# ST 563

# Introduction to Statistical Learning

## Diabetes Prediction in Pima Indians Diabetes Database

**Authors:**

Chung-Pang Hsu, Kerem Ziya Akdemir, Yan Liu

**Team Number:**

Group J

**Submission Date:** 12/2/2021

## 1. Introduction

Diabetes is a common health problem, and like in every disease, early diagnosis is important. Understanding the underlying causes of diabetes is crucial in order to prevent it from the very beginning. Correlating diabetes cases with some predictors may help in this regard. In this project, Pima Indians Diabetes Database is used to detect diabetes cases, and to understand the important factors causing this disease. The dataset is available on Kaggle via UCI Machine Learning (UCI Machine Learning, 2016).

The scientific questions investigated within the scope of this project are what the important predictors for diabetes are, and which classification model works best to predict diabetes cases by using different predictors. With this purpose in mind, an exploratory data analysis has been conducted at first. There were numerous invalid and missing data, and these were handled with different data imputation techniques in R. After that, various machine learning models are used to classify diabetes outcome of a patient, including k-nearest neighbors (KNN), linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), logistic regression, naive Bayes, support vector machine (SVM), classification tree, boosting tree, and random forest. 10-fold cross-validation (CV) is used in outer loops to reduce the variance of the results, and different sampling approaches including 5 fold CV and repeated CV.

Section 2 of this report represents the methods used for this project. It starts with exploratory data analysis, followed by data cleaning and imputation techniques. After that, the models used are detailed individually to clarify the approaches and techniques used. In section 3, model results are analyzed and discussed to select the best model as well as important predictors. Finally, all the cited figures and tables, R codes used throughout the project, and the related output are listed in the Appendix.

## 2. Methods

### 2.1. Data Preprocessing

Data consist of 768 observations and 8 predictors. The last column is outcome coded as a class variable. 0 means patients do not have diabetes, and 1 means patient has diabetes. 8 predictors are elaborated below:

- **Pregnancies:** Number of times pregnant
- **Glucose:** Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- **Blood Pressure:** Diastolic blood pressure (mmHg)
- **Skin Thickness:** Triceps skinfold thickness (mm)
- **Insulin:** 2-Hour serum insulin (mu U/ml)
- **BMI:** Body mass index ($kg/m^2$)
- **Diabetes pedigree function**
- **Age:** Age of the patient in years

After looking at the data, it is detected that some 0 values in glucose, blood pressure, skin thickness, insulin, and BMI columns, which does not make sense. These values are regarded as invalid. Since only 0.6%, 4.5% and 1.4% of values in glucose, blood pressure, and BMI are zeros respectively, invalid entries in these columns are replaced with medians of those predictors. Medians are calculated just for values higher than 0 to eliminate the impact of invalid values on the median.

29% and 48% of data in skin thickness and insulin are 0, which is high. Thus, replacing those invalid values with the median was not a good option. In this sense, a more complex approach was used to replace invalid values in skin thickness and insulin. It is observed that skin thickness is correlated with BMI and the relationship was looking linear. Linear regression between the valid values of BMI and skin thickness has been done to predict invalid skin thickness values. The resultant regression line on top of the original data can be seen in Figure 1. Regression diagnostics were carried out to make sure the two predictors are linearly correlated. The results of diagnostics are shown in Figure 2. In Residuals vs Fitted plot, there is no clear pattern. The red trend line is nearly flat, the model seems to capture true linear relationship. Also, from the same plot, we can see that the variability of residuals is not that pronounced. Thus, we can say that the model satisfies

constant variance assumption adequately. Also, from Normal Q-Q plot, model seems to satisfy normality of errors. Prediction errors are acquired with only valid data. Since errors look normally distributed, a normal distribution is created with the mean and standard deviation of the regression errors. Random values from this distribution are added on top of our predictions to add a stochastic element between the predictors and to decrease collinearity issues among variables.

To replace invalid insulin values, a similar regression approach was adopted. Insulin values are highly correlated with glucose levels, which makes sense from a biological perspective. However, the relationship was not linear. To transform the relationship to linear, insulin values are transformed by utilizing natural logarithm (ln). The relationship between ln transformed insulin and glucose can be seen in Figure 3. After transformation, the relationship was looking like linear. Again, a linear regression model is used and diagnostics of this regression can be seen in Figure 4. In Residuals vs Fitted plot, there is no clear pattern. The red trend line is nearly flat, the model seems to capture true linear relationship. Also, from the same plot, we can see that the variability of residuals is not that pronounced. Thus, we can say that the model satisfies constant variance assumption adequately. Also, from Normal Q-Q plot, there are some values not fitted to normal distribution but it seems reasonable to assume the normality of errors. Again, prediction errors are acquired with only valid data. Since errors look normally distributed, a normal distribution is created with the mean and standard deviation of the regression errors. Random values from this distribution are added on top of our predictions to add a stochastic element between the predictors and to decrease collinearity issues among variables. The final regression line between glucose and insulin can be seen in Figure 5.

### 2.2. Exploratory Data Analysis

After preprocessing the dataset, an exploratory data analysis was conducted to understand useful relationship between predictors that can help classify the outcome. A pairs plot showing correlations, density plots, and boxplots are shown in Figure 6, Figure 7, and Figure 8. Correlation plots are helpful to visualize the pairwise relationships between a set of quantitative variables by displaying their correlations using color or shading. From Figure 6, the highest correlation was found between BMI and Skin Thickness with a 0.679 Pearson correlation coefficient. None of the

correlation coefficients is above 0.7, which is considered as being highly related. There seems to be no strong correlation among different variables other than insulin-glucose and BMI-skin thickness. These correlations make sense since we modeled them with respect to one another. It seems like output is mostly related to glucose levels. Other relevant predictors might be pregnancies, body mass index, and age, but their predictive power may be lower than glucose levels. It makes sense that glucose levels are the leading predictor for diabetes detection.

Multicollinearity is a statistical phenomenon in which predictor variables in a logistic regression model are highly correlated. If more than two predictors are closely related, we call the situation multicollinearity. Such a situation cannot be detected by simply inspecting the correlation plot. Instead, we may look at the variance inflation factor (VIF). Generally, a VIF value larger than 5 or 10 indicates a problematic amount of multicollinearity. In this logistic regression model, all of the variables have VIFs that are lower than 2 (see Appendix for more detail), indicating there is no strong multicollinearity.

### 2.3. Models Used to Detect Diabetes

Before performing the model building, the entire dataset is required to be split into a test set and training set. All training or/and tuning procedures are only executed within the training dataset. The test set is not involved any training procedure. The selected data split method is bootstrap in this project. The whole dataset was bootstrapped into 10 splits. The final accuracy rate is calculated as the mean of 10 splits. The following is the model method in this report

#### 2.3.1. Logistic Regression

Logistic regression models are used mostly as a tool for data analysis and inference, where the main goal is to understand the role of the predictors in explaining the outcome. Logistic regression does not make many of the key assumptions of linear regression and general linear models that are based on ordinary least squares algorithms – particularly regarding linearity, normality, homoscedasticity, and measurement level. Our data meets all the assumptions for logistic regression. First, the response is Outcome which is binary. Second, the

observations are different patients which are independent of each other. Third, there is no multicollinearity among the predictors as we proved above. Therefore, we fit the full data with a logistic regression model. Based on the summary (Table 1.), the Residual deviance is more than 200 less than the null deviance at 528 degrees of freedom, indicating there is at least one predictor that is significantly associated with response, Outcome. We further want to test whether Glucose has any association with Outcome, $H_0$: $\beta$(glucose)=0 vs $H_1$: $\beta$ (glucose)$\neq$0. The p-value is less than 0.001, rejecting $H_0$. There is enough evidence to support that Glucose has a significant association with Outcome. Based on similar tests, we found Pregnancies, BMI, and DiabetesPedigreeFunction also has significant associations with Outcome. This result is further confirmed when we performed backward and best selections, since Glucose, Pregnancies, BMI, and DiabetesPedigreeFunction also get selected in the above two models. (see Part 4. for more detail) In terms of predicting performance, the accuracy rates are very close to each other among all three logistic regression models. However, the Best selection model has a much lower area under ROC curves than the other two. (Table 2)

### 2.3.2. Classification Tree and Random Forest

One main advantage of trees is that they can be displayed graphically, and are easily interpreted even by a non-expert - this is especially true for small trees. The reason is that trees are very easy to explain to people since they more closely mirror human decision-making. Also, trees can easily handle categorical predictors without the need to create dummy variables. Since Outcome is binary, we applied a Classification Tree to fit our data. The pruned tree using minimum cp is present in Figure 9.

In interpreting the results of a classification tree, we are often interested in both the class prediction corresponding to a particular terminal node region and the class proportions among the training observations that fall into that region. An ideal node would be the one with all observations that are from the same class for a classification problem. As shown in Figure 9, Glucose<124 is the root node with a 1-No Information Rate (NIR)=0.36. Then 44% of observations with Glucose>124 were further split by BMI<30. These are consistent with our

conclusion in the logistic regression model that Glucose and BMI have a significant association with Outcome.

However, the major disadvantage of Classification Trees is that they are unstable, meaning that a small change in the data can lead to a large change in the structure of the optimal tree. This point is proved in our 10 folds cross-validation. The shapes of the optimal trees are significantly different. (see Part 4. for more detail) Therefore, Classification Trees are often relatively inaccurate.

Random Forest is based on the bagging algorithm and uses Ensemble Learning technique. Random forests provide an improvement over bagging by decorrelating the trees. It forces each split to consider only a subset of the predictors. As in bagging, we build a number of decision trees on bootstrapped training samples. However, when building these decision trees, each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates from the full set of p predictors. In this project, we chose mtry=3 as the tuning parameter in the model. As expected, Random Forest showed a better prediction performance than Classification Tree according to the accuracy rates and area under ROC curves. (Table 2.)

A disadvantage of random forest is that the resulting model is often difficult or impossible to interpret, as we are averaging many trees rather than looking at a single tree. We can still compute variable importance scores. Using the varImpPlot() function, we can view the importance of each variable based upon the mean decrease of accuracy. From Figure 10, Glucose is the most important predictor in predicting Outcome. This conclusion is consistent with our conclusion in the logistic regression model that Glucose has a significant association with Outcome.

### 2.3.3. Linear Discriminant Analysis (LDA) and Quadratic Discriminant Analysis (QDA)

LDA and QDA are generative models, which means that they model how input predictors are distributed within each class. They make use of the Bayes theorem to convert this information to classification probabilities. One of our assumptions within these models is that predictors are normally distributed. Thus, we use normal (Gaussian) densities to model the distributions of predictors.

There is one strong difference between LDA and QDA. In LDA, we model predictor distributions by using Gaussian densities with different means but the same covariance matrix for each class. On the other hand, in QDA, we model predictor distributions by using Gaussian densities with different means and also difference covariance matrices for each class. We assume the covariance matrices for each class should be really close or the same in LDA. As a result of this, the classification boundaries of LDA are linear whereas, in QDA, they are defined with quadratic equations and have curvatures.

Two different LDA and QDA models were tried for this project. At first, all predictors are used in the models with 5-fold repeated CV. When accuracy rates and area under ROC curves are examined, it is determined that LDA works better than QDA. After that, only Glucose, BMI, Pregnancies, and Age predictors are used. It is observed that both LDA and QDA's classification accuracies increased. This conforms with the exploratory data analysis as these predictors were shown to be important in classifying the outcome. Consequently, we only included LDA and QDA models with only Glucose, BMI, Pregnancies, and Age used as predictors.

### 2.3.4. K-Nearest Neighbors Classification

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors. Both for classification and regression, a useful technique can be to assign weights to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. However, in this project, all nearer neighbors are weighted equally. In the KNN procedure, hyperparameter K is tuned from K=2 to K=51 with 5-fold cross-validation. According to the result, even with tuned K, the KNN performs with the lowest accuracy rate among all other models. It just brings up a few percentages from NIR. It probably can improve some level if local weight is applied.

### 2.3.5. Naive Bayes classifier

Based on Figure 6, none of the correlation coefficients is above 0.7 meaning none of the predictors is highly correlated to any other predictors. In other words, within each class, the predictors are near to be independent. Thus, Naïve Bayes is considered a method. In statistics, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naïve) independence assumptions between the features. However, each class of distribution is not a truly perfect normal distribution. The kernel density estimation might be slightly better matching than the model without kernel density estimation. To compare the performance of kernel density estimation, there are two naive Bayes models built with and without kernel density estimation. From the accuracy of the average rate and AUC, the model with kernel density estimation is better than the one without it a bit. Thus, kernel density estimation is suggested to be used in the model.

### 2.3.6. Support Vector Machine

In machine learning, support vector machines are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. SVMs are one of the most robust prediction methods, being based on statistical learning frameworks. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. Also, there are some hyperparameters needed to be tuned in these SVM models. For the example of the linear and Radial Kernel in the project, the cost is the hyperparameters for both models and sigma only for Radial model. In the accuracy rate, the linear SVM is much better than Radial SVM. It can explain that linear model is more suitable than Radial SVM.

## 3. Discussion and Conclusions

In this project, we fit the Pima Indians Diabetes dataset to various machine learning models including KNN, LDA, QDA, Logistic regression, Naive Bayes, SVM, Classification Tree,

Boosting tree, and Random Forest. The predicting performance of the above models were summarized in Table 2. We averaged the predicting accuracies and the Area Under the ROC curves (AUCs) from 10 bootstrap splits and found that the Logistic regression model with all the 8 predictors has the highest accuracy (0.77) and AUC (0.84), indicating it is the best model to predict diabetes in patients. No information rate (NIR) of the original data was 0.65. This means that our logistic regression model was made an improvement and successful in classifying diabetes cases. Based on the Wald Test results of this model, we revealed that plasma glucose concentration has a significant association with a diabetes diagnosis. Consistently, the importance plot in Random Forest showed that blood Glucose is the most important predictor in diabetes prediction. This outcome also makes sense from a health science perspective.

One shortcoming of this study was the imbalance issue of the outcome. The ideal dataset for binary type data is balanced. It means 50% "yes" or "1", and 50% "no" or "0". If we have a highly imbalanced dataset, we could select the wrong model as a good by being highly accurate. Therefore, a balanced dataset is recommended. In the evaluation of this project, the dataset are not perfectly balanced. The outcomes of diabetes cases are distributed as 0 (65%) and 1 (35%). Although the imbalance issue is not that severe, synthetic minority over-sampling technique (SMOTE) may be used to get more evenly distributed outcomes. This could have increased our model's ability of successful classification. Moreover, there were a lot of missing data as discussed before. Even though we filled the invalid data with data imputation techniques, a fuller data set, and maybe even with a higher number of observations would have a positive impact on the accuracy of our model. Nevertheless, we were able to detect diabetes cases with our model by using different predictors, and associate predictors, which are most importable in diabetes detection.

# 4. Appendix

## 4.1. Figures and Tables

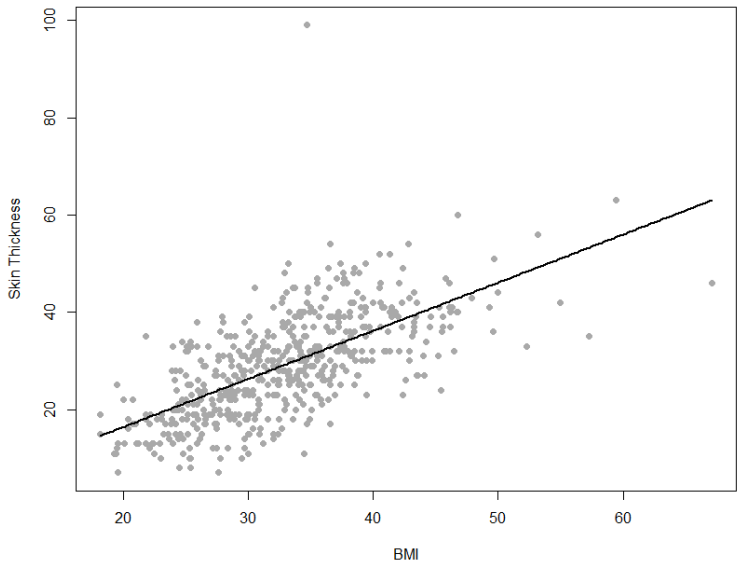



**Figure 1:** Linear Regression between BMI and Skin Thickness

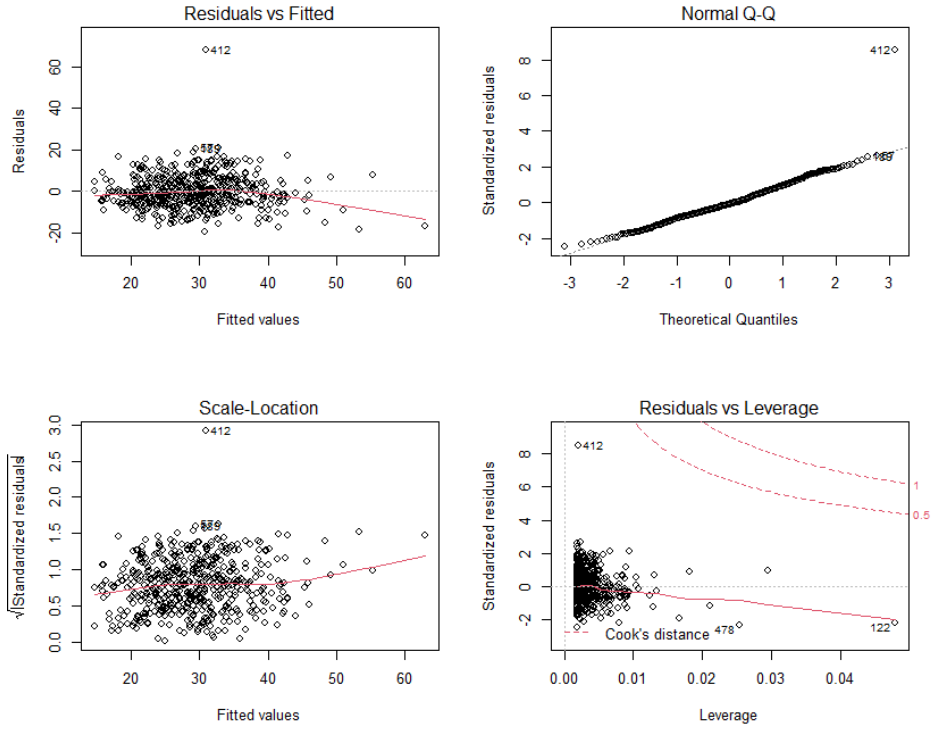



**Figure 2:** Model Diagnostics for Skin Thickness Regression

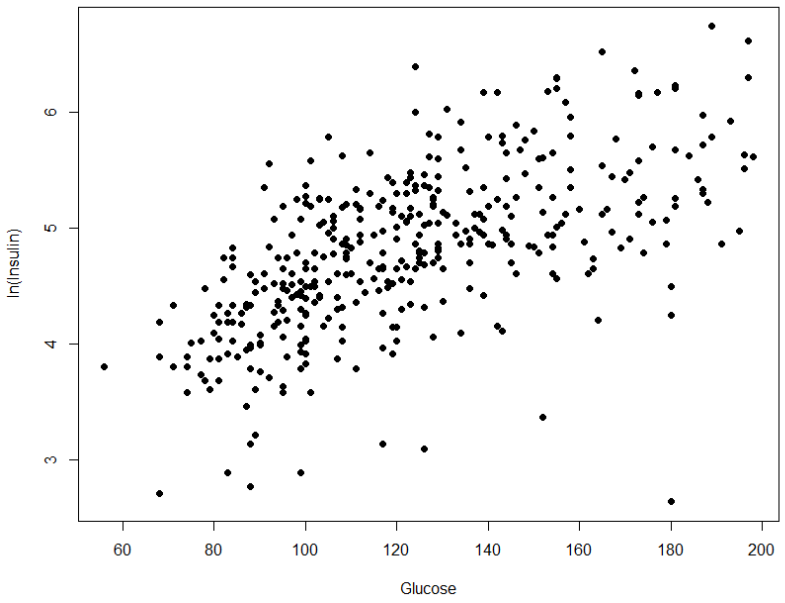



**Figure 3:** Scatterplot of ln Transformed Insulin vs Glucose

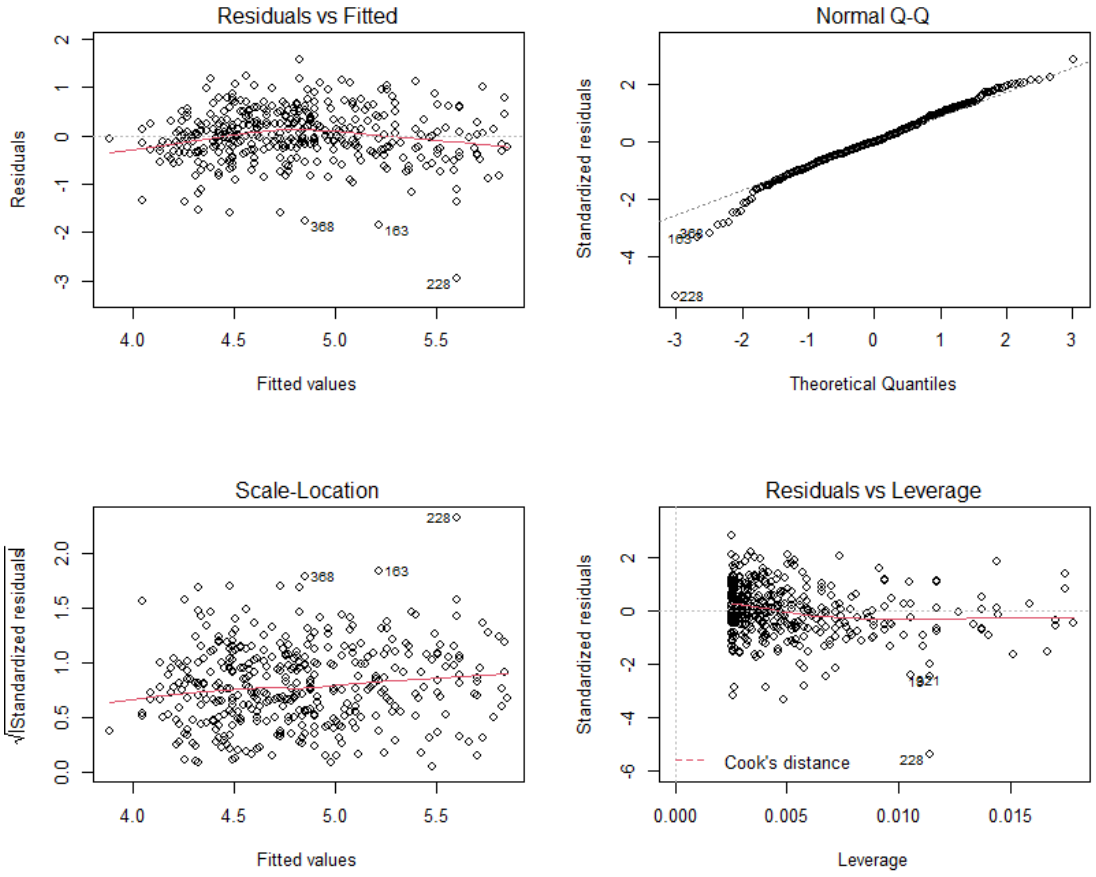



**Figure 4:** Model Diagnostics for Insulin Regression

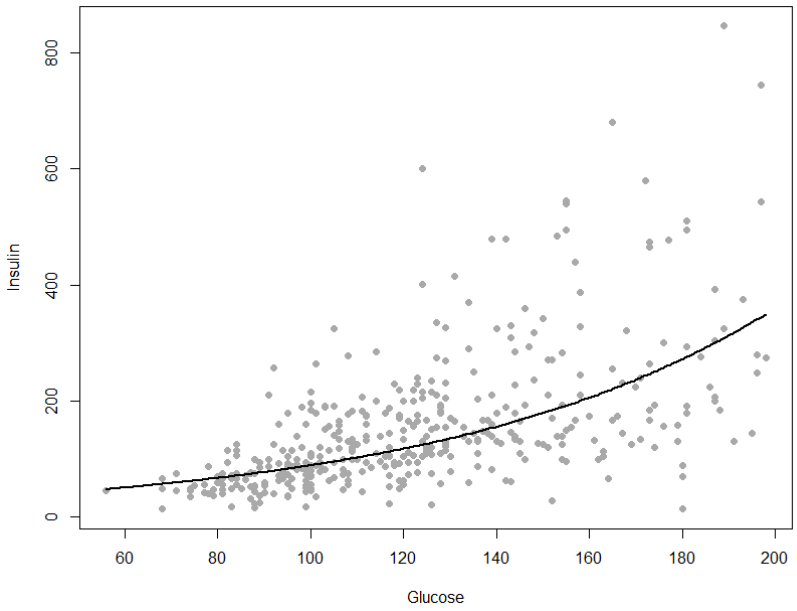



**Figure 5:** Log-Linear Regression between Glucose and Insulin


**Figure 6:** Pairs plot of All Predictors

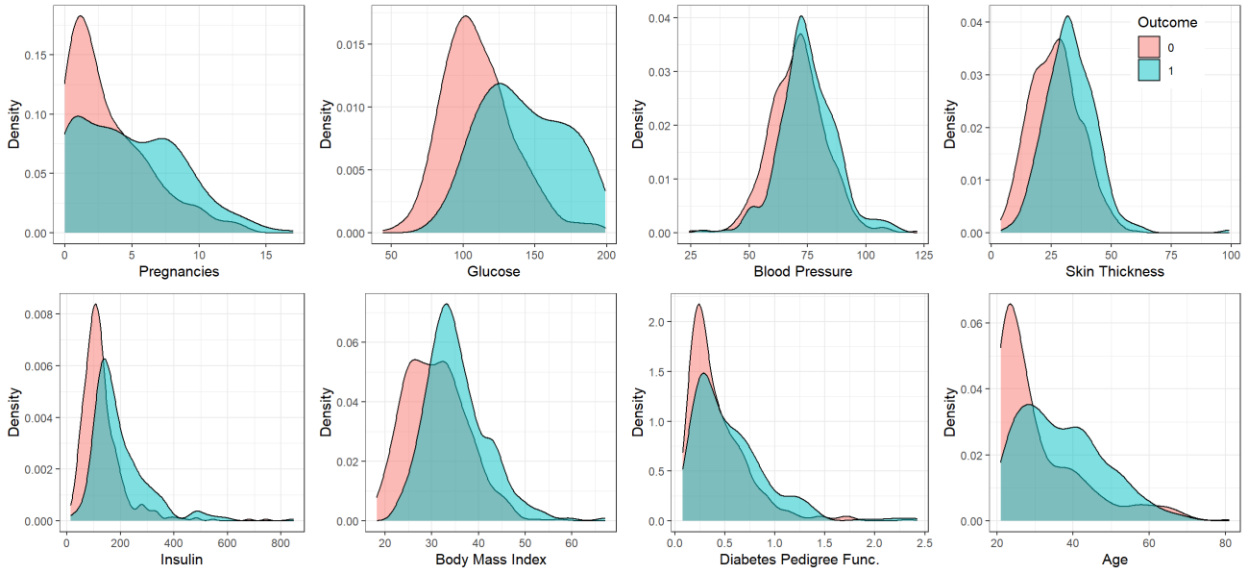



**Figure 7:** Density plot of All Predictors

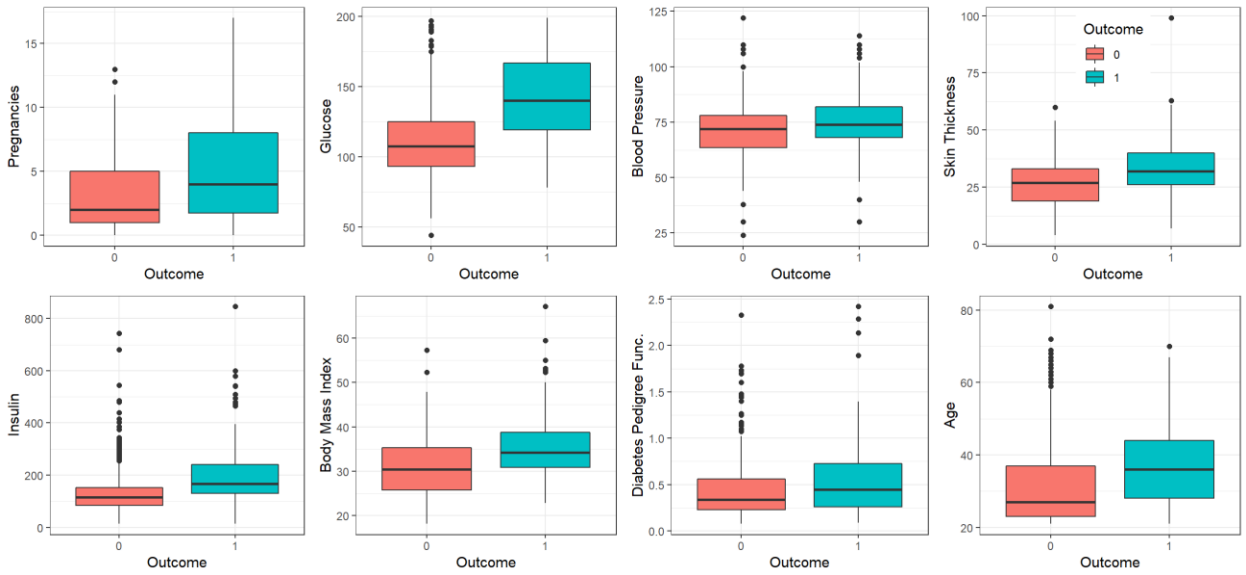



**Figure 8:** Boxplot of All Predictors

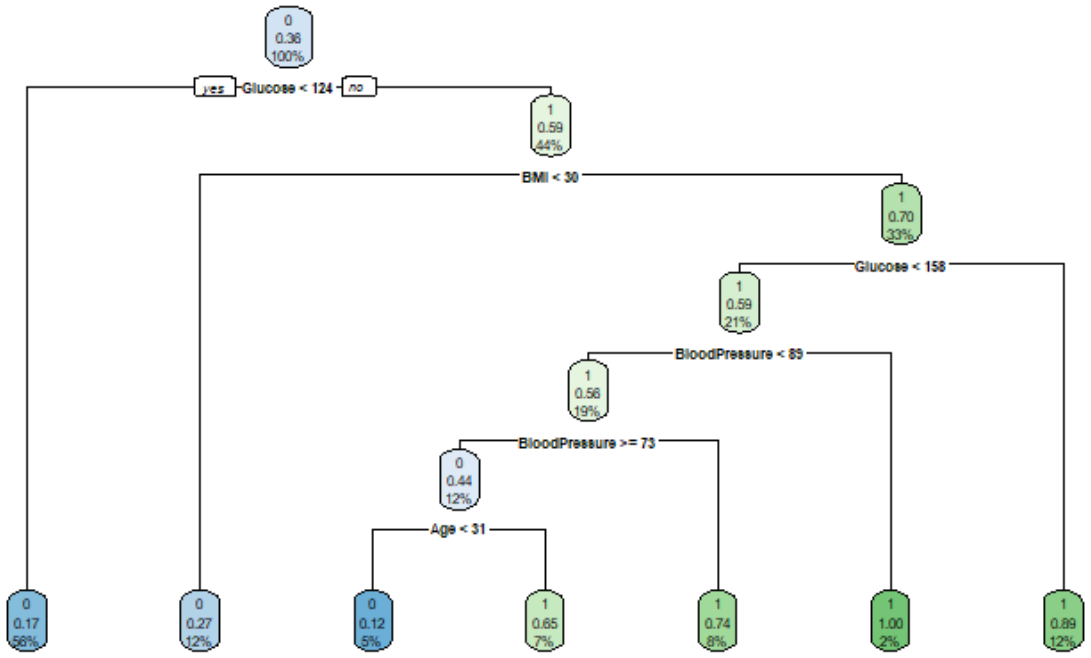



**Figure 9:** Pruned tree using minimum cp for the training data

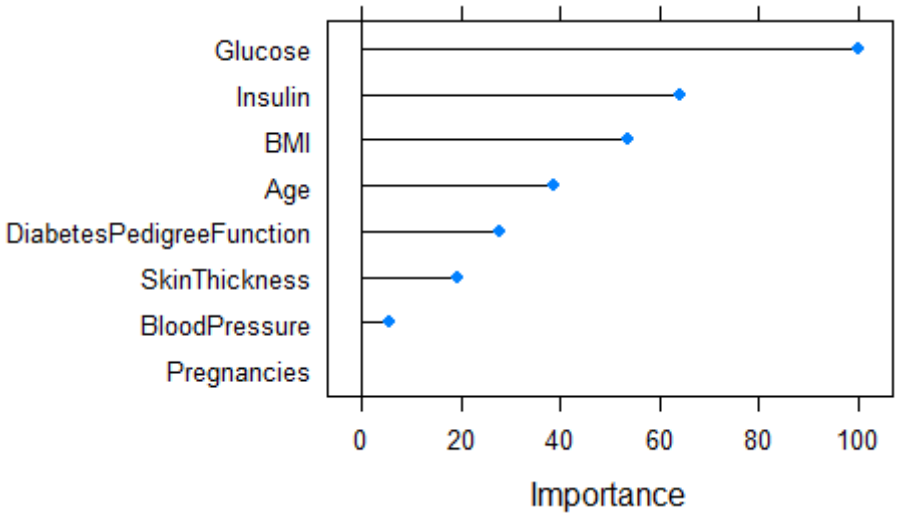



**Figure 10:** Relative importance of predictors based on the mean decrease of accuracy

**Table 1.** Logistic regression model's Coefficients and other parameters.

```
## glm(formula = Outcome ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8173  -0.7216  -0.3750   0.7163   2.3246
##
## Coefficients:
##                            Estimate Std. Error z value Pr(>|z|)
## (Intercept)              -9.1378438  1.0144784  -9.007  < 2e-16 ***
## Pregnancies               0.1016023  0.0384362   2.643 0.008208 **
## Glucose                   0.0354587  0.0053220   6.663 2.69e-11 ***
## BloodPressure            -0.0136066  0.0101248  -1.344 0.178984
## SkinThickness             0.0186847  0.0147773   1.264 0.206082
## Insulin                  -0.0001191  0.0015087  -0.079 0.937057
## BMI                       0.0892003  0.0241463   3.694 0.000221 ***
## DiabetesPedigreeFunction  0.8977257  0.3650915   2.459 0.013936 *
## Age                       0.0207614  0.0114574   1.812 0.069978 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 699.06  on 536  degrees of freedom
## Residual deviance: 494.77  on 528  degrees of freedom
## AIC: 512.77
##
## Number of Fisher Scoring iterations: 5
```

**Table 2.** The averaged Accuracy and AUC of ROC for each model

| Model | Accuracy | AUC of ROC |
|---|---|---|
| KNN | 0.6736 | 0.6881 |
| LDA | 0.7655 | 0.8417 |
| QDA | 0.7465 | 0.8116 |
| LDA with few predictors | 0.7654 | 0.8388 |
| QDA with few predictors | 0.7567 | 0.8245 |
| Logistic Regression | 0.77 | 0.8418 |
| Back | 0.7676 | 0.8405 |
| Best | 0.77 | 0.7086 |
| Naive Bayes | 0.7616 | 0.8289 |
| Naive Bayes With Kernel | 0.7668 | 0.8455 |
| SVM Linear | 0.7658 | 0.7148 |
| Radial | 0.6972 | 0.6193 |
| Classification Tree | 0.7348 | 0.7454 |
| Boost Tree | 0.7559 | 0.8173 |
| Random Forest | 0.7651 | 0.8377 |

### 4.2. Model Codes

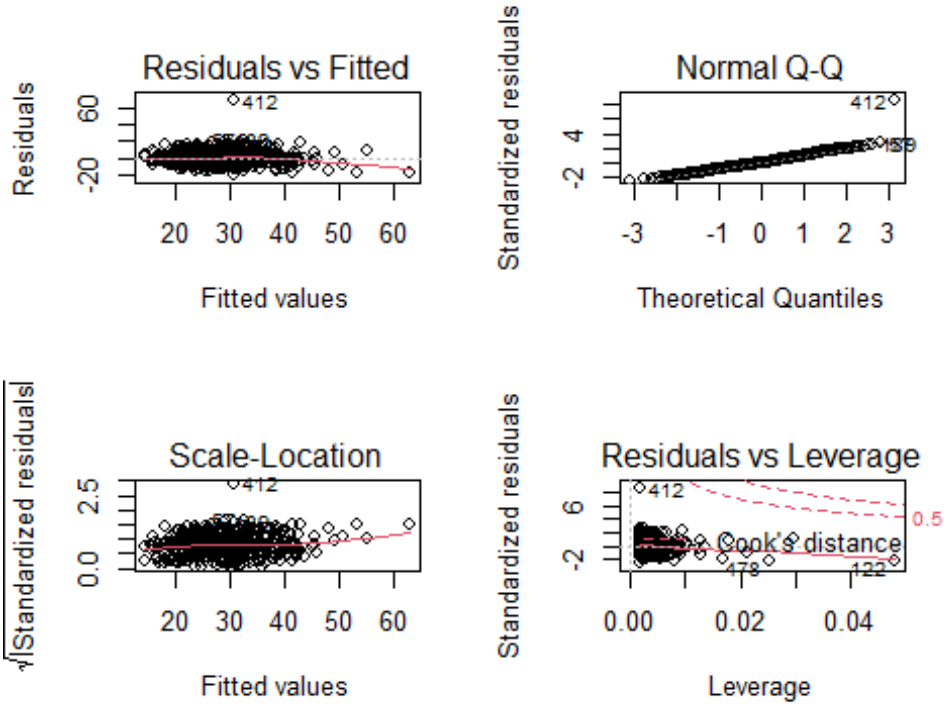
## Import library

```
library(caret)
library(ISLR2)
library(rsample)
library(pROC)
library(ggplot2)
library(klaR)
library(splines)
library(tidyverse)
library(corrplot)
library(GGally)
library(gridExtra)
library(glue)
library(PerformanceAnalytics)
```

```
library(rpart)
library(rpart.plot)
library(bestglm)
library(randomForest)
library(kernlab)
```

## Data cleaning and processing

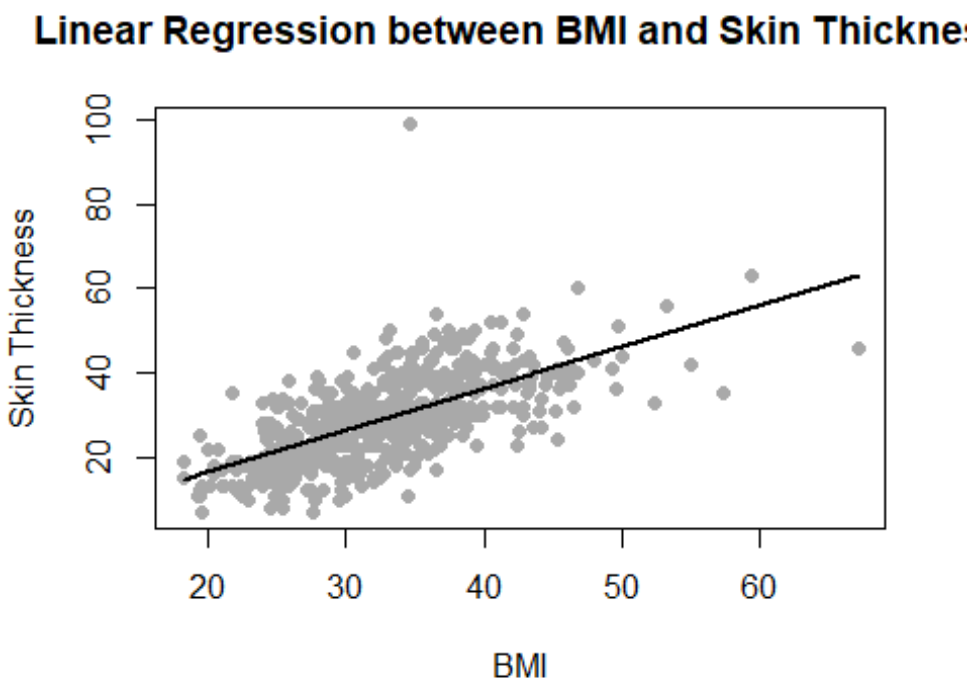
```
#Import diabetes Table and view. Read diabetes data using relative add
ress
diabetes <- read.csv("diabetes.csv", header = TRUE)

#setting seed for reproducible results
set.seed(12345)

#altering 0 values in glucose, blood pressure and BMI as medians of th
at predictors
diabetes$Glucose[diabetes$Glucose==0] <- median(diabetes$Glucose[diabe
tes$Glucose>0])
diabetes$BloodPressure[diabetes$BloodPressure==0] <- median(diabetes$B
loodPressure[diabetes$BloodPressure>0])
diabetes$BMI[diabetes$BMI==0] <- median(diabetes$BMI[diabetes$BMI>0])

#for skin thickness, filling missing data (zeros) by building a linear
 regression with BMI
#selecting rows where thickness is 0
invalid_skin_rows <- as.integer(rownames(diabetes[diabetes$SkinThickne
ss==0,]))

#filtering data with respect to valid and invalid data rows (where thi
ckness is 0)
invalid_BMI <- diabetes[invalid_skin_rows,'BMI']
valid_BMI <- diabetes[-invalid_skin_rows,'BMI']
valid_skin <- diabetes[-invalid_skin_rows,'SkinThickness']

#building a linear model and predicting for missing thicknesses
linear_model_data <- data.frame(BMI=valid_BMI, Thickness=valid_skin)
linear_model_for_skin <- lm(Thickness ~ BMI, data = linear_model_data)

par(mfrow = c(2,2))
plot(linear_model_for_skin)
```




When we plot the regression diagnostics graphs, we seem to satisfy the assumptions. From Residuals vs Fitted plot, there is no clear pattern. The red trend line is nearly flat, the model seems to capture true linear relationship. Also, from the same plot, we can see that he variability of residuals are not that pronounced. Thus, we can say that the model satisfies constant variance assumption adequately. Also, from Normal Q-Q plot, model seems to satisfy normality of errors.

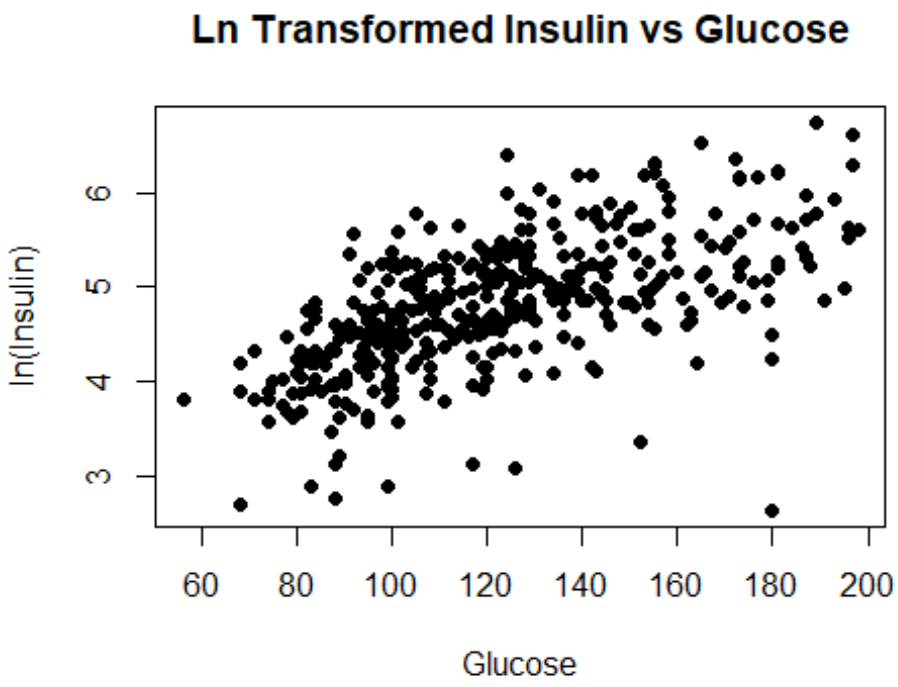
```
predict_skin <- predict(linear_model_for_skin, newdata = data.frame(BM
I=invalid_BMI))

#finding errors of our linear regression model by predicting for valid
 data onlv and creating random errors
predict_error <- predict(linear_model_for_skin, newdata = data.frame(B
MI=valid_BMI))
error_lm <- valid_skin - predict_error
error_mean <- mean(error_lm)
error_sd <- sd(error_lm)
random_errors <- rnorm(length(predict_skin), mean=error_mean, sd=error
_sd)

#finding final predictions by adding random errors to our predictions
(this step is done to reduce collinearity between predictors)
final_prediction_skin <- predict_skin + random_errors
diabetes$SkinThickness[invalid_skin_rows] <- round(final_prediction_sk
in)
```

```r
# Create prediction grid to see regression line
xgrid <- list(BMI = seq(min(valid_BMI), max(valid_BMI), len=201))
# Perform prediction to see regression line
fitted_values <- predict(linear_model_for_skin, newdata = xgrid)
par(mfrow = c(1,1))
plot(valid_BMI, valid_skin, pch=19, col = "darkgray", xlab = "BMI", yl
ab = "Skin Thickness", main = "Linear Regression between BMI and Skin
Thickness")
lines(xgrid$BMI, fitted_values, lwd=2)
```



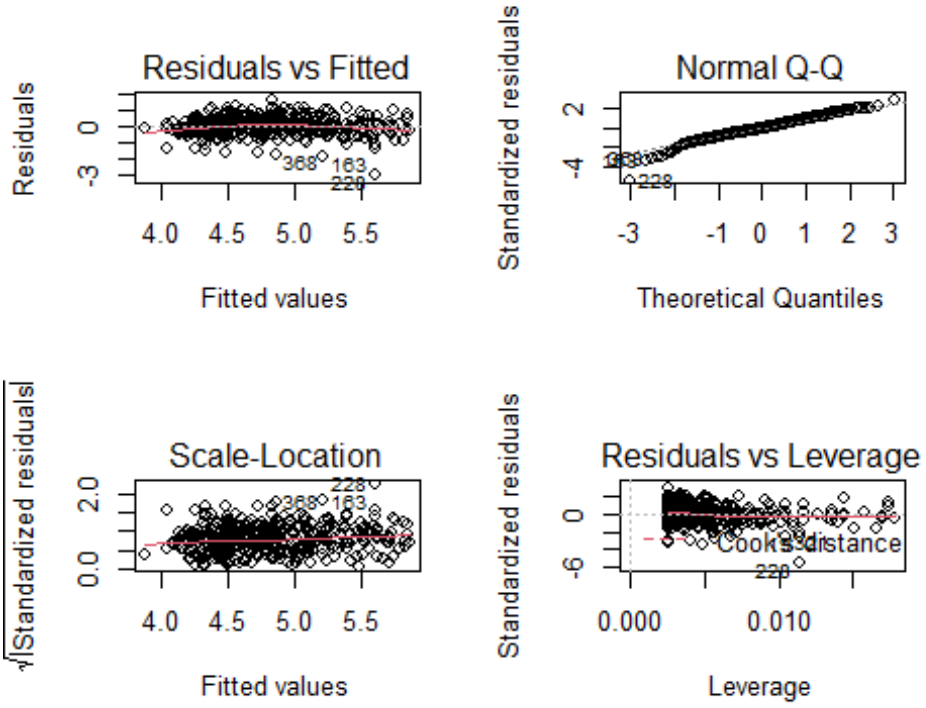

For insulin, filling missing data (zeros) by building a log linear regression with glucose

```r
#selecting rows where insulin is 0
invalid_insulin_rows <- as.integer(rownames(diabetes[diabetes$Insulin=
=0,]))
#filtering data with respect to valid and invalid data rows (where ins
ulin is 0)
invalid_glucose <- diabetes[invalid_insulin_rows,'Glucose']
valid_glucose <- diabetes[-invalid_insulin_rows,'Glucose']
valid_insulin <- diabetes[-invalid_insulin_rows,'Insulin']

plot(valid_glucose, log(valid_insulin), pch=19, col = "black", xlab =
"Glucose", ylab = "ln(Insulin)", main = "Ln Transformed Insulin vs Glu
cose")
```




When we natural log transform insulin, they seem to follow a fairly linear relationship.

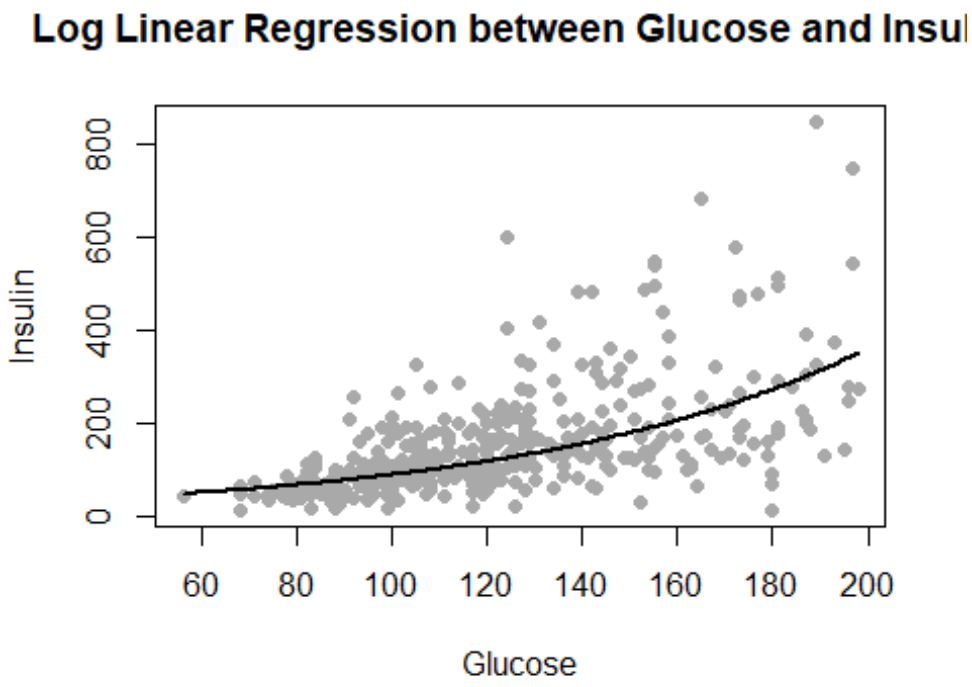
```
#building a log linear model and predicting for missing insulin data
log_model_data <- data.frame(Glucose=valid_glucose, Insulin=valid_insu
lin)
log_model_for_insulin <- lm(log(Insulin) ~ Glucose, data = log_model_d
ata)

par(mfrow = c(2,2))
plot(log_model_for_insulin)
mtext("Model Diagnostics for Insulin", side=3, line=22, at=-0.005)
```




When we plot the regression diagnostics graphs, we seem to satisfy the assumptions. From Residuals vs Fitted plot, there is no clear pattern. The red trend line is nearly flat, the model seems to capture true linear relationship. Also, from the same plot, we can see that he variability of residuals are not that pronounced. Thus, we can say that the model satisfies constant variance assumption adequately. Also, from Normal Q-Q plot, there are some values not fitted to normal distribution but it seems reasonable to assume the normality of errors.

```
predict_insulin <- exp(predict(log_model_for_insulin, newdata = data.f
rame(Glucose=invalid_glucose)))

#finding errors of our log linear regression model by predicting for v
alid data only and creating random errors
predict_error_insulin <- exp(predict(log_model_for_insulin, newdata =
data.frame(Glucose=valid_glucose)))
error_insulin <- valid_insulin - predict_error_insulin
error_mean_insulin <- mean(error_insulin)
error_sd_insulin <- sd(error_insulin)/10
random_errors_insulin <- rnorm(length(predict_insulin), mean=error_mea
n_insulin, sd=error_sd_insulin)

#finding final predictions by adding random errors to our predictions
(this step is done to reduce collinearity between predictors)
final_prediction_insulin <- predict_insulin + random_errors_insulin
diabetes$Insulin[invalid_insulin_rows] <- round(final_prediction_insul
in)
```

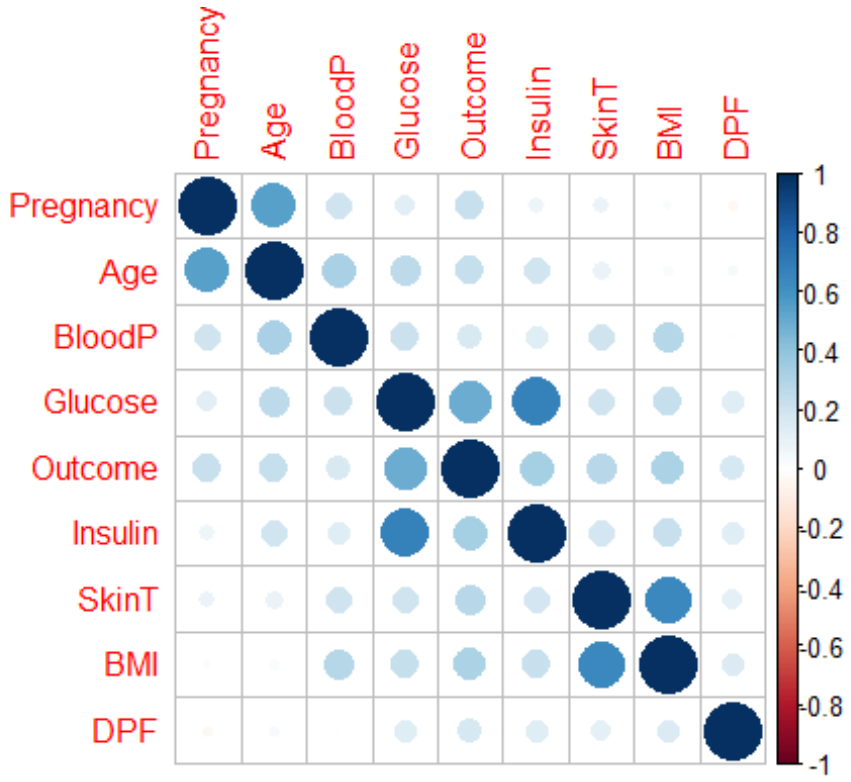
```r
# Create prediction grid to see regression line
xgrid <- list(Glucose = seq(min(valid_glucose), max(valid_glucose), le
n=201))
# Perform prediction to see regression line
fitted_values <- exp(predict(log_model_for_insulin, newdata = xgrid))

par(mfrow = c(1,1))
plot(valid_glucose, valid_insulin, pch=19, col = "darkgray", xlab = "G
lucose", ylab = "Insulin", main = "Log Linear Regression between Gluco
se and Insulin")
lines(xgrid$Glucose, fitted_values, lwd=2)
```



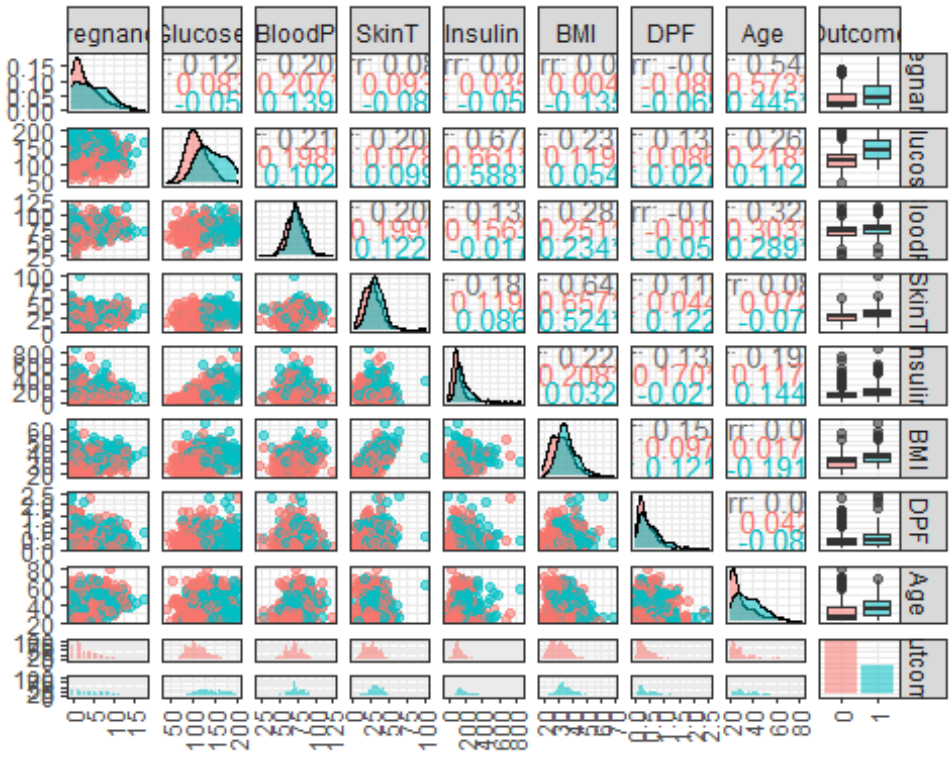

## Variables Inspection

```r
names(diabetes)[names(diabetes) == 'DiabetesPedigreeFunction'] <- 'DPF
'
names(diabetes)[names(diabetes) == 'BloodPressure'] <- 'BloodP'
names(diabetes)[names(diabetes) == 'SkinThickness'] <- 'SkinT'
names(diabetes)[names(diabetes) == 'Pregnancies'] <- 'Pregnancy'

#summary statistics
cat('Descriptive statistics of the data:')
```

```
## Descriptive statistics of the data:

summary(diabetes)

##    Pregnancy         Glucose           BloodP           SkinT      
##  Min.   : 0.000   Min.   : 44.00   Min.   : 24.00   Min.   : 4.00  
##  1st Qu.: 1.000   1st Qu.: 99.75   1st Qu.: 64.00   1st Qu.:21.00  
##  Median : 3.000   Median :117.00   Median : 72.00   Median :29.00  
##  Mean   : 3.845   Mean   :121.66   Mean   : 72.39   Mean   :29.01  
##  3rd Qu.: 6.000   3rd Qu.:140.25   3rd Qu.: 80.00   3rd Qu.:36.00  
##  Max.   :17.000   Max.   :199.00   Max.   :122.00   Max.   :99.00  
##     Insulin           BMI              DPF              Age        
##  Min.   : 14.0   Min.   :18.20   Min.   :0.0780   Min.   :21.00  
##  1st Qu.: 97.0   1st Qu.:27.50   1st Qu.:0.2437   1st Qu.:24.00  
##  Median :130.0   Median :32.30   Median :0.3725   Median :29.00  
##  Mean   :153.8   Mean   :32.46   Mean   :0.4719   Mean   :33.24  
##  3rd Qu.:180.0   3rd Qu.:36.60   3rd Qu.:0.6262   3rd Qu.:41.00  
##  Max.   :846.0   Max.   :67.10   Max.   :2.4200   Max.   :81.00  
##     Outcome     
##  Min.   :0.000  
##  1st Qu.:0.000  
##  Median :0.000  
##  Mean   :0.349  
##  3rd Qu.:1.000  
##  Max.   :1.000

#correlation plot
M<-cor(diabetes)
corrplot(M, order = 'AOE')
```



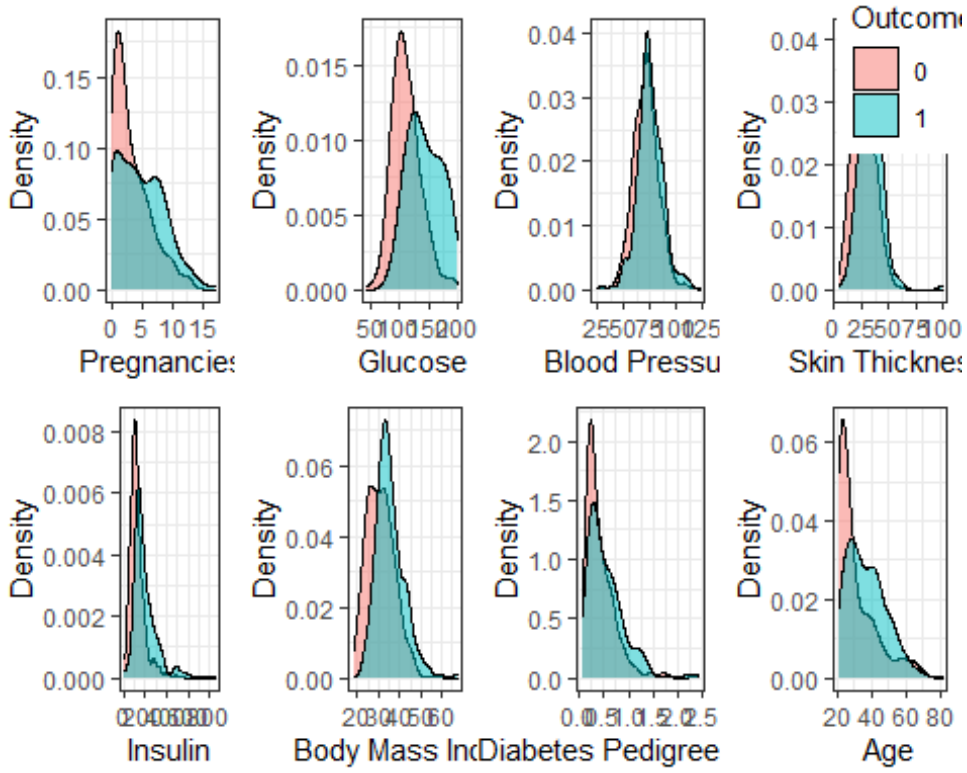

```
#changing outcome to factor
diabetes$Outcome <- as.factor(diabetes$Outcome)

#pairs plot
ggpairs(diabetes, aes(colour = Outcome, alpha = 0.4)) + theme_bw() + t
heme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
```




There seems to be no strong correlation among different variables other than insulin-glucose and BMI-skin thickness. Plotting density graphs for predictors

```
num_predictors <- length(diabetes)-1
names_predictors <- colnames(diabetes)

den1 <- ggplot(diabetes, aes(x = Pregnancy, fill = Outcome)) + geom_de
nsity(alpha = 0.5) + labs(y="Density", x="Pregnancies") + theme_bw() +
 theme(legend.position = "none")
den2 <- ggplot(diabetes, aes(x = Glucose, fill = Outcome)) + geom_dens
ity(alpha = 0.5) + labs(y="Density", x="Glucose") + theme_bw() + theme
(legend.position = "none")
den3 <- ggplot(diabetes, aes(x = BloodP, fill = Outcome)) + geom_densi
ty(alpha = 0.5) + labs(y="Density", x="Blood Pressure") + theme_bw() +
 theme(legend.position = "none")
den4 <- ggplot(diabetes, aes(x = SkinT, fill = Outcome)) + geom_densit
y(alpha = 0.5) + labs(y="Density", x="Skin Thickness") + theme_bw() +
theme(legend.position = c(0.71, 0.82))
den5 <- ggplot(diabetes, aes(x = Insulin, fill = Outcome)) + geom_dens
ity(alpha = 0.5) + labs(y="Density", x="Insulin") + theme_bw() + theme
(legend.position = "none")
den6 <- ggplot(diabetes, aes(x = BMI, fill = Outcome)) + geom_density(
alpha = 0.5) + labs(y="Density", x="Body Mass Index") + theme_bw() + t
heme(legend.position = "none")
den7 <- ggplot(diabetes, aes(x = DPF, fill = Outcome)) + geom_density(
```

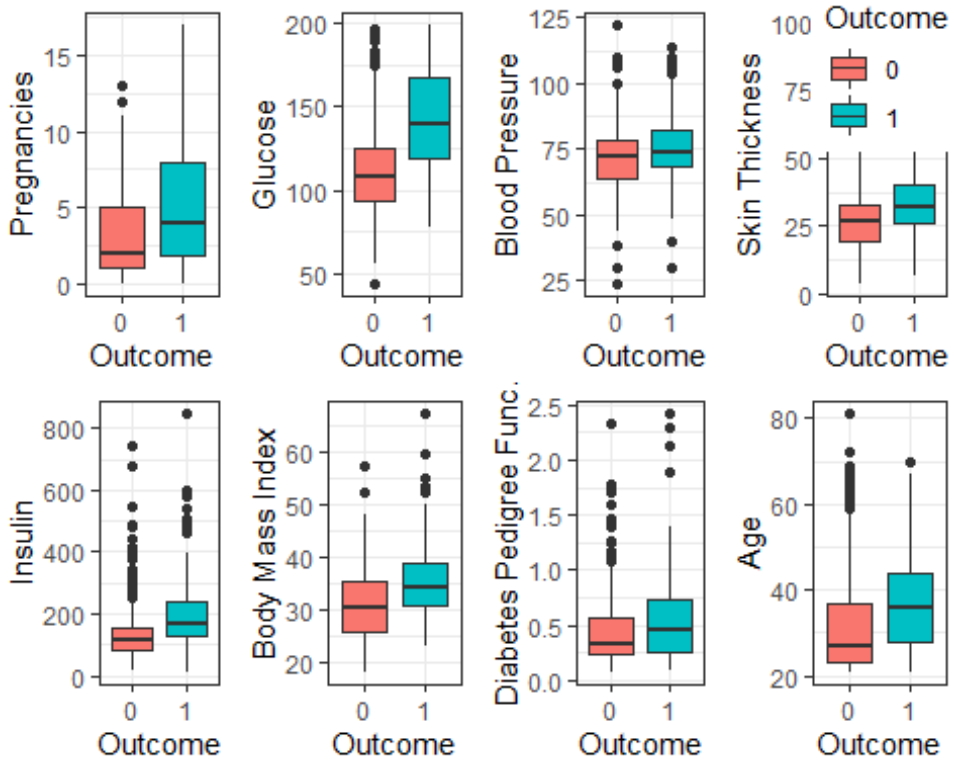
```
alpha = 0.5) + labs(y="Density", x="Diabetes Pedigree Func.") + theme_
bw() + theme(legend.position = "none")
den8 <- ggplot(diabetes, aes(x = Age, fill = Outcome)) + geom_density(
alpha = 0.5) + labs(y="Density", x="Age") + theme_bw() + theme(legend.
position = "none")
grid.arrange(den1, den2, den3, den4, den5, den6, den7, den8, ncol=4)
```




plotting boxplots for predictors

```
box1 <- ggplot(diabetes, aes(x = Outcome, y = Pregnancy))+ geom_boxplo
t(aes(fill = Outcome)) + theme_bw() + theme(legend.position = "none")
+ labs(y="Pregnancies")
box2 <- ggplot(diabetes, aes(x = Outcome, y = Glucose))+ geom_boxplot(
aes(fill = Outcome)) + theme_bw() + theme(legend.position = "none") +
labs(y="Glucose")
box3 <- ggplot(diabetes, aes(x = Outcome, y = BloodP))+ geom_boxplot(a
es(fill = Outcome)) + theme_bw() + theme(legend.position = "none") + l
abs(y="Blood Pressure")
box4 <- ggplot(diabetes, aes(x = Outcome, y = SkinT))+ geom_boxplot(ae
s(fill = Outcome)) + theme_bw() + theme(legend.position = c(0.5, 0.8))
 + labs(y="Skin Thickness")
box5 <- ggplot(diabetes, aes(x = Outcome, y = Insulin))+ geom_boxplot(
aes(fill = Outcome)) + theme_bw() + theme(legend.position = "none") +
labs(y="Insulin")
box6 <- ggplot(diabetes, aes(x = Outcome, y = BMI))+ geom_boxplot(aes(
```

```
fill = Outcome)) + theme_bw() + theme(legend.position = "none") + labs
(y="Body Mass Index")
box7 <- ggplot(diabetes, aes(x = Outcome, y = DPF))+ geom_boxplot(aes(
fill = Outcome)) + theme_bw() + theme(legend.position = "none") + labs
(y="Diabetes Pedigree Func.")
box8 <- ggplot(diabetes, aes(x = Outcome, y = Age))+ geom_boxplot(aes(
fill = Outcome)) + theme_bw() + theme(legend.position = "none") + labs
(y="Age")
grid.arrange(box1, box2, box3, box4, box5, box6, box7, box8, ncol=4)
```




It seems like output is mostly related with glucose levels. Other revelant predictors might be pregnancies, body mass index, insulin, skin thickness, and age, but their predictive power may be lower than glucose levels. It makes sense that glucose levels are the leading predictor for diabetes detection.

## Perform Model building and performance Evaluation

```
#Set split number
split<-10
#Define train and test data with Boost sample split
bst_sample<-bootstraps(diabetes,times=split)

#Initial variables
NIR<-rep(NA,split)
```

```
#Initialization for KNN
KNN_ROC<-list()
KNN_Accuracy_Rate<-rep(NA,split)
KNN_AUC<-rep(NA,split)
#Initialization for LDA
LDA_ROC<-list()
LDA_Accuracy_Rate<-rep(NA,split)
LDA_AUC<-rep(NA,split)
#Initialization for QDA
QDA_ROC<-list()
QDA_Accuracy_Rate<-rep(NA,split)
QDA_AUC<-rep(NA,split)
#Initialization for enhanced LDA
En_LDA_ROC<-list()
En_LDA_Accuracy_Rate<-rep(NA,split)
En_LDA_AUC<-rep(NA,split)
#Initialization for enhanced QDA
En_QDA_ROC<-list()
En_QDA_Accuracy_Rate<-rep(NA,split)
En_QDA_AUC<-rep(NA,split)
#Initialization for Logistic
Logistic_ROC<-list()
Logistic_Accuracy_Rate<-rep(NA,split)
Logistic_AUC<-rep(NA,split)
Back_Logistic_ROC<-list()
Back_Logistic_Accuracy_Rate<-rep(NA,split)
Back_Logistic_AUC<-rep(NA,split)
Best_Logistic_ROC<-list()
Best_Logistic_Accuracy_Rate<-rep(NA,split)
Best_Logistic_AUC<-rep(NA,split)
#Initialization for NaiveBayes
NB_ROC<-list()
NB_Accuracy_Rate<-rep(NA,split)
NB_AUC<-rep(NA,split)
#Initialization
K_NB_ROC<-list()
K_NB_Accuracy_Rate<-rep(NA,split)
K_NB_AUC<-rep(NA,split)
#Initialization for SVM
L_SVM_ROC<-list()
L_SVM_Accuracy_Rate<-rep(NA,split)
L_SVM_AUC<-rep(NA,split)
#Initialization
R_SVM_ROC<-list()
R_SVM_Accuracy_Rate<-rep(NA,split)
```

```
R_SVM_AUC<-rep(NA,split)
#Initialization for Classification
Classification_ROC<-list()
Classification_Accuracy_Rate<-rep(NA,split)
Classification_AUC<-rep(NA,split)
#Initialization for Boost Tree
Boost_ROC<-list()
Boost_Accuracy_Rate<-rep(NA,split)
Boost_AUC<-rep(NA,split)
#Initialization for RandomForest
RandomForest_ROC<-list()
RandomForest_Accuracy_Rate<-rep(NA,split)
RandomForest_AUC<-rep(NA,split)


for (i in 1:split){
  train<-training(bst_sample$splits[[i]])
  NIR[i] <- max(table(train$Outcome)/nrow(train))
}
```

## KNN model

```
#Perfrom KNN model building
for (i in 1:split){
  train<-training(bst_sample$splits[[i]])
  test<-testing(bst_sample$splits[[i]])
  ## K values for tuning
  kgrid <- expand.grid(k = seq(2,51))
  train$Outcome <- as.factor(train$Outcome)
  ## LOOCV tuning
  tr <- trainControl(method = "cv",
                     number = 5)

  ## Train k
  fit <- train(Outcome ~ .,
               data = train,
               method = "knn",
               tuneGrid = kgrid,
               trControl = tr)

  #Plot tune result
  plot(fit)

  #View tuned K
  fit$bestTune$k

  # Refit the model with best K
```

```
  tuned_knn_class <- train(Outcome ~ .,
                           data = train,
                           method = "knn",
                           tuneGrid = expand.grid(k = fit$bestTune$k),
                           trControl = trainControl(method = "none"))

  #Confusion matrix
  knn_pre<-predict(tuned_knn_class,newdata=test)
  knn_result<-confusionMatrix(data=as.factor(test$Outcome),reference=k
nn_pre)
  #Accuracy rate
  KNN_Accuracy_Rate[i]<-knn_result$overall[[1]]

  #Plot ROC and AUC
  knn_pre_ROC<-predict(tuned_knn_class,newdata=test,type = "prob")
  knn_roccurve <- roc(response = test$Outcome,
                      predictor = knn_pre_ROC[,2])

  KNN_ROC[[i]]<-ggroc(knn_roccurve, legacy.axes = TRUE, lwd=2) +  them
e_bw(base_size = 18) + ggtitle(paste("KNN ROC BST",i))
  KNN_AUC[i]<-auc(knn_roccurve)
}
```

## LDA model

```
#Perfrom LDA model building
#changing outcome to factor
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
train$Outcome <- as.factor(train$Outcome)


#LDA model
caret_lda <- train(Outcome ~ .,
                   data = train,
                   method = "lda",
                   trControl = trainControl(method = "repeatedcv", num
ber = 5,
                                            repeats = 50))

lda_predict <- predict(caret_lda, newdata = test, type = 'raw')
lda_result <- confusionMatrix(data=lda_predict, reference=test$Outcome
)


lda_predict_prob <- predict(caret_lda, newdata = test, type = 'prob')
```

```
LDA_Accuracy_Rate[i]<-lda_result$overall["Accuracy"]
roccurve_lda <- roc(response = test$Outcome, predictor = lda_predict_p
rob[,2], quiet = TRUE)
LDA_ROC[[i]]<-ggroc(roccurve_lda, legacy.axes = TRUE, lwd=1) + theme_b
w(base_size = 18) + ggtitle("ROC Curve for LDA") + theme(plot.title =
element_text(hjust = 0.5))
LDA_AUC[i]<-auc(roccurve_lda)
}
```

## QDA model

```
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
caret_qda <- train(Outcome ~ .,
                   data = train,
                   method = "qda",
                   trControl = trainControl(method = "repeatedcv", num
ber = 5,
                                            repeats = 50))

qda_predict <- predict(caret_qda, newdata = test, type = 'raw')
qda_result <- confusionMatrix(data=qda_predict, reference=test$Outcome
)


qda_predict_prob <- predict(caret_qda, newdata = test, type = 'prob')
QDA_Accuracy_Rate[i]<-qda_result$overall["Accuracy"]
roccurve_qda <- roc(response = test$Outcome, predictor = qda_predict_p
rob[,2], quiet = TRUE)
QDA_ROC[[i]]<-ggroc(roccurve_qda, legacy.axes = TRUE, lwd=1) + theme_b
w(base_size = 18) + ggtitle("ROC Curve for QDA") + theme(plot.title =
element_text(hjust = 0.5))
QDA_AUC[i]<-auc(roccurve_qda)
}
```

## LDA model enhanced (with only few predictors)

```
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
caret_lda <- train(Outcome ~Glucose+BMI+Pregnancy+Age,
                   data = train,
                   method = "lda",
                   trControl = trainControl(method = "repeatedcv", num
ber = 5,
                                            repeats = 50))
```

```
lda_predict <- predict(caret_lda, newdata = test, type = 'raw')
lda_result <- confusionMatrix(data=lda_predict, reference=test$Outcome
)

En_LDA_Accuracy_Rate[i]<-lda_result$overall["Accuracy"]

lda_predict_prob <- predict(caret_lda, newdata = test, type = 'prob')

roccurve_lda <- roc(response = test$Outcome, predictor = lda_predict_p
rob[,2], quiet = TRUE)
En_LDA_ROC[[i]]<-ggroc(roccurve_lda, legacy.axes = TRUE, lwd=1) + them
e_bw(base_size = 18) + ggtitle("ROC Curve for Enhanced LDA") + theme(p
lot.title = element_text(hjust = 0.5))
En_LDA_AUC[i]<-auc(roccurve_lda)
}
```

## QDA model enhanced (with only few predictors)

```
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
caret_qda <- train(Outcome ~Glucose+BMI+Pregnancy+Age,
                   data = train,
                   method = "qda",
                   trControl = trainControl(method = "repeatedcv", num
ber = 5,
                                            repeats = 50))

qda_predict <- predict(caret_qda, newdata = test, type = 'raw')
qda_result <- confusionMatrix(data=qda_predict, reference=test$Outcome
)
En_QDA_Accuracy_Rate[i]<-qda_result$overall["Accuracy"]


qda_predict_prob <- predict(caret_qda, newdata = test, type = 'prob')

roccurve_qda <- roc(response = test$Outcome, predictor = qda_predict_p
rob[,2], quiet = TRUE)
En_QDA_ROC[[i]]<-ggroc(roccurve_qda, legacy.axes = TRUE, lwd=1) + them
e_bw(base_size = 18) + ggtitle("ROC Curve for Enhanced QDA") + theme(p
lot.title = element_text(hjust = 0.5))
En_QDA_AUC[i]<-auc(roccurve_qda)
}
```

## Logistic regression and selections

```
################################################
#Logistic
```

```r
#logistical regression. No clear collinearity according to VIF.
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
glm<- glm(Outcome~.,train,family="binomial")
summary(glm)

# predicted probability of glm
glm_prob <- predict(glm, test,type = "response")
# predicted outcome of glm
glm_pred <- rep(0, length(glm_prob))
glm_pred[glm_prob > 0.5] <- 1
Logistic_Accuracy_Rate[i]<-mean(glm_pred == test$Outcome)

roccurve_glm <- roc(response = test$Outcome, predictor = glm_prob)
Logistic_ROC[[i]]<-ggroc(roccurve_glm, legacy.axes = TRUE, lwd=2) +the
me_bw(base_size = 18)
Logistic_AUC[i]<-auc(roccurve_glm)

################################################
back<-step(glm) #backward selection based on AIC 4 variables
back

# predicted probability of back
back_prob <- predict(back, test,type = "response")
# ROC and its auc
roccurve_back <- roc(response = test$Outcome, predictor = back_prob)
Back_Logistic_ROC[[i]]<-ggroc(roccurve_back, legacy.axes = TRUE, lwd=2
) +theme_bw(base_size = 18)
Back_Logistic_AUC[i]<-auc(roccurve_back)

# predicted outcome of glm for back
back_pred <- rep(0, length(glm_prob))
back_pred[back_prob > 0.5] <- 1
Back_Logistic_Accuracy_Rate[i]<-mean(back_pred == test$Outcome)

train_back<-as.data.frame(model.matrix(Outcome~.-1, data = train))


################################################
best<-bestglm(train_back, IC="AIC")
best$BestModel
# predicted probability of glm for best
best_prob <- predict(best$BestModel, test,type = "response")
# ROC and its auc
roccurve_best <- roc(response = test$Outcome, predictor = best_prob)
```

```
Best_Logistic_ROC[[i]]<-ggroc(roccurve_best, legacy.axes = TRUE, lwd=2
) +theme_bw(base_size = 18)
Best_Logistic_AUC[i]<-auc(roccurve_best)

# predicted outcome of glm
best_pred <- rep(0, length(glm_prob))
best_pred[glm_prob > 0.5] <- 1
Best_Logistic_Accuracy_Rate[i]<-mean(best_pred == test$Outcome)
}

## Step:  AIC=679.23
## Outcome ~ Pregnancy + Glucose + Insulin + BMI + DPF
##
##             Df Deviance    AIC
## <none>           667.23 679.23
## - Insulin    1   677.82 687.82
## - DPF        1   681.55 691.55
## - Pregnancy  1   703.66 713.66
## - BMI        1   712.29 722.29
## - Glucose    1   791.55 801.55
##
## Step:  AIC=693.69
## Outcome ~ Pregnancy + Glucose + BloodP + BMI + DPF + Age
##
##             Df Deviance    AIC
## <none>           679.69 693.69
## - DPF        1   682.26 694.26
## - Pregnancy  1   685.13 697.13
## - BloodP     1   686.26 698.26
## - Age        1   699.35 711.35
## - BMI        1   726.88 738.88
## - Glucose    1   828.98 840.98
## Start:  AIC=746.19
## Outcome ~ Pregnancy + Glucose + BloodP + SkinT + Insulin + BMI +
##     DPF + Age
##
##             Df Deviance    AIC
## - Age        1   728.27 744.27
## - BloodP     1   728.28 744.28
## - DPF        1   729.56 745.56
## <none>           728.19 746.19
## - Insulin    1   730.51 746.51
## - SkinT      1   733.96 749.96
## - BMI        1   734.01 750.01
## - Pregnancy  1   768.68 784.68
## - Glucose    1   826.17 842.17
```

```
##
##
## Step:  AIC=762.98
## Outcome ~ Pregnancy + Glucose + SkinT + BMI + DPF + Age
##
##             Df Deviance    AIC
## <none>           748.98 762.98
## - SkinT      1   751.92 763.92
## - Age        1   754.16 766.16
## - Pregnancy  1   755.23 767.23
## - DPF        1   757.94 769.94
## - BMI        1   766.39 778.39
## - Glucose    1   861.01 873.01
## Start:  AIC=709.03
## Outcome ~ Pregnancy + Glucose + BloodP + SkinT + Insulin + BMI +
##     DPF + Age
##
##             Df Deviance    AIC
## <none>           691.03 709.03
## - SkinT      1   693.33 709.33
## - Insulin    1   693.76 709.76
## - Age        1   694.89 710.89
## - BloodP     1   695.02 711.02
## - Pregnancy  1   706.17 722.17
## - DPF        1   706.49 722.49
## - BMI        1   712.09 728.09
## - Glucose    1   781.51 797.51
```

## NaiveBayes

```
#Perfrom NaiveBayes W/O Kernel
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
nb <- NaiveBayes(as.factor(Outcome) ~ .,
                     data = train,
                     usekernel = F)

#Confusion matrix
ns_pre<-predict(nb,newdata=test,type = "response")
ns_result<-confusionMatrix(data=as.factor(test$Outcome),reference=ns_p
re$class)
#Accuracy rate
NB_Accuracy_Rate[i]<-ns_result$overall["Accuracy"]

ns_pre_ROC<-predict(nb,newdata=test,type = "prob")
ns_roccurve <- roc(response = test$Outcome,
```

```
                    predictor = ns_pre_ROC$posterior[,2])
NB_ROC[[i]]<-ggroc(ns_roccurve, legacy.axes = TRUE, lwd=2) +  theme_bw
(base_size = 18) + ggtitle(paste("NaiveBayes w/o Kernel ROC BST",i))

NB_AUC[i]<-auc(ns_roccurve)
}

#Perfrom NaiveBayes With Kernel
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
nb_K <- NaiveBayes(as.factor(Outcome) ~ .,
                   data = train,
                   usekernel = T)

#Confusion matrix
ns_k_pre<-predict(nb_K,newdata=test,type = "response")
ns_k_result<-confusionMatrix(data=as.factor(test$Outcome),reference=ns
_k_pre$class)
#Accuracy rate
K_NB_Accuracy_Rate[i]<-ns_k_result$overall["Accuracy"]


ns_k_pre_ROC<-predict(nb_K,newdata=test,type = "prob")
ns_k_roccurve <- roc(response = test$Outcome,
                     predictor = ns_k_pre_ROC$posterior[,2])
K_NB_ROC[[i]]<-ggroc(ns_k_roccurve, legacy.axes = TRUE, lwd=2) +  them
e_bw(base_size = 18) + ggtitle(paste("NaiveBayes w/ Kernel ROC BST",i)
)

K_NB_AUC[i]<-auc(ns_k_roccurve)
}
```

## SVM Linear Model

```
#SVM Linear Model
# Tuning grid
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
tune_grid <- expand.grid(cost = exp(seq(-7,3,len=11)))
# Train the model
sv_caret <- train(as.factor(Outcome) ~ .,
                  data = train,
                  method = "svmLinear2",
                  tuneGrid = tune_grid,
                  trControl = tr)
```

```
#Plot SVM
plot(sv_caret)

#View tune cost
sv_caret$bestTune

# Final model
sv_final <- train(as.factor(Outcome) ~ .,
                          data = train,
                          method = "svmLinear2",
                          tuneGrid = expand.grid(cost = sv_caret$bestTu
ne),
                          trControl = trainControl(method = "none"))

#Confusion matrix
svm_pre<-predict(sv_final,newdata=test)
sv_result<-confusionMatrix(data=as.factor(test$Outcome),reference=svm_
pre)
#Accuracy rate
L_SVM_Accuracy_Rate[i]<-sv_result$overall["Accuracy"]


sv_roccurve <- roc(response = test$Outcome,predictor=as.numeric(svm_pr
e))
L_SVM_ROC[[i]]<-ggroc(sv_roccurve, legacy.axes = TRUE, lwd=2) +  theme
_bw(base_size = 18) + ggtitle(paste("SVM Linear Kernel ROC BST",i))

L_SVM_AUC[i]<-auc(sv_roccurve)
}
```

## Radial

```
# Train the model with Radial
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
sv_caret_radial <- train(as.factor(Outcome) ~ .,
                          data = train,
                          method = "svmRadial",
                          preProcess = c("center", "scale"),
                          tuneGrid = expand.grid(C = exp(seq(-7,3,len=1
1)),sigma = 10^(seq(-3,3,len=7))),
                          trControl = tr)
```

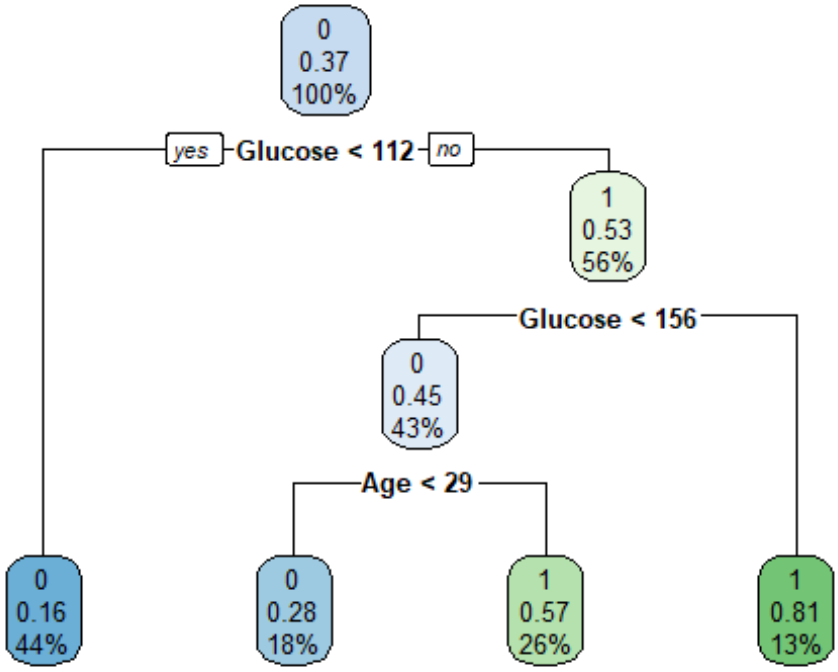
```
#Plot SVM
plot(sv_caret_radial)

#View tune cost
sv_caret_radial$bestTune

# Final model
sv_final_radial <- train(as.factor(Outcome) ~ .,
                  data = train,
                  method = "svmRadial",
                  preProcess = c("center", "scale"),
                  tuneGrid = expand.grid(sv_caret_radial$bestTune),
                  trControl = trainControl(method = "none"))

#Confusion matrix
svm_pre_radial<-predict(sv_final_radial,newdata=test)
sv_result_radial<-confusionMatrix(data=as.factor(test$Outcome),referen
ce=svm_pre_radial)
#Accuracy rate
R_SVM_Accuracy_Rate[i]<-sv_result_radial$overall["Accuracy"]


sv_roccurve_radial <- roc(response = test$Outcome,predictor=as.numeric
(svm_pre_radial))
R_SVM_ROC[[i]]<-ggroc(sv_roccurve_radial, legacy.axes = TRUE, lwd=2) +
  theme_bw(base_size = 18)+ ggtitle(paste("SVM Radial Kernel ROC BST",
i))

R_SVM_AUC[i]<-auc(sv_roccurve_radial)
}
```

## Classification tree

```
#classification tree
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
tree_class <- rpart(
  Outcome ~ .,
  data = train,
  method = 'class',
  parms = list(split = "information"),
  control = rpart.control(
    xval = 10,
    minbucket = 2,
    cp = 0
  )
```

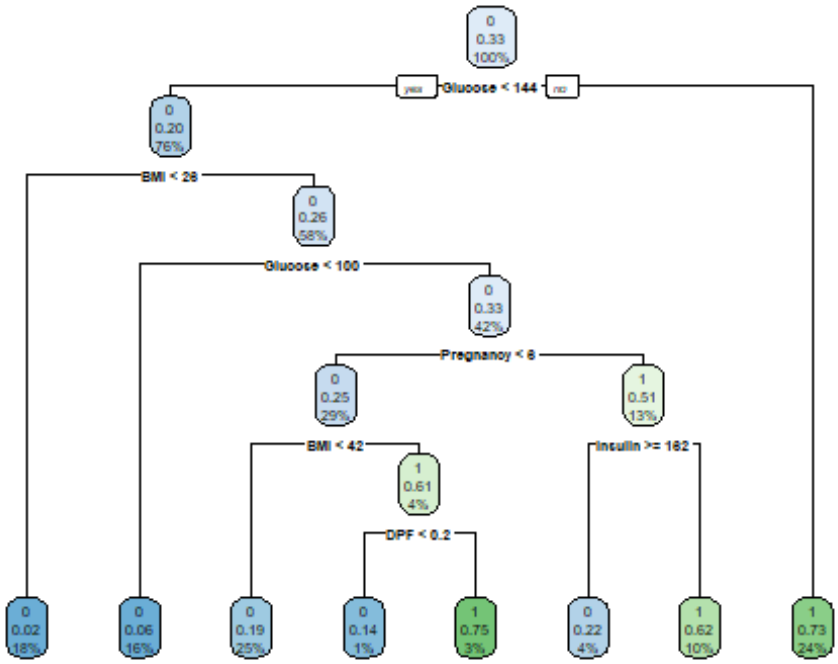
```
)
printcp(tree_class)
cp <- tree_class$cptable
tree_class_final <- prune(tree_class, cp = cp[3,1])#used minimum
rpart.plot(tree_class_final)

# test error rate using min cp
tree_pred <- predict(tree_class_final, newdata=test, type = "class")

Classification_Accuracy_Rate[i]<-mean(tree_pred == test$Outcome)

tree_prob <- predict(tree_class_final, newdata=test, type = "prob")

roccurve_tree <- roc(response = test$Outcome, predictor = tree_prob[,2
])
Classification_ROC[[i]]<-ggroc(roccurve_tree, legacy.axes = TRUE, lwd=
2) +theme_bw(base_size = 18)
Classification_AUC[i]<-auc_tree<-auc(roccurve_tree)

klaR::errormatrix(true = test$Outcome, predicted = tree_pred, relative
 = TRUE)
confusionMatrix(tree_pred,test$Outcome)
}

##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
gnancy
## [8] SkinT
##
## Root node error: 282/768 = 0.36719
##
## n= 768
##
##           CP nsplit rel error  xerror     xstd
## 1  0.1081560      0  1.000000 1.00000 0.047371
## 2  0.0921986      2  0.783688 0.86879 0.045804
## 3  0.0425532      3  0.691489 0.76950 0.044246
## 4  0.0319149      4  0.648936 0.71986 0.043335
## 5  0.0212766      5  0.617021 0.69858 0.042916
```

```
## 6  0.0177305      8  0.553191 0.70922 0.043128
## 7  0.0141844     10  0.517730 0.69858 0.042916
## 8  0.0106383     12  0.489362 0.68085 0.042553
## 9  0.0094563     25  0.326241 0.62411 0.041304
## 10 0.0088652     28  0.297872 0.61348 0.041054
## 11 0.0070922     30  0.280142 0.60993 0.040969
## 12 0.0053191     38  0.223404 0.55674 0.039632
## 13 0.0047281     41  0.205674 0.49291 0.037836
## 14 0.0035461     45  0.184397 0.48582 0.037622
## 15 0.0017730     65  0.078014 0.43617 0.036041
## 16 0.0000000     69  0.070922 0.44326 0.036277
```



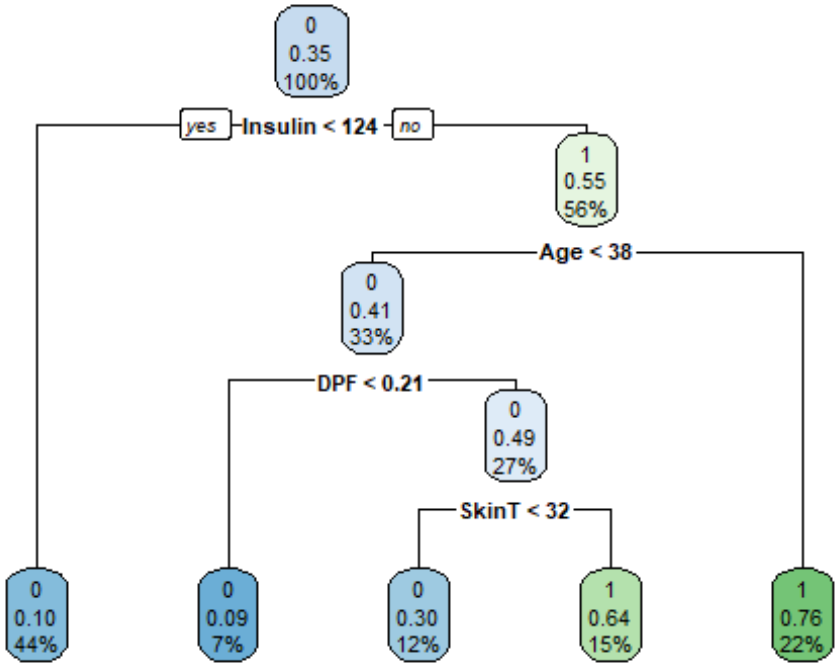

```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
gnancy
## [8] SkinT
##
```

```
## Root node error: 254/768 = 0.33073
##
## n= 768
##
##          CP nsplit rel error  xerror     xstd
## 1 0.3346457      0  1.000000 1.00000 0.051332
## 2 0.0177165      1  0.665354 0.67323 0.045391
## 3 0.0157480      7  0.547244 0.61024 0.043791
## 4 0.0144357     15  0.393701 0.58661 0.043144
## 5 0.0078740     18  0.350394 0.56693 0.042585
## 6 0.0059055     28  0.267717 0.48425 0.040015
## 7 0.0039370     51  0.102362 0.46850 0.039480
## 8 0.0019685     59  0.070866 0.42913 0.038075
## 9 0.0000000     61  0.066929 0.42520 0.037929
```

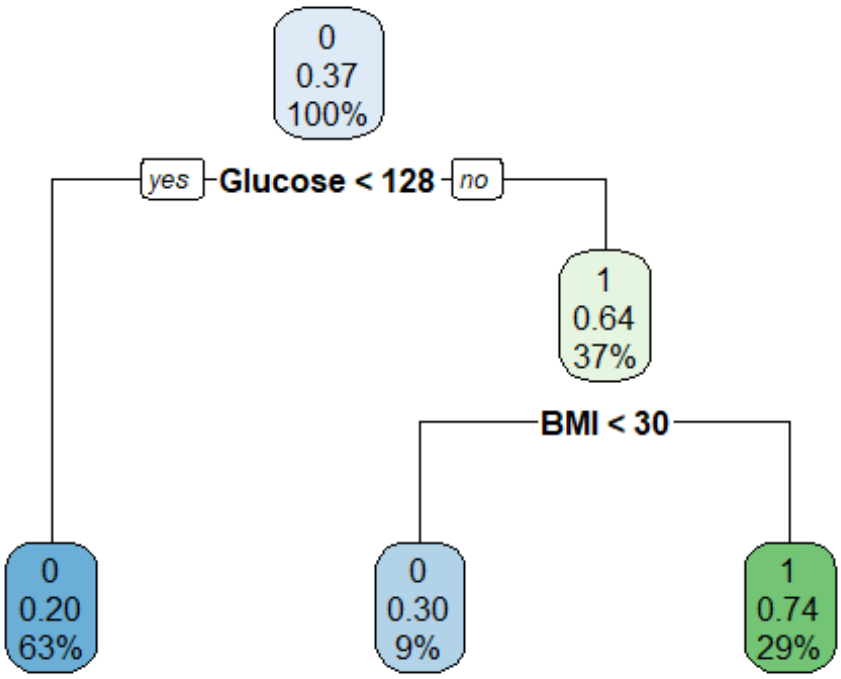



```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
```

```
gnancy
## [8] SkinT
##
## Root node error: 271/768 = 0.35286
##
## n= 768
##
##           CP nsplit rel error  xerror     xstd
## 1  0.1660517      0  1.000000 1.00000 0.048867
## 2  0.0590406      2  0.667897 0.76015 0.045306
## 3  0.0295203      4  0.549815 0.67897 0.043648
## 4  0.0239852      5  0.520295 0.60517 0.041907
## 5  0.0221402      7  0.472325 0.59410 0.041625
## 6  0.0147601      8  0.450185 0.57934 0.041240
## 7  0.0135301      9  0.435424 0.53875 0.040126
## 8  0.0110701     18  0.309963 0.48708 0.038580
## 9  0.0086101     22  0.265683 0.46494 0.037871
## 10 0.0073801     26  0.225092 0.41697 0.036225
## 11 0.0055351     32  0.180812 0.42066 0.036357
## 12 0.0046125     38  0.147601 0.42066 0.036357
## 13 0.0036900     42  0.129151 0.41697 0.036225
## 14 0.0018450     51  0.095941 0.39114 0.035272
## 15 0.0000000     55  0.088561 0.38745 0.035132
```

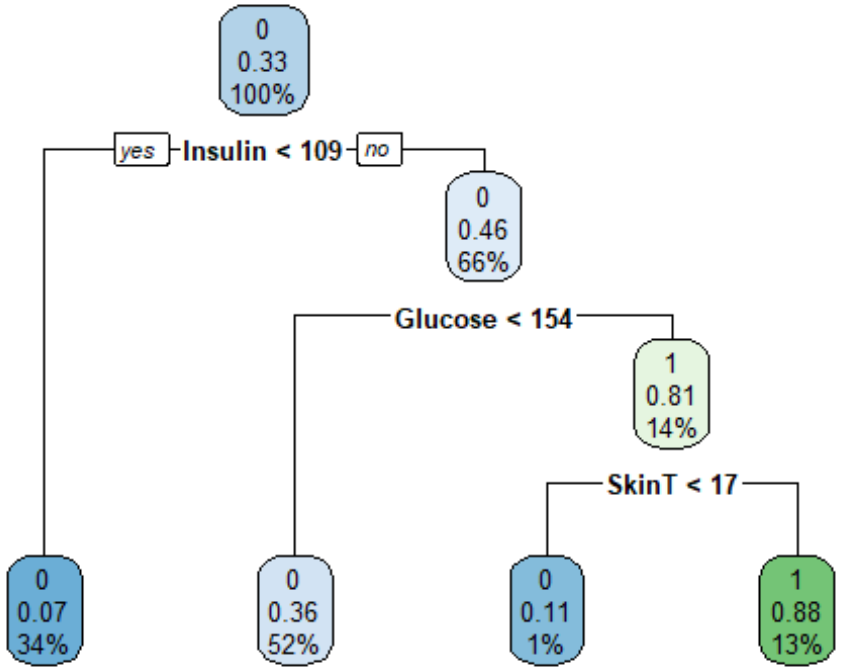


```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
gnancy
## [8] SkinT
##
## Root node error: 281/768 = 0.36589
##
## n= 768
##
##           CP nsplit rel error  xerror     xstd
## 1  0.2846975      0  1.000000 1.00000 0.047504
## 2  0.0960854      1  0.715302 0.71530 0.043351
## 3  0.0308422      2  0.619217 0.64057 0.041777
## 4  0.0213523      5  0.526690 0.64769 0.041937
## 5  0.0142349      8  0.451957 0.58719 0.040505
## 6  0.0133452      9  0.437722 0.59431 0.040683
## 7  0.0124555     14  0.366548 0.59431 0.040683
## 8  0.0118624     16  0.341637 0.58007 0.040326
## 9  0.0106762     19  0.306050 0.54804 0.039487
## 10 0.0088968     22  0.274021 0.53381 0.039098
## 11 0.0071174     29  0.209964 0.50890 0.038390
## 12 0.0053381     32  0.188612 0.46263 0.036982
## 13 0.0035587     39  0.149466 0.46619 0.037095
## 14 0.0026690     52  0.103203 0.45196 0.036639
## 15 0.0017794     56  0.092527 0.44128 0.036288
## 16 0.0000000     58  0.088968 0.44484 0.036406
```



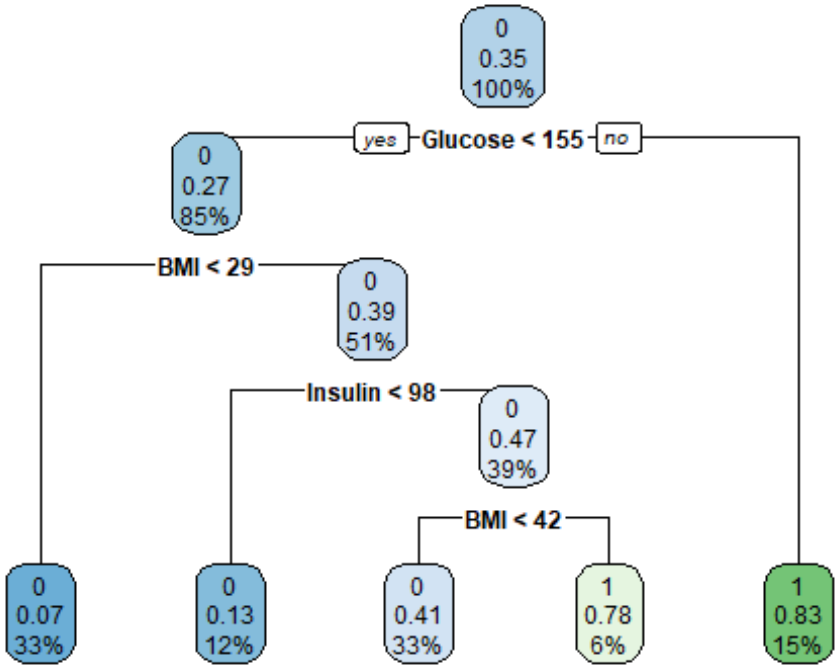

```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
gnancy
## [8] SkinT
##
## Root node error: 251/768 = 0.32682
##
## n= 768
##
##           CP nsplit rel error  xerror     xstd
## 1  0.1334661      0  1.000000 1.00000 0.051788
## 2  0.0278884      2  0.733068 0.75299 0.047557
## 3  0.0258964      3  0.705179 0.74104 0.047300
## 4  0.0239044      7  0.577689 0.74900 0.047472
## 5  0.0199203     10  0.505976 0.69721 0.046312
## 6  0.0139442     12  0.466135 0.63347 0.044736
## 7  0.0132802     14  0.438247 0.58964 0.043548
```

```
## 8  0.0119522     19  0.358566 0.56574 0.042862
## 9  0.0099602     24  0.298805 0.53785 0.042026
## 10 0.0092961     28  0.258964 0.52191 0.041529
## 11 0.0079681     31  0.231076 0.49801 0.040757
## 12 0.0059761     46  0.111554 0.42629 0.038233
## 13 0.0039841     51  0.079681 0.39044 0.036838
## 14 0.0019920     52  0.075697 0.32669 0.034097
## 15 0.0000000     56  0.067729 0.31474 0.033540
```



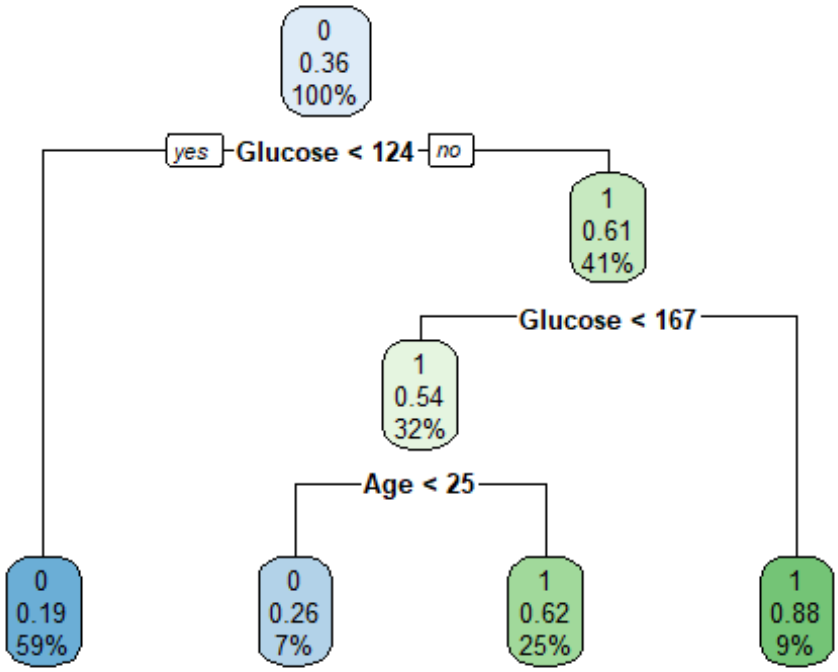

```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
gnancy
## [8] SkinT
##
## Root node error: 270/768 = 0.35156
##
## n= 768
```

```
##
##           CP nsplit rel error  xerror     xstd
## 1  0.2851852      0  1.000000 1.00000 0.049006
## 2  0.0320988      1  0.714815 0.75185 0.045261
## 3  0.0259259      4  0.618519 0.67778 0.043728
## 4  0.0240741      5  0.592593 0.66296 0.043395
## 5  0.0185185     11  0.396296 0.64074 0.042878
## 6  0.0148148     12  0.377778 0.61481 0.042248
## 7  0.0111111     14  0.348148 0.55185 0.040588
## 8  0.0074074     17  0.314815 0.50741 0.039294
## 9  0.0055556     28  0.229630 0.45556 0.037643
## 10 0.0037037     39  0.155556 0.40000 0.035681
## 11 0.0024691     55  0.092593 0.35185 0.033793
## 12 0.0000000     63  0.070370 0.35185 0.033793
```



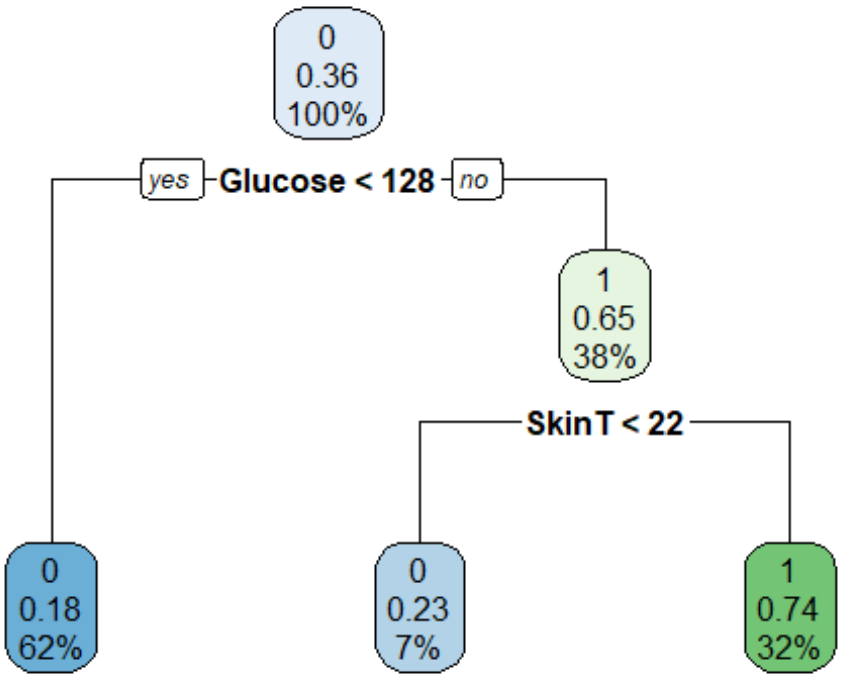

```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
```

```
gnancy
## [8] SkinT
##
## Root node error: 278/768 = 0.36198
##
## n= 768
##
##           CP nsplit rel error  xerror     xstd
## 1  0.2589928      0  1.000000 1.00000 0.047907
## 2  0.0467626      1  0.741007 0.76259 0.044564
## 3  0.0239808      3  0.647482 0.69784 0.043314
## 4  0.0215827      6  0.575540 0.69784 0.043314
## 5  0.0131894      7  0.553957 0.65827 0.042470
## 6  0.0125899     15  0.402878 0.59712 0.041032
## 7  0.0089928     22  0.294964 0.52878 0.039217
## 8  0.0071942     24  0.276978 0.52158 0.039012
## 9  0.0053957     44  0.122302 0.45324 0.036917
## 10 0.0035971     46  0.111511 0.42086 0.035823
## 11 0.0020555     55  0.079137 0.44245 0.036560
## 12 0.0017986     62  0.064748 0.44604 0.036680
## 13 0.0000000     64  0.061151 0.44604 0.036680
```



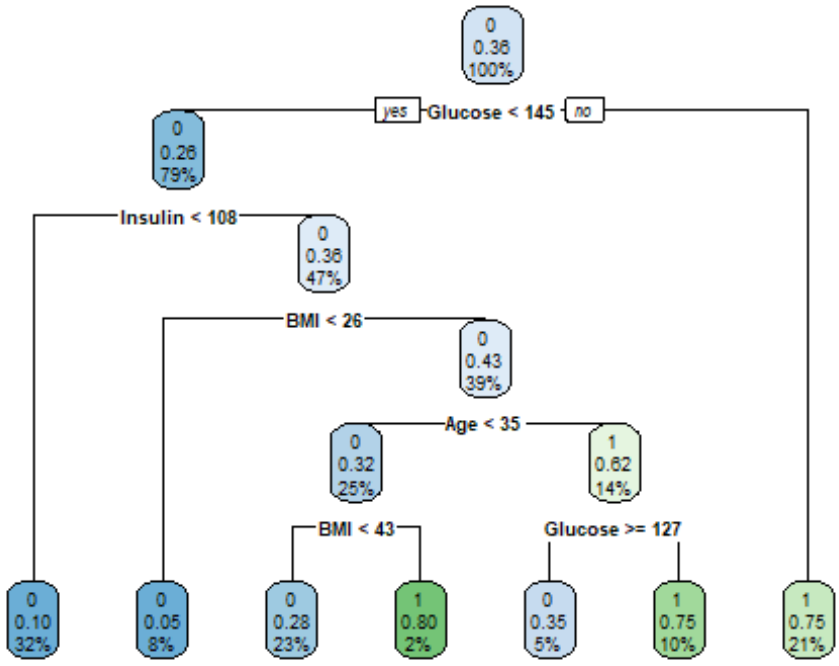

```
##
## Classification tree:
```

```
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
gnancy
## [8] SkinT
##
## Root node error: 275/768 = 0.35807
##
## n= 768
##
##           CP nsplit rel error  xerror     xstd
## 1  0.3236364      0  1.000000 1.00000 0.048314
## 2  0.1018182      1  0.676364 0.67636 0.043172
## 3  0.0230303      2  0.574545 0.60000 0.041389
## 4  0.0145455      5  0.505455 0.59273 0.041206
## 5  0.0109091     13  0.367273 0.53818 0.039748
## 6  0.0090909     17  0.323636 0.53818 0.039748
## 7  0.0072727     22  0.276364 0.52727 0.039438
## 8  0.0054545     35  0.181818 0.49455 0.038469
## 9  0.0036364     43  0.134545 0.46545 0.037556
## 10 0.0018182     56  0.083636 0.38909 0.034896
## 11 0.0000000     58  0.080000 0.39636 0.035168
```



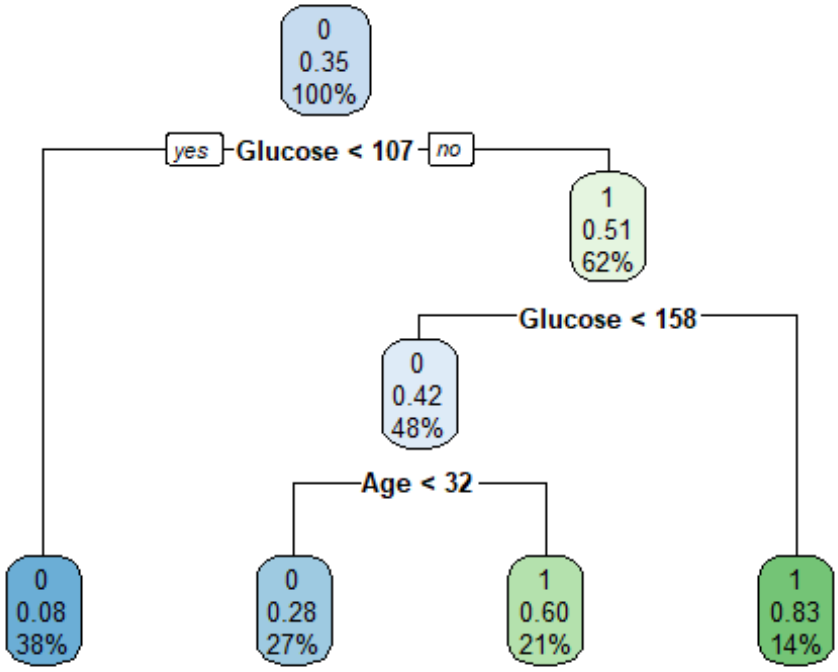

```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
gnancy
## [8] SkinT
##
## Root node error: 275/768 = 0.35807
##
## n= 768
##
##           CP nsplit rel error  xerror     xstd
## 1  0.2836364      0  1.000000 1.00000 0.048314
## 2  0.0315152      1  0.716364 0.73455 0.044368
## 3  0.0218182      6  0.549091 0.66182 0.042852
## 4  0.0127273      7  0.527273 0.62545 0.042012
## 5  0.0121212      9  0.501818 0.56727 0.040544
## 6  0.0109091     13  0.450909 0.56727 0.040544
## 7  0.0090909     24  0.330909 0.53455 0.039645
```

```
## 8  0.0087273     26  0.312727 0.52727 0.039438
## 9  0.0072727     33  0.247273 0.50182 0.038690
## 10 0.0063636     46  0.149091 0.48727 0.038246
## 11 0.0054545     50  0.123636 0.44364 0.036837
## 12 0.0036364     52  0.112727 0.41818 0.035958
## 13 0.0000000     63  0.072727 0.38909 0.034896
```




```
##
## Classification tree:
## rpart(formula = Outcome ~ ., data = train, method = "class",
##     parms = list(split = "information"), control = rpart.control(xv
al = 10,
##         minbucket = 2, cp = 0))
##
## Variables actually used in tree construction:
## [1] Age       BloodP    BMI       DPF       Glucose   Insulin   Pre
gnancy
## [8] SkinT
##
## Root node error: 268/768 = 0.34896
##
## n= 768
##
##           CP nsplit rel error  xerror     xstd
```

```
## 1  0.1361940      0  1.000000 1.00000 0.049288
## 2  0.1194030      2  0.727612 0.79104 0.046226
## 3  0.0373134      3  0.608209 0.70896 0.044620
## 4  0.0335821      4  0.570896 0.64925 0.043286
## 5  0.0205224      5  0.537313 0.59328 0.041898
## 6  0.0186567      8  0.473881 0.58582 0.041702
## 7  0.0161692      9  0.455224 0.57836 0.041503
## 8  0.0149254     12  0.406716 0.56716 0.041200
## 9  0.0111940     14  0.376866 0.54104 0.040468
## 10 0.0102612     18  0.332090 0.52239 0.039923
## 11 0.0093284     22  0.291045 0.48881 0.038895
## 12 0.0074627     30  0.216418 0.49254 0.039012
## 13 0.0037313     39  0.149254 0.45896 0.037924
## 14 0.0018657     55  0.082090 0.42537 0.036764
## 15 0.0000000     59  0.074627 0.42910 0.036897
```




## Boost tree

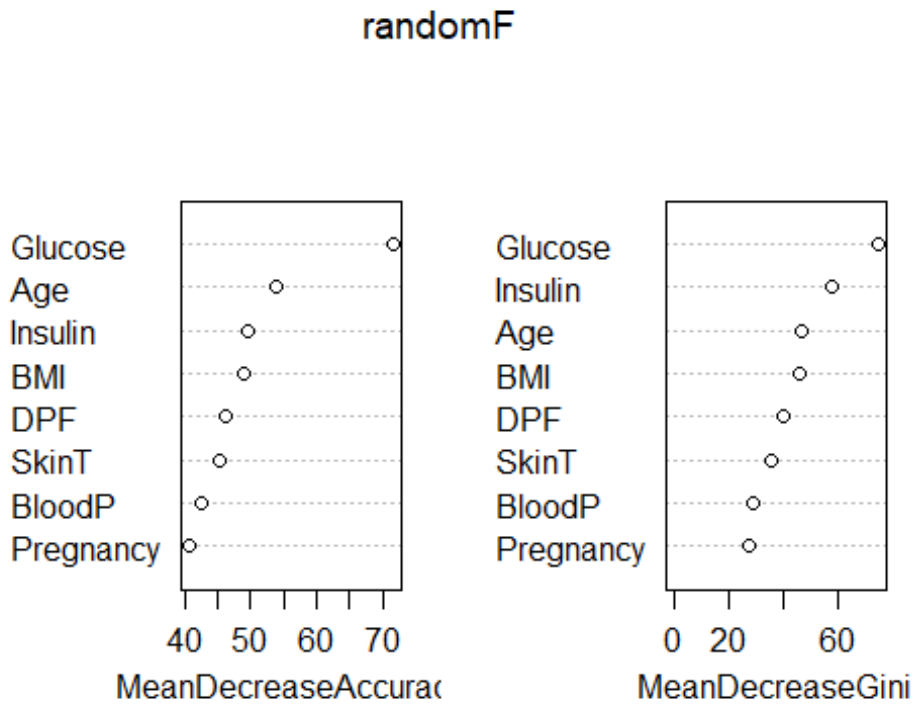
```
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
cvcontrol <- trainControl(method = "repeatedcv",
                          number = 5,
                          allowParallel = TRUE)
grid <- expand.grid(
```

```
  n.trees = c(10, 50, 100, 500, 1000),
  interaction.depth = c(1:3),
  shrinkage = c(0.01, 0.05, 0.1),
  n.minobsinnode = c(5,10)
)
capture <- capture.output(
  train.gbm <- train(
    Outcome ~ .,
    data = train,
    method = "gbm",
    trControl = cvcontrol,
    tuneGrid = grid
  )
)
train.gbm

boost_pred <- predict(train.gbm, newdata=test, type = "raw")
Boost_Accuracy_Rate[i]<-mean(boost_pred == test$Outcome)


boost_prob <- predict(train.gbm, newdata=test, type = "prob")

roccurve_tree <- roc(response = test$Outcome, predictor = boost_prob[,
2])
Boost_ROC[[i]]<-ggroc(roccurve_tree, legacy.axes = TRUE, lwd=2) +theme
_bw(base_size = 18)
Boost_AUC[i]<-auc(roccurve_tree)
}
```

## RandomForest tree

```
for (i in 1:split){
train<-training(bst_sample$splits[[i]])
test<-testing(bst_sample$splits[[i]])
randomF<-randomForest(Outcome~., data=train, mtry=3, importance=TRUE)
randomF

randomF_pred <- predict(randomF, newdata=test, type = "response")
RandomForest_Accuracy_Rate[i]<-mean(randomF_pred == test$Outcome)


randomF_prob <- predict(randomF, newdata=test, type = "prob")

roccurve_randomF <- roc(response = test$Outcome, predictor = randomF_p
rob[,2])
RandomForest_ROC[[i]]<-ggroc(roccurve_randomF, legacy.axes = TRUE, lwd
=2) +theme_bw(base_size = 18)
```

```
RandomForest_AUC[i]<-auc(roccurve_randomF)

varImpPlot(randomF)#Glucose is the most important one.
}
```




## Comparison Model Performance

```
#Report matrix
report <- rbind(NIR, KNN_Accuracy_Rate, LDA_Accuracy_Rate, QDA_Accurac
y_Rate, En_LDA_Accuracy_Rate, En_QDA_Accuracy_Rate, Logistic_Accuracy_
Rate,
                Back_Logistic_Accuracy_Rate, Best_Logistic_Accuracy_Ra
te, NB_Accuracy_Rate, K_NB_Accuracy_Rate,
                L_SVM_Accuracy_Rate, R_SVM_Accuracy_Rate, Classificati
on_Accuracy_Rate, Boost_Accuracy_Rate,
                RandomForest_Accuracy_Rate,
                KNN_AUC, LDA_AUC, QDA_AUC, En_LDA_AUC, En_QDA_AUC, Log
istic_AUC,  Back_Logistic_AUC, Best_Logistic_AUC, NB_AUC, K_NB_AUC,
                L_SVM_AUC,  R_SVM_AUC, Classification_AUC,  Boost_AUC,
 RandomForest_AUC)


report <- cbind(report, rowMeans(report))

col_name<-rep(NA,split+1)
```

```r
col_name[split+1]<-"AVE"
for (i in 1:split){
  col_name[i]<- paste("Splt#",i)
}
colnames(report)<-col_name
report
```

```
##                                 Splt# 1   Splt# 2   Splt# 3   Splt#
4   Splt# 5
## NIR                           0.6328125 0.6692708 0.6471354 0.634114
6 0.6731771
## KNN_Accuracy_Rate             0.6920415 0.6561404 0.6815068 0.647272
7 0.6928328
## LDA_Accuracy_Rate             0.8200692 0.7473684 0.7294521 0.763636
4 0.7508532
## QDA_Accuracy_Rate             0.7889273 0.7438596 0.7260274 0.734545
5 0.7474403
## En_LDA_Accuracy_Rate          0.8131488 0.7438596 0.7328767 0.770909
1 0.7815700
## En_QDA_Accuracy_Rate          0.8027682 0.7543860 0.7123288 0.767272
7 0.7576792
## Logistic_Accuracy_Rate        0.8166090 0.7473684 0.7328767 0.770909
1 0.7542662
## Back_Logistic_Accuracy_Rate   0.8166090 0.7508772 0.7260274 0.763636
4 0.7508532
## Best_Logistic_Accuracy_Rate   0.8166090 0.7473684 0.7328767 0.770909
1 0.7542662
## NB_Accuracy_Rate              0.7820069 0.7543860 0.7431507 0.789090
9 0.7645051
## K_NB_Accuracy_Rate            0.7785467 0.7649123 0.7397260 0.785454
5 0.8088737
## L_SVM_Accuracy_Rate           0.8096886 0.7438596 0.7397260 0.760000
0 0.7474403
## R_SVM_Accuracy_Rate           0.7058824 0.6842105 0.7123288 0.712727
3 0.6177474
## Classification_Accuracy_Rate 0.7474048 0.7228070 0.7020548 0.785454
5 0.7167235
## Boost_Accuracy_Rate           0.7923875 0.7578947 0.7500000 0.760000
0 0.7508532
## RandomForest_Accuracy_Rate    0.8062284 0.7473684 0.7534247 0.770909
1 0.7713311
## KNN_AUC                       0.7164801 0.6525517 0.7373387 0.645023
2 0.7061711
## LDA_AUC                       0.8763441 0.8232700 0.8115702 0.836451
5 0.8523579
## QDA_AUC                       0.8407944 0.8164969 0.7854440 0.820127
```

```
2 0.8308011
## En_LDA_AUC                   0.8724490 0.8242676 0.8111555 0.830459
8 0.8497928
## En_QDA_AUC                   0.8668532 0.8195422 0.7904204 0.820310
6 0.8390884
## Logistic_AUC                 0.8770024 0.8200672 0.8121404 0.838958
2 0.8501875
## Back_Logistic_AUC            0.8745337 0.8196472 0.8107926 0.831499
1 0.8500888
## Best_Logistic_AUC            0.7387536 0.6726347 0.7082577 0.671863
5 0.7068863
## NB_AUC                       0.8686087 0.8295705 0.8044166 0.845500
1 0.8452545
## K_NB_AUC                     0.8654817 0.8587105 0.8099114 0.860479
3 0.8708070
## L_SVM_AUC                    0.7608075 0.6887010 0.7007413 0.685528
2 0.7206985
## R_SVM_AUC                    0.6362464 0.6129896 0.6518065 0.623165
8 0.5000000
## Classification_AUC           0.7874698 0.7899821 0.6612669 0.732789
2 0.7403068
## Boost_AUC                    0.8306452 0.8410690 0.8097558 0.818048
4 0.8256215
## RandomForest_AUC             0.8644942 0.8446918 0.8078379 0.841892
9 0.8453532
##                                Splt# 6   Splt# 7   Splt# 8   Splt#
9  Splt# 10
## NIR                          0.6484375 0.6380208 0.6419271 0.641927
1 0.6510417
## KNN_Accuracy_Rate            0.6912281 0.6595745 0.7137809 0.720689
7 0.7163121
## LDA_Accuracy_Rate            0.7508772 0.7801418 0.7597173 0.775862
1 0.7765957
## QDA_Accuracy_Rate            0.7508772 0.7659574 0.7526502 0.737931
0 0.7163121
## En_LDA_Accuracy_Rate         0.7508772 0.7588652 0.7561837 0.768965
5 0.7765957
## En_QDA_Accuracy_Rate         0.7473684 0.7730496 0.7314488 0.782758
6 0.7375887
## Logistic_Accuracy_Rate       0.7578947 0.7872340 0.7597173 0.786206
9 0.7872340
## Back_Logistic_Accuracy_Rate  0.7543860 0.7765957 0.7667845 0.782758
6 0.7872340
## Best_Logistic_Accuracy_Rate  0.7578947 0.7872340 0.7597173 0.786206
9 0.7872340
```

```
## NB_Accuracy_Rate             0.7298246 0.7659574 0.7597173 0.768965
5 0.7588652
## K_NB_Accuracy_Rate           0.7578947 0.7765957 0.7526502 0.765517
2 0.7375887
## L_SVM_Accuracy_Rate          0.7578947 0.7659574 0.7526502 0.782758
6 0.7836879
## R_SVM_Accuracy_Rate          0.6771930 0.7127660 0.7314488 0.710344
8 0.7269504
## Classification_Accuracy_Rate 0.7438596 0.7411348 0.7279152 0.748275
9 0.7127660
## Boost_Accuracy_Rate          0.7649123 0.7836879 0.7526502 0.741379
3 0.7234043
## RandomForest_Accuracy_Rate   0.7543860 0.8049645 0.7632509 0.768965
5 0.7553191
## KNN_AUC                      0.7395946 0.6625224 0.6927844 0.815848
5 0.7632767
## LDA_AUC                      0.8404324 0.8496864 0.8450481 0.855723
5 0.8263583
## QDA_AUC                      0.8017838 0.8270609 0.8144878 0.805459
1 0.7738646
## En_LDA_AUC                   0.8288108 0.8483983 0.8387097 0.850809
3 0.8330454
## En_QDA_AUC                   0.8222703 0.8340054 0.8268817 0.835906
2 0.7895793
## Logistic_AUC                 0.8402703 0.8526546 0.8457272 0.857005
5 0.8243522
## Back_Logistic_AUC            0.8398378 0.8462142 0.8467459 0.861759
5 0.8243522
## Best_Logistic_AUC            0.7121622 0.7035730 0.7276174 0.747823
3 0.6964057
## NB_AUC                       0.7982703 0.8428539 0.8322581 0.822285
1 0.8003344
## K_NB_AUC                     0.8246486 0.8639113 0.8509338 0.839752
1 0.8100306
## L_SVM_AUC                    0.7055405 0.7142137 0.7142332 0.736792
9 0.7247980
## R_SVM_AUC                    0.6135135 0.6310484 0.6682513 0.638801
3 0.6594873
## Classification_AUC           0.7596216 0.7434196 0.7021788 0.802120
6 0.7350237
## Boost_AUC                    0.7945946 0.8486783 0.8259762 0.814700
1 0.7712455
## RandomForest_AUC             0.8243784 0.8646953 0.8426146 0.846482
6 0.7919198
##                                   AVE
```

```
## NIR                              0.6477865
## KNN_Accuracy_Rate                0.6871379
## LDA_Accuracy_Rate                0.7654573
## QDA_Accuracy_Rate                0.7464528
## En_LDA_Accuracy_Rate             0.7653852
## En_QDA_Accuracy_Rate             0.7566649
## Logistic_Accuracy_Rate           0.7700316
## Back_Logistic_Accuracy_Rate      0.7675762
## Best_Logistic_Accuracy_Rate      0.7700316
## NB_Accuracy_Rate                 0.7616470
## K_NB_Accuracy_Rate               0.7667760
## L_SVM_Accuracy_Rate              0.7643663
## R_SVM_Accuracy_Rate              0.6991599
## Classification_Accuracy_Rate     0.7348396
## Boost_Accuracy_Rate              0.7577169
## RandomForest_Accuracy_Rate       0.7696148
## KNN_AUC                          0.7131591
## LDA_AUC                          0.8417242
## QDA_AUC                          0.8116320
## En_LDA_AUC                       0.8387898
## En_QDA_AUC                       0.8244858
## Logistic_AUC                     0.8418365
## Back_Logistic_AUC                0.8405471
## Best_Logistic_AUC                0.7085977
## NB_AUC                           0.8289352
## K_NB_AUC                         0.8454666
## L_SVM_AUC                        0.7152055
## R_SVM_AUC                        0.6235310
## Classification_AUC               0.7454179
## Boost_AUC                        0.8180335
## RandomForest_AUC                 0.8374361
```

## References

UCI Machine Learning. (2016). *Pima Indians Diabetes Database*. https://www.kaggle.com/uciml/pima-indians-diabetes-database