Pstat 131 Final Project

Daniel Pry, Nick Van Daelen, Rick Zheng

3/15/2022

Predicting if NCAA Basketball players will be drafted to the NBA based on their college stats.

Table of Contents

2. Loading Libraries
3. Introduction
4. Loading Data
5. Data Cleaning
6. Exploratory Data Analysis
7. Test/Training Split
8. Model Fitting
9. Model Performance Selection
10. Conclusion

Loading Libraries

```
knitr::opts_chunk$set(echo = TRUE, warning = FALSE)
library(tidyverse)
library(ISLR)
library(glmnet)
library(tree)
library(maptree)
library(randomForest)
library(gbm)
library(ROCR)
library(ggplot2)
library(ROSE)
```

```
## Warning: package 'ROSE' was built under R version 4.1.3
```

```
library(class)
library(reshape2)
library(psych)
```

```
## Warning: package 'psych' was built under R version 4.1.3
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.1.3
```

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.1.3
```

```
library(MASS)
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.1.3
```

```
library(skimr)
```

```
## Warning: package 'skimr' was built under R version 4.1.3
```

Introduction

Every year the NBA holds their draft, where teams will pick to sign new players on to their team. Per the NBA rules for the draft there are not many rules against who can't be drafted other than the player must be at least one year removed from high school. Naturally this causes NCAA basketball teams to be the biggest pool of talent for the NBA to draft athletes from. Historically about ninety percent of players drafted come from Division one programs in the NCAA. In this project we will be attempting to use players' statistics from their last season in the NCAA from the years 2009 to 2020 to predict if a player will be drafted. The data set includes about 20000 players where only about 500 end up getting drafted.

Loading Data

Loading Data Set: source: https://www.kaggle.com/adityak2003/college-basketball-players-20092021

```r
college_stats <- read.csv('CollegeBasketballPlayers2009-2021.csv',
                          na.strings = c("", "N/A", "None", "-"))
```

Data Cleaning

In this chunk we add a column drafted that denotes a drafted player with 1 and undrafted player as 0.

```
stats <- college_stats %>% mutate(drafted = ifelse(is.na(pick), 0, 1))
```

WE name some of the predictors in this chunk.

```
s <- stats %>% group_by(pid) %>%
  summarise(player_name = player_name,
            team = team, conf = conf, seasons = length(pid),
            Min_per  = Min_per, Ortg  = Ortg, usg  = usg, eFG  = eFG, TS_per  = TS_per,
            ORB_per  = ORB_per, DRB_per  = DRB_per, TO_per  = TO_per, AST_per  = AST_per,
            FTM  = FTM, FTA  = FTA, FT_per = FT_per, twoPM = twoPM, twoPA = twoPA,
            twoP_per = twoP_per, TPM = TPM, TPA = TPA, TP_per = TP_per, blk_per = blk_per,
            stl_per = stl_per, college_year = yr, ht = ht, adjoe  = adjoe, year = year,
            pid = pid, Rec.Rank = Rec.Rank, ast.tov = ast.tov, rimmade = rimmade,
            rim_att = rimmade.rimmiss, rim_per = rimmade..rimmade.rimmiss.,
            midmade = midmade, mid_att = midmade.midmiss,
            mid_per = midmade..midmade.midmiss., dunksmade = dunksmade,
            dunks_att = dunksmiss.dunksmade, dunk_per = dunksmade..dunksmade.dunksmiss.,
            drtg = drtg, adrtg = adrtg, stops = stops, bpm = bpm, min_played = mp,
            off_reb = oreb, def_reb = dreb, total_reb = treb, ast = ast, stl = stl,
            blk = blk, pts = pts, Position = X, drafted = drafted)
```

```
## 'summarise()' has grouped output by 'pid'. You can override using the '.groups' argument.
```

```
s_cleaned <- s %>% slice_max(year)
head(s_cleaned)
```

```
## # A tibble: 6 x 54
## # Groups:   pid [6]
##     pid player_name team  conf  seasons Min_per  Ortg   usg   eFG TS_per ORB_per
##   <int> <chr>       <chr> <chr>   <int>   <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl>
## 1     2 DeAndrae R~ Troy  SB          2    11.5  67.1  16.9  34.5   36.1     3.1
## 2     3 Pooh Willi~ Utah~ WAC         3    64.5 106.   18.7  48.1   52.4     0.8
## 3     5 Jesus Verd~ Sout~ BE          1    72    96.2  21.8  45.7   48.0     2.1
## 4     8 Mike Hornb~ Pepp~ WCC         1    44.5  97.7  16    53.6   53.7     4.1
## 5     9 Anthony Br~ Paci~ BW          1    56.2  96.5  22    52.8   54.3     8.3
## 6    11 Nick Rodge~ Butl~ Horz        2     1   121.   16.8  75     75       0
## # ... with 43 more variables: DRB_per <dbl>, TO_per <dbl>, AST_per <dbl>,
## #   FTM <int>, FTA <int>, FT_per <dbl>, twoPM <int>, twoPA <int>,
## #   twoP_per <dbl>, TPM <int>, TPA <int>, TP_per <dbl>, blk_per <dbl>,
## #   stl_per <dbl>, college_year <chr>, ht <chr>, adjoe <dbl>, year <int>,
## #   Rec.Rank <dbl>, ast.tov <dbl>, rimmade <int>, rim_att <int>, rim_per <dbl>,
## #   midmade <int>, mid_att <int>, mid_per <dbl>, dunksmade <int>,
## #   dunks_att <int>, dunk_per <dbl>, drtg <dbl>, adrtg <dbl>, stops <dbl>, ...
```

Cleaning abnormal values in college_year and ht column.

```r
s_cleaned[, 27][s_cleaned[, 27] == 0] <- NA
s_cleaned[, 27][s_cleaned[, 27] == "So"] <- NA
s_cleaned[, 27][s_cleaned[, 27] == "Jr"] <- NA
s_cleaned[, 26][s_cleaned[, 26] == 0] <- NA
```

In the next two columns we clean the ht column to read heights in inches. Since right now they are denoted as a month corresponding to feet followed by a dash denoting inches. For example May denotes 5 feet, June denotes 6 feet, and July denotes 7 feet. We also set all the NA values in this column to the column average of 76.27 since many of the models are unable to use NA values and this method makes more sense than setting them to zeros.

```r
#changing ht
s_cleaned %>% dplyr::select(ht)
```

```
## Adding missing grouping variables: ‘pid‘

## # A tibble: 25,745 x 2
## # Groups:   pid [25,745]
##      pid ht
##    <int> <chr>
##  1     2 2-Jun
##  2     3 4-Jun
##  3     5 4-Jun
##  4     8 4-Jun
##  5     9 8-Jun
##  6    11 2-Jun
##  7    13 5-Jun
##  8    15 Jun-00
##  9    18 6-Jun
## 10    20 5-Jun
## # ... with 25,735 more rows
```

```r
begin_height <- data.frame(do.call("rbind", strsplit(as.character(s_cleaned$ht), "-", fixed = TRUE)))
```

```r
begin_height["X1"][begin_height["X1"] == "Jun"] <- 72
begin_height["X1"][begin_height["X1"] == "Jul"] <- 84
begin_height["X1"][begin_height["X1"] == "May"] <- 60
begin_height["X2"][begin_height["X2"] == "Jun"] <- 72
begin_height["X2"][begin_height["X2"] == "Jul"] <- 84
begin_height["X2"][begin_height["X2"] == "May"] <- 60
begin_height$X1 <- as.numeric(as.character(begin_height$X1))
begin_height$X2 <- as.numeric(as.character(begin_height$X2))
begin_height$height <- begin_height$X1 + begin_height$X2
df_new <- cbind(s_cleaned, begin_height)
df_new %>% dplyr::select(-c('X1', 'X2', 'ht'))
```

```
## # A tibble: 25,745 x 54
## # Groups:   pid [25,745]
##      pid player_name     team     conf  seasons Min_per  Ortg   usg   eFG TS_per
##    <int> <chr>           <chr>    <chr>   <int>   <dbl> <dbl> <dbl> <dbl>  <dbl>
##  1     2 DeAndrae Ross   Troy     SB          2    11.5  67.1  16.9  34.5   36.1
##  2     3 Pooh Williams   Utah St. WAC         3    64.5 106.   18.7  48.1   52.4
##  3     5 Jesus Verdejo   South F~ BE          1    72    96.2  21.8  45.7   48.0
##  4     8 Mike Hornbuckle Pepperd~ WCC         1    44.5  97.7  16    53.6   53.7
##  5     9 Anthony Brown   Pacific  BW          1    56.2  96.5  22    52.8   54.3
##  6    11 Nick Rodgers    Butler   Horz        2     1   121.   16.8  75     75
##  7    13 Dana Smith      Longwood ind         2    81.8  99.8  27.7  49.9   53.8
##  8    15 Matt Beck       Fordham  A10         2     1.3   0     0     0      0
##  9    18 Justin Drummond Wagner   NEC         1    82.8  99.7  20.5  48.8   53.1
## 10    20 Jamal Smith     Wagner   NEC         1    80.4  92.5  23    43.5   45.3
## # ... with 25,735 more rows, and 44 more variables: ORB_per <dbl>,
## #   DRB_per <dbl>, TO_per <dbl>, AST_per <dbl>, FTM <int>, FTA <int>,
## #   FT_per <dbl>, twoPM <int>, twoPA <int>, twoP_per <dbl>, TPM <int>,
## #   TPA <int>, TP_per <dbl>, blk_per <dbl>, stl_per <dbl>, college_year <chr>,
## #   adjoe <dbl>, year <int>, Rec.Rank <dbl>, ast.tov <dbl>, rimmade <int>,
## #   rim_att <int>, rim_per <dbl>, midmade <int>, mid_att <int>, mid_per <dbl>,
## #   dunksmade <int>, dunks_att <int>, dunk_per <dbl>, drtg <dbl>, ...
```

```
df_new$height <- df_new$height %>% as.numeric() %>% replace_na(76.27)
df_new %>% dplyr::select(height)
```

```
## Adding missing grouping variables: ‘pid‘
```

```
## # A tibble: 25,745 x 2
## # Groups:   pid [25,745]
##      pid height
##    <int>  <dbl>
##  1     2     74
##  2     3     76
##  3     5     76
##  4     8     76
##  5     9     80
##  6    11     74
##  7    13     77
##  8    15     72
##  9    18     78
## 10    20     77
## # ... with 25,735 more rows
```

```
s_cleaned <- df_new
```

Lastly, to finish cleaning our data we must get rid of players in the data who have not yet had the chance to either be drafted or to not be drafted. These players who are still playing in college will not help our model. A very good active college player would, as of now, have a zero in the "drafted" column. This could potentially weaken our model as they may go on to be drafted in the future. THerefore, we remove players whose "year" is 2021.

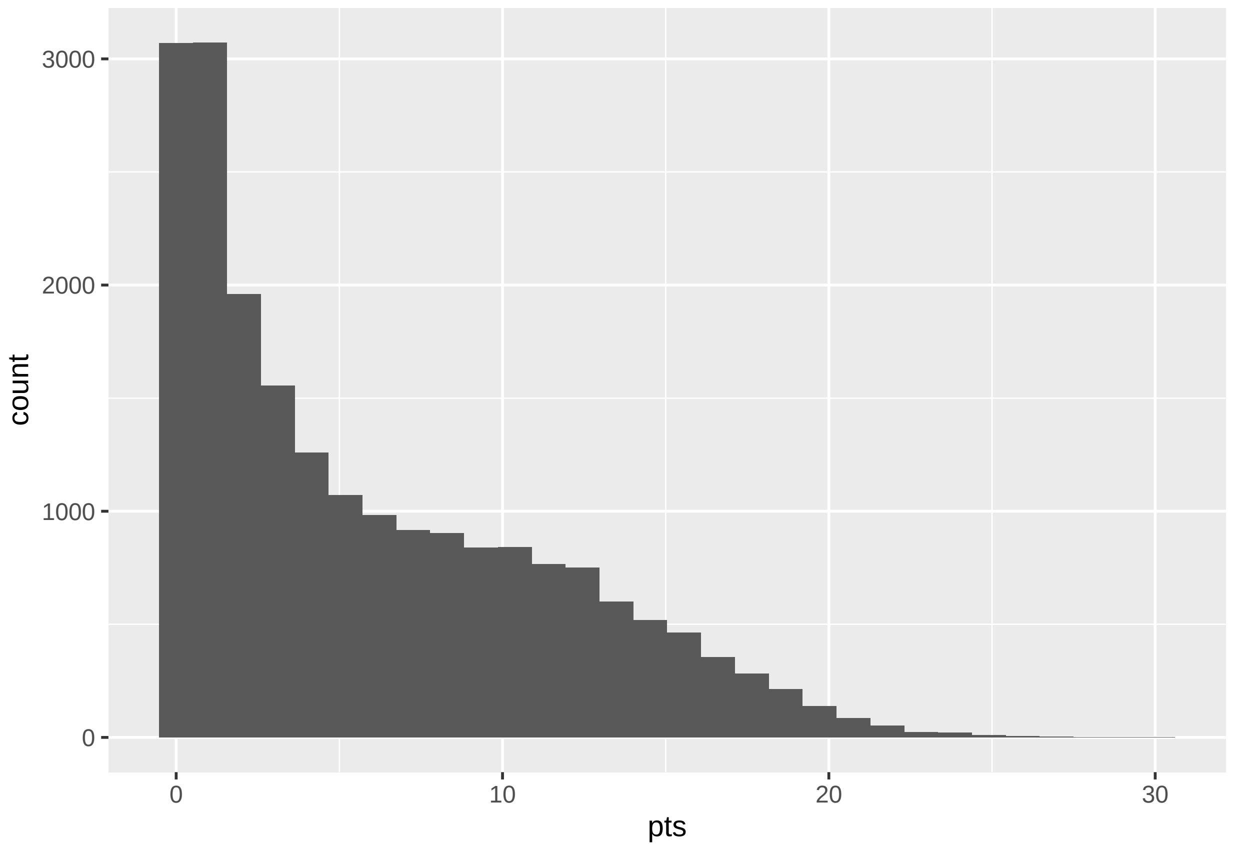
```
s_cleaned <- subset(s_cleaned, year != 2021)
```

Exploratory Data Analysis

This histogram shows the average points scored per game on the x-axis and the frequency number of players in the data set that score average that number of points.

```r
ggplot(s_cleaned, aes(x = pts)) +
  geom_histogram()
```

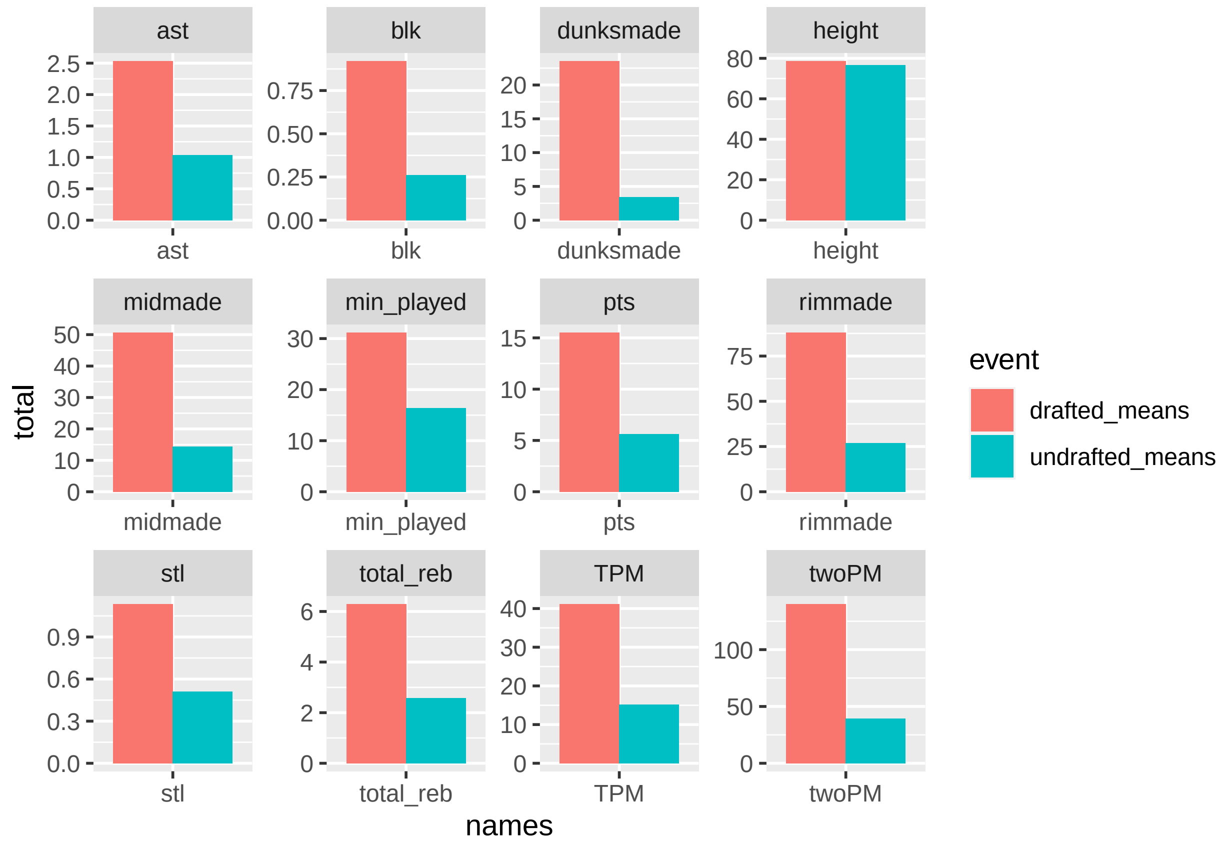
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



In the following plot we compare the averages of certain predictors for drafted vs non-drafted players. From these it is possible to make potential assumptions on what predictors might be significant for predicting if a player is drafted. For example the difference in average height of drafted and non-drafted is not very different so this might imply that height will not be a very significant predictor in many of the models. Many of the other predictors do have very different averages for drafted and non-drafted players so it is very likely some of these other predictors will be signifcant in predicting if a player is drafted.

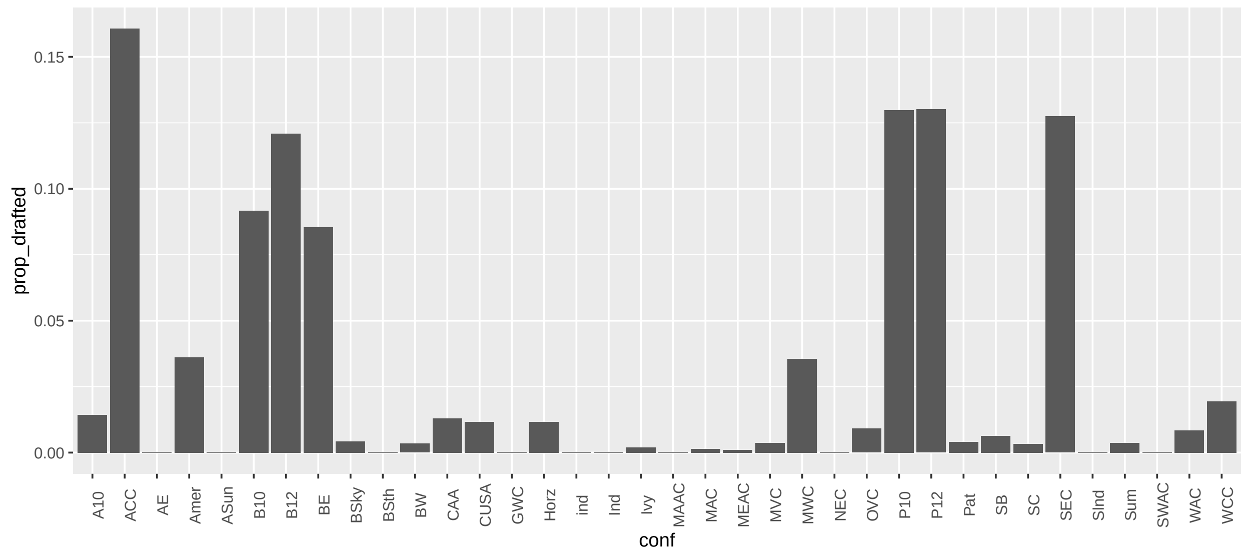
```r
# group by drafted or not
drafted_means <- s_cleaned %>% subset(drafted == 1) %>% sapply(mean, na.rm=TRUE)
drafted_means <- as.data.frame(drafted_means)
undrafted_means <- s_cleaned %>% subset(drafted == 0) %>% sapply(mean, na.rm=TRUE)
undrafted_means <- as.data.frame(undrafted_means)
# taking averages of drafted and undrafted players
drafted_vs_undrafted <- undrafted_means %>% mutate(drafted_means)
# transforming data frame for plotting
```

```
drafted_vs_undrafted$names <- rownames(drafted_vs_undrafted)
rownames(drafted_vs_undrafted) <- NULL
drafted_vs_undrafted <- gather(drafted_vs_undrafted, event, total, drafted_means:undrafted_means)
d_vs_u <- drafted_vs_undrafted %>% filter(names == "pts" | names == "dunksmade" |
                                            names == "min_played" | names == "ast" |
                                            names == "TPM" | names == "midmade"|
                                            names == "blk" | names == "total_reb" |
                                            names == "twoPM" | names == "rimmade" |
                                            names == "stl" | names == "height")
# plotting of means
avg_bars <- d_vs_u %>% ggplot(aes(names, total, fill=event)) +
  geom_bar(stat = "identity", position = 'dodge') + facet_wrap(~ names, scales = "free")
avg_bars
```



Making a barplot of the percentage of players from each conference that are drafted. We notice that there are a handful of conferences that have a drastically higher percentage of players drafted. Additionally, there are a few conferences where no players have been drafted.

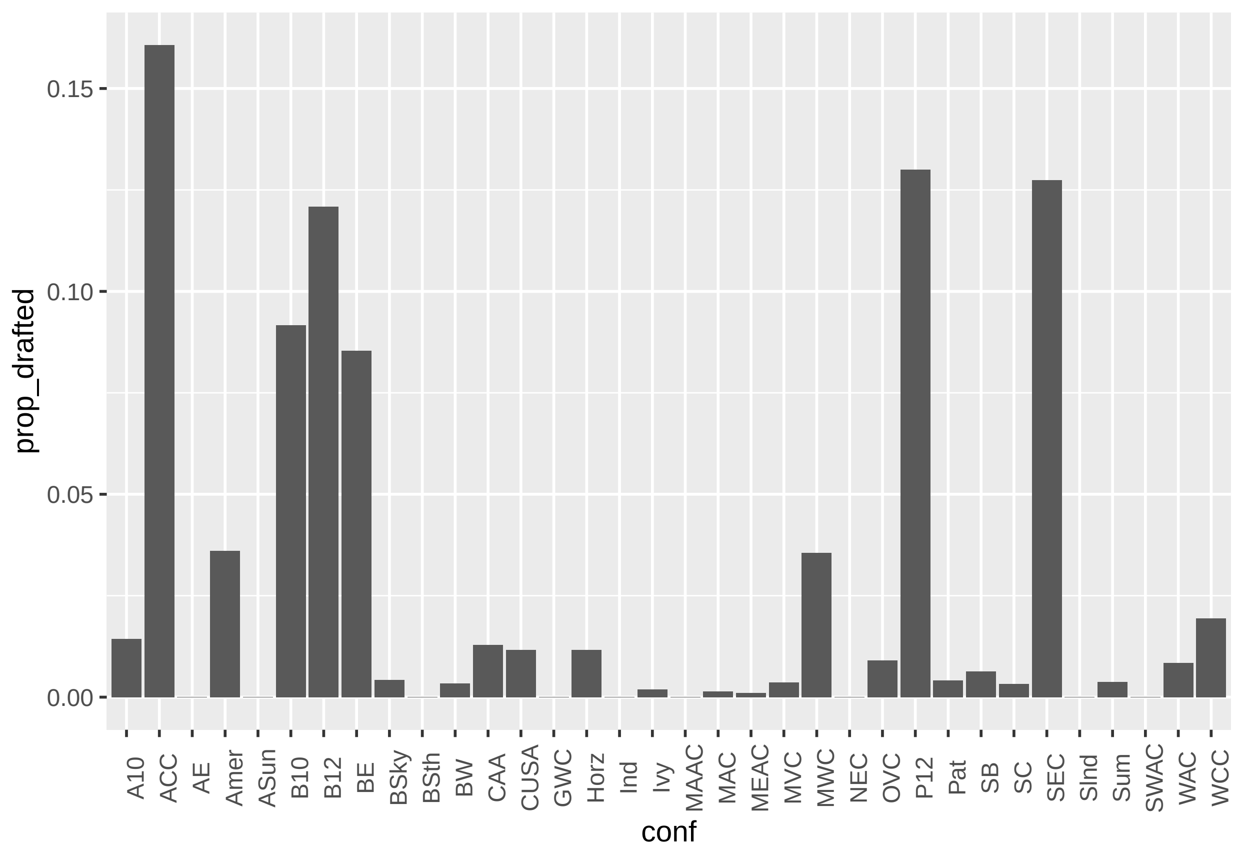
```
draft_by_conf <-  s_cleaned %>% group_by(conf) %>% summarise(prop_drafted = mean(drafted))
ggplot(draft_by_conf) + geom_bar(aes(x = conf, y = prop_drafted), stat = "identity") +
  theme(axis.text.x = element_text(angle = 90))
```



This plot exposes a few problems with the naming of the conferences in the Data. The independent conference is listed as 2 different conferences (ind and Ind) as a result of a discrepency in the capitalization and the PAC-12 was formerly known as the PAC-10 which is why there is a P10 as well as a P12. We fix these issues below and reprint the bar graph.

```
s_cleaned["conf"][s_cleaned["conf"] == "ind"] <- "Ind"
s_cleaned["conf"][s_cleaned["conf"] == "P10"] <- "P12"
```

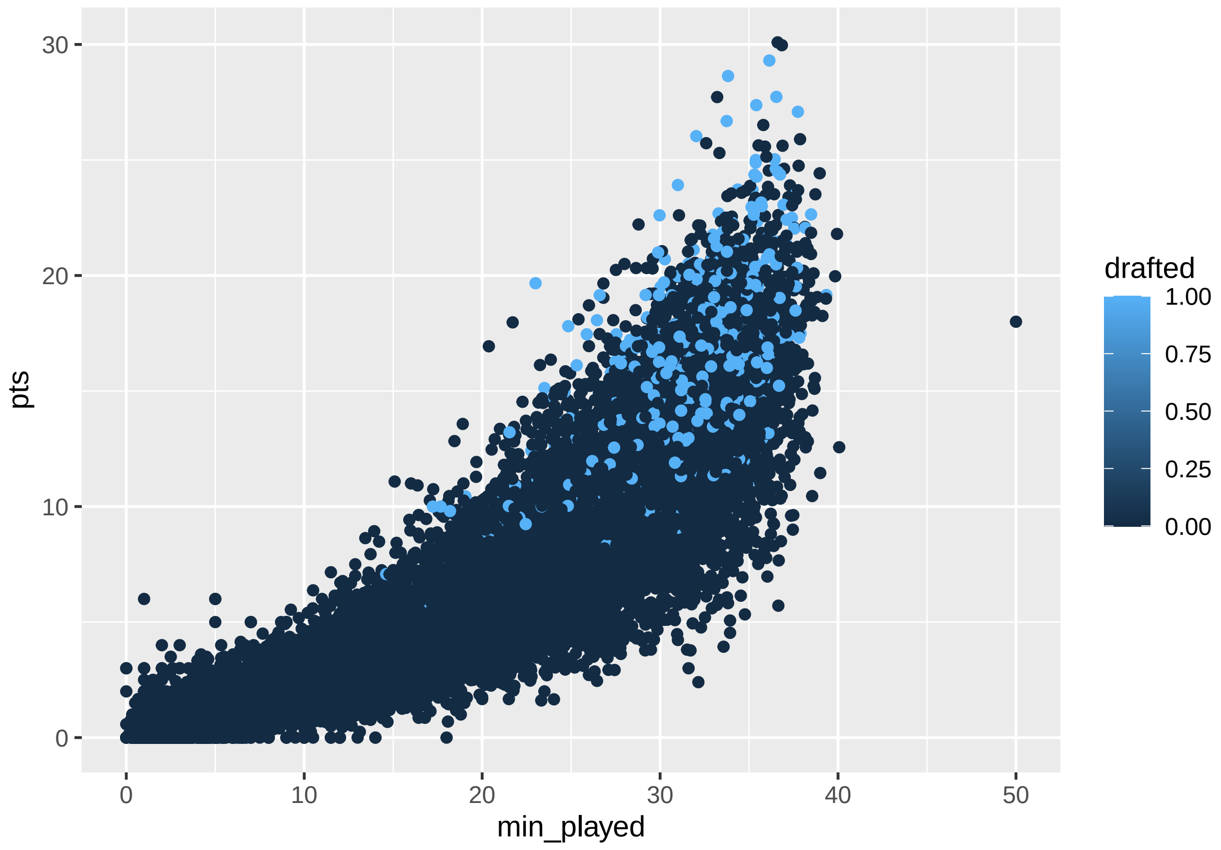
```
draft_by_conf <-  s_cleaned %>% group_by(conf) %>% summarise(prop_drafted = mean(drafted))
ggplot(draft_by_conf) + geom_bar(aes(x = conf, y = prop_drafted), stat = "identity") +
  theme(axis.text.x = element_text(angle = 90))
```



To better understand the relationship between our predictors, we make scatteplots with 2 predictors on the axis and color indicating whether that player was drafted.

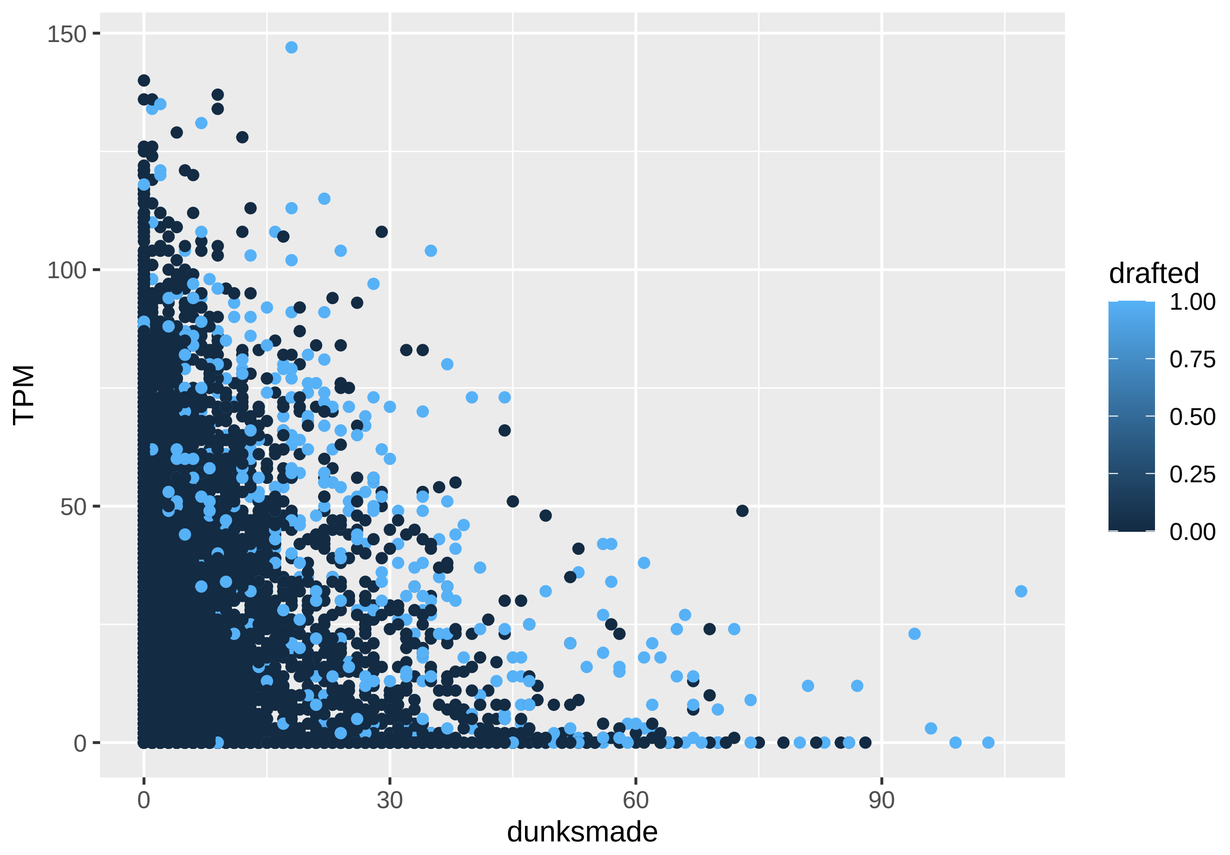
This first scatterplot has minutes played on the x axis and Points on the y axis. As expected these two variables are highly positively correlated with players who would go on to be drafted having higher values of both ending up in the top right of this plot.

```
s_cleaned %>% ggplot(aes(x=min_played, y = pts, col = drafted)) + geom_point()
```



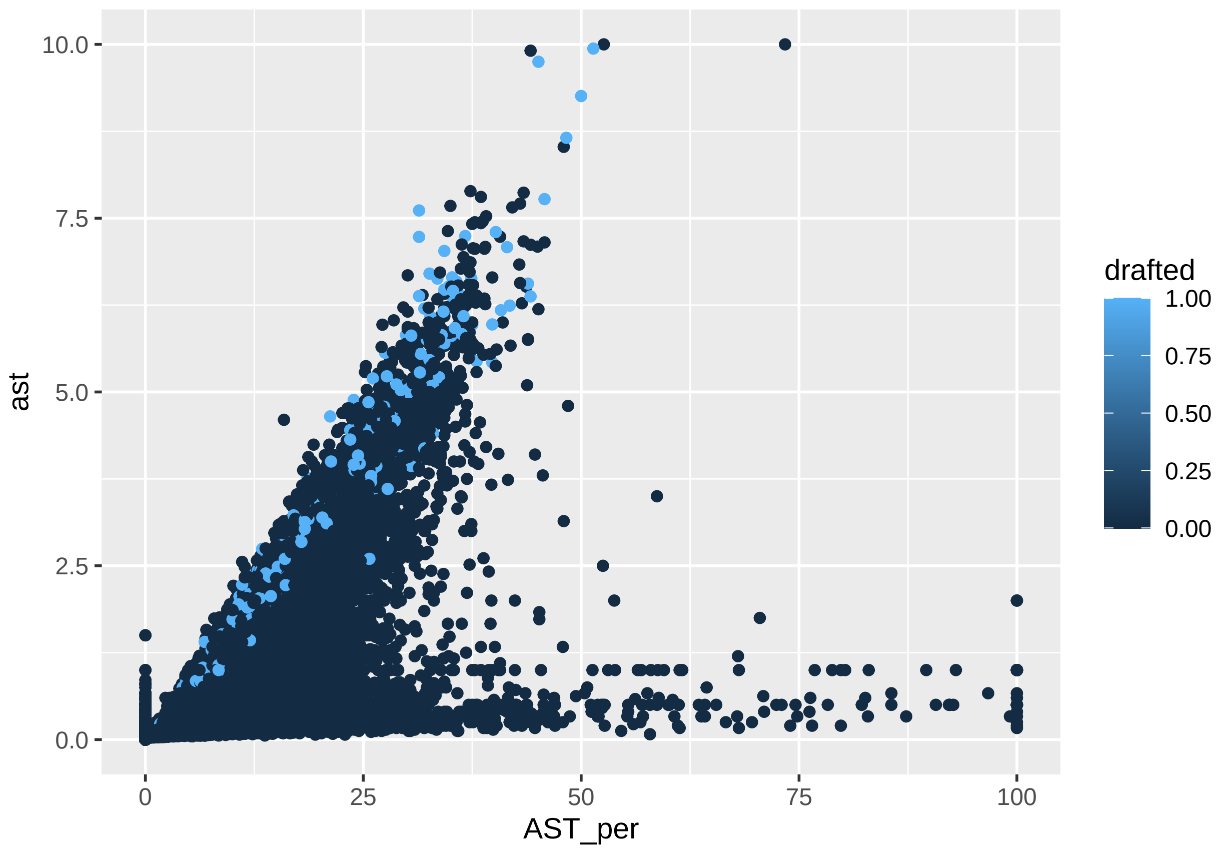
We made another scatterplot to examine the relationship between Dunks made and three pointers made.

```
s_cleaned %>% ggplot(aes(x=dunksmade, y = TPM, col = drafted)) + geom_point()
```



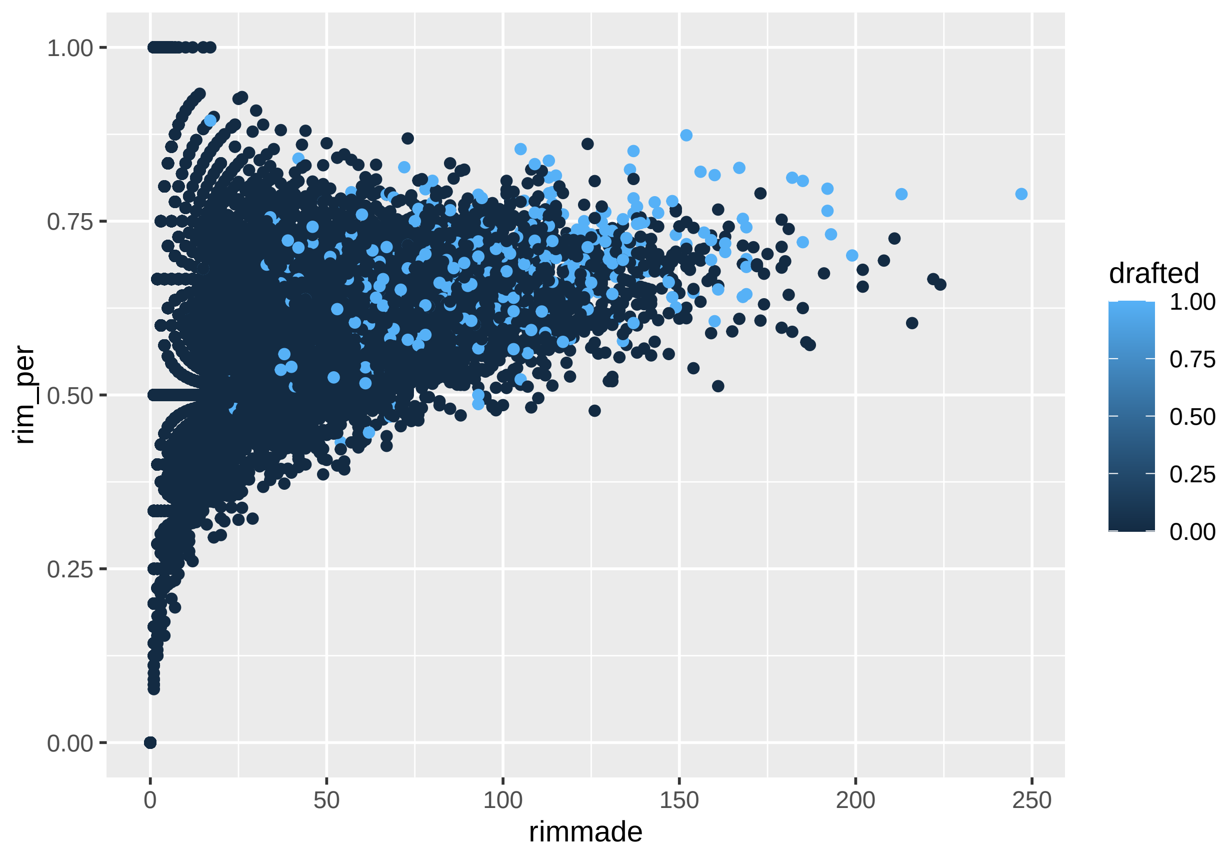
The metrics of assists (ast) and assist percentage are base off of similar stats and therefore we expect them to be highly correlated. We wonder if they should both be included in the model as predictors. We plot a scatterplot of the data on these two variables below.

```
s_cleaned %>% ggplot(aes(x=AST_per, y = ast, col = drafted)) + geom_point()
```



From the scatterplot we can tell that the y axis (ast) is more informative to a players draft status than the x axis (ast_per) is. There are so many players that dont play all that often and could, as a result of sampling variation have a high assist percentage. None of these players wind up getting drafted. Therefore, we believe it is best to use assists as a predictor rather than assist percentage because the two metrics are highly correlated for players with a significant amount of data, but just looking at assist percentage could be misleading because of the players that aren't particularly good but still have a high assist percentage due to their small sample size.

```
s_cleaned %>% ggplot(aes(x=rimmade, y = rim_per, col = drafted)) + geom_point()
```



```r
s_cleaned2 <- na.omit(s_cleaned)
data <- cor(s_cleaned2[sapply(s_cleaned2,is.numeric)])

data1 <- melt(data)
data1 <- data1 %>% filter(value < 1)
data2 <- data1[order(data1$value),]

colnames(data2) <- c("Variable1", "Variable2", "Correlation")
head(data2, 10)
```

```
##       Variable1 Variable2 Correlation
## 2449         X2        X1  -0.9804456
## 2499         X1        X2  -0.9804456
## 1838        bpm     adrtg  -0.6681862
## 1937      adrtg       bpm  -0.6681862
## 159      TO_per      Ortg  -0.6316886
## 454        Ortg    TO_per  -0.6316886
## 368         TPA   ORB_per  -0.6250078
## 908     ORB_per       TPA  -0.6250078
## 472       adjoe    TO_per  -0.6026993
## 1110     TO_per     adjoe  -0.6026993
```

Test/Training Split

In order to simplify our code when we fit our models later, we start by simplifying our dataset to only include certain columns that will be used as predictors. Additionaly, categorical predictors such as conference, seasons, and drafted must be converted to factors.

```
sc <- s_cleaned
sc$conference <- as.factor(s_cleaned$conf)
sc$num_seasons <- as.factor(s_cleaned$seasons)
sc$drafted <- factor(s_cleaned$drafted, levels = c(0,1))
stats_df <- sc %>% ungroup() %>%
  dplyr::select(c(pid, year, conference, num_seasons, Min_per, usg, FTM, FT_per,
                  twoPM, twoP_per, TPM, TP_per, adjoe, rimmade, rim_per, midmade,
                  mid_per, dunksmade, dunk_per, stops, min_played, off_reb,
                  def_reb, total_reb, ast, stl, blk, pts, height, drafted))
```

Random forests cannot contain missing values in predictor columns, therefore the columns with many missing values (rimmade, midmade, dunksmade, rim_per, mid_per, and dunk_per) will have their missing values replaced with 0.

```
sapply(stats_df, function(x) sum(is.na(x)))
```

```
##         pid        year  conference num_seasons     Min_per         usg
##           0           0           0           0           0           0
##         FTM      FT_per       twoPM    twoP_per         TPM      TP_per
##           0           0           0           0           0           0
##       adjoe     rimmade     rim_per     midmade     mid_per   dunksmade
##           0        2418        4112        2418        4144        2418
##    dunk_per       stops  min_played     off_reb     def_reb   total_reb
##       12152          12           9           9           9           9
##         ast         stl         blk         pts      height     drafted
##           9           9           9           9           0           0
```

```
stats_df[is.na(stats_df)] <- 0
sapply(stats_df, function(x) sum(is.na(x)))
```

```
##         pid        year  conference num_seasons     Min_per         usg
##           0           0           0           0           0           0
##         FTM      FT_per       twoPM    twoP_per         TPM      TP_per
##           0           0           0           0           0           0
##       adjoe     rimmade     rim_per     midmade     mid_per   dunksmade
##           0           0           0           0           0           0
##    dunk_per       stops  min_played     off_reb     def_reb   total_reb
##           0           0           0           0           0           0
##         ast         stl         blk         pts      height     drafted
##           0           0           0           0           0           0
```

We will split the data into a test set and a training set. Since we have a relatively large number of observations, we should have a large enough test set if we used only 15 percent of the data for the test set, leaving many observations in the training set to fit our models.

We sample for our test set using a stratified random sample. This is a good idea in this case because …

```
set.seed(123)
test_set <- stats_df %>% group_by(year) %>% sample_frac(size = 0.15)
test_set <- test_set[order(test_set$pid),]
training_set <- stats_df[!stats_df$pid %in% test_set$pid,]
table(test_set$drafted)
```

```
##
##    0    1
## 3025   92
```

```
table(training_set$drafted)
```

```
##
##     0     1
## 17169   489
```

Now that we've split the data into test and training sets, we can remove the year column as it will not be used as a predictor in the model. Player id is also no longer necessary. Both of these columns will be removed from both the training set as well as the test set

```
training_set <- training_set %>% ungroup()
test_set <- test_set %>% ungroup()
training_set <- dplyr::select(training_set, -c(pid,year))
test_set <- dplyr::select(test_set, -c(pid,year))
```

The two classes we have (Drafted vs not drafted) are extremely imbalanced. Less than 3 percent of the players in out data get drafted. This is a problem because if left this way our models will incorrectly classify many of the drafted players as not drafted. It could even have a relatively low error rate by simply classifying every observation as not drafted. Something needs to be done about this.

In order to mitigate this problem we will employ sampling techniques using the ovun_sample() function. In this case we will both oversample and undersample. The minority class(drafted players) will be oversampled with replacement while the majority class (undrafted players) will be undersampled with replacement. We will keep the same number of datapoints as there were in the origional training set.

```
set.seed(112)
balanced_train_set <- ovun.sample(drafted ~ ., data = training_set, method = "both", p=0.3,
table(balanced_train_set$drafted)
```

```
##
##     0     1
## 12362  5296
```

Model Fitting

Our process for fitting the models will follow the following steps:

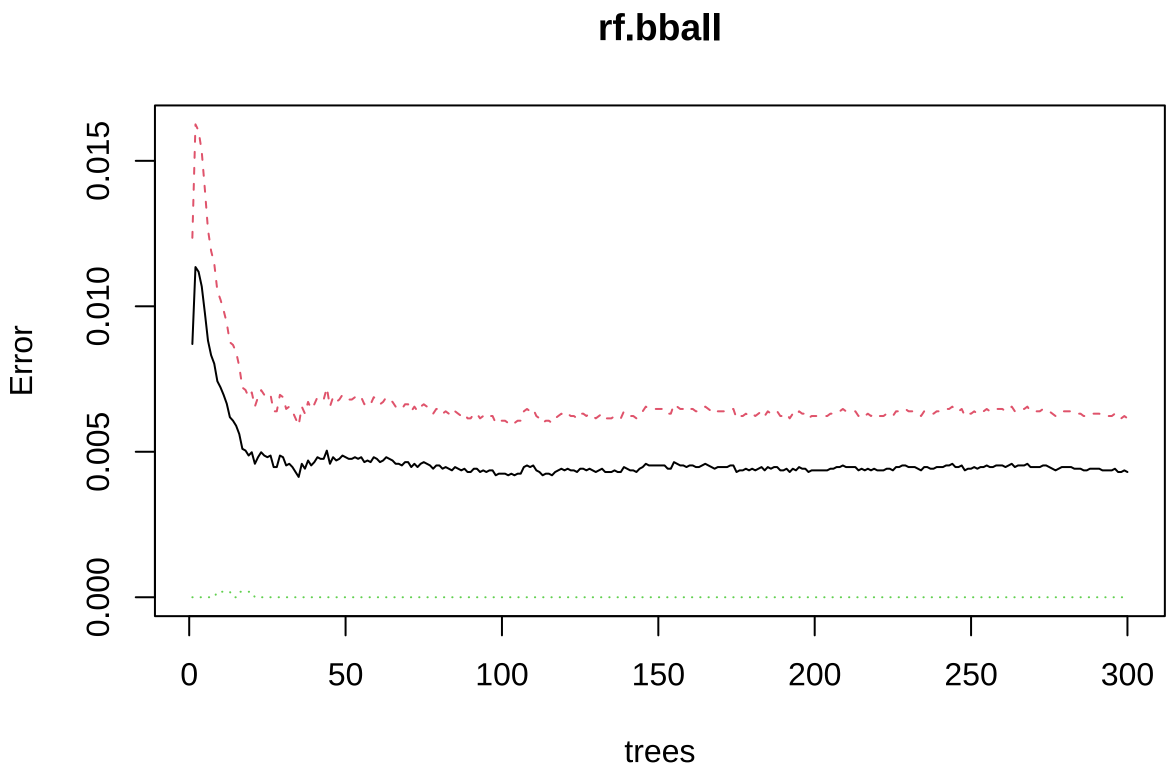
1. First we'll fit an initial models using parameters that seem intuitive.
2. Next, we'll use cross validation to tune those parameters to find the best fit
3. Then we will fit a model using the optimal parameters we found and analyze each of the models using some sort of visualization.

When using cross validation, we will do this by looping through a list of reasonable numbers for that parameter. We will fit a model on all of the training set with the exception of the fold being used as the validation set for that iteration. We'll record the validation error, or out-of-bag error in the case of the random forest model, to estimate the test error. We will then use the value for the parameter that results in the lowest estimated test error when constructing our final, optimal model.

Random Forest:

Fit an initial random forest model. Here I use mtry = 5 as it is close to the square root of the number of predictors we have. We'll use cross validation later to determine the optimal number for this.

```
rf.bball = randomForest(drafted ~ conference + num_seasons + Min_per + usg + FTM + FT_per + twoPM +
                          twoP_per + TPM + TP_per + adjoe + rimmade + rim_per + midmade + mid_per +
                          dunksmade + dunk_per + stops + min_played + off_reb + def_reb + total_reb +
                          ast + stl + blk + pts + height, data=balanced_train_set,
                        mtry=5, ntree=300, importance=TRUE)
plot(rf.bball)
```



To cross validate we will loop through all the reasonable values for m. We fit a model with each of the m's and record the out-of-bag error rate. We also make note of the classification error for the true values because if we are trying to predict which players get drafted, it is important to correctly classify as many of the drafted players as possible. Since the proportion of players that get drafted is small, minimizing the overall error might not do the best job at correctly classifying as many of the drafted players as possible.

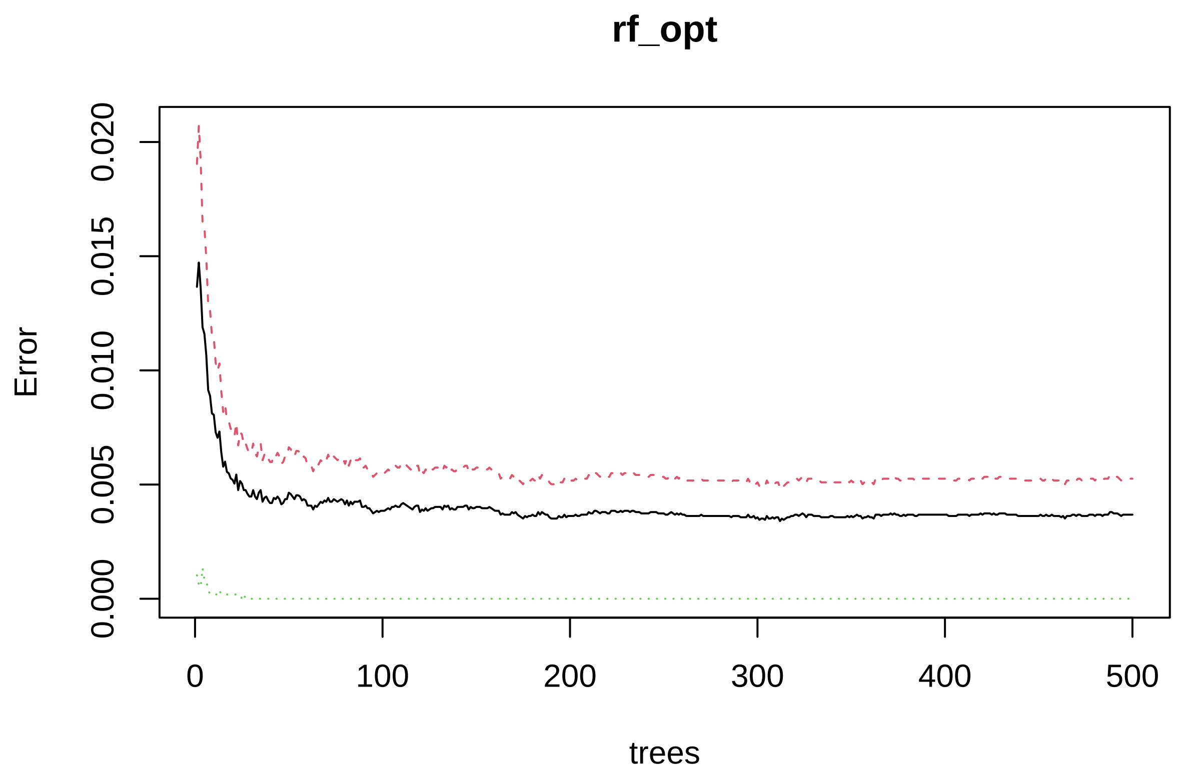
```
m = c()
tce = c()
fce = c()
oob = c()
ntree = 500
for (i in 2:15){
  rf = randomForest(drafted ~ conference + num_seasons + Min_per + usg + FTM + FT_per + twoPM +
                      twoP_per + TPM + TP_per + adjoe + rimmade + rim_per + midmade + mid_per +
                      dunksmade + dunk_per + stops + min_played + off_reb + def_reb +
                      total_reb + ast + stl + blk + pts + height, data=balanced_train_set,
                    mtry=i, ntree=ntree, importance=TRUE)

  m[i-1] <- i
  tce[i-1] <- rf$confusion[2,3]
  oob[i-1] <- rf$err.rate[ntree, 1]
  fce[i-1] <- rf$confusion[1,3]
}
err = data.frame(cbind(m=m, class_error_true = tce, class_error_false = fce, out_of_bag = oob))
small_err <- err %>% filter(out_of_bag==min(out_of_bag))
m_optimal = max(small_err$m)
```

It looks like We fit a model with this parameter and plot the results below.

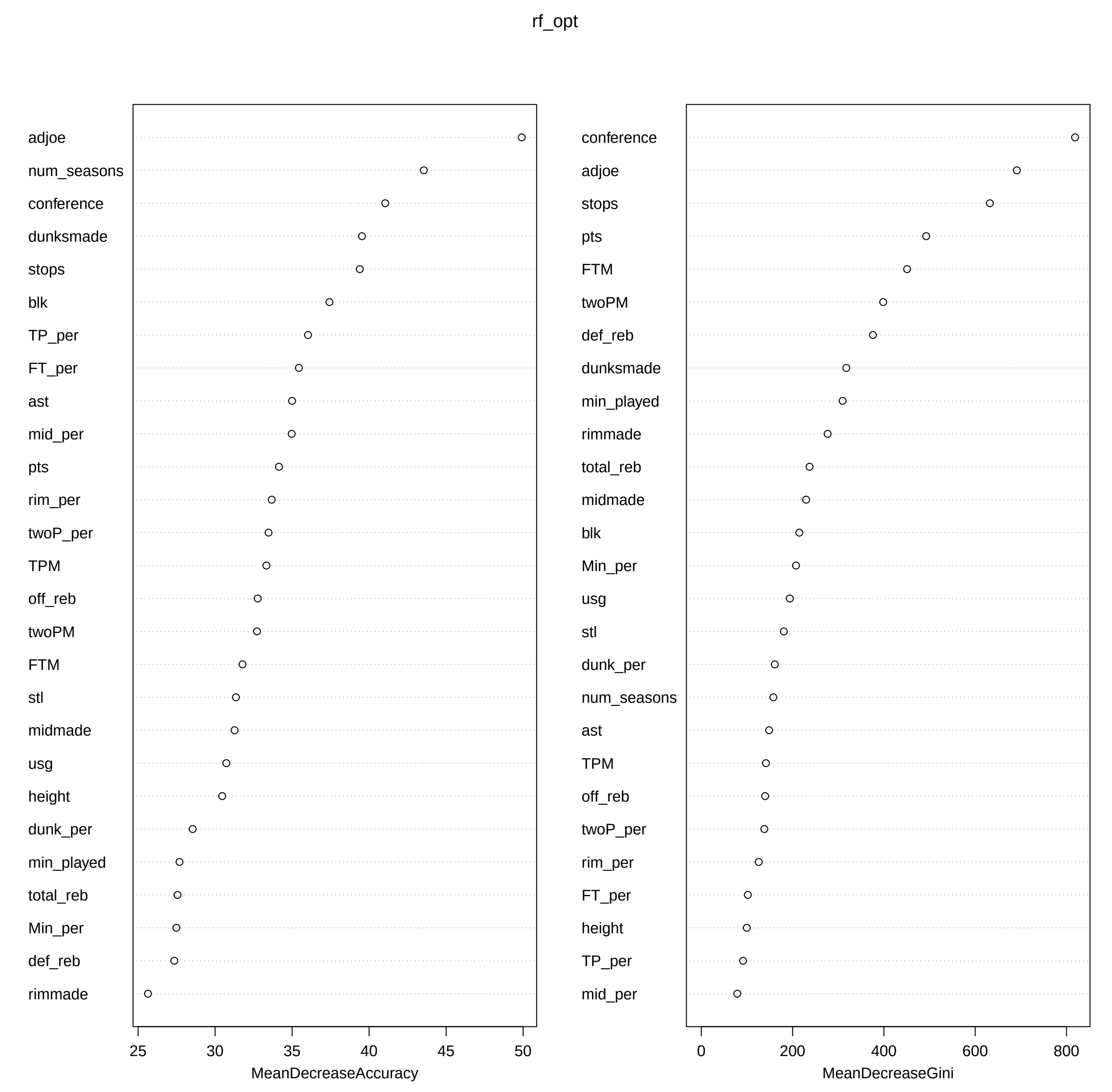
```
rf_opt = randomForest(drafted ~ conference + num_seasons + Min_per + usg + FTM + FT_per + twoPM +
                        twoP_per + TPM + TP_per + adjoe + rimmade + rim_per + midmade + mid_per +
                        dunksmade + dunk_per + stops + min_played + off_reb + def_reb +
                        total_reb + ast + stl + blk + pts + height, data=balanced_train_set,
                      mtry=m_optimal, ntree=ntree, importance=TRUE)
rf_opt
```

```
##
## Call:
##  randomForest(formula = drafted ~ conference + num_seasons + Min_per +      usg + FTM + FT_per + twoI
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 0.37%
## Confusion matrix:
##       0    1 class.error
## 0 12297   65 0.005258049
## 1     0 5296 0.000000000
```

```
plot(rf_opt)
```



```
varImpPlot(rf_opt)
```



Logistic Regression:

We'll Start by fitting a logistic regression model with all of the predictors.

```
full_model <- glm(drafted ~ ., data = balanced_train_set, family = 'binomial')
summary(full_model)
```

```
##
## Call:
## glm(formula = drafted ~ ., family = "binomial", data = balanced_train_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6860  -0.0215  -0.0001   0.0554   2.7920
```

```
##
## Coefficients:
##                  Estimate Std. Error z value Pr(>|z|)    
## (Intercept)    -4.891e+01  2.676e+00 -18.274  < 2e-16 ***
## conferenceACC   2.047e+00  2.668e-01   7.670 1.72e-14 ***
## conferenceAE   -1.777e+01  7.644e+02  -0.023 0.981449    
## conferenceAmer  2.212e-02  3.259e-01   0.068 0.945889    
## conferenceASun -1.747e+01  6.973e+02  -0.025 0.980016    
## conferenceB10   1.218e+00  2.853e-01   4.269 1.97e-05 ***
## conferenceB12   1.869e+00  2.657e-01   7.033 2.02e-12 ***
## conferenceBE    9.480e-01  2.732e-01   3.470 0.000521 ***
## conferenceBSky -7.590e-01  3.828e-01  -1.983 0.047360 *  
## conferenceBSth -1.821e+01  6.642e+02  -0.027 0.978130    
## conferenceBW   -1.273e+00  4.637e-01  -2.745 0.006048 ** 
## conferenceCAA   4.238e-02  3.503e-01   0.121 0.903703    
## conferenceCUSA -1.378e+00  3.452e-01  -3.992 6.56e-05 ***
## conferenceGWC  -1.748e+01  1.416e+03  -0.012 0.990152    
## conferenceHorz -3.621e-01  3.925e-01  -0.922 0.356282    
## conferenceInd  -1.892e+01  1.204e+03  -0.016 0.987469    
## conferenceIvy  -3.745e-01  4.870e-01  -0.769 0.441912    
## conferenceMAAC -1.748e+01  6.718e+02  -0.026 0.979238    
## conferenceMAC  -1.934e+00  4.883e-01  -3.961 7.46e-05 ***
## conferenceMEAC -1.694e+00  4.543e-01  -3.729 0.000192 ***
## conferenceMVC  -1.237e+00  4.260e-01  -2.904 0.003686 ** 
## conferenceMWC   5.880e-01  2.816e-01   2.088 0.036784 *  
## conferenceNEC  -1.637e+01  6.999e+02  -0.023 0.981346    
## conferenceOVC  -7.540e-01  4.568e-01  -1.650 0.098855 .  
## conferenceP12   2.032e+00  2.610e-01   7.786 6.92e-15 ***
## conferencePat   1.940e-01  4.979e-01   0.390 0.696838    
## conferenceSB   -6.405e-01  3.731e-01  -1.717 0.086005 .  
## conferenceSC   -3.760e+00  9.333e-01  -4.028 5.62e-05 ***
## conferenceSEC   2.155e+00  2.654e-01   8.121 4.63e-16 ***
## conferenceSlnd -1.673e+01  5.470e+02  -0.031 0.975603    
## conferenceSum  -1.136e+00  5.406e-01  -2.101 0.035651 *  
## conferenceSWAC -1.785e+01  5.555e+02  -0.032 0.974369    
## conferenceWAC  -5.171e-01  4.431e-01  -1.167 0.243229    
## conferenceWCC   2.516e-01  3.354e-01   0.750 0.453208    
## num_seasons2   -3.099e+00  2.018e-01 -15.355  < 2e-16 ***
## num_seasons3   -3.458e+00  2.154e-01 -16.052  < 2e-16 ***
## num_seasons4   -4.410e+00  2.048e-01 -21.535  < 2e-16 ***
## num_seasons5   -6.542e+00  4.836e-01 -13.529  < 2e-16 ***
## num_seasons6   -1.860e+01  6.441e+03  -0.003 0.997696    
## Min_per        -1.280e-01  1.053e-02 -12.154  < 2e-16 ***
## usg             1.071e-01  2.477e-02   4.323 1.54e-05 ***
## FTM             9.135e-03  2.353e-03   3.882 0.000104 ***
## FT_per          2.103e+00  5.694e-01   3.694 0.000221 ***
## twoPM           2.517e-02  5.039e-03   4.996 5.86e-07 ***
## twoP_per        2.178e+00  9.484e-01   2.296 0.021669 *  
## TPM             5.622e-02  6.055e-03   9.285  < 2e-16 ***
## TP_per         -1.747e-01  4.149e-01  -0.421 0.673688    
## adjoe           4.544e-02  4.789e-03   9.488  < 2e-16 ***
## rimmade        -2.252e-02  3.717e-03  -6.060 1.36e-09 ***
## rim_per         2.832e+00  5.764e-01   4.913 8.98e-07 ***
## midmade         9.428e-05  4.683e-03   0.020 0.983937    
## mid_per         2.262e+00  6.928e-01   3.266 0.001093 **
## dunksmade       4.868e-02  5.517e-03   8.823  < 2e-16 ***
## dunk_per        1.460e+00  1.764e-01   8.273  < 2e-16 ***
## stops           2.306e-02  4.574e-03   5.042 4.62e-07 ***
## min_played      2.768e-01  3.134e-02   8.832  < 2e-16 ***
## off_reb        -1.837e+03  9.061e+02  -2.027 0.042669 *
## def_reb        -1.837e+03  9.061e+02  -2.027 0.042624 *
## total_reb       1.837e+03  9.061e+02   2.027 0.042643 *
## ast             6.563e-01  6.010e-02  10.921  < 2e-16 ***
## stl             2.547e-01  1.850e-01   1.377 0.168510
## blk             5.392e-01  1.238e-01   4.354 1.34e-05 ***
## pts            -1.400e-01  6.457e-02  -2.167 0.030207 *
## height          3.734e-01  2.887e-02  12.938  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 21570.9  on 17657  degrees of freedom
## Residual deviance:  3534.3  on 17594  degrees of freedom
## AIC: 3662.3
##
## Number of Fisher Scoring iterations: 19
```

We notice that there are definitely some predictors that are more useful than others. We will use backward stepwise selection to determine the best subset of predictors for our model.

```
step_model <- full_model %>% stepAIC(trace = FALSE)
summary(step_model)
```

```
##
## Call:
## glm(formula = drafted ~ conference + num_seasons + Min_per +
##     usg + FTM + FT_per + twoPM + twoP_per + TPM + adjoe + rimmade +
##     rim_per + mid_per + dunksmade + dunk_per + stops + min_played +
##     off_reb + def_reb + total_reb + ast + blk + pts + height,
##     family = "binomial", data = balanced_train_set)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.6682  -0.0216  -0.0001   0.0553   2.7932
##
## Coefficients:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -4.844e+01  2.639e+00 -18.355  < 2e-16 ***
## conferenceACC   2.067e+00  2.666e-01   7.754 8.92e-15 ***
## conferenceAE   -1.809e+01  7.567e+02  -0.024 0.980931
## conferenceAmer  5.749e-02  3.251e-01   0.177 0.859631
## conferenceASun -1.742e+01  7.041e+02  -0.025 0.980264
## conferenceB10   1.256e+00  2.845e-01   4.416 1.01e-05 ***
## conferenceB12   1.879e+00  2.660e-01   7.064 1.62e-12 ***
## conferenceBE    9.880e-01  2.720e-01   3.632 0.000281 ***
## conferenceBSky -7.208e-01  3.825e-01  -1.885 0.059487 .
## conferenceBSth -1.806e+01  6.676e+02  -0.027 0.978419    
## conferenceBW   -1.230e+00  4.618e-01  -2.664 0.007730 ** 
## conferenceCAA   1.091e-01  3.459e-01   0.315 0.752437    
## conferenceCUSA -1.357e+00  3.450e-01  -3.932 8.43e-05 ***
## conferenceGWC  -1.743e+01  1.420e+03  -0.012 0.990205    
## conferenceHorz -3.456e-01  3.927e-01  -0.880 0.378791    
## conferenceInd  -1.899e+01  1.202e+03  -0.016 0.987399    
## conferenceIvy  -3.570e-01  4.879e-01  -0.732 0.464354    
## conferenceMAAC -1.749e+01  6.714e+02  -0.026 0.979213    
## conferenceMAC  -1.973e+00  4.905e-01  -4.023 5.75e-05 ***
## conferenceMEAC -1.722e+00  4.550e-01  -3.785 0.000154 ***
## conferenceMVC  -1.225e+00  4.266e-01  -2.873 0.004071 ** 
## conferenceMWC   6.096e-01  2.811e-01   2.169 0.030113 *  
## conferenceNEC  -1.633e+01  6.995e+02  -0.023 0.981378    
## conferenceOVC  -6.960e-01  4.520e-01  -1.540 0.123581    
## conferenceP12   2.039e+00  2.611e-01   7.809 5.76e-15 ***
## conferencePat   1.369e-01  5.038e-01   0.272 0.785832    
## conferenceSB   -6.334e-01  3.727e-01  -1.700 0.089216 .  
## conferenceSC   -3.731e+00  9.188e-01  -4.060 4.90e-05 ***
## conferenceSEC   2.160e+00  2.657e-01   8.131 4.25e-16 ***
## conferenceSlnd -1.667e+01  5.463e+02  -0.031 0.975659    
## conferenceSum  -1.065e+00  5.391e-01  -1.975 0.048301 *  
## conferenceSWAC -1.780e+01  5.576e+02  -0.032 0.974540    
## conferenceWAC  -5.102e-01  4.416e-01  -1.155 0.247930    
## conferenceWCC   2.703e-01  3.345e-01   0.808 0.419121    
## num_seasons2   -3.078e+00  2.007e-01 -15.330  < 2e-16 ***
## num_seasons3   -3.455e+00  2.152e-01 -16.057  < 2e-16 ***
## num_seasons4   -4.411e+00  2.044e-01 -21.575  < 2e-16 ***
## num_seasons5   -6.529e+00  4.816e-01 -13.557  < 2e-16 ***
## num_seasons6   -1.858e+01  6.410e+03  -0.003 0.997687    
## Min_per        -1.310e-01  1.021e-02 -12.826  < 2e-16 ***
## usg             1.124e-01  2.430e-02   4.626 3.72e-06 ***
## FTM             7.974e-03  2.153e-03   3.703 0.000213 ***
## FT_per          2.114e+00  5.632e-01   3.754 0.000174 ***
## twoPM           2.239e-02  3.582e-03   6.249 4.12e-10 ***
## twoP_per        2.239e+00  8.019e-01   2.791 0.005248 ** 
## TPM             5.211e-02  5.273e-03   9.881  < 2e-16 ***
## adjoe           4.465e-02  4.869e-03   9.171  < 2e-16 ***
## rimmade        -2.238e-02  3.078e-03  -7.272 3.53e-13 ***
## rim_per         2.837e+00  4.961e-01   5.717 1.08e-08 ***
## mid_per         2.240e+00  5.901e-01   3.796 0.000147 ***
## dunksmade       4.845e-02  5.524e-03   8.771  < 2e-16 ***
## dunk_per        1.458e+00  1.761e-01   8.278  < 2e-16 ***
## stops           2.751e-02  3.138e-03   8.765  < 2e-16 ***
## min_played      2.743e-01  3.111e-02   8.820  < 2e-16 ***
## off_reb        -1.958e+03  9.001e+02  -2.175 0.029600 *  
## def_reb        -1.959e+03  9.001e+02  -2.176 0.029560 *  
## total_reb       1.958e+03  9.001e+02   2.176 0.029580 *  
## ast             6.623e-01  6.000e-02  11.039  < 2e-16 ***
## blk             4.987e-01  1.178e-01   4.234 2.29e-05 ***
## pts            -1.043e-01  5.763e-02  -1.810 0.070292 .  
## height          3.663e-01  2.820e-02  12.990  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 21570.9  on 17657  degrees of freedom
## Residual deviance:  3536.3  on 17597  degrees of freedom
## AIC: 3658.3
##
## Number of Fisher Scoring iterations: 19
```

It looks like our code dropped a few of the predictors to make a better model. We notice that the AIC for the step_model is lower than it was in the full_model which is a good thing.

Next, we must determine which threshold is best to use to make predictions. We'll loop through all of the reasonable thresholds and then record the true positive rate, false positive rate, and total error for each. We'll use these metrics to decide on which threshold is best.

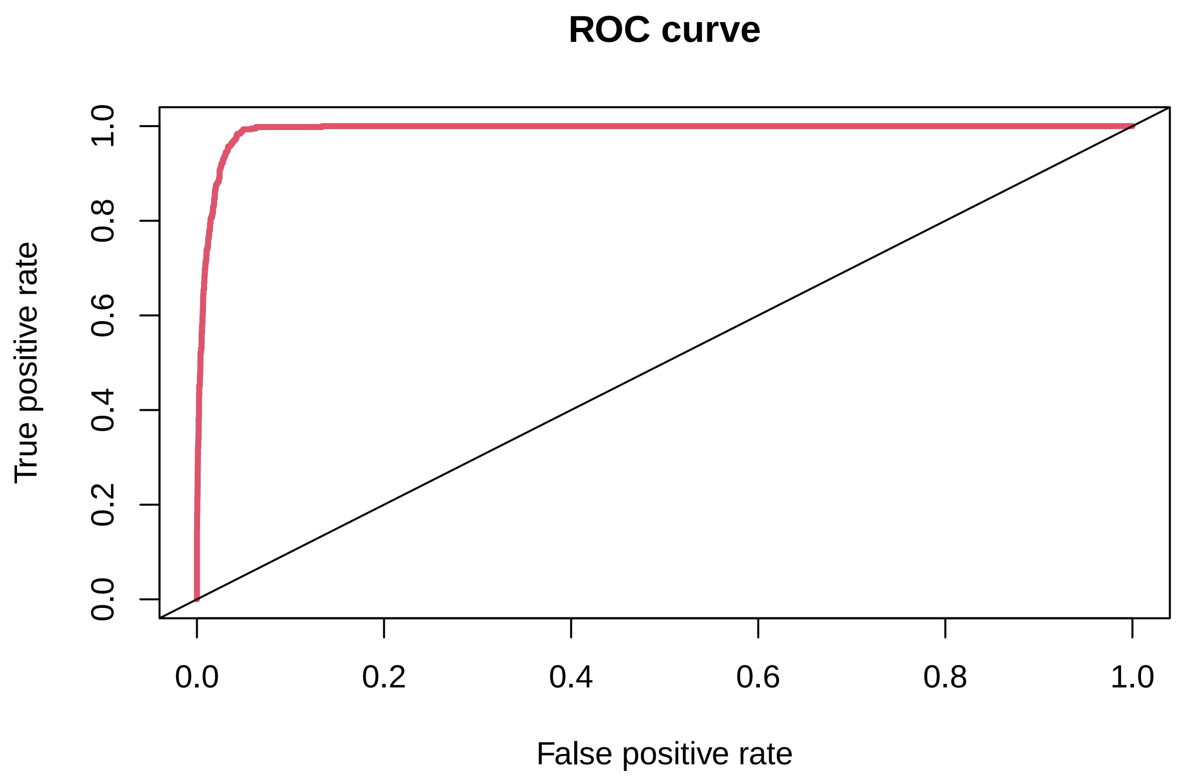
```
prob.training = predict(step_model, type="response")
t <- c()
tpr <- c()
fpr <- c()
error <- c()

for (i in 1:9){

  bal_trn_st_w_pred = balanced_train_set %>%
    mutate(pred_drafted=as.factor(ifelse(prob.training<=(i/10), "No", "Yes")))
  con_mat <- table(pred=bal_trn_st_w_pred$pred_drafted, true=balanced_train_set$drafted)

  t[i] <- i/10
  tpr[i] <- con_mat[2,2]/(con_mat[2,2] + con_mat[1,2])
  fpr[i] <- con_mat[2,1]/(con_mat[2,1] + con_mat[1,1])
  error[i] <- (con_mat[1, 2] + con_mat[2,1])/(con_mat[1, 2] + con_mat[2,1] +
                                                con_mat[2, 2] + con_mat[1,1])
}
err_logit = data.frame(cbind(t=t, true_pos_rate = tpr, false_pos_rate = fpr, total_error = error))
err_logit
```

```
##     t true_pos_rate false_pos_rate total_error
## 1 0.1     0.9981118     0.07887073  0.05578208
## 2 0.2     0.9950906     0.05937551  0.04303998
## 3 0.3     0.9886707     0.04829316  0.03720693
## 4 0.4     0.9724320     0.04109367  0.03703704
## 5 0.5     0.9490181     0.03292348  0.03833956
## 6 0.6     0.9307024     0.02871704  0.04088798
## 7 0.7     0.8834970     0.02313541  0.05113829
## 8 0.8     0.8342145     0.01828183  0.06252124
## 9 0.9     0.7413142     0.01067788  0.08506060
```

```
small_err_logit <- err_logit %>% filter(total_error==min(total_error))
optimal_threshold = max(small_err_logit$t)
```

Constructing and ROC Curve

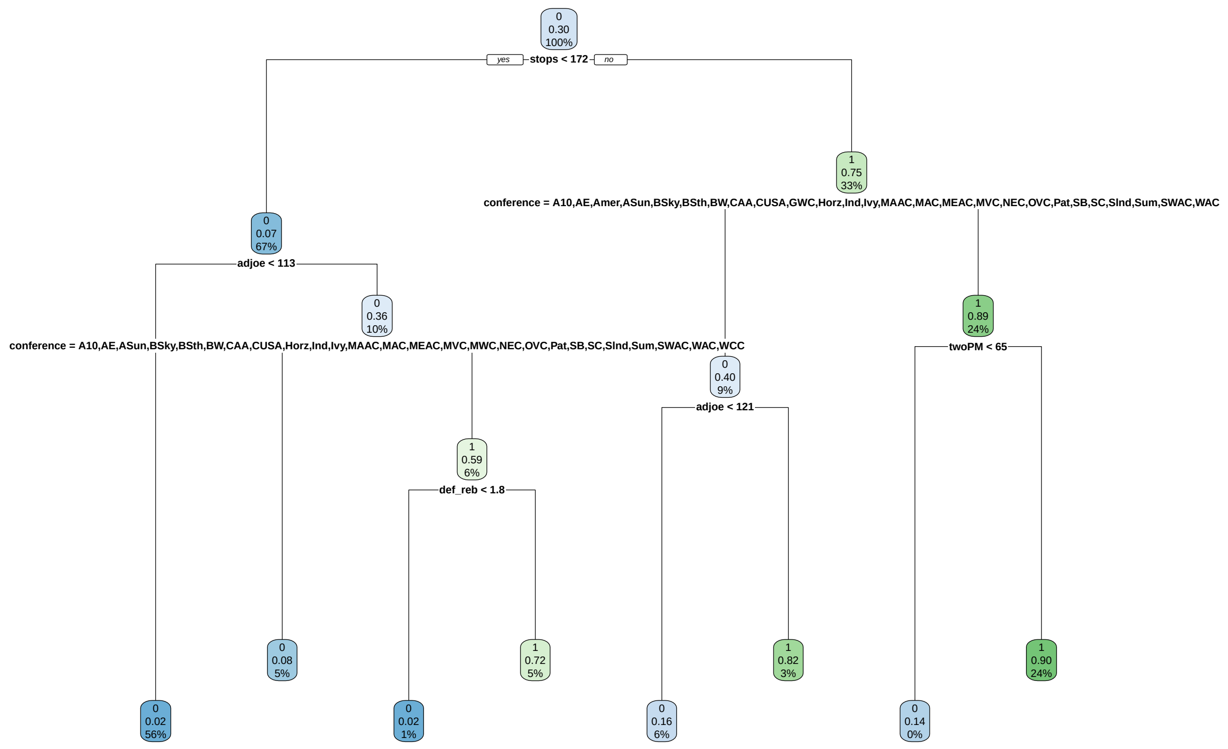
```r
pred = prediction(prob.training, balanced_train_set$drafted)
perf = performance(pred, measure="tpr", x.measure="fpr")
plot(perf, col=2, lwd=3, main="ROC curve")
abline(0,1)
```



Boosted Trees

In this section we fit boosted trees model to the data.

```r
#Cross Validation
set.seed(123)
boosted_fit <- rpart(drafted ~., data = balanced_train_set, method = "class", xval = 10)
rpart.plot(boosted_fit, extra = 106, yesno = TRUE)
```

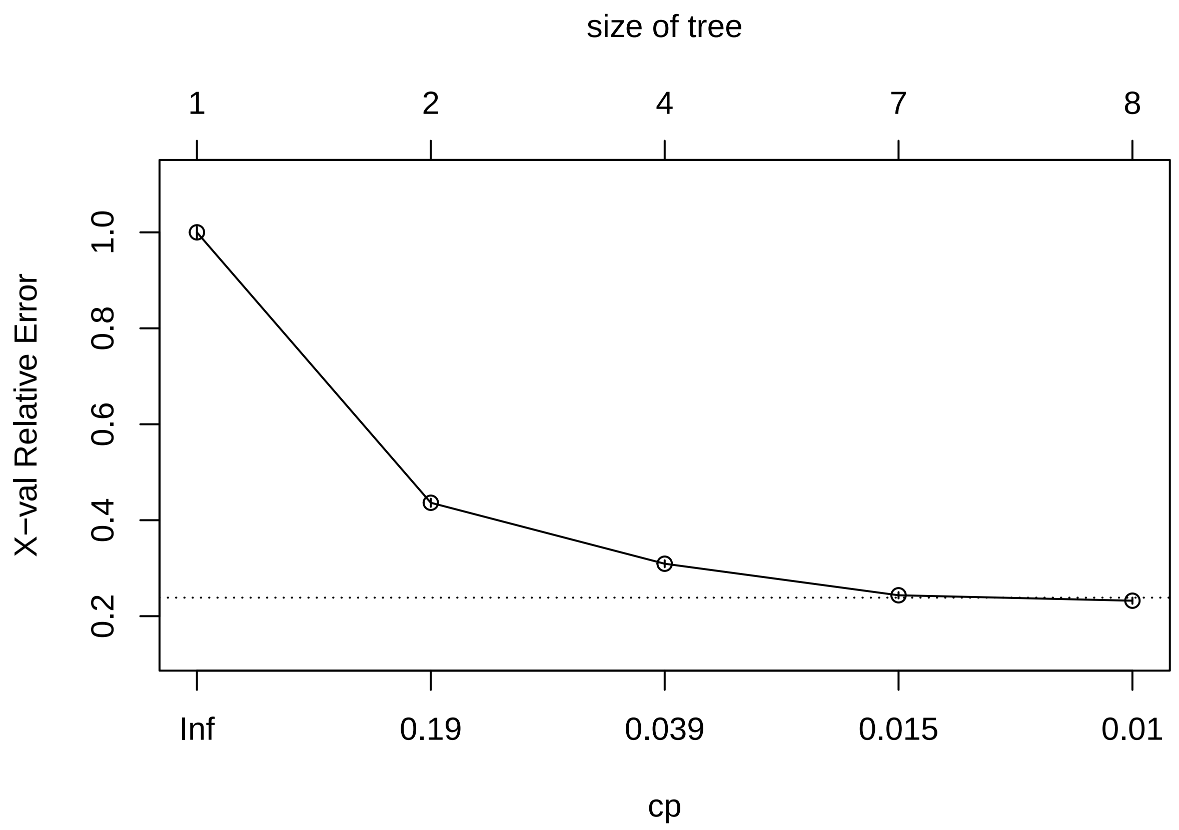


0
0.30
100%
yes
stops < 172
no
1
0.75
33%
conference = A10,AE,Amer,ASun,BSky,BSth,BW,CAA,CUSA,GWC,Horz,Ind,Ivy,MAAC,MAC,MEAC,MVC,NEC,OVC,Pat,SB,SC,SInd,Sum,SWAC,WAC
0
0.07
67%
adjoe < 113
0
0.36
10%
conference = A10,AE,ASun,BSky,BSth,BW,CAA,CUSA,Horz,Ind,Ivy,MAAC,MAC,MEAC,MVC,MWC,NEC,OVC,Pat,SB,SC,SInd,Sum,SWAC,WAC,WCC
1
0.89
24%
twoPM < 65
0
0.40
9%
adjoe < 121
1
0.59
6%
def_reb < 1.8
0
0.08
5%
1
0.72
5%
1
0.82
3%
1
0.90
24%
0
0.02
56%
0
0.02
1%
0
0.16
6%
0
0.14
0%

```
printcp(boosted_fit)
```

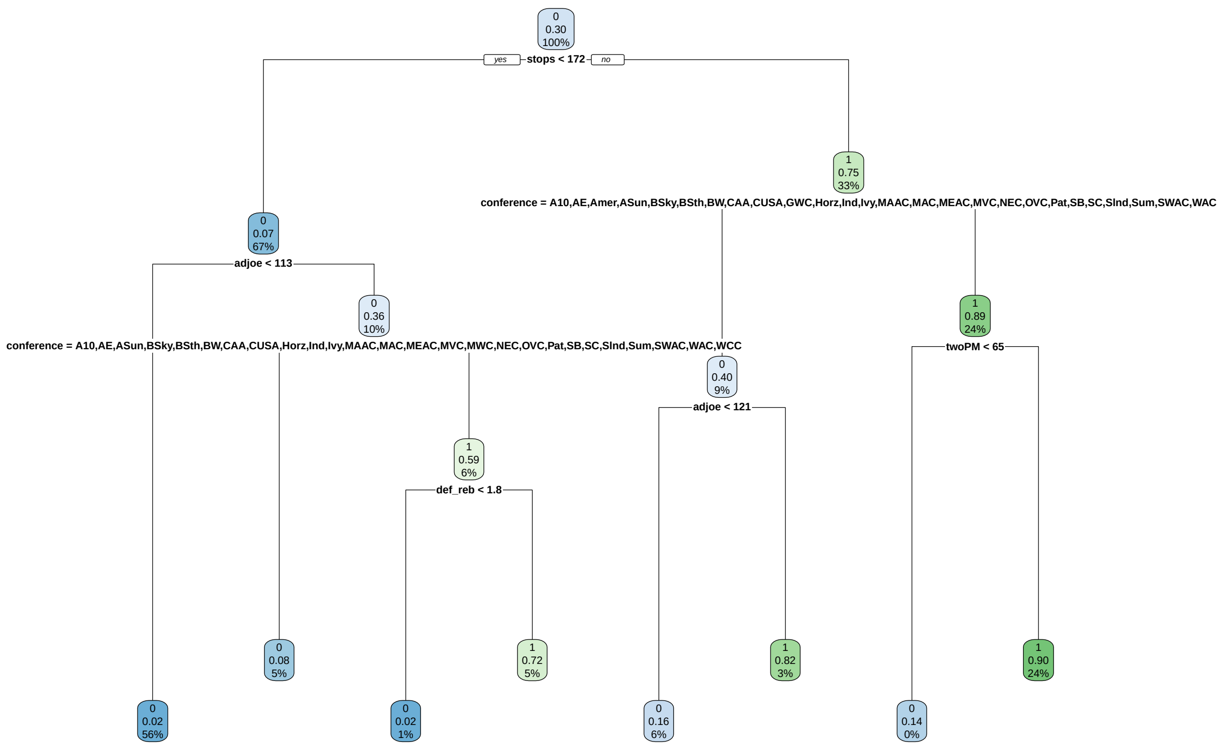
```
##
## Classification tree:
## rpart(formula = drafted ~ ., data = balanced_train_set, method = "class",
##     xval = 10)
##
## Variables actually used in tree construction:
## [1] adjoe      conference def_reb    stops      twoPM
##
## Root node error: 5296/17658 = 0.29992
##
## n= 17658
##
##         CP nsplit rel error  xerror      xstd
## 1 0.566276      0   1.00000 1.00000 0.0114974
## 2 0.066654      1   0.43372 0.43637 0.0084624
## 3 0.022659      3   0.30042 0.30929 0.0072790
## 4 0.010574      6   0.23244 0.24358 0.0065294
## 5 0.010000      7   0.22187 0.23225 0.0063874
```

```
plotcp(boosted_fit)
```



We prune the boosted tree in the following chunk.

```r
#Potential Pruning
new_prune_boosted <- prune(boosted_fit,
                  cp = boosted_fit$cptable[which.min(boosted_fit$cptable[, "xerror"]), "CP"])
rm(boosted_fit)
rpart.plot(new_prune_boosted, extra = 106, yesno = TRUE)
```


0
0.30
100%
yes
stops < 172
no
1
0.75
33%
conference = A10,AE,Amer,ASun,BSky,BSth,BW,CAA,CUSA,GWC,Horz,Ind,Ivy,MAAC,MAC,MEAC,MVC,NEC,OVC,Pat,SB,SC,SInd,Sum,SWAC,WAC
0
0.07
67%
adjoe < 113
0
0.36
10%
conference = A10,AE,ASun,BSky,BSth,BW,CAA,CUSA,Horz,Ind,Ivy,MAAC,MAC,MEAC,MVC,MWC,NEC,OVC,Pat,SB,SC,SInd,Sum,SWAC,WAC,WCC
1
0.89
24%
twoPM < 65
0
0.40
9%
adjoe < 121
1
0.59
6%
def_reb < 1.8
0
0.02
56%
0
0.08
5%
0
0.02
1%
1
0.72
5%
0
0.16
6%
1
0.82
3%
0
0.14
0%
1
0.90
24%

K-Nearest-Neighbors

```
str(training_set)
```

```
## tibble [17,658 x 28] (S3: tbl_df/tbl/data.frame)
##  $ conference : Factor w/ 34 levels "A10","ACC","AE",..: 27 33 8 34 11 15 16 1 23 23 ...
##  $ num_seasons: Factor w/ 6 levels "1","2","3","4",..: 2 3 1 1 1 2 2 2 1 1 ...
##  $ Min_per    : num [1:17658] 11.5 64.5 72 44.5 56.2 1 81.8 1.3 82.8 80.4 ...
##  $ usg        : num [1:17658] 16.9 18.7 21.8 16 22 16.8 27.7 0 20.5 23 ...
##  $ FTM        : int [1:17658] 3 53 45 14 64 0 123 0 76 64 ...
##  $ FT_per     : num [1:17658] 0.5 0.815 0.672 0.519 0.561 0 0.658 0 0.697 0.533 ...
##  $ twoPM      : int [1:17658] 4 43 67 25 93 0 126 0 82 128 ...
##  $ twoP_per   : num [1:17658] 0.364 0.394 0.427 0.397 0.528 0 0.483 0 0.439 0.456 ...
##  $ TPM        : int [1:17658] 4 49 50 28 0 2 33 0 25 12 ...
##  $ TP_per     : num [1:17658] 0.222 0.368 0.325 0.452 0 0.667 0.363 0 0.431 0.218 ...
##  $ adjoe      : num [1:17658] 65.8 103.9 104.1 93.2 97.9 ...
##  $ rimmade    : int [1:17658] 1 30 0 0 0 0 82 0 0 0 ...
##  $ rim_per    : num [1:17658] 0.5 0.536 0 0 0 ...
##  $ midmade    : int [1:17658] 3 13 0 0 0 0 44 0 0 0 ...
##  $ mid_per    : num [1:17658] 0.333 0.245 0 0 0 ...
##  $ dunksmade  : int [1:17658] 0 6 0 0 0 0 3 0 0 0 ...
##  $ dunk_per   : num [1:17658] 0 0.857 0 0 0 ...
##  $ stops      : num [1:17658] 17.4 118 115 84.2 128 ...
##  $ min_played : num [1:17658] 18.9 26 33.2 18 22.9 ...
##  $ off_reb    : num [1:17658] 0.571 0.151 0.63 0.7 1.424 ...
##  $ def_reb    : num [1:17658] 1.14 1.48 2.33 1.43 3.3 ...
##  $ total_reb  : num [1:17658] 1.71 1.64 2.96 2.13 4.73 ...
##  $ ast        : num [1:17658] 1.429 2.091 1.963 1.1 0.849 ...
##  $ stl        : num [1:17658] 0.143 0.212 0.481 0.567 0.455 ...
##  $ blk        : num [1:17658] 0 0.0909 0 0.1333 0.3333 ...
##  $ pts        : num [1:17658] 3.29 8.67 12.19 4.93 7.58 ...
##  $ height     : num [1:17658] 74 76 76 76 80 74 77 72 78 77 ...
##  $ drafted    : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```r
table(training_set$num_seasons)
```

```
##
##    1    2    3    4    5    6
## 5260 4760 2645 4713  274    6
```

```r
table(training_set$conference)
```

```
##
##  A10  ACC   AE Amer ASun  B10  B12   BE BSky BSth   BW  CAA CUSA  GWC Horz  Ind
##  589  626  410  302  478  527  465  545  600  518  492  456  678  117  508  179
##  Ivy MAAC  MAC MEAC  MVC  MWC  NEC  OVC  P12  Pat   SB   SC  SEC Slnd  Sum SWAC
##  446  520  591  807  462  512  532  650  544  415  665  517  608  735  459  772
##  WAC  WCC
##  503  430
```

Before we begin, we have to do something about the categorical variables we are using as predictors. A k-nearest-neighbors model would not work with variables like conference and num_seasons as we currently have them. We will dummycode these variables in both the training set as well as the test set.

```r
conference <- as.data.frame(dummy.code(balanced_train_set$conference))
num_seasons <- as.data.frame(dummy.code(balanced_train_set$num_seasons))
conference_t <- as.data.frame(dummy.code(test_set$conference))
num_seasons_t <- as.data.frame(dummy.code(test_set$num_seasons))
trn_st <- cbind(balanced_train_set, conference, num_seasons)
trn_st <- trn_st %>% dplyr::select(-one_of(c("conference", "num_seasons","6")))
tst_st <- cbind(test_set, conference_t, num_seasons_t)
tst_st <- tst_st %>% dplyr::select(-one_of(c("conference", "num_seasons", "6")))
```

```r
# YTrain is the true labels for drafted on the training set, XTrain is the design matrix
YTrain = trn_st$drafted
XTrain = trn_st %>% dplyr::select(-drafted) %>% scale(center = TRUE, scale = TRUE)
# YTest is the true labels for drafted on the test set, Xtest is the design matrix
YTest = tst_st$drafted
XTest = tst_st %>% dplyr::select(-drafted) %>% scale(center = TRUE, scale = TRUE)
```

```r
do.chunk <- function(chunkid, folddef, Xdat, Ydat, ...){
  # Get training index
  train = (folddef!=chunkid)
  # Get training set by the above index
  Xtr = Xdat[train,]
  # Get responses in training set
  Ytr = Ydat[train]
  # Get validation set
  Xvl = Xdat[!train,]
  # Get responses in validation set
  Yvl = Ydat[!train]
  # Predict training labels
  predYtr = knn(train=Xtr, test=Xtr, cl=Ytr, ...)
  # Predict validation labels
  predYvl = knn(train=Xtr, test=Xvl, cl=Ytr, ...)
  data.frame(fold = chunkid,
  train.error = mean(predYtr != Ytr), # Training error for each fold
  val.error = mean(predYvl != Yvl)) # Validation error for each fold
}
```

Here we execute the cross validation using the do.chunk function we created above. We will take the averages of the validation error for each value for k and store the lowest on as a variable called k_optimal

```
nfold = 5
folds = cut(1:nrow(trn_st), breaks=nfold, labels=FALSE) %>% sample()
error.folds = NULL
# Give possible number of nearest neighbours to be considered
allK = 2:8
# Loop through different number of neighbors
for (k in allK){
# Loop through different chunk id
  for (j in seq(nfold)){
    tmp = do.chunk(chunkid=j, folddef=folds, Xdat=XTrain, Ydat=YTrain, k=k)
    tmp$neighbors = k # Record the last number of neighbor
    error.folds = rbind(error.folds, tmp) # combine results
  }
}
head(error.folds, 10)
```

```
##    fold train.error  val.error neighbors
## 1     1 0.007291519 0.01557191         2
## 2     2 0.007715722 0.01614274         2
## 3     3 0.006725188 0.01670442         2
## 4     4 0.008282013 0.01359388         2
## 5     5 0.007503894 0.01783692         2
## 6     1 0.015574119 0.02066818         3
## 7     2 0.016351667 0.02265647         3
## 8     3 0.014724621 0.02265006         3
## 9     4 0.016776386 0.02322288         3
## 10    5 0.016919156 0.02633069         3
```

```
smallest_err <- error.folds %>% group_by(neighbors) %>%
  summarise(neighbors = neighbors, avg_val.err = mean(val.error)) %>%
  ungroup() %>% filter(avg_val.err==min(avg_val.err))
```

```
## `summarise()` has grouped output by 'neighbors'. You can override using the `.groups` argument.
```

```r
k_optimal = max(smallest_err$neighbors)
```

This values for k_optimal is what we will use to evaluate the performance of this model and compare this k-nearest-neighbor model to the other models that we fit.

Model Performance and Selection

Make predictions of the draft status of players in the test set using the Random forest model. Print a confusion matrix and record the test error

```
yhat.rf = predict (rf_opt, newdata = test_set)
rf.err = table(pred = yhat.rf, truth = test_set$drafted)
test.rf.err = 1 - round(sum(diag(rf.err))/sum(rf.err), 6)
rf.err
```

```
##     truth
## pred    0    1
##    0 3007   36
##    1   18   56
```

```
print(paste('Random Forest Error: ', 100*test.rf.err, '%'))
```

```
## [1] "Random Forest Error:  1.7324 %"
```

Now we will make predictions using the logistic regression model

```
prob.test = predict(step_model, newdata = test_set, type="response")
test_set_w_pred = test_set %>%
  mutate(pred_drafted=as.factor(ifelse(prob.test<=(optimal_threshold), 0, 1)))
log.err <- table(pred = test_set_w_pred$pred_drafted, true = test_set$drafted)
test.log.err = round(1 - sum(diag(log.err))/sum(log.err), 6)
log.err
```

```
##     true
## pred    0    1
##    0 2917    3
##    1  108   89
```

```
print(paste('Logistic Regression Error: ', 100*test.log.err, '%'))
```

```
## [1] "Logistic Regression Error:  3.5611 %"
```

The following are predictions using the boosted tree model.

```
class.pred <- predict(new_prune_boosted, test_set, type = "class")
boost.err <- table(pred = class.pred, true = test_set$drafted)
test.boost.err = 1 - round(sum(diag(boost.err))/sum(boost.err), 6)
boost.err
```

```
##     true
## pred    0    1
##    0 2857   13
##    1  168   79
```

```
paste("Boosted Tree error: ", 100*test.boost.err,"%")
```

```
## [1] "Boosted Tree error:  5.8069 %"
```

The following are predictions using K-nearest-neighbors.

```
predYtst = knn(train=XTrain, test=XTest, cl=YTrain, k = k_optimal)
test.error = round(mean(predYtst != YTest), 6)
knn.err <- table(pred=predYtst, true=test_set$drafted)
knn.err
```

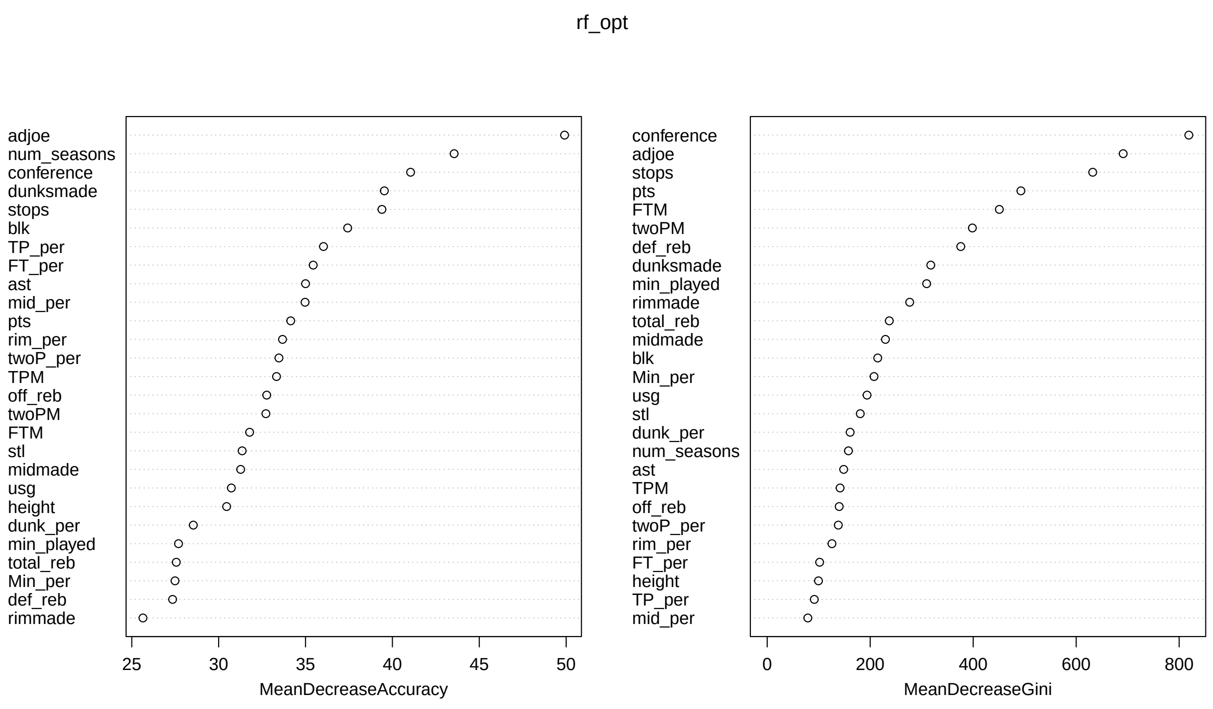
```
##     true
## pred    0    1
##    0 2792   46
##    1  233   46
```

```
print(paste('K-Nearest-Neighbor Error: ', 100 * test.error, '%'))
```

```
## [1] "K-Nearest-Neighbor Error:  8.9509 %"
```

The model with the lowest error when making predictions for the test set is the random forest model. Here is a plot that shows the importance of each predictor in the model.

```
varImpPlot(rf_opt)
```



Conclusion

The model with the lowest error when making predictions for the test set is the random forest model. That model correctly classified 56 of the 92 players in the test set who would be drafted. It also correctly classified 3007 of the 3025 players who were not drafted. of the 74 players from the test set that our model predicted would be drafted, 56 of them actually would be drafted.

The logistic regression model had the second lowest error. This model actually had correctly identified more of the players that went on to be drafted. The reason that the test error is higher is that it classified over 100 players as drafted who would ultimately not get drafted. This likely has to do with our threshold. Had the threshold been set higher we would have seen more players incorrectly classified as undrafted and less players incorrectly classified as drafted. This could be tweaked to reflect what we find most important. If our goal is to correctly classify as many drafted players, we could slightly lower the threshold. If we instead were more focused on maximizing the likelihood that a player that we classify as drafted actually gets drafted we could slightly raise the threshold. These trade offs can improve our model to better serve our goals for it.

The boosted decision tree model we fit had a test error rate of just over five percent. It surprised me that the boosted model did not perform better than the logistic regression model. In fact, it had more false positives and more false negatives. Overall, it isn't necessarily bad at predicting players' draft status, it's just that some of our other models were better.

Lastly, the model with the highest error was the k-nearest-neighbors model with a test error of just over 10 percent. For this reason it is the least useful of our models. In addition, this model took the longest time and most computing power to run by far. In the future, I will most likely not use a k-nearest-neighbor model as a first choice when I encounter a classification problem. It was still good to see in this project how it performed compared to some of our other models.