NFL Machine Learning Project

Thomas Zwiller

2025-05-11

```r
#importing nflfastR library
library(nflfastR)
#importing RSQLite (nflfastR is dependent on it)
library(RSQLite)
```

```
## Warning: package 'RSQLite' was built under R version 4.4.1
```

```r
#importing DBI (nflfastR is dependent on it)
library(DBI)
#importing Random Forest for later use
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.4.1
```

```
## randomForest 4.7-1.2
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```r
#importing tidyr to help drop NA values
library(tidyr)
#importing caret to check predictions with a confusion matrix
library(caret)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

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

```r
#importing ggplot to make our plots
library(ggplot2)
#importing ggthemes to make themed plots
library(ggthemes)
```

We used NFLFastR to import our data and then saved the raw data so we could import it each time we needed it moving forward instead of making an API call.

```
NFL_Season_2020 <- readRDS(file = "/Users/TomTheIntern/Desktop/Mendoza/Mod 2/Maching Learning/Project/NF

NFL_Season_2021 <- readRDS(file = "/Users/TomTheIntern/Desktop/Mendoza/Mod 2/Maching Learning/Project/NF

NFL_Season_2022 <- readRDS(file = "/Users/TomTheIntern/Desktop/Mendoza/Mod 2/Maching Learning/Project/NF

NFL_Season_2023 <- readRDS(file = "/Users/TomTheIntern/Desktop/Mendoza/Mod 2/Maching Learning/Project/NF

#we also included the 2000 season for later testing
NFL_Season_2000 <- load_pbp(seasons = 2000, file_type = "rds")
```

We then created our response variable, which was if the home team won or not.

```
#We created our response variable by looking at the final score of each game
NFL_Season_2020$home_win <- ifelse(NFL_Season_2020$home_score > NFL_Season_2020$away_score, 1, 0)
NFL_Season_2021$home_win <- ifelse(NFL_Season_2021$home_score > NFL_Season_2021$away_score, 1, 0)
NFL_Season_2022$home_win <- ifelse(NFL_Season_2022$home_score > NFL_Season_2022$away_score, 1, 0)
NFL_Season_2023$home_win <- ifelse(NFL_Season_2023$home_score > NFL_Season_2023$away_score, 1, 0)
NFL_Season_2000$home_win <- ifelse(NFL_Season_2000$home_score > NFL_Season_2000$away_score, 1, 0)

#and saved the season data as a data frame
NFL_Season_2020 <- as.data.frame(NFL_Season_2020)
NFL_Season_2021 <- as.data.frame(NFL_Season_2021)
NFL_Season_2022 <- as.data.frame(NFL_Season_2022)
NFL_Season_2023 <- as.data.frame(NFL_Season_2023)
NFL_Season_2000 <- as.data.frame(NFL_Season_2000)
```

Here we began to start cleaning our data by experimenting on the NFL_Season_2023 data set.

Our first goal was to remove any categorical data columns that had more than 64 values. The play-by-play data included categories such as the name of the player making the play and a brief description of the play. Because of this, we had to eliminate the columns or risk overwhelming any algorithm we tried to utilize.

We kept factors with 64 or fewer to include the name of each team and whether they were at home or on the road. We eliminated any factors with only one level.

We also wanted to include all the numeric values, like seconds remaining in the half or quarter, which would have been removed if we didn't specifically apply the filter to factor variables.

We also realized that we needed to remove any plays that were "untimed downs", mainly kickoffs and PATs, as the vast majority of those observations contained numerous NA's that made them nearly impossible to include in the model.

```
vals <- rep(NA, ncol(NFL_Season_2023))

for(i in 1:ncol(NFL_Season_2023)){
  vals[i] <- length(unique(NFL_Season_2023[,i]))
}

vals
```

```
##   [1]  4720   285   285    32    32     2    22    32     2    32    33    99
```

```
##  [13]    63   901  1800  3592     3     1    34     2     5     4     2   901
##  [25]  1569    39   123 41921     8   110     2     2     3     2     2     2
##  [37]     3     4    75    76     4     4     4    67     1     1     4     4
##  [49]     3    31    33   390   400     4     4    53    44    48    48    91
##  [61]    53    51    95 40631 40572 40564 40565 40593 40568 40530     1     1
##  [73] 40640 41558 41442 41442 14846 14846 20667 20667 19124 16128 12393  9471
##  [85] 12412 12412  9473  9473 19159 19159 16156 16156 35356 35356 35389 35389
##  [97] 39703 40974 41351 35162 35162 41439 41548 14712 14712 20495 20495  4011
## [109] 15711  3512  9322  3517  3517  9399  9399  4020  4020 16082 16082     3
## [121]     3     3     3     3     3     3     3     3     2     3     3     3
## [133]     3     3     3     2     2     2     2     2     3     3     3     3
## [145]     3     3     3     3     2     2     3     3     3     3     3     3
## [157]     3     3     2     2     3     2     3     3     3     3     3     3
## [169]     3     3   117   117    95   487   478    95   350   346    82     9
## [181]     9     6     4     4     4     1     1   256   251     2     2    81
## [193]    81     1     1     1     1     1     1    40    40    42    42     1
## [205]     1    26    26   630   602     1     1   506   487   133   131    33
## [217]   304   293     4     4     4    33    27  1275    55  1197    55   949
## [229]   895    33   872   828    33     1     1     1     1     1     1     3
## [241]   771   729    33     1     1     1   585   559    98    98    33   283
## [253]   276     8     8     6    33    38   427   412     9     6    10    10
## [265]   387   371   131   131   139   137    33    79    33  1116  1047    48
## [277]     2     3    51     2     2     2     2     9     9     1 15276 16010
## [289]    72     2    11  4720   271 41903    33   268   285     1     1     9
## [301]     2     1 39130     1    34     9  6294    19   521    10     2     2
## [313]     5     5    79    12    15   889   889  1417  1539  3271  3387    40
## [325]    45     2    61    62    53    43     2     4     7    64    20    35
## [337]    35    30    30     2     2   117    29   350    67   475    64     2
## [349]     2     3     2     2   117   357   487   363    67   370   567   579
## [361]   575   588     2     2 41558 17190 15372    25 11114 11656 37714 36760
## [373]     2
```

```r
names(NFL_Season_2023)[vals <= 1]
```

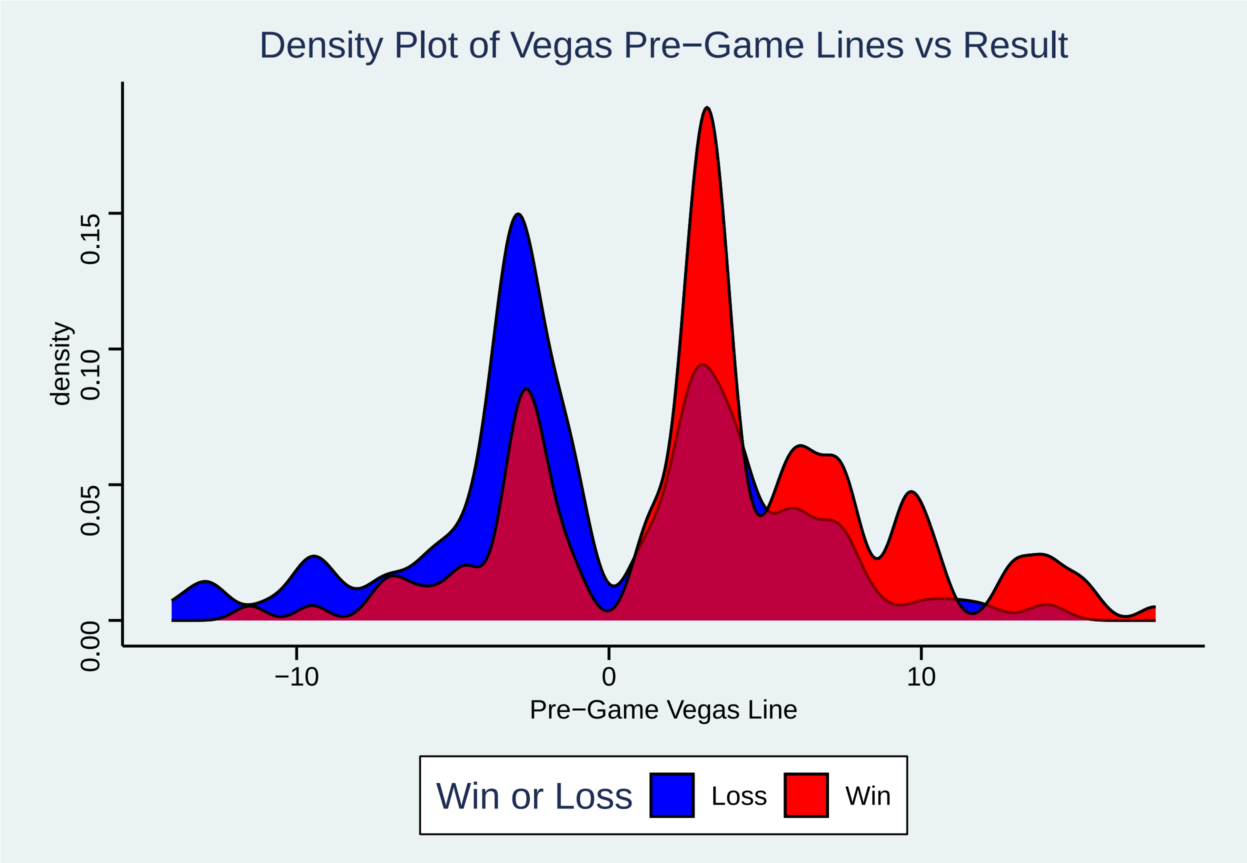
```
##  [1] "quarter_end"
##  [2] "extra_point_result"
##  [3] "two_point_conv_result"
##  [4] "extra_point_prob"
##  [5] "two_point_conversion_prob"
##  [6] "lateral_sack_player_id"
##  [7] "lateral_sack_player_name"
##  [8] "lateral_punt_returner_player_id"
##  [9] "lateral_punt_returner_player_name"
## [10] "kickoff_returner_player_name"
## [11] "kickoff_returner_player_id"
## [12] "lateral_kickoff_returner_player_id"
## [13] "lateral_kickoff_returner_player_name"
## [14] "own_kickoff_recovery_player_id"
## [15] "own_kickoff_recovery_player_name"
## [16] "tackle_for_loss_2_player_id"
## [17] "tackle_for_loss_2_player_name"
## [18] "assist_tackle_3_player_id"
## [19] "assist_tackle_3_player_name"
## [20] "assist_tackle_3_team"
```

```
## [21] "assist_tackle_4_player_id"
## [22] "assist_tackle_4_player_name"
## [23] "assist_tackle_4_team"
## [24] "tackle_with_assist_2_player_id"
## [25] "tackle_with_assist_2_player_name"
## [26] "tackle_with_assist_2_team"
## [27] "season"
## [28] "play_clock"
## [29] "play_deleted"
## [30] "st_play_type"
## [31] "end_yard_line"
```

```r
#this returns factors that are less than or equal to 64 but greater than 1
nfl_factors <- NFL_Season_2023[,which(vals <= 64 & vals > 1 & sapply(NFL_Season_2023, is.factor))]

#this returns just the numeric values
nfl_nums <- NFL_Season_2023[ , sapply(NFL_Season_2023, is.numeric)]

#this binds the two frames together into our usable data
nfl_use <- cbind(nfl_nums, nfl_factors)
```

With cleaned data, we decided to determine key variables. Based on our research, we first checked pre-game Vegas lines.

```r
vegas_spread <- ggplot(NFL_Season_2023, aes(x = spread_line,
                        fill = as.factor(home_win))) + # Set fill as region variable
  geom_density() + # Use geom_density to get density plot
  geom_density(alpha = 0.5) +
  theme_stata() + # Set theme for plot
  theme(panel.grid.major = element_blank(), # Turn of the background grid
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  labs(x = "Pre-Game Vegas Line", # Set plot labels
       title = "Density Plot of Vegas Pre-Game Lines vs Result",
       fill = "Win or Loss") +
  scale_fill_manual(
    values = c("1" = "red", "0" = "blue"),
    labels = c("1" = "Win", "0" = "Loss"))

vegas_spread # Generate plot
```

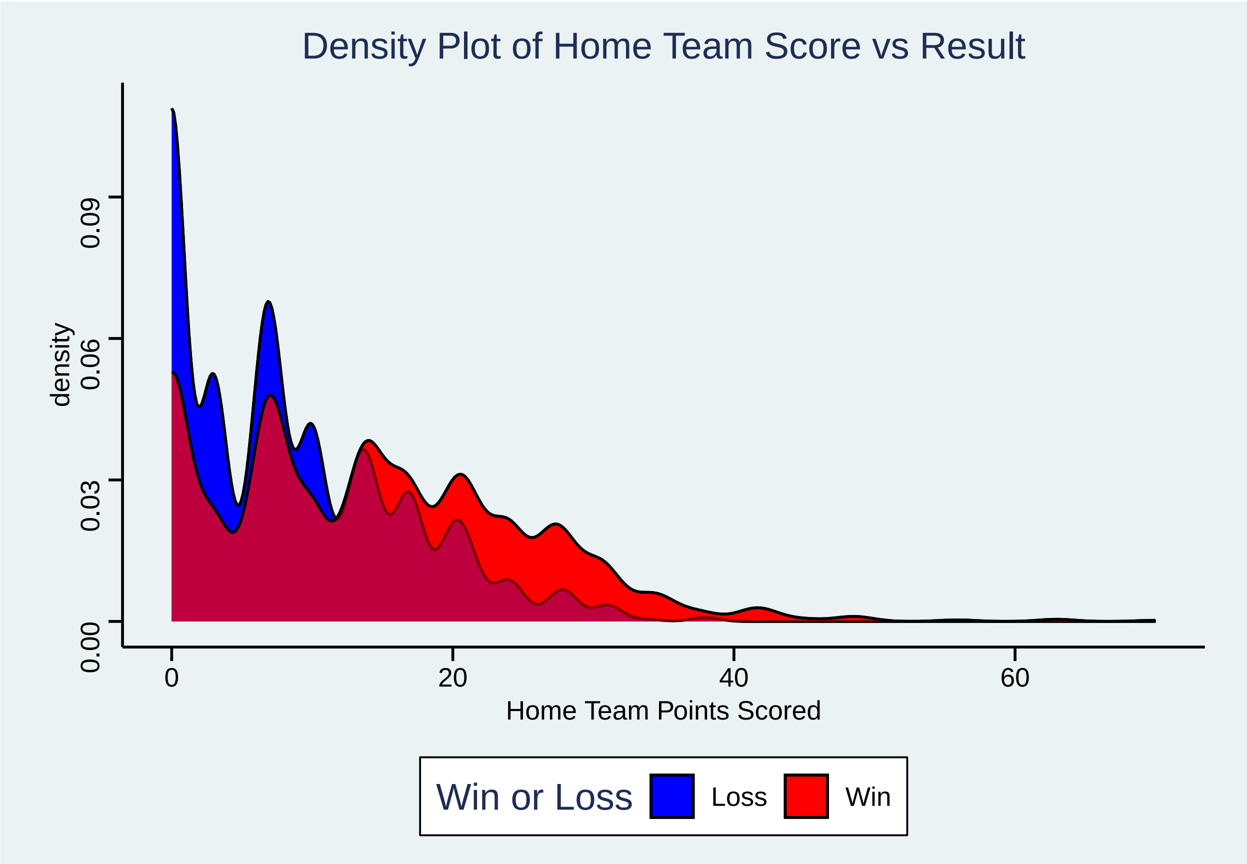


Initially the result might seem counter-intuitive, but the PBP data actually records the number of points favored by as a positive number, while the underdog is associated with a negative value. This is the inverse of how Vegas records its lines.

It's easy to see that generally, teams who are favored were likely to win their game, but it wasn't always the rule.

We also decided to look at the home team score to see if there was a point threshold where the home team became more likely to win.

```
home_team_score <- ggplot(NFL_Season_2023, aes(x = total_home_score,
                          fill = as.factor(home_win))) + # Set fill as region variable
  geom_density() + # Use geom_density to get density plot
  geom_density(alpha = 0.5) +
  theme_stata() + # Set theme for plot
  theme(panel.grid.major = element_blank(), # Turn of the background grid
        panel.grid.minor = element_blank(),
        panel.border = element_blank(),
        panel.background = element_blank()) +
  labs(x = "Home Team Points Scored", # Set plot labels
       title = "Density Plot of Home Team Score vs Result",
       fill = "Win or Loss") +
  scale_fill_manual(
    values = c("1" = "red", "0" = "blue"),
    labels = c("1" = "Win", "0" = "Loss"))

home_team_score
```



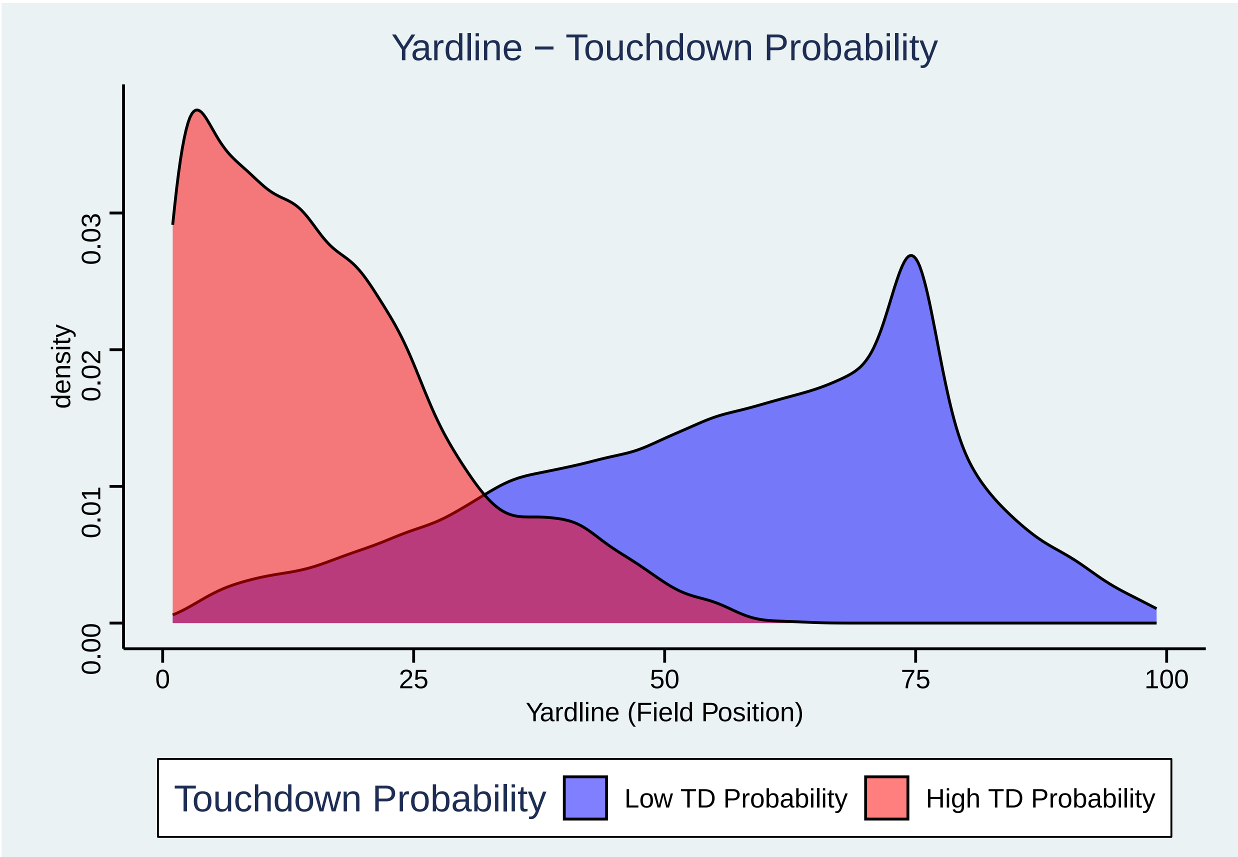
By looking at the plot, it appears that teams who score 15 or fewer points are more likely to lose a given NFL game. However, teams become much more likely to win when they score 16 points or more.

In the NFL, teams tend to score fewer than 40 points, but those who do are much more likely to win.

Finally, we decided to look at touchdown probability for a given drive relative to field position.

```r
# Ensure 'yardline_100' and 'td_prob' are numeric
nfl_use$yardline_100 <- as.numeric(nfl_use$yardline_100)
nfl_use$td_prob <- as.factor(ifelse(nfl_use$td_prob > 0.5, "1", "0"))

# Remove rows with missing values
nfl_use_cleaned <- nfl_use[!is.na(nfl_use$yardline_100) & !is.na(nfl_use$td_prob), ]

# Generate Density Plot
td_prob <- ggplot(nfl_use_cleaned, aes(x = yardline_100, fill = td_prob)) +
  geom_density(alpha = 0.5) +
  theme_stata() + # Simplify theme setup
  theme(
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank()) +
  labs(
    x = "Yardline (Field Position)",
    title = "Yardline - Touchdown Probability",
    fill = "Touchdown Probability") +
```

```r
  scale_fill_manual(
    values = c("1" = "red", "0" = "blue"),
    labels = c("1" = "High TD Probability", "0" = "Low TD Probability"))

# Display the plot
td_prob
```



Based on the graph, we see that there is a low touchdown probability around the 75 yard yard line (the teams own 25). However, once teams get past the 70 (their own 30) the probability begins to decrease before becoming more likely at the opponents 30 yard line. There is a slight dip around the opponents 5 yard line, which likely indicates that teams who get that close to the goal line are more likely to kick a field goal.

```r
nfl_use <- drop_na(nfl_use, posteam)
nfl_use <- drop_na(nfl_use, down)
```

So with out data cleaned and prepared for the analysis, we created our first model, a logistic regression model that accounted for:

The team with the ball The team who was at home The scoring margin at the time of the play The number of seconds remaining in the half The number of seconds remaining in the game The down of the play The yards to go after the play Where the ball was on the field The number of timeouts for the team with the ball The number of timeouts for the team on defense The Vegas spread going into the game

We made the model family binomial to reflect that the outcomes are binary, and trained the model on predicting the chances of the home team winning.

```r
win_model_1.0 <- glm(home_win ~ posteam + home_team + score_differential +
                       half_seconds_remaining + game_seconds_remaining + down
                     + ydstogo + yardline_100 + posteam_timeouts_remaining +
                       defteam_timeouts_remaining +
                       spread_line
                     , data = NFL_Season_2023, family = "binomial")

win_model_1.1 <- glm(home_win ~ posteam + home_team + score_differential +
                       half_seconds_remaining+ posteam_timeouts_remaining +
                       defteam_timeouts_remaining +
                       spread_line
                     , data = NFL_Season_2023, family = "binomial")

summary(win_model_1.1)
```

```
##
## Call:
## glm(formula = home_win ~ posteam + home_team + score_differential +
##     half_seconds_remaining + posteam_timeouts_remaining + defteam_timeouts_remaining +
##     spread_line, family = "binomial", data = NFL_Season_2023)
##
## Coefficients:
##                              Estimate Std. Error z value Pr(>|z|)
## (Intercept)                -1.048e+00  9.933e-02 -10.554  < 2e-16 ***
## posteamATL                  8.387e-01  1.120e-01   7.491 6.83e-14 ***
## posteamBAL                 -1.309e+00  1.171e-01 -11.179  < 2e-16 ***
## posteamBUF                  5.159e-01  1.206e-01   4.277 1.90e-05 ***
## posteamCAR                  1.761e+00  1.364e-01  12.907  < 2e-16 ***
## posteamCHI                  6.006e-01  1.165e-01   5.155 2.54e-07 ***
## posteamCIN                 -2.105e-01  1.121e-01  -1.878 0.060426 .
## posteamCLE                 -1.507e-01  1.147e-01  -1.313 0.189018
## posteamDAL                  9.513e-01  1.223e-01   7.777 7.43e-15 ***
## posteamDEN                 -5.153e-02  1.167e-01  -0.442 0.658710
## posteamDET                 -4.721e-01  1.139e-01  -4.145 3.39e-05 ***
## posteamGB                  -4.794e-02  1.124e-01  -0.427 0.669622
## posteamHOU                 -3.427e-01  1.123e-01  -3.050 0.002285 **
## posteamIND                 -5.295e-01  1.150e-01  -4.604 4.14e-06 ***
## posteamJAX                 -7.582e-01  1.152e-01  -6.583 4.62e-11 ***
## posteamKC                  -5.354e-01  1.173e-01  -4.565 4.99e-06 ***
## posteamLA                  -6.395e-01  1.067e-01  -5.996 2.02e-09 ***
## posteamLAC                  5.564e-01  1.201e-01   4.631 3.64e-06 ***
## posteamLV                   4.615e-01  1.200e-01   3.845 0.000120 ***
## posteamMIA                  6.965e-01  1.225e-01   5.688 1.29e-08 ***
## posteamMIN                 -3.666e-01  1.148e-01  -3.192 0.001413 **
## posteamNE                  -1.561e-01  1.235e-01  -1.265 0.205987
## posteamNO                   3.675e-01  1.147e-01   3.204 0.001356 **
## posteamNYG                  4.003e-01  1.137e-01   3.522 0.000429 ***
## posteamNYJ                  1.606e-01  1.173e-01   1.369 0.170890
## posteamPHI                  4.025e-01  1.145e-01   3.517 0.000437 ***
## posteamPIT                 -9.377e-01  1.095e-01  -8.566  < 2e-16 ***
## posteamSEA                 -3.407e-01  1.103e-01  -3.090 0.002004 **
## posteamSF                  -4.415e-01  1.149e-01  -3.842 0.000122 ***
## posteamTB                  -1.598e-01  1.122e-01  -1.424 0.154537
```

```
## posteamTEN                  4.582e-01  1.191e-01   3.849 0.000119 ***
## posteamWAS                  3.962e-01  1.187e-01   3.338 0.000845 ***
## home_teamATL                2.550e-01  1.132e-01   2.253 0.024261 *
## home_teamBAL                1.213e+00  1.174e-01  10.327  < 2e-16 ***
## home_teamBUF                3.568e-01  1.234e-01   2.892 0.003827 **
## home_teamCAR               -1.359e+00  1.351e-01 -10.057  < 2e-16 ***
## home_teamCHI                6.267e-01  1.149e-01   5.456 4.87e-08 ***
## home_teamCIN                1.215e+00  1.136e-01  10.693  < 2e-16 ***
## home_teamCLE                2.868e+00  1.310e-01  21.900  < 2e-16 ***
## home_teamDAL                1.266e+00  1.353e-01   9.364  < 2e-16 ***
## home_teamDEN                5.262e-01  1.159e-01   4.540 5.63e-06 ***
## home_teamDET                1.386e+00  1.240e-01  11.177  < 2e-16 ***
## home_teamGB                 1.128e+00  1.149e-01   9.819  < 2e-16 ***
## home_teamHOU                1.515e+00  1.114e-01  13.596  < 2e-16 ***
## home_teamIND                5.692e-01  1.114e-01   5.109 3.24e-07 ***
## home_teamJAX                2.550e-01  1.190e-01   2.143 0.032076 *
## home_teamKC                 6.423e-01  1.213e-01   5.297 1.18e-07 ***
## home_teamLA                 1.199e+00  1.126e-01  10.644  < 2e-16 ***
## home_teamLAC               -1.461e+00  1.260e-01 -11.597  < 2e-16 ***
## home_teamLV                 1.014e+00  1.157e-01   8.765  < 2e-16 ***
## home_teamMIA                2.564e-01  1.283e-01   1.999 0.045582 *
## home_teamMIN               -4.549e-01  1.194e-01  -3.809 0.000139 ***
## home_teamNE                -1.628e+00  1.320e-01 -12.332  < 2e-16 ***
## home_teamNO                 1.673e-01  1.187e-01   1.409 0.158702
## home_teamNYG                6.460e-01  1.086e-01   5.948 2.71e-09 ***
## home_teamNYJ                4.136e-01  1.116e-01   3.705 0.000211 ***
## home_teamPHI                7.830e-01  1.248e-01   6.272 3.57e-10 ***
## home_teamPIT                1.292e+00  1.108e-01  11.667  < 2e-16 ***
## home_teamSEA                1.015e+00  1.134e-01   8.949  < 2e-16 ***
## home_teamSF                 6.656e-01  1.251e-01   5.322 1.03e-07 ***
## home_teamTB                 5.483e-01  1.133e-01   4.838 1.31e-06 ***
## home_teamTEN                5.782e-01  1.131e-01   5.115 3.14e-07 ***
## home_teamWAS               -1.792e+00  1.311e-01 -13.672  < 2e-16 ***
## score_differential          8.889e-03  1.173e-03   7.576 3.57e-14 ***
## half_seconds_remaining     -1.232e-04  2.557e-05  -4.817 1.45e-06 ***
## posteam_timeouts_remaining  1.286e-01  1.840e-02   6.988 2.79e-12 ***
## defteam_timeouts_remaining  1.365e-01  1.907e-02   7.161 8.02e-13 ***
## spread_line                 1.143e-01  3.451e-03  33.110  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 57462  on 41924  degrees of freedom
## Residual deviance: 46377  on 41857  degrees of freedom
## AIC: 46513
##
## Number of Fisher Scoring iterations: 5
```

Based on the initial results, which team has the ball and at home was generally considered to be statistically significant. This makes sense, as in the one season we examined, some teams finished with a losing record while others finished with a winning record. This does mean that the model will be incredibly biased towards teams if we were to deploy it, so we need to include multiple years of data, and likely update it after each week of the regular season.

game_seconds_remaining was less significant than half_seconds_remaining, so we may need to consider removing game_seconds remaining and only use half_seconds remaining

The down was considered to be insignificant, and may need to be removed.

Both timeouts for the team on offense and the team on defense was highly significant.

The spread_line (the Vegas spread) was highly significant.

The down and field position (yardline_100) were considered insignificant and might be removed when we add in the other data sets down the road.

The AIC of the model was 46,511, which seems high but we have 32 variables for the posteam variable and another 32 for the home team, which are making that figure rather large. Earlier iterations of the model had an AIC of 50,000+.

Now we can try training the model and then testing it.

```
set.seed(111111)
#getting the number of observations
num_obs <- nrow(nfl_use)
#getting a random set of rows for training data
train_data_rows <- sample(1:num_obs, 0.80*num_obs)
#creating testing data
train_data <- nfl_use[train_data_rows , ]
#using the remaining rows for testing data
test_data <- nfl_use[-train_data_rows , ]
```

Let's re-train the model on the training set.

```
win_model_1.0.1 <- glm(home_win ~ posteam + home_team + score_differential +
                         half_seconds_remaining + game_seconds_remaining + down
                       + ydstogo + yardline_100 + posteam_timeouts_remaining +
                         defteam_timeouts_remaining +
                         spread_line
                       , data = train_data, family = "binomial")
```

And then test the model using the unseen data.

```
#Making predictions
pred_1 <- predict(win_model_1.1, newdata = test_data)
#converting them out of log and into normalized %s
pred_1 <- 1 / (1 + exp(-pred_1))
#converting to wins if above 50%
pred_1 <- ifelse(pred_1 >= 0.5, 1, 0)

pred_1 <- as.factor(pred_1)

pred_1 <- unname(pred_1)

test_data$home_win <- as.factor(test_data$home_win)

confusionMatrix(test_data$home_win, pred_1, positive = "1")
```

```
## Confusion Matrix and Statistics
##
```

```
##           Reference
## Prediction    0    1
##          0 2185 1509
##          1  917 3774
##
##                Accuracy : 0.7107
##                  95% CI : (0.7008, 0.7204)
##     No Information Rate : 0.6301
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4029
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7144
##             Specificity : 0.7044
##          Pos Pred Value : 0.8045
##          Neg Pred Value : 0.5915
##              Prevalence : 0.6301
##          Detection Rate : 0.4501
##    Detection Prevalence : 0.5595
##       Balanced Accuracy : 0.7094
##
##        'Positive' Class : 1
##
```

Our initial model had an accuracy of 71.07%, but struggled to pick games in which team actually wound up winning, suggesting the model struggles with teams who are able to complete a comeback victory.

After experimenting with a simple logistic regression model, we decided to see if we could gain any additional insight from using a RandomForest model. This did mean that we needed to clean our data a little bit differently because we could now try and use different predictor variables.

```
#setting the seed for repeat-ability
set.seed(111111)

#binding all four of the NFL seasons
total_data <- rbind(NFL_Season_2020, NFL_Season_2021, NFL_Season_2022, NFL_Season_2023, NFL_Season_2000)

#setting any NA data points into unknowns
total_data[is.na(total_data)] <- "unknown"

total_data[] <- lapply(total_data, function(col) {
  if (!is.numeric(col) & !is.integer(col)) {
    col <- factor(col)
  }
  return(col)
})

for(i in 1:ncol(total_data)){
  vals[i] <- length(unique(total_data[,i]))
}

vals
```

```
##   [1]   5061   1380   1380     32     32      2     22     34      3     33
##  [11]     36    100    289    902   1802   3602      3      2     38      2
##  [21]      5      5      3   1503   1570     44    127 218029     10    120
##  [31]      2      2      3      2      2      2      3      4     84     90
##  [41]      4      4      4     85      4      3      4      4      3     33
##  [51]     33   1181   1266      5      5     60     50     59     57    100
##  [61]     61     59    101 196522 196052 196151 196073 195789 196057 195385
##  [71]     23      2 196823 206778 211260 211262  72547  72547 100156 100156
##  [81]  76216  64473  49495  38104  49636  49636  38118  38118  76457  76457
##  [91]  64614  64614 151521 151521 151953 151953 189202 204852 206244 150544
## [101] 150548 209079 210040  71718  71718  99154  99154  15769  60604  13771
## [111]  36839  13849  13849  37754  37754  15887  15887  64261  64261      3
## [121]      3      3      3      3      3      3      3      3      2      3
## [131]      3      3      3      3      3      3      3      3      3      3
## [141]      3      3      3      3      3      3      3      3      3      2
## [151]      3      3      3      3      3      3      3      3      3      3
## [161]      3      3      3      3      3      3      3      3      3      3
## [171]    337    335    105   1349   1260    104   1005    958    107     50
## [181]     50     23     14     14     10      1      1    788    764     12
## [191]     12    306    305      9      9    506    529     16     16    106
## [201]    106    173    174     34     34    156    154   1641   1509      1
## [211]      1   1340   1255    380    370     33   1095   1022     19     24
## [221]     24     33     33   3716    482   3246    463   2575   2317     33
## [231]   2163   1965     33      2      2      2      2      2      2      3
## [241]   2102   1902     33      2      2      2   1585   1472    290    286
## [251]     33    973    914     42     42     23     33     79   1728   1579
## [261]     27     16     40     40   1131   1071    400    389    396    383
## [271]     33    119     33   3371   2983     53      2      4     64      3
## [281]      3      2      2     46     48      5  50367  55023     82      2
## [291]     12   5031   1067 181448     36   1057   1124      2      2     19
## [301]      3      1  76014      1     37     10  24480     24    613     11
## [311]      3      3      6      6     91     29     31   1498   1499   1575
## [321]   1576   4525   4644     48     54      2     81     78     70     58
## [331]      2      4      8     82     31     93     93     50     62      2
## [341]      3    330     41    998     85   1251     83      2      2      3
## [351]      2      2    337   1052   1348   1041     85   1101   1486   1602
## [361]   1527   1651      2      2 206777  68688  53610     30  39288  40057
## [371] 147796 144594      2
```

```r
high_drop_names <- names(total_data)[vals >= 33]

low_drop_names <- names(total_data)[vals < 3]

# Drop the specified columns from the dataset
cleaned_data <- total_data[, !(colnames(total_data) %in% high_drop_names)]

cleaned_data <- cleaned_data[, !(colnames(cleaned_data) %in% low_drop_names)]

cleaned_data$home_win <- as.factor(ifelse(total_data$home_score > total_data$away_score, 1, 0))

cleaned_data$home_score <- total_data$home_score
cleaned_data$away_score <- total_data$away_score
cleaned_data$game_seconds_remaining <- as.numeric(total_data$game_seconds_remaining)
cleaned_data$spread_line <- total_data$spread_line
```

```r
cleaned_data$old_game_id <- as.numeric(total_data$old_game_id)
cleaned_data$yardline_100 <- as.numeric(total_data$yardline_100)
cleaned_data$total_home_score <- as.numeric(total_data$total_home_score)
cleaned_data$total_away_score <- as.numeric(total_data$total_away_score)
cleaned_data$half_seconds_remaining <- as.numeric(total_data$half_seconds_remaining)

# Define the columns to drop
drop_column_names <- c("lateral_receiving_yards",
          "lateral_rusher_player_id",
          "lateral_rusher_player_name",
          "lateral_interception_player_id",
          "lateral_interception_player_name",
          "lateral_punter_returner_player_id",
          "lateral_punt_returner_player_name",
          "home_score",
          "away_score")

# Remove the specified columns from cleaned_data
cleaned_data <- cleaned_data[, !(names(cleaned_data) %in% drop_column_names)]
```

Let's try training the random forest model, and then making predictions.

```r
set.seed(111111)
#getting the number of observations
num_obs <- nrow(cleaned_data)
#getting a random set of rows for training data
train_data_rows <- sample(1:num_obs, 0.50*num_obs)
#creating testing data
train_data <- cleaned_data[train_data_rows , ]
#using the remaining rows for testing data
test_data <- cleaned_data[-train_data_rows , ]

win_tree_model <- randomForest(home_win ~ . ,
                               data = train_data,
                               mtry = floor(ncol(train_data) * 0.333),
                               ntree = 200,
                               nodesize = 5,
                               progress = TRUE)

#Making predictions
pred_1 <- predict(win_tree_model, newdata = test_data, type = "prob")

#converting to wins if above 50%
pred_1 <- as.factor(ifelse(pred_1[, 2] >= 0.5, 1, 0))
confusionMatrix(test_data$home_win, pred_1, positive = "1")
```

Two things are happening here. One, we are over fitting our data by limiting the node size to 5 when we are training on nearly 80k plays. We also used 200 trees. As a result, the model was able to predict every single instance correctly. Not good.

We tried to correct this with our next model, but issue number two, which was much less obvious, was still a huge problem. That will be explained after the next chunk.

```r
#first, we set up a new data frame to serve as the basis for the next model.
cleaned_data_2 <- cleaned_data

#and dropped any columns that had little to no statistical significance.
drop_column_names_2 <- c(
  "qb_dropback",
  "field_goal_result",
  "first_down_rush",
  "first_down_penalty",
  "third_down_failed",
  "fourth_down_failed",
  "punt_in_endzone",
  "punt_out_of_bounds",
  "punt_downed",
  "solo_tackle",
  "lateral_reception",
  "lateral_return",
  "lateral_recovery",
  "forced_fumble_player_2_team",
  "forced_fumble_player_2_player_id",
  "forced_fumble_player_2_player_name",
  "tackle_with_assist",
  "fumbled_2_team",
  "fumble_recovery_2_yards",
  "lateral_rushing_yards",
  "fumble_recovery_2_player_id",
  "lateral_punt_returner_player_id"
)



#we then filtered the names out
cleaned_data_2 <- cleaned_data_2[, !(names(cleaned_data_2) %in% drop_column_names_2)]

#creating testing data
train_data <- cleaned_data_2[train_data_rows , ]
#using the remaining rows for testing data
test_data <- cleaned_data_2[-train_data_rows , ]

win_tree_model_2 <- randomForest(home_win ~ . ,
                              data = train_data,
                              mtry = sqrt(ncol(train_data)),
                              ntree = 100,
                              nodesize = 250,
                              progress = TRUE)

pred_2 <- predict(win_tree_model_2, newdata = test_data, type = "prob")
#converting to wins if above 50%
pred_2 <- as.factor(ifelse(pred_2[, 2] >= 0.5, 1, 0))

confusionMatrix(test_data$home_win, pred_2, positive = "1")
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction     0     1
##          0 43024 48535
##          1 19032 90346
##
##                Accuracy : 0.6637
##                  95% CI : (0.6617, 0.6658)
##     No Information Rate : 0.6912
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3039
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.6505
##             Specificity : 0.6933
##          Pos Pred Value : 0.8260
##          Neg Pred Value : 0.4699
##              Prevalence : 0.6912
##          Detection Rate : 0.4496
##    Detection Prevalence : 0.5443
##       Balanced Accuracy : 0.6719
##
##        'Positive' Class : 1
##
```

Despite our best efforts to guard against over fitting, the model was still over fitting, by a lot. This is where realization number two happened.

We realized that because we were asking the model to make game predictions and then feeding it individual pieces from each game, we essentially gave the model the answers and asked it to repeat them back to us.

The problem wasn't that we were conducting random samples, it was the way in which we pulled those samples. We needed to give the model whole games and then test it on unseen whole games as opposed to giving it portions of each game and then giving it unseen portions from the same game.

The difference is essentially the same as if I gave you pages out of a book and asked you to predict how the last chapter finished. You would be able to tell me major plot points from the book you were reading, as well as themes and character names, which would be helpful and you might get some things right.

However, this is like me giving you random pages from a book and then asking you to tell me how Chapter 7 started and ended. You might not have the exact pages that the chapter began and finished on, but you had pages from THAT chapter, as well as pages before the chapter (so you should know how it would begin) and pages after the chapter (so you should know how it would end).

We realized that the model was able to learn how to make predictions based off of the match-up than the actual game situation, so we needed to train it on the first two seasons (2020 and 2021) and then test it on the next two unseen seasons (2022 and 2023). Like in the book example, some things would carry over, like the Chiefs being a good football team, the Jets being rather poor, etc. But, while the match-ups would include the same teams, it wouldn't include an identical result, and the model would have to learn to go off of game situation rather than just the two teams.

```
#this was our final removal of columns that had little predictive power
cleaned_final_names <- c(
"punt_blocked",
```

```
"first_down_pass",
"third_down_converted",
"fourth_down_converted",
"incomplete_pass",
"interception",
"punt_inside_twenty",
"fumble_forced",
"fumble_out_of_bounds",
"safety",
"penalty",
"fumble_lost",
"qb_hit",
"pass_attempt",
"return_touchdown",
"field_goal_attempt",
"punt_attempt",
"fumble",
"complete_pass",
"assist_tackle",
"lateral_rush",
"fumble_recovery_2_team",
"fumble_recovery_2_player_name",
"replay_or_challenge_result",
"lateral_kickoff_returner_player_id",
"lateral_kickoff_returner_player_name",
"defensive_two_point_conv",
"old_game_id",
"kickoff_downed",
"kickoff_fair_catch",
"own_kickoff_recovery_player_id",
"own_kickoff_recovery_player_name",
"defensive_two_point_attempt",
"own_kickoff_recovery",
"xyac_median_yardage",
"down",
"punt_fair_catch",
"kickoff_in_endzone",
"tackled_for_loss",
"success",
"play_type_nfl",
"series_result",
"fumble_not_forced",
"kickoff_out_of_bounds",
"timeout",
"extro_point_prob",
"defteam_timeouts_remaining",
"run_gap",
"game_half",
"play_type",
"drive_inside20",
"drive_ended_with_score",
"drive_quarter_start",
"drive_start_transition",
```

```r
"kickoff_inside_twenty",
"first_down",
"pass_touchdown",
"rush_touchdown",
"sack",
"extra_point_attempt",
"two_point_attempt",
"touchdown",
"extra_point_prob",
"extra_point_result",
"kickoff_attempt",
"two_point_conv_result",
"rush_attempt",
"run_location",
"ydstogo",
"drive_play_count",
"fixed_drive_result",
"drive_first_downs",
"drive_quarter_end",
"drive_end_transition"
)

cleaned_final <- cleaned_data_2[, !(names(cleaned_data_2) %in% cleaned_final_names)]
```

This is where the magic happens. We partitioned our data along seasons instead of doing so randomly, meaning that the data we were testing our model was unseen at a game level, not just a play level.

```r
#training data
NFL_2020_NFL_2021 <- cleaned_final[cleaned_final$season == 2020 |
                                   cleaned_final$season == 2021 | cleaned_final$season == 2000, ]

#testing data
NFL_2022_NFL_2023 <- cleaned_final[cleaned_final$season == 2022 |
                                   cleaned_final$season == 2023,]

#2000 Season Testing Data
NFL_2000_Test <- cleaned_final[cleaned_final$season == 2000 , ]
```

Now we could begin trying to train a few different models with different parameters.

Our first was just a general model, guarded against over fitting.

```r
Test_Try <- randomForest(home_win ~ .,
                         data = NFL_2020_NFL_2021,
                         mtry = ncol(NFL_2020_NFL_2021) * .333,
                         ntree = 150,
                         nodesize = 750,
                         progress = TRUE)

Test_Try$importance
```

```
##                              MeanDecreaseGini
## home_team                        8.842131e+03
```

```
## away_team                      1.054881e+04
## week                           1.652011e+03
## posteam_type                   6.100307e-01
## qtr                            2.900133e+02
## goal_to_go                     1.719181e-01
## pass_length                    9.386563e-01
## pass_location                  6.045776e-01
## home_timeouts_remaining        2.028941e+01
## away_timeouts_remaining        5.666308e+01
## posteam_timeouts_remaining     3.610871e+00
## season                         3.262097e+02
## special_teams_play             1.200554e+02
## roof                           2.554512e+02
## surface                        9.589225e+02
## wind                           5.888481e+03
## game_seconds_remaining         4.893152e+01
## spread_line                    7.759647e+03
## yardline_100                   7.625680e-02
## total_home_score               7.114181e+03
## total_away_score               6.636463e+03
## half_seconds_remaining         4.545255e+00
```

This version removed the season and then the week.

```
#the chosen one
NFL2020_NFL2021_Tree <- randomForest(home_win ~ . -season -week,
                              data = NFL_2020_NFL_2021,
                              mtry = sqrt(ncol(NFL_2020_NFL_2021)),
                              ntree = 150,
                              nodesize = 1000,
                              maxnodes = 60)

NFL2020_NFL2021_Tree$importance
```

```
##                            MeanDecreaseGini
## home_team                      5038.3576977
## away_team                      5881.1330488
## posteam_type                      3.8311851
## qtr                             306.8028307
## goal_to_go                        0.0000000
## pass_length                       3.8263425
## pass_location                     3.0084377
## home_timeouts_remaining          45.1825040
## away_timeouts_remaining          95.8788249
## posteam_timeouts_remaining       12.1346791
## special_teams_play              104.8008600
## roof                            191.9310278
## surface                         511.1552960
## wind                           3024.7709492
## game_seconds_remaining           64.2763687
## spread_line                    6068.9063931
## yardline_100                      0.4314791
## total_home_score               5618.0322325
```

```
## total_away_score              5297.6194059
## half_seconds_remaining           6.9757183
```

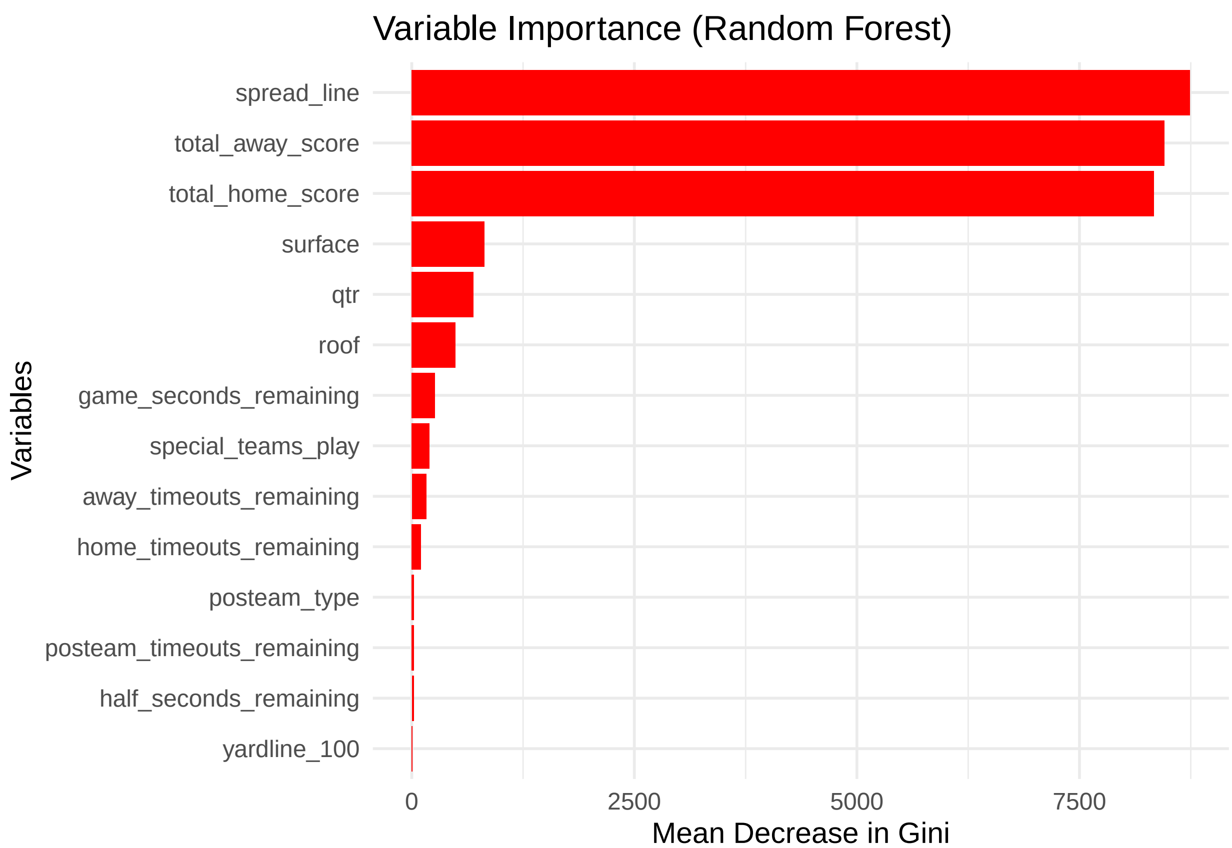
One thing re realized is that while the home team and the away team were incredibly strong predictors, this likely created bias within the model. When we tested the current iteration against the 2000 season, the model's performance dropped off rather significantly.

So we tried a model that did not include the home and away teams, making it focus more on the on the field product, but also included the spread which served as a control for opponent quality.

```r
set.seed(111111)
#no home or away team present
no_team_model <- randomForest(home_win ~ . -season -week
                              -home_team -away_team
                              -wind -goal_to_go -pass_length
                              -pass_location,
                              data = NFL_2020_NFL_2021,
                              mtry = sqrt(ncol(NFL_2020_NFL_2021)),
                              ntree = 150,
                              nodesize = 2000,
                              maxnodes = 90)

no_team_model$importance
```

```
##                             MeanDecreaseGini
## posteam_type                       25.999344
## qtr                               693.299530
## home_timeouts_remaining           104.574883
## away_timeouts_remaining           162.517792
## posteam_timeouts_remaining         22.964668
## special_teams_play                199.197617
## roof                              493.759055
## surface                           817.013829
## game_seconds_remaining            259.764270
## spread_line                      8741.097278
## yardline_100                        7.969758
## total_home_score                 8338.523346
## total_away_score                 8453.628034
## half_seconds_remaining             22.513243
```

```r
importance_df <- as.data.frame(no_team_model$importance)

importance_df$Variable <- rownames(importance_df)

ggplot(importance_df, aes(x = reorder(Variable, MeanDecreaseGini), y = MeanDecreaseGini)) +
  geom_bar(stat = "identity", fill = "red") +
  coord_flip() +
  labs(title = "Variable Importance (Random Forest)",
       x = "Variables",
       y = "Mean Decrease in Gini") +
  theme_minimal()
```



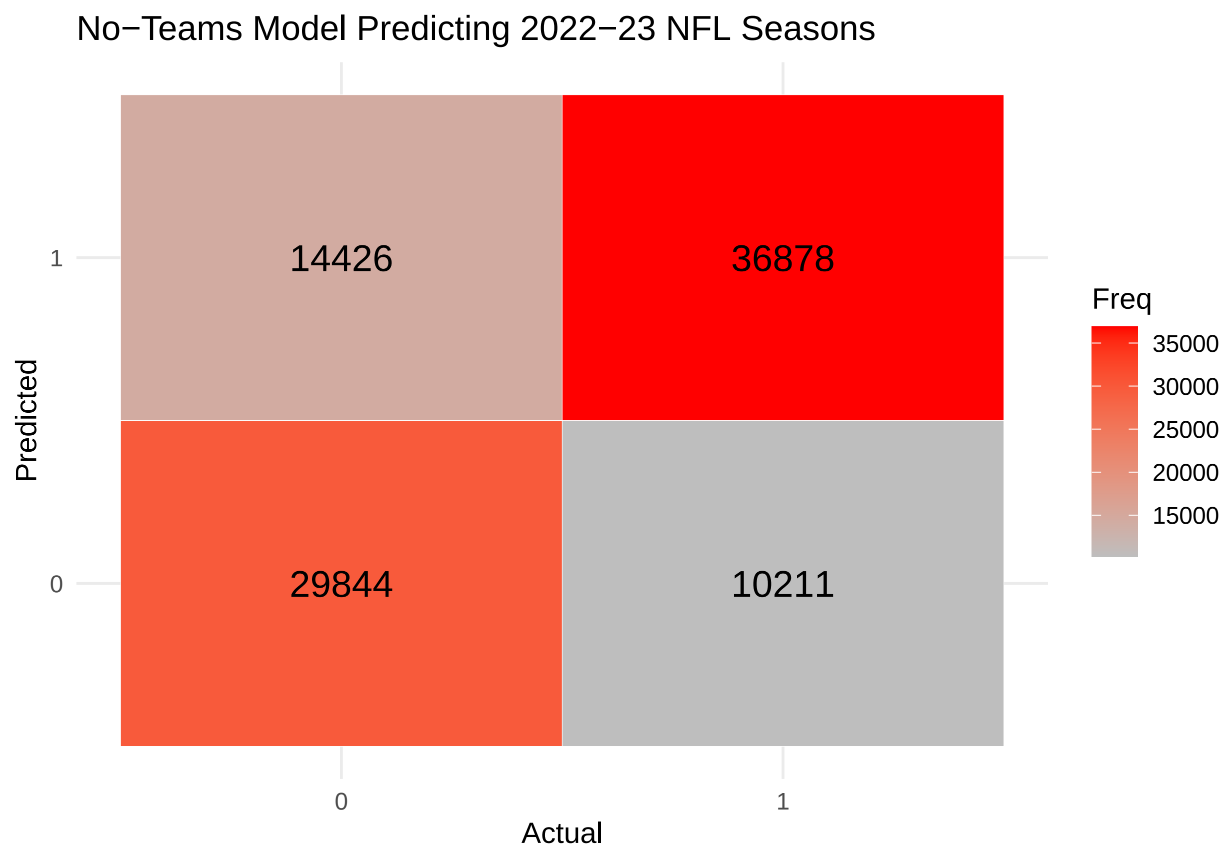
Now with a version of the model we can trust, we can start making predictions and grading the models performance.

```
pred_3 <- predict(no_team_model, newdata = NFL_2022_NFL_2023, type = "prob")

#converting to wins if above 50%
pred_3_fact <- as.factor(ifelse(pred_3[, 2] >= 0.5, 1, 0))

confusionMatrix(NFL_2022_NFL_2023$home_win, pred_3_fact, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction      0     1
##           0 29844 10211
##           1 14426 36878
##
##                  Accuracy : 0.7303
##                    95% CI : (0.7274, 0.7332)
##       No Information Rate : 0.5154
##       P-Value [Acc > NIR] : < 2.2e-16
##
##                     Kappa : 0.4586
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
```

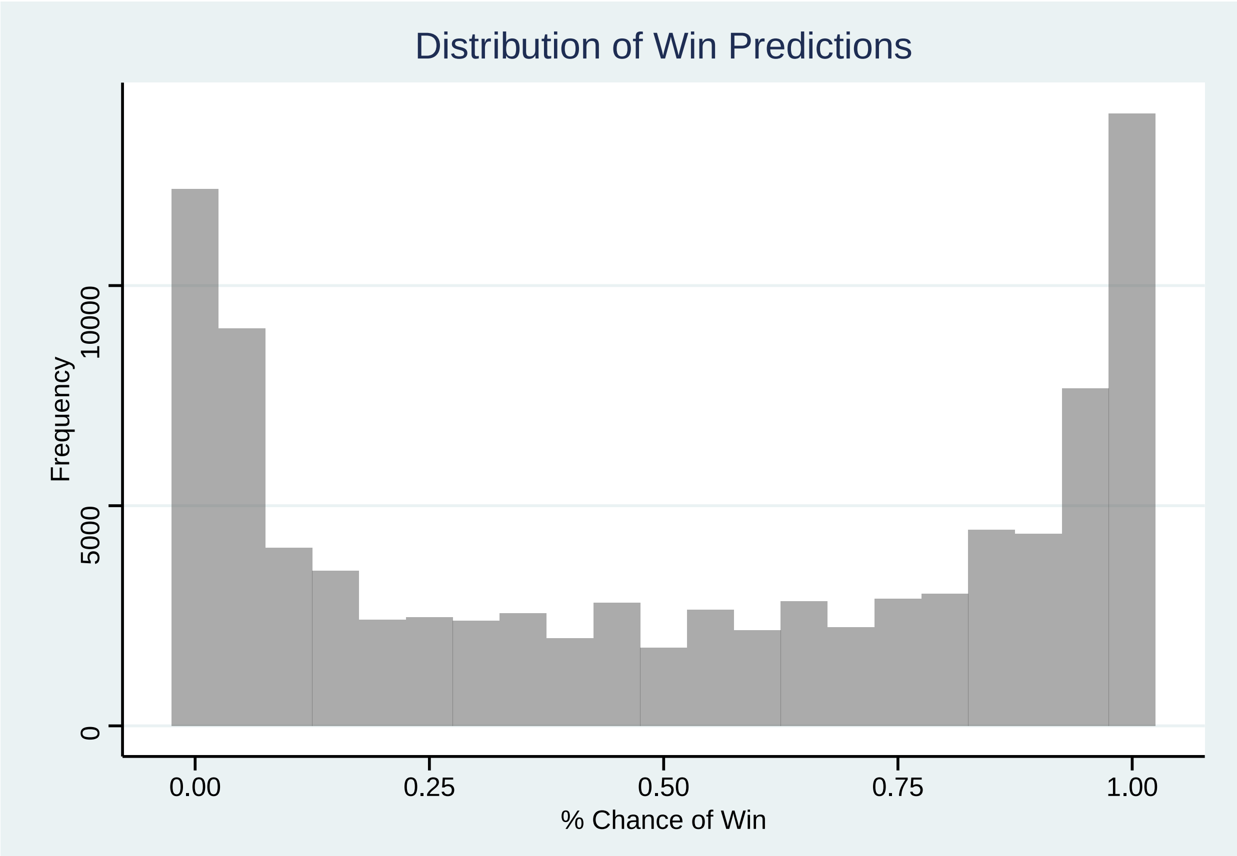
```
##             Sensitivity : 0.7832
##             Specificity : 0.6741
##          Pos Pred Value : 0.7188
##          Neg Pred Value : 0.7451
##              Prevalence : 0.5154
##          Detection Rate : 0.4037
##    Detection Prevalence : 0.5616
##       Balanced Accuracy : 0.7286
##
##        'Positive' Class : 1
##
```

```r
pred_3_confusion <- confusionMatrix(NFL_2022_NFL_2023$home_win, pred_3_fact, positive = "1")

pred_3_table <- as.data.frame(pred_3_confusion$table)

ggplot(pred_3_table, aes(x = Reference, y = Prediction, fill = Freq)) +
  #setting our tile coloring
  geom_tile(color = "white") +
  #setting the color and size of the text
  geom_text(aes(label = Freq), color = "black", size = 5) +
  #setting the color of our gradient
  scale_fill_gradient(low = "gray", high = "red") +
  #adding in labels
  labs(title = "No-Teams Model Predicting 2022-23 NFL Seasons", x = "Actual", y = "Predicted") +
  #plus our theme
  theme_minimal()
```



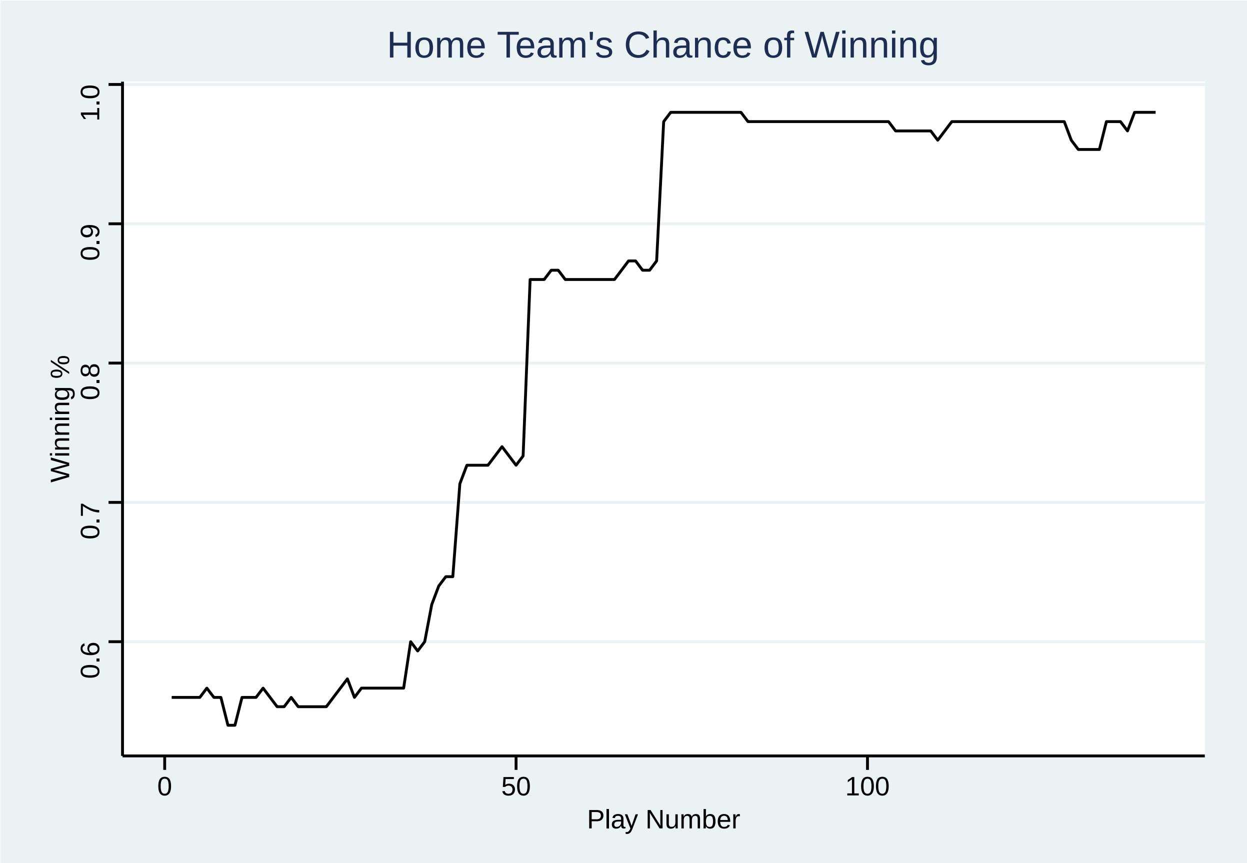
We then looked at the distribution of the %'s for a given win prediction.

```
library(ggplot2)

#Converting to a data frame
pred_df <- data.frame(class_0 = pred_3[, 1], class_1 = pred_3[, 2])

#And then plotting a historgram of the win predictions
ggplot(pred_df, aes(x = class_1)) +
  geom_histogram(binwidth = 0.05, alpha = 0.5) +
  labs(title = "Distribution of Win Predictions", x = "% Chance of Win", y = "Frequency") +
  theme_stata()
```



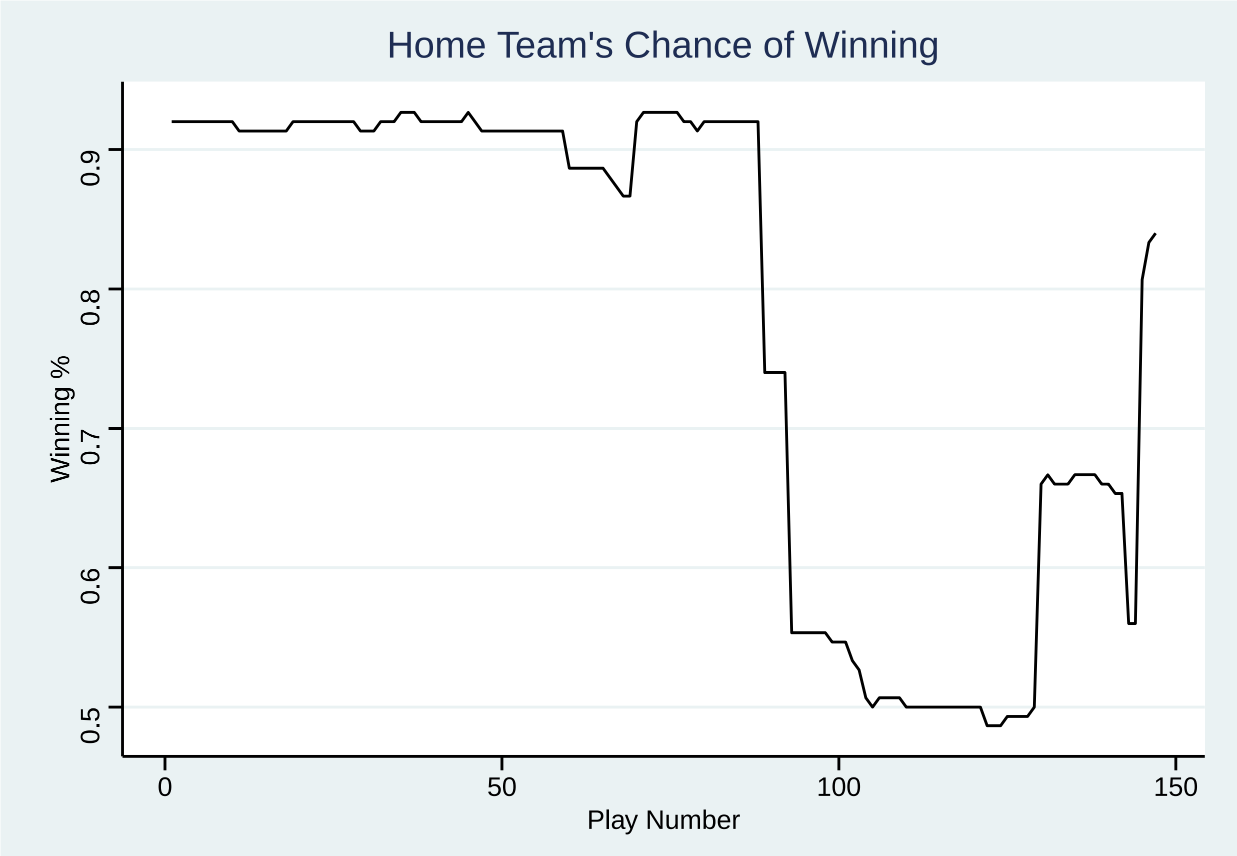
We also decided to test on a few specific games just to see how the model measured up against ESPN's FPI. The first was Denver at Detroit in 2023.

```
game_test <- NFL_2022_NFL_2023[NFL_2022_NFL_2023$home_team == "DET" & NFL_2022_NFL_2023$away_team == "DE

game_test_pred <- predict(no_team_model, newdata = game_test, type = "prob")

# Assuming game_pred_list is a list of predictions
game_pred_df <- data.frame(game_pred = unlist(game_test_pred))  # Convert list to data frame

# Plotting the predictions
ggplot(data = game_pred_df, aes(x = seq_along(game_pred.1), y = game_pred.1)) +
  geom_line() +
  labs(x = "Play Number", y = "Winning %", title = "Home Team's Chance of Winning") +
  theme_stata() +
  theme(panel.grid = element_blank())
```



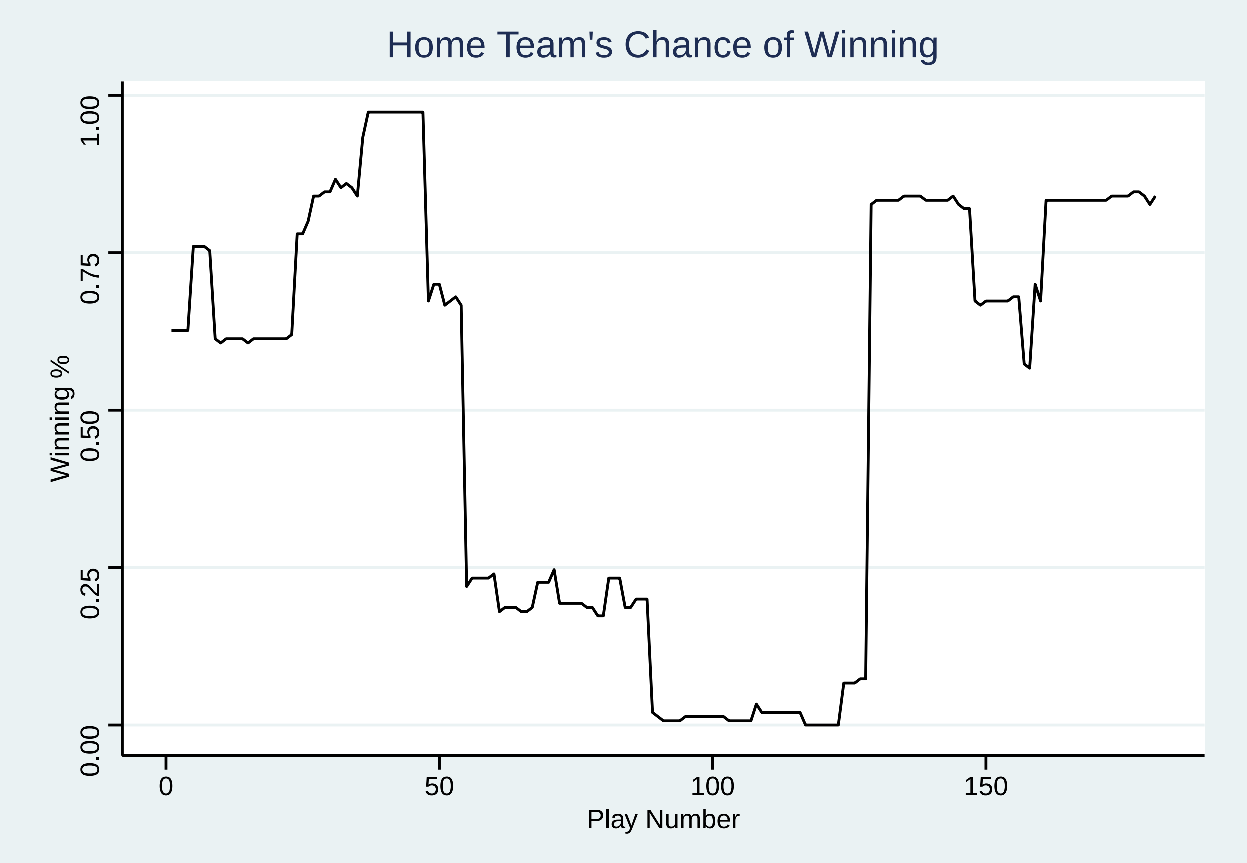
The next was Chicago at Detroit.

```
game_test <- NFL_2022_NFL_2023[NFL_2022_NFL_2023$home_team == "DET" & NFL_2022_NFL_2023$away_team == "CH

game_test_pred <- predict(no_team_model, newdata = game_test, type = "prob")

# Assuming game_pred_list is a list of predictions
game_pred_df <- data.frame(game_pred = unlist(game_test_pred))  # Convert list to data frame

# Plotting the predictions
ggplot(data = game_pred_df, aes(x = seq_along(game_pred.1), y = game_pred.1)) +
  geom_line() +
  labs(x = "Play Number", y = "Winning %", title = "Home Team's Chance of Winning") +
  theme_stata() +
  theme(panel.grid = element_blank())
```



And finally the Bills at Philly.

```
game_test <- NFL_2022_NFL_2023[NFL_2022_NFL_2023$home_team == "PHI" & NFL_2022_NFL_2023$away_team == "BU

game_test_pred <- predict(no_team_model, newdata = game_test, type = "prob")

# Assuming game_pred_list is a list of predictions
game_pred_df <- data.frame(game_pred = unlist(game_test_pred))  # Convert list to data frame

# Plotting the predictions
ggplot(data = game_pred_df, aes(x = seq_along(game_pred.1), y = game_pred.1)) +
  geom_line() +
  labs(x = "Play Number", y = "Winning %", title = "Home Team's Chance of Winning") +
  theme_stata() +
  theme(panel.grid = element_blank())
```



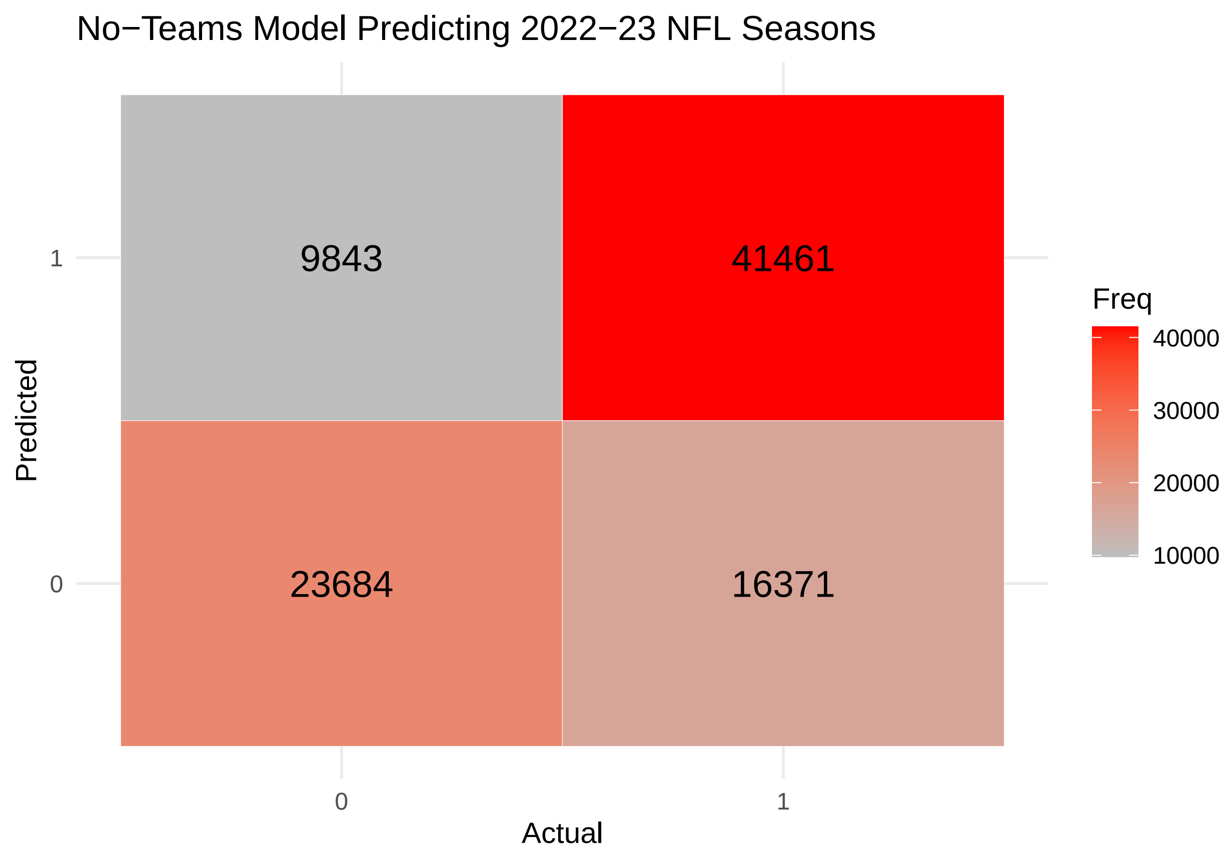
And then we tried to train a model based off of historical data and then apply it to more modern seasons.

```r
NFL2000_Tree <- randomForest(home_win ~ . -season -week
                              -home_team -away_team -goal_to_go
                              -pass_length,
                              data = NFL_2000_Test,
                              mtry = sqrt(ncol(NFL_2000_Test)),
                              ntree = 150,
                              nodesize = 1000,
                              maxnodes = 60)

pred_2000_teams <- predict(NFL2000_Tree, newdata = NFL_2022_NFL_2023, type = "prob")

#converting to wins if above 50%
pred_2000_fact <- as.factor(ifelse(pred_2000_teams[, 2] >= 0.5, 1, 0))

confusionMatrix(NFL_2022_NFL_2023$home_win, pred_2000_fact, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction     0     1
##          0 23684 16371
##          1  9843 41461
##
##                Accuracy : 0.7131
```

```
##                  95% CI : (0.7101, 0.716)
##     No Information Rate : 0.633
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4067
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.7169
##             Specificity : 0.7064
##          Pos Pred Value : 0.8081
##          Neg Pred Value : 0.5913
##              Prevalence : 0.6330
##          Detection Rate : 0.4538
##    Detection Prevalence : 0.5616
##       Balanced Accuracy : 0.7117
##
##        'Positive' Class : 1
##
```

```
pred_2000_noteam <- confusionMatrix(NFL_2022_NFL_2023$home_win, pred_2000_fact, positive = "1")

pred_2000_noteam_table <- as.data.frame(pred_2000_noteam$table)

ggplot(pred_2000_noteam_table, aes(x = Reference, y = Prediction, fill = Freq)) +
  #setting our tile coloring
  geom_tile(color = "white") +
  #setting the color and size of the text
  geom_text(aes(label = Freq), color = "black", size = 5) +
  #setting the color of our gradient
  scale_fill_gradient(low = "gray", high = "red") +
  #adding in labels
  labs(title = "No-Teams Model Predicting 2022-23 NFL Seasons", x = "Actual", y = "Predicted") +
  #plus our theme
  theme_minimal()
```



In general, the model performed a little more poorly than the previous versions, suggesting that the model needs more recent data to make predictions, but that it is still flexible enough to make solid predictions when it can only use old data.

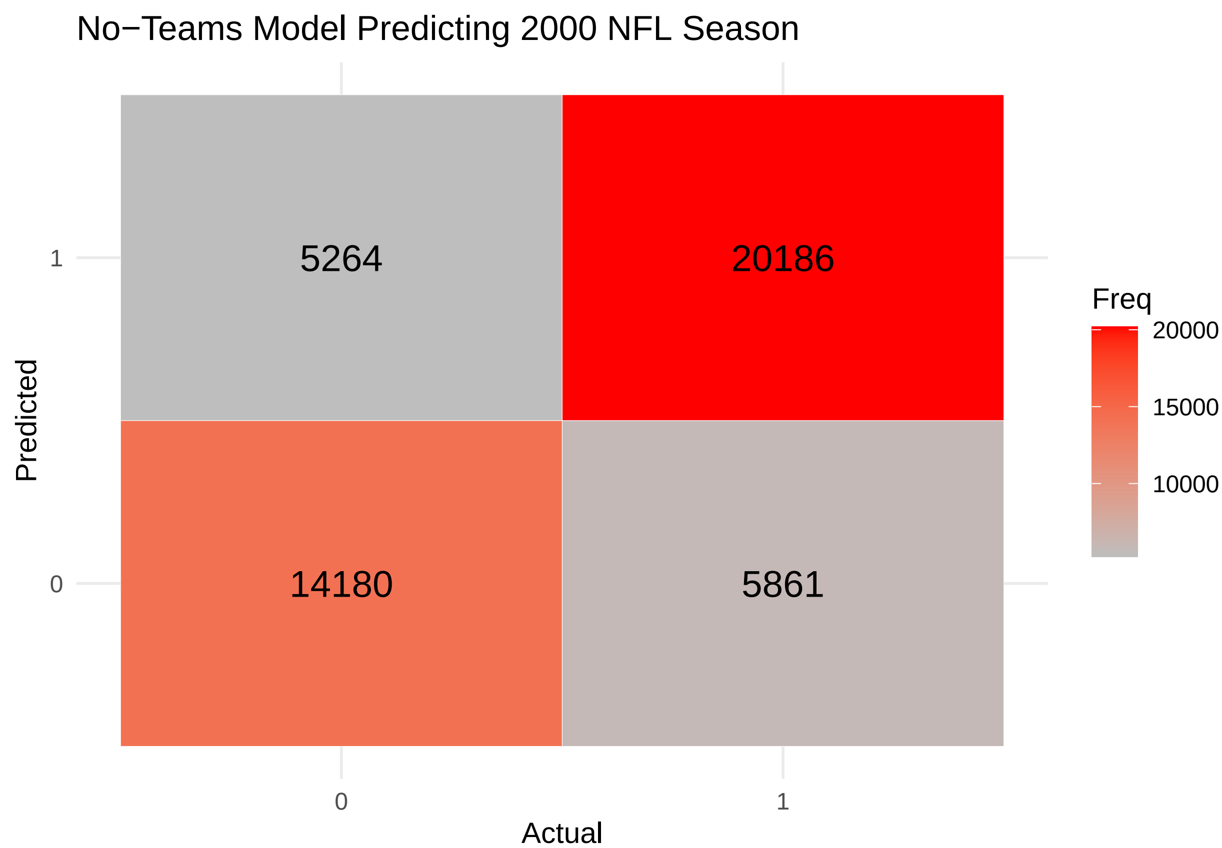
In addition, the model became much more consistent and accurate when it had access to three or more seasons, suggesting that more seasons would likely improve the model.

```
modern_stats <- cleaned_final[
  cleaned_final$season == 2020|
  cleaned_final$season == 2021|
  cleaned_final$season == 2022|
  cleaned_final$season == 2023 ,
]

no_team_modern <- randomForest(home_win ~ . -season -week
                              -home_team -away_team
                              -wind -goal_to_go -pass_length -pass_location -roof - surface - special_te
                              data = modern_stats,
                              mtry = sqrt(ncol(modern_stats)),
                              ntree = 150,
                              nodesize = 2000,
                              maxnodes = 90)

pred_mod <- predict(no_team_modern, newdata = NFL_2000_Test, type = "prob")

#converting to wins if above 50%
pred_mod_fact <- as.factor(ifelse(pred_mod[, 2] >= 0.5, 1, 0))
```

```r
confusionMatrix(NFL_2000_Test$home_win, pred_mod_fact, positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 14180  5861
##          1  5264 20186
##
##                Accuracy : 0.7554
##                  95% CI : (0.7515, 0.7594)
##     No Information Rate : 0.5726
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.5023
##
##  Mcnemar's Test P-Value : 1.599e-08
##
##             Sensitivity : 0.7750
##             Specificity : 0.7293
##          Pos Pred Value : 0.7932
##          Neg Pred Value : 0.7075
##              Prevalence : 0.5726
##          Detection Rate : 0.4437
##    Detection Prevalence : 0.5595
##       Balanced Accuracy : 0.7521
##
##        'Positive' Class : 1
##
```

```r
pred_mod_confusion <- confusionMatrix(NFL_2000_Test$home_win, pred_mod_fact, positive = "1")

pred_mod_table <- as.data.frame(pred_mod_confusion$table)

ggplot(pred_mod_table, aes(x = Reference, y = Prediction, fill = Freq)) +
  #setting our tile coloring
  geom_tile(color = "white") +
  #setting the color and size of the text
  geom_text(aes(label = Freq), color = "black", size = 5) +
  #setting the color of our gradient
  scale_fill_gradient(low = "gray", high = "red") +
  #adding in labels
  labs(title = "No-Teams Model Predicting 2000 NFL Season", x = "Actual", y = "Predicted") +
  #plus our theme
  theme_minimal()
```



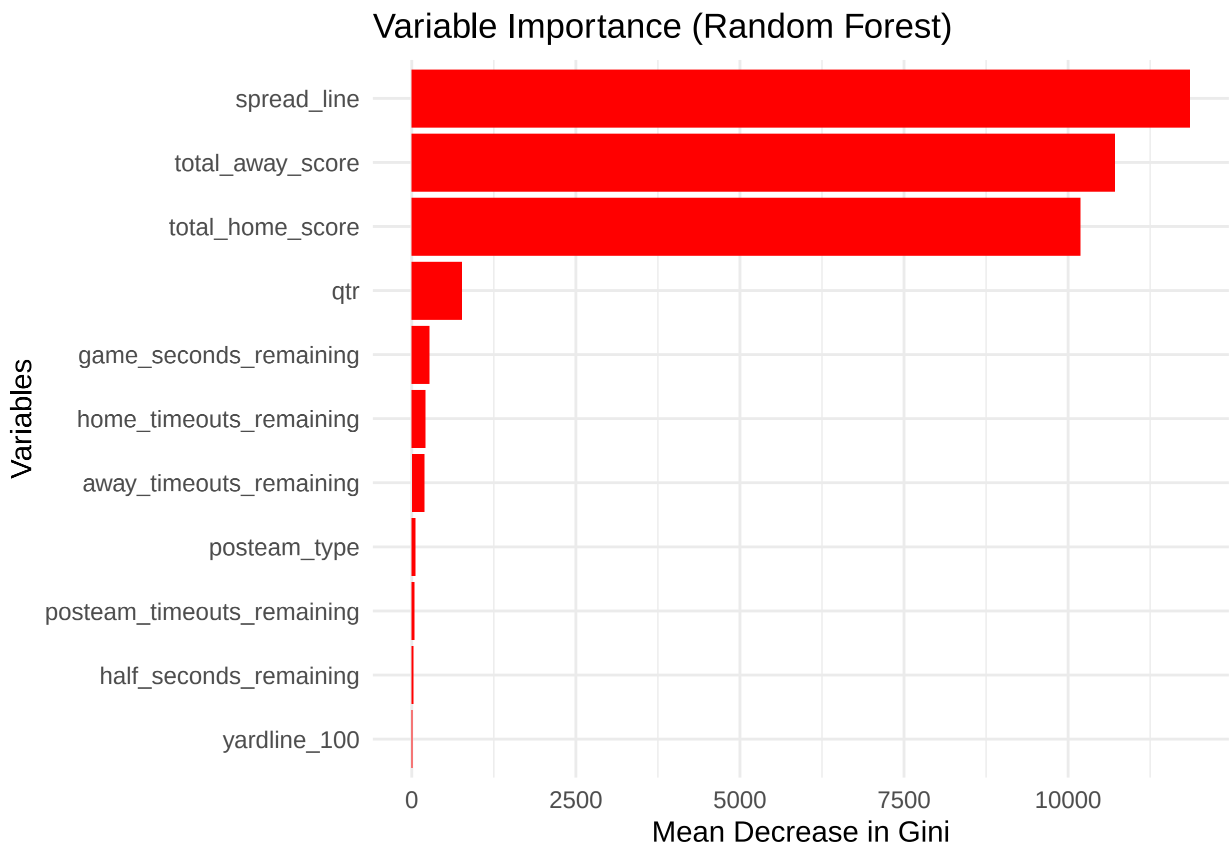
The no-team modern model was able to improve it's accuracy to over 75%, which shows that including more seasons is incredibly beneficial to the model.

```
no_team_modern$importance
```

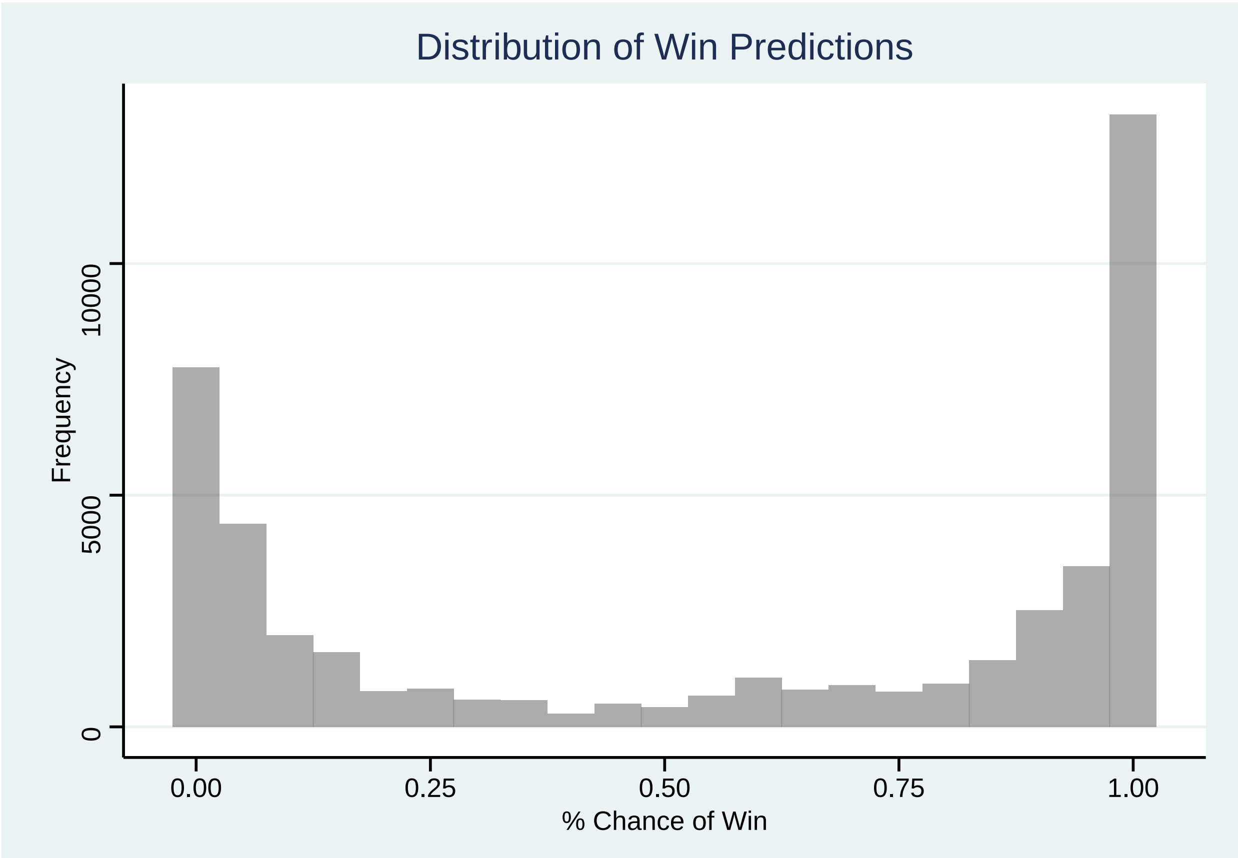
```
##                               MeanDecreaseGini
## posteam_type                         58.156853
## qtr                                 765.139126
## home_timeouts_remaining             209.353114
## away_timeouts_remaining             190.174834
## posteam_timeouts_remaining           43.467574
## game_seconds_remaining              269.226195
## spread_line                       11856.393029
## yardline_100                          9.119562
## total_home_score                  10187.555214
## total_away_score                  10714.470563
## half_seconds_remaining               23.633489
```

```
importance_modern <- as.data.frame(no_team_modern$importance)

importance_modern$Variable <- rownames(importance_modern)

ggplot(importance_modern, aes(x = reorder(Variable, MeanDecreaseGini), y = MeanDecreaseGini)) +
  geom_bar(stat = "identity", fill = "red") +
  coord_flip() +
```

```
  labs(title = "Variable Importance (Random Forest)",
       x = "Variables",
       y = "Mean Decrease in Gini") +
  theme_minimal()
```




```
#Converting to a data frame
pred_mod <- data.frame(class_0 = pred_mod[, 1], class_1 = pred_mod[, 2])

#And then plotting a historgram of the win predictions
ggplot(pred_mod, aes(x = class_1)) +
  geom_histogram(binwidth = 0.05, alpha = 0.5) +
  labs(title = "Distribution of Win Predictions", x = "% Chance of Win", y = "Frequency") +
  theme_stata()
```


Distribution of Win Predictions
Frequency
0
5000
10000
0.00
0.25
0.50
0.75
1.00
% Chance of Win