Team Project

José Ignacio Díez Ruiz 100487766
Carlos Roldán Piñero 100484904
Pablo Vidal Fernández 100483812

2023-01-19

Descriptive Analysis and Preprocessing

Reading the data and setting the NAs

Before starting the analysis, we need to make some preprocessing on our data. Let us start by loading it into memory and listing the names of the columns.

```
data <- read.csv("diabetes.csv")
colnames(data)
```

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

Of these, our target variable is `Outcome`, which has two levels. For convenience, we transform it into a factor variable which R can trat accordingly.

```
data$Outcome <- factor(data$Outcome, c(0, 1), c("Negative", "Positive"))
```

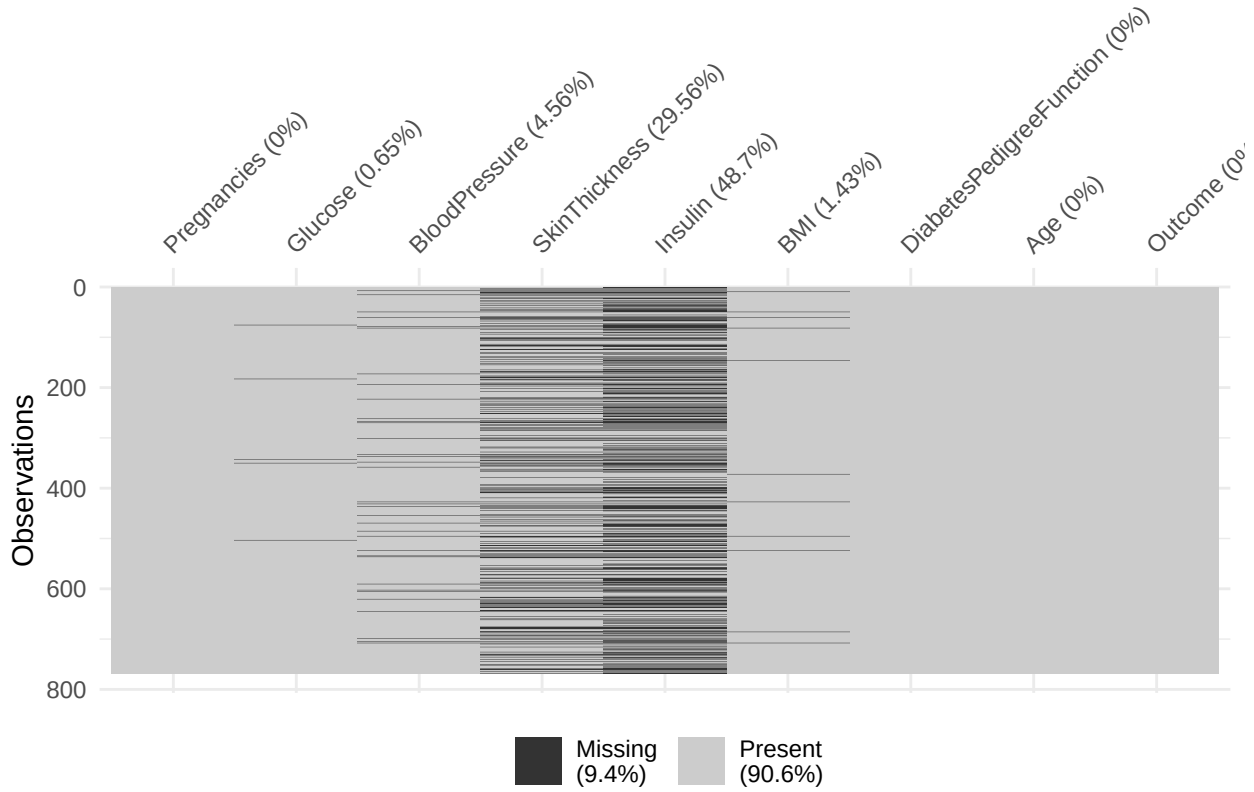
Now we need to address a particularity of the chosen data: not-a-number (NaN) instances are encoded as zeros in variables where that value is imposible [1]. These are:

- Glucose
- BloodPressure
- SkinThickness
- Insulin
- BMI

In order for us to later trat them correctly, we need to manually change them to the existing `NA` type. As we do so, we record the number of NaNs instances in each of those variables. For convenience, we define a function.

[1] Remember we are dealing with medical data, not with artificial one. Hence, there are constraints on the values a variable may take.

```r
set_nas <- function(data, fields) {
    percentage <- list()
    for (field in fields) {
        data[[field]][data[[field]] == 0]   <- NA
        percentage[[field]] <- 100 * sum(is.na(data[[field]])) / nrow(data)
    }
    return(list(data = data, percentage = percentage))
}

# Correctly label NaNs
na_fields   <- c("Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI")
data_na     <- set_nas(data, na_fields)
data        <- data_na$data
percentages <- data_na$percentage

# Visualize them
vis_miss(data)
```



Now the next logical step is to impute those NaN values. We do have some concern about the imputation of the "Insulin" variable which is almost half-filled of NaNs. Nevertheless, we decide to impute it. On the followed strategy, we use the "Predictive Mean Matching Imputation" (PMM in short) as it behaves much more robustly than naive mean or median imputations.

```r
data_im <- mice(data, 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
```

```
data    <- complete(data_im)
```

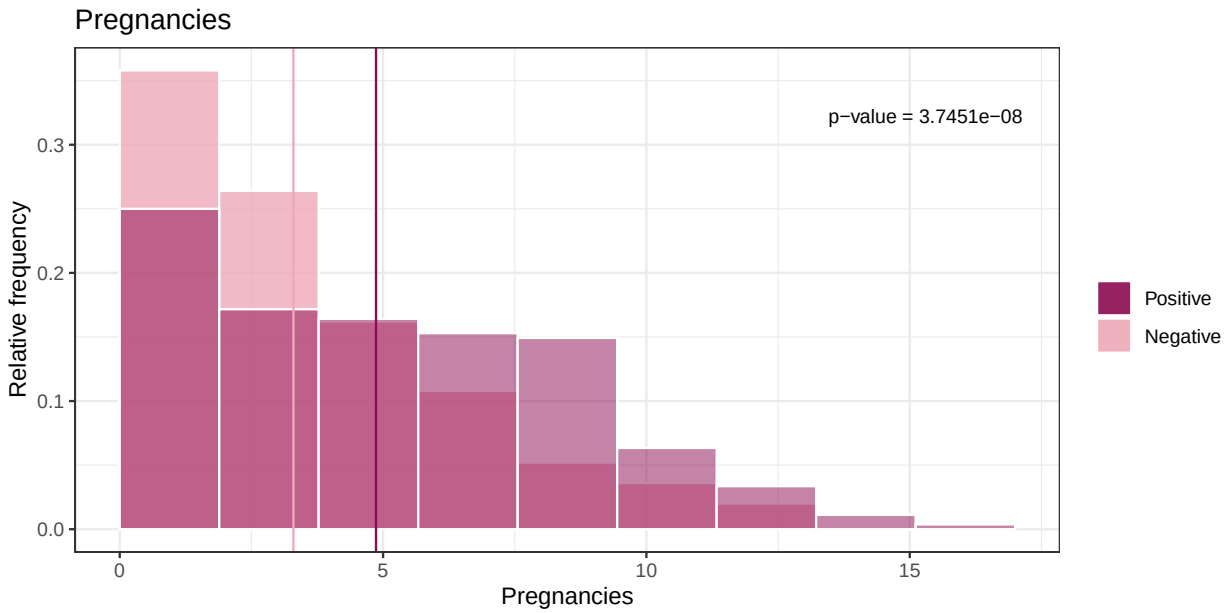
Visualization of the data

Before we proceed any further, we are going to describe our data. First, we look individually to each of the attributes, we see both visually and with with a two-sample Wilcox-test whether there is significant difference between the two groups (having or not diabetes), and if some transformation may be desirable to ensure normality compatibility. For that purpose, we define the following function.

```
histogram_by_groups <- function(data, var, label = NULL) {
    stat_t <- wilcox.test(as.formula(paste(var, "~ Outcome")), data)
    data0  <- data[data$Outcome == "Negative", ]
    data1  <- data[data$Outcome == "Positive", ]
    if (is.null(label)) {
        label <- var
    }
    p <- ggplot(data0, aes(x = eval(parse(text = var)))) +
        geom_histogram(
            aes(y = after_stat(count / sum(count)), fill = "Negative"),
            bins = 10, colour = "white", alpha = 0.8, boundary = 0
            ) +
        geom_histogram(data = data1,
            aes(
                x = eval(parse(text = var)),
                y = after_stat(count / sum(count)), fill = "Positive"
                ),
            bins = 10, colour = "white",
            alpha = 0.5, boundary = 0, inherit.aes = FALSE) +
            theme_bw() +
            scale_fill_manual(
                name = "",
                breaks = c("Positive", "Negative"),
                values = c("Positive" = "deeppink4", "Negative" = "pink2")
            ) +
            xlab(label) + ylab("Relative frequency") + ggtitle(label) +
            geom_vline(xintercept = mean(data1[[var]]), colour = "deeppink4") +
            geom_vline(xintercept = mean(data0[[var]]), colour = "pink2")
    p + annotate(
            "text",
            x = 0.9 * max(data1[var]),
            y = 0.9 * max(ggplot_build(p)$data[[1]]["y"]),
            label = sprintf("p-value = %.4e", stat_t$p.value),
            size = 3
        )
}
```

Let us start with the number of pregnancies. We can see that people who have diabetes have had more pregnancies than those who do not have diabetes. We see that it seems to be somewhat based on the p-value alone. We also note it exhibits a heavily right-skewed behaviour. As such, a logarithmic transformation would make sense to get a distribution more compatible with the normal one. Nonetheless, a problem here arises in dealing with the null values [2]. For that purpose we shift the variable by one unit. As a consistency measure, we will apply this shift to all variables we log-transform.

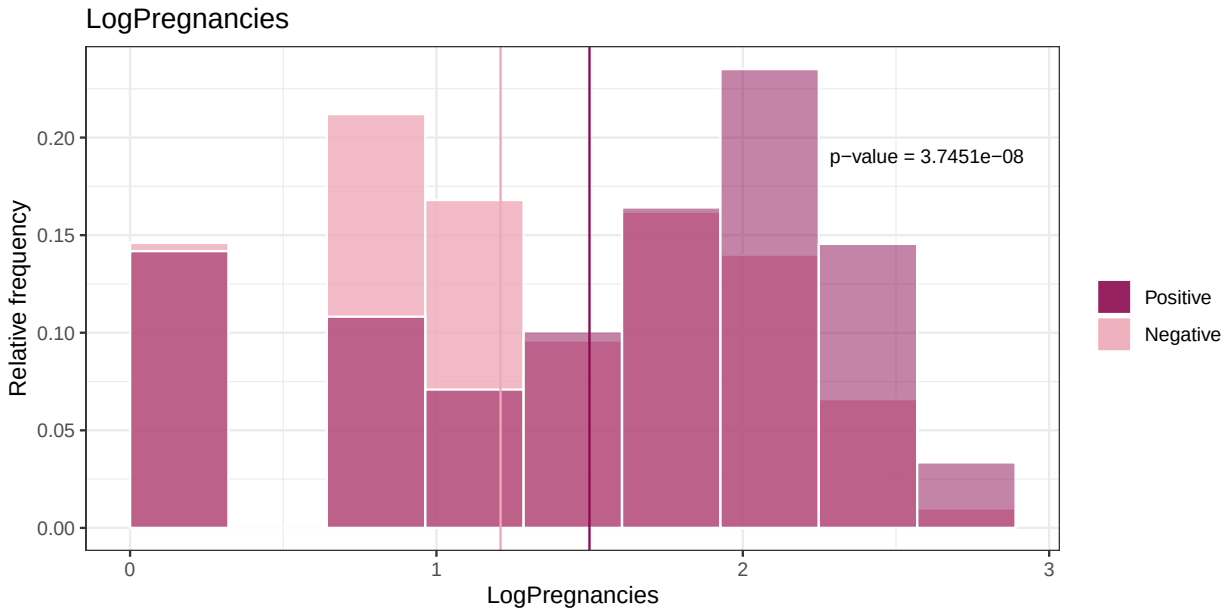
```
histogram_by_groups(data, "Pregnancies")
```



The change as we see does help in obtaining a more centered distribution with which we may better apply the posterior analysis.

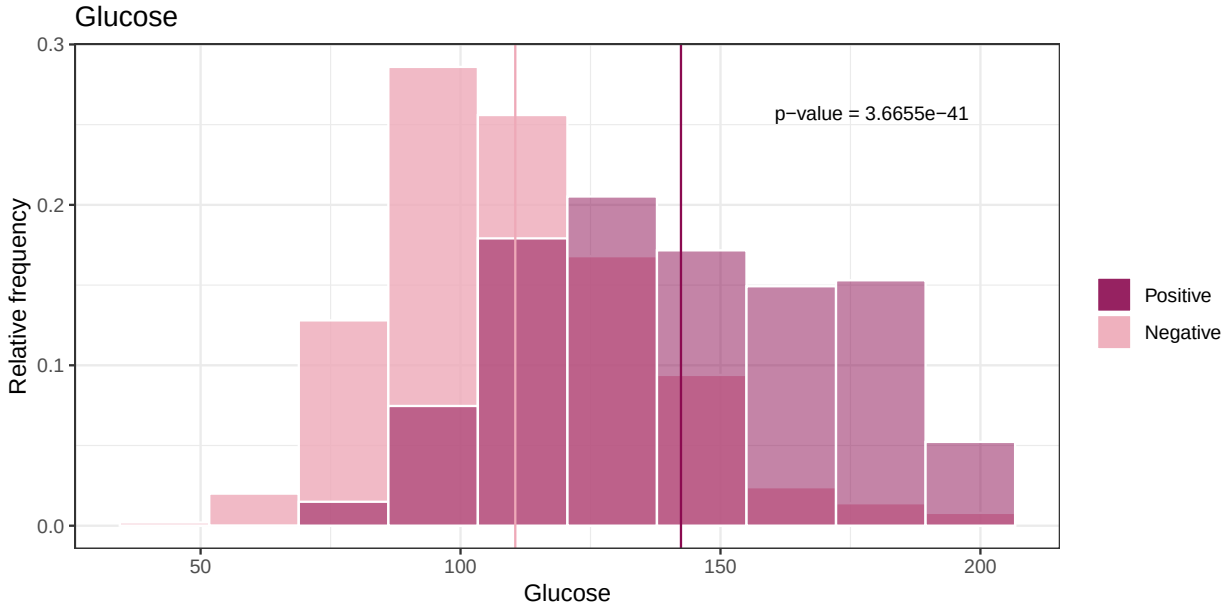
```
data$LogPregnancies <- log(data$Pregnancies + 1)
histogram_by_groups(data, "LogPregnancies")
```

[2] Remember the domain of the logarithmic function is $(0, \infty)$.



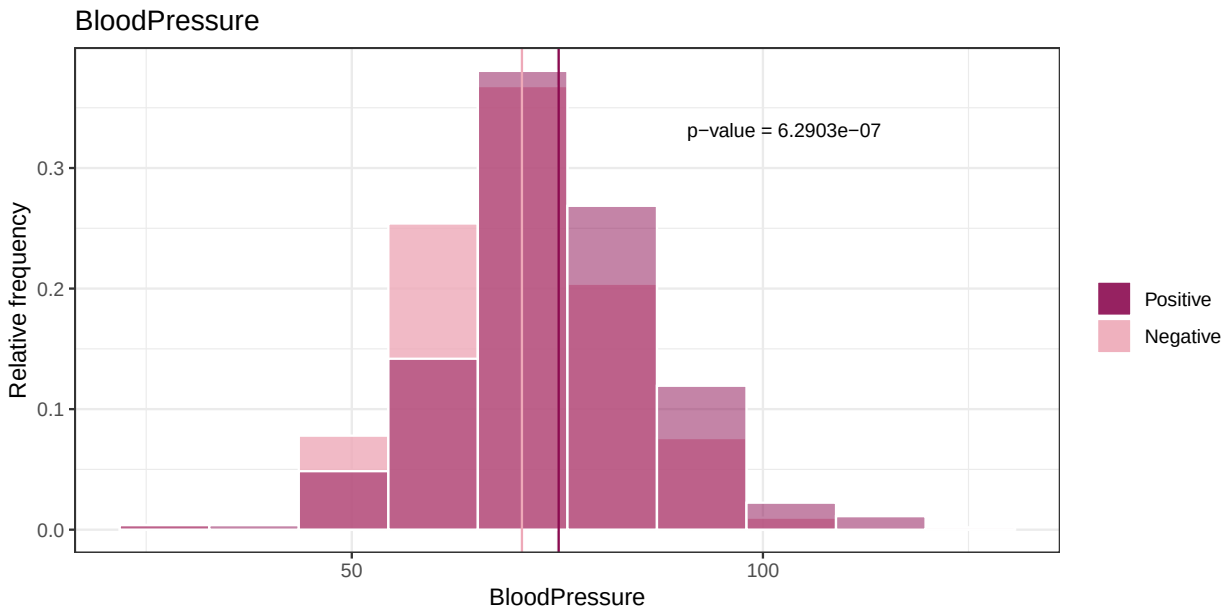
The next variable to visualize is the glucose. People with diabetes exhibit higher glucose values. The p-value is very small which indicates a highly significant difference between the two groups. We also observe that the distribution is already well-centered and resembles a normal distribution. Hence, we decide not to transform the data.

```
histogram_by_groups(data, "Glucose")
```



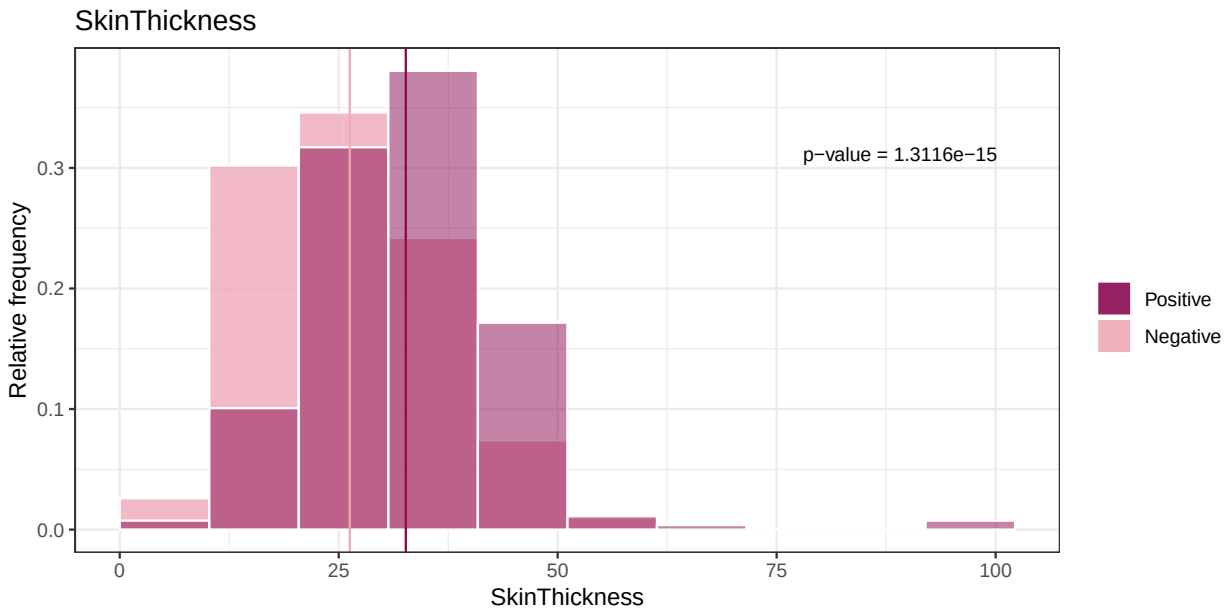
We move onwards to blood pressure. In this case, although the distributions appear as normal, there does not seem to exist a highly significant difference between the groups in contrast to what the p-value states, more so compared with the glucose variable. Nonetheless, there seem to be an slight indication of higher blood pressure for people with diabetes.

```
histogram_by_groups(data, "BloodPressure")
```



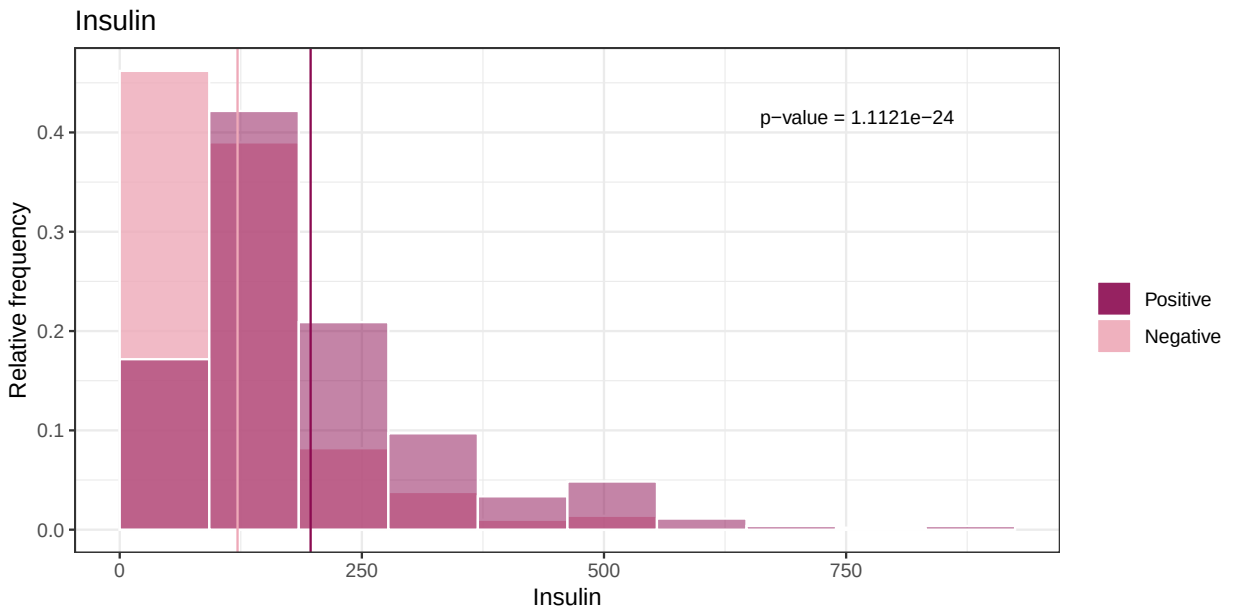
Now skin thickness is interesting, because the distributions are visually to those of the blood pressure but the presence of outliers is appreciable. We will later deal with those but for now let us mantain this variable as it is. Note that the population with diabetes appear to exhibit a thicker skin.

```
histogram_by_groups(data, "SkinThickness")
```



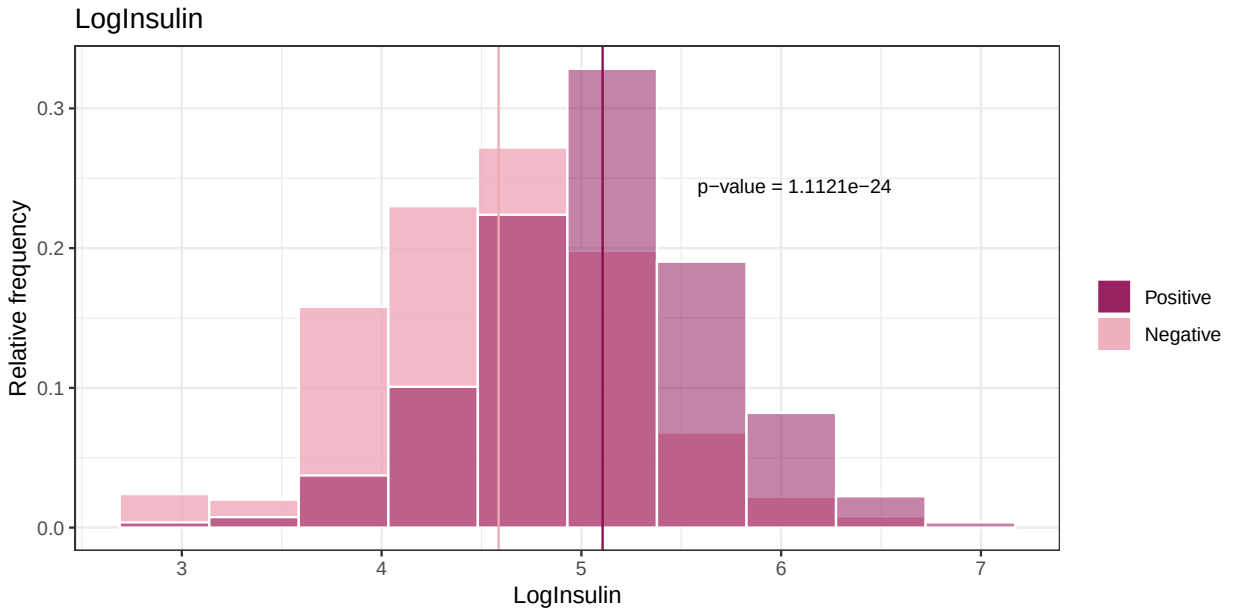
Insulin, as we may have expected from the name of the property itself, appears to be a relevant. The median of the distributions does indeed seem to differ, with the one for the diabetes population being slightly higher. It is also right-skewed, so we decide to log-transform it.

```
histogram_by_groups(data, "Insulin")
```



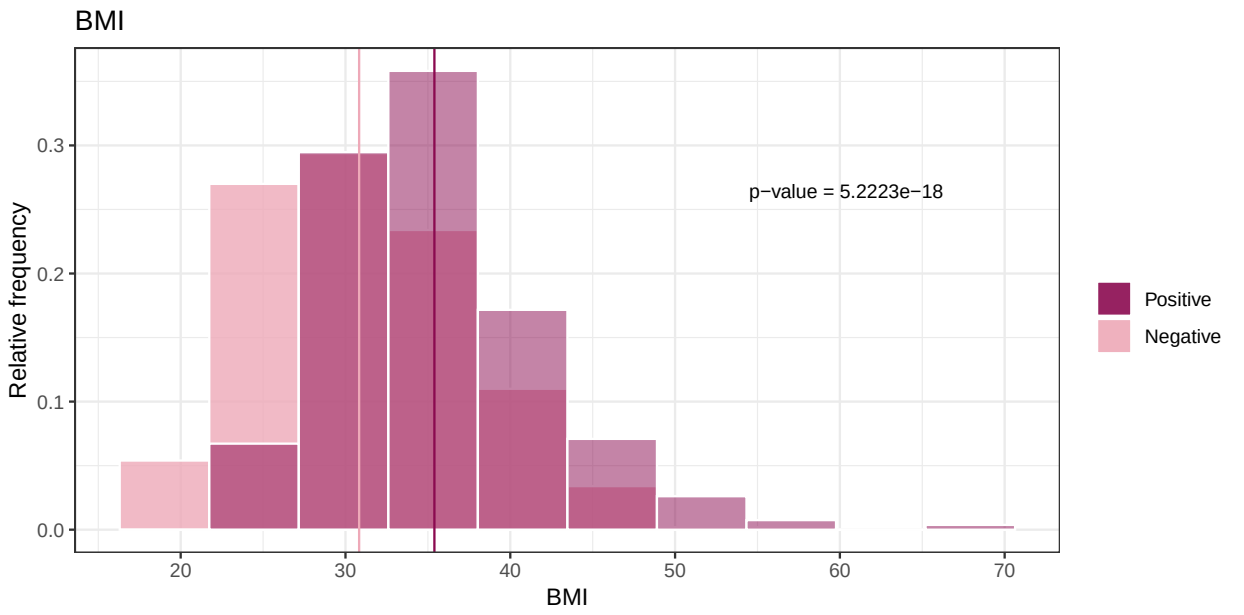
It does appear that the transformation impproves the the symmetrization of the data, although some left-skewness appears. We will later see if outlier detection get to target those values or not.

```
data$LogInsulin <- log(data$Insulin + 1)
histogram_by_groups(data, "LogInsulin")
```



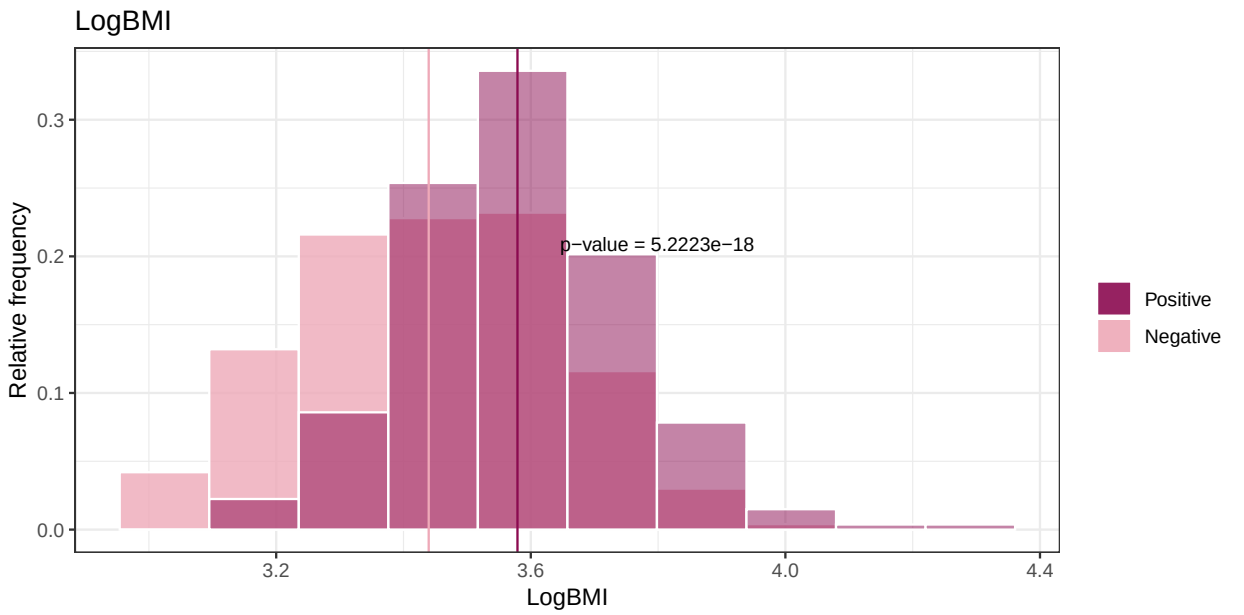
Body Mass Index (BMI) again exhibits this tendency of leaning towards a more right-skewed distribution. It does also follow the tendency of being slightly higher for people with diabetes. As such we log-transform to try and get a more normalized variable.

```
histogram_by_groups(data, "BMI")
```



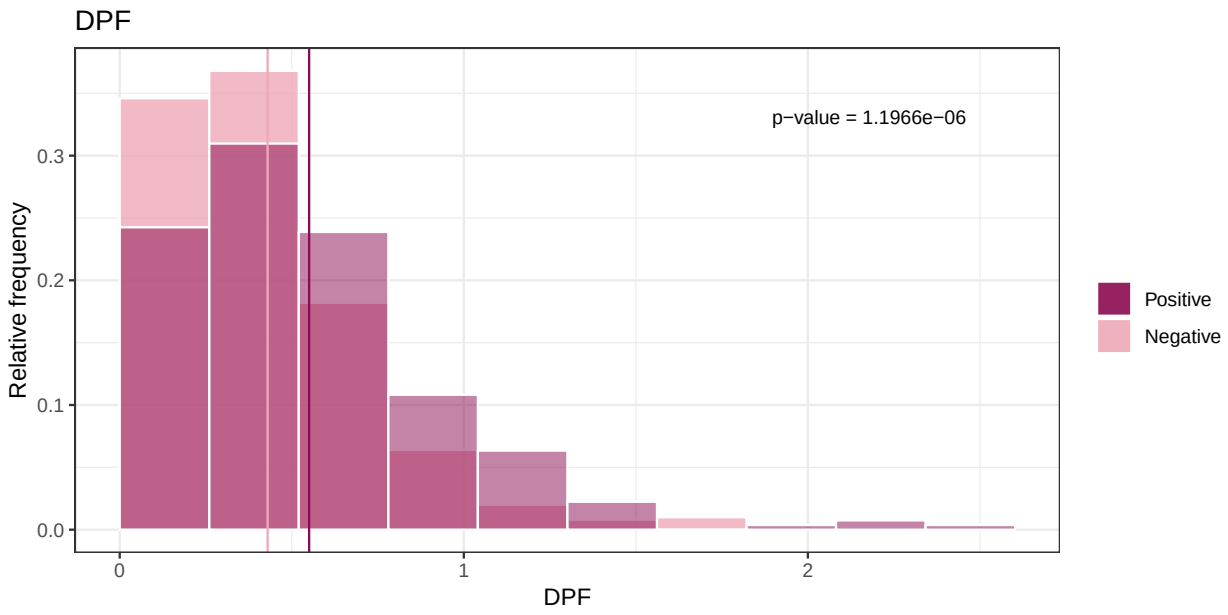
As we may visually judge, it is the case that the log transformation centers the data and provides a more normal-like distribution. There is some presence of seemingly outlying points to the right.

```
data$LogBMI <- log(data$BMI + 1)
histogram_by_groups(data, "LogBMI")
```



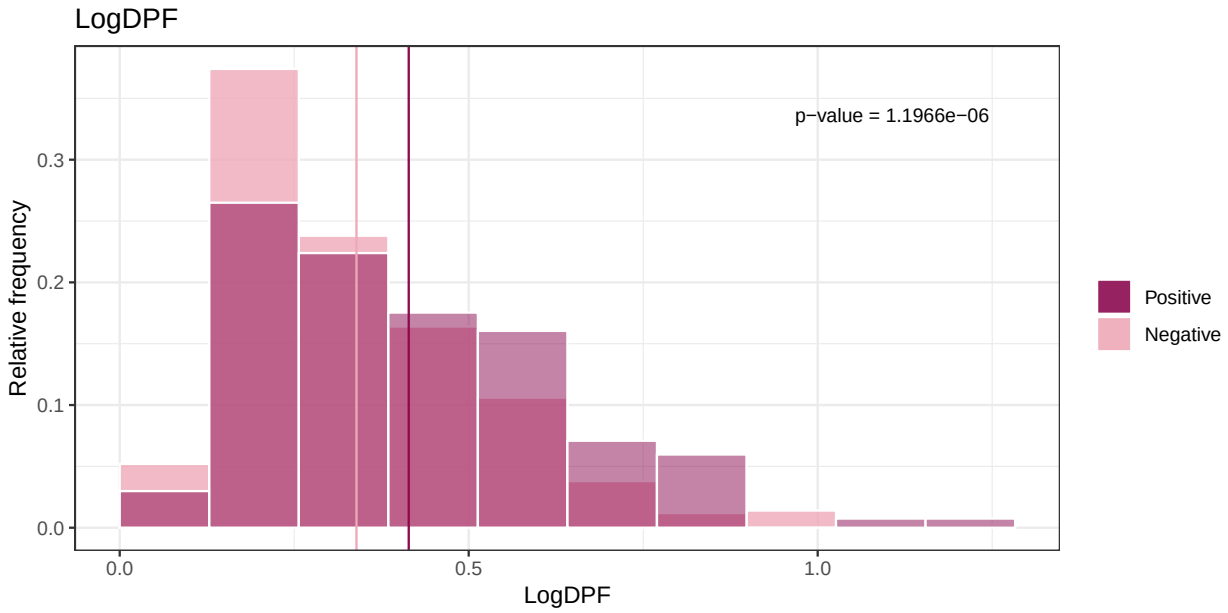
The Diabetes Pedigree Function (DPF) is the again a flagrant right-skewed. It again has higher values for the positive set. This is to be expected from the definition of this very function as a risk indication for diabetes. Let us try to log-transform it.

```
histogram_by_groups(data, "DiabetesPedigreeFunction", "DPF")
```



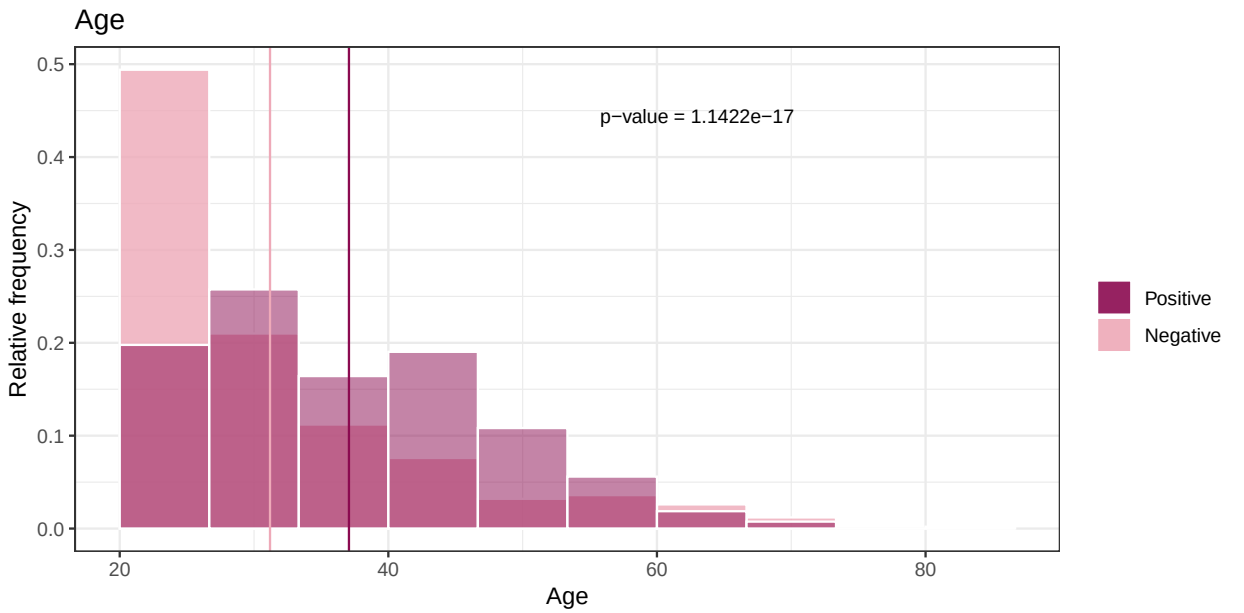
The improvement is highly noticeable. We retain thus this transformed variable.

```
data$LogDPF <- log(data$DiabetesPedigreeFunction + 1)
histogram_by_groups(data, "LogDPF")
```



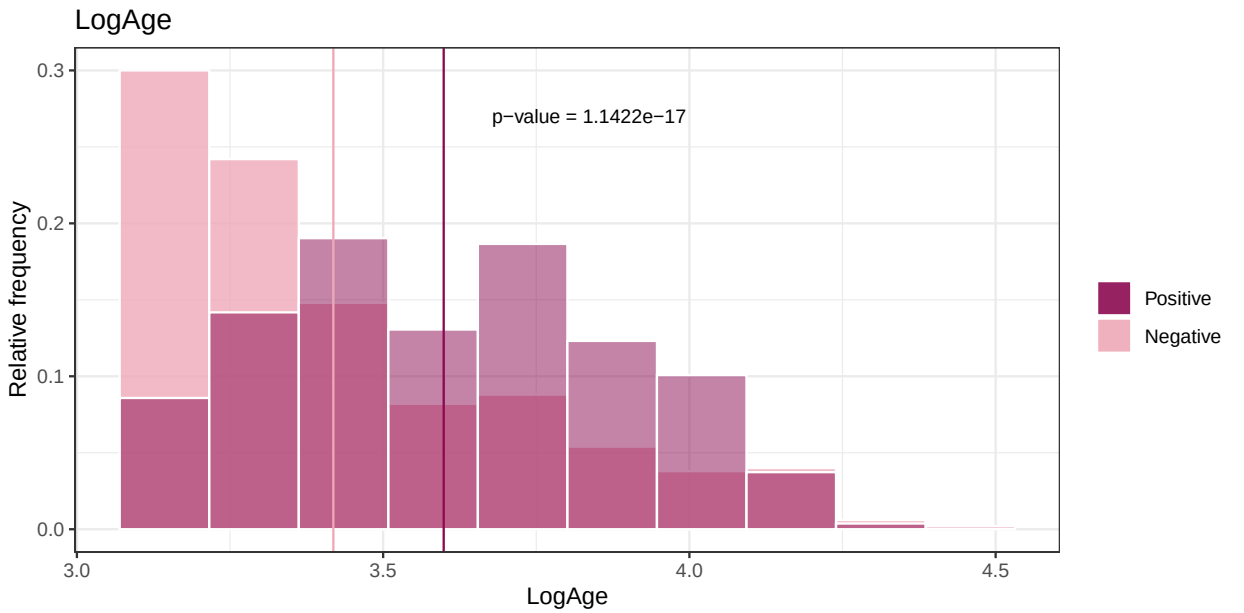
We may note that young people, as with other illnesses have less tendency to suffer diabetes than the elders. The age is expected to exhibit a right-skewed distribution, as is indeed the case. In an attempt to improve the symmetry, we once again use logarithms to transform the variable.

```
histogram_by_groups(data, "Age")
```



The transformation does help although not by much. This is a common problem of the age variable. We keep the transformation anyway as it does seem to help with the symmetry for the positive group.

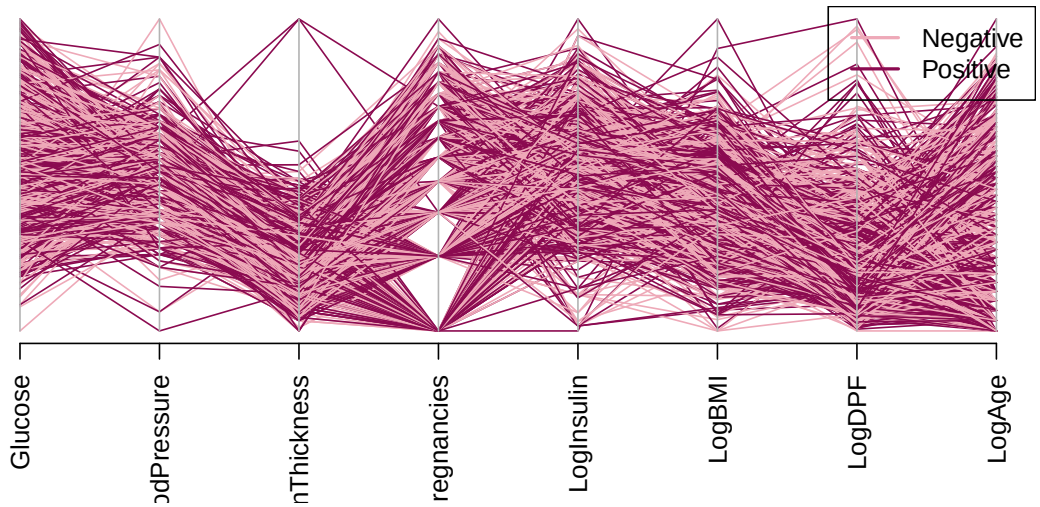
```
data$LogAge <- log(data$Age + 1)
histogram_by_groups(data, "LogAge")
```




```
# Some convenience
df  <- subset(data, select = -c(Pregnancies, Insulin, BMI, DiabetesPedigreeFunction, Age))
df0     <-  df[df$Outcome == "Negative", ]
df1     <-  df[df$Outcome == "Positive", ]
xnames  <- names(df)[! names(df) %in% c("Outcome")]
```

Now, let's take a look at some multivariate plots. We'll begin by inspecting the Parallel Coordinate Plot:

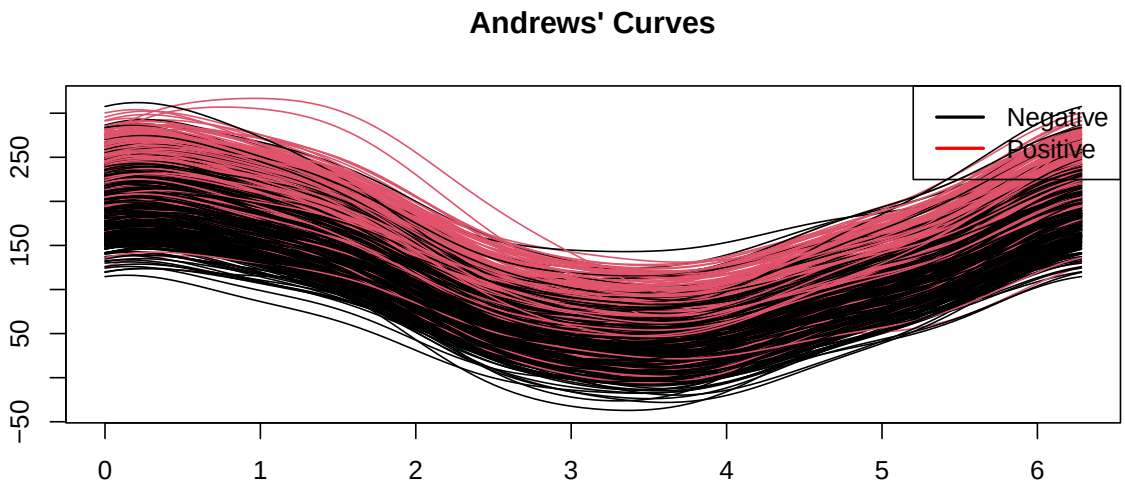
```
par(las = 2)
parcoord(df[xnames], col = c("pink2", "deeppink4"))
legend("topright", legend = c("Negative", "Positive"),
       col = c("pink2", "deeppink4"), lty = 1, lwd = 2)
```



It seems that, overall, the positive lines are over negative ones. This is most notable on the Glucose and log transformed BMI. They are two of the most significant features according to the p-value.

The Andrew's plot is the following:

```
andrewsplot(as.matrix(df[xnames]), df$Outcome, style = "cart")
legend("topright", legend = c("Negative", "Positive"),
       col = c("black", "red"), lty = 1, lwd = 2)
```



Again, we see that the two groups are different. The group of people who have diabetes tend to have more volatile curves, reaching higher and lower values along the curve.

Multivariate characteristics and outlier identification

We are goig to use a multivariate approach to identifying the outliers in our data. In the process, we will need to compute the mean and covariance.

We start then by finding the mean vector and the covariance matrix. In order to reduce the sensitivity to outliers in this computation, we use a robust estimation using the "Fast MCD" (Minimum Covariance Determinant) estimator. We set main parameter, `alpha`, which determines the percentage of the data to use, at 0.85.

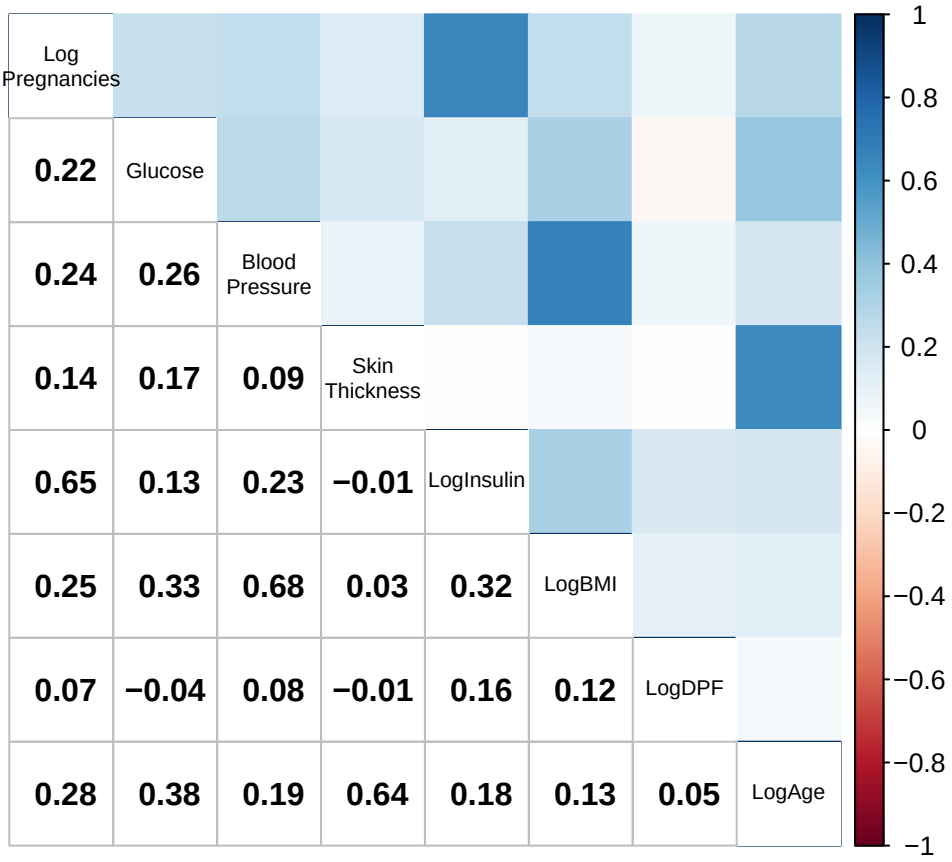
```
our_corrplot <- function(cov_mat) {
    colnames(cov_mat) <- c("Log\nPregnancies",
                           "Glucose",
                           "Blood\nPressure",
                           "Skin\nThickness",
                           "LogInsulin",
                           "LogBMI",
                           "LogDPF",
                           "LogAge")
    corrplot.mixed(cov2cor(cov_mat), lower = "number", upper = "color",
        diag = "n", tl.col = "black", tl.cex = 0.65,
        lower.col = "black")
}

mcd_est <- CovMcd(df[xnames], alpha = 0.85, nsamp = "deterministic")
```

```
## [1] "The mean vector is:"
```

```
##         Glucose   BloodPressure   SkinThickness LogPregnancies      LogInsulin
##     120.1496503      71.8447552      28.2419580       1.3436900       4.7547273
```

```
##         LogBMI         LogDPF         LogAge
##      3.4848680      0.3503205      3.4662863

## [1] "The covariance matrix is:"

##                 Glucose BloodPressure SkinThickness LogPregnancies LogInsulin
## Glucose        950.4770       82.6318       76.4278         3.5080    14.1659
## BloodPressure   82.6318      145.3171       32.5017         1.6227     1.1007
## SkinThickness   76.4278       32.5017      105.9615         0.7482     1.6907
## LogPregnancies   3.5080        1.6227        0.7482         0.6205    -0.0071
## LogInsulin      14.1659        1.1007        1.6907        -0.0071     0.4954
## LogBMI           1.5547        0.8148        1.4366         0.0050     0.0464
## LogDPF           0.3945       -0.0852        0.1478        -0.0011     0.0215
## LogAge           2.7553        1.4676        0.6184         0.1606     0.0416
##                 LogBMI  LogDPF LogAge
## Glucose         1.5547  0.3945 2.7553
## BloodPressure   0.8148 -0.0852 1.4676
## SkinThickness   1.4366  0.1478 0.6184
## LogPregnancies 0.0050 -0.0011 0.1606
## LogInsulin      0.0464  0.0215 0.0416
## LogBMI          0.0422  0.0045 0.0085
## LogDPF          0.0045  0.0354 0.0030
## LogAge          0.0085  0.0030 0.1019
```

```
our_corrplot(mcd_est$cov)
```



It is interesting to seggregate by the class of the Outcome variable. This way, we can both, get the mean vector and covariance matrix for each of the classes, and the outliers for both sets.

```
mcd_neg <- CovMcd(df0[xnames], alpha = 0.85, nsamp = "deterministic")
mcd_pos <- CovMcd(df1[xnames], alpha = 0.85, nsamp = "deterministic")
```

```
## [1] "### Negative class ###"
```

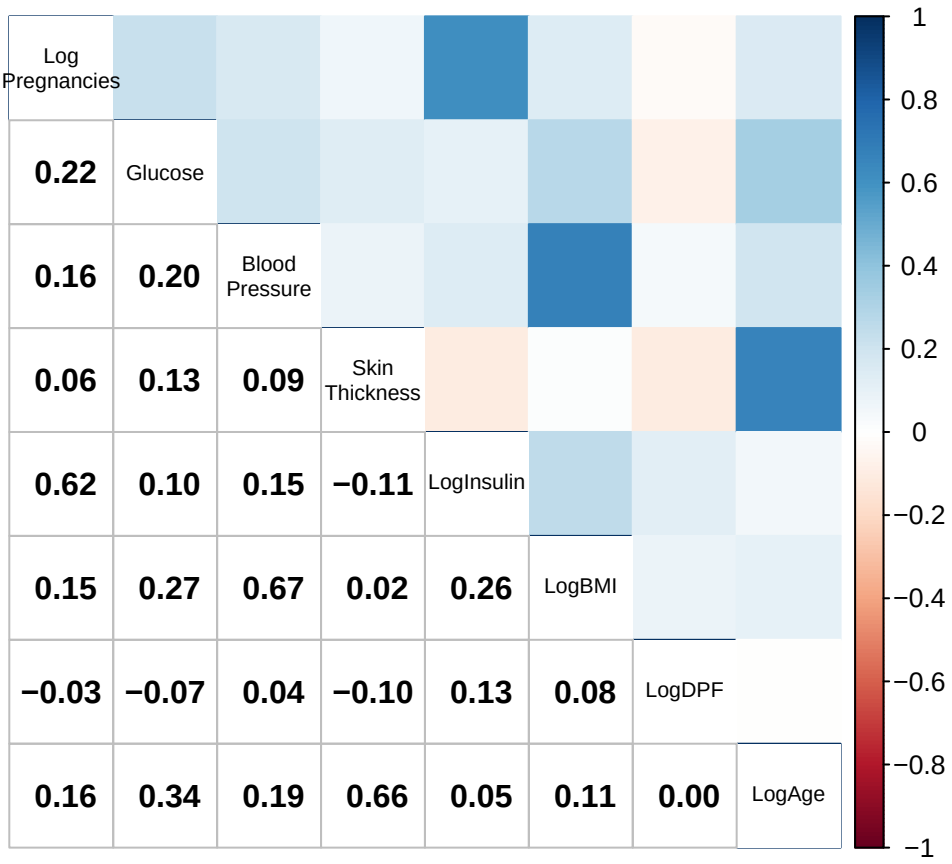
```
## [1] "The mean vector is:"
```

```
##        Glucose  BloodPressure  SkinThickness LogPregnancies     LogInsulin
##    108.3443709     69.9735099     26.2052980      1.2157181      4.5500091
##         LogBMI         LogDPF         LogAge
##      3.4390641      0.3221247      3.3850307
```

```
## [1] "The covariance matrix is:"
```

```
##                 Glucose BloodPressure SkinThickness LogPregnancies LogInsulin
## Glucose        551.4092       61.0765       39.9690         1.1180     9.7425
## BloodPressure   61.0765      135.6526       24.6780         1.1780     0.7855
## SkinThickness   39.9690       24.6780      107.3314         0.6773     1.0128
## LogPregnancies   1.1180        1.1780        0.6773         0.5716    -0.0553
## LogInsulin       9.7425        0.7855        1.0128        -0.0553     0.4509
## LogBMI           0.7122        0.6555        1.4554         0.0025     0.0358
## LogDPF          -0.1059       -0.1477        0.0782        -0.0130     0.0143
## LogAge           1.1126        1.1650        0.5893         0.1493     0.0102
##                LogBMI  LogDPF  LogAge
## Glucose        0.7122 -0.1059  1.1126
## BloodPressure  0.6555 -0.1477  1.1650
## SkinThickness  1.4554  0.0782  0.5893
## LogPregnancies 0.0025 -0.0130  0.1493
## LogInsulin     0.0358  0.0143  0.0102
## LogBMI         0.0434  0.0030  0.0068
## LogDPF         0.0030  0.0287 -0.0001
## LogAge         0.0068 -0.0001  0.0887
```

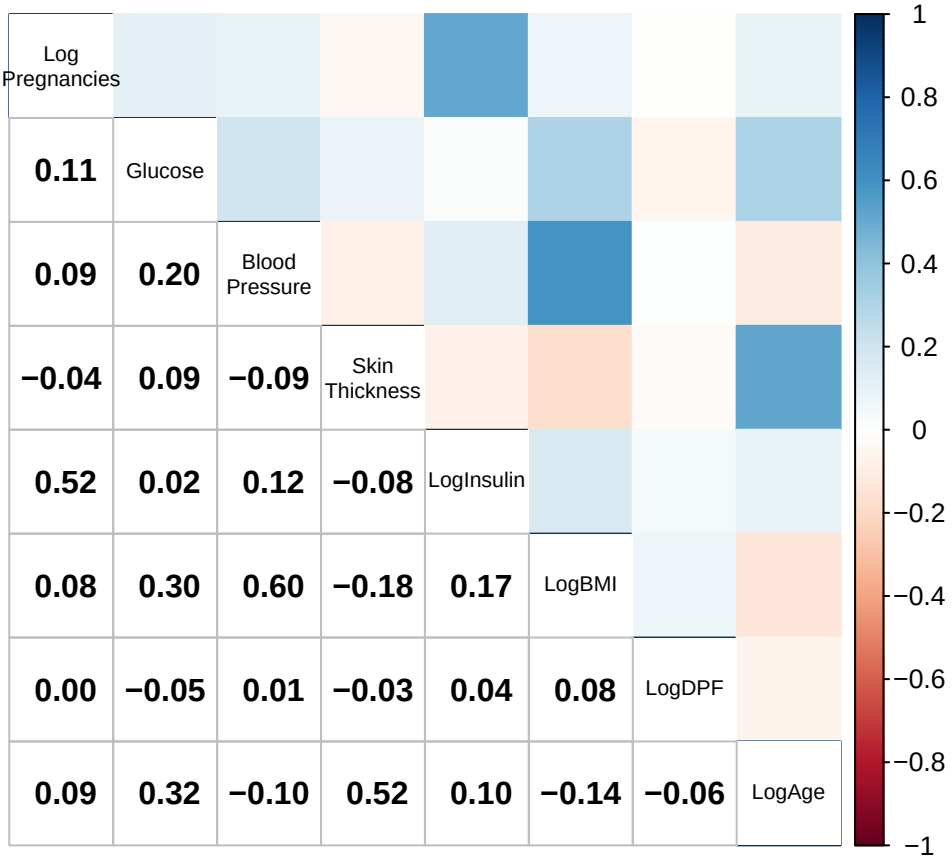
```
our_corrplot(mcd_neg$cov)
```




```
## [1] "### Positive class ###"

## [1] "The mean vector is:"

##        Glucose  BloodPressure  SkinThickness LogPregnancies     LogInsulin
##    142.1953125     75.0664062     31.8789062      1.5249309      5.1218071
##         LogBMI         LogDPF         LogAge
##      3.5730027      0.4042787      3.5961270

## [1] "The covariance matrix is:"

##                 Glucose BloodPressure SkinThickness LogPregnancies LogInsulin
## Glucose        935.9644       41.7494       26.6141        -1.1404     9.4190
## BloodPressure   41.7494      142.4736       22.5587         0.8700     0.1370
## SkinThickness   26.6141       22.5587       92.0943        -0.7162     0.6969
## LogPregnancies  -1.1404        0.8700       -0.7162         0.6922    -0.0389
## LogInsulin       9.4190        0.1370        0.6969        -0.0389     0.3567
## LogBMI           0.4144        0.6173        0.9839        -0.0252     0.0173
## LogDPF          -0.0296       -0.1248        0.0140        -0.0050     0.0054
## LogAge           0.8406        1.1191       -0.2837         0.1275     0.0170
##                 LogBMI  LogDPF  LogAge
## Glucose         0.4144 -0.0296  0.8406
## BloodPressure   0.6173 -0.1248  1.1191
## SkinThickness   0.9839  0.0140 -0.2837
## LogPregnancies -0.0252 -0.0050  0.1275
```

```
## LogInsulin      0.0173  0.0054  0.0170
## LogBMI          0.0294  0.0028 -0.0070
## LogDPF          0.0028  0.0432 -0.0038
## LogAge         -0.0070 -0.0038  0.0862
```

```
our_corrplot(mcd_pos$cov)
```



Let us focus first on the mean vectors. The here observed particularities are nothing new, as they were already present on above histograms:

- Higher number of pregnancies seems to increase the chances on developing diabetes.
- The glucose and insulin levels of the positive population is higher in contrast to the negative one.

Looking now at the representations for the covariance matrices, the major changes are that the correlation between skin thickness and diabetes pedigree function is lower in the group who do not have diabetes than in the one having the disease. Moreover, the correlation between BMI [3] and age is positive for the sane group while negative on the positive one.

We now search for outliers. The idea is to use Mahalanobis distance. As we suppose that our data $X \sim \mathcal{N}(\mu, \Sigma)$, we have $D_M(x,\mu)^2 \sim \chi^2_p$, with D_M the Mahalanobis distance. Hence we may set our outlier criteria as

$$D_M(x,\mu)^2 > \chi^2_{p,0.95^{1/n}} \,, \tag{1}$$

the 0.95^{th} quantile of the χ^2_p distribution. We then drop these outliers.

[3] Remember this is the logarithm.

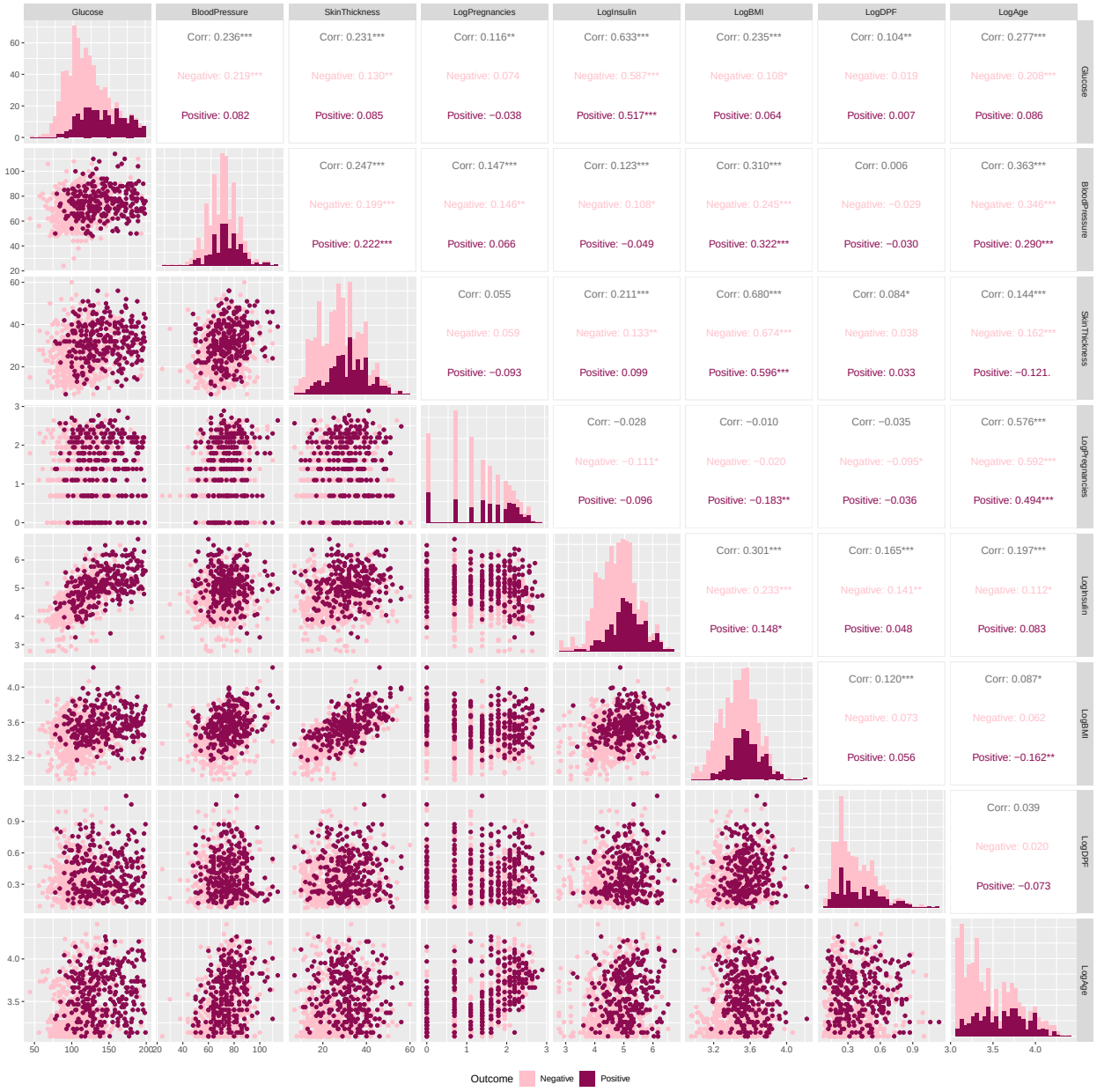
```
p   <- length(xnames)
n0  <- nrow(df0)
n1  <- nrow(df1)
df0_clean <- df0[mcd_neg$mah < qchisq(0.95^(1 / n0), p), ]
df1_clean <- df1[mcd_pos$mah < qchisq(0.95^(1 / n1), p), ]
df_clean  <- rbind(df0_clean, df1_clean)
```

```
## [1] "The negative set contained 6 outliers."
```

```
## [1] "The positive set contained 5 outliers."
```

As a final summary of this section, we perform a plot in which the histograms of different populations are observed, as well as scatterplots of pairs of variables and correlations.

```
ggpairs(df_clean, aes(color = Outcome), legend = 1,
        columns = xnames,
        diag = list(continuous = "barDiag")
        ) +
    theme(legend.position = "bottom") +
    scale_fill_manual(values = c("pink", "deeppink4")) +
    scale_color_manual(values = c("pink", "deeppink4")) + labs(fill = "Outcome")
```



Supervised Classification

```
color_1 <- "deepskyblue2"
color_2 <- "darkorchid4"
color_3 <- "seagreen2"
color_4 <- "indianred2"
```

We split the data into the predictors (X) and the variable we want to predict (Y):

```r
X <- df_clean[,xnames]
Y <- df_clean$Outcome
```

```r
n <- nrow(X)
p <- ncol(X)
c(n,p)
```

```
## [1] 757   8
```

```r
n_no <- sum(Y=="Negative")
n_yes <- sum(Y=="Positive")
c(n_no,n_yes)
```

```
## [1] 494 263
```

```r
pr_no <- n_no/n
pr_yes <- n_yes/n
c(pr_no,pr_yes)
```

```
## [1] 0.652576 0.347424
```

To create the training and test partitions we will make a 70/30 partition, that is, 70% of the individuals will go to the training partition and the remaining 30% to the test partition.

In order to do that, first we are going to compute the number of individuals of each partition:

```r
n_train <- floor(.7*n) ## 70/30 partition for train and test
n_test <- n - n_train
c(n_train,n_test)
```

```
## [1] 529 228
```

With that computed, we can generate the index of the individuals that are going to belong to the training partition:

```r
i_train <- sort(sample(1:n,n_train))
```

The individuals that are going to belong to the testing partition will be the ones that do not belong to the training partition, thus we can generate the training and testing partitions by:

```r
X_train <- X[i_train,]
X_test <- X[-i_train,]
Y_train <- Y[i_train]
Y_test <- Y[-i_train]
```

It may be interesting to check the proportion of individuals of each class in both the train and the test partitions.

	Classified_as_negative	Classified_as_positive
Instances_actually_negative	True negative (TN)	False positive (FP)
Instances_actually_positive	False negative (FN)	True positive (TP)

```
np_train <- sum(Y_train=="Negative")/n_train; np_train
```

```
## [1] 0.6616257
```

```
pp_train <- sum(Y_train=="Positive")/n_train; pp_train
```

```
## [1] 0.3383743
```

```
np_test <- sum(Y_test=="Negative")/n_test; np_test
```

```
## [1] 0.6315789
```

```
pp_test <- sum(Y_test=="Positive")/n_test; pp_test
```

```
## [1] 0.3684211
```

As the problem is unbalanced, we want to define some key concepts before starting with the methods.

The confusion matrix is defined as:

With this in mind, we can also define:

- $TPR = TP/(TP + FN)$ (True Positive Rate i.e. Sensitivity)
- $FNR = FN/(TP + FN)$ (False Negative Rate)
- $FPR = FP/(FP + TN)$ (False Positive Rate)
- $TNR = TN/(FP + TN)$ (True Negative Rate i.e Specificity)
- Accuracy $= (TP + TN)/(TP + TN + FP + FN)$
- TER $= (FP + FN)/(TP + TN + FP + FN) = 1 -$ Accuracy (Text Error Rate)
- $BAC = (TPR + TNR)/2$ (Balanced Accuracy)

K-Nearest Neighbors (KNN)

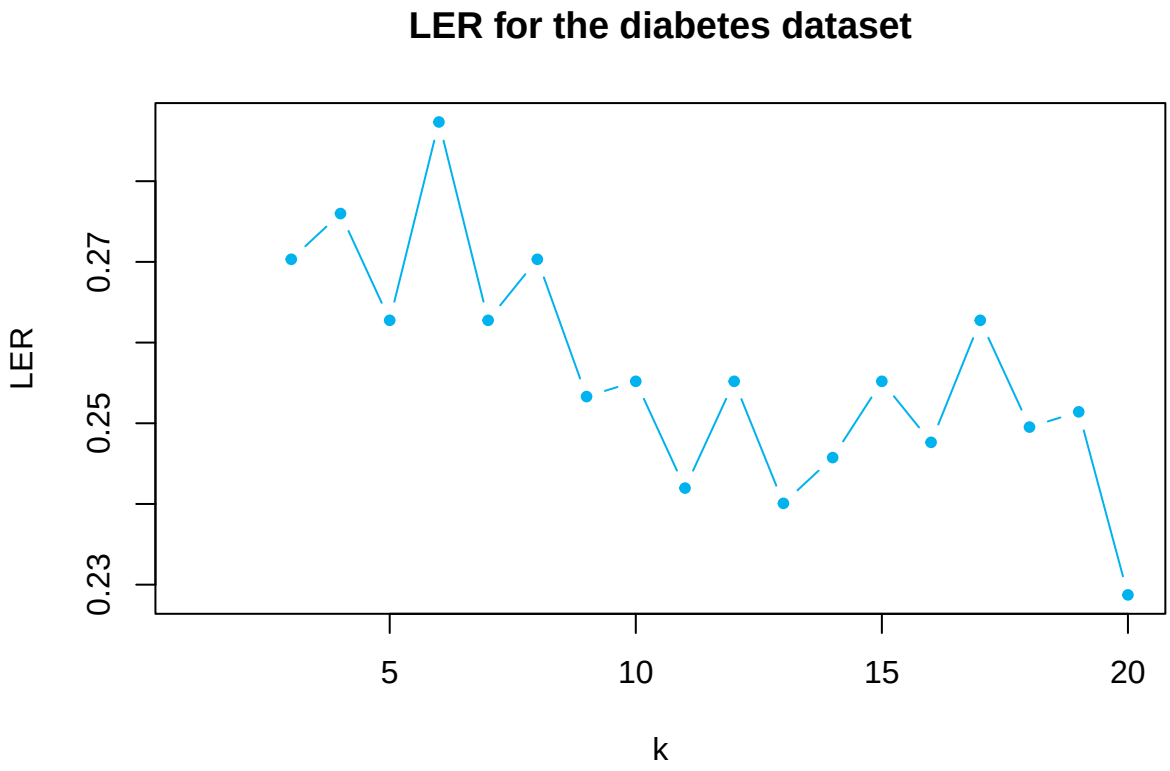
The first supervised classification method that we are going to test is K-Nearest Neighbors.

First, we are going to find the optimum value for K value using cross-validation.

```
LER <- rep(NA,20)
for (i in 3 : 20){
  knn_output <- knn.cv(X_train,Y_train,k=i)
  LER[i] <- 1 - mean(knn_output==Y_train)
}
LER
```

```
##  [1]        NA        NA 0.2703214 0.2759924 0.2627599 0.2873346 0.2627599
##  [8] 0.2703214 0.2533081 0.2551985 0.2419660 0.2551985 0.2400756 0.2457467
## [15] 0.2551985 0.2476371 0.2627599 0.2495274 0.2514178 0.2287335
```

```
plot(1:20,LER,pch=20,col=color_1,type="b",
     xlab="k",ylab="LER",main="LER for the diabetes dataset")
```



It seems that the optimal K is

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

```
## [1] 20
```

Now, with the K selected, we can now compute the predictions for the testing partition.

```
knn_Y_test <- knn(X_train,X_test,Y_train,k=k,prob=T)
```

With the predictions computed, we can show the confusion matrix:

```
cm_knn <- confusionMatrix(table(Y_test,knn_Y_test), positive = "Positive")

cm_knn$table %>% kable() %>% kable_styling(latex_options = "striped")
```

Looking at some of the metrics defined earlier, we can see that the results are not great.

	Negative	Positive
Negative	119	25
Positive	42	42

```
cm_knn$overall["Accuracy"]
```

```
##  Accuracy
## 0.7061404
```

```
cm_knn$byClass["Sensitivity"]
```

```
## Sensitivity
##   0.6268657
```

```
cm_knn$byClass["Specificity"]
```

```
## Specificity
##   0.7391304
```

We compute the Test Error Rate by:

```
knn_TER <- mean(Y_test!=knn_Y_test)
knn_TER
```

```
## [1] 0.2938596
```

Given a confusion matrix, we can also compute the BAC. First, we define the following function:

```
bac <- function(cm){
  return((cm[1,1]/(cm[1,1] + cm[1,2]) + (cm[2,2]/(cm[2,1] + cm[2,2])))/2)
}
```

And we compute the BAC:

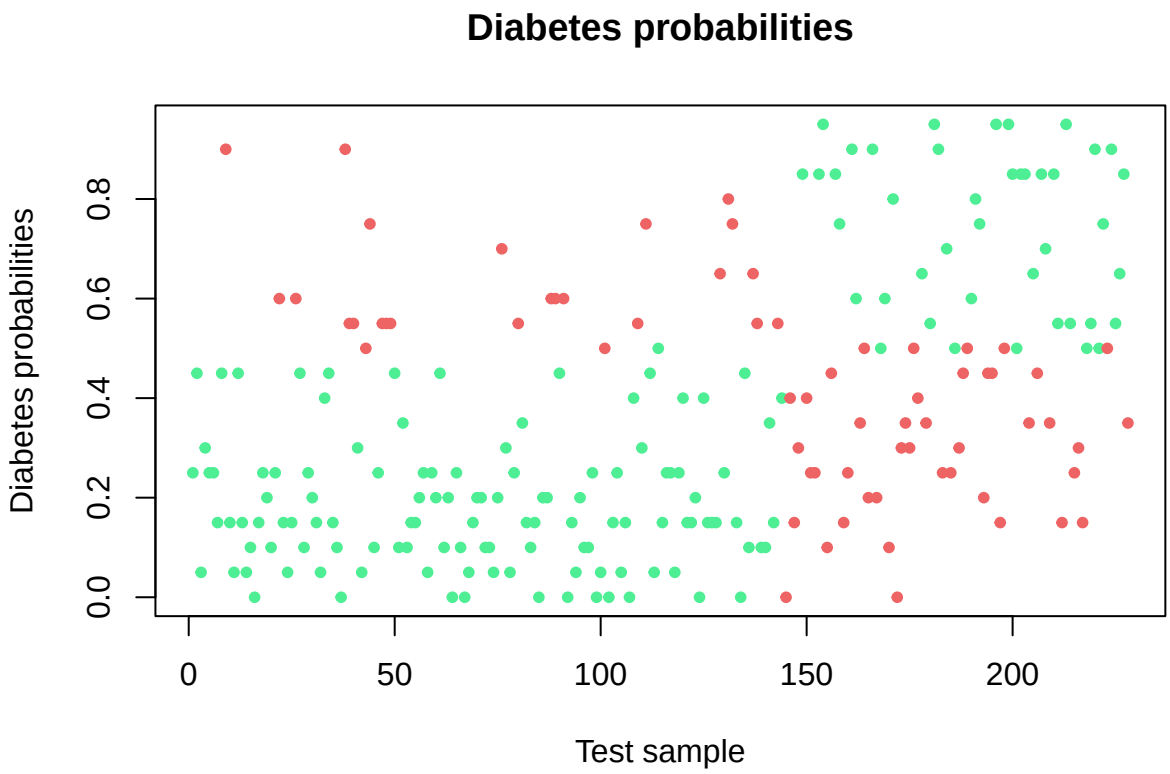
```
bac(cm_knn$table)
```

```
## [1] 0.6631944
```

It is noticeably lower than the accuracy.

Finally, we are going to plot the probability of being positive on diabetes for each instance. The ones in green are correctly classified, while those in red have been classified incorrectly.

```
prob_knn_Y_test <- ifelse(knn_Y_test == "Positive", attributes(knn_Y_test)$prob, 1 - attributes(knn_Y_t
prob <- 2*ifelse(knn_Y_test == "-1", prob_knn_Y_test, 1-prob_knn_Y_test) - 1
colors_errors <- c(color_4,color_3)[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="Diabetes probabilities",main="Diabetes probabilities")
```



Logistic Regression

Next, we are going to test is logistic regression. We start by training the model and checking the summary:

```
lr_train <- multinom(Y_train~.,data=X_train)
```

```
## # weights:  10 (9 variable)
## initial  value 366.674859
## iter  10 value 246.148910
## final  value 232.645473
## converged
```

```
summary(lr_train)
```

```
## Call:
## multinom(formula = Y_train ~ ., data = X_train)
##
## Coefficients:
##                       Values   Std. Err.
## (Intercept)    -21.644038019 3.249891768
## Glucose          0.036669722 0.005433883
## BloodPressure    0.002365727 0.011326409
## SkinThickness    0.006328879 0.014969685
## LogPregnancies   0.389072277 0.188201673
```

	x
Negative	161
Positive	67

	Negative	Positive
Negative	125	19
Positive	36	48

```
## LogInsulin       0.204131478 0.240032810
## LogBMI           2.966770120 0.866709567
## LogDPF           1.665437122 0.619367964
## LogAge           0.986695067 0.473299130
##
## Residual Deviance: 465.2909
## AIC: 483.2909
```

In order to see which are the most significant coefficients, we can use the t-test:

```
t_test_lr_train <- summary(lr_train)$coefficients/summary(lr_train)$standard.errors
sort(abs(t_test_lr_train),decreasing=TRUE)
```

```
##        Glucose    (Intercept)         LogBMI         LogDPF         LogAge
##      6.7483457      6.6599258      3.4230269      2.6889300      2.0847177
## LogPregnancies     LogInsulin  SkinThickness  BloodPressure
##      2.0673157      0.8504316      0.4227797      0.2088682
```

We can see that the most important variables are *Glucose*, *LogBMI*, *LogDPF* and *LogAge*.

Next, we make the predictions and check the number of instances classified in group:

```
lr_test <- predict(lr_train,newdata=as.data.frame(X_test))
summary(lr_test) %>% kable() %>% kable_styling(latex_options = "striped")
```

We compute the confusion matrix:

```
cm_lg_default <- confusionMatrix(table(Y_test,lr_test))
cm_lg_default$table %>% kable() %>% kable_styling(latex_options = "striped")
```

It can be observed that the model classifies well those individuals who do not have diabetes, but has a hard time classifying those individuals who have diabetes. This is because the dataset is not balanced.

Now, we compute both the TER and the accuracy:

```
lr_TER <- mean(Y_test!=lr_test)
lr_TER
```

```
## [1] 0.2412281
```

```
lr_ACC <- 1 - lr_TER
lr_ACC
```

```
## [1] 0.7587719
```

The value obtained is not bad, but we must keep in mind that the classifier is doing really bad in classifying diabetic individuals. This can be seen more clearly if we calculate the Balanced Accuracy (BAC).

We compute the BAC by using it:

```
bac(cm_lg_default$table)
```

```
## [1] 0.7197421
```

It is observed that the value obtained is noticeably worse than the accuracy, as it balances sensitivity and specificity. We can also check those:

```
cm_lg_default$byClass["Sensitivity"]
```

```
## Sensitivity
##   0.7763975
```

```
cm_lg_default$byClass["Specificity"]
```

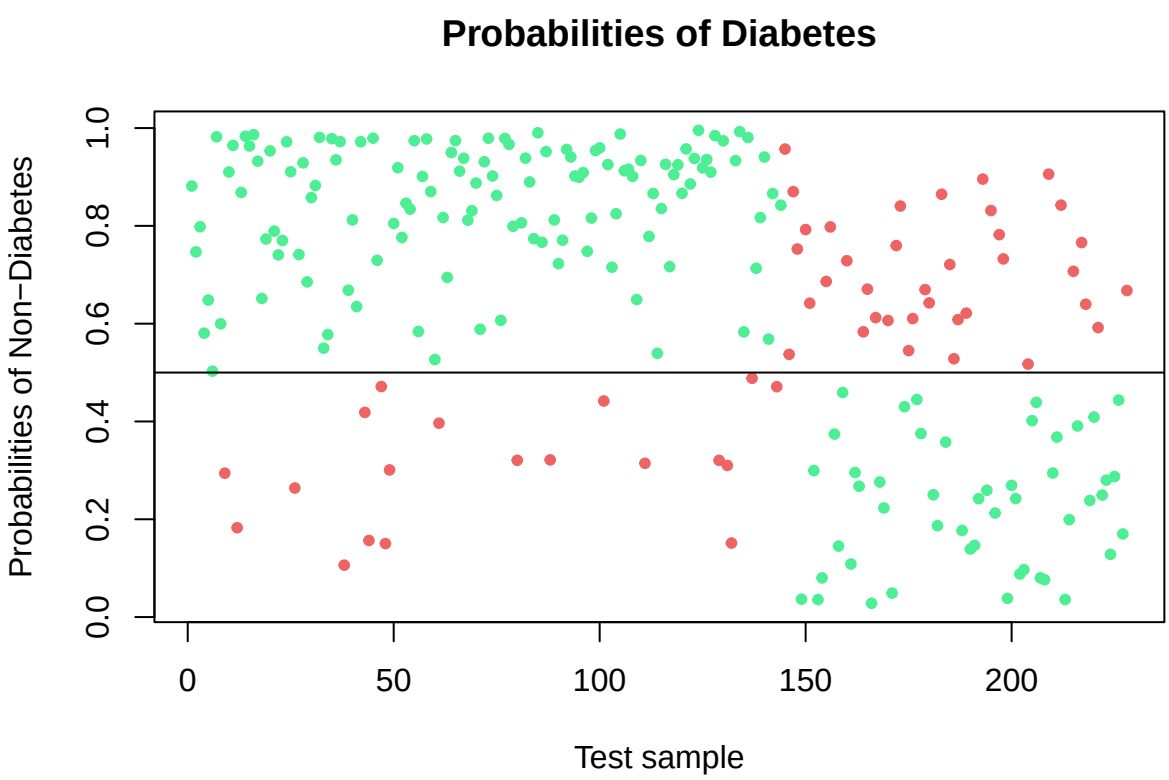
```
## Specificity
##   0.7164179
```

The results are not good.

```
prob_lr_test <- 1 - predict(lr_train,newdata=X_test,type ="probs")
head(prob_lr_test)
```

```
##         6        11        19        29        30        31
## 0.8815139 0.7468028 0.7982075 0.5805886 0.6483164 0.5028685
```

```
colors_errors <- c(color_4,color_3)[1*(Y_test==lr_test)+1]
plot(1:n_test,prob_lr_test,col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Probabilities of Non-Diabetes",
     main="Probabilities of Diabetes")
abline(h=0.5)
```



We are not happy with the performance of the model so we are going to try to improve it.

In order to try to improve it, we are going to first do stepwise model selection. The information criterion that we are going to use in order to evaluate the measure the performance of the model is the *Bayesian Information Criterion* (BIC), and it is defined as

$$BIC = -2l(model) + npar(model) \cdot \log(n)$$

and it aims to balance the model complexity and fitness. We do the stepwide model selection:

```
### BIC
final_df <- as.data.frame(cbind(X_train, Y_train))
final_df$Y_train <- ifelse(final_df$Y_train=="Negative", 0, 1)
mod_zero  <- glm(Y_train ~ 1, family = binomial, data = final_df)
mod_all   <- glm(Y_train ~ ., family = binomial, data = final_df)
model_glm <- MASS::stepAIC(mod_zero, scope = list(lower = mod_zero, upper = mod_all), direction = "both
```

```
model_glm$coefficients
```

```
##  (Intercept)      Glucose       LogBMI      LogAge       LogDPF
## -22.88994363   0.03907509   3.20959104   1.56192570   1.56821657
```

We can see that the variables selected by the model are the same as the four more significant variables selected by the t-test.

By predicting with the new model we obtain very similar results as the previous ones:

	Negative	Positive
Negative	126	18
Positive	38	46

```
X_test_2 <- X_test[, names(model_glm$coefficients)[-1]]

pred <- predict(model_glm, X_test_2, type="response") > 0.5
pred <- c("Negative", "Positive")[(1*pred)+1]
cm <- table(Y_test, pred)
cm %>% kable() %>% kable_styling(latex_options = "striped")
```

And almost the same BAC:

```
BAC <- bac(cm)
BAC
```

```
## [1] 0.7113095
```

As the logistic regression is a scoring classifier (i.e. a classifier that predicts a real value representing the probability that the instance belongs to a certain class, in our case, to be positive in diabetes) we can choose the threshold we can define the threshold at which we decide that the instance is considered to be positive for diabetes. Until now, we have been using 0.5 as the threshold.

Note that this is one of the many options available to handle imbalanced problems. Other approaches may be oversampling or undersampling, but we have chosen to use thresholding because we are using scoring classifiers.

We define a function that takes the model, the data and a given threshold and returns the prediction given that threshold.

```
get_logistic_pred = function(model, data, threshold = 0.5) {
  probs = predict(model,newdata=data,type="response")
  ifelse(probs > threshold, "Positive", "Negative")
}
```

Thus, we can compute the the new predictions for different cuts (0.1, 0.5 and 0.9) by:

```
test_pred_10_lg = get_logistic_pred(model_glm, data = X_test_2, threshold = 0.1)
test_pred_50_lg = get_logistic_pred(model_glm, data = X_test_2, threshold = 0.5)
test_pred_90_lg = get_logistic_pred(model_glm, data = X_test_2, threshold = 0.9)
```

For these thresholds, we now can compute they accuracy, sensitivity and specificity:

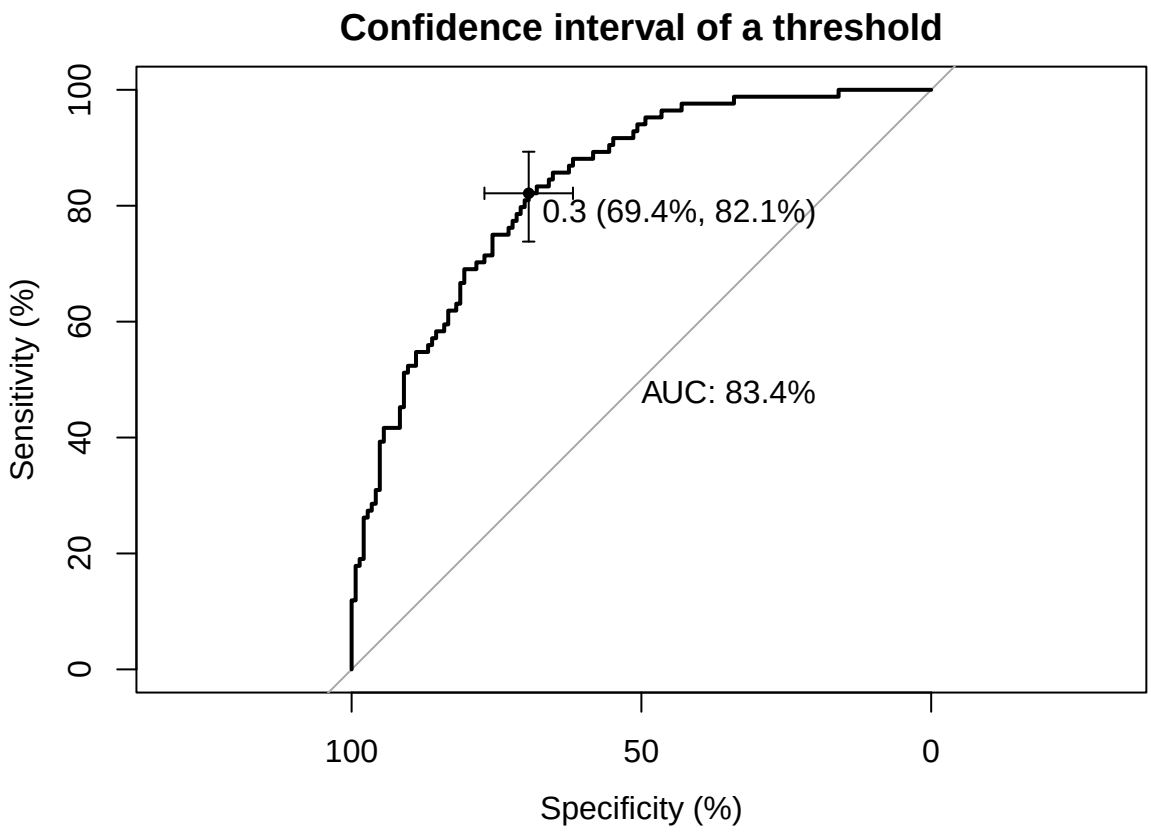
```
test_tab_10_lg = table(predicted = test_pred_10_lg, actual = Y_test)
test_tab_50_lg = table(predicted = test_pred_50_lg, actual = Y_test)
test_tab_90_lg = table(predicted = test_pred_90_lg, actual = Y_test)

test_con_mat_10_lg = confusionMatrix(test_tab_10_lg, positive = "Positive")
test_con_mat_50_lg = confusionMatrix(test_tab_50_lg, positive = "Positive")
test_con_mat_90_lg = confusionMatrix(test_tab_90_lg, positive = "Positive")
```

	Accuracy	Sensitivity	Specificity
c = 0.10	0.6052632	0.9761905	0.3888889
c = 0.50	0.7543860	0.5476190	0.8750000
c = 0.90	0.6754386	0.1190476	1.0000000

```
metrics = rbind(
  c(test_con_mat_10_lg$overall["Accuracy"],
    test_con_mat_10_lg$byClass["Sensitivity"],
    test_con_mat_10_lg$byClass["Specificity"]),

  c(test_con_mat_50_lg$overall["Accuracy"],
    test_con_mat_50_lg$byClass["Sensitivity"],
    test_con_mat_50_lg$byClass["Specificity"]),

  c(test_con_mat_90_lg$overall["Accuracy"],
    test_con_mat_90_lg$byClass["Sensitivity"],
    test_con_mat_90_lg$byClass["Specificity"])

)
rownames(metrics) = c("c = 0.10", "c = 0.50", "c = 0.90")
colnames(metrics) = c("Accuracy", "Sensitivity", "Specificity")

metrics %>% kable() %>% kable_styling(latex_options = "striped")
```

As can be seen, with the threshold=0.1 we obtain a high sensitivity but a low specificity and, for the threshold=0.9 a low sensitivity but a low specificity. We can plot a ROC curve to search for the best threshold.

```
test_prob = predict(model_glm, newdata = X_test_2, type = "response")

ROC_lr = plot.roc(Y_test, test_prob,
         main="Confidence interval of a threshold", percent=TRUE,
         ci=TRUE, of="thresholds",
         thresholds="best",
         print.thres="best",
         print.auc=TRUE)
```



It is around 0.4

```
ROC_lr
```

```
##
## Call:
## plot.roc.default(x = Y_test, predictor = test_prob, main = "Confidence interval of a threshold",
##
## Data: test_prob in 144 controls (Y_test Negative) < 84 cases (Y_test Positive).
## Area under the curve: 83.45%
## 95% CI (2000 stratified bootstrap replicates):
##  thresholds sp.low sp.median sp.high se.low se.median se.high
##   0.2863612  61.81     69.44   77.08  73.81     82.14   89.32
```

If we compute the predictions with the new threshold:

```
pred <- predict(model_glm, X_test_2, type = "response") >
  as.numeric(rownames(ROC_lr$ci$specificity))
pred <- pred * 1
pred <- as.factor(c("Negative", "Positive")[(1*pred)+1])
cm_lr <- `confusionMatrix`(table(Y_test, pred), positive="Positive")
cm_lr$table %>% kable() %>% kable_styling(latex_options = "striped")
```

We can see a much better confusion matrix, and, if we compute de BAC:

	Negative	Positive
Negative	100	44
Positive	15	69

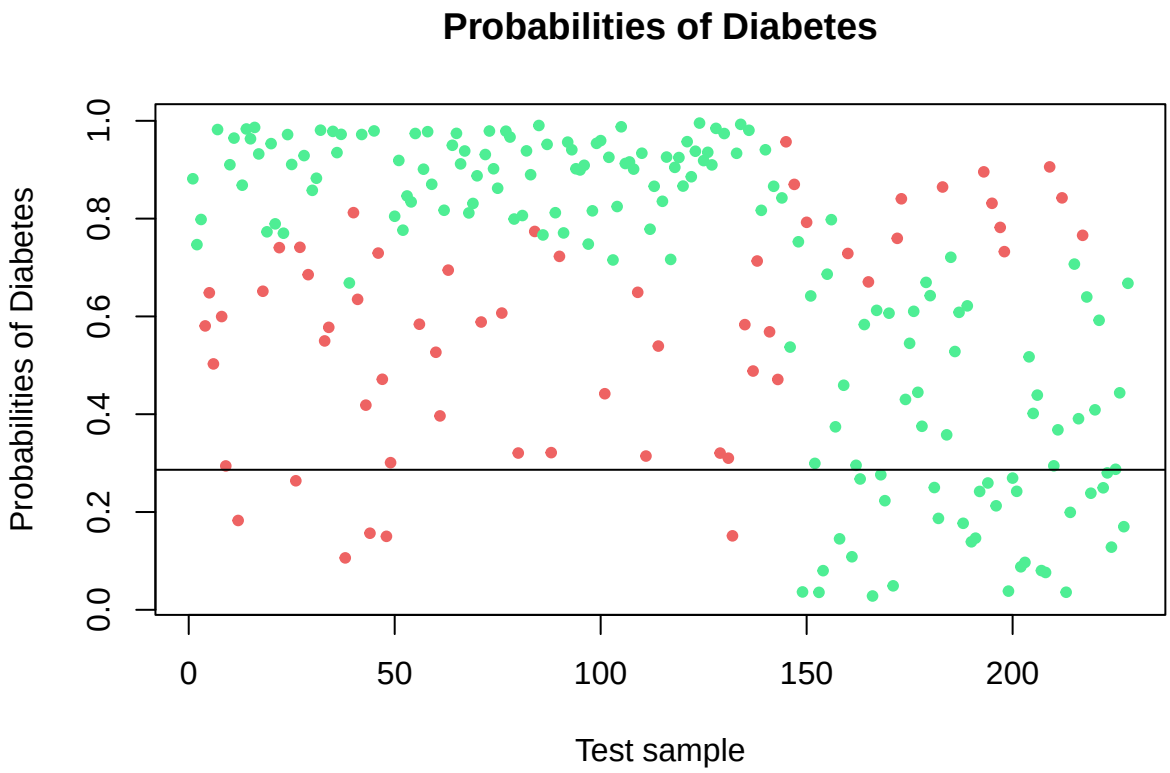
```
BAC <- bac(cm_lr$table)
BAC
```

```
## [1] 0.7579365
```

It has improved a quite a bit.

Finally, we plot the probability of diabetes for each instance, as well as the threshold.

```
colors_errors <- c(color_4,color_3)[1*(Y_test==pred)+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=as.numeric(rownames(ROC_lr$ci$specificity)))
```



Methods based on the Bayes Theorem

We are going to try three methods based on the Bayes Theorem:

- Linear discriminant analysis

	Negative	Positive
Negative	125	19
Positive	36	48

- Quadratic discriminant analysis
- Naive Bayes

These methods estimate the prior probabilities as the proportion of available observations belonging to each class and they make predictions selecting the class for which the posterior probability becomes maximum.

As they take into account the prior probabilities, these methods are very convenient for unbalanced problems (as the problem we are facing), so minimal tuning may be needed.

Linear discriminant analysis (LDA)

First, we train the model by:

```
lda_train <- lda(Y_train~.,data=X_train)
```

As it is a bayesian method, it consider prior probabilities so it is taking into account that the dataset is balanced. The prior probabilities are:

```
lda_train$prior
```

```
##  Negative  Positive
## 0.6616257 0.3383743
```

that are the same as the proportion of people with and without diabetes in our dataset:

```
c(pr_no, pr_yes)
```

```
## [1] 0.652576 0.347424
```

With the model fitted, we can make the predictions as:

```
lda_test <- predict(lda_train,newdata=X_test)
```

And save the predicted classes as:

```
lda_Y_test <- lda_test$class
```

The confusion matrix is:

```
cm_LDA <- confusionMatrix(table(Y_test,lda_Y_test), positive="Positive")
cm_LDA$table %>% kable() %>% kable_styling(latex_options = "striped")
```

Computing the Accuracy and the TER might be interesting too.

```r
lda_TER <- mean(Y_test!=lda_Y_test)
lda_TER
```

```
## [1] 0.2412281
```

```r
lda_accuracy <- 1 - lda_TER
lda_accuracy
```

```
## [1] 0.7587719
```

If we compute the BAC we can see that it is much better than the default logistic regression.

```r
bac(cm_LDA$table)
```

```
## [1] 0.7197421
```

The reason behind this is that, as said earlier, due to its bayesian nature, it takes into account that the problem is unbalanced. If we compute its sensitivity and specificity:

```r
cm_LDA$byClass["Sensitivity"]
```

```
## Sensitivity
##   0.7164179
```

```r
cm_LDA$byClass["Specificity"]
```

```
## Specificity
##   0.7763975
```

It can be seen that the specificity is much larger than the sensitivity. We can also compute the BAC by:

```r
bac(cm_LDA$table)
```

```
## [1] 0.7197421
```

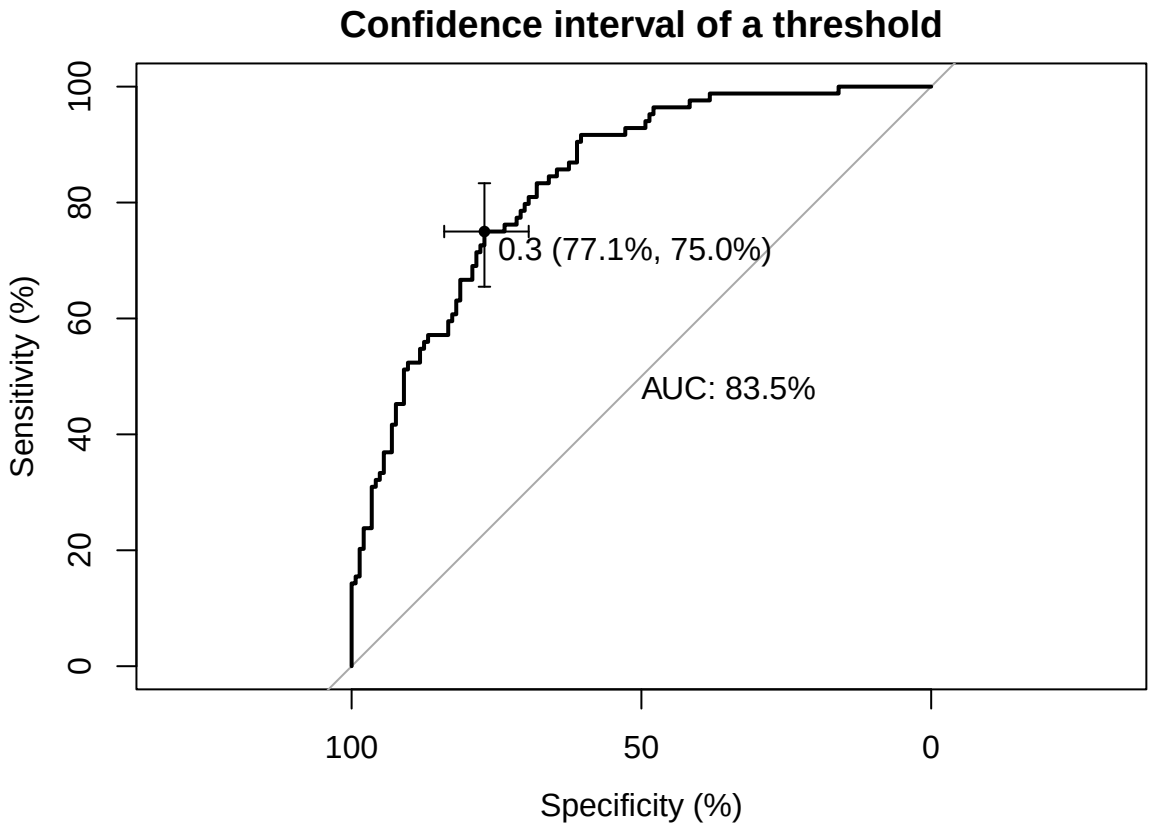
Recall can obtain the conditional probabilities of the classifications made with the test sample:

```r
prob_lda_Y_test <- lda_test$posterior
head(prob_lda_Y_test)
```

```
##     Negative  Positive
## 6  0.8868738 0.1131262
## 11 0.7787013 0.2212987
## 19 0.8458111 0.1541889
## 29 0.5125756 0.4874244
## 30 0.6676824 0.3323176
## 31 0.5513849 0.4486151
```

Thanks to this, we can treat it as a scoring classifier. In order to find the optimum threshold, we can plot a ROC curve:

```
test_prob = predict(lda_train, newdata = X_test, type = "response")
test_prob <- test_prob$posterior[,2]

ROC_lda = plot.roc(Y_test, test_prob,
                main="Confidence interval of a threshold", percent=TRUE,
                ci=TRUE, of="thresholds",
                thresholds="best",
                print.thres="best",
                print.auc=TRUE)
```



According to the ROC curve, the optimum threshold is:

```
as.numeric(rownames(ROC_lda$ci$specificity))
```

```
## [1] 0.332903
```

If we compute the predictions with the new threshold:

```
pred <- predict(lda_train, X_test, type = "response")$posterior[,2] >
  as.numeric(rownames(ROC_lda$ci$specificity))
pred <- pred * 1
pred <- as.factor(c("Negative", "Positive")[(1*pred)+1])
cm_lda_opt <- confusionMatrix(table(Y_test, pred), positive="Positive")
cm_lda_opt$table %>% kable() %>% kable_styling(latex_options = "striped")
```

	Negative	Positive
Negative	111	33
Positive	21	63

We can also compute the BAC:

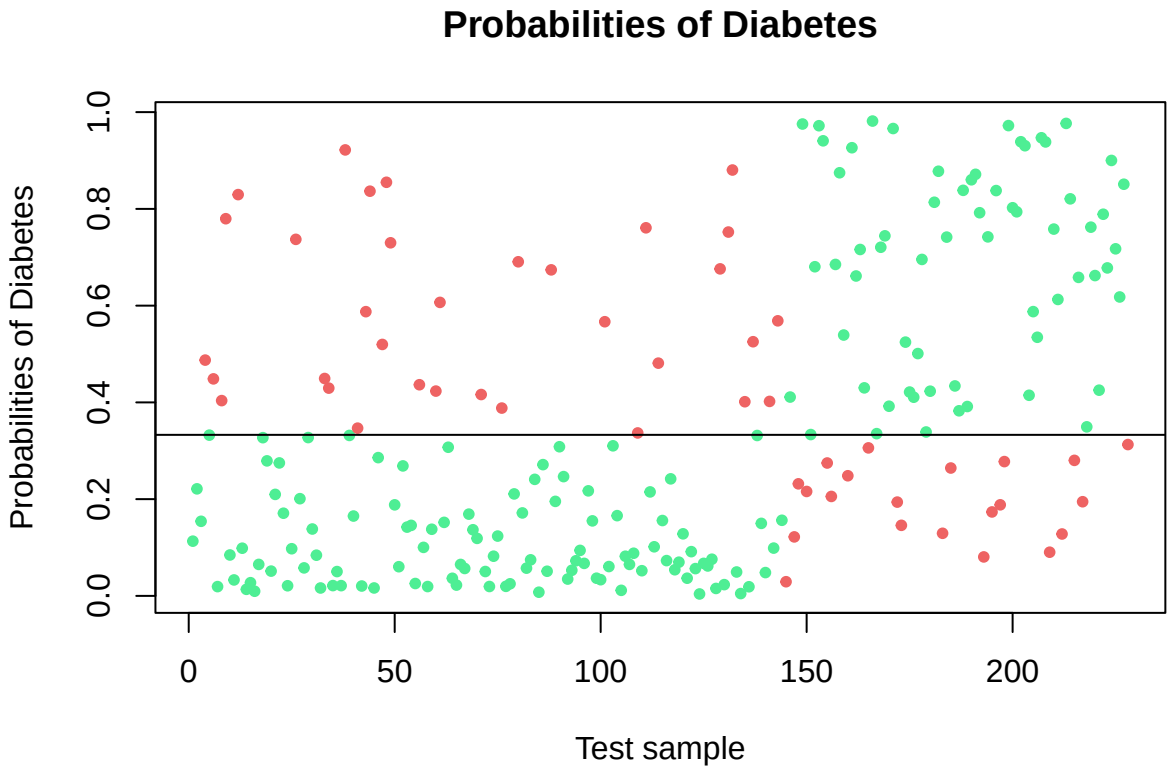
```
BAC <- bac(cm_lda_opt$table)
BAC
```

```
## [1] 0.7604167
```

It is similar to the BAC computed in the logistic regression after tuning the threshold.

Finally, we plot the probability of being positive of diabetes and the threshold:

```
probs <- predict(lda_train, X_test, type = "response")$posterior[,2]
colors_errors <- c(color_4,color_3)[1*(Y_test==pred)+1]
plot(1:n_test,probs,col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Probabilities of Diabetes",
     main="Probabilities of Diabetes")
abline(h=as.numeric(rownames(ROC_lda$ci$specificity)))
```



Quadraric discriminant analysis (QDA)

First, we train the model by:

	Negative	Positive
Negative	121	23
Positive	36	48

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

The workflow for this method will be mostly the same as with the LDA.

We make the predictions as:

```
qda_test <- predict(qda_train,newdata=X_test)
```

We can also save the predictions as:

```
qda_Y_test <- qda_test$class
```

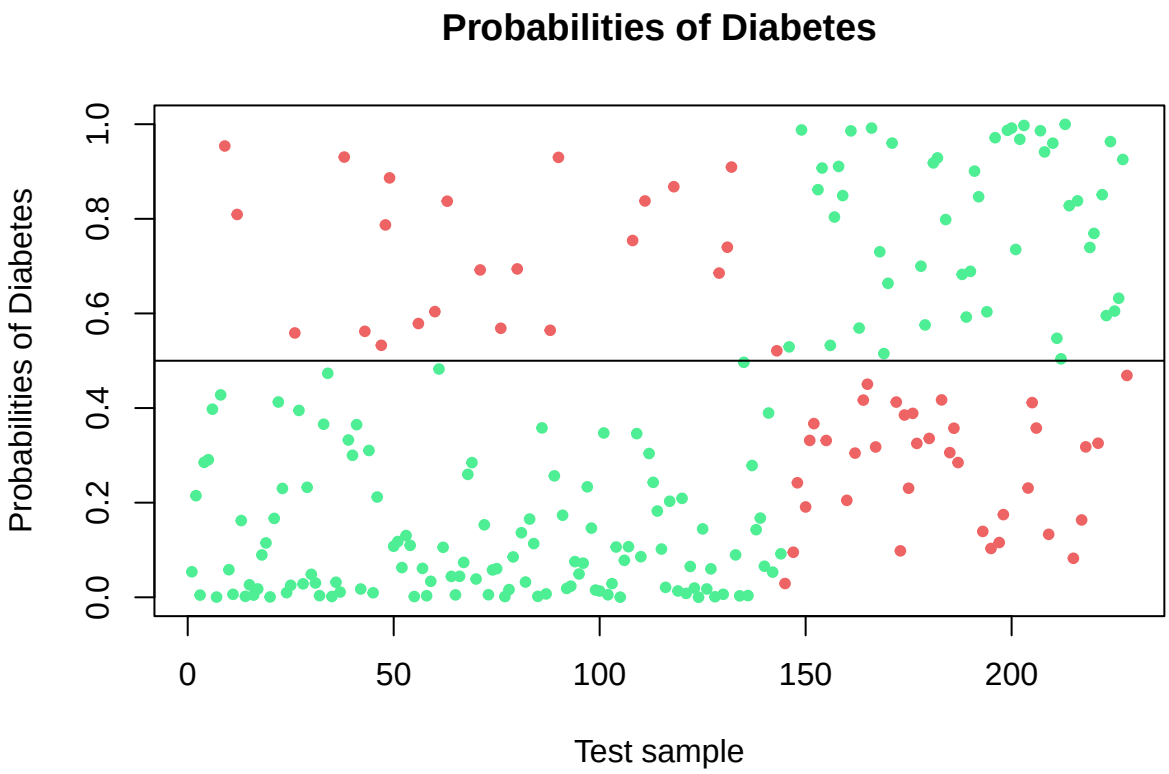
And the confusion table is:

```
cm_QDA <- confusionMatrix(table(Y_test,qda_Y_test), positive="Positive")
cm_QDA$table %>% kable() %>% kable_styling(latex_options = "striped")
```

This is the same confusion table as for the default LDA! This means that accuracy, sensitivity, specificity... will be all the same.

Finally, we plot the probabilities of having diabetes and the default threshold.

```
probs <- predict(qda_train, X_test, type = "response")$posterior[,2]
colors_errors <- c(color_4,color_3)[1*(Y_test==qda_Y_test)+1]
plot(1:n_test,probs,col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Probabilities of Diabetes",
     main="Probabilities of Diabetes")
abline(h=0.5)
```



Naive Bayes (NB)

We start by training the naive-bayes model:

```
nb_train <- gaussian_naive_bayes(X_train,Y_train)
```

And computing the predictions:

```
nb_test <- predict(nb_train,newdata=as.matrix(X_test),type="prob")
```

Next, we generate the vector of classifications:

```
nb_Y_test <- as.factor(ifelse(nb_test[,2]>0.5, "Positive", "Negative"))
```

By computing the confusion matrix, we can see that the method yields good results:

```
cm_nb <- confusionMatrix(table(Y_test,nb_Y_test), positive = "Positive")
cm_nb$table
```

```
##           nb_Y_test
## Y_test     Negative Positive
##   Negative      115       29
##   Positive       29       55
```

It may be interesting to compute TER, accuracy, sensitivity, specificity...

```
nb_TER <- mean(Y_test!=nb_Y_test)
print(paste0("Test Error Rate: ",nb_TER))
```

```
## [1] "Test Error Rate: 0.254385964912281"
```

```
nb_acc <- 1 - nb_TER
print(paste0("Accuracy: ",nb_acc))
```

```
## [1] "Accuracy: 0.745614035087719"
```

```
cm_nb$byClass["Sensitivity"]
```

```
## Sensitivity
##   0.6547619
```

```
cm_nb$byClass["Specificity"]
```

```
## Specificity
##   0.7986111
```

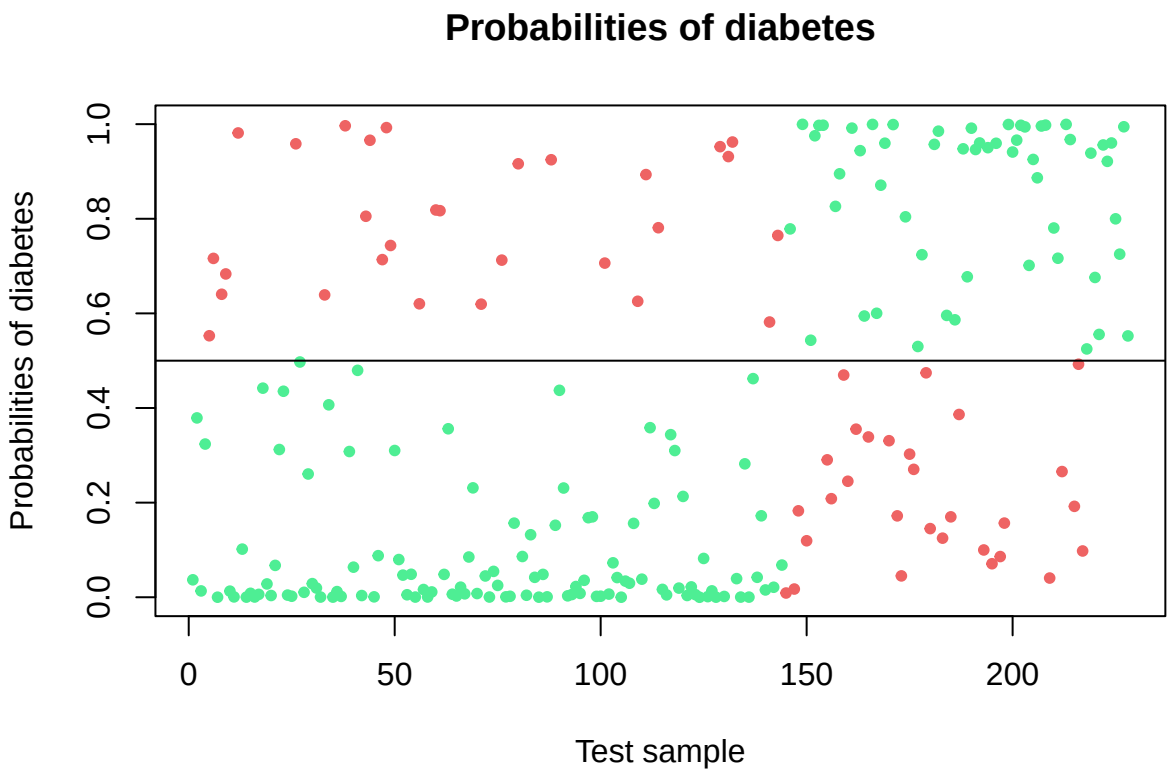
The results are good, specially for the specificity. We can also compute the BAC:

```
bac(cm_nb$table)
```

```
## [1] 0.7266865
```

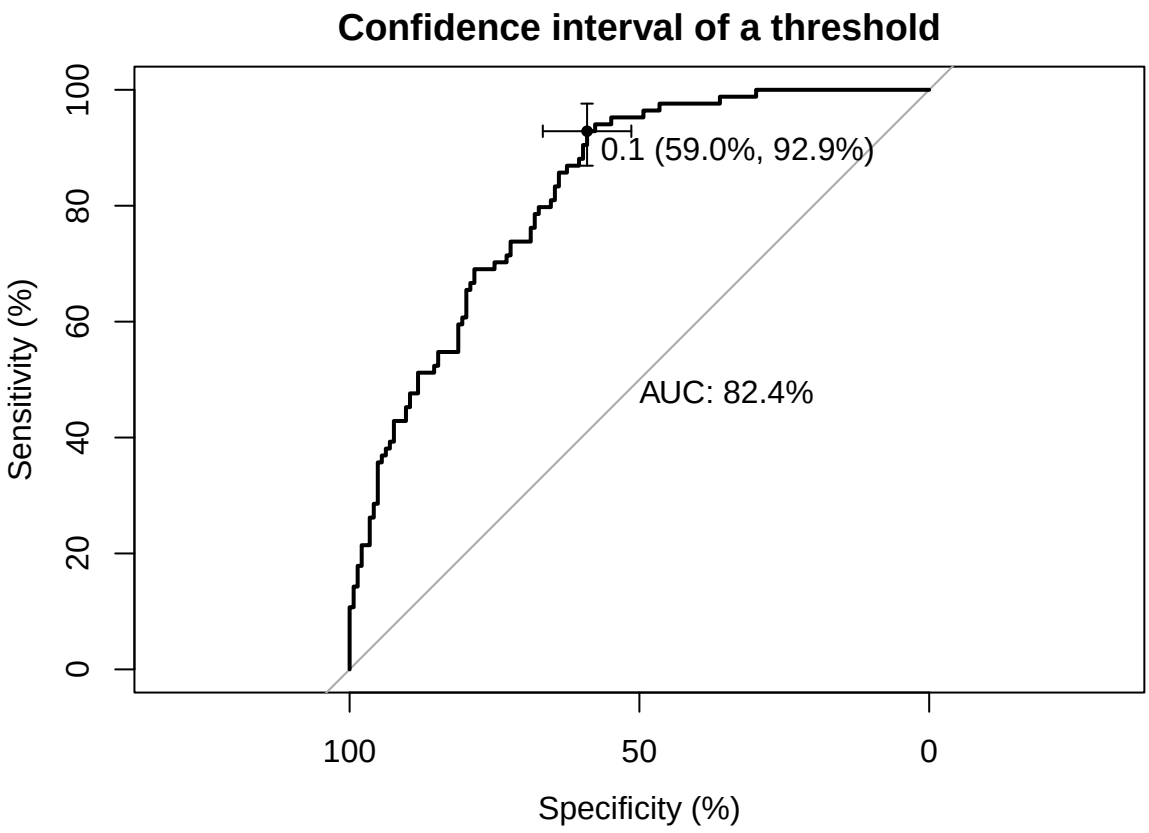
The BAC is also as good as the accuracy. Ploting the probabilities of diabetes for each instance:

```
test_prob <-  predict(nb_train, newdata = as.matrix(X_test), type = "prob")
prob_nb_Y_test <- test_prob[,2]
colors_errors <- c(color_4,color_3)[1*(Y_test==nb_Y_test)+1]
plot(1:n_test,prob_nb_Y_test,col=colors_errors,pch=20,type="p",
     xlab="Test sample",ylab="Probabilities of diabetes",
     main="Probabilities of diabetes")
abline(h=0.5)
```



We can see different threshold ploting the ROC curve:

```
ROC_nb = plot.roc(Y_test, prob_nb_Y_test,
                  main="Confidence interval of a threshold", percent=TRUE,
                  ci=TRUE, of="thresholds",
                  thresholds="best",
                  print.thres="best",
                  print.auc=TRUE)
```




```
ROC_nb
```

```
##
## Call:
## plot.roc.default(x = Y_test, predictor = prob_nb_Y_test, main = "Confidence interval of a threshold"
##
## Data: prob_nb_Y_test in 144 controls (Y_test Negative) < 84 cases (Y_test Positive).
## Area under the curve: 82.44%
## 95% CI (2000 stratified bootstrap replicates):
##  thresholds sp.low sp.median sp.high se.low se.median se.high
##  0.09282791  51.39     59.03   66.67   86.9     92.86   97.62
```

Conclusions of the section

We have tried five different supervised classification methods:

- Logistic regression (LR)
- Linear discriminant analysis (LDA)
- Quadratic discriminant analysis (QDA)
- Naive Bayes (NB)
- K-Nearest Neighbors (KNN)

The results of the best classifier obtained for each of the methods can be seen in the following table:

	KNN	LR	LDA	QDA	NB
Accuracy	0.7061404	0.7412281	0.7631579	0.7412281	0.7456140
Balanced Accuracy	0.6631944	0.7579365	0.7604167	0.7058532	0.7266865
Sensitivity	0.6268657	0.6106195	0.6562500	0.6760563	0.6547619
Specificity	0.7391304	0.8695652	0.8409091	0.7707006	0.7986111

```
results_table <- data.frame(x=c(0,0,0,0))
for(i in 1:length(predictors_tables)){
  v <- c(predictors_tables[[i]]$overall["Accuracy"],
         bac(predictors_tables[[i]]$table),
         predictors_tables[[i]]$byClass["Sensitivity"],
         predictors_tables[[i]]$byClass["Specificity"]
         )
  results_table <- cbind(results_table, v)
}
results_table <- results_table[,-1]
rownames(results_table) <- c("Accuracy", "Balanced Accuracy", "Sensitivity", "Specificity")
colnames(results_table) <- c("KNN", "LR", "LDA", "QDA", "NB")

results_table %>% kable() %>% kable_styling(latex_options = "striped")
```

As we were faced with an unbalanced problem, it was not enough to look at metrics such as accuracy, and we had to consult others such as balanced accuracy, sensitivity or specificity. According to these metrics, the default LR did quite poorly, but after thresholding with the help of the ROC curve, we got really good results.

On the other hand, and as we have commented throughout this section, Bayesian methods have the advantage in this type of problem that they take into account the a priori probability, so that by default they already consider that the problem is unbalanced. Thanks to this, they achieve very good results by default, but, depending on the levels of sensitivity and specificity to be achieved, it may be a good idea to perform thresholding.

We believe that considering what levels of sensitivity and specificity we want to obtain is especially important in medical problems (such as the one we are facing), since, for certain problems, we may be especially interested in correctly classifying positive individuals (Sensitivity), in others negative individuals (Specificity), or in others in finding a balance (Balanced Accuracy).