Engligh Premier League correct score prediction

Michael Lai

2023-04-12

Introduction & Overview

Football (or soccer in American English) is arguably the most popular sport on the planet, with over 4 billion fans watching the games worldwide (1). And among the professional leagues around the world, the English Premier League (EPL) is in turn arguably the most commercially successful, with its viewership skyrocketing in recent years. Established since 1992, the top-flight professional league in England currently comprises 20 teams from the region and (on a few occasions) Wales. With each team playing each other once at home and once away, there are a total of 380 matches per season. Each team will be awarded 3 points for a win, 1 point for a draw and no point for a loss. The team with the most number of points at the end of the season becomes the EPL Championship, while the three teams at the bottom will be relegated to the second tier of the league ladder and replaced by the top 3 teams from that tier. With the exception of 2019-2020 where COVID-19 disrupted the normal schedule of the tournament, a typical season starts in August and ends in May the next year.

And always comes with any major sporting activity is the gambling business behind it. The betting companies offer a plethora of betting choices, ranging from Full-Time Result (Home Team Win/Home Team Loss/Draw), Total Goals (total number of goals above or below a certain threshold), Correct Scores (predicting the exact scores for both teams) and a myriad of other types, each having a different ranges of odds. Obviously, the more difficult to predict a certain outcome, the higher its odds will be.

While not encouraging reckless gambling, some people have pondered whether it is possible to use statistical models to predict the outcome of a game so as to make a predictable profit over the long run or, in layman term, "beat the bookies". Indeed, an article on Medium tries to explore such as possibility by using Python to predict the Full-Time Result and claim to have moderate success. (2) Inspired by such an analysis, this article will explore the possibility of predicting the Correct Scores by R using past match results, and if it goes well, beating the bookies in this betting category.

Method & Analysis

To gather data from all EPL previous seasons into a single datasheet would be a daunting task. Luckily, someone has generously performed this task and uploaded the data onto the online data science community site Kaggle. (3) The problem here is that the last results were only last updated on 10 Apr 2022, by which time the season 2021-22 had yet finished. Therefore, in order to fill in this gap I have downloaded the file, manually updated the entries all the way up to 10 Apr 2023 by referring to the official EPL website (4). The updated file is uploaded to Github as an Excel file "EPL_results_2023-04-10.xlsx" and can be downloaded via the link:

https://github.com/diplomike/EPL_results_prediction/raw/9580aadcef53662dc7c54e6a9025b53504644b0a/EPL_results_2023-04-10.xlsx

On the other hand, the download can be executed by the following code:

```
if(!file.exists("EPL_results.xlsx"))
download.file("https://github.com/diplomike/EPL_results_prediction/raw/
9580aadcef53662dc7c54e6a9025b53504644b0a/EPL_results_2023-04-10.xlsx",
"EPL_results.xlsx", mode="wb")
```

After downloading the file, it is read into R.

```
read_xlsx("EPL_results.xlsx")
```

```
## # A tibble: 11,480 x 23
##    Season  DateTime HomeT~1 AwayT~2  FTHG  FTAG FTR   HTHG  HTAG  HTR   Referee HS
##    <chr>   <chr>    <chr>   <chr>   <dbl> <dbl> <chr> <chr> <chr> <chr> <chr>   <chr>
##  1 1993-94 1993-08~ Arsenal Covent~     0     3 A     NA    NA    NA    NA      NA
##  2 1993-94 1993-08~ Aston ~ QPR         4     1 H     NA    NA    NA    NA      NA
##  3 1993-94 1993-08~ Chelsea Blackb~     1     2 A     NA    NA    NA    NA      NA
##  4 1993-94 1993-08~ Liverp~ Sheffi~     2     0 H     NA    NA    NA    NA      NA
##  5 1993-94 1993-08~ Man Ci~ Leeds       1     1 D     NA    NA    NA    NA      NA
##  6 1993-94 1993-08~ Newcas~ Totten~     0     1 A     NA    NA    NA    NA      NA
##  7 1993-94 1993-08~ Oldham  Ipswich     0     3 A     NA    NA    NA    NA      NA
##  8 1993-94 1993-08~ Sheffi~ Swindon     3     1 H     NA    NA    NA    NA      NA
##  9 1993-94 1993-08~ Southa~ Everton     0     2 A     NA    NA    NA    NA      NA
## 10 1993-94 1993-08~ West H~ Wimble~     0     2 A     NA    NA    NA    NA      NA
## # ... with 11,470 more rows, 11 more variables: AS <chr>, HST <chr>, AST <chr>,
## #   HC <chr>, AC <chr>, HF <chr>, AF <chr>, HY <chr>, AY <chr>, HR <chr>, AR <chr>,
## #   and abbreviated variable names 1: HomeTeam, 2: AwayTeam
```

Although there are many columns for detailed statistics of each match, we are primarily concerned with the first six variables, namely the Season, DateTime (Date of the match), HomeTeam, AwayTeam, the FTHG (full time home score) and FTAG (full time away score). The remaining variables are omitted from for the time being to simplify the project. Also the time of the matches are only present in the recent seasons, so we will just ignore this piece of information and modify DateTime into a date-only variable.

```
epl_results <- read_xlsx("EPL_results.xlsx") %>%
  select(Season, DateTime, HomeTeam, AwayTeam, FTHG, FTAG) %>% mutate(DateTime=date(DateTime))
```

The data frame below shows the number of records for each season.

```
epl_results %>% group_by(Season) %>% summarize(matches=n()) %>% print(n = 30)
```

```
## # A tibble: 30 x 2
##    Season  matches
##    <chr>     <int>
##  1 1993-94     462
##  2 1994-95     462
##  3 1995-96     380
##  4 1996-97     380
##  5 1997-98     380
##  6 1998-99     380
##  7 1999-00     380
##  8 2000-01     380
##  9 2001-02     380
```

```
## 10 2002-03     380
## 11 2003-04     380
## 12 2004-05     380
## 13 2005-06     380
## 14 2006-07     380
## 15 2007-08     380
## 16 2008-09     380
## 17 2009-10     380
## 18 2010-11     380
## 19 2011-12     380
## 20 2012-13     380
## 21 2013-14     380
## 22 2014-15     380
## 23 2015-16     380
## 24 2016-17     380
## 25 2017-18     380
## 26 2018-19     380
## 27 2019-20     380
## 28 2020-21     380
## 29 2021-22     380
## 30 2022-23     296
```

It can be seen that the first few seasons of the EPL consisted of 22 teams (22 * 21 = 462 matches); on the other hand, a little over three quarters of the matches of the current season have been played. To verify the manual inputs of the data in the current season are correct, the code below produces the league table as of 10 Apr 2023, and the statistics are compared to the official source to confirm its accuracy.

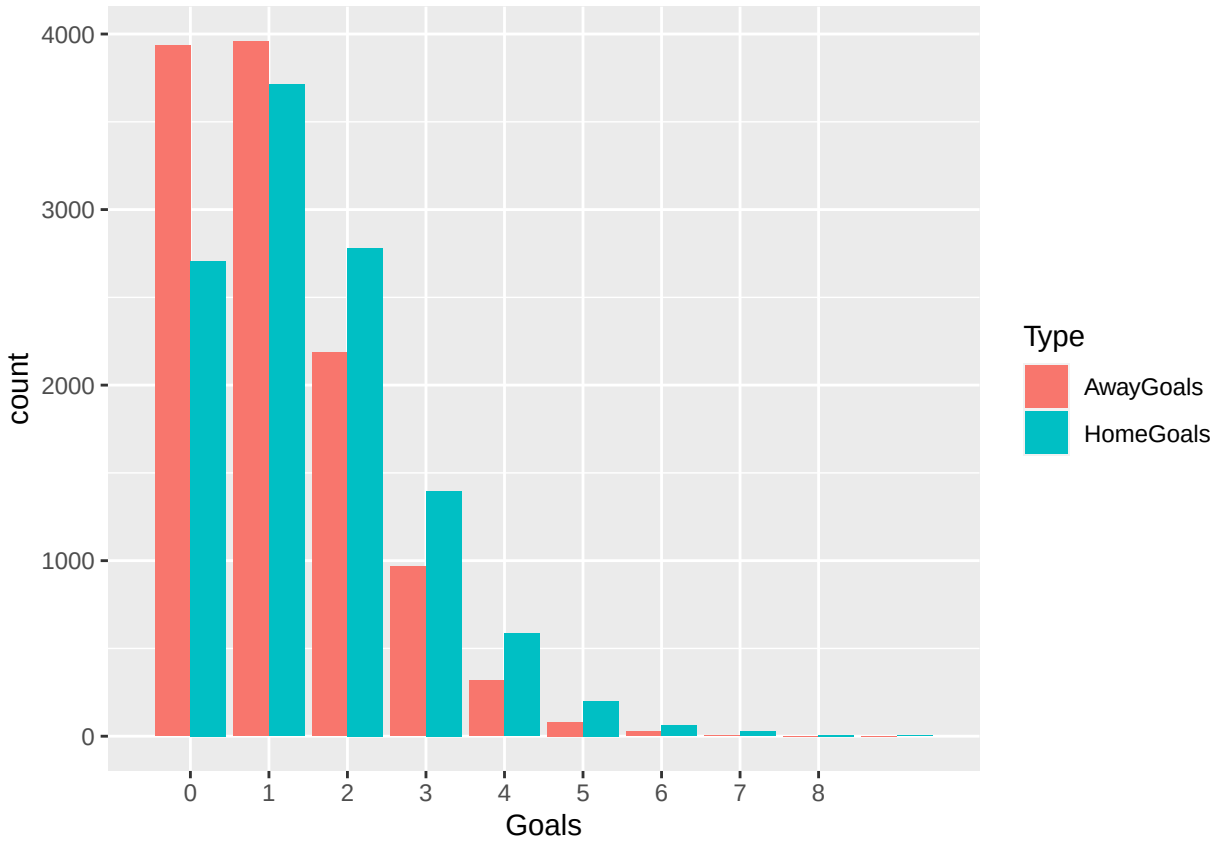
```
epl_results %>% filter(Season=="2022-23") %>% select(HomeTeam, AwayTeam, FTHG, FTAG) %>%
  pivot_longer(HomeTeam:AwayTeam, names_to= "venue", values_to = "Team") %>%
  mutate(Won = ifelse((FTHG-FTAG)*(venue=="HomeTeam")+(FTAG-FTHG)*(venue=="AwayTeam") > 0, 1, 0),
         Drawn = ifelse(FTHG == FTAG, 1, 0),
         Lost = ifelse((FTHG-FTAG)*(venue=="HomeTeam")+(FTAG-FTHG)*(venue=="AwayTeam") < 0, 1, 0),
         GF = ifelse(venue=="HomeTeam", FTHG, FTAG),
         GA = ifelse(venue=="AwayTeam", FTHG, FTAG),
         GD = GF-GA, Pts = Won * 3 + Drawn) %>%
  group_by(Team) %>% summarize(Played = n(), Won=sum(Won), Drawn=sum(Drawn), Lost=sum(Lost),
                               GF=sum(GF),GA=sum(GA), GD=sum(GD), Pts=sum(Pts)) %>%
  arrange(desc(Pts), desc(GD),desc(GF))
```

```
## # A tibble: 20 x 9
##    Team           Played   Won Drawn  Lost    GF    GA    GD   Pts
##    <chr>           <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Arsenal            30    23     4     3    72    29    43    73
##  2 Man City           29    21     4     4    75    27    48    67
##  3 Newcastle          29    15    11     3    48    21    27    56
##  4 Man United         29    17     5     7    44    37     7    56
##  5 Tottenham          30    16     5     9    55    42    13    53
##  6 Aston Villa        30    14     5    11    41    40     1    47
##  7 Brighton           28    13     7     8    52    36    16    46
##  8 Liverpool          29    12     8     9    50    35    15    44
##  9 Brentford          30    10    13     7    47    40     7    43
## 10 Fullam             29    11     6    12    39    40    -1    39
## 11 Chelsea            30    10     9    11    29    31    -2    39
```

```
## 12 Crystal Palace      30     8     9    13    29    40   -11    33
## 13 Wolves              30     8     7    15    24    42   -18    31
## 14 West Ham            29     8     6    15    27    39   -12    30
## 15 Bournemouth         30     8     6    16    28    57   -29    30
## 16 Leeds               30     7     8    15    39    54   -15    29
## 17 Everton             30     6     9    15    23    43   -20    27
## 18 Nottingham          30     6     9    15    24    54   -30    27
## 19 Leicester           30     7     4    19    40    52   -12    25
## 20 Southampton         30     6     5    19    24    51   -27    23
```

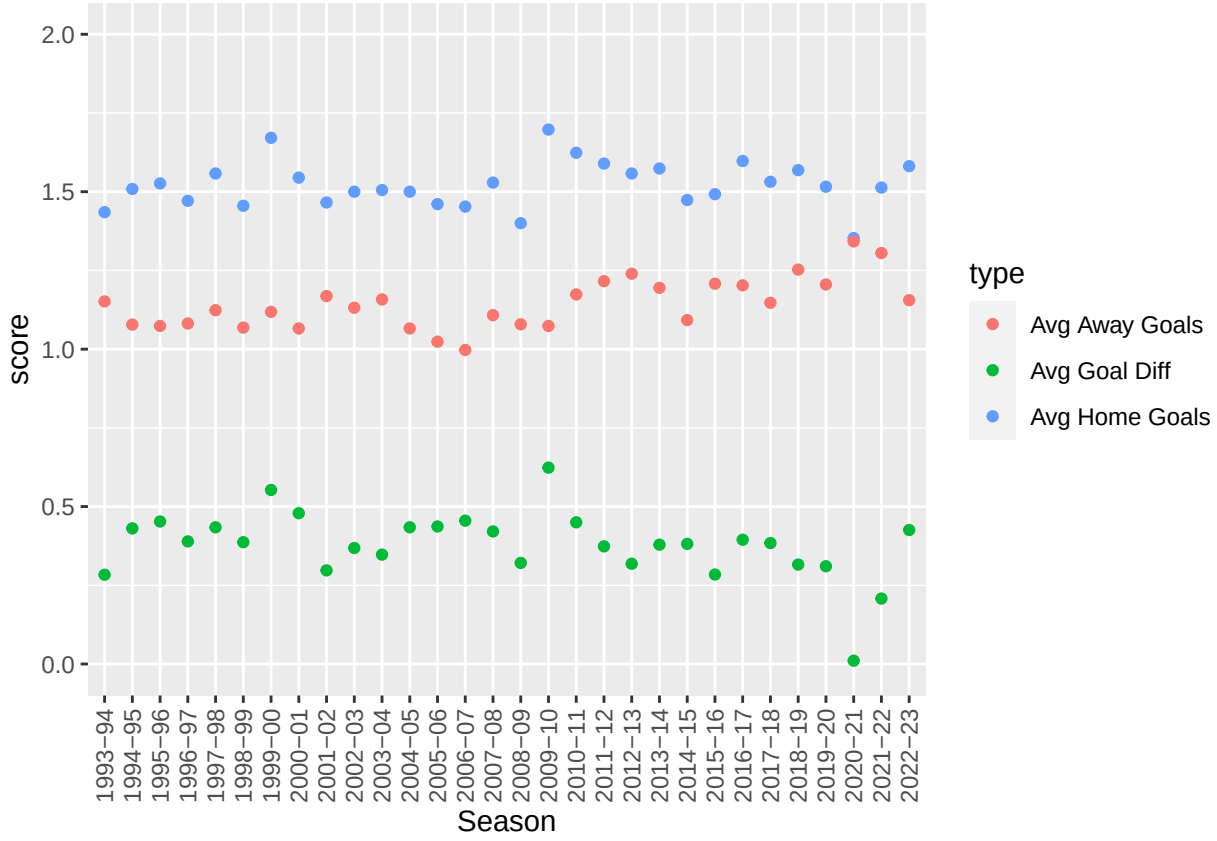
First of all, we would like to investigate the score difference between home and away matches. The so-called Home Field Advantage is a well established phenomenon observed in various sports worldwide and with literature explaining the factors that might contribute to this (5). Nonetheless, it is still a good practice to review the current statistics to inspect visually if such a phenomenon exists in the EPL. Below are the code and results.

```
epl_results %>% mutate(HomeGoals= FTHG, AwayGoals = FTAG) %>%
  pivot_longer(HomeGoals:AwayGoals, values_to = "Goals", names_to = "Type") %>%
  ggplot(aes(Goals, fill = Type)) + geom_bar(position = "dodge") +
  scale_x_discrete(breaks = 0:8,limits=(0:8))
```



The graph clearly suggests a strong Home Field Advantage in the EPL, with the Home Team scores skewed much more to the right than the Away Team scores. Among the speculations as to what constituted to Home Field Advantage, one suggestion is the presence of home fans at the stadium. Therefore, it would be interesting to see how this effect was offset by COVID-19, where during prolonged periods of time teams were forced to play behind close doors, which was largely the case for Season 2020-21 (6).

```
epl_results %>% group_by(Season) %>%
  summarize("Avg Home Goals" = mean(FTHG),"Avg Away Goals" = mean(FTAG),
            "Avg Goal Diff"= mean(FTHG)-mean(FTAG)) %>%
  pivot_longer("Avg Home Goals":"Avg Goal Diff", names_to = "type", values_to = "score") %>%
  ggplot(aes(Season, score, color=type)) + geom_point() +
  scale_x_discrete(guide = guide_axis(angle = 90)) + ylim(0,2)
```



A very interesting pattern emerges from this graph. It shows an almost negligible Home Field Advantage for Season 2020-21 in terms of average home-away team goal difference. Regardless of whether this is indeed attributable to the absence of the crowds, this season shall be excluded in our training data due to its abnormal home field effect. Also data prior to the 2000-01 season are a bit outdated. Therefore we will remove the pre-millennial seasons and use only Seasons 2000-01 to 2020 plus 2021-22 as our training data, while the incomplete Season 2022-23 will be our dataset for evaluation.

```
epl_results <- epl_results %>% filter(Season >="2000-01" & Season != "2020-21")
TrainSeasons <- filter(epl_results, Season!="2022-23") %>% group_by(Season) %>%
  summarize %>% pull(Season)
TestSeason <- "2022-23"
```

With all these parameters defined, we can now inspect the descriptive statistics of both the Home and Away team scores in the training set - namely the mean, median, mode and standard deviation. First let us see the goal averages:

```
epl_results %>% filter(Season %in% TrainSeasons) %>%
  summarize (AvgHomeGoal = mean(FTHG), AvgAwayGoal = mean(FTAG))
```

```
## # A tibble: 1 x 2
##   AvgHomeGoal AvgAwayGoal
##         <dbl>       <dbl>
## 1        1.53        1.15
```

As shown earlier, the home team on average scores more goals than the away team in a match. Next we will check the goal medians:

```
epl_results %>% filter(Season %in% TrainSeasons) %>%
  summarize (MedianHomeGoal = median(FTHG), MedianAwayGoal = median(FTAG))
```

```
## # A tibble: 1 x 2
##   MedianHomeGoal MedianAwayGoal
##            <dbl>          <dbl>
## 1              1              1
```

While there is a difference in the average number of goals scored, the median home and away scores are the same as both distributions skew to the left with both medians residing at the lower end. We then look at the goal distributions of the goals in separate tables:

```
epl_results %>% filter(Season %in% TrainSeasons) %>% .$FTHG %>% table
```

```
## .
##    0    1    2    3    4    5    6    7    8    9
## 1861 2547 1981  984  400  138   47   15    6    1
```

```
epl_results %>% filter(Season %in% TrainSeasons) %>% .$FTAG %>% table
```

```
## .
##    0    1    2    3    4    5    6    7    9
## 2725 2739 1538  673  224   59   20    1    1
```

Again the most frequent scores for both teams are 1 due to skewness of the distribution, though for the away teams it only slightly outnumbers a nil score. As for the goal standard deviations (SD):

```
epl_results %>% filter(Season %in% TrainSeasons) %>%
  summarize (HomeGoalSD = sd(FTHG), AwayGoalSD = sd(FTAG))
```

```
## # A tibble: 1 x 2
##   HomeGoalSD AwayGoalSD
##        <dbl>      <dbl>
## 1       1.30       1.14
```

Because home teams have a wider spread of scores, it is expected that their goal SD is also higher than that of the away teams.

As the aim of this project is to predict the Correct Scores for both home and away teams, which must be in integers, the most suitable estimations at this stage should be the modes. By definition it is the number which appears most frequently for a dataset, so it should give the highest accuracy if we predict with the score modes, which are both 1 in this case, when no other additional information is available. This will give us the naive accuracy.

```
(epl_results %>% filter(Season == TestSeason) %>%
   rename (HomeGoalAccurary=FTHG, AwayGoalAccurary=FTAG) %>%
   select(HomeGoalAccurary, AwayGoalAccurary) == 1) %>% colMeans %>% print %>% prod
```

```
## HomeGoalAccurary AwayGoalAccurary
##           0.3209           0.3277
```

```
## [1] 0.1052
```

We can see both home and away score predictions have an accuracy rate of around 32-33%, with an overall accuracy of roughly 10.5%. In other words, we can expect to guess about 1 in 10 times correctly if we guess every single match as 1-1. In order to beat the bookies, we will have to raise this figure significantly.

Instead of working on the accuracy, however, we would use the Root Mean Square Error (RMSE) of the predictions to guide our training. It shows the degree to which how close (or far off) our predictions are when compared to the actual figures, so we can gauge the performance in tiny amount. Similar to the accuracy, the naive RMSE can be calculated to by guessing 1-1 for every match.

```
(epl_results %>% filter(Season == TestSeason) %>%
   rename (HomeGoalRMSE=FTHG, AwayGoalRMSE=FTAG) %>%
   select(HomeGoalRMSE, AwayGoalRMSE) - 1)^2 %>%
  colMeans(na.rm=T) %>% sqrt %>% print %>% sum
```

```
## HomeGoalRMSE AwayGoalRMSE
##        1.527        1.142
```

```
## [1] 2.669
```

If allowed to guess with decimal places, such figures would be close to their standard deviations; yet as the scores can be only predicted in integers the RMSE would inevitably be higher. Although a smaller RMSE does not necessarily imply better accuracy on a minute scale, by improving this we shall be able to nudge closer to the actual figures and increase the accuracy.

To forecast match scores from past results, we would first create data frames which contain scores of both home and away teams, and then build regression models. We would also try random forests to and compare the results. The final accuracy will be evaluated by the test set, i.e. matches of the current 2022-23 season, to see if either algorithm could increase the success rate of the Correct Score prediction.

Results

As each row of the epl_results data frame only contains results of a single match, we need to wrangle the data to include the past scores. To keep the number of predictors at a reasonable level, we only include results from the 20 past matches.

```
n=20
```

In addition, to make the variable names as succinct as possible the following abbreviations are used:

H = Home team (1st place) /home match (2nd place) A = Away team (1st place) /away match (2nd place) G = Goals scored g = Goals conceded X = head-to-head encounters Number suffix = order of previous matches under the same conditions

For example, HHG02 would be the number of goals scored by the home team in their 2nd last home match; AHg05 would be the number of goals conceded by the away team in their 5th last home match. Starting with past home results of the home teams and past away results of the away teams, the data frame RcntHA is created:

```
RcntHA <- epl_results %>%
  rename(TmpSeason = Season, MatchDate=DateTime, TmpAwayTeam=AwayTeam,
         HG=FTHG, AG=FTAG) %>% # Change conflicting colnames
  inner_join(epl_results, c("HomeTeam")) %>% filter(MatchDate>DateTime) %>%
  group_by(HomeTeam, MatchDate) %>% slice_max(DateTime, n=n) %>%
  mutate(Match=str_pad(rank(-rank(DateTime)), 2, pad=0)) %>% ungroup %>%
  select(-Season, -DateTime,-AwayTeam) %>% # Create home team home match results
  rename(AwayTeam = TmpAwayTeam, TmpHomeTeam = HomeTeam, HHG = FTHG, HHg = FTAG) %>%
  pivot_wider(names_from = Match, names_sep = "", values_from = c(HHG, HHg)) %>%
  inner_join(epl_results, c("AwayTeam")) %>% filter(MatchDate>DateTime) %>%
  group_by(AwayTeam, MatchDate) %>% slice_max(DateTime, n=n) %>%
  mutate(Match=str_pad(rank(-rank(DateTime)), 2, pad=0)) %>% ungroup %>%
  select(-Season, -DateTime, -HomeTeam) %>% # Create away team away match results
  rename(AAG = FTAG, AAg = FTHG) %>%
  pivot_wider(names_from = Match, names_sep = "", values_from = c(AAG, AAg)) %>%
  rename_at(vars(starts_with("Tmp")),~str_replace(.,"Tmp","")) %>% # Restore conflicting colnames
  arrange(Season, HomeTeam, MatchDate)
```

However, 20 past matches are only an arbitrary number and do not necessarily constitute to the best predictors. After all, it should be the recent performances of the teams which count the most, while results of the distant past may not be of much significance. In order to determine the optimal number of matches to be included, we will use cross-validation to tune that parameter. And to exclude rows with no data for the past 20 matches, we first determine the maximum size of the dataset.

```
set_sze <- RcntHA %>% filter(Season %in% TrainSeasons)%>% select(matches("HHG|AAG")) %>%
  is.na %>% rowSums %>% {.==0} %>% sum
```

Then the following function will calculate the average RMSE with 100 Monte Carlo simulations via cross-validation.

```
RcntHA_RMSE <- function(i) rowMeans( # average the RMSE
  sapply (1:100, function (repetitions) { # Perform Monte Carlo simulations
    set_idx <- RcntHA %>% filter(Season %in% TrainSeasons) %>%
      select(matches(paste0("(HHG|AAG)",str_pad(1:i, 2, pad=0)))) %>%
      is.na %>% rowSums %>% {which(.==0)} %>% sample(set_sze) # Exclude rows with missing data
    val_idx <- sample(set_idx, set_sze * 0.1) # Create the validation set
    train_idx <- setdiff(set_idx, val_idx) # Create the training set
    linear_model <- lm(cbind(HG,AG) ~ ., data = RcntHA %>% # Build linear model
                         filter(Season %in% TrainSeasons) %>% .[train_idx,] %>%
                         select(matches(paste0("(HHG|AAG)", str_pad(1:i, 2, pad=0))), HG, AG))
    (predict(linear_model, RcntHA %>% filter(Season %in% TrainSeasons) %>% .[val_idx,])
      - RcntHA %>% filter(Season %in% TrainSeasons) %>% .[val_idx,] %>% select(HG,AG)) ^2 %>%
      colMeans %>%sqrt})) # Predict scores with the model & compute its RMSE
```

We will apply 1 to 20 matches to the above function to see which one produces the least overall RMSE.

```
sapply(1:n, RcntHA_RMSE) %>% as.data.frame() %>% rename_all(~paste(1:n, "match(es)")) %>%
  t %>% as.data.frame() %>% mutate(Total = HG + AG)
```

```
##                  HG    AG Total
## 1 match(es)  1.277 1.127 2.404
## 2 match(es)  1.282 1.120 2.402
```

```
## 3 match(es)  1.272 1.108 2.380
## 4 match(es)  1.260 1.111 2.370
## 5 match(es)  1.262 1.108 2.370
## 6 match(es)  1.253 1.109 2.362
## 7 match(es)  1.250 1.109 2.359
## 8 match(es)  1.245 1.104 2.349
## 9 match(es)  1.246 1.107 2.353
## 10 match(es) 1.248 1.101 2.349
## 11 match(es) 1.235 1.107 2.342
## 12 match(es) 1.239 1.110 2.349
## 13 match(es) 1.236 1.107 2.343
## 14 match(es) 1.240 1.102 2.342
## 15 match(es) 1.230 1.101 2.331
## 16 match(es) 1.228 1.100 2.329
## 17 match(es) 1.235 1.101 2.337
## 18 match(es) 1.235 1.107 2.342
## 19 match(es) 1.226 1.094 2.320
## 20 match(es) 1.231 1.099 2.330
```

As 17 matches produce least RMSE, we will use that as our parameter in our linear regression model and evaluate its RMSE with the test dataset.

```
linear_model <- lm(cbind(HG,AG) ~ ., data = RcntHA %>% filter(Season %in% TrainSeasons) %>%
                     select(matches(paste0("(HHG|AAG)",str_pad(1:17, 2, pad=0))), HG, AG))
```

```
(predict(linear_model, RcntHA %>% filter(Season==TestSeason))
 - RcntHA %>% filter(Season==TestSeason) %>% select(HG,AG))^2 %>%
  colMeans(na.rm = T) %>% sqrt %>% print %>% sum
```

```
##    HG    AG
## 1.336 1.140
```

```
## [1] 2.477
```

It can be seen that the RMSE for the home score is reduced by a significant extent, while that of away score barely changes. In the next section, we will include results of counter home/away conditions, i.e. past away results of the home teams and past home results of the away teams, to see if we can further improve the performance.

Although past match results of the teams in question under the same home/away condition should be most relevant, results of their counter home/away conditions may also hold some weight as for how the teams are performing in general. In view of this, the data frame RcntHAAH is created from counter joining the home and away teams in the RcntHA data frame, so each team's previous counter home/away scores are also included in addition to their results under the same conditions.

```
RcntHAAH <- RcntHA %>%
  rename_all(~paste0("Tmp",.)) %>%  # Change conflicting colnames
  inner_join(RcntHA, c("TmpHomeTeam"="AwayTeam")) %>% filter(TmpMatchDate>MatchDate) %>%
  group_by(TmpHomeTeam, TmpMatchDate) %>% slice_max(MatchDate) %>% ungroup %>%
```

```
  rename(HAG01 = AG, HAg01 = HG) %>% select(matches("Tmp|HA|AA")) %>%
  rename_at(vars(starts_with("AA")), # Create home team away match results
            ~paste0("HA", str_sub(.,3,3), str_pad(as.numeric(str_sub(.,4))+1, 2, pad=0))) %>%
  inner_join(RcntHA, c("TmpAwayTeam"="HomeTeam")) %>% filter(TmpMatchDate>MatchDate) %>%
  group_by(TmpAwayTeam, TmpMatchDate) %>% slice_max(MatchDate) %>% ungroup %>%
  rename(AHG01 = HG, AHg01 = AG) %>% select(matches("Tmp|HA|AH|HH")) %>%
  rename_at(vars(starts_with("HH")), # Create away team home match results
            ~paste0("AH", str_sub(.,3,3), str_pad(as.numeric(str_sub(.,4))+1, 2, pad=0))) %>%
  rename_at(vars(starts_with("Tmp")),~str_replace(.,"Tmp","")) %>% # Restore conflicting colnames
  select(Season, MatchDate, HomeTeam, AwayTeam, HG, AG, matches("HH|AA|HA|AH")) %>%
  arrange(HomeTeam, MatchDate) # Sort the variables and records
```

Notice because we are using the existing RcntHA data frame to append the past records, the "current" home/away match in that appending data frame is actually a previous match for the team concerned, hence there are 21 past match results rather than 20 in this group. And again we will use cross-validation to tune the optimal number of matches, and determine the maximum size of the dataset by excluding rows with no data for any of the training variables.

```
set_sze <- RcntHAAH %>%filter(Season %in% TrainSeasons)%>% select(matches("HHG|AAG|HAG|AHG")) %>%
  is.na %>% rowSums %>% {.==0} %>% sum
```

The following function will calculate the average RMSE with 100 Monte Carlo simulations via cross-validation.

```
RcntHAAH_RMSE <- function(i) rowMeans( # average the RMSE
  sapply (1:100, function (repetitions) { # Perform Monte Carlo simulations
    set_idx <- RcntHAAH %>% filter(Season %in% TrainSeasons) %>%
      select(matches(paste0("(HHG|AAG)",str_pad(1:17, 2, pad=0))),
             matches(paste0("(HAG|AHG)",str_pad(1:i, 2, pad=0)))) %>%
      is.na %>% rowSums %>% {which(.==0)} %>% sample(set_sze) # Exclude rows with missing data
    val_idx <- sample(set_idx, set_sze * 0.1) # Create the validation set
    train_idx <- setdiff(set_idx, val_idx) # Create the training set
    linear_model <- lm(cbind(HG,AG) ~ ., data = RcntHAAH %>% # Build linear model
                         filter(Season %in% TrainSeasons) %>% .[train_idx,] %>%
                         select(matches(paste0("(HHG|AAG)",str_pad(1:17, 2, pad=0))),
                                matches(paste0("(HAG|AHG)",str_pad(1:i, 2, pad=0))), HG, AG))
    (predict(linear_model, RcntHAAH %>% filter(Season %in% TrainSeasons) %>% .[val_idx,])
      - RcntHAAH %>% filter(Season %in% TrainSeasons) %>% .[val_idx,] %>% select(HG,AG)) ^2 %>%
      colMeans %>%sqrt})) # Predict scores with the model & compute its RMSE
```

Similarly, we will apply 1 to 20 matches to the above function to see which one produces the least overall RMSE.

```
sapply(1:n, RcntHAAH_RMSE) %>% as.data.frame() %>% rename_all(~paste(1:n, "match(es)")) %>%
  t %>% as.data.frame() %>% mutate(Total = HG + AG)
```

```
##                 HG    AG Total
## 1 match(es)  1.231 1.096 2.327
## 2 match(es)  1.231 1.097 2.328
## 3 match(es)  1.232 1.098 2.330
## 4 match(es)  1.234 1.100 2.334
## 5 match(es)  1.229 1.102 2.330
```

```
## 6 match(es)  1.224 1.100 2.324
## 7 match(es)  1.228 1.093 2.321
## 8 match(es)  1.235 1.096 2.331
## 9 match(es)  1.226 1.101 2.327
## 10 match(es) 1.226 1.092 2.319
## 11 match(es) 1.227 1.092 2.319
## 12 match(es) 1.228 1.097 2.325
## 13 match(es) 1.230 1.102 2.332
## 14 match(es) 1.226 1.103 2.328
## 15 match(es) 1.232 1.100 2.332
## 16 match(es) 1.232 1.101 2.333
## 17 match(es) 1.229 1.095 2.324
## 18 match(es) 1.236 1.092 2.328
## 19 match(es) 1.233 1.097 2.330
## 20 match(es) 1.233 1.088 2.321
```

As 13 matches produce least RMSE, we will use that as our parameter in our linear regression model and evaluate its RMSE with the test dataset.

```
linear_model <- lm(cbind(HG,AG) ~ ., data = RcntHAAH %>% filter(Season %in% TrainSeasons) %>%
                     select(matches(paste0("(HHG|AAG)",str_pad(1:17, 2, pad=0))),
                            matches(paste0("(HAG|AHG)",str_pad(1:13, 2, pad=0))), HG, AG))
```

```
(predict(linear_model, RcntHAAH %>% filter(Season==TestSeason))
 - RcntHAAH %>% filter(Season==TestSeason) %>% select(HG,AG)) ^2 %>%
  colMeans(na.rm = T) %>%  sqrt %>% print %>% sum
```

```
##    HG    AG
## 1.321 1.144
```

```
## [1] 2.465
```

While the RMSE for the home score is reduced by another slight extent, that of away score actually increases a little. Nonetheless, as the overall RMSE still decreases we will keep this model as our best estimate at the moment. In the next section, we will include results of head-to-head encounters of the teams, both home and away, to see if we can further improve the performance.

Although we might want to see if a longer head-to-head history would provide better insight into our prediction, the number of available training records decreases dramatically as we increase the number of past encounters:

```
epl_results %>% select(DateTime, HomeTeam, AwayTeam) %>% rename(MatchDate = DateTime) %>%
  inner_join(epl_results, c("HomeTeam", "AwayTeam")) %>% filter(MatchDate > DateTime) %>%
  group_by(MatchDate, HomeTeam, AwayTeam) %>% slice_max(DateTime, n=5) %>%
  mutate(HHX = rank(-rank(DateTime))) %>% ungroup %>%
  group_by(HHX) %>% summarize(count=n())
```

```
## # A tibble: 5 x 2
##     HHX count
```

```
##   <dbl> <int>
## 1     1  6732
## 2     2  5581
## 3     3  4664
## 4     4  3896
## 5     5  3209
```

Therefore, we will only stick to two last head-to-head encounters between the teams, one under the same home/away condition and one under the opposite. These results will be denoted with an X between the match type (H/A) and goal type (G/g), and with just the digit 1 at the end as only the last head-to-head result is retrieved. For example, HHXg1 is the number of goals conceded by the home team in their last home match against the same opponents; AHXG1 is the number of goals scored by the away team in their last home match when their opponents visited.

The following code will create the results of these encounters and append them to the existing RcntHAAH data frame.

```
RcntHAAHX <- epl_results %>% select(DateTime, HomeTeam, AwayTeam) %>% rename(MatchDate=DateTime) %>%
  inner_join(epl_results, c("HomeTeam", "AwayTeam")) %>%
  filter(MatchDate>DateTime) %>% group_by(MatchDate, HomeTeam, AwayTeam) %>%
  slice_max(DateTime, n=1) %>% mutate(Match=1) %>% ungroup %>% # Create same home/away results
  select(-Season, -DateTime) %>% rename (HHXG = FTHG, HHXg = FTAG) %>%
  pivot_wider(names_from = Match,  names_sep = "", values_from = c(HHXG, HHXg)) %>%
  inner_join(epl_results, c("AwayTeam"="HomeTeam", "HomeTeam"="AwayTeam")) %>%
  filter(MatchDate>DateTime) %>% group_by(MatchDate, HomeTeam, AwayTeam)%>%
  slice_max(DateTime, n=1) %>% mutate(Match=1) %>% ungroup %>% # Create counter home/away results
  select(-Season, -DateTime) %>% rename (AHXG = FTHG, AHXg = FTAG) %>%
  pivot_wider(names_from = Match,  names_sep = "", values_from = c(AHXG, AHXg)) %>%
  right_join(RcntHAAH, c("MatchDate","HomeTeam", "AwayTeam")) %>%
  select(Season, MatchDate, HomeTeam, AwayTeam, HG, AG,
         starts_with(c("HHXG","AHXG","HHG","AAG","HAG","AHG")))
```

As we have decided to include only the last two encounters at maximum, there is no parameter to tune. Instead, we would like to see if the RMSE can be further improved by including these variables into our model. If not, we might just leave them out.

```
linear_model <- lm(cbind(HG,AG) ~ ., data = RcntHAAHX %>% filter(Season %in% TrainSeasons) %>%
                     select(matches(paste0("(HHG|AAG)",str_pad(1:17, 2, pad=0))),
                            matches(paste0("(HAG|AHG)",str_pad(1:13, 2, pad=0))),
                            matches("X"), HG, AG))
```

```
(predict(linear_model, RcntHAAHX %>% filter(Season==TestSeason))
 - RcntHAAHX %>% filter(Season==TestSeason) %>% select(HG,AG)) ^2 %>%
  colMeans(na.rm = T) %>%  sqrt  %>% print %>% sum
```

```
##    HG    AG
## 1.323 1.145
```

```
## [1] 2.468
```

Unfortunately, despite inclusion of the extra match results in the model its prediction performance does not get any better. Therefore, we will ditch the head-to-head results and use only the previous model as our final algorithm. We will restore it and this time evaluate the performance with prediction accuracy.

```
linear_model <- lm(cbind(HG,AG) ~ ., data = RcntHAAH %>% filter(Season %in% TrainSeasons) %>%
                     select(matches(paste0("(HHG|AAG)",str_pad(1:17, 2, pad=0))),
                            matches(paste0("(HAG|AHG)",str_pad(1:13, 2, pad=0))), HG, AG))
```

```
(predict(linear_model, RcntHAAH %>% filter(Season=="2022-23")) %>% round
 == RcntHAAH %>% filter(Season=="2022-23") %>% select(HG,AG)) %>%
  colMeans(na.rm = T) %>%  print %>% prod
```

```
##     HG     AG
## 0.3013 0.2720
```

```
## [1] 0.08193
```

Ironically, despite decrease in the RMSE of the predicted scores, the prediction accuracy on the contrary worsens significantly compared to the naive accuracy. In other words, our goal does not fare well. However, before we make such a conclusion we would try one more algorithm - the random forests - to see if they would be able to make any better results.

Random forests have been hailed as a very powerful algorithm in machine learning. By averaging results of an ensemble of decision trees, the algorithm can make predictions which are rather immune to sample bias, as compared to just using one decision tree. This can well be summed up by the statement: A large number of relatively uncorrelated trees operating as a committee will outperform any individual one (7). The use of bootstrap aggregation and the arbitrary selection of features in each decision tree create the randomness necessary for the trees to de-correlate with each other.

In relation to the current project, we will use the most popular package randomForest, already been loaded at the beginning, to try predicting the Correct Scores. To see if we can get the best prediction results with the most variables, though it may not necessarily be the case, we include every variable gathered in the data frame RcntHAAHX. That is, we would not shortlist any of the past matches as did in the linear models, and we would include the latest head-to-head encounter results as well.

Before so, we will have to decide on the parameter mtry - the number of random features to be selected for each decision tree. Although the algorithm already has a default value for it (total no. of predictors/3 rounded down to the integer, which in this case is $168/3 = 56$) (8), we would still like to see if its performance can be boosted with a different mtry. The randomForest package has a built-in function tuneRF suited for the task. Yet as in cross-validation of the linear models, we need to create a dataset with no missing data for the training variables.

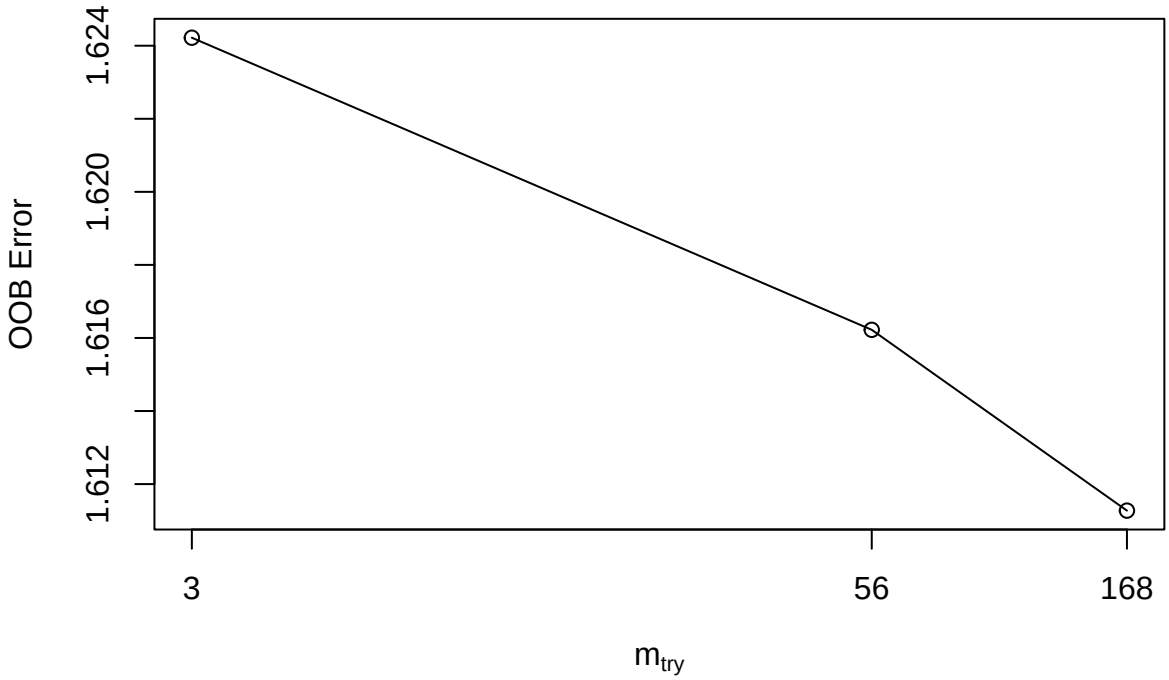
```
RF_all <- RcntHAAHX %>% filter(Season %in% TrainSeasons) %>%
  select(HG, AG, matches("(HHG|AAG)"), matches("(HAG|AHG)"), matches("X")) %>%
  filter(rowSums(is.na(.))==0)
```

Then we will tune the mtry using the filtered dataset RF_all. To reduce the computation time, the mtry parameter will increase by 20 for every trial.

```
tuneRF(select(RF_all, -HG, -AG), RF_all$HG, ntreeTry = 50, stepFactor = 20)
```

```
## mtry = 56  OOB error = 1.616
## Searching left ...
## mtry = 3     OOB error = 1.624
```

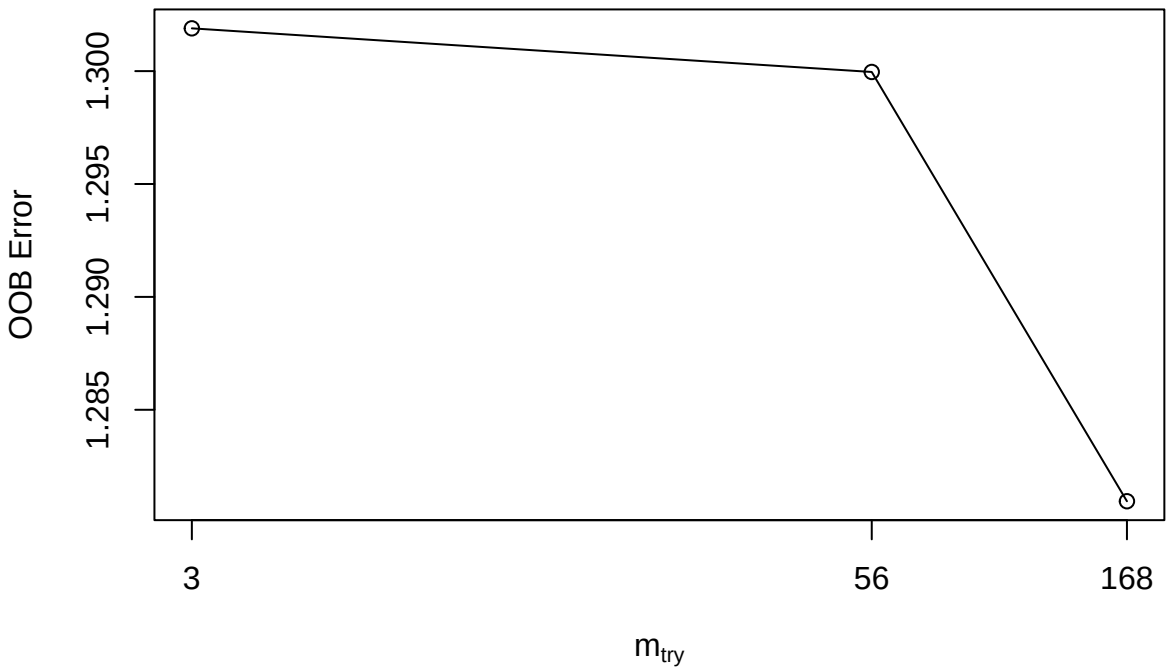
```
## -0.004947 0.05
## Searching right ...
## mtry = 168   OOB error = 1.611
## 0.003062 0.05
```




```
##     mtry OOBError
## 3      3    1.624
## 56    56    1.616
## 168  168    1.611
```

```
tuneRF(select(RF_all, -HG, -AG), RF_all$AG, ntreeTry = 50, stepFactor = 20)
```

```
## mtry = 56  OOB error = 1.3
## Searching left ...
## mtry = 3     OOB error = 1.302
## -0.001486 0.05
## Searching right ...
## mtry = 168   OOB error = 1.281
## 0.01462 0.05
```



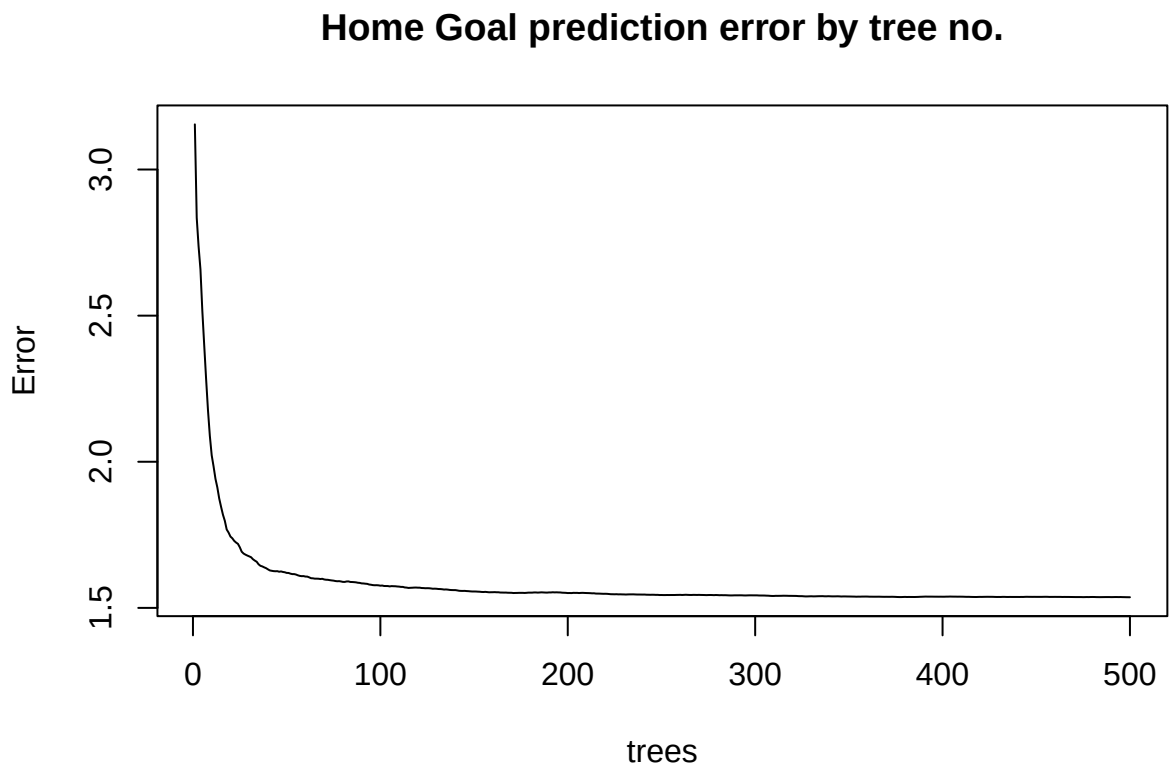

```
##     mtry OOBError
## 3      3    1.302
## 56    56    1.300
## 168  168    1.281
```

Tuning results support including 168 predictors, i.e. all parameters for both randaom forest models! Taking heed of this, we now create models to predict the home and away scores.

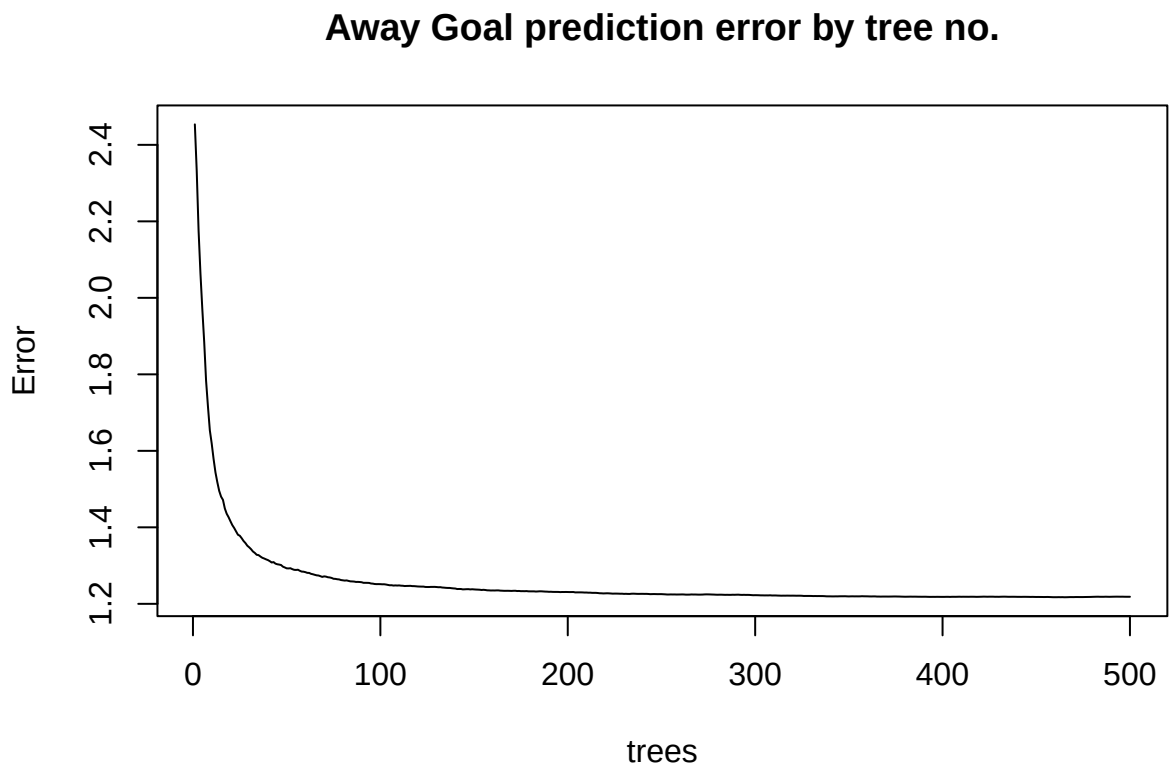
```
HG_rf <- randomForest(HG ~ .,  data = RF_all %>% select(-AG), mtry=168)
AG_rf <- randomForest(AG ~ .,  data = RF_all %>% select(-HG), mtry=168)
```

The default number of tress (ntree) used by the algorithm is 500. Plotting the errors against this parameter tells if that would suffice for the models.

```
plot(HG_rf, main = "Home Goal prediction error by tree no.")
```




```
plot(AG_rf, main = "Away Goal prediction error by tree no.")
```



As the graphs show the errors have more or less stabilized by 500 trees, we would just keep that number and not tune it further. With the parameters determined, let us see if we can predict better with random forests.

```
(predict(HG_rf, RcntHAAHX %>% filter(Season==TestSeason))
 - RcntHAAHX %>% filter(Season==TestSeason) %>% select(HG)) ^2 %>%
  colMeans(na.rm = T) %>%  sqrt
```

```
##    HG
## 1.325
```

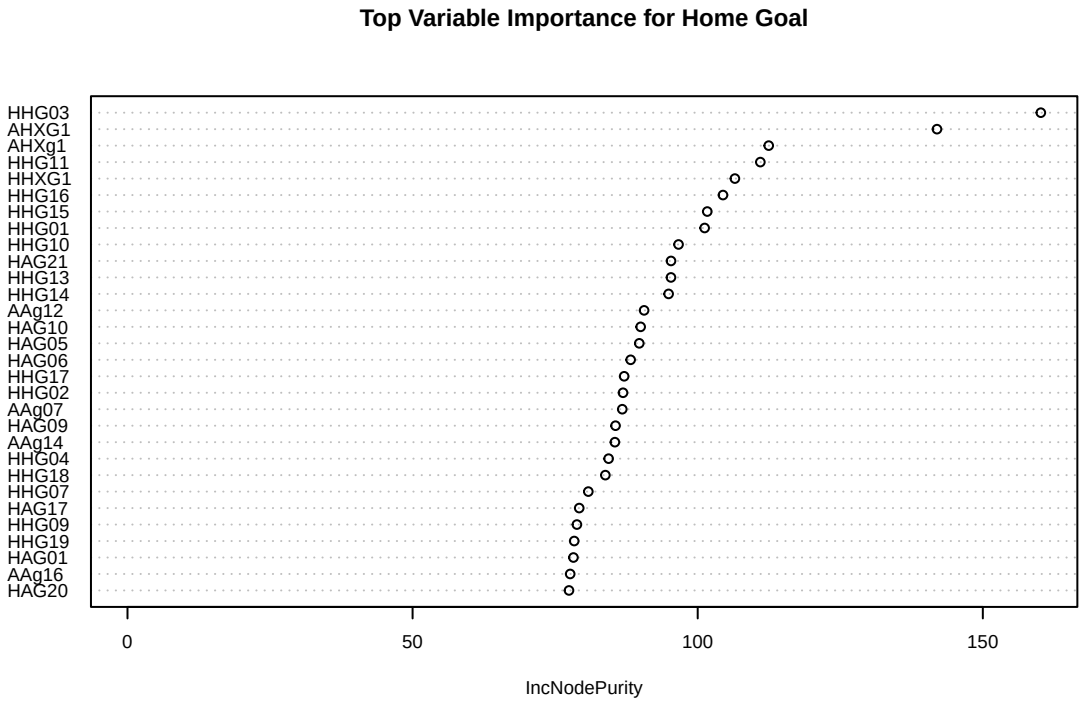
```
(predict(AG_rf, RcntHAAHX %>% filter(Season==TestSeason))
  - RcntHAAHX %>% filter(Season==TestSeason) %>% select(AG)) ^2 %>%
  colMeans(na.rm = T) %>%  sqrt
```

```
##    AG
## 1.135
```

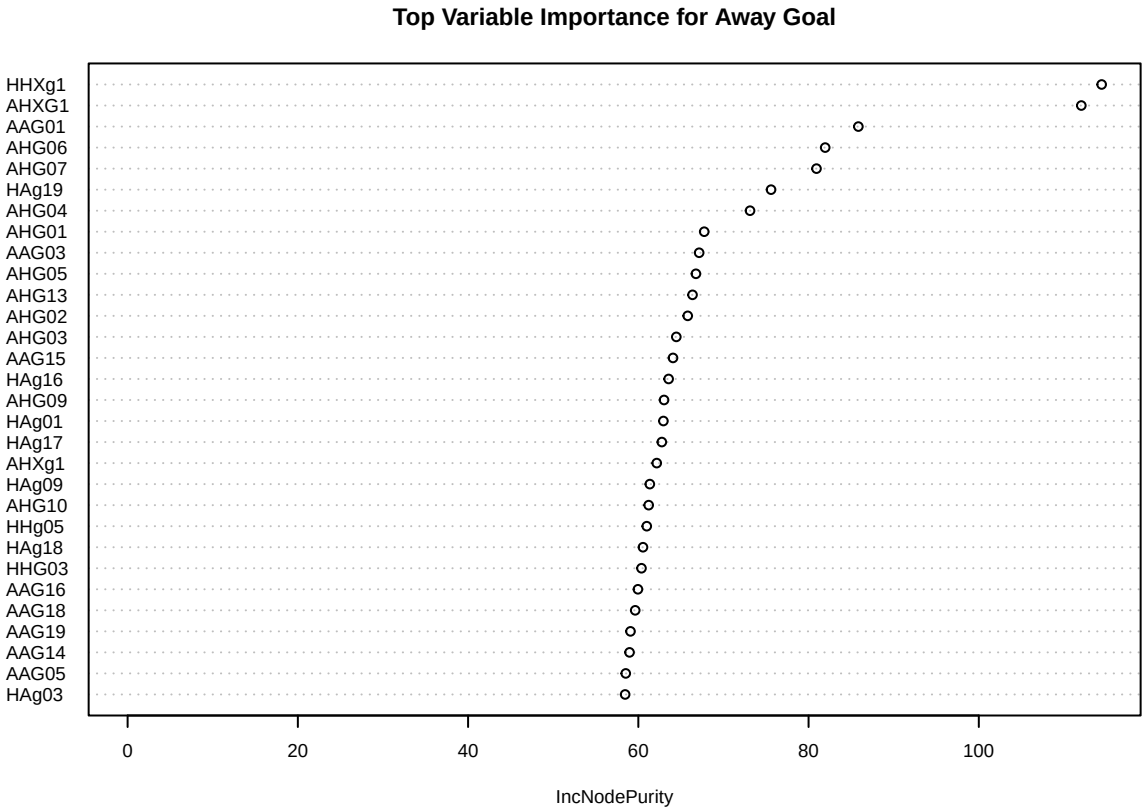
The RMSE of HG is slightly higher than that in the RcntHAAH linear model, while that of AG is slightly lower - it is a close call!

In this final attempt, we shall look at the variable importance in the forest models to see if we can be more selective of the predictors used.

```r
varImpPlot(HG_rf, main = "Top Variable Importance for Home Goal", cex = 0.6)
```




```r
varImpPlot(AG_rf, main = "Top Variable Importance for Away Goal", cex = 0.6)
```



It can be seen from the graphs some variables are indeed more essential than others in predicting the scores. For example, goals scored by home team in their past home matches hold more weight in predicting their goals to be scored than their away counterparts, whose predictor weights are more widely spread. To see if we can improve the RMSE by selecting more distinctive features, we will filter variables which rank top 100 in terms of their importance, then used this subset to rerun the models.

```
RF_shtlst_HG <- select(RF_all, HG, importance(HG_rf) %>% as.data.frame %>%
                         slice_max(order_by = IncNodePurity, n=100) %>% rownames)
RF_shtlst_AG <- select(RF_all, AG, importance(AG_rf) %>% as.data.frame %>%
                         slice_max(order_by = IncNodePurity, n=100) %>% rownames)
```

Afterwards the forest models are created with the shortlisted variables again, and the RMSE recomputed.

```
HG_rf <- randomForest(HG ~ .,  data = RF_shtlst_HG)
AG_rf <- randomForest(AG ~ .,  data = RF_shtlst_AG)
```

```
(predict(HG_rf, RcntHAAHX %>% filter(Season==TestSeason))
 - RcntHAAHX %>% filter(Season==TestSeason) %>% select(HG)) ^2 %>%
  colMeans(na.rm = T) %>%  sqrt
```

```
##    HG
## 1.322
```

```
(predict(AG_rf, RcntHAAHX %>% filter(Season==TestSeason))
  - RcntHAAHX %>% filter(Season==TestSeason) %>% select(AG)) ^2 %>%
  colMeans(na.rm = T) %>%  sqrt
```

```
##   AG
## 1.14
```

The overall results are more or less the same as before. We would stick to these models, but instead of computing the RMSE we would evaluate with the accuracy as this is what counts as a successful Correct Score Prediction.

```
HG_Accy <- (predict(HG_rf, RcntHAAHX %>% filter(Season==TestSeason)) %>% round ==
              RcntHAAHX %>% filter(Season==TestSeason) %>% select(HG)) %>% mean(na.rm = T)
AG_Accy <- (predict(AG_rf, RcntHAAHX %>% filter(Season==TestSeason)) %>% round ==
              RcntHAAHX %>% filter(Season==TestSeason) %>% select(AG)) %>% mean(na.rm = T)
data.frame(Category = c("Home Goal", "Away Goal", "Overall"),
           Accuracy = c(HG_Accy, AG_Accy, HG_Accy * AG_Accy))
```

```
##    Category Accuracy
## 1 Home Goal   0.3504
## 2 Away Goal   0.3034
## 3   Overall   0.1063
```

Unfortunately, despite the slight increase in the accuracy compared to the linear models, it is still not better than the naive accuracy computed at the beginning. Therefore the goal of this project cannot be achieved.

Conclusion The failure to improve the Correct Score prediction accuracy despite a comprehensive database is rather disappointing. While it might be tempting to believe the statistical models fail completely, the contradictory decrease in RMSE suggests not all effort is futile. Using the last forest models, perhaps a look of the confusion matrices would shed some light as to how the predictions fail:

```
confusionMatrix(predict(HG_rf, RcntHAAHX %>% filter(Season==TestSeason)) %>% round %>% factor,
                RcntHAAHX %>% filter(Season==TestSeason) %>% .$HG %>% factor)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5  6  7  9
##          0  0  0  0  0  0  0  0  0  0
##          1 33 45 14 10  2  1  0  0  0
##          2 23 31 35 15 12  2  1  1  1
##          3  0  3  0  2  2  0  1  0  0
##          4  0  0  0  0  0  0  0  0  0
##          5  0  0  0  0  0  0  0  0  0
##          6  0  0  0  0  0  0  0  0  0
##          7  0  0  0  0  0  0  0  0  0
##          9  0  0  0  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.35
```

```
##                  95% CI : (0.289, 0.415)
##     No Information Rate : 0.338
##     P-Value [Acc > NIR] : 0.362
##
##                   Kappa : 0.118
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5 Class: 6
## Sensitivity             0.000    0.570    0.714  0.07407   0.0000   0.0000  0.00000
## Specificity             1.000    0.613    0.535  0.97101   1.0000   1.0000  1.00000
## Pos Pred Value            NaN    0.429    0.289  0.25000      NaN      NaN      NaN
## Neg Pred Value          0.761    0.736    0.876  0.88938   0.9316   0.9872  0.99145
## Prevalence              0.239    0.338    0.209  0.11538   0.0684   0.0128  0.00855
## Detection Rate          0.000    0.192    0.150  0.00855   0.0000   0.0000  0.00000
## Detection Prevalence    0.000    0.449    0.517  0.03419   0.0000   0.0000  0.00000
## Balanced Accuracy       0.500    0.591    0.625  0.52254   0.5000   0.5000  0.50000
##                      Class: 7 Class: 9
## Sensitivity           0.00000  0.00000
## Specificity           1.00000  1.00000
## Pos Pred Value            NaN      NaN
## Neg Pred Value        0.99573  0.99573
## Prevalence            0.00427  0.00427
## Detection Rate        0.00000  0.00000
## Detection Prevalence  0.00000  0.00000
## Balanced Accuracy     0.50000  0.50000
```

```
confusionMatrix(predict(AG_rf, RcntHAAHX %>% filter(Season==TestSeason)) %>% round %>% factor,
                RcntHAAHX %>% filter(Season==TestSeason) %>% .$AG %>% factor)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1  2  3  4  5
##          0  0  0  0  0  0  0
##          1 65 57 38  8  6  2
##          2 14 18 13  8  4  0
##          3  0  0  0  1  0  0
##          4  0  0  0  0  0  0
##          5  0  0  0  0  0  0
##
## Overall Statistics
##
##                Accuracy : 0.303
##                  95% CI : (0.245, 0.367)
##     No Information Rate : 0.338
##     P-Value [Acc > NIR] : 0.881
##
##                   Kappa : 0.013
##
##  Mcnemar's Test P-Value : NA
##
```

```
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity             0.000    0.760   0.2549  0.05882   0.0000  0.00000
## Specificity             1.000    0.252   0.7596  1.00000   1.0000  1.00000
## Pos Pred Value            NaN    0.324   0.2281  1.00000      NaN      NaN
## Neg Pred Value          0.662    0.690   0.7853  0.93133   0.9573  0.99145
## Prevalence              0.338    0.321   0.2179  0.07265   0.0427  0.00855
## Detection Rate          0.000    0.244   0.0556  0.00427   0.0000  0.00000
## Detection Prevalence    0.000    0.752   0.2436  0.00427   0.0000  0.00000
## Balanced Accuracy       0.500    0.506   0.5072  0.52941   0.5000  0.50000
```

One interesting pattern observed from the above matrices is that, the home score model does succeed in predicting the majority of 1 goal and 2 goals correctly, with sensitivity above 50% in both classes. Nonetheless, the models fail to predict nil score at all, which constitute to almost one-fourth of the home team scores and even more so for away team scores, thereby seriously undermining prediction accuracy. One reason is the skewness of the distributions plays a significant part in swaying the predictions away from zero. The RMSE concerns by how much the predictions would deviate from the actual scores, the more the deviation the higher the RMSE; in contrast, the accuracy only concerns whether such estimations are exactly correct; it does not make a difference if they miss by an inch or a mile. As a result a discrepancy exists between the two parameters during the training process.

Moreover, if the exact scores of a football match in reality depends on many random factors, or variables beyond the dataset, most importantly luck, it is not quite possible to predict them with an unreasonably level of accuracy. An analogy would be that while we can forecast weather temperatures within a range with confidence, it would not help if the correct predictions have to be within 0.1 degrees Celsius. The random noise there simply far outweighs the observable signals on such a minute scale.

Despite the unsatisfactory outcome, this is still a very meaningful project, with a lot of statistical knowledge, machine learning algorithms and coding skills being employed along the way. Perhaps if time allows in the future, more variables can be added to see if the random forests can make better forecasts. On the other hand though, trying to out-win the bookies by Correct Scores may not be a realistic idea due to the above rationale. Perhaps Full-Time Result or Total Goals, which allow a large margin of randomness, are more suitable candidates for the challenge.

References 1. Premier League viewership and online betting numbers https://www.sportsmole.co.uk/football/features/premier-league-viewership-and-online-betting-numbers_506592.html

2. Betting on the English Premier League https://towardsdatascience.com/betting-on-the-english-premier-league-making-money-with-machine-learning-fb6938760c64

3. English Premier League (EPL) Results https://www.kaggle.com/datasets/irkaal/english-premier-league-results

4. While the portal of the website is https://www.premierleague.com, specific details of each match can be retrieved choosing Premier League -> Results in the menu bar than filter the results by Season.

5. Is There An Actual Home Field Advantage When A Sports Team Plays In Their Home Stadium? https://www.scienceabc.com/social-science/is-there-an-actual-home-field-advantage-when-a-sports-team-plays-in-their-home-stadium.html

6. How has the COVID-19 pandemic affected Premier League football? https://www.premierleague.com/news/1682374

7. The Random Forest Classifier https://towardsdatascience.com/understanding-random-forest-58381e0602d2

8. randomForest: Classification and Regression with Random Forest https://www.rdocumentation.org/packages/randomForest/versions/4.7-1.1/topics/randomForest