An Ensemble Model to Predict Student Placement Status

Manyun Zou

```r
# don't show warning text
knitr::opts_chunk$set(warning = FALSE, message = FALSE)

# Loading all the packages here
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

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

```r
library(tidyr)
library(gmodels)
library(ggplot2)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'

## The following object is masked from 'package:tidyr':
##
##     smiths
```

```r
library(psych)
```

```
##
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
library(class)
library(caret)
```

```
## Loading required package: lattice
```

```r
library(rpart) # decision trees
library(C50) # decision trees
library(kernlab) # svm
```

```
##
## Attaching package: 'kernlab'

## The following object is masked from 'package:psych':
##
##     alpha

## The following object is masked from 'package:ggplot2':
##
##     alpha
```

```r
library(randomForest) # random forest
```

```
## randomForest 4.7-1.1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:psych':
##
##     outlier

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
library(irr) # to calculate kappa
```

```
## Loading required package: lpSolve
```

```r
library(mlbench)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following object is masked from 'package:gmodels':
##
##     ci

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

1. Data Acquisition

```r
sheet_id <- "1mo7Q3c42sOJEC5y85NsBeRws4TquCJHS"
placement <- read.csv(sprintf("https://docs.google.com/uc?id=%s&export=download", sheet_id))
str(placement)
```

```
## 'data.frame':    215 obs. of  15 variables:
##  $ sl_no         : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ gender        : chr  "M" "M" "M" "M" ...
##  $ ssc_p         : num  67 79.3 65 56 85.8 ...
##  $ ssc_b         : chr  "Others" "Central" "Central" "Central" ...
##  $ hsc_p         : num  91 78.3 68 52 73.6 ...
##  $ hsc_b         : chr  "Others" "Others" "Central" "Central" ...
##  $ hsc_s         : chr  "Commerce" "Science" "Arts" "Science" ...
##  $ degree_p      : num  58 77.5 64 52 73.3 ...
##  $ degree_t      : chr  "Sci&Tech" "Sci&Tech" "Comm&Mgmt" "Sci&Tech" ...
##  $ workex        : chr  "No" "Yes" "No" "No" ...
##  $ etest_p       : num  55 86.5 75 66 96.8 ...
##  $ specialisation: chr  "Mkt&HR" "Mkt&Fin" "Mkt&Fin" "Mkt&HR" ...
##  $ mba_p         : num  58.8 66.3 57.8 59.4 55.5 ...
##  $ status        : chr  "Placed" "Placed" "Placed" "Not Placed" ...
##  $ salary        : int  270000 200000 250000 NA 425000 NA NA 252000 231000 NA ...
```

The dataset I am using for my final project is the Campus Recruitment dataset acquired from Kaggle (https://www.kaggle.com/datasets/benroshan/factors-affecting-campus-placement). This dataset records a total of 215 students' interview information and whether they get a placement or not.

The columns in the dataset represent the following information: - gender: gender, - ssc_p: secondary education percentage - how much scores the students get in the 10th grade. For instance, if one student get 120/200 in their 10th grade, the percentage would be 60%. In other words, the higher the percentage is, the better the student perform in that grade. The same explanation does apply to all following columns with "percentage." - ssc_b: secondary education board of education - central/others - hsc_p: higher secondary education percentage - 12th grade, - hsc_b: higher secondary education board of education - central/others, - hsc_s: specialization in higher secondary education, - degree_p: degree percentage, - degree_t: undergraduate field of degree, - workex: work experience, - etest_p: employability test percentage, - specialisation: MBA specialization, - mba_p: MBA percentage, - status: status of placement, - salary: salary offered by corporate to candidates.

The purpose of this project is to develop a model to project the candidates' placement result, which is represented by "status" column - "Placed" and "Not Placed." This would be a classification task.

2. Data Exploration

```r
# drop the id column
drops <- c("sl_no")
placement <- placement[ , !(names(placement) %in% drops)]

# drop the salary column
# since it is not our target variable and is not suited for predictors either
drops2 <- c("salary")
placement <- placement[ , !(names(placement) %in% drops2)]

# read other columns
str(placement)
```

```
## 'data.frame':    215 obs. of  13 variables:
##  $ gender        : chr  "M" "M" "M" "M" ...
##  $ ssc_p         : num  67 79.3 65 56 85.8 ...
##  $ ssc_b         : chr  "Others" "Central" "Central" "Central" ...
##  $ hsc_p         : num  91 78.3 68 52 73.6 ...
##  $ hsc_b         : chr  "Others" "Others" "Central" "Central" ...
##  $ hsc_s         : chr  "Commerce" "Science" "Arts" "Science" ...
##  $ degree_p      : num  58 77.5 64 52 73.3 ...
##  $ degree_t      : chr  "Sci&Tech" "Sci&Tech" "Comm&Mgmt" "Sci&Tech" ...
##  $ workex        : chr  "No" "Yes" "No" "No" ...
##  $ etest_p       : num  55 86.5 75 66 96.8 ...
##  $ specialisation: chr  "Mkt&HR" "Mkt&Fin" "Mkt&Fin" "Mkt&HR" ...
##  $ mba_p         : num  58.8 66.3 57.8 59.4 55.5 ...
##  $ status        : chr  "Placed" "Placed" "Placed" "Not Placed" ...
```

```r
# exploring categorical features
table(placement$gender)
```

```
## 
##   F   M 
##  76 139
```

```r
table(placement$ssc_b)
```

```
## 
## Central  Others 
##     116      99
```

```r
table(placement$hsc_b)
```

```
## 
## Central  Others 
##      84     131
```

```r
table(placement$hsc_s)
```

```
##
##     Arts Commerce  Science
##       11      113       91
```

```r
table(placement$degree_t)
```

```
##
## Comm&Mgmt    Others  Sci&Tech
##       145        11        59
```

```r
table(placement$workex)
```

```
##
##  No Yes
## 141  74
```

```r
table(placement$specialisation)
```

```
##
## Mkt&Fin  Mkt&HR
##     120      95
```

```r
table(placement$status)
```

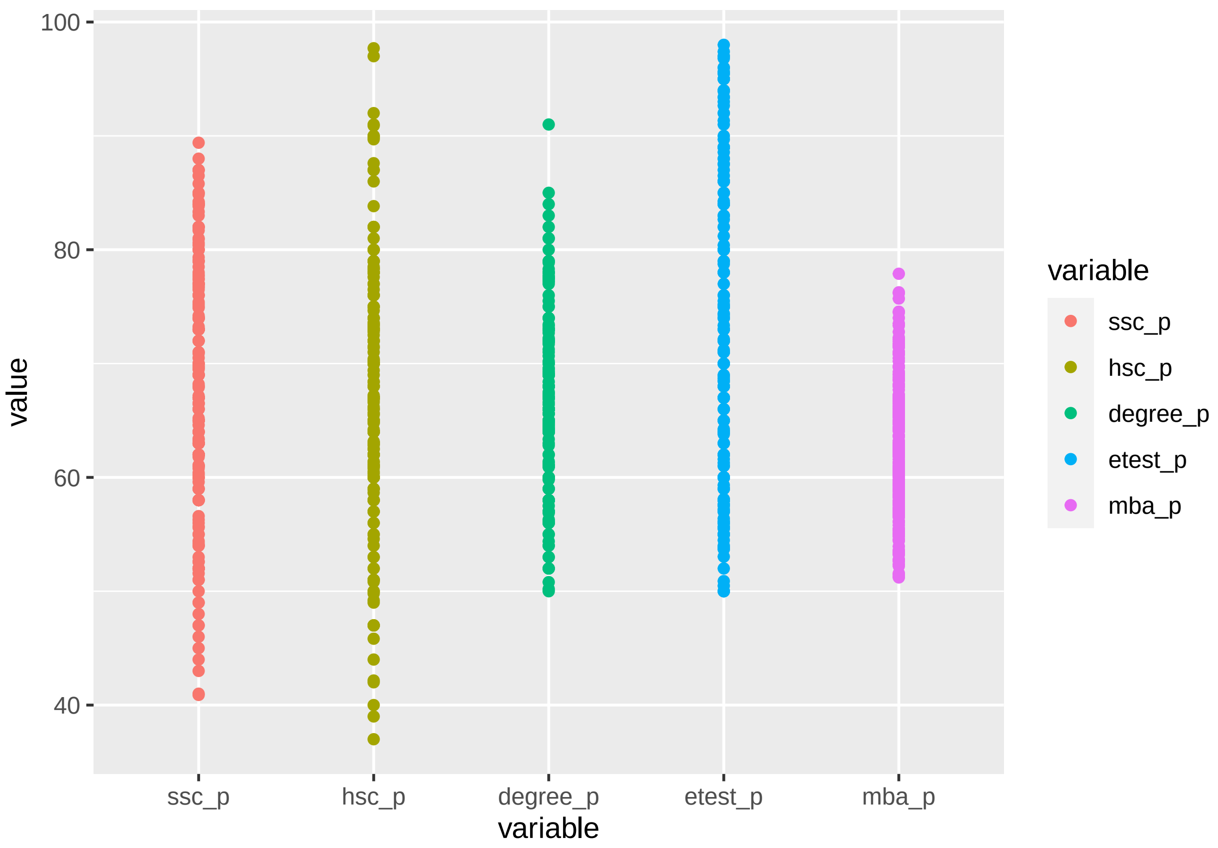
```
##
## Not Placed     Placed
##         67        148
```

As shown above, we can see that in this dataset, all the categorical features have 2 to 3 categories. We can apply dummy coding or one-hot encoding accordingly later.

```r
# exploring numeric features
summary(placement[c("ssc_p","hsc_p","degree_p","etest_p", "mba_p")])
```

```
##      ssc_p           hsc_p          degree_p        etest_p         mba_p
##  Min.   :40.89   Min.   :37.00   Min.   :50.00   Min.   :50.0   Min.   :51.21
##  1st Qu.:60.60   1st Qu.:60.90   1st Qu.:61.00   1st Qu.:60.0   1st Qu.:57.95
##  Median :67.00   Median :65.00   Median :66.00   Median :71.0   Median :62.00
##  Mean   :67.30   Mean   :66.33   Mean   :66.37   Mean   :72.1   Mean   :62.28
##  3rd Qu.:75.70   3rd Qu.:73.00   3rd Qu.:72.00   3rd Qu.:83.5   3rd Qu.:66.25
##  Max.   :89.40   Max.   :97.70   Max.   :91.00   Max.   :98.0   Max.   :77.89
```

```r
# Only keep the numerical features
categorical_features <- c("gender","ssc_b","hsc_b","hsc_s","degree_t","workex","specialisation","status
placement_numeric <- placement[ , !(names(placement) %in% categorical_features)]
ggplot(data = melt(placement_numeric), aes(x=variable, y=value)) +  geom_point(aes(colour=variable))
```



As shown above, since all the numerical features are percentages, their ranges are relatively similar to each other, from 40% to 100%.

Next, I want to see if there are any missing values in the dataset.

```
# identify missing values
any(is.na(placement))
```

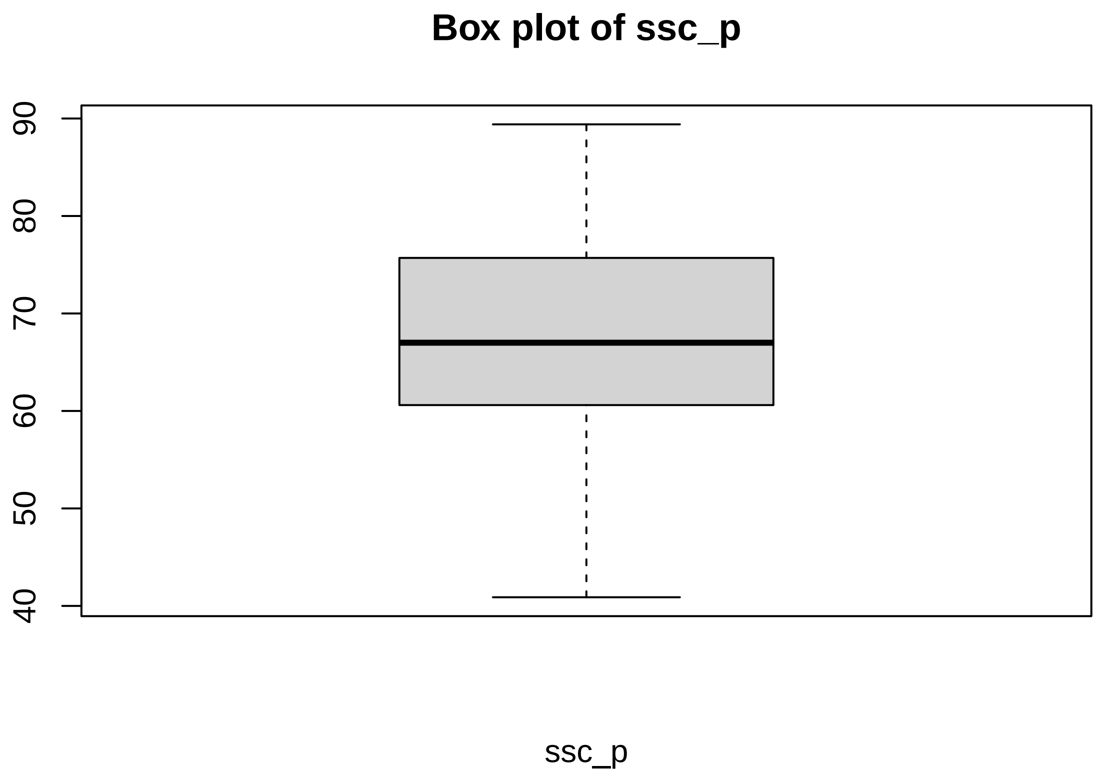
```
## [1] FALSE
```

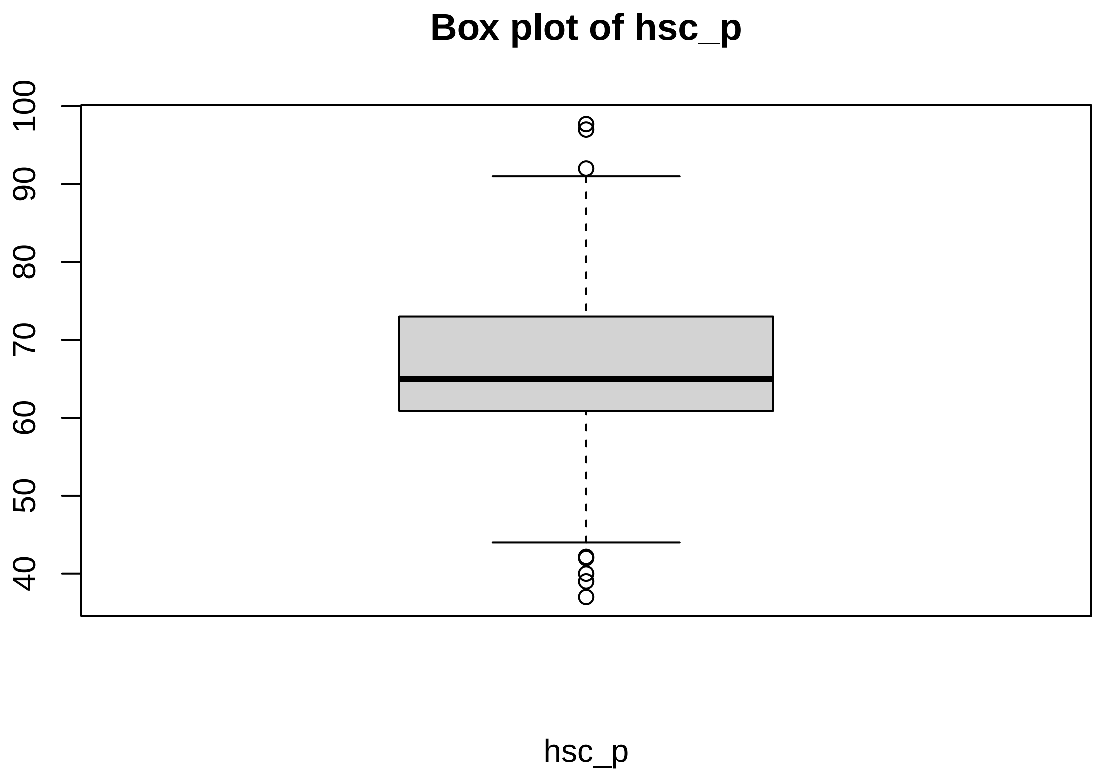
There is no missing value in the dataset. But just for demonstration purpose, I will impute missing data as following:

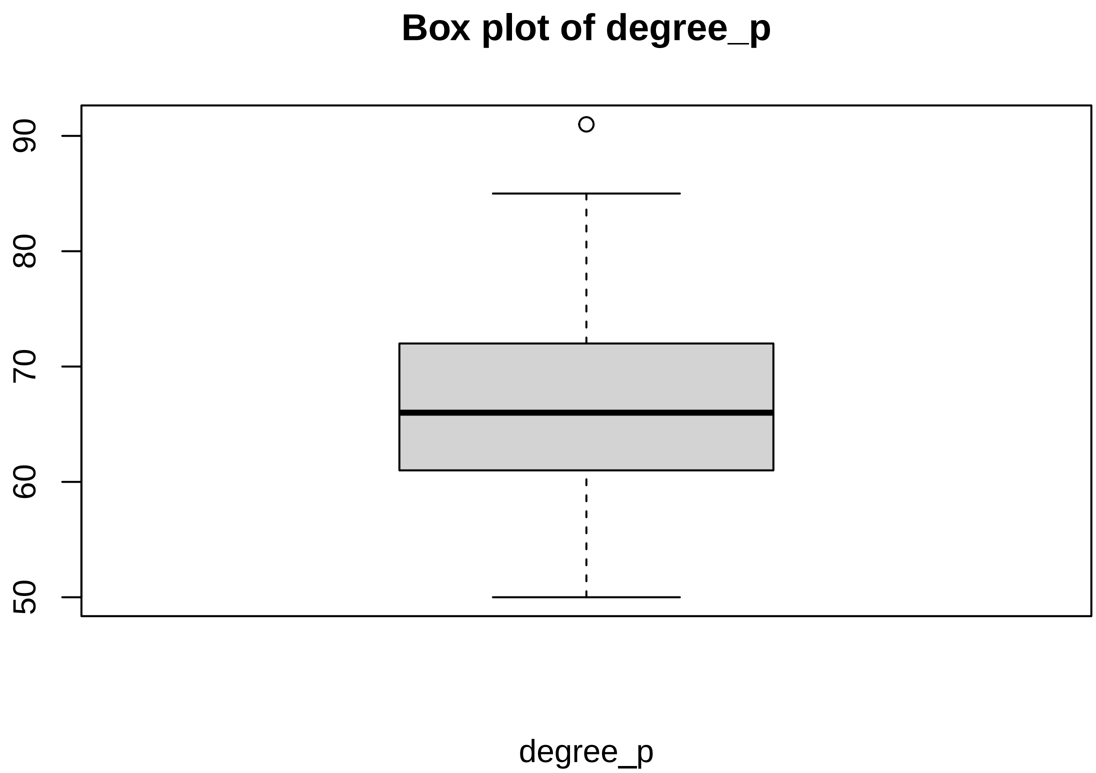
```
# (imagine if the dataset has already splited into test and train sets)
# impute missing data in training set
#placementTrain <- placementTrain %>%  mutate(across(where(is.numeric), ~replace_na(., median(., na.rm=

# impute missing data in test set
#placementTest <- placementTest %>%  mutate(across(where(is.numeric), ~replace_na(., median(., na.rm=TR
```

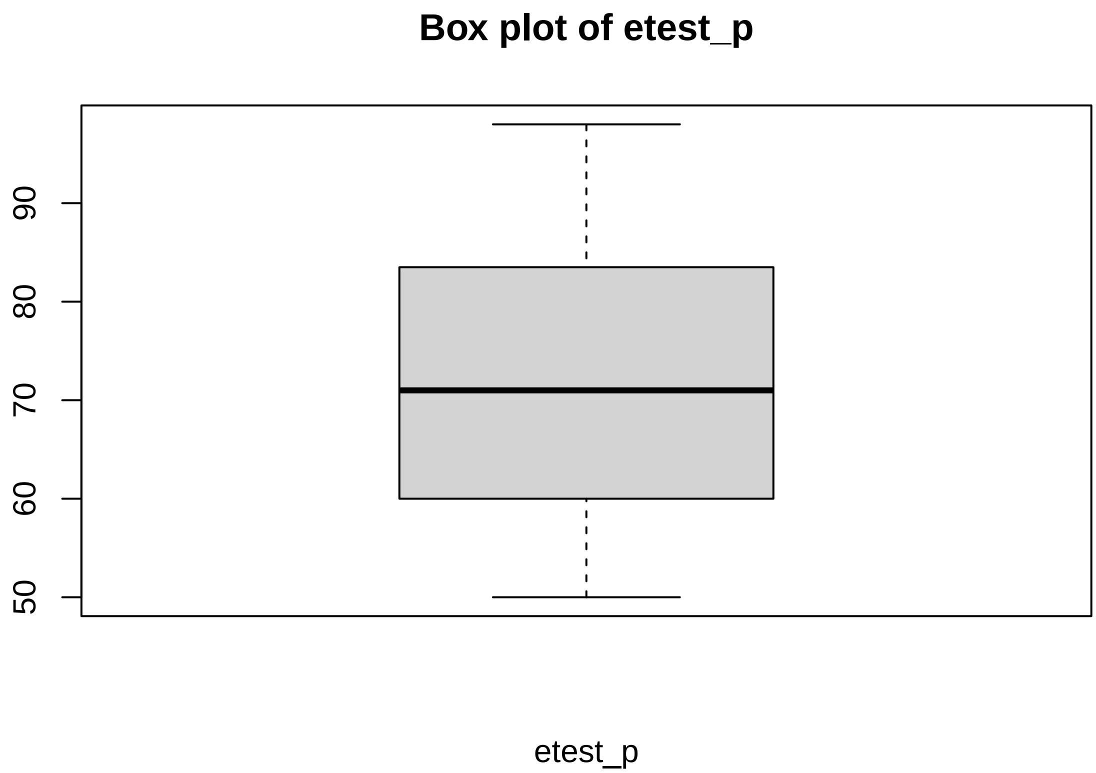
And if there are any outliers.

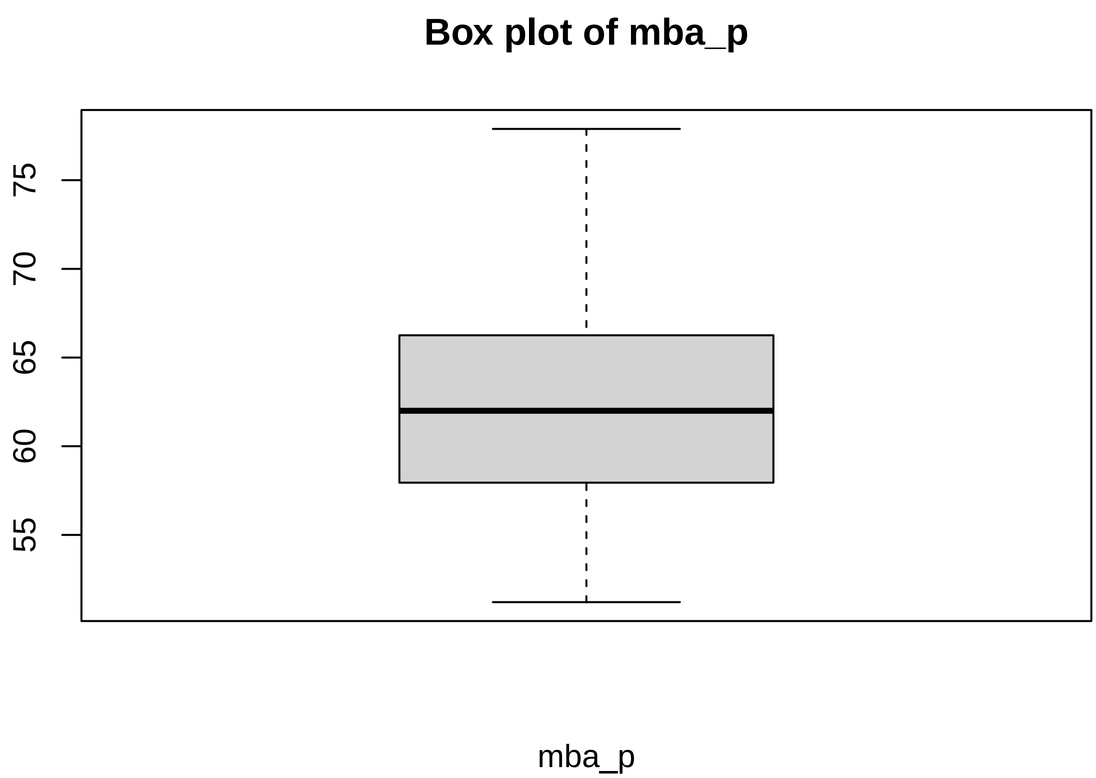
```
lapply(X=c("ssc_p", "hsc_p", "degree_p", "etest_p", "mba_p"),FUN=function(s)boxplot(placement[,s],main=
```


Box plot of ssc_p
90
80
70
60
50
40
ssc_p


Box plot of hsc_p
100
90
80
70
60
50
40
hsc_p


Box plot of degree_p
90
80
70
60
50
degree_p


Box plot of etest_p
90
80
70
60
50
etest_p




```
## [[1]]
## [[1]]$stats
##        [,1]
## [1,] 40.89
## [2,] 60.60
## [3,] 67.00
## [4,] 75.70
## [5,] 89.40
##
## [[1]]$n
## [1] 215
##
## [[1]]$conf
##          [,1]
## [1,] 65.3729
## [2,] 68.6271
##
## [[1]]$out
## numeric(0)
##
## [[1]]$group
## numeric(0)
##
## [[1]]$names
## [1] ""
##
```

```
##
## [[2]]
## [[2]]$stats
##      [,1]
## [1,] 44.0
## [2,] 60.9
## [3,] 65.0
## [4,] 73.0
## [5,] 91.0
##
## [[2]]$n
## [1] 215
##
## [[2]]$conf
##          [,1]
## [1,] 63.69616
## [2,] 66.30384
##
## [[2]]$out
## [1] 97.70 39.00 37.00 40.00 92.00 42.16 97.00 42.00
##
## [[2]]$group
## [1] 1 1 1 1 1 1 1 1
##
## [[2]]$names
## [1] ""
##
##
## [[3]]
## [[3]]$stats
##      [,1]
## [1,]   50
## [2,]   61
## [3,]   66
## [4,]   72
## [5,]   85
##
## [[3]]$n
## [1] 215
##
## [[3]]$conf
##          [,1]
## [1,] 64.81469
## [2,] 67.18531
##
## [[3]]$out
## [1] 91
##
## [[3]]$group
## [1] 1
##
## [[3]]$names
## [1] ""
##
```

```
##
## [[4]]
## [[4]]$stats
##      [,1]
## [1,] 50.0
## [2,] 60.0
## [3,] 71.0
## [4,] 83.5
## [5,] 98.0
##
## [[4]]$n
## [1] 215
##
## [[4]]$conf
##          [,1]
## [1,] 68.46776
## [2,] 73.53224
##
## [[4]]$out
## numeric(0)
##
## [[4]]$group
## numeric(0)
##
## [[4]]$names
## [1] ""
##
##
## [[5]]
## [[5]]$stats
##        [,1]
## [1,] 51.210
## [2,] 57.945
## [3,] 62.000
## [4,] 66.255
## [5,] 77.890
##
## [[5]]$n
## [1] 215
##
## [[5]]$conf
##          [,1]
## [1,] 61.10456
## [2,] 62.89544
##
## [[5]]$out
## numeric(0)
##
## [[5]]$group
## numeric(0)
##
## [[5]]$names
## [1] ""
```

As shown in the box plot, there are some outliers in the numeric variables. And I can find them out using z-score.

```
# detect outliers using z-index

# z-index function
findOutliers <- function(x){
  m <- mean(x)
  sd <- sd(x)
  z <- abs((x - m) / sd)
  rows.outliers <- which(z > 2.5)
  return (rows.outliers)
}

# find rows with outliers
sapply(placement_numeric[1:5], findOutliers)
```

```
## $ssc_p
## integer(0)
##
## $hsc_p
## [1]  25  43  50 178
##
## $degree_p
## [1]  22 198
##
## $etest_p
## integer(0)
##
## $mba_p
## [1] 20
```

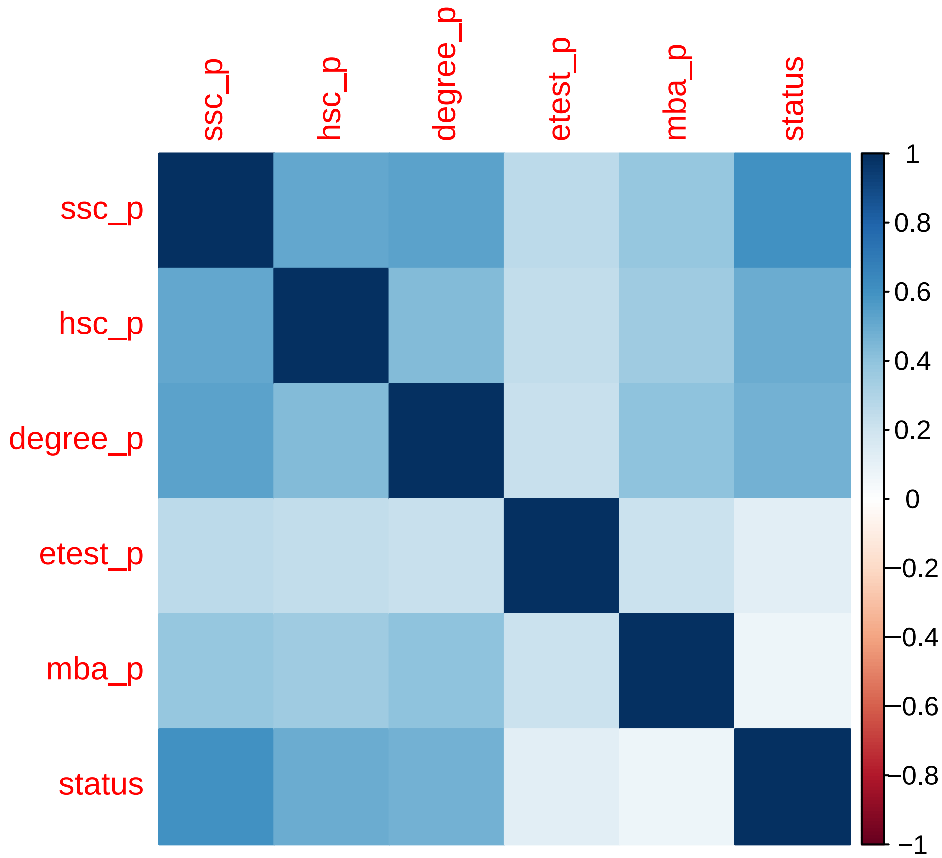
```
#apply(student[5:16], findOutliers)
```

As shwon above, according to the Z-index, there are 7 outliers in the dataset, not a large portion. I will remove them.

```
out_ind <- sapply(placement_numeric[1:5], findOutliers)
out_ind <- unique(unname(unlist(out_ind)))

# remove outliers from the original dataset
placement.no <- placement[-c(out_ind), ]
```

Now we will use correlation analysis to see if numerical predictors are related to the target variable.

```
# first to encode the target variable
categorical_features <- c("gender","ssc_b","hsc_b","hsc_s","degree_t","workex","specialisation")
payment_numeric <- placement[ , !(names(placement) %in% categorical_features)] %>%
  mutate(status = ifelse(status=="Placed", 1, 0))

#visualize a correlation matrix for numeric variables
M <- cor(payment_numeric)
corrplot(M, method="color")
```



As shown above, the placement status is somewhat related to "ssc_p" (secondary education test percentage), "hsc_p" (higher secondary education test percentage), "degree_p" (degree test percentage) but not much correlated with "etest_p" (employability test percentage) and "mba_p" (MBA test percentage).

I will use chi-squared test to test the association among categorical variables.

```
# Null hypothesis: the variables are independent. If p <0.05, we must reject the null hypothesis.
# Chi-square test Reference: https://support.minitab.com/en-us/minitab/help-and-how-to/statistics/table
# http://www.sthda.com/english/wiki/chi-square-test-of-independence-in-r#google_vignette

categorical_features <- c("gender","ssc_b","hsc_b","hsc_s","degree_t","workex","specialisation","status
placement_categorical <- placement[ , (names(placement) %in% categorical_features)]
#head(placement_categorical)
```

Next I will run chi-squared test for categorical features.

```
# chi-squared test for gender
lapply(placement_categorical[,-1], function(x) chisq.test(placement_categorical[,1], x))
```

```
## $ssc_b
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 1] and x
## X-squared = 0.020101, df = 1, p-value = 0.8873
##
```

```
##
## $hsc_b
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 1] and x
## X-squared = 0.67362, df = 1, p-value = 0.4118
##
##
## $hsc_s
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 1] and x
## X-squared = 1.9998, df = 2, p-value = 0.3679
##
##
## $degree_t
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 1] and x
## X-squared = 2.9682, df = 2, p-value = 0.2267
##
##
## $workex
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 1] and x
## X-squared = 1.2066, df = 1, p-value = 0.272
##
##
## $specialisation
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 1] and x
## X-squared = 1.9965, df = 1, p-value = 0.1577
##
##
## $status
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 1] and x
## X-squared = 1.3818, df = 1, p-value = 0.2398
```

As shown above, the gender feature is not associated with any other categorical feature.

```r
# chi-squared test for ssc_b
lapply(placement_categorical[,-2], function(x) chisq.test(placement_categorical[,2], x))
```

```
## $gender
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 2] and x
## X-squared = 0.020101, df = 1, p-value = 0.8873
##
##
## $hsc_b
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 2] and x
## X-squared = 76.454, df = 1, p-value < 2.2e-16
##
##
## $hsc_s
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 2] and x
## X-squared = 0.75357, df = 2, p-value = 0.6861
##
##
## $degree_t
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 2] and x
## X-squared = 2.2257, df = 2, p-value = 0.3286
##
##
## $workex
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 2] and x
## X-squared = 0.20559, df = 1, p-value = 0.6502
##
##
## $specialisation
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 2] and x
## X-squared = 0.38233, df = 1, p-value = 0.5364
##
##
## $status
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 2] and x
## X-squared = 0.15933, df = 1, p-value = 0.6898
```

The “ssc_b” has association with “hsc_b” since the p value is smaller than 0.05. Hence, I may eliminate one of these features later.

```r
# chi-squared test for hsc_b
lapply(placement_categorical[,-3], function(x) chisq.test(placement_categorical[,3], x))
```

```
## $gender
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 3] and x
## X-squared = 0.67362, df = 1, p-value = 0.4118
##
##
## $ssc_b
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 3] and x
## X-squared = 76.454, df = 1, p-value < 2.2e-16
##
##
## $hsc_s
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 3] and x
## X-squared = 5.3227, df = 2, p-value = 0.06985
##
##
## $degree_t
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 3] and x
## X-squared = 4.01, df = 2, p-value = 0.1347
##
##
## $workex
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 3] and x
## X-squared = 0.17249, df = 1, p-value = 0.6779
##
##
## $specialisation
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 3] and x
## X-squared = 0, df = 1, p-value = 1
##
##
```

```
## $status
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 3] and x
## X-squared = 0.0095175, df = 1, p-value = 0.9223
```

Again, the "hsc_b" and "ssc_b" have strong association and may be eliminated later.

```r
# chi-squared test for hsc_s
lapply(placement_categorical[,-4], function(x) chisq.test(placement_categorical[,4], x))
```

```
## $gender
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 4] and x
## X-squared = 1.9998, df = 2, p-value = 0.3679
##
##
## $ssc_b
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 4] and x
## X-squared = 0.75357, df = 2, p-value = 0.6861
##
##
## $hsc_b
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 4] and x
## X-squared = 5.3227, df = 2, p-value = 0.06985
##
##
## $degree_t
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 4] and x
## X-squared = 123.41, df = 4, p-value < 2.2e-16
##
##
## $workex
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 4] and x
## X-squared = 1.0589, df = 2, p-value = 0.5889
##
##
## $specialisation
```

```
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 4] and x
## X-squared = 6.4426, df = 2, p-value = 0.0399
##
##
## $status
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 4] and x
## X-squared = 1.1147, df = 2, p-value = 0.5727
```

The "hsc_s" has association with "degree_t" and "specialisation." Hence, I may remove "hsc_s" feature later.

```r
# chi-squared test for degree_t
lapply(placement_categorical[,-5], function(x) chisq.test(placement_categorical[,5], x))
```

```
## $gender
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 5] and x
## X-squared = 2.9682, df = 2, p-value = 0.2267
##
##
## $ssc_b
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 5] and x
## X-squared = 2.2257, df = 2, p-value = 0.3286
##
##
## $hsc_b
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 5] and x
## X-squared = 4.01, df = 2, p-value = 0.1347
##
##
## $hsc_s
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 5] and x
## X-squared = 123.41, df = 4, p-value < 2.2e-16
##
##
## $workex
```

```
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 5] and x
## X-squared = 2.4079, df = 2, p-value = 0.3
##
##
## $specialisation
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 5] and x
## X-squared = 2.9963, df = 2, p-value = 0.2235
##
##
## $status
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 5] and x
## X-squared = 2.969, df = 2, p-value = 0.2266
```

The "degree_t" has strong association with "hsc_s." The later one may be removed later, as it has strong association with two other features.

```r
# chi-squared test for workex
lapply(placement_categorical[,-6], function(x) chisq.test(placement_categorical[,6], x))
```

```
## $gender
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 6] and x
## X-squared = 1.2066, df = 1, p-value = 0.272
##
##
## $ssc_b
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 6] and x
## X-squared = 0.20559, df = 1, p-value = 0.6502
##
##
## $hsc_b
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 6] and x
## X-squared = 0.17249, df = 1, p-value = 0.6779
##
##
## $hsc_s
```

```
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 6] and x
## X-squared = 1.0589, df = 2, p-value = 0.5889
##
##
## $degree_t
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 6] and x
## X-squared = 2.4079, df = 2, p-value = 0.3
##
##
## $specialisation
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 6] and x
## X-squared = 7.0683, df = 1, p-value = 0.007846
##
##
## $status
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 6] and x
## X-squared = 15.154, df = 1, p-value = 9.907e-05
```

The "workex" has fairly strong association with "specialisation" and target variable "status."

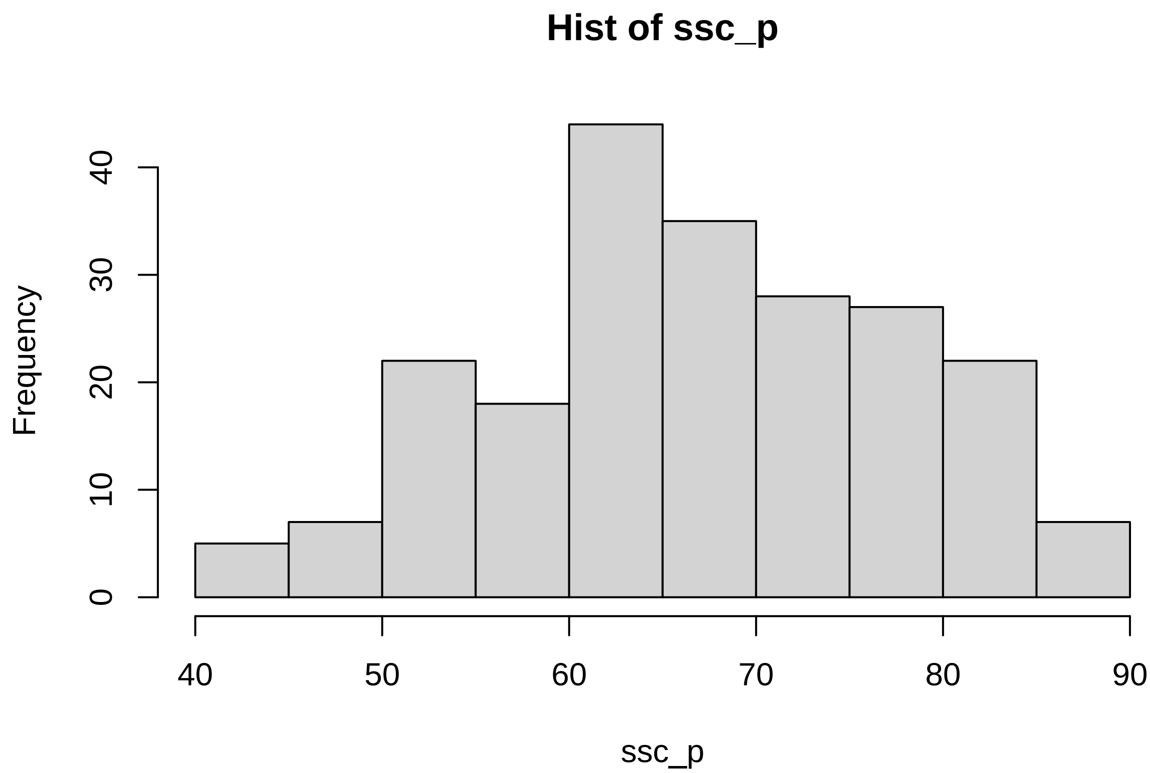
```r
# chi-squared test for specialisation
lapply(placement_categorical[,-7], function(x) chisq.test(placement_categorical[,7], x))
```

```
## $gender
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 7] and x
## X-squared = 1.9965, df = 1, p-value = 0.1577
##
##
## $ssc_b
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 7] and x
## X-squared = 0.38233, df = 1, p-value = 0.5364
##
##
## $hsc_b
##
```

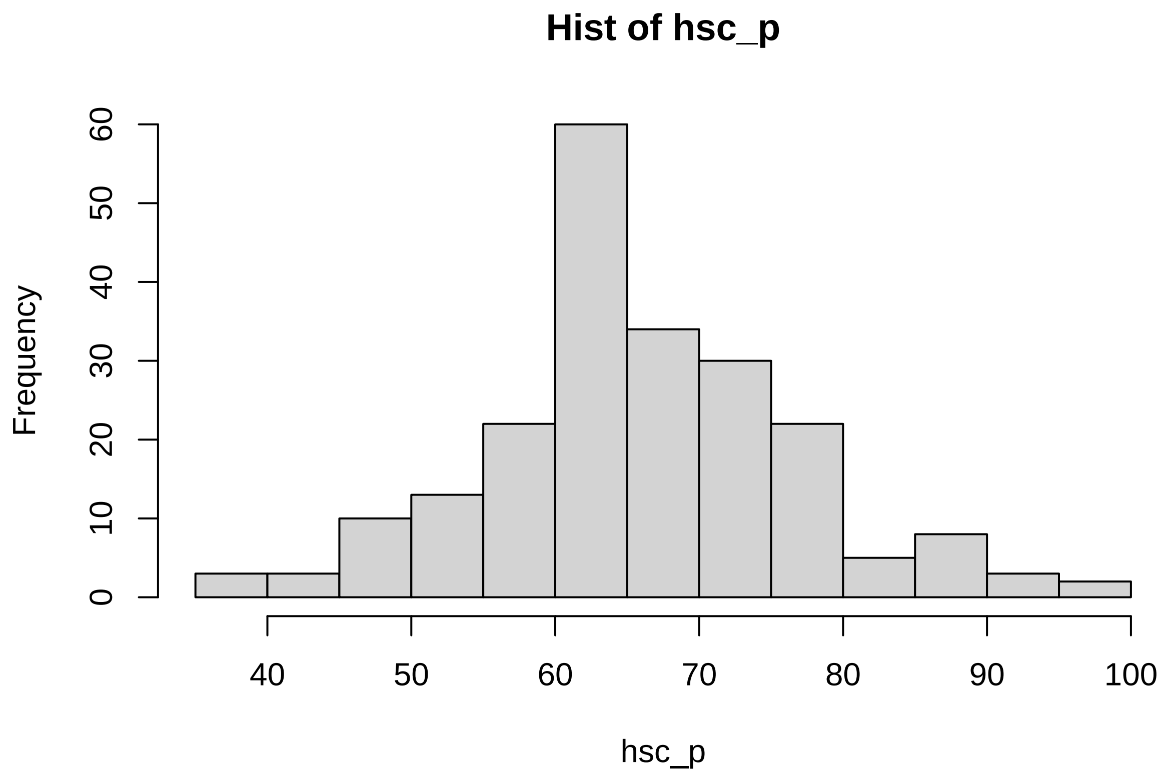
```
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 7] and x
## X-squared = 0, df = 1, p-value = 1
##
##
## $hsc_s
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 7] and x
## X-squared = 6.4426, df = 2, p-value = 0.0399
##
##
## $degree_t
##
##  Pearson's Chi-squared test
##
## data:  placement_categorical[, 7] and x
## X-squared = 2.9963, df = 2, p-value = 0.2235
##
##
## $workex
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 7] and x
## X-squared = 7.0683, df = 1, p-value = 0.007846
##
##
## $status
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  placement_categorical[, 7] and x
## X-squared = 12.44, df = 1, p-value = 0.0004202
```

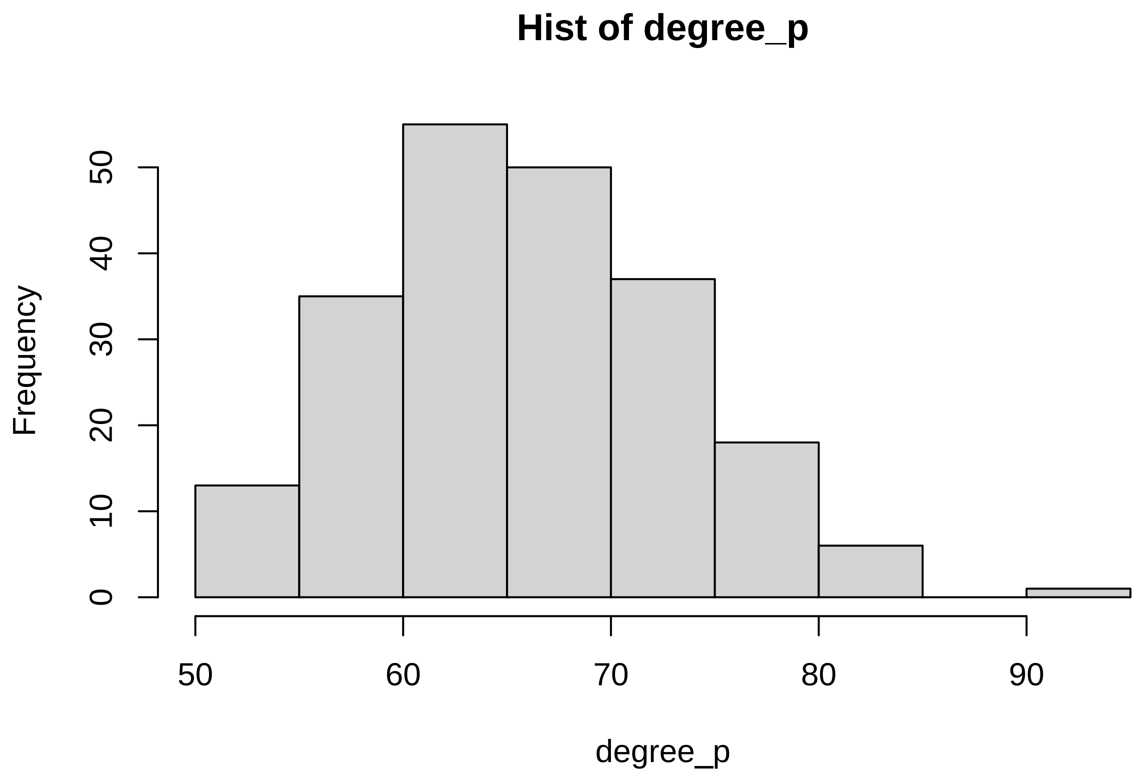
The "specialisation" has association with "hsc_s", "workex" and target variable "status."

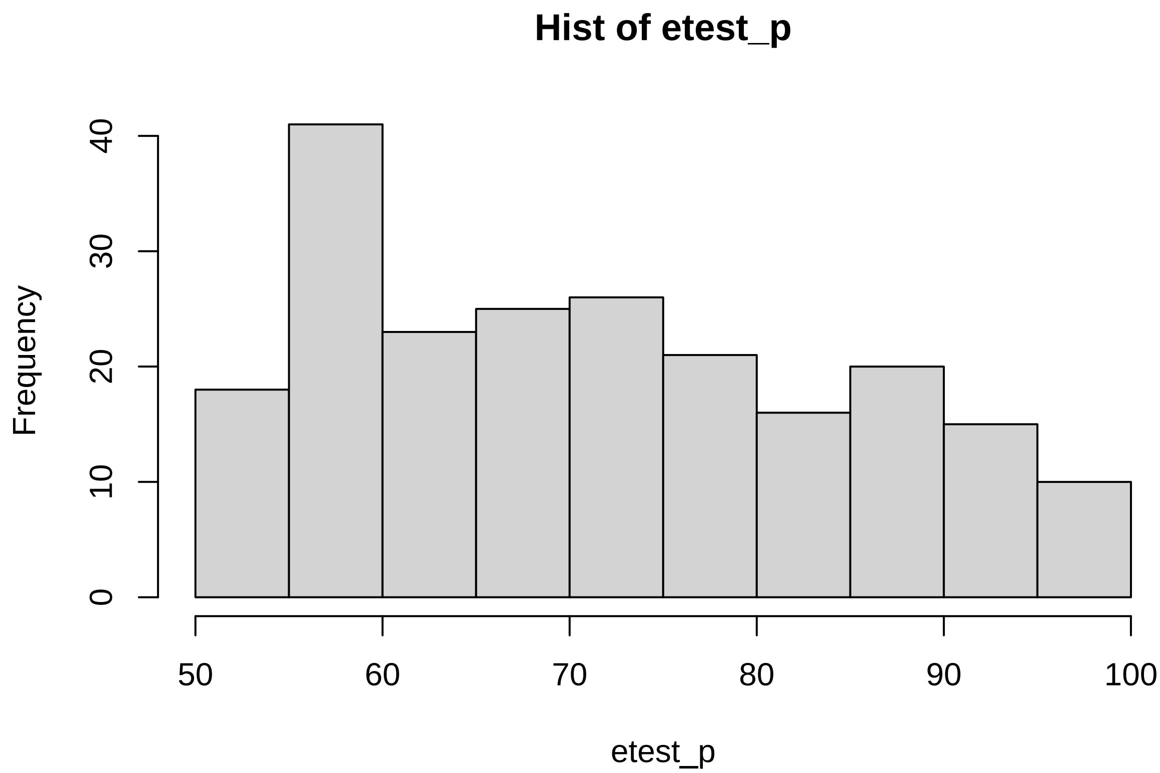
```
# evaluation of distribution
lapply(X=c("ssc_p", "hsc_p", "degree_p", "etest_p", "mba_p"),FUN=function(s)hist(placement[,s],main=pas
```



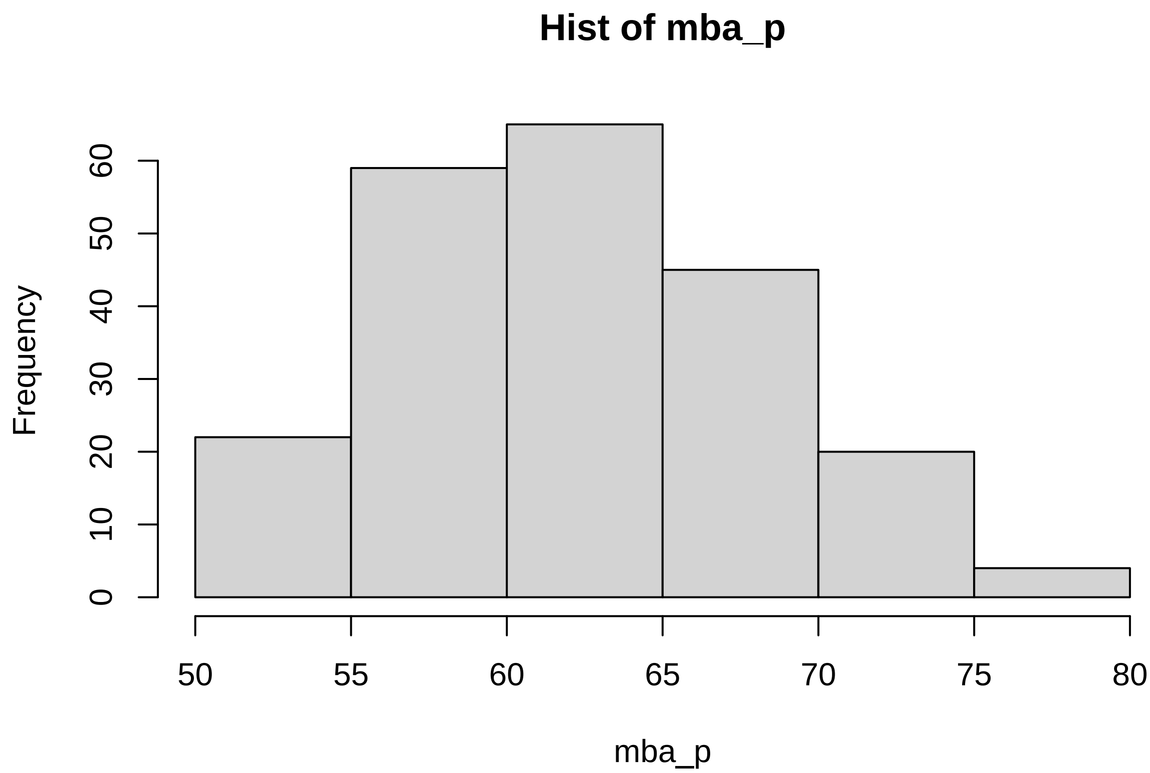
Hist of ssc_p
Frequency
0
10
20
30
40
40
50
60
70
80
90
ssc_p


Hist of hsc_p
Frequency
0
10
20
30
40
50
60
40
50
60
70
80
90
100
hsc_p


Hist of degree_p
Frequency
0
10
20
30
40
50
50
60
70
80
90
degree_p



Hist of etest_p
Frequency
0
10
20
30
40
50
60
70
80
90
100
etest_p




```
## [[1]]
## $breaks
##  [1] 40 45 50 55 60 65 70 75 80 85 90
##
## $counts
##  [1]  5  7 22 18 44 35 28 27 22  7
##
## $density
##  [1] 0.004651163 0.006511628 0.020465116 0.016744186 0.040930233 0.032558140
##  [7] 0.026046512 0.025116279 0.020465116 0.006511628
##
## $mids
##  [1] 42.5 47.5 52.5 57.5 62.5 67.5 72.5 77.5 82.5 87.5
##
## $xname
## [1] "placement[, s]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[2]]
## $breaks
##  [1]  35  40  45  50  55  60  65  70  75  80  85  90  95 100
```

```
##
## $counts
##  [1]  3  3 10 13 22 60 34 30 22  5  8  3  2
##
## $density
##  [1] 0.002790698 0.002790698 0.009302326 0.012093023 0.020465116 0.055813953
##  [7] 0.031627907 0.027906977 0.020465116 0.004651163 0.007441860 0.002790698
## [13] 0.001860465
##
## $mids
##  [1] 37.5 42.5 47.5 52.5 57.5 62.5 67.5 72.5 77.5 82.5 87.5 92.5 97.5
##
## $xname
## [1] "placement[, s]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[3]]
## $breaks
##  [1] 50 55 60 65 70 75 80 85 90 95
##
## $counts
## [1] 13 35 55 50 37 18  6  0  1
##
## $density
## [1] 0.0120930233 0.0325581395 0.0511627907 0.0465116279 0.0344186047
## [6] 0.0167441860 0.0055813953 0.0000000000 0.0009302326
##
## $mids
## [1] 52.5 57.5 62.5 67.5 72.5 77.5 82.5 87.5 92.5
##
## $xname
## [1] "placement[, s]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[4]]
## $breaks
##  [1]  50  55  60  65  70  75  80  85  90  95 100
##
## $counts
##  [1] 18 41 23 25 26 21 16 20 15 10
##
## $density
##  [1] 0.016744186 0.038139535 0.021395349 0.023255814 0.024186047 0.019534884
##  [7] 0.014883721 0.018604651 0.013953488 0.009302326
```

```
##
## $mids
##  [1] 52.5 57.5 62.5 67.5 72.5 77.5 82.5 87.5 92.5 97.5
##
## $xname
## [1] "placement[, s]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
##
## [[5]]
## $breaks
## [1] 50 55 60 65 70 75 80
##
## $counts
## [1] 22 59 65 45 20  4
##
## $density
## [1] 0.02046512 0.05488372 0.06046512 0.04186047 0.01860465 0.00372093
##
## $mids
## [1] 52.5 57.5 62.5 67.5 72.5 77.5
##
## $xname
## [1] "placement[, s]"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

As shwon above, all the numeric variables are somewhat normally distributed.

3. Data Cleaning & Shaping

```r
# split into train and test sets
set.seed(111)

# set samples range
samples <- sample.int(n = nrow(placement), size = floor(.7*nrow(placement)), replace = F)

# for k-NN
# train set
placementTrain <- placement[samples, 1:12]
# validation test
placementTest <- placement[-samples, 1:12]
# save target variable separately
```

```
placementTrain.labels <- placement[samples, 13]
placementTest.labels <- placement[-samples, 13]

# for Decision Trees
placementTrain_tree <- placement[samples, ]
placementTest_tree <- placement[-samples, ]
```

Next, we will normalize the numerical features for the k-NN algorithm model using Z-score standardization. Because Decision Trees and OneR Rule Learners do not need normalization, we will save the normalized data in another dataframe.

```
set.seed(111)

# create Z-score Standardization
z_standardization <- function(data) {
  standardized.df <- data.frame(matrix(ncol = ncol(data), nrow=nrow(data)))
  names(standardized.df) <- names(data)

  for (i in seq_along(data)){
      column_data <- data[[i]]
      mean_value <- mean(column_data, na.rm=TRUE)
      sd_value <- sd(column_data, na.rm=TRUE)
      standardized.df[[i]] <- (column_data - mean_value) / sd_value
  }

  return (standardized.df)
}

# normalize the numerical features in train set
placementTrain_numeric <- placementTrain[, !(colnames(placementTrain) %in% c("gender", "ssc_b","hsc_b",
# create a new standardization dataset
placementTrain.standardized <- z_standardization(placementTrain_numeric)
#studentTrain.standardized <- as.data.frame(scale(studentTrain_numeric))

# add categorical features to the new dataset
placementTrain.standardized$gender <- placementTrain$gender
placementTrain.standardized$ssc_b <- placementTrain$ssc_b
placementTrain.standardized$hsc_b <- placementTrain$hsc_b
placementTrain.standardized$hsc_s <- placementTrain$hsc_s
placementTrain.standardized$degree_t <- placementTrain$degree_t
placementTrain.standardized$workex <- placementTrain$workex
placementTrain.standardized$specialisation <- placementTrain$specialisation

# normalize the numerical features in test set
placementTest_numeric <- placementTest[, !(colnames(placementTest) %in% c("gender", "ssc_b","hsc_b", "h
placementTest.standardized <-  z_standardization(placementTest_numeric)
placementTest.standardized$gender <- placementTest$gender
placementTest.standardized$ssc_b <- placementTest$ssc_b
placementTest.standardized$hsc_b <- placementTest$hsc_b
placementTest.standardized$hsc_s <- placementTest$hsc_s
placementTest.standardized$degree_t <- placementTest$degree_t
placementTest.standardized$workex <- placementTest$workex
placementTest.standardized$specialisation <- placementTest$specialisation
```

Now we will do some feature engineering. First, we need to encode categorical features.

```
# train set
# dummy coding for binary feature
placementTrain.standardized$gender <- ifelse(placementTrain.standardized$gender=="M", 1, 0)
placementTrain.standardized$ssc_b <- ifelse(placementTrain.standardized$ssc_b=="Central", 1, 0)
placementTrain.standardized$hsc_b <- ifelse(placementTrain.standardized$hsc_b=="Central", 1, 0)
placementTrain.standardized$workex <- ifelse(placementTrain.standardized$workex=="Yes", 1, 0)
placementTrain.standardized$specialisation <- ifelse(placementTrain.standardized$specialisation=="Mkt&Fi

# one-hot coding for other categorical features
placementTrain.standardized_oneHot <- placementTrain.standardized %>%
  model.matrix(~ hsc_s + degree_t, data = .) %>%
  as.data.frame()

placementTrain.standardized <- cbind(placementTrain.standardized, placementTrain.standardized_oneHot[2:5
drops <- c("hsc_s", "degree_t")
placementTrain.standardized <- placementTrain.standardized[ , !(names(placementTrain.standardized) %in%

# test set
placementTest.standardized$gender <- ifelse(placementTest.standardized$gender=="M", 1, 0)
placementTest.standardized$ssc_b <- ifelse(placementTest.standardized$ssc_b=="Central", 1, 0)
placementTest.standardized$hsc_b <- ifelse(placementTest.standardized$hsc_b=="Central", 1, 0)
placementTest.standardized$workex <- ifelse(placementTest.standardized$workex=="Yes", 1, 0)
placementTest.standardized$specialisation <- ifelse(placementTest.standardized$specialisation=="Mkt&Fin

# one-hot coding for other categorical features
placementTest.standardized_oneHot <- placementTest.standardized %>%
  model.matrix(~ hsc_s + degree_t, data = .) %>%
  as.data.frame()

placementTest.standardized <- cbind(placementTest.standardized, placementTest.standardized_oneHot[2:5])
drops <- c("hsc_s", "degree_t")
placementTest.standardized <- placementTest.standardized[ , !(names(placementTest.standardized) %in% dr
```

```
# for SVM
# train data
placementTrain_svm <- placementTrain.standardized
placementTrain_svm$status <- as.factor(placementTrain.labels)

# test data
placementTest_svm <- placementTest.standardized
placementTest_svm$status <- as.factor(placementTest.labels)
```

4. K-NN model

4.1 Model Construction

```
set.seed(111)
student_knn_pred <- knn(train=placementTrain.standardized, test=placementTest.standardized, cl=placemen
```

I chose the k = 12 because there are 150 instances in the train set, and by default, the k value should roughly equal to the square root of the instance number. I will test out multiple k values in the model evaluation.

4.2 Model Evaluation

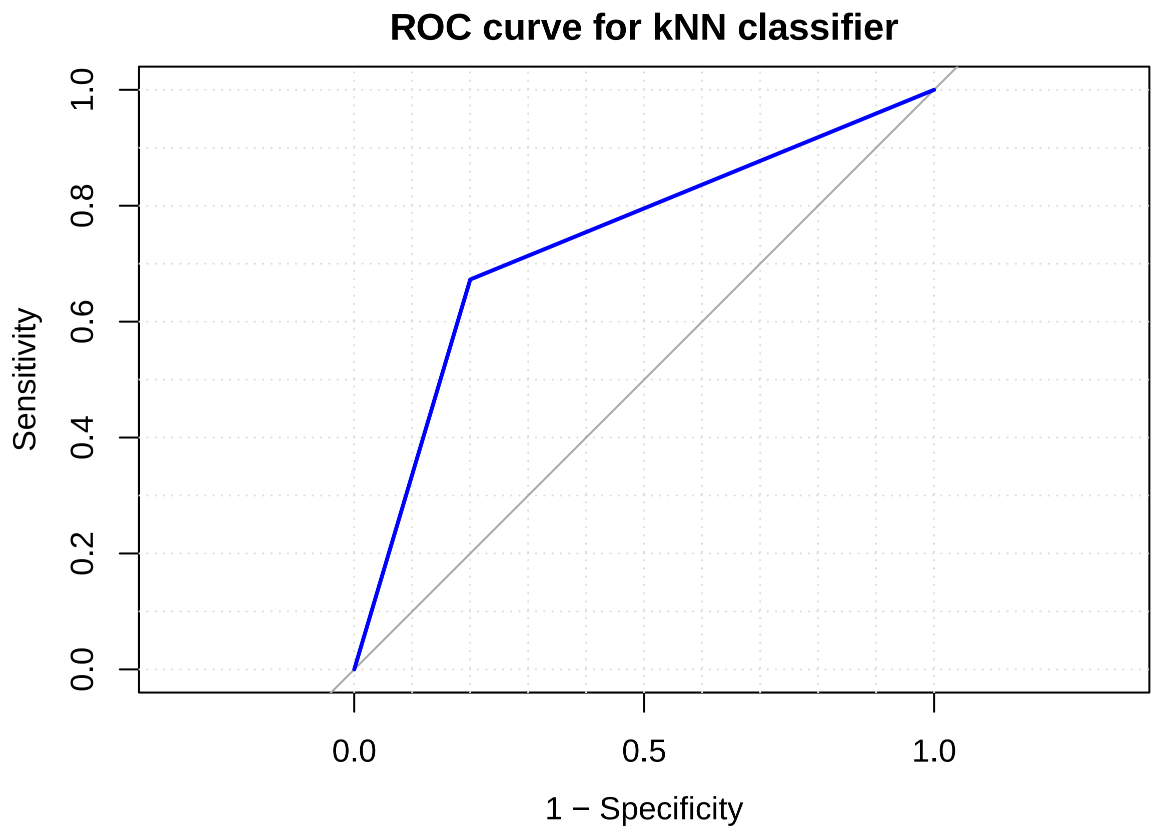
4.2.1 Confusion Matrix

```r
confusionMatrix(factor(student_knn_pred, levels=c("Placed","Not Placed")), factor(placementTest.labels,
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   Placed Not Placed
##   Placed         37         18
##   Not Placed      2          8
##
##                Accuracy : 0.6923
##                  95% CI : (0.5655, 0.8009)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.0803636
##
##                   Kappa : 0.2857
##
##  Mcnemar's Test P-Value : 0.0007962
##
##             Sensitivity : 0.9487
##             Specificity : 0.3077
##          Pos Pred Value : 0.6727
##          Neg Pred Value : 0.8000
##              Prevalence : 0.6000
##          Detection Rate : 0.5692
##    Detection Prevalence : 0.8462
##       Balanced Accuracy : 0.6282
##
##        'Positive' Class : Placed
##
```

As shown above, the overall accuracy of the model is (37+8)/65 = 69.2%. A decent accuracy rate. Looking closer… Also, the Kappa value of 0.28 shows that the model has a fair agreement. Could be better. Looking closer, about 94.9% of "Placed" class is predicted correctly while only 30.8% of "Not Placed" is predicted correctly. This indicates that the model prediction might be affected by the data imbalance because only 27.3% of target variables in the train set belong to the "Not Placed" class.

```r
# ROC curve
# consider put three curves together

student_knn_pred_num <- ifelse(student_knn_pred=="Placed", 1, 0)
placementTest.labels_num <- ifelse(placementTest.labels=="Placed", 1, 0)

knn_roc <- roc(student_knn_pred_num, placementTest.labels_num)
plot(knn_roc, main="ROC curve for kNN classifier", col="blue", lwd=2, grid=TRUE, legacy.axes=TRUE)
```



4.3 Model Tuning & Performance Improvement

4.3.1 Alternative k

To improve the k-NN model, we can test alternative values of k.

```
k_values <- c(6,7,8,9,10,11,12,13,14,15)
k_values_this <- c()
knn_accuracy_list <- c()

for (k_val in k_values){
  set.seed(111)
  student_knn_pred <- knn(train=placementTrain.standardized,
                          test=placementTest.standardized,
                          cl=placementTrain.labels,
                          k=k_val)

  accuracyRate <- 1 - mean(student_knn_pred != placementTest.labels)
  k_values_this<- c(k_val, k_values_this)
  knn_accuracy_list <- c(accuracyRate, knn_accuracy_list)
  #print(j)
}

knn_improved_accuracy <- data.frame(
  k_values = k_values_this,
```

```
  accuracy_rate = knn_accuracy_list,
  stringsAsFactors = FALSE
)

knn_improved_accuracy
```

```
##    k_values accuracy_rate
## 1        15     0.7384615
## 2        14     0.7230769
## 3        13     0.7384615
## 4        12     0.6923077
## 5        11     0.7692308
## 6        10     0.7846154
## 7         9     0.7692308
## 8         8     0.7692308
## 9         7     0.7692308
## 10        6     0.7076923
```

Surpringsly, the accuracy rate reaches the highest when k=10, which would be my improved hyperparameter. With the new k value, the overall accuracy is boosted from 69.2% to 78.5%. Good improvement!

```
set.seed(111)
student_knn_pred <- knn(train=placementTrain.standardized,
                       test=placementTest.standardized,
                       cl=placementTrain.labels,
                       k=10)

confusionMatrix(factor(student_knn_pred, levels=c("Placed","Not Placed")), factor(placementTest.labels,
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   Placed Not Placed
##   Placed         38         13
##   Not Placed      1         13
##
##                Accuracy : 0.7846
##                  95% CI : (0.6651, 0.8769)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.001288
##
##                   Kappa : 0.5139
##
##  Mcnemar's Test P-Value : 0.003283
##
##             Sensitivity : 0.9744
##             Specificity : 0.5000
##          Pos Pred Value : 0.7451
##          Neg Pred Value : 0.9286
##              Prevalence : 0.6000
##          Detection Rate : 0.5846
##    Detection Prevalence : 0.7846
```

```
##       Balanced Accuracy : 0.7372
##
##        'Positive' Class : Placed
##
```

Also, when k is set to 10, the precision rate of "Not Placed" class jumps to 50%, which is better than 30.8% before, as well as the share of 27.3% in the train set.

4.3.2 Reduce Model Complexity

As shown in the previous section, the "ssc_b" and "hsc_s" are strongly associated with other features but not associated with the target variable. Hence, I will remove these two from the predictor list.

```r
set.seed(111)

placementTrain.standardized_reduce <- placementTrain.standardized[, !(colnames(placementTrain.standardi
placementTest.standardized_reduce <- placementTest.standardized[, !(colnames(placementTest.standardized)

student_knn_pred_reduce <- knn(train=placementTrain.standardized_reduce, test=placementTest.standardize

confusionMatrix(factor(student_knn_pred_reduce, levels=c("Placed","Not Placed")), factor(placementTest.
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   Placed Not Placed
##   Placed         38         14
##   Not Placed      1         12
##
##                Accuracy : 0.7692
##                  95% CI : (0.6481, 0.8647)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.003088
##
##                   Kappa : 0.4755
##
##  Mcnemar's Test P-Value : 0.001946
##
##             Sensitivity : 0.9744
##             Specificity : 0.4615
##          Pos Pred Value : 0.7308
##          Neg Pred Value : 0.9231
##              Prevalence : 0.6000
##          Detection Rate : 0.5846
##    Detection Prevalence : 0.8000
##       Balanced Accuracy : 0.7179
##
##        'Positive' Class : Placed
##
```

However, the decreased complexity doesn't bring up the overall accuracy rate or individual precision rate as much as the updated k value. So I may stick to the new k value.

5. Decision Trees

5.1 Model Construction

```
placementTrain_tree$status <- as.factor(placementTrain_tree$status)
placementTest_tree$status <- as.factor(placementTest_tree$status)
# Build the decision tree
myTree <- C5.0(status ~ ., data = placementTrain_tree, trials=1)
myTree
```

```
##
## Call:
## C5.0.formula(formula = status ~ ., data = placementTrain_tree, trials = 1)
##
## Classification Tree
## Number of samples: 150
## Number of predictors: 12
##
## Tree size: 10
##
## Non-standard options: attempt to group attributes
```

```
# Predict with the tree model
mytree_predict <- predict(myTree, newdata = placementTest_tree, type="class")
```

5.2 Model Evaluation

5.2.1 Confusion Matrix

```
confusionMatrix(factor(mytree_predict, levels=c("Placed", "Not Placed")), factor(placementTest_tree$sta
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   Placed Not Placed
##   Placed         36          9
##   Not Placed      3         17
##
##                Accuracy : 0.8154
##                  95% CI : (0.6997, 0.9008)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.0001732
##
##                   Kappa : 0.6
##
##  Mcnemar's Test P-Value : 0.1489147
##
##             Sensitivity : 0.9231
##             Specificity : 0.6538
```

```
##          Pos Pred Value : 0.8000
##          Neg Pred Value : 0.8500
##              Prevalence : 0.6000
##          Detection Rate : 0.5538
##    Detection Prevalence : 0.6923
##       Balanced Accuracy : 0.7885
##
##        'Positive' Class : Placed
##
```

The Decision Tree classifer instantly has a higher accuracy rate, reaching 81.5%. Looking closer, 92.3% of "Placed" class is classified correctly while 65.4% of "Not Placed" is predicted correctly. The "Placed" label is more correctly predicted because it is the dominant class, as 60% of the test labels are that. But either way, the individual precision rate is already higher than that of kNN classifier, whether original or improved.

5.2.2 k-fold CV

```r
set.seed(111)
folds <- createFolds(placement$status, k=10)

cv_results <- lapply(folds, function(x){
  placement_train <- placement[ -x, ]
  placement_test <- placement[x, ]
  placement_train$status <- as.factor(placement_train$status)
  placement_test$status <- as.factor(placement_test$status)
  placement_model <- C5.0(status ~., data=placement_train)
  placement_pred <- predict(placement_model, placement_test)
  placement_actual <- placement_test$status
  #kappa <- kappa2(data.frame(placement_actual, placement_pred))$value
  agree <- table(placement_pred == placement_actual)
  #print(agree)
  accuracy <- agree[['TRUE']]/nrow(placement_test)
  #print(nrow(placement_test))

  return(accuracy)
})

cv_results
```

```
## $Fold01
## [1] 0.6818182
##
## $Fold02
## [1] 0.6
##
## $Fold03
## [1] 0.9545455
##
## $Fold04
## [1] 0.8181818
##
## $Fold05
```

```
## [1] 0.9090909
##
## $Fold06
## [1] 0.8636364
##
## $Fold07
## [1] 0.8636364
##
## $Fold08
## [1] 0.7619048
##
## $Fold09
## [1] 0.7727273
##
## $Fold10
## [1] 0.9
```

```
# the average of all kappas
# Question: I keep getting negative kappas score
mean(unlist(cv_results))
```

```
## [1] 0.8125541
```

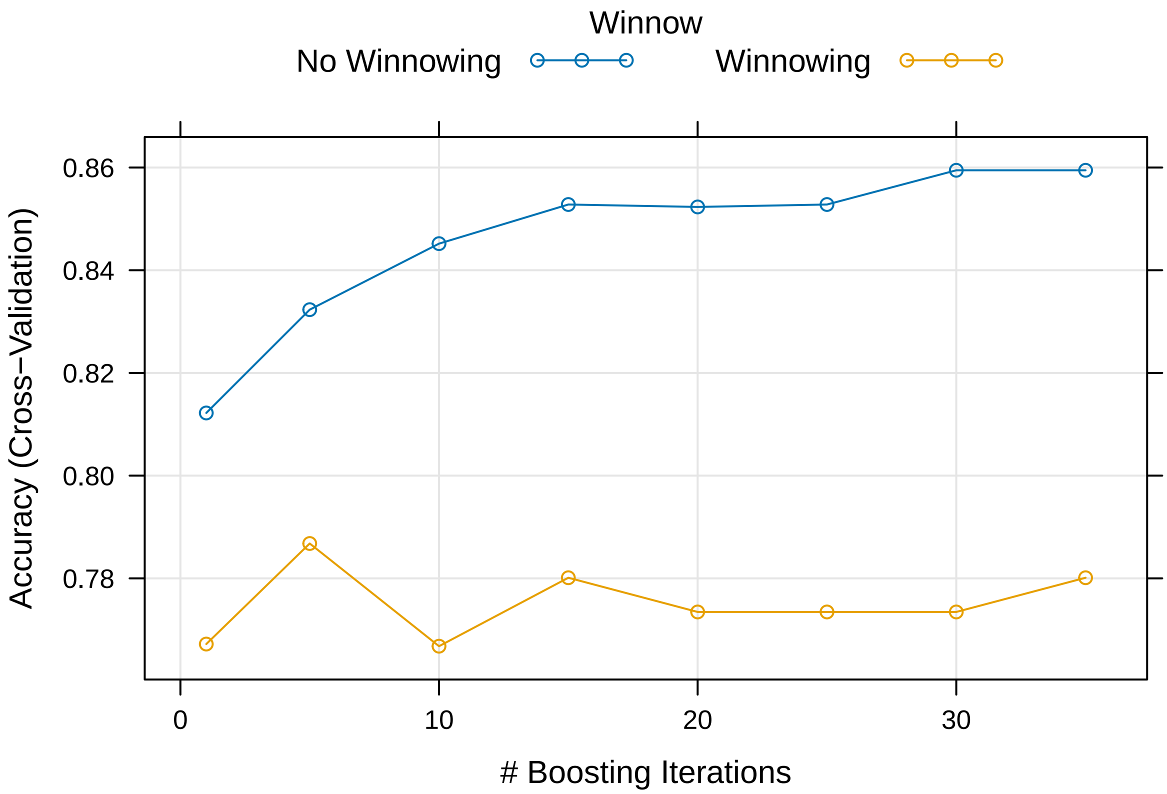
As shown above, even after the 10-fold cross validation, the decision tree classifier can still reach an accuracy rate of 81.25% on average.

5.3 Model Tuning & Performance Improvement

5.3.1 Hyperparameters

The Decision Trees have three hyperparameters: model, trials, and winnow. Here I will make trials and winnow part of the control object to tune the model.

```
set.seed(111)
library(mlbench)

fitControl <- trainControl(method="cv",
                           number = 10,
                           selectionFunction="oneSE")

grid <- expand.grid(model="tree", trials=c(1,5,10,15,20,25,30,35), winnow=c(TRUE,FALSE))

set.seed(111)
mdl <- train(status ~., data=placementTrain_tree, method="C5.0", metric="accuracy", trControl=fitControl

plot(mdl)
```



As shown above, when setting winnow to "False" and trials to 30, the accuracy rate of decision tree classifier can reach the highest at around 86%.

5.3.2 Mistake Cost

I can also add mistake costs to boost the accuracy. Since the "Not Placed" class is more easily to be classified as "Placed", I will add more mistake cost to that accordingly.

```
matrix_dimensions <- list(c("Placed", "Not Placed"), c("Placed", "Not Placed"))
names(matrix_dimensions) <- c("predicted", "actual")
error_cost <- matrix(c(0,5,8,0), nrow=2, dimnames=matrix_dimensions)
error_cost
```

```
##              actual
## predicted     Placed Not Placed
##   Placed           0          8
##   Not Placed       5          0
```

```
mytree_cost <- C5.0(status ~ ., data = placementTrain_tree, costs=error_cost)
mytree_cost_pred <- predict(mytree_cost, newdata = placementTest_tree)

confusionMatrix(factor(mytree_cost_pred, levels=c("Placed","Not Placed")), factor(placementTest_tree$sta
```

```
## Confusion Matrix and Statistics
##
```

```
##             Reference
## Prediction   Placed Not Placed
##   Placed         36          8
##   Not Placed      3         18
##
##                Accuracy : 0.8308
##                  95% CI : (0.7173, 0.9124)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 5.522e-05
##
##                   Kappa : 0.6358
##
##  Mcnemar's Test P-Value : 0.2278
##
##             Sensitivity : 0.9231
##             Specificity : 0.6923
##          Pos Pred Value : 0.8182
##          Neg Pred Value : 0.8571
##              Prevalence : 0.6000
##          Detection Rate : 0.5538
##    Detection Prevalence : 0.6769
##       Balanced Accuracy : 0.8077
##
##        'Positive' Class : Placed
##
```

And it does boost the overall accuracy a little bit, although the precision rate for "Not Placed" is not lower than the other. I guess the solution might be collecting more "Not Placed" instances in the data.

6. SVM

6.1 Model Construction

```r
# train the model
svm_classifier <- ksvm(status ~., data=placementTrain_svm, kernel="vanilladot")
```

```
##  Setting default kernel parameters
```

```r
# predict with the model
svm_predictions <- predict(svm_classifier, placementTest_svm)
```

6.2 Model Evaluation

```r
confusionMatrix(factor(svm_predictions, levels=c("Placed", "Not Placed")),
                factor(placementTest_svm$status, levels=c("Placed","Not Placed")), positive="Placed")
```

```
## Confusion Matrix and Statistics
```

```
##
##              Reference
## Prediction   Placed Not Placed
##   Placed         35         10
##   Not Placed      4         16
##
##                Accuracy : 0.7846
##                  95% CI : (0.6651, 0.8769)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.001288
##
##                   Kappa : 0.5333
##
##  Mcnemar's Test P-Value : 0.181449
##
##             Sensitivity : 0.8974
##             Specificity : 0.6154
##          Pos Pred Value : 0.7778
##          Neg Pred Value : 0.8000
##              Prevalence : 0.6000
##          Detection Rate : 0.5385
##    Detection Prevalence : 0.6923
##       Balanced Accuracy : 0.7564
##
##        'Positive' Class : Placed
##
```

6.3 Model Tuning & Performance Improvement

```
# train the model
svm_classifier_rbf <- ksvm(status ~., data=placementTrain_svm, kernel="rbfdot")
# predict with the model
svm_predictions_rbf <- predict(svm_classifier_rbf, placementTest_svm)
# calculate agreement
agreement_rbf <- svm_predictions_rbf == placementTest_svm$status
table(agreement_rbf)
```

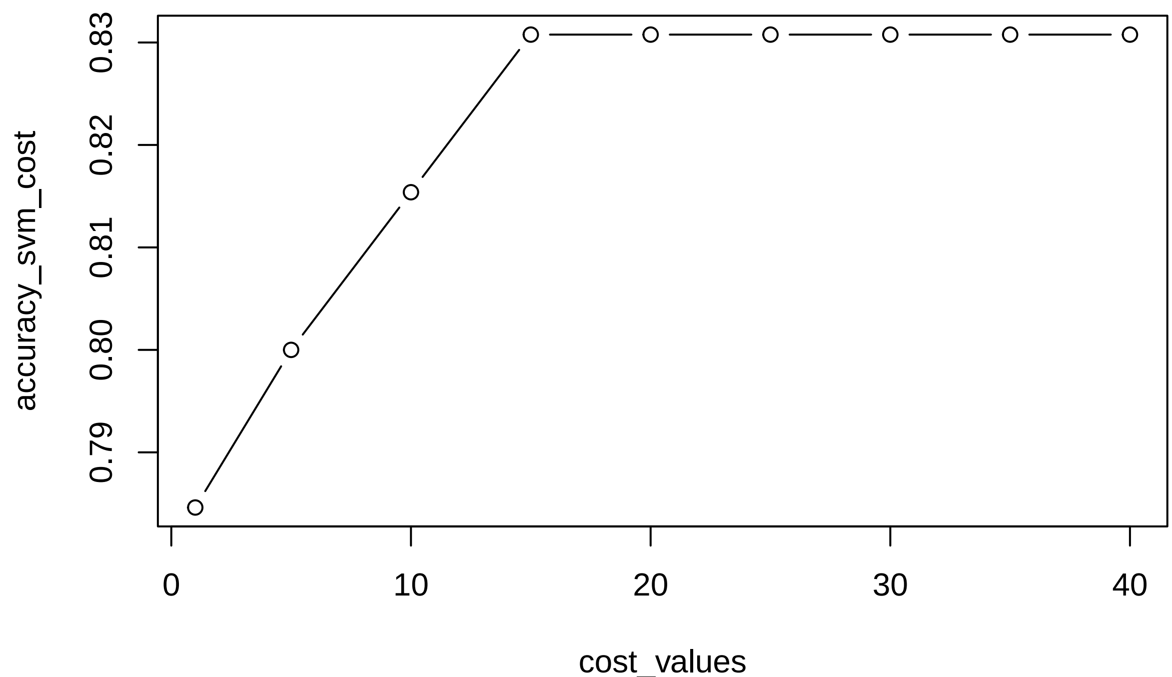
```
## agreement_rbf
## FALSE  TRUE
##    12    53
```

The accuracy rate is 74.6% after changing to radial basis, down from 78.5%.

```
cost_values <- c(1, seq(from=5, to=40, by=5))
accuracy_svm_cost <- sapply(cost_values, function(x){
  set.seed(111)
  m <- ksvm(status ~., data=placementTrain_svm, kernel="vanilladot", C=x)
  pred <- predict(m, placementTest_svm)
  agree <- table(pred == placementTest_svm$status)
  accuracy <- agree[['TRUE']]/nrow(placementTest_svm)
  return(accuracy)
})
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```r
plot(cost_values, accuracy_svm_cost, type="b")
```



As shown above, the cost parameters of SVM model can only boost the accuracy rate to 83% when C is set to 15.

7. Ensemble Model

7.1 Homogeneous ensemble

I will be using random forest as my homogeneous ensemble method.

```r
set.seed(111)

# construct random forests model
```

```r
placement_rf <- randomForest(status ~., data=placementTrain_tree, ntree=100)
placement_rf_predict <- predict(placement_rf, newdata=placementTest_tree)
# evaluate the model
confusionMatrix(factor(placement_rf_predict, levels=c("Placed","Not Placed")), factor(placementTest_tre
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   Placed Not Placed
##   Placed         37         10
##   Not Placed      2         16
##
##                Accuracy : 0.8154
##                  95% CI : (0.6997, 0.9008)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.0001732
##
##                   Kappa : 0.5946
##
##  Mcnemar's Test P-Value : 0.0433081
##
##             Sensitivity : 0.9487
##             Specificity : 0.6154
##          Pos Pred Value : 0.7872
##          Neg Pred Value : 0.8889
##              Prevalence : 0.6000
##          Detection Rate : 0.5692
##    Detection Prevalence : 0.7231
##       Balanced Accuracy : 0.7821
##
##        'Positive' Class : Placed
##
```

```r
tree_values <- c(50,60,70,80,90,100,110,120,130,140,150)
tree_this <- c()
accuracy_rf_list <- c()

for (tree_val in tree_values){
  set.seed(111)
  placement_rf <- randomForest(status ~., data=placementTrain_tree, ntree=tree_val)
  placement_rf_predict <- predict(placement_rf, newdata=placementTest_tree)

  cm <- confusionMatrix(factor(placement_rf_predict, levels=c("Placed", "Not Placed")), factor(placemen
  accuracy_val <- cm$overall[['Accuracy']]
  accuracy_rf_list <- c( accuracy_val, accuracy_rf_list)
  tree_this <- c(tree_val, tree_this)
}

rf_improved_accuracy <- data.frame(
  k_values = tree_this,
  accuracy_rate = accuracy_rf_list,
  stringsAsFactors = FALSE
)
```

```
rf_improved_accuracy
```

```
##    k_values accuracy_rate
## 1       150     0.8153846
## 2       140     0.8153846
## 3       130     0.8153846
## 4       120     0.8153846
## 5       110     0.8153846
## 6       100     0.8153846
## 7        90     0.8153846
## 8        80     0.8153846
## 9        70     0.8153846
## 10       60     0.8153846
## 11       50     0.8153846
```

As shown above, the adjustment of tree number doesn't affect the random forest classifier much...

7.2 Heterogeneous ensemble

```
set.seed(111)

# Model construction
placement_ensemble <- function(i, h, j){
  # knn
  placement_knn_pred <- knn(train=placementTrain.standardized,
                            test=i,
                            cl=placementTrain.labels,
                            k=10)

  # Decision Trees
  myTree <- C5.0(status ~ ., data = placementTrain_tree, costs=error_cost, trials=1)
  mytree_predict <- predict(myTree, newdata = h, type="class")

  # SVM
  m <- ksvm(status ~., data=placementTrain_svm, kernel="vanilladot", C=15)
  svm_pred <- predict(m, j)

  # merge all the prediction into a dataframe
  predictOutcome <- data.frame(
                               model1 = placement_knn_pred,
                               model2 = mytree_predict,
                               model3 = svm_pred)

  # select the most frequent one from each row
  finalPredictOutcome <- apply(predictOutcome, 1, function(x) names(which.max(table(x))))

  return (finalPredictOutcome)
}
```

```r
# Model prediction
placement_ensemble_predict <- placement_ensemble(placementTest.standardized, placementTest_tree, placeme
```

```
##  Setting default kernel parameters
```

```r
# evaluate the model
confusionMatrix(factor(placement_ensemble_predict, levels=c("Placed","Not Placed")), factor(placementTes
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction   Placed Not Placed
##   Placed         38         10
##   Not Placed      1         16
##
##                Accuracy : 0.8308
##                  95% CI : (0.7173, 0.9124)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 5.522e-05
##
##                   Kappa : 0.6259
##
##  Mcnemar's Test P-Value : 0.01586
##
##             Sensitivity : 0.9744
##             Specificity : 0.6154
##          Pos Pred Value : 0.7917
##          Neg Pred Value : 0.9412
##              Prevalence : 0.6000
##          Detection Rate : 0.5846
##    Detection Prevalence : 0.7385
##       Balanced Accuracy : 0.7949
##
##        'Positive' Class : Placed
##
```

As shown above, the heterogeneous ensemble model can result in 83.1% overall accuracy rate.