heart_attack

Oumar Kane

15/03/2022

Variables selection with heart attack dataset

We have recuperated a dataset named heart_attack which is very interesting for a debut.

We can recuperate the dataset by indicating the path toward it.

```
heart_data = read.csv("E:/Oumar/Ordinateur Dell/oumar/documents/Cours/IA data forest/master semestre 2/
```

Exploration

We have to explore the data set and see how we can do perfectly the data processes.

Let's attach the dataset to manipulate the columns.

```
attach(heart_data)
```

We have to see the five first lines of the dataset and verify the types.

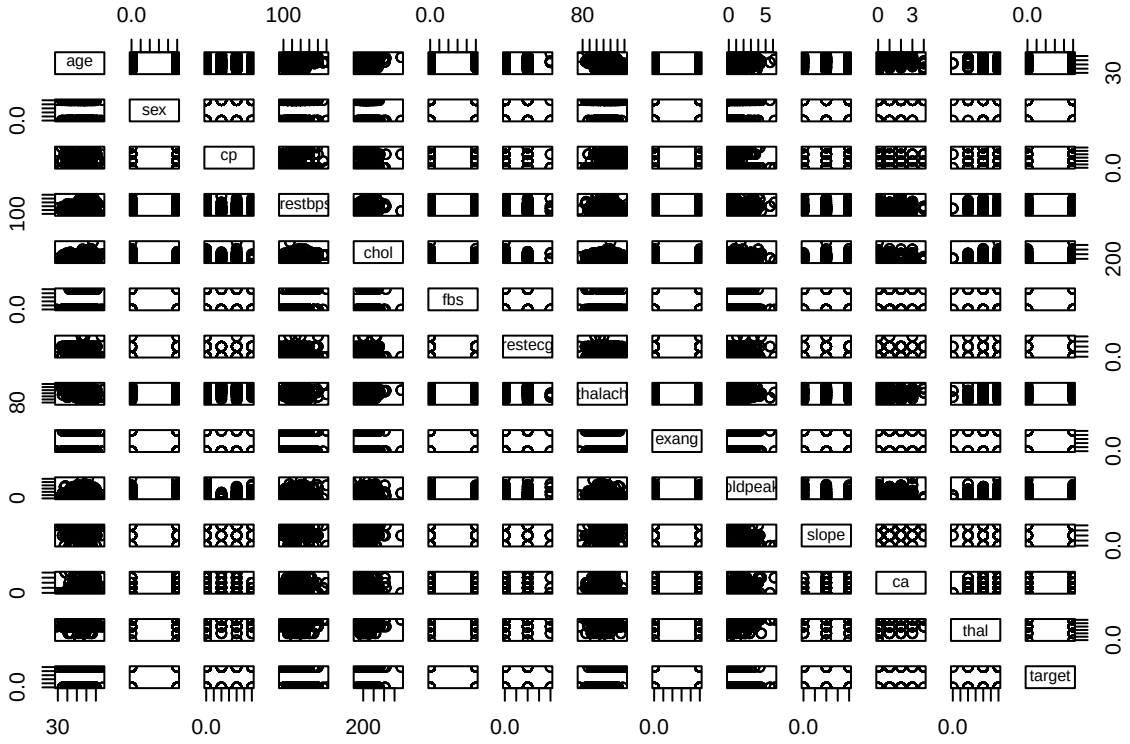
```
head(heart_data)
```

```
##   age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 1  52   1  0      125  212   0       1     168     0     1.0     2  2    3
## 2  53   1  0      140  203   1       0     155     1     3.1     0  0    3
## 3  70   1  0      145  174   0       1     125     1     2.6     0  0    3
## 4  61   1  0      148  203   0       1     161     0     0.0     2  1    3
## 5  62   0  0      138  294   1       1     106     0     1.9     1  3    2
## 6  58   0  0      100  248   0       0     122     0     1.0     1  0    2
##   target
## 1      0
## 2      0
## 3      0
## 4      0
## 5      0
## 6      1
```

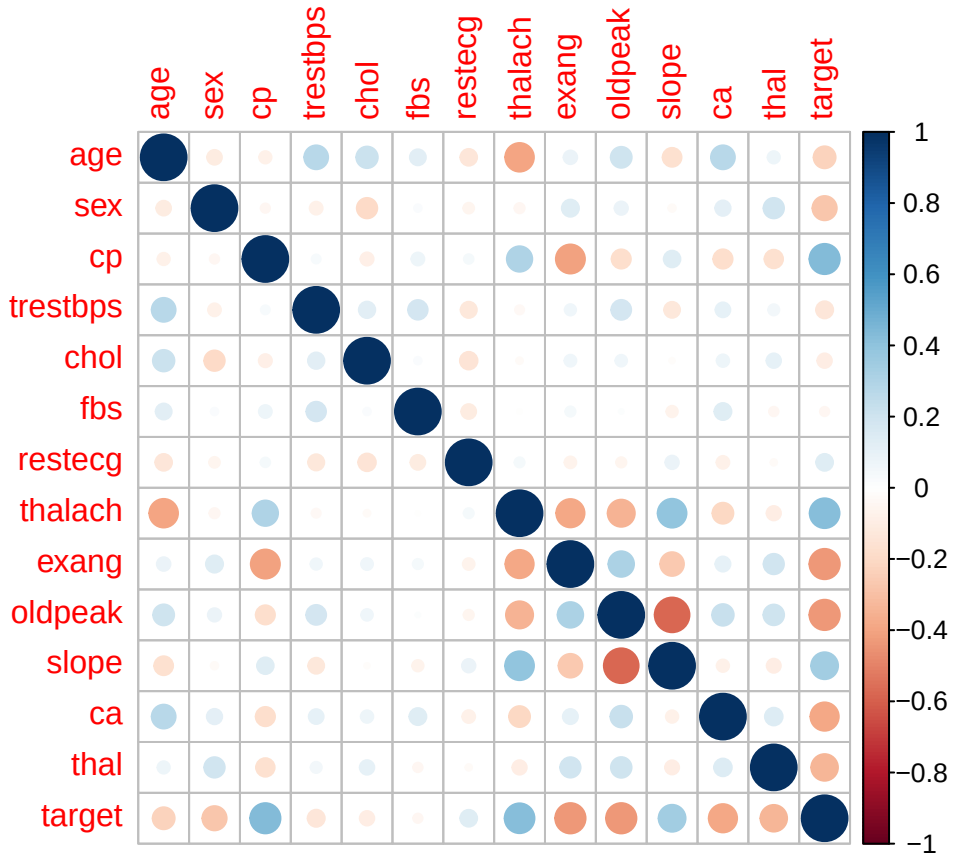
The data is already clean. We have some categorical variables, but those variables are encoded. Let's profile deeper the dataset.

Let's see if the different variables are correlated together with a pair plot and a corrplot.

```
pairs(heart_data)
```

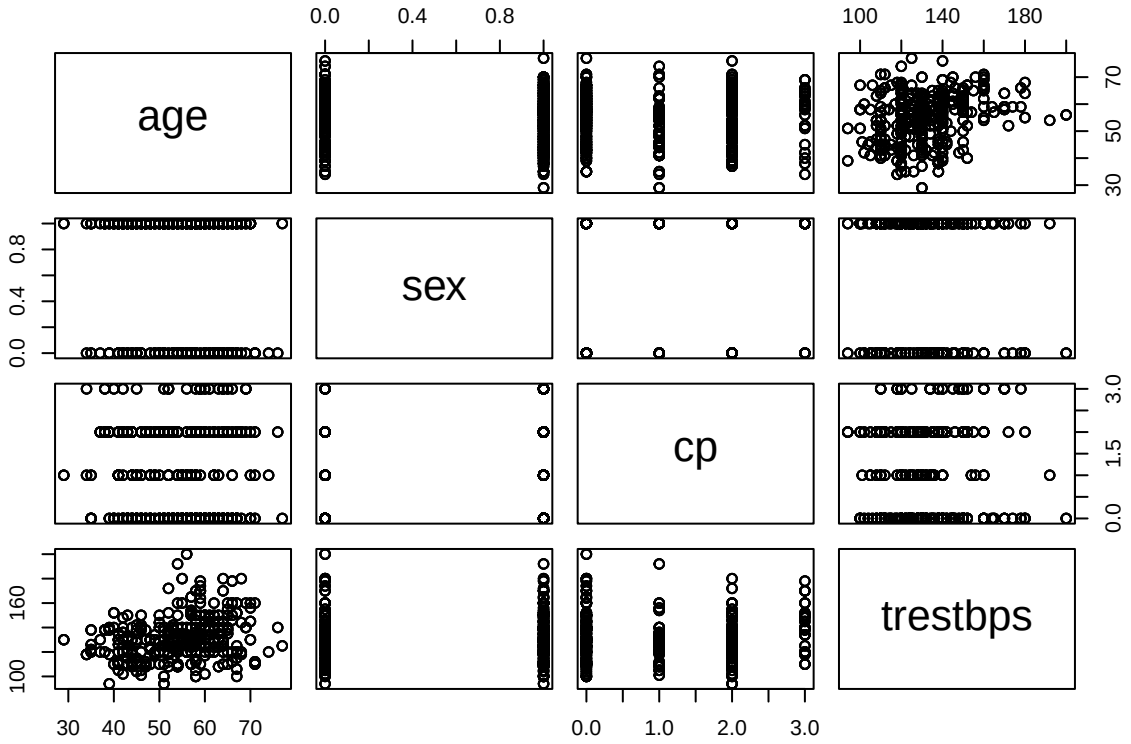



```
corrplot(cor(heart_data))
```



The plots traced between two variables are not distinct if we use the whole dataset. But we can do it again with only a few variables.

```
# choose only four first variables
pairs(heart_data[, 1:4])
```

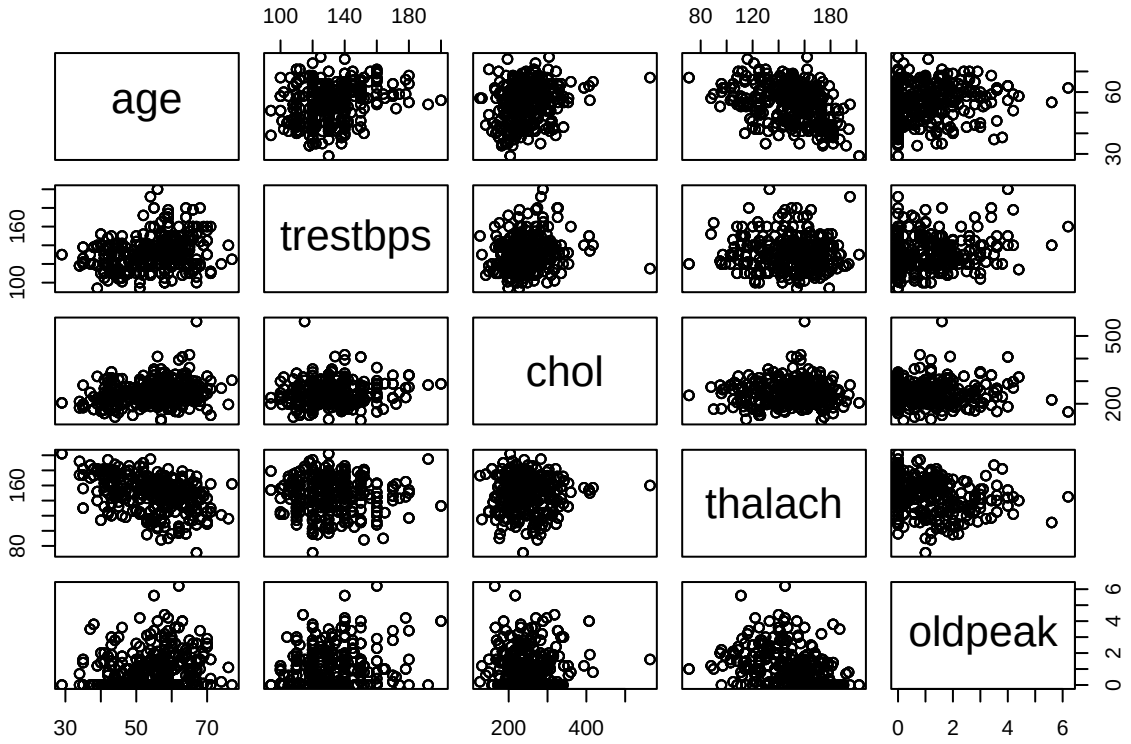


We see that, with the categorical variables, we did not obtain good distributions. We can't interpret the plots with the categorical variables but only with the quantitative variables. Let's recuperate only the quantitative variables in a new data frame.

```
quanti = heart_data[, c(1, 4, 5, 8, 10)]
```

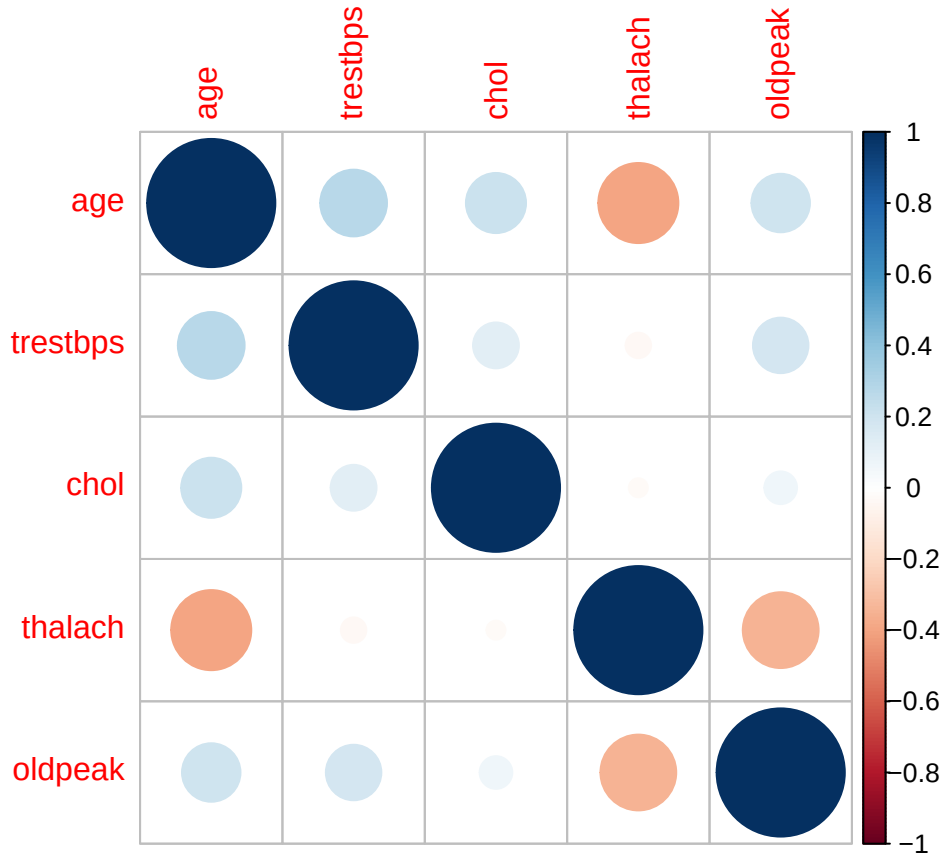
Let's trace a new pair plot without categorical variables.

```
pairs(quanti)
```



We can see that some variables like thalach and age or chol and age are correlated.

```
# Let's trace the corrplot to see more clearly the variables interactions
corrplot(cor(quanti))
```



Conclusion of corrplot: We see that many variables are correlated together. We see correlations between :

- The age variable and the other quantitative variables;
- oldpeak, trestbps and thalach
- chol and trestbps.

Abnormal values

We must change the type of the target to factor at first.

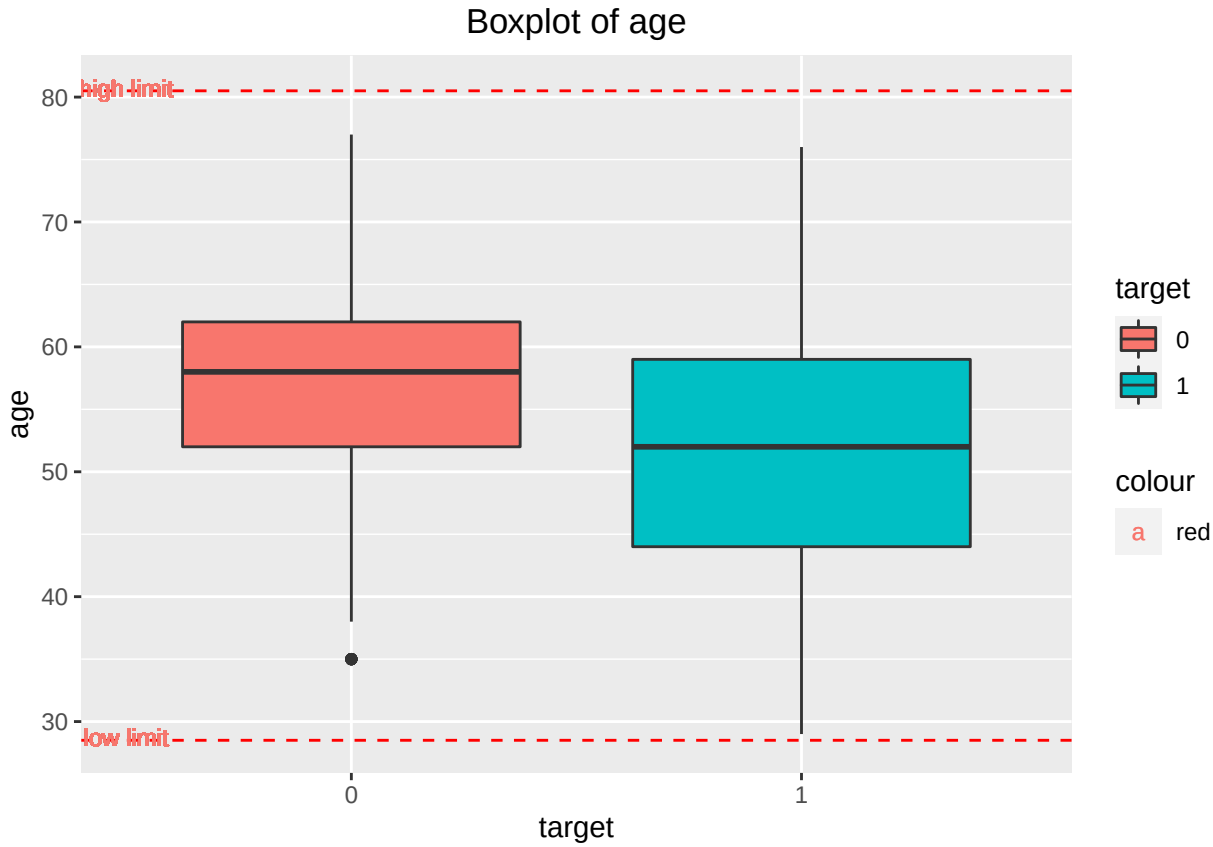
```
heart_data[,14] = as.factor(heart_data[,14])
```

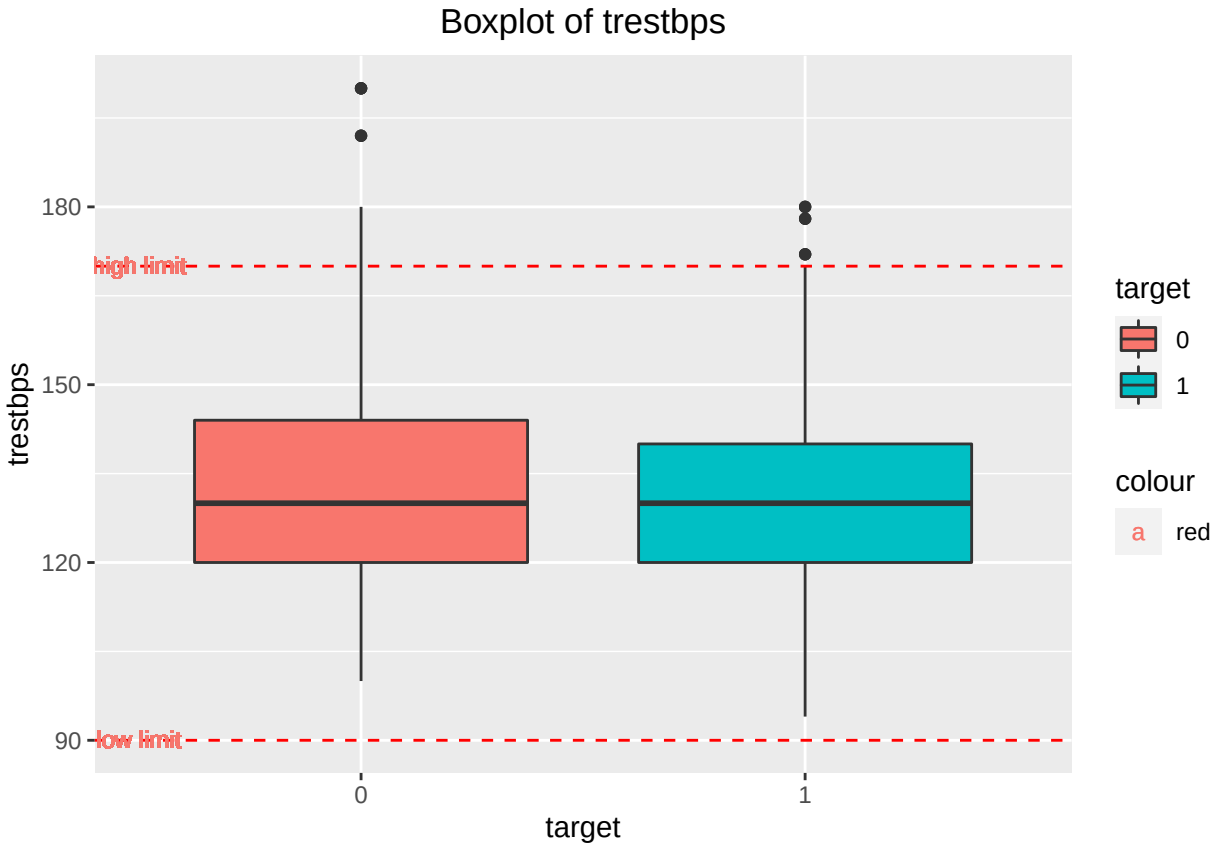
Let's verify if the data set contains abnormal values with box plots.

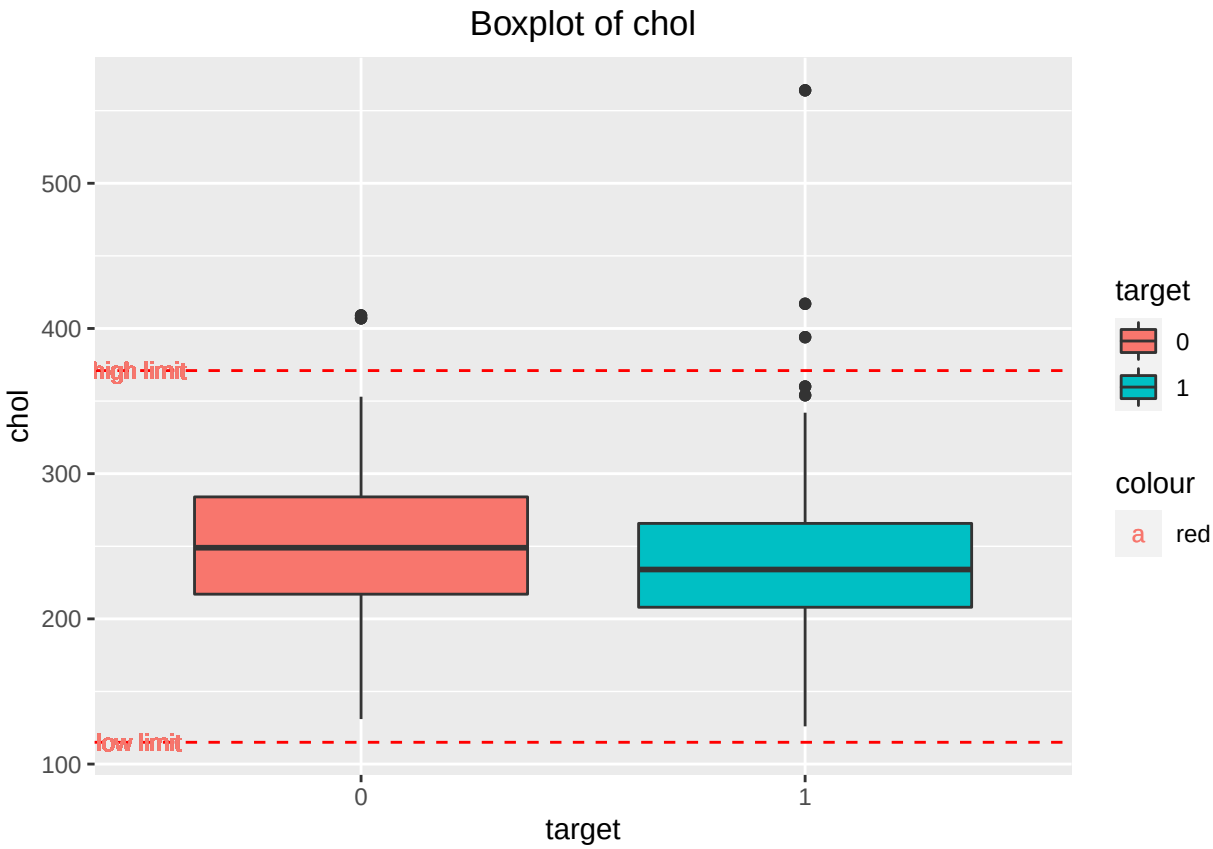
We can create a function that we will use for tracing box plot for each variable of the data set. We can add value limitations to the boxplots.

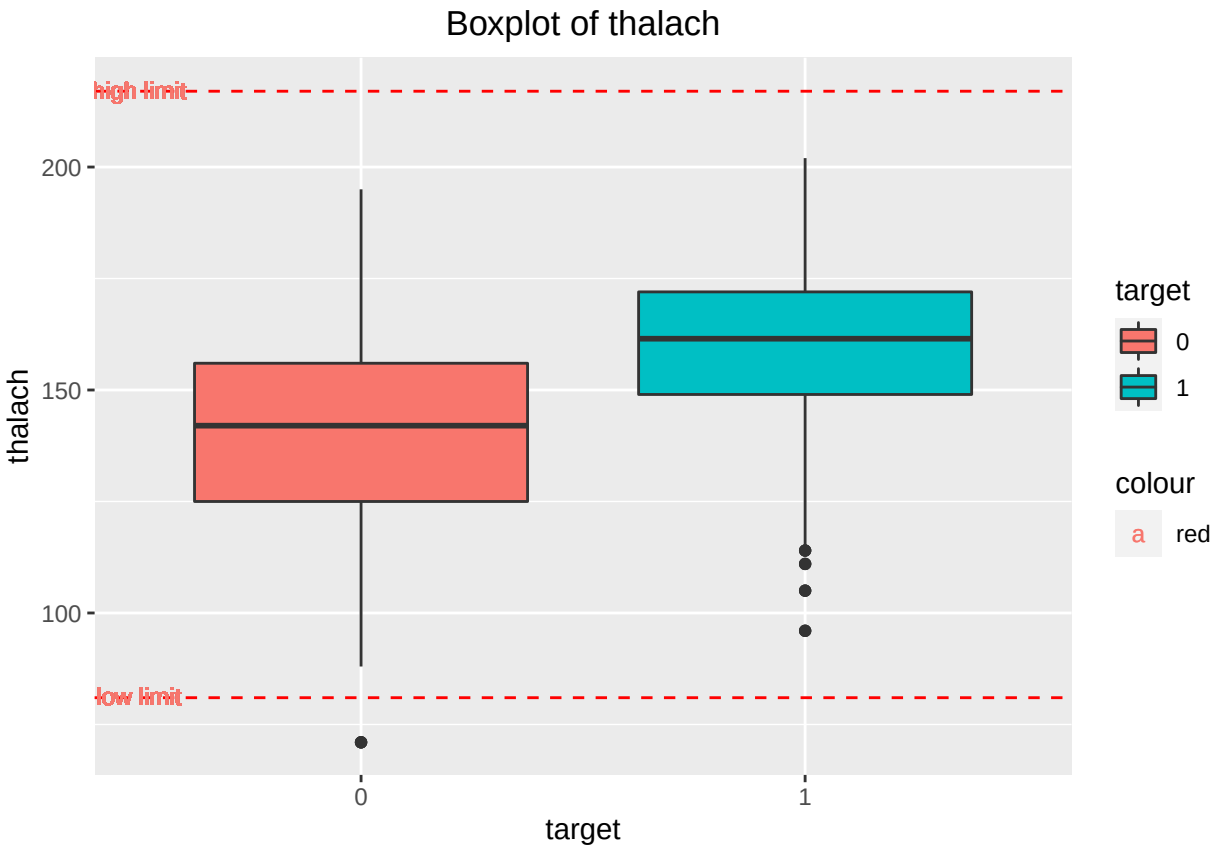
```
trace_boxplots = function(variables, data, quanti_data = NULL)
  for(i in 1:length(variables)){
    # Recuperate the data set
    if(is.null(quanti_data)){
      quanti_data = data
    }
```

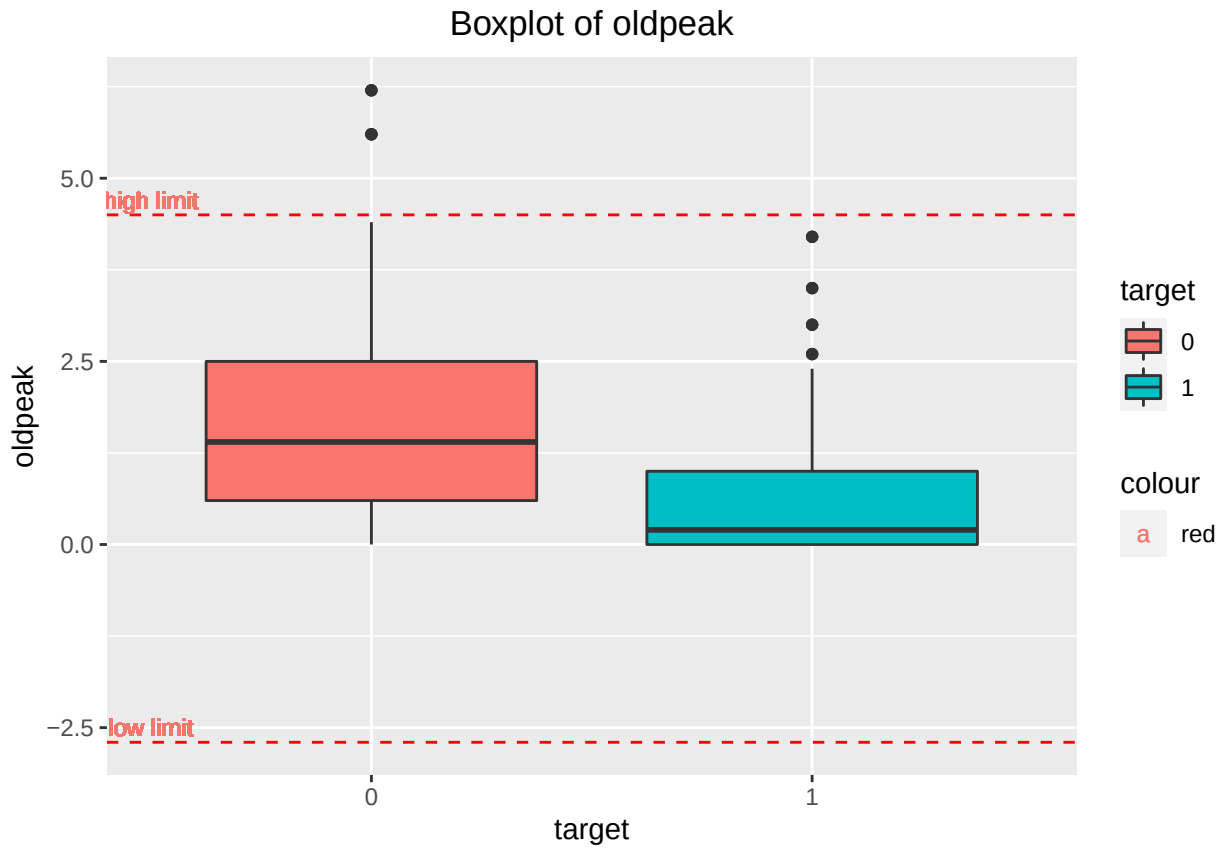
```
    ## Calculate the limitations
    # quantile 0.25
    q1 = quantile(quanti_data[, i], 0.25)

    # quantile 0.75
    q2 = quantile(quanti_data[, i], 0.75)

    # interquartile
    inter_q = q2 - q1

    # high limit
    high_limit = q2 + 1.5*inter_q

    # low limit
    low_limit = q1 - 1.5*inter_q

    # Get the variable
    variable = variables[i]

    # create the graphic
    plot = ggplot(data = data, aes_string("target", variable, fill = "target")) +
      geom_boxplot() +
      ggtitle(paste("Boxplot of", variable)) +
      geom_hline(yintercept = high_limit, color = "red", linetype = "dashed") +
      geom_hline(yintercept = low_limit, color = "red", linetype = "dashed") +
      geom_text(aes(0.5, high_limit+.2, label = "high limit", color = "red"), size = 3) +
      geom_text(aes(0.5, low_limit+.2, label = "low limit", color = "red"), size = 3) +
      theme(plot.title = element_text(hjust = 0.5))

    # Show out the plot
    print(plot)
  }

# Recuperate the quantitative variables' names.
quanti_variables = names(quanti)

# Let's use, now, the function to trace the box plots.
trace_boxplots(quanti_variables, heart_data, quanti)
```


Boxplot of age
high limit
low limit
80
70
60
50
40
30
age
0
1
target
target
0
1
colour
a
red


Boxplot of trestbps
trestbps
180
150
120
90
high limit
low limit
0
1
target
target
0
1
colour
a
red


Boxplot of chol
chol
500
400
300
200
100
high limit
low limit
0
1
target
target
0
1
colour
a
red


Boxplot of thalach
high limit
low limit
200
150
100
thalach
0
1
target
target
0
1
colour
a
red

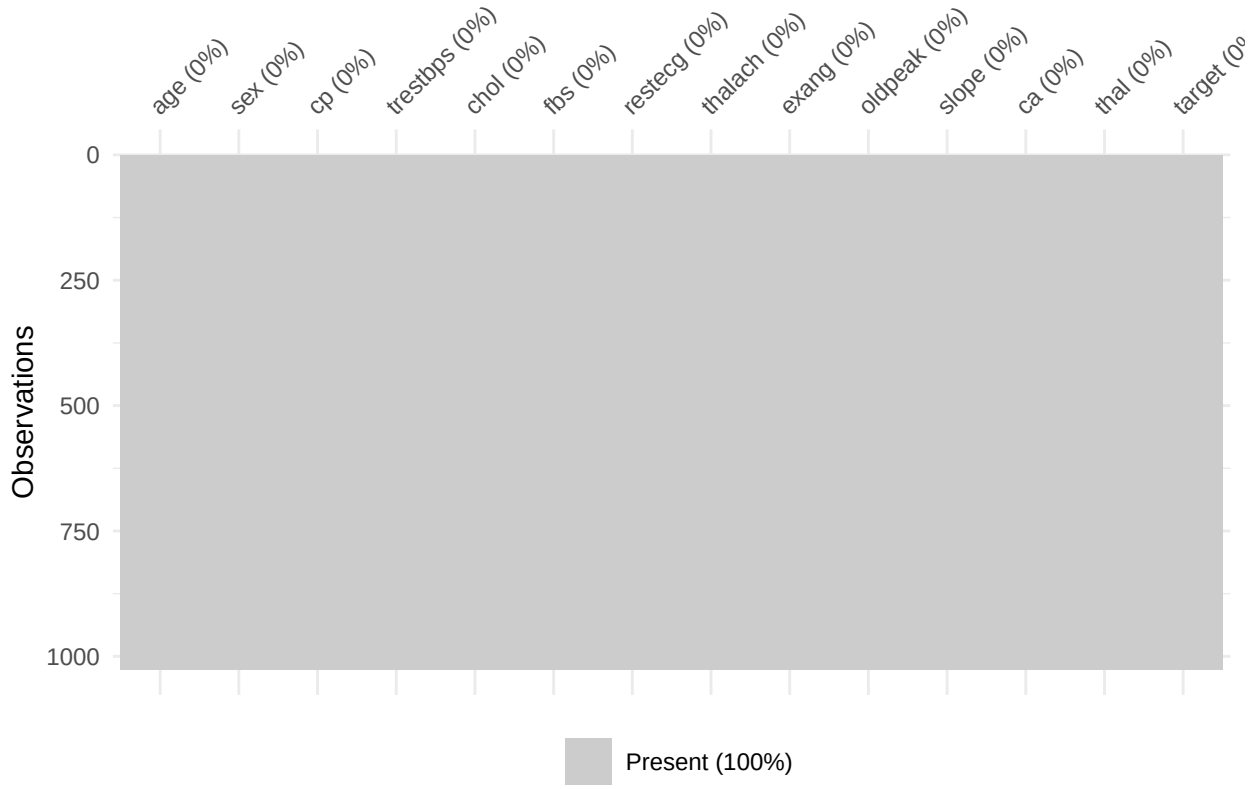


We can see, with limitations, some abnormal values on boxplots but in a low quantity. We can delete all abnormal values. But, for the moment, we can maintain them and go deeper into the exploration.

na values

Let's check if the data set contains missing values with a heatmap.

```
vis_miss(heart_data)
```



The data set doesn't contain any missing values. Very good !!

We can begin the processing.

Preprocessing

Let's change the type of the categorical variables to factor

```
for (i in c(2, 3, 6, 7, 9, 11, 12, 13, 14)){
  heart_data[, i] = as.factor(heart_data[, i])
}
```

Let's see if we obtain a good fit with a logistic regression model

We use a logistic regression model because the target `target` is a binary categorical variable.

```
summary(glm(target~., data = heart_data, family = binomial))
```

```
##
## Call:
## glm(formula = target ~ ., family = binomial, data = heart_data)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -2.8582  -0.2917   0.0718   0.4167   3.1908
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.081901   2.028691  -0.040 0.967797
## age          0.026846   0.013950   1.924 0.054297 .
## sex1        -1.992347   0.314204  -6.341 2.28e-10 ***
## cp1          0.886380   0.308803   2.870 0.004100 **
## cp2          2.006394   0.286281   7.008 2.41e-12 ***
## cp3          2.409722   0.391965   6.148 7.86e-10 ***
## trestbps    -0.024979   0.006537  -3.821 0.000133 ***
## chol        -0.005462   0.002307  -2.367 0.017914 *
## fbs1         0.380096   0.319620   1.189 0.234356
## restecg1     0.397268   0.217975   1.823 0.068374 .
## restecg2    -0.800417   1.536998  -0.521 0.602530
## thalach      0.021692   0.006525   3.324 0.000886 ***
## exang1      -0.750331   0.248746  -3.016 0.002557 **
## oldpeak     -0.403411   0.132156  -3.053 0.002269 **
## slope1      -0.595618   0.472076  -1.262 0.207057
## slope2       0.799689   0.504500   1.585 0.112941
## ca1         -2.334076   0.286781  -8.139 3.99e-16 ***
## ca2         -3.597039   0.444870  -8.086 6.19e-16 ***
## ca3         -2.288131   0.532138  -4.300 1.71e-05 ***
## ca4          1.565677   0.930256   1.683 0.092363 .
## thal1        2.796813   1.466219   1.908 0.056456 .
## thal2        2.404646   1.421542   1.692 0.090727 .
## thal3        0.991243   1.423972   0.696 0.486359
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1420.24  on 1024  degrees of freedom
## Residual deviance:  606.82  on 1002  degrees of freedom
## AIC: 652.82
##
## Number of Fisher Scoring iterations: 6
```

Some variables' p values are over 0.05. So they don't add much information to the model.

We can begin the selection using the likelihood ratio

```
# create a function calculating the likelihood ratio
LRT_dev = function(mod1, mod2){
  p_value =  1 - pchisq(deviance(mod1) - deviance(mod2), df.residual(mod1) - df.residual(mod2))
  result = p_value < 0.05
  print(paste("p_value is : ", p_value))
  print(paste("p_value < 0.05 : ", as.logical(p_value < 0.05)))
  return(result)
}
```

Forward selection with likelihood ratio

==========

Let's add a function that gives us a formula automatically

```
get_formula_add = function(target, variables){
 return(as.formula(                        # Create formula
  paste(paste(target, " ~ "), paste(variables, collapse = " + "))))
}
get_formula_rem = function(target, variables, all_variables){
 return(as.formula(                        # Create formula
  paste(paste(target, " ~ "), paste(paste(paste(all_variables, collapse = " + "), " - ")), paste(variabl
}
```

We can recuperate in a variable the names of the columns to test our function

```
variables = names(heart_data)[-length(heart_data)]
glm(get_formula_add("target", variables), data = heart_data, family = binomial)
```

```
##
## Call:  glm(formula = get_formula_add("target", variables), family = binomial,
##     data = heart_data)
##
## Coefficients:
## (Intercept)          age         sex1          cp1          cp2          cp3
##   -0.081901     0.026846    -1.992347     0.886380     2.006394     2.409722
##    trestbps         chol         fbs1     restecg1     restecg2      thalach
##   -0.024979    -0.005462     0.380096     0.397268    -0.800417     0.021692
##      exang1      oldpeak       slope1       slope2          ca1          ca2
##   -0.750331    -0.403411    -0.595618     0.799689    -2.334076    -3.597039
##         ca3          ca4        thal1        thal2        thal3
##   -2.288131     1.565677     2.796813     2.404646     0.991243
##
## Degrees of Freedom: 1024 Total (i.e. Null);  1002 Residual
## Null Deviance:       1420
## Residual Deviance: 606.8     AIC: 652.8
```

Our function goes well as we expected it.

Let's made a function that performs the LRT analysis directly.

```
process_LRT = function(target, variables, data, model, family, mode = "forward"){
  # We initialize the model to NULL.
  mod = NULL

  # if the chosen mode is forward we will process a forward selection.
```

```
if(mode == "forward"){

  # print out the title.
  print("Forward Selection :")
  print("--------------------------")

  # Let's get a model which uses as a feature a constant (that's the current model).
  mod = model(target~1, data = data, family = family)

  # We initialize an empty list that will stock the features to add to the model (we include those fe
  testing_variables = c()

  # We initialize the index of the first feature to add in testing_variables.
  j = 1

  # We iterate on each index of the list of variables to test.
  for(i in 1:length(variables)){

    # Add in testing_variables a new variable to test.
    testing_variables[j] = variables[i]

    # Include testing features in a new model.
    new_model = model(get_formula_add(target, testing_variables), data = data, family = family)

    # Print out the name of the variable to test
    print(paste("Variable ", variables[i]))

    # We verify if the variable to test is relevant or not with the LRT analysis.
    if(LRT_dev(mod, new_model) == 1){
      # If the variable is relevant
      # We increment the length of the testing variables to 1 for the following variable to test.
      j = j + 1

      # We recuperate the current model as the new model.
      mod = new_model

      # We can print out that the last tested variable is relevant.
      print(paste("The variable ", paste(variables[i], " is relevant")))
    }
    else{
      # If the variable is not relevant
      # We print out that the last tested variable is not relevant.
      print(paste("The variable ", paste(variables[i], " is not relevant")))
    }

    # Separate tests.
    print("==========================================================================================
  }
}
# if the chosen mode is backward we will process a backward selection.
else if(mode == "backward"){
  # print out the title.
  print("Backward Selection :")
```

```
    print("--------------------------")

    # Let's include all features in the current model.
    mod = model(get_formula_add(target, variables), data = data, family = family)

    # We initialize an empty list that will stock the features to remove from the model.
    testing_variables = c()

    # We initialize the index of the first feature to add in testing_variables.
    j = 1

    # We iterate on each index of the list of variables to test.
    for(i in 1:length(variables)){

      # Add in testing_variables a new variable to test.
      testing_variables[j] = variables[i]

      # Create a new model which does not include the testing features.
      new_model = model(get_formula_rem(target, testing_variables, variables), data = data, family = fa

      # Print out the name of the variable to test
      print(paste("Variable ", variables[i]))

      # We verify if the variable to test is relevant or not with the LRT analysis.
      if(LRT_dev(new_model, mod) == 0){
        # If the variable is not relevant
        # We increment the length of the testing variables to 1 for the following variable to test.
        j = j + 1

        # We recuperate the current model as the new model.
        mod = new_model

        # We can print out that the last tested variable is not relevant.
        print(paste("The variable ", paste(variables[i], " is not relevant")))
      }
      else{
        # If the variable is relevant
        # We print out that the last tested variable is relevant. So we don't remove the variable from
        print(paste("The variable ", paste(variables[i], " is relevant")))
      }
      # Separate tests.
      print("==========================================================================================
    }
  }
  # If the chosen mode doesn't exist, we raise an error.
  else{
    print(paste(paste("Error ! The choice ", mode), " doesn't exist"))
  }
  # For the forward, or the backward, we will finally return the final mode that contains the selected
  return(mod)
}
```

We can process the forward selection with the created function.

```
model_final_forw = process_LRT("target", variables, heart_data, glm, binomial, "forward")
```

```
## [1] "Forward Selection :"
## [1] "--------------------------"
## [1] "Variable  age"
## [1] "p_value is :  1.10245146345278e-13"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  age  is relevant"
## [1] "==========================================================================================================
## [1] "Variable  sex"
## [1] "p_value is :  0"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  sex  is relevant"
## [1] "==========================================================================================================
## [1] "Variable  cp"
## [1] "p_value is :  0"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  cp  is relevant"
## [1] "==========================================================================================================
## [1] "Variable  trestbps"
## [1] "p_value is :  0.000169102807049737"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  trestbps  is relevant"
## [1] "==========================================================================================================
## [1] "Variable  chol"
## [1] "p_value is :  0.0011530054348099"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  chol  is relevant"
## [1] "==========================================================================================================
## [1] "Variable  fbs"
## [1] "p_value is :  0.518289635253512"
## [1] "p_value < 0.05 :  FALSE"
## [1] "The variable  fbs  is not relevant"
## [1] "==========================================================================================================
## [1] "Variable  restecg"
## [1] "p_value is :  0.00375926672530991"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  restecg  is relevant"
## [1] "==========================================================================================================
## [1] "Variable  thalach"
## [1] "p_value is :  1.22124532708767e-15"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  thalach  is relevant"
## [1] "==========================================================================================================
## [1] "Variable  exang"
## [1] "p_value is :  7.35191679746006e-06"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  exang  is relevant"
## [1] "==========================================================================================================
## [1] "Variable  oldpeak"
## [1] "p_value is :  1.54321000422897e-14"
```

```
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  oldpeak  is relevant"
## [1] "================================================================================================"
## [1] "Variable  slope"
## [1] "p_value is :  0.000864762685505505"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  slope  is relevant"
## [1] "================================================================================================"
## [1] "Variable  ca"
## [1] "p_value is :  0"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  ca  is relevant"
## [1] "================================================================================================"
## [1] "Variable  thal"
## [1] "p_value is :  2.78633116579385e-10"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  thal  is relevant"
## [1] "================================================================================================"
```

Conclusion :

The final model doesn't contain the fbs variable.

```
# summary of the model
summary(model_final_forw)
```

```
##
## Call:
## model(formula = get_formula_add(target, testing_variables), family = family,
##     data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8702  -0.2991   0.0755   0.4399   3.1679
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.119738   2.107657  -0.057 0.954696
## age          0.026592   0.013945   1.907 0.056534 .
## sex1        -1.969469   0.313760  -6.277 3.45e-10 ***
## cp1          0.899337   0.307083   2.929 0.003404 **
## cp2          2.068496   0.282871   7.313 2.62e-13 ***
## cp3          2.453102   0.390858   6.276 3.47e-10 ***
## trestbps    -0.023878   0.006437  -3.710 0.000208 ***
## chol        -0.005238   0.002294  -2.284 0.022386 *
## restecg1     0.393543   0.218002   1.805 0.071039 .
## restecg2    -0.851297   1.525128  -0.558 0.576721
## thalach      0.022018   0.006525   3.374 0.000740 ***
## exang1      -0.723590   0.247712  -2.921 0.003488 **
## oldpeak     -0.416989   0.131778  -3.164 0.001554 **
## slope1      -0.618069   0.469275  -1.317 0.187815
## slope2       0.750442   0.500569   1.499 0.133828
## ca1         -2.281180   0.280992  -8.118 4.73e-16 ***
```

```
## ca2         -3.520334   0.436022  -8.074 6.82e-16 ***
## ca3         -2.192514   0.521209  -4.207 2.59e-05 ***
## ca4          1.719545   0.962903   1.786 0.074133 .
## thal1        2.684574   1.571897   1.708 0.087663 .
## thal2        2.234492   1.525980   1.464 0.143112
## thal3        0.811145   1.527764   0.531 0.595463
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1420.24  on 1024  degrees of freedom
## Residual deviance:  608.24  on 1003  degrees of freedom
## AIC: 652.24
##
## Number of Fisher Scoring iterations: 6
```

Let's process now the backward selection.

```
model_final_back = process_LRT("target", variables, heart_data, glm, binomial, "backward")
```

```
## [1] "Backward Selection :"
## [1] "--------------------------"
## [1] "Variable  age"
## [1] "p_value is :  0.0532387547007871"
## [1] "p_value < 0.05 :  FALSE"
## [1] "The variable  age  is not relevant"
## [1] "=============================================================================================
## [1] "Variable  sex"
## [1] "p_value is :  6.34337027349829e-12"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  sex  is relevant"
## [1] "=============================================================================================
## [1] "Variable  cp"
## [1] "p_value is :  4.44089209850063e-16"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  cp  is relevant"
## [1] "=============================================================================================
## [1] "Variable  trestbps"
## [1] "p_value is :  0.000464297725977225"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  trestbps  is relevant"
## [1] "=============================================================================================
## [1] "Variable  chol"
## [1] "p_value is :  0.0343792772062943"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  chol  is relevant"
## [1] "=============================================================================================
## [1] "Variable  fbs"
## [1] "p_value is :  0.243700583458427"
## [1] "p_value < 0.05 :  FALSE"
## [1] "The variable  fbs  is not relevant"
```

```
## [1] "=========================================================================================================
## [1] "Variable  restecg"
## [1] "p_value is :  0.191442156594169"
## [1] "p_value < 0.05 :  FALSE"
## [1] "The variable  restecg  is not relevant"
## [1] "=========================================================================================================
## [1] "Variable  thalach"
## [1] "p_value is :  0.00233122695649768"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  thalach  is relevant"
## [1] "=========================================================================================================
## [1] "Variable  exang"
## [1] "p_value is :  0.00307809092902378"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  exang  is relevant"
## [1] "=========================================================================================================
## [1] "Variable  oldpeak"
## [1] "p_value is :  0.000447998959877682"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  oldpeak  is relevant"
## [1] "=========================================================================================================
## [1] "Variable  slope"
## [1] "p_value is :  1.08118381503264e-06"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  slope  is relevant"
## [1] "=========================================================================================================
## [1] "Variable  ca"
## [1] "p_value is :  0"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  ca  is relevant"
## [1] "=========================================================================================================
## [1] "Variable  thal"
## [1] "p_value is :  8.00956745372616e-10"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  thal  is relevant"
## [1] "=========================================================================================================
```

conclusion :

The final model doesn't contain the following variables : age, fbs and restecg.

```
# summary of the model
summary(model_final_back)
```

```
##
## Call:
## model(formula = get_formula_rem(target, testing_variables, variables),
##     family = family, data = data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8426  -0.3117   0.0821   0.4410   3.2375
##
```

```
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.759212   2.125365   0.828 0.407828
## sex1        -2.013263   0.309750  -6.500 8.05e-11 ***
## cp1          0.954620   0.306490   3.115 0.001841 **
## cp2          2.109382   0.282959   7.455 9.01e-14 ***
## cp3          2.442079   0.384484   6.352 2.13e-10 ***
## trestbps    -0.021768   0.005975  -3.643 0.000269 ***
## chol        -0.005319   0.002167  -2.454 0.014113 *
## thalach      0.017998   0.006114   2.944 0.003242 **
## exang1      -0.729760   0.246122  -2.965 0.003027 **
## oldpeak     -0.435713   0.127416  -3.420 0.000627 ***
## slope1      -0.593005   0.460981  -1.286 0.198304
## slope2       0.770110   0.491292   1.568 0.116993
## ca1         -2.224647   0.273918  -8.122 4.60e-16 ***
## ca2         -3.270601   0.413742  -7.905 2.68e-15 ***
## ca3         -2.141769   0.537328  -3.986 6.72e-05 ***
## ca4          1.581474   0.881925   1.793 0.072940 .
## thal1        2.800038   1.787780   1.566 0.117300
## thal2        2.304987   1.747861   1.319 0.187254
## thal3        0.948599   1.748739   0.542 0.587510
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1420.24  on 1024  degrees of freedom
## Residual deviance:  615.22  on 1006  degrees of freedom
## AIC: 653.22
##
## Number of Fisher Scoring iterations: 6
```

Let's make variables selection with the stepwise trick

The stepwise function (stepAIC) takes, as main parameters, the model (logistic) and the direction that can be forward, backward, or both. We can choose both directions to obtain the best possible model. The time complexity of the stepwise method is very high but we will finally obtain a good selection.

```
model = glm(target~., data = heart_data, family = binomial)
step.model = stepAIC(model, direction = "both")
```

```
## Start:  AIC=652.82
## target ~ age + sex + cp + trestbps + chol + fbs + restecg + thalach +
##     exang + oldpeak + slope + ca + thal
##
##            Df Deviance    AIC
## - fbs       1   608.24 652.24
## - restecg   2   610.59 652.59
## <none>          606.82 652.82
## - age       1   610.55 654.55
## - chol      1   612.27 656.27
## - exang     1   615.88 659.88
## - oldpeak   1   616.56 660.56
```

```
## - thalach   1   618.58 662.58
## - trestbps  1   622.06 666.06
## - slope     2   634.80 676.80
## - thal      3   653.01 693.01
## - sex       1   652.38 696.38
## - cp        3   679.09 719.09
## - ca        4   747.81 785.81
##
## Step:  AIC=652.24
## target ~ age + sex + cp + trestbps + chol + restecg + thalach +
##     exang + oldpeak + slope + ca + thal
##
##            Df Deviance    AIC
## - restecg   2   612.01 652.01
## <none>          608.24 652.24
## + fbs       1   606.82 652.82
## - age       1   611.91 653.91
## - chol      1   613.31 655.31
## - exang     1   616.75 658.75
## - oldpeak   1   618.75 660.75
## - thalach   1   620.37 662.37
## - trestbps  1   622.53 664.53
## - slope     2   635.44 675.44
## - thal      3   655.70 693.70
## - sex       1   652.92 694.92
## - cp        3   686.09 724.09
## - ca        4   748.43 784.43
##
## Step:  AIC=652.01
## target ~ age + sex + cp + trestbps + chol + thalach + exang +
##     oldpeak + slope + ca + thal
##
##            Df Deviance    AIC
## <none>          612.01 652.01
## + restecg   2   608.24 652.24
## + fbs       1   610.59 652.59
## - age       1   615.22 653.22
## - chol      1   618.87 656.87
## - exang     1   620.31 658.31
## - oldpeak   1   622.98 660.98
## - thalach   1   624.19 662.19
## - trestbps  1   628.65 666.65
## - slope     2   640.87 676.87
## - thal      3   657.38 691.38
## - sex       1   658.37 696.37
## - cp        3   690.25 724.25
## - ca        4   754.60 786.60
```

```
# summary of the stepwise model
summary(step.model)
```

```
##
## Call:
## glm(formula = target ~ age + sex + cp + trestbps + chol + thalach +
```

```
##     exang + oldpeak + slope + ca + thal, family = binomial, data = heart_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8172  -0.3056   0.0789   0.4425   3.2446
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.432526   2.194867   0.197 0.843778
## age          0.024750   0.013866   1.785 0.074273 .
## sex1        -1.987341   0.311799  -6.374 1.84e-10 ***
## cp1          0.925378   0.307153   3.013 0.002589 **
## cp2          2.086274   0.282856   7.376 1.63e-13 ***
## cp3          2.424310   0.387894   6.250 4.11e-10 ***
## trestbps    -0.025043   0.006309  -3.969 7.21e-05 ***
## chol        -0.005976   0.002244  -2.663 0.007740 **
## thalach      0.022119   0.006561   3.371 0.000748 ***
## exang1      -0.714079   0.247315  -2.887 0.003885 **
## oldpeak     -0.414750   0.128417  -3.230 0.001239 **
## slope1      -0.607436   0.461220  -1.317 0.187832
## slope2       0.795311   0.491264   1.619 0.105467
## ca1         -2.321347   0.280060  -8.289  < 2e-16 ***
## ca2         -3.468958   0.429465  -8.077 6.62e-16 ***
## ca3         -2.272658   0.534734  -4.250 2.14e-05 ***
## ca4          1.682084   0.917953   1.832 0.066887 .
## thal1        2.761054   1.715714   1.609 0.107556
## thal2        2.253987   1.674329   1.346 0.178237
## thal3        0.892851   1.675488   0.533 0.594110
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1420.24  on 1024  degrees of freedom
## Residual deviance:  612.01  on 1005  degrees of freedom
## AIC: 652.01
##
## Number of Fisher Scoring iterations: 6
```

Conclusion of the stepwise selection :

The variables that the model decided to use for his training are age, sex, cp, trestbps, chol, thalach, exang, oldpeak, slope, ca, and thal. So only the fbs and restecg variables didn't be chosen.

Make a selection with the random forest classification model :

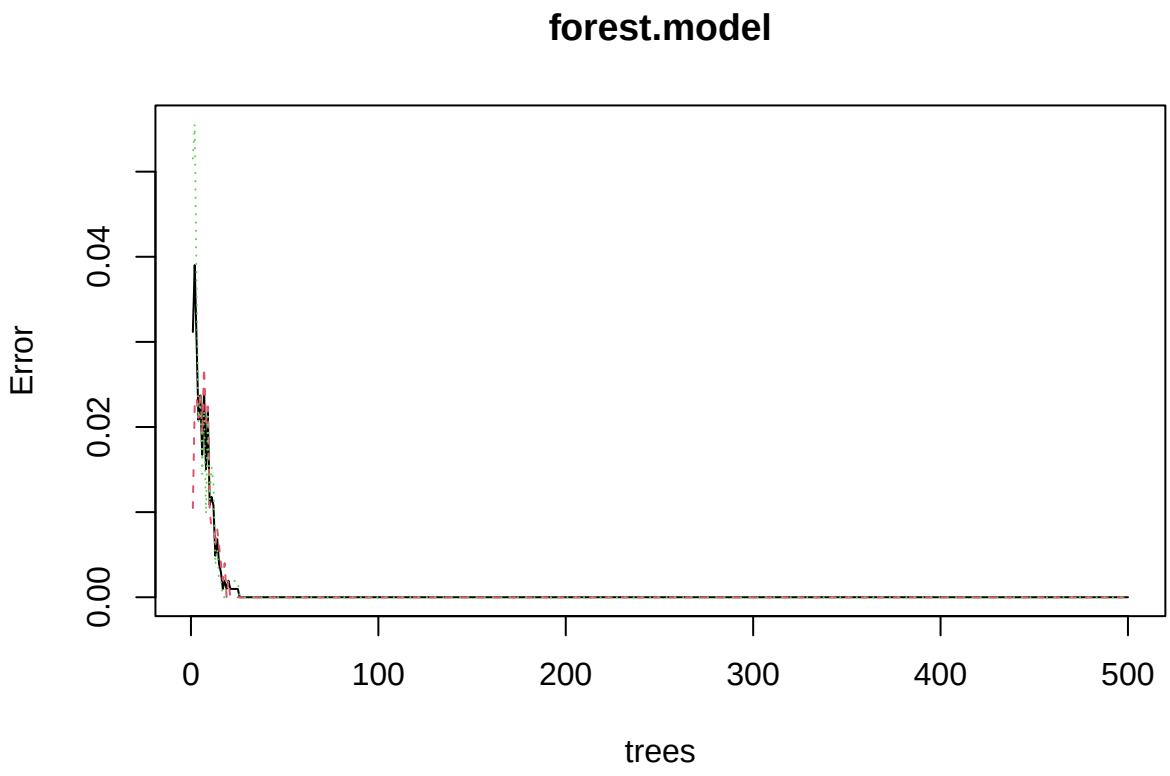
This method selects the best variables by using the random forest classification model. The random forest model is known to be a non-parametrical model.

```
forest.model = randomForest(target~., data = heart_data, importance = TRUE)
```

```
summary(forest.model)
```

```
##                 Length Class  Mode
## call               4   -none- call
## type               1   -none- character
## predicted       1025   factor numeric
## err.rate        1500   -none- numeric
## confusion          6   -none- numeric
## votes           2050   matrix numeric
## oob.times       1025   -none- numeric
## classes            2   -none- character
## importance        52   -none- numeric
## importanceSD      39   -none- numeric
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            14   -none- list
## y               1025   factor numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
## terms              3   terms  call
```

```
# plot the OOB errors
plot(forest.model)
```



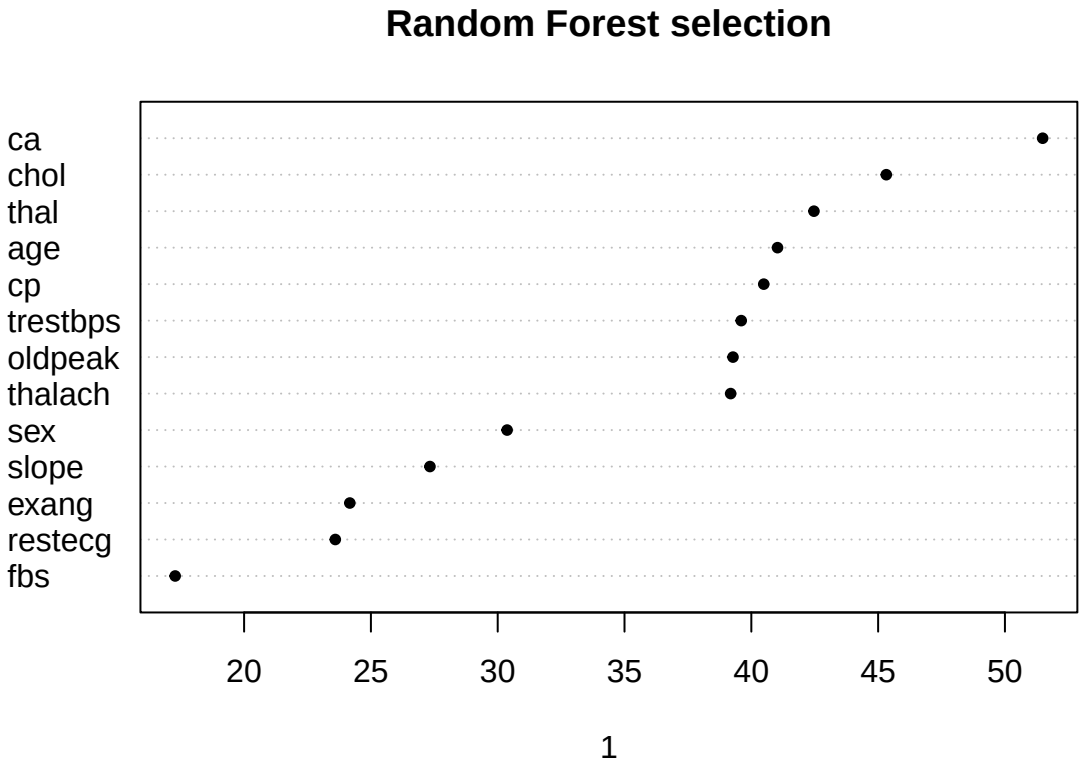
Here we see that we can obtain a good fit only with 90 trees.

Let's make another random forest model with only 90 trees.

```
forest.model = randomForest(target~., data = heart_data, importance = TRUE, ntrees = 90)
```

Let's plot the classification of the variables

```
varImpPlot(forest.model, type = 1, c = 1, pch = 20, cex = 1, main = "Random Forest selection")
```



The most relevant variables are the caa, thall, sex, cp, oldpeak and thalachh variables that we previously selected with the other methods.

With the VSURF package we will interprate the selection and visualize the best choices.

```
y = heart_data[, 14]
x = heart_data[, -14]
th1 = VSURF_thres(x, y, ntree = 90)
```

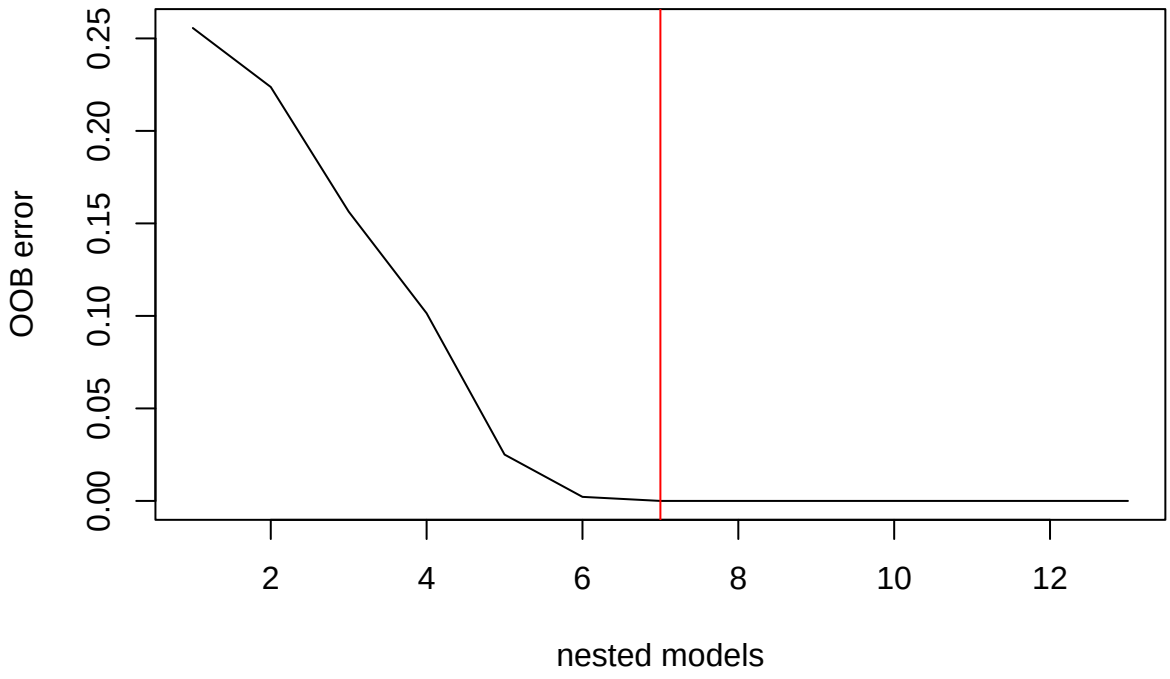
```
## Thresholding step
## Estimated computational time (on one core): 6 sec.
##   |                                                                      |
```

```
vs1 = VSURF_interp(x, y, vars = th1$varselect.thres)
```

```
##
## Interpretation step (on 13 variables)
## Estimated computational time (on one core): between 35.8 sec. and  91 sec.
##   |                                                                              |
```

Let's visualize the number of choosed variables

```
plot(vs1)
```



The model chose the 7 best variables finally (the number can change over the training we can choose a seed to fix it). Let's print out the names of those variables.

```
names(x)[vs1$varselect.interp]
```

```
## [1] "ca"      "thal"    "cp"      "oldpeak" "thalach" "age"     "chol"
```

With the randomForest model, the following variables are not relevant: sex, slope, exang, restecg, trestbps, and fbs.

Conclusion: With the random forest we found many not relevant variables and with the stepwise, the not relevant variables were fbs and restecg. We must, after that, verify if all those variables are actually not relevant with the help of the LRT analysis.

We created a function that can process the likelihood ratio on a full data set. But we can also create a function that can process the likelihood ratio on a few variables only. We must, before that, maintain some variables as relevant and we will investigate, after that, the other variables to check if they are relevant or not. Let's create the function and test it with the not relevant variables.

Match function.

The match function will be made like the process LRT function but with different arguments

```
match_variables_LRT = function(target, relevant_variables, variables_to_check, data, model, family){
  print("Let's check if the provided variables are important :")
  print("---------------------------")
  mod = model(get_formula_add(target, relevant_variables), data = data, family = family)
  testing_variables = relevant_variables
  j = length(relevant_variables) + 1
  for(i in 1:length(variables_to_check)){
      testing_variables[j] = variables_to_check[i]
      new_model = model(get_formula_add(target, testing_variables), data = data, family = family)
      print(paste("Variable ", variables_to_check[i]))
      if(LRT_dev(mod, new_model) == 1){
        print(paste("The variable ", paste(variables_to_check[i], " is relevant")))
      }
      else{
        print(paste("The variable ", paste(variables_to_check[i], " is not relevant")))
      }
      print("=============================================================================================
  }
}
```

Test some variables with the match function

The variables to test with the match function are listed as following : 'sex', 'slope', 'exang', 'restecg', 'trestbps', 'fbs'. Another variables will be considerate as relevant.

```
# recuperate before that the relevant variables
variables_to_exclude = c('target', 'sex', 'slope', 'exang', 'restecg', 'trestbps', 'fbs')
relevant_variables = names(heart_data[,-which(names(heart_data) %in% variables_to_exclude)])
variables_to_check = variables_to_exclude[-1]
match_variables_LRT("target", relevant_variables, variables_to_check, heart_data, glm, "binomial")
```

```
## [1] "Let's check if the provided variables are important :"
## [1] "---------------------------"
## [1] "Variable  sex"
## [1] "p_value is :  1.5627596661183e-08"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  sex  is relevant"
## [1] "=============================================================================================
## [1] "Variable  slope"
## [1] "p_value is :  3.09368004658417e-05"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  slope  is relevant"
## [1] "=============================================================================================
```

```
## [1] "Variable  exang"
## [1] "p_value is :  0.00157931497192321"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  exang  is relevant"
## [1] "================================================================================================
## [1] "Variable  restecg"
## [1] "p_value is :  0.0115851831950611"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  restecg  is relevant"
## [1] "================================================================================================
## [1] "Variable  trestbps"
## [1] "p_value is :  0.000812258829805845"
## [1] "p_value < 0.05 :  TRUE"
## [1] "The variable  trestbps  is relevant"
## [1] "================================================================================================
## [1] "Variable  fbs"
## [1] "p_value is :  0.767339118128731"
## [1] "p_value < 0.05 :  FALSE"
## [1] "The variable  fbs  is not relevant"
## [1] "================================================================================================
```

Conclusion: The not relevant variable will be finally: fbs. We can create the final model which will contain only the relevant variables.

final model

```
# update the relevant variables
variables_to_exclude = c('target', 'fbs')
relevant_variables = names(heart_data[,-which(names(heart_data) %in% variables_to_exclude)])
final_model = glm(get_formula_add("target", relevant_variables), data = heart_data, family = "binomial")
print(final_model)
```

```
##
## Call:  glm(formula = get_formula_add("target", relevant_variables),
##     family = "binomial", data = heart_data)
##
## Coefficients:
## (Intercept)          age         sex1          cp1          cp2          cp3
##   -0.119738     0.026592    -1.969469     0.899337     2.068496     2.453102
##    trestbps         chol     restecg1     restecg2      thalach       exang1
##   -0.023878    -0.005238     0.393543    -0.851297     0.022018    -0.723590
##     oldpeak       slope1       slope2          ca1          ca2          ca3
##   -0.416989    -0.618069     0.750442    -2.281180    -3.520334    -2.192514
##         ca4        thal1        thal2        thal3
##    1.719545     2.684574     2.234492     0.811145
##
## Degrees of Freedom: 1024 Total (i.e. Null);  1003 Residual
## Null Deviance:       1420
## Residual Deviance: 608.2     AIC: 652.2
```

We got a final model but we must split the dataset into a training set et a testing set if we want to make good predictions.

##Sampling: Learning vs Testing

We can now try to split the dataset into training set and testing set.

```
heart_data[, 1:13] = heart_data[, which(names(heart_data) %in% relevant_variables)]
set.seed(111)
d = sort(sample(nrow(heart_data), nrow(heart_data) * 0.7))
```

Training sample

```
appren <- heart_data[d, ]
summary(appren)
```

```
##       age        sex     cp         trestbps          chol       fbs
##  Min.   :29.00   0:216   0:351   Min.   : 94.0   Min.   :126.0   0:353
##  1st Qu.:48.00   1:501   1:120   1st Qu.:120.0   1st Qu.:211.0   1:354
##  Median :56.00           2:191   Median :130.0   Median :239.0   2: 10
##  Mean   :54.71           3: 55   Mean   :131.8   Mean   :246.4
##  3rd Qu.:61.00                   3rd Qu.:140.0   3rd Qu.:276.0
##  Max.   :77.00                   Max.   :200.0   Max.   :564.0
##     restecg      thalach     exang       oldpeak slope   ca
##  Min.   : 71.0   0:472   Min.   :0.000   0: 55   0:405   0:  4
##  1st Qu.:132.0   1:245   1st Qu.:0.000   1:346   1:154   1: 46
##  Median :151.0           Median :0.800   2:316   2: 97   2:369
##  Mean   :148.2           Mean   :1.105           3: 50   3:298
##  3rd Qu.:165.0           3rd Qu.:1.800           4: 11
##  Max.   :202.0           Max.   :6.200
##       thal       target
##  Min.   :29.00   0:363
##  1st Qu.:48.00   1:354
##  Median :56.00
##  Mean   :54.71
##  3rd Qu.:61.00
##  Max.   :77.00
```

Test sample

```
test <- heart_data[-d, ]
summary(test)
```

```
##       age        sex     cp         trestbps          chol       fbs
##  Min.   :29.00   0: 96   0:146   Min.   : 94.0   Min.   :126.0   0:144
##  1st Qu.:46.00   1:212   1: 47   1st Qu.:120.0   1st Qu.:211.0   1:159
##  Median :54.00           2: 93   Median :130.0   Median :241.5   2:  5
##  Mean   :53.79           3: 22   Mean   :131.1   Mean   :245.1
##  3rd Qu.:61.00                   3rd Qu.:140.0   3rd Qu.:274.0
##  Max.   :76.00                   Max.   :200.0   Max.   :417.0
##     restecg      thalach     exang       oldpeak slope   ca
##  Min.   : 71.0   0:208   Min.   :0.0000   0: 19   0:173   0:  3
##  1st Qu.:140.8   1:100   1st Qu.:0.0000   1:136   1: 72   1: 18
##  Median :155.0           Median :0.8000   2:153   2: 37   2:175
##  Mean   :151.2           Mean   :0.9925           3: 19   3:112
##  3rd Qu.:169.0           3rd Qu.:1.6000           4:  7
```

```
##  Max.   :202.0           Max.   :6.2000
##       thal       target
##  Min.   :29.00   0:136
##  1st Qu.:46.00   1:172
##  Median :54.00
##  Mean   :53.79
##  3rd Qu.:61.00
##  Max.   :76.00
```

##Modelization

The model used here is the logistic regression which can be found in R as glm.

We introduce all the variables in our model except fbs

```
get_formula_add("target", relevant_variables)
```

```
## target ~ age + sex + cp + trestbps + chol + restecg + thalach +
##     exang + oldpeak + slope + ca + thal
## <environment: 0x0000000029c5c420>
```

```
m.logit <- glm(get_formula_add("target", relevant_variables), data = appren, family = binomial)

summary(m.logit)
```

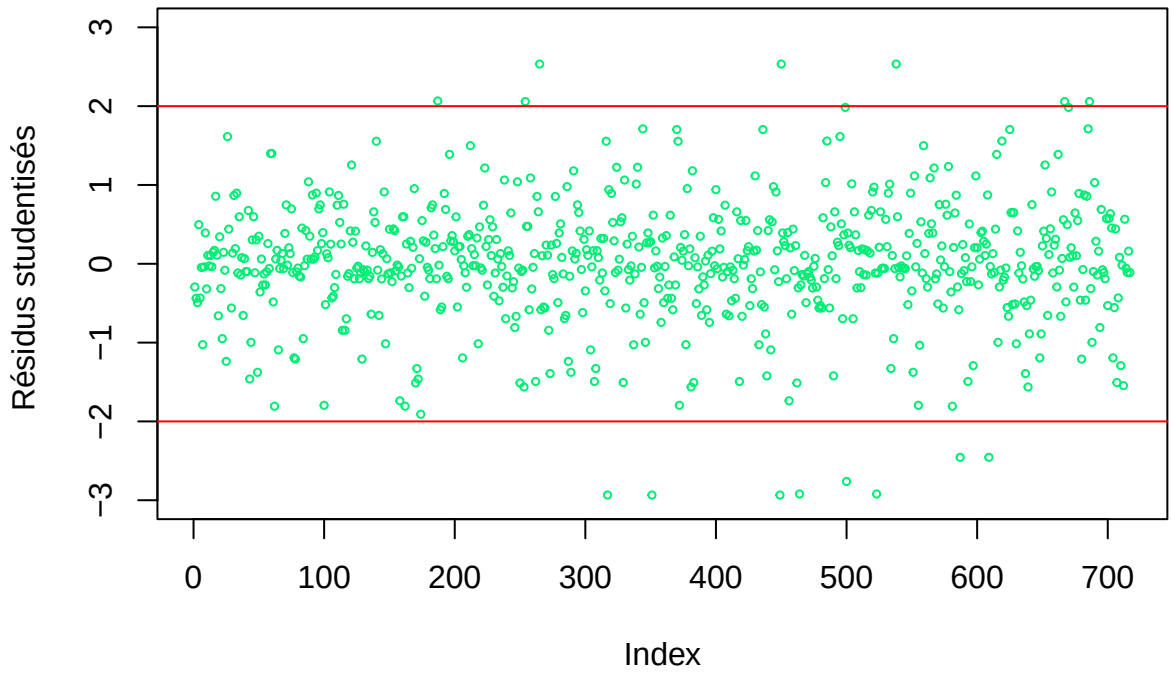
```
##
## Call:
## glm(formula = get_formula_add("target", relevant_variables),
##     family = binomial, data = appren)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.8930  -0.2946  -0.0175   0.3889   3.5742
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.408053   3.135008  -0.130 0.896440
## age          0.036586   0.017273   2.118 0.034165 *
## sex1        -2.188142   0.402166  -5.441 5.30e-08 ***
## cp1          0.752574   0.369164   2.039 0.041491 *
## cp2          2.201779   0.367993   5.983 2.19e-09 ***
## cp3          2.674045   0.491954   5.436 5.46e-08 ***
## trestbps    -0.027040   0.007663  -3.528 0.000418 ***
## chol        -0.005019   0.002880  -1.743 0.081415 .
## restecg      0.028007   0.008425   3.324 0.000886 ***
## thalach1    -1.303677   0.313884  -4.153 3.28e-05 ***
## exang       -0.384293   0.159690  -2.406 0.016106 *
## oldpeak1    -0.695942   0.568925  -1.223 0.221233
## oldpeak2     0.785125   0.617305   1.272 0.203423
## slope1      -2.714364   0.357412  -7.594 3.09e-14 ***
## slope2      -3.782429   0.557062  -6.790 1.12e-11 ***
## slope3      -2.213234   0.623017  -3.552 0.000382 ***
## slope4       2.211206   1.200693   1.842 0.065533 .
```

```
## ca1          2.780465   2.630672   1.057 0.290539
## ca2          1.941985   2.589566   0.750 0.453299
## ca3          0.500167   2.590777   0.193 0.846914
## thal               NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 993.86  on 716  degrees of freedom
## Residual deviance: 396.24  on 697  degrees of freedom
## AIC: 436.24
##
## Number of Fisher Scoring iterations: 6
```

##Model validation: Quality and robustness indicators

After obtaining a model, it is necessary to diagnose the regression in order to validate or not the model.

```
par(mfrow = c(1, 1))
plot(rstudent(m.logit), type = "p", cex = 0.5, ylab = "Résidus studentisés ",
    col = "springgreen2", ylim = c(-3, 3))
abline(h = c(-2, 2), col = "red")
```



```
(chi2 <- with(m.logit, null.deviance - deviance))
```

```
## [1] 597.6195
```

```
(ddl <- with(m.logit, df.null - df.residual))
```

```
## [1] 19
```

```
(pvalue <- pchisq(chi2, ddl, lower.tail = F))
```

```
## [1] 1.604395e-114
```

We are now going to try to validate on the test sample that we have previously defined

Here are the steps we will follow to validate our model

On the training sample and on the test sample:

We calculate a confusion matrix: and therefore we measure an error rate the air is evaluated under the ROC curve

```
appren.p <- cbind(appren, predict(m.logit, newdata = appren, type = "link",
    se = TRUE))
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = residual.scale, type = if
## (type == : prediction from a rank-deficient fit may be misleading
```

```
head(appren.p)
```

```
##   age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 3  70   1  0      145  174   1     125       1   2.6       0     0  3   70
## 4  61   1  0      148  203   1     161       0   0.0       2     1  3   61
## 5  62   0  0      138  294   1     106       0   1.9       1     3  2   62
## 6  58   0  0      100  248   0     122       0   1.0       1     0  2   58
## 7  58   1  0      114  318   2     140       0   4.4       0     3  1   58
## 8  55   1  0      160  289   0     145       1   0.8       1     1  3   55
##   target       fit    se.fit residual.scale
## 3      0 -3.131066 0.7015677              1
## 4      0 -2.305155 0.4636912              1
## 5      0 -2.075381 0.7232811              1
## 6      1  2.043871 0.4724677              1
## 7      0 -2.355427 0.9168720              1
## 8      0 -6.821050 0.6639242              1
```

```
appren.p <- within(appren.p, {
    PredictedProb <- plogis(fit)
    LL <- plogis(fit - (1.96 * se.fit))
    UL <- plogis(fit + (1.96 * se.fit))
})
tail(appren.p)
```

```
##      age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 1019  41   1  0      110  172   0     158       0   0.0       2     0  3   41
## 1020  47   1  0      112  204   1     143       0   0.1       2     0  2   47
## 1022  60   1  0      125  258   0     141       1   2.8       1     1  3   60
## 1023  47   1  0      110  275   0     118       1   1.0       1     1  2   47
## 1024  50   0  0      110  254   0     159       0   0.0       2     0  2   50
## 1025  54   1  0      120  188   1     113       0   1.4       1     1  3   54
##      target        fit    se.fit residual.scale          UL           LL
## 1019      0  0.7765746 0.4058239              1 0.828067020 0.4952901025
## 1020      1  1.7646972 0.3976772              1 0.927179146 0.7281486682
## 1022      0 -6.4167461 0.6345284              1 0.005635206 0.0004708868
## 1023      0 -5.0826933 0.6618586              1 0.022194844 0.0016923623
## 1024      1  4.3522825 0.4981302              1 0.995172646 0.9669443156
## 1025      0 -5.0922584 0.6090823              1 0.019870331 0.0018586219
##      PredictedProb
## 1019   0.684941393
## 1020   0.853796971
## 1022   0.001631299
## 1023   0.006164938
## 1024   0.987286333
## 1025   0.006106609
```

```
appren.p <- cbind(appren.p, pred.target = factor(ifelse(appren.p$PredictedProb >
    0.5, 1, 0)))
head(appren.p)
```

```
##   age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal
## 3  70   1  0      145  174   1     125       1   2.6       0     0  3   70
## 4  61   1  0      148  203   1     161       0   0.0       2     1  3   61
## 5  62   0  0      138  294   1     106       0   1.9       1     3  2   62
## 6  58   0  0      100  248   0     122       0   1.0       1     0  2   58
## 7  58   1  0      114  318   2     140       0   4.4       0     3  1   58
## 8  55   1  0      160  289   0     145       1   0.8       1     1  3   55
##   target       fit    se.fit residual.scale          UL           LL
## 3      0 -3.131066 0.7015677              1 0.147292892 0.0109204398
## 4      0 -2.305155 0.4636912              1 0.198400644 0.0386426338
## 5      0 -2.075381 0.7232811              1 0.341245158 0.0295117155
## 6      1  2.043871 0.4724677              1 0.951195997 0.7535867829
## 7      0 -2.355427 0.9168720              1 0.363927479 0.0154816609
## 8      0 -6.821050 0.6639242              1 0.003990823 0.0002967446
##   PredictedProb pred.target
## 3   0.041843855           0
## 4   0.090696903           0
## 5   0.111512771           0
## 6   0.885326857           1
## 7   0.086635358           0
## 8   0.001089387           0
```

```
(m.confusion <- as.matrix(table(appren.p$pred.target, appren.p$target)))
```

```
##
##       0   1
##   0 313  34
##   1  50 320
```

```
m.confusion <- unclass(m.confusion)
# Taux d'erreur
Tx_err <- function(y, ypred) {
    mc <- table(y, ypred)
    error <- (mc[1, 2] + mc[2, 1])/sum(mc)
    print(error)
}
Tx_err(appren.p$pred.target, appren.p$target)
```

```
## [1] 0.1171548
```

calculation of the error rate on the training sample

```
test.p <- cbind(test, predict(m.logit, newdata = test, type = "response", se = TRUE))
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = residual.scale, type = if
## (type == : prediction from a rank-deficient fit may be misleading
```

```
test.p <- cbind(test.p, pred.target <- factor(ifelse(test.p$fit > 0.5, 1, 0)))
(m.confusiontest <- as.matrix(table(test.p$pred.target, test.p$target)))
```

```
##
##       0   1
##   0 111  18
##   1  25 154
```

calculation of the error rate on the test sample

```
m.confusiontest <- unclass(m.confusiontest)

Tx_err(test.p$pred.target, test.p$target)
```

```
## [1] 0.1396104
```

Construction of ROC curves:

This curve, or rather the area under it, represents the sensitivity/specificity of the model. A model is good if positives (1) were predicted positives and 0 were predicted 0. Generally, we are interested in both the shape of the curve and the area under it: 1–> Ideal model, 0.5 –> Random model;

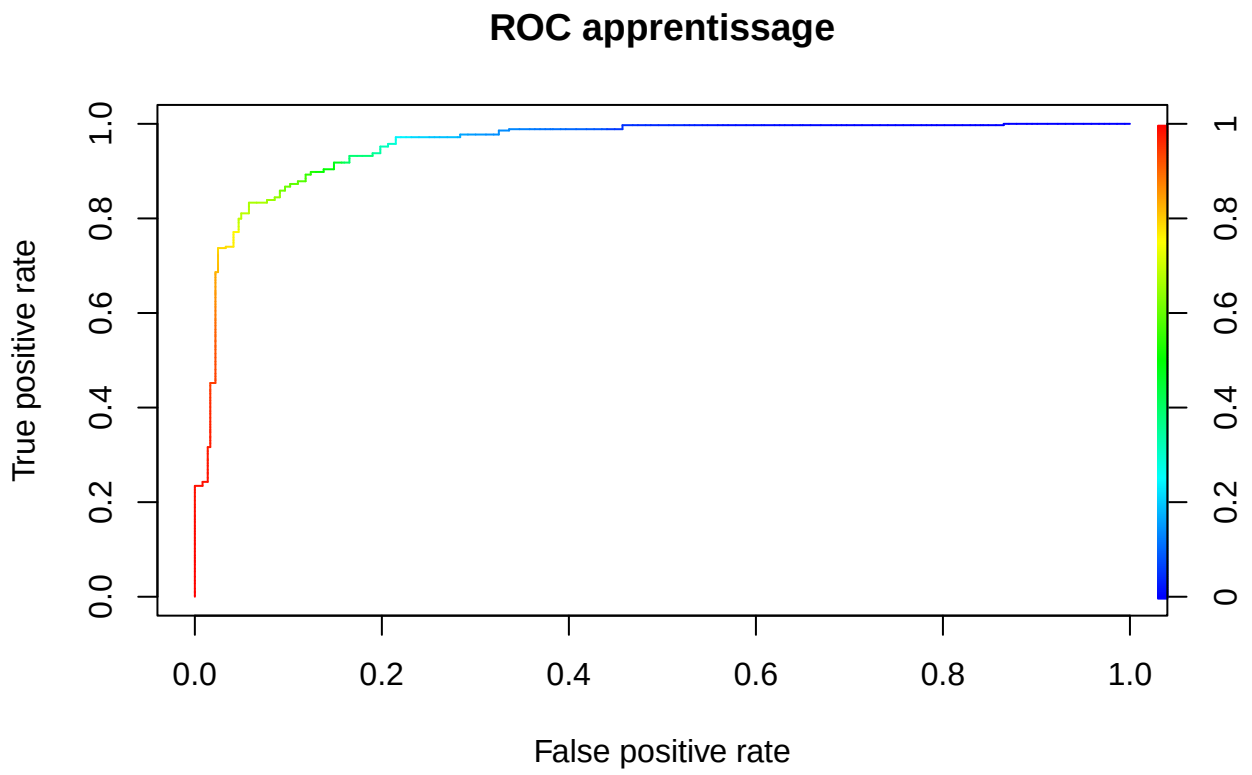
Principle of the ROC curve: if the test gives a numerical result with a threshold t such that the prediction is positive if x > t, and the prediction is negative if x < t, then as t increases:

the specificity increases. but the sensitivity decreases. The ROC curve represents the change in sensitivity (rate of true positives) as a function of 1 - specificity (rate of false positives) when the threshold t is varied.

It is an increasing curve between the point (0,0) and the point (1, 1) and in principle above the first bisector. A random prediction would give the first bisector. The better the prediction, the more the curve is above

the first bisector. An ideal prediction is the horizontal y=1 on]0,1] and the point (0,0). The area under the ROC curve (AUC, Area Under the Curve) gives an indicator of the quality of the prediction (1 for an ideal prediction, 0.5 for a random prediction).

```
Pred = prediction(appren.p$PredictedProb, appren.p$target)
Perf = performance(Pred, "tpr", "fpr")
plot(Perf, colorize = TRUE, main = "ROC apprentissage")
```




```
perf <- performance(Pred, "auc")
perf@y.values[[1]]
```

```
## [1] 0.9541719
```

To get the area under the curve, we will rather use

```
Predtest = prediction(test.p$fit, test.p$target)
Perftest = performance(Predtest, "tpr", "fpr")
perftest <- performance(Predtest, "auc")
perftest@y.values[[1]]
```

```
## [1] 0.9203574
```

We will make the roc curve with each set (training and testing) and we will verify if the model is overfitted.

```
par(mfrow = c(1, 2))
plot(Perf, colorize = TRUE, main = "Training ROC - AUC= 0.95")
plot(Perftest, colorize = TRUE, main = "Testing ROC - AUC = 0.92 ")
```