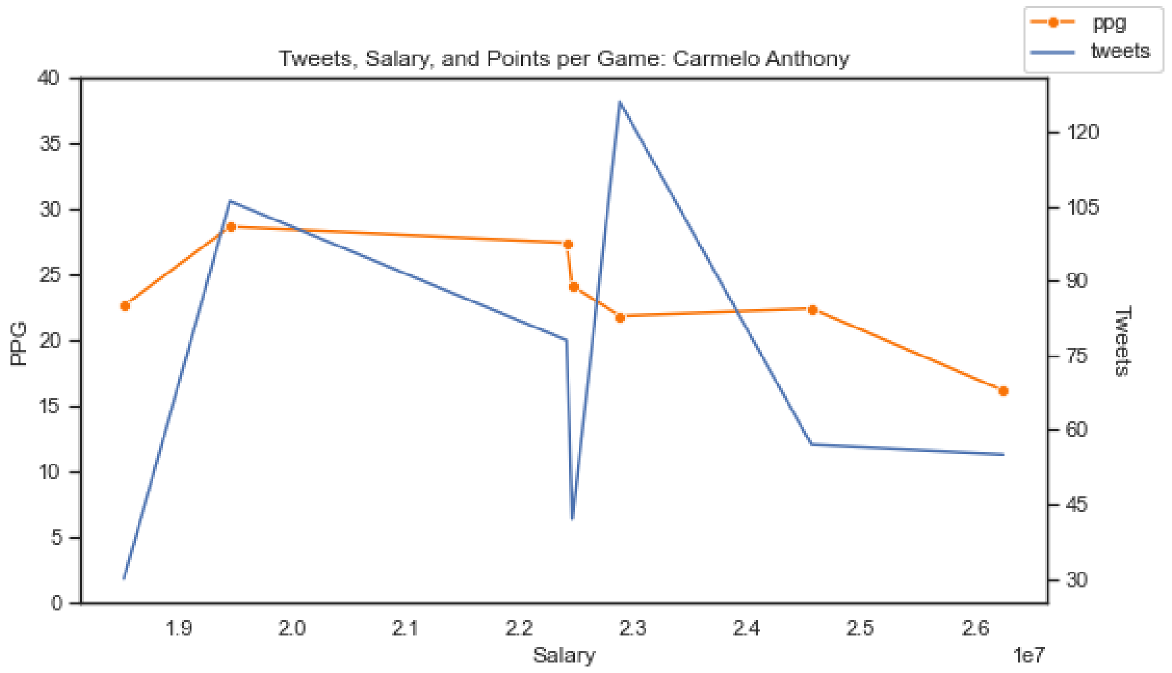
Using this function, I can now call the function to return a data frame of players' tweets, points per game, and salary for a season, and I can set this call to a variable to further chart this data. So, I set data frame variables for LeBron James, Chris Paul, and Carmelo Anthony, and I call the function for each of these players from 2011-2017. To plot this, I created a second function to create and save plots. The two parameters for this are "player_name," which will be set as a string to help with chart and file titling, and "data_frame," which will just take pandas data frames (that we set as separate variables for each player) to use for charting. I created a variable called "no_space_name" which replaces spaces with underscores in the given name, and this variable will be used for the naming of our PNG file as computers more easily handle underscores in file names. Then, I begin plotting. I set the size to 9x5 and the style to white, and this style is the best to me as it creates a nice frame around our chart without any unnecessary horizontal lines. I have decided to create a chart with two Y axes as well, so I set the first axis as "ax1." For both lines, I plot salary on the X axis, and this salary is put into scientific notation (as noted in the corner of the axis by 1e7) for ease of reading as player salaries often go into the tens of millions of dollars. For the first line, I plot points per game with the marker equal to "o," which gives nice dots on the points where data actually lie. With this line plot, I also set the title of the chart equal to "Tweets, Salary and Points per Game:" + str(player_name) (which would be "Tweets, Salary, and Points per Game: LeBron James" for example), and I set the axis titles. I also set the Y axis limits for points per game from 0 to 40 as this seems like an appropriate range to measure points per game given that players rarely exceed 40 points per game in a season. This axis scale will give a good picture of shifts between players' points per game stats for different seasons. Then, I create a second axis and plot the number of tweets over salary, and I also use the mako palette for this line to utilize a color similar to the color of the Twitter logo so that the line might be more easily interpreted as the number of tweets. I set the major locator for the MaxNLocator to an integer so that the second axis has integer ticks similar to the first axis. I add a legend for each axis so that the reader can also understand what each line in the plot is measuring. Finally, I save the figure using savefig and the "no_space_name" variable so that graphs will be saved and easily readable. For example, LeBron's chart would be saved as a PNG file named "LeBron_James_twitter_stats.png".

At the end of coding, I was extremely surprised to see the results of my analysis in the different player graphs. I did not find an overall trend that I could spot for all players as I think these kinds of conclusions would require much more data and more time-specific analysis with tweets and game data. However, on a player to player basis, there are some interesting insights to be made about how each player tweets, earns, and performs.

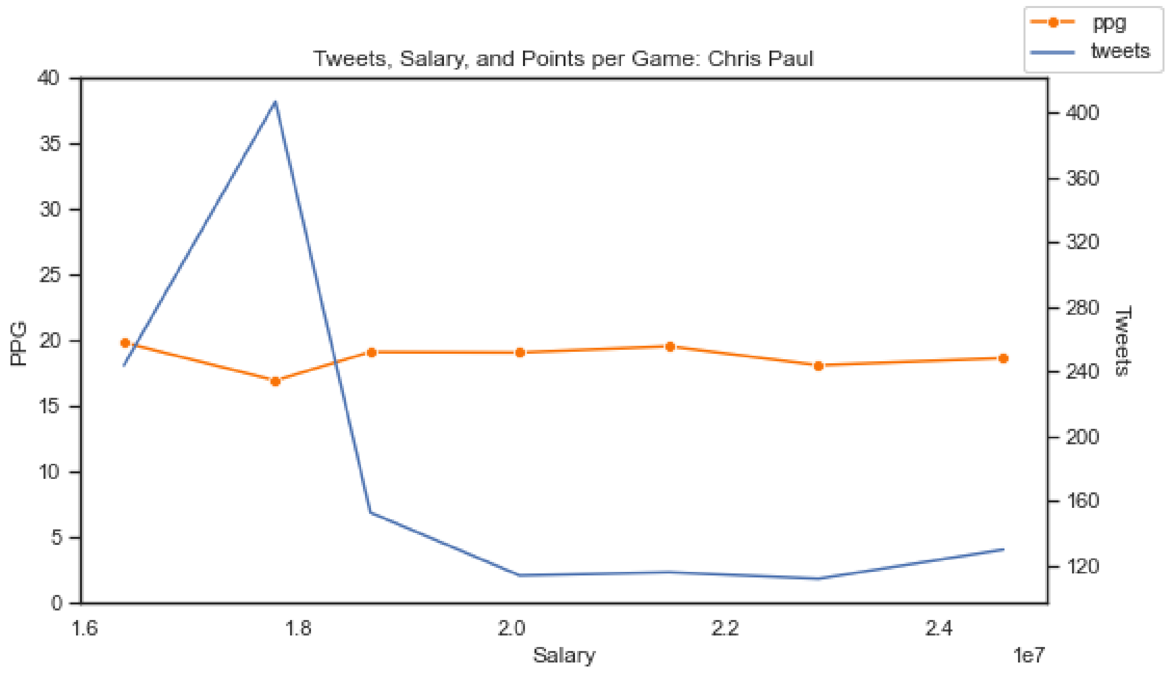
I will start with LeBron James. While LeBron earns at his lower salary range, LeBron does not tweet at all and has some of his highest points per game stats. Then, as his salary increases, LeBron's points per game remain at a plateau around 25-27 points, but his tweets increase immensely as he quickly moves from tens to hundreds of tweets between seasons. So, while LeBron's salary increases, this may not necessarily encourage him to score more points but may instead encourage more leisure and social media activity for the King. This is a very subjective conclusion, but the graph clearly supports this view.



For Carmelo Anthony, the graph is somewhat all over the place at first glance. At the lower end of his salary range, as Melo's points per game increases, his tweets increase as well. Then, as his salary further increases, both his points per game and tweets moderately decrease. Then, suddenly, Carmelo's tweets dramatically drop and his points per game also drops from around 27 to 24 points per game as his salary experiences another increase. Another sudden fluctuation occurs again as Melo gets another salary increase, loses a couple more points per game, and starts tweeting much more frequently at over 120 tweets for that particular season. In the final two increases of his salary, his tweet frequency falls dramatically while his points per game get a slight bump, and then both tweets and points per game fall rather significantly as Melo reaches his highest salary. Carmelo's relationship between tweets, points per game, and salary earnings is quite erratic and spurious, but it could be interpreted from the graph that tweeting hurt Carmelo's game performance but also enabled him to start earning higher salaries as the most significant increase in his tweets corresponds to a significant reduction in his points per game. However, this relationship is a bit difficult to interpret from this graph as higher tweet counts initially led to an increase in his salary and points per game. Causation cannot be concluded from these graphs, but attempting to interpret these graphs is nonetheless interesting. No matter what, it seems fairly clear that after Carmelo got his first salary increase, paying Carmelo Anthony more money did not encourage him to score more points whatsoever as the points line steadily decreases with only slight bumps after his initial salary bump.



For the final player, I will look at Chris Paul's Twitter performance. One thing is clear from Chris Paul's graph: Chris Paul consistently scores above 15 points per game no matter what his tweets or salary are. However, Chris Paul does tweet a lot, and after his first salary increase, Chris Paul jumped from ~240 tweets/season to over 400 tweets, and it is clear that his points per game experienced a dip following these changes as it fell from about 20 ppg to around 17 ppg. It seems that Chris Paul may have realized this, stopped tweeting so much, and then started to score closer to 20 ppg while also starting to earn more money. However, his slight surge in tweets with a corresponding dip in ppg at the higher end of his salary range suggests again that tweeting may negatively affect Chris Paul's ppg performance, but it may actually enhance his earnings potential. Thus, while tweeting may contribute to worse game performance for CP3, it may actually have allowed him to earn more in the first place, and his lower, but moderately high, amount of tweets may be the perfect sweet spot for not affecting his game performance while also allowing him to continue earning more money.



In all, these conclusions from graphs are very subjective and not conclusive or causal in any statistically significant way, but the insights that these visualizations provide are intriguing nonetheless. I think that if I were to continue developing this project, I would want to figure out a way to make more time-specific visualizations that could visualize when players started tweeting more and their game-specific stats. This way I could answer questions relating to player mention frequency within certain weeks or other tweet frequency within certain periods and then relate that to their specific game performance. Then, I could also track whether certain tweet frequencies increased the likelihood of certain players meeting specific contract goals that would pay the player bonus (clauses like whether or not players become All Stars or not). I would also hope to figure out a different way to get NBA players' stats as the Balldontlie API was good for this project but isn't really good for big data analysis due to its rate limit of 60 requests per minute. Because of this limit, I had to add various time.sleep() methods throughout my get_tweet_info function to ensure that I didn't exceed 60 requests per minute while also retrieving all the desired data. These sleep methods thus made the function take forever to work with. I think a CSV for players' season averages and game stats would be much more useful and time efficient for this reason. I might also want to figure out a way to create more easily readable graphs as, even though the graphs look interesting, the graphs are hard to read due to the presence of three axes for three different continuous variables. I think that two separate graphs with two axes (like one graph for salary and PPG and another graph for salary and tweets) graphed over the season_start would be much easier to interpret. This would give a better picture of changes between years as well. Initially, I wanted a graph with two line plots over a bar chart for salary, but this turned out to be a less viable option than the graphs I came up with. At the end of the day, I am extremely happy that I got my functions to work and was able to come up with some nice visualizations from this data, and I think the final graphs are complex and insightful.

Data Sources

- Salaries and Players CSV
  - Blog: https://www.r-bloggers.com/2020/08/nba-salaries/
  - Data.World Workspace with CSV Files: https://data.world/rainer27/twitter-nba-analysis/workspace/file?agentid=datadavis&datasetid=nba-salaries&filename=players.csv
- Balldontlie API
  - https://www.balldontlie.io/?shell#introduction
- Twitter API
  - https://developer.twitter.com/en/docs/twitter-api
- StackOverflow Help Sources
  - MaxNLocator for Graphing → https://stackoverflow.com/questions/30327153/seaborn-pylab-changing-xticks-from-float-to-int
  - Twitter OAuth Authentication → https://stackoverflow.com/questions/21651846/oauth-access-token-request-twitter-api-and-oauth-verifier-field/21653087#21653087
  - First Dictionary Key → https://www.geeksforgeeks.org/python-get-the-first-key-in-dictionary/
  - Pandas Value to Item → https://stackoverflow.com/questions/53255796/how-to-get-a-single-value-as-a-string-from-pandas-data-frame
  - Outer Merge → https://www.tutorialspoint.com/python-merge-pandas-dataframe-with-outer-join
  - Moving Axis Labels in Plots with Padding → https://stackoverflow.com/questions/43875258/how-to-change-the-positions-of-subplot-titles-and-axis-labels-in-seaborn-facetgr
  - Two-Axis Legends → https://stackoverflow.com/questions/47591650/second-y-axis-time-series-seaborn
- Time Module

  - https://www.programiz.com/python-programming/time/sleep
- Break Statement
  - https://www.tutorialspoint.com/python/python_loop_control.htm