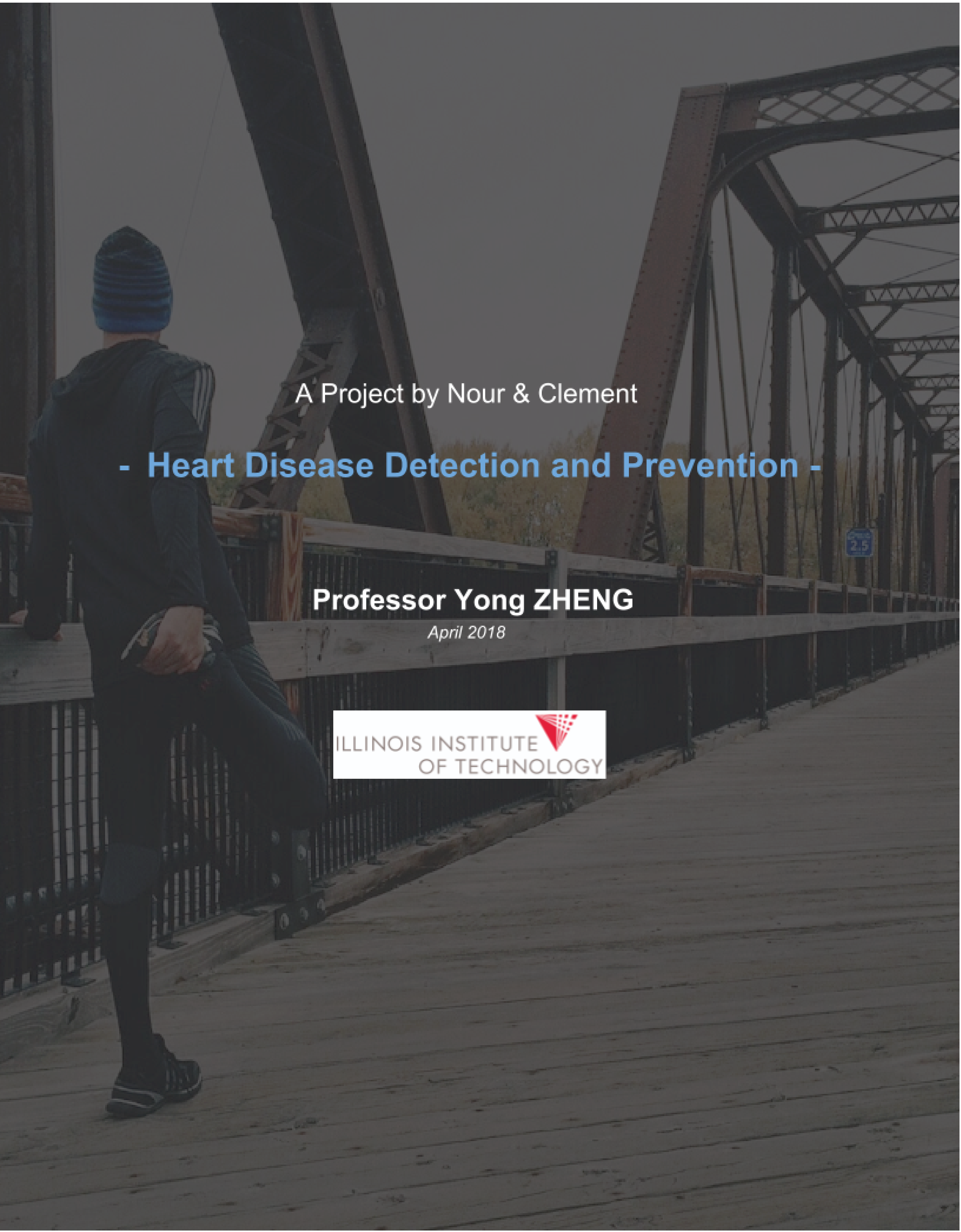


A Project by Nour & Clement

- Heart Disease Detection and Prevention -

Professor Yong ZHENG

April 2018

ILLINOIS INSTITUTE OF TECHNOLOGY

Agenda

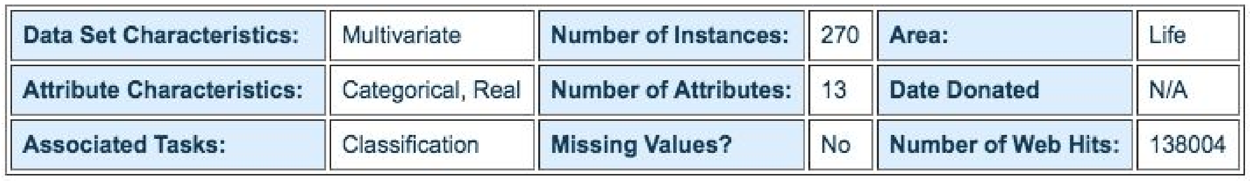
Nour Al-Habash
Clement Francois Dit Hardy

ILLINOIS INSTITUTE OF TECHNOLOGY

1. The data

1.1. Data overview

We have found our dataset on https://archive.ics.uci.edu/ml/index.php . Our dataset is linked to the medical field, in particular, it overlooks various factors that makes an individual more susceptible to developing a heart conditions or not.



Data Set Characteristics:	Multivariate	**Number of Instances:**	270	**Area:**	Life
Attribute Characteristics:	Categorical, Real	**Number of Attributes:**	13	**Date Donated**	N/A
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	138004

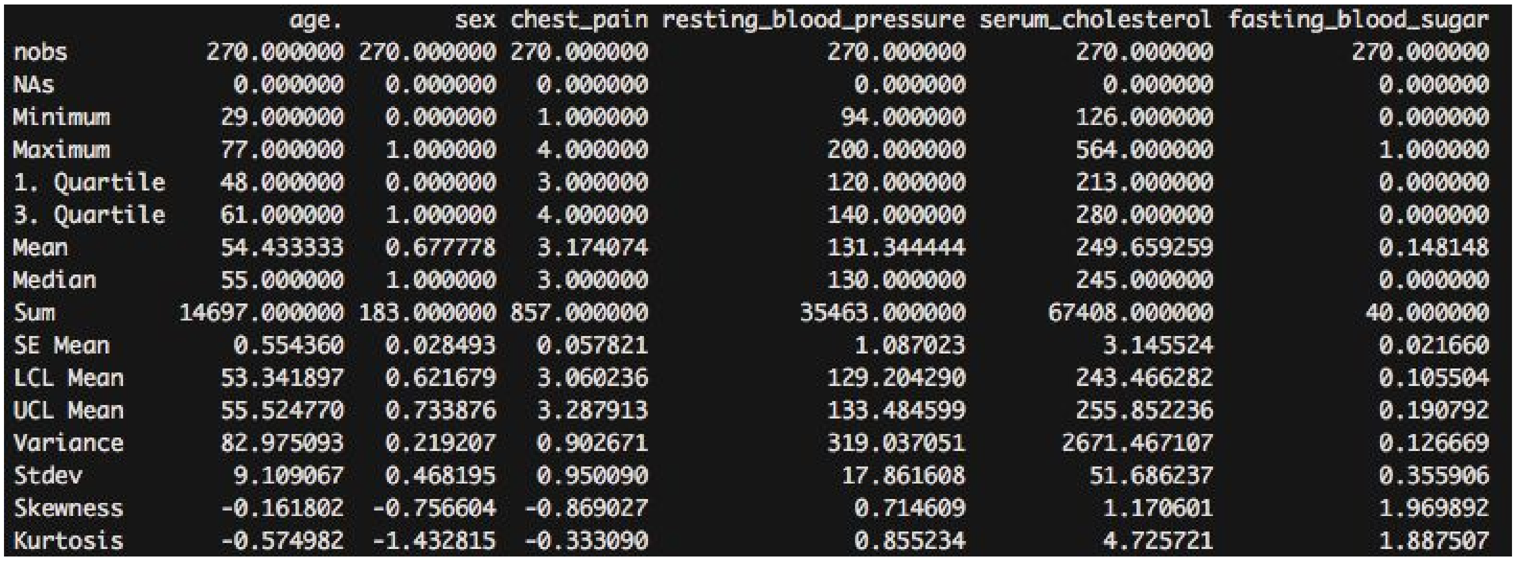
The dataset has 13 features and 1 binary label which are:

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholesterol in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

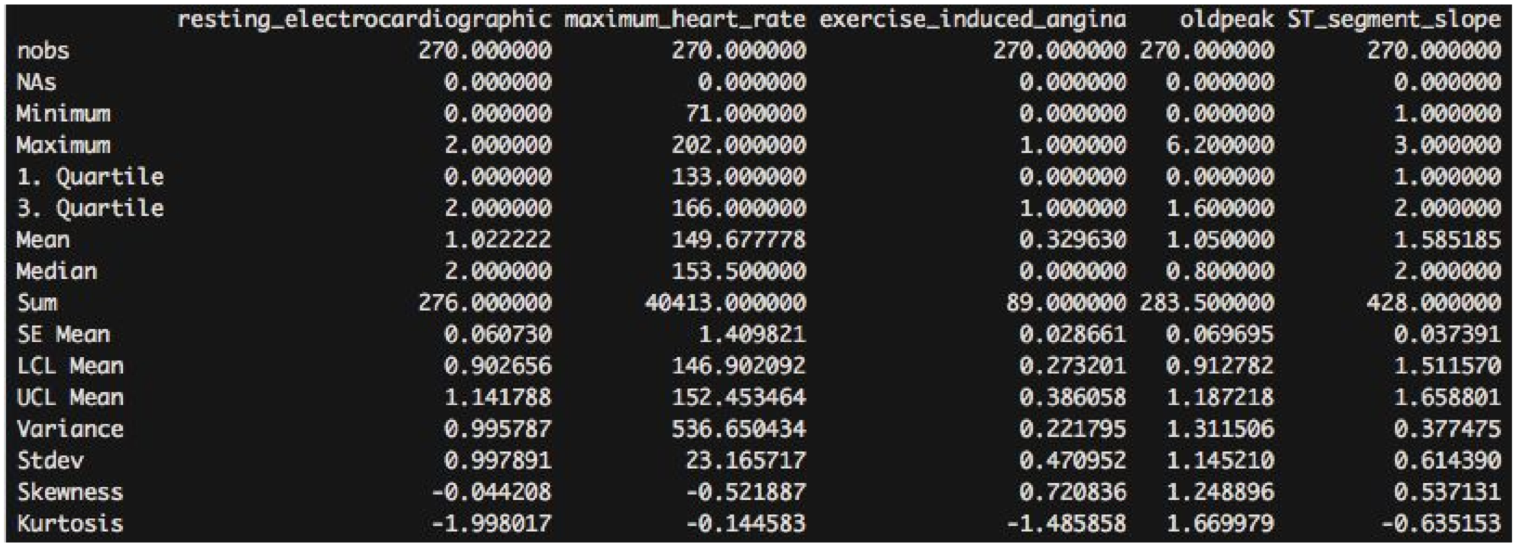
And

14. Absence (1) or presence (2) of heart disease

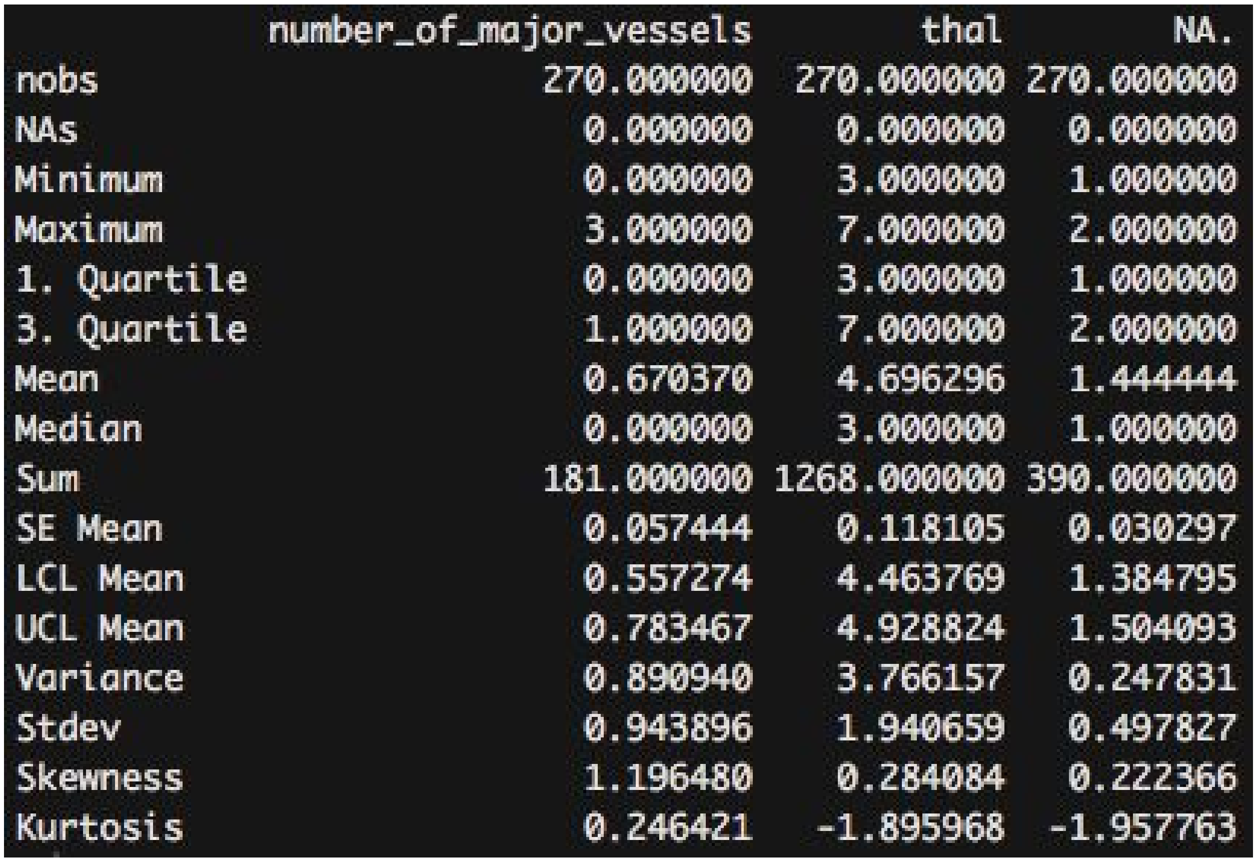
Simple statistics on our data:



```
                   age.        sex chest_pain resting_blood_pressure serum_cholesterol fasting_blood_sugar
nobs        270.000000 270.000000 270.000000             270.000000        270.000000          270.000000
NAs           0.000000   0.000000   0.000000               0.000000          0.000000            0.000000
Minimum      29.000000   0.000000   1.000000              94.000000        126.000000            0.000000
Maximum      77.000000   1.000000   4.000000             200.000000        564.000000            1.000000
1. Quartile  48.000000   0.000000   3.000000             120.000000        213.000000            0.000000
3. Quartile  61.000000   1.000000   4.000000             140.000000        280.000000            0.000000
Mean         54.433333   0.677778   3.174074             131.344444        249.659259            0.148148
Median       55.000000   1.000000   3.000000             130.000000        245.000000            0.000000
Sum       14697.000000 183.000000 857.000000           35463.000000      67408.000000           40.000000
SE Mean       0.554360   0.028493   0.057821               1.087023          3.145524            0.021660
LCL Mean     53.341897   0.621679   3.060236             129.204290        243.466282            0.105504
UCL Mean     55.524770   0.733876   3.287913             133.484599        255.852236            0.190792
Variance     82.975093   0.219207   0.902671             319.037051       2671.467107            0.126669
Stdev         9.109067   0.468195   0.950090              17.861608         51.686237            0.355906
Skewness     -0.161802  -0.756604  -0.869027               0.714609          1.170601            1.969892
Kurtosis     -0.574982  -1.432815  -0.333090               0.855234          4.725721            1.887507
```



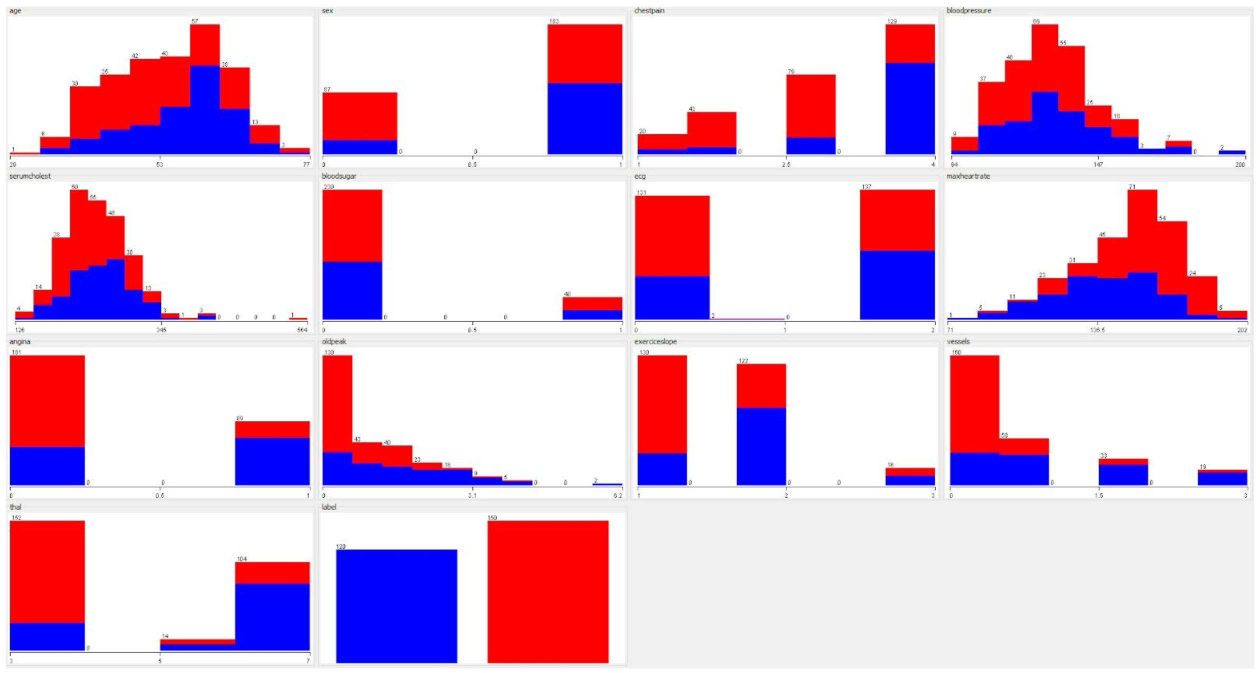
```
            resting_electrocardiographic maximum_heart_rate exercise_induced_angina    oldpeak ST_segment_slope
nobs                          270.000000         270.000000              270.000000 270.000000       270.000000
NAs                             0.000000           0.000000                0.000000   0.000000         0.000000
Minimum                         0.000000          71.000000                0.000000   0.000000         1.000000
Maximum                         2.000000         202.000000                1.000000   6.200000         3.000000
1. Quartile                     0.000000         133.000000                0.000000   0.000000         1.000000
3. Quartile                     2.000000         166.000000                1.000000   1.600000         2.000000
Mean                            1.022222         149.677778                0.329630   1.050000         1.585185
Median                          2.000000         153.500000                0.000000   0.800000         2.000000
Sum                           276.000000       40413.000000               89.000000 283.500000       428.000000
SE Mean                         0.060730           1.409821                0.028661   0.069695         0.037391
LCL Mean                        0.902656         146.902092                0.273201   0.912782         1.511570
UCL Mean                        1.141788         152.453464                0.386058   1.187218         1.658801
Variance                        0.995787         536.650434                0.221795   1.311506         0.377475
Stdev                           0.997891          23.165717                0.470952   1.145210         0.614390
Skewness                       -0.044208          -0.521887                0.720836   1.248896         0.537131
Kurtosis                       -1.998017          -0.144583               -1.485858   1.669979        -0.635153
```



```
             number_of_major_vessels        thal        NA.
nobs                      270.000000  270.000000 270.000000
NAs                         0.000000    0.000000   0.000000
Minimum                     0.000000    3.000000   1.000000
Maximum                     3.000000    7.000000   2.000000
1. Quartile                 0.000000    3.000000   1.000000
3. Quartile                 1.000000    7.000000   2.000000
Mean                        0.670370    4.696296   1.444444
Median                      0.000000    3.000000   1.000000
Sum                       181.000000 1268.000000 390.000000
SE Mean                     0.057444    0.118105   0.030297
LCL Mean                    0.557274    4.463769   1.384795
UCL Mean                    0.783467    4.928824   1.504093
Variance                    0.890940    3.766157   0.247831
Stdev                       0.943896    1.940659   0.497827
Skewness                    1.196480    0.284084   0.222366
Kurtosis                    0.246421   -1.895968  -1.957763
```

We used a function in R to display the detailed characteristics of our data set for each feature.
We also displayed several plots to visualize our data. The plots are histograms representing the number of subjects for a value for each feature, according to their classification (blue for sick people, red for healthy people).

As we can see there are various features, some numerical, some categorical (some binary, some not). Yet, all the data uses numbers (even the categorical features), which is good for some classifiers as it will require less preprocessing.



1.2. Problem and our prediction

The first problem we come to ask ourselves is whether all features are equally important in predicting the health status of a patient, or not.
A logical thought that comes to our minds, for example, is that age is very important in determining whether someone is sick or not. After all, young people are said to be healthier and unlikely to have heart attacks.

We will do some hypothesis testing to see whether this is true or not, and to understand our data better.

2. Hypothesis testing

2.1. First test

In this part we will set the hypothesis that:

Ho: the average age of people that have a heart disease is more than 65
Ha: the average age of people that have a heart disease is less than 65

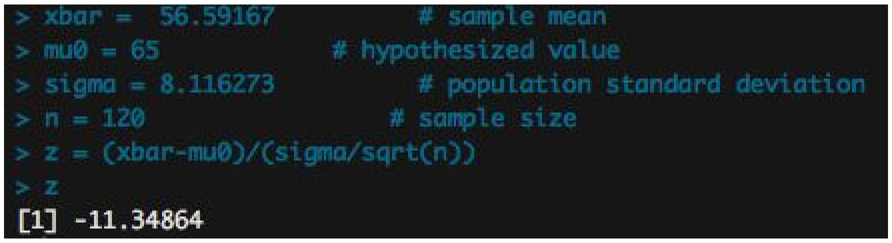
In other words: it's a one tailed test

Ho: $\mu \geqq 65$
Ha: $\mu < 65$

We will use 90 % confidence interval. $\alpha = 0.1$ and n=270

As n is large we can assimilate the sample's std as the population's std. We know that average age is 54.4, the variance is 89.2. Nevertheless in our case the sample is not the entire dataset. In fact we focus only on the people that have a disease.
In this case n=120 mean=56.59167 var=65.873880 std=8.116273 .

We compute the test statistic:



```
> xbar =  56.59167            # sample mean
> mu0 = 65              # hypothesized value
> sigma = 8.116273            # population standard deviation
> n = 120                   # sample size
> z = (xbar-mu0)/(sigma/sqrt(n))
> z
[1] -11.34864
```

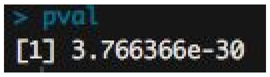
Do we fall in the rejection zone ?

In our case $\alpha = 0.1$ and it is a one tailed test. So Z$\alpha = 1,2816 \rightarrow$ this is our critical value.

As Zstat < Z_α we fall into the rejection zone. We have to reject Ho at a 90% level.

We can try making the same test at 95% level.
In that case $Z_\alpha = 1,6449$
As Zstat < Z_α we fall into the rejection zone. We have to reject Ho at a 95% level.

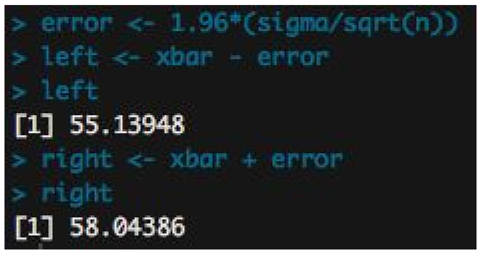
By P-value:



```
> pval
[1] 3.766366e-30
```

As it turns out to be less than the .05 significance level, we reject the null hypothesis that μ ≥ 65

By Confidence Interval:



```
> error <- 1.96*(sigma/sqrt(n))
> left <- xbar - error
> left
[1] 55.13948
> right <- xbar + error
> right
[1] 58.04386
```

Our interval is [55.13948 ; 58.04386], our value of 65 is far from falling inside this interval. We can say that at a 95 % we reject Ho.

Conclusion: The average age of sick person is less than 65.

2.2. Second test

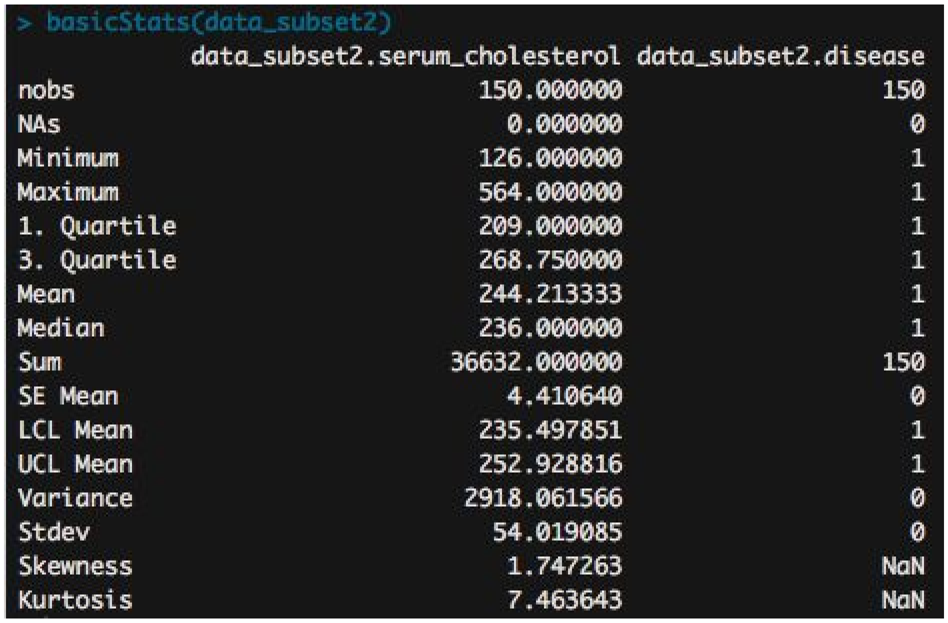
In this part we will set the hypothesis that:

Ho: the serum cholesterol averages at 250 mg/dl for any not sick individual
Ha: The average amount of serum cholesterol is different than 250 mg/dl

In other words: it's a two tailed tailed test

Ho: $\mu = 250$
Ha: $\mu \neq 250$

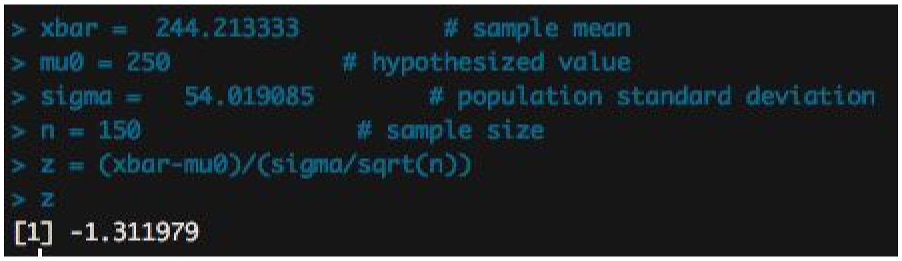
where μ_0 is a hypothesized value of the true population mean μ.
Suppose that = 0.05 (95% confidence to make the conclusions).
Our sample here is the non-sick individuals. so n = 150 are chosen for this test.
The basic statistics are given below:



```
> basicStats(data_subset2)
            data_subset2.serum_cholesterol data_subset2.disease
nobs                            150.000000                  150
NAs                               0.000000                    0
Minimum                         126.000000                    1
Maximum                         564.000000                    1
1. Quartile                     209.000000                    1
3. Quartile                     268.750000                    1
Mean                            244.213333                    1
Median                          236.000000                    1
Sum                           36632.000000                  150
SE Mean                           4.410640                    0
LCL Mean                        235.497851                    1
UCL Mean                        252.928816                    1
Variance                       2918.061566                    0
Stdev                            54.019085                    0
Skewness                          1.747263                  NaN
Kurtosis                          7.463643                  NaN
```

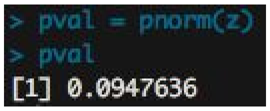
σ is assumed known and n is large, so this is a z test.

Critical values:
For = 0.05 (95%) the critical Z values are ±1.96


```
> xbar =  244.213333          # sample mean
> mu0 = 250             # hypothesized value
> sigma =   54.019085          # population standard deviation
> n = 150                  # sample size
> z = (xbar-mu0)/(sigma/sqrt(n))
> z
[1] -1.311979
```

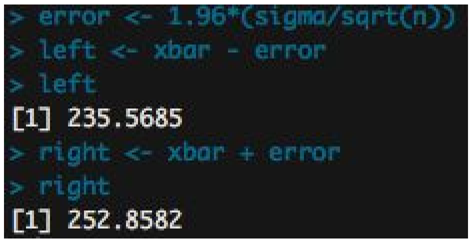
In our case Zstat = -1.311979
As we said before our non-rejection zone is [-1.96 ; + 1.96]
As Zstat falls into the non rejection zone, we keep Ho.
The serum cholesterol averages at 250 mg/dl for any not sick individual.

<u>P-value:</u>


```
> pval = pnorm(z)
> pval
[1] 0.0947636
```

P_value > 0.05 ; at a 95% confidence interval we can keep Ho. This confirms the previous result.

Confidence Interval:


```
> error <- 1.96*(sigma/sqrt(n))
> left <- xbar - error
> left
[1] 235.5685
> right <- xbar + error
> right
[1] 252.8582
```

We see that 250 falls into our confidence interval.

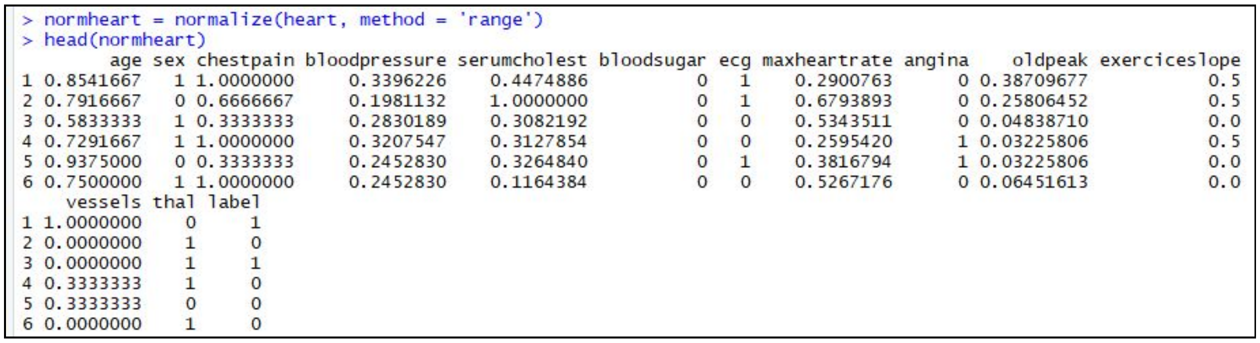
Conclusion:

We valid Ho. The serum cholesterol averages at 250 mg/dl for any not sick individual.

3. Classification Model

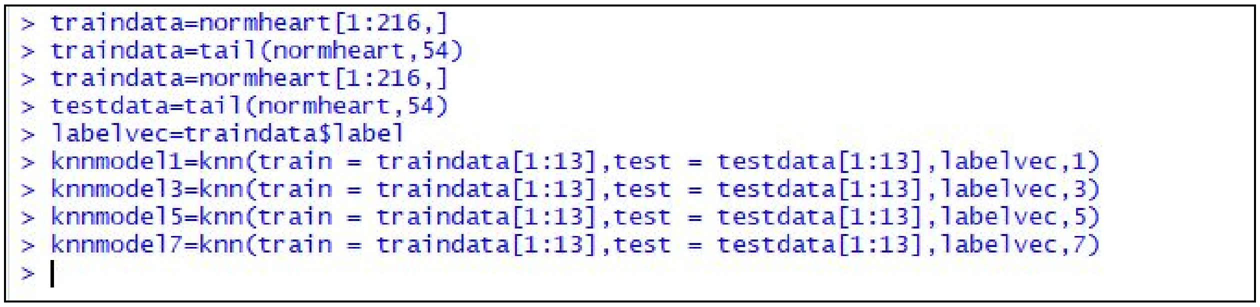
3.1. KNN

The KNN classifiers is one of the most commons classifiers, and we expect it to work well with numerical data such as ours.
But before applying the classifier to our dataset, we need to preprocess it a little. All our data is already numerical, but it needs to be normalized. Since we have a lot of features, it takes time to do it by hand, so I looked for a library that would help me do that. I installed and loaded the BBmisc package, which has a simple normalize() function.
Here's a glimpse of the data after normalization (with range between 0 and 1):



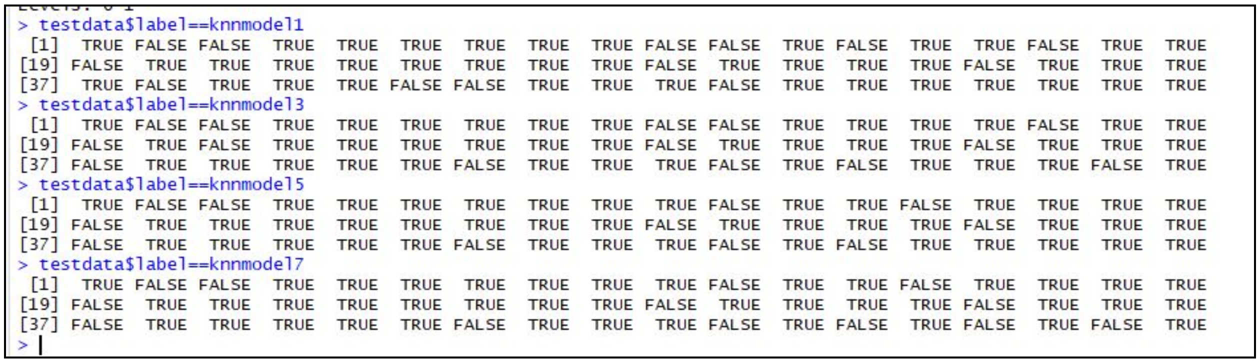
```
> normheart = normalize(heart, method = 'range')
> head(normheart)
        age sex chestpain bloodpressure serumcholest bloodsugar ecg maxheartrate angina    oldpeak exerciceslope
1 0.8541667   1 1.0000000     0.3396226    0.4474886          0   1    0.2900763      0 0.38709677           0.5
2 0.7916667   0 0.6666667     0.1981132    1.0000000          0   1    0.6793893      0 0.25806452           0.5
3 0.5833333   1 0.3333333     0.2830189    0.3082192          0   0    0.5343511      0 0.04838710           0.0
4 0.7291667   1 1.0000000     0.3207547    0.3127854          0   0    0.2595420      1 0.03225806           0.5
5 0.9375000   0 0.3333333     0.2452830    0.3264840          0   1    0.3816794      1 0.03225806           0.0
6 0.7500000   1 1.0000000     0.2452830    0.1164384          0   0    0.5267176      0 0.06451613           0.0
    vessels thal label
1 1.0000000    0     1
2 0.0000000    1     0
3 0.0000000    1     1
4 0.3333333    1     0
5 0.3333333    0     0
6 0.0000000    1     0
```

Now that the data is normalized, I can load the class library and apply the knn classifier. I create two variables to store have distincts training data and test data, with a 5:1 ratio.

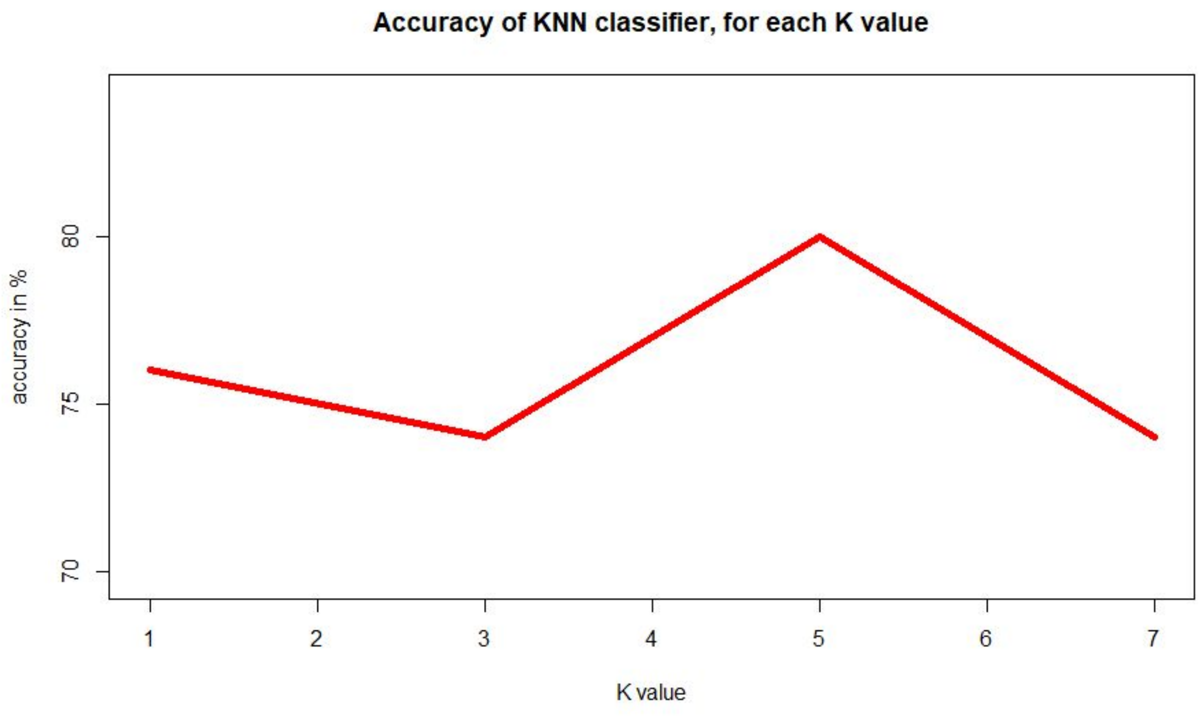


```
> traindata=normheart[1:216,]
> traindata=tail(normheart,54)
> traindata=normheart[1:216,]
> testdata=tail(normheart,54)
> labelvec=traindata$label
> knnmodel1=knn(train = traindata[1:13],test = testdata[1:13],labelvec,1)
> knnmodel3=knn(train = traindata[1:13],test = testdata[1:13],labelvec,3)
> knnmodel5=knn(train = traindata[1:13],test = testdata[1:13],labelvec,5)
> knnmodel7=knn(train = traindata[1:13],test = testdata[1:13],labelvec,7)
> 
```

Then we applied the classifier for k values of 1, 3, 5 and 7.
We then compared the results of each classifier with the actual label value of the test data.



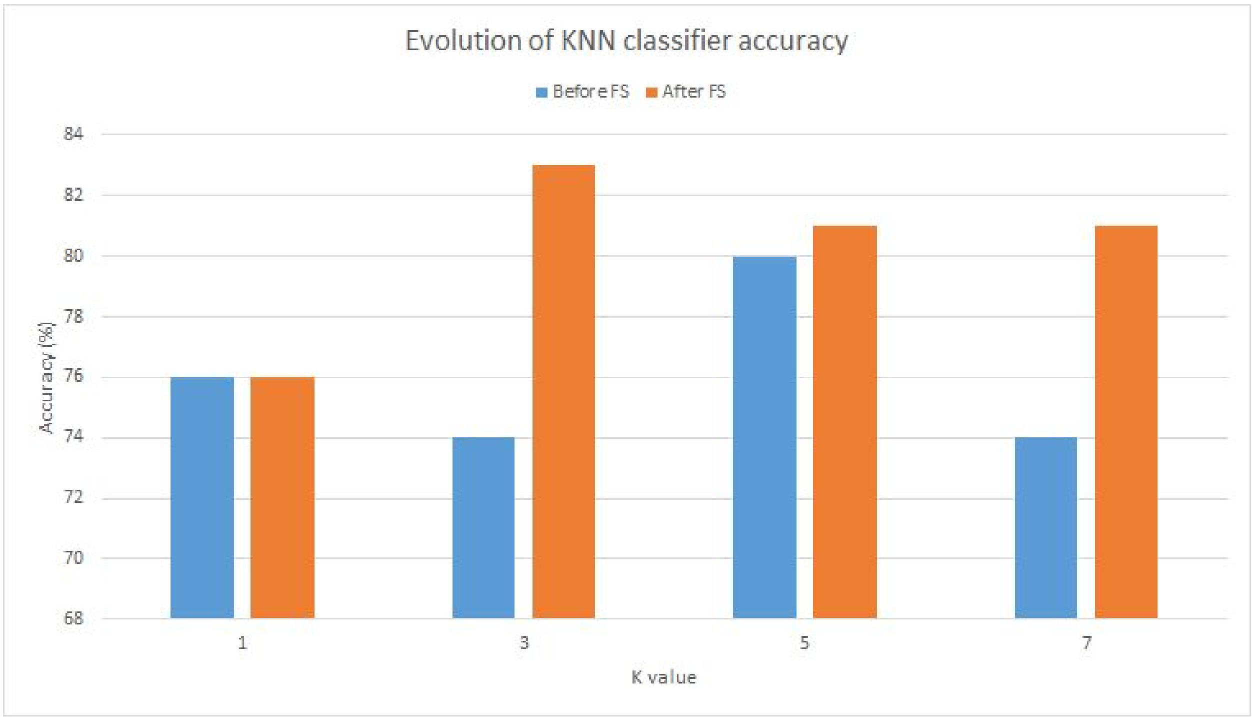
```
> testdata$label==knnmodel1
 [1]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
[19] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[37]  TRUE FALSE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
> testdata$label==knnmodel3
 [1]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
[19] FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[37] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE
> testdata$label==knnmodel5
 [1]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[19] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[37] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> testdata$label==knnmodel7
 [1]  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
[19] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
[37] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE FALSE  TRUE
> 
```



As we can see the accuracy is highest for a K value of 5. We get 80% accuracy, which is quite good.

Yet, since there are a lot of features, maybe some of them are not very useful and have a bad impact on accuracy. Therefore, we decide to rerun the classifiers after a feature selection. For this, we basically used the features that were kept in the backward selection process we used when building the logistic regression model (it's several pages down).
To sum up, we selected the most important features (3,5,7,8,9,10,12 and 13) and built the models again.

Indeed, the classifiers' results are improved for all values of K. And the maximum accuracy value we had for this classifier jumps from 80 to 85% for K=3.

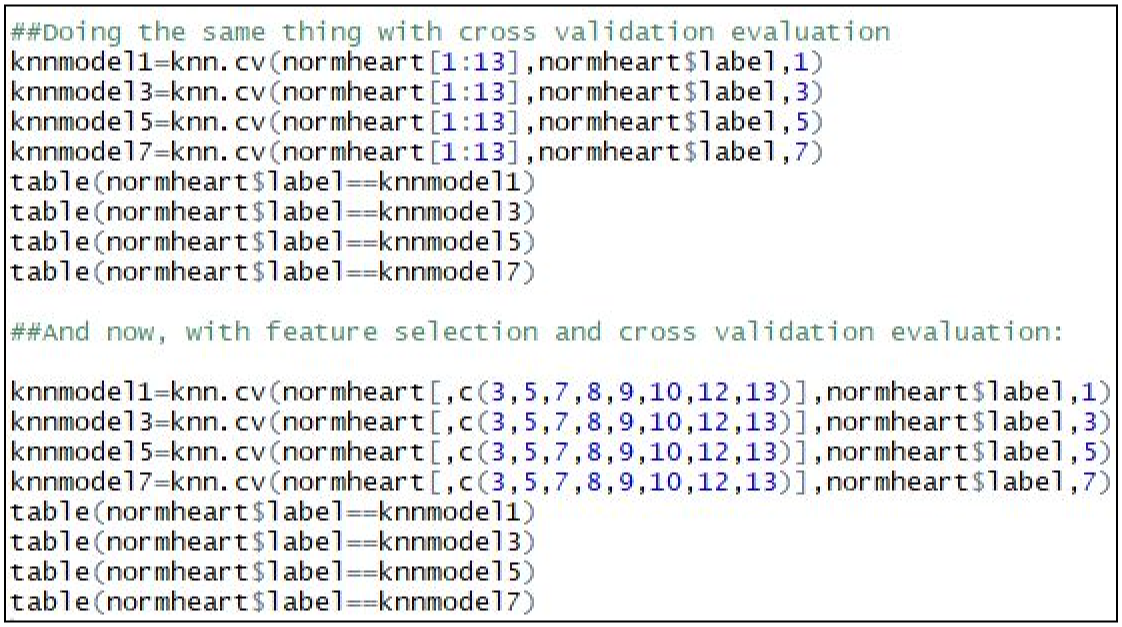


This means our feature selection was successful.

However the evaluation method we used was a classic Hold Out Evaluation (with a 80% training set). This evaluation method is usually less accurate when we work with small sized data set like this one.

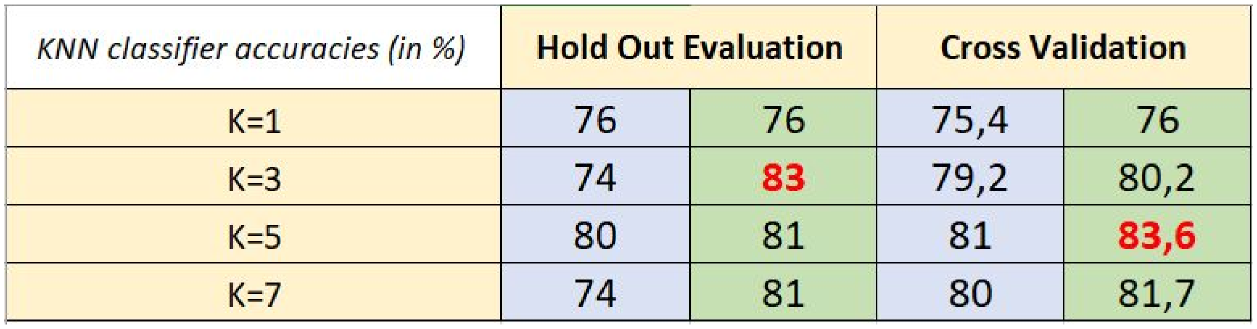
On small data sets, it’s usually better to use cross validation to compute the accuracy. This is what we decided to do next for this classifier. Since it can easily be done in R: the class package has a built-in knn.cv function for that.

Here’s the R code using the entire data set:



```
##Doing the same thing with cross validation evaluation
knnmodel1=knn.cv(normheart[1:13],normheart$label,1)
knnmodel3=knn.cv(normheart[1:13],normheart$label,3)
knnmodel5=knn.cv(normheart[1:13],normheart$label,5)
knnmodel7=knn.cv(normheart[1:13],normheart$label,7)
table(normheart$label==knnmodel1)
table(normheart$label==knnmodel3)
table(normheart$label==knnmodel5)
table(normheart$label==knnmodel7)

##And now, with feature selection and cross validation evaluation:

knnmodel1=knn.cv(normheart[,c(3,5,7,8,9,10,12,13)],normheart$label,1)
knnmodel3=knn.cv(normheart[,c(3,5,7,8,9,10,12,13)],normheart$label,3)
knnmodel5=knn.cv(normheart[,c(3,5,7,8,9,10,12,13)],normheart$label,5)
knnmodel7=knn.cv(normheart[,c(3,5,7,8,9,10,12,13)],normheart$label,7)
table(normheart$label==knnmodel1)
table(normheart$label==knnmodel3)
table(normheart$label==knnmodel5)
table(normheart$label==knnmodel7)
```

And now the result (you can try the script yourself).



KNN classifier accuracies (in %)	**Hold Out Evaluation**		**Cross Validation**	
K=1	76	76	75,4	76
K=3	74	**83**	79,2	80,2
K=5	80	81	81	**83,6**
K=7	74	81	80	81,7

Before feature selection in blue, after feature selection in green.

So for this classifier, for most tests, the difference is small between Hold

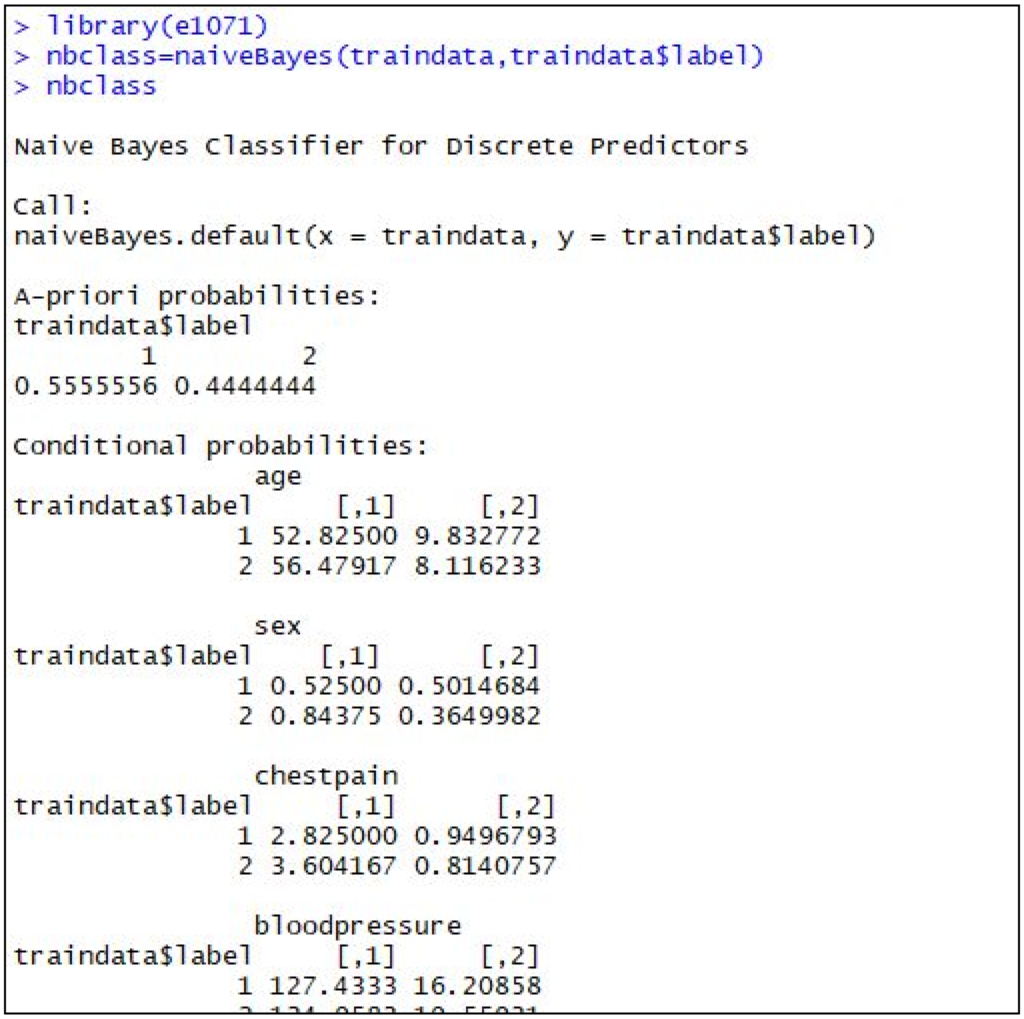
out Evaluation and Cross validation. We do notice some difference for K=3 or 7 before FS.
Also, the new evaluation method brings our maximum accuracy from 83% to 83.6%.

But in general accuracies are in the same range, the max accuracy is similar and we also get the conclusion that feature selection is successful.

So for the next classifiers we choose to use and show results for hold out evaluation (since it is more simple), and will display cross validation results if we obtain major accuracy differences in our tests. Otherwise, this report will be much longer than we already plan it to be for little gain.

3.2. Naive Bayes

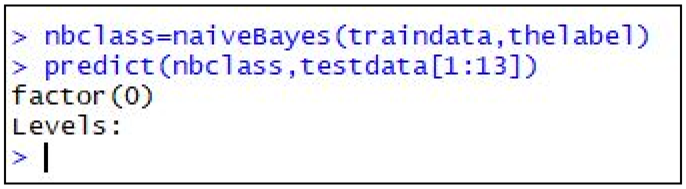
Now we will try using a Naive Bayes classifier with our data. We will need to install and load the e1071 package and its naiveBayes() function.



```
> library(e1071)
> nbclass=naiveBayes(traindata,traindata$label)
> nbclass

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = traindata, y = traindata$label)

A-priori probabilities:
traindata$label
        1         2 
0.5555556 0.4444444 

Conditional probabilities:
               age
traindata$label     [,1]     [,2]
              1 52.82500 9.832772
              2 56.47917 8.116233

               sex
traindata$label    [,1]      [,2]
              1 0.52500 0.5014684
              2 0.84375 0.3649982

               chestpain
traindata$label     [,1]      [,2]
              1 2.825000 0.9496793
              2 3.604167 0.8140757

               bloodpressure
traindata$label     [,1]     [,2]
              1 127.4333 16.20858
```

Now that we have built the model, let's try it with our test data, still being 20% of the data set.

Unfortunately, in R, we have an error that displays and prevents us from reaching our goal and to use our naive bayes model:

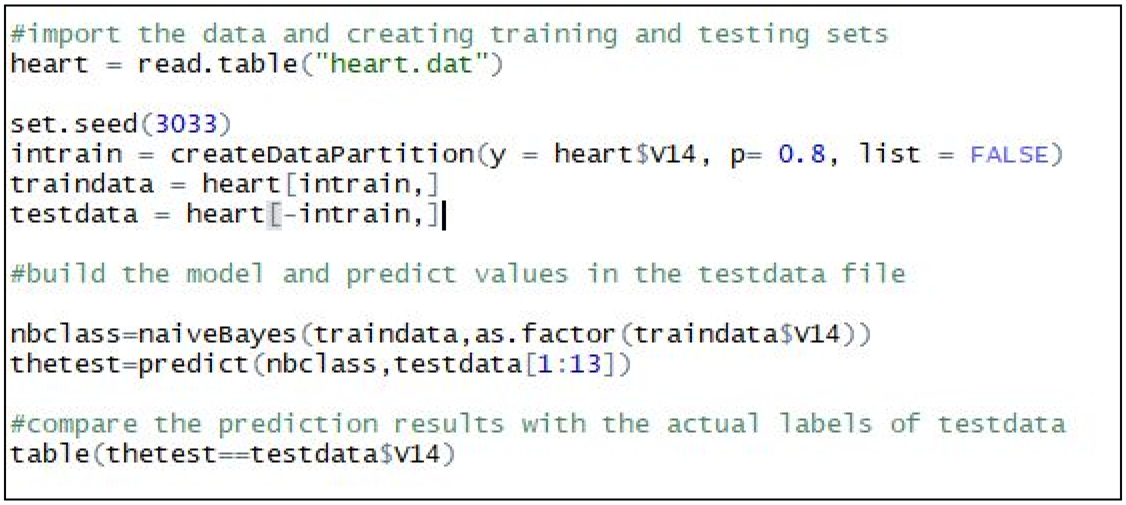


```
> nbclass=naiveBayes(traindata,thelabel)
> predict(nbclass,testdata[1:13])
factor(0)
Levels: 
> 
```

Our searches indicate that this might be due to the original data not being discretized to nominal data.
However this shouldn't be the case, since the documentation (and previous uses in other works) show that the function from this package automatically discretizes numerical data.

After numerous searches online and a lot of time spent, we realized the issue was linked to data formatting. Since there are strings in the label/class vector, we needed to use as.factor() on it to make it usable by the function. Otherwise nbclass$levels would always return NULL.

Therefore, we corrected our script and obtained an accuracy result:

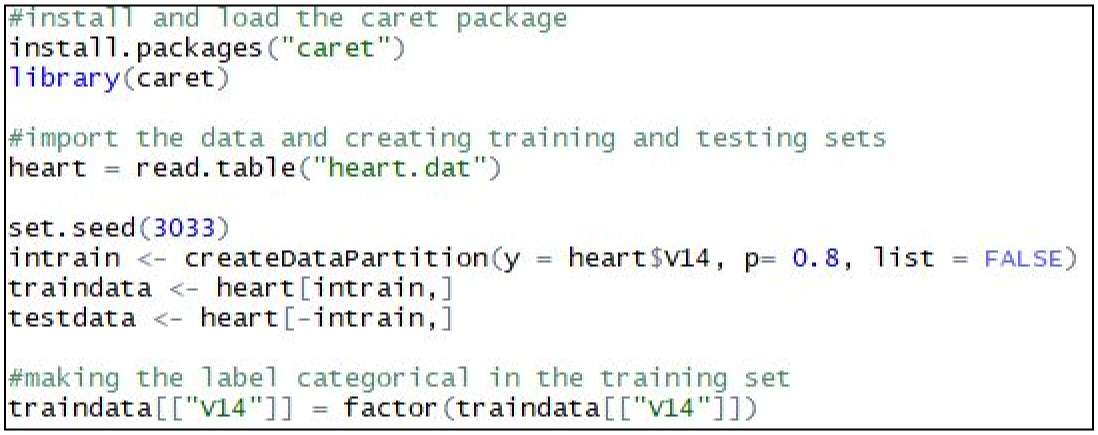


```
#import the data and creating training and testing sets
heart = read.table("heart.dat")

set.seed(3033)
intrain = createDataPartition(y = heart$V14, p= 0.8, list = FALSE)
traindata = heart[intrain,]
testdata = heart[-intrain,]

#build the model and predict values in the testdata file

nbclass=naiveBayes(traindata,as.factor(traindata$V14))
thetest=predict(nbclass,testdata[1:13])

#compare the prediction results with the actual labels of testdata
table(thetest==testdata$V14)
```

The accuracy from the prediction using the testing data (still a hold out evaluation with 20% of the data for testing) is 81%, which is similar to what we got on other classifiers.

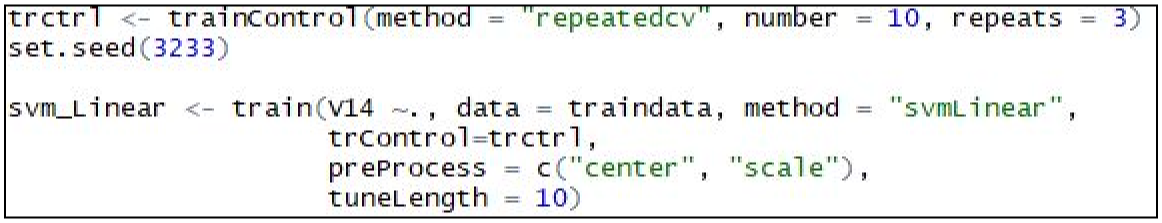
3.3. Support Vector Machine

Here we tried implementing a SVM classifier to see if it could have better performances than the other classifiers we tried. So to do this we installed and loaded the caret package (Classification And REgression Tasks).
After reading the documentation, on the train function we are going to use, we decided to import again the data and recreate the training and testing sets.



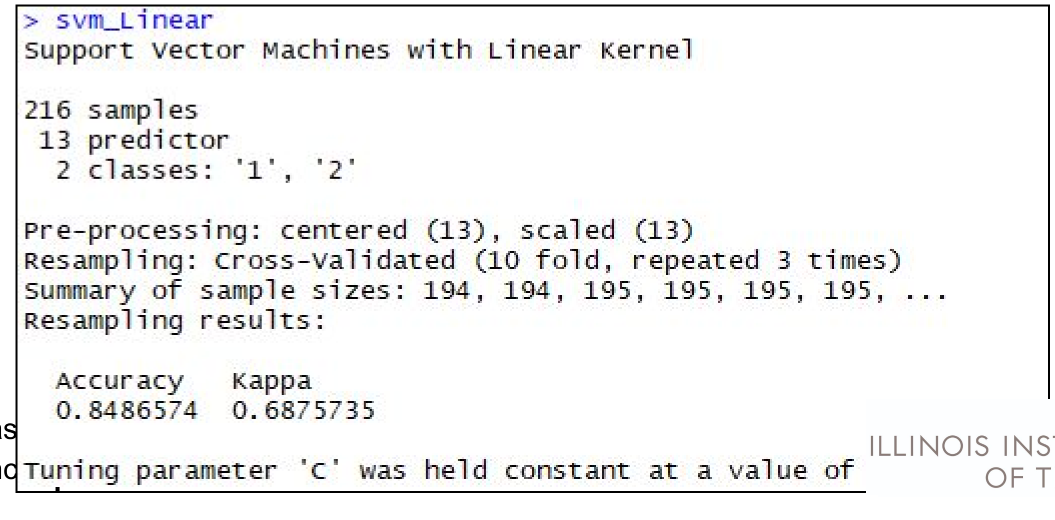
```
#install and load the caret package
install.packages("caret")
library(caret)

#import the data and creating training and testing sets
heart = read.table("heart.dat")

set.seed(3033)
intrain <- createDataPartition(y = heart$V14, p= 0.8, list = FALSE)
traindata <- heart[intrain,]
testdata <- heart[-intrain,]

#making the label categorical in the training set
traindata[["V14"]] = factor(traindata[["V14"]])
```

This time we didn't change the column headers' names, and we made the label column categorical. Then we built our model with the train function:



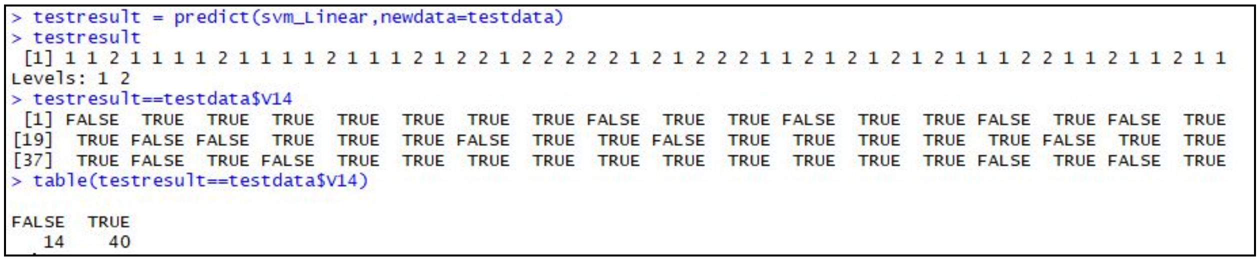
```
trctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
set.seed(3233)

svm_Linear <- train(V14 ~., data = traindata, method = "svmLinear",
                    trControl=trctrl,
                    preProcess = c("center", "scale"),
                    tuneLength = 10)
```

Here is the resulting model we ended up getting:



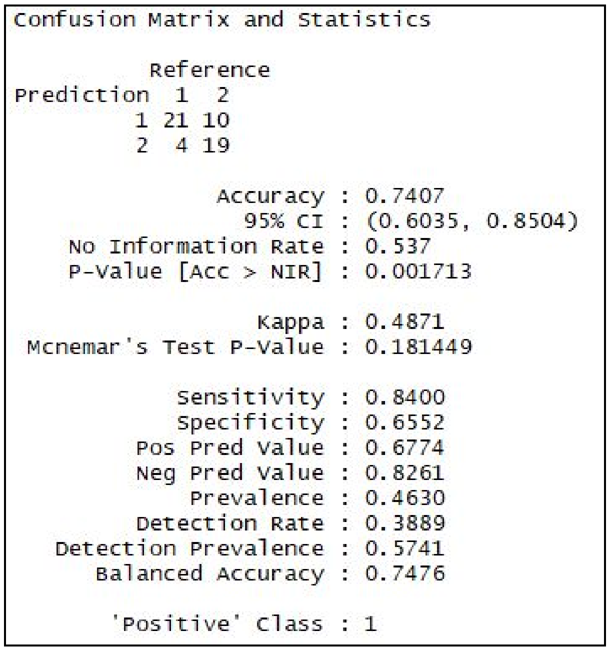
```
> svm_Linear
Support Vector Machines with Linear Kernel

216 samples
 13 predictor
  2 classes: '1', '2'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 194, 194, 195, 195, 195, 195, ...
Resampling results:

  Accuracy   Kappa
  0.8486574  0.6875735

Tuning parameter 'C' was held constant at a value of
```

This model seems to have an 85% accuracy, but let's check this accuracy value with our testing set, using the predict function:



```
> testresult = predict(svm_Linear,newdata=testdata)
> testresult
 [1] 1 1 2 1 1 1 1 2 1 1 1 1 2 1 1 1 2 1 2 2 1 2 2 2 2 1 2 1 2 2 2 1 1 2 1 2 1 2 1 2 1 1 1 2 2 1 1 2 1 1 2 1 1
Levels: 1 2
> testresult==testdata$V14
 [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
[19]  TRUE FALSE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE
[37]  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE
> table(testresult==testdata$V14)

FALSE  TRUE 
   14    40 
```

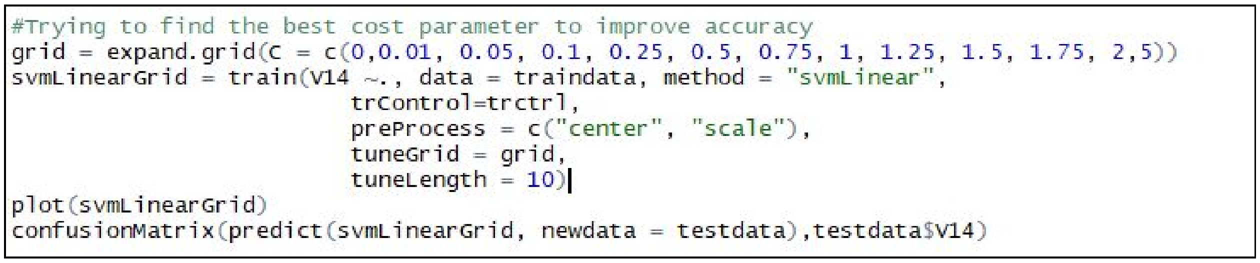
As we see, we actually obtain a 74% accuracy. Let's display the confusion matrix for more clarity on the results:



```
Confusion Matrix and Statistics

          Reference
Prediction  1  2
         1 21 10
         2  4 19
                                          
               Accuracy : 0.7407          
                 95% CI : (0.6035, 0.8504)
    No Information Rate : 0.537           
    P-Value [Acc > NIR] : 0.001713        
                                          
                  Kappa : 0.4871          
 Mcnemar's Test P-Value : 0.181449        
                                          
            Sensitivity : 0.8400          
            Specificity : 0.6552          
         Pos Pred Value : 0.6774          
         Neg Pred Value : 0.8261          
             Prevalence : 0.4630          
         Detection Rate : 0.3889          
   Detection Prevalence : 0.5741          
      Balanced Accuracy : 0.7476          
                                          
       'Positive' Class : 1               
```

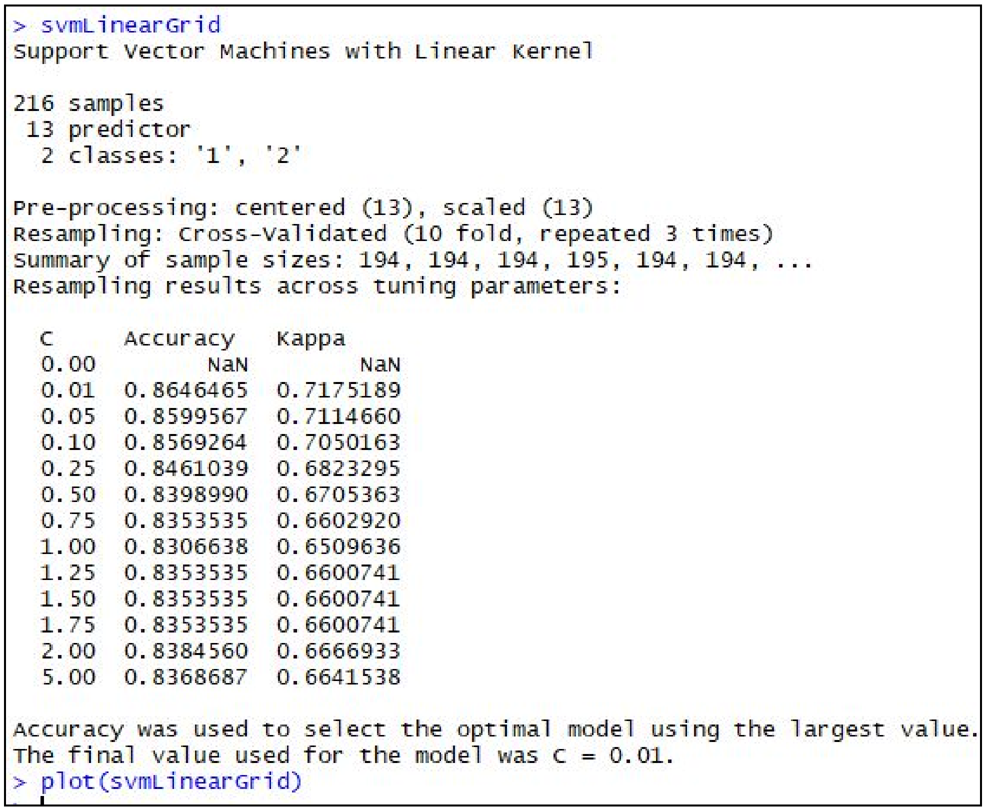
Most of the errors made are for sick patients diagnosed as healthy, which is quite bad for the our main goal (which is to predict patients' health).

We then tried improving this model by finding the best cost parameter:



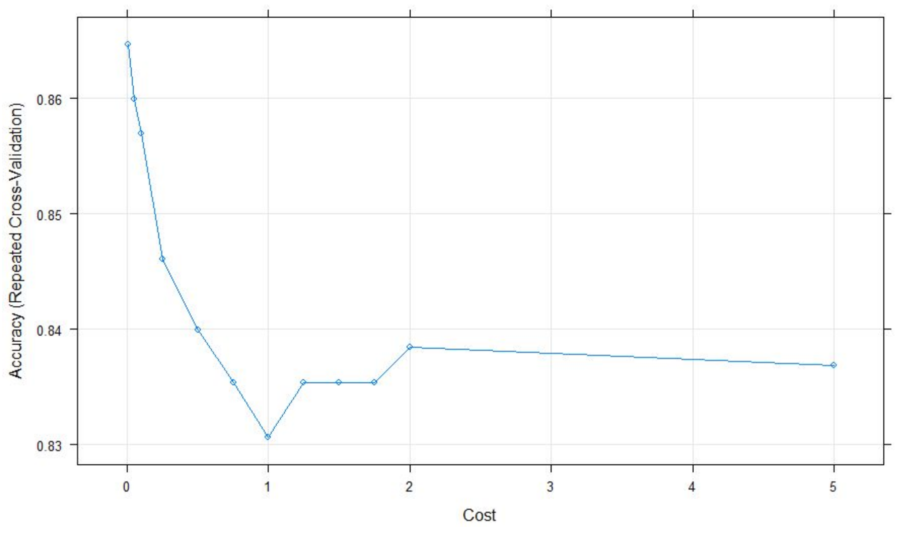
```
#Trying to find the best cost parameter to improve accuracy
grid = expand.grid(C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2,5))
svmLinearGrid = train(V14 ~., data = traindata, method = "svmLinear",
                          trControl=trctrl,
                          preProcess = c("center", "scale"),
                          tuneGrid = grid,
                          tuneLength = 10)
plot(svmLinearGrid)
confusionMatrix(predict(svmLinearGrid, newdata = testdata),testdata$V14)
```

Here's the result:



```
> svmLinearGrid
Support Vector Machines with Linear Kernel

216 samples
 13 predictor
  2 classes: '1', '2'

Pre-processing: centered (13), scaled (13)
Resampling: Cross-Validated (10 fold, repeated 3 times)
Summary of sample sizes: 194, 194, 194, 195, 194, 194, ...
Resampling results across tuning parameters:

  C     Accuracy   Kappa
  0.00        NaN        NaN
  0.01  0.8646465  0.7175189
  0.05  0.8599567  0.7114660
  0.10  0.8569264  0.7050163
  0.25  0.8461039  0.6823295
  0.50  0.8398990  0.6705363
  0.75  0.8353535  0.6602920
  1.00  0.8306638  0.6509636
  1.25  0.8353535  0.6600741
  1.50  0.8353535  0.6600741
  1.75  0.8353535  0.6600741
  2.00  0.8384560  0.6666933
  5.00  0.8368687  0.6641538

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was C = 0.01.
> plot(svmLinearGrid)
```

The best accuracy is achieved for a cost value of 0.01. We then reuse the predict function on our testing data set with this new parameter, and we managed to get a better accuracy with 1 less false negative value.
We now have a 76% accuracy.

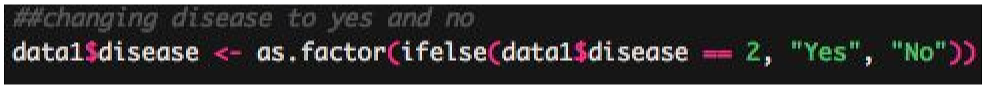


We now have an improved SVM classifier model.

3.4. Logistic regression

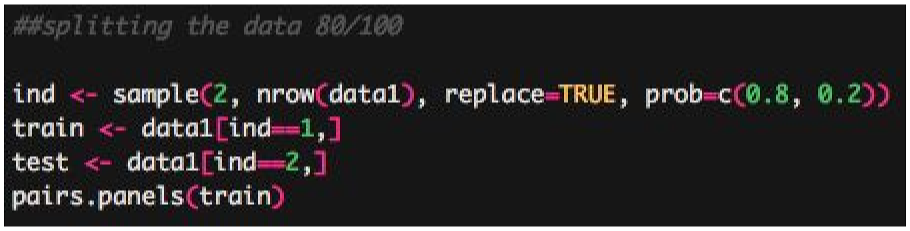
In our case for this logistic regression we will use the class of the disease as the y variable, as it is the only categorical variable and the one we want to predict. The others features will be our x variables.

We preprocess our data by changing the label into a nominal binary variable.



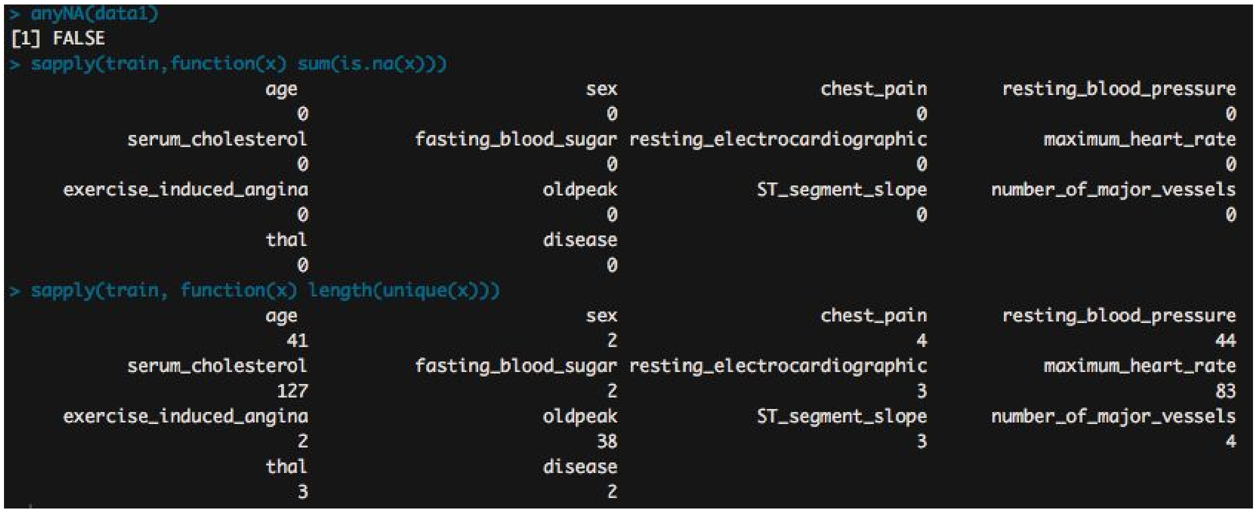
```
##changing disease to yes and no
data1$disease <- as.factor(ifelse(data1$disease == 2, "Yes", "No"))
```

We build our train and test dataset with a 0.8 ratio:



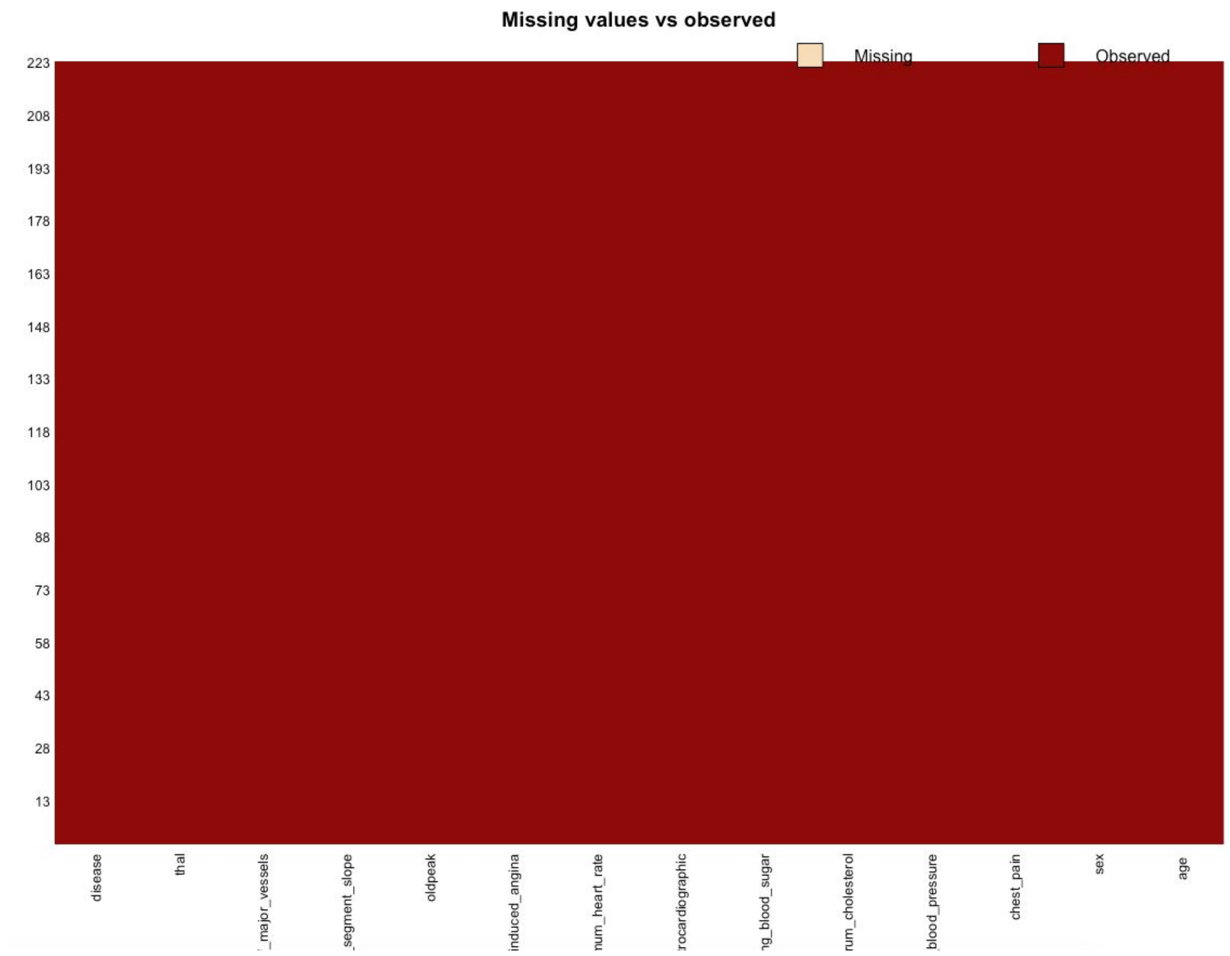
```
##splitting the data 80/100

ind <- sample(2, nrow(data1), replace=TRUE, prob=c(0.8, 0.2))
train <- data1[ind==1,]
test <- data1[ind==2,]
pairs.panels(train)
```

We check on the missing values, and the frequence of the variables:

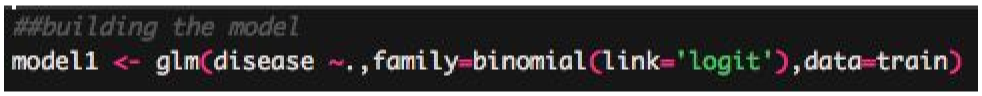


```
> anyNA(data1)
[1] FALSE
> sapply(train,function(x) sum(is.na(x)))
                     age                      sex               chest_pain   resting_blood_pressure
                       0                        0                        0                        0
         serum_cholesterol      fasting_blood_sugar resting_electrocardiographic        maximum_heart_rate
                       0                        0                        0                        0
   exercise_induced_angina                  oldpeak         ST_segment_slope   number_of_major_vessels
                       0                        0                        0                        0
                    thal                  disease
                       0                        0
> sapply(train, function(x) length(unique(x)))
                     age                      sex               chest_pain   resting_blood_pressure
                      41                        2                        4                       44
         serum_cholesterol      fasting_blood_sugar resting_electrocardiographic        maximum_heart_rate
                     127                        2                        3                       83
   exercise_induced_angina                  oldpeak         ST_segment_slope   number_of_major_vessels
                       2                       38                        3                        4
                    thal                  disease
                       3                        2
```

Nour Al-Habash
Clement Francois Dit Hardy
ILLINOIS INSTITUTE OF TECHNOLOGY



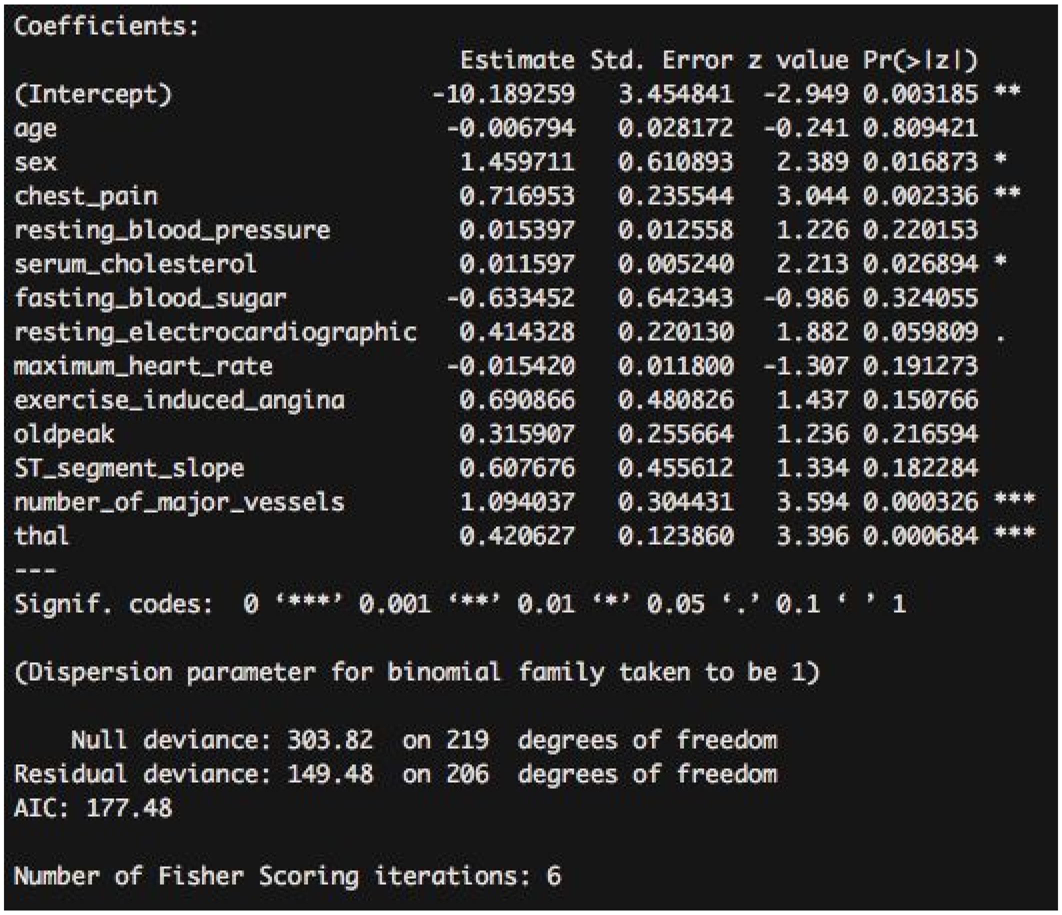
We can now build our model. As our dataset is clean and there is no missing values, we do not need to remove any variables from it. In order to build a logistic regression we use glm() function as follow:



```
##building the model
model1 <- glm(disease ~.,family=binomial(link='logit'),data=train)
```

ILLINOIS INSTITUTE OF TECHNOLOGY

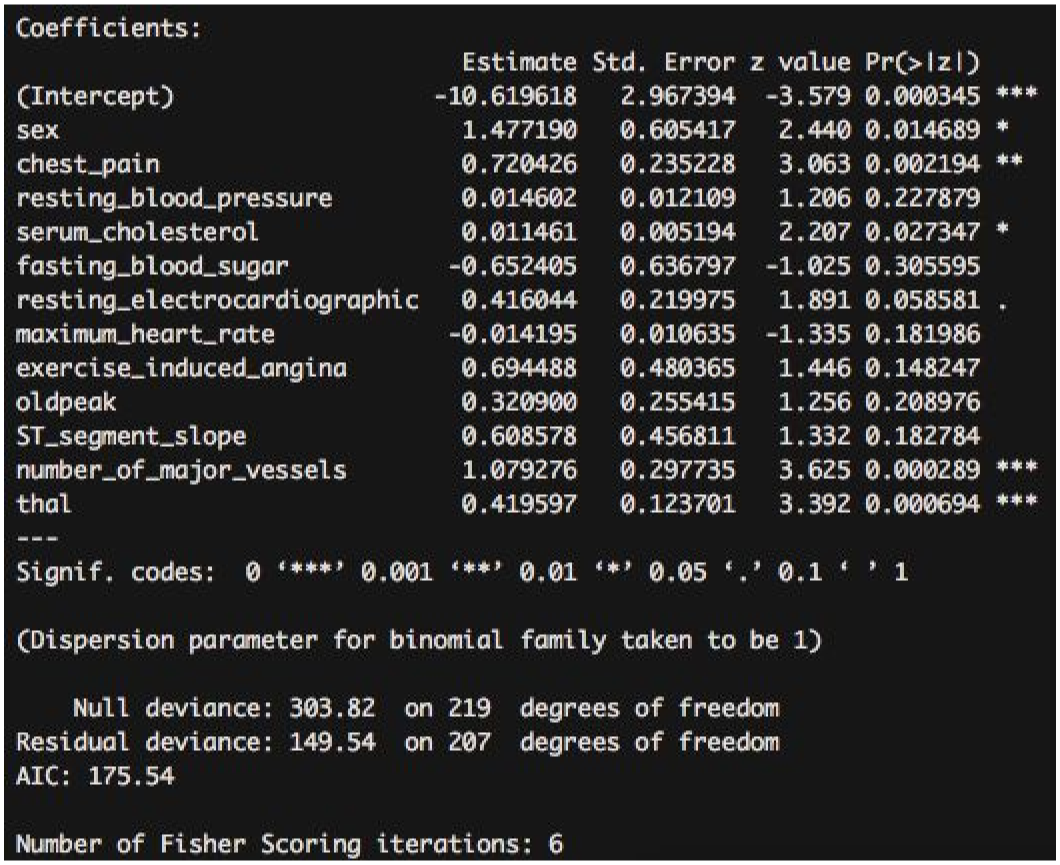
<u>Output:</u>



```
Coefficients:
                              Estimate Std. Error z value Pr(>|z|)    
(Intercept)                 -10.189259   3.454841  -2.949 0.003185 ** 
age                          -0.006794   0.028172  -0.241 0.809421    
sex                           1.459711   0.610893   2.389 0.016873 *  
chest_pain                    0.716953   0.235544   3.044 0.002336 ** 
resting_blood_pressure        0.015397   0.012558   1.226 0.220153    
serum_cholesterol             0.011597   0.005240   2.213 0.026894 *  
fasting_blood_sugar          -0.633452   0.642343  -0.986 0.324055    
resting_electrocardiographic  0.414328   0.220130   1.882 0.059809 .  
maximum_heart_rate           -0.015420   0.011800  -1.307 0.191273    
exercise_induced_angina       0.690866   0.480826   1.437 0.150766    
oldpeak                       0.315907   0.255664   1.236 0.216594    
ST_segment_slope              0.607676   0.455612   1.334 0.182284    
number_of_major_vessels       1.094037   0.304431   3.594 0.000326 ***
thal                          0.420627   0.123860   3.396 0.000684 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 303.82  on 219  degrees of freedom
Residual deviance: 149.48  on 206  degrees of freedom
AIC: 177.48

Number of Fisher Scoring iterations: 6
```

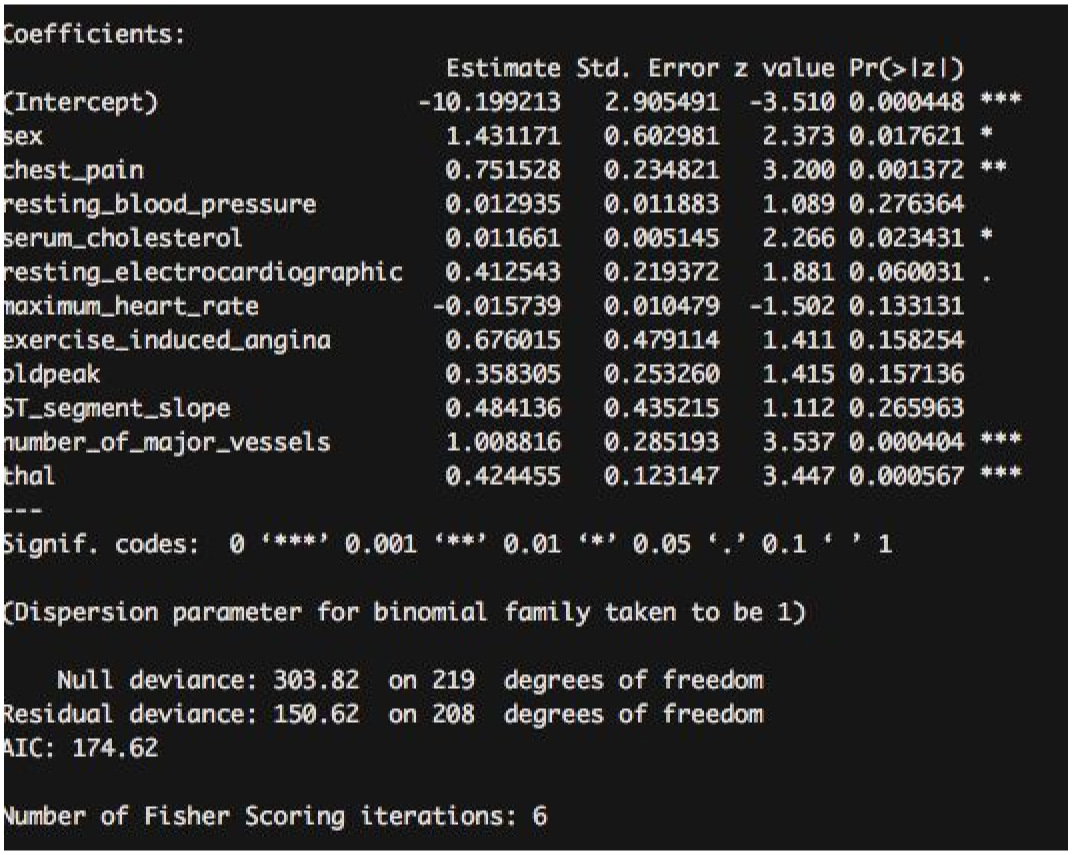
A large (insignificant) p-value suggests that changes in the predictor are not associated with changes in the response.
To proceed on the variable selection we decided to evaluate the relevance of our variables with their P-value associated.

We should remove “age” variable because it has the highest p-Value. Once removed we will build a new model and from there repeat the same principle until every p-value is under 0.05.
We will proceed on the same test each time we remove a variable.



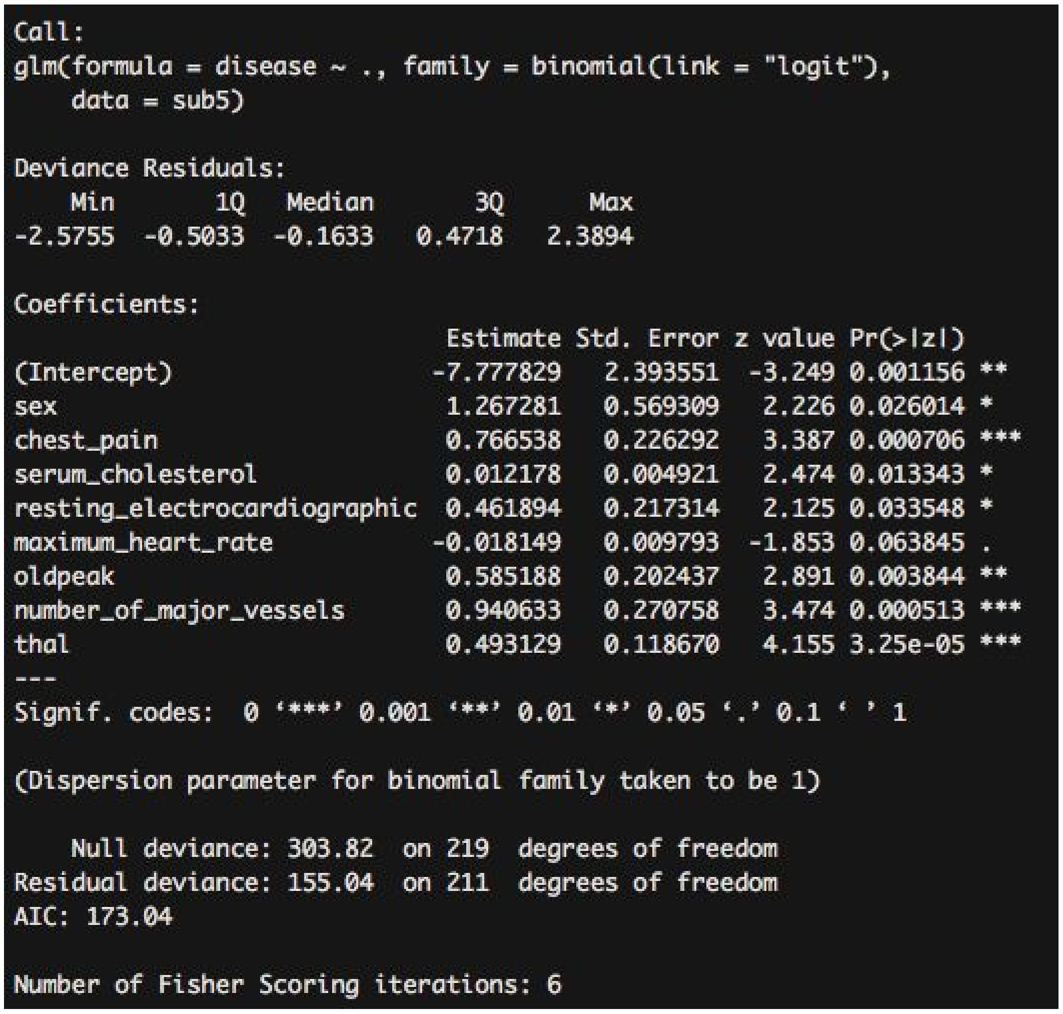
```
Coefficients:
                              Estimate Std. Error z value Pr(>|z|)    
(Intercept)                 -10.619618   2.967394  -3.579 0.000345 ***
sex                           1.477190   0.605417   2.440 0.014689 *  
chest_pain                    0.720426   0.235228   3.063 0.002194 ** 
resting_blood_pressure        0.014602   0.012109   1.206 0.227879    
serum_cholesterol             0.011461   0.005194   2.207 0.027347 *  
fasting_blood_sugar          -0.652405   0.636797  -1.025 0.305595    
resting_electrocardiographic  0.416044   0.219975   1.891 0.058581 .  
maximum_heart_rate           -0.014195   0.010635  -1.335 0.181986    
exercise_induced_angina       0.694488   0.480365   1.446 0.148247    
oldpeak                       0.320900   0.255415   1.256 0.208976    
ST_segment_slope              0.608578   0.456811   1.332 0.182784    
number_of_major_vessels       1.079276   0.297735   3.625 0.000289 ***
thal                          0.419597   0.123701   3.392 0.000694 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 303.82  on 219  degrees of freedom
Residual deviance: 149.54  on 207  degrees of freedom
AIC: 175.54

Number of Fisher Scoring iterations: 6
```

Now we should remove the fasting_blood_sugar variable as it doesn’t have an impact on the outcome of the model.



```
Coefficients:
                             Estimate Std. Error z value Pr(>|z|)    
(Intercept)                -10.199213   2.905491  -3.510 0.000448 ***
sex                          1.431171   0.602981   2.373 0.017621 *  
chest_pain                   0.751528   0.234821   3.200 0.001372 ** 
resting_blood_pressure       0.012935   0.011883   1.089 0.276364    
serum_cholesterol            0.011661   0.005145   2.266 0.023431 *  
resting_electrocardiographic 0.412543   0.219372   1.881 0.060031 .  
maximum_heart_rate          -0.015739   0.010479  -1.502 0.133131    
exercise_induced_angina      0.676015   0.479114   1.411 0.158254    
oldpeak                      0.358305   0.253260   1.415 0.157136    
ST_segment_slope             0.484136   0.435215   1.112 0.265963    
number_of_major_vessels      1.008816   0.285193   3.537 0.000404 ***
thal                         0.424455   0.123147   3.447 0.000567 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 303.82  on 219  degrees of freedom
Residual deviance: 150.62  on 208  degrees of freedom
AIC: 174.62

Number of Fisher Scoring iterations: 6
```

On the same process, after removing the following variables: resting blood pressure, ST_segment slope, exercise induced angina. We ended up with the following model.



```
Call:
glm(formula = disease ~ ., family = binomial(link = "logit"),
    data = sub5)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5755  -0.5033  -0.1633   0.4718   2.3894

Coefficients:
                             Estimate Std. Error z value Pr(>|z|)
(Intercept)                 -7.777829   2.393551  -3.249 0.001156 **
sex                          1.267281   0.569309   2.226 0.026014 *
chest_pain                   0.766538   0.226292   3.387 0.000706 ***
serum_cholesterol            0.012178   0.004921   2.474 0.013343 *
resting_electrocardiographic 0.461894   0.217314   2.125 0.033548 *
maximum_heart_rate          -0.018149   0.009793  -1.853 0.063845 .
oldpeak                      0.585188   0.202437   2.891 0.003844 **
number_of_major_vessels      0.940633   0.270758   3.474 0.000513 ***
thal                         0.493129   0.118670   4.155 3.25e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 303.82  on 219  degrees of freedom
Residual deviance: 155.04  on 211  degrees of freedom
AIC: 173.04

Number of Fisher Scoring iterations: 6
```

We can write our model as follow:

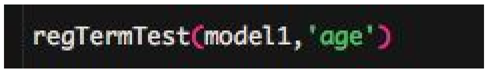
$$log(\frac{P}{1-P}) = -7.777829 + 1.267281 * sex + 0.766538 * chestPain + 0.012178 * serumCholesterol +$$

$$0.461894 * restingElectrocardiographic - 0.018149 * maximumHeartRate + 0.585188 * oldpeak$$

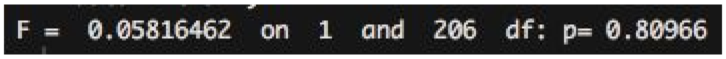
$$+ 0.940633 * numberOfMajoressels + 0.493129 * Thal$$

We could have use the wald test also to select the variables that matters. As follow on each variables.

Wald test:



```
regTermTest(model1,'age')
```



```
F =  0.05816462  on  1  and  206  df: p= 0.80966
```

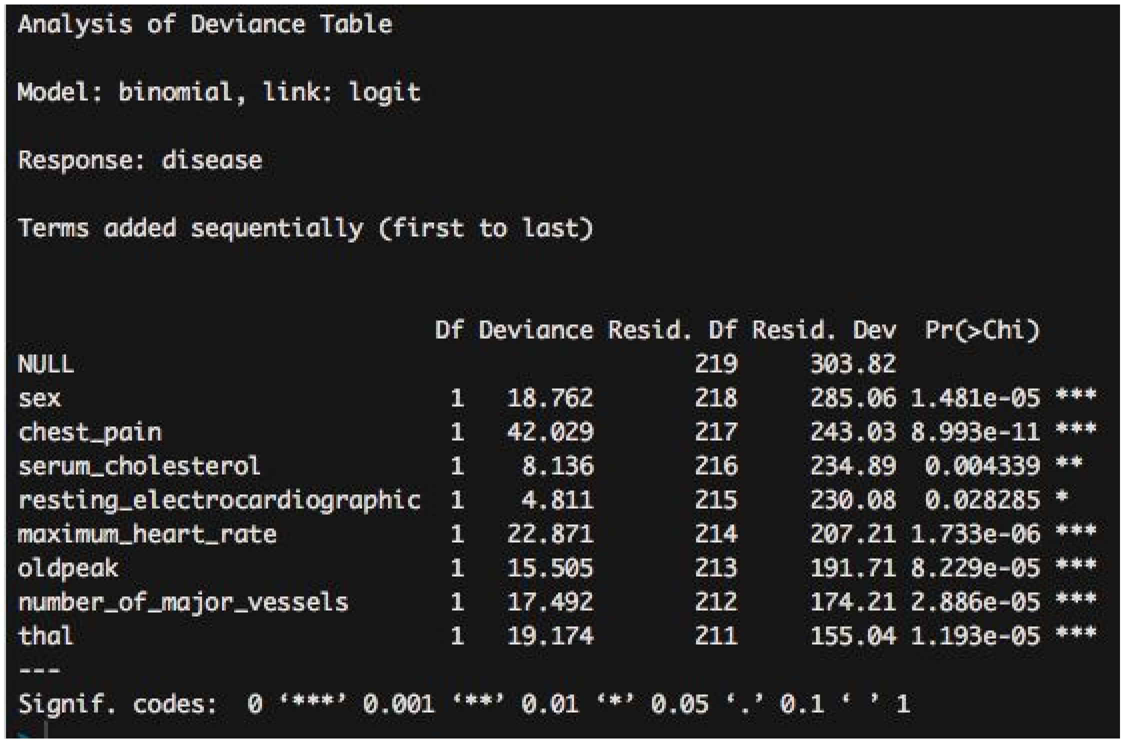
The idea is to test the hypothesis that the coefficient of an independent variable in the model is significantly different from zero.
It ends up being the same result on our dataset, because at a high N number, P-value test is considered similar to a wald test.

Interpretation of our model:

For example here we can say that:
The odd of "success" (having a heart disease) increase of 0.493129 at each increment of the thal variable. Or $e^{0.493129} = 1.63$, the odd increase of 63% each time we increment the "thal" variable.
The only feature that have a negative impact on the having a disease statment is the maximum heart rate. In fact the higher the maximum heart rate of an individual is, the less chance he will have to get an heart disease.

In addition we can run the anova() function on the model to analyze the table of deviance:
Anova is a method to compare differents groups' means with their variance. The null hypothesis is that all mean are equal. The alternative hypothesis says that there is at least one mean that is different than the others.

Anova:

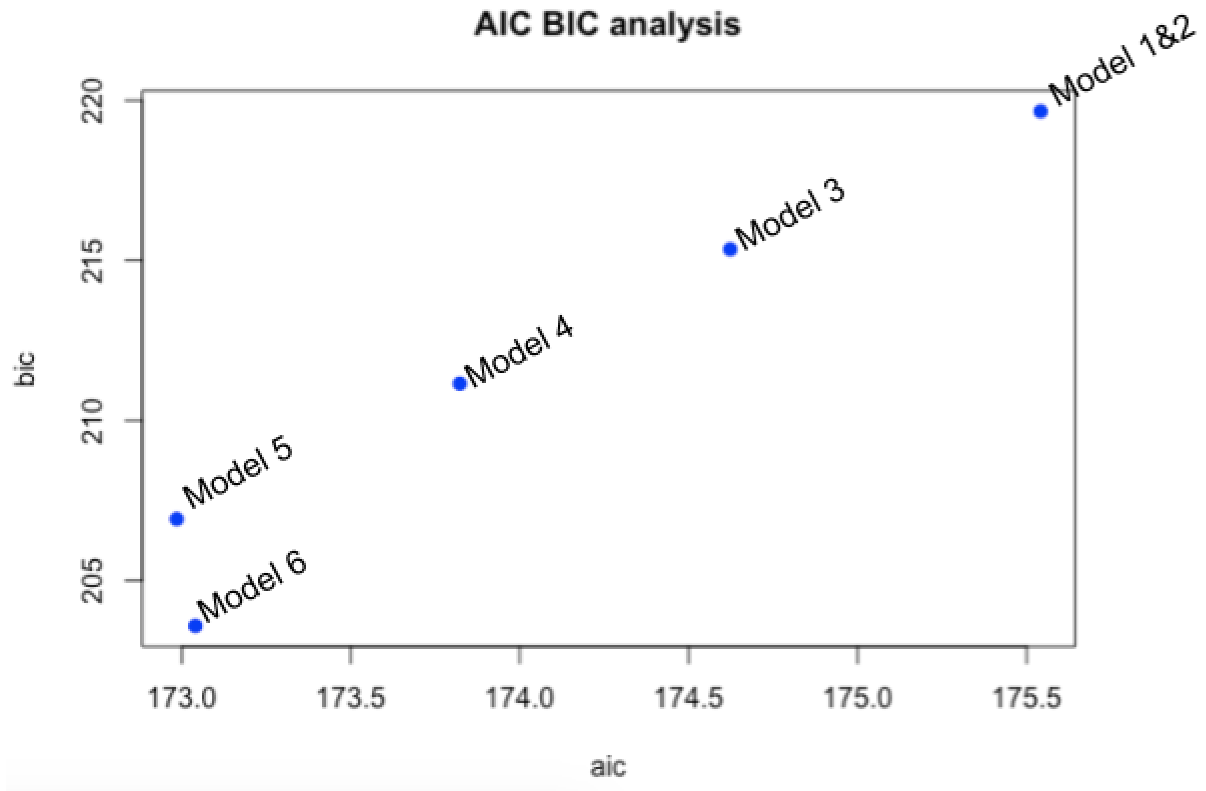


```
Analysis of Deviance Table

Model: binomial, link: logit

Response: disease

Terms added sequentially (first to last)


                             Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
NULL                                           219     303.82
sex                           1   18.762       218     285.06 1.481e-05 ***
chest_pain                    1   42.029       217     243.03 8.993e-11 ***
serum_cholesterol             1    8.136       216     234.89  0.004339 **
resting_electrocardiographic  1    4.811       215     230.08  0.028285 *
maximum_heart_rate            1   22.871       214     207.21 1.733e-06 ***
oldpeak                       1   15.505       213     191.71 8.229e-05 ***
number_of_major_vessels       1   17.492       212     174.21 2.886e-05 ***
thal                          1   19.174       211     155.04 1.193e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

The difference between the null deviance and the residual deviance shows how our model is doing against the null model (a model with only the intercept). The wider this gap, the better. We can see that resting_electrocardiographic for example does not seems to improve our model. Otherwise all the other variables are pretty equal on this aspect. Reading the P-values we can also state that all p-values are smaller than alpha, so we reject Ho.

Testing out model performances:

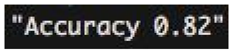
We decided to compute the AIC and BIC of every model we have built after removing the variable based on the P-value. After what we have plotted the result on the graph below.



In this plot we can see that every point is representing a model. We want to choose the model that minimizes AIC and BIC. In this case we better choose the point on the left bottom corner, which corresponds to the model 6.

Prediction with the model 6:

By setting the parameter type='response', R will output probabilities in the form of P(y=1|X). Our decision boundary will be 0.5. If P(y=1|X) > 0.5 then y = 1 otherwise y=0.

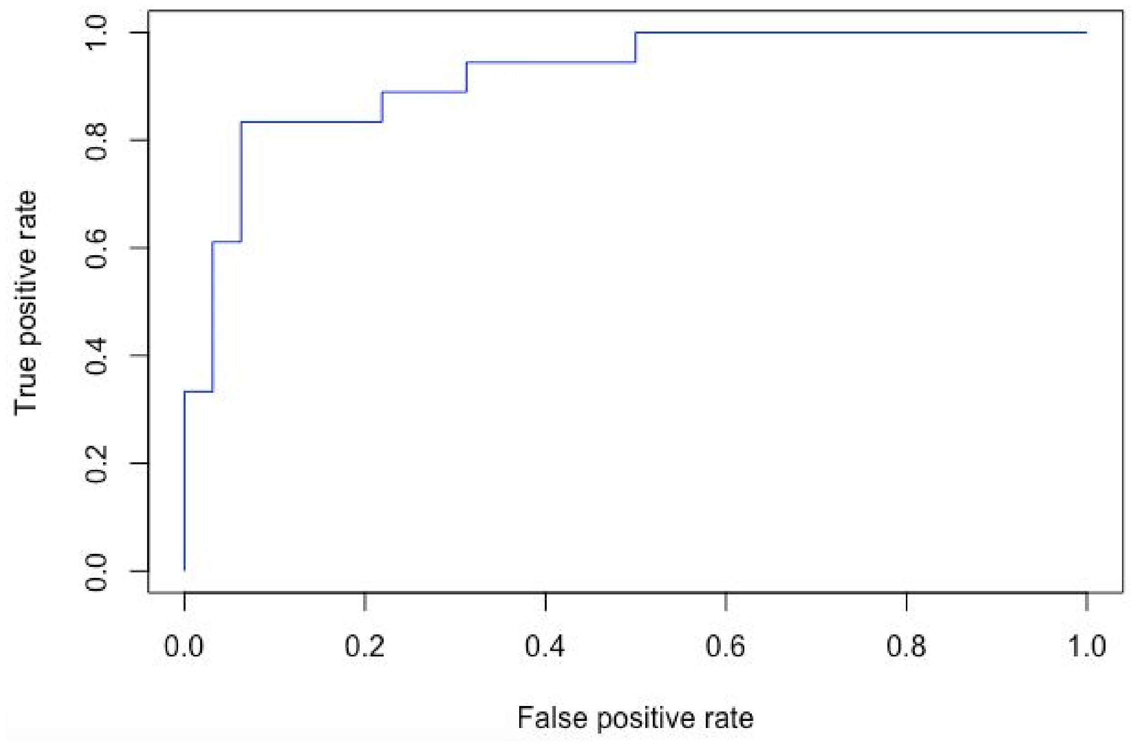


```
"Accuracy 0.82"
```

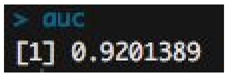
We have a 82% accuracy which is not bad. In our case as we are dealing with heart disease we rather set up a threshold value pretty high. In fact it is better to be wrong on the fact that we predict somebody that is sick but he is not in reality rather than the opposite.

ROC and AUC:

The ROC is a curve generated by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings



The AUC is the area under the ROC curve.



```
> auc
[1] 0.9201389
```

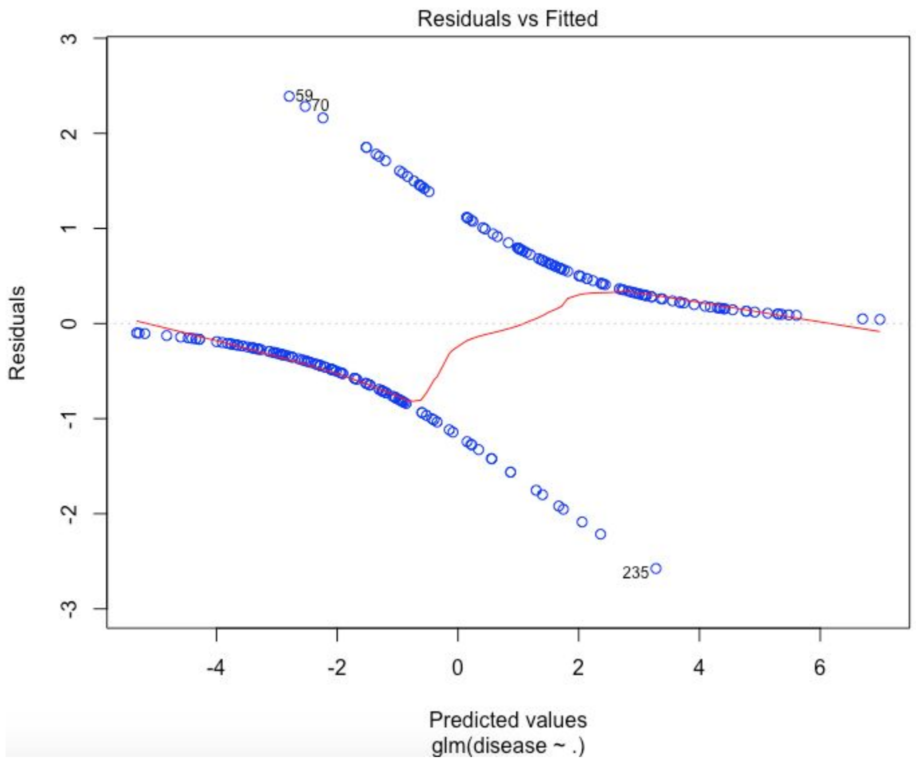
Our AUC is really close to 1. That means our model have a good predictive ability.

Residual Analysis:

<code>:

```
plot(model6,col='blue')
```

Output:



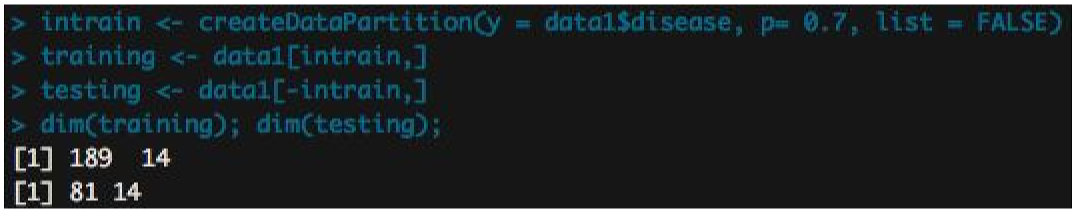
As we are doing a binary logistic regression, our label has two values. 1 or 2 in our case. This is why we have two lines.
If the true value is 1, then we always predict more, and residuals have to be negative (the bottom points) and if the true value is 2, then we underestimate, and residuals have to be positive (the top points).
Moreover we see the the red line is inside the [-1;1] interval in term of residuals.

3.5. Decision tree

To work on big datasets, we can directly use some machine learning packages. The developer community of R programming language has built the great packages Caret to make the work easier. We just need to call functions for implementing algorithms with the right parameters.
The R programming machine learning caret package(Classification And REgression Training) holds tons of functions that helps to build predictive models.
The "rplot.plot" package will help to get a visual plot of the decision tree.

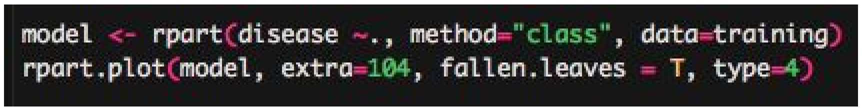
For our Hold-out validation here, after tests we decided to go on with a 0,7 ratio.

Data slicing: 70:30 ratio

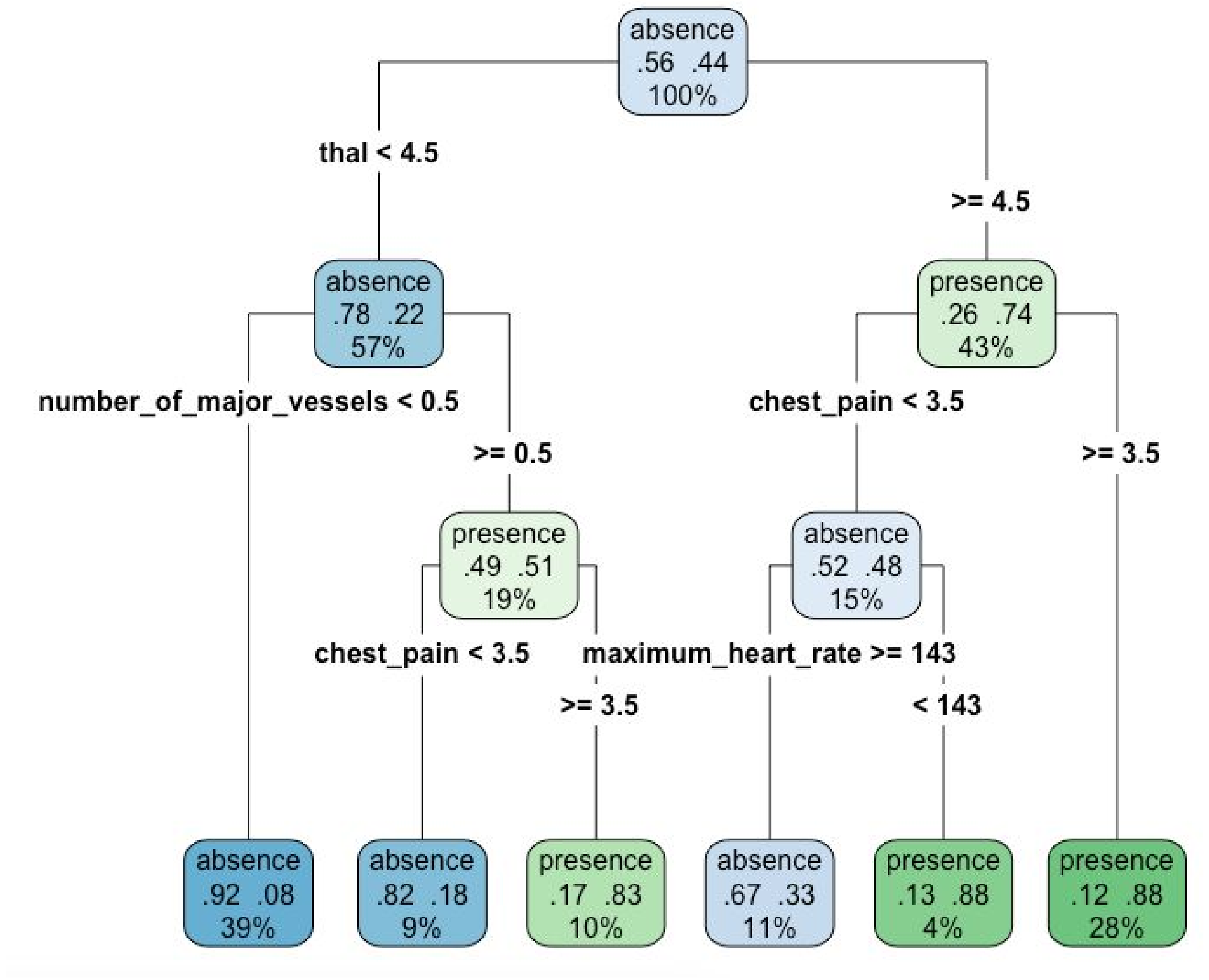


```
> intrain <- createDataPartition(y = data1$disease, p= 0.7, list = FALSE)
> training <- data1[intrain,]
> testing <- data1[-intrain,]
> dim(training); dim(testing);
[1] 189  14
[1] 81 14
```

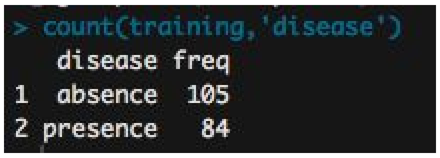
Our model and its decision tree:



```
model <- rpart(disease ~., method="class", data=training)
rpart.plot(model, extra=104, fallen.leaves = T, type=4)
```



On the first level we can read that on our training dataset there are **56%** of the people that don't have an heart disease. We can confirm that with the following <code>:

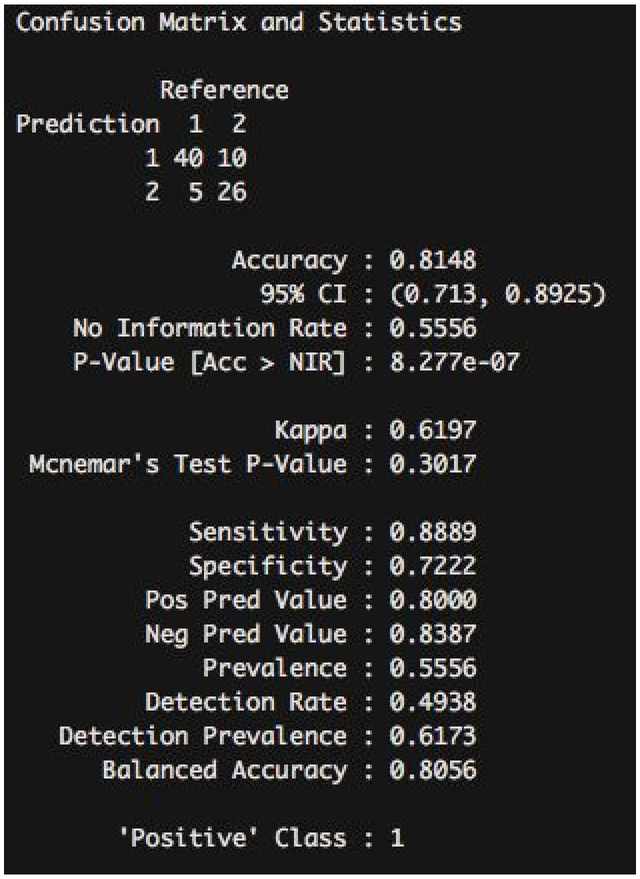


```
> count(training,'disease')
   disease freq
1  absence  105
2 presence   84
```

showing that 105 persons out of 189 does not have a heart disease.

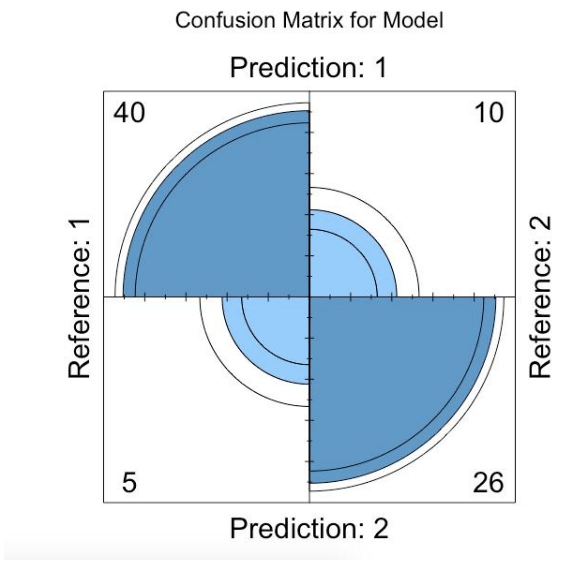
The next variable that splits our dataset in two part is the Thal variable. For any thal<4.5, **57%** of the subject doesn't have a heart disease and from those people if they have a number of blood vessel < 0.5 they have almost no chance of having a heart disease (0.08%). In the other hand, the people with thal >= 4.5 and a chest pain index pain > 3 have 28% of having an heart disease.

Computing accuracy with confusion matrix:



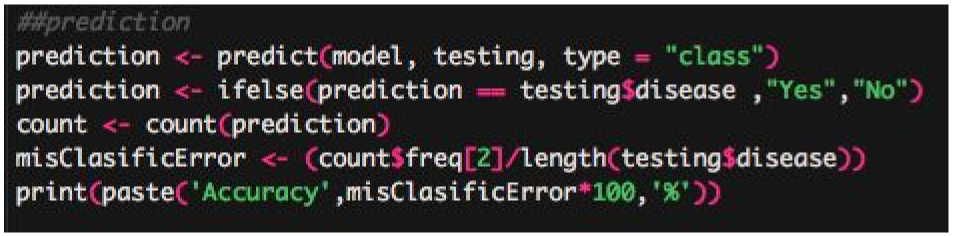
```
Confusion Matrix and Statistics

          Reference
Prediction  1  2
         1 40 10
         2  5 26

               Accuracy : 0.8148
                 95% CI : (0.713, 0.8925)
    No Information Rate : 0.5556
    P-Value [Acc > NIR] : 8.277e-07

                  Kappa : 0.6197
 Mcnemar's Test P-Value : 0.3017

            Sensitivity : 0.8889
            Specificity : 0.7222
         Pos Pred Value : 0.8000
         Neg Pred Value : 0.8387
             Prevalence : 0.5556
         Detection Rate : 0.4938
   Detection Prevalence : 0.6173
      Balanced Accuracy : 0.8056

       'Positive' Class : 1
```

Another way to visualize it in a better way is by using: fourfoldplot()

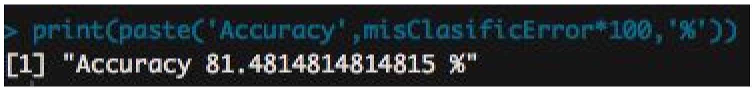


<u>Observations:</u>

We have a good accuracy of 81% (40+26/81). There is more mistakes made when the actual value was “no heart disease” and we predicted an heart disease than the opposite.

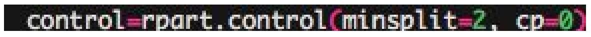


```
##prediction
prediction <- predict(model, testing, type = "class")
prediction <- ifelse(prediction == testing$disease ,"Yes","No")
count <- count(prediction)
misClasificError <- (count$freq[2]/length(testing$disease))
print(paste('Accuracy',misClasificError*100,'%'))
```



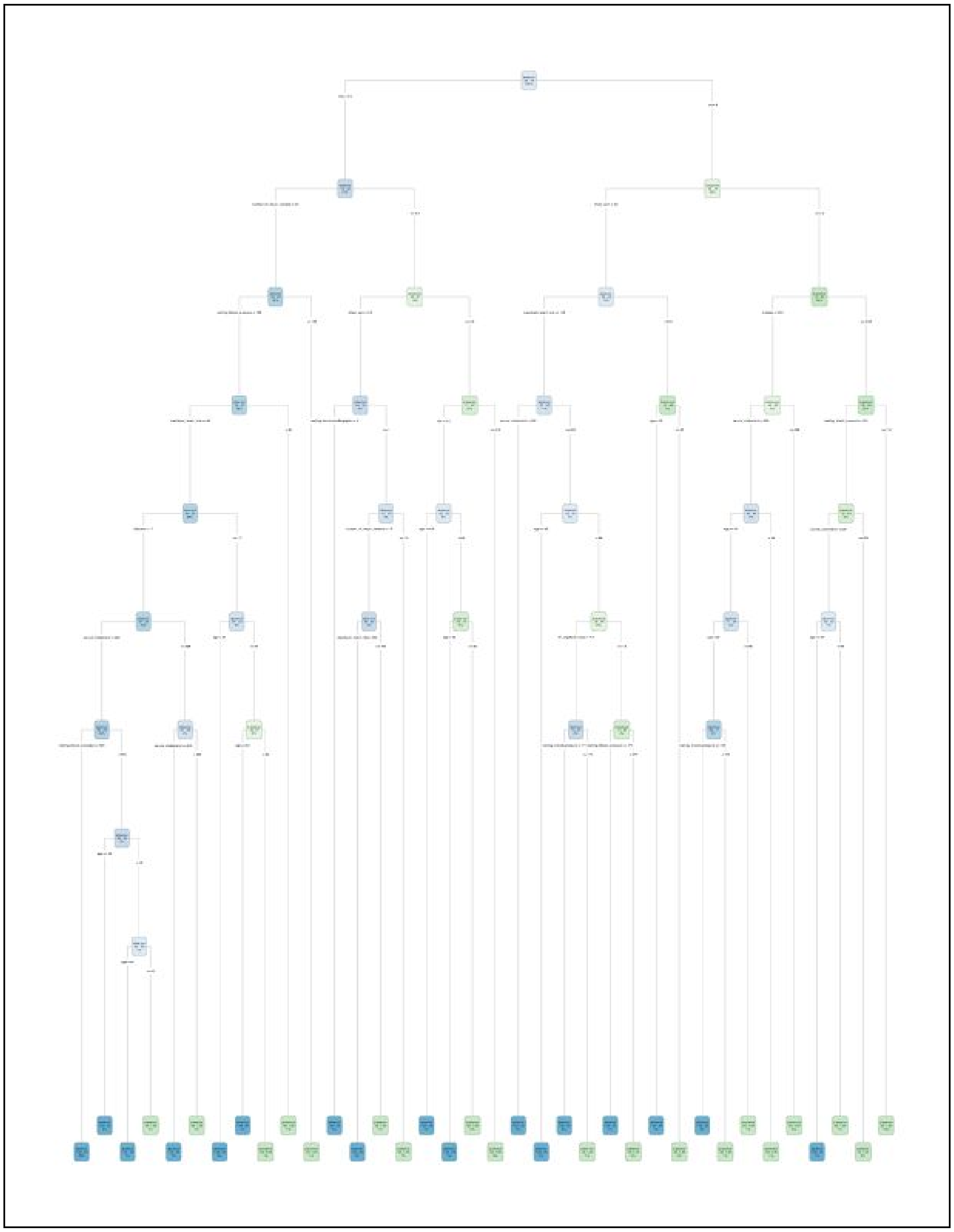
```
> print(paste('Accuracy',misClasificError*100,'%'))
[1] "Accuracy 81.4814814814815 %"
```

Just by curiosity we will plot a more complex graph and study its outcome: The first one we want to unleash is the cp parameter, this is the metric that stops splits that aren’t deemed important enough. The other one we want to open up is minsplit. We will max both out and reduce cp to zero and minsplitto 2.



```
control=rpart.control(minsplit=2, cp=0)
```

As we can read below, the outcome is not really clear and relevant, but this tree has the advantages of being more precise as it is based on more parameters.

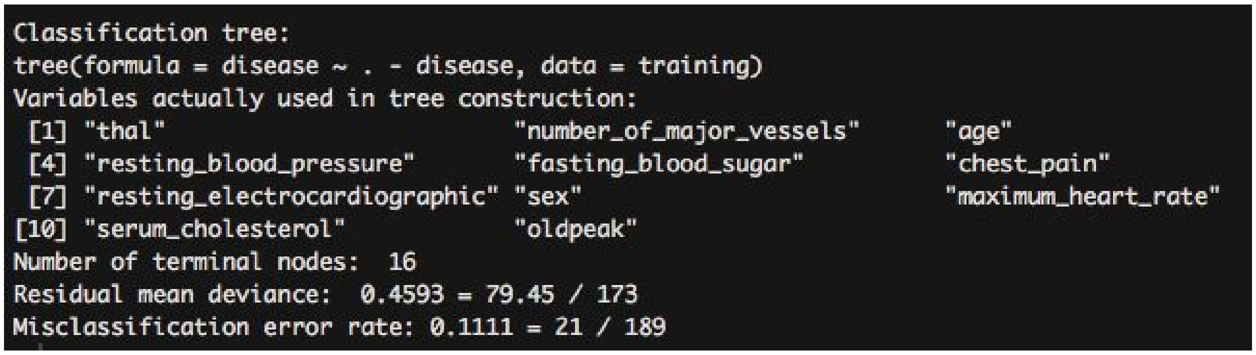


Using tree() method:

<code>:

```
tree.heart <- tree(disease~ . ,training)
```

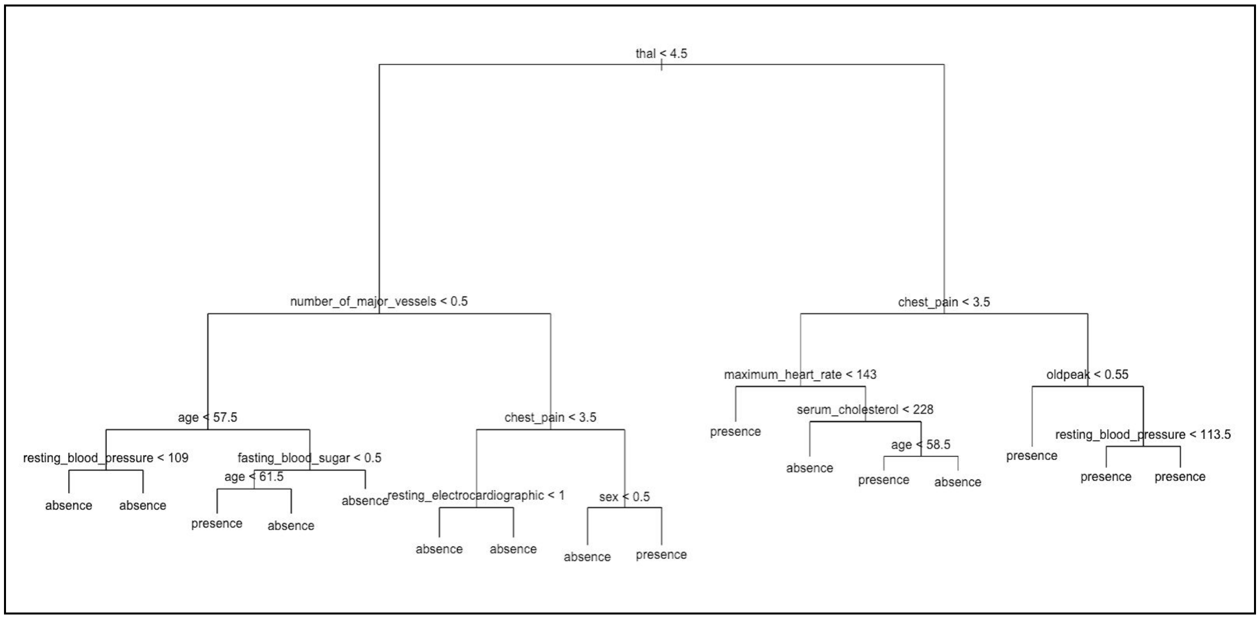
Output:



```
Classification tree:
tree(formula = disease ~ . - disease, data = training)
Variables actually used in tree construction:
 [1] "thal"                          "number_of_major_vessels"      "age"
 [4] "resting_blood_pressure"        "fasting_blood_sugar"          "chest_pain"
 [7] "resting_electrocardiographic"  "sex"                          "maximum_heart_rate"
[10] "serum_cholesterol"             "oldpeak"
Number of terminal nodes:  16
Residual mean deviance:  0.4593 = 79.45 / 173
Misclassification error rate: 0.1111 = 21 / 189
```

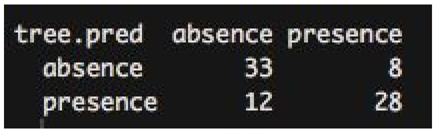
We have an error of 11%. In this case, we are using 11 features. We are not using the slope of the peak and exercise induced angina.

Decision tree:



The most important indicator nevertheless remains being Thal.
In order to properly evaluate the performance of our classification tree on these data, we must estimate the test error rather than simply computing the training

We use the function predict( ), and print out the confusion matrix.

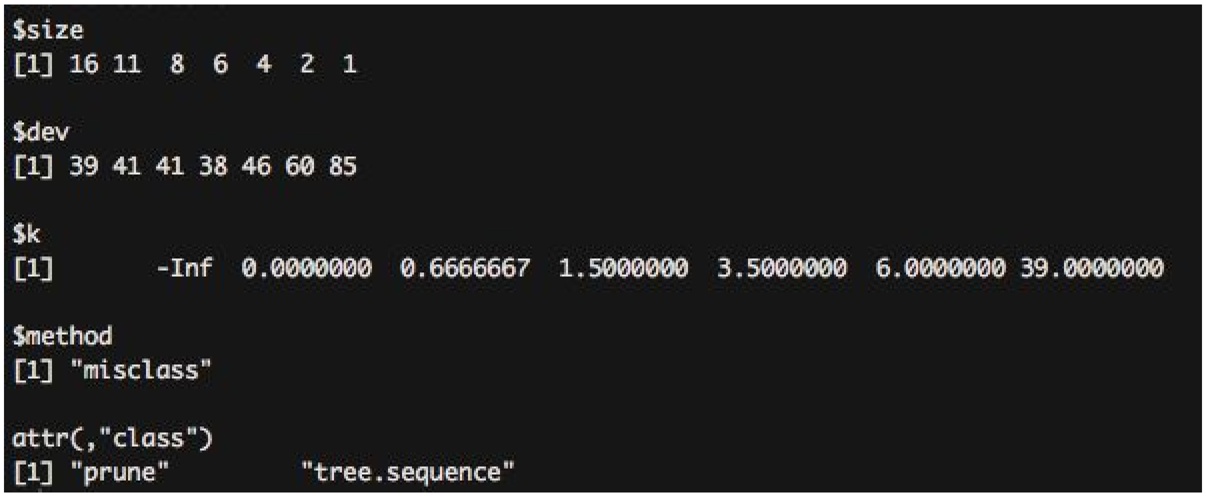


```
tree.pred  absence presence
  absence       33        8
  presence      12       28
```

Our model have 33+28 / 81 = 75%.

<u>Model selection with cross validation:</u>

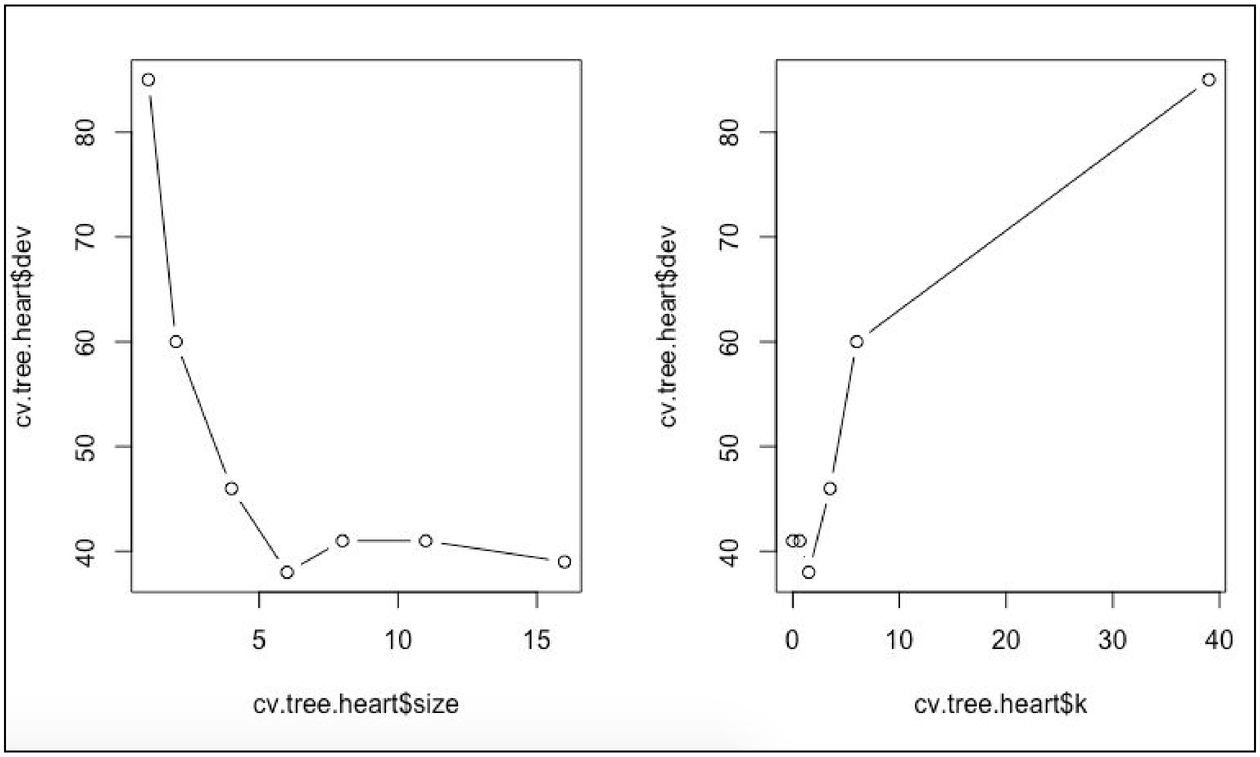
The function cv.tree() will perform the cross validation in order to determine the optimal level of complexity.We use the argument FUN=prune.misclass in order to indicate that we want the classification error to guide the cross validation.



```
$size
[1] 16 11  8  6  4  2  1

$dev
[1] 39 41 41 38 46 60 85

$k
[1]      -Inf  0.0000000  0.6666667  1.5000000  3.5000000  6.0000000 39.0000000

$method
[1] "misclass"

attr(,"class")
[1] "prune"         "tree.sequence"
```

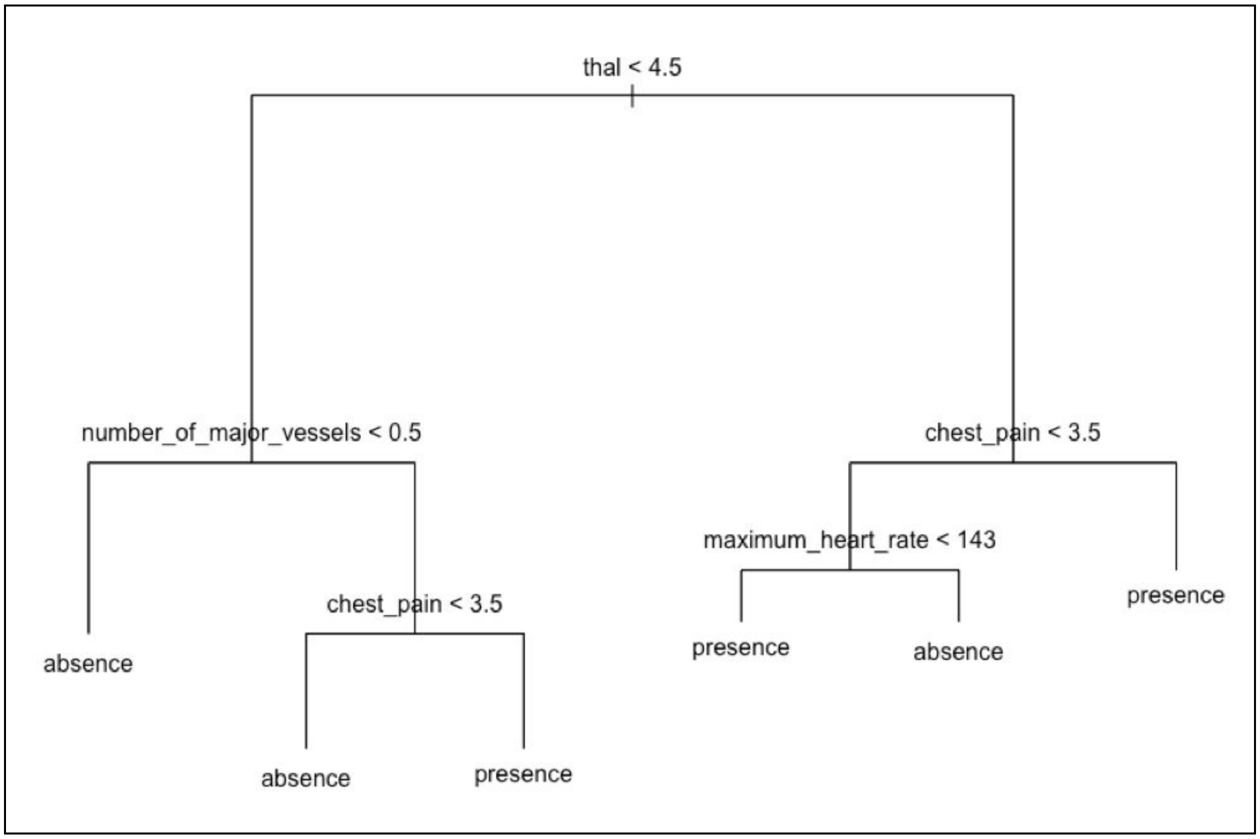
Dev corresponds to the cross validation error rate. The tree with 6 finals nodes is the one with the lowest error.

We now plot the error rate as a function of both size and K (value of cost-complexity parameter) .



We can now apply the prune.misclass() in order to prune the tree and obtain the 6 nodes tree.

This is our new decision tree with the 6 final leaves:



4. Conclusion

4.1. Our selected model

The model we prefer to select, among all the classifiers we used, is the Logistic Regression Model.

Indeed, despite it not having the maximum accuracy we have compared to the others (that title belongs to our KNN classifier), it is still very close to it.

But in addition to its good performance, using a logistic Regression would provide us with classification probabilities, which is good for the tool we aim to build.
We would like to provide a tool that tells the patient not only if he is sick or not, but also what he could change to be healthy.

Therefore, showing this probability value vary when the user inputs corrected values would be good from a medical standpoint.

Logistic Regression

82% accuracy

Good performance, low variance, and provides probabilities.

4.2. Problems we faced

We faced several annoying issues in our project.

For example, we had formatting issues when using several functions that wouldn't work unless the argument used was a certain type. And the error messages were often unclear so some digging and lost time had to occur before making progress (like when we used the Naive Bayes classifier for example).

The decision tree building could also prove tricky to understand as the output result in R didn't look the same according to the tree building function we used (and we tried many).

5. Further work for improvements

We think it would be interesting to implement this project with a working UI, in a browser or in an app for example. Something simple and intuitive enough for medical professionals or curious internet roamers to try and use.

Also, we had the idea of actually predicting a patient's health evolution in the future using TimeSeries.
You might wonder how we could do that if we do not have time data ? An idea we had was to use the age attribute: we would simulate our patient's health features in time by using other patients from the training data at different age (this would also include some data processing, to discretize the age attribute into appropriate bins). Then we could build a Time Series model and predict the patient's future.

These are one simple improvement and one more complex improvement to eventually implement in our project to improve it.
We hope however that you will still find some worth in our work as it currently is. Thanks for reading up to this point.