Exploring the ability of artificial intelligence to predict football results and rankings

Project Report for MSc in Data Science

Author: Christos Frantzis

Supervisor: Dr. David Weston

Department of Computer Science and Information Systems

Birkbeck, University of London

September 2021

Academic Declaration

This report is substantially the result of my own work, expressed in my own words, except where explicitly indicated in the text. I have read and understood the sections on plagiarism in the Programme Handbook and the College website. I give my permission for it to be submitted to the JISC Plagiarism Detection Service.

Abstract

Machine learning is a form of artificial intelligence that is used to construct predictive algorithms in sports and other fields.

This project reviews the available literature, and its aim is to improve the performance and accuracy of the proposed models by these studies. Six unique classifiers are trained and tested to identify the most accurate on predicting the multi-class match result of a full season. The average accuracy of these studies was 59% and the challenge is to predict results that cannot be explained by the match statistics but other factors that are often not measurable.

Additional features are engineered based on available and scraped data to offer further information on the classifiers and their importance is explored.

Feature selection reduced the size of the dataset and produced greater scores.

After hyperparameter tuning, the models forged the best results among all studies and a ranking table was formed for the purpose of comparison.

Overall, a simple Artificial Neural Network was the top classifier and achieved 68% predictive accuracy with an adequate estimate of draws which is the most difficult result to predict due to its randomness. Based on its prediction and the ranking table, the champion was predicted accurately as well as five of the top six teams and two of the three relegated teams.

Table of Contents

1. Introduction 3

1.1. Report Structure 3

1.2. Related Work 4

1.3. Objectives 5

1.4. Amendments 6

1.5. Tools & Techniques 6

2. Data Preparation 7

2.1. Data Collection 7

2.2. Data Exploration 8

2.3. Feature Engineering 10

2.4. Missing & Non-numeric Data 13

3. Methodology 14

3.1. Pre-processing & Feature Selection 15

3.2. Models 16

3.3. Hyperparameter Tuning 18

3.4. Metrics 18

4. Critical Evaluation 19

4.1. Results Comparison 19

4.2. Limitations 25

5. Conclusion 25

5.1. Summary 25

5.2. Future Work 26

6. References 28

7. Appendix 31

I. Dataset Features 31

II. Code 39

1. Introduction

Football is probably the most famous sport in the world with around 4 billion fans (Statistics & Data, 2020). English Premier League (EPL) audience reached 3.2 billion people in season 2018/2019 (Premier League, 2019). Each year different types of data emerge, and people are interested in them for different reasons. The backroom staff can isolate these data to improve their training methods and tactics, evaluate each players' strengths and prevent injuries or single out the best from a list of potential signings. The board can assess the worth of the team's best players and make a decision on offers or find ways to increase profit by targeting specific data and attract sponsors. Most importantly these data are used for match prediction as fans are intrigued by correctly estimating the potential winner and betting on their pick.

However, football can be quite unpredictable and unexpected results often occur. A team can dominate a match and not be able convert its chances to goals, but its opponent may score in their first shot on target and win. These events are extremely difficult to be determined by algorithms, which require improvement.

This project explores a range of these algorithms that are supplied with a large number of known and constructed features within different seasons and after making fine adjustments for a positive impact on their results, attempts to identify the best of them.

1.1. Report Structure

The project report is organised as follows:

- Section 1 revisits papers discussed in project proposal and focuses mostly on the features used to produce these results, identifies the objectives and questions to be answered by the project, discloses adjustments that had to be made on the proposal aims and acknowledges the tools used to accomplish the task.

- Section 2 concentrates on the dataset used, where it was found, how it was expanded and the way it was manipulated to fit the project's criteria.

- Section 3 describes the chosen models and their underline structure and hyperparameters, explains the rationale behind specific data processing and metrics selection.

- Section 4 evaluates and presents the answers to the questions specified in Section 1 and identifies the limitations that affected the project.

- Section 5 issues a brief summary of all steps followed to complete the project and the overall outcome, as well as future testing and developments plans.

- Section 6 and 7 provide the bibliography and the appendix respectively, where the full code of the project and additional figures be located.

1.2. Related Work

A few papers have attempted to predict accurately the outcome of a football match. The majority of these result-based studies used data from at least ten seasons and are presented below in chronological order.

Tsakonas et al. (2003) were one of the first to use Support Vector Machines (SVM) for this objective. They focused on features containing information about injuries of home and away teams, the scores of each team for the last five games, the point difference of the teams competing, and they defined a host factor to indicate the total points at home and away matches up to that time.

Ulmer et al. (2013) were not able to acquire a range of data and settled with basic features like score, winner and goals scored by each team. Then they calculated the form of each home and away team for the last 4 and 7 games as an indicator of a streak and a goal difference feature was used on selected models because it resulted in controversial results in some of the models with the most accurate being Random Forest (RF) and Support Vector Machines (SVM).

Igiri & Nwachukwu (2014) used an Artificial Neural Network (ANN) for multiclass classification to predict win, lose or draw and Logistic Regression (LR) for binary classification to predict win or loss. Features like performance index, goals, shots, corners, betting odds, streak and manager's streak for each home and away team proved to be useful. However, they only used data from one season of the EPL.

Baboota & Kaur (2018) focused on features with a host factor and each new feature was calculated for both home and away teams. They managed to extract ratings for each team from FIFA's video game database and identified goal difference as one of the most crucial features for the task. Finally, they designed a streak and form feature which indicated each team's result streak and overall performance (goals, shots on target, corners) respectively, based on previous matches. Their best results were produced by Extreme Gradient Boosting (XGBoost), Random Forest and Support Vector Machines.

Hessels (2019) isolated team rankings from a built-in R library and produced the cumulative sum of points, goals and shots on target for each team as well as goal difference. Form and weighted form were calculated based on the last 4 and 7 games. Form provided a sum of the points collected from previous matches and weighted form displayed a score based on the previous results. Random Forest and Support Vector Machines were implemented to replicate Ulmer's study.

Raju et al. (2020) used a dataset consisting of five seasons of the EPL and created features like home/away team goals scored per game, home/away team goals conceded per game and home/away win percentage. Their models managed to produce great results as SVM and LR for multi-classification resulted in 65% and 65.3% accuracy respectively.

An average of each model's predicting accuracy (except ANN which used only one season) from the studies mentioned above is displayed below:

Models	**Average Predicting Accuracy (Average Dataset Size)**
Random Forest	54.6% - 12 Seasons
Support Vector Machines	58.2% - 10 Seasons
Linear Regression*	65.3% - 5 Seasons
Extreme Gradient Boosting	58% - 10 Seasons
Total	59% - 9 Seasons

* Igiri et al. used LR for binary classification; it is not included in the average

1.3. Objectives

The purpose of this project is to incorporate the best attributes of each of the above studies and identify new methods to improve results predictions. A combination of similar features will be constructed, and further additions will be explored. A range of different classifiers will emphasise mainly on accuracy and less consideration will be given on computational cost.

The main questions this project aims to answer and additional details regarding them are outlined below:

- Can the selected classifiers go the extra mile and achieve consistent predicting accuracy of at least 60%?

  - Experiment with multiple classifiers of variable complexity and feed them with no less than eight seasons of data.

  - Attempt to produce equivalent or better results than Raju et al.

- To what extent did the engineered features and their addition to the original dataset impacted the prediction accuracy?

  - Provide results before and after feature engineering.

  - Provide results after feature selection of the combined features.

  - Identify the most informative features of the new dataset.

- How well did the best scoring classifier predict the final ranking table?

  - Compute rankings based on predicted results and compare side-by-side with actual table.

1.4. Amendments

A number of amendments from the project proposal have resulted after taking into consideration the feedback that it received.

First, the web scraping process is de-emphasised as it would be extremely time consuming to collect the entire dataset and it would require additional hours for personal development training to accomplish it. Additionally, the majority of the websites mentioned in the proposal are not user friendly for scraping purposes or they do not allow scraping of their data via software. However, an attempt to produce a limited number of features via this technique will be made.

Second, the 20/21 season will be discarded as it is the season COVID-19 affected the most where no fans were allowed to attend any game for the whole season and the home advantage factor was possibly affected. Therefore, a sample of seasons with more equal distribution will be selected to produce the best possible results.

Third, the website deployment will be abandoned in the interest of accomplishing an exceptional outcome. The project will experiment with an increased number of models comparing to three that were declared in the proposal, in order to provide better results than any of the studies mentioned in Subsection 1.2. Therefore, Django and Google Cloud will not be used so the use of SQL will not be required as a database will not be necessary.

1.5. Tools & Techniques

In order to deliver the results presented in this report various tools were utilised and a list of these can be seen below:

- Text Editor:
  - Microsoft Word
- Spreadsheet Viewer:
  - Google Sheets
- Coding:
  - Programming Language: Python
    - Python Libraries:
      - Pandas (Dataset Manipulation)
      - Numpy (Scientific Computing)
      - BeautifulSoup (Web Scraping)
      - Itertools (Dictionary Manipulation)
      - Scikit Learn, XGBoost (Machine Learning Classifiers & Metrics)

    - Keras, Tensorflow (Neural Network Classifier)
    - Matplotlib, Seaborn
  - Integrated Development Environment (IDE): Visual Studio Code (VSCode)
  - Jupyter Notebooks
- Version Control:
  - GitHub
- Troubleshooting:
  - Official documentation of mentioned tools
  - Stack Overflow

2. Data Preparation

The initial idea was to scrap multiple seasons of official data through various websites, but it was rejected after a number of failed attempts as described in Subsection 1.4. Additionally, the majority of these websites contained mostly features from betting sites like odds for result winner or range of goals which are out of the scope of this project.

2.1. Data Collection

The original dataset was obtained from the data science website Kaggle, and it is called "All Premier League Matches 2010-2021". The size of the dataset is approximately 2.3 MB. According to the description, the dataset is compiled with data from the official English Premier League website and contains specifics from 4070 matches. A single EPL season involves 20 teams which play each other two times bringing the total number matches to 380 per year. The dataset consists of 113 features that seven are categorical and the rest of them are numeric containing very specific data from each match for both the home and away team. A list of the original dataset features can be seen in Appendix I.

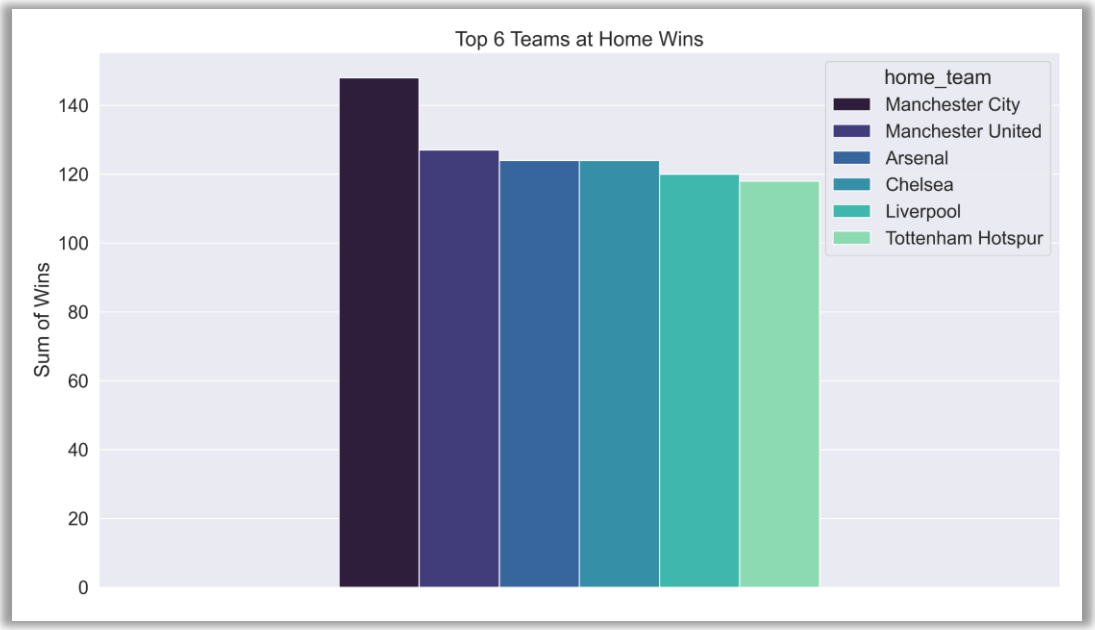
It is worth pointing out that 110 matches are missing from the 2020/2021 season but the choice to discard it from the project was already made prior to this finding. Therefore, 433,200 data points from 3800 matches will result from season 2010/2011 to 2019/2020 after the removal of the affected 'COVID-19' season.

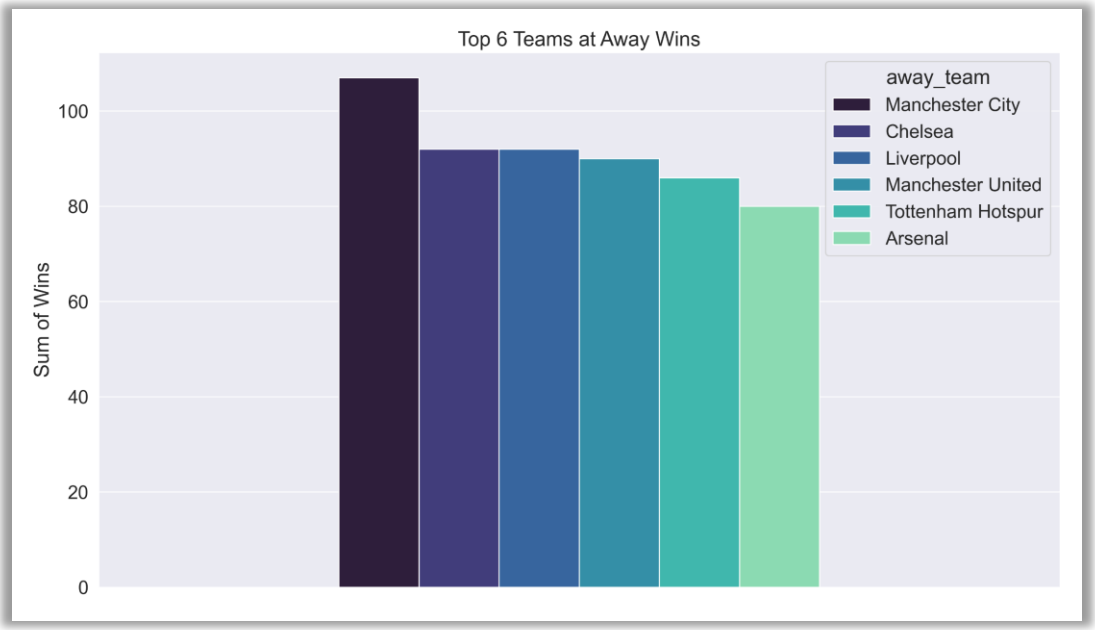
Additional data from the public website *FIFA Index* (https://www.fifaindex.com/) will be scraped using Python and BeautifulSoup. This website contains ratings such as Attack, Midfield, Defence and Overall, for each team from each year's FIFA video game. These ratings are calculated before the start of the season by specialised algorithms used by Electronic Arts Sports and their data science team.

2.2. Data Exploration

The first column of the dataset is *match_id* and contained values like M1, M2 etc. to distinguish between each match. The letter M was removed from each value of the column to better assist sorting based on that column as without a letter its values can be treated as integers and not strings. A column on the last position of the dataset named *game_result* was created to be the target variable of the project and points out the winner based on the score provided. The letters A, D and H indicate an away win, a draw and a home win respectively.

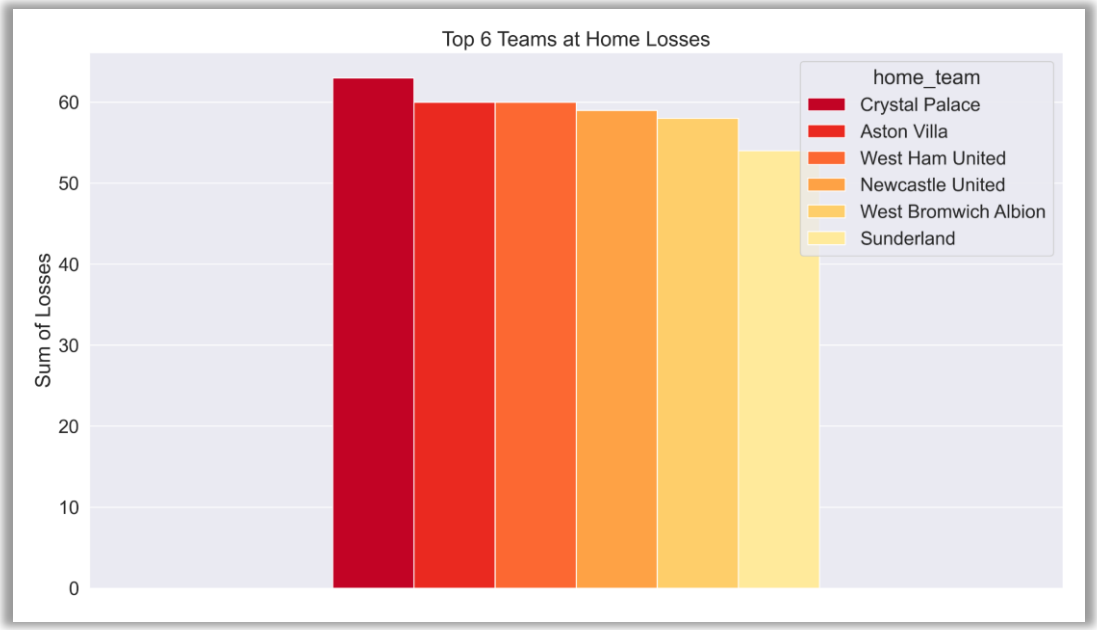
Within 10 seasons, 37 unique teams have participated in the EPL. The top six teams on home and away wins can be visualised below.

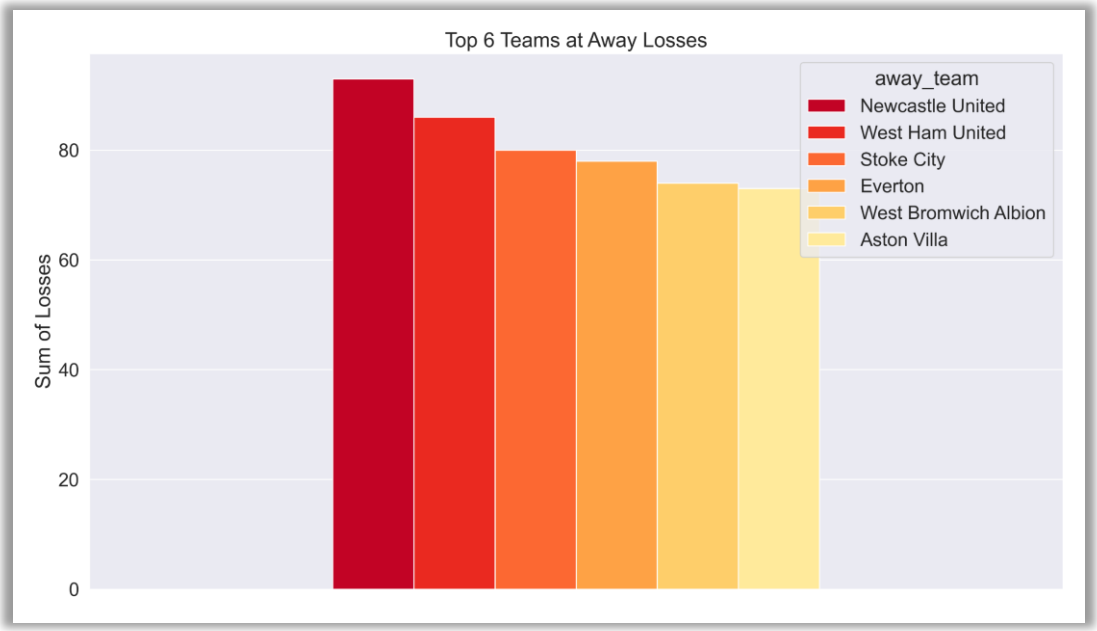




The teams are the same within the two graphs and Manchester City being the best with more than 250 combined wins. In total two teams from Manchester and four teams from London can be rightfully called the best teams in the modern era of the EPL.

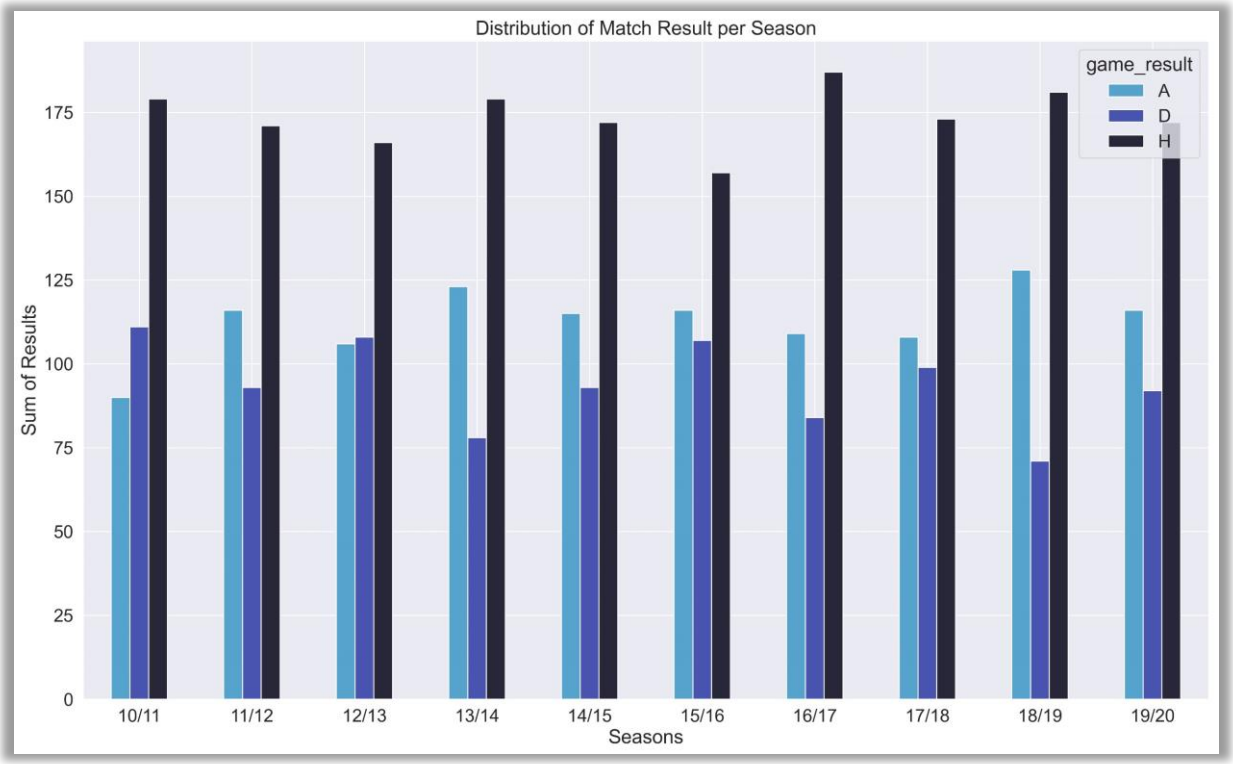
On the contrary, the unique teams with the most home and away losses are eight and the majority of them are not located in the biggest cities of England.





However, a percentage and not an aggregate of losses might be more conclusive, to capture teams that have been promoted and relegated multiple times and not only teams that have been in the EPL for many years and competed in a larger number of games resulting in more defeats.

The distribution of each class of the *game_ result* feature over each season can be observed in the following bar graph.



Although the proportion of each class varies over each season, home win (H) is the majority class every year with a large discrepancy between draw (D) and away win (A) classes. This finding was expected as attending fans and playing at home where most of the training and matches are played can be an enormous uplift for the hosts. Draw class is at close proximity to away win, with away win being the majority class between them except seasons 2010/2011 and 2012/2013.

As a result, the three classes are not equally spread in the dataset causing a common issue referred as the class imbalance problem and explains the struggle for high quality predicting results on past papers.

2.3. Feature Engineering

With the focus on boosting predicting accuracy, new features will be formed. These features are calculated for each season independently and do not carry information from previous years. Each feature will offer separate data for both home and away teams competing in a match to correlate with the original features. In order to create these features and keep track of them, each team is stored as a key to a Python dictionary and the keys' values are comprised of the corresponding estimate for each team. The total number of new features created is 34 including the game result target variable, are described in detail below and their unique labels are listed in Appendix I.

Attack, Midfield, Defence, Overall Ratings

Inspired by Baboota & Kaur (2018), FIFA ratings were extracted as described in Subsection 2.1. Ten instances were missing after the web scraping process, and they were added manually. Names of two teams had slight variations to the names of the dataset and were alternated to be compatible with the rest of the code. For each match, four home and four away team ratings for that specific season were assigned accordingly. These features aim to indicate a team's strength and best attributes depending on formed rosters before the start of each season. They are not interchangeable based on results, but it will be considered for a future addition.

Win, Draw & Lose Streaks

Three features for home and three features for the away team were introduced to point out the form of a team based on their current streak record of wins, draws and losses. Each of these features indicates the required streak accumulated until the previous match. Therefore, every team on 1st fixture has 3 unique streak variables with a value of zero. An example of a single team's home matches and win, draw and away streaks is shown on the table below.

Matchday	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Result	H	H	D	H	A	H	H	H	A	A
Home Win Streak	0	1	2	0	1	0	1	2	3	0
Home Draw Streak	0	0	0	1	0	0	0	0	0	0
Home Away Streak	0	0	0	0	0	1	0	0	0	1

- Interesting Facts from Exploratory Data Analysis:

- The top home win streak was recorded by Liverpool in season 2019/2020 with 17 consecutive home wins.

- The top away win streak was recorded by Manchester City in season 2017/2018 with 10 consecutive away wins.

- The top home and away lose streaks were recorded by Wolverhampton in season 2011/2012 with 9 and Queens Park Rangers in season 2014/2015 with 11 straight losses.

Cumulative Home, Away, Total Combined Team Points & Point Difference

Five features in total were manufactured for both home and away teams to sum up all the points acquired when playing specifically at home or away, total points (home and away results combined) for the home and away teams and calculate the point difference as an

indicator of the current ranking between competing teams up to the previous match. The following table illustrates features assigned to the home team when playing either at home or away.

Matchday	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Playing at Home	☑		☑	☑			☑		☑	
Result	H	H	D	H	A	H	H	H	A	A
Home Points Acum	0	3	3	4	7	7	7	10	10	10
Home Team Total Points Acum	0	3	6	7	10	10	13	16	19	19

Cumulative Home, Away, Total Combined Team Goals & Goal Difference

Eight new features for home/away teams and 2 combined were constructed up to the last match as the rest of the features and they covered:

- Goals scored specifically at home or away
- Total goals scored by the current home and away team
- Total goals conceived by the current home and away team
- Goals achieved/Goals against difference by subtracting the above respective features

A sample of the home team's variable is demonstrated as follows:

Matchday	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Playing at Home	☑		☑	☑			☑		☑	
Score	2-0	3-1	1-1	1-0	0-2	4-2	1-0	0-0	0-1	1-2
Home Goals For Acum	0	2	2	3	4	4	4	5	5	5
Home Goals Against Acum	0	0	0	1	1	1	1	1	1	2
Home Total Goals Acum	0	2	5	6	7	7	11	12	12	12
Home Total Goals Against Acum	0	0	1	2	2	4	6	6	6	7

Form - Last 3 & Last 6

Two features for either home and away team were fabricated to identify the current form for each team on the last 3 and last 6 matches until the last match. For each team a Python list was initiated to include the points acquired after each match as a string, with

the latest being on the left of the string. A substring containing only the 3 or 6 latest matches' points was appended to the list which was then included as a value to a Python dictionary matching up with each team. Then, each integer value of the 3-letter or 6-letter string is multiplied by a number to give extra value and weight to the latest matches. For last 3 matches form, the latest match points acquired are calculated by 1.2, the second last match points are calculated by 0.8 and the third last is multiplied by 0.4. These values were selected to produce a single fractional part of a decimal number. For last 6 matches form, each value of the string is multiplied by 1.2, 1, 0.8, 0.6, 0.4 and 0.2 respectively. The maximum value for last 3 form is 7.2 and the minimum 0. The maximum value for last 6 form is 12.6 and the minimum 0. The computed string for each team is not included on the final dataset but is displayed below for the purpose of comprehension.

Matchday	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Result	H	H	D	H	A	H	H	H	A	A
Last 3 (String)	'0'	'3'	'33'	'133'	'313'	'031'	'303'	'330'	'333'	'033'
Form (Last 3)	0	3.6	6	4.8	5.6	2.8	4.8	6	7.2	3.6

The form value for matchday 10 is calculated as follows:

(0*1.2) + (3*0.8) + (3*0.4) = 0 + 2.4 + 1.2 = 3.6

Matchday	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Result	H	H	D	H	A	H	H	H	A
Last 6 (String)	'0'	'3'	'33'	'133'	'3133'	'03133'	'303133'	'330313'	'333031'
Form (Last 6)	0	3.6	6.6	6.6	8.8	6.8	8.4	9.4	10.4

The form value for matchday 10 is calculated as follows:

(3*1.2) + (3*1) + (3*0.8) + (0*0.6) + (3*0.4) + (1*0.2) = 3.6 + 3 + 2.4 + 0 + 1.2 + 0.2 = 10.4

2.4. Missing & Non-numeric Data

There is a number of information absent from the original dataset that required to be dealt with, as the majority of the machine learning algorithms used in the project do not support datasets with missing or non-numeric values.

Non-Numeric Values

The first two fixtures of each season came with a large number of non-numeric values (NaN). It is a common issue with public datasets to have corrupted data which were not recorded or were not able to be gathered.

A decision was made to discard the first 3 matchdays of each season to accommodate the creation of features *Form - Last 3 & Last 6* and provide more representing information for each team.

Furthermore, NaN values were still existing randomly after the removal of these fixtures. These were replaced with the mean value of each respective column instead of a zero to preserve the columns' average and not create new extreme values.

Missing & 'Extra' Match

There was a match missing from the 2nd fixture (17th August 2015) of season 2015/2016. In this match Liverpool won 1-0 at home against AFC Bournemouth. The match was manually added to the dataset via Google Sheets. Features *link_match, season, date, home_team, away_team, result_full, result_ht* were appended to accommodate the expected steps of the analysis. The rest of the features could not be retrieved and left blank. Nevertheless, it was decided to remove the first 3 matchdays of each season as reported above that proved to be extremely convenient.

Additionally, a match from the 1st fixture of season 2011/2012 was placed between following fixtures. The match between Tottenham Hotspur and Everton was postponed due to riots emerged after the death of a local British man by a police gunshot and it was played on 11th January 2012 instead. This discrepancy confused the code while feature engineering and as a result it was manually moved via Google Sheets to the 1st fixture despite matches being sorted based on dates.

After these changes the *match_id* was reset to coordinate with the rest of the dataset.

Finally, two datasets will be exported to test the models proposed in Subsection 3.1. The first dataset will include the original features with the addition of game_result target variable and the second dataset will comprise of the original and engineered features (148 in total) for the purpose of comparison.

3. Methodology

Multiple approaches are going to be set side by side and will indicate if and in what degree the engineered features impact the outcome. First, the selected models are going to be introduced to the dataset with the original features. Then, the uncondensed final dataset with the combined features will be tested and a brief comparison will be made. Finally, an attempt to improve the model's performance by feature selection and hyperparameter tuning techniques will be made and the overall results will be reported and evaluated.

3.1. Pre-processing & Feature Selection

Before feeding the dataset to the models, further modifications need to take place to reduce possible bias and error.

First, features that imply the winner or provide information about the match of each row were discarded and are listed below:

- match_id: The index of the dataset is stored to the memory and match_id is not required
- link_match & date: Link on the official Premier League website and date of the match is irrelevant to the project
- home_team & away_team: Team names were removed so the classifier does not become biased on a specific team rather focus on the match statistics to predict the winner
- result_full & result_ht: Both imply the full- and half-time score of the current match
- goal_home_ft, goal_away_ft, goal_home_ht, goal_away_ht, sg_match_ft & sg_match_ht: Provide values of current match and not up to the previous match as the engineered features.

Second, the target variable game_result was encoded as an integer via Scikit-Learn's Label Encoder, and it was assigned a 0 for an away win, a 1 for a draw and a 2 for a home win. This helped avoiding errors while training and testing some of the classifiers.

Third, the train and test datasets were initiated formulated on predefined seasons. The train dataset included data from seasons 2010/2011 till 2018/2019 and the test dataset included season 2019/2020. The train/test split was not completely random as the project objectives made it clear this was the season to work upon.

Then, features were scaled with the help of Scikit-Learn's Min Max Scaler which converts each column's values to a range from 0 to 1. This technique is essential because many of the models and their hyperparameters cannot interact with unscaled data or data with different and higher scales given can be assigned more significant weight and considered more important than other features.

Finally, for feature selection, Random Forest Feature Importance was calculated based on the training dataset and after setting a threshold, 42 features were selected, and the train and test datasets were adjusted. Of these 42 features, 7 out of 34 resulted from the engineered and 35 out of 99 from the original features. These feature selection method was chosen over Recursive Feature Elimination for its efficiency and not having a requirement to set the appropriate number of features.

3.2. Models

Based on the related literature and on general performance in multiclass problems, the following classifiers where chosen:

Logistic Regression (LR)

Logistic Regression introduced by Cox (1958) is a statistical model which is fed with historical data and approximates the probability of an event to happen and ideally is used for binary classification purposes. It was selected for its simplicity, efficient training and easy maintenance and updating.

In order to calculate and predict the multiclass game result target variable, changes to the default settings need to be done. Hyperparameter *multi_class* is required to be set on multinomial to initiate multinomial logistic regression for multiclassification problems. Different optimisation solvers and parameters to adjust the strength of penalty will be explored.

Random Forest (RF)

Random Forest is a classification model that was proposed by Breiman (2001) and is formed by multiple decision trees working as a group. Each decision tree predicts one of the classes and the final prediction is made based on the majority of all the decision trees' predictions. Although training this model is sometimes a slow process, it is rewarding with low overfitting risk, good accuracy and performance on multiclass problems, and they are not influenced by outliers.

There is no specific hyperparameter to be set for multiclassification on Random Forest but a number of trees, a range of different samples to split the nodes of each tree and the number of features to choose from while looking for the ideal split will be tested when tuning the model.

Support Vector Machines (SVM)

Cortes and Vapnik (1995) established the Support Vector Machine classifier that are based on the idea of identifying a line called hyperplane that separates the classes in the best possible way. The most important part is the data points close to the hyperplane called support vectors because they institute the separation criteria for each class. The higher the distance between support vectors, the most accurate the classification will be.

SVM was selected for its efficiency and fast training along with its effectiveness on multiclass problems.

The most important hyperparameter to test will be the kernel of the model which receives the data and transforms them in a higher dimension space to find the right fit for the hyperplane.

K-Nearest Neighbours (KNN)

K-Nearest Neighbours, proposed by Altman (1992), is based on the idea of resemblance and the assumption that similar data points are probably in close proximity with each other. The distance between these points is calculated by default using Euclidean methods.

KNN is very easy to interpret and implement multiclassification tasks and it provides fast training. The most significant hyperparameter to consider is the number of neighbours that form a group of similar attributes.

Extreme Gradient Boosting (XGBoost)

XGBoost presented by Chen & Guestrin (2016) is also a decision tree based classifier like RF and it usually performs better due its gradient boosting technique. Gradient boosting minimises error using the incorporated gradient descent optimisation algorithm which tries to identify the local minimum value inside a function.

XGBoost can handle large datasets, has good performance, is less prone to overfitting and is fast. It has a variety of significant hyperparameters which were evaluated in the model tuning process.

Artificial Neural Network (ANN)

ANN was first presented by McCulloch and Pitts (1943) and its name was influenced by the neural network of the human brain where neurons transfer information between different areas.

Each ANN consists of three components called input layer, hidden layer and output layer. The input layer is the place where the initial data points are entered. Then, inside the hidden layers mathematic operations occur and specific weights are assigned to each point. Finally, the output layer is responsible for producing an output on which predictions will be formed.

ANNs have the ability to recognise patterns and dependencies even on incomplete and imbalanced datasets and is extremely reliable on datasets with a large number of features like ours. However, they are computationally more expensive than the rest of the models examined and can be slow to train.

In order to deal with the multiclassification problem, the loss function of the output layer has to be set as *sparse_ categorical_ crossentropy* with in our case produces a vector of 3

predictions which needs to be converted to an array in order to calculate the required metrics. A loss function indicates how well the model predicts the anticipated outcome.

Regarding the hyperparameters, manual selection from a range of epochs and batch size will be explored. Epochs refer to the number of times the constructed ANN model will pass the whole training dataset into it and the batch size indicates the amount of samples that go through the ANN at one time. If the project's dataset size was 500 data points and the batch size was set to 10 it means 10 data points can be passed at once through the network. Considering the definition of an epoch, it will take 50 batches to complete a full epoch.

3.3. Hyperparameter Tuning

For the purpose of achieving the best possible model performance, each classifier needs to go through a process of tweaking. Every classifier has different hyperparameters for specific reasons (binary/multiclass classification) and a range of them has to be explored to identify the optimal values.

Grid Search, which is a part of Scikit Learn library, is the tool of choice that will undertake the task of distinguishing between a variety of hyperparameters and it will be performed on the training dataset. It is a very exhaustive and computationally expensive method A Python dictionary is created to fit all the hyperparameters that the researcher wants to test and the rest of them that are not mentioned, will be set to default.

Then, Grid Search passes every possible combination of hyperparameters to the pre-defined model which is tested on the training dataset with Cross Validation methods. Cross Validation resamples the data to increase the randomness and effectiveness after every turn.

Finally, the best score is presented along with a Python dictionary where the best combination of the provided hyperparameters is displayed. These can be then implemented on the specific model and be tested on the unseen data.

3.4. Metrics

With a focus on models' performance, different tools of quantitative assessment will be used to evaluate it.

To visualise the true labels of a class and the predictions of each model, a 3x3 confusion matrix will be estimated and presented as a graph. The classification accuracy on the train and test dataset will also be calculated and displayed.

The classification report by Scikit Learn provides information about precision, recall and F1 score. Precision identifies the percentage of the correct true positive predicted labels of a class. Recall is similar to precision, but it shows the correct true positive labels over all

positive predicted labels that could have been produced. F1 score is one of the most used metrics for imbalanced data classification and it combines both precision and recall in a single metric. It is generally more effective and accurate within multiclass problems and if the weighted average is selected, it takes into account class imbalance by estimating the number of instances of each class based on the dataset size and defines different weight on them. A score of 1.0 or 100% represents perfect precision and recall values and it has zero false positives and false negatives.

Matthew's Correlation Coefficient (MCC) includes all values of a confusion matrix (true positive, true negative, false positive, false negative) and is one of the best options for multiclassification problems as it determines all values and catches instances of imbalanced classes. It ranges from -1 to 1 and indicates the correlation of true and predicted labels. A score of 1 will be rewarded to the best possible model.

4. Critical Evaluation

The project inspected 3 different methods to create an appealing overall opinion.

First method tested the original dataset with 100 relevant features on the proposed classifiers. Then, the same classifiers were introduced to the final dataset of 133 original and engineered features and both results are set side by side below. Lastly, the final dataset was compressed by feature selection techniques and the final verdict on the best classifier was made.

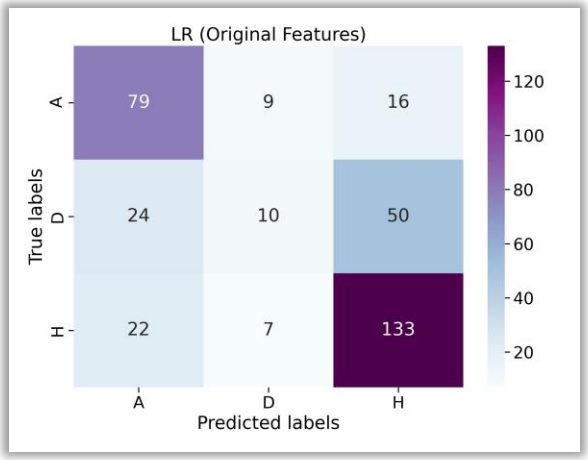
Predictions on each match result from the top classifier were mapped back to the dataset and a ranking table of season 2019/2020 was constructed.

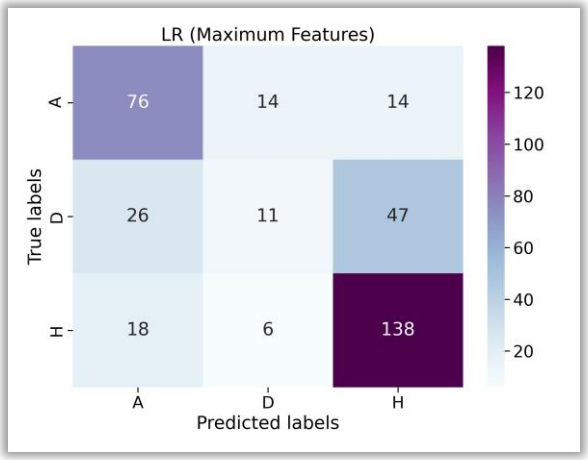
4.1. Results Comparison

The two first methods metrics and confusion matrices will be stack up against each other to stimulate critical thinking.

Logistic Regression

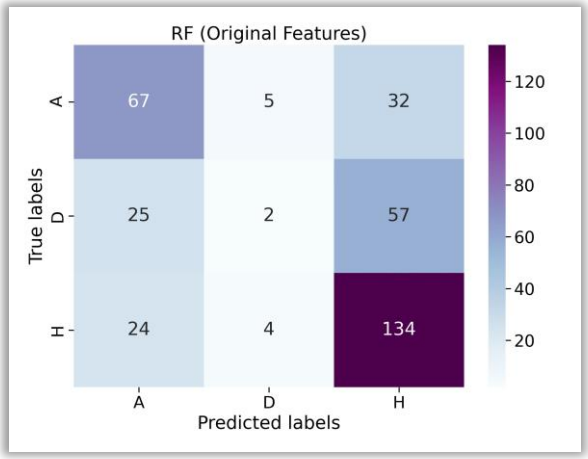
	Method 1 (Original Features)	**Method 2 (Max Features)**
Training Accuracy	68.13 %	68.92 %
Testing Accuracy	63.43 %	64.29 %
MCC	0.42	0.43
F1 Score (Weighted Average)	0.59	0.6

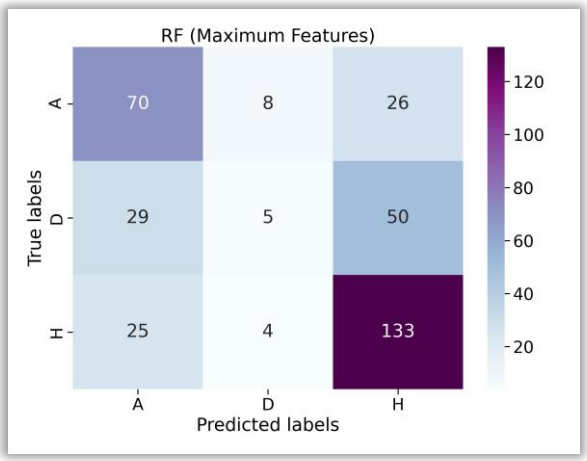




Random Forest

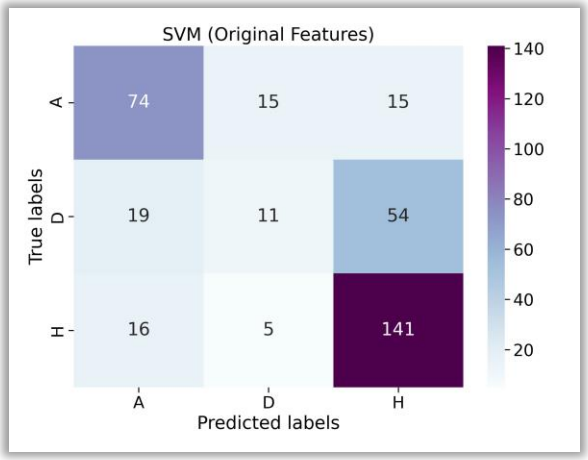
	Method 1 (Original Features)	**Method 2 (Max Features)**
Training Accuracy	100 %	100 %
Testing Accuracy	58 %	59.43 %
MCC	0.32	0.35
F1 Score (Weighted Average)	0.51	0.54

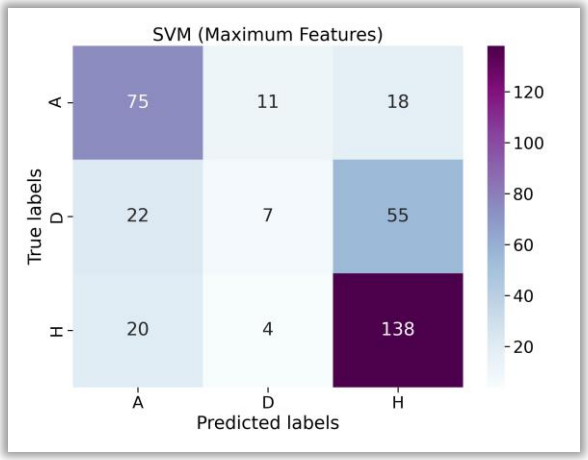




Support Vector Machines

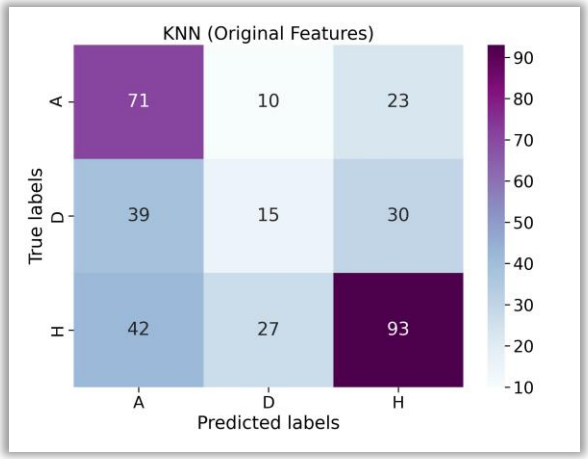
	Method 1 (Original Features)	**Method 2 (Max Features)**
Training Accuracy	75.37 %	74.95 %
Testing Accuracy	64.57 %	62.86 %
MCC	0.43	0.41

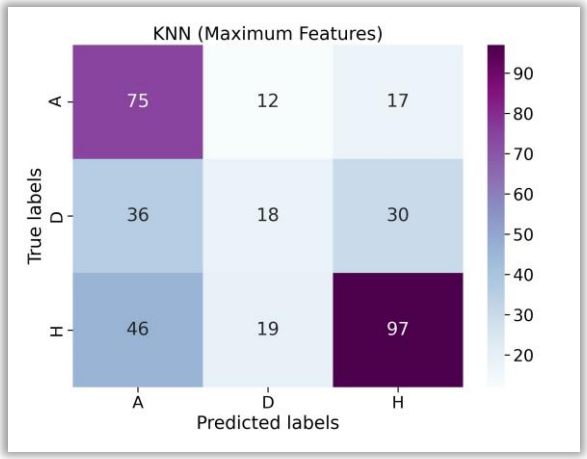




F1 Score (Weighted Average)	0.6	0.58

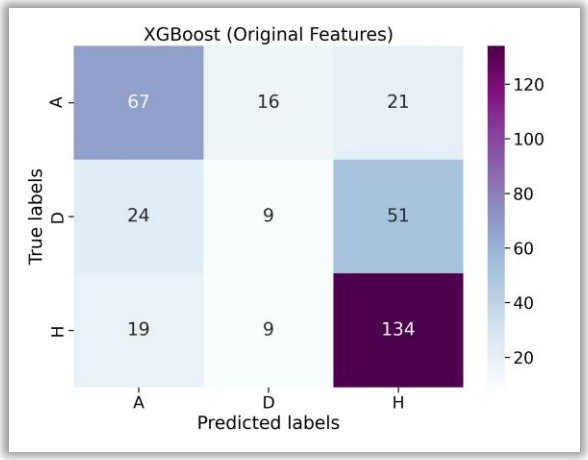
K-Nearest Neighbours

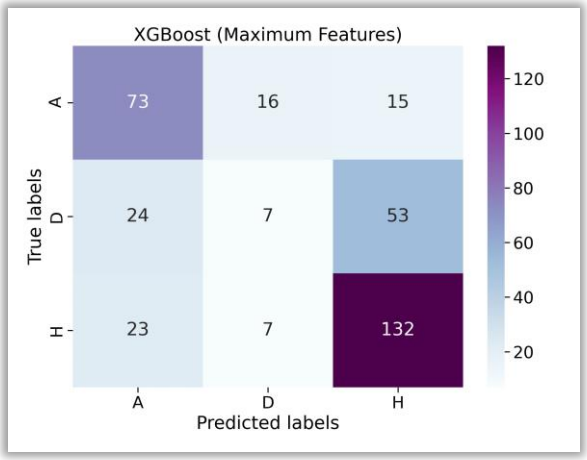




	Method 1 (Original Features)	**Method 2 (Max Features)**
Training Accuracy	68 %	67.4 %
Testing Accuracy	51.14 %	54.29 %
MCC	0.24	0.3
F1 Score (Weighted Average)	0.5	0.53

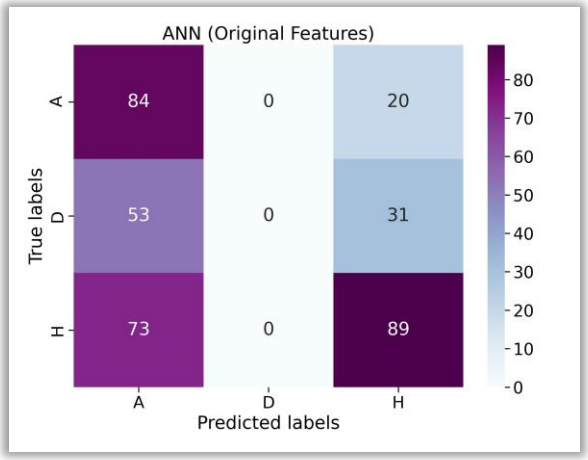
XGBoost

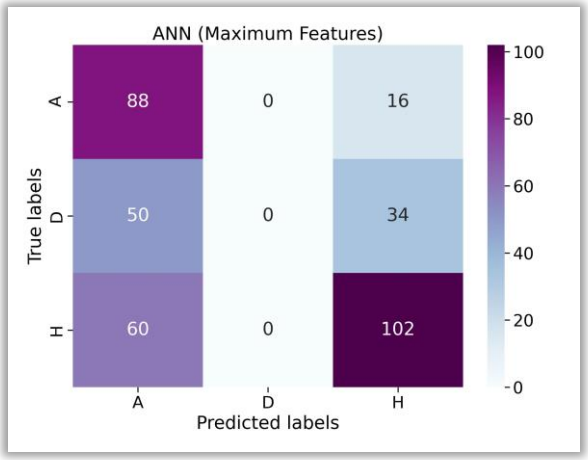




	Method 1 (Original Features)	**Method 2 (Max Features)**
Training Accuracy	100 %	100 %
Testing Accuracy	60 %	60.57 %
MCC	0.36	0.37
F1 Score (Weighted Average)	0.56	0.56

Artificial Neural Network



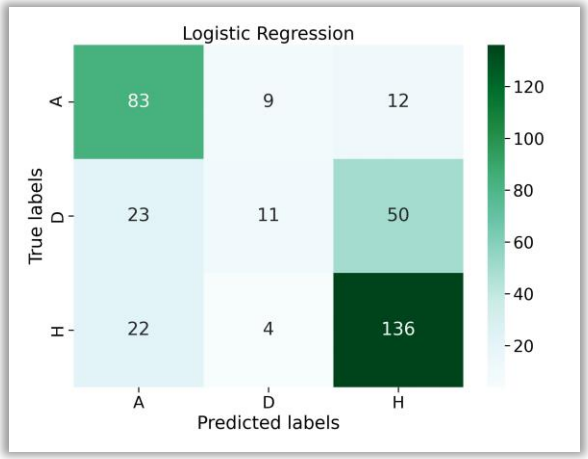


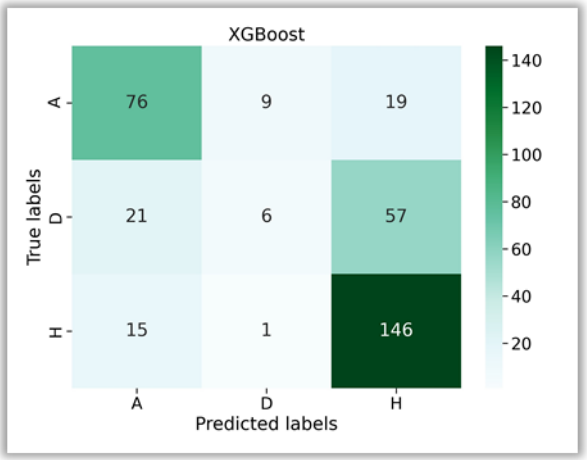
	Method 1 (Original Features)	**Method 2 (Max Features)**
Training Accuracy	54.32 %	54.13 %
Testing Accuracy	49.43 %	54.29 %
MCC	0.24	0.31
F1 Score (Weighted Average)	0.43	0.47

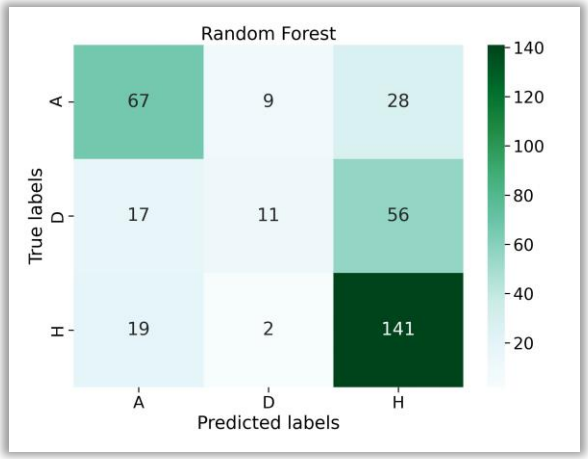
The above models were tested with their default hyperparameters and only Support Vector Machines performed better on the original dataset, so it is proof there is value inside the engineered features and further testing is necessary. Random Forest and XGBoost overfitted the training dataset and probably did not make reliable predictions on the unseen data which was the testing dataset.

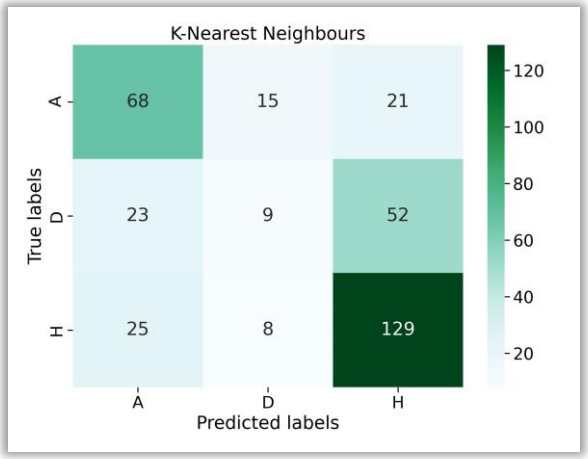
After the feature selection process discussed in Subsection 3.1, the size of the dataset was reduced to 42 features, more than three times less magnitude. Along with extensive hyperparameter tuning, this dataset managed to produce fascinating results that outperformed all the above models and are presented unitedly as follows.

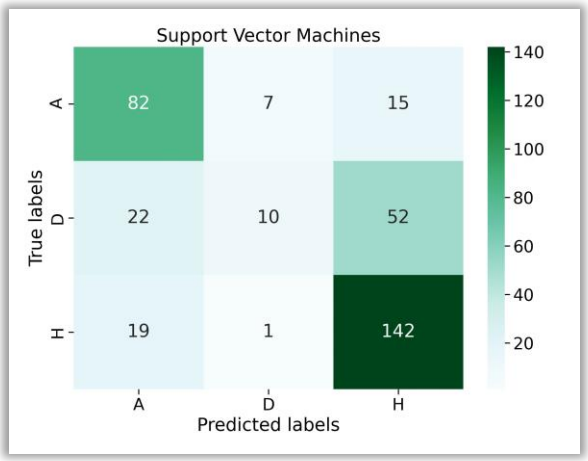
Confusion Matrices of Combined Models

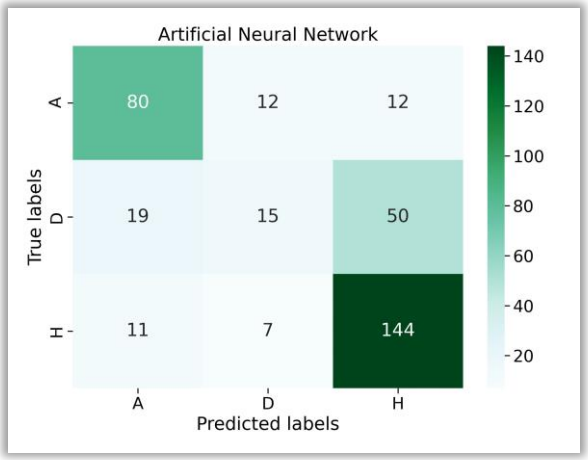












	LR	RF	SVM	KNN	XG Boost	ANN
Training Accuracy	67.37 %	100 %	67.43 %	66.03 %	67.65 %	67.43 %
Testing Accuracy	65.71 %	62.57 %	66.86 %	58.86 %	65.14 %	68.29 %
MCC	0.46	0.4	0.48	0.34	0.45	0.5
F1 Score (Weighted Average)	0.61	0.58	0.62	0.55	0.59	0.65

Artificial Neural Network was not only the best classifier, but it was also the most improved by the selected features and hyperparameter adjustments. Additionally, hyperparameter tuning cannot be praised enough as in the first two methods ANN did not manage to predict any draw result and ultimately predicted the most true positive draw values.

Conclusively, the following ranking table was produced on Google Sheets after exporting the predicted match result labels. Overall, the champion was predicted accurately along with five of the top 6 teams with minimal inconsistency on the exact position and two out of the last 3 teams. Crystal Palace was mistakenly relegated instead of Watford.

Ranking	Teams	Home Points	Away Points	Total Predicted	Actual Total
1	Liverpool	57	38	95	99
2	Manchester City	46	39	85	81
3	Manchester United	52	27	79	66
4	Tottenham Hotspur	50	28	78	59
5	Everton	43	22	65	49
6	Chelsea	43	21	64	66
7	Leicester City	35	29	64	62
8	Wolverhampton Wanderers	30	27	57	59
9	Arsenal	40	15	55	56
10	Burnley	33	22	55	54
11	West Ham United	32	20	52	39
12	Newcastle United	33	15	48	44
13	Sheffield United	32	16	48	54
14	Southampton	15	29	44	52
15	Brighton and Hove Albion	20	22	42	41
16	Watford	30	8	38	34
17	Aston Villa	27	9	36	35
18	Crystal Palace	25	9	34	43
19	AFC Bournemouth	21	8	29	34
20	Norwich City	23	4	27	21

4.2. Limitations

The project faced a series of challenges that some of them are explored in Subsection 1.4 and the rest is outlined below:

- Football is a very unpredictable sport with three possible outcomes as opposed to the majority of sports that demand either a winner or a loser. The project could not include features like injuries, weather conditions and pitch condition, history of past meetups between teams, rivalries like derbies and recruitment or dismissal of team's coach that could majorly influence a match result.
- Many of the original dataset's features are redundant due to the fact they constitute small variations of themselves. Increased data quality and availability has to be explored further.
- Limited time to complete and submit the Data Science project in addition to mitigating circumstances that arose, pushed back the work plan schedule and submission deadline.

5. Conclusion

5.1. Summary

In this project and specifically in Subsection 1.3 three research questions were set. This section reports the answers to these questions and concludes the project tasks.

Question 1: Can the selected classifiers go the extra mile and achieve consistent predicting accuracy of at least 60%?
The project achieved the target and it managed to overcome the expectations. The top model managed to classify accurately 68.2% of the testing dataset where the average of the six models reached 64.5%. However, there is a possibility of some data leakage as the feature engineering process was done on the whole dataset before splitting and further testing is required to confirm its validity.

Question 2: To what extent did the engineered features and their addition to the original dataset impacted the prediction accuracy?
Based on the results after feeding the models only with the original dataset and comparing them with the second method's results with the combined dataset, it is safe to say the engineered features did have a significant impact on the outcome as the majority of the classifiers improved their performance after their addition. On the other hand, after the feature selection process only 20% (7 out of 34) were selected from the engineered features list and 35% (35 out of 99) of the features from the original dataset and requires future testing and development.

Question 3: How well did the best scoring classifier predict the final ranking table?

The league table presented on Subsection 4.1 includes the team ranking based total points from the predicted match results of the best model which was the Artificial Neural Network. The first three fixtures were removed during data pre-processing, so these results were added manually to calculate the points of each team. On the last column of the table, the actual total team points are presented for comparison. In total, more than half of the teams were awarded points similar to the actual table and especially the top teams that rule most of the matches.

The champion Liverpool was correctly predicted as did Manchester City on the 2nd place. Although Manchester United finished 3rd in both, the ANN predictions resulted in awarding the team with 13 points more. The largest discrepancy was Everton placed seven positions higher amongst the top 5 teams with 16 points more and Tottenham Hotspur which was awarded 19 more points than the actual table. This probably indicates these teams dominated matches that did not result in a win with other non-measurable factors influencing the actual result. Crystal Palace was awarded 9 less points and was wrongly relegated instead of Watford. Another possible explanation for this is Crystal Palace is a part of the Premier League for many years and most of the time finishes in the bottom and has more experience and mentality on matches for relegation battle.

5.2. Future Work

The process of this project can be tested and developed further in various ways. Some of them are examined below:

- The majority of the engineered features was rejected by the feature selection methods. New features can be explored in the future like derby factor, injured franchise players, manager win streak, matches between league games (in European or domestic Cup competitions), team budgets. Relegated and promoted teams can be distinguished between each season and final position of each team in the previous year can be documented and tested for its relevance. Attack, Midfield, Defence and Overall ratings can be interactively changed based on the performance of each team every fixture in order to appropriately represent each team's current form and strengths.
- The dataset can be expanded year by year and the missing features and values should be traced. Alternative leagues can be also explored in order to increase training performance of the models and the accurate classification of a Draw as the result, that proved to be the hardest task. Individual player statistics can be considered if they influence in any way the prediction accuracy of the classifiers.
- The project can be taken to the next level by considering and testing more advanced techniques and models like deep learning. However, the volume of the data has to increase exponentially to feed the stacked neural networks. Also, additional metrics can be introduced to tackle the class imbalance problem mentioned in Subsection 2.2.

- A website may eventually be deployed to make predictions before the start of each season where fans can comment and share their thoughts while interacting with fascinating tabular data from previous seasons.
- Finally, another project that will be able to run in parallel can take advantage of the 19/20 season and its purpose will be to investigate the impact COVID had on the home advantage factor and how much influence the team's supporters have on the result of a match when they are present and not.

6. References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M. and Kudlur, M., 2016. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265-283).

Altman, N.S., 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3), pp.175–185.

Baboota, R. and Kaur, H., 2018. Predictive analysis and modelling football results using machine learning approach for English Premier League. *International Journal of Forecasting*, 35(2), pp.741-755.

Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32.

Brownlee, J., 2016. Machine learning mastery with python. *Machine Learning Mastery Pty Ltd*, *527*.

Chen, T., & Guestrin, C. (2016). XGBoost: a scalable tree boosting system. *In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785–794).

Chollet, F. & others, 2015. Keras. Available at: https://github.com/fchollet/keras.

Constantinou, A.C., Fenton, N.E. and Neil, M., 2012. pi-football: A Bayesian network model for forecasting Association Football match outcomes. *Knowledge-Based Systems*, 36, pp.322-339.

Cortes, C. & Vapnik, V., 1995. Support-vector networks. *Machine learning*, 20(3), pp.273–297.

Cox, D.R., 1958. The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, 20(2), pp.215–232.

Freitas, P., 2021, *All Premier League Matches 2010-2021 Dataset*. Available at: https://www.kaggle.com/pablohfreitas/all-premier-league-matches-20102021

Géron, A., 2019. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media.

Hackeling, G., 2017. *Mastering Machine Learning with scikit-learn*. Packt Publishing Ltd.

Harris, C.R., Millman, K.J., van der Walt, S.J. et al. *Array programming with NumPy*. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

Haykin, S., 1994. *Neural networks: a comprehensive foundation*, Prentice Hall PTR.

Hessels, J., 2019. Improving the prediction of soccer match results by means of Machine Learning. *Tilburg University, The Netherlands*.

Ho, T.K., 1995. Random decision forests. In Proceedings of 3rd international conference on document analysis and recognition. pp. 278–282.

Hunter, J.D., 2007. Matplotlib: A 2D graphics environment. *Computing in science & engineering*, 9(3), pp.90–95.

Igiri, C.P. and Nwachukwu, E.O., 2014. An improved prediction system for football a match result. *IOSR journal of Engineering*, 4(12), pp.12-20.

Kluyver, T. et al., 2016. Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt, eds. *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. pp. 87–90.

McCulloch, W.S. & Pitts, W., 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), pp.115–133

McKinney, W. (2010). Data structures for statistical computing in Python. *In Proceedings of the 9th python in science conference* (pp. 51–56).

Microsoft Corporation, 2018. Microsoft Word, Available at: https://products.office.com/word

Owramipur, F., Eskandarian, P. and Mozneb, F.S., 2013. Football result prediction with Bayesian network in Spanish League-Barcelona team. *International Journal of Computer Theory and Engineering*, 5(5), p.812.

Premier League, 2019. *Premier League global audience on the rise*. Available at: https://www.premierleague.com/news/1280062

Python Software Foundation. *Python Language Reference*, version 3.9. Available at: http://www.python.org

Raju, M.A., Mia, M.S., Sayed, M.A. and Uddin, M.R., 2020, December. Predicting the Outcome of English Premier League Matches using Machine Learning. In *2020 2nd International Conference on Sustainable Technologies for Industry 4.0* (STI) (pp. 1-6). IEEE.

Richardson, L., 2007. Beautiful soup documentation. *April*.

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011

Statistics & Data, 2020. *Most Popular Sports in the World – (1930/2020)*. Available at: https://statisticsanddata.org/most-popular-sports-in-the-world/

Tsakonas, A., Dounias, G., Shtovba, S. and Vivdyuk, V., 2003. Forecasting football match outcomes with Support Vector Machines. *Herald of Zhytomyr Engineering-Technological Institute*. 1: 181–186.

Ulmer, B., Fernandez, M. and Peterson, M., 2013. Predicting soccer match results in the English premier league *(Doctoral dissertation, Doctoral dissertation, Ph. D. dissertation, Stanford)*.

Van Rossum, G., 2020. *The Python Library Reference, release 3.8.2*, Python Software Foundation.

Waskom, M., Botvinnik, O., Gelbart, M., Ostblom, J., Hobson, P., Lukauskas, S., Gemperline, D.C., Augspurger, T., Halchenko, Y., Warmenhoven, J. and Cole, J.B., 2020. Seaborn: statistical data visualization.

Weidman, S., 2019. *Deep Learning from Scratch: Building with Python from First Principles*. " O'Reilly Media, Inc.".

7. Appendix

I. Dataset Features

Original Features

Features names of the original dataset and their respective descriptions are displayed below as presented by Freitas (2021) on Kaggle website.

Feature Names	**Description**
link_match	page link of match in Premier League Official website
season	match season
date	match date
home_team	home team
away_team	away team
result_full	match result
result_ht	match result in 1st time
home_clearances	home team clearances in the match
home_corners	home team corners in the match
home_fouls_conceded	home team fouls conceded in the match
home_offsides	home team offsides in the match
home_passes	home team passes in the match
home_possession	home team possession in the match
home_red_cards	home team red cards in the match
home_shots	home team shots in the match
home_shots_on_target	home team shots on target in the match
home_tackles	home team tackles in the match
home_touches	home team touches in the match
home_yellow_cards	home team yellow cards in the match
away_clearances	away team clearances in the match
away_corners	away team corners in the match
away_fouls_conceded	away team fouls conceded in the match
away_offsides	away team offsides in the match
away_passes	away team passes in the match
away_possession	away team possession in the match
away_red_cards	away team red cards in the match
away_shots	away team shots in the match
away_shots_on_target	away team shots on target in the match
away_tackles	away team tackles in the match
away_touches	away team touches in the match
away_yellow_cards	away team yellow cards in the match

goal_home_ft	home team goals scored in the match
goal_away_ft	away team goals scored in the match
sg_match_ft	goals difference between home team and away team in the match
goal_home_ht	home team goals scored in 1st time in the match
goal_away_ht	away team goals scored in 1st time in the match
sg_match_ht	goals difference between home team and away team in 1st time in the match
clearances_avg_H	average home team clearances in matches as host in the season; accumulated until the last match
corners_avg_H	average home team corners in matches as host in the season; accumulated until the last match
fouls_conceded_avg_H	average home team fouls conceded in matches as host in the season; accumulated until the last match
offsides_avg_H	average home team offsides in matches as host in the season; accumulated until the last match
passes_avg_H	average home team passes in matches as host in the season; accumulated until the last match
possession_avg_H	average home team possession in matches as host in the season; accumulated until the last match
red_cards_avg_H	average home team red cards in matches as host in the season; accumulated until the last match
shots_avg_H	average home team shots in matches as host in the season; accumulated until the last match
shots_on_target_avg_H	average home team shots on target in matches as host in the season; accumulated until the last match
tackles_avg_H	average home team tackles in matches as host in the season; accumulated until the last match
touches_avg_H	average home team touches in matches as host in the season; accumulated until the last match

yellow_cards_avg_H	average home team yellow cards in matches as host in the season; accumulated until the last match
goals_scored_ft_avg_H	average home team goals scored in matches as host in the season; accumulated until the last match
goals_conced_ft_avg_H	average home team goals conceded in matches as host in the season; accumulated until the last match
sg_match_ft_acum_H	goals difference in matches as host in the season; accumulated until the last match
goals_scored_ht_avg_H	average home team goals scored in 1st time in matches as host in the season; accumulated until the last match
goals_conced_ht_avg_H	average home team goals conceded in 1st time in matches as host in the season; accumulated until the last match
sg_match_ht_acum_H	goals difference in 1st time in matches as host in the season; accumulated until the last match
performance_acum_H	home team percentage of points gains in matches as host in the season; accumulated until the last match
clearances_avg_A	average away team clearances in matches as host in the season; accumulated until the last match
corners_avg_A	average away team corners in matches as host in the season; accumulated until the last match
fouls_conceded_avg_A	average away team fouls conceded in matches as host in the season; accumulated until the last match
offsides_avg_A	average away team offsides in matches as host in the season; accumulated until the last match
passes_avg_A	average away team passes in matches as host in the season; accumulated until the last match
possession_avg_A	average away team possession in matches as host in the season; accumulated until the last match

red_cards_avg_A	average away team red cards in matches as host in the season; accumulated until the last match
shots_avg_A	average away team shots in matches as host in the season; accumulated until the last match
shots_on_target_avg_A	average away team shots on target in matches as host in the season; accumulated until the last match
tackles_avg_A	average away team tackles in matches as host in the season; accumulated until the last match
touches_avg_A	average away team touches in matches as host in the season; accumulated until the last match
yellow_cards_avg_A	average away team yellow cards in matches as host in the season; accumulated until the last match
goals_scored_ft_avg_A	average away team goals scored in matches as host in the season; accumulated until the last match
goals_conced_ft_avg_A	average away team goals conceded in matches as host in the season; accumulated until the last match
sg_match_ft_acum_A	goals difference in matches as host in the season; accumulated until the last match
goals_scored_ht_avg_A	average away team goals scored in 1st time in matches as host in the season; accumulated until the last match
goals_conced_ht_avg_A	average away team goals conceded in 1st time in matches as host in the season; accumulated until the last match
sg_match_ht_acum_A	goals difference in 1st time in matches as host in the season; accumulated until the last match
performance_acum_A	away team percentage of points gains in matches as host in the season; accumulated until the last match
clearances_avg_home	average home team clearances in the season; accumulated until the last match
corners_avg_home	average home team corners in the season; accumulated until the last match

fouls_conceded_avg_home	average home team fouls conceded in the season; accumulated until the last match
offsides_avg_home	average home team offsides in the season; accumulated until the last match
passes_avg_home	average home team passes in the season; accumulated until the last match
possession_avg_home	average home team possession in the season; accumulated until the last match
red_cards_avg_home	average home team red cards in the season; accumulated until the last match
shots_avg_home	average home team shots in the season; accumulated until the last match
shots_on_target_avg_home	average home team shots on target in the season; accumulated until the last match
tackles_avg_home	average home team tackles in the season; accumulated until the last match
touches_avg_home	average home team touches in the season; accumulated until the last match
yellow_cards_avg_home	average home team yellow cards in the season; accumulated until the last match
goals_scored_ft_avg_home	average home team goals scored in the season; accumulated until the last match
goals_conced_ft_avg_home	average home team goals conceded in the season; accumulated until the last match
sg_match_ft_acum_home	goals difference in the season; accumulated until the last match
goals_scored_ht_avg_home	average home team goals scored in 1st time in the season; accumulated until the last match
goals_conced_ht_avg_home	average home team goals conceded in 1st time in the season; accumulated until the last match
sg_match_ht_acum_home	goals difference in 1st time in the season; accumulated until the last match
performance_acum_home	home team percentage of points gains in the season; accumulated until the last match
clearances_avg_away	average away team clearances in the season; accumulated until the last match
corners_avg_away	average away team corners in the season; accumulated until the last match
fouls_conceded_avg_away	average away team fouls conceded in the season; accumulated until the last match
offsides_avg_away	average away team offsides in the season; accumulated until the last match

passes_avg_away	average away team passes in the season; accumulated until the last match
possession_avg_away	average away team possession in the season; accumulated until the last match
red_cards_avg_away	average away team red cards in the season; accumulated until the last match
shots_avg_away	average away team shots in the season; accumulated until the last match
shots_on_target_avg_away	average away team shots on target in the season; accumulated until the last match
tackles_avg_away	average away team tackles in the season; accumulated until the last match
touches_avg_away	average away team touches in the season; accumulated until the last match
yellow_cards_avg_away	average away team yellow cards in the season; accumulated until the last match
goals_scored_ft_avg_away	average away team goals scored in the season; accumulated until the last match
goals_conced_ft_avg_away	average away team goals conceded in the season; accumulated until the last match
sg_match_ft_acum_away	goals difference in the season; accumulated until the last match
goals_scored_ht_avg_away	average away team goals scored in 1st time in the season; accumulated until the last match
goals_conced_ht_avg_away	average away team goals conceded in 1st time in the season; accumulated until the last match
sg_match_ht_acum_away	goals difference in 1st time in the season; accumulated until the last match
performance_acum_away	away team percentage of points gains in the season; accumulated until the last match

Engineered Features

Feature Names	**Description**
home_att_rating	FIFA index home attack rating
home_mid_rating	FIFA index home midfield rating
home_def_rating	FIFA index home defence rating
home_ovr_rating	FIFA index home overall rating
away_att_rating	FIFA index away attack rating

away_mid_rating	FIFA index away midfield rating
away_def_rating	FIFA index away defence rating
away_ovr_rating	FIFA index away overall rating
home_win_streak	Home team streak of wins until the last match
home_draw_streak	Home team streak of draws until the last match
home_lose_streak	Home team streak of losses until the last match
away_win_streak	Away team streak of wins until the last match
away_draw_streak	Away team streak of draws until the last match
away_lose_streak	Away team streak of losses until the last match
home_points_acum	Points accumulated at home by current home team until the last match
away_points_acum	Points accumulated away of home by current away team until the last match
home_team_total_points_acum	Total points accumulated for home team until last match
away_team_total_points_acum	Total points accumulated for away team until last match
point_difference	Point difference of home and away team until last match
home_goals_acum	Goals scored at home until the last match of home team
home_goals_against_acum	Goals conceived at home until the last match of home team
away_goals_acum	Goals scored away of home until the last match of away team
away_goals_against_acum	Goals conceived away of home until the last match of away team

home_team_total_goals_acum	Total goals scored by home team until last match
away_team_total_goals_acum	Total goals scored by away team until last match
goal_difference	Goals scored difference of home and away team until last match
home_team_total_goals_against_acum	Total goals conceived until last match of home team
away_team_total_goals_against_acum	Total goals conceived until last match of away team
goal_against_difference	Goals conceived difference of home and away team until last match
home_team_form_last_3	Home team form in last 3 matches; accumulated until last match
away_team_form_last_3	Away team form in last 3 matches; accumulated until last match
home_team_form_last_6	Home team form in last 6 matches; accumulated until last match
away_team_form_last_6	Away team form in last 6 matches; accumulated until last match
game_result	Target Variable (H, D, A)

II. Code

Web_Scraping.ipynb

```
def ratings_export(link):

    """ Extract Attack, Midfield, Defence & Overall team ratings from fifaindex.com website
    and return a dictionary with team names as keys and a list of ratings as each key value
    """


    import requests
    from bs4 import BeautifulSoup # might need to install it first: pip install BeautifulSoup4

    page = requests.get(link)
    soup = BeautifulSoup(page.content, 'html.parser')

    new_data1=[]
    new_data2=[]

    data1 = [element.text for element in soup.find_all(class_="badge badge-dark rating r2")] # extracting group:r2 of team ratings from HTML page
    j=0
    while j<len(data1):
        new_data1.append(data1[j:j+4])
        j+=4

    data2 = [element.text for element in soup.find_all(class_="badge badge-dark rating r3")] # extracting group:r3 of team ratings from HTML page
    i=0
    while i<len(data2):
        new_data2.append(data2[i:i+4])
        i+=4

    ranking_data = new_data1 + new_data2 # combining different lists of ratings to a single list

    team_data = [element.text for element in soup.find_all(class_="link-team")] # extracting team names from HTML page
    team_data = [v for i, v in enumerate(team_data) if i % 2 != 0] # removing not required data from the list
```

```
    finaldict = dict(zip(team_data, ranking_data)) # combining team names to their
respective ratings

    return finaldict


seasons = ['10/11', '11/12', '12/13', '13/14', '14/15', '15/16', '16/17', '17/18', '18/19',
'19/20']
season_links = ['https://www.fifaindex.com/teams/fifa11_7/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa12_9/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa13_10/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa14_13/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa15_14/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa16_73/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa17_173/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa18_278/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa19_353/?league=13&order=desc',
                'https://www.fifaindex.com/teams/fifa20_419/?league=13&order=desc',]


team_ratings_per_season = {}

for index, link in enumerate(season_links):
    team_ratings_per_season["%s" %seasons[index]] = ratings_export(link)


for k in sorted(team_ratings_per_season):
  print(k, ":")
  # check if all teams have 4 unique ratings
  print("Does each team have 4 unique ratings?", all(len(val)==4 for val in
team_ratings_per_season[k].values()))
  for key, value in team_ratings_per_season[k].items():
    if len(value) < 4:
      print(key, ":", value)
  print("No of teams:", len(team_ratings_per_season[k]))
  print('')


team_ratings_per_season['10/11']["Everton"] = ['80','79','78','78']
team_ratings_per_season['11/12']["Arsenal"] = ['84','82','80','81']
team_ratings_per_season['12/13']["Southampton"] = ['73','74','71','73']
team_ratings_per_season['12/13']["Arsenal"] = ['80','79','80','80']
```

```
team_ratings_per_season['13/14']["Tottenham Hotspur"] = ['82','79','78','81']
team_ratings_per_season['14/15']["Arsenal"] = ['81','80','80','80']
team_ratings_per_season['16/17']["Leicester City"] = ['79','78','76','78']
team_ratings_per_season['17/18']["Everton"] = ['77','80','79','79']
team_ratings_per_season['18/19']["Everton"] = ['80','79','78','79']
team_ratings_per_season['19/20']["Leicester City"] = ['77','78','79','78']

# Changing names of 2 teams to match premier_df dataset on a different Jupyter Notebook
team_ratings_per_season['17/18']["Brighton and Hove Albion"] = team_ratings_per_season['17/18'].pop('Brighton & Hove Albion')
team_ratings_per_season['18/19']["Brighton and Hove Albion"] = team_ratings_per_season['18/19'].pop('Brighton & Hove Albion')
team_ratings_per_season['19/20']["Brighton and Hove Albion"] = team_ratings_per_season['19/20'].pop('Brighton & Hove Albion')
team_ratings_per_season['15/16']["AFC Bournemouth"] = team_ratings_per_season['15/16'].pop('Bournemouth')
team_ratings_per_season['16/17']["AFC Bournemouth"] = team_ratings_per_season['16/17'].pop('Bournemouth')
team_ratings_per_season['17/18']["AFC Bournemouth"] = team_ratings_per_season['17/18'].pop('Bournemouth')
team_ratings_per_season['18/19']["AFC Bournemouth"] = team_ratings_per_season['18/19'].pop('Bournemouth')


for k in sorted(team_ratings_per_season):
  print(k, ":")
  # check if all teams have 4 unique ratings
  print("Does each team have 4 unique ratings?", all(len(val)==4 for val in team_ratings_per_season[k].values()))
  print("No of teams:", len(team_ratings_per_season[k]))
  print('')


%store team_ratings_per_season
```

Data_Expolration_&_Feature_Engineering.ipynb

```
import pandas as pd
pd.set_option("display.max_columns", None) # display all columns
```

```
premier_df = pd.read_csv('C:/Users/Christos/Desktop/Project/Dataset/df_full_premierleague.csv')

print("Dataset Dimensions:", premier_df.shape , "\n")
print("Types of dataset features:")
premier_df.dtypes.value_counts()


premier_df

# remove letter:M from match_id feature and converting it to int
premier_df['match_id'] = premier_df['match_id'].map(lambda x: x.lstrip('M')).astype(int)
premier_df.head()


team_names = premier_df['home_team'].unique()
print(sorted(team_names))


premier_df['season'].astype(str) # convert column:season from object to string type
premier_df = premier_df[premier_df.season != "20/21"] # remove 20/21 Covid season rows

print("Dataset Dimensions:", premier_df.shape , "\n")
print("Last 5 rows of the updated dataset:")
premier_df.tail()


pd.options.mode.chained_assignment = None
premier_df['home_team'].astype(str) # convert column:home_team from object to string type
premier_df['away_team'].astype(str) # convert column:away_team from object to string type

premier_df['result_full'].astype(str) # convert column:result_full from object to string type

premier_df['game_result'] = '0'

for i in range(len(premier_df['result_full'])):
    if premier_df['result_full'][i][0] > premier_df['result_full'][i][2]:
        #premier_df['game_result'][i] = premier_df['home_team'][i]
        premier_df['game_result'][i] = 'H'
```

```
    elif premier_df['result_full'][i][0] == premier_df['result_full'][i][2]:
        premier_df['game_result'][i] = 'D'
        #premier_df['game_result'][i] = 'D'
    else:
        #premier_df['game_result'][i] = premier_df['away_team'][i]
        premier_df['game_result'][i] = 'A'

premier_df


import matplotlib.pyplot as plt
import seaborn as sns


""" Distribution of Match Result per Season """

fig = plt.figure()
sns.set(rc={'figure.figsize':(20, 12)})
sns.set_context("notebook", font_scale=1.6, rc={"lines.linewidth": 2.5})
sns.set_palette("icefire")
ax =
premier_df.groupby(['season'])['game_result'].value_counts(sort=False).unstack().plot.b
ar()
ax.tick_params(axis='x', rotation=0)
ax.set_xlabel('Seasons')
ax.set_ylabel('Sum of Results')
ax.set_title('Distribution of Match Result per Season')
plt.show()
fig = ax.get_figure()
fig.savefig('C:/Users/Christos/Desktop/seasons_distribution.png', dpi=300)


""" Top 6 Teams at Home Wins """

fig = plt.figure()
sns.set(rc={'figure.figsize':(14, 8)})
sns.set_context("notebook", font_scale=1.6, rc={"lines.linewidth": 2.5})
sns.set_palette("mako")
ax = premier_df[premier_df['game_result'] ==
'H'].groupby(['game_result'])['home_team'].value_counts(sort=True).nlargest(6).unstac
k().plot.bar()
ax.tick_params(axis='x', rotation=0)
ax.set_xlabel('')
```

```
ax.set_xticks('')
ax.set_ylabel('Sum of Wins')
ax.set_title('Top 6 Teams at Home Wins')
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/home_wins.png', dpi=300)


""" Top 6 Teams at Away Wins """

fig = plt.figure()
sns.set(rc={'figure.figsize':(14, 8)})
sns.set_context("notebook", font_scale=1.6, rc={"lines.linewidth": 2.5})
sns.set_palette("mako")
ax = premier_df[premier_df['game_result'] ==
'A'].groupby(['game_result'])['away_team'].value_counts(sort=True).nlargest(6).unstac
k().plot.bar()
ax.tick_params(axis='x', rotation=0)
ax.set_xlabel('')
ax.set_xticks('')
ax.set_ylabel('Sum of Wins')
ax.set_title('Top 6 Teams at Away Wins')
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/away_wins.png', dpi=300)


""" Top 6 Teams at Home Losses """

fig = plt.figure()
sns.set(rc={'figure.figsize':(14, 8)})
sns.set_context("notebook", font_scale=1.6, rc={"lines.linewidth": 2.5})
sns.set_palette("YlOrRd_r")
ax = premier_df[premier_df['game_result'] ==
'A'].groupby(['game_result'])['home_team'].value_counts(sort=True).nlargest(6).unstac
k().plot.bar()
ax.tick_params(axis='x', rotation=0)
ax.set_xlabel('')
ax.set_xticks('')
ax.set_ylabel('Sum of Losses')
ax.set_title('Top 6 Teams at Home Losses')
fig = ax.get_figure()
fig.tight_layout()
```

```
fig.savefig('C:/Users/Christos/Desktop/home_losses.png', dpi=300)


""" Top 6 Teams at Away Losses """

fig = plt.figure()
sns.set(rc={'figure.figsize':(14, 8)})
sns.set_context("notebook", font_scale=1.6, rc={"lines.linewidth": 2.5})
sns.set_palette("YlOrRd_r")
ax = premier_df[premier_df['game_result'] == 'H'].groupby(['game_result'])['away_team'].value_counts(sort=True).nlargest(6).unstack().plot.bar()
ax.tick_params(axis='x', rotation=0)
ax.set_xlabel('')
ax.set_xticks('')
ax.set_ylabel('Sum of Losses')
ax.set_title('Top 6 Teams at Away Losses')
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/away_losses.png', dpi=300)


%store -r team_ratings_per_season


import copy

attack = copy.deepcopy(team_ratings_per_season)
midfield = copy.deepcopy(team_ratings_per_season)
defence = copy.deepcopy(team_ratings_per_season)
overall = copy.deepcopy(team_ratings_per_season)

for key, values in attack.items():
    for item, rating in values.items():
        values[item] = rating[0]
        attack[key] = values

for key, values in midfield.items():
    for item, rating in values.items():
        values[item] = rating[1]
        midfield[key] = values

for key, values in defence.items():
```

```
    for item, rating in values.items():
        values[item] = rating[2]
        defence[key] = values

for key, values in overall.items():
    for item, rating in values.items():
        values[item] = rating[3]
        overall[key] = values


print(attack['10/11'], "\n\n", midfield['10/11'], "\n\n", defence['10/11'], "\n\n",
overall['10/11'])


import itertools

premier_df['home_team'].astype(str) # convert column:home_team from object to string
type
premier_df['away_team'].astype(str) # convert column:away_team from object to string
type

# Attack, Midfield, Defence, Overall ratings for Home & Away teams
premier_df['home_att_rating'] = '0'
premier_df['home_mid_rating'] = '0'
premier_df['home_def_rating'] = '0'
premier_df['home_ovr_rating'] = '0'
premier_df['away_att_rating'] = '0'
premier_df['away_mid_rating'] = '0'
premier_df['away_def_rating'] = '0'
premier_df['away_ovr_rating'] = '0'


pd.options.mode.chained_assignment = None

seasons = ['10/11', '11/12', '12/13', '13/14', '14/15', '15/16', '16/17', '17/18', '18/19',
'19/20']

# create new dictionary to keep slices of premier_df as dataframes per season
pandas_per_season = {}
for x in seasons:
    pandas_per_season[x] = premier_df[premier_df['season'] == x]

# assign each rating to each column of each season
```

```
for season, season_pandas in pandas_per_season.items():
    season_pandas['home_att_rating'] =
season_pandas['home_team'].map(attack[season])
    season_pandas['home_mid_rating'] =
season_pandas['home_team'].map(midfield[season])
    season_pandas['home_def_rating'] =
season_pandas['home_team'].map(defence[season])
    season_pandas['home_ovr_rating'] =
season_pandas['home_team'].map(overall[season])
    season_pandas['away_att_rating'] =
season_pandas['away_team'].map(attack[season])
    season_pandas['away_mid_rating'] =
season_pandas['away_team'].map(midfield[season])
    season_pandas['away_def_rating'] =
season_pandas['away_team'].map(defence[season])
    season_pandas['away_ovr_rating'] =
season_pandas['away_team'].map(overall[season])

# combine datasets into one again
dataframes = [pandas_per_season['10/11'], pandas_per_season['11/12'],
pandas_per_season['12/13'],
              pandas_per_season['13/14'], pandas_per_season['14/15'],
pandas_per_season['15/16'],
              pandas_per_season['16/17'], pandas_per_season['17/18'],
pandas_per_season['18/19'],
              pandas_per_season['19/20']]

new_premier_df = pd.concat(dataframes)
new_premier_df


def home_streak_creator(streak_name, streak_result, target_team):
    # create 2 lists of all seasons and all teams per seasons
    pandas_per_season_list = []
    team_names_list = []

    for season, season_pandas in pandas_per_season.items():
        season_pandas['win_notwin'] = 0
        win_notwin_list = []
        count = 0
        for y, row in season_pandas.iterrows():
            if row['game_result'] == streak_result:
                count = 1
```

```
            else:
                count = 0
            win_notwin_list.append(count)
        season_pandas['win_notwin'] = win_notwin_list
        team_names = season_pandas[target_team].unique().tolist()
        team_names_list.append(team_names)
        pandas_per_season_list.append(season_pandas)

    all_seasons_ls = []
    for i in range(len(pandas_per_season_list)):
        season_ls = []
        for team in team_names_list[i]:
                df_copy                                                    =
pandas_per_season_list[i][pandas_per_season_list[i].home_team == team].copy()
                ls = []
                cumsum = 0
                for _, row in df_copy.iterrows():
                    if row[-1] == 1:
                        cumsum += 1
                    else:
                        cumsum = 0
                    ls.append(cumsum)
                df_copy[streak_name] = ls
                # shift values one row down as streak starts from 2nd fixture
                df_copy[streak_name] = df_copy[streak_name].shift(1).fillna(0).astype(int)
                season_ls.append(df_copy)

        single_season_df = pd.concat(season_ls, ignore_index=True)
        all_seasons_ls.append(single_season_df)

    all_seasons_df = pd.concat(all_seasons_ls, ignore_index=True)
    all_seasons_df = all_seasons_df.sort_values(['match_id']).reset_index(drop=True)

    return all_seasons_df[streak_name]



def away_streak_creator(streak_name, streak_result, target_team):
    # create 2 lists of all seasons and all teams per seasons
    pandas_per_season_list = []
    team_names_list = []

    for season, season_pandas in pandas_per_season.items():
```

```
        season_pandas['win_notwin'] = 0
        win_notwin_list = []
        count = 0
        for y, row in season_pandas.iterrows():
            if row['game_result'] == streak_result:
                count = 1
            else:
                count = 0
            win_notwin_list.append(count)
        season_pandas['win_notwin'] = win_notwin_list
        team_names = season_pandas[target_team].unique().tolist()
        team_names_list.append(team_names)
        pandas_per_season_list.append(season_pandas)

    all_seasons_ls = []
    for i in range(len(pandas_per_season_list)):
        season_ls = []
        for team in team_names_list[i]:
                df_copy                                                                    =
pandas_per_season_list[i][pandas_per_season_list[i].away_team == team].copy()
                ls = []
                cumsum = 0
                for _, row in df_copy.iterrows():
                    if row[-1] == 1:
                        cumsum += 1
                    else:
                        cumsum = 0
                    ls.append(cumsum)
                df_copy[streak_name] = ls
                # shift values one row down as streak starts from 2nd fixture
                df_copy[streak_name] = df_copy[streak_name].shift(1).fillna(0).astype(int)
                season_ls.append(df_copy)

        single_season_df = pd.concat(season_ls, ignore_index=True)
        all_seasons_ls.append(single_season_df)

    all_seasons_df = pd.concat(all_seasons_ls, ignore_index=True)
    all_seasons_df = all_seasons_df.sort_values(['match_id']).reset_index(drop=True)

    return all_seasons_df[streak_name]
```

```
new_premier_df['home_win_streak'] = home_streak_creator('home_win_streak', 'H',
'home_team')
new_premier_df['home_draw_streak'] = home_streak_creator('home_draw_streak',
'D', 'home_team')
new_premier_df['home_lose_streak'] = home_streak_creator('home_lose_streak', 'A',
'home_team')
new_premier_df['away_win_streak'] = away_streak_creator('away_win_streak', 'A',
'away_team')
new_premier_df['away_draw_streak'] = away_streak_creator('away_draw_streak',
'D', 'away_team')
new_premier_df['away_lose_streak'] = away_streak_creator('away_lose_streak', 'H',
'away_team')


premier_df = new_premier_df

pandas_per_season = {}
for x in seasons:
    pandas_per_season[x] = premier_df[premier_df['season'] == x]

premier_df


premier_df_per_season = []

for season, season_pandas in pandas_per_season.items():
    premier_df_per_season.append(season_pandas)


def home_points_acum_creator(df):
    df['home_points_acum'] = 0

    home_team_points = {}
    for x in range(len(df['game_result'])):
        if df["home_team"].iloc[x] in home_team_points:
            team1 = home_team_points[df["home_team"].iloc[x]]
            if df['game_result'].iloc[x] == 'H':
                team1.append(team1[-1] + 3)
                home_team_points[df["home_team"].iloc[x]] = team1
            elif df['game_result'].iloc[x] == 'A':
                team1.append(team1[-1])
                home_team_points[df["home_team"].iloc[x]] = team1
            else:
```

```
                team1.append(team1[-1] + 1)
                home_team_points[df["home_team"].iloc[x]] = team1
        else:
            team1 = [0]
            if df['game_result'].iloc[x] == 'H':
                team1.append(3)
                home_team_points[df["home_team"].iloc[x]] = team1
            elif df['game_result'].iloc[x] == 'A':
                team1.append(0)
                home_team_points[df["home_team"].iloc[x]] = team1
            else:
                team1.append(1)
                home_team_points[df["home_team"].iloc[x]] = team1


    for team in home_team_points:
        count = 0
        for y in range(len(df['game_result'])):
            if df['home_team'].iloc[y] == team:
                df['home_points_acum'].iloc[y] = home_team_points[team][count]
                count += 1

    return df


for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] = home_points_acum_creator(premier_df_per_season[i])


def away_points_acum_creator(df):
    df['away_points_acum'] = 0

    away_team_points = {}
    for x in range(len(df['game_result'])):
        if df["away_team"].iloc[x] in away_team_points:
            team1 = away_team_points[df["away_team"].iloc[x]]
            if df['game_result'].iloc[x] == 'H':
                team1.append(team1[-1])
                away_team_points[df["away_team"].iloc[x]] = team1
            elif df['game_result'].iloc[x] == 'A':
                team1.append(team1[-1] + 3)
                away_team_points[df["away_team"].iloc[x]] = team1
            else:
```

```
                team1.append(team1[-1] + 1)
                away_team_points[df["away_team"].iloc[x]] = team1
        else:
            team1 = [0]
            if df['game_result'].iloc[x] == 'H':
                team1.append(0)
                away_team_points[df["away_team"].iloc[x]] = team1
            elif df['game_result'].iloc[x] == 'A':
                team1.append(3)
                away_team_points[df["away_team"].iloc[x]] = team1
            else:
                team1.append(1)
                away_team_points[df["away_team"].iloc[x]] = team1


    for team in away_team_points:
        count = 0
        for y in range(len(df['game_result'])):
            if df['away_team'].iloc[y] == team:
                df['away_points_acum'].iloc[y] = away_team_points[team][count]
                count += 1

    return df


for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] = away_points_acum_creator(premier_df_per_season[i])


def total_points_acum_creator(df):
    df['home_team_total_points_acum'] = 0
    df['away_team_total_points_acum'] = 0
    df['point_difference'] = 0

    total_points_dict = {}
    for x in range(len(df["game_result"])):
        if all(k in total_points_dict for k in (df["home_team"].iloc[x],
df["away_team"].iloc[x])):
            team1 = total_points_dict[df["home_team"].iloc[x]]
            team2 = total_points_dict[df["away_team"].iloc[x]]

            if df["game_result"].iloc[x] == 'H':
                team1.append(team1[-1] + 3)
```

```
                team2.append(team2[-1])
                total_points_dict[df["home_team"].iloc[x]] = team1
                total_points_dict[df["away_team"].iloc[x]] = team2
            elif df["game_result"].iloc[x] == 'A':
                team1.append(team1[-1])
                team2.append(team2[-1] + 3)
                total_points_dict[df["home_team"].iloc[x]] = team1
                total_points_dict[df["away_team"].iloc[x]] = team2
            else:
                team1.append(team1[-1] + 1)
                team2.append(team2[-1] + 1)
                total_points_dict[df["home_team"].iloc[x]] = team1
                total_points_dict[df["away_team"].iloc[x]] = team2
        else:
            team1 = [0]
            team2 = [0]

            if df['game_result'].iloc[x] == 'H':
                team1.append(3)
                team2.append(0)
                total_points_dict[df["home_team"].iloc[x]] = team1
                total_points_dict[df["away_team"].iloc[x]] = team2
            elif df['game_result'].iloc[x] == 'A':
                team1.append(0)
                team2.append(3)
                total_points_dict[df["home_team"].iloc[x]] = team1
                total_points_dict[df["away_team"].iloc[x]] = team2
            else:
                team1.append(1)
                team2.append(1)
                total_points_dict[df["home_team"].iloc[x]] = team1
                total_points_dict[df["away_team"].iloc[x]] = team2

    for team in total_points_dict:
        count = 0
        for y in range(len(df["game_result"])):
            if df['home_team'].iloc[y] == team:
                df['home_team_total_points_acum'].iloc[y]                    =
total_points_dict[team][count]
                count += 1
            elif df['away_team'].iloc[y] == team:
                df['away_team_total_points_acum'].iloc[y]                    =
total_points_dict[team][count]
```

```
            count += 1
    df['point_difference'] = df['away_team_total_points_acum'] -
df['home_team_total_points_acum']

    return df



for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] = total_points_acum_creator(premier_df_per_season[i])


premier_df_per_season[0][['home_team', 'away_team', 'result_full',
'home_points_acum', 'home_team_total_points_acum',
'away_team_total_points_acum', 'point_difference']].head(20)


def home_goals_acum_creator(df):
    df['home_goals_acum'] = 0

    home_team_goals = {}
    for x in range(len(df['game_result'])):
        if df["home_team"].iloc[x] in home_team_goals:
            team1 = home_team_goals[df["home_team"].iloc[x]]
            team1.append(team1[-1] + int(df['result_full'].iloc[x][0]))

            home_team_goals[df["home_team"].iloc[x]] = team1

        else:
            team1 = [0]
            team1.append(int(df['result_full'].iloc[x][0]))

            home_team_goals[df["home_team"].iloc[x]] = team1

    for team in home_team_goals:
        count = 0
        for y in range(len(df['game_result'])):
            if df['home_team'].iloc[y] == team:
                df['home_goals_acum'].iloc[y] = home_team_goals[team][count]
                count += 1

    return df
```

```
for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] = home_goals_acum_creator(premier_df_per_season[i])


def home_goals_against_acum_creator(df):
    df['home_goals_against_acum'] = 0

    home_team_goals_against = {}
    for x in range(len(df['game_result'])):
        if df["home_team"].iloc[x] in home_team_goals_against:
            team1 = home_team_goals_against[df["home_team"].iloc[x]]
            team1.append(team1[-1] + int(df['result_full'].iloc[x][2]))

            home_team_goals_against[df["home_team"].iloc[x]] = team1

        else:
            team1 = [0]
            team1.append(int(df['result_full'].iloc[x][2]))

            home_team_goals_against[df["home_team"].iloc[x]] = team1

    for team in home_team_goals_against:
        count = 0
        for y in range(len(df['game_result'])):
            if df['home_team'].iloc[y] == team:
                df['home_goals_against_acum'].iloc[y]                    =
home_team_goals_against[team][count]
                count += 1

    return df


for i in range(len(premier_df_per_season)):
    premier_df_per_season[i]                                             =
home_goals_against_acum_creator(premier_df_per_season[i])


def away_goals_acum_creator(df):
    df['away_goals_acum'] = 0

    away_team_goals = {}
```

```
    for x in range(len(df['game_result'])):
        if df["away_team"].iloc[x] in away_team_goals:
            team2 = away_team_goals[df["away_team"].iloc[x]]
            team2.append(team2[-1] + int(df['result_full'].iloc[x][2]))

            away_team_goals[df["away_team"].iloc[x]] = team2

        else:
            team2 = [0]
            team2.append(int(df['result_full'].iloc[x][2]))

            away_team_goals[df["away_team"].iloc[x]] = team2

    for team in away_team_goals:
        count = 0
        for y in range(len(df['game_result'])):
            if df['away_team'].iloc[y] == team:
                df['away_goals_acum'].iloc[y] = away_team_goals[team][count]
                count += 1

    return df


for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] = away_goals_acum_creator(premier_df_per_season[i])


def away_goals_against_acum_creator(df):
    df['away_goals_against_acum'] = 0

    away_team_goals_against = {}
    for x in range(len(df['game_result'])):
        if df["away_team"].iloc[x] in away_team_goals_against:
            team2 = away_team_goals_against[df["away_team"].iloc[x]]
            team2.append(team2[-1] + int(df['result_full'].iloc[x][0]))

            away_team_goals_against[df["away_team"].iloc[x]] = team2

        else:
            team2 = [0]
            team2.append(int(df['result_full'].iloc[x][0]))

            away_team_goals_against[df["away_team"].iloc[x]] = team2
```

```
    for team in away_team_goals_against:
        count = 0
        for y in range(len(df['game_result'])):
            if df['away_team'].iloc[y] == team:
                df['away_goals_against_acum'].iloc[y] =
away_team_goals_against[team][count]
                count += 1

    return df


for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] =
away_goals_against_acum_creator(premier_df_per_season[i])


def total_goals_acum_creator(df):
    df['home_team_total_goals_acum'] = 0
    df['away_team_total_goals_acum'] = 0
    df['goal_difference'] = 0

    total_goals_dict = {}
    for x in range(len(df["game_result"])):
        if all(k in total_goals_dict for k in (df["home_team"].iloc[x],
df["away_team"].iloc[x])):
            team1 = total_goals_dict[df["home_team"].iloc[x]]
            team2 = total_goals_dict[df["away_team"].iloc[x]]

            team1.append(team1[-1] + int(df['result_full'].iloc[x][0]))
            team2.append(team2[-1] + int(df['result_full'].iloc[x][2]))
            total_goals_dict[df["home_team"].iloc[x]] = team1
            total_goals_dict[df["away_team"].iloc[x]] = team2
        else:
            team1 = [0]
            team2 = [0]

            team1.append(int(df['result_full'].iloc[x][0]))
            team2.append(int(df['result_full'].iloc[x][2]))
            total_goals_dict[df["home_team"].iloc[x]] = team1
            total_goals_dict[df["away_team"].iloc[x]] = team2

    for team in total_goals_dict:
```

```
        count = 0
        for y in range(len(df["game_result"])):
            if df['home_team'].iloc[y] == team:
                df['home_team_total_goals_acum'].iloc[y] = total_goals_dict[team][count]
                count += 1
            elif df['away_team'].iloc[y] == team:
                df['away_team_total_goals_acum'].iloc[y] = total_goals_dict[team][count]
                count += 1

    df['goal_difference'] = df['away_team_total_goals_acum'] - df['home_team_total_goals_acum']

    return df


for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] = total_goals_acum_creator(premier_df_per_season[i])


def total_goals_against_acum_creator(df):
    df['home_team_total_goals_against_acum'] = 0
    df['away_team_total_goals_against_acum'] = 0
    df['goal_against_difference'] = 0

    total_goals_against_dict = {}
    for x in range(len(df["game_result"])):
        if all(k in total_goals_against_dict for k in (df["home_team"].iloc[x], df["away_team"].iloc[x])):
            team1 = total_goals_against_dict[df["home_team"].iloc[x]]
            team2 = total_goals_against_dict[df["away_team"].iloc[x]]

            team1.append(team1[-1] + int(df['result_full'].iloc[x][2]))
            team2.append(team2[-1] + int(df['result_full'].iloc[x][0]))
            total_goals_against_dict[df["home_team"].iloc[x]] = team1
            total_goals_against_dict[df["away_team"].iloc[x]] = team2
        else:
            team1 = [0]
            team2 = [0]

            team1.append(int(df['result_full'].iloc[x][2]))
            team2.append(int(df['result_full'].iloc[x][0]))
            total_goals_against_dict[df["home_team"].iloc[x]] = team1
            total_goals_against_dict[df["away_team"].iloc[x]] = team2
```

```
    for team in total_goals_against_dict:
        count = 0
        for y in range(len(df["game_result"])):
            if df['home_team'].iloc[y] == team:
                df['home_team_total_goals_against_acum'].iloc[y] =
total_goals_against_dict[team][count]
                count += 1
            elif df['away_team'].iloc[y] == team:
                df['away_team_total_goals_against_acum'].iloc[y] =
total_goals_against_dict[team][count]
                count += 1

    df['goal_against_difference'] = df['away_team_total_goals_against_acum'] -
df['home_team_total_goals_against_acum']

    return df


for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] =
total_goals_against_acum_creator(premier_df_per_season[i])


premier_df_per_season[0][['home_team', 'away_team', 'result_full',
'home_goals_against_acum', 'away_goals_against_acum',
'home_team_total_goals_against_acum', 'away_team_total_goals_against_acum',
'goal_against_difference']].tail(50)


def form_creator(df):
    import copy

    form_dict = {}
    for i in range(len(df["game_result"])):
        if all(k in form_dict for k in (df["home_team"].iloc[i], df["away_team"].iloc[i])):
            team1 = form_dict[df["home_team"].iloc[i]]
            team2 = form_dict[df["away_team"].iloc[i]]

            if df["game_result"].iloc[i] == 'H':
                team1.append('3' + team1[-1])
                team2.append('0' + team2[-1])
                form_dict[df["home_team"].iloc[i]] = team1
```

```
            form_dict[df["away_team"].iloc[i]] = team2
        elif df["game_result"].iloc[i] == 'A':
            team1.append('0' + team1[-1])
            team2.append('3' + team2[-1])
            form_dict[df["home_team"].iloc[i]] = team1
            form_dict[df["away_team"].iloc[i]] = team2
        else:
            team1.append('1' + team1[-1])
            team2.append('1' + team2[-1])
            form_dict[df["home_team"].iloc[i]] = team1
            form_dict[df["away_team"].iloc[i]] = team2

    else:
        team1 = ['0']
        team2 = ['0']

        if df["game_result"].iloc[i] == 'H':
            team1.append('3')
            team2.append('0')
            form_dict[df["home_team"].iloc[i]] = team1
            form_dict[df["away_team"].iloc[i]] = team2
        elif df["game_result"].iloc[i] == 'A':
            team1.append('0')
            team2.append('3')
            form_dict[df["home_team"].iloc[i]] = team1
            form_dict[df["away_team"].iloc[i]] = team2
        else:
            team1.append('1')
            team2.append('1')
            form_dict[df["home_team"].iloc[i]] = team1
            form_dict[df["away_team"].iloc[i]] = team2



form_3_dict = copy.deepcopy(form_dict)
form_6_dict = copy.deepcopy(form_dict)
last_3 = {}
last_6 = {}

for key in form_3_dict:
    form_3_dict[key] = [elem[:3] for elem in form_3_dict[key]]
    last_3_list = []
    for item in form_3_dict[key][0:2]:
```

```
            last_3_list.append(round((int(item[0]) * 1.2), 2))
            last_3[key] = last_3_list
        last_3_list.append(round((int(form_3_dict[key][2][0]) * 1.2) +
(int(form_3_dict[key][2][1]) * 0.8), 2))
        last_3[key] = last_3_list
        for item in form_3_dict[key][3:]:
            last_3_list.append(round((int(item[0]) * 1.2) + (int(item[1]) * 0.8) +
(int(item[2]) * 0.4), 2))
            last_3[key] = last_3_list

    for key in form_6_dict:
        form_6_dict[key] = [elem[:6] for elem in form_6_dict[key]]
        last_6_list = []
        for item in form_6_dict[key][0:2]:
            last_6_list.append(round((int(item[0]) * 1.2), 2))
            last_6[key] = last_6_list
        last_6_list.append(round((int(form_6_dict[key][2][0]) * 1.2) +
(int(form_6_dict[key][2][1]) * 1), 2))
        last_6[key] = last_6_list

        last_6_list.append(round((int(form_6_dict[key][3][0]) * 1.2) +
(int(form_6_dict[key][3][1]) * 1) +
                           (int(form_6_dict[key][3][2]) * 0.8), 2))
        last_6[key] = last_6_list

        last_6_list.append(round((int(form_6_dict[key][4][0]) * 1.2) +
(int(form_6_dict[key][4][1]) * 1) +
                           (int(form_6_dict[key][4][2]) * 0.8) + (int(form_6_dict[key][4][3])
* 0.6), 2))
        last_6[key] = last_6_list

        last_6_list.append(round((int(form_6_dict[key][5][0]) * 1.2) +
(int(form_6_dict[key][5][1]) * 1) +
                           (int(form_6_dict[key][5][2]) * 0.8) + (int(form_6_dict[key][5][3])
* 0.6) +
                           (int(form_6_dict[key][5][4]) * 0.4), 2))
        last_6[key] = last_6_list

        for item in form_6_dict[key][6:]:
            last_6_list.append(round((int(item[0]) * 1.2) + (int(item[1]) * 1) + (int(item[2])
* 0.8) +
                               (int(item[3]) * 0.6) + (int(item[4]) * 0.4) + (int(item[5]) * 0.2),
2))
```

```
            last_6[key] = last_6_list



    df['home_team_form_last_3'] = 0
    df['away_team_form_last_3'] = 0
    df['home_team_form_last_6'] = 0
    df['away_team_form_last_6'] = 0

    for team in last_3:
            count = 0
            for y in range(len(df["game_result"])):
                if df['home_team'].iloc[y] == team:
                    df['home_team_form_last_3'].iloc[y] = last_3[team][count]
                    count += 1
                elif df['away_team'].iloc[y] == team:
                    df['away_team_form_last_3'].iloc[y] = last_3[team][count]
                    count += 1

    for team in last_6:
            count = 0
            for y in range(len(df["game_result"])):
                if df['home_team'].iloc[y] == team:
                    df['home_team_form_last_6'].iloc[y] = last_6[team][count]
                    count += 1
                elif df['away_team'].iloc[y] == team:
                    df['away_team_form_last_6'].iloc[y] = last_6[team][count]
                    count += 1

    return df


for i in range(len(premier_df_per_season)):
    premier_df_per_season[i] = form_creator(premier_df_per_season[i])


premier_df_per_season[0][['home_team', 'away_team', 'result_full',
'home_team_form_last_3', 'away_team_form_last_3', 'home_team_form_last_6',
'away_team_form_last_6']].head(50)


final_premier_df = pd.concat(premier_df_per_season)
final_premier_df.head()
```

```
final_premier_df.tail(10)


game_result_df = final_premier_df.pop('game_result')
final_premier_df['game_result'] = game_result_df

final_premier_df.head()


print('Initial features: 114 \nFinal features:', final_premier_df.shape[1])
print('New features created:', final_premier_df.shape[1] - 114)


season_list = ['10/11', '11/12', '12/13', '13/14', '14/15', '15/16', '16/17', '17/18', '18/19',
'19/20']
for i in season_list:
    row_index = (final_premier_df.season.values == i).argmax()
    final_premier_df                                                    =
final_premier_df.drop(final_premier_df.index[row_index:row_index+30])

print("Dataset Dimensions:", final_premier_df.shape , "\n")
final_premier_df.head()


final_premier_df.to_csv('C:/Users/Christos/Desktop/final_premier_league_dataset.cs
v', index=False)
```

Machine_Learning_Algorithms - Final Test - Feature Selection.ipynb

```
import pandas as pd

pd.set_option("display.max_columns", None) # display all columns

premier_df                                                              =
pd.read_csv('C:/Users/Christos/Desktop/Project/Dataset/final_premier_league_datas
et.csv')

print("Dataset Dimensions:", premier_df.shape , "\n")


premier_df.head()
```

```
print("Types of dataset features:")
premier_df.dtypes.value_counts()


# features that indicate the winner of each row
premier_df.drop(['match_id', 'link_match', 'home_team', 'away_team', 'date',
'result_full', 'result_ht', 'goal_home_ft',
                 'goal_away_ft', 'goal_home_ht', 'goal_away_ht', 'sg_match_ft',
'sg_match_ht'], axis=1, inplace=True)

premier_df.head()


count = 0
for column in premier_df.isna().sum():
    count += column

print(count, "NaN values exist within the dataset.")


column_means = premier_df.mean()
premier_df.fillna(column_means, inplace=True)

print("Are there any NaN values in the dataset?")
print(premier_df.isnull().values.any())


print('Before Labelling:')
premier_df['game_result']


from sklearn.preprocessing import LabelEncoder

labelencoder = LabelEncoder()

premier_df['game_result'] = labelencoder.fit_transform(premier_df['game_result'])


print('Definitions: \nH -> 2 \nD -> 1 \nA -> 0')
print('\nAfter Labelling:')
premier_df['game_result']
```

```
pd.options.mode.chained_assignment = None

X_train = premier_df[premier_df['season'] != '19/20']
X_train.drop(['season'], axis=1, inplace=True)

X_train


X_test = premier_df[premier_df['season'] == '19/20']
X_test.drop(['season'], axis=1, inplace=True)

X_test


y_train = X_train.pop('game_result')
y_test = X_test.pop('game_result')


print("X_train Size:", X_train.shape, "\nX_test Size:", X_test.shape, "\n")
print("y_train Size:", y_train.shape, "\ny_test Size:", y_test.shape)


from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
# fit scaler on train & test data
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import SelectFromModel

# Setting & training the model
model = RandomForestClassifier(n_estimators=300, random_state=0, n_jobs=-1)
model.fit(X_train, y_train)
# applying feature selection based on criteria
feat_sel = SelectFromModel(clf, threshold=0.008)

# fit on the train dataset & transform train and test dataset
feat_sel.fit(X_train, y_train)
X_train = feat_sel.transform(X_train)
```

```
X_test = feat_sel.transform(X_test)

print(X_train.shape, X_test.shape)


selected_features = premier_df.drop(['season', 'game_result'], axis=1)
feat_importances = pd.Series(model.feature_importances_,
index=selected_features.columns)

for i in range(len(feat_importances)):
    if feat_importances[i] > 0.008:
        print(feat_importances.index[i])




from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from xgboost import XGBClassifier
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
from tensorflow import random
from numpy.random import seed
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score,
matthews_corrcoef
import matplotlib.pyplot as plt
import seaborn as sns


grid = {'multi_class': ['multinomial', 'ovr'],
        'solver': ['newton-cg', 'lbfgs'],
        'C': [100, 100, 10, 1.0, 0.1, 0.01],
        'penalty': ['l2']}
model = LogisticRegression()
grid_search = GridSearchCV(model, grid, n_jobs=-1, scoring = 'f1_weighted')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
model = LogisticRegression(multi_class='multinomial', solver='newton-cg', C=10, penalty='l2', random_state=1)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Metrics
matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("\nClassification Report: \n", report)
print("Train Accuracy: %.2f %%" % (accuracy_score(y_train, model.predict(X_train))*100))
print("Test Accuracy: %.2f %%" % (accuracy_score(y_test, y_pred)*100))
print("MCC: %.2f " % (matthews_corrcoef(y_test, y_pred)))

# Plotting the confusion matrix
fig = plt.figure(figsize=(9, 7))
ax = plt.subplot()
sns.set_context("notebook", font_scale=1.7, rc={"lines.linewidth": 2.8})
sns.heatmap(matrix, cmap="BuGn", annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Logistic Regression');
ax.xaxis.set_ticklabels(['A', 'D', 'H']);
ax.yaxis.set_ticklabels(['A', 'D', 'H']);
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/LR.jpeg', dpi=300)


grid = {'n_estimators': [100, 200, 300, 400, 500, 600, 800, 1000],
        'max_features': ['auto', 'sqrt'],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]}
model = RandomForestClassifier()
grid_search = GridSearchCV(model, grid, n_jobs=-1, scoring = 'f1_weighted')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)


model = RandomForestClassifier(max_features='sqrt', min_samples_leaf=1, min_samples_split=2, n_estimators=100, random_state=2)
```

```
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Metrics
matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Classification Report: \n", report)
print("Train Accuracy: %.2f %%" % (accuracy_score(y_train,
model.predict(X_train))*100))
print("Test Accuracy: %.2f %%" % (accuracy_score(y_test, y_pred)*100))
print("MCC: %.2f " % (matthews_corrcoef(y_test, y_pred)))

# Plotting the confusion matrix
fig = plt.figure(figsize=(9, 7))
ax = plt.subplot()
sns.set_context("notebook", font_scale=1.7, rc={"lines.linewidth": 2.8})
sns.heatmap(matrix, cmap="BuGn", annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Random Forest');
ax.xaxis.set_ticklabels(['A', 'D', 'H']);
ax.yaxis.set_ticklabels(['A', 'D', 'H']);
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/RF.jpeg', dpi=300)


grid = {'C': [0.01, 0.01, 0.1, 1, 10, 100],
        'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
        'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}
model = SVC()
grid_search = GridSearchCV(model, grid, n_jobs=-1, scoring = 'f1_weighted')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)


model=SVC(kernel='rbf', C=100, gamma=0.01, random_state=3)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

```
# Metrics
matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Classification Report: \n", report)
print("Train Accuracy: %.2f %%" % (accuracy_score(y_train,
model.predict(X_train))*100))
print("Test Accuracy: %.2f %%" % (accuracy_score(y_test, y_pred)*100))
print("MCC: %.2f " % (matthews_corrcoef(y_test, y_pred)))

# Plotting the confusion matrix
fig = plt.figure(figsize=(9, 7))
ax = plt.subplot()
sns.set_context("notebook", font_scale=1.7, rc={"lines.linewidth": 2.8})
sns.heatmap(matrix, cmap="BuGn", annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Support Vector Machines');
ax.xaxis.set_ticklabels(['A', 'D', 'H']);
ax.yaxis.set_ticklabels(['A', 'D', 'H']);
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/SVM.jpeg', dpi=300)


grid = {'leaf_size': [1, 2, 4, 8, 16],
        'n_neighbors': [2, 4, 8, 12, 16, 20, 24],
        'weights': ['uniform']}
model = KNeighborsClassifier()
grid_search = GridSearchCV(model, grid, n_jobs=-1, scoring = 'f1_weighted')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)


odel = KNeighborsClassifier(leaf_size=1, n_neighbors=16, weights='uniform')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Metrics
matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Classification Report: \n", report)
```

```
print("Train Accuracy: %.2f %%" % (accuracy_score(y_train, model.predict(X_train))*100))
print("Test Accuracy: %.2f %%" % (accuracy_score(y_test, y_pred)*100))
print("MCC: %.2f " % (matthews_corrcoef(y_test, y_pred)))

# Plotting the confusion matrix
fig = plt.figure(figsize=(9, 7))
ax = plt.subplot()
sns.set_context("notebook", font_scale=1.7, rc={"lines.linewidth": 2.8})
sns.heatmap(matrix, cmap="BuGn", annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('K-Nearest Neighbours');
ax.xaxis.set_ticklabels(['A', 'D', 'H']);
ax.yaxis.set_ticklabels(['A', 'D', 'H']);
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/KNN.jpeg', dpi=300)


grid = {'objective': ['multi:softmax'],
        'n_estimators': [10, 20, 40, 65, 100],
        'max_depth': [1, 2, 4, 6],
        'verbosity': [0],
        'use_label_encoder': [False]}
model = XGBClassifier()
grid_search = GridSearchCV(model, grid, n_jobs=1, scoring = 'f1_weighted')
grid_search.fit(X_train, y_train)
print("Best hyperparameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)


# Model performed better with max_depth = 1 on the test set
model = XGBClassifier(max_depth=1, n_estimators=100, objective='multi:softmax',
use_label_encoder=False, verbosity = 0, seed=2)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)

# Metrics
matrix = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print("Classification Report: \n", report)
```

```
print("Train Accuracy: %.2f %%" % (accuracy_score(y_train, model.predict(X_train))*100))
print("Test Accuracy: %.2f %%" % (accuracy_score(y_test, y_pred)*100))
print("MCC: %.2f " % (matthews_corrcoef(y_test, y_pred)))

# Plotting the confusion matrix
fig = plt.figure(figsize=(9, 7))
ax = plt.subplot()
sns.set_context("notebook", font_scale=1.7, rc={"lines.linewidth": 2.8})
sns.heatmap(matrix, cmap="BuGn", annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('XGBoost');
ax.xaxis.set_ticklabels(['A', 'D', 'H']);
ax.yaxis.set_ticklabels(['A', 'D', 'H']);
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/XGBoost.jpeg', dpi=300)


seed(7)
random.set_seed(7)

model = Sequential()
model.add(Dense(60, input_dim=42, activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(3, activation='softmax'))
model.compile(loss='sparse_categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
model.fit(X_train, y_train, epochs=60, batch_size=100, verbose=0)


y_pred = model.predict(X_test)

predictions = np.argmax(y_pred, axis=-1)

# Metrics
matrix = confusion_matrix(y_test, predictions)
report = classification_report(y_test, predictions)
print("Classification Report: \n", report)
print("Train Accuracy: %.2f %%" % (accuracy_score(y_train, np.argmax(model.predict(X_train), axis=-1))*100))
print("Test Accuracy: %.2f %%" % (accuracy_score(y_test, predictions)*100))
```

```
print("MCC: %.2f " % (matthews_corrcoef(y_test, predictions)))

# Plotting the confusion matrix
fig = plt.figure(figsize=(9, 7))
ax = plt.subplot()
sns.set_context("notebook", font_scale=1.7, rc={"lines.linewidth": 2.8})
sns.heatmap(matrix, cmap="BuGn", annot=True, fmt='g', ax=ax);
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Artificial Neural Network');
ax.xaxis.set_ticklabels(['A', 'D', 'H']);
ax.yaxis.set_ticklabels(['A', 'D', 'H']);
fig = ax.get_figure()
fig.tight_layout()
fig.savefig('C:/Users/Christos/Desktop/ANN.jpeg', dpi=300)


# Result predictions of best model - ANN
predicted_results = pd.DataFrame(predictions)
predicted_results

predicted_results.to_csv('C:/Users/Christos/Desktop/predicted_results.csv',
index=False)
```