2020 College Basketball Tournament
Final Report

By:
Matthew Forey
Nathan Keckley
Andrew Locker
Ricardo Munoz

The University of Texas at San Antonio

DA 6813

Executive Summary

Every March, basketball fans around the country do their best to predict the outcomes of college basketball games during the annual NCAA Men's Basketball Tournament. Fans compete to predict the most wins correctly by filling out a bracket for all 63 games that are played over 3 weeks. Historically, many people have been able to predict a large number of early round games, but it is extremely rare for anyone to continue picking correct winners as upsets occur and the variations of possible matchups increase in later rounds of the tournament. With this year's tournament canceled, we were motivated to attempt to find a model that will help us predict games accurately in a couple different metrics, such as total scores, point differential, and classify a team as a winner or loser.

There are many reasons why one basketball team might be selected to win over another team when predicting who will win a game. The basketball statistics that teams accumulate throughout the season can create a descriptive framework of how a team may perform better in some areas than others. For example, statistics like points per game or field goal percentage can give a good description of how well a team performs on offense, just as statistics like points allowed per game or forced turnovers could be indicators of how strong a team is on defense.

We will present a model that performs well using team statistics variables for two opposing teams to predict game winners for any particular game. Since the statistics of each team change year over year we will have to use a holdout sample to test the data. We hope to create models that accurately represent which predictors are important for each of the three response variables. As basketball fans we hope to be able to use these models for monetary gain for years to come.

Problem

The purpose of the study began with the simple idea that as sports fans we love to predict the outcome of a game. Truth be told there is no better feeling that telling your friends and coworkers what will happen in a game and have your opinions hold true. As we continued to discuss just how far we can take this analysis we realized there might be a monetary opportunity behind the data. For many years Las Vegas oddsmakers have made a fortune by setting a strong line that will attract bettors on both sides of the bet. The group is convinced that the spreads or odds that the bookmakers put out are skewed in a direction where most people will typically bet. For example, we believe that a good basketball team will typically be overvalued due to their popularity. After further discussion, the group wanted to analyze the following questions.

- Which team should win the game?
  The most popular question for all sports fans is who is going to win the game. Using logistic regression we plan to calculate the true odds of a team winning a game and see if there is an edge on either team. For simple terms, if Las Vegas has odds predicted that a particular team has a 60% chance to win a game, but we project that probability to be closer to 70% there might be a monetary advantage.

- What predictors (statistics) are most important for winning games?

The landscape of the game of basketball has been changing over the last few years. The group is hopeful to find out which factors lead to wins more often than others. For context we are looking at whether a team's offense or defense is more important in determining who wins the game. The group will analyze many variables and try to pinpoint which statistics are the most important.

- Is there an edge to be found between what we project the game total to be and what oddsmakers think?

Similar to the first question proposed the group is confident we can find a discrepancy between our model and betting odds. The most common way to bet on a game is to give a particular team some points to start the game in order to make the game fair. Additionally, a common bet is to predict whether the combined score for both teams in a game will go over or stay under the number that bookmakers have stated. For example a game between Texas and Duke may have a projected total of 150 points. With the use of linear regression we are hopeful to find a large difference between our model and the total that may be a worthwhile investment.

Literature Review

Sports have been around since the dawn of civilization, and inevitably following suit, was the market for predicting the outcome of games, whether it was taking bets at the coliseum, or a friendly argument between peers. Sports modeling and analytics on the other hand, is relatively a more novel concept. Analytics has a wide use in sports from helping optimize teams to predicting outcomes. When it comes to predictive modeling, there are many top statisticians in the field working and driving progress.

Dean Oliver, an NBA coach, who is referenced more for his work in statistics and analytics, helped coin the "Four Factors". These four factors are the most significant drivers in determining the outcome of a game. They include(in order of significance) Effective field goal percentage, Turnover Percentage, Offensive Rebound Percentage, and Free Throw Rate. These four factors help account for many aspects of a team's overall skill and these key stats end up having large roles to play in making the most of their possessions and being effective, along with earning extra possessions. These "Four Factors", attributed to Dean Oliver, are unanimously accepted by everyone in the fieldand should be weighted as such in modeling.

Another noteworthy statistician in the area of college basketball, Ken Pomeroy, broke a lot of ground in introducing the need to normalize the variables to be able to measure a lot of the team variables on a scale relative to the rest of the league. This is important because it helps compare the statistics to the level of their competition and helps provide an appropriate weight. He also proposed adding built in weights to variables such as the home teams' stats being granted an extra weight of 1.014 to account for an advantage. Pomeroy also emphasizes transforming all appropriate variables into percentage rates, to normalize the numbers to reflect an average efficiency such as turnover rate percentage, instead of turnover count, to put it in a proper perspective.

In this study (cited below), Shi, Moorthy, and Zimmerman tried various models: a decision tree, rule learners, neural networks, and an ensemble method. By using adjusted efficiencies suggested by Ken Pomeroy, and the important variables such as Oliver's "Four Factors", they were able to obtain a maximum accuracy slightly above 70%. This value was expected, after the season had been going on for over 40 days, the accuracy neared the glass ceiling observed by most statisticians.

Drawing conclusions from their findings, we can make a few assumptions. As far as expectations, we can hope to achieve as high of an accuracy of 70-75% at our peak performance, due to these glass ceilings from the nature of sports itself. There are a few "intangibles" we can try to account for such as luck, leadership, and home field advantage, which have been attempted to be measured by those that are top in their field such as Pomeroy and are publicly available. There will still be a lot of variability left in the air, as the athletes driving the sport itself are amateurs. For our purposes, it would appear that a more robust model such as a support vector machine would help stabilize the model from large outliers. Since we are trying to classify wins and losses, and find value between professional sports lines, it would be ideal for our predictive models to not be heavily influenced by significant outliers.

Zimmermann, A., Moorthy, S., & Shi, Z. (1970, January 1). [PDF] Predicting NCAAB Match Outcomes Using ML Techniques - Some Results and Lessons Learned: Semantic Scholar. Retrieved from https://arxiv.org/pdf/1310.3607.pdf

Methods

The analysis that the team is prepared to do on the data will consist of linear and logistic regression, tree-based models, and discriminatory analysis. Alongside the creation of machine learning models, the team will also prepare visualizations to further understand the data.

The logistic regression of the project will focus on trying to predict who will win a particular game. Given that the final score of a game is given in the data the team will easily check what team won the game and create a binary column by the name of win. Using the predictors in the dataset, the team will look to create new predictors based on the given variables. Once all the predictors have been prepared, we will analyze which predictors most influence the accurate prediction of a team winning the game.

The linear regression portion of the project will include the team trying to predict two different variables. First, we will focus on trying to predict what the total score of a game should be given two different teams. Secondly, we will try to accurately predict which team should be favored to win the game and by how much. This will allow us to figure out what the “point-spread” should be. A point-spread is a common term used in the gambling industry that tries to make the playing field even. For example, a very good team will play against a subpar team and will win most of the time. If the subpar team is given a certain amount of points to start the game off, the result of the game could change and thus make for a much closer game. Both the logistic and linear regression will require the same assumptions of normality in its data.

The team will also look at other separate models to try and predict each of the three objectives stated above. Some of the models that the team has in mind include but are not limited

to Random Forest, Decision-Trees, and SVM models. We want to explore which predictors affect the three different response variables the most. Additionally, we are interested in seeing if there are certain predictors that closely recreate what we see on tv. The landscape of the game of basketball has changed in a way that many teams are playing at a much faster pace and shooting more three pointers. We are also curious to see if we can find out whether offensive or defensive stats lead to more victories. Lastly, we are hoping to find an edge in the betting market but efficiently predicting what a given total or point spread should be and compare it against the Las Vegas odds.

Data

The initial dataset consists of 10,518 observations and 59 predictors that describe the data. The data consist of all the college basketball games between two Division-I basketball teams for the 2019-2020 season. Some of the predictors include but are not limited to the score of the game, how fast a team plays and how efficiently either team shoots the ball. In college basketball the team's stats change with each given year, we plan on using the first few months of data as our training set and then use the final two months as our data set, as opposed to pulling in prior years observations.

After digging deep into the data the group was able to determine that there weren't any major issues with the predictors. Figure 1 of the appendix shows some of the major predictors used in the models. The only predictor that was skewed was the strength of schedule variable, so the group created a margin variable between the two teams and used that non-skewed variable instead. Our next step was to make sure that our response variables had no issues. To study the game spreads, the group created the margin variable which was simply the winning or losing margin for the home teams. The total variable was a simple addition of the total game score and will be used to predict the number of games that should be scored in a given game. After further investigation the group was not able to find any issues with these two variables, as shown in figure 2 of the appendix. Our last created variable will be the winner of any given game in our dataset. For example, we could use the home team as the 'positive class' and if they won the game, the column would have a 1. Once this was created we were ready to move on to the modeling section of the analysis.

Results

As mentioned previously, the group will be building three different aspects of the dataset. First we will take a look at the results for our game spreads models. These models were created to find if there were any discrepancies between the projected oddsmakers spread and our model spread. The group was able to build 8 models, and their respective results are shown below. The comparison metric used for the models was the mean squared error.

Model	Mean Squared Error
Linear	**127.24**
Ridge	127.26
Elastic-Net	127.28
Decision Tree	151.36
PLS	127.26
PCR	127.29
Random Forest	140.29
SVM	127.93

We can see from the table above that the best model in terms of mean squared error was the linear model. The ridge and elastic net fell slightly behind and the ensemble methods proved to not be effective in this case. Now that we had our best model, the team wanted to explore if there was an opportunity for profit. The testing set, which consisted of 1,764 observations of games between February and March, was used to compare our predictions to the actual returns. It is important to note that before the analysis was conducted it was agreed upon by the group that the purpose of the model was not to bet every single game. Instead, it was purposed that we only bet games were there was a sizeable difference between our model and the oddsmakers predictions. In the gambling industry a common measure of profitability is by looking at units won. For the purpose of this analysis we will simply look at return on investment.

The table below shows how often our model recommended a winning bet in our testing set. We first see that the model is accurately able to predict about 55% of the games. This number was encouraging as game spreads are supposed to make the game a coin flip. Nevertheless, since the sportsbooks collect on a premium on every bet simply clearing a 50% accuracy would not dictate a profit. We see that the model did a good job of generating a winning bet when the difference between our model and the oddsmakers is greater than 6 points. The model preforms even better as the difference in predictions increases. The group was particularly interested with the sample size of these predictions. We see that when the difference between predictions is greater than 10 points we would have made an ROI of 18%. A ROI of 18% is very good considering this is over the span of 2 months. Even though further studies of volatility in this model have to be studied, the fact that the model generated returns greater than the average expected return of the S&P 500 in a year are highly encouraging.

	Games	Right	Wrong	Accuracy	Profit (Units)	ROI
All games	1,764	972	792	55.10%	3.6	0.20%
Difference +- 3	1,025	583	442	56.88%	38.5	3.76%
Difference +- 4	825	474	351	57.45%	40.5	4.92%
Difference +- 5	621	368	253	59.26%	52.9	8.52%
Difference +- 6	462	279	183	60.39%	49.8	10.78%
Difference +- 7	317	192	125	60.57%	35.3	11.14%
Difference +- 8	218	136	82	62.39%	32.2	14.77%
Difference +- 9	139	86	53	61.87%	19.1	13.74%
Difference +- 10	75	48	27	64.00%	13.5	18.00%

The game totals aspect of the analysis had some mixed reviews. Before we dig into the findings it is worth mentioning that these predictions are simply whether more points will be scored than the oddsmakers or less. The same 8 models were created and the results, in terms of mean squared error, are shown below. We can see that the best model was a linear regression which barely beat out the svm model. After discussion as a group we decided to go with the linear model due to ease of understanding and lesser complexity. We can see that the ensemble methods were also poor predictors in this analysis. With a mse of 252.52 we will proceed with the linear model as our best model.

Model	Mean Squared Error
Linear	**252.52**
Ridge	252.94
Elastic-Net	252.86
Decision Tree	284.60
PLS	252.97
PCR	252.94
Random Forest	274.72
SVM	252.57

Similar to the game spreads, we wanted to explore the possibility of a profit using this model. The table below shows that we have some mixed results but definitely something encouraging that we can build on. First we can see that if we would have bet every game we would have made a return of 1.34%. Once, again the purpose of this model was not to bet every game but instead to figure out the profitability of big differences between our predictions and oddsmakers predictions. We see an encouraging increase at difference equals 4 but the fact the model does not improve in a linear manner had us a bit concerned. We do see some large return with difference of 9 and 10 points at 10% and 19% respectively. The group was encouraged by the results but was still concerned with the sample size. Additionally, we wanted to explore how the model preformed in high scoring games and low scoring games. We were able to figure out that our model preforms well on high scoring games but not so much on low scoring games. After careful consideration we came to the conclusion that the model was heavily concentrated on offensive stats and other defensive stats would need to be retrieved to improve the model.

	Games	Right	Wrong	Accuracy	Profit (Units)	ROI
All games	1,764	982	782	55.67%	23.6	1.34%
Difference +- 3	851	491	360	57.70%	45.9	5.39%
Difference +- 4	620	370	250	59.68%	58	9.35%
Difference +- 5	445	255	190	57.30%	20.5	4.61%
Difference +- 6	289	161	128	55.71%	4.1	1.42%
Difference +- 7	185	104	81	56.22%	4.5	2.43%
Difference +- 8	115	61	54	53.04%	-4.5	-3.91%
Difference +- 9	55	33	22	60.00%	5.5	10.00%
Difference +- 10	31	20	11	64.52%	5.9	19.03%
Prediction < 120	34	16	18	47.06%	-5.4	-15.88%
Prediction > 160	43	29	14	67.44%	10.7	24.88%

The final aspect of our model was to simply predict who would win a game. In this particular section there were three models that were created: a logistic regression, svm, and random forest. The table below shows just how well each model performed in terms of test set accuracy. Based on the results we see that the logistic model barely beats out the svm model. We previously mentioned that the "glass ceiling" that many basketball statisticians have referred to hovers around 70%. Based on our results we are not that far away from their predictions and we will look into more predictors, specifically defensive stats, in the future. The next step in this analysis would be to convert the probabilities into money line odds and figure out if any discrepancies exist. The purpose of this section was to create models that would simply predict a game that could be used in simple conversations or even the next year's end of season tournament.

Model	Accuracy	Sensitivity	Specificity
Logistic Regression	.6678	.6659	.6697
SVM	.6672	.6808	.6529
Random Forest	.5907	.5789	.6022

Conclusion & Recommendations

As previously mentioned the linear regression models were the best models used for game spreads and game totals. For classifying the winner of any given game the logistic regression was our best model. The group was very encouraged with the results of the game spreads model and hopes to improve on it to use it next season. We are hopeful that the addition of a few predictors could lower our mse and increase our R-squared. The plan as of right now is to search for opportunities where predictions between oddsmakers and our models are more than 6 points. For the game totals, the group will look into adding more defensive predictors before deciding if the model can create a consistent profit. Lastly, the logistic model for predicting game winners will be used to predict the end of year tournament next year.

After the conclusion of the analysis the group discussed how we could possibly improve these models and talk about other things we could do different. The first agreeance was that we had to incorporate more defensive stats. We saw in the analysis how our models that predicted high totals worked very well and we hope to improve on that model. Next we think it would be a good idea to capture hot streaks. College basketball is a very streaky game and if we can quantify just how important a winning streak is, we are sure we can improve our models. Our last improvement would be to find a way to quantify a player's injury. Injuries are a part of sports but if we could effectively predict how much a certain player's worth is to a team we are confident we can create better models and profit.

Appendix

Figure 1

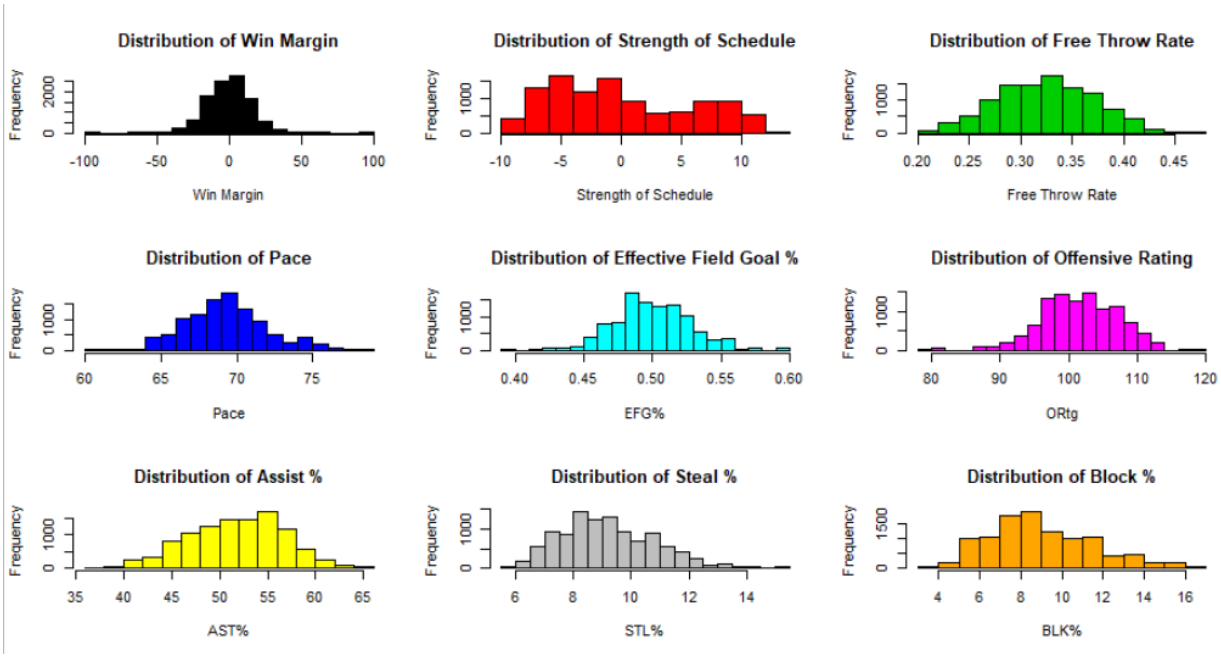
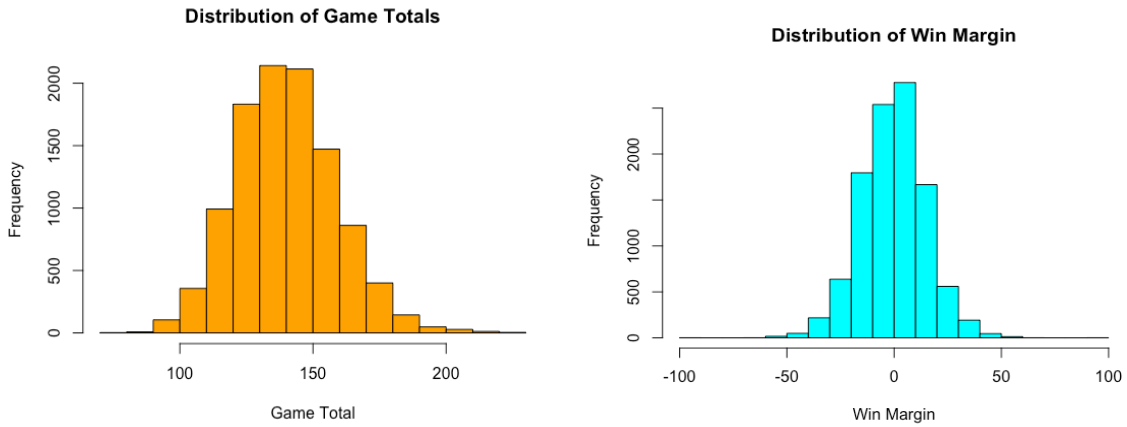


Figure 2



Code

```
library(e1071)
library(caret)
library(MASS)
library(ggplot2)
library(glmnet)
library(ggplot2)
library(corrplot)
library(partykit)
library(pROC)
library(tidyverse)
library(ggimage)
library(ggplot2)
library(tree)
library(randomForestSRC)
library(lattice)
library(ggthemes)


### Call the Function.R script that has functions to calculate accuracy
source("Functions.R")

### Read in the data
data <- read.csv("Dataset.csv", header = TRUE)

### Attach the data
attach(data)
### Create response variables to be used in analysis
data$WinMargin <- data$oppscore - data$teamscore
data$total <- data$oppscore + data$teamscore

hist(data$WinMargin, col = 5, xlab = "Win Margin", main = "Distribution of Win Margin")
skewness(data$WinMargin)

hist(data$total, col = "orange", xlab = "Game Total", main = "Distribution of Game Totals")
skewness(data$total)

data <- data[c(1:6,10:65)]

### Check the structure of the data
str(data)

#data$Win <- gsub("0", "Loss", data$Win)
#data$Win <- gsub("1", "Win", data$Win)
data$Win <- as.factor(data$Win)
```

```
### Make sure all predictors are in correct data type
data$Neutral_Location <- as.factor(data$Neutral_Location)
data$Inter_Conference <- as.factor(data$Inter_Conference)
data$Inter_State <- as.factor(data$Inter_State)
data$G <- as.numeric(data$G)
data$Overall_W <- as.numeric(data$Overall_W)
data$Overall_L <- as.numeric(data$Overall_L)
data$SRS <- as.numeric(data$SRS)
data$SOS <- as.numeric(data$SOS)
data$Home_W <- as.numeric(data$Home_W)
data$Home_L <- as.numeric(data$Home_L)
data$Away_W <- as.numeric(data$Away_W)
data$Away_L <- as.numeric(data$Away_L)
data$Tm.Total.Pts <- as.numeric(data$Tm.Total.Pts)
data$Tm.Pts_Allowed <- as.numeric(data$Tm.Pts_Allowed)
data$Pace <- as.numeric(data$Pace)
data$ORtg <- as.numeric(data$ORtg)
data$FTr <- as.numeric(data$FTr)
data$X3PAr<- as.numeric(data$X3PAr)
data$TS.<- as.numeric(data$TS.)
data$TRB.<- as.numeric(data$TRB.)
data$AST.<- as.numeric(data$AST.)
data$STL.<- as.numeric(data$STL.)
data$BLK.<- as.numeric(data$BLK.)
data$eFG.<- as.numeric(data$eFG.)
data$TOV.<- as.numeric(data$TOV.)
data$ORB.<- as.numeric(data$ORB.)
data$FT.FGA<- as.numeric(data$FT.FGA)

## Change the factors to numeric columns
data$G_2 <- as.numeric(data$G_2)
data$Overall_W_2 <- as.numeric(data$Overall_W_2)
data$Overall_L_2 <- as.numeric(data$Overall_L_2)
data$SRS_2 <- as.numeric(data$SRS_2)
data$SOS_2 <- as.numeric(data$SOS_2)
data$Home_W_2 <- as.numeric(data$Home_W_2)
data$Home_L_2 <- as.numeric(data$Home_L_2)
data$Away_W_2 <- as.numeric(data$Away_W_2)
data$Away_L_2 <- as.numeric(data$Away_L_2)
data$Tm.Total.Pts_2 <- as.numeric(data$Tm.Total.Pts_2)
data$Tm.Pts_Allowed_2 <- as.numeric(data$Tm.Pts_Allowed_2)
data$Pace_2 <- as.numeric(data$Pace_2)
data$ORtg_2 <- as.numeric(data$ORtg_2)
data$FTr_2 <- as.numeric(data$FTr_2)
data$X3PAr_2 <- as.numeric(data$X3PAr_2)
```

```
data$TS._2 <- as.numeric(data$TS._2)
data$TRB_2. <- as.numeric(data$TRB._2)
data$AST._2 <- as.numeric(data$AST._2)
data$STL._2 <- as.numeric(data$STL._2)
data$BLK._2 <- as.numeric(data$BLK._2)
data$eFG._2 <- as.numeric(data$eFG._2)
data$TOV._2 <- as.numeric(data$TOV._2)
data$ORB._2<- as.numeric(data$ORB._2)
data$FT.FGA_2<- as.numeric(data$FT.FGA_2)

### Create a few columns by getting the difference in team 1 vs team 2
data$FTr_Mar <- (FTr-FTr_2)
data$Pace_Mar <- (Pace-Pace_2)
data$TOV_Mar <- (TOV.-TOV._2)
data$eFG_Mar <- (eFG.-eFG._2)
data$ORB_Mar <- (ORB.-ORB._2)
data$SOS_Mar <- (SOS-SOS_2)
data$AST_Mar <- (AST.-AST._2)
data$STL_Mar <- (STL.-STL._2)

str(data)

vegasData <- read.csv("VegasDataTotals.csv", header = TRUE)

### Split the data based on Game ID
### Game ID includes 1,764 games played in Feb and Mar
### that we were able to get vegas odds on
train <- vegasData$GameID
dataTrain <- data[-train,]
dataTest  <- data[train,]
rm(train)

### Create new dataframe with desired columns
winResults <- dataTest[c(5:6,10)]

#################################################################################
###
#################################################################################
###
#################################################################################
###

### The following will be used for predicting game totals

### Create the totalformula
totalsformula <- (total~ Pace + Pace_2 + SOS + SOS_2 + eFG. + eFG._2
```

```
              + FTr + FTr_2 + TOV. + TOV._2 +STL. +STL._2 +
                ORtg + ORtg_2 + ORB. + ORB._2 + BLK. + BLK._2 +
                    Neutral_Location + Inter_Conference + Inter_State)

### LinearModel for game totals
lmModelTotals <- lm(totalsformula,
                    data = dataTrain)

### Perform backward model to remove insignifacnt variables
lmModelTotals <- step(lmModelTotals, direction = 'both')

### Check the results
lmModelTotals
summary(lmModelTotals)

### Check variable importance
varIMP <- varImp(lmModelTotals, scale = FALSE)

varIMP[1]

totalvarimpnames <- c("Pace_2","Pace", "ORtg", "ORtg_2", "STL.", "STL._2",
         "BLK._2", "BLK.", "Neutral_Location", "ORB.", "ORB._2", "eFG.", "eFG._2")

overall <-varIMP$Overall

varimpTable <- data.frame(Variable = totalvarimpnames,
                          Overall =overall)

ggplot(data=varimpTable, aes(x=Variable, y=Overall)) +
  geom_bar(stat="identity", fill="steelblue")+
  ggtitle("Variable Importance")+
  theme_fivethirtyeight() + coord_flip()


### Predict on the test set
vegasData$LinearPreds <- predict(lmModelTotals, dataTest)

## Plot the predicted vs actual points
plot(vegasData$Game.Total, vegasData$LinearPreds, main = "Linear Model \nActual vs
Predicted",
     xlab = "Actual", ylab = "Predicted", col = 4)

linearMSE = mean((vegasData$LinearPreds - vegasData$Game.Total)^2)

################################################################################
###
```

```
### Create grids for ridge model
indx <- createFolds(dataTrain$total, returnTrain = TRUE)
ctrl <- trainControl(method = "cv", index = indx)
ridgeGrid <- expand.grid(lambda = seq(0, .1, length = 15))

### Create Ridge Model for game total
ridgeTotalsModel <- train(totalsformula,
                          data = dataTrain,
                          method = "ridge",
                          tuneGrid = ridgeGrid,
                          trControl = ctrl,
                          preProc = c("center", "scale"))
ridgeTotalsModel

vegasData$RidgeModel <- predict(ridgeTotalsModel, dataTest)

plot(vegasData$Game.Total, vegasData$RidgeModel, main = "Ridge Model \nActual vs
Predicted",
    xlab = "Actual", ylab = "Predicted", col = 6)

ridgeMSE = mean((vegasData$RidgeModel - vegasData$Game.Total)^2)

################################################################################
###

### Create train control for elastic net
train_cont <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 5,
                           search = "random",
                           verboseIter = TRUE)

### Create elasticnet model for game totals
totalsElasticNet <- train(totalsformula,
                          data = dataTrain,
                          method = "glmnet",
                          preProcess = c("center", "scale"),
                          tuneLength = 10,
                          trControl = train_cont)

### Check the results and best tunes
totalsElasticNet
totalsElasticNet$bestTune

# Make predictions on test set
```

```
vegasData$Elastic_Net <- predict(totalsElasticNet, dataTest)

## Plot the results
plot(vegasData$Game.Total, vegasData$Elastic_Net, main = "Elastic-Net Model \nActual vs
Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)

enetMSE = mean((vegasData$Elastic_Net - vegasData$Game.Total)^2)

################################################################################
###

### Decision Tree
treeModel = tree(totalsformula,
                 data = dataTrain)

treeModel
summary(treeModel)

crossValTree = cv.tree(treeModel)
# Note: Best size = 6
which.min(crossValTree$size)

treeModel = prune.tree(treeModel, best = 6)
plot(treeModel)
text(treeModel, pretty = 0)

plot(crossValTree$size, crossValTree$dev, type = 'b', main = "Size vs Standard Deviation",
    xlab = "Size", ylab = "Standard Deviation")

# Get predictions on the test data
vegasData$TreePreds = predict(treeModel, dataTest)

# Plot the observed values against the predicted values
plot(vegasData$TreePreds, vegasData$Game.Total, main = "Tree Model \nActual vs Predicted")

# Compute the test error rate
treeMSE = mean((vegasData$TreePreds - vegasData$Game.Total)^2)

## Plot the results
treeTotalPlot <- plot(vegasData$Game.Total, vegasData$TreePreds, main = "Tree Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)

################################################################################
###
```

```
### PLS Model
plsTotalsModel <- train(totalsformula,
                        data = dataTrain,
                        method = "pls",
                        tuneGrid = expand.grid(ncomp = 1:22),
                        trControl = ctrl)
plsTotalsModel

plot(plsTotalsModel)

vegasData$PLSModel <- predict(plsTotalsModel, dataTest)

## Plot the results
plot(vegasData$Game.Total, vegasData$PLSModel, main = "PLS Model \nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)

plsMSE = mean((vegasData$PLSModel - vegasData$Game.Total)^2)

################################################################################
###

pcrTotalsModel <- train(totalsformula,
                        data = dataTrain,
                        method = "pcr",
                        tuneGrid = expand.grid(ncomp = 1:22),
                        trControl = ctrl)
pcrTotalsModel

vegasData$PCRModel <- predict(pcrTotalsModel, dataTest)

## Plot the results
plot(vegasData$Game.Total, vegasData$PCRModel, main = "PCR Model \nActual vs
Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)

pcrMSE = mean((vegasData$PCRModel - vegasData$Game.Total)^2)

################################################################################
###
### Random Forest

forestModel <- rfsrc(totalsformula,
                     data = dataTrain,
                     importance = TRUE,
                     mtry = 23,
```

```
                ntree = 1000)

forestModel

data.frame(importance = forestModel$importance + 5) %>% # add a large +ve constant
      log() %>%
      tibble::rownames_to_column(var = "variable") %>%
      ggplot(aes(x = reorder(variable,importance), y = importance)) +
      geom_bar(stat = "identity", fill = "orange", color = "black", width = 0.5)+
      coord_flip() +
      labs(x = "Variables", y = "Log-transformed variable importance") +
      theme_classic()

#mtry.valuestotal <- seq(15,21,4)
#nodesize.valuestotal <- seq(4,8,2)
#ntree.valuestotal <- seq(4e3,6e3,1e3)

# Create a data frame containing all combinations
#hyper_gridtotal <- expand.grid(mtry = mtry.valuestotal, nodesize = nodesize.valuestotal, ntree
= ntree.valuestotal)

# Create an empty vector to store OOB error values
#oob_errtotal <- c()

# Write a loop over the rows of hyper_grid to train the grid of models
#for (i in 1:nrow(hyper_gridtotal)) {
 #
  #     # Train a Random Forest model
   #     modeltotal <- rfsrc(totalsformula,
    #                data = dataTrain,
     #                mtry = hyper_gridtotal$mtry[i],
      #               nodesize = hyper_gridtotal$nodesize[i],
       #              ntree = hyper_gridtotal$ntree[i])
        #

       # Store OOB error for the model
#       oob_errtotal[i] <- modeltotal$err.rate[length(modeltotal$err.rate)]
#}

# Identify optimal set of hyperparmeters based on OOB error
#opt_i_total <- which.min(oob_errtotal)
#print(hyper_gridtotal[opt_i_total,])

### The optimal solution is mtry = 15 , nodesize = 8, and ntree = 4000
forestModel <- rfsrc(totalsformula,
                     data = dataTrain,
```

```
                    mtry = 15,
                    nodesize = 8,
                    ntree = 4000)

vegasData$ForestModel <- predict.rfsrc(forestModel, dataTest)$predicted

plot(vegasData$Game.Total, vegasData$ForestModel, main = "Forest Model \nActual vs
Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)

rforestMSE = mean((vegasData$ForestModel - vegasData$Game.Total)^2)

################################################################################
###
### SVM Model

#svmTotalModel <- tune.svm(totalsformula,
            #     data = dataTrain,
             #    gamma = seq(.01,.1, by = .02),
              #   cost = seq(.1,1,by =.2))

### Check the best parameter
#svmTotalModel$best.parameters
#svmTotalModel$performances
#svmTotalModel

### Create the model SVM model using the best parameters
svmTotalModel <- svm(totalsformula,
            data = dataTrain,
            gamma = .01,
            cost = .03)
svmTotalModel

### Prediction for SVM Model
vegasData$svmPreds <- predict(svmTotalModel, dataTest)

svmMSE = mean((vegasData$svmPreds - vegasData$Game.Total)^2)

plot(vegasData$Game.Total, vegasData$svmPreds, main = "SVM Model \nActual vs Predicted",
                xlab = "Actual", ylab = "Predicted", col = 4)


################################################################################
###

### Call the functions to get the results of all the models
```

```
accLin = round(accLinear(),3)
accRidge = round(accRidge(),3)
accENET = round(accElasticNet(),3)
accTree = round(accTree(),3)
accPLS = round(accPLSModel(),3)
accPCR = round(accPCRModel(),3)
accForest = round(accForestModel(),3)
accSVM = round(accSVMModel(),3)

paste("The linear model's accuracy is:", accLin, "%")
paste("The ridge model's accuracy is:", accRidge, "%")
paste("The elastic-net model's accuracy is:", accENET, "%")
paste("The decision tree model's accuracy is:", accENET, "%")
paste("The PLS model's accuracy is:", accPLS, "%")
paste("The PCR model's accuracy is:", accPCR, "%")
paste("The Forest model's accuracy is:", accForest, "%")
paste("The SVM model's accuracy is:", accSVM, "%")

totalMSE <- data.frame(LinearMSE = linearMSE,
                       RandomForest = rforestMSE,
                       DecisionTree = treeMSE,
                       SVMMSE = svmMSE,
                       Ridge = ridgeMSE,
                       Enet = enetMSE,
                       PCR = pcrMSE,
                       PLS = plsMSE)
totalMSE

write.csv(vegasData, "FinalTotalResults.csv")

##############################################################################
###
### Comparision of Models vs Actual Score

par(mfrow = c(3,3))

plot(vegasData$Game.Total, vegasData$LinearPreds, main = "Linear Model \nActual vs
Predicted",
                    xlab = "Actual", ylab = "Predicted", col = 4)
abline(0,1)

plot(vegasData$Game.Total, vegasData$RidgeModel, main = "Ridge Model \nActual vs
Predicted",
                   xlab = "Actual", ylab = "Predicted", col = 5)
abline(0,1)
```

```
plot(vegasData$Game.Total, vegasData$Elastic_Net, main = "Elastic-Net Model \nActual vs
Predicted",
                 xlab = "Actual", ylab = "Predicted", col = 6)
abline(0,1)

plot(vegasData$Game.Total, vegasData$TreePreds, main = "Tree Model \nActual vs Predicted",
                 xlab = "Actual", ylab = "Predicted", col = "orange")
abline(0,1)

plot(vegasData$Game.Total, vegasData$PLSModel, main = "PLS Model \nActual vs Predicted",
                xlab = "Actual", ylab = "Predicted", col = 8)
abline(0,1)

plot(vegasData$Game.Total, vegasData$PCRModel, main = "PCR Model \nActual vs
Predicted",
                xlab = "Actual", ylab = "Predicted", col = 3)
abline(0,1)

plot(vegasData$Game.Total, vegasData$ForestModel, main = "Forest Model \nActual vs
Predicted",
                  xlab = "Actual", ylab = "Predicted", col = "#009999")
abline(0,1)

plot(vegasData$Game.Total, vegasData$svmPreds, main = "SVM Model \nActual vs Predicted",
                xlab = "Actual", ylab = "Predicted", col = 2)
abline(0,1)
dev.off()

##############################################################################
###
### Comparision of Models vs Vegas Projections

par(mfrow = c(3,3))

plot(vegasData$Vegas_Over_Under, vegasData$LinearPreds, main = "Linear Model \nActual vs
Predicted",
                  xlab = "Vegas Over/Under", ylab = "Predicted", col = 4)
abline(0,1)

plot(vegasData$Vegas_Over_Under, vegasData$RidgeModel, main = "Ridge Model \nActual vs
Predicted",
                 xlab = "Vegas Over/Under", ylab = "Predicted", col = 5)
abline(0,1)
```

```
plot(vegasData$Vegas_Over_Under, vegasData$Elastic_Net, main = "Elastic-Net Model
\nActual vs Predicted",
                xlab = "Vegas Over/Under", ylab = "Predicted", col = 6)
abline(0,1)

plot(vegasData$Vegas_Over_Under, vegasData$TreePreds, main = "Tree Model \nActual vs
Predicted",
                xlab = "Vegas Over/Under", ylab = "Predicted", col = "orange")
abline(0,1)

plot(vegasData$Vegas_Over_Under, vegasData$PLSModel, main = "PLS Model \nActual vs
Predicted",
               xlab = "Vegas Over/Under", ylab = "Predicted", col = 8)
abline(0,1)

plot(vegasData$Vegas_Over_Under, vegasData$PCRModel, main = "PCR Model \nActual vs
Predicted",
               xlab = "Vegas Over/Under", ylab = "Predicted", col = 3)
abline(0,1)

plot(vegasData$Vegas_Over_Under, vegasData$ForestModel, main = "Forest Model \nActual vs
Predicted",
                  xlab = "Vegas Over/Under", ylab = "Predicted", col = "#009999")
abline(0,1)

plot(vegasData$Vegas_Over_Under, vegasData$svmPreds, main = "SVM Model \nActual vs
Predicted",
               xlab = "Vegas Over/Under", ylab = "Predicted", col = 2)
abline(0,1)
dev.off()

##############################################################################
###
##############################################################################
###
##############################################################################
###

spreadVegasData <- read.csv("Spreads.csv", header = TRUE)
spreadVegasData <- spreadVegasData[c(2:5)]

### Create forumula for point spread
spreadFormula <- (WinMargin~ Pace + Pace_2 + SOS + SOS_2 + eFG. + eFG._2
            + FTr + FTr_2 + TOV. + TOV._2 +STL. +STL._2 +
                ORtg + ORtg_2 + ORB. + ORB._2 + BLK. + BLK._2 + + Neutral_Location +
                Inter_Conference + Inter_State)
```

```
###old formula
#SOS_Mar + FTr_Mar + Pace_Mar
#+ eFG_Mar + TOV_Mar + ORB_Mar + Neutral_Location +

### Linear Model for Point Spread
lmSpreadModel <- lm(spreadFormula,
            data = dataTrain)

### Perform backward model to remove insignifacnt variables
lmSpreadModel <- stepAIC(lmSpreadModel, direction = 'both')

### Check the results
lmSpreadModel
summary(lmSpreadModel)

### Check variable importance
varIMP <- varImp(lmSpreadModel, scale = FALSE)
varIMP

spreadOverall <-varIMP$Overall

totalvarimpnamesSpread <- c("Pace","Pace_2", "SOS", "SOS_2", "eFG.", "eFG._2",
                  "FTr", "FTr_2", "TOV.", "TOV._2", "STL.", "STL._2", "ORtg",
                  "ORtg_2","ORB.","ORB._2","BLK.","BLK._2")

varimpTableSpread <- data.frame(Variable = totalvarimpnamesSpread,
                      Overall =spreadOverall)

ggplot(data=varimpTableSpread, aes(x=Variable, y=Overall)) +
  geom_bar(stat="identity", fill="steelblue")+
  ggtitle("Variable Importance")+
  theme_fivethirtyeight() + coord_flip()

### Predict the data on the test set
spreadVegasData$LinearSpreads <- predict(lmSpreadModel, dataTest)

## Plot the predicted vs actual points
plot(spreadVegasData$WinMargin, spreadVegasData$LinearSpreads, main = "Linear Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)

lmSpreadMSE = mean((spreadVegasData$LinearSpreads - spreadVegasData$WinMargin)^2)

#############################################################################
###
```

```
### Create grids for ridge model
indx <- createFolds(dataTrain$total, returnTrain = TRUE)
ctrl <- trainControl(method = "cv", index = indx)
ridgeGrid <- expand.grid(lambda = seq(0, .1, length = 15))

### Create Ridge Model for game total
ridgeSpreadModel <- train(spreadFormula,
                data = dataTrain,
                method = "ridge",
                tuneGrid = ridgeGrid,
                trControl = ctrl,
                preProc = c("center", "scale"))
ridgeSpreadModel

spreadVegasData$RidgeModel <- predict(ridgeSpreadModel, dataTest)

ridgeSpreadMSE = mean((spreadVegasData$RidgeModel - spreadVegasData$WinMargin)^2)


################################################################################
###

### Create train control for elastic net
train_cont <- trainControl(method = "repeatedcv",
                        number = 10,
                        repeats = 5,
                        search = "random",
                        verboseIter = TRUE)

### Create elasticnet model for game totals
spreadsElasticNet <- train(spreadFormula,
                data = dataTrain,
                method = "glmnet",
                preProcess = c("center", "scale"),
                tuneLength = 10,
                trControl = train_cont)

### Check the results and best tunes
spreadsElasticNet
spreadsElasticNet$bestTune

# Make predictions on test set
spreadVegasData$Elastic_Net <- predict(spreadsElasticNet, dataTest)

enetSpreadMSE = mean((spreadVegasData$Elastic_Net - spreadVegasData$WinMargin)^2)
```

```
#############################################################################
###
### Create a pls model
plsSpreadModel <- train(spreadFormula,
                data = dataTrain,
                method = "pls",
                tuneGrid = expand.grid(ncomp = 1:22),
                trControl = ctrl)
plsSpreadModel
### Plot
plot(plsSpreadModel)

spreadVegasData$PLSModel <- predict(plsSpreadModel, dataTest)

plsSpreadMSE = mean((spreadVegasData$PLSModel - spreadVegasData$WinMargin)^2)

#############################################################################
###
### Create a PCR Model
pcrSpreadModel <- train(spreadFormula,
                data = dataTrain,
                method = "pcr",
                tuneGrid = expand.grid(ncomp = 1:22),
                trControl = ctrl)

pcrSpreadModel

spreadVegasData$PCRModel <- predict(pcrSpreadModel, dataTest)

pcrSpreadMSE = mean((spreadVegasData$PCRModel - spreadVegasData$WinMargin)^2)

#############################################################################
###

### Decision Tree
treeSpreadModel = tree(spreadFormula,
                data = dataTrain)

treeSpreadModel
summary(treeSpreadModel)

crossValTreeSpread = cv.tree(treeSpreadModel)
# Note: Best size = 7
low = which.min(crossValTreeSpread$size)
```

```
treeSpreadModel = prune.tree(treeSpreadModel, best = low)
plot(treeSpreadModel)
text(treeSpreadModel, pretty = 0)

#plot(crossValTree$size, crossValTree$dev, type = 'b', main = "Size vs Standard Deviation",
 #    xlab = "Size", ylab = "Standard Deviation")

# Get predictions on the test data

spreadVegasData$TreePreds = predict(treeSpreadModel, dataTest)

# Plot the observed values against the predicted values
plot(spreadVegasData$TreePreds, spreadVegasData$WinMargin)

# Compute the test error rate
treeSpreadMSE = mean((spreadVegasData$TreePreds - spreadVegasData$WinMargin)^2)

## Plot the results
plot(spreadVegasData$WinMargin, spreadVegasData$TreePreds, main = "Tree Model \nActual
vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)

################################################################################
###
### Random Forest
forestSpreadModel <- rfsrc(spreadFormula,
                  data = dataTrain,
                  importance = TRUE,
                  ntree = 1000)

forestSpreadModel

data.frame(importance = forestSpreadModel$importance + 5) %>% # add a large +ve constant
       log() %>%
       tibble::rownames_to_column(var = "variable") %>%
       ggplot(aes(x = reorder(variable,importance), y = importance)) +
       geom_bar(stat = "identity", fill = "orange", color = "black", width = 0.5)+
       coord_flip() +
       labs(x = "Variables", y = "Log-transformed variable importance") +
       theme_classic()

#mtry.values <- seq(15,21,4)
#nodesize.values <- seq(4,8,2)
#ntree.values <- seq(4e3,6e3,1e3)
```

```
# Create a data frame containing all combinations
#hyper_grid <- expand.grid(mtry = mtry.values, nodesize = nodesize.values, ntree =
ntree.values)

# Create an empty vector to store OOB error values
#oob_err <- c()

# Write a loop over the rows of hyper_grid to train the grid of models
#for (i in 1:nrow(hyper_grid)) {

       # Train a Random Forest model
 #       model <- rfsrc(spreadFormula,
  #                     data = dataTrain,
   #                    mtry = hyper_grid$mtry[i],
    #                   nodesize = hyper_grid$nodesize[i],
     #                  ntree = hyper_grid$ntree[i])
      #

       # Store OOB error for the model
      # oob_err[i] <- model$err.rate[length(model$err.rate)]
#}

# Identify optimal set of hyperparmeters based on OOB error
#opt_i <- which.min(oob_err)
#print(hyper_grid[opt_i,])


### The optimal solution is mtry = 15, nodesize = 8, and ntree 5000

forestSpreadModel <- rfsrc(spreadFormula,
                           data = dataTrain,
                           mtry = 15,
                           nodesize = 8,
                           ntree = 5000)

forestSpreadModel

spreadVegasData$ForestModel <- predict.rfsrc(forestSpreadModel, dataTest)$predicted

plot(spreadVegasData$WinMargin, spreadVegasData$ForestModel, main = "Forest Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)

rforestSpreadMSE = mean((spreadVegasData$ForestModel - spreadVegasData$WinMargin)^2)
```

```
#############################################################################
###

#svmSpreadModel <- tune.svm(spreadFormula,
            #          data = dataTrain,
             #         gamma = seq(.01,.1, by = .02),
              #        cost = seq(.1,1,by =.2))

### Check the best parameter
#svmSpreadModel$best.parameters
#svmSpreadModel$performances
#svmSpreadModel

### Optimal cost = .9, gamma = .01, epsilon = .1

### Create the model SVM model using the best parameters
svmSpreadModel <- svm(spreadFormula,
                  data = dataTrain,
                  gamma = .01,
                  cost = .9)
svmSpreadModel

### Prediction for SVM Model
spreadVegasData$svmPreds <- predict(svmSpreadModel, dataTest)

svmSpreadMSE = mean((spreadVegasData$svmPreds - spreadVegasData$WinMargin)^2)

#############################################################################
###
### Comparision of Models

accSpreadLinear = round(accSpreadLinear(),3)
accSpreadRidge = round(accSpreadRidge(),3)
accSpreadENET = round(accSpreadElasticNet(),3)
accSpreadTree = round(accSpreadTree(),3)
accSpreadPLS = round(accSpreadPLSModel(),3)
accSpreadPCR = round(accSpreadPCRModel(),3)
accSpreadForest = round(accSpreadForestModel(),3)
accSpreadSVM = round(accSpreadSVMModel(),3)

paste("The linear model's accuracy is:", accSpreadLinear, "%")
paste("The ridge model's accuracy is:", accSpreadRidge, "%")
paste("The elastic-net model's accuracy is:", accSpreadENET, "%")
paste("The decision tree model's accuracy is:", accSpreadTree, "%")
paste("The PLS model's accuracy is:", accSpreadPLS, "%")
paste("The PCR model's accuracy is:", accSpreadPCR, "%")
```

```
paste("The Forest model's accuracy is:", accSpreadForest, "%")
paste("The SVM model's accuracy is:", accSpreadSVM, "%")

spreadMSE <- data.frame(LinearMSE = lmSpreadMSE,
                        RandomForest = rforestSpreadMSE,
                        DecisionTree = treeSpreadMSE,
                        SVMMSE = svmSpreadMSE,
                        Ridge = ridgeSpreadMSE,
                        Enet = enetSpreadMSE,
                        PCR = pcrSpreadMSE,
                        PLS = plsSpreadMSE)

write.csv(spreadVegasData, "FinalSpreadResults.csv")

##############################################################################
###
### Comparision of Models vs Actual Score

par(mfrow = c(3,3))

plot(spreadVegasData$WinMargin, spreadVegasData$LinearSpreads, main = "Linear Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)
abline(0,1)

plot(spreadVegasData$WinMargin, spreadVegasData$RidgeModel, main = "Ridge Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 5)
abline(0,1)

plot(spreadVegasData$WinMargin, spreadVegasData$Elastic_Net, main = "Elastic-Net Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 6)
abline(0,1)

plot(spreadVegasData$WinMargin, spreadVegasData$TreePreds, main = "Tree Model \nActual
vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = "orange")
abline(0,1)

plot(spreadVegasData$WinMargin, spreadVegasData$PLSModel, main = "PLS Model \nActual
vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 8)
abline(0,1)
```

```
plot(spreadVegasData$WinMargin, spreadVegasData$PCRModel, main = "PCR Model \nActual
vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 3)
abline(0,1)

plot(spreadVegasData$WinMargin, spreadVegasData$ForestModel, main = "Forest Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 1)
abline(0,1)

plot(spreadVegasData$WinMargin, spreadVegasData$svmPreds, main = "SVM Model \nActual
vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 2)
abline(0,1)
dev.off()

##############################################################################
###
### Comparision of Models vs Vegas Projections

par(mfrow = c(3,3))

plot(spreadVegasData$Actual.Spread, spreadVegasData$LinearSpreads, main = "Linear Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 4)
abline(0,1)

plot(spreadVegasData$Actual.Spread, spreadVegasData$RidgeModel, main = "Ridge Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 5)
abline(0,1)

plot(spreadVegasData$Actual.Spread, spreadVegasData$Elastic_Net, main = "Elastic-Net
Model \nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 6)
abline(0,1)

plot(spreadVegasData$Actual.Spread, spreadVegasData$TreePreds, main = "Tree Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = "orange")
abline(0,1)

plot(spreadVegasData$Actual.Spread, spreadVegasData$PLSModel, main = "PLS Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 8)
abline(0,1)
```

```
plot(spreadVegasData$Actual.Spread, spreadVegasData$PCRModel, main = "PCR Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 3)
abline(0,1)

plot(spreadVegasData$Actual.Spread, spreadVegasData$ForestModel, main = "Forest Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = "#009999")
abline(0,1)

plot(spreadVegasData$Actual.Spread, spreadVegasData$svmPreds, main = "SVM Model
\nActual vs Predicted",
    xlab = "Actual", ylab = "Predicted", col = 2)
abline(0,1)
dev.off()

################################################################################
###
################################################################################
###
################################################################################
###

### The following will be used for predicting who wins the game

### Create a logistic regression model
glm <- glm(Win~ SOS_Mar+FTr_Mar+Pace_Mar+eFG_Mar+TOV_Mar+ORB_Mar,
           data = dataTrain,
           family = "binomial")

### Get the summary of the model
summary(glm)
glm

### Generate probabilities
winResults$GLMProb = predict.glm(glm, dataTest, type = "response")

### Generate win loss based on criteria
winResults$GLMPred = ifelse(winResults$GLMProb >= .5,1,0)

### Create a confusion matric
confusionMatrix(as.factor(winResults$GLMPred), as.factor(winResults$Win))

varIMPGLM <- varImp(glm, scale = FALSE)
```

```
glmvar = varIMPGLM

totalvarimpnamesGLM<- c("SOS_Margin","FTr_Margin", "Pace_Margin", "eFG_Margin",
"TOV_Margin", "ORB Margin")

varimpTableGLM <- data.frame(Variable = totalvarimpnamesGLM,
                             Overall = varIMPGLM)

ggplot(data=varimpTableGLM, aes(x=Variable, y=Overall)) +
  geom_bar(stat="identity", fill="steelblue")+
  ggtitle("Variable Importance")+
  theme_fivethirtyeight() + coord_flip()


### Get the roc and plot it
glmROC <- roc(dataTest$Win, winResults$GLMProb)
auc(glmROC)
plot(glmROC, main = "GLM Model")
legend("bottomright", c("AUC = 0.7269 "))

### create empty columns
winResults$PredictedMoneyLine <- NA

### Predict Moneyline odds
for(i in 1:nrow(winResults)){
      if(winResults[i,4] >= .5){
            winResults[i,4] = winResults[i,4] * 100
            winResults[i,6]=  -(winResults[i,4])/(100 - winResults[i,4]) * 100
      }else {
            winResults[i,4] = winResults[i,4] * 100
            winResults[i,6]= ((100 - winResults[i,4])/(winResults[i,4]) * 100)
      }
}

##############################################################################
###

### SVM Model
### Tune the model to get the best parameters
#svmModel <- tune.svm(Win~
SOS_Mar+FTr_Mar+Pace_Mar+eFG_Mar+TOV_Mar+ORB_Mar,
   #             data = dataTrain,
    #            gamma = seq(.01,.1, by = .02),
     #           cost = seq(.1,1,by =.2))

### Check the best parameter
```

```
#svmModel$best.parameters
#svmModel$performances
#svmModel

### Create the model SVM model using the best parameters
svmModel <- svm(Win~ SOS_Mar+FTr_Mar+Pace_Mar+eFG_Mar+TOV_Mar+ORB_Mar,
            data = dataTrain,
            gamma = .09,
            cost = .9,
            probability = TRUE)

winResults$svmPredict <- predict(svmModel, dataTest, type = "response")

confusionMatrix(winResults$svmPredict, winResults$Win)

##############################################################################
###

### Create a Random Forest Model
forestGameModel <- rfsrc(Win~
SOS_Mar+FTr_Mar+Pace_Mar+eFG_Mar+TOV_Mar+ORB_Mar,
                     data = dataTrain,
                     importance = TRUE,
                     ntree = 1000)

forestGameModel

### Make the predictions
forestGameModelPreds <- predict(forestGameModel, dataTest)

winResults$RForestPreds <- forestGameModelPreds$class

### Create a confusion matric
confusionMatrix(forestGameModelPreds$class, as.factor(winResults$Win))
```