**DOMAIN DRIVEN DATA MINING**

PREDICTIVE ANALYTICS TECHNIQUES ON PREMIER LEAGUE DATA

Daniel James Collins  
  
  
  
  
  
  
  
A project submitted in partial fulfilment of the  
requirements for the Glyndŵr University award of  
**B.Sc.(Hons) in Computer Science**undertaken within the Department of Computing,  
Institute for Arts, Science and Technology,  
Wrexham Glyndŵr University

May 2021

# Preamble

## Acknowledgements

Project Supervisor: John Worden

Guidance from: Vic Grout

## Abstract

The aim of this project is to build a data pipeline in Python and conduct data analysis techniques to identify the most influential factors towards winning a game and then using them as features. The features are then ran through various Machine Learning (ML) models and evaluated using a performance score. The performance score is then compared to a baseline score to determine its overall success rate at predicting future outcomes of football matches.

This document begins with an introduction to the project and states the goals, objectives, and layout of the report. The second section features a systematic literature review to address important questions; *Q1: What is the best model to use for Football Prediction? and Q2: Which are the most influential features to use to predict the outcome of a match.* After discussing the literature, the next section outlines and explains the methodology used and the process that is followed for creating the data pipeline, followed by evidence of the implementation detailing the steps taken to build the pipeline, refine the models, and generate performance scores. These performance scores are then evaluated and compared to a baseline to determine its predictive performance.

It can be concluded that the Support Vector Machine (SVM) and Random Forest (RF) models achieve the highest performance rating and are considered the most useful ML algorithms for predicting football outcomes using this pipeline.

Table of Contents

[Preamble i](#_Toc71663904)

[Acknowledgements i](#_Toc71663905)

[Abstract ii](#_Toc71663906)

[List of Program Code v](#_Toc71663907)

[Main Body 1](#_Toc71663908)

[Chapter 1: Introduction 1](#_Toc71663909)

[1.1. Background 1](#_Toc71663910)

[1.2. Aims and Objectives 1](#_Toc71663911)

[1.3. Layout of This Report 1](#_Toc71663912)

[Chapter 2: Literature Review 2](#_Toc71663913)

[2.1. Overview of Literature Review 2](#_Toc71663914)

[2.2. Advantages and Disadvantages of Widely-used Supervised ML Classifiers 2](#_Toc71663915)

[2.3. Widely-used Binary ML Classifiers 3](#_Toc71663916)

[2.4. How are Features Selected and Evaluated? 4](#_Toc71663917)

[2.5. Current State of the Art Predictive Models on Football Data 5](#_Toc71663918)

[Chapter 3: Methodology 17](#_Toc71663919)

[3.1. Preparation 17](#_Toc71663920)

[3.2. Initial Analysis 17](#_Toc71663921)

[3.3. Data Collection & Development Planning 17](#_Toc71663922)

[3.4. Data Analysis and Results 17](#_Toc71663923)

[3.5. Evaluation 18](#_Toc71663924)

[3.6. Reporting Results and Conclusions 18](#_Toc71663925)

[3.7. Project Planning and Timescales 18](#_Toc71663926)

[Chapter 4: Implementation 19](#_Toc71663927)

[4.1. Generate the Download Links of Source Data 19](#_Toc71663928)

[4.2. Data Cleaning 7](#_Toc71663929)

[4.3. Exploratory Data Analysis (EDA) 7](#_Toc71663930)

[4.4. Visualisations 7](#_Toc71663931)

[4.5 Model Building 7](#_Toc71663932)

[Chapter 5: Critical Evalutation of Project 8](#_Toc71663933)

[Chapter 6: Conclusion 9](#_Toc71663934)

[End Matter 10](#_Toc71663935)

[References 10](#_Toc71663936)

[Appendix A: Project Documentation 12](#_Toc71663937)

[A.1. Project Proposal 12](#_Toc71663938)

[A.2. Project Specification 14](#_Toc71663939)

[Appendix B: Program Code 16](#_Toc71663940)

## List of Program Code

The program code can be found in Appendix B – Program Code at the end of the document.

It includes four scripts;

* Main.py
* Download\_data.py
* Utils.py
* Exploratory\_data\_analysis.py

# Main Body

## Chapter 1: Introduction

### Background

Football is one of the most popular sports on the planet. Due to the often unpredictable nature of football results, it has quickly solidified its place as the most popular market for sports betting in the UK. [1] It generates lots of dynamic and complex data that, if analysed correctly, can help to consistently predict future outcomes of matches. This makes it an ideal environment for implementing a data pipeline.

The key questions this project addresses is *Q1: What is the best model to use for Football prediction and why?* and *Q2: Which are the most influential features to use to predict the outcome of a match?,* as stated in the Literature Review section.

The aim of this project is to build a data pipeline that can efficiently analyse large amounts of raw historic Premier League data in order to predict accurate outcomes of future Premier League games. Theoretical frameworks are followed to produce a data pipeline. The predictions to be made are categorical, making it a *classification* problem and therefore a classification technique is utilized. This gives us a probability for each class, which is then used to measure how likely an event is to happen. Multiple classifiers are used to test the model. The final results are then compared with the results from other complex models from professional odds-makers to determine its success.

Data is now being utilized in almost every industry and business function, and has become one of the most important factors in the success of production, labor and capital [2]. Big data analytics in particular is set to play a huge factor in the success of any economic innovation in the future [2].

### 

### 1.2. Aims and Objectives

The overall aim is to develop a data pipeline to transform raw match data into insightful information that can be used for predicting the future outcome of each match. This is achieved by completing the following objectives;

1. Locate, access and store the appropriate raw data in a database
2. Clean the raw data (removing outliers, imputing missing data, converting data types to a consistent format) so it is ready to be queried and explored
3. Generate features that have reasonable predictive power (features that tell us something about what we hope to predict)
4. Use the selected features in a predictive model to predict the outcome of sporting events
5. Evaluate the predictions by comparing their predictive accuracy rating to that of professional odds-makers

### Layout of This Report

The report begins with a preamble consisting of acknowledgements, a list of figures used, a list of the program code used, and a glossary, to help the reader understand the purpose, philosophy and context of the report.

The main body is split into 8 chapters:

* Chapter 1: Introduction
* Chapter 2: Literature Review
* Chapter 3: Methodology
* Chapter 4: Design
* Chapter 5: Implementation
* Chapter 6: Evaluation of Project
* Chapter 7: Critical Evaluation of Project
* Chapter 8: Conclusion

Each chapter has interrelated sub-sections to sort the information into an easy-to-follow format, in the hope it makes the information less overwhelming and easier to read.

The report is concluded with the end matter, which includes references used, an appendix section featuring project documentation, and the program code used, which may provide useful references to the reader for further examination or research.

## Chapter 2: Literature Review

### 2.1. Overview of Literature Review

This systematic literature review addresses two relevant questions; *Q1: What is the best model to use for Football prediction?* and *Q2: Which are the most influential features to use to predict the outcome of a match*.

A review of similar state-of-the-art models that use Premier League data is conducted to better understand of the advantages and disadvantages of different classifiers, to understand what methods and techniques are used to select and evaluate features. This information can be used as a basis for answering the above questions, and also states which classifiers and methods are best performing in similar models.

Overall, this literature review implies the best model to use for predicting Home/ Away results is the open-source framework XgBoost, whereas the best performing model for predicting draws is random forest. A common problem in all models is keeping the right features, and removing the lowest-performing features from the model. This review shows this problem can be addressed using statistical methods or hierarchical clustering techniques such as a dendogram to identify and remove highly correlated features.

I also researched the definition and components of a data pipeline, what the process includes and methods of cleaning data.

### 2.2. Advantages and Disadvantages of Widely-used Supervised ML Classifiers

**Supervised Machine Learning (SML)** is a type of **Machine Learning (ML)** algorithm used to solve a classification or regression problem. It works by teaching a model to produce the same class labels as its predefined class labels and generalizes accurately on unseen data [3]. Generally speaking, this is an algorithm that is able to predict the labels of unseen data, based on previous examples of labeled data. The aim of SML classification models is to separate classes of a problem by only using training data to make the margin as wide as possible [3].

The type of problem can be defined based on the output variable. The problem can be defined as either:

* **Binary classification** – refers to a problem that has two possible values for the output variable that can be categorised into one of two categories, such as 0 or 1. [4]
* **Multi class** or **Multinomial classification** – refers to a problem that has two or more possible values for the output variable, each of which is put into one of three or more classes. [4]

For the purpose of this literature review, we are going to be discussing Binary Classification, as that is most appropriate for Football data.

### 2.3. Widely-used Binary ML Classifiers

Here are some commonly-used binary ML classifiers and their associated problems:

* **Naive Bayes** – This method applies Bayes’ theory of assuming conditional independence between each pair of features [Naive]. This method performs well even with small amounts of training data and is fast, however the problem is that it produces bad estimates and is therefore only generally used as a quick estimator. [5]
* **Logistic Regression** – This is another basic and popular algorithm used in classification problems. It is the same as Linear Regression, but it applies the Sigmoid Function to deal with outliers. This technique is used to determine the relationship between the predictor variables and a binary outcome. It’s important to note that the predictor variables can be either categorical or continuous. [6] The biggest problem with using this technique is that it’s difficult to determine the correct predictor variables to be used, as if multicollinearity variables are used as predictor variables the outcome will be less precise. [6]
* **K-nearest** **Neighbors** –This algorithm assumes that similar data exists within close proximity. It works by sorting the data into a collection of distances from smallest to largest, and then choosing the first entries of the collection as the k variables, obtaining its label and then calculating the mode of all K labels. [7] Whilst this is a easy to implement, the problem with this algorithm is it performs increasing slower as the number of examples, predictors and variables increase. [7]
* **Support Vector Machine (SVM)** – This is another simple ML algorithm used in classification. It works by identifying multiple hyperplanes in an N-dimensional space (N being the number of features that uniquely classify the data points), with the objective of finding the plane with the maximum margin. The key advantage to using SVM is it achieves accurate results with minimal computational power on small data sets, however on the contrary it performs poorly on large data sets. [8] Another issue with SVM is if there is a higher number of features for each data set than the number of training samples, then it will underperform. [9]
* **Decision Tree** - This ML algorithm is a decision-making strategy that resembles a tree with branches representing decisions. Implementing this algorithm involves choosing which features and conditions are to be used. It is a great visualisation technique to help understand the data, and can perform on both numerical or categorical data. [10] The disadvantages include a risk of overfitting data by creating a tree that is too complex, instability caused by small variations in the data, and trees can become biased if the some classes begin to dominate. [10]

### 2.4. How are Features Selected and Evaluated?

Once the features have been determined, there are normally lots of features which creates a greater feature space dimension. This is not ideal, so the features need to be evaluated and only the best features selected. This process is referred to as *Feature Selection*.

A review of similar predictive models found that the method for feature selection is often dependent on the model, and often varies.

Model 1  
One predictive model reviewed [11], which was also based on football data, selected features by dividing them into two classes, *Class A* and *Class B*. *Class A* contained features for Home and Away teams and *Class B* contained *Differential* features. [11] By storing the *Differential* features in *Class B*, the information of two variables is encoded into one, reducing the problem hyperspace and therefore reducing the chance of the model getting stuck at a local minimum. It also has a better univariate distribution, making it easier to perform statistical techniques on as opposed to multivariate. They then tested the feature set with multiple binary ML classifiers to test its performance. [11]

Model 2  
Another model reviewed [12] was a model created with Python and is based on football data, specifically for use in the popular game Fantasy Football. This model evaluated its features by first removing any correlated features, as these features often capture lots of the same data as other features. To do this, they used a dendogram, which is a hierarchical clustering technique that groups features together based on similarity. [12] After reviewing the results in the dendogram, they removed features which were highly correlated with other features, and tested their performance again to double check if they negatively or positive influenced the performance, with features negatively influencing the model being removed. They then use a drop-column method to determine the remaining features’ importance. Any features that had a low performance score were removed. [12]

### 2.5. Current State of the Art Predictive Models on Football Data

#### Model 1: Predictive analysis and modeling football results using machine learning approach for English Premier League

In this journal, the authors built a generalized predictive model that is used to predict English Premier League results. They used feature engineering and exploratory data analysis techniques to create a feature set which is used to determine the most important factors for predicting the final result of a football match. By using machine learning they have created a highly accurate predictive system. [11]

The data source used was from Football UK, which is a public based source in the United Kingdom. This data obtained was of 11 seasons from 2005 to 2016. This data was targeted because it contained lots of match statistics, more than was available prior to 2005. To determine the rating statistics they used data from a public online database which can be found here: <https://www.fifaindex.com/>.

In this model the home/ away factor is treated as a global characteristic. This means the feature values are computed for both home and away teams for each feature this engineered. Each feature is computed for each season independently with no feature values inherited from the previous seasons. [11] Each feature value is computed at a season level, rather than over the entire database.

They have incorporated ratings for Attack, Midfield, Defence and Overall for each team. The rating statistics are sourced from the previously stated database (https://www.fifaindex.com). This database provides data for season-wide rating statistics for each team, which accounts for part of the overall team's strength across seasons [11]. To determine the ratings, they used the exact algorithm used in the video-game FIFA by EA Sports.

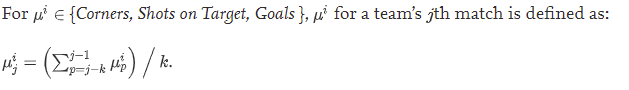
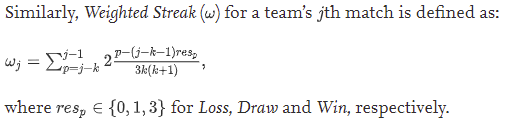
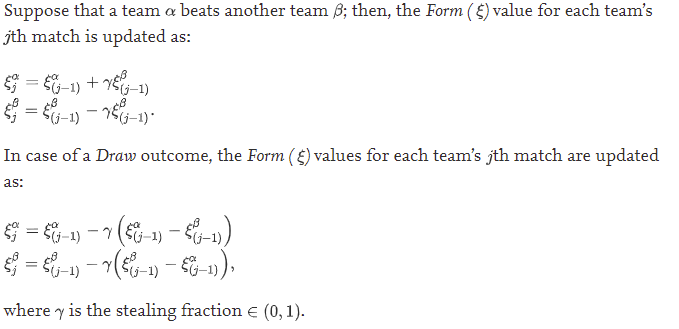
##### 2.5.1. Ratings Used in This Model

A list of the ratings that are determined in this model:

* Attack
* Midfield
* Defence
* Overall
* Home Team Rating
* Away Team Rating

##### 2.5.2. Features generated in This Model

A list of features generated in this model:

* Goal Difference   
    
  *The equation used to determine goal difference for a team’s kth match, where GS = Goals Scored, and GC = Goal Conceded [11]*
* Performance metric (Corners, shots on target and goals)  
    
  *The equation used to determine a team’s performance for the jth match [11]*
* Streak   
  This feature is used to measure how a team is performing, regardless of who the opponents are.  
    
  *The equation used to define recent performance [11]*
* Weighted Streak  
  This feature is similar to the streak feature above, but with time-dependant weights on the scores of the previous games for each team.  
    
  *The equation used to determine Weighted Streak [11]*
* Form  
  This feature is provides insight on a team’s recent performances. The value for form is updated after each match played. Form and streak features are both team factors, the only difference being that form takes into consideration the opponent of the previous matches. There is a higher coefficient if a weaker team beats a stronger team, and vice-versa. A higher value suggests better form [11].  
    
  *A screenshot of how the Form value is calculated [11].*

##### 2.5.3. Features Selected in This Model

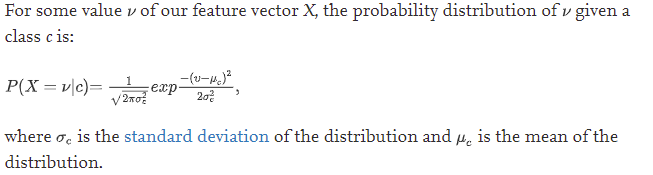
Feature engineering: How did they choose which features to use?

After they had generated various features, they then selected the best performing and most relevant features to reduce the overall amount of features. They are left with 33 features.

The features are separated and put into two groups; Class A contains individual features for Home and Away teams, and Class B contains the differential features. [11]

##### 2.5.4. Machine Learning Models Used to Test Selected Features

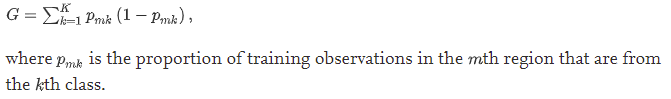
The machine learning models used to test the feature sets are **Gaussian naive Bayes**, **support vector machine**, **gradient boosting** and **random forest**.

The Gaussian naive Bayes model is a classification algorithm in which all features are assumed to be independent. This was used to test both Class A and B, in which it performed better on Class B.  
*Here is a screenshot of the Gaussian naive Bayes algorithm [11].*

This model achieved a performance score of 0.519 mean accuracy.

For the support vector machine model, a supervised machine learning technique called Recursive Feature Elimination (RFE) was used, which was useful in determining the most optimal features to use. It achieved a higher accuracy rating than the Bayes model, however it was unable to model the occurrence of draws which made it unsuitable to use.

The random forest and gradient boosting models perform feature selection during fitting.

*This is a screenshot of the Random Forest algorithm [11].*

The random forest model achieved a test accuracy rating of 0.57.

For gradient boosting, the open-source framework XgBoost was used to implement gradient boosting on the set of features. XgBoost performs gradient boosting with regularized formalization to manage over-fitting issues and therefore makes it perform better.

The gradient boosting model was considered the best performing, with a mean accuracy score of 0.506, however it achieved the highest score for modelling draws which was the least likely event of the match.

##### 2.5.5. Advantages of this model

This study shown that open-source software XgBoost is a great option for performing gradient boosting on a feature set, and can be used to better predict draws than the naive Bayes algorithm. It also shows the random forest model achieves the highest accuracy for home/ away predictions.

##### 2.5.6. Disadvantages of this model

There are lots of missing values in the streak and weighted streak features. Due to this, the value for the initial k game week was dropped from the database for each season to improve performance results. Therefore this system is unable to predict the initial k matches in a season.

##### 2.5.7 Conclusion

Using the open-source framework XgBoost achieved the highest accuracy when predicting draws, whereas the random forest model achieved the highest accuracy for home/ away predictions with an accuracy rating of 0.57.

#### Model 2: Predicting Football Matches Results using Bayesian Networks for English Premier League (EPL)

This model uses Bayesian Networks (BNs) to predict the result of matches, which can be either Home win (H), Away win (A) or Draw (D). They have used data from the English Premier Leauge (EPL) for seasons 10/11, 11/12 and 12/13. They apply the resampling procedure K-fold cross validation to measure the accuracy of the predictions. They achieved a predictive accuracy of 75.09% on average over the three seasons.

##### Data Used in This Model

The data used in this model is sourced from: <http://www.football-data.co.uk>.

The results from this paper suggest the resampling procedure K-fold cross validation is a good option for testing algorithm performance for Bayesian approaches.

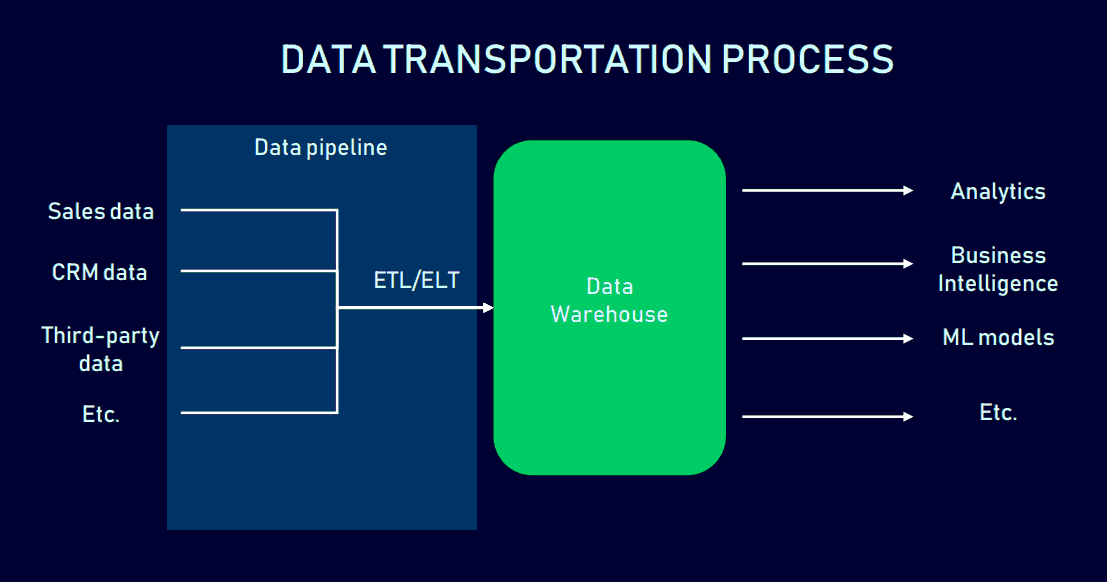
##### Conclusion

Overall, this literature review suggests the use of multiple binary ML classifiers to test the performance of the model, however based on the review of similar models, it implies that the best performing model for predicting Home/ Away results is gradient boosting, specifically the open-source software XgBoost, whereas the best performing model for predicting Draws is random forest.

The biggest issue with current football prediction models is often about choosing the best features to use, the method of evaluating the chosen features, and the type of problem that is defined for the model.

##### 2.5.8 Definition and Components of a Data Pipeline

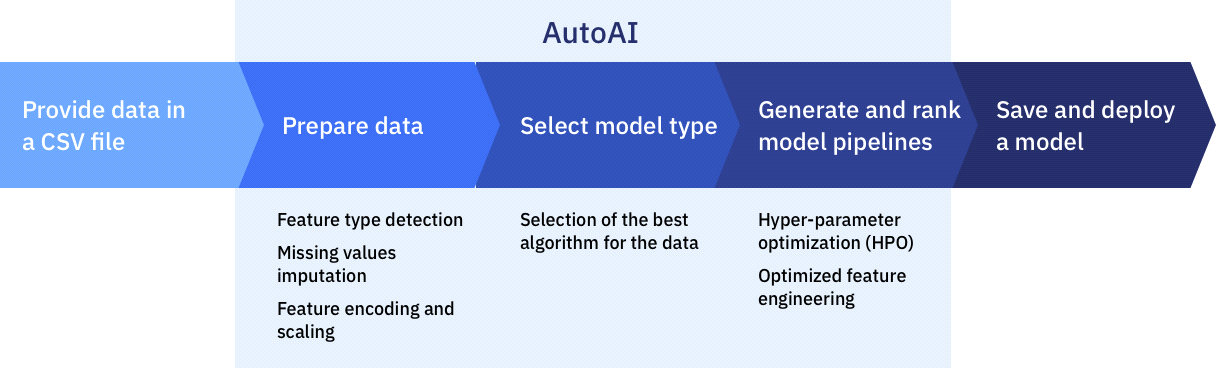
A Data Pipeline (DP) is a combination of tools, techniques and processes used to integrate data and discover new information and patterns. The purpose of a DP is to transfer raw data into an ideal environment for it to be analysed.

  
*[ALTEXSOFT] The process of transporting raw data into a Data Warehouse (DW)*

The data requires separation from its initial storage for a number of reasons, as stated below:

* To save computational power - By using optimal computation power soley for one task, it therefore increasing performance power
* To store and structure the data in a way that makes sense for analysis
* To increase security and privacy of data, restricting who can access certain data
* To reduce the risk of losing data, as it's backed up in its previous storage location [14]

In a DP there are multiple tasks that are usually carried out in a specific order, similar to a waterfall methodology. Once one task is done, we go onto the next.

  
*[15] An illustration showing the process of a data pipeline.*

**Data Storage**

Numeric data is often saved as .CSV files as this allows them to easily be imported to or exported from a program that uses tables to store the data, such as on Microsoft Excel [16]. This data type is great for storing numeric data as the data fields are delimited and separated with a comma. This makes it easier to interact with the data as it presents it in a structured and easier to read manner.

Once we have the data source stored in a csv file, we can then use a program such as Python to feed the data into the pipeline and store it in a database where it can be queried. It is important to transfer the data into an appropriate database that is suitable for querying the data, rather than its initial database, as the initial database is not always suitable.

**SQLite3 Database**

The standard python library features a module called "sqlite3" which is intended to be used when working with an SQLite database. SQLite is a lightweight database that uses C library. The advantage of using this database is that it doesn't need a server to access the database - you can simply access the database using a non-standard variant of SQL [17]. There is also a free-to-use, open source tool called DB Browser that is compatible with SQLite which allows you to make it easier to create, search and edit a database [18].

For the purpose of this project, an SQLite3 database is used to store the data at it is an appropriate database for querying the data from a Python script, and it also supports the DB Browser software to make it easier to browse the database.

**Data Cleaning**

Data cleaning refers to the finding and correcting of errors in the data. Dirty data, or data that has not been cleaned, leads to innacurate data and therefore incorrect decision making [18]. It is for this reason that data cleaning is so important.

By addressing errors in the data, we can in turn make the data more accurate, readable and make better decisions. Errors in the data refer to missing values, random erroneous data, spelling mistakes, different formats, replicated entries and any violations of data integrity [19].

The majority of scientific studies and complex models use and rely upon assumptions to ensure the validity of the results and to avoid undesirable outcomes [20]. According to Jason Osborne, author of Best practices in Data Cleaning, "cleaning data and addressing assumptions can have important benefits on the power, effect size and accuracy of population estimates" [20].

Data cleaning usually consists of two phases;

* Error detection (analysing the database for errors)
* Error repair (updating the database)   
  [DATACLEANING]

We can either use quantitative or qualitative error detection techniques. Quantitative techniques use statistical methods to identify errors, whereas qualitative techniques identifies errors by specifying patterns or constraints and highlighting data that violates them [19].

***Handling missing data, outlier removal***

By knowing the cause of the missing data, we can apply the most appropriate method for analysis [PIGOTT]. There are a number of methods that can be used for handling missing data.

We could use **complete-case analysis**, which only uses cases that are not missing variables. This is appropriate if there is only a few observations, as we can assume the missing data is **MCAR** (Missing Completely At Random) in accordance to Rubin's terminology [21]. The researcher can use standard methods to compute estimates making it easy to implement, however if there are large amounts of missing data, this method is not appropriate as there may not be an adequate amount of data remaining. [21]

Another method we could use is **available-case analysis**, which utilises all available data to estimate the parameters of the model. We can use different cases of data to estimate parameters of interest within the data set. When variables are moderately correlated within regression models this method can be applied to provide consistent estimates. [21] The more correlated the variables, the more inadequate the results become.

If the previous two methods are not appropriate, we could use **Single-Value Imputation**. This method involves the researcher filling the missing value with a more plausible value, such as the mean of all cases that observe the variable. This is referred to as mean imputation. [21] The problem with this method is it will likely lead to variables being underestimated, as the true value is likely not the same as the mean value, and the results will therefore be biased. [21]

To remove outliers from the data, we must inspect the data and use our judgement to determine outliers. We can determine outliers by identifying data that is believed to be erroneous and contributing to the larger range of data. Once the outliers are identified we can set their value to 0. [22]

**Feature Engineering**

Feature Engineering (FE) refers to the process of using domain knowledge to extract features from raw data in order to improve model accuracy. The workflow is an iterative process in which we create new features from old features to improve model accuracy and performance. FE plays a key role in successfully predicting important process information when developing a data-driven machine learning model. [23] These features are then used to improve the machine learning algorithms to better model performance.

The process of Feature Engineering is as follows:

* **Brainstorm ideas for features** – Research other projects and discuss feature ideas to base the model on.
* **Create features** – You can automatically extract features, manually construct features or use a combination of both depending on your needs and constraints.
* **Select features** – List and score the features and determine the best features to base the model on.
* **Evaluate model performance** – Using the chosen features, estimate the accuracy of the model. This can be done on unseen data to test its performance.  
  [24]

**Feature Selection**

Feature selection is focused on removing non-informative and unneeded predictors from the model. Non-informative variables will reduce the accuracy of the model and make the predictions unreliable. [24]

There are two main methods for feature selection; **supervised** and **unsupervised**. To help decide which is the most appropriate method, we must determine whether the features are selected based on the target variable, or not. If they are then it is a supervised method, if not then it is unsupervised. [24] Examples of supervised selection methods can be split into wrapper and filter methods.

Alternatively, we can use **machine learning algorithms** to select features. These algorithms can perform automatically during the learning process of the model, and are referred to as intrinsic feature selection methods. Examples of these algorithms include penalized regression models like decision-trees and random forest. [24]

***What makes a good feature?***

**Repeated Discrete Values**

A good feature value will appear multiple times within the data set. This allows the model to learn exactly how this specific feature value relates to the label. By having multiple examples it enables the model to see the feature in more than one setting and improves its judgement for when it’s a good predictor for the label. [25]

**Easily distinguishable**

All features should be clearly defined with labels that make sense of the value. If the value can't be made sense of then it can lead to unclear values. Noisy data can be another cause for unclear values. The value should always be in an appropriate format in correlation to its label name. [25]

**A combination of available data**

Instead of focusing on available data for a feature, the requirements of the model should be the main priority. Often the variable should be a combination of available data to produce a variable that will make a good predictor. [26]

***State-of-the-art feature engineering techniques***

**Pandas library**  
As the model is created using Python, the **Pandas** library is suitable for data manipulation and visualization.

**SciPy Library**

This library provides an implementation of useful statistical measures, such as:

* **SelectKBest** - This selects the top K variables
* **SelectPercentile** - This selects the top perciential variables  
  [Brownlee (used in Feature Engineering)]

These techniques can be used during feature selection to help select the best features.

**Binning**

This is a technique applied to prevent over fitting and make the model more robust. It involves grouping continuous values into smaller groups referred to as bins. By grouping the continuous data into smaller groups, we can minimise the number of categories which can be useful for data visualisation. The disadvantage of this technique is its negative impact on model performance, as information is lost and the data becomes regularized [27].

**Filter methods**

Filter selection methods work by using statistical techniques that evaluate the relationship between all input variables and the target variable [24]. The scores obtained are used to filter the input variables to choose the variables most appropriate for the model.

**Wrapper methods**

Wrapper selection methods work by creating a number of models using different subsets of input features. The features that create the best performing model possible are chosen in accord to a performance metric. [24]

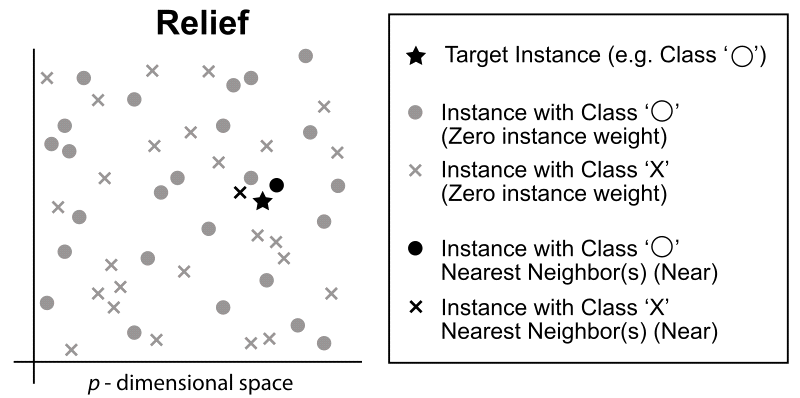
**Embedded methods**

This method works by learning which features contribute the most to the model accuracy whilst the model is beingcreated. An example of an embedded feature selection method is regularization, also known as a penalized method. This method works by introducing added constraints to the algorithm that make the model less complex. Examples of penalized methods include lasso and ridge regression. [28]

**Recursive Feature Elimination (RFE)**

The RFE method can be used to select the best features. This method recursively removing attributes and building the model on the remaining attributes. By using the model accuracy, it is able to identify which particular attributes (or combination of attributes) contribute the highest when predicting the target attribute. [28]

**RReliefF algorithm**The original relief algorithm was designed by Kira and Rendall in 1992 and is used for feature selection. The RReliefF algorithm is an extension inspired by the relief algorithm. It is known as a relief-based algorithm (RBA) and has been adapted to address problems with the original algorithm, such as performing more reliably in noisy environments and generalizing to solve multi-class and regression problems. [29]

*An illustration of Relief neighbor selection for scoring. [30].*

In this algorithm, the parameters are set to 100 points which are evaluated with 10 neighbors with exponential rank correction [Bocca]. The algorithm is iterated 10 times before the results are clustered. Any features that have a negative importance are dropped and the remaining features are scaled to provide a total importance value. The total importance is regarded as the percent of the change, and features are then added until the value reaches 90% [31].

## Chapter 3: Methodology

### 3.1. Preparation

Before beginning the project, the analyst should first ensure the relevant software is installed. The following software is required;

* PyCharm Community Edition 2020.2.3
* SQLite3
* DB Browser (used for SQLite3)
* Github Desktop

### 3.2. Initial Analysis

The initial analysis required is to determine what type of data is required, what data type the data should be saved as, what makes a good feature, and the appropriate machine learning models to use. This is all covered in the Literature Review section of the report.

### 3.3. Data Collection & Development Planning

To begin, research of an appropriate data source to use is conducted. The data source that is found to be most suitable is from a reliable and trusted website called <https://www.football-data.co.uk/englandm.php> which offers free downloads of .csv files containing the latest football data from the Premier League.

The next step is to store the data in a suitable database. The most appropriate choice is *SQLite3* database which can be queried from Python and explored using the free software DB Browser.

Now that we have the data in an appropriate environment for querying, we must first clean the data to make sure it is all the same file type and there are no errors or outliers, as this can cause errors and inaccuracy within the model.

### 3.4. Data Analysis and Results

This stage involves choosing features to base the model on and then extracting or manually constructing the features.

Once the features have been determined, we must test the features and score them, depending on the method used for selection. Once we have tested and scored all the features, we can decide which features are most important and remove the less important features.

The charting module Plotly will be used to create visualisations of the data. This library of functions and tools allows the user to generate line charts, bar charts, histograms, scatterplots and more, which can be used to find any insightful patterns in the data and help us determine which are the most influential features to use. Plotly is a free, open source software library available on Python.

Another library to use for visualisation of the data is NumPy for Python. This is a free, open source library similar to Plotly but is effective for providing mathematical and statistical functions for use on matrices and arrays.

Once we have selected the best possible features, we must run the model using various classification algorithms to test its accuracy rating. The results will show us which classifier performs best on the model.

### 3.5. Evaluation

Now that we have obtained accuracy ratings for the predictions, we can assess the model’s accuracy by comparing the accuracy rating to a baseline.

### 3.6. Reporting Results and Conclusions

After running the code, the baseline and the performance scores will be generated in the console, displaying the scores as a value between 0 and 1. For example, a performance rating of 0.3 would be regarded as a 33% success rate at predicting future matches. The results will then be documented in the conclusion section.

### 3.7. Project Planning and Timescales

The steps of a data pipeline are often sequential, with one stage taking place only after the previous stage is complete. Therefore a waterfall methodology is followed. This allows for a continuous, steady flow of progress leading up to the project deadline.

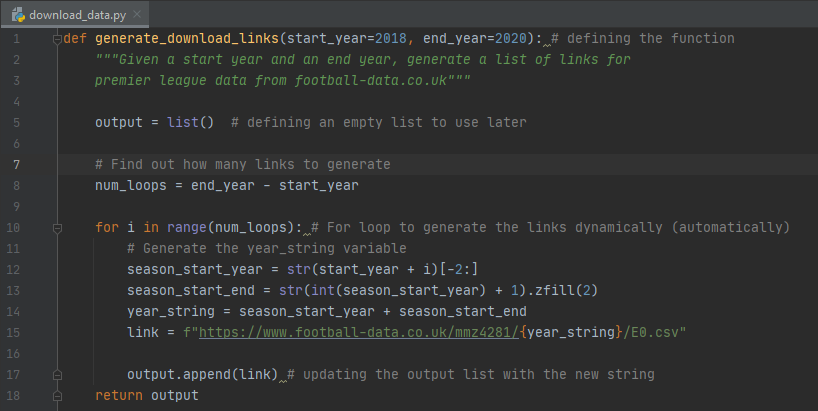
I used Microsoft Project to plan the project and produce timescales that can be followed to ensure I am on track and in control of my progress.

## Chapter 4: Implementation

### 4.1. Generate the Download Links of Source Data

**Download\_data.py script**

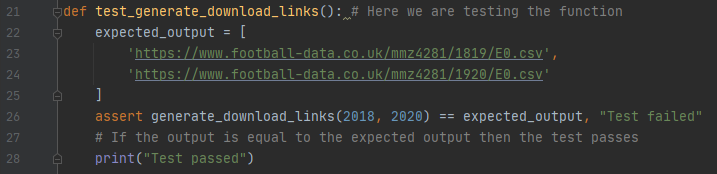
First of all, I created a new python script and named it “generate\_download\_data”.   
In this script, I defined a new function called “generate\_download\_links”, which will automatically generate the relevant download links for the source data given a start year and end year.



As you can see, I have chosen to generate download links for all seasons between 2018 – 20.   
Next I define an empty list named “output” which is used to store the results.  
I also define the variable “num\_loops” which subtracts the start year (2018) from the end year (2020) and gives us 2. This value is used to determine the range of the for loop, and ultimately how many links to generate in the output list.  
Next a year\_string variable needs to be defined so it can be inserted into the URL of the link. To do this, I created two other variables to be used – season\_start\_year & season\_start\_end.  
Season\_start\_year takes the last two digits of the start\_year (18) and stores it as a string.  
Season\_start\_end takes the season start\_year (18) and adds 1.  
The zfill function is used to add two zero’s to the beginning of the variable. This is used as during testing I found an error when retrieving seasons prior to 2010 (due to their being a single digit).

Testing the Generate Download Links function

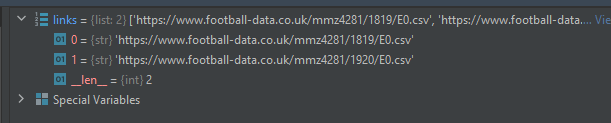
The code below is used to test the “generate\_download\_data” function.



I have defined the expected output and used assert to check whether the expected output is equal to the actual output. If true, then it will print “Test passed”. Else it will print “Test Failed”.

Now I will run the following code to test the function:

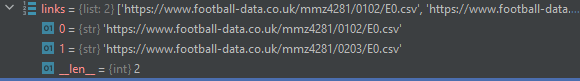




As you can see, the test passed as it provided the expected links as stated above.

Now we will test whether it works with one digit years prior to 2010.

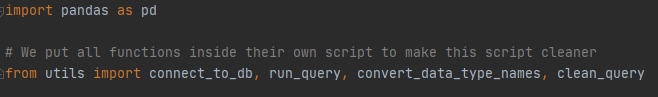




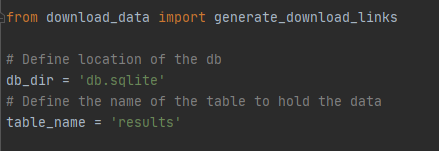
As you can see, the test passed and the function works as expected.

### 4.2. Data Cleaning

main.py script  
Now that we can generate the links to the source data, I need to clean the data and then load the data and connect to the database from the main.py script. Firstly, I have created some useful functions in a separate script names utils, including connect\_to\_db (connect to database), run\_query, convert\_data\_type\_names and clean\_query. This means I don’t have to create the functions in the main script (as this can make the script messy), and instead I can simply call them from the main script when required by importing them. I will also be using Pandas’ data analysis library as this contains useful functions for data analysis.

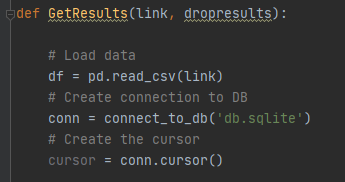


Next, I have imported the data from the download links into a table called ‘results’ in the sqlite3 database. I now need to define a connection to the db location and then define the name of the table to hold the data. This is done with the following code:

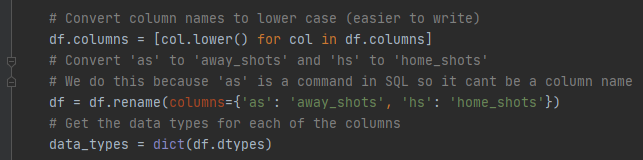


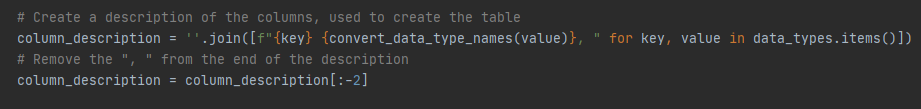
From here, I can now begin cleaning the imported data.

To begin, I first define a function called “GetResults”. I then tell Python to load the data and create a connection to the database using a predefined function called ‘connect\_to\_db’, which does what it says. The next step is to create the cursor which I can use to run queries.

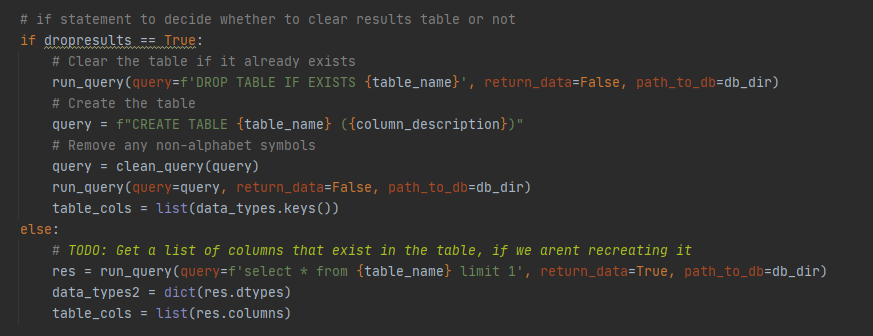


The next step in the “GetResults” function is to begin the data cleaning process. First, I converted all the column names to lower case (so the names are easier to write to) using a list comprehension. I then renamed some of the original columns to something similar as some of the predefined column names that were with the original data were commands in SQL which could cause problems.

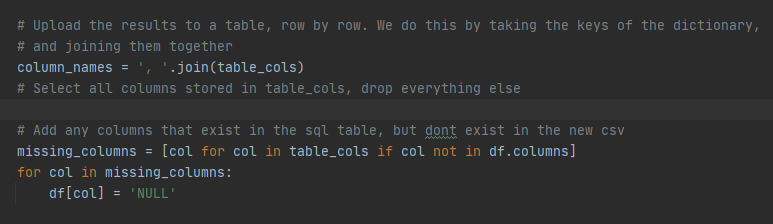


Next  


The code then enters an if statement to determine whether it should clear the results table or not. This is shown below:



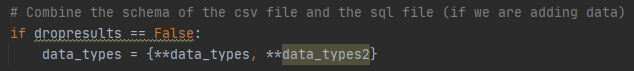
If the table already exists, then it will clear it so the results aren’t duplicated, and then creates a new empty table in its place and runs a few of the predefined data cleaning functions from the utils script to quickly clean the data and remove any non-alphabetic symbols. Else, it will retrieve the columns that exist in the table and stores the data types in a dictionary and table columns in a list.



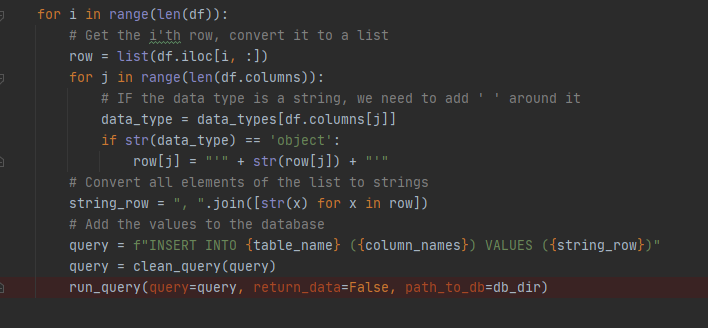
We can then upload the results to a table row by row by taking the keys of the dictionary and then joining them together. So we are selecting all columns stored in table\_cols and dropping everything else. From here, we can add the missing columns that exist in the sql table but don’t exist in the new csv file, so if the column names in later seasons are different, the missing columns will contain a “NULL” value.



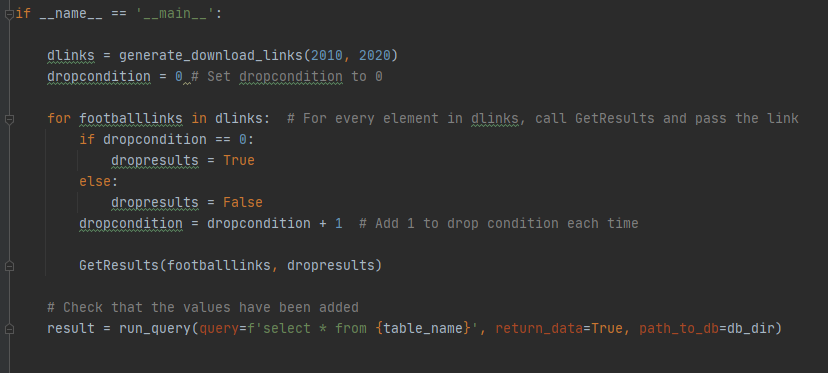
I then tell it to remove any columns that exist in the new csv, but not in the sql table.



The schema of the csv file is then combined with the sql file (only if we are adding data).



The final step in the “GetResults” function is a for loop, which retrieves the i’th row and converts it to a list. If the data type is a string, then I tell it to add ‘ ‘ around it, so Python can identify it as a string. I then convert all elements of the list to string, and add the values to the database before running the predefined “clean\_query” function to clean the values of the query and then running it.

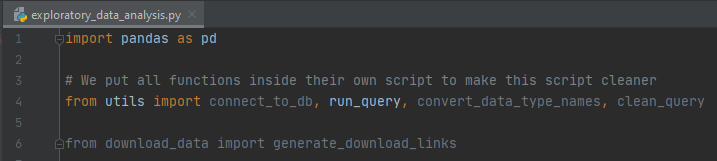


At the bottom of the main.py script, under the “GetResults” function, I have created an if statement containing a nested for loop that calls the GetResults function and passes the link for every element in ‘dlinks’. By specifify the years in the arguments for “generate\_download\_links”, I tell the program to retrieve all data for seasons between the start year and end year. For example, I have added “2010” as the first argument, and “2020” as the second argument, so it will generate the download links for all seasons between 2010 and 2020.

### 4.3. Exploratory Data Analysis (EDA)

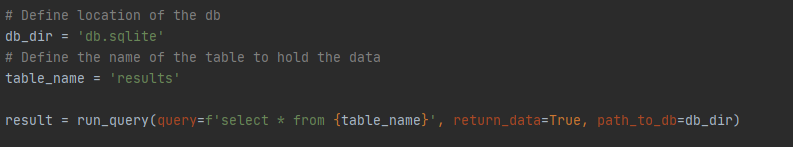
exploratory\_data\_analysis.py

Now that I have the data in an appropriate format, I can begin exploring the dataset to better understand the data I’m working with, find trends and patterns, and to think of new information that I can generate that would be useful to use as a feature.



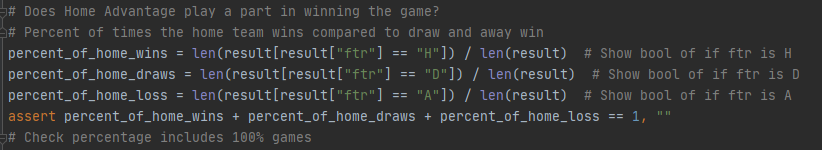
First of all, I imported Pandas library which includes lots of useful data analysis tools. I then imported the relevant functions (which were created in a separate script called utils) that I am planning on using. This includes connect to database, run query, convert data type names and clean query.

The next step was to define the location of the dataset, and the name of the table which holds the data. I then ran the run\_query function I created in the previous script to store all the relevant data in a “result” data frame. This will allow me to view all the columns and rows of the data frame and inspect the values.



Now I have all the data available in a data frame, I can begin analysing the data set.

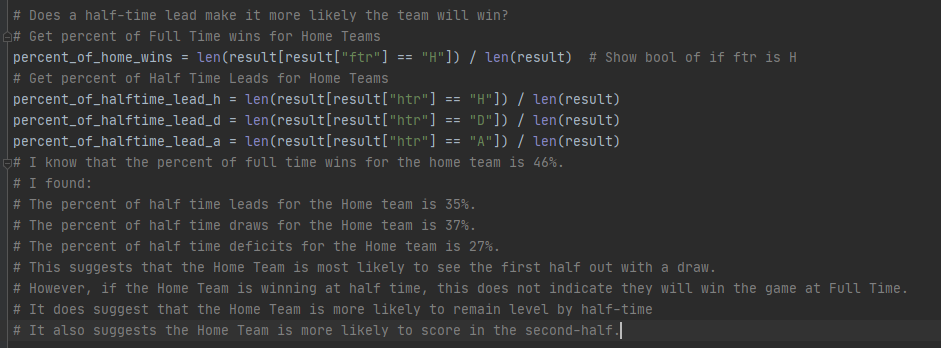
Percent of Home Team Wins, Draws and Losses  
The first question I wondered was whether Home Advantage played a part in winning the game. Would this be a good feature to select?



To find this information, I needed to first create a query that calculates the percentage of home wins, home draws and home losses. I then implemented an assert to check that the queries are using 100% of the data, just to make sure it’s not missing any data as that could affect the results.

From this, **I found that when a team plays at home, the team wins 46% of the time, draws 21% of the time, and loses 32% of the time.**

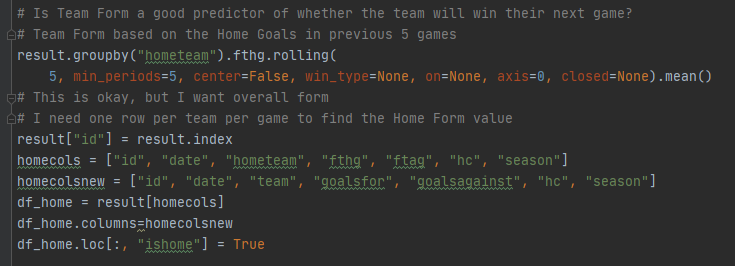
Percent of half-time leads compared to full-time results

The second question I was curious of is whether a half-time lead for the Home team can tell us if the Home team is more likely to win the match? If so, would this be a good feature?

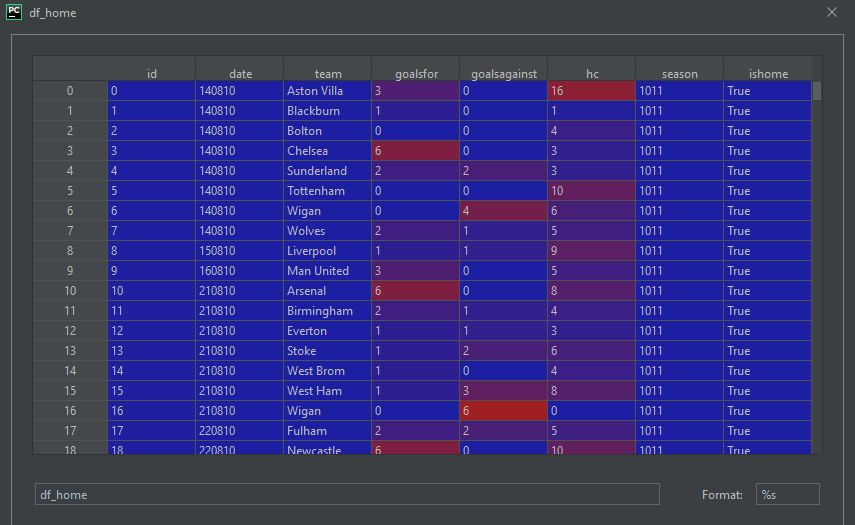
To find this information, I created a query to show a bool result of it the ftr is home. The first query I asked to find the percent of home wins, which gave me a percentage of 46%.

Next I wanted to find the percent of half-time leads for the Home teams. I used the queries shown above to calculate the percent of Home team wins, draws and losses at half-time. From this, I found the percent of half time leads for the Home team is 35%, the percent of half-time draws for the Home teamis 37%, and the percent of half-time deficits for the Home team is 27%.

This suggests the Home team is more likely to win the game if they stay defensively solid and don’t concede in the first half. However, if the Home team is winning at half-time, this does not suggest they are more likely to win the game. The Home team is more likely to be level with the Away team at half-time. This information also suggests the Home team is more likely to score in the second-half rather than the first-half.

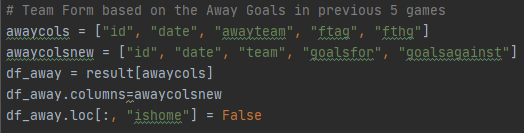
Home Team Form (based on previous 5 home games)  
Next I wanted to see whether Team Form based on Home Goals in previous 5 games was a good predictor. To do this, I used the rolling function to only include the last 5 games in the results, but not the most recent game, and grouped the result data frame by "home team" to only display results for home games.   
This gave me a data frame showing goals scored and conceded in the last 5 Home games per team (excluding the most recent game). This was okay but ideally I needed one row per team per game in order to find the Home Form value.

To do this, I first create a new column in the result data set called “id” which can be used to reference certain games later on, and assigned the ID value to the row number – this will be used to sort the results onto separate, individual rows later on. I then created a variable called “homecols” that stores the relevant data under the columns “id”, “date”, “hometeam”, “fthg”, “ftag”, “hc”, and “season” (which will be used later during the visualisation stage). I then create a second variable called “homecolsnew” that lists the columns I want to use in a new data frame called “df\_home”. The “df\_home” dataframe displays only the results under the columns specified in the “homecolsnew” variable. To make sure it only includes data for the home team, I added the last line of code which ensures the data frame “df\_home” only displays results where the “ishome” column is equal to ‘True’.

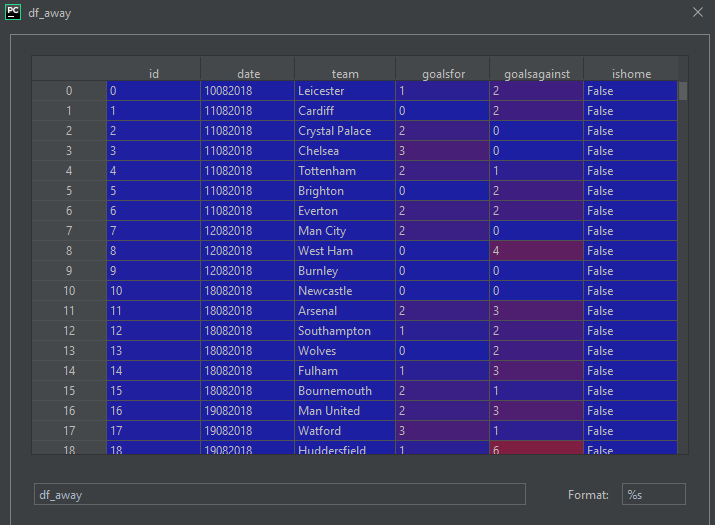


This is the “df\_home” data frame, displaying results for the home team of each game, under the columns specified on the previous page.

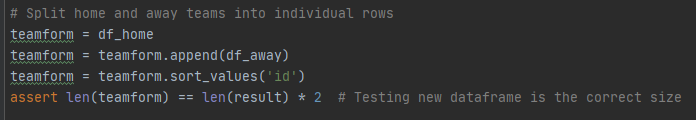
Away Team Form (based on previous 5 away games)  
I then created another Data Frame to display Team Form based on the Away Goals in the previous 5 games. I used the same query but changed "home" to "away", and made sure the team was away using a "ishome" = False boolean value.



This gave me a data frame called “df\_away” showing goals scored and conceded in the last 5 matches for the Away team per team (excluding the most recent match).



This is the “df\_away” data frame, displaying the data for the away team for each match including id, date of match, team name, goals scored, goals conceded, and a column called “ishome” which confirms whether the team is away or not.

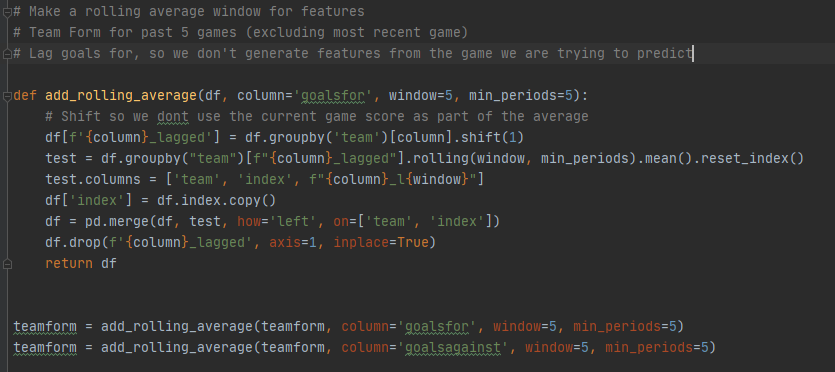


Here I have merged both home and away teams into a newly created data frame called “teamform”, which will be used to calculate and display the form of the home and away team and measuring a “attackingstrength” and “defensivestrength” for each team based on the previous 5 games to determine their current form. The second to last line sorts the results by the row id, which splits the home and away teams into individual rows. The last line is added to check the new data frame is the correct size (it should be x2 the size of team form, as it consists of two seperate data sets for both teams for each match.

Rolling Average Function

I now have information such as the percent of home team wins, draws and losses, home team form and away team form.

Next, I want to rank each team’s attacking and defending strength based on the average goals for and against in the last 5 games to determine each team’s attacking and defensive strength. Before I do this, I need to apply a “rolling average” window for the features. I want to determine each team’s form based on the previous 5 games – excluding the most recent game. I will do this by lagging “goals for” so we don’t generate features from the game we are trying to predict.



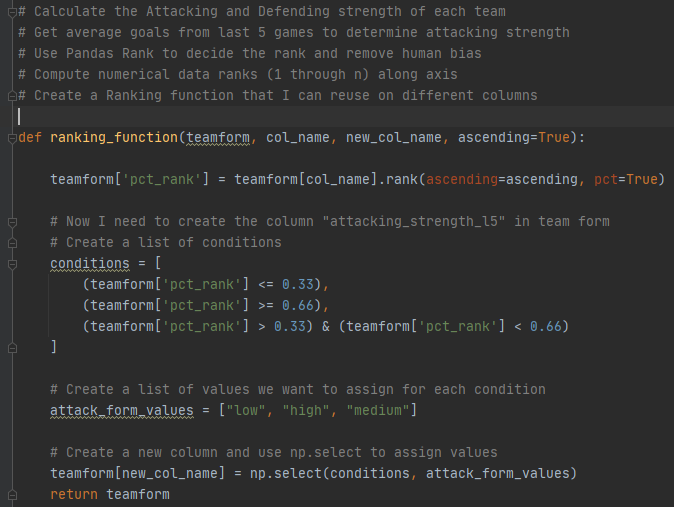
I first define the rolling average function for goals for in a window of 5, with minimum periods being set to 5. I then use the shift command and give it an argument of 1 so it doesn’t include the current game score as part of the average.

The rolling average function is applied to the teamform data frame for “goals for” and “goals against” to get the last 5 goals for and against which will then be used in Pandas’ ranking function to determine their form based on the results.

Pandas’ Ranking Function  
Attacking strength

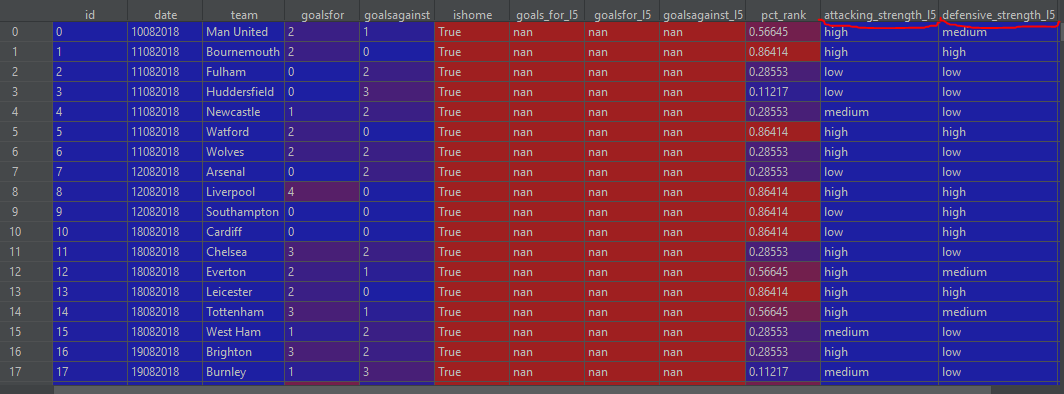
To calculate the attacking strength of each team, I used Pandas ranking function to determine the attacking strength of each team based on the average goals for each team in the last 5 games.

I chose to use Pandas’ ranking function to decide the rank in order to remove any potential human bias.



Here is the code I used to implement Pandas rank function to determine each team’s attacking and defending form. I applied the function to the teamform data frame which uses the rolling average function to record goals scored and conceded in the previous 5 games. This function determines the rank of each team based on the conditions I defined. I created a list of values to assign to each condition. If the average goals scored in the last 5 games is equal to or below 0.33, then it will be determined as “low” attacking strength, if its equal to or greater than 0.66, then it is “high”, else if it is inbetween these values, then it is “medium”. I added a new column to teamform and used numpy’s select to select the conditions and attack form values.

I then created two new columns called “attacking\_strength\_l5” and “defensive\_strength\_l5” to the teamform data frame, which will store and show the rankings for each team for each match.

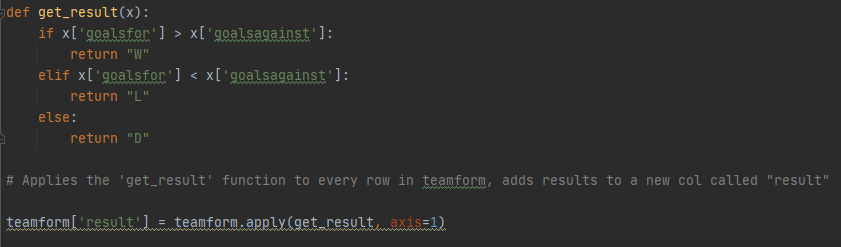


Here is the Data Frame for team form including the new attacking and defensive strength form for each team calculated based on their current form.

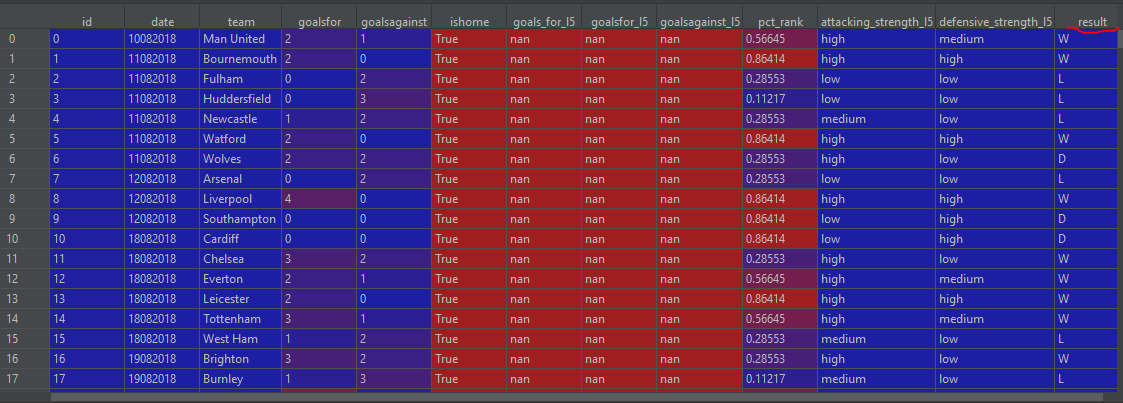
Get Match Result

Another important piece of information which is not already included in the teamform data frame is whether the team won or not, without looking at the scores.

To calculate this, I added the following code to the script:



This added a column named “result” to the team form Data Frame and calculates whether the respective team won, lost or drawn the match.



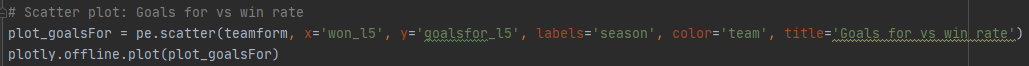
Here is the updated team form Data Frame including the new “result” column.

### 4.4. Visualisations

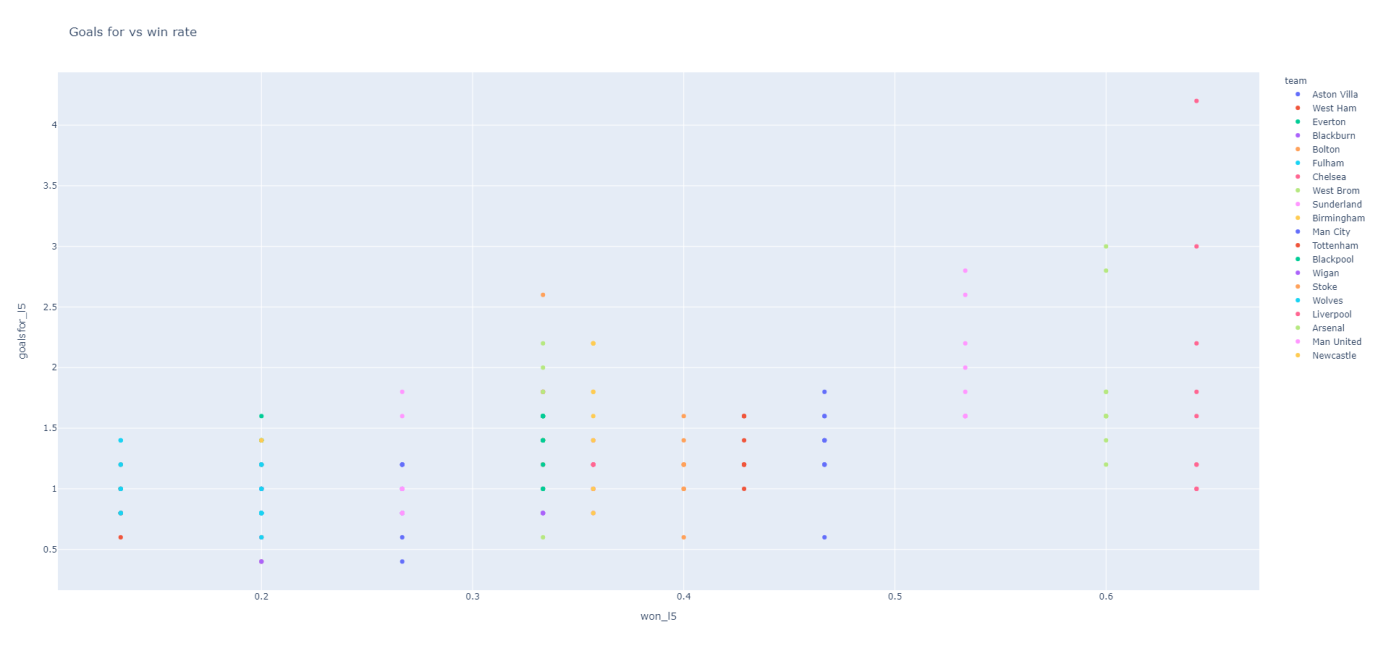
The purpose of this section is to visualise the dataframes and identify the most influential factors towards winning a game. Each potential factor will be measured against the win rate and plotted in scatter plots. If the resulting scatterplot shows a positive correlation between the factor and the win rate, then that shows it is influential towards the win rate and would make a good feature.

#### 4.4.1 Goals scored vs Win Rate

The screenshot below displays the code I used to generate the following scatter plot:



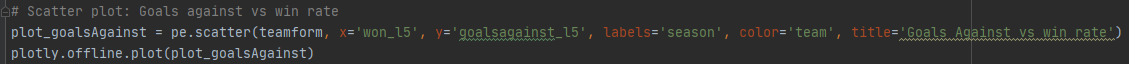
The screenshot below displays the scatter plot generated by the above code:



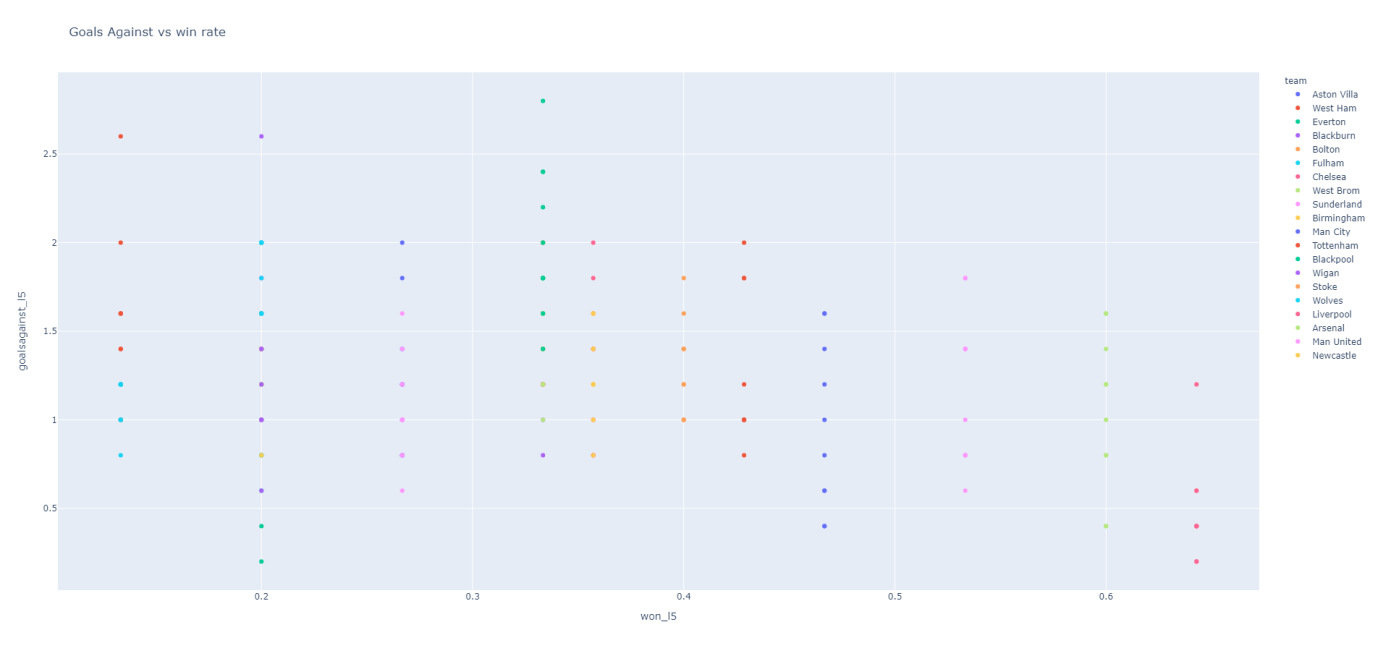
This scatter plot clearly shows a positive correlation betwen goals scored and win rate, indicating the more goals a team scores, the higher their chance of winning – which is what we would expect. Therefore Goals For would make a good feature.

#### 4.4.2 Goals Conceded vs Win Rate

The screenshot below displays the code I used to generate the following scatter plot:



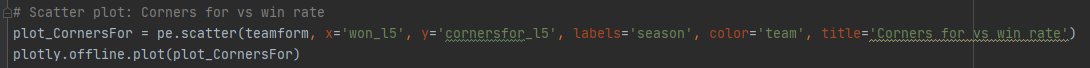
The screenshot below displays the scatter plot generated by the above code:



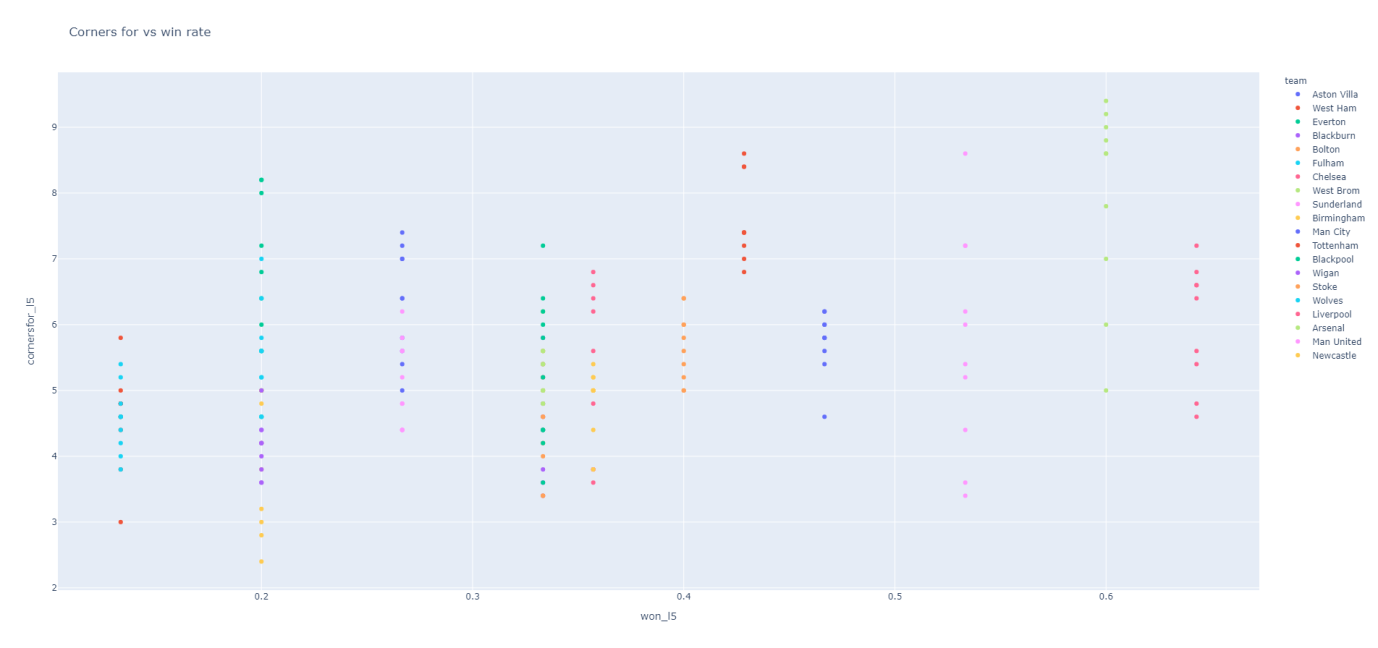
This scatter plot represents a negative correlation between goals conceded and win rate. This indicates that more goals conceded has a negative relationship with the win rate. This would not make a good feature.

#### 4.4.3 Corners For vs Win Rate

The screenshot below displays the code I used to generate the following scatter plot:



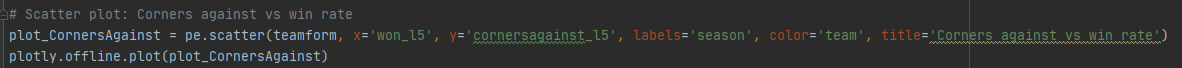
The screenshot below displays the scatter plot generated by the above code:



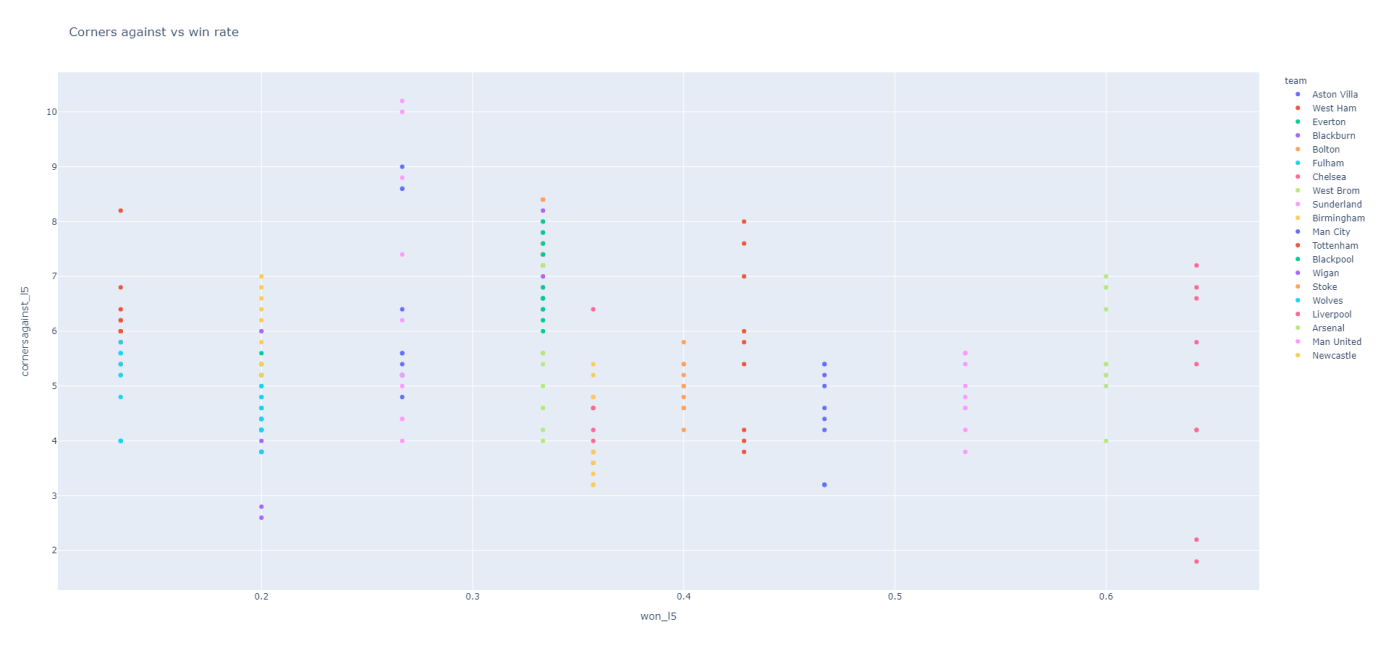
This scatter plot shows a positive correlation between corners for and the win rate, and therefore Corners For would make a good feature.

#### 4.4.4 Corners Against vs Win Rate

The screenshot below displays the code that is used to generate the following scatter plot:

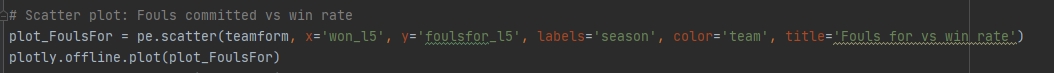


The screenshot below displays the scatter plot generated by the code above:

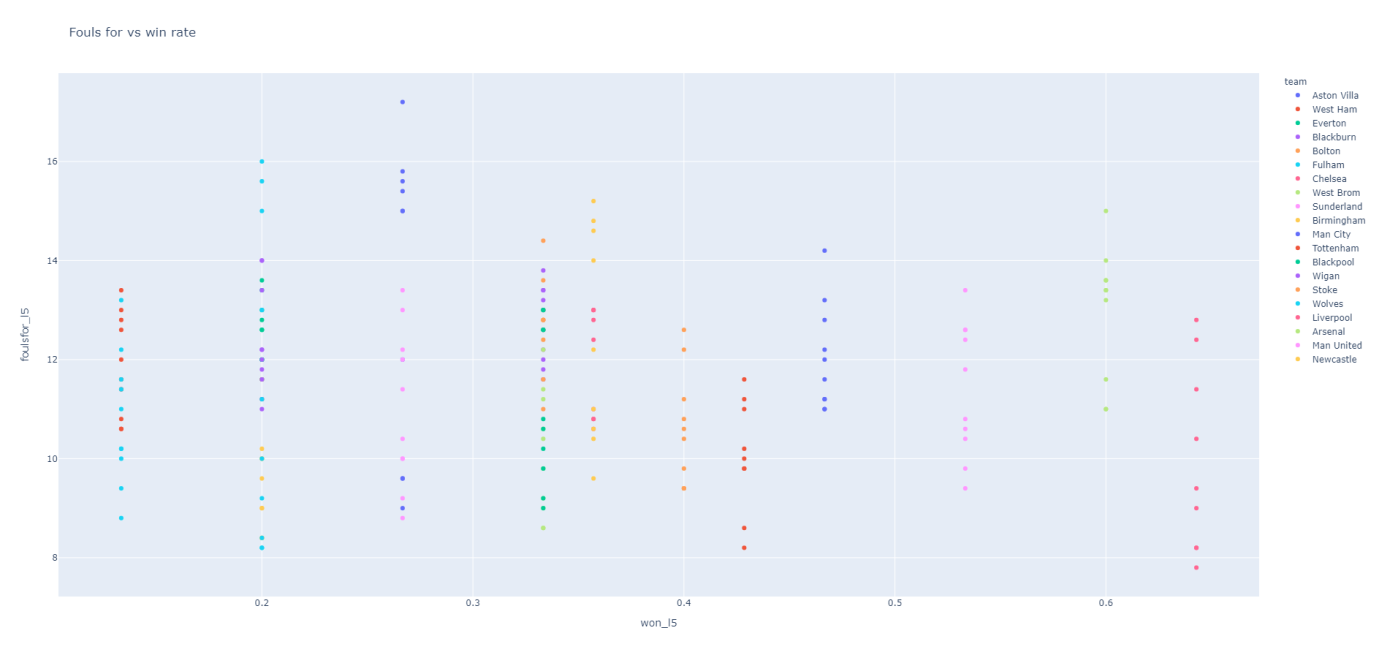
This scatter plot represents a negative correlation between corners conceded and win rate, and therefore shows Corners Against would not make a good feature.

#### 4.4.5 Fouls Committed vs Win Rate

The screenshot below displays the code that is used to generate the following scatter plot:



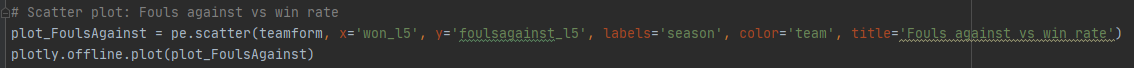
The screenshot below displays the scatter plot generated by the code above:



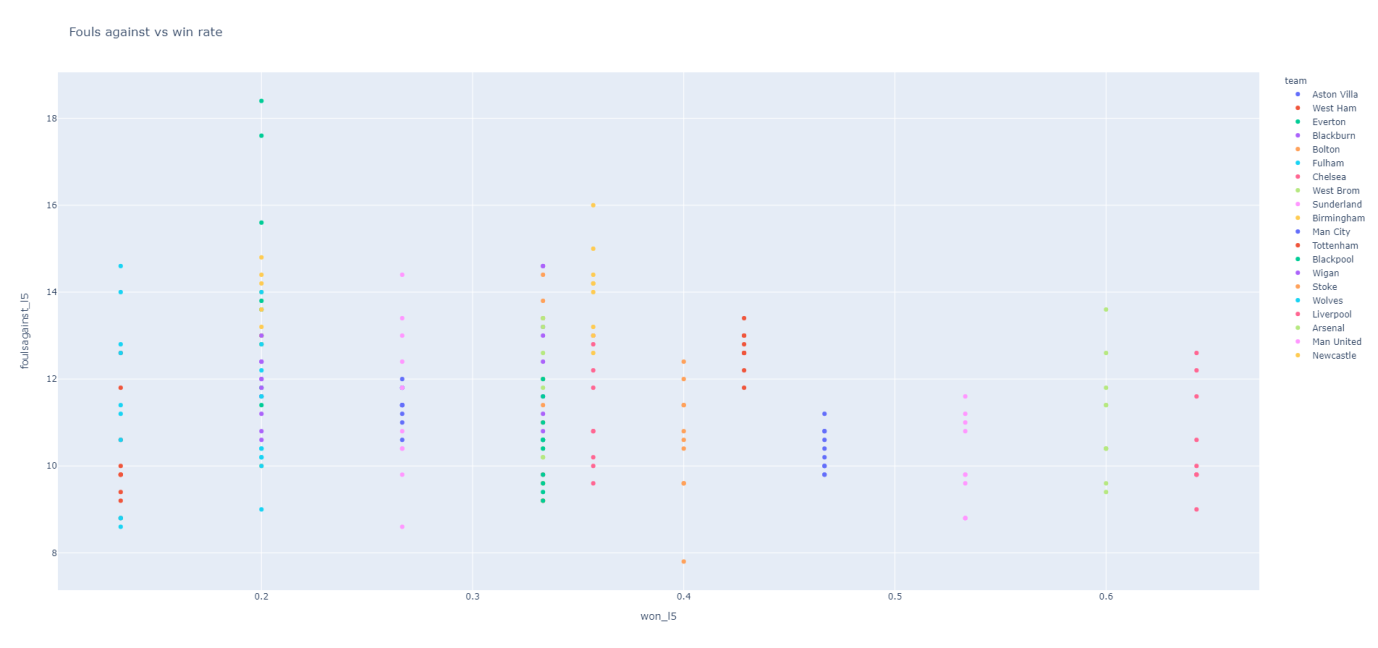
This scatter plot represents a negative correlation between fouls commited and win rate, and therefore shows Fouls Committed is would not make a good feature.

#### 4.4.6 Fouls Against vs Win Rate

The screenshot below displays the code that is used to generate the following scatter plot:



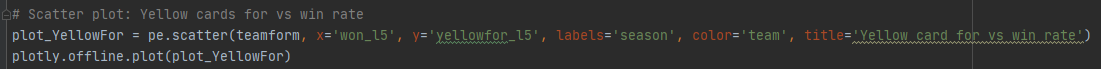
The screenshot below displays the scatter plot generated by the code above:



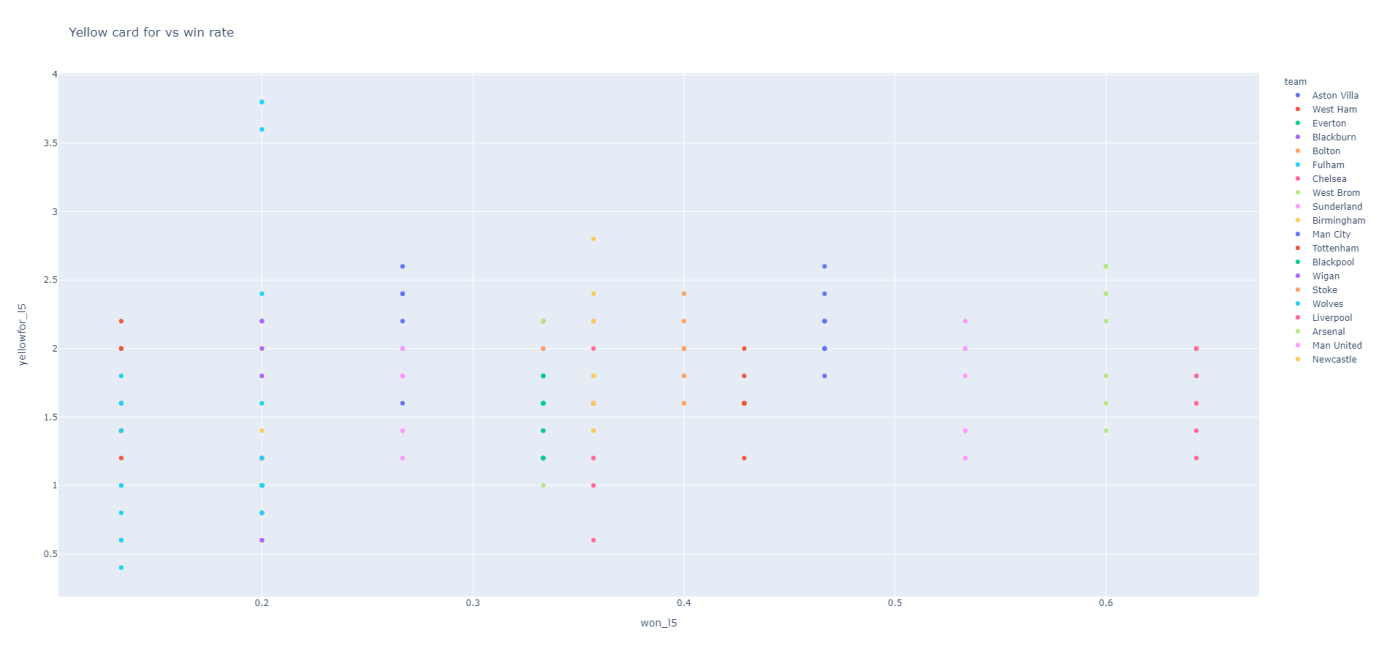
This scatter plots shows no clear correlation between fouls against and win rate, and therefore Fouls Against would not be a good feature.

#### 4.4.7 Yellow Cards vs Win Rate

The screenshot below displays the code that is used to generate the following scatter plot:



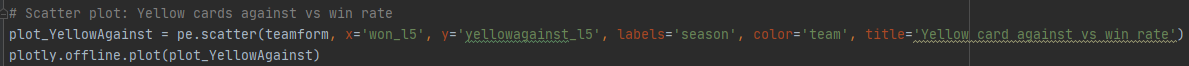
The screenshot below displays the scatter plot generated by the code above:



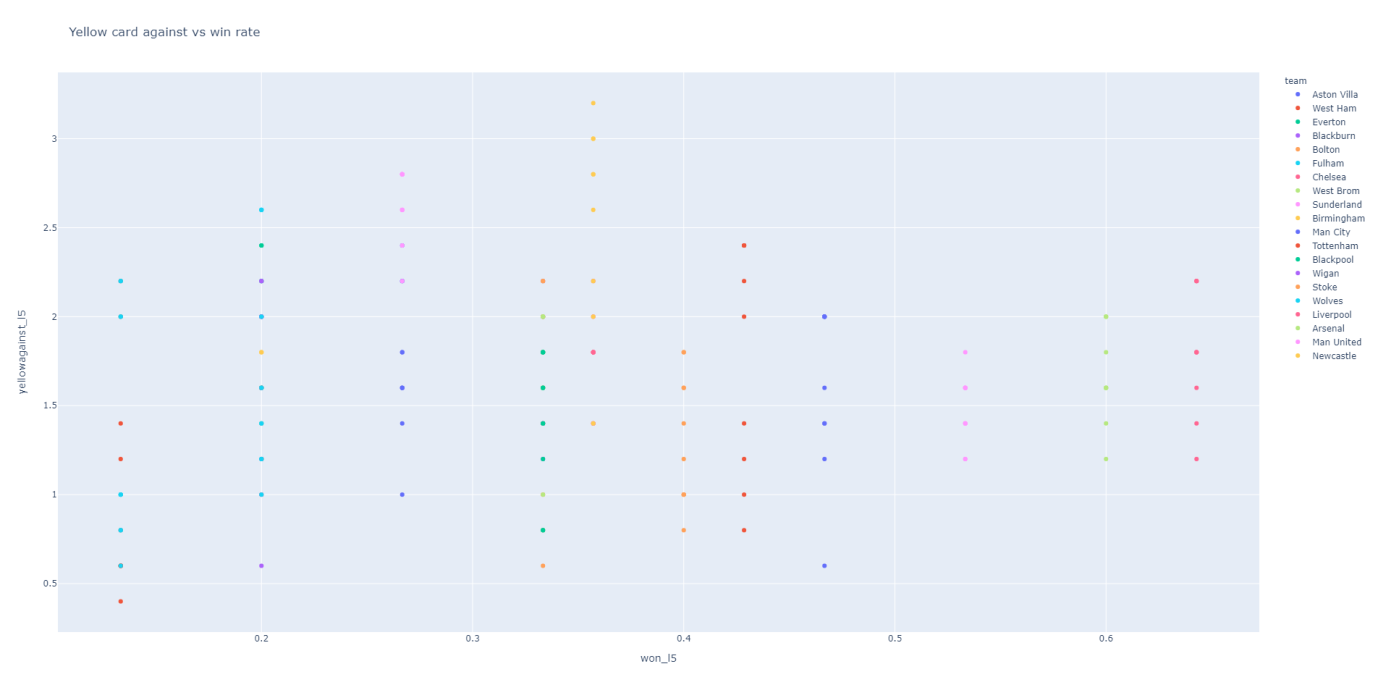
This scatter plot shows no clear correlation between the number of yellow cards a teams picks up and whether they win the game or not, and therefor would not make a good feature.

#### 4.4.8 Yellow Cards Against vs Win Rate

The screenshot below displays the code that is used to generate the following scatter plot:

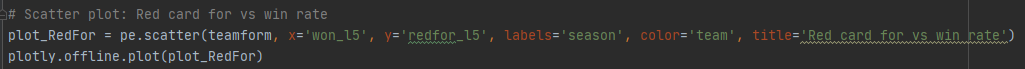


The screenshot below displays the scatter plot generated by the code above:

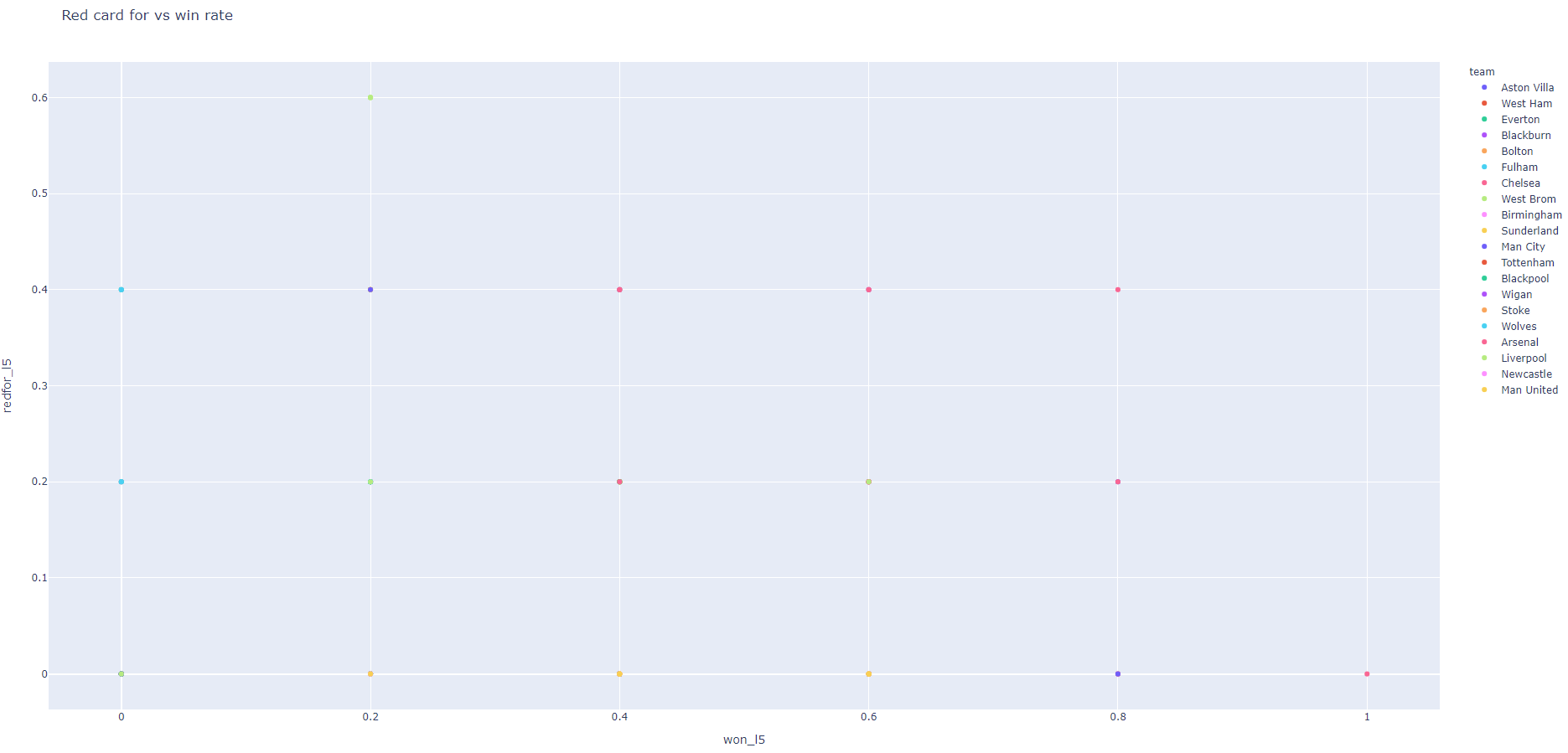
 This scatter plot represents no clear correlation between the number of yellow cards a teams picks up and whether they win the game or not, and therefore would not make a good feature.

#### 4.4.9 Red Card For vs Win Rate

The screenshot below displays the code that is used to generate the following scatter plot:



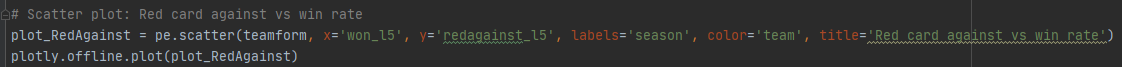
The screenshot below displays the scatter plot generated by the code above:



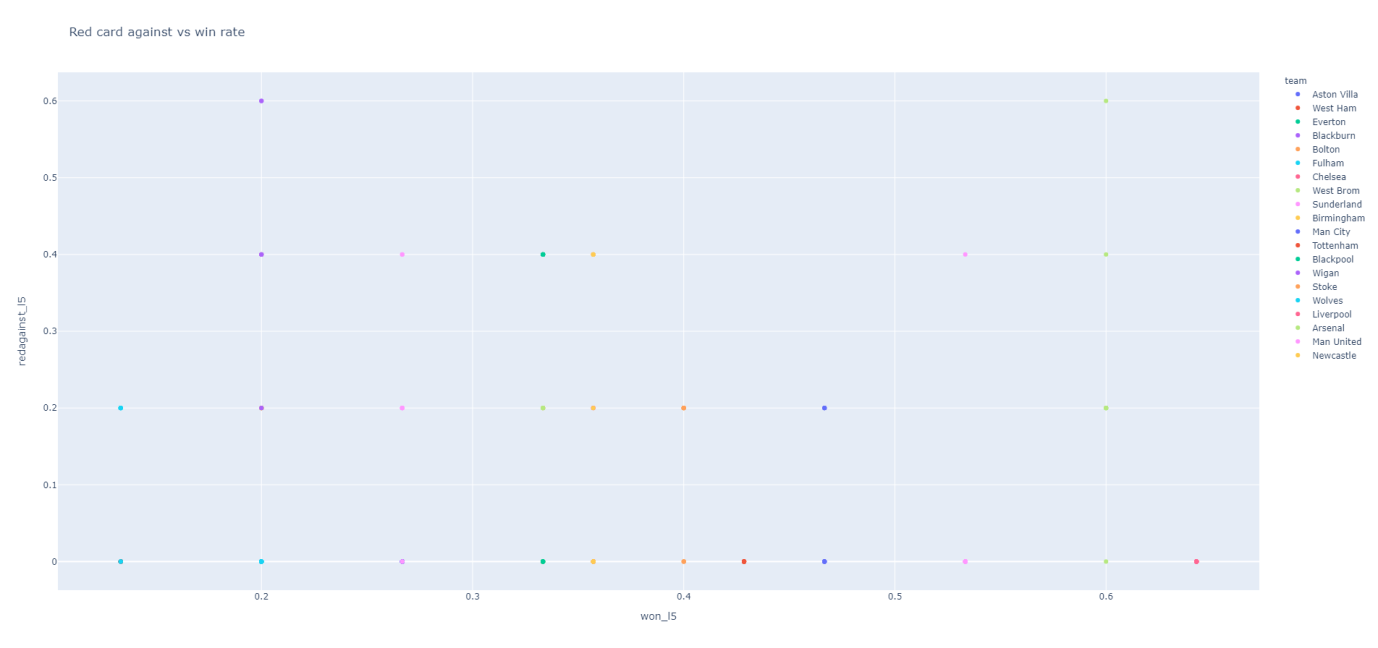
This scatter plot represents no clear correlation between the number of red cards a teams picks up and whether they win the game or not, and therefore would not make a good feature.

#### 4.4.10 Red Cards Against vs Win Rate

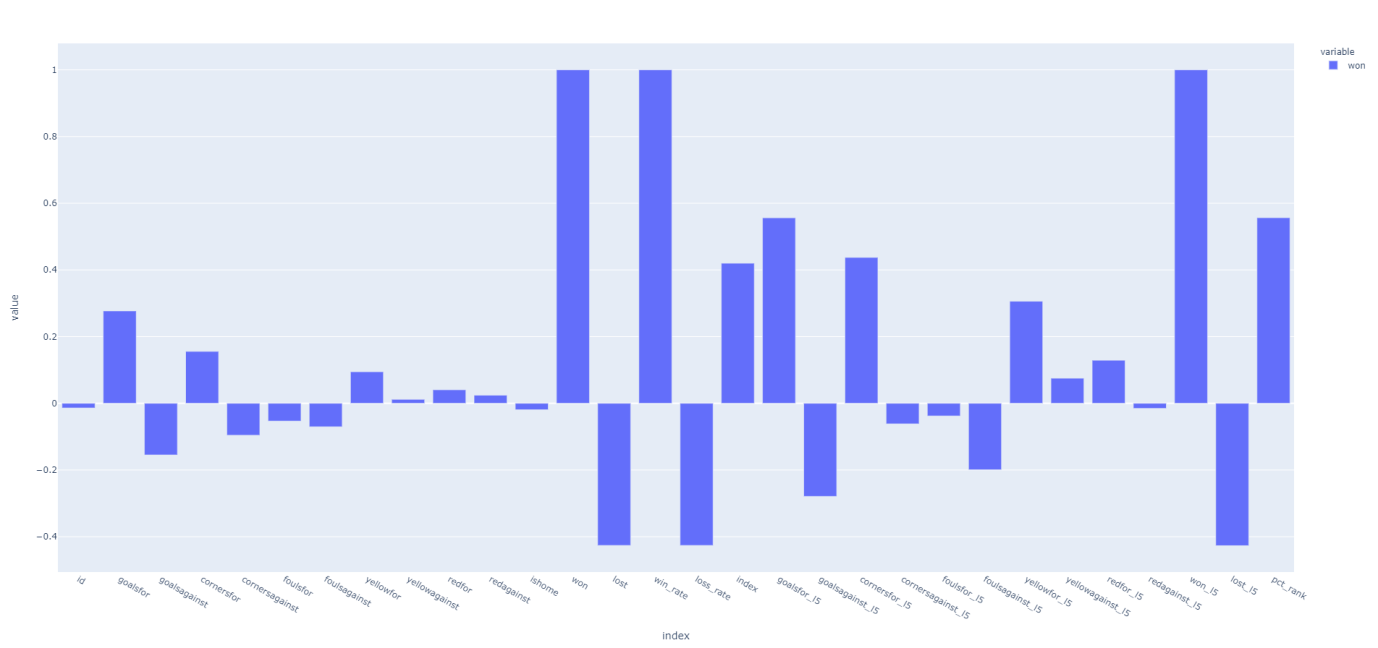
The screenshot below displays the code that is used to generate the following scatter plot:



The screenshot below displays the scatter plot generated by the code above:



There appears to be a negative correlation between the number of red cards a team receives and their win rate. This is expected as a team is at a disadvantage if they get a player sent off and are therefore less likely to win. This means Red Cards Against would not make a good feature.

4.4.11 Bar Chart showing how each feature correlates to the win rate******

From this bar chart, we can see that the most influential features towards win rate are "goalsfor\_l5", "corners for l5", "yellow for l5" and "PCT rank".

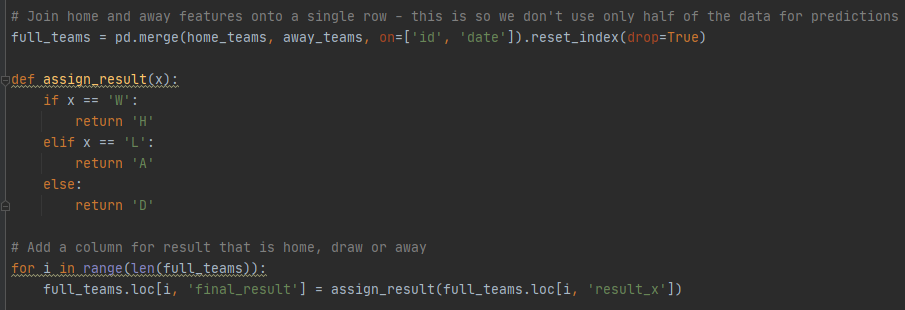
For this reason, I chose the following features to input into the models:

* Goals\_for\_l5
* Corners\_for\_l5
* Yellow\_for\_l5

The last step before I build the model is to combine the home and away teams (grouped by id and date) into a new dataframe called "full teams". This is to ensure we use the full range of data when making the prediction, rather than half the data.

I then defined a function called "assign\_result", which determines the whether the result was a win, draw, or loss.

I then used a for loop to iterate through the "full teams" dataframe that I just created, and add the result of x under a new column called "final\_result".



### Model Building

#### 4.5.1 Select Features to Use

Based on the visualisations in the previous section, I have chosen to use the following information as features for the model:

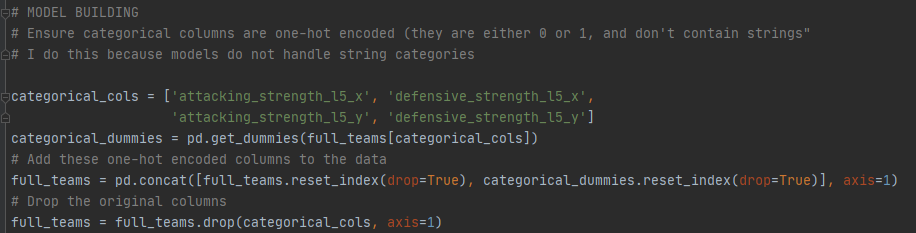
* Goals scored
* Corners for
* Fouls against

I chose these features because they appear to be the most influential factors towards winning the game.

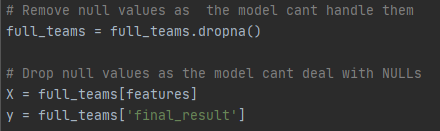
#### Refine the model

Before refining the model, I first ensure that all categorical columns are one-hot encoded (this means they are either 0 or 1) and don't contain strings. I do this because models cannot handle string categories.

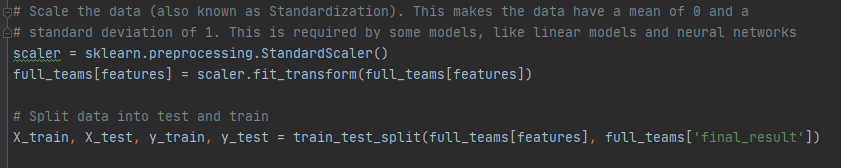
I then add the one-hot encoded columns to the "full\_teams" dataframe, drop the original columns and then redefine the features for their new columns. Here’s the code I used:



I also wanted to ensure there were no null values in the results, as models can’t handle null values. I used the following code to ensure all null values were dropped from the “full\_teams” dataframe.



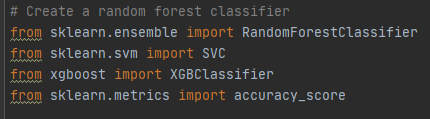
The final step when refining the model was to standardize the data. This ensures the data has a mean of 0 and a standard deviation of 1. I decided to do this as some models (such as linear models and neural networks) require the data to be scaled between 0 and 1. The data is then split until “test” and “train”. The “train” data is used to create the model, whereas the “test” data is used to verifying if the outcome is what was expected. This is the code I used:



#### 4.5.3Select ML Model to use

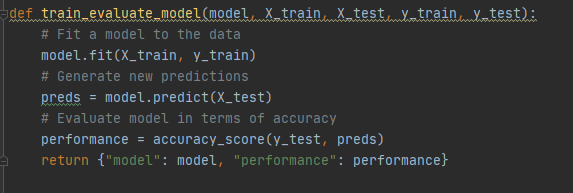
I decided to use three different types of models to test the data as they all learn from the data in different ways. The models chosen to test the data are:

* **Support Vector Machine (linear classification model)**
* **Random forest model**
* **Xgboost**

I then imported the models and also accuracy\_score from the sklearn.metric library to use for evaluating the model’s performance. I imported the Random Forest classifier from the sklearn.ensemble library, SVC linear model from sklearn.svm, and the XGBClassifier from xgboost.

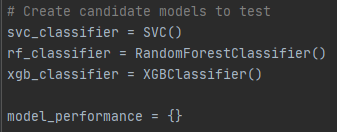
#### Generate Predictions

The next step was to train each model and evaluate it on a test set using balanced\_accuracy\_score. To do this, I fitted a model to the data, generated new predictions and evaluated the model to determine it’s accuracy. This is demonstrated in the following code:

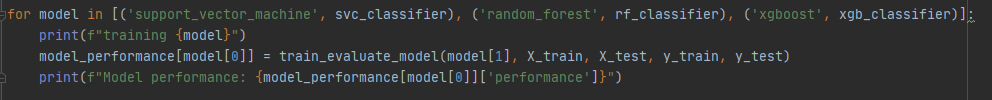


The models are now ready to be created and tested.

Next I created the candidate models that I aim to test the data on, and also an empty array to store the model performance values, using the following code:

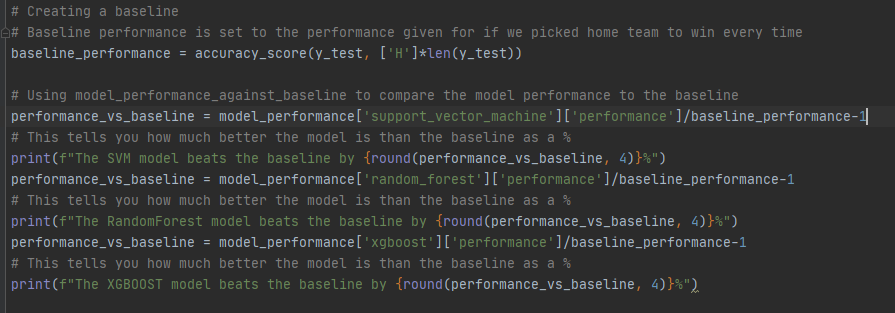


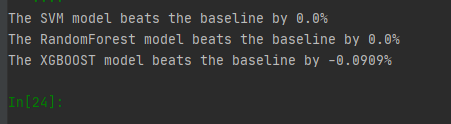
I then create a for loop to test each model. I created a list, containing the name of each model, and the model object, then I’m looping through the list and passing the model object to the evaluate\_model function, which trains it and gets its performance and saves the results in a dictionary.



#### Evaluate the Model

Now that I have the performance scores of each model, I can evaluate their performance using a baseline performance which is set to the performance given for if we picked home team to win every game, and using model\_performance\_against\_baseline to compare the model performance to the baseline.





The baseline value for if we bet on the home team each game is 0.44 (44%).  
SVM achieved a performance score of 0.44 (44% success).  
Random Forest also achieved a performance score of 0.44 (44% success).  
XGBoost achieved an accuracy score of 0.4 (40% success).

Unfortunately, due to the limited availability of free data, there accuracy of the model is poor and has the exact same performance rating as the baseline.

## Chapter 5: Critical Evalutation of Project

I believe I’ve achieved exactly what I set out to do in the project proposal which was to create the data pipeline and conduct data analysis techniques on the data to generate predictions. Unfortunately the performance of the model was no better than the baseline performance, however this project was about the process of creating the pipeline rather than the results.

I found the literature review to be a long process however I also acknowledge its importance for understanding what I’m talking about and helping me understand how the process of creating a pipeline more thoroughly.

I enjoyed working on this project and I hope that somebody else can find my work useful in the future.

## Chapter 6: Conclusion

The goal of this project was to develop a data pipeline that is able to take information from various online sources and store it in an SQLite3 database, clean the data, generate features with reasonable predictive power, use the features in a predicting model to predict the outcome of sporting events and evaluate the performance of each prediction.

The goal was achieved which was the process of creating the pipeline itself, however the performance scores were poor in the end were poor. This is down to the limited data and a lack of resources available, however with more time the features and model could be further improved.

It can be concluded that this study suggests the best model to use that offers the highest performance rating is the SVM and Random Forest models, which achieved a performance\_rating of 44%, and the most influential features to use are goals scored, corners for, and fouls against. This answers the questions raised in the literature review.

Thank you for taking the time to read this report.

# End Matter

## References

[1] A. Brent, "Which sport is the most popular for betting?", Sports Mole. [Online]. Available: https://www.sportsmole.co.uk/off-the-pitch/features/which-sport-is-the-most-popular-for-betting\_299783.html. [Accessed: 23- Feb- 2021].

[2] J. Manyika et al., "Big data: the next frontier for innovation, competition, and productivity", McKinsey Global Institute, 2011.

[3] M. Pérez-Ortiz, S. Jiménez-Fernández, P. A. Gutiérrez, E. Alexandre, C. Hervás-Martínez and S. Salcedo-Sanz, "A Review of Classification Problems and Algorithms in Renewable Energy Applications", MDPI, 2016.

[4] A. Ortner, "Top 10 Binary Classification Algorithms [a Beginner’s Guide]", Medium, 2020. [Online]. Available: https://medium.com/@alex.ortner.1982/top-10-binary-classification-algorithms-a-beginners-guide-feeacbd7a3e2. [Accessed: 01- Mar- 2021].

[5]"1.9. Naive Bayes — scikit-learn 0.24.1 documentation", Scikit-learn.org, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/naive\_bayes.html. [Accessed: 01- Mar- 2021].

[6] P. Ranganathan and R. Aggarwal, "Common pitfalls in statistical analysis: Linear regression analysis", Perspectives in Clinical Research, vol. 8, no. 2, p. 100, 2017. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5543767/.

[7] O. Harrison, "Machine Learning Basics with the K-Nearest Neighbors Algorithm", Towards Data Science, 2018. [Online]. Available: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761. [Accessed: 01- Mar- 2021].

[8] R. Ghandi, "Support Vector Machine — Introduction to Machine Learning Algorithms", Towards data science, 2018. [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47. [Accessed: 01- Mar- 2021].

[9] D. K, "Top 4 advantages and disadvantages of Support Vector Machine or SVM", Medium, 2019. [Online]. Available: https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107#:~:text=SVM%20algorithm%20is%20not%20suitable,samples%2C%20the%20SVM%20will%20underperform. [Accessed: 01- Mar- 2021].

[10] P. Gupta, "Decision Trees in Machine Learning", Medium, 2017. [Online]. Available: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052. [Accessed: 01- Mar- 2021].

[11] R. Baboota and H. Kaur, "Predictive analysis and modelling football results using machine learning approach for English Premier League", International Journal of Forecasting, vol. 35, no. 2, pp. 741-755, 2019. Available: https://www.sciencedirect.com/science/article/pii/S0169207018300116#tbl2.

[12] "Python for Fantasy Football - Feature Engineering for Machine Learning", FantasyFutopia, 2019. [Online]. Available: http://www.fantasyfutopia.com/python-for-fantasy-football-feature-engineering-for-machine-learning/. [Accessed: 02- Mar- 2021].

[13] N. Razali, A. Mustapha, F. Yatim and R. Ab Aziz, "Predicting Football Matches Results using Bayesian Networks for English Premier League (EPL)", IOP Conference Series: Materials Science and Engineering, vol. 226, p. 012099, 2017. Available: https://iopscience.iop.org/article/10.1088/1757-899X/226/1/012099.

[14] "What is a Data Pipeline? - Dremio", Dremio.com. [Online]. Available: https://www.dremio.com/data-lake/data-pipeline/. [Accessed: 11- Feb- 2021].

[15] Illustration of how AutoAI builds a machine learning model. 2020.

[16] "How to create a CSV file", Computerhope.com, 2021. [Online]. Available: https://www.computerhope.com/issues/ch001356.htm#:~:text=CSV%20is%20a%20simple%20file,Microsoft%20Excel%20or%20OpenOffice%20Calc.&text=Its%20data%20fields%20are%20most,or%20delimited%2C%20by%20a%20comma. [Accessed: 12- Feb- 2021].

[17] "sqlite3 — DB-API 2.0 interface for SQLite databases — Python 3.9.1 documentation", Docs.python.org. [Online]. Available: https://docs.python.org/3/library/sqlite3.html. [Accessed: 15- Feb- 2021].

[18] "DB Browser for SQLite", Sqlitebrowser.org. [Online]. Available: https://sqlitebrowser.org/. [Accessed: 15- Feb- 2021].

[19] I. Ilyas and X. Chu, Data cleaning, 1st ed. Association of Computing Machinery and Morgan & Claypool Publishers, 2019, pp. 1-2.

[20] J. Osborne, Best practices in data cleaning. Thousand Oaks, Calif.: Sage, 2013, pp. 4-8.

[21] T. Pigott, "A Review of Methods for Missing Data", Educational Research and Evaluation, vol. 7, no. 4, pp. 362-363, 2001. Available: https://galton.uchicago.edu/~eichler/stat24600/Admin/MissingDataReview.pdf. [Accessed 15 February 2021].

[22] M. Greenwood-Nimmo and K. Shields, "An Introduction to Data Cleaning Using Internet Search Data", Australian Economic Review, vol. 50, no. 3, pp. 363-372, 2017. Available: 10.1111/1467-8462.12235.

[23] D. Shah, J. Wang and Q. He, "Feature engineering in big data analytics for IoT-enabled smart manufacturing – Comparison between deep learning and statistical learning", Computers & Chemical Engineering, vol. 141, p. 106970, 2020. Available: https://www.sciencedirect.com/science/article/pii/S0098135420300363#sec0016.

[24] J. Brownlee, "Discover Feature Engineering, How to Engineer Features and How to Get Good at It", Machine Learning Mastery, 2020. [Online]. Available: https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/. [Accessed: 16- Feb- 2021].

[25] "Representation: Qualities of Good Features", Google Developers. [Online]. Available: https://developers.google.com/machine-learning/crash-course/representation/qualities-of-good-features. [Accessed: 16- Feb- 2021].

[26] T. Bock, "What is Feature Engineering?", ADVANCED ANALYSIS | MACHINE LEARNING | USING DISPLAYR | WHAT IS.... .

[27]"What is Binning?", Docs.tibco.com. [Online]. Available: https://docs.tibco.com/pub/spotfire/7.0.1/doc/html/bin/bin\_what\_is\_binning.htm. [Accessed: 16- Feb- 2021].

[28] P. Vaish, "The Comprehensive Guide for Feature Engineering", A Data Analyst. .

[29] I. Kononenko, E. Simec and M. Robnik-sikonja, "Overcoming the Myopia of Inductive Learning Algorithms with RELIEFF", The International Journal of Research on Intelligent Systems for Real Life Complex Problems, vol. 7, pp. 39–55, 1997. Available: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.4740&rep=rep1&type=pdf. [Accessed 16 February 2021].

[30] Relief Wiki. 2018.

[31] F. Bocca and L. Rodrigues, "The effect of tuning, feature engineering, and feature selection in data mining applied to rainfed sugarcane yield modelling", Computers and Electronics in Agriculture, vol. 128, pp. 67-76, 2016. Available: https://www.sciencedirect.com/science/article/pii/S0168169916306391#b0150.

## Appendix A: Project Documentation

### A.1. Project Proposal

|  |  |
| --- | --- |
| **Computing Project Proposal Form** |  |

**Student Name: Daniel James Collins email address: s18001869@glyndwr.mail.ac.uk**

Course: **BSc Computer Science**

|  |
| --- |
| **Proposed Project Title:**  Domain Driven Data Mining: Predictive Analysis Techniques |

|  |
| --- |
| **Requested Supervisor(s):**  Bindu Jose |

|  |
| --- |
| **Project Outline:**  The aim is to design an analytical model that will demonstrate predictive analysis techniques, and develop methodologies and tools for predicting football outcomes and calculating odds. |

|  |
| --- |
| **Rationale for choice of Project:**  I have chosen to base my research on Football data, specifically the Premier League, as this is a competition which generates lots of statistical data, which I can use to demonstrate the different tools and techniques used in predictive analysis. I want to pursue a career in data science when I leave university, and this will be a good topic to gain experience using modern data mining techniques. |

|  |
| --- |
| **People with whom you have discussed the project (e.g. employer, members of the lecturing staff):** |

|  |
| --- |
| **Research Areas of Study:**  Data Science, SQL |

|  |
| --- |
| **Intended Deliverables (i.e. intended research outcomes and product implementation):**  A github repository consisting of python scripts and a sqlite3 database |

|  |
| --- |
| **Resources Needed by Project (other than those already available at Glyndŵr or your place of work):** |
| **Any restrictions, adaptations or other considerations due to Covid-19?** |

**Student Signature: Daniel Collins Date:**

### A.2. Project Specification

|  |  |
| --- | --- |
| **Computing Project Specification Form** |  |

**Student Surname: Collins email address: s18001869@mail.glyndwr.ac.uk**

**Student Forenames: Daniel James**

**Course: BSc Computer Science**

**Supervisor: John Worden, Bindu Jose**

|  |
| --- |
| **Project Title:** Domain Driven Data Mining: Predictive Analysis Techniques |

|  |
| --- |
| **Project Aims and Objectives:**  Develop a data pipeline consisting of the following steps:  - Takes information from various online sources and stores it in a SQLite3 database  - Clean this data, removing outliers and imputing missing data  - Generate features that have reasonable predictive power (features that tell us something about what we hope to predict)  - Use the features in a predictive model to predict the outcome of sporting events.  - Evaluate the predictions of this model |

|  |
| --- |
| **Research Areas of Study:**  predictive analysis, data science, programming, SQL |

|  |
| --- |
| **Methodology:**  - Undertake a literature review of available methodologies and tools used for predictive analysis techniques and models. Notably, the following topics will be explored:  - Data Storage  - SQLite3 database design  - Data Cleaning  - Handling missing data  - Outlier removal  - Feature Engineering  - Explore makes a good feature, and research state of the art feature engineering techniques  - Model building and evaluation  - Explore the best performing models in the industry, explore their strengths and weaknesses in relation to this task  - Explore methods of evaluating models, including developing an effective baseline to compare models against.  - Evaluate the data sources available online, that could be used in the project. Design a database structure that can support this data in the optimal way.  - Design scripts to extract data from online sources and store it in the SQLite3 database.  - Develop scripts for cleaning the data and generating features  - Pick a list of candidate models, and test each model at predicting the outcome of sporting events. Evaluate these models against a baseline and pick the best performing one.  - Provide a set of documentation and a user-guide for the model  - Consider how the model could be improved further in the future |

|  |
| --- |
| **Project Outcomes and Deliverables:**  A github repository consisting of python scripts and an sqlite3 database |

|  |
| --- |
| **Project Timetable:**  The final report submission date is Friday 7th May and the project demonstration is on Thursday 13th May. I will be liaising with my project supervisor on a weekly basis, where I will ask any questions as well as providing updates on progress. Information from meetings will be recorded in the project log book.  Prepare poster for poster party: 8 days  Write literature review: 20 days  Analyse and compare tools: 7 days  Design and create SQLite3 database: 15 days  Design and develop Python model: 25 days  Implement, evaluate and test model and provide recommendations: 15 days  Document model and create user-guide: 8 days  Consider future adjustments and uses: 7 days  Prepare for demonstration of artefact: 10 days |

**Student Signature: D.Collins Date: 04/12/2020**

**Agreed by Supervisor**

## Appendix B: Program Code

Main.py

import pandas as pd  
  
# We put all functions inside their own script to make this script cleaner  
from utils import connect\_to\_db, run\_query, convert\_data\_type\_names, clean\_query  
  
from download\_data import generate\_download\_links  
  
# Define location of the db  
db\_dir = 'db.sqlite'  
# Define the name of the table to hold the data  
table\_name = 'results'  
  
  
def GetResults(link, dropresults):  
  
 # Load data  
 df = pd.read\_csv(link)  
 # Add season to the data  
 df['season'] = link.split('/')[-2]  
 # Create connection to DB  
 conn = connect\_to\_db('db.sqlite')  
 # Create the cursor  
 cursor = conn.cursor()  
  
 # Convert column names to lower case (easier to write)  
 df.columns = [col.lower() for col in df.columns]  
 # Convert 'as' to 'away\_shots' and 'hs' to 'home\_shots'  
 # We do this because 'as' is a command in SQL so it cant be a column name  
 df = df.rename(columns={'as': 'away\_shots', 'hs': 'home\_shots'})  
 # Get the data types for each of the columns  
 data\_types = dict(df.dtypes)  
  
 # Create a description of the columns, used to create the table  
 column\_description = ''.join([f"{key} {convert\_data\_type\_names(value)}, " for key, value in data\_types.items()])  
 # Remove the ", " from the end of the description  
 column\_description = column\_description[:-2]  
  
 # if statement to decide whether to clear results table or not  
 if dropresults == True:  
 # Clear the table if it already exists  
 run\_query(query=f'DROP TABLE IF EXISTS {table\_name}', return\_data=False, path\_to\_db=db\_dir)  
 # Create the table  
 query = f"CREATE TABLE {table\_name} ({column\_description})"  
 # Remove any non-alphabet symbols  
 query = clean\_query(query)  
 run\_query(query=query, return\_data=False, path\_to\_db=db\_dir)  
 table\_cols = list(data\_types.keys())  
 else:  
 # *TODO: Get a list of columns that exist in the table, if we arent recreating it* res = run\_query(query=f'select \* from {table\_name} limit 1', return\_data=True, path\_to\_db=db\_dir)  
 data\_types2 = dict(res.dtypes)  
 table\_cols = list(res.columns)  
  
 # Upload the results to a table, row by row. We do this by taking the keys of the dictionary,  
 # and joining them together  
 column\_names = ', '.join(table\_cols)  
 # Select all columns stored in table\_cols, drop everything else  
  
 # Add any columns that exist in the sql table, but dont exist in the new csv  
 missing\_columns = [col for col in table\_cols if col not in df.columns]  
 for col in missing\_columns:  
 df[col] = 'NULL'  
  
 # Drop any columns that exist in the new csv, but not in the sql table.  
 df = df[table\_cols]  
  
 # Combine the schema of the csv file and the sql file (if we are adding data)  
 if dropresults == False:  
 data\_types = {\*\*data\_types, \*\*data\_types2}  
  
 for i in range(len(df)):  
 # Get the i'th row, convert it to a list  
 row = list(df.iloc[i, :])  
 for j in range(len(df.columns)):  
 # IF the data type is a string, we need to add ' ' around it  
 data\_type = data\_types[df.columns[j]]  
 if str(data\_type) == 'object':  
 row[j] = "'" + str(row[j]) + "'"  
 # Convert all elements of the list to strings  
 string\_row = ", ".join([str(x) for x in row])  
 # Add the values to the database  
 query = f"INSERT INTO {table\_name} ({column\_names}) VALUES ({string\_row})"  
 query = clean\_query(query)  
 run\_query(query=query, return\_data=False, path\_to\_db=db\_dir, params={})  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 dlinks = generate\_download\_links(2010, 2020)  
 dropcondition = 0 # Set dropcondition to 0  
  
 for footballlinks in dlinks: # For every element in dlinks, call GetResults and pass the link  
 if dropcondition == 0:  
 dropresults = True  
 else:  
 dropresults = False  
 dropcondition = dropcondition + 1 # Add 1 to drop condition each time  
 print(f"Running data for {footballlinks}")  
 GetResults(footballlinks, dropresults)  
  
 # Check that the values have been added  
 result = run\_query(query=f'select \* from {table\_name}', return\_data=True, path\_to\_db=db\_dir)

download.data.py

def generate\_download\_links(start\_year=2018, end\_year=2020): # defining the function  
 *"""Given a start year and an end year, generate a list of links for  
 premier league data from football-data.co.uk"""* output = list() # defining an empty list to use later  
  
 # Find out how many links to generate  
 num\_loops = end\_year - start\_year  
  
 for i in range(num\_loops): # For loop to generate the links dynamically (automatically)  
 # Generate the year\_string variable  
 season\_start\_year = str(start\_year + i)[-2:]  
 season\_start\_end = str(int(season\_start\_year) + 1).zfill(2)  
 year\_string = season\_start\_year + season\_start\_end  
 link = f"https://www.football-data.co.uk/mmz4281/{year\_string}/E0.csv"  
  
 output.append(link) # updating the output list with the new string  
 return output  
  
  
def test\_generate\_download\_links(): # Here we are testing the function  
 expected\_output = [  
 'https://www.football-data.co.uk/mmz4281/1819/E0.csv',  
 'https://www.football-data.co.uk/mmz4281/1920/E0.csv'  
 ]  
 assert generate\_download\_links(2018, 2020) == expected\_output, "Test failed"  
 # If the output is equal to the expected output then the test passes  
 print("Test passed")  
  
  
def test\_generate\_download\_links\_one\_digit\_year():  
 expected\_output = [  
 'https://www.football-data.co.uk/mmz4281/0102/E0.csv',  
 'https://www.football-data.co.uk/mmz4281/0203/E0.csv'  
 ]  
 assert generate\_download\_links(2001, 2003) == expected\_output, "Test failed"  
 # Test whether the calculation works for years prior to 2010  
 print("Test passed")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Prevents the code beneath from running all code if I'm importing just one function, rather than the whole code  
 # \_\_name\_\_ refers to the script that's being run  
 test\_generate\_download\_links()  
 test\_generate\_download\_links\_one\_digit\_year()  
 links = generate\_download\_links(2001, 2020)  
 # for link in links:  
 # get\_football\_data(link)

utils.py

import os  
import re  
import sqlite3  
import pandas as pd  
  
  
def connect\_to\_db(path\_to\_db):  
 *"""Connect to local sqlite3 database  
 """* # Establish a connection to the database  
 if not os.path.exists(path\_to\_db): # Check if the path to the DB exists  
 print('DB not found, creating DB at this location')  
 conn = sqlite3.connect(path\_to\_db)  
 # Return the connection object to use in queries  
 return conn  
  
  
def run\_query(\*, query, params=None, return\_data=True, path\_to\_db=None) -> pd.DataFrame:  
 *"""Function to run a query on the DB while still keeping the column names. Returns a DataFrame  
 """* # Create connection object  
 conn = connect\_to\_db(path\_to\_db)  
 # Create cursor (which you use to run queries)  
 cursor = conn.cursor()  
 # Run query  
 cursor.execute(query, params if params is not None else [])  
 # Commit any changes  
 conn.commit()  
 # Get column names and apply to the data frame  
 if return\_data:  
 names = cursor.description # Get the column names  
 name\_list = []  
 for name in names:  
 name\_list.append(name[0]) # Add the column names to a list  
 # Convert the result into a DataFrame and add column names  
 df = pd.DataFrame(cursor.fetchall(), columns=name\_list)  
 conn.close()  
 return df  
 # Close the connection  
 conn.close()  
  
  
def convert\_data\_type\_names(x):  
 *"""Convert the python data type names into the sqlite data type names  
 """* # Make sure x is a string  
 x = str(x)  
 if 'object' in x:  
 return 'TEXT'  
 elif 'float' in x:  
 return 'REAL'  
 elif 'int' in x:  
 return 'INTEGER'  
  
  
def clean\_query(query):  
 *"""Remove any non-alphabetic symbols from query text  
 """* query = query.replace('<', 'u')  
 query = query.replace('>', 'o')  
 query = query.replace('nan', 'NULL')  
 # Define all the symbols we want to be allowed in the text  
 regex = re.compile("[^a**-**zA**-**Z0123456789 ,()\_']")  
 # Replace any characters that arent defined above, with nothing  
 # First parameter is the replacement, second parameter is your input string  
 return regex.sub('', query)

exploratory\_date\_analysis.py

import pandas as pd  
import numpy as np  
import plotly.express as pe  
import plotly.io as pio  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import roc\_auc\_score  
import sklearn  
  
  
# all functions are inside their own script to make this script cleaner  
from utils import connect\_to\_db, run\_query, convert\_data\_type\_names, clean\_query  
  
from download\_data import generate\_download\_links  
  
# Define location of the db  
db\_dir = 'db.sqlite'  
# Define the name of the table to hold the data  
table\_name = 'results'  
  
result = run\_query(query=f'select \* from {table\_name}', return\_data=True, path\_to\_db=db\_dir)  
  
cols = ['fthg', 'ftag', 'hthg',  
 'htag', 'home\_shots', 'away\_shots', 'hst', 'ast',  
 'hf', 'af', 'hc', 'ac', 'hy', 'ay', 'hr', 'ar']  
result.loc[result['season'] == '1415', cols] = result.loc[result['season'] == '1415', cols]/10  
  
result.groupby('season').mean()  
  
  
# Does Home Advantage play a part in winning the game?  
# Percent of times the home team wins compared to draw and away win  
percent\_of\_home\_wins = len(result[result["ftr"] == "H"]) / len(result) # Show bool of if ftr is H  
percent\_of\_home\_draws = len(result[result["ftr"] == "D"]) / len(result) # Show bool of if ftr is D  
percent\_of\_home\_loss = len(result[result["ftr"] == "A"]) / len(result) # Show bool of if ftr is A  
assert np.round(percent\_of\_home\_wins + percent\_of\_home\_draws + percent\_of\_home\_loss) == 1, ""  
# Check percentage includes 100% games  
  
# Get percent of Full Time wins for Home Teams  
percent\_of\_home\_wins = len(result[result["ftr"] == "H"]) / len(result) # Show bool of if ftr is H  
print(percent\_of\_home\_wins)  
# Get percent of Half Time Leads for Home Teams  
percent\_of\_halftime\_lead\_h = len(result[result["htr"] == "H"]) / len(result)  
percent\_of\_halftime\_lead\_d = len(result[result["htr"] == "D"]) / len(result)  
percent\_of\_halftime\_lead\_a = len(result[result["htr"] == "A"]) / len(result)  
  
# Is Team Form a good predictor of whether the team will win their next game?  
# Team Form based on the Home Goals in previous 5 games  
result.groupby("hometeam").fthg.rolling(  
 5, min\_periods=5, center=False, win\_type=None, on=None, axis=0, closed=None).mean()  
# This is okay, but I want overall form  
# I need one row per team per game to find the Home Form value  
result["id"] = result.index  
homecols = ["id", "date", "hometeam", "fthg", "ftag", "hc", "ac", "hf", "af", "hy", "ay", "hr", "ar", "season"]  
homecolsnew = ["id", "date", "team", "goalsfor", "goalsagainst", "cornersfor", "cornersagainst", "foulsfor",  
 "foulsagainst", "yellowfor", "yellowagainst", "redfor", "redagainst", "season"]  
df\_home = result[homecols]  
df\_home.columns=homecolsnew  
df\_home.loc[:, "ishome"] = True  
  
# Team Form based on the Away Goals in previous 5 games  
awaycols = ["id", "date", "awayteam", "ftag", "fthg", "ac", "hc", "af", "hf", "ay", "hy", "hr", "ar", "season"]  
awaycolsnew = ["id", "date", "team", "goalsfor", "goalsagainst", "cornersfor", "cornersagainst", "foulsfor",  
 "foulsagainst", "yellowfor", "yellowagainst", "redagainst", "redfor", "season"]  
df\_away = result[awaycols]  
df\_away.columns=awaycolsnew  
df\_away.loc[:, "ishome"] = False  
  
# Split home and away teams into individual rows  
teamform = df\_home  
teamform = teamform.append(df\_away)  
teamform = teamform.sort\_values('id')  
assert len(teamform) == len(result) \* 2 # Testing new dataframe is the correct size  
  
  
# Make a rolling average window for features  
# Team Form for past 5 games (excluding most recent game)  
# Lag goals for, so we don't generate features from the game we are trying to predict  
  
  
  
def get\_result(x):  
 if x['goalsfor'] > x['goalsagainst']:  
 return "W"  
 elif x['goalsfor'] < x['goalsagainst']:  
 return "L"  
 else:  
 return "D"  
  
  
# Applies the 'get\_result' function to every row in teamform, adds results to a new col called "result"  
  
  
teamform['result'] = teamform.apply(get\_result, axis=1)  
  
# Team Form : Add columns  
teamform['won'] = teamform['result'] == 'W' # Adds "won" col to teamform df stating if team won  
teamform['lost'] = teamform['result'] == 'L' # Adds "lost" col to teamform stating if team lost  
  
win\_rates = teamform.groupby(['team', 'season']).won.mean().reset\_index() # The win rate of all teams  
win\_rates.columns = ['team', 'season', 'win\_rate']  
teamform = pd.merge(teamform, win\_rates, on=['team', 'season'])  
teamform['won'] = teamform['win\_rate']  
  
loss\_rates = teamform.groupby(['team', 'season']).lost.mean().reset\_index() # The win rate of all teams  
loss\_rates.columns = ['team', 'season', 'loss\_rate']  
teamform = pd.merge(teamform, loss\_rates, on=['team', 'season'])  
teamform['lost'] = teamform['loss\_rate']  
  
def add\_rolling\_average(df, column='goalsfor', window=5, min\_periods=5):  
 # Shift so I dont use the current game score as part of the average  
 df[f'{column}\_lagged'] = df.groupby('team')[column].shift(1)  
 test = df.groupby("team")[f"{column}\_lagged"].rolling(window, min\_periods).mean().reset\_index()  
 test.columns = ['team', 'index', f"{column}\_l{window}"]  
 df['index'] = df.index.copy()  
 df = pd.merge(df, test, how='left', on=['team', 'index'])  
 df.drop(f'{column}\_lagged', axis=1, inplace=True)  
 return df  
  
  
teamform = add\_rolling\_average(teamform, column='goalsfor', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='goalsagainst', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='cornersfor', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='cornersagainst', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='foulsfor', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='foulsagainst', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='yellowfor', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='yellowagainst', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='redfor', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='redagainst', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='won', window=5, min\_periods=5)  
teamform = add\_rolling\_average(teamform, column='lost', window=5, min\_periods=5)  
  
  
# Calculate the Attacking and Defending strength of each team  
# Get average goals from last 5 games to determine attacking strength  
# Use Pandas Rank to decide the rank and remove human bias  
# Compute numerical data ranks (1 through n) along axis  
# Create a Ranking function that I can reuse on different columns  
  
def ranking\_function(teamform, col\_name, new\_col\_name, ascending=True):  
  
 teamform['pct\_rank'] = teamform[col\_name].shift(1).rank(ascending=ascending, pct=True)  
  
 # Now I need to create the column "attacking\_strength\_l5" in team form  
 # Create a list of conditions  
 conditions = [  
 (teamform['pct\_rank'] <= 0.33),  
 (teamform['pct\_rank'] >= 0.66),  
 (teamform['pct\_rank'] > 0.33) & (teamform['pct\_rank'] < 0.66)  
 ]  
  
 # Create a list of values we want to assign for each condition  
 attack\_form\_values = ["low", "high", "medium"]  
  
 # Create a new column and use np.select to assign values  
 teamform[new\_col\_name] = np.select(conditions, attack\_form\_values)  
 return teamform  
  
  
teamform = ranking\_function(teamform, "goalsfor", "attacking\_strength\_l5" )  
teamform = ranking\_function(teamform, "goalsagainst", "defensive\_strength\_l5", ascending=False)  
  
  
teamform['pct\_rank'] = teamform['goalsfor\_l5'].rank(pct=True)  
  
# Now I need to create the column "attacking\_strength\_l5" in team form  
# Create a list of conditions  
conditions = [  
 (teamform['pct\_rank'] <= 0.33),  
 (teamform['pct\_rank'] >= 0.66),  
 (teamform['pct\_rank'] > 0.33) & (teamform['pct\_rank'] < 0.66)  
 ]  
  
# Create a list of values we want to assign for each condition  
attack\_form\_values = ["low", "high", "medium"]  
  
# Create a new column and use np.select to assign values  
teamform["attacking\_strength\_l5"] = np.select(conditions, attack\_form\_values)  
  
# Display updated DataFrame  
teamform.head()  
  
# Add a Column to check whether the team won or not  
# First define the get\_result function  
  
  
# teamform['result'] = teamform.apply(lambda x: get\_result(x), axis=1)  
  
  
  
# VISUALISATION  
  
pio.renderers.default = 'browser'  
  
# Plot goals for each match for home team  
import plotly  
home\_teams = teamform[teamform['ishome']==True]  
away\_teams = teamform[teamform['ishome']==False]  
home\_goals = pe.histogram(home\_teams, x='goalsfor', title='Goals scored by home team (histogram)')  
plotly.offline.plot(home\_goals)  
away\_goals = pe.histogram(away\_teams, x='goalsfor', title='Goals scored by away team (histogram)')  
plotly.offline.plot(away\_goals)  
  
# Team Summary : Add columns  
# team\_summary = teamform.groupby(['team', 'season'])['won', 'lost', 'goalsfor', 'goalsagainst',  
# 'goalsfor\_l5', 'goalsagainst\_l5', 'cornersfor\_l5',  
# 'cornersagainst\_l5', 'foulsfor\_l5', 'foulsagainst\_l5',  
# 'yellowfor\_l5', 'yellowagainst\_l5', 'redfor\_l5', 'redagainst\_l5'].mean().reset\_index()  
# team\_summary.corr() # This gives me a correlation matrix. Easy way to visualise data.  
  
  
# Scatter plot: Goals for vs win rate  
plot\_goalsFor = pe.scatter(teamform, x='won\_l5', y='goalsfor\_l5', labels='season', color='team', title='Goals for vs win rate')  
plotly.offline.plot(plot\_goalsFor)  
# plot\_goalsFor.show()  
# This shows a positive correlation suggesting a better attacking strength based on form may equate to a higher win rate  
# This would make a good feature.  
  
# Scatter plot: Goals against vs win rate  
plot\_goalsAgainst = pe.scatter(teamform, x='won\_l5', y='goalsagainst\_l5', labels='season', color='team', title='Goals Against vs win rate')  
plotly.offline.plot(plot\_goalsAgainst)  
# Bar Chart: Goals Against vs win rate  
  
  
# Scatter plot: Corners for vs win rate  
plot\_CornersFor = pe.scatter(teamform, x='won\_l5', y='cornersfor\_l5', labels='season', color='team', title='Corners for vs win rate')  
plotly.offline.plot(plot\_CornersFor)  
# Bar Chart: Corners for vs win rate  
# bar\_CornersFor = pe.bar(teamform, x='won\_l5', y='cornersfor\_l5', labels='season', color='team', title='Corners for vs win rate')  
# plotly.offline.plot(bar\_CornersFor)  
  
  
# Scatter plot: Corners against vs win rate  
plot\_CornersAgainst = pe.scatter(teamform, x='won\_l5', y='cornersagainst\_l5', labels='season', color='team', title='Corners against vs win rate')  
plotly.offline.plot(plot\_CornersAgainst)  
# Bar Chart: Corners against vs win rate  
# bar\_CornersAgainst = pe.bar(teamform, x='won\_l5', y='cornersagainst\_l5', labels='season', color='team', title='Corners against vs win rate')  
# plotly.offline.plot(bar\_CornersAgainst)  
  
  
# Scatter plot: Fouls committed vs win rate  
plot\_FoulsFor = pe.scatter(teamform, x='won\_l5', y='foulsfor\_l5', labels='season', color='team', title='Fouls for vs win rate')  
plotly.offline.plot(plot\_FoulsFor)  
# Bar Chart: Fouls committed vs win rate  
# bar\_FoulsFor = pe.bar(teamform, x='won\_l5', y='foulsfor\_l5', labels='season', color='team', title='Fouls for vs win rate')  
# plotly.offline.plot(bar\_FoulsFor)  
  
  
# Scatter plot: Fouls against vs win rate  
  
# Bar Chart: Fouls against vs win rate  
# bar\_FoulsAgainst = pe.bar(teamform, x='won\_l5', y='foulsagainst\_l5', labels='season', color='team', title='Fouls against vs win rate')  
# plotly.offline.plot(bar\_FoulsAgainst)  
  
# Scatter plot: Yellow cards for vs win rate  
plot\_YellowFor = pe.scatter(teamform, x='won\_l5', y='yellowfor\_l5', labels='season', color='team', title='Yellow card for vs win rate')  
plotly.offline.plot(plot\_YellowFor)  
# Bar Chart: Yellow cards for vs win rate  
# bar\_YellowFor = pe.bar(teamform, x='won\_l5', y='yellowfor\_l5', labels='season', color='team', title='Yellow card for vs win rate')  
# plotly.offline.plot(bar\_YellowFor)  
  
# Scatter plot: Yellow cards against vs win rate  
plot\_YellowAgainst = pe.scatter(teamform, x='won\_l5', y='yellowagainst\_l5', labels='season', color='team', title='Yellow card against vs win rate')  
plotly.offline.plot(plot\_YellowAgainst)  
# Bar Chart: Yellow cards against vs win rate  
# bar\_YellowAgainst = pe.bar(teamform, x='won\_l5', y='yellowagainst\_l5', labels='season', color='team', title='Yellow card against vs win rate')  
# plotly.offline.plot(bar\_YellowAgainst)  
  
  
# Scatter plot: Red card for vs win rate  
plot\_RedFor = pe.scatter(teamform, x='won\_l5', y='redfor\_l5', labels='season', color='team', title='Red card for vs win rate')  
plotly.offline.plot(plot\_RedFor)  
# Bar Chart: Red card for vs win rate  
# bar\_RedFor = pe.bar(teamform, x='won\_l5', y='redfor\_l5', labels='season', color='team', title='Red card for vs win rate')  
# plotly.offline.plot(bar\_RedFor)  
  
  
# Scatter plot: Red card against vs win rate  
plot\_RedAgainst = pe.scatter(teamform, x='won\_l5', y='redagainst\_l5', labels='season', color='team', title='Red card against vs win rate')  
plotly.offline.plot(plot\_RedAgainst)  
# Bar Chart: Red card against vs win rate  
# bar\_RedAgainst = pe.bar(teamform, x='won\_l5', y='redagainst\_l5', labels='season', color='team', title='Red card against vs win rate')  
# plotly.offline.plot(bar\_RedAgainst)  
  
  
# Bar Chart showing how each feature correlates to each other  
cor = teamform.corr()['won']  
fig = pe.bar(cor)  
plotly.offline.plot(fig)  
  
  
# Join home and away features onto a single row - this is so we don't use only half of the data for predictions  
full\_teams = pd.merge(home\_teams, away\_teams, on=['id', 'date']).reset\_index(drop=True)  
  
# Evaluation metric - Accuracy, because the classes (H/D/A) are fairly balanced  
def assign\_result(x):  
 if x == 'W':  
 return 'H'  
 elif x == 'L':  
 return 'A'  
 else:  
 return 'D'  
  
# Add a column for result that is home, draw or away  
for i in range(len(full\_teams)):  
 full\_teams.loc[i, 'final\_result'] = assign\_result(full\_teams.loc[i, 'result\_x'])  
  
  
# MODEL BUILDING  
# Ensure categorical columns are one-hot encoded (they are either 0 or 1, and don't contain strings"  
# I do this because models do not handle string categories  
  
categorical\_cols = ['attacking\_strength\_l5\_x', 'defensive\_strength\_l5\_x',  
 'attacking\_strength\_l5\_y', 'defensive\_strength\_l5\_y']  
categorical\_dummies = pd.get\_dummies(full\_teams[categorical\_cols])  
# Add these one-hot encoded columns to the data  
full\_teams = pd.concat([full\_teams.reset\_index(drop=True), categorical\_dummies.reset\_index(drop=True)], axis=1)  
# Drop the original columns  
full\_teams = full\_teams.drop(categorical\_cols, axis=1)  
# Redefine features with the new columns  
features = [  
 # Home team features  
 'goalsfor\_l5\_x', 'cornersfor\_l5\_x', 'foulsagainst\_l5\_x',  
 # Away team features  
 'goalsagainst\_l5\_y', 'cornersagainst\_l5\_y', 'foulsfor\_l5\_y'  
] + list(categorical\_dummies.columns)  
  
# Remove null values as the model cant handle them  
full\_teams = full\_teams.dropna()  
  
# Drop null values as the model cant deal with NULLs  
X = full\_teams[features]  
y = full\_teams['final\_result']  
  
# Scale the data  
scaler = sklearn.preprocessing.StandardScaler()  
full\_teams[features] = scaler.fit\_transform(full\_teams[features])  
  
# Split data into test and train  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(full\_teams[features], full\_teams['final\_result'])  
  
# Create a random forest classifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from xgboost import XGBClassifier  
from sklearn.metrics import accuracy\_score  
  
# Models to test  
# Linear model  
# Random forest  
# XGBoost  
  
# Train each model, evaluate it on a test set, using balanced\_accuracy\_score  
  
def train\_evaluate\_model(model, X\_train, X\_test, y\_train, y\_test):  
 # Fit a model to the data  
 model.fit(X\_train, y\_train)  
 # Generate new predictions  
 preds = model.predict(X\_test)  
 # Evaluate model in terms of accuracy  
 performance = accuracy\_score(y\_test, preds)  
 return {"model": model, "performance": performance}  
  
  
# Create candidate models to test  
svc\_classifier = SVC()  
rf\_classifier = RandomForestClassifier()  
xgb\_classifier = XGBClassifier()  
  
model\_performance = {}  
  
for model in [('support\_vector\_machine', svc\_classifier), ('random\_forest', rf\_classifier), ('xgboost', xgb\_classifier)]:  
 print(f"training {model}")  
 model\_performance[model[0]] = train\_evaluate\_model(model[1], X\_train, X\_test, y\_train, y\_test)  
 print(f"Model performance: {model\_performance[model[0]]['performance']}")  
  
# *TODO: Compare the performance of each model and see which is the best*# Creating a baseline  
# Baseline performance is set to the performance given for if we picked home team to win every time  
baseline\_performance = accuracy\_score(y\_test, ['H']\*len(y\_test))  
  
# Using model\_performance\_against\_baseline to compare the model performance to the baseline  
performance\_vs\_baseline = model\_performance['support\_vector\_machine']['performance']/baseline\_performance-1  
# This tells you how much better the model is than the baseline as a %  
print(f"The SVM model beats the baseline by {round(performance\_vs\_baseline, 4)}%")  
performance\_vs\_baseline = model\_performance['random\_forest']['performance']/baseline\_performance-1  
# This tells you how much better the model is than the baseline as a %  
print(f"The RandomForest model beats the baseline by {round(performance\_vs\_baseline, 4)}%")  
performance\_vs\_baseline = model\_performance['xgboost']['performance']/baseline\_performance-1  
# This tells you how much better the model is than the baseline as a %  
print(f"The XGBOOST model beats the baseline by {round(performance\_vs\_baseline, 4)}%")