Mini Project 2

Brenden Bready

2024-02-25

Expected Runs

1. Calculating run expectancy

To calculate run expectancy, I sourced data from the baseballr package, which I then stored in a csv file. The data scraping details can be found in the appendix, and I will read in the data below.

```
# Reading in our play by play data, the data was scraped in a previous R script
pbp = read.csv("pbpdata2021.csv")
```

As a first step in the analysis, I will create a new column called "bases", which provides details for which baserunner situation is occurring during each pitch. 0 for no base runners, 1 for a runner on first, up to 123 for runners on first second third.

```
# Making cases for different situations to help with looping
pbp = pbp %>%
  mutate(bases = case_when(
    is.na(matchup.postOnFirst.fullName) & is.na(matchup.postOnSecond.fullName)
    & is.na(matchup.postOnThird.fullName) ~ 0,
    is.na(matchup.postOnSecond.fullName) & is.na(matchup.postOnThird.fullName) ~ 1,
    is.na(matchup.postOnFirst.fullName) & is.na(matchup.postOnThird.fullName) ~ 2,
    is.na(matchup.postOnFirst.fullName) & is.na(matchup.postOnSecond.fullName) ~ 3,
    is.na(matchup.postOnThird.fullName) ~ 12,
    is.na(matchup.postOnSecond.fullName) ~ 13,
    is.na(matchup.postOnFirst.fullName) ~ 23,
    TRUE ~ 123
  ))
```

Since I only need information about the start of the at bat and the final score at the end of the inning, I can filter the data to only include information about the first pitch.

```
# Filter data to only include first pitch
pitchone = pbp %>%
  filter(pitchNumber == 1)
```

I will need information about the scores at the end of the inning, so I will filter the data to only include information about the score at the end of each half inning.

```r
# Getting all of the last at bats of an inning (for final inning scores)
lastbats = pitchone %>%
  group_by(game_pk, about.inning, about.halfInning) %>%
  dplyr::slice(n())
```

In the next chunk, I will calculate my expected runs. I start by initializing a new data frame to store my results, and then loop through each out and base runner combination to calculate all of the expected runs. I do this by obtaining the scores at the start of each base runner and out combination, subtract the difference between the end of the inning score and the situation, and then divide by the number of occurrences. Final results are summarized in the table at the end.

```r
# Start by initializing a data frame to store results
expruns = data.frame(outs = integer(), bases = integer(), exruns = double())

# Set the loop to go through each out and base runner combination
for (i in unique(pitchone$count.outs.start)){
  for (j in unique(pitchone$bases)){

    # Filter to get current score at each baserunner/out combo
    scores = pitchone %>%
      filter(bases == j & count.outs.start == i)

    # Calculate expected runs by joining the last at bats with each of the scores
    # at the start of the inning, ungroup results, sum and divide by total number of
    # situations where that event occured
    exprun = right_join(lastbats,
                        scores,
                        by = c("game_pk", "game_date",
                               "about.inning", "about.halfInning")) %>%
      ungroup() %>%
      summarise(sum((result.homeScore.x + result.awayScore.x) -
                      (result.homeScore.y + result.awayScore.y))/n())

    # Store results in a data frame and join with others
    result = data.frame(outs = i, bases = j, exruns = round(exprun, 3))
    expruns = rbind(expruns, result)
  }
}
colnames(expruns)[3] <- "exruns"
expruns = expruns %>%
  arrange(outs, bases)

# Make the columns the outs and the rows the results
exprunsclean = pivot_wider(expruns, names_from = outs, values_from = exruns)
```

Table 1: Expected runs for out and base runner combinations

bases	0	1	2
0	0.567	0.341	0.095
1	0.680	0.390	0.129
2	0.780	0.464	0.164
3	0.833	0.443	0.130

bases	0	1	2
12	1.325	0.672	0.441
13	1.213	0.723	0.472
23	1.921	0.940	0.562
123	1.982	1.402	0.721

As a sanity check, the table above seems to make sense. As the base runners advance, the expected runs increase (down the columns), and as the number of outs increases, the expected runs decrease (across the rows). Our highest value is 1.98 expected runs when the bases are loaded with 0 outs, and our lowest expected runs are 0.1 when there are no base runners and 2 outs.

2. Runs Created

a.

To find the total number of runs created by a player, I started by adding a column to my dataset that indicated the current amount of expected runs going into each plate appearance (based on the values from the previous part). This join is seen in the chunk below.

```
# adding current expected runs as a column to dataset
pitchone = left_join(pitchone, expruns,
                             by = c("count.outs.start" = "outs", "bases" = "bases"))
```

From here, I calculated the total runs created by a player by adding the difference between the score at the start and end of their at bat with the difference of the expected runs at the start and end of their at bat. Let's consider an example. Say a player is going up to bat with 0 outs, and a base runner on on third. At the start of his at bat, the expected runs is 0.83. The batter then hits a single, and the base runner goes home and scores. The score increased by 1, and there is now a man on first with 0 outs (expected runs 0.68). Based on this, I would say the total runs created by the player in that play is 0.85 runs. The chunk below provides the code for getting the total runs created during each play of the game by each batter.

```
# Runs created during that at bat
pitchone = pitchone %>%
  group_by(game_pk, about.inning, about.halfInning) %>%
  mutate(runscreated = (lead(exruns) - exruns) +
           (lead(result.homeScore) - result.homeScore) +
           (lead(result.awayScore) - result.awayScore)) %>%
  ungroup() %>%
  mutate(runscreated = ifelse(is.na(runscreated), 0, runscreated))
```

From here, we can now sum the total runs created by each batter, and filter to only include information about players with over 100 at bats. The code for filtering can be found in the appendix, and a table to summarize the top 10 players can be seen below.

Table 2: Total Runs Created (Over 100 at bats)

Player	total_runs_created	atBats
Carlos Correa	46.218	586
Xander Bogaerts	43.887	569
Nicky Lopez	42.152	508

Player	total_runs_created	atBats
José Ramírez	39.487	575
Vladimir Guerrero Jr.	38.285	652
Yuli Gurriel	36.450	555
Bo Bichette	36.067	652
Jake Cronenworth	34.710	598
Yoán Moncada	33.111	579
Jesse Winker	28.063	501

b.

To find the top players by runs created per at bat, I can take the information from the above table, divide each total runs created by the number of at bats, and then sort again by the new average. A summary of the top 10 players can be seen in the table below.

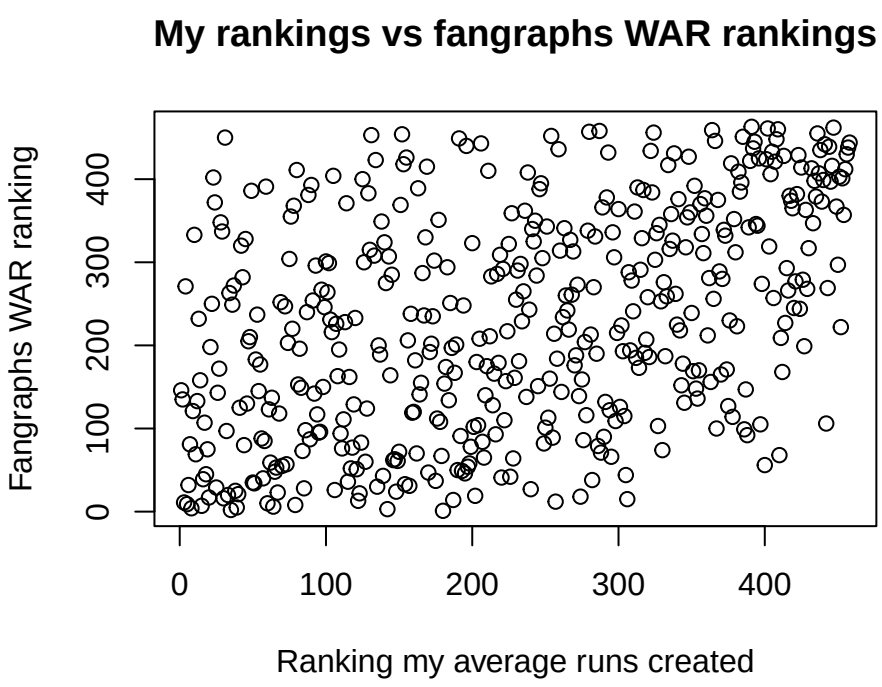
Table 3: Average Runs Created per At Bat (Over 100 at bats)

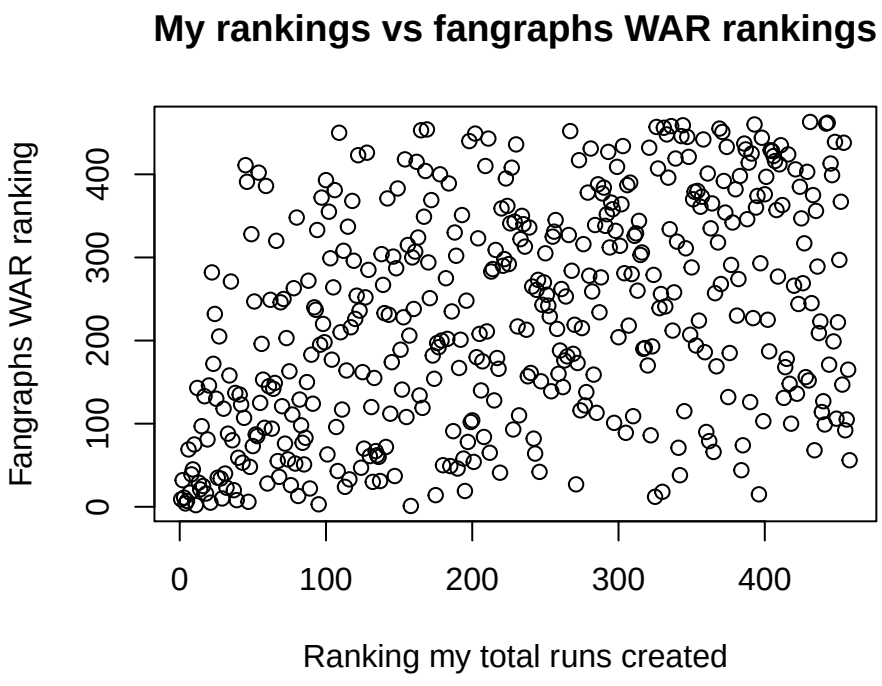
Player	average_runs_created	atBats
Evan Longoria	0.090	243
Frank Schwindel	0.083	198
Nicky Lopez	0.083	508
Matt Beaty	0.082	220
Carlos Correa	0.079	586
Xander Bogaerts	0.077	569
Darin Ruf	0.076	292
José Ramírez	0.069	575
Mike Trout	0.068	142
Seby Zavala	0.067	104

c.

The batting WAR on fangraphs and my rankings are quite different. The only players that appear in the top 10 of both of our rankings are Carlos Correa (9 for them, 5 for me), and Jose Ramirez (4 for them, 8 for me). Some players, such as Nicky Lopez, are close, but there are other major disagreements. My top player for average runs created, Evan Longoria, is ranked 146th on their rankings. Frank Schwindel is my number 2, and is ranked 135 for them. The total runs created has less major disparities, but it is biased because more at bats may lead to more runs created. In total runs created, the biggest disagreement is between Yuli Gurriel, ranked 6th for me and 69th for them. Less drastic than the average runs created metric, but still a very large difference.

To show just how different the rankings were, consider the following two graphs. These graphs plot my rankings against fangraphs rankings for the same player for both total runs created and average runs created. If my rankings and fangraphs rankings were identical, all the points would lie perfectly along the diagonal. This shows how different the rankings are, and how fangraphs uses a very different (and I'm sure more complex and accurate) approach into ranking their players.


My rankings vs fangraphs WAR rankings
Fangraphs WAR ranking
0
100
200
300
400
0
100
200
300
400
Ranking my average runs created


My rankings vs fangraphs WAR rankings
Fangraphs WAR ranking
0
100
200
300
400
0
100
200
300
400
Ranking my total runs created

Expected Goals

Details for the code cleaning of the shots data will be left out of the main report, but can be found in the appendix.

1. Building and testing xG model

a. My xG model

For my xG model, I will fit an xgBoost model to try and capture a large number of features in my data. I originally tried many logistic regression models, but many of them would output predictions less than 0, which makes no sense in the context of the problem.

Before fitting the model, I will create a new variable to indicate whether or not a player took a shot with their dominant foot. I am determining a player's dominant foot simply by looking at which foot they took more shots with. While this method is not perfect, it should work well for players who take many shots, because most players will try and shoot on their dominant foot when possible. Code for creating this variable can be seen below.

```
# Adding a new variable to encode a player's dominant foot
shots = shots %>%
  group_by(player.name) %>%
  mutate(dominantfoot = case_when(
    sum(shot.body_part.name == "Right Foot") > sum(shot.body_part.name == "Left Foot") ~ 1,
    sum(shot.body_part.name == "Right Foot") < sum(shot.body_part.name == "Left Foot") ~ 0,
    TRUE ~ NA
  )) %>%
  ungroup()

# Now add a variable to indicate if they shot with their preferred foot
shots = shots %>%
  mutate(shot.foot_preferred = case_when(
    (shot.body_part.name == "Right Foot") & (dominantfoot == 1) ~ 1,
    (shot.body_part.name == "Left Foot") & (dominantfoot == 0) ~ 1,
    TRUE ~ 0
  ))
```

Another caution I had before beginning my modeling was ensuring that my angle variable was set up correctly. I found the angle by using the arc sin, and then added 90 degrees to ensure that all of my angles were between 0 and 90 (0 being a straight on shot). To confirm the angle and distance calculation were correct, I filtered the data to a penalty kick, which should show a distance of 12 yards and angle of 0 degrees. As seen below, all angles are less than 2 degrees and the distance is very close to 12 (if not exactly 12), so this calculation is correct.

```
# Calculate distance and angle to goal
shots = shots %>%
  mutate(shot_distance = sqrt(((location_x - x_goal)^2 + (location_y - (y_goal))^2)),
         shot_angle = asin( (location_x - x_goal) / shot_distance)*(180/pi) + 90)

# Filter data to penalty to confirm angle and distance
angletab = head(shots %>%
  filter(shot.type.name == "Penalty") %>%
  select(shot.type.name, shot_distance, shot_angle))
```

```r
kable(angletab, caption = "Summary of distance and angles for penalties")
```

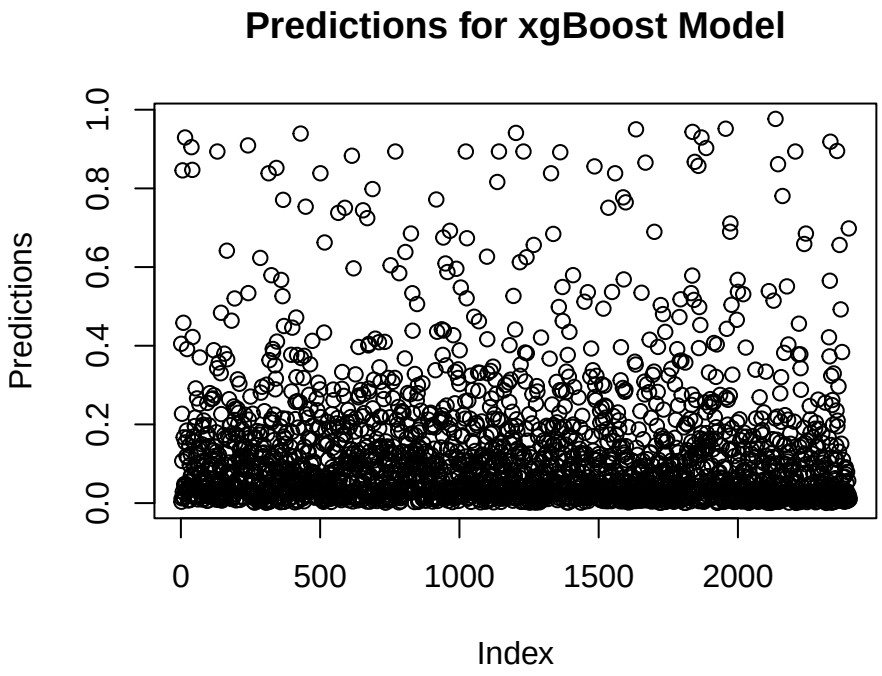
Table 4: Summary of distance and angles for penalties

shot.type.name	shot_distance	shot_angle
Penalty	11.90168	0.9628636
Penalty	12.00000	0.0000000
Penalty	11.80000	0.0000000
Penalty	11.80042	0.4855458
Penalty	11.80381	1.4563586
Penalty	12.10372	1.4202655

For my xgBoost model, I used an 80/20 train-test split to validate my data. I then fit the xgBoost model using the training data, with 10 different predictors as seen in the code below. Apart from distance and angle, the other 8 variables are all indicator variables based on the data provided. Predictions were calculated using the test set, and a plot of the predictions can be seen below.

```r
# Fitting xgboost with 80/20 train-test split
train_pct <- 0.8
train_data <- shots %>%
  sample_frac(train_pct)
test_data <- shots %>%
  anti_join(train_data %>% select(id), by = 'id')

trainformod <- xgb.DMatrix(data = as.matrix(train_data[, c("shot_distance",
                                                           "shot_angle",
                                                           "shot.foot_preferred",
                                                           "shot.first_time",
                                                           "shot.one_on_one",
                                                           "shot.openplay",
                                                           "shot.redirect",
                                                           "shot.freekick",
                                                           "shot.penalty",
                                                           "shot.foot",
                                                           "shot.head")]),
                           label = train_data$goalind)


# Set XGBoost parameters
params <- list(
  objective = "binary:logistic",  # Binary classification
  eval_metric = "logloss"          # Logarithmic loss (log-loss) as evaluation metric
)

# Train the XGBoost model
xg_model <- xgboost(data = trainformod,
                    params = params,
                    nrounds = 100,                # Number of boosting rounds (iterations)
                    verbose = FALSE)

testformod = xgb.DMatrix(data = as.matrix(test_data[, c("shot_distance",
```

```
                                                    "shot_angle",
                                                    "shot.foot_preferred",
                                                    "shot.first_time",
                                                    "shot.one_on_one",
                                                    "shot.openplay",
                                                    "shot.redirect",
                                                    "shot.freekick",
                                                    "shot.penalty",
                                                    "shot.foot",
                                                    "shot.head")]),
                              label = test_data$goalind)
xgpreds = predict(xg_model,testformod)
plot(xgpreds, ylab = "Predictions", main = "Predictions for xgBoost Model")
```



When looking at the MSE and MAE of this model, we can see that the model had an MSE of just 0.014 and MAE of 0.072. These results are summarized in the table below.

Table 5: MSE, MAE, and Bias based on 80/20 xgBoost Model

modName	MSE	MAE	bias
xgBoost	0.015	0.073	Unbiased

b. Testing existing class models

In class, we examined 6 different potential xG models. Four of the models were logistic regression models based on distance only, distance splines, distance * angle, and then distance * angle splines. The other two

models were random forests, one based on distance and angle, and the other based on 11 different predictors. I will leave the code for evaluating the models in the appendix, but below are two tables summarizing the results for a 50/50 train test split and an 80/20 train test split.

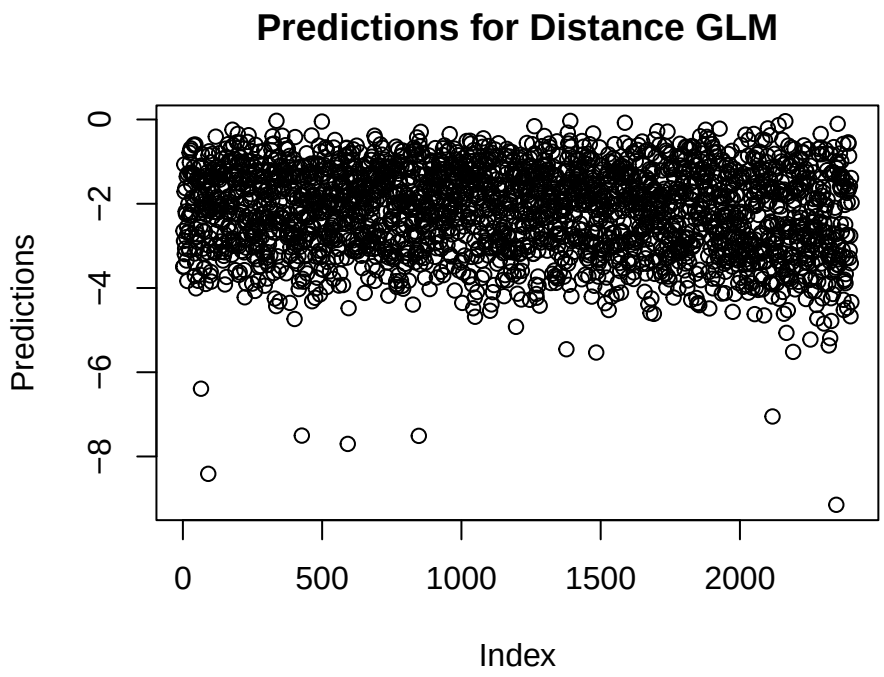
Table 6: MSE, MAE, and Bias based on 50/50 train-test split

modName	MSE	MAE	bias
Logistic Distance	6.284	2.301	Biased
Distance Spline	6.284	2.301	Biased
Angle*Distance	6.413	2.322	Biased
Distance*Angle spline	6.450	2.326	Biased
Distance, Angle RF	0.025	0.102	Unbiased
All vars RF	0.011	0.076	Unbiased

Table 7: MSE, MAE, and Bias based on 80/20 train-test split

modName	MSE	MAE	bias
Logistic Distance	6.259	2.306	Biased
Distance Spline	6.259	2.306	Biased
Angle*Distance	6.439	2.337	Biased
Distance*Angle spline	6.521	2.345	Biased
Distance, Angle RF	0.025	0.101	Unbiased
All vars RF	0.010	0.074	Unbiased
xgBoost	0.015	0.073	Unbiased

Based on the above results, we see that the random forest models significantly outperform all of the logistic regression models. Many of these models were extraordinarily biased, and when looking at the plot of predictions, the model was absolutely horrible. Consider the plot below, which shows the predictions for the 80/20 split logistic regression model using only distance as a predictor.



In the model, we see nearly all of our predictions are negative, which does not make any sense for xG. All xG values should be between 0 and 1, which also means getting MSE and MAE values greater than 1 is really horrible. Random forests and xgBoost solved this problem, only outputting values between 0 and 1. Because it had the lower MSE, I will use the random forest with all of the variables as my model for the next section.

2. Top Performing Players

To calculate the players best at scoring their opportunities, I apply predictions for the xG of each shot to all shots in our dataset. From here I will calculate the number of "Goals above xG", which is the total number of goals a player has scored minus the number of goals we expect them to score. If a player had a goals above xG value over 0, we would say they score more goals than expected. The code below shows how the calculation is applied in R (similar calculations for the next parts), and the table shows the top 10 players in goals above xG.

```
# Apply predictions from model over entire dataset
shots$rfpreds = predict(all_rf_mod, shots)

# Calculate goals above xG for all players
topPlayersOverall = shots %>%
  group_by(player.name) %>%
  summarise(goalAbovexG = round((sum(goalind) - sum(rfpreds)), 2)) %>%
  arrange(desc(goalAbovexG)) %>%
  mutate('Player Name' = player.name, 'Goals Above xG' = goalAbovexG) %>%
  select('Player Name', 'Goals Above xG')
```

Table 8: Top Players based on Goals above xG

Player Name	Goals Above xG
Lionel Andrés Messi Cuccittini	79.39
Luis Alberto Suárez Díaz	16.74
Samuel Eto"o Fils	8.61
Alexis Alejandro Sánchez Sánchez	7.21
Thierry Henry	5.68
David Villa Sánchez	5.58
Ivan Rakitić	4.97
Pedro Eliezer Rodríguez Ledesma	4.56
Ousmane Dembélé	4.47
Bojan Krkíc Pérez	3.28

Unsurprisingly, we see many familiar names at the top. Even watching soccer my entire life (and being a huge Messi fan), Lionel Messi is far beyond what I would have expected compared to the rest of the players. We see lots of other widely recognized top strikers such as Suarez and Eto'o at the top, with Ronaldo probably not having enough games in this dataset to make the top (only 9 goals in dataset).

When considering other situations such as free kicks, penalties, long shots, and short shots, we see some other names make our top 10. For each category Messi is still the top player, but there are many new names. For free kicks, we see players such as Ronaldinho and Neymar both have over 1 goals above xG, which is very impressive for free kick situations. The top 10 are summarized below.

Table 9: Top Players for Free Kicks

Player Name	Goals Above xG
Lionel Andrés Messi Cuccittini	4.45
Ronaldo de Assis Moreira	2.13
Neymar da Silva Santos Junior	1.24
José Javier Barkero Saludes	0.92
Rubén Castro Martín	0.90
Roberto Torres Morales	0.87
Abel Gómez Moreno	0.84
Joaquín Sánchez Rodríguez	0.84
Diego Mario Buonanotte Rende	0.83
Gabriel Appelt Pires	0.81

Penalties are less exciting, but we finally see Cristiano make the list. With a bit more data he might even be above Messi on this list, being widely regarded as an outstanding penalty kick taker. Of course, many players take no penalties in their career, so this list is definitely biased as to which names appear on it.

Table 10: Top Players for Penalties

Player Name	Goals Above xG
Lionel Andrés Messi Cuccittini	3.58
Ronaldo de Assis Moreira	1.43
Saúl Ñíguez Esclapez	0.67
Cristiano Ronaldo dos Santos Aveiro	0.50
Emilio José Viqueira Moure	0.48
Diego Forlán Corazo	0.37

Player Name	Goals Above xG
Marcos Antonio Senna da Silva	0.35
Ruud van Nistelrooy	0.34
Guilherme Magdalena Siqueira	0.33
Andoni Iraola Sagarna	0.27

Close range shots and long shots are again areas where Messi thrives, and here we get a full breakdown as to where he generates his extraordinarily high goals above xG. I considered short shots to be those shot closer than 18 yards, and long shots to be further than 18 yards. As the 18 yard mark is an arc, some of these shots will still be in the box, but for simplicity sake I will still consider them as long shots. Here we see our true number 9 strikers such as David Villa and Suarez go towards the top of the list in short shots, and some great wingers such as Alexis Sanchez and Christian Tello go towards the top of the long shots list. Being wingers, each of them probably score many curling shots from the sides, often beyond the 18 yard mark.

Table 11: Top Players for Short Shots

Player Name	Goals Above xG
Lionel Andrés Messi Cuccittini	59.35
Luis Alberto Suárez Díaz	15.48
David Villa Sánchez	6.80
Ivan Rakitić	6.11
Samuel Eto”o Fils	5.47
Ousmane Dembélé	4.69
Xavier Hernández Creus	4.45
Thierry Henry	4.35
Pedro Eliezer Rodríguez Ledesma	4.12
Bojan Krkíc Pérez	3.95

Table 12: Top Players for Long Shots

Player Name	Goals Above xG
Lionel Andrés Messi Cuccittini	20.04
Alexis Alejandro Sánchez Sánchez	4.26
Cristian Tello Herrera	4.04
Samuel Eto”o Fils	3.14
Sergio Leonel Agüero del Castillo	2.32
Diego Forlán Corazo	2.31
Anssumane Fati	2.05
Adriano Correia Claro	1.38
Thierry Henry	1.33
Luis Alberto Suárez Díaz	1.26

To conclude, I will look at one category where I would not expect Messi to be the top. When looking at headers, I'm very surprised to see Cesc Fabregas at the top. But behind him, we see our first defender make the list in Sergio Ramos. Most of the goals Ramos scores are headers off corners, and he is widely regarded as one of the best. This category might not give insight as to who to play at striker, but it is certainly useful to know that Ramos and Mathieu are two center backs any team would want to put in the box to attack corners with.

Table 13: Top Players for Headers

Player Name	Goals Above xG
Francesc Fàbregas i Soler	3.28
Sergio Ramos García	2.62
David Villa Sánchez	2.50
Luis Alberto Suárez Díaz	2.45
Jérémy Mathieu	1.64
Ronaldo de Assis Moreira	1.31
Marco Gastón Ruben Rodríguez	1.29
Willian José da Silva	1.23
Seydou Kéita	1.20
Thiago Alcântara do Nascimento	1.08

It's unsurprising to see Messi sweep the top, but this was definitely interesting to see other less familiar names pop up towards the top of some of the categories. There are many improvements that could still be made to the xG model to get even more accurate results, but this random forest model seems to capture the data fairly well when looking at the top charts. This was a really fun project!

Appendix

```
knitr::opts_chunk$set(echo = FALSE)
knitr::opts_chunk$set(fig.pos = "!h", fig.align = "center", fig.width = 5, fig.height = 4)
ggplot2::theme_set(ggplot2::theme_bw())
# Loading necessary libraries
library(baseballr)
library(tidyverse)
library(kableExtra)
library(knitr)
library(randomForest)
library(xgboost)
# Getting dates, removing July 13th
dates = (ymd("2021-04-01"):ymd("2021-10-03")) %>%
  as.Date(origin="1970-01-01")
dates = dates[-104]

# Joining all data
gamepks = data.frame()
for (i in 1:length(dates)){
  gamepks = bind_rows(gamepks, get_game_pks_mlb(dates[i], level_ids = 1))
}

# Getting full play by play
fullpbp = data.frame()
for (j in 1:length(gamepks$game_pk)){
  fullpbp = bind_rows(fullpbp, get_pbp_mlb(gamepks$game_pk[j]) %>%
    arrange(game_pk,
            about.inning,
            about.atBatIndex))
}

# select our relevant information, arrange in game order
pbpdata2021 = fullpbp %>%
  select(game_pk,
         game_date,
         about.inning,
         about.halfInning,
         about.atBatIndex,
         pitchNumber,
         last.pitch.of.ab,
         count.outs.start,
         result.homeScore,
         result.awayScore,
         matchup.postOnFirst.fullName, #missing if no one on this base
         matchup.postOnSecond.fullName,#missing if no one on this base
         matchup.postOnThird.fullName  #missing if no one on this base
         ) %>%
  arrange(game_pk,
          about.inning,
          about.atBatIndex)

# Write into a csv so it does not have to be run each time
```

```r
write.csv(pbpdata2021, "pbpdata2021.csv", row.names = FALSE)
# Reading in our play by play data, the data was scraped in a previous R script
pbp = read.csv("pbpdata2021.csv")
# Making cases for different situations to help with looping
pbp = pbp %>%
  mutate(bases = case_when(
    is.na(matchup.postOnFirst.fullName) & is.na(matchup.postOnSecond.fullName)
    & is.na(matchup.postOnThird.fullName) ~ 0,
    is.na(matchup.postOnSecond.fullName) & is.na(matchup.postOnThird.fullName) ~ 1,
    is.na(matchup.postOnFirst.fullName) & is.na(matchup.postOnThird.fullName) ~ 2,
    is.na(matchup.postOnFirst.fullName) & is.na(matchup.postOnSecond.fullName) ~ 3,
    is.na(matchup.postOnThird.fullName) ~ 12,
    is.na(matchup.postOnSecond.fullName) ~ 13,
    is.na(matchup.postOnFirst.fullName) ~ 23,
    TRUE ~ 123
  ))
# Filter data to only include first pitch
pitchone = pbp %>%
  filter(pitchNumber == 1)
# Getting all of the last at bats of an inning (for final inning scores)
lastbats = pitchone %>%
  group_by(game_pk, about.inning, about.halfInning) %>%
  dplyr::slice(n())
# Start by initializing a data frame to store results
expruns = data.frame(outs = integer(), bases = integer(), exruns = double())

# Set the loop to go through each out and base runner combination
for (i in unique(pitchone$count.outs.start)){
  for (j in unique(pitchone$bases)){

    # Filter to get current score at each baserunner/out combo
    scores = pitchone %>%
      filter(bases == j & count.outs.start == i)

    # Calculate expected runs by joining the last at bats with each of the scores
    # at the start of the inning, ungroup results, sum and divide by total number of
    # situations where that event occured
    exprun = right_join(lastbats,
                        scores,
                        by = c("game_pk", "game_date",
                               "about.inning", "about.halfInning")) %>%
      ungroup() %>%
      summarise(sum((result.homeScore.x + result.awayScore.x) -
                      (result.homeScore.y + result.awayScore.y))/n())

    # Store results in a data frame and join with others
    result = data.frame(outs = i, bases = j, exruns = round(exprun, 3))
    expruns = rbind(expruns, result)
  }
}
colnames(expruns)[3] <- "exruns"
expruns = expruns %>%
  arrange(outs, bases)
```

```
# Make the columns the outs and the rows the results
exprunsclean = pivot_wider(expruns, names_from = outs, values_from = exruns)
kable(exprunsclean, caption = "Expected runs for out and base runner combinations")
# adding current expected runs as a column to dataset
pitchone = left_join(pitchone, expruns,
                             by = c("count.outs.start" = "outs", "bases" = "bases"))
# Runs created during that at bat
pitchone = pitchone %>%
  group_by(game_pk, about.inning, about.halfInning) %>%
  mutate(runscreated = (lead(exruns) - exruns) +
           (lead(result.homeScore) - result.homeScore) +
           (lead(result.awayScore) - result.awayScore)) %>%
  ungroup() %>%
  mutate(runscreated = ifelse(is.na(runscreated), 0, runscreated))
totalrunscreated = pitchone %>%
  group_by(matchup.batter.fullName) %>%
  summarise(total_runs_created = round(sum(runscreated), 3), atBats = n()) %>%
  filter(atBats >= 100) %>%
  arrange(desc(total_runs_created)) %>%
  mutate(Player = matchup.batter.fullName) %>%
  select(Player, total_runs_created, atBats)

kable(head(totalrunscreated, 10), caption = "Total Runs Created (Over 100 at bats)")
# Average runs created per at bat
avgrunscreated = pitchone %>%
  group_by(matchup.batter.fullName) %>%
  summarise(average_runs_created = round(sum(runscreated)/n(), 3), atBats = n()) %>%
  filter(atBats >= 100) %>%
  arrange(desc(average_runs_created)) %>%
  mutate(Player = matchup.batter.fullName) %>%
  select(Player, average_runs_created, atBats)

kable(head(avgrunscreated, 10), caption = "Average Runs Created per At Bat (Over 100 at bats)")
fangraphs = read_csv("fangraphs.csv")
colnames(fangraphs)[1] <- "fgrank"
# Show difference between my rankings and fangraphs
avgrunscreated = avgrunscreated %>%
  mutate(myrank = row_number())

ranksjoined = left_join(avgrunscreated, fangraphs, by = c("Player" = "Name"))
plot(ranksjoined$myrank, ranksjoined$fgrank,
     xlab = "Ranking my average runs created",
     ylab = "Fangraphs WAR ranking",
     main = "My rankings vs fangraphs WAR rankings")
# Show difference between my rankings and fangraphs
totalrunscreated = totalrunscreated %>%
  mutate(myrank = row_number())

ranksjoined2 = left_join(totalrunscreated, fangraphs, by = c("Player" = "Name"))
plot(ranksjoined2$myrank, ranksjoined2$fgrank,
     xlab = "Ranking my total runs created",
     ylab = "Fangraphs WAR ranking",
     main = "My rankings vs fangraphs WAR rankings")
```

```r
# Reading in data
shots <- read_rds("la_liga_shots.rds")
# cleaning the data
shots <- shots %>%
  select(id, index, period, timestamp, minute, second, possession, duration,
         off_camera, location, possession_team.id, possession_team.name,
         play_pattern.id, play_pattern.name, team.id,
         team.name,
         player.id, player.name, position.id,
         position.name,
         starts_with("shot")) %>%
  select(-shot.freeze_frame) %>%
  as_tibble() %>%
  mutate(location = purrr::map(location, setNames, c("location_x","location_y"))) %>%
  unnest_wider(location)

#clean boolean data
shots = shots %>%
  mutate_at(vars(shot.first_time,
                 shot.one_on_one,
                 shot.redirect,
                 shot.aerial_won,
                 shot.open_goal,
                 shot.saved_to_post,
                 shot.deflected,
                 shot.saved_off_target,
                 shot.follows_dribble),
            ~ ifelse(!is.na(.) & . == TRUE, 1, 0)
  ) %>%
  mutate(shot.foot_head = case_when(
    str_detect(str_to_lower(shot.body_part.name), "foot") ~ "foot",
    str_detect(str_to_lower(shot.body_part.name), "head") ~ "head",
    TRUE ~ "other"),
    goalind = ifelse(shot.outcome.name == "Goal", 1, 0)

  )
# Creating an indicator to see if the outcome was a goal or not
# This will be useful for logistic regression
shots = shots %>%
  mutate(shot.isgoal = ifelse(shot.outcome.name == "Goal", 1, 0))
ymin <- 0 # minimum width
ymax <- 80 # maximum width
xmin <- 0 # minimum length
xmax <- 120 # maximum length

y_goal <- ymax/2
x_goal <- xmax
# Adding a new variable to encode a player's dominant foot
shots = shots %>%
  group_by(player.name) %>%
  mutate(dominantfoot = case_when(
    sum(shot.body_part.name == "Right Foot") > sum(shot.body_part.name == "Left Foot") ~ 1,
    sum(shot.body_part.name == "Right Foot") < sum(shot.body_part.name == "Left Foot") ~ 0,
```

```
    TRUE ~ NA
  )) %>%
  ungroup()

# Now add a variable to indicate if they shot with their preferred foot
shots = shots %>%
  mutate(shot.foot_preferred = case_when(
    (shot.body_part.name == "Right Foot") & (dominantfoot == 1) ~ 1,
    (shot.body_part.name == "Left Foot") & (dominantfoot == 0) ~ 1,
    TRUE ~ 0
  ))
# Calculate distance and angle to goal
shots = shots %>%
  mutate(shot_distance = sqrt(((location_x - x_goal)^2 + (location_y - (y_goal))^2)),
         shot_angle = asin( (location_x - x_goal) / shot_distance)*(180/pi) + 90)

# Filter data to penalty to confirm angle and distance
angletab = head(shots %>%
  filter(shot.type.name == "Penalty") %>%
  select(shot.type.name, shot_distance, shot_angle))

kable(angletab, caption = "Summary of distance and angles for penalties")
# Need to make all variables numeric (change shot type and foot or header)
shots = shots %>%
  mutate(shot.foot = ifelse(shot.foot_head == "foot", 1, 0),
         shot.head = ifelse(shot.foot_head == "head", 1, 0),
         shot.openplay = ifelse(shot.type.name == "Open Play", 1, 0),
         shot.freekick = ifelse(shot.type.name == "Free Kick", 1, 0),
         shot.penalty = ifelse(shot.type.name == "Penalty", 1, 0))
# Fitting xgboost with 80/20 train-test split
train_pct <- 0.8
train_data <- shots %>%
  sample_frac(train_pct)
test_data <- shots %>%
  anti_join(train_data %>% select(id), by = 'id')

trainformod <- xgb.DMatrix(data = as.matrix(train_data[, c("shot_distance",
                                                           "shot_angle",
                                                           "shot.foot_preferred",
                                                           "shot.first_time",
                                                           "shot.one_on_one",
                                                           "shot.openplay",
                                                           "shot.redirect",
                                                           "shot.freekick",
                                                           "shot.penalty",
                                                           "shot.foot",
                                                           "shot.head")]),
                           label = train_data$goalind)


# Set XGBoost parameters
params <- list(
  objective = "binary:logistic",  # Binary classification
```

```
  eval_metric = "logloss"          # Logarithmic loss (log-loss) as evaluation metric
)

# Train the XGBoost model
xg_model <- xgboost(data = trainformod,
                    params = params,
                    nrounds = 100,                # Number of boosting rounds (iterations)
                    verbose = FALSE)

testformod = xgb.DMatrix(data = as.matrix(test_data[, c("shot_distance",
                                                        "shot_angle",
                                                        "shot.foot_preferred",
                                                        "shot.first_time",
                                                        "shot.one_on_one",
                                                        "shot.openplay",
                                                        "shot.redirect",
                                                        "shot.freekick",
                                                        "shot.penalty",
                                                        "shot.foot",
                                                        "shot.head")]),
                         label = test_data$goalind)
xgpreds = predict(xg_model,testformod)
plot(xgpreds, ylab = "Predictions", main = "Predictions for xgBoost Model")
# Function for MSE
mse <- function(actual, preds){
  return (mean((actual - preds)^2))
}

# Function for MAE
mae <- function(actual, preds){
  return(mean(abs(actual - preds)))
}
mymse = round(mse(test_data$shot.statsbomb_xg, xgpreds),3)
mymae = round(mae(test_data$shot.statsbomb_xg, xgpreds),3)
myresults = data.frame(modName = "xgBoost", MSE = mymse,
                       MAE = mymae, bias = "Unbiased")
kable(myresults, caption = "MSE, MAE, and Bias based on 80/20 xgBoost Model")
# Evaluating class models: 50/50 train test split
set.seed(888)
train_pct <- 0.5
train_data <- shots %>%
  sample_frac(train_pct)
test_data <- shots %>%
  anti_join(train_data %>% select(id), by = 'id')
# Set up dataframe to store results
results5050 = data.frame(modName = character(), MSE = double(),
                         MAE = double(), bias = character())

# First model
distance_mod <- glm(goalind ~ shot_distance,
                    data = train_data,
                    family = binomial)
test_data$preds = predict(distance_mod, test_data)
```

```r
# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Logistic Distance", MSE = mse1, MAE = mae1,
                     bias = "Biased")
results5050 = rbind(results5050, result1)

# Second model
distance_mod_2 <- glm(goalind ~ splines::ns(shot_distance),
                    data = train_data,
                    family = binomial)
test_data$preds = predict(distance_mod_2, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Distance Spline", MSE = mse1, MAE = mae1,
                     bias = "Biased")
results5050 = rbind(results5050, result1)

# Third model
angle_mod <- glm(goalind ~ shot_distance*shot_angle,
                 data = train_data,
                 family = binomial)
test_data$preds = predict(angle_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Angle*Distance", MSE = mse1, MAE = mae1,
                     bias = "Biased")
results5050 = rbind(results5050, result1)

# Fourth model
angle_spline_mod <- glm(goalind ~ shot_distance*splines::ns(shot_angle, df = 3),
                        data = train_data,
                        family = binomial)
test_data$preds = predict(angle_spline_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Distance*Angle spline", MSE = mse1, MAE = mae1,
                     bias = "Biased")
results5050 = rbind(results5050, result1)

# Fifth model
angle_rf_mod <- randomForest(goalind ~ shot_distance + shot_angle,
                             data = train_data)
test_data$preds = predict(angle_rf_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
```

```
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Distance, Angle RF", MSE = mse1, MAE = mae1,
                     bias = "Unbiased")
results5050 = rbind(results5050, result1)

# Sixth model
all_rf_mod <- randomForest(goalind ~ shot_distance +
                             shot_angle +
                             shot.type.name +
                             shot.technique.name +
                             shot.foot_head +
                             shot.first_time +
                             shot.one_on_one +
                             shot.redirect +
                             shot.aerial_won +
                             shot.open_goal +
                             shot.follows_dribble
                           ,
                           data = train_data)
test_data$preds = predict(all_rf_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "All vars RF", MSE = mse1, MAE = mae1,
                     bias = "Unbiased")
results5050 = rbind(results5050, result1)
kable(results5050, caption = "MSE, MAE, and Bias based on 50/50 train-test split")
# Evaluating class models: 80/20 train test split
set.seed(888)
train_pct <- 0.8
train_data <- shots %>%
  sample_frac(train_pct)
test_data <- shots %>%
  anti_join(train_data %>% select(id), by = 'id')
# Set up dataframe to store results
results8020 = data.frame(modName = character(), MSE = double(),
                         MAE = double(), bias = character())

# First model
distance_mod <- glm(goalind ~ shot_distance,
                    data = train_data,
                    family = binomial)
test_data$preds = predict(distance_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Logistic Distance", MSE = mse1, MAE = mae1,
                     bias = "Biased")
results8020 = rbind(results8020, result1)

# Second model
```

```r
distance_mod_2 <- glm(goalind ~ splines::ns(shot_distance),
                      data = train_data,
                      family = binomial)
test_data$preds = predict(distance_mod_2, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Distance Spline", MSE = mse1, MAE = mae1,
                     bias = "Biased")
results8020 = rbind(results8020, result1)

# Third model
angle_mod <- glm(goalind ~ shot_distance*shot_angle,
                 data = train_data,
                 family = binomial)
test_data$preds = predict(angle_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Angle*Distance", MSE = mse1, MAE = mae1,
                     bias = "Biased")
results8020 = rbind(results8020, result1)

# Fourth model
angle_spline_mod <- glm(goalind ~ shot_distance*splines::ns(shot_angle, df = 3),
                        data = train_data,
                        family = binomial)
test_data$preds = predict(angle_spline_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Distance*Angle spline", MSE = mse1, MAE = mae1,
                     bias = "Biased")
results8020 = rbind(results8020, result1)

# Fifth model
angle_rf_mod <- randomForest(goalind ~ shot_distance + shot_angle,
                             data = train_data)
test_data$preds = predict(angle_rf_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "Distance, Angle RF", MSE = mse1, MAE = mae1,
                     bias = "Unbiased")
results8020 = rbind(results8020, result1)

# Sixth model
all_rf_mod <- randomForest(goalind ~ shot_distance +
                             shot_angle +
```

```
                              shot.type.name +
                              shot.technique.name +
                              shot.foot_head +
                              shot.first_time +
                              shot.one_on_one +
                              shot.redirect +
                              shot.aerial_won +
                              shot.open_goal +
                              shot.follows_dribble
                            ,
                            data = train_data)
test_data$preds = predict(all_rf_mod, test_data)

# MSE, MAE
mse1 = round(mse(test_data$shot.statsbomb_xg, test_data$preds), 3)
mae1 = round(mae(test_data$shot.statsbomb_xg, test_data$preds), 3)
result1 = data.frame(modName = "All vars RF", MSE = mse1, MAE = mae1,
                     bias = "Unbiased")
results8020 = rbind(results8020, result1)
results8020 = rbind(results8020, myresults)
kable(results8020, caption = "MSE, MAE, and Bias based on 80/20 train-test split")
test_data$preds = predict(distance_mod, test_data)
plot(test_data$preds, ylab = "Predictions", main = "Predictions for Distance GLM")
# Apply predictions from model over entire dataset
shots$rfpreds = predict(all_rf_mod, shots)

# Calculate goals above xG for all players
topPlayersOverall = shots %>%
  group_by(player.name) %>%
  summarise(goalAbovexG = round((sum(goalind) - sum(rfpreds)), 2)) %>%
  arrange(desc(goalAbovexG)) %>%
  mutate('Player Name' = player.name, 'Goals Above xG' = goalAbovexG) %>%
  select('Player Name', 'Goals Above xG')
kable(head(topPlayersOverall, 10), caption = "Top Players based on Goals above xG")
topPlayersfreekick = shots %>%
  filter(shot.freekick == 1) %>%
  group_by(player.name) %>%
  summarise(goalAbovexG = round((sum(goalind) - sum(rfpreds)), 2)) %>%
  arrange(desc(goalAbovexG)) %>%
  mutate('Player Name' = player.name, 'Goals Above xG' = goalAbovexG) %>%
  select('Player Name', 'Goals Above xG')

kable(head(topPlayersfreekick, 10), caption = "Top Players for Free Kicks")
topPlayerspenalties = shots %>%
  filter(shot.penalty == 1) %>%
  group_by(player.name) %>%
  summarise(goalAbovexG = round((sum(goalind) - sum(rfpreds)), 2)) %>%
  arrange(desc(goalAbovexG)) %>%
  mutate('Player Name' = player.name, 'Goals Above xG' = goalAbovexG) %>%
  select('Player Name', 'Goals Above xG')

kable(head(topPlayerspenalties, 10), caption = "Top Players for Penalties")
topPlayersshort = shots %>%
```

```
  filter(shot_distance < 18) %>%
  group_by(player.name) %>%
  summarise(goalAbovexG = round((sum(goalind) - sum(rfpreds)), 2)) %>%
  arrange(desc(goalAbovexG)) %>%
  mutate('Player Name' = player.name, 'Goals Above xG' = goalAbovexG) %>%
  select('Player Name', 'Goals Above xG')

kable(head(topPlayersshort, 10), caption = "Top Players for Short Shots")
topPlayerslong = shots %>%
  filter(shot_distance > 18) %>%
  group_by(player.name) %>%
  summarise(goalAbovexG = round((sum(goalind) - sum(rfpreds)), 2)) %>%
  arrange(desc(goalAbovexG)) %>%
  mutate('Player Name' = player.name, 'Goals Above xG' = goalAbovexG) %>%
  select('Player Name', 'Goals Above xG')

kable(head(topPlayerslong, 10), caption = "Top Players for Long Shots")
topPlayershead = shots %>%
  filter(shot.head == 1) %>%
  group_by(player.name) %>%
  summarise(goalAbovexG = round((sum(goalind) - sum(rfpreds)), 2)) %>%
  arrange(desc(goalAbovexG)) %>%
  mutate('Player Name' = player.name, 'Goals Above xG' = goalAbovexG) %>%
  select('Player Name', 'Goals Above xG')

kable(head(topPlayershead, 10), caption = "Top Players for Headers")
```