Predicting BWF Tournament Outcomes Using Random Forest

Huiming Zhou

December 2024

Contents

1 Introduction

A badminton match consists of up to 3 games. Usually, people only concern themselves with the winner of the match, but the score difference can actually demonstrate the skill levels between athletes more effectively. Simply knowing who won or lost doesn't directly reflect the skill gap between players. However, a significant score difference can better highlight the winner's excellence. Just like before a match, you can never accurately predict who will win. But based on our statistical data from the games, we can predict who will ultimately win and by what margin.

2 Data Exploration

2.1 Data Import

I first saw a dataset on Kaggle regarding the BWF (Badminton World Federation) World Tour, which contains data for all BWF World Tour (2018-04/2021) tournaments. Each record is a specific match played at a tournament. The data is split up per discipline (men's singles, women's singles, men's doubles, women's doubles, and mixed doubles). All data is scraped from the official BWF website.

Men's singles is my favorite discipline, which is chosen for my initial dataset. The dataset is 3761 rows * 39 features, containing information about tournament level, name, location, and date, as well as details about each match, including the detailed game points history and player names.

The drawback of the dataset is that players' information is insufficient for analysis. To fill this gap, I scraped the time-corresponding world rankings from the official BWF website, which contains their current ranking and the points they had at the specific week. For simplicity, I chose the statistics of the top 300 athletes of the last available week of 2018, 2019, 2020, and 2021.

2.2 Data Processing

After importing the data, the first step was to create the target column, which is computed as the difference between team one total points and team two total points. Then, I removed columns that are closely related to my target, such as the exact game score of each set, the exact points history of each set, and the total points for both teams. Additionally, irrelevant columns such as the nationalities of both players and the location and name of each tournament were also dropped.

Since the data is pulled from the BWF official site, there are no missing values except for the scoring information for set 3, as some matches end at set 2. I directly filled these NA values with 0.

Handling Categorical Columns

- **retired:** This column indicates if a team retired during a match. I removed rows where retired is TRUE to ensure these uncommon cases are not counted.
- **tournament_type:** This indicates the difficulty level of each tournament. As most top athletes are required to play in high-level tournaments, I selected matches from level 500 or higher tournaments. These tournaments have relatively stable participants, which is better for model training. The column is encoded with the level number directly.
- **round:** I used the number of survivors as encoding for each round. For instance: c('Quarter final' = 4, 'Semi final' = 2, 'Final' = 1).
- **date:** I extracted the year from the date column as I needed to combine the match results table with the rankings table. Since the name format is slightly different in the two tables, I performed transformations to match these two tables. The date, retired, and discipline columns were then dropped after the joining process.
- **players:** After joining the match results table with the rankings table using the year and player's name, I used the rankings and points to substitute the names of players and created new columns as the difference in rankings and points between player 1 and player 2.

The dataset is now fully processed.

2.3 Summary Statistics

Variable	Mean	Median	SD	Min	Max
tournament_type	729.0587219	750	226.1620	500	1250
round	10.6804836	8	5.5743	1	16
winner	1.4844560	1	0.4999	1	2
nb_sets	2.3773748	2	0.4849	2	3
team_one_most_consecutive_points	5.7227979	5	2.0533	1	16
team_two_most_consecutive_points	5.6899827	5	2.0579	2	17
team_one_game_points	2.2098446	2	1.9479	0	11
team_two_game_points	2.0941278	2	1.8922	0	11
team_one_most_consecutive_points_game_1	4.4240069	4	1.7420	1	16
team_two_most_consecutive_points_game_1	4.5164076	4	1.7905	1	13
team_one_most_consecutive_points_game_2	4.5397237	4	1.8397	1	13
team_two_most_consecutive_points_game_2	4.3955095	4	1.8286	1	17
team_one_most_consecutive_points_game_3	1.7193437	0	2.4966	0	13
team_two_most_consecutive_points_game_3	1.6873921	0	2.4546	0	13
target	0.1269430	1	12.5882	-32	28
year	2018.8721934	2019	0.9199	2018	2021

Variable	Mean	Median	SD	Min	Max
month	6.0414508	7	3.7844	1	12
day	16.7063903	17	7.4081	1	31
RANK_PLAYER1	19.3298791	16	17.5758	1	141
POINTS_PLAYER1	53235.1208981	50970	26138.31	1670	107590
RANK_PLAYER2	20.1096718	15	18.5365	1	147
POINTS_PLAYER2	51617.5302245	52360	24692.77	1290	96890
rank_difference	-0.7797927	-1	23.7461	-123	114
points_difference	1617.5906736	140	25958.63	-78830	92550

The whole dataset is now 1158 rows * 24 features

3 Modeling I

The dataset was split into an 80% training set and a 20% testing set, with a random seed set to 666 for reproducibility. The goal is to predict the match outcome (score difference) using Random Forest.

3.1 Decision Tree Model

I began by building a decision tree using the `rpart` package to establish a baseline for model performance. The decision tree produced a validation error of 32.09869. As shown in Figure **??**, the most important feature in the model is the winner variable, which clearly divides the outcome based on the match result.

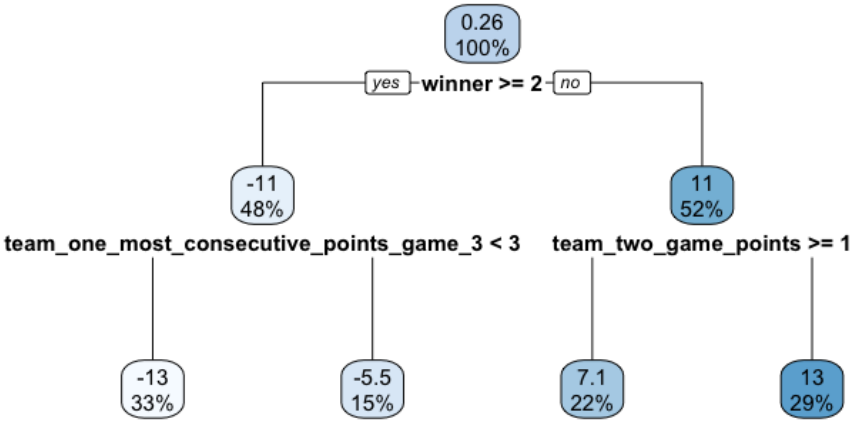

Figure 1: Decision Tree

3.2 Random Forest Model

A Random Forest model is a complex version of a decision tree, through each iteration, a new decision tree is planted based on chosen parameters and then the outcome of all decision trees would be taken account, which exhibits its ability to handle a large number of variables and its robustness to overfitting. The two most important hyperparameters in Random Forest are the number of trees (`ntree`) and the number of variables (`mtry`) considered at each split.

3.2.1 Optimizing `ntree`

To find the optimal number of trees (`ntree`), I first iterated through values from 10 to 300 with `mtry` fixed at 1. The results of this tuning process are shown in Figure 2. As the number of trees increased beyond 200, both the out-of-bag (OOB) error and the validation error stabilized, indicating diminishing returns from adding more trees. For stability, `ntree` was set to 300 for the final model.

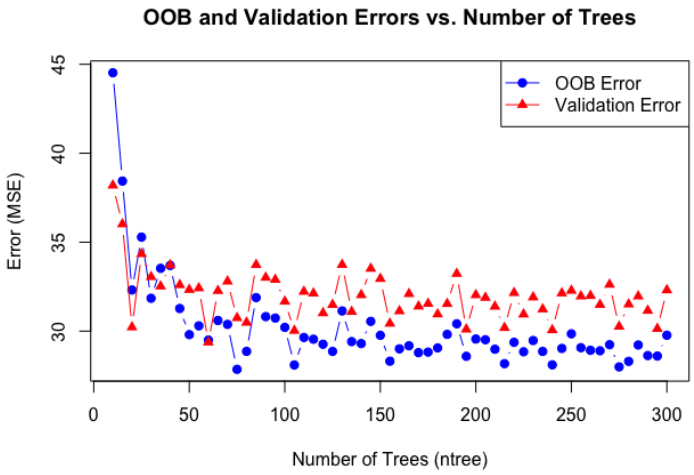

Figure 2: Effect of `ntree` on MSE

3.2.2 Optimizing `mtry`

Once `ntree` was set to 300, I proceeded to optimize the `mtry` parameter by iterating through values from 1 to 24. The best performance was achieved when `mtry` was set to 9, as shown in Figure 9, where the lowest validation error occurred at this point, with an OOB error of 18.668 and a validation error of 18.666.

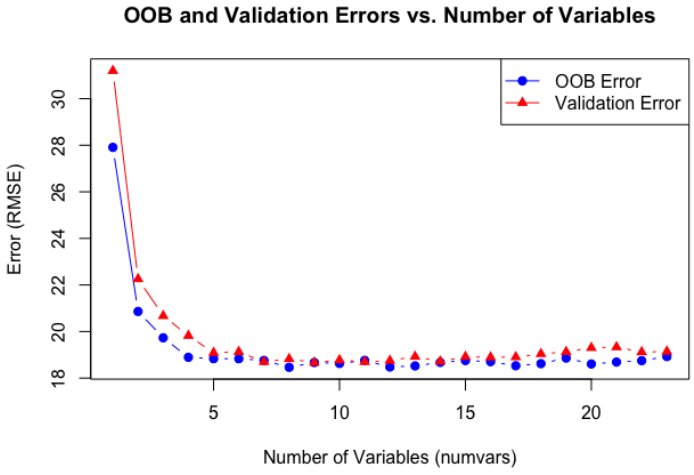

Figure 3: Effect of `mtry` on MSE

3.2.3 Variable Importance

After tuning the hyperparameters, the final Random Forest model was trained with `ntree = 300` and `mtry = 9`. And checking the variable importance in Figure 4, the winner is still the most influential variable, followed by team one most consecutive points game 3, which also appears in the single decision tree.

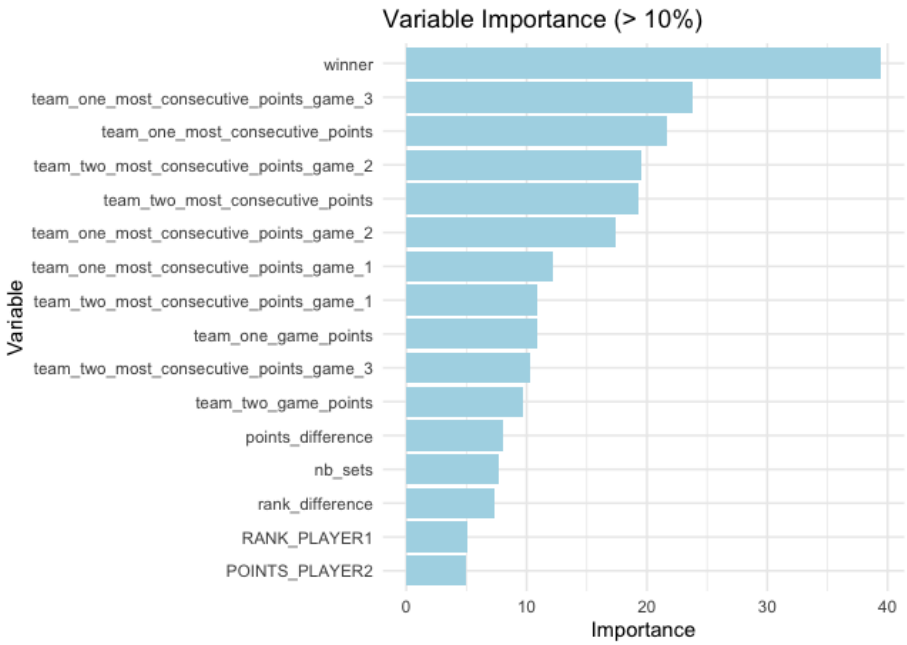

Figure 4: Variable Importance

3.3 Linear Model

I also want to check the performance of linear model as there's linearship between target and input, for example team two most consecutive points game 2.The mse i get is 17.75502, which is slightly lower than random forest, and i believe its because of the linearity exists.

Variable	Estimate	Std. Error	t value	Pr(¿—t—)
team_one_most_consecutive_points_game_1	1.139	0.1338	8.508	$< 2 \times 10^{-16}$
team_two_most_consecutive_points_game_1	−1.216	0.1336	−9.096	$< 2 \times 10^{-16}$
team_one_most_consecutive_points_game_2	1.203	0.1339	8.984	$< 2 \times 10^{-16}$
team_two_most_consecutive_points_game_2	−1.401	0.1327	−10.562	$< 2 \times 10^{-16}$
team_one_most_consecutive_points_game_3	0.5337	0.1580	3.379	0.00076
winner	-1.460×10^{1}	5.495	−26.580	$< 2 \times 10^{-16}$

Table 2: Summary of regression coefficients, standard errors, t values

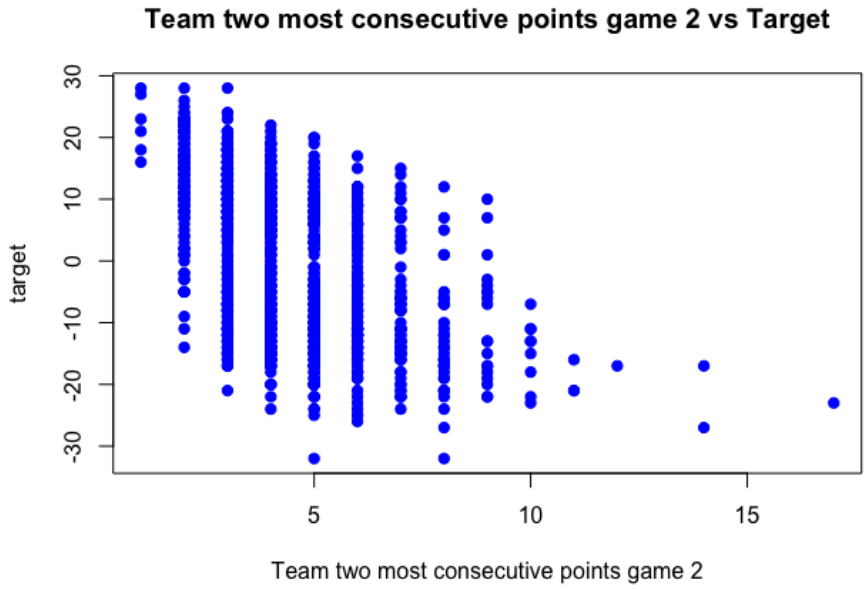

Figure 5: linearity

4 Model Comparison

Based on the model performance examined by validation error(mse), the linear model is the best at predicting the difference of score when the winner is known. The random forest is working definitely better than the decision tree which also relies a lot on the winner.

5 Dataset Transformation

Linear models perform well only when the relationship between variables is linear. In contrast, random forests excel with non-linear datasets due to their robustness, making them effective even with smaller feature sets. To better suit the random forest model, I removed the "winner" variable. This change not only makes the dataset more non-linear but also helps uncover the true difference between the two players.

After removing the "winner" column, the dataset now has 1,158 rows and 23 features. I believe the random forest model will perform better than the linear model, with only a slight increase in validation error.

6 Modeling II

6.1 Linear Model

This time, I start by fitting a linear model to test my hypothesis. The summary of the regression results is as follows: From the statistics, it is evident that

Variable	**Estimate**	**Std. Error**	**t value**	**Pr(¿—t—)**
team_one_game_points	1.123	0.1282	8.757	$< 2 \times 10^{-16}$
team_two_game_points	−1.181	0.1334	−8.856	$< 2 \times 10^{-16}$
team_one_most_consecutive_points_game_1	1.466	0.1778	8.247	5.70×10^{-16}
team_two_most_consecutive_points_game_1	−1.434	0.1779	−8.061	2.39×10^{-15}
team_one_most_consecutive_points_game_2	1.635	0.1773	9.222	$< 2 \times 10^{-16}$
team_two_most_consecutive_points_game_2	−1.684	0.1764	−9.543	$< 2 \times 10^{-16}$
team_one_most_consecutive_points_game_3	1.755	0.2016	8.704	$< 2 \times 10^{-16}$
team_two_most_consecutive_points_game_3	−1.250	0.2110	−5.924	4.47×10^{-9}

Table 3: Summary of regression coefficients, standard errors, t values

game points have become more important compared to the previous model, and several other variables are now significantly impactful. The final mean squared error (MSE) I obtained is 30.93965, nearly double that of the previous model, which further highlights the significance of the "winner" variable.

6.2 Decision Tree

Since the "winner" variable, which was the most important predictor in the earlier model, has been removed, it is expected that the decision tree's output will be significantly affected. The updated decision tree looks as follows:

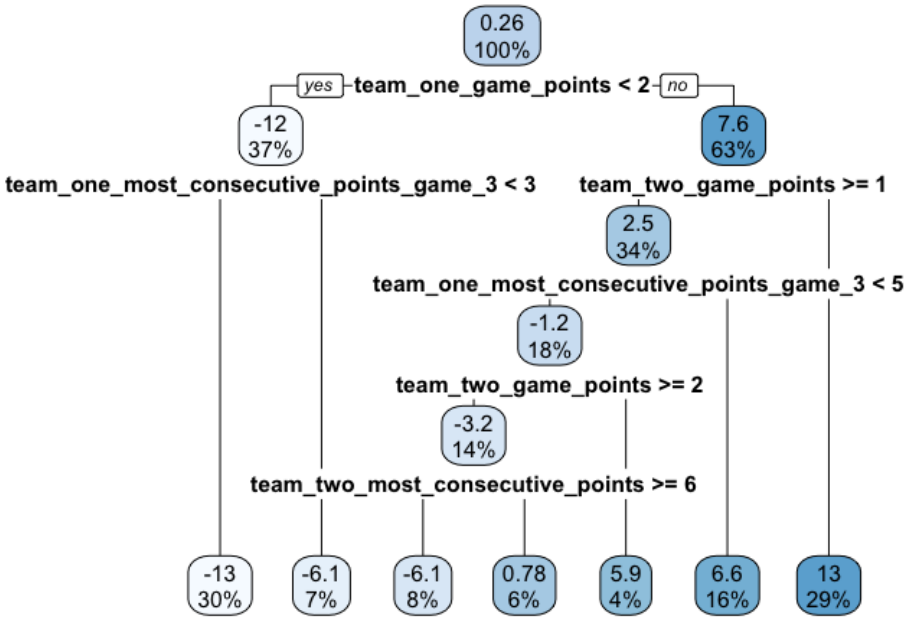

Figure 6: New decision tree

Now, "team one game points" is the variable used for the first split. The tree has almost twice as many nodes as the previous decision tree, which may lead to overfitting. The validation error has increased to 43.17176, representing a 30% increase compared to the original MSE.

6.3 Random Forest Model

Even though the most important variable was removed, the inherent robustness of random forests prevents any single variable from overwhelming the model and causing overfitting. Using the same process as before, I found that with 300 trees and 11 variables per split, the out-of-bag (OOB) error is 23.34295, and the validation error is 24.48316—showing only a slight difference of 6. "Team one game points" has now emerged as the most influential variable.

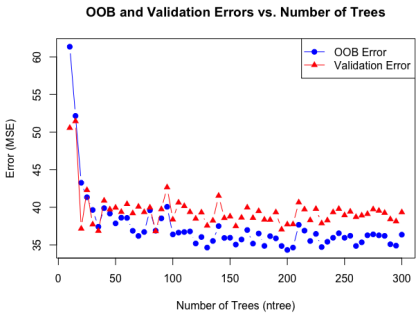

Figure 7: New ntree

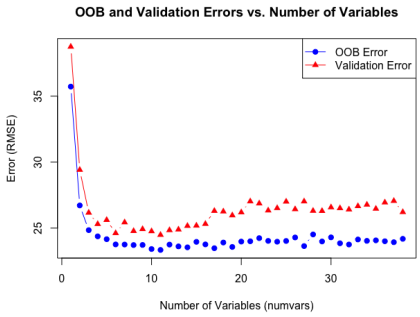

Figure 8: New numvars

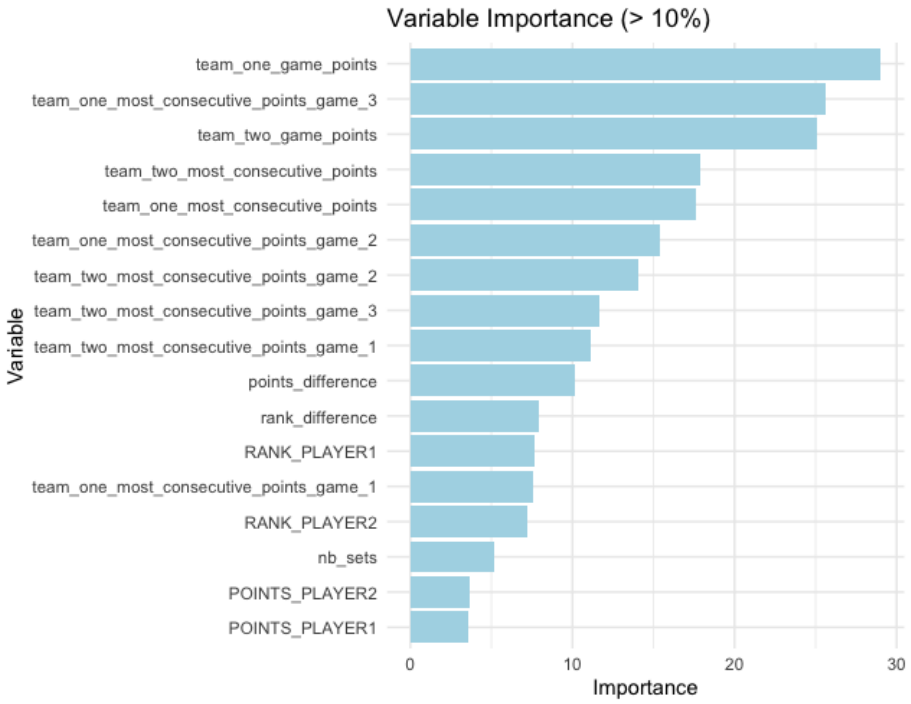

Figure 9: New variable importance

7 Conclusion

The Random Forest model successfully predicted the match outcomes with lower mse than linear model. The model's robustness and ability to handle complex

relationships between features make it a powerful tool for predicting badminton match outcomes based on historical data. With the known winner, linear model is doing better, but without the winner, random forest is the best predicting model. I will try test model on other disciplines in the future and try remove more columns to prove teh robustness of random forest.

8 References

References

[1] Sander, P. (2022). *Badminton BWF World Tour dataset*. Kaggle. https://www.kaggle.com/datasets/sanderp/badminton-bwf-world-tour/data

[2] Badminton World Federation. (2018, November 28). *BWF World Tour rankings, 2018, week 48*. Badminton World Federation. https://bwfworldtour.bwfbadminton.com/rankings/?id=9&cat_id=57&ryear=2018&week=48&page_size=25&page_no=1

[3] Badminton World Federation. (2019, November 28). *BWF World Tour rankings, 2019, week 48*. Badminton World Federation. https://bwfworldtour.bwfbadminton.com/rankings/?id=9&cat_id=57&ryear=2019&week=48&page_size=100&page_no=1

[4] Badminton World Federation. (2020, October 22). *BWF World Tour rankings, 2020, week 43*. Badminton World Federation. https://bwfworldtour.bwfbadminton.com/rankings/?id=9&cat_id=57&ryear=2020&week=43&page_size=100&page_no=1

[5] Badminton World Federation. (2021, November 30). *BWF World Tour rankings, 2021, week 48*. Badminton World Federation. https://bwfworldtour.bwfbadminton.com/rankings/?id=9&cat_id=57&ryear=2021&week=48&page_size=100&page_no=1

9 Appendix

```
install.packages("lubridate")
install.packages("reshape2")
install.packages("GGally")
library(reshape2)
library(lubridate)
library(dplyr)
library(readr)
library(rvest)
library(stringr)
library(randomForest)
library(ggplot2)
library(rpart)
library(rpart.plot)
library(GGally)



data = read_csv('/Users/vanris/Documents/UG-STA6801/Final Project/ms.csv')
head(data)
data$target = data$team_one_total_points - data$team_two_total_points

data_need = data[,!names(data) %in%
          c("game_1_score",'game_2_score','game_3_score',
          "game_1_scores",'game_2_scores','game_3_scores',
          'team_one_game_points_game_1', 'team_two_game_points_game_1',
          'team_one_game_points_game_2', 'team_two_game_points_game_2',
          'team_one_game_points_game_3', 'team_two_game_points_game_3',
          'team_one_total_points', 'team_two_total_points','city','tournament',
          'country','team_one_nationalities','team_two_nationalities',
          'team_one_player_two_nationalit','team_two_player_two_nationality'
          )]
head(data_need)
data_high_level = data_need %>%
  filter(tournament_type %in%
           c('HSBC BWF World Tour Finals', 'HSBC BWF World Tour Super 1000',
            'HSBC BWF World Tour Super 750', 'HSBC BWF World Tour Super 500'))
tournament_type_map = c('HSBC BWF World Tour Finals' = 1250,
                        'HSBC BWF World Tour Super 1000' = 1000,
                        'HSBC BWF World Tour Super 750' = 750,
                        'HSBC BWF World Tour Super 500' = 500)

data_high_level$tournament_type = tournament_type_map[data_high_level$tournament_type]
data_good = data_high_level %>%
  filter(retired != 'TRUE')


data_need = data_good

dates = dmy(data_need$date)
data_need$year = year(dates)
data_need$month = month(dates)
data_need$day = day(dates)

round_distribution = data_need %>%
  count(round)

data_need = data_need %>%
  filter(!round %in% c('Qualification quarter final',
                       'Qualification round of 16'))

round_map = c('Round of 32' = 16, 'Round of 16'= 8, 'Round of 64'= 32,
              'Quarter final'= 4, 'Semi final'= 2, 'Final'= 1,
              'Round 3' = 4, 'Round 2' = 4, 'Round 1' = 4)

data_need$round = round_map[data_need$round]


data_need = data_need[,!(names(data_need)) %in%
```

```
                        c('date','retired','discipline')]


# retired == TRUE is 20
#year 2018
url1 <- "https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2018&week=48&page_size=100&page_no=1"
url2 = 'https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2018&week=48&page_size=100&page_no=2'
url3 = 'https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2018&week=48&page_size=100&page_no=3'

webpage1 <- read_html(url1)
webpage2 = read_html(url2)
webpage3 = read_html(url3)

# 找到表格节点
table_node1 <- webpage1 %>% html_node(".rankings-table")
table_node2 = webpage2 %>% html_node(".rankings-table")
table_node3 = webpage3 %>% html_node(".rankings-table")

# 将表格节点转换为数据框
rankings_table1 <- table_node1 %>% html_table(header = TRUE)
rankings_table1 <- as.data.frame(rankings_table1)

rankings_table2 <- table_node2 %>% html_table(header = TRUE)
rankings_table2 <- as.data.frame(rankings_table2)

rankings_table3 <- table_node3 %>% html_table(header = TRUE)
rankings_table3 <- as.data.frame(rankings_table3)

rankings_table_2018 = rbind(rankings_table1,rankings_table2,rankings_table3)

# 清理和整理数据
rankings_table_2018 <- rankings_table_2018 %>%
  mutate(RANK = as.numeric(RANK),
         PLAYER = trimws(PLAYER),
         `CHANGE +/-` = as.numeric(`CHANGE +/-`),
         POINTS = as.numeric(gsub(",", "", POINTS))) %>%
  filter(!is.na(RANK) & PLAYER != "")

# 查看数据框
print(rankings_table_2018)




#year 2019
url1 <- "https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2019&week=48&page_size=100&page_no=1"
url2 = 'https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2019&week=48&page_size=100&page_no=2'
url3 = 'https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2019&week=48&page_size=100&page_no=3'

webpage1 <- read_html(url1)
webpage2 = read_html(url2)
webpage3 = read_html(url3)

# 找到表格节点
table_node1 <- webpage1 %>% html_node(".rankings-table")
table_node2 = webpage2 %>% html_node(".rankings-table")
table_node3 = webpage3 %>% html_node(".rankings-table")

# 将表格节点转换为数据框
rankings_table1 <- table_node1 %>% html_table(header = TRUE)
rankings_table1 <- as.data.frame(rankings_table1)
```

```
rankings_table2 <- table_node2 %>% html_table(header = TRUE)
rankings_table2 <- as.data.frame(rankings_table2)

rankings_table3 <- table_node3 %>% html_table(header = TRUE)
rankings_table3 <- as.data.frame(rankings_table3)

rankings_table_2019 = rbind(rankings_table1,rankings_table2,rankings_table3)

# 清理和整理数据
rankings_table_2019 <- rankings_table_2019 %>%
  mutate(RANK = as.numeric(RANK),
         PLAYER = trimws(PLAYER),
         `CHANGE +/-` = as.numeric(`CHANGE +/-`),
         POINTS = as.numeric(gsub(",", "", POINTS))) %>%
  filter(!is.na(RANK) & PLAYER != "")

# 查看数据框
print(rankings_table_2019)




#year 2020
url1 <- "https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2020&week=43&page_size=100&page_no=1"
url2 = 'https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2020&week=43&page_size=100&page_no=2'
url3 = 'https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2020&week=43&page_size=100&page_no=3'

webpage1 <- read_html(url1)
webpage2 = read_html(url2)
webpage3 = read_html(url3)

# 找到表格节点
table_node1 <- webpage1 %>% html_node(".rankings-table")
table_node2 = webpage2 %>% html_node(".rankings-table")
table_node3 = webpage3 %>% html_node(".rankings-table")

# 将表格节点转换为数据框
rankings_table1 <- table_node1 %>% html_table(header = TRUE)
rankings_table1 <- as.data.frame(rankings_table1)

rankings_table2 <- table_node2 %>% html_table(header = TRUE)
rankings_table2 <- as.data.frame(rankings_table2)

rankings_table3 <- table_node3 %>% html_table(header = TRUE)
rankings_table3 <- as.data.frame(rankings_table3)

rankings_table_2020 = rbind(rankings_table1,rankings_table2,rankings_table3)

# 清理和整理数据
rankings_table_2020 <- rankings_table_2020 %>%
  mutate(RANK = as.numeric(RANK),
         PLAYER = trimws(PLAYER),
         `CHANGE +/-` = as.numeric(`CHANGE +/-`),
         POINTS = as.numeric(gsub(",", "", POINTS))) %>%
  filter(!is.na(RANK) & PLAYER != "")

# 查看数据框
print(rankings_table_2020)



#year 2021
url1 <- "https://bwfworldtour.bwfbadminton.com/rankings/?
```

```
id=9&cat_id=57&ryear=2021&week=48&page_size=100&page_no=1"
url2 = 'https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2021&week=48&page_size=100&page_no=2'
url3 = 'https://bwfworldtour.bwfbadminton.com/rankings/?
id=9&cat_id=57&ryear=2021&week=48&page_size=100&page_no=3'

webpage1 <- read_html(url1)
webpage2 = read_html(url2)
webpage3 = read_html(url3)

# 找到表格节点
table_node1 <- webpage1 %>% html_node(".rankings-table")
table_node2 = webpage2 %>% html_node(".rankings-table")
table_node3 = webpage3 %>% html_node(".rankings-table")

# 将表格节点转换为数据框
rankings_table1 <- table_node1 %>% html_table(header = TRUE)
rankings_table1 <- as.data.frame(rankings_table1)

rankings_table2 <- table_node2 %>% html_table(header = TRUE)
rankings_table2 <- as.data.frame(rankings_table2)

rankings_table3 <- table_node3 %>% html_table(header = TRUE)
rankings_table3 <- as.data.frame(rankings_table3)

rankings_table_2021 = rbind(rankings_table1,rankings_table2,rankings_table3)

# 清理和整理数据
rankings_table_2021 <- rankings_table_2021 %>%
  mutate(RANK = as.numeric(RANK),
         PLAYER = trimws(PLAYER),
         `CHANGE +/-` = as.numeric(`CHANGE +/-`),
         POINTS = as.numeric(gsub(",", "", POINTS))) %>%
  filter(!is.na(RANK) & PLAYER != "")

# 查看数据框
print(rankings_table_2021)
new_row1 <- data.frame('RANK' = 38, 'COUNTRY / TERRITORY' = 'THA',
                       'PLAYER' = 'Suppanyu Avihingsanon',  'CHANGE +/-' = 6,
                       'POINTS' = 11160, 'BREAKDOWN' = NA)
new_row2 = data.frame('RANK' = 44, 'COUNTRY / TERRITORY' = 'THA',
                      'PLAYER' = 'Tanongsak Saensomboonsuk', 'CHANGE +/-' = 7,
                      'POINTS' = 8980, 'BREAKDOWN' = NA)
colnames(new_row1) <- colnames(rankings_table_2021)
colnames(new_row2) <- colnames(rankings_table_2021)
rankings_table_2021 = rbind(rankings_table_2021, new_row1, new_row2)

rankings_table_2018$year = rep(2018, nrow(rankings_table_2018))
rankings_table_2019$year = rep(2019, nrow(rankings_table_2019))
rankings_table_2020$year = rep(2020, nrow(rankings_table_2020))
rankings_table_2021$year = rep(2021, nrow(rankings_table_2021))

rankings_table = rbind(rankings_table_2018,rankings_table_2019,
                       rankings_table_2020,rankings_table_2021)




standardize_name <- function(name) {
  name %>%
    str_to_lower() %>%
    str_replace_all(" ", "") %>%
    str_replace_all(",", "") %>%
    str_split(pattern = "") %>%
```

```
    unlist() %>%
    sort() %>%
    paste(collapse = "")
}

rankings_table$PLAYER_STD <- apply(rankings_table, 1,
                                   function(row) standardize_name(row['PLAYER']))
game_table = data_need
game_table$PLAYER1_STD <- apply(data_need, 1,
                                function(row)
standardize_name(row['team_one_players']))
game_table$PLAYER2_STD <- apply(data_need, 1,
                                function(row)
standardize_name(row['team_two_players']))




merged_table_player1 <- left_join(game_table, rankings_table,
                                  by = c("PLAYER1_STD" = "PLAYER_STD", "year" =
"year")) %>%
  rename(RANK_PLAYER1 = RANK, POINTS_PLAYER1 = POINTS)

merged_table_player2 <- left_join(merged_table_player1, rankings_table,
                                  by = c("PLAYER2_STD" = "PLAYER_STD","year" = "year"))
%>%
  rename(RANK_PLAYER2 = RANK, POINTS_PLAYER2 = POINTS)

merged_table_player2 <- merged_table_player2 %>%
  mutate(team_one_most_consecutive_points_game_3 =
           ifelse(is.na(team_one_most_consecutive_points_game_3),
                  0, team_one_most_consecutive_points_game_3)) %>%
  mutate(team_two_most_consecutive_points_game_3 =
           ifelse(is.na(team_two_most_consecutive_points_game_3),
                  0, team_two_most_consecutive_points_game_3))
merged_table <- merged_table_player2 %>%
  select(-PLAYER1_STD, -PLAYER2_STD,
         -PLAYER.x,-PLAYER.y,
         -BREAKDOWN.x, -BREAKDOWN.y,
         -`COUNTRY / TERRITORY.x`, -`COUNTRY / TERRITORY.y`,
         -`CHANGE +/-.x`, -`CHANGE +/-.y`,
         -team_one_players,-team_two_players)

merged_table$rank_difference = merged_table$RANK_PLAYER1 -
  merged_table$RANK_PLAYER2
merged_table$points_difference = merged_table$POINTS_PLAYER1 -
  merged_table$POINTS_PLAYER2

na_rows <- merged_table[rowSums(is.na(merged_table)) > 0, ]

print(na_rows)




# Custom summary function for each column
summarize_column <- function(column) {
  if(is.numeric(column)) {
    c(Mean = mean(column, na.rm = TRUE),
      Median = median(column, na.rm = TRUE),
      SD = sd(column, na.rm = TRUE),
      Min = min(column, na.rm = TRUE),
      Max = max(column, na.rm = TRUE),
      NA_Count = sum(is.na(column)))
  } else if(is.factor(column) || is.character(column)) {
    c(Levels = length(unique(column)),
      NA_Count = sum(is.na(column)))
  } else {
```

```
    c()
  }
}

# Apply the custom summary function to each column
summary_table <- lapply(merged_table, summarize_column)

# Convert the list of summaries into a data frame
summary_table <- do.call(rbind, summary_table)

# Convert the row names to a column
summary_table <- data.frame(Variable = rownames(summary_table),
                            summary_table, row.names = NULL)

# Print the summary table
print(summary_table)
```

```
# MODELLING

set.seed(666)
n = nrow(merged_table)
train_indices = sample(seq_len(n), size = 0.8 * n)
train_data = merged_table[train_indices, ]
test_data = merged_table[-train_indices, ]
```

```
#Decision Tree
tree_model = rpart(target ~ ., data = train_data, method = "anova")
predictions = predict(tree_model, test_data)
validation_error <- mean((predictions - test_data$target)^2)
validation_error
rpart.plot(tree_model)
```

```
ntree_values <- seq(10, 300, by = 5)
oob_errors = numeric(length = length(ntree_values))
validation_errors = numeric(length = length(ntree_values))


for (i in seq_along(ntree_values)){
  ntree_value = ntree_values[i]
```

```
  rf_model <- randomForest(target ~ ., data = train_data, ntree = ntree_value,
                           mtry = 1, importance = TRUE)

  # Get the OOB error estimate (Mean Squared Error for regression)
  oob_error <- rf_model$mse[ntree_value]  # Access MSE for OOB

  # Make predictions on the test set
  predictions <- predict(rf_model, newdata = test_data)

  # Calculate MSE on the test set (validation error)
  validation_error <- mean((predictions - test_data$target)^2)

  # Store the errors
  oob_errors[i] <- oob_error
  validation_errors[i] <- validation_error

}


plot(ntree_values, oob_errors, type = "b", col = "blue", pch = 19,
     xlab = "Number of Trees (ntree)", ylab = "Error (MSE)",
     main = "OOB and Validation Errors vs. Number of Trees",
     ylim = c(min(c(oob_errors, validation_errors)),
              max(c(oob_errors, validation_errors))))
lines(ntree_values, validation_errors, type = "b", col = "red", pch = 17)

# Add a legend
legend("topright", legend = c("OOB Error", "Validation Error"),
       col = c("blue", "red"), pch = c(19, 17), lty = 1)



# i will take iteration 100 as the article does and now determine the numvars(0)
numvars = seq(1, ncol(train_data)-1, by = 1)
oob_errors = numeric(length = length(numvars))
validation_errors = numeric(length = length(numvars))

for (i in numvars){
  mtry = numvars[i]

  rf_model <- randomForest(target ~ ., data = train_data, ntree = 300,
                           mtry = mtry, importance = TRUE)

  # Get the OOB error estimate (Mean Squared Error for regression)
  oob_error <- rf_model$mse[ntree_value]  # Access MSE for OOB

  # Make predictions on the test set
  predictions <- predict(rf_model, newdata = test_data)

  # Calculate MSE on the test set (validation error)
  validation_error <- mean((predictions - test_data$target)^2)

  # Store the errors
  oob_errors[i] <- oob_error
  validation_errors[i] <- validation_error

}



plot(numvars, oob_errors, type = "b", col = "blue", pch = 19,
     xlab = "Number of Variables (numvars)", ylab = "Error (MSE)",
     main = "OOB and Validation Errors vs. Number of Variables",
     ylim = c(min(c(oob_errors, validation_errors)),
              max(c(oob_errors, validation_errors))))

lines(numvars, validation_errors, type = "b", col = "red", pch = 17)

# Add a legend
```

```
legend("topright", legend = c("OOB Error", "Validation Error"),
       col = c("blue", "red"), pch = c(19, 17), lty = 1)



min_validation_error_index = which.min(validation_errors)
best_numvar = numvars[min_validation_error_index]
oob_errors[min_validation_error_index]
validation_errors[min_validation_error_index]


# The final model has numvars = 9 and iteration = 300
# oob = 18.66805 , validation rmse = 18.66649



final_rf_model = randomForest(target ~ ., data = train_data, ntree = 300,
                              mtry = 9, importance = TRUE)
importances = importance(final_rf_model,type = 1)
var_imp_df = data.frame(Variable = rownames(importances),
                        Importance = importances[, 1])

threshold = 0.1 * max(var_imp_df$Importance)
important_vars_df = var_imp_df[var_imp_df$Importance > threshold, ]

ggplot(important_vars_df, aes(x = Importance,
                              y = reorder(Variable, Importance))) +
  geom_bar(stat = 'identity', fill = 'lightblue') +
  labs(title = "Variable Importance (> 10%)", x = "Importance",
       y = "Variable") +
  theme_minimal()




# linear regression
lm1 = lm(target ~ ., data = train_data)
summary(lm1)

plot(merged_table$team_two_most_consecutive_points_game_2, merged_table$target,
     main="Team two most consecutive points game 2 vs Target",
     xlab="Team two most consecutive points game 2",
     ylab="target",
     pch=19, col="blue")


test_data_without_shares = test_data[, !names(test_data) %in% 'target']

predictions = predict(lm1, newdata = test_data_without_shares)

mse = mean((predictions - test_data$target)^2)
mse

# got rmse = 17.75502 which is way higher than random forest
```

```
#only remove winner

data_less_winner = merged_table %>%
  select(-winner)

# MODELLING

set.seed(666)
n = nrow(data_less_winner)
train_indices = sample(seq_len(n), size = 0.8 * n)
train_data = data_less_winner[train_indices, ]
test_data = data_less_winner[-train_indices, ]




# linear regression
lm1 = lm(target ~ ., data = train_data)
summary(lm1)
test_data_without_shares = test_data[, !names(test_data) %in% 'target']

predictions = predict(lm1, newdata = test_data_without_shares)

mse = mean((predictions - test_data$target)^2)
mse

#mse 30.93965



#Decision Tree 43.17161
tree_model = rpart(target ~ ., data = train_data, method = "anova")
predictions = predict(tree_model, test_data)
validation_error <- mean((predictions - test_data$target)^2)
validation_error
rpart.plot(tree_model)
```

```
ntree_values <- seq(10, 300, by = 5)
oob_errors = numeric(length = length(ntree_values))
validation_errors = numeric(length = length(ntree_values))


for (i in seq_along(ntree_values)){
  ntree_value = ntree_values[i]

  rf_model <- randomForest(target ~ ., data = train_data, ntree = ntree_value,
                           mtry = 1, importance = TRUE)

  # Get the OOB error estimate (Mean Squared Error for regression)
  oob_error <- rf_model$mse[ntree_value]  # Access MSE for OOB

  # Make predictions on the test set
  predictions <- predict(rf_model, newdata = test_data)

  # Calculate MSE on the test set (validation error)
  validation_error <- mean((predictions - test_data$target)^2)

  # Store the errors
  oob_errors[i] <- oob_error
  validation_errors[i] <- validation_error

}


plot(ntree_values, oob_errors, type = "b", col = "blue", pch = 19,
     xlab = "Number of Trees (ntree)", ylab = "Error (MSE)",
     main = "OOB and Validation Errors vs. Number of Trees",
     ylim = c(min(c(oob_errors, validation_errors)),
              max(c(oob_errors, validation_errors))))
lines(ntree_values, validation_errors, type = "b", col = "red", pch = 17)

# Add a legend
legend("topright", legend = c("OOB Error", "Validation Error"),
       col = c("blue", "red"), pch = c(19, 17), lty = 1)



# i will take iteration 300 as the article does and now determine the numvars(0)
numvars = seq(1, ncol(data)-1, by = 1)
oob_errors = numeric(length = length(numvars))
validation_errors = numeric(length = length(numvars))

for (i in numvars){
  mtry = numvars[i]

  rf_model <- randomForest(target ~ ., data = train_data, ntree = 300,
                           mtry = mtry, importance = TRUE)

  # Get the OOB error estimate (Mean Squared Error for regression)
  oob_error <- rf_model$mse[ntree_value]  # Access MSE for OOB

  # Make predictions on the test set
  predictions <- predict(rf_model, newdata = test_data)

  # Calculate MSE on the test set (validation error)
  validation_error <- mean((predictions - test_data$target)^2)

  # Store the errors
  oob_errors[i] <- oob_error
  validation_errors[i] <- validation_error

}
```

```
plot(numvars, oob_errors, type = "b", col = "blue", pch = 19,
     xlab = "Number of Variables (numvars)", ylab = "Error (RMSE)",
     main = "OOB and Validation Errors vs. Number of Variables",
     ylim = c(min(c(oob_errors, validation_errors)),
              max(c(oob_errors, validation_errors))))

lines(numvars, validation_errors, type = "b", col = "red", pch = 17)

# Add a legend
legend("topright", legend = c("OOB Error", "Validation Error"),
       col = c("blue", "red"), pch = c(19, 17), lty = 1)



min_validation_error_index = which.min(validation_errors)
best_numvar = numvars[min_validation_error_index]
oob_errors[min_validation_error_index]
validation_errors[min_validation_error_index]


# The final model has numvars = 11 and iteration = 300
# oob = 23.34295 , validation rmse = 24.48316



final_rf_model = randomForest(target ~ ., data = train_data, ntree = 300,
                              mtry = 11, importance = TRUE)
importances = importance(final_rf_model,type = 1)
var_imp_df = data.frame(Variable = rownames(importances),
                        Importance = importances[, 1])

threshold = 0.1 * max(var_imp_df$Importance)
important_vars_df = var_imp_df[var_imp_df$Importance > threshold, ]

ggplot(important_vars_df, aes(x = Importance,
                              y = reorder(Variable, Importance))) +
  geom_bar(stat = 'identity', fill = 'lightblue') +
  labs(title = "Variable Importance (> 10%)", x = "Importance",
       y = "Variable") +
  theme_minimal()









# data_less_info = merged_table %>%
```

```
#   select(-winner, -points_difference, -rank_difference)
#
# # MODELLING
#
# set.seed(666)
# n = nrow(data_less_info)
# train_indices = sample(seq_len(n), size = 0.8 * n)
# train_data = data_less_info[train_indices, ]
# test_data = data_less_info[-train_indices, ]
#
#
# ntree_values <- seq(10, 300, by = 5)
# oob_errors = numeric(length = length(ntree_values))
# validation_errors = numeric(length = length(ntree_values))
#
#
# for (i in seq_along(ntree_values)){
#   ntree_value = ntree_values[i]
#
#   rf_model <- randomForest(target ~ ., data = train_data, ntree = ntree_value,
#   mtry = 1, importance = TRUE)
#
#   # Get the OOB error estimate (Mean Squared Error for regression)
#   oob_error <- rf_model$mse[ntree_value]  # Access MSE for OOB
#
#   # Make predictions on the test set
#   predictions <- predict(rf_model, newdata = test_data)
#
#   # Calculate MSE on the test set (validation error)
#   validation_error <- mean((predictions - test_data$target)^2)
#
#   # Store the errors
#   oob_errors[i] <- oob_error
#   validation_errors[i] <- validation_error
#
# }
#
#
# plot(ntree_values, oob_errors, type = "b", col = "blue", pch = 19,
#      xlab = "Number of Trees (ntree)", ylab = "Error (MSE)",
#      main = "OOB and Validation Errors vs. Number of Trees",
#      ylim = c(min(c(oob_errors, validation_errors)),
# max(c(oob_errors, validation_errors))))
# lines(ntree_values, validation_errors, type = "b", col = "red", pch = 17)
#
# # Add a legend
# legend("topright", legend = c("OOB Error", "Validation Error"),
#        col = c("blue", "red"), pch = c(19, 17), lty = 1)
#
#
#
# i will take iteration 100 as the article does and now determine the numvars(0)
# numvars = seq(1, ncol(data)-1, by = 1)
# oob_errors = numeric(length = length(numvars))
# validation_errors = numeric(length = length(numvars))
#
# for (i in numvars){
#   mtry = numvars[i]
#
#   rf_model <- randomForest(target ~ ., data = train_data, ntree = 300,
#  mtry = mtry, importance = TRUE)
#
#   # Get the OOB error estimate (Mean Squared Error for regression)
#   oob_error <- rf_model$mse[ntree_value]  # Access MSE for OOB
#
#   # Make predictions on the test set
#   predictions <- predict(rf_model, newdata = test_data)
#
#   # Calculate MSE on the test set (validation error)
```

```
#   validation_error <- mean((predictions - test_data$target)^2)
#
#   # Store the errors
#   oob_errors[i] <- oob_error
#   validation_errors[i] <- validation_error
#
# }
#
#
#
# plot(numvars, oob_errors, type = "b", col = "blue", pch = 19,
#      xlab = "Number of Variables (numvars)", ylab = "Error (RMSE)",
#      main = "OOB and Validation Errors vs. Number of Variables",
#      ylim = c(min(c(oob_errors, validation_errors)),
#  max(c(oob_errors, validation_errors))))
#
# lines(numvars, validation_errors, type = "b", col = "red", pch = 17)
#
# # Add a legend
# legend("topright", legend = c("OOB Error", "Validation Error"),
#        col = c("blue", "red"), pch = c(19, 17), lty = 1)
#
#
#
# min_validation_error_index = which.min(validation_errors)
# best_numvar = numvars[min_validation_error_index]
# oob_errors[min_validation_error_index]
# validation_errors[min_validation_error_index]
#
#
# # The final model has numvars = 10 and iteration = 300
# # oob = 22.5482 , validation rmse = 23.7828
#
#
#
# final_rf_model = randomForest(target ~ ., data = train_data, ntree = 300,
#  mtry = 7, importance = TRUE)
# importances = importance(final_rf_model,type = 1)
# var_imp_df = data.frame(Variable = rownames(importances),
#  Importance = importances[, 1])
#
# threshold = 0.1 * max(var_imp_df$Importance)
# important_vars_df = var_imp_df[var_imp_df$Importance > threshold, ]
#
# ggplot(important_vars_df, aes(x = Importance,
#  y = reorder(Variable, Importance))) +
#   geom_bar(stat = 'identity', fill = 'lightblue') +
#   labs(title = "Variable Importance (> 10%)", x = "Importance",
# y = "Variable") +
#   theme_minimal()
#
# #Decision Tree 43.17161
# tree_model = rpart(target ~ ., data = train_data, method = "anova")
# predictions = predict(tree_model, test_data)
# validation_error <- mean((predictions - test_data$target)^2)
# validation_error
# rpart.plot(tree_model)
#
#
#
# # linear regression
# lm1 = lm(target ~ ., data = train_data)
#
# test_data_without_shares = test_data[, !names(test_data) %in% 'target']
#
# predictions = predict(lm1, newdata = test_data_without_shares)
#
# mse = mean((predictions - test_data$target)^2)
# mse
```

```
# got rmse = 30.89157 which is way higher than random forest
```


```
# got rmse = 30.89157 which is way higher than random forest
```