```
#Install packages if already not installed; if installed previously skip this
step

#install.packages('haven')
#install.packages('dplyr')
#install.packages("My.stepwise")
#install.packages('rpart')
#install.packages('e1071')
#install.packages('randomForest')
#install.packages('caret')
#install.packages('gbm')

#To understand various model on R using caret package, please refer to the link
below:
#https://topepo.github.io/caret/available-models.html

#Call libraries you will use for the code

library(haven)
library(dplyr)
library(My.stepwise)
library(rpart)
library(e1071)
library(randomForest)
library(caret)
library(gbm)

#Set working directory to where the data is present

getwd()
setwd('C://OSU 2019-2021//Semester - 3//BAN 5753//Week 6//R & Python Code//')

#Read the sas file using haven library; alternately you can also use sasdata
package

df = read_sas('pmad_pva.sas7bdat')

#Perform Summary Statistics on the data to understand the data

head(df,5) #Print top 5 rows
names(df) #Print names of all columns
dim(df) #Print the dimension of the dataset
str(df) #Print data summary (Part I)
summary(df) #Print data summary (Part II)
sum(is.na(df)) #Check number of null values in the dataset

#Convert data into factors (categorical data); variable selected based on
inspection on the EDA performed previously

df$TargetB <- factor(df$TargetB)
df$StatusCat96NK <- factor(df$StatusCat96NK)
df$DemCluster <- factor(df$DemCluster)
df$DemGender <- factor(df$DemGender)
df$DemHomeOwner <- factor(df$DemHomeOwner)

#Confirm the changes made above
```

```
#Impute Null Values for Continous variables
# Return the column names containing missing observations

list_na <- colnames(df)[ apply(df, 2, anyNA) ]

print(list_na)

#Impute Missing data with the Mean
#we stored the columns name with the missing values in the list called list_na.
#We will use this list we need to compute of the mean with the argument
#na.rm = TRUE. This argument is compulsory because the columns have missing data,
#and this tells R to ignore them.

# Create mean

average_missing <- apply(df[,colnames(df) %in% list_na],
                         2,
                         mean,
                         na.rm =  TRUE)
print(average_missing)

#Replace the NA Values
#Create flag for records with missing value

df_imp <- df %>%
  mutate(replace_mean_GiftAvgCard36  = ifelse(is.na(GiftAvgCard36),
average_missing[1], GiftAvgCard36),
         replace_mean_DemAge  = ifelse(is.na(DemAge), average_missing[2],
DemAge),
         replace_mean_DemMedIncome  = ifelse(is.na(DemMedIncome),
average_missing[3], DemMedIncome),
         flag_null_GiftAvgCard36  = factor(ifelse(is.na(GiftAvgCard36), 1, 0)),
         flag_null_DemAge  = factor(ifelse(is.na(DemAge), 1, 0)),
         flag_null_DemMedIncome  = factor(ifelse(is.na(DemMedIncome), 1, 0)))

df_imp = subset(df_imp,select = -c(GiftAvgCard36,DemAge,DemMedIncome))

#Confirm the changes after imputation

list_na_confirm <- colnames(df_imp)[ apply(df_imp, 2, anyNA) ]
print(list_na_confirm)

#To check for additional methods to replace null values please refer to the link
below
#https://www.guru99.com/r-replace-missing-values.html

#The following code splits 70% of the data selected randomly into training set
and the remaining 30% sample into test data set.

## 70% of the sample size

smp_size <- floor(0.70 * nrow(df_imp))

## set the seed to make your partition reproducible

set.seed(123)
```

```
Logistic_accuracy_rate <- function(model, data){
  target = c(data$TargetB)
  glm.probs <- predict(model,newdata = data, type="response")
  pred_target <- ifelse(glm.probs>=0.5,1,0)
  df_class <- cbind(target,pred_target)
  class_tbl <- xtabs(~target + pred_target, data=df_class)
  class_pct <- class_tbl/length(target)
  classification_rate <- (class_pct[1,1]+class_pct[2,2])*100
  print(classification_rate) #Accuracy of the model
}

#Model 1 - Logistic Regression (No Variable Selection)

#Perform Logistic Regression

Logistic_model1 <- glm(TargetB~.,family=binomial(link='logit'),data = train)
summary(Logistic_model1)

#Accuracy For Logistic Regression

Logistic_accuracy_rate(Logistic_model1,train)
Logistic_accuracy_rate(Logistic_model1,test)


#Model 2 - Logistic Regression (Stewise Variable Selection)

#Variable Selection using Stepwise (P Value)

variable_list <- c("GiftCntCard36","GiftAvg36","GiftTimeFirst",
                   "PromCntAll","PromCntCardAll","DemMedHomeValue",
                   "replace_mean_DemAge","flag_null_DemAge","GiftCnt36",
                   "GiftCntCardAll","GiftAvgAll","PromCnt12","PromCntCard12",
                   "StatusCat96NK","DemGender","DemPctVeterans",
                   "replace_mean_DemMedIncome","flag_null_DemMedIncome",
                   "GiftCntAll","GiftAvgLast","GiftTimeLast","PromCnt36",
                   "PromCntCard36","StatusCatStarAll","DemHomeOwner",
                   "replace_mean_GiftAvgCard36")

Logistic_model_Variable_Selection = My.stepwise.glm(Y = "TargetB",
                                                    variable.list =
variable_list,
                                                    in.variable = "NULL",
                                                    data = train,
                                                    sle = 0.05,
                                                    sls = 0.05,
                                                    myfamily =
binomial(link='logit'))

#Run a second logistic Regression based on the variables selected in the
#last iterations in the output

train_2 = subset(train,select = c(TargetB,GiftCnt36,
                                  GiftTimeLast,
                                  DemMedHomeValue,
                                  GiftCntCardAll,
```

```
                         family=binomial(link='logit'),
                         data = train_2)

#Accuracy For Logistic Regression

Logistic_accuracy_rate(Logistic_model2,train)
Logistic_accuracy_rate(Logistic_model2,test)

#Create a function to return accuracy rate for the test data

decision_accuracy_rate <- function(model, data){
  target = c(data$TargetB)
  glm.probs <- predict(model,data, type="class")
  #pred_target <- ifelse(glm.probs>=0.5,1,0)
  pred_target <- glm.probs
  df_class <- cbind(target,pred_target)
  class_tbl <- xtabs(~target + pred_target, data=df_class)
  class_pct <- class_tbl/length(target)
  classification_rate <- (class_pct[1,1]+class_pct[2,2])*100
  print(classification_rate) #Accuracy of the model
}

#Model 3 - Decision Tree
# grow tree

Decision_Tree_Model <- rpart(TargetB ~ .,
                             data = train,
                             method = 'class',
                             parms = list(split = "information"))

decision_accuracy_rate(Decision_Tree_Model,train)
decision_accuracy_rate(Decision_Tree_Model,test)

printcp(Decision_Tree_Model) # display the results
plotcp(Decision_Tree_Model) # visualize cross-validation results
summary(Decision_Tree_Model) # detailed summary of splits

# plot tree

plot(Decision_Tree_Model,
     uniform=TRUE,
     main="Classification Tree for TargetB")

text(Decision_Tree_Model,
     use.n=TRUE,
     all=TRUE,
     cex=.8)

# create attractive postscript plot of tree

post(Decision_Tree_Model,
     file = "Decision_Tree_Model.ps",
     title = "Classification Tree for TargetB")

#Prune back the tree to avoid overfitting the data.
#Typically, you will want to select a tree size that minimizes the cross-
```

```
Prune_Decision_Tree_Model <- prune(Decision_Tree_Model,
                                   cp=
Decision_Tree_Model$cptable[which.min(Decision_Tree_Model$cptable[,"xerror"]),"CP

decision_accuracy_rate(Prune_Decision_Tree_Model,train)
decision_accuracy_rate(Prune_Decision_Tree_Model,test)

# plot the pruned tree

plot(Prune_Decision_Tree_Model,
     uniform=TRUE,
     main="Pruned Classification Tree for TargetB")

text(Prune_Decision_Tree_Model,
     use.n=TRUE,
     all=TRUE,
     cex=.8)

post(Prune_Decision_Tree_Model,
     file = "Prune_Decision_Tree_Model.ps",
     title = "Pruned Classification Tree for TargetB")

#To understand the syntax better, refer to the link below
#https://www.statmethods.net/advstats/cart.html

#Model-4 (SVM)

#Create a function to return accuracy rate for the data

svm_accuracy_rate <- function(model, data){
  target = c(data$TargetB)
  x <- subset(data, select=-c(TargetB))
  glm.probs <- predict(model,x)
  #pred_target <- ifelse(glm.probs>=0.5,1,0)
  pred_target <- glm.probs
  df_class <- cbind(target,pred_target)
  class_tbl <- xtabs(~target + pred_target, data=df_class)
  class_pct <- class_tbl/length(target)
  classification_rate <- (class_pct[1,1]+class_pct[2,2])*100
  print(classification_rate) #Accuracy of the model
}


# Use data selected in stepwsise regression

train_2 = subset(train,select = c(TargetB,
                                  GiftCnt36,
                                  GiftTimeLast,
                                  DemMedHomeValue,
                                  GiftCntCardAll,
                                  GiftAvg36,
                                  flag_null_GiftAvgCard36,
                                  StatusCatStarAll,
                                  PromCnt12))

test_2 = subset(test,select = c(TargetB,
```

```
                                    flag_null_GiftAvgCard36,
                                    StatusCatStarAll,
                                    PromCnt12))


#SVM MODEL 1 using Linear Kernel

svm_model1 <- svm(TargetB ~ . ,
                  data=train_2,
                  kernel="linear")

print(svm_model1)
summary(svm_model1)
svm_accuracy_rate(svm_model1, train_2)
svm_accuracy_rate(svm_model1, test_2)

#SVM MODEL 2 using Polynomial Kernel

svm_model2 <- svm(TargetB ~ . ,
                  data=train_2,
                  kernel="polynomial")

print(svm_model2)
summary(svm_model2)
svm_accuracy_rate(svm_model2, train_2)
svm_accuracy_rate(svm_model2, test_2)

#SVM MODEL 3 using radial basis Kernel

svm_model3 <- svm(TargetB ~ . ,
                  data=train_2,
                  kernel="radial")

print(svm_model3)
summary(svm_model3)
svm_accuracy_rate(svm_model3, train_2)
svm_accuracy_rate(svm_model3, test_2)

#SVM MODEL 4 using sigmoid basis Kernel

svm_model4 <- svm(TargetB ~ . ,
                  data=train_2,
                  kernel="sigmoid")

print(svm_model4)
summary(svm_model4)
svm_accuracy_rate(svm_model4, train_2)
svm_accuracy_rate(svm_model4, test_2)

#Model-5 (Random Forest)

# Define the control

trControl <- trainControl(method = "cv",
                          number = 10,
                          search = "grid")
```

```
Random_Forest_Model <- train(TargetB ~ .,
                             data = train,
                             method = "rf",
                             metric = "Accuracy",
                             trControl = trControl)

# Print the results
print(Random_Forest_Model)

prediction <- predict(Random_Forest_Model, test)
confusionMatrix(prediction,test$TargetB)

#Model - 6 (Gradient Boosting)

# Define the control

trControl <- trainControl(method = "cv",
                          number = 10,
                          search = "grid")

set.seed(1234)

# Run the model

Gradient_Boosting_Model <- train(TargetB ~ .,
                             data = train,
                             method = "gbm",
                             metric = "Accuracy",
                             trControl = trControl)

# Print the results

print(Gradient_Boosting_Model)

prediction <- predict(Gradient_Boosting_Model, test)
confusionMatrix(prediction,test$TargetB)
```