Data Science with R Master Program
Nurlaida
Assessment: College Admission

DESCRIPTION

Background and Objective:
Every year thousands of applications are being submitted by international students for admission in colleges of the USA. It becomes an iterative task for the Education Department to know the total number of applications received and then compare that data with the total number of applications successfully accepted and visas processed. Hence to make the entire process easy, the education department in the US analyze the factors that influence the admission of a student into colleges. The objective of this exercise is to analyze the same.

Domain: Education

Dataset Description:

Attribute	Description
GRE	Graduate Record Exam Scores
GPA	Grade Point Average
Rank	It refers to the prestige of the undergraduate institution. The variable rank takes on the values 1 through 4. Institutions with a rank of 1 have the highest prestige, while those with a rank of 4 have the lowest.
Admit	It is a response variable; admit/don't admit is a binary variable where 1 indicates that student is admitted and 0 indicates that student is not admitted.
SES	SES refers to socioeconomic status: 1 - low, 2 - medium, 3 - high.
Gender_male	Gender_male (0, 1) = 0 -> Female, 1 -> Male
Race	Race – 1, 2, and 3 represent Hispanic, Asian, and African-American

Analysis Tasks: Analyze the historical data and determine the key drivers for admission.
Predictive:

1. **Find the missing values. (if any, perform missing value treatment)**

   Run the packages
   Code:
   ```
   library(dplyr)
   library("RColorBrewer")
   library(randomForest)
   library(caret)
   library(caTools)
   library(rpart)
   ```

```
library(rpart.plot)
library(FSelector)
library(data.tree)
library(ggpubr)
```

Set working directory

```
setwd("G:/Data Science/R/Projects/College Admission")
getwd()
```

```
> #set working directory
> setwd("G:/Data Science/R/Projects/College Admission")
> getwd()
[1] "G:/Data Science/R/Projects/College Admission"
>
```

Import and explore data

Code:

```
college_admission <- read.csv("College_admission.csv")
View(college_admission)
str(college_admission)
class(college_admission)
summary(college_admission)
```

```
> #import and explore data
> college_admission <- read.csv("College_admission.csv")
> View(college_admission)
> str(college_admission)
'data.frame':	400 obs. of  7 variables:
 $ admit      : int  0 1 1 1 0 1 1 0 1 0 ...
 $ gre        : int  380 660 800 640 520 760 560 400 540 700 ...
 $ gpa        : num  3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
 $ ses        : int  1 2 2 1 3 2 2 2 1 1 ...
 $ Gender_Male: int  0 0 0 1 1 1 1 0 1 0 ...
 $ Race       : int  3 2 2 2 2 1 2 2 1 2 ...
 $ rank       : int  3 3 1 4 4 2 1 2 3 2 ...
> class(college_admission)
[1] "data.frame"
> summary(college_admission)
     admit             gre             gpa             ses         Gender_Male         Race            rank      
 Min.   :0.0000   Min.   :220.0   Min.   :2.260   Min.   :1.000   Min.   :0.000   Min.   :1.000   Min.   :1.000  
 1st Qu.:0.0000   1st Qu.:520.0   1st Qu.:3.130   1st Qu.:1.000   1st Qu.:0.000   1st Qu.:1.000   1st Qu.:2.000  
 Median :0.0000   Median :580.0   Median :3.395   Median :2.000   Median :0.000   Median :2.000   Median :2.000  
 Mean   :0.3175   Mean   :587.7   Mean   :3.390   Mean   :1.992   Mean   :0.475   Mean   :1.962   Mean   :2.485  
 3rd Qu.:1.0000   3rd Qu.:660.0   3rd Qu.:3.670   3rd Qu.:3.000   3rd Qu.:1.000   3rd Qu.:3.000   3rd Qu.:3.000  
 Max.   :1.0000   Max.   :800.0   Max.   :4.000   Max.   :3.000   Max.   :1.000   Max.   :3.000   Max.   :4.000  
>
```

Find the missing values. (if any, perform missing value treatment)

Code:

```
is.na(college_admission)
colSums(is.na(college_admission))
```

```
> #Find the missing values. (if any, perform missing value treatment)
> is.na(college_admission)
       admit   gre   gpa   ses Gender_Male  Race  rank
  [1,] FALSE FALSE FALSE FALSE       FALSE FALSE FALSE
  [2,] FALSE FALSE FALSE FALSE       FALSE FALSE FALSE
  [3,] FALSE FALSE FALSE FALSE       FALSE FALSE FALSE
  [4,] FALSE FALSE FALSE FALSE       FALSE FALSE FALSE
  [5,] FALSE FALSE FALSE FALSE       FALSE FALSE FALSE
  [6,] FALSE FALSE FALSE FALSE       FALSE FALSE FALSE
[141,] FALSE FALSE FALSE FALSE       FALSE FALSE FALSE
[142,] FALSE FALSE FALSE FALSE       FALSE FALSE FALSE
 [ reached getOption("max.print") -- omitted 258 rows ]
> colSums(is.na(college_admission))
      admit         gre         gpa         ses Gender_Male        Race        rank
          0           0           0           0           0           0           0
> |
```

Comment: From the output, there is no missing values

Checking empty values

Code:

```
colSums(college_admission==' ')
```

```
> #checking empty values
> colSums(college_admission==' ')
      admit         gre         gpa         ses Gender_Male        Race        rank
          0           0           0           0           0           0           0
> |
```

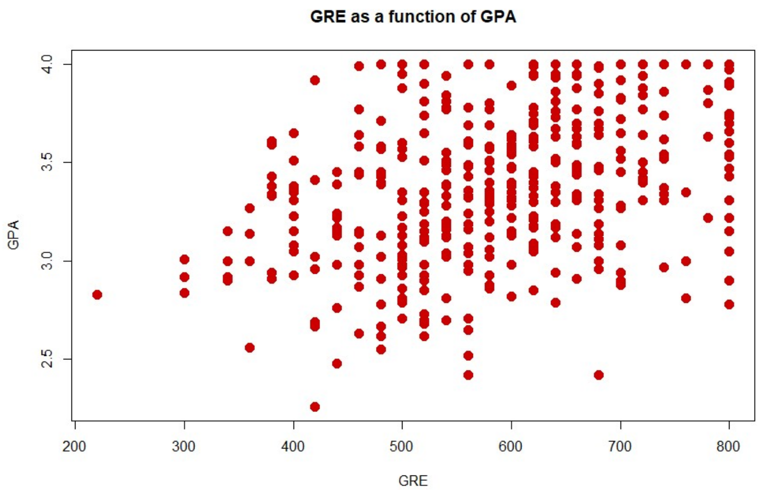
Comment: From the output, there is no empty values

2. **Find outliers (if any, then perform outlier treatment)**

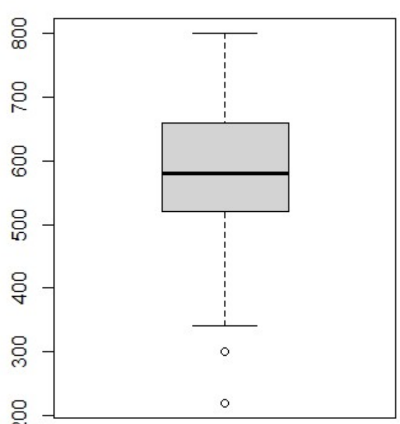
Code:

```
plot(college_admission$gre, college_admission$gpa,
     pch = 19,         # Solid circle
     cex = 1.5,        # Make 150% size
     col = "#cc0000",  # Red
     main = "GRE as a function of GPA",
     xlab = "GRE",
     ylab = "GPA")

boxplot(college_admission$gre)
boxplot(college_admission$gpa)
boxplot(college_admission$rank)
```

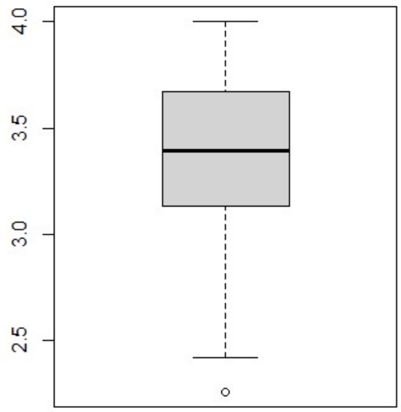
```
> #Find outliers (if any, then perform outlier treatment)
> plot(college_admission$gre, college_admission$gpa,
+      pch = 19,          # Solid circle
+      cex = 1.5,         # Make 150% size
+      col = "#cc0000",   # Red
+      main = "GRE as a function of GPA",
+      xlab = "GRE",
+      ylab = "GPA")
> boxplot(college_admission$rank) #there is one outlier for rank
> #Find outliers (if any, then perform outlier treatment)
> plot(college_admission$gre, college_admission$gpa,
+      pch = 19,          # Solid circle
+      cex = 1.5,         # Make 150% size
+      col = "#cc0000",   # Red
+      main = "GRE as a function of GPA",
+      xlab = "GRE",
+      ylab = "GPA")
> boxplot(college_admission$gre) #there are two outliers for gre
> boxplot(college_admission$gpa) #there is one outlier for gpa
> boxplot(college_admission$rank) #there is no outlier for rank
>
```



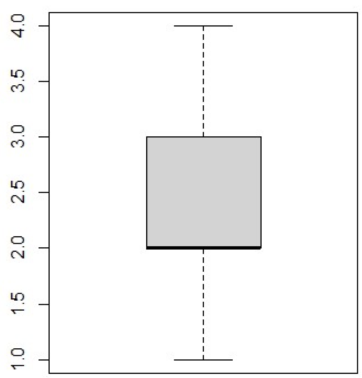
Boxplot gre



Boxplot gpa



Boxplot rank



Comment:
There are two outliers for gre
There is one outlier for gpa
There is no outlier for rank

Removing outliers from gre
Code:

```
college_admission1 <- college_admission
bench_gre <- 520 - 1.5*IQR(college_admission1$gre)
bench_gre
college_admission1 <- filter(college_admission1, gre > 310)
boxplot(college_admission1$gre)
```

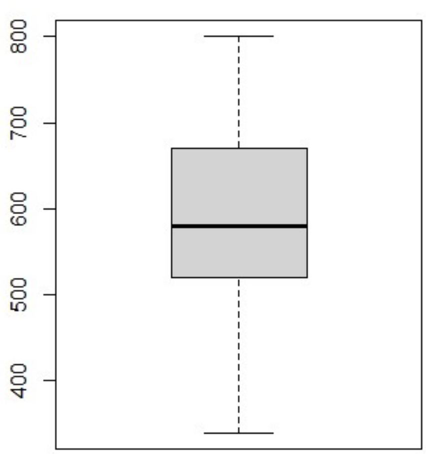
Removing outliers from gpa

Code:

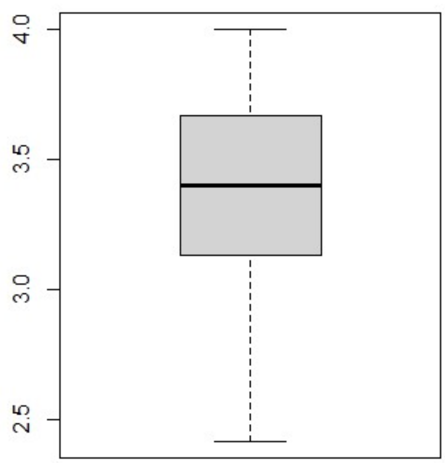
```
bench_gpa <- 3.13 - 1.5*IQR(college_admission1$gpa)
bench_gpa
college_admission1 <- filter(college_admission1, gpa > 2.32)
boxplot(college_admission1$gpa)

summary(college_admission1)
```

```
> #removing outliers from gre
> college_admission1 <- college_admission
> bench_gre <- 520 - 1.5*IQR(college_admission1$gre)
> bench_gre
[1] 310
> college_admission1 <- filter(college_admission1, gre > 310)
> boxplot(college_admission1$gre)
> #removing outliers from gpa
> bench_gpa <- 3.13 - 1.5*IQR(college_admission1$gpa)
> bench_gpa
[1] 2.32
> college_admission1 <- filter(college_admission1, gpa > 2.32)
> boxplot(college_admission1$gpa)
> summary(college_admission1)
     admit            gre             gpa             ses          Gender_Male          Race      
 Min.   :0.000   Min.   :340.0   Min.   :2.420   Min.   :1.000   Min.   :0.0000   Min.   :1.000  
 1st Qu.:0.000   1st Qu.:520.0   1st Qu.:3.135   1st Qu.:1.000   1st Qu.:0.0000   1st Qu.:1.000  
 Median :0.000   Median :580.0   Median :3.400   Median :2.000   Median :0.0000   Median :2.000  
 Mean   :0.319   Mean   :591.2   Mean   :3.398   Mean   :1.995   Mean   :0.4709   Mean   :1.967  
 3rd Qu.:1.000   3rd Qu.:670.0   3rd Qu.:3.670   3rd Qu.:3.000   3rd Qu.:1.0000   3rd Qu.:3.000  
 Max.   :1.000   Max.   :800.0   Max.   :4.000   Max.   :3.000   Max.   :1.0000   Max.   :3.000  
      rank      
 Min.   :1.000  
 1st Qu.:2.000  
 Median :2.000  
 Mean   :2.476  
 3rd Qu.:3.000  
 Max.   :4.000  
> 
```

Boxplot gre with no outliers



Boxplot gpa with no outliers



3. **Find whether the data is normally distributed or not. Use the plot to determine the same.**

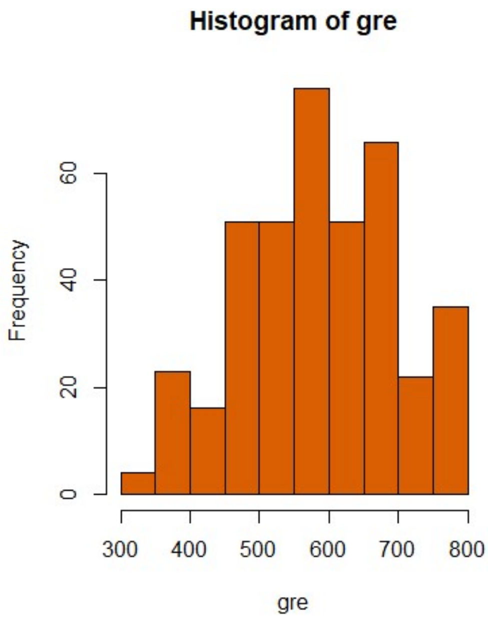
   Code:

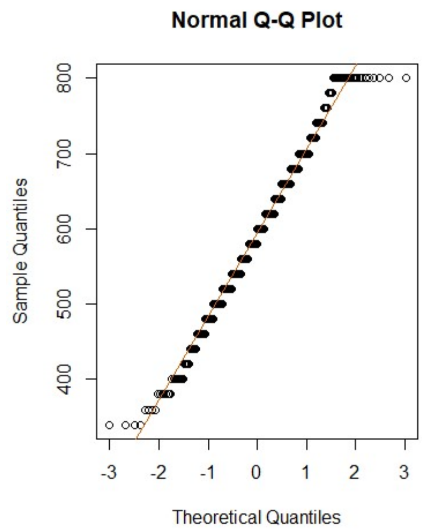
```
#gre
hist(college_admission1$gre,
     xlab = 'gre',
     main = 'Histogram of gre',
     col = '#D95F02')

qqnorm(college_admission1$gre)
qqline(college_admission1$gre, col = '#D95F02')
ggdensity(college_admission1$gre, main="gre", xlab = "gre disrtibution")

gr <- college_admission1$gre
plotNormalHistogram(gr)
#gre is normally distributed

#gpa
hist(college_admission1$gpa,
     xlab = 'gpa',
     main = 'Histogram of gpa',
     col = '#1B9E77')

qqnorm(college_admission1$gpa)
qqline(college_admission1$gpa, col = '#1B9E77')
ggdensity(college_admission1$gpa, main="gpa", xlab = "gpa disrtibution")
```

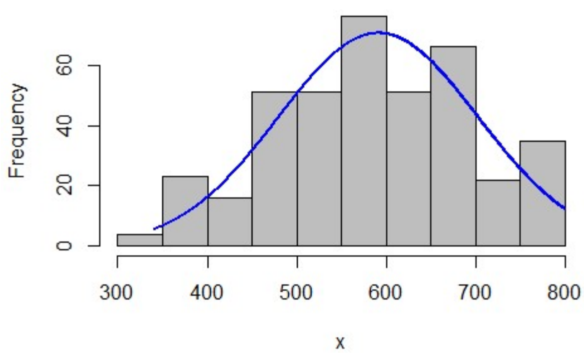
```
gp <- college_admission1$gpa
plotNormalHistogram(gp)
#gpa is normally distributed

#rank
hist(college_admission1$rank,
     xlab = 'rank',
     main = 'Histogram of rank',
     col = '#1B9E77')

qqnorm(college_admission1$rank)

rk <- college_admission$rank
plotNormalHistogram(rk)
#rank is normally distributed

#Admit
qqnorm(college_admission1$admit)
```

```
> #Find whether the data is normally distributed or not. Use the plot to determine the same.
> #gre
> hist(college_admission1$gre,
+      xlab = 'gre',
+      main = 'Histogram of gre',
+      col = '#D95F02')
> qqnorm(college_admission1$gre)
> qqline(college_admission1$gre, col = '#D95F02')
> ggdensity(college_admission1$gre, main="gre", xlab = "gre disrtibution")
> gr <- college_admission1$gre
> plotNormalHistogram(gr)
> #gpa
> hist(college_admission1$gpa,
+      xlab = 'gpa',
+      main = 'Histogram of gpa',
+      col = '#1B9E77')
> qqnorm(college_admission1$gpa)
> qqline(college_admission1$gpa, col = '#1B9E77')
> ggdensity(college_admission1$gpa, main="gpa", xlab = "gpa disrtibution")
> gp <- college_admission1$gpa
> plotNormalHistogram(gp)
> #rank
> hist(college_admission1$rank,
+      xlab = 'rank',
+      main = 'Histogram of rank',
+      col = '#1B9E77')
> qqnorm(college_admission1$rank)
> rk <- college_admission1$rank
> plotNormalHistogram(rk)
```

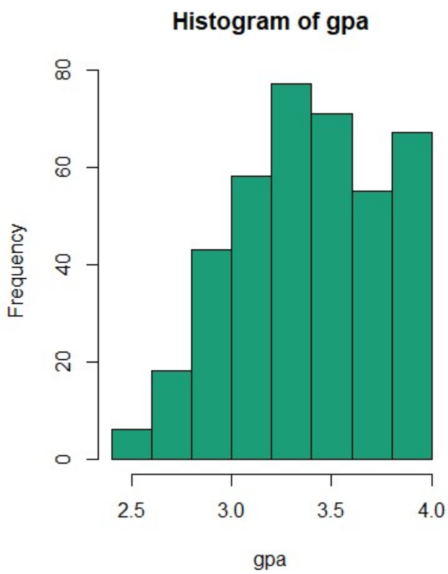
GRE Plots


Histogram of gre
Frequency
0
20
40
60
300
400
500
600
700
800
gre

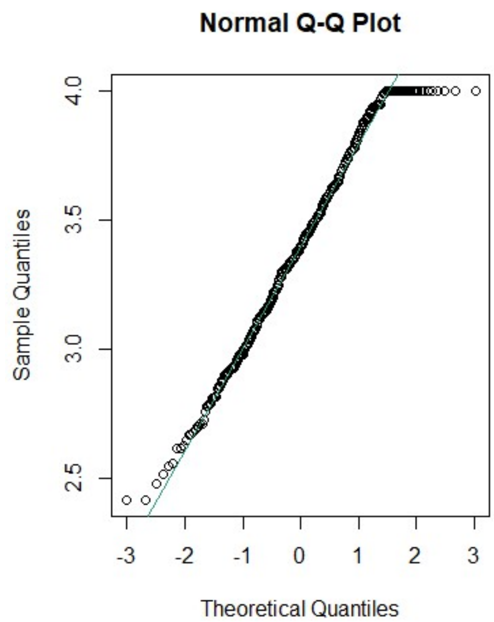

Normal Q-Q Plot
Sample Quantiles
400
500
600
700
800
-3
-2
-1
0
1
2
3
Theoretical Quantiles


Frequency
0
20
40
60
300
400
500
600
700
800
x

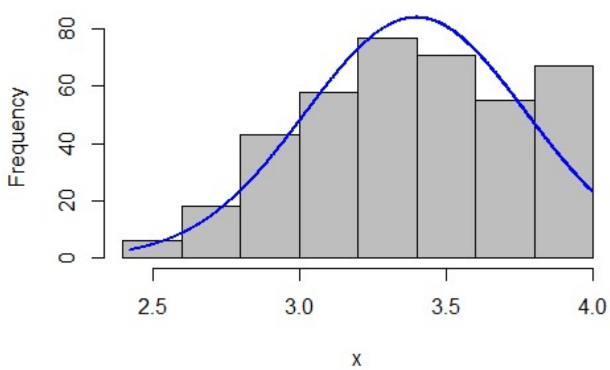
GPA Plots



Histogram of gpa
Frequency
0
20
40
60
80
2.5
3.0
3.5
4.0
gpa

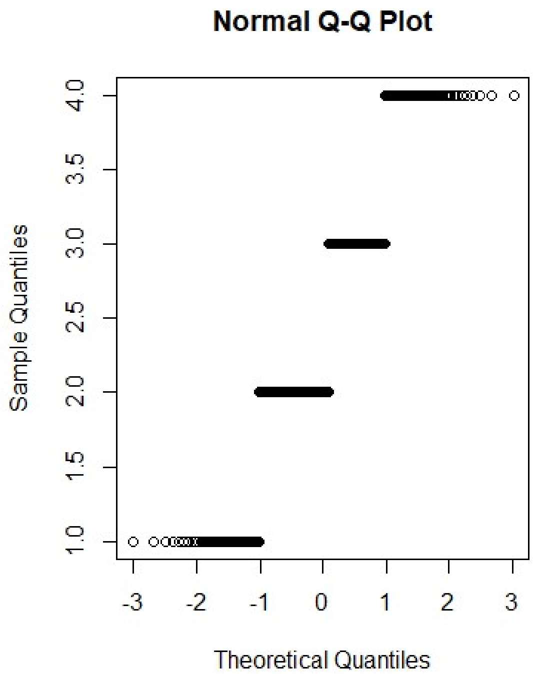


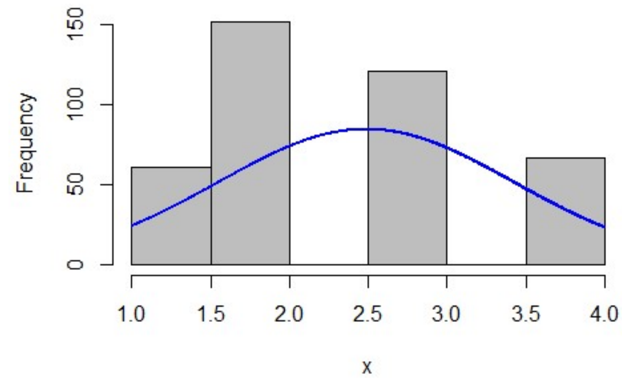
Normal Q-Q Plot
Sample Quantiles
2.5
3.0
3.5
4.0
-3
-2
-1
0
1
2
3
Theoretical Quantiles



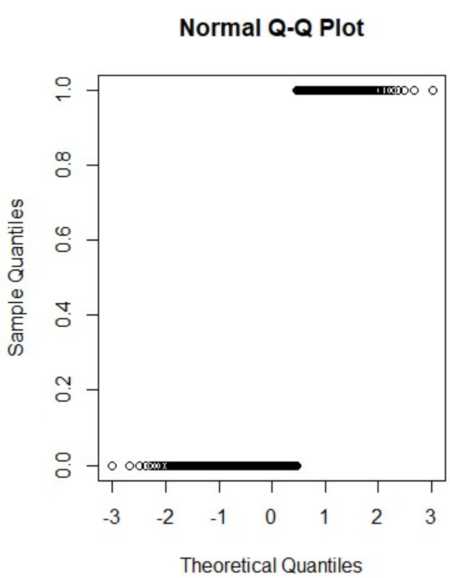
Frequency
0
20
40
60
80
2.5
3.0
3.5
4.0
x

Rank Plot





Admit Plot



Comment:

- gre is normally distributed
- gpa is normally distributed
- rank is normally distributed

4. **Find the structure of the data set and if required, transform the numeric data type to factor and vice-versa.**

   Code:
```
View(college_admission1)
str(college_admission1)
college_admission1$rank <- as.factor(college_admission1$rank) #transform
rank into factor data type
college_admission1$admit <- as.factor(college_admission1$admit) #transform
admit into factor data type
```

```
> #Find the structure of the data set and if required, transform the numeric data type to factor and vice
-versa.
> View(college_admission1)
> str(college_admission1)
'data.frame':	395 obs. of  7 variables:
 $ admit      : int  0 1 1 1 0 1 1 0 1 0 ...
 $ gre        : int  380 660 800 640 520 760 560 400 540 700 ...
 $ gpa        : num  3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
 $ ses        : int  1 2 2 1 3 2 2 2 1 1 ...
 $ Gender_Male: int  0 0 0 1 1 1 1 0 1 0 ...
 $ Race       : int  3 2 2 2 2 1 2 2 1 2 ...
 $ rank       : int  3 3 1 4 4 2 1 2 3 2 ...
> college_admission1$rank <- as.factor(college_admission1$rank) #transform rank into factor data type
> college_admission1$admit <- as.factor(college_admission1$admit) #transform admit into factor data type
> str(college_admission1)
'data.frame':	395 obs. of  7 variables:
 $ admit      : Factor w/ 2 levels "0","1": 1 2 2 2 1 2 2 1 2 1 ...
 $ gre        : int  380 660 800 640 520 760 560 400 540 700 ...
 $ gpa        : num  3.61 3.67 4 3.19 2.93 3 2.98 3.08 3.39 3.92 ...
 $ ses        : int  1 2 2 1 3 2 2 2 1 1 ...
 $ Gender_Male: int  0 0 0 1 1 1 1 0 1 0 ...
 $ Race       : int  3 2 2 2 2 1 2 2 1 2 ...
 $ rank       : Factor w/ 4 levels "1","2","3","4": 3 3 1 4 4 2 1 2 3 2 ...
> |
```

5. **Normalize the data if not normally distributed.**

   Comment: All data is normal

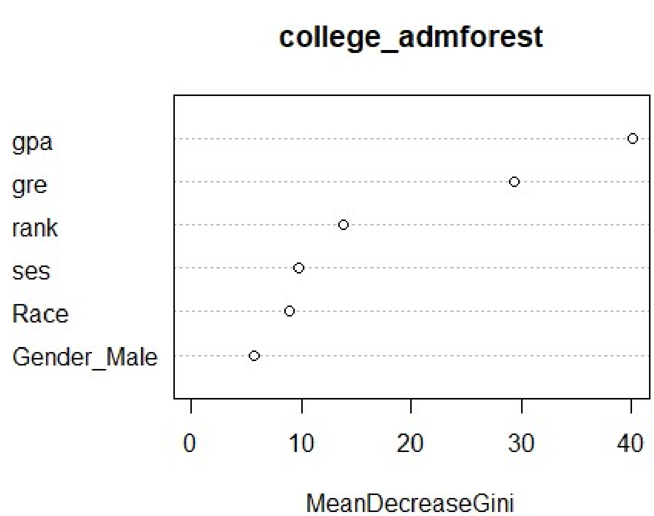
6. **Use variable reduction techniques to identify significant variables.**

   Code:
```
#In this case we use random forest
set.seed(123)
id <- sample(2, nrow(college_admission1), prob = c(0.7, 0.3), replace =
      TRUE)
colforest_train <- college_admission1[id==1,]
colforest_test <- college_admission1[id==2,]
str(colforest_train)

bestmtry <- tuneRF(colforest_train, colforest_train$admit,stepFactor =
            1.2, improve = 0.01, trace = TRUE, plot = TRUE)

college_admforest <- randomForest(admit~., data = colforest_train)
college_admforest

importance(college_admforest)
#gpa, gre, and rank are significant variable
```

```
varImpPlot(college_admforest)

pred_college <- predict(college_admforest, newdata = colforest_test, type
                   = "class")
pred_college

confusionMatrix(table(pred_college, colforest_test$admit))
```

```
> #In this case I use random forest
> set.seed(123)
> id <- sample(2, nrow(college_admission1), prob = c(0.7, 0.3), replace = TRUE)
> colforest_train <- college_admission1[id==1,]
> colforest_test <- college_admission1[id==2,]
> str(colforest_train)
'data.frame':   281 obs. of  7 variables:
 $ admit      : Factor w/ 2 levels "0","1": 1 2 2 2 2 1 1 2 1 2 ...
 $ gre        : int  380 800 760 560 540 700 440 760 700 700 ...
 $ gpa        : num  3.61 4 3 2.98 3.39 3.92 3.22 4 3.08 4 ...
 $ ses        : int  1 2 2 2 1 1 3 3 2 2 ...
 $ Gender_Male: int  0 0 1 1 1 0 0 1 0 1 ...
 $ Race       : int  3 2 1 2 1 2 2 2 2 1 ...
 $ rank       : Factor w/ 4 levels "1","2","3","4": 3 1 2 1 3 2 1 1 2 1 ...
> bestmtry <- tuneRF(colforest_train, colforest_train$admit,stepFactor = 1.2,
+                    improve = 0.01, trace = TRUE, plot = TRUE)
mtry = 2  OOB error = 0%
Searching left ...
Searching right ...
> college_admforest <- randomForest(admit~., data = colforest_train)
> college_admforest

Call:
 randomForest(formula = admit ~ ., data = colforest_train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 31.67%
Confusion matrix:
    0  1 class.error
0 176 17   0.0880829
1  72 16   0.8181818
> importance(college_admforest) #gpa, gre, and rank are significant variable
            MeanDecreaseGini
gre                29.330143
gpa                40.061191
ses                 9.750423
Gender_Male         5.650087
Race                8.911130
rank               13.773984
> varImpPlot(college_admforest)
```

```
> pred_college <- predict(college_admforest, newdata = colforest_test, type = "class")
> pred_college
  2   4   5   8  11  16  20  21  24  26  31  32  34  37  50  53  58  59  65  67  68  69  71  73  84 
  0   0   0   0   0   0   1   0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   0   1   0 
 87  88  89  97 104 106 107 111 114 115 118 126 132 134 137 138 139 145 150 151 167 173 174 179 181 
  0   0   0   0   0   0   0   0   1   0   1   1   0   0   0   0   1   0   1   0   0   0   0   0   0 
183 189 190 193 195 202 206 216 219 220 222 223 230 238 240 246 248 249 250 256 260 261 262 264 271 
  0   0   0   0   0   0   1   1   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   0 
275 276 277 281 294 295 296 297 300 301 303 304 313 316 317 320 321 324 327 330 333 334 340 347 352 
  0   0   0   0   0   0   0   0   0   1   0   0   1   0   0   0   1   1   0   0   0   0   0   0   1 
356 360 363 366 373 376 377 380 382 384 386 391 393 394 
  0   0   0   0   0   0   0   0   0   1   0   1   0   0 
Levels: 0 1
> confusionMatrix(table(pred_college, colforest_test$admit))
Confusion Matrix and Statistics

            
pred_college  0  1
           0 67 28
           1  9 10
                                          
               Accuracy : 0.6754          
                 95% CI : (0.5814, 0.7601)
    No Information Rate : 0.6667          
    P-Value [Acc > NIR] : 0.464828        
                                          
                  Kappa : 0.1654          
                                          
 Mcnemar's Test P-Value : 0.003085        
                                          
            Sensitivity : 0.8816          
            Specificity : 0.2632          
         Pos Pred Value : 0.7053          
         Neg Pred Value : 0.5263          
             Prevalence : 0.6667          
         Detection Rate : 0.5877          
   Detection Prevalence : 0.8333          
      Balanced Accuracy : 0.5724          
                                          
       'Positive' Class : 0               
                                          
> 
```



Comment:

According to RandomForest method, GPA, GRE, and Rank are the significant variables with accuracy is 67.54%.

7. **Run logistic model to determine the factors that influence the admission process of a student (Drop insignificant variables)**
8. **Calculate the accuracy of the model and run validation techniques.**

   Code: (for task no 7 and 8)

```
#Split the data set into training and testing model
split_logistic <- sample.split(college_admission1, SplitRatio = 0.8)
split_logistic
train_logistic <- subset(college_admission1, split = "TRUE")
test_logistic <- subset(college_admission1, split = "FALSE")

#Calculate the accuracy of the model and run validation techniques.
#Train the model, using independent variable: gre and gpa
college_model <- glm(admit ~ ., data = train_logistic, family =
                 'binomial')
summary(college_model)

#Train the model, using independent variable: gre, gpa, and rank
college_model1 <- glm(admit ~ gre + gpa + rank, data = train_logistic,
                 family = 'binomial')
summary(college_model1)

#Train the model, using independent variable: gre and gpa
college_model2 <- glm(admit ~ gre + gpa, data = train_logistic, family =
                 'binomial')
summary(college_model2)

#Train the model, using independent variable: gpa and rank
college_model3 <- glm(admit ~ gpa + rank, data = train_logistic, family =
                  'binomial')
summary(college_model3)

#Calculate the accuracy
res <- predict(college_model3, test_logistic, type = "response")
res
res <- predict(college_model3, train_logistic, type = "response")
res

#Validation Technique
confmatrix <- table(Actual_value=train_logistic$admit, Predicted_value =
              res > 0.5)
confmatrix

#Accuracy
(confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)
```

```
> #Run logistic model to determine the factors that influence the admission process of a student
> #(Drop insignificant variables)
> #Split the data set into training and testing model
> split_logistic <- sample.split(college_admission1, SplitRatio = 0.8)
> split_logistic
[1] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> train_logistic <- subset(college_admission1, split = "TRUE")
> test_logistic <- subset(college_admission1, split = "FALSE")
> #Calculate the accuracy of the model and run validation techniques.
> #Train the model, using independent variable: gre and gpa
> college_model <- glm(admit ~ ., data = train_logistic, family = 'binomial')
> summary(college_model)

Call:
glm(formula = admit ~ ., family = "binomial", data = train_logistic)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.7556  -0.8636  -0.6343   1.1511   2.0995

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.511223   1.214247  -2.892 0.003832 **
gre          0.002348   0.001120   2.096 0.036069 *
gpa          0.862203   0.337221   2.557 0.010564 *
ses         -0.166695   0.142488  -1.170 0.242047
Gender_Male -0.197759   0.230408  -0.858 0.390727
Race        -0.156908   0.139984  -1.121 0.262332
rank2       -0.702579   0.319380  -2.200 0.027819 *
rank3       -1.333781   0.348594  -3.826 0.000130 ***
rank4       -1.548823   0.421505  -3.675 0.000238 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 494.62  on 394  degrees of freedom
Residual deviance: 450.99  on 386  degrees of freedom
AIC: 468.99

Number of Fisher Scoring iterations: 4
```

```
> #Train the model, using independent variable: gre, gpa, and rank
> college_model1 <- glm(admit ~ gre + gpa + rank, data = train_logistic, family = 'binomial')
> summary(college_model1)

Call:
glm(formula = admit ~ gre + gpa + rank, family = "binomial",
    data = train_logistic)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.6444  -0.8682  -0.6394   1.1558   2.0886

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -4.131515   1.160348  -3.561 0.000370 ***
gre          0.002442   0.001114   2.191 0.028424 *
gpa          0.813733   0.333900   2.437 0.014807 *
rank2       -0.700406   0.317622  -2.205 0.027443 *
rank3       -1.331987   0.346262  -3.847 0.000120 ***
rank4       -1.535019   0.418909  -3.664 0.000248 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 494.62  on 394  degrees of freedom
Residual deviance: 454.09  on 389  degrees of freedom
AIC: 466.09

Number of Fisher Scoring iterations: 4

> #Train the model, using independent variable: gre and gpa
> college_model2 <- glm(admit ~ gre + gpa, data = train_logistic, family = 'binomial')
> summary(college_model2)

Call:
glm(formula = admit ~ gre + gpa, family = "binomial", data = train_logistic)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.2837  -0.9003  -0.7161   1.2993   1.9824

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) -5.06161    1.10078  -4.598 4.26e-06 ***
gre          0.00282    0.00108   2.611  0.00902 **
gpa          0.76297    0.32215   2.368  0.01787 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 494.62  on 394  degrees of freedom
Residual deviance: 475.08  on 392  degrees of freedom
AIC: 481.08

Number of Fisher Scoring iterations: 4
```

```
> #Train the model, using independent variable: gpa and rank
> college_model3 <- glm(admit ~ gpa + rank, data = train_logistic, family = 'binomial')
> summary(college_model3)

Call:
glm(formula = admit ~ gpa + rank, family = "binomial", data = train_logistic)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.5086  -0.8690  -0.6646   1.1570   2.0900

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -3.5067     1.1140  -3.148 0.001645 **
gpa           1.0646     0.3143   3.387 0.000705 ***
rank2        -0.7034     0.3148  -2.235 0.025447 *
rank3        -1.3790     0.3425  -4.026 5.68e-05 ***
rank4        -1.5745     0.4159  -3.786 0.000153 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 494.62  on 394  degrees of freedom
Residual deviance: 458.97  on 390  degrees of freedom
AIC: 468.97

Number of Fisher Scoring iterations: 4

> #Calculate the accuracy
> res <- predict(college_model3, test_logistic, type = "response")
> res
         1          2          3          4          5          6          7          8
0.26064479 0.27314050 0.67951089 0.15640030 0.12324615 0.26572612 0.41719054 0.28267081 0.21
```

```
> #Validation Technique
> confmatrix <- table(Actual_value=train_logistic$admit, Predicted_value = res > 0.5)
> confmatrix
            Predicted_value
Actual_value FALSE TRUE
           0   253   16
           1    95   31
> #Accuracy
> (confmatrix[[1,1]] + confmatrix[[2,2]]) / sum(confmatrix)
[1] 0.7189873
>
```

Comment:

From testing various models the best model with highest accuracy is college_model3 (gpa+rank) and it gives 71.89% accuracy.

9. Try other modelling techniques like decision tree and SVM and select a champion model.

10. Determine the accuracy rates for each kind of model

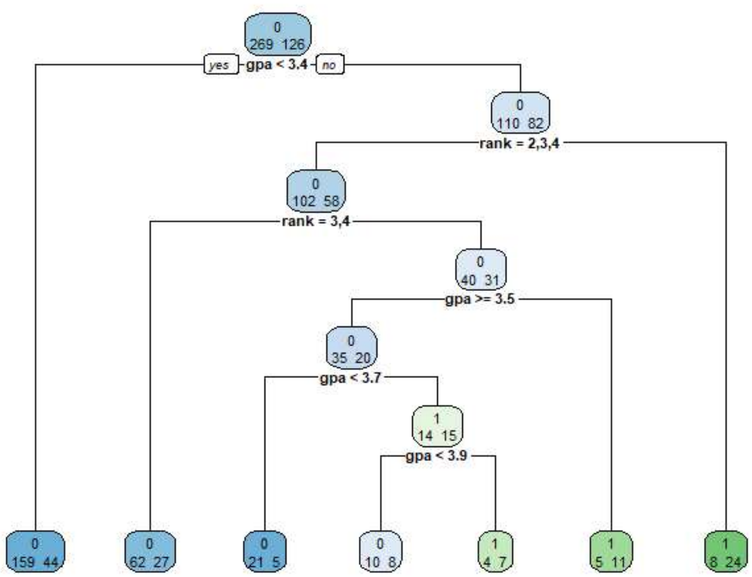
Code: (for number 9 and 10)

```
#Decision Tree
#Eliminating unmeaningful variable
college_admissionDT <- select(college_admission1, admit, gpa, rank)
```

```
#Split the data set into training and testing model
set.seed(123)
split_dt <- sample.split(college_admissionDT$admit, SplitRatio = 0.8)
split_dt
train_dt <- subset(college_admissionDT, split = "TRUE")
test_dt <- subset(college_admissionDT, split = "FALSE")

#Training Test
tree <- rpart(admit ~., data = train_dt)

#Prediction
tree.admit.predict <- predict(tree, test_dt, type = 'class')

#Confusion Matrix
confusionMatrix(tree.admit.predict, test_dt$admit)

prp(tree)
rpart.plot(tree,extra=1, cex=0.7)
```

```
> #Try other modeling techniques like decision tree and SVM and select a champion model
> #Decision Tree
> #Eliminating unmeaningful variable
> college_admissionDT <- select(college_admission1, admit, gpa, rank)
> #Split the data set into training and testing model
> set.seed(123)
> split_dt <- sample.split(college_admissionDT$admit, SplitRatio = 0.8)
> split_dt
  [1]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
 [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
 [45]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
 [67]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
 [89]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[111]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE
[133]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
[155] FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE
[177] FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[199]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
[221]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
[243]  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[265]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE
[287]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[309]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[331]  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[353]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
[375]  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> train_dt <- subset(college_admissionDT, split = "TRUE")
> test_dt <- subset(college_admissionDT, split = "FALSE")
> #Training Test
> tree <- rpart(admit ~., data = train_dt)
> #Prediction
> tree.admit.predict <- predict(tree, test_dt, type = 'class')
```

```
> #Confusion Matrix
> confusionMatrix(tree.admit.predict, test_dt$admit)
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 252  84
         1  17  42

               Accuracy : 0.7443
                 95% CI : (0.6983, 0.7866)
    No Information Rate : 0.681
    P-Value [Acc > NIR] : 0.003576

                  Kappa : 0.3146

 Mcnemar's Test P-Value : 5.125e-11

            Sensitivity : 0.9368
            Specificity : 0.3333
         Pos Pred Value : 0.7500
         Neg Pred Value : 0.7119
             Prevalence : 0.6810
         Detection Rate : 0.6380
   Detection Prevalence : 0.8506
      Balanced Accuracy : 0.6351

       'Positive' Class : 0

> prp(tree)
> rpart.plot(tree,extra=1, cex=0.7)
> |
```



Code:

```
#SVM
#Split the data set into training and testing model
set.seed(123)
partitionsvm <- createDataPartition(y = college_admission1$admit, p = 0.8, list = FALSE)
training_svm <- college_admission1[partitionsvm,]
testing_svm <- college_admission1[-partitionsvm,]
```

```
dim(training_svm)
dim(testing_svm)

#Train the method
control_svm <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
svm_linear <- train(admit~ gpa + rank, data = training_svm, method = "svmLinear",
              trControl = control_svm, preProcess = c("center", "scale"),
              tuneLength = 10)

#Testing the method
test_predsvm <- predict(svm_linear, newdata = testing_svm)
test_predsvm

#Validation
confusionMatrix(table(test_predsvm, testing_svm$admit))

#Improve Model Performance
grid <- expand.grid(C = c(0, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2))
set.seed(123)
svm_linear_grid <- train(admit~ gpa + rank, data = training_svm,
                   method = "svmLinear",
                   preProcess = c ("center", "scale"),
                   tuneGrid = grid,
                   tuneLength = 10)
svm_linear_grid
plot(svm_linear_grid)

test_pred_grid <- predict(svm_linear_grid, newdata = testing_svm)
test_pred_grid

confusionMatrix(table(test_pred_grid, testing_svm$admit))
```

```
> #SVM
> #Split the data set into training and testing model
> set.seed(123)
> partitionsvm <- createDataPartition(y = college_admission1$admit, p = 0.8, list = FALSE)
> training_svm <- college_admission1[partitionsvm,]
> testing_svm <- college_admission1[-partitionsvm,]
> dim(training_svm)
[1] 317   7
> dim(testing_svm)
[1] 78  7
> #Train the method
> control_svm <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
> svm_linear <- train(admit~ gpa + rank, data = training_svm, method = "svmLinear",
+                     trControl = control_svm, preProcess = c("center", "scale"),
+                     tuneLength = 10)
> #Testing the method
> test_predsvm <- predict(svm_linear, newdata = testing_svm)
> test_predsvm
 [1] 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 1 0 1 0 0 0 0 0 0
[70] 0 0 0 0 0 0 0 0 0
Levels: 0 1
> #Validation
> confusionMatrix(table(test_predsvm, testing_svm$admit))
Confusion Matrix and Statistics


test_predsvm  0  1
           0 49 22
           1  4  3

               Accuracy : 0.6667
                 95% CI : (0.5508, 0.7694)
    No Information Rate : 0.6795
    P-Value [Acc > NIR] : 0.6466290

                  Kappa : 0.055

 Mcnemar's Test P-Value : 0.0008561

            Sensitivity : 0.9245
            Specificity : 0.1200
         Pos Pred Value : 0.6901
         Neg Pred Value : 0.4286
             Prevalence : 0.6795
         Detection Rate : 0.6282
   Detection Prevalence : 0.9103
      Balanced Accuracy : 0.5223

       'Positive' Class : 0

> #Improve Model Performance
> grid <- expand.grid(C = c(0, 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2))
> set.seed(123)
> svm_linear_grid <- train(admit~ gpa + rank, data = training_svm,
+                          method = "svmLinear",
+                          preProcess = c ("center", "scale"),
+                          tuneGrid = grid,
+                          tuneLength = 10)
There were 27 warnings (use warnings() to see them)
> svm_linear_grid
Support Vector Machines with Linear Kernel

317 samples
  2 predictor
  2 classes: '0', '1'

Pre-processing: centered (4), scaled (4)
Resampling: Bootstrapped (25 reps)
Summary of sample sizes: 317, 317, 317, 317, 317, 317, ...
Resampling results across tuning parameters:

  C     Accuracy   Kappa
  0.00        NaN         NaN
  0.01  0.6839005  0.00000000
  0.05  0.6824785  0.03707409
  0.10  0.6834309  0.09248141
  0.25  0.6834309  0.09854201
  0.50  0.6834309  0.10541334
  0.75  0.6834309  0.10541334
  1.00  0.6834309  0.10541334
  1.25  0.6834309  0.10541334
  1.50  0.6834309  0.10541334
  1.75  0.6834309  0.10541334
  2.00  0.6834309  0.10541334

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.01.
> plot(svm_linear_grid)
> test_pred_grid <- predict(svm_linear_grid, newdata = testing_svm)
> test_pred_grid
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
[70] 0 0 0 0 0 0 0 0 0
Levels: 0 1
```

```
> confusionMatrix(table(test_pred_grid, testing_svm$admit))
Confusion Matrix and Statistics


test_pred_grid  0  1
             0 53 25
             1  0  0

               Accuracy : 0.6795
                 95% CI : (0.5642, 0.7807)
    No Information Rate : 0.6795
    P-Value [Acc > NIR] : 0.5539

                  Kappa : 0

 Mcnemar's Test P-Value : 1.587e-06

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.6795
         Neg Pred Value :    NaN
             Prevalence : 0.6795
         Detection Rate : 0.6795
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : 0

> |
```

Comment:

Algorithm	Accuracy (%)
Logistic Regression	71.89
Decision Tree	74.43
SVM	67.95
Random Forest	67.54

11. Select the most accurate model

Comment:

Based on table above, the best model is Decision Tree.

12. Identify other Machine learning or statistical techniques

Comment:

Other statistical techniques are Naïve Bayes, K-Nearest Neighbor, Artificial Neural Network, Stochastis Gradient Descent.

Descriptive:

Categorize the average of grade point into High, Medium, and Low (with admission probability percentages) and plot it on a point chart.
Cross grid for admission variables with GRE Categorization is shown below:

GRE	Categorized
0-440	Low
440-580	Medium
580+	High

Code:

```
#Descriptive
#Categorize the average of grade point into High, Medium, and Low
#(with admission probability percentages) and plot it on a point chart.

?cut
max(college_admission1$gre)
cut(college_admission$gre, breaks = c(0, 440, 580, Inf),
    labels = c("Low", "Medium", "High"))
college_admission_bin <- college_admission
college_admission_bin$grebin <- cut(college_admission$gre, breaks = c(0, 440,
    580, Inf), labels = c("Low", "Medium", "High"))
View(college_admission_bin)

collab.df <- college_admission_bin[,2:3]
kmeans <- kmeans(collab.df, 3)
plot(collab.df[c("gre", "gpa")], col = kmeans$cluster)
points(kmeans$centers[,c("gre", "gpa")], col = 1:3, pch = 8, cex = 2)
```

```
> max(college_admission1$gre)
[1] 800
> cut(college_admission$gre, breaks = c(0, 440, 580, Inf),
+     labels = c("Low", "Medium", "High"))
  [1] Low    High   High   High   Medium High   Medium Low    Medium High   High   Low    High   High   High   Medium High   Low    High  
 [20] Medium Medium High   High   High   High   High   High   Medium High   Medium Medium High   High   High   Low    Low    Medium Medium
 [39] Medium Medium Medium Medium High   Medium High   Medium Medium Medium Low    Low    High   Low    High   High   High   High   Medium
 [58] Low    Low    High   High   Medium High   High   Medium High   High   High   Medium High   High   Low    Medium Medium High   High  
 [77] Medium High   Medium High   High   High   Medium Low    Medium Medium High   High   High   High   High   High   High   Medium High  
 [96] High   High   Medium High   Low    Low    Medium Low    Medium High   High   High   Medium Low    Medium High   Low    Low    High  
[115] High   High   Low    High   High   Low    Medium Medium Medium Medium High   Medium High   High   Medium Medium High   High   Medium
[134] Medium Medium Medium Medium High   High   High   High   High   High   Medium Medium Low    Medium Medium Medium High   High   Low   
[153] High   Medium High   Medium Medium Medium High   High   High   High   High   Medium Medium High   Low    High   Medium High   Low   
[172] Medium High   High   Medium High   Medium High   High   Low    High   Medium High   Medium Medium High   Medium Medium Medium Medium
[191] High   High   High   Low    High   Medium High   Low    High   Medium High   Medium High   Low    High   High   High   High   Medium
[210] Medium High   Medium Medium High   High   High   Low    Medium Medium Medium Medium High   Medium High   High   High   High   Medium
[229] Medium High   Medium High   Low    Low    High   High   High   Medium Medium High   Low    Medium High   High   Medium High   High  
[248] Low    High   High   High   High   Medium Medium High   High   Medium High   Medium High   High   Low    Medium High   Medium Low   
[267] Medium High   High   Medium High   Medium High   High   Medium High   Medium Medium High   High   High   Low    High   Medium Low   
[286] High   High   High   High   Low    High   High   High   High   Medium Medium Medium Medium Medium High   High   High   Low    High  
[305] Low    Medium Medium Medium Medium Low    Medium High   High   Medium Medium Low    Low    High   Medium Medium Medium Medium Medium
[324] Low    Medium High   High   Medium Medium Medium High   High   Low    Medium Medium High   Medium High   Medium High   Medium Medium
[343] Medium Medium Medium Medium High   Medium Low    High   High   High   Medium High   Medium High   High   High   Medium High   Medium
[362] Medium High   Medium Medium Medium Medium High   Medium High   Medium High   High   High   Medium Medium High   High   High   Medium
[381] High   Medium Medium High   Medium Low    High   Medium High   High   High   High   High   High   Medium High   Medium Medium High  
[400] High  
Levels: Low Medium High
> college_admission_bin <- college_admission
> college_admission_bin$grebin <- cut(college_admission$gre, breaks = c(0, 440, 580, Inf),
+                                     labels = c("Low", "Medium", "High"))
> View(college_admission_bin)
> collab.df <- college_admission_bin[,2:3]
> kmeans <- kmeans(collab.df, 3)
> plot(collab.df[c("gre", "gpa")], col = kmeans$cluster)
> points(kmeans$centers[,c("gre", "gpa")], col = 1:3, pch = 8, cex = 2)
> 
```

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function   Addins

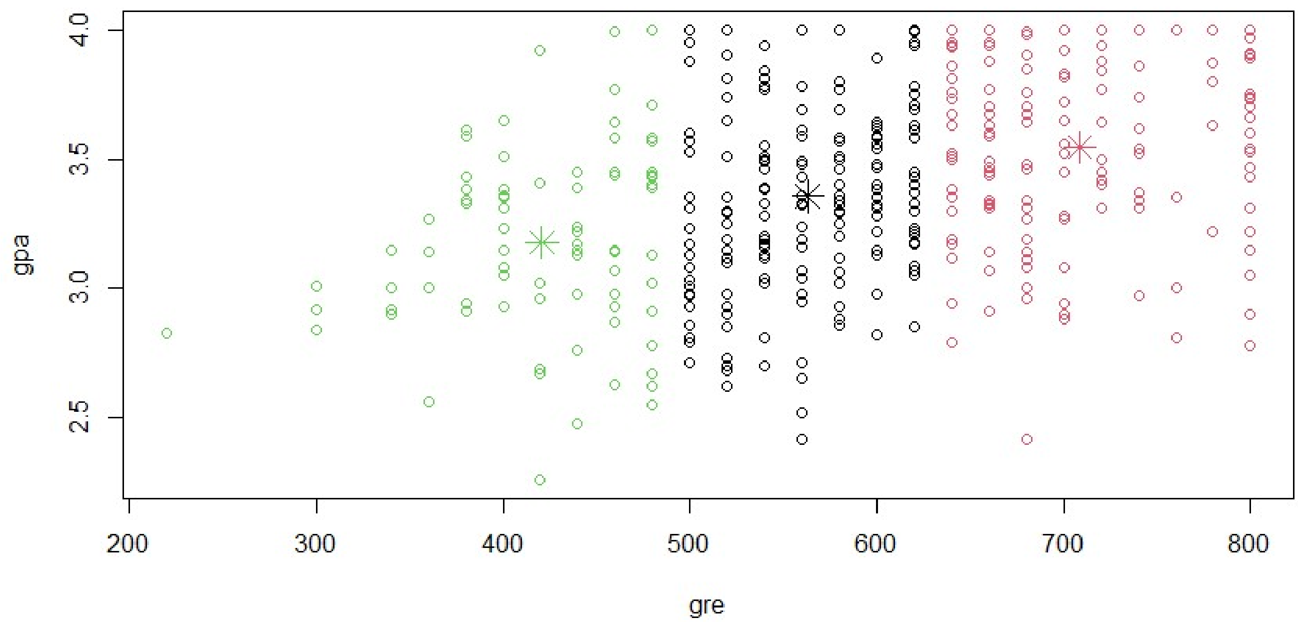
Ecommerse Project.R × | College Admission.R* × | college_admission_bin × | college_admission1 × | college_admission ×

Filter

	admit	gre	gpa	ses	Gender_Male	Race	rank	grebin
1	0	380	3.61	1	0	3	3	Low
2	1	660	3.67	2	0	2	3	High
3	1	800	4.00	2	0	2	1	High
4	1	640	3.19	1	1	2	4	High
5	0	520	2.93	3	1	2	4	Medium
6	1	760	3.00	2	1	1	2	High
7	1	560	2.98	2	1	2	1	Medium
8	0	400	3.08	2	0	2	2	Low
9	1	540	3.39	1	1	1	3	Medium
10	0	700	3.92	1	0	2	2	High
11	0	800	4.00	1	1	1	4	High
12	0	440	3.22	3	0	2	1	Low
13	1	760	4.00	3	1	2	1	High
14	0	700	3.08	2	0	2	2	High
15	1	700	4.00	2	1	1	1	High
16	0	480	3.44	3	0	1	3	Medium
17	0	780	3.87	2	0	3	4	High
18	0	360	2.56	3	1	3	3	Low
19	0	800	3.75	1	1	3	2	High
20	1	540	3.81	1	0	3	1	Medium
21	0	500	3.17	3	0	2	3	Medium
22	1	660	3.63	1	0	1	2	High
23	0	600	2.82	1	0	3	4	High
24	0	680	3.19	1	0	1	4	High
25	1	760	3.35	2	0	2	2	High
26	1	800	3.66	2	1	1	1	High
27	1	620	3.61	2	0	1	1	High
28	1	520	3.74	2	0	3	4	Medium
29	1	780	3.22	1	0	1	2	High
30	0	520	3.29	1	0	1	1	Medium
31	0	540	3.78	1	1	1	4	Medium
32	0	760	3.35	2	1	1	3	High
33	0	600	3.40	3	0	1	3	High
34	1	800	4.00	3	0	1	3	High
35	0	360	3.14	1	1	2	1	Low
36	0	400	3.05	3	0	2	2	Low

Showing 1 to 38 of 400 entries, 8 total columns

Point chart (GPA vs GRE)



GRE	Categorized	Color
0-440	Low	Green
440-580	Medium	Black
580+	High	Red