Photo by Michael Lee on Unsplash

Web Scraping Advanced Football Statistics

Collecting data is always fun

Sergi Lehkyi Jun 23, 2019 · 9 min read

Lately I've being debating about the role of luck in football. Or maybe it's not pure luck, but a skill? Do teams win their leagues purely on skill? Or the importance of luck is quite huge? Who is lucky and who is not? Did that team deserve the relegation? And many-many more.

But as I am a data guy, I thought, so let's get the data and find that out. Although, how do you measure luck? How do you measure skill? There is no such a single metric like

played. Because in some moments the player of one team just don't have enough luck to score the winning goal on last minutes after total domination over the opponent and it ends up as equalizer and both teams get 1 point, while it was clear that the first team deserved the victory. One team was lucky, another wasn't. Yes, in this situation it's a luck, because one team did everything, created enough dangerous moments, but didn't score. It happens. And that's why we love football. Because everything can happen here.

Although you cannot measure luck, but you get an understanding of how the team played based on a relatively new metric in football — xG, or expected goals.

> xG — is a statistical measure of the quality of chances created and conceded

You can find the data with this metric at understat.com. This is the web I am about to scrap.

Understanding the data

So what the heck is xG and why is it important. Answer we can find at understat.com home page.

> Expected goals (xG) is the new revolutionary football metric, which allows you to evaluate team and player performance.
>
> In a low-scoring game such as football, final match score does not provide a clear picture of performance.
>
> This is why more and more sports analytics turn to the advanced models like xG, which is a statistical

> conceded.
>
> Our goal was to create the most precise method for shot quality evaluation.
>
> For this case, we trained neural network prediction algorithms with the large dataset (>100,000 shots, over 10 parameters for each).

The researchers trained neural network based on situations that led to goals and now it gives us an estimation of how many real chances did the team have during the match. Because you can have 25 shots during the game, but if they all are from long distance or from low angle or too weak, shorter — low quality shots, they won't lead to the goal. While some "experts" that didn't see the game will say that the team dominated, created tons of chances bla-bla-bla-bla. Quality of those chances matters. And that's where the xG metric becomes very handy. With this metric you now understand that Messi creates goals in conditions where it's very hard to score, or the goalkeeper makes save where it should've being goal. All these things add up and we see champions that have skilled players and some luck and we see losers that might have good players, but don't have enough luck. And my intent with this project is to understand and present these numbers to show the role of luck in today's football.

Let's begin

We start by importing libraries that will be used in this project:

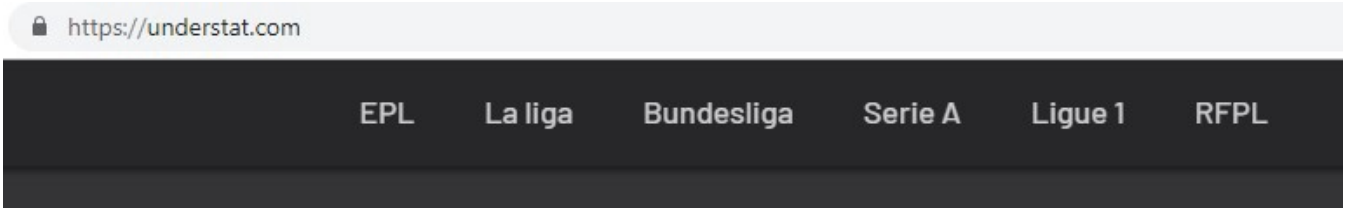
- numpy — fundamental package for scientific computing with Python
- pandas — library providing high-performance, easy-to-use data structures and data analysis tools
- requests — is the only Non-GMO HTTP library for Python, safe for human consumption. (love this line from official docs :D)
- BeautifulSoup — a Python library for pulling data out of HTML and XML files.

```
pd.read_csv)
import requests
from bs4 import BeautifulSoup
```

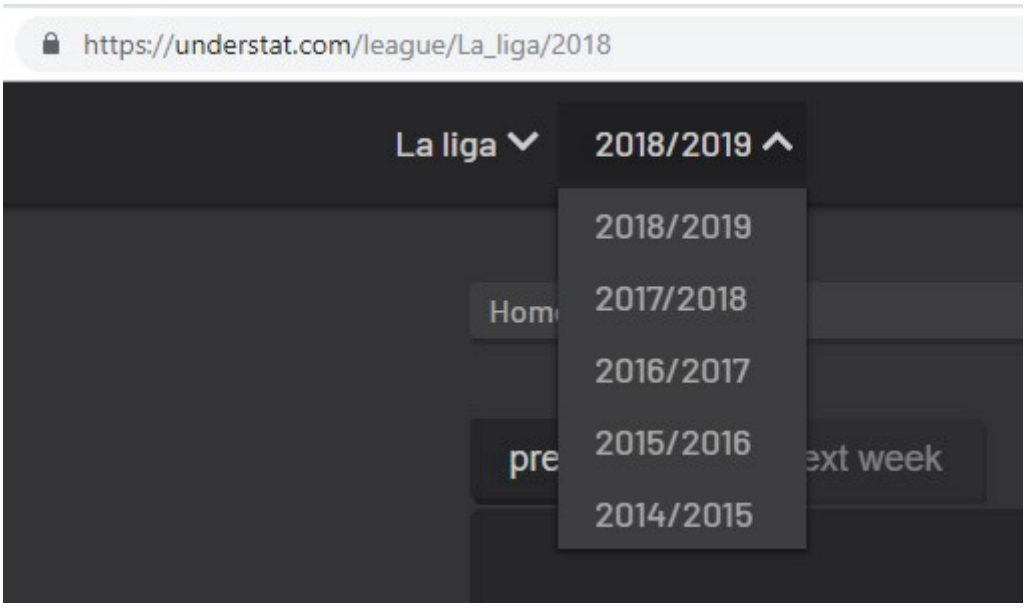
Website research and structure of data

In any web scraping project first thing you have to do is to research the web-page you want to scrape and understand how it works. That's fundamental. So we start from there.

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



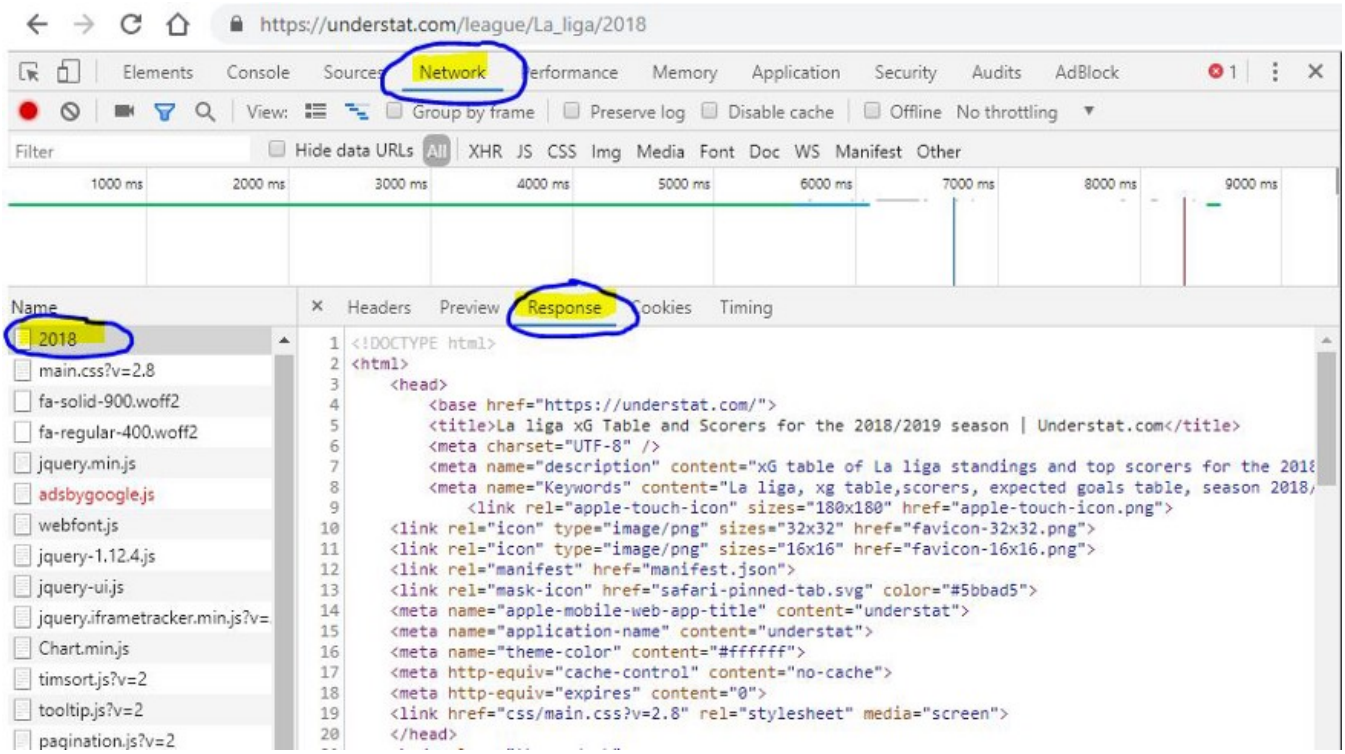
And we also see that the data collected is starting from season 2014/2015. Another notion we make is the structure of URL. It is ' `https://understat.com/league'` + `'/name_of_the_league'` + `'/year_start_of_the_season '`



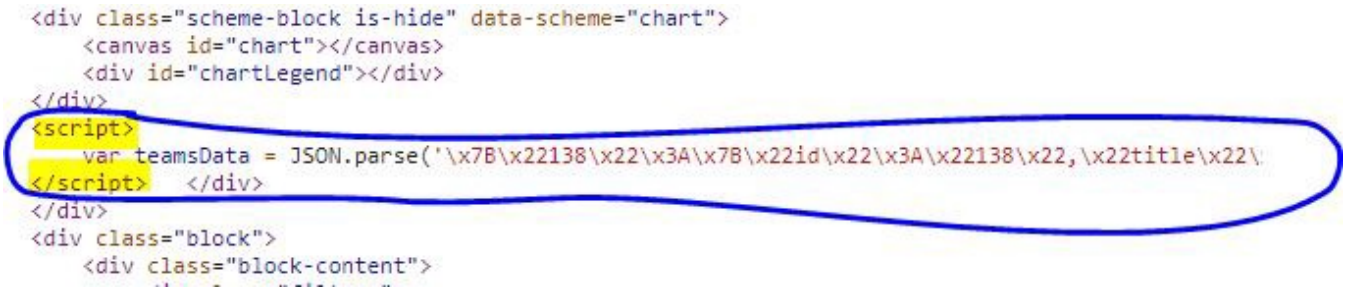
So we create variables with this data to be able to select any season or any league.

```
# create urls for all seasons of all leagues
base_url = 'https://understat.com/league'
```

Next step is to understand where the data is located on the web-page. For this we open Developer Tools in Chrome, go to tab “Network”, find file with data (in this case 2018) and check the “Response” tab. This is what we will get after running *requests.get(URL)*



After going through content of the web-page we find that the data is stored under “script” tag and it is JSON encoded. So we will need to find this tag, get JSON from it and convert it into Python readable data structure.




```
# Starting with latest data for Spanish league, because I'm a
Barcelona fan
url = base_url+'/'+leagues[0]+'/'+seasons[4]
res = requests.get(url)

soup = BeautifulSoup(res.content, "lxml")
```

Working with JSON

We found that the data interesting us is stored in teamsData variable, after creating a soup of html tags it becomes just a string, so we find that text and extract JSON from it.

```
import json

string_with_json_obj = ''

# Find data for teams
for el in scripts:
    if 'teamsData' in el.text:
        string_with_json_obj = el.text.strip()

# print(string_with_json_obj)

# strip unnecessary symbols and get only JSON data
ind_start = string_with_json_obj.index("('")+2
ind_end = string_with_json_obj.index("')")
json_data = string_with_json_obj[ind_start:ind_end]

json_data = json_data.encode('utf8').decode('unicode_escape')
```

Once we have gotten our JSON and cleaned it up we can convert it into Python dictionary and check how it looks (commented print statement).

Understanding data with Python

When we start to research the `data` we understand that this is a dictionary of dictionaries of 3 keys: *id*, *title* and *history*. The first layer of dictionary uses *ids* as keys too.

Also from this 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).

We can gather teams names after going over the first layer dictionary.

```
# Get teams and their relevant ids and put them into separate
dictionary
```

The *history* is the array of dictionaries where keys are names of metrics (read column names) and values are values, despite how tautological is that :D.

We understand that column names repeat over and over again so we add them to separate list. Also checking how the sample values look like.

```
# EDA to get a feeling of how the JSON is structured
# Column names are all the same, so we just use first element
columns = []
# Check the sample of values per each column
values = []
for id in data.keys():
    columns = list(data[id]['history'][0].keys())
    values = list(data[id]['history'][0].values())
    break
```

After outputting few print statements we find that Sevilla has the id=138, so getting all the data for this team to be able to reproduce the same steps for all teams in the league.

```
sevilla_data = []
for row in data['138']['history']:
    sevilla_data.append(list(row.values()))

df = pd.DataFrame(sevilla_data, columns=columns)
df.head(2)
```

For the sake of leaving this article clean I won't add the content of created DataFrame, but in the end you will find links to IPython notebooks on Github and Kaggle with all code and outputs. Here just samples for the context.

So, wualya, congrats! We have the data for all matches of Sevilla in season 2018–2019 within La Liga! Now we want to get that data for all Spanish teams. Let's loop through those bites baby!

```
for id, team in teams.items():
    teams_data = []
    for row in data[id]['history']:
        teams_data.append(list(row.values()))

df = pd.DataFrame(teams_data, columns=columns)
dataframes[team] = df
print('Added data for{}.'.format(team))
```

After this code finishes running we have 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. Let's fix that!

```
for team, df in dataframes.items():
    dataframes[team]['ppda_coef'] = dataframes[team]
['ppda'].apply(lambda x: x['att']/x['def'] if x['def'] != 0 else 0)
    dataframes[team]['oppda_coef'] = dataframes[team]
['ppda_allowed'].apply(lambda x: x['att']/x['def'] if x['def'] != 0
else 0)
```

Now we have all our numbers, but for every single game. What we need is the totals for the team. Let's find out the columns we have to sum up. For this we go back to original table at understat.com and we find that all metrics should be summed up and only PPDA and OPPDA are means in the end.

```
cols_to_sum = ['xG', 'xGA', 'npxG', 'npxGA', 'deep', 'deep_allowed',
'scored', 'missed', 'xpts', 'wins', 'draws', 'loses', 'pts',
'npxGD']
cols_to_mean = ['ppda_coef', 'oppda_coef']
```

We are ready to calculate our totals and means. For this we loop through dictionary *dataframes* and call .sum() and .mean() DataFrame methods that return Series, that's

```
frames = []
for team, df in dataframes.items():
    sum_data = pd.DataFrame(df[cols_to_sum].sum()).transpose()
    mean_data = pd.DataFrame(df[cols_to_mean].mean()).transpose()
    final_df = sum_data.join(mean_data)
    final_df['team'] = team
    final_df['matches'] = len(df)
    frames.append(final_df)

full_stat = pd.concat(frames)
```

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

```
full_stat = full_stat[['team', 'matches', 'wins', 'draws', 'loses',
'scored', 'missed', 'pts', 'xG', 'npxG', 'xGA', 'npxGA', 'npxGD',
'ppda_coef', 'oppda_coef', 'deep', 'deep_allowed', 'xpts']]

full_stat.sort_values('pts', ascending=False, inplace=True)
full_stat.reset_index(inplace=True, drop=True)
full_stat['position'] = range(1,len(full_stat)+1)
```

Also in the original table we have values of differences between expected metrics and real. Let’s add those too.

```
full_stat['xG_diff'] = full_stat['xG'] - full_stat['scored']
full_stat['xGA_diff'] = full_stat['xGA'] - full_stat['missed']
full_stat['xpts_diff'] = full_stat['xpts'] - full_stat['pts']
```

Converting floats to integers where appropriate.

```
cols_to_int = ['wins', 'draws', 'loses', 'scored', 'missed', 'pts',
'deep', 'deep_allowed']
full_stat[cols_to_int] = full_stat[cols_to_int].astype(int)
```

Prettifying output and final view of a DataFrame.

```
'xGA_diff', 'npxGA', 'npxGD', 'ppda_coef', 'oppda_coef', 'deep',
'deep_allowed', 'xpts', 'xpts_diff']

full_stat = full_stat[col_order]
full_stat.columns = ['#', 'team', 'M', 'W', 'D', 'L', 'G', 'GA',
'PTS', 'xG', 'xG_diff', 'NPxG', 'xGA', 'xGA_diff', 'NPxGA', 'NPxGD',
'PPDA', 'OPPDA', 'DC', 'ODC', 'xPTS', 'xPTS_diff']

pd.options.display.float_format = '{:,.2f}'.format
full_stat.head(10)
```

	#	team	M	W	D	L	G	GA	PTS	xG	xG_diff	NPxG	xGA	xGA_diff	NPxGA	NPxGD	PPDA	OPPDA	DC	ODC	xPTS	xPTS_diff
0	1	Barcelona	38	26	9	3	90	36	87	83.28	-6.72	76.58	44.93	8.93	43.44	33.14	9.02	16.40	417	171	73.96	-13.04
1	2	Atletico Madrid	38	22	10	6	55	29	76	51.87	-3.13	48.73	41.43	12.43	37.72	11.01	11.07	11.10	252	190	59.43	-16.57
2	3	Real Madrid	38	21	5	12	63	46	68	68.65	5.65	61.97	48.68	2.68	42.73	19.24	8.90	14.78	341	168	64.77	-3.23
3	4	Valencia	38	15	16	7	51	35	61	61.88	10.88	56.57	42.85	7.85	36.91	19.66	12.96	9.47	278	215	65.16	4.16
4	5	Sevilla	38	17	8	13	62	47	59	69.16	7.16	64.54	46.71	-0.29	41.51	23.03	10.65	10.02	321	211	65.08	6.08
5	6	Getafe	38	15	14	9	48	35	59	47.03	-0.97	42.58	44.23	9.23	39.02	3.56	8.77	5.70	186	196	53.19	-5.81
6	7	Espanyol	38	14	11	13	48	50	53	50.16	2.16	47.18	54.62	4.62	48.55	-1.36	9.86	9.82	241	241	50.09	-2.91
7	8	Athletic Club	38	13	14	11	41	45	53	44.44	3.44	38.92	47.16	2.16	43.44	-4.53	8.30	11.30	221	185	50.01	-2.99
8	9	Real Sociedad	38	13	11	14	45	46	50	47.99	2.99	40.55	48.09	2.09	45.68	-5.13	9.94	9.49	194	208	51.13	1.13
9	10	Alaves	38	13	11	14	39	50	50	40.87	1.87	38.64	54.53	4.53	50.07	-11.43	11.23	7.10	129	270	44.02	-5.98

Original table:

Table Charts overall home away Start date End date

№	Team	M	W	D	L	G	GA	PTS	xG	NPxG	xGA	NPxGA	NPxGD	PPDA	OPPDA	DC	ODC	xPTS
1	Barcelona	38	26	9	3	90	36	87	83.28 -6.72	76.58	44.93 +8.93	43.44	+33.14	8.43	15.24	417	171	73.96 -13.04
2	Atletico Madrid	38	22	10	6	55	29	76	51.87 -3.13	48.73	41.43 +12.43	37.72	+11.01	10.30	10.29	252	190	59.43 -16.57
3	Real Madrid	38	21	5	12	63	46	68	68.65 +5.65	61.97	48.68 +2.68	42.73	+19.24	8.67	13.52	341	168	64.77 -3.23
4	Valencia	38	15	16	7	51	35	61	61.88 +10.88	56.57	42.85 +7.85	36.91	+19.66	12.08	9.03	278	215	65.16 +4.16
5	Sevilla	38	17	8	13	62	47	59	69.16 +7.16	64.54	46.71 -0.29	41.51	+23.03	10.08	9.50	321	211	65.08 +6.08
6	Getafe	38	15	14	9	48	35	59	47.03 -0.97	42.58	44.23 +9.23	39.02	+3.56	8.57	5.38	186	196	53.19 -5.81
7	Espanyol	38	14	11	13	48	50	53	50.16 +2.16	47.18	54.62 +4.62	48.55	-1.36	9.31	9.26	241	241	50.09 -2.91
8	Athletic Club	38	13	14	11	41	45	53	44.44 +3.44	38.92	47.16 +2.16	43.44	-4.53	7.97	10.54	221	185	50.01 -2.99
9	Real Sociedad	38	13	11	14	45	46	50	47.99 +2.99	40.55	48.09 +2.09	45.68	-5.13	9.36	9.19	194	208	51.13 +1.13
10	Real Betis	38	14	8	16	44	52	50	51.68 +7.68	46.35	53.54 +1.54	49.83	-3.48	8.12	11.34	241	220	52.39 +2.39
11	Alaves	38	13	11	14	39	50	50	40.87 +1.87	38.64	54.53 +4.53	50.07	-11.43	10.51	6.72	129	270	44.02 -5.98
12	Eibar	38	11	14	13	46	50	47	56.63 +10.63	50.69	47.88 -2.12	44.91	+5.78	6.83	10.26	235	177	56.75 +9.75
13	Leganes	38	11	12	15	37	43	45	41.25 +4.25	37.53	43.94 +0.94	40.80	-3.26	11.51	9.22	155	190	51.15 +6.15
14	Villarreal	38	10	14	14	49	52	44	54.80 +5.80	50.33	56.12 +4.12	52.40	-2.08	9.89	10.46	232	263	52.18 +8.18
15	Levante	38	11	11	16	59	66	44	54.09 +4.91	48.89	78.86 +12.86	72.01	-23.13	9.61	6.58	206	295	40.23 -3.77
16	Celta Vigo	38	10	11	17	53	62	41	48.70 +4.30	43.50	63.43 +1.43	58.23	-14.73	10.35	10.89	195	248	41.63 +0.63
17	Real Valladolid	38	10	11	17	32	51	41	41.64 +0.64	36.44	57.65 +6.65	50.83	-14.39	10.17	8.55	139	243	41.80 +0.80
18	Girona	38	9	10	19	37	53	37	42.99 +5.99	38.53	55.33 +2.33	48.39	-10.86	10.08	9.44	177	261	45.17 +8.17
19	SD Huesca	38	7	12	19	43	65	33	51.38 +8.38	46.18	63.43 -1.57	56.74	-10.56	9.59	7.68	171	259	44.96 +11.96
20	Rayo Vallecano	38	8	8	22	41	70	32	47.23 +8.23	43.52	62.32 -7.68	54.00	-10.48	9.20	6.99	133	253	43.02 +11.02

code here, but will leave a link to entire scraping solution at my Github and Kaggle.

Final dataset

After looping through all leagues and all seasons and few manipulation steps to make data exportable I've got a CSV file with scraped numbers. The dataset is available here.

Conclusion

Hope you found it useful and got some valuable info. Anyway, if you reached this point I just want to say thank you for reading, for allocating your time, energy and attention to my 5 cents and wish you a lot of love and happiness, you're awesome!

Originally published at http://sergilehkyi.com on June 23, 2019.

Data Science Web Scraping Football Data Sports Analytics