STAT * 6801

STATISTICAL LEARNING FINAL PROJECT

Title : Case Study Using Pima Indian Diabetes Dataset

- **Group Members:**
    - **Mohammad Sakibul Islam [ID : 1265299]**
    - **Poojan Umeshbhai Vadaliya [ID: 1281587]**

ABSTRACT:

The main objective of the project is to analyze the effectiveness of different classification models to predict outcome in the context of Pima Indian Diabetes Dataset. By showing comparison of different performance metrics (e.g. accuracy, precision, recall, area under the ROC curve etc) for different classification models, the project provides a comprehensive overview of the effectiveness of these models while making predictions for a binary classification problem. The project also highlights the analysis of the feature variables to identify the variables that have a significant impact on the outcome variable in order to predict whether a person shows signs of diabetes or not. After the evaluation of different models, it is evident that proposed two level ensemble approach (where Generalized Linear Model (GLM), Generalized additive model (GAM), Support Vector machine (SVM) model and Random Forest are trained individually in the first level and the outcome predictions from these models are provided as input to Neural Network model in second level for making the final prediction of the outcome) show higher accuracy (99%) on the test dataset compared to the other model approach.

INTRODUCTION:

Diabetes is a long-term metabolic disease characterized by high blood glucose levels, because of insufficient insulin production or ineffective insulin utilization by the body [1]. The Pima Indian Diabetes is selected for this project because the dataset allows investigation of a wide range of demographic and health related characteristics regarding diabetes to obtain insightful information about the key features that impact the risk of diabetes. So, addressing the influence of these factors on the occurrence of diabetes may provide insights into effective approaches for the risk minimization of diabetes.

DESCRIPTION OF DATASET:

The Pima Indians Diabetes dataset is derived from a study conducted by the National Institute of Diabetes and Digestive and Kidney Disease. The research investigated the health characteristics of 768 adult female Pima Indians living near Phoenix, with specific focus on the prevalence of diabetes and associated factors. The main objective of this dataset is to detect and analyze the key risk parameters associated with diabetes to gain insights into the critical factors impacting the diabetes occurrence among the female Pima Indians.

The dataset contains eight feature predictors to predict the binary dependent variable which indicates the presence or absence of diabetes. The Pima Indians Diabetes dataset consists of the following features:

- **Pregnant:** Represents the number of how many times a female Pima Indian has been pregnant.
- **Glucose:** Indicates plasma glucose concentration at 2 hours in an oral glucose tolerance test. By measuring the amount of glucose, the variable provides information on blood sugar level and metabolic health.
- **Diastolic:** Represents diastolic blood pressure, which is measured in (mmHg). Blood pressure is a physiological measure that indicates the force generated by circulating blood.
- **Triceps:** Represents triceps skinfold thickness, measured in mm.
- **Insulin:** Indicates two-hours serum insulin level, measured in mu U/ml. It represents the concentration of insulin (a hormone) present in the blood two hours after a meal.
- **BMI:** Represents body mass index, which is calculated by dividing the mass of the body by the square of height of a person. So, BMI is determined by using the formula:

$$\text{BMI} = \frac{weight\ in\ kg}{(\text{height in meter})2}$$

- **Diabetes:** Represents diabetes pedigree function which provides the probability of diabetes occurrence of a person by considering the person's family history of diabetes.
- **Age:** Represents age in years of female Pima Indians.
- **Test:** Represents the binary classification outcome variable to indicate whether a female has diabetes or not, where 1 indicates the presence of diabetes (positive) and 0 indicates the absence of diabetes (negative) in the participants.

In this project, the performance of different models (e.g.: Generalized Linear Models, Random Forest, Neural Networks, etc.) will be evaluated on the basis of its complexity level and compared to obtain insights into the effectiveness of the model to predict the outcome in context of a binary classification problem. The project includes the task of data pre-processing of Pima Indian dataset, analysis of different models based on various performance metrics and comparative assessment of the performance to find the most effective model in predicting whether the individual is diabetic or not.

DATA-PREPROCESSING:

- **Handling unrealistic values:** Since some of the feature variables (e.g.: glucose, diastolic, triceps, insulin, BMI) logically can't have 0 value, these 0 values are replaced with NA (Not Available).
- **Handling missing values:** The missing values in the dataset are imputed with the median value of the corresponding column.
- **Handling Outliers:** Box plots and summary statistics are used to investigate and detect whether there are any data points that are significantly unusual from the rest of the data points.

- **Evaluating Feature importance:** Correlation analysis by showing a heatmap is used for obtaining insights into the relationships between variables and for identifying the most important features for predicting the target variable.
- **Scaling features:** Scaling has been applied to numeric feature predictors of the dataset to standardize the values, which leads to effective model training and improved performance.
- **Splitting data in train and test set:** The test dataset is generated by taking a random sample of 100 cases and the remaining cases are assigned to the training dataset.

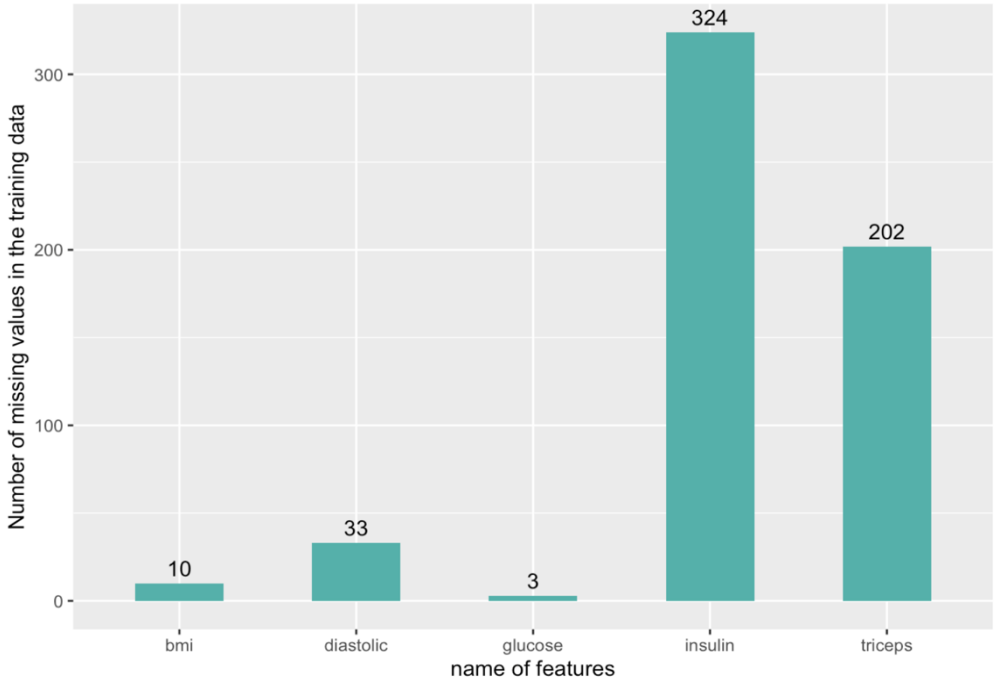


Figure 1.: Features with missing values

Out of 9 features, figure 1 represents 5 features which includes BMI, diastolic, glucose, insulin and triceps containing missing values where insulin is the one with the most frequent missing value, hence handling it by imputing the missing places with median helped deal with the problem to rectify the issue.

Heatmap in figure 2 displays the intercorrelation among individual features, which helps to evaluate how positively or negatively the feature are correlated with each other along with the target variable and provide insightful information about how a feature is affecting another feature.

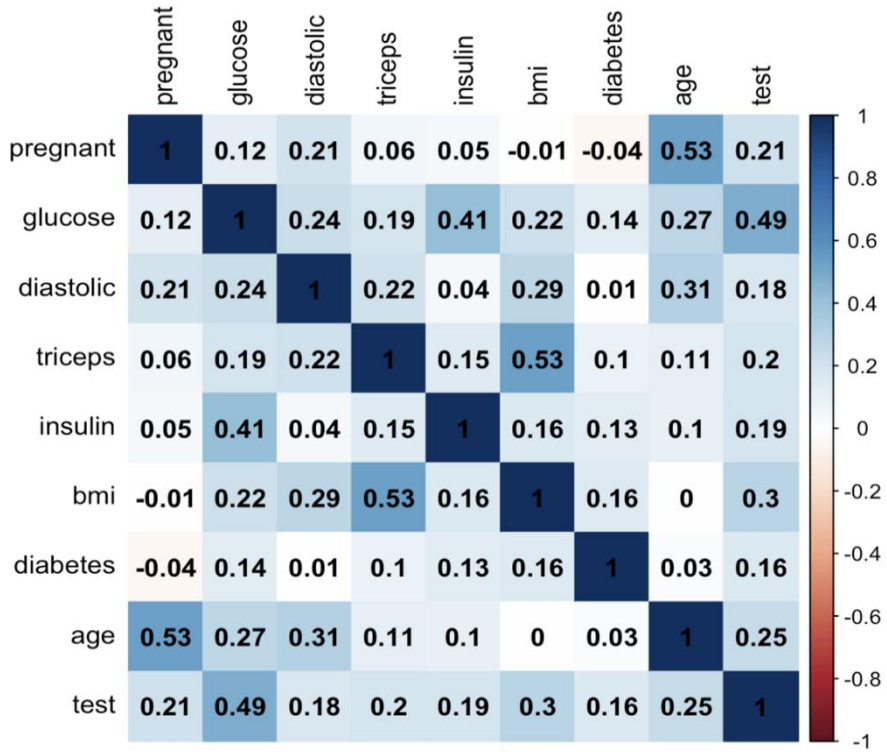


Figure 2: Correlation Analysis

ANALYSIS OF DIFFERENT MODELS FOR PREDICTION:

- **Generalized Linear Model (GLM)**

  GLMs use logistic regression to predict the outcome of the binary classification problem. Initially, the entire training dataset has been used to train the Generalized Linear Model (GLM) and predict the outcome on the test dataset using a classification threshold of 0.5 to categorize outcomes into positive and negative labels. Then, in order to obtain an improved model performance and prevent overfitting, stepwise AIC variable selection method has been applied on the GLM to get the subset model containing the most relevant

feature predictors. Following this, the reduced subset model has been used to predict the outcome and the model performance has been evaluated based on it.

- **Generalized Additive Model (GAM)**

  By using smooth functions, a generalized additive model offers enhanced flexibility in modeling of non-linear patterns among variables in the dataset. A generalized additive model has been trained using all the available feature predictors in the training dataset and the model performance has been evaluated on the test dataset.

- **Trees and Random Forest**

  Initially a default tree model has been trained using all the feature variables and a visual representation plot of the tree structure is made. Then, cross-validation is used to select the optimal tree size.

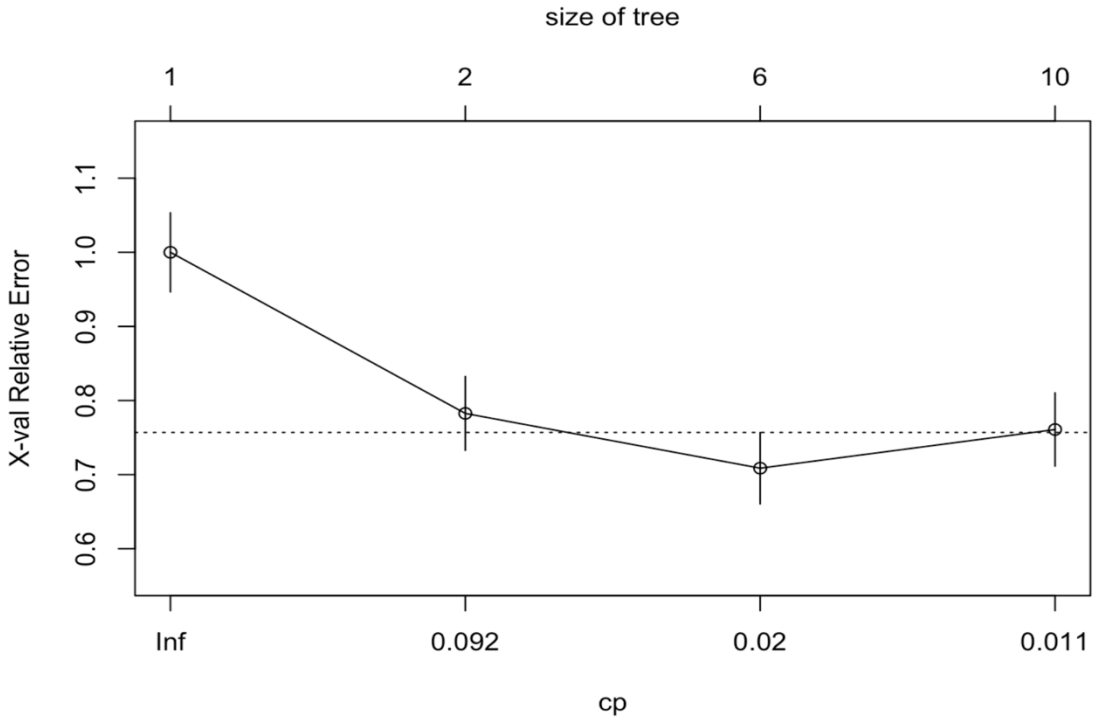


Figure 3: Decision Tree Complexity Parameter Analysis

By visualizing the impact of different complexity parameters, the complexity parameter associated with lowest value of cross-validated error is found, which indicates the optimal tree size. Then pruning (elimination of branches that have lower impact) is applied to get the final simplified tree and predictions are made on the test dataset.

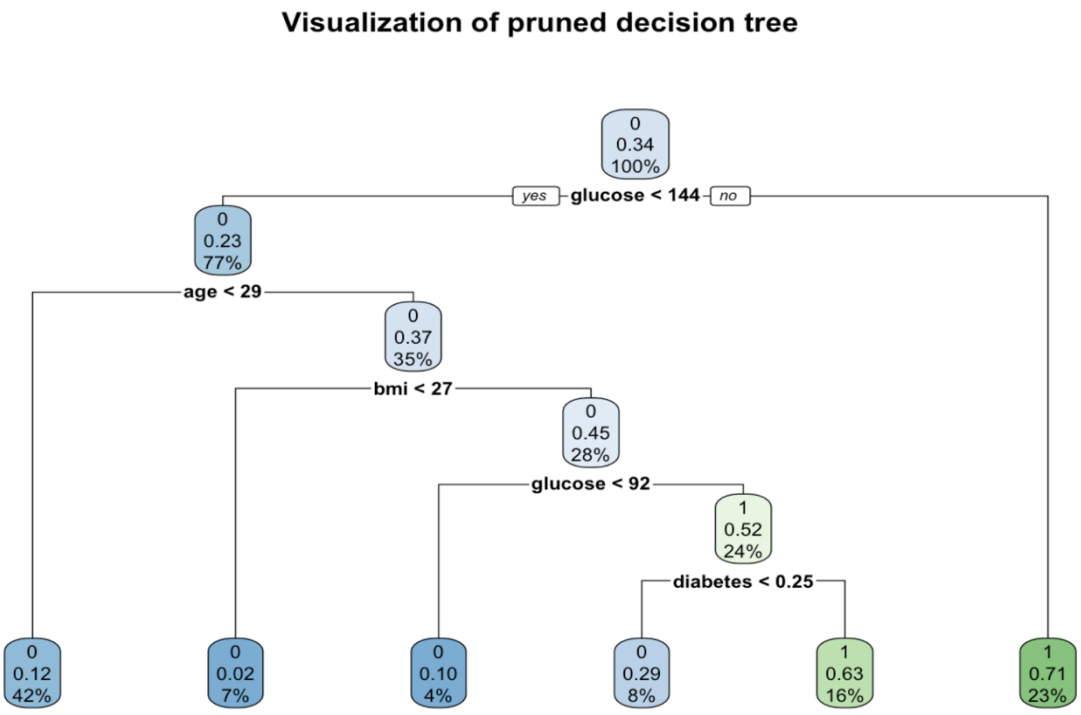


Figure 4: Pruned decision tree model with optimal complexity parameter

In case of a random forest model, the predictions of several decision trees are combined, which is an ensemble approach. A random forest model has also been trained to predict the outcomes of Pima Indians diabetes dataset. By visualizing the variable of importance using the random forest model, the most significant predictor can be identified.

- **Support Vector Machine Model (SVM)**

  Support vector machine is a supervised learning model that determines an optimal line or hyperplane to effectively separate classes in the dataset to make predictions for a classification problem. Two approaches, linear and radial SVM, are applied with 15-fold cross validation for model training and then predictions are made on the test dataset for these models.

  In the case of linear SVM, a hyperplane is used to differentiate two classes and in case of radial SVM, a flexible decision boundary is offered, which is more effective in providing improved predictions.

- **K-Nearest neighbor (KNN)**

  In the case of the K-nearest neighbor model, which is a supervised learning model, the majority class of k-nearest neighbors of a data point is considered for making predictions of the classification for the data point. K-Nearest neighbor model has been trained using the scaled train dataset and hyperparameter tuning has been applied to enhance model performance and finally, evaluation has been performed on the test dataset.

- **Neural Network**

  Neural network is a machine learning model consisting of interconnected artificial neurons[3].

  - **Unscaled Data:** First neural network model is trained using unscaled data and the architecture comprises of two hidden layers.

- **Scaled Data:** Second Neural network model was trained with a scaled data and 10 hidden layers were present to assess the impact of the difference in scaling and complexity in model performance.

- **Two Level Ensemble Approach**

  In the first level (can refer Figure 5), GLM, GAM, SVM, KNN and random forest are trained independently resulting for the output predictions of these models which are provided as features to train the artificial neural network model in the second level. By predicting outcome on the test dataset and evaluating model performance, the effectiveness of this approach has been assessed.

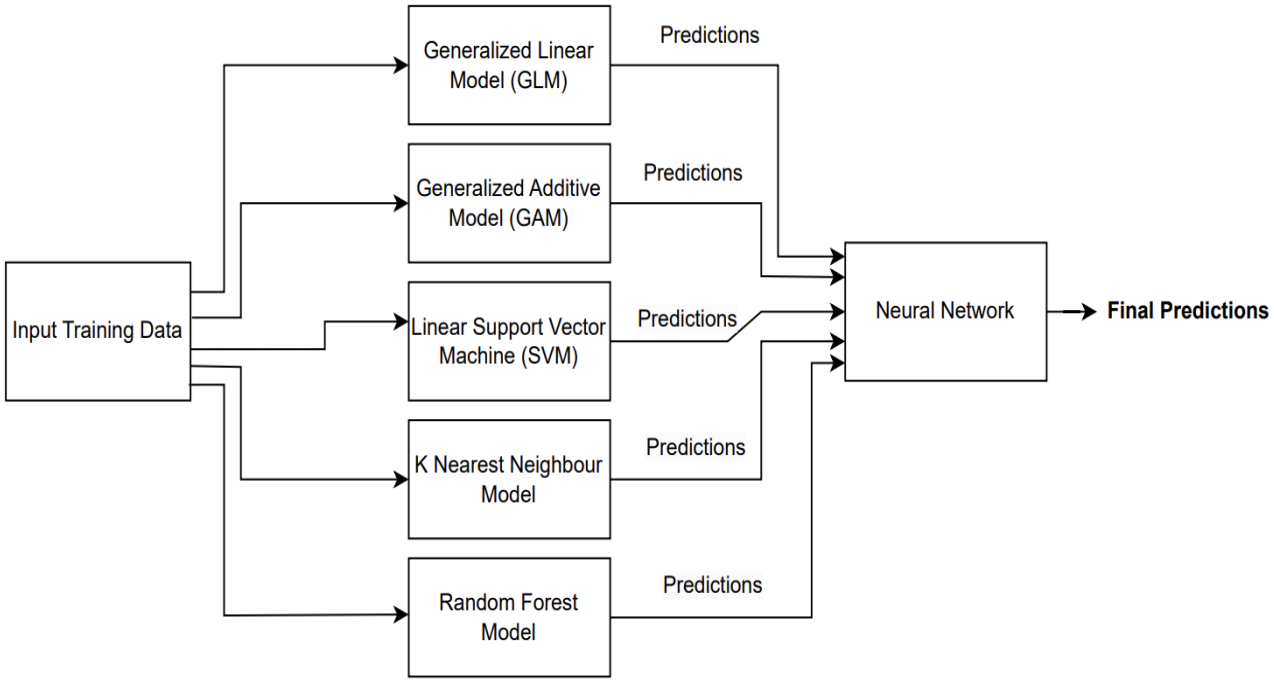


Figure 5: Architecture for two level ensemble Approach

- **Ensemble Approach of combining multiple model predictions**

  In this approach, initially GLM, GAM, SVM, KNN, random forest and neural network models are trained individually and predictions are made on the test dataset independently. Then, the predictions from these models are combined using a majority voting technique, with the most frequent prediction from the trained models is selected as final outcome.

EVALUATION METRIC OF THE MODEL PERFORMANCE

- **Accuracy**

  Accuracy is a performance metric to measure how effective the model is in predicting the outcome correctly, which is determined by dividing the accurately predicted occurrences by the total number of instances.

- **Confusion Matrix**

  Confusion matrix provides an overview of the overall performance of the model by showing the count of True Negative (correctly predicted negative instances), True Positive (correctly predicted positive class instances), False Negative (incorrectly predicted as negative instances) and False Positive predictions (incorrectly predicted as positive instances) in a table.

- **False Positive Rate and False Negative Rate**

  When a model incorrectly predicts and classifies a negative class as a positive class, it is known as false positive prediction. False positive rate is calculated by dividing the count of false positive predictions by the total count of actual negative instances.

Again, when a model incorrectly classifies a positive class as a negative class, it is known as false negative prediction. False negative rate is determined as the proportion of the number of false negative predictions to the total count of actual positive instances.

- **Precision and Recall**

  Precision is calculated by dividing the count of true positive predictions by the sum of true positive and false positive instances, which actually represents how accurate the model is when it classifies an instance as a positive class.

  Again, Recall is computed as the proportion of the number of true positive instances to the sum of true positive and false negative predictions, which represents how effective the model is in predicting actual positive class instances.

- **ROC Curve**

  The Receiver Operating Characteristic (ROC) curve provides a visual representation of the model's capacity to differentiate between two classes by plotting the true positive rate against the false positive rate. To measure the overall performance of a model using ROC curve, the area under ROC curve (AUC) metric is used, where higher value of AUC represents a better model performance.

RESULTS

The following table shows the comparative analysis by providing the measures of different evaluation metrics for 13 different approaches in context of the Pima Indian Dataset.

Comparative performance measures of different models

Method	Accuracy	False Positive Rate	False Negative Rate	Precision	Recall	ROC-AUC score
Generalized Linear Model (GLM)	0.76	0.076	0.542	0.923	0.759	0.6901
Reduced Generalized Linear Model (Subset model)	0.79	0.0769	0.457	0.923	0.789	0.7329
Generalized Additive Model (GAM)	0.78	0.1077	0.4286	0.892	0.7945	0.7319
Default Tree Model	0.8	0.1846	0.2286	0.815	0.8689	0.7934
Pruned Tree Model having optimal tree size	0.8	0.1846	0.2286	0.815	0.8689	0.7934
Random Forest (RF)	0.98	0.0153	0.0286	0.9846	0.9846	0.978
K- Nearest Neighbor Model (KNN)	0.82	0.0769	0.3714	0.923	0.8219	0.7758
Linear Support Vector Machine Model (SVM)	0.77	0.0769	0.5142	0.923	0.769	0.7044
Radial Support Vector Machine Model (SVM)	0.83	0.046	0.4	0.9538	0.8158	0.7769
Neural Network Model with unscaled Data	0.62	0.323	0.4857	0.6769	0.7213	0.5956
Neural Network Model with scaled Data	0.83	0.2154	0.0857	0.7846	0.944	0.8495
Two level Ensemble Approach (1st Level: GLM+GAM+SVM+KNN +RF, 2nd Level: ANN)	0.99	0	0.0286	1	0.9848	0.9857
Ensemble Approach of combining multiple models (Majority Voting)	0.82	0.0769	0.3714	0.923	0.821	0.7758

ANALYSIS OF THE MODEL PERFORMANCES

By observing the performance metric on the table, it is evident that the two-level ensemble approach and random forest model have displayed higher accuracy compared to the other model approach. As an ensemble learning method, random forest combines the predictions from several decision trees to minimize overfitting which results in better prediction on the test dataset. Regarding the two-level ensemble approach, since the outcomes of different models are provided as feature predictors to the neural network model to predict the target variable, the neural network can effectively learn from these in the second level of training, which lead to improved overall model performance and accuracy.

Also, In the case of neural network, there is a significant difference in accuracy and model performance between unscaled and scaled data-based training. So, feature scaling has a significant influence on the accuracy of the neural network model since it guarantees balanced contribution of each numeric feature predictors by standardizing the values, which leads to improved model performance.

The reduced GLM subset model has higher accuracy and higher ROC-AUC value than the GLM model using all the feature predictors implies that more effective predictions can be achieved by training the model with the most relevant features supporting the requirement of less data and better result. The reason behind the better performance of the subset model is that it prevents overfitting and eliminates the influence of unimportant variables by considering only the features that have significant impact on the outcome.

Regarding the Support Vector Machine model, the radial SVM model exhibits higher accuracy compared to the linear SVM because radial SVM offers a flexible decision boundary, which results in effective classification.

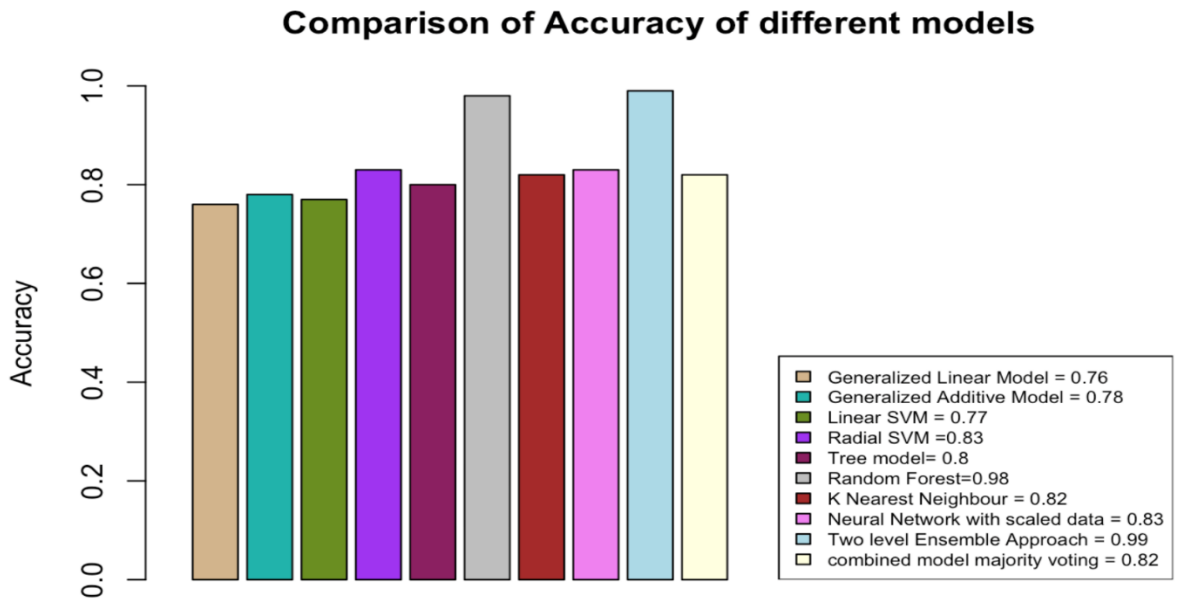


Figure 6: Comparative Analysis of accuracy for different models

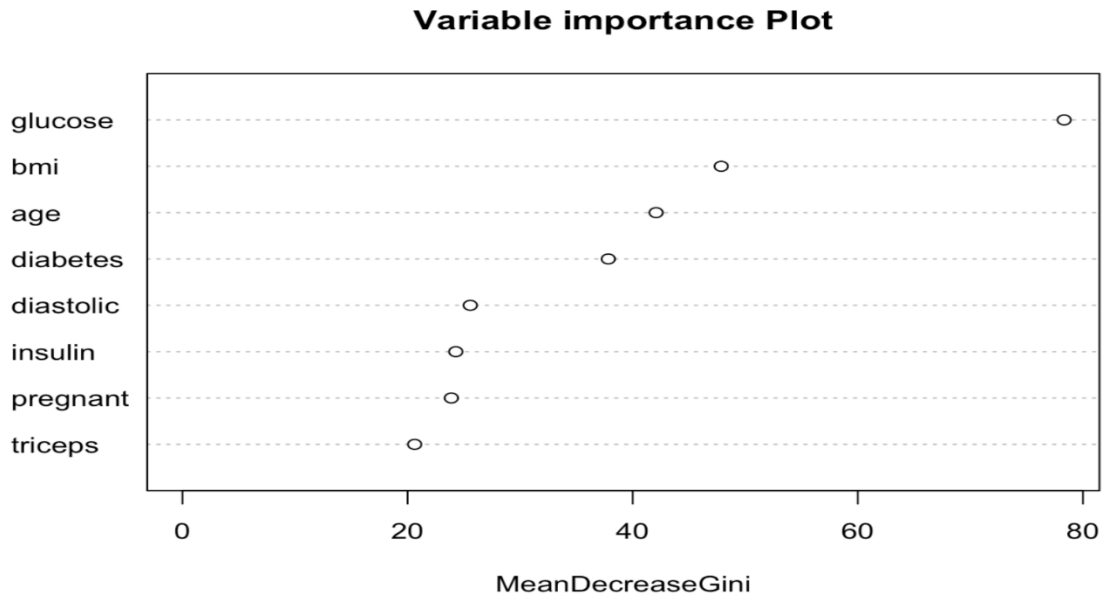


Figure 7: Variable Importance Plot

From the variable importance plot (Figure 7), it is evident that glucose is the most significant predictor feature in the Pima Indian Diabetes dataset to predict the outcome variable.

CONCLUSION:

After analyzing 13 individual techniques for the prediction of the Pima Indian dataset from the most basic technique which was a simple Generalized Linear Model and increasing the complexity leading to Neural Networks, Two-Level Ensemble Approach can be considered as the best model of all resulting into a 99% of accuracy followed by random forest with overall 98% accuracy on the test dataset.

Taking into account the ROC-AUC curve, where the optimal model achieved a score of 0.98, indicates that the model is making accurate predictions compared to other models. Utilizing the predictions obtained from different classification models in the first level to train neural networks in the second level proved to be more effective approach in the prediction of the binary outcome. Additionally, Glucose is the most effective feature followed by BMI and Age to predict the target variable.

REFERENCES:

1) World health organization, Diabetes [accessed on 2023] Retrieved from: https://www.who.int/health-topics/diabetes#tab=tab_1
2) Benarbia, Meriem, "A Machine Learning Approach to Predicting the Onset of Type II Diabetes in a Sample of Pima Indian Women" (2022). *CUNY Academic Works.* https://academicworks.cuny.edu/gc_etds/4895
3) M. Abedini, A. Bijari, and T. Banirostam, “Classification of Pima Indian diabetes dataset using ensemble of decision tree, logistic regression and neural network,” Ijarcce, vol. 9, no. 7, pp. 1–4, 2020, doi: 10.17148/ijarcce.2020.9701.
4) Link to Dataset: https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

Final Project

Mohammad Sakibul Islam & Poojan Vadaliya

2023-12-04

```
data(pima, package="faraway")
col_names <- colnames(pima)
print(col_names)
```

```
## [1] "pregnant"  "glucose"   "diastolic" "triceps"   "insulin"   "bmi"
## [7] "diabetes"  "age"       "test"
```

DATA PRE-PROCESSING

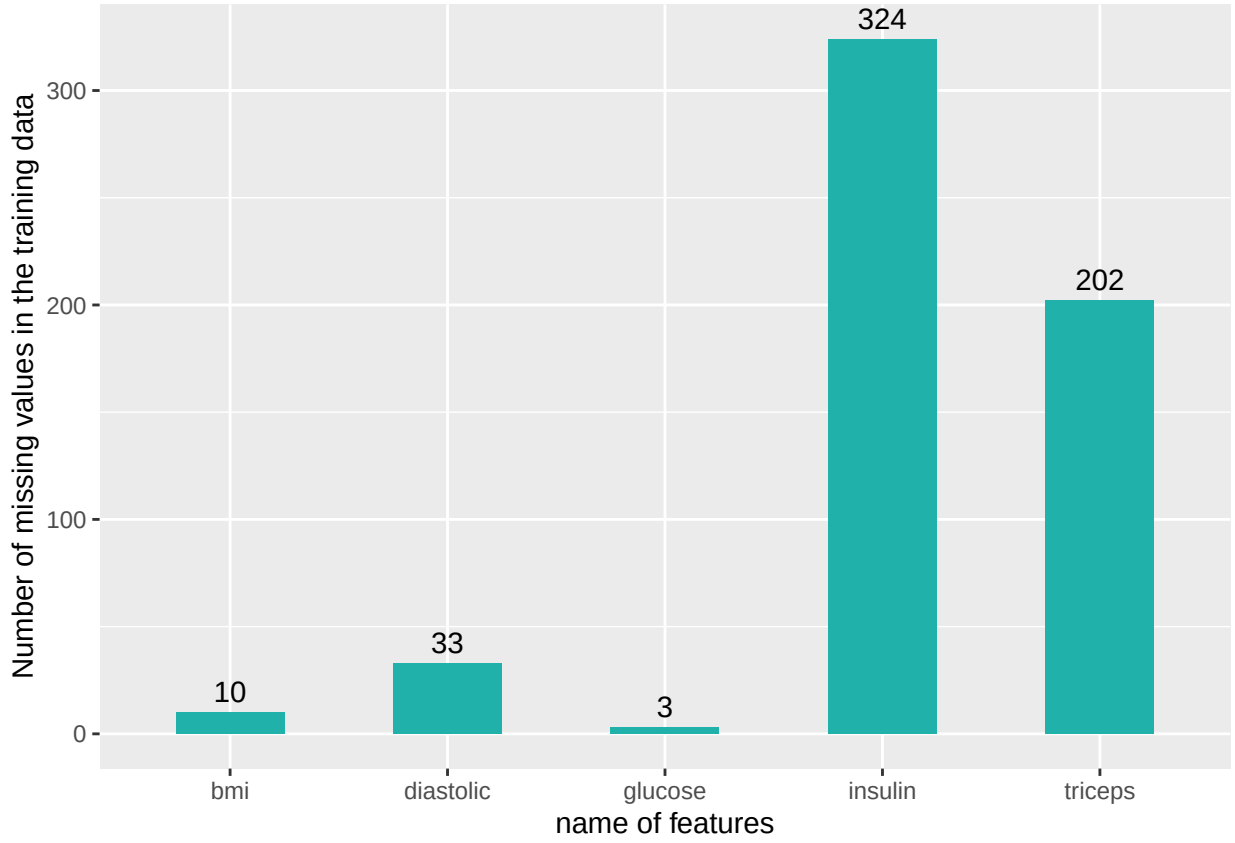
```
## replacing impossible values by NA (since glucose, diastolic,triceps,insulin and  bmi
## columns cannot have 0 value logically)

pima[c('glucose','diastolic','triceps','insulin','bmi')] <- replace(
  pima[c('glucose','diastolic','triceps','insulin','bmi')],
  pima[c('glucose','diastolic','triceps','insulin','bmi')] == 0, NA)
```

```
## Here, creating  test set taking random sample of 100 cases which contain no missing values

# selecting cases without missing values
complete_cases <- na.omit(pima)

set.seed(543)

#considering test set as a random sample of size 100
test_set_indices <- sample(nrow(complete_cases),100)
# get the indices of test set
test_dataset <- complete_cases[test_set_indices, ]

# considering the remaining cases as training set
train_dataset <- pima[- test_set_indices, ]
# number of rows in the training data set
nrow(train_dataset)
```

```
## [1] 668
```

```
library(ggplot2)
# plot the columns having missing values
```

```r
number_of_missing_values <- colSums(is.na(train_dataset))

columns_with_missing_data <- data.frame(feature = names(number_of_missing_values),
                                        count = number_of_missing_values)
columns_with_missing_data <- columns_with_missing_data[columns_with_missing_data$count >0, ]
ggplot(columns_with_missing_data, aes(x = feature, y = count)) +
  geom_col(width = 0.5, fill = "lightseagreen") +
  geom_text(aes(label = count), vjust = -0.5) +
  xlab("name of features") +
  ylab("Number of missing values in the training data")
```




```r
# Find columns having missing values
columns_having_NA <- colnames(train_dataset)[colSums(is.na(train_dataset))>0]
print(columns_having_NA)
```

```
## [1] "glucose"   "diastolic" "triceps"   "insulin"   "bmi"
```

```r
# impute missing values with median value of the corresponding column
for(col in columns_having_NA){
  median_val <- median(train_dataset[[col]], na.rm = TRUE)
  train_dataset[[col]][is.na(train_dataset[[col]])] <- median_val
}
```

```r
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.3     v readr     2.1.4
## v forcats   1.0.0     v stringr   1.5.0
## v lubridate 1.9.2     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.0
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

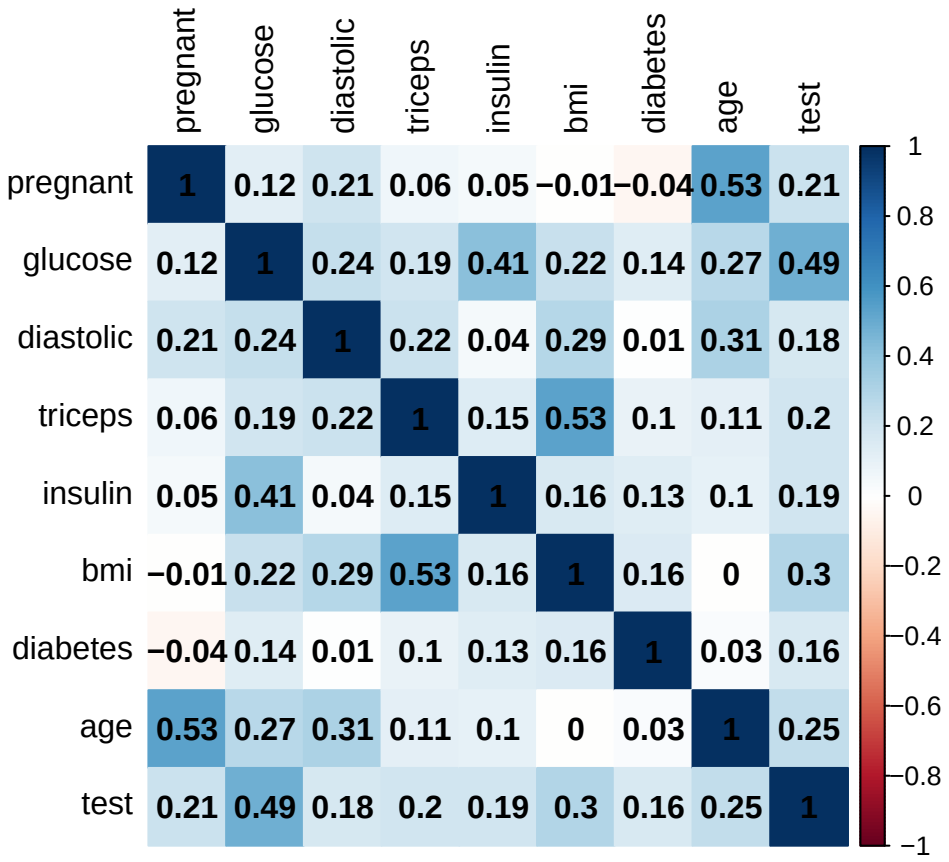
```r
## numeric feature columns
numeric_feature_columns <- train_dataset %>%
  select(-test) %>%
  select_if(is.numeric)

numeric_columns_name <- names(numeric_feature_columns)
numeric_columns_name
```

```
## [1] "pregnant"  "glucose"   "diastolic" "triceps"   "insulin"   "bmi"
## [7] "diabetes"  "age"
```

```r
library(ggplot2)
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
## Calculating correlation matrix for the training set
correlation_matrix <- cor(train_dataset)
corrplot(correlation_matrix, method ="color", addCoef.col = "black", tl.col ="black")
```

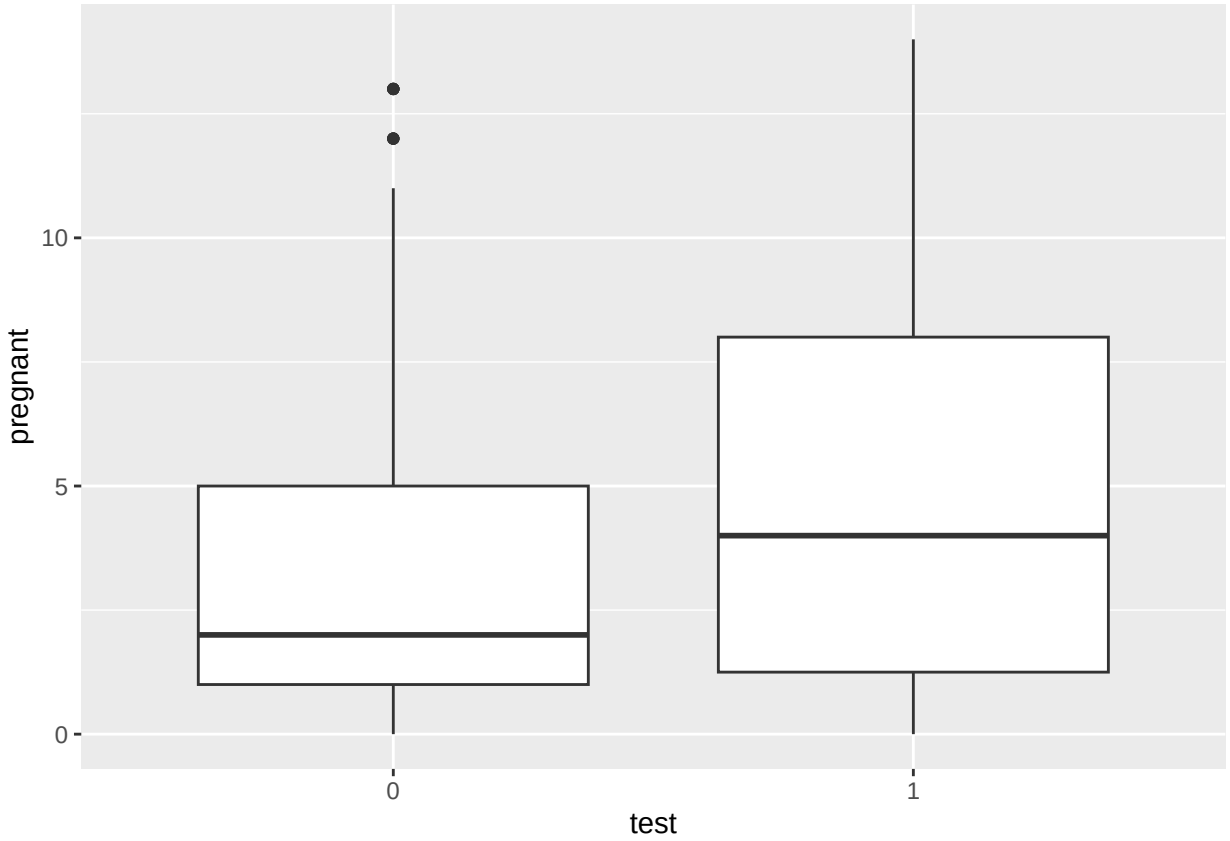



```
## print correlation with target variable 'test'
cor_with_target <- correlation_matrix['test', ]
cor_with_target
```

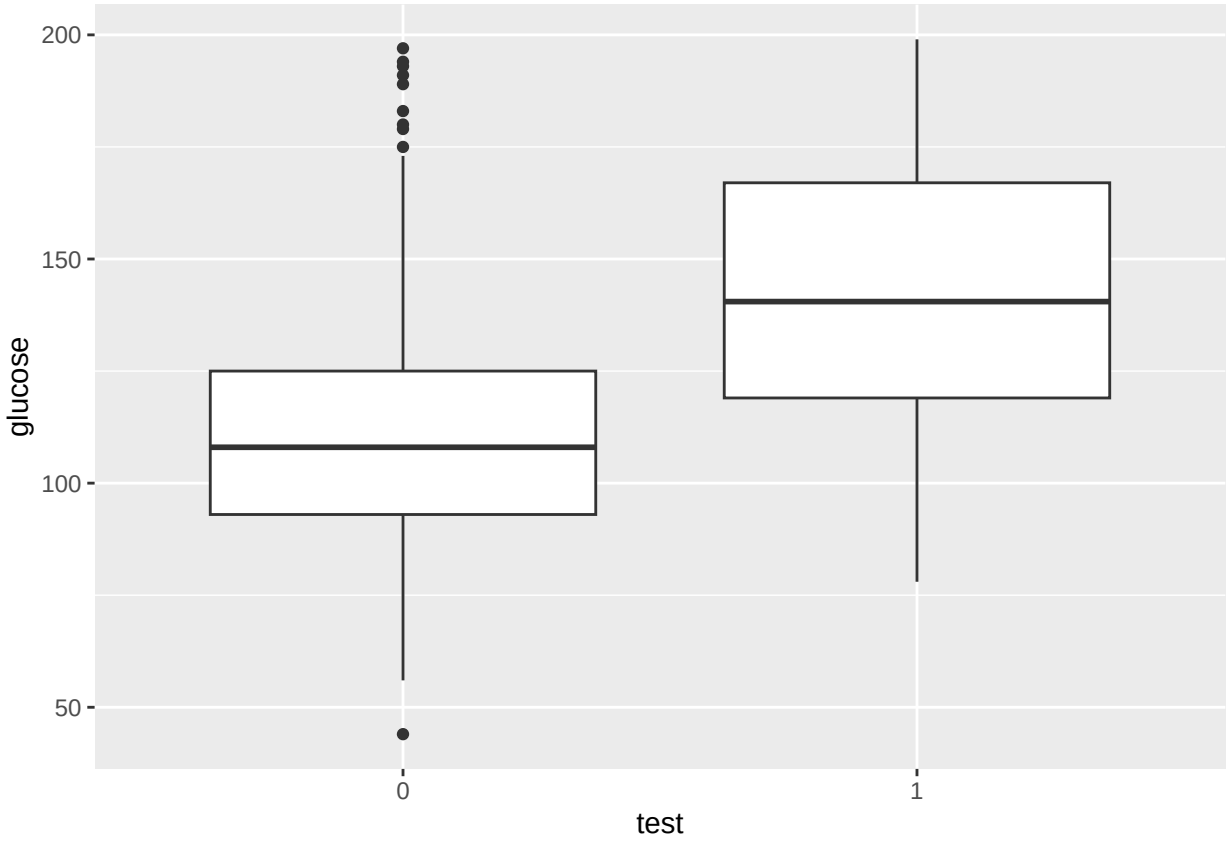
```
##  pregnant   glucose diastolic   triceps   insulin       bmi  diabetes       age
## 0.2128139 0.4855857 0.1790200 0.1962702 0.1945301 0.2990465 0.1621535 0.2489214
##      test
## 1.0000000
```

from this correlation plot , we can figure out that glucose, bmi, age , pregnant and diabetes are highly correlated with target 'test' variable

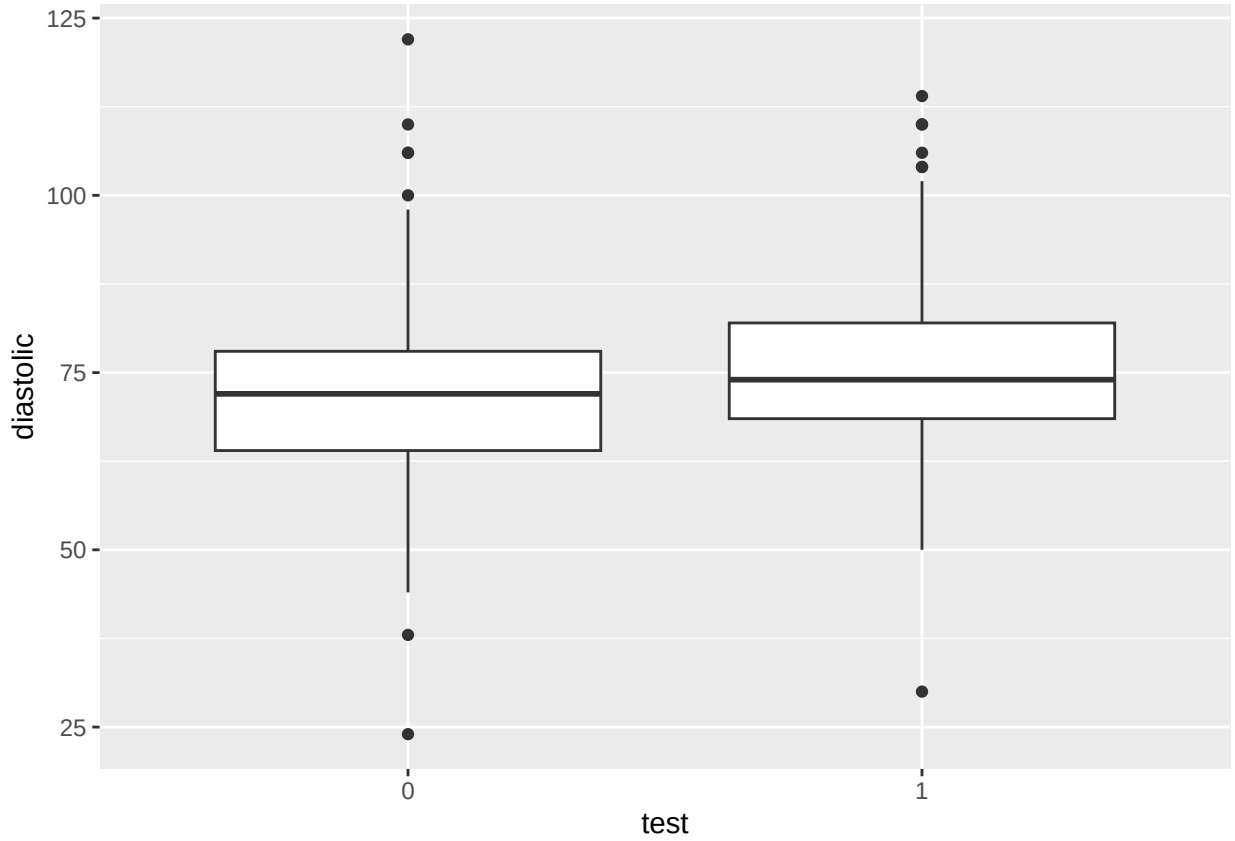
```
## encode test as a factor to explicitly indicate that it is categorical variable with two labels
train_dataset$test <- as.factor(train_dataset$test)
test_dataset$test <- as.factor(test_dataset$test)

library(ggplot2)
# creating boxplots

ggplot(train_dataset, aes(x = test, y = pregnant))+ geom_boxplot()
```



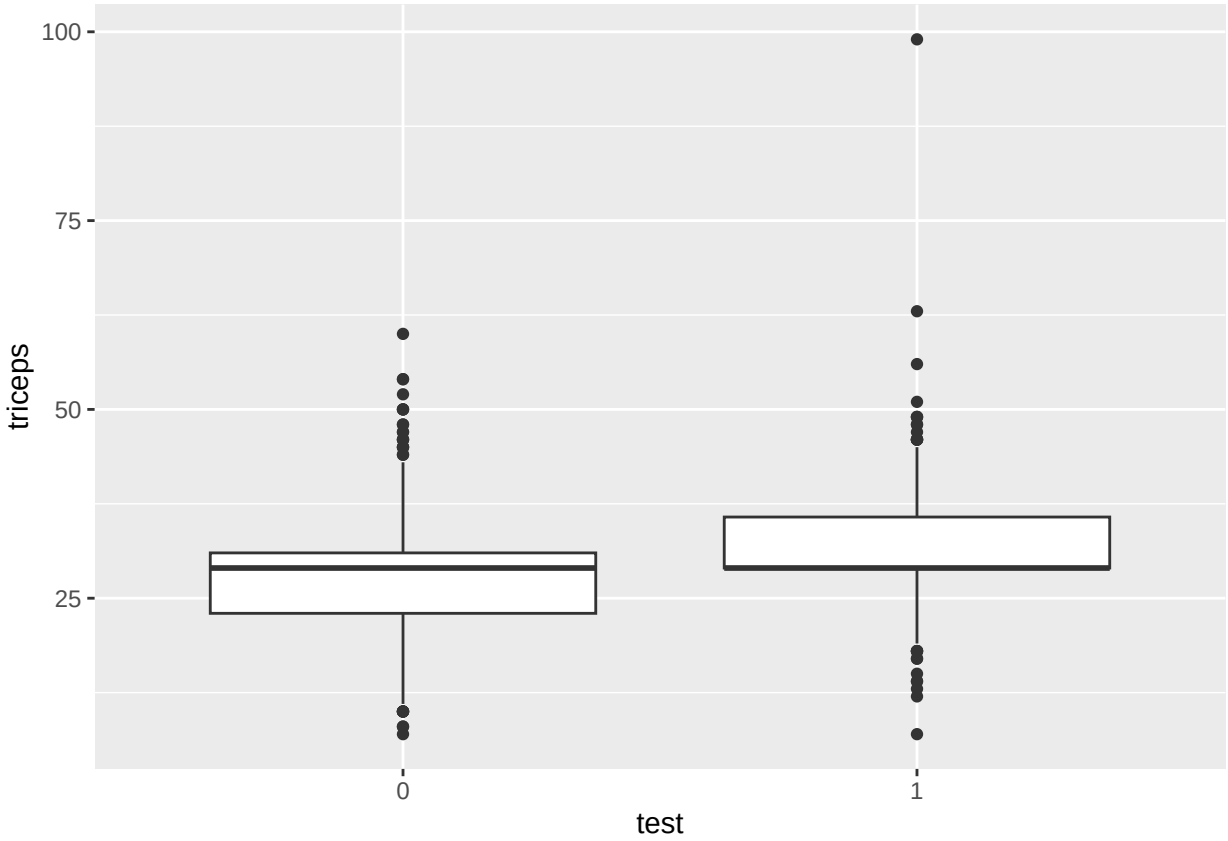

```
ggplot(train_dataset, aes(x = test, y = glucose))+ geom_boxplot()
```



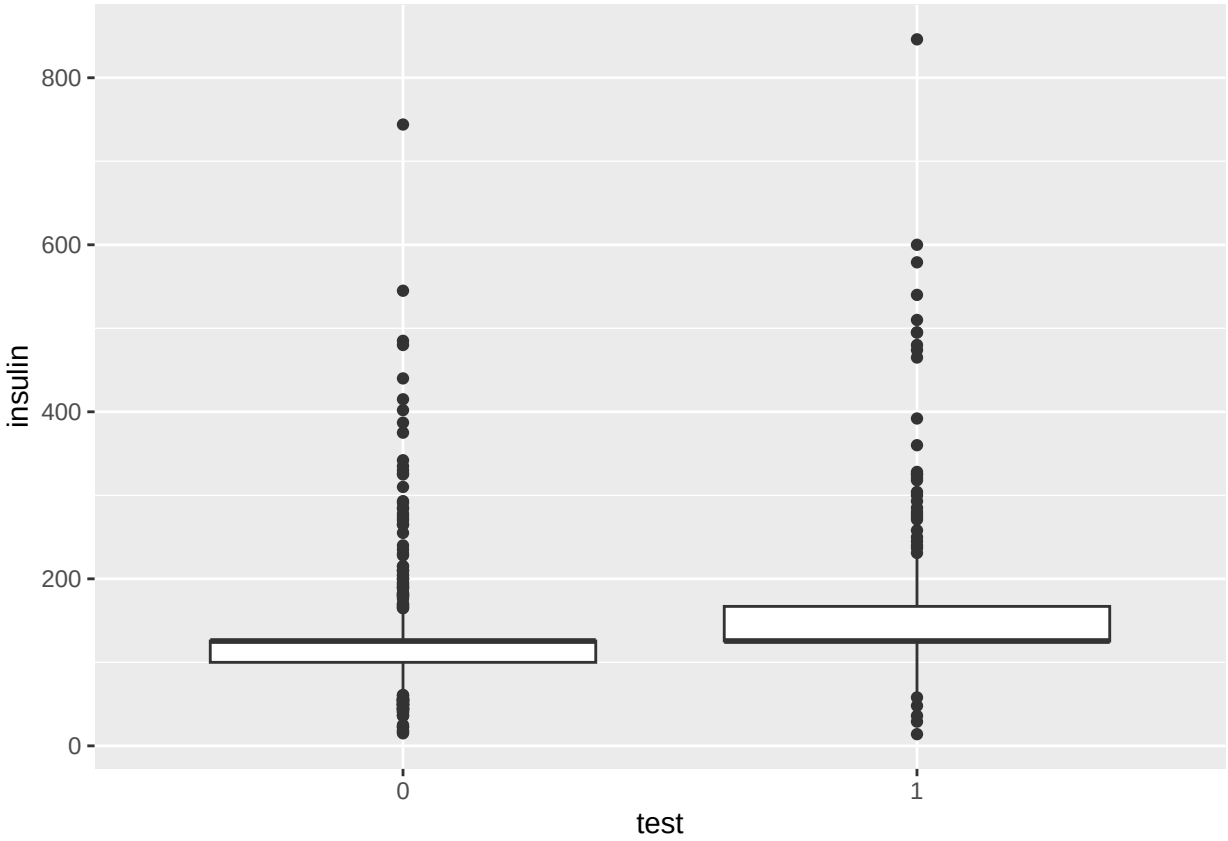

```
ggplot(train_dataset, aes(x = test, y = diastolic))+ geom_boxplot()
```



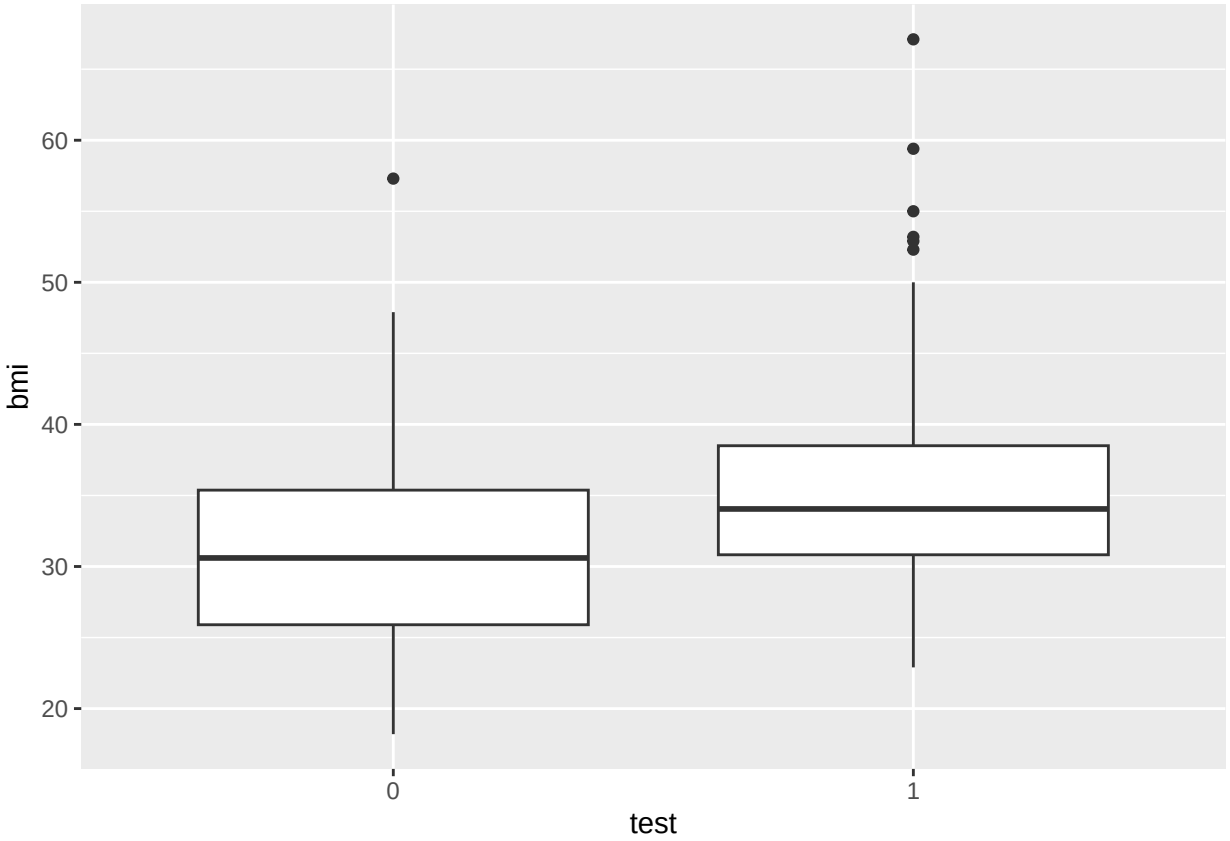

```
ggplot(train_dataset, aes(x = test, y = triceps))+ geom_boxplot()
```



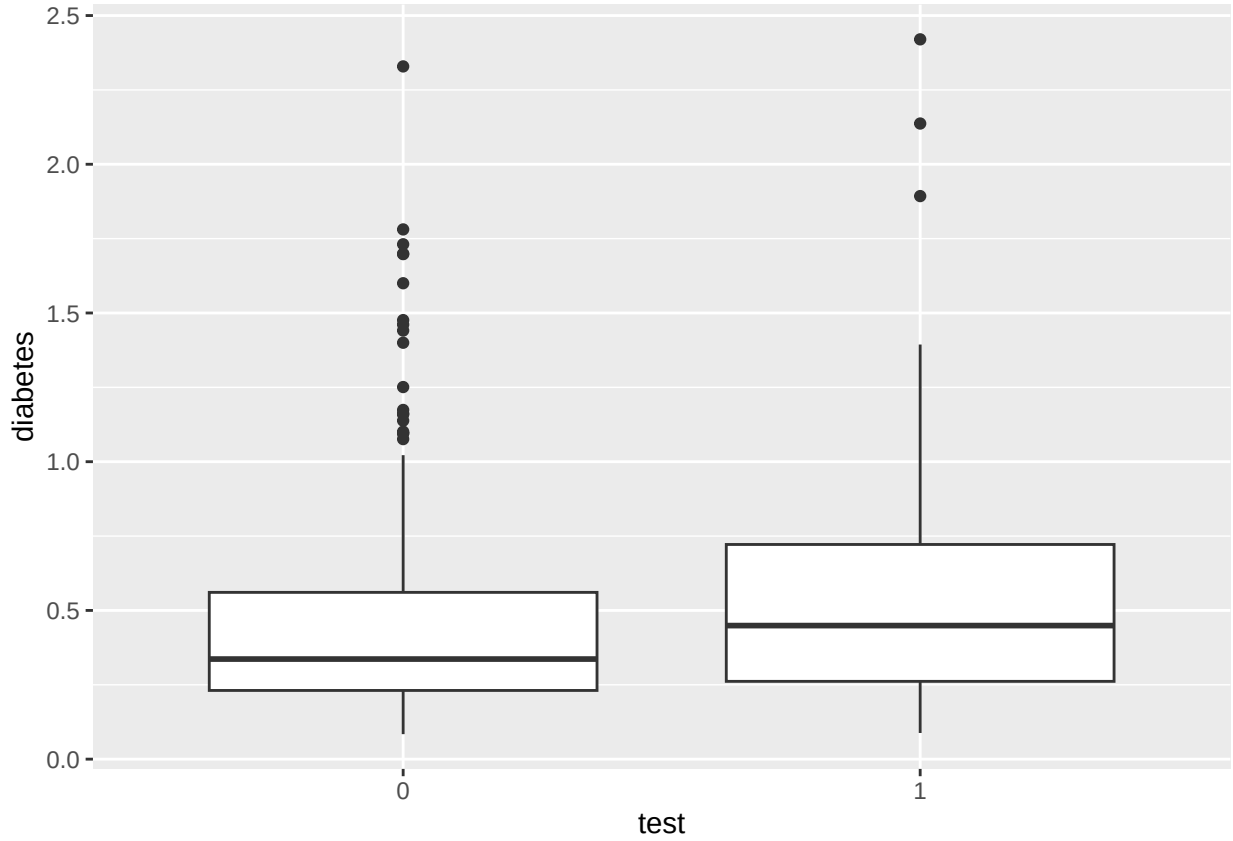

```
ggplot(train_dataset, aes(x = test, y = insulin))+ geom_boxplot()
```



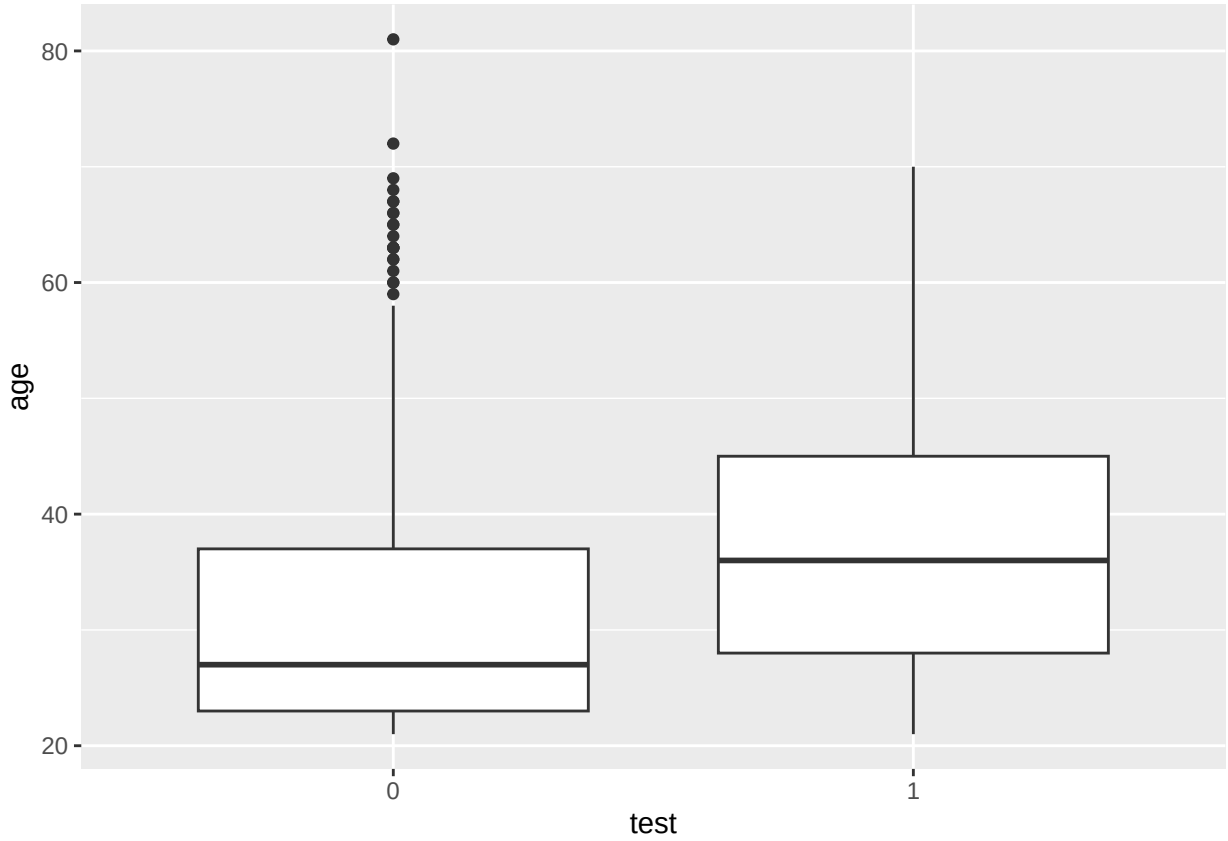

```
ggplot(train_dataset, aes(x = test, y = bmi))+ geom_boxplot()
```




```
ggplot(train_dataset, aes(x = test, y = diabetes))+ geom_boxplot()
```




```
ggplot(train_dataset, aes(x = test, y = age))+ geom_boxplot()
```



GLM MODEL

```
## fitting a GLM model on the training data set
glm_model <- glm(test ~ . , family = "binomial", data = train_dataset)

# print the summary of the model
summary(glm_model)
```

```
##
## Call:
## glm(formula = test ~ ., family = "binomial", data = train_dataset)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.076630   0.876750 -10.353  < 2e-16 ***
## pregnant     0.113030   0.034026   3.322 0.000894 ***
## glucose      0.036727   0.004112   8.931  < 2e-16 ***
## diastolic   -0.008452   0.009366  -0.902 0.366871
## triceps     -0.003019   0.013843  -0.218 0.827383
## insulin     -0.001179   0.001206  -0.978 0.328042
## bmi          0.097486   0.019259   5.062 4.15e-07 ***
## diabetes     0.793675   0.315169   2.518 0.011794 *
## age          0.018869   0.009989   1.889 0.058902 .
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 860.19  on 667  degrees of freedom
## Residual deviance: 624.88  on 659  degrees of freedom
## AIC: 642.88
##
## Number of Fisher Scoring iterations: 5
```

```
## predicting the response probabilities on test dataset
predict_probabilities <- predict(glm_model, type = "response", newdata = test_dataset)

## converting the probabilities into test predictions considering threshold as 0.5
predictions <- ifelse(predict_probabilities >= 0.5, "Positive", "Negative")


##
library(dplyr)
actual_testset <- ifelse(test_dataset$test == 0, "Negative","Positive")
glm_confusion_matrix <- table(actual = actual_testset,
                              predicted = predictions)
print(glm_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       60        5
##   Positive       19       16
```

```
#try another approach
#test_dataset <-mutate(test_dataset,
#predicted =  ifelse(predict_probabilities>=0.5,"Positive","Negative"))
#xtabs(~ actual_testset + predicted , test_dataset)
```

```
## A function which returns accuracy, false positive rate, false negative rate

evaluation_metrics <- function(confusion_matrix) {
  ## calculating accuracy
  accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
  cat("Accuracy : ", accuracy, "\n")
  true_positive_prediction <- confusion_matrix[1,1]
  false_positive_prediction <- confusion_matrix[1,2]
  true_negative_prediction <- confusion_matrix[2,2]
  false_negative_prediction <- confusion_matrix[2,1]

  false_positive_rate <- false_positive_prediction / (true_positive_prediction +
                                                        false_positive_prediction)
  cat("False Positive Rate : ", false_positive_rate,"\n")
  false_negative_rate <- false_negative_prediction / (false_negative_prediction +
                                                        true_negative_prediction)
  cat("False Negative Rate : ", false_negative_rate,  "\n")

  precision <- true_positive_prediction / (true_positive_prediction +
```

```
                                          false_positive_prediction)
  cat("precision : ", precision,  "\n")

  recall <- true_positive_prediction / (true_positive_prediction +
                                          false_negative_prediction)
  cat("recall : ", recall,  "\n")
  return(list(accuracy = accuracy,
              false_positive_rate = false_positive_rate,
              false_negative_rate = false_negative_rate,
              precision = precision,
              recall = recall))
}
```

```
#install.packages("pROC", repos = "https://cran.r-project.org")
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.

##
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
## A function that plots ruc curve and shows AUC (Area Under Curve )
roc_curve_plot <- function(predictions){
  ## Creating a ROC curve
  roc_curve <- roc(test_dataset$test, predictions)

  ## plot ROC curve
  plot(roc_curve, main ="ROC curve", xlab ="False Positive Rate",
       ylab ="True Positive Rate")

  # Calculate the area under the curve
  auc_value <- auc(roc_curve)
  cat("AUC: ", auc_value, "\n")
}
```

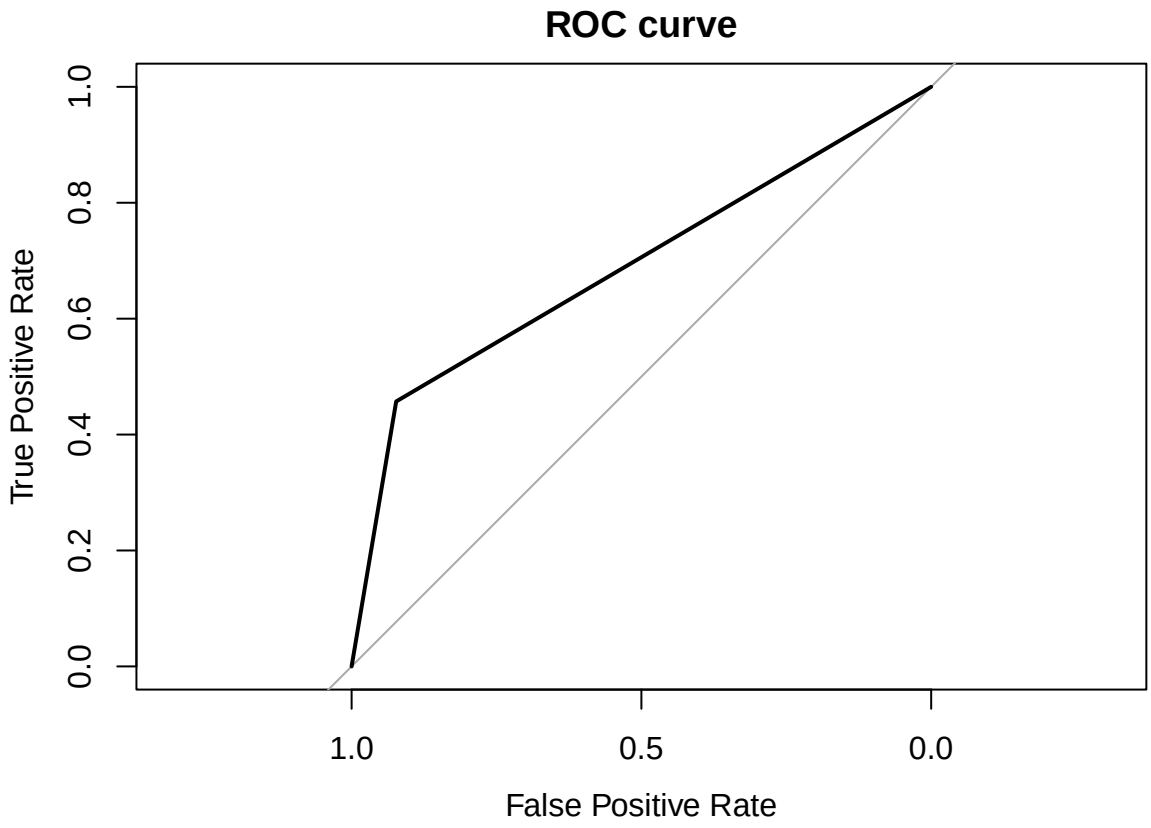
```
# Evaluating glm model performance
glm_model_performance <- evaluation_metrics(glm_confusion_matrix)
```

```
## Accuracy :  0.76
## False Positive Rate :  0.07692308
## False Negative Rate :  0.5428571
## precision :  0.9230769
## recall :  0.7594937
```

```
predictions <- ifelse(predict_probabilities >=0.5, 1, 0)
roc_curve_plot(predictions)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
## AUC:  0.6901099
```

Reduced GLM Subset Model

```
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
# using step to get subset model
subset_model <- step(glm_model, trace = 0)
summary(subset_model)
```

```
##
## Call:
## glm(formula = test ~ pregnant + glucose + bmi + diabetes + age,
##     family = "binomial", data = train_dataset)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.382449   0.790158 -11.874  < 2e-16 ***
## pregnant     0.110394   0.033735   3.272  0.00107 **
## glucose      0.034878   0.003754   9.290  < 2e-16 ***
## bmi          0.089463   0.015898   5.627 1.83e-08 ***
## diabetes     0.782090   0.315120   2.482  0.01307 *
## age          0.016997   0.009663   1.759  0.07860 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 860.19  on 667  degrees of freedom
## Residual deviance: 626.54  on 662  degrees of freedom
## AIC: 638.54
##
## Number of Fisher Scoring iterations: 5
```

```
## predicting the response probabilities on test dataset
predict_probabilities <- predict(subset_model, type = "response",
                                 newdata = test_dataset)

## converting the probabilities into test predictions considering threshold as 0.5
predictions <- ifelse(predict_probabilities >=0.5, "Positive", "Negative")


##
library(dplyr)
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")
glm_confusion_matrix <- table(actual = actual_testset,
                                 predicted = predictions)
print(glm_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       60        5
##   Positive       16       19
```

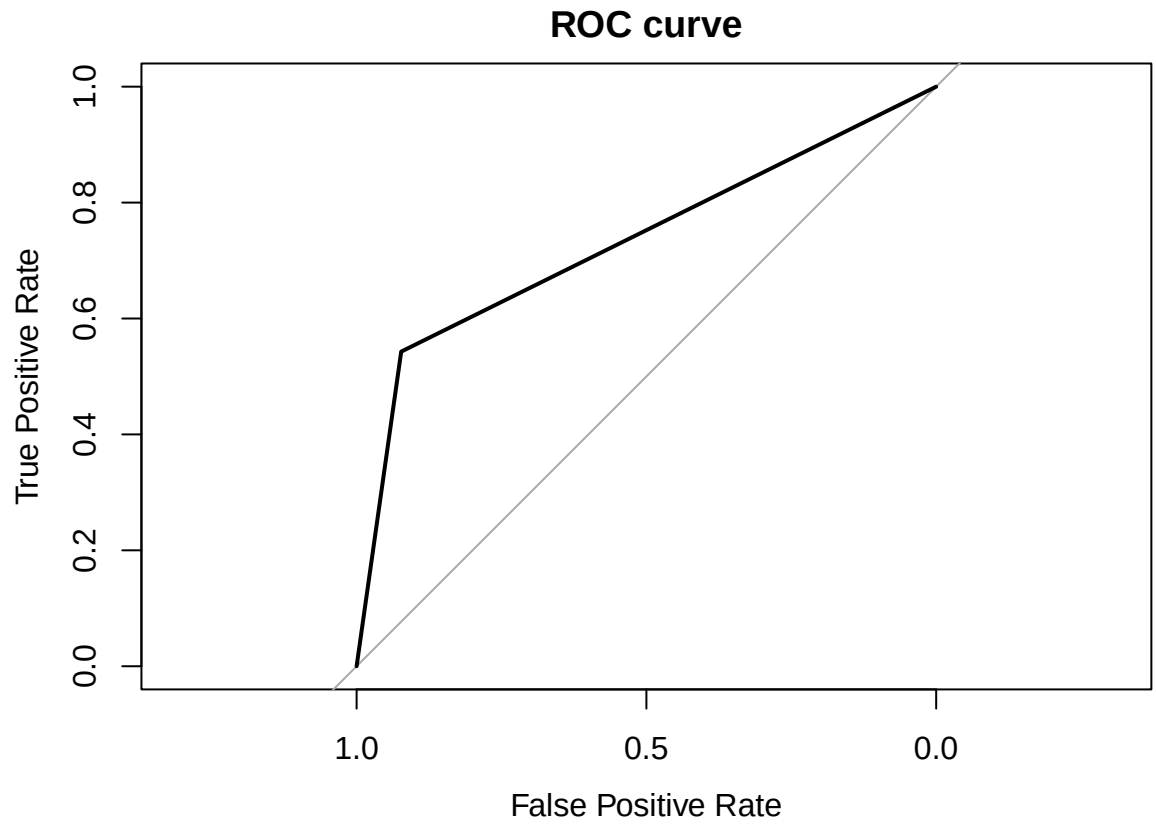
```
# Evaluating subset model performance
subset_model_performance <- evaluation_metrics(glm_confusion_matrix)
```

```
## Accuracy :  0.79
## False Positive Rate :  0.07692308
## False Negative Rate :  0.4571429
## precision :  0.9230769
## recall :  0.7894737
```

```
predictions <- ifelse(predict_probabilities >=0.5, 1, 0)
roc_curve_plot(predictions)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
## AUC:  0.732967
```

Generalized Additive Model (GAM)

```
library(mgcv)
```

```
## Loading required package: nlme
```

```
##
## Attaching package: 'nlme'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
## This is mgcv 1.8-42. For overview type 'help("mgcv-package")'.
```

```
## fitting an additive model for the test using all the available predictors
additive_model <- gam(test ~ s(pregnant) + s(glucose)+ s(diastolic) + s(triceps)
                      + s(insulin) + s(bmi) + s(diabetes) + s(age),
                      data = train_dataset, family = binomial)

summary(additive_model)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## test ~ s(pregnant) + s(glucose) + s(diastolic) + s(triceps) +
##     s(insulin) + s(bmi) + s(diabetes) + s(age)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -1.0554     0.1201  -8.789   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                edf Ref.df Chi.sq  p-value
## s(pregnant)  1.000  1.000  1.574 0.209716
## s(glucose)   1.000  1.000 77.638  < 2e-16 ***
## s(diastolic) 1.000  1.000  1.376 0.240817
## s(triceps)   1.000  1.000  0.034 0.854282
## s(insulin)   3.400  4.242  2.947 0.580752
## s(bmi)       3.815  4.791 30.684 1.29e-05 ***
## s(diabetes)  2.211  2.787 11.179 0.007303 **
## s(age)       3.380  4.214 20.813 0.000502 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.368   Deviance explained = 33.5%
## UBRE = -0.089866  Scale est. = 1         n = 668
```

A predictor with a smaller p-value is typically considered to have a stronger relationship with the response. So,'glucose' predictor has the strongest relation to response

```
## gam model predictions on the test set
gam_model_predictions <- predict(additive_model, type ="response",
                                 newdata = test_dataset)
## converting the probabilities into test predictions considering threshold as 0.5
gam_predictions <- ifelse(gam_model_predictions >=0.5, "Positive",
                      "Negative")
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")
gam_confusion_matrix <- table(actual = actual_testset,
                                 predicted = gam_predictions)
print(gam_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       58        7
##   Positive       15       20
```

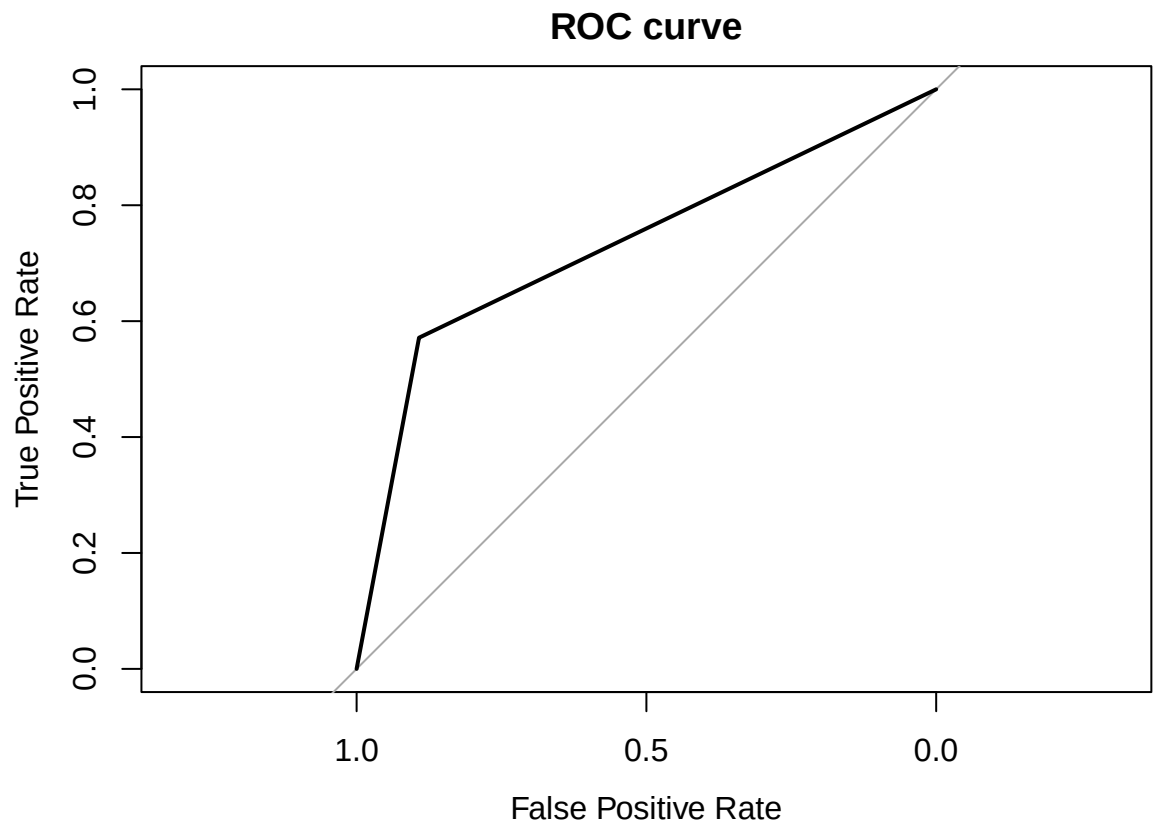
```
# Evaluating gam model performance
gam_model_performance <- evaluation_metrics(gam_confusion_matrix)
```

```
## Accuracy :  0.78
## False Positive Rate :  0.1076923
## False Negative Rate :  0.4285714
## precision :  0.8923077
## recall :  0.7945205
```

```
gam_predictions <- ifelse(gam_model_predictions >=0.5, 1, 0)
roc_curve_plot(gam_predictions)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

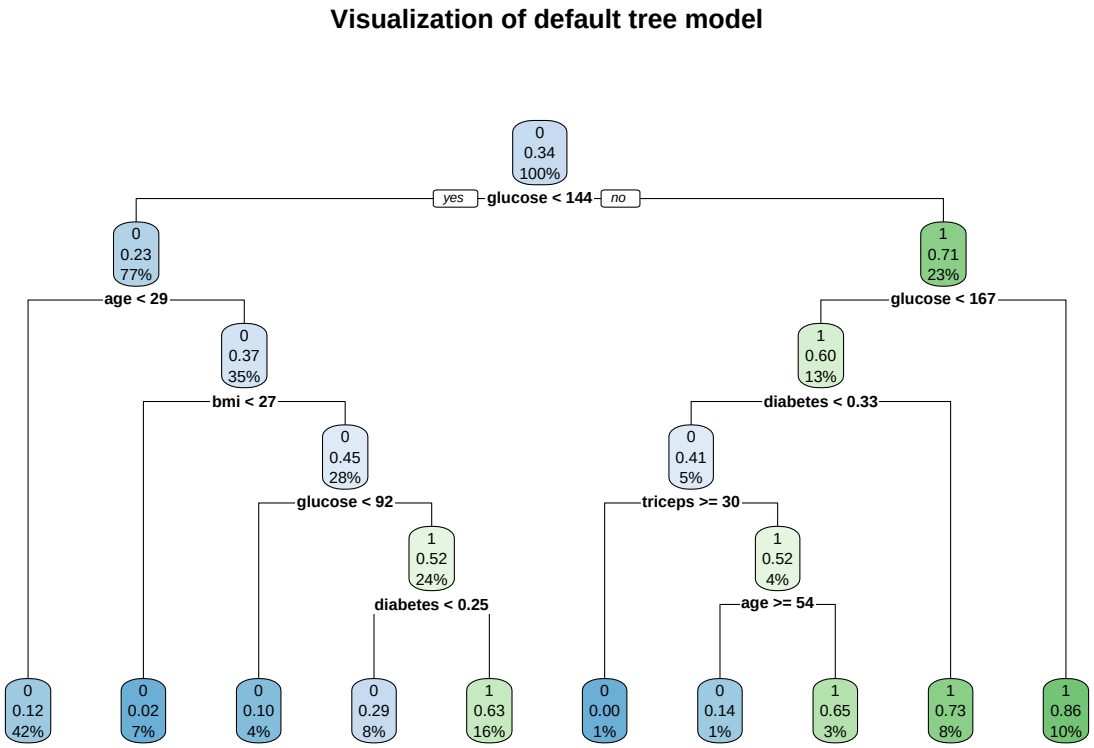



```
## AUC:  0.7318681
```

We know that high AUC (Area Under Curve) indicate better model performance.

Default TREE Model

```
## fitting a default tree model with diabetes test as a response and all other variables as response
options(repos = c(CRAN = "https://cran.r-project.org"))
library(rpart)
library(rpart.plot)


# fit the decision tree model
tree_model <- rpart( test ~ ., data = train_dataset)
# creating a plot of the tree
rpart.plot(tree_model, main = "Visualization of default tree model")
```




```
## tree predictions on the test set
tree_model_prediction <- predict(tree_model, type ="class",
                                 newdata = test_dataset)

## labeling predictions as negative and positive
tree_model_predictions<- ifelse(tree_model_prediction == 0, "Negative", "Positive")

# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test == 0, "Negative","Positive")

## create a confusion matrix to evaluate model performance
tree_confusion_matrix <- table(actual = actual_testset,
```

```
                                predicted = tree_model_predictions)
print(tree_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       53       12
##   Positive        8       27
```

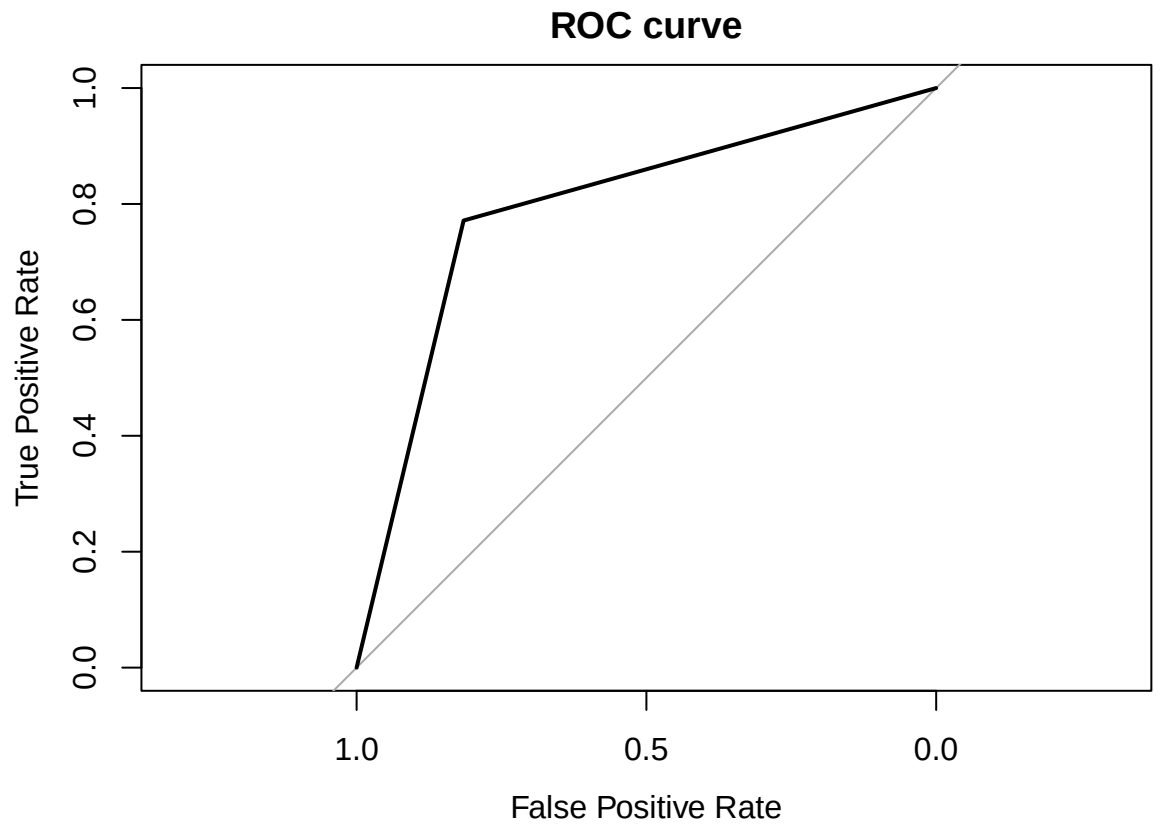
```
### Model Performance Evaluation
tree_model_performance <- evaluation_metrics(tree_confusion_matrix)
```

```
## Accuracy :  0.8
## False Positive Rate :  0.1846154
## False Negative Rate :  0.2285714
## precision :  0.8153846
## recall :  0.8688525
```

```
#tree_model_performance
roc_curve_plot(as.numeric(tree_model_prediction))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```



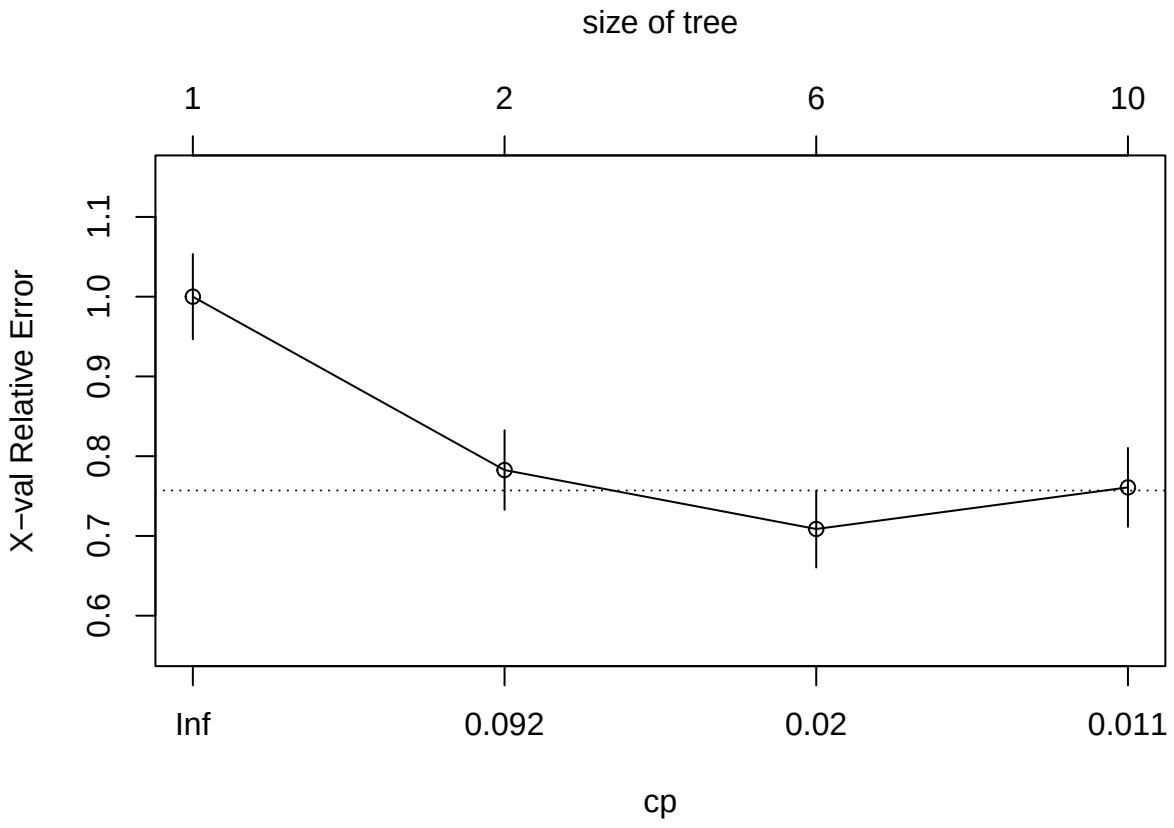

```
## AUC:  0.7934066
```

Pruned Tree Model with optimal tree size

```
## Using cross-validation to select the optimal tree size
## selecting appropriate optimal tree size is necessary for better performance
decision_tree_models <- rpart( formula = test ~ . ,
                                data = train_dataset,
                                method = "class")
## printing the complexity parameter which helps to select appropriate tree size
printcp(decision_tree_models)
```

```
##
## Classification tree:
## rpart(formula = test ~ ., data = train_dataset, method = "class")
##
## Variables actually used in tree construction:
## [1] age      bmi      diabetes glucose  triceps
##
## Root node error: 230/668 = 0.34431
##
## n= 668
##
##         CP nsplit rel error  xerror     xstd
## 1 0.286957      0   1.00000 1.00000 0.053393
## 2 0.029348      1   0.71304 0.78261 0.049857
## 3 0.013043      5   0.59565 0.70870 0.048264
## 4 0.010000      9   0.54348 0.76087 0.049411
```

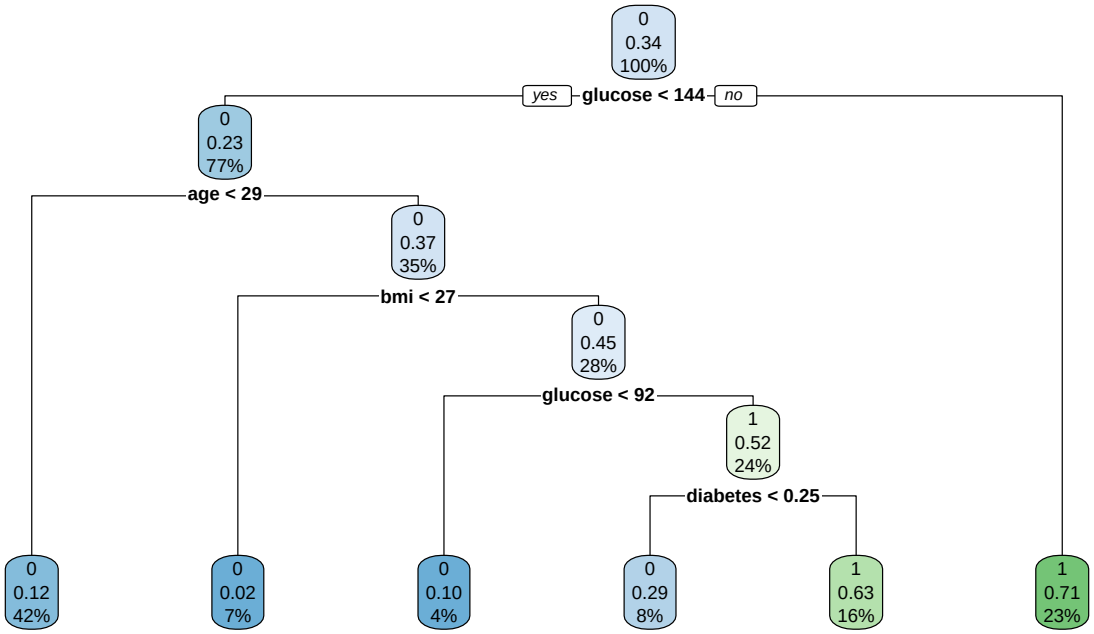
```
## visualizing complexity parameter
plotcp(decision_tree_models)
```



the row having the lowest value of xerror (cross-validated error) indicate the optimal tree size. Choosing the size of tree by selecting the value of CP which corresponds to the minimum value of xerror.

```
cv_error <- decision_tree_models$cptable[, "xerror"]
min_cv_error <- which.min(cv_error)

optimal_complexity_parameter <- decision_tree_models$cptable[min_cv_error,"CP"]

## Pruning the decision tree to get the final simplified tree
final_tree_model <- prune(decision_tree_models, cp = optimal_complexity_parameter )
rpart.plot(final_tree_model, main = "Visualization of pruned decision tree")
```

Visualization of pruned decision tree




```
## tree predictions on the test set
test_dataset$test <- as.factor(test_dataset$test)
tree_model_prediction <- predict(final_tree_model,
                                 newdata = test_dataset,type ="class")
## labeling predictions as negative and positive
tree_model_predictions<- ifelse(tree_model_prediction == 0, "Negative", "Positive")
# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")

tree_confusion_matrix <- table(actual = actual_testset,
                                 predicted = tree_model_predictions)
print(tree_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       53       12
##   Positive        8       27
```

```
### pruned final tree Model Performance Evaluation
tree_model_performance <- evaluation_metrics(tree_confusion_matrix)
```

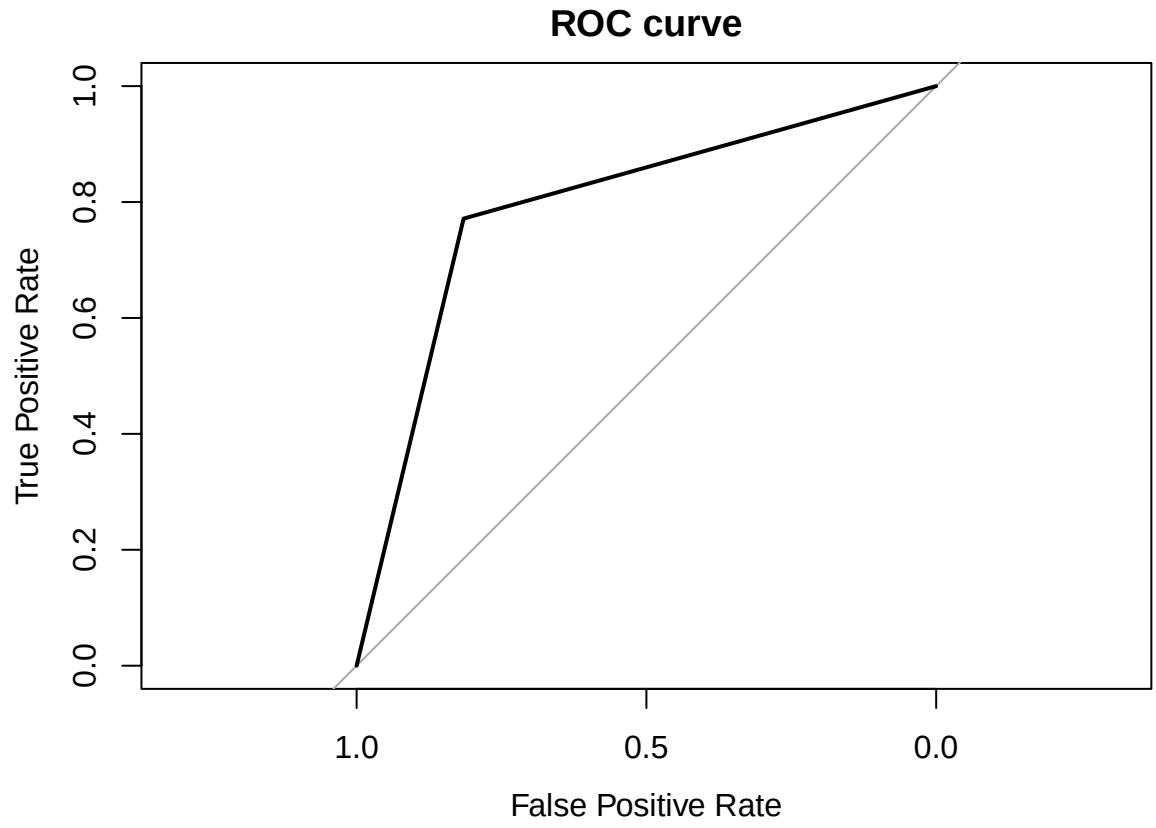
```
## Accuracy :  0.8
## False Positive Rate :  0.1846154
## False Negative Rate :  0.2285714
## precision :  0.8153846
```

```
## recall :  0.8688525
```

```
#tree_model_performance

## plotting ROC curve for the final pruned tree model
roc_curve_plot(as.numeric(tree_model_prediction))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
## AUC:  0.7934066
```

RANDOM FOREST

```
library(randomForest)
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'

## The following object is masked from 'package:dplyr':
##
##     combine

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
# fitting a random forest model to the training set
random_forest_model <- randomForest(test ~ . , data = train_dataset)

# using random forest model to make prediction on test data set
random_forest_prediction <- predict(random_forest_model,
                                    newdata = test_dataset)
## labeling predictions as negative and positive
random_forest_predictions<- ifelse(random_forest_prediction == 0, "Negative", "Positive")
# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")

random_forest_confusion_matrix <- table(actual = actual_testset,
                                predicted = random_forest_predictions)
print(random_forest_confusion_matrix)
```

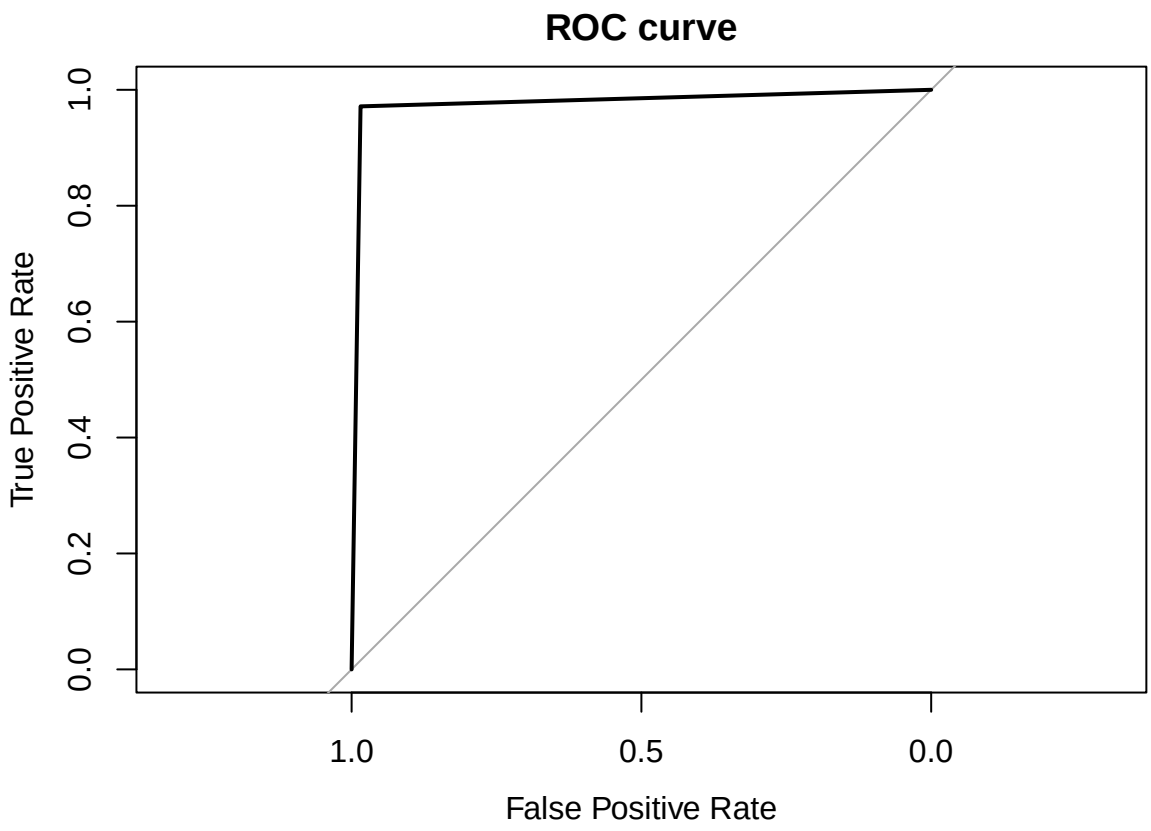
```
##           predicted
## actual     Negative Positive
##   Negative       64        1
##   Positive        1       34
```

```
### random forest Model Performance Evaluation
random_forest_performance <- evaluation_metrics(random_forest_confusion_matrix)
```

```
## Accuracy :  0.98
## False Positive Rate :  0.01538462
## False Negative Rate :  0.02857143
## precision :  0.9846154
## recall :  0.9846154
```

```
#random_forest_performance
roc_curve_plot(as.numeric(random_forest_prediction))
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```



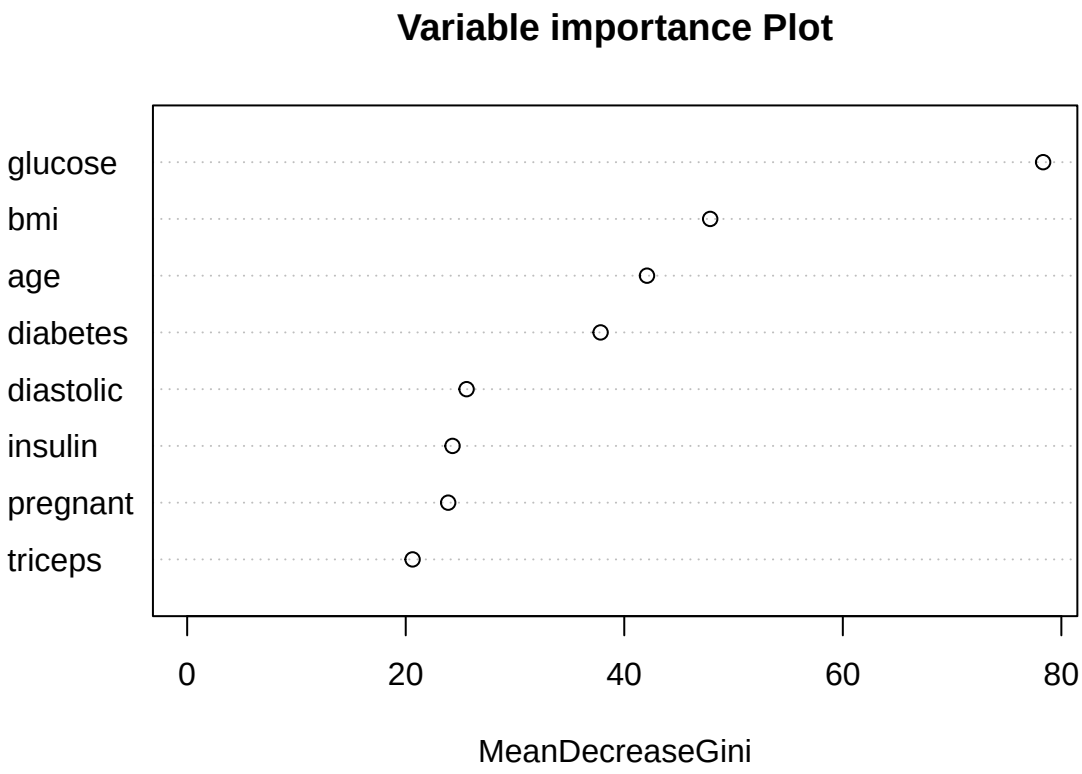

```
## AUC:  0.978022
```

```
# Get the variable importance
importance <- importance(random_forest_model)
importance
```

```
##           MeanDecreaseGini
## pregnant          23.88604
## glucose           78.33072
## diastolic         25.56960
## triceps           20.62764
## insulin           24.28230
## bmi               47.86265
## diabetes          37.83183
## age               42.07799
```

so, we can see that glucose is the most important predictor and triceps is least important predictor

```
# creating a plot showing variable importance
varImpPlot(random_forest_model, main ="Variable importance Plot")
```



K-NEAREST NEIGHBOURS (KNN MODEL)

```
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```
# Applying scaling to numeric feature predictors of training and test dataset
scaled_training_data <- scale(train_dataset[,numeric_columns_name])
scaled_test_data <- scale(test_dataset[,numeric_columns_name])
## training knn model with  optimal k value
knn_model <- train( scaled_training_data,
                    as.factor(train_dataset$test),
                    method = "knn",
                    tuneGrid = data.frame(k = 1:5))

knn_model_pred <- predict(knn_model, newdata = scaled_test_data)
```

```r
## labeling predictions as negative and positive
knn_predictions<- ifelse(knn_model_pred == 0, "Negative", "Positive")
# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")

knn_confusion_matrix <- table(actual = actual_testset,
                              predicted = knn_predictions)
print(knn_confusion_matrix)
```

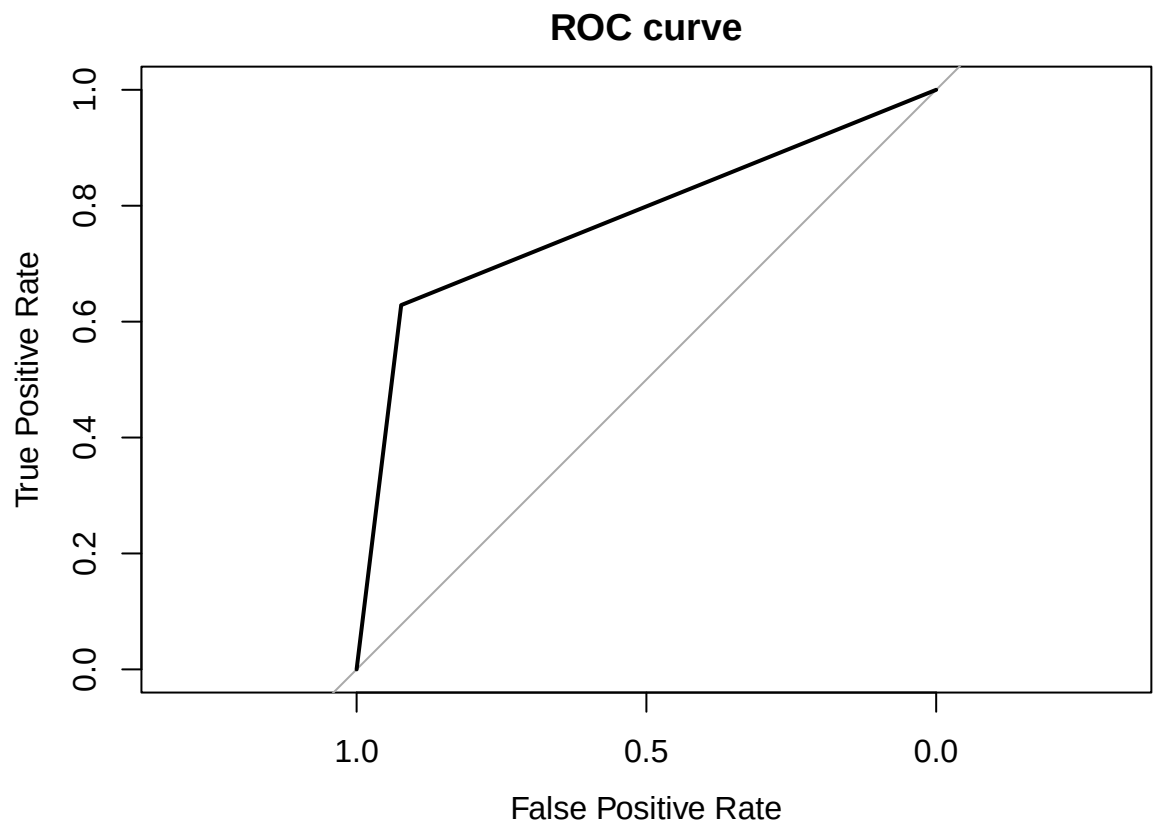
```
##           predicted
## actual     Negative Positive
##   Negative       60        5
##   Positive       13       22
```

```r
###KNN Model Performance Evaluation
knn_model_performance <- evaluation_metrics(knn_confusion_matrix)
```

```
## Accuracy :  0.82
## False Positive Rate :  0.07692308
## False Negative Rate :  0.3714286
## precision :  0.9230769
## recall :  0.8219178
```

```r
#knn_model_performance
roc_curve_plot(as.numeric(knn_model_pred))
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```

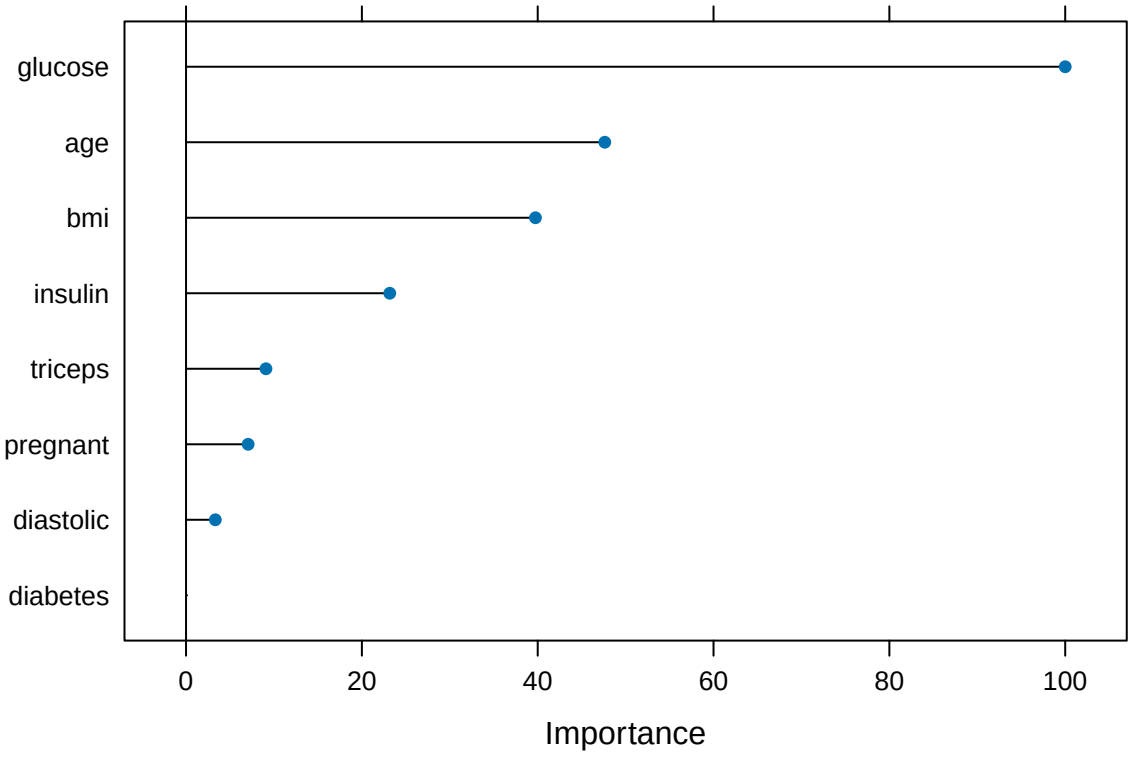



```
## AUC:  0.7758242
```

```
importance <-varImp(knn_model)
print(importance)
```

```
## ROC curve variable importance
##
##           Importance
## glucose      100.000
## age           47.632
## bmi           39.743
## insulin       23.174
## triceps        9.092
## pregnant       7.066
## diastolic      3.330
## diabetes       0.000
```

```r
plot(importance)
```



Linear Support Vector Machine(SVM)

```r
library(e1071)
# Applying scaling to numeric feature predictors of training and test dataset
scaled_training_data <- scale(train_dataset[,numeric_columns_name])
scaled_test_data <- scale(test_dataset[,numeric_columns_name])

# combining scaled numeric data with target variable
standardized_training_data <- cbind(as.data.frame(scaled_training_data),
                                    test = train_dataset$test)
set.seed(42)
# fit a linear Support vector machine model with 15-fold cross validation
linear_svm_model <-svm(test ~ ., data = standardized_training_data,
                       cross = 15, kernel ="linear")

# using linear Support vector machine model to make prediction on test data set
linear_svm_prediction <- predict(linear_svm_model,
                                    newdata = scaled_test_data)

## labeling predictions as negative and positive
linear_svm_predictions<- ifelse(linear_svm_prediction == 0, "Negative", "Positive")
```

```r
# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test == 0, "Negative","Positive")

linear_svm_confusion_matrix <- table(actual = actual_testset,
                                     predicted = linear_svm_predictions)
print(linear_svm_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       60        5
##   Positive       18       17
```

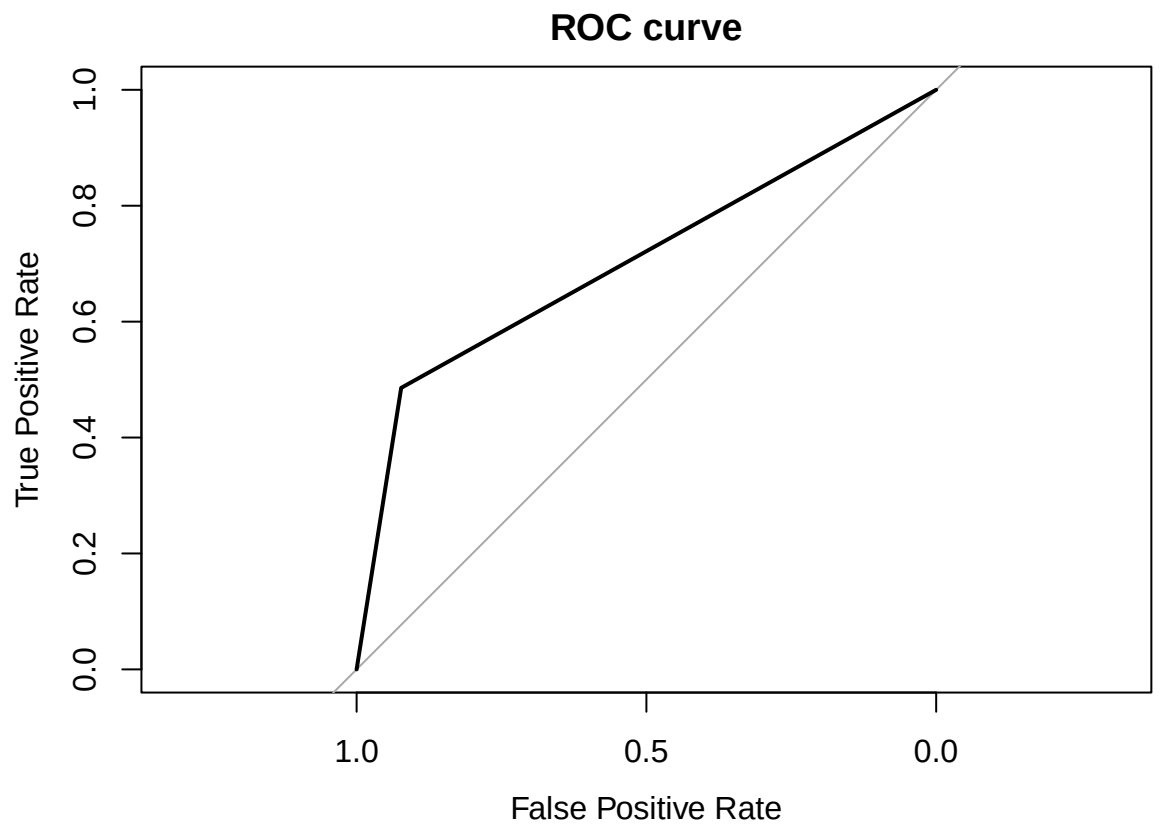
```r
### pruned linear svm Model Performance Evaluation
linear_svm_model_performance <- evaluation_metrics(linear_svm_confusion_matrix)
```

```
## Accuracy :  0.77
## False Positive Rate :  0.07692308
## False Negative Rate :  0.5142857
## precision :  0.9230769
## recall :  0.7692308
```

```r
#linear_svm_model_performance
roc_curve_plot(as.numeric(linear_svm_prediction))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
## AUC:  0.7043956
```

Radial Support Vector Machine(SVM)

```
## fit a radial svm model having 10-fold cross validation

set.seed(42)
radial_svm_model <- svm(test ~ ., data = standardized_training_data,
                        kernel = "radial", cross = 10)
# using radial Support vector machine model to make prediction on test data set
radial_svm_prediction <- predict(radial_svm_model,
                                 newdata = scaled_test_data)

## labeling predictions as negative and positive
radial_svm_predictions<- ifelse(radial_svm_prediction == 0, "Negative", "Positive")
# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")

radial_svm_confusion_matrix <- table(actual = actual_testset,
                                predicted = radial_svm_predictions)
print(radial_svm_confusion_matrix)
```

```
##           predicted
```

```
## actual     Negative Positive
##   Negative       62        3
##   Positive       14       21
```

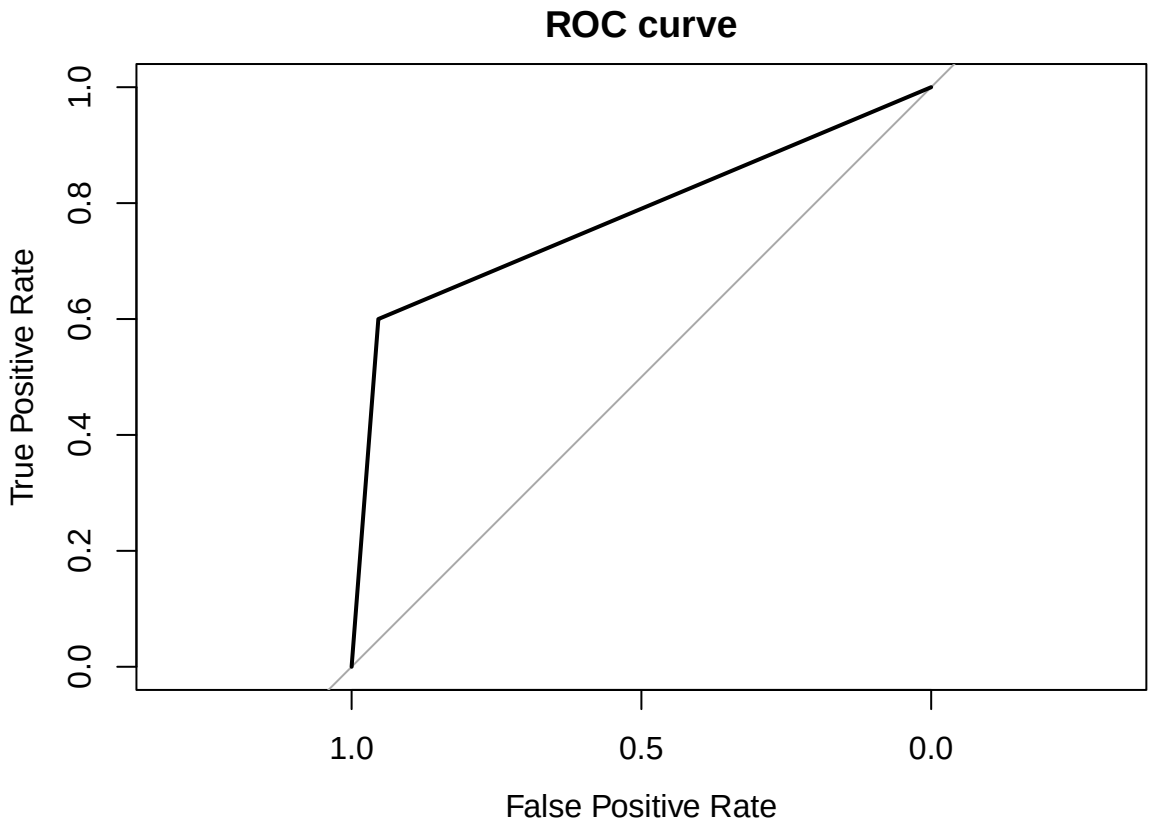
```
### pruned radial svm Model Performance Evaluation
radial_svm_model_performance <- evaluation_metrics(radial_svm_confusion_matrix)
```

```
## Accuracy :  0.83
## False Positive Rate :  0.04615385
## False Negative Rate :  0.4
## precision :  0.9538462
## recall :  0.8157895
```

```
#radial_svm_model_performance
roc_curve_plot(as.numeric(radial_svm_prediction))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
## AUC:  0.7769231
```

Neural Network with Unscaled Data

```
library(nnet)
```

```
##
## Attaching package: 'nnet'
```

```
## The following object is masked from 'package:mgcv':
##
##     multinom
```

```
set.seed(84)
## Fitting a neural network with two hidden layer
neural_net_model <- nnet(test ~ ., data = train_dataset,
                            size = 2 , linear.outout =TRUE)
```

```
## # weights:  21
## initial  value 448.284447
## iter  10 value 422.793585
## iter  20 value 403.637229
## iter  30 value 402.265250
## iter  40 value 402.105436
## iter  50 value 400.627970
## iter  60 value 392.790818
## iter  70 value 392.636066
## iter  80 value 392.634064
## iter  90 value 391.124470
## iter 100 value 389.423728
## final  value 389.423728
## stopped after 100 iterations
```

```
neural_net_prediction <- predict(neural_net_model, newdata = test_dataset,
                                 type ="class")
#neural_net_prediction
## labeling predictions as negative and positive
neural_net_predictions<- ifelse(neural_net_prediction == 0, "Negative", "Positive")
# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")

neural_net_confusion_matrix <- table(actual = actual_testset,
                                 predicted = neural_net_predictions)
print(neural_net_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       44       21
##   Positive       17       18
```

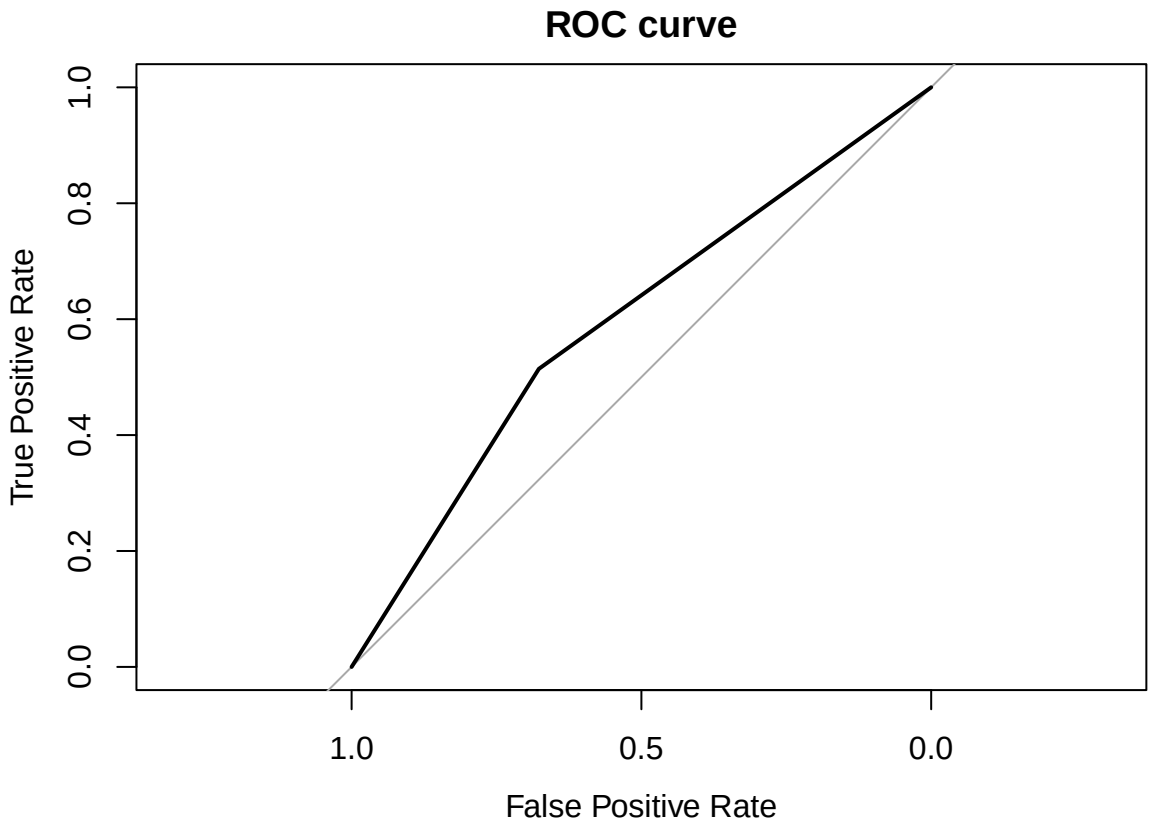
```
### Neural net Model Performance Evaluation
neural_net_model_performance <- evaluation_metrics(neural_net_confusion_matrix)
```

```
## Accuracy :  0.62
## False Positive Rate :  0.3230769
## False Negative Rate :  0.4857143
## precision :  0.6769231
## recall :  0.7213115
```

```
#neural_net_model_performance
roc_curve_plot(as.numeric(neural_net_prediction))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
## AUC:  0.5956044
```

Neural Network with Scaled Data

```
# Applying scaling to numeric feature predictors of training and test dataset
scaled_training_data <- scale(train_dataset[,numeric_columns_name])
scaled_test_data <- scale(test_dataset[,numeric_columns_name])

# combining scaled numeric data with target variable
```

```r
standardized_training_data <- cbind(as.data.frame(scaled_training_data),
                                    test = train_dataset$test)
set.seed(89)
## fit neural net model with standardized data with 10 hidden layers
standardized_neural_net_model <- nnet(test ~ ., data = standardized_training_data,
                                      size = 10, linear.output =TRUE)
```

```
## # weights:  101
## initial  value 486.267986
## iter  10 value 291.067399
## iter  20 value 241.879680
## iter  30 value 208.227353
## iter  40 value 189.846625
## iter  50 value 178.770593
## iter  60 value 165.045094
## iter  70 value 152.771306
## iter  80 value 143.936879
## iter  90 value 137.524708
## iter 100 value 135.946509
## final  value 135.946509
## stopped after 100 iterations
```

```r
standardized_neural_net_prediction <- predict(standardized_neural_net_model,
                                              newdata = scaled_test_data,
                                              type ="class")
## labeling predictions as negative and positive
standardized_neural_net_predictions<- ifelse(standardized_neural_net_prediction == 0,
                                             "Negative", "Positive")
# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")

neural_net_confusion_matrix <- table(actual = actual_testset,
                                     predicted = standardized_neural_net_predictions)
print(neural_net_confusion_matrix)
```

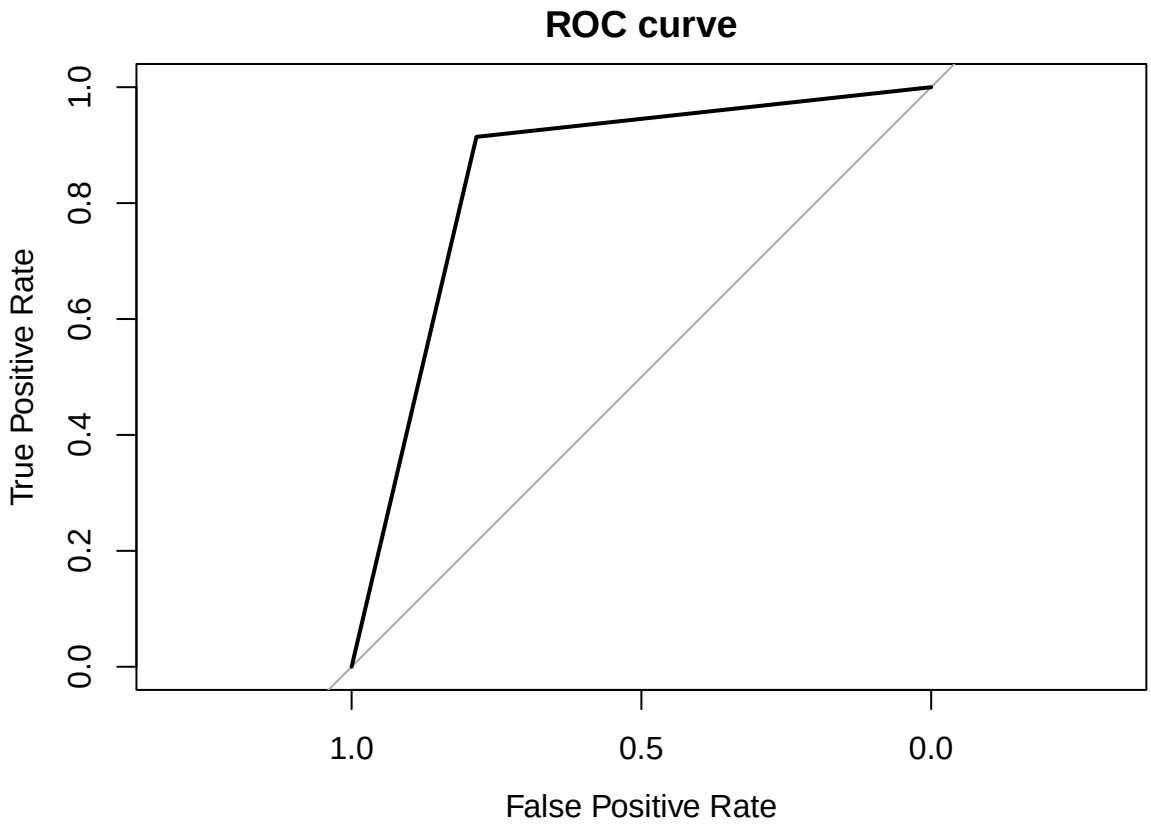
```
##           predicted
## actual     Negative Positive
##   Negative       51       14
##   Positive        3       32
```

```r
### Neural net Model Performance Evaluation
neural_net_model_performance <- evaluation_metrics(neural_net_confusion_matrix)
```

```
## Accuracy :  0.83
## False Positive Rate :  0.2153846
## False Negative Rate :  0.08571429
## precision :  0.7846154
## recall :  0.9444444
```

```r
#neural_net_model_performance
roc_curve_plot(as.numeric(standardized_neural_net_prediction))
```

```
## Setting levels: control = 0, case = 1

## Setting direction: controls < cases
```




```
## AUC:  0.8494505
```

Two Level Ensemble Approach

Implementing two level approach, where in the first level GLM,GAM, SVM, random forest are trained independently and the output of these models are provided to neural network for training in the second level.

```
# Fit GLM, GAM, SVM and Random Forest
glm_model <- glm(test ~ . , family = "binomial", data = train_dataset)
additive_model <- gam(test ~ s(pregnant) + s(glucose)+ s(diastolic) + s(triceps)
                      + s(insulin) + s(bmi) + s(diabetes) + s(age),
                      data = train_dataset, family = binomial)
linear_svm_model <-svm(test ~ ., data = train_dataset,
                       cross = 15, kernel ="linear")
random_forest_model <- randomForest(test ~ . , data = train_dataset)
knn_model <- train( scaled_training_data,
                    as.factor(train_dataset$test),
                    method = "knn",
                    tuneGrid = data.frame(k = 1:5))
```

```r
# get predictions
glm_model_pred <- predict(glm_model, type = "response", newdata = test_dataset)
additive_model_pred <- predict(additive_model, type ="response", newdata = test_dataset)
linear_svm_model_pred <- predict(random_forest_model, newdata = test_dataset)
random_forest_model_pred <-predict(linear_svm_model, newdata = test_dataset)
knn_model_pred <- predict(knn_model, newdata = scaled_test_data)

## converting the probabilities into test predictions considering threshold as 0.5
glm_model_pred <- ifelse(glm_model_pred >= 0.5, 1, 0)
additive_model_pred <- ifelse(additive_model_pred >= 0.5, 1, 0)
```

```r
merged_data <- data.frame(glm = glm_model_pred,
                          gam = additive_model_pred,
                          svm = linear_svm_model_pred,
                          random_forest = random_forest_model_pred,
                          knn = knn_model_pred,
                          actual_test = test_dataset$test)

#merged_data
neural_net_model <- nnet(actual_test ~ .,
                         data = merged_data,
                         size = 5,trace = FALSE)
neural_net_pred <- predict(neural_net_model, newdata = merged_data,
                           type ="class")
```

```r
## labeling predictions as negative and positive
neural_net_predicted<- ifelse(neural_net_pred == 0, "Negative", "Positive")
# labeling test dataset target column as negative and positive
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")

neural_net_confusion_matrix <- table(actual = actual_testset,
                                     predicted = neural_net_predicted)
print(neural_net_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       65        0
##   Positive        1       34
```

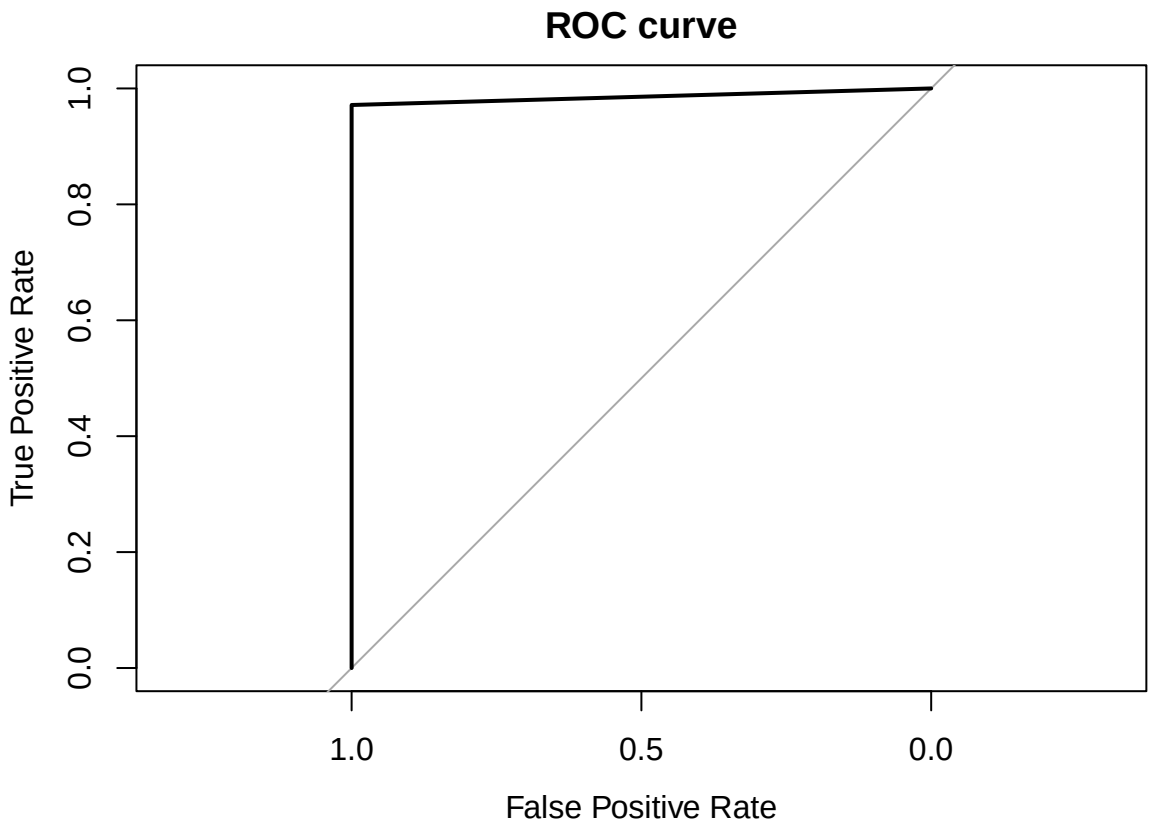
```r
### two_level_ensemble_model_performance Evaluation
two_level_ensemble_model_performance <- evaluation_metrics(neural_net_confusion_matrix)
```

```
## Accuracy :  0.99
## False Positive Rate :  0
## False Negative Rate :  0.02857143
## precision :  1
## recall :  0.9848485
```

```r
#two_level_ensemble_model_performance
roc_curve_plot(as.numeric(neural_net_pred))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
## AUC:  0.9857143
```

Ensemble approach of combining different model predictions (Majority Voting)

```
# Fit GLM, GAM, SVM and Random Forest
glm_model <- glm(test ~ . , family = "binomial", data = train_dataset)
additive_model <- gam(test ~ s(pregnant) + s(glucose)+ s(diastolic) + s(triceps)
                      + s(insulin) + s(bmi) + s(diabetes) + s(age),
                      data = train_dataset, family = binomial)
linear_svm_model <-svm(test ~ ., data = train_dataset,
                       cross = 15, kernel ="linear")
random_forest_model <- randomForest(test ~ . , data = train_dataset)
tree_model <- rpart( test ~ ., data = train_dataset)
knn_model <- train( scaled_training_data,
                    as.factor(train_dataset$test),
                    method = "knn",
                    tuneGrid = data.frame(k = 1:5))
standardized_neural_net_model <- nnet(test ~ ., data = standardized_training_data,
                                      size = 10, linear.output =TRUE,trace = FALSE)
```

```r
# get predictions
glm_model_pred <- predict(glm_model, type = "response", newdata = test_dataset)
additive_model_pred <- predict(additive_model, type ="response", newdata = test_dataset)
linear_svm_model_pred <- predict(random_forest_model, newdata = test_dataset)
random_forest_model_pred <-predict(linear_svm_model, newdata = test_dataset)
tree_model_pred <- predict(final_tree_model, newdata = test_dataset,type ="class")
knn_model_pred <- predict(knn_model, newdata = scaled_test_data)
neural_net_model_pred <-predict(standardized_neural_net_model, newdata = scaled_test_data,
                                          type ="class")

## converting the probabilities into test predictions considering threshold as 0.5
glm_model_pred <- ifelse(glm_model_pred >= 0.5, 1, 0)
additive_model_pred <- ifelse(additive_model_pred >= 0.5, 1, 0)
```

```r
predictions <- data.frame(glm = glm_model_pred,
                          gam = additive_model_pred,
                          svm = linear_svm_model_pred,
                          random_forest = random_forest_model_pred,
                          tree = tree_model_pred,
                          knn = knn_model_pred,
                          neural_net = neural_net_model_pred)

combined_pred <- as.factor(apply(predictions, 1 ,function(row){
  ifelse(sum(row == "0") > sum(row == "1"), 0, 1)
}))

merged_data <- cbind(predictions, combined_prediction = combined_pred)
merged_data <- cbind(merged_data, actual =test_dataset$test)
```

```r
## labeling predictions as negative and positive
combined_predicted_result<- ifelse(combined_pred == 0, "Negative", "Positive")
# labeling test data set target column as negative and positive
actual_testset <- ifelse(test_dataset$test ==0, "Negative","Positive")

combined_model_confusion_matrix <- table(actual = actual_testset,
                                 predicted = combined_predicted_result)
print(combined_model_confusion_matrix)
```

```
##           predicted
## actual     Negative Positive
##   Negative       60        5
##   Positive       13       22
```

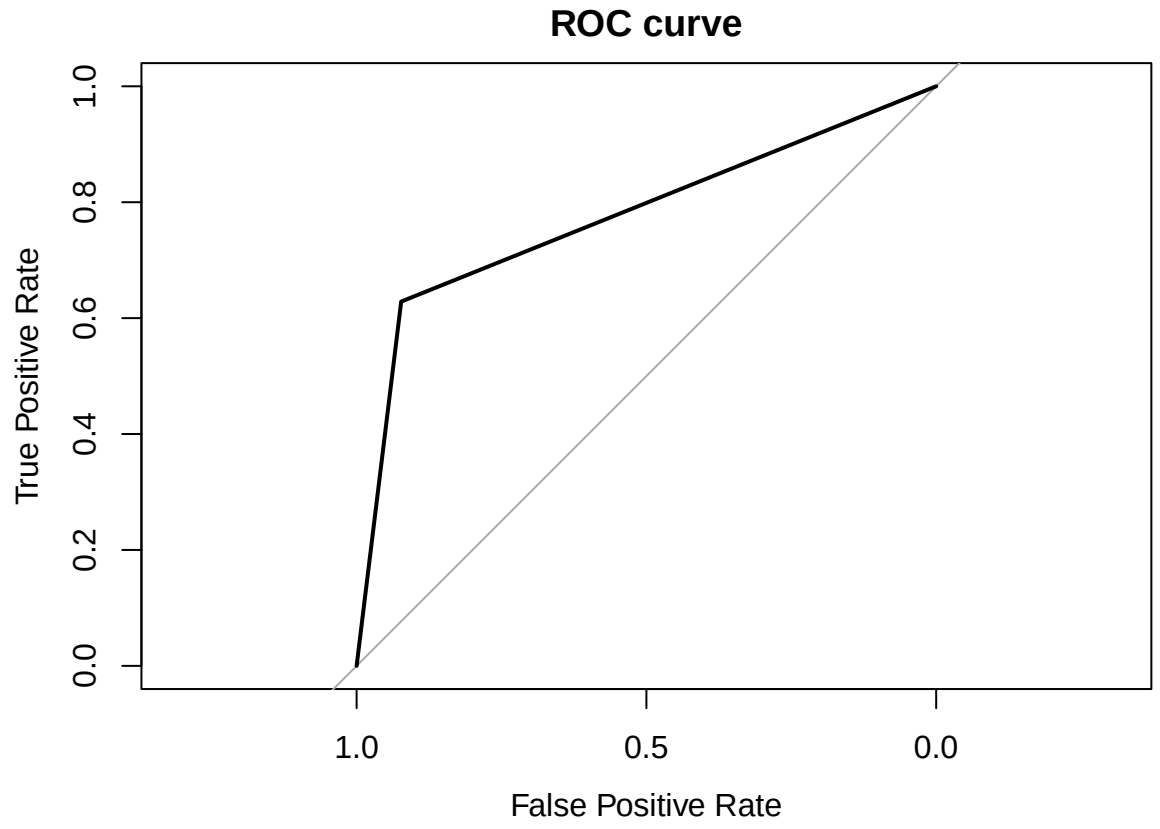
```r
### Neural net Model Performance Evaluation
combined_model_performance <- evaluation_metrics(combined_model_confusion_matrix)
```

```
## Accuracy :  0.82
## False Positive Rate :  0.07692308
## False Negative Rate :  0.3714286
## precision :  0.9230769
## recall :  0.8219178
```

```r
#combined_model_performance
roc_curve_plot(as.numeric(combined_pred))
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```




```
## AUC:  0.7758242
```

Comparative Accuracy analysis

```r
## comparative Plot of performance metric for different models

classification_models <- c("Generalized Linear Model = 0.76",
                           "Generalized Additive Model = 0.78", "Linear SVM = 0.77",
                           "Radial SVM =0.83","Tree model= 0.8",
                           "Random Forest=0.98", "K Nearest Neighbour = 0.82",
                           "Neural Network with scaled data = 0.83",
                           "Two level Ensemble Approach = 0.99",
                           "combined model majority voting = 0.82")

accuracy_val <- c(glm_model_performance$accuracy, gam_model_performance$accuracy,
```

```
                  linear_svm_model_performance$accuracy,radial_svm_model_performance$accuracy,
                  tree_model_performance$accuracy,random_forest_performance$accuracy,
                  knn_model_performance$accuracy,neural_net_model_performance$accuracy,
                  two_level_ensemble_model_performance$accuracy,
                  combined_model_performance$accuracy)

df <- data.frame(models = classification_models,
                 accuracy = accuracy_val)
colors <- c("tan","lightseagreen","olivedrab","purple","maroon4","grey","brown",
            "violet","lightblue","lightyellow")
barplot(df$accuracy, col = colors, ylim =c(0,1), xlim = c(0,21),
        main = "Comparison of Accuracy of different models",ylab = "Accuracy")

legend("bottomright", legend = df$models,fill = colors, cex = 0.65)
```