Data set

This data set 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. In particular, all patients here are females at least 21 years old of Pima Indian heritage.

Columns Description:

- Pregnancies- To express the Number of pregnancies
- Glucose- To express the Glucose level in blood
- BloodPressure (diastolic)- To express the Blood pressure measurement (mmHg)
- SkinThickness- To express the thickness of the skin (mm)
- Insulin- To express the Insulin level in blood (U/mL)
- BMI- To express the Body mass index (weight in kg/(height in m)^2)
- DiabetesPedigreeFunction- To express the Diabetes percentage (The pedigree provides a synthesis of the diabetes mellitus history in ancestors and the genetic relationship with the subject. It utilizes information from a person's family history to predict how likely a subject can get diabetes)
- Age- To express the age (years)
- Outcome- To express the final result 1 is Yes and 0 is No

This data contains information about year 1990.

Some important information is printed below:

```
diabetes = read.csv("Diabetes.csv",sep = ",", dec = ".") #Read the csv data
attach(diabetes)
names(diabetes)  #Add column names
```

```
## [1] "Pregnancies"              "Glucose"
## [3] "BloodPressure"            "SkinThickness"
## [5] "Insulin"                  "BMI"
## [7] "DiabetesPedigreeFunction" "Age"
## [9] "Outcome"
```

```
p = ncol(diabetes) - 1  #Number of features
p
```

```
## [1] 8
```

```
n = nrow(diabetes)  #Number of rows
n
```

```
## [1] 768
```

```
summary(diabetes) #Print a summary of the data and the features
```

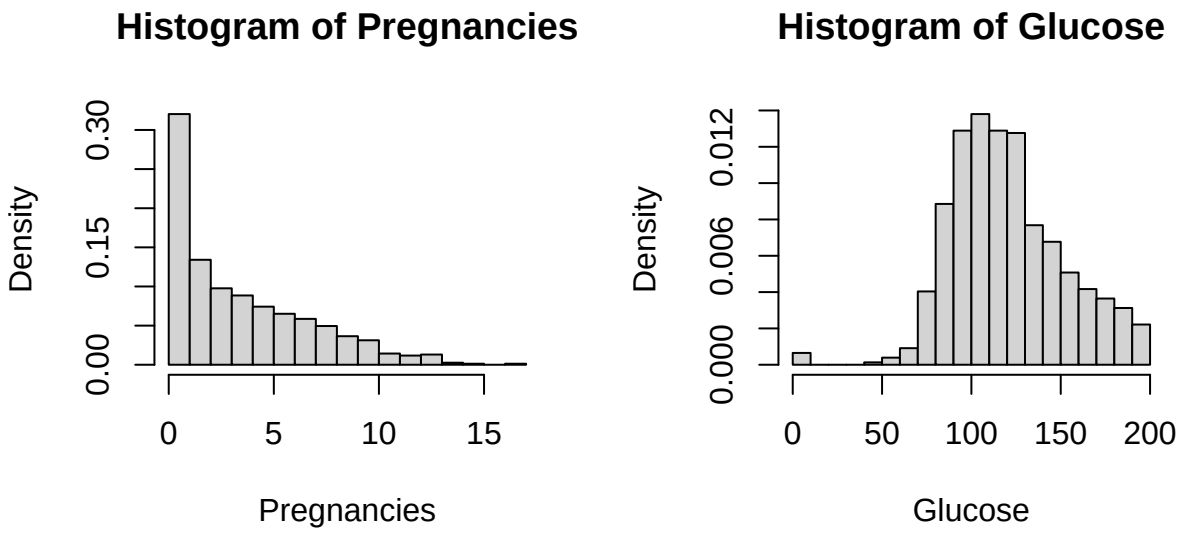
```
##   Pregnancies        Glucose      BloodPressure    SkinThickness
##  Min.   : 0.000   Min.   :  0.0   Min.   :  0.00   Min.   : 0.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.: 0.00
##  Median : 3.000   Median :117.0   Median : 72.00   Median :23.00
##  Mean   : 3.845   Mean   :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
##  Min.   :0.000
##  1st Qu.:0.000
##  Median :0.000
##  Mean   :0.349
##  3rd Qu.:1.000
##  Max.   :1.000
```

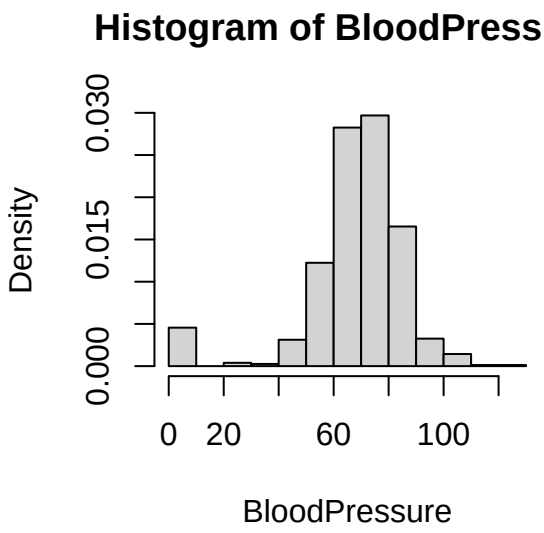
Graphical representation of the raw data

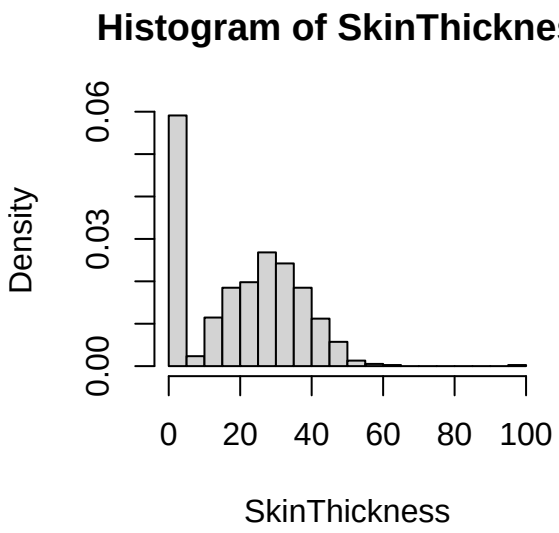
Below, the histograms of the quantitative features are presented .

```
par(mfrow = c(1,2))
hist(Pregnancies, freq = F, nclass = 16)
hist(Glucose, freq = F, nclass = 16)
```

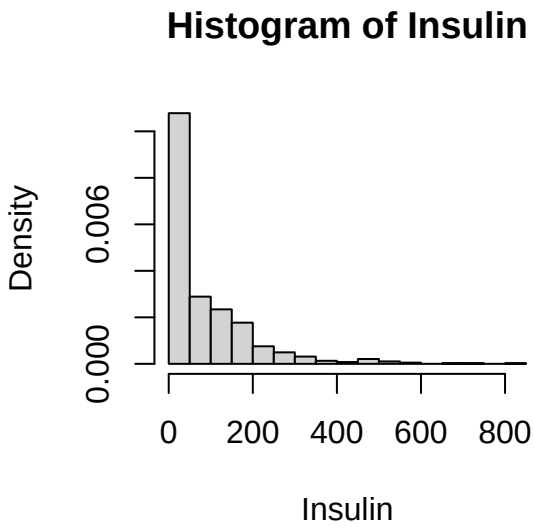


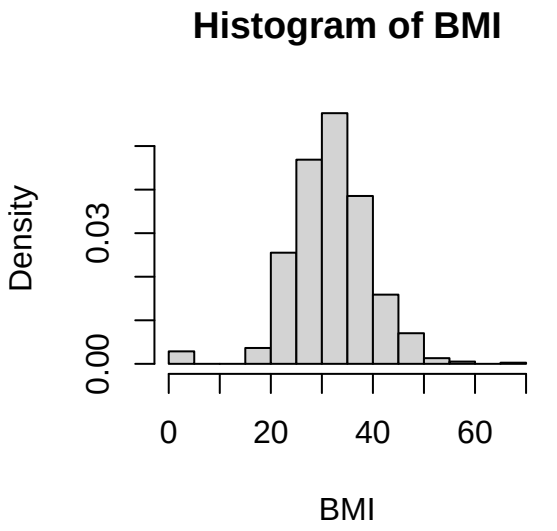

```
hist(BloodPressure, freq = F, nclass = 16)
hist(SkinThickness, freq = F, nclass = 16)
```



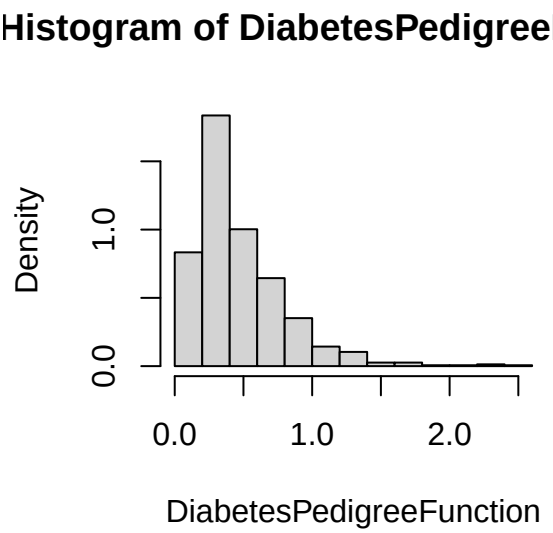



```
hist(Insulin, freq = F, nclass = 16)
hist(BMI, freq = F, nclass = 16)
```



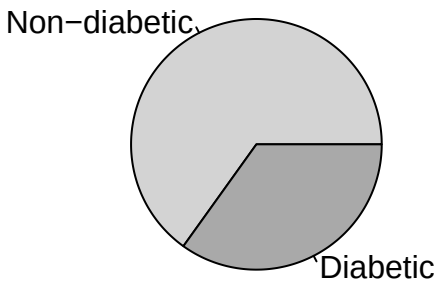



```
hist(DiabetesPedigreeFunction, freq = F, nclass = 16)
hist(Age, freq = F, nclass = 16)
```



Also, a pie chart of the outcome binary variable is shown.

```
pie(table(Outcome), labels = c('Non-diabetic','Diabetic'),edges=300,
    radius = 1,col=c('lightgrey','darkgrey'))
```



Some zero values seem to be out of the reasonable proportions or to be outliers. In the next step, will investigate which of these variables could or could not present zero values. Other values that will be investigated are pregnancies, as pregnancies above 10 which could imply some kind of outliers of the initial survey. In order to investigate this, the team will look up some Indian data, in particular of the Pima population. Also, it can be seen that in the studied population, there is not a huge difference between diabetic and non-diabetic percentages, so no there is a good balance between the two classes in the study.

Data imputation

Some of the variables are impossible to be zero, these are:

1. Glucose: A person with low glucose it is likely to pass out, he will surely die without glucose
2. Diastolic blood pressure: no blood pressure means no heartbeat
3. Skin thickness: 0 mm means no skin
4. Insulin: a person below 40 U/mL is likely to enter in coma
5. BMI: as it is defined, 0 means no weight

Implying that those zeros values that clearly show up in the histogram, must be imputed later.

```
mean(Pregnancies)
```

```
## [1] 3.845052
```

```
sd(Pregnancies)
```

```
## [1] 3.369578
```

When evaluating the values of the feature Pregnancies, the mean of the population seems to be right according to the World bank data for 1990 https://data.worldbank.org/indicator/SP.DYN.TFRT.IN?locations=IN&view=chart, which states that the mean of the pregnancies per woman is about 4.

Other evaluation that can be carried out, is to compute if there is some abnormality with the age. In this case, the team propose to evaluate whether exist any instance in which Pregnancies*9/12 + 13 > Age.

```
table(Pregnancies [Pregnancies*9/12 + 13 > Age])
```

```
## < table of extent 0 >
```

Imputing zero values with Predictive Mean Matching

First, the zero values are changed to "NA" when the zero is physically impossible. Later, the values are imputed using the library *"mice"*. The method used to impute is the Predictive Mean Matching:

$$x_j = \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon_j$$

The variables with the more values to impute are:

```
NA_insulin <- sum(diabetes$Insulin==0)
NA_SkinThickness <- sum(diabetes$SkinThickness==0)
print(paste("The number of insulin missing values: ",NA_insulin))
```

```
## [1] "The number of insulin missing values:  374"
```

```
print(paste("The number of insulin missing values: ",NA_SkinThickness))
```

```
## [1] "The number of insulin missing values:  227"
```

```
diabetes[diabetes$Glucose==0,"Glucose"] <- NA
diabetes[diabetes$BloodPressure==0,"BloodPressure"] <- NA
diabetes[diabetes$SkinThickness==0,"SkinThickness"] <- NA
diabetes[diabetes$Insulin==0,"Insulin"] <- NA
diabetes[diabetes$BMI==0,"BMI"] <- NA
diabetes[diabetes$Age==0,"Age"] <- NA

n_miss <- apply(is.na(diabetes),1,sum)  #Calculate the number of NA per column
table(n_miss)
```

```
## n_miss
##   0   1   2   3   4
## 392 142 199  28   7
```

```
diabetes_imp <- mice(diabetes,m=1,method="pmm")
```

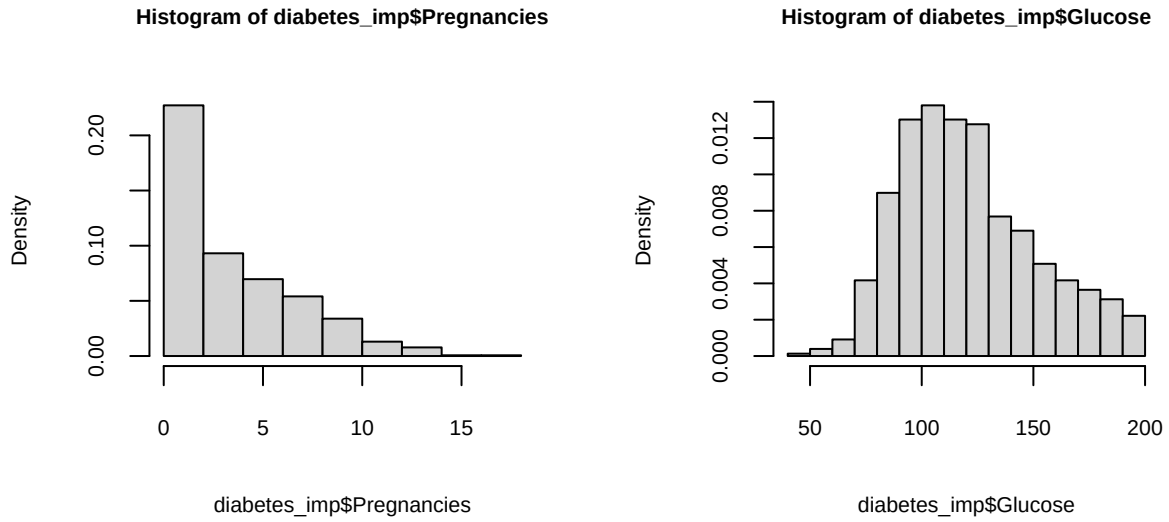
```
##
##  iter imp variable
##   1   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
##   2   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
##   3   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
##   4   1  Glucose  BloodPressure  SkinThickness  Insulin  BMI
##   5   1  Glucose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

```
diabetes_imp <- complete(diabetes_imp)
n <- nrow(diabetes_imp)
```

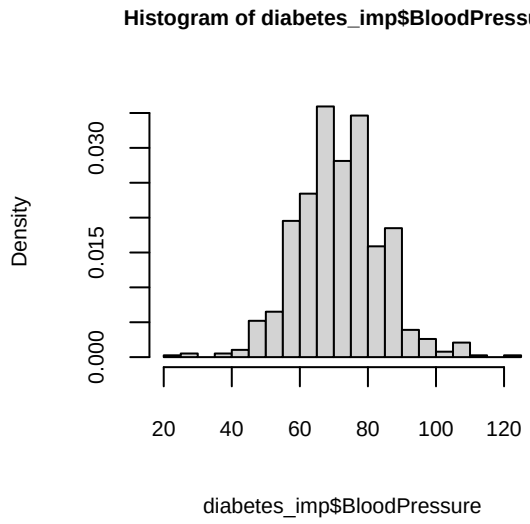
Now, almost every row have been imputed with the corresponding linear regression. Some few rows have been removed because still with the regression prediction, no reasonable physical values have been generated.

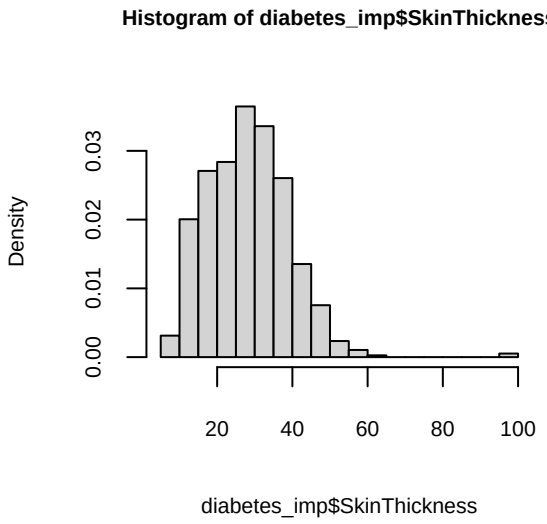
The resulting histograms for the imputed variables are shown below:

```
par(mfrow = c(1,2))
hist(diabetes_imp$Pregnancies, freq = F,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)
hist(diabetes_imp$Glucose, freq = F, nclass = 16,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)
```

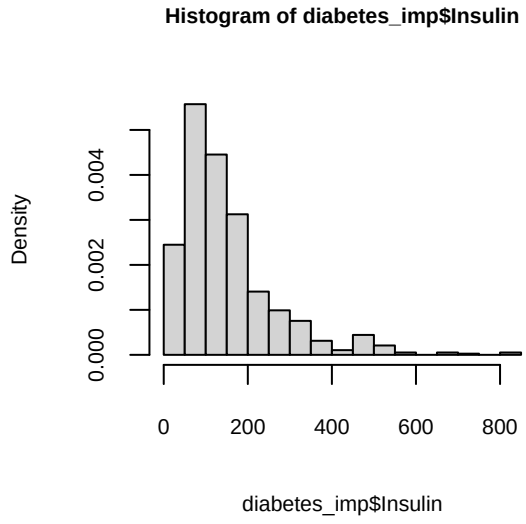


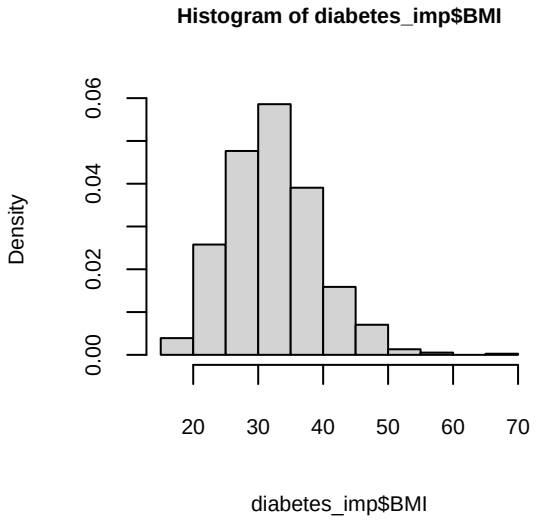

```
hist(diabetes_imp$BloodPressure, freq = F, nclass = 16,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7
hist(diabetes_imp$SkinThickness, freq = F, nclass = 16,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7
```



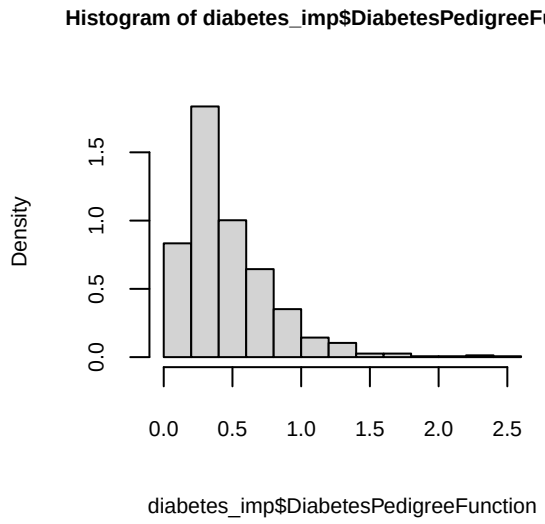


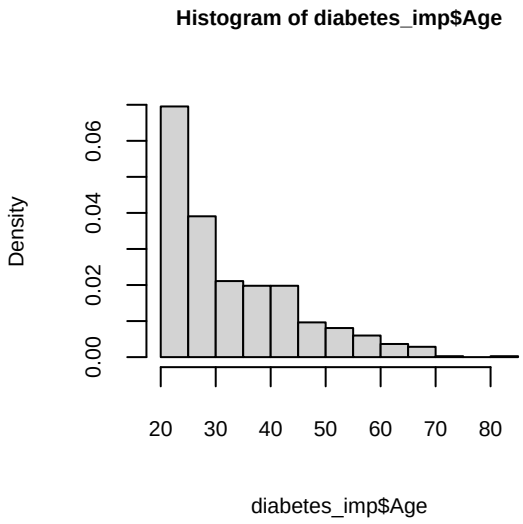

```
hist(diabetes_imp$Insulin, freq = F, nclass = 16,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)
hist(diabetes_imp$BMI, freq = F, nclass = 16,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)
```






```
hist(diabetes_imp$DiabetesPedigreeFunction, freq = F, nclass = 16,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,
hist(diabetes_imp$Age, freq = F, nclass = 16,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)
```





Variable transformation

Looking at the histograms, it is clear that there are some variables with highly right-skewed. In order to ease the posterior analysis, we will take logarithms or square root transformation when it would be convenient.

$$skew(X) \geq \ skew(Log(X)) + 0.20$$

then X will be transformed.

$$skew(X) < skew(Log(X)) + 0.20$$

then X will not be transformed.

```
vect<- list(diabetes_imp$Pregnancies,
            diabetes_imp$Glucose,
            diabetes_imp$BloodPressure,
            diabetes_imp$SkinThickness,
            diabetes_imp$Insulin,
            diabetes_imp$BMI,
            diabetes_imp$DiabetesPedigreeFunction,
            diabetes_imp$Age)
sk <- lapply(vect,FUN=skewness)
```

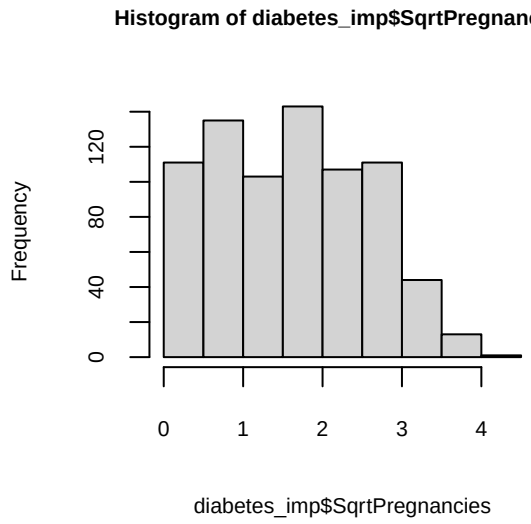
```
vect<- list(sqrt(diabetes_imp$Pregnancies),
            log(diabetes_imp$Glucose),
            log(diabetes_imp$BloodPressure),
            log(diabetes_imp$SkinThickness),
            log(diabetes_imp$Insulin),
            log(diabetes_imp$BMI),
            log(diabetes_imp$DiabetesPedigreeFunction),
            log(diabetes_imp$Age))
sk2 <- lapply(vect,FUN=skewness)

headers <- names(diabetes_imp)
```

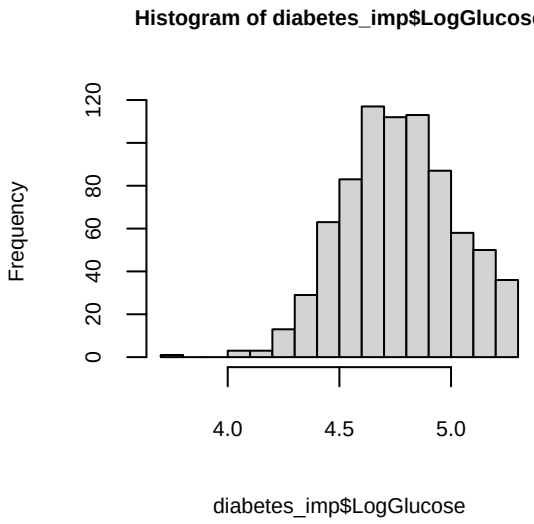
```r
#For every feature, we compare the skewness of the transformed variables vs the variable
for (i in 1:length(sk)){
  if (abs(sk[[i]])<=abs(sk2[[i]])+0.2){
    print(paste(headers[i],": Do not make transformation",sep=""))
  }
  else {
    print(paste(headers[i],": Make transformation",sep=""))
  }
  print(sk[[i]])
  print(sk2[[i]])
}
```

```
## [1] "Pregnancies: Make transformation"
## [1] 0.8981549
## [1] -0.1583176
## [1] "Glucose: Make transformation"
## [1] 0.5283525
## [1] -0.06492608
## [1] "BloodPressure: Do not make transformation"
## [1] 0.1348607
## [1] -0.7804911
## [1] "SkinThickness: Make transformation"
## [1] 0.8913575
## [1] -0.50222
## [1] "Insulin: Make transformation"
## [1] 2.019757
## [1] -0.1268481
## [1] "BMI: Make transformation"
## [1] 0.5883214
## [1] -0.05719422
## [1] "DiabetesPedigreeFunction: Make transformation"
## [1] 1.912418
## [1] 0.1137321
## [1] "Age: Make transformation"
## [1] 1.125188
## [1] 0.5993976
```

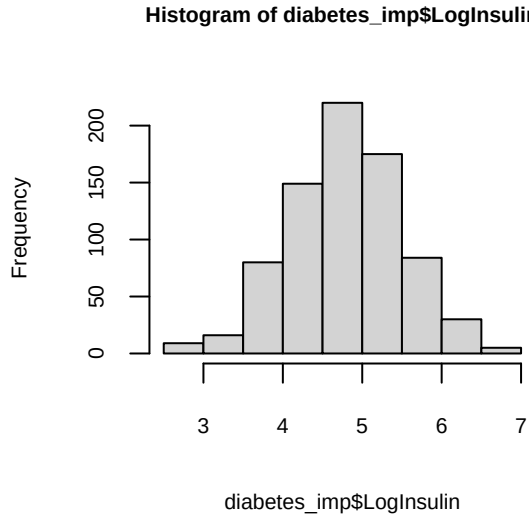
For the resulting variables, the histograms are shown below.

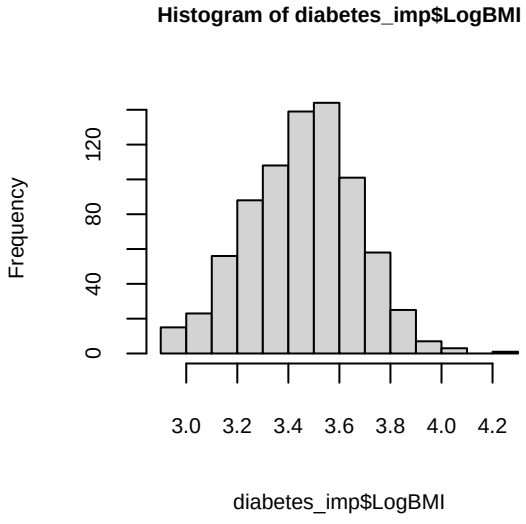
```r
par(mfrow = c(1,2))


diabetes_imp$SqrtPregnancies  <- sqrt(diabetes_imp$Pregnancies)
hist(diabetes_imp$SqrtPregnancies,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)

diabetes_imp$LogGlucose <- log(diabetes_imp$Glucose)
hist(diabetes_imp$LogGlucose,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)
```



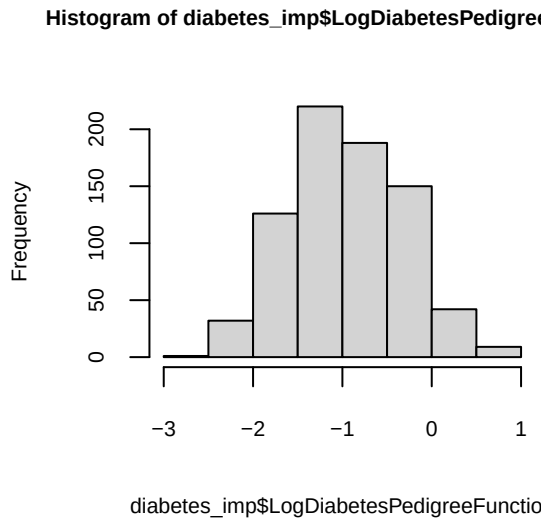


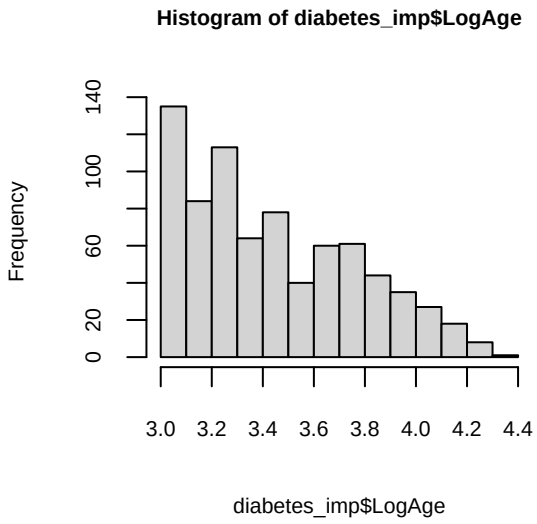

```
diabetes_imp$LogInsulin  <- log(diabetes_imp$Insulin)
hist(diabetes_imp$LogInsulin,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)

diabetes_imp$LogBMI  <- log(diabetes_imp$BMI)
hist(diabetes_imp$LogBMI,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)
```






```
diabetes_imp$LogDiabetesPedigreeFunction  <- log(diabetes_imp$DiabetesPedigreeFunction)
hist(diabetes_imp$LogDiabetesPedigreeFunction,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)

diabetes_imp$LogAge  <- log(diabetes_imp$Age)
hist(diabetes_imp$LogAge,cex.main=0.7,cex.sub=0.7,cex.lab=0.7,cex.axis=0.7)
```





Identifying outliers

Besides the values imputed before, which were non possible zeros for physical values, it is key to identify the observations that may introduce some errors to the model coming from some failures in the measure of the data. With the transformed data, it will be performed the Mahalanobis distance to every point, setting a threshold for outliers.

1- Mean vector, correlation matrix and covariance matrix calculations:

```
diabetes_trans <- diabetes_imp[,c("SqrtPregnancies",
                  "LogGlucose",
                  "BloodPressure",
                  "SkinThickness",
                  "LogInsulin",
                  "LogBMI",
                  "LogDiabetesPedigreeFunction",
                  "LogAge",
                  "Outcome")]

colMeans(diabetes_trans)  #Means of every variable after the imputation
```

```
##             SqrtPregnancies                  LogGlucose
##                   1.6954773                   4.7692240
##               BloodPressure               SkinThickness
##                  72.3593750                  28.6940104
##                  LogInsulin                      LogBMI
##                   4.7913277                   3.4564746
## LogDiabetesPedigreeFunction                     LogAge
##                  -0.9599401                   3.4488023
##                     Outcome
##                   0.3489583
```

```
COV <- cov(diabetes_trans)
COV
```

```
##                              SqrtPregnancies LogGlucose BloodPressure
## SqrtPregnancies                 0.971673964 0.02557407    1.96078634
## LogGlucose                      0.025574066 0.06295729    0.70392379
## BloodPressure                   1.960786338 0.70392379  150.54343220
## SkinThickness                   0.591876012 0.55062386   28.50515401
## LogInsulin                      0.062506014 0.11846960    1.42239153
## LogBMI                         -0.002751399 0.01300480    0.74160591
## LogDiabetesPedigreeFunction    -0.029100613 0.01779161   -0.02370529
## LogAge                          0.175384640 0.02181704    1.34186907
## Outcome                         0.084704914 0.05801986    0.96569426
##                              SkinThickness LogInsulin       LogBMI
## SqrtPregnancies                 0.5918760 0.06250601 -0.002751399
## LogGlucose                      0.5506239 0.11846960  0.013004804
## BloodPressure                  28.5051540 1.42239153  0.741605909
## SkinThickness                 116.9636147 1.30981654  1.540332255
## LogInsulin                      1.3098165 0.51124108  0.041883054
## LogBMI                          1.5403323 0.04188305  0.045087391
## LogDiabetesPedigreeFunction     0.7388500 0.04790810  0.020466093
## LogAge                          0.5669981 0.05345782  0.005554579
## Outcome                         1.2816235 0.10594116  0.032914691
##                              LogDiabetesPedigreeFunction      LogAge    Outcome
## SqrtPregnancies                           -0.029100613 0.175384640 0.08470491
## LogGlucose                                 0.017791610 0.021817037 0.05801986
## BloodPressure                             -0.023705291 1.341869068 0.96569426
## SkinThickness                              0.738849969 0.566998135 1.28162348
## LogInsulin                                 0.047908104 0.053457816 0.10594116
## LogBMI                                     0.020466093 0.005554579 0.03291469
## LogDiabetesPedigreeFunction                0.415150313 0.008997508 0.05550401
## LogAge                                     0.008997508 0.104136642 0.04233793
## Outcome                                    0.055504006 0.042337929 0.22748262
```

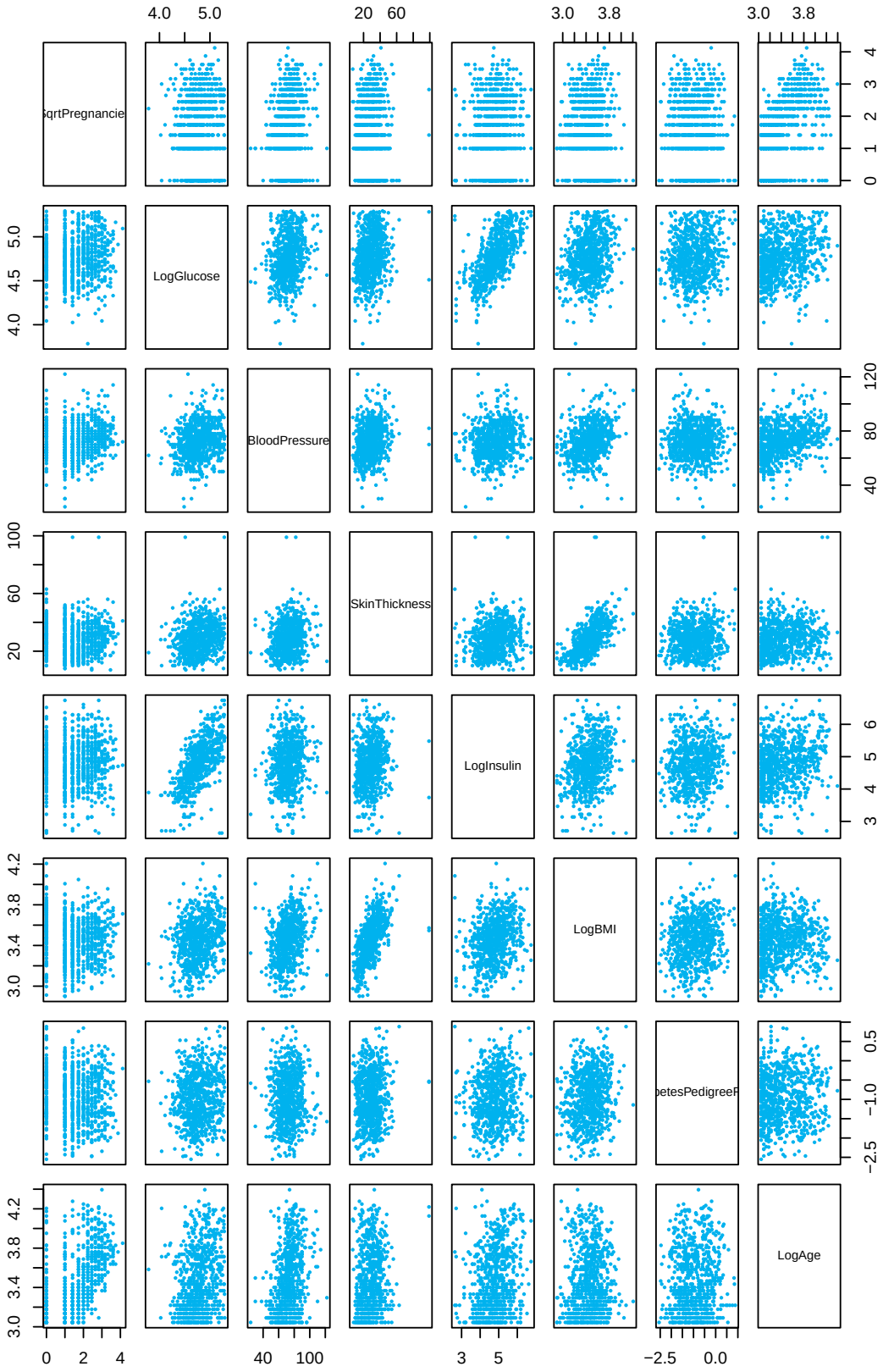
```
COR <- cor(diabetes_trans)       #Correlation matrix
COR
```

```
##                              SqrtPregnancies LogGlucose BloodPressure
## SqrtPregnancies                   1.00000000  0.1033990   0.162120929
## LogGlucose                        0.10339904  1.0000000   0.228650279
## BloodPressure                     0.16212093  0.2286503   1.000000000
## SkinThickness                     0.05551944  0.2029115   0.214816232
## LogInsulin                        0.08868469  0.6603453   0.162134478
## LogBMI                           -0.01314515  0.2440918   0.284652315
## LogDiabetesPedigreeFunction      -0.04581832  0.1100499  -0.002998553
## LogAge                            0.55135268  0.2694455   0.338904589
## Outcome                           0.18016657  0.4848189   0.165019261
##                              SkinThickness LogInsulin      LogBMI
## SqrtPregnancies                 0.05551944 0.08868469 -0.01314515
## LogGlucose                      0.20291149 0.66034526  0.24409185
## BloodPressure                   0.21481623 0.16213448  0.28465231
## SkinThickness                   1.00000000 0.16938393  0.67075121
## LogInsulin                      0.16938393 1.00000000  0.27586583
## LogBMI                          0.67075121 0.27586583  1.00000000
## LogDiabetesPedigreeFunction     0.10602986 0.10399044  0.14959074
## LogAge                          0.16246299 0.23168427  0.08106284
## Outcome                         0.24846256 0.31065458  0.32500356
##                              LogDiabetesPedigreeFunction     LogAge   Outcome
```

```
## SqrtPregnancies                -0.045818321 0.55135268 0.1801666
## LogGlucose                      0.110049878 0.26944553 0.4848189
## BloodPressure                  -0.002998553 0.33890459 0.1650193
## SkinThickness                   0.106029859 0.16246299 0.2484626
## LogInsulin                      0.103990442 0.23168427 0.3106546
## LogBMI                          0.149590737 0.08106284 0.3250036
## LogDiabetesPedigreeFunction     1.000000000 0.04327308 0.1806124
## LogAge                          0.043273075 1.00000000 0.2750766
## Outcome                         0.180612388 0.27507664 1.0000000
```

2- Generate some graphs to see visually the correlations between the variables. It could also be used to look for outliers (typically points quite far from the masses).

```
color_1 <- "deepskyblue2"
color_2 <- "seagreen2"
color_3 <- "orange2"

pairs(diabetes_trans[,-9],pch=16,cex=0.5,col=color_1)
```


LogGlucose
BloodPressure
SkinThickness
LogInsulin
LogBMI
LogAge

For instance, when looking at the correlation plots, it can be seen that the variables SkinThinkness and LodBMI have a strong positive correlation, as well as LogGlucose and LogInsulin. Then, it can be quantified looking at the correlation matrix.

3- Select the 85% of points which are "more centered", and compute the correlation matrix (MCD) with those points. In this situation, we avoid noise imposed by the outliers, which might change the results of the matrix.

```
MCD_est <- CovMcd(diabetes_trans[,-9],alpha=0.85,nsamp="deterministic")
m_MCD <- MCD_est$center
m_MCD
```

```
##             SqrtPregnancies                  LogGlucose
##                   1.7358378                   4.7666967
##               BloodPressure               SkinThickness
##                  71.9052925                  28.3941504
##                  LogInsulin                      LogBMI
##                   4.8046038                   3.4535086
## LogDiabetesPedigreeFunction                     LogAge
##                  -0.9690564                   3.4295596
```

```
S_MCD <- MCD_est$cov
S_MCD
```

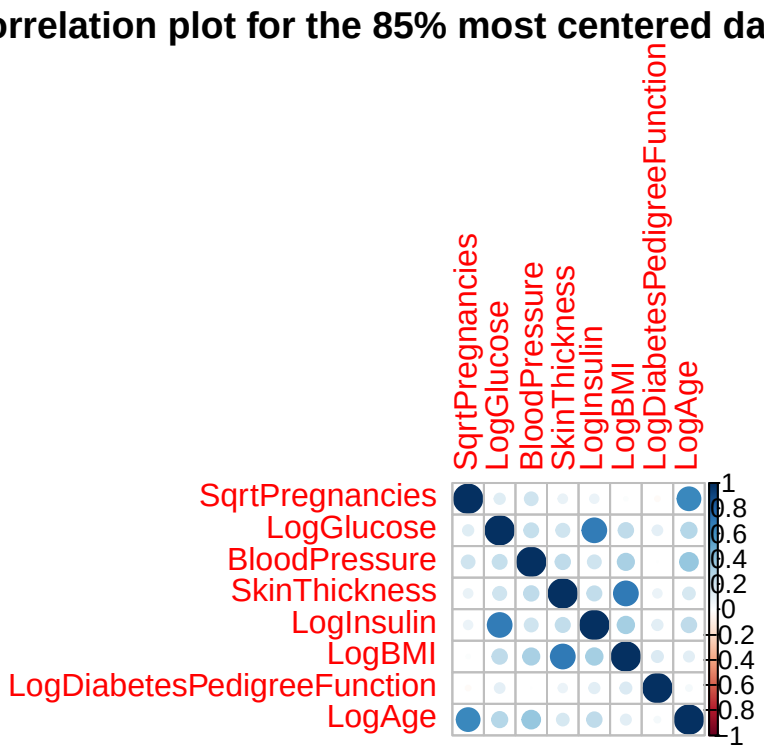
```
##                              SqrtPregnancies LogGlucose BloodPressure
## SqrtPregnancies                 1.010578090 0.03357398  2.384788e+00
## LogGlucose                      0.033573978 0.06468362  7.173852e-01
## BloodPressure                   2.384788131 0.71738522  1.388858e+02
## SkinThickness                   0.971971750 0.55434370  3.141988e+01
## LogInsulin                      0.064487508 0.12795968  1.762823e+00
## LogBMI                          0.003119559 0.01404795  8.124186e-01
## LogDiabetesPedigreeFunction    -0.018243419 0.01970360  2.229831e-03
## LogAge                          0.210221190 0.02362527  1.488591e+00
##                              SkinThickness LogInsulin      LogBMI
## SqrtPregnancies                 0.9719717 0.06448751 0.003119559
## LogGlucose                      0.5543437 0.12795968 0.014047948
## BloodPressure                  31.4198804 1.76282275 0.812418646
## SkinThickness                 107.9213302 1.84224771 1.549783933
## LogInsulin                      1.8422477 0.52182985 0.051145841
## LogBMI                          1.5497839 0.05114584 0.045167460
## LogDiabetesPedigreeFunction     0.6120739 0.06226895 0.021399793
## LogAge                          0.5747003 0.06090412 0.008654399
##                              LogDiabetesPedigreeFunction      LogAge
## SqrtPregnancies                           -0.018243419 0.210221190
## LogGlucose                                 0.019703603 0.023625265
## BloodPressure                              0.002229831 1.488591268
## SkinThickness                              0.612073880 0.574700265
## LogInsulin                                 0.062268951 0.060904123
## LogBMI                                     0.021399793 0.008654399
## LogDiabetesPedigreeFunction                0.442047608 0.008895934
## LogAge                                     0.008895934 0.105543194
```

```
R_MCD <- cov2cor(S_MCD)
R_MCD
```

```
##                              SqrtPregnancies LogGlucose BloodPressure
## SqrtPregnancies                  1.00000000  0.1313169  0.2012963860
```

```
## LogGlucose                        0.13131692  1.0000000  0.2393461104
## BloodPressure                     0.20129639  0.2393461  1.0000000000
## SkinThickness                     0.09307115  0.2098110  0.2566386125
## LogInsulin                        0.08880271  0.6964846  0.2070691034
## LogBMI                            0.01460143  0.2598980  0.3243681207
## LogDiabetesPedigreeFunction      -0.02729523  0.1165236  0.0002845827
## LogAge                            0.64368951  0.2859331  0.3888046634
##                             SkinThickness LogInsulin     LogBMI
## SqrtPregnancies                0.09307115 0.08880271 0.01460143
## LogGlucose                     0.20981103 0.69648462 0.25989797
## BloodPressure                  0.25663861 0.20706910 0.32436812
## SkinThickness                  1.00000000 0.24548780 0.70194757
## LogInsulin                     0.24548780 1.00000000 0.33314479
## LogBMI                         0.70194757 0.33314479 1.00000000
## LogDiabetesPedigreeFunction    0.08861672 0.12965001 0.15144754
## LogAge                         0.17028353 0.25951778 0.12534556
##                            LogDiabetesPedigreeFunction     LogAge
## SqrtPregnancies                          -0.0272952306 0.64368951
## LogGlucose                                0.1165236203 0.28593308
## BloodPressure                             0.0002845827 0.38880466
## SkinThickness                             0.0886167216 0.17028353
## LogInsulin                                0.1296500129 0.25951778
## LogBMI                                    0.1514475410 0.12534556
## LogDiabetesPedigreeFunction               1.0000000000 0.04118527
## LogAge                                    0.0411852692 1.00000000
```

```
corrplot(R_MCD,mar=c(0,0,1,0),title = "Correlation plot for the 85% most centered data")
```



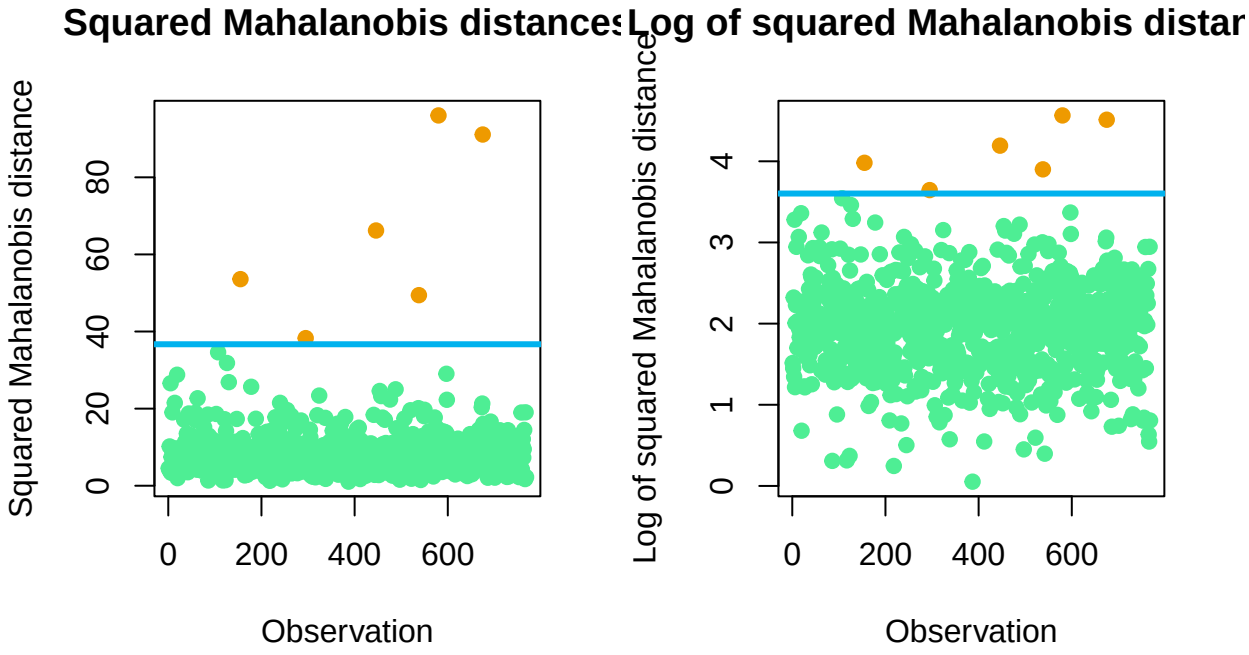
4- The main step is to compute the Mahalanobis distance. After this MCD matrix is computed, the Mahalanobis distance is calculated for every row. The outliers are the points with a distance greater than 95^(1/n) th quantile of the chi-square distribution:

$$X \sim N(\mu, \Sigma), \quad D_M \sim X_p^2$$

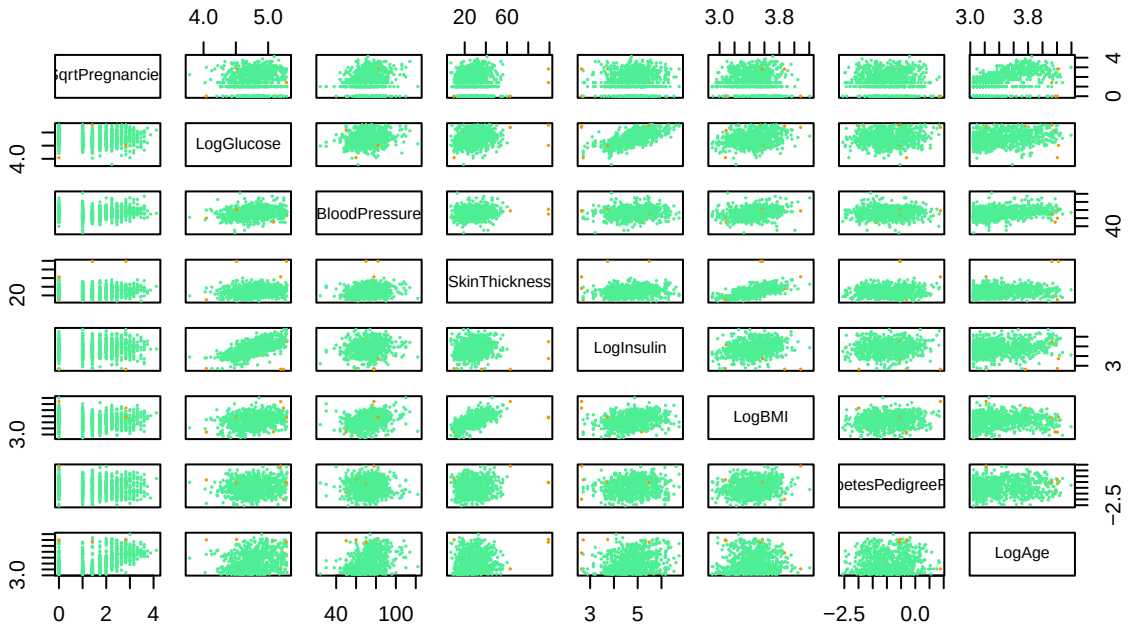
```
diabetes_sq_Mah_MCD <- MCD_est$mah
col_outliers_Mah_MCD <- rep(color_2,n)
outliers_Mah_MCD <- which(diabetes_sq_Mah_MCD>qchisq(.99^(1/n),p))
outliers_Mah_MCD
```

```
## [1] 155 295 446 538 580 675
```

```
col_outliers_Mah_MCD[outliers_Mah_MCD] <- color_3
par(mfrow=c(1,2))
plot(1:n,diabetes_sq_Mah_MCD,pch=19,col=col_outliers_Mah_MCD,
     main="Squared Mahalanobis distances",xlab="Observation",ylab="Squared Mahalanobis distance")
abline(h=qchisq(.99^(1/n),p),lwd=3,col=color_1)
plot(1:n,log(diabetes_sq_Mah_MCD),pch=19,col=col_outliers_Mah_MCD,
     main="Log of squared Mahalanobis distances",
     xlab="Observation",ylab="Log of squared Mahalanobis distance")
abline(h=log(qchisq(.99^(1/n),p)),lwd=3,col=color_1)
```




```
pairs(diabetes_trans[,-9],pch=16,col=col_outliers_Mah_MCD,cex=0.35)
```



After identifying the outliers, those are removed from the data matrix. Finally the correlation and covariance matrix can be calculated correctly.

```
diabetes_trans <- diabetes_trans[-outliers_Mah_MCD,]
n <- nrow(diabetes_trans)
n
```

```
## [1] 762
```

A final computation of the covariance matrix is shown below differentiating each group by outcome of the diabetes. This is done in order to find out if there is any hidden correlation for each group when looking at the matrix for the whole data.

```
par(mfrow = c(1,2))


df_quant <- diabetes_trans[,-9]
df0 <- diabetes_trans[diabetes_trans$Outcome == 0,]
df0_quant <- df0[,-9]
df1 <- diabetes_trans[diabetes_trans$Outcome == 1,]
df1_quant <- df1[,-9]

print("Global data")
```

```
## [1] "Global data"
```

```
print(colMeans(diabetes_trans))
```

```
##             SqrtPregnancies                 LogGlucose
##                    1.699548                    4.768263
##               BloodPressure               SkinThickness
##                   72.380577                   28.492126
##                  LogInsulin                      LogBMI
```

```
##                     4.799699                     3.455822
## LogDiabetesPedigreeFunction                       LogAge
##                    -0.962422                     3.444848
##                      Outcome
##                     0.347769
```

```
print("Outcome = 0")
```

```
## [1] "Outcome = 0"
```

```
print(colMeans(df0))
```

```
##             SqrtPregnancies                  LogGlucose
##                    1.569299                    4.681037
##               BloodPressure               SkinThickness
##                   70.919517                   26.631791
##                  LogInsulin                      LogBMI
##                    4.633393                    3.406953
## LogDiabetesPedigreeFunction                       LogAge
##                   -1.046943                    3.378932
##                     Outcome
##                    0.000000
```

```
print("Outcome = 1")
```

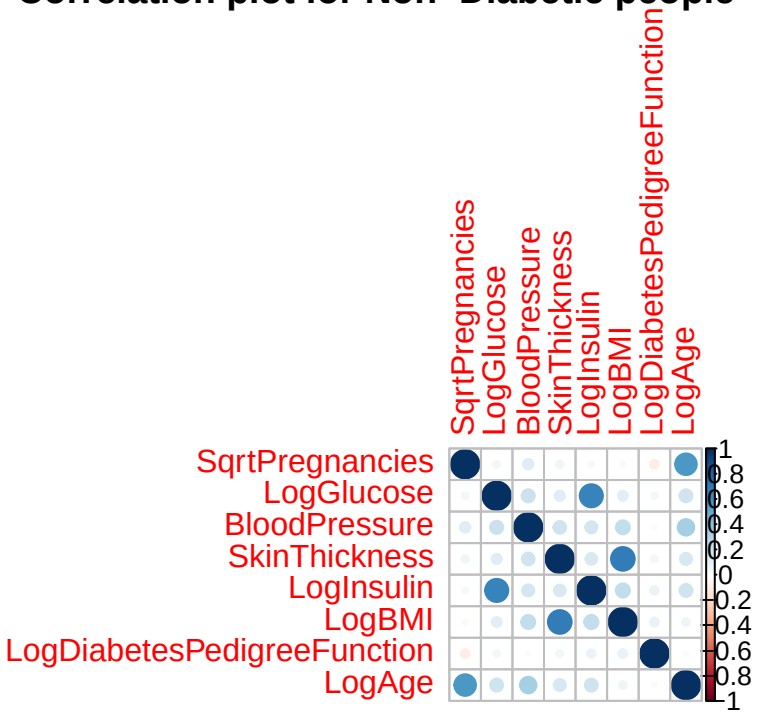
```
## [1] "Outcome = 1"
```

```
print(colMeans(df1))
```

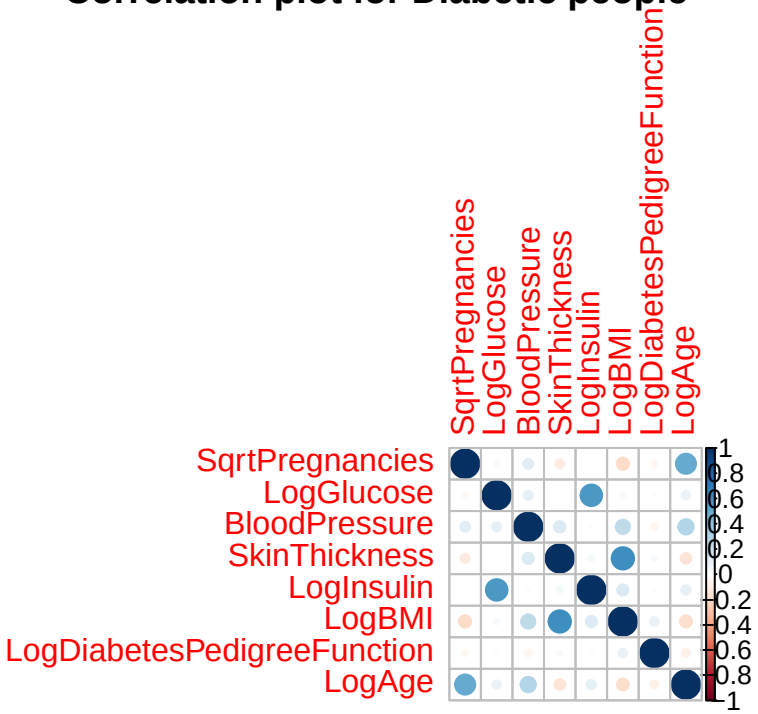
```
##             SqrtPregnancies                  LogGlucose
##                   1.9438262                   4.9318516
##               BloodPressure               SkinThickness
##                  75.1207547                  31.9811321
##                  LogInsulin                      LogBMI
##                   5.1115998                   3.5474740
## LogDiabetesPedigreeFunction                       LogAge
##                  -0.8039056                   3.5684717
##                     Outcome
##                   1.0000000
```

```
S_0 <- cor(df0[,-9])
S_1 <- cor(df1[,-9])
```

```
corrplot(S_0,title="Correlation plot for Non-Diabetic people",mar=c(0,0,1,0))
```




```
corrplot(S_1,title="Correlation plot for Diabetic people",mar=c(0,0,1,0))
```

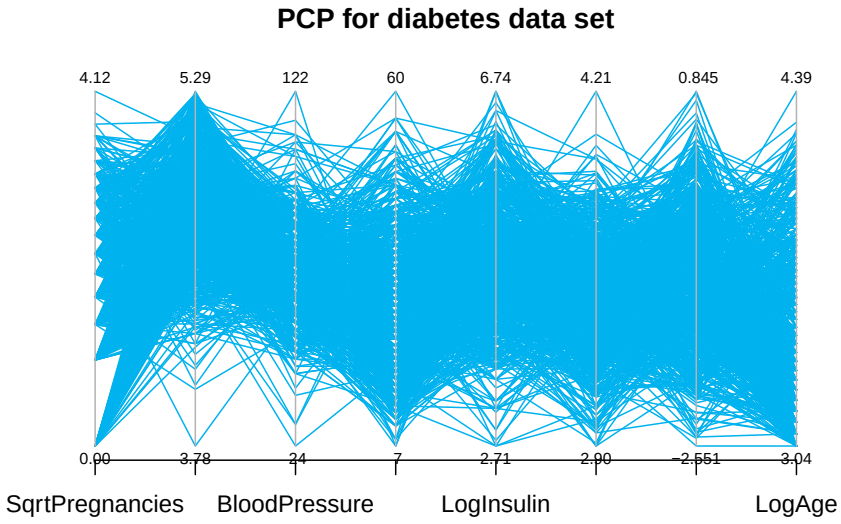


The correlation plots show that the correlation between variables is more or less the same for every pair of variables. For example, one thing that it can be pointed out, is that the correlations are slightly stronger for the non-diabetic people (outcome = 0) than for the diabetic people (outcome = 1)

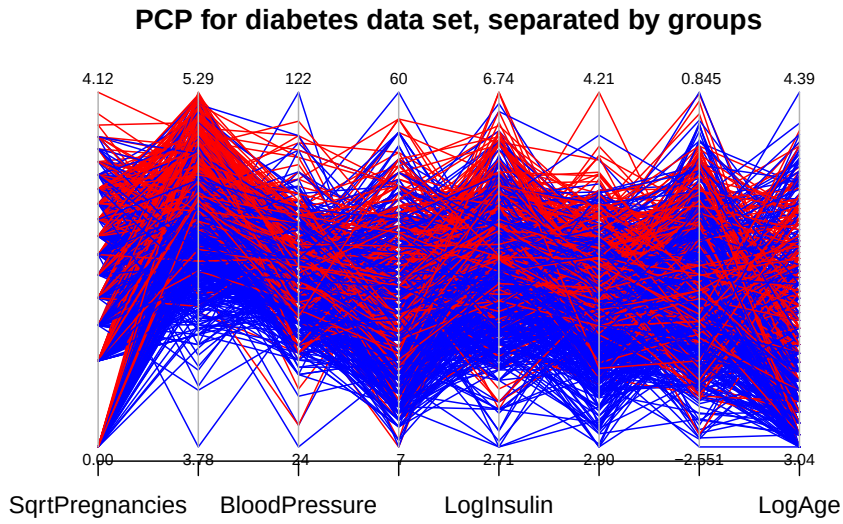
Data visualization

First, we will use some visual tools to identify trends for the outcome. The parallel coordinates are shown below.

```
par(mfrow=c(1,1))

parcoord(df_quant,col=color_1,var.label=TRUE,main="PCP for diabetes data set", cex=0.5)
```

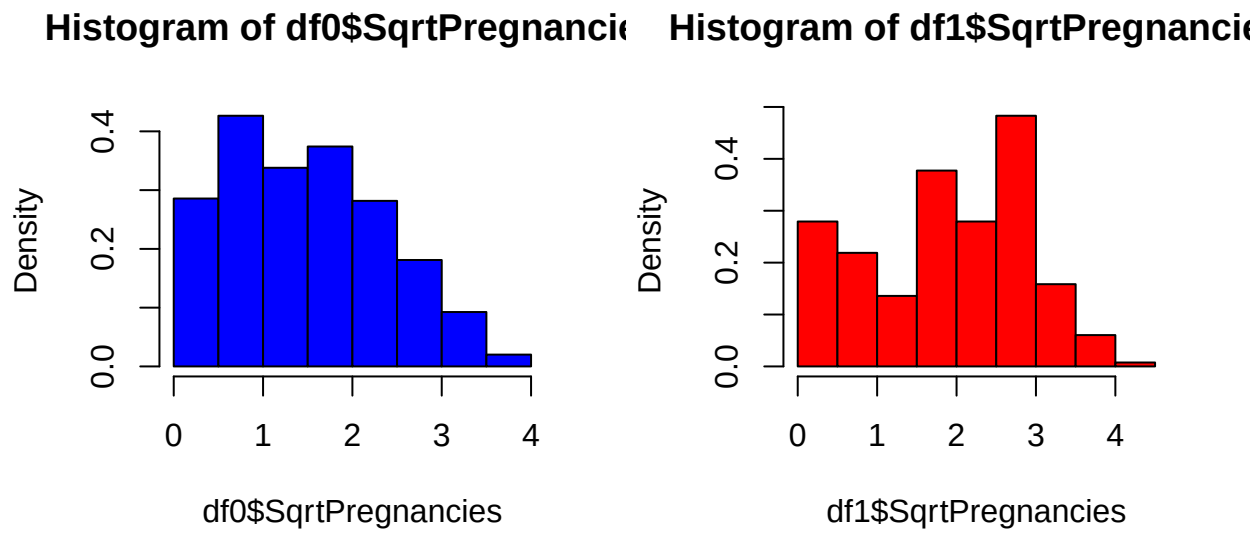



```
diabetes_colors <- c("red","blue")[1*(diabetes_trans$Outcome==0)+1]


parcoord(df_quant,col=diabetes_colors,var.label=TRUE,
         main="PCP for diabetes data set, separated by groups")
```

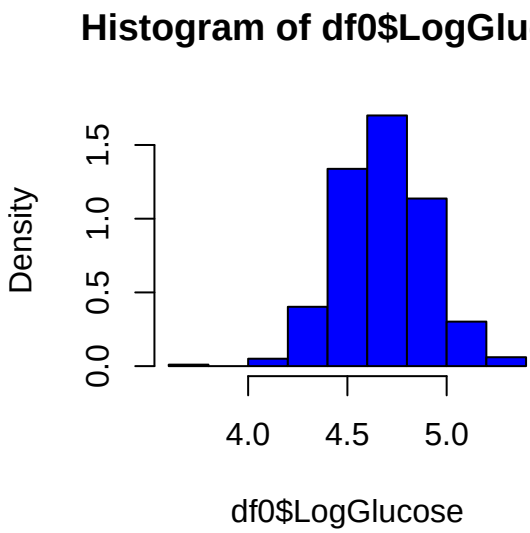


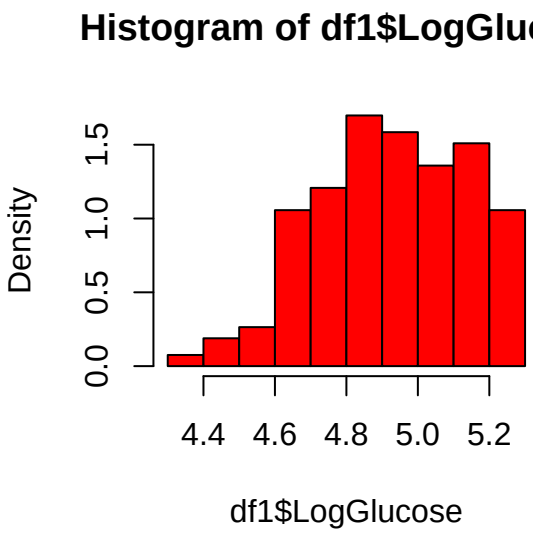
Now, we show the parallel coordinates plot separating by groups, in red non-diabetic and in blue diabetic.

```
par(mfrow = c(1,2))

hist(df0$SqrtPregnancies, freq = F, col="blue")
hist(df1$SqrtPregnancies, freq = F, col="red")
```

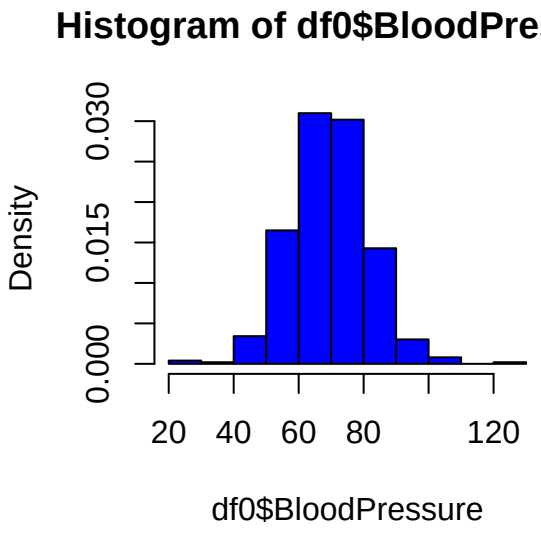


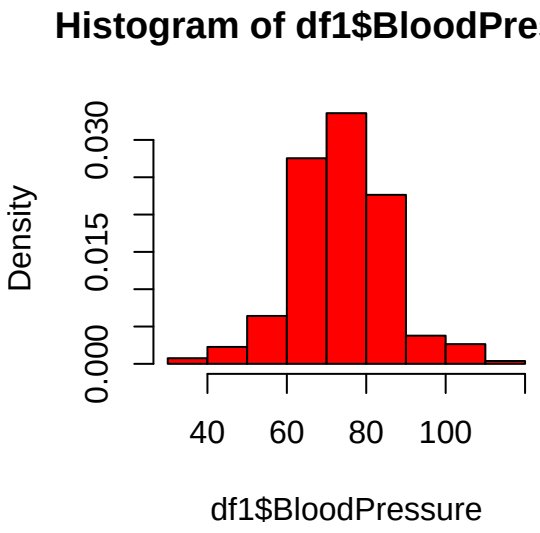

```
hist(df0$LogGlucose, freq = F, col="blue")
hist(df1$LogGlucose, freq = F, col="red")
```



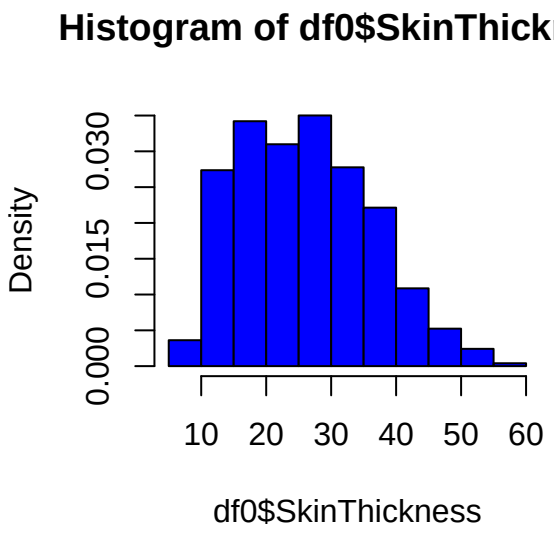


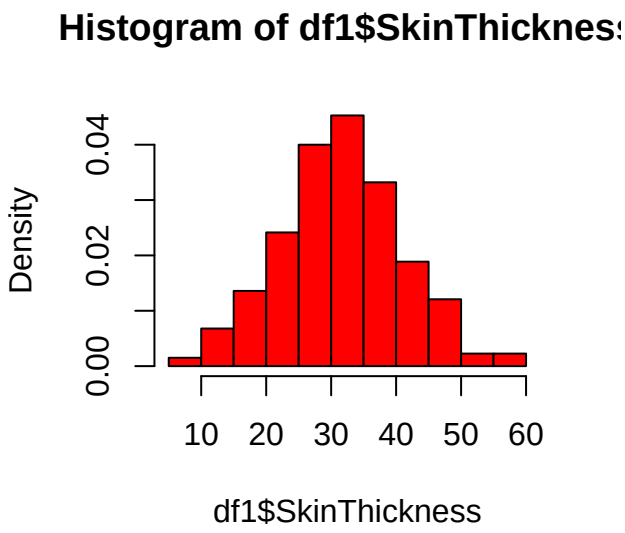

```
hist(df0$BloodPressure, freq = F, col="blue")
hist(df1$BloodPressure, freq = F, col="red")
```



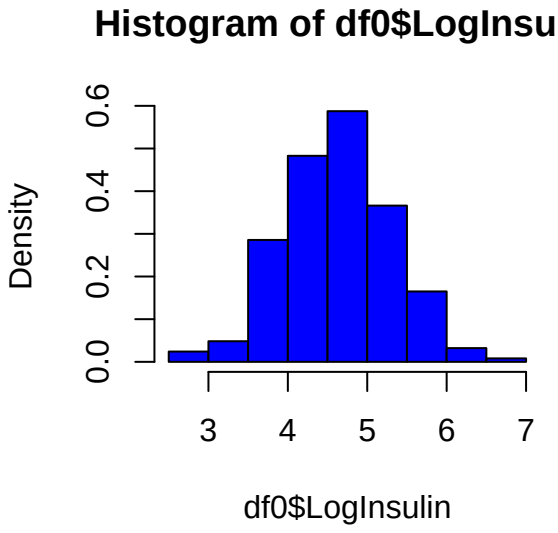


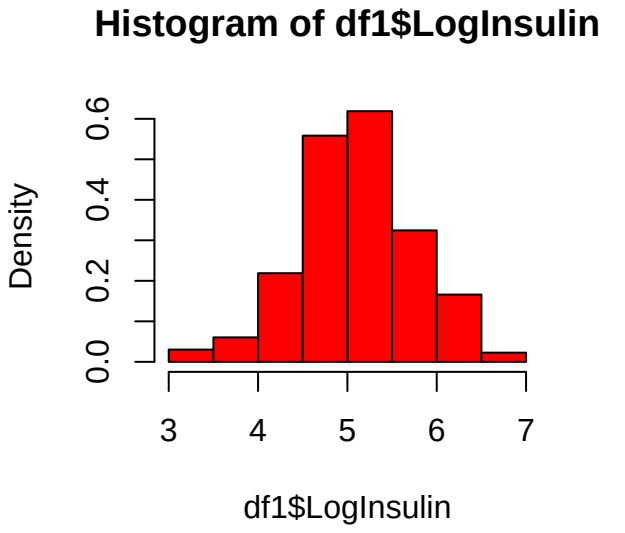

```
hist(df0$SkinThickness, freq = F, col="blue")
hist(df1$SkinThickness, freq = F, col="red")
```



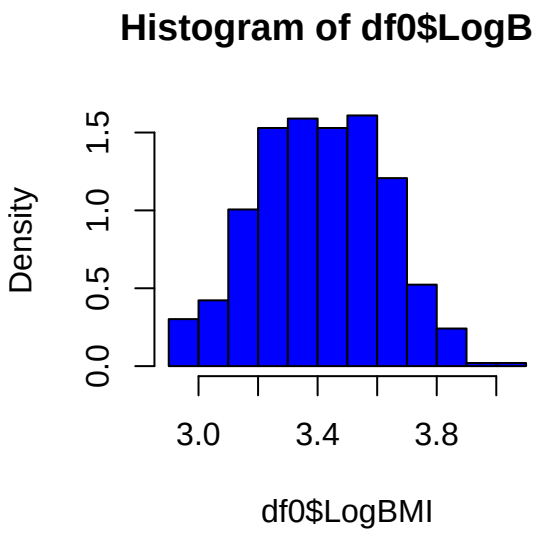


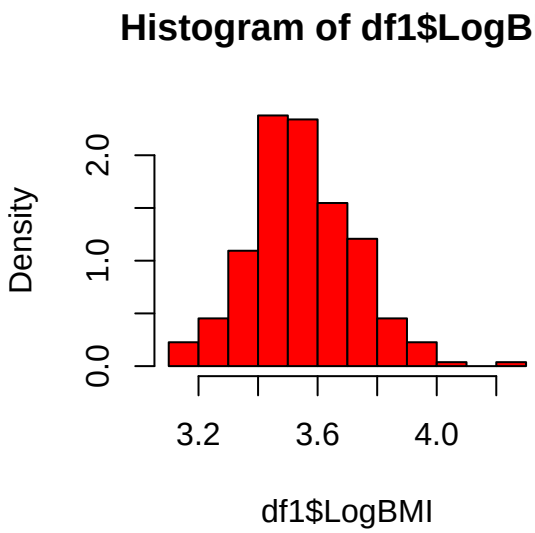

```
hist(df0$LogInsulin, freq = F, col="blue")
hist(df1$LogInsulin, freq = F, col="red")
```



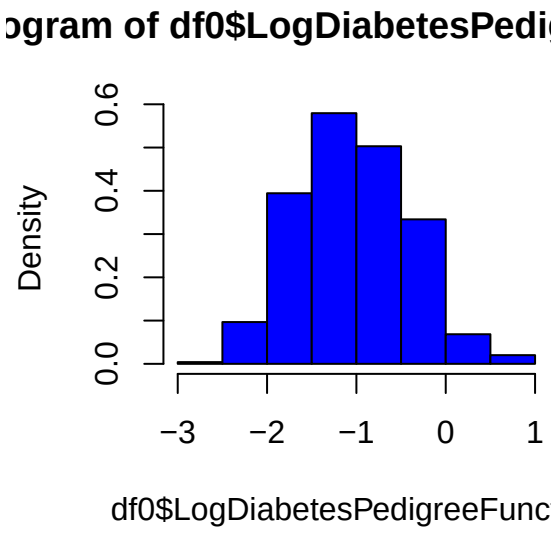


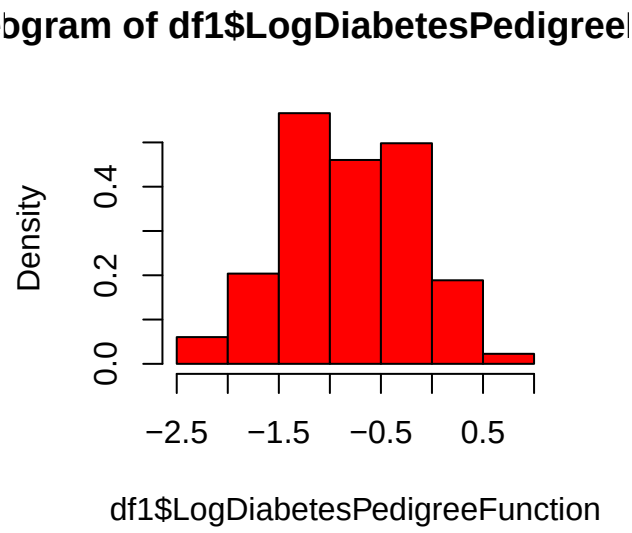

```
hist(df0$LogBMI, freq = F, col="blue")
hist(df1$LogBMI, freq = F, col="red")
```



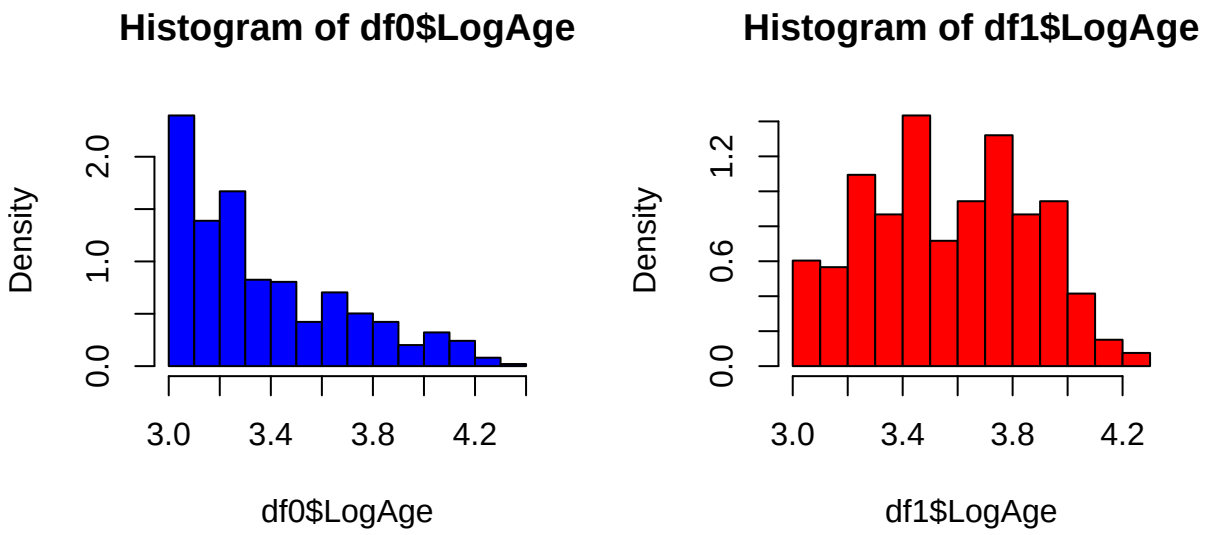



```
hist(df0$LogDiabetesPedigreeFunction, freq = F, col="blue")
hist(df1$LogDiabetesPedigreeFunction, freq = F, col="red")
```






```
hist(df0$LogAge, freq = F, col="blue")
hist(df1$LogAge, freq = F, col="red")
```



Taking a look at the graphs, in red, we have represented the histograms for the variables for the non-diabetic people, while at the left in blue, diabetic people are shown. The differences that are noticed are: - Higher concentration of glucose for diabetics. This is something that can be previously supposed, as the diabetes is an illness that affects the biological generation of insulin which lowers the glucose in blood. - Insulin is higher for diabetic people. This point is could be the consequence of two factors: almost half of the insulin variable is imputed and maybe some of those diabetic people are already diagnosed, implying that many of them could be taking synthetic insulin. - Age distribution is different for diabetic and non-diabetic people. This could be affected for instance, by the age of which every people discover that he/she has diabetes.

Principal component analysis

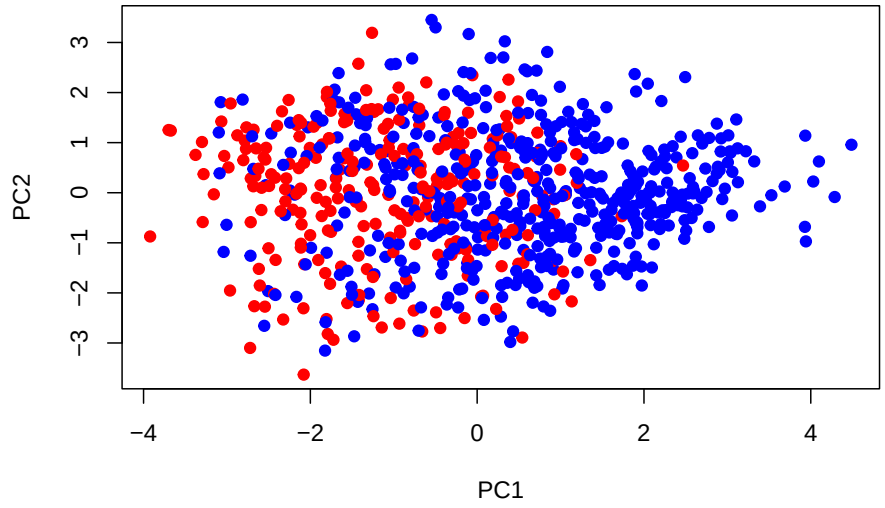
In the steps below, it will be calculated the principal component analysis for the whole data set, and then visualize the patterns in every principal component, for each group of diabetes.

```
df_quant_pcs = prcomp(df_quant,scale=TRUE,center=TRUE)

df_quant_pcs
```

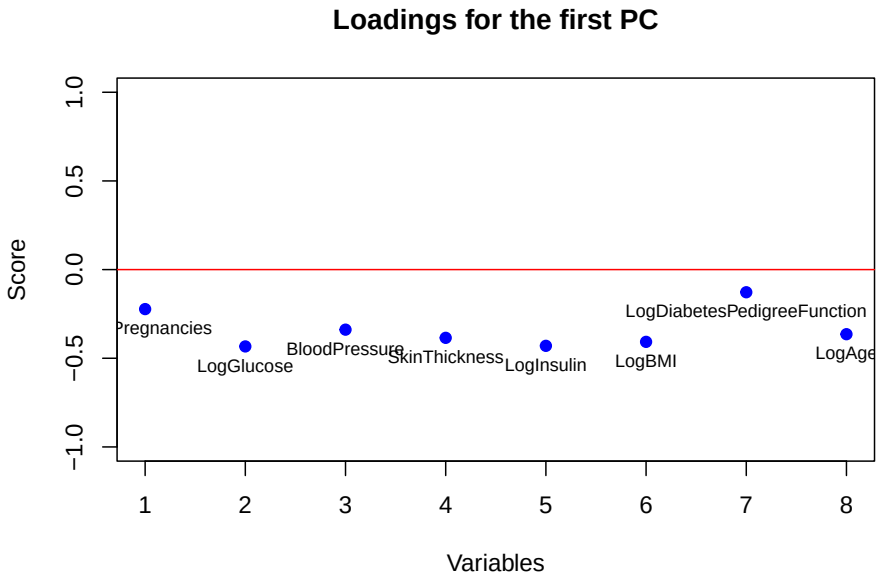
```
## Standard deviations (1, .., p=8):
## [1] 1.6044128 1.2332400 1.0939408 0.9772561 0.8793150 0.6175731 0.5795999
## [8] 0.5125574
##
## Rotation (n x k) = (8 x 8):
##                                  PC1         PC2        PC3           PC4
## SqrtPregnancies             -0.2227051  0.58630129 -0.2286988 -0.1961868743
## LogGlucose                  -0.4332707  0.01520441  0.5202513  0.1763147631
## BloodPressure               -0.3386402  0.12444176 -0.2701115  0.1887925160
## SkinThickness               -0.3848635 -0.37821635 -0.4115781 -0.0009951065
## LogInsulin                  -0.4299005 -0.03843576  0.5249486  0.1679127774
## LogBMI                      -0.4074325 -0.44012489 -0.3102191  0.0126991686
## LogDiabetesPedigreeFunction -0.1277664 -0.19604071  0.2088244 -0.9178868291
## LogAge                      -0.3643377  0.51367994 -0.1387954 -0.1546221510
##                                    PC5         PC6         PC7          PC8
## SqrtPregnancies             -0.369784741  0.61348812 -0.04982182  0.01643736
## LogGlucose                  -0.007911996  0.03219592 -0.62511883 -0.34412824
```

```
## BloodPressure                 0.833221528  0.21523364 -0.01209942  0.14319299
## SkinThickness                -0.290920032 -0.15276929 -0.38315744  0.53352826
## LogInsulin                   -0.124490955  0.08869096  0.55428544  0.42349273
## LogBMI                       -0.126186532  0.13111412  0.36401824 -0.61503535
## LogDiabetesPedigreeFunction   0.228717665  0.05951215 -0.02843303  0.04957854
## LogAge                       -0.023900993 -0.72410847  0.13870166 -0.12679247
```

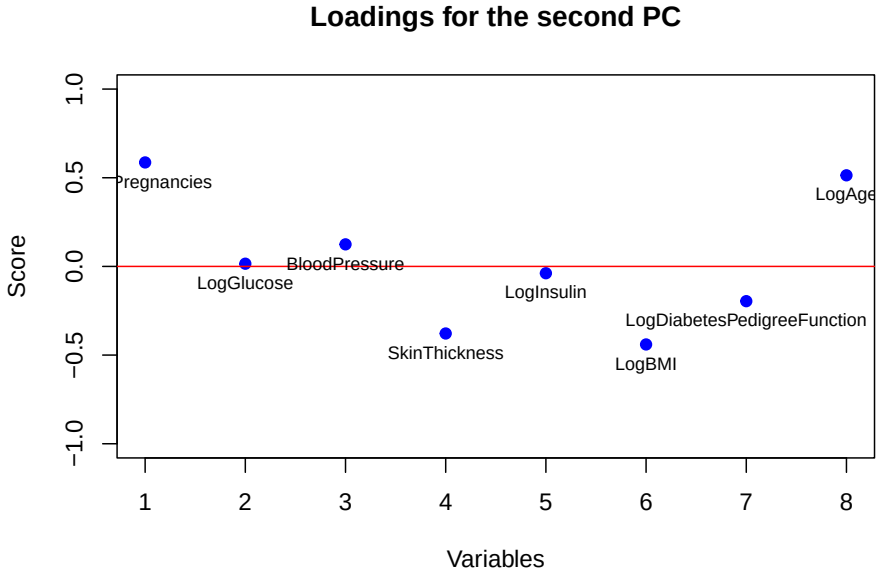
```
par(mfrow=c(1,1))
plot(df_quant_pcs$x[,1:2],pch=19,col=diabetes_colors)
```




```
#See loadings for first PC
plot(1:8,df_quant_pcs$rotation[,1],pch=19,col="blue",main="Loadings for the first PC",
     xlab="Variables",ylab="Score",ylim=c(-1,1))
abline(h=0,col="red")
text(1:8,df_quant_pcs$rotation[,1],labels=colnames(df_quant),pos=1,col="black",cex=0.75)
```

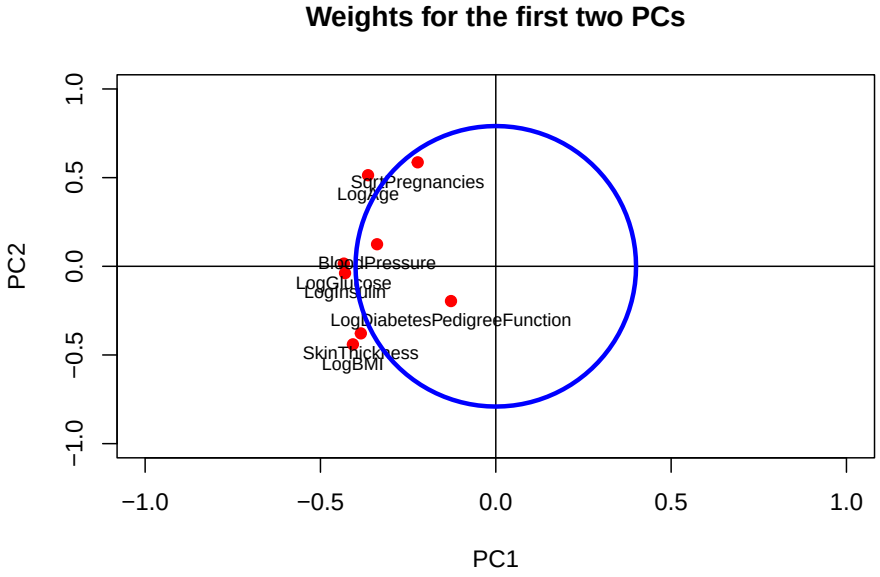



```
#See loadings for second PC
plot(1:8,df_quant_pcs$rotation[,2],pch=19,col="blue",main="Loadings for the second PC",
     xlab="Variables",ylab="Score",ylim=c(-1,1))
abline(h=0,col="red")
text(1:8,df_quant_pcs$rotation[,2],labels=colnames(df_quant),pos=1,col="black",cex=0.75)
```

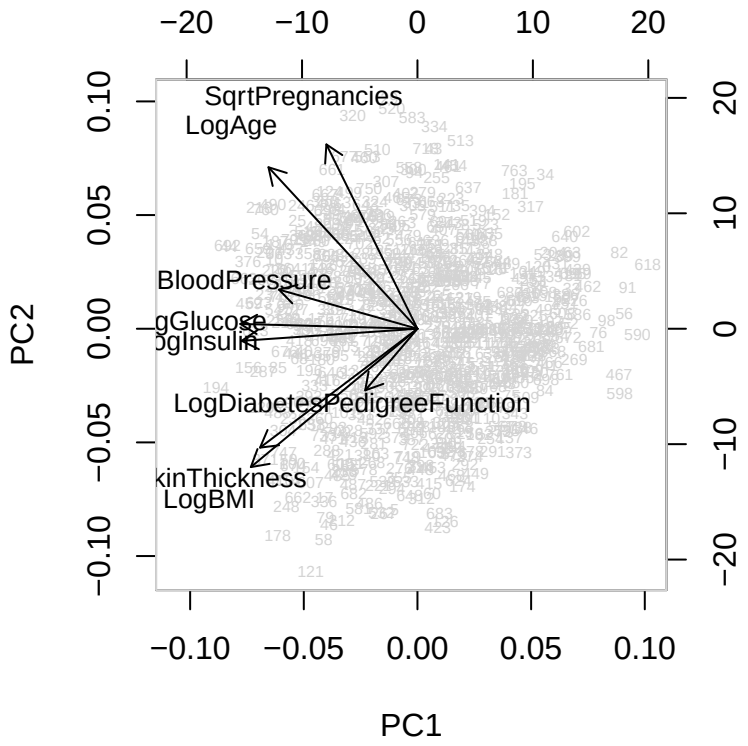


When watching the graphs above, it is clear that the two groups can be differentiated in the first principal component and are mixed in the second one. Also, looking at the loading, the first PC is not clear about the meaning, though it could be used as some metric to identify diabetes for the most negative values. On the other hand, the second PC is a metric of the maturity of a person, age and pregnancies are the most considered variables in this case.

```
plot(df_quant_pcs$rotation[,1:2],pch=19,col="red",main="Weights for the first two PCs",
     xlim=c(-1,1),ylim=c(-1,1))
abline(h=0,v=0)
text(df_quant_pcs$rotation[,1:2],labels=colnames(df_quant),pos=1,col="black",cex=0.75)
library(plotrix)
draw.circle(0,0,0.4,border="blue",lwd=3)
```




```
biplot(df_quant_pcs,col=c("lightgrey","black"),cex=c(0.5,0.8))
```



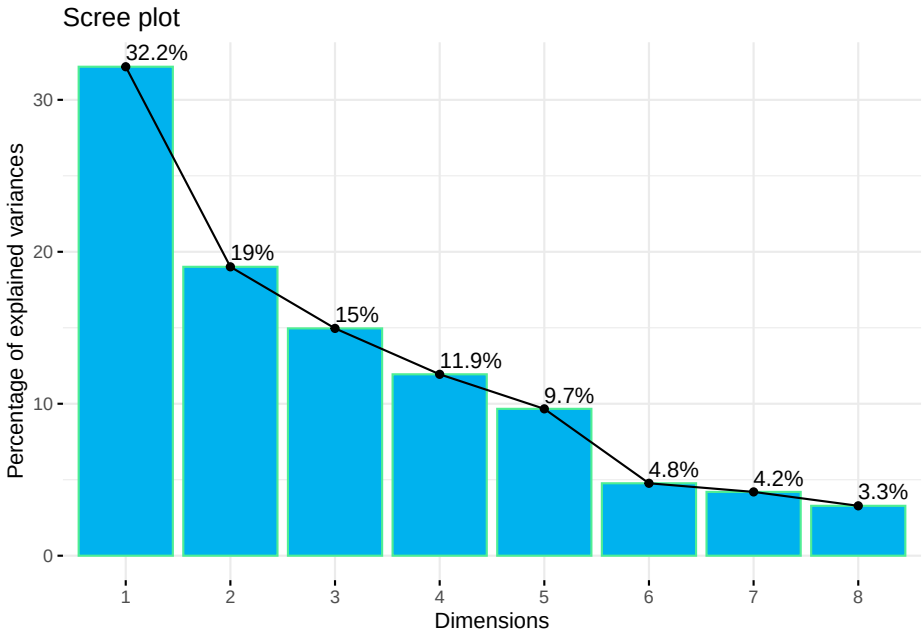
Then, performing the analysis the percentage of the variance explained is seen in the graph. One thing that it could be pointed out, is that as the number of features here is not extremely large, and there are not highly correlated variables, the variance explained by the first 2 o 3 principal components, are not enough to obtain some conclusive results of the whole data set.

For instance, taking the first 3 PCs, it explains 65.6 % of the variance, using 37.5 % of the variables. This not as useful as in other cases with high correlation and high number of variables.

```
df_quant_pcs$sdev^2
```

```
## [1] 2.5741405 1.5208810 1.1967065 0.9550294 0.7731949 0.3813965 0.3359360
## [8] 0.2627151
```

```
fviz_eig(df_quant_pcs,ncp=17,addlabels=T,barfill=color_1,barcolor=color_2)
```

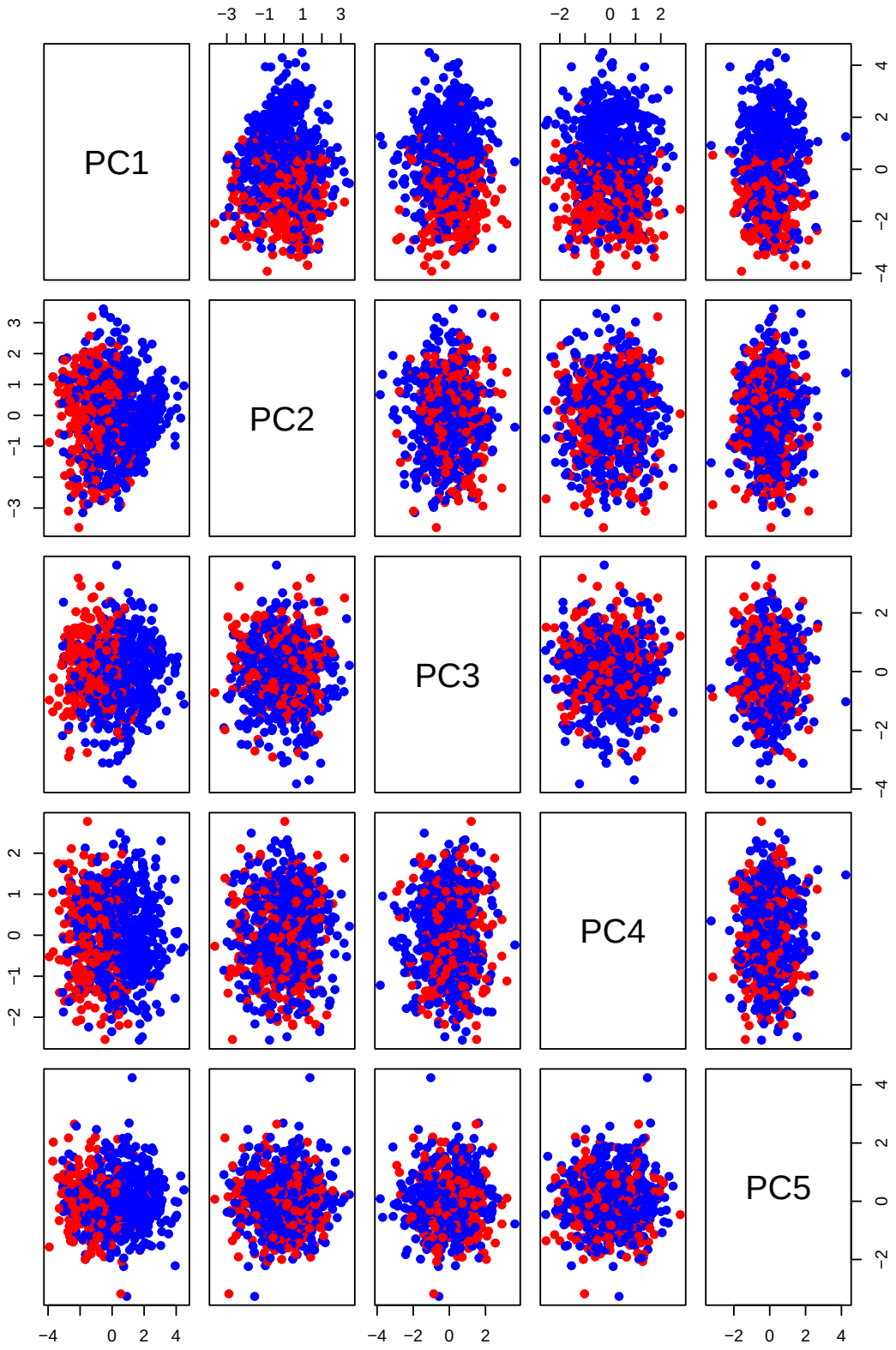



```
get_eigenvalue(df_quant_pcs)
```

```
##        eigenvalue variance.percent cumulative.variance.percent
## Dim.1  2.5741405        32.176756                    32.17676
## Dim.2  1.5208810        19.011012                    51.18777
## Dim.3  1.1967065        14.958832                    66.14660
## Dim.4  0.9550294        11.937868                    78.08447
## Dim.5  0.7731949         9.664936                    87.74940
## Dim.6  0.3813965         4.767456                    92.51686
## Dim.7  0.3359360         4.199200                    96.71606
## Dim.8  0.2627151         3.283939                   100.00000
```

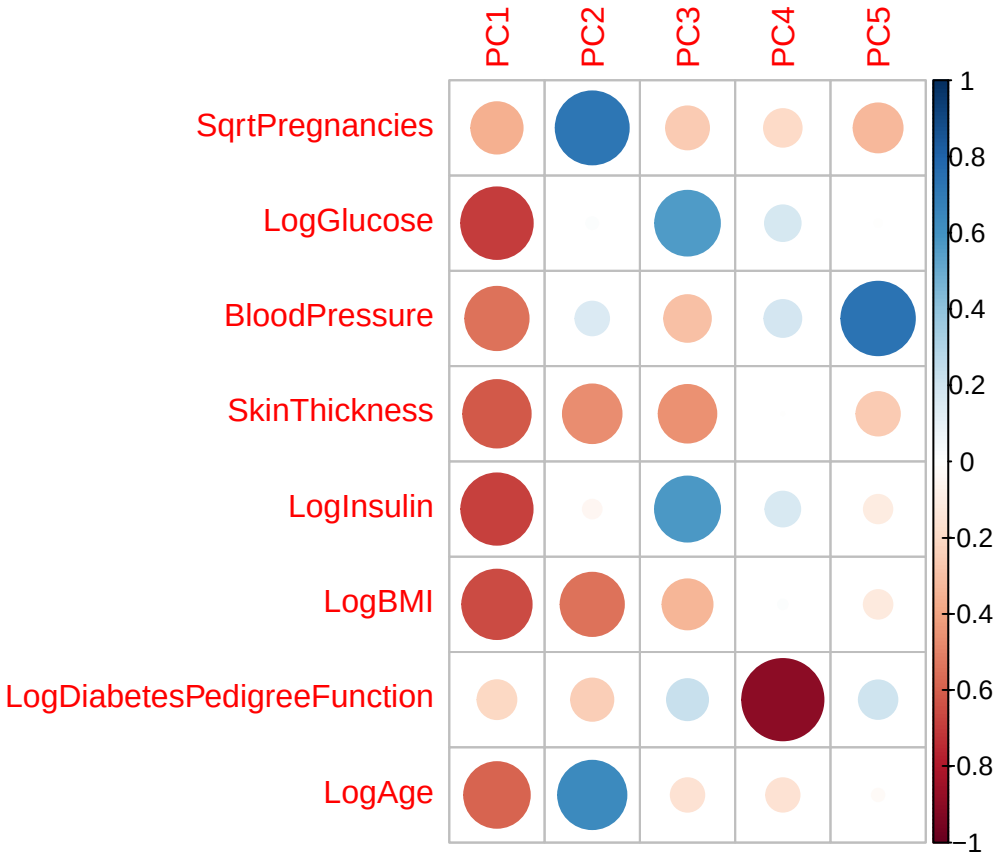
Using the elbow method, it can be selected the optimal number of PCs to be used in the analysis. In this case, the number of PCs could be 3 or 5. We will use 5 PCs in the following steps.

```
pairs(df_quant_pcs$x[,1:5],col=diabetes_colors,pch=19,main="The first five PCs")
```


The first five PCs
PC1
PC2
PC3
PC4
PC5

By generating the correlation plots, it is clear that the first principal component is a measure of the diabetes. It tends to assign lower values for patients with diabetes than non-diabetic patients. Other thing that can be pointed out in the first PC, is that the blue points (non-diabetic) seems to be equally distributed along the range with sort of a normal distribution. On the other hand, the red points (diabetic) are tending to right (between values of -4 and 0).

```
corrplot(cor(df_quant,df_quant_pcs$x[,1:5]),is.corr=T)
```



In addition to the grapsh of loadings, a correlation plot can be used to have a notion of the loadings in every PC. It is very relevant that the first principal component is highly negative correlated with all the variables that are related to the diabetes.

Independent Component Analysis

In the independent component analysis, we are obtaining variables which are independent from each other and follow a non-Gaussian distribution (skewness and kurtosis coefficient to be maximized).

```
library("ica")
diabetes_quant_ica <- icafast(df_quant,nc=8,alg="par")

Z <- diabetes_quant_ica$S
dim(Z)
```

```
## [1] 762   8
```

```
head(Z)
```

```
##          [,1]        [,2]       [,3]        [,4]        [,5]        [,6]
```

```
## 1 -0.4395504  0.52933526 -0.6182178  0.7533818  0.39413203  1.2293484
## 2  0.1522176  0.17619134  0.3212570 -1.0945276  0.04490304 -0.6182977
## 3  1.7192900 -0.23483842 -2.2502571  1.0151233  0.27288693  1.0385133
## 4  0.4574130  0.01013335  0.5567757 -1.1529629 -0.41233647 -1.1005780
## 5 -0.5779083  3.54836604 -0.9235966  1.7013889 -0.23768150 -0.8348892
## 6  1.0785817 -0.60814285  0.1299985 -0.3772725 -0.66651976  0.3488779
##         [,7]        [,8]
## 1 -1.0814987  0.19670396
## 2 -1.5137102  0.02364578
## 3  0.5543044 -0.02252208
## 4  0.6039703 -0.75114438
## 5 -1.3490753  2.23900100
## 6  0.7891433 -0.70211642
```

```
colnames(Z) <- sprintf("IC-%d",seq(1,8))

cov(Z)
```

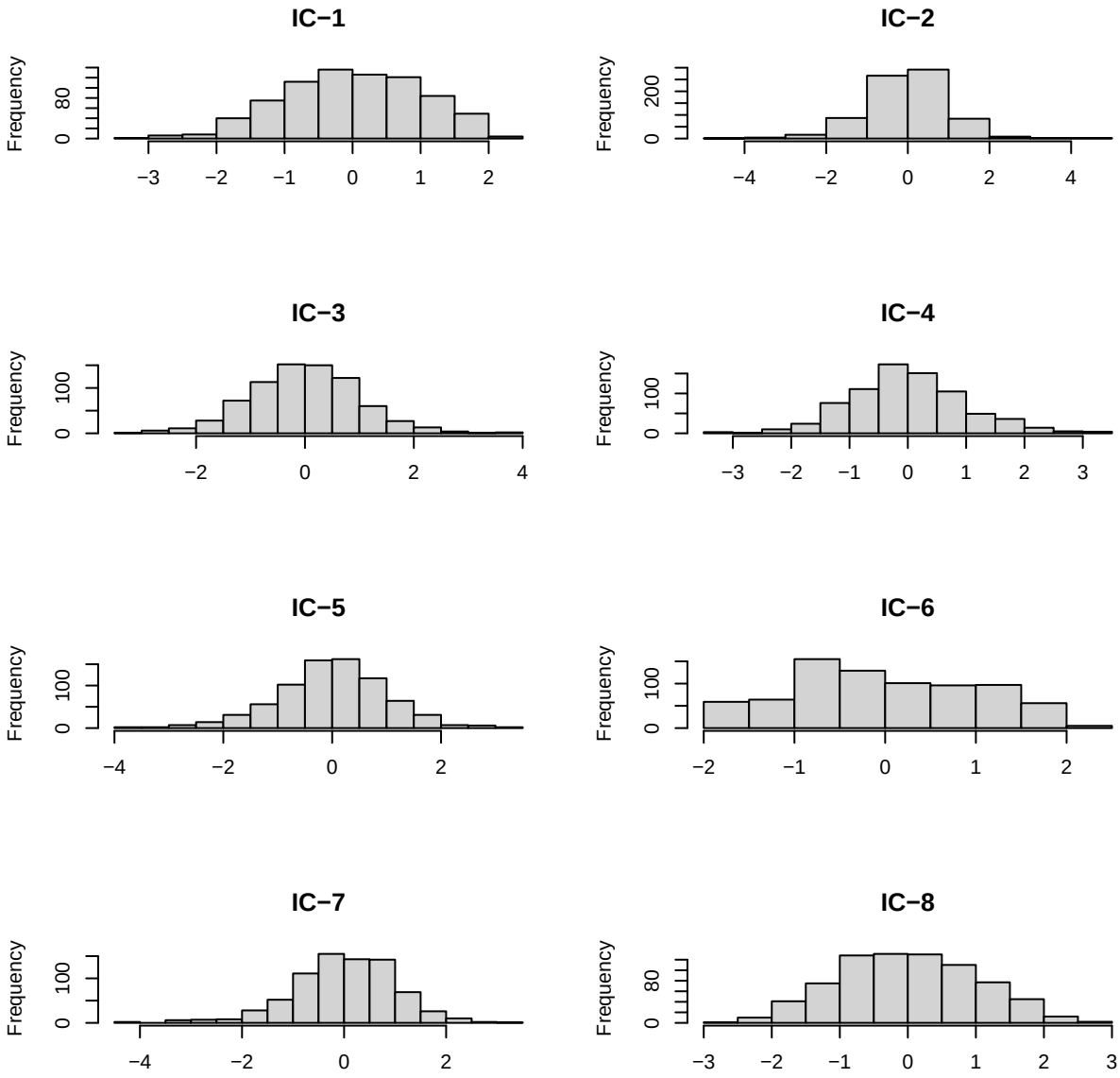
```
##               IC-1          IC-2          IC-3          IC-4          IC-5
## IC-1  1.001314e+00  2.279648e-15 -3.670029e-14  5.757740e-14  2.719117e-14
## IC-2  2.279648e-15  1.001314e+00  2.302811e-14 -2.844309e-14 -2.186969e-14
## IC-3 -3.670029e-14  2.302811e-14  1.001314e+00 -3.668227e-13 -2.234504e-13
## IC-4  5.757740e-14 -2.844309e-14 -3.668227e-13  1.001314e+00  3.082861e-13
## IC-5  2.719117e-14 -2.186969e-14 -2.234504e-13  3.082861e-13  1.001314e+00
## IC-6  4.355149e-15  3.096906e-15  6.755946e-15  7.346051e-15 -1.488762e-14
## IC-7  4.612810e-15 -1.469021e-14 -1.091679e-13  1.211117e-13  1.116456e-13
## IC-8 -3.412894e-14  1.740726e-14  2.124938e-13 -3.212101e-13 -1.827310e-13
##               IC-6          IC-7          IC-8
## IC-1  4.355149e-15  4.612810e-15 -3.412894e-14
## IC-2  3.096906e-15 -1.469021e-14  1.740726e-14
## IC-3  6.755946e-15 -1.091679e-13  2.124938e-13
## IC-4  7.346051e-15  1.211117e-13 -3.212101e-13
## IC-5 -1.488762e-14  1.116456e-13 -1.827310e-13
## IC-6  1.001314e+00 -2.510425e-14 -1.924410e-15
## IC-7 -2.510425e-14  1.001314e+00 -7.526618e-14
## IC-8 -1.924410e-15 -7.526618e-14  1.001314e+00
```

```
Z_norm <- Z * sqrt((n-1)/n)
cov(Z_norm)
```

```
##               IC-1          IC-2          IC-3          IC-4          IC-5
## IC-1  1.000000e+00  2.278365e-15 -3.666067e-14  5.749971e-14  2.715436e-14
## IC-2  2.278365e-15  1.000000e+00  2.299795e-14 -2.840894e-14 -2.184147e-14
## IC-3 -3.666067e-14  2.299795e-14  1.000000e+00 -3.663417e-13 -2.231498e-13
## IC-4  5.749971e-14 -2.840894e-14 -3.663417e-13  1.000000e+00  3.078832e-13
## IC-5  2.715436e-14 -2.184147e-14 -2.231498e-13  3.078832e-13  1.000000e+00
## IC-6  4.349670e-15  3.087219e-15  6.747407e-15  7.338934e-15 -1.486865e-14
## IC-7  4.605827e-15 -1.467735e-14 -1.090233e-13  1.209518e-13  1.115034e-13
## IC-8 -3.408581e-14  1.738701e-14  2.122137e-13 -3.207864e-13 -1.824898e-13
##               IC-6          IC-7          IC-8
## IC-1  4.349670e-15  4.605827e-15 -3.408581e-14
## IC-2  3.087219e-15 -1.467735e-14  1.738701e-14
## IC-3  6.747407e-15 -1.090233e-13  2.122137e-13
## IC-4  7.338934e-15  1.209518e-13 -3.207864e-13
## IC-5 -1.486865e-14  1.115034e-13 -1.824898e-13
## IC-6  1.000000e+00 -2.507132e-14 -1.920189e-15
```

```
## IC-7 -2.507132e-14  1.000000e+00 -7.516248e-14
## IC-8 -1.920189e-15 -7.516248e-14  1.000000e+00
```

```
par(mfrow=c(4,2))
sapply(colnames(Z),function(cname){hist(as.data.frame(Z)[[cname]],
                                        main=cname,xlab="")})
```




```
##           IC-1                        IC-2
## breaks    numeric,13                  numeric,11
## counts    integer,12                  integer,10
## density   numeric,12                  numeric,10
## mids      numeric,12                  numeric,10
## xname     "as.data.frame(Z)[[cname]]" "as.data.frame(Z)[[cname]]"
## equidist TRUE                         TRUE
##           IC-3                        IC-4
```

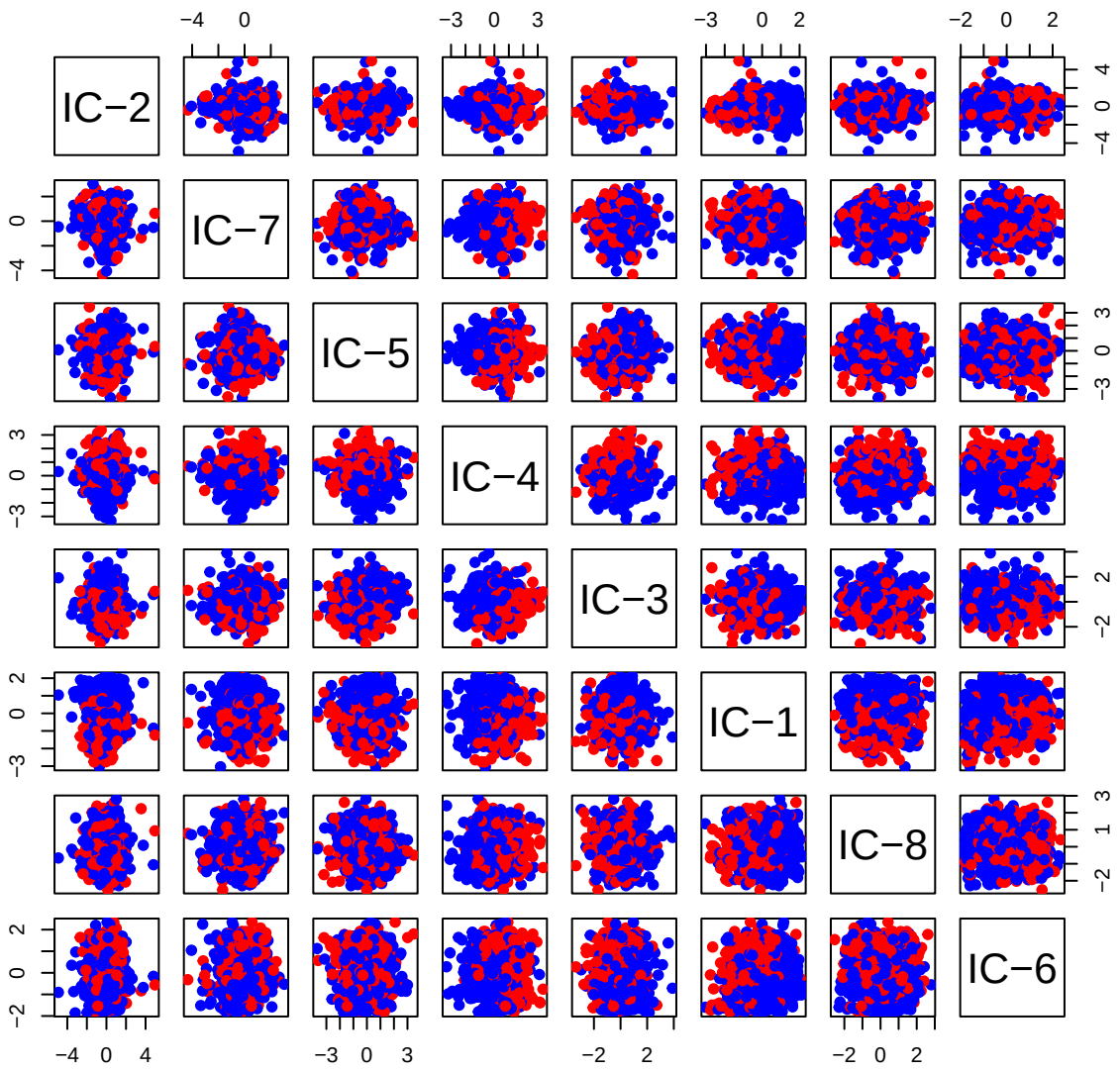
```
## breaks   numeric,16                  numeric,15
## counts   integer,15                  integer,14
## density  numeric,15                  numeric,14
## mids     numeric,15                  numeric,14
## xname    "as.data.frame(Z)[[cname]]" "as.data.frame(Z)[[cname]]"
## equidist TRUE                        TRUE
##          IC-5                        IC-6
## breaks   numeric,16                  numeric,10
## counts   integer,15                  integer,9
## density  numeric,15                  numeric,9
## mids     numeric,15                  numeric,9
## xname    "as.data.frame(Z)[[cname]]" "as.data.frame(Z)[[cname]]"
## equidist TRUE                        TRUE
##          IC-7                        IC-8
## breaks   numeric,17                  numeric,13
## counts   integer,16                  integer,12
## density  numeric,16                  numeric,12
## mids     numeric,16                  numeric,12
## xname    "as.data.frame(Z)[[cname]]" "as.data.frame(Z)[[cname]]"
## equidist TRUE                        TRUE
```

When obtaining the ICs, the histograms are not shown to be highly skewed, but it can also be because that is generating lot of points far from the mean.

```
par(mfrow=c(8,8))

neg_entropy <- function(z){1/12 * mean(z^3)^2 + 1/48 * mean(z^4)^2}
Z_neg_entropy <- apply(Z,2,neg_entropy)
ic_sort <- sort(Z_neg_entropy,decreasing=TRUE,index.return=TRUE)$ix
ic_sort
```

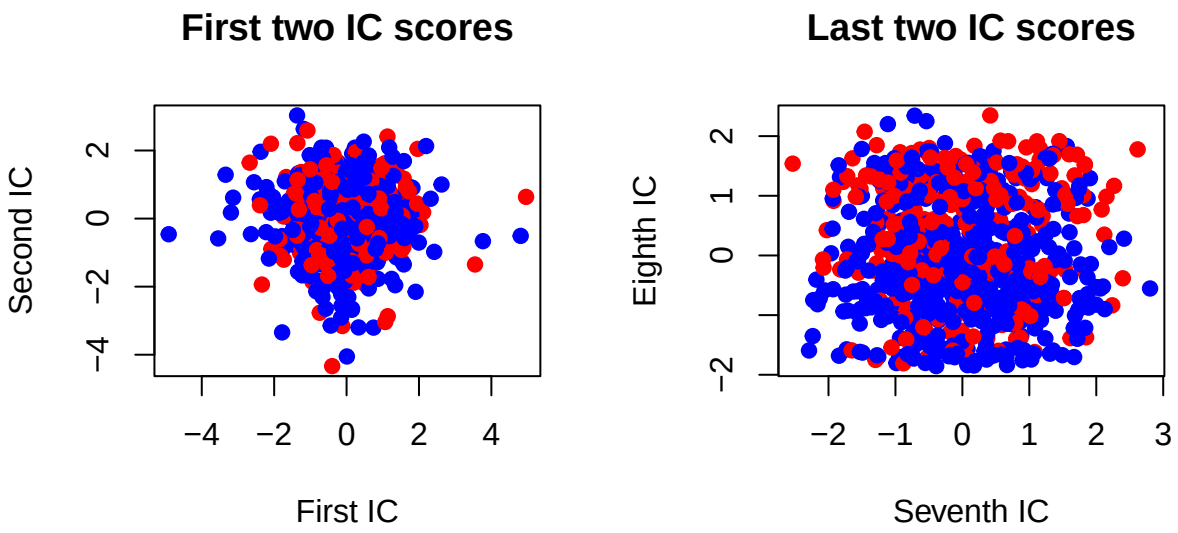
```
## [1] 2 7 5 4 3 1 8 6
```

```
Z_ic_imp <- Z[,ic_sort]

pairs(Z_ic_imp[,1:8],col=diabetes_colors,pch=19,main="Correlation between ICs")
```

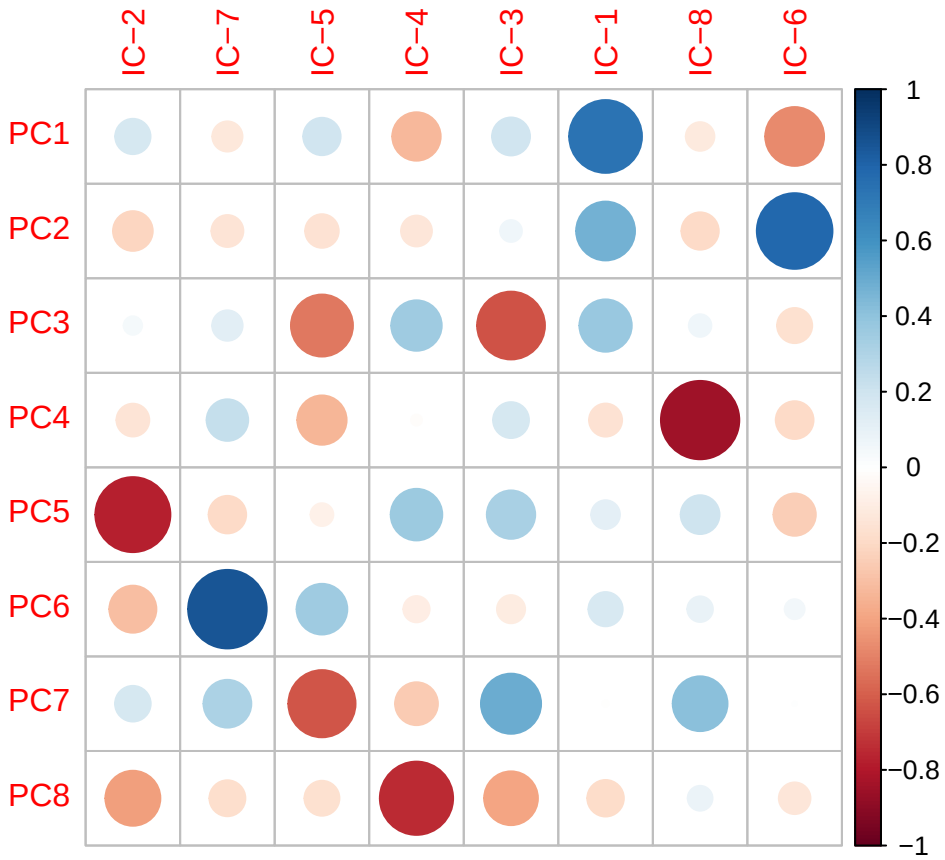


First groups of independent components which have more entropy, shows observations which are far from the means of the variables. Of course, in this case outliers have been removed, still are some points far from the mean which are not classified as outliers by our algorithm presented above. The IC with less entropy, are separating the groups, something similat as the first PC of the

```
par(mfrow=c(1,2))
plot(Z_ic_imp[,1:2],pch=19,col=diabetes_colors,
     xlab="First IC",ylab="Second IC",main="First two IC scores")

plot(Z_ic_imp[,7:8],pch=19,col=diabetes_colors,
     xlab="Seventh IC",ylab="Eighth IC",main="Last two IC scores")
```




```
corrplot(cor(df_quant_pcs$x,Z_ic_imp),is.corr=T)
```



The correlation between PCs and ICs is shown in the graph shows high negative correlation between PC-1 and independent components with lower entropy, which it has been pointed out that are separating the groups.

Factor analysis

1- Estimation of the parameters M_0 and Σ_ν from the original data.

2- Estimation of the parameter M_1 .

3- Final estimation of the M matrix with the varimax criterion. This criterion makes that the factors are highly correlated with some variables while being non-correlated with all the others.

For this assignment, having eight variables, two factors have been considered for this method.

```
df_scale = scale(df_quant)
df_scale_pc <- prcomp(df_scale)
r <- 2  #Choose this number bc explains variance with PCs

#Estimation of M and Sig_nu
M_0 <- df_scale_pc$rotation[,1:r] %*% diag(df_scale_pc$sdev[1:r])  #Use sdev as it is standard deviatio
M_0
```

```
##                                 [,1]        [,2]
## SqrtPregnancies             -0.3573109  0.72305022
## LogGlucose                  -0.6951451  0.01875068
## BloodPressure               -0.5433186  0.15346656
## SkinThickness               -0.6174799 -0.46643154
## LogInsulin                  -0.6897379 -0.04740052
## LogBMI                      -0.6536899 -0.54277964
## LogDiabetesPedigreeFunction -0.2049901 -0.24176526
## LogAge                      -0.5845480  0.63349067
```

```
S <- cov(df_scale)
Sigma_nu_0 <- diag(diag(S - M_0 %*% t(M_0)))
Sigma_nu_0
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.3495273 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [2,] 0.0000000 0.5164217 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [3,] 0.0000000 0.0000000 0.6812529 0.0000000 0.0000000 0.0000000 0.0000000
## [4,] 0.0000000 0.0000000 0.0000000 0.4011602 0.0000000 0.0000000 0.0000000
## [5,] 0.0000000 0.0000000 0.0000000 0.0000000 0.5220148 0.0000000 0.0000000
## [6,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.2780798 0.0000000
## [7,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.8995286
## [8,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##           [,8]
## [1,] 0.0000000
## [2,] 0.0000000
## [3,] 0.0000000
## [4,] 0.0000000
## [5,] 0.0000000
## [6,] 0.0000000
## [7,] 0.0000000
## [8,] 0.2569932
```

```
#Computing MM'
MM <- S - Sigma_nu_0
MM_eig <- eigen(MM)
MM_values <- MM_eig$values
MM_vectors <- MM_eig$vectors
M_1 <- MM_eig$vectors[,1:r] %*% diag(MM_eig$values[1:r])^(1/2)
```

```
M_1
```

```
##            [,1]        [,2]
## [1,] -0.3548882  0.62375717
## [2,] -0.5953149  0.01227858
## [3,] -0.4416923  0.07759134
## [4,] -0.5816565 -0.40777230
## [5,] -0.5909781 -0.03758256
## [6,] -0.6434312 -0.51959290
## [7,] -0.1447272 -0.10501670
## [8,] -0.5870321  0.58935404
```

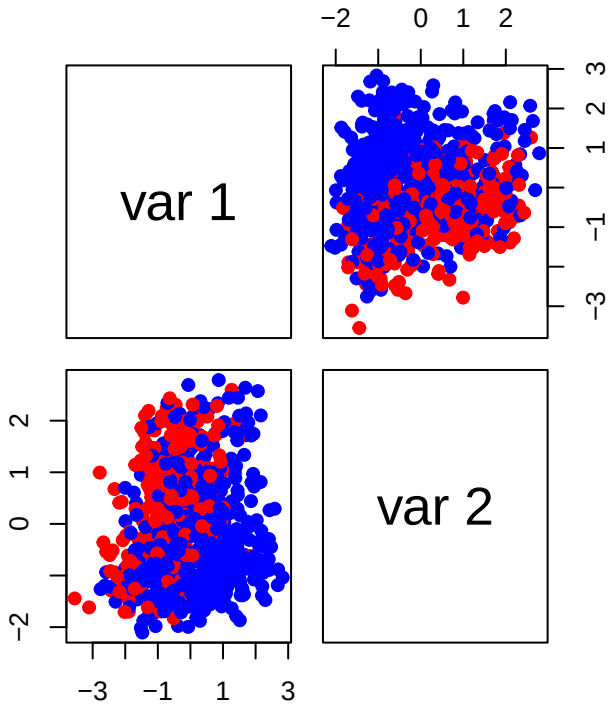
```
#Last estimation
M <- varimax(M_1)
M <- loadings(M)[1:8,1:r]
M
```

```
##             [,1]         [,2]
## [1,]  0.09857199  0.710846127
## [2,] -0.46460851  0.372410384
## [3,] -0.30299540  0.330615029
## [4,] -0.70967763  0.030986263
## [5,] -0.49154550  0.330228163
## [6,] -0.82678796 -0.020054267
## [7,] -0.17874716  0.004890671
## [8,] -0.10647851  0.824989205
```

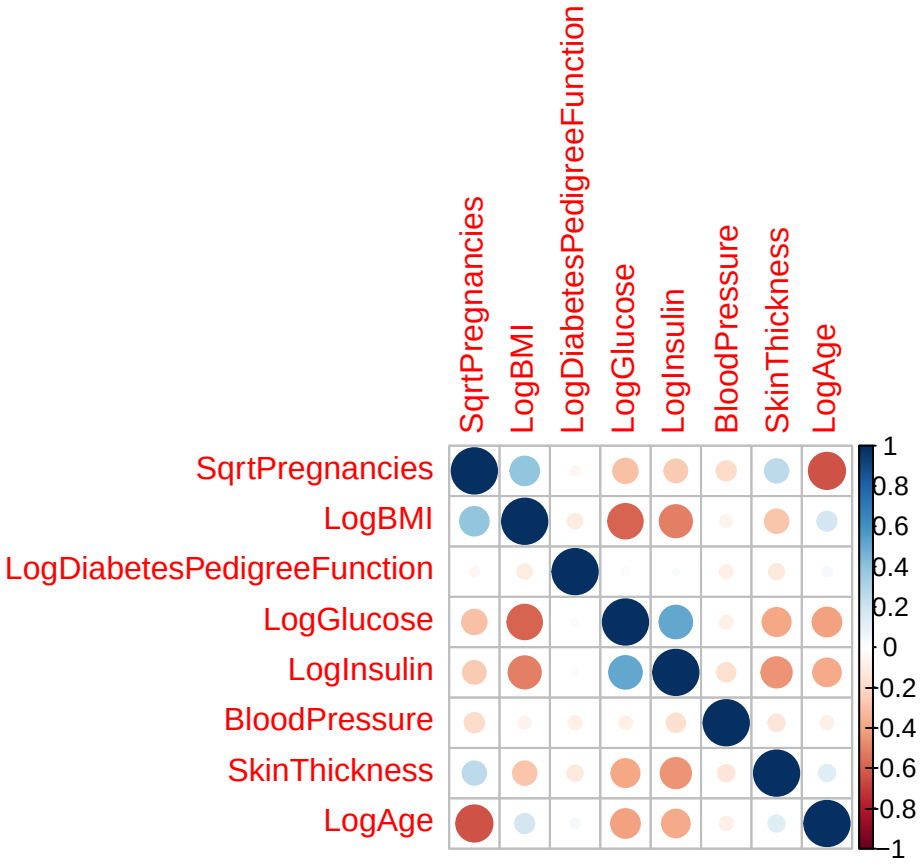
```
Sigma_nu <- diag(diag(S - M %*% t(M)))
Sigma_nu
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4849813 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [2,] 0.0000000 0.6454494 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
## [3,] 0.0000000 0.0000000 0.7988875 0.0000000 0.0000000 0.0000000 0.0000000
## [4,] 0.0000000 0.0000000 0.0000000 0.4953975 0.0000000 0.0000000 0.0000000
## [5,] 0.0000000 0.0000000 0.0000000 0.0000000 0.6493324 0.0000000 0.0000000
## [6,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.3160195 0.0000000
## [7,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.9680255
## [8,] 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000 0.0000000
##           [,8]
## [1,] 0.0000000
## [2,] 0.0000000
## [3,] 0.0000000
## [4,] 0.0000000
## [5,] 0.0000000
## [6,] 0.0000000
## [7,] 0.0000000
## [8,] 0.3080551
```

```
#Factor calculations
Fact <- df_scale %*% solve(Sigma_nu) %*% M %*% solve(t(M) %*% solve(Sigma_nu) %*% M)
pairs(Fact,pch=19,col=diabetes_colors)
```




```
#Error calculations
Nu <- df_scale - Fact %*% t(M)
corrplot(cor(Nu),order="hclust")
```



When looking at the correlation plot for the error matrix, it is clear that the factor model is not explaining all the existing correlation between the variables. For instance, there is a high correlation between the errors of the features $\sqrt{(Pregnancies)}$ and $log(Age)$.

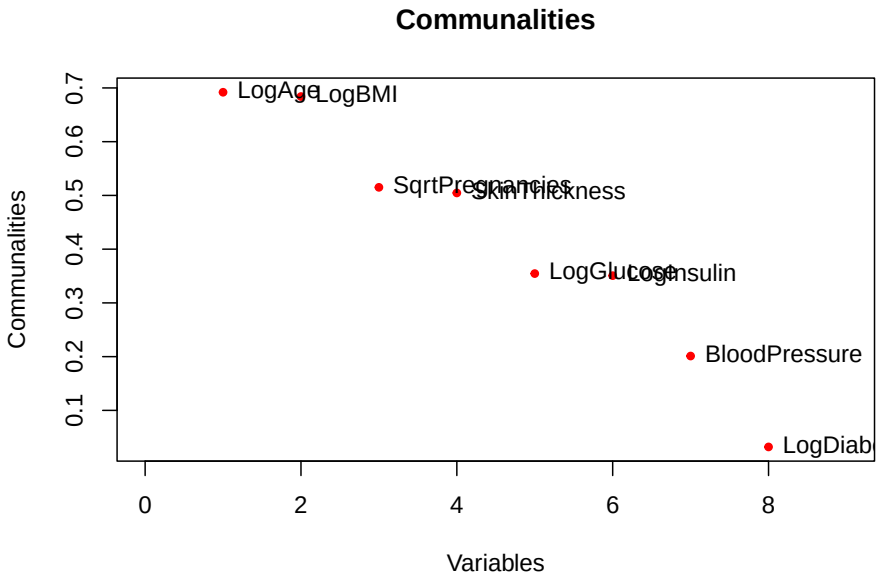
Another important conclusion, is that those error correlations do not depend on the number of factors for this data set. A bigger amount of factors have been tried, with similar or worse results in terms of correlation among errors.

Lastly, the uniqueness and communality are calculated for each variable. The communalities, are the ratio of the variance explained by the factors, while the uniqueness is the ratio of the variance which is not explained by the factors.

```
comM <- diag(M %*% t(M))
comM
```

```
## [1] 0.51501865 0.35455056 0.20111251 0.50460248 0.35066762 0.68398051 0.03197446
## [8] 0.69194486
```

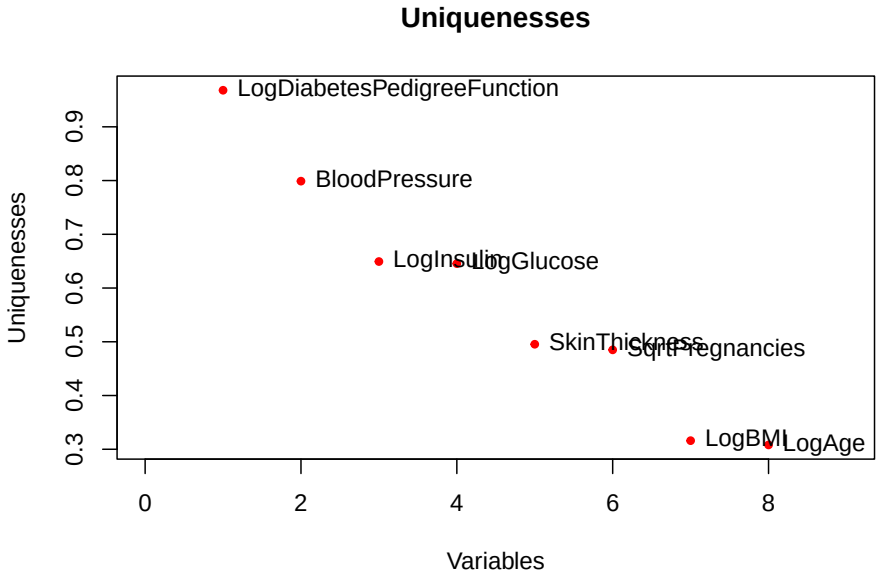
```
names(comM) <- colnames(df_scale)
plot(1:p,sort(comM,decreasing=TRUE),pch=20,col="red",xlim=c(0,9),
     xlab="Variables",ylab="Communalities",main="Communalities")
text(1:p,sort(comM,decreasing=TRUE),labels=names(sort(comM,decreasing=TRUE)),
     pos=4,col="black")
```




```
uniqM <- 1 - comM
uniqM
```

```
##             SqrtPregnancies                 LogGlucose
##                   0.4849813                   0.6454494
##               BloodPressure               SkinThickness
##                   0.7988875                   0.4953975
##                  LogInsulin                      LogBMI
##                   0.6493324                   0.3160195
## LogDiabetesPedigreeFunction                     LogAge
##                   0.9680255                   0.3080551
```

```
plot(1:p,sort(uniqM,decreasing=TRUE),pch=20,col="red",xlim=c(0,9),
     xlab="Variables",ylab="Uniquenesses",main="Uniquenesses")
text(1:p,sort(uniqM,decreasing=TRUE),labels=names(sort(uniqM,decreasing=TRUE)),
     pos=4,col="black")
```



In this particular case, the two factors calculated, explain reasonably $Log(Age)$ and $Log(BMI)$ while do a poor job with $Log(DiabetesPedigree)$ and $BloodPreassure$.

Unsupervised clasification

The aim of unsupervised classification is to see whether or not the groups obtained resemble those defined by the categorical variable outcome, diabetics and non-diabetics.

We only use the quantitative variables to see what groups are formed and for checking if there are other characteristics in the sample that are not covered by outcome.

```
X <- as.data.frame(df_scale)
Y <- diabetes_trans$Outcome

n <- nrow(X)
p<-ncol(X)
```

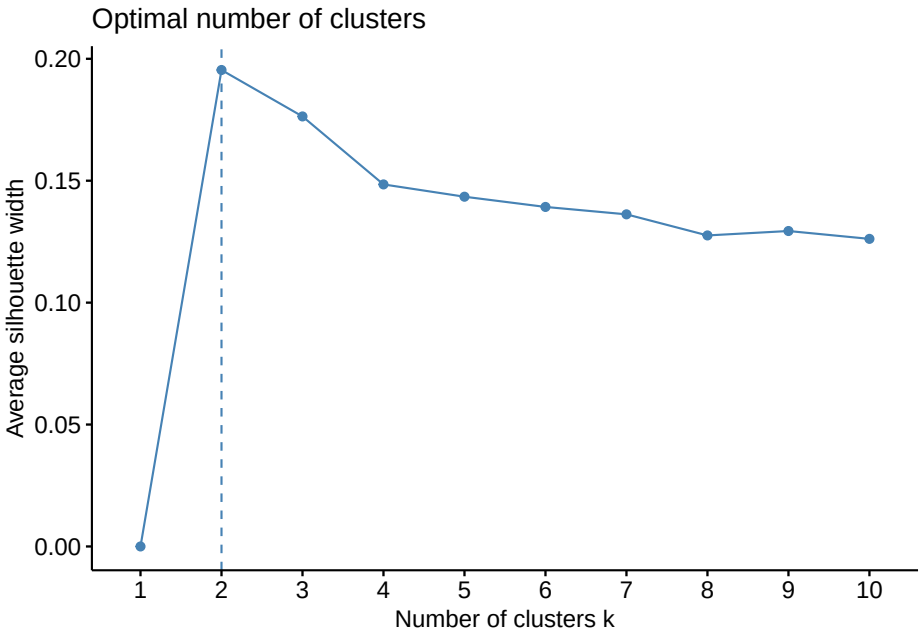
K-Means

The first unsupervised method is K-Means algorithm. First, we have to decide the optimal number of clusters for this dataset. The desired situation is that the optimal number of cluster is two, and the algorithms can distinguish between diabetics and non-diabetics without considering the predictor variable. To calculate the optimal number of clusters we propose:

- Maximizing the average shilouette
- Maximizing the gap statistic (with one standard error and prioritizing small number of clusters)

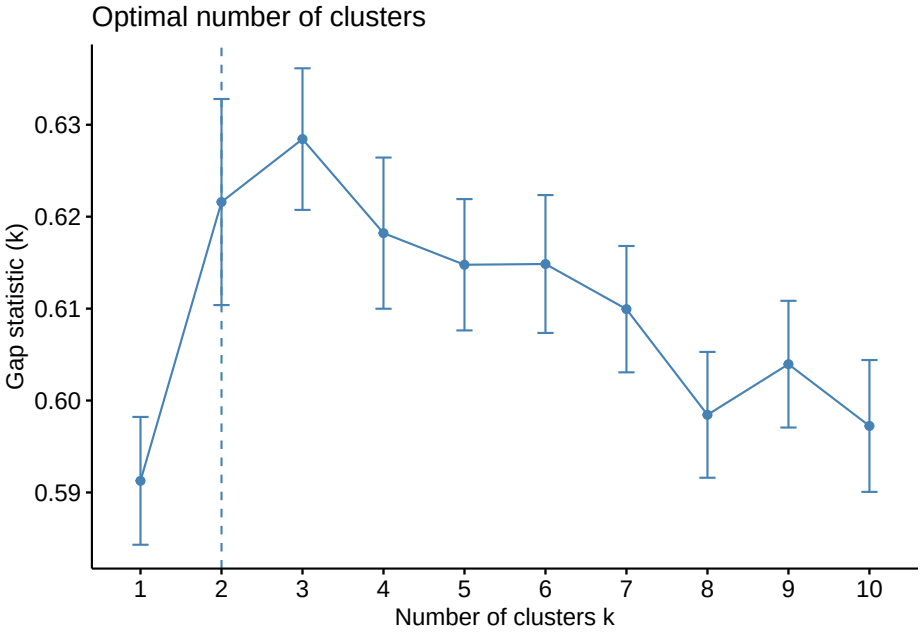
We train K-Mean algorithm for k clusters from 1 to 10 and then compute the average silhouette and the gap statistic:

```
par(mfrow=c(1,1))
fviz_nbclust(X,kmeans,method="silhouette",k.max=10)
```



It is clear that the maximum average shilouette is reached at k=2 which is aligned with the cardinality of the response variable, this could imply that K-Means could be separating those classes.

```
library(cluster)
gap_stat <- clusGap(X,FUN=kmeans,K.max=10,B=100)
fviz_gap_stat(gap_stat,linecolor="steelblue",maxSE=list(method="firstSEmax",SE.factor=1))
```



In the case of the gap statistic, we select the value of K as the smallest k such that the value of Gap (k) is not more than 1 standard deviation error away from the first local maximum, in this case, it is K=2 aswell.

After deciding the cluster number (k=2), the algorithm is trained. 20 random starting points are selected for the method to assure the optimal convergence for this algorithm. We store the best classification of them.

```
cl_kmeans <- kmeans(X,2,nstart=20)
cl_predict <- abs(cl_kmeans$cluster - 2)
```

Finally, we can compare the outcome of the model with the classes of the response to see how similar they are. One key point of this assumption, is that the outcome of the K-Means algorithm is an unsupervised classification, so the prediction of clusters may be switched (0 and 1 may not correspond to 0 and 1 in the response). Classes might be interchanged as the model does not have a reference number (0 or 1) to assign to each cluster, it is just detecting two different groups of instances.

```
ER_kmeans = 1/n*sum(abs(Y-cl_predict))
ER_kmeans
```

```
## [1] 0.2979003
```

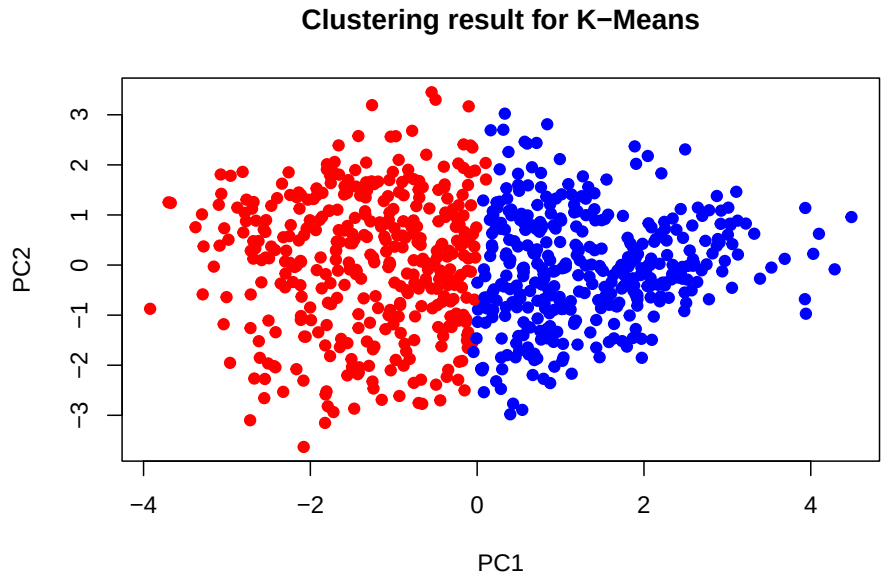
```
confusionMatrix(data=as.factor(cl_predict), reference = as.factor(Y), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 323  53
##          1 174 212
##
##                Accuracy : 0.7021
##                  95% CI : (0.6682, 0.7344)
##     No Information Rate : 0.6522
##     P-Value [Acc > NIR] : 0.001968
##
##                   Kappa : 0.4066
##
##  Mcnemar's Test P-Value : 1.657e-15
##
##             Sensitivity : 0.8000
##             Specificity : 0.6499
##          Pos Pred Value : 0.5492
##          Neg Pred Value : 0.8590
##              Prevalence : 0.3478
##          Detection Rate : 0.2782
##    Detection Prevalence : 0.5066
##       Balanced Accuracy : 0.7249
##
##        'Positive' Class : 1
##
```

The resultant accuracy is around 0.70 which is greater than that obtained with the naive classifier that labels all instances as the majority class non-diabetic (0.65). Notice that the sensitivity of diabetic instances is really high (0.80) but the specificity is not very good (0.65).

Now, we represent the classification of the instances in the two obtained groups. With this purpose we make use of the first two principal components. Notice that the two groups are divided by the first PC. This is due to the employed metric is the euclidean one.

```
cluster_colors <- c("red","blue")[1*(cl_predict==0)+1]
plot(df_quant_pcs$x[,1],df_quant_pcs$x[,2],col=cluster_colors, xlab= "PC1", ylab="PC2",
     main = "Clustering result for K-Means", cex=1, pch=19 )
```



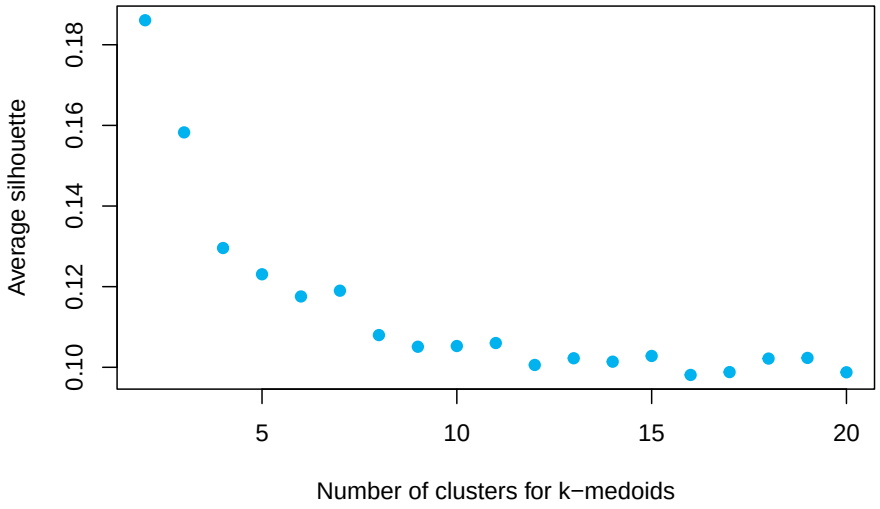
The K-Means is clearly finding two clusters which can be deduced by the first principal component. Recall that the first PC was the component in which we could see diabetics instances for negative values.

K-Medoids

As K-Means is sensitive to outliers because it uses sample mean vectors as cluster centers and the Euclidean distance, we also try K-Medoids. In this case, the metric used is Manhattan distance and the centers will be the medoids.

As in the previous case, we maximize the silhouette value trying different K values. The selected K is 2.

```
X_K <- matrix(NA,nrow=1,ncol=19)
for (i in 1:19){
  pam_X_Gower_mat <- pam(X,k=i+1,diss=FALSE, metric = "euclidean")
  X_K[i] <- pam_X_Gower_mat$silinfo$avg.width
}
plot(2:20,X_K,pch=19,col="deepskyblue2",xlab="Number of clusters for k-medoids",ylab="Average silhouett
```




```
medoids_k <- which.max(X_K)+1
```

We use Partitioning Around Medoids (PAM) that is the standard K-Medoids algorithm to classify the instances in two different clusters. Then we compare with outcome classes to see if they keep any relationship.

```
cl_medoids <- pam(X,k=medoids_k,diss=FALSE, metric= "manhattan")
cl_predict_med <- abs(cl_medoids$clustering - 2)
confusionMatrix(data=as.factor(cl_predict_med), reference = as.factor(Y), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 377  96
##          1 120 169
##
##                Accuracy : 0.7165
##                  95% CI : (0.6831, 0.7483)
##     No Information Rate : 0.6522
##     P-Value [Acc > NIR] : 9.009e-05
##
##                   Kappa : 0.3881
##
##  Mcnemar's Test P-Value : 0.1176
##
##             Sensitivity : 0.6377
##             Specificity : 0.7586
##          Pos Pred Value : 0.5848
##          Neg Pred Value : 0.7970
##              Prevalence : 0.3478
##          Detection Rate : 0.2218
##    Detection Prevalence : 0.3793
##       Balanced Accuracy : 0.6981
##
```

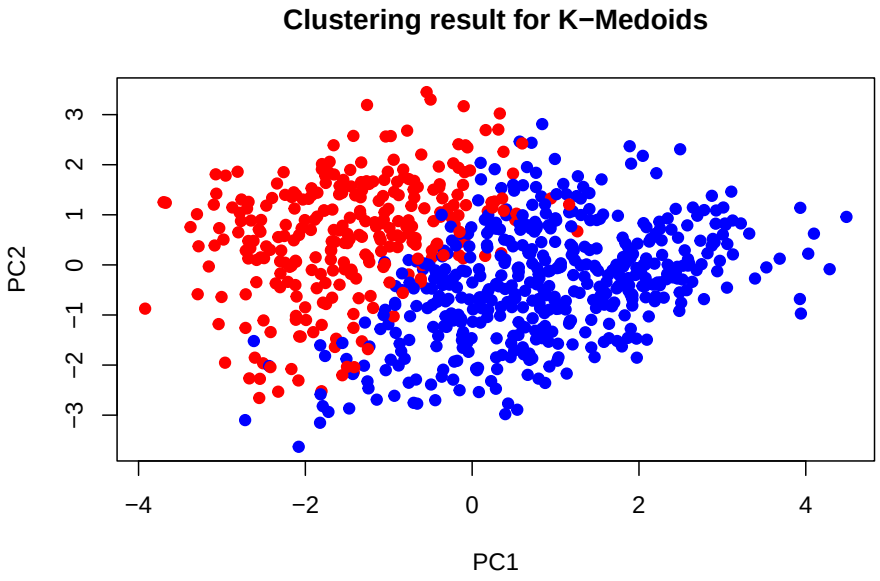
```
##         'Positive' Class : 1
##
```

```
ER_kmeans = 1/n*sum(abs(Y-cl_predict_med))
ER_kmeans
```

```
## [1] 0.2834646
```

After performing the K-Medoids algorithm, we can compare the clusters given to the diabetics response. This method gives almost the same results than K-means in terms of accuracy, but results in terms of sensitivity and specificity vary greatly. It could be said that one of those methods could be selected When prioritizing one of the two metrics (sensitivity/specificity tradeoff). Looking at the graph, we can see that the clusters are separated by the first principal component as the previous case, but has some intrinsic noise inside.

```
cluster_colors <- c("red","blue")[1*(cl_predict_med==0)+1]
plot(df_quant_pcs$x[,1],df_quant_pcs$x[,2],col=cluster_colors, xlab= "PC1", ylab="PC2",
     main = "Clustering result for K-Medoids", cex=1, pch=19 )
```



Hierarchical clustering

Agglomerative algorithm

We start with an agglomerative algorithm. It starts with as many clusters as observations and merges them based on the metric. We try with Manhattan distance. Due to the high number of observations, the correspondent dendogram will not be very visual so we decide not to include it.

```
man_dist_X <- daisy(X,metric="manhattan",stand=FALSE)
single_X <- hclust(man_dist_X,method="single")
cl_single_X <- cutree(single_X,2) - 1
```

```
# Confusion table
confusionMatrix(data=as.factor(cl_single_X), reference = as.factor(Y), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
```

```
## Prediction   0   1
##          0 497 264
##          1   0   1
##
##                Accuracy : 0.6535
##                  95% CI : (0.6186, 0.6873)
##     No Information Rate : 0.6522
##     P-Value [Acc > NIR] : 0.4864
##
##                   Kappa : 0.0049
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.003774
##             Specificity : 1.000000
##          Pos Pred Value : 1.000000
##          Neg Pred Value : 0.653088
##              Prevalence : 0.347769
##          Detection Rate : 0.001312
##    Detection Prevalence : 0.001312
##       Balanced Accuracy : 0.501887
##
##        'Positive' Class : 1
##
```

```
ER_kmeans = 1/n*sum(abs(Y-cl_single_X))
ER_kmeans
```

```
## [1] 0.3464567
```

Single linkage algorithm for agglomerative clustering seems to give poor results, as only one instance is belonging to the minority group. Although single linkage was the algorithm selected to perform agglomerative hierarchical clustering (for its capabilities of creating long clusters), it is clearly not performing well for this set.

Divisive algorithm

The divisive algorithm starts with only one cluster and start dividing it.

```
diana_X <- diana(X,metric="manhattan",diss=FALSE)
cl_diana_X <- abs(cutree(diana_X,2) - 2)

confusionMatrix(data=as.factor(cl_diana_X), reference = as.factor(Y), positive = "1")
```

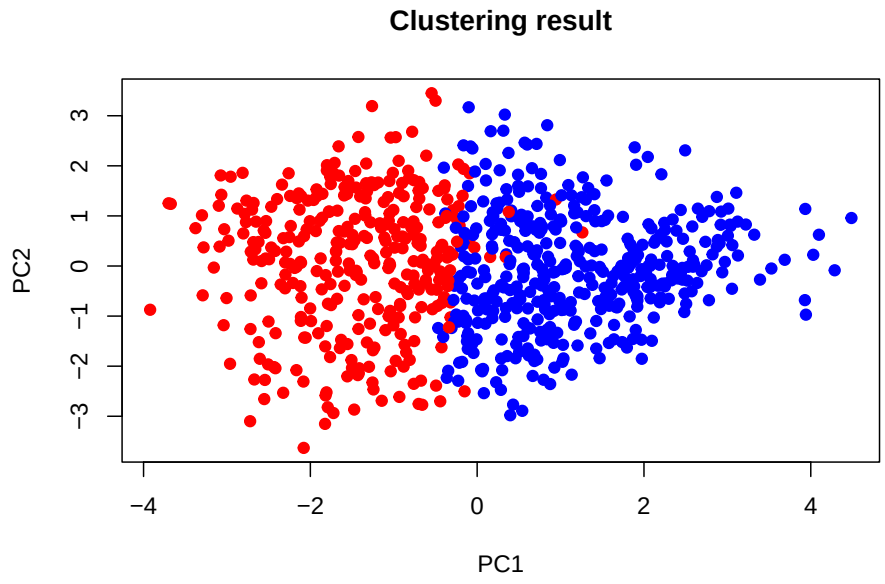
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 353  69
##          1 144 196
##
##                Accuracy : 0.7205
##                  95% CI : (0.6871, 0.7521)
##     No Information Rate : 0.6522
##     P-Value [Acc > NIR] : 3.42e-05
##
```

```
##                  Kappa : 0.422
##
##  Mcnemar's Test P-Value : 3.97e-07
##
##             Sensitivity : 0.7396
##             Specificity : 0.7103
##          Pos Pred Value : 0.5765
##          Neg Pred Value : 0.8365
##              Prevalence : 0.3478
##          Detection Rate : 0.2572
##    Detection Prevalence : 0.4462
##       Balanced Accuracy : 0.7249
##
##        'Positive' Class : 1
##
```

```
ER_diana = 1/n*sum(abs(Y-cl_diana_X))
ER_diana
```

```
## [1] 0.2795276
```

```
cluster_colors <- c("red","blue")[1*(cl_diana_X==0)+1]
plot(df_quant_pcs$x[,1],df_quant_pcs$x[,2],col=cluster_colors, xlab= "PC1", ylab="PC2",
     main = "Clustering result", cex=1, pch=19 )
```



The divisive algorithm ends up with similar results as the one seen from K-Means, separating clases with the first principal component. This particular case presents good accuracy for the true positive class and bad accuracy for the negative class.

Model-based cluster

The idea for this unsupervised algorithm, is to group by model clusters. Those models are multivariate Gaussians with particular characteristics. Those multivariate gaussians are computed maximizing the likelihood. After the models are generated considering the different hypothesis, the model with maximum Bayesian Information Criterion (BIC), is selected to cluster the instances.

The methodologies to implement are the following:

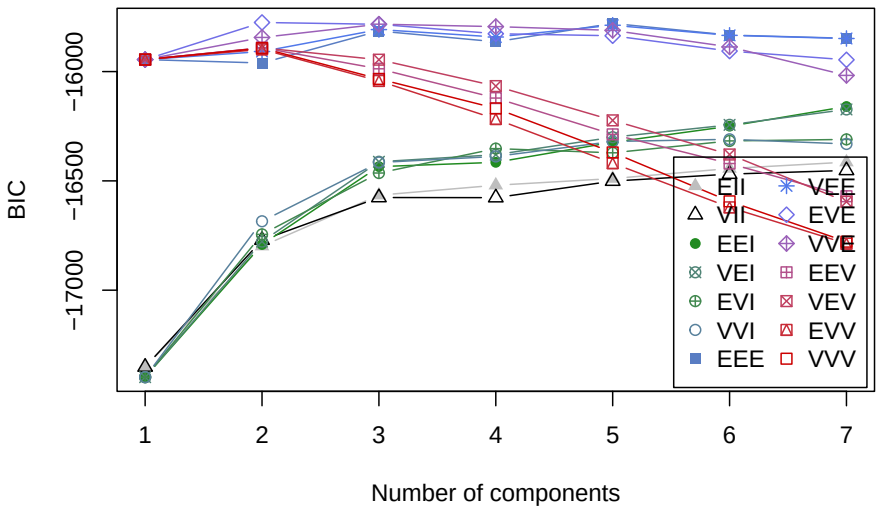
- Ellipsoidal (full covariance matrix)
- Diagonal (diagonal covariance matrix)

With different configurations of volumes and orientations.

```
BIC_X <- mclustBIC(X,G=1:7)
BIC_X
```

```
## Bayesian Information Criterion (BIC):
##         EII       VII       EEI       VEI       EVI       VVI       EEE
## 1 -17351.42 -17351.42 -17397.87 -17397.87 -17397.87 -17397.87 -15945.15
## 2 -16798.67 -16771.05 -16791.14 -16777.80 -16744.77 -16684.84 -15960.72
## 3 -16565.47 -16576.13 -16434.77 -16412.53 -16464.04 -16415.79 -15814.45
## 4 -16519.93 -16576.72 -16415.15 -16380.19 -16352.40 -16388.26 -15863.72
## 5 -16489.41 -16501.13 -16323.38 -16300.21 -16371.58 -16319.74 -15779.77
## 6 -16443.44 -16469.05 -16249.85 -16243.77 -16317.21 -16309.88 -15833.06
## 7 -16414.31 -16452.25 -16159.78 -16173.29 -16310.28 -16331.11 -15847.66
##         VEE       EVE       VVE       EEV       VEV       EVV       VVV
## 1 -15945.15 -15945.15 -15945.15 -15945.15 -15945.15 -15945.15 -15945.15
## 2 -15908.24 -15775.43 -15843.80 -15891.05 -15891.39 -15898.21 -15892.64
## 3 -15807.84 -15783.70 -15783.22 -15987.32 -15945.62 -16042.77 -16033.57
## 4 -15841.84 -15826.90 -15794.35 -16121.11 -16066.39 -16216.94 -16168.29
## 5 -15788.07 -15836.04 -15811.56 -16287.19 -16223.14 -16418.97 -16369.79
## 6 -15834.28 -15906.19 -15887.15 -16420.71 -16379.29 -16621.01 -16593.58
## 7 -15848.69 -15946.20 -16017.47 -16570.29 -16595.43 -16787.11 -16778.21
##
## Top 3 models based on the BIC criterion:
##     EVE,2     EEE,5     VVE,3
## -15775.43 -15779.77 -15783.22
```

```
# Have a look at the results

plot(BIC_X)
```



The selected algorithm is the ellipsoidal, equal volume and orientation with two clusters.

```
Mclust_X <- Mclust(X,x=BIC_X)$classification
cl_Mclust_X  <- abs(Mclust_X - 2)

confusionMatrix(data=as.factor(cl_Mclust_X ), reference = as.factor(Y), positive = "1")
```

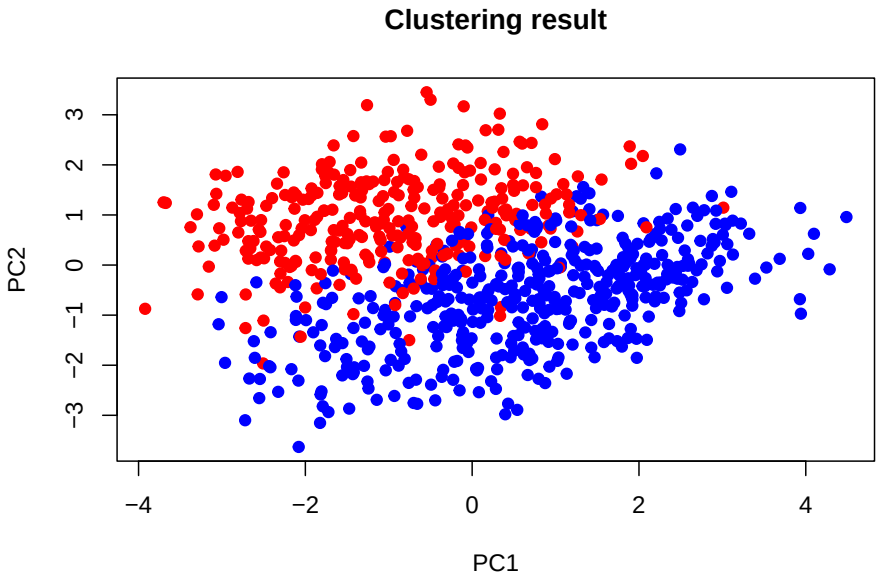
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 345 112
##          1 152 153
##
##                Accuracy : 0.6535
##                  95% CI : (0.6186, 0.6873)
##     No Information Rate : 0.6522
##     P-Value [Acc > NIR] : 0.48637
##
##                   Kappa : 0.2623
##
##  Mcnemar's Test P-Value : 0.01638
##
##             Sensitivity : 0.5774
##             Specificity : 0.6942
##          Pos Pred Value : 0.5016
##          Neg Pred Value : 0.7549
##              Prevalence : 0.3478
##          Detection Rate : 0.2008
##    Detection Prevalence : 0.4003
##       Balanced Accuracy : 0.6358
##
##        'Positive' Class : 1
##
```

```
ER_Mclust  = 1/n*sum(abs(Y-cl_Mclust_X ))
ER_Mclust
```

```
## [1] 0.3464567
```

Although the BIC is selected to perform the best model, the accuracy and the prediction of the true positive class is very poor.

```
cluster_colors <- c("red","blue")[1*(cl_Mclust_X==0)+1]
plot(df_quant_pcs$x[,1],df_quant_pcs$x[,2],col=cluster_colors, xlab= "PC1", ylab="PC2",
     main = "Clustering result", cex=1, pch=19 )
```



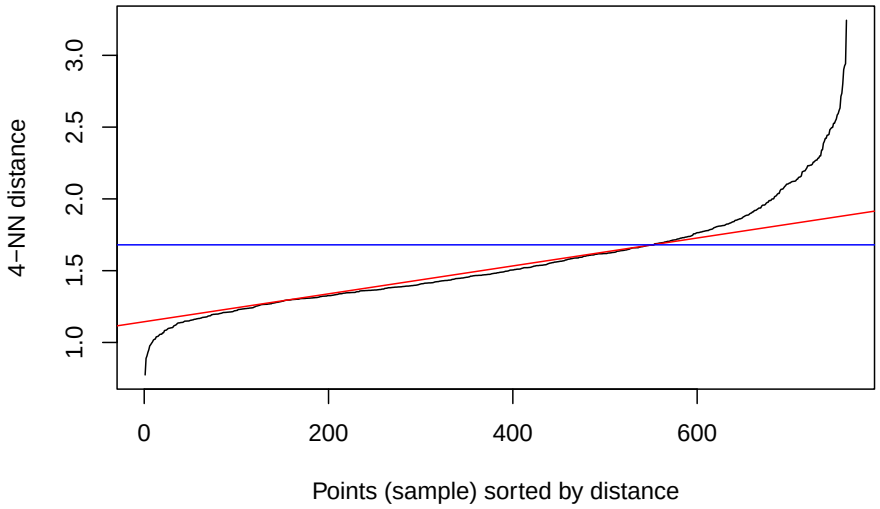
We can observe that the two groups are not well differentiated by the PC1 and what is more they are better determined by PC2. This is the reason that the makes the groups not match with diabetics and non-diabetics.

Density-based (DBSCAN)

The density based clustering considers different regions in which the density of points is very large, while separating the classes in the zones of less density. Before constructing the model, it could be assumed that the model will not perform well, as the separation between classes does not happens in a low density zone, quite the opposite (see Principal Component Analysis section).

For this model we will assume MP (minimum number of close observations) to 5, usual value fo that parameter. The threshold then is selected at the change in slope for the graph below:

```
minPts <- 5
kNNdistplot(X,k=minPts-1)
abline(a=1.14444,b=.000972,col="red")
abline(h=1.68, col="blue")
```



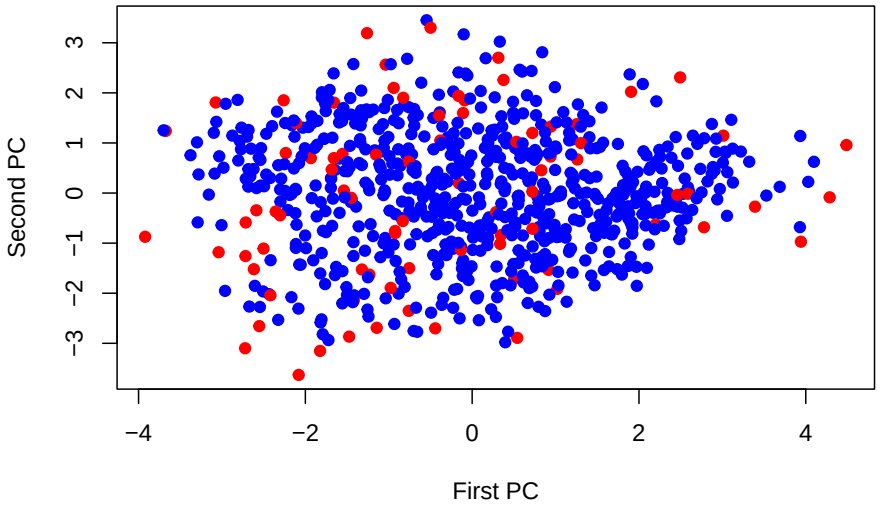
$$\epsilon = 1.68$$

After selecting the two parameters of the model, it can be constructed and every instance is assigned into a cluster.

```
dbscan_X<- dbscan(X,eps=1.68,minPts=minPts)
dbscan_X
```

```
## DBSCAN clustering for 762 objects.
## Parameters: eps = 1.68, minPts = 5
## Using euclidean distances and borderpoints = TRUE
## The clustering contains 1 cluster(s) and 82 noise points.
##
##   0   1
##  82 680
##
## Available fields: cluster, eps, minPts, dist, borderPoints
```

```
colors_dbscan_X <- c("red","blue")[dbscan_X$cluster+1]
plot(df_quant_pcs$x[,1],df_quant_pcs$x[,2],pch=19,col=colors_dbscan_X,xlab="First PC",ylab="Second PC")
```



As it happened before, the algorithm struggles to find the two clusters in which diabetics and non-diabetics are contained, instead, the model just finds one unique cluster and many points which are noisy instances (points far from the mean vector).

Supervised clasification

There are many methods available to perform supervised classification and no method dominates all over all possible data sets, so we need to try every different method to compare the different performances.

we are going to make use of the outcome variable to try to get better classifications of the instances between diabetics and non-diabetics. For this reason, we divide the data set in training sample for tuning the parameters and test sample for evaluating the model performance (holdout). The classic proportions are used in the split, 70% for training and 30% for testing.

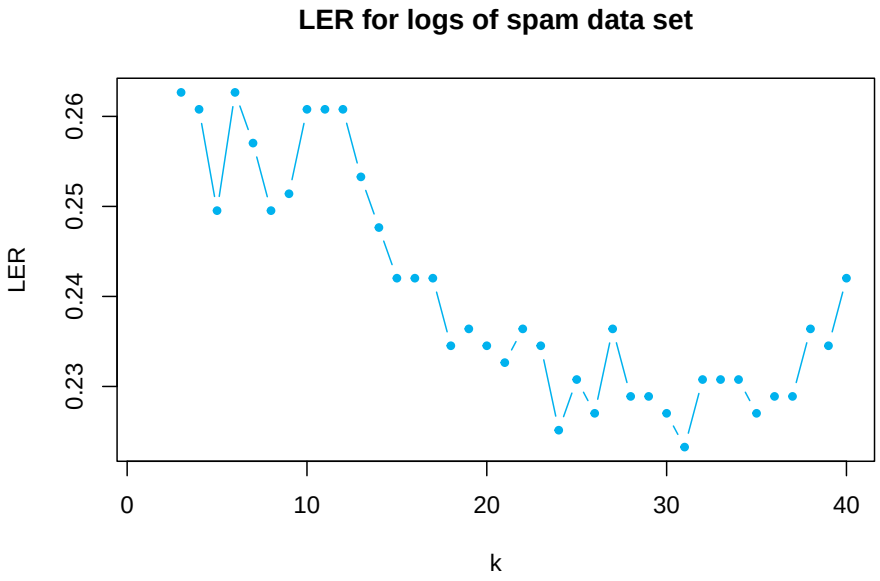
```
n_train <- floor(.7*n)
n_test <- n - n_train
shuf <- sample(1:n,n)
i_train<-shuf[1:n_train]
i_test<-shuf[(n_train+1):n]
X_train <- X[i_train,]
Y_train <- Y[i_train]
X_test <- X[i_test,]
Y_test <- Y[i_test]
```

KNN

The first supervised algorithm we are studying will be K-Nearest Neighbors (KNN) which classifies the instances according to the most common class among its neighbors. We use LOOCV (Leave one out cross validation) for optimizing the parameter K referring to the number of neighbors considered by minimizing the error rate.

```
LER <- rep(NA,40)
for (i in 3 : 40){
  knn_output <- knn.cv(X_train,Y_train,k=i)
```

```
  LER[i] <- 1 - mean(knn_output==Y_train)
}
plot(1:40,LER,pch=20,col=color_1,type="b",
     xlab="k",ylab="LER",main="LER for logs of spam data set")
```




```
# Take k as the one that gives the minimum LOOCV error rate

k <- which.min(LER)
k
```

```
## [1] 31
```

After defining the number of neighbours, the final model can be constructed with the training test. After constructing the model, it is evaluated using the test set. The best value for the parameter K is 31 neighbors to consider. We represent the confusion matrix and the more representative statistics for the classification.

```
# Classify the responses in the test data set with the k selected
knn_Y_test <- knn(X_train,X_test,Y_train,k=k,prob=T)

confusionMatrix(data=as.factor(knn_Y_test), reference = as.factor(Y_test), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 134  45
##          1  11  39
##
##                Accuracy : 0.7555
##                  95% CI : (0.6945, 0.8097)
##     No Information Rate : 0.6332
##     P-Value [Acc > NIR] : 5.262e-05
##
##                   Kappa : 0.4246
##
```

```
##  Mcnemar's Test P-Value : 1.035e-05
##
##             Sensitivity : 0.4643
##             Specificity : 0.9241
##          Pos Pred Value : 0.7800
##          Neg Pred Value : 0.7486
##              Prevalence : 0.3668
##          Detection Rate : 0.1703
##    Detection Prevalence : 0.2183
##       Balanced Accuracy : 0.6942
##
##        'Positive' Class : 1
##
```

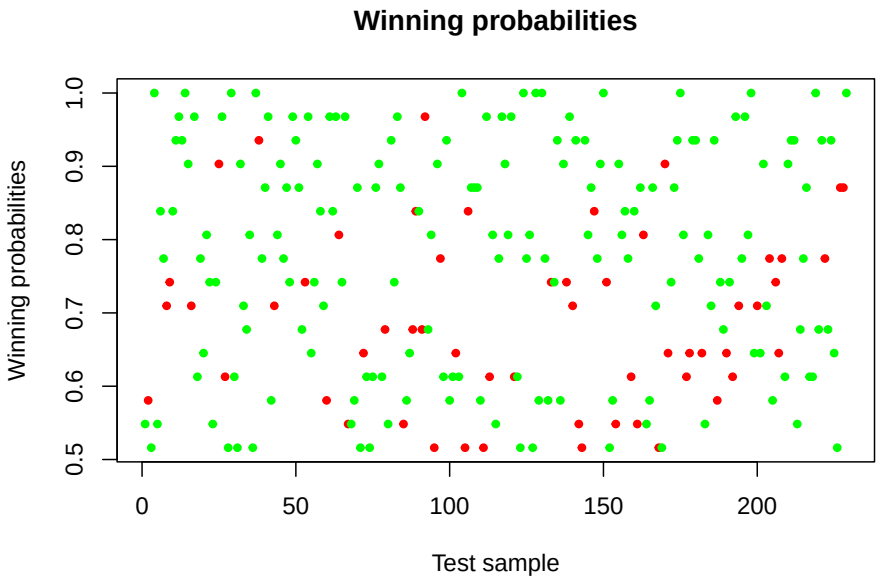
```
knn_num = as.numeric(knn_Y_test) - 1
LER_KNN  = 1/n_test*sum(abs(Y_test-knn_num))
LER_KNN
```

```
## [1] 0.2445415
```

The results improve greatly against the unsupervised ones. The overall accuracy is 0.76 and the specificity is 0.92. The problem is that sensitivity is only 0.46 which implies that less than half of diabetics group is found. This is due to the positive class is minority.

We show the performance of the classification against the winning probabilities. As it was expected the vast proportion of the errors are produced when the probabilities are near to the threshold 0.5 which decides the class to label.

```
prob_knn_Y_test <- attributes(knn_Y_test)$prob

# In green, good classifications, in red, wrong classifications

colors_errors <- c("red","green")[1*(Y_test==knn_Y_test)+1]
plot(1:n_test,prob_knn_Y_test,col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Winning probabilities",main="Winning probabilities")
```



Note that one way to improve this model, or at least improve the sensitivity (at specificity cost) could be to tune the threshold and generate a more balanced model.

Methods based on Bayes Theorem

The conditional probabilities for belonging an instance to a given class can be computed thanks to the Bayes theorem. With this aim, prior probabilities and the density function of each class must be estimated. We are assuming that the densities are multivariate Gaussian, which gives rise to the linear and quadratic classifiers.

Linear discriminant analysis - LDA

In this method, we assume that the mean vectors of the two classes are different but the covariance matrix is the same. We estimate the hyper-parameters prior probabilities π_0, π_1, mean vectors μ_0, μ_1 and covariance matrix Σ in the training sample. Then we evaluate in the training set the performance of the model.

```
# Estimate the unknown parameters with the training sample
lda_train <- lda(Y_train~.,data=as.data.frame(X_train))

# Classify the observations in the test sample
lda_test <- predict(lda_train,newdata=as.data.frame(X_test))

# The vector of classifications made can be found here
lda_Y_test <- lda_test$class

confusionMatrix(data=lda_Y_test, reference = as.factor(Y_test), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 127  39
##          1  18  45
##
##                Accuracy : 0.7511
##                  95% CI : (0.6899, 0.8057)
##     No Information Rate : 0.6332
##     P-Value [Acc > NIR] : 9.507e-05
##
##                   Kappa : 0.4344
##
##  Mcnemar's Test P-Value : 0.008071
##
##             Sensitivity : 0.5357
##             Specificity : 0.8759
##          Pos Pred Value : 0.7143
##          Neg Pred Value : 0.7651
##              Prevalence : 0.3668
##          Detection Rate : 0.1965
##    Detection Prevalence : 0.2751
##       Balanced Accuracy : 0.7058
##
##        'Positive' Class : 1
##
```

```
LDA_num = as.numeric(lda_Y_test) - 1
TER_LDA  = 1/n_test*sum(abs(Y_test-LDA_num))
```

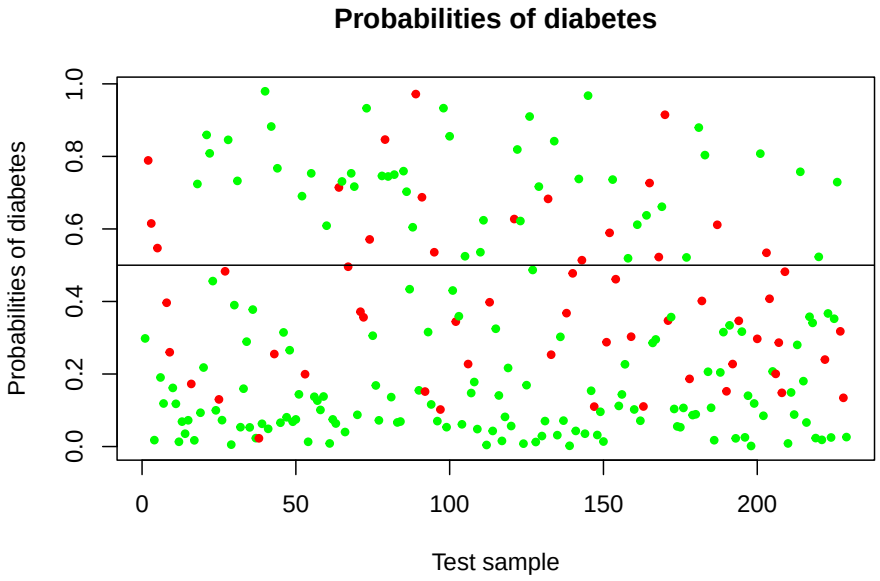
```
TER_LDA
```

```
## [1] 0.2489083
```

The accuracy is almost the same as the previous obtained with KNN. Although the specificity is reduced a bit, the sensitivity grows to 0.53 which still is considered to be a bad result.

We plot the instances against the winning probabilities. As it was expected the red points representing the miss classifications are gathered around the central line.

```
# Conditional probabilities of the classifications made with the test sample
prob_lda_Y_test <- lda_test$posterior
colors_errors <- c("red","green")[1*(Y_test==LDA_num)+1]
plot(1:n_test,prob_lda_Y_test[,2],col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Probabilities of diabetes",
     main="Probabilities of diabetes")
abline(h=0.5)
```



Quadratic discriminant analysis - QDA

In this method, we assume that the mean vectors and the covariance matrices of the two classes are different. We estimate the hyper-parameters prior probabilities π_0, π_1, mean vectors μ_0, μ_1 and covariance matrices Σ_0, Σ_1 in the training sample. Then we evaluate in the training set the performance of the model.

```
# Estimate the unknown parameters with the training sample

qda_train <- qda(Y_train~.,data=as.data.frame(X_train))

# Classify the observations in the test sample

qda_test <- predict(qda_train,newdata=as.data.frame(X_test))

# The vector of classifications made can be found here

qda_Y_test <- qda_test$class
```

```r
confusionMatrix(data=qda_Y_test, reference = as.factor(Y_test), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 124  34
##          1  21  50
##
##                Accuracy : 0.7598
##                  95% CI : (0.6991, 0.8136)
##     No Information Rate : 0.6332
##     P-Value [Acc > NIR] : 2.847e-05
##
##                   Kappa : 0.4656
##
##  Mcnemar's Test P-Value : 0.1056
##
##             Sensitivity : 0.5952
##             Specificity : 0.8552
##          Pos Pred Value : 0.7042
##          Neg Pred Value : 0.7848
##              Prevalence : 0.3668
##          Detection Rate : 0.2183
##    Detection Prevalence : 0.3100
##       Balanced Accuracy : 0.7252
##
##        'Positive' Class : 1
##
```

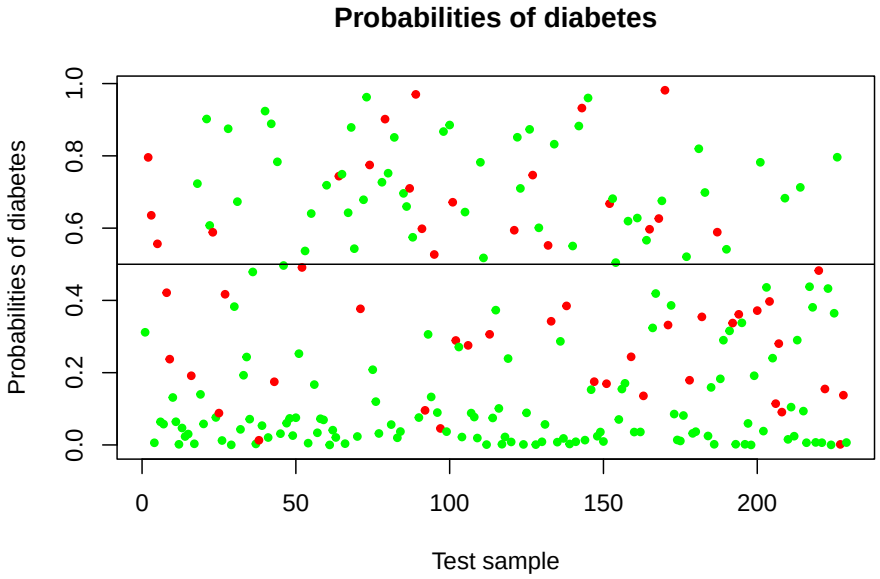
```r
QDA_num = as.numeric(qda_Y_test) - 1
TER_QDA  = 1/n_test*sum(abs(Y_test-QDA_num))
TER_QDA
```

```
## [1] 0.2401747
```

We observe very similar results than the ones seen in the previous model (LDA). This better classification of the positive instances is derived by considering a specific covariance matrix for the diabetic population.

We plot the conditional probabilities of winning. In green the successes and in red the failures.

```r
# Conditional probabilities of the classifications made with the test sample
prob_qda_Y_test <- qda_test$posterior
colors_errors <- c("red","green")[1*(Y_test==QDA_num)+1]
plot(1:n_test,prob_qda_Y_test[,2],col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Probabilities of diabetes",
     main="Probabilities of diabetes")
abline(h=0.5)
```



Naive Bayes

This method assumes that the predictors are independent variables. Therefore, the variances are estimated but the covariances are assumed to be just 0, so there is no need to estimate them. This can be justified in terms of computational flexibility but, Naive Bayes should perform worse than both LDA and QDA.

We estimate the parameters with the training sample. In this case the covariance matrix will be diagonal saving computational time.

```
nb_train <- gaussian_naive_bayes(as.matrix(X_train),as.factor(Y_train))

# Classify the observations in the test sample

nb_test <- predict(nb_train,newdata= as.matrix(X_test),type="prob")
nb_Y_test <- c(0,1)[apply(nb_test,1,which.max)]
```

```
confusionMatrix(data=as.factor(nb_Y_test), reference = as.factor(Y_test),positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 115  34
##          1  30  50
##
##                Accuracy : 0.7205
##                  95% CI : (0.6576, 0.7776)
##     No Information Rate : 0.6332
##     P-Value [Acc > NIR] : 0.003266
##
##                   Kappa : 0.3923
##
##  Mcnemar's Test P-Value : 0.707660
##
```

```
##             Sensitivity : 0.5952
##             Specificity : 0.7931
##          Pos Pred Value : 0.6250
##          Neg Pred Value : 0.7718
##              Prevalence : 0.3668
##          Detection Rate : 0.2183
##    Detection Prevalence : 0.3493
##       Balanced Accuracy : 0.6942
##
##        'Positive' Class : 1
##
```

```
nb_num = as.numeric(nb_Y_test) - 1
TER_NB  = 1/n_test*sum(abs(Y_test-nb_num))
TER_NB
```

```
## [1] 1.017467
```

As expected, Naive Bayes obtains worse results than the previous methods, will a very low sensitivity, which seems to be the worse defect of all these methods.

Logistic regression

We employ logistic regression to classify the instances by its probabilities. First we fit the model in the training set to estimate the regression coefficients $\beta_0, \dots \beta_8$ and then we evaluate in the test set.

```
# Estimate the parameters of the logistic regression model with the training sample
lr_train <- multinom(Y_train~.,data=as.data.frame(X_train))
```

```
## # weights:  10 (9 variable)
## initial  value 369.447447
## iter  10 value 242.033223
## final  value 238.382459
## converged
```

```
# Classify the responses in the test data set and estimate the test error rate
lr_test <- predict(lr_train,newdata=as.data.frame(X_test))

confusionMatrix(data=as.factor(lr_test), reference = as.factor(Y_test),positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 129  39
##          1  16  45
##
##                Accuracy : 0.7598
##                  95% CI : (0.6991, 0.8136)
##     No Information Rate : 0.6332
##     P-Value [Acc > NIR] : 2.847e-05
##
##                   Kappa : 0.4514
##
##  Mcnemar's Test P-Value : 0.003012
##
##             Sensitivity : 0.5357
```

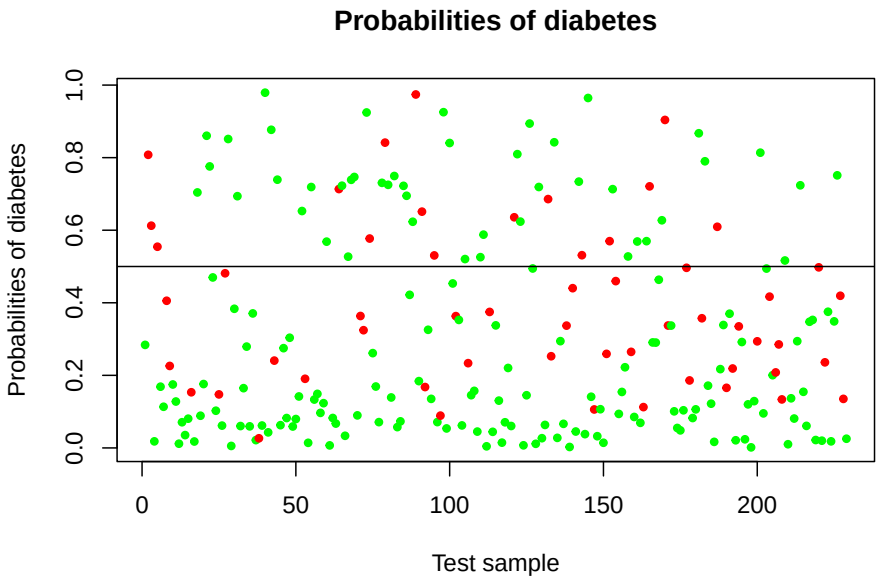
```
##             Specificity : 0.8897
##          Pos Pred Value : 0.7377
##          Neg Pred Value : 0.7679
##              Prevalence : 0.3668
##          Detection Rate : 0.1965
##    Detection Prevalence : 0.2664
##       Balanced Accuracy : 0.7127
##
##        'Positive' Class : 1
##
```

```
lr_num = as.numeric(lr_test) - 1
TER_lr  = 1/n_test*sum(abs(Y_test-lr_num))
TER_lr
```

```
## [1] 0.2401747
```

This model produces one of the best accuracy 0.76 but the sensitivity decreases to 0.54 so the diabetics group is still not well captured.

```
# Conditional probabilities of the classifications made with the test sample
prob_lr_test <- predict(lr_train,newdata=X_test,type ="probs")
colors_errors <- c("red","green")[1*(Y_test==lr_num)+1]
plot(1:n_test,prob_lr_test,col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Probabilities of diabetes",
     main="Probabilities of diabetes")
abline(h=0.5)
```



By means of a t-test, we rank the different coefficients score to order its relevance in the former model. As we can see LogInsulin is in the last position and it can be because it was a variable with many missing values and the imputation performed could not be perfect for this reason.

```
# To see which are the most significant coefficients, we use the t-test that is computed next
t_test_lr_train <- summary(lr_train)$coefficients/summary(lr_train)$standard.errors

# Sort the absolute value of the t-test in decreasing order
```

```
sort(abs(t_test_lr_train),decreasing=TRUE)
```

```
##                  (Intercept)                   LogGlucose
##                    8.4623763                    7.2638175
##                       LogBMI                       LogAge
##                    4.8158755                    3.5189658
## LogDiabetesPedigreeFunction                   LogInsulin
##                    3.2633747                    1.3387390
##              SqrtPregnancies                SkinThickness
##                    0.6372691                    0.5697498
##               BloodPressure
##                    0.3815724
```

Although we are not considering many different variables, we will perform a BIC selection to check if removing some of them we obtain a model with a better performance.

```
# Try to improve the test error rate by deleting predictors without discriminatory power
step_lr_train <- step(lr_train,direction="backward",trace=1,  K= "log(n)")
```

After performing the BIC selection, those four variables are the selected predictors:

```
#Show the selected predictors
step_lr_train$coefnames
```

```
## [1] "(Intercept)"                "LogGlucose"
## [3] "LogBMI"                     "LogDiabetesPedigreeFunction"
## [5] "LogAge"
```

```
# Classify the responses in the test data set with this new model
step_lr_test <- predict(step_lr_train,newdata=X_test)

confusionMatrix(data=as.factor(step_lr_test), reference = as.factor(Y_test),positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 126  38
##          1  19  46
##
##                Accuracy : 0.7511
##                  95% CI : (0.6899, 0.8057)
##     No Information Rate : 0.6332
##     P-Value [Acc > NIR] : 9.507e-05
##
##                   Kappa : 0.4374
##
##  Mcnemar's Test P-Value : 0.01712
##
##             Sensitivity : 0.5476
##             Specificity : 0.8690
##          Pos Pred Value : 0.7077
##          Neg Pred Value : 0.7683
##              Prevalence : 0.3668
##          Detection Rate : 0.2009
##    Detection Prevalence : 0.2838
##       Balanced Accuracy : 0.7083
```

```
##
##        'Positive' Class : 1
##
```

```
step_lr_num = as.numeric(step_lr_test) - 1
TER_lr  = 1/n_test*sum(abs(Y_test-step_lr_num))
TER_lr
```

```
## [1] 0.2489083
```

We obtain similar results, so the final step is to tune the hyper-parameters of the model to get a more balanced model with similar specificity or sensitivity.

We obtain an improvement applying predictor selection method in the accuracy but the true positive rate of diabetes remains very low.

Finally, we will try to tune some hyper-parameters of the logistic model, in order to obtain a more balanced model. The hyper-parameter to tune will be the $F1 - score$ which is a metric heavily used in unbalanced cases.

$$F1 = \frac{2 * precision * recall}{precision + recall}$$

Where:

- Recall: $\frac{tp}{tp+fp}$
- Precision: $\frac{tp}{tp+fn}$

and

tp:true positives, fp:false positives, tn:true negatives, fn:false negatives

The hyper-parameters to tune will be the threshold which classifies instances from the probabilities given by the model, and the weights in which instances are being considered. We will use the grid search (two nested 'for') method in order to obtain the best pair of parameter for maximizing the F1-score.

```
F1_score = 0
threshold = 0
acc_tuned = 0
sens_tuned = 0
spec_tuned = 0

alpha_vec = seq(0.25,0.75,0.025)
w_vec <- c(4,4.5,5,5.5,6,6.5,7)

ind<-which(Y_train==1)
weights <- rep(1,length(Y_train))


for (alpha in alpha_vec){
  for (w in w_vec){
    weights[ind] <- w

    lr_hpo <- multinom(Y_train ~ LogDiabetesPedigreeFunction + LogAge + LogBMI + LogGlucose,
                       data=as.data.frame(X_train),weights = weights)
    Y_pred <- predict(lr_hpo, newdata=X_train, type='probs')
    H <- table(Y_pred > alpha, Y_train == 1)
    tp = H[2,2]
```

```
    tn = H[1,1]
    fp = H[2,1]
    fn = H[1,2]
    total = tp+tn+fp+fn
    # Calculating F1-score
    acc = (tp+tn)/total
    pre = (tp)/(tp+fp)
    rec = (tp)/(tp+fn)
    F1 = 2*rec*pre/(pre+rec)

    if (F1>F1_score){
      F1_score <- F1
      threshold <- alpha
      w_tuned <- w
      acc_tuned <- acc
      sens_tuned <- rec
      spec_tuned <- tn/(tn+fp)
      Y_pred_tuned <- Y_pred
} } }
```

```
#Print the results
print(paste('Threshold: ',threshold,' Weight: ',w_tuned, ' F1-score: ',F1_score, 'Accuracy: ', acc_tuned
```

```
## [1] "Threshold:  0.7  Weight:  6.5  F1-score:  0.707207207207207 Accuracy:  0.75609756097561"
```

Now, lets train the optimized model and then evaluate it in the test set:

```
weights[ind] <- w_tuned
# Estimate the parameters of the logistic regression model with the training sample
lr_hpo <- multinom(Y_train ~ LogDiabetesPedigreeFunction + LogAge + LogBMI + LogGlucose,
                   data=as.data.frame(X_train),weights = weights)
```

```
## # weights:  6 (5 variable)
## initial  value 1059.475465
## iter  10 value 566.889052
## final  value 566.889029
## converged
```

```
Y_prob_test <- predict(lr_hpo, newdata=X_test, type='probs')
Y_pred_test <- rep(0,length(Y_test))
indexs <- which(Y_prob_test > threshold)
Y_pred_test[indexs] <- 1

confusionMatrix(data=as.factor(Y_pred_test), reference = as.factor(Y_test),positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 104  23
##          1  41  61
##
##                Accuracy : 0.7205
##                  95% CI : (0.6576, 0.7776)
##     No Information Rate : 0.6332
##     P-Value [Acc > NIR] : 0.003266
```

```
##
##                   Kappa : 0.4243
##
##  Mcnemar's Test P-Value : 0.033587
##
##             Sensitivity : 0.7262
##             Specificity : 0.7172
##          Pos Pred Value : 0.5980
##          Neg Pred Value : 0.8189
##              Prevalence : 0.3668
##          Detection Rate : 0.2664
##    Detection Prevalence : 0.4454
##       Balanced Accuracy : 0.7217
##
##        'Positive' Class : 1
##
```

```
lr_num = as.numeric(lr_test) - 1
TER_lr  = 1/n_test*sum(abs(Y_test-lr_num))
TER_lr
```

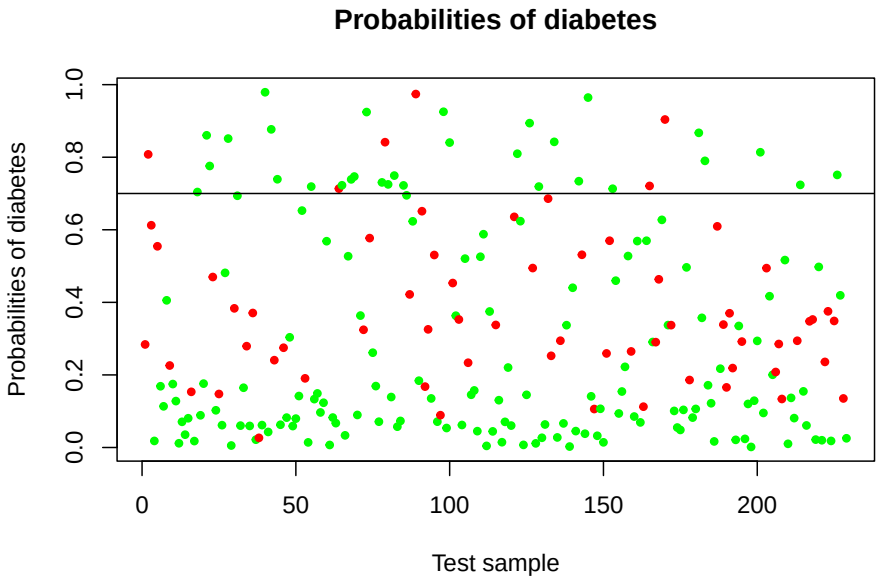
```
## [1] 0.2401747
```

After estimating the performance of the model with the test set, we can conclude that the sensitivity has grown to 0.73 while specificity has decreased to 0.72. The tuning of the F1-score implies to balance the model and both classes prediction, generally at the cost of the accuracy. In this particular case, the accuracy has decreased to 0.72, not consider a great loss in this metric.

One key point when dealing with unbalanced classes (we could say that this problem is unbalanced as one class has two times the instances as the other), is to define which of the metrics is to be maximized. Usually, missing some diabetic person is more costly than classify wrongly a non-diabetic, so the F-score can be tuned accordingly taking that difference into consideration.

Finally, we can obtain the plot in which we can see the instances well classified in green and the instances wrong classified in red. It is clear that the model is miss classifying instances in the positive category, but with the benefit of capturing almost all the instances belonging to that class.

```
# Conditional probabilities of the classifications made with the test sample
colors_errors <- c("red","green")[1*(Y_test==Y_pred_test)+1]
plot(1:n_test,prob_lr_test,col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Probabilities of diabetes",
     main="Probabilities of diabetes")
abline(h=threshold)
```


Probabilities of diabetes
Probabilities of diabetes
0.0
0.2
0.4
0.6
0.8
1.0
0
50
100
150
200
Test sample

Conclusions

After studying the data, transforming and visualizing and performing dimension reduction techniques, several supervised and unsupervised methods, we have reach to the following conclusions:

- The selected data set is slightly imbalanced so when it comes to evaluate the performance of the classification algorithms, we do not only take into account the accuracy but also a trade-off between sensitivity and specificity.
- The existence of a large amount of NA values in insulin and skin thickness make the imputation not to be so accurate and therefore they are discarded in the variable selection.
- PCA and ICA allow a good visualization while factor analysis works poorly due to the small number of factors, correlation matrix of errors shows existing correlations among them that have not been explained by the model. In addition, we would like to highlight the capacity of PC1 to split the instances in diabetics and non-diabetics group.
- In unsupervised classification, both partitional clustering methods K-means and K-medoids yields to two clusters that considerably match with outcome classes. Within hierarchical clustering, the agglomerative method with single linkage produces one cluster with only one observation resembling to the naive classifier, majority class. Divisive algorithm results in a very balanced classification with satisfactory both sensitivity and specificity. In model-based clustering the resulting configuration was ellipsoidal with equal volume and orientation but leads to poor results. Similarly, selecting a threshold of 1.68 and minimum number of points 5, DBSCAN does not perform properly. The common point of all techniques is that all tend to clusterize in two groups. The better results obtained are around 0.7 in all accuracy, sensitivity and specificity.
- In supervised classification, we can see an improvement in comparison with the unsupervised methods. Still, the imbalanced class problem present difficulties when selecting the model. In this case, all the supervised algorithms were trained with very low sensitivity, meaning that the positive class was not well classified. Therefore, it is key to tune the hyper-parameters of the model in order to get the sensitivity-specificity trade off according to the problem needs. In this case, we maximized the F1-score for the logistic regression, obtaining balanced algorithms with similar values for sensitivity and specificity.
- Finally, we could deduce that the non-diabetic people which are classified with high probabilities of having diabetes, should consider the possibility of keeping with medical consulting, as they might 'awake' the deceased in the future. This could be the reason for not getting a higher accuracy.