R Notebook

Alex Baker, James Kitch, Jackson Smith, Matthew Sheridan

Introduction

Introduction Sports games are designed around the idea of finding the team that plays the “best” over a given period of time. Professional baseball teams have nine innings, soccer teams ninety minutes, and American football, hockey, and basketball clubs have sixty minutes to score more points than their opponents. While officiating, weather, and “luck” inevitably play a role in deciding the outcome, the end result of a game should hopefully be evident by in-game statistics. Bill James’ arguably invented the field of sports analytics in 1977 with his first edition of Baseball Abstract, attempting to analyze past player performance and predict future team success based on quantitative measurements rather than qualitative scouting reports. This quantitative view of player and team statistics, now known as “sabermetrics”, exploded in baseball where it is very easy to generate a large number of individual statistics for every player.

The National Football League (NFL) was slower to adopt sabermetrics, but the rise of remote sensing software in recent years has made it much easier for players and individuals to acquire quantitative game statistics. Football presents an interesting area of analysis since it is naturally discretized into distinct plays which can be measured quantitatively. How many net yards were gained? Was it a run or a rush play? Was it a missed field goal attempt, and if so from how many yards? Motivated to explore these questions, in this project we sought to explore the relationship between NFL in-game statistics and the final score differential.

R is a natural environment in which to analyze this as it contains the nflfastR package, which contains accumulated play-by-play data from 1999 through 2021, with additional predictors beginning in 2006. We hope to create a parsimonious, streamlined model capable of predicting the score differential of an NFL game. While we will have some variables with relatively clear-cut relationships with score differential, our analysis will also focus on how more “strategic” variables relate to success. More turnovers in a game and more total yards should correlate strongly with score differential, but we are also interested in variables that have a more unclear relationship with the final outcome. Are missed extra point attempts indicative of the team as a whole having a bad day? What about third down conversion rate? And quarterback hits? Is it better if a quarterback is able to spread his passes to different receivers, or when one receiver dominates the game? These are the kinds of interactions and trends we hope to expose in our model. By using metrics that are not generally used to predict game outcomes, we hope to find insight into predicting wins that are not conventionally expected.

In our project, we hope to answer the following questions: How do in-game statistics relate to the final score differential of a game? Which predictors are most significantly associated with score differential? What about predictors that aren’t classically associated with offense? Do some predictors have a different impact as the game progresses? For instance, the first half might be more indicative of team strategy and the second half of the current score in the game.

Data Cleaning, EDA

The data originally obtained from nflfastR is very granular, with one row for every play in a season. We acquired the play-by-play data for 2017, 2018, 2019, and 2021, skipping 2020 because that was the “COVID season” where game dynamics may have been different without fans in the stadium. The play-by-play level was too granular for our purposes, as it would have led to a sparse design matrix due to the high variability of plays and the low frequency of certain game events such as quarterback hits, touchdowns, and turnovers. In order to explore the relationship between total in-game statistics and score differential, we aggregated the data by team and by game. For example, we would sum the passing yards over all of one team's plays in a game to get their total passing yards in the game. Aggregating the plays to the game-level allowed us to create a richer data matrix that would provide more interpretable, useful results. We chose a wide range of predictors from the nflfastR dataset, encompassing predictors which we thought would directly relate to score differential such as touchdowns and interceptions as well as variables whose relationship was less intuitively clear such as unique receivers, quarterback hits, and missed extra point attempts. After aggregation and preliminary variable selection our dataset looks as follows:

Insert dataset head

The included predictors are:

Game_id: ID to distinguish games.

Team_Name: What team are statistics being recorded for

Home: dummy variable, 1 if the given team was at home for the game and 0 if they were away.

TFL: the number of offensive plays from a team that are Tacked For a Loss (i.e. tackles behind the line of scrimmage) in a game.

QB_hit: the number of hits on a given team's quarterback in a game.

Pass_Yrd: Total game passing yards from a given team.

Rush_Yrd: Total game rushing yards from a given team.

FG_Attempt: Total field goal attempts in a game by a given team.

XP_missed: Total missed extra-point attempts in a game

FG_missed: Total missed field-goal attempts in a game

FUM: Total team fumbles.

INT: Total team interceptions.

Fourth_Dwn_Con: Total fourth-down conversions.

Third_Dwn_Con: Total third-down conversions

Pen: Total penalties committed by a given team

RedZone_Plays: Total number of plays run within the 20 yard line for a given team.

Unique_Receivers: Number of different receivers that caught a ball for a team in a game.

Shotgun_Plays: Number of plays run out of the shotgun formation for a team.

Half1_RunPass_Ratio: Runs/Passes in the first half for a team.

Third_and_Longs: Third downs converted from over 5 yards away for a team.

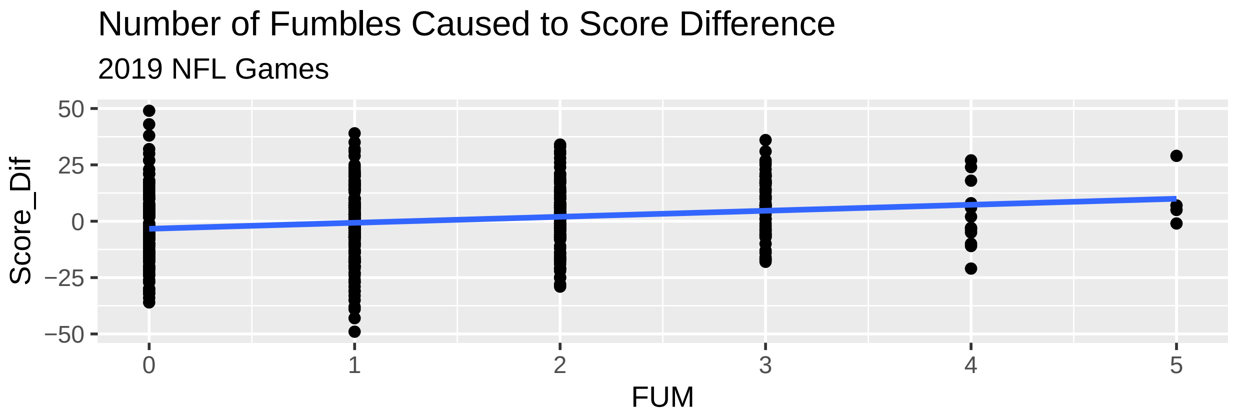
Half1_AirYrds: Total air yards (ball travels in air to meet receiver) in the first half.

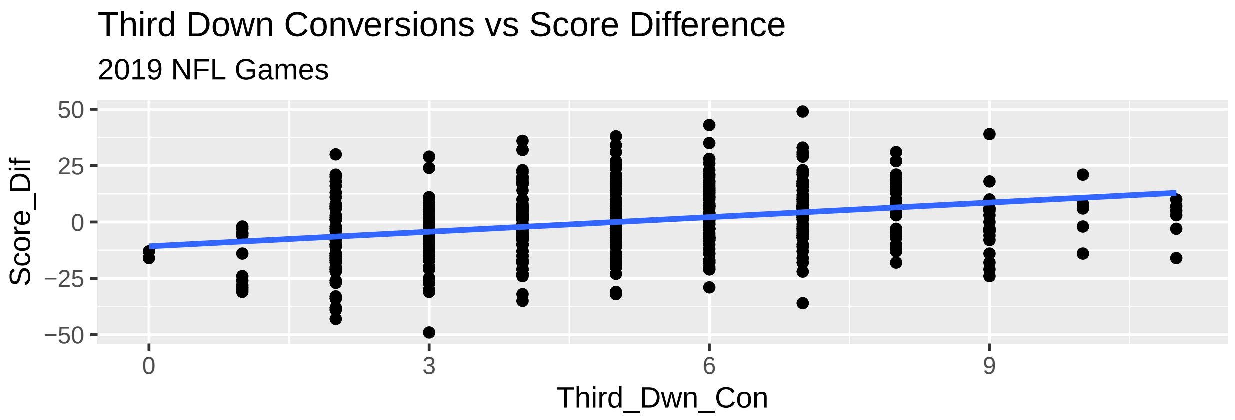
Score_Dif: month: 1 = January, 2 = February, etc.

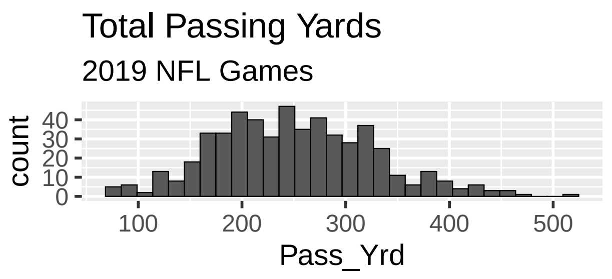
Below, we plotted the distributions for a number of different potential predictors, as well as their relationship with the response variable of interest. Most predictors are slightly right-skewed, as they have a positive support and have a slight "bell" shape for the most common values but are stretched out by the few exceptional games where a team throws for 500 yards or rushes for 250 yards. However, it is mild right-skewness as it is not to the extent that log-transforming them would make the distributions more symmetric. For example, if we log-transform `Pass_Yrd` we actually find that the resulting distribution is left-skewed, suggesting that the log transformation was actually too strong! While we found that a square root transformation would make the `Pass_Yrd` and `Rush_Yrd` distributions more symmetric, it would also make interpretations much less intuitive. Since the primary goal of this project is interpretation-focused and the relationship between offensive yards and score differential is still relatively linear (and the distributions of predictors still relatively normal despite very slight skew), we elected to not do any transformations here.

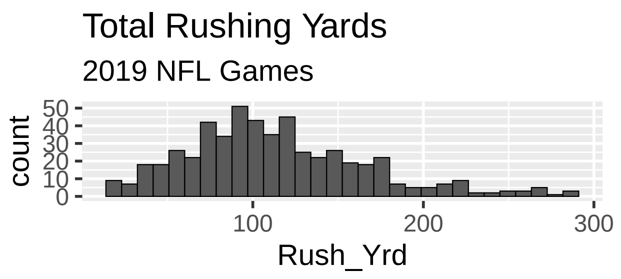
Furthermore, we can do a preliminary investigation of the relationships between some of the predictors and the outcome variable of score differential in 2019. Notably, we can see that the number of fumbles caused by a team is positively associated with score differential and that it is a relatively linear trend. The number of third down conversions is also positively associated with score differential, but this seems more complicated than a "simple" linear trend. Teams with a very large number (> 8) of third down conversions in a game appear to have a more negative score differential than teams with 5-7 third down conversions. This suggests that in future models investigating the effects of quadratic or higher order polynomial terms could be reasonable.

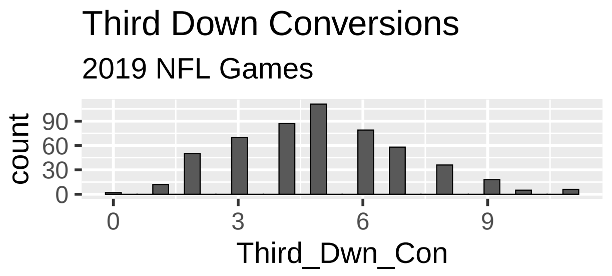
Below is a correlation table examining all of the predictors we have in our aggregated dataset. Notably, no predictors have a correlation close to 1 or -1 which would indicate collinearity and cause problems with our model. The largest correlations appear between `TD` and `Pass_Yrd`, `TD` and `Rush_Yrd`, and `FG` and `FG_missed`, with most other correlations $< |0.15|$.

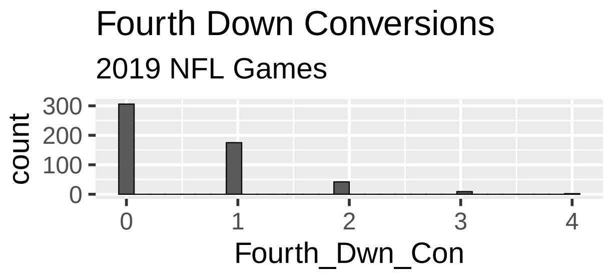


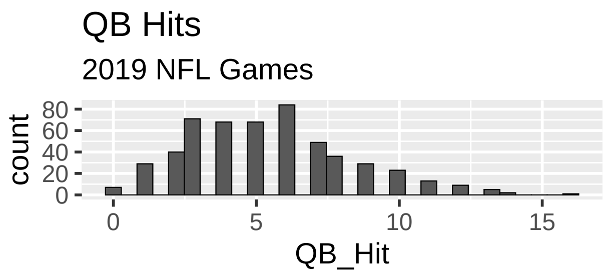












```
cor_matrix <- cor(data_2019[4:18])
cor_matrix[upper.tri(cor_matrix)] <- NA
cor_matrix <- cor_matrix %>% round(2)
cor_matrix
```

```
##                    TFL QB_Hit Pass_Yrd Rush_Yrd FG_attempt XP_missed FG_missed
## TFL               1.00     NA       NA       NA         NA        NA        NA
## QB_Hit            0.06   1.00       NA       NA         NA        NA        NA
## Pass_Yrd         -0.01   0.06     1.00       NA         NA        NA        NA
## Rush_Yrd         -0.13  -0.25    -0.21     1.00         NA        NA        NA
## FG_attempt        0.02  -0.03     0.08     0.15       1.00        NA        NA
## XP_missed        -0.06  -0.03     0.04     0.10      -0.04      1.00        NA
## FG_missed         0.05   0.05     0.10     0.04       0.41      0.02      1.00
## FUM               0.00  -0.01    -0.07    -0.01       0.18      0.09      0.06
## INT              -0.12  -0.12    -0.16     0.11       0.10      0.03      0.03
## Fourth_Dwn_Con   -0.05   0.11     0.08     0.01      -0.02      0.03      0.00
## Third_Dwn_Con    -0.10  -0.07     0.31     0.18       0.08      0.04     -0.02
## Pen               0.07   0.12     0.12    -0.02       0.00     -0.01      0.00
## RedZone_Plays    -0.20  -0.18     0.31     0.32       0.22      0.13     -0.03
## Unique_Receivers -0.03   0.02     0.15    -0.04       0.05      0.00     -0.02
## Shotgun_Plays     0.05   0.28     0.23    -0.08       0.05     -0.02     -0.02
##                    FUM   INT Fourth_Dwn_Con Third_Dwn_Con   Pen RedZone_Plays
## TFL                 NA    NA             NA            NA    NA            NA
## QB_Hit              NA    NA             NA            NA    NA            NA
## Pass_Yrd            NA    NA             NA            NA    NA            NA
## Rush_Yrd            NA    NA             NA            NA    NA            NA
## FG_attempt          NA    NA             NA            NA    NA            NA
## XP_missed           NA    NA             NA            NA    NA            NA
## FG_missed           NA    NA             NA            NA    NA            NA
## FUM               1.00    NA             NA            NA    NA            NA
## INT               0.09  1.00             NA            NA    NA            NA
## Fourth_Dwn_Con   -0.10 -0.02           1.00            NA    NA            NA
## Third_Dwn_Con     0.06  0.06          -0.04          1.00    NA             NA
## Pen               0.04 -0.02          -0.04         -0.02  1.00             NA
## RedZone_Plays     0.04  0.18           0.13          0.34 -0.07           1.00
## Unique_Receivers -0.01 -0.02           0.08          0.10  0.05           0.04
## Shotgun_Plays    -0.09 -0.15           0.27          0.13  0.13           0.04
##                  Unique_Receivers Shotgun_Plays
## TFL                            NA            NA
## QB_Hit                         NA            NA
## Pass_Yrd                       NA            NA
## Rush_Yrd                       NA            NA
## FG_attempt                     NA            NA
## XP_missed                      NA            NA
## FG_missed                      NA            NA
## FUM                            NA            NA
## INT                            NA            NA
## Fourth_Dwn_Con                 NA            NA
## Third_Dwn_Con                  NA            NA
## Pen                            NA            NA
## RedZone_Plays                  NA            NA
## Unique_Receivers             1.00            NA
## Shotgun_Plays                0.12             1
```

Initial Modeling:

Our baseline model includes all available predictors to get a sense for the relationships between variables and to final score difference. It is very unlikely to be our best-performing final model, but will be a good starting point for our analysis.

As discussed previously in our EDA, the assumptions of linear regression do generally hold for our data. The baseline model includes predicting score differential from the main effects of all available predictors. The distribution of each of the individual predictors is found to be relatively unskewed (without need for transformations). There also is a shown linear relationship between our predictor variables and score differential. There is no clear fanning (constant variance holds). Lastly, there is likely some dependence among the data (i.e., a team that has many quarterback hits is more likely to have more interceptions), but these should not greatly affect our models.

The code for initializing the models has been hidden for space constraints. The linear model was using all predictors, the stepwise model stepped from an intercept only model to all predictors, and the randomforest has not been tuned (YET) and uses all predictors, as well as the default parameters. We can see that the RMSE's are pretty close together for all the models, but the stepwise linear model seems to outperform all.

Analysis

Stepwise Models

We also ran stepwise variable selection backwards and in both directions, per the basis of our modeling project. We have seen firsthand how often forward selection leaves out significant predictors, and since one of our primary aims is to find less-well-known predictors, we believed the forward selection method to be too risky since it messes with the fundamental goal of our project. Forward selection also handicaps the AIC analysis, giving us further reason to avoid it for better inferential, predictive, and reasonable models. Our preliminary stepwise variable selection with simple predictors from a baseline linear model yielded a linear model with the following formula:

```r
#Stepwise Models
#Stepwise Backward
step.back = step(lm1, direction = "backward", k = 2, trace = F)
formula(step.back)
```

```
## Score_Dif ~ Home + QB_Hit + Pass_Yrd + Rush_Yrd + FG_attempt +
##     XP_missed + FG_missed + FUM + INT + Fourth_Dwn_Con + Third_Dwn_Con +
##     RedZone_Plays + Unique_Receivers + Shotgun_Plays
```

We can note that this immediately removes TLF and Pen; additionally, this model has a Multiple R-squared = 0.5439, an F-statistic of 135.2 and a p-value < 0.00000000000000022. In the coefficients table in the appendix, we can also see that XP_missed has a p-value > 0.05, indicating overfitting, which does not bode well for the more complex stepwise model.

With the both directional stepwise variable selection model starting from the baseline linear model all the way through the model with all interaction effects, we find the following formula:

```
#Stepwise Both
step.both = step(lm1, scope = c(lower = formula(Score_Dif ~ 1), upper = formula(Score_Dif ~ (. - Game_id - Team_Name)^2)), direc
```

```
## Warning in model.matrix.default(Terms, m, contrasts.arg = object$contrasts):
## variable 'Team_Name' is absent, its contrast will be ignored
```

```
formula(step.both)
```

```
## Score_Dif ~ Home + QB_Hit + Pass_Yrd + Rush_Yrd + FG_attempt +
##     XP_missed + FG_missed + FUM + INT + Fourth_Dwn_Con + Third_Dwn_Con +
##     RedZone_Plays + Unique_Receivers + Shotgun_Plays + FG_attempt:RedZone_Plays +
##     Pass_Yrd:RedZone_Plays + QB_Hit:RedZone_Plays + Fourth_Dwn_Con:Shotgun_Plays +
##     Pass_Yrd:Fourth_Dwn_Con + Fourth_Dwn_Con:RedZone_Plays +
##     Home:Fourth_Dwn_Con + Pass_Yrd:Unique_Receivers + QB_Hit:Pass_Yrd +
##     XP_missed:RedZone_Plays + QB_Hit:FG_attempt + QB_Hit:FUM +
##     INT:Shotgun_Plays + FG_attempt:Third_Dwn_Con
```

However, although the model appropriately disposes of countless insignificant predictors, it still includes several that are insignificant by their p-value, namely: Home, Pass_Yrd, Fourth_Dwn_Con, Pass_Yrd:Unique_Receivers, QB_Hit:Pass_Yrd, XP_missed:RedZone_Plays, QB_Hit:FG_attempt, QB_Hit:FUM, INT:Shotgun_Plays, and FG_attempt:Third_Dwn_Con, most of which are interaction terms that correspond to specific in-game situations or patterns. For example, while a home field advantage helps, the reasons for the advantage are often beyond numerical quantities such as being used to the weather or loud fans. Passing yards are obviously correlated with the number of unique receivers or QB hits since QB's cannot throw as much when hit more often/with less time in the pocket, and a smaller number of targetable receivers limits the number of potential yards since the team is more susceptible to defensive strategies. And of course the number of third down conversions inversely relates to the number of field goal attempts. That relates to simple football rules and strategies.

Random Forest

A random forest was fit using the combined game data from years 2017-2019. One advantage to using a random forest in this case is that it represents a good way to measure the predictive power of the predictors chosen, due to the non-parametric nature of the random forest and lack of assumptions needed to fit the model. After fitting a random forest, we can plot the Variable Importance of the predictors, to see how much predictive power is lost when the predictor is removed from the tree, and the greater the disparity the more important said predictor is. The one disadvantage to this method is that while we are able to tell that a certain predictor is very important, we are unable to interpret whether the relationship between the predictor and the response is positive or negative. For example, we see that QB_hits (the number of hits a team's quarterback takes) is considered important. We may assume, as logic follows, that minimizing the amount of times your quarterback is hit is most beneficial for a large score differential. However, we cannot determine that this is the case from only this plot.

A random forest was fit over a range of different hyper-parameters. Different values of mtry, ranging from 1 (essentially a random forest) to 12, reflect the number of variables randomly sampled at each split within the random forest, where higher values add more complexity. Secondly is maxnodes, which represents the maximum number of terminal nodes present in the resulting forest. The number of trees within our forest was fixed at 200 to maintain a reasonable run time. After cross-validating the RMSE between predicted and true values among training (2017-2019 data) and testing (2021 data) sets, the lowest test set RMSE was selected, and it was found that the optimal set of hyper parameters from our tests is mtry = 8 and maxnodes = 1000. Looking at the calculated differences in RMSEs on training and testing data, no models appear to be egregiously over-fit. Our selected hyper-parameters do not produce one of the largest magnitude RMSE differences, so it appears that our tuned model is acceptable.

Using the aforementioned hyper-parameters, the model was fit again and a variable importance plot was generated. This plot shows which variables have the most predictive power in the random forest. We can see that the highest predictors in terms of variable importance are Red Zone Plays and rushing yards, which appear to be in a tier of their own. This is followed by another tier that includes shotgun plays, QB hits, interceptions and passing yards. The rest of the predictors are of much lower importance based on this plot. This suggests that these mentioned predictors have the most important relationships with score differential for any given game.

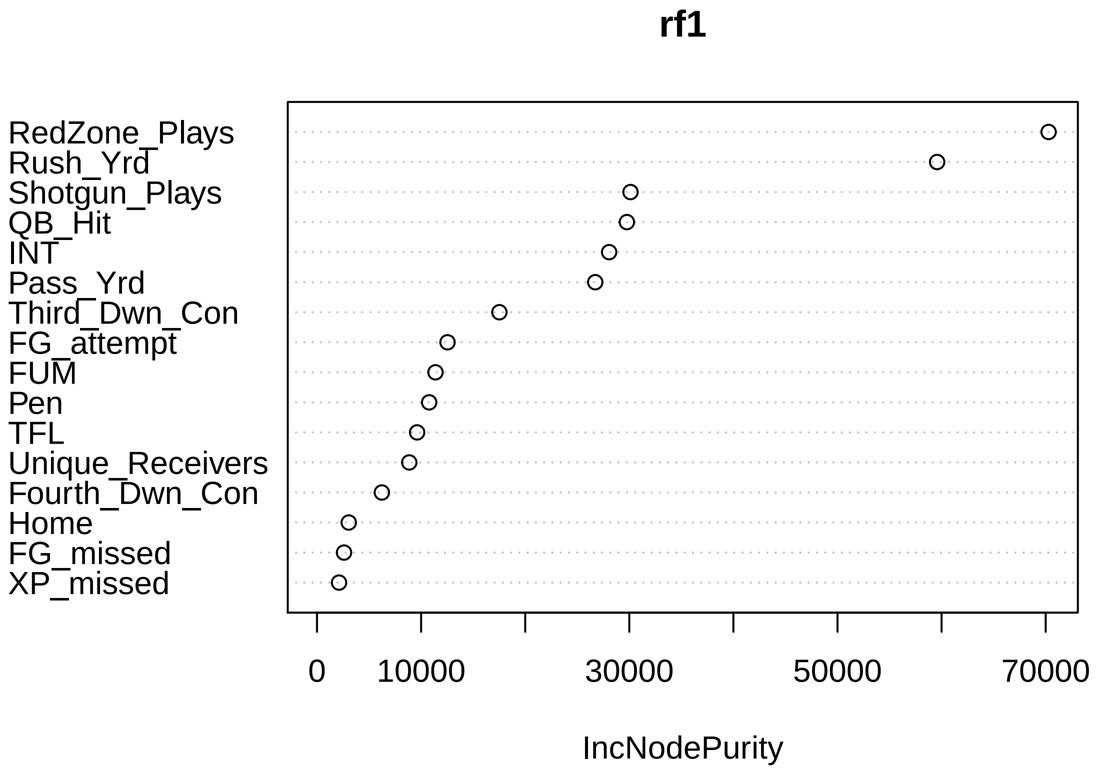
Var imp plot

It seems obvious that the number of touchdowns a team scores would have a large impact on their score differential. In the broader context of football, it makes sense that red zone plays would be considered highly important for predicting score differential because, considering that the red zone is the area within 20 yards of the opponent's end zone, this is where points are scored, and the more plays that are run in this area the more points should be scored. It also makes logical sense that rushing yards should be almost as important, both because teams that are leading by large amounts tend to run much more in order to possess the ball, and conversely teams that have great success running tend to have more control of the game and time of possession. In terms of the "second tier" predictors, QB_hits makes sense as a solid predictor. Presumably, the fewer hits your quarterback is taking, then the better chance you have of putting up more points than your opponent. The number of interceptions a team gets gives more opportunities to score while having to drive fewer yards down the field than if the ball was obtained otherwise (such as a punt, or a kickoff). The number of passing yards and shotgun plays are slightly less interpretable in terms of their importance. Passing yards could be a positive predictor because a team that amasses lots of passing yardage likely scores many points. However, a team that is behind by a large amount may gain many passing yards towards the end of the game in a meaningless effort to catch up to the winning team (this is referred to as "garbage time" statistics, when the outcome of the game is relatively determined, but play still occurs and players amass stats). Similarly, the number of stats taken out of the shotgun could be the mark of a team being successfully aggressive in their passing game, or that a team is struggling behind and trying to play catch-up. It would be best to investigate the nature of the relationship of these predictors with our chosen response in some sort of other linear model in order to examine the sign and size of the associated coefficient.

```r
maxnodes <- c(100, 500, 1000)
mtry <- c(1,5,8,10,12)

combs <- expand.grid(mtry, maxnodes)
colnames(combs) <- c("mtry", "maxnodes")
combs$RMSE_train <- rep(NA, nrow(combs))
combs$RMSE_test <- rep(NA, nrow(combs))

set.seed(139)
for(i in 1:nrow(combs)) {
  temp_forest <- randomForest(Score_Dif~.-Game_id-Team_Name-year,data=data_2017_19, mtry = combs$mtry[i], maxnodes = combs$maxno
  combs$RMSE_train[i] <- RMSE(data_2017_19$Score_Dif, temp_forest$predicted)
  combs$RMSE_test[i] <- RMSE(data_2021$Score_Dif, predict(temp_forest, new = data_2021))
}
combs$RMSE_difference <- combs$RMSE_test - combs$RMSE_train

rf1 <- randomForest(Score_Dif~.-Game_id-Team_Name-year,
                    data=data_2017_19,
                    mtry = combs[which.min(combs$RMSE_test),]$mtry,
                    maxnodes = combs[which.min(combs$RMSE_test),]$maxnodes,
                    ntree=200)

rf1$oob.times
```

```
##    [1] 70 76 76 68 68 74 79 75 63 67 80 90 80 73 70 79 64 76 80 77 86 73 76 82
##   [25] 73 75 81 73 78 80 79 86 64 79 81 70 74 79 73 79 73 69 79 72 64 83 73 69
##   [49] 64 74 73 62 85 67 77 77 72 73 71 67 57 71 65 59 71 87 78 54 73 67 71 82
##   [73] 72 67 77 67 75 65 73 70 66 75 71 68 80 77 65 81 72 86 83 76 77 64 71 59
##   [97] 75 63 69 76 70 84 77 74 58 72 61 68 77 69 73 59 73 76 75 64 60 76 77 92
##  [121] 73 72 74 74 70 82 81 74 61 68 72 82 86 64 71 78 79 79 78 69 83 64 85 62
##  [145] 81 74 62 94 80 69 84 71 70 81 79 79 75 80 79 80 75 66 74 74 72 69 86 73
##  [169] 82 78 78 76 64 74 72 60 68 68 81 81 81 82 82 65 54 72 77 85 74 70 79 75
##  [193] 78 62 83 84 65 73 81 82 77 74 81 78 72 67 77 67 64 65 66 79 84 70 70 78
##  [217] 69 81 70 86 61 67 77 83 70 76 80 59 76 78 62 76 84 68 62 81 77 88 74 74
##  [241] 69 71 74 77 72 86 77 73 77 70 72 67 73 80 65 75 75 80 74 80 70 70 71 84
##  [265] 72 85 66 74 75 76 71 64 78 84 77 73 65 84 76 85 83 82 80 77 65 73 72 70
##  [289] 73 73 76 70 74 83 75 75 83 69 73 86 85 82 83 72 74 88 67 89 68 76 76 69
##  [313] 68 76 89 70 69 73 68 67 72 69 80 77 73 85 68 78 71 78 68 70 69 69 71 60
##  [337] 77 71 69 70 77 70 73 69 81 77 66 78 84 79 69 73 76 88 69 72 69 74 73 77
##  [361] 77 74 79 62 76 75 83 64 74 72 77 64 84 79 63 69 65 73 78 83 81 84 74 62
##  [385] 81 79 63 66 68 86 70 66 72 71 59 64 74 77 74 69 80 66 69 82 61 77 76 78
##  [409] 71 74 75 85 90 71 90 86 71 73 68 82 72 83 84 79 72 66 67 63 68 77 81 70
##  [433] 81 87 71 68 72 78 66 75 77 70 68 75 70 67 73 80 64 77 70 75 60 57 69 64
##  [457] 69 71 73 80 79 73 69 64 74 70 63 72 74 73 75 86 71 70 73 74 59 73 63 67
##  [481] 79 73 84 60 73 72 62 64 73 76 55 72 63 72 67 73 68 79 75 64 80 76 75 65
##  [505] 76 73 66 74 83 75 73 78 64 68 76 73 79 75 75 74 61 70 76 74 71 68 69 87
##  [529] 73 67 70 82 75 78 67 85 63 75 78 71 67 62 67 69 73 72 73 75 66 68 76 77
##  [553] 67 74 73 84 85 70 67 70 78 76 84 67 78 72 79 71 76 73 85 74 84 82 71 67
##  [577] 78 83 75 78 79 65 65 79 72 68 67 75 66 90 73 81 75 67 73 78 62 71 81 66
##  [601] 72 73 79 76 77 83 66 69 72 74 78 79 84 85 71 86 81 82 77 82 75 69 71 85
##  [625] 63 75 57 76 69 69 77 64 68 74 55 72 75 82 77 78 77 69 75 68 69 77 75 91
##  [649] 76 66 87 73 70 71 74 87 75 71 66 65 72 65 67 76 75 72 73 72 79 63 66 72
##  [673] 65 68 68 79 70 70 74 67 72 64 77 68 79 67 68 63 72 75 71 72 67 73 62 66
##  [697] 66 68 81 77 78 71 75 76 92 70 70 82 81 72 84 66 75 62 63 74 77 76 76 80
##  [721] 92 74 77 81 75 74 66 87 82 64 79 73 72 84 72 84 74 62 73 77 63 79 76 69
##  [745] 59 71 69 67 72 64 83 72 71 76 78 80 72 80 72 81 76 73 75 75 78 77 79 76
##  [769] 62 82 64 67 86 70 74 68 67 80 80 63 81 71 77 77 72 62 82 75 62 83 86 86
##  [793] 76 77 78 69 88 74 75 77 71 67 64 76 84 75 73 55 69 74 78 79 63 77 81 75
##  [817] 84 70 69 66 74 63 61 67 63 82 69 80 75 75 66 64 67 71 75 77 85 65 67 67
##  [841] 91 78 66 60 78 76 77 79 69 65 67 77 74 76 69 66 70 84 79 80 70 72 68 84
##  [865] 74 67 77 69 74 71 83 71 82 76 71 67 66 81 66 71 81 85 74 89 72 66 79 89
##  [889] 64 70 84 76 52 71 82 66 79 74 67 72 64 78 80 79 75 68 82 71 91 75 75 70
##  [913] 73 75 60 69 60 72 67 82 65 73 73 64 68 67 70 79 86 71 72 71 87 75 64 65
##  [937] 69 68 83 71 68 88 75 51 85 69 76 64 69 85 83 75 74 85 62 71 73 79 76 72
##  [961] 62 67 79 68 76 70 74 74 64 71 76 79 83 75 64 68 72 61 76 77 59 76 76 69
##  [985] 81 77 73 83 71 71 70 79 78 67 69 65 64 80 73 58 82 78 82 67 71 75 79 70
## [1009] 78 77 57 88 88 80 76 73 72 80 74 67 83 81 67 73 84 75 88 77 73 78 75 77
## [1033] 70 73 67 70 74 75 76 76 66 71 74 64 73 73 75 57 83 70 68 64 78 70 71 74
## [1057] 88 82 95 76 80 70 72 77 58 67 73 75 68 77 65 71 74 72 61 70 68 63 65 70
## [1081] 66 60 74 76 76 81 58 79 66 64 80 74 81 69 78 79 67 69 83 72 70 65 64 78
## [1105] 72 75 77 76 70 73 84 78 69 83 70 67 73 74 71 78 72 76 68 86 62 77 64 69
## [1129] 76 67 84 73 68 71 73 68 73 71 71 64 78 71 81 64 80 78 67 78 68 72 59 73
## [1153] 72 69 79 72 83 68 71 89 84 65 81 79 71 80 75 67 79 72 76 74 76 75 64 76
## [1177] 76 71 68 77 73 70 65 67 73 77 65 75 78 68 76 62 87 74 65 76 77 73 69 70
## [1201] 75 70 73 65 68 78 79 66 66 60 64 65 57 68 69 73 64 75 72 81 75 70 71 92
## [1225] 68 60 73 78 70 74 60 79 72 74 70 69 77 72 77 76 77 89 78 75 57 73 73 86
## [1249] 77 75 80 72 71 66 76 71 80 66 63 76 69 64 63 71 68 83 68 85 67 71 79 91
## [1273] 69 72 80 76 81 79 79 74 66 59 73 77 64 78 66 79 73 76 57 74 82 69 73 62
## [1297] 68 78 77 63 84 72 71 68 69 80 77 72 88 63 70 72 78 72 59 75 74 72 82 71
## [1321] 77 72 78 67 81 78 86 70 80 78 76 79 75 86 72 72 63 65 68 58 77 73 78 81
## [1345] 73 67 84 65 70 72 87 76 67 79 70 68 73 87 81 72 66 74 61 69 67 67 66 65
## [1369] 54 65 66 85 72 73 66 76 81 68 65 67 76 63 67 67 68 72 67 72 74 73 71 68
## [1393] 68 79 86 90 59 75 72 55 71 81 76 79 71 76 56 68 62 79 77 68 73 76 64 70
## [1417] 71 66 67 72 86 74 74 66 89 75 75 83 83 69 71 61 76 65 84 67 67 81 90 78
## [1441] 68 75 74 78 71 72 72 69 65 73 67 91 72 87 75 70 88 68 66 81 72 70 81 72
## [1465] 74 76 69 71 54 69 71 79 69 81 79 75 79 82 76 71 69 77 74 70 72 76 72 68
## [1489] 72 81 73 82 82 77 71 71 71 77 86 65 74 77 73 86 69 79 58 71 62 81 65 77
## [1513] 68 84 77 80 79 83 68 63 82 73 74 85 68 79 65 68 78 67 81 76 76 74 79 66
## [1537] 68 70 75 88 85 83 70 80 73 59 76 60 72 76 72 77 70 71 69 69 66 71 81 67
## [1561] 69 84 74 82 70 80 82 67 71 72 72 64 67 80 76 70 71 67 66 79 80 74 72 69
## [1585] 66 82 79 72 83 83 84 83 71 68 86 78 83 73 69 79 77 68
```

```
varImpPlot(rf1)
```



A random forest was fit using the combined game data from years 2017-2019. One advantage to using a random forest in this case is that it represents a good way to measure the predictive power of the predictors chosen, due to the non-parametric nature of the random forest and lack of assumptions needed to fit the model. After fitting a random forest with desired predictors, we can plot the Variable Importance of the predictors, to see how much predictive power is lost when the predictor is removed from the tree, and the greater the disparity the more important said predictor is. The one disadvantage to this method is that while we are able to tell that a certain predictor is very important, we are unable to interpret whether the relationship between the predictor and the response is positive or negative. For example, we see that QB__hits (the number of hits a team's quarterback takes) is considered important. We may assume, as logic follows, that minimizing the amount of times your quarterback is hit is most beneficial for a large score differential. However, we cannot determine that this is the case from only this plot.

A random forest was fit over a range of different hyper-parameters. Different values of mtry, ranging from 1 (essentially a random forest) to 12, reflect the number of variables randomly sampled at each split within the random forest, where higher values add more complexity. Secondly is maxnodes, which represents the maximum number of terminal nodes present in the resulting forest. The number of trees within our forest was fixed at 200 to maintain a reasonable run time. After cross-validating the RMSE between predicted and true values among training (2017-2019 data) and testing (2021 data) sets, the lowest test set RMSE was selected, and it was found that the optimal set of hyper parameters from our tests is mtry = 8 and maxnodes = 1000. Looking at the calculated differences in RMSEs on training and testing data, no models appear to be egregiously over-fit. Our selected hyper-parameters do not produce one of the largest magnitude RMSE differences, so it appears that our tuned model is acceptable.

Using the aforementioned hyper-parameters, the model was fit again and a variable importance plot was generated. This plot shows which variables have the most predictive power in the random forest. We can see that the highest predictors in terms of variable importance are Red Zone Plays and rushing yards, which appear to be in a tier of their own. This is followed by another tier that includes shotgun plays, QB hits, interceptions and passing yards. The rest of the predictors are of much lower importance based off of this plot. This suggests that these mentioned predictors have the most important relationships with score differential for any given game.

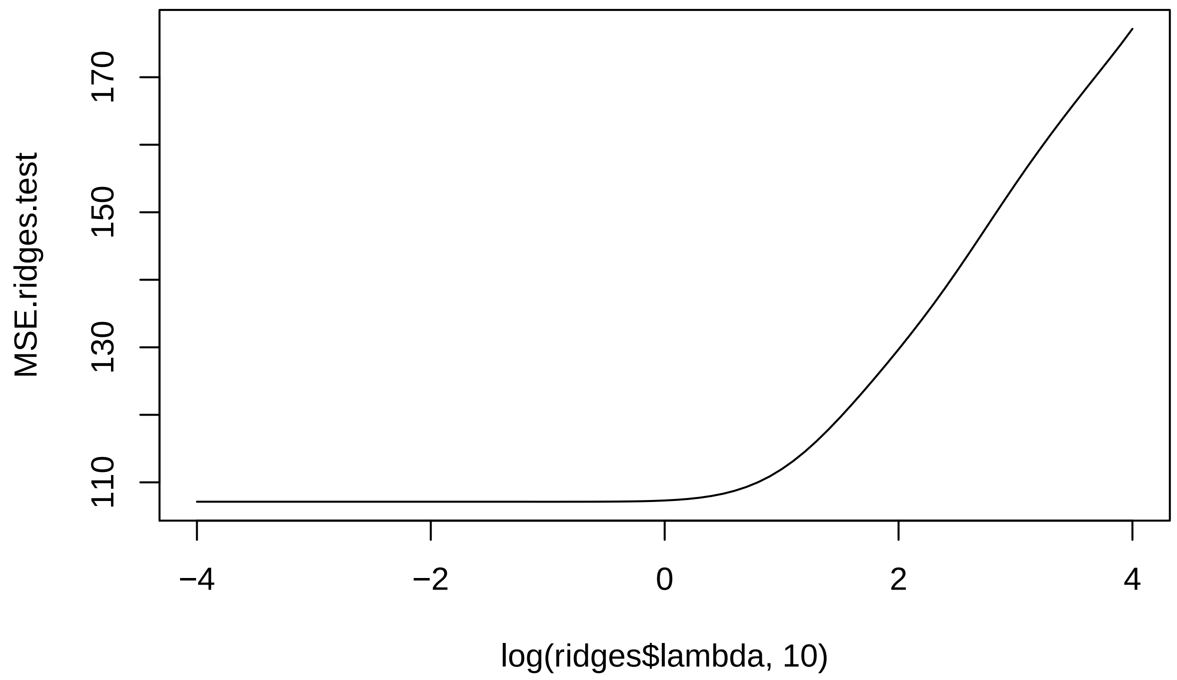
It seems obvious that the number of touchdowns a team scores would have large impact on their score differential. In the broader context of football, it makes sense that red zone plays would be considered highly important for predicting score differential because, considering that the red zone is the area within 20 yards of the opponent's end zone, this is where points are scored, and the more plays that are run in this area the more points should be scored. It also makes logical sense that rushing yards should be almost as important, both because teams that are leading by large amounts tend to run much more in order to posses the ball, and conversely teams that have great success running tend to have more control of the game and time of possession. In terms of the "second tier" predictors, QB__hits makes sense as a solid predictor. Presumably, the fewer hits your quarterback is taking, then the better chance you have of putting up more points than your opponent. The number of interceptions a team gets gives more opportunities to score while having to drive fewer yards down the field than if the ball was obtained otherwise (such as a punt, or a kickoff). The number of passing yards and shotgun plays are slightly less interpretable in terms of their importance. Passing yards could be a positive predictor because a team that amasses lots of passing yardage likely scores many points. However, a team that is behind by a large amount may gain many passing yards towards the end of the game in a meaningless effort to catch up to the winning team (this is referred to as "garbage time" statistics, when the outcome of the game is relatively determined, but play still occurs and players amass stats). Similarly, the number of stats taken out of the shotgun could be the mark of a team being successfully aggressive in their passing game, or that a team is struggling behind and trying to play catch-up. It would be best to investigate the nature of the relationship of these predictors with our chosen response in some sort of other linear model in order to examine the sign and size of the associated coefficient.

Ridge Regression and LASSO

```
library(glmnet)
```

```
## Loaded glmnet 4.1-2
```

```
library(Rcpp)
library(Matrix)
#Baseline Linear Models
X = model.matrix(lm1)[,-1] #drop the intercept

#Ridge Model
ridges = glmnet(X, data_2017_19$Score_Dif, alpha = 0, lambda = 10^seq(-4,4,0.1), standardize = F)
#coef(ridges)
#ridges$lambda

n.test = nrow(data_2021)
lm.test = lm(formula(lm1), data = data_2021)
Xtest = model.matrix(lm.test)[,-1] #drop the intercept again
yhats.ridges = predict(ridges, Xtest)
residuals.ridge = (data_2021$Score_Dif - yhats.ridges)^2
MSE.ridges.test = apply(residuals.ridge, 2, sum)/n.test

plot(MSE.ridges.test~log(ridges$lambda,10),type="l")
```



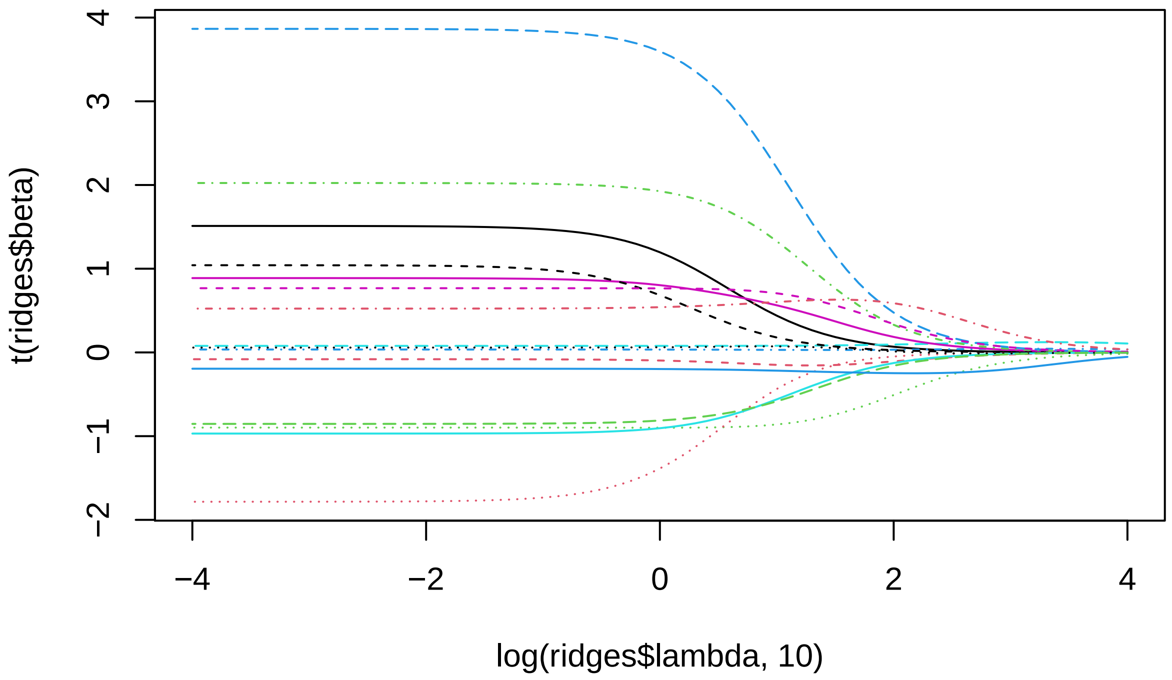
```
ridges$lambda[which.min(MSE.ridges.test)]
```

```
## [1] 0.1
```

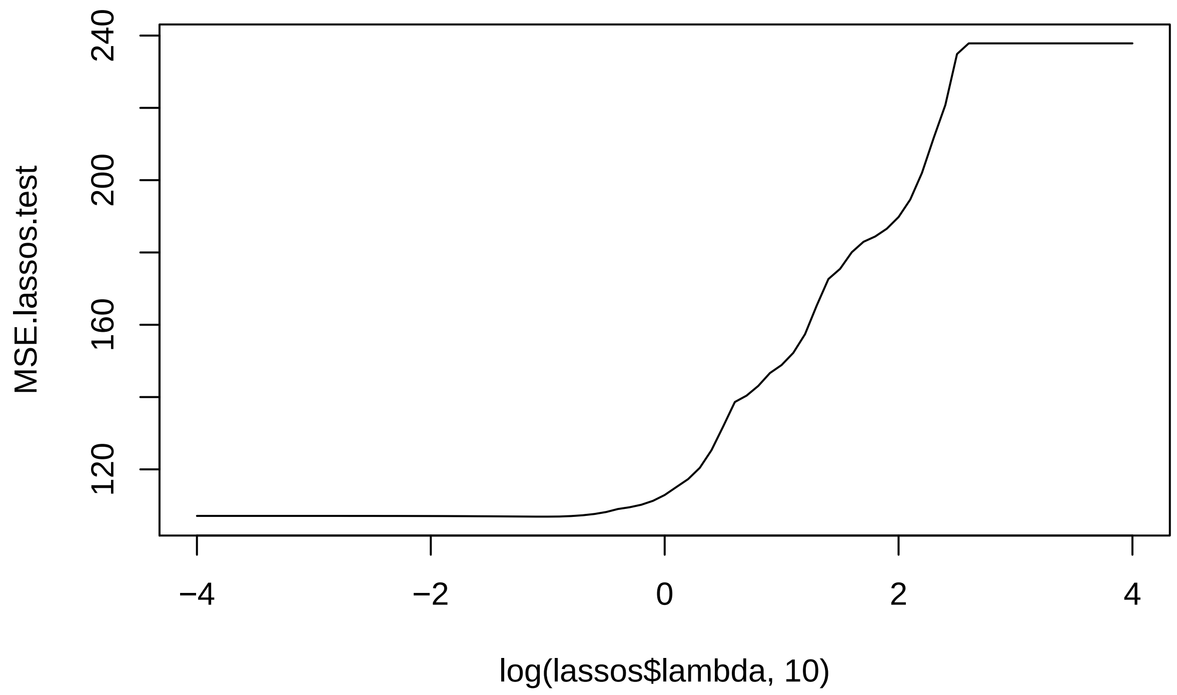
```
min(MSE.ridges.test)
```

```
## [1] 107.1261
```

```
matplot(log(ridges$lambda, 10), t(ridges$beta),type="l")
```



```
#Lasso Model
lassos = glmnet(X, data_2017_19$Score_Dif, alpha = 1, lambda = 10^seq(-4,4,0.1), standardize = F)
#coef(lassos)
#lassos$lambda

yhats.lassos = predict(lassos, Xtest)
residuals.lasso = (data_2021$Score_Dif - yhats.lassos)^2
MSE.lassos.test = apply(residuals.lasso, 2, sum)/n.test

plot(MSE.lassos.test~log(lassos$lambda,10),type="l")
```



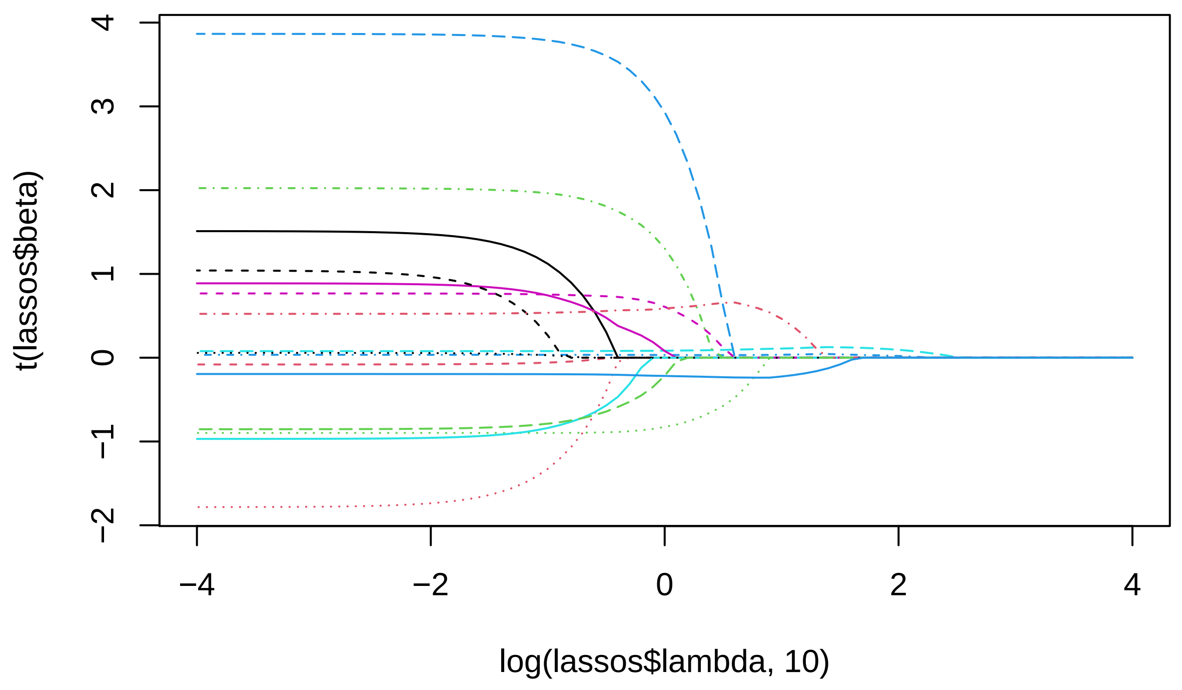
```
lassos$lambda[which.min(MSE.lassos.test)]
```

```
## [1] 0.1
```

```
min(MSE.lassos.test)
```

```
## [1] 106.9294
```

```
matplot(log(lassos$lambda, 10), t(lassos$beta),type="l")
```



```
#Stepwise Models
#Stepwise Backward
step.back = step(lm1, direction = "backward", k = 2, trace = F)
formula(step.back)
```

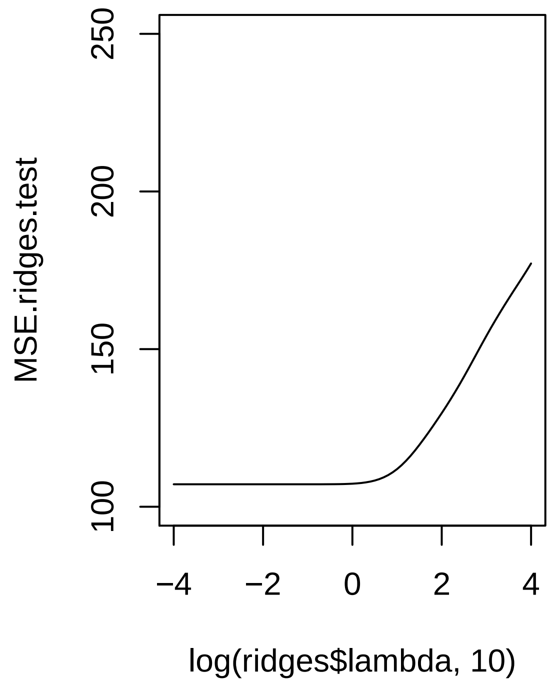
```
## Score_Dif ~ Home + QB_Hit + Pass_Yrd + Rush_Yrd + FG_attempt +
##     XP_missed + FG_missed + FUM + INT + Fourth_Dwn_Con + Third_Dwn_Con +
##     RedZone_Plays + Unique_Receivers + Shotgun_Plays
```

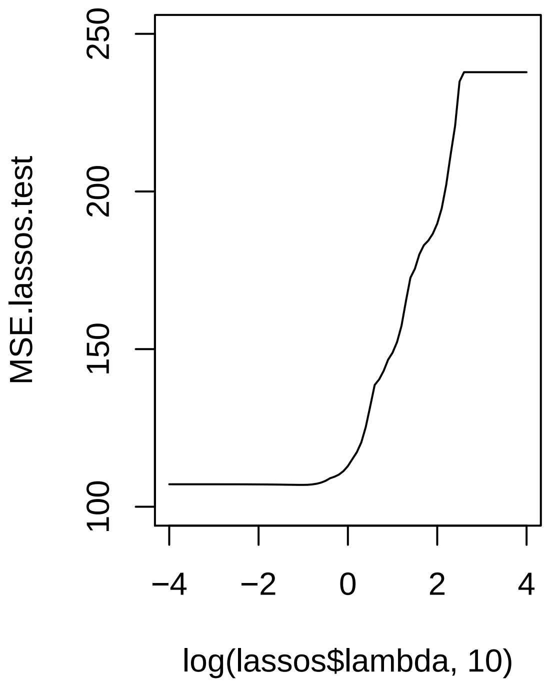
```
#Stepwise Both
step.both = step(step.back,
                 scope = c(lower = Score_Dif ~ . - Game_id - Team_Name -year, upper = formula(Score_Dif ~ (. - Game_id - Team_Na
                 direction = "both", trace = F)
step.both.int = step(step.back,
                 scope = c(lower = Score_Dif ~1,
                           upper = formula(Score_Dif ~ (. - Game_id - Team_Name -year)^2)),
                 direction = "both", trace = F)
```

The next modeling methods we chose to use were Ridge and Lasso regressions. Both of these models shrink our β_j coefficients towards 0 in an effort to expand on the ordinary least squares regression's unbiased estimates by reducing the Type I error rate through a penalizing λ. It is critical to note that the β_0 coefficient is unaffected by either penalized loss function, which means the intercept could theoretically remain consistent if the model weren't to change.

First, we fit a Ridge regression model, which uses a squared λ penalty factor, then the Lasso regression with its absolute value λ followed. The Lasso typically has an advantage over the Ridge in that it punishes the β_j coefficients more harshly, pushing the insignificant and unimportant predictors all the way to 0 if they are truly irrelevant.

```
par(mfrow=c(1,2))
plot(MSE.ridges.test~log(ridges$lambda,10),type="l", ylim = c(100, 250))
plot(MSE.lassos.test~log(lassos$lambda,10),type="l", ylim = c(100, 250))
```

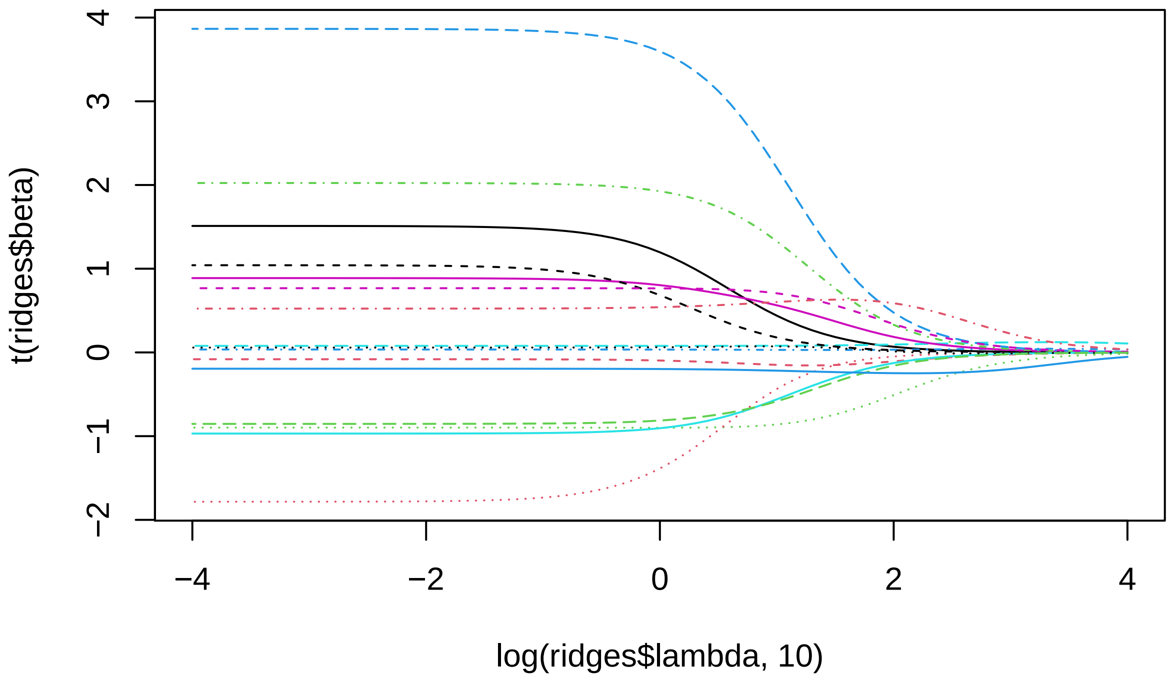




The MSE in the test plot demonstrates that this shrinkage, in both the Ridge and Lasso regressions, greatly improves the model estimates. Accordingly, the Ridge's best λ turns out to be $log_{10}(\lambda) \approx 0.5$ and $\lambda = 3.162278$, which matches the graph's minimum MSE well, from a visual inspection. The corresponding MSE for the best λ's Ridge is 109.7959. In a similar plot for the Lasso regression, the optimal λ came out as $log_{10}(\lambda) = -1$ so $\lambda = 0.1$. The Lasso also performed better than the Ridge regression, as expected, with an MSE of 107.4165, which is verifiably less than the Ridge's MSE ($107.4165 < 109.7959$). That means that the Lasso regression model is $\sim 2.2\%$ better than the Ridge regression model in terms of MSE, and we logically know that the $\lambda's$ cannot be compared due to the difference in calculation methods. This makes sense since our data does appear to have several unimportant predictors mixed in with a few significant ones, as is common in the real world beyond pure statistical problems. It is also crucial to note that the Lasso MSE tops out at 240 while the Ridge MSE does not appear to stop growing beyond our chosen λ bounds. For the Lasso, we have already found our optimal $\lambda = 0.01$, but the MSE boundary shows that MSE does not increase when $\lambda >\sim 2.3$.

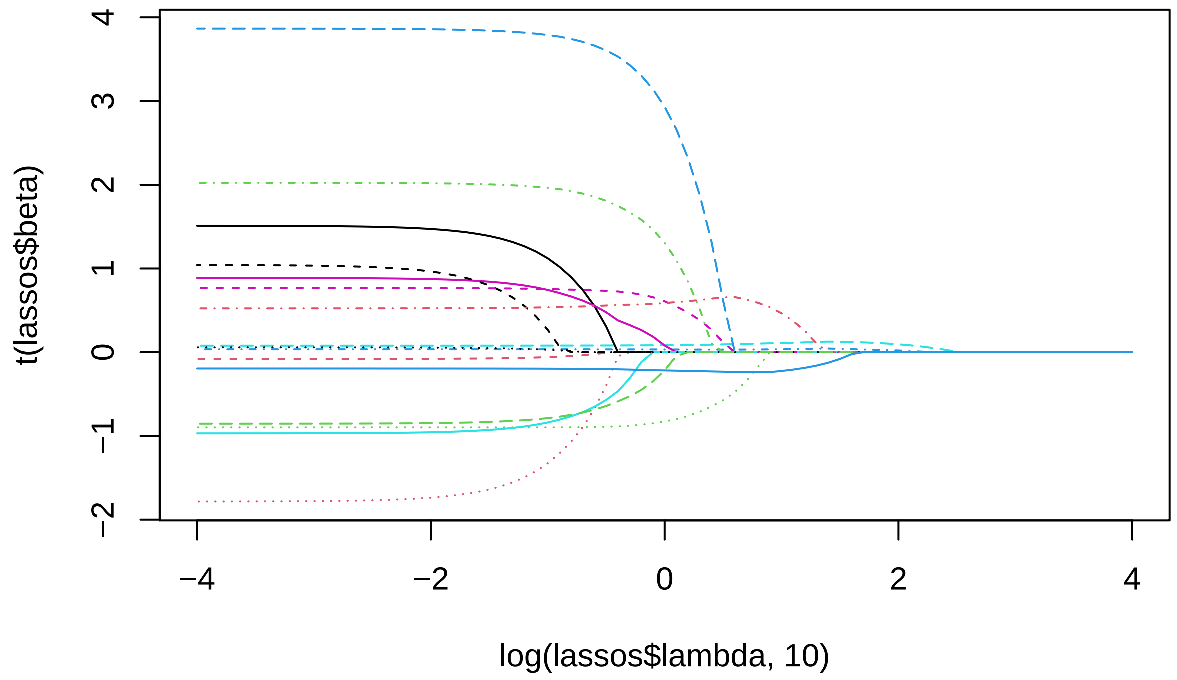
Looking at the Ridge model's β_j trajectory plots, we can see wide variability for many of the predictors, with some shrinking to 0 incredibly quickly while others more gradually or steeply further along the $log(\lambda's)$.

```r
matplot(log(ridges$lambda, 10), t(ridges$beta),type="l")
```



In the Lasso's version of the test β_j trajectory plots, the trajectories are much more enticing. all of the predictors are significant, until they aren't, dropping off sharply after specific lambda values, and almost all of them have shrunk to 0 by the time $\lambda = 0.1$ or shortly thereafter.

```
matplot(log(lassos$lambda, 10), t(lassos$beta),type="l")
```



We can be thankful for this result, albeit expected, since the Lasso regression's ability to shrink coefficients to 0 often makes it more conservative and more interpretable, leading to an easy and efficient modeling choice.

LMER Modeling

```
lmer_all <- lmer(Score_Dif ~ Home +TFL + QB_Hit + Pass_Yrd + Rush_Yrd + FG_attempt +
    XP_missed + FG_missed + FUM + INT + Fourth_Dwn_Con + Third_Dwn_Con + Pen +
    RedZone_Plays + Unique_Receivers + Shotgun_Plays + (1|Team_Name),data=data_2017_19)

lmer_step <- lmer(Score_Dif ~ Home + QB_Hit + Pass_Yrd + Rush_Yrd + FG_attempt +
    XP_missed + FG_missed + FUM + INT + Fourth_Dwn_Con + Third_Dwn_Con +
    RedZone_Plays + Unique_Receivers + Shotgun_Plays + (1|Team_Name),
    data=data_2017_19)

lmer_step_slope <- lmer(Score_Dif ~ Home + QB_Hit + Pass_Yrd + Rush_Yrd + FG_attempt +
    XP_missed + FG_missed + FUM + INT + Fourth_Dwn_Con + Third_Dwn_Con +
    RedZone_Plays + Unique_Receivers + Shotgun_Plays + (1+Rush_Yrd|Team_Name),
    data=data_2017_19)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 5.85676 (tol = 0.002, component 1)

## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, : Model is nearly unidentifiable: very large eig
##  - Rescale variables?
```

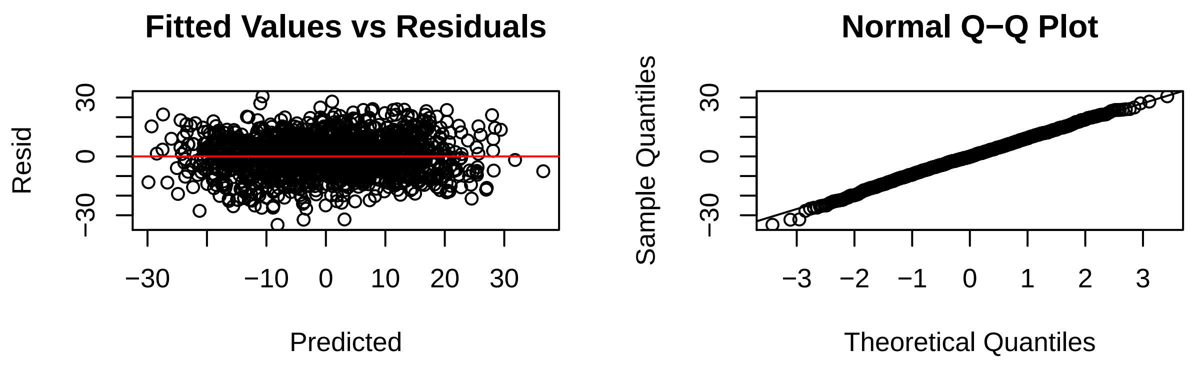
Next, we fit a Linear Mixed-Effects model with random intercepts by team to see if that had any impact on our predictions. Given that observations can be grouped by team and there are 32 teams, a mixed-effects model was a natural next step in our modeling approach. We fit two different types of mixed-effects models: one with all available predictors, and one with the predictors chosen by sequential variable selection. We found that the inclusion of a random intercept did not improve prediction substantially, with the random intercept term only accounting for ≈ 9% of the total variance in the model. Most coefficient estimates were very similar to the standard OLS run previously. Some of this may be due to the fact that there are a relatively large number of observations per team. Although there are 32 teams, each team has 48 - 60 "measurements" or games in the dataset and thus the benefit of linear mixed-effects modeling in shrinking outlier intercepts towards the mean is less pronounced.

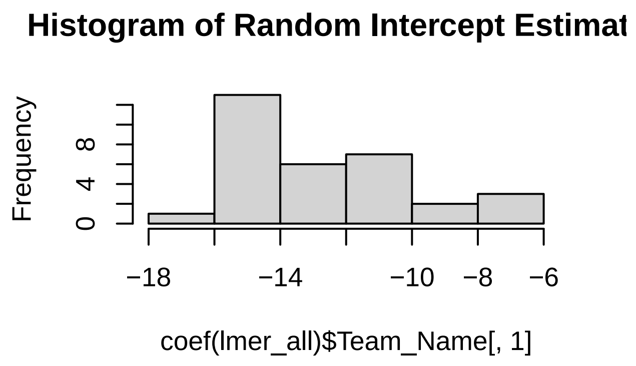
Nonetheless, a random intercept model with only the predictors returned by the stepwise variable selection process as outlined previously decreased the AIC and maintained the impact of the random intercept. Notably, the coefficients in the linear mixed-effects model did not change substantially from the ordinary least squares model with the same predictors. While most of the coefficients were slightly closer to zero in the linear mixed-effects model, the t statistics (i.e. level of significance of the coefficients) generally stayed the same as the addition of the random intercept helped decrease the variability in the model.

We also fit a linear mixed-effects model with a random intercept by team and random slope for the relationship of total rushing yards with final score differential. In theory, if some teams were primarily a "passing" team, total rushing yards might have less of an outcome on final score differential than if their primary game strategy relied on the run. However, we find that this random slope model actually worsens the model AIC suggesting that this is not a particularly important trend.

The fitted values vs residuals plot and Normal Q-Q plots are both reasonable for this random-intercept mixed-effects model, demonstrating that constant variance and normality of residuals are acceptable assumptions for this model. However, there is some concern with the distribution of fitted random intercept estimates–the histogram is neither normal-shaped or symmetric. This could potentially be fixed with **Blank but it is probably not an issue at the end of the day**.

```r
par(mfrow=c(2, 2))
plot(resid(lmer_all)~predict(lmer_all), main="Fitted Values vs Residuals", xlab="Predicted", ylab="Resid")
abline(h=0, col = "red")
#normality of residuals
qqnorm(resid(lmer_all))
qqline(resid(lmer_all))
#check normality of random effects - non-normal
hist(coef(lmer_all)$Team_Name[,1], main="Histogram of Random Intercept Estimates")
```





Ridge, Lasso, and Stepwise Regression

```r
list.models <- list(lmer_all, lmer_step, lmer_step_slope, lm1, step.both, step.both.int)
list.all.models <- list(lmer_all, lmer_step, lmer_step_slope, lm1, step.both, step.both.int)
model.results <- data.frame(unlist(lapply(list.models, AIC)),
                            unlist(lapply(list.models, BIC)),
                            unlist(lapply(list.models, predict, newdata=data_2017_19) %>% lapply(RMSE, data_2017_19$Score_Dif)),
                            unlist(lapply(list.models, predict, newdata=data_2021) %>% lapply(RMSE, data_2021$Score_Dif))) %>% d

names(model.results) <- c("AIC", "BIC", "RMSE.Train", "RMSE.Test")
rownames(model.results) <- c("lmer", "lmer_stepwise", "lmer_stepwise_rslope", "lm.all", "lm.step", "lm.step.interaction")
model.rmse <- rbind(c(RMSE(predict(lassos, X), data_2017_19$Score_Dif),
                      RMSE(predict(lassos, Xtest), data_2021$Score_Dif)),
                    c(RMSE(predict(ridges, X), data_2017_19$Score_Dif),
                      RMSE(predict(ridges, Xtest), data_2021$Score_Dif)),
                    c(RMSE(rf1$predicted, data_2017_19$Score_Dif),
                      RMSE(predict(rf1, data_2021), data_2021$Score_Dif))) %>% data.frame()
names(model.rmse) <- c("RMSE.Train", "RMSE.Test")

rmse.models <- rbind(model.results[,c(3, 4)], model.rmse)
rownames(rmse.models) <- c("lmer", "lmer_stepwise", "lmer_stepwise_rslope", "lm.all", "lm.step", "lm.step.interaction", "Ridge",
knitr::kable(rmse.models)
```

	RMSE.Train	RMSE.Test
lmer	9.360256	10.60430
lmer_stepwise	9.371261	10.57038
lmer_stepwise_rslope	9.321312	10.58357
lm.all	9.851668	10.35031
lm.step	9.853232	10.35228
lm.step.interaction	9.563922	10.37966
Ridge	11.595004	12.23013
Lasso	10.441002	10.99376
Random Forest	10.492822	10.50129

```r
t(model.results)
```

```
##                    lmer lmer_stepwise lmer_stepwise_rslope       lm.all
## AIC        11875.698003  11869.880422         11871.617887 11911.880212
## BIC        11977.899158  11961.323561         11973.819041 12008.702358
## RMSE.Train     9.360256      9.371261             9.321312     9.851668
## RMSE.Test     10.604300     10.570382            10.583571    10.350315
##                 lm.step lm.step.interaction
## AIC        11908.388740        11840.904367
## BIC        11994.452870        12002.274611
## RMSE.Train     9.853232            9.563922
## RMSE.Test     10.352279           10.379658
```

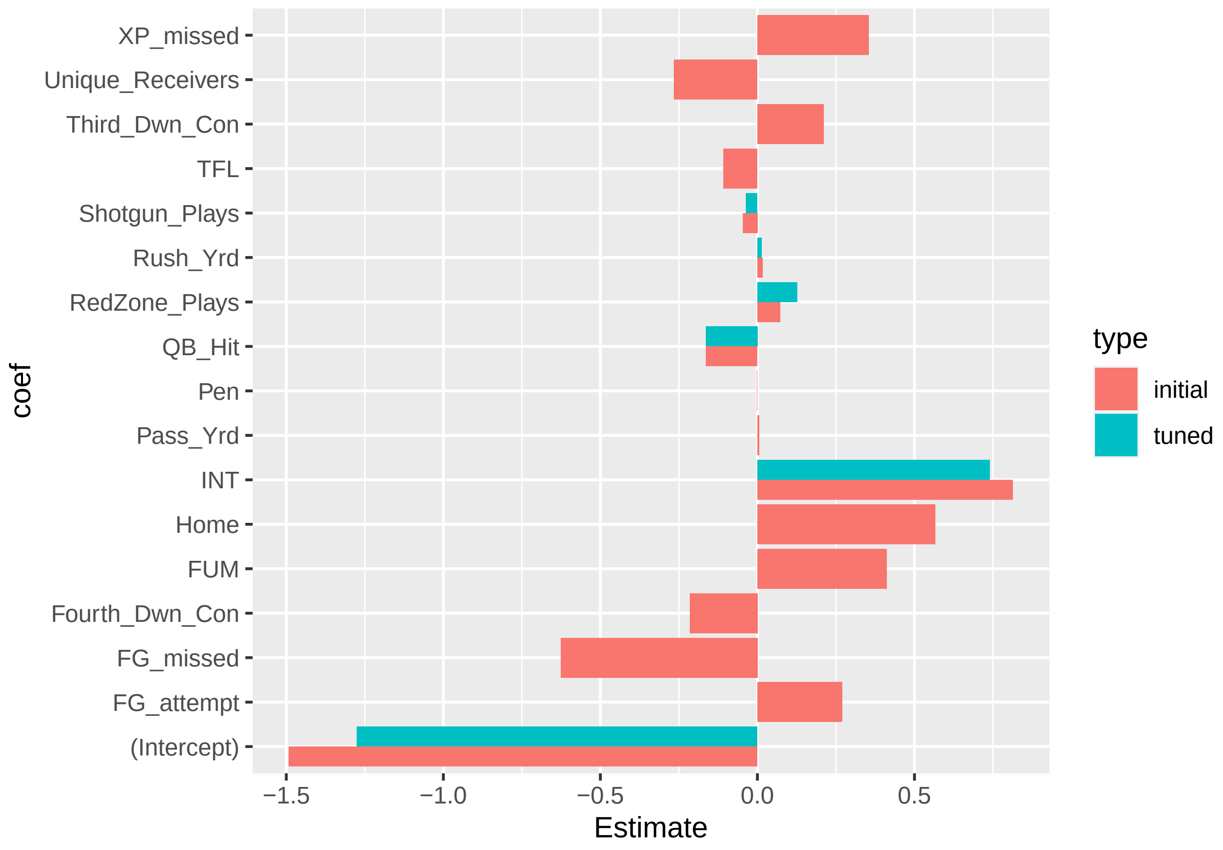
```r
knitr::kable(model.results[,1:2])
```

	AIC	BIC
lmer	11875.70	11977.90
lmer_stepwise	11869.88	11961.32
lmer_stepwise_rslope	11871.62	11973.82
lm.all	11911.88	12008.70
lm.step	11908.39	11994.45
lm.step.interaction	11840.90	12002.27

Logistic applications:

Although our main focus was predicting score differential, we thought that predicting the probability of a win would be an interesting extension to our project, and would highlight some of the oversights of our models as well as some interesting scenarios that just can't be handled meaningfully (looking at you, Bill Belichick). In this section, we fit a glm model with the parameter of being in the binomial family for the response, which in this case is wins. Our assumptions are minimal, as seen in the lectures, and all we need is independent observations which we generally have. We recorded each teams' wins for each game, to be used as a response. Then, we created two glm's, one with the top 5 predictors from the random forest variable importance (Rush Yards, RedZone Plays, Qb Hits, Shotgun plays, and interceptions) as well as a model with all predictors except for the game id, year, team, and score differential. Using these models, we could create models showing what things contributed to the probability of a win. Doing this, we could compare the two models' coefficients and see what changed from model to model:

```r
data_2017_19_test = data.frame(data_2017_19)

data_2017_19_test$wins = ifelse(data_2017_19$Score_Dif > 0, 1, 0)

win.log = glm(wins~.-Score_Dif-year, data=data_2017_19_test[,seq(3, ncol(data_2017_19_test))], family="binomial")

best.log = glm(wins~RedZone_Plays + Rush_Yrd+Shotgun_Plays +QB_Hit+INT, data = data_2017_19_test, family="binomial")

init_coef_log = data.frame(summary(win.log)$coefficients)
best_coef_log = data.frame(summary(best.log)$coefficients)

init_coef_log$coef = row.names(init_coef_log)
best_coef_log$coef = row.names(best_coef_log)
init_coef_log$type = "initial"
best_coef_log$type = "tuned"

ggplot(rbind(init_coef_log, best_coef_log), aes(fill=type, y=Estimate, x=coef)) +
    geom_bar(position="dodge", stat="identity") + coord_flip()
```



Our plot shows most notably that the intercept lost magnitude when predictors were removed, and as a result changed the magnitude of the kept predictors. Most importantly of which was rushing yards, whose coefficient decreased a lot from the initial model (which may not look like a ton on this plot). The decrease from the initial model to the tuned model was 0.016 to 0.013, and is significant because rushing yards are usually on the scale of about 120 to 180 per team per game.

Interestingly, both models had a worse train accuracy than test accuracy, which was in the years 2017-2019, as seen in the table below. Showing that our model wasn't too overfit and did have some predictive power.

```
new_2021 = data.frame(data_2021)
new_2021$year = 2021

new_2021$wins = ifelse(data_2021$Score_Dif > 0, 1, 0)

predicted_wins_initial = 1*(predict(win.log, type = "response")>0.5)
predicted_wins_best = 1*(predict(best.log, type = "response")>0.5)

Train_Acc_Init = sum(predicted_wins_initial == data_2017_19_test$wins)/nrow(data_2017_19_test)
Train_Acc_Best = sum(predicted_wins_best == data_2017_19_test$wins)/nrow(data_2017_19_test)

pred_test_init = 1*(predict(win.log, newdata=new_2021, type = "response")>0.5)
pred_test_best = 1*(predict(best.log, newdata=new_2021, type="response")>0.5)

Test_Acc_Init = sum(pred_test_init == new_2021$wins)/nrow(new_2021)
Test_Acc_Best = sum(pred_test_best == new_2021$wins)/nrow(new_2021)

data.frame(Train_Acc_Init, Train_Acc_Best, Test_Acc_Init, Test_Acc_Best)
```

```
##   Train_Acc_Init Train_Acc_Best Test_Acc_Init Test_Acc_Best
## 1      0.7940075      0.7509363      0.806701     0.7860825
```

Using these models, we were able to explore some interesting extensions, such as the Patriots week 4 bout versus the Buccaneers. In this game, the Patriots exceeded expectations and held the defending super bowl champions to 6 points in the first half, and 13 in the second, good for a win in many weeks. Importantly, they played them close all game and the Bucs won only on a last second field goal. What is strange, however, is that our model showed the Patriots as having a 3.78% probability of a win in the tuned model, and a 1.21% chance in the initial model. The Bucs were given a 70.2% and 86.5% probability in the tuned and untuned model, respectively. So, why did our model completely miss on a game that, honestly, should have been won by the Patriots? Taking a closer look, we can see that the Pats had -1 rush yards (no, not an error). Our model heavily favors rush yards and gives passing yards almost no weights, especially in the tuned model where passing yards meant nothing. The Patriots often perform unusual game plans, such as only throwing 3 times in a 14-10 win versus the Buffalo Bills on December 6, 2021. These game plans clearly throw off our model, showing that it really isn't robust to model a win probability logistically off of in game data (for the Patriots, at least). There's something about the "eye test" (a theoretical, informal "test" that involves years of watching and analyzing games, making snap judgements about outcomes and decisions while watching the game). Someone watching the week 4 Patriots-Bucs game surely wouldn't be giving the Patriots a ~4% chance of winning after seeing them absolutely dominate in the first half, yet our model says that not having any rush yards is highly correlated with losing the game.

Importantly, the model did actually get that prediction correct, but odd scenarios like this can mess with the model's accuracy (off the top of my head, the falcon's game versus the saints where they had 34 rushing yards and won is an example of a false prediction). Overall, rating one metric too highly can mess with the model, just as in any model, so we must be careful about determining direct causation.

In the long run, we'd be able to claim that rushing for more yards likely means you are winning and don't have to pass, which points to there being a win, but a team could just rush for every play and rack up the yards and still lose, which is why we cannot say that rushing yards increase your chances of winning, but rather we associate higher amounts of rushing yards with a higher chance of winning.

```
ne.game.init = predict(win.log, newdata = new_2021[new_2021$Game_id=="2021_04_TB_NE" & new_2021$Home==1,], type="response"); ne.
```

```
##        123
## 0.01212202
```

```
tb.game.init = predict(win.log, newdata = new_2021[new_2021$Game_id=="2021_04_TB_NE" & new_2021$Home==0,], type="response"); tb.
```

```
##       124
## 0.8652279
```

```
ne.game.best = predict(best.log, newdata = new_2021[new_2021$Game_id=="2021_04_TB_NE" & new_2021$Home==1,], type="response"); ne
```

```
##       123
## 0.0378281
```

```
tb.game.best = predict(best.log, newdata = new_2021[new_2021$Game_id=="2021_04_TB_NE" & new_2021$Home==0,], type="response"); tb
```

```
##       124
## 0.7022904
```

```
data.frame(NE_Chance_Tuned = ne.game.best, NE_Chance_Untuned = ne.game.init, TB_Chance_Tuned = tb.game.best, TB_Chance_Untuned =
```

```
##  NE_Chance_Tuned NE_Chance_Untuned TB_Chance_Tuned TB_Chance_Untuned
##        0.0378281        0.01212202       0.7022904         0.8652279
```

Extension: 1st Half Predictors

```
Many strategies during NFL games are determined by the context of the game as it unfolds. For instance, a team that is trailing
```

These predictors are incorporated into previously generated models to weigh their comparative importance with older predictors, while removing predictors similar to our first half ones (rushing yards, passing yards, QB hits).

A random forest was fitted using these new predictors. It was run before log-transforming the Run/Pass ratio data, as there are no underlying distributional assumptions. The variable importance plot produced with this new data in the presence of old predictors shares the same general structure as before. The new 1st half predictors can be seen grouped together in what appears to be their own "tier" of importance, below the same top 6 important predictors as before. This may be a result of multicollinearity between the newly introduced predictors. Half1__QB__Hits appears to be a relatively unimportant predictor.

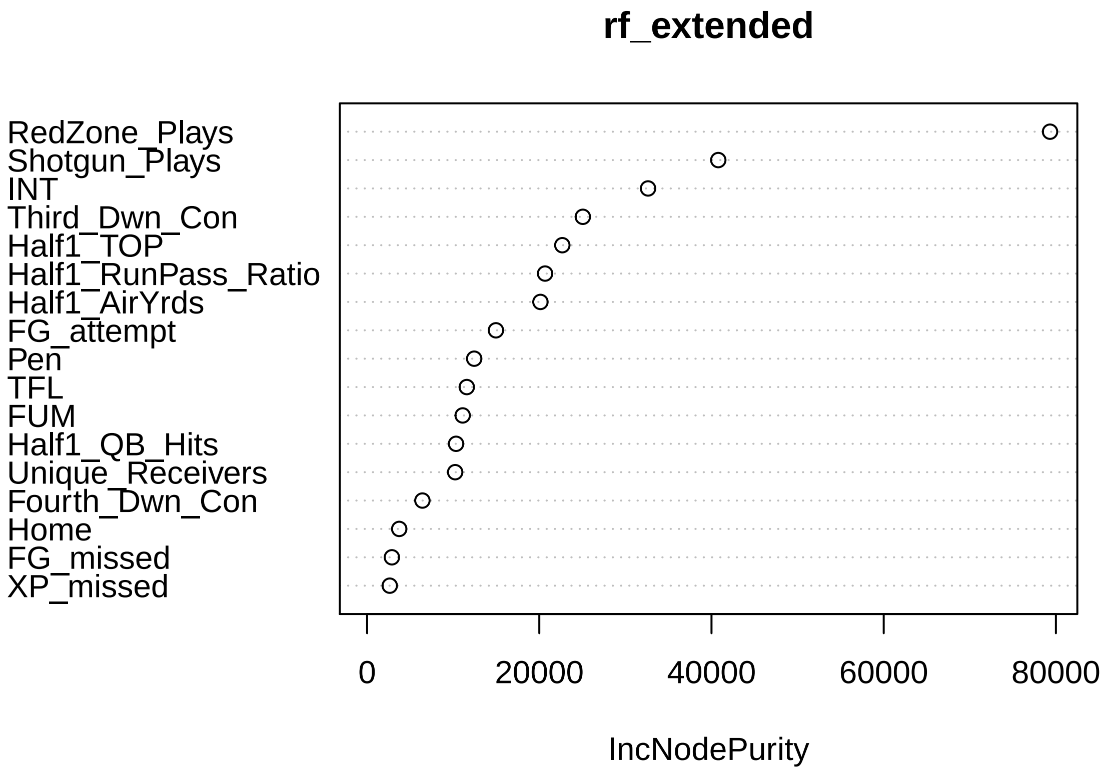
```
################ FIRST HALF PREDICTORS EXTENSION ################

library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
temp.df <- c()
for(i in c("17","18","19")) {
  temp.raw <- get(paste0("data", i))
  for(j in unique(temp.raw$game_id)) {
    temp <- subset(temp.raw, game_id == j)
    for(k in c(temp$home_team[1], temp$away_team[1])) {
      Half1_RunPass_Ratio <-
        sum(temp$rush_attempt[temp$posteam == k & temp$game_half == "Half1"], na.rm=TRUE) /
        sum(temp$pass_attempt[temp$posteam == k & temp$game_half == "Half1"], na.rm=TRUE)
      Half1_AirYrds <- sum(temp$air_yards[temp$posteam == k & temp$game_half == "Half1"], na.rm=TRUE)
      Half1_QB_Hits <-  sum(temp$qb_hit[temp$posteam == k & temp$game_half == "Half1"] == 1, na.rm=TRUE)
      ## Calculate Time of Possession
      pos <- period_to_seconds(ms(temp$time[temp$posteam == k & temp$game_half == "Half1"]))
      Half1_TOP <- abs(sum(c(0, lag(pos)[2:length(pos)]) - pos , na.rm=TRUE))
      ## build final DF with data to add
      temp.df <- as.data.frame(rbind(cbind(Half1_RunPass_Ratio,
                                           Half1_AirYrds,
                                           Half1_QB_Hits,
                                           Half1_TOP),
                                     temp.df))
    }
  }
}

Extended_Data <- cbind(data_2017_19[,!(colnames(data_2017_19) %in% c("QB_Hit", "Rush_Yrd", "Pass_Yrd"))], temp.df)
```

```
rf_extended <- randomForest(Score_Dif~.-Game_id-Team_Name-year,
                      data=Extended_Data,
                      mtry = combs[which.min(combs$RMSE_test),]$mtry,
                      maxnodes = combs[which.min(combs$RMSE_test),]$maxnodes,
                      ntree=200)
varImpPlot(rf_extended)
```



```
step.backward.extended = step(lm(Score_Dif~.- Game_id - Team_Name -year, data=Extended_Data), scope = c(lower = formula(Score_Di

step.both.extended = step(lm(Score_Dif~.- Game_id - Team_Name -year, data=Extended_Data), scope = c(lower = formula(Score_Dif ~1
```

```
We also conducted backwards/both selection based on AIC to see if the new 1st half variables are present. We see that backward a
```

Conclusion

National Football League data provides a ripe opportunity for statistical analysis, especially when able to access play-by-play level data using the `nflfastR` package. While most variables have a slight right skew to them and only take on positive values, this did not present tremendous challenges for our models. Going forward, we're interested to continue investigating the impact of variables that provide a clear relationship towards "success" in a game–such as total passing yards or total rushing yards–as well as more "strategic" variables such as the number of third and fourth down conversions. Given the violations of symmetric distributions in many of the predictors and the potential for non-linear relationships to arise, examining the performance and interpretations of Random Forest models will also be a focus in our analysis.