American International University-Bangladesh (AIUB)

Department of Computer Science
Faculty of Science & Technology (FST)

Course name: Introduction to Data Science
Semester: Summer 22-23
Section: A

Finalterm Project

Submitted To: Tohedul Islam
Assistant Professor , FACULTY OF SCIENCE & TECHNOLOGY
AMERICAN INTERNATIONAL UNIVERSITY–BANGLADESH

Submitted By:

Student Name	**ID**
MD. KAMRUZZAMAN	20-44222-3

Date Of Submission: August 16, 2023

About the dataset: This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether 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. All patients here are females.

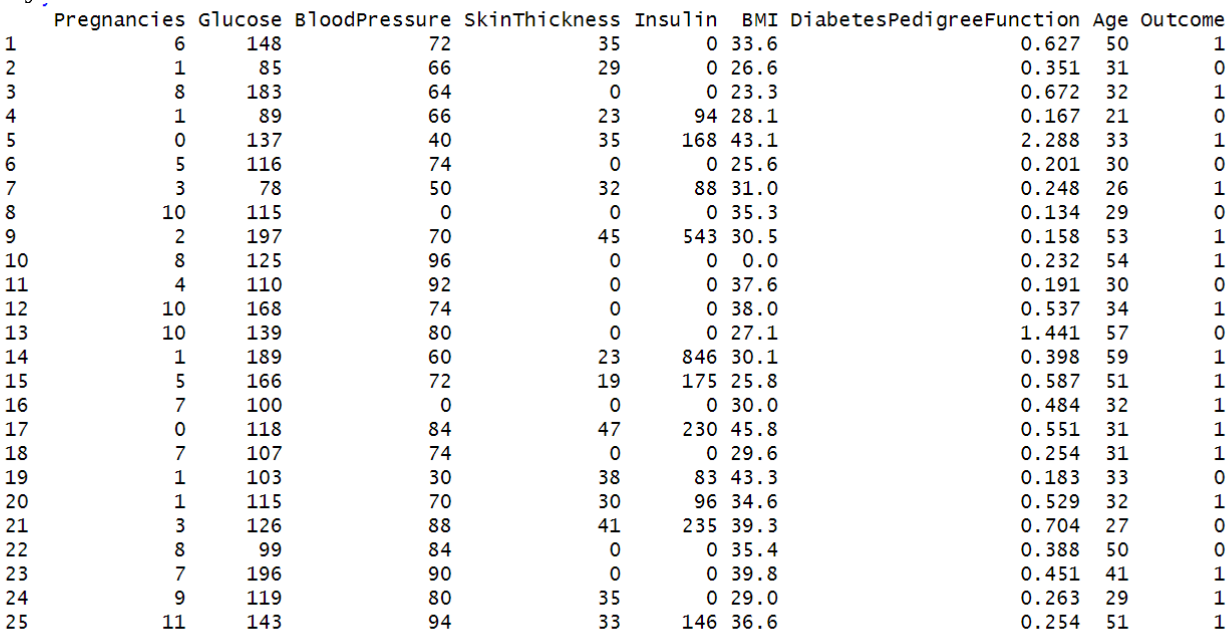
From the data set in the (.csv) file we can find several variables, some of them are independent (several medical predictor variables) and only one target dependent variable (Outcome).

Here are the details of dataset attributes:
Pregnancies - To express the Number of pregnancies.
Glucose - To express the Glucose level in blood .
BloodPressure - To express the Blood pressure measurement.
SkinThickness - To express the thickness of the skin.
Insulin - To express the Insulin level in blood.
BMI - To express the Body mass index.
DiabetesPedigreeFunction - To express the Diabetes percentage.
Age - To express the age.
Outcome - To express the final result 1 is Yes and 0 is No.

Link of the Dataset: https://www.kaggle.com/datasets/whenamancodes/predict-diabities

Data Import and view

Code:
mydata<-read.csv("E:/Academic/AIUB/Semester 9/Data Science/Final/Project/MyProject/Dataset/diabetes.csv",header=TRUE,sep=",")
mydata



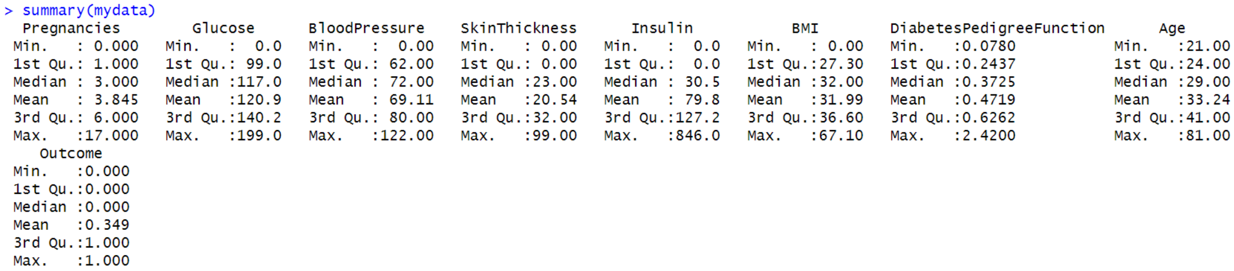
```
   Pregnancies Glucose BloodPressure SkinThickness Insulin  BMI DiabetesPedigreeFunction Age Outcome
1            6     148            72            35       0 33.6                    0.627  50       1
2            1      85            66            29       0 26.6                    0.351  31       0
3            8     183            64             0       0 23.3                    0.672  32       1
4            1      89            66            23      94 28.1                    0.167  21       0
5            0     137            40            35     168 43.1                    2.288  33       1
6            5     116            74             0       0 25.6                    0.201  30       0
7            3      78            50            32      88 31.0                    0.248  26       1
8           10     115             0             0       0 35.3                    0.134  29       0
9            2     197            70            45     543 30.5                    0.158  53       1
10           8     125            96             0       0  0.0                    0.232  54       1
11           4     110            92             0       0 37.6                    0.191  30       0
12          10     168            74             0       0 38.0                    0.537  34       1
13          10     139            80             0       0 27.1                    1.441  57       0
14           1     189            60            23     846 30.1                    0.398  59       1
15           5     166            72            19     175 25.8                    0.587  51       1
16           7     100             0             0       0 30.0                    0.484  32       1
17           0     118            84            47     230 45.8                    0.551  31       1
18           7     107            74             0       0 29.6                    0.254  31       1
19           1     103            30            38      83 43.3                    0.183  33       0
20           1     115            70            30      96 34.6                    0.529  32       1
21           3     126            88            41     235 39.3                    0.704  27       0
22           8      99            84             0       0 35.4                    0.388  50       0
23           7     196            90             0       0 39.8                    0.451  41       1
24           9     119            80            35       0 29.0                    0.263  29       1
25          11     143            94            33     146 36.6                    0.254  51       1
```

Description: Data are read from a CSV file and store it in a variable named mydata.

Summary of dataset

Code:

```
mydata<-read.csv("E:/Academic/AIUB/Semester 9/Data Science/Final/Project/MyProject/Dataset/diabetes.csv",header=TRUE,sep=",")
mydata
summary(mydata)
```



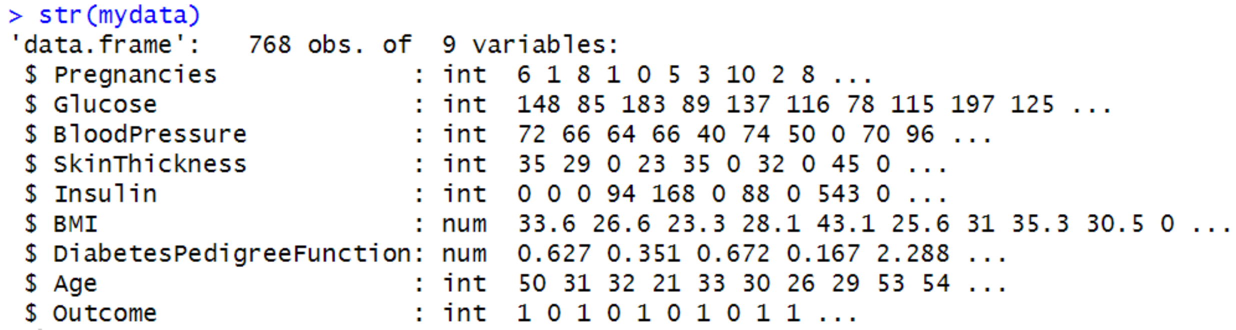
```
> summary(mydata)
  Pregnancies        Glucose       BloodPressure     SkinThickness      Insulin           BMI        DiabetesPedigreeFunction      Age       
 Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00   Min.   :  0.0   Min.   : 0.00   Min.   :0.0780           Min.   :21.00  
 1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00   1st Qu.:  0.0   1st Qu.:27.30   1st Qu.:0.2437           1st Qu.:24.00  
 Median : 3.000   Median :117.0   Median : 72.00   Median :23.00   Median : 30.5   Median :32.00   Median :0.3725           Median :29.00  
 Mean   : 3.845   Mean   :120.9   Mean   : 69.11   Mean   :20.54   Mean   : 79.8   Mean   :31.99   Mean   :0.4719           Mean   :33.24  
 3rd Qu.: 6.000   3rd Qu.:140.2   3rd Qu.: 80.00   3rd Qu.:32.00   3rd Qu.:127.2   3rd Qu.:36.60   3rd Qu.:0.6262           3rd Qu.:41.00  
 Max.   :17.000   Max.   :199.0   Max.   :122.00   Max.   :99.00   Max.   :846.0   Max.   :67.10   Max.   :2.4200           Max.   :81.00  
    Outcome     
 Min.   :0.000  
 1st Qu.:0.000  
 Median :0.000  
 Mean   :0.349  
 3rd Qu.:1.000  
 Max.   :1.000  
```

Description: This generates a summary of the mydata data using the summary() function. It's a useful function for initial data exploration and understanding the characteristics of your dataset.

Display dataset

Code:

```
mydata<-read.csv("E:/Academic/AIUB/Semester 9/Data Science/Final/Project/MyProject/Dataset/diabetes.csv",header=TRUE,sep=",")
mydata
str(mydata)
```

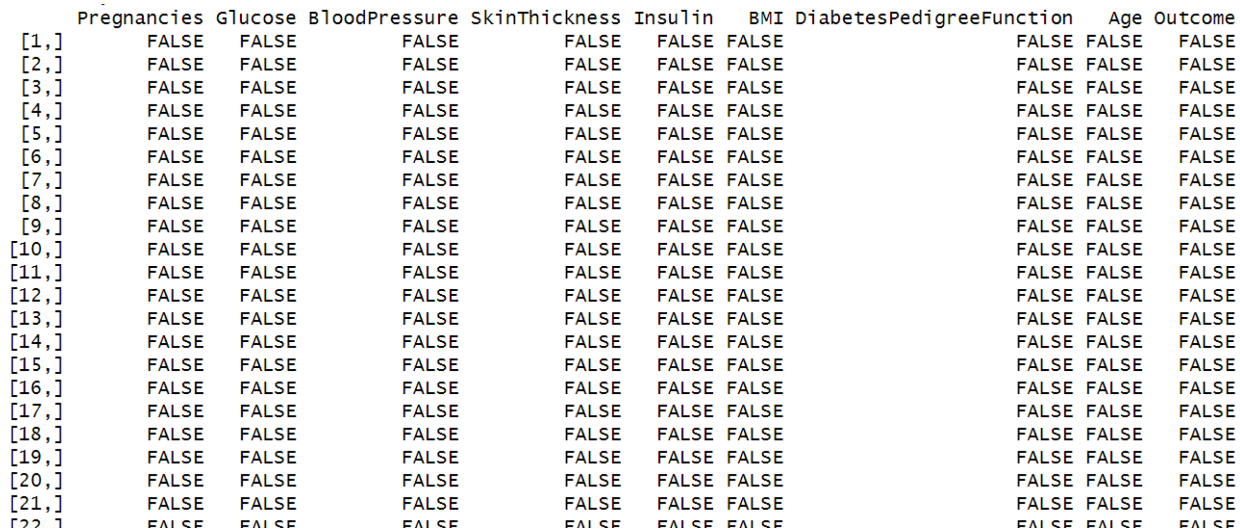


```
> str(mydata)
'data.frame':   768 obs. of  9 variables:
 $ Pregnancies             : int  6 1 8 1 0 5 3 10 2 8 ...
 $ Glucose                 : int  148 85 183 89 137 116 78 115 197 125 ...
 $ BloodPressure           : int  72 66 64 66 40 74 50 0 70 96 ...
 $ SkinThickness           : int  35 29 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                 : int  1 0 1 0 1 0 1 0 1 1 ...
```

Description: “str” displays structures of R objects. “str” used for displaying the contents of the data. str () is an alternative function to display the summary of the output produced, especially when the data set is huge.

Missing value detection

Code: is.na(mydata)

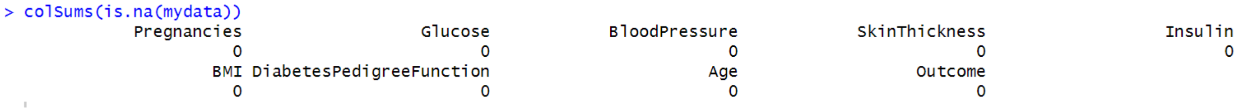


```
      Pregnancies Glucose BloodPressure SkinThickness Insulin   BMI DiabetesPedigreeFunction   Age Outcome
 [1,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
 [2,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
 [3,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
 [4,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
 [5,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
 [6,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
 [7,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
 [8,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
 [9,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[10,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[11,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[12,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[13,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[14,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[15,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[16,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[17,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[18,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[19,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[20,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[21,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
[22,]       FALSE   FALSE         FALSE         FALSE   FALSE FALSE                    FALSE FALSE   FALSE
```

Description: The is.na() function is used to identify missing values in a data frame or vector. It returns a logical vector of the same length as the input data, where each element is TRUE if the corresponding element in the data frame or vector is missing, and FALSE otherwise.

Missing Value count in each column

Code: colSums(is.na(mydata))



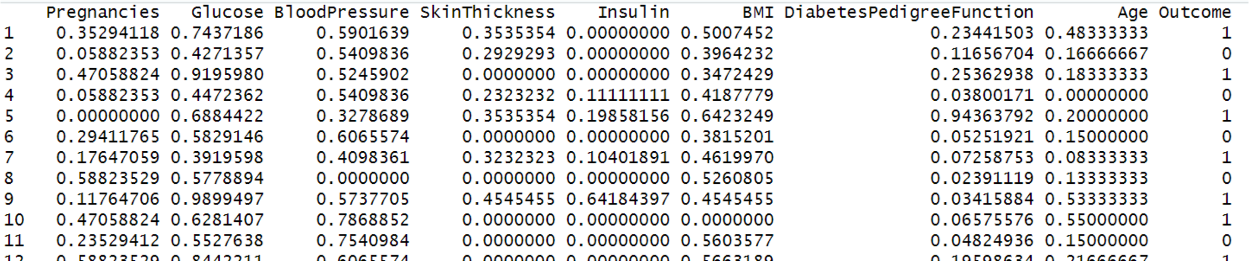
```
> colSums(is.na(mydata))
             Pregnancies                  Glucose            BloodPressure            SkinThickness                  Insulin 
                       0                        0                        0                        0                        0 
                     BMI DiabetesPedigreeFunction                      Age                  Outcome 
                       0                        0                        0                        0 
```

Description: The colSums() function is used to calculate the sum of elements for each column.

Normalization

Code: mydata

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
mydata <-as.data.frame(lapply(mydata[1:9],normalize))
mydata
```

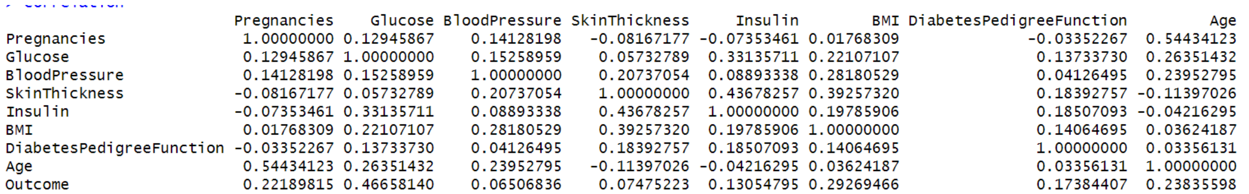


```
   Pregnancies   Glucose BloodPressure SkinThickness    Insulin       BMI DiabetesPedigreeFunction        Age Outcome
1   0.35294118 0.7437186     0.5901639     0.3535354 0.00000000 0.5007452               0.23441503 0.48333333       1
2   0.05882353 0.4271357     0.5409836     0.2929293 0.00000000 0.3964232               0.11656704 0.16666667       0
3   0.47058824 0.9195980     0.5245902     0.0000000 0.00000000 0.3472429               0.25362938 0.18333333       1
4   0.05882353 0.4472362     0.5409836     0.2323232 0.11111111 0.4187779               0.03800171 0.00000000       0
5   0.00000000 0.6884422     0.3278689     0.3535354 0.19858156 0.6423249               0.94363792 0.20000000       1
6   0.29411765 0.5829146     0.6065574     0.0000000 0.00000000 0.3815201               0.05251921 0.15000000       0
7   0.17647059 0.3919598     0.4098361     0.3232323 0.10401891 0.4619970               0.07258753 0.08333333       1
8   0.58823529 0.5778894     0.0000000     0.0000000 0.00000000 0.5260805               0.02391119 0.13333333       0
9   0.11764706 0.9899497     0.5737705     0.4545455 0.64184397 0.4545455               0.03415884 0.53333333       1
10  0.47058824 0.6281407     0.7868852     0.0000000 0.00000000 0.0000000               0.06575576 0.55000000       1
11  0.23529412 0.5527638     0.7540984     0.0000000 0.00000000 0.5603577               0.04824936 0.15000000       0
12  0.58823529 0.8442211     0.6065574     0.0000000 0.00000000 0.5663189               0.19598634 0.21666667       1
```

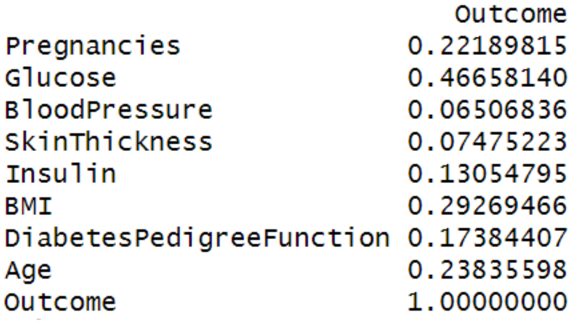
Description: My implemented normalization method known as "min-max normalization". This type of normalization scales the values of each variable to a range between 0 and 1 based on their minimum and maximum values. Data normalization can be performed using various methods, including manual calculations or through libraries like caret.

Correlation

Code: correlation<- cor(mydata)
correlation



```
                         Pregnancies    Glucose BloodPressure SkinThickness     Insulin        BMI DiabetesPedigreeFunction         Age
Pregnancies               1.00000000 0.12945867    0.14128198   -0.08167177 -0.07353461 0.01768309              -0.03352267  0.54434123
Glucose                   0.12945867 1.00000000    0.15258959    0.05732789  0.33135711 0.22107107               0.13733730  0.26351432
BloodPressure             0.14128198 0.15258959    1.00000000    0.20737054  0.08893338 0.28180529               0.04126495  0.23952795
SkinThickness            -0.08167177 0.05732789    0.20737054    1.00000000  0.43678257 0.39257320               0.18392757 -0.11397026
Insulin                  -0.07353461 0.33135711    0.08893338    0.43678257  1.00000000 0.19785906               0.18507093 -0.04216295
BMI                       0.01768309 0.22107107    0.28180529    0.39257320  0.19785906 1.00000000               0.14064695  0.03624187
DiabetesPedigreeFunction -0.03352267 0.13733730    0.04126495    0.18392757  0.18507093 0.14064695               1.00000000  0.03356131
Age                       0.54434123 0.26351432    0.23952795   -0.11397026 -0.04216295 0.03624187               0.03356131  1.00000000
Outcome                   0.22189815 0.46658140    0.06506836    0.07475223  0.13054795 0.29269466               0.17384407  0.23835598
```

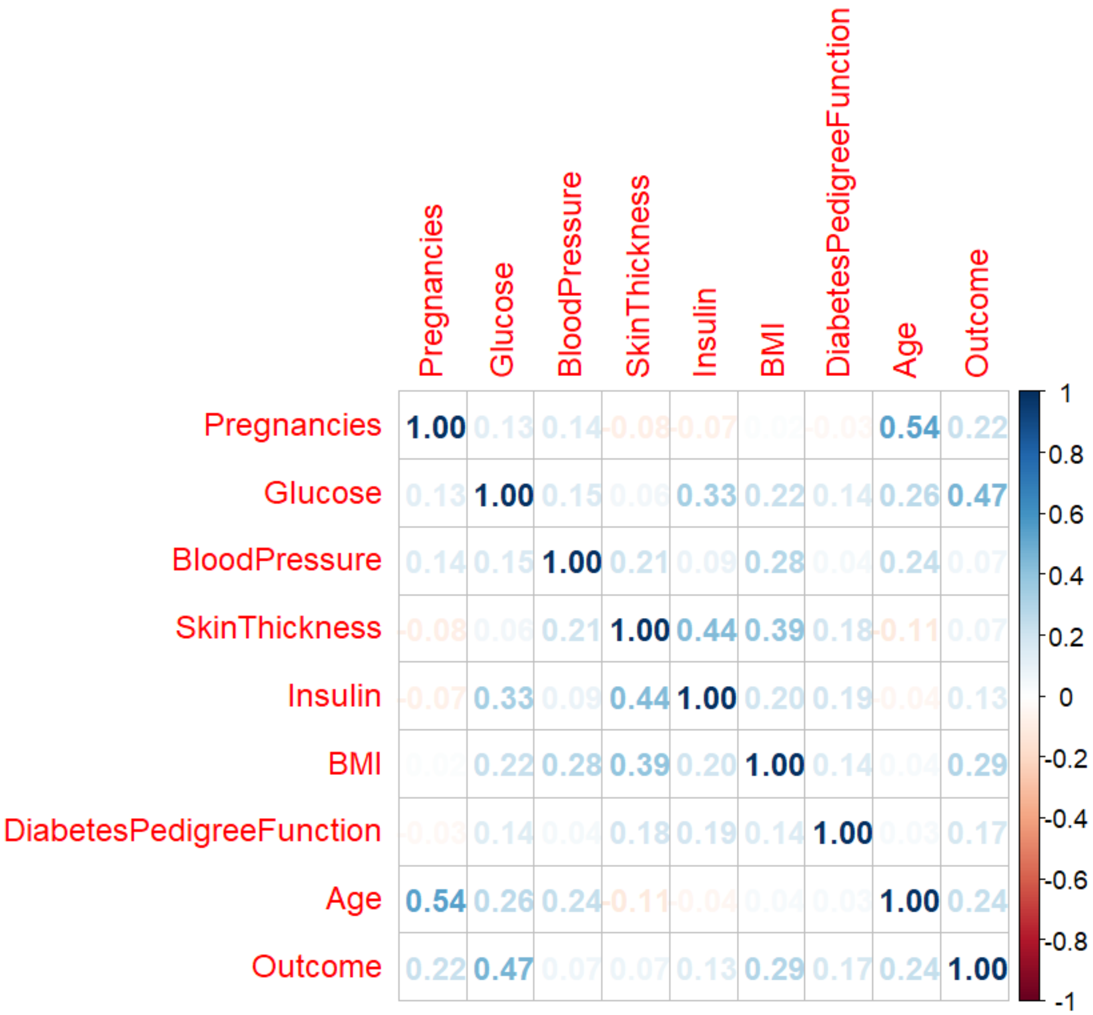


```
                            Outcome
Pregnancies              0.22189815
Glucose                  0.46658140
BloodPressure            0.06506836
SkinThickness            0.07475223
Insulin                  0.13054795
BMI                      0.29269466
DiabetesPedigreeFunction 0.17384407
Age                      0.23835598
Outcome                  1.00000000
```

Description: Calculated the correlation between variables using the cor() function. This function computes the correlation coefficient between pairs of variables in a data frame.

Correlation Plotting

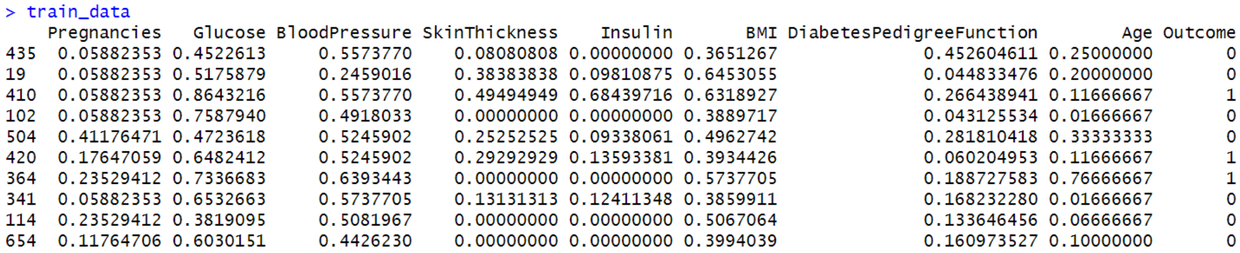
Code: install.packages("corrplot")
library(corrplot)
cor_matrix <- cor(mydata)
corrplot(cor_matrix, method = "number")



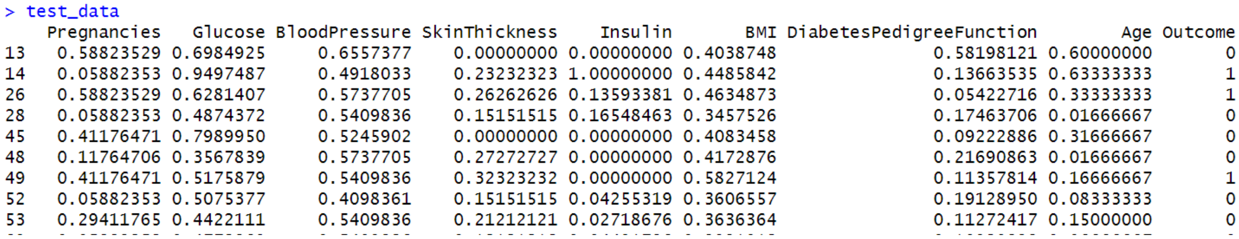
Description: Correlation plotting for better visualization.

Dividing dataset into training and test set

Code: train_indices<-sample(nrow(mydata), 0.8 * nrow(mydata))
train_indices
train_data<-mydata[train_indices,]
train_data
test_data<-mydata[-train_indices,]
test_data



```
> train_data
    Pregnancies   Glucose BloodPressure SkinThickness    Insulin       BMI DiabetesPedigreeFunction        Age Outcome
435  0.05882353 0.4522613     0.5573770    0.08080808 0.00000000 0.3651267              0.452604611 0.25000000       0
19   0.05882353 0.5175879     0.2459016    0.38383838 0.09810875 0.6453055              0.044833476 0.20000000       0
410  0.05882353 0.8643216     0.5573770    0.49494949 0.68439716 0.6318927              0.266438941 0.11666667       1
102  0.05882353 0.7587940     0.4918033    0.00000000 0.00000000 0.3889717              0.043125534 0.01666667       0
504  0.41176471 0.4723618     0.5245902    0.25252525 0.09338061 0.4962742              0.281810418 0.33333333       0
420  0.17647059 0.6482412     0.5245902    0.29292929 0.13593381 0.3934426              0.060204953 0.11666667       1
364  0.23529412 0.7336683     0.6393443    0.00000000 0.00000000 0.5737705              0.188727583 0.76666667       1
341  0.05882353 0.6532663     0.5737705    0.13131313 0.12411348 0.3859717              0.168232280 0.01666667       0
114  0.23529412 0.3819095     0.5081967    0.00000000 0.00000000 0.5067064              0.133646456 0.06666667       0
654  0.11764706 0.6030151     0.4426230    0.00000000 0.00000000 0.3994039              0.160973527 0.10000000       0
```

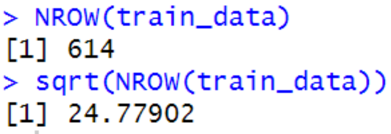


```
> test_data
    Pregnancies   Glucose BloodPressure SkinThickness    Insulin       BMI DiabetesPedigreeFunction         Age Outcome
13   0.58823529 0.6984925     0.6557377    0.00000000 0.00000000 0.4038748               0.58198121 0.60000000       0
14   0.05882353 0.9497487     0.4918033    0.23232323 1.00000000 0.4485842               0.13663535 0.63333333       1
26   0.58823529 0.6281407     0.5737705    0.26262626 0.13593381 0.4634873               0.05422716 0.33333333       1
28   0.05882353 0.4874372     0.5409836    0.15151515 0.16548463 0.3457526               0.17463706 0.01666667       0
45   0.41176471 0.7989950     0.5245902    0.00000000 0.00000000 0.4083458               0.09222886 0.31666667       0
48   0.11764706 0.3567839     0.5737705    0.27272727 0.00000000 0.4172876               0.21690863 0.01666667       0
49   0.41176471 0.5175879     0.5409836    0.32323232 0.00000000 0.5827124               0.11357814 0.16666667       1
52   0.05882353 0.5075377     0.4098361    0.15151515 0.04255319 0.3606557               0.19128950 0.08333333       0
53   0.29411765 0.4422111     0.5409836    0.21212121 0.02718676 0.3636364               0.11272417 0.15000000       0
```

Description: To split our data into 80% for training and 20% for testing set, sample() function used.

Find the total number of observation and value of K

Code: NROW(train_data)
sqrt(NROW(train_data))

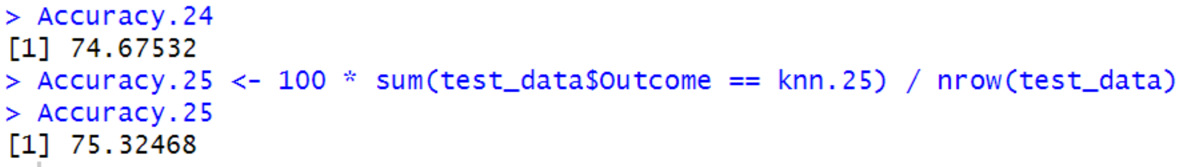


```
> NROW(train_data)
[1] 614
> sqrt(NROW(train_data))
[1] 24.77902
```

Description: Calculated the number of observations in the training data set. The reason for doing this is to initialize the value of 'K' in the KNN model. One of the ways to find the optimal K value is to calculate the square root of the total number of observations in the data set. This square root will give the 'K' value.

Accuracy of dividing the data into training and test set approach

Code: knn.24 <- knn(train = train_data[, -9], test = test_data[, -9], cl = train_data$Outcome, k = 24)
knn.25 <- knn(train = train_data[, -9], test = test_data[, -9], cl = train_data$Outcome, k = 25)
Accuracy.24 <- 100 * sum(test_data$Outcome == knn.24) / nrow(test_data)
Accuracy.24
Accuracy.25 <- 100 * sum(test_data$Outcome == knn.25) / nrow(test_data)
Accuracy.25



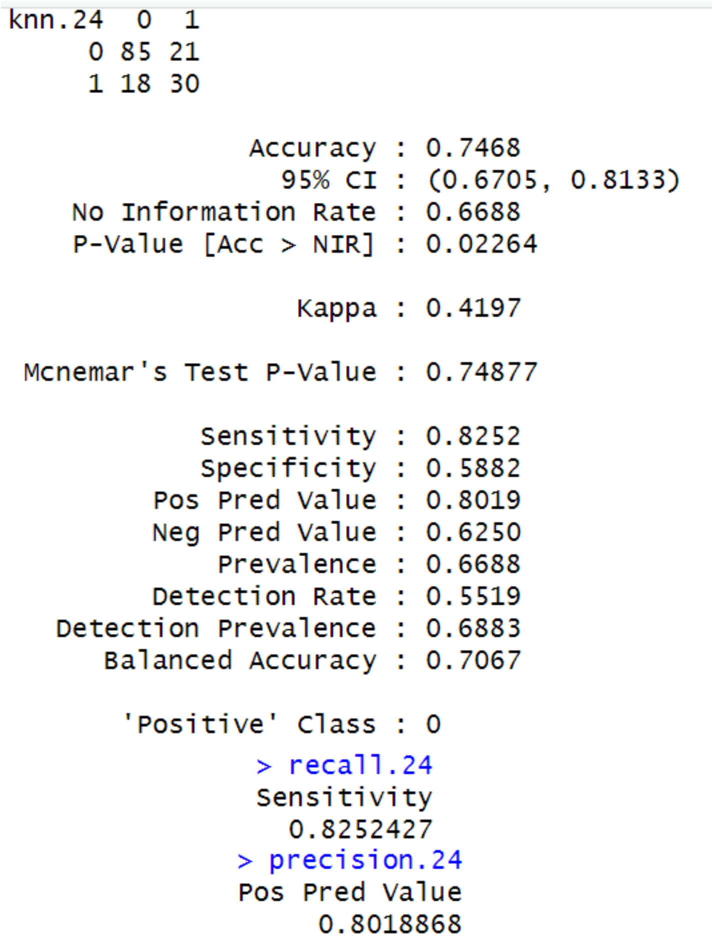
```
> Accuracy.24
[1] 74.67532
> Accuracy.25 <- 100 * sum(test_data$Outcome == knn.25) / nrow(test_data)
> Accuracy.25
[1] 75.32468
```

Description: Calculated the accuracy of dividing the data into training and test set approach.

Confusion matrix of dividing the data into training and test set approach

Code: For K=24
confusion_matrix.24 <- table(knn.24, test_data$Outcome)
confusion_matrix.24
confusion_matrix_summary.24 <- confusionMatrix(confusion_matrix.24)

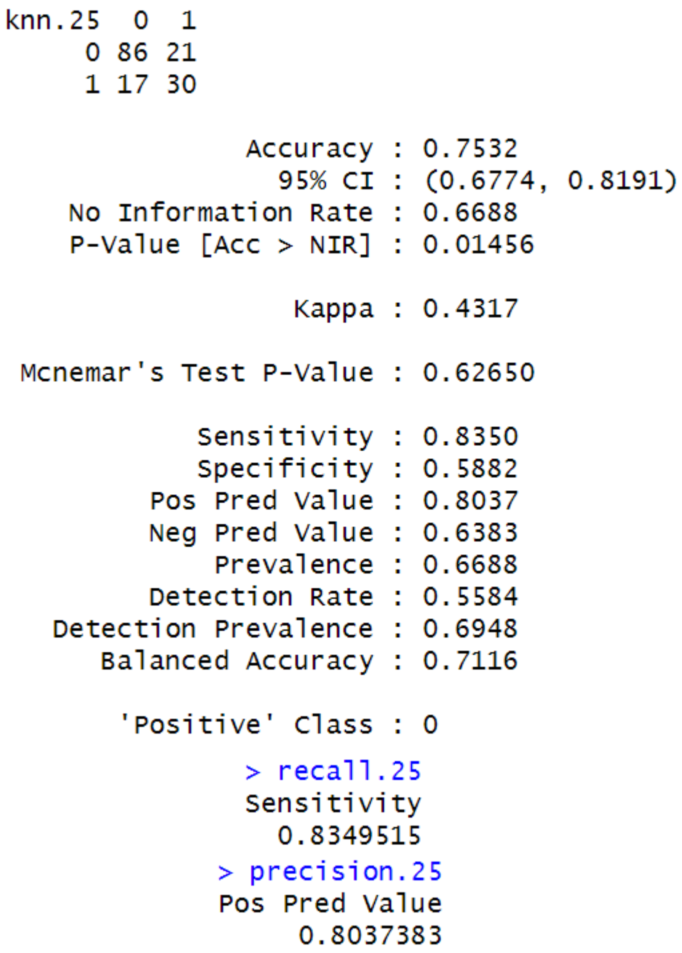
confusion_matrix_summary.24
recall.24 <- confusion_matrix_summary.24$byClass["Sensitivity"]
recall.24
precision.24 <- confusion_matrix_summary.24$byClass["Pos Pred Value"]
precision.24



```
knn.24  0  1
     0 85 21
     1 18 30

               Accuracy : 0.7468
                 95% CI : (0.6705, 0.8133)
    No Information Rate : 0.6688
    P-Value [Acc > NIR] : 0.02264

                  Kappa : 0.4197

 Mcnemar's Test P-Value : 0.74877

            Sensitivity : 0.8252
            Specificity : 0.5882
         Pos Pred Value : 0.8019
         Neg Pred Value : 0.6250
             Prevalence : 0.6688
         Detection Rate : 0.5519
   Detection Prevalence : 0.6883
      Balanced Accuracy : 0.7067

       'Positive' Class : 0
```

```
> recall.24
Sensitivity
  0.8252427
> precision.24
Pos Pred Value
     0.8018868
```

Code: For K=25
confusion_matrix.25 <- table(knn.25, test_data$Outcome)
confusion_matrix.25
confusion_matrix_summary.25 <- confusionMatrix(confusion_matrix.25)
confusion_matrix_summary.25



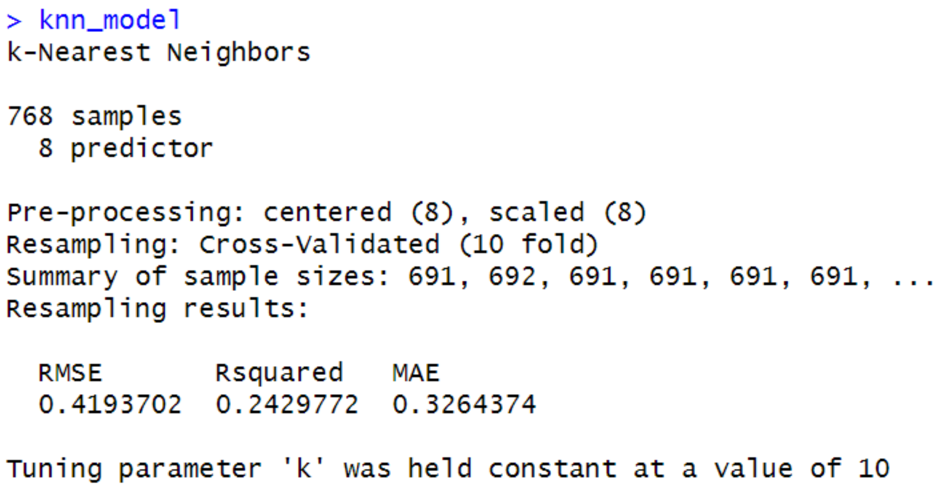
```
knn.25  0  1
     0 86 21
     1 17 30

               Accuracy : 0.7532
                 95% CI : (0.6774, 0.8191)
    No Information Rate : 0.6688
    P-Value [Acc > NIR] : 0.01456

                  Kappa : 0.4317

 Mcnemar's Test P-Value : 0.62650

            Sensitivity : 0.8350
            Specificity : 0.5882
         Pos Pred Value : 0.8037
         Neg Pred Value : 0.6383
             Prevalence : 0.6688
         Detection Rate : 0.5584
   Detection Prevalence : 0.6948
      Balanced Accuracy : 0.7116

       'Positive' Class : 0

> recall.25
Sensitivity
  0.8349515
> precision.25
Pos Pred Value
     0.8037383
```

Description: Confusion matrix generated to calculate the accuracy of the KNN model with K value set to 24 and 25 respectively. Recall and Precision value also calculated for this KNN classifier.

10-fold cross validation

Code:

```
ctrl <- trainControl(method = "cv", number = 10)
knn_model <- train(
  Outcome ~ .,
  data = mydata,
  method = "knn",
  trControl = ctrl,
  preProcess = c("center", "scale"),
  tuneGrid = data.frame(k =10)
)
knn_model
```



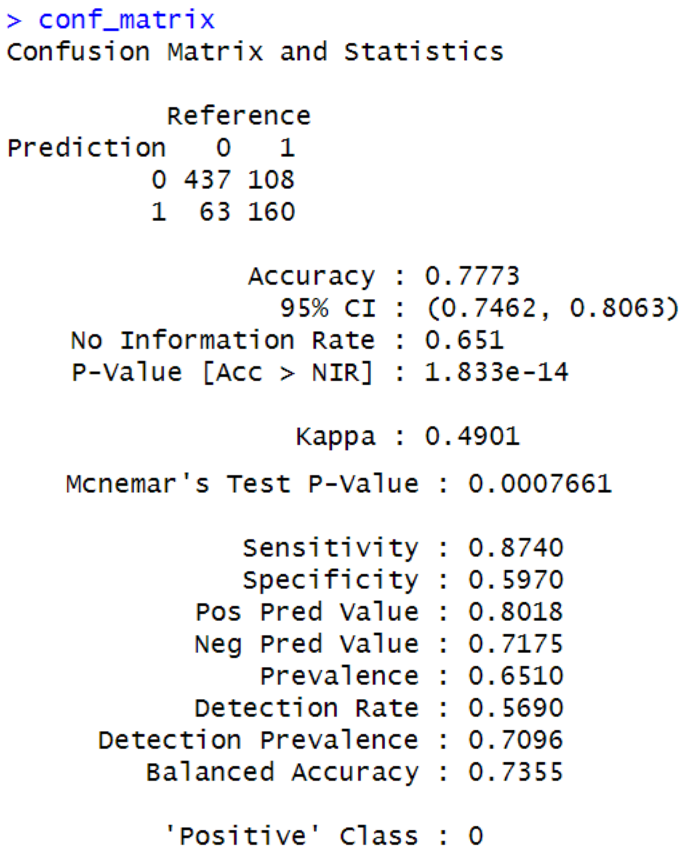
```
> knn_model
k-Nearest Neighbors 

768 samples
  8 predictor

Pre-processing: centered (8), scaled (8) 
Resampling: Cross-Validated (10 fold) 
Summary of sample sizes: 691, 692, 691, 691, 691, 691, ... 
Resampling results:

  RMSE       Rsquared   MAE      
  0.4193702  0.2429772  0.3264374

Tuning parameter 'k' was held constant at a value of 10
```

Description: 10-fold cross validation approach implemented here and Root Mean Square Error, R-squared (Coefficient of Determination), Mean Absolute Error calculated.

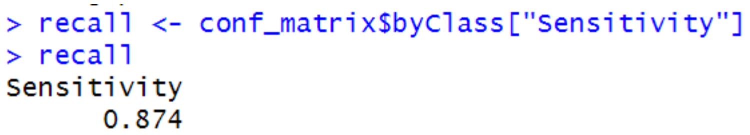
10-fold cross validation Confusion matrix

Code:

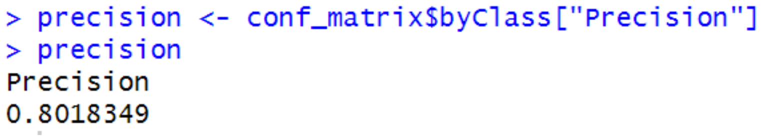
```
ctrl <- trainControl(method = "cv", number = 10)
knn_model <- train(
  Outcome ~ .,
  data = mydata,
  method = "knn",
  trControl = ctrl,
  preProcess = c("center", "scale"),
  tuneGrid = data.frame(k =10)
)
knn_model

conf_matrix
```



```
> conf_matrix
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 437 108
         1  63 160

               Accuracy : 0.7773
                 95% CI : (0.7462, 0.8063)
    No Information Rate : 0.651
    P-Value [Acc > NIR] : 1.833e-14

                  Kappa : 0.4901

 Mcnemar's Test P-Value : 0.0007661

            Sensitivity : 0.8740
            Specificity : 0.5970
         Pos Pred Value : 0.8018
         Neg Pred Value : 0.7175
             Prevalence : 0.6510
         Detection Rate : 0.5690
   Detection Prevalence : 0.7096
      Balanced Accuracy : 0.7355

       'Positive' Class : 0
```



```
> recall <- conf_matrix$byClass["Sensitivity"]
> recall
Sensitivity
      0.874
```



```
> precision <- conf_matrix$byClass["Precision"]
> precision
Precision
0.8018349
```

<u>Description:</u> Confusion matrix generated to calculate the accuracy of the KNN model with K value set to 10. Recall and Precision value also calculated for this KNN classifier.