PIMA Indians Diabetes Prediction Project

HarvardX: PH125.9x Data Science - Choose your own project

Usha V

04 March, 2022

Contents

Chapter 1

Overview

This project is done for 'Choose-your-own project' of the HervardX: PH125.9x Data Science: Capstone course. The present report starts with a general idea of the project followed by analysis and report.

For this, the given dataset is prepared and setup. An exploratory data analysis is carried out in order to develop a machine learning algorithm that could predict whether a patient is diabetic or not. Results are explained. Finally, the report ends with some concluding remarks.

1.1 Introduction

Diabetes is one of the fastest growing chronic life threatening disease that has already affected 422 million people worldwide according to the report of World Health Organization (WHO), in 2018. Due to the presence of a relatively long asymptomatic phase, early detection of diabetes is always desired for a clinically meaningful outcome. Around 50% of all people suffering from diabetes are undiagnosed because of its long-term asymptomatic phase.

The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

The dataset consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

There are 768 observations and 8 independent variables in the dataset. The target variable indicates the test result of the patient. It is 1 when the test result is positive and 0 when the test result is negative.

This project will make a performance comparison between different machine learning algorithms in order to assess the correctness in classifying data with respect to efficiency and effectiveness of each algorithm based on the metrics accuracy, precision, sensitivity and specificity, for better diagnosis.

Diagnosis in an early stage is essential to facilitate the subsequent clinical management of patients and increase the life of diabetic patients.

The major models used and tested are supervised learning models (algorithms that learn from labelled data), which are most used in these kinds of data analysis.

The utilization of data science and machine learning approaches in medical fields proves to be prolific as such approaches may be considered of great assistance in the decision making process of medical practitioners. With an unfortunate increasing trend of diabetic cases, comes also a big deal of data which is of significant use in furthering clinical and medical research, and much more to the application of data science and machine learning in the aforementioned domain.

1.2 Aim of the project

The objective of this report is to train machine learning models to predict whether a patient is diabetic or not. Data is transformed to reveal patterns in the dataset and create a more robust analysis. As previously said, the optimal model will be selected following the metrics accuracy, sensitivity, and f1 score, amongst other factors. We will later define these metrics. We can use machine learning method to extract the features of diabetes and classify them. It would be helpful to determine whether a given sample appears to be Diabetic("1") or not("0").

The machine learning models that we will apply in this report try to create a classifier that provides a high accuracy level combined with a low rate of false-negatives (high sensitivity).

1.3 Dataset

The present report covers the PIMA Indians Diabetes DataSet (https://www.kaggle.com/uciml/pima-indians-diabetes-database/version/1). The Pima Indian Diabetes Dataset, originally from the National Institute of Diabetes and Digestive and Kidney Diseases, contains information of 768 women from a population near Phoenix, Arizona, USA. The outcome tested was Diabetes, 258 tested positive and 500 tested negative. Therefore, there is one target (dependent) variable and the 8 attributes (TYNECKI, 2018): pregnancies, OGTT(Oral Glucose Tolerance Test), blood pressure, skin thickness, insulin, BMI(Body Mass Index), age, pedigree diabetes function. The Pima population has been under study by the National Institute of Diabetes and Digestive and Kidney Diseases at intervals of 2 years since 1965. As epidemiological evidence indicates that T2DM results from interaction of genetic and environmental factors. The Pima Indians Diabetes Dataset includes information about attributes that could and should be related to the onset of diabetes and its future complications.

The .csv format file containing the data is loaded from my personal github account.

The dataset's features describe characteristics of the diabetic patients. The features information are specified below:

- Attribute Description:

  1. Pregnancies — Number of times pregnant
  2. Glucose — The blood plasma glucose concentration after a 2 hour oral glucose tolerance test (mg/dL)
  3. BloodPressure — Diastolic blood pressure (mm/Hg)
  4. SKinThickness — Skinfold Triceps skin fold thickness (mm)
  5. Insulin — 2 Hour serum insulin (mu U/ml)
  6. BMI — Body mass index (weight in kg/(height in m)^2)
  7. DiabetesPedigreeFunction — A function that determines the risk of type 2 diabetes based on family history, the larger the function, the higher the risk of type 2 diabetes.
  8. Age
  9. Outcome — Whether the person is diagnosed with type 2 diabetes (1 = yes, 0 = no)

The dataset has nine attributes(parameters) in which there are eight independent variables (Pregnancies, Glucose, Blood Pressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age) and one dependent variable (Outcome). There are 768 observations and 8 independent variables in the dataset. The target variable indicates the test result of the patient. It is 1 when the test result is positive and 0 when the test result is negative.

Chapter 2

Methods and Analysis

2.1 Data Analysis

Before we study the data set, let's convert the output variable ('Outcome') into a categorical variable. This is necessary because our output will be in the form of 2 classes, True or False. Where true will denote that a patient has diabetes and false denotes that a person is diabetes free.

```
data$Outcome <- factor(data$Outcome, levels = c(0,1), labels = c("False", "True"))
```

By observing our dataset, we found that it contains 768 observations with 9 variables.

```
str(data)
```

```
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 0 23 35 0 32 0 45 0 ...
 $ Insulin                 : int  0 0 0 94 168 0 88 0 543 0 ...
 $ BMI                     : num  33.6 26.6 23.3 28.1 43.1 25.6 31 35.3 30.5 0 ...
 $ DiabetesPedigreeFunction: num  0.627 0.351 0.672 0.167 2.288 ...
 $ Age                     : int  50 31 32 21 33 30 26 29 53 54 ...
 $ Outcome                 : Factor w/ 2 levels "False","True": 2 1 2 1 2 1 2 1 2 2 ...
```

```
head(data)
```

```
  Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI
1           6     148            72            35       0 33.6
2           1      85            66            29       0 26.6
3           8     183            64             0       0 23.3
4           1      89            66            23      94 28.1
5           0     137            40            35     168 43.1
6           5     116            74             0       0 25.6
  DiabetesPedigreeFunction Age Outcome
1                    0.627  50    True
2                    0.351  31   False
```

```
3                     0.672  32    True
4                     0.167  21   False
5                     2.288  33    True
6                     0.201  30   False
```

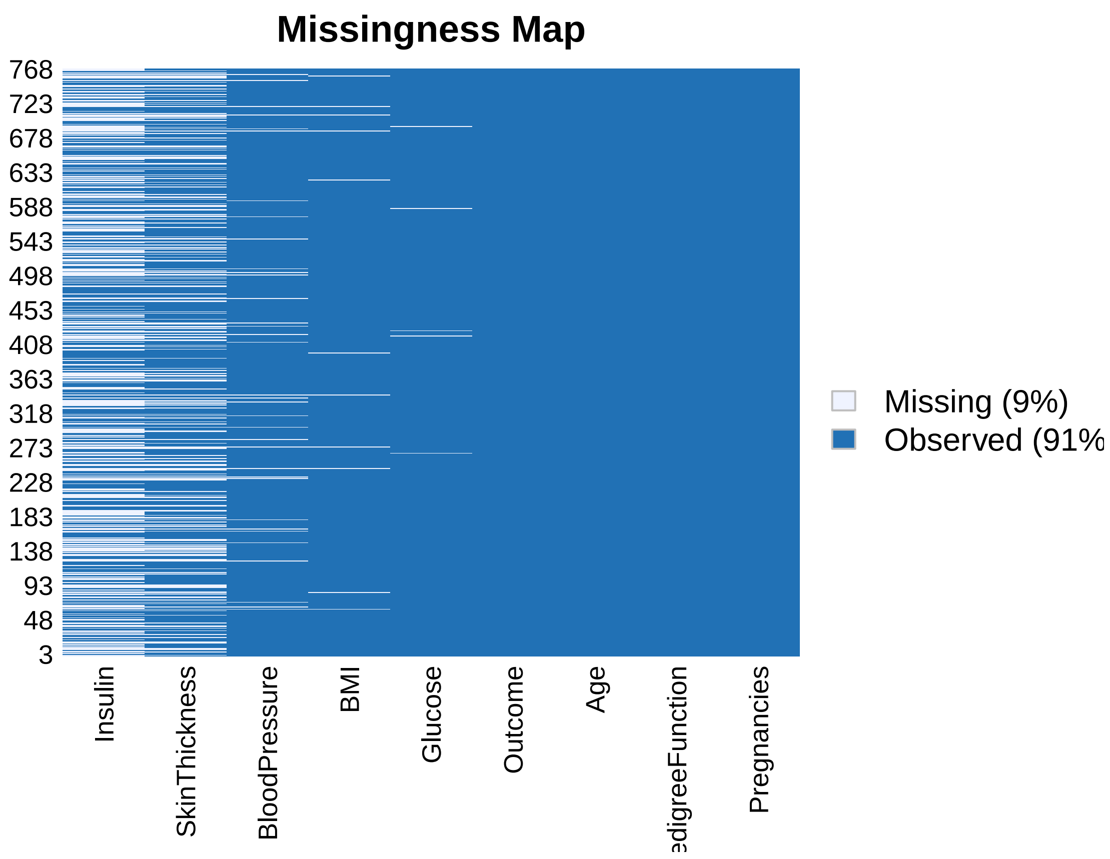
```r
summary(data)
```

```
  Pregnancies        Glucose      BloodPressure    SkinThickness
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00
    Insulin           BMI        DiabetesPedigreeFunction      Age
 Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00
 1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00
 Median : 30.5   Median :32.00   Median :0.3725           Median :29.00
 Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24
 3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00
 Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00
  Outcome
 False:500
 True :268
```

While analyzing the structure of the data set, we can see that the minimum values for Glucose, Bloodpressure, Skinthickness, Insulin, and BMI are all zero. This is not ideal since no one can have a value of zero for Glucose, blood pressure, etc. Therefore, such values are treated as missing observations.

```r
data[, 2:7][data[, 2:7] == 0] <- NA
```

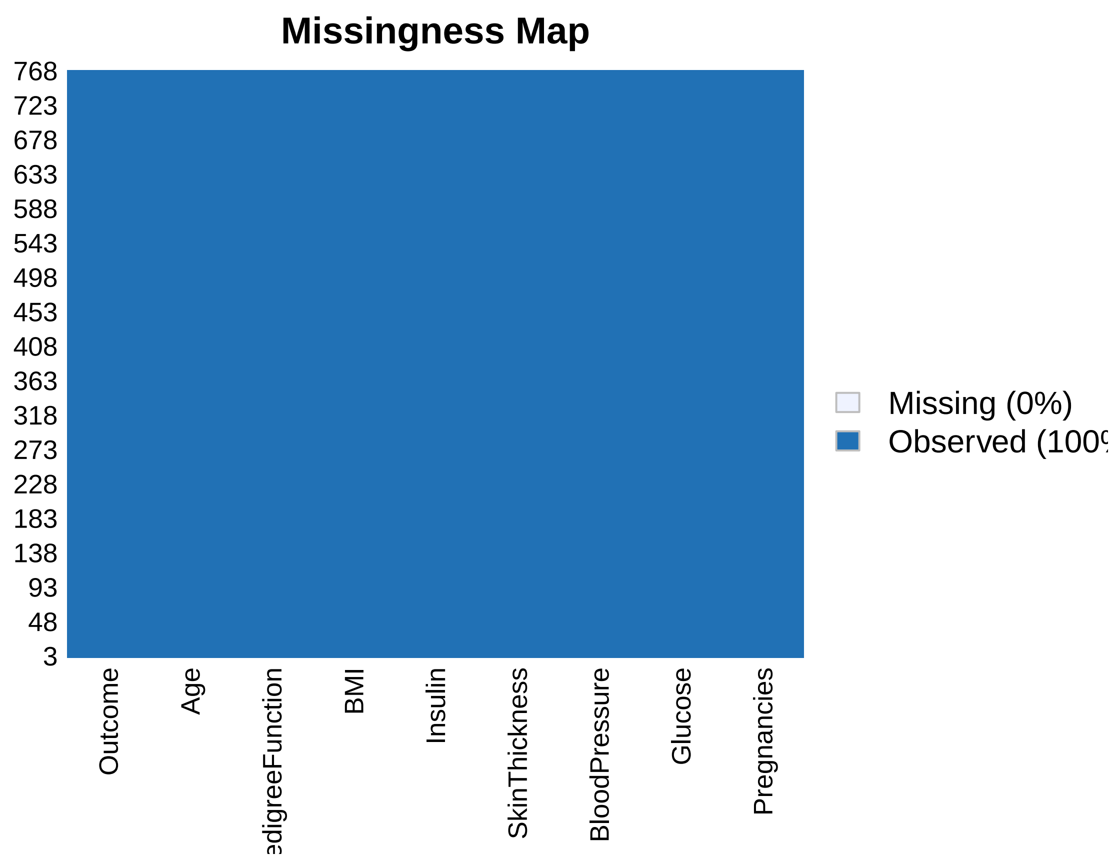
To check how many missing values we have now, let's visualize the data:



The above illustrations show that our data set has plenty missing values and removing all of them will leave us with an even smaller data set, therefore, we can perform imputations by using the mice package in R.

```
 iter imp variable
  1   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  1   2  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  1   3  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  1   4  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  1   5  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  2   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  2   2  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  2   3  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  2   4  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  2   5  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  3   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  3   2  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  3   3  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  3   4  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  3   5  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  4   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  4   2  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  4   3  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  4   4  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  4   5  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  5   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  5   2  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  5   3  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  5   4  Glucose  BloodPressure  SkinThickness  Insulin  BMI
  5   5  Glucose  BloodPressure  SkinThickness  Insulin  BMI
```

To check if there are still any missing values, let's use the missmap plot:



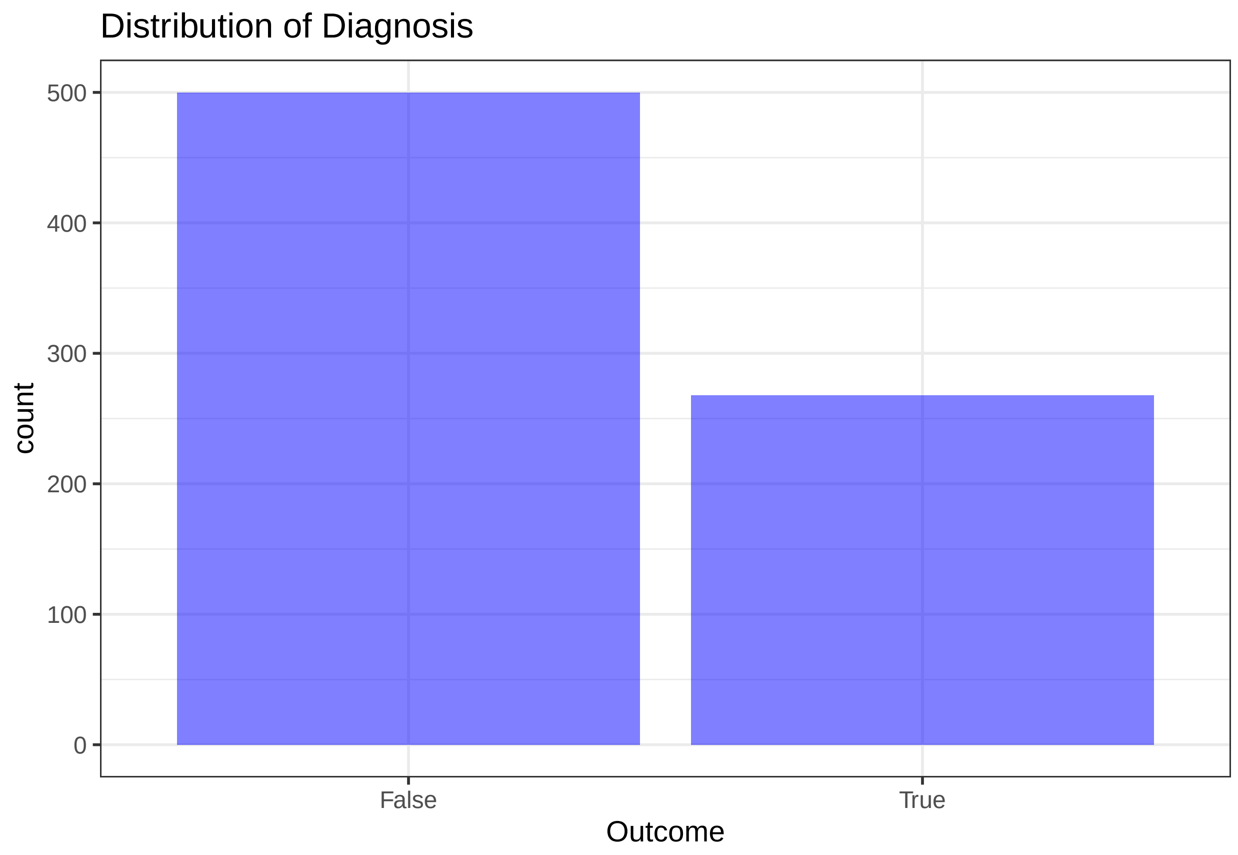
The output looks good, there is no missing data.

```
prop.table(table(data$Outcome))
```

```
    False      True 
0.6510417 0.3489583 
```

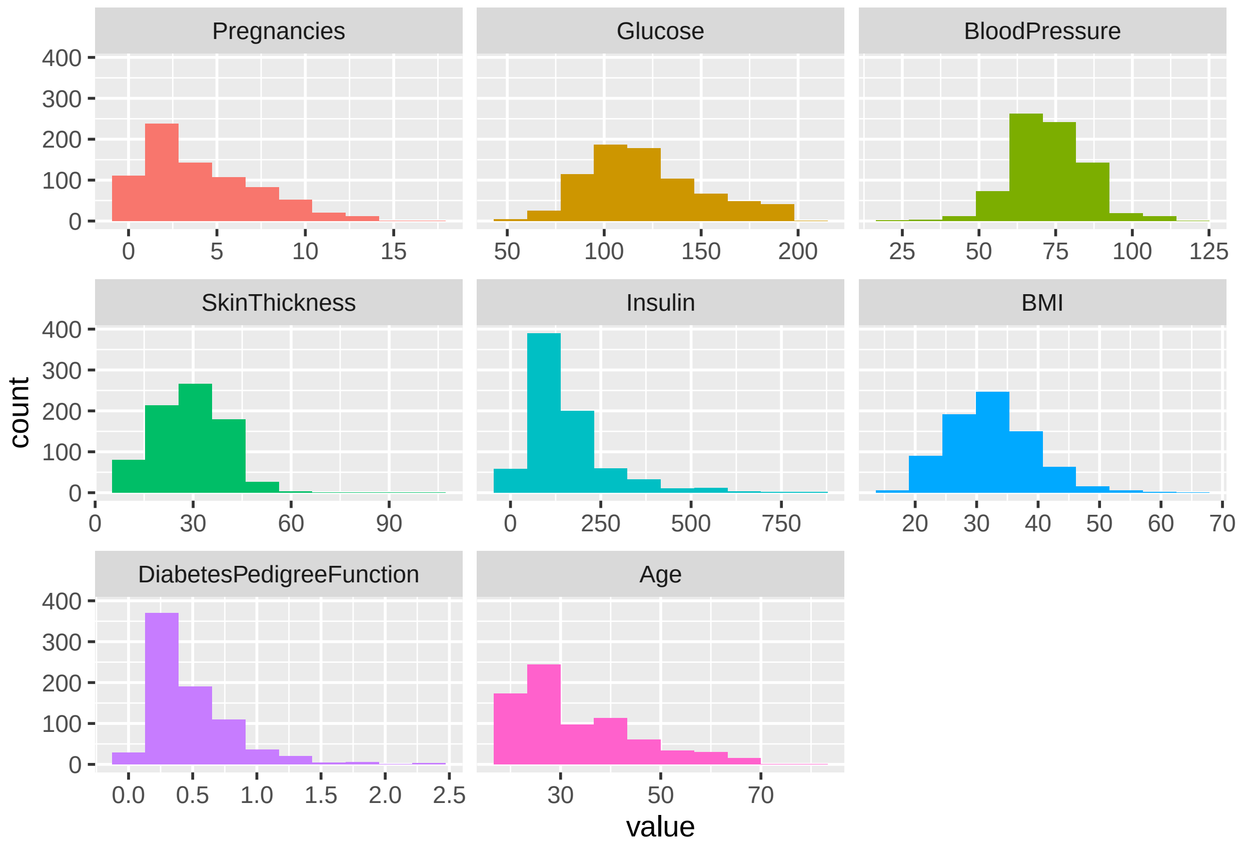
By analysing the dataset, we discover that it is a bit unbalanced in its proportions.

```
options(repr.plot.width=4, repr.plot.height=4)
ggplot(data, aes(x=Outcome))+geom_bar(fill="blue",alpha=0.5)+theme_bw()+labs(title="Distribution of Dia
```



Also the plot of proportions confirms that the target variable is slightly unbalanced.

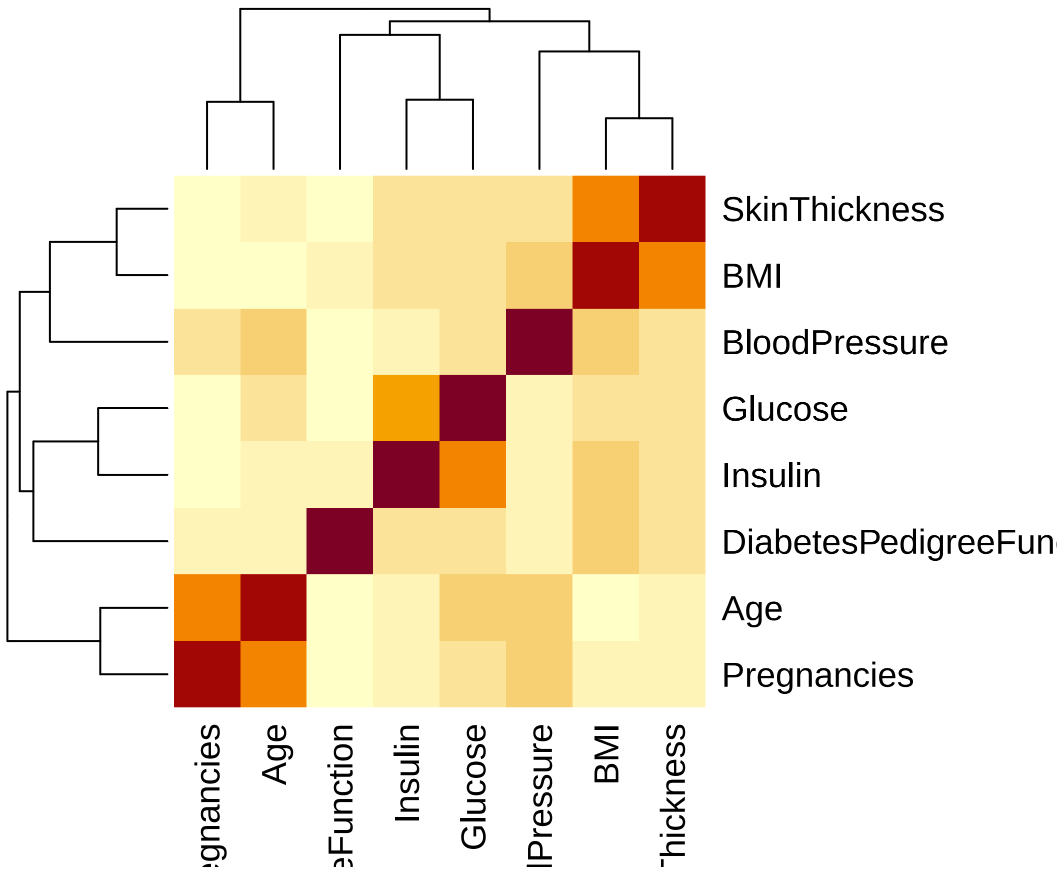
```
plot_num(data %>% select (-Outcome), bins=10)
```



Most of the variables of the dataset are normally distributed as shown in the previous plot.

Now we have to check if there is any correlation between variables as machine learning algorithms assume that the predictor variables are independent from each other.

```
correlationMatrix <- cor(data[,1:ncol(data)-1])
heatmap(correlationMatrix)
```



As shown by this heatmap, there is not much correlation between features except for some values that are approx greater than 0.5 The features are:

Age-Pregnancies : Pregnancies can increase with age and stop after a certain age.

Glucose-Diabetes : Higher glucose count has higher probability of being diagnosed with diabetes

Glucose-Insulin : Higher level Glucose means more Insulin

BMI-SkinThickness : Higher the BMI, fatter the person is.

```
# find attributes that are highly corrected (ideally >0.90)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.9)
# print indexes of highly correlated attributes
print(highlyCorrelated)
```

```
integer(0)
```

There are no convincing relationship between the independent parameters. The original dataset can be used for further exploration.

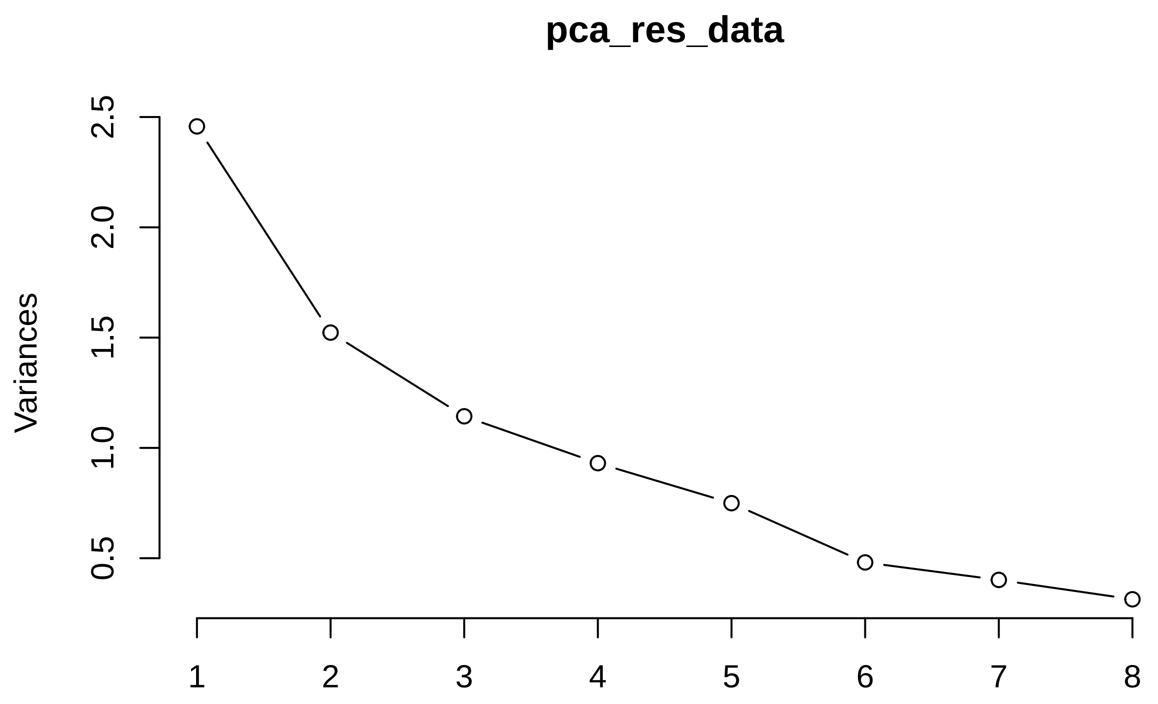
2.2 Modelling Approach

2.2.1 Modelling

Principal Component Analysis (PCA).

To avoid redundancy and relevancy, we used the function 'prncomp' to calculate the Principal Component Analysis (PCA) and select the rights components to avoid correlated variables that can be detrimental to our clustering analysis. One of the common problems in analysis of complex data comes from a large number of variables, which requires a large amount of memory and computation power. This is where PCA comes in. It is a technique to reduce the dimension of the feature space by feature extraction. The main idea of PCA is to reduce the dimensionality of a data set consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset, up to the maximum extent. The same is done by transforming the variables to a new set of variables, which are known as the principal components (or simply, the PCs) and are orthogonal, ordered such that the retention of variation present in the original variables decrease as we move down in the order.

```
pca_res_data <- prcomp(data[,1:ncol(data)-1], center = TRUE, scale = TRUE)
plot(pca_res_data, type="l")
```

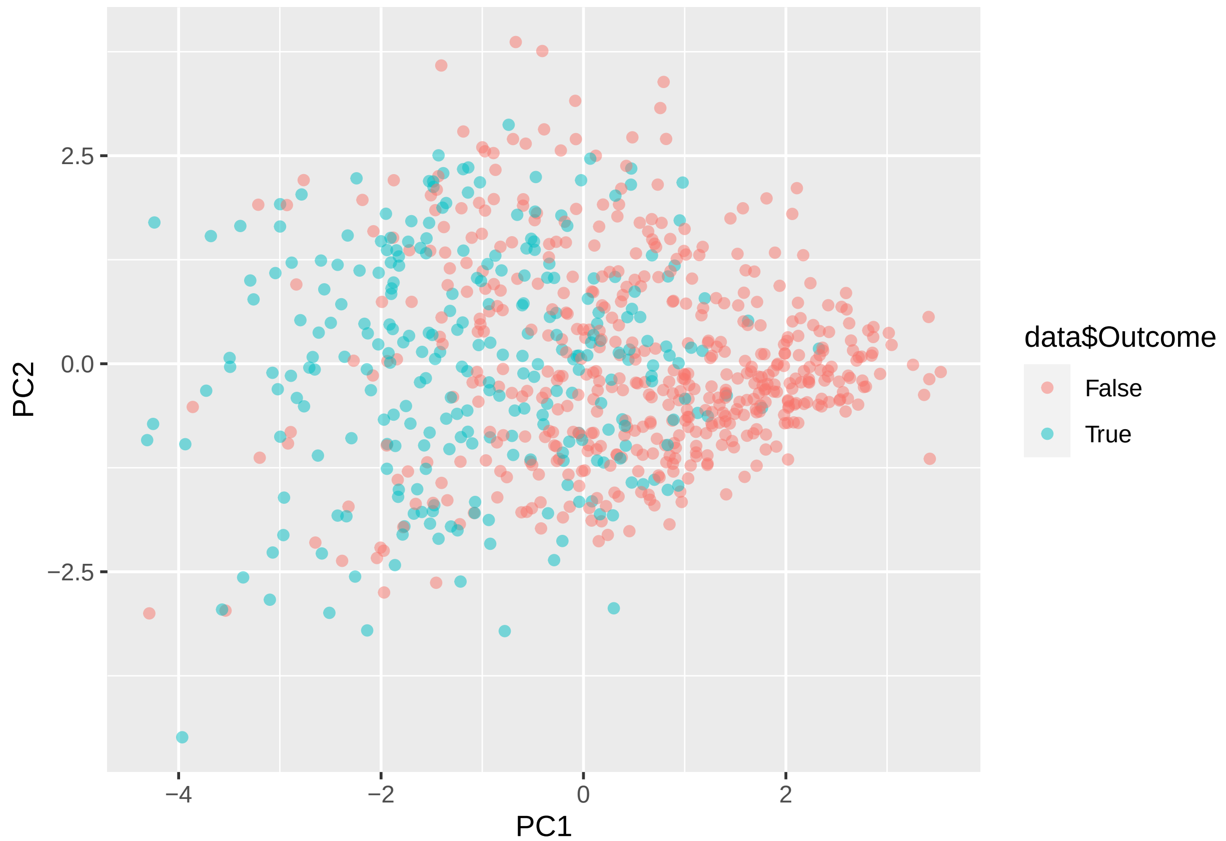


```
summary(pca_res_data)
```

```
Importance of components:
                          PC1    PC2    PC3    PC4     PC5     PC6     PC7
Standard deviation     1.5676 1.2340 1.0693 0.9648 0.86572 0.69343 0.63369
Proportion of Variance 0.3072 0.1903 0.1429 0.1163 0.09368 0.06011 0.05019
Cumulative Proportion  0.3072 0.4975 0.6404 0.7568 0.85048 0.91059 0.96078
                           PC8
Standard deviation     0.56014
Proportion of Variance 0.03922
Cumulative Proportion  1.00000
```

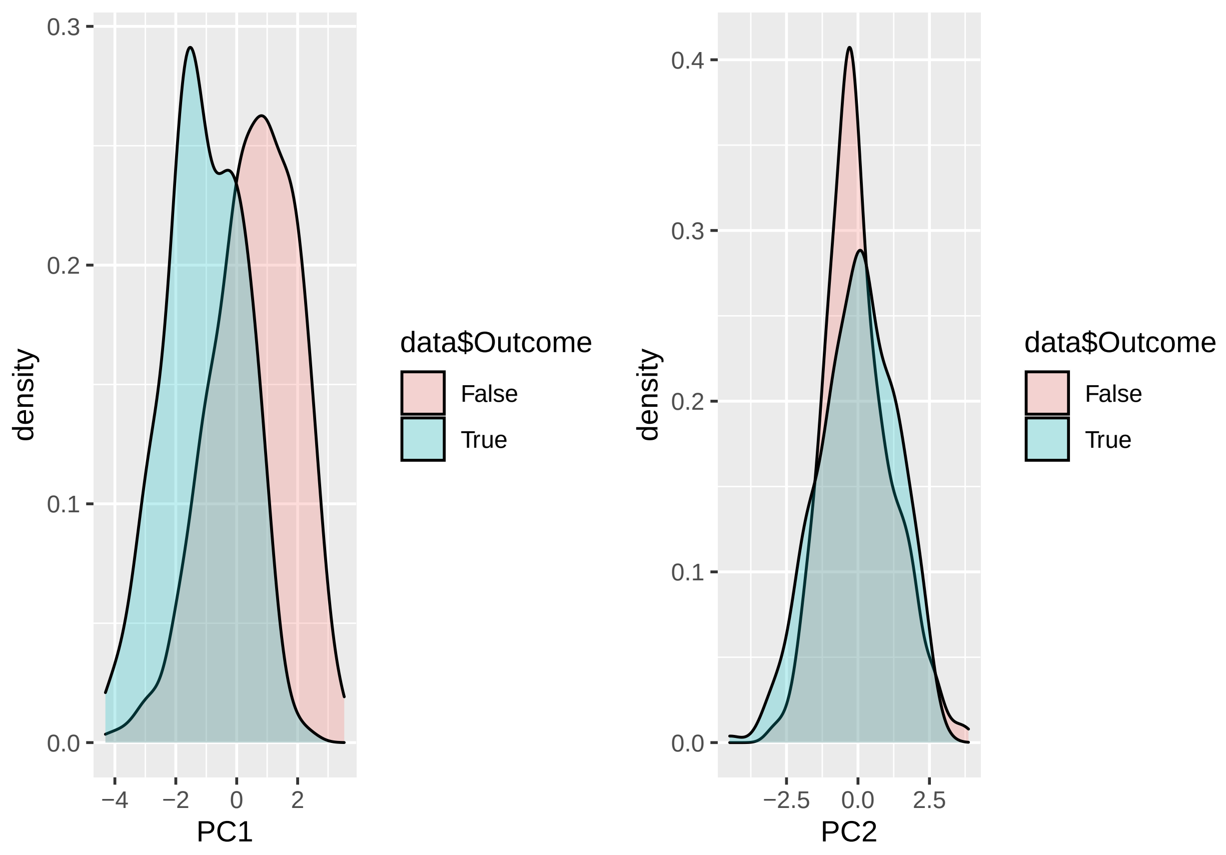
As we can observe from the above table, first three components explains the 0.6434 of the variance. We need all principal components to explain more than 0.99. Hence all components are important for our analysis.

```
pca_df <- as.data.frame(pca_res_data$x)
ggplot(pca_df, aes(x=PC1, y=PC2, col=data$Outcome)) + geom_point(alpha=0.5)
```



The data of the first 2 components can be easily separated into two classes. This is caused by the fact that the variance explained by these components is not large. The data can be easily separated.

```
g_pc1 <- ggplot(pca_df, aes(x=PC1, fill=data$Outcome)) + geom_density(alpha=0.25)
g_pc2 <- ggplot(pca_df, aes(x=PC2, fill=data$Outcome)) + geom_density(alpha=0.25)
grid.arrange(g_pc1, g_pc2, ncol=2)
```



Linear Discriminant Analysis (LDA)

Another approach is to use the Linear Discriminant Analysis (LDA) instead of PCA. LDA takes in consideration the different classes and could get better results. The particularity of LDA is that it models the distribution of predictors separately in each of the response classes, and then it uses Bayes' theorem to estimate the probability. It is important to know that LDA assumes a normal distribution for each class, a class-specific mean, and a common variance.

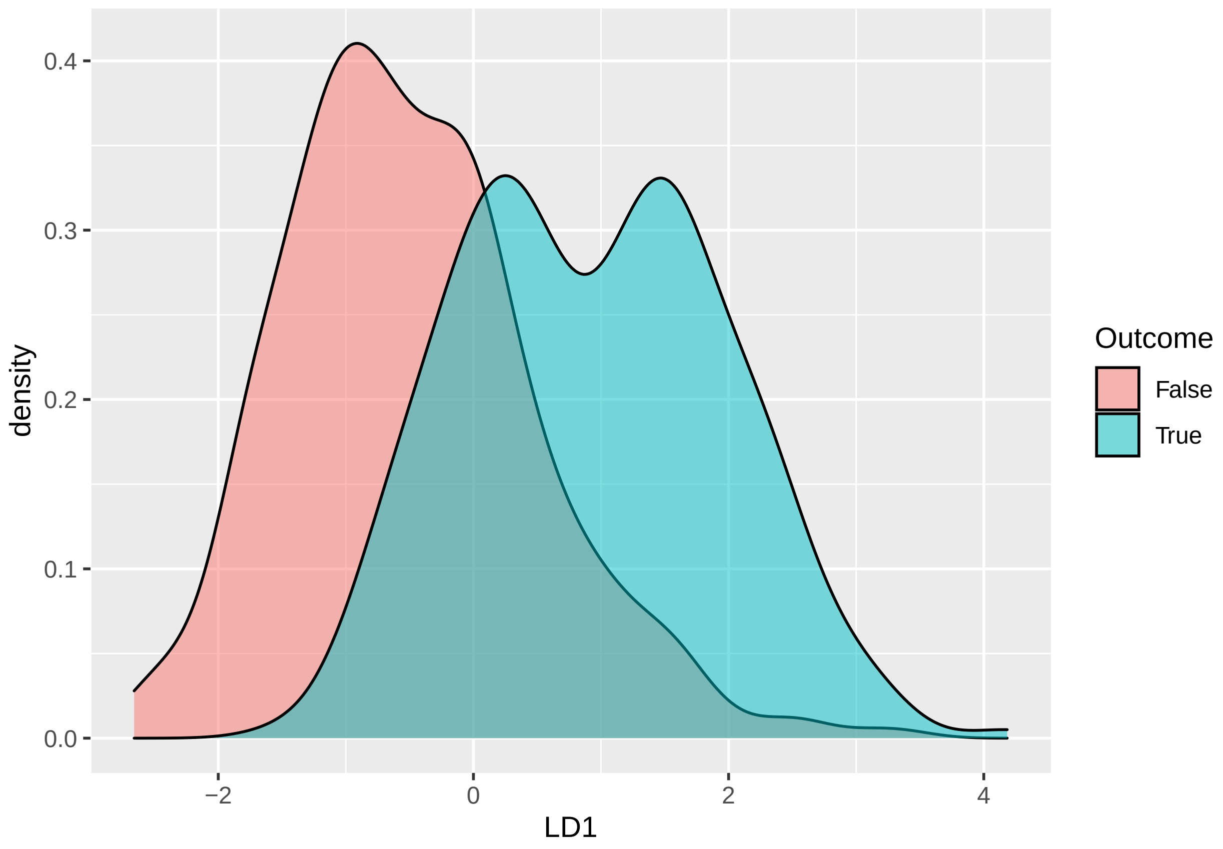
```
lda_res_data <- MASS::lda(Outcome~., data = data, center = TRUE, scale = TRUE)
lda_res_data
```

```
Call:
lda(Outcome ~ ., data = data, center = TRUE, scale = TRUE)

Prior probabilities of groups:
    False      True 
0.6510417 0.3489583 

Group means:
      Pregnancies  Glucose BloodPressure SkinThickness  Insulin      BMI
False    3.298000 110.7800      70.87800      26.82600 128.5600 30.83180
True     4.865672 142.2276      75.41791      32.78731 197.6007 35.35224
      DiabetesPedigreeFunction      Age
False                 0.429734 31.19000
True                  0.550500 37.06716
```

```
Coefficients of linear discriminants:
                                   LD1
Pregnancies                0.0930655784
Glucose                    0.0287655598
BloodPressure             -0.0041754546
SkinThickness              0.0040696570
Insulin                   -0.0003082013
BMI                        0.0605629974
DiabetesPedigreeFunction   0.5884108162
Age                        0.0088846635
```

```
#Data frame of the LDA for visualization purposes
lda_df_predict <- predict(lda_res_data, data)$x %>% as.data.frame() %>% cbind(Outcome=data$Outcome)
```

```
ggplot(lda_df_predict, aes(x=LD1, fill=Outcome)) + geom_density(alpha=0.5)
```



2.2.2 Model creation

We are going to get a training and a testing set to use when building some models. We split the modified dataset into Train (80%) and Test (20%), in order to predict whether a patient is diabetic or not, by building machine learning classification models.

```
set.seed(1815)
data_sampling_index <- createDataPartition(data$Outcome, times=1, p=0.8, list = FALSE)
```

```
train_data <- data[data_sampling_index, ]
test_data <- data[-data_sampling_index, ]
```

We analysed the proportion of training set and test set also.

```
prop.table(table(train_data$Outcome)) * 100
```

```
   False     True 
65.04065 34.95935 
```

```
prop.table(table(test_data$Outcome)) * 100
```

```
   False     True 
65.35948 34.64052 
```

2.2.3 Naive Bayes Model

The Naive Bayesian classifier is based on Bayes' theorem with the independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c). Naive Bayes classifier assume that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.

```
model_naiveb <- naiveBayes(Outcome ~ ., data = train_data)
prediction_naiveb <- predict(model_naiveb, test_data)
confusionmatrix_naiveb <- confusionMatrix(prediction_naiveb, test_data$Outcome)
confusionmatrix_naiveb
```

```
Confusion Matrix and Statistics

          Reference
Prediction False True
     False    78   17
     True     22   36
                                          
               Accuracy : 0.7451          
                 95% CI : (0.6684, 0.812) 
    No Information Rate : 0.6536          
    P-Value [Acc > NIR] : 0.009661        
                                          
                  Kappa : 0.4493          
                                          
 Mcnemar's Test P-Value : 0.521839        
                                          
            Sensitivity : 0.7800          
            Specificity : 0.6792          
         Pos Pred Value : 0.8211          
         Neg Pred Value : 0.6207          
             Prevalence : 0.6536          
         Detection Rate : 0.5098
   Detection Prevalence : 0.6209
      Balanced Accuracy : 0.7296

       'Positive' Class : False
```

The accuracy of this model can be seen through the confusion matrix. Important metrics to be identified are: Sensitivity (recall) represent the true positive rate: the proportions of actual positives correctly identified. Specificity is the true negative rate: the proportion of actual negatives correctly identified. Accuracy is the general score of the classifier model performance as it is the ratio of how many samples are correctly classified to all samples. F1 score: the harmonic mean of precision and sensitivity. Accuracy and F1 score would be used to compare the result with the benchmark model. Precision: the number of correct positive results divided by the number of all positive results returned by the classifier.

2.2.4 Logistic Regression Model

Logistic Regression is widely used for binary classification like (0,1). The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features).

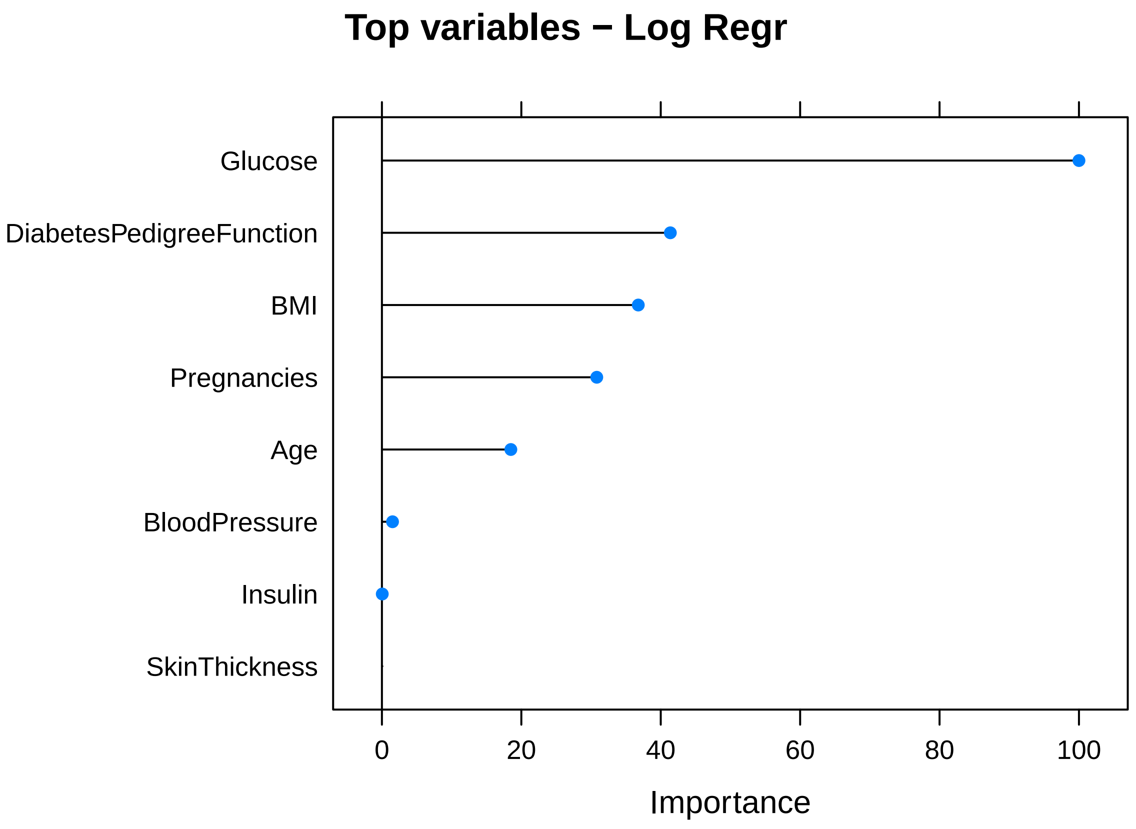
```
model_logreg <- train(Outcome~.,
                      data=train_data,
                      method="glm")
prediction_logreg<- predict(model_logreg,test_data)
# Check results
confusionmatrix_logreg <- confusionMatrix(prediction_logreg, test_data$Outcome)
confusionmatrix_logreg
```

```
Confusion Matrix and Statistics

          Reference
Prediction False True
     False    85   20
     True     15   33

               Accuracy : 0.7712
                 95% CI : (0.6965, 0.8352)
    No Information Rate : 0.6536
    P-Value [Acc > NIR] : 0.001098

                  Kappa : 0.4834

 Mcnemar's Test P-Value : 0.498962

            Sensitivity : 0.8500
            Specificity : 0.6226
         Pos Pred Value : 0.8095
         Neg Pred Value : 0.6875
             Prevalence : 0.6536
         Detection Rate : 0.5556
   Detection Prevalence : 0.6863
```

```
      Balanced Accuracy : 0.7363

       'Positive' Class : False
```

The most important variables that permit the best prediction and contribute the most to the model are the following:

```
plot(varImp(model_logreg), main="Top variables - Log Regr")
```



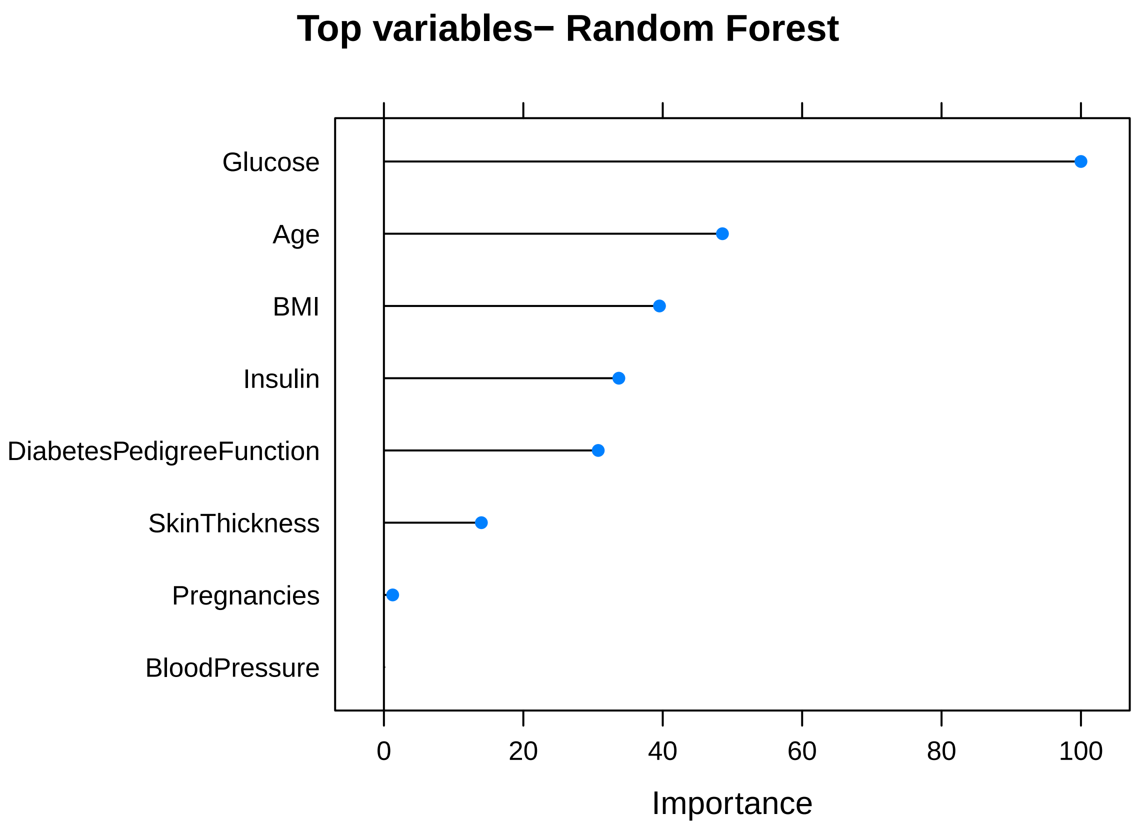
2.2.5 Random Forest Model

Random forests are a very popular machine learning approach that addresses the shortcomings of decision trees using a clever idea. The goal is to improve prediction performance and reduce instability by averaging multiple decision trees (a forest of trees constructed with randomness). Random forest is another ensemble method based on decision trees. It split data into sub-samples, trains decision tree classifiers on each sub-sample and averages prediction of each classifier. Splitting dataset causes higher bias but it is compensated by large decrease in variance. Random Forest is a supervised learning algorithm and it is flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of it's simplicity and the fact that it can be used for both classification and regression tasks. Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

```
model_randomforest <- train(Outcome~.,
                            data=train_data,
                            method="rf")
prediction_randomforest <- predict(model_randomforest, test_data)
#Check results
confusionmatrix_randomforest <- confusionMatrix(prediction_randomforest, test_data$Outcome)
confusionmatrix_randomforest
```

```
Confusion Matrix and Statistics

          Reference
Prediction False True
     False    84   25
     True     16   28

               Accuracy : 0.732
                 95% CI : (0.6545, 0.8003)
    No Information Rate : 0.6536
    P-Value [Acc > NIR] : 0.02367

                  Kappa : 0.3836

 Mcnemar's Test P-Value : 0.21152

            Sensitivity : 0.8400
            Specificity : 0.5283
         Pos Pred Value : 0.7706
         Neg Pred Value : 0.6364
             Prevalence : 0.6536
         Detection Rate : 0.5490
   Detection Prevalence : 0.7124
      Balanced Accuracy : 0.6842

       'Positive' Class : False
```

```
plot(varImp(model_randomforest), main="Top variables- Random Forest")
```



2.2.6 K Nearest Neighbor (KNN) Model

KNN (K-Nearest Neighbors) is one of many (supervised learning) algorithms used in data mining and machine learning, it's a classifier algorithm where the learning is based on "how similar" is a data from other. K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions).

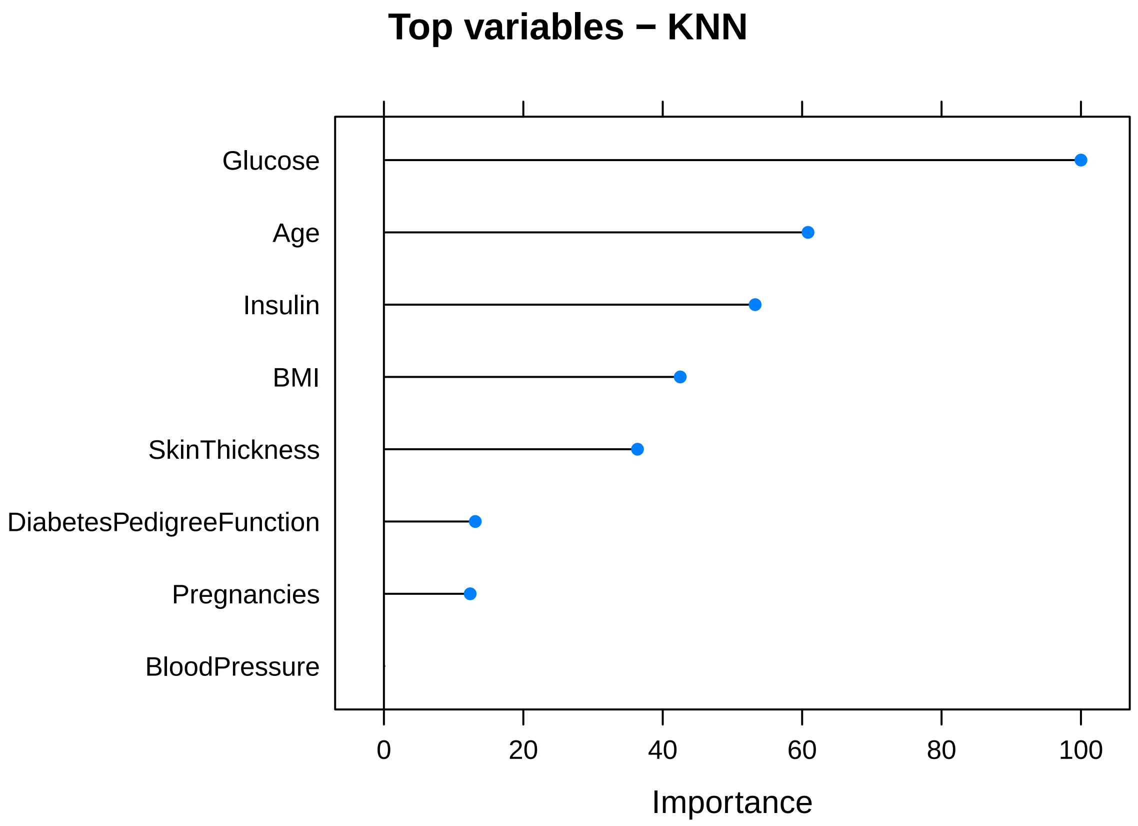
```
model_knn <- train(Outcome~.,
                   data=train_data,
                   method="knn",
                   tuneLength=5) #The tuneLength parameter tells the algorithm to try different default
                   #In this case we used 5 default values
prediction_knn <- predict(model_knn, test_data)
confusionmatrix_knn <- confusionMatrix(prediction_knn, test_data$Outcome)
confusionmatrix_knn
```

```
Confusion Matrix and Statistics

          Reference
Prediction False True
     False    80   19
     True     20   34

               Accuracy : 0.7451
```

```
                95% CI : (0.6684, 0.812)
    No Information Rate : 0.6536
    P-Value [Acc > NIR] : 0.009661

                  Kappa : 0.4396

 Mcnemar's Test P-Value : 1.000000

            Sensitivity : 0.8000
            Specificity : 0.6415
         Pos Pred Value : 0.8081
         Neg Pred Value : 0.6296
             Prevalence : 0.6536
         Detection Rate : 0.5229
   Detection Prevalence : 0.6471
      Balanced Accuracy : 0.7208

       'Positive' Class : False
```

The most important variables that permit the best prediction and contribute the most to the model are the following:

```
plot(varImp(model_knn), main="Top variables - KNN")
```



2.2.7 Neural Network with PCA Model

Artificial Neural Networks (NN) are a types of mathematical algorithms originating from the simulation of networks of biological neurons. An artificial Neural Network consists of nodes (called neurons) and edges (called synapses). Input data is transmitted through the weighted synapses to the neurons where calculations are processed and then either sent to further neurons or represent the output.

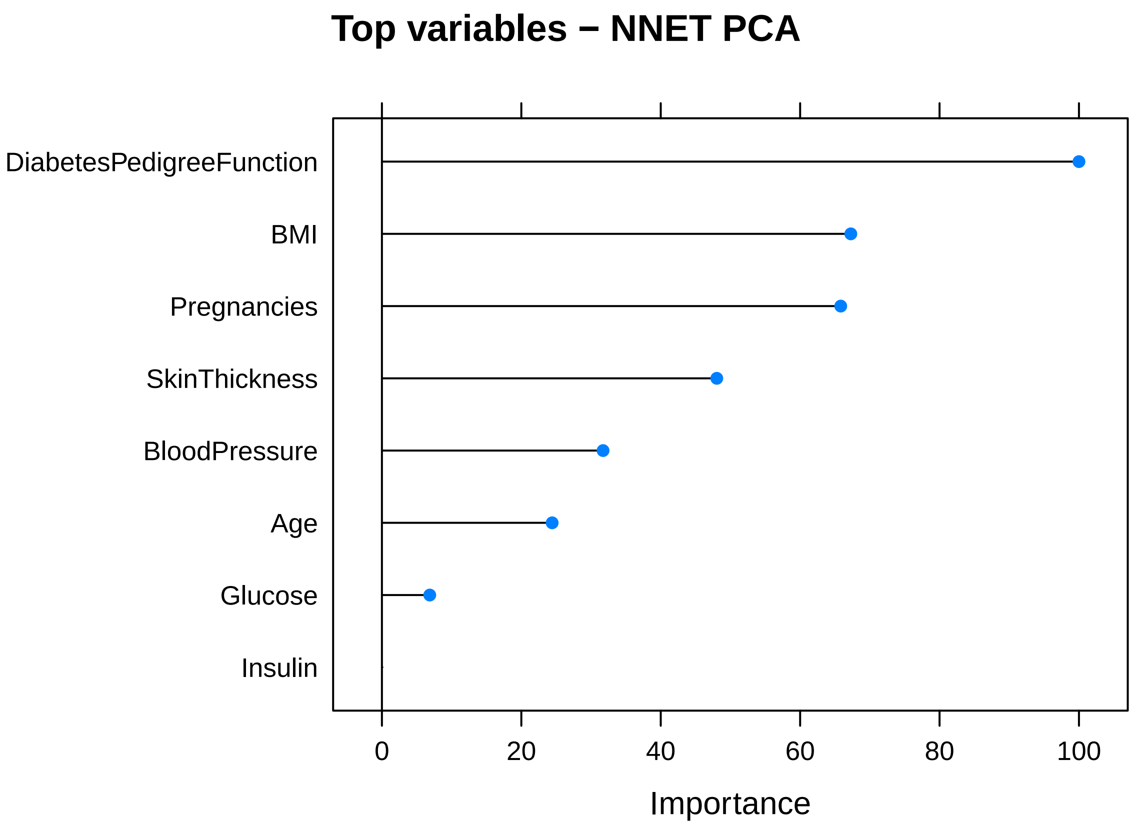
Neural Networks take in the weights of connections between neurons. The weights are balanced, learning data point in the wake of learning data point . When all weights are trained, the neural network can be utilized to predict the class or a quantity, if there should arise an occurrence of regression of a new input data point. With Neural networks, extremely complex models can be trained and they can be utilized as a kind of black box, without playing out an unpredictable complex feature engineering before training the model. Joined with the "deep approach" even more unpredictable models can be picked up to realize new possibilities.

```
model_nnet_pca <- train(Outcome~.,
                        data=train_data,
                        method="nnet",
                        trace=FALSE)
prediction_nnet_pca <- predict(model_nnet_pca, test_data)
confusionmatrix_nnet_pca <- confusionMatrix(prediction_nnet_pca, test_data$Outcome)
confusionmatrix_nnet_pca
```

```
Confusion Matrix and Statistics

          Reference
Prediction False True
     False    85   20
     True     15   33

               Accuracy : 0.7712
                 95% CI : (0.6965, 0.8352)
    No Information Rate : 0.6536
    P-Value [Acc > NIR] : 0.001098

                  Kappa : 0.4834

 Mcnemar's Test P-Value : 0.498962

            Sensitivity : 0.8500
            Specificity : 0.6226
         Pos Pred Value : 0.8095
         Neg Pred Value : 0.6875
             Prevalence : 0.6536
         Detection Rate : 0.5556
   Detection Prevalence : 0.6863
      Balanced Accuracy : 0.7363

       'Positive' Class : False
```

The most important variables that permit the best prediction and contribute the most to the model are the following:

```
plot(varImp(model_nnet_pca), main="Top variables - NNET PCA")
```



2.2.8 Neural Network with LDA Model

We are going to create a training and test set of LDA data created in previous chapters:

```
train_data_lda <- lda_df_predict[data_sampling_index, ]
test_data_lda <- lda_df_predict[-data_sampling_index, ]
```

```
model_nnet_lda <- train(Outcome~.,
                        data = train_data_lda,
                        method="nnet",
                        trace=FALSE)
prediction_nnet_lda <- predict(model_nnet_lda, test_data_lda)
confusionmatrix_nnet_lda <- confusionMatrix(prediction_nnet_lda, test_data_lda$Outcome)
confusionmatrix_nnet_lda
```

```
Confusion Matrix and Statistics

          Reference
Prediction False True
     False    84   19
     True     16   34
```

```
               Accuracy : 0.7712
                 95% CI : (0.6965, 0.8352)
    No Information Rate : 0.6536
    P-Value [Acc > NIR] : 0.001098

                  Kappa : 0.488

 Mcnemar's Test P-Value : 0.735317

            Sensitivity : 0.8400
            Specificity : 0.6415
         Pos Pred Value : 0.8155
         Neg Pred Value : 0.6800
             Prevalence : 0.6536
         Detection Rate : 0.5490
   Detection Prevalence : 0.6732
      Balanced Accuracy : 0.7408

       'Positive' Class : False
```

Chapter 3

Results

We can now compare and evaluate the results obtained for the above calculations.

```
confusionmatrix_list <- list(
  Naive_Bayes=confusionmatrix_naiveb,
  Logistic_regr=confusionmatrix_logreg,
  Random_Forest=confusionmatrix_randomforest,
  KNN=confusionmatrix_knn,
  Neural_PCA=confusionmatrix_nnet_pca,
  Neural_LDA=confusionmatrix_nnet_lda)
confusionmatrix_list_results <- sapply(confusionmatrix_list, function(x) x$byClass)
confusionmatrix_list_results %>% knitr::kable()
```

	Naive_Bayes	Logistic_regr	Random_Forest	KNN	Neural_PCA	Neural_LDA
Sensitivity	0.7800000	0.8500000	0.8400000	0.8000000	0.8500000	0.8400000
Specificity	0.6792453	0.6226415	0.5283019	0.6415094	0.6226415	0.6415094
Pos Pred Value	0.8210526	0.8095238	0.7706422	0.8080808	0.8095238	0.8155340
Neg Pred Value	0.6206897	0.6875000	0.6363636	0.6296296	0.6875000	0.6800000
Precision	0.8210526	0.8095238	0.7706422	0.8080808	0.8095238	0.8155340
Recall	0.7800000	0.8500000	0.8400000	0.8000000	0.8500000	0.8400000
F1	0.8000000	0.8292683	0.8038278	0.8040201	0.8292683	0.8275862
Prevalence	0.6535948	0.6535948	0.6535948	0.6535948	0.6535948	0.6535948
Detection Rate	0.5098039	0.5555556	0.5490196	0.5228758	0.5555556	0.5490196
Detection Prevalence	0.6209150	0.6862745	0.7124183	0.6470588	0.6862745	0.6732026
Balanced Accuracy	0.7296226	0.7363208	0.6841509	0.7207547	0.7363208	0.7407547

Chapter 4

Discussion

We will now describe the metrics that we will compare in this section.

Accuracy is the most important metric. It is the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to turn it into a percentage.

Precision is the number of True Positives divided by the number of True Positives and False Positives. Put another way, it is the number of positive predictions divided by the total number of positive class values predicted. It is also called the Positive Predictive Value (PPV). A low precision can also indicate a large number of False Positives.

Recall (Sensitivity) is the number of True Positives divided by the number of True Positives and the number of False Negatives. Put another way it is the number of positive predictions divided by the number of positive class values in the test data. It is also called Sensitivity or the True Positive Rate. Recall can be thought of as a measure of a classifiers completeness. A low recall indicates many False Negatives.

The F1 Score is the 2 x ((precision x recall) / (precision + recall)). It is also called the F Score or the F Measure. Put another way, the F1 score conveys the balance between the precision and the recall.

The best results for sensitivity (detection of diabetes) is Neural Network with LDA model which also has a great F1 score.

```r
confusionmatrix_results_max <- apply(confusionmatrix_list_results, 1, which.is.max)
output_report <- data.frame(metric=names(confusionmatrix_results_max),
                            best_model=colnames(confusionmatrix_list_results)[confusionmatrix_results_ma
                            value=mapply(function(x,y) {confusionmatrix_list_results[x,y]},
                                         names(confusionmatrix_results_max),
                                         confusionmatrix_results_max))
rownames(output_report) <- NULL
output_report
```

```
                metric    best_model     value
1          Sensitivity Logistic_regr 0.8500000
2          Specificity   Naive_Bayes 0.6792453
3       Pos Pred Value   Naive_Bayes 0.8210526
4       Neg Pred Value Logistic_regr 0.6875000
5            Precision   Naive_Bayes 0.8210526
6               Recall    Neural_PCA 0.8500000
7                   F1 Logistic_regr 0.8292683
8           Prevalence   Naive_Bayes 0.6535948
9       Detection Rate Logistic_regr 0.5555556
```

```
10 Detection Prevalence Random_Forest 0.7124183
11    Balanced Accuracy    Neural_LDA 0.7407547
```

Chapter 5

Conclusion

This paper treats the PIMA Indian Diabetes diagnosis problem as a pattern classification problem. In this report we investigated several machine learning model and we selected the optimal model by selecting a high accuracy level combined with a low rate of false-negatives (the means that the metric is high sensitivity).

The Optimal algorithm for Sensitivity, F1, Balanced Accuracy and other metrics are given in the previous table.

Chapter 6

Appendix - Environment

```r
print("Operating System:")
```

```
[1] "Operating System:"
```

```r
R.version
```

```
               _
platform       x86_64-w64-mingw32
arch           x86_64
os             mingw32
system         x86_64, mingw32
status
major          4
minor          1.2
year           2021
month          11
day            01
svn rev        81115
language       R
version.string R version 4.1.2 (2021-11-01)
nickname       Bird Hippie
```