Final Project: English Premier League 2022-2023

Tharawit Disyawongs

2023-11-30

Overview

This project is a part of HarvardX PH125.9x Data Science: Capstone course. In this project, we will predict English Premier League soccer results from match statistic data in 2022-2023 season, which can be found on https://www.kaggle.com/datasets/thamersekhri/premier-league-stats-2022-2023 . The data set contains results and statistics of all 380 soccer matches happening in that season. In this report, we will start with the overview of the project and its background, followed by data preparation and data exploration. Then, we will perform data analysis along with developing several machine learning algorithms. The performance of each algorithm will be evaluated based on overall accuracy. Then, we will compare the results of different algorithms and come down to the conclusion, and also provide suggestions for future work.

Background

Soccer is one of the most popular sports worldwide. And the English Premier League is one of the most famous soccer leagues. In a soccer match, there are two teams with 11 players each. Both teams try to kick the ball into the other team's goal to score points, and the team with more points (score more goals) wins. If both teams have the same goals at the end of the game, it's a draw. The basic soccer rules can be found here https://www.soccer.com/guide/rules-of-soccer-guide .

In the English league system, there are 20 teams in the Premier League. Each team plays one game at their own stadium (home game) and one at the other team's stadium (away game). Thus, there are 380 league games in total in each season.

Soccer is perceived as one of the most difficult sports to predict. As the nature of low-scoring game, real-time decision making from referees with minimum help and continuous game play with minimum break. All these things make soccer full of surprises and that makes it so unpredictable.

Evaluation Criteria and Approach

In this project, we will normalize the soccer results into three types, home team win, draw and away team win. From there, we will perform classification prediction, and the evaluation criteria that we use for evaluating algorithm performance is overall accuracy (the proportion of the correctly predicted results). Basically, we will make classification predictions using multiple algorithms and compare their overall accuracy.

Overall accuracy can be calculated as the ratio of the number of correct predictions to the total number of predictions.

For 3-class classification model, the formula is as follows:

$$Accuracy = \frac{TP_1 + TP_2 + TP_3}{TP_1 + TP_2 + TP_3 + FP_1 + FP_2 + FP_3}$$

Where:

TP_i is the number of true positives (correct prediction) for class i.

FP_i is the number of false positives (wrong prediction) for class i.

Data preparation

In this project, we will use the data set in CSV format from https://www.kaggle.com/datasets/thamersekhri/premier-league-stats-2022-2023 . The data will be split into two sets, **train_set** and **test_set**. The former will be used for developing machine learning algorithms, while the latter will be used for validation purposes.

We start the data preparation process by reading CSV file, load it into a data frame object and exploring its structure.

```r
# Load CSV file and store the data in a data frame
# Premier_League.csv can be downloaded from
# https://www.kaggle.com/datasets/thamersekhri/premier-league-stats-2022-2023/
stats_file <- "Premier_League.csv"
premier_league_stats <- read.csv(stats_file)

# Explore premier_league_stats data structure
str(premier_league_stats)
```

```
## 'data.frame':    380 obs. of  39 variables:
##  $ date            : chr  "28th May 2023" "28th May 2023" "28th May 2023" "28th May 2023" ...
##  $ clock           : chr  "4:30pm" "4:30pm" "4:30pm" "4:30pm" ...
##  $ stadium         : chr  "Emirates Stadium" "Villa Park" "Gtech Community Stadium" "Stamford Bridge
##  $ attendance      : chr  "60,095" "42,212" "17,120" "40,130" ...
##  $ Home.Team       : chr  "Arsenal" "Aston Villa" "Brentford" "Chelsea" ...
##  $ Goals.Home      : int  5 2 1 1 1 1 1 2 2 4 ...
##  $ Away.Team       : chr  "Wolverhampton Wanderers" "Brighton and Hove Albion" "Manchester City" "Ne
##  $ Away.Goals      : int  0 1 0 1 1 0 4 1 1 4 ...
##  $ home_possessions: num  51 40.3 34.4 64.4 66 37.8 52.1 48.2 53.2 30.9 ...
##  $ away_possessions: num  49 59.7 65.6 35.6 34 62.2 47.9 51.8 46.8 69.1 ...
##  $ home_shots      : int  14 12 11 22 15 13 19 13 21 15 ...
##  $ away_shots      : int  6 8 17 13 7 7 11 16 10 30 ...
##  $ home_on         : int  8 5 4 5 3 6 2 4 8 10 ...
##  $ away_on         : int  0 4 3 4 4 2 7 3 3 8 ...
##  $ home_off        : int  4 5 4 9 8 5 7 6 7 5 ...
##  $ away_off        : int  4 3 6 5 2 3 3 9 6 11 ...
##  $ home_blocked    : int  2 2 3 8 4 2 10 3 6 0 ...
##  $ away_blocked    : int  2 1 8 4 1 2 1 4 1 11 ...
##  $ home_pass       : num  89 75.3 79.3 88.9 85.7 73.1 77.3 87.8 84.2 74.5 ...
##  $ away_pass       : num  88 83.6 89.8 83.3 69.9 83.9 78 88.7 84 86.9 ...
##  $ home_chances    : int  3 4 2 2 1 0 1 2 4 1 ...
##  $ away_chances    : int  0 3 1 2 0 0 3 1 2 5 ...
##  $ home_corners    : int  8 4 3 10 5 9 12 3 5 2 ...
##  $ away_corners    : int  4 3 4 3 4 3 3 5 4 9 ...
##  $ home_offside    : int  1 0 3 2 2 0 1 2 1 0 ...
##  $ away_offside    : int  0 6 0 1 2 1 1 3 1 0 ...
##  $ home_tackles    : num  82.4 42.9 64.7 42.9 40 50 54.5 63.6 78.9 71.4 ...
##  $ away_tackles    : num  44.4 15.4 35.7 42.9 52.6 58.3 46.2 58.8 66.7 75 ...
##  $ home_duels      : num  47.8 52.2 50 54.5 58.3 53.8 54.5 68.8 46.4 54.5 ...
##  $ away_duels      : num  52.2 47.8 50 45.5 41.7 46.2 45.5 31.3 53.6 45.5 ...
##  $ home_saves      : int  0 3 2 3 3 1 3 2 2 4 ...
##  $ away_saves      : int  3 3 3 5 2 5 1 2 6 6 ...
##  $ home_fouls      : int  8 15 12 9 9 11 8 8 14 4 ...
##  $ away_fouls      : int  11 16 8 11 13 12 5 10 10 10 ...
##  $ home_yellow     : int  0 4 4 0 0 1 3 1 1 0 ...
##  $ away_yellow     : int  0 4 0 0 2 3 0 1 2 2 ...
##  $ home_red        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ away_red        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ links           : chr  "https://www.skysports.com/football/arsenal-vs-wolverhampton-wanderers/465
```

From the data structure, premier_league_stats has 380 objects, representing 380 English Premier League matches in 2022-2023 seasons, with 39 columns(variables) representing match statistics

Some of the column name formats are not consistent, such as Goals.Home, Away.Goals and away_shots. So, we will do a bit of renaming to make them more consistent. Also, we observe that home_team, away_team and attendance are **char**, so we will convert them to **factor** and **numeric** to make them more suitable for using in prediction model development.

```r
# Rename columns and converting data types for data further exploration and processing
premier_league_stats <- premier_league_stats %>% rename("home_team"="Home.Team",
                                                        "away_team"="Away.Team",
                                                        "home_goals"="Goals.Home",
                                                        "away_goals"="Away.Goals",
                                                        "home_on_target"="home_on",
                                                        "away_on_target"="away_on",
                                                        "home_off_target"="home_off",
                                                        "away_off_target"="away_off"
)

#Convert home team name from char to factor
premier_league_stats$home_team <- as.factor(premier_league_stats$home_team)

#Convert away team name from char to factor
premier_league_stats$away_team <- as.factor(premier_league_stats$away_team)

#Convert attendance from char to numeric
premier_league_stats$attendance <- as.numeric(gsub(',','',premier_league_stats$attendance))
```

After that, we check if any column contains NA values. Per our observation, the attendance column contains 5 NA values. So, we will exclude that incomplete column from our algorithm development.

```r
#Find a column containing NA
colSums(is.na(premier_league_stats))
```

```
##            date           clock         stadium      attendance 
##               0               0               0               5 
##       home_team      home_goals       away_team      away_goals 
##               0               0               0               0 
## home_possessions away_possessions      home_shots      away_shots 
##               0               0               0               0 
##   home_on_target  away_on_target home_off_target away_off_target 
##               0               0               0               0 
##     home_blocked    away_blocked       home_pass       away_pass 
##               0               0               0               0 
##     home_chances    away_chances    home_corners    away_corners 
##               0               0               0               0 
##     home_offside    away_offside    home_tackles    away_tackles 
##               0               0               0               0 
##       home_duels      away_duels      home_saves      away_saves 
##                0                0                0                0
##       home_fouls       away_fouls      home_yellow      away_yellow
##                0                0                0                0
##         home_red         away_red            links
##                0                0                0
```

The next process is to add a new variable call **result**. This variable will contain match results without score, and it will be used for outcome evaluation. The variable will be a factor with three levels, **H** representing home team win, **D** representing draw and **A** representing away team win.

```
# Create a new column named "result" to summarize the match result to H/D/A
# H means home team wins, D means draw and A means away team wins
premier_league_stats$result <- apply(premier_league_stats[, c('home_goals', 'away_goals')], 1,
                                     function(row) {if (row[1] > row[2]) {
                                                        return("H")
                                                      } else if (row[1] == row[2]) {
                                                        return("D")
                                                      } else {
                                                        return("A")
                                                      }
                                                    })

# Re-order levels in result column
premier_league_stats$result <- factor(premier_league_stats$result, levels=c('H', 'D', 'A'))
```

After that, we delete unused columns and re-check the data frame structure again to ensure that all variables are in proper formats with no NA value.

```
# Delete unused columns
premier_league_stats <- subset(premier_league_stats,
                               select = -c(date, clock, stadium, attendance,
                                           home_goals, away_goals, links))

# Display data frame structure
str(premier_league_stats)
```

```
## 'data.frame':    380 obs. of  33 variables:
##  $ home_team       : Factor w/ 20 levels "Arsenal","Aston Villa",..: 1 2 4 6 7 8 10 11 14 17 ...
##  $ away_team       : Factor w/ 20 levels "Arsenal","Aston Villa",..: 20 5 13 15 16 3 18 19 9 12 ...
##  $ home_possessions: num  51 40.3 34.4 64.4 66 37.8 52.1 48.2 53.2 30.9 ...
##  $ away_possessions: num  49 59.7 65.6 35.6 34 62.2 47.9 51.8 46.8 69.1 ...
##  $ home_shots      : int  14 12 11 22 15 13 19 13 21 15 ...
##  $ away_shots      : int  6 8 17 13 7 7 11 16 10 30 ...
##  $ home_on_target  : int  8 5 4 5 3 6 2 4 8 10 ...
##  $ away_on_target  : int  0 4 3 4 4 2 7 3 3 8 ...
##  $ home_off_target : int  4 5 4 9 8 5 7 6 7 5 ...
##  $ away_off_target : int  4 3 6 5 2 3 3 9 6 11 ...
##  $ home_blocked    : int  2 2 3 8 4 2 10 3 6 0 ...
##  $ away_blocked    : int  2 1 8 4 1 2 1 4 1 11 ...
##  $ home_pass       : num  89 75.3 79.3 88.9 85.7 73.1 77.3 87.8 84.2 74.5 ...
##  $ away_pass       : num  88 83.6 89.8 83.3 69.9 83.9 78 88.7 84 86.9 ...
##  $ home_chances    : int  3 4 2 2 1 0 1 2 4 1 ...
##  $ away_chances    : int  0 3 1 2 0 0 3 1 2 5 ...
##  $ home_corners    : int  8 4 3 10 5 9 12 3 5 2 ...
##  $ away_corners    : int  4 3 4 3 4 3 3 5 4 9 ...
##  $ home_offside    : int  1 0 3 2 2 0 1 2 1 0 ...
##  $ away_offside    : int  0 6 0 1 2 1 1 3 1 0 ...
##  $ home_tackles    : num  82.4 42.9 64.7 42.9 40 50 54.5 63.6 78.9 71.4 ...
##  $ away_tackles    : num  44.4 15.4 35.7 42.9 52.6 58.3 46.2 58.8 66.7 75 ...
##  $ home_duels      : num  47.8 52.2 50 54.5 58.3 53.8 54.5 68.8 46.4 54.5 ...
##  $ away_duels      : num  52.2 47.8 50 45.5 41.7 46.2 45.5 31.3 53.6 45.5 ...
##  $ home_saves      : int  0 3 2 3 3 1 3 2 2 4 ...
##  $ away_saves      : int  3 3 3 5 2 5 1 2 6 6 ...
##  $ home_fouls      : int  8 15 12 9 9 11 8 8 14 4 ...
##  $ away_fouls      : int  11 16 8 11 13 12 5 10 10 10 ...
##  $ home_yellow     : int  0 4 4 0 0 1 3 1 1 0 ...
##  $ away_yellow     : int  0 4 0 0 2 3 0 1 2 2 ...
##  $ home_red        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ away_red        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ result          : Factor w/ 3 levels "H","D","A": 1 1 1 2 2 1 3 1 1 2 ...
```

```
# Check if we have NA in the data set
sum(is.na(premier_league_stats))
```

```
## [1] 0
```

The next process is creating train and test sets. Train set will be used for algorithm development while the test set will be used for evaluating results. As the data set is small (only 380 observations), to ensure that we have enough data for both algorithm development and validation, we use 70% of data as train set, and use the rest of 30% for test set. In short, as below, 264 observations will be used for model training, and 116 observations will be used for validation

```
###########################################################
# Create test and train data sets
###########################################################
set.seed(1)
test_index <- createDataPartition(premier_league_stats$result, times = 1,
                                  p = 0.3, list = FALSE)
test_set <- premier_league_stats[test_index,]
train_set <- premier_league_stats[-test_index,]

# Check the size of test set and train set
nrow(test_set)
```

```
## [1] 116
```

```
nrow(train_set)
```

```
## [1] 264
```

Data analysis and modeling approaches

In this section, we will go through several data analysis and modeling approaches. Then, we will present the result of each model.

Model 1: Home team effect

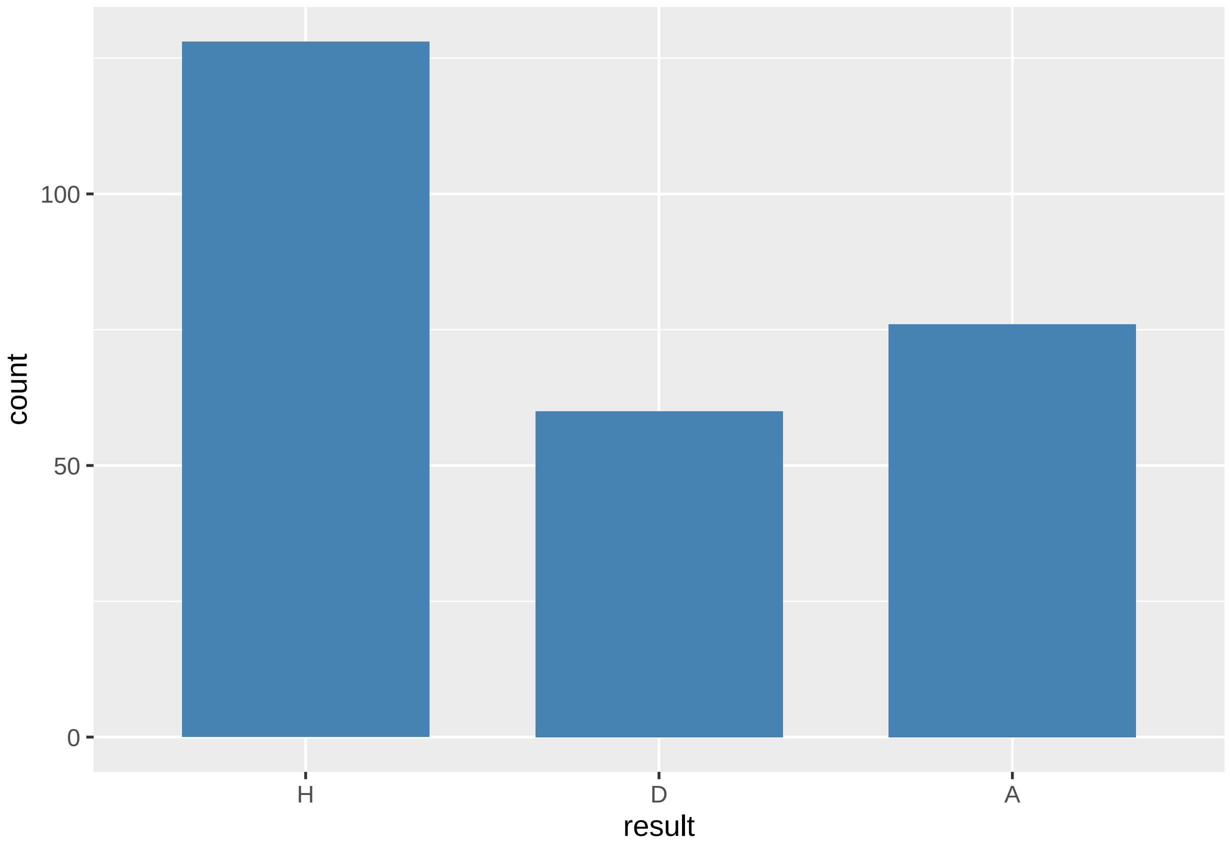
If we ask someone who has no idea about sports at all to guess the results, what is likely to happen is that he/she will make a random guess. By doing so, the chance of having the correct answers would be around 0.33, as the following Monte Carlo simulation.

```
set.seed(1)
results <- replicate(10000, {
  guess = sample(c('H', 'D', 'A'), nrow(test_set), replace=TRUE)
  mean(guess==test_set$result)
})
mean(results)
```

```
## [1] 0.3335836
```

However, lots of people know about sports, and they know about the advantage of home teams over away teams. To see that more vividly, we create a bar chart to virtualize the home team effect as the following.

```
# Plot to virtualize home team effect
ggplot(train_set, aes(x=result)) +
  geom_bar(stat="count", width=0.7, fill="steelblue")
```



From the bar chart, the number of home team wins is higher than any other result. So, in the first model, we will predict the home team to win for every match. The accuracy that we got from the test set is as the following.

```
# Predict all home teams to win
model_1_accu <- mean(test_set$result=='H')

# Save result to a data frame for further display
results <- data_frame(Method = "Model 1: Home team effect",
                      Accuracy = round(model_1_accu, 7))

results %>% knitr::kable()
```

Method	Accuracy
Model 1: Home team effect	0.4827586

Data exploration: Find predictors via Kruskal-Wallis Test

To develop further models, we need to answer one important question - what the strong predictors for predicting soccer match results are. To answer that, we will do a simple check via Kruskal-Wallis Test for all the potential predictors.

The Kruskal-Wallis test is a tool used in statistics to check if there are any significant differences between independent groups. Kruskal-Wallis test does not tell which groups are different from each other - it only indicates whether there are overall differences. More information about Kruskal-Wallis test can be found here, https://rcompanion.org/handbook/F_08.html .

As the following, we run Kruskal-Wallis Test on all variables, and look at their p-values. The ones that have p-values less than 0.05 are considered statistically significant, and we will use them for further model developments.

```r
# Put all column names into a variable, except result
c_names <- colnames(train_set)
c_names <- c_names[ !c_names == 'result']

# Run Kruskal-Wallis test
kruskal_test<-sapply(c_names , function(c){
  p<-kruskal.test(train_set$result ~ train_set[,c])$p.value
  return(p)
})

# Put Kruskal-Wallis test results in data frame format and also check if any variables is significant
kruskal_test_df <- data.frame(predictors = names(kruskal_test),
                              p_value = kruskal_test, row.names = NULL)
kruskal_test_df$significance <- kruskal_test_df$p_value < 0.05
kruskal_test_df %>%  arrange(p_value)
```

```
##         predictors      p_value significance
## 1      away_chances 6.967726e-09         TRUE
## 2      home_chances 2.672772e-07         TRUE
## 3    away_on_target 4.252778e-07         TRUE
## 4    home_on_target 1.779872e-05         TRUE
## 5         home_team 5.600949e-05         TRUE
## 6        away_shots 6.803684e-04         TRUE
## 7          home_red 1.265437e-02         TRUE
## 8         away_team 4.680009e-02         TRUE
## 9   away_off_target 5.075971e-02        FALSE
## 10      away_yellow 6.372962e-02        FALSE
## 11     away_blocked 9.543001e-02        FALSE
## 12         away_red 1.171533e-01        FALSE
## 13       home_shots 1.324699e-01        FALSE
## 14        away_pass 1.645195e-01        FALSE
## 15       away_saves 1.711639e-01        FALSE
## 16     away_corners 1.915204e-01        FALSE
## 17     home_corners 3.349115e-01        FALSE
## 18     away_tackles 3.860872e-01        FALSE
## 19       home_duels 3.974600e-01        FALSE
## 20       away_duels 3.974600e-01        FALSE
## 21      home_tackles 4.027706e-01        FALSE
## 22 home_possessions 4.089790e-01        FALSE
## 23 away_possessions 4.089790e-01        FALSE
## 24        home_saves 4.105094e-01        FALSE
## 25       home_yellow 4.132725e-01        FALSE
## 26        home_fouls 4.506528e-01        FALSE
## 27      home_offside 4.630684e-01        FALSE
## 28   home_off_target 5.180947e-01        FALSE
## 29        away_fouls 6.172172e-01        FALSE
## 30      home_blocked 6.378829e-01        FALSE
## 31         home_pass 6.935811e-01        FALSE
## 32      away_offside 7.056230e-01        FALSE
```

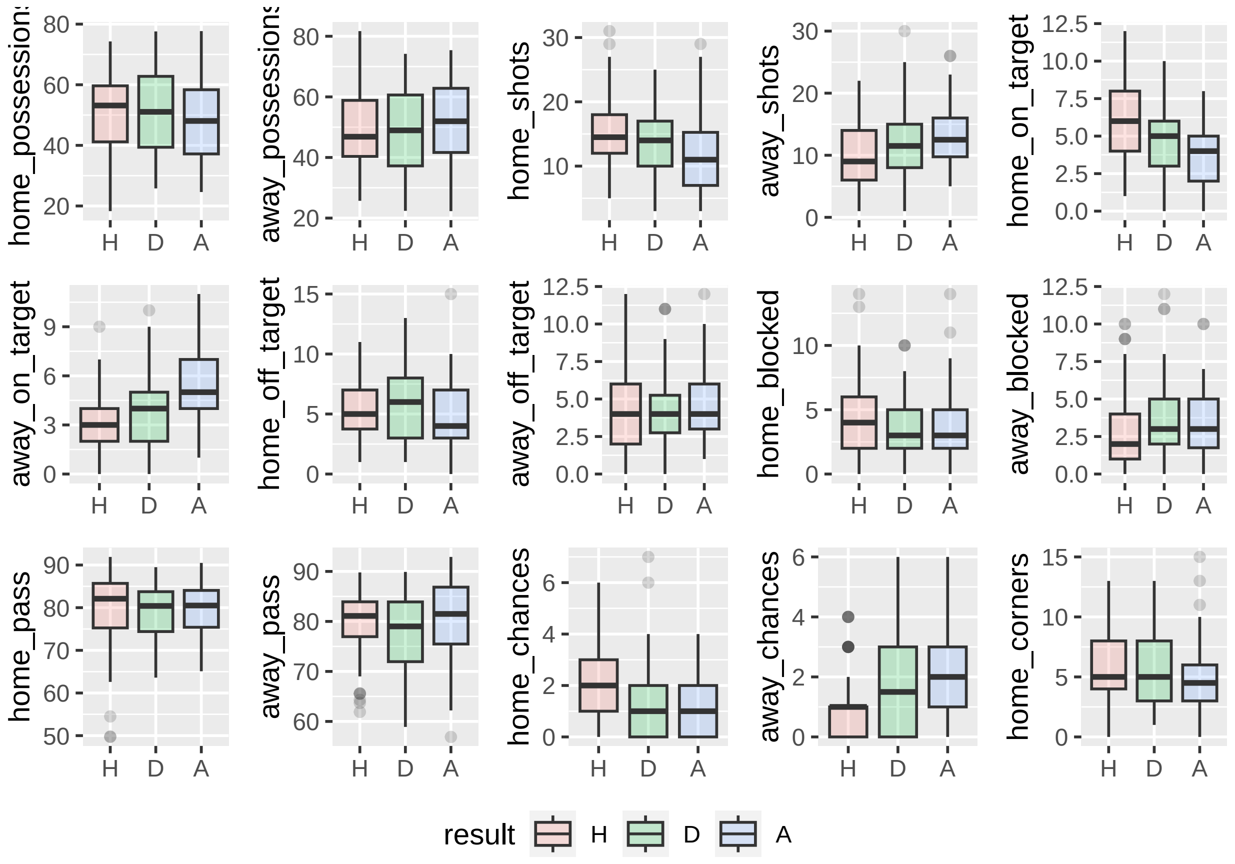
Data exploration: Find predictors via data observation (box plots)

Another technique that we will use in the predictor selection process is by observing variabilities via box plots. Box plot is a simple but powerful enough tool to observe data variability for each predictor. Basically, we will create the plot of each variable, and generate three boxplots based on different match results. Then, we will observe if there are overlaps between those boxplots. Variables that have less overlapped areas among H, D and A are considered stronger predictors. The overlapped area that we primarily look at is the box area, representing 50% of data in that group (25th percentile to 75th percentile), and the median of each group.

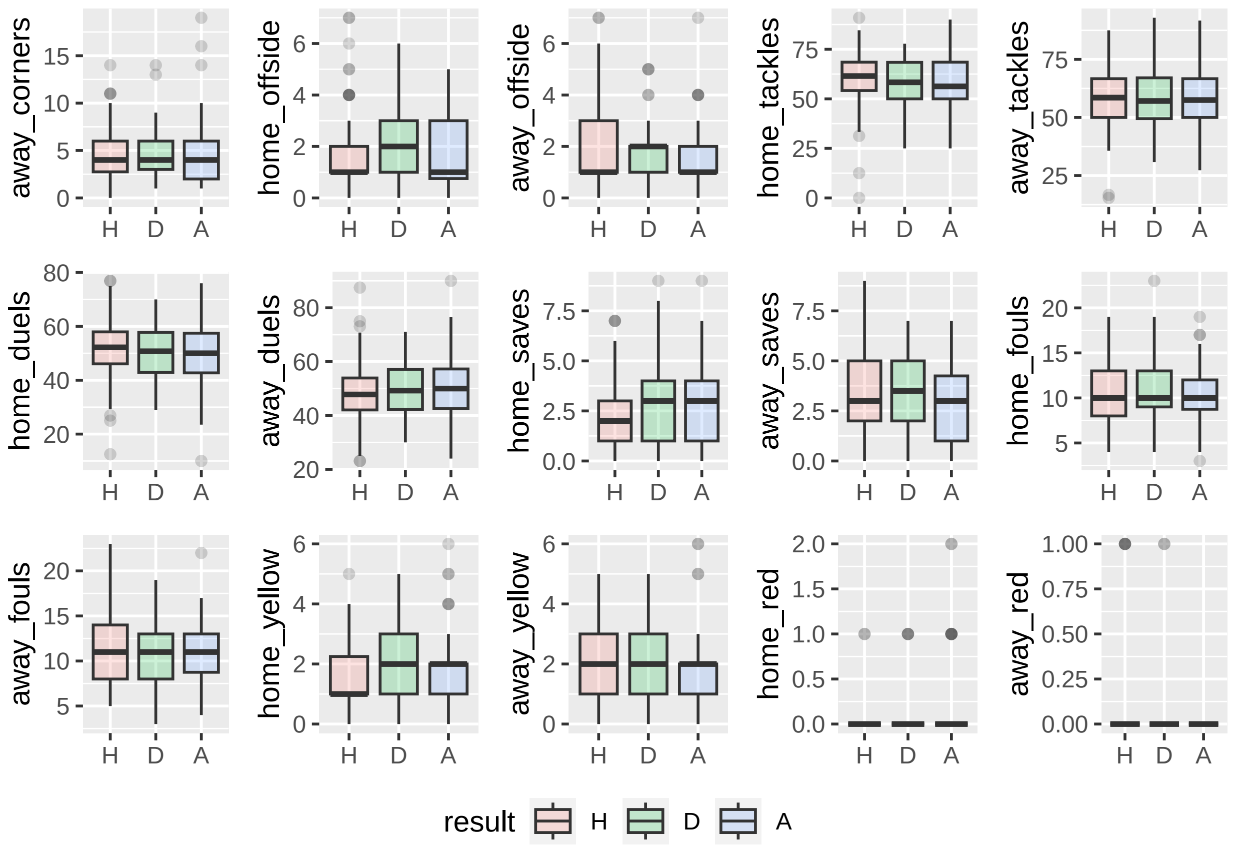
Given that, we create box plots for all variables as the following.

```r
# Create box plots for all variables except home_team & away_team
# since they are not suitable for box plots.
boxplot_list <-lapply(c_names[3:32], function(column) {
  train_set %>% ggplot(aes(result, !!sym(column), fill = result)) +
    geom_boxplot(alpha = 0.2) + theme(axis.title.x = element_blank())
})

# Put box plots into a figure and display them
figure1 <- ggarrange(plotlist = boxplot_list[1:15], ncol = 5, nrow = 3,
                    common.legend = TRUE, legend = "bottom")
figure1
```



```
figure2 <- ggarrange(plotlist = boxplot_list[16:30], ncol = 5, nrow = 3,
                     common.legend = TRUE, legend = "bottom")
figure2
```



By observing box plots, the variables that tend to be good predictors (compared to others) are home_shots, away_shots, home_on_target, away_on_target, away_pass, home_chances, away_chances, home_offside and away_offside.

Now that we have predictor lists, both from Kruskal-Wallis Test and box plots, we are ready to go further on model developments.

Model 2: Rpart with predictor list from box plot observation

Decision tree algorithm is a good candidate for classification problems. And the first one that we choose is Rpart algorithm. The **Rpart** name comes from **Recursive Partitioning**, which is the process used by decision trees to split data into subsets based on criteria. In this model, we will use the predictor(feature) list from box plot observation. The code is as below.

```
train_rpart_b <- train( result ~ home_shots
                        + away_shots + home_on_target + away_on_target + away_pass
                        + home_chances + away_chances + home_offside + away_offside,
                        method = "rpart",
                        data = train_set)

model_2_accu <- confusionMatrix(predict(train_rpart_b, test_set, type = "raw"),
                                test_set$result)$overall["Accuracy"]

results <- rbind(results,c("Model 2: Rpart with predictor list from box plot observation",
                           round(model_2_accu, 7)))
results %>% knitr::kable()
```

Method	Accuracy
Model 1: Home team effect	0.4827586
Model 2: Rpart with predictor list from box plot observation	0.5344828

From the above result, the accuracy looks better than model 1. However, in the next model, let's try the predictor list from Kruskal-Wallis Test and see if it could improve the accuracy.

Model 3: Rpart with predictor list from Kruskal-Wallis Test

In this model, we use RPart algorithm with predictors from Kruskal-Wallis Test, the code looks like the following.

```r
train_rpart_k <- train( result ~ home_team + away_team + away_shots + home_on_target
                        + away_on_target + home_chances + away_chances + home_red,
                        method = "rpart",
                        data = train_set)

model_3_accu <- confusionMatrix(predict(train_rpart_k, test_set, type = "raw"),
                                test_set$result)$overall["Accuracy"]

results <- rbind(results,c("Model 3: Rpart with predictor list from Kruskal-Wallis Test",
                           round(model_3_accu, 7)))
results %>% knitr::kable()
```

Method	Accuracy
Model 1: Home team effect	0.4827586
Model 2: Rpart with predictor list from box plot observation	0.5344828
Model 3: Rpart with predictor list from Kruskal-Wallis Test	0.5344828

The result is exactly the same as using Rpart with the predictor list from box plots. It appears that both model 2 and model 3 generate exactly the same outcome. Usually, Rpart is prone to overfitting, and it could be the case here where some overlapped strong predictors from both lists could dictate the outcomes. So, in this model, the change in predictor list does not make any improvement.

```r
#Compare model 2 prediction to model 3 prediction
identical(predict(train_rpart_b, test_set, type = "raw"),
          predict(train_rpart_k, test_set, type = "raw"))
```

```
## [1] TRUE
```

Model 4: KNN with predictor list from box plot observation

KNN, k-nearest neighbors algorithm, uses proximity to make classifications or predictions about the grouping of a data point. For classification problems, KNN works by assigning a class label to a data point on the basis of a majority vote. In another word, the label that most frequently appears around a given data point will be used. In this model, we will try using KNN with the predictor list from box plots.

```r
set.seed(1)
train_knn_b <- train(result ~ home_shots
                     + away_shots + home_on_target + away_on_target + away_pass
                     + home_chances + away_chances + home_offside + away_offside,
                     tuneGrid = expand.grid(k = seq(50, 100, by = 1)),
                     method ="knn",
                     data = train_set)

model_4_accu <- confusionMatrix(predict(train_knn_b, test_set, type = "raw"),
                                test_set$result)$overall[["Accuracy"]]

results <- rbind(results,c("Model 4: KNN with predictor list from box plot observation",
                           round(model_4_accu, 7)))
results %>% knitr::kable()
```

Method	Accuracy
Model 1: Home team effect	0.4827586
Model 2: Rpart with predictor list from box plot observation	0.5344828
Model 3: Rpart with predictor list from Kruskal-Wallis Test	0.5344828
Model 4: KNN with predictor list from box plot observation	0.5258621

The accuracy we got is worse than the previous RPart models. Usually, KNN is sensitive to outliers, and we ignore that during the predictor selection process from box plot observation. So, it could be the case here. In the next model, let's try KNN with the predictor list from Kruskal-Wallis Test, which outliners are taken into account, then observe if there is any improvement.

Model 5: KNN with predictor list from Kruskal-Wallis Test

In this model, we will use KNN with statistically significant predictors from Kruskal-Wallis Test. The code and the result is as the following.

```
set.seed(1)
train_knn_k <- train(result ~ home_team + away_team + away_shots + home_on_target +
                       away_on_target + home_chances + away_chances + home_red,
                     tuneGrid = expand.grid(k = seq(50, 100, by = 1)),
                     method ="knn",
                     data = train_set)

model_5_accu <- confusionMatrix(predict(train_knn_k, test_set, type = "raw"),
                                test_set$result)$overall[["Accuracy"]]

results <- rbind(results,c("Model 5: KNN with predictor list from Kruskal-Wallis Test",
                           round(model_5_accu, 7)))
results %>% knitr::kable()
```

Method	Accuracy
Model 1: Home team effect	0.4827586
Model 2: Rpart with predictor list from box plot observation	0.5344828
Model 3: Rpart with predictor list from Kruskal-Wallis Test	0.5344828
Model 4: KNN with predictor list from box plot observation	0.5258621
Model 5: KNN with predictor list from Kruskal-Wallis Test	0.5775862

The accuracy in this model is much better than the previous one. Also, it's the best result by far. It seems like using the predictor list from Kruskal-Wallis Test works well with KNN algorithm.

Model 6: Random Forest with all predictors

Now, we will try another powerful decision tree algorithm - Random Forest. Random Forest is an ensemble of decision trees. Basically, it builds and combines multiple decision trees to generate better accuracy. It can also deal with a large number of predictors. So, it seems to be a good algorithm to use in our case. However, we will try something different here. We will start by throwing all predictors to Random Forest and let it figure out how to deal with them.

```r
set.seed(1)
train_rf_a <- randomForest(result ~ ., data=train_set,na.action = na.pass)

model_6_accu <-confusionMatrix(predict(train_rf_a, test_set), test_set$result)$overall["Accuracy"]

results <- rbind(results,c("Model 6: Random forest with all predictors", round(model_6_accu, 7)))
results %>% knitr::kable()
```

Method	Accuracy
Model 1: Home team effect	0.4827586
Model 2: Rpart with predictor list from box plot observation	0.5344828
Model 3: Rpart with predictor list from Kruskal-Wallis Test	0.5344828
Model 4: KNN with predictor list from box plot observation	0.5258621
Model 5: KNN with predictor list from Kruskal-Wallis Test	0.5775862
Model 6: Random forest with all predictors	0.5862069

Even without specifying any predictor or tuning parameter, Random Forest can produce the most accurate prediction so far. However, let's see if we can get a better result by adjusting some predictors.

Model 7: Random Forest with predictor list from box plot observation

Now, we will use Random Forest algorithm with the predictor list from box plots. The code is as below.

```r
set.seed(1)
train_rf_b <- randomForest(result ~home_shots
                           + away_shots + home_on_target + away_on_target + away_pass
                           + home_chances + away_chances + home_offside + away_offside,
                           data=train_set,na.action = na.pass)

model_7_accu <- confusionMatrix(predict(train_rf_b, test_set), test_set$result)$overall["Accuracy"]

results <- rbind(results,c("Model 7: Random Forest with predictor list from box plot observation",
                           round(model_7_accu, 7)))
results %>% knitr::kable()
```

Method	Accuracy
Model 1: Home team effect	0.4827586
Model 2: Rpart with predictor list from box plot observation	0.5344828
Model 3: Rpart with predictor list from Kruskal-Wallis Test	0.5344828
Model 4: KNN with predictor list from box plot observation	0.5258621
Model 5: KNN with predictor list from Kruskal-Wallis Test	0.5775862
Model 6: Random forest with all predictors	0.5862069
Model 7: Random Forest with predictor list from box plot observation	0.5948276

When using selected predictors, Random Forest can generate a better result. The accuracy that we got from this model is better than the previous ones.

Model 8: Random Forest with predictor list from Kruskal-Wallis Test

In our last model, we will use Random Forest algorithm with the predictor list from Kruskal-Wallis Test, which is statistically significant, then observe the result.

```r
set.seed(1)
train_rf_k <- randomForest(result ~ home_team + away_team + away_shots + home_on_target
                           + away_on_target + home_chances + away_chances + home_red,
                           data=train_set,na.action = na.pass)
model_8_accu <- confusionMatrix(predict(train_rf_k, test_set), test_set$result)$overall["Accuracy"]

results <- rbind(results,c("Model 8: Random Forest with predictor list from Kruskal-Wallis Test",
                           round(model_8_accu, 7)))
results %>% knitr::kable()
```

Method	Accuracy
Model 1: Home team effect	0.4827586
Model 2: Rpart with predictor list from box plot observation	0.5344828
Model 3: Rpart with predictor list from Kruskal-Wallis Test	0.5344828
Model 4: KNN with predictor list from box plot observation	0.5258621
Model 5: KNN with predictor list from Kruskal-Wallis Test	0.5775862
Model 6: Random forest with all predictors	0.5862069
Model 7: Random Forest with predictor list from box plot observation	0.5948276
Model 8: Random Forest with predictor list from Kruskal-Wallis Test	0.612069

The accuracy that we got is the highest by far. It seems like, from all the models that we develop in this project, Random Forest with statistically significant predictors chosen from Kruskal-Wallis Test is the one that generates the best result.

Conclusion

In this project, we try to predict the results of English Premier League soccer, one of the most difficult sports to predict. We have gone through the process of data cleansing, predictor selection and a number of model developments, such as linear model, Rpart, KNN and Random Forest. For the predictor selection process, choosing statistically significant predictors considering p-values from Kruskal-Wallis Test seems to be the best way. And from the result comparison using overall accuracy, Random Forest seems to be the algorithm that provides the highest accuracy.

In a 3-class classification problem, a random guess would generate around 0.33 accuracy. However, in our best model, Random Forest with the predictor list from Kruskal-Wallis Test, we can generate the overall accuracy at 0.612069, which is quite an improvement.

Future Work

While our models show satisfactory results, there are several areas to explore for enhancing the accuracy of English Premier League soccer prediction.

- Historical Results: The English Premier League has a rich history, with teams facing each other a number of times. Exploring statistics from historical results can provide insights for predicting future outcomes. Learning patterns from past matches may offer a reliable way of forecasting.
- Recent Team Form: In sports, the recent performance of a team often plays an important role. Teams can be in good or bad form, and that impacts their game performances. Given that, the recent form,

such as the results of the last 5 or 10 matches, could be a strong predictor. And that should be something worth testing in our future model.

- Player Data: Team performance is on the top of player performance. So, individual player data could significantly enhance prediction models. Features such as player form, skills, injuries, and other relevant statistics could be important indicators for predicting match results. By taking player-specific information into account, we may achieve a more accurate prediction.

References

- https://www.kaggle.com/datasets/thamersekhri/premier-league-stats-2022-2023
- https://rcompanion.org/handbook/F_08.html
- https://cran.r-project.org/web/packages/rpart/index.html
- https://cran.r-project.org/web/packages/randomForest/