Using Python to Scrape Soccer Scores and Solve for Optimized Team Ratings

John Knight

The University of Texas Rio Grande Valley

MATH-6340 Computing for Math and Data Science

Prof. Mike Lindstrom

December 5, 2023

Introduction

In association football ("soccer"), it is often a subject of interest to rank the set of teams in a given competition. In league competitions, each team typically plays every other team once at home and once away, meaning that teams can be fairly compared at the conclusion of the competition with every team having played the same schedule. However, when wishing to compare teams at some point *during* the season, the standings do not account for differences in the relative difficulty of opponents faced, nor for the proportion of home or away games played.

To determine the official final places in a league, teams are ranked based on points accumulated. Typically in modern soccer this is calculated as three points for each win, and one point for each draw. Alternative metrics can be used if one wishes to determine the quality of each team, perhaps for the purposes of predicting upcoming matches. A common approach is to rank teams by their goal difference (GD; goals scored minus goals conceded).

Additionally, in recent years a new football metric called "expected goals" (xG) has gained in popularity. xG calculates a value for every shot based on the distance from goal, angle, position of defenders, and other factors, regardless of whether the shot results in a goal. [1] Some consider xG to be a better reflection of a football team's ability than actual goals scored and conceded, since it removes a layer of noise related to shots being converted into goals. [2] It is less popular with traditionalists, who have described xG as "the most useless stat in the history of football". [3]

My aim is to build an application that can scrape the latest set of match results for a designated league from an online source, and use linear algebra to solve for optimized team ratings using either goals or expected goals, as dictated by the user. The program will also be able to compare the predictive performance of goals and xG by calculating ratings for every game in past seasons using an iterative process and assessing the mean absolute difference between predicted scores and actual scores.

Methods

Python 3 was chosen for this application for a couple of reasons. Firstly, because Python is the most widely used language for data science [4], increasing the likely usefulness of the program's functions for incorporation in future data science projects. Secondly, because Python's libraries are ideal for the tasks required. Below is a list of the libraries utilized (I will go into more detail on BeautifulSoup and Linprog further down):

- **Requests** – for interacting with web pages.
- **BeautifulSoup** – for scraping and parsing html.
- **Pandas** – data frames to store and manipulate the data.
- **Numpy** – matrices for linear algebra.
- **Linprog** (from scipy.optimize) – for linear programming.
- **Matplotlib** – for outputting graphs & charts.

The program begins with the `create_season` function which asks the user to choose from one of the 'big five' leagues of Europe: Premier League (England), La Liga (Spain), Bundesliga (Germany), Serie A (Italy), or Ligue 1 (France). It then asks the user to choose any season from 2017-2018 onwards, or press enter to use the current season. This choice of league and season is then returned and used to create an instance of the `Season` class, via which the core of the program is run.

Upon instantiation, the `Season` class populates its own data frame variable `self.__df` by running the `scrape_scores` function. Using the given league id and season, this passes a url into the `requests.get` function to access the popular soccer statistics website Football Reference. [5] This returns HTML content that can be parsed using the BeautifulSoup library.

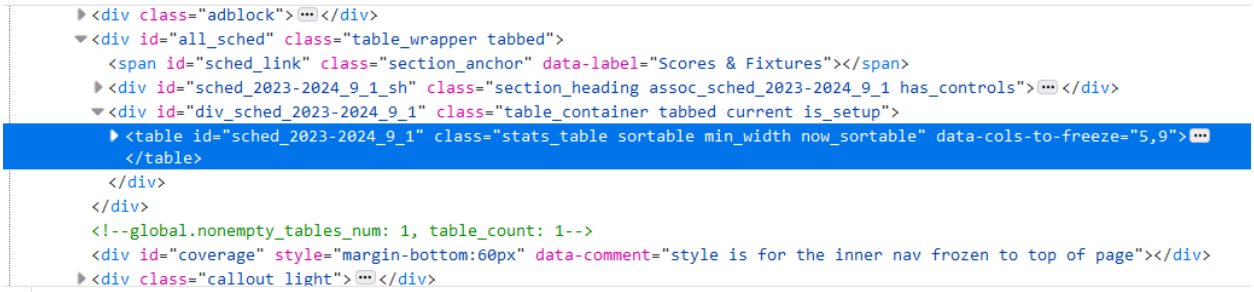

Figure 1. Inspecting the underlying HTML code for the Football Reference website.

BeautifulSoup works by converting the HTML into a 'tree' in which one can search for and extract data. Elements can be searched for based on characteristics such as their type, id, or class. In this case, the element required is a table, whose attributes can be found by right-clicking on the table in the web browser and selecting Inspect (Figure 1). The format of the Scores & Fixtures page on Football Reference is such that the desired table is always the first table found on the page. Since we only require the first table, this can be returned using the simple command `soup.find('table')` and then pandas is able to convert the table into a data frame using the `read_html` function (Figure 2).

```
r = requests.get(url) # gets the html
soup = BeautifulSoup(r.content, 'html.parser') # Converts html to BS object

table = soup.find('table') # Scores & Fixtures is always the first table

# Convert the table into a pandas DataFrame
df = pd.read_html(str(table), header=0)[0]
```

Figure 2. This code transforms a web table into a pandas data frame.

Once the `Season` data frame has been populated, the user is given two choices: Get Current Ratings, or Perform Analysis. If Get Current Ratings is chosen, the user is asked to choose from either Goals or Expected Goals, and the `solve` function calculates ratings for each team in the specified league, along with a home advantage coefficient, using the chosen metric. The fundamental idea is that each game can then be predicted using the two teams' ratings plus home advantage, as follows:

$$forecast = rating_h - rating_a + homeadv$$

The optimal solution is calculated using the linprog library from scipy.optimize. There are several steps required to prepare the data for the optimization algorithm. An initial matrix of zeros is created with m rows corresponding to each match in the dataset, and $n + 1$ columns corresponding to each unique team, plus home advantage. Then, using a `dict` that maps each team to its column index, the columns in each row corresponding to the home and away team playing in that game are set to 1 and -1 respectively. This means the home team's rating will be multiplied by 1, the away team's rating will be multiplied by -1, and teams not playing in the game will count as zero. Home advantage is set to 1 in all rows, as it applies to every game. Note that a uniform home advantage factor is applied across all games; while it would be possible to investigate whether differing levels of home advantage are appropriate for each team, the scope of this project is to create a single ranking of teams rather than separate rankings for home and away games.

To find the optimum solution, the solver aims to minimize the mean absolute difference between the forecast score and the actual scores. Since this is a linear optimization problem, and absolute values are nonlinear, the code in Figure 3 adds two identity matrices, using the `np.eye` function, to represent the positive and negative sides of the absolute values.

```
# Extend coeffs_matrix to handle absolute differences
extended_coeffs_matrix = np.hstack([coeffs_matrix, -np.eye(m), np.eye(m)])
```

Figure 3. Identity matrices are added to represent auxiliary variables.

The goal of the solver is to minimize the sum of the objective function $c^T x$, where x is the vector of ratings being optimized, and c is a vector of constants with $n + 1$ zeros and $2m$ ones, meaning only the auxiliary variables are being optimized. This is subject to the constraint $A_{eq} x = b_{eq}$ where A_{eq} is the extended coefficients matrix described above, and b_{eq} is the vector of actual match results (for example if the home team scores 2 goals and the away team scores 3 goals, the target score is -1). An additional constraint was included to set the mean of the team ratings to zero. Although the ratings only exist relative to one another within a closed system, and therefore would work the same with any mean, setting the mean to zero allows for more intuitive reading of ratings and comparison between different leagues and seasons.

Finally, the linear program is solved using the bounds seen in Figure 4. Team ratings are bound between -5 and 5, which should realistically encompass any team competing in a major soccer league. Auxiliary variables are bound to be nonnegative, according to their role. The 'highs' method proved to be most successful, as it converged in every trial, whereas the 'interior-point' and 'revised simplex' methods sometimes did not converge.

```
x_bounds = (-5, 5) # Team ratings should not be outside this range
bounds = [x_bounds] * (n + 1) + [(0, None)] * (2 * m)  # Auxiliary variables are non-negative

# Solve the linear program
result = linprog(c, A_eq=A_eq, b_eq=b_eq, bounds=bounds, method='highs')
```

Figure 4. Bounds for linear optimization program.

Once the linear program has been solved, ratings are output to the console. When the user has selected the Get Current Ratings option, this is the end of the program. However, when running the Perform Analysis option, the solve function is called many times in a loop by calling the `get_all_ratings` function. This loop solves for the ratings prior to every game in the dataset, meaning only games played prior to the date of each game are included in the optimization problem (an extra parameter `end_date` is required for this purpose). This is done for both G and xG to allow for comparison between the two metrics.

When all games in the dataset have been forecast, the function `__calc_abs_diffs` calculates the absolute difference between the forecast score in each game and the actual score. Note that goals are always used as the target variable, regardless of whether G or xG were used for the prediction. This is because goals are the 'currency' of interest that football analysts would like to predict, and it allows for comparison between the predictive value of the two metrics (if both variables are simply predicting themselves, any comparison would be meaningless).

Finally, the mean absolute differences using both G and xG are output to the console, and a chart is displayed using the matplotlib library. The chart shows the 5 game rolling mean absolute difference between the forecast scores and the actual scores, with one line for G and another for xG. This chart allows for comparison between the two metrics as a time series to see how their predictive power changes as the season progresses and sample size increases.

Results

The Get Current Ratings method proved very successful, instantly calculating ratings based on the latest data. Figure 5 shows an example of ratings for the English Premier League as of December 4, 2023, using xG as the independent variable. Newcastle Utd are rated as the best team on +1.34, and Sheffield United are rated as the worst team on -1.59. Home advantage was calculated to be worth 0.27 goals per game.

```
Here are the current ratings up to and including games on December 03, 2023 :

Newcastle Utd 1.34
Manchester City 1.21
Arsenal 0.94
Liverpool 0.77
Brentford 0.71
Chelsea 0.61
Aston Villa 0.14
Brighton 0.14
Tottenham 0.07
Everton -0.03
Manchester Utd -0.19
Wolves -0.23
Fulham -0.26
Crystal Palace -0.33
Nott'ham Forest -0.36
West Ham -0.39
Bournemouth -0.46
Burnley -0.79
Luton Town -1.29
Sheffield Utd -1.59

Home advantage: 0.27
```

Figure 5. Output of ratings for the Premier League on December 4, 2023. Note that the output of December 3 reflects the last game played, not the current date.

The Perform Analysis process was run for each of the 30 eligible seasons (6 seasons, from 2017-2018 onwards, for the 5 different leagues). Table 1 shows a summary of the mean absolute difference, and the frequency at which each metric proved to be the better predictor. It turns out that xG was superior in 29 of the 30 trials, with an overall mean absolute difference 0.12 lower than goals. An expanded version can be found in Table 2 in the appendix.

Table 1. Comparison of the predictive performance goals and expected goals over 30 seasons.

Metric	Better Predictor	Mean Absolute Difference
Goals	1	1.60
Expected Goals	29	1.48

Figure 6 shows an example of the plot output during the Perform Analysis process. It plots the 5 game rolling mean absolute difference, in this case for France's Ligue 1 in 2021-2022. While the graph does convey the information intended, the natural variance of week-to-week results creates a lot of noise which renders the graphs difficult to draw inference from.

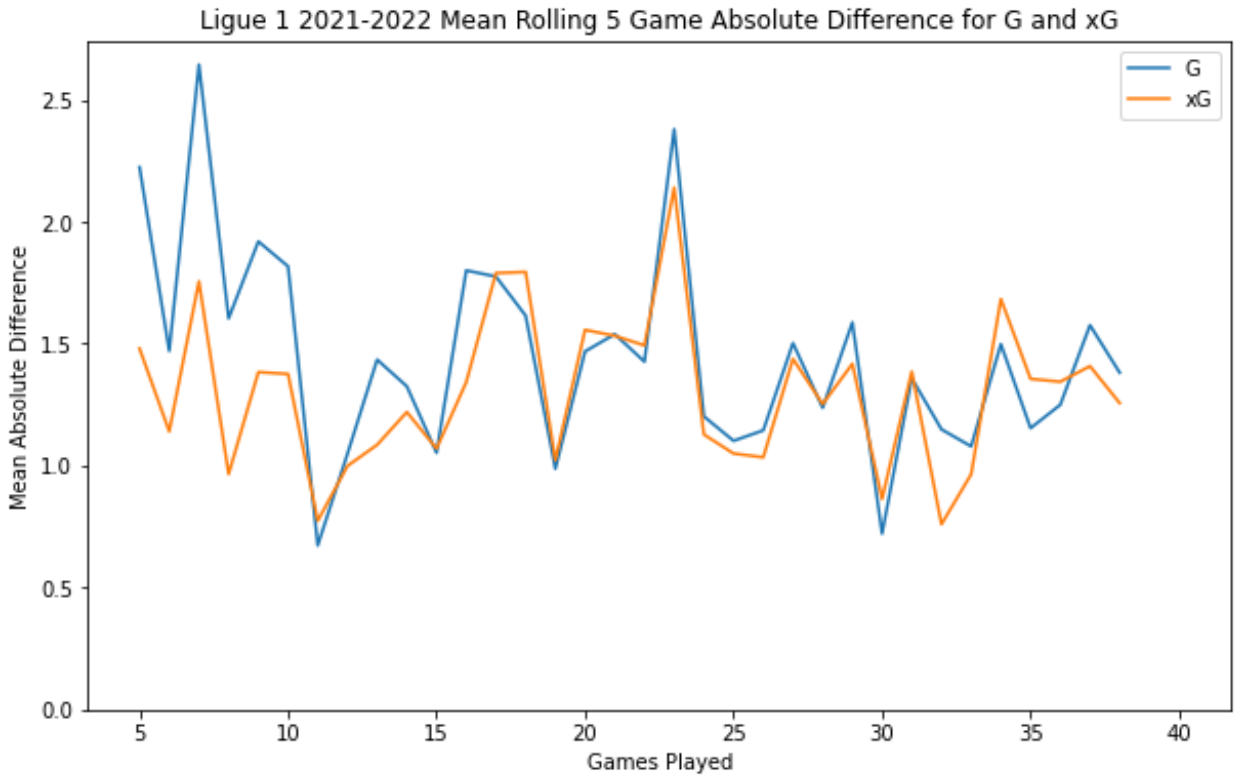

Figure 6. Example of the graphical output in the program's Perform Analysis process.

Discussion

Since this program provides a method to quickly calculate team ratings over a large number of games, it could provide a framework to form ratings that could be incorporated as a feature for learning algorithms, such as artificial neural networks. The results of this study would suggest that expected goals are a more useful metric than goals for predicting future match outcomes.

There is much potential for development to build on the framework of this program. For example, there are other metrics available on Football Reference, such as the number of touches in the penalty area, that could easily be used in this program in place of G/xG, and tested to judge their efficacy. Also, it would be simple to apply the same principles to any other sport involving a closed system of teams contesting matches with scores, such as American football, basketball, or baseball.

The graph output could be developed to provide more meaningful information. It may be the case that the amount of variance in the results of a single league is too high to judge the relative performance of a metric as a time series throughout the weeks of the season. Perhaps by combining many seasons and graphing them together, the increased sample size would reduce variance and provide a more interesting visualization.

Appendix

References

[1] Whitmore, J. (2023, November 10). *What is expected goals (XG)?*. Opta Analyst. https://theanalyst.com/na/2023/08/what-is-expected-goals-xg/

[2] Octosport.io. (2022, April 15). *Expected goals: Can they predict future goals?*. Geek Culture. https://medium.com/geekculture/expected-goals-can-they-predict-future-goals-24a1b1d0279d

[3] MacInnes, P. (2017, November 22). *Never mind Jeff Stelling's derision, expected goals – XG – is here to stay*. The Guardian. https://www.theguardian.com/football/blog/2017/nov/22/jeff-stelling-expected-goals-stats-xg-soccer-am

[4] Canales Luna, J. (2023, March 10). *Top programming languages for data scientists in 2023*. datacamp.com. https://www.datacamp.com/blog/top-programming-languages-for-data-scientists-in-2022

[5] Pro Football Reference. Sports Reference. https://www.pro-football-reference.com/

Tables

Table 2. Expanded version of Table 1, showing a comparison of the predictive performance goals and expected goals over 30 seasons.

League	Season	G	xG	Superior
England	2017-2018	1.757	1.541	xG
England	2018-2019	1.604	1.481	xG
England	2019-2020	1.716	1.464	xG
England	2020-2021	1.661	1.482	xG
England	2021-2022	1.649	1.56	xG
England	2022-2023	1.607	1.564	xG
Spain	2017-2018	1.63	1.532	xG
Spain	2018-2019	1.409	1.306	xG
Spain	2019-2020	1.441	1.31	xG
Spain	2020-2021	1.392	1.283	xG
Spain	2021-2022	1.392	1.429	G
Spain	2022-2023	1.463	1.341	xG
Germany	2017-2018	1.577	1.497	xG
Germany	2018-2019	1.829	1.816	xG
Germany	2019-2020	1.959	1.809	xG
Germany	2020-2021	1.672	1.539	xG
Germany	2021-2022	1.917	1.73	xG

Germany	2022-2023	1.911	1.777	xG
Italy	2017-2018	1.512	1.405	xG
Italy	2018-2019	1.437	1.304	xG
Italy	2019-2020	1.658	1.485	xG
Italy	2020-2021	1.563	1.348	xG
Italy	2021-2022	1.534	1.394	xG
Italy	2022-2023	1.506	1.483	xG
France	2017-2018	1.625	1.519	xG
France	2018-2019	1.485	1.392	xG
France	2019-2020	1.569	1.488	xG
France	2020-2021	1.623	1.473	xG
France	2021-2022	1.516	1.393	xG
France	2022-2023	1.429	1.398	xG

Code

https://github.com/capybara2/Soccer-Solver