```
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
```