Atlanta United Project 1

Jake Federman

2025-03-23

Project 1: Predict the npxg + xA leaders for the 3/22-3/23 match weekend

Read in the data

```
matchlog <- read_csv(file = "atlutd_datascientist_project1_matchlog.csv")
```

```
## New names:
## Rows: 72168 Columns: 35
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (2): player_name, team_name dbl (32): ...1, match_id, player_id,
## competition_id, season_id, team_id, pl... date (1): birth_date
## 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.
## * `` -> `...1`
```

```
schedule <- read_csv(file = "atlutd_datascientist_project1_schedule.csv")
```

```
## New names:
## Rows: 2523 Columns: 11
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (1): match_status dbl (9): ...1, match_id, season_id, match_week,
## competition_stage_id, home_... date (1): match_date
## 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.
## * `` -> `...1`
```

Clustering Algorithm for Positions

I will construct a k-means clustering algorithm to try to tease out who is a goalkeeper, defender, midfielder, and forward. I believe the distributions of (npxg + xA)/90 will be normal within each position group, but each position likely has a different mean and variance. Of course, positions in soccer aren't as straightforward as the four main types, as there are attacking midfielders, defensive midfielders, and other players that play between positions. However, the algorithm should still give me clusters of players that have normal distributions.

The clustering algorithm will take into account the following variables: * Touches inside the box per 90 min (separates forwards) * Defensive actions per 90 min (should separate defenders, and maybe midfielders) * Forward passes / passes (separates goalkeepers and possibly defenders)

I thought about using shots, as they would increase with players that play further up. However, since I will eventually be predicting npxg (combined with xA) for a game, I will play it safe by not using variables that the metric directly depends on to prevent data leakage.

Using only these 3 variables separates the position groups surprisingly well. I originally tried many more variables, but performance improved as I took variables away.

```
#Create dataframe with stats to be used in clustering algorithm,
#Put the metrics on a per 90 minute basis
matchlog_cluster <-
  matchlog %>%
  filter(season_name %in% c(2024, 2025)) %>%
  group_by(player_id, player_name) %>%
  summarize(total_minutes = sum(player_match_minutes, na.rm = TRUE),
            box_touches_90 =
              (90*(sum(player_match_touches_inside_box, na.rm = TRUE)/total_minutes)),
            def_actions_90 =
              (90*(sum(player_match_defensive_actions, na.rm = TRUE)/total_minutes)),
            forward_pass_rate =
              sum(player_match_forward_passes, na.rm = TRUE)/
              sum(player_match_passes, na.rm = TRUE)) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'player_id'. You can override using the
## `.groups` argument.
```

```
#Standardize
standardize <- function(x) {
  mu <- mean(x, na.rm = TRUE)
  sigma <- sd(x, na.rm = TRUE)
  return( (x - mu)/sigma )
}

matchlog_cluster <-
  matchlog_cluster %>%
  mutate(z_box_touches_90 = standardize(box_touches_90),
         z_def_actions_90 = standardize(def_actions_90),
         z_forward_pass_rate = standardize(forward_pass_rate)) %>%
  select(-c(3:6)) %>%
  drop_na()
```

```
#K-means clustering
set.seed(1738)

clustering_input <-
  matchlog_cluster %>%
  select(-c(1:2))

position_clusters <-
  kmeans(clustering_input, centers = 4, nstart = 30)
```

```
matchlog_cluster$cluster <- position_clusters$cluster
matchlog_cluster <-
  matchlog_cluster %>%
  mutate(position = case_when(
    cluster == 1 ~ "GK",
    cluster == 2 ~ "MF",
    cluster == 3 ~ "FW",
    cluster == 4 ~ "DF"
  )) %>%
  select(-cluster)
```

Analyzing the clusters

```
position_clusters$centers %>%
  kable(format = "latex", booktabs = TRUE, digits = 4,
        caption = "Centers of Clusters",
        align = c("l", "c", "c", "c")) %>%
  kable_styling(position = "center", latex_options = "HOLD_position")
```

Table 1: Centers of Clusters

z_box_touches_90	z_def_actions_90	z_forward_pass_rate
-0.9975	-1.5943	2.2171
-0.3588	0.9165	-0.3547
1.3365	0.3299	-0.6889
-0.4955	-0.5445	0.2207

The first cluster contains players who never get the ball in the opponent's box, don't do defensive actions, and make most of their passes forward. This description matches that of a goalkeeper.

The second cluster, which corresponds to mostly midfielders, has players who perform lots of defensive actions, make some passes backwards, and infrequently receive the ball in the box.

The third cluster is defined by strikers, as they have lots of touches in the box.

The final cluster, which corresponds to defenders, has players that don't commit a ton of defensive actions, don't get the ball in the box, and make a decent amount of their passes forward.

These clusters largely align with general principles in soccer.

```
#Radar plots
position_avgs <-
  matchlog_cluster %>%
  group_by(position) %>%
  summarize(box_touches = mean(z_box_touches_90, na.rm = TRUE),
            def_actions = mean(z_def_actions_90, na.rm = TRUE),
            forward_pass_rate = mean(z_forward_pass_rate, na.rm = TRUE)
  ) %>%
  ungroup()

scaled_pos_avgs <-
```

```
  position_avgs %>%
  select(-position) %>%
  mutate(across(everything(), ~ (. - min(.)) / (max(.) - min(.)))) %>%
  as.data.frame()

radar_data <-
  rbind(
  rep(1, ncol(scaled_pos_avgs)),
  rep(0, ncol(scaled_pos_avgs)),
  scaled_pos_avgs
)

rownames(radar_data) <-
  c("max", "min", paste0(position_avgs$position))

par(mfrow = c(2, 2))
par(mar = c(1, 2, 2, 1))

for (i in 3:nrow(radar_data)) {
  radarchart(rbind(radar_data[1:2, ], radar_data[i, ]),
             pcol = i,
             pfcol = scales::alpha(i, 0.5),
             plwd = 2,
             title = rownames(radar_data)[i])
}
```

Setup

My ultimate goal is to do a hierarchical normal model with a Bayesian structure within each position group to make my predictions for this upcoming weekend's games. So, I need to check whether now that I have put the players into clusters according to their position groups, whether the distributions of (npxg + xA)/90 min are normal.

Note: I am not taking a player's overall npxg + xA and putting it on a per 90 minute basis. Instead, I am finding each player's npxg + xA per 90 minutes for each game they play, and taking the average of those. I am doing that because for the hierarchical Bayesian model, I will be treating each game like a unique observation, so the player means are the average of all their single game npxg + xA per 90 minutes values. I will check if this leads to a distribution that is different from what I would get if I was using their raw npxg + xA per 90 min numbers, but I don't anticipate this leading to any problems unless there is a trend of players in the league as a whole being more productive on a minute-by-minute basis when they play more/less. While some can argue that subs often perform better because they come into the game with fresh legs when everyone else is tired, they also have to take some time to adjust to the game, so I don't anticipate the effect of the way I'm handling the data to be particularly large.

```
#Join the positions back to matchlog
matchlog <-
  matchlog %>%
  filter(season_name %in% c(2024, 2025)) %>%
  inner_join(matchlog_cluster, by = c("player_id", "player_name"))
```

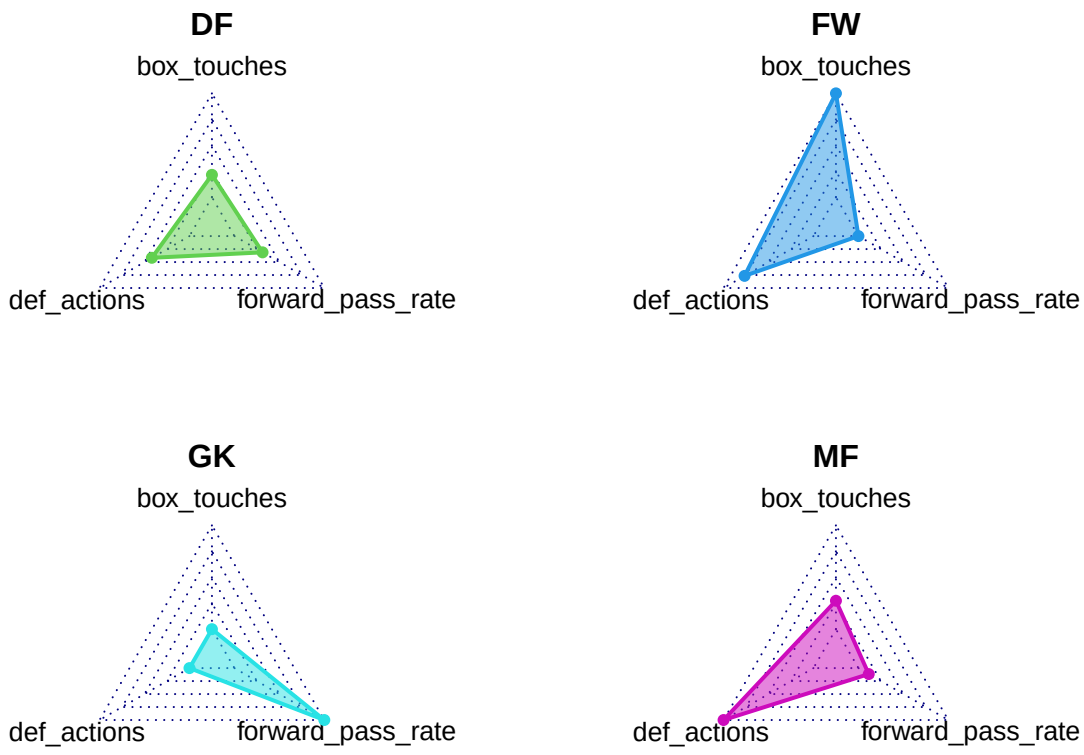
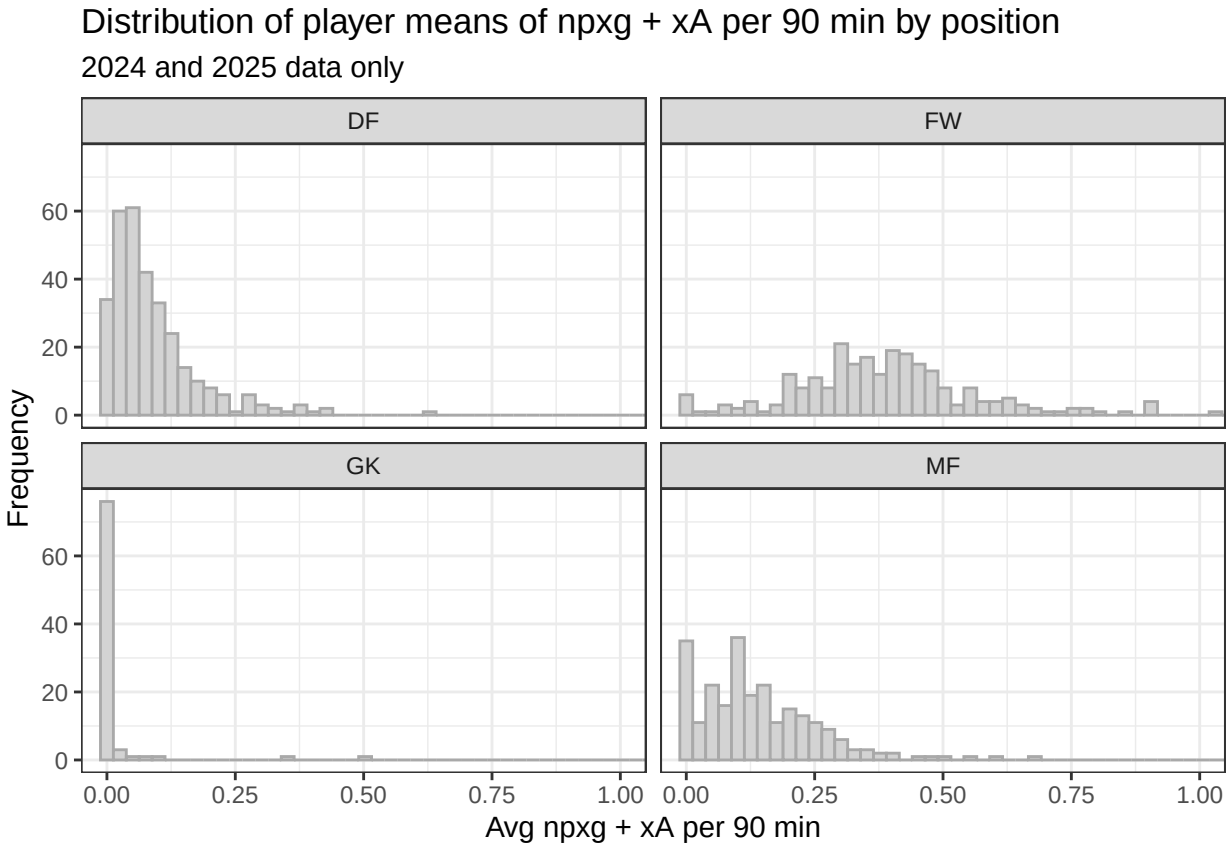


Figure 1: Radar plots displaying the play styles of each position using the variables in the clustering algorithm

```r
#Find (npxg + xA)/90
npxg_xa_90_player_means <-
  matchlog %>%
  mutate(npxg_xa_per_90 =
           90*((player_match_np_xg/player_match_minutes)
               + (player_match_xa/player_match_minutes))) %>%
  group_by(player_id, player_name, position) %>%
  summarize(total_matches = n(),
            total_minutes = sum(player_match_minutes),
            mean_npxg_xa_per_90 = mean(npxg_xa_per_90, na.rm = TRUE)) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'player_id', 'player_name'. You can
## override using the `.groups` argument.
```

```r
#graphing
ggplot(data = npxg_xa_90_player_means) +
  geom_histogram(mapping = aes(x = mean_npxg_xa_per_90),
                 color = "darkgrey", fill = "lightgrey", bins = 100) +
  facet_wrap(~position) +
  labs(
    x = "Avg npxg + xA per 90 min",
    y = "Frequency",
    title = "Distribution of player means of npxg + xA per 90 min by position",
    subtitle = "2024 and 2025 data only"
  ) +
  theme_bw() +
  coord_cartesian(xlim = c(0, 1))
```



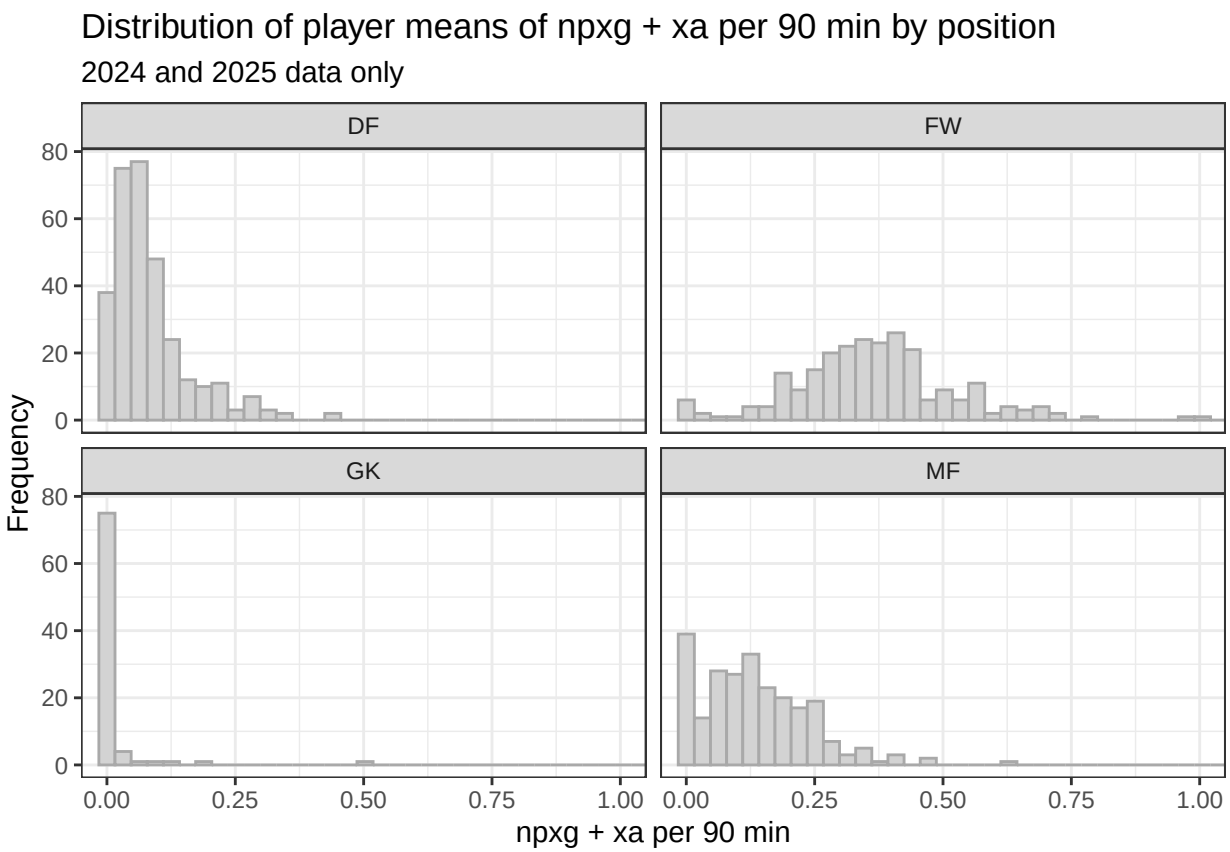
The distributions for forwards and midfielders are the only ones that are remotely close to normal distributions. For defenders, most players have less than 0.25 npxg + xA per 90 minutes on average, with a few reaching 0.25. Of course, pretty much all goalkeepers have 0 npxg + xA.

```
#Use the other definition to set up checks that my method
#doesn't yield a significantly different distribution
npxg_xa_90_player_means2 <-
  matchlog %>%
  group_by(player_id, player_name, position) %>%
  summarize(total_matches = n(),
            total_minutes = sum(player_match_minutes),
            npxg_xa_per_90 =
              90*((sum(player_match_np_xg) + sum(player_match_xa))/
                    sum(player_match_minutes))) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'player_id', 'player_name'. You can
## override using the `.groups` argument.
```

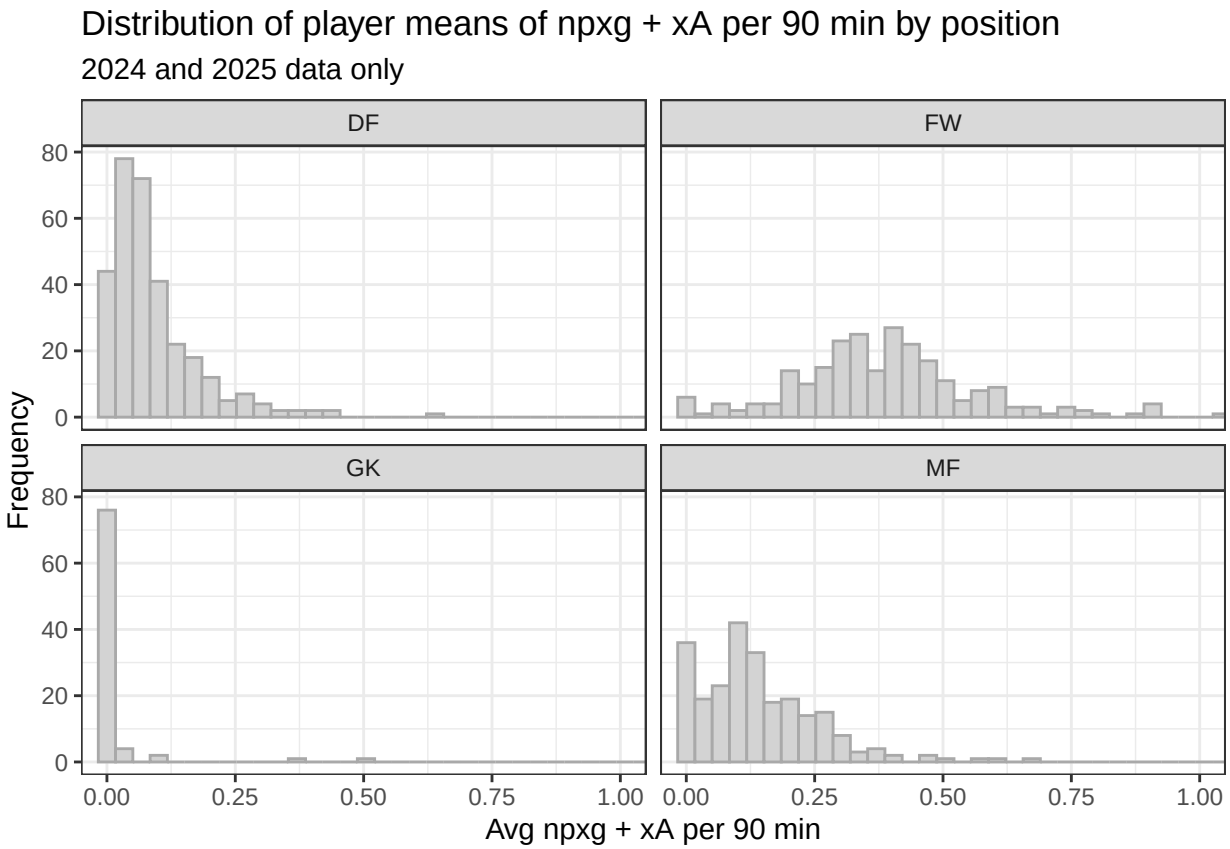
```
#graphing
ggplot(data = npxg_xa_90_player_means2) +
  geom_histogram(mapping = aes(x = npxg_xa_per_90),
                 color = "darkgrey", fill = "lightgrey", bins = 75) +
  facet_wrap(~position) +
  labs(
    x = "npxg + xa per 90 min",
```

```
    y = "Frequency",
    title = "Distribution of player means of npxg + xa per 90 min by position",
    subtitle = "2024 and 2025 data only"
  ) +
  theme_bw() +
  coord_cartesian(xlim = c(0, 1))
```



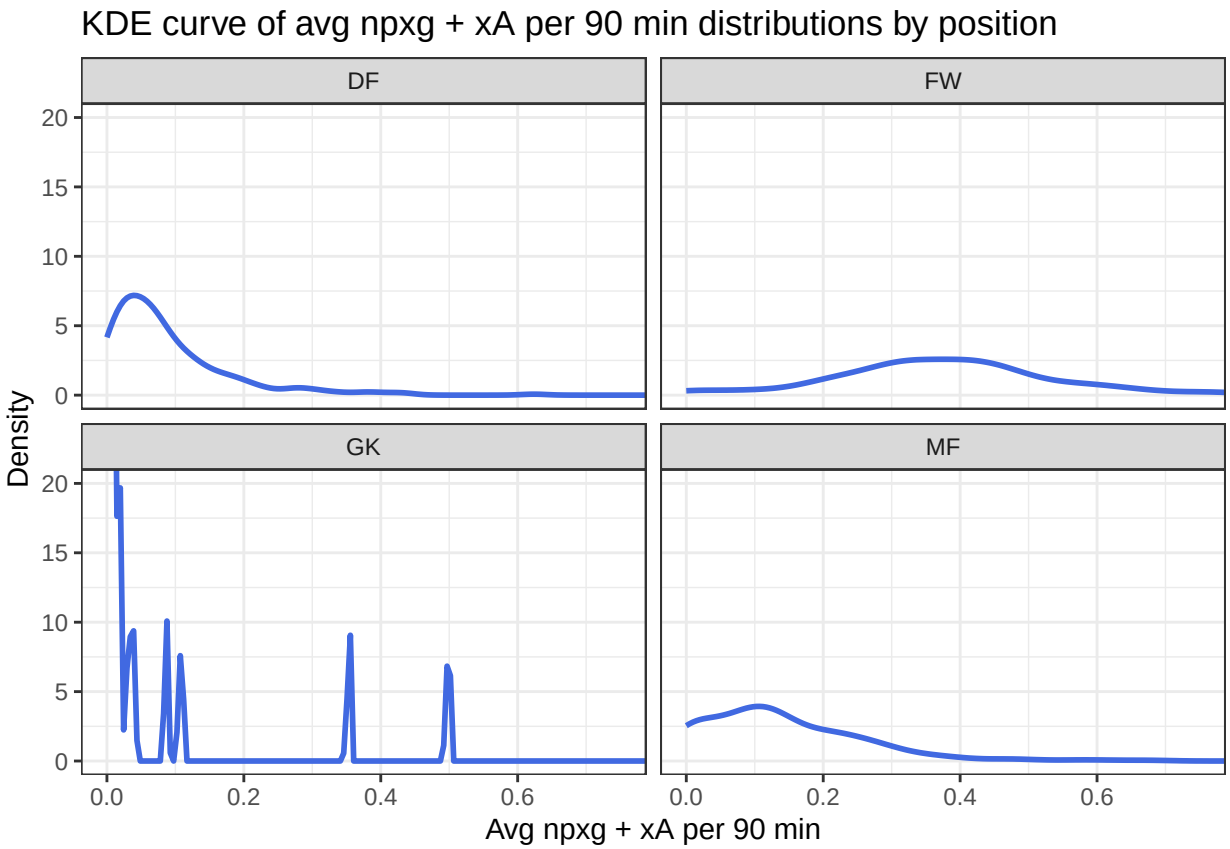
These distributions are basically the same thing. Due to time constraints, I will not be constructing a hypothesis test, but that is what I would do to ensure they are basically the same distribution.

```
ggplot(data = npxg_xa_90_player_means) +
  geom_histogram(mapping = aes(x = mean_npxg_xa_per_90),
                 color = "darkgrey", fill = "lightgrey", bins = 75) +
  facet_wrap(~position) +
  labs(
    x = "Avg npxg + xA per 90 min",
    y = "Frequency",
    title = "Distribution of player means of npxg + xA per 90 min by position",
    subtitle = "2024 and 2025 data only"
  ) +
  theme_bw() +
  coord_cartesian(xlim = c(0, 1))
```




```
ggplot(data = npxg_xa_90_player_means) +
  geom_density(mapping = aes(x = mean_npxg_xa_per_90),
               color = "royalblue", size = 1) +
  facet_wrap(~position) +
  labs(
    x = "Avg npxg + xA per 90 min",
    y = "Density",
    title = "KDE curve of avg npxg + xA per 90 min distributions by position"
  ) +
  coord_cartesian(xlim = c(0, 0.75), ylim = c(0, 20)) +
  theme_bw()
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



The KDE plot shows that the only distribution that is remotely close to normal is the forwards.

Assumption: For the rest of this project, for the sake of simplicity and time constraints, I will assume that the distribution for forwards is normal. I am not sure it is not normal, but I will use CLT to justify my claim.

It is interesting how for forwards their average npxg + xA /90 is greater than 0.3 for most games, and for everybody else, it pretty much never is. Let's verify that.

```
npxg_xa_90_player_means %>%
  group_by(position) %>%
  summarise(
    total_players = n(),
    above_0_35 = sum(mean_npxg_xa_per_90 > 0.35, na.rm = TRUE),
    prop_above = round(above_0_35/total_players, 3)
  ) %>%
  select(position, prop_above) %>%
  kable(format = "latex", booktabs = TRUE, digits = 4,
        col.names = c("Position",
                      "Proportion of players with avg npxG + xA > 0.35"),
        caption = "Most forwards have a very high average npxG + xA,
        and everyone else has a low average.",
        align = c("l", "c")) %>%
  kable_styling(position = "center", latex_options = "HOLD_position")
```

Table 2: Most forwards have a very high average npxG + xA, and everyone else has a low average.

Position	Proportion of players with avg npxG + xA > 0.35
DF	0.026
FW	0.577
GK	0.024
MF	0.050

Only 2.6 of defenders and 5.0% of midfielders have a mean_npxg_xa_per_90 above 0.35, but 57.7% of forwards eclipse that mark. I am starting to become even more confident the top 10 league leaders in npxg + xA this weekend will all be forwards.

We just checked what percentage of players have an average npxg + xA per 90 above 0.35. Now, let's check what percentage of all performances of npxg + xA per 90 greater than 0.35 came from each position.

```
npxg_xa_performances <-
  matchlog %>%
  group_by(position) %>%
  drop_na() %>%


  mutate(npxg_xa_per_90 =
           90*((player_match_np_xg/player_match_minutes)
               + (player_match_xa/player_match_minutes))) %>%

  summarize(high_npxg_xa_performance =
              sum(90*((player_match_np_xg/player_match_minutes)
               + (player_match_xa/player_match_minutes)) > 0.35),
            very_high_npxg_xa_performance =
              sum(90*((player_match_np_xg/player_match_minutes)
               + (player_match_xa/player_match_minutes)) > 0.75)) %>%
  ungroup() %>%
  mutate(prop_high_npxg_xa_performances =
           round((high_npxg_xa_performance/sum(high_npxg_xa_performance)), 3),
         prop_very_high_npxg_xa_performances =
           round((very_high_npxg_xa_performance/sum(very_high_npxg_xa_performance)), 3)
  )
npxg_xa_performances %>%
  kable(format = "latex", booktabs = TRUE, digits = 4,
        col.names = c("Position", "num_high_npxg_xA",
                      "num_very_high_npxg_xA", "prop_high_npxg_xA",
                      "prop_very_high_npxg_xA"),
        caption = "Forwards account for most of the
        high and very high npxG + xA performances.",
        align = c("l", "r", "r", "r", "r")) %>%
  kable_styling(position = "center", latex_options = "HOLD_position")
```

Table 3: Forwards account for most of the high and very high npxG + xA performances.

Position	num_high_npxg_xA	num_very_high_npxg_xA	prop_high_npxg_xA	prop_very_high_npxg_xA
DF	490	121	0.159	0.109
FW	2050	831	0.663	0.751
GK	4	1	0.001	0.001
MF	546	153	0.177	0.138

Forwards make up two-thirds of the performances where npxg + xA per 90 minutes is greater than 0.35. The percentage only increases as we increase the threshold for a high npxg + xA performance, and as we approach the threshold of 0.75 npxg + xA/90 min, which should be closer to the league lead for a given weekend, forwards make up 75% of those performances.

Assumption: Due to time constraints, I will assume that the top 10 npxg + xA performers this upcoming weekend will all be forwards. While this is likely inaccurate, it is highly likely that at least 6 of the top 10 will be forwards.

Unfortunately, this isn't a great assumption to make not only because there are lots of midfielders and defenders that occassionally have high npxg + xA performances, but also because it is based on the fact that forwards get more npxg + xA *per 90 minutes*. However, the leaderboard I will be building is for total npxg + xA, not on a per 90 minute basis. So, the assumption is weak if forwards were far more likely to get subbed off early than defenders and midfielders/forwards play a statistically significant amount of minutes less than defenders and midfielders every match. Sadly, that notion is true, and the t-tests below show forwards do play a statistically significant amount less than midfielders and defenders every match.

```
t.test(
  player_match_minutes ~ position,
  data = matchlog %>% filter(position %in% c("FW", "DF"))
)
```

```
##
##  Welch Two Sample t-test
##
## data:  player_match_minutes by position
## t = 29.161, df = 11481, p-value < 0.00000000000000022
## alternative hypothesis: true difference in means between group DF and group FW is not equal to 0
## 95 percent confidence interval:
##  15.85147 18.13605
## sample estimates:
## mean in group DF mean in group FW
##         79.52340         62.52964
```

```
t.test(
  player_match_minutes ~ position,
  data = matchlog %>% filter(position %in% c("FW", "MF"))
)
```

```
##
##  Welch Two Sample t-test
##
## data:  player_match_minutes by position
## t = -7.5602, df = 8951.9, p-value = 0.00000000000004422
```

```
## alternative hypothesis: true difference in means between group FW and group MF is not equal to 0
## 95 percent confidence interval:
##  -6.412422 -3.771813
## sample estimates:
## mean in group FW mean in group MF
##         62.52964         67.62176
```

While a t-test may not be the ideal hypothesis test for this question, as the underlying data may not be normal, we can use CLT to claim that it is normal for the purpose of running this test.

While the assumption that forwards will take all 10 spots on the leaderboard is not great, I will still make it due to time constraints.

As a result of this assumption, I will only be considering forwards for the rest of this project.

```
matchlog_fw <-
  matchlog %>%
  filter(position == "FW")

npxg_xa_90_player_means_fw <-
  npxg_xa_90_player_means %>%
  filter(position == "FW")
```

Hierarchical Normal Bayesian Model for npxg per 90 Minutes for Forwards

We will use a Hierarchical Normal model with Gibbs sampling to estimate the true talent npxg + xA /90 min of all forwards.

Let $y_{ij} \sim$ Normal(mu, $sigma^2$), where $i = 1,\ldots, n$

- y_{ij} = the npxg + xA /90 min in the ith game played by forward j
- mu_j = forward j's average npxg + xA /90 min
- $sigma^2$ = variation within each forward's distribution of npxg + xA /90 min values

The player means, mu_j, share a common prior: $mu_j \sim$ Normal(mu_0, tau^2)

- mu_0 is the global mean of all forwards' average npxg + xA /90 min
- tau^2 is the gloal variance of all forwards' average npxg + xA /90 min... variation between players

We will use a non-informative joint prior for mu_0, $sigma^2$, and tau^2: p(mu_0, $sigma^2$, tau^2) is proportional to $1 * 1/tau * 1/sigma^2$.

Joint posterior distribution for p(mu, mu_0, tau^2, $sigma^2$ | y) is proportional to p(y | mu, $sigma^2$) * p(mu | mu_0, tau^2) * p(mu_0, tau^2, $sigma^2$), which unfortunately I will not type out.

The steps of the Gibbs sampler are:

1. Set initial parameter values for mu_0, tau^2, and $sigma^2$
2. Iteratively sample from the conditional distribution of each parameter given current samples of the other parameters

- Sample each mu_j from p(mu_j | $mu_0, tau^2, sigma^2, y$)

- Sample mu_0 from p($mu_0 \mid mu_j, tau^2, sigma^2, y$)
- Sample tau^2 from p($tau^2 \mid mu_j, mu_0, sigma^2, y$)
- Sample $sigma^2$ from p($tau^2 \mid mu_j, mu_0, tau^2, y$)

Unfortunately, those conditional distributions are also too long for me to type out in this document.

When we are done Gibbs sampling, we will:

- Check convergence, discard burn-in, and combine the chains
- Thin the chains
- Sample from the posterior predictive distributions for each player
- Find each player's posterior predictive mean as their projection

The primary benefit of using a hierarchical model is it allows us to take into account variability within a single player's performances as well as variability between players.

I will only use data from 2024 and 2025, as it is recent enough to still paint a good picture of the player's (expected) goal scoring and assisting ability.

Put the Data in Wide Format

```
matchlog_fw_wide <-
  matchlog_fw %>%
  mutate(player_match_np_xg_xa = player_match_np_xg + player_match_xa) %>%
  select(player_id, player_match_np_xg_xa, player_match_minutes) %>%
  filter(!is.na(player_match_np_xg_xa)) %>%
  mutate(npxg_xa_90 = as.numeric(90*(player_match_np_xg_xa/player_match_minutes))) %>%
  select(-c(player_match_np_xg_xa, player_match_minutes)) %>%
  group_by(player_id) %>%
  mutate(row_num = row_number()) %>%
  pivot_wider(names_from = player_id, values_from = npxg_xa_90) %>%
  select(-row_num)
```

Since the true population variance is unknown, we cannot easily sample from the posterior distribution of npxg performances. So, we must use the Gibbs Sampler.

As a reminder, the steps of the Gibbs Sampler are:

- Set initial parameter values for mu_0, tau^2, and $sigma^2$
- Iteratively sample from the conditional distribution of each parameter given current samples of the other parameters

Calculating necessary statistics

Here is a list of variables we will have to create:

- m = The number of forwards... akin to the number of columns in the data set.
- n = a vector of the number of npxg + xA performances/games played for each player
- means = a vector with each player's mean npxg + xA per 90 minutes... the player-specific mean
- ntot = the total number of npxg + xA performances/games played for all players in the dataset
- Then we need to declare these variables for Gibbs sampling:

- numsamp = the number of iterations for the Gibbs sampler
- sigsq.samp = a vector storing the within player variances from each iteration
- tausq.samp = a vector storing the between player variances from each iteration
- mu0.samp = a vector storing samples of the global mean npxg + xA per 90 minutes across all players
- mu.samp = a matrix storing samples of the individual player mean npxg + xA per 90 minutes values for each iteration. Has m columns (for m forwards) and numsamp rows (for number of iterations)
- sigsq = initial value for within player variance
- tausq = initial value for between player variance
- mu0 = initial value for global mean of npxg + xA per 90 min across all players
- mu = vector storing initial values for the average npxg + xA per 90 min for each player

```
#m = number of players in the dataset
m <- length(matchlog_fw_wide[1,])
n <- rep(NA,m)
means <- rep(NA,m)

#convert the df to a matrix
matchlog_fw_wide <- as.matrix(matchlog_fw_wide)
#make sure everything is numeric
matchlog_fw_wide <- apply(matchlog_fw_wide, 2, as.numeric)

#n = a vector with the number of npxg + xAper 90 performances for each player
#means = a vector with each player's average npxg + xA per 90 min
for (i in 1:m){
  n[i] <- sum(!is.na(matchlog_fw_wide[, i]))
  means[i] <- mean(matchlog_fw_wide[, i], na.rm = TRUE)
}
#ntot = total npxg + xA performances/games played
ntot <- sum(n)
```

Initializing variables for Gibbs sampling

```
#numsamp = number of iterations of Gibbs sampler
numsamp <- 50000

#Creating empty vectors of length numsamp to hold the values
#we will generate for each parameter from the Gibbs sampler
sigsq.samp <- rep(NA,numsamp)
tausq.samp <- rep(NA,numsamp)
mu0.samp <- rep(NA,numsamp)
mu.samp <- matrix(NA,nrow=numsamp,ncol=m)

#Set the initial values of the parameters
sigsq <- 1
tausq <- 1
mu0 <- 1
mu <- rep(NA,m)
```

Using Gibbs to sample the first chain

```
set.seed(679)
for (i in 1:numsamp){
   # sampling mu's
   for (j in 1:m){
      curvar <- 1/(n[j]/sigsq + 1/tausq)
      curmean <- (means[j]*n[j]/sigsq + mu0/tausq)*curvar
      mu[j] <- rnorm(1,mean=curmean,sd=sqrt(curvar))
   }
   # sampling mu0
   mu0 <- rnorm(1,mean=mean(mu),sd=sqrt(tausq/m))
   # sampling tausq
   sumsq.mu <- sum((mu-mu0)^2)
   tausqinv <- rgamma(1,shape=((m-1)/2),rate=(sumsq.mu/2))
   tausq <- 1/tausqinv
   # sampling sigsq
   sumsq.y <- 0
   for (j in 1:m){
     valid_y_j <-
       matchlog_fw_wide[, j][!is.na(matchlog_fw_wide[, j])]
      sumsq.y <-
        sumsq.y + sum((valid_y_j - mu[j])^2)
   }
   sigsqinv <- rgamma(1,shape=(ntot/2),rate=(sumsq.y/2))
   sigsq <- 1/sigsqinv
   # storing sampled values
   mu.samp[i,] <- mu
   mu0.samp[i] <- mu0
   tausq.samp[i] <- tausq
   sigsq.samp[i] <- sigsq
}
```

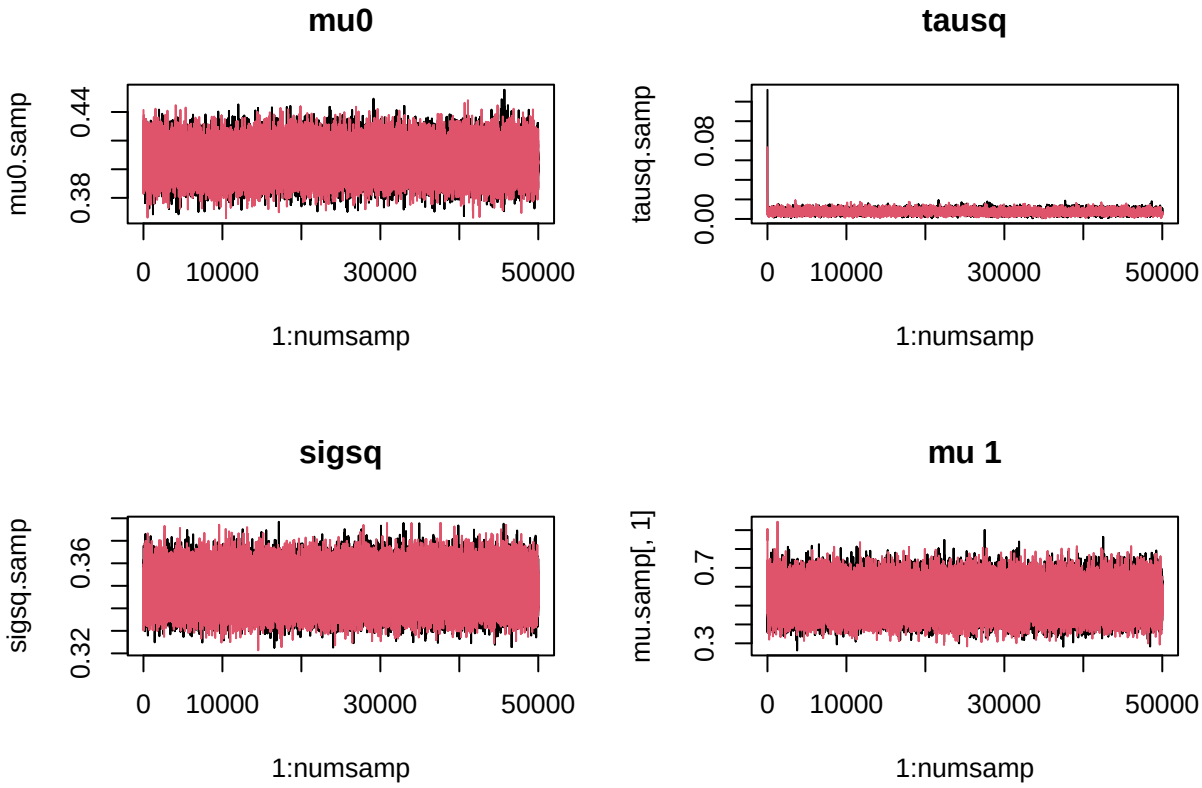
Run a second chain with different initial values

```
#New vectors and new initial values
sigsq.samp2 <- rep(NA,numsamp)
tausq.samp2 <- rep(NA,numsamp)
mu0.samp2 <- rep(NA,numsamp)
mu.samp2 <- matrix(NA,nrow=numsamp,ncol=m)

sigsq <- 0.5
tausq <- 0.25
mu0 <- 0.45
mu <- rep(NA,m)

#Running the second chain
for (i in 1:numsamp){
   # sampling mu's
   for (j in 1:m){
      curvar <- 1/(n[j]/sigsq + 1/tausq)
```

```
      curmean <- (means[j]*n[j]/sigsq + mu0/tausq)*curvar
      mu[j] <- rnorm(1,mean=curmean,sd=sqrt(curvar))
   }
   # sampling mu0
   mu0 <- rnorm(1,mean=mean(mu),sd=sqrt(tausq/m))
   # sampling tausq
   sumsq.mu <- sum((mu-mu0)^2)
   tausqinv <- rgamma(1,shape=((m-1)/2),rate=(sumsq.mu/2))
   tausq <- 1/tausqinv
   # sampling sigsq
   sumsq.y <- 0
   for (j in 1:m){
     valid_y_j <-
       matchlog_fw_wide[, j][!is.na(matchlog_fw_wide[, j])]
      sumsq.y <-
        sumsq.y + sum((valid_y_j - mu[j])^2)
   }
   sigsqinv <- rgamma(1,shape=(ntot/2),rate=(sumsq.y/2))
   sigsq <- 1/sigsqinv
   # storing sampled values
   mu.samp2[i,] <- mu
   mu0.samp2[i] <- mu0
   tausq.samp2[i] <- tausq
   sigsq.samp2[i] <- sigsq
}
```

Checking for Convergence

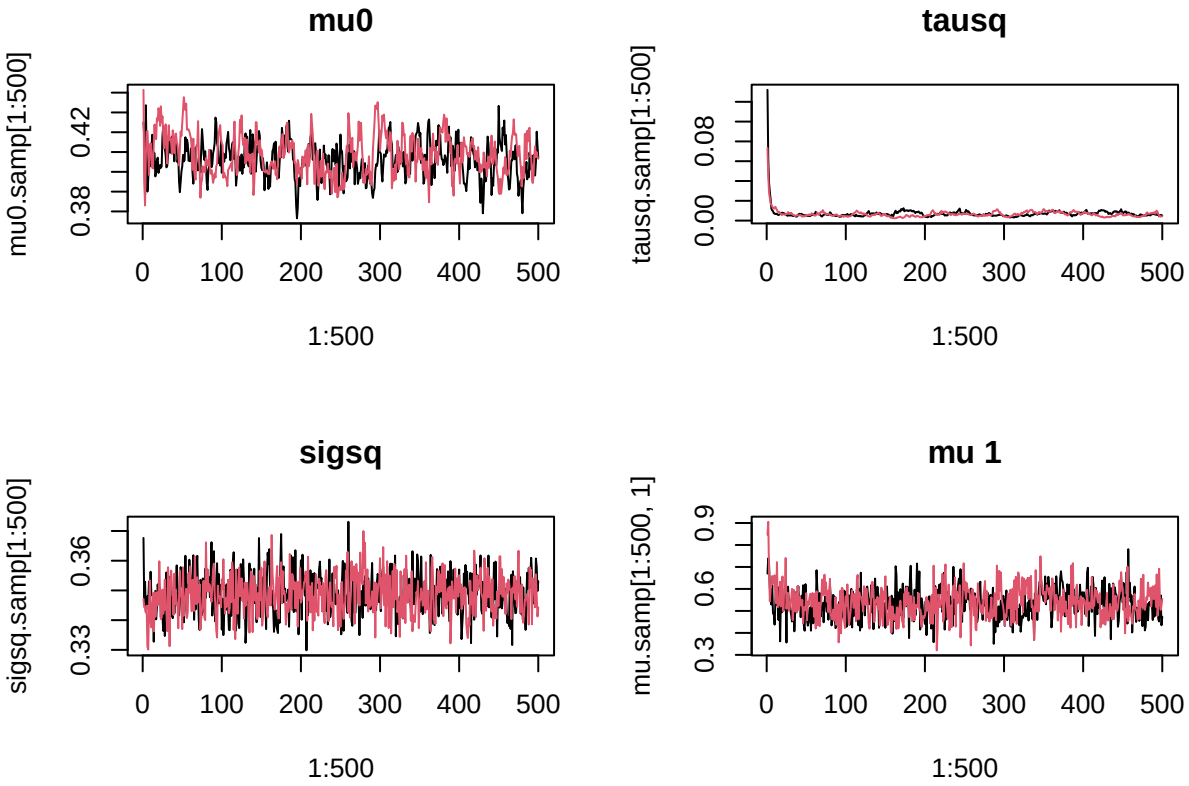
```
par(mfrow=c(2,2))
ymin <- min(mu0.samp,mu0.samp2)
ymax <- max(mu0.samp,mu0.samp2)
plot(1:numsamp,mu0.samp,type="l",main="mu0",ylim=c(ymin,ymax))
lines(1:numsamp,mu0.samp2,col=2)
ymin <- min(tausq.samp,tausq.samp2)
ymax <- max(tausq.samp,tausq.samp2)
plot(1:numsamp,tausq.samp,type="l",main="tausq",ylim=c(ymin,ymax))
lines(1:numsamp,tausq.samp2,col=2)
ymin <- min(sigsq.samp,sigsq.samp2)
ymax <- max(sigsq.samp,sigsq.samp2)
plot(1:numsamp,sigsq.samp,type="l",main="sigsq",ylim=c(ymin,ymax))
lines(1:numsamp,sigsq.samp2,col=2)
ymin <- min(mu.samp[,1],mu.samp2[,1])
ymax <- max(mu.samp[,1],mu.samp2[,1])
plot(1:numsamp,mu.samp[,1],type="l",main="mu 1",ylim=c(ymin,ymax))
lines(1:numsamp,mu.samp2[,1],col=2)
```



Both chains converge very quickly for all of the parameters.

Checking Convergence (focusing on first 500 iters)

```
par(mfrow=c(2,2))
ymin <- min(mu0.samp[1:500],mu0.samp2[1:500])
ymax <- max(mu0.samp[1:500],mu0.samp2[1:500])
plot(1:500,mu0.samp[1:500],type="l",main="mu0",ylim=c(ymin,ymax))
lines(1:500,mu0.samp2[1:500],col=2)
ymin <- min(tausq.samp[1:500],tausq.samp2[1:500])
ymax <- max(tausq.samp[1:500],tausq.samp2[1:500])
plot(1:500,tausq.samp[1:500],type="l",main="tausq",ylim=c(ymin,ymax))
lines(1:500,tausq.samp2[1:500],col=2)
ymin <- min(sigsq.samp[1:500],sigsq.samp2[1:500])
ymax <- max(sigsq.samp[1:500],sigsq.samp2[1:500])
plot(1:500,sigsq.samp[1:500],type="l",main="sigsq",ylim=c(ymin,ymax))
lines(1:500,sigsq.samp2[1:500],col=2)
ymin <- min(mu.samp[1:500,1],mu.samp2[1:500,1])
ymax <- max(mu.samp[1:500,1],mu.samp2[1:500,1])
plot(1:500,mu.samp[1:500,1],type="l",main="mu 1",ylim=c(ymin,ymax))
lines(1:500,mu.samp2[1:500,1],col=2)
```



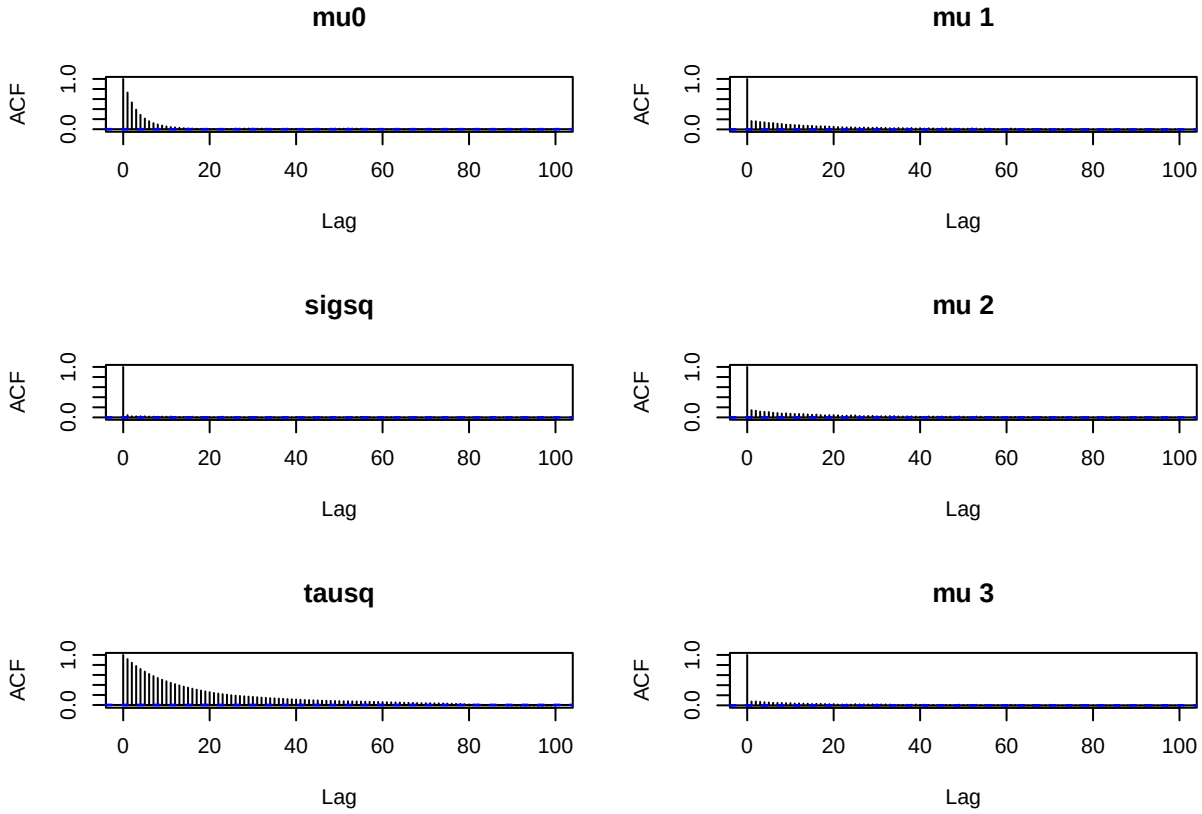
Even when we zoom in on the first 500 iterations, both chains still show convergence very quickly for all parameters. However, to be safe, we will still throw away the first 100 iterations as burn-in before we combine the chains.

Throwing away the first 100 iterations as burn-in, combining chains

```
mu.postburn <- rbind(mu.samp[101:numsamp,],mu.samp2[101:numsamp,])
mu0.postburn <- c(mu0.samp[101:numsamp],mu0.samp2[101:numsamp])
tausq.postburn <- c(tausq.samp[101:numsamp],tausq.samp2[101:numsamp])
sigsq.postburn <- c(sigsq.samp[101:numsamp],sigsq.samp2[101:numsamp])
```

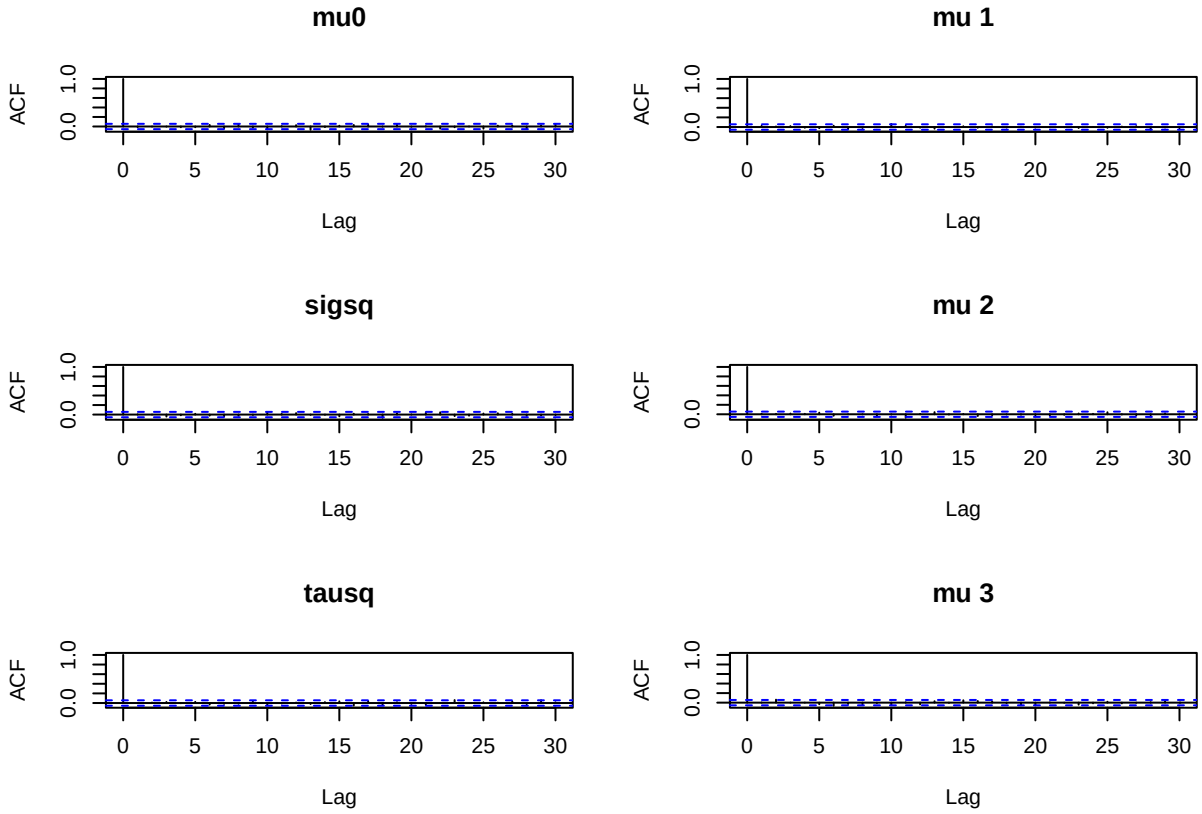
Checking autocorrelation of draws and thinning

```
### autocorrelation of draws
par(mfcol=c(3,2))
acf(mu0.postburn,main="mu0", lag.max = 100)
acf(sigsq.postburn,main="sigsq", lag.max = 100)
acf(tausq.postburn,main="tausq", lag.max = 100)
acf(mu.postburn[,1],main="mu 1", lag.max = 100)
acf(mu.postburn[,2],main="mu 2", lag.max = 100)
acf(mu.postburn[,3],main="mu 3", lag.max = 100)
```



We need 82 lags for the autocorrelation of every parameter to fall within the dashed blue lines. So, we will thin the chains, keeping only every 82nd iteration.

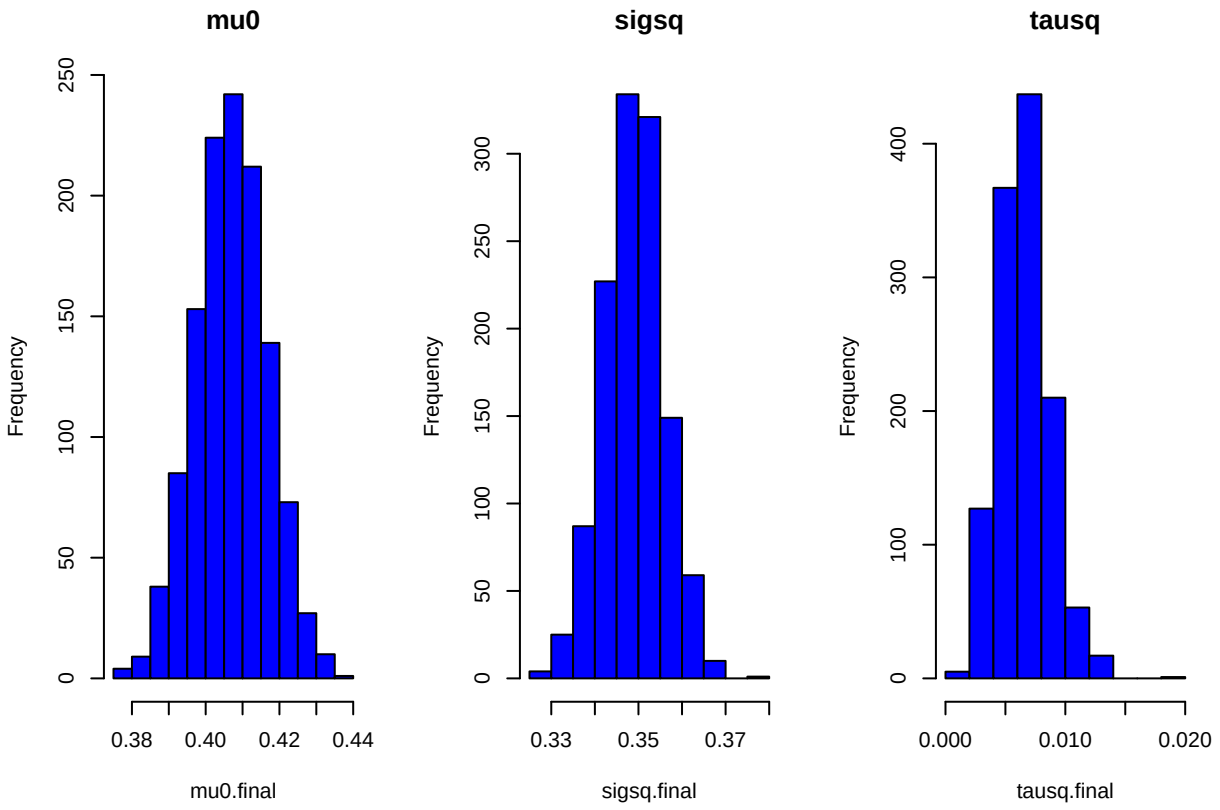
```
numpostburn <- length(mu0.postburn)
temp <- 82*c(1:(numpostburn/82))
mu0.final <- mu0.postburn[temp]
sigsq.final <- sigsq.postburn[temp]
tausq.final <- tausq.postburn[temp]
mu.final <- mu.postburn[temp,]

par(mfcol=c(3,2))
acf(mu0.final,main="mu0")
acf(sigsq.final,main="sigsq")
acf(tausq.final,main="tausq")
acf(mu.final[,1],main="mu 1")
acf(mu.final[,2],main="mu 2")
acf(mu.final[,3],main="mu 3")
```



Now, the autocorrelation is sufficiently low to proceed.

Analyzing the posterior distributions of global parameters

```
par(mfrow=c(1,3))
hist(mu0.final,main="mu0",col="blue")
hist(sigsq.final,main="sigsq",col="blue")
hist(tausq.final,main="tausq",col="blue")
```



The posterior distributions of the 3 global variables are all normal.

Analyzing posterior means

```
mu.postmean <- apply(mu.final,2,mean)
mu0.postmean <- mean(mu0.final)
tausq.postmean <- mean(tausq.final)
sigsq.postmean <- mean(sigsq.final)
mu0.postmean
```

```
## [1] 0.4069725
```

```
tausq.postmean
```

```
## [1] 0.006575151
```

```
sigsq.postmean
```

```
## [1] 0.3489507
```

The average npxg + xA / 90 min for a striker according to the posterior mean is 0.407.

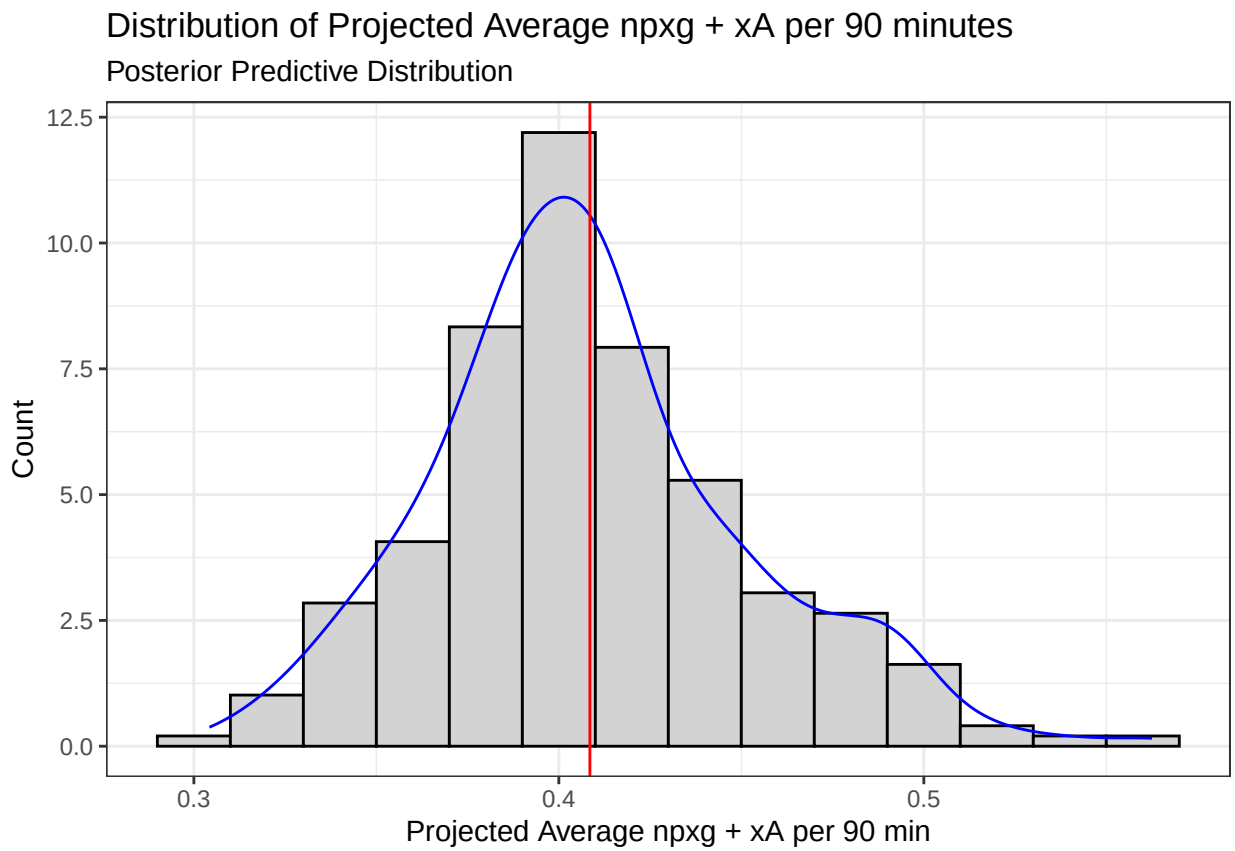
Generate Predictions for Each Player from the Posterior Predictive Distribution

We can use the posterior predictive to make predictions on future observations, and link them back to their player id.

```
set.seed(2015)

#Grab the player ids
player_ids <- colnames(matchlog_fw_wide)

#Sample the posterior predictive values, link them to the player id
post_pred_samples <- list()
for (i in 1:length(mu.postmean)){
  post_pred_values <-
    rnorm(1000, mean = mu.postmean[i], sd = sqrt(sigsq.postmean))
  post_pred_samples[[player_ids[i]]] <- post_pred_values
}

#Find the posterior predictive mean for each individual player,
#keeping it linked to the player id
post_pred_means <- sapply(post_pred_samples, mean)
```

Visualizing the Posterior Predictive Distribution

```
final_npxg_xa_90_projections = data.frame(player_id = names(post_pred_means),
                              projected_npxg_xa_90 = as.numeric(post_pred_means))

ggplot(data = final_npxg_xa_90_projections,
       aes(x = projected_npxg_xa_90)) +
  geom_histogram(mapping = aes(y = after_stat(density)),
                 color = "black", fill = "lightgray", binwidth = 0.02) +
  geom_density(color = "blue") +
  geom_vline(xintercept =
               mean(final_npxg_xa_90_projections$projected_npxg_xa_90),
             color = "red") +
  labs(
    x = "Projected Average npxg + xA per 90 min",
    y = "Count",
    title = "Distribution of Projected Average npxg + xA per 90 minutes",
    subtitle = "Posterior Predictive Distribution"
  ) +
  theme_bw()
```



The posterior predictive distribution is relatively normal, and it has a mean around 0.41. Most players have projected npxg + xA per 90 minutes values of 0.35-0.45, but exceptionally bad players have projections closer to 0.3, while exceptionally good ones have projections above 0.5.

Playing Time Rule

The last step is applying a playing time rule. The npxg + xA values that I projected were on a per 90 minute basis. However, I need to project total npxg + xA in the next game, so I need to divide my projections by 90, and then multiply them by their projected number of minutes.

To project number of minutes, I will simply use each player's minutes played per game in 2025. I am only using 2025 data because there is so much roster turnover that affects everyone's playing time between seasons.

I will make one adjustment to the projections - if there is a player who played zero minutes in his team's most recent game, I will cut his minutes projection for this game in half. This will be my way of accounting for potential injuries.

Re-attaching the npxG + xA values to their teams

```
final_npxg_xa_90_projections <-
  final_npxg_xa_90_projections %>%
  mutate(player_id = as.numeric(player_id))

matchlog_fw_players <-
```

```
  matchlog_fw %>%
  select(player_id, team_id) %>%
  distinct() %>%
  inner_join(final_npxg_xa_90_projections, by = "player_id")

matchlog_fw_players <-
  matchlog_fw_players %>%
  mutate(proj_npxg_xa_per_min = projected_npxg_xa_90/90)
```

Setting up Schedule Data Frames

```
schedule_2025 <-
  schedule %>%
  mutate(match_year = year(match_date)) %>%
  filter(match_year == 2025 & match_week <= 5)
```

```
#Get number of matches played by each team
num_matches_played <-
  schedule_2025 %>%
  filter(match_status == "available") %>%
  select(home_team_id, away_team_id) %>%
  pivot_longer(cols = c(home_team_id, away_team_id),
               values_to = "team_id") %>%
  group_by(team_id) %>%
  summarize(matches_played = n()) %>%
  ungroup()
```

It appears that not all of the matches from matchweek 4 were loaded into the data set when the file was sent to me, so a few teams only have 3 matches played.

```
#Get match_id for most recent match played by a team
most_recent_match <-
  schedule_2025 %>%
  filter(match_status == "available") %>%
  select(match_week, match_id, home_team_id, away_team_id) %>%
  pivot_longer(cols = c(home_team_id, away_team_id), values_to = "team_id") %>%
  group_by(team_id) %>%
  slice_max(order_by = match_week, with_ties = FALSE) %>%
  select(team_id, match_id, match_week) %>%
  arrange(team_id)
```

```
#Get total minutes played for each player in fw data set this season
matchlog_fw_pt <-
  matchlog_fw %>%
  filter(season_name == 2025) %>%
  group_by(player_id, team_id) %>%
  summarize(total_minutes = sum(player_match_minutes, na.rm = TRUE)) %>%
  ungroup()
```

```
## `summarise()` has grouped output by 'player_id'. You can override using the
## `.groups` argument.
```

```
#Join that with team matches list to get how many games the team has played
matchlog_fw_pt <-
  matchlog_fw_pt %>%
  inner_join(num_matches_played, by = 'team_id')

#Get average minutes in a match

#Note: the average purposely includes games they didn't play in.
#I am not looking for average minutes in games that they played in,
#which is what most websites mean when they report minutes per game
#in any sport. I want to include matches where they played 0 minutes,
#as it will help me predict how many minutes they will play in the next match.
matchlog_fw_pt <-
  matchlog_fw_pt %>%
  mutate(minutes_per_match = total_minutes/matches_played)

#Combine this table with the npxg + xA per minute table
matchlog_fw_players <-
  matchlog_fw_players %>%
  inner_join(matchlog_fw_pt, by = c('player_id', 'team_id'))
```

Make Final Projections

```
#Create projection of npxg + xA

#Multiply average minutes played and npxg +xA per minute
npxg_xa_projections <-
  matchlog_fw_players %>%
  mutate(pred_npxg_xa = proj_npxg_xa_per_min * minutes_per_match) %>%
  select(player_id, team_id, pred_npxg_xa)

#Find players that didn't play in their last game

most_recent_match_fw <-
  most_recent_match %>%
  inner_join(matchlog_fw, by = c("team_id", "match_id")) %>%
#Everybody in this table did play in their last game, add dnp indicator
  mutate(dnp_recent_indicator = 0)

#Get the list of players who didn't play in their last game
#Add an indicator variable to them before joining them back
#to the npxg projections
absent_players <-
  npxg_xa_projections %>%
  anti_join(most_recent_match_fw, by = 'player_id') %>%
  mutate(dnp_recent_indicator = 1)

most_recent_match_full <-
  most_recent_match_fw %>%
  full_join(absent_players,
            by = c('player_id', 'team_id', 'dnp_recent_indicator')) %>%
```

```
  select(player_id, team_id, dnp_recent_indicator)

npxg_xa_projections <-
  npxg_xa_projections %>%
  inner_join(most_recent_match_full,
             by = c('player_id', 'team_id')) %>%
#For players that missed their last game,
#halve their npxg projection for next game
  mutate(final_pred_npxg_xa = case_when(
    dnp_recent_indicator == 1 ~ pred_npxg_xa * 0.5,
    dnp_recent_indicator == 0 ~ pred_npxg_xa
  )) %>%
  select(player_id, team_id, final_pred_npxg_xa)

#Bring the player and team names back
matchlog_fw_names <-
  matchlog_fw %>%
  select(player_id, player_name, team_id, team_name) %>%
  distinct()

#Join the player and team names back to the projections
npxg_xa_projections <-
  npxg_xa_projections %>%
  inner_join(matchlog_fw_names,
             by = c('player_id', 'team_id')) %>%
  select(1, 4, 2, 5, 3)
```

Get top 10

```
npxg_xa_projections %>%
  arrange(desc(final_pred_npxg_xa)) %>%
  select(player_name, team_name, final_pred_npxg_xa) %>%
  head(10) %>%
  kable(format = "latex", booktabs = TRUE, digits = 4,
        col.names = c("Player", "Team", "Predicted npxG + xA"),
        caption = "Top 10 Predicted npxG + xA Leaders This Weekend",
        align = c("l", "l", "r")) %>%
  kable_styling(position = "center", latex_options = "HOLD_position")
```

Table 4: Top 10 Predicted npxG + xA Leaders This Weekend

Player	Team	Predicted npxG + xA
Gabriel Pec	LA Galaxy	0.5174
Anders Dreyer	San Diego FC	0.5117
Christian Benteke	DC United	0.5101
Cristian Arango	San Jose Earthquakes	0.4924
Adrian Alonso Martínez Batista	New York City FC	0.4839
Federico Bernardeschi	Toronto FC	0.4713
Ignatius Ganago	New England Revolution	0.4711
Brian White	Vancouver Whitecaps	0.4686
Diego Rossi	Columbus Crew	0.4664
Petar Musa	FC Dallas	0.4651