Web Scraper For Extracting English Premier League Football Players' Data

Adedamola Adesoye
College of Engineering
Northeastern University
Toronto, ON
adesoye.a@northeastern.edu

Abstract

In this report, I explained the details of my web scraping endeavor where I extracted data of interest for every English Premier League player from multiple FBREF webpages. I cleaned the data by casting the data types of specific columns to their appropriate data types. Finally, I saved the files in CSV format.

1 Introduction

In this project, I plan to scrape data about every English Premier League (EPL) team's player. Every player's inclusion in the final dataset means they have featured, at least once, in their team's matchday squad in the current EPL season. Data extracted for outfield players (i.e. non-goalkeepers) will be different from data extracted for goalkeepers since their performances are largely judged based on different statistics. There are 20 EPL teams.

1.1 Problem definition

The project is undertaken with the following scenario in mind: a Fantasy Premier League (FPL) [1] participant wants datasets that they can manipulate to inform them on which players are most likely to produce the most FPL points in the next game week.
The result of this project will be two datasets (one for outfield players, the other for goalkeepers) containing each player's data. The data for each player will be collected with the consideration of what statistics are best for evaluating a player's performance; the data collected for goalkeepers will be different from data collected for outfield players.
Although applying this differentiation to every outfield position (i.e. defender, midfielder, striker) will help to better evaluate each outfield player's performance, this is out of the scope of this project. Other data can also be collected (or left out of) for the final dataset to improve the quality of decision-making. This is also out of the scope of this project.

1.2 Motivation

I am a football fan, so I picked this domain for this web scraping project to make the project more interesting. Also, increasing my level of interest makes it more likely that I will expand the project in the future.

2 Methodology

I built the web scraper for this project in a Jupyter Notebook and set up a virtual environment for the project. The required libraries are listed in the *requirements.txt* file zipped with the Notebook. Python was used as my programming language.

3 Solution

After importing the libraries necessary for the successful completion of the project, I sent a

GET request to return the HTML file for the desired webpage [2] containing all EPL teams' data.

```python
base_url = 'https://fbref.com'
html = requests.get(f'{base_url}/en/comps/9/Premier-League-Stats') # getting the html file of the url's webpage
bs = BeautifulSoup(html.text, 'html.parser') # parsing the returned file as html
```

Figure 1: Code for requesting webpage and parsing the file as HTML

I picked this webpage as the starting point of my scraping endeavours because it contains the URL of the webpage for each EPL team which serves as the repository for every team's players' statistics.

Rk	Squad	MP	W	D	L	GF	GA	GD	Pts	Pts/MP	xG	xGA	xGD	xGD/90	Last 5
1	Liverpool	21	14	6	1	47	18	+29	48	2.29	45.7	24.0	+21.7	+1.03	D D W W W
2	Manchester City	20	13	4	3	48	23	+25	43	2.15	39.7	19.1	+20.6	+1.03	W D W W W
3	Arsenal	21	13	4	4	42	20	+22	43	2.05	39.4	16.7	+22.7	+1.08	W D L L W
4	Aston Villa	21	13	4	4	43	27	+16	43	2.05	37.2	26.7	+10.4	+0.50	W D L W D
5	Tottenham	21	12	4	5	44	31	+13	40	1.90	36.4	35.5	+0.9	+0.04	W W L W D
6	West Ham	21	10	5	6	35	32	+3	35	1.67	29.9	37.7	-7.8	-0.37	W W W D D
7	Brighton	21	8	8	5	38	33	+5	32	1.52	35.9	30.7	+5.1	+0.24	L D W D D
8	Manchester Utd	21	10	2	9	24	29	-5	32	1.52	29.9	33.7	-3.8	-0.18	D L W L D
9	Chelsea	21	9	4	8	35	31	+4	31	1.48	41.6	29.7	+11.9	+0.57	W L W W W
10	Newcastle Utd	21	9	2	10	41	32	+9	29	1.38	40.5	36.5	+4.1	+0.19	W L L L L
11	Wolves	21	8	5	8	30	31	-1	29	1.38	27.7	34.1	-6.3	-0.30	L W W W D
12	Bournemouth	20	7	4	9	28	39	-11	25	1.25	28.4	32.9	-4.5	-0.22	W W W L L
13	Fulham	21	7	3	11	28	36	-8	24	1.14	25.2	36.5	-11.3	-0.54	L L L W L
14	Brentford	20	6	4	10	29	33	-4	22	1.10	33.8	26.5	+7.3	+0.37	L L L L W
15	Crystal Palace	21	5	6	10	22	34	-12	21	1.00	24.9	31.6	-6.7	-0.32	D D L W L
16	Nott'ham Forest	21	5	5	11	26	38	-12	20	0.95	25.0	31.9	-6.9	-0.33	L L W W L
17	Everton	21	8	3	10	24	28	-4	17	0.81	31.2	28.9	+2.3	+0.11	W L L L D
▼ 18	Luton Town	20	4	4	12	24	38	-14	16	0.80	20.2	39.6	-19.5	-0.97	L W W L D
▼ 19	Burnley	21	3	3	15	21	42	-21	12	0.57	18.6	35.4	-16.9	-0.80	L W L L D
▼ 20	Sheffield Utd	21	2	4	15	17	51	-34	10	0.48	18.7	42.1	-23.4	-1.12	L D L L D

Figure 2: Snippet of the contents of scraper's first target webpage

As you may have guessed, the names of the teams are hyperlinked, each linked to their respective URL, as shown in Figure 2. The paths to be extracted are embedded in anchor tags as shown in the Figure 3.

```html
<a href="/en/squads/822bd0ba/Liverpool-Stats">Liverpool</a> == $0
```

Figure 3: Anchor HTML tag for Liverpool text shown in Figure 2

To extract the path, I used the code (shown in Figure 4) to find and extract paths with *href* values matching the specified regular expression (regex).

```python
team_a_tag = bs.find_all('a', {'href':re.compile('\/en\/squads\/[a-z0-9]*\/[A-Za-z\-]*\-Stats')})[:20]

# concatenate the base url with the path of each team
team_urls = [f"{base_url}{a_tag['href']}" for a_tag in team_a_tag]
```

Figure 4 Code to extract each team's URL and append it to the created team_urls list

I chose the specified regex to uniquely identify the desired paths because there were other (undesired) elements with anchor tags on the webpage. The regex was defined with the observed common path pattern in mind: starts with '/en/squads/' followed by (any number of) lowercase string characters and/or numbers followed by '/' followed by (any number of) upper and/or lowercase string characters followed by '-Stats'.

I selected only the first 20 elements because more (undesired) elements were matching that regex. After this, I created a list containing the base URL concatenated with each path.

```
team_urls[:5]
```

```
['https://fbref.com/en/squads/822bd0ba/Liverpool-Stats',
 'https://fbref.com/en/squads/b8fd03ef/Manchester-City-Stats',
 'https://fbref.com/en/squads/18bb7c10/Arsenal-Stats',
 'https://fbref.com/en/squads/8602292d/Aston-Villa-Stats',
 'https://fbref.com/en/squads/361ca564/Tottenham-Hotspur-Stats']
```

Figure 5: First five items in the team_urls list

With this list at hand, I can loop through it and collect the players' data for each team stored on their webpage. The data of interest (DOI) are stored in five different tables (e.g.: Liverpool's players' shooting data stored in one of the tables of interest as shown in Figure 6).

Shooting 2023-2024 Liverpool: Premier League   Glossary   Toggle Per90 Stats

					Standard												Expected					
Player	Nation	Pos	Age	90s	Gls	Sh	SoT	SoT%	Sh/90	SoT/90	G/Sh	G/SoT	Dist	FK	PK	PKatt	xG	npxG	npxG/Sh	G-xG	np:G-xG	Matches
Mohamed Salah	EGY	FW	31-227	19.3	14	59	26	44.1	3.05	1.35	0.17	0.38	16.6	1	4	6	14.3	9.4	0.16	-0.3	+0.6	Matches
Alisson	BRA	GK	31-118	19.0	0	0	0		0.00	0.00				0	0	0	0.0	0.0		0.0	0.0	Matches
Virgil van Dijk	NED	DF	32-204	18.3	1	22	7	31.8	1.20	0.38	0.05	0.14	13.1	1	0	0	1.6	1.6	0.07	-0.6	-0.6	Matches
Dominik Szoboszlai	HUN	MF	23-095	17.7	2	38	9	23.7	2.15	0.51	0.05	0.22	26.0	4	0	0	1.8	1.8	0.05	+0.2	+0.2	Matches
Trent Alexander-Arnold	ENG	DF	25-113	16.6	2	29	4	13.8	1.74	0.24	0.07	0.50	24.2	3	0	0	1.8	1.5	0.05	+0.2	+0.5	Matches
Luis Díaz	COL	FW	27-015	13.5	3	36	12	33.3	2.66	0.89	0.08	0.25	16.1	0	0	0	3.9	3.9	0.11	-0.9	-0.9	Matches
Alexis Mac Allister	ARG	MF	25-035	13.2	1	9	3	33.3	0.68	0.23	0.11	0.33	29.0	0	0	0	0.2	0.2	0.02	+0.8	+0.8	Matches
Darwin Núñez	URU	FW	24-218	13.4	7	61	27	44.3	4.55	2.01	0.11	0.26	14.8	0	0	0	9.4	9.4	0.15	-2.4	-2.4	Matches
Ibrahima Konaté	FRA	DF	24-248	10.8	0	8	2	25.0	0.74	0.18	0.00	0.00	12.5	0	0	0	0.4	0.4	0.05	-0.4	-0.4	Matches
Joe Gomez	ENG	DF	26-250	11.4	0	12	1	8.3	1.06	0.09	0.00	0.00	17.9	0	0	0	0.5	0.5	0.04	-0.5	-0.5	Matches
Cody Gakpo	NED	FW,MF	24-266	9.3	3	26	10	38.5	2.78	1.07	0.12	0.30	15.7	0	0	0	4.4	4.4	0.17	-1.4	-1.4	Matches
Joël Matip	CMR	DF	32-173	8.7	0	5	0	0.0	0.58	0.00	0.00		10.6	0	0	0	0.5	0.5	0.10	-0.5	-0.5	Matches
Diogo Jota	POR	FW,MF	27-055	8.4	7	26	11	42.3	3.09	1.31	0.27	0.64	12.4	0	0	0	3.3	3.3	0.13	+3.7	+3.7	Matches
Andrew Robertson	SCO	DF	29-323	8.0	1	2	2	100.0	0.25	0.25	0.50	0.50	8.2	0	0	0	0.5	0.5	0.23	+0.5	+0.5	Matches
Wataru Endo	JPN	MF	30-353	7.8	1	6	2	33.3	0.77	0.26	0.17	0.50	22.7	0	0	0	0.3	0.3	0.05	+0.7	+0.7	Matches
Kostas Tsimikas	GRE	DF	27-261	7.4	0	2	0	0.0	0.27	0.00	0.00		27.1	0	0	0	0.1	0.1	0.03	-0.1	-0.1	Matches
Curtis Jones	ENG	MF	22-363	7.3	1	12	3	25.0	1.63	0.41	0.08	0.33	16.5	0	0	0	1.7	1.7	0.14	-0.7	-0.7	Matches
Ryan Gravenberch	NED	MF	21-257	7.3	0	13	5	38.5	1.79	0.69	0.00	0.00	19.6	0	0	0	1.4	1.4	0.11	-1.4	-1.4	Matches
Harvey Elliott	ENG	MF,FW	20-299	5.3	1	18	5	27.8	3.41	0.95	0.06	0.20	23.7	0	0	0	0.7	0.7	0.04	+0.3	+0.3	Matches
Jarell Quansah	ENG	DF	20-364	2.9	0	1	0	0.0	0.34	0.00	0.00		10.1	0	0	0	0.1	0.1	0.06	-0.1	-0.1	Matches
Caoimhín Kelleher	IRL	GK	25-066	2.0	0	0	0		0.00	0.00				0	0	0	0.0	0.0		0.0	0.0	Matches
Conor Bradley	NIR	DF	20-203	0.9	0	3	1	33.3	3.29	1.10	0.00	0.00	15.5	0	0	0	0.3	0.3	0.09	-0.3	-0.3	Matches
Ben Doak	SCO	FW	18-078	0.2	0	0	0		0.00	0.00				0	0	0	0.0	0.0		0.0	0.0	Matches
Owen Beck	WAL	DF	21-172	0.1	0	0	0		0.00	0.00				0	0	0	0.0	0.0		0.0	0.0	Matches
Bobby Clark	ENG	MF	18-355	0.1	0	0	0		0.00	0.00				0	0	0	0.0	0.0		0.0	0.0	Matches
Kaide Gordon	ENG	FW	19-115	0.0	0	0	0		0.00	0.00				0	0	0	0.0	0.0		0.0	0.0	Matches
James McConnell	ENG	MF	19-137	0.0	0	0	0		0.00	0.00				0	0	0	0.0	0.0		0.0	0.0	Matches
Squad Total			**27.2**	**21.0**	**44**	**388**	**130**	**33.5**	**18.48**	**6.19**	**0.10**	**0.31**	**17.9**	**9**	**4**	**6**	**45.7**	**40.7**	**0.11**	**-1.7**	**-0.7**	
Opponent Total			**26.8**	**21.0**	**17**	**236**	**68**	**28.8**	**11.24**	**3.24**	**0.07**	**0.24**	**17.4**	**9**	**1**	**1**	**24.0**	**23.2**	**0.10**	**-7.0**	**-7.2**	

Figure 6: Liverpool's players' shooting data stored in one of the tables of interest

To extract the DOI from each table, I defined five different functions. For example, the function shown in Figure 7 is responsible for extracting the DOI from one row in the table shown in Figure 6.

```python
def get_player_shoot_stat(row, club_name):
    position = row.find('td', {'data-stat':'position'}).get_text()
    if position != 'GK':
        name = row.find('th').get_text()
        age = row.find('td', {'data-stat':'age'}).get_text()
        position = row.find('td', {'data-stat':'position'}).get_text()
        minutes_90s = row.find('td', {'data-stat':'minutes_90s'}).get_text()
        goals = row.find('td', {'data-stat':'goals'}).get_text()
        pens_made = row.find('td', {'data-stat':'pens_made'}).get_text()
        xg = row.find('td', {'data-stat':'xg'}).get_text()
        npxg = row.find('td', {'data-stat':'npxg'}).get_text()
        return [age, name, position, club_name, minutes_90s, goals, pens_made, xg, npxg]
```

Figure 7: Function for extracting the DOI from one row in the table shown in Figure 6

The function takes in a *tr* tag and extracts the DOI from respective *children*. I filtered out rows containing goalkeepers (*position* equal to *GK*) because the data stored in this table are not ideal for evaluating a goalkeeper's performance. The function was defined to return a list. Each list will serve as a row in the final dataset. The lists were initially stored in a dictionary (as shown in Figure 8) with the *age* (in *years-days*) and player *name* serving as unique IDs for each player.

```
outfield_players
```

```
defaultdict(list,
            {('31-227', 'Mohamed Salah'): ['Mohamed Salah',
              'FW',
              'Liverpool',
              '19.3',
              '14',
              '4',
              '14.3',
              '9.4',
              '8',
              '8.5',
              '88'],
             ('32-204', 'Virgil van Dijk'): ['Virgil van Dijk',
              'DF',
              'Liverpool',
              '18.3',
              '1',
              '0',
```

Figure 8: Resulting dictionary after extracting DOI for outfield players

This was designed with the assumption that no two players in the EPL have the same name and same age. I save the list as values of keys in a dictionary (instead of items in a list) because I still have the intention of extracting other data for each player from different tables and I need a unique ID to indicate which list to append the incoming data. The tables' rows are all ordered according to the *90s* column so a simpler alternative is to append the new data according to the index of the row it is coming from (i.e. Mohammed Salah's data is stored in the first row of every outfield players' table of interest). I didn't take this approach because I realized this as I wrote this report and after I finished the project.

As shown in Figure 7, *club_name* was passed into the function. It is the name of the team's players' data we are scraping, and it was extracted before the function call.

The club's name was extracted from the heading of the web page (sample shown in Figure 9).

2023-2024 Liverpool Stats

Figure 9: Sample of heading of the webpage where club_name was extracted from

I manipulated the data with the code shown in Figure 10 to extract each team’s name. It wasn't extracted from the starting web page because not all the names were written in full.

```
club_name = ' '.join(bs.find('div', {'data-template': 'Partials/Teams/Summary'}).find('span').get_text().split()[1:-1])
```

Figure 10: Data to extract each team's name

I stored the outfield players’ and goalkeepers’ data in *outfield_players* and *gk_s* respectively.

```
outfield_players = defaultdict(list)
gk_s = defaultdict(list)
```

Figure 11: Initial variables pointing to the recently extracted data

These variables were initialized as *defaultdict* objects instead of the more commonly used *dict* to avoid getting errors when accessing non-existent keys in the dictionary.
As planned, I looped through the list of team URLs as shown in Figure 12.

```
for url in team_urls:
    html = requests.get(url)
    bs = BeautifulSoup(html.text, 'html.parser')
```

Figure 12: Code for looping through each URL in team_urls

I extracted and stored rows from the five different tables of interest (TOI) in five different variables (sample shown in Figure 13).

```
shoot_stat_rows = bs.find('table', {'id':'stats_shooting_9'}).find('tbody').find_all('tr')
```

Figure 13: Variable pointing to the rows extracted from the table shown in Figure 6

Then I looped through each row in each variable, passed them into their respective functions and stored the returned data in the appropriate dictionary (sample shown in Figure 14).

```
for row in shoot_stat_rows:
    player_stats = get_player_shoot_stat(row, club_name)
    if player_stats:
        outfield_players[player_stats[0], player_stats[1]] = player_stats[1:]
```

Figure 14: Code that calls function shown in Figure 7 on each row stored in the variable shown in Figure 13

As shown in Figure 14, I neglected the *age* in the list because its only purpose was to collaborate in uniquely identifying each player.
After extracting the data for each team, I integrated a four-second delay. This was designed to evade the penalty put in place by Sports Reference (owner of FBREF) to minimize scraping activities on their sites. They limit users to 20 requests per minute [3]. Exceeding this limit will result in placing a user's session in jail for one hour. Hence, the integration of the four-second delay (max. of about 15 requests per minute).
After successfully extracting the data for each team, I extracted the lists (values of the dictionaries) and used *pandas* to convert them to data frames.

```
outfield_df.head()
```

	name	position	club_name	minutes_90s	goals	pens_made	xg	npxg	assists	xg_assist	sca
0	Mohamed Salah	FW	Liverpool	19.3	14	4	14.3	9.4	8	8.5	88
1	Virgil van Dijk	DF	Liverpool	18.3	1	0	1.6	1.6	2	1.1	30
2	Dominik Szoboszlai	MF	Liverpool	17.7	2	0	1.8	1.8	2	4.2	73
3	Trent Alexander-Arnold	DF	Liverpool	16.6	2	0	1.8	1.5	3	5.1	91
4	Luis Díaz	FW	Liverpool	13.5	3	0	3.9	3.9	1	1.6	64

Figure 15: First five rows of outfield players' data frame

I also cast the numeric values to the appropriate data type (int or float) to aid future mathematical operations to be carried out on the respective columns.
I saved the data on my local machine with a file name that has a suffix containing the date the file was saved. This was done to allow the user to keep track of the changes in the data over time.

```
outfield_df.to_csv(f"outfield_players_data {datetime.now().date().strftime('%d-%m-%Y')}.csv", index=False)
```

Figure 16: Saving outfield dataset in CSV format

For pleasure's sake, I grouped the data for the outfield players by *club_name*, computed the sum of their goals scored and expected goals and plotted them on the x and y axes respectively.

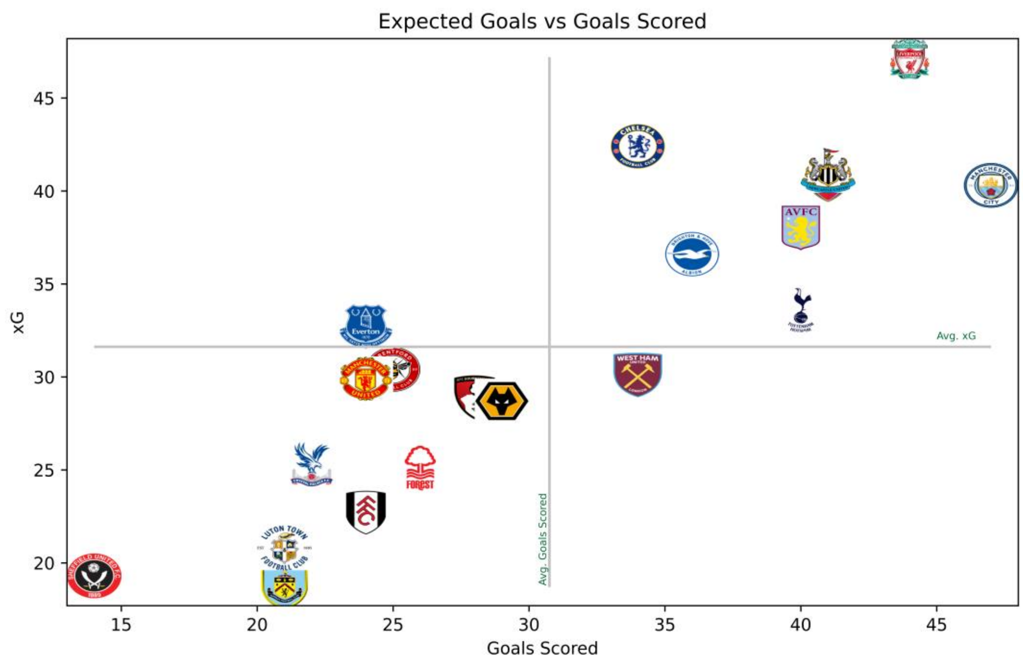

Figure 17: Expected Goals vs Goals Scored

The centre of each team's logo represents the location of their data point. From the plot we gather insights like, Chelsea has scored significantly fewer goals than they were expected to, Manchester City has scored at least five more goals than they were expected to, etc.

4 Challenges and Outlook

4.1 Challenges

Some of the challenges I faced were the issue of debugging my code and coming up with the code to bring an idea to fruition. To combat this, I employed OpenAI's ChatGPT to speed up my debugging and code creation efforts. Another challenge I faced was limiting the scope of the project. As a huge football fan, naturally, I was tempted to go way out of the scope of the requirements. I considered adding more features to this project, e.g. researching what statistics are most relevant for each player in better evaluating their performance, configuring the notebook to run every time the data on the website is refreshed, etc. I resolved this issue by remaining disciplined, which allowed me to satisfy the requirements of the project in due time.

4.2 Outlook

I could extend this project in the future to extract more appropriate data for better evaluating each player's performance. I could then use this data to train models that provide recommendations on what players to include in a squad for an upcoming FPL game week. The data provided by Sports Reference can also be used to train models for football betting predictions.

Acknowledgments

The source of all the scraped data is https://fbref.com. I consulted Oreilly's Web Scraping Book [4] to enhance my understanding of web scraping practices. I employed OpenAI's ChatGPT for faster debugging and code generation. I adapted FCPython's [5] code to produce the scatter plot.

References

[1] The Scout. (2023, July 5). *FPL Basics: What is FPL and how to play*. Premier League. https://www.premierleague.com/news/2173986

[2] *Premier League stats*. FBref.com. (n.d.). https://fbref.com/en/comps/9/Premier-League-Stats

[3] *Bot/scraping/crawler traffic on sports-reference.com sites*. Sports Reference. (2022, October 26). https://www.sports-reference.com/bot-traffic.html

[4] MITCHELL, R. (2024). *Web scraping with python: Data extraction from the modern web*. O'REILLY MEDIA.

[5] FCPythonADMIN. (2021, September 29). *Creating scatter plots with Club Badges in python*. FC Python. https://fcpython.com/visualisation/creating-scatter-plots-with-club-logos-in-python