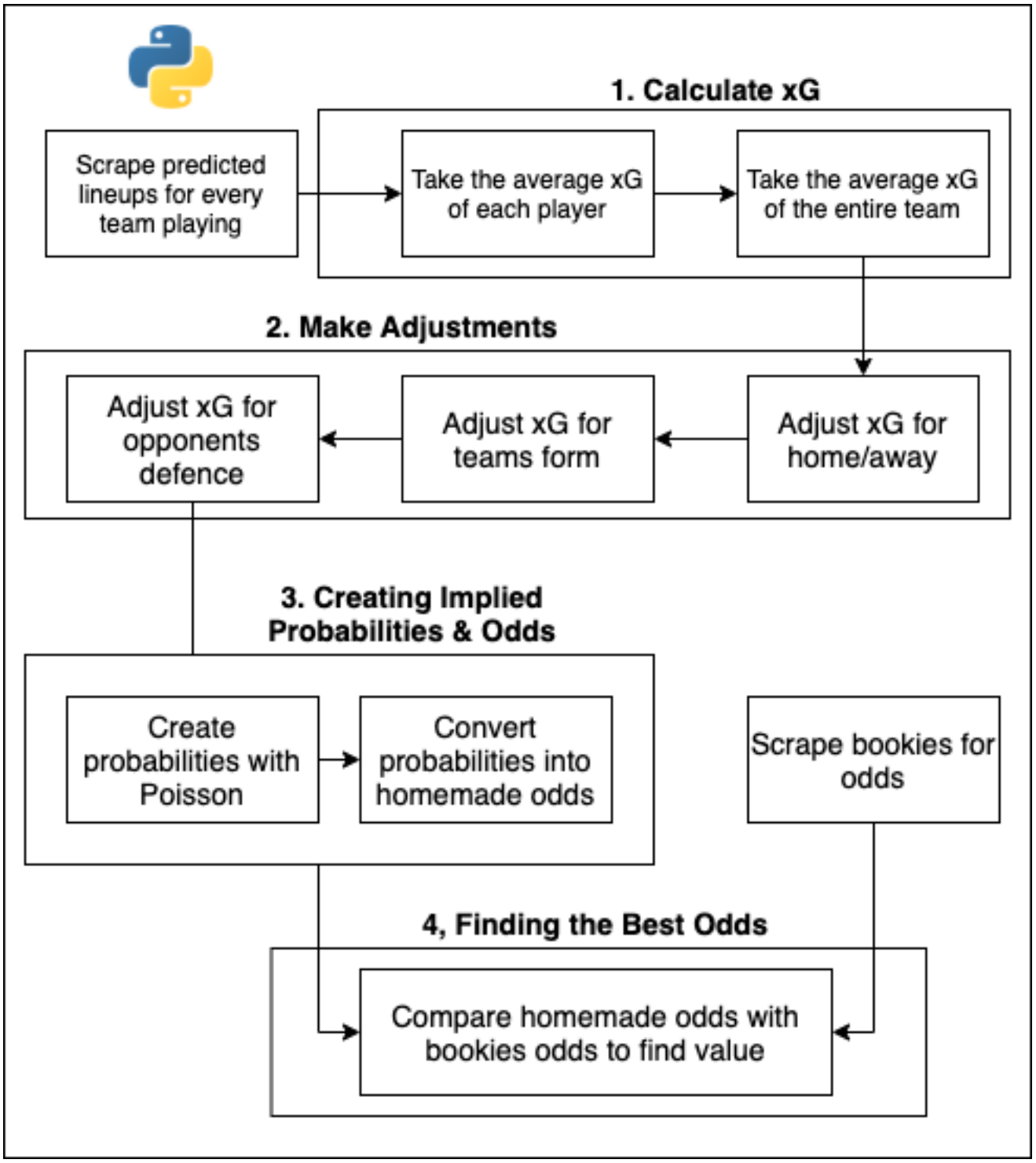
Python Betting Model for Six Football Leagues



Using statistics, Pandas, BeautifulSoup and AWS to identify value bets

form	team_name	avG	avG_adjusted	players_missing	home_away_adjustment	home_away	fixture
0.88400	Crystal Palace	1.16402	1.1318908496370539	0.00000	1.1	home	Crystal Palace-v-Leicester
0.94800	Chelsea	1.17586	1.2261844515515863	0.00000	1.1	home	Chelsea-v-Aston Villa
1.04400	Aston Villa	2.54014	2.5193107387419422	0.00000	0.95	away	Chelsea-v-Aston Villa
1.07600	Everton	1.38377	1.6378261505441367	0.00000	1.1	home	Everton-v-Manchester City
1.03600	Manchester City	1.97131	1.9401611831060424	0.00000	0.95	away	Everton-v-Manchester City
1.08400	Leicester	1.24543	1.2825430637490751	0.00000	0.95	away	Crystal Palace-v-Leicester

Last year I built a football betting model (algorithm) in Python to help me make data-driven predictions and to identify betting opportunities in the English Premier League (EPL).

This year I re-built the system from the ground up to find betting opportunities across **six** different leagues (EPL, La Liga, Bundesliga, Ligue 1, Serie A and RFPL).

After completing my last model in late December 2019 I began putting it to the test with £25 of bets every week. Unfortunately, I only managed to fit in eight weeks of betting before COVID-19 cut the EPL short.

The good news is that I broke even during this period, I bet £200 and I got £200 back.

But I'm not here to break even.

In this article I'm going to explain my methodology, technical implementation and betting strategy in order to help you create your own betting model with Python.

The new model leverages a lot of the code that was used in the previous model and can be simplified into four steps:

1. Expected Goals

"Expected goals" (xG) is a much better reflection of a team's performance than shots or shots on target. Instead of considering every "shot on target" equally, xG considers the quality of each shot taken by looking at where it was taken, what foot it was taken with and the "style of play".

If you're interested in learning more about this revolutionary metric, I talk about it more in this article:

This step considers the predicted line-ups of each team and looks at their individual xG values over the last n (usually 6) games. I've previously taken the lineups from Fantasy Football Scout:

But in order to gather lineups from leagues all over the world I have to scrape lineups from a website that covers games all over the world!

I take the HTML for every fixture in a given season:

Then I get every fixture ID from this list that is in the current game week (current_fixtures pulled from API Football):

I pull the HTML from every fixture:

And I parse the HTML:

The program then sums every players average expected goals (avG) from a database of player attributes and divides it by the number of players (11). In other words, it's the average expected goals for the entire team over a given period.

team_name	avG
Crystal Palace	1.16402
Chelsea	1.17586
Aston Villa	2.54014
Everton	1.38377
Manchester City	1.97131
Leicester	1.24543

This feature was written under the assumption that betting markets become more efficient as the event draws nearer. Because as more bets are made, the " Wisdom of the Crowd" effect takes over (collective opinion is more accurate than one expert opinion). That's why I make all of my bets well in advance of the games with the predicted lineups.

2. Adjustments

Once xG has been calculated for every team that's playing on a given day, the following adjustments are made:

- **Home/away** — I assume that teams who are playing at home are expected to score 10% more goals on average, whilst teams who are playing away will score 5% less on average.
- **DefenceofOpposition**— By comparing the opposition's previous xG with their actual goals scored in a game, I create a "defence factor" which I use to adjust their average xG. E.g. if a team had an xG of 2.35 and they only scored one goal then the opposition must have defended well.

xG_diff_avg = xG_diff_total / fixture_history

- **Form** — This adjustment considers how many of the last n games the team has won/drawn/lost when playing at home or away. It also considers their recent “streak”, whether that be a streak of wins, draws, or losses.

This creates the final dataframe before calculating probabilities and odds.

3. Creating Implied Probabilities & Odds

Now that we have a team's average expected goals for a given game, we can put this into Poisson’s distribution.

$$f(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Poisson distribution formula

This equation is a discrete probability distribution that predicts the probability of discrete events over a fixed time interval (e.g. goals scored over ninety minutes).

	0	1	2	3	4	5	6
Burnley	0.59009	0.31126	0.08209	0.01443	0.00190	0.00020	0.00002
Liverpool	0.23406	0.33990	0.24680	0.11946	0.04337	0.01260	0.00305
West Bromwich Albion	0.53239	0.33561	0.10578	0.02223	0.00350	0.00044	0.00005
Wolverhampton Wanderers	0.39374	0.36699	0.17103	0.05314	0.01238	0.00231	0.00036
Leeds	0.11085	0.24383	0.26815	0.19661	0.10811	0.04756	0.01744
West Ham	0.41829	0.36457	0.15888	0.04616	0.01006	0.00175	0.00025
Brighton	0.42924	0.36303	0.15351	0.04328	0.00915	0.00155	0.00022
Tottenham	0.47243	0.35426	0.13283	0.03320	0.00622	0.00093	0.00012

I make the assumption that because scoring more than five goals is so unlikely in a given game, I only calculate the probabilities of zero to five goals (although this season is FILLED with goals). So now we’ve got every team's probabilities for scoring goals, it’s time to create our own odds.

3.5 Creating ‘Homemade’ Odds

My betting strategy currently focuses on “over and under” bets, which are concerned with teams scoring over x or under y amount of goals. E.g. I bet that Manchester United will score over two goals against Aston Villa.

Team Total Goals			
Man Utd		Aston Villa	
Over 1.5 8/13	Under 1.5 6/5	Over 1.5 7/4	Under 1.5 2/5
Over 2.5 7/4	Under 2.5 2/5	Over 2.5 6/1	Under 2.5 1/10
Over 3.5 5/1	Under 3.5 1/8	Over 3.5 20/1	Under 3.5 1/66
Over 4.5 14/1	Under 4.5 1/33		

In order to work out the probabilities of these events, I simply take the sum of probabilities for a team scoring that many goals or more.

E.g. for “over 1.5” goals I would sum the probabilities of a team scoring 2, 3, 4 and 5 goals.

Odds are simply the inverse of implied probabilities, so inverting this number provides me with my very own homemade odds for “overs and unders”!

	O0_5	O1_5	O2_5	O3_5	U0_5	U1_5	U2_5
Burnley	2.43968	10.13896	60.46684	475.24505	1.69465	1.10945	1.01684
Liverpool	1.31212	2.36839	5.70024	17.86757	4.27242	1.74230	1.21840
West Bromwich Albion	2.13875	7.57842	38.20834	253.50843	1.87833	1.15208	1.02693
Wolverhampton Wanderers	1.65058	4.18659	14.74332	68.07336	2.53976	1.31453	1.07324
Leeds	1.15706	1.61178	2.83866	6.42377	9.02102	2.81946	1.60557
West Ham	1.71992	4.61154	17.25047	84.66711	2.39069	1.27736	1.06186
Brighton	1.75281	4.81954	18.52686	93.47379	2.32971	1.26221	1.05733
Tottenham	1.89593	5.77420	24.77794	139.71166	2.11674	1.20965	1.04220

I’ve gone over some of the code that I used to create these dataframes in a previous article if you’re interested:

4. Finding the Best Odds & Strategy

After calculating my own complete set of odds, the program then proceeds to scrape data from over twenty betting websites. It returns the best odds for each betting market that I'm interested in along with the name of the associated bookmaker.

This data is formatted in exactly the same way as my previous dataframe so that I end up with this:

	O0_5	O1_5	O2_5	O3_5	U0_5	U1_5	U2_5
Burnley	1.33000	2.50000	6.50000	15.00000	3.40000	1.53000	1.14000
Liverpool	1.02000	1.20000	1.66000	2.70000	15.00000	4.33000	2.25000
West Bromwich Albion	2.30000	8.00000	29.00000	81.00000	1.67000	1.09000	1.01000
Wolverhampton Wanderers	1.53000	3.60000	10.00000	34.00000	2.50000	1.30000	1.06000
Leeds	1.11000	1.57000	2.75000	6.00000	6.50000	2.40000	1.44000
West Ham	1.29000	2.30000	5.25000	15.00000	3.80000	1.62000	1.17000
Brighton	1.38000	2.75000	7.00000	19.00000	3.15000	1.44000	1.11000
Tottenham	1.30000	2.45000	5.75000	15.00000	3.60000	1.57000	1.14000

In order to find "value" bets (where I believe a bookmaker has predicted an outcome to be more unlikely than I believe it to be) I need to find the difference between the bookies odds and my homemade odds :

When values within my "df_betting_alpha" dataframe are positive then this implies that there is "value" in that bet, because the bookies are better odds than my homemade odds imply:

	O0_5	O1_5	O2_5	O3_5	U0_5	U1_5	U2_5
Burnley	-1.10968	-7.63896	-53.96684	-460.24505	1.70535	0.42055	0.12316
Liverpool	-0.29212	-1.16839	-4.04024	-15.16757	10.72758	2.58770	1.03160
West Bromwich Albion	0.16125	0.42158	-9.20834	-172.50843	-0.20833	-0.06208	-0.01693
Wolverhampton Wanderers	-0.12058	-0.58659	-4.74332	-34.07336	-0.03976	-0.01453	-0.01324
Leeds	-0.04706	-0.04178	-0.08866	-0.42377	-2.52102	-0.41946	-0.16557
West Ham	-0.42992	-2.31154	-12.00047	-69.66711	1.40931	0.34264	0.10814
Brighton	-0.37281	-2.06954	-11.52686	-74.47379	0.82029	0.17779	0.05267
Tottenham	-0.59593	-3.32420	-19.02794	-124.71166	1.48326	0.36035	0.09780

So when the bookies odds are higher than the homemade odds, that implies that this is potentially a good bet to make. In the above picture the model correctly identified that betting on Liverpool to not score as many goals as usual would've been a profitable bet to make (U1_5 & U2_5).

Liverpool

1 - 1

Full-Time

West Bromwich Albion

4.5 Betting Strategy

Once the program has all of these dataframes, there is a final stage of analysis to help identify the most profitable bets.

For each bet that could potentially be made, there are three threshold values that need to be exceeded in order for the bet to be made:

- **Alpha Threshold** — The bookies must have under-predicted this outcome by a given amount.
- **Homemade Odds Threshold** — The minimum homemade probability of an outcome.
- **Bookies Odds Threshold** — The minimum probability of an outcome provided by the bookies. This also implies a minimum odds that I'm willing to bet on.

These thresholds allow the betting strategy to be adjusted appropriately, it is possible to take on more risk by reducing these values and by making more bets. I've found it more profitable to make a few bets with higher thresholds.

The amount that is placed on each bet is dictated by the Kelly Criterion:

$$f^* = \frac{bp - q}{b}$$

- f = the fraction of the bankroll to bet
- b = the decimal odds – 1
- p = the probability of winning
- q = the probability of losing, which is 1 – p

This formula suggests how much of your total bankroll should be placed on each bet based on the probability of winning that bet.

Every dataframe that has been created during this analysis is automatically uploaded into a datalake (S3) for future analysis. Each row is split out and saved into its own file in order to partition the data by team.

This code snippet demonstrates how to save a dataframe row by row:

The algorithm is now capable of making and comparing odds on outcomes across six different football leagues.

Betting opportunities are identified by analysing four metrics (xG, form, home/away and opposition defence) to generate “homemade” odds which are then compared against the best odds offered by the bookies.

Expanding this algorithm across six different leagues allows more betting opportunities to be identified throughout the season.

In the future I’m going to create a database which will allow backtesting of betting strategies to further optimise this betting process, but for now I’m going to enjoy another season of football!

```
for index, row in df.iterrows():
 single_data_row = {}
 for series_index, series_value in row.items():
 single_data_row[series_index] = series_value
 team = index.replace(' ', '_')
 csv_buffer = StringIO()
 headers = list(single_data_row.keys())
 writer = csv.DictWriter(csv_buffer, fieldnames=headers)
 writer.writeheader()
 writer.writerow(single_data_row)
 filepath =
f'data/{season}/overunders/{league_name}/{team}/{current_round_formatted}/{filena
me}'
 logger.info(f'Saving to {filepath}')
 s3_response = s3_resource.Object(s3_bucket,
filepath).put(Body=csv_buffer.getvalue())
```