2024 NBA Playoffs Prediction

Lina Nguyen

01/14/25

Load Data

```
library(tidyverse)
# library(DataExplorer)
library(ggplot2)
player_data <- read_csv("/Users/linanguyen/Desktop/OKC/Provided Data/player_game_data.csv")
team_data <- read_csv("/Users/linanguyen/Desktop/OKC/Provided Data/team_game_data.csv")
```

```
# EDA for player_data
# DataExplorer::create_report(player_data)
```

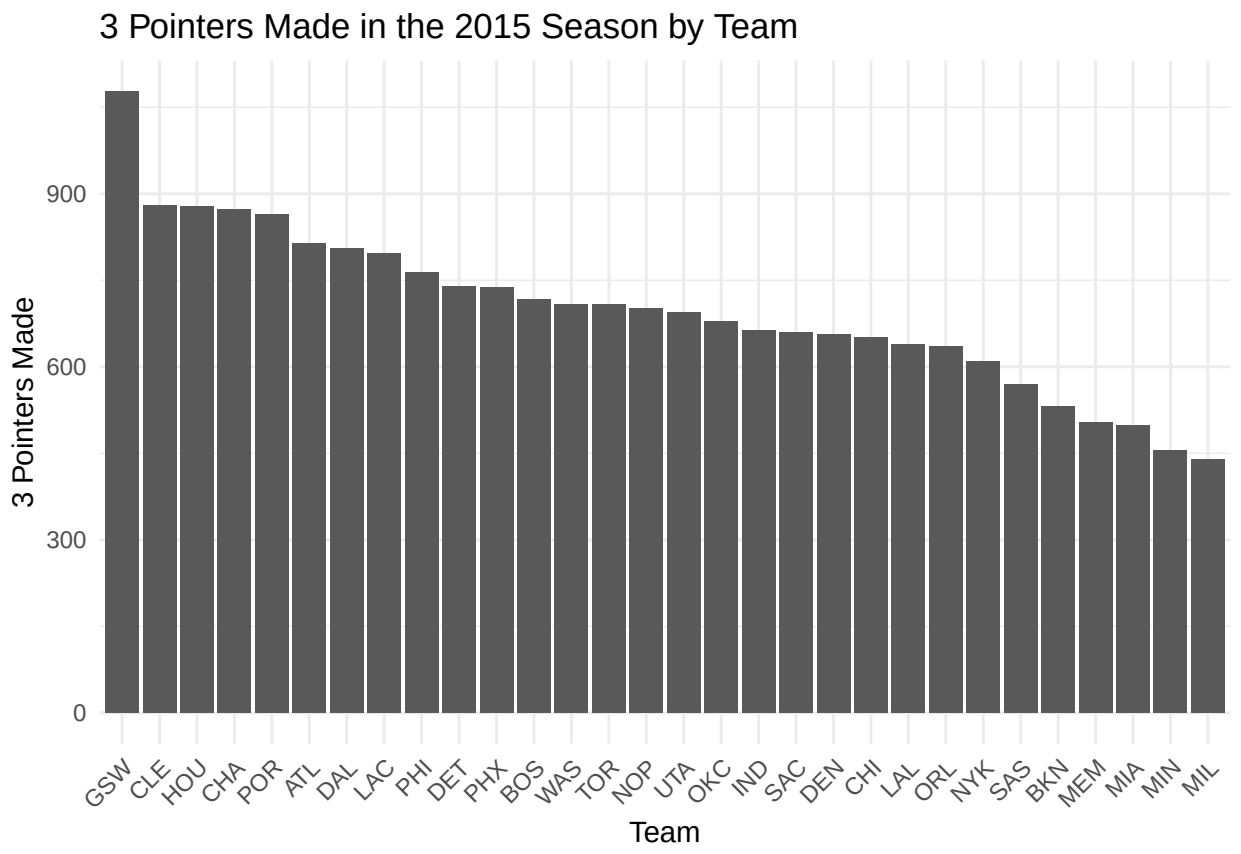
```
# EDA for team_data
# DataExplorer::create_report(team_data)
```

```
# View GSW efg and oefg
# eFG% = (P2M + 1.5*P3M)/(P2A + P3A)

# subset the dataset to get the 2015 regular season for GSW on offense and defense
GSWO2015 <- subset(team_data, season == 2015 & gametype == 2 & off_team == "GSW")
GSWD2015 <- subset(team_data, season == 2015 & gametype == 2 & def_team == "GSW")

# calculate eFG%
GSWOeFG <- sum((GSWO2015$fg2made + 1.5*GSWO2015$fg3made)/(GSWO2015$fgattempted))/nrow(GSWO2015)
GSWDeFG <- sum((GSWD2015$fg2made + 1.5*GSWD2015$fg3made)/(GSWD2015$fgattempted))/nrow(GSWD2015)
```

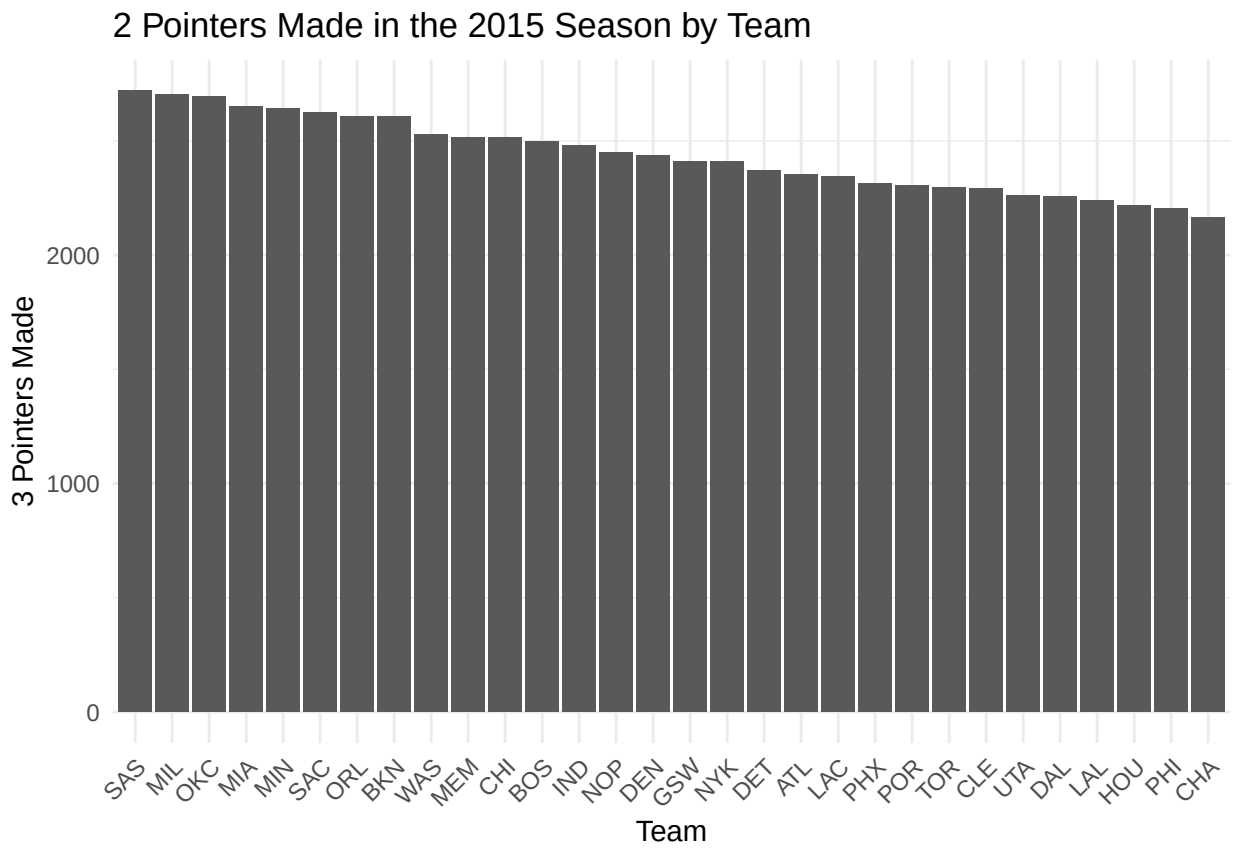
Warriors Offensive in 2015-2016: 56.4% eFG
Warriors Defensive in 2015-2016: 47.9% eFG

```
szn15 <- subset(team_data, season == 2015 & gametype == 2)
# exploring eFG further
# 3pts
szn15_agg <- aggregate(fg3made ~ off_team, data = szn15, sum)

# Sort the aggregated data by 3 pointers made in descending order
szn15_agg <- szn15_agg[order(-szn15_agg$fg3made), ]

szn15_agg$off_team <- factor(szn15_agg$off_team, levels = szn15_agg$off_team)
```

```r
# Create the bar chart
ggplot(szn15_agg, aes(x = off_team, y = fg3made)) +
  geom_bar(stat = 'identity') +
  labs(title = "3 Pointers Made in the 2015 Season by Team", x = "Team", y = "3 Pointers Made") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

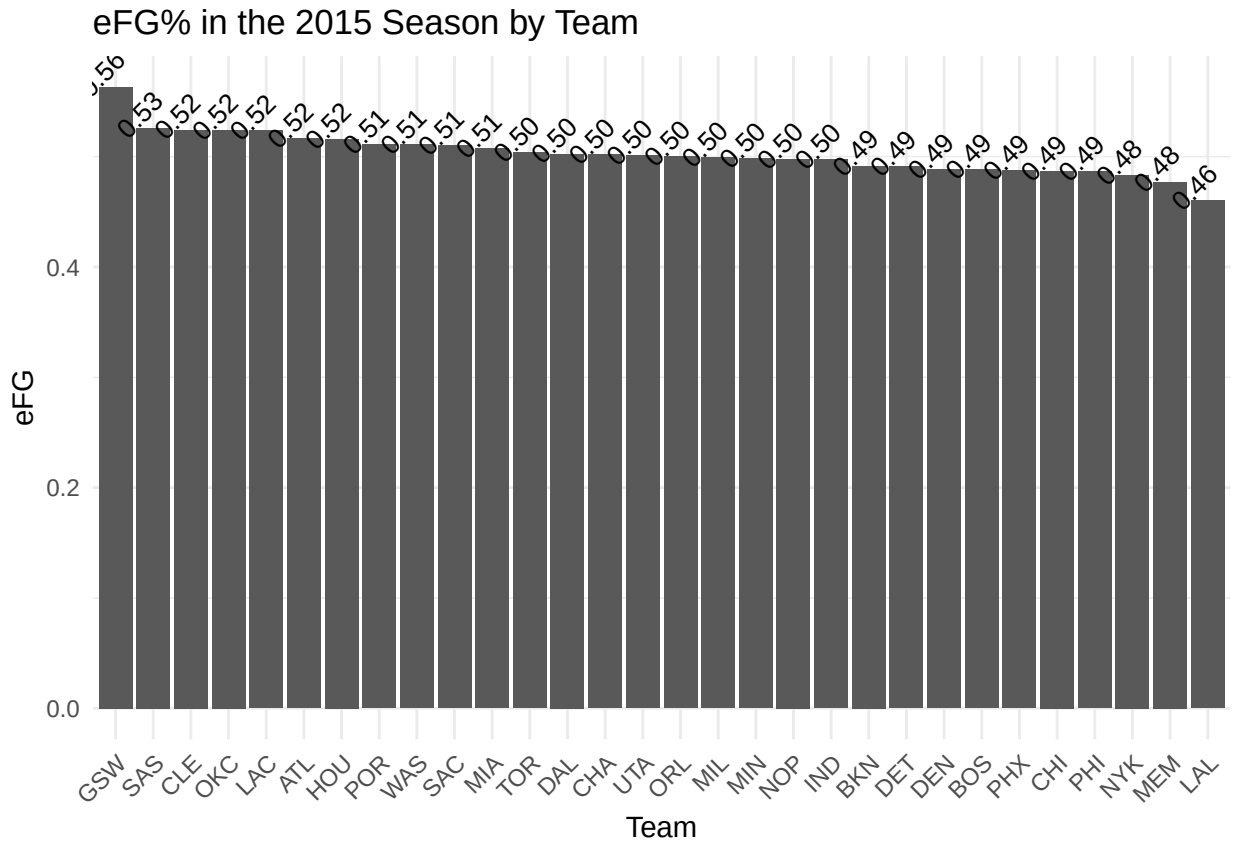



```r
# eFG% = (P2M + 1.5*P3M)/(P2A + P3A)
# The 2015-2016 GSW were known for their 3 point shooting, and the graphic below shows that they signif

# 2pts
szn15_agg_2 <- aggregate(fg2made ~ off_team, data = szn15, sum)

# Sort the aggregated data by 3 pointers made in descending order
szn15_agg_2 <- szn15_agg_2[order(-szn15_agg_2$fg2made), ]

# Convert off_team to a factor with levels ordered by fg2made
szn15_agg_2$off_team <- factor(szn15_agg_2$off_team, levels = szn15_agg_2$off_team)

# Create the bar chart
ggplot(szn15_agg_2, aes(x = off_team, y = fg2made)) +
  geom_bar(stat = 'identity') +
  labs(title = "2 Pointers Made in the 2015 Season by Team", x = "Team", y = "3 Pointers Made") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# On the contrary, GSW came up around average on 2 pointers made compared to the rest of the NBA.

# eFG% for all teams
efg2015 <- subset(
  team_data,
  gametype ==2 & season == 2015,
  select = c(season, gametype, off_team, fg2made, fg3made, fgattempted)
)

efg2015_agg <- efg2015 %>%
  group_by(off_team) %>%
  summarise(fg2made = sum(fg2made), fg3made = sum(fg3made), fgattempted = sum(fgattempted))

efg2015_agg$efg <- (efg2015_agg$fg2made + 1.5 * efg2015_agg$fg3made) / efg2015_agg$fgattempted


# Sort the aggregated data by 3 pointers made in descending order
efg2015_agg <- efg2015_agg[order(-efg2015_agg$efg), ]

# Convert off_team to a factor with levels ordered by fg2made
efg2015_agg$off_team <- factor(efg2015_agg$off_team, levels = efg2015_agg$off_team)

# Create the bar chart
ggplot(efg2015_agg, aes(x = off_team, y = efg)) +
  geom_bar(stat = 'identity') +
  labs(title = "eFG% in the 2015 Season by Team", x = "Team", y = "eFG") +
```

```
  geom_text(aes(label = sprintf("%.2f", efg)), vjust = -0.5, color = "black", size = 3.5, angle = 45) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# The 2015 GSW's three point game played a huge role in their eFG%, and 0.03% higher than second place
```

```
# There are two rows for every nbagameid, and this will be split up using off_home = 1 or 0. For the da

# subset data to 2014-2023 regular season
reg14_23 <- subset(
  team_data[, c("nbagameid", "off_team", "def_team", "off_win", "season", "gametype", "fg2made", "fg3ma
  season >= 2014 & season <= 2023 & gametype == 2
)

# split reg14_23 into off_home = 1, and off_home = 0
reg14_23_1 <- subset(
  reg14_23,
  off_home == 1
)

reg14_23_0 <- subset(
  reg14_23,
  off_home == 0
)
```

```r
# calculate efg%. Offense will be the the off_team when off_home = 1 in the merge later
reg14_23_1$Oefg <- (reg14_23_1$fg2made + 1.5*reg14_23_1$fg3made)/(reg14_23_1$fgattempted)
reg14_23_0$Defg <- (reg14_23_0$fg2made + 1.5*reg14_23_0$fg3made)/(reg14_23_0$fgattempted)

# combine both dataframes, but only include Defg
reg14_23_0 <- reg14_23_0 %>%
  select(nbagameid, Defg)

combined_df <- merge(reg14_23_1, reg14_23_0, by = "nbagameid")

# Drop rows where OeFG and DeFG are equal
combined_unique <- combined_df %>%
  filter(Oefg != Defg) %>%
  select(-season, -gametype, -fg2made, -fg3made, -fgattempted, -off_home)

# find the percentage of times team with higher eFG wins that game
combined_unique <- combined_unique %>%
  mutate(higher_efg_W = case_when(
    Oefg > Defg & off_win == 1 ~ 1,
    Oefg > Defg & off_win == 0 ~ 0,
    Oefg < Defg & off_win == 1 ~ 0,
    Oefg < Defg & off_win == 0 ~ 1
  ))

higher_efg_W <- sum(combined_unique$higher_efg_W == 1) / nrow(combined_unique)
higher_efg_W
```

```
## [1] 0.8160141
```

Percent time team with higher eFG% wins a game: 81.6%

```r
# OREB% = reboffensive/reboundchance

# subset data to 2014-2023 regular season
reb14_23 <- subset(
  team_data[, c("nbagameid", "off_team", "def_team", "off_win", "season", "reboffensive", "reboundchanc
  season >= 2014 & season <= 2023 & gametype == 2
)

# split reg14_23 into off_home = 1, and off_home = 0
reb14_23_1 <- subset(
  reb14_23,
  off_home == 1
)

reb14_23_0 <- subset(
  reb14_23,
  off_home == 0
)

# calculate OREB%. Offense will be the the off_team when off_home = 1 in the merge later
reb14_23_1$Oreb <- reb14_23_1$reboffensive/reb14_23_1$reboundchance
reb14_23_0$Dreb <- reb14_23_0$reboffensive/reb14_23_0$reboundchance
```

```r
# combine both dataframes, but only include Defg
reb14_23_0 <- reb14_23_0 %>%
  select(nbagameid, Dreb)

combined_df_reb <- merge(reb14_23_1, reb14_23_0, by = "nbagameid")

# Drop rows where OeFG and DeFG are equal
combined_unique_reb <- combined_df_reb %>%
  filter(Oreb != Dreb) %>%
  select(-season, -gametype, -off_home, -off_team, -def_team, -reboundchance, -nbagameid)

# find the percentage of times team with higher Oreb wins that game
combined_unique_reb <- combined_unique_reb %>%
  mutate(higher_Oreb_W = case_when(
    Oreb > Dreb & off_win == 1 ~ 1,
    Oreb > Dreb & off_win == 0 ~ 0,
    Oreb < Dreb & off_win == 1 ~ 0,
    Oreb < Dreb & off_win == 0 ~ 1
  ))

higher_Oreb_W <- sum(combined_unique_reb$higher_Oreb_W == 1) / nrow(combined_unique_reb)
higher_Oreb_W
```

```
## [1] 0.5562274
```

Percent team with more Oreb wins game: 55.6%

```r
# subset data by players who have scored at least 25 points per game played in the 2014-2023 regular se
# calculate

# subset data by season regular 2014-2023 season, players who have scored at least 25 points
player25 <- subset(
  player_data[, c("nbagameid", "nbapersonid", "team", "season", "points", "gametype")],
  season >= 2014 & season <= 2023 & gametype == 2 & points >= 25)

# count number of times a player played on that team per season
playergamecount <- player25 %>%
  select(nbagameid, nbapersonid, team, season) %>%
  group_by(nbapersonid, team, season) %>%
  summarise(playergamecount = n(), .groups = 'drop')

# count number of games a team has per season
gamecount <- team_data %>%
  filter(season >= 2014 & season <= 2023 & gametype == 2) %>%
  select(nbagameid, off_team, season)  %>%
  group_by(off_team, season) %>%
  summarise(gamecount = n(), .groups = 'drop') %>%
  rename(team = off_team)

# find percentage of amount of games played by the player per season
merged_df <- left_join(playergamecount, gamecount, by = c("team", "season"))
merged_df$playeravail <- merged_df$playergamecount/merged_df$gamecount
```

```r
# filter players who played at least 25% of all their games
avgavail25 <- merged_df %>%
  filter(playeravail >= 0.25) %>%
  select(nbapersonid, season, team, playeravail)

# of avgavail25, what percent of games were they available on average?
# player_data to find merge with avgavail25 on nbapersonid to get missed count
avgavail25_all_games <- avgavail25 %>%
  left_join(player_data, by = c("nbapersonid", "season", "team")) %>%
  select(nbapersonid, season, team, missed, playeravail)

avgavail25_percent_avail <- sum(avgavail25_all_games$missed == 0)/nrow(avgavail25_all_games)
```

Average Availability vailability for players who play 25% of possible in-season games, and score at least 25 points for 2014-2023: 86.3% of games

```r
# What % of playoff series are won by the team with home court advantage? Give your answer by round. Us

# team_data will be used to find playoff data from 2014-2021 season, where the off_home = 1.
# figuring out which games are being played in which round:
# step 1: group by season, unique off_team, and def_team where the order doesn't matter. Get the date o
# step 2: group by the season, and unique off_team and def_team combination, sort by the dates, and num

# subset gametype = 4, season = 2014-2021, off_home = 1
playoff <- subset(
  team_data[, c("season", "gametype", "nbagameid", "off_win", "off_home", "off_team", "def_team", "game
  season >= 2014 & season <= 2021 & gametype == 4 & off_home == 1
)

# create helper columns that organizes the orders of off_team and def_team to create an easier comparis
playoff1 <- playoff %>%
  mutate(team_min = pmin(off_team, def_team),
         team_max = pmax(off_team, def_team))

# add the first gamedate to gbgamecount
firstgamedate <- playoff1 %>%
  select(team_min, team_max, season, gamedate) %>%
  group_by(team_min, team_max, season) %>%
  arrange(gamedate) %>%
  slice(1) %>%
  ungroup()

# group by season, sort by gamedate. slice first 8 = round 1, slice 9-12 = round 2, slice 13-14 = round
playoff_round1 <- firstgamedate %>%
  group_by(season) %>%
  arrange(gamedate) %>%
  slice(1:8)

playoff_round2 <- firstgamedate %>%
  group_by(season) %>%
  arrange(gamedate) %>%
  slice(9:12)
```

```r
playoff_round3 <- firstgamedate %>%
  group_by(season) %>%
  arrange(gamedate) %>%
  slice(13:14)

playoff_round4 <- firstgamedate %>%
  group_by(season) %>%
  arrange(gamedate) %>%
  slice(15)

# merge playoff with playoff_round(n) to calculate percent wins at home
round1 <- playoff_round1 %>%
  left_join(playoff1, by = c("season", "team_min", "team_max", "gamedate")) %>%
  select(-off_team, -def_team, -gametype)


round1_home_wins <- sum(round1$off_win) / nrow(round1)

round2 <- playoff_round2 %>%
  left_join(playoff1, by = c("season", "team_min", "team_max", "gamedate")) %>%
  select(-off_team, -def_team, -gametype)

round2_home_wins <- sum(round2$off_win) / nrow(round2)

round3 <- playoff_round3 %>%
  left_join(playoff1, by = c("season", "team_min", "team_max", "gamedate")) %>%
  select(-off_team, -def_team, -gametype)

round3_home_wins <- sum(round3$off_win) / nrow(round3)

round4 <- playoff_round4 %>%
  left_join(playoff1, by = c("season", "team_min", "team_max", "gamedate")) %>%
  select(-off_team, -def_team, -gametype)

round4_home_wins <- sum(round4$off_win) / nrow(round4)

round1_home_wins
```

```
## [1] 0.703125
```

```r
round2_home_wins
```

```
## [1] 0.59375
```

```r
round3_home_wins
```

```
## [1] 0.625
```

```r
round4_home_wins
```

```
## [1] 0.875
```

Home Team Wins for Playoffs:

Round 1: 70.3%
Round 2: 59.4%
Conference Finals: 62.5%
Finals: 87.5%

```
# team_data shows 2 rows for every nbagameid. To get net ranking, team_data will be subsetted into off_
# net rating = [ORTG]points/(possessions/100) - [DRTG]points allowed/(defensive possessions/ 100)
# The 2 subsetted data sets will be merged back together to calculate net rankings. Offense and defense

# subset data for regular season, 2014-2021 season, by offense and defense
off_sub <- subset(
  team_data,
  season >= 2014 & season <= 2021 & gametype == 2 & off_home ==1,
  select = c("season", "nbagameid", "off_team", "possessions", "points", "def_team", "off_home")
)

def_sub <- subset(
  team_data,
  season >= 2014 & season <= 2021 & gametype == 2 & off_home == 0,
  select = c("season", "nbagameid", "off_team", "possessions", "points", "def_team", "off_home")
)

# calculate ORTG and DRTG
off_sub$ORTG <- off_sub$points / (off_sub$possessions / 100)
def_sub$DRTG <- def_sub$points / (def_sub$possessions / 100)

# merging the datasets
nr <- off_sub %>%
  full_join(def_sub, by = "nbagameid") %>%
  select(season.x, nbagameid, off_team.x, ORTG, def_team.x, DRTG)

# rename columns
colnames(nr) <- c("season", "nbagameid", "off_team", "ORTG", "def_team", "DRTG")

# calculate net rankings
nr$Onet_ranking <- nr$ORTG - nr$DRTG
nr$Dnet_ranking <- nr$DRTG - nr$ORTG

# calculate team season net ranking
off_nr <- nr %>%
  group_by(season, off_team) %>%
  summarise(mean_o_nr = mean(Onet_ranking, na.rm = TRUE), .groups = 'drop') %>%
  rename(team = off_team)

def_nr <- nr %>%
  group_by(season, def_team) %>%
  summarise(mean_d_nr = mean(Dnet_ranking, na.rm = TRUE), .groups = 'drop') %>%
  rename(team = def_team)

net_ranking <- off_nr %>%
  full_join(def_nr, by = c("season", "team")) %>%
  mutate(net_ranking = (mean_o_nr + mean_d_nr) / 2) %>%
  select(season, team, net_ranking)
```

```r
# select teams that have a +5 net rating
net_5 <- subset(
  net_ranking,
  net_ranking >= 5
)


# create dataframe of teams that play in round 2 the following play off year

# subset gametype = 4, season = 2014-2021, off_home = 1
playoff2 <- subset(
  team_data[, c("season", "gametype", "nbagameid", "off_win", "off_home", "off_team", "def_team", "game
  season >= 2015 & season <= 2022 & gametype == 4 & off_home == 1
)

# create helper columns that organizes the orders of off_team and def_team to create an easier comparis
playoff3 <- playoff2 %>%
  mutate(team_min = pmin(off_team, def_team),
         team_max = pmax(off_team, def_team))

# add the first gamedate to gbgamecount
firstgamedate1 <- playoff3 %>%
  select(team_min, team_max, season, gamedate) %>%
  group_by(team_min, team_max, season) %>%
  arrange(gamedate) %>%
  slice(1) %>%
  ungroup()

# group by season, sort by gamedate. slice first 8 = round 1, slice 9-12 = round 2, slice 13-14 = round

Q7_playoff_round2 <- firstgamedate1 %>%
  group_by(season) %>%
  arrange(gamedate) %>%
  slice(9:12)

# merge playoff with playoff_round(n) to calculate percent wins at home
Q7_round2 <- Q7_playoff_round2 %>%
  left_join(playoff3, by = c("season", "team_min", "team_max", "gamedate")) %>%
  select(-off_team, -def_team, -gametype)


# create dataframe of list of teams that played in round 2

Q7_round2_team_min <- Q7_round2 %>%
  select(season, team_min) %>%
  mutate(season = season - 1) %>%
  rename(team = team_min)

Q7_round2_team_max <- Q7_round2 %>%
  select(season, team_max) %>%
  mutate(season = season - 1) %>%
  rename(team = team_max)
```

```r
Q7_round2_teams <- rbind(Q7_round2_team_min, Q7_round2_team_max)

# merge net_5 with Q7_round2_teams
net_5_success <- net_5 %>%
  inner_join(Q7_round2_teams, by = c("season", "team")) %>%
  select(season, team, net_ranking)

# calculate percentage of teams that make it to the second round of the playoffs with +5 net ranking in
net_5_success_percentage <- nrow(net_5_success) / nrow(net_5)


# On teams that have a +5 rating in regular season and make it to the second round of the playoffs, wha
# net_5_success will tell us teams with +5 ratings and make it to the second round of play offs

# subset player_data by 2014-2021 regular season, add up total minutes played by player, team, and seas
player_reg <- subset(
  player_data,
  gametype == 2 & season >= 2014 & season <= 2021,
  select = c("season", "nbapersonid", "team", "seconds", "gametype")
)

player_reg_seconds <- player_reg %>%
  group_by(season, team, nbapersonid) %>%
  summarise(total_seconds = sum(seconds), .groups = 'drop')

top_5_player_reg <- player_reg_seconds %>%
  group_by(season, team) %>%
  arrange(desc(total_seconds), .by_group = TRUE) %>%
  slice(1:5)

# subset player_data by 2014-2021 playoffs
player_playoffs <- subset(
  player_data,
  gametype == 4 & season >= 2014 & season <= 2021,
  select = c("season", "nbapersonid", "team", "seconds", "gametype", "nbagameid")
)

# merge round2 with player_playoffs to find players on each team who were in the playoffs
round2_players <- round2 %>%
  left_join(player_playoffs, by = c("nbagameid", "season")) %>%
  select(-team_min, -team_max)

# merge round2_players with top_5_player_reg to find if top 5 players are playing in round 2
top5_round2_players <- top_5_player_reg %>%
  left_join(round2_players, by =c("nbapersonid", "team", "season")) %>%
  drop_na()

top5_players_percentage_playoffs <- nrow(top5_round2_players) / nrow(top_5_player_reg)

net_5_success_percentage
```

```
## [1] 0.6363636
```

```
top5_players_percentage_playoffs
```

```
## [1] 0.2666667
```

Percent of +5.0 net rating teams making the 2nd round next year: 63.6%
Percent of top 5 minutes played players who played in those 2nd round series: 26.7%

Playoffs Modeling

```
# NBA Play offs: 16 teams - 8 each from East and West Conferences
# Teams in each conference ranked by win/loss record from the regular season
# Top 6 teams from each conference automatically progress to the playoffs
# 7-10th place teams in each conference play for the remaining four spots in the play in tournament

# play in tournament: 7-10th place teams play each other. 7-8th place play each other and handed the 7t

# NBA Play offs seed: top 6 is W/L record in regular season
# if there is a tie, head-to-head record is used to determine which team receives the better seed
# if there is a tie higher seed goes to a divisional champion
# if there is a tie, win-loss record against teams in their division
# if there is a tie, win-loss record percentage against conference teams
# 7-8th seed is decided by play-in tournament

# NBA Playoffs: best of 7 elimination format
# first round: 1st v 8th, 2nd vs 7th, 3rd vs 6th, 4th vs. 5th
# team with better regular-season rewarded with home-court advantage
# home/away: 2-2-1-1-1
# teams with a better seed plays games 1,2,5,7 at home

# Part 1: Which teams make it to the playoffs based on their regular season performance
# we will only look at 2023-2024 season
# break data into eastern and western conference
# team_data: regular season (gametype = 2) used to predict which teams will make it to the top 6 play o
# use probability to predict chances of winning per team and game
# Find features that play the most significant role in leading a team to the playoffs
```

```
# look at team_data 2023-2024 regular season

reg_szn2023 <- team_data %>%
  filter(
    season == 2023 & gametype == 2
  ) %>%
  mutate(
    eFG = (fg2made + 1.5 * fg3made) / fgattempted,
    TORatio = turnovers / possessions,
    reb_percentage = reboffensive / reboundchance,
    FTRate = ftmade / fgattempted,
    west_conference = case_when(
      off_team %in% c("OKC", "DEN", "MIN", "LAC", "DAL", "PHX", "LAL", "NOP", "SAC", "GSW", "HOU", "UTA
      TRUE ~ 0
```

```
    )
  ) %>%
  group_by(nbagameid) %>%
  mutate(win = ifelse(points == max(points), 1, 0)) %>%
  ungroup() %>%
  select(nbagameid, off_team, eFG, TORatio, reb_percentage, FTRate, off_home, points, def_team, west_co

head(reg_szn2023)
```

```
## # A tibble: 6 x 11
##   nbagameid off_team   eFG TORatio reb_p~1 FTRate off_h~2 points def_t~3 west_~4
##       <dbl> <chr>    <dbl>   <dbl>   <dbl>  <dbl>   <dbl>  <dbl> <chr>     <dbl>
## 1  22300197 MIN      0.484  0.0667   0.404  0.283       0    115 PHX           1
## 2  22300764 TOR      0.439  0.178    0.283  0.133       1     99 SAS           0
## 3  22300324 OKC      0.485  0.131    0.356  0.235       0    123 SAC           1
## 4  22300002 MIL      0.549  0.141    0.18   0.244       1    110 NYK           0
## 5  22301216 TOR      0.538  0.141    0.383  0.172       0    116 CHA           0
## 6  22300027 MIL      0.674  0.127    0.143  0.112       0    130 CHA           0
## # ... with 1 more variable: win <dbl>, and abbreviated variable names
## #   1: reb_percentage, 2: off_home, 3: def_team, 4: west_conference
```

```
# view team wins and rankings to check with the model later
team_wins <- reg_szn2023 %>%
  group_by(off_team) %>%
  summarise(total_wins = sum(win), total_games = n()) %>%
  ungroup() %>%
  mutate(
    rank = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15
    )
  )

head(team_wins)
```

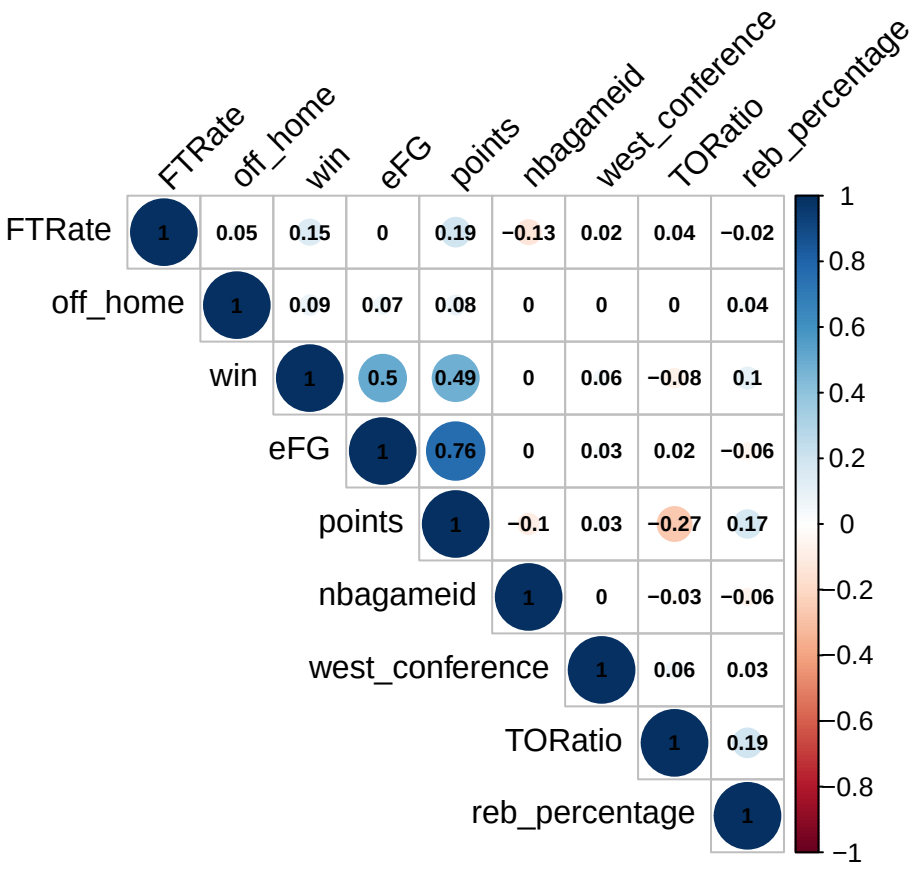
```
## # A tibble: 6 x 4
##   off_team total_wins total_games  rank
##   <chr>         <dbl>       <int> <dbl>
## 1 ATL              36          82    10
## 2 BKN              32          82    11
```

```
## 3 BOS              64          82     1
## 4 CHA              21          82    13
## 5 CHI              39          82     9
## 6 CLE              48          82     4
```

```
# view correlation plot to see if we have to address multicollinearity
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
corr <- reg_szn2023 %>%
  select(-off_team, -def_team)

cor_matrix <- cor(corr)
corrplot(cor_matrix, method = "circle", type = "upper", order = "hclust",
         addCoef.col = "black", tl.col = "black", tl.srt = 45,
         number.cex = 0.7)
```

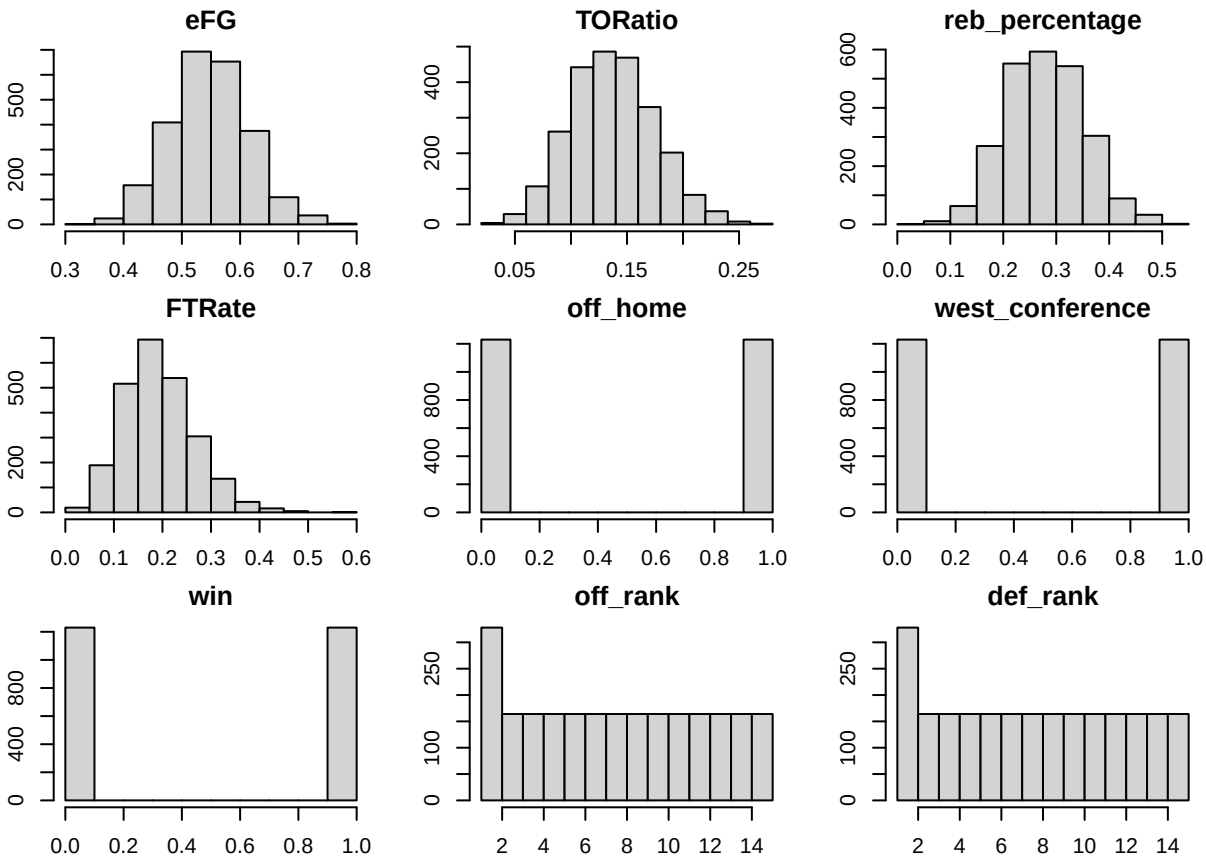



```
# points will be dropped because it is obviously heavily correlated with eFG
reg_szn2023 <- reg_szn2023 %>%
  select(-points, -nbagameid)
```

```
# add end of season rankings to the model
reg_szn2023 <- reg_szn2023 %>%
```

```
  mutate(
    off_rank = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15,
      TRUE ~ NA_real_),
    def_rank = case_when(def_team %in% c("BOS", "OKC") ~ 1,
      def_team %in% c("NYK", "DEN") ~ 2,
      def_team %in% c("MIL", "MIN") ~ 3,
      def_team %in% c("CLE", "LAC") ~ 4,
      def_team %in% c("ORL", "DAL") ~ 5,
      def_team %in% c("IND", "PHX") ~ 6,
      def_team %in% c("PHI", "NOP") ~ 7,
      def_team %in% c("MIA", "LAL") ~ 8,
      def_team %in% c("CHI", "SAC") ~ 9,
      def_team %in% c("ATL", "GSW") ~ 10,
      def_team %in% c("BKN", "HOU") ~ 11,
      def_team %in% c("TOR", "UTA") ~ 12,
      def_team %in% c("CHA", "MEM") ~ 13,
      def_team %in% c("WAS", "SAS") ~ 14,
      def_team %in% c("DET", "POR") ~ 15,
      TRUE ~ NA_real_)
)
```

```
# view distribution of data
# data sets give a relatively normal distribution. There is no need to rescale the data to keep the int
numeric_columns <- sapply(reg_szn2023, is.numeric)

# set margins
par(mar = c(2,2,2,2))
par(mfrow = c(3, 3))
for (col in names(reg_szn2023)[numeric_columns]) {
  hist(reg_szn2023[[col]], main = col, xlab = col)
}
```




```
# scale data so ranking doesn't skew the results
#reg_szn2023$off_rank <- scales::rescale(reg_szn2023$off_rank)
#reg_szn2023$def_rank <- scales::rescale(reg_szn2023$def_rank)
```

```
# split the data into training and testing
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.2
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
set.seed(2581)
split_index <- sample(1:nrow(reg_szn2023), 0.7*nrow(reg_szn2023))
train <- reg_szn2023[split_index, ]
test <- reg_szn2023[-split_index, ]
```

```r
# use h2o automal to retrieve a baseline model
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```r
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /var/folders/13/__jfy01d5nxd0qk1cmrhcpzh0000gn/T//RtmpWw7eps/file6d8523f0465c/h2o_linanguyen_sta
##     /var/folders/13/__jfy01d5nxd0qk1cmrhcpzh0000gn/T//RtmpWw7eps/file6d85739b5472/h2o_linanguyen_sta
##
##
## Starting H2O JVM and connecting: .... Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         4 seconds 535 milliseconds
##     H2O cluster timezone:       America/Los_Angeles
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.44.0.3
##     H2O cluster version age:    1 year and 25 days
##     H2O cluster name:           H2O_started_from_R_linanguyen_cfp368
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.54 GB
##     H2O cluster total cores:    8
```

```
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.0 (2021-05-18)

## Warning in h2o.clusterInfo():
## Your H2O cluster version is (1 year and 25 days) old. There may be a newer version available.
## Please download and install the latest version from: https://h2o-release.s3.amazonaws.com/h2o/latest
```

```
# convert to h2o objects
train_h2o <- as.h2o(train)
```

```
##   |                                                                      |
```

```
test_h2o <- as.h2o(test)
```

```
##   |                                                                      |
```

```
target <- "win"
features <- setdiff(names(train), target)

# train h2o model
model <- h2o.automl(
  x = features,
  y = target,
  training_frame = train_h2o,
  max_models = 10,
  seed = 1
)
```

```
##   |                                                                      |
## 20:08:33.56: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:33.56: _response param, We have detected that your response column has only 2 unique values (0
## 20:08:35.528: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:35.528: _response param, We have detected that your response column has only 2 unique values (
## 20:08:36.712: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:36.712: _response param, We have detected that your response column has only 2 unique values (
## 20:08:38.816: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:38.816: _response param, We have detected that your response column has only 2 unique values (
## 20:08:39.535: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:39.535: _response param, We have detected that your response column has only 2 unique values (
## 20:08:42.784: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:42.784: _response param, We have detected that your response column has only 2 unique values (
## 20:08:43.536: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:43.537: _response param, We have detected that your response column has only 2 unique values (
## 20:08:43.994: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:43.995: _response param, We have detected that your response column has only 2 unique values (
## 20:08:44.371: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:44.371: _response param, We have detected that your response column has only 2 unique values (
```

```
## 20:08:44.833: _train param, Dropping bad and constant columns: [off_team, def_team]
## 20:08:44.833: _response param, We have detected that your response column has only 2 unique values (
## 20:08:46.148: _train param, Dropping unused columns: [off_team, def_team]
## 20:08:46.148: _response param, We have detected that your response column has only 2 unique values (
##
## 20:08:47.272: _train param, Dropping unused columns: [off_team, def_team]
## 20:08:47.272: _response param, We have detected that your response column has only 2 unique values (
```

```
best_model <- model@leader
lb <- model@leaderboard
print(lb, n = nrow(lb))
```

```
##                                                 model_id      rmse       mse
## 1  StackedEnsemble_BestOfFamily_1_AutoML_1_20250114_200832 0.3924666 0.1540300
## 2     StackedEnsemble_AllModels_1_AutoML_1_20250114_200832 0.3927144 0.1542246
## 3                           GBM_1_AutoML_1_20250114_200832 0.3955064 0.1564253
## 4                           GLM_1_AutoML_1_20250114_200832 0.3976485 0.1581243
## 5                           GBM_2_AutoML_1_20250114_200832 0.4018911 0.1615165
## 6                           GBM_3_AutoML_1_20250114_200832 0.4031384 0.1625206
## 7                           XRT_1_AutoML_1_20250114_200832 0.4034089 0.1627387
## 8                           DRF_1_AutoML_1_20250114_200832 0.4037745 0.1630338
## 9                           GBM_4_AutoML_1_20250114_200832 0.4040789 0.1632798
## 10                      XGBoost_3_AutoML_1_20250114_200832 0.4196714 0.1761240
## 11                      XGBoost_1_AutoML_1_20250114_200832 0.4404393 0.1939867
## 12                      XGBoost_2_AutoML_1_20250114_200832 0.4485575 0.2012038
##          mae     rmsle mean_residual_deviance
## 1  0.3214864 0.2760597              0.1540300
## 2  0.3215668 0.2761749              0.1542246
## 3  0.3253369 0.2780840              0.1564253
## 4  0.3393767 0.2813541              0.1581243
## 5  0.3251146 0.2819346              0.1615165
## 6  0.3251601 0.2828327              0.1625206
## 7  0.3289355 0.2827843              0.1627387
## 8  0.3237588 0.2830241              0.1630338
## 9  0.3219219 0.2834259              0.1632798
## 10 0.3290816 0.2938251              0.1761240
## 11 0.3490783 0.3099957              0.1939867
## 12 0.3497598 0.3152077              0.2012038
##
## [12 rows x 6 columns]
```

```
# predict the test set
predictions <- h2o.predict(best_model, test_h2o)
```

```
##   |                                                                              |
```

```
# convert predictions to a dataframe
predictions_df <- as.data.frame(predictions)

# view the predictions
# print(predictions_df)
binary_predictions <- ifelse(predictions_df$predict >= 0.5, 1, 0)
```

```r
#h2o.shutdown(prompt = FALSE)

# the best model is the StackedEnsemble_AllModels_1_AutoML_1_20240630_120957, but the performance metri
```

```r
# Calculate accuracy, sensitivity, specificity, precision, recall, and F1
test_predictions <- test
test_predictions$predictions <- binary_predictions
```

```r
# calculate for accuracy
test_predictions <- test_predictions %>%
  mutate(cm = case_when(
    win == 1 & predictions == 1 ~ "TP",
    win == 0 & predictions == 1 ~ "FP",
    win == 1 & predictions == 0 ~ "FN",
    win == 0 & predictions == 0 ~ "TN"

  )
)

accuracy <- (sum(test_predictions$cm == "TP") + sum(test_predictions$cm == "TN"))/nrow(test_predictions)
print(paste("Accuracy: ", accuracy))
```

```
## [1] "Accuracy:  0.794037940379404"
```

```r
sensitivity <- (sum(test_predictions$cm == "TP")) / ((sum(test_predictions$cm == "TP") + sum(test_predic
print(paste("Sensitivity: ", sensitivity))
```

```
## [1] "Sensitivity:  0.775132275132275"
```

```r
specificity <- (sum(test_predictions$cm == "TN")) / (sum(test_predictions$cm == "TN") + sum(test_predict
print(paste("Specificity: ", specificity))
```

```
## [1] "Specificity:  0.813888888888889"
```

```r
precision = sum(test_predictions$cm == "TP") / (sum(test_predictions$cm == "TP") + sum(test_predictions$
print(paste("Precision: ", precision))
```

```
## [1] "Precision:  0.813888888888889"
```

```r
recall <- sum(test_predictions$cm == "TP") / (sum(test_predictions$cm == "TP") + sum(test_predictions$cm
print(paste("Recall: ", recall))
```

```
## [1] "Recall:  0.775132275132275"
```

```r
F1 = 2*precision*recall / (precision + recall)
print(paste("F1: ", F1))
```

```
## [1] "F1:  0.794037940379404"
```

```r
# check for nba rankings after making predictions
test_predictions <- test_predictions %>%
  select(-win, -cm) %>%
  rename(win = predictions)

# append training and testing set
check <- rbind(train, test_predictions)
```

```r
# check nba rankings

check_rankings_eastern <- check %>%
  filter(west_conference == 0) %>%
  group_by(off_team) %>%
  summarise(total_wins = sum(win))%>%
  arrange(desc(total_wins))


check_rankings_eastern <- check_rankings_eastern %>%
  mutate(
    actual_ranking = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15
    )
  )

print(check_rankings_eastern)
```

```
## # A tibble: 15 x 3
##    off_team total_wins actual_ranking
##    <chr>         <dbl>          <dbl>
##  1 BOS              60              1
##  2 MIL              53              3
##  3 IND              50              6
##  4 CLE              48              4
##  5 NYK              48              2
##  6 MIA              47              8
##  7 ORL              47              5
##  8 PHI              45              7
##  9 CHI              39              9
## 10 ATL              33             10
```

```
## 11 BKN              27             11
## 12 TOR              26             12
## 13 CHA              19             13
## 14 DET              13             15
## 15 WAS              13             14
```

```
# for the eastern conference, seeds 2-8 are off.
check_rankings_western <- check %>%
  filter(west_conference == 1) %>%
  group_by(off_team) %>%
  summarise(total_wins = sum(win)) %>%
  arrange(desc(total_wins))

check_rankings_western  <- check_rankings_western %>%
  mutate(
    actual_ranking = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15
    )
  )

print(check_rankings_western)
```

```
## # A tibble: 15 x 3
##    off_team total_wins actual_ranking
##    <chr>         <dbl>          <dbl>
##  1 OKC              60              1
##  2 DEN              57              2
##  3 MIN              56              3
##  4 LAC              52              4
##  5 PHX              52              6
##  6 DAL              51              5
##  7 NOP              48              7
##  8 GSW              47             10
##  9 LAL              47              8
## 10 SAC              47              9
## 11 HOU              37             11
## 12 UTA              32             12
## 13 SAS              21             14
## 14 MEM              19             13
## 15 POR              18             15
```

```
# for the western conference, seeds aren't too far skewed from the normal except for the warriors placi
# for both the eastern and western conference, the model struggles to classify wins for seeds 2-8. For
```

```
# model #2: retrain data, but each nbagameid is unique. Ops stats and def stats will be in the same row
reg_szn2023_2 <- team_data %>%
  filter(
    season == 2023 & gametype == 2
  ) %>%
  mutate(
    eFG = (fg2made + 1.5 * fg3made) / fgattempted,
    TORatio = turnovers / possessions,
    reb_percentage = reboffensive / reboundchance,
    FTRate = ftmade / fgattempted,
    west_conference = case_when(
      off_team %in% c("OKC", "DEN", "MIN", "LAC", "DAL", "PHX", "LAL", "NOP", "SAC", "GSW", "HOU", "UTA
      TRUE ~ 0
    )
  ) %>%
  group_by(nbagameid) %>%
  mutate(win = ifelse(points == max(points), 1, 0)) %>%
  ungroup() %>%
  select(nbagameid, off_team, eFG, TORatio, reb_percentage, FTRate, off_home, points, west_conference, w

# add end of season rankings to the model
reg_szn2023_2 <- reg_szn2023_2 %>%
  mutate(
    rank = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15,
      TRUE ~ NA_real_))
```

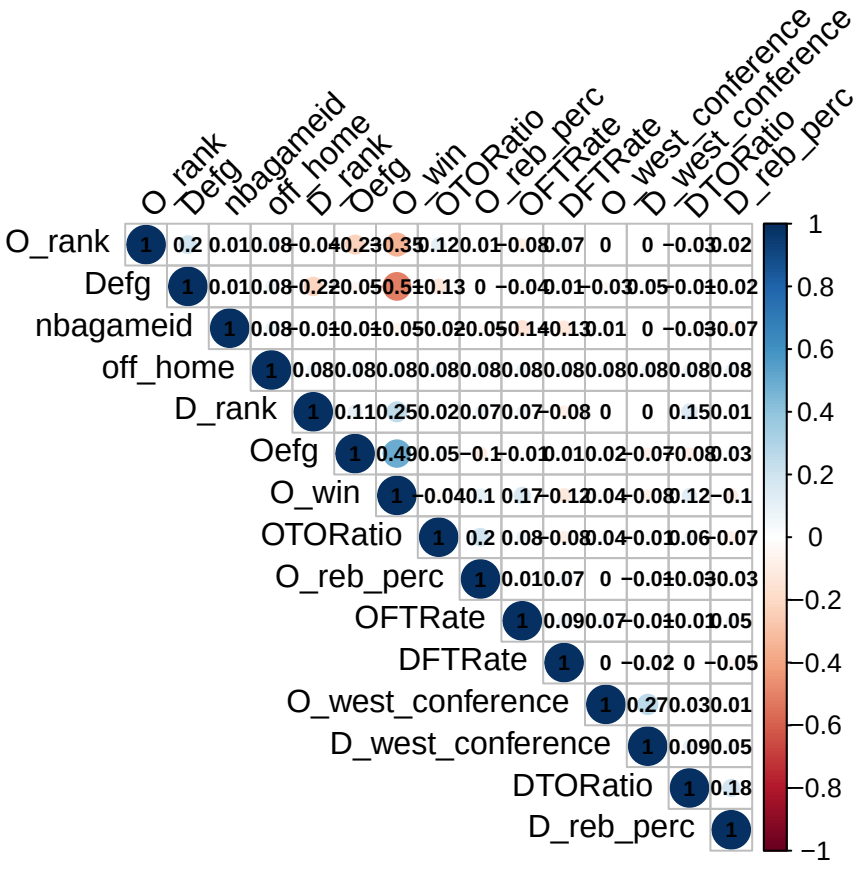
```
# split data by off_home = 1 and off_home = 0
off_home <- reg_szn2023_2 %>%
  filter(off_home == 1)

off_away <- reg_szn2023_2 %>%
  filter(off_home == 0)

# rename off_home and off_away columns to reflect if the team is on offense or defense
off_home <- off_home %>%
```

```
  rename(Oefg = eFG,
         OTORatio = TORatio,
         O_reb_perc = reb_percentage,
         OFTRate = FTRate,
         O_points = points,
         O_west_conference = west_conference,
         O_rank = rank,
         O_win = win
  )

off_away <- off_away %>%
  rename(
    def_team = off_team,
    Defg = eFG,
    DTORatio = TORatio,
    D_reb_perc = reb_percentage,
    DFTRate = FTRate,
    D_west_conference = west_conference,
    D_rank = rank,
    D_points = points
  ) %>%
  select(-off_home, -win)

# merge dataframes
uniqueid_reg_szn2023 <- off_home %>%
  full_join(off_away, by = c("nbagameid")) %>%
  select(-D_points, -O_points)
```

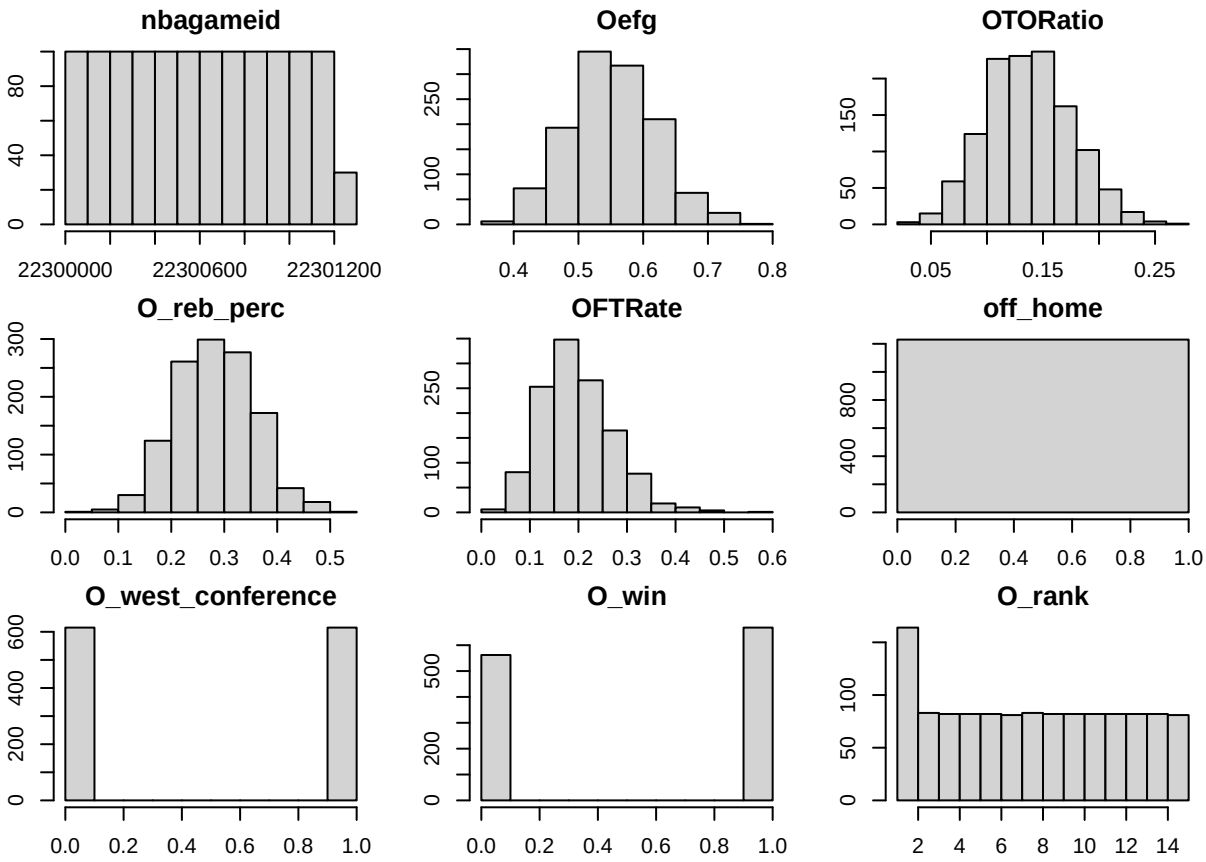
```
corr2 <- uniqueid_reg_szn2023 %>%
  select(c(-off_team, -def_team))
cor_matrix_2 <- cor(corr2)
```

```
## Warning in stats::cor(x, ...): the standard deviation is zero
```

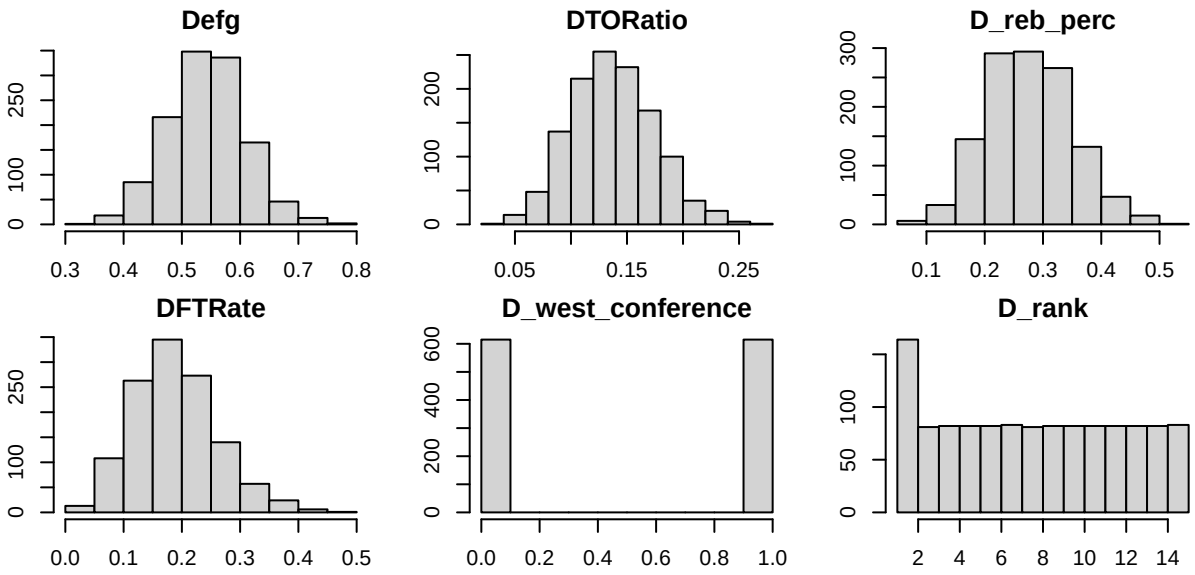
```
cor_matrix_2[is.na(cor_matrix_2)] <- mean(cor_matrix_2, na.rm = TRUE)
clust <- hclust(as.dist(1 - cor_matrix_2), method = "ward.D2")

corrplot(cor_matrix_2, method = "circle", type = "upper", order = "hclust",
         addCoef.col = "black", tl.col = "black", tl.srt = 45,
         number.cex = 0.7)
```




```
# view distribution of data
numeric_columns <- sapply(uniqueid_reg_szn2023, is.numeric)

# set margins
par(mar = c(2,2,2,2))
par(mfrow = c(3, 3))
for (col in names(uniqueid_reg_szn2023[numeric_columns])) {
  hist(uniqueid_reg_szn2023[[col]], main = col, xlab = col)
}
```




```
# scale data
# scale data so ranking doesn't skew the results
uniqueid_reg_szn2023$O_rank <- scales::rescale(uniqueid_reg_szn2023$O_rank)
uniqueid_reg_szn2023$D_rank <- scales::rescale(uniqueid_reg_szn2023$D_rank)
```




```
# split the data into training and testing
set.seed(2581)
split_index <- sample(1:nrow(uniqueid_reg_szn2023), 0.7*nrow(uniqueid_reg_szn2023))
train <- uniqueid_reg_szn2023[split_index, ]
test <- uniqueid_reg_szn2023[-split_index, ]
```

```
# use automl to retrieve a baseline model

#library(h2o)
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         27 seconds 554 milliseconds
##     H2O cluster timezone:       America/Los_Angeles
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.44.0.3
##     H2O cluster version age:    1 year and 25 days
##     H2O cluster name:           H2O_started_from_R_linanguyen_cfp368
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.22 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
```

```
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.0 (2021-05-18)

## Warning in h2o.clusterInfo():
## Your H2O cluster version is (1 year and 25 days) old. There may be a newer version available.
## Please download and install the latest version from: https://h2o-release.s3.amazonaws.com/h2o/latest
```

```
# convert to h2o objects
train_h2o <- as.h2o(train)
```

```
##   |                                                                      |
```

```
test_h2o <- as.h2o(test)
```

```
##   |                                                                      |
```

```
target <- "O_win"
features <- setdiff(names(train), target)

# train h2o model
model <- h2o.automl(
  x = features,
  y = target,
  training_frame = train_h2o,
  max_models = 10,
  seed = 1
)
```

```
##   |                                                                      |
## 20:08:54.298: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:54.298: _response param, We have detected that your response column has only 2 unique values (
## 20:08:54.990: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:54.990: _response param, We have detected that your response column has only 2 unique values (
## 20:08:55.245: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:55.245: _response param, We have detected that your response column has only 2 unique values (
## 20:08:56.178: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:56.178: _response param, We have detected that your response column has only 2 unique values (
## 20:08:56.697: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:56.697: _response param, We have detected that your response column has only 2 unique values (
## 20:08:57.293: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:57.293: _response param, We have detected that your response column has only 2 unique values (
## 20:08:57.887: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:57.887: _response param, We have detected that your response column has only 2 unique values (
## 20:08:58.461: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:58.461: _response param, We have detected that your response column has only 2 unique values (
## 20:08:59.81: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:59.81: _response param, We have detected that your response column has only 2 unique values (0,
## 20:08:59.366: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:08:59.366: _response param, We have detected that your response column has only 2 unique values (
```

```
## 20:08:59.891: _train param, Dropping unused columns: [off_team, def_team, off_home]
## 20:08:59.891: _response param, We have detected that your response column has only 2 unique values (
## 20:09:00.402: _train param, Dropping unused columns: [off_team, def_team, off_home]
## 20:09:00.402: _response param, We have detected that your response column has only 2 unique values (
```

```
best_model <- model@leader
lb <- model@leaderboard
print(lb, n = nrow(lb))
```

```
##                                                 model_id      rmse        mse
## 1     StackedEnsemble_AllModels_1_AutoML_2_20250114_200854 0.2731242 0.07459683
## 2  StackedEnsemble_BestOfFamily_1_AutoML_2_20250114_200854 0.2738444 0.07499075
## 3                            GBM_2_AutoML_2_20250114_200854 0.2908901 0.08461705
## 4                            GBM_3_AutoML_2_20250114_200854 0.2941771 0.08654016
## 5                            GBM_4_AutoML_2_20250114_200854 0.2945704 0.08677174
## 6                            GLM_1_AutoML_2_20250114_200854 0.2991885 0.08951374
## 7                         XGBoost_3_AutoML_2_20250114_200854 0.2995969 0.08975828
## 8                            GBM_1_AutoML_2_20250114_200854 0.3041893 0.09253110
## 9                            XRT_1_AutoML_2_20250114_200854 0.3103770 0.09633389
## 10                        XGBoost_1_AutoML_2_20250114_200854 0.3112989 0.09690702
## 11                        XGBoost_2_AutoML_2_20250114_200854 0.3115305 0.09705126
## 12                           DRF_1_AutoML_2_20250114_200854 0.3181634 0.10122795
##          mae     rmsle mean_residual_deviance
## 1  0.2153762 0.1955456             0.07459683
## 2  0.2162349 0.1957539             0.07499075
## 3  0.2163490 0.2067930             0.08461705
## 4  0.2205485 0.2076380             0.08654016
## 5  0.2259052 0.2097842             0.08677174
## 6  0.2569554 0.2198903             0.08951374
## 7  0.2257567 0.2150794             0.08975828
## 8  0.2480004 0.2140595             0.09253110
## 9  0.2359388 0.2202187             0.09633389
## 10 0.2357639 0.2227623             0.09690702
## 11 0.2311001 0.2235649             0.09705126
## 12 0.2457356 0.2247447             0.10122795
##
## [12 rows x 6 columns]
```

```
# predict the test set
predictions <- h2o.predict(best_model, test_h2o)
```

```
##   |                                                                              |
```

```
# convert predictions to a dataframe
predictions_df <- as.data.frame(predictions)

# view the predictions
# print(predictions_df)
binary_predictions <- ifelse(predictions_df$predict >= 0.5, 1, 0)

h2o.shutdown(prompt = FALSE)

# The best model is StackedEnsemble_AllModels_1_AutoML_1_20240630_123823, and performance has improved
```

```r
# Calculate accuracy, sensitivity, specificity, precision, recall, and F1
test_predictions <- test
test_predictions$predictions <- binary_predictions
```

```r
# calculate for accuracy
test_predictions <- test_predictions %>%
  mutate(cm = case_when(
    O_win == 1 & predictions == 1 ~ "TP",
    O_win == 0 & predictions == 1 ~ "FP",
    O_win == 1 & predictions == 0 ~ "FN",
    O_win == 0 & predictions == 0 ~ "TN"

  )
)

accuracy <- (sum(test_predictions$cm == "TP") + sum(test_predictions$cm == "TN"))/nrow(test_predictions)
print(paste("Accuracy: ", accuracy))
```

```
## [1] "Accuracy:  0.943089430894309"
```

```r
sensitivity <- (sum(test_predictions$cm == "TP")) / ((sum(test_predictions$cm == "TP") + sum(test_predic
print(paste("Sensitivity: ", sensitivity))
```

```
## [1] "Sensitivity:  0.955882352941177"
```

```r
specificity <- (sum(test_predictions$cm == "TN")) / (sum(test_predictions$cm == "TN") + sum(test_predict
print(paste("Specificity: ", specificity))
```

```
## [1] "Specificity:  0.927272727272727"
```

```r
precision = sum(test_predictions$cm == "TP") / (sum(test_predictions$cm == "TP") + sum(test_predictions$
print(paste("Precision: ", precision))
```

```
## [1] "Precision:  0.942028985507246"
```

```r
recall <- sum(test_predictions$cm == "TP") / (sum(test_predictions$cm == "TP") + sum(test_predictions$cm
print(paste("Recall: ", recall))
```

```
## [1] "Recall:  0.955882352941177"
```

```r
F1 = 2*precision*recall / (precision + recall)
print(paste("F1: ", F1))
```

```
## [1] "F1:  0.948905109489051"
```

```r
# the performance of the model is a lot better now that there is an opposing team in the dataset
```

```r
# check for nba rankings after making predictions
test_predictions <- test_predictions %>%
  select(c(-O_win, -cm)) %>%
  rename(O_win = predictions)

# append training and testing set
check <- rbind(train, test_predictions)

check_rankings_eastern_O <- check %>%
  filter(O_west_conference == 0) %>%
  group_by(off_team) %>%
  summarise(total_wins = sum(O_win))%>%
  arrange(desc(total_wins))

check_rankings_eastern_D <- check %>%
  filter(D_west_conference == 0) %>%  # Filter for Eastern Conference defensive teams
  group_by(def_team) %>%  # Group by defensive team
  summarise(total_wins = sum(case_when(
    O_win == 1 ~ 0,  # If offensive team wins, count as 0
    O_win == 0 ~ 1   # If offensive team loses, count as 1
  ))) %>%  # Summarise total wins
  arrange(desc(total_wins))

# merge offense and defense stats
rankings_eastern <- merge(
  check_rankings_eastern_O,
  check_rankings_eastern_D,
  by.x = "off_team",
  by.y = "def_team", all= TRUE)

rankings_eastern <- rankings_eastern %>%
  mutate(total_wins = total_wins.x + total_wins.y) %>%
  select(off_team, total_wins) %>%
  arrange(desc(total_wins))

check_rankings_eastern <- rankings_eastern %>%
  mutate(
    actual_ranking = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15
```

```
    )
  )

print(check_rankings_eastern)
```

```
##    off_team total_wins actual_ranking
## 1       BOS         65              1
## 2       NYK         51              2
## 3       CLE         50              4
## 4       MIL         49              3
## 5       IND         48              6
## 6       PHI         47              7
## 7       ORL         45              5
## 8       MIA         44              8
## 9       CHI         38              9
## 10      ATL         36             10
## 11      BKN         32             11
## 12      TOR         24             12
## 13      CHA         20             13
## 14      WAS         17             14
## 15      DET         14             15
```

```
# Eastern rankings have significantly improved, and the rankings that are switched around have total_wi
```

```
check_rankings_western_O <- check %>%
  filter(O_west_conference == 1) %>%
  group_by(off_team) %>%
  summarise(total_wins = sum(O_win))%>%
  arrange(desc(total_wins))

check_rankings_western_D <- check %>%
  filter(D_west_conference == 1) %>%  # Filter for Eastern Conference defensive teams
  group_by(def_team) %>%  # Group by defensive team
  summarise(total_wins = sum(case_when(
    O_win == 1 ~ 0,  # If offensive team wins, count as 0
    O_win == 0 ~ 1   # If offensive team loses, count as 1
  ))) %>%  # Summarise total wins
  arrange(desc(total_wins))

# merge offense and defense stats
rankings_western <- merge(
  check_rankings_western_O,
  check_rankings_western_D,
  by.x = "off_team",
  by.y = "def_team", all= TRUE)

rankings_western<- rankings_western %>%
  mutate(total_wins = total_wins.x + total_wins.y) %>%
  select(off_team, total_wins) %>%
  arrange(desc(total_wins))

check_rankings_western_O <- check %>%
  filter(O_west_conference == 1) %>%
```

```
  group_by(off_team) %>%
  summarise(total_wins = sum(O_win))%>%
  arrange(desc(total_wins))

check_rankings_western_D <- check %>%
  filter(D_west_conference == 1) %>%  # Filter for Eastern Conference defensive teams
  group_by(def_team) %>%  # Group by defensive team
  summarise(total_wins = sum(case_when(
    O_win == 1 ~ 0,  # If offensive team wins, count as 0
    O_win == 0 ~ 1   # If offensive team loses, count as 1
  ))) %>%  # Summarise total wins
  arrange(desc(total_wins))

# merge offense and defense stats
rankings_western <- merge(
  check_rankings_western_O,
  check_rankings_western_D,
  by.x = "off_team",
  by.y = "def_team", all= TRUE)

rankings_western<- rankings_western %>%
  mutate(total_wins = total_wins.x + total_wins.y) %>%
  select(off_team, total_wins) %>%
  arrange(desc(total_wins))

check_rankings_western  <- rankings_western %>%
  mutate(
    actual_ranking = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15
    )
  )

print(check_rankings_western)
```

```
##    off_team total_wins actual_ranking
## 1       DEN         58              2
## 2       MIN         58              3
## 3       OKC         57              1
## 4       LAC         51              4
```

```
## 5       DAL         49              5
## 6       LAL         49              8
## 7       PHX         49              6
## 8       NOP         48              7
## 9       GSW         46             10
## 10      SAC         44              9
## 11      HOU         41             11
## 12      UTA         33             12
## 13      MEM         27             13
## 14      SAS         21             14
## 15      POR         19             15
```

```
# The western rankings are a lot better, and rankings that are switched around have total_wins that are

# the second model performed better than the first model. We will now add roster data to the model. To
```

```
# Part 2: Players impact leading to the playoffs
# tie the results from part 1 to see which players per team and what features impact the most on the te
# filter by teams that make the play offs into player_data (gametype = 2)

# using the top 16 teams from the last model, predict probability of wins for each of the play off seri

# for player_data, we are going to sort players by team and the average number of seconds in order. Bas

player_2023 <- player_data %>%
  filter(season == 2023 & gametype == 2) %>%
  group_by(nbapersonid, team) %>%
  summarise(avg_seconds = sum(seconds) / n(), .groups = 'drop') %>%
  select(nbapersonid, team, avg_seconds)

ranked_players_2023 <- player_2023 %>%
  group_by(team) %>%
  arrange(desc(avg_seconds)) %>%
  mutate(rank = row_number())
head(ranked_players_2023)
```

```
## # A tibble: 6 x 4
## # Groups:   team [5]
##   nbapersonid team  avg_seconds  rank
##         <dbl> <chr>       <dbl> <int>
## 1      201942 CHI         2183.     1
## 2     1627734 SAC         2139.     1
## 3     1629632 CHI         2104.     2
## 4     1628969 BKN         2085.     1
## 5     1631094 ORL         2044.     1
## 6      201142 PHX         2038.     1
```

```
roster_per_game <- player_data %>%
  filter(season == 2023 & gametype == 2) %>%
  select(nbagameid, team, nbapersonid, seconds)
head(roster_per_game)
```

```
## # A tibble: 6 x 4
```

```
##   nbagameid team  nbapersonid seconds
##       <dbl> <chr>       <dbl>   <dbl>
## 1  22300184 MIA       1626196    1566
## 2  22300184 MIA       1630181       0
## 3  22300184 MIA       1631107       0
## 4  22300184 MIA        200768       0
## 5  22300184 MIA       1631288       0
## 6  22300184 MIA       1630696    1610
```

```
roster_per_game1 <- roster_per_game %>%
  # merge with top 15 player to get ranks of each player on the roster
  full_join(ranked_players_2023, by = c("nbapersonid", "team")) %>%
  select(nbagameid, nbapersonid, team, seconds, rank)
head(roster_per_game1)
```

```
## # A tibble: 6 x 5
##   nbagameid nbapersonid team  seconds  rank
##       <dbl>       <dbl> <chr>   <dbl> <int>
## 1  22300184     1626196 MIA      1566    10
## 2  22300184     1630181 MIA         0    19
## 3  22300184     1631107 MIA         0    12
## 4  22300184      200768 MIA         0     5
## 5  22300184     1631288 MIA         0    17
## 6  22300184     1630696 MIA      1610    18
```

```
roster_per_game_wide<- roster_per_game1 %>%
  pivot_wider(
    names_from = rank,
    values_from = seconds,
    names_prefix = "rank_"
  ) %>%
  select(-c(nbapersonid))
head(roster_per_game_wide)
```

```
## # A tibble: 6 x 37
##   nbagameid team  rank_10 rank_19 rank_12 rank_5 rank_17 rank_18 rank_2 rank_1
##       <dbl> <chr>   <dbl>   <dbl>   <dbl>  <dbl>   <dbl>   <dbl>  <dbl>  <dbl>
## 1  22300184 MIA      1566      NA      NA     NA      NA      NA     NA     NA
## 2  22300184 MIA        NA       0      NA     NA      NA      NA     NA     NA
## 3  22300184 MIA        NA      NA       0     NA      NA      NA     NA     NA
## 4  22300184 MIA        NA      NA      NA      0      NA      NA     NA     NA
## 5  22300184 MIA        NA      NA      NA     NA       0      NA     NA     NA
## 6  22300184 MIA        NA      NA      NA     NA      NA    1610     NA     NA
## # ... with 27 more variables: rank_8 <dbl>, rank_7 <dbl>, rank_20 <dbl>,
## #   rank_4 <dbl>, rank_9 <dbl>, rank_16 <dbl>, rank_15 <dbl>, rank_11 <dbl>,
## #   rank_6 <dbl>, rank_3 <dbl>, rank_13 <dbl>, rank_14 <dbl>, rank_30 <dbl>,
## #   rank_28 <dbl>, rank_24 <dbl>, rank_31 <dbl>, rank_26 <dbl>, rank_22 <dbl>,
## #   rank_21 <dbl>, rank_23 <dbl>, rank_25 <dbl>, rank_32 <dbl>, rank_27 <dbl>,
## #   rank_34 <dbl>, rank_29 <dbl>, rank_33 <dbl>, rank_35 <dbl>
```

```
# roster per game but with unique nbagameid
roster_per_game_wide_unique <- roster_per_game_wide %>%
```

```
  group_by(nbagameid, team) %>%
  summarise(
    across(starts_with("rank_"), sum, na.rm = TRUE),
    .groups = 'drop'
  )

head(roster_per_game_wide_unique)
```

```
## # A tibble: 6 x 37
##   nbagameid team  rank_10 rank_19 rank_12 rank_5 rank_17 rank_18 rank_2 rank_1
##       <dbl> <chr>   <dbl>   <dbl>   <dbl>  <dbl>   <dbl>   <dbl>  <dbl>  <dbl>
## 1  22300001 CLE         0       0       0  1918.       0       0  1694    1265
## 2  22300001 IND         0       0       0  2244.       0       0  2056       0
## 3  22300002 MIL       716       0     786   993        0       0  2142    2159
## 4  22300002 NYK      2009       0       0     0       0       0  2154.   2495
## 5  22300003 MIA      1530.      0       0  1531       0       0  1293    1797
## 6  22300003 WAS         0     396     742  1174       0       0  1858    1700
## # ... with 27 more variables: rank_8 <dbl>, rank_7 <dbl>, rank_20 <dbl>,
## #   rank_4 <dbl>, rank_9 <dbl>, rank_16 <dbl>, rank_15 <dbl>, rank_11 <dbl>,
## #   rank_6 <dbl>, rank_3 <dbl>, rank_13 <dbl>, rank_14 <dbl>, rank_30 <dbl>,
## #   rank_28 <dbl>, rank_24 <dbl>, rank_31 <dbl>, rank_26 <dbl>, rank_22 <dbl>,
## #   rank_21 <dbl>, rank_23 <dbl>, rank_25 <dbl>, rank_32 <dbl>, rank_27 <dbl>,
## #   rank_34 <dbl>, rank_29 <dbl>, rank_33 <dbl>, rank_35 <dbl>
```

```
# view the number of times 0 appears in each column to determine the cut off of ranks
zero_counts <- colSums(roster_per_game_wide_unique == 0.0, na.rm = TRUE)
sorted_zero_counts <- sort(zero_counts, decreasing = TRUE)
print(sorted_zero_counts)
```

```
##   rank_34   rank_35   rank_33   rank_31   rank_32   rank_29   rank_30   rank_27
##      2460      2460      2457      2454      2454      2446      2438      2424
##   rank_28   rank_25   rank_26   rank_23   rank_24   rank_22   rank_21   rank_20
##      2417      2396      2380      2365      2350      2335      2244      2167
##   rank_18   rank_19   rank_17   rank_16   rank_15   rank_14   rank_13   rank_12
##      2128      2093      1882      1733      1686      1575      1445      1286
##   rank_11   rank_10    rank_9    rank_8    rank_6    rank_5    rank_7    rank_3
##      1097       996       912       739       698       695       654       546
##    rank_4    rank_2    rank_1 nbagameid      team
##       515       389       385         0         0
```

```
# rank_1 to rank_11 will be kept, everything else will be dropped because the amount of missing data is
```

```
roster_clean <- select(roster_per_game_wide_unique, nbagameid, team, rank_1, rank_2, rank_3, rank_4, ra
```

```
# merge roster_clean with uniqueid_reg_szn2023
# we need to convert each nbagameid to a unique game id and assign each team to offense or defense

key <- team_data %>%
  filter(gametype == 2, season == 2023) %>%
  select(c(nbagameid, off_team, off_home)) %>%
  rename(team = off_team)
```

```
roster_clean <- roster_clean %>%
  left_join(key, by = c("nbagameid", "team")) %>%
  select(c(nbagameid, team, rank_1, rank_2, rank_3, rank_4, rank_5, rank_6, rank_7, rank_8, rank_9, rank

roster_clean_O <- roster_clean %>%
  filter(off_home == 1) %>%
  rename(
    off_team = team,
    O_rank_1 = rank_1,
    O_rank_2 = rank_2,
    O_rank_3 = rank_3,
    O_rank_4 = rank_4,
    O_rank_5 = rank_5,
    O_rank_6 = rank_6,
    O_rank_7 = rank_7,
    O_rank_8 = rank_8,
    O_rank_9 = rank_9,
    O_rank_10 = rank_10,
    O_rank_11 = rank_11
  )


roster_clean_D <- roster_clean %>%
  filter(off_home == 0) %>%
  rename(
    def_team = team,
    D_rank_1 = rank_1,
    D_rank_2 = rank_2,
    D_rank_3 = rank_3,
    D_rank_4 = rank_4,
    D_rank_5 = rank_5,
    D_rank_6 = rank_6,
    D_rank_7 = rank_7,
    D_rank_8 = rank_8,
    D_rank_9 = rank_9,
    D_rank_10 = rank_10,
    D_rank_11 = rank_11
  ) %>%
  select(-off_home)

# merge two datasets
roster_clean_uniqueid <- roster_clean_O %>%
  left_join(roster_clean_D, by = c("nbagameid"))
print(roster_clean_uniqueid)
```

```
## # A tibble: 1,230 x 26
##    nbagameid off_team O_rank_1 O_rank_2 O_rank_3 O_ran~1 O_ran~2 O_ran~3 O_ran~4
##        <dbl> <chr>       <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1  22300001 IND             0    2056     1836    1812.   2244.   1648.   1227
##  2  22300002 MIL          2159    2142     2127    1337     993    1438    1253
##  3  22300003 MIA          1797    1293        0    1878.   1531    1772       0
##  4  22300004 CHI          2005    2124.    2139.      0    1577       0     741.
##  5  22300005 OKC             0    1913.    1809    2116    1831    2179    1571.
```

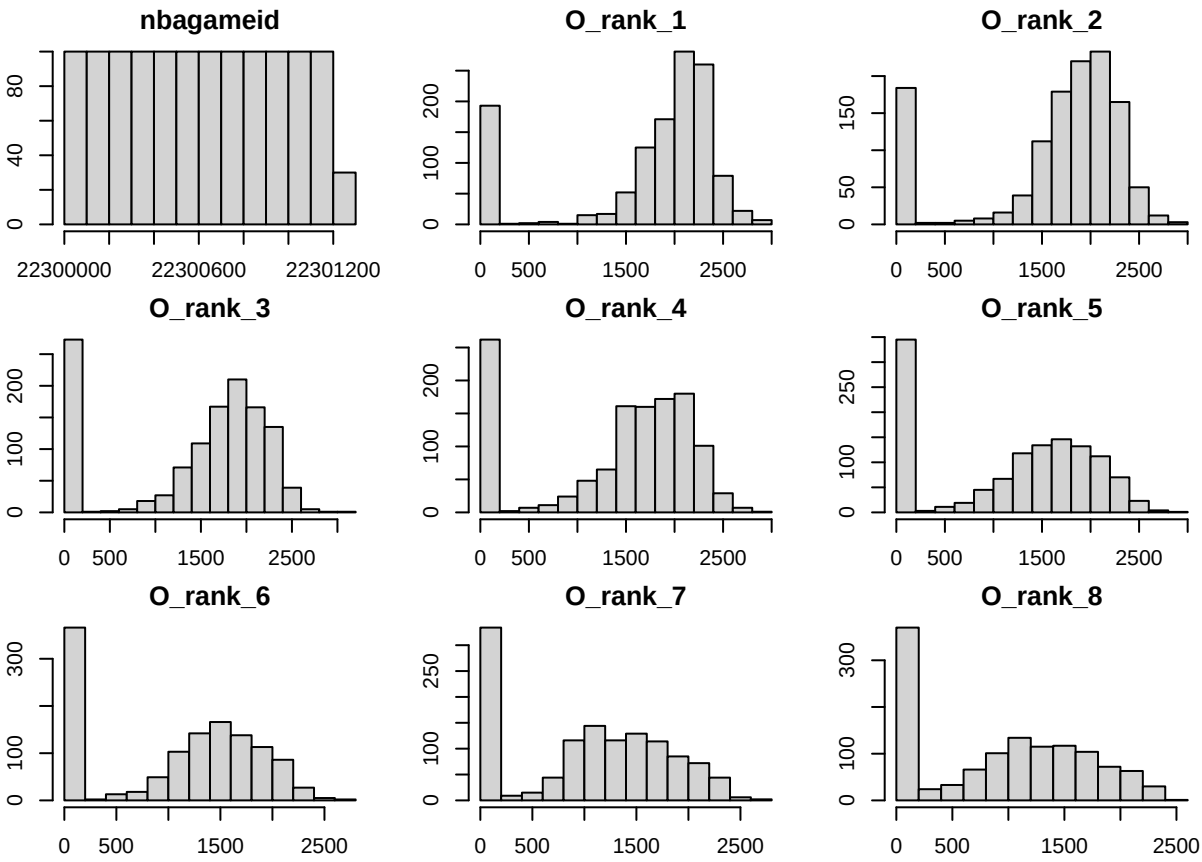
```
##  6  22300006 DEN          2138    2064.    2214    2114    2349    1191     878
##  7  22300007 POR             0    2619     2173       0    1887     558       0
##  8  22300008 DET             0       0     2139.   1361    2310    1940       0
##  9  22300009 WAS          1870    1741     1841.    981    1801    2025    1178
## 10  22300010 BOS          1993    1604     1764    1965.   1829.   1478       0
## # ... with 1,220 more rows, 17 more variables: O_rank_8 <dbl>, O_rank_9 <dbl>,
## #   O_rank_10 <dbl>, O_rank_11 <dbl>, off_home <dbl>, def_team <chr>,
## #   D_rank_1 <dbl>, D_rank_2 <dbl>, D_rank_3 <dbl>, D_rank_4 <dbl>,
## #   D_rank_5 <dbl>, D_rank_6 <dbl>, D_rank_7 <dbl>, D_rank_8 <dbl>,
## #   D_rank_9 <dbl>, D_rank_10 <dbl>, D_rank_11 <dbl>, and abbreviated variable
## #   names 1: O_rank_4, 2: O_rank_5, 3: O_rank_6, 4: O_rank_7
```

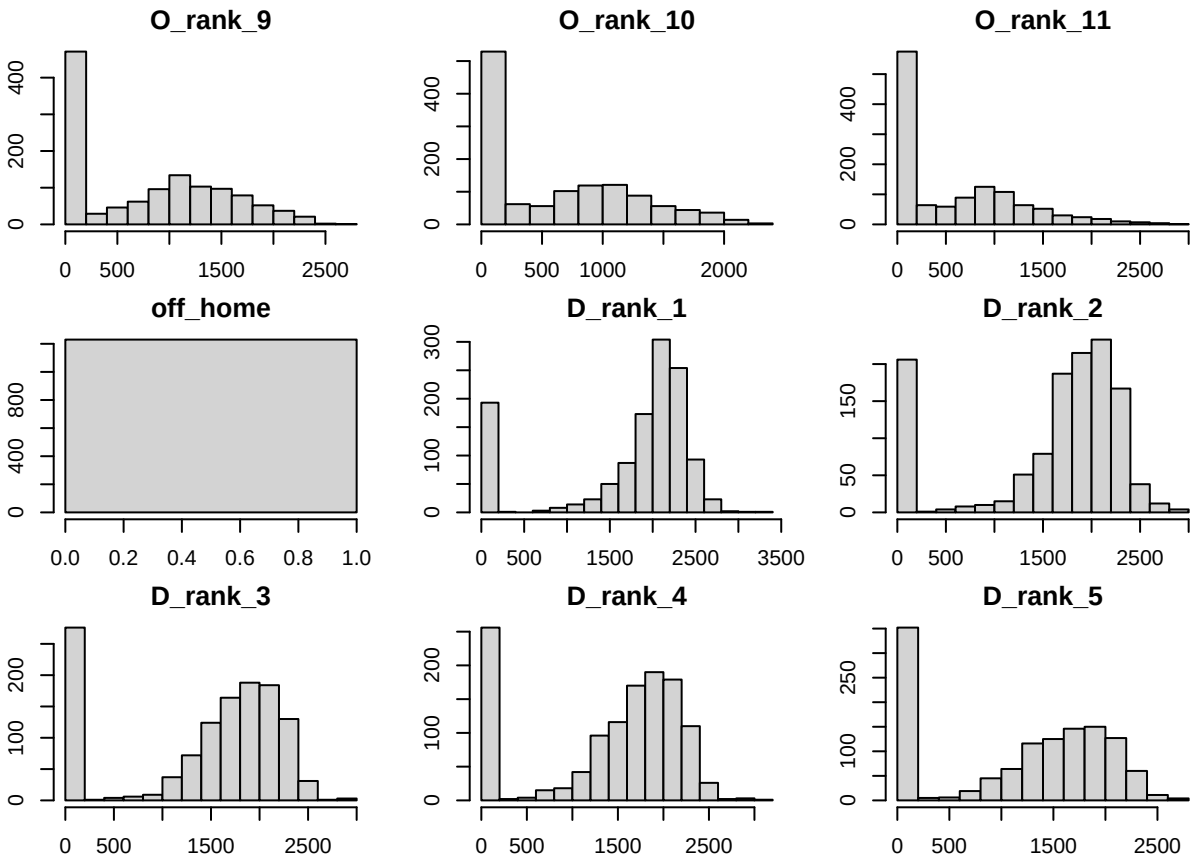
```
# merge uniqueid_reg_szn2023 vs. roster_clean_uniqueid
model3_clean_df <- roster_clean_uniqueid %>%
  left_join(uniqueid_reg_szn2023, by = c("nbagameid", "off_team", "off_home", "def_team"))
head(model3_clean_df)
```

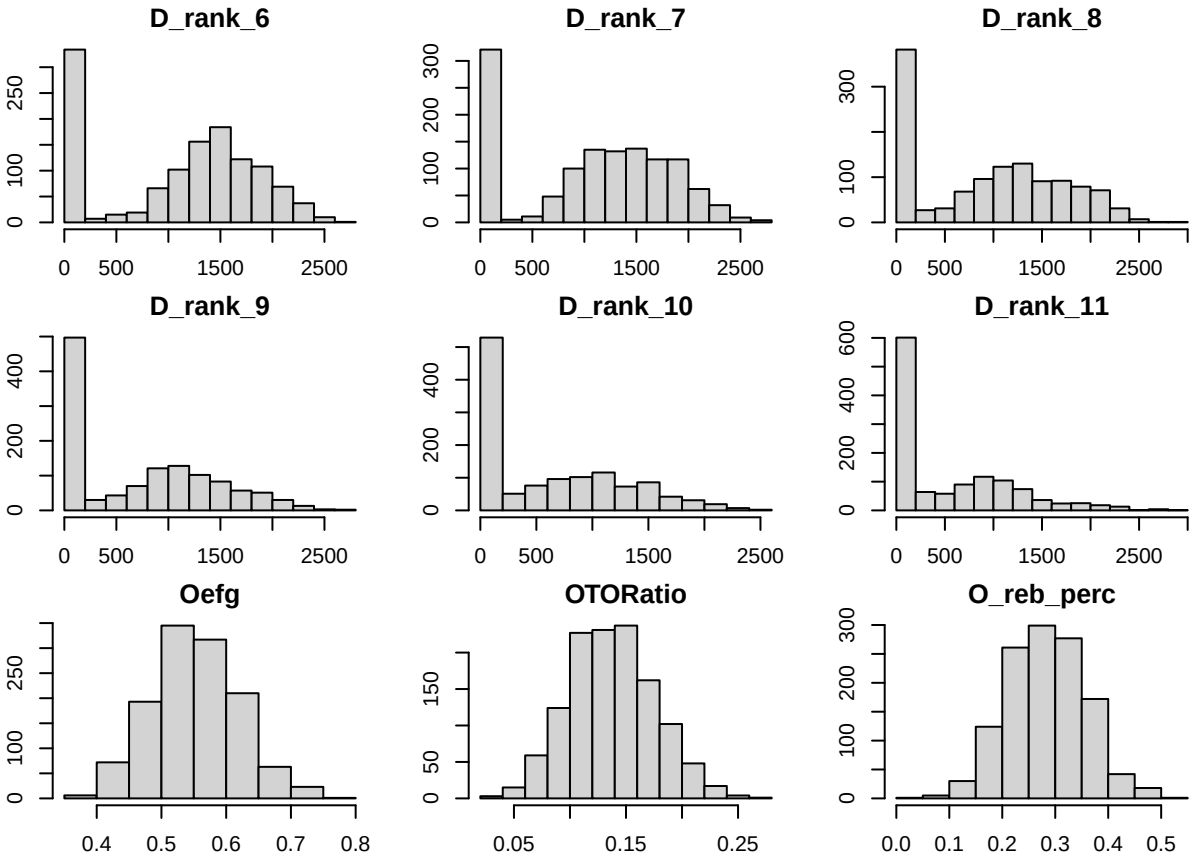
```
## # A tibble: 6 x 39
##   nbagameid off_team O_rank_1 O_rank_2 O_rank_3 O_rank_4 O_ran~1 O_ran~2 O_ran~3
##       <dbl> <chr>       <dbl>    <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>
## 1  22300001 IND             0    2056     1836     1812.   2244.   1648.   1227
## 2  22300002 MIL          2159    2142     2127     1337     993    1438    1253
## 3  22300003 MIA          1797    1293        0     1878.   1531    1772       0
## 4  22300004 CHI          2005    2124.    2139.       0    1577       0     741.
## 5  22300005 OKC             0    1913.    1809     2116    1831    2179    1571.
## 6  22300006 DEN          2138    2064.    2214     2114    2349    1191     878
## # ... with 30 more variables: O_rank_8 <dbl>, O_rank_9 <dbl>, O_rank_10 <dbl>,
## #   O_rank_11 <dbl>, off_home <dbl>, def_team <chr>, D_rank_1 <dbl>,
## #   D_rank_2 <dbl>, D_rank_3 <dbl>, D_rank_4 <dbl>, D_rank_5 <dbl>,
## #   D_rank_6 <dbl>, D_rank_7 <dbl>, D_rank_8 <dbl>, D_rank_9 <dbl>,
## #   D_rank_10 <dbl>, D_rank_11 <dbl>, Oefg <dbl>, OTORatio <dbl>,
## #   O_reb_perc <dbl>, OFTRate <dbl>, O_west_conference <dbl>, O_win <dbl>,
## #   O_rank <dbl>, Defg <dbl>, DTORatio <dbl>, D_reb_perc <dbl>, ...
```

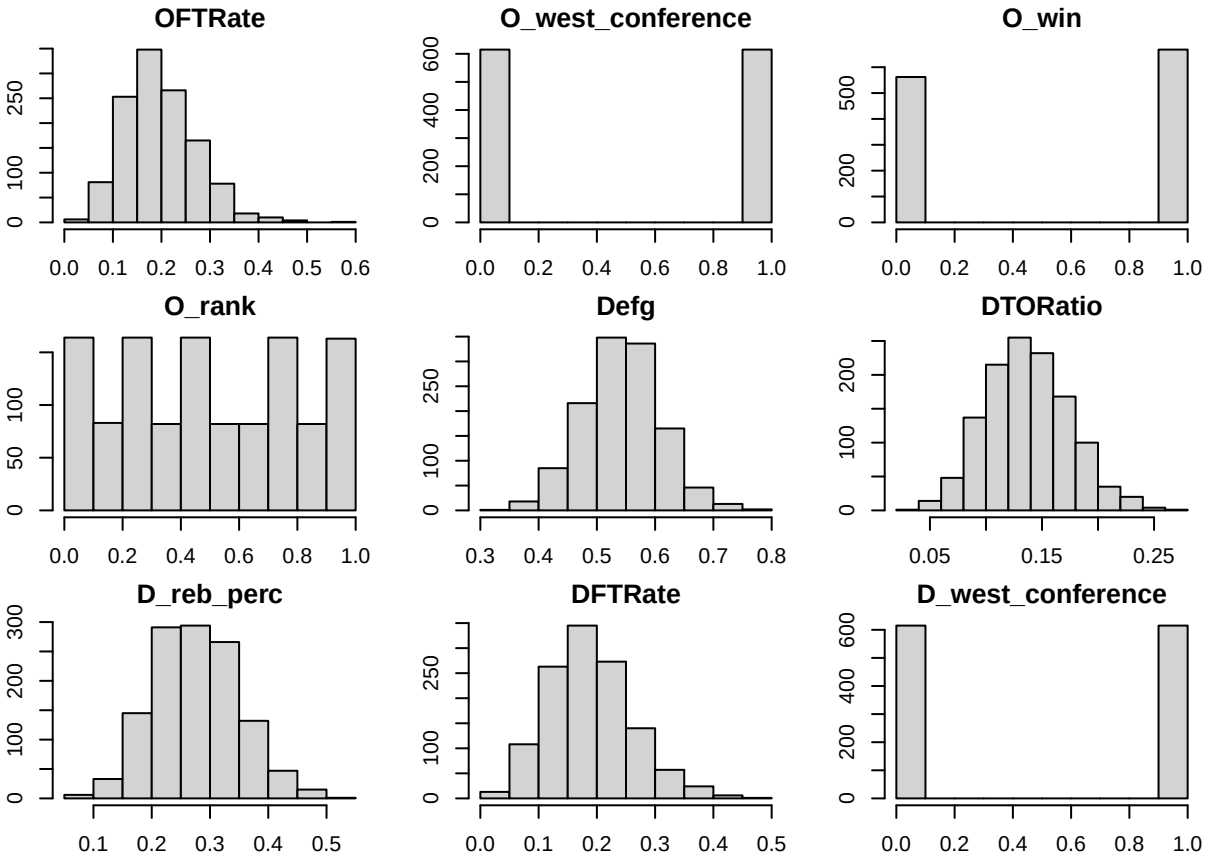
```
# this dataset merges the dataset from model 2, with data from offense and defense per game, as well as
```

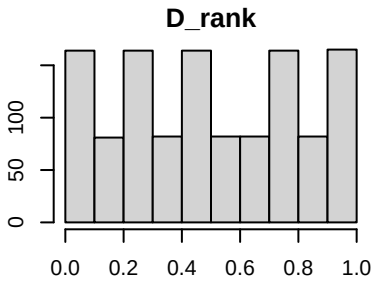
```
# view distribution of data
numeric_columns <- sapply(model3_clean_df, is.numeric)

par(mar = c(2,2,2,2))
par(mfrow = c(3, 3))
for (col in names(model3_clean_df[numeric_columns])) {
  hist(model3_clean_df[[col]], main = col, xlab = col)
}
```


nbagameid
0
40
80
22300000
22300600
22301200
O_rank_1
0
100
200
0
500
1500
2500
O_rank_2
0
50
150
0
500
1500
2500
O_rank_3
0
100
200
0
500
1500
2500
O_rank_4
0
100
200
0
500
1500
2500
O_rank_5
0
100
250
0
500
1500
2500
O_rank_6
0
100
300
0
500
1500
2500
O_rank_7
0
100
250
0
500
1500
2500
O_rank_8
0
100
300
0
500
1500
2500


O_rank_9
0
200
400
0
500
1500
2500
O_rank_10
0
200
400
0
500
1000
2000
O_rank_11
0
200
400
0
500
1500
2500
off_home
0
400
800
0.0
0.2
0.4
0.6
0.8
1.0
D_rank_1
0
100
200
300
0
500
1500
2500
3500
D_rank_2
0
50
150
0
500
1500
2500
D_rank_3
0
100
200
0
500
1500
2500
D_rank_4
0
100
200
0
500
1500
2500
D_rank_5
0
100
250
0
500
1500
2500


D_rank_6
D_rank_7
D_rank_8
D_rank_9
D_rank_10
D_rank_11
Oefg
OTORatio
O_reb_perc


OFTRate
O_west_conference
O_win
O_rank
Defg
DTORatio
D_reb_perc
DFTRate
D_west_conference




```
corr3 <- model3_clean_df[numeric_columns]

# Compute the correlation matrix
cor_matrix_3 <- cor(corr3, use = "complete.obs")
```

```
## Warning in stats::cor(x, ...): the standard deviation is zero
```

```
print(cor_matrix_3)
```

```
##                    nbagameid      O_rank_1     O_rank_2     O_rank_3
## nbagameid         1.000000000  0.0081911757 -0.071812214  0.046469816
## O_rank_1          0.008191176  1.0000000000  0.166026147  0.137263481
## O_rank_2         -0.071812214  0.1660261466  1.000000000  0.148513862
## O_rank_3          0.046469816  0.1372634810  0.148513862  1.000000000
## O_rank_4         -0.048968888  0.0747597087  0.061923212  0.154945860
## O_rank_5         -0.122640056  0.0137398888 -0.050661804  0.077259503
## O_rank_6         -0.046401231 -0.1542981783  0.042811960 -0.063471148
## O_rank_7          0.102319015 -0.0296151871  0.140656889  0.015812545
## O_rank_8         -0.055144490 -0.0099823419 -0.106675709 -0.058846633
## O_rank_9          0.059227582 -0.1727107834 -0.201589949 -0.071495904
## O_rank_10        -0.067474920  0.0002782541 -0.064909174 -0.197491347
## O_rank_11        -0.097867312 -0.1524974410  0.065073632 -0.093065718
## off_home                   NA            NA           NA           NA
## D_rank_1          0.057211863  0.0902710593  0.075727173  0.045060281
```

```
## D_rank_2          -0.048902267  0.0812446091  0.054145816  0.011725947
## D_rank_3           0.035275694  0.0535078539  0.027546671  0.032048340
## D_rank_4          -0.072851129  0.0608217398  0.063657755  0.013486520
## D_rank_5          -0.121956600  0.0783305719  0.052119199  0.002030186
## D_rank_6          -0.048387513  0.0346877242  0.023920502  0.025536630
## D_rank_7           0.098482622  0.0304693365  0.010097763  0.016710486
## D_rank_8          -0.103734509  0.0082276204  0.007366989  0.022760581
## D_rank_9           0.046598761 -0.0220672010 -0.033301702  0.023108151
## D_rank_10         -0.021799681 -0.0390215476 -0.013272547 -0.035805514
## D_rank_11         -0.057450645 -0.0237716342 -0.046052423 -0.003386731
## Oefg              -0.012842173  0.0538475241  0.053767160  0.071440230
## OTORatio          -0.023077830 -0.1295929801 -0.054071805  0.003855058
## O_reb_perc        -0.048040024 -0.0807200443 -0.007951229 -0.038319913
## OFTRate           -0.140846743  0.0632963259  0.057883114  0.055558410
## O_west_conference  0.008171976  0.0517088500 -0.059672409  0.137843822
## O_win             -0.052386552  0.1472754584  0.092512193  0.109160415
## O_rank             0.010560821 -0.2600204412 -0.150327976 -0.194445361
## Defg               0.009532702 -0.0470471357 -0.035868305 -0.039520592
## DTORatio          -0.029229427 -0.0612759796 -0.112066457 -0.029199941
## D_reb_perc        -0.073814090 -0.0792616658 -0.061352691 -0.022322311
## DFTRate           -0.126321604 -0.0432704698 -0.005416299 -0.035519194
## D_west_conference -0.004462645  0.0039910717 -0.040619232  0.027726993
## D_rank            -0.005886045 -0.0984428709 -0.036891398 -0.011916967
##                       O_rank_4     O_rank_5      O_rank_6      O_rank_7
## nbagameid         -0.048968888 -0.122640056 -0.0464012307  1.023190e-01
## O_rank_1           0.074759709  0.013739889 -0.1542981783 -2.961519e-02
## O_rank_2           0.061923212 -0.050661804  0.0428119600  1.406569e-01
## O_rank_3           0.154945860  0.077259503 -0.0634711475  1.581254e-02
## O_rank_4           1.000000000 -0.024973450 -0.0127660370  5.889409e-02
## O_rank_5          -0.024973450  1.000000000  0.0096115955 -7.278216e-03
## O_rank_6          -0.012766037  0.009611595  1.0000000000 -3.617801e-03
## O_rank_7           0.058894095 -0.007278216 -0.0036178008  1.000000e+00
## O_rank_8           0.008609908  0.005853009  0.0003399896 -3.032012e-02
## O_rank_9          -0.076384864 -0.009251304 -0.0829281476 -1.134112e-01
## O_rank_10         -0.168365884 -0.092862403  0.0024519782  2.715135e-02
## O_rank_11         -0.251935336 -0.119931893 -0.0940177111 -1.276308e-01
## off_home                    NA           NA            NA            NA
## D_rank_1           0.048563311 -0.002721267  0.0009330084  1.008063e-02
## D_rank_2           0.027219972  0.053999376 -0.0204599597  4.169286e-02
## D_rank_3           0.028961095 -0.020958608  0.0297814501 -1.071992e-02
## D_rank_4           0.021009782  0.060603086  0.0159013981  4.593009e-02
## D_rank_5           0.021424315 -0.009597280 -0.0319428662  1.804863e-03
## D_rank_6           0.029300973  0.003365049 -0.0519738277  2.227805e-02
## D_rank_7          -0.014641049 -0.015060293  0.0260651380  5.472739e-02
## D_rank_8           0.017936023  0.033242466  0.0220210120  1.810030e-02
## D_rank_9          -0.022903583 -0.008823931 -0.0137194107  3.710314e-03
## D_rank_10          0.010556692 -0.020833278 -0.0119430605 -3.870766e-02
## D_rank_11         -0.076887616 -0.012144167  0.0624740566 -6.730463e-02
## Oefg               0.067595377 -0.005270510  0.0814399458  8.113282e-02
## OTORatio          -0.049334873 -0.002898440 -0.0206427662 -1.687917e-02
## O_reb_perc        -0.016727514 -0.017557254 -0.0392989586  2.988492e-02
## OFTRate            0.064122197  0.029111135  0.0146789414 -3.344690e-02
## O_west_conference  0.055856974  0.161273260  0.0603039390 -6.430256e-02
## O_win              0.103504888  0.006611133 -0.0156315495  7.593999e-02
```

```
## O_rank            -0.056173429 -0.041280873 -0.0607474254 -1.219657e-01
## Defg              -0.072262225 -0.007527534  0.0348834288 -3.064387e-02
## DTORatio          -0.045313986 -0.001575250 -0.0447756852 -4.386660e-02
## D_reb_perc         0.014678811  0.027867115  0.0446143313  2.438181e-05
## DFTRate            0.019752595  0.027248763  0.0340109418 -1.979583e-02
## D_west_conference  0.053378898  0.036788019 -0.0143504202 -5.698331e-02
## D_rank            -0.013086820 -0.024250236 -0.0161392172 -2.064380e-02
##                        O_rank_8      O_rank_9     O_rank_10     O_rank_11
## nbagameid         -0.0551444898  0.0592275816 -0.0674749197 -9.786731e-02
## O_rank_1          -0.0099823419 -0.1727107834  0.0002782541 -1.524974e-01
## O_rank_2          -0.1066757088 -0.2015899488 -0.0649091742  6.507363e-02
## O_rank_3          -0.0588466333 -0.0714959041 -0.1974913467 -9.306572e-02
## O_rank_4           0.0086099080 -0.0763848636 -0.1683658836 -2.519353e-01
## O_rank_5           0.0058530087 -0.0092513044 -0.0928624033 -1.199319e-01
## O_rank_6           0.0003399896 -0.0829281476  0.0024519782 -9.401771e-02
## O_rank_7          -0.0303201156 -0.1134112456  0.0271513459 -1.276308e-01
## O_rank_8           1.0000000000 -0.1027869795 -0.1042850633 -1.253243e-02
## O_rank_9          -0.1027869795  1.0000000000 -0.0055744174  1.064611e-02
## O_rank_10         -0.1042850633 -0.0055744174  1.0000000000  2.560410e-02
## O_rank_11         -0.0125324284  0.0106461119  0.0256041046  1.000000e+00
## off_home                     NA            NA            NA            NA
## D_rank_1          -0.0379751708 -0.0007513911  0.0163685497  2.688237e-05
## D_rank_2           0.0026060939 -0.0290393686  0.0141145367  5.800086e-03
## D_rank_3           0.0220268103  0.0050958174  0.0083114906 -5.183975e-02
## D_rank_4          -0.0188898496 -0.0316810744  0.0215009395 -2.358138e-02
## D_rank_5           0.0103889761 -0.0135300624 -0.0057422159  1.310829e-02
## D_rank_6          -0.0242933181 -0.0020191533  0.0184614823 -4.382244e-02
## D_rank_7          -0.0017569255  0.0289400528  0.0107164610 -2.528240e-02
## D_rank_8          -0.0354185690 -0.0123274231  0.0159581210 -1.664627e-02
## D_rank_9           0.0001752845  0.0161069805 -0.0187193273  3.526812e-02
## D_rank_10          0.0327248950  0.0083121057 -0.0401889626  3.555678e-02
## D_rank_11         -0.0115521110  0.0265513735 -0.0071575668  4.263727e-02
## Oefg               0.0107197831 -0.0683328364  0.0528544415  2.361454e-02
## OTORatio           0.0342315292 -0.0068739516 -0.0344147307 -1.667087e-02
## O_reb_perc         0.0295295018  0.0301061140  0.0619050234  6.408960e-02
## OFTRate            0.0182289106 -0.0123827525 -0.0259335200  3.787126e-02
## O_west_conference  0.0572984366 -0.0227924933  0.0105999837 -6.865751e-02
## O_win              0.0308206848 -0.0313028955  0.0363085198  5.437268e-02
## O_rank            -0.0607344137  0.1597439151 -0.0263544057  1.726943e-02
## Defg              -0.0152537578 -0.0058264057  0.0308197102 -4.928438e-02
## DTORatio           0.0525339720  0.0642654473  0.0195244513 -1.795383e-02
## D_reb_perc        -0.0103526601 -0.0067836279  0.0352659050 -1.784376e-02
## DFTRate            0.0155260230  0.0217169154  0.0276187342  1.248371e-02
## D_west_conference  0.0515778283  0.0348554292 -0.0058744136 -2.321888e-03
## D_rank             0.0010063345  0.0019322399 -0.0139483410  5.892481e-02
##                   off_home      D_rank_1     D_rank_2     D_rank_3     D_rank_4
## nbagameid               NA  5.721186e-02 -0.048902267  0.035275694 -0.072851129
## O_rank_1                NA  9.027106e-02  0.081244609  0.053507854  0.060821740
## O_rank_2                NA  7.572717e-02  0.054145816  0.027546671  0.063657755
## O_rank_3                NA  4.506028e-02  0.011725947  0.032048340  0.013486520
## O_rank_4                NA  4.856331e-02  0.027219972  0.028961095  0.021009782
## O_rank_5                NA -2.721267e-03  0.053999376 -0.020958608  0.060603086
## O_rank_6                NA  9.330084e-04 -0.020459960  0.029781450  0.015901398
## O_rank_7                NA  1.008063e-02  0.041692860 -0.010719917  0.045930087
```

```
## O_rank_8               NA -3.797517e-02  0.002606094  0.022026810 -0.018889850
## O_rank_9               NA -7.513911e-04 -0.029039369  0.005095817 -0.031681074
## O_rank_10              NA  1.636855e-02  0.014114537  0.008311491  0.021500940
## O_rank_11              NA  2.688237e-05  0.005800086 -0.051839749 -0.023581377
## off_home                1            NA           NA           NA           NA
## D_rank_1               NA  1.000000e+00  0.140784027  0.111225218  0.082941364
## D_rank_2               NA  1.407840e-01  1.000000000  0.173973924  0.075395287
## D_rank_3               NA  1.112252e-01  0.173973924  1.000000000  0.169684936
## D_rank_4               NA  8.294136e-02  0.075395287  0.169684936  1.000000000
## D_rank_5               NA -1.731442e-02 -0.042413558 -0.004538501 -0.016088948
## D_rank_6               NA -9.929000e-02  0.085372466  0.056474397 -0.033778708
## D_rank_7               NA  3.455099e-02  0.130405038  0.074250146  0.061917719
## D_rank_8               NA -4.594965e-02 -0.105110653 -0.003586858 -0.028260455
## D_rank_9               NA -7.337770e-02 -0.149316893 -0.119551684 -0.039774585
## D_rank_10              NA -1.415468e-01 -0.144140723 -0.247319576 -0.173743094
## D_rank_11              NA -1.517921e-01 -0.010967254 -0.085778357 -0.230065733
## Oefg                   NA -8.478750e-02 -0.033908898 -0.071573868 -0.080060410
## OTORatio               NA  3.623498e-02  0.019958607 -0.044054430 -0.042355405
## O_reb_perc             NA -5.831377e-02 -0.061998778 -0.027123472  0.007244799
## OFTRate                NA -6.764238e-02  0.054763497 -0.012756682 -0.018759561
## O_west_conference      NA  3.762652e-02 -0.009488473  0.018706117 -0.013000504
## O_win                  NA -1.123619e-01 -0.082524932 -0.078525479 -0.044330724
## O_rank                 NA  7.526411e-02  0.043227138  0.033387070 -0.035822440
## Defg                   NA  5.699442e-02  0.110856684  0.135521726  0.062512205
## DTORatio               NA -1.382464e-01 -0.107888475 -0.042986783 -0.078970881
## D_reb_perc             NA -8.131050e-02 -0.023865797  0.043885815 -0.092994925
## DFTRate                NA  3.036428e-02  0.076020781 -0.037200971  0.020206464
## D_west_conference      NA  3.681013e-02 -0.069865805  0.167252484  0.086563220
## D_rank                 NA -2.716692e-01 -0.206883732 -0.202274463 -0.111613856
##                      D_rank_5     D_rank_6      D_rank_7     D_rank_8
## nbagameid         -0.121956600 -0.048387513  0.0984826223 -0.103734509
## O_rank_1           0.078330572  0.034687724  0.0304693365  0.008227620
## O_rank_2           0.052119199  0.023920502  0.0100977630  0.007366989
## O_rank_3           0.002030186  0.025536630  0.0167104857  0.022760581
## O_rank_4           0.021424315  0.029300973 -0.0146410492  0.017936023
## O_rank_5          -0.009597280  0.003365049 -0.0150602932  0.033242466
## O_rank_6          -0.031942866 -0.051973828  0.0260651380  0.022021012
## O_rank_7           0.001804863  0.022278054  0.0547273907  0.018100304
## O_rank_8           0.010388976 -0.024293318 -0.0017569255 -0.035418569
## O_rank_9          -0.013530062 -0.002019153  0.0289400528 -0.012327423
## O_rank_10         -0.005742216  0.018461482  0.0107164610  0.015958121
## O_rank_11          0.013108290 -0.043822444 -0.0252823999 -0.016646268
## off_home                    NA           NA            NA           NA
## D_rank_1          -0.017314417 -0.099290000  0.0345509906 -0.045949651
## D_rank_2          -0.042413558  0.085372466  0.1304050381 -0.105110653
## D_rank_3          -0.004538501  0.056474397  0.0742501459 -0.003586858
## D_rank_4          -0.016088948 -0.033778708  0.0619177195 -0.028260455
## D_rank_5           1.000000000 -0.044956514 -0.0215418033 -0.015073026
## D_rank_6          -0.044956514  1.000000000 -0.0121543740 -0.008053698
## D_rank_7          -0.021541803 -0.012154374  1.0000000000 -0.046775195
## D_rank_8          -0.015073026 -0.008053698 -0.0467751950  1.000000000
## D_rank_9          -0.023897752 -0.076574066 -0.0920773053 -0.120991869
## D_rank_10          0.015074414 -0.007330629  0.0084384436 -0.102697396
## D_rank_11         -0.045431391 -0.092110383 -0.1377525007 -0.013163852
```

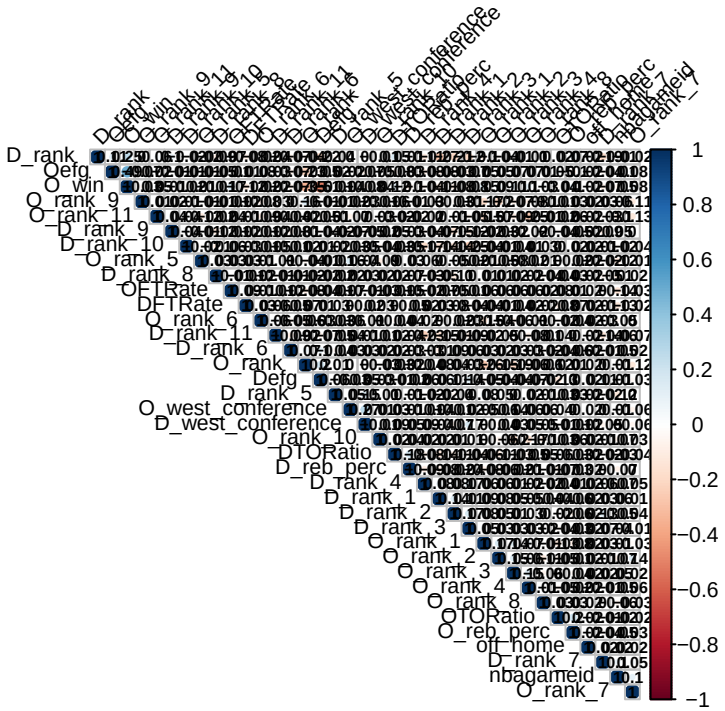
```
## Oefg               0.016579326 -0.074833556 -0.0368884463 -0.047866954
## OTORatio          -0.032606145 -0.043798771 -0.0120599418 -0.044008437
## O_reb_perc         0.031189689 -0.059775134 -0.0369319850  0.028561480
## OFTRate           -0.011610422  0.015091090  0.0001582047 -0.010089228
## O_west_conference  0.053443408  0.034051996 -0.0049438716  0.015350097
## O_win             -0.009687556 -0.068994829 -0.0716462153 -0.009483912
## O_rank             0.006247220  0.073874540 -0.0003381653 -0.015644212
## Defg               0.059703728  0.102974821  0.1123396279 -0.016187186
## DTORatio           0.004785885  0.021216281 -0.0178663266  0.018505835
## D_reb_perc         0.005220148  0.027428524  0.0008233862  0.069236014
## DFTRate            0.025562737 -0.051015187  0.0104994237 -0.008505070
## D_west_conference  0.149924185  0.029881906 -0.0514822411  0.026576158
## D_rank            -0.042843850 -0.068826429 -0.1896373196 -0.085963651
##                        D_rank_9    D_rank_10    D_rank_11         Oefg
## nbagameid          0.0465987610 -0.021799681 -0.057450645 -0.012842173
## O_rank_1          -0.0220672010 -0.039021548 -0.023771634  0.053847524
## O_rank_2          -0.0333017021 -0.013272547 -0.046052423  0.053767160
## O_rank_3           0.0231081511 -0.035805514 -0.003386731  0.071440230
## O_rank_4          -0.0229035832  0.010556692 -0.076887616  0.067595377
## O_rank_5          -0.0088239312 -0.020833278 -0.012144167 -0.005270510
## O_rank_6          -0.0137194107 -0.011943060  0.062474057  0.081439946
## O_rank_7           0.0037103141 -0.038707662 -0.067304627  0.081132818
## O_rank_8           0.0001752845  0.032724895 -0.011552111  0.010719783
## O_rank_9           0.0161069805  0.008312106  0.026551374 -0.068332836
## O_rank_10         -0.0187193273 -0.040188963 -0.007157567  0.052854441
## O_rank_11          0.0352681245  0.035556784  0.042637270  0.023614544
## off_home                     NA           NA           NA           NA
## D_rank_1          -0.0733777034 -0.141546766 -0.151792119 -0.084787503
## D_rank_2          -0.1493168934 -0.144140723 -0.010967254 -0.033908898
## D_rank_3          -0.1195516845 -0.247319576 -0.085778357 -0.071573868
## D_rank_4          -0.0397745850 -0.173743094 -0.230065733 -0.080060410
## D_rank_5          -0.0238977518  0.015074414 -0.045431391  0.016579326
## D_rank_6          -0.0765740663 -0.007330629 -0.092110383 -0.074833556
## D_rank_7          -0.0920773053  0.008438444 -0.137752501 -0.036888446
## D_rank_8          -0.1209918692 -0.102697396 -0.013163852 -0.047866954
## D_rank_9           1.0000000000  0.040342612  0.016604309 -0.010505376
## D_rank_10          0.0403426123  1.000000000  0.050663246 -0.005479448
## D_rank_11          0.0166043095  0.050663246  1.000000000  0.031885580
## Oefg              -0.0105053758 -0.005479448  0.031885580  1.000000000
## OTORatio          -0.0353994840  0.004166315  0.044447348  0.050732851
## O_reb_perc        -0.0478499145  0.023093908  0.002489461 -0.098887165
## OFTRate           -0.0053799438  0.060231923 -0.009936913 -0.010250076
## O_west_conference -0.0662120890 -0.026847005  0.037128109  0.019364208
## O_win              0.0053032147 -0.023637202  0.016572031  0.494638773
## O_rank             0.0053511161  0.022472571  0.020967099 -0.229850548
## Defg              -0.0352828485  0.007698618 -0.066468687 -0.045072744
## DTORatio           0.0530230901 -0.028210705  0.020321878 -0.080484261
## D_reb_perc         0.0298874035  0.049728848  0.038598671  0.032211568
## DFTRate            0.0171301311  0.029867174  0.064467151  0.011401812
## D_west_conference -0.0457773837  0.050533066  0.005041742 -0.070209089
## D_rank             0.0962096770 -0.024650972  0.035546794  0.111307465
##                       OTORatio    O_reb_perc       OFTRate O_west_conference
## nbagameid         -0.023077830 -0.0480400237 -0.1408467435      0.0081719762
## O_rank_1          -0.129592980 -0.0807200443  0.0632963259      0.0517088500
```

```
## O_rank_2          -0.054071805 -0.0079512286  0.0578831136     -0.0596724085
## O_rank_3           0.003855058 -0.0383199130  0.0555584103      0.1378438220
## O_rank_4          -0.049334873 -0.0167275137  0.0641221966      0.0558569742
## O_rank_5          -0.002898440 -0.0175572538  0.0291111351      0.1612732599
## O_rank_6          -0.020642766 -0.0392989586  0.0146789414      0.0603039390
## O_rank_7          -0.016879172  0.0298849224 -0.0334468974     -0.0643025602
## O_rank_8           0.034231529  0.0295295018  0.0182289106      0.0572984366
## O_rank_9          -0.006873952  0.0301061140 -0.0123827525     -0.0227924933
## O_rank_10         -0.034414731  0.0619050234 -0.0259335200      0.0105999837
## O_rank_11         -0.016670873  0.0640896019  0.0378712641     -0.0686575079
## off_home                    NA            NA            NA                NA
## D_rank_1           0.036234976 -0.0583137710 -0.0676423830      0.0376265177
## D_rank_2           0.019958607 -0.0619987783  0.0547634969     -0.0094884730
## D_rank_3          -0.044054430 -0.0271234725 -0.0127566819      0.0187061167
## D_rank_4          -0.042355405  0.0072447987 -0.0187595608     -0.0130005038
## D_rank_5          -0.032606145  0.0311896890 -0.0116104216      0.0534434077
## D_rank_6          -0.043798771 -0.0597751343  0.0150910897      0.0340519964
## D_rank_7          -0.012059942 -0.0369319850  0.0001582047     -0.0049438716
## D_rank_8          -0.044008437  0.0285614799 -0.0100892275      0.0153500965
## D_rank_9          -0.035399484 -0.0478499145 -0.0053799438     -0.0662120890
## D_rank_10          0.004166315  0.0230939085  0.0602319229     -0.0268470051
## D_rank_11          0.044447348  0.0024894606 -0.0099369134      0.0371281089
## Oefg               0.050732851 -0.0988871646 -0.0102500760      0.0193642078
## OTORatio           1.000000000  0.1976903529  0.0778223366      0.0429901340
## O_reb_perc         0.197690353  1.0000000000  0.0079514937      0.0002135527
## OFTRate            0.077822337  0.0079514937  1.0000000000      0.0691142784
## O_west_conference  0.042990134  0.0002135527  0.0691142784      1.0000000000
## O_win             -0.037844057  0.1025542919  0.1669875368      0.0359059393
## O_rank             0.118994620  0.0126475798 -0.0804913922      0.0024476108
## Defg              -0.134285450 -0.0023827942 -0.0385572941     -0.0346692383
## DTORatio           0.060689158 -0.0311873506 -0.0110603869      0.0312553315
## D_reb_perc        -0.071880170 -0.0254893881  0.0520973801      0.0112213243
## DFTRate           -0.080669277  0.0738426481  0.0877864974     -0.0032411987
## D_west_conference -0.010528281 -0.0120585011 -0.0063273608      0.2650406504
## D_rank             0.016723923  0.0694380664  0.0652326352     -0.0039495300
##                          O_win        O_rank         Defg     DTORatio
## nbagameid         -0.052386552  0.0105608209  0.009532702 -0.029229427
## O_rank_1           0.147275458 -0.2600204412 -0.047047136 -0.061275980
## O_rank_2           0.092512193 -0.1503279758 -0.035868305 -0.112066457
## O_rank_3           0.109160415 -0.1944453610 -0.039520592 -0.029199941
## O_rank_4           0.103504888 -0.0561734292 -0.072262225 -0.045313986
## O_rank_5           0.006611133 -0.0412808728 -0.007527534 -0.001575250
## O_rank_6          -0.015631550 -0.0607474254  0.034883429 -0.044775685
## O_rank_7           0.075939987 -0.1219656653 -0.030643870 -0.043866598
## O_rank_8           0.030820685 -0.0607344137 -0.015253758  0.052533972
## O_rank_9          -0.031302896  0.1597439151 -0.005826406  0.064265447
## O_rank_10          0.036308520 -0.0263544057  0.030819710  0.019524451
## O_rank_11          0.054372682  0.0172694290 -0.049284381 -0.017953826
## off_home                    NA            NA            NA           NA
## D_rank_1          -0.112361920  0.0752641063  0.056994423 -0.138246417
## D_rank_2          -0.082524932  0.0432271383  0.110856684 -0.107888475
## D_rank_3          -0.078525479  0.0333870704  0.135521726 -0.042986783
## D_rank_4          -0.044330724 -0.0358224399  0.062512205 -0.078970881
## D_rank_5          -0.009687556  0.0062472202  0.059703728  0.004785885
```

```
## D_rank_6          -0.068994829  0.0738745397  0.102974821  0.021216281
## D_rank_7          -0.071646215 -0.0003381653  0.112339628 -0.017866327
## D_rank_8          -0.009483912 -0.0156442124 -0.016187186  0.018505835
## D_rank_9           0.005303215  0.0053511161 -0.035282849  0.053023090
## D_rank_10         -0.023637202  0.0224725707  0.007698618 -0.028210705
## D_rank_11          0.016572031  0.0209670994 -0.066468687  0.020321878
## Oefg               0.494638773 -0.2298505481 -0.045072744 -0.080484261
## OTORatio          -0.037844057  0.1189946197 -0.134285450  0.060689158
## O_reb_perc         0.102554292  0.0126475798 -0.002382794 -0.031187351
## OFTRate            0.166987537 -0.0804913922 -0.038557294 -0.011060387
## O_west_conference  0.035905939  0.0024476108 -0.034669238  0.031255331
## O_win              1.000000000 -0.3454667730 -0.505136047  0.122927034
## O_rank            -0.345466773  1.0000000000  0.199103870 -0.030816352
## Defg              -0.505136047  0.1991038702  1.000000000 -0.013636355
## DTORatio           0.122927034 -0.0308163517 -0.013636355  1.000000000
## D_reb_perc        -0.097047953  0.0220894209 -0.023230440  0.184503603
## DFTRate           -0.116421389  0.0652914606  0.008256403 -0.001790459
## D_west_conference -0.078340231  0.0028241663  0.046899115  0.086454244
## D_rank             0.254856288 -0.0430708384 -0.215131283  0.152991286
##                     D_reb_perc      DFTRate D_west_conference       D_rank
## nbagameid         -7.381409e-02 -0.126321604      -0.004462645 -0.005886045
## O_rank_1          -7.926167e-02 -0.043270470       0.003991072 -0.098442871
## O_rank_2          -6.135269e-02 -0.005416299      -0.040619232 -0.036891398
## O_rank_3          -2.232231e-02 -0.035519194       0.027726993 -0.011916967
## O_rank_4           1.467881e-02  0.019752595       0.053378898 -0.013086820
## O_rank_5           2.786712e-02  0.027248763       0.036788019 -0.024250236
## O_rank_6           4.461433e-02  0.034010942      -0.014350420 -0.016139217
## O_rank_7           2.438181e-05 -0.019795832      -0.056983305 -0.020643804
## O_rank_8          -1.035266e-02  0.015526023       0.051577828  0.001006334
## O_rank_9          -6.783628e-03  0.021716915       0.034855429  0.001932240
## O_rank_10          3.526590e-02  0.027618734      -0.005874414 -0.013948341
## O_rank_11         -1.784376e-02  0.012483708      -0.002321888  0.058924811
## off_home                     NA           NA                NA           NA
## D_rank_1          -8.131050e-02  0.030364280       0.036810126 -0.271669236
## D_rank_2          -2.386580e-02  0.076020781      -0.069865805 -0.206883732
## D_rank_3           4.388582e-02 -0.037200971       0.167252484 -0.202274463
## D_rank_4          -9.299493e-02  0.020206464       0.086563220 -0.111613856
## D_rank_5           5.220148e-03  0.025562737       0.149924185 -0.042843850
## D_rank_6           2.742852e-02 -0.051015187       0.029881906 -0.068826429
## D_rank_7           8.233862e-04  0.010499424      -0.051482241 -0.189637320
## D_rank_8           6.923601e-02 -0.008505070       0.026576158 -0.085963651
## D_rank_9           2.988740e-02  0.017130131      -0.045777384  0.096209677
## D_rank_10          4.972885e-02  0.029867174       0.050533066 -0.024650972
## D_rank_11          3.859867e-02  0.064467151       0.005041742  0.035546794
## Oefg               3.221157e-02  0.011401812      -0.070209089  0.111307465
## OTORatio          -7.188017e-02 -0.080669277      -0.010528281  0.016723923
## O_reb_perc        -2.548939e-02  0.073842648      -0.012058501  0.069438066
## OFTRate            5.209738e-02  0.087786497      -0.006327361  0.065232635
## O_west_conference  1.122132e-02 -0.003241199       0.265040650 -0.003949530
## O_win             -9.704795e-02 -0.116421389      -0.078340231  0.254856288
## O_rank             2.208942e-02  0.065291461       0.002824166 -0.043070838
## Defg              -2.323044e-02  0.008256403       0.046899115 -0.215131283
## DTORatio           1.845036e-01 -0.001790459       0.086454244  0.152991286
## D_reb_perc         1.000000e+00 -0.047378713       0.052019689  0.009654269
```

```
## DFTRate           -4.737871e-02  1.000000000      -0.023580660 -0.079707967
## D_west_conference  5.201969e-02 -0.023580660       1.000000000 -0.002444947
## D_rank             9.654269e-03 -0.079707967      -0.002444947  1.000000000
```

```
cor_matrix_3[is.na(cor_matrix_3)] <- mean(cor_matrix_3, na.rm = TRUE)
clust_3 <- hclust(as.dist(1 - cor_matrix_3), method = "ward.D2")


par(oma = c(2, 2, 2, 2))
par(mar = c(2, 2, 2, 2))

# Plot the correlation matrix
corrplot(cor_matrix_3,
         method = "circle",
         type = "upper",
         order = "hclust",
         addCoef.col = "black",
         tl.col = "black",
         tl.srt = 45,
         tl.cex = 0.7,  # Text label size
         number.cex = 0.5,  # Number size
         cl.cex = 0.7)  # Color legend size
```




```
# scale data so ranking doesn't skew the results
model3_clean_df$O_rank_1 <- scales::rescale(model3_clean_df$O_rank_1)
```

```r
model3_clean_df$O_rank_2 <- scales::rescale(model3_clean_df$O_rank_2)
model3_clean_df$O_rank_3 <- scales::rescale(model3_clean_df$O_rank_3)
model3_clean_df$O_rank_4 <- scales::rescale(model3_clean_df$O_rank_4)
model3_clean_df$O_rank_5 <- scales::rescale(model3_clean_df$O_rank_5)
model3_clean_df$O_rank_6 <- scales::rescale(model3_clean_df$O_rank_6)
model3_clean_df$O_rank_7 <- scales::rescale(model3_clean_df$O_rank_7)
model3_clean_df$O_rank_8 <- scales::rescale(model3_clean_df$O_rank_8)
model3_clean_df$O_rank_9 <- scales::rescale(model3_clean_df$O_rank_9)
model3_clean_df$O_rank_10 <- scales::rescale(model3_clean_df$O_rank_10)
model3_clean_df$O_rank_11 <- scales::rescale(model3_clean_df$O_rank_11)

model3_clean_df$D_rank_1 <- scales::rescale(model3_clean_df$D_rank_1)
model3_clean_df$D_rank_2 <- scales::rescale(model3_clean_df$D_rank_2)
model3_clean_df$D_rank_3 <- scales::rescale(model3_clean_df$D_rank_3)
model3_clean_df$D_rank_4 <- scales::rescale(model3_clean_df$D_rank_4)
model3_clean_df$D_rank_5 <- scales::rescale(model3_clean_df$D_rank_5)
model3_clean_df$D_rank_6 <- scales::rescale(model3_clean_df$D_rank_6)
model3_clean_df$D_rank_7 <- scales::rescale(model3_clean_df$D_rank_7)
model3_clean_df$D_rank_8 <- scales::rescale(model3_clean_df$D_rank_8)
model3_clean_df$D_rank_9 <- scales::rescale(model3_clean_df$D_rank_9)
model3_clean_df$D_rank_10 <- scales::rescale(model3_clean_df$D_rank_10)
model3_clean_df$D_rank_11 <- scales::rescale(model3_clean_df$D_rank_11)

print(model3_clean_df)
```

```
## # A tibble: 1,230 x 39
##    nbagameid off_team O_rank_1 O_rank_2 O_rank_3 O_ran~1 O_ran~2 O_ran~3 O_ran~4
##        <dbl> <chr>       <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
##  1  22300001 IND         0        0.685    0.584   0.636   0.754   0.630   0.439
##  2  22300002 MIL         0.731    0.714    0.677   0.469   0.334   0.549   0.448
##  3  22300003 MIA         0.609    0.431    0       0.659   0.514   0.677   0
##  4  22300004 CHI         0.679    0.708    0.681   0       0.530   0       0.265
##  5  22300005 OKC         0        0.638    0.576   0.743   0.615   0.832   0.562
##  6  22300006 DEN         0.724    0.688    0.705   0.742   0.789   0.455   0.314
##  7  22300007 POR         0        0.873    0.692   0       0.634   0.213   0
##  8  22300008 DET         0        0        0.681   0.478   0.776   0.741   0
##  9  22300009 WAS         0.633    0.580    0.586   0.344   0.605   0.773   0.422
## 10  22300010 BOS         0.675    0.535    0.561   0.690   0.615   0.565   0
## # ... with 1,220 more rows, 30 more variables: O_rank_8 <dbl>, O_rank_9 <dbl>,
## #   O_rank_10 <dbl>, O_rank_11 <dbl>, off_home <dbl>, def_team <chr>,
## #   D_rank_1 <dbl>, D_rank_2 <dbl>, D_rank_3 <dbl>, D_rank_4 <dbl>,
## #   D_rank_5 <dbl>, D_rank_6 <dbl>, D_rank_7 <dbl>, D_rank_8 <dbl>,
## #   D_rank_9 <dbl>, D_rank_10 <dbl>, D_rank_11 <dbl>, Oefg <dbl>,
## #   OTORatio <dbl>, O_reb_perc <dbl>, OFTRate <dbl>, O_west_conference <dbl>,
## #   O_win <dbl>, O_rank <dbl>, Defg <dbl>, DTORatio <dbl>, ...
```

```r
# split the data into training and testing
set.seed(2581)
split_index <- sample(1:nrow(model3_clean_df), 0.7*nrow(model3_clean_df))
train <- model3_clean_df[split_index, ]
test <- model3_clean_df[-split_index, ]
```

```r
# use automl to retrieve a baseline model

library(h2o)
h2o.init()
```

```
##
## H2O is not running yet, starting it now...
##
## Note:  In case of errors look at the following log files:
##     /var/folders/13/__jfy01d5nxd0qk1cmrhcpzh0000gn/T//RtmpWw7eps/file6d854687e2d7/h2o_linanguyen_sta
##     /var/folders/13/__jfy01d5nxd0qk1cmrhcpzh0000gn/T//RtmpWw7eps/file6d8533db56e/h2o_linanguyen_star
##
##
## Starting H2O JVM and connecting: .... Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         4 seconds 225 milliseconds
##     H2O cluster timezone:       America/Los_Angeles
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.44.0.3
##     H2O cluster version age:    1 year and 25 days
##     H2O cluster name:           H2O_started_from_R_linanguyen_ese095
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   3.54 GB
##     H2O cluster total cores:    8
##     H2O cluster allowed cores:  8
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     R Version:                  R version 4.1.0 (2021-05-18)

## Warning in h2o.clusterInfo():
## Your H2O cluster version is (1 year and 25 days) old. There may be a newer version available.
## Please download and install the latest version from: https://h2o-release.s3.amazonaws.com/h2o/latest
```

```r
# convert to h2o objects
train_h2o <- as.h2o(train)
```

```
##   |                                                                      |
```

```r
test_h2o <- as.h2o(test)
```

```
##   |                                                                      |
```

```r
target <- "O_win"
features <- setdiff(names(train), target)

# train h2o model
model <- h2o.automl(
```

```
  x = features,
  y = target,
  training_frame = train_h2o,
  max_models = 10,
  seed = 1
)
```

```
##   |                                                                              |
## 20:09:17.269: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:17.269: _response param, We have detected that your response column has only 2 unique values (
## 20:09:19.876: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:19.876: _response param, We have detected that your response column has only 2 unique values (
## 20:09:21.56: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:21.56: _response param, We have detected that your response column has only 2 unique values (0,
## 20:09:23.931: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:23.932: _response param, We have detected that your response column has only 2 unique values (
## 20:09:24.801: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:24.801: _response param, We have detected that your response column has only 2 unique values (
## 20:09:27.900: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:27.900: _response param, We have detected that your response column has only 2 unique values (
## 20:09:29.238: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:29.238: _response param, We have detected that your response column has only 2 unique values (
## 20:09:30.110: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:30.110: _response param, We have detected that your response column has only 2 unique values (
## 20:09:30.968: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:30.968: _response param, We have detected that your response column has only 2 unique values (
## 20:09:31.654: _train param, Dropping bad and constant columns: [off_team, def_team, off_home]
## 20:09:31.654: _response param, We have detected that your response column has only 2 unique values (
## 20:09:33.77: _train param, Dropping unused columns: [off_team, def_team, off_home]
## 20:09:33.77: _response param, We have detected that your response column has only 2 unique values (0,
##
## 20:09:33.926: _train param, Dropping unused columns: [off_team, def_team, off_home]
## 20:09:33.926: _response param, We have detected that your response column has only 2 unique values (
```

```
best_model <- model@leader
lb <- model@leaderboard
print(lb, n = nrow(lb))
```

```
##                                                 model_id      rmse        mse
## 1     StackedEnsemble_AllModels_1_AutoML_1_20250114_200917 0.2814631 0.07922146
## 2  StackedEnsemble_BestOfFamily_1_AutoML_1_20250114_200917 0.2848294 0.08112781
## 3                            GLM_1_AutoML_1_20250114_200917 0.2974662 0.08848613
## 4                            GBM_1_AutoML_1_20250114_200917 0.3024089 0.09145114
## 5                            GBM_2_AutoML_1_20250114_200917 0.3042243 0.09255243
## 6                            GBM_4_AutoML_1_20250114_200917 0.3065232 0.09395645
## 7                            GBM_3_AutoML_1_20250114_200917 0.3116740 0.09714067
## 8                         XGBoost_1_AutoML_1_20250114_200917 0.3264974 0.10660053
## 9                            XRT_1_AutoML_1_20250114_200917 0.3300903 0.10895961
## 10                           DRF_1_AutoML_1_20250114_200917 0.3316777 0.11001010
## 11                        XGBoost_3_AutoML_1_20250114_200917 0.3379424 0.11420505
## 12                        XGBoost_2_AutoML_1_20250114_200917 0.3459811 0.11970295
##          mae     rmsle mean_residual_deviance
## 1  0.2260197 0.2043577             0.07922146
```

```
## 2  0.2332468 0.2081517            0.08112781
## 3  0.2524031 0.2335420            0.08848613
## 4  0.2434468 0.2146679            0.09145114
## 5  0.2294774 0.2151602            0.09255243
## 6  0.2267413 0.2155188            0.09395645
## 7  0.2300143 0.2195638            0.09714067
## 8  0.2544756 0.2348664            0.10660053
## 9  0.2652575 0.2341318            0.10895961
## 10 0.2681185 0.2346518            0.11001010
## 11 0.2550477 0.2401184            0.11420505
## 12 0.2518506 0.2435057            0.11970295
##
## [12 rows x 6 columns]
```

```
# predict the test set
predictions <- h2o.predict(best_model, test_h2o)
```

```
##   |                                                                              |
```

```
# convert predictions to a dataframe
predictions_df <- as.data.frame(predictions)

# view the predictions
# print(predictions_df)
binary_predictions <- ifelse(predictions_df$predict >= 0.5, 1, 0)



# The best model with the roster is the StackedEnsemble_AllModels_1_AutoML_2_20240630_131132. The perfo
```

```
test_predictions <- test
test_predictions$predictions <- binary_predictions
```

```
# calculate for accuracy
test_predictions <- test_predictions %>%
  mutate(cm = case_when(
    O_win == 1 & predictions  == 1 ~ "TP",
    O_win == 0 & predictions == 1 ~ "FP",
    O_win == 1 & predictions == 0 ~ "FN",
    O_win == 0 & predictions == 0 ~ "TN"

  )
)

accuracy <- (sum(test_predictions$cm == "TP") + sum(test_predictions$cm == "TN"))/nrow(test_predictions)
print(paste("Accuracy: ", accuracy))
```

```
## [1] "Accuracy:  0.943089430894309"
```

```
sensitivity <- (sum(test_predictions$cm == "TP")) / ((sum(test_predictions$cm == "TP") + sum(test_predi
print(paste("Sensitivity: ", sensitivity))
```

```
## [1] "Sensitivity:  0.933035714285714"
```

```
specificity <- (sum(test_predictions$cm == "TN")) / (sum(test_predictions$cm == "TN") + sum(test_predic
print(paste("Specificity: ", specificity))
```

```
## [1] "Specificity:  0.958620689655172"
```

```
precision = sum(test_predictions$cm == "TP") / (sum(test_predictions$cm == "TP") + sum(test_predictions$
print(paste("Precision: ", precision))
```

```
## [1] "Precision:  0.972093023255814"
```

```
recall <- sum(test_predictions$cm == "TP") / (sum(test_predictions$cm == "TP") + sum(test_predictions$cm
print(paste("Recall: ", recall))
```

```
## [1] "Recall:  0.933035714285714"
```

```
F1 = 2*precision*recall / (precision + recall)
print(paste("F1: ", F1))
```

```
## [1] "F1:  0.952164009111617"
```

```
# The performance metrics of this model is a little better than model 2, but it includes roster informa
```

```
# check for nba rankings after making predictions
test_predictions <- test_predictions %>%
  select(c(-O_win, -cm)) %>%
  rename(O_win = predictions)

# append training and testing set
check <- rbind(train, test_predictions)
```

```
# check nba rankings

check_rankings_eastern_O <- check %>%
  filter(O_west_conference == 0) %>%
  group_by(off_team) %>%
  summarise(total_wins = sum(O_win))%>%
  arrange(desc(total_wins))

check_rankings_eastern_D <- check %>%
  filter(D_west_conference == 0) %>%  # Filter for Eastern Conference defensive teams
  group_by(def_team) %>%  # Group by defensive team
  summarise(total_wins = sum(case_when(
    O_win == 1 ~ 0,  # If offensive team wins, count as 0
    O_win == 0 ~ 1   # If offensive team loses, count as 1
  ))) %>%  # Summarise total wins
  arrange(desc(total_wins))

# merge offense and defense stats
```

```r
rankings_eastern <- merge(
  check_rankings_eastern_O,
  check_rankings_eastern_D,
  by.x = "off_team",
  by.y = "def_team", all= TRUE)

rankings_eastern <- rankings_eastern %>%
  mutate(total_wins = total_wins.x + total_wins.y) %>%
  select(off_team, total_wins) %>%
  arrange(desc(total_wins))

check_rankings_eastern <- rankings_eastern %>%
  mutate(
    actual_ranking = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15
    )
  )
print(check_rankings_eastern)
```

```
##    off_team total_wins actual_ranking
## 1       BOS         64              1
## 2       NYK         51              2
## 3       MIL         50              3
## 4       CLE         48              4
## 5       ORL         47              5
## 6       IND         46              6
## 7       PHI         46              7
## 8       MIA         45              8
## 9       CHI         39              9
## 10      ATL         35             10
## 11      BKN         34             11
## 12      TOR         25             12
## 13      CHA         20             13
## 14      WAS         15             14
## 15      DET         14             15
```

```r
# With this model the eastern rankings are completely correct.
```

```r
check_rankings_western_O <- check %>%
  filter(O_west_conference == 1) %>%
  group_by(off_team) %>%
  summarise(total_wins = sum(O_win))%>%
  arrange(desc(total_wins))

check_rankings_western_D <- check %>%
  filter(D_west_conference == 1) %>%  # Filter for Eastern Conference defensive teams
  group_by(def_team) %>%  # Group by defensive team
  summarise(total_wins = sum(case_when(
    O_win == 1 ~ 0,  # If offensive team wins, count as 0
    O_win == 0 ~ 1   # If offensive team loses, count as 1
  ))) %>%  # Summarise total wins
  arrange(desc(total_wins))

# merge offense and defense stats
rankings_western <- merge(
  check_rankings_western_O,
  check_rankings_western_D,
  by.x = "off_team",
  by.y = "def_team", all= TRUE)

rankings_western<- rankings_western %>%
  mutate(total_wins = total_wins.x + total_wins.y) %>%
  select(off_team, total_wins) %>%
  arrange(desc(total_wins))

check_rankings_western_O <- check %>%
  filter(O_west_conference == 1) %>%
  group_by(off_team) %>%
  summarise(total_wins = sum(O_win))%>%
  arrange(desc(total_wins))

check_rankings_western_D <- check %>%
  filter(D_west_conference == 1) %>%  # Filter for Eastern Conference defensive teams
  group_by(def_team) %>%  # Group by defensive team
  summarise(total_wins = sum(case_when(
    O_win == 1 ~ 0,  # If offensive team wins, count as 0
    O_win == 0 ~ 1   # If offensive team loses, count as 1
  ))) %>%  # Summarise total wins
  arrange(desc(total_wins))

# merge offense and defense stats
rankings_western <- merge(
  check_rankings_western_O,
  check_rankings_western_D,
  by.x = "off_team",
  by.y = "def_team", all= TRUE)

rankings_western<- rankings_western %>%
  mutate(total_wins = total_wins.x + total_wins.y) %>%
  select(off_team, total_wins) %>%
  arrange(desc(total_wins))
```

```
check_rankings_western  <- rankings_western %>%
  mutate(
    actual_ranking = case_when(
      off_team %in% c("BOS", "OKC") ~ 1,
      off_team %in% c("NYK", "DEN") ~ 2,
      off_team %in% c("MIL", "MIN") ~ 3,
      off_team %in% c("CLE", "LAC") ~ 4,
      off_team %in% c("ORL", "DAL") ~ 5,
      off_team %in% c("IND", "PHX") ~ 6,
      off_team %in% c("PHI", "NOP") ~ 7,
      off_team %in% c("MIA", "LAL") ~ 8,
      off_team %in% c("CHI", "SAC") ~ 9,
      off_team %in% c("ATL", "GSW") ~ 10,
      off_team %in% c("BKN", "HOU") ~ 11,
      off_team %in% c("TOR", "UTA") ~ 12,
      off_team %in% c("CHA", "MEM") ~ 13,
      off_team %in% c("WAS", "SAS") ~ 14,
      off_team %in% c("DET", "POR") ~ 15
    )
  )

print(check_rankings_western)
```

```
##    off_team total_wins actual_ranking
## 1       DEN         57              2
## 2       MIN         57              3
## 3       OKC         57              1
## 4       DAL         51              5
## 5       LAC         51              4
## 6       NOP         50              7
## 7       PHX         48              6
## 8       LAL         47              8
## 9       GSW         46             10
## 10      SAC         45              9
## 11      HOU         43             11
## 12      UTA         31             12
## 13      MEM         27             13
## 14      SAS         23             14
## 15      POR         18             15
```

```
# With this model, rankings 6, 7, 9 and 10 are off, but the total_wins isn't that far off the rankings.
# The model was able to 100% rank the eastern conference well, but the western conference still has som
```

```
# At this moment, I've run out of time to learn how to create a synthetic dataset in R, so uniqueid_reg
library(openxlsx)
```

```
## Warning: package 'openxlsx' was built under R version 4.1.2
```

```
write.xlsx(model3_clean_df, "/Users/linanguyen/Desktop/OKC/model3.xlsx", rowNames = FALSE)
playoffs_round1 <- read_csv("/Users/linanguyen/Desktop/OKC/Generated Data/round1.csv")
```

```
## Rows: 56 Columns: 39
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): off_team, def_team
## dbl (37): nbagameid, O_rank_1, O_rank_2, O_rank_3, O_rank_4, O_rank_5, O_ran...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# use automl to retrieve a baseline model

# library(h2o)
# h2o.init()

# convert to h2o objects
train_h2o <- as.h2o(model3_clean_df)
```

```
##   |                                                                              |
```

```
test_h2o <- as.h2o(playoffs_round1)
```

```
##   |                                                                              |
```

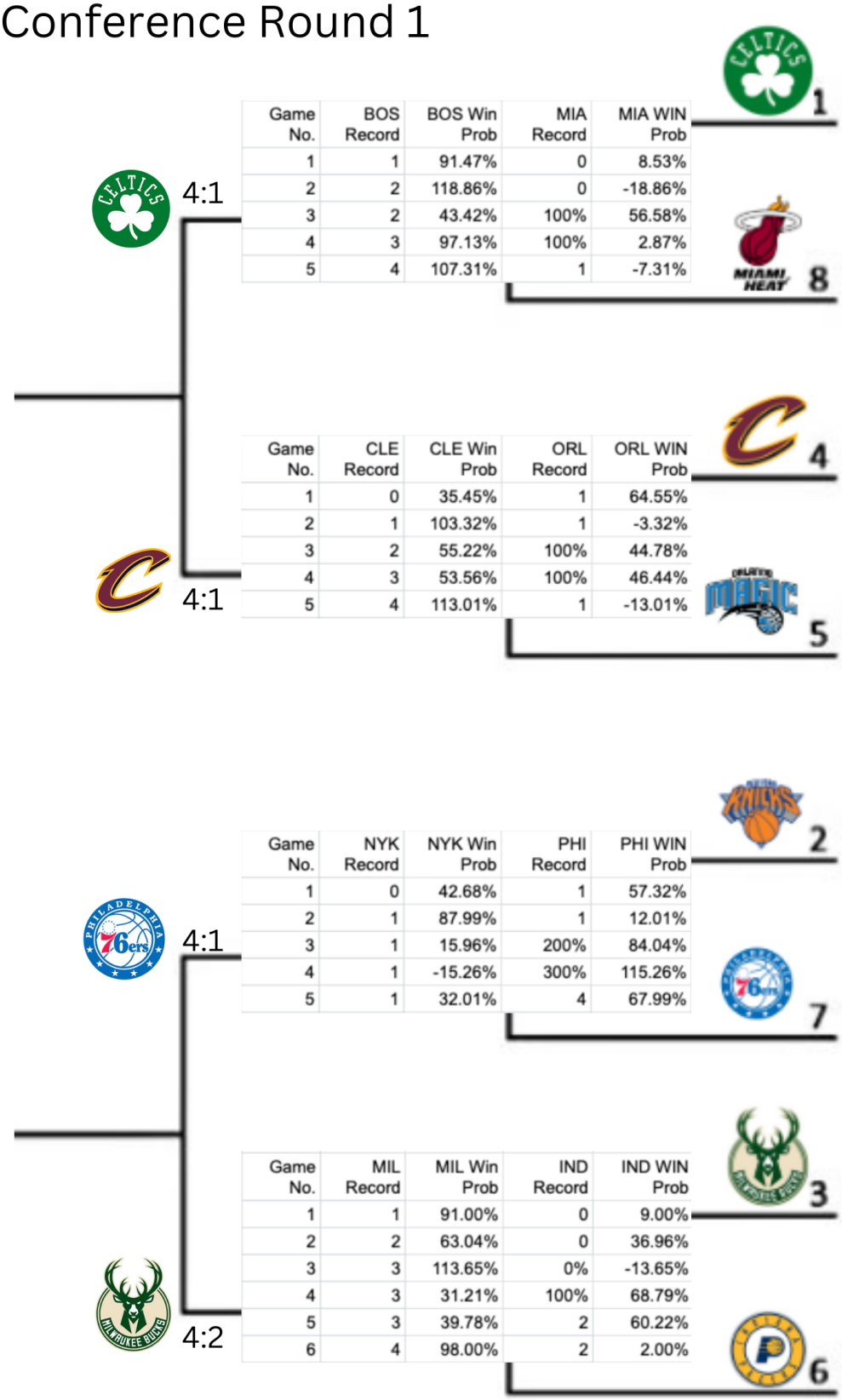
```
#target <- "O_win"
#features <- setdiff(names(train), target)

# train h2o model
#model <- h2o.automl(
#  x = features,
#  y = target,
#  training_frame = train_h2o,
#  max_models = 10,
#  seed = 1
#)

#best_model <- model@leader
#lb <- model@leaderboard
#print(lb, n = nrow(lb))

# predict the test set
predictions <- h2o.predict(best_model, test_h2o)
```

```
##   |                                                                              |
```

```
# convert predictions to a dataframe
predictions_df <- as.data.frame(predictions)

# view the predictions
# print(predictions_df)
binary_predictions <- ifelse(predictions_df$predict >= 0.5, 1, 0)

# h2o.shutdown(prompt = FALSE)
```

```
playoffs_round1$predictions <- binary_predictions
playoffs_round1$chances <- predictions_df
```

Eastern Conference Round 1



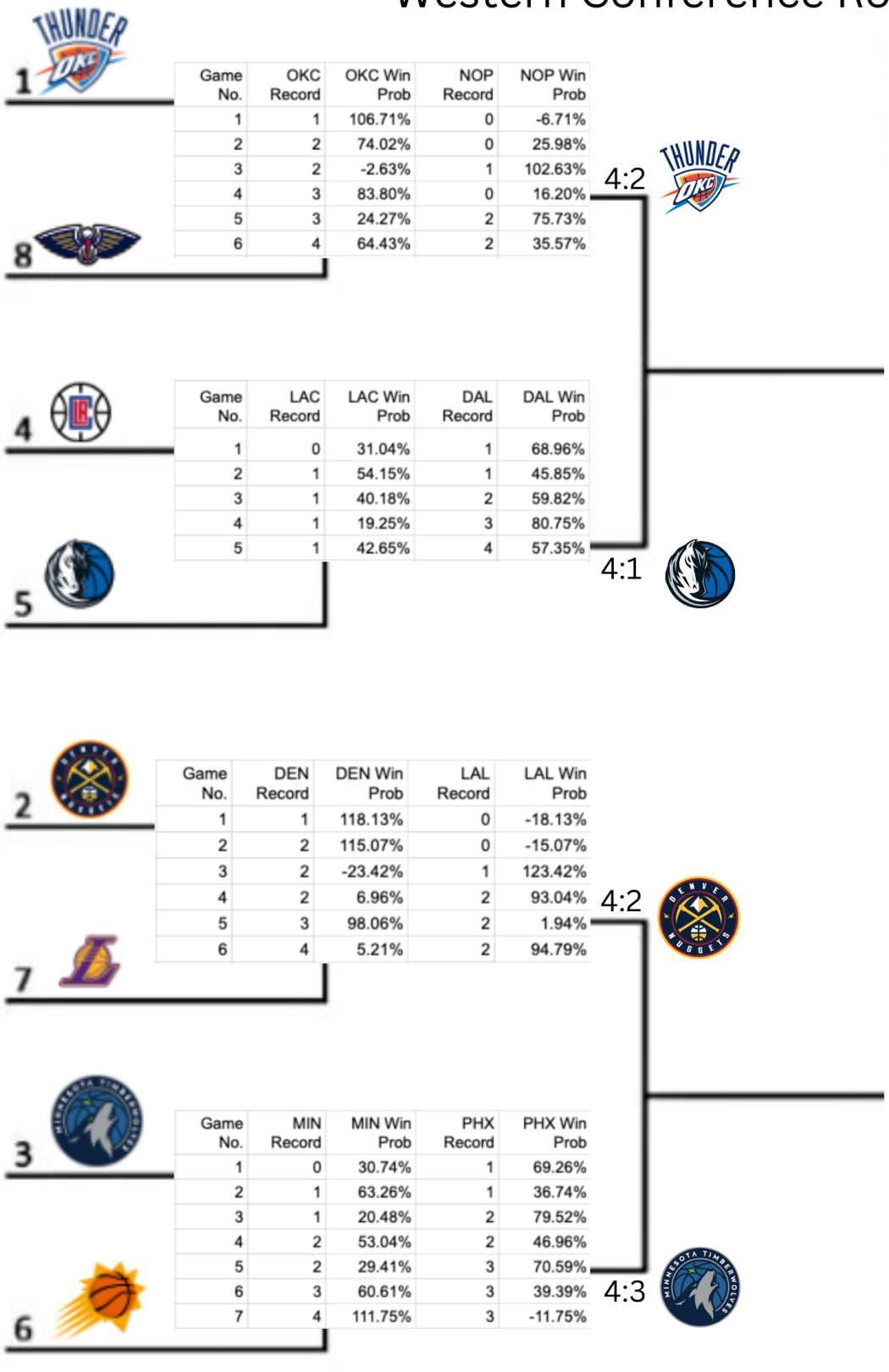
Game No.	BOS Record	BOS Win Prob	MIA Record	MIA WIN Prob
1	1	91.47%	0	8.53%
2	2	118.86%	0	-18.86%
3	2	43.42%	100%	56.58%
4	3	97.13%	100%	2.87%
5	4	107.31%	1	-7.31%

Game No.	CLE Record	CLE Win Prob	ORL Record	ORL WIN Prob
1	0	35.45%	1	64.55%
2	1	103.32%	1	-3.32%
3	2	55.22%	100%	44.78%
4	3	53.56%	100%	46.44%
5	4	113.01%	1	-13.01%

Game No.	NYK Record	NYK Win Prob	PHI Record	PHI WIN Prob
1	0	42.68%	1	57.32%
2	1	87.99%	1	12.01%
3	1	15.96%	200%	84.04%
4	1	-15.26%	300%	115.26%
5	1	32.01%	4	67.99%

Game No.	MIL Record	MIL Win Prob	IND Record	IND WIN Prob
1	1	91.00%	0	9.00%
2	2	63.04%	0	36.96%
3	3	113.65%	0%	-13.65%
4	3	31.21%	100%	68.79%
5	3	39.78%	2	60.22%
6	4	98.00%	2	2.00%

Western Conference Round 1



Game No.	OKC Record	OKC Win Prob	NOP Record	NOP Win Prob
1	1	106.71%	0	-6.71%
2	2	74.02%	0	25.98%
3	2	-2.63%	1	102.63%
4	3	83.80%	0	16.20%
5	3	24.27%	2	75.73%
6	4	64.43%	2	35.57%

Game No.	LAC Record	LAC Win Prob	DAL Record	DAL Win Prob
1	0	31.04%	1	68.96%
2	1	54.15%	1	45.85%
3	1	40.18%	2	59.82%
4	1	19.25%	3	80.75%
5	1	42.65%	4	57.35%

Game No.	DEN Record	DEN Win Prob	LAL Record	LAL Win Prob
1	1	118.13%	0	-18.13%
2	2	115.07%	0	-15.07%
3	2	-23.42%	1	123.42%
4	2	6.96%	2	93.04%
5	3	98.06%	2	1.94%
6	4	5.21%	2	94.79%

Game No.	MIN Record	MIN Win Prob	PHX Record	PHX Win Prob
1	0	30.74%	1	69.26%
2	1	63.26%	1	36.74%
3	1	20.48%	2	79.52%
4	2	53.04%	2	46.96%
5	2	29.41%	3	70.59%
6	3	60.61%	3	39.39%
7	4	111.75%	3	-11.75%

```
# Besides the amount of wins and losses being off, Pheonix has made it to round 2 of play offs even tho
```

```
# round 2
# making a new synthetic dataset
playoffs_round2 <- read_csv("/Users/linanguyen/Desktop/OKC/Generated Data/round2.csv")
```

```
## Rows: 28 Columns: 39
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): off_team, def_team
## dbl (37): nbagameid, O_rank_1, O_rank_2, O_rank_3, O_rank_4, O_rank_5, O_ran...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# use automl to retrieve a baseline model

#library(h2o)
#h2o.init()

# convert to h2o objects
train_h2o <- as.h2o(model3_clean_df)
```

```
##   |                                                                              |
```

```
test_h2o <- as.h2o(playoffs_round2)
```

```
##   |                                                                              |
```

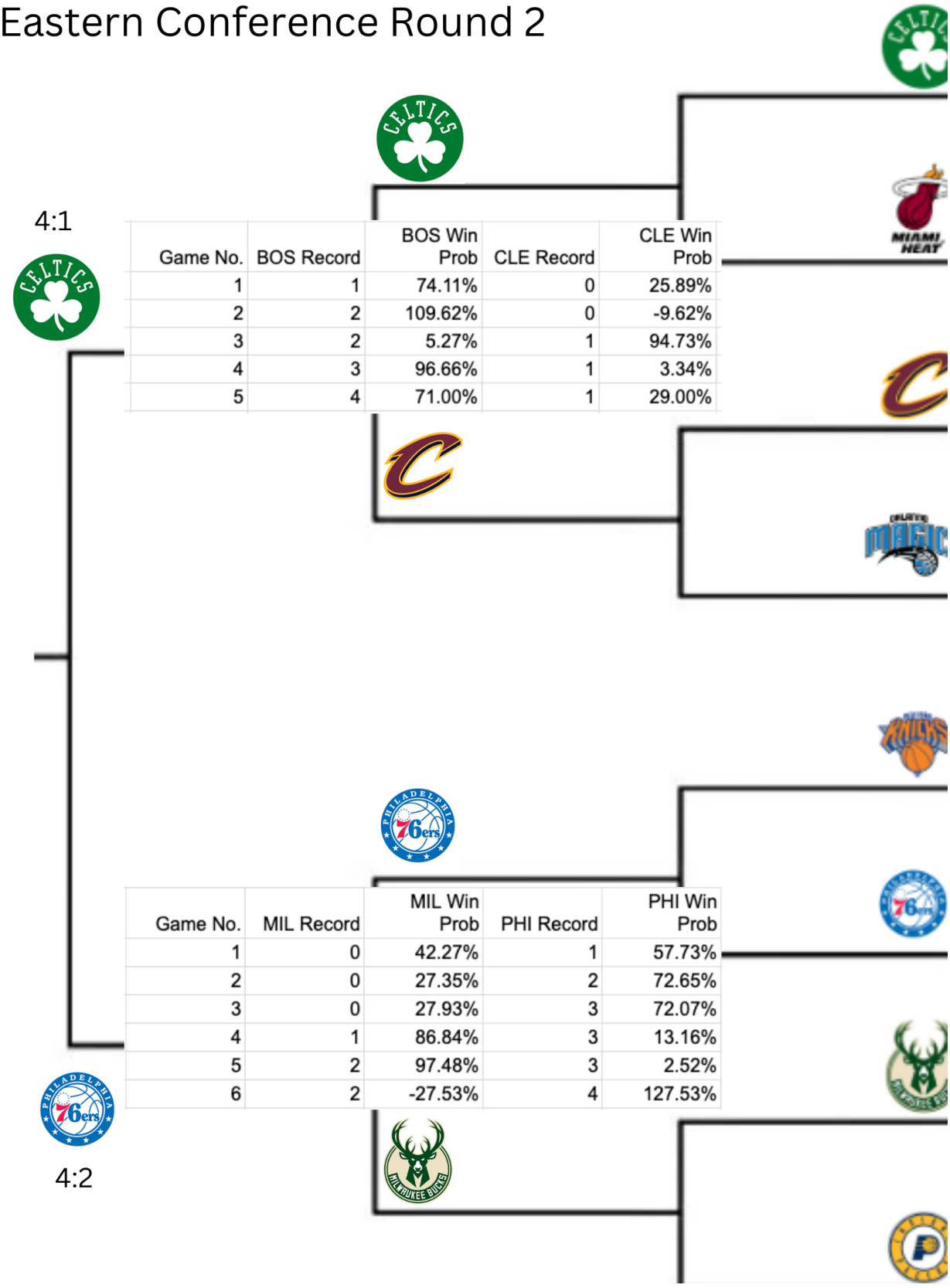
```
#target <- "O_win"
#features <- setdiff(names(train), target)

# train h2o model
#model <- h2o.automl(
#  x = features,
#  y = target,
#  training_frame = train_h2o,
#  max_models = 10,
#  seed = 1
#)

#best_model <- model@leader
#lb <- model@leaderboard
#print(lb, n = nrow(lb))

# predict the test set
predictions <- h2o.predict(best_model, test_h2o)
```

```
##   |                                                                              |
```

```r
# convert predictions to a dataframe
predictions_df <- as.data.frame(predictions)

# view the predictions
# print(predictions_df)
binary_predictions <- ifelse(predictions_df$predict >= 0.5, 1, 0)

# h2o.shutdown(prompt = FALSE)
```

```r
playoffs_round2$predictions <- binary_predictions
playoffs_round2$chances <- predictions_df
```

Eastern Conference Round 2



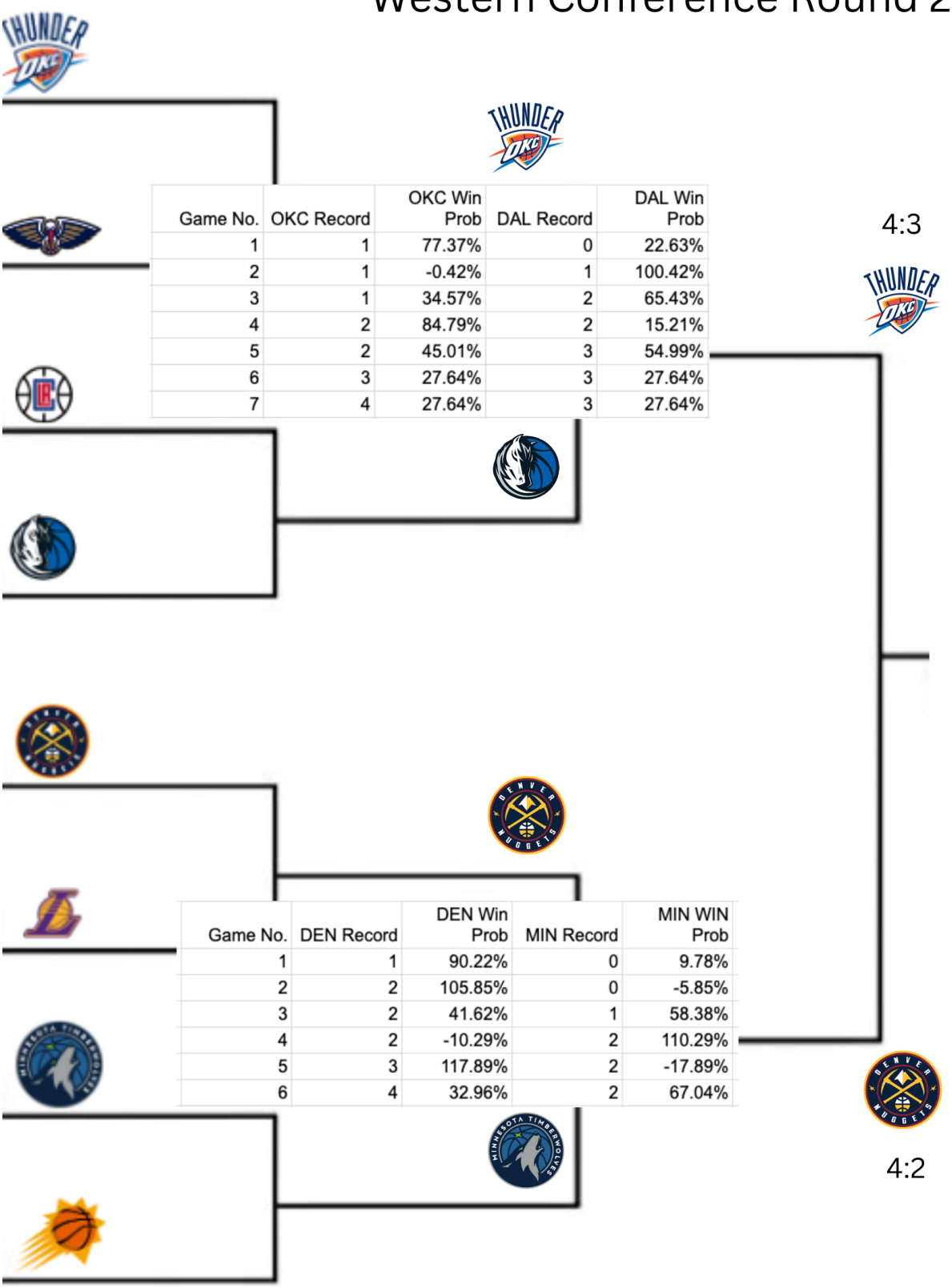
4:1

Game No.	BOS Record	BOS Win Prob	CLE Record	CLE Win Prob
1	1	74.11%	0	25.89%
2	2	109.62%	0	-9.62%
3	2	5.27%	1	94.73%
4	3	96.66%	1	3.34%
5	4	71.00%	1	29.00%

Game No.	MIL Record	MIL Win Prob	PHI Record	PHI Win Prob
1	0	42.27%	1	57.73%
2	0	27.35%	2	72.65%
3	0	27.93%	3	72.07%
4	1	86.84%	3	13.16%
5	2	97.48%	3	2.52%
6	2	-27.53%	4	127.53%

4:2

Western Conference Round 2



Game No.	OKC Record	OKC Win Prob	DAL Record	DAL Win Prob
1	1	77.37%	0	22.63%
2	1	-0.42%	1	100.42%
3	1	34.57%	2	65.43%
4	2	84.79%	2	15.21%
5	2	45.01%	3	54.99%
6	3	27.64%	3	27.64%
7	4	27.64%	3	27.64%

4:3

Game No.	DEN Record	DEN Win Prob	MIN Record	MIN WIN Prob
1	1	90.22%	0	9.78%
2	2	105.85%	0	-5.85%
3	2	41.62%	1	58.38%
4	2	-10.29%	2	110.29%
5	3	117.89%	2	-17.89%
6	4	32.96%	2	67.04%

4:2

```
# conference finals
# making a new synthetic dataset
playoffs_round3 <- read_csv("/Users/linanguyen/Desktop/OKC/Generated Data/Round3.csv")
```

```
## Rows: 14 Columns: 39
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): off_team, def_team
## dbl (37): nbagameid, O_rank_1, O_rank_2, O_rank_3, O_rank_4, O_rank_5, O_ran...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# use automl to retrieve a baseline model

#library(h2o)
#h2o.init()

# convert to h2o objects
train_h2o <- as.h2o(model3_clean_df)
```

```
##   |                                                                              |
```

```
test_h2o <- as.h2o(playoffs_round3)
```

```
##   |                                                                              |
```

```
#target <- "O_win"
#features <- setdiff(names(train), target)

# train h2o model
#model <- h2o.automl(
#  x = features,
#  y = target,
#  training_frame = train_h2o,
#  max_models = 10,
#  seed = 1
#)

#best_model <- model@leader
#lb <- model@leaderboard
#print(lb, n = nrow(lb))

# predict the test set
predictions <- h2o.predict(best_model, test_h2o)
```

```
##   |                                                                              |
```

```
# convert predictions to a dataframe
predictions_df <- as.data.frame(predictions)
```

```
# view the predictions
# print(predictions_df)
binary_predictions <- ifelse(predictions_df$predict >= 0.5, 1, 0)

#h2o.shutdown(prompt = FALSE)
```

```
playoffs_round3$predictions <- binary_predictions
playoffs_round3$chances <- predictions_df
```

```
# nba finals
# making a new synthetic dataset
playoffs_round4 <- read_csv("/Users/linanguyen/Desktop/OKC/Generated Data/Round4.csv")
```

```
## Rows: 7 Columns: 39
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (2): off_team, def_team
## dbl (37): nbagameid, O_rank_1, O_rank_2, O_rank_3, O_rank_4, O_rank_5, O_ran...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

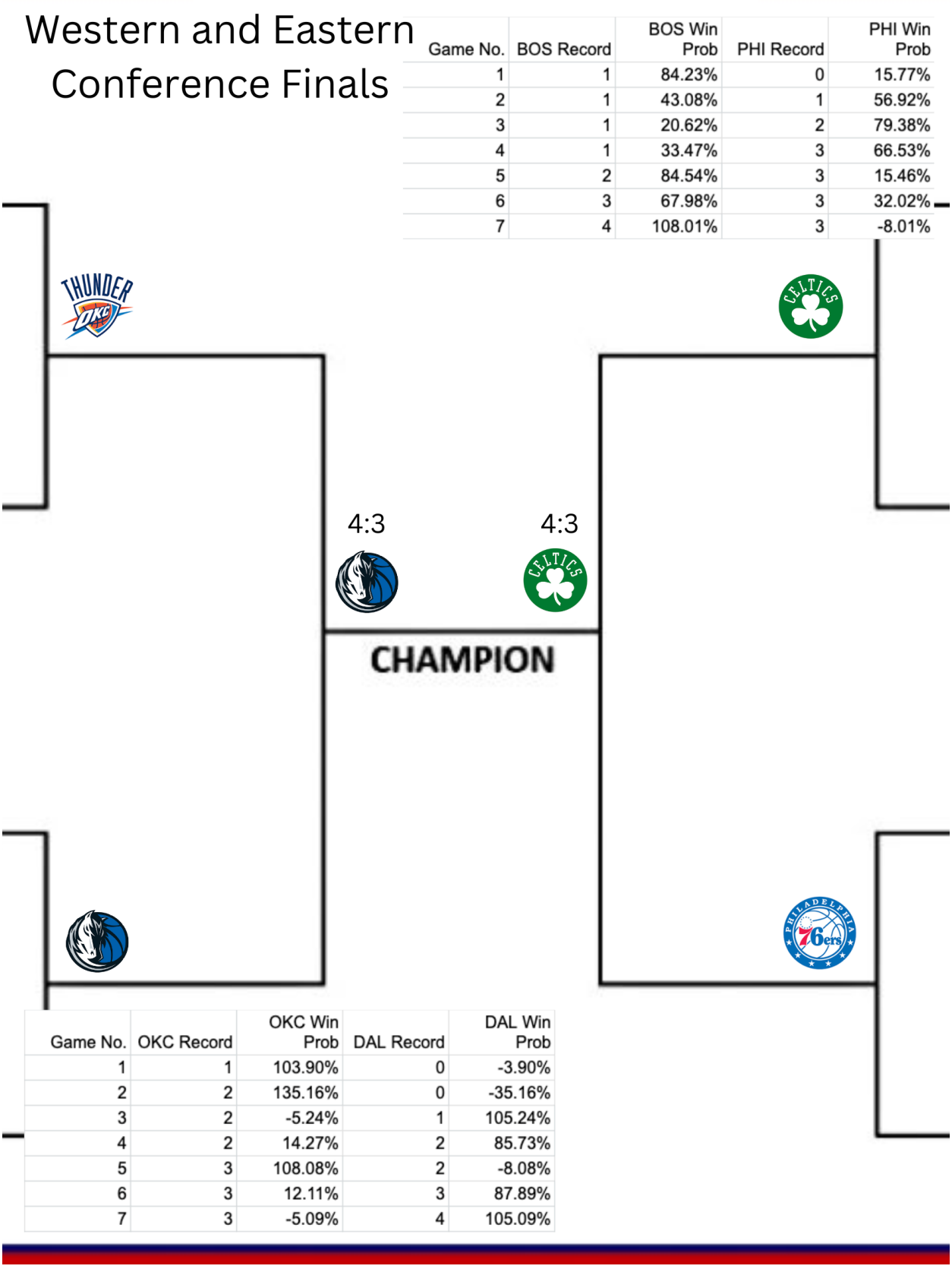
```
# use automl to retrieve a baseline model

#library(h2o)
#h2o.init()

# convert to h2o objects
train_h2o <- as.h2o(model3_clean_df)
```

```
##   |                                                                              |
```

```
test_h2o <- as.h2o(playoffs_round4)
```

```
##   |                                                                              |
```

```
#target <- "O_win"
#features <- setdiff(names(train), target)

# train h2o model
#model <- h2o.automl(
#  x = features,
#  y = target,
#  training_frame = train_h2o,
#  max_models = 10,
#  seed = 1
#)

#best_model <- model@leader
```



Game No.	BOS Record	BOS Win Prob	PHI Record	PHI Win Prob
1	1	84.23%	0	15.77%
2	1	43.08%	1	56.92%
3	1	20.62%	2	79.38%
4	1	33.47%	3	66.53%
5	2	84.54%	3	15.46%
6	3	67.98%	3	32.02%
7	4	108.01%	3	-8.01%

Game No.	OKC Record	OKC Win Prob	DAL Record	DAL Win Prob
1	1	103.90%	0	-3.90%
2	2	135.16%	0	-35.16%
3	2	-5.24%	1	105.24%
4	2	14.27%	2	85.73%
5	3	108.08%	2	-8.08%
6	3	12.11%	3	87.89%
7	3	-5.09%	4	105.09%

Figure 1: Conference Finals

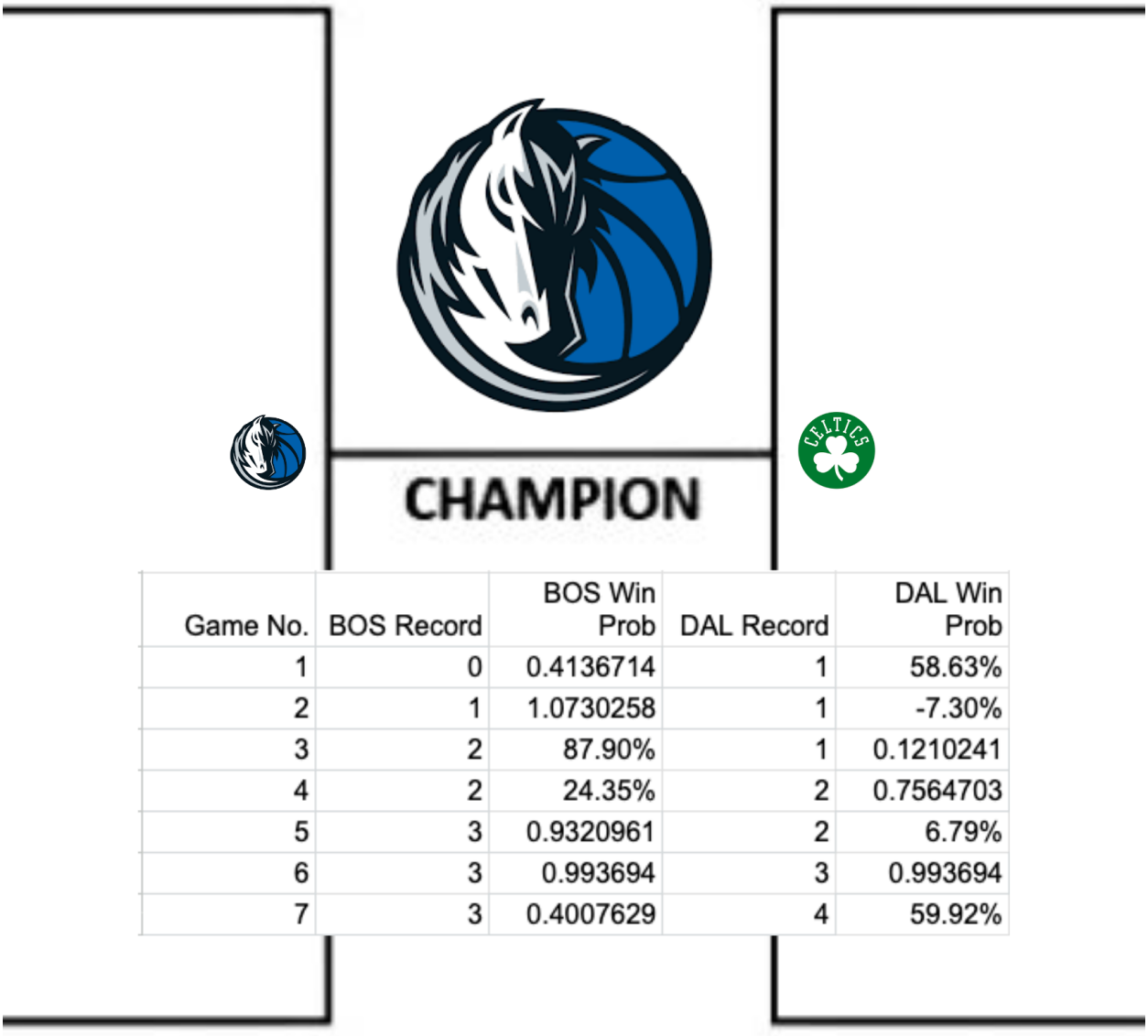
```
#lb <- model@leaderboard
#print(lb, n = nrow(lb))

# predict the test set
predictions <- h2o.predict(best_model, test_h2o)
```

```
##   |                                                                      |
```

```
# convert predictions to a dataframe
predictions_df <- as.data.frame(predictions)

# view the predictions
# print(predictions_df)
binary_predictions <- ifelse(predictions_df$predict >= 0.5, 1, 0)

h2o.shutdown(prompt = FALSE)
```

```
playoffs_round4$predictions <- binary_predictions
playoffs_round4$chances <- predictions_df
```

NBA Finals



Game No.	BOS Record	BOS Win Prob	DAL Record	DAL Win Prob
1	0	0.4136714	1	58.63%
2	1	1.0730258	1	-7.30%
3	2	87.90%	1	0.1210241
4	2	24.35%	2	0.7564703
5	3	0.9320961	2	6.79%
6	3	0.993694	3	0.993694
7	3	0.4007629	4	59.92%

ANSWER :

```
# The model used to predict the NBA play offs is a stacked ensemble model, which combines many differen
# If I had more time, I would go back and figure out how to create trained synthetic data that is more
```

```
# Findings
# Boston Celtics had bad luck in the 2022 and 2023 nba finals, but were successful in their regular sea
# Phoenix Suns in were successful in their conference in 2020 and 2021 regular season, but were unsucce
```