PIMA INDIANS DIABETES

DATASET

AMEER HAMZA IFTIKHAR
SALVATORE CALABRESE

INTRODUCTION

About the data

The Pima Indians Diabetes dataset is a well-known dataset used in medical and scientific research to study the prevalence of diabetes among the Pima Indian population. This community, native to Arizona and northern Mexico, has experienced a high rate of diabetes in recent decades.

The dataset contains information on 768 Pima Indian patients, including factors such as age, BMI, blood pressure, and family history of diabetes, as well as the outcome of the diabetes test (positive or negative). The dataset was collected by the National Institute of Diabetes and Digestive and Kidney Diseases and made publicly available for research purposes.

Studies using this dataset have contributed to improving our understanding of diabetes and its causes, as well as developing predictive models to identify patients at high risk of developing the disease.

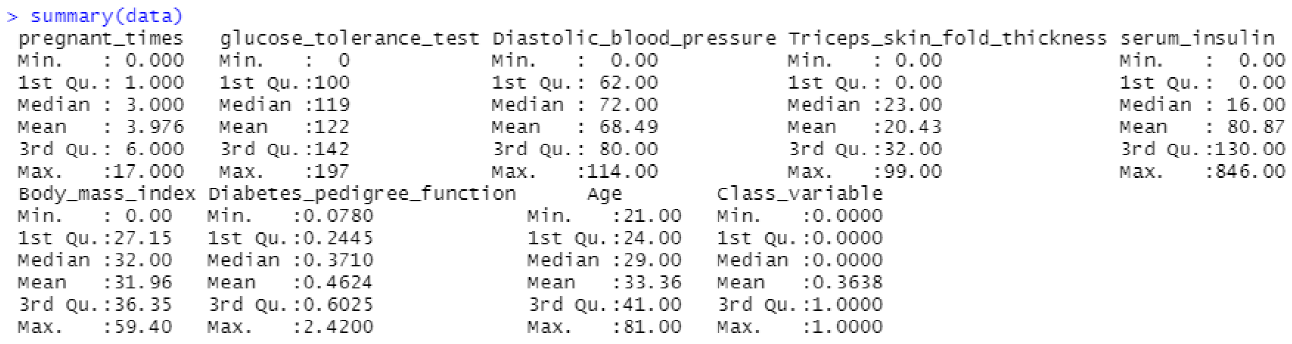
The Pima Indians Diabetes dataset contains the following variables:

- **pregnant_times**: Number of times pregnant
- **glucose_tolerance_test**: Plasma glucose concentration a 2 hours in an oral glucose tolerance test
- **Diastolic_blood_pressure**: Diastolic blood pressure (mm Hg)
- **Triceps_skin_fold_thickness**: Triceps skin fold thickness (mm)
- **serum_insulin**: 2-Hour serum insulin (mu U/ml)
- **Body_mass_index**: BMI (weight in kg/(height in m)^2)
- **Diabetes_pedigree_function**: Diabetes pedigree function
- **Age**: Age in years
- **Class_variable**: Class variable (0 or 1) where 0 means non-diabetic and 1 means diabetic.

These variables can be used to explore various relationships between the factors and the likelihood of diabetes, as well as to develop predictive models for diabetes diagnosis and prevention.

Inspect the data

First, for simplicity, we change the name of the loaded dataset in “data”.
Now we can start to inspect the data.



```
> summary(data)
 pregnant_times   glucose_tolerance_test Diastolic_blood_pressure Triceps_skin_fold_thickness serum_insulin
 Min.   : 0.000   Min.   :  0            Min.   :  0.00           Min.   : 0.00               Min.   :  0.00
 1st Qu.: 1.000   1st Qu.:100            1st Qu.: 62.00           1st Qu.: 0.00               1st Qu.:  0.00
 Median : 3.000   Median :119            Median : 72.00           Median :23.00               Median : 16.00
 Mean   : 3.976   Mean   :122            Mean   : 68.49           Mean   :20.43               Mean   : 80.87
 3rd Qu.: 6.000   3rd Qu.:142            3rd Qu.: 80.00           3rd Qu.:32.00               3rd Qu.:130.00
 Max.   :17.000   Max.   :197            Max.   :114.00           Max.   :99.00               Max.   :846.00
 Body_mass_index Diabetes_pedigree_function      Age        Class_variable
 Min.   : 0.00   Min.   :0.0780             Min.   :21.00   Min.   :0.0000
 1st Qu.:27.15   1st Qu.:0.2445             1st Qu.:24.00   1st Qu.:0.0000
 Median :32.00   Median :0.3710             Median :29.00   Median :0.0000
 Mean   :31.96   Mean   :0.4624             Mean   :33.36   Mean   :0.3638
 3rd Qu.:36.35   3rd Qu.:0.6025             3rd Qu.:41.00   3rd Qu.:1.0000
 Max.   :59.40   Max.   :2.4200             Max.   :81.00   Max.   :1.0000
```

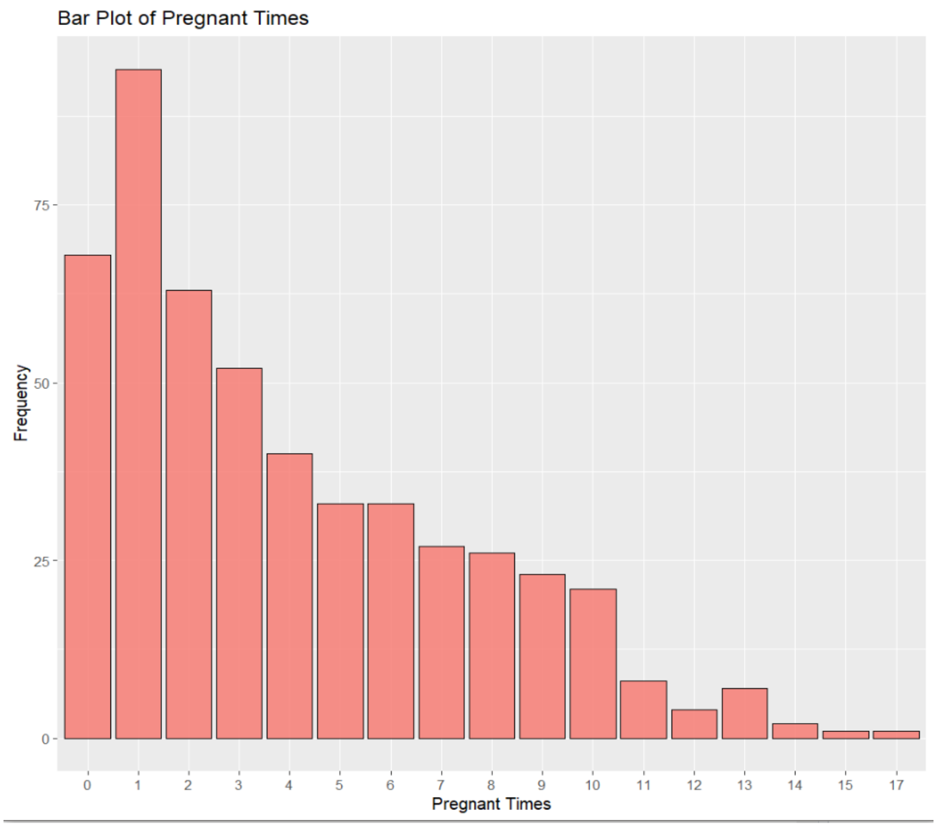
```
> str(data)
'data.frame':	503 obs. of  9 variables:
 $ pregnant_times              : int  6 1 8 1 5 3 10 8 4 10 ...
 $ glucose_tolerance_test      : int  148 85 183 89 116 78 115 125 110 139 ...
 $ Diastolic_blood_pressure    : int  72 66 64 66 74 50 0 96 92 80 ...
 $ Triceps_skin_fold_thickness: int  35 29 0 23 0 32 0 0 0 0 ...
 $ serum_insulin               : int  0 0 0 94 0 88 0 0 0 0 ...
 $ Body_mass_index             : num  33.6 26.6 23.3 28.1 25.6 31 35.3 0 37.6 27.1 ...
 $ Diabetes_pedigree_function : num  0.627 0.351 0.672 0.167 0.201 ...
 $ Age                         : int  50 31 32 21 30 26 29 54 30 57 ...
 $ Class_variable              : int  1 0 1 0 0 1 0 1 0 0 ...
```

EXPLORATORY DATA ANALYSIS

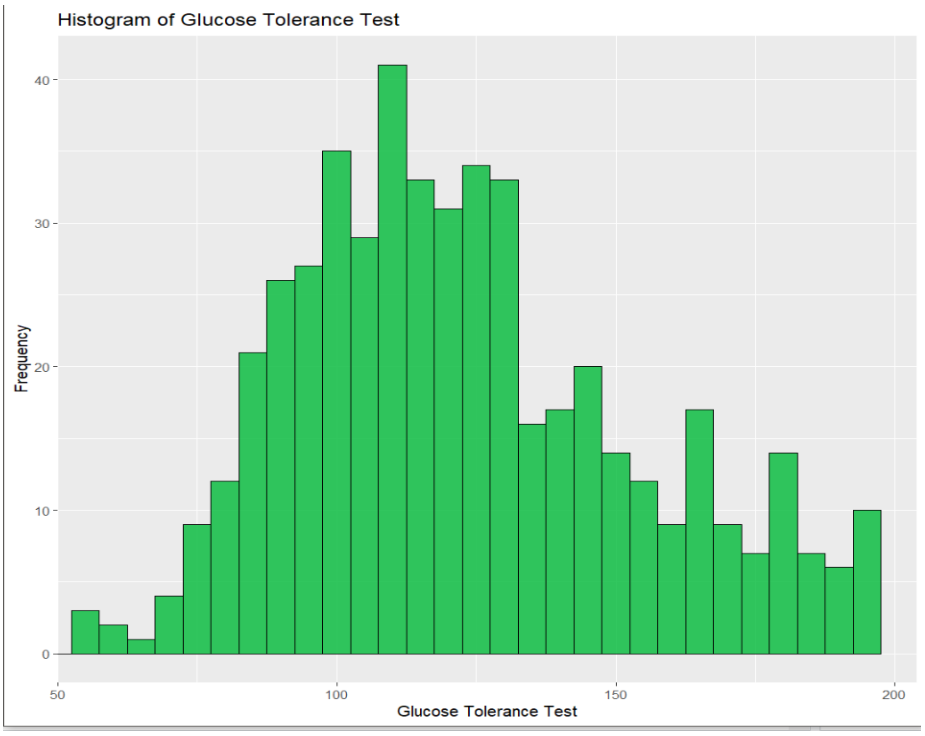
Visualize the data

In this paragraph we use “ggplot2” package to create plot and have a first view of the distribution of our variables and look for outliers.

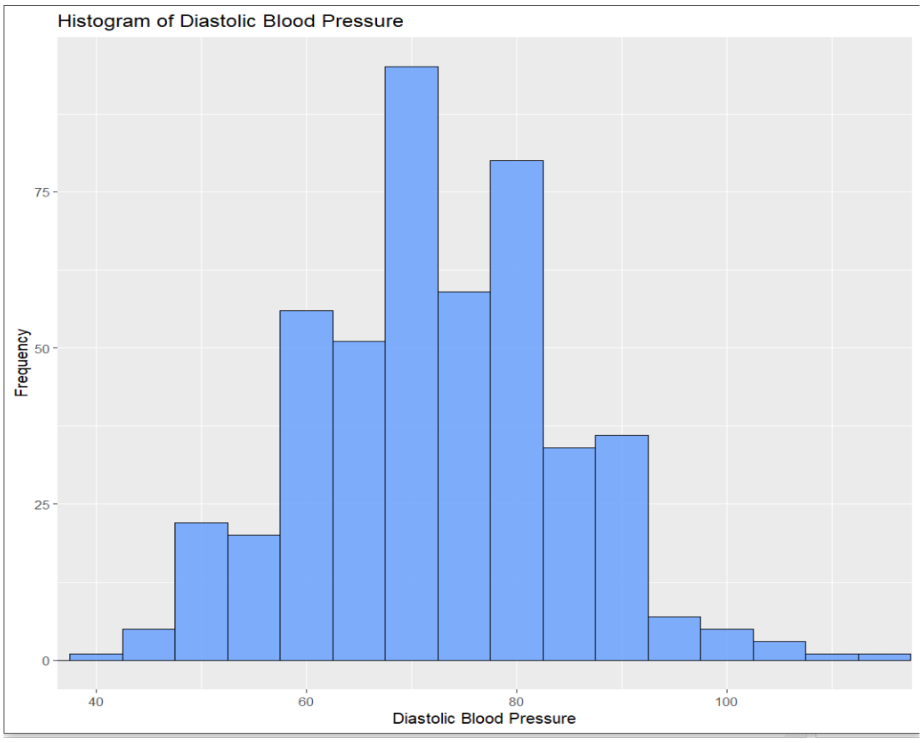
```
# Variable: pregnant_times
ggplot(data, aes(x = as.factor(pregnant_times))) +
  geom_bar(fill = "#F8766D", color = "black", alpha = 0.8) +
  labs(x = "Pregnant Times", y = "Frequency", title = "Bar Plot of Pregnant Times")
```



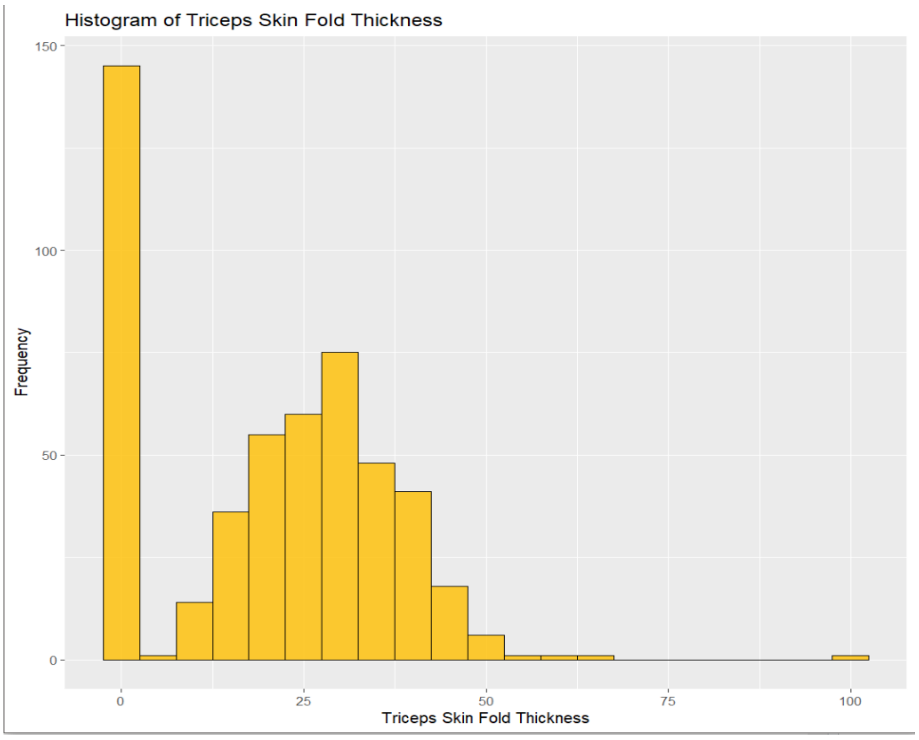
```
# Variable: glucose_tolerance_test
ggplot(data, aes(x = glucose_tolerance_test)) +
  geom_histogram(color = "black", fill = "#00BA38", alpha = 0.8, binwidth = 5) +
  labs(x = "Glucose Tolerance Test", y = "Frequency", title = "Histogram of Glucose Tolerance Test") +
  coord_cartesian(xlim = c(57, max(data$glucose_tolerance_test)))
```



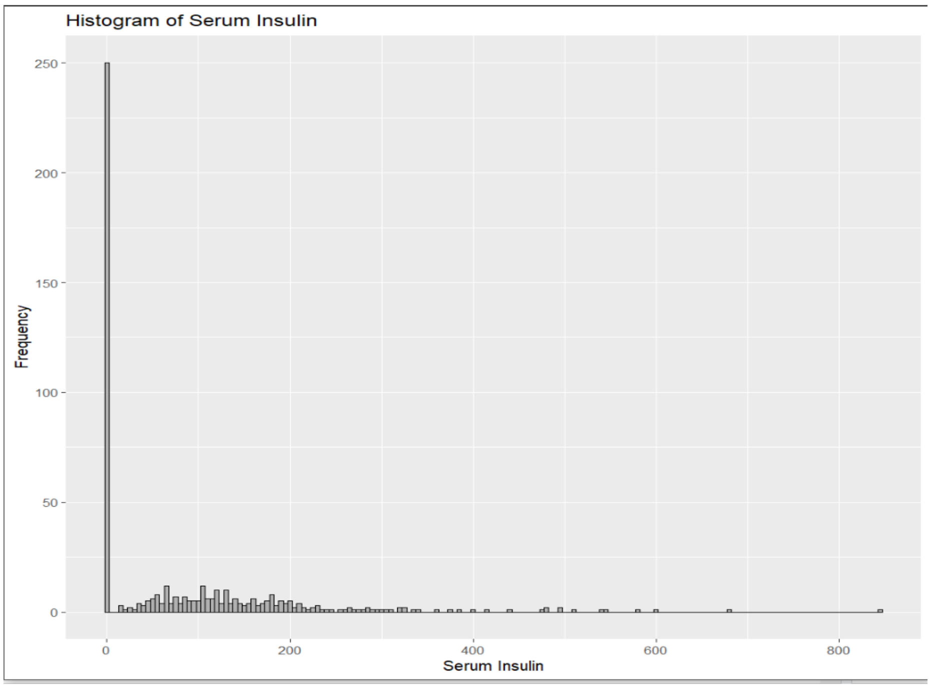
```
# Variable: Diastolic_blood_pressure
ggplot(data, aes(x = Diastolic_blood_pressure)) +
  geom_histogram(color = "black", fill = "#619CFF", alpha = 0.8, binwidth = 5) +
  labs(x = "Diastolic Blood Pressure", y = "Frequency", title = "Histogram of Diastolic Blood Pressure") +
  coord_cartesian(xlim = c(40, max(data$Diastolic_blood_pressure)))
```



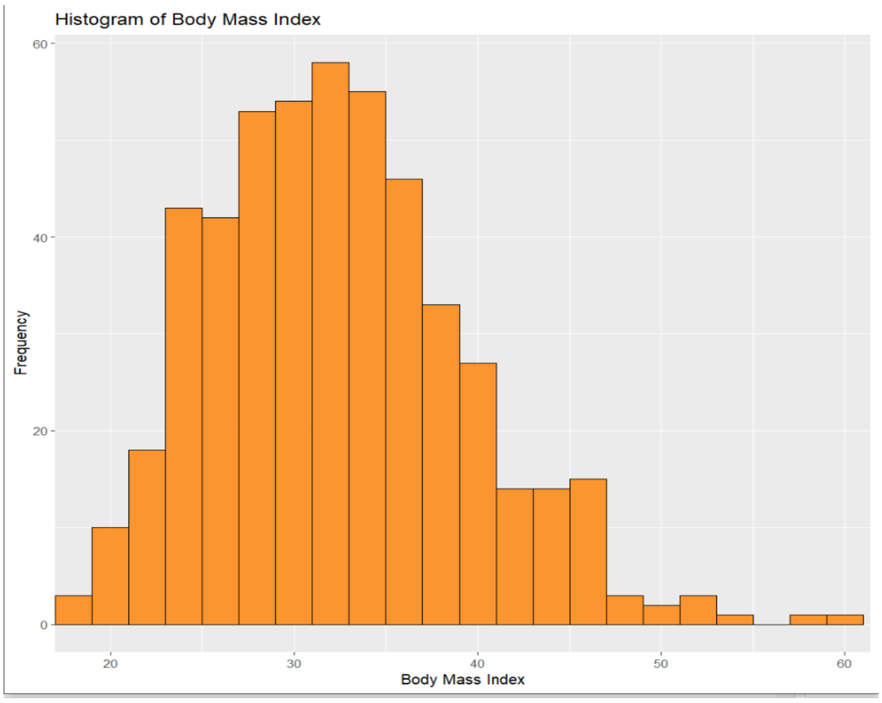
```r
# Variable: Triceps_skin_fold_thickness
ggplot(data, aes(x = Triceps_skin_fold_thickness)) +
  geom_histogram(color = "black", fill = "#FFBF00", alpha = 0.8, binwidth = 5) +
  labs(x = "Triceps Skin Fold Thickness", y = "Frequency", title = "Histogram of Triceps Skin Fold Thickness")
```



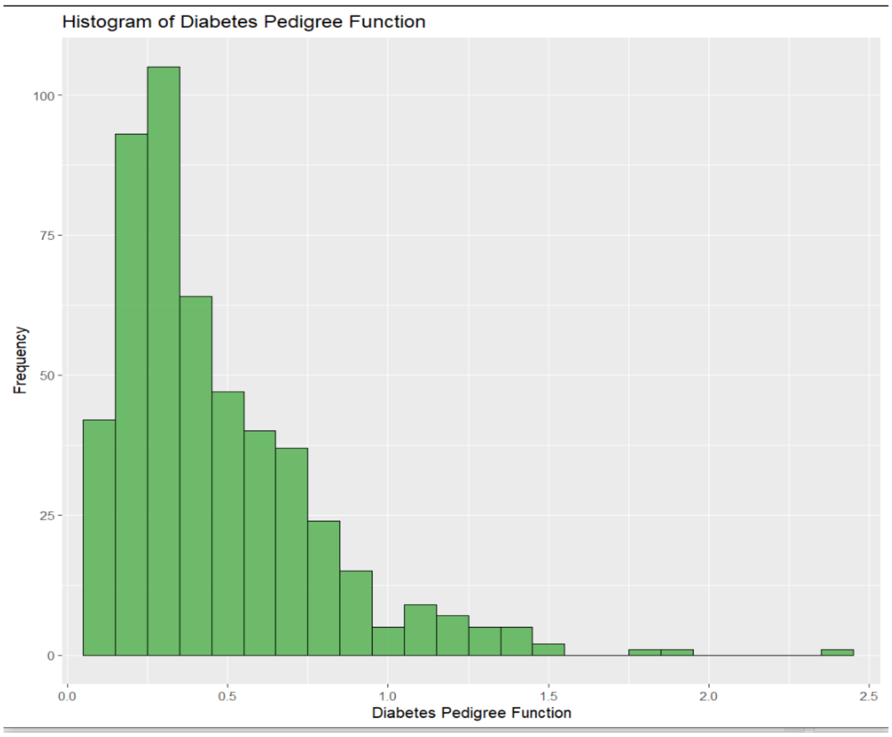
```r
# Variable: serum_insulin
ggplot(data, aes(x = serum_insulin)) +
  geom_histogram(color = "black", fill = "#A2A2A2", alpha = 0.8, binwidth = 5) +
  labs(x = "Serum Insulin", y = "Frequency", title = "Histogram of Serum Insulin")
```



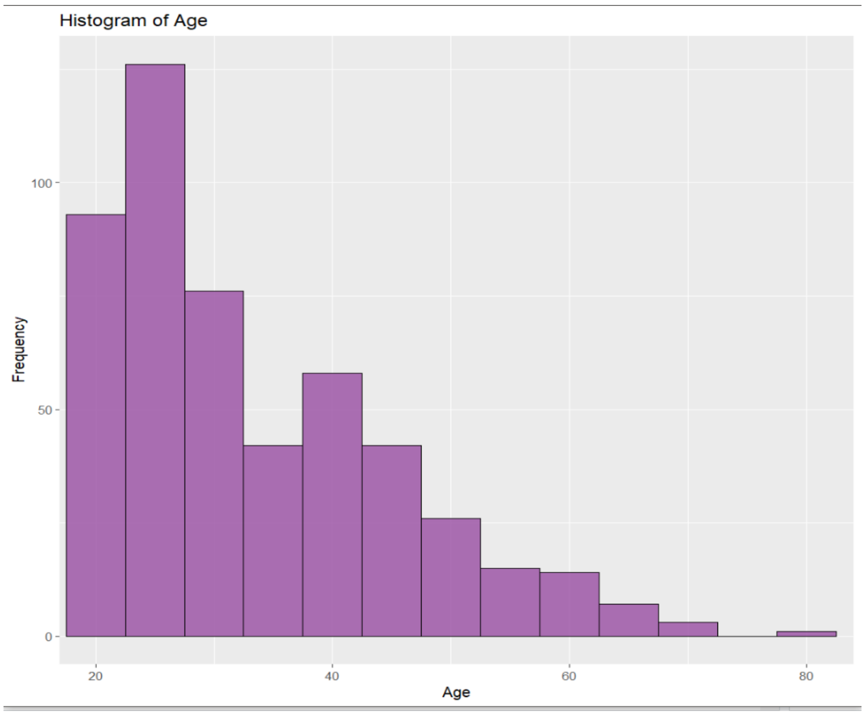
```
# Variable: Body_mass_index
ggplot(data, aes(x = Body_mass_index)) +
  geom_histogram(color = "black", fill = "#FF7F00", alpha = 0.8, binwidth = 2) +
  labs(x = "Body Mass Index", y = "Frequency", title = "Histogram of Body Mass Index") +
  coord_cartesian(xlim = c(19, max(data$Body_mass_index)))
```



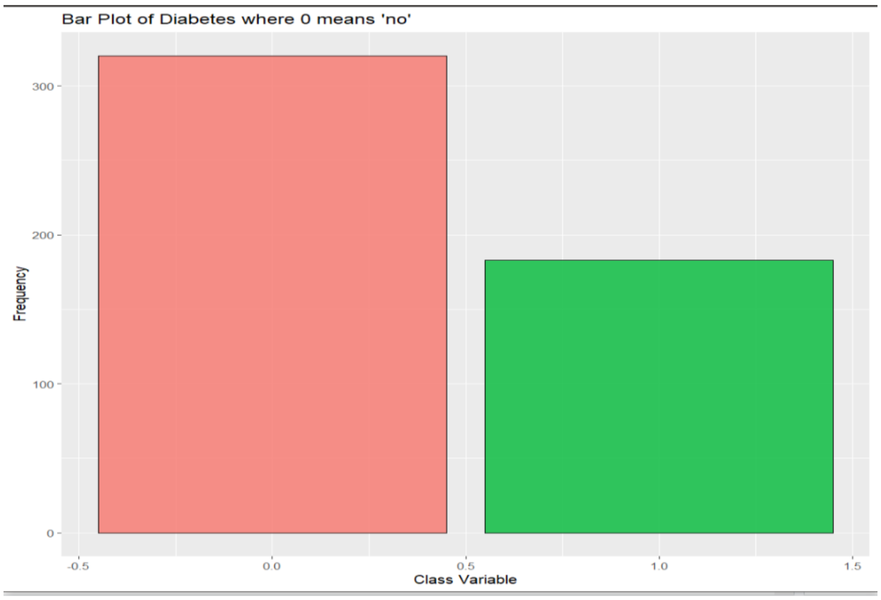
```
# Variable: Diabetes_pedigree_function
ggplot(data, aes(x = Diabetes_pedigree_function)) +
  geom_histogram(color = "black", fill = "#4DAF4A", alpha = 0.8, binwidth = 0.1) +
  labs(x = "Diabetes Pedigree Function", y = "Frequency", title = "Histogram of Diabetes Pedigree Function") +
  coord_cartesian(xlim = c(0.1, max(data$Diabetes_pedigree_function)))
```



```r
# Variable: Age
ggplot(data, aes(x = Age)) +
  geom_histogram(color = "black", fill = "#984EA3", alpha = 0.8, binwidth = 5) +
  labs(x = "Age", y = "Frequency", title = "Histogram of Age") +
  coord_cartesian(xlim = c(20, max(data$Age)))
```

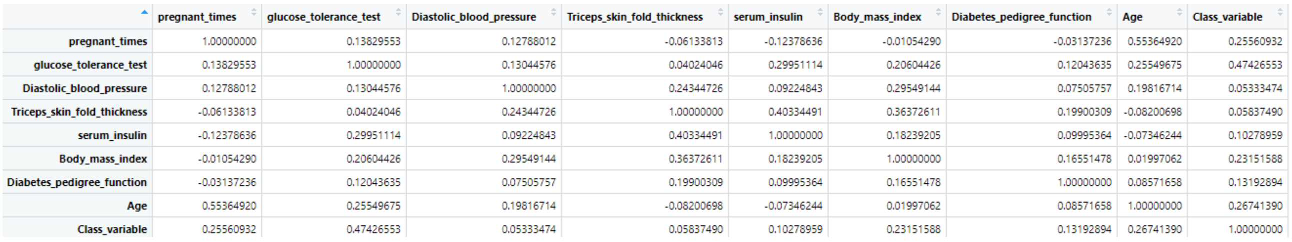


```r
# Variable: Class_variable
ggplot(data, aes(x = Class_variable)) +
  geom_bar(fill = c("#F8766D", "#00BA38"), color = "black", alpha = 0.8) +
  labs(x = "Class Variable", y = "Frequency", title = "Bar Plot of Diabetes where 0 means 'no'")
```



Relationships between variables

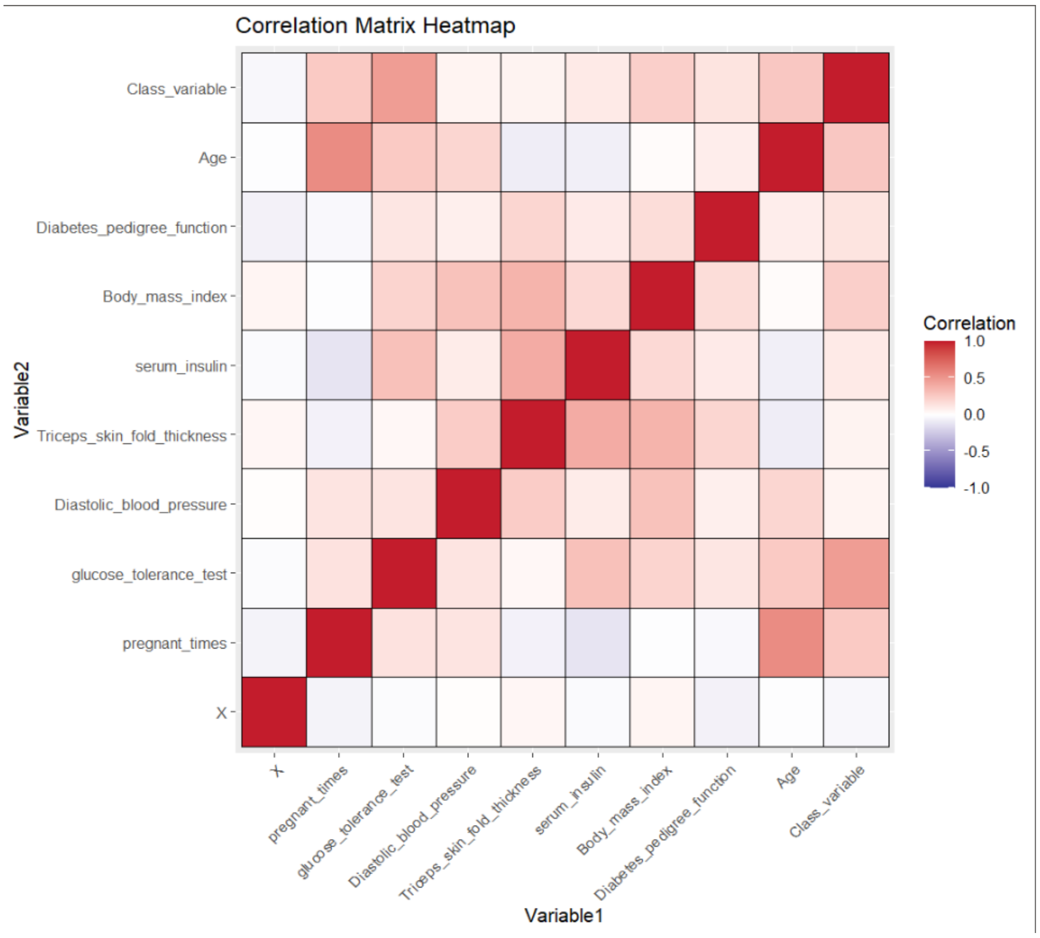
```r
# Compute the correlation matrix
correlation_matrix <- cor(data)
```



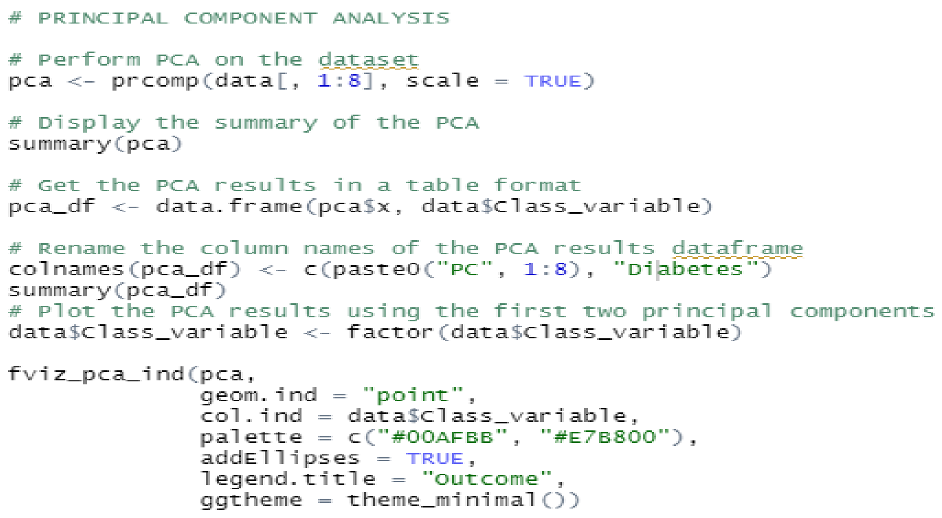
	pregnant_times	glucose_tolerance_test	Diastolic_blood_pressure	Triceps_skin_fold_thickness	serum_insulin	Body_mass_index	Diabetes_pedigree_function	Age	Class_variable
pregnant_times	1.00000000	0.13829553	0.12788012	-0.06133813	-0.12378636	-0.01054290	-0.03137236	0.55364920	0.25560932
glucose_tolerance_test	0.13829553	1.00000000	0.13044576	0.04024046	0.29951114	0.20604426	0.12043635	0.25549675	0.47426553
Diastolic_blood_pressure	0.12788012	0.13044576	1.00000000	0.24344726	0.09224843	0.29549144	0.07505757	0.19816714	0.05333474
Triceps_skin_fold_thickness	-0.06133813	0.04024046	0.24344726	1.00000000	0.40334491	0.36372611	0.19900309	-0.08200698	0.05837490
serum_insulin	-0.12378636	0.29951114	0.09224843	0.40334491	1.00000000	0.18239205	0.09995364	-0.07346244	0.10278959
Body_mass_index	-0.01054290	0.20604426	0.29549144	0.36372611	0.18239205	1.00000000	0.16551478	0.01997062	0.23151588
Diabetes_pedigree_function	-0.03137236	0.12043635	0.07505757	0.19900309	0.09995364	0.16551478	1.00000000	0.08571658	0.13192894
Age	0.55364920	0.25549675	0.19816714	-0.08200698	-0.07346244	0.01997062	0.08571658	1.00000000	0.26741390
Class_variable	0.25560932	0.47426553	0.05333474	0.05837490	0.10278959	0.23151588	0.13192894	0.26741390	1.00000000

To have a better view we can create a heatmap of the correlation matrix:

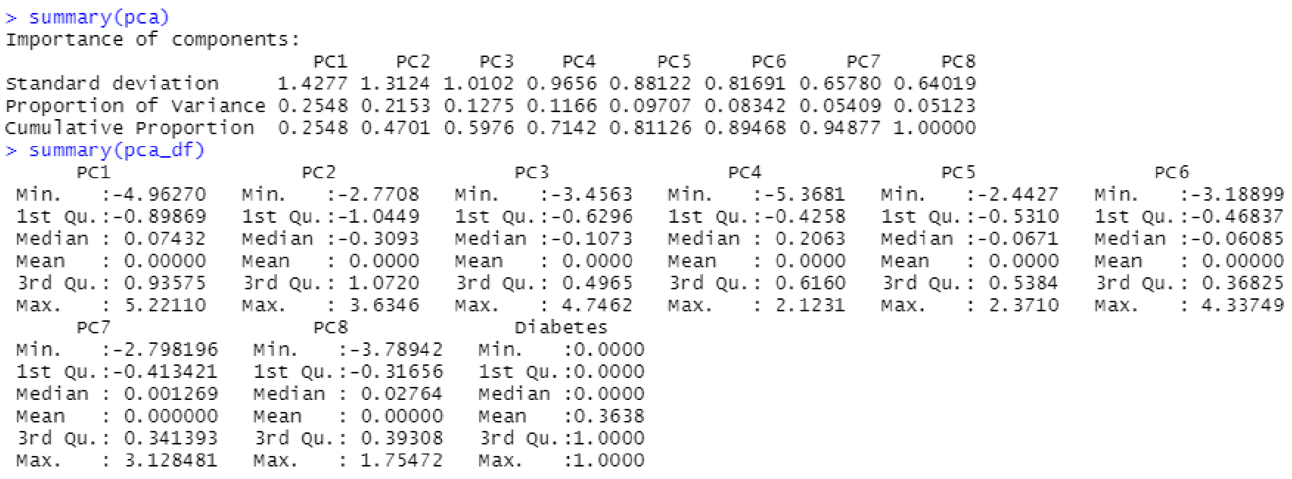
```r
# Create the correlation matrix
correlation_df <- as.data.frame(as.table(correlation_matrix))
colnames(correlation_df) <- c("Variable1", "Variable2", "Correlation")

# Plot the correlation matrix as a heatmap
ggplot(correlation_df, aes(Variable1, Variable2, fill = Correlation)) +
  geom_tile(color = "black") +
  scale_fill_gradient2(low = "#313695", mid = "white", high = "#C31C2C", midpoint = 0, limit = c(-1, 1),
                       name = "Correlation", na.value = "transparent") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title = "Correlation Matrix Heatmap")
```



Principal component analysis



```
# PRINCIPAL COMPONENT ANALYSIS

# Perform PCA on the dataset
pca <- prcomp(data[, 1:8], scale = TRUE)

# Display the summary of the PCA
summary(pca)

# Get the PCA results in a table format
pca_df <- data.frame(pca$x, data$Class_variable)

# Rename the column names of the PCA results dataframe
colnames(pca_df) <- c(paste0("PC", 1:8), "Diabetes")
summary(pca_df)
# Plot the PCA results using the first two principal components
data$Class_variable <- factor(data$Class_variable)

fviz_pca_ind(pca,
             geom.ind = "point",
             col.ind = data$Class_variable,
             palette = c("#00AFBB", "#E7B800"),
             addEllipses = TRUE,
             legend.title = "Outcome",
             ggtheme = theme_minimal())
```

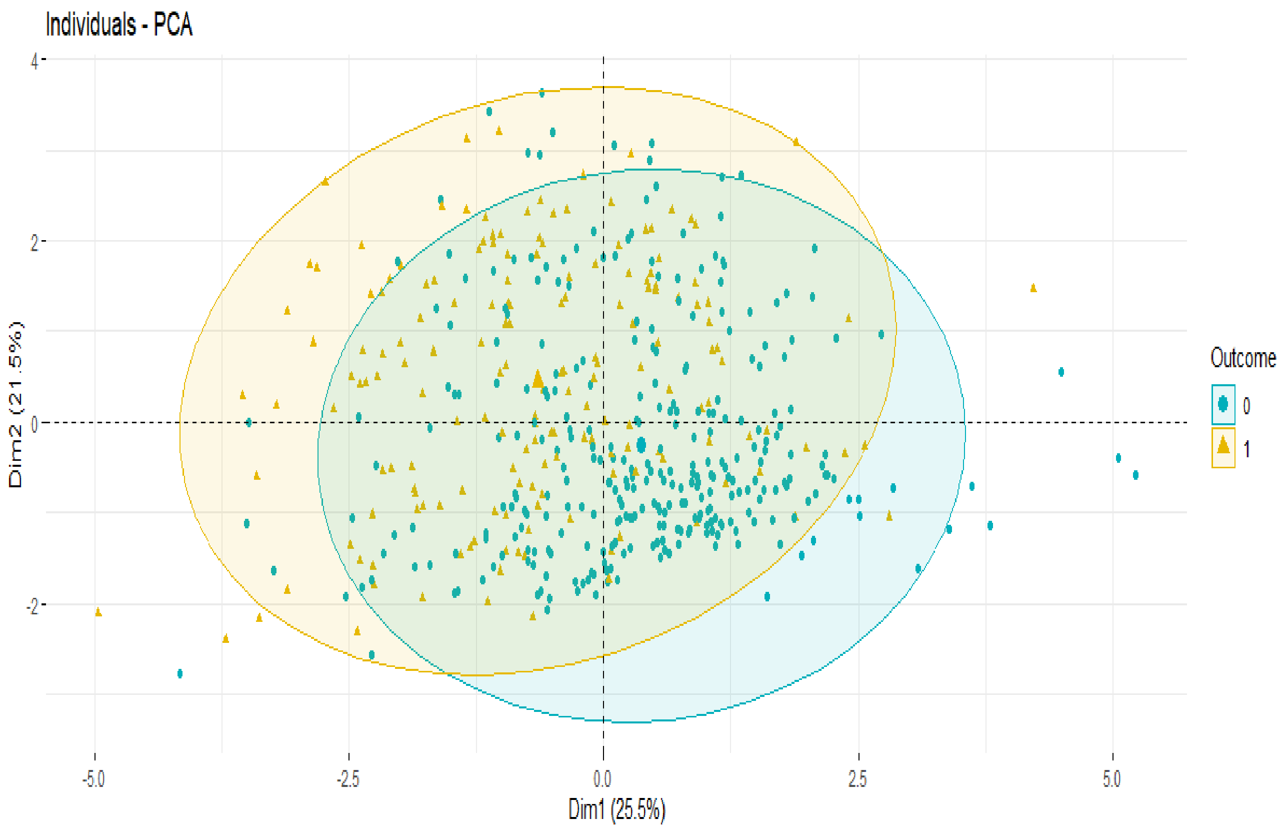


```
> summary(pca)
Importance of components:
                          PC1    PC2    PC3    PC4     PC5     PC6     PC7     PC8
Standard deviation     1.4277 1.3124 1.0102 0.9656 0.88122 0.81691 0.65780 0.64019
Proportion of Variance 0.2548 0.2153 0.1275 0.1166 0.09707 0.08342 0.05409 0.05123
Cumulative Proportion  0.2548 0.4701 0.5976 0.7142 0.81126 0.89468 0.94877 1.00000
> summary(pca_df)
      PC1                PC2                PC3                PC4                PC5                PC6          
 Min.   :-4.96270   Min.   :-2.7708   Min.   :-3.4563   Min.   :-5.3681   Min.   :-2.4427   Min.   :-3.18899  
 1st Qu.:-0.89869   1st Qu.:-1.0449   1st Qu.:-0.6296   1st Qu.:-0.4258   1st Qu.:-0.5310   1st Qu.:-0.46837  
 Median : 0.07432   Median :-0.3093   Median :-0.1073   Median : 0.2063   Median :-0.0671   Median :-0.06085  
 Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000  
 3rd Qu.: 0.93575   3rd Qu.: 1.0720   3rd Qu.: 0.4965   3rd Qu.: 0.6160   3rd Qu.: 0.5384   3rd Qu.: 0.36825  
 Max.   : 5.22110   Max.   : 3.6346   Max.   : 4.7462   Max.   : 2.1231   Max.   : 2.3710   Max.   : 4.33749  
      PC7                 PC8                Diabetes     
 Min.   :-2.798196   Min.   :-3.78942   Min.   :0.0000  
 1st Qu.:-0.413421   1st Qu.:-0.31656   1st Qu.:0.0000  
 Median : 0.001269   Median : 0.02764   Median :0.0000  
 Mean   : 0.000000   Mean   : 0.00000   Mean   :0.3638  
 3rd Qu.: 0.341393   3rd Qu.: 0.39308   3rd Qu.:1.0000  
 Max.   : 3.128481   Max.   : 1.75472   Max.   :1.0000  
```

The summary of the PCA shows the importance of each principal component (PC) and the amount of variance explained by each one. The first principal component (PC1) has the highest standard deviation of 1.4277 and explains 25.48% of the variance in the data. The second principal component (PC2) has a standard deviation of 1.3124 and explains 21.53% of the variance in the data. Together, the first two principal components account for 47.01% of the total variance in the data. The third and fourth principal components (PC3 and PC4) have standard deviations of 1.0102 and 0.9656, respectively, and each explain about 12% of the variance in the data.

In general, a high percentage of variance explained by the first few (usually 2 or 3 to better represent the results graphically) principal components indicates that the PCA has effectively captured the main patterns in the data.

```
# Plot the PCA results using the first two principal components
data$Class_variable <- factor(data$Class_variable)

fviz_pca_ind(pca,
             geom.ind = "point",
             col.ind = data$Class_variable,
             palette = c("#00AFBB", "#E7B800"),
             addEllipses = TRUE,
             legend.title = "Outcome",
             ggtheme = theme_minimal())
```

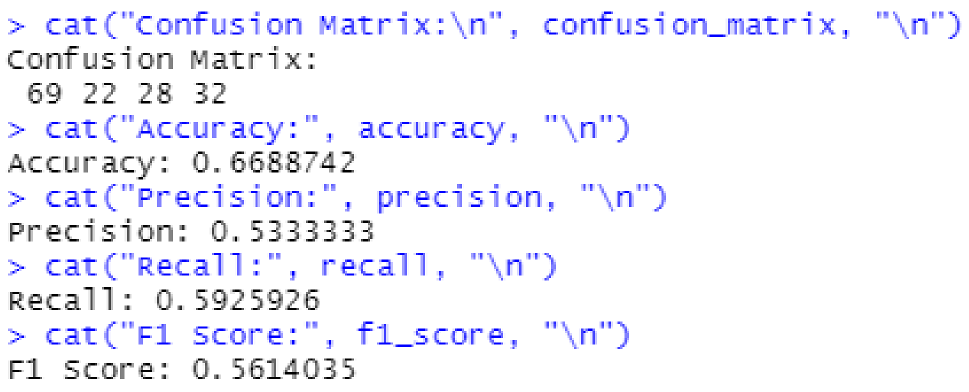


Logistic Regression

```
# Split the data into training and testing sets
set.seed(42)  # For reproducibility
train_indices <- sample(1:nrow(data), nrow(data) * 0.7)  # 70% for training
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]

# Perform logistic regression
model <- glm(Class_variable ~ ., data = train_data, family = binomial)
summary(model)
```

```
Call:
glm(formula = Class_variable ~ ., family = binomial, data = train_data)

Deviance Residuals: 
    Min       1Q   Median       3Q      Max  
-2.4900  -0.6504  -0.3862   0.6658   2.9250  

Coefficients:
                          Estimate Std. Error z value Pr(>|z|)    
(Intercept)             -8.0408929  1.0646780  -7.552 4.27e-14 ***
X                       -0.0001364  0.0005942  -0.230 0.818449    
pregnant_times           0.1614077  0.0484937   3.328 0.000873 ***
glucose_tolerance_test   0.0379068  0.0057438   6.600 4.12e-11 ***
Diastolic_blood_pressure -0.0171083  0.0083884  -2.040 0.041399 *  
Triceps_skin_fold_thickness  0.0013942  0.0109047   0.128 0.898266    
serum_insulin           -0.0009718  0.0012978  -0.749 0.453981    
Body_mass_index          0.0691258  0.0216073   3.199 0.001378 ** 
Diabetes_pedigree_function  0.0363156  0.4635075   0.078 0.937550    
Age                      0.0307612  0.0145124   2.120 0.034035 *  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 455.55  on 351  degrees of freedom
Residual deviance: 316.58  on 342  degrees of freedom
AIC: 336.58

Number of Fisher Scoring iterations: 5
```

```
# Make predictions on the test set
predicted_classes <- predict(model, newdata = test_data, type = "response")
predicted_classes <- ifelse(predicted_classes > 0.5, 1, 0)

# Evaluate the model
confusion_matrix <- table(predicted_classes, test_data$Class_variable)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
f1_score <- 2 * precision * recall / (precision + recall)

# Print the evaluation metrics
cat("Confusion Matrix:\n", confusion_matrix, "\n")
cat("Accuracy:", accuracy, "\n")
cat("Precision:", precision, "\n")
cat("Recall:", recall, "\n")
cat("F1 Score:", f1_score, "\n")
```



```
> cat("Confusion Matrix:\n", confusion_matrix, "\n")
Confusion Matrix:
 69 22 28 32
> cat("Accuracy:", accuracy, "\n")
Accuracy: 0.6688742
> cat("Precision:", precision, "\n")
Precision: 0.5333333
> cat("Recall:", recall, "\n")
Recall: 0.5925926
> cat("F1 Score:", f1_score, "\n")
F1 Score: 0.5614035
```

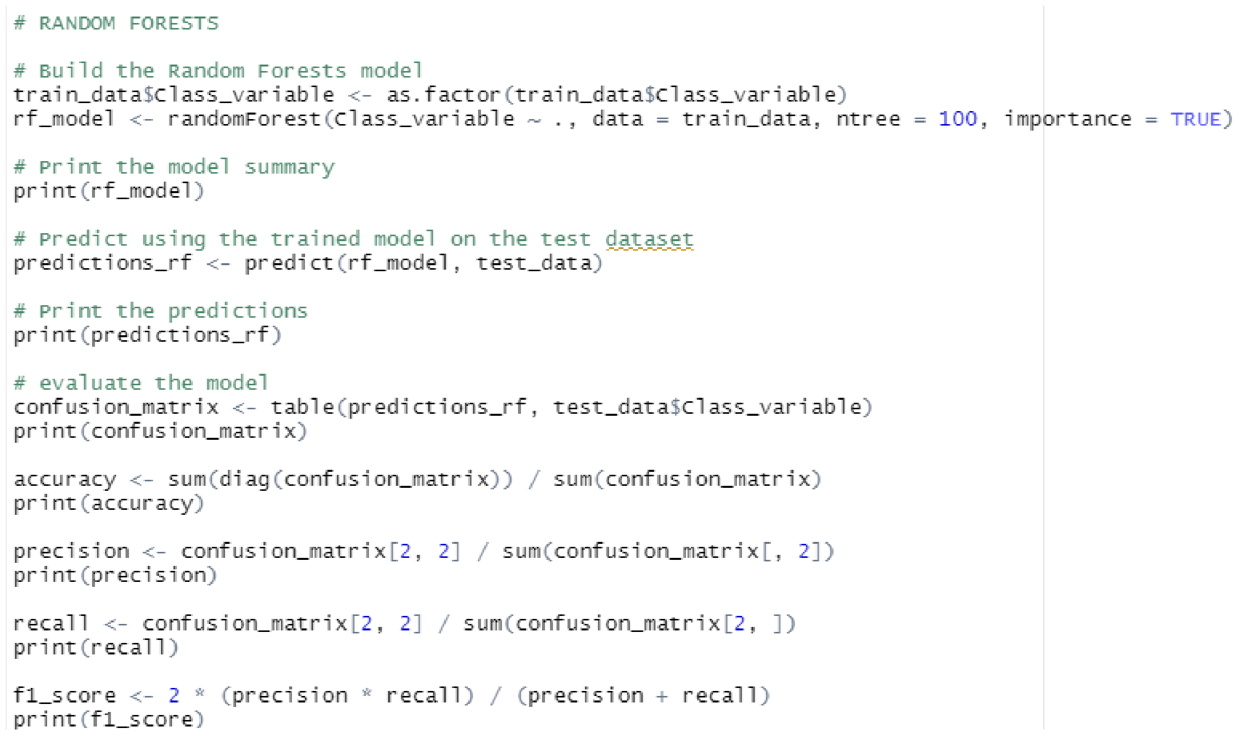
Logistic regression is a statistical modelling technique used for binary classification tasks. It is a generalized linear model that models the relationship between the predictor variables and the probability of the binary outcome. In logistic regression, the outcome variable is binary, typically represented as 0 and 1. The goal is to estimate the probability that an observation belongs to the positive class (1) given its predictor variables.

The logistic regression model applies the logistic function to transform a linear combination of the predictor variables. The logistic function maps any real-valued input to a value between 0 and 1, which can be interpreted as the predicted probability of belonging to the positive class. The model estimates the coefficients, also known as the weights or parameters, for each predictor variable. These coefficients represent the effect of each predictor on the log-odds of the positive outcome. The log-odds, also called the logit, is the natural logarithm of the odds of the positive outcome. The estimation of the coefficients is typically done using maximum likelihood estimation, where the model finds the coefficients that maximize the likelihood of observing the given data. This process involves optimizing the log-likelihood function, which measures how well the model fits the observed data.

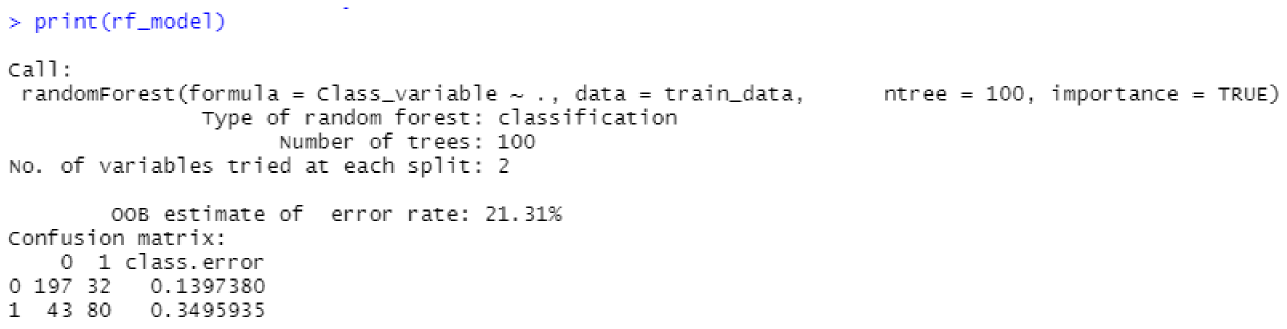
To evaluate the performance of a logistic regression model, various metrics can be used, such as accuracy, precision, recall, and the area under the receiver operating characteristic (ROC) curve. These metrics assess the model's ability to correctly classify instances and its discrimination power.

Logistic regression assumes that the relationship between the predictors and the log-odds of the positive outcome is linear. However, by including interactions and polynomial terms, logistic regression can capture nonlinear relationships between the predictors and the outcome.

Random Forests



```
# RANDOM FORESTS

# Build the Random Forests model
train_data$Class_variable <- as.factor(train_data$Class_variable)
rf_model <- randomForest(Class_variable ~ ., data = train_data, ntree = 100, importance = TRUE)

# Print the model summary
print(rf_model)

# Predict using the trained model on the test dataset
predictions_rf <- predict(rf_model, test_data)

# Print the predictions
print(predictions_rf)

# evaluate the model
confusion_matrix <- table(predictions_rf, test_data$Class_variable)
print(confusion_matrix)

accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(accuracy)

precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
print(precision)

recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
print(recall)

f1_score <- 2 * (precision * recall) / (precision + recall)
print(f1_score)
```



```
> print(rf_model)

Call:
 randomForest(formula = Class_variable ~ ., data = train_data,      ntree = 100, importance = TRUE)
               Type of random forest: classification
                     Number of trees: 100
No. of variables tried at each split: 2

        OOB estimate of  error rate: 21.31%
Confusion matrix:
    0  1 class.error
0 197 32   0.1397380
1  43 80   0.3495935
```

The random forest model is an ensemble learning method that combines the predictions of multiple decision trees to make accurate predictions. It is a powerful and popular algorithm for both classification and regression tasks. The basic idea behind random forests is to create an ensemble of decision trees, where each tree is trained on a random subset of the original dataset. This random subset is created through a process called bootstrapping, which involves sampling observations from the original dataset with replacement. This means that each tree is trained on a slightly different subset of the data, introducing diversity among the trees.

In addition to bootstrapping, random forests introduce randomness in the construction of each decision tree. When splitting a node in a decision tree, instead of considering all the features, random forests only consider a random subset of features. This further enhances the diversity among the trees and prevents individual trees from dominating the ensemble.

To make predictions with a random forest, each tree in the ensemble independently predicts the target variable based on the input features. For classification tasks, the final prediction is determined by majority voting, where the class that receives the most votes from the trees is chosen. For regression tasks, the final prediction is typically the average of the predictions from all the trees.

Random forests offer several advantages. They are robust against overfitting, as the randomness in the ensemble reduces the risk of individual trees memorizing noise in the training data. They also handle high-dimensional data well and can handle missing values and outliers without much preprocessing.

The performance of a random forest model can be evaluated using metrics such as accuracy, precision, recall, or mean squared error, depending on the nature of the problem. Additionally, feature importance can be assessed to understand which variables have the most significant impact on the predictions.

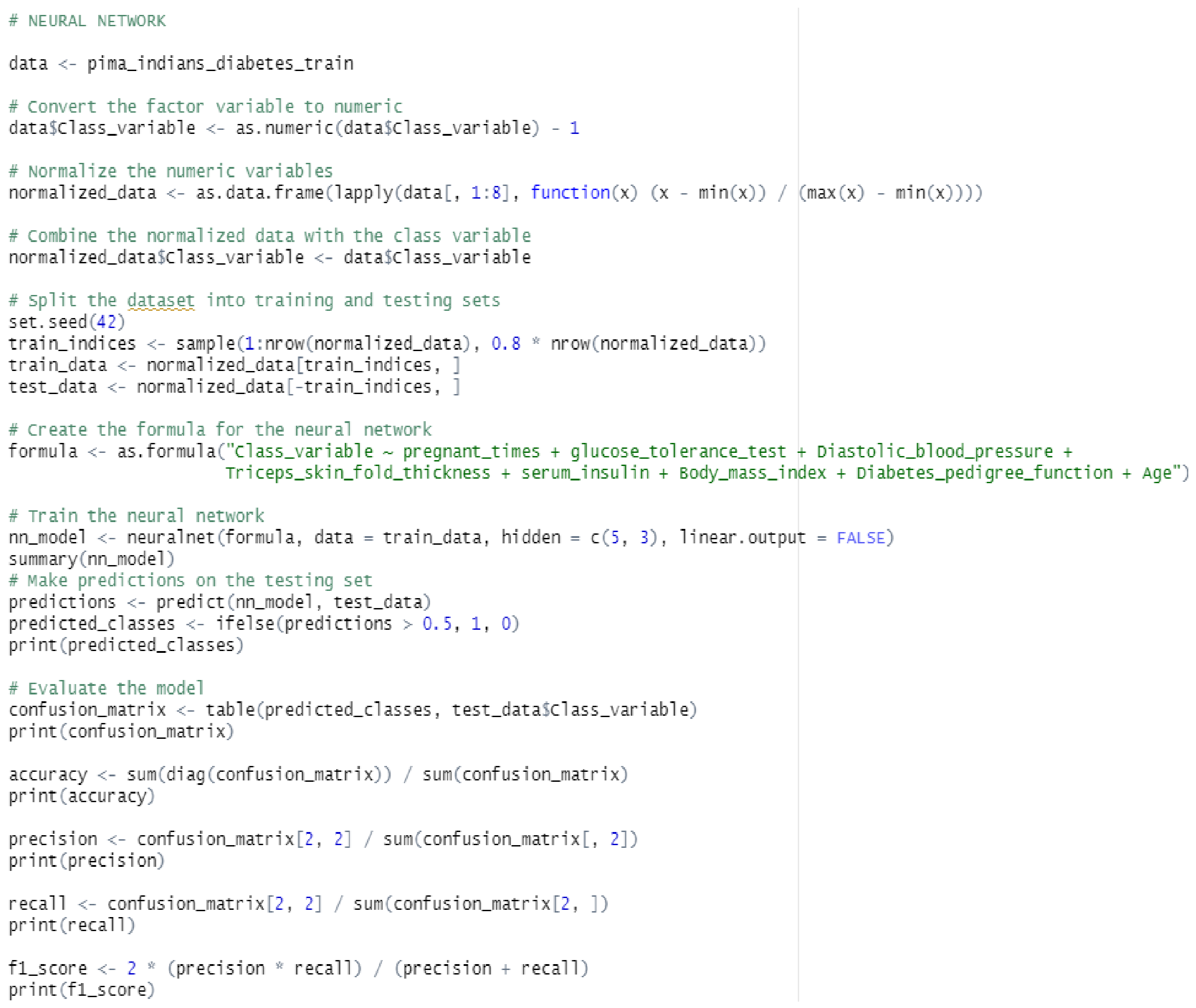
In summary, the random forest model is an ensemble learning method that combines the predictions of multiple decision trees. By introducing randomness through bootstrapping and feature selection, random forests provide robust and accurate predictions. They are widely used in various domains for classification and regression tasks due to their ability to handle complex data and mitigate overfitting.

Let's break down the different parts of the summary:

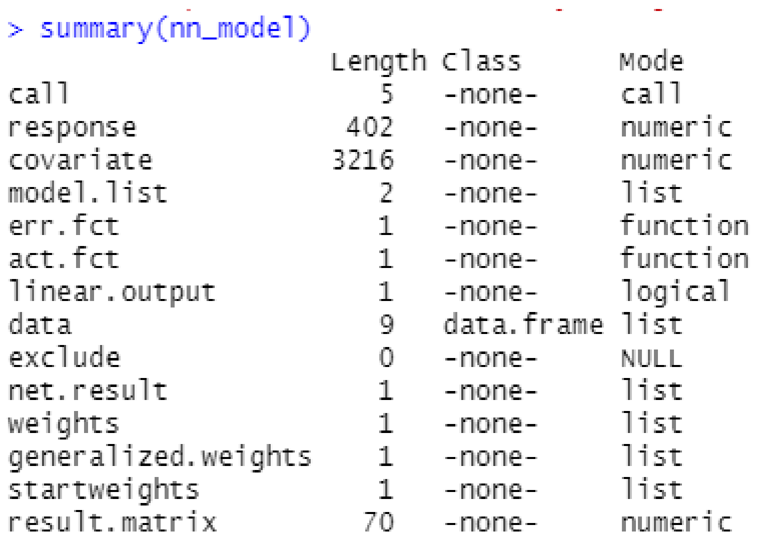
- Type of random forest: This line indicates that the Random Forest model we built is for classification.
- Number of trees: It specifies the number of trees grown in the Random Forest model. In this case, we have used 100 trees.
- No. of variables tried at each split: This line indicates the number of predictor variables randomly selected and evaluated at each split during the tree-building process. In this example, two variables are tried at each split.
- OOB estimate of error rate: OOB stands for "out-of-bag" and refers to the error rate estimated on the training data that was not used in building each individual tree. It serves as an estimate of how well the model might perform on unseen data. In this case, the estimated error rate is 25.65%, which means the model is expected to make incorrect predictions for approximately 25.65% of the observations.
- Confusion matrix: The confusion matrix provides a tabular representation of the model's performance by comparing the predicted class labels against the actual class labels. The rows correspond to the actual class labels, and the columns correspond to the predicted class labels. In our case, the confusion matrix is as follows:
  The values in the cells indicate the number of observations that fall into each category. For example, there are 265 observations that belong to class 0 and were correctly predicted as class 0, while there are 55 observations that belong to class 0 but were mistakenly predicted as class 1. The class.error column shows the error rate for each class.

These results provide an overview of the model's performance and can be used to evaluate its accuracy and effectiveness in classifying the data.

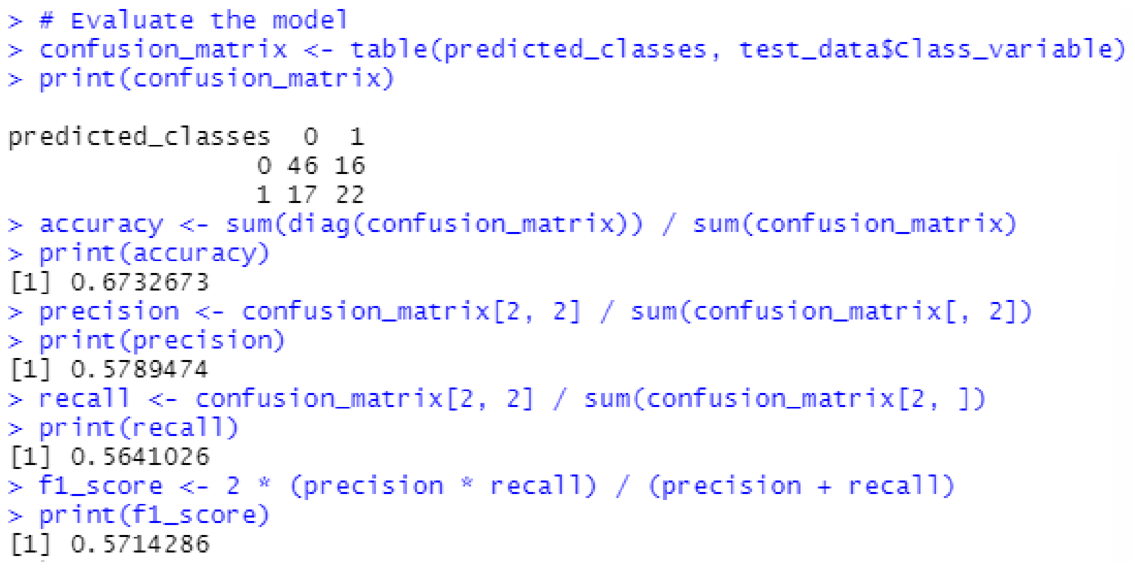
Neural Networks



```r
# NEURAL NETWORK

data <- pima_indians_diabetes_train

# Convert the factor variable to numeric
data$Class_variable <- as.numeric(data$Class_variable) - 1

# Normalize the numeric variables
normalized_data <- as.data.frame(lapply(data[, 1:8], function(x) (x - min(x)) / (max(x) - min(x))))

# Combine the normalized data with the class variable
normalized_data$Class_variable <- data$Class_variable

# Split the dataset into training and testing sets
set.seed(42)
train_indices <- sample(1:nrow(normalized_data), 0.8 * nrow(normalized_data))
train_data <- normalized_data[train_indices, ]
test_data <- normalized_data[-train_indices, ]

# Create the formula for the neural network
formula <- as.formula("Class_variable ~ pregnant_times + glucose_tolerance_test + Diastolic_blood_pressure +
                      Triceps_skin_fold_thickness + serum_insulin + Body_mass_index + Diabetes_pedigree_function + Age")

# Train the neural network
nn_model <- neuralnet(formula, data = train_data, hidden = c(5, 3), linear.output = FALSE)
summary(nn_model)
# Make predictions on the testing set
predictions <- predict(nn_model, test_data)
predicted_classes <- ifelse(predictions > 0.5, 1, 0)
print(predicted_classes)

# Evaluate the model
confusion_matrix <- table(predicted_classes, test_data$Class_variable)
print(confusion_matrix)

accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(accuracy)

precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
print(precision)

recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
print(recall)

f1_score <- 2 * (precision * recall) / (precision + recall)
print(f1_score)
```



```
> summary(nn_model)
                    Length Class      Mode
call                   5   -none-     call
response             402   -none-     numeric
covariate           3216   -none-     numeric
model.list             2   -none-     list
err.fct                1   -none-     function
act.fct                1   -none-     function
linear.output          1   -none-     logical
data                   9   data.frame list
exclude                0   -none-     NULL
net.result             1   -none-     list
weights                1   -none-     list
generalized.weights    1   -none-     list
startweights           1   -none-     list
result.matrix         70   -none-     numeric
```



```
> # Evaluate the model
> confusion_matrix <- table(predicted_classes, test_data$Class_variable)
> print(confusion_matrix)

predicted_classes  0  1
                0 46 16
                1 17 22
> accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
> print(accuracy)
[1] 0.6732673
> precision <- confusion_matrix[2, 2] / sum(confusion_matrix[, 2])
> print(precision)
[1] 0.5789474
> recall <- confusion_matrix[2, 2] / sum(confusion_matrix[2, ])
> print(recall)
[1] 0.5641026
> f1_score <- 2 * (precision * recall) / (precision + recall)
> print(f1_score)
[1] 0.5714286
```

The model used in this analysis is a neural network, a powerful machine learning algorithm inspired by the structure and functioning of the human brain. Neural networks consist of interconnected nodes, called neurons, organized in layers. Each neuron takes input from the previous layer, performs a computation, and passes the output to the next layer to predict, in this case, whether a woman has diabetes or not, represented by 0 and 1, respectively.

The neural network is trained using the backpropagation algorithm, which involves iteratively adjusting the weights between neurons to minimize the difference between predicted and actual values. The training process involves feeding the network with known input-output pairs from the training dataset and updating the weights accordingly. To prepare the data for training, the numeric variables are normalized to a common scale using min-max normalization. This normalization ensures that each feature contributes equally to the training process, regardless of its original range. The neural network architecture is specified with two hidden layers: the first layer has five neurons, and the second layer has three neurons. The number of neurons and layers is determined through experimentation and domain knowledge, aiming to find a balance between model complexity and performance.

Once the neural network is trained, it is evaluated using the testing dataset. The model predicts the "Class_variable" for each instance in the testing set, and the predicted classes are compared with the actual classes to assess the accuracy of the model. Additionally, a confusion matrix is created to provide a detailed analysis of the model's performance, including metrics such as true positives, true negatives, false positives, and false negatives.

In summary, the applied model utilizes a neural network, trained using the backpropagation algorithm, to predict diabetes in Pima Indian women based on various features. The model's performance is evaluated using accuracy metrics and a confusion matrix. Neural networks have the advantage of capturing complex relationships in the data and can be effective in solving classification problems like this one.

The summary of the model_nn object provides information about various components of the neural network model. Here are some comments on the key components:

- call: It shows the function call used to create the neural network model.
- response: It represents the response variable used in the model, which consists of 402 observations of numeric type.
- covariate: It indicates the covariates or predictors used in the model, with a total of 3216 observations of numeric type.

- err.fct and act.fct: These are the error and activation functions used in the model, respectively. They determine how the model calculates errors and processes input data.
- linear.output: It is a logical value that indicates whether the output from the model is linear or not.
- data: It represents the dataset used for training the model, consisting of nine variables in a data frame format.
- net.result: It is a list containing the resulting values from the neural network model.
- weights, generalized.weights, and startweights: These are lists representing the weights used in the model.
- result.matrix: It is a numeric matrix with 70 entries, representing the results obtained from the neural network model.

Compare The Results

	Logistic Regression	Random Forests	Neural Network
Accuracy	0.6688742	0.6887417	0.6732673
Precision	0.5333333	0.6166667	0.5789474
Recall	0.5925926	0.6065574	0.5641026
F1 Score	0.5614035	0.6115702	0.5714286

The logistic regression model achieved an accuracy of 0.6689, which means it correctly predicted the outcome for approximately 66.9% of the instances. However, its precision of 0.5333 suggests that it had a relatively low ability to correctly identify positive cases. The recall score of 0.5926 indicates that the model captured around 59.3% of the true positive cases. The F1 score of 0.5614, which considers both precision and recall, reflects a moderate overall performance for the logistic regression model.

On the other hand, the random forests model showed a slightly higher accuracy of 0.6887, indicating that it had a better overall predictive power than logistic regression. Its precision score of 0.6167 suggests a higher ability to correctly classify positive cases compared to logistic regression. However, the recall score of 0.6066 indicates that it captured slightly fewer true positive cases than logistic regression. The F1 score of 0.6116 suggests a relatively balanced performance between precision and recall for random forests.

Lastly, the neural network model achieved an accuracy of 0.6733, similar to logistic regression but lower than random forests. The precision score of 0.5789 indicates that it had moderate success in correctly identifying positive cases. The recall score of 0.5641 suggests that the neural network captured around 56.4% of the true positive cases. The F1 score of 0.5714 indicates a relatively balanced performance between precision and recall for the neural network.

Overall, based on these results, the random forests model seems to perform slightly better than the logistic regression and neural network models in terms of accuracy, precision, recall, and F1 score.