Can I Beat the Bookies?
Betting on the English Premier League with Logistic Regression

Nick Campanelli

Problem Statement:

Football (soccer) is the world's most popular sport. Millions around the globe tune in to celebrate, agonize, and wager over scores, results and championships. Experts suggest that worldwide the sports betting industry is worth up to 1 trillion USD. With so much money at stake on the outcomes of football matches, being able to accurately predict a team's results using past data is a tantalizing prospect. Using past results from the English Premier League, the most lucrative league competition in the world, can I build a model that can correctly predict enough results to beat the bookies and record a profit across a season of betting?

Data:

The primary dataset in use here is "EPL Results 1993-2018" compiled from Sam Lawson and available in CSV format on Kaggle. The dataset contains 9,665 matches with 11 observations recorded for each match. The ones of primary interest to my analysis are home/away team, full time home/full time away goals, and full time result. The dataset will also be appended with additional columns that denote league position from the previous season, opponent's all time win percentage, and head to head win percentage. Other features that may be added are a team's form or net transfer spend in the previous transfer windows. Any numerical features that can be compiled from available data may help in regression analysis. Having obtained my dataset from Kaggle, it was already well organized and not missing any data. However, I did need to change the format to aid in analysis, pushing the date string to a datetime object, and adding several columns that were not included originally. My steps are outlined below:

I was able to load the CSV file straight into a dataframe without any difficulties, dropping a few columns to make it less messy. I also had to convert the 'Date' column to a datetime object with pandas to_datetime passing the optional argument 'dayfirst = True' to get it to work. After that, I created a new column that assigned a random result to each row (game). Once these preliminary steps were completed, my dataframe looked as follows:

	Date	HomeTeam	AwayTeam	FTHG	FTAG	FTR	Season	randomResult
0	1993-08-14	Arsenal	Coventry	0	3	A	1993-94	3
1	1993-08-14	Aston Villa	QPR	4	1	H	1993-94	0
2	1993-08-14	Chelsea	Blackburn	1	2	A	1993-94	1
3	1993-08-14	Liverpool	Sheffield Weds	2	0	H	1993-94	1
4	1993-08-14	Man City	Leeds	1	1	D	1993-94	3
5	1993-08-14	Newcastle	Tottenham	0	1	A	1993-94	1
6	1993-08-14	Oldham	Ipswich	0	3	A	1993-94	3
7	1993-08-14	Sheffield United	Swindon	3	1	H	1993-94	1
8	1993-08-14	Southampton	Everton	0	2	A	1993-94	0
9	1993-08-14	West Ham	Wimbledon	0	2	A	1993-94	1

The dataframe above, though simple and organized, was not what I needed for my analysis. Due to the game by game evaluation I wanted to carry out I needed a hierarchical index were every team had all of its game grouped chronologically season by season. This required copying my original dataframe and creating two new columns called ‘team’ and ‘opponent’ in both the original and the copy. The original took every home team into new column ‘team’ and every away team into new column ‘opponent’. The copied dataframe did the opposite. When the two dataframes were concatenated together, every game was represented twice, but with a different value in the ‘team’ and ‘opponent’ column.

From there, I wrote six functions to add numerical features. The first function calculated an all time win ratio for every team and mapped it to the opponent column. The second function calculated each team’s head to head win percentage against every other opponent and mapped it to the opponent column. The third function added the column ‘Result’. Because each game is represented twice, the ‘Result’ column had to have a different value for each instance of a given game (unless it was a draw). The fourth function added in information from another dataset to include previous season’s league position, which necessarily dropped the first season worth of data. The fifth function denoted each game as home or away for the indexed team with a 0 or 1. The final function calculated a team’s form in respect to the number of games won in a row. Finally I set and ordered a three level index on the ‘Season’, ‘Team’ and ‘Date’ columns to finish my manipulations. This is what my cleaned and organized dataframe looked like:

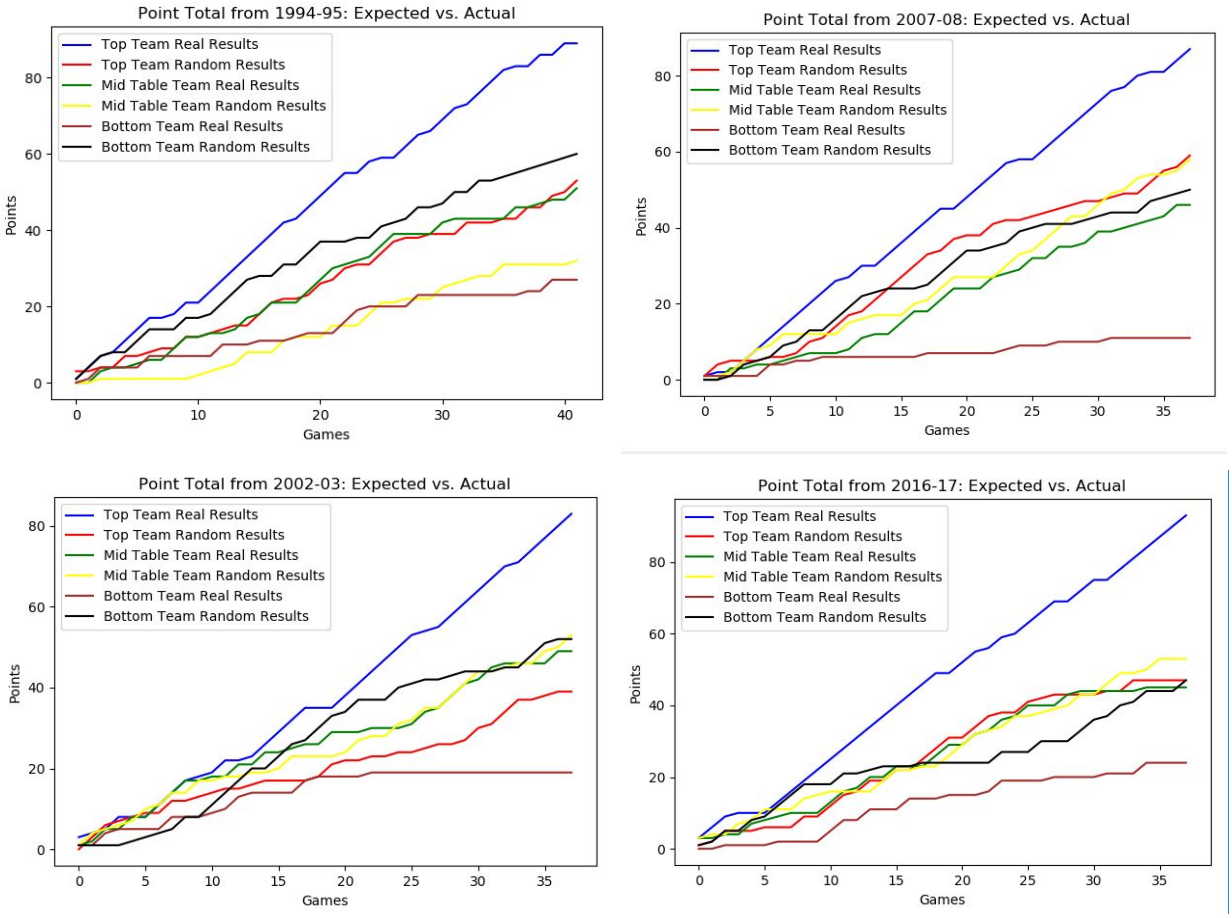
			head2head_win_ratio	HomeAway	Opp_League_Finish_Last_Year	opp_win_ratio	Streak	Result
Season	team	Date						
1994-95	Arsenal	1994-08-20	0.525000	1	16.0	0.436923	0	Win
		1994-08-23	0.545455	0	5.0	0.414062	1	Loss
		1994-08-28	0.280000	0	8.0	0.510714	0	Loss
		1994-08-31	0.558824	1	2.0	0.354575	0	Draw
		1994-09-10	0.571429	0	12.0	0.242268	0	Draw

Data Exploration and Statistical Analysis

Randomized Model:

Before diving fully into a multinomial logistic regression, I wanted to evaluate whether we can predict a team’s PPG (points per game) accurately with a randomized point generation. To

assess this, I assigned each game a randomized result, home win, away win, or draw and distributed points for these results (win: 3, draw: 1, loss: 0). This allowed me to generate a line plot of the cumulative points and random points for each team. When looking at all 20 teams per season, this plot is extremely overwhelming. Looking instead at three teams per season, the top team, the 10th place team, and the bottom team, a clear pattern emerged:

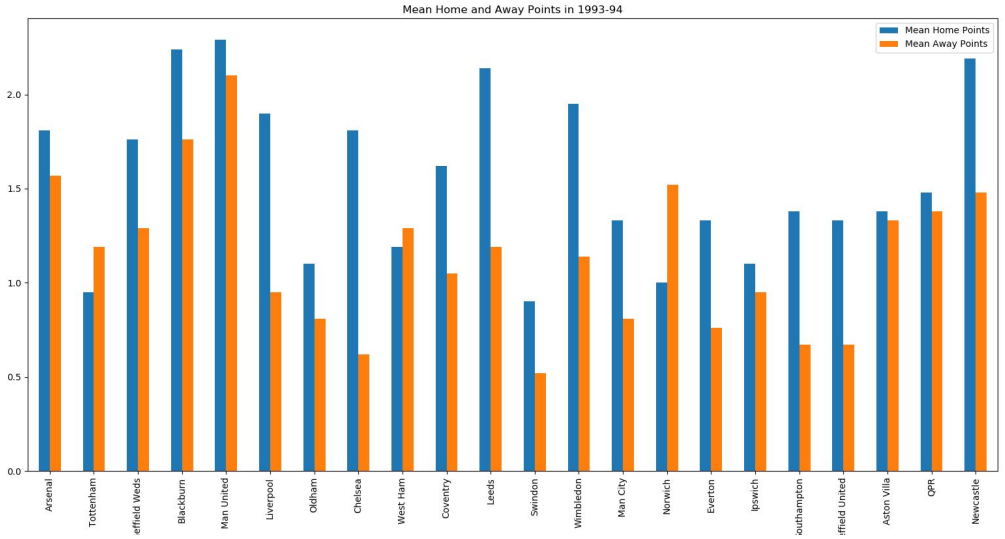


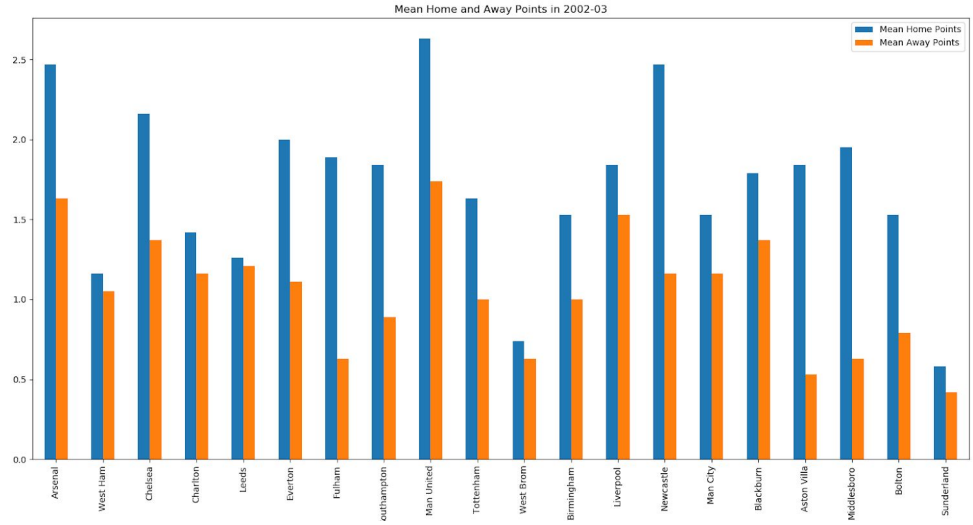
In the figures above, representing four separate seasons, we have 3 team's real and randomized cumulative points represented in a line plot. The main observation we can make is that a totally randomized model does a poor job of predicting results on a game by game basis. Interestingly however, the mid table team's results(green and yellow lines) are somewhat accurately modeled by a randomized predictor (at least insofar as total points at the end of the season are concerned). These plots confirm to me that there is variation in a team's results than can be explained by other factors. Therefore, using a multinomial logistic regression model makes sense.

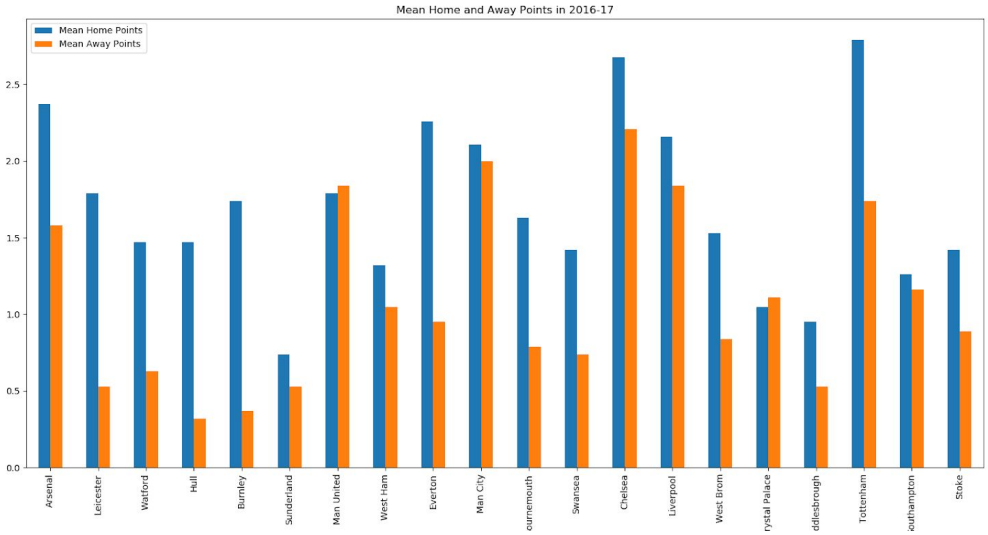
Home/Away Team:

Given that randomized results don't do a great job of predicting a team's next result, the obvious next step is to evaluate the hypothesis that a team being home or away affects the result. This assumption is commonly held to be true in most sports, therefore I expected to see

that teams collect more points at home than they do away. When I compared a team's mean point collection at home and away, a clear pattern emerged (see figures below).







There are 3 separate seasons represented above, with every team's mean PPG value for matches played home and away. Blue represents a team's home PPG and orange represents a team's away PPG. The teams are in no particular order, and the identity of the teams is not important at this stage. What is important is the pattern that almost all teams exhibit. Excluding a few notable exceptions, it seems that teams tend to pick up significantly more points when playing at home vs. away. But is this a true statistically significant difference? Let's evaluate this hypothesis with a 2 sample t-test for a difference in population means (alpha = 0.01). The null hypothesis in this test is that there is no statistical difference in mean PPG between the two samples. That is, a team playing at home does NOT collect more PPG than that same team playing away from home. By evaluating this metric across all seasons recorded, any confounding variables that might have a season by season effect should be minimized. It will also be illustrative to compare teams who have played in the Premier League every year since its inception, and teams who have bounced back up and down from the lower divisions. Below are 8 team's t-test results summarized:

Arsenal:

Matches: 882 (441 home, 441 away)
Mean PPG Home: 2.20 points
Mean PPG Away: 1.65 points
Test Statistic: 6.65
P-value: 5.26e-11

Chelsea:

Matches: 882 (441 home, 441 away)
Mean PPG Home: 2.19 points
Mean PPG Away: 1.63 points
Test Statistic: 6.73
P-value: 2.98e-11

Manchester United:

Matches: 882 (441 home, 441 away)
Mean PPG Home: 2.36 points
Mean PPG Away: 1.88 points
Test Statistic: 6.10
P-value: 1.59e-9

Liverpool:

Matches: 882 (441 home, 441 away)
Mean PPG Home: 2.08 points
Mean PPG Away: 1.45 points
Test Statistic: 7.44
P-value: 2.45e-13

Portsmouth:

Matches: 226 (113 home, 113 away)
Mean PPG Home: 1.47 points
Mean PPG Away: 0.80 points
Test Statistic: 4.47
P-value: 1.17e-05

Birmingham:

Matches: 226 (113 home, 113 away)
Mean PPG Home: 1.47 points
Mean PPG Away: 0.79 points
Test Statistic: 4.73
P-value: 3.65e-06

Leicester:

Matches: 384 (192 home, 192 away)
Mean PPG Home: 1.38 points
Mean PPG Away: 0.85 points
Test Statistic: 4.23
P-value: 2.95e-05

Watford:

Matches: 114 (57 home, 57 away)
Mean PPG Home: 1.14 points
Mean PPG Away: 0.47 points
Test Statistic: 3.22
P-value: 0.0017

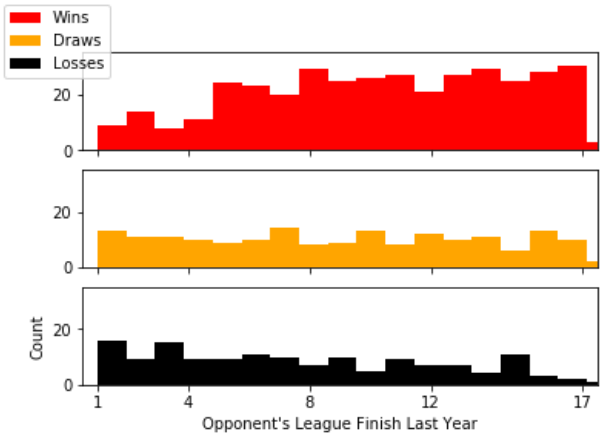
I can reject the null hypothesis in this case as there is a statistically significant difference in mean PPG at home vs. mean PPG away. Every p-value is significant at an alpha of 0.01. This holds true for teams that have been in the Premier League every season (Arsenal, Chelsea, Manchester United, Liverpool) and teams that have bounced between the Premier League and lower divisions (Portsmouth, Birmingham, Leicester, Watford). This statistical test confirms that home/away should be a very useful feature for predicting whether a team will win/lose/draw its next game.

Opponent's League Position:

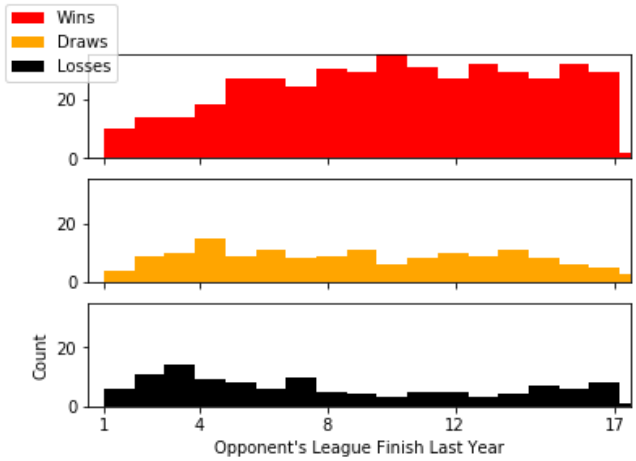
Another feature that might be useful in predicting the result of a match is an opponent's league position from last season. My intuition tells me that losses and wins will have an inverse relationship to one another in terms of an opponent's league finish last year. That is, any given team will have a greater count of wins against low placed opponents. The inverse is true of losses. If we evaluate, by count, how many wins/draws/losses each team has amassed overall

against opponents in different positions we can assess whether there is any correlation between opponent's league finish last year and PPG. One thing to note is the formatting for newly promoted teams. At the end of every season, the 3 bottom teams are relegated to the division below and the 3 top teams of the division below are promoted. I assigned these 3 teams a previous season ranking of 20th (last place). This method clumps the newly promoted teams into 1 group. This makes intuitive sense because in general I know that newly promoted teams struggle, and I don't have data on these team's performance in the lower league. Below I've plotted the counts of wins/draws/losses by opponent's league position of some of the teams mentioned above.

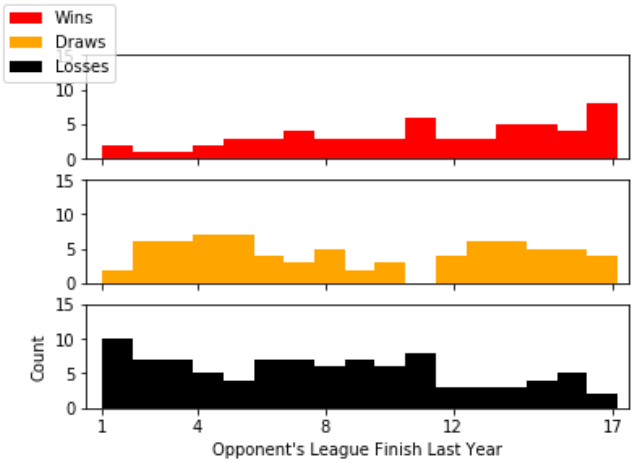
Arsenal:



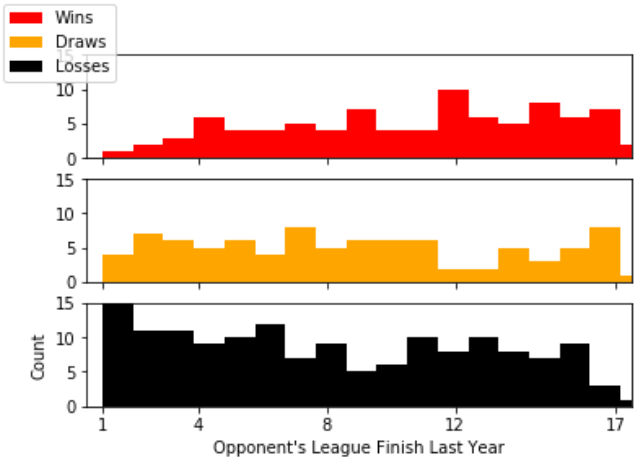
Manchester United:



Birmingham:



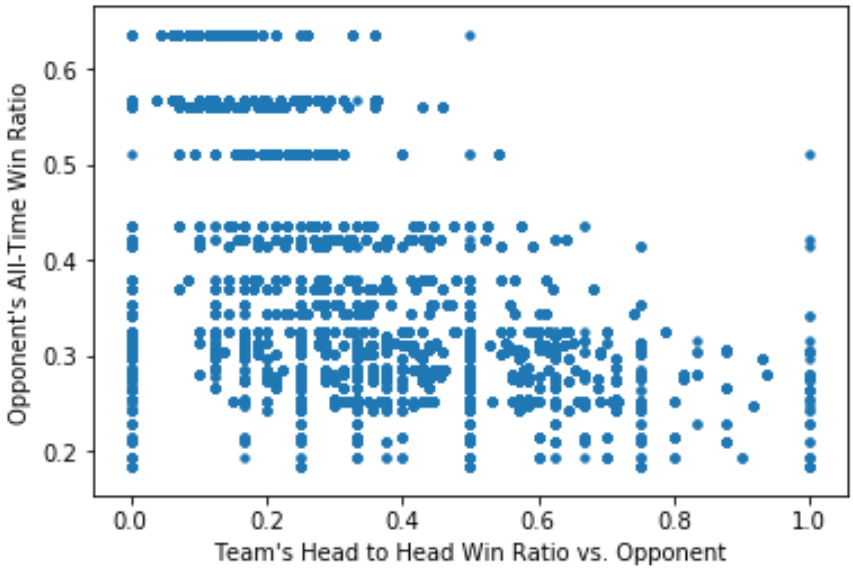
Leicester:



While each team has a significantly different distribution of wins/losses/draws, I do notice that there are two trends that seem to hold true. First is that every team wins more games against the bottom teams than the top teams. This is particularly striking for teams like Leicester and Birmingham who tend to be less competitive in the league. The inverse of this trend is true for losses. In this sense, opponent's league finish last year seems like it will be a useful feature for predicting whether a team will win/lose/draw a specific game. However, the distribution of draws for each team doesn't seem to have a specific correlation with an opponent's league position. Perhaps to successfully predict when a team will draw we need a different feature.

Head to Head and All-Time Ratio:

Two more features that I added to my data are head to head win ratio with every one of a team's opponents, and an opponent's all time win ratio. I suspect that these variables may offer a lot of the same information to my model and that including them together will result in collinearity in my regression analysis. To check this I plotted my two numerical variables against one another and checked the correlation coefficient (R):



The correlation coefficient is -0.51. This makes some intuitive sense, and we can see it illustrated in the figure above. As teams face an opponent with a lower all time win ratio, their head to head win ratio with said opponent goes up. However, it is not a perfect correlation. At the lower end of the all time win ratio there is more variation in head to head win ratio. Based on the plot and the value of R, I think that both features will be useful predicting whether a team will win or lose a particular game and will be keeping both in my model.

Model Building:

The first model I am going to use in predicting my data is a multinomial logistic regression. Implementing the training and testing sets, instantiating the model, and tuning for C with grid search cross validation is carried out in the code snippet below:

```
all_games_modeled = pd.get_dummies(double_df,prefix = ['opp'], columns = ['opponent'])
train = all_games_modeled.loc['1994-95':'2015-16']
X_train = train.drop(['FTHG','FTAG','HomeTeam','AwayTeam', 'Random Points','randomResult','FTR','Points','Result'],axis = 1)
y_train = train['Result']
test = all_games_modeled.loc['2016-17']
X_test = test.drop(['FTHG','FTAG','HomeTeam','AwayTeam', 'Random Points','randomResult','FTR','Points','Result'],axis = 1)
y_test = test['Result']
logreg_grid = LogisticRegression(multi_class='multinomial',solver = 'lbfgs',max_iter=1000)
params = {'C':[0.001, 0.01, 0.1, 1, 10, 100, 1000]}
CLF = GridSearchCV(logreg_grid,params,cv = 5,scoring='accuracy')
CLF.fit(X_train,y_train)
CLF.best_params_
```

The testing and training set are manually set. The training set includes all seasons barring the final season, 2016-17, and the testing set is just that final season. This split is

preferable to taking a random split of games because I am interested in predicting a team's results in a sequential fashion, not as a random subset of games across seasons. X_train and X_test include the features head to head win ratio, home or away, opponent's league finish last year, opponent's all time win ratio, and win streak. Running a grid search for C over the training set returned an optimal value of 0.1. Using this in a logistic regression model fit on the training set and tested on the test set returned a model accuracy of 0.595. When looking at the classification report (reproduced below) and the model coefficients, I see that the logistic regression model has relatively high precision and recall for wins and losses, but is comparatively quite poor at predicting draws. In particular, the model only predicted a draw 8% of the time that the match actually ended in a draw. The coefficients (below the classification report) correspond to the features head to head win ratio, home or away, opponent's league finish last year, opponent's all time win ratio, and win streak in that order. As we can see, head to head win ratio in the first column of the coefficient matrix is the most predictive. Home and away, in the next column is also somewhat predictive. Interestingly, opponent's league finish last year adds almost no information to the model.

	precision	recall	f1-score	support
Draw	0.41	0.08	0.14	168
Loss	0.60	0.72	0.66	296
Win	0.60	0.76	0.67	296
micro avg	0.59	0.59	0.59	760
macro avg	0.54	0.52	0.49	760
weighted avg	0.56	0.59	0.55	760

```
[[-1.14704687 -0.02145869  0.00653336 -0.82418705  0.03736893]
 [-1.93267458 -0.63671924 -0.02204734  0.5614991  -0.07155235]
 [ 3.07972146  0.65817793  0.01551397  0.26268795  0.03418342]]
```

Even with a tuned C, the multinomial logistic regression doesn't perform as well as I would have hoped. Given this, I wanted to try a different classification method. I decided to run a random forest classifier on the same training and testing sets to see if the accuracy or precision could be improved. Running a grid search cross validation with 5 folds for the parameters of n_estimators and max depth returned optimal values of 250 and 7 respectively. The code for this is below:

```
train = double_df.loc['1994-95':'2015-16']
X_train = train.drop(['opponent','FTHG','FTAG','HomeTeam','AwayTeam', 'Random Points','randomResult','FTR','Points',
                      'Result'],axis = 1)
y_train = train['Points']
test = double_df.loc['2016-17']
X_test = test.drop(['opponent','FTHG','FTAG','HomeTeam','AwayTeam', 'Random Points','randomResult','FTR','Points',
                    'Result'],axis = 1)
y_test = test['Points']
rf_grid = RandomForestClassifier(max_features='sqrt')
params = {'n_estimators':[250,500,750],'max_depth': np.linspace(7,13,6)}
clf = GridSearchCV(rf_grid,params,cv = 5,scoring='accuracy')
clf.fit(X_train,y_train)
print(clf.best_params_,clf.best_score_)
```

Using a random forest classifier instantiated with these parameters trained on the training set and evaluated on the test set returned a model accuracy of 0.582. Based on the

accuracy score, this model was less effective than the logistic regression model I used first. Looking at the classification report tells a similar story:

	precision	recall	f1-score	support
Draw	0.34	0.08	0.13	168
Loss	0.60	0.68	0.63	296
Win	0.59	0.77	0.67	296
micro avg	0.58	0.58	0.58	760
macro avg	0.51	0.51	0.48	760
weighted avg	0.54	0.58	0.53	760

The random forest classifier has lower precision and recall scores across the board. It seems that due to some creative feature engineering on my part the logistic regression model performs just as well as the more complex random forest classifier. Perhaps with a greater range of features, the random forest classifier could be improved. For now, however, due to simplicity, interpretability and speed I am going to bin the random forest classifier and dive a little deeper into the logistic regression model.

Can it Perform Better:

The first thing I notice that may offer a simple route to improve my model's predictive power is to reduce the classification problem from three classes to two. Both the precision and recall of draws in my logistic regression model is comparatively low. In particular, the model only predicted a draw in 8% of true draws! In the context of the original problem and question, if I can improve the precision of my model in predicting wins by combining losses and draws into a single target variable then potential monetary gains will be increased (if you choose to only bet on wins). So let's give that a try! Combining the losses and draws into 1 category, fitting a binomial logistic regression to the training set and generating predictions and a classification report on the testing set shows that the model has a higher precision (but lower recall.)

	precision	recall	f1-score	support
Loss/Draw	0.75	0.86	0.80	464
Win	0.71	0.56	0.63	296
micro avg	0.74	0.74	0.74	760
macro avg	0.73	0.71	0.71	760
weighted avg	0.74	0.74	0.73	760

As expected, the Loss/Draw category, now useless, has quite high precision and recall. More importantly to our use case, the precision of the algorithm in predicting wins has gone up to 0.71. This represents a significant improvement from the 0.60 in the original model with multinomial classification. For my use case, this increase in precision is worth the trade off in lost complexity given by a multinomial classification problem. If my model is correct 7/10 times that it predicts a team will win a game, then as a bettor, I can bet on only wins and be guaranteed profit in volume. In this sense, recall is unimportant. When my model predicts a win, I want to be relatively confident that it has predicted correctly.

Another avenue that might offer room for improvement is to only train the model for the teams that have been in the Premier League since its inception. They have the largest number of games played, and as a consequence will have more of an identifiable pattern of wins and losses than teams who have played only 2 or 3 seasons. Just for interest sake, I ran a model with the current top 6 English clubs (Arsenal, Liverpool, Manchester United, Manchester City, Chelsea, and Tottenham) to see if my intuition holds true:

	precision	recall	f1-score	support
Loss/Draw	0.55	0.55	0.55	86
Win	0.73	0.73	0.73	142
micro avg	0.66	0.66	0.66	228
macro avg	0.64	0.64	0.64	228
weighted avg	0.66	0.66	0.66	228

Interestingly, precision for wins only went up a little bit. My intuition was incorrect in this case.

Conclusions:

Did I succeed in producing a model that can make money by betting on predicted wins in the Premier League using real odds given by sports betting companies? Let's find out! First I will need the odds given by bookies for each game in my test set, the 2016/17 Premier League season. Thankfully, football.co.uk has me covered. They collect and maintain a massive dataset on betting odds from hundreds of different sources for each game in every Premier League season. This is an incredible resource and is specifically maintained for practicing and informing betting strategies. Perfect. Ideally, I would find the best individual odds for each game where my model predicts a win and only place bets with that company. However, with so many different companies and odds listed in the dataset, this is a massively complex task. Instead, I will use the odds supplied by the website Bet365.com, one of the world's leading online gambling companies. Without going into too many details, I will pull out the odds given for the full time result as supplies by Bet365 in decimal format for every team in the Premier League. Everytime my model predicts a win for a given team (as detailed above in the binomial logistic regression), I will simulate a $100 bet on that game (this costs me $110 with the fee required to place a bet). If my model predicts a Loss/Draw I will abstain from betting. To calculate profit in the event of a correct prediction and bet, I take the decimal odds given for that particular result, multiply by my stake of $100 and subtract my original stake plus the betting fee for pure profit. For example, if the given odds for a home team win are 1.65 and my model correctly predicts the result, I walk away with $165. $100 of this is my original stake and $10 I pay no matter what. Therefore, my profit on this bet is $55. If I lose this bet (AKA my model predicts a win for a given team but they don't win that game), I lose $110. Under this betting strategy and assessment of profit, I simulated bets on all 20 teams in the Premier League for the 2016/17 season and calculated a profit/loss for each team:

	Total Bets Placed	Profit/Loss
Arsenal	24	-366.0
Bournemouth	8	361.0
Burnley	9	1230.0
Chelsea	27	403.0
Crystal Palace	8	-69.0
Everton	12	120.0
Hull	4	541.0
Leicester	7	40.0
Liverpool	20	-376.0
Man City	19	111.0
Man United	29	-717.0
Middlesbrough	6	-17.0
Southampton	9	-467.0
Stoke	7	585.0
Sunderland	3	948.0
Swansea	3	405.0
Tottenham	21	614.0
Watford	3	335.0
West Brom	2	145.0
West Ham	10	144.0

In total I placed 231 bets of $100. With the added betting fee of $10, my financial outlay was $25,410. In total, I walked away with $29,380. This is a pure profit of $3,970. In the 9 months it takes to complete a Premier League season, my investment generated a return of 15.5%!

Does this mean I'm going to take my model and go change the world of sports gambling? Probably not. When predicting soccer there are simply too many unpredictabilities. Perhaps the 2016/17 season was one of the most easily predicted on record? Or perhaps as more television revenue has flowed into the game results have become more unpredictable, rendering my model obsolete moving forward. However, with enough capital investment it does seem that with this betting strategy and model there is some profit to be made. If nothing else, this has offered a fascinating case study into using domain knowledge to construct a relatively straightforward and interpretable model that can perform strongly with a narrowly defined use case. Thanks for reading!