Premier Predictions
Final Year Project Report

DT211C
BSc in Computer Science (Infrastructure)

Jack D’Arcy

Dr. James Carswell

School of Computer Science

Technological University, Dublin

11/4/20

Abstract

This project looks at developing a full-scale web application that involves the gathering of data through machine learning models from past football matches for it to be used to predict future fixtures. It will be used to predict future fixtures in a more user orientated way. It could aid betting, but would have little to no correlation with that industry.

The technology will seek to improve standards by which accurate predictions are made. Research to date implies a lack of advanced statistics, used to justify predictions of matches with existing models. Groups of football matches will be analysed, down to finite details, that will look towards future games, where a more educated prediction can be made.

Many existing systems were created solely for betting purposes, offering various odds not just in contrast with what they have predicted, but in aid of their predictions as well. This system will seek to offer predictions to every football fan, with users being able to create accounts and track their progress over the season. Their own sentiment towards their prediction acts as a parameter for the system’s predictor. This helps provide a more human feel to the system as the project’s predictor is there to be beaten.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:

Jack D'Arcy

Jack D’Arcy

11/4/20

Acknowledgements

I would like to thank my supervisor Dr James Carswell for giving up his time to help guide me with the project. I would also like to thank Dr Svetlana Hensman, Damian Gordon and Dr Martin McHugh for lending their knowledge to complete the project. I also have to praise my classmates who helped with the testing procedure and helped out where necessary. Finally I thank my family who have continuously encouraged me throughout the process.

Table of Contents

Table of Figures

1. Introduction

1.1. Project Background

Over many years, the game of football has become the most popular sport on the planet. As of 12 years ago, 265 million people were counted as playing it in some capacity (1) . Playing football isn't everything however, as the fanbase that tunes in or attends their favourite team's matches is even bigger, with each of the fans passionate about showcasing the sport that unites people. The viewership for major events in football speaks volumes to back this up. 2018's FIFA World Cup Final had over 3.5 billion viewers (2), pushing the most recent Olympics in Rio de Janeiro (3) close for the title of most viewed sporting event of all time. It should also be noted that football was played in both these events.

This unparalleled fanbase aims to be as in-touch with the sport as possible. Often, fans trade and compare opinions on what might happen in the future regarding results or on general developments in the game. Money also becomes involved here with large sums being traded with bets. In 2013, the betting industry was said to be worth between £435billion and £625billion, with 70% of this revenue coming from football alone (4) . However, bets and/or predictions are seldom made with a statistical background, which would form a more solid forecast of upcoming matches. "Experts" or opinions of former players are relied on mostly in the industry now, with major sporting networks using statistics to examine a team or player's form at the time.

The odds for a game can often form the bedrock of an educated prediction, as they can push someone to favour a win, loss, or draw, based on the financial return that they may see from betting. Some would favour the most likely team to win, but the lure of the underdog can greatly improve the amount of money involved.

Football has also become unignorable. Whether it be a player who is so talented that they can achieve celebrity status, or simply a pair of goal posts in a local field, its effect can be seen everywhere. It may not be to everyone's taste to play or to follow football, but it simply cannot be ignored in everyday life. In saying that, the zealousness of football fans is unrivalled in comparison to any other sport. For example, in Ireland, when the country reaches a major tournament, it almost feels like everything stops whenever the national team plays. Given its globality and involvement with money-making, this fuels the importance of good quality predictions.

There are already platforms out there that satisfy the criteria somewhat. For example, Sky Sports use a game called Super 6[1] every Saturday to engage with fans' predictions, by allowing them to guess how six games will go on that day. The more accurate the results are, the more money is won. However, what Sky is missing is a comparison with a prediction that is made purely from statistics. Instead, they use the opinion of ex-players. This can easily lead to bias, which is why this project uses the opinion of the user, together with what the statistics point towards. The interaction with an avid football fan who would be desperate to be as in-depth as possible with the current form of teams/players is what's most important.

To integrate fans further into the sport they are so invested in, this project incorporates parameters input from users and FIFA/Opta statistics into a match predictor application that uses detailed statistics to simulate future matches and provide a concrete forecast for how the matches will finish. Users can make a prediction of their own, either with or without the aid of the same statistics used for the application's prediction. Points are awarded initially for correctly predicting who will win or if a match will finish in a draw, then a bonus is awarded if the exact result is matched.

The uniqueness of this project is the main challenge. The Super 6 game involves a lot of money, which serves as an appeal to fans, while this project doesn't. In contrast, it instead uses rich statistics taken primarily from the first half of the 2019/20 Premier League season for use in the second half of the same season. It would not have been plausible to use multiple years-worth of statistics, given that some teams are better or worse than they were a few years ago. Of course teams get relegated and promoted every year, so comparing a team's record in a lower league to that of the Premier League wouldn't have been logical. Users have background knowledge of past events (eg. Manchester United beating Chelsea 4-0 on the opening weekend of the season) to date available to them and can utilise that to try and beat the system's predictor. Accounts are created, secured and tested throughout the use of the application to involve the user as much as possible, while also making it as user friendly as possible regarding design and layout.

1.2. Project Description

Premier Predictions is a web-based application that allows a user to create an account to go up against a trained machine learning-based predictor, that uses rich statistics to predict the week's Premier League results. The user's own gut feeling is also passed as a feature to see how the model's accuracy is altered. The predictor is formed through an algorithm, which is created to

[1] super6.skysports.com

determine whether the game will end in a home win, away win, or draw. The user can access a limited amount of these statistics, such as the current league table, and can use them to aid their own prediction. The system's own prediction is hidden from the user until their prediction is made. There is a scoring rubric in place whereby a correct outcome results in 1 point and a correct final score is rewarded with 3 points. While the model's predictions are kept from the user, they can give a large amount of their accumulated points to check them. It is assumed the predictor will beat the user a majority of the time given it's proven accuracy on a test set, which is one of the reasons why a social aspect between users is implemented. Similar to Fantasy Football[2], multiple users can create a league among themselves to play against each other as well as the predictor. Points are also traded in Fantasy Football. For example a player can completely change their team to make the game easier, but like users checking this system's predictions, it comes at a cost of points. The user doesn't have to take on the predictor as it is understood that it could be quite difficult to try and beat it. The application will be in place for the latter stages of the season with the knowledge of 12 seasons back to the 2007/08 season as well as the current season so far.

1.3. Project Aims and Objectives

The target is to develop and deploy a fully-functioning web application with user-friendly design. The application will have multiple screens such as the home screen, gameweek fixtures screen and login screen amongst. There are many milestone points that were reached in the lifetime of the project's development. These were all crucial to achieving the overall end goal.

1.3.1 Gathering Data

Obtaining an accurate source from which data can be utilised was essential. Various sources were researched to see where the best quality data would come from. It was even considered developing a neural network from scratch if nothing suitable could be found. Official sources were contacted such as FIFA and Opta. Opta are the main statisticians for the Premier League, with many TV networks and existing applications using their collections of data already. The official Premier League website also uses Opta as it's main source for statistics. The data source is the foundation for this project and without it, the project simply wouldn't operate at all. There was a delay in response from both organisations so other possibilities were explored. It would not be optimal to

[2] https://fantasy.premierleague.com/

use an unofficial source of data, given how detailed the data that FIFA and Opta have at their disposal, but it has to be considered.

As time passed, there was still no response from either organisation. As a result of this, an unofficial source had to be utilised. Unfortunately, with a lot of APIs to be found on the Internet, a cost incurred, possibly because of the specific nature of what is required.

1.3.2 Database Implementation

User data, as well as general data that is printed on the application's pages must be kept in a back-end database. This was to make it easy to access, and easy to manipulate with the coding aspect of the project. Links, primary keys and foreign keys between that data's tables were determined so that it was properly organised. The data used for the predictor is available in an Excel format as a .csv file, as the data taken from the API could not be simplified properly from its original JSON format into something readable for a machine learning model.

1.3.3 Machine Learning Algorithm Development

This data was in a standardised format, so it had to be manipulated and processed in order for any prediction algorithm to work to its full efficiency. Different classifiers had to be determined that would best suit the data available. Taking the data from an Excel file and putting it through the trained predictor is the focal point for the application. This required continuous fine tuning to create a high standard of accuracy that would beat a user with minimal doubt.

1.3.4 Web App Development

All of this needs to be displayed and communicated to a user in a graphical manner, so this is where the creation of the web application comes into focus. The design is paramount in user interaction, as users of the application need to be sure of what they are doing and where. The web application contains numerous pages that lead into one another and is hosted on a local server where it can be accessed. The data is displayed to the user on some pages, as well as their own statistics from their experience playing against the predictor so far.

1.3.5 Additional Features

Additional features include the fantasy league function that can be carried out by multiple users. A user can set up a league, and get others to join where they can compete against each other to see who can come out on top in the end. There is also a general leaderboard containing all users, as well as the predictor itself. The user can also view upcoming fixtures in order to plan out how to play for that particular gameweek.

1.4. Project Scope

It would be less complex to make this a purely betting application that tries to come up with better odds than the bookmakers. The motivation for this project is not money-driven in any sense. Instead, it looks to broaden the horizon of prediction in football, whereby a user competes against a trained predictor. The study of machine learning prediction as a whole thus far has not utilised a user's gut feeling as a component in its reasoning behind a result. It is difficult to avoid the betting aspect however, as users who engage in bettung can use the knowledge garnered from their actions on the app to inform how exactly to bet. Unfortunately, this is unavoidable but this is solely down to the user themselves as there is no betting functionality to be found anywhere natively in the application itself.

The data used to predict isn't available to the user until after their prediction is made in order to discourage betting. However, if the user is a frequent better, the chances are they will still use this application to help them make money from their accumulator. In saying that, while other similar systems use money as a prize for predicting accurately, this project has a more social ideology behind it. User integration is at the forefront more so than any other similar application that's already available. Personal opinion is a novelty as only the user can offer an opinion. The system is solely using statistics to make predictions and not the opinions of supposed experts. Bias is completely avoided this way.

1.5. Thesis Roadmap

Literature Review

This chapter focuses on similar applications out there, and previous Final Year Projects. The whole area of prediction in Football is explored in a deeper way here, looking at the various things each app or project has while comparing and contrasting with this project. All of the other applications and projects have both similarities and differences with this project, and cross-examining them presented interesting results. Other research is depicted here, namely Final Year Projects that have been completed in previous years that are much like Premier Predictions.

Design

The design process for the project is discussed here. Prototypal versions are shown with various original designs for aspects such as screens and screen content. Analysis has been completed on how far the project has come since previous ideas such as in the proposal at the beginning to now. It's intriguing to highlight how much a project can change over time and this project is no exception.

Development

The general development of the project is outlined in this chapter. The languages that were chosen and how they worked in tandem with one another to present this final application are examined. The challenges, points of difficulty and different attempts are addressed and the solutions or alternate routes taken to these are also discussed. The system's architecture is also analysed, with decisions that were made being inspected.

Testing and Evaluation

This chapter describes how the application was tested and how it was analysed, as well as the results of the evaluation. All users' feedback from trials is depicted including how user-friendly the project is. General tests that had to be done throughout the lifetime of the project are also discussed.

Conclusions and Future Work

This final chapter talks about what has been drawn from the project as a whole, reflecting on the application itself and how it could perhaps be improved with future work and how.

2. Literature Review

2.1. Introduction

In this chapter, the background research for the project is depicted. Two similar systems were analysed in detail in order to differentiate between the project and what is already out there. This shows the project's contribution to the state of the art and addresses any issues that were found. Further to this, past projects are also examined for the same reason. The topic of predicting with sport (not just football) has been a reasonably popular topic to pursue in projects over the years making for interesting comparisons with this project, in addition to contrasts.

2.2. Alternative Existing Solutions to Your Problem

2.2.1 Super 6

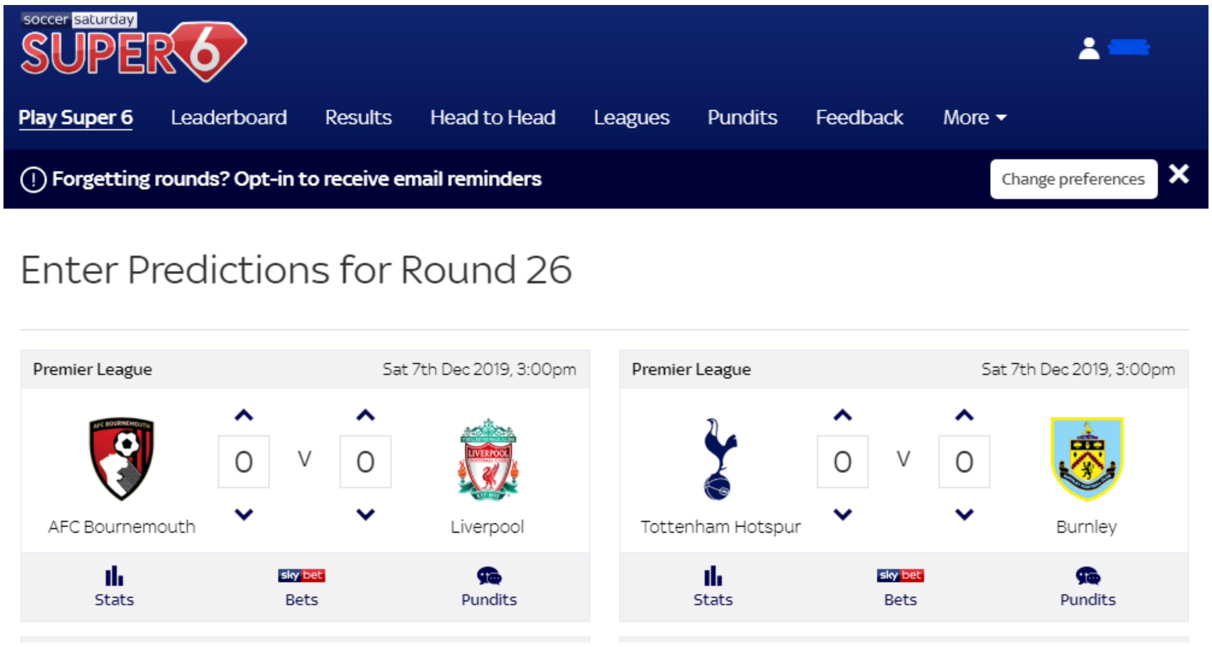
Sky Sports is the most popular sports network in Ireland and the UK (5). It maintains its status each year by offering more and more live games to subscribing customers. Competitors such as BT Sport struggle to achieve the same success but try to offer exclusivity of their own such as broadcasting the UEFA Champions League. What sets Sky apart from its rivals however, is its in-depth coverage of the English leagues. No other network can rival this and Sky's Super 6 game is a focal point of this.

Super 6 is very similar to this project. Users create accounts and attempt to predict the exact results of six particular games picked by Sky either over the weekend or in midweek. All of these games are played at the same time so it makes for more excitement as one goal across the 6 games can change the face of that particular gameweek. Sky also incorporates the use of the lower leagues in England in order to keep people interested in Super 6. For example, the Premier League and Championship don't have any fixtures when there is an international break on, as the vast majority of players have commitments to their respective countries, but lower leagues such as League 1 and League 2 both still play. In this case, Sky picks six games from those leagues and the game continues as normal.

A similar points system is in place for Super 6 as well. In this game, two points are awarded for the correct result and 5 points are given for the correct score. Over the years, Sky and its presenters and pundits have become more popular with the public. Sky have realised this, and in Super 6 they allow users to take on Jeff Stelling, the host of Soccer Saturday, a live score program in which the six games are covered live in studio, but not broadcast. Stelling is made out to be “the master” of the game as prizes are awarded for those who can beat his tally of points for the week. This has been tested during research and it is indeed quite difficult to beat him. However, it’s unclear whether it’s actually Jeff Stelling’s predictions or an educated predictor that this project definitely uses. The benefit of the doubt has to be given as he has been involved with the game for a long time and he’s also not the only individual that offers a prediction. Other pundits on the same programme are all former players who give updates on the games and their predictions are also displayed on the website, when a user is making their prediction. Users cannot go against them however, with their predictions displayed simply to guide them.

The main aspect of Super 6 is to get as many points as possible, not to simply beat Jeff Stelling. Millions of people play the game across the season, so to win it does take a lot of knowhow with prediction. For example, at the time of writing, the person in first place on the leaderboard has predicted 21 correct scores out of a possible 120. It’s clear that even with a high amount of knowledge, which the leader appears to have, it’s still very hard to correctly predict the score. However, the main jackpot of £250,000 is won when all 6 scores on any given gameweek are correctly predicted. This is a very rare occurrence which is clear given the amount of money involved, so much so that Sky will invite the winners to the studio for an interview regarding how they did it. Only a very small handful of players out of millions of players win over a season.

The case of postponed games is also taken into consideration. The jackpot reduces to £20,000 if one game is postponed, with £5,000 being rewarded for two games being called off. If three or less games are played for that week, then the jackpot is voided for that week. Of course, this project isn’t for money purposes, but as matches have been postponed numerous times previously around the winter months many times before so this must be taken into consideration.



Forgetting rounds? Opt-in to receive email reminders

Enter Predictions for Round 26

Premier League — Sat 7th Dec 2019, 3:00pm — AFC Bournemouth 0 v 0 Liverpool

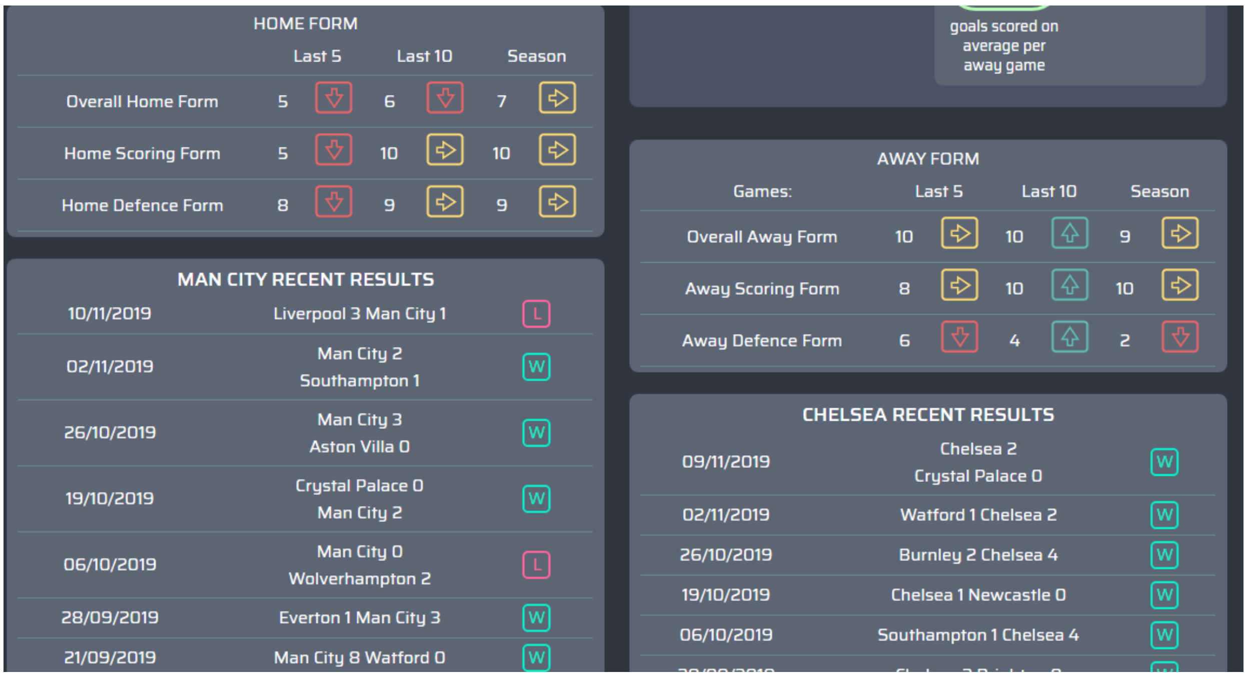
Premier League — Sat 7th Dec 2019, 3:00pm — Tottenham Hotspur 0 v 0 Burnley

Figure 1: Super 6 Predictions page

2.2.2 Predictz.com

Predictz.com is a website that contains predictions for the next five days worth of fixtures. It is, however, a tips application that offers a detailed prediction, but then links to different betting sites and shows the odds to go with or against what Predictz have forecasted. They do take multiple betting sites into account which helps a user to see who the clear favourite is. Due to it being laden with betting promotion, the website is for people over 18 years of age, which makes it a lot less universal than this project. Interestingly however, there is a section on the website that offers a selection of statistics from around the world. A lot of them seem to be football related records such as the Brazilian team Flamengo being unbeaten in 19 games, at the time of writing. This can definitely sway a user into going one way or another regarding where to bet. Of course Premier Predictions has weak links to betting but the similarities in usage of data are similar. A user can see the best form teams in general, at home, away or the best or worst attacking and defensive form at that time. The detail has to be commended but the fact that it’s all geared towards betting makes it less impressive. With that being said, a user cannot simply search a certain team and analyse their form. They must go to that team's next game to look through how they are doing.

For that team’s next game though, the detail behind the website's prediction is impressive. For example, in analysis of one particular big game of the season, Manchester City versus Chelsea at the Etihad Stadium. For context, prior to the international break that preceded this game, City had just lost to the leaders Liverpool and sat in 4th place while Chelsea had won six games in a row in the Premier League and were in 3rd place, one point above their opponents. Chelsea had also won a club-record seven away games in a row in all competitions, but City's pedigree as back-to-back champions couldn't be ignored. All of this information is available to see on the website, as well as the record between the two teams in previous seasons which is important to consider. These are all pieces of data that had previously been planned to be integrated into this project. Links are also supplied to give the user more of a background to the match, but in terms of usability, these links aren't clear to see. Also, when looking at the home and away form as seen in Figure 2 below, the green red and yellow arrows' context isn't clear. For uneducated users, this can appear as quite confusing so the design is something that could be improved to make it more accessible. This is an aspect that is given special attention in this project as the idea is to make it as user friendly as possible for anyone.



goals scored on average per away game

HOME FORM

	Last 5	Last 10	Season
Overall Home Form	5 ↓	6 ↓	7 →
Home Scoring Form	5 ↓	10 →	10 →
Home Defence Form	8 ↓	9 →	9 →

AWAY FORM

Games:	Last 5	Last 10	Season
Overall Away Form	10 →	10 ↑	9 →
Away Scoring Form	8 →	10 ↑	10 →
Away Defence Form	6 ↓	4 ↑	2 ↓

MAN CITY RECENT RESULTS

10/11/2019	Liverpool 3 Man City 1	L
02/11/2019	Man City 2 Southampton 1	W
26/10/2019	Man City 3 Aston Villa 0	W
19/10/2019	Crystal Palace 0 Man City 2	W
06/10/2019	Man City 0 Wolverhampton 2	L
28/09/2019	Everton 1 Man City 3	W
21/09/2019	Man City 8 Watford 0	W

CHELSEA RECENT RESULTS

09/11/2019	Chelsea 2 Crystal Palace 0	W
02/11/2019	Watford 1 Chelsea 2	W
26/10/2019	Burnley 2 Chelsea 4	W
19/10/2019	Chelsea 1 Newcastle 0	W
06/10/2019	Southampton 1 Chelsea 4	W

Figure 2: A brief look into Predictz.com's prediction statistics

The prediction was a score of 3-2 in favour of Manchester City, which was close. The final score ended up being 2-1 to Manchester City. One thing that Predictz seems to be lacking in particular is the chance to go back and check how correct the predictor was. Unless the predictions are saved beforehand, there's no way of knowing how accurate their predictor is. Instead, for further testing, the predictions were inputted into Super 6 for that round of matches, and a result of 7 points (1 correct score and 1 correct result) was recorded, so this shows that the accuracy from the website can be off on occasion, with the week selected for testing potentially one of these occasions.

2.3 Existing Final Year Projects

2.3.1 Szymon Bialkowski, et al (2019) - Predictive Data Scraping Football Hub

This project aimed to develop a fault-tolerant application that relied heavily on Machine Learning techniques. The data aspect is huge and offers different methods into predicting football games and visualising data. A lot of different technologies were considered but it was mainly completed through Python. Some of the databases and servers sounded unfamiliar which is why almost all technologies and languages used in this project are well-known and previously used. A RESTful API was used as well, which is a crucial foundation to this project as well. This project incorporated the use of the SportMonks API and utilised one of the paid subscription options in order to use it properly. The student cited the documentation as being a big help as it saved a lot of time. The final product ended up being a mobile application that contained rich statistics for teams, players and managers alike.

2.3.2 John Kiernan, et al (2015) - Football Simulator

The goal for this project was to go in-depth to try and create a match predictor that is basically perfect. The final application allows users to test theories regarding a team's best player playing in a completely different position. The detail in which it goes into is impressive. Users can pick two teams to go against each other, edit the tactics and see who is the winner based on the logic of the app. A lot of different variations of different formations are analysed, as well as the minute detail that can go into a football match. The work which is done here has been completed by the developer themselves by the developer themselves without the use of an API. User accounts had been initially planned, as well as the ability to edit players and teams regarding transfers, for example but such was the detail that needed to go into the basic functionality of the project that these user accounts had to be omitted. In terms of programming, the majority of it was done through Java servlets and JSP. The project was deployed as a web application. jQuery and AJAX were used to clean up the design aspects. The developer appears to highly recommend the use of servlets for a web application. In the end the predictor's best test was 75.9% accurate, with stats from goals scored to pass accuracy being included. The worst case was 49.4% accurate, but with that being said the result was correct, it was simply the detail that let the simulation down. Nonetheless, it's still impressive to have the worst result having that high of an accuracy rate.

2.3.3 Ashley Fitzgerald, et al (2019) - Predicting the price of used cars using Machine Learning

This project disregards the sporting aspect completely and interestingly uses very similar techniques for a more real-world setting. The developer also created a web application, with the use of Django. The data that was needed for the project to function properly included over 1,200,000 listings of used cars in the USA. This was because using an Irish data source would give off fewer results and therefore less detailed data. It appears that the data source was not an API, but a dataset that could be easily downloaded and manipulated for the app. The algorithm to predict future prices still had to be implemented however, as well as the fact that the amount of data was so vast that it had to be munged to erase outliers for example. A user would enter the year, mileage, state, manufacturer and model of a car from a variety of options and then would get an estimate on the price of that particular car based on the data the app uses. User accounts were also implemented but it's not clear whether users could keep track of previous searches or add their favourites to a list or not.

2.4. Technologies researched

2.4.1 Programming languages

Python

Python is an object-oriented programming language that has emerged over recent decades as a major tool in scientific computer purposes. It has numerous libraries available that can work with data. Pandas, NumPy and Scikit-Learn are a few examples. It fits for this project for exactly that reason. The data that has to be munged and then used in a prediction algorithm is vital to the whole structure of the application. It is also integrated into the Django framework so all functionality within the app itself can be done through Python, although other languages were examined throughout the process.

JavaScript

JavaScript could provide the bedrock for a variety of functions that can be carried out on the web application. More importantly in the whole context of the project, it supports the use of machine learning models. JavaScript as a language is highly popular in terms of populating frameworks. Its use in running servers through Node.js is also a big helping hand for the application.

JSON

Much of the data that is dealt with from APIS is communicated through JSON. JSON is used in tandem with JavaScript as an object notation. Data that's received in JSON can be transformed into JavaScript or even Python as has been tested. Large amounts of data are written in JSON on APIs so for this project that data has to be sorted and converted into something readable and more importantly, programmable.

2.4.2 Data Source

API

An application program interface (API) is a set of routines, protocols, and tools for building software applications (6). APIs generally contain mass amounts of data that can be accessed through JSON requests. For the project, an issue presents itself however. Numerous avenues which are discussed below were explored in the hope that one would work well with the application. Unfortunately locating one that suited and also didn't cost to use was challenging. Ideally there should be no costs involved, but some costs may be occurred in development of an application like this due to the data required and the popularity of football worldwide.

The prospect of scraping the data from an API was also considered. This involved using the Postman software to not only save requests, but save the responses to a JSON file on my own machine. This by itself can be manipulated by the Python code but the possibility of using Excel to print all of the data in formatted columns into a .csv file was also a possibility. From here, the data can easily be accessed and used within the code using Pandas. The main downfall of this however is having to navigate through the whole API to try and get all data required.

This was trialled out to see how it would work. Using SportMonks[3] football API that was tailored towards the Premier League's historical data, an attempt was made within a free trial period to get as much data required as possible. This in itself required a structured plan as it had to be decided what aspects of data are really needed for the application to reach its full potential.

Unfortunately as more data was scraped through Postman, it became more difficult to sort into CSV format from JSON and with time becoming more limited, the use of an API had to be abandoned to simplify the task at hand.

[3] https://www.sportmonks.com/

Datasets

Luckily there was an alternative available to utilise. Through the Football-Data[4] website, CSV files that contain match data for every season all the way back to 1993/94 were available. A fair measure that protracts the progression or regression of each team in the league was required, which is why nearly 13 seasons worth of data has been used. The datasets contained simple information that could be manipulated into greater detail that would augment prediction accuracy. They also contained betting odds which were duly erased given the scope of this application. Other datasets were manually created that would work in tandem with these existing ones to improve the predictor.

2.4.3 Database

MongoDB

MongoDB works as an open-source database that was created with a scalability purpose. It is often used when working with big data, which is what the application involves. Creating a database and populating it within the command line interface is straightforward. The JSON format is supported in it so the JSON files from Postman could have been inputted easily.

Firebase

Firebase is another possibility to use for this application due to it working well as a back-end database. The current prototype makes use of Firebase given its suitability for user authentication but could be subject to change. Firebase is a realtime NoSQL database that operates with the cloud. When offline, a user can try to store data, and then Firebase will insert it once an Internet connection is established. This is an interesting feature as users would have been able to make their predictions offline, and not have to worry about obtaining a network connection in time for that week's deadline.

SQLite

Ultimately after choosing Django as the framework, SQLite was the obvious choice. It’s built into Django projects by default, and after developing most of the functionality, there was no cause for change. SQLite can easily be accessed, by logging in through the admin page of a Django project. Once inside and instances are created via Django’s shell, data can easily be added and deleted.

[4] http://www.football-data.co.uk/data.php

Printing the necessary data from this database was very simple as well with the links within the framework.

2.4.4 Frameworks/ Libraries

Django

Django is a high-level framework that works together with Python, which straightaway brought it into serious consideration for the project. It’s quickly set up with minimal code and can aid in the quick development of a web application. It stands out amongst others, such as Ionic or AngularJS because of its relationship with the main programming language of the project, as well as its application development speed. It ticks a lot of boxes and the way it ties Python code together with HTML and back-end databases is unparalleled.

Pandas

Pandas is a Python library that can supply well-visualised data structures along with data analysis methods. Processing the data from the datasets into the predictor can be a tedious process. Pandas is particularly useful in helping to sort data as well as to make observations of how the data is spread out, for example.

NumPy

Arrays that can be taken straight from a dataset to be manipulated are formed using NumPy. Some machine learning models only take an input from an array so NumPy certainly comes in handy when testing out different algorithms.

Scikit-Learn

Scikit-Learn contains a large variety of different ready-made machine learning algorithms. Both supervised and unsupervised problems are catered for. While some algorithms can be hard-coded into a script, Scikit-Learn can use them as imports where data is taken in and outputted in a minimal amount of code. For this project, it's particularly useful as a number of different algorithms are tested out, with the most accurate algorithm for both the training and test set being used. The hyperparameters within some of these algorithms could be altered to try to increase the accuracy on the test set as much as possible.

2.4.5 Machine Learning Algorithms

Decision Trees

A root node, interior nodes and leaf nodes make up a tree, where the root and interior nodes depict a test that is done on a particular feature of a query. The leaf nodes then depict a possible predicted classification for that query. They work together with the entropy to help determine the probability of an event. In essence, a high probability yields a low entropy and vice versa. Decision trees are built into the Scikit-Learn library in Python, so integrating it simply requires importing it from the library and using it. Essentially, it's a set of back-to-back questions posed within the data (7). The input is judged through these trees and eventually an output based on the decisions made is created.

Support Vector Machine

Training this model requires a decision boundary, otherwise known as a separating hyperplane, that will best disperse the different levels of a target feature as the maximum margin. The training set instances are plotted on the margin extents therefore defining the margins. The margin is known as a hyperplane, which essentially separates and classifies the data that is inputted (8). These are the support vectors and represent the decision boundary. Weights can then be added accordingly to the decision boundary.

Logistic Regression

A logistic regression models output is a dependent variable that has a limited number of possible values. It tends to work best when the output data must be categorical in nature. In the case of the projects predictor, the output is categorical as the results would be win, lose or draw. The data is fitted to a logistic curve, which is why it is used a lot in medical science, as it could predict whether a person could have a heart attack within a certain time frame based on information such as a person's age, gender and overall health. For this project, it appears to suit well as a future match result based on historical statistics is sought after.

XGBoost

Extreme Gradient Boosting is a model that is growing in popularity within applied machine learning (9). It constructs a model as a collection of weaker models. For example, a number of decision trees back to back. For example, an input could be a person's details, and then a tree could decide whether they like something would be based on their age which gives a prediction score. Their frequency of use of that same thing can be calculated with another tree, and then both scores are added together. Put simply, with more trees, the higher the score will be, as more of the model will be integrated. When applied to a training or test set, if the accuracy isn't satisfactory enough, the hyperparameters within the model can be altered specifically to tailor to a particular prediction problem (10).

2.4.6 Integrated Development Environments (IDEs)

Jupyter Notebook

Jupyter is a highly effective development environment particularly for visualising data. Code can be executed in blocks and to see how datasets or dataframes are looking at a given time, without the need of a print statement. In that respect, it's very efficient in developing the predictor as particularly in the data cleansing phase, visualisation is key when analysing columns and rows within dataframes. It can also work alongside a Django kernel which can be a big aid for taking user input.

PyCharm

PyCharm is known as the most-popular IDE for Python (11). Created by JetBrains, it was initially used when JSON requests for the Sportmonks API were an aspect for the project. Once the CSV datasets were secured however, it was made redundant. Jupyter Notebook files can be exported as .py files which can then be run in PyCharm but there would be little point, given that the predictor's results are printed to a CSV file which can then be printed to the application.

Sublime Text

Sublime Text has various different uses. For this project it's where the web-app is developed through Django. The whole project can be dropped into the sidebar where all files can be accessed with ease. Dual windows are also allowed, unlike other IDEs of a similar nature such as Notepad++. Python files, HTML files and CSS files alike can be used at once in this environment making it the clear choice to construct the main application.

2.5. Conclusions

When comparing each of these projects and systems to Premier Predictions, it's very clear that there are a lot of similarities and not many differences between them when analysing individually. Most interestingly is the fact that only one of them has a definite use of an API as opposed to datasets as CSV files. The data in football can be extremely detailed compared to that of cars, for example.

Out of the 5 systems studied, only 2 of them carry clear user integration with them. Super 6, which is most like this project, and the Football Simulator allow the login of a user and monitor their progress somewhat during their use of the app. The user experience is essential for this project, as is its interaction with the implemented data. In saying that, the game aspect that is included is also synonymous with Super 6 and Football Simulator. Football Simulator allows the use of picking teams and having them compete against each other based on each of the players stats, much like the FIFA video game simulator. Super 6 is similar to the standard Fantasy Football where users compete against each other and there is one overall winner. This is a lot more similar to this project as users will be able to create leagues against their friends while taking on the predictor.

Most notably, is that all of these systems use statistics somewhat. While some of them don't have an API, statistics themselves are key in what they are trying to achieve respectively. Predictz.com is the most similar to Premier Predictions in this respect as it shows the statistics behind the prediction that is made for a match. The Manchester City vs. Chelsea example mentioned (P15) proves this as data ranging from head-to-head results to historical facts are included. This prediction turned out to be accurate as well given that City did win by a one-goal margin, but not the exact same score.

	User Integration	API use	Dataset use (CSV files)	Prediction using statistics	Phone app	Web app	Game aspect (user vs. predictor)
Super 6	🟩 ✓	⬛	⬛	🟫 ✓	🟩 ✓	🟩 ✓	🟩 ✓
Predictz.com	🟥 X	⬛	⬛	🟨 ✓	🟥 X	🟩 ✓	🟨 ✓
Data Scraping Football Hub	🟨 ✓	🟩 ✓	🟥 X	🟩 ✓	🟩 ✓	🟥 X	🟫 ✓
Football Simulator	🟩 ✓	🟥 X	🟩 ✓	🟨 ✓	🟥 X	🟩 ✓	🟩 ✓
Predicting used car prices	🟫 ✓	🟥 X	🟩 ✓	🟩 ✓	🟥 X	🟩 ✓	🟥 X
Premier Predictions	🟩 ✓	🟥 X	🟩 ✓	🟩 ✓	🟥 X	🟩 ✓	🟩 ✓

Figure 3: A comparison between all systems researched and this project

3. Experiment Design

3.1 Introduction

This chapter looks at the original concept designs of Premier Predictions. Various software methodologies used to map the development of the project are discussed as well as the overall design. Further to this, there is particular focus on the user. Their experience throughout the usage of the application is documented including multiple screens that can be visited on the app, as well as thoughts behind the design of the screens. The screen's functionality is put under scrutiny, as well as the system architecture, covering both front and back-end parts of the application. The ideology behind how the project will be tested and evaluated will also be discussed.

3.2. Software Methodology

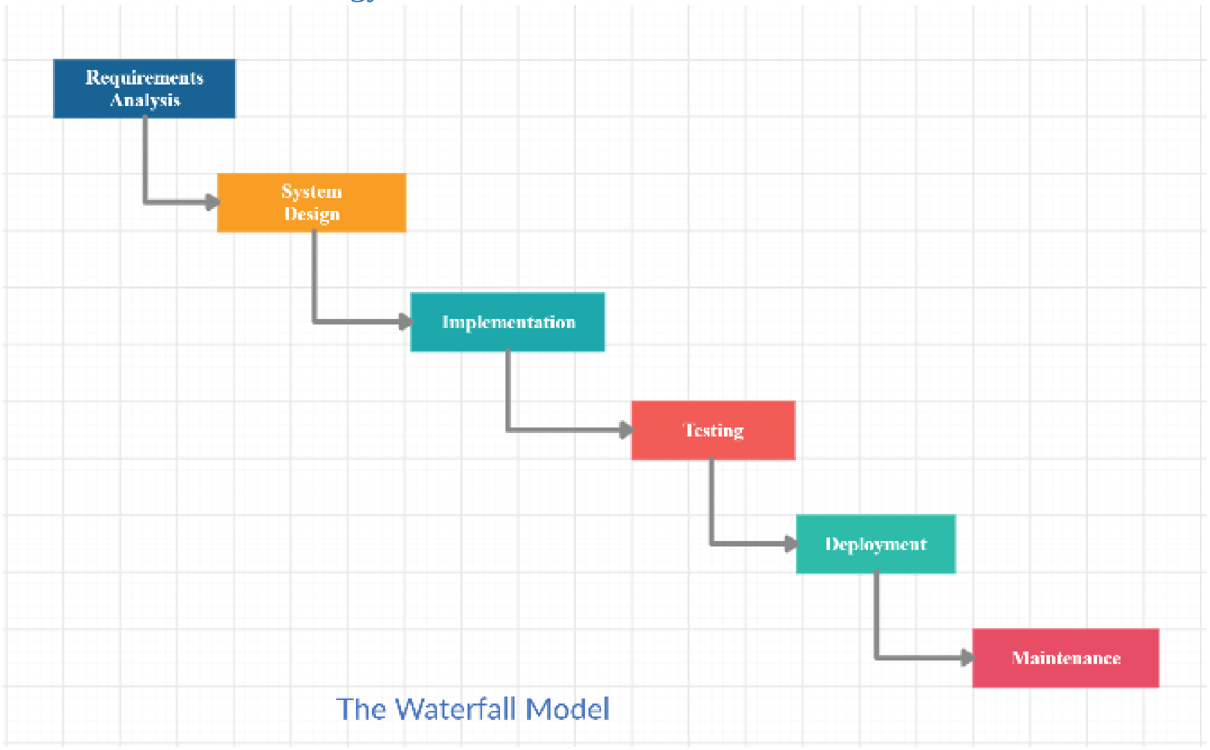


Figure 4: The Waterfall Model

The Waterfall Model is a six-step methodology that can be used in the development cycle of software. It follows logical steps which matches well with the development of Premier Predictions. It is of paramount importance to have organisation like this in the structure of development. It gives a direct focus for the scope of the project and keeps it all within the context. All requirements are analysed and considered before designing the system based on the results, then they are all implemented and ready to be tested. The application can then be deployed based on the success of testing and is then maintained from there onwards. The testing aspect is a crucial point, however, as the entire system is under scrutiny. User feedback could end up overhauling aspects of the project 2and could see earlier iterations of the Waterfall Model revised once more. It’s difficult to know what requirements are necessary before the testing stage, but something has to be in place to be critiqued by users.

3.3. Overview of System

The project is a 3-tier model , including a presentation layer, application layer and a data layer. In terms of development the Waterfall model will be considered, with the main variables being before the Deployment stage. The development itself involves a number of steps; particular features need to be working to full capacity before moving to the next one. For example, when training the predictor it’s essential to have one or two parameters working before adding more detailed parameters to it. This overall process results in the final deliverable but begins with basic structural decisions such as the communication between layers. The approach can be seen below.

1. Design a web-based application hosted on a server and create the login and register service, allowing for direct communication between layers.
2. Integrate feature. (e. g. using previous years’ Premier League data)
3. Test feature. (e. g. are predictions based on the data accurate?)
4. Repeat 2 and 3 for each feature and implement results into the application.

The predictor becomes more unique with more features added to it, as does the project as a whole. For this project features could act as parameters, as the predictors accuracy increases with added parameters. The primary feature regarding the predictor would be the example used of the previous season’s Premier League data (2018/19) that is also part of the end result with all the other seasons included.

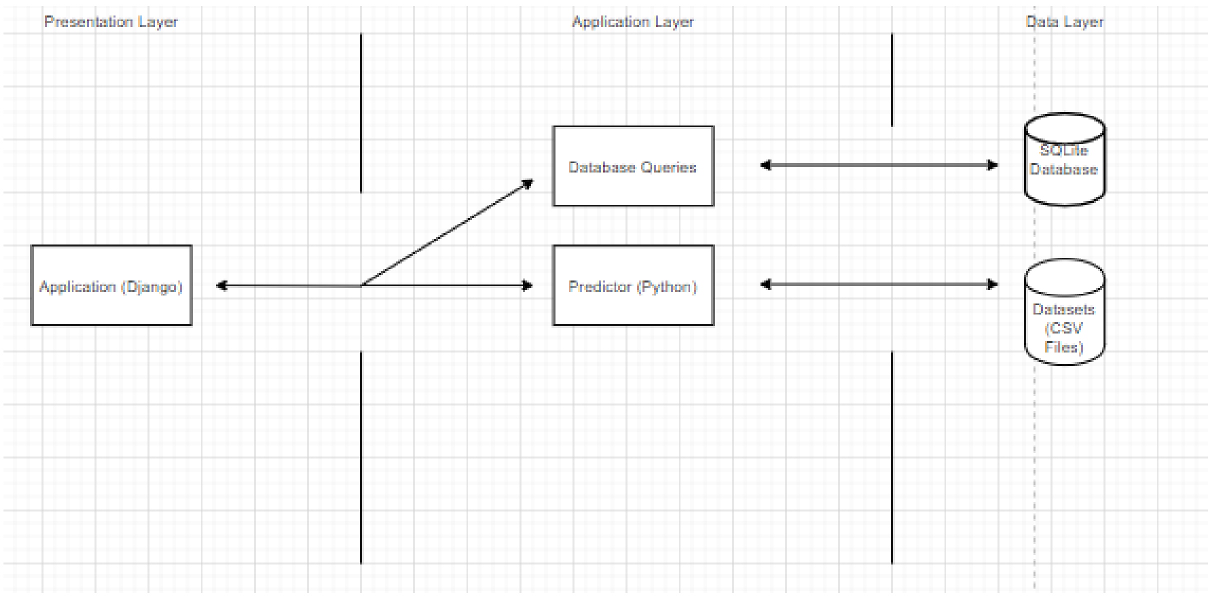


Figure 5: Overview of the system architecture

3.4. Front-End

The front-end tier is essentially the presentation layer. This is what the user sees and experiences throughout their use of the application. They can log in, make predictions, create a league or join a league and check upcoming fixtures and previous results as well as their own statistics through the application's interface.

A basic skeleton prototype was created based on what the user should see throughout their use of the app. It gives an idea of the layout and design of the final project regarding placement of buttons and spacing to name a few aspects. Nielsen's Heuristics (12) were taken into consideration here as the usability of the application is just as important as any other component.

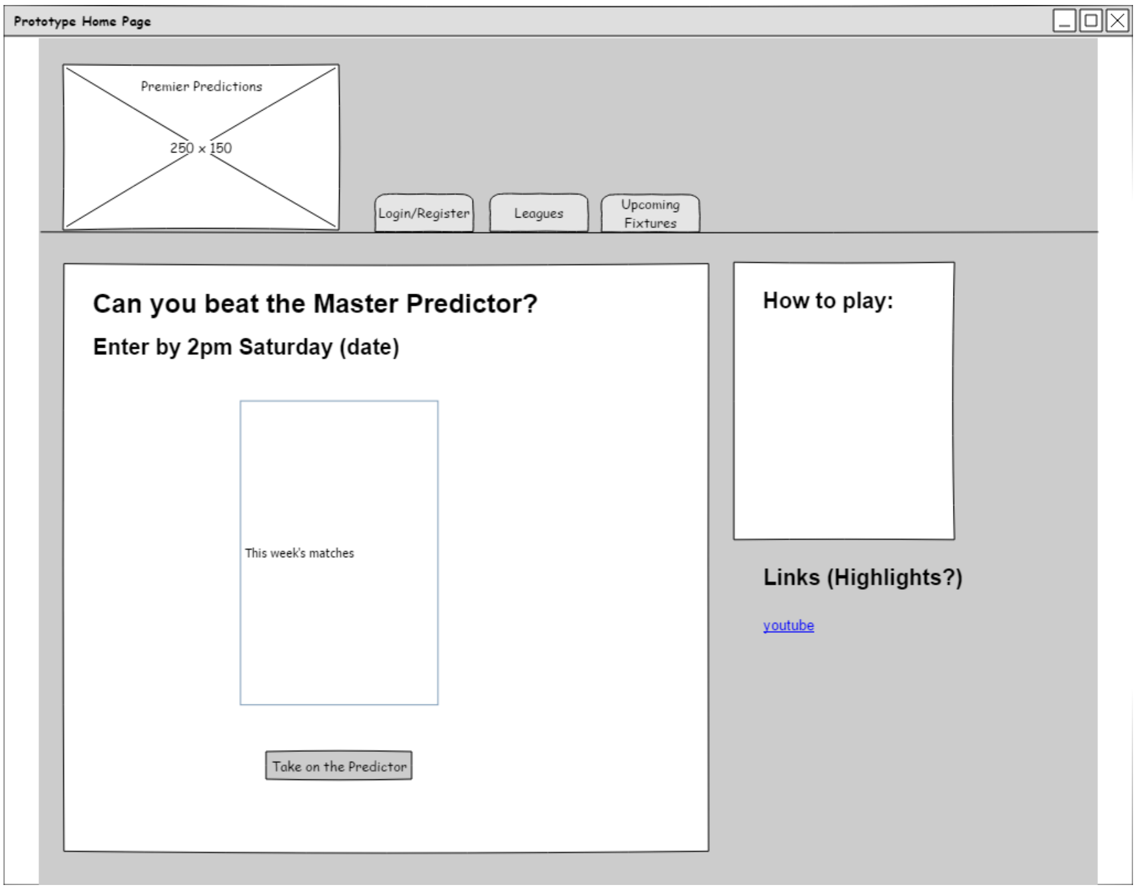


Figure 6: Prototype Home Page

The home page is what the user will first encounter. They are faced with the rules of the game and the fixtures of that week to entice them to either create an account or get their predictions in time for the upcoming games. The deadline for submissions would be just before the first match for that week kicks off, and also gives the date and time to help the user remember to take part. In the corner, a YouTube playlist of recent Premier League highlights is displayed to again try and entice users to avail of this system. This acts as a welcome screen as it is the only page along with the other options on the navigation-bar that can be accessed without signing in.

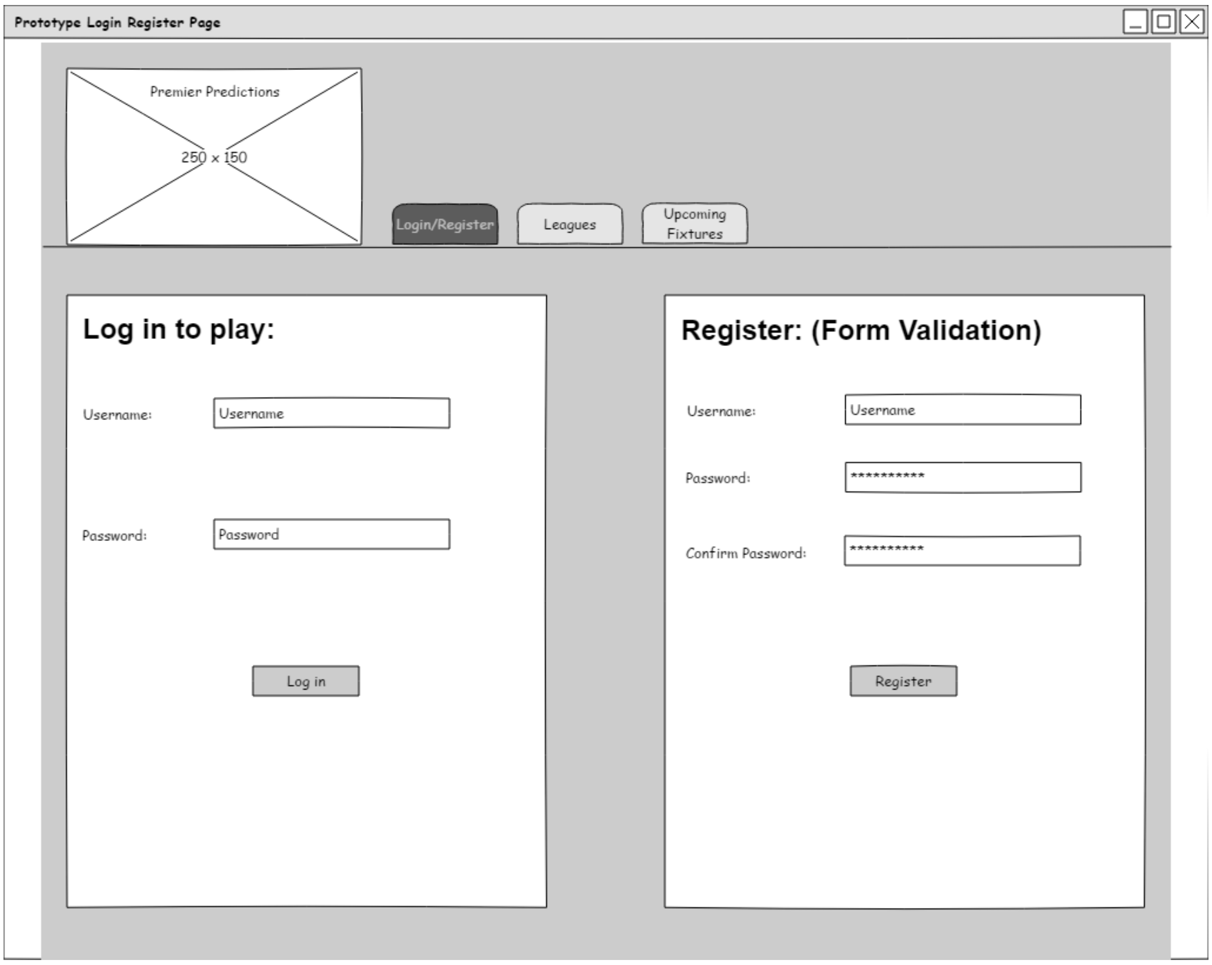
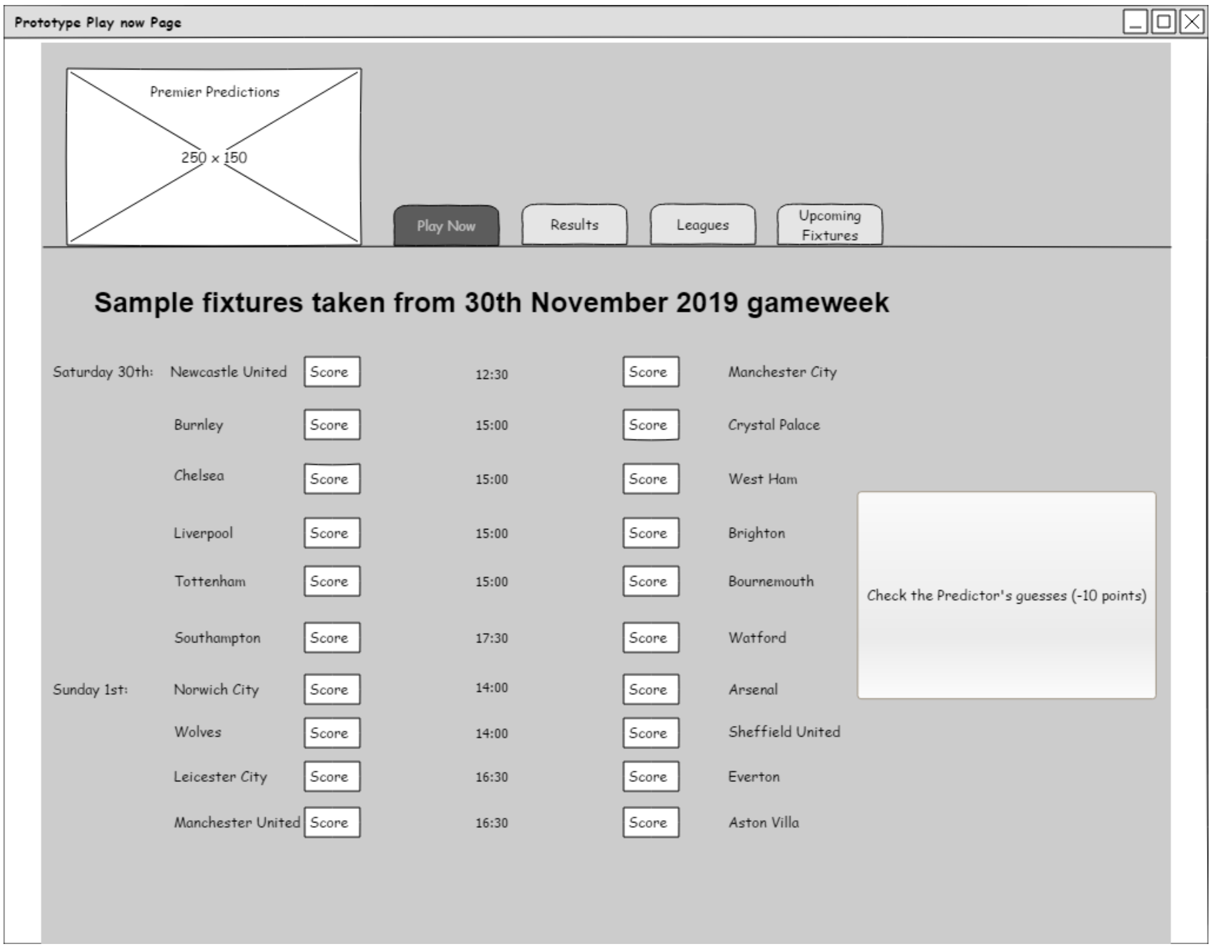


Figure 7: Prototype Login/Register Page

The logical next step would be for the user to login or register an account. Despite only a username and password being necessary to create an account, form validation is in place to ensure that users cannot have the same name. The password also has to meet the required strength to be used. From here, users can still access the home page by clicking on the logo as well as the leagues page and upcoming fixtures page. For security reasons an email could still be required. For example, when changing password an authentication email could be sent. This can be implemented easily in Django.

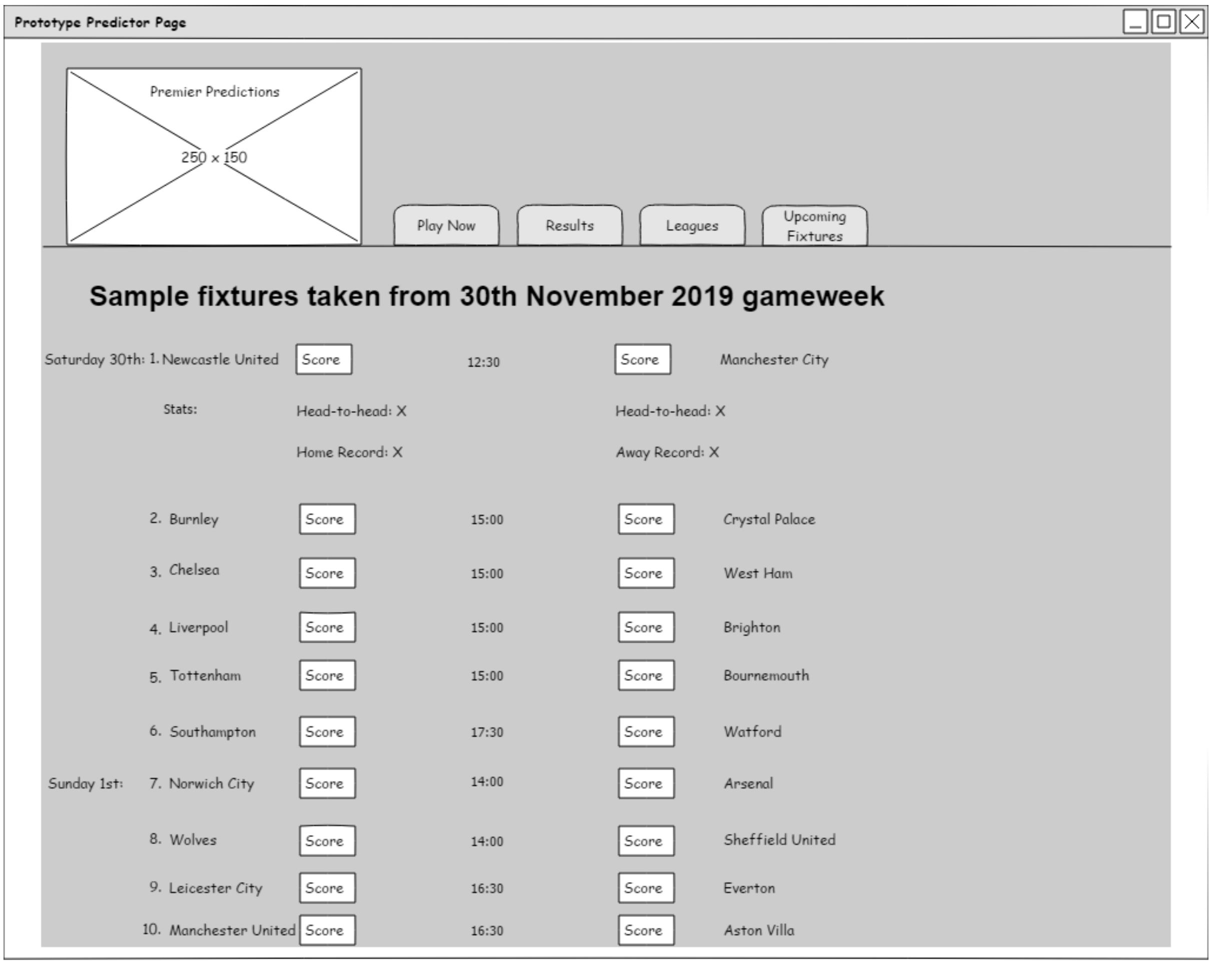


Sample fixtures taken from 30th November 2019 gameweek

Saturday 30th:	Newcastle United	Score	12:30	Score	Manchester City
	Burnley	Score	15:00	Score	Crystal Palace
	Chelsea	Score	15:00	Score	West Ham
	Liverpool	Score	15:00	Score	Brighton
	Tottenham	Score	15:00	Score	Bournemouth
	Southampton	Score	17:30	Score	Watford
Sunday 1st:	Norwich City	Score	14:00	Score	Arsenal
	Wolves	Score	14:00	Score	Sheffield United
	Leicester City	Score	16:30	Score	Everton
	Manchester United	Score	16:30	Score	Aston Villa

Figure 8: Prototype Play now Page

Upon signing in, the user is instantly taken to the Play Now page, where they can use all the application has to offer. This page displays all of that week's fixtures, as well as a field beside each team in which the user can input the amount of goals they think the respective team will score in that game. To the right is a button which takes the user to the predictor's guesses for that week. The aim is to beat the predictor, so this comes at a cost of 20 points to the user. This was initially meant to be 10 points as seen in the above prototype page, but it was then thought that the impact to a users points had to be greater to discourage its use even more. This will mean that users will have to take a hit in order to do very well, much like Fantasy Football. These inputted values are stored in a database and will be overwritten once the next gameweek's predictions are entered.

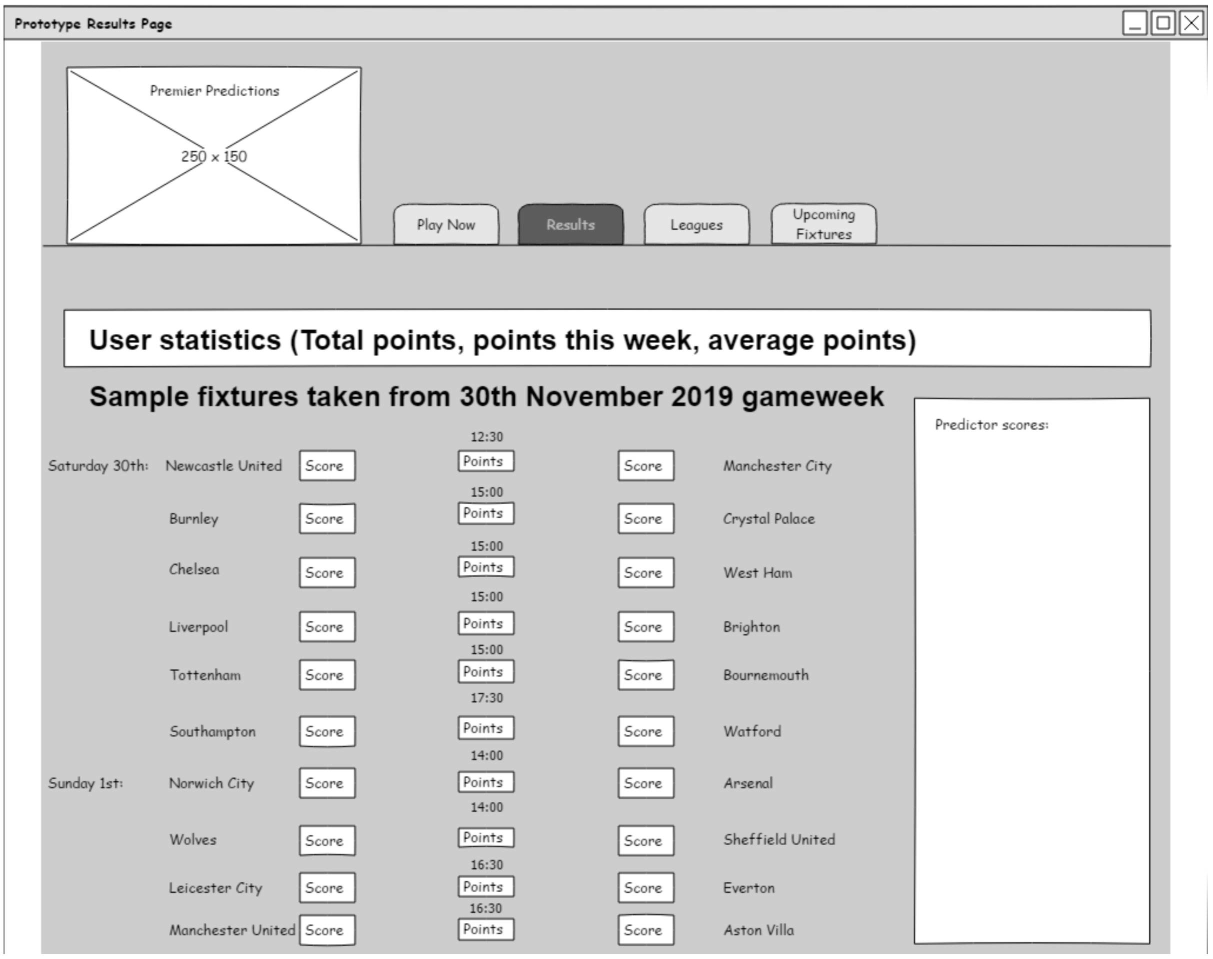


Sample fixtures taken from 30th November 2019 gameweek

Saturday 30th:	1. Newcastle United	Score	12:30	Score	Manchester City
	Stats:	Head-to-head: X		Head-to-head: X	
		Home Record: X		Away Record: X	
	2. Burnley	Score	15:00	Score	Crystal Palace
	3. Chelsea	Score	15:00	Score	West Ham
	4. Liverpool	Score	15:00	Score	Brighton
	5. Tottenham	Score	15:00	Score	Bournemouth
	6. Southampton	Score	17:30	Score	Watford
Sunday 1st:	7. Norwich City	Score	14:00	Score	Arsenal
	8. Wolves	Score	14:00	Score	Sheffield United
	9. Leicester City	Score	16:30	Score	Everton
	10. Manchester United	Score	16:30	Score	Aston Villa

Figure 9: Prototype Predictor Page

This predictor page can only be accessed through the Play Now page, as ideally the predictor's guesses and factual reasoning behind these statistics would be kept under wraps. The goal is to beat the predictor so seeing its logic behind predictions would defeat the purpose to an extent. By clicking on one of the fixtures all of the data behind the prediction is displayed. Given that a correct result is one point and a correct score is three, giving up 20 points to see this information shouldn't be seen as worth it.


Prototype Results Page

Premier Predictions

250 x 150

Play Now | Results | Leagues | Upcoming Fixtures

User statistics (Total points, points this week, average points)

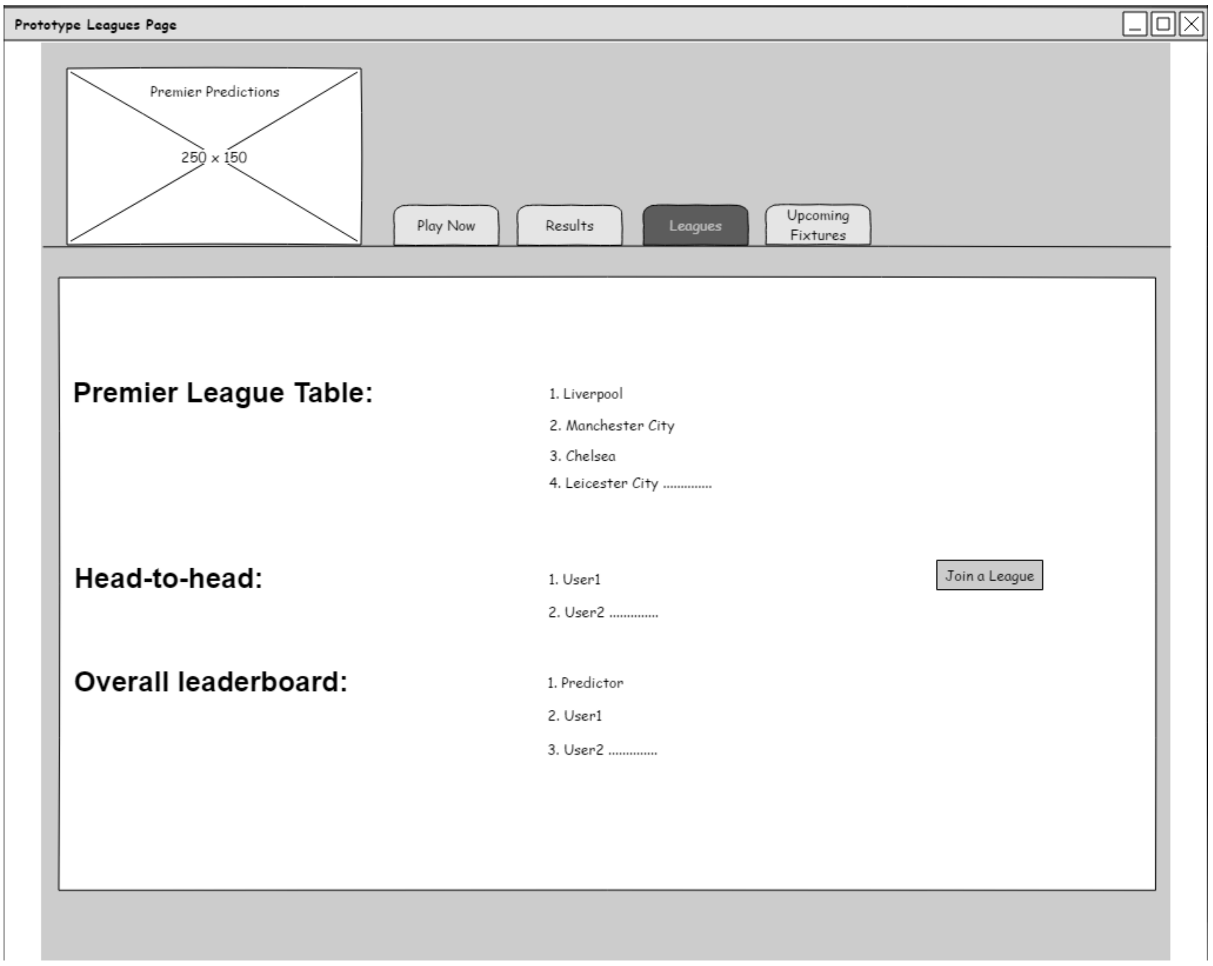
Sample fixtures taken from 30th November 2019 gameweek

Saturday 30th:	Newcastle United	Score	12:30 Points	Score	Manchester City
	Burnley	Score	15:00 Points	Score	Crystal Palace
	Chelsea	Score	15:00 Points	Score	West Ham
	Liverpool	Score	15:00 Points	Score	Brighton
	Tottenham	Score	15:00 Points	Score	Bournemouth
	Southampton	Score	17:30 Points	Score	Watford
Sunday 1st:	Norwich City	Score	14:00 Points	Score	Arsenal
	Wolves	Score	14:00 Points	Score	Sheffield United
	Leicester City	Score	16:30 Points	Score	Everton
	Manchester United	Score	16:30 Points	Score	Aston Villa

Predictor scores:

Figure 10: Prototype Results Page

Here the user can see the final score in each game, as well as the amount of points earned from them. Also displayed here, above the final scores, are the users own personal statistics. The total points they have gotten so far, their count for the week and average points per week can be seen. To the right, the predictors points won from each match are displayed, but nothing else. Anything the predictor guesses can only be seen on the predictor page.



Premier League Table:

1. Liverpool
2. Manchester City
3. Chelsea
4. Leicester City ....

Head-to-head:

1. User1
2. User2 ....

Overall leaderboard:

1. Predictor
2. User1
3. User2 ....

Figure 11: Prototype Leagues Page

The leagues page is one of the few that can be accessed without the use of an account. Here, the Premier League table at the time is depicted as well as any leagues a user may be in, if they are signed in. An overall leaderboard can also be seen which includes all users and the predictor itself. The user has the option to join or create a league here to face off against other particular users. This option isn't displayed if the user is not signed in. The league table is there to guide the user somewhat in their prediction.

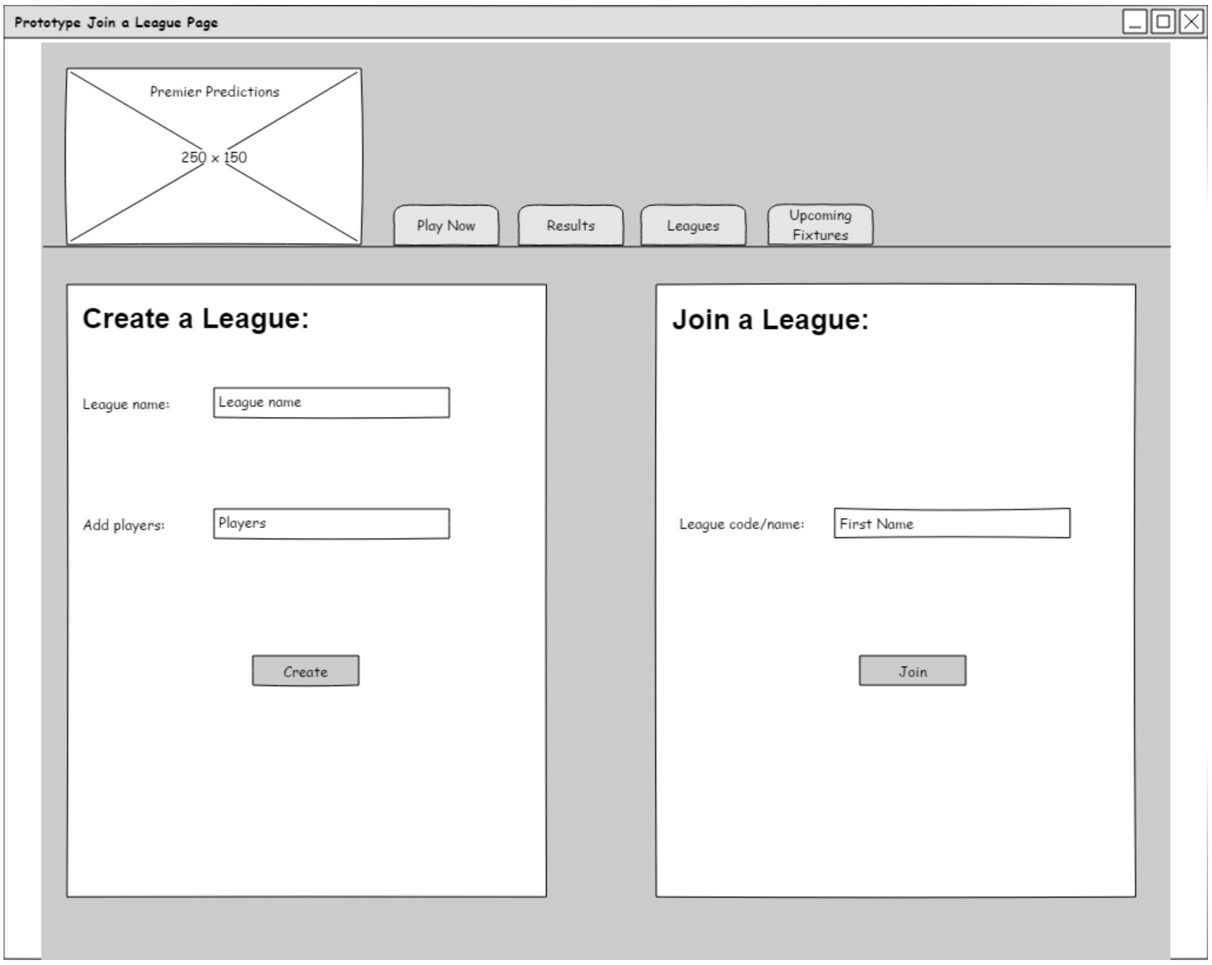


Figure 12: Prototype Join a League Page

The user is taken to the page shown above when the "Join a League" button is clicked. To create a league, the user has to give their league a name and then include all the users that will be part of it. A code will then be generated, and other users will be able to join the league either by inputting this code or the name of the league. This is a similar system to that of Fantasy Football's and adds to the overall user experience.

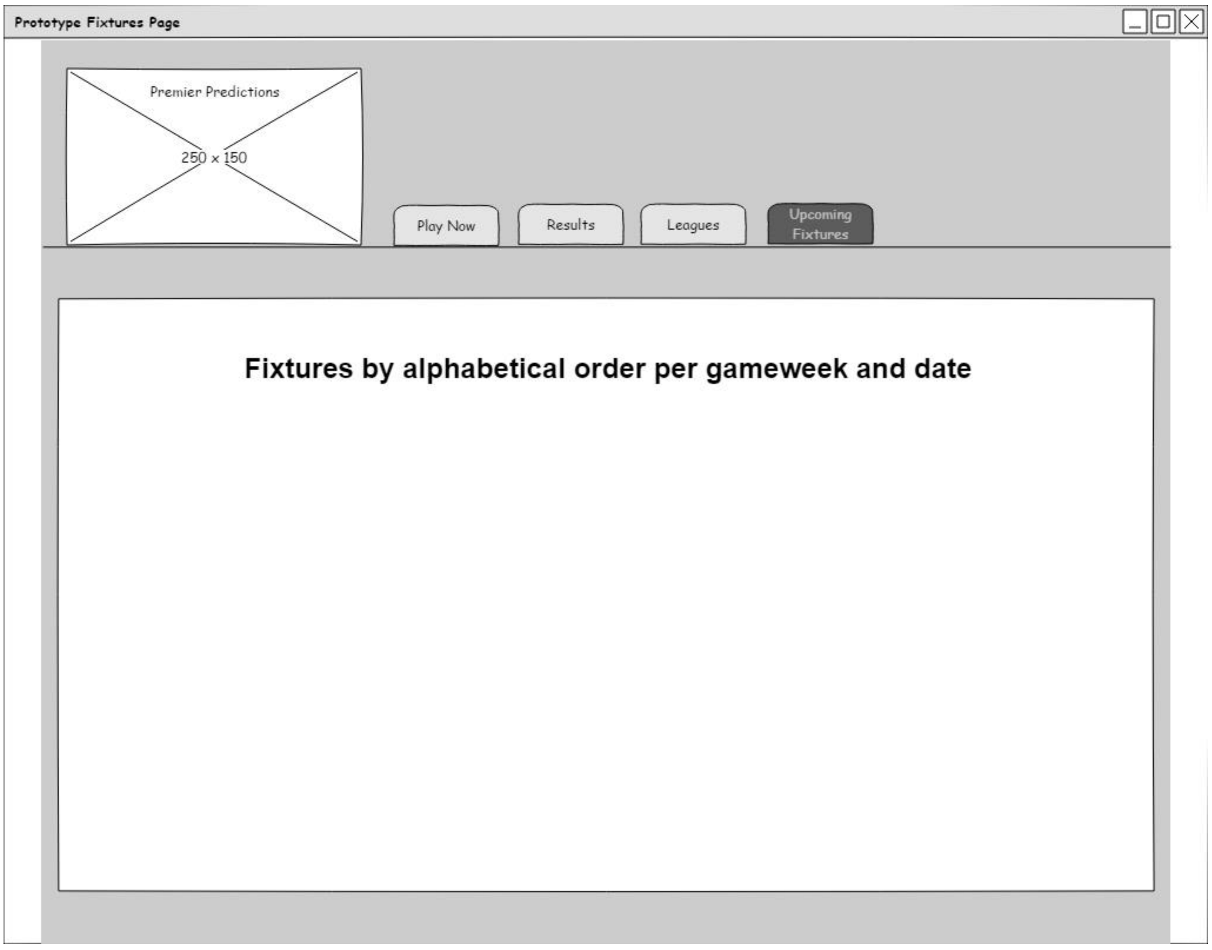


Figure 13: Prototype Fixtures Page

The fixtures page simply shows upcoming games in the Premier League. Again, this is here to try and help the user with their prediction as they can see if a team has a match against a team higher than them in the league coming up. It also acts as a reminder, as there are some weeks in which league games are played during the week as well as at the weekend. International breaks can be noted here too where there are no games on for around two weeks.

Use Case Diagrams

Use Case Diagrams are generally used to depict what the system can do as well as track the path of a user. The below diagrams outline the progress of the system's functionality and of what the user can do.

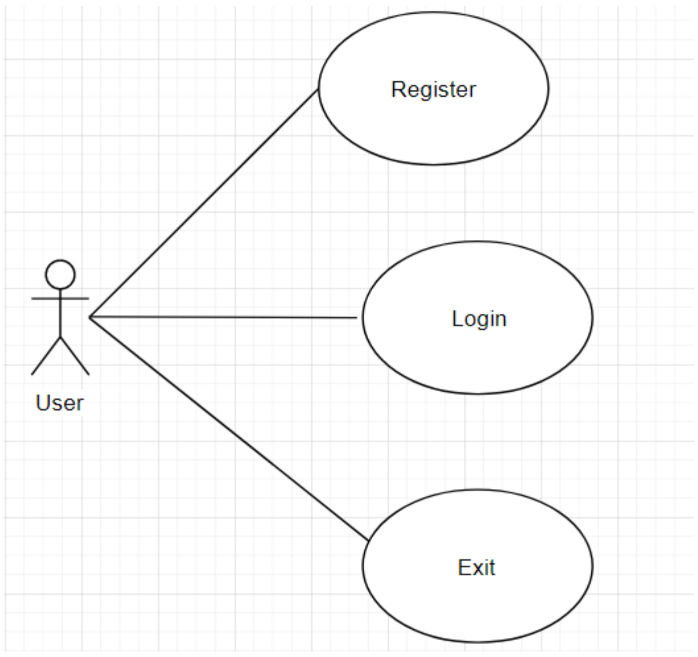


Figure 14: First iteration Use Case Diagram

Initially, the user can register, login if they already have an account, or vacate the application if logged in. This is the absolute basic functionality for the system.

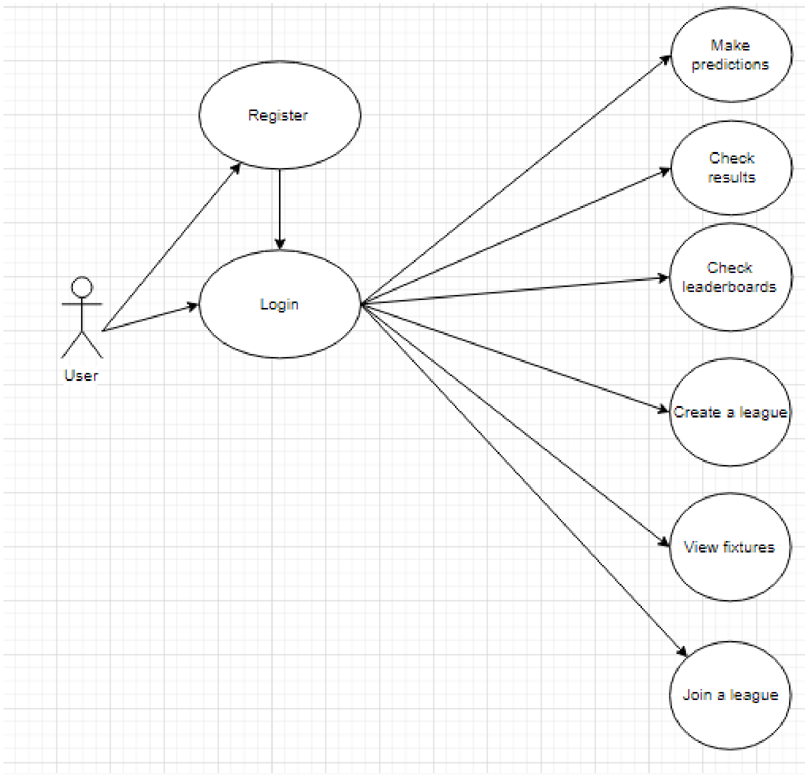


Figure 15: Second iteration Use Case Diagram

More options are available to the user here. After registering and signing in, they have full access to the application. All of these use cases are essentially pages with different purposes that can be visited.

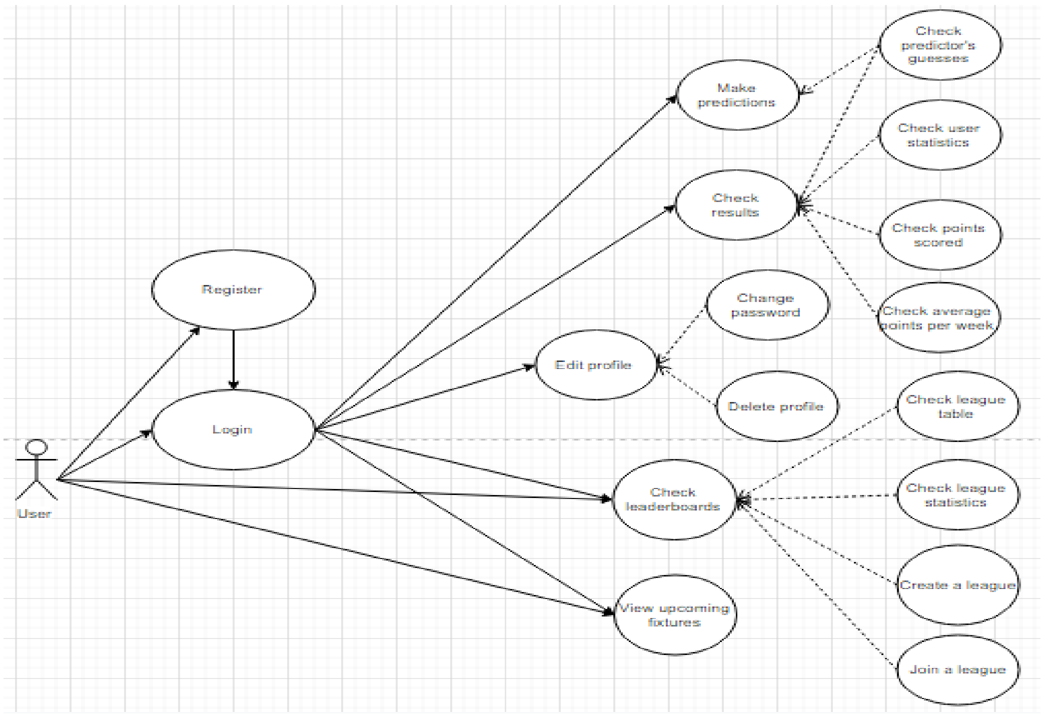


Figure 16: Final iteration Use Case Diagram

All possible use cases are shown here. Every functionality on each page can be seen as the full capabilities of the application are laid out.

3.5. Middle-Tier

The application layer contains all of the logic that is used to meet the final project's requirements. This is code used to program the majority of the system is located as well as other software components. All scripting utilised to set up and test the predictor is also here. The functionality required to use an account is provided here, including making predictions with the account and the statistics for that account. The algorithm in place to take data and predict outcomes is an example of the capabilities of this middle-tier.

3.6. Back-End

The back-end acts as the data layer. This is where all data used for the project either from the various datasets are located. It can only be accessed through what is in the application layer.

The SQLite database containing components such as user data and data that doesn't change (e.g. the list of teams in the league). Some of the data however, such as the list of fixtures, is constantly evolving as each week goes by. This is different in respect to the data used to create the predictor. SQLite is chosen as it's readily built into and works seamlessly with Django. A backup database is also a possibility in the case of something happening to the SQLite database. This could also fulfil security purposes as the loss of user accounts can be prevented in the event of SQL injection amongst other possible attacks.

The CSV data represents most of the data that is being constantly added to. For example, a list of previous results for a team is going to change repeatedly, each time that team plays another match.

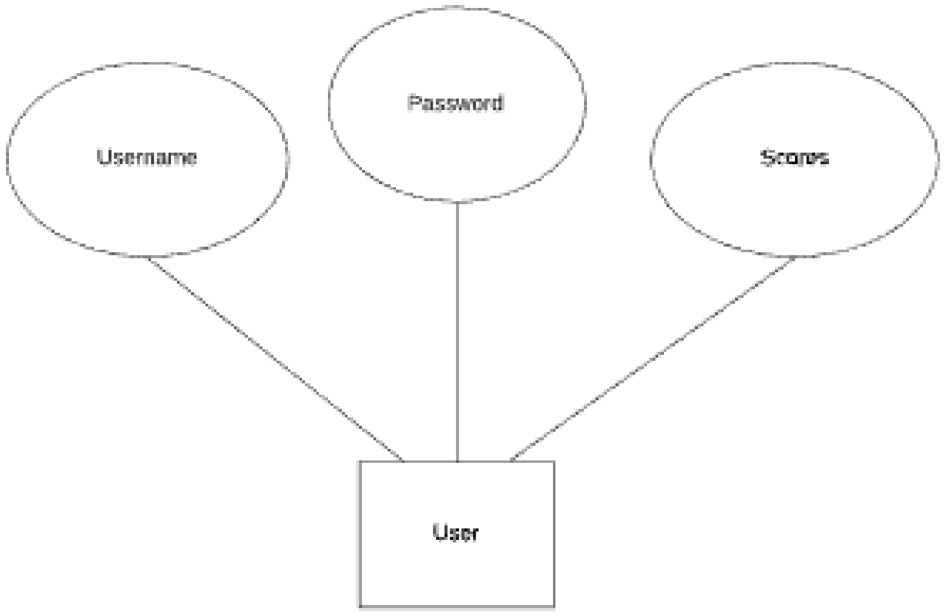


Figure 17: First iteration Entity Relationship Diagram

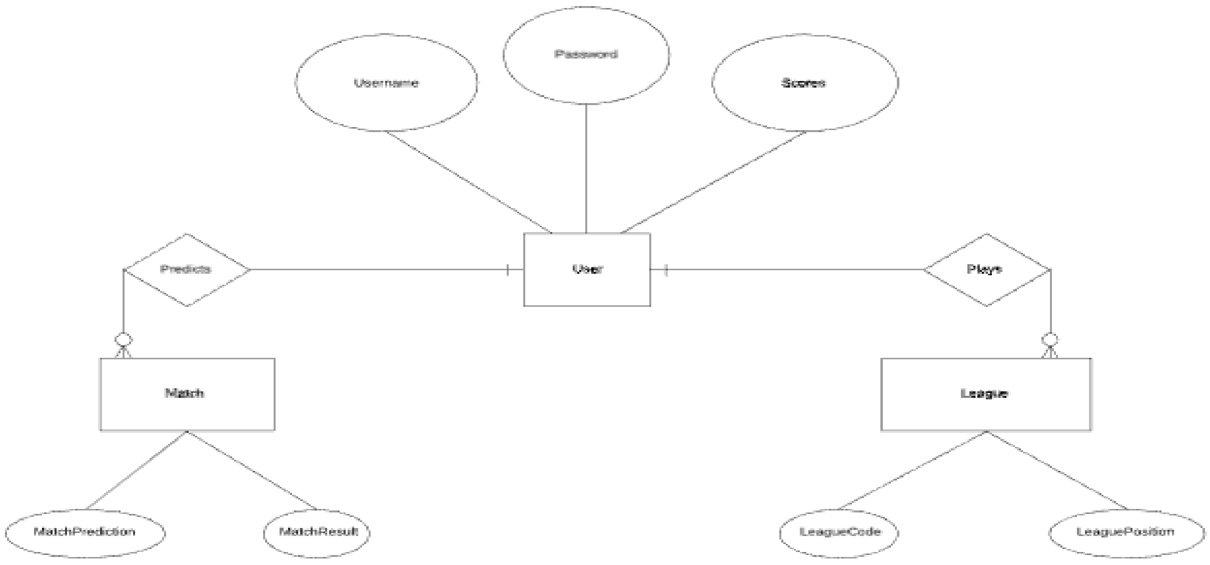


Figure 18: Second iteration Entity Relationship Diagram

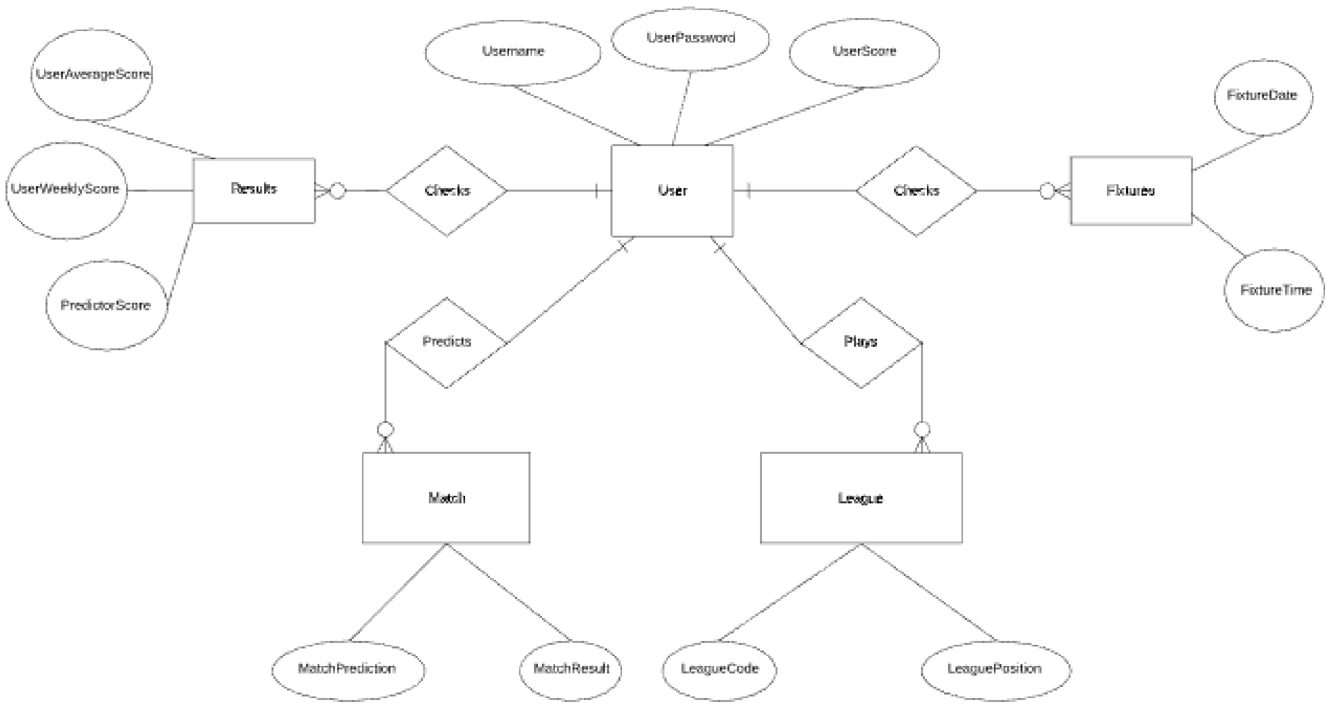


Figure 19: Final iteration Entity Relationship Diagram

3.7. Conclusions

The design of the system was the main focus in this chapter. The models and methodology used for development was discussed as well as a summary of the system architecture. The details around the three tiers including the user experience throughout the application was touched on also. All three layers respective functionalities were laid out, including the storage of data that is used. Each individual page that can be visited was depicted through a rough prototypal version of the final project. The test plan in place to examine the projects efficiency was also discussed as feature by feature is completed.

4. Prototype Development

4.1. Introduction

The development chapter follows on from the design chapter in outlining how the design is implemented into the system. The development process is examined including decisions made to date, as well as issues faced throughout. The methodologies spoken about previously are put into practice here as the finished article takes place. All code and files used for the development can be found in the GitHub repository at https://github.com/jdarcy98/FYP_Premier_Predictions

4.2. Experiment Development

The primary target was to secure an API or data source of some sort. Given the Premier League and Opta's lack of response after a while, it became clear that an unofficial but accurate API would have to be used. Previous similar projects had made use of many different APIs, but all appeared to have a monetary subscription attached to them. Of course it's not ideal to spend money trying to complete a project like this but the reasons behind it are understandable.

A trial plan was procured from SportMonks that would have, upon expiration, cost a significant amount per month to use. The idea was to try and scrape as much data as possible for the back-end database. This database, as mentioned previously, contains data that doesn't change. Using the documentation available with the API, the 2-week trial was utilised to understand the API and JSON requests in general. The data was found to be vast. The possibility of using the API on a paid basis remained a possibility given that it would have been needed for just a few months. This was not in the initial plan but may have become an essential spend for the sake of the project.

Unfortunately, issues arose trying to sort the JSON text scraped from the API into a concise CSV format, and with time starting to be in short supply, a previously established back up plan had to be initiated. Through the Football Data website, there are a large amount of datasets containing results for any given year, not just for the Premier League, but across the major leagues in Europe, as well as their second tiers. This was ideal, and while they contained less detailed data than that from Sportmonks, they had enough to be manipulated into functions to create more complex features.

4.3. Front-End

A basic login to save user data was constructed. This, along with the procurement of data, is the foundation of the project and it was vital to develop them at an early stage. The user simply enters a username and password which is then confirmed to be correct and it's all stored in the back-end database. This was then taken to the next level with the introduction of Django.

While previous experience using this framework may have been lacking at first, using Python and developing web applications was not. That experience proved invaluable as it became easier to use Django because it ties the two together. Forming these ties was the initial challenge, however. As previously mentioned, user integration is the forefront of the project and was chosen to be implemented first, mainly because there is a different and much more secure way of account creation and logging in through Django.

Form validation can be used with one simple line of code. When creating a form, all that needs to be created is an if statement for when the data that is entered is being passed through the POST request. Using the form name and "is_valid()" afterwards within the statement asks Django to check all necessary aspects of the text and that each space is entered correctly. Also, there is a specified email field that can be used to ensure genuine email addresses are entered. These two simple tools are a very efficient way of dealing with possible security issues as an attacker could easily use this as a window to bring down the website through SQL injection, for example. That would disrupt the database where all user data is kept and is meant to be kept private as well. While this isn't a site that contains highly personal information, it's still an absolutely necessary measure to take to make sure everything from top to bottom works as smoothly as possible.

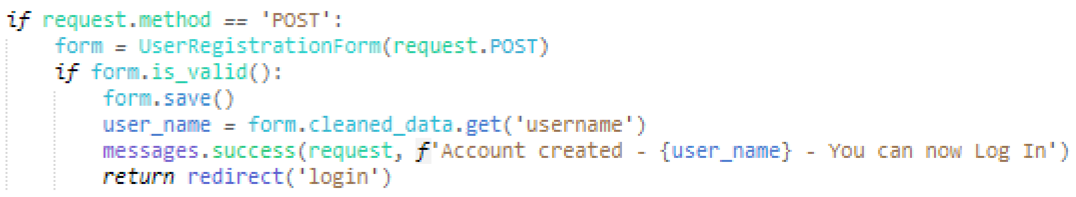
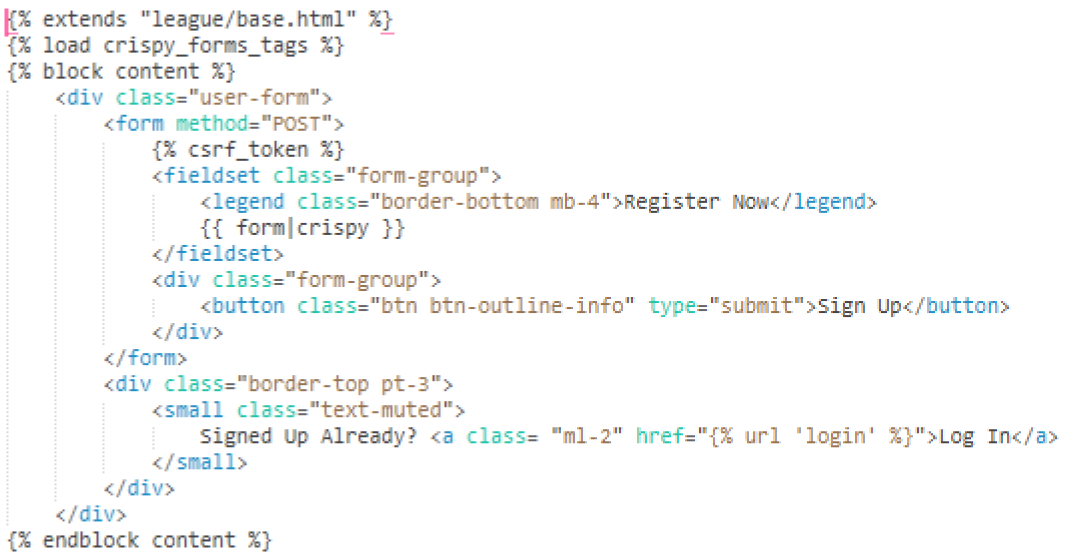

```
if request.method == 'POST':
    form = UserRegistrationForm(request.POST)
    if form.is_valid():
        form.save()
        user_name = form.cleaned_data.get('username')
        messages.success(request, f'Account created - {user_name} - You can now Log In')
        return redirect('login')
```

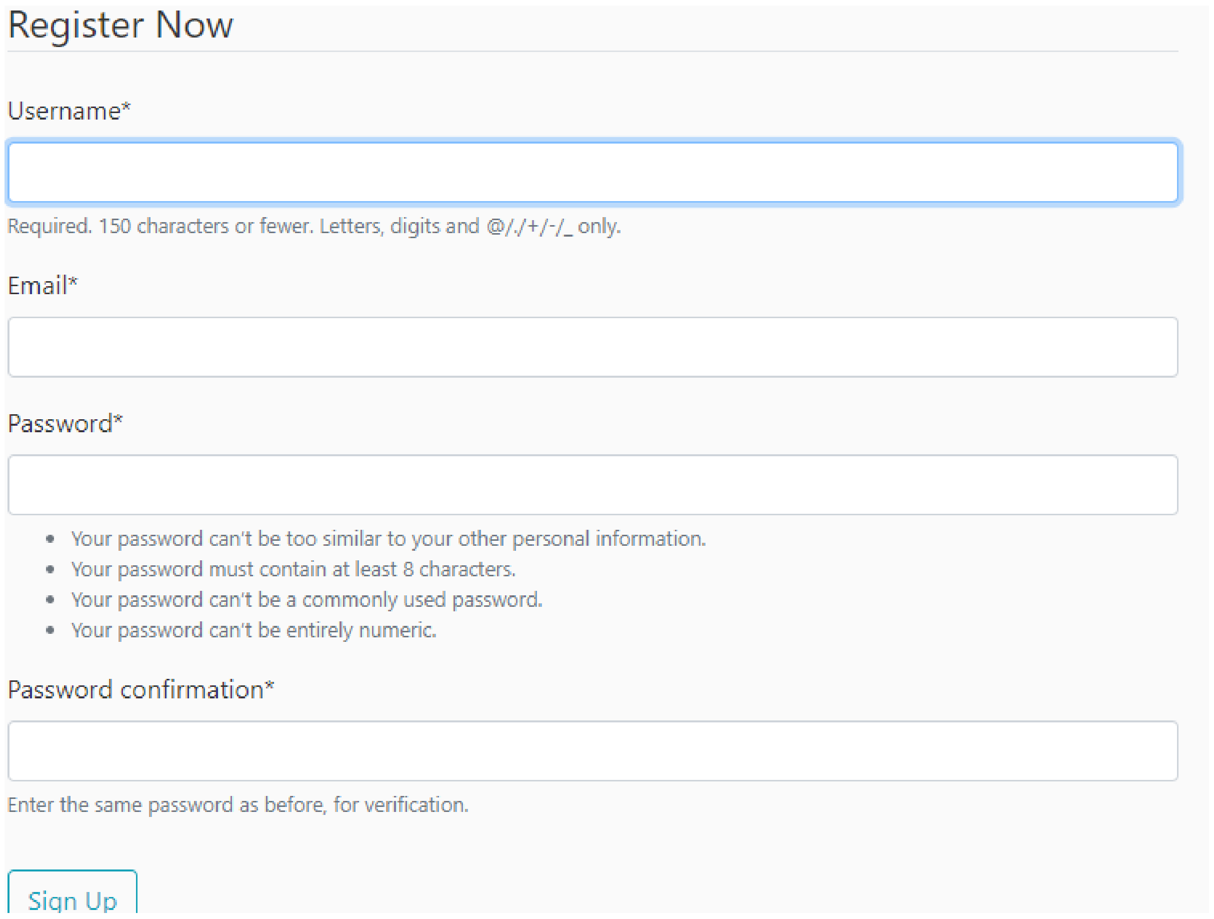
Figure 20: Snippet of the user register form

The Crispy Forms library was then used to raise the standard of design when filling out any forms across the site. Using the tags available within the relevant html file makes this happen. Django also comes with its own versions of registration forms and update forms, so integrating these into the front-end was reasonably simple. Bootstrap was also used to make the design match that of modern websites. The style is also compiled through a singular style sheet, used through the base page which is passed as a template to all pages on the site.



```
{% extends "league/base.html" %}
{% load crispy_forms_tags %}
{% block content %}
    <div class="user-form">
        <form method="POST">
            {% csrf_token %}
            <fieldset class="form-group">
                <legend class="border-bottom mb-4">Register Now</legend>
                {{ form|crispy }}
            </fieldset>
            <div class="form-group">
                <button class="btn btn-outline-info" type="submit">Sign Up</button>
            </div>
        </form>
        <div class="border-top pt-3">
            <small class="text-muted">
                Signed Up Already? <a class= "ml-2" href="{% url 'login' %}">Log In</a>
            </small>
        </div>
    </div>
{% endblock content %}
```

Figure 21: An example of where the Crispy Forms library is used



Register Now

Username*

Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.

Email*

Password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

Password confirmation*

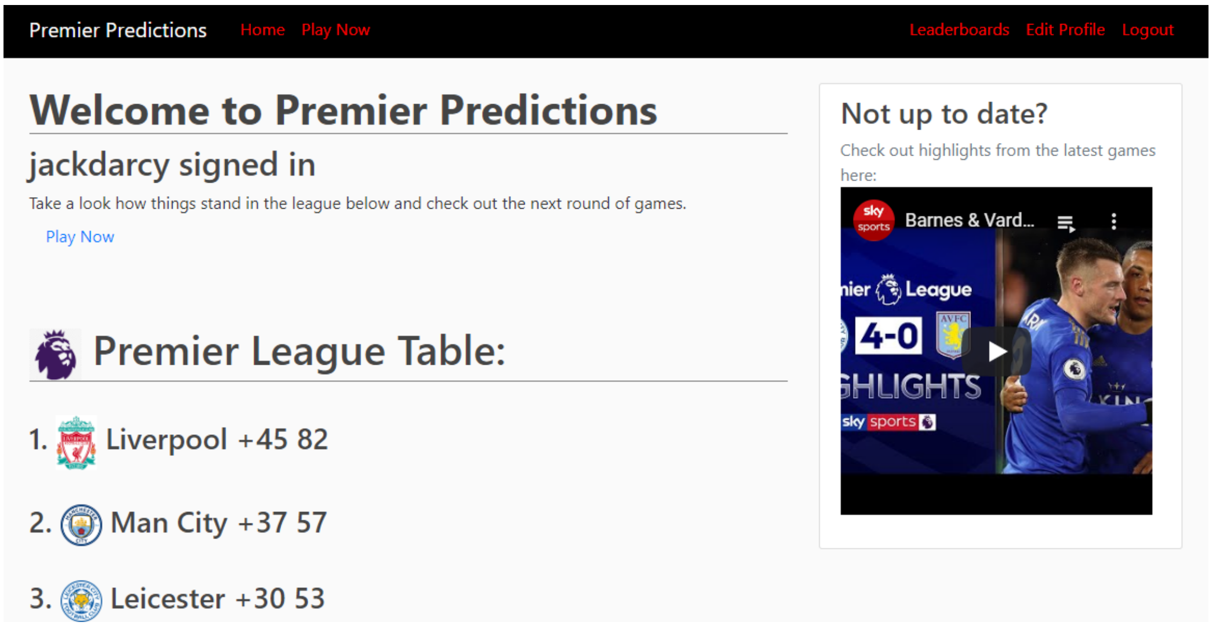
Enter the same password as before, for verification.

Figure 22: How the register form appears on the application

The next logical step would be to form the home page. Keeping to the template set out in the prototype was important, but not crucial. When developing details such as the base HTML file to be used on all pages, a more modern look was needed. This was created with the user in mind, to make things as clear as possible for them to make their experience on the site simpler. Replacing a large logo with numerous tabs lined up beside it, a standard navigation bar with the title of the website and headings leading to the home page and fixtures page on the left side, and all account-oriented pages on the right side. These pages would include the login page, register page and edit profile page and leaderboard page once the user has logged in. The fixtures page also changes to a predictions page, which still lists the fixtures for that week but includes fields for the users input, as well as an opportunity to check what the predictor is guessing for those games at a cost of points.

The home page keeps mostly the same throughout, with the user being led back to that screen after logging in. The one minor difference is there is now an option to go straight to the predictions page. The current league table is displayed on the home page, with its details being printed from the back-end database that is changed manually as the table itself changes. This is included to give users who may not have much background knowledge a general idea of how some of the matches may go. For example, if Liverpool, the leaders of the 2019/20 season were playing Norwich, the team rooted to the bottom of the league, it would be a safe assumption to say Liverpool would be comfortably victorious.

To the right-hand side, as included in the prototype, is a field containing an embedded Premier League highlights playlist from YouTube, which acts as another method for a user to brush up on their knowledge before making any predictions. These aspects are important as it then makes the leaderboard containing all users more competitive, not to mention the fact that their gut feeling that gets passed as a parameter to the predictor is a well-educated guess. It would be simple for users to simply log into the site and throw in any score because they don’t know much about the sport, so including these features are to help the user play properly.


Premier Predictions Home Play Now Leaderboards Edit Profile Logout

Welcome to Premier Predictions

jackdarcy signed in

Take a look how things stand in the league below and check out the next round of games.

Play Now

Premier League Table:

1. Liverpool +45 82
2. Man City +37 57
3. Leicester +30 53

Not up to date?

Check out highlights from the latest games here:

Figure 23: Home page as seen when a user has just logged in.

If a user isn't logged in and they try to access some pages, such as the predictions page or the predictor page, they will be redirected to the login page. On this page, there is also an option to create a new account. Again, this comes from one simple line of code. Putting '@login_required' before the view in question will ensure this to happen.

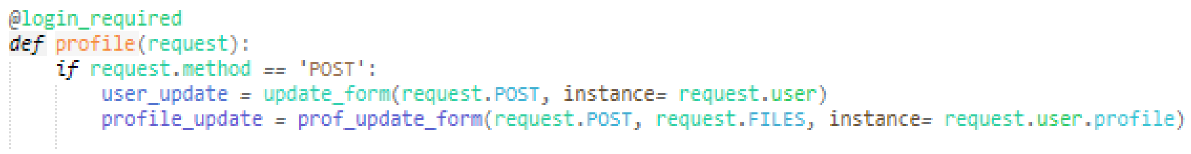
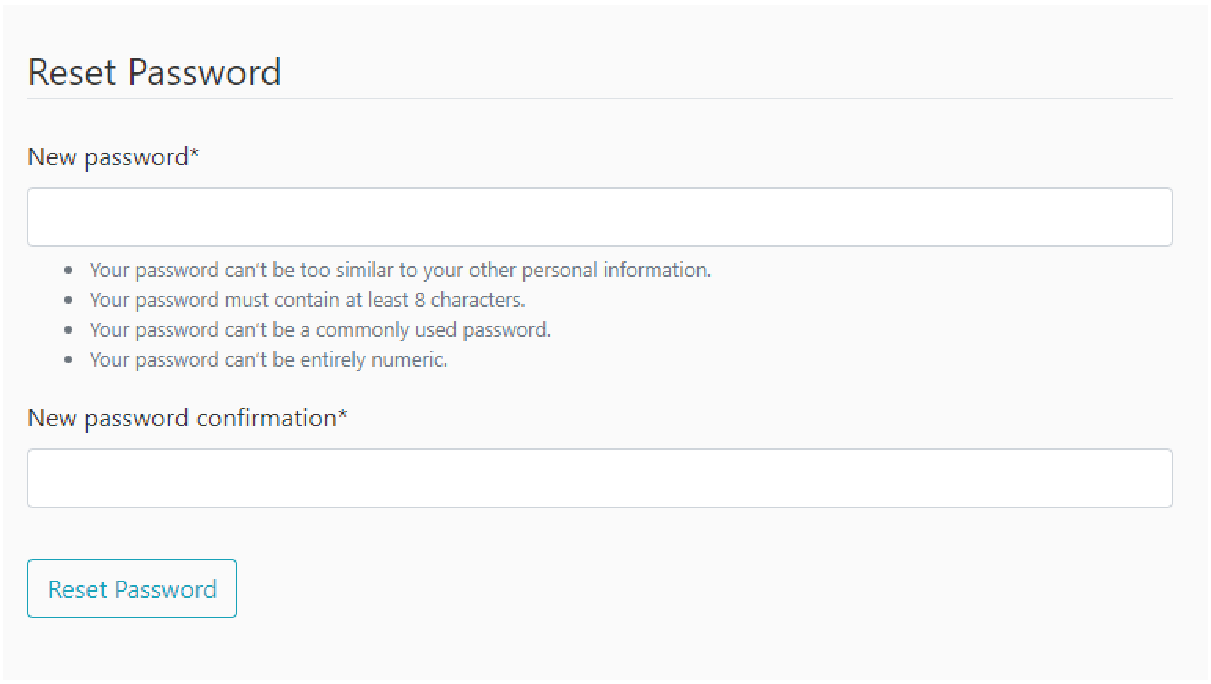

```
@login_required
def profile(request):
    if request.method == 'POST':
        user_update = update_form(request.POST, instance= request.user)
        profile_update = prof_update_form(request.POST, request.FILES, instance= request.user.profile)
```

Figure 24: A sample of where @login_required is used, on the profile update functionality.

A users profile consists of a user name, password, profile picture (for added functionality) and email address. The email address is included for the purpose of updating the password. This tightens the security functionality of the application once again. It's much safer to update a password via email authentication instead of simply entering in a new password alongside the current one. It's also very helpful in case the user has completely forgotten their password and cannot access their account. The user is taken into consideration at every point of the front-end and this password update functionality is no exception. Implementing it had a few levels of obscurity, however. It takes about 4 different HTML pages to fully complete this task. Firstly the user's email must be entered, which is one page. Django again has an automated email template to make things easier. Once the email is sent and the link included is followed, the user is then asked to enter and re-enter their new password in a new window. It must be noted that Django also includes its own password checker to meet the required standards, ensuring that user passwords can't be easily guessed.


Reset Password

New password*

- Your password can't be too similar to your other personal information.
- Your password must contain at least 8 characters.
- Your password can't be a commonly used password.
- Your password can't be entirely numeric.

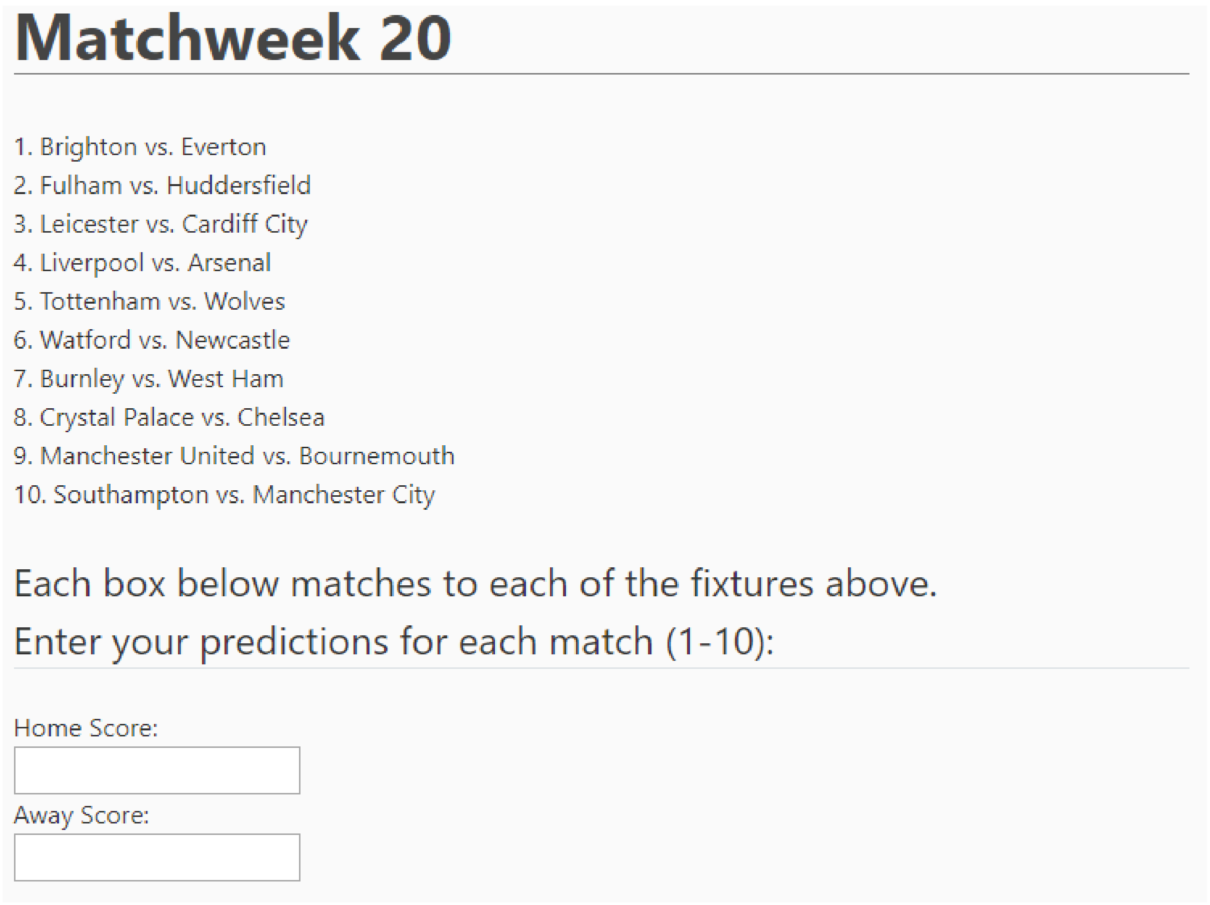
New password confirmation*

Figure 25: Reset password page accesses through email

When predicting the upcoming games, users must enter an amount of goals that they reckon the home team and away team will score within fields that only accept numbers. Of course, there is the option through Django's Decimal Field to go all the way into the millions with goals, but to curb this the maximum number of digits was capped at 2, meaning 99 is the highest value that can be added. The lowest score that can be entered is 0 as it's impossible to enter negative goals for a match. This is all implemented into a similar template to the fixtures page, with the main difference being the two fields to enter the predicted score. With there being 10 matches on an average matchweek, this predictions form has to be looped 10 times, taking in 10 instances for one submission that correlates to each match ID as specified in the fixtures model. It is possible for a user to leave a match completely blank, and in that case the prediction is taken as a 0-0 draw as nothing was entered. Getting the functionality working here was probably the hardest challenge in the front-end.

Django's character fields are seen as the most basic input for a form on an application, and to get this working more familiarity was required with other fields, given that the input in this case is purely numeric. It appeared to be a safe assumption to execute this through an integer field. However, when the fields didn't appear on the web page itself, as well as the fact that no arguments were accepted for the field, it became clear that a decimal field would be the optimal way forward. These fields appeared instantly and the process became much smoother from there onwards, from the application layer side. It also turned out that Django doesn't work well with forms being repeated over and over on the same page. The form was therefore changed to a model form set, but this brought problems of its own. To make things clearer for the user, a for loop containing each form in the form set was to be included alongside the decimal fields, so it was clear what the user was doing. This for loop actually did the same thing as the initial form that was in place. It would only take one input.

Unfortunately, for the sake of maintaining the functionality, the matches had to be listed at the top half of the play now page with the fields to enter predictions located further down. As much as this is an inconvenience for the user, instructions are listed at the top of the page laying out what to do. The user is the main focus for the front-end application, and this obstacle was not going to drastically change that. Each score that is predicted is added as an instance to a back-end model that is exclusive to the logged-in user. It became extremely difficult to match each fixture to each prediction, so this field would have to be added manually. Each prediction instance is identified by number to try simplify the process somewhat for the user.


Matchweek 20

1. Brighton vs. Everton
2. Fulham vs. Huddersfield
3. Leicester vs. Cardiff City
4. Liverpool vs. Arsenal
5. Tottenham vs. Wolves
6. Watford vs. Newcastle
7. Burnley vs. West Ham
8. Crystal Palace vs. Chelsea
9. Manchester United vs. Bournemouth
10. Southampton vs. Manchester City

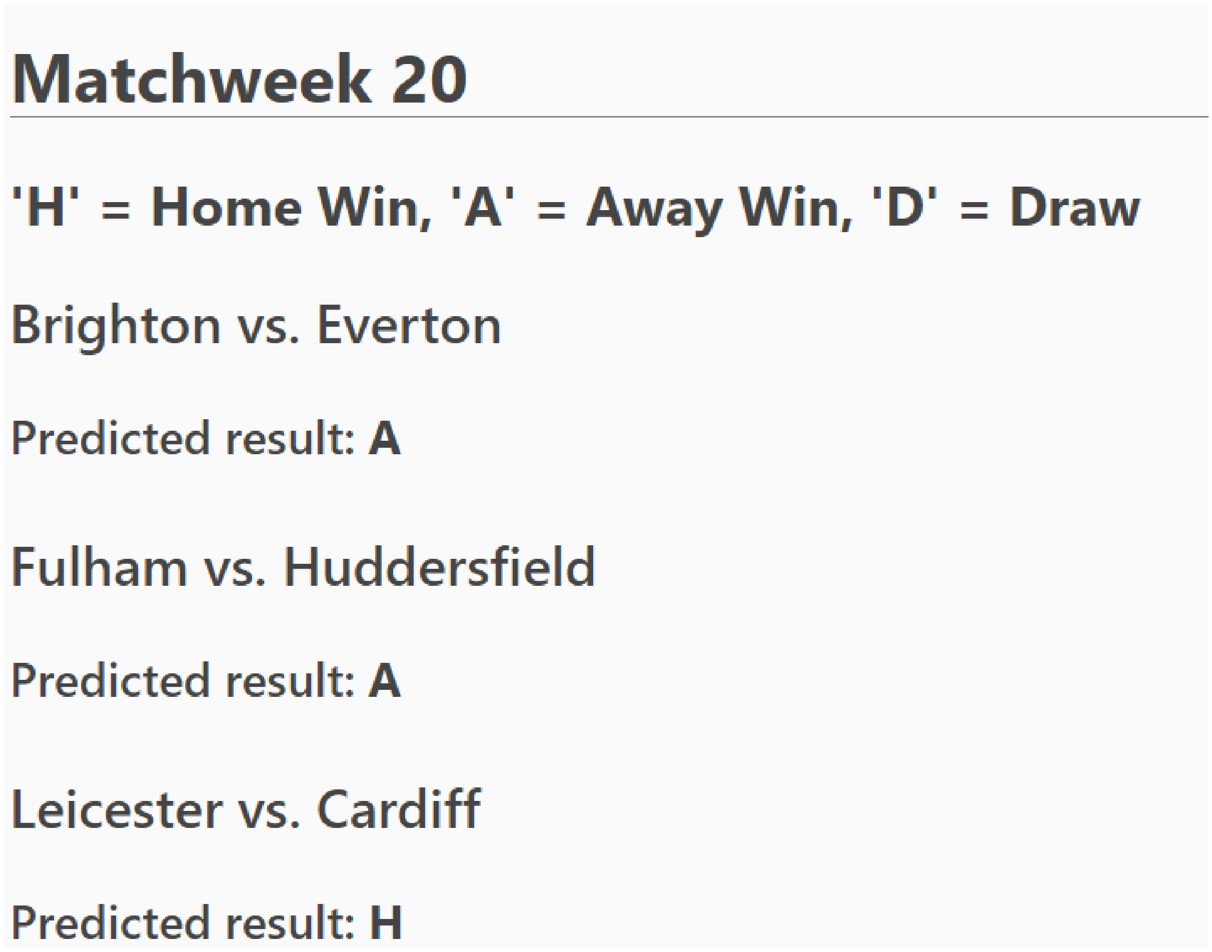
Each box below matches to each of the fixtures above.
Enter your predictions for each match (1-10):

Home Score:

Away Score:

Figure 26: The start of the Play Now page, the two fields at the bottom are repeated for each match

The predictors guesses can then be accessed right at the bottom of the page. It is located here to try and keep it away from the user’s view as looking at the predictor’s guesses is not condoned in any way. In fact, on the button that leads to the page, it warns that 20 points will be lost and poses the question of whether it's worth it for the user to look at. The page itself displays the list of fixtures again, except this time it's taken from the CSV file exported from the predictor script. Beneath each fixture is the predicted result, with each possible outcome defined at the top of the page, as each prediction is simply given as 'H', 'A' or 'D'.



Matchweek 20

'H' = Home Win, 'A' = Away Win, 'D' = Draw

Brighton vs. Everton

Predicted result: **A**

Fulham vs. Huddersfield

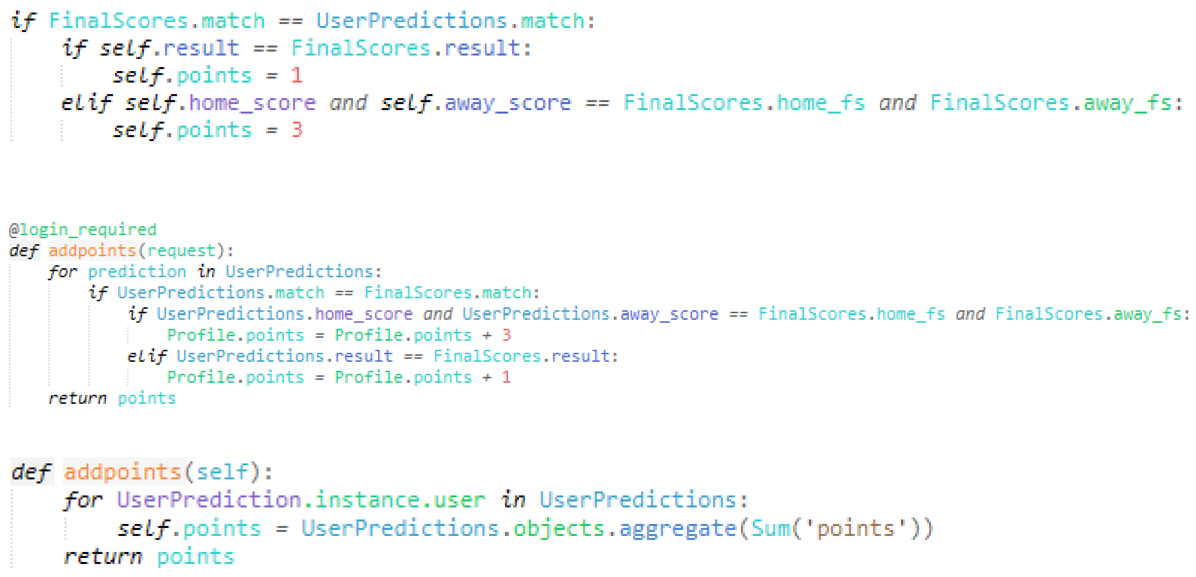
Predicted result: **A**

Leicester vs. Cardiff

Predicted result: **H**

Figure 27: The predictor page

The final aspect of development on the front-end was adding scores for users and including each one in a leaderboard. These aspects were logically the last thing to include given that it relies on everything else working to full capacity. The scores had to be linked to correct and incorrect predictions, which meant the predictions had to be linked to the final scores in each match. This would apply to each profile that made predictions on the most recent matches, with all of them listed in a leaderboard that is given its own page. However, the efforts to try and link the 3 models together to achieve this was ultimately in vain. While trying to link the predicted scores to the final scores to generate the required points, the functionality didn't work. Instead, this field would have to be added up manually by comparing the two models.



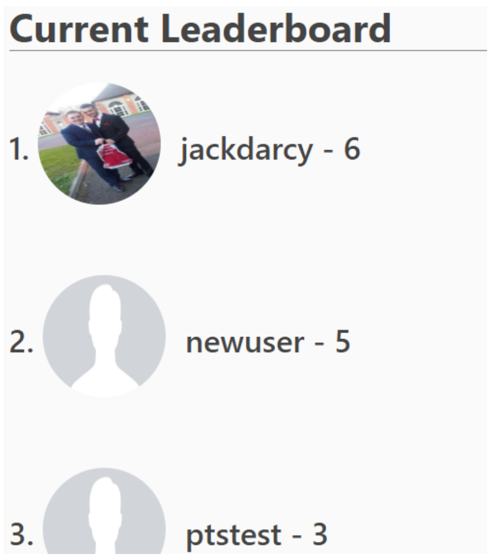
```
if FinalScores.match == UserPredictions.match:
    if self.result == FinalScores.result:
        self.points = 1
    elif self.home_score and self.away_score == FinalScores.home_fs and FinalScores.away_fs:
        self.points = 3
```

```
@login_required
def addpoints(request):
    for prediction in UserPredictions:
        if UserPredictions.match == FinalScores.match:
            if UserPredictions.home_score and UserPredictions.away_score == FinalScores.home_fs and FinalScores.away_fs:
                Profile.points = Profile.points + 3
            elif UserPredictions.result == FinalScores.result:
                Profile.points = Profile.points + 1
    return points
```

```
def addpoints(self):
    for UserPrediction.instance.user in UserPredictions:
        self.points = UserPredictions.objects.aggregate(Sum('points'))
    return points
```

Figure 28: Different attempts at getting the points system to work

The leaderboard page solely consists of a list of users and their total score to date. The initial rubric of 1 point for a correct result and 3 points for a correct score is maintained, therefore the maximum score for a given week is 30. Using Sky Sports' Super 6 system as an example again, guessing their six results fully correctly is deemed nearly impossible, and Premier Predictions is another level above that, with 10 correct results needed for a clean sweep. The predictor cannot really be included in the leaderboard as first projected because it predicts a result, not a score. In this respect, the predictor may not be seen as a useful tool, but the testing phase of the model itself plays down this claim.


Current Leaderboard

1. jackdarcy - 6

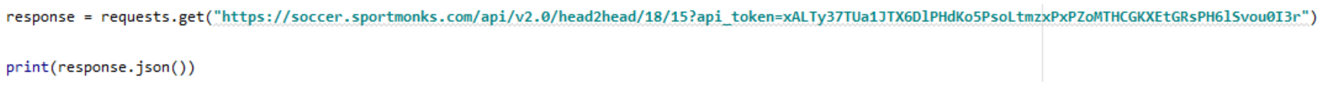
2. newuser - 5

3. ptstest - 3

Figure 29: The leaderboard page

4.4. Middle-Tier

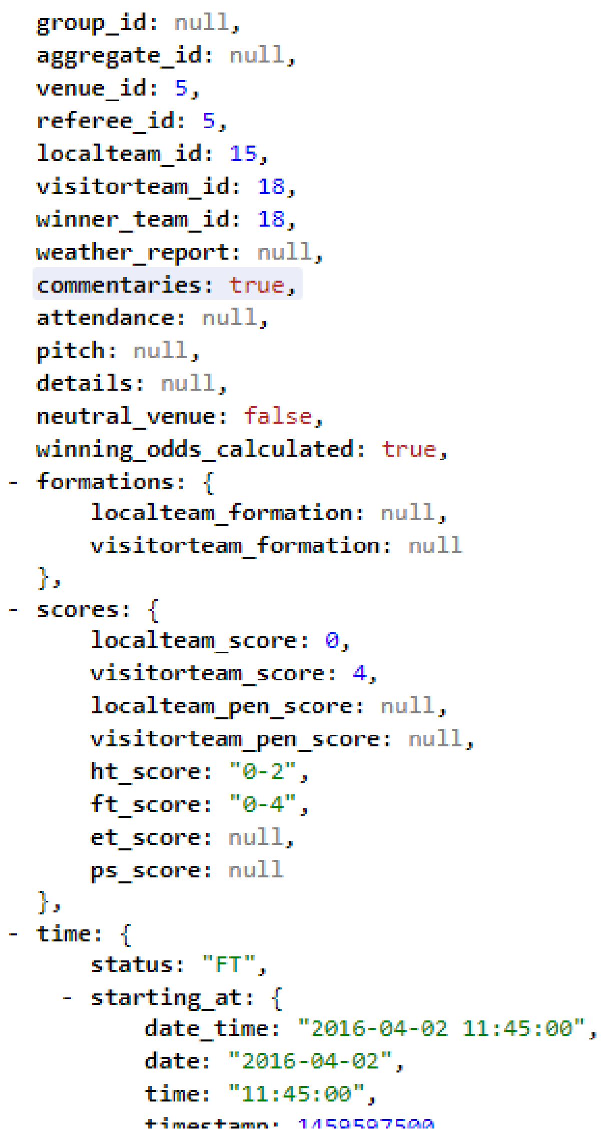
When an API was to be used as the main data source, a few JSON requests to get data from the API were attempted through Python. These worked well but after further research some of this data wasn’t necessary.



```
response = requests.get("https://soccer.sportmonks.com/api/v2.0/head2head/18/15?api_token=xALTy37TUa1JTX6DlPHdKo5PsoLtmzxPxPZoMTHCGKXEtGRsPH6lSvou0I3r")

print(response.json())
```

Figure 30: Sample JSON request in Python

This JSON request example was used to get head-to-head data for matches between Chelsea and Aston Villa. Note the token that has to be included at the end in order for the request to work. This was a token that is generated only by having a SportMonks account and would return a 404 error without the required subscription. This response was tested within Python to see how the correlation works, and the output was the exact same as within a browser.



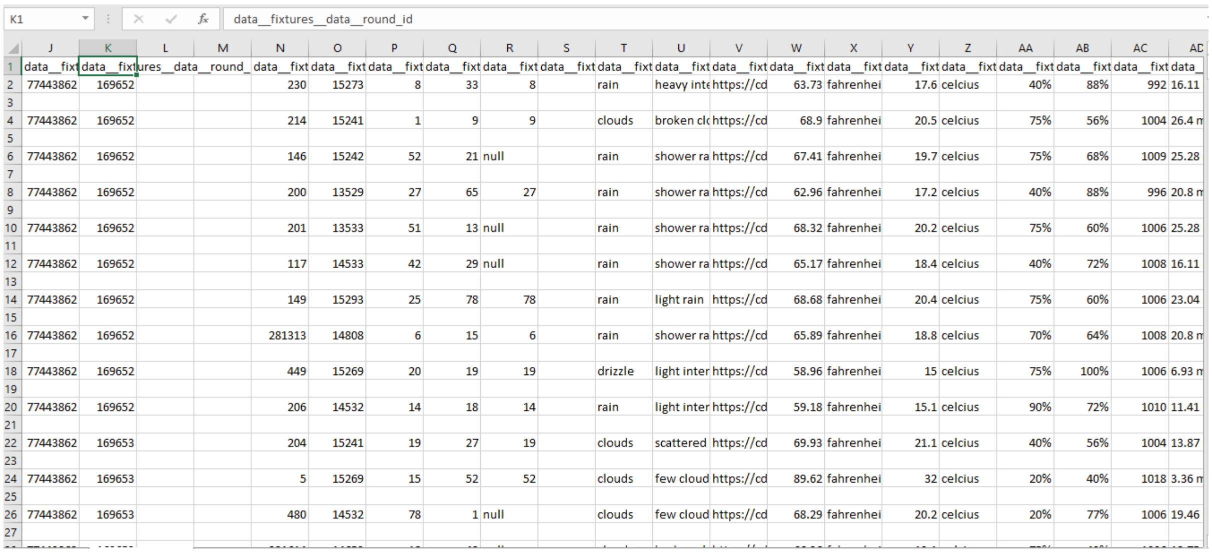
```
    group_id: null,
    aggregate_id: null,
    venue_id: 5,
    referee_id: 5,
    localteam_id: 15,
    visitorteam_id: 18,
    winner_team_id: 18,
    weather_report: null,
    commentaries: true,
    attendance: null,
    pitch: null,
    details: null,
    neutral_venue: false,
    winning_odds_calculated: true,
  - formations: {
        localteam_formation: null,
        visitorteam_formation: null
    },
  - scores: {
        localteam_score: 0,
        visitorteam_score: 4,
        localteam_pen_score: null,
        visitorteam_pen_score: null,
        ht_score: "0-2",
        ft_score: "0-4",
        et_score: null,
        ps_score: null
    },
  - time: {
        status: "FT",
      - starting_at: {
            date_time: "2016-04-02 11:45:00",
            date: "2016-04-02",
            time: "11:45:00",
            timestamp: 1459597500
```

Figure 31: Sample JSON response

The above is a snippet of the response from the API for that same request. Here, the last time Chelsea and Aston Villa played in the Premier League can be seen, including the half time score, full time score, weather, appointed referee and time of day.

Of course the use of an API was abandoned, with less detailed data being available to pass into a predictor. It was also deemed that this head-to-head data can be inaccurate. Using the example of Chelsea and Aston Villa's head-to-head record again, the last time they played each other in the Premier League was in 2016, a year in which Aston Villa got relegated and Chelsea as defending champions had an extremely poor season by their usual standards and finished 10th. In essence, teams can go through rough patches that can last over a few years, or or have great seasons. Using Manchester United as another example, their head-to-head record against the majority of teams in the league up to 2013 was unparalleled. However, their form has dipped significantly over the past 7 years, reducing the credibility of that record almost completely, so this feature had to be left out.

Using this season's fixtures as an example again, Postman returned a response in pretty print as seen above. This made it easier to read the data, which was already vast, and the documentation alongside the API helped to understand what it all means. The standout offering from Postman however, had to be the fact that the response can be downloaded and saved into a JSON file. Microsoft Excel has a feature whereby data within a JSON file can be imported into a comma-separated values file. This was to be integrated into the Python program as using the Pandas library, this data could be manipulated from Excel spreadsheets into dataframes.

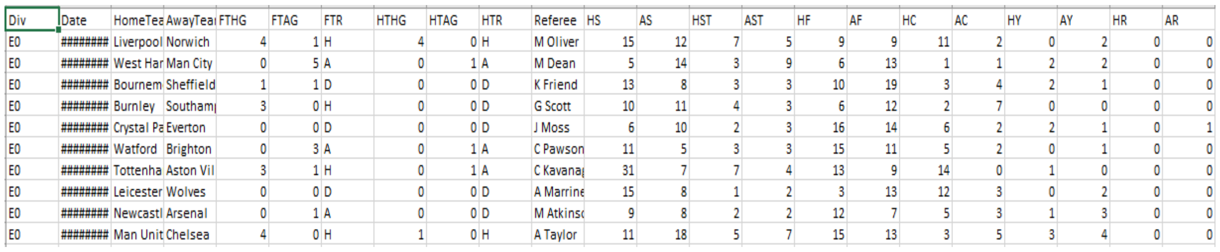

K1 | data__fixtures__data__round_id

	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD
1	data__fixt	data__fixtures__data__round_			data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data__fixt	data_
2	77443862	169652			230	15273	8	33	8		rain	heavy inte	https://cd	63.73	fahrenhei	17.6	celcius	40%	88%	992	16.11
3																					
4	77443862	169652			214	15241	1	9	9		clouds	broken clc	https://cd	68.9	fahrenhei	20.5	celcius	75%	56%	1004	26.4 m
5																					
6	77443862	169652			146	15242	52	21	null		rain	shower ra	https://cd	67.41	fahrenhei	19.7	celcius	75%	68%	1009	25.28
7																					
8	77443862	169652			200	13529	27	65	27		rain	shower ra	https://cd	62.96	fahrenhei	17.2	celcius	40%	88%	996	20.8 m
9																					
10	77443862	169652			201	13533	51	13	null		rain	shower ra	https://cd	68.32	fahrenhei	20.2	celcius	75%	60%	1006	25.28
11																					
12	77443862	169652			117	14533	42	29	null		rain	shower ra	https://cd	65.17	fahrenhei	18.4	celcius	40%	72%	1008	16.11
13																					
14	77443862	169652			149	15293	25	78	78		rain	light rain	https://cd	68.68	fahrenhei	20.4	celcius	75%	60%	1006	23.04
15																					
16	77443862	169652			281313	14808	6	15	6		rain	shower ra	https://cd	65.89	fahrenhei	18.8	celcius	70%	64%	1008	20.8 m
17																					
18	77443862	169652			449	15269	20	19	19		drizzle	light inter	https://cd	58.96	fahrenhei	15	celcius	75%	100%	1006	6.93 m
19																					
20	77443862	169652			206	14532	14	18	14		rain	light inter	https://cd	59.18	fahrenhei	15.1	celcius	90%	72%	1010	11.41
21																					
22	77443862	169653			204	15241	19	27	19		clouds	scattered	https://cd	69.93	fahrenhei	21.1	celcius	40%	56%	1004	13.87
23																					
24	77443862	169653			5	15269	15	52	52		clouds	few cloud	https://cd	89.62	fahrenhei	32	celcius	20%	40%	1018	3.36 m
25																					
26	77443862	169653			480	14532	78	1	null		clouds	few cloud	https://cd	68.29	fahrenhei	20.2	celcius	20%	77%	1006	19.46
27																					

Figure 32: A snippet of the csv file containing the 2019/20 fixtures

Unfortunately however, this convenient functionality wasn’t retained after the Postman software was updated. A new source was required and one that would be reliable and ideally more cost-efficient than the Sportmonks API. The Football Data site has an ample amount of datasets which had already been secured. An API is of course a much more ideal tool to have, but after considering how minimally the predictor would appear in the final application, something that’s automatically updated wouldn’t really have been required.

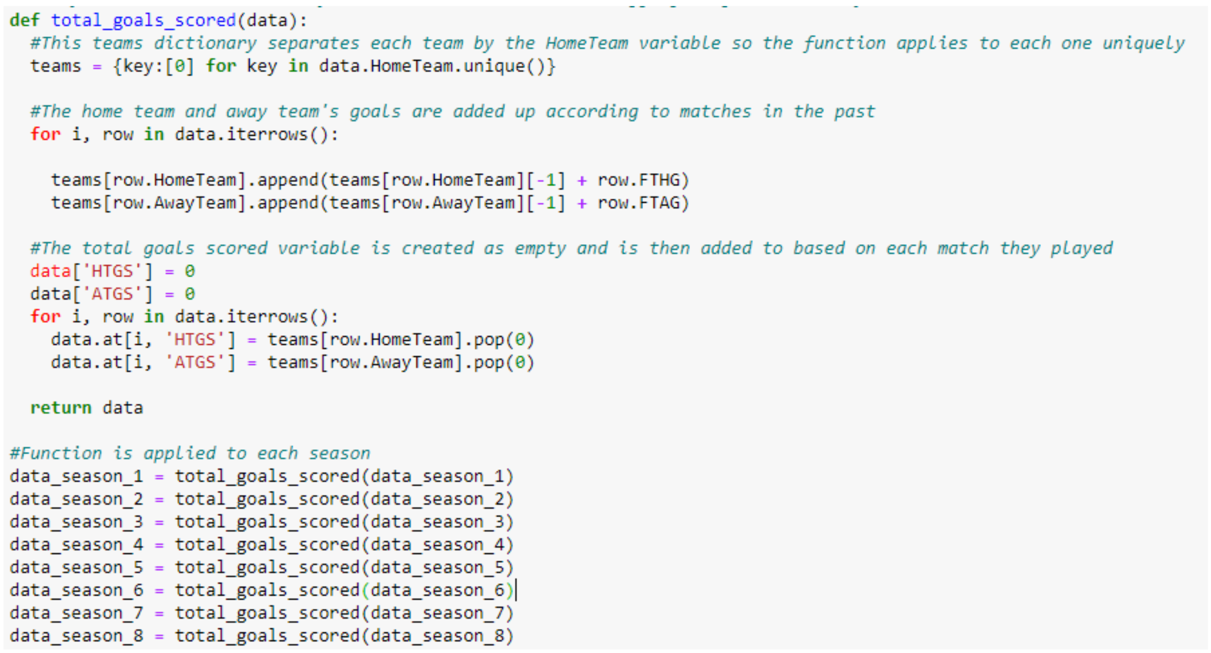
Datasets from the 2019/20 season back to the 2010/11 season were downloaded, giving a total of 10 years worth of data to work with. The features within these datasets, however, were basic to the extent that trends couldn’t be tracked such as a team’s performance over a given amount of time. The tools to calculate features like these however were included, and through Python functions could be created and added to the dataset, making for a much better-trained predictor.



Div	Date	HomeTea	AwayTeai	FTHG	FTAG	FTR	HTHG	HTAG	HTR	Referee	HS	AS	HST	AST	HF	AF	HC	AC	HY	AY	HR	AR
E0	########	Liverpool	Norwich	4	1	H	4	0	H	M Oliver	15	12	7	5	9	9	11	2	0	2	0	0
E0	########	West Har	Man City	0	5	A	0	1	A	M Dean	5	14	3	9	6	13	1	1	2	2	0	0
E0	########	Bournem	Sheffield	1	1	D	0	0	D	K Friend	13	8	3	3	10	19	3	4	2	1	0	0
E0	########	Burnley	Southam	3	0	H	0	0	D	G Scott	10	11	4	3	6	12	2	7	0	0	0	0
E0	########	Crystal Pa	Everton	0	0	D	0	0	D	J Moss	6	10	2	3	16	14	6	2	2	1	0	1
E0	########	Watford	Brighton	0	3	A	0	1	A	C Pawson	11	5	3	3	15	11	5	2	0	1	0	0
E0	########	Tottenha	Aston Vil	3	1	H	0	1	A	C Kavana	31	7	7	4	13	9	14	0	1	0	0	0
E0	########	Leicester	Wolves	0	0	D	0	0	D	A Marrine	15	8	1	2	3	13	12	3	0	2	0	0
E0	########	Newcastl	Arsenal	0	1	A	0	0	D	M Atkinsc	9	8	2	2	12	7	5	3	1	3	0	0
E0	########	Man Unit	Chelsea	4	0	H	1	0	H	A Taylor	11	18	5	7	15	13	3	5	3	4	0	0

Figure 33: A snippet of the Football Data dataset for the 2019/20 season

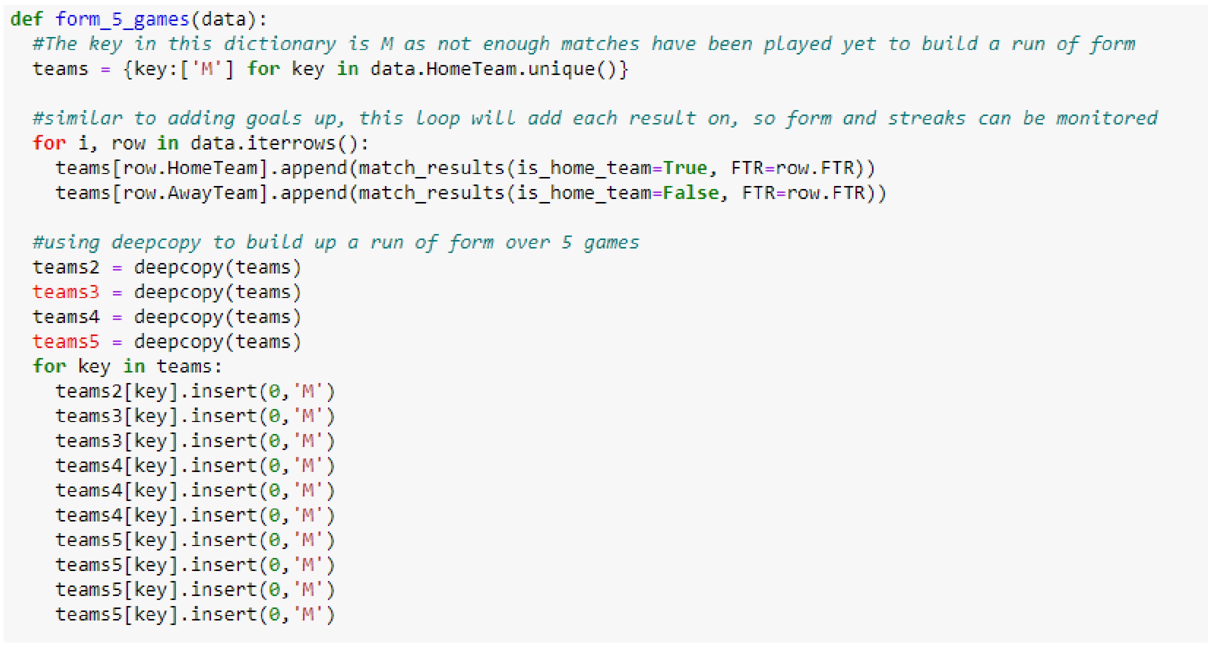
Some of these features weren't necessary to include for prediction either. The referee, in reality, shouldn't alter the performance of a team. Theoretically it could be said that they might, but in this instance a referees bias is not tangible in any way. The main features to take from these datasets to be manipulated would be the goals scored and the full time result. The full time result represents the target feature, but can be used to define the amount of points a team has picked up in a match, and then over 3 or 5 matches to track their form going into a match. Firstly, however, the goals scored can be added up to find the total amount of goals a team has scored and conceded over a season. The Pandas library is crucial to this data cleansing process. Each dataset is taken in as a data frame one by one, with the 2019/20 season being smaller than the others because it hasn't concluded yet. The goal scoring stats are passed to functions that iterates through each row which represents a fixture and adds the goals a team scored to a dictionary. This dictionary is called 'teams', and includes a key for each respective team in the league for that year. Each team is searched for as either the home team or away team and the amount of goals they score in each match is appended on. Once this process is complete it leaves two new features, the home team goals scored and away team goals scored. The function is then applied to each data frame so all seasons have the same features throughout. The same process applies to getting the amount of goals a team has conceded too.



```python
def total_goals_scored(data):
  #This teams dictionary separates each team by the HomeTeam variable so the function applies to each one uniquely
  teams = {key:[0] for key in data.HomeTeam.unique()}

  #The home team and away team's goals are added up according to matches in the past
  for i, row in data.iterrows():

    teams[row.HomeTeam].append(teams[row.HomeTeam][-1] + row.FTHG)
    teams[row.AwayTeam].append(teams[row.AwayTeam][-1] + row.FTAG)

  #The total goals scored variable is created as empty and is then added to based on each match they played
  data['HTGS'] = 0
  data['ATGS'] = 0
  for i, row in data.iterrows():
    data.at[i, 'HTGS'] = teams[row.HomeTeam].pop(0)
    data.at[i, 'ATGS'] = teams[row.AwayTeam].pop(0)

  return data

#Function is applied to each season
data_season_1 = total_goals_scored(data_season_1)
data_season_2 = total_goals_scored(data_season_2)
data_season_3 = total_goals_scored(data_season_3)
data_season_4 = total_goals_scored(data_season_4)
data_season_5 = total_goals_scored(data_season_5)
data_season_6 = total_goals_scored(data_season_6)
data_season_7 = total_goals_scored(data_season_7)
data_season_8 = total_goals_scored(data_season_8)
```

Figure 34: The function for calculating the total goals scored for a team over a season

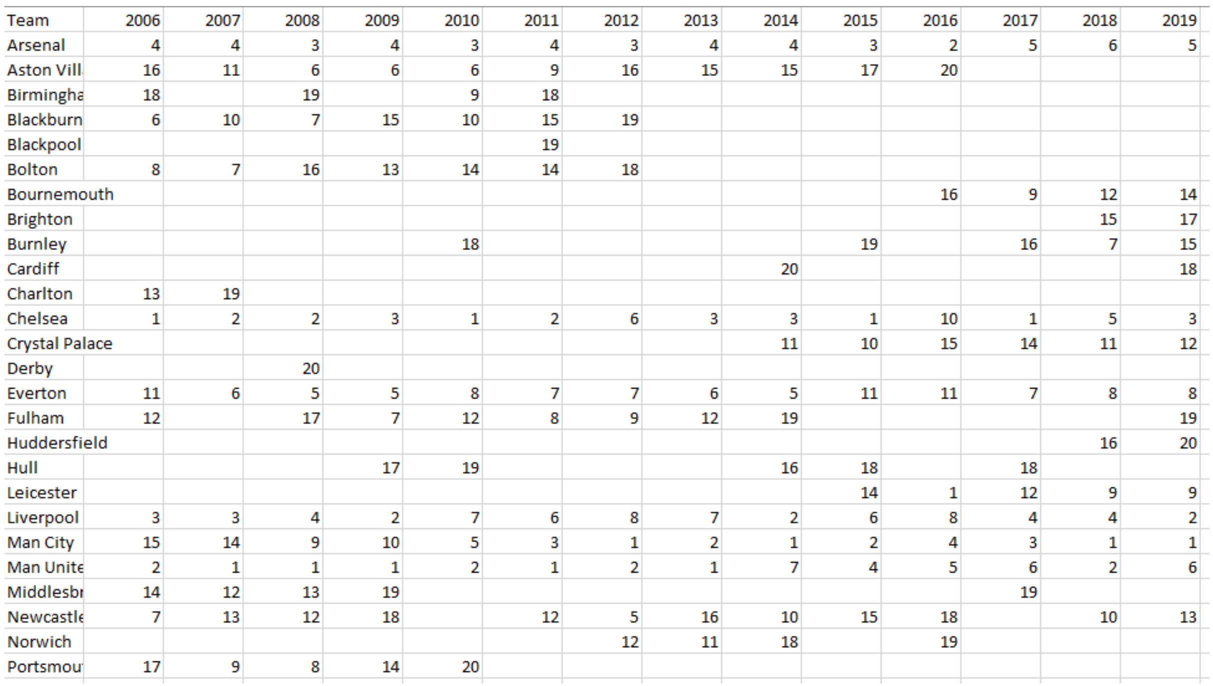
Defining what constitutes 3 points, 1 point or no points for the home team was next. Using the full time result feature, together with a simple if else statement, can define a home win as 3 points, an away win as 0 points and a draw as 1 point. This can then be used in another function similar to the goal scoring statistics to get the total points for a team over a season. This is a very important feature to have, as it directly maps to the kind of season a team is having. From this feature the difference in points between two teams can be calculated by simply subtracting the away teams points from the home teams points. A similar principle is used to put together a run of form for a team over 5 games. By checking if the team is at home and if they win, a 'W' is returned, a loss returns an 'L' and a draw returns a 'D'. If the team hasn't played the necessary 5 games to build up a run of form for the better or worse, an 'M' is returned. Each letter is added on to a string totalling 5 letters that is rewritten as each new match is added. The least recent match out of the 5 is popped out with the newest one taking its place. The deepcopy tool is used to replicate the teams dictionary 5 times.



```python
def form_5_games(data):
  #The key in this dictionary is M as not enough matches have been played yet to build a run of form
  teams = {key:['M'] for key in data.HomeTeam.unique()}

  #similar to adding goals up, this loop will add each result on, so form and streaks can be monitored
  for i, row in data.iterrows():
    teams[row.HomeTeam].append(match_results(is_home_team=True, FTR=row.FTR))
    teams[row.AwayTeam].append(match_results(is_home_team=False, FTR=row.FTR))

  #using deepcopy to build up a run of form over 5 games
  teams2 = deepcopy(teams)
  teams3 = deepcopy(teams)
  teams4 = deepcopy(teams)
  teams5 = deepcopy(teams)
  for key in teams:
    teams2[key].insert(0,'M')
    teams3[key].insert(0,'M')
    teams3[key].insert(0,'M')
    teams4[key].insert(0,'M')
    teams4[key].insert(0,'M')
    teams4[key].insert(0,'M')
    teams5[key].insert(0,'M')
    teams5[key].insert(0,'M')
    teams5[key].insert(0,'M')
    teams5[key].insert(0,'M')
```

Figure 35: A snippet of the function that tracks a teams form, each possible result (W, L or D) is already defined

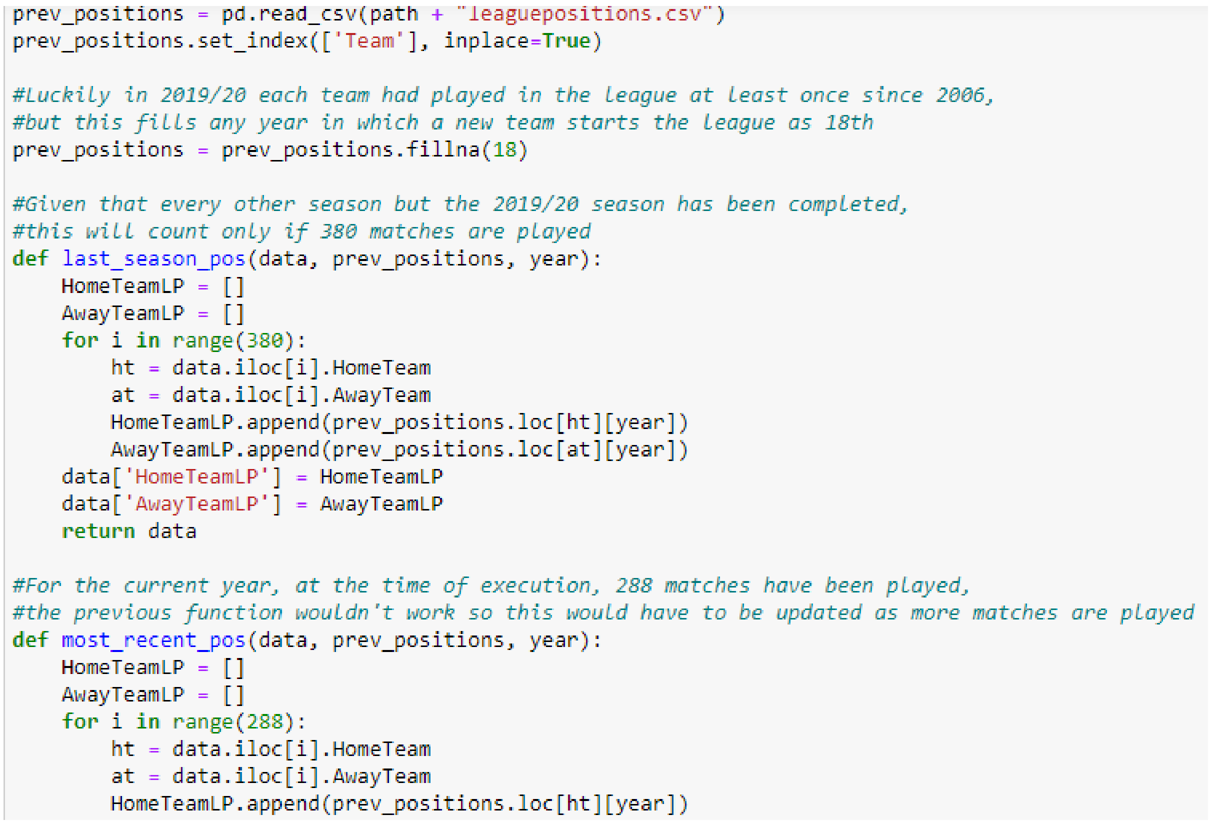
All new features that have been created thus far are added as columns to each data frame. Each team's previous league position in relation to a season is then calculated. A manually-made dataset was used for this. By researching the previous league tables on the Premier League's official website, a CSV file was created for each team who has played in the league since 2006. Their league positions for each year were then entered into columns, with teams that were relegated for a given year being left blank. This is another important feature to include as a teams league position can set their pedigree for the next year. People will remember the position a team finished in the previous season primarily. For example, if a team wins the league, they are known as the outright champions until someone else wins it. A similar logic applies to the rest of the table.



Team	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Arsenal	4	4	3	4	3	4	3	4	4	3	2	5	6	5
Aston Vill	16	11	6	6	6	9	16	15	15	17	20			
Birmingha	18		19		9	18								
Blackburn	6	10	7	15	10	15	19							
Blackpool						19								
Bolton	8	7	16	13	14	14	18							
Bournemouth											16	9	12	14
Brighton													15	17
Burnley					18					19		16	7	15
Cardiff									20					18
Charlton	13	19												
Chelsea	1	2	2	3	1	2	6	3	3	1	10	1	5	3
Crystal Palace									11	10	15	14	11	12
Derby			20											
Everton	11	6	5	5	8	7	7	6	5	11	11	7	8	8
Fulham	12		17	7	12	8	9	12	19					19
Huddersfield													16	20
Hull				17	19				16	18		18		
Leicester										14	1	12	9	9
Liverpool	3	3	4	2	7	6	8	7	2	6	8	4	4	2
Man City	15	14	9	10	5	3	1	2	1	2	4	3	1	1
Man Unite	2	1	1	1	2	1	2	1	7	4	5	6	2	6
Middlesbr	14	12	13	19								19		
Newcastle	7	13	12	18		12	5	16	10	15	18		10	13
Norwich							12	11	18		19			
Portsmou	17	9	8	14	20									

Figure 36: Some of the data used in the leaguepositions.csv dataset

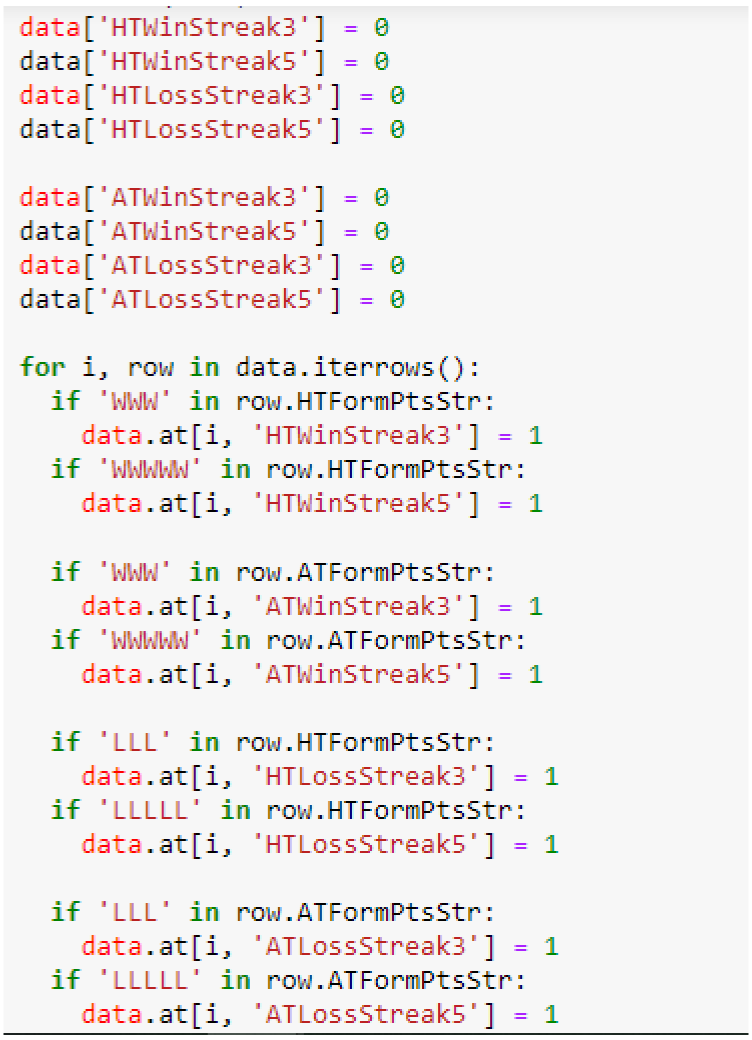
This data is then passed through another function after being taken in as its own data frame. Each blank spot is filled with the position '18', as the Pandas library does not deal well with NaN values at all. Luckily in the 2019/20 season, all teams had played in the league at least once since 2006. With that being said, NaN values were a recurring issue with the dataset for the current season. Given that this dataset was smaller than the rest because it was still ongoing, some functions felt the values needed to match the rest of the data sets were being passed as NaN, namely the function in question. Separate functions had to be used in this case, with one iterating through 380 rows for a completed season, and the other, made especially for the 2019/20 season, iterates through the amount of games that were played altogether at the time. It can be a bit tedious having to go back and update this by hand every time new games are played, but when putting the predictors preparation into perspective, it has to be done to maintain its credibility. The more games that are played also gives a greater chance of improving the accuracy too. Both of these functions take in a data frame that represents that season's data, another data frame that represents each team's previous position, and a year that is entered by column, numbered from 0-12.



```python
prev_positions = pd.read_csv(path + "leaguepositions.csv")
prev_positions.set_index(['Team'], inplace=True)

#Luckily in 2019/20 each team had played in the league at least once since 2006,
#but this fills any year in which a new team starts the league as 18th
prev_positions = prev_positions.fillna(18)

#Given that every other season but the 2019/20 season has been completed,
#this will count only if 380 matches are played
def last_season_pos(data, prev_positions, year):
    HomeTeamLP = []
    AwayTeamLP = []
    for i in range(380):
        ht = data.iloc[i].HomeTeam
        at = data.iloc[i].AwayTeam
        HomeTeamLP.append(prev_positions.loc[ht][year])
        AwayTeamLP.append(prev_positions.loc[at][year])
    data['HomeTeamLP'] = HomeTeamLP
    data['AwayTeamLP'] = AwayTeamLP
    return data

#For the current year, at the time of execution, 288 matches have been played,
#the previous function wouldn't work so this would have to be updated as more matches are played
def most_recent_pos(data, prev_positions, year):
    HomeTeamLP = []
    AwayTeamLP = []
    for i in range(288):
        ht = data.iloc[i].HomeTeam
        at = data.iloc[i].AwayTeam
        HomeTeamLP.append(prev_positions.loc[ht][year])
```

Figure 37: The functions to add each teams previous league position to a particular season

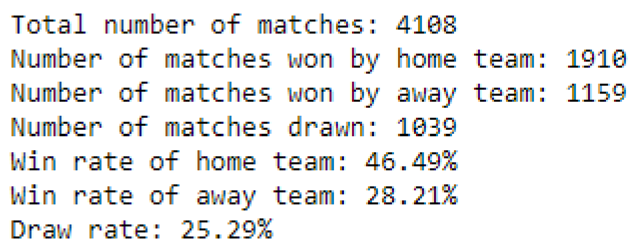
The next feature to add was the matchweek for each fixture. This is important as when processing the data the first 3 match weeks can be disregarded as no trends in form can be monitored. A separate function had to be created in this case as well for the current season. All of the seasons data frames were then added together to form one large data frame containing all 13 seasons. From here the differential features are calculated, such as the goal difference, points difference, and difference in league position between two teams that are about to play each other. Putting all of this data in one set is vital to train the predictor properly. It would be much more difficult for a model to go through 13 separate datasets to train itself, so keeping it as one makes it a smoother process. At this stage winning and losing streaks are calculated. This particular function searches each row for a string of 3 Ws, 5 Ws or 3Ls or 5 Ls and adds to a streak column for either the home team or away team if any are found.



```
data['HTWinStreak3'] = 0
data['HTWinStreak5'] = 0
data['HTLossStreak3'] = 0
data['HTLossStreak5'] = 0

data['ATWinStreak3'] = 0
data['ATWinStreak5'] = 0
data['ATLossStreak3'] = 0
data['ATLossStreak5'] = 0

for i, row in data.iterrows():
  if 'WWW' in row.HTFormPtsStr:
    data.at[i, 'HTWinStreak3'] = 1
  if 'WWWWW' in row.HTFormPtsStr:
    data.at[i, 'HTWinStreak5'] = 1

  if 'WWW' in row.ATFormPtsStr:
    data.at[i, 'ATWinStreak3'] = 1
  if 'WWWWW' in row.ATFormPtsStr:
    data.at[i, 'ATWinStreak5'] = 1

  if 'LLL' in row.HTFormPtsStr:
    data.at[i, 'HTLossStreak3'] = 1
  if 'LLLLL' in row.HTFormPtsStr:
    data.at[i, 'HTLossStreak5'] = 1

  if 'LLL' in row.ATFormPtsStr:
    data.at[i, 'ATLossStreak3'] = 1
  if 'LLLLL' in row.ATFormPtsStr:
    data.at[i, 'ATLossStreak5'] = 1
```

Figure 38: The streaks function, which represents if a team is in very good or very poor form

Each new feature is again added as a column to the full dataset. This is then written to a CSV file for easier readability and to check if any mistakes had occurred. The data is read back into a new data frame which discounts the first three match weeks for each season. Irrelevant columns are then dropped, namely those that were made to create any difference features, as the difference is all that's required. It's important to get to know the data that's being dealt with, so some statistics behind the data itself were calculated. This can help to understand the predictors' reasoning also.



```
Total number of matches: 4108
Number of matches won by home team: 1910
Number of matches won by away team: 1159
Number of matches drawn: 1039
Win rate of home team: 46.49%
Win rate of away team: 28.21%
Draw rate: 25.29%
```

Figure 39: General facts calculated from the full dataset

As can be seen, the most likely outcome of a match is that the home team will win. Essentially more than 4 games out of 10 will end in a home win. This is quite interesting given that the front-end part of the application deals with 10 matches at a time for each week. The predictor takes all of this into account as well. A scatter matrix of all the remaining features was also composed through the built-in matplot library. This can help to see data trends across the set, and gives an insight into what the predictor itself will be dealing with.

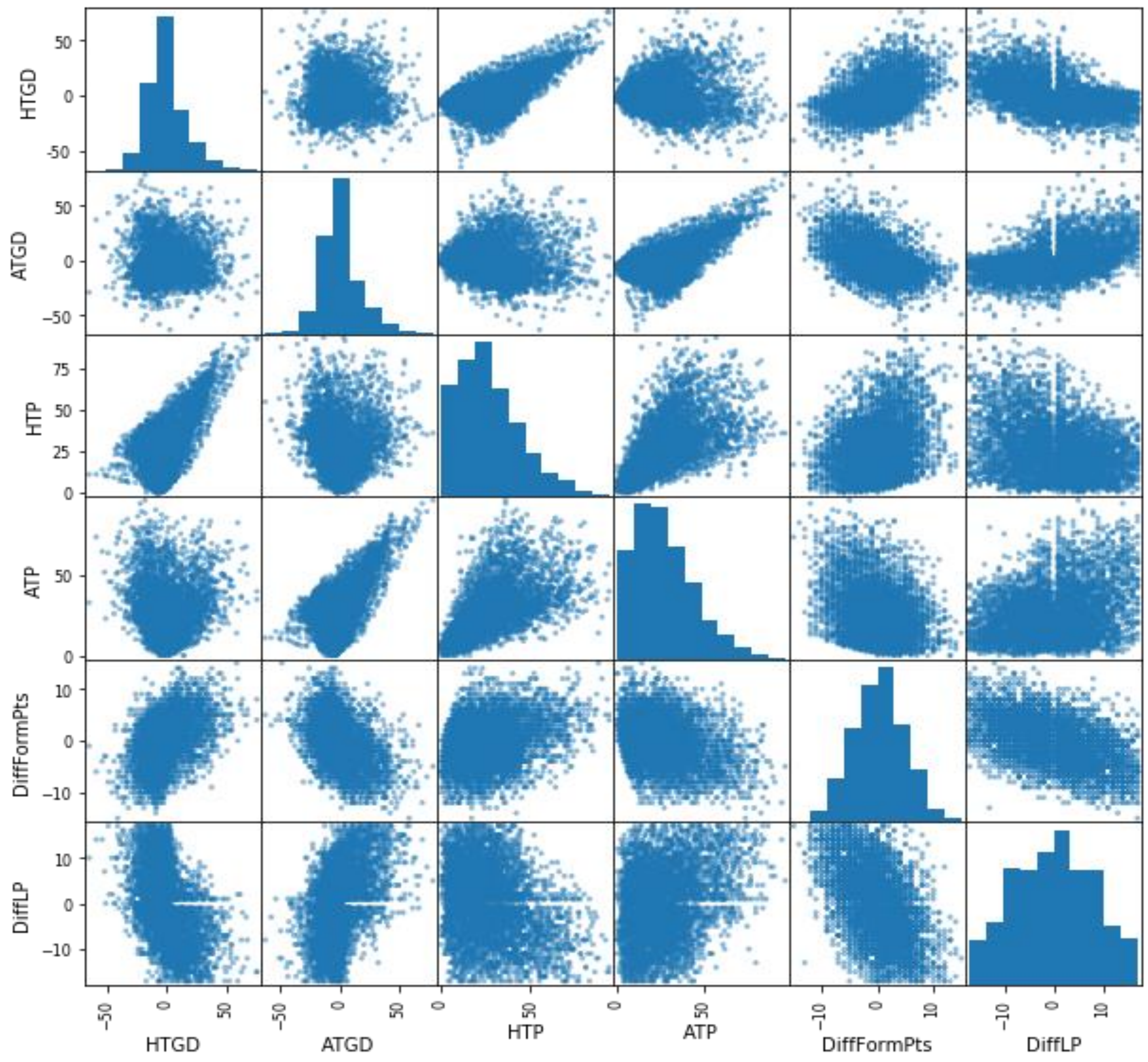
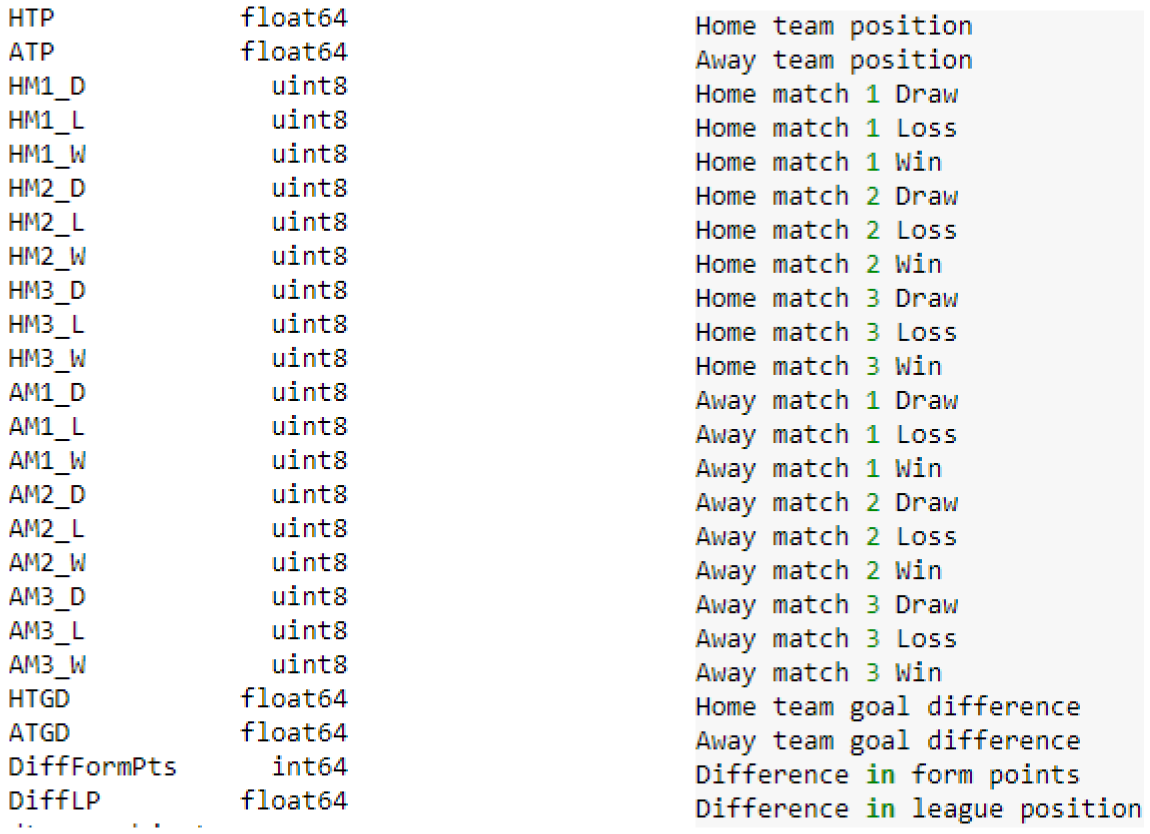


Figure 40: A scatter matrix created for all features used in the predictor

The full dataset is then split into a target set, containing just the full-time results, and another set containing everything else but the full-time result. The columns are then preprocessed using Pandas' get dummies feature that sets all data types in the data set to integers if they were objects to begin with. Machine learning models require a numerical input, as it's a lot easier to track the data that way so it can be trained properly. Words are difficult to analyse for algorithms like this and they tend to not work even in the slightest when objects or strings are passed through. The set itself remains an object however but that's irrelevant when it comes down to initialising the model, it's the features that are most important.



```
HTP              float64
ATP              float64
HM1_D              uint8
HM1_L              uint8
HM1_W              uint8
HM2_D              uint8
HM2_L              uint8
HM2_W              uint8
HM3_D              uint8
HM3_L              uint8
HM3_W              uint8
AM1_D              uint8
AM1_L              uint8
AM1_W              uint8
AM2_D              uint8
AM2_L              uint8
AM2_W              uint8
AM3_D              uint8
AM3_L              uint8
AM3_W              uint8
HTGD             float64
ATGD             float64
DiffFormPts        int64
DiffLP           float64
```

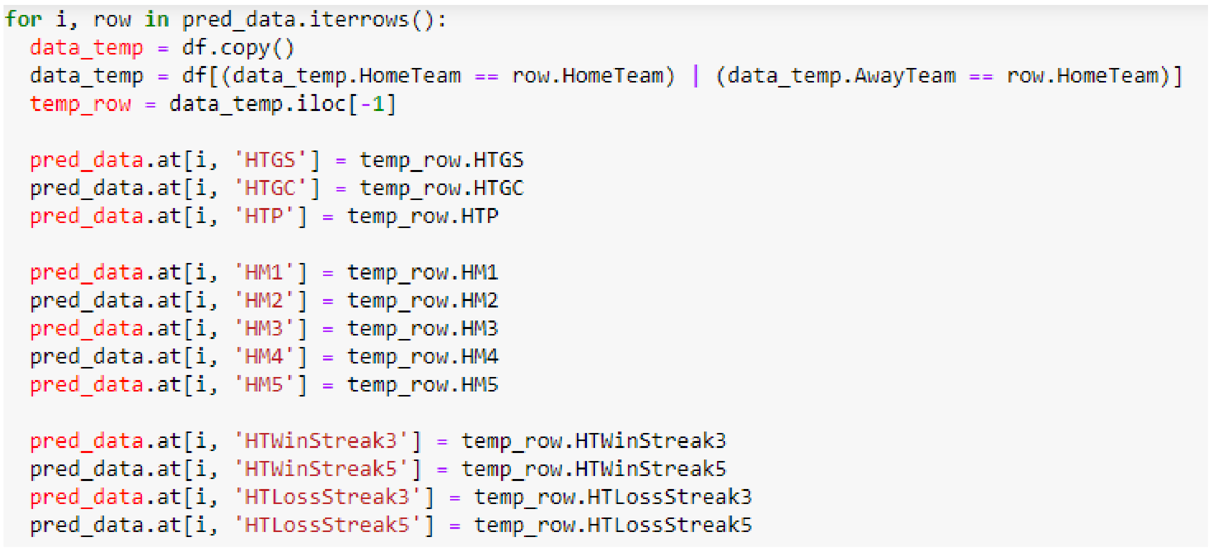
```
Home team position
Away team position
Home match 1 Draw
Home match 1 Loss
Home match 1 Win
Home match 2 Draw
Home match 2 Loss
Home match 2 Win
Home match 3 Draw
Home match 3 Loss
Home match 3 Win
Away match 1 Draw
Away match 1 Loss
Away match 1 Win
Away match 2 Draw
Away match 2 Loss
Away match 2 Win
Away match 3 Draw
Away match 3 Loss
Away match 3 Win
Home team goal difference
Away team goal difference
Difference in form points
Difference in league position
```

Figure 41: Each feature included in the training set after preprocessing and what they represent

Both sets are then split using Scikit-Learns train_test_split functionality. Arguments that are passed through this include the size of the test set and whether it is to be randomised or not. The last 50 games, representing the last 5 games for each team, is a reasonable barometer to go by and was chosen to be the test sets size. There is no point in randomising as this predictor is aiming to predict future matches, and matches picked from years ago wouldn't represent a good test. With all features preprocessed and a training set and test set created, a model then needs to be formed to fit all of this and create a test prediction.

The initial plan here was to create a unique model through the TensorFlow library or Fastai, but little research has been done into either for this kind of project and it was best to go with a ready-made version from Scikit-Learn's library. These algorithms were a lot more familiar having done a lot of research into each one that may be related in any way to football match prediction. The one that applied the most after looking into it and especially after testing was Extreme Gradient Boosting (XGBoost). Integrating this model into the script simply required fitting a classifier variable onto the training sets and then applying it in the correct manner. While its initial performance was poor on the test set but exceptional on the training set, the possibility of tuning its hyperparameters opened itself up. This is a complicated procedure to do for a lot of models but for XGBoost there are only a few lines of code required. The accuracy is then monitored to see if there are any improvements. For this particular model there was, with a jump of 10% accuracy on the test set.

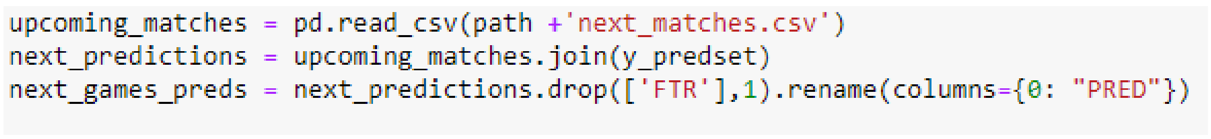
However, now that an optimal accuracy has been found on a reasonably large testing set, this all needs to be applied to future fixtures. Another dataset was made consisting of upcoming fixtures with only the home team, away team and full-time result as columns. The full dataset is read back in with the same features added into the predictor copied onto the upcoming fixtures set. The predictor is expecting all of these features again in order to make a feasible guess at what is to happen next. The data frame is copied over, with the most recent data for each team being passed on. This is fine for most of the features but all those that involve differences will need to be recalculated, which is a straightforward task as seen already.



```
for i, row in pred_data.iterrows():
  data_temp = df.copy()
  data_temp = df[(data_temp.HomeTeam == row.HomeTeam) | (data_temp.AwayTeam == row.HomeTeam)]
  temp_row = data_temp.iloc[-1]

  pred_data.at[i, 'HTGS'] = temp_row.HTGS
  pred_data.at[i, 'HTGC'] = temp_row.HTGC
  pred_data.at[i, 'HTP'] = temp_row.HTP

  pred_data.at[i, 'HM1'] = temp_row.HM1
  pred_data.at[i, 'HM2'] = temp_row.HM2
  pred_data.at[i, 'HM3'] = temp_row.HM3
  pred_data.at[i, 'HM4'] = temp_row.HM4
  pred_data.at[i, 'HM5'] = temp_row.HM5

  pred_data.at[i, 'HTWinStreak3'] = temp_row.HTWinStreak3
  pred_data.at[i, 'HTWinStreak5'] = temp_row.HTWinStreak5
  pred_data.at[i, 'HTLossStreak3'] = temp_row.HTLossStreak3
  pred_data.at[i, 'HTLossStreak5'] = temp_row.HTLossStreak5
```

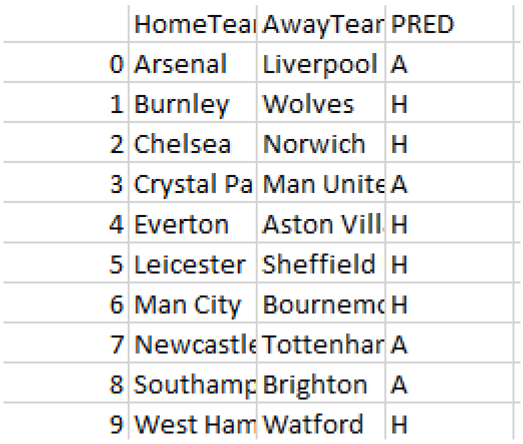
Figure 42: Each feature copied over from the main dataset to the upcoming fixtures set

The upcoming fixtures dataset only contains the next 10 fixtures. It is possible to have it simulate the rest of the season, but with the application asking users to predict 10 fixtures at a time, it makes the most sense to keep it this way. Once the tasks are redone with all of the data similarly to before with the training and test sets, the predictor, with the hyperparameters set the same as before, can now guess what will happen in these future matches. Adding these to a readable CSV file that can be read and printed on the front-end was the next task. This simply involves converting the predictions, which are outputted as a NumPy array, to a data frame, and then dropping the full-time result column in the upcoming matches data frame and adding the predictions on as a new column.



```
upcoming_matches = pd.read_csv(path +'next_matches.csv')
next_predictions = upcoming_matches.join(y_predset)
next_games_preds = next_predictions.drop(['FTR'],1).rename(columns={0: "PRED"})
```

Figure 43: Adding the predicted results alongside each upcoming fixture



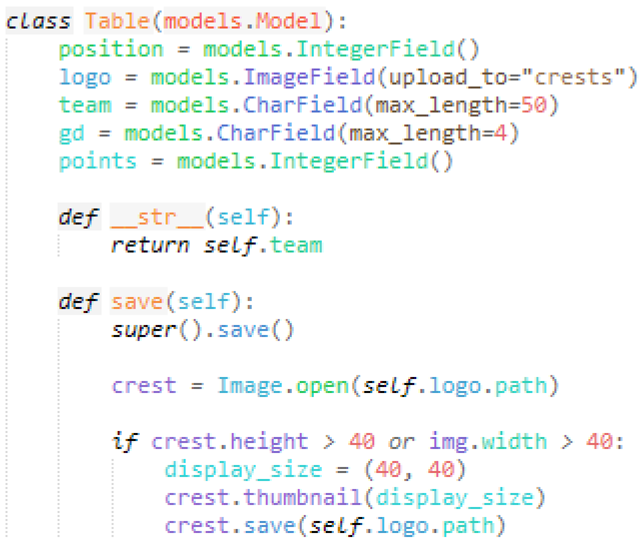
	HomeTeai	AwayTear	PRED
0	Arsenal	Liverpool	A
1	Burnley	Wolves	H
2	Chelsea	Norwich	H
3	Crystal Pa	Man Unite	A
4	Everton	Aston Vill	H
5	Leicester	Sheffield	H
6	Man City	Bournemc	H
7	Newcastle	Tottenhar	A
8	Southamp	Brighton	A
9	West Ham	Watford	H

Figure 44: The predictions made for the next round of games

4.5. Back-End

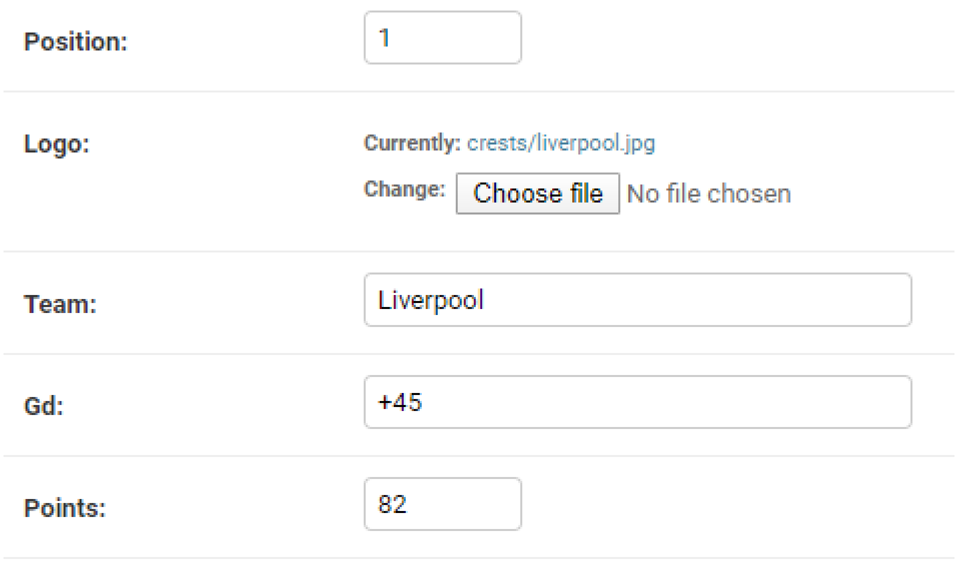
The prototypal database only contained user data, a username and a password. The next step for this was to implement a secure authentication system with the login, where usernames cannot be the same and passwords have to be of a certain strength to be accepted. As well as user data, the back-end database contains the current league standings, the next round of fixtures, different predictions from different users, the last match weeks results and scores for each user. The user has to have a profile to make their experience of the application more personified. This is fulfilled by creating a separate users app within the project, with a profile model included in it. This model is conveyed on a page where the user can update their profile picture, username or email. The profile model was then updated to include the users score up to that point and for that particular week.

Given that an API couldn’t be procured to supply some of the up-to-date data, the non-user data had to be manually updated. With the league table model, there are 20 instances for each position, and when the table changes each team that moves is changed around. This can admittedly only go so far as when matches are being played, the table is changing almost every second, so the table can only be updated after games have concluded. The table model contains fields for the team name in that position, their crest which is uploaded through an image, the teams goal difference and the amount of points they have. Uploading the crests proved a slightly tricky task at first as it wasn’t clear that they required their own folder to go to. This was rectified swiftly however with the use of one media folder which also contained folders for profile pictures and stadium pictures which are used for other models. Including the goal difference is important as it informs the user of why a team with the same amount of points as another team is above that team. The whole model is then printed by order of position to the home page where it can help the user.



```
class Table(models.Model):
    position = models.IntegerField()
    logo = models.ImageField(upload_to="crests")
    team = models.CharField(max_length=50)
    gd = models.CharField(max_length=4)
    points = models.IntegerField()

    def __str__(self):
        return self.team

    def save(self):
        super().save()

        crest = Image.open(self.logo.path)

        if crest.height > 40 or img.width > 40:
            display_size = (40, 40)
            crest.thumbnail(display_size)
            crest.save(self.logo.path)
```

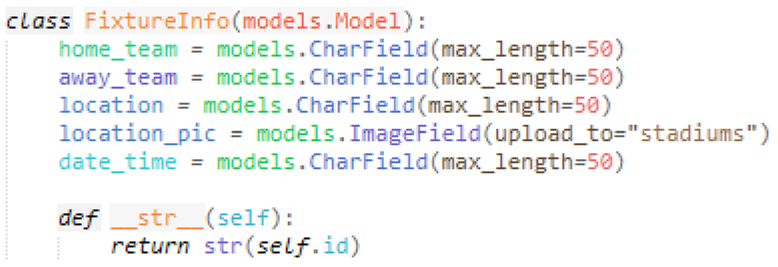
Figure 45: Creating the Table model which displays the Premier League standings



Position:	1
Logo:	**Currently:** crests/liverpool.jpg **Change:** Choose file No file chosen
Team:	Liverpool
Gd:	+45
Points:	82

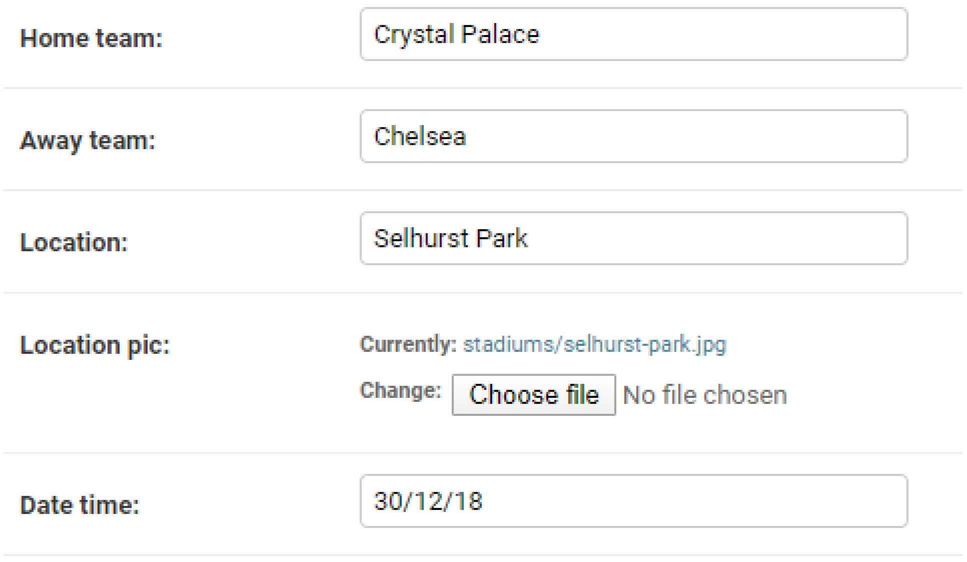
Figure 46: An example of an instance in the Table model

The league table model was a simple introduction into working with the back-end of Django. Moving into the fixtures model is where complications started to arise however. This model had to be linked with the user predictions model and this can be challenging with a lack of experience using SQLite. Foreign keys don't work the same way as with normal SQL as they can only be used for a whole model, instead of a field within a model which would have been a lot more straightforward. Within the fixtures model itself however were basic details about each match that is coming up for that particular week. These details include the home team, away team, the stadium at which the match is being played (most probably the home teams stadium), an image for that stadium, and the date and time of when that match is being played. Similarly to the league table model, this is changed as the next round of matches are played, and the contents are simply displayed to a page to aid the user. It is assumed after seeing the league table and then seeing who is playing who that the user will be able to make a logical guess as to how that match might go.



```
class FixtureInfo(models.Model):
    home_team = models.CharField(max_length=50)
    away_team = models.CharField(max_length=50)
    location = models.CharField(max_length=50)
    location_pic = models.ImageField(upload_to="stadiums")
    date_time = models.CharField(max_length=50)

    def __str__(self):
        return str(self.id)
```

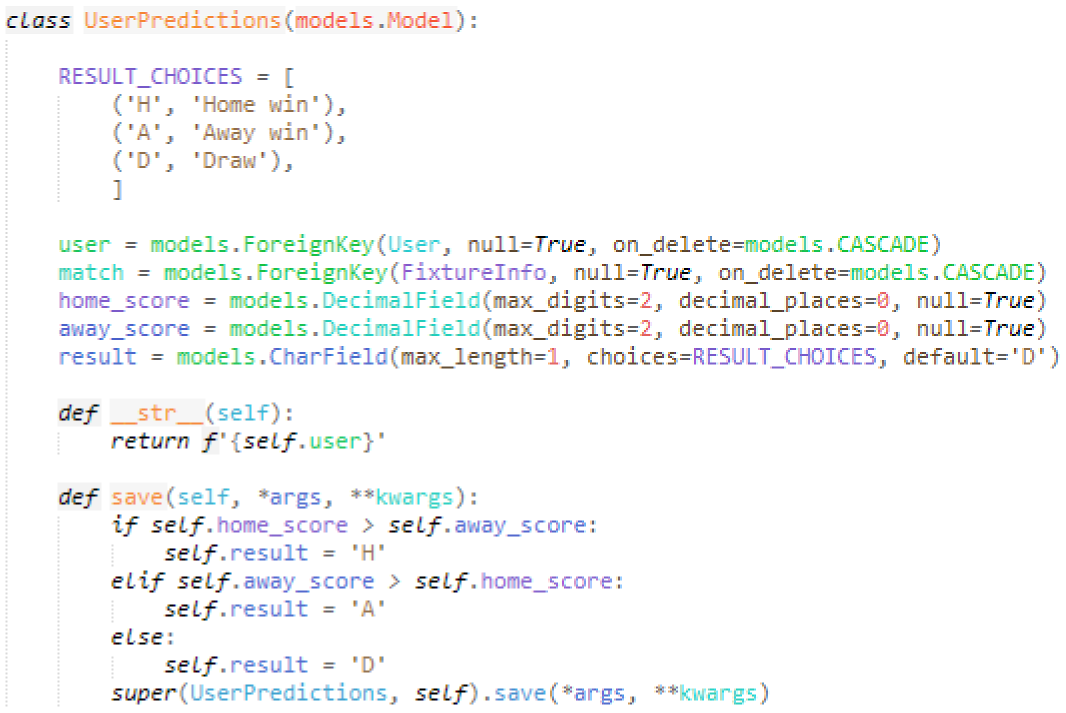
Figure 47: Creating the fixtures model



Home team:	Crystal Palace
Away team:	Chelsea
Location:	Selhurst Park
Location pic:	Currently: stadiums/selhurst-park.jpg Change: Choose file No file chosen
Date time:	30/12/18

Figure 48: An instance for one of the fixtures

The predictions model is supposed to be directly linked through a match field to the fixtures model, but this is where complications began. The predictions model consists of the user making the predictions (the user logged in who fills out the form set), the match (defined by number 1-10), the home teams predicted score, the away teams predicted score and the full-time result. The only fields that the user adds are the home score and away score, the result is generated based on if there are more home goals than away goals or vice versa, or if they are the same. The results default value is also entered as a draw because if a user leaves one of the forms blank, it's assumed the input is 0-0. It would become a bit confusing for users if they had to fill out the ID for a match alongside this, so instead each form in the form set is displayed from first to tenth. Adding an input field for the match would have solved the link between the two models, showing which match a result was entered for. This was deemed as something that could be added manually however, as each user makes 10 predictions in the same order of how the fixtures are listed. For example, if the first match played on a weekend was played early on a Saturday, then the ID for that match would be 1, and would be the first match the user predicts in the list.



```
class UserPredictions(models.Model):

    RESULT_CHOICES = [
        ('H', 'Home win'),
        ('A', 'Away win'),
        ('D', 'Draw'),
        ]

    user = models.ForeignKey(User, null=True, on_delete=models.CASCADE)
    match = models.ForeignKey(FixtureInfo, null=True, on_delete=models.CASCADE)
    home_score = models.DecimalField(max_digits=2, decimal_places=0, null=True)
    away_score = models.DecimalField(max_digits=2, decimal_places=0, null=True)
    result = models.CharField(max_length=1, choices=RESULT_CHOICES, default='D')

    def __str__(self):
        return f'{self.user}'

    def save(self, *args, **kwargs):
        if self.home_score > self.away_score:
            self.result = 'H'
        elif self.away_score > self.home_score:
            self.result = 'A'
        else:
            self.result = 'D'
        super(UserPredictions, self).save(*args, **kwargs)
```

Figure 49: Creating the user predictions model

These predictions need to be compared to the final scores, however. A final scores model that links back to the fixtures model was created. When determining the score that the user obtains from each match, these two models are put together to see if the user has gotten the result or the exact score correct. This ties into a scores model, which adds points on where they are won. The final scores model is changed constantly as the matches are also changed, so while this model is refreshed, the users score is added to if they got any predictions correct. Again these updates are done manually. The scores have to be added to the scores field in the profile model, while a separate field calculates how many points the user won over that week. The initial plan of displaying the users average points per week had to be left out however. This is because the results per week couldn't be saved easily. Upon further research other existing systems didn't have an aspect like this either. While it would have been ideal to have more to make this project as unique as possible, this was unfortunately a bridge too far and other components had to be prioritised.

4.6. Conclusions

Developing the application as a whole had different challenges for each respective layer. In particular, the front-end and middle-tier required a lot of work. The predictor was created and tested first, as it’s the main aspect that sets the application apart from what’s already out there. In terms of the front end, the design and a lot of functionality ended up changing noticeably from the prototype and the initial design structure. This mainly came down to being unfamiliar with the Django framework when starting development. Learning how it worked came quickly enough, and it also became clear that some things would have to change along the way with Django working in its own way. The lack of an API also became more noticeable the more the application was developed as well. At first, an API was to be used to scrape data from, however when developing the back-end data it became clear that an API would have been able to update some of the data itself, namely the Premier League table and upcoming fixtures. It is a slightly more tedious approach to have to update this data everytime matches are played, but even with an API there would still have been aspects that would have to be updated constantly. For example, the API wouldn’t know to update every user's score after each match week, so that would still need to be done. Aside from this though the application runs as well as was hoped before developing it. The testing and evaluation phase will truly determine if this is the case however.

5. Testing and Evaluation

5.1. Introduction

The development of the project is one thing, but testing it is another case altogether. The system has to be critiqued properly in order for it to meet the standards. In this chapter, the ways in which the application is tested and evaluated are discussed. Both testing and evaluating are as important as each other in trying to make the project reach its full potential. Without undertaking tests or evaluations it wouldn't be clear as to where the application stands or how far away from the final deliverable it is. The planning and execution of testing and evaluating the system is outlined, as well as obstacles that presented themselves throughout the process. Once these are completed, the results are analysed with changes made accordingly.

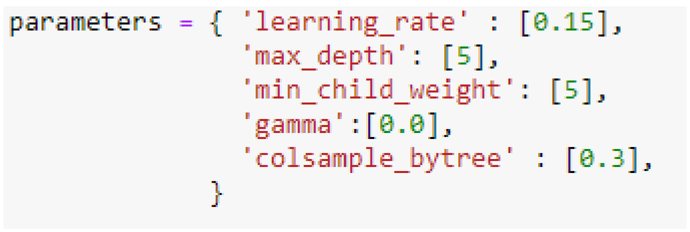
5.2. System Testing

It's crucial for the project to be tested constantly throughout the development life cycle, and thereafter. This is a web application only so constantly testing every single aspect of each tier functionality-wise as well as design-wise is a vital part of the whole project.

The project needs to be backed up wherever possible. Version control plays a big part because issues that may present themselves can be isolated and previous versions may hold the key to solving them. It also erases any doubt of the entire project being deleted. Multiple editions of the project would be harder to delete than a singular entity. GitHub is a highly effective tool here for rolling back previous versions in case something worked before that stops working, for example. Constantly committing new changes to the project after various tests achieves this.

The algorithm used for prediction could simply be tested by seeing how accurate it is in relation to matches on that week. A large component of the project is for the predictor to be as accurate as possible so it's difficult for users to beat. Using historical data and recent statistics that feed into a machine learning model this should occur. However, there are multiple different machine learning algorithms out there used for prediction, so a few samples were tested for accuracy before making a final decision. Also, despite the project not being used for betting purposes, comparisons to bookmakers' odds for matches to be tested can gauge how accurate the predictor is.

The COVID-19 pandemic highly affected this initial plan to test the predictor on upcoming matches. Due to the scale of the virus, the Premier League was suspended with the last fixtures being played on the weekend of the 7th of March. With no real-world case to test the predictor on, a back-up plan was put in motion. The test set used for the predictor included the last 50 games played in the league, ending with Leicester City's 4-0 win over Aston Villa. Four different Scikit-Learn algorithms were tested on this set - Decision Trees, Support Vector Machines, Extreme Gradient Boosting and Logistic Regression. Each of these were put through a round of functions whereby the speed at which they are trained and the speed of making a prediction was calculated, as well as their accuracy on the training and test sets respectively. All of these factors tied together would determine which algorithm the predictor would be based on. The accuracy on the test set would be of particular interest, because that basically correlates to how accurate the model is after being trained. This was calculated by finding the algorithms F1 score. The formula behind this is 2*((precision*recall)/(precision+recall)), but fortunately Scikit-Learn has a built in tool that works it out. Some algorithms performance were outright poor, such as that of the Decision Trees. Despite having a perfect 100% accuracy on the training set, it had a 48% accuracy on the test set, meaning that less than half of the matches passed through were guessed correctly. The accuracy of the test set for both the Support Vector Machine and Logistic Regression were reasonably better, with an accuracy of 54 and 52 respectively. While it was up for debate between these two and XGBoost, the logical choice, given its performance on both sets, was XGBoost. With a 96% accuracy on the training set and 50% on the test set, it was clear that it understood the data very well. The training set accuracy for Logistic Regression and Support Vector Machine was similar to the test set accuracy, so when applying either of them to future matches they would probably struggle. There is also a straightforward way of changing XGBoost's hyperparameters. By altering aspects such as its learning rate or maximum depth, the accuracy on the test set would hopefully improve. Admittedly, it is quite difficult to know what hyperparameters would augment the algorithm's accuracy to its full potential, so a number of different combinations were tried out. The F1 score ended up rising 10% to 60% on the test set, meaning in theory for every 10 matches that are played, the results of 6 of them would be predicted correctly.



```
parameters = { 'learning_rate' : [0.15],
               'max_depth': [5],
               'min_child_weight': [5],
               'gamma':[0.0],
               'colsample_bytree' : [0.3],
             }
```

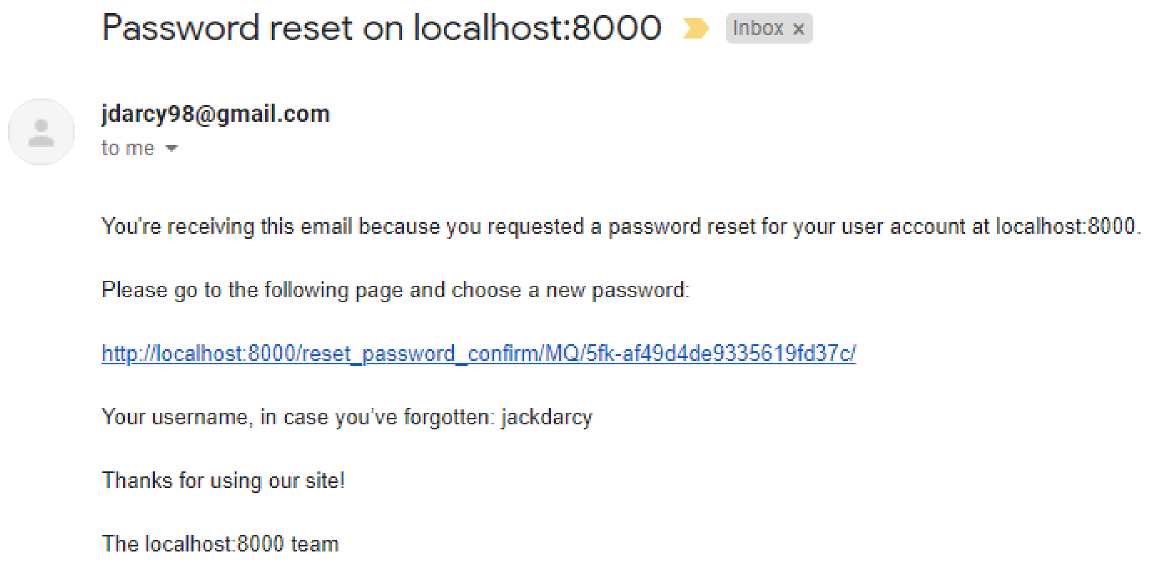
Figure 50: The hyperparameters in place that increased XGBoost's accuracy

The ideal scenario that was developed was to predict the next round of games, which was firstly going to be the weekend of the 2nd of May, but then the season was suspended indefinitely. The next best thing to do in the circumstances was to base the predictor on the second half of last season, the 2018/19 season. The results are already known for all of the games, but using what the predictor guessed against the bookmakers odds, as per the original plan, would be a viable test. The seasonal datasets used for training the model contained bookmakers odds in decimal places to begin with. During the development these were removed though, as this project was not meant to be geared toward betting in any way. However, for testing purposes, these odds are a great tool to determine if the predictor could be a better standard to go by. Together with the 2018/19 season’s results, this test can mimic what was planned in the first place. Unfortunately however, the aspect of passing the user's gut feeling to the predictor, while it was tested in development, had to be left out of the final deliverable given that a real-world case would be needed to use it properly.

The predictors development had to be changed slightly to make this happen. The 2019/20 season is left out in this case, but the model retains its accuracy on a larger test set through XGBoost. 60% of the final 190 results from last season were guessed correctly. This is then compared to Bet365’s odds for the same matches. It was quite an interesting process as a few things were noticed. The odds for these games never saw the draw as favourite, while the predictor did and in some cases prevailed over the odds by backing the draw. There was one case where the odds for a home win or away win for the same match were the same, so this couldn’t be counted. In the end, the odds ended up guessing the correct result 57% of the time. While the predictor didn’t beat the odds by much, it still beat them. It's also unclear as to what Bet365 use to generate their odds, but what is clear is that more features can be added to this predictor, which will only make it more accurate. Beating an official standard over half a season is a fair achievement. Considering the model can only be improved upon as well the predictor passed the testing procedure soundly.

The predictor is only part of what needs to be tested though. The full user experience needs to be tracked also, while checking that everything is working as well as it should. This project is centred around user integration so the importance of the user not facing any errors throughout their time on the application is huge.

The test plan below was undertaken constantly, as per the Waterfall model. It was important to ensure each aspect was working to its full capacity before moving on to the next feature. Some features, such as changing a users password, only work in certain cases however. Django includes a feature where an automated email can be sent to authenticate a user so they can safely change their password. This was tested on an external email address but because the application is technically unknown, the email wouldn't send. However on Gmail there is a way of switching an email address to allow for unknown applications emails to be received. Of course this isn't recommended in the long term but for testing purposes it was necessary.


Password reset on localhost:8000

jdarcy98@gmail.com
to me

You're receiving this email because you requested a password reset for your user account at localhost:8000.

Please go to the following page and choose a new password:

http://localhost:8000/reset_password_confirm/MQ/5fk-af49d4de9335619fd37c/

Your username, in case you've forgotten: jackdarcy

Thanks for using our site!

The localhost:8000 team

Figure 51: Sample email sent when a user needs to change their password

Potential users were asked to browse through the application and offer their thoughts on what could be improved. Below are questions that double up as requirements that have to be met for the user. Each one must be passed so the finished article can start to take shape. These potential users would be football fans ideally as interest in the sport is paramount for the application to be understood fully. However this group had to be narrowed down to some classmates due to the pandemic. Below is an example of one of the tests.

Test Plan

Test no.	**Test Description**	**Expected Outcome**	**Pass/Fail?**
1.	Does the website load when the address is entered into a browser?	The web app loads and the user begins at the home page.	PASS
2.	When a user closes their browser	The web app shuts	PASS

	and/or navigates away from the web app, does the site close?	down when not being used.	
3.	Can a user create an account with a unique username?	User creates an account secured with a password.	PASS
4.	Is user data being passed to the database?	Username and password entered to create an account is saved in the back-end database.	PASS
5.	With an account, can a user login?	Registered user can login and is transported to the Play Now page.	PASS
6.	User tries to login with incorrect details.	Error message is displayed for the user to see.	PASS
7.	Can a user sign out?	User is signed out and returned to the home screen.	PASS
8.	Can a user change their password?	Users password changes and database is updated.	PASS
9.	Can a user delete their account?	Account is deleted and removed from database	FAIL
10.	User enters score predictions.	These values are	Stored in

		stored in a database and compared to the predictors predictions.	database but can’t be compared
11.	User clicks a button to check predictors' guesses.	20 points are deducted from the users account and they are taken to the predictor page.	User can check predictions but points aren’t deducted. Warning message displayed instead.
12.	User clicks on match on predictor page.	Statistics used in the algorithm to reinforce the predictors' guesses are printed from the dataset.	PASS
13.	User checks results.	User is taken to a profile page where their points are displayed.	PASS
14.	Can the user see the predictors score for the week on the results page?	The predictor's score is displayed, but not what they guessed or why.	Predictors score removed from the project.
14.	User checks leaderboard page.	The Premier League	PASS

		table is displayed along with any head-to-head leagues they are in and the overall leaderboard.	🟩
15.	“Join a league” button clicked.	User is taken to join a league page.	FAIL Overall leaderboard used instead
16.	Can a user create a league?	League is created with randomly generated code for others to join.	FAIL
17.	Can a user join another league?	User enters in code and is added to the league.	FAIL

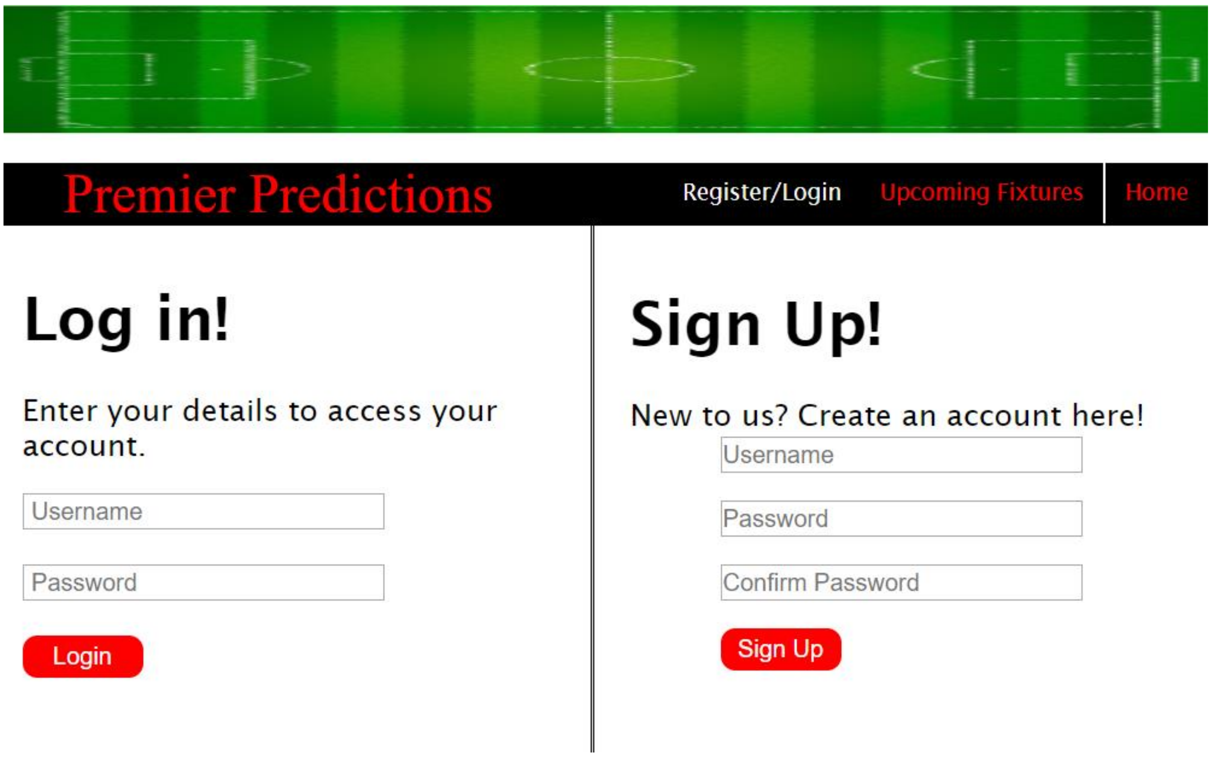
Figure 52: Test Plan in place for applications basic functionalities

5.3. System Evaluation

To properly meet an industry level of quality, it’s pivotal to take the opinions of users into account. As mentioned already, football fans are the target market for this application, so they are the perfect candidates to evaluate the system. From previous projects that were researched it’s clear that different people can see different possibilities with applications, and changes are made based on them. Another person can spot things that maybe don’t appear clearly to the developer. In saying this, the usability of the application can also be tested here.

The standard of applications in general is quite high now so getting users to test it out can only be an aid. Nielsen’s Heuristics were discussed previously and they will act as a benchmark for the usability of Premier Predictions. Despite the complicated nature that may arise from the development, the primary aim is to make this very easy to use and to navigate. Other possible volunteers included people with IT experience in the industry right now. They can offer an opinion either from their own previous work or simply from what they think is suitable.

Of course with the pandemic mentioned previously, this ideal plan couldn’t work. In its place, classmates were asked to evaluate the application overall before Django was fully introduced. Django was always inclined to improve the background functionality but these opinions are valued nonetheless. Other people in the same field are bound to think differently about certain aspects, and taking these on board to make the application more universal is vital. With that being said, some of the things that had to be changed anyway were pointed out for needing improvement, such as the design. The application has come a long way design wise since the evaluation. Most of the work done at the time of evaluation was with the predictor, which can only be critiqued through the methods already discussed. Given the lack of experience with Django however, it was good to have some direction as to what exactly to change with the design and functionality. For example, using Bootstrap to optimise the design was one recommendation that certainly changed the application for the better.



Log in!

Enter your details to access your account.

Sign Up!

New to us? Create an account here!

Figure 53: Login page before evaluation

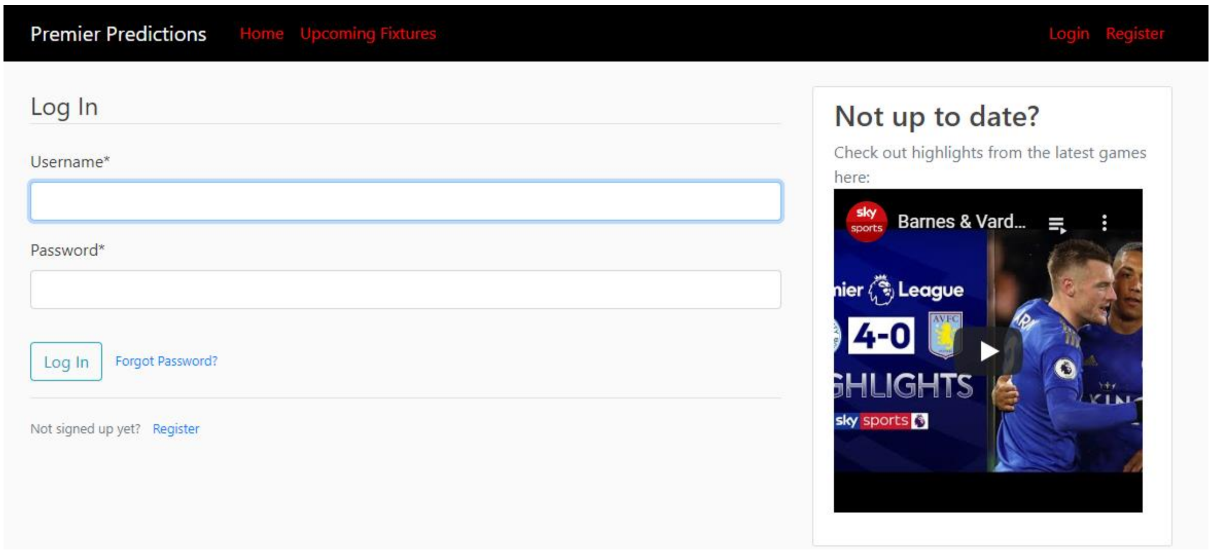


Figure 54: Login page after evaluation

5.4. Conclusions

This chapter outlined how the project was tested and evaluated, and how it changed from the original plan. Without the pandemic the plan would have been adhered to, but due to its unprecedented nature, the plan had to change. It was great to see the predictor have a higher accuracy than the bookmakers odds, but ideally this would have been done with a user's predictions. The user prediction can take a lot more things into account, and without being swayed by bias or by looking at the favoured result they could possibly beat the predictors accuracy over a number of weeks of testing. However, the input of the users who volunteered to go through the application in its reasonable early stage of development was more than helpful. Using Django was a new experience and the feedback made it easier, despite some aspects not working to full capacity in the end. The questions that had to be asked when testing the application were also discussed. These were asked every time a change, minor or major, was needed that could have altered the whole flow of the project. The importance of this whole process was very significant as changes had to be made, and a bit of guidance using Django was needed so that the project could reach its full promise. There would also be little point in having a predictor that isn't tested, so obtaining the F1 score, tuning the hyperparameters and comparing it to bookmakers odds are all worthwhile tasks for what is the main complexity of the overall project.

6. Conclusions and Future Work

6.1. Introduction

When completing a project of this scale, there were bound to be some issues that would present themselves over time. Challenges were always going to be faced on the way and this chapter discusses each one that was faced to date. Getting through these issues and dealing with them effectively is what will improve the project in the end. Some problems were foreseen and others simply came from nowhere. This chapter also talks about the conclusions drawn from the project's development and how it may be improved through future work.

6.2. Conclusions

When researching existing systems to try make this project as unique as possible, it was clear that the user integration would be the main attraction. Between interacting with the user through the application layer, and adding their prediction as a parameter to the predictor, there are aspects that make Premier Predictions stand out from what is already out there. Using an API as was initially intended could have made it even more unique, as would the development of a machine learning model through TensorFlow or Fastai, as opposed to using one of Scikit-Learn's ready-made algorithms. There is value in using these however as was seen through the testing procedure. With further research into either of these libraries though a more tailored model could have been constructed.

The design changed noticeably throughout the process. The final application mostly maintains what was intended with the skeleton prototype that was drawn out. The interim prototype compared to the final application however is almost unrecognisable. The back-end database design was changed somewhat given that SQLite had not been used yet, and the relationships weren't fully analysed in the application's context. The overall system architecture had to change as well with the API being abandoned. Instead, a number of CSV files are used as data passed into the predictor which is then printed out on the sites predictor page. The use cases were mostly adhered to as well. These were drawn out in detail because of the user experience being so important.

The development aspect of the project, admittedly, could have gone better. While the predictor was the main complexity of the project, it could be said that too much time was spent developing it, neglecting the application slightly. The time spent developing the predictor could have been lessened in favour of trying to develop parts of the application that ended up not working as intended, such as the scoring system. The predictor did end up working and performing better than initially expected, especially after having to change from an API to the CSV datasets. The fact that some components of the user input have to be done manually is disappointing and probably represents the lowest point of the project overall. However the final design and other functionalities are as good as was hoped. The user experience as said before is the most important part of the application and for the most part it must be seen as a quality experience.

The testing and evaluation plan had to be changed because of the unprecedented COVID-19 pandemic. There was nothing to be done about this but to change the plan to cater for the new situation. The predictors accuracy was still measured by its F1 score, but unfortunately the uniqueness of testing it with and without the users gut feeling couldn't be done because of the pandemic erasing any real-world situation. While predictions were made for what was thought to be the next round of fixtures, the accuracy in this respect couldn't be tested. In its place came the bookmakers odds, which are seen as money-makers so must be reasonably accurate at the very least. The predictors accuracy ended up being greater than that of the bookmakers', which gives a sense of achievement after developing it from start to finish. This is definitely one part of the project that ended up working well, even after testing. The idea of getting football fans to evaluate the application also had to be discontinued due to the virus. The backup for this was to send what had been done before the work with Django had fully started to some classmates, to get them to evaluate it. Again this acted as the next best thing as plenty of recommendations were given as to how to improve the application overall through design and functionality. Some even gave a few tips on how Django would work with my project in particular and the changes ended up being drastic.

Overall the project fulfils the majority of requirements set out at the start. It's unfortunate that some aspects had to be left out as circumstances didn't allow them to be tested properly, leaving no point in including them because it wouldn't have been known for sure if they didn't work. The end result of the predictor could have been better too, but it still ended up being a highly useful tool to have. Predicting 60% of a given set of matches compared to the bookmakers' 57% is no mean feat, especially when it cannot be regressed.

6.3. Future Work

There is more room for future work than first anticipated. The features that didn't work properly in the end can be optimised with some more work. More and more features can also be added to make the application stand out amongst previous systems that were researched. With more research into Django and SQLite this can certainly be done. The lack of experience with both is what caused the application to trip up slightly in the end, but as mentioned before, there were bound to be challenges to face along the way. An API could certainly be a big step to doing this, as with an interface that updates itself regularly, the application can go to the next level.

With the predictor, there will always be room for improvement no matter how much work is done on it. An accuracy of 60% is a great start with the limited amount of features included, so adding more things like the weather as was first intended, or social media sentiment can make the predictor go from strength to strength. The logical next step in terms of output would be to predict the exact score of a match. While the accuracy of full-time results beats the bookmakers, they also have favoured odds on final scores and scores at half-time. The possibilities with the predictor are endless and exciting to say the least.

Bibliography

1. Kunz M. A large-scale FIFA survey involving its then 207 member associations shows that football has strengthened its position as the world’s number one sport since the last Big Count in the year 2000. Among the most pleasing signs is the continuing growth of the women’s game. 2006;3.
2. FIFA.com. 2018 FIFA World Cup^{TM} - News - More than half the world watched record-breaking 2018 World Cup - FIFA.com [Internet]. www.fifa.com. [cited 2019 Oct 30]. Available from: https://www.fifa.com/worldcup/news/more-than-half-the-world-watched-record-breaking-2018-world-cup
3. Olympic Games: TV viewership worldwide 2016 [Internet]. Statista. [cited 2019 Oct 30]. Available from: https://www.statista.com/statistics/287966/olympic-games-tv-viewership-worldwide/
4. Football betting - the global gambling industry worth billions - BBC Sport [Internet]. [cited 2019 Nov 11]. Available from: https://www.bbc.com/sport/football/24354124
5. Sky Sports popularity & fame | YouGov [Internet]. [cited 2019 Nov 12]. Available from: https://yougov.co.uk/topics/media/explore/tv_channel/Sky_Sports
6. What is API - Application Program Interface? Webopedia Definition [Internet]. [cited 2019 Dec 6]. Available from: https://www.webopedia.com/TERM/A/API.html
7. Ganegedara T. Intuitive Guide to Understanding Decision Trees [Internet]. Medium. 2019 [cited 2020 Apr 2]. Available from: https://towardsdatascience.com/light-on-math-machine-learning-intuitive-guide-to-understanding-decision-trees-adb2165ccab7
8. Support Vector Machines: A Simple Explanation [Internet]. KDnuggets. [cited 2020 Apr 2]. Available from: https://www.kdnuggets.com/support-vector-machines-a-simple-explanation.html/
9. Brownlee J. A Gentle Introduction to XGBoost for Applied Machine Learning [Internet]. Machine Learning Mastery. 2016 [cited 2020 Apr 2]. Available from: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/
10. Restrepo M. Doing XGBoost hyper-parameter tuning the smart way — Part 1 of 2 [Internet]. Medium. 2019 [cited 2020 Apr 9]. Available from: https://towardsdatascience.com/doing-xgboost-hyper-parameter-tuning-the-smart-way-part-1-of-2-f6d255a45dde
11. 11 BEST Python IDEs in 2020 [Internet]. [cited 2020 Apr 2]. Available from: https://www.guru99.com/python-ide-code-editor.html

12. Experience WL in R-BU. 10 Heuristics for User Interface Design: Article by Jakob Nielsen [Internet]. Nielsen Norman Group. [cited 2019 Dec 6]. Available from: https://www.nngroup.com/articles/ten-usability-heuristics/
13. Premier League History, Origins & List of Past Champions [Internet]. [cited 2019 Dec 7]. Available from: https://www.premierleague.com/history