BIOS611 Project

Xavier Loffree

2024-11-12

```r
library(maps)
library(ggplot2)
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## x purrr::map()    masks maps::map()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```r
library(mclust)
```

```
## Package 'mclust' version 6.1.1
## Type 'citation("mclust")' for citing this R package in publications.
##
## Attaching package: 'mclust'
##
## The following object is masked from 'package:purrr':
##
##     map
##
## The following object is masked from 'package:maps':
##
##     map
```

```r
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

```r
library(nnet)
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
```

```
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

```r
library(data.table)
```

```
##
## Attaching package: 'data.table'
##
## The following objects are masked from 'package:lubridate':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
##
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
##
## The following object is masked from 'package:purrr':
##
##     transpose
```

```r
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```r
df <- read.csv("Fifa_world_cup_matches.csv", header=TRUE)
```

Clean data and add variables

```r
#Make sure corresponding var names for teams 1 and 2 differ only by the team number
#In other words, some of the column names have typos that we need to fix
colnames(df)[which(colnames(df)=="attempts.inside.the.penalty.area..team2")] <-
  "attempts.inside.the.penalty.area.team2"
colnames(df)[which(colnames(df)=="completed.line.breaksteam1")] <-
  "completed.line.breaks.team1"
colnames(df)[which(colnames(df)=="completed.defensive.line.breaksteam1")] <-
  "completed.defensive.line.breaks.team1"



#Add total goals column
df$total.goals <- df$number.of.goals.team1 + df$number.of.goals.team2

#Add total attempts column
df$total.attempts <- df$total.attempts.team1 + df$total.attempts.team2
```

```r
#Add total attempted line breaks column
df$total.attempted.defensive.line.breaks <-
  df$attempted.defensive.line.breaks.team1 +
  df$attempted.defensive.line.breaks.team2

#Add indicator variable for if game is an elimination game
df$elimination <- as.factor(c(rep(0,48), rep(1,16)))

#Convert percentages to numerical vars
pct_cols <- c("possession.team1", "possession.team2", "possession.in.contest")
df[,pct_cols] <- lapply(df[,pct_cols], function(x) as.numeric(gsub("%", "", x)))

#Add match outcome variable, 1=team1 win, 2=team2 win, 0=tie
df$outcome <- as.factor(ifelse(df$number.of.goals.team1>df$number.of.goals.team2,1,
                     ifelse(df$number.of.goals.team1<df$number.of.goals.team2,2,0)))


write.csv(
  df, "match_data_clean.csv",
          row.names = FALSE)
```

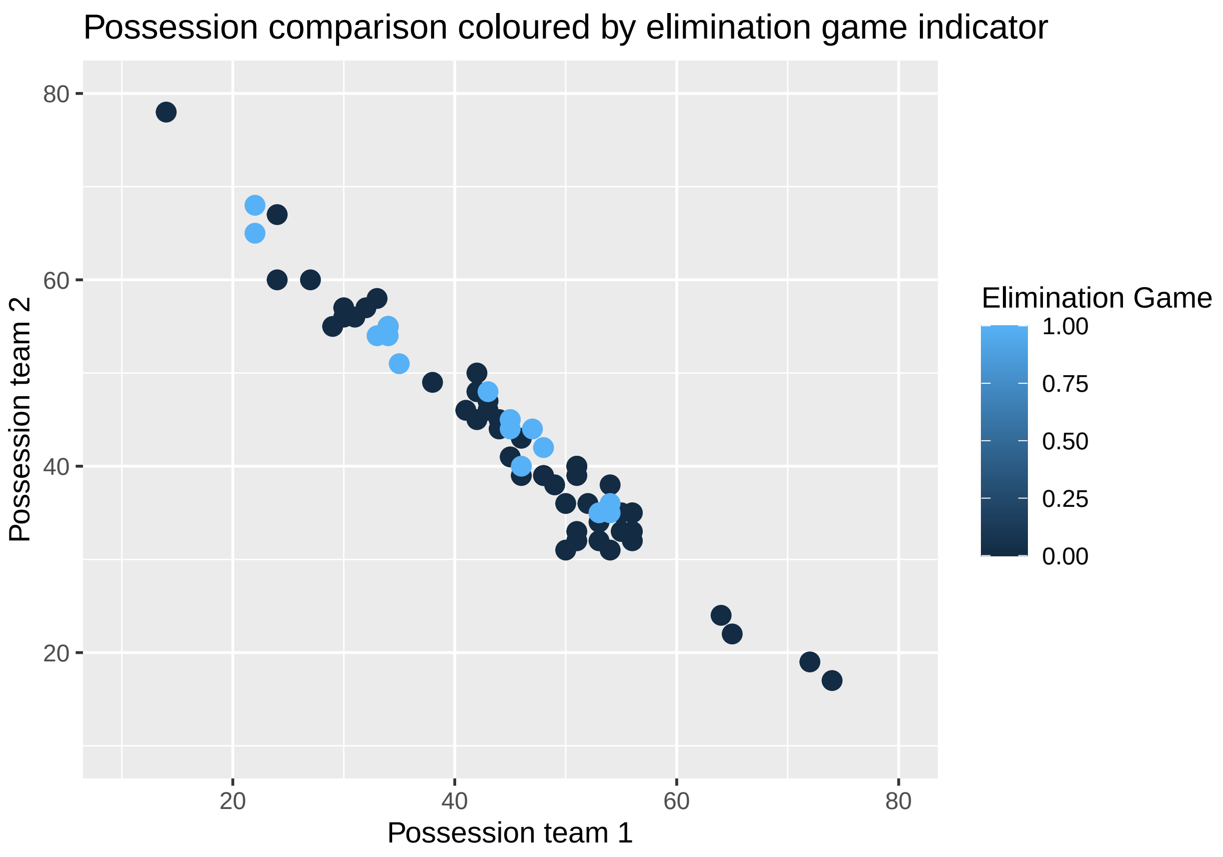
#Create df with average match data by country

```r
df <- read.csv(
  "match_data_clean.csv",
  header=TRUE)

#Create df to store country avg stats
avg_country_df <- data.frame()

#Create df for match data for each individual team (for use later)
match_data_team <- data.frame()


#Create new df with info summarized by country
countries <- unique(df$team1)
for (country in countries) {

  #Create subset df for games when country is team 1
  df_ss1 <- df %>% filter(team1 == country)
  df_ss1 <- df_ss1 %>% rename(fouls.committed = fouls.against.team1)
  df_ss1 <- df_ss1 %>% rename(fouls.drawn = fouls.against.team2)
  #Only keep variables related to country (remove team 2 vars)
  df_ss1 <- df_ss1 %>% select(!contains("2"))
  #Remove 1 from colnames
  colnames(df_ss1) <- gsub("1", "", colnames(df_ss1), "team_num")
  df_ss1$team_num <- 1

  #Vice versa
  df_ss2 <- df %>% filter(team2 == country)
  df_ss2 <- df_ss2 %>% rename(fouls.committed = fouls.against.team2)
  df_ss2 <- df_ss2 %>% rename(fouls.drawn = fouls.against.team1)
  #Only keep variables related to country (remove team 2 vars)
  df_ss2 <- df_ss2 %>% select(!contains("1"))
  #Remove 1 from colnames
  colnames(df_ss2) <- gsub("2", "", colnames(df_ss2), "team_num")
  df_ss2$team_num <- 2

  #Combine both ss dfs
  df_combined <- rbind(df_ss1, df_ss2)

  #Remove columns from which we can't take the average
  df_combined <- df_combined %>%
    select(-c("date", "hour", "category", "elimination"))

  #Create df for match data for each individual team (for use later)
  match_data_team <- rbind(match_data_team, df_combined)

  #Take means of columns
  df_summary <- df_combined[,-which(names(df_combined) %in% "outcome")] %>% group_by(team) %>%
    summarise_all(mean)

  #Add country row to df
  avg_country_df <- rbind(avg_country_df, df_summary)

}

#Add columns for longitude and latitude of capital cities
#Import this data
capitals <- read.csv("country-capital-lat-long-population.csv", header=TRUE)
#Convert to upper case and change column names to match data formats
capitals$Country <- toupper(capitals$Country)
colnames(capitals)[1] <- "team"

#Need to make some manual additions to capitals df
#Some country names are inconsistent
#Not all teams in the World Cup are actualy countries
capitals <- capitals %>% add_row(team="ENGLAND", Capital.City="London",
                                 Latitude=51.5085, Longitude=-0.1257,
                                 Population=9046485,Capital.Type="Capital")
capitals <- capitals %>% add_row(team="KOREA REPUBLIC", Capital.City="Seoul",
                                 Latitude=37.5683, Longitude=126.9778,
                                 Population=9963497,Capital.Type="Capital")
capitals <- capitals %>% add_row(team="UNITED STATES",
                                 Capital.City="Washington, D.C.",
                                 Latitude=38.8951, Longitude=-77.0364,
                                 Population=5206593,Capital.Type="Capital")
capitals <- capitals %>% add_row(team="WALES",
                                 Capital.City="Cardiff",
                                 Latitude=51.481583, Longitude=-3.179090,
                                 Population=372089,Capital.Type="Capital")
capitals <- capitals %>% add_row(team="IRAN",
                                 Capital.City="Tehran",
                                 Latitude=35.6944, Longitude=51.4215,
                                 Population=8895947,Capital.Type="Capital")

#Merge
```

```r
coords_df <- merge(capitals, avg_country_df, by="team")
write.csv(
  coords_df,
  "avg_team_data.csv",
  row.names = FALSE)

#Add column to match_team_data indicating result of the match
match_data_team$result <- as.factor(ifelse(match_data_team$outcome==0, 0,
                    ifelse(match_data_team$outcome==match_data_team$team_num, 1, 2)))
write.csv(
  match_data_team,
  "match_data_team.csv",
  row.names = FALSE)
```

Exploratory visualizations

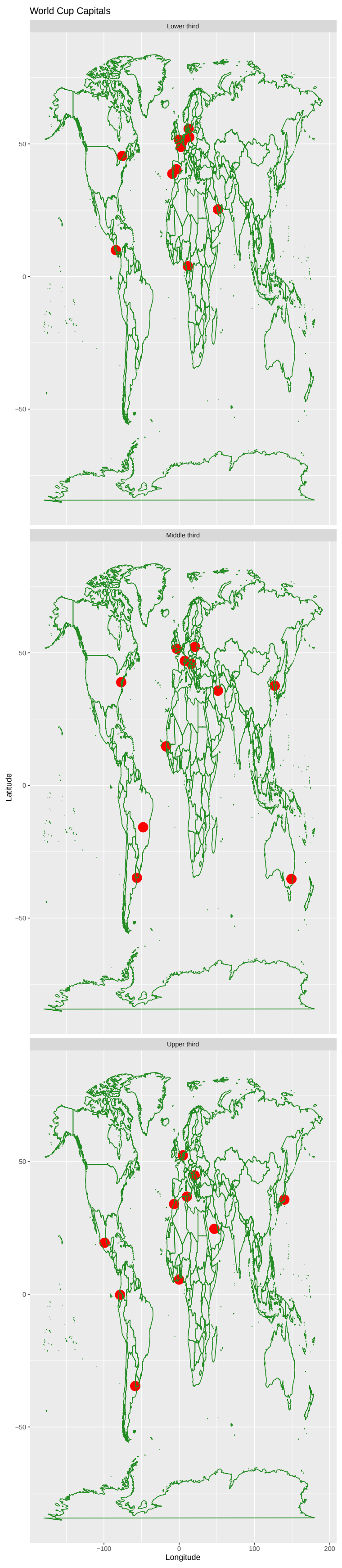
```r
#Is the possession split more evenly in elimination games than in group games?
ggplot(df, aes(x = possession.team1, y = possession.team2, color = elimination)) +
  geom_point(size = 3) +
  labs(x = "Possession team 1", y = "Possession team 2", color = "Elimination Game") +
  ggtitle("Possession comparison coloured by elimination game indicator") +
  scale_x_continuous(limits = c(10, 80)) +
  scale_y_continuous(limits = c(10, 80))
```



The distribution of the total number of goals is skewed to the right. There appears to be a positive correlation between total attempts and total attempted line breaks. Games that feature more total attempts and total attempted line breaks tend to have a greater number of goals scored. You can visualize this by selecting groups of points in the scatter plot and seeing how the barplot bars fill up. There does not appear to be any significant difference in possession split when comparing elimination vs group stage games.

Does the number of fouls comitted differ by geography?

```r
coords_df <- read.csv(
  "avg_team_data.csv",
  header=TRUE
)

#Does the number of fouls committed per game differ by geographic region?

#Add fouls category by splitting into three quantiles
coords_df$fouls_category <-
  cut(coords_df$fouls.committed,
      breaks = c(0,
                 quantile(coords_df$fouls.committed, 0.33),
                 quantile(coords_df$fouls.committed, 0.66),
                 quantile(coords_df$fouls.committed, 1)),
                               labels = c("Lower third",
                                          "Middle third",
                                          "Upper third"))




#Put capital city of each team on map
#Facet by number of fouls

map_outline <- map_data("world")

foul_map <- ggplot(coords_df, aes(x = Longitude, y = Latitude)) +
    geom_point(color = "red", size = 6) +
    labs(x = "Longitude", y = "Latitude", title = "World Cup Capitals")

# Add the map layer and facet by 'fouls_category'
foul_map <- foul_map +
    geom_path(data = map_outline, aes(x = long, y = lat, group = group), color = "forestgreen") +
    facet_wrap(~ fouls_category, ncol=1)


foul_map
```


World Cup Capitals
Lower third
Middle third
Upper third
Latitude
Longitude
50
0
−50
−100
0
100
200

```r
ggsave("foul_map.png")
```

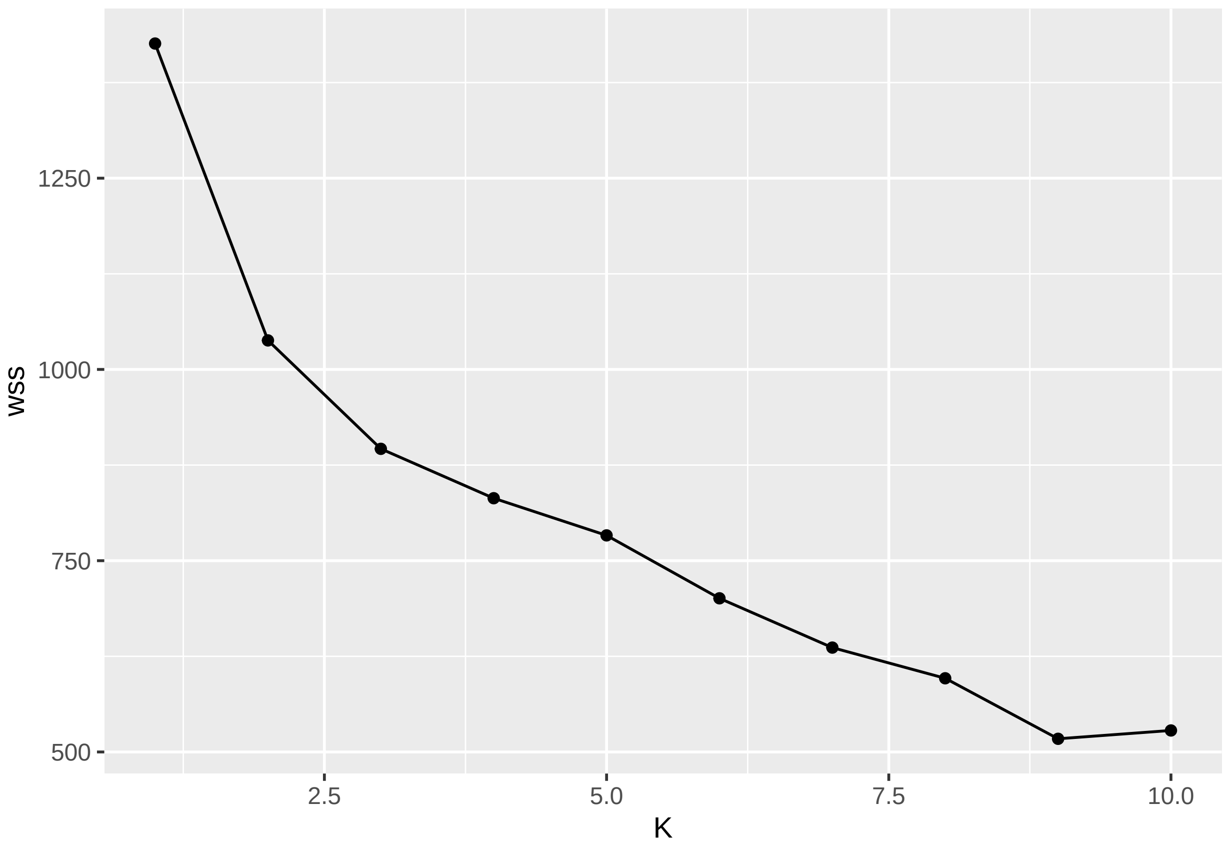
```
## Saving 6.5 x 30 in image
```

It does not look like there are any geographical patterns with regard to number of fouls comitted.

Does geography influence play style?

That is, can we cluster the data such that the clusters represent distinct geographical regions?

```r
coords_df <- read.csv(
  "avg_team_data.csv",
  header=TRUE
)

#Remove variables not to be included in kmeans
kmeans_df <- coords_df[,-c(1:6,ncol(coords_df))]
#Scale data
kmeans_df <- scale(kmeans_df)


#Find k that minimizes within sum of squares (wss)

#Store wss for each k value
wss <- list()
for (i in 1:10) {
  # Fit the model: km.out
  kmeans_wss <- kmeans(kmeans_df, centers = i)$tot.withinss
  # Save the within cluster sum of squares
  wss[[i]] <- kmeans_wss



}


#Scree plot
scree_df <-  data.frame(wss=unlist(wss), k=1:10)

kmeans_scree <- ggplot(scree_df, aes(x = k, y = wss)) +
    geom_point()+
    geom_line() +
    xlab('K')
kmeans_scree
```



Looks like the rate of decrease drops off after k=5. Try clustering with k=5. This seems promising given that it is close to the number of continents. We can creatively group countries together to make 5 groups

```
set.seed(42)
k5 <- kmeans(kmeans_df, centers = 5, nstart = 20)
k5$size
```

```
## [1] 6 8 1 9 8
```

```
#Add clusters to df
coords_df$cluster5 <- as.factor(k5$cluster)
```

There is a group of one! Let's see which country did not fit into any group. Any guesses?

```
#Show how the countries were categorized
cluster_summary <- coords_df %>%
  group_by(cluster5) %>%
  summarize(team = paste(team, collapse = ", "))
print(cluster_summary)
```

```
## # A tibble: 5 x 2
##   cluster5 team
##   <fct>    <chr>
## 1 1        ARGENTINA, BRAZIL, ENGLAND, FRANCE, GERMANY, PORTUGAL
## 2 2        BELGIUM, CANADA, CROATIA, DENMARK, KOREA REPUBLIC, MEXICO, TUNISIA, ~
## 3 3        SPAIN
## 4 4        CAMEROON, ECUADOR, MOROCCO, NETHERLANDS, QATAR, SENEGAL, SWITZERLAND~
## 5 5        AUSTRALIA, COSTA RICA, GHANA, IRAN, JAPAN, POLAND, SAUDI ARABIA, SER~
```

```
write.csv("cluster_summary.csv")
```

```
## "","x"
## "1","cluster_summary.csv"
```

Spain is the odd one out. Spain is known for having a distinct, possession-based play style. At first glance, there doesn't seem to be any obvious grouping by geography. Group 1 seems like it is the teams that did better in the tournament. Let's evaluate more carefully.

Let's split the countries into 5 groups based on geography. Europe and Africa can be their own groups. Forming the other 3 groups is more interesting. Let's try a group of countries from the Americas, not including Canada and the USA. Let's make another group for Asian countries. The last group can be Canada, USA, and Australia. The reasoning here is that Canada, USA, and Australia seem more culturally similar to each other and Mexico, central America, and South America seem more culturally similar to each other. Objectively, this is at least true in terms of language.

```r
continent_g1 <- c("latam", "can_us_aus", "eur", "latam", "afr", "can_us_aus",
                  "latam", "eur", "eur", "latam", "eur", "eur", "eur", "afr",
                  "asia", "asia", "asia", "latam", "afr", "eur", "eur", "eur",
                  "asia", "asia", "afr", "eur", "eur", "eur", "afr", "can_us_aus",
                  "latam", "eur")

coords_df$cont_g1 <- continent_g1

#Confusion matrix
table(coords_df$cluster5, coords_df$cont_g1)
```

```
##
##     afr asia can_us_aus eur latam
##   1   0    0          0   4     2
##   2   1    1          2   3     1
##   3   0    0          0   1     0
##   4   3    1          0   3     2
##   5   1    3          1   2     1
```

Does not look like there is any intelligent clustering by geography.

Let's check the adjusted Rand index to be sure of this.

```r
# adjusted Rand index
adjusted_rand <- adjustedRandIndex(coords_df$cluster5, coords_df$cont_g1)
print(adjusted_rand)
```

```
## [1] -0.03497156
```

The value close to zero indicates random assignment!

So how did the algorithm make the clustering decisions? Let's check if it clustered the teams based on how well they performed in the tournament. The five groups can be: 1. Eliminated in group stage (n=16) 2. Eliminated in round of 16 (n=8) 3. Eliminated in quarter finals (n=4) 4. Eliminated in semi-finals (n=2) 5. Finalists (n=2)

```r
coords_df$team
```

```
##  [1] "ARGENTINA"      "AUSTRALIA"      "BELGIUM"        "BRAZIL"
##  [5] "CAMEROON"       "CANADA"         "COSTA RICA"     "CROATIA"
##  [9] "DENMARK"        "ECUADOR"        "ENGLAND"        "FRANCE"
## [13] "GERMANY"        "GHANA"          "IRAN"           "JAPAN"
## [17] "KOREA REPUBLIC" "MEXICO"         "MOROCCO"        "NETHERLANDS"
## [21] "POLAND"         "PORTUGAL"       "QATAR"          "SAUDI ARABIA"
## [25] "SENEGAL"        "SERBIA"         "SPAIN"          "SWITZERLAND"
## [29] "TUNISIA"        "UNITED STATES"  "URUGUAY"        "WALES"
```

```
place_g2 <- c("final", "16", "group", "quarter", "group", "group", "group",
              "semi", "group", "group", "quarter", "final", "group", "group",
              "group", "16", "16", "group", "semi", "quarter", "16", "quarter",
              "group", "group", "16", "group", "16", "16", "group", "16",
              "group", "group")
coords_df$place_g2 <- place_g2

#Confusion matrix
table(coords_df$cluster5, coords_df$place_g2)
```

```
##
##      16 final group quarter semi
##   1  0     2     1       3    0
##   2  2     0     5       0    1
##   3  1     0     0       0    0
##   4  2     0     5       1    1
##   5  3     0     5       0    0
```

Still does not look promising…

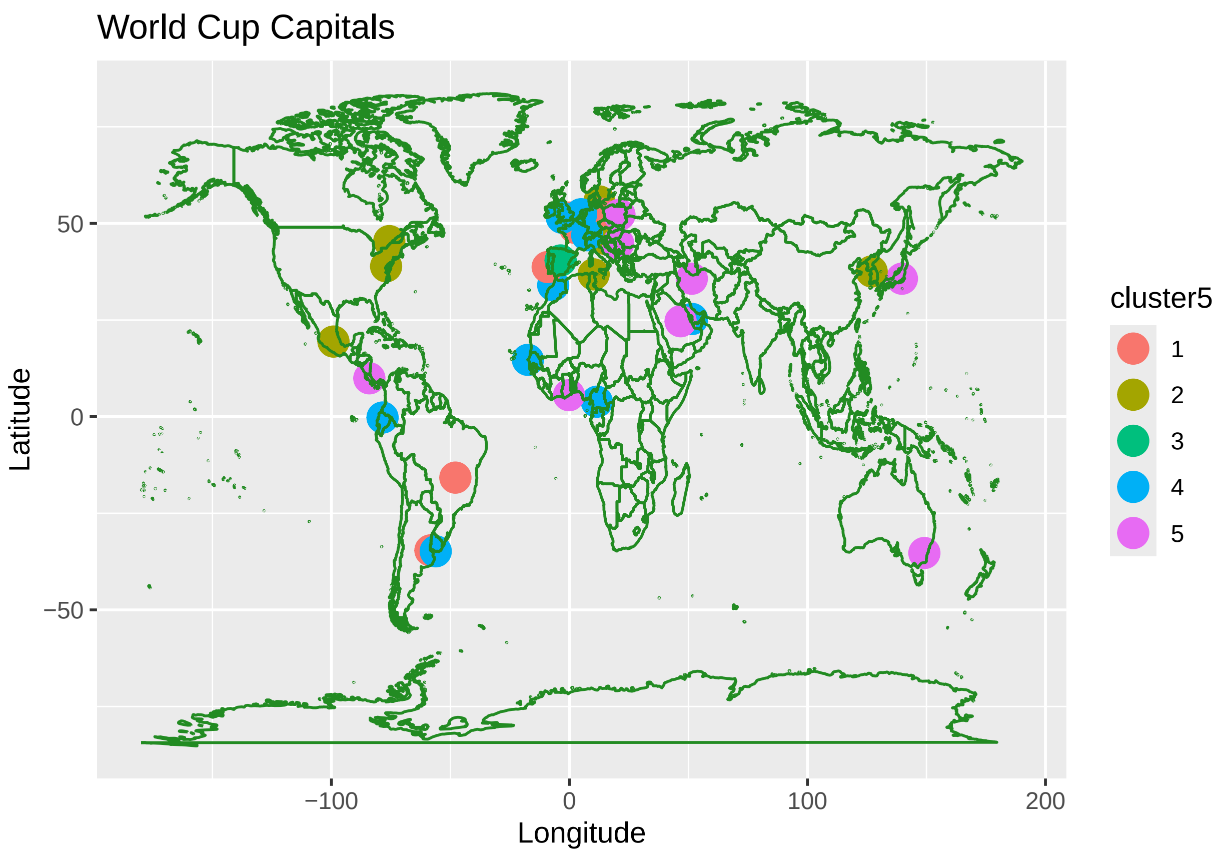
```
# adjusted Rand index
adjusted_rand <- adjustedRandIndex(coords_df$cluster5, coords_df$place_g2)
print(adjusted_rand)
```

```
## [1] 0.05464436
```

The value is still very close to zero.

Let's visualize the clusters on a map.

```
map_outline <- map_data("world")

foul_map <- ggplot(coords_df, aes(x = Longitude, y = Latitude, colour = cluster5)) +
    geom_point(size = 5) +
    labs(x = "Longitude", y = "Latitude", title = "World Cup Capitals")
# Add the map layer and facet
foul_map <- foul_map +
    geom_path(data = map_outline, aes(x = long, y = lat, group = group), color = "forestgreen")

foul_map
```



The clustering on visualized on the map does not appear to reveal any patterns.

Can we successfully predict the outcome of a match given the match statistics?

```r
#Make tie the reference variable
match_data_team$result <- relevel(match_data_team$result, ref = "0")
#Create df to use in model
#Remove variables that would make such predictions too easy (vars related to
#number of goals)
#Also remove variables time, data, team
match_data_team_model <- match_data_team[,-which(names(match_data_team) %in%
                                              c("team",
                                        "number.of.goals.team", "date",
                                        "hour", "category", "team_num", "outcome",
                                        "conceded.team",
                                        "goal.inside.the.penalty.area.team",
                                        "goal.outside.the.penalty.area.team",
                                        "own.goals.team", "assists.team",
                                        "penalties.scored.team"))]


set.seed(42)
train_indices <- sample(1:nrow(match_data_team_model),
                        0.8 * nrow(match_data_team_model))
```

```r
train_data <- match_data_team_model[train_indices, ]
test_data <- match_data_team_model[-train_indices, ]

#Multinomial logistic regression
mlog_reg_outcome <- multinom(result ~., data = train_data)
```

```
## # weights:  123 (80 variable)
## initial  value 112.058453
## iter  10 value 88.078984
## iter  20 value 66.176740
## iter  30 value 54.052117
## iter  40 value 44.254872
## iter  50 value 36.781641
## iter  60 value 29.577750
## iter  70 value 17.944363
## iter  80 value 0.543290
## iter  90 value 0.000760
## final  value 0.000047
## converged
```

```r
logregsum <- summary(mlog_reg_outcome)

predictions <- predict(mlog_reg_outcome, newdata = test_data)

conf_matrix <- table(test_data$result, predictions)
conf_matrix
```

```
##    predictions
##     0 1 2
##   0 2 2 2
##   1 1 3 3
##   2 1 4 8
```

```r
#Accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
accuracy
```

```
## [1] 0.5
```

```r
#Pvalues
p_values <- summary(mlog_reg_outcome)$coefficients / summary(mlog_reg_outcome)$standard.errors
p_values <- (1 - pnorm(abs(p_values), 0, 1)) * 2
p_values
```

```
##   (Intercept) possession.team possession.in.contest total.attempts.team
## 1           0       0.9117139             0.7402454           0.9565844
## 2           0       0.9305087             0.7083593           0.8772336
##   on.target.attempts.team off.target.attempts.team
## 1               0.5306534                0.8290913
## 2               0.7684007                0.6375899
##   attempts.inside.the.penalty.area.team attempts.outside.the.penalty.area..team
## 1                             0.9166035                               0.9628749
## 2                             0.8816902                               0.9701695
##   left.channel.team left.inside.channel.team central.channel.team
## 1         0.8872270                0.9542121            0.7215989
## 2         0.8800422                0.8795477            0.9256529
##   right.inside.channel.team right.channel.team total.offers.to.receive.team
## 1                 0.8121954          0.9850896                    0.9615330
## 2                 0.9462036          0.9989084                    0.9722376
##   inbehind.offers.to.receive.team inbetween.offers.to.receive.team
## 1                       0.9770130                        0.9614093
## 2                       0.9785763                        0.9727139
##   infront.offers.to.receive.team
## 1                      0.9686744
## 2                      0.9983309
##   receptions.between.midfield.and.defensive.lines.team
## 1                                            0.9362522
## 2                                            0.8345420
##   attempted.line.breaks.team completed.line.breaks.team
## 1                  0.9735520                  0.9861574
## 2                  0.9745519                  0.9738413
##   attempted.defensive.line.breaks.team completed.defensive.line.breaks.team
## 1                            0.9189088                           0.9261415
## 2                            0.8911630                           0.9064440
##   yellow.cards.team red.cards.team fouls.committed fouls.drawn offsides.team
## 1         0.6814301              0       0.8940253   0.9733527     0.9954811
## 2         0.7476326              0       0.9190823   0.9647434     0.9132599
##   passes.team passes.completed.team crosses.team crosses.completed.team
## 1   0.9550451             0.9687258     0.852658              0.8207928
## 2   0.9803412             0.9867796     0.998552              0.8297805
##   switches.of.play.completed.team corners.team free.kicks.team
## 1                       0.9537292    0.9937072       0.9774835
## 2                       0.9559119    0.7604904       0.9826513
##   goal.preventions.team forced.turnovers.team defensive.pressures.applied.team
## 1             0.9021742             0.9756222                        0.9525727
## 2             0.8445103             0.9469199                        0.9327718
##   total.goals total.attempts total.attempted.defensive.line.breaks
## 1   0.8545369      0.7360873                             0.9867276
## 2   0.6659776      0.8416486                             0.9673868
```

The accuracy of our model was 0.5. This means that we can predict the outcome of the match with an accuracy greater than random guessing (which would have an accuracy of 0.33).

However, the very high p-values indicate we may have a problem with multicollinearity. Let's fix this by using elastic net as a variable selection method.

Elastic net

```r
# Prepare data for elastic net
#Train test split
x <- model.matrix(result ~ ., data = train_data)[,-1]
y <- train_data$result
x_test <- model.matrix(result ~ ., data = test_data)[, -1]
y_test <- test_data$result

# Fit elastic net regression
en_model <- cv.glmnet(as.matrix(x), y, family = "multinomial", alpha = 0.5,
                      type.measure = "class")

#Optimal lambda
```

```
en_model$lambda.min
```

```
## [1] 0.01042711
```

```
#Selected coefficients for each class (tie, win, loss)
coef(en_model, s = en_model$lambda.min)
```

```
## $`0`
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                                                 1
## (Intercept)                                          -5.201259998
## possession.team                                       .
## possession.in.contest                                -0.271558517
## total.attempts.team                                   0.016950624
## on.target.attempts.team                              -0.034544765
## off.target.attempts.team                             -0.029452585
## attempts.inside.the.penalty.area.team                 0.081873222
## attempts.outside.the.penalty.area..team               .
## left.channel.team                                    -0.052755758
## left.inside.channel.team                             -0.089319737
## central.channel.team                                  0.123658346
## right.inside.channel.team                            -0.021914475
## right.channel.team                                    .
## total.offers.to.receive.team                          .
## inbehind.offers.to.receive.team                      -0.018163062
## inbetween.offers.to.receive.team                      .
## infront.offers.to.receive.team                        .
## receptions.between.midfield.and.defensive.lines.team -0.001548933
## attempted.line.breaks.team                            0.007349975
## completed.line.breaks.team                           -0.002498571
## attempted.defensive.line.breaks.team                  0.107237472
## completed.defensive.line.breaks.team                  .
## yellow.cards.team                                    -0.099386115
## red.cards.team                                       -0.229614964
## fouls.committed                                       0.091174935
## fouls.drawn                                           0.077652816
## offsides.team                                        -0.131378059
## passes.team                                           0.003721572
## passes.completed.team                                 0.000055232
## crosses.team                                          .
## crosses.completed.team                               -0.093601229
## switches.of.play.completed.team                       .
## corners.team                                          .
## free.kicks.team                                       .
## goal.preventions.team                                 .
## forced.turnovers.team                                 0.063448691
## defensive.pressures.applied.team                      .
## total.goals                                          -0.280894463
## total.attempts                                        .
## total.attempted.defensive.line.breaks                 .
##
## $`1`
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                                                  1
## (Intercept)                                        4.5820472472
## possession.team                                   -0.0742935039
## possession.in.contest                              0.0450872394
## total.attempts.team                                .
## on.target.attempts.team                            0.5277888276
## off.target.attempts.team                           .
## attempts.inside.the.penalty.area.team              .
## attempts.outside.the.penalty.area..team           -0.1039322043
## left.channel.team                                  0.0249476506
## left.inside.channel.team                           .
## central.channel.team                              -0.0350247782
## right.inside.channel.team                          .
## right.channel.team                                -0.0069044491
## total.offers.to.receive.team                       .
## inbehind.offers.to.receive.team                    0.0095927859
## inbetween.offers.to.receive.team                  -0.0048195429
## infront.offers.to.receive.team                     0.0009216114
## receptions.between.midfield.and.defensive.lines.team  .
## attempted.line.breaks.team                        -0.0150222911
## completed.line.breaks.team                         0.0293018516
## attempted.defensive.line.breaks.team               .
## completed.defensive.line.breaks.team               0.0592266414
## yellow.cards.team                                  .
## red.cards.team                                     0.9124829614
## fouls.committed                                   -0.0409275837
## fouls.drawn                                        .
## offsides.team                                      .
## passes.team                                       -0.0005736958
## passes.completed.team                              .
## crosses.team                                      -0.1356517612
## crosses.completed.team                             .
## switches.of.play.completed.team                    .
## corners.team                                       0.0133965585
## free.kicks.team                                    .
## goal.preventions.team                             -0.0080140246
## forced.turnovers.team                              .
## defensive.pressures.applied.team                   .
## total.goals                                        .
## total.attempts                                    -0.0331309041
## total.attempted.defensive.line.breaks             -0.0352115397
##
## $‘2‘
## 40 x 1 sparse Matrix of class "dgCMatrix"
##                                                             1
## (Intercept)                                        0.619212751
## possession.team                                    0.014468748
## possession.in.contest                              .
## total.attempts.team                                .
## on.target.attempts.team                           -0.113189037
## off.target.attempts.team                           0.100885891
## attempts.inside.the.penalty.area.team             -0.113128523
## attempts.outside.the.penalty.area..team            0.065284338
## left.channel.team                                  .
## left.inside.channel.team                           0.169754916
## central.channel.team                                  .
## right.inside.channel.team                             .
## right.channel.team                                    .
## total.offers.to.receive.team                          .
## inbehind.offers.to.receive.team                       .
## inbetween.offers.to.receive.team                      0.003450775
## infront.offers.to.receive.team                       -0.001606506
## receptions.between.midfield.and.defensive.lines.team  .
## attempted.line.breaks.team                            .
## completed.line.breaks.team                            .
## attempted.defensive.line.breaks.team                 -0.017639144
## completed.defensive.line.breaks.team                 -0.001771919
## yellow.cards.team                                     0.053530294
## red.cards.team                                        .
## fouls.committed                                       .
## fouls.drawn                                          -0.063248754
## offsides.team                                         0.282118695
## passes.team                                           .
## passes.completed.team                                 .
## crosses.team                                          0.029791623
## crosses.completed.team                                0.092773913
## switches.of.play.completed.team                       .
## corners.team                                         -0.232368807
## free.kicks.team                                      -0.006910415
## goal.preventions.team                                 0.097213578
## forced.turnovers.team                                -0.050036358
## defensive.pressures.applied.team                      .
## total.goals                                           0.047233677
## total.attempts                                        .
## total.attempted.defensive.line.breaks                 0.015639976
```

```r
# Predicted class probabilities
predicted_probs <- predict(en_model, newx = x_test, s = "lambda.min", type = "response")
predicted_probs
```

```
## , , 1
## 
##              0          1           2
## 1  0.033934488 0.16484911 0.801216403
## 7  0.015252861 0.24193912 0.742808024
## 12 0.442119848 0.43717593 0.120704225
## 14 0.555256344 0.29297057 0.151773085
## 23 0.977762417 0.01707736 0.005160223
## 31 0.080038641 0.73960967 0.180351685
## 38 0.707418642 0.23972474 0.052856620
## 44 0.014142508 0.14783870 0.838018795
## 53 0.741362138 0.10169627 0.156941595
## 55 0.868240375 0.05716647 0.074593156
## 57 0.029111675 0.05393278 0.916955544
## 72 0.010924588 0.22228589 0.766789525
## 76 0.396770941 0.57484788 0.028381175
## 78 0.519227973 0.09072939 0.390042633
## 82 0.179654540 0.62399025 0.196355214
## 83 0.055791884 0.37446211 0.569746001
## 91 0.033744443 0.05489412 0.911361437
## 93  0.323169741 0.37429684 0.302533418
## 102 0.040959922 0.06078481 0.898255267
## 105 0.271318559 0.42237486 0.306306584
## 108 0.027038196 0.86151831 0.111443491
## 109 0.531280747 0.26480560 0.203913649
## 113 0.153319152 0.66385910 0.182821749
## 117 0.028960012 0.14710304 0.823936951
## 120 0.001366045 0.60301331 0.395620642
## 126 0.052622399 0.20689736 0.740480244
```

```
# Predicted class labels
predicted_classes <- predict(en_model, newx = x_test, s = "lambda.min", type = "class")
predicted_classes
```

```
##         1
##   [1,] "2"
##   [2,] "2"
##   [3,] "0"
##   [4,] "0"
##   [5,] "0"
##   [6,] "1"
##   [7,] "0"
##   [8,] "2"
##   [9,] "0"
##  [10,] "0"
##  [11,] "2"
##  [12,] "2"
##  [13,] "1"
##  [14,] "0"
##  [15,] "1"
##  [16,] "2"
##  [17,] "2"
##  [18,] "1"
##  [19,] "2"
##  [20,] "1"
##  [21,] "1"
##  [22,] "0"
##  [23,] "1"
##  [24,] "2"
##  [25,] "1"
##  [26,] "2"
```

```
#Evaluate performance

# Confusion matrix
conf_matrix <- table(Predicted = predicted_classes, Actual = y_test)
conf_matrix
```

```
##          Actual
## Predicted 0 1 2
##         0 4 2 2
##         1 0 5 3
##         2 2 0 8
```

```
# Calculate accuracy
accuracy <- sum(diag(conf_matrix)) / sum(conf_matrix)
```

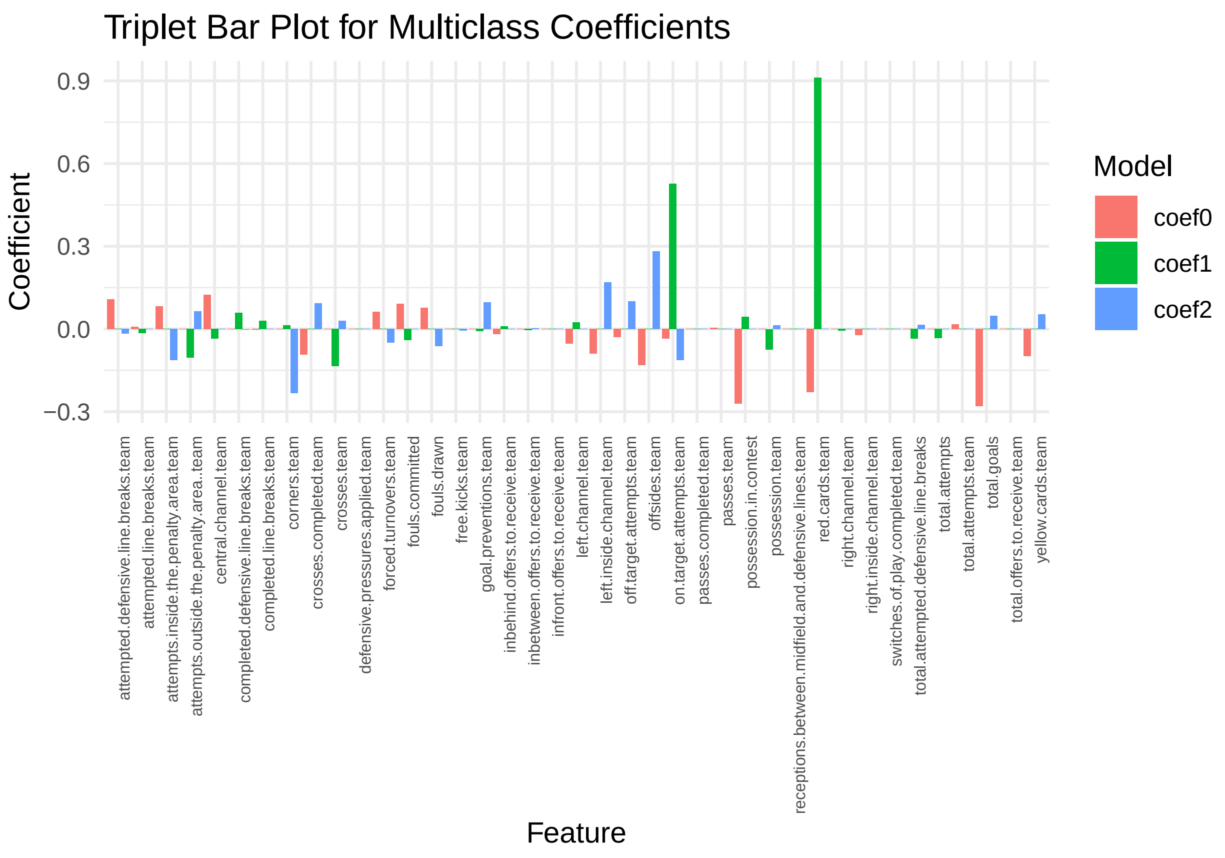
```
accuracy
```

```
## [1] 0.6538462
```

The accuracy of 65.38% is significantly greater than using multinomial logistic regression without variable selection. It appears that variable selection helped improve our model.

Which variables are most important for these predictions?

```
#Coefficients
coefs <- coef(en_model, s = "lambda.min")
#Coefficient df
en_coef_df <- data.frame(Variable=c(colnames(x)),
           coef0=coefs[1][[1]][2:40],
           coef1=coefs[2][[1]][2:40],
           coef2=coefs[3][[1]][2:40])
#Convert data to long format for plotting
long_df <- melt(data.table(en_coef_df), id.vars="Variable",
     variable.name = "Model", value.name = "Value")



#Plot the triplet bar graph
#Compare coefficient values across all three classes (tie, loss, win)
ggplot(long_df, aes(x = Variable, y = Value, fill = Model)) +
  geom_bar(stat = "identity", position = "dodge") +  # position dodge places bars next to each other
  theme_minimal() +
  labs(title = "Triplet Bar Plot for Multiclass Coefficients",
       x = "Feature",
       y = "Coefficient") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size=6))
```

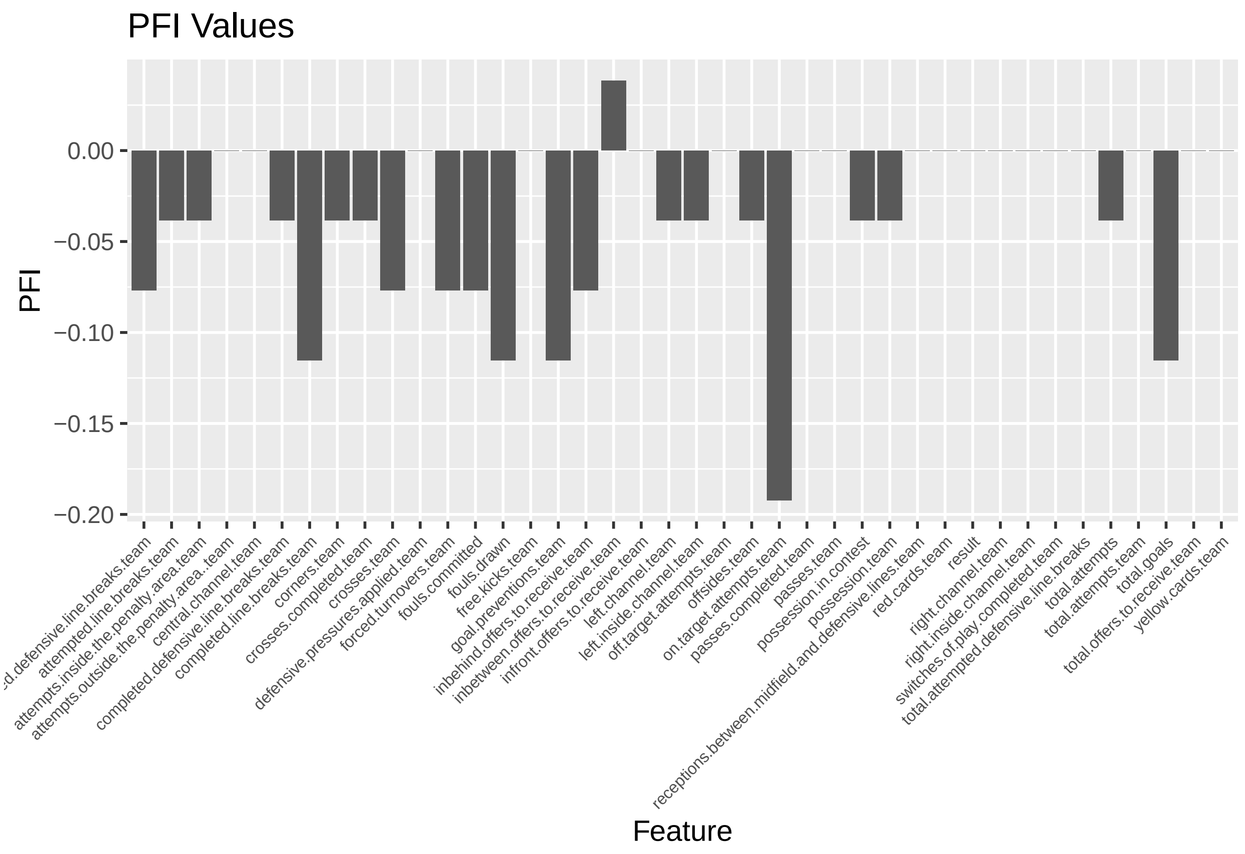


It looks like red cards and on target attempts were most important for determining the outcome of the games, based on the plot. Surprisingly, the log odds of winning versus tieing increase significantly for each red card a team receives! Although I do not have data to support this, I would suspect that when a winning team received a red card, it often happened near the end of the game. This way, the team would not have to play a man down for a long time. Another surprising result is that possession was not particularly influential in determining match outcome.

#Permutation feature importance

The coefficient values provide info on how important the variables were for prediction. Permutation feature importance (PFI) is another metric that can be used to assess the influence of a predictor on the model's prediction. PFI is the difference in model performance when one variable in the testing set is randomly permuted. Let's calculate PFI values for the elastic net model.

```r
set.seed(42)
#Vector to store PFI values
pfi_vect <- NULL

for (i in 1:ncol(test_data)) {
  #Clone test data
  test_data_per <- test_data
  #Randomly permute column i
  test_data_per[,i] <- sample(test_data_per[,i])
  #Create test data with randomly permuted column i
  x_test_per <- model.matrix(result ~ ., data = test_data_per)[, -1]

  # Predicted class labels
  predicted_classes_per <- predict(en_model, newx = x_test_per, s = "lambda.min",
                                   type = "class")
  # Confusion matrix
  conf_matrix_per <- table(Predicted = predicted_classes_per, Actual = y_test)

  # Calculate accuracy
  accuracy_per <- sum(diag(conf_matrix_per)) / sum(conf_matrix_per)
  acc_diff <- accuracy_per - accuracy
  #Append to pfi vector
  pfi_vect <- c(pfi_vect, acc_diff)

}


#Create pfi df
pfi_df <- data.frame(Var=colnames(test_data),
                     PFI=pfi_vect)

# Plot the coefficients for each class
ggplot(pfi_df, aes(x=Var, y=PFI)) +
  geom_bar(stat = "identity") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "PFI Values",
       x = "Feature",
       y = "PFI") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1, size=6))
```



A negative PFI values indicates that the model accuracy decreased when permuting that particular variable, and vice versa for positive PFI values. A PFI value of 0 indicates that there was no change in model accuracy.

The PFI results differed from the coefficients values. Notably, the PFI for red cards was 0. Forced turnovers was the second lowest PFI value, even though it did not have a particularly large coefficient value. Total attempts on goal was the most influential variable in terms of PFI and in terms of coefficient values

Can we predict the total number of goals scored in a game?

Let's try with xgboost

```r
set.seed(42)
#Remove vars that would make such a prediction too easy
#Also remove, data, time, team name, etc.
df_reg <- df[,-which(names(df) %in% c("team1", "team2",
                                        "number.of.goals.team1",
                                      "number.of.goals.team2","date",
                                      "number.of.goals.team1",
                                        "hour", "category",
                                        "conceded.team1","conceded.team2",
                                        "goal.inside.the.penalty.area.team1",
                                      "goal.inside.the.penalty.area.team2",
                                        "goal.outside.the.penalty.area.team1",
                                      "goal.outside.the.penalty.area.team2",
                                        "own.goals.team1", "own.goals.team2",
                                      "assists.team1", "assists.team2",
                                        "penalties.scored.team1",
                                      "penalties.scored.team2", "outcome",
                                      "elimination"))]



#Train test split
train_indices_reg <- sample(1:nrow(df_reg),
                        0.8 * nrow(df_reg))
train_data_reg <- data.matrix(df_reg[train_indices_reg, ])
test_data_reg <- df_reg[-train_indices_reg, ]

#Define predictor and response variables in training set
train_x = data.matrix(train_data_reg
                     [,-which(colnames(train_data_reg)=="total.goals")])
train_y = train_x[,which(colnames(train_data_reg)=="total.goals")]

#Define predictor and response variables in testing set
test_x = data.matrix(test_data_reg
                     [,-which(colnames(test_data_reg)=="total.goals")])
test_y = test_x[,which(colnames(test_data_reg)=="total.goals")]

#Define final training and testing sets
xgb_train = xgb.DMatrix(data = train_x, label = train_y)
xgb_test = xgb.DMatrix(data = test_x, label = test_y)


#Define watchlist
watchlist = list(train=xgb_train, test=xgb_test)

#Fit XGBoost model and display training and testing data at each round
xgb_model = xgb.train(data = xgb_train, max.depth = 3,
                   watchlist=watchlist, nrounds = 70)
```

```
## [1]  train-rmse:16.351229    test-rmse:16.484171
## [2]  train-rmse:11.811347    test-rmse:12.128655
## [3]  train-rmse:8.581002 test-rmse:8.840908
## [4]  train-rmse:6.285137 test-rmse:6.465342
## [5]  train-rmse:4.673629 test-rmse:4.822035
## [6]  train-rmse:3.511651 test-rmse:3.435052
## [7]  train-rmse:2.675627 test-rmse:2.498768
## [8]  train-rmse:2.069472 test-rmse:1.834512
## [9]  train-rmse:1.636494 test-rmse:1.290090
## [10] train-rmse:1.314316 test-rmse:0.962761
## [11] train-rmse:1.081714 test-rmse:0.778236
## [12] train-rmse:0.899081 test-rmse:0.742409
## [13] train-rmse:0.757631 test-rmse:0.729652
## [14] train-rmse:0.641829 test-rmse:0.705980
## [15] train-rmse:0.550699 test-rmse:0.720267
## [16] train-rmse:0.475856 test-rmse:0.719716
## [17] train-rmse:0.413701 test-rmse:0.714229
## [18] train-rmse:0.362479 test-rmse:0.729419
## [19] train-rmse:0.320795 test-rmse:0.723379
## [20] train-rmse:0.280176 test-rmse:0.736308
## [21] train-rmse:0.248736 test-rmse:0.736752
## [22] train-rmse:0.224459 test-rmse:0.740297
## [23] train-rmse:0.199794 test-rmse:0.751140
## [24] train-rmse:0.178695 test-rmse:0.773267
## [25] train-rmse:0.159814 test-rmse:0.775373
## [26] train-rmse:0.140390 test-rmse:0.793187
## [27] train-rmse:0.124773 test-rmse:0.806212
## [28] train-rmse:0.112396 test-rmse:0.812152
## [29] train-rmse:0.100422 test-rmse:0.823292
## [30] train-rmse:0.085693 test-rmse:0.827910
## [31] train-rmse:0.074425 test-rmse:0.831044
## [32] train-rmse:0.069612 test-rmse:0.829186
## [33] train-rmse:0.061815 test-rmse:0.830828
## [34] train-rmse:0.054888 test-rmse:0.830283
## [35] train-rmse:0.044927 test-rmse:0.829337
## [36] train-rmse:0.037754 test-rmse:0.827525
## [37] train-rmse:0.033995 test-rmse:0.826099
## [38] train-rmse:0.027800 test-rmse:0.826097
## [39] train-rmse:0.025128 test-rmse:0.823633
## [40] train-rmse:0.021566 test-rmse:0.825214
## [41] train-rmse:0.019278 test-rmse:0.825974
## [42] train-rmse:0.016346 test-rmse:0.826906
## [43] train-rmse:0.014241 test-rmse:0.827798
## [44] train-rmse:0.012579 test-rmse:0.827602
## [45] train-rmse:0.010724 test-rmse:0.827319
## [46] train-rmse:0.009265 test-rmse:0.825807
## [47] train-rmse:0.007897 test-rmse:0.825636
## [48] train-rmse:0.007088 test-rmse:0.825305
## [49] train-rmse:0.006257 test-rmse:0.825281
## [50] train-rmse:0.005463 test-rmse:0.825470
## [51] train-rmse:0.004776 test-rmse:0.825384
## [52] train-rmse:0.004159 test-rmse:0.825666
## [53] train-rmse:0.003795 test-rmse:0.825810
## [54] train-rmse:0.003397 test-rmse:0.825761
## [55] train-rmse:0.003107 test-rmse:0.825530
## [56] train-rmse:0.002710 test-rmse:0.825682
## [57] train-rmse:0.002434 test-rmse:0.825919
## [58] train-rmse:0.002196 test-rmse:0.825908
## [59] train-rmse:0.001994 test-rmse:0.825953
## [60] train-rmse:0.001813 test-rmse:0.826013
## [61] train-rmse:0.001633 test-rmse:0.825864
## [62] train-rmse:0.001436 test-rmse:0.825625
## [63] train-rmse:0.001226 test-rmse:0.825853
## [64] train-rmse:0.001136 test-rmse:0.825832
## [65] train-rmse:0.000982 test-rmse:0.825749
## [66] train-rmse:0.000850 test-rmse:0.825914
## [67] train-rmse:0.000769 test-rmse:0.825961
## [68] train-rmse:0.000662 test-rmse:0.825972
## [69] train-rmse:0.000588 test-rmse:0.825928
## [70] train-rmse:0.000534 test-rmse:0.825980
```

```
#Avoid overfitting by stopping when rmse starts to increase
xgb_final_model = xgb.train(data = xgb_train, max.depth = 3,
                  watchlist=watchlist, nrounds = 17)
```

```
## [1]  train-rmse:16.351229    test-rmse:16.484171
## [2]  train-rmse:11.811347    test-rmse:12.128655
## [3]  train-rmse:8.581002 test-rmse:8.840908
## [4]  train-rmse:6.285137 test-rmse:6.465342
## [5]  train-rmse:4.673629 test-rmse:4.822035
## [6]  train-rmse:3.511651 test-rmse:3.435052
## [7]  train-rmse:2.675627 test-rmse:2.498768
## [8]  train-rmse:2.069472 test-rmse:1.834512
## [9]  train-rmse:1.636494 test-rmse:1.290090
## [10] train-rmse:1.314316 test-rmse:0.962761
## [11] train-rmse:1.081714 test-rmse:0.778236
## [12] train-rmse:0.899081 test-rmse:0.742409
## [13] train-rmse:0.757631 test-rmse:0.729652
## [14] train-rmse:0.641829 test-rmse:0.705980
## [15] train-rmse:0.550699 test-rmse:0.720267
## [16] train-rmse:0.475856 test-rmse:0.719716
## [17] train-rmse:0.413701 test-rmse:0.714229
```

```
#Predictions
y_pred <- predict(xgb_model, test_x)
#MSE
mean((test_y - y_pred)^2)
```

```
## [1] 0.6822436
```

```
#MAE
caret::MAE(test_y, y_pred)
```

```
## [1] 0.6715505
```

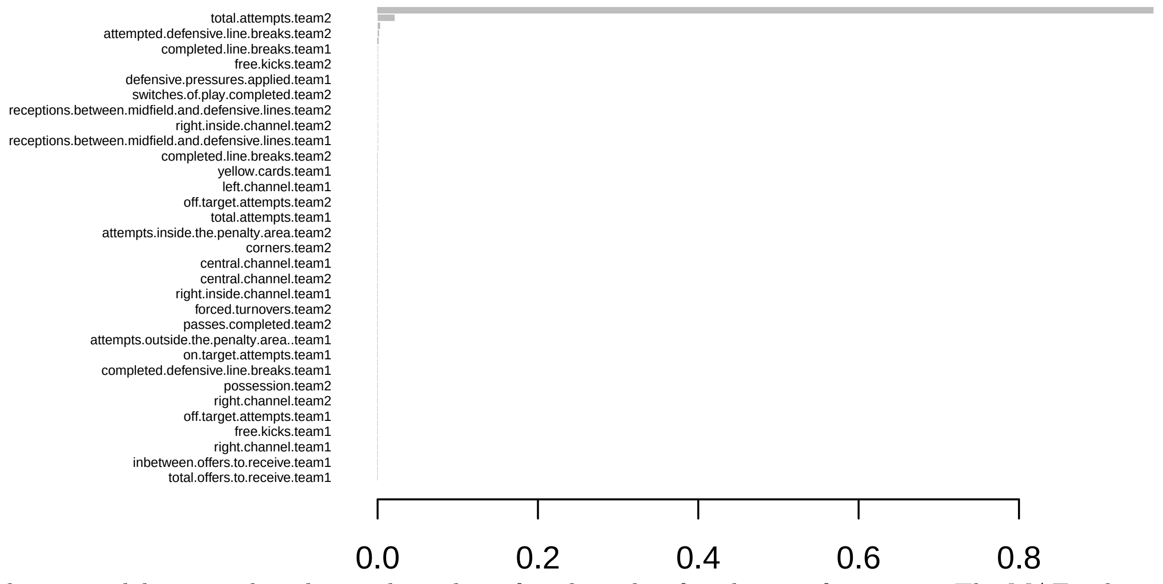
```
#RMSE
caret::RMSE(test_y, y_pred)
```

```
## [1] 0.8259804
```

```r
#Feature Importance
importance <- xgb.importance(feature_names = colnames(train_x), model = xgb_model)
print(importance)
```

```
##                                              Feature         Gain
##                                               <char>        <num>
##  1:                                   total.attempts 9.678584e-01
##  2:                             total.attempts.team2 2.095350e-02
##  3:                 inbehind.offers.to.receive.team2 2.783976e-03
##  4:                attempted.defensive.line.breaks.team2 1.660710e-03
##  5:                            possession.in.contest 1.311086e-03
##  6:                     completed.line.breaks.team1 5.544672e-04
##  7:                                    crosses.team1 4.089936e-04
##  8:                                 free.kicks.team2 3.926047e-04
##  9:                          on.target.attempts.team2 3.499617e-04
## 10:                  defensive.pressures.applied.team1 3.262044e-04
## 11:                                 possession.team1 3.066205e-04
## 12:                 switches.of.play.completed.team2 3.006619e-04
## 13:                        left.inside.channel.team1 2.711760e-04
## 14: receptions.between.midfield.and.defensive.lines.team2 2.294133e-04
## 15:                  infront.offers.to.receive.team1 2.112531e-04
## 16:                       right.inside.channel.team2 1.771938e-04
## 17:                           goal.preventions.team2 1.588886e-04
## 18: receptions.between.midfield.and.defensive.lines.team1 1.568871e-04
## 19:                 inbetween.offers.to.receive.team2 1.547011e-04
## 20:                      completed.line.breaks.team2 1.423262e-04
## 21:             attempts.outside.the.penalty.area..team2 1.321056e-04
## 22:                               yellow.cards.team1 1.154738e-04
## 23:                            crosses.completed.team1 9.661294e-05
## 24:                               left.channel.team1 8.683481e-05
## 25:               attempts.inside.the.penalty.area.team1 6.979150e-05
## 26:                        off.target.attempts.team2 6.815463e-05
## 27:                       attempted.line.breaks.team1 6.672005e-05
## 28:                             total.attempts.team1 6.311372e-05
## 29:                              fouls.against.team2 6.145737e-05
## 30:               attempts.inside.the.penalty.area.team2 5.697983e-05
## 31:                           forced.turnovers.team1 5.589613e-05
## 32:                                    corners.team2 4.857006e-05
## 33:                    total.offers.to.receive.team2 4.319852e-05
## 34:                            central.channel.team1 4.281584e-05
## 35:                       attempted.line.breaks.team2 4.063803e-05
## 36:                            central.channel.team2 3.692065e-05
## 37:                              fouls.against.team1 3.575754e-05
## 38:                       right.inside.channel.team1 3.490957e-05
## 39:                 switches.of.play.completed.team1 1.904818e-05
## 40:                           forced.turnovers.team2 1.800367e-05
## 41:                                    corners.team1 1.548746e-05
## 42:                           passes.completed.team2 1.305868e-05
## 43:                  defensive.pressures.applied.team2 9.648503e-06
## 44:             attempts.outside.the.penalty.area..team1 8.587246e-06
## 45:                                     passes.team1 8.011818e-06
## 46:                          on.target.attempts.team1 7.509195e-06
## 47:                attempted.defensive.line.breaks.team1 6.315991e-06
## 48:                completed.defensive.line.breaks.team1 4.508257e-06
## 49:                            crosses.completed.team2 4.159283e-06
## 50:                                  possession.team2 3.811083e-06
## 51:                 total.attempted.defensive.line.breaks 3.627776e-06
## 52:                                right.channel.team2 3.544055e-06
## 53:                      inbehind.offers.to.receive.team1 3.256426e-06
## 54:                            off.target.attempts.team1 2.628080e-06
## 55:                      infront.offers.to.receive.team2 9.526783e-07
## 56:                                  free.kicks.team1 8.823403e-07
## 57:                            left.inside.channel.team2 6.079539e-07
## 58:                                right.channel.team1 5.209141e-07
## 59:                                      passes.team2 4.993013e-07
## 60:                    inbetween.offers.to.receive.team1 3.001950e-07
## 61:                                 left.channel.team2 6.874079e-08
## 62:                        total.offers.to.receive.team1 2.885974e-08
##                                              Feature         Gain
##            Cover   Frequency
##            <num>       <num>
##  1: 0.1447316103 0.094827586
##  2: 0.0383697813 0.037356322
##  3: 0.0572564612 0.040229885
##  4: 0.0326043738 0.022988506
##  5: 0.0341948310 0.054597701
##  6: 0.0171968191 0.017241379
##  7: 0.0090457256 0.008620690
##  8: 0.0130218688 0.008620690
##  9: 0.0156063618 0.014367816
## 10: 0.0107355865 0.008620690
## 11: 0.0310139165 0.066091954
## 12: 0.0211729622 0.017241379
## 13: 0.0290258449 0.037356322
## 14: 0.0046719682 0.008620690
## 15: 0.0101391650 0.008620690
## 16: 0.0190854871 0.011494253
## 17: 0.0086481113 0.005747126
## 18: 0.0099403579 0.011494253
## 19: 0.0048707753 0.008620690
## 20: 0.0139165010 0.008620690
## 21: 0.0195825050 0.020114943
## 22: 0.0140159046 0.011494253
## 23: 0.0078528827 0.011494253
## 24: 0.0108349901 0.011494253
## 25: 0.0107355865 0.020114943
## 26: 0.0038767396 0.008620690
## 27: 0.0334990060 0.025862069
## 28: 0.0075546720 0.017241379
## 29: 0.0260437376 0.017241379
## 30: 0.0055666004 0.008620690
## 31: 0.0218687873 0.031609195
## 32: 0.0219681909 0.017241379
## 33: 0.0117296223 0.008620690
## 34: 0.0097415507 0.014367816
## 35: 0.0148111332 0.011494253
## 36: 0.0147117296 0.014367816
## 37: 0.0197813121 0.020114943
## 38: 0.0110337972 0.017241379
## 39: 0.0204771372 0.022988506
## 40: 0.0009940358 0.002873563
## 41: 0.0071570577 0.005747126
## 42: 0.0031809145 0.002873563
## 43: 0.0061630219 0.005747126
## 44: 0.0058648111 0.011494253
## 45: 0.0048707753 0.002873563
## 46: 0.0170974155 0.022988506
## 47: 0.0073558648 0.008620690
## 48: 0.0106361829 0.008620690
## 49: 0.0299204771 0.022988506
## 50: 0.0016898608 0.005747126
## 51: 0.0013916501 0.002873563
## 52: 0.0068588469 0.005747126
## 53: 0.0020874751 0.005747126
## 54: 0.0108349901 0.017241379
## 55: 0.0226640159 0.017241379
## 56: 0.0101391650 0.008620690
## 57: 0.0030815109 0.002873563
## 58: 0.0050695825 0.005747126
## 59: 0.0070576541 0.005747126
## 60: 0.0189860835 0.014367816
## 61: 0.0030815109 0.005747126
## 62: 0.0028827038 0.005747126
##            Cover   Frequency
```

```r
#Plot feature importance
xgb.plot.importance(importance_matrix = importance)
```



The xgboost model can predict the total number of goals with a fair degree of accuracy. The MAE is less than one (0.6715505).

We have seen earlier that our dataset contains a high degree of multicollinearity. Xgboost handles multicollinearity quite neatly on its own. Also, our dataset contains more predictors than observations. Xgboost can also handle this efficiently. To offer some evidence for the previous statements, let's try to build a linear regression model with the same variables we used for the xgboost model. I suspect that the performance of this model will be poor without employing any kind of method to deal with the multicollinearity or any kind of dimension reduction.

Comparison to linear model

```r
#Linear regression model
lin_model <- lm(total.goals~., data=as.data.frame(train_data_reg))
#Predictions
lin_pred <- predict(lin_model, test_data_reg)
```

```
## Warning in predict.lm(lin_model, test_data_reg): prediction from rank-deficient
## fit; attr(*, "non-estim") has doubtful cases
```

```r
#Evaluate performance
#MSE
mean((test_y - lin_pred)^2)
```

```
## [1] 2955.616
```

```r
#MAE
caret::MAE(test_y, lin_pred)
```

```
## [1] 38.34181
```

```r
#RMSE
caret::RMSE(test_y, lin_pred)
```

```
## [1] 54.36558
```

```r
mean(df$total.goals)
```

```
## [1] 2.6875
```

As suspected, the model's performance is poor. All three of the evaluation metrics are horrendous considering that the mean number of total goals is 2.6875. Furthermore, the fit is rank-deficient since there are more predictors than observations. This results in very poor predictions. This example illustrates the importance of reducing multicollinearity and variable selection.