|  |
| --- |
| Center for Modeling and Simulation (2018-20) |
| Modeling Shot quality Using Expected Goals metric |
| Report 1 |
|  |
| **Nirmitk Tripathii** |
| **1/8/2021** |

**ABSTRACT:**

|  |
| --- |
| This is a short report detailing the amount and kind of work I have been doing and gives an intermediate introduction to the techniques of web scraping football data from various websites and how to manipulate and engineer it to do valuable analytical work. This data will also be involved in build Machine Learning models in the future. This report sets the scene for the upcoming Machine Learning models involving Expected Goals (and beyond), measuring shot quality and quality of play, which will be the contents of the next report |

# Table of Contents

Contents

[Table of Contents 1](#_Toc78808465)

[Introduction 2](#_Toc78808466)

[Data Acquisition 2](#_Toc78808467)

[Web-scraping 3](#_Toc78808468)

[Website research and structure of data 3](#_Toc78808469)

[Working with JSON 6](#_Toc78808470)

[Understanding data with Python 6](#_Toc78808471)

[Manipulations to make data as in the original source 8](#_Toc78808472)

[Shot location dataset from Understat 11](#_Toc78808473)

[Web-scraping player shot data 11](#_Toc78808474)

[Final Outcome 17](#_Toc78808475)

[Future Work: 21](#_Toc78808476)

[References: 21](#_Toc78808477)

# Introduction

In the era of big data and artificial intelligence, many different domains such as healthcare, finance, banking, retail, travel, social media and many more have started employing mathematical models on the available “big data” to analyze and improve their existing systems. Sports analytics is one such branch which has been attracting increasing interest due to availability of advanced sensing technologies for data acquisition which make the data available. This, coupled with the ability of mathematics and statistics to model anything within a given set of rules, makes mathematical modeling of sports, such as football, an interesting topic of intrigue to dive into.

Football and maths start from the same point- **“laws of the game”**. For football, these laws are decide by the International Football Association Board. For maths, these laws are defined by nature/axioms. With a lot of hardwork and a little bit of creativity, and inspiration, both footballers and mathematicians seek to reach their respective goals. It is this particular area of modeling football, where this project will be headed towards.

# Data Acquisition

Historically speaking, one of the biggest hindrances in doing any kind of sports data analytics was the lack of data availability. The fundamental difficulty in attempting to make an objective study of team performance at soccer is a lack of routinely recorded quantitative data. The first requirement, therefore, is the availability of this data and how to obtain it.   
  
One of the earliest forays into statistical analysis and modeling of football was made by Charles Reep in 1968, where he was trying to analyse the relationship between number of passes, possession and goals scored. In his pioneering work, he utilized the data he acquired through the 1950s to model goal scoring probability on the number of passes in a pass chain.

He defined and designed a notational technique which enabled footballing data from a match to be recorded on paper. The real-time continuous stream of actions of the game are broken down in a series of on-the-ball discrete events like pass, shot, tackle, foul etc. and recorded. Furthermore, a detailed categorization of these events is also done, adding up more detailed information related to each such event. For instance, a pass event recorded is further categorized by additional features like the length, the direction, the outcome and the height of the particular pass. 60 years ago, this cumbersome process was done by hand on pen and paper. Nowadays, there are dedicated professionals and even automated softwares (under human supervision) that record football events as such and report it to huge databases created, managed and utilized by some of the very prominent sporting organizations and companies. Since 2012, many such data acquisition and statistical and mathematical analysis companies have cropped up. Presently, the 3 most prominent organisations are StatsBomb, Wyscout and Opta.

One of the major issue faced by modern day data analysts and modelers is the issue of proprietorship of this data by those data acquiring organisations, making on-the-ball event data sets almost impossible to be accessed by anyone other than those who pay for it, basically creating a pay-wall between amateur data analysts and sports enthusiasts.

However in 2019, events data for various matches in various competitions for different seasons was made available by Statsbomb but not for all the matches in the season. For example, events data for only Barcelona’s matches for all the seasons from 2007/08 to present was available for public access by Statsbomb.

Wyscout is another organization that keeps track of events data for every team and every match across various first tier leagues around the world for several seasons. They have made the complete set of events data across the Top 5 European Football Leagues (English Premier League, Spanish La Liga, Italian Serie A, German Bundesliga and French Ligue 1) for all matches for the entire 2017-18 season, freely available for public access to be utilized in football data analysis.

## Web-scraping

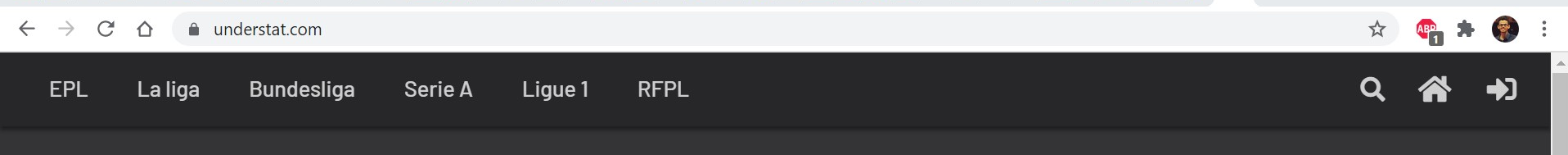
Apart from [Stasbomb](https://github.com/statsbomb/open-data) and [Wyscout](file:///C:\Users\Nirmit\Desktop\6.%09https:\figshare.com\collections\Soccer_match_event_dataset\4415000\5) data, certain valuable events data like shots taken are available on certain websites like [understat.com](https://understat.com/) from where these shot events data can be web-scraped and utilised.   
This can be achieved by utilising data scraping techniques in python. I started with iporting the following libraries:  
We start by importing libraries that will be used in this project:

* **numpy** – fundamental package for scientific computing with Python
* **pandas** – library providing high-performance, easy-to-use data structures and data analysis tools
* **requests** – is the only Non-GMO HTTP library for Python, safe for human consumption. (quoted from official docs)
* **BeautifulSoup** – a Python library for pulling data out of HTML and XML files.
* **json**- Python library to handle and work with JSON format data.

### Website research and structure of data

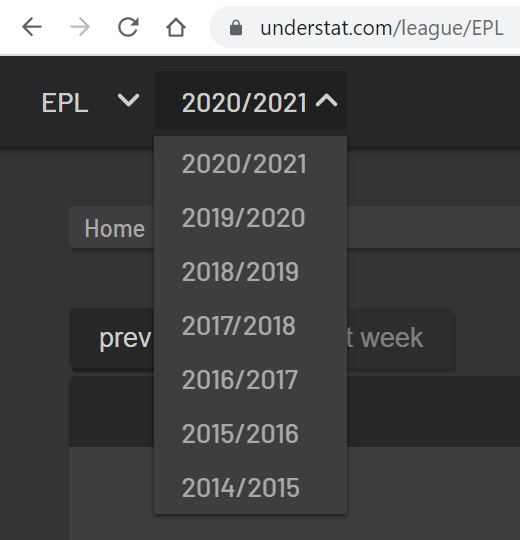
In any web scraping project first thing that needs to be done is researching the web-page one wants to scrape and understand the working behind it. That’s fundamental.

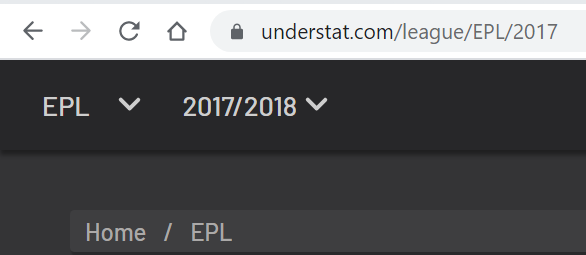
On the home page we can notice that the site has data for 6 European Leagues:



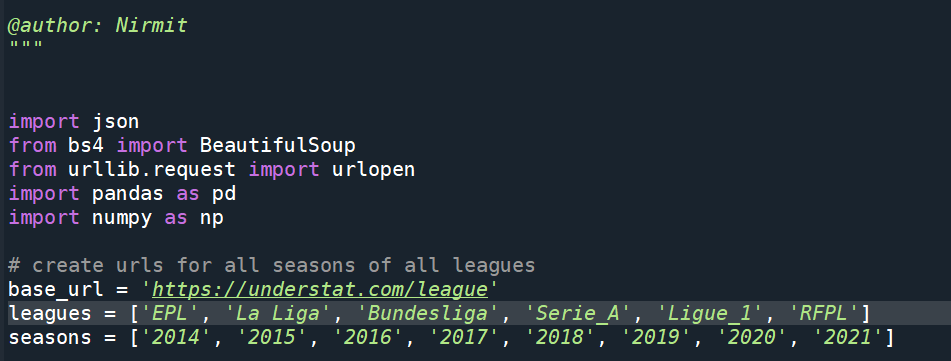
* **EPL (English Premier League)**
* **La Liga (Spanish Division 1)**
* **Bundesliga (German Top Division)**
* **Serie A (Italian Premier League)**
* **RFPL (Russian Division 1)**

Also from the website, we can see that shots data from the season 2014-15 are available.

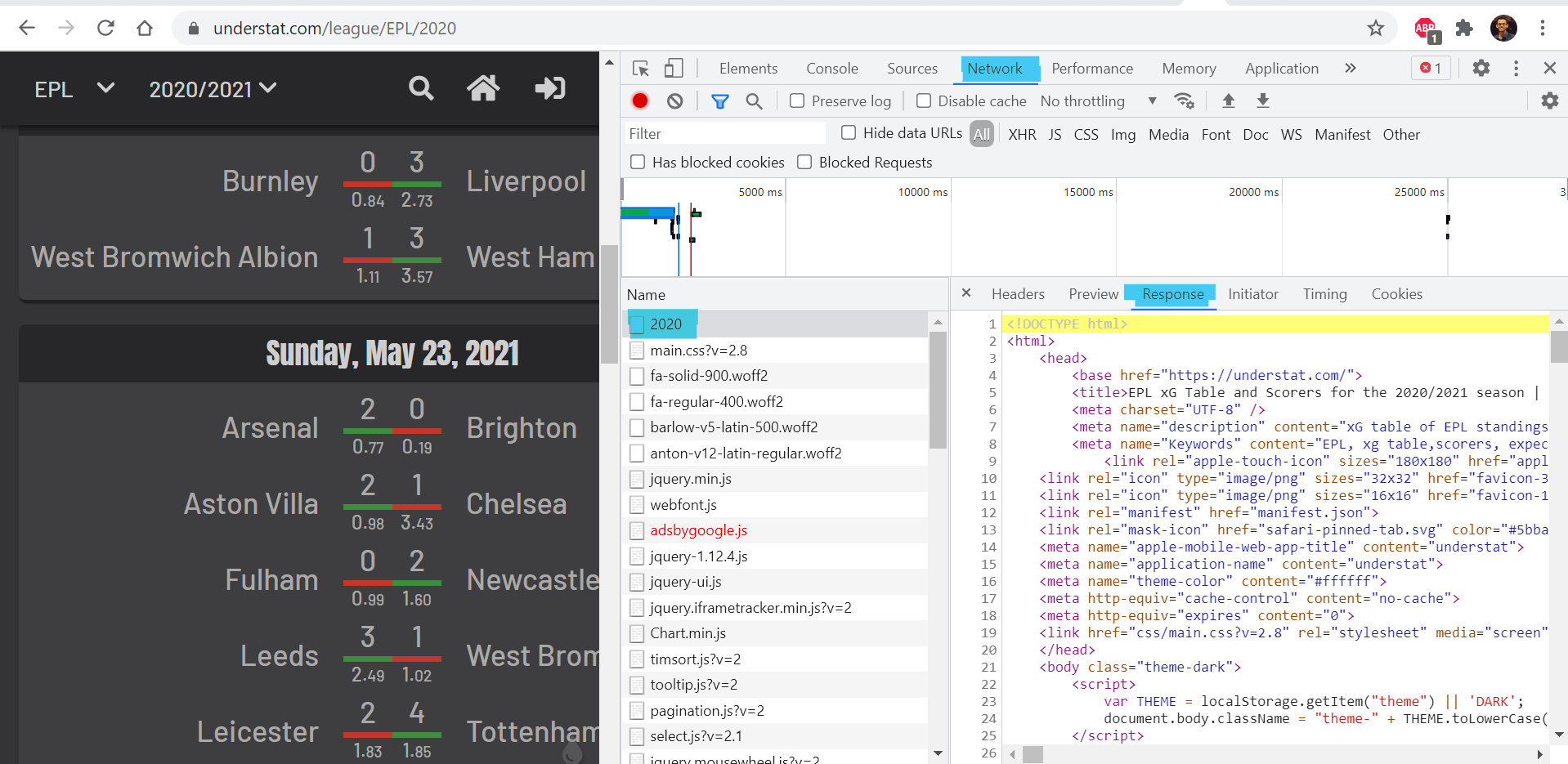


Another interesting information that can be seen and deduced is the fact that the URL of the webpage is a structured one, i.e, ***‘https://understat.com/’ + ‘name\_of\_the\_league/ ‘ + ‘/year\_start\_of\_the\_season****‘*. For example, if we want to see the information from EPL 2017/18 season, the URL of that webpage will be: 

Therefore we can automate the process of accessing the webpages of all the leagues for all the seasons in our program, which will make our code automatically access all these webpages during the web-scraping.



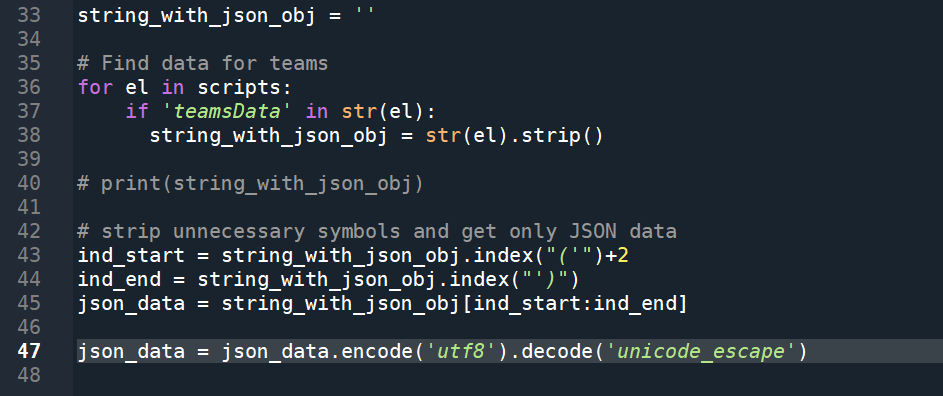
Next step is to understand and find out where the data is located on the web-page. For this we open *Developer Tools* in Chrome (Press F12), go to tab **“Network”**, find file with data (in this case 2020) and check the **“Response”** tab. This is what we will get after running requests.get(URL).



After going through the contents of the webpage, I found out that the valuable data is located just after the 3rd occurrence of **“<script>”** in the **response** tab, which is JSON encoded. So, I’ll have to find this tag, get JSON from it and convert it into Python readable data structure.

******

### Working with JSON

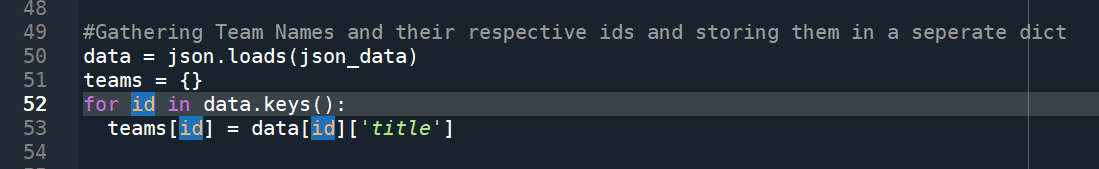
As mentioned above, the valuable data is stored in *teamsData* variable (3rd instance of **“<script>”)**, after creating a soup of html tags it becomes just a string, so we find that text and extract JSON from it.

Once we get the JSON data as a dictionary, we can now manipulate it like any other python dict object.

### Understanding data with Python

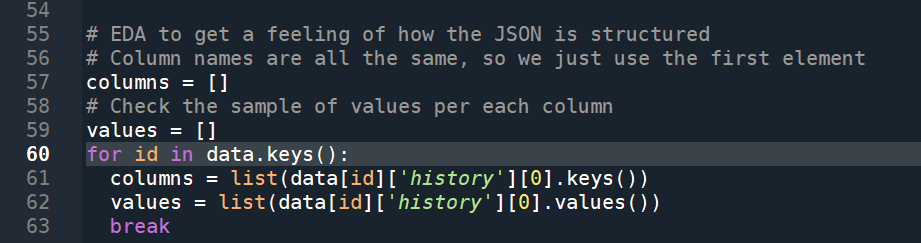
If we dive into the python\_dict we got from JSON data, we’ll find that the python dictionary is actually a nested dictionary of 3 keys: ***id***, ***title*** and ***history***. The first layer of dictionary uses ***ids*** as keys too.

Hence, we can gather team names after going over the first layer dictionary.

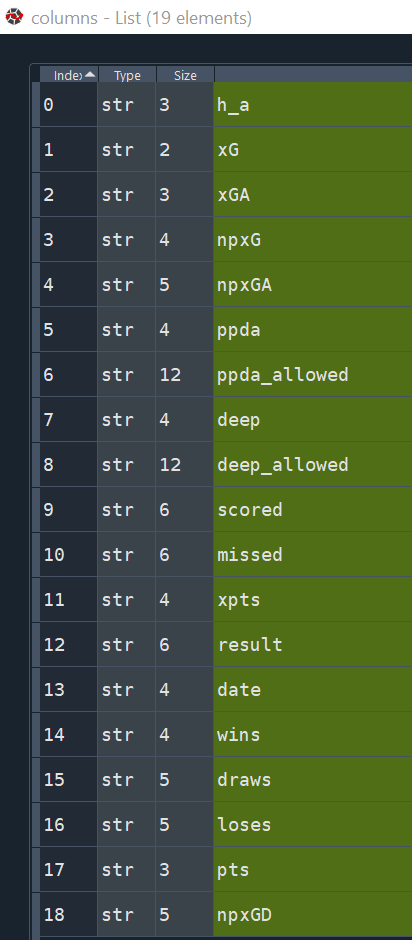


Also from the nested dictionary structure we understand that ***history*** has data regarding every single match the team played in its own league (League Cup or Champions League games are not included). The ***history*** is the array of dictionaries where keys are names of metrics (read column names) and values are the values of the respective metrics.

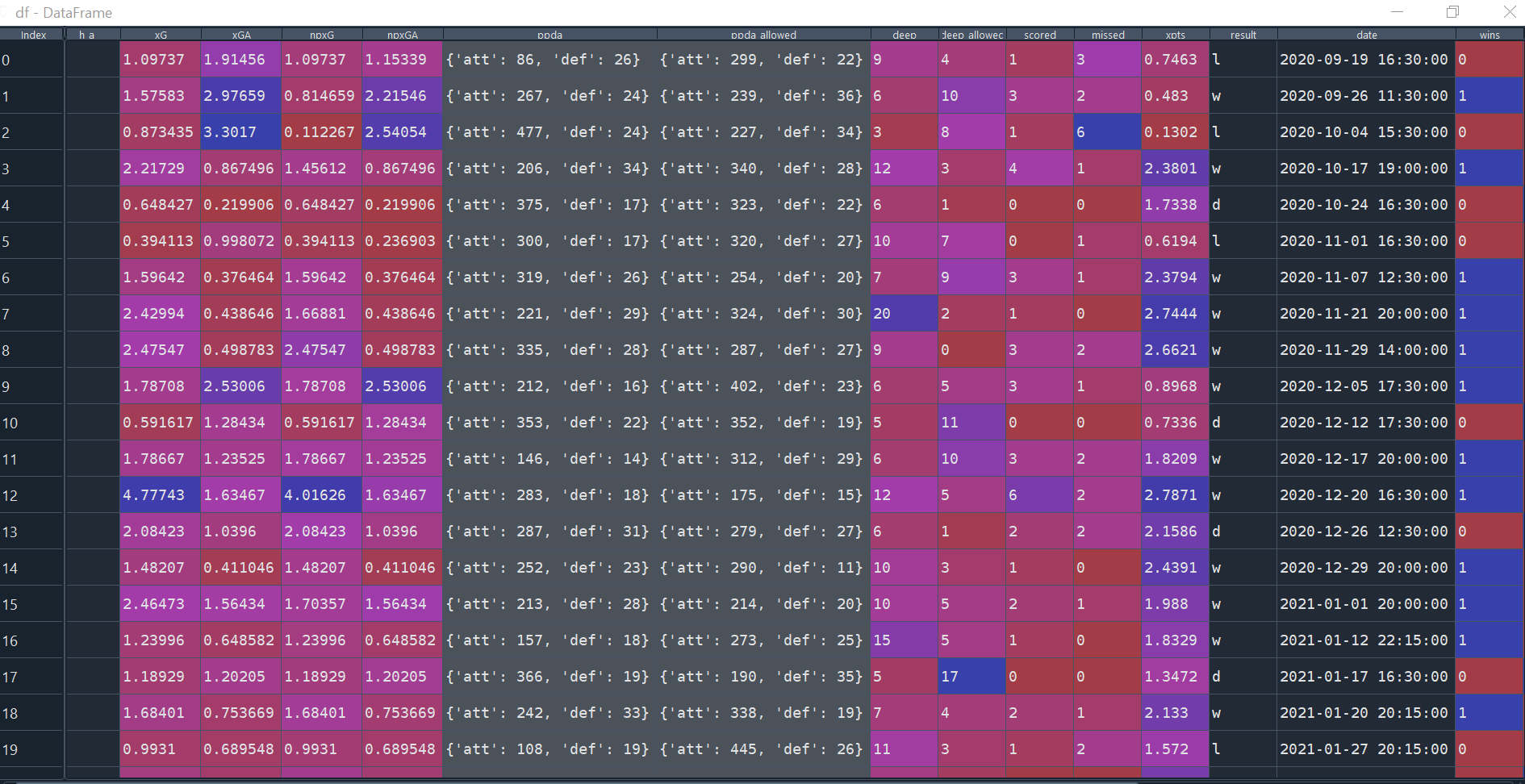
It is understood that column names repeat over and over again so I add them to separate list. Also checking how the sample values look like.



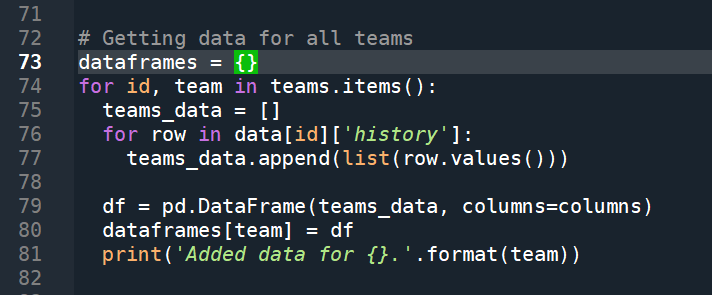
The metrics stored for every team in the data is given below:



The team\_id for Manchester United is 89. Therefore, getting all the data for this team as a test case to reproduce the same steps for all the teams in the league. The resulting dataframe with data for Manchester United during the entire season looks like this:



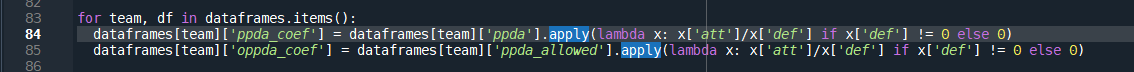
Having successfully accessed a single team’s data for an entire season into a dataframe, I looped through the dictionary for all the teams for the entire season to get the data for all of them.



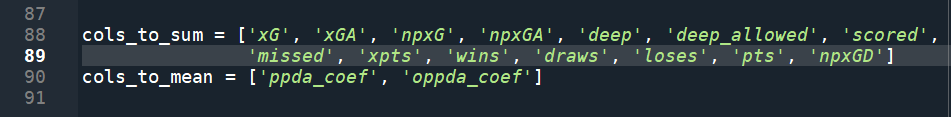
The above code gives us a dictionary of DataFrames where key is the name of the team and value is the DataFrame with all games of that team.

### Manipulations to make data as in the original source

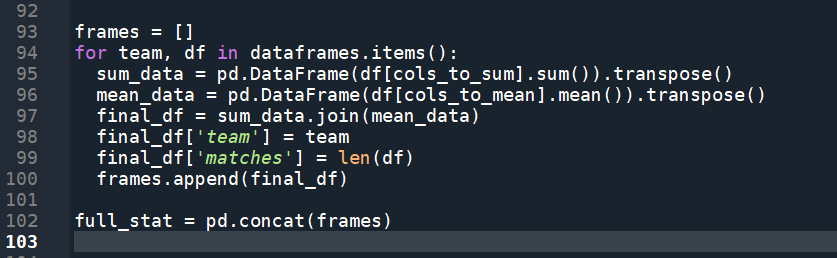
When we look at the content of DataFrame we can notice that such metrics as PPDA and OPPDA (**ppda** and **ppda\_allowed**) are represented as total amounts of attacking/defensive actions, but in the original table it is shown as coefficients. In order to fix that, we need to divide the 2 parameters to get the coefficients.



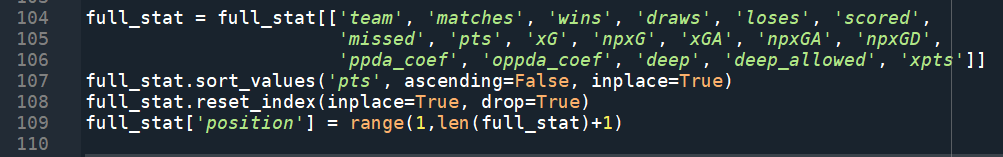
Now we have all our numbers, but for every single game. What we need is the totals for the team. For that we have to find out the columns to sum up. For this we go back to original table at [understat.com](https://understat.com/league/La_liga/2018) and find that all metrics should be summed up and only PPDA and OPPDA are to be averaged in the end.



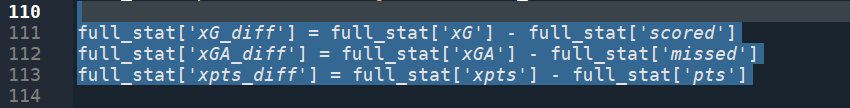
In order to calculate our totals and means, we loop through dictionary of dataframes and call **.sum()** and **.mean()** DataFrame methods that return Series (that’s why I added **.transpose()** to those calls). We put these new DataFrames into a list and after that concat them into a new DataFrame ***full\_stats***.



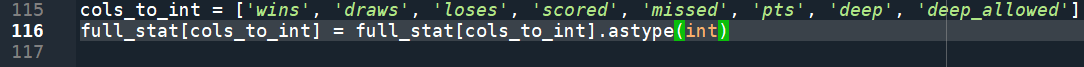
Then, I reorder columns for better readability, sort rows based on points, reset index and add column ‘position’.



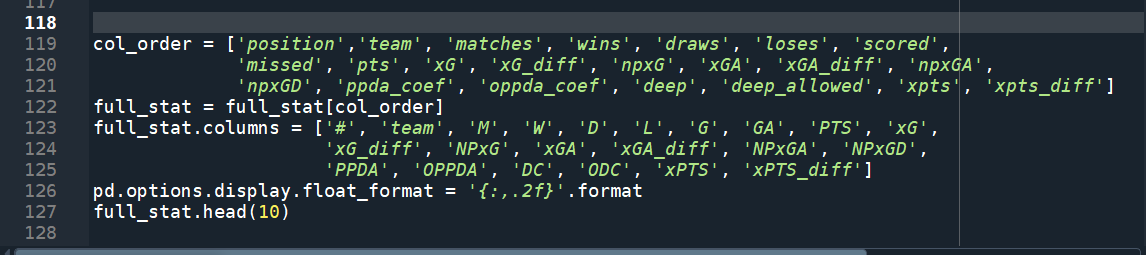
Also in the original table we have values of differences between expected metrics and real.



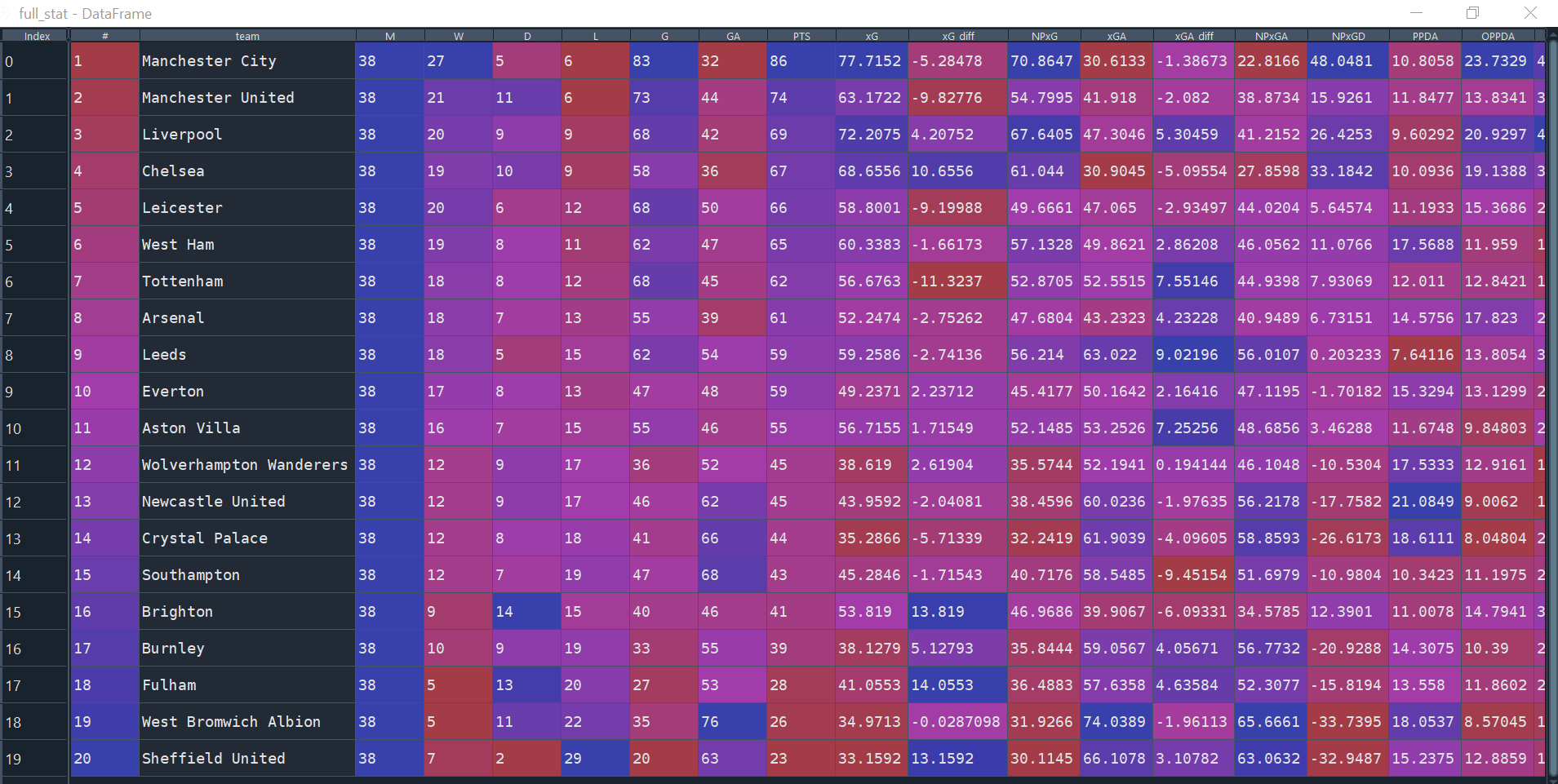
Converting data types where appropriate.



Cleaning the data to make the DataFrame look more like the original table.



Final DataFrame:



Original table:  
  


Now when we got our numbers for one season from one league we can replicate the code and put it into the loop to get all data for all seasons of all leagues.

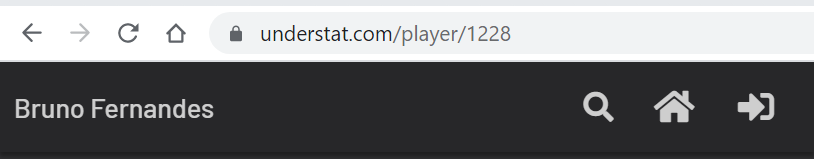
### Shot location dataset from Understat

Shot location (X-Y coordinates) of each and every shot taken by all the players in the above mentioned 6 leagues from 2014/15 season to 2020/21 season can aalso be scraped similarly. However, the github account of ***aritrartira*** already had the data available for public use, so there was no need to web-scrape it. (This will be useful in future works, when I build up an “Expected Goals” model on the probability of a shot being scored)

### Web-scraping player shot data

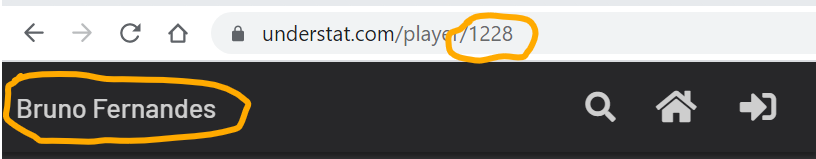
Similar to the understat website for league data, we can research into each individual player’s data from the same website. Following the website research for player data we found that a similar URL structure, as that of league data, exists for Understat player data which goes like:

***‘https://understat.com/’ + ‘player/‘1 + ‘player id***.

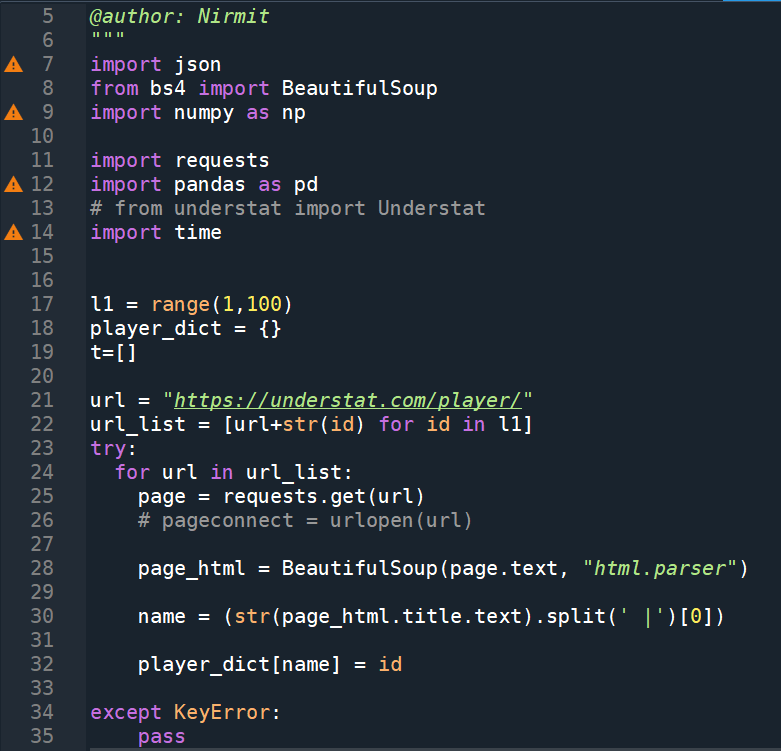


The existing issue here is the fact that we need to know the “player\_ids” of every player in order to scrape each and every player data. For example, ***Bruno******Fernandes*** here has a player\_id ***1228.***

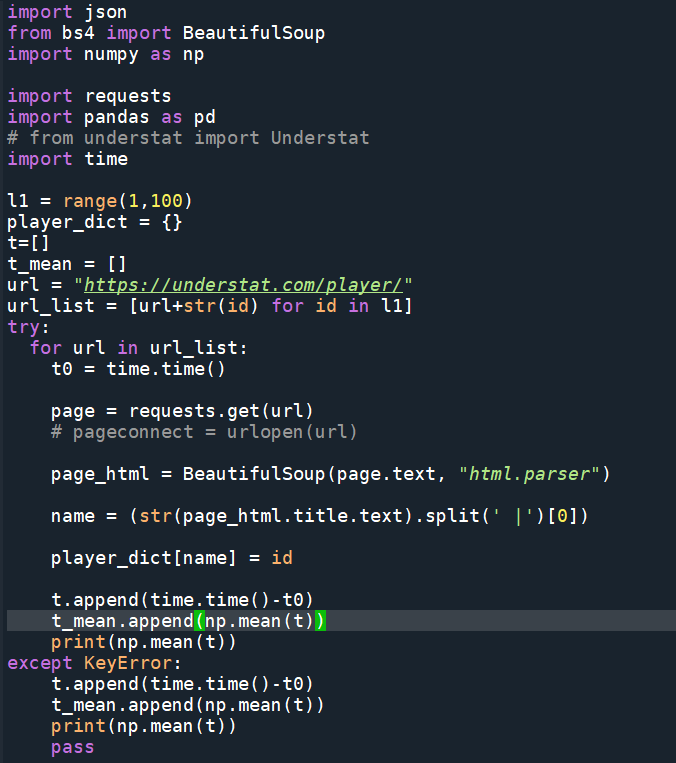
In order to overcome this particular problem, I decided to create a dictionary of players with player names and their ids as key-value pairs. For that I understood that the title of the webpage holds the player name for the player-id mentioned in the URL:

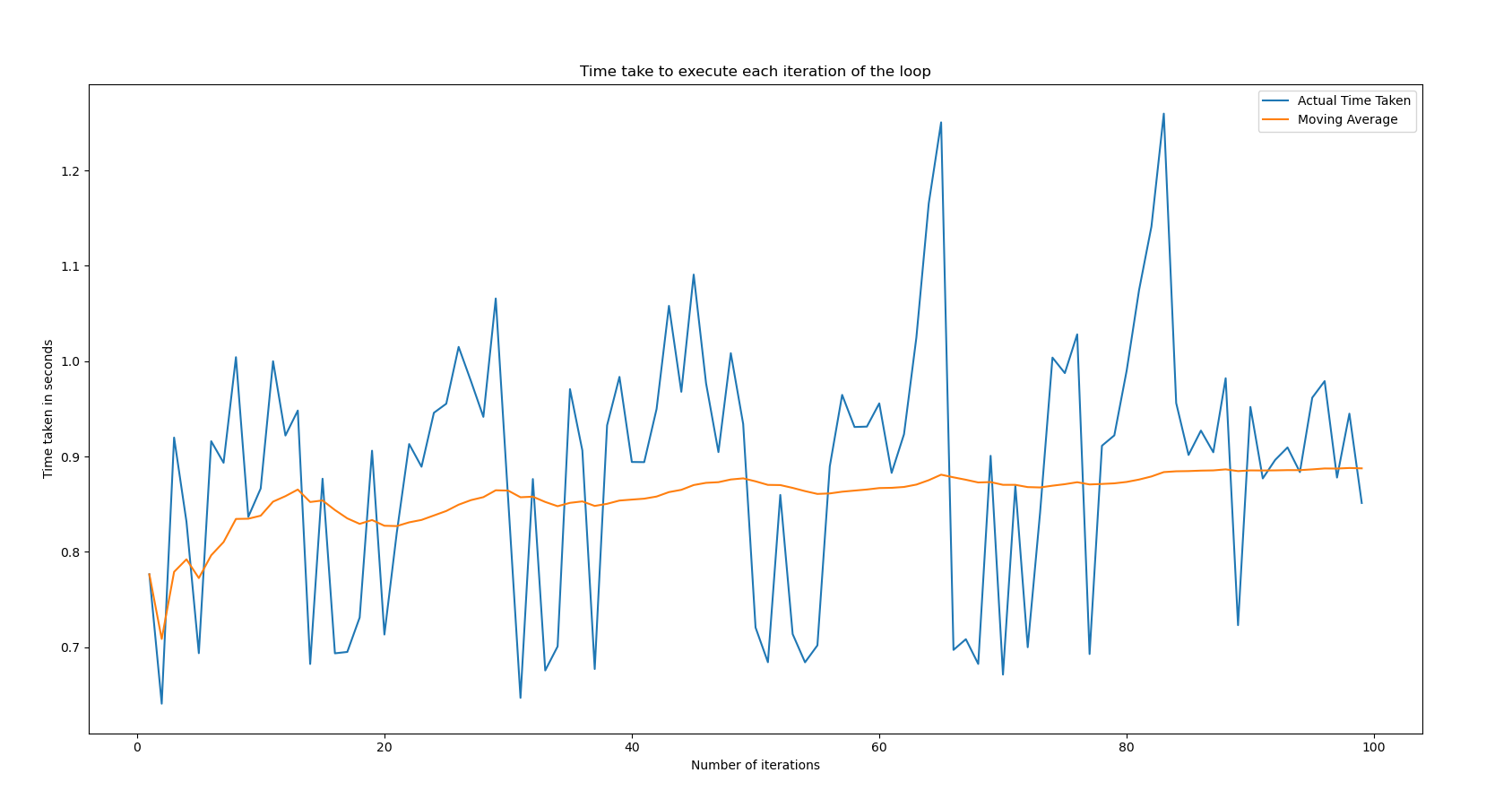


This was enough for me to manufacture a code to get a player name-player id dictionary for first 100 players.



However, it took the code a significant amount of time to execute this code for just 100 players. So I decided to time the code in order to get the average time the code takes to execute for 100 players.





The average time taken for the code to create a dictionary of 100 player name-player id key-value pairs was 0.8875 seconds/iteration, or 88.75 seconds in total. For 100 or even 1000 players, this is a reasonable time duration, but for higher number of players (which can even be as high as 10000), we need to implement multithreading and/or multiprocessing techniques, as opposed to regular python processing techniques which execute the codes sequentially but asynchronously.

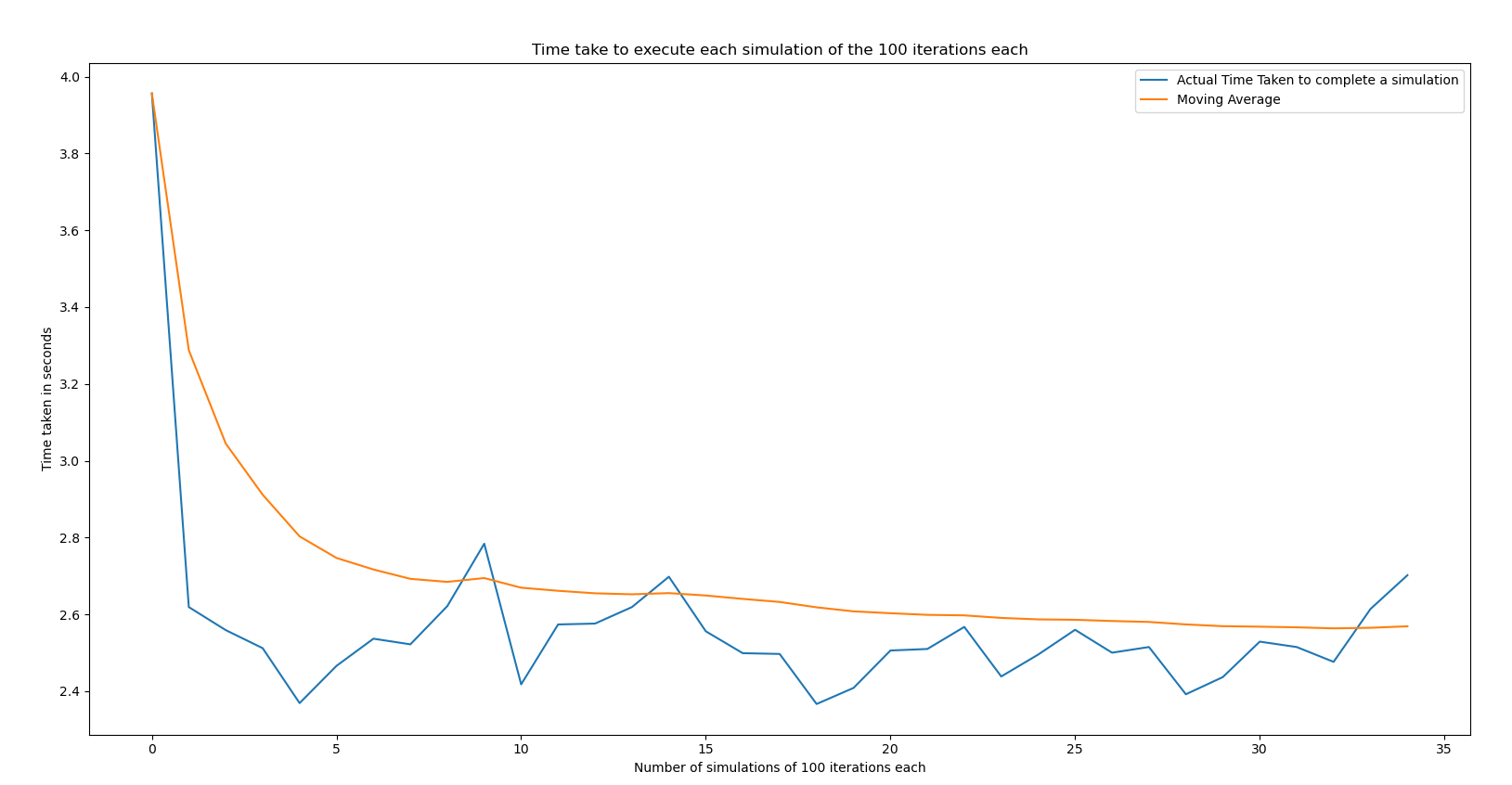
#### Mutiprocessing and Mutithreading:

Multiprocessing refers to the ability of the system to divide a task into multiple processes and then execute those processes on multiple processors (usually = No. of cores in the system’s CPU). Several processes/tasks are performed simultaneously by the CPU, with each task being executed by a separate processor.

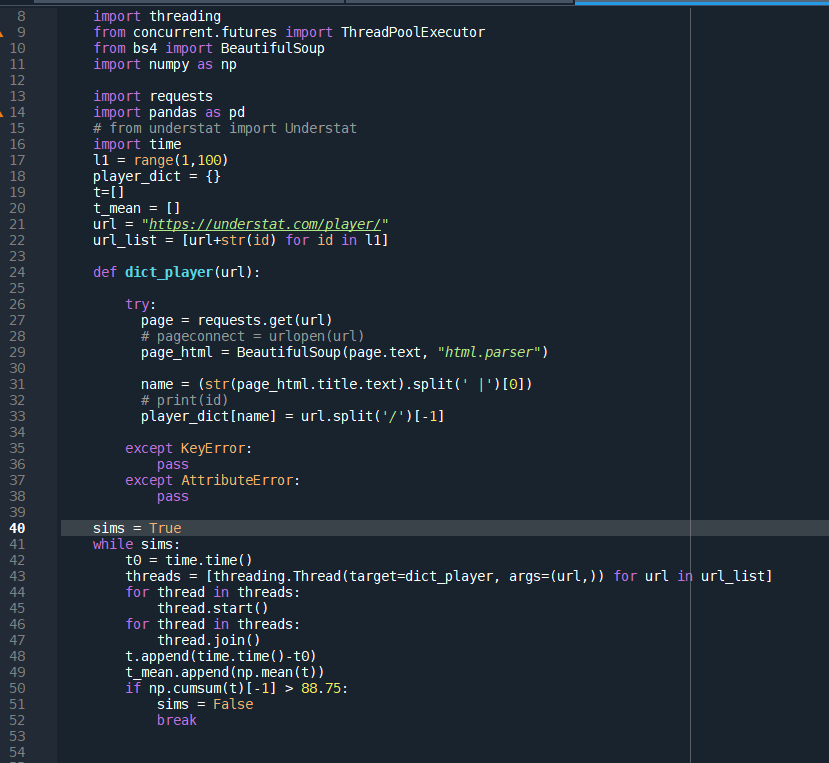
Just like multiprocessing, multithreading is also a way of achieving multitasking, by utilising the concept of threads. A **thread** is a subset of a process that is a sequence of such instructions within a program that can be executed independently of other code. **Multithreading** is, therefore, defined as the ability of a processor to execute such multiple “threads” synchronously, by making use of a technique called **context switching.** *In context switching, the state of a thread is saved and state of another thread is loaded whenever any interrupt (due to I/O or manually set) takes place. Context switching takes place so frequently that all the threads appear to be running in parallel.*

Multithreading increases code execution speed if the code is getting slowed down due to is I/O or peripheral or network bottlenecks with a significant number of interrupts and doesn’t help much in CPU heavy computation tasks. Multiprocessing increases the speed regardless of the code being CPU heavy or due to I/O or network bottlenecks, by dividing the processes into number of processors thereby multiplying the load that the CPU can handle simultaneously.

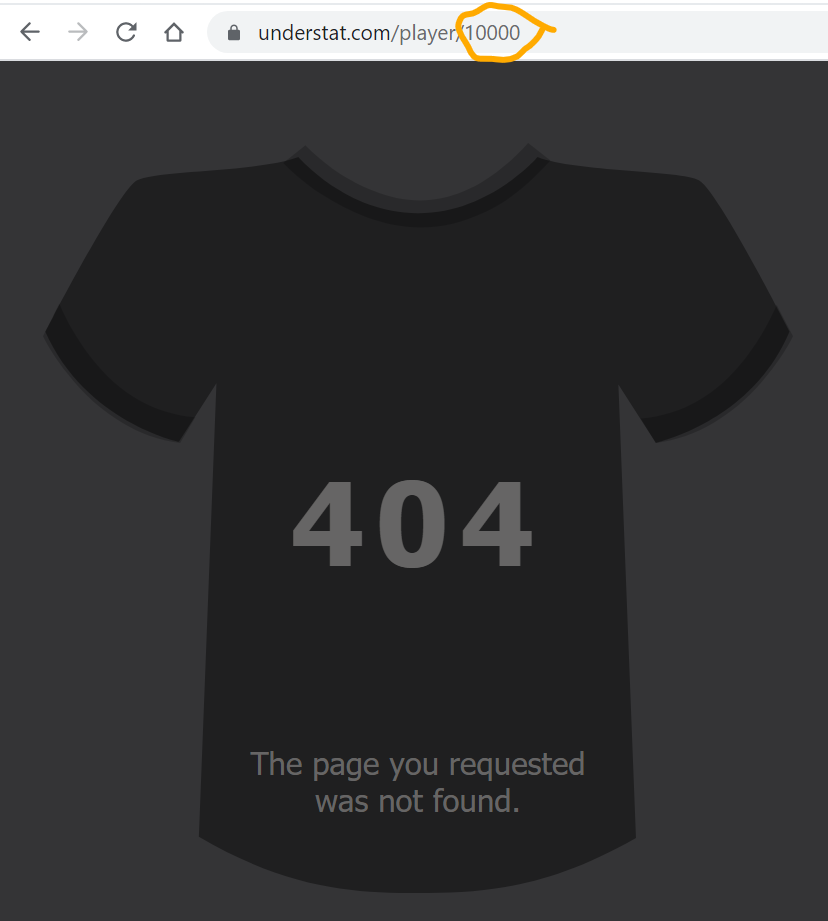
Since, the present problem is that of network interrupt and bottleneck and less of a computational heavy-load, I assumed that multithreading will be a better technique to go forward with, in order to hasten the execution of this code. Therefore, I ran the code and time its execution for multithreading.

When I simulated the entire code 100 times and timed it, first asynchronously and then by multithreading, I found that the code was significantly faster for multithreading.

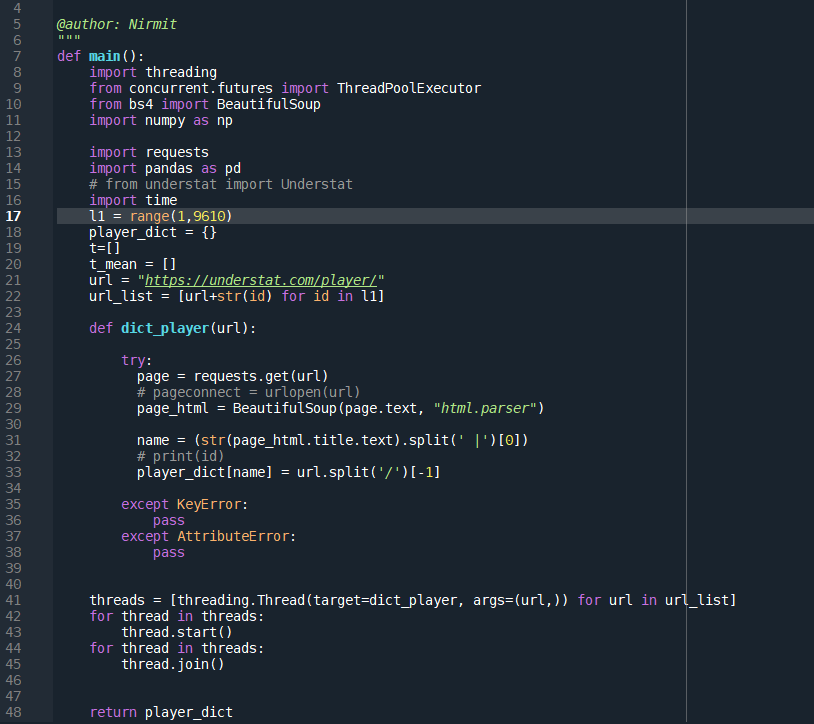
In fact, when I timed my code, I realised that, on one hand, asynchronously running my code for creating a player\_dict for 100 players was taking me 88.75 seconds (mentioned previously), while on the other hand, I was able to repeat the same process (called as simulations in the plot above) 35 times within 88.75 seconds using multithreading. This meant that my code to scrap player names-player ids dictionary from the understat site was made 35 times faster using multithreading. The moving average came out to be 2.57 seconds, i.e, it took only 2.57 seconds, on average, for the code to create the player\_dict dictionary of 100 players (Asynchronously it took 88.75 seconds, and 88.75/2.57 = 34.5- no. of times the code was faster using multithreading).



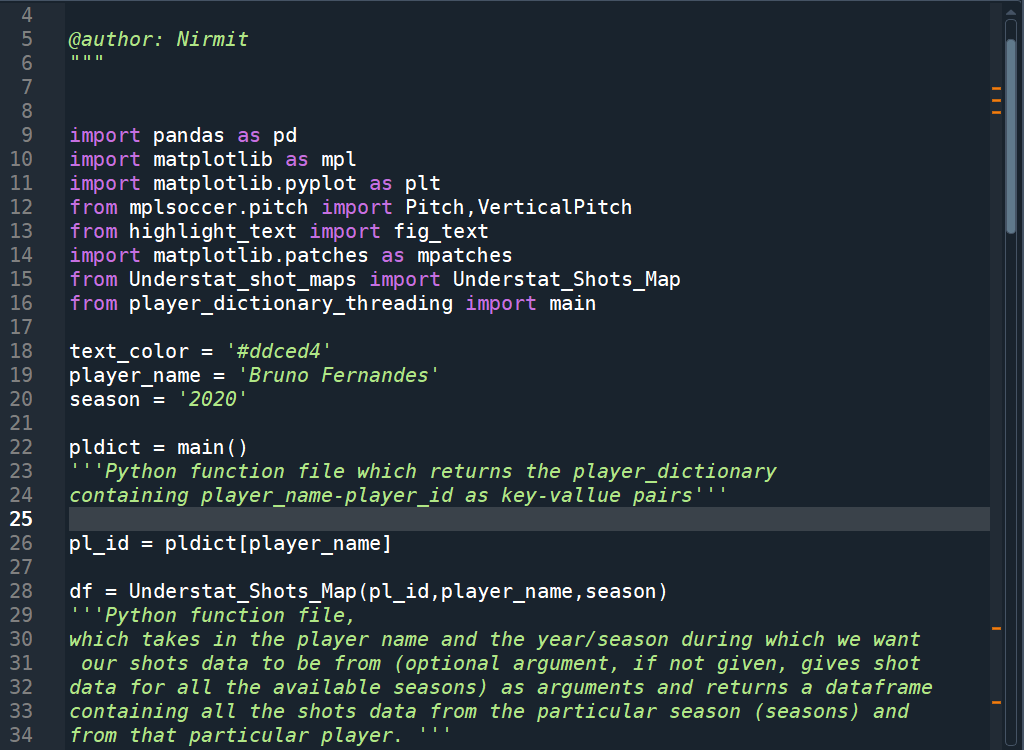
### Final Outcome

I know that the player ids start from id =1, but the last id is unknown. For example, player\_id ***1228*** gave ***Bruno*** ***Fernandes’*** stats but player\_id ***10000*** gave ***Error code: 404.*** 

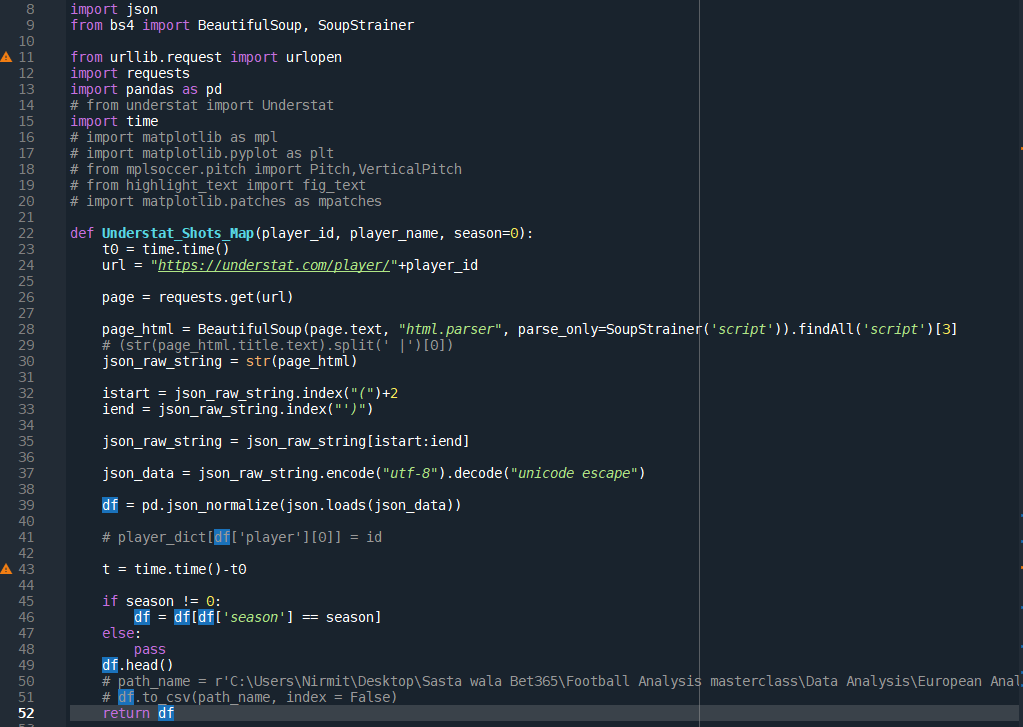
After a bit of brute-force, I eventually found out that the understat website had player data for 9609 players, and player stats webpages do not exist from player\_id 9610 onwards.



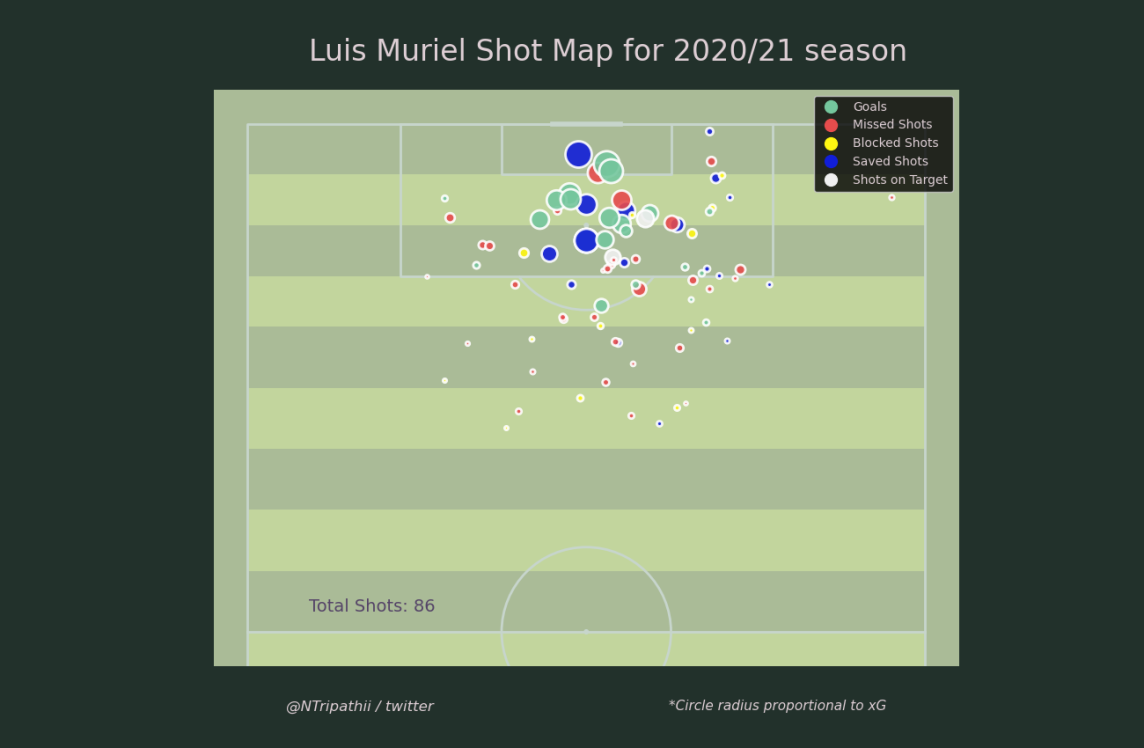
Therefore I created a separate function python file where I am creating a ***url\_list*** containing complete urls of all the players from 1 to 9609, which then is utilised in creating a complete player\_dictionary using multithreading. This ***“player\_dict”*** dictionary, is in turn, returned by the function (shown above).

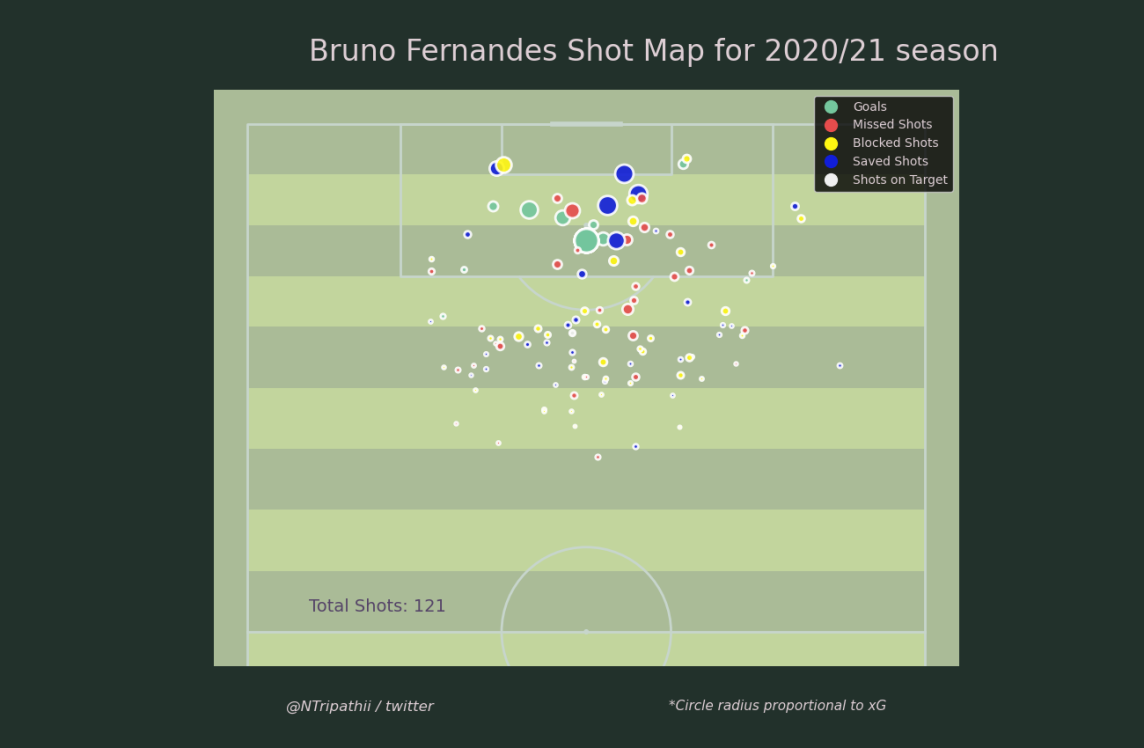


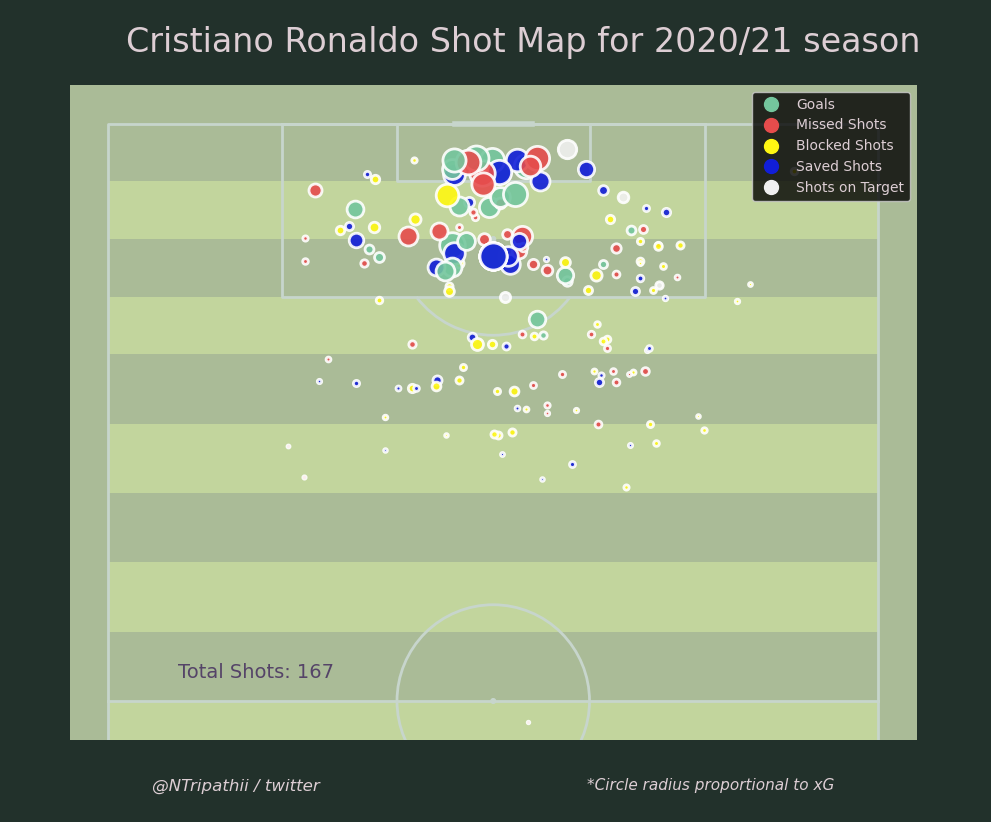
This particular function file is then imported into a main file where I am creating shot maps for player of users choice. This main file, in turn calls another function file, which takes in the player id, the player name and the year/season during which we want our shots data to be from (optional argument, if not given, gives shot data for all the available seasons) as arguments and returns a dataframe containing all the shots data from the particular season (seasons) and from that particular player. The main file which calls all the functions and creates shot maps is shown above, while the python function file which web-scrapes each player’s shots data into a dataframe and returns it to the main file (above), is shown below.

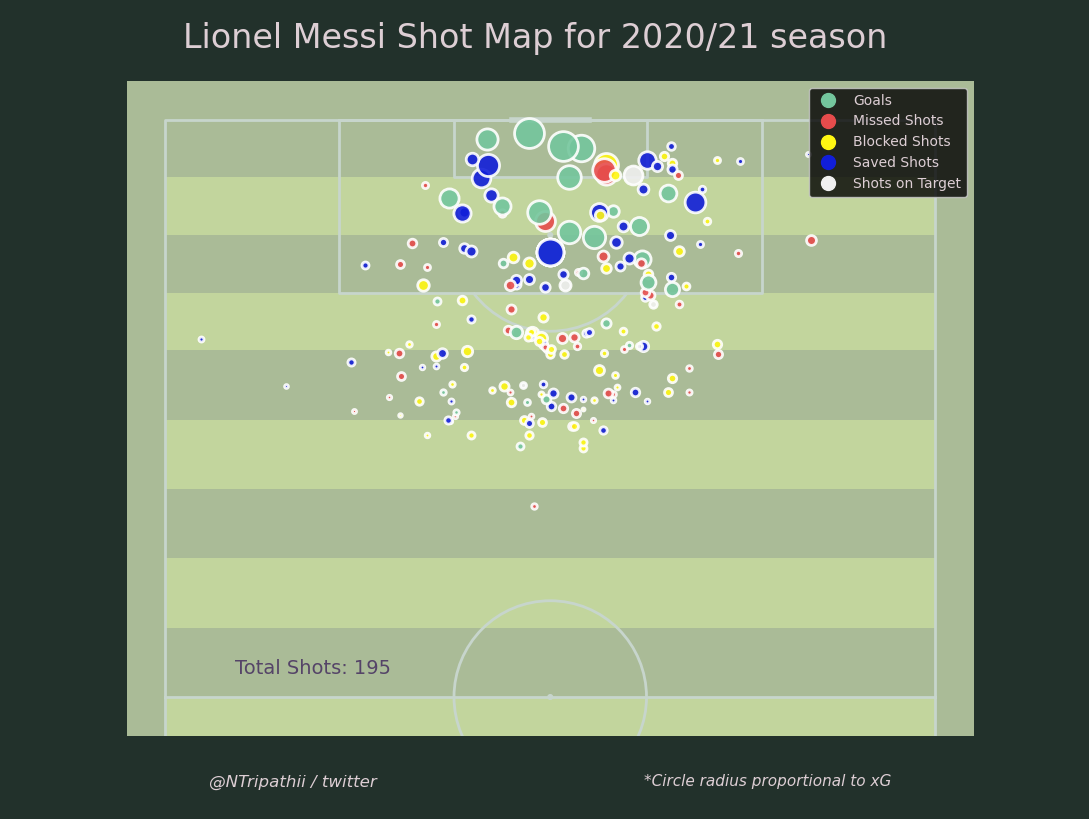


With the data thus collected and web-scraped by me, I tried visualising this particular data on matplotlib libraries. For instance, following is the plot of all the shots taken by a player (Luis Muriel, Bruno Fernandes, Cristiano Ronaldo, and Lionel Meessi) for his team in the entire season of 2020/21:









# Future Work:

In future research, I’ll be looking to build an Expected Goals from the available football literature and data, that maybe useful to people working in football.

The following questions will be studied:

* How to build an Expected Goals model, which measures the quality of shots in football games?
* How to visualize the Expected Goals model to deliver insight into what makes an effective shot?

Once the Expected Goal Model is developed, further work will be done to build models on more complex but relevant metrics like pass chains, possession chains and randomness as modelling parameters.

# References:

1. Pollard, Richard, and Charles Reep. "Measuring the effectiveness of playing strategies at soccer." Journal of the Royal Statistical Society: Series D (The Statistician) 46, no. 4 (1997): 541-550.
2. Skill and Chance in Association Football By C. REEP and B. BENJAMIN. (1968)
3. Soccermatics by David Sumpter
4. <https://github.com/aritrartira/Football-Shots-Dataset-for-xG-models/blob/main/data/Shots.csv>
5. <https://github.com/statsbomb/open-data>
6. <https://figshare.com/collections/Soccer_match_event_dataset/4415000/5>
7. <https://www.geeksforgeeks.org/multiprocessing-python-set-1/>
8. <https://www.geeksforgeeks.org/multithreading-python-set-1/>
9. <https://understat.com/>