```
#-----------------------------Initial Setup ------------------------

# Set random number seed for reproducibility
set.seed(7406)

# Set the working directory
setwd("C:/Users/ns14555/Desktop/Projects/003 Project 3 (HW5)/003 Project
work")
# Check the current working directory to confirm it was set correctly
getwd()

# Load required libraries
library(car)
library(caret)
library(glmnet)
library(ggplot2)
library(gridExtra)
library(MASS)
library(e1071)
library(nnet)
library(class)
library(randomForest)
library(xgboost)
library(gbm)

#------------------Data Loading and Preparation----------------

# Load Data
d_data <-read.table("C:/Users/ns14555/Desktop/Projects/003 Project 3
(HW5)/003 Project work/diabetes_binary_health_indicators_BRFSS2015.csv",
head=T, sep=",")
# Get one-fourth of the data
split_index <- createDataPartition(d_data$Diabetes_binary, p = .25,
                                   list =FALSE)
# Create the fourth dataset
one_fourth_data <- d_data[split_index, ]
d_data <- d_data[split_index, ]

# Exploratory Data Analysis
# Dimension
dim(d_data)
# 253680 rows 22 columns
```

```
head(d_data)
summary(d_data)
#Check for missing value
colSums(is.na(d_data))
#Datatype of the fields
str(d_data)
#Changing data types
columns_to_convert <- c("Diabetes_binary","HighBP","CholCheck","HighChol",
                        "Smoker","Stroke","HeartDiseaseorAttack",
                      "PhysActivity","Fruits","Veggies","HvyAlcoholConsump"
                       ,"AnyHealthcare","NoDocbcCost","GenHlth","DiffWalk",
                        "Sex","Age","Education", "Income")
d_data[columns_to_convert] <-lapply(d_data[columns_to_convert], as.factor)
#Datatype checking again
str(d_data)
#We need to reduce the number of variables.
model_step <- stepAIC(glm(Diabetes_binary ~.,data = d_data,family
=binomial),
                      direction = "both")
summary(model_step)
columns_to_remove
<-c("PhysActivity","PhysHlth","Fruits","Veggies","Education",
                      "MentHlth","AnyHealthcare","NoDocbcCost")
d_data <- subset(d_data, select = !names(d_data) %in% columns_to_remove)
dim(d_data)
head(d_data)
#Histogram for BMI
ggplot(d_data, aes(x=BMI)) +geom_histogram(
  bandwidth = 1, fill = 'skyblue', color='black') +labs(
    title='Histogram of BMI', x="BMI", y = "Frequency")
#Bar plots for categorical variables
categorical_vars <- c("HighBP","CholCheck","HighChol", "Smoker",
                    "Stroke","HeartDiseaseorAttack","HvyAlcoholConsump"
                      ,"DiffWalk","Sex")
plot_list <-list()
for (var in categorical_vars) {
  plot<-ggplot(d_data, aes_string(x=factor(d_data$Diabetes_binary),
                                         fill =var))
+geom_bar(position="fill") +
    labs(title=paste('Bar plot of',var), x = "Diabetes
Binary",y="Proportion") +
    scale_fill_manual(values = c("skyblue", "lightcoral"))
  plot_list[[var]] <- plot
}
grid.arrange(grobs=plot_list, ncol=3)
#number of patients with and without diabetes
table(d_data$Diabetes_binary)
#   0    1
# 54627 8793
#column names
names(d_data)
#Defining response and predictor variables
response_variable <- 'Diabetes_binary'
predictor_variables <- c("HighBP", "HighChol", "CholCheck","BMI","Smoker",
                        "Stroke","HeartDiseaseorAttack","HvyAlcoholConsump"
                         ,"AnyHealthcare","GenHlth","MentHlth","DiffWalk",
                         "Sex","Age")

n = dim(d_data)[1] # total number of observations
n1 = round(n/10) # number of observations randomly selected for testing
data
# Split the data into training and test set
train <- sample(c(TRUE, FALSE), nrow(d_data),replace=TRUE,
                prob=c(0.8, 0.2))
d_train <- d_data[train, ]
d_test <- d_data[!train,]

dim(d_train) ## Distribution of the data labels in the training data
#50650   14
dim(d_test) ## Distribution of the data labels in the testing data
#12770   14

#---------------------------Model Building--------------------------

# Initialize testing error collection
train_MSE <- NULL
test_MSE <- NULL

# Model 1:Linear Discriminant Analysis (LDA)
model_lda <- lda(d_train[,1] ~., data = d_train[,2:14])
pred_lda_train <- predict(model_lda, d_train[,-1])$class
train_MSE <- round(c(train_MSE,
mean(pred_lda_train!=d_train$Diabetes_binary)),8)
pred_lda_test <- predict(model_lda, d_test[,-1])$class
test_MSE <- round(c(test_MSE,
mean((pred_lda_test!=d_test$Diabetes_binary))),8)
```

```r
# Model 2:Quadratic Discriminant Analysis (QDA)
model_qda <- qda(d_train[,1] ~., data = d_train[,2:14])
pred_qda_train <- predict(model_qda, d_train[,-1])$class
train_MSE <- round(c(train_MSE,
mean(pred_qda_train!=d_train$Diabetes_binary)),8)
pred_qda_test <- predict(model_qda, d_test[,-1])$class
test_MSE <- round(c(test_MSE,
mean((pred_qda_test!=d_test$Diabetes_binary))),8)

# Model 3: Naive Bayes
model_naiveBayes <- naiveBayes(d_train[,1] ~., data = d_train[,2:14])
pred_naiveBayes_train <- predict(model_naiveBayes, d_train[,-1])
train_MSE <- round(c(train_MSE,
mean(pred_naiveBayes_train!=d_train$Diabetes_binary)),8)
pred_naiveBayes_test <- predict(model_naiveBayes, d_test[,-1])
test_MSE <- round(c(test_MSE,
mean((pred_naiveBayes_test!=d_test$Diabetes_binary))),8)

# Model 4: Multinomial logisitic regression
model_lr <- multinom(d_train[,1] ~., data = d_train[,2:14])
pred_lr_train <- predict(model_lr, d_train[,-1])
train_MSE <- round(c(train_MSE,
mean(pred_lr_train!=d_train$Diabetes_binary)),8)
pred_lr_test <- predict(model_lr, d_test[,-1])
test_MSE <- round(c(test_MSE,
mean((pred_lr_test!=d_test$Diabetes_binary))),8)

# Model 5: KNN with several values
k_list <- c(2,3,4,5,6)
xnew <- d_train[,-1];
xnew2 <- d_test[,-1];
train_errors <-NULL
test_errors <-NULL
for (i in 1: 5){
  kk <- k_list[i];
  pred4 <- knn(d_train[,-1], xnew, d_train[,1], k=kk);
  train_errors <- rbind( train_errors, cbind(kk,
                                             mean( pred4 != d_train[,1])));
  pred4.test <- knn(d_train[,-1], xnew2, d_train[,1], k=kk);
  test_errors <- rbind( test_errors, cbind(kk,
                                           mean( pred4.test!=
                                                   d_test[,1])));
```

```r
}
results <- data.frame(
  K = train_errors[, 1],  # K-values
  Train_Error = train_errors[, 2],  # Training errors
  Test_Error = test_errors[, 2]  # Testing errors
)
results
#k=5 is the best k value
model_knn <- knn(d_train[,-1], d_train[,-1], d_train[,1], k=5)
train_MSE <- round(c(train_MSE,
mean(model_knn!=d_train$Diabetes_binary)),8)
pred_knn_test <- knn(d_train[,-1], d_test[,-1], d_train[,1], k=5)
test_MSE <- round(c(test_MSE,
mean((pred_knn_test!=d_test$Diabetes_binary))),8)

# Model 6: Random Forest
model_rf <- randomForest(d_train[,1] ~., data = d_train[,2:14], mtry=4,
ntree=500)
pred_rf_train <- predict(model_rf, d_train[,-1])
train_MSE <- round(c(train_MSE, mean(pred_rf_train !=
d_train$Diabetes_binary)),8)
pred_rf_test <- predict(model_rf, d_test[,-1])
test_MSE <- round(c(test_MSE, mean(pred_rf_test !=
d_test$Diabetes_binary)),8)

# Model 7: Boosting
# Identifying columns that need one-hot encoding
# Ensure labels are binary (0 or 1)
d_train$Diabetes_binary <- as.numeric(d_train$Diabetes_binary == 1)
d_test$Diabetes_binary <- as.numeric(d_test$Diabetes_binary == 1)
vars_to_encode <- c("HighBP", "HighChol", "CholCheck", "Smoker", "Stroke",
                    "HeartDiseaseorAttack", "HvyAlcoholConsump", "GenHlth",
                    "DiffWalk", "Sex", "Age", "Income")

# Performing one-hot encoding
dummy_model_train <- dummyVars(~ ., data = d_train[, vars_to_encode])
encoded_train_data <- predict(dummy_model_train, newdata = d_train)
dummy_model_test <- dummyVars(~ ., data = d_test[, vars_to_encode])
encoded_test_data <- predict(dummy_model_test, newdata = d_test)

# Rebuilding the dataset excluding the original columns that were encoded
columns_to_remove <- vars_to_encode
full_train_data <- cbind(d_train[, !(names(d_train) %in%
columns_to_remove)], encoded_train_data)
full_test_data <- cbind(d_test[, !(names(d_test) %in% columns_to_remove)],
encoded_test_data)
# Add this conversion just before creating the DMatrix
full_train_data$Diabetes_binary <-
as.numeric(full_train_data$Diabetes_binary == 1)
full_test_data$Diabetes_binary <- as.numeric(full_test_data$Diabetes_binary
== 1)
# Defining parameter grid for XGBoost
param_grid_xgb <- list(
  list(eta = 0.01, max_depth = 3, gamma = 0),
  list(eta = 0.1, max_depth = 6, gamma = 0.1),
  list(eta = 0.3, max_depth = 9, gamma = 0.3)
)

# Function to train XGBoost model using different parameters
cross_validate_xgb <- function(params, full_train_data) {
  dtrain <- xgb.DMatrix(data = as.matrix(full_train_data[, -1]), label =
full_train_data$Diabetes_binary)
  model <- xgboost(
    params = params,
    data = dtrain,
    nrounds = 100,
    objective = "binary:logistic",
    eval_metric = "error"
  )
  return(model)
}

# Cross-validation across parameter grid
models_xgb <- lapply(param_grid_xgb, function(params)
cross_validate_xgb(params, full_train_data))

# Evaluate each model and find the best one
errors_cv_xgb <- sapply(models_xgb, function(mod) {
  pred <- predict(mod, as.matrix(full_train_data[, -1]))
  mean(pred != full_train_data$Diabetes_binary)
})

best_model_index <- which.min(errors_cv_xgb)
best_parameters <- param_grid_xgb[[best_model_index]]
best_model <- models_xgb[[best_model_index]]
```

```r
# Retraining the best model with selected parameters
mod_final <- xgboost(
  data = as.matrix(full_train_data[, -1]),
  label = full_train_data$Diabetes_binary,
  nrounds = 100,
  params = best_parameters
)

# Prediction and error calculation
pred_train_final <- predict(mod_final, as.matrix(full_train_data[, -1]))
binary_pred_train_final <- ifelse(pred_train_final >= 0.5, 1, 0)
train_err_final <- round(mean(binary_pred_train_final !=
full_train_data$Diabetes_binary), 8)

pred_test_final <- predict(mod_final, as.matrix(full_test_data[, -1]))
binary_pred_test_final <- ifelse(pred_test_final >= 0.5, 1, 0)
test_err_final <- round(mean(binary_pred_test_final !=
full_test_data$Diabetes_binary), 8)
train_MSE <- round(c(train_MSE, train_err_final),8)
test_MSE <- round(c(test_MSE, test_err_final),8)
mod_final$params

#Tables for models and Errors
# Table for k values with training and testing errors
models <-c('LDA','QDA','Naive Bayes','Logistic Regression','KNN','Random
Forest','Boosting')
model_results <- data.frame(
  Models = models,
  Train_Error = train_MSE,
  Test_Error = test_MSE
)
model_results

#---------------------------Monte Carlo
Cross-Validation----------------------

set.seed(7407) # Reset seed fr reproducibility
B <- 5
TEALL <- matrix(nrow = B, ncol =5) # Preallocate matrix for efficiency

for (b in 1:B){
  #-------------Initial Preparation-----------
  indices <- sample(1:nrow(d_data), size = round(0.2*nrow(d_data)))
  train_data <- d_data[-indices,]
  test_data <- d_data[indices,]

  #------------- Model Building------------------
  # Model 1:Linear Discriminant Analysis (LDA)
  model_1 <- lda(train_data[,1] ~., data = train_data[,2:14])
  pred_1 <- predict(model_1, test_data[,-1])$class
  te1 <- round(mean((pred_1!=test_data$Diabetes_binary)),8)

  # Model 2:Quadratic Discriminant Analysis (QDA)
  model_2 <- qda(train_data[,1] ~., data = train_data[,2:14])
  pred_2 <- predict(model_2, test_data[,-1])$class
  te2 <- round(mean((pred_2!=test_data$Diabetes_binary)),8)

  # Model 3: Naive Bayes
  model_3 <- naiveBayes(train_data[,1] ~., data = train_data[,2:14])
  pred_3 <- predict(model_3, test_data[,-1])
  te3<- round(mean((pred_3!=test_data$Diabetes_binary)),8)

  # Model 4: Multinomial logisitic regression
  model_4 <- multinom(train_data[,1] ~., data = train_data[,2:14])
  pred_4 <- predict(model_4, test_data[,-1])
  te4 <- round(mean((pred_lr_test!=test_data$Diabetes_binary)),8)

  # Model 5: KNN with several values
  k_list <- c(2,3,4,5,6);
  xnew <- train_data[,-1];
  xnew2 <- test_data[,-1];
  train_errors <-NULL
  test_errors <-NULL
  for (i in 1: 5){
    kk <- k_list[i];
    pred5 <- knn(train_data[,-1], xnew, train_data[,1], k=kk);
    train_errors <- rbind( train_errors, cbind(kk,
                                             mean( pred5 !=
train_data[,1])));
    pred5.test <- knn(train_data[,-1], xnew2, train_data[,1], k=kk);
    test_errors <- rbind( test_errors, cbind(kk,
                                           mean( pred5.test!=
                                                   test_data[,1])));
  }
  results_k <- data.frame(
    K = train_errors[, 1],  # K-values
    Train_Error = train_errors[, 2],  # Training errors
    Test_Error = test_errors[, 2]  # Testing errors
  )
  # results
  #k=3 is the best k value
  pred_5 <- knn(train_data[,-1], test_data[,-1], train_data[,1], k=5)
  te5 <- round(mean((pred_5!=test_data$Diabetes_binary)),8)

  # Model 6: Random Forest
  model_6 <- randomForest(Diabetes_binary ~., data = train_data, mtry=4,
ntree=500)
  pred_6 <- predict(model_6, test_data[,-1])
  te6 <- round(mean((pred_6 != test_data$Diabetes_binary)), 8)

  # Model 7: Boosting
  # Identifying columns that need one-hot encoding
  # Ensure labels are binary (0 or 1)
  train_data$Diabetes_binary <- as.numeric(train_data$Diabetes_binary == 1)
  test_data$Diabetes_binary <- as.numeric(test_data$Diabetes_binary == 1)
  vars_to_encode <- c("HighBP", "HighChol", "CholCheck", "Smoker","Stroke",
                      "HeartDiseaseorAttack","HvyAlcoholConsump","GenHlth",
                      "DiffWalk", "Sex", "Age", "Income")

  # Performing one-hot encoding
  dummy_model_train <- dummyVars(~ ., data = train_data[, vars_to_encode])
  encoded_train_data <- predict(dummy_model_train, newdata = train_data)
  dummy_model_test <- dummyVars(~ ., data = test_data[, vars_to_encode])
  encoded_test_data <- predict(dummy_model_test, newdata = test_data)

  # Rebuilding the dataset excluding the original columns that were encoded
  columns_to_remove <- vars_to_encode
  full_train_data <- cbind(train_data[, !(names(train_data) %in%
columns_to_remove)], encoded_train_data)
  full_test_data <- cbind(test_data[, !(names(test_data) %in%
columns_to_remove)], encoded_test_data)
  # Add this conversion just before creating the DMatrix
  full_train_data$Diabetes_binary <-
as.numeric(full_train_data$Diabetes_binary == 1)
  full_test_data$Diabetes_binary <-
as.numeric(full_test_data$Diabetes_binary == 1)
  # Defining parameter grid for XGBoost
  param_grid_xgb <- list(
    list(eta = 0.01, max_depth = 3, gamma = 0),
    list(eta = 0.1, max_depth = 6, gamma = 0.1),
    list(eta = 0.3, max_depth = 9, gamma = 0.3)
  )

  # Function to train XGBoost model using different parameters
  cross_validate_xgb <- function(params, full_train_data) {
    dtrain <- xgb.DMatrix(data = as.matrix(full_train_data[, -1]), label =
full_train_data$Diabetes_binary)
    model <- xgboost(
      params = params,
      data = dtrain,
      nrounds = 100,
      objective = "binary:logistic",
      eval_metric = "error"
    )
    return(model)
  }

  # Cross-validation across parameter grid
  models_xgb <- lapply(param_grid_xgb, function(params)
cross_validate_xgb(params, full_train_data))

  # Evaluate each model and find the best one
  errors_cv_xgb <- sapply(models_xgb, function(mod) {
    pred <- predict(mod, as.matrix(full_train_data[, -1]))
    mean(pred != full_train_data$Diabetes_binary)
  })

  best_model_index <- which.min(errors_cv_xgb)
  best_parameters <- param_grid_xgb[[best_model_index]]
  best_model <- models_xgb[[best_model_index]]

  # Retraining the best model with selected parameters
  mod_final <- xgboost(
    data = as.matrix(full_train_data[, -1]),
    label = full_train_data$Diabetes_binary,
    nrounds = 100,
    params = best_parameters
  )

  pred_test_final <- predict(mod_final, as.matrix(full_test_data[, -1]))
  binary_pred_test_final <- ifelse(pred_test_final >= 0.5, 1, 0)
  te7 <- round(mean(binary_pred_test_final !=
```

```r
full_test_data$Diabetes_binary), 8)

  #Collect the testing errors
  TEALL = cbind(te1, te2, te3, te4, te5,te6, te7)
}
results_k
#best k value = 3
TEALL
mod_final$params
colnames(TEALL) <- c('LDA','QDA','Naive Bayes','Logistic
Regression','KNN','Random Forest', 'Boosting')
#Before Cross Validation
model_results
#After Cross Validation
TEALL

# From the mean standard error, we found the LDA to be the best

#------------------------------ The End ------------------------------
```