```
# =============== Loading Libraries ============
library(readxl)   # for reading excel files
library(caTools) # for splitting dataset
library(MLmetrics) # for machine learning metrics
library(caret)  # for feature importance
library(lattice)
library(tidyverse)
library(ggpubr) # for creating and customizing 'ggplot2'- based publication ready plots
library(factoextra)
library(rpart)
library(caTools)
library(randomForest)
library(kernlab)
library(readr)
library(rpart.plot)
library(naivebayes)
library(stats)

# ============= Removing  Existing R-Objects in Environment ===========
rm(list = ls())

# ===================== Loading Dataset ===============
df <- read.csv("College_admission.csv", header = T)
View(df)



# =================== EDA ========================
dim(df)
str(df)
summary(df)
```

```
# ========== Question-1 ======================================
# checking and handling NA
sapply(df,function(x) sum(is.na(x))) #no NAs

# ========== Question-2 ======================================
# checking for outliers
boxplot(df)
boxplot.stats(df$gre)$out
boxplot.stats(df$gpa)$out
df <- df[(df$gre >300 & df$gpa != 2.26),]
boxplot.stats(df$gre)$out
boxplot.stats(df$gpa)$out
boxplot(df)

# creating grade column
df$grade <- ifelse(df$gre<=440,'Low', ifelse(df$gre>440&df$gre<=580,'Medium',
ifelse(df$gre>580,'High','Other')))
table(df$grade)
View(df)
#plotting barplot for grade and admit side-by-side
barplot(table(df$admit, df$grade),
        beside = T,
        axisnames = T,
        xlab = 'Grade-Admit(0/1)',
        ylab = 'Frequency',
        main = 'Grade-Admit Chart'
)
#Line Chart for Grade
plot(table(df$grade),type = "o",col = "red", xlab = "Grade", ylab = "Frequency",
     main = "Grade chart")
```

```
# ========== Question-3 ====================================
# structure of data-frame and converting required numeric column to factor and vice-verse
str(df)
df <- df %>% mutate_at(c(1,5,6), funs(factor(.)))
df$ses <- factor(df$ses, ordered = T, levels = c(1:3))
df$grade <- factor(df$grade, ordered = T, levels = c('Low','Medium','High'))
df$rank <- factor(df$rank, ordered = T, levels = c(4:1))
str(df)
View(df)

# ========== Question-4 ====================================
# checking normality
ggdensity(df$gre,
        xlab = 'GRE-Score',
        ylab = 'Density',
        main = 'GRE Score Chart')
ggdensity(df$gpa,
        xlab = 'GPA-Score',
        ylab = 'Density',
        main = 'GPA Score Chart')
sapply(df[2:3], function(x) shapiro.test(x))

# ========== Question-5 ====================================
# Standardization Data
df[2:3] <- sapply(df[2:3], function(x) scale(x, center = T, scale = T))
summary(df)
View(df)
mydf <- df #dataframe for decision-tree
```

```
# ========== Question-6 ====================================
# creating dummy variables for factor attributes
dummies<- data.frame(sapply(df[,c(1,4:7)],
                      function(x) data.frame(model.matrix(~x,data =df[,c(1,4:7)]))[,-1]))




View(dummies)
df <- cbind(df[,c(2:3)], dummies)

# data splitting
sample <- sample.split(df, SplitRatio = 0.7)
train <- df[sample,]
View(train)
test <- df[!sample,]

# Feature Importance
pca <- prcomp(train[,-3])
summary(pca)
fviz_eig(pca)
fviz_pca_var(pca,
         col.var = "contrib", # Color by contributions to the PC
         gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
         repel = TRUE     # Avoid text overlapping
)

# ========== Question-7 & 8 ====================================

# logistic model
set.seed(123) #for randomness
```

```
#building model
lmodel <- glm(formula = admit~.,
              data = train,
              family = 'binomial')
summary(lmodel)

#testing model on test data
pred <- predict(lmodel,
                type = 'response',
                newdata = test[,-3])
test$actual <- factor(ifelse(test$admit == 1,'Yes','No'))
test$pred <- factor(ifelse(pred >= 0.5, 'Yes','No')) #taking cutoff as 0.5
View(test)

#confusion-matrix
confmatrix <- confmatirx <- confusionMatrix(test$pred,
                              test$actual,
                              positive = 'Yes')
confmatirx

#function to get optimal cut-off
perform_fn <- function(cutoff)
{
  pred <- factor(ifelse(pred >= cutoff, "Yes", "No"))
  conf <- confusionMatrix(pred,test$actual, positive = "Yes")
  acc <- conf$overall[1]
  sens <- conf$byClass[1]
  spec <- conf$byClass[2]
  out <- t(as.matrix(c(sens, spec, cutoff,acc)))
```

```
  colnames(out) <- c("sensitivity", "specificity", 'cutoff', 'accuracy')
  return(out)
}



s = seq(.01,.80,length=100)
s

OUT = matrix(0,100,4)

OUT

for(i in 1:100)
{
  OUT[i,] = perform_fn(s[i])
}
OUT



# Let's choose a cutoff value of 0.52070707  for final model
test_cutoff <- factor(ifelse(pred >=0.52070707 , "Yes", "No"))
conf_final <- confusionMatrix(test_cutoff, test$actual, positive = "Yes")

pval <- conf_final$overall[6]
acc <- conf_final$overall[1]
sens <- conf_final$byClass[1]
spec <- conf_final$byClass[2]

pval
acc
```

```
sens
spec
# ========== Question- 9, 10, 11, 12 ====================================
#random forest model

#train-control for random forest
control <- trainControl(method="repeatedcv",
                        number=10,
                        repeats=3,
                        savePredictions=TRUE,
                        classProbs=TRUE,
                        summaryFunction = twoClassSummary)
#building model
rfmodel <- rpart(admit~.,
                 data = train,
                 method = 'class')

# make predictions on the test set
predrf <- predict(rfmodel,
                  test,
                  type = 'class')

test$predrf <- factor(ifelse(predrf == 1,'Yes','No'))
#confusion-matrix
rfconmatrix <- confusionMatrix(test$predrf,test$actual,positive = 'Yes')
rfconmatrix

# ===================================================================
#decision tree model
```

```
dttrain <- mydf[sample,]
dttest <- mydf[!sample,]

#building model
dtmodel <- rpart(admit ~ .,
          data = dttrain,
          method = "class",
          control = rpart.control(minsplit = 500,
                    minbucket = 250,
                    cp = 0.05))

# make predictions on the test set
tree.predict <- predict(dtmodel, dttest, type = "class")

#confusion-matrix
confusionMatrix(tree.predict, as.factor(dttest$admit), positive = '1')

# ==============================================================
#naive bayes

#building model
nbmodel <- naive_bayes(admit ~ ., data = dttrain, usekernel = T)

# make predictions on the test set
nbpred <- predict(nbmodel, dttest)

#confusion-matrix
test_conf2 <- confusionMatrix(nbpred, factor(dttest$admit),positive = '1')
test_conf2
# ==============================================================
```