Final Project

R Markdown

Introduction

The objective is to accurately predict the Golden State Warriors winning percentage using several counting and efficiency statistics.

Context

The Golden State warriors are an NBA team based in San Francisco. They have been one of the most dominant basketball teams of the last decade and have won three championships in that time frame. They currently hold the record for best the best record in a season ever with 73 wins and 9 losses. This model will use data from four seasons: 2021-2022 (the most recent season), 2020-2021, 2018-2019, 2016-2017. I chose these seasons because I believe they are the best representation of Golden States core players and system, (I left out 2019-2020 due to the Warriors star player, Stephen Curry's injury and absence).

Additional Curriosities (Optional Read)

Whenever there is team success in basketball, we often hear talk of "the intangibles". These are things players do that don't show up in the stat sheet. For one example, take a player who defends well and forces the teams opposing shooter to miss. This would not be recorded unless the player blocks the shot or steals the ball. However, many of these "intangibles" show up in the stat sheet in other ways. Take that same example from before. When players defend well, not only do they force more open shots, but they have more opportunities for a rebound, so we may see team rebounds go up when players are defending well. Another way this could show up is in the teams field goal percentage. When a team forces a missed shot, they often look to push the ball up the court before their opponents defense can get set, leading to more open shots and often a higher field goal percentage. This makes me curious if team success can actually be, somewhat accurately explained by these 16 statistics I have chosen.

Loading and Tidying the Data

The csv file containing the data was created from data on sports reference. The website allowed me to remove any columns I didn't want prior to downloading the csv files for each season. I then combined them into one csv file and formatted the names of the predictor variables.

```
# read in data
gsw <- read_csv('Golden State Combined.csv') %>% rename(W_L = 'W/L', FG_per = 'FG%', Three_per = '3P%',
```

```
## Rows: 318 Columns: 20
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr   (2): Opp, W/L
```

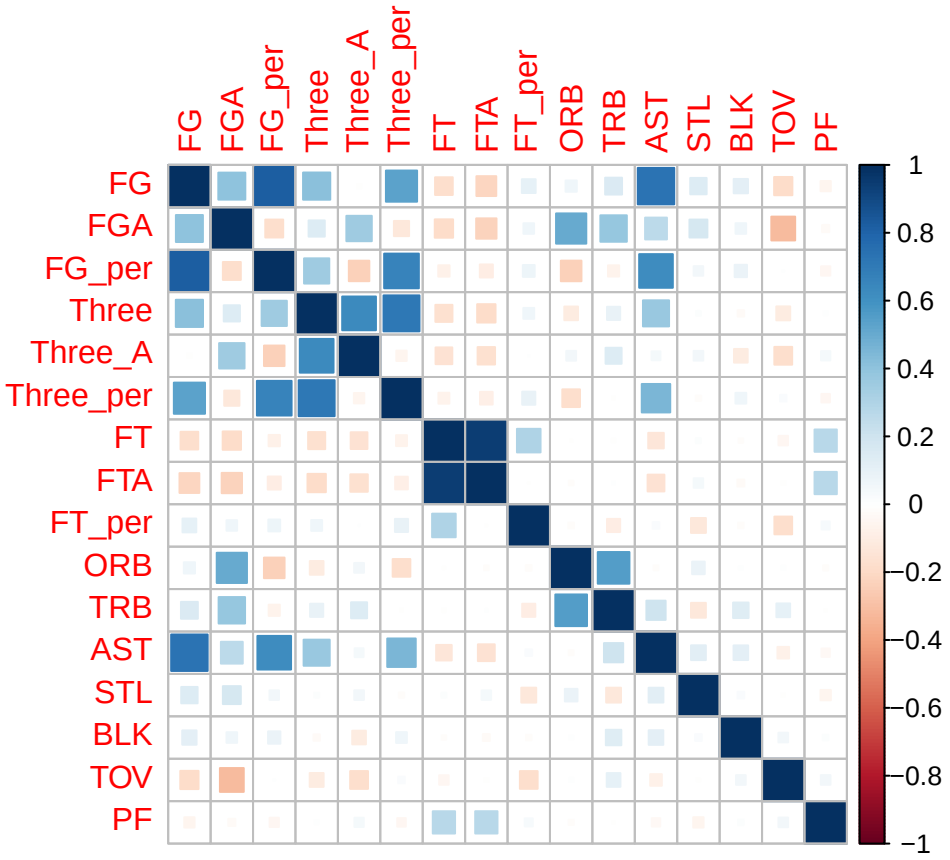
```
## dbl  (17): Gm, FG, FGA, FG%, 3P, 3PA, 3P%, FT, FTA, FT%, ORB, TRB, AST, STL,...
## date  (1): Date
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
# order the data starting with the most recent game
gsw <- gsw[order(gsw$Date, decreasing = TRUE),]

set.seed(608)

# Convert the Win Loss variable to factor
gsw$W_L <- as.factor(gsw$W_L)
```

Exploratory Data Analysis

I will be conducting Exploratory Data Analysis on the entire data set because the set itself is only 318 observations. There will be a total of five plots that will give me an idea about the way the data is distributed.

We will begin with a corrplot.



Expected Findings:

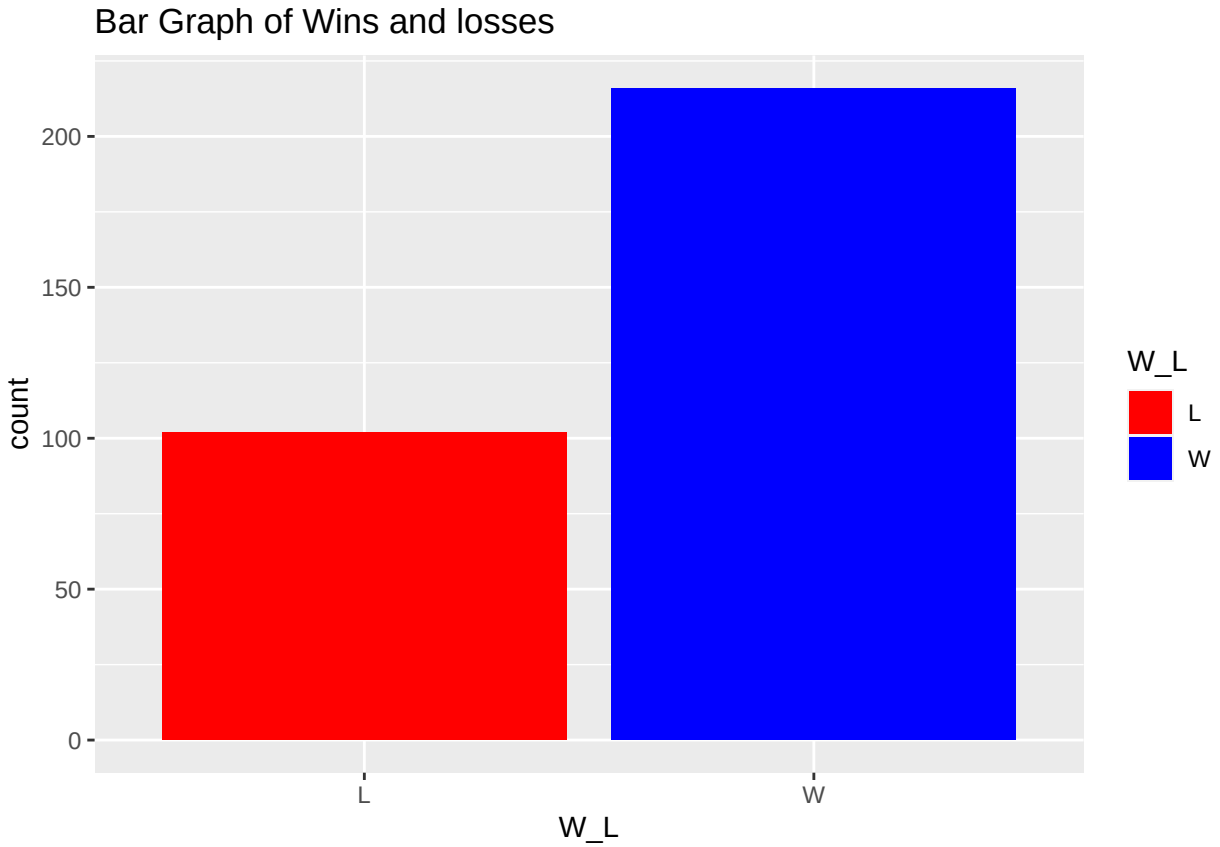
1. AST and FG/FG% - Assists often involve passing to a player who has a more open shot/drive than

yourself. Higher assists usually mean higher shot quality, thus better efficiency and more shots made. Positive correlation is expected.
2. TRB and ORB - Getting an offensive rebound increases the total rebounds stat. Positive correlation is expected.
3. TOV/FGA - Higher turnovers lower possessions and consequently less opportunities for field goals. Negative Correlation is expected.

Unexpected Findings:

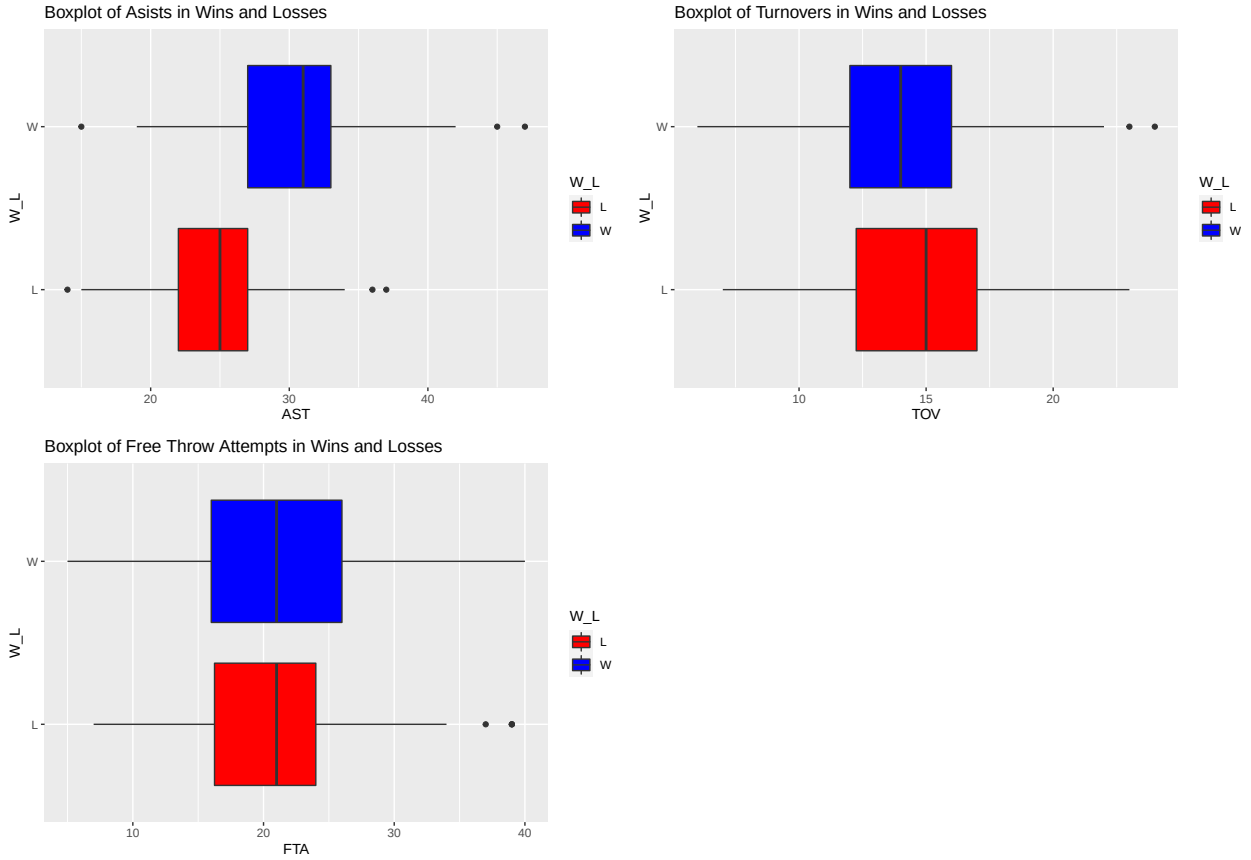
1. AST and TOV - Higher assists usually implies more ball movement and passing which can lead to bad pass turnovers. I expected to see a stronger positive correlation between the two.

Here is a simple bar graph showing the distribution of the W_L variable



There are about twice as many wins as there are losses, so stratifying on the W_L variable for my samples is very important. In order to avoid fitting on models that have too high a ratio of wins to losses, I will also stratify the folds as well.

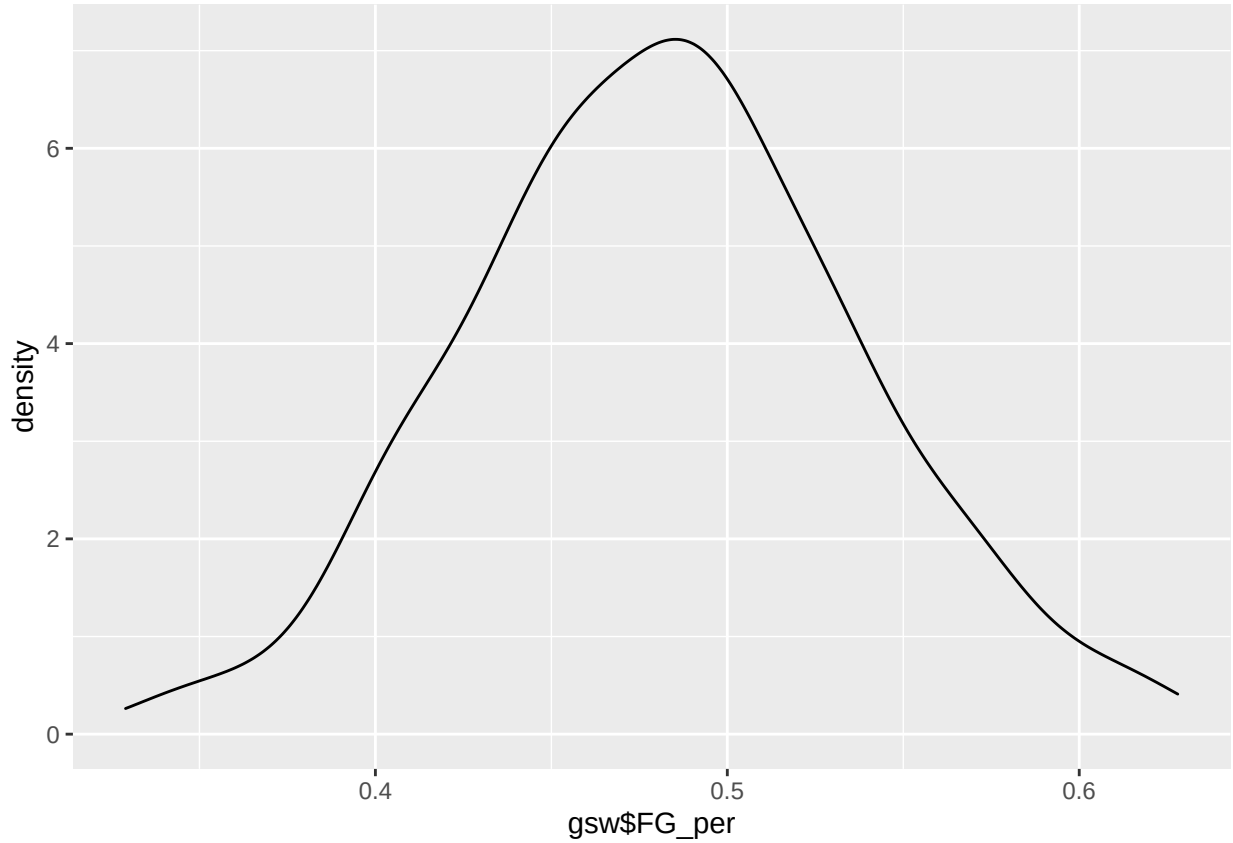
Now we will take a look at three different variable distributions in Wins vs. in Losses.



Here we see a dramatic difference in team assists in wins vs losses. The warriors tend to have more assists in wins and slightly more turnovers in losses. On average they have around the same amount of free throws, but the ceiling is higher wins. It's no surprise that the most drastic difference in wins vs. losses comes from assists. These warriors team's offenses relied heavily on ball movement to create open shots.

Lastly, let's take a look at the density curve of Field Goal Percentage.

```
## Warning: Use of ‘gsw$FG_per‘ is discouraged. Use ‘FG_per‘ instead.
```



Most games they have a field goal percentage between 45 and 55 which is considered above average. Generally shooting below 40 percent from the field is considered bad for a time while shooting above 50 percent is very good. There are some extremes on both ends, but generally they shoot well.

Three Point Shooting in the NBA (Optional Read)

Before we go any further, it's important that I take some time to explain the evolution of three point shooting, and its importance in basketball.

This is a shot showing the most frequent shot locations over 12 seasons.

A couple decades ago, the three pointer was considered a bad shot the majority of the time, and the graphic agrees with that idea early on. In the picture we see a consistent, albeit slow, increase in percentages of three point shots (shots from outside the outermost arc) taken. That is, until the 2015/16 season. The warriors groundbreaking offence, lead by Steph Curry, is the biggest cause for the huge increase in shots taken from behind the arc. After the warriors success, the rest of the league followed suit, and begun focusing more of there energy on creating open shots from behind the three point line. Steph, who is almost universally considered the greatest shooter of all time, had a huge part in the shifting of the NBA to a three point-heavy offense with his historic three point efficiency on high attempts. Ten years ago you would be lucky so see ten made three pointers from a team. As of today Steph curry has made 10 three pointers in a game 22 times himself. The influence of the warriors is seen in every team in the NBA, and today we regularly see players who will take half of their shots from the three point line. For this reason, I expect that three point attempts, three point makes, and three point percentage will all play a large part in predicting wins and losses. Today, the NBA is more reliant on the three than it ever has been.

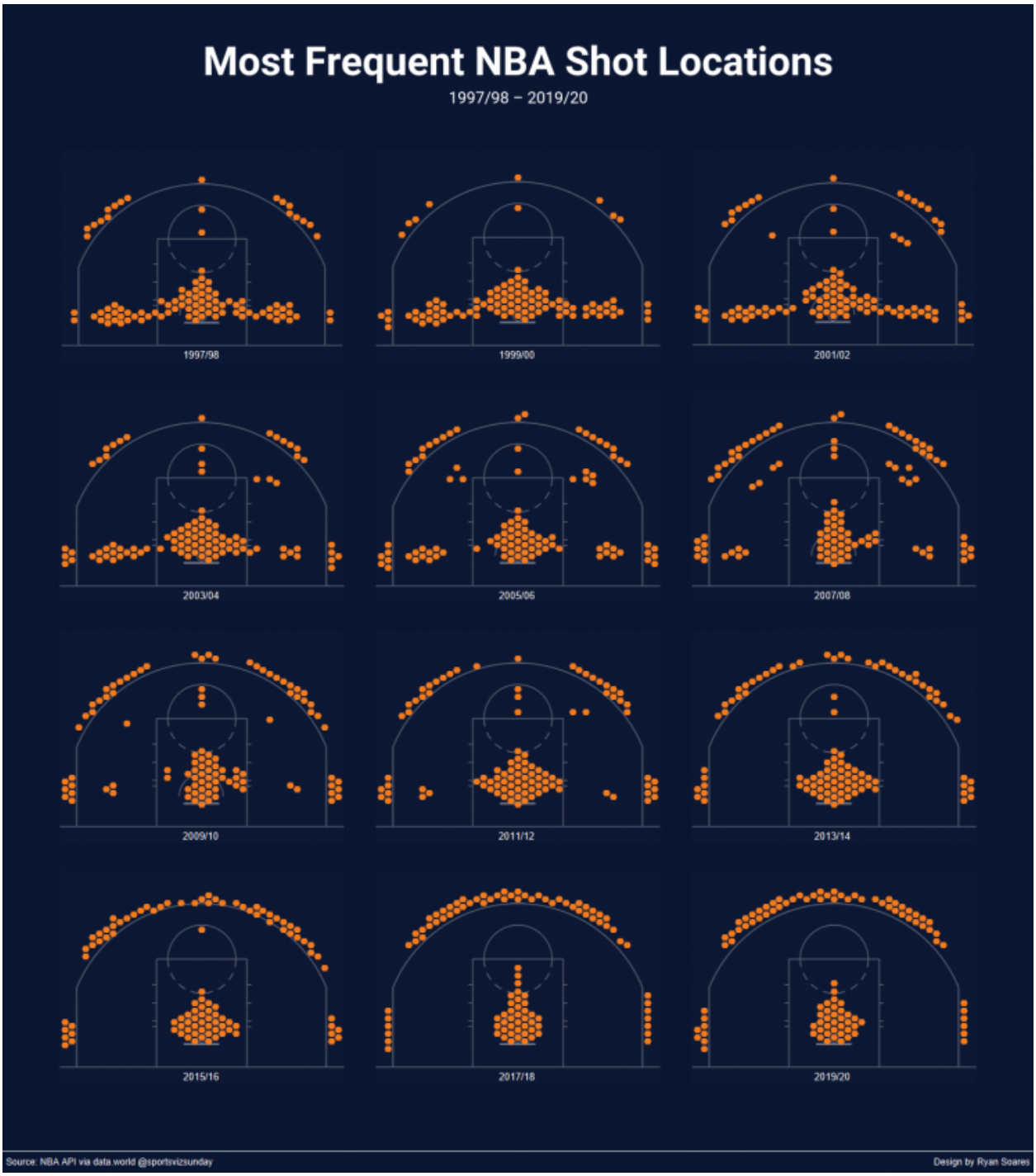


Figure 1: Source; 'Epic Visualization of Every NBA Shot Taken Since 1997'

Getting ready to fit out models

Splitting the Data

```
#Split the data using stratified sampling on the Wins and Losses variable
gswSplit <- initial_split(gsw, prop = 0.80,
                                 strata = W_L)

# Store the training and Test sets in variables
gsw_train <- training(gswSplit)
gsw_test <- testing(gswSplit)
```

The testing Data has 65 observations, while the training data has 253 observations. The data was split 80/20.

Creating the Recipe

Here I will create the recipe I will be using when fitting the models, as well as the folds that will be used for k-fold cross validation.

```
# Create recipe, center and scale the predictors
gswRecipe <- recipe(W_L ~ FG + FGA + FG_per + Three + Three_A + Three_per + FT + FTA + FT_per + ORB +TRB

# Create k=5 folds
gsw_fold <- vfold_cv(gsw_train, v = 7, strata = W_L)
```

Building and Running the models

Random Forest Model

The first model I will be fitting will be a random forest model.

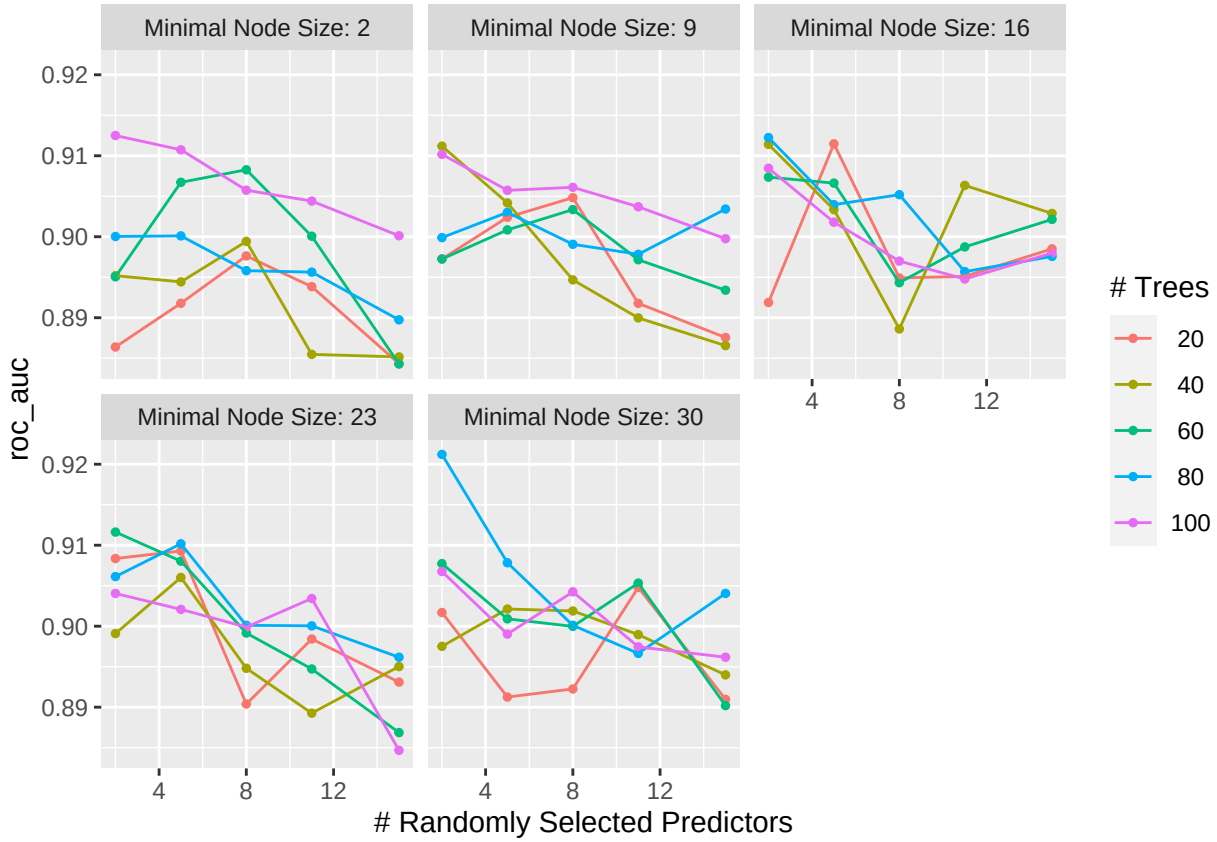
```
# Build Random Forest Model and tune the parameters
rf <- rand_forest(mtry = tune(), min_n = tune(), trees = tune()) %>%
  set_engine("ranger") %>%
  set_mode("classification")

# create workflow and add model, recipe
rf_wf <- workflow() %>%
  add_model(rf) %>%
  add_recipe(gswRecipe)

# set up grid
rf_grid <-  grid_regular(mtry(range = c(2, 15)), trees(range = c(20, 100)), min_n(range = c(2, 30)), lev
```

For this model I will tune the number of trees, the number of predictors sampled at each split, and the minimum number of observations required for another split.

```
# tune parameters with grid and cross validation
tune_rf <- tune_grid(
  rf_wf,
  resamples = gsw_fold,
  grid = rf_grid
)
```

Now lets use autoplot to take a look at AUC of the model with different parameter values.

```
autoplot(tune_rf, metric = 'roc_auc')
```



Based on the graphs we can conclude the that the best model would have the following parameters: mtry = 2 trees = 80 min_n = 30

```
show_best(tune_rf, metric = 'roc_auc') %>% select(-.estimator, -.config)
```

```
## # A tibble: 5 x 7
##    mtry trees min_n .metric  mean     n std_err
##   <int> <int> <int> <chr>   <dbl> <int>   <dbl>
## 1     2    80    30 roc_auc 0.921     7  0.0145
## 2     2   100     2 roc_auc 0.912     7  0.0142
## 3     2    80    16 roc_auc 0.912     7  0.0149
## 4     2    60    23 roc_auc 0.912     7  0.0189
## 5     5    20    16 roc_auc 0.911     7  0.0119
```

This function confirms our observations from the graphs.

```
best_rf <- select_best(tune_rf, metric = 'roc_auc')
```

Our best Random Forest model has a mean AUC_ROC of 0.9212121. Now lets see how well our other models perform.

Boosted Tree Model

Next, we will fitting a boosted tree model.

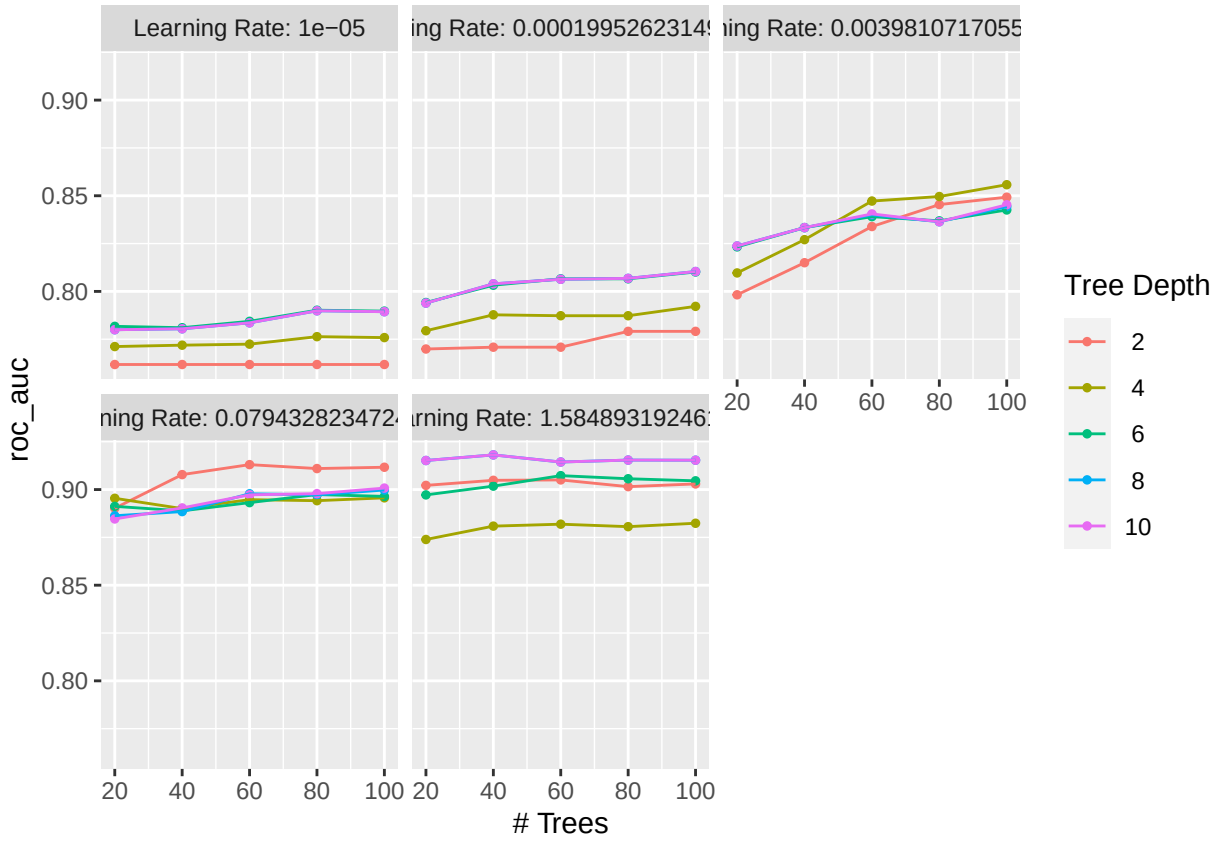
```
# build model tuning the tree depth learning rate and number of trees
gsw_boost <- boost_tree(tree_depth = tune(), learn_rate = tune(), trees = tune()) %>%
  set_engine("xgboost") %>%
  set_mode("classification")

# create workflow and add  model and recipe
boost_wf <- workflow() %>%
  add_model(gsw_boost)  %>%
  add_recipe(gswRecipe)

# set up grid
boost_grid <-  grid_regular(tree_depth(range = c(2, 10)), trees(range = c(20, 100)), learn_rate(range =
```

Here I will tune the parameters tree_depth, learn_rate, and trees.

```
# tune parameters with grid and cross validation
boost_tune <- tune_grid(
  boost_wf,
  resamples = gsw_fold,
  grid = boost_grid,
)
```

Again, lets take a look at the autoplot.

```
autoplot(boost_tune, metric = 'roc_auc')
```



From the graphs we can see the the best values are, tree depth of 8, a learning rate of 1.585, and around 40 trees.

```
show_best(boost_tune, metric = 'roc_auc') %>% select(-.estimator, -.config)
```

```
## # A tibble: 5 x 7
##   trees tree_depth learn_rate .metric  mean     n std_err
##   <int>      <int>      <dbl> <chr>   <dbl> <int>   <dbl>
## 1    40          8       1.58 roc_auc 0.918     7  0.0184
## 2    40         10       1.58 roc_auc 0.918     7  0.0184
## 3    80          8       1.58 roc_auc 0.915     7  0.0182
## 4    80         10       1.58 roc_auc 0.915     7  0.0182
## 5   100          8       1.58 roc_auc 0.915     7  0.0178
```

The model with these parameters gives us a mean AUC of 0.9181169, which does not beat out our random forest model.

Now lets store the model

```
best_boost <- select_best(boost_tune, metric = 'roc_auc')
```

Logistic Regression

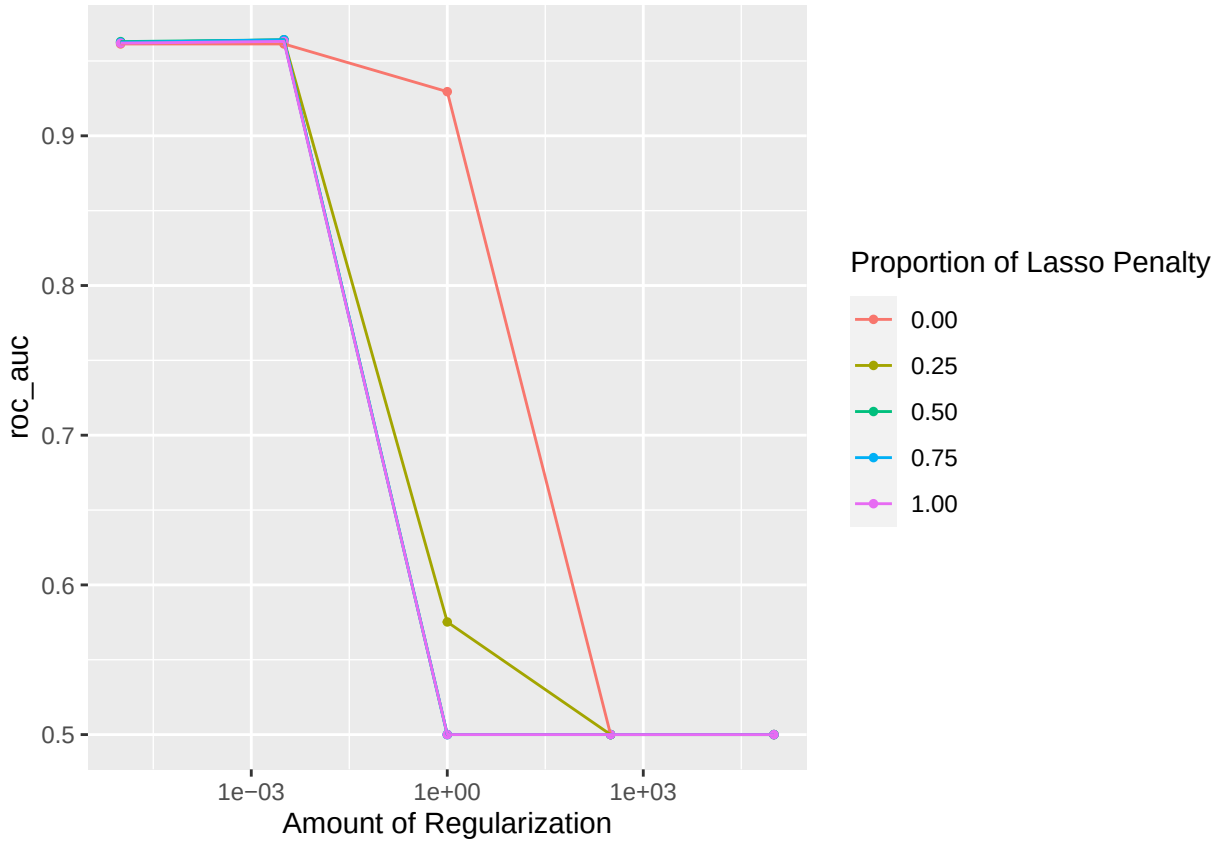
Since this is One-Class Classification variable (binary outcome variable), it seems like a good idea to fit a logistic regression model as well.

```
# Build Model and tune
gsw_log <- logistic_reg(penalty = tune(), mixture = tune()) %>%
  set_engine("glmnet") %>%
  set_mode("classification")

# create logistic workflow
log_wf <- workflow() %>%
  add_model(gsw_log) %>%
  add_recipe(gswRecipe)

# set up grid
log_grid <- grid_regular(penalty(range=c(-5,5)), mixture(range= c(0,1)), levels = 5)
```

We will tune the penalty (amount of regularization), and the mixture (proportion of lasso regularization).

```
log_tune <- tune_grid(
  log_wf,
  resamples = gsw_fold,
  grid = log_grid,
)
```

```
autoplot(log_tune, metric = 'roc_auc')
```



For this plot it would likely be easier to show the metrics before making assumptions because both values are continuous.

```
show_best(log_tune, metric = 'roc_auc') %>% select(-.estimator, -.config)
```

```
## # A tibble: 5 x 6
##   penalty mixture .metric  mean     n std_err
##     <dbl>   <dbl> <chr>   <dbl> <int>   <dbl>
## 1 0.00316    0.75 roc_auc 0.964     7  0.0121
## 2 0.00316    0.25 roc_auc 0.964     7  0.0122
## 3 0.00316    0.5  roc_auc 0.964     7  0.0122
## 4 0.00316    1    roc_auc 0.963     7  0.0120
## 5 0.00001    0.5  roc_auc 0.963     7  0.0124
```

Here we see our best values are, penalty = 0.02154435 and mixture = 0.003162278. This model has the best mean AUC yet at 0.9641558, which makes it considerably better than our Random Forest and Boosted trees Models. Additionally, the high mixture value tells us the there is a high proportion of lasso regularization being used.

Now lets store this model.

```
best_log <- select_best(log_tune, metric = 'roc_auc')
```

Logistic regression is now the model to beat.

K-Nearest Neighbors

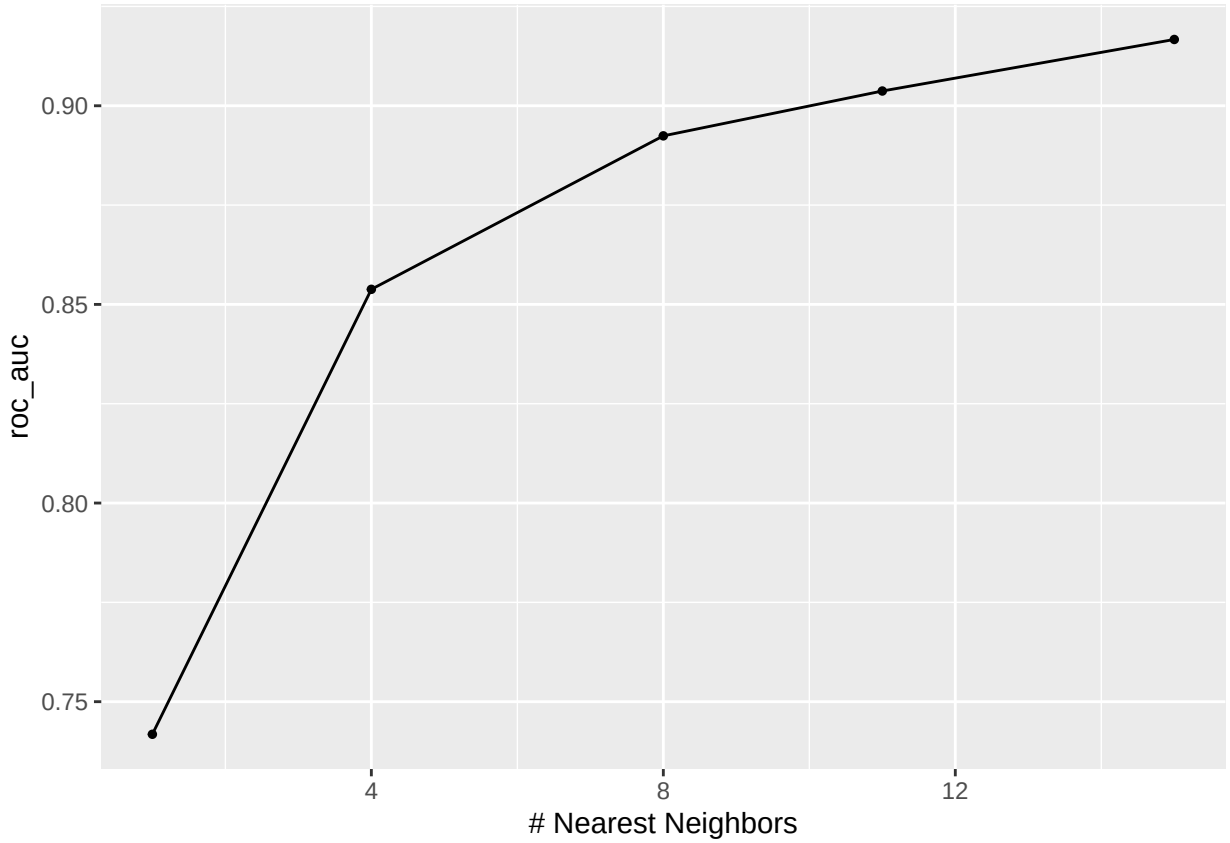
The last model I will be fitting is a K-Nearest Neighbors model.

```
# Build the model
gsw_knn <-
  nearest_neighbor(
    neighbors = tune(),
    mode = 'classification') %>%
  set_engine("kknn")

# Create a workflow object
knn_wf <- workflow() %>%
  add_model(gsw_knn) %>%
  add_recipe(gswRecipe)

# set up our grid
knn_grid <- grid_regular(neighbors(range = c(1,15)), levels = 5)
```

```
tune_knn <- log_tune <- tune_grid(
  knn_wf,
  resamples = gsw_fold,
  grid = knn_grid,
)
```

Now, we tune the neighbor parameter.

```
autoplot(tune_knn, metric = 'roc_auc')
```



So it seems here that the best value for neighbors is 15.

```
show_best(tune_knn, metric = 'roc_auc') %>% select(-.estimator, -.config)
```

```
## # A tibble: 5 x 5
##   neighbors .metric  mean     n std_err
##       <int> <chr>   <dbl> <int>   <dbl>
## 1        15 roc_auc 0.917     7  0.0190
## 2        11 roc_auc 0.904     7  0.0215
## 3         8 roc_auc 0.892     7  0.0241
## 4         4 roc_auc 0.854     7  0.0354
## 5         1 roc_auc 0.742     7  0.0301
```

With k = 15, we are given a mean AUC of 0.9166883. This still does not beat our Logistic Regression model. For consistency we will still store our model.

```
best_knn <- select_best(tune_knn, metric = 'roc_auc')
```

Finalizing the Model

Logistic Regression outperformed every other model. Now it is time ti finalize the model and fit it to our testing set.

```
# Finalize the workflow
finalwf <- finalize_workflow(log_wf , best_log)

# fit finalized workflow
final_fit <- fit(finalwf, data = gsw_train)
```

```
augment(final_fit, new_data = gsw_test) %>% roc_auc(W_L, estimate = c(.pred_W))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary        0.0574
```

We see here that out AUC is much lower on the testing set than on the training set.

What Went Wrong

When seeing this problem, the first thing that comes to my mind is over fitting. Initially I had 5 folds, and got a similar AUC to this when running on the testing set. After adding more folds, I thought the AUC would significantly increase. It is possible that there is still over fitting in this model, but their is another issue to consider. Logistic regression assumes that the predictor variables are linearly related to log odds. If this assumption is faulty in this case, it may explain why this model fails. For, further investigation, I am going to finalize the second best performing model.

Finalizing the Random Forest Model

The second best performing model is the Random Forest Model with a mean AUC of 0.9212121. Let's fit this one.

```
# Finalize the workflow
finalwf2 <- finalize_workflow(rf_wf , best_rf)

# fit finalized workflow
final_fit2 <- fit(finalwf2, data = gsw_train)
```

Now lets see if we can get a better AUC.

```
augment(final_fit2, new_data = gsw_test) %>% roc_auc(W_L, estimate = c(.pred_W))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.145
```

This gives us an AUC of 0.1493506. Which is Abysmal. However, after doing some research on why my model went wrong, the answer could have been right in front of my face. The overlap between total rebounds and Offensive rebounds calls for an interaction step. Similarly there is overlap between three point percent and field goal percentage, as well ass three point makes and field goal makes. Since I am not satisfied with my models performance, I will quickly make a new recipe, and run it with my random forest model, as it was my best performing non linear model.

A Quick Rework

First lets rework the recipe.

```
UpdatedRec <- recipe(W_L ~ FG + FGA + FG_per + Three + Three_A + Three_per + FT + FTA + FT_per + ORB +T
```

Now I will simply go through the previous steps using the updated recipe.

I hid some code to avoid repetition.

```
show_best(utune_rf, metric = 'roc_auc') %>% select(-.estimator, -.config)
```

```
## # A tibble: 5 x 7
##    mtry trees min_n .metric  mean     n std_err
##   <int> <int> <int> <chr>   <dbl> <int>   <dbl>
## 1     5    80     2 roc_auc 0.912     7  0.0168
## 2     2    60     9 roc_auc 0.912     7  0.0221
## 3     8    80     9 roc_auc 0.909     7  0.0190
## 4     8   100    30 roc_auc 0.909     7  0.0196
## 5     2    60     2 roc_auc 0.908     7  0.0190
```

```
ubest_rf <- select_best(utune_rf, metric = 'roc_auc')
```

We are given an AUC of 0.9167100 on the training set. Which is similar to the precious AUC.

```
# Finalize the workflow
ufinalwf <- finalize_workflow(urf_wf , ubest_rf)

# fit finalized workflow
ufinal_fit <- fit(ufinalwf, data = gsw_train)

augment(ufinal_fit, new_data = gsw_test) %>% roc_auc(W_L, estimate = c(.pred_W))
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.116
```

This leaves us with an AUC of 0.1504329 , which is comparable to the previous one. Having seen my models fail three times despite doing everything I could think of to do, I am ready to make my conclusion.

Conclusion

After running my models with increased folds, added interaction steps, and at different levels in the grid (not included here), I have come to the conclusion that simple counting and efficiency stats are not nearly enough to predict winning. It was a very optimistic project from the start, and to think I could come close to predicting something as complicated as a basketball game with these simple stats was ambitious, to say the least. Despite the conclusion, I am extremely happy with the work that I put in. It is very possible that there are intangibles that don't show up in any stat sheet. Additionally, it's possible that predicting a win can better be done with more obscure, advanced basketball stats such as RAPTOR, DRAPTOR, and TS% (true shooting). This will definitely be something I will explore in the future. As of now, I will operate on the assumption that "intangibles" do play a huge factor in winning.