PSTAT131 Final Project

Joseph Chang, Tom Wei, Akul Bajaj

3/11/2022

Packages

Here we installed necessary packages

```
#install.packages("tidyverse")
library(ggplot2)
library(GGally)
library(tidyverse)
library(glmnet)
library(knitr)
library(dplyr)
library(tidyverse)
library(modelr)
library(pander)
library(corrplot)
library(readxl)
library(ISLR)
library(tidymodels)
library(ggthemes)
library(naniar)
library(ROCR)
library(maptree)
library(tree)
library(factoextra)
library(cluster)
library(randomForest)
```

Introduction

After a long nine to five at work, many people across the United States look to unwind with a bottle of beer and one of the world's greatest pastimes, live sports! From October to May we watch basketball stars like Lebron James dropping dimes, or Stephen Curry splashing from threes, but what factors lead these players to such success in the basketball industry, and specifically what factors affect the salaries of NBA players, which can actually range from $377,645 all the way to $45,780,966. In our project we use statistical variables such as "PPG", or "Offensive Rating" in different statistical machine learning models to predict the salaries of NBA players in the 2021-2022 season.

We apply several statistical machine learning models like logistic regression and clustering to the data in order to predict the salary of a player based on eight main predictors, carefully selected after several correlation calculations and observations. We plan to explore the benefits and detriments of each model, by calculating statistics such as mean squared error, area under the curve, etc. We hope you enjoy!

Some of players maybe repeated because he got traded during the season.

Before we dive into the data, the definitions of the variables should be clarified first. All the variables contained in our datasets will be listed below.

PPG: Points per game.

RPG: Rebounds per game.

APG: Assists per game.

SPG: Steals per game.

BPG: Blocks per game.

TPG: Turnovers per game.

MPG: Minutes per game.

Usage Rate: an estimate of the percentage of team plays used by a player while he was on the floor.

Free throw %: Free throw percentage.

Three-point %: Three point shot percentage.

Effective shooting %: a statistic that adjusts field goal percentage to account for the fact that three-point field goals count for three points while field goals only count for two points. Its goal is to show what field goal percentage a two-point shooter would have to shoot at to match the output of a player who also shoots three-pointers.

True shooting %: an advanced statistic that measures a player's efficiency at shooting the ball. It is intended to more accurately calculate a player's shooting than field goal percentage, free throw percentage, and three-point field goal percentage taken individually.

Versatility Index: measures a player's ability to produce in more than one statistic. The metric uses points, assists, and rebounds. The average player will score around a five on the index, while top players score above 10. Calculated by: Versatility Index Formula=[(PPG)*(RPG)*APG)]^(0.333)

Offensive Rating: measures an individual player's efficiency at producing points for the offense.

Defensive rating: measures an individual player's efficiency at preventing the other team from scoring points.

Player Efficiency Rating: a method of determining a player's impact on the game by measuring their per-minute performance. Rather than judging a player solely on their stats, their PER is a much more thorough performance indicator. It details a player and compares their value to that of other players in the league.

Win shares: a player statistic which attempts to divvy up credit for team success to the individuals on the team.

Box Plus Minus: estimates a basketball player's contribution to the team when that player is on the court.

Value Over Replacement: estimates each player's overall contribution to the team, measured vs. what a theoretical "replacement player" would provide, where the "replacement player" is defined as a player on minimum salary or not a normal member of a team's rotation.

Minutes Percent: percentage of team minutes used by a player while he was on the floor.

Free throws attempted: total number of free throws attempted.

Offensive box plus minus: estimates a basketball player's offensive contribution to the team when that player is on the court.

Read data

First, we read in the data that we downloaded. The first dataset is from a website called NBA stuffer, which contains many basketball statistics in an excel file. The second dataset is from kaggle and contains more basketball statistics, some different than the first dataset. The last dataset is from basketball-reference.com and lists out every players' salary in the 2021-2022 season.

```
# NBA stuffer
X2020_2021_NBA_Stats_Player_Box_Score_Advanced_Metrics <- read_excel("2020-2021 NBA Stats  Player Box S
bball_stats <- as.data.frame(X2020_2021_NBA_Stats_Player_Box_Score_Advanced_Metrics)
my_colnames <- c('Rank', 'Player', 'Team', 'Position', 'Age', 'Games_Played', 'MPG', 'Minutes_percent',
colnames(bball_stats) <- my_colnames
new_bball_stats <- bball_stats[-1,-1]

# kaggle
labels <- c('Player','Position', 'Age', 'Team', 'Games', 'Minutes_played', 'Player_Efficiency_Rating',
data_advanced <- read.csv("nba2021_advanced.csv", col.names = labels, na= "XXX")

# basektball-reference salaries
labels2 <- c('Rank', 'Player', 'Salary', 'Use', 'Guaranteed')
salaries <- read.csv("nba_salaries_21-22.csv", col.names = labels2)
```

Merge data into "master" dataset

This is where we merge the three datasets into one "master" dataset. This "master" dataset removes duplicated columns as well as columns that show unrelated/insignificant statistics. In addition, this "master" dataset changes all columns (except Player and Salary) to dbl for more convenient use later on. Small remark: a new column called Total_Minutes was added to see if it has any effect on data.

```
combine <- inner_join(new_bball_stats, data_advanced, by = 'Player') %>%
  inner_join(salaries, by = 'Player')

selection <- subset(combine, select= -c(2:4,29:33,35:45,54,56:57))
less_data <- selection %>% relocate(c(PPG,RPG,APG,SPG,BPG,TPG), .before = MPG)

conversion <- less_data[,2:35] %>% mutate_if(is.character,as.numeric) %>% mutate(Total_Minutes = Games_
conversion1 <- conversion %>% add_column(less_data$Player)
names(conversion1)[names(conversion1) == "less_data$Player"] <- "Player"
conversion2 <- conversion1 %>% relocate((Player), .before = Games_Played)
conversion3 <- conversion2 %>% relocate((Total_Minutes), .before=MPG)
```

Filter data

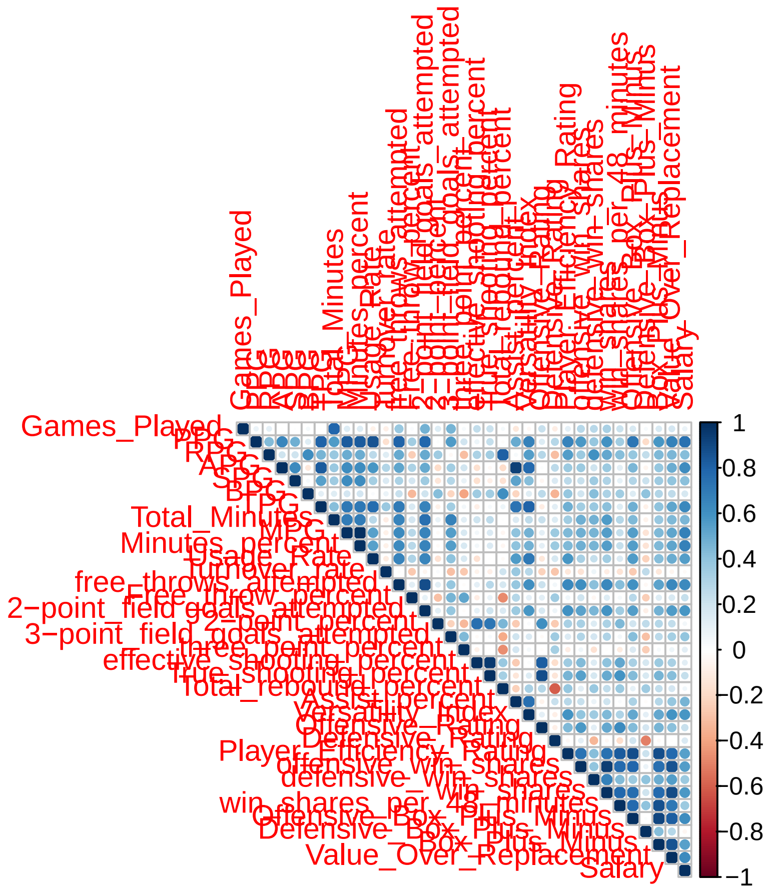
Next, we wanted to filter out players who did not see the court often or were injured during the season. Keeping such players otherwise would have skewed the data and produced outliers. Here, players with total minutes less than 336 minutes and less than 7 agmes were taken out. (The entire season consisted of 72 games. 10% of 72 games is around 7 games. A full game is 48 minutes, so 7 full games is 336 minutes). In summary, a player had to play in 90% of the games to be considered. (The previous code was used to reduce columns, here we reduce rows).

```
reduced_data <- conversion3 %>% filter(Games_Played >= 7 & Total_Minutes >= 336) %>% select(Player:Sala
```

Correlation to Salary

Since there are still a lot of predictors (columns) left, we need a way to only show the important ones. The rest can be omitted. Thus, we used a correlation plot and found correlation coefficients as related to salary.

```
my_cor <- reduced_data %>% select_if(is.numeric) %>% drop_na() %>% cor() %>% round(3)
corrplot(my_cor, method = "circle", type = "upper", cex.pch=10)
```



```
salarycor <- reduced_data %>% select(Salary, Games_Played:Value_Over_Replacement) %>% drop_na()
cor(salarycor)[,"Salary"]
```

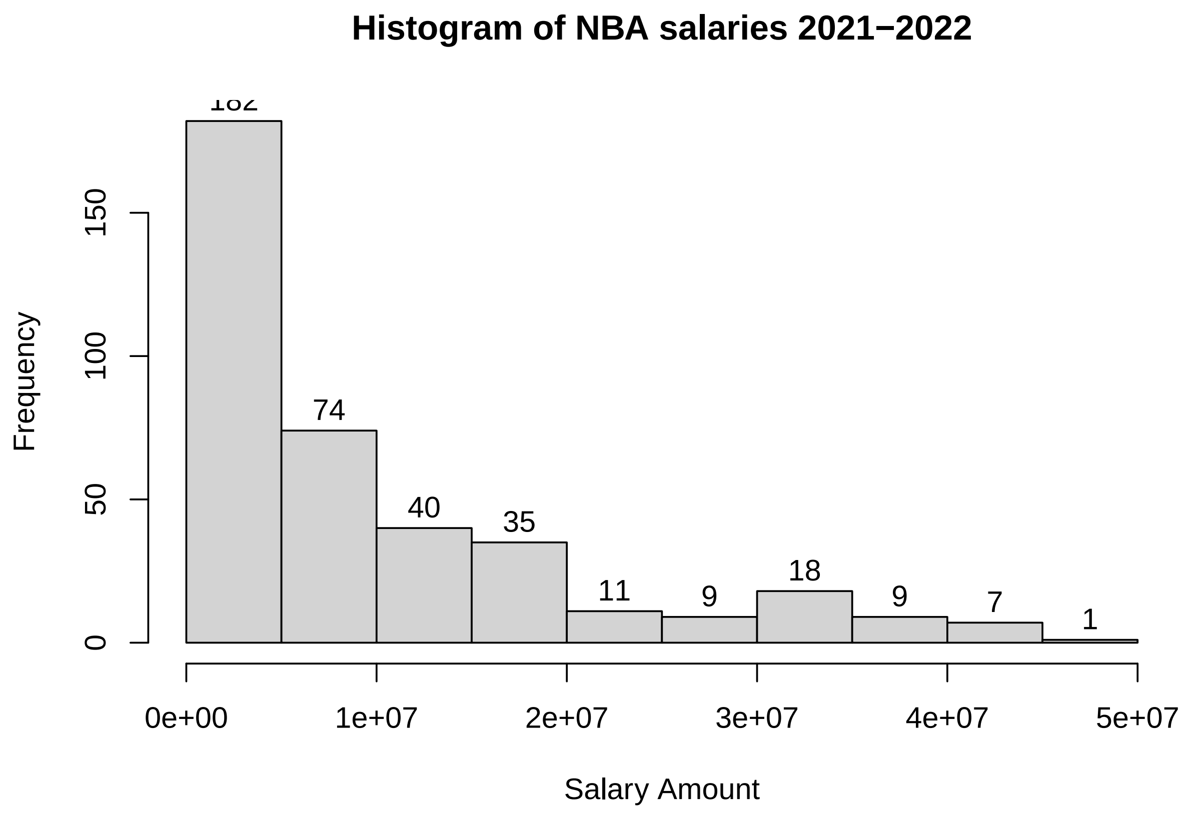
```
##                    Salary              Games_Played
##                   1.00000                   0.08676
##                       PPG                       RPG
##                   0.73988                   0.42568
##                       APG                       SPG
##                   0.62033                   0.41657
##                       BPG                       TPG
##                   0.17649                   0.64799
##             Total_Minutes                       MPG
##                      0.45932                        0.67001
##              Minutes_percent                     Usage_Rate
##                      0.67008                        0.55813
##                Turnover_rate          free_throws_attempted
##                     -0.02539                        0.63296
##           Free_throw_percent 2-point_field goals_attempted
##                      0.23258                        0.58382
##              2-point_percent 3-point_field_goals_attempted
##                      0.05285                        0.39700
##           three_point_percent    effective_shooting_percent
##                      0.12758                        0.11977
##         True_shooting_percent          Total_rebound_percent
##                      0.23653                        0.05951
##               Assist_percent               Versatility_Index
##                      0.46091                        0.58910
##              Offensive_Rating                Defensive_Rating
##                      0.22285                        0.15608
##      Player_Efficiency_Rating           offensive_win_shares
##                      0.54481                        0.55423
##           defensive_win_shares                     win_shares
##                      0.36287                        0.57743
##      win_shares_per_48_minutes       Offensive_Box_Plus_Minus
##                      0.36651                        0.62317
##      Defensive_Box_Plus_Minus                  Box_Plus_Minus
##                     -0.04593                        0.54717
##        Value_Over_Replacement
##                      0.63365
```

Based from the correlation, we will filter those correlations with Salary above 0.6, meaning we will only use PPG, APG, MPG, TPG, Minutes_percent, free_throws_attempted, Offensive_Box_Plus_Minus, Value_Over_Replacement.

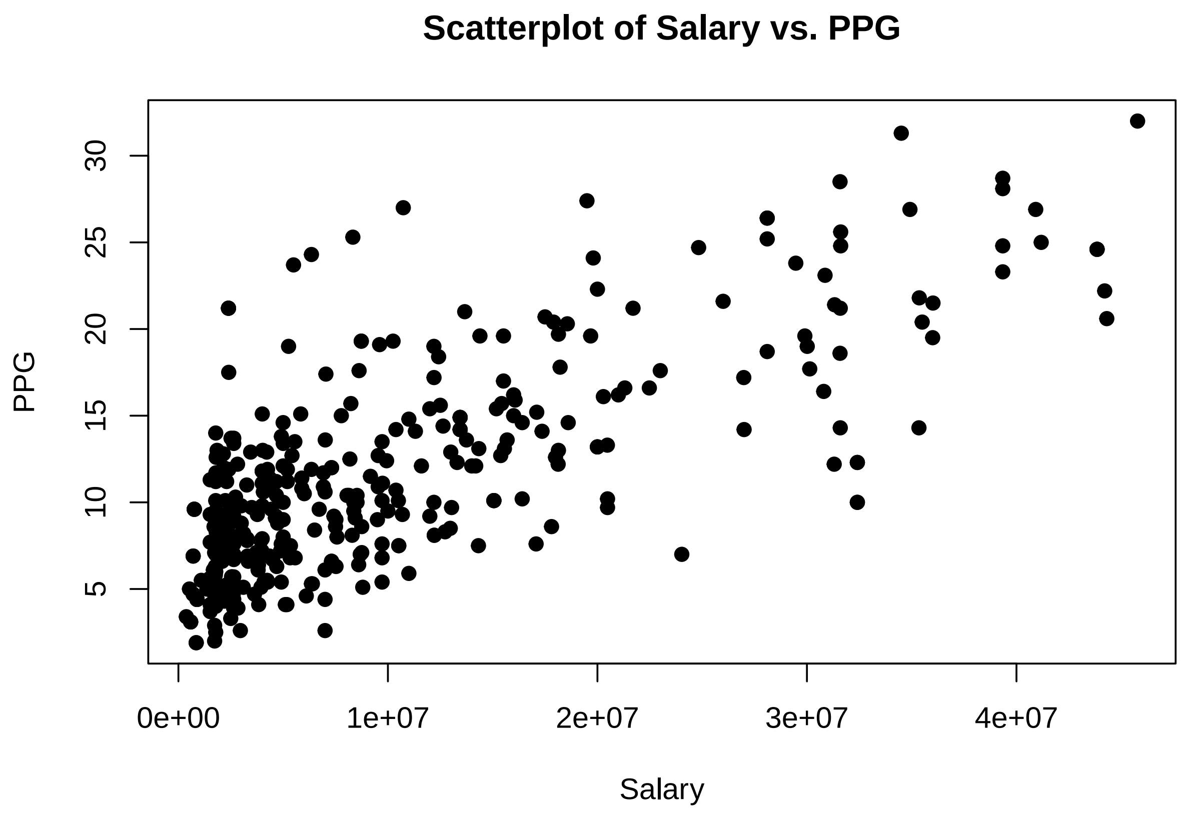
Final dataset and Histograms

This is where we put the final dataset. It has our players, their salaries, and the 8 predictors we want. We will be referring to our_data the rest of the project. As a start to visualize the data, we plotted the response variable Salary using a histogram and the scatter plots of Salary vs. the 8 predictors which has the greatest correlation with salary.

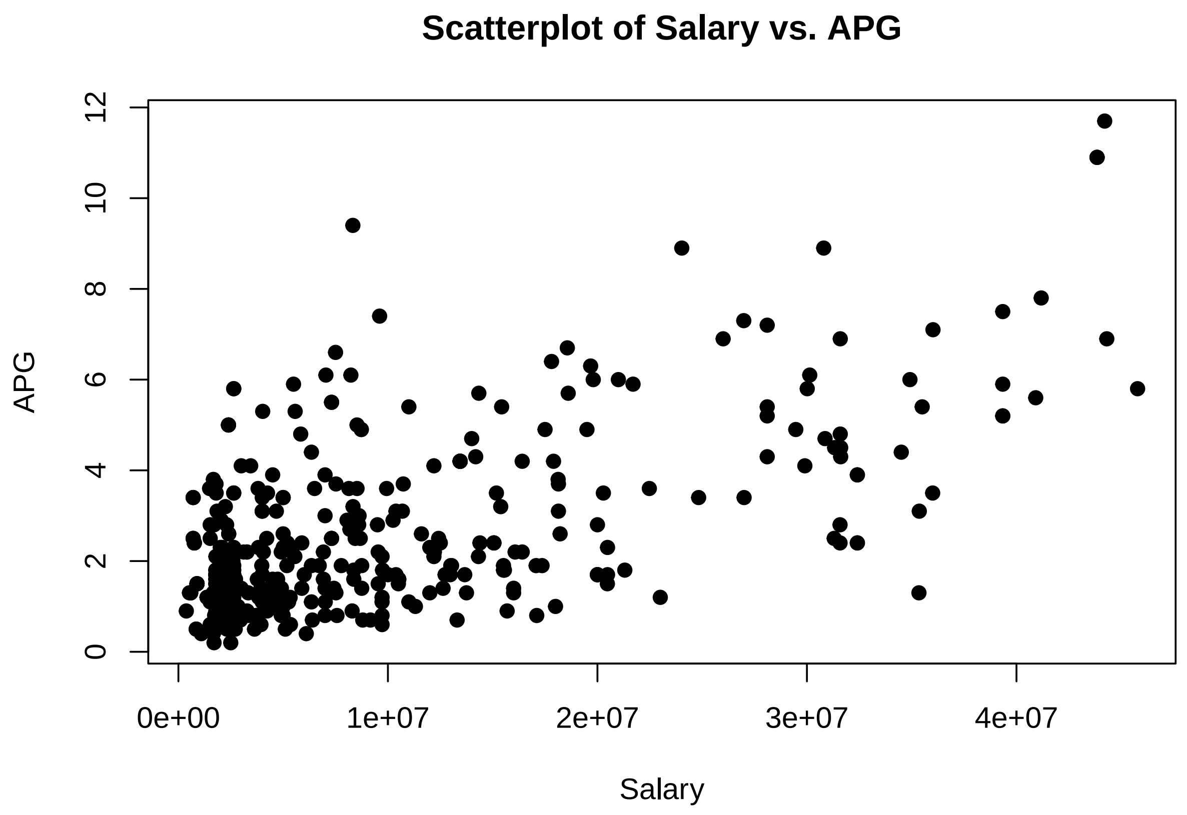
```
our_data <- reduced_data %>% select(Player, Salary, PPG, APG, MPG, TPG, MPG, Minutes_percent, free_thro

hist(our_data$Salary, main = "Histogram of NBA salaries 2021-2022", xlab="Salary Amount",breaks = "Stur
```



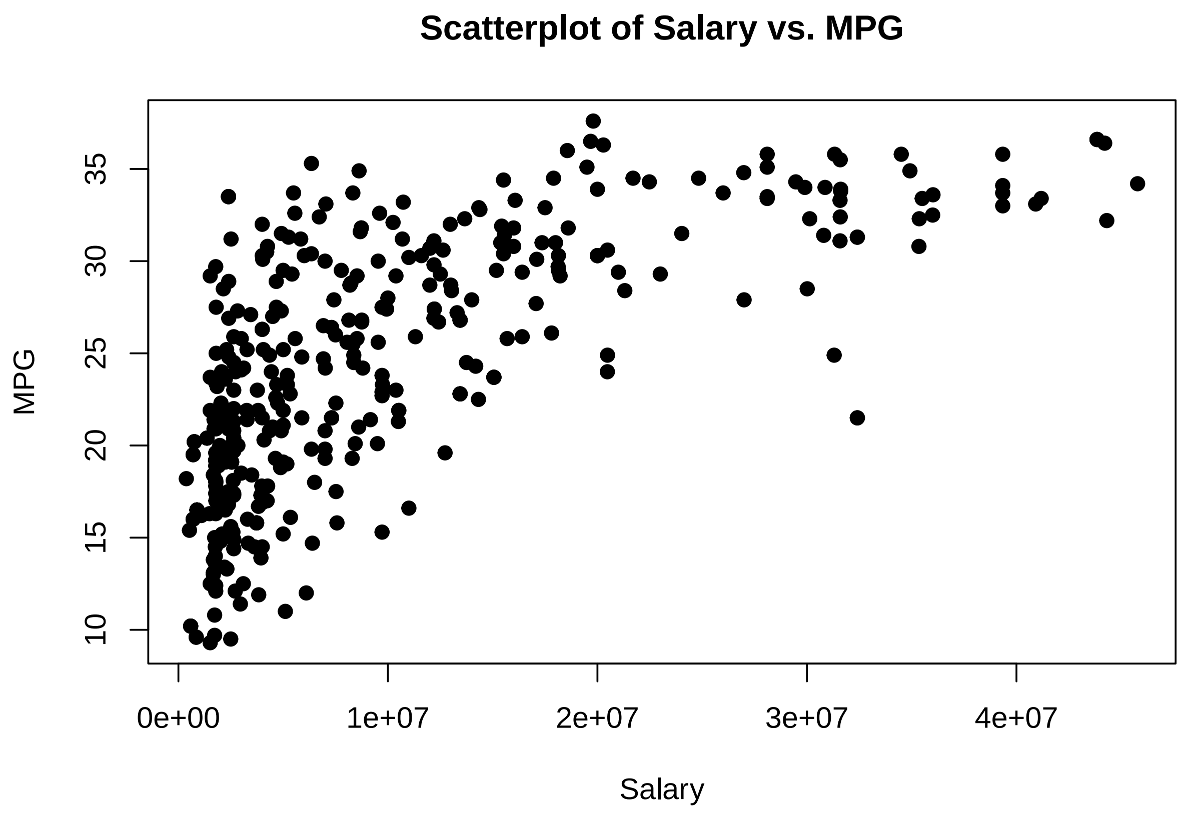
```
plot(our_data$Salary, our_data$PPG, main="Scatterplot of Salary vs. PPG",
   xlab="Salary", ylab="PPG", pch=19)
```



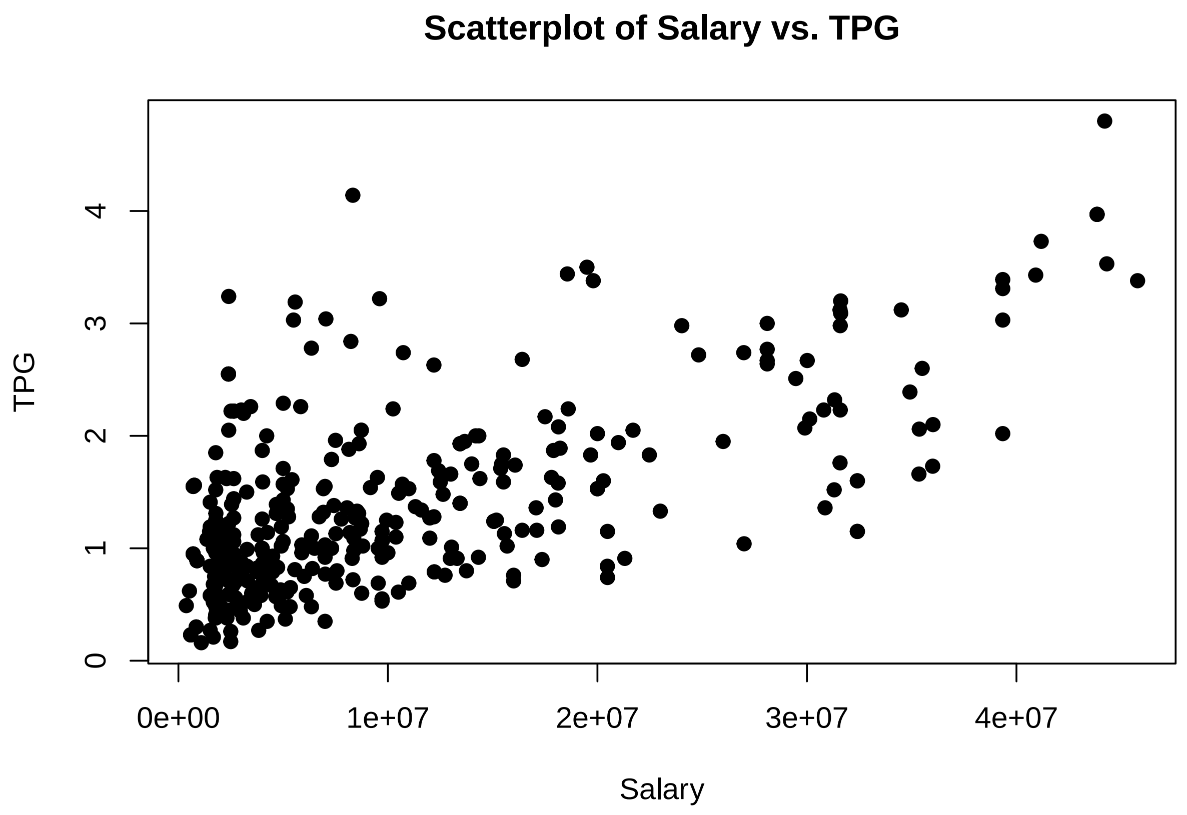
```
plot(our_data$Salary, our_data$APG, main="Scatterplot of Salary vs. APG",
   xlab="Salary", ylab="APG", pch=19)
```



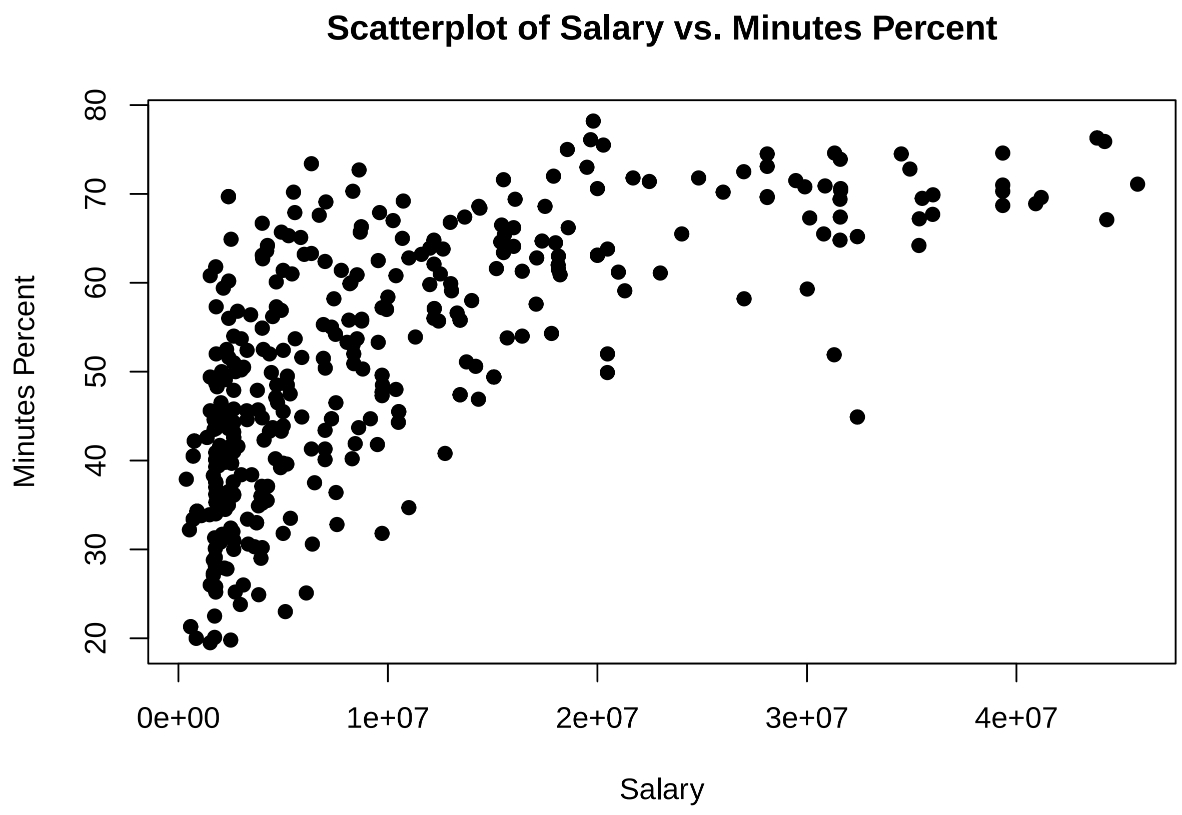
```r
plot(our_data$Salary, our_data$MPG, main="Scatterplot of Salary vs. MPG",
   xlab="Salary", ylab="MPG", pch=19)
```



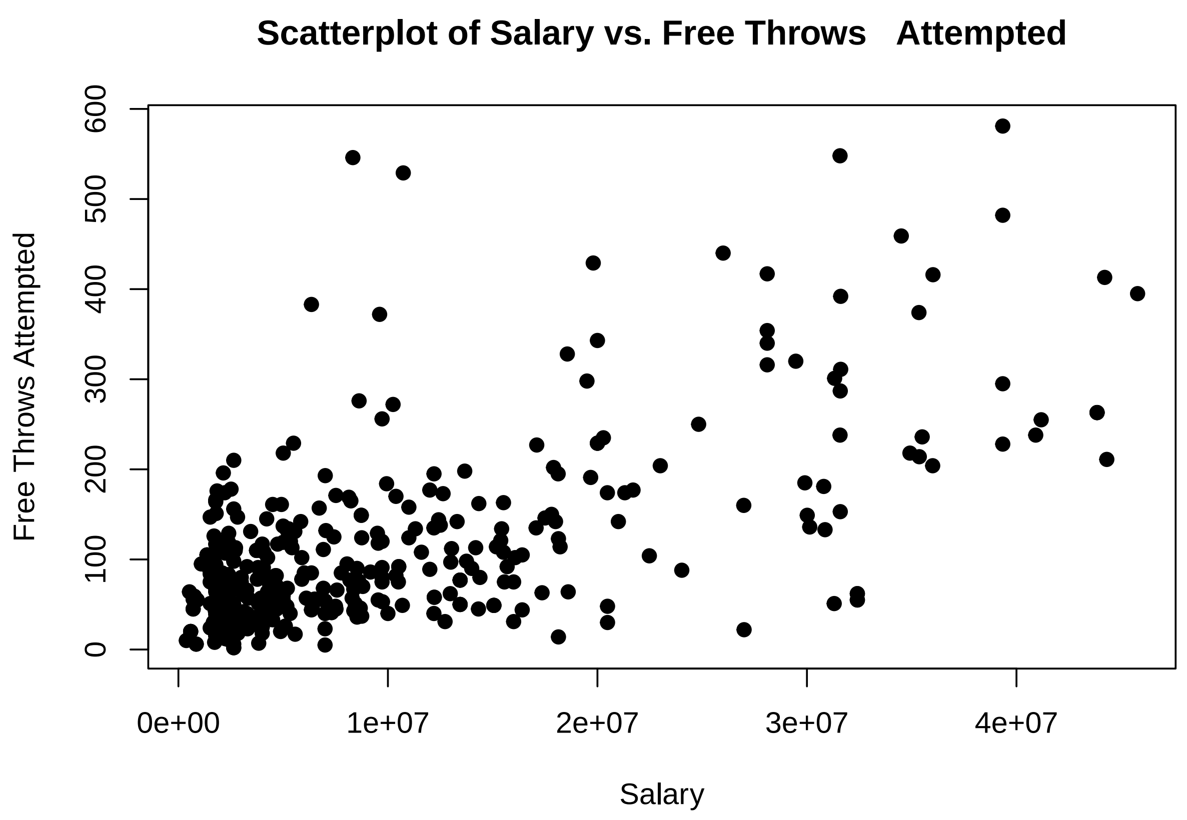
```
plot(our_data$Salary, our_data$TPG, main="Scatterplot of Salary vs. TPG",
   xlab="Salary", ylab="TPG", pch=19)
```



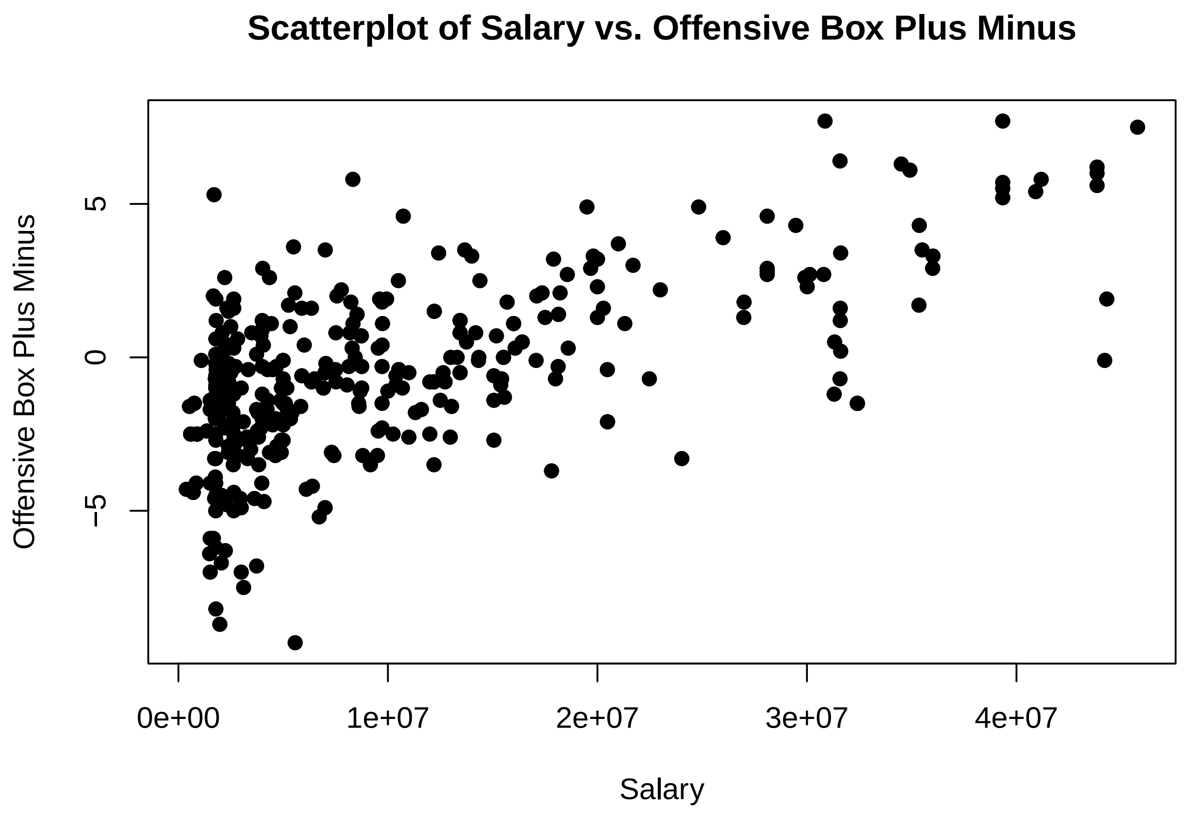
```
plot(our_data$Salary, our_data$Minutes_percent, main="Scatterplot of Salary vs. Minutes Percent",
    xlab="Salary", ylab="Minutes Percent", pch=19)
```



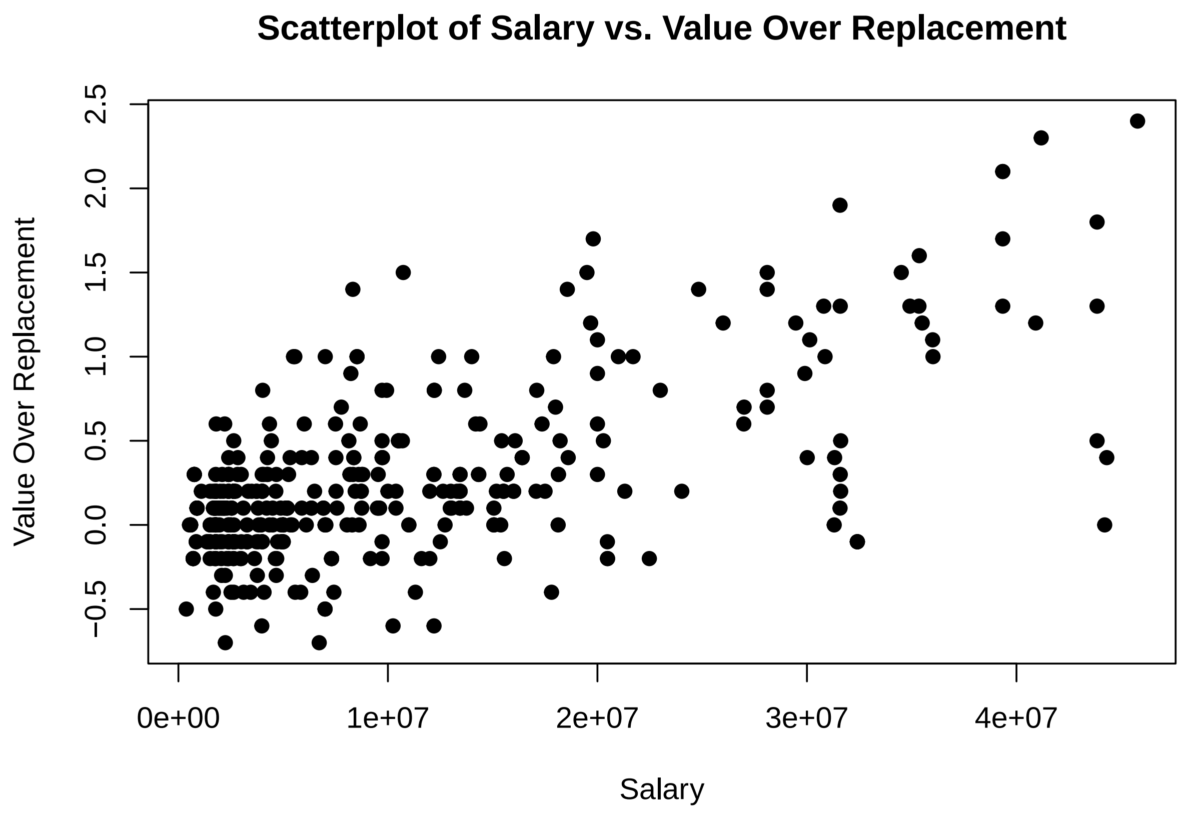
```
plot(our_data$Salary, our_data$free_throws_attempted, main="Scatterplot of Salary vs. Free Throws   Att
   xlab="Salary", ylab="Free Throws Attempted", pch=19)
```



```
plot(our_data$Salary, our_data$Offensive_Box_Plus_Minus, main="Scatterplot of Salary vs. Offensive Box 
   xlab="Salary", ylab="Offensive Box Plus Minus", pch=19)
```



```
plot(our_data$Salary, our_data$Value_Over_Replacement, main="Scatterplot of Salary vs. Value Over Repla
   xlab="Salary", ylab="Value Over Replacement", pch=19)
```



Training and Testing data

In our models we need testing and training datas. Training data will be 80% and testing will be 20%. Here we define them and will proceed with these in mind.

```
set.seed(3112022)
fit_data <- our_data[-1]
new_data <- resample_partition(fit_data, p = c(test=0.2, train=0.8))
training <- as.data.frame(new_data$train)
testing <- as.data.frame(new_data$test)
```

Exploratory Data Analysis

Our linear regression, logistic regression, ridge and lasso, regression decision tree, and random forest use the same training dataset, which contains 309 observations; and the same testing dataset, which contains 77 observations, 386 players in total.

Linear regression

Our first model is linear regression. There were a total of 3 fitted models. The first model consists of all 8 predictors. Based from the summary of that first fit, the second fit contains only predictors that were significant. Since the first and second fit were closely aligned, the third fit looked at the summary of the first fit and chose more signicant predictors than the second fit.

```
fit1 <- lm(Salary ~ PPG + APG + MPG + TPG + Minutes_percent + free_throws_attempted + Offensive_Box_Plus
summary(fit1)
```

```
##
## Call:
## lm(formula = Salary ~ PPG + APG + MPG + TPG + Minutes_percent +
##     free_throws_attempted + Offensive_Box_Plus_Minus + Value_Over_Replacement,
##     data = training)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -19238393  -3666432   -320330   2562992  25441992
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -5323369    1674227   -3.18   0.0016 **
## PPG                        454929     179858    2.53   0.0119 *
## APG                       1107092     355792    3.11   0.0020 **
## MPG                      -7948717   10760389   -0.74   0.4607
## TPG                        191955    1128868    0.17   0.8651
## Minutes_percent           3918259    5168273    0.76   0.4490
## free_throws_attempted        8874       6805    1.30   0.1932
## Offensive_Box_Plus_Minus   193023     263729    0.73   0.4648
## Value_Over_Replacement    3707858    1405745    2.64   0.0088 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6300000 on 300 degrees of freedom
## Multiple R-squared:  0.618,  Adjusted R-squared:  0.608
## F-statistic: 60.7 on 8 and 300 DF,  p-value: <2e-16
```

Here is the respective training and testing mean squared error for fit1 (all predictors)

```
train.predict1 <- predict(fit1, training)
test.predict1 <- predict(fit1, testing)
mean((train.predict1-training$Salary)^2)
```

```
## [1] 3.859e+13
```

```
mean((test.predict1-testing$Salary)^2)
```

```
## [1] 5.39e+13
```

Here is the fit2 model, which takes predictors that are significant from fit1, as shown in the summary of fit1

```
fit2 <- lm(Salary ~ PPG + APG + Value_Over_Replacement, data= training)
summary(fit2)
```

```
##
## Call:
## lm(formula = Salary ~ PPG + APG + Value_Over_Replacement, data = training)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -18291447  -3697069   -371688   2935890  25291063
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -3240037     857052   -3.78  0.00019 ***
## PPG                      782251      92103    8.49  9.0e-16 ***
## APG                     1237489     248849    4.97  1.1e-06 ***
## Value_Over_Replacement  4376932    1020652    4.29  2.4e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6320000 on 305 degrees of freedom
## Multiple R-squared:  0.61,   Adjusted R-squared:  0.606
## F-statistic:  159 on 3 and 305 DF,  p-value: <2e-16
```

Here is the training and testing mean squared error respectively for fit2

```
train.predict2 <- predict(fit2, training)
test.predict2 <- predict(fit2, testing)
mean((train.predict2-training$Salary)^2)
```

```
## [1] 3.94e+13
```

```
mean((test.predict2-testing$Salary)^2)
```

```
## [1] 5.555e+13
```

Here is the fit3 model, which takes predictors that are more significant than fit2 from the summary in fit1.

```
fit3 <- lm(Salary ~ APG + Value_Over_Replacement, data= fit_data)
summary(fit3)
```

```
##
## Call:
## lm(formula = Salary ~ APG + Value_Over_Replacement, data = fit_data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -26880280  -4606166  -1323534   3243761  26048134
##
## Coefficients:
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)             2075769     619850    3.35  0.00089 ***
## APG                     2164600     214602   10.09  < 2e-16 ***
## Value_Over_Replacement  9131247     850494   10.74  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7200000 on 383 degrees of freedom
## Multiple R-squared:  0.527,  Adjusted R-squared:  0.525
## F-statistic:  213 on 2 and 383 DF,  p-value: <2e-16
```

Here is the training and testing mean squared error respectively for fit3

```r
train.predict3 <- predict(fit3, training)
test.predict3 <- predict(fit3, testing)
mean((train.predict3-training$Salary)^2)
```

```
## [1] 4.873e+13
```

```r
mean((test.predict3-testing$Salary)^2)
```

```
## [1] 6.215e+13
```

Based from the 3 fitted models, I chose the model with the least MSE for training and testing. The first model had the least MSE for both training and testing so I will use fit1. This makes sense as the largest R-squared was fit1.

Logistic regression

Since all the nine variables we are going to use are numeric variables, and the logistic regression cannot be used to analyze numeric variables, we will create a new predictor called "salarygreater". "Salarygreater" tests whether a player's salary is greater than the average salary of the league. By producing this new binomial predictor, we can use logistic regression to analyze the salary condition of the players in a new perspective.

First, we will calculate the average salary of all players, and create the new binary variable "salarygreater" in the training dataset to test whether a certain player's salary is greater than the average. If the player's salary is greater than the average, it will show "1" in the "salarygreater"; if not, it will show "0".

```r
mean_train_salary <- mean(training$Salary)
training_salarymean<- training %>% mutate(salarygreater=factor(ifelse(Salary >= mean_train_salary, 1, 0
```

Next, we will fit a logistic regression on the "salarygreater" in the training dataset. After that, we will predict based on the "majority rule": if the predicted probability is greater than 0.5, classify the observation as 1, otherwise, classify it as 0.

```r
training_logit <- glm(salarygreater ~ PPG + APG + MPG + TPG + Minutes_percent + free_throws_attempted +

salarymean_pred_training <- predict(training_logit, training_salarymean, type="response")
train_maj_rule <- ifelse(salarymean_pred_training > 0.5, 1,0)
```

After that, we will define a function called "calc_error_rate" and use it to calculate the error rate of the of the logistic model on the training dataset, which is created above.

```r
calc_error_rate <- function(predicted.value, true.value){
  return(mean(true.value!=predicted.value))}
calc_error_rate(train_maj_rule, training_salarymean$salarygreater)
```

```
## [1] 0.1942
```

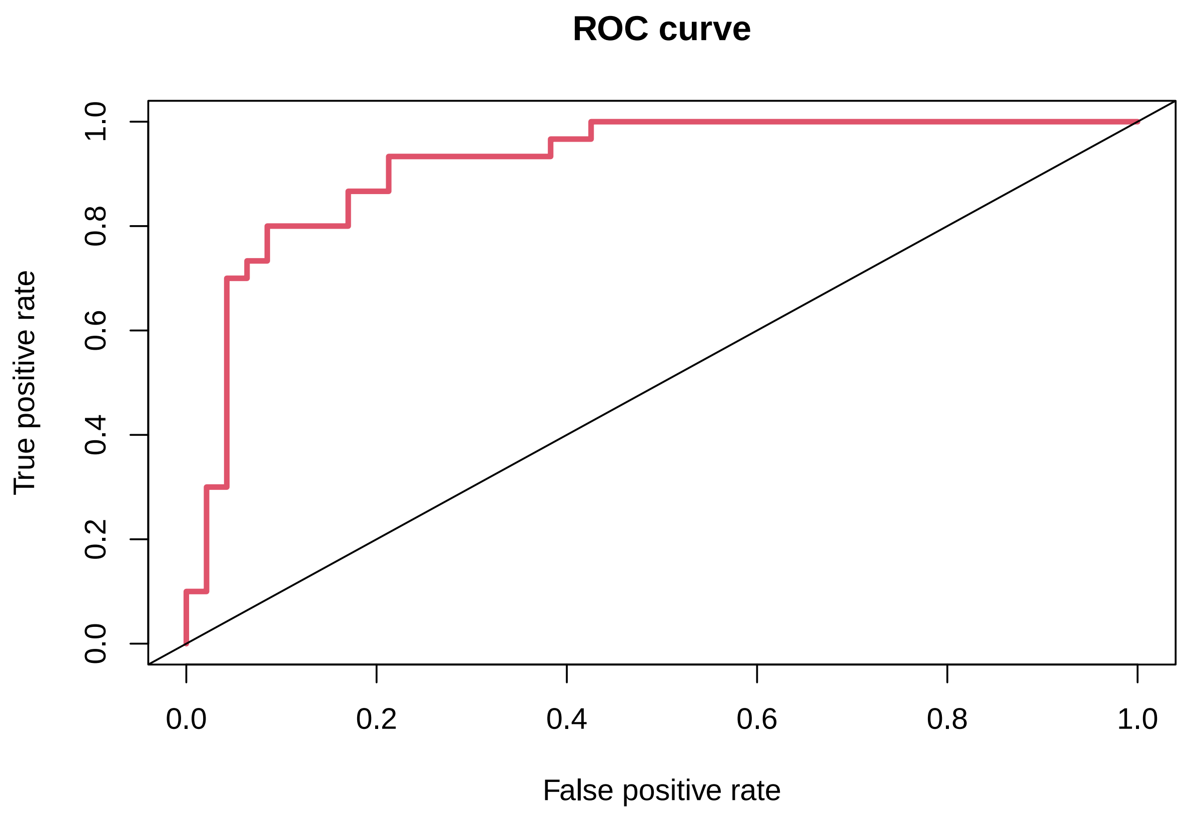
Similary, we will do the same process on the testing dataset to predict based on the logistic model created above and calculate the test error.

```r
testing_salarymean<- testing %>% mutate(salarygreater=factor(ifelse(Salary >= mean_train_salary, 1, 0),
salarymean_pred_testing <- predict(training_logit,  testing_salarymean, type="response")
test_maj_rule2 <- ifelse(salarymean_pred_testing > 0.5, 1,0)
calc_error_rate(test_maj_rule2, testing_salarymean$salarygreater)
```

```
## [1] 0.1558
```

Finally, we can plot an ROC curve and calculate the area under the curve (AUC) for the test data to see the performance of the logistic regression model.

```r
pred <- prediction(salarymean_pred_testing, testing_salarymean$salarygreater)
perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf, col = 2, lwd = 3, main = "ROC curve")
abline(0, 1)
```



AUC is shown below.

```
auc <- performance(pred, "auc")@y.values[[1]]
auc
```

```
## [1] 0.9184
```

Training, Testing for Ridge/ Lasso

I chose to do ridge & lasso because it eliminates overfitting through regularization. To start, I defined the training and testing sets for this process. In each training and testing, I had to split as x.train and y train as well as x.test and y.test. The x in front means all predictors except the response variable (Salary). The y in front signifies the response (Salary)

```
x <- model.matrix(Salary ~ ., data= fit_data)
y <- fit_data$Salary

x.train <- as.matrix(training[,-1])
y.train <- as.matrix(training$Salary)
x.test <- as.matrix(testing[,-1])
y.test <- as.matrix(testing$Salary)
```

Ridge / Lasso

Ridge regression

This part is dedicated for the ridge regression. I first defined the list of lambda values the model can take, and used 5-folds cross-validation on the training data to predict the coefficients of each predictor. This represents the distance of each predictor to the response. The lower the distance, the better.

```
lambda.list.ridge = 1000 * exp(seq(0, log(1e-5), length = 100))

ridge.mod = cv.glmnet(x.train, y.train, alpha=0,lambda=lambda.list.ridge, nfolds=5)

ridge.pred_1 = predict(ridge.mod, s = ridge.mod$lambda.min, type="coefficients", newx=x.test)
```

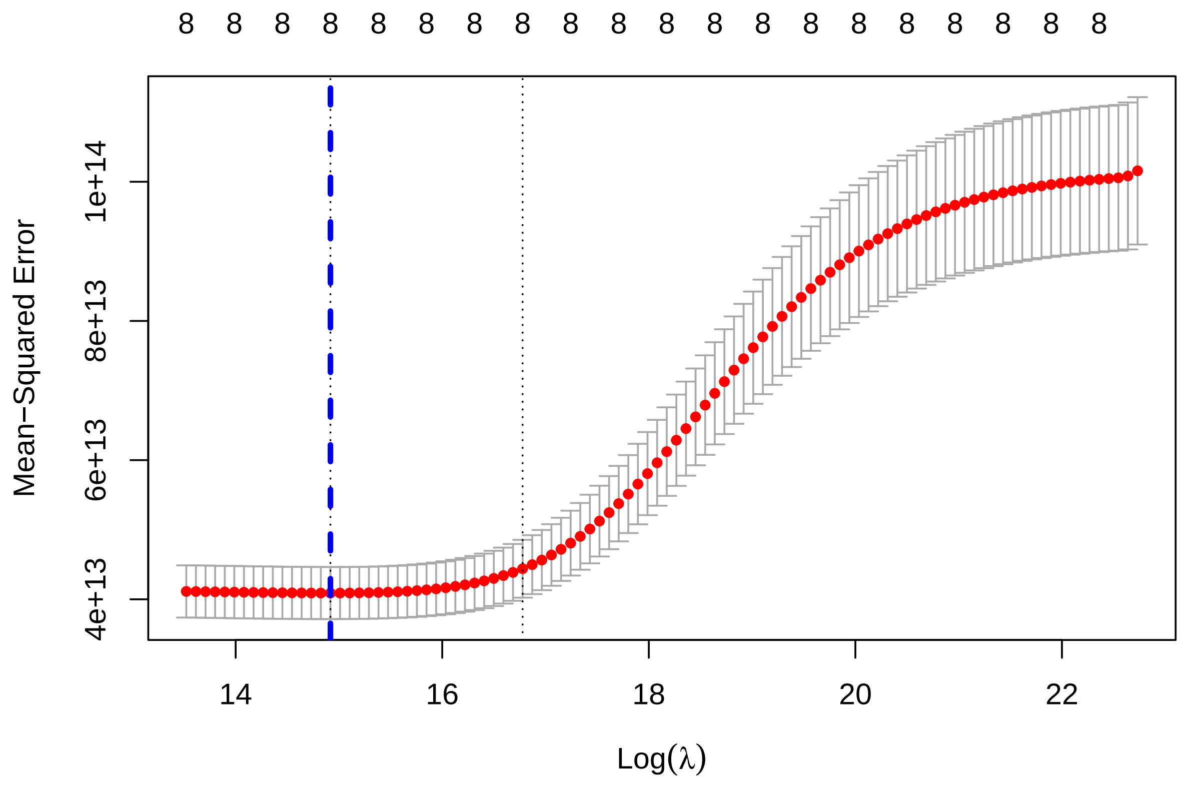
Then, I found the mean squared error for this ridge model.

```
ridge.pred=predict(ridge.mod, s = ridge.mod$lambda.min, newx=x.test)
mean((ridge.pred-y.test)^2)
```

```
## [1] 5.408e+13
```

Next, I used cross-validation to choose the best tuning parameter.

```
set.seed(3142022)
cv.out.ridge = cv.glmnet(x.train, y.train, alpha= 0)
plot(cv.out.ridge)
abline(v=log(cv.out.ridge$lambda.min), col = "blue", lwd=3, lty=2)
```



```
bestlam = cv.out.ridge$lambda.min
bestlam
```

```
## [1] 3010560
```

Here is the mean squared error for best lambda.

```
ridge.pred=predict(ridge.mod, s = bestlam, newx=x.test)
mean((ridge.pred-y.test)^2)
```

```
## [1] 5.408e+13
```

This prints out the coefficients of the best lambda. The glmnet function selects the minimum lambda and that lambda is used to predict the coefficients.

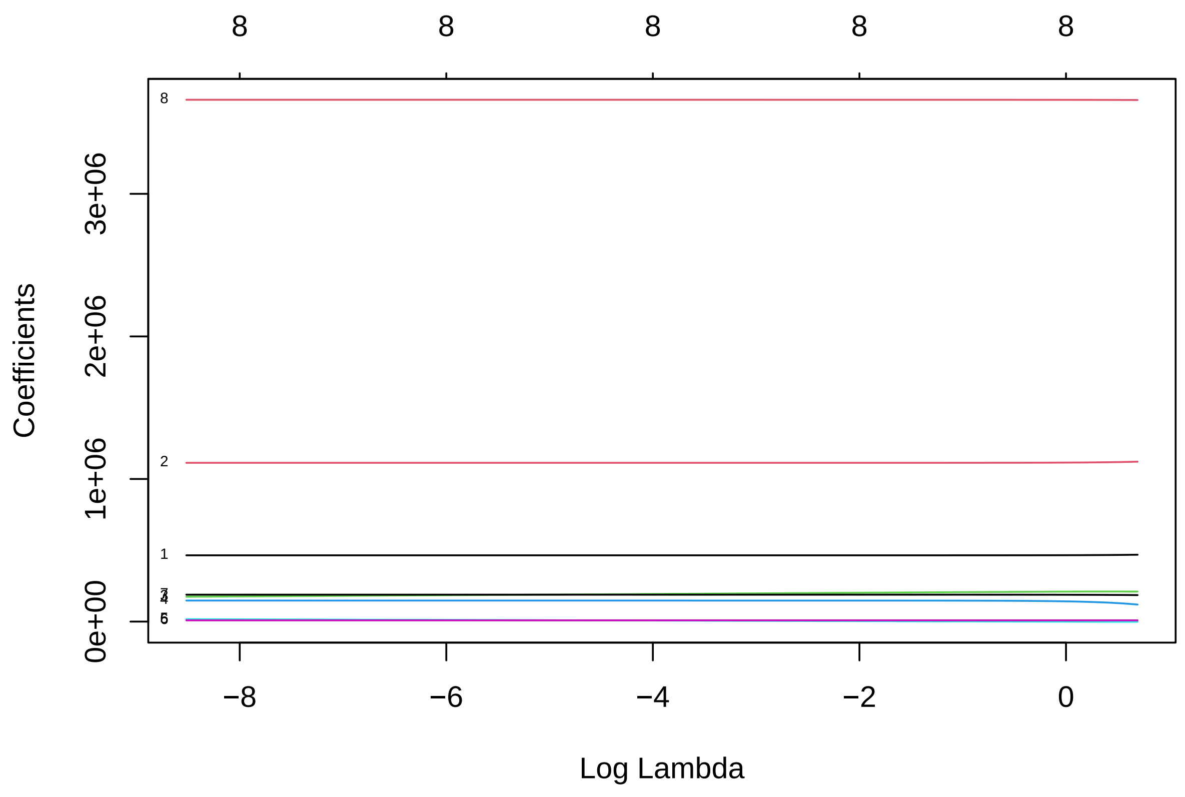
```
out = glmnet(x,y,alpha=0)
predict(out, type="coefficients", s=bestlam)
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                                  s1
## (Intercept)                -4562064
## (Intercept)                       .
## PPG                          241527
## APG                           758927
## MPG                           146306
## TPG                           843915
## Minutes_percent                71181
## free_throws_attempted           9386
## Offensive_Box_Plus_Minus      425212
## Value_Over_Replacement       2959665
```

lasso regression

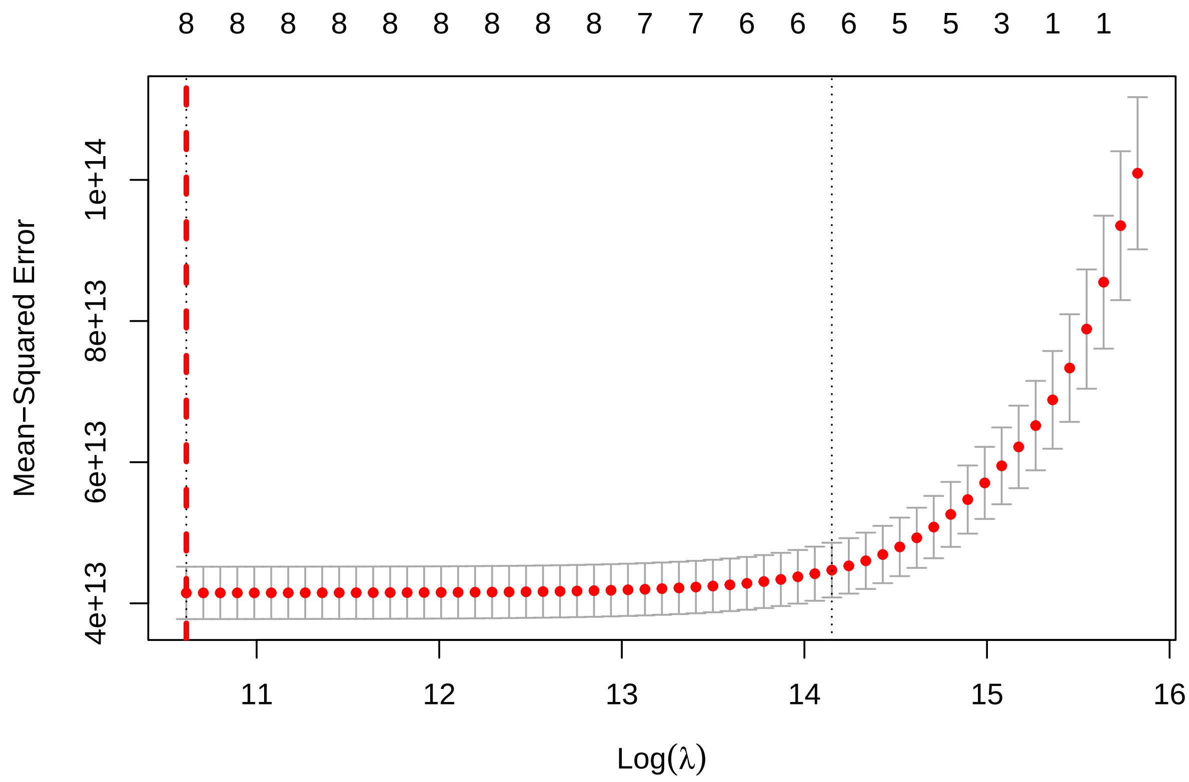
Next, I fit a lasso model using the glmnet function with alpha as 1 with 5 folds cross validation. I plotted the model to see if coefficients can be zero.

```
set.seed(3142022)
lambda.list.lasso = 2 * exp(seq(0, log(1e-4), length = 100))
lasso.mod <- glmnet(x.train, y.train, alpha=1, lambda=lambda.list.lasso, nfolds = 5)
plot(lasso.mod, xvar="lambda", label=TRUE)
```



In this step, I performed cross validation and computed test error.

```
cv.out.lasso = cv.glmnet(x.train,y.train,alpha=1)
plot(cv.out.lasso)
abline(v=log(cv.out.lasso$lambda.min), col="red", lwd=3, lty=2)
```



```
bestlam2 = cv.out.lasso$lambda.min
lasso.pred = predict(lasso.mod, s = bestlam2, newx = x.test)
mean((lasso.pred-y.test)^2)
```

```
## [1] 5.408e+13
```

Like the ridge regression, I used this step to print out the coefficients of the best lambda. The glmnet function selects the minimum lambda and that lambda is used to predict the coefficients of the lasso model.

```
out = glmnet(x, y, alpha=1, lambda=lambda.list.lasso)
lasso.coef = predict(out, type="coefficients", s=bestlam)
lasso.coef
```

```
## 10 x 1 sparse Matrix of class "dgCMatrix"
##                                  s1
## (Intercept)                -5063779
## (Intercept)                       .
## PPG                          469084
## APG                         1266036
## MPG                          160656
## TPG                         -882401
## Minutes_percent               39006
## free_throws_attempted          8341
## Offensive_Box_Plus_Minus     225082
## Value_Over_Replacement      3852047
```

Regression Decision Tree:

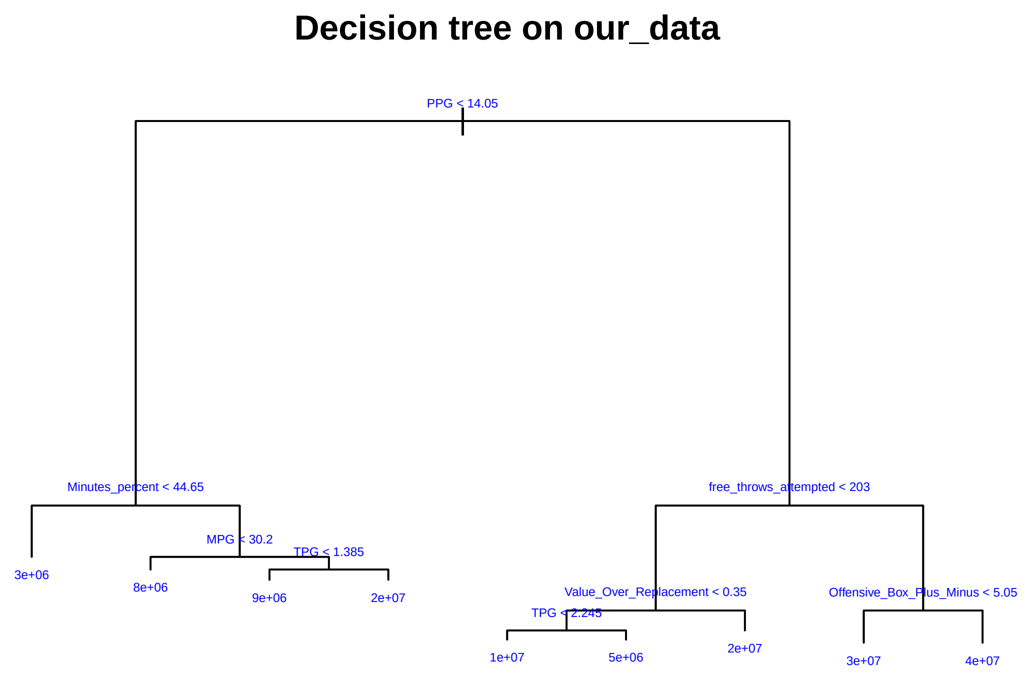
Here, we start by using the tree() function to create a decision tree using our_data. Then we take a summary of this tree to see variables used in tree construction, number of terminal nodes, and residual information.

```
tree_our_data=tree(Salary~. , data = our_data)
summary(tree_our_data)
```

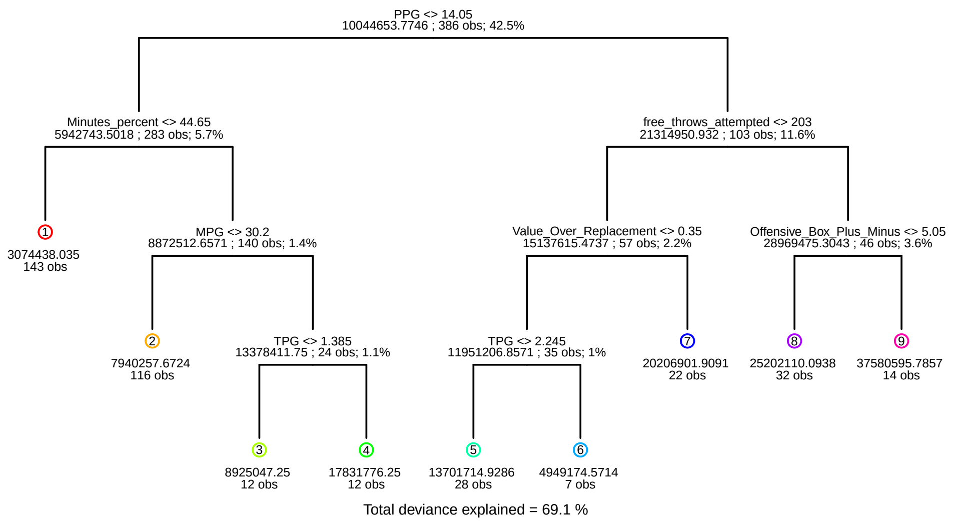
```
##
## Regression tree:
## tree(formula = Salary ~ ., data = our_data)
## Variables actually used in tree construction:
## [1] "PPG"                        "Minutes_percent"
## [3] "MPG"                        "TPG"
## [5] "free_throws_attempted"    "Value_Over_Replacement"
## [7] "Offensive_Box_Plus_Minus"
## Number of terminal nodes:  9
## Residual mean deviance:  3.44e+13 = 1.3e+16 / 377
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -29300000  -2560000   -652000         0   2050000  24500000
```

Here we plot our decision tree using two different tree plotting functions, plot and draw tree. Both trees show the same idea and have 12 terminal nodes.

```
# plot the fitted tree
plot(tree_our_data)
text(tree_our_data, pretty = 0, cex = .4, col = "blue")
title("Decision tree on our_data", cex = 0.8)
```

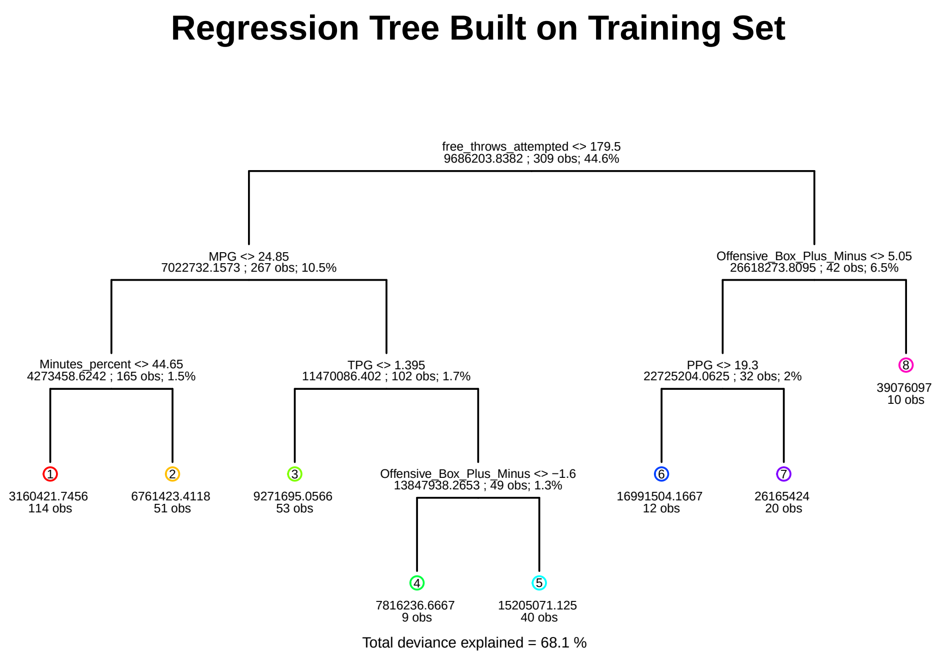

Decision tree on our_data
PPG < 14.05
Minutes_percent < 44.65
free_throws_attempted < 203
MPG < 30.2
TPG < 1.385
Value_Over_Replacement < 0.35
Offensive_Box_Plus_Minus < 5.05
TPG < 2.245
3e+06
8e+06
9e+06
2e+07
1e+07
5e+06
2e+07
3e+07
4e+07

```r
draw.tree(tree_our_data, nodeinfo=TRUE, cex = 0.4)
```


PPG <> 14.05
10044653.7746 ; 386 obs; 42.5%
Minutes_percent <> 44.65
5942743.5018 ; 283 obs; 5.7%
free_throws_attempted <> 203
21314950.932 ; 103 obs; 11.6%
1
3074438.035
143 obs
MPG <> 30.2
8872512.6571 ; 140 obs; 1.4%
Value_Over_Replacement <> 0.35
15137615.4737 ; 57 obs; 2.2%
Offensive_Box_Plus_Minus <> 5.05
28969475.3043 ; 46 obs; 3.6%
2
7940257.6724
116 obs
TPG <> 1.385
13378411.75 ; 24 obs; 1.1%
TPG <> 2.245
11951206.8571 ; 35 obs; 1%
7
20206901.9091
22 obs
8
25202110.0938
32 obs
9
37580595.7857
14 obs
3
8925047.25
12 obs
4
17831776.25
12 obs
5
13701714.9286
28 obs
6
4949174.5714
7 obs
Total deviance explained = 69.1 %

Furthermore, we fit the regression decision tree model to the training set and plot the tree using the draw.tree() command.

```r
# Fit model on training set
tree.nba = tree(Salary~. , data = training)

# Plot the tree
draw.tree(tree.nba, nodeinfo=TRUE, cex = 0.4)
title("Regression Tree Built on Training Set")
```


Regression Tree Built on Training Set
free_throws_attempted <> 179.5
9686203.8382 ; 309 obs; 44.6%
MPG <> 24.85
7022732.1573 ; 267 obs; 10.5%
Offensive_Box_Plus_Minus <> 5.05
26618273.8095 ; 42 obs; 6.5%
Minutes_percent <> 44.65
4273458.6242 ; 165 obs; 1.5%
TPG <> 1.395
11470086.402 ; 102 obs; 1.7%
PPG <> 19.3
22725204.0625 ; 32 obs; 2%
8
39076097
10 obs
1
3160421.7456
114 obs
2
6761423.4118
51 obs
3
9271695.0566
53 obs
Offensive_Box_Plus_Minus <> −1.6
13847938.2653 ; 49 obs; 1.3%
6
16991504.1667
12 obs
7
26165424
20 obs
4
7816236.6667
9 obs
5
15205071.125
40 obs
Total deviance explained = 68.1 %

We can see in the summary that the tree modeled by the training set has 10 terminal nodes, whereas the previous tree had 12 terminal nodes.

```
summary(tree.nba)
```

```
##
## Regression tree:
## tree(formula = Salary ~ ., data = training)
## Variables actually used in tree construction:
## [1] "free_throws_attempted"    "MPG"
## [3] "Minutes_percent"          "TPG"
## [5] "Offensive_Box_Plus_Minus" "PPG"
## Number of terminal nodes:  8
## Residual mean deviance:  3.31e+13 = 9.96e+15 / 301
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -19800000  -2570000   -771000         0   2190000  25600000
```

Now here we do a prediction on the test set using the predict() command. We specify type='vector' because we are working with a regression decision tree. Then we calculate the mean squared error which is 6.68e+13 and root mean squared error which is 8173153.

```
# Predict on test set
tree.pred = predict(tree.nba, testing, type="vector")
```

```
# mean squared error
y = mean((tree.pred-testing$Salary)^2)
y
```

```
## [1] 7.287e+13
```

```
# root mean squared error
z=sqrt(y)
z
```

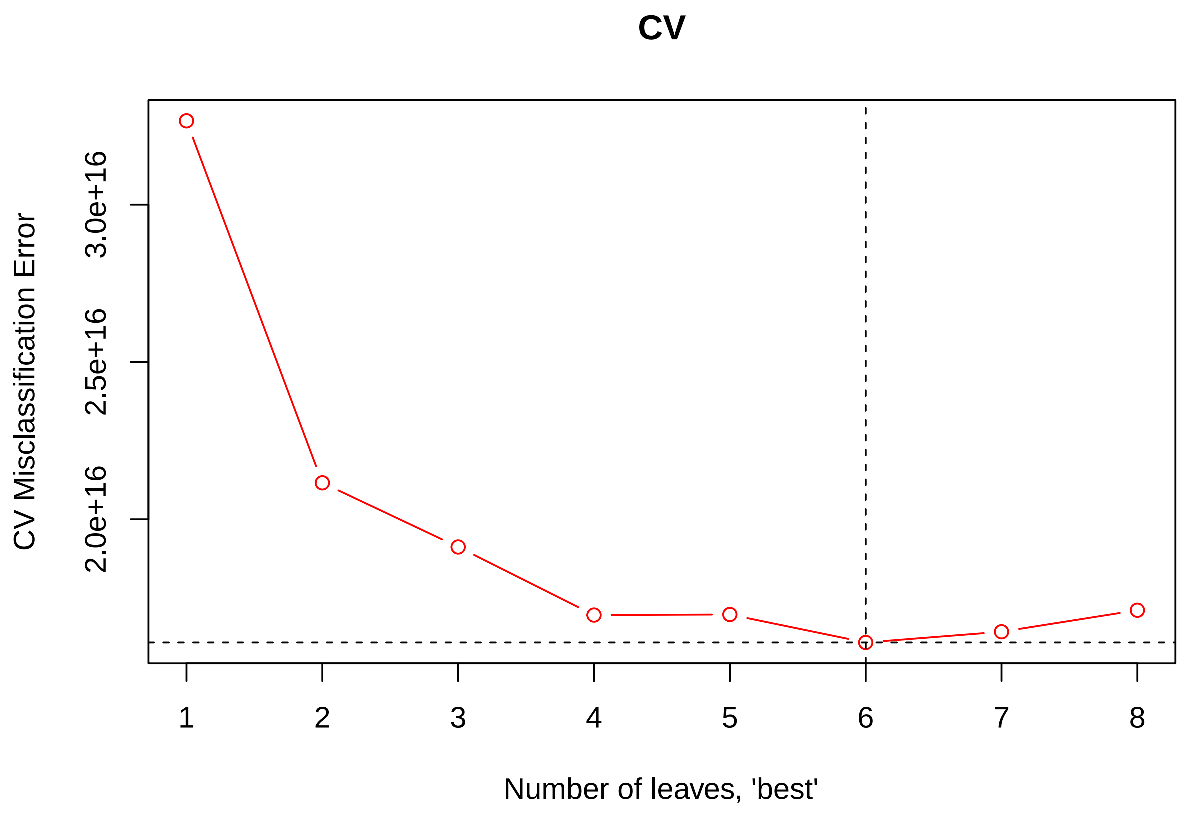
```
## [1] 8536683
```

To calculate the best number of terminal nodes we do a 10-fold cross validation. We plot size versus cross-validation error rate and add a vertical line at the minimum error. From this we can see 9 is the ideal number of terminal nodes, because the cross-validation misclassification error is the lowest at that point.

```
set.seed(3142022)

#K=10-Fold cross validation
cv = cv.tree(tree.nba, K=10)

# the tree with smaller size
best.cv = min(cv$size[cv$dev == min(cv$dev)])
best.cv
```

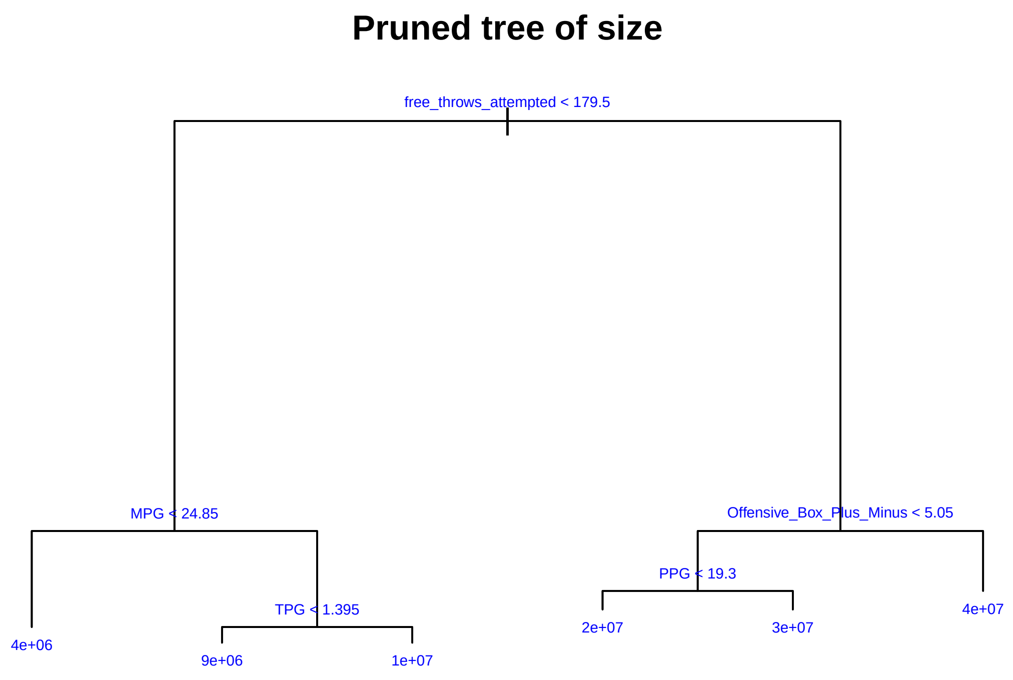
```
## [1] 6
```

```
# Plot size vs. cross-validation error rate
 plot(cv$size , cv$dev, type="b", xlab = "Number of leaves, \'best\'",
      ylab = "CV Misclassification Error", col = "red", main="CV")
 abline(v=best.cv, lty=2)

# Add lines to identify complexity parameter
min.error = which.min(cv$dev) # Get minimum error index
abline(h = cv$dev[min.error],lty = 2)
```



We can now plot the pruned tree with 9 terminal nodes. The mean squared error (MSE) is lowest when the number of terminal nodes is 9.

```
# Prune tree
pt.cv = prune.tree(tree.nba, best=best.cv)

# # Plot pruned tree
plot(pt.cv)
text(pt.cv, pretty=0, col = "blue", cex = .5)
title("Pruned tree of size")
```



```r
# Predict on test set
pred.pt.cv = predict(pt.cv, testing, type="vector")

# mean squared error
w = mean((pred.pt.cv-testing$Salary)^2)
w
```

```
## [1] 8.291e+13
```

```r
# root mean squared error
g=sqrt(w)
g
```

```
## [1] 9105549
```

The mean squared error is lower (8100514) than that of the non-pruned tree; the pruned tree with 9 terminal nodes is a better model.

random forest

First, we will fit a random forest model on the “Salarygreater” variable whichis whether a player’s salary is greater than the average salary of the league.

```
rf_our_data = randomForest(salarygreater ~ PPG + APG + MPG + TPG + Minutes_percent + free_throws_attemp
rf_our_data
```

```
##
## Call:
##  randomForest(formula = salarygreater ~ PPG + APG + MPG + TPG +      Minutes_percent + free_throws_a
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 18.12%
## Confusion matrix:
##     0  1 class.error
## 0 174 27      0.1343
## 1  29 79      0.2685
```

Next, we will check the error rate on the test dataset.

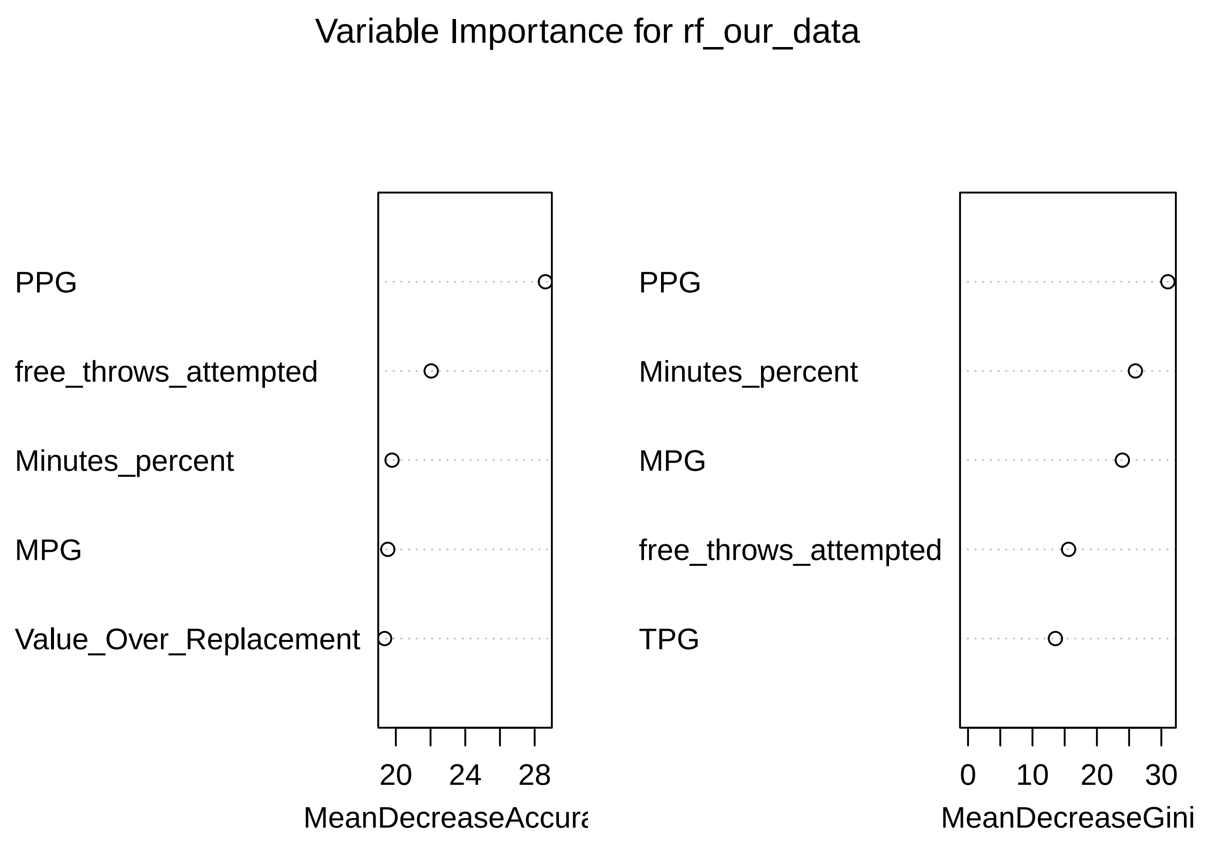
```
yhat.rf = predict (rf_our_data, newdata = testing_salarymean)
# Confusion matrix
rf.err = table(pred = yhat.rf, truth = testing_salarymean$salarygreater)
test.rf.err = 1 - sum(diag(rf.err))/sum(rf.err)
test.rf.err
```

```
## [1] 0.1948
```

The test set error rate is 0.1818.

Then we will get a plot with decreasing order of importance based on Model Accuracy and Gini value.

```
varImpPlot(rf_our_data, sort=T, main="Variable Importance for rf_our_data", n.var=5)
```



From the graphs we can see that among all the trees in the random forest, PPG is the most important variable in terms of Model Accuracy and Gini index.

PCA

Next, I used PCA to find the most influential predictors in the model. I first created dataset called dat that includes all predictors and found their variances. Using Principle component Analysis, I found center(mean), scale (standard deviation), rotation (principle component loadings), and x(matrix of all principle component vectors)

```
dat <- our_data %>% select(-1)
summary(dat)
```

```
##      Salary              PPG            APG              MPG
##  Min.   :  377645   Min.   : 1.9   Min.   : 0.20   Min.   : 9.3
##  1st Qu.: 2401537   1st Qu.: 6.9   1st Qu.: 1.30   1st Qu.:19.3
##  Median : 5565202   Median :10.0   Median : 2.05   Median :24.2
##  Mean   :10044654   Mean   :11.4   Mean   : 2.65   Mean   :24.4
##  3rd Qu.:13611280   3rd Qu.:14.2   3rd Qu.: 3.60   3rd Qu.:30.3
##  Max.   :45780966   Max.   :32.0   Max.   :11.70   Max.   :37.6
##       TPG       Minutes_percent free_throws_attempted Offensive_Box_Plus_Minus
##  Min.   :0.16   Min.   :19.5    Min.   :  2           Min.   :-9.30
##  1st Qu.:0.83   1st Qu.:40.1    1st Qu.: 46           1st Qu.:-2.40
##  Median :1.15   Median :50.5    Median : 77           Median :-0.80
##  Mean   :1.37   Mean   :50.8    Mean   :111           Mean   :-0.61
##  3rd Qu.:1.71   3rd Qu.:63.2    3rd Qu.:142           3rd Qu.: 1.10
##  Max.   :4.80   Max.   :78.2    Max.   :581           Max.   : 7.70
##  Value_Over_Replacement
##  Min.   :-0.700
##  1st Qu.:-0.100
##  Median : 0.100
##  Mean   : 0.245
##  3rd Qu.: 0.400
##  Max.   : 2.400
```

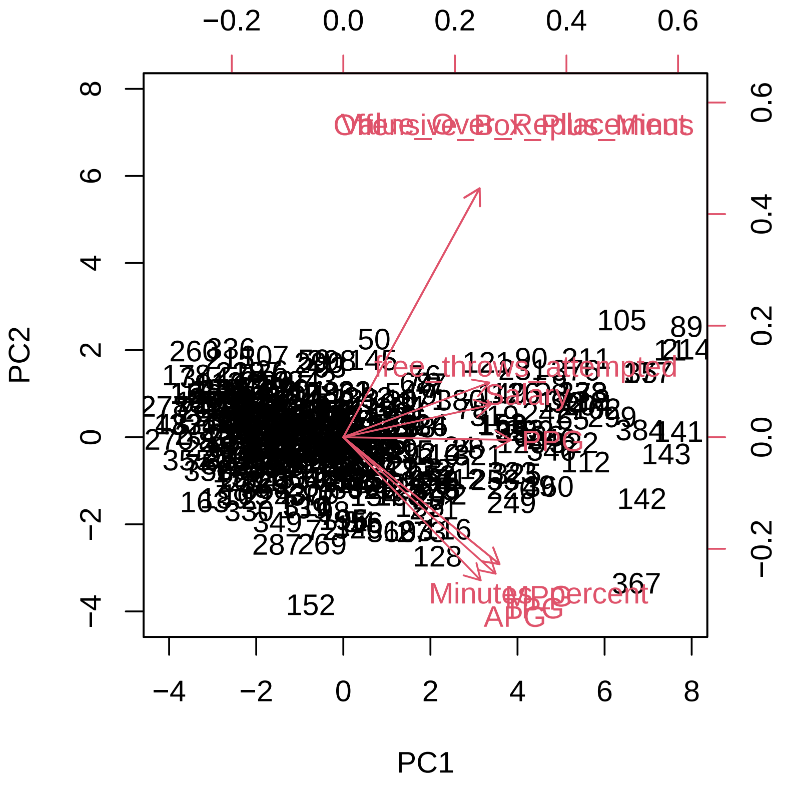
```
apply(dat, 2, var)
```

```
##                   Salary                      PPG                      APG
##                1.090e+14                3.761e+01                3.856e+00
##                      MPG                      TPG          Minutes_percent
##                4.720e+01                6.469e-01                2.047e+02
##    free_throws_attempted Offensive_Box_Plus_Minus   Value_Over_Replacement
##                1.011e+04                7.724e+00                2.455e-01
```

```
pr.out = prcomp(dat, scale = TRUE)
means <- pr.out$center
sds <- pr.out$scale
loads <- pr.out$rotation
all <- pr.out$x
```

I plotted the first and second prinicple components to visualize the data. I then output the standard deviation and variance of of each prinicple component. Last, I found pve, the number of principle components needed.

```
biplot(pr.out, scale = 0)
```



```
pr.out$sdev
```

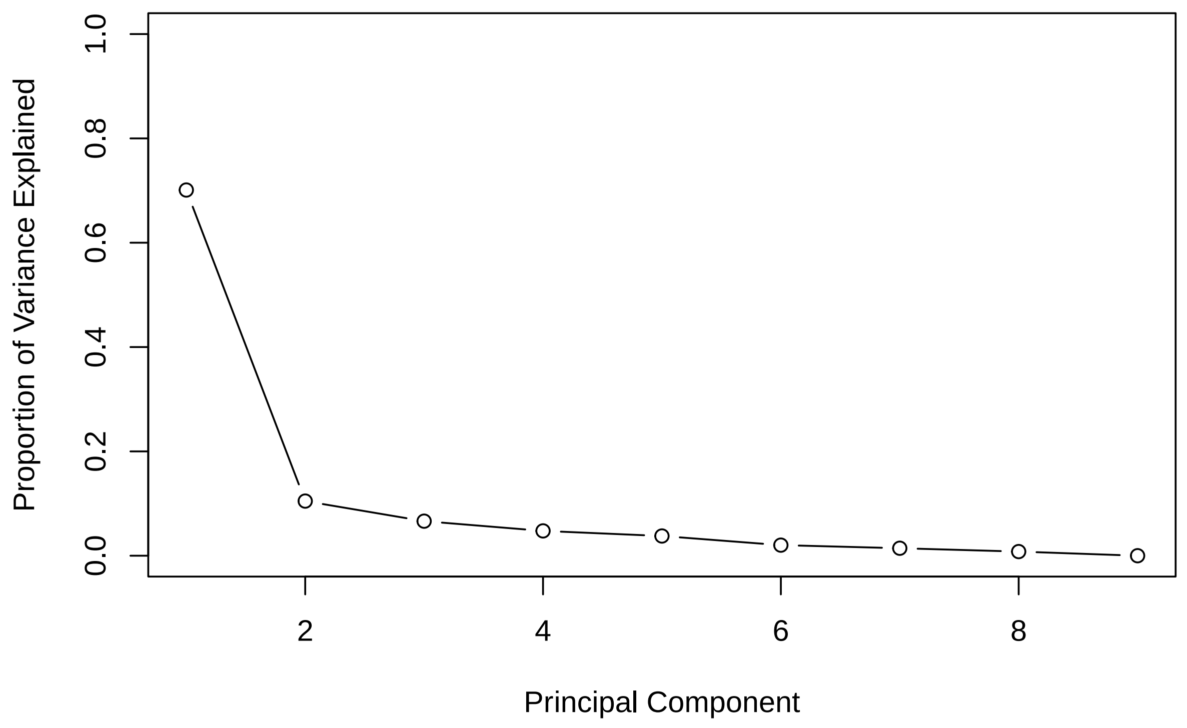
```
## [1] 2.511785 0.970954 0.772104 0.654397 0.583431 0.427380 0.359670 0.267184
## [9] 0.003455
```

```
pr.var = pr.out$sdev^2
pve = pr.var/sum(pr.var)
pve
```

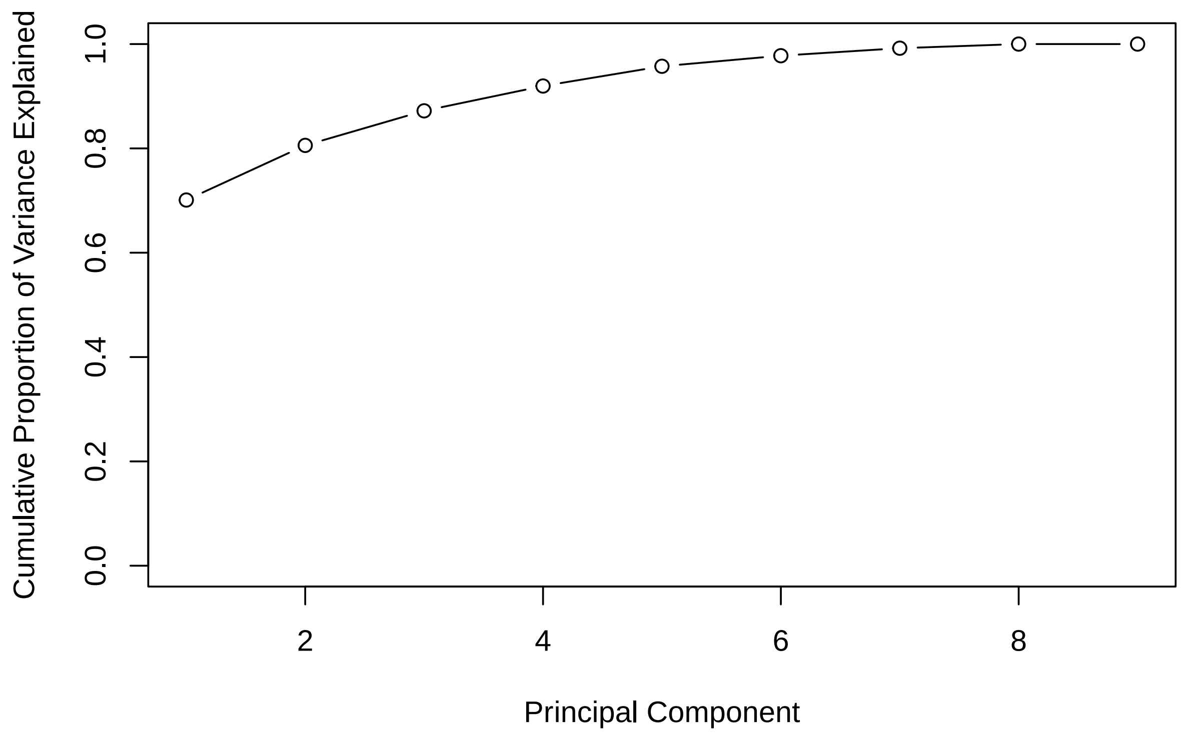
```
## [1] 7.010e-01 1.048e-01 6.624e-02 4.758e-02 3.782e-02 2.029e-02 1.437e-02
## [8] 7.932e-03 1.326e-06
```

I plotted two graphs : the proportion of variance explained and the cumulative proportion of variance explained.

```
plot(pve, xlab = "Principal Component", ylab = "Proportion of Variance Explained", ylim = c(0,1), type =
```



```
plot(cumsum(pve), xlab=  "Principal Component", ylab = "Cumulative Proportion of Variance Explained", yl
```



Based from the plot, the elbow point shows point 2, meaning I should consider the first two PCA because they are the largest 2 values.

```
pr.out$rotation[,1]
```

```
##                   Salary                      PPG                      APG
##                   0.3297                   0.3759                   0.3078
##                      MPG                      TPG          Minutes_percent
##                   0.3501                   0.3414                   0.3500
##    free_throws_attempted Offensive_Box_Plus_Minus   Value_Over_Replacement
##                   0.3273                   0.3054                   0.3053
```

The first principal component explains 70% of the variance in data, the next principal component explains 10.48%. Having greater PPG is the best "indicator" for variance explained in data.

Function to predict

```
# salary_prediction <- function(m, PPG, APG, MPG, TPG, Minutes_percent, free_throws_attempted, Offensiv
#
#   pre_new <- predict(m, data.frame(PPG = points, APG = assists, MPG = minutes, TPG = turnovers, Minut
#
#   msg <- paste("PPG:", points, ",APG:", assists, ",MPG:", minutes, ",TPG:", turnovers, ",Minutes_perc
#
#   print(msg)
# }
#
#
# model <- lm(Salary ~ PPG+ APG + MPG + TPG + Minutes_percent + free_throws_attempted + Offensive_Box_P
# salary_prediction(model, 16, APG = 4, 30, 3, 75, 400, 4.5, 3)


# user.name <- readline(prompt = "Please type your name ")
#   print(paste("Users name is", user.name))
#
#
# install.packages("nlme")
# library(nlme)
#
# salary_prediction <- function(m, PPG, APG){
#
#   pre_new <- predict(m, c(PPG = points, APG = assists))
#
#   msg <- paste("PPG:", points, ",APG", assists, " :Expected Salary: $", as.character(pre_new))
#   print(msg)
# }
#
# model <- lm(Salary ~ PPG+APG, data = our_data)
# salary_prediction(model, 16,3)
```

Conclusion