**Introduction**

I model and predict [English Premier League](https://en.wikipedia.org/wiki/Premier_League) (EPL) game outcomes using Bayesian methods. Specifically, I estimated goals scored by each team in a given game as independent Poisson processes, taking the difference of the estimated points scored on each side to determine game winners. More broadly, one may call this a hierarchical Bayesian Poisson model.

Why model goals scored using a Poisson distribution? it “is a discrete probability distribution that expresses the probability of a given number of events occurring in a fixed interval of time with a known constant rate.” In the context of soccer, the fixed interval of time is the 90 minutes of a game (disregarding injury time and over time), and the known constant rate is the expected number of goals scored per minute. Importantly, I must make the assumption that the rate of scored goals is the same across all minutes of a game. Additionally, when computing the difference between Poisson distributions, I must assume that the two distributions are independent of one another.

Using Poisson distributions to model soccer scores is certainly not a novel concept.

**Data Collection**

For this project I retrieved game scores and outcomes for the previous three seasons of EPL games (i.e. from the 2016-17 season through the 2018-2019 season).

Before impementing some models, I need to “wrangle” the raw data a bit in order to put it in a more workable format for OpenBUGs. (Specifically, I need to put it into a list.)

**Modeling**

My model is formally defined as follows.

\[  
\begin{array}{c}  
g\_h \sim \mathcal{Pois}(\lambda\_{h,i,j}) \\  
g\_a \sim \mathcal{Pois}(\lambda\_{a,i,j}) \\  
\log(\lambda\_{h,i,j}) = \text{baseline}\_h + (z\_i – z\_j) \\  
\log(\lambda\_{a,i,j}) = \text{baseline}\_a + (z\_j – z\_i). \\  
\end{array}  
\]

This model estimates the goals scored by the home team, \(g\_h\), and the goals scored by the away team, \(g\_a\), in a given game between home team, \(\text{tm}\_h\), and away team, \(\text{tm}\_a\), as random variables coming from independent Poisson processes, \(\mathcal{Pois}(\lambda\_{h,i,j})\) and \(\mathcal{Pois}(\lambda\_{a,i,j})\). The log of the rate of goals scored by the home team, \(\lambda\_{h,i,j}\), in a game between \(\text{tm}\_i\) and \(\text{tm}\_j\) is modeled as the sum of a “baseline” average of goals scored by any given team playing at home, \(\text{baseline}\_h\), and the difference between the team “strength” \(z\) of teams \(i\) and \(j\) in a given game. I define the log of the goal rate by the away team, \(\lambda\_{a,i,j}\), in a similar fashion. It is important to distinguish the baseline levels for home and away so as to account for. (One should expect to find that \(\text{baseline}\_h > \text{baseline}\_a\) in the posterior estimates.)

Since I am employing a Bayesian approach, I need to model priors as well. I define them as follows.

\[  
\begin{array}{c}  
\text{baseline}\_h \sim \mathcal{N}(0, 2^2) \\  
\text{baseline}\_a \sim \mathcal{N}(0, 2^2) \\  
z\_{i} \sim \mathcal{N}(z\_{\text{all}} , \sigma^2\_{\text{all}}) \quad \text{tm}\_i > 1 \\  
z\_{\text{all}} \sim \mathcal{N}(0, 2^2) \\  
\sigma\_{\text{all}} \sim \mathcal{U}(0, 2).  
\end{array}  
\]

There are a couple of things to note about these priors. First, I must “zero”-anchor the strength estimate \(z\) of one team. (This is manifested by \(\text{tm}\_i > 1\).) Here, I choose the first team alphabetically—Arsenal. Second, the priors are intentionally defined to be relatively vague (although not too vauge) so as to allow the posterior estimates to be heavily defined by the data rather than the priors. Note that the standard deviation of the overall team strength parameter \(z\_{\text{all}}\), defined as \(2\) on a log scale, corresponds to an interval of \(\left[e^{-2}, e^2\right] = \left[0.13, 7.40\right]\) on an unstransformed scale, i.e. goals scored per game.

I leverage the {R2OpenBUGs} package to create this model with R on the “frontend” and generate the results using the OpenBUGS engine on the “backend”. Regarding the implementation itself, note that I run 100,000 simulations (n.iter), minus 1,000 “burn-in” runs (n.burn).

The raw results are as follows. (As a quick “validation” of these results, note that \(\text{baseline}\_h > \text{baseline}\_a\), as hypothesized.)

**Interpretation & Discussion**

Next, I correspond the strength estimates \(z\) to teams. Notably, I must “re-add” the zero-anchored team—Arsenal (whose \(z\) is assigned a dummy value of 1). To do this, I impute its credible set quantiles using the values of the overall strength term \(z\_{\text{all}}\).

It’s not surprising to see that the strength (\(z\)) corresponding to all but three teams—Liverpool, Man City, and Tottenham—is negative. These three teams, followed closely by Arsenal have been regarded as the best teams for the past two or three EPL seasons. So, relative to Arsenal, all other teams (aside from the top three) are viewed as “worse” by the model.

Note that the \(z\) estimates above should not be interpreted as goals scored by the teams because they are relative to the strength of Arsenal. To facilitate such an interpretation, I need to translate \(z\) to goals scored per game. To do this, for each \(z\), I (1) subtract the average value of all \(z\)’s, (2) add the posterior mean of \(\text{baseline}\_{h}\), and (3) exponentiate.

The plot below shows the results of this transformation.

**Predictions**

I can make predictions of game results for the historical data, given the model. (Since the model estimates were calculated using this data, I’m really predicting on the “training” set here.) Specifically, I simulate the score for both teams in each matchup (1,140 in all) 1,000 times, choosing the result inferred by the mode of each side’s simulated score. (For example, if the mode of the 1,000 simulated scores for the away team is 1 and that of the home team is 2, then the predicted outcome is a win for the home team.) A breakdown of the predicted and actual outcomes is shown below.

I make a couple of observations:

* The most common outcome is an actual win by the home team and a predicted win by the home team.
* The model never predicts a tie. (This may seem “unreasonable”, but Bååth also found this to be true for his final model.)
* The model predicts the outcome correctly in 447 + 216 = 663 of 1,140 games (i.e., 58%).

The next couple of visuals provide more details regarding the simulated outcomes.

From the above graph of the mode of goals scored by both sides, it’s apparent that a 2-1 scores in favor of the home side is the most common outcome.

The above histogram illustrating the mean (instead of the mode) of the simulated goals provides a bit more nuance to our understanding of modes shown before.

Finally, the above visual shows the predicted outcomes (inferred from the prior graph of predicted modes).

To better understand how the model works on a team-level basis, let’s look at how well it predicts for each team.

## # A tibble: 26 x 4

## tm\_h is\_correct n pct

##

## 1 Arsenal TRUE 46 81%

## 2 Man City TRUE 45 79%

## 3 Tottenham TRUE 43 75%

## 4 Liverpool TRUE 41 72%

## 5 Chelsea TRUE 39 68%

## 6 Cardiff TRUE 12 63%

## 7 Everton TRUE 36 63%

## 8 Fulham TRUE 12 63%

## 9 Man United TRUE 35 61%

## 10 Huddersfield TRUE 23 61%

## 11 Burnley TRUE 34 60%

## 12 Stoke TRUE 22 58%

## 13 Bournemouth TRUE 30 53%

## 14 Crystal Palace TRUE 30 53%

## 15 Sunderland TRUE 10 53%

## 16 Swansea TRUE 20 53%

## 17 West Ham TRUE 30 53%

## 18 Watford TRUE 29 51%

## 19 Newcastle TRUE 19 50%

## 20 Leicester TRUE 28 49%

## 21 Middlesbrough TRUE 9 47%

## 22 West Brom TRUE 18 47%

## 23 Wolves TRUE 9 47%

## 24 Brighton TRUE 15 39%

## 25 Southampton TRUE 22 39%

## 26 Hull TRUE 6 32%

In most cases, the model predicts the outcome correctly (see is\_correct) with greater than 50% accuracy, although there are also teams for which its accuracy is less than 50%.

**Conclusion**

In summary, I have created a hierarchical Poisson model to predict scores—and, consequently, game outcomes—for EPL games for the three seasons starting in 2016 and ending in 2018. The model has an training set prediction accuracy of 663.

**Future Work**

My model can certainly be improved. One major flaw of the model is that it does not account for temporal effects, i.e. differences in team strength across seasons. [8](https://tonyelhabr.rbind.io/post/bayesian-statistics-english-premier-league/#fn8) The consequences of this flaw are compounded by the fact that the pool of teams in each EPL season changes. At the end of each season, the three “worst” EPL teams (by win-loss-tie record) are “relegated” to a secondary league, and, in turn, three secondary league teams are “promoted” to the EPL in their place. [9](https://tonyelhabr.rbind.io/post/bayesian-statistics-english-premier-league/#fn9) Consequently, one might say that the estimates of the teams that do not appear in all seasons are exaggerated.

**Appendix**

**Code**

See all relevant R code below.

library(tidyverse)

# Data Collection

seasons <- 2016L:2018L

scrape\_epl\_data <- function(season = lubridate::year(Sys.Date()) - 1L) {

s1 <- season %>% str\_sub(3, 4) %>% as.integer()

s2 <- s1 + 1L

path <- sprintf('<http://www.football-data.co.uk/mmz4281/%2d%2d/E0.csv>', s1, s2)

data\_raw <- path %>% read\_csv()

data <-

data\_raw %>%

janitor::clean\_names() %>%

mutate\_at(vars(date), ~as.Date(., '%d/%m/%y')) %>%

select(

date,

tm\_h = home\_team,

tm\_a = away\_team,

g\_h = fthg,

g\_a = ftag

) %>%

mutate(

g\_total = g\_h + g\_a,

g\_diff = g\_h - g\_a,

result =

case\_when(

g\_h > g\_a ~ 'h',

g\_h < g\_a ~ 'a',

TRUE ~ 't'

),

tm\_winner =

case\_when(

g\_h > g\_a ~ tm\_h,

g\_h < g\_a ~ tm\_a,

TRUE ~ NA\_character\_

)

) %>%

mutate\_at(vars(matches('season|^g\_')), as.integer)

}

data <-

tibble(season = seasons) %>%

mutate(data = map(season, scrape\_epl\_data)) %>%

unnest(data)

pull\_distinctly <- function (data, var = -1, ..., decreasing = FALSE) {

var <- tidyselect::vars\_pull(names(data), !!rlang::enquo(var))

sort(unique(data[[var]]), decreasing = decreasing, ...)

}

pull2 <- function(data, ...) {

data %>%

pull(...) %>%

as.factor() %>%

as.integer()

}

tms <- data %>% distinct(tm\_h) %>% arrange(tm\_h) %>% pull(tm\_h)

n\_tm <- tms %>% length()

n\_gm <- data %>% nrow()

n\_season <- seasons %>% length()

data\_list <-

list(

g\_h = data %>% pull(g\_h),

g\_a = data %>% pull(g\_a),

tm\_h = data %>% pull2(tm\_h),

tm\_a = data %>% pull2(tm\_a),

season = data %>% pull2(season),

n\_tm = n\_tm,

n\_gm = n\_gm,

n\_season = n\_season

)

data\_list

model\_1 <- glue::glue\_collapse('model {

for(g in 1:n\_gm) {

g\_h[g] ~ dpois(lambda\_h[tm\_h[g], tm\_a[g]])

g\_a[g] ~ dpois(lambda\_a[tm\_h[g], tm\_a[g]])

}

for(h in 1:n\_tm) {

for(a in 1:n\_tm) {

lambda\_h[h, a] <- exp(baseline\_h + (z[h] - z[a]))

lambda\_a[h, a] <- exp(baseline\_a + (z[a] - z[h]))

}

}

z[1] <- 0

for(t in 2:n\_tm) {

z[t] ~ dnorm(z\_all, tau\_all)

}

z\_all ~ dnorm(0, 0.25)

tau\_all <- 1 / pow(sigma\_all, 2)

sigma\_all ~ dunif(0, 2)

baseline\_h ~ dnorm(0, 0.25)

baseline\_a ~ dnorm(0, 0.25)

}')

path\_model\_1 <- 'model\_1.txt'

write\_lines(model\_1, path\_model\_1)

if(eval\_model\_1) {

# inits\_1 <- list(n\_tm = data\_list$n\_tm)

inits\_1 <- NULL

params\_1 <-

c(

paste0('baseline', c('\_a', '\_h')),

paste0('sigma\_all'),

paste0('z', c('', '\_all'))

)

res\_sim\_1 <-

R2OpenBUGS::bugs(

# debug = TRUE,

data = data\_list,

inits = inits\_1,

model.file = path\_model\_1,

parameters.to.save = params\_1,

DIC = FALSE,

n.chains = 1,

n.iter = 10000,

n.burnin = 1000

)

}

# Model 1 Interpretation

z\_var\_lvls <- sprintf('z[%d]', 2:n\_tm)

var\_lvls <- c(paste0('baseline', c('\_a', '\_h')), 'sigma\_all', z\_var\_lvls, 'z\_all')

res\_sim\_summ\_1 <-

res\_sim\_1$summary %>%

as\_tibble(rownames = 'var') %>%

# Re-order these.

mutate\_at(vars(var), ~factor(., levels = var\_lvls)) %>%

arrange(var) %>%

# Then re-coerce var back to its original data type.

mutate\_at(vars(var), as.character)

res\_sim\_summ\_1

tms\_info <-

tibble(tm = tms) %>%

mutate(tm\_idx = row\_number())

tms\_info

# tms\_colors <-

# teamcolors::teamcolors %>%

# as\_tibble() %>%

# filter(league == 'epl') %>%

# select(tm = name, color\_1 = primary) %>%

# mutate\_at(

# vars(tm),

# ~case\_when(

# str\_detect(., 'Bournemouth') ~ 'Bournemouth',

# str\_detect(., 'Brighton') ~ 'Brighton',

# str\_detect(., 'Leicester') ~ 'Leicester',

# str\_detect(., 'Manchester C') ~ 'Man City',

# str\_detect(., 'Manchester U') ~ 'Man United',

# str\_detect(., 'Stoke') ~ 'Stoke',

# str\_detect(., 'Swansea') ~ 'Swansea',

# str\_detect(., 'Tottenham') ~ 'Tottenham',

# str\_detect(., 'West Brom') ~ 'West Brom',

# str\_detect(., 'West Ham') ~ 'West Ham',

# TRUE ~ .

# )

# )

# tms\_colors

# tms\_info %>% anti\_join(tms\_colors)

res\_sim\_summ\_1\_z <-

bind\_rows(

res\_sim\_summ\_1 %>%

filter(var == 'z\_all') %>%

mutate(var = 'z[1]') %>%

mutate(tm\_idx = 1L) %>%

mutate\_at(vars(matches('%$')), ~{. - mean}) %>%

mutate(mean = 0),

res\_sim\_summ\_1 %>%

filter(var %>% str\_detect('^z\\[')) %>%

mutate(

tm\_idx =

var %>%

str\_replace\_all('(^z\\[)([0-9]+)(\\]$)', '\\2') %>%

as.integer()

)

) %>%

left\_join(tms\_info, by = 'tm\_idx') %>%

select(-tm\_idx) %>%

select(tm, everything()) %>%

arrange(tm)

res\_sim\_summ\_1\_z

theme\_custom <- function(...) {

theme\_light(base\_size = 12) +

theme(

legend.position = 'bottom',

legend.title = element\_blank(),

axis.title.x = element\_text(hjust = 1),

axis.title.y = element\_text(hjust = 1),

...

)

}

.lab\_subtitle <- 'Seasons 2016-17 - 2018-19'

.visualize\_res\_sim\_summ <- function(data, ...) {

data %>%

arrange(-mean) %>%

mutate\_at(vars(tm), ~forcats::fct\_reorder(., mean)) %>%

ggplot() +

aes(x = tm) +

geom\_pointrange(aes(y = mean, ymin = `2.5%`, ymax = `97.5%`)) +

theme\_custom() +

coord\_flip() +

labs(

subtitle = .lab\_subtitle ,

caption = glue::glue(

'Posterior mean and 95% equitailed credible set depicted.

{tms[1]} used as "zero"-anchor team.'

),

y = NULL

)

}

visualize\_res\_sim\_summ\_z <- function(data, ...) {

data %>%

.visualize\_res\_sim\_summ() +

geom\_hline(aes(yintercept = 0)) +

labs(

title = glue::glue('Model\'s estimated strength (z)'),

x = 'Posterior mean of z'

)

}

visualize\_res\_sim\_summ\_z\_adj <- function(data, ...) {

data %>%

.visualize\_res\_sim\_summ() +

labs(

title = glue::glue('Model\'s estimated goals scored per game'),

x = 'Posterior mean of z transformed to goals'

)

}

export\_png <-

function(x,

path,

...,

units = 'in',

width = 7,

height = 5) {

ggsave(

plot = x,

filename = path,

units = units,

width = width,

height = height,

...

)

}

n\_sim <- 1000

path\_preds <- file\_path\_out('preds\_1.rds')

eval\_preds <- !fs::file\_exists(path\_preds)

if(eval\_preds) {

.baseline\_h <- res\_sim\_summ\_1 %>% filter(var == 'baseline\_h') %>% pull(mean)

.baseline\_a <- res\_sim\_summ\_1 %>% filter(var == 'baseline\_a') %>% pull(mean)

.extract\_tab\_max <- function(tab) {

tab[ which.max(tab)] %>% names() %>% as.integer()

}

do\_predict <- function(i) {

data\_filt <- data %>% slice(i)

.tm\_h <- data\_filt %>% pull(tm\_h)

.tm\_a <- data\_filt %>% pull(tm\_a)

z\_h <- res\_sim\_summ\_1\_z %>% filter(tm == .tm\_h) %>% pull(mean)

z\_a <- res\_sim\_summ\_1\_z %>% filter(tm == .tm\_a) %>% pull(mean)

g\_h <- rpois(n\_sim, exp(.baseline\_h + (z\_h - z\_a)))

g\_a <- rpois(n\_sim, exp(.baseline\_a + (z\_a - z\_h)))

tab\_h <- table(g\_h)

tab\_a <- table(g\_a)

result\_sign <- sign(g\_h - g\_a)

tab\_result <- table(result\_sign)

result\_mode <- .extract\_tab\_max(tab\_result)

tibble(

g\_h\_mode = .extract\_tab\_max(tab\_h),

g\_a\_mode = .extract\_tab\_max(tab\_a),

result\_mode = result\_mode,

g\_h\_mean = mean(g\_h),

g\_a\_mean = mean(g\_a)

)

}

set.seed(42)

preds <-

tibble(idx = 1:n\_gm) %>%

mutate(res = map(idx, do\_predict)) %>%

unnest(res)

preds

}

preds\_tidy <-

preds %>%

gather(key = 'key', value = 'value', -idx) %>%

select(idx, key, value)

preds\_tidy

.key\_lab\_g\_stem <- 'Team Goals'

.key\_lab\_g\_h\_prefix <- sprintf('Home %s, %%s', .key\_lab\_g\_stem)

.key\_lab\_g\_a\_prefix <- sprintf('Away %s, %%s', .key\_lab\_g\_stem)

keys\_info <-

tribble(

~key, ~key\_lab,

'g\_h\_mode', sprintf(.key\_lab\_g\_h\_prefix, 'Mode'),

'g\_a\_mode', sprintf(.key\_lab\_g\_a\_prefix, 'Mode'),

'g\_h\_mean', sprintf(.key\_lab\_g\_h\_prefix, 'Mean'),

'g\_a\_mean', sprintf(.key\_lab\_g\_a\_prefix, 'Mean'),

'result\_mode', 'Result, Mode'

) %>%

mutate(idx = row\_number()) %>%

mutate\_at(vars(key\_lab), ~forcats::fct\_reorder(., idx)) %>%

select(-idx)

keys\_info

preds\_tidy\_aug <-

preds\_tidy %>%

inner\_join(keys\_info)

preds\_tidy\_aug

preds\_aug <-

preds %>%

inner\_join(data %>% mutate(idx = row\_number())) %>%

mutate\_at(

vars(result\_mode),

~case\_when(

. == 1 ~ 'h',

. == -1 ~ 'a',

. == 0 ~ 't'

)

)

conf\_mat\_tidy <-

preds\_aug %>%

count(result\_mode, result)

conf\_mat\_correct <-

conf\_mat\_tidy %>%

group\_by(is\_correct = result\_mode == result)

conf\_mat\_correct\_summ <-

conf\_mat\_correct %>%

summarise\_at(vars(n), ~sum(.)) %>%

ungroup() %>%

mutate(frac = n / sum(n))

conf\_mat\_correct

conf\_mat\_correct\_h <- conf\_mat\_correct %>% filter(is\_correct, result == 'h')

conf\_mat\_correct\_t <- conf\_mat\_correct %>% filter(is\_correct, result == 't')

conf\_mat\_correct\_a <- conf\_mat\_correct %>% filter(is\_correct, result == 'a')

conf\_mat\_correct\_summ\_yes <- conf\_mat\_correct\_summ %>% filter(is\_correct)

viz\_conf\_mat <-

conf\_mat\_tidy %>%

mutate(frac = n / sum(n)) %>%

mutate(n\_lab = sprintf('%s (%s)', scales::comma(n), scales::percent(frac))) %>%

mutate\_at(

vars(matches('result')),

~forcats::fct\_relevel(., c('h', 't', 'a'))

) %>%

mutate\_at(

vars(matches('result')),

~forcats::fct\_recode(., Home = 'h', Away = 'a', Tie = 't')

) %>%

ggplot() +

aes(x = result\_mode, y = result) +

geom\_tile(aes(fill = n)) +

geom\_text(aes(label = n\_lab), size = 5, fontface = 'bold', color = 'black') +

# scale\_fill\_manual(limits = c(0, 0.5)) +

scale\_fill\_viridis\_c(alpha = 0.5, begin = 0, end = 1, option = 'E') +

theme\_custom() +

theme(

legend.position = 'none',

panel.grid.major = element\_blank(),

panel.grid.minor = element\_blank()

) +

labs(

title = 'Comparison of Predicted and Actual Game Outcomes',

subtitle = .lab\_subtitle,

x = 'Predicted Result',

y = 'Actual Result'

)

viz\_conf\_mat

viz\_g\_mode <-

preds\_tidy\_aug %>%

filter(key %>% str\_detect('^g.\*mode$')) %>%

ggplot() +

aes(x = value, fill = key\_lab) +

scale\_fill\_brewer(palette = 'Set1') +

geom\_bar(position = 'dodge', alpha = 0.8) +

theme\_custom() +

theme(

panel.grid.major.x = element\_blank()

) +

labs(

title = 'Simulated Mode of Goals Scored By Home and Away Teams',

subtitle = .lab\_subtitle,

caption = .lab\_caption\_n\_gm,

x = 'Goals Scored',

y = 'Count of Games'

)

viz\_g\_mode

viz\_g\_mean <-

preds\_tidy\_aug %>%

# This is done so that the `aes()` can be defined before data is actually passed into the whole ggplot pipeline.

filter(row\_number() == 0) %>%

ggplot() +

aes(x = value, fill = key\_lab) +

scale\_fill\_brewer(palette = 'Set1') +

geom\_histogram(data = filter(preds\_tidy\_aug, key == 'g\_h\_mean'), alpha = 1, binwidth = 0.2) +

geom\_histogram(data = filter(preds\_tidy\_aug, key == 'g\_a\_mean'), alpha = 0.5, binwidth = 0.2) +

theme(

panel.grid.major.x = element\_blank()

) +

theme\_custom() +

labs(

title = 'Simulated Mean of Goals Scored By Home and Away Teams',

subtitle = .lab\_subtitle ,

caption = .lab\_caption\_n\_gm,

x = 'Goals Scored',

y = 'Count of Games'

)

viz\_g\_mean

preds\_tidy\_res <-

preds\_tidy %>%

filter(key == 'res\_mode') %>%

mutate\_at(

vars(value),

list(value\_lab = ~case\_when(

. == 1 ~ 'Home Team Wins',

. == 0 ~ 'Draw',

. == -1 ~ 'Away Team Wins'

))

) %>%

mutate\_at(vars(value\_lab), ~forcats::fct\_reorder(., value))

preds\_tidy\_res %>% count(value\_lab)

viz\_result\_mode <-

preds\_tidy %>%

filter(key == 'result\_mode') %>%

mutate\_at(

vars(value),

list(value\_lab = ~case\_when(

. == 1 ~ 'Home Team Wins',

. == 0 ~ 'Draw',

. == -1 ~ 'Away Team Wins'

))

) %>%

mutate\_at(vars(value\_lab), ~forcats::fct\_reorder(., value)) %>%

ggplot() +

aes(x = value\_lab) +

geom\_bar(position = 'dodge') +

theme\_custom() +

theme(

panel.grid.major.x = element\_blank()

) +

labs(

title = 'Simulated Result of Games',

subtitle = .lab\_subtitle ,

caption = .lab\_caption\_n\_gm,

x = NULL,

y = 'Count of Games'

)

viz\_result\_mode

preds\_by\_tm <-

preds\_aug %>%

group\_by(tm\_h) %>%

count(is\_correct = result\_mode == result) %>%

mutate(frac = n / sum(n)) %>%

ungroup() %>%

filter(is\_correct) %>%

arrange(-frac) %>%

mutate(pct = scales::percent(frac)) %>%

select(-frac)