Homework_SL_1

Alberto De Benedittis

30/3/2021

Exercise 1

Load the data

As usual, the first thing to do is to import the file. We simply read it through the `read.csv()` function. Indeed, although the file is in a *txt* format, we notice that the values inside the document are separated by a comma so considering the document as a *csv* is appropriate. Then we associate this data set to a new variable in order to manipulate it without modifying the original data set. This is useful in the case we make some mistakes.

```
df <-  read.csv('disease.txt')
```

Perform basic data exploration

```
dim(df)
```

```
## [1] 392   9
```

```
names(df)
```

```
## [1] "C1"      "C2"      "C3"      "C4"      "C5"      "C6"      "C7"
## [8] "C8"      "disease"
```

```
summary(df)
```

```
##        C1               C2              C3              C4
##  Min.   : 0.000   Min.   : 56.0   Min.   : 24.00   Min.   : 7.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.:21.00
##  Median : 2.000   Median :119.0   Median : 70.00   Median :29.00
##  Mean   : 3.301   Mean   :122.6   Mean   : 70.66   Mean   :29.15
##  3rd Qu.: 5.000   3rd Qu.:143.0   3rd Qu.: 78.00   3rd Qu.:37.00
##  Max.   :17.000   Max.   :198.0   Max.   :110.00   Max.   :63.00
##        C5               C6              C7              C8
##  Min.   : 14.00   Min.   :18.20   Min.   :0.0850   Min.   :21.00
##  1st Qu.: 76.75   1st Qu.:28.40   1st Qu.:0.2697   1st Qu.:23.00
##  Median :125.50   Median :33.20   Median :0.4495   Median :27.00
##  Mean   :156.06   Mean   :33.09   Mean   :0.5230   Mean   :30.86
##  3rd Qu.:190.00   3rd Qu.:37.10   3rd Qu.:0.6870   3rd Qu.:36.00
##  Max.   :846.00   Max.   :67.10   Max.   :2.4200   Max.   :81.00
##     disease
##  Min.   :0.0000
##  1st Qu.:0.0000
##  Median :0.0000
##  Mean   :0.3316
##  3rd Qu.:1.0000
##  Max.   :1.0000
```

```
head(df)
```

```
##   C1  C2 C3 C4  C5   C6    C7 C8 disease
## 1  3  83 58 31  18 34.3 0.336 25       0
## 2  0  94 70 27 115 43.5 0.347 21       0
## 3  0 128 68 19 180 30.5 1.391 25       1
## 4  1 153 82 42 485 40.6 0.687 23       0
## 5  0  95 80 45  92 36.5 0.330 26       0
## 6  0 126 84 29 215 30.7 0.520 24       0
```

The second thing to do is to understand what we have. So we need to explore the data set and understand its characteristic and the kind of data that we have.
From this basic data exploration we get that: the data set is made by 392 rows and 9 columns; we have 8 numerical categories (*C1* to *C8*); the last column (disease) although looks numerical has to be transformed into a categorical one; we do not have NAs. Moreover, with the command `summary()` we are able to understand some basics information regarding the numerical variables such as the mean, the median and the distribution of the quartiles.

We could transform the last column of our data-set into a categorical variable to make it simpler our future analysis

```
df$disease <- as.factor(df$disease)

#contrasts(df$disease)
df$disease <- as.factor(ifelse(df$disease== '1', 'yes','no'))
```

Since we have changed the data set; I think that it is useful to do another `summary()` in order to have information about the number of people with the disease and the ones without.

```
summary(df)
```

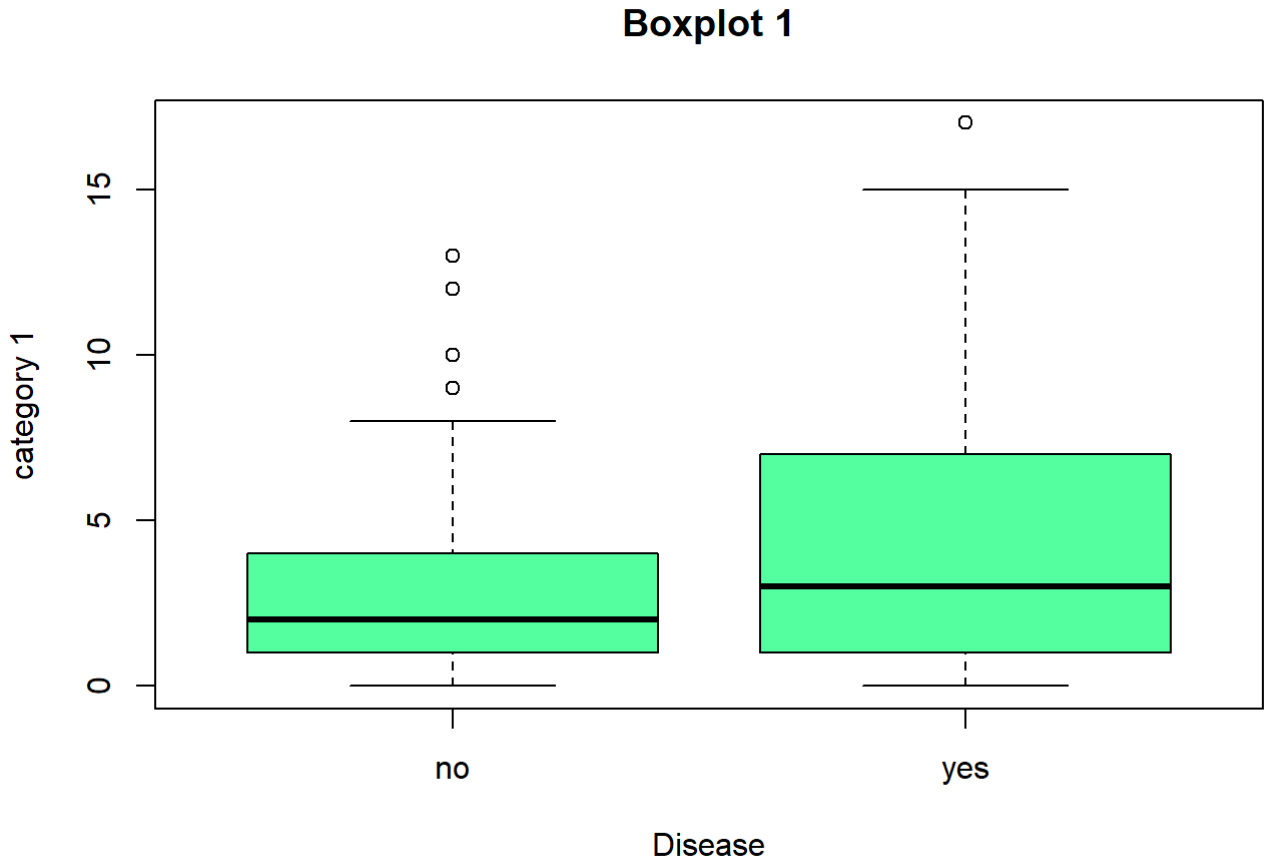
```
##        C1               C2              C3               C4
##  Min.   : 0.000   Min.   : 56.0   Min.   : 24.00   Min.   : 7.00
##  1st Qu.: 1.000   1st Qu.: 99.0   1st Qu.: 62.00   1st Qu.:21.00
##  Median : 2.000   Median :119.0   Median : 70.00   Median :29.00
##  Mean   : 3.301   Mean   :122.6   Mean   : 70.66   Mean   :29.15
##  3rd Qu.: 5.000   3rd Qu.:143.0   3rd Qu.: 78.00   3rd Qu.:37.00
##  Max.   :17.000   Max.   :198.0   Max.   :110.00   Max.   :63.00
##        C5               C6              C7               C8        disease
##  Min.   : 14.00   Min.   :18.20   Min.   :0.0850   Min.   :21.00   no :262
##  1st Qu.: 76.75   1st Qu.:28.40   1st Qu.:0.2697   1st Qu.:23.00   yes:130
##  Median :125.50   Median :33.20   Median :0.4495   Median :27.00
##  Mean   :156.06   Mean   :33.09   Mean   :0.5230   Mean   :30.86
##  3rd Qu.:190.00   3rd Qu.:37.10   3rd Qu.:0.6870   3rd Qu.:36.00
##  Max.   :846.00   Max.   :67.10   Max.   :2.4200   Max.   :81.00
```

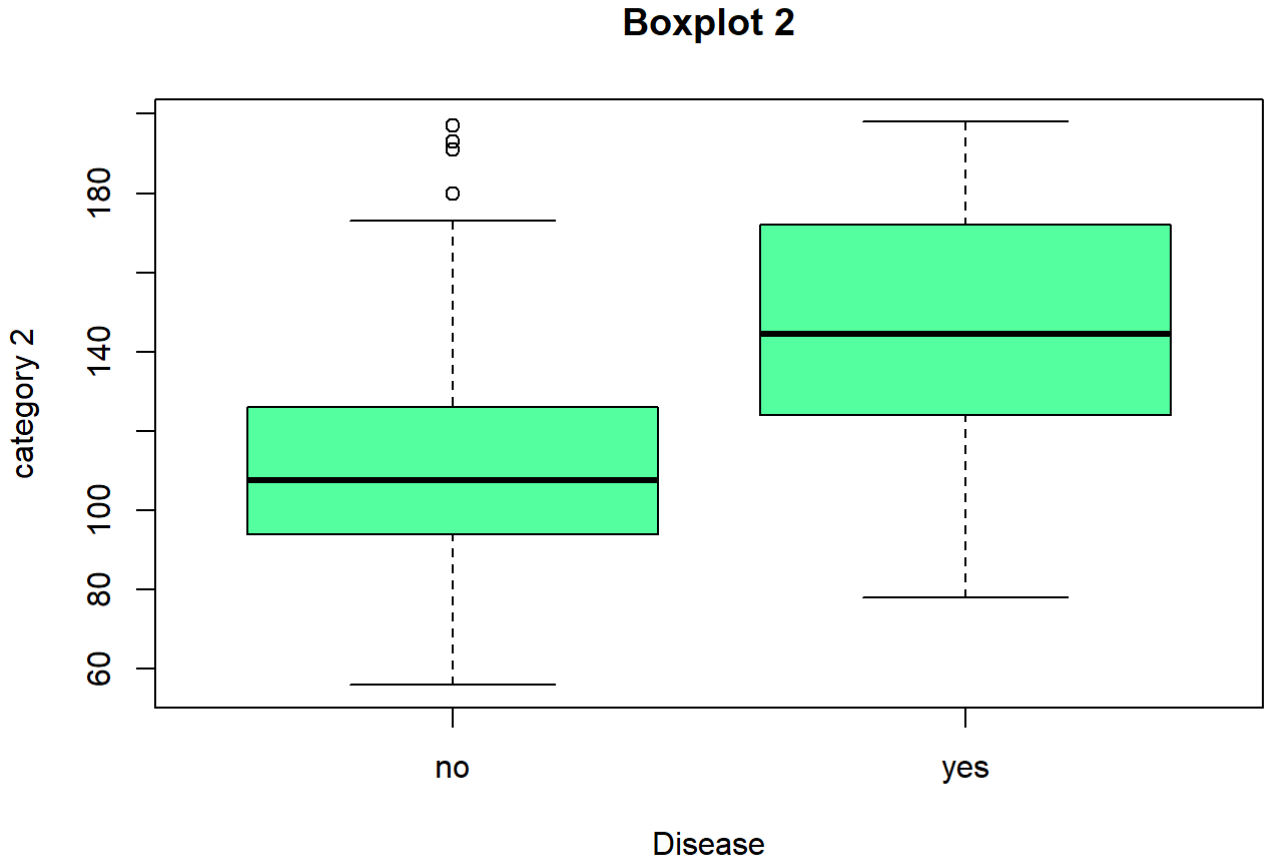
```
prop.table(summary(df$disease))
```

```
##        no       yes
## 0.6683673 0.3316327
```

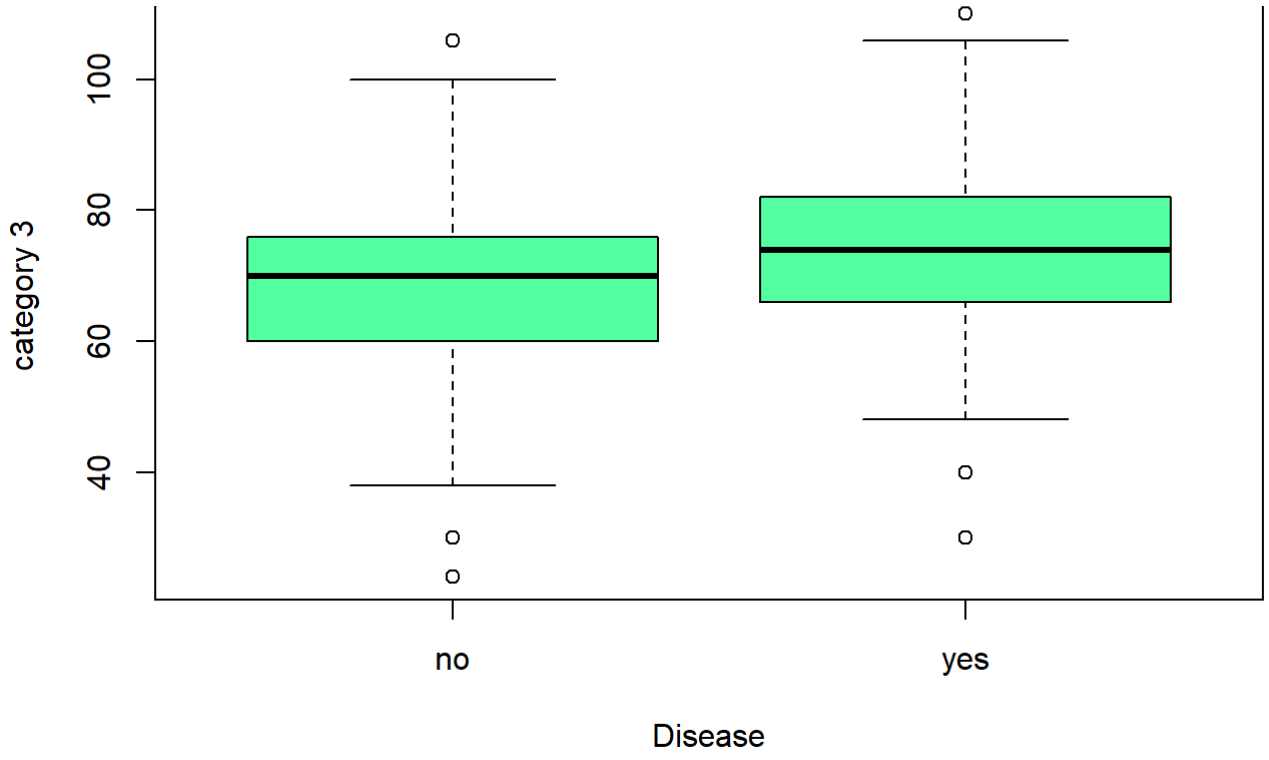
From this new summary we get that the number of people with the disease is one third of the population. Once, we have the variable disease as categorical we can ask ourselves how the values of the other variables change according to have or not the disease. Hence, we can call the function `boxplot()` for each of the numerical variable as see if we can starting to spot some useful information.

```
for (i in 1:8){
  title <- sprintf("Boxplot %d",i)
  boxplot(df[,i] ~ df$disease, main = title, xlab = 'Disease', ylab =  sprintf("category %d",
i), col = 'seagreen1')
}
```

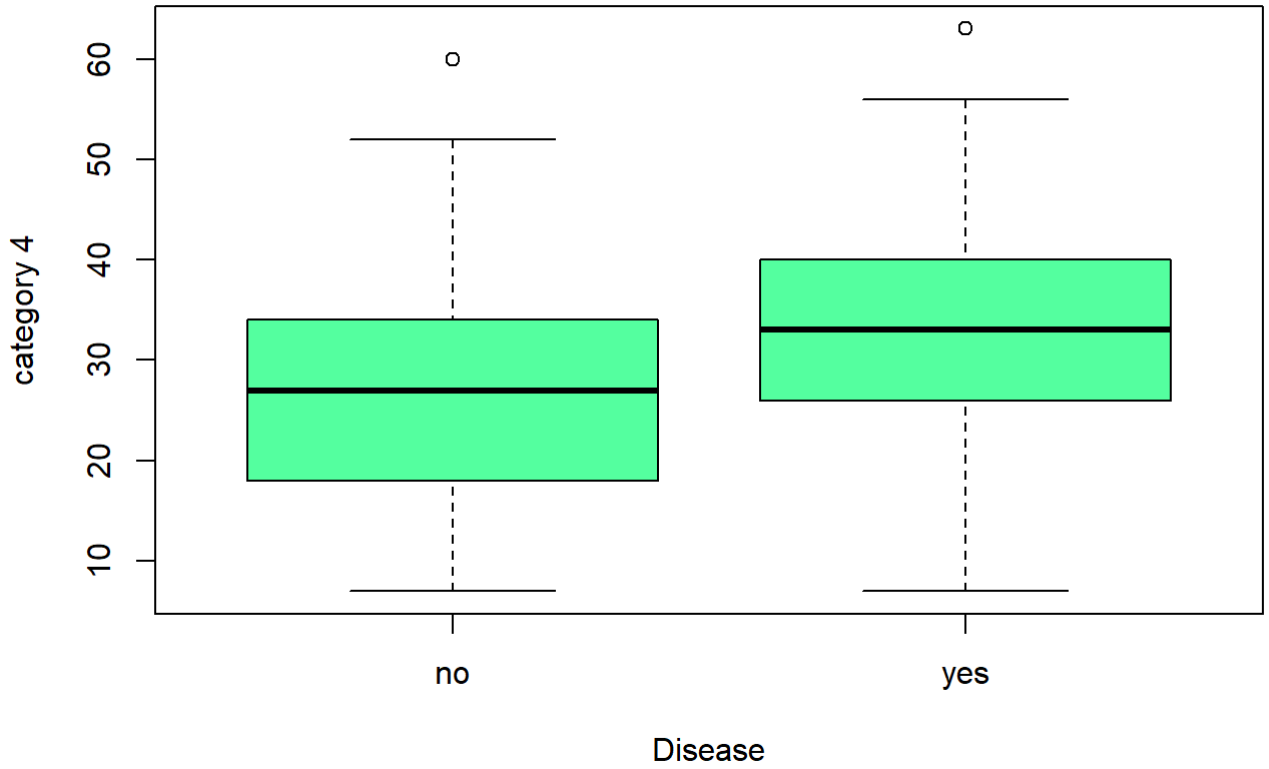

Boxplot 1
category 1
0
5
10
15
no
yes
Disease

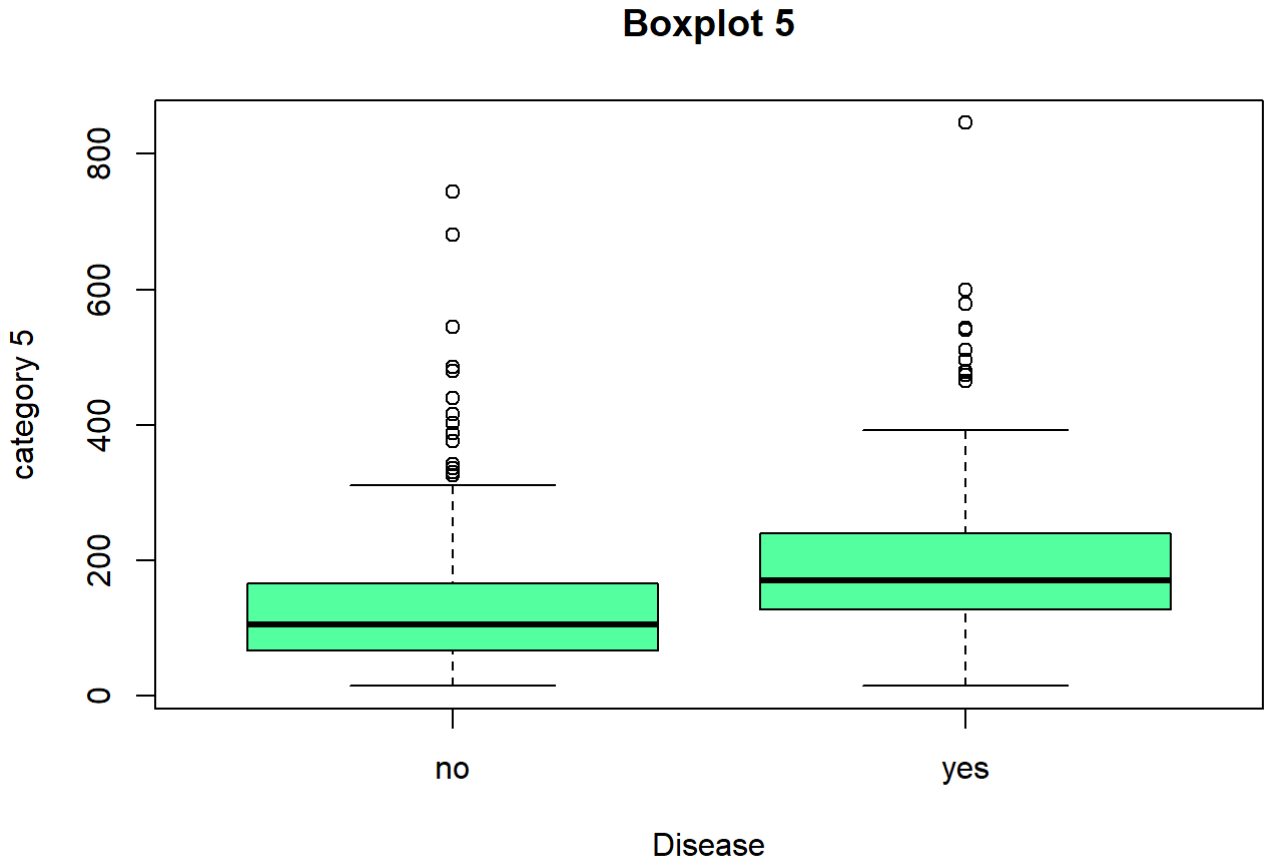

Boxplot 2
category 2
60
80
100
140
180
no
yes
Disease

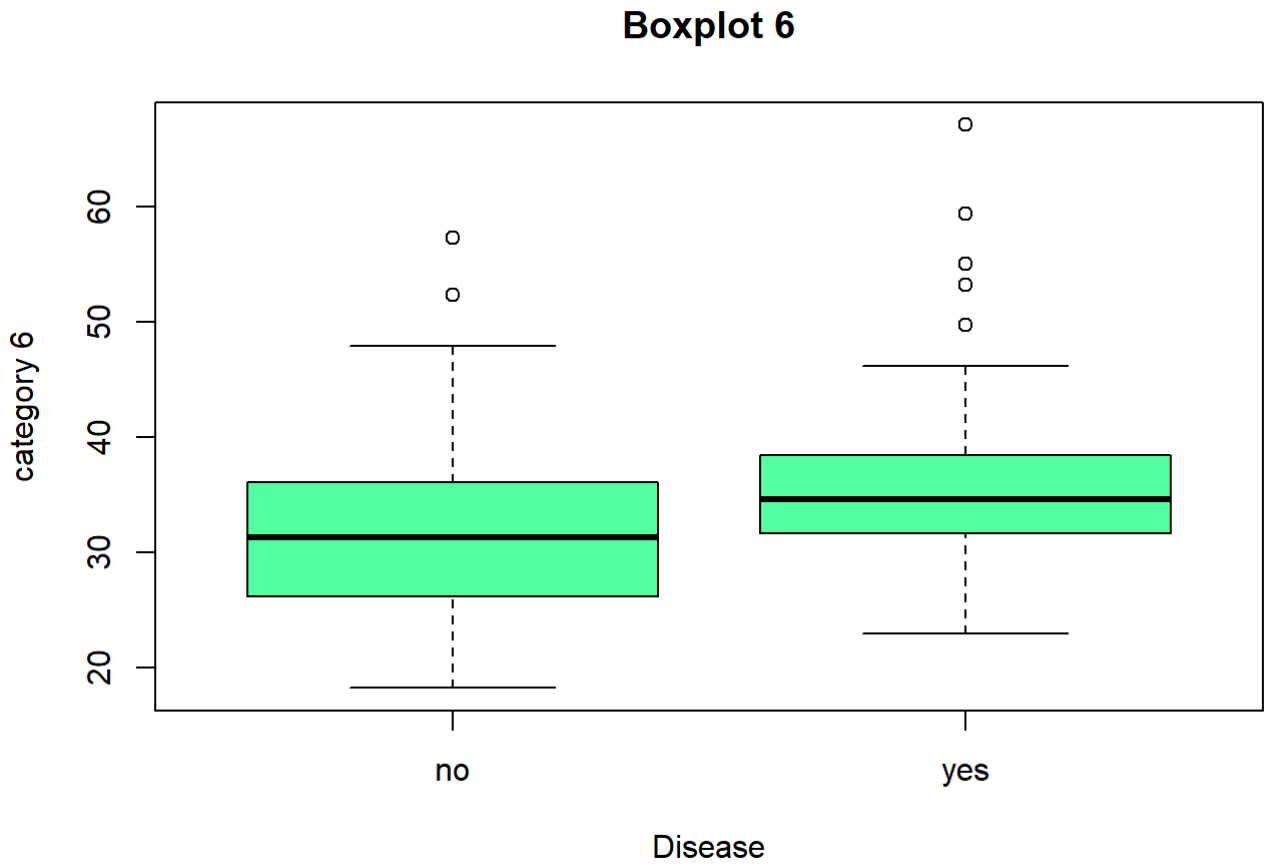

Boxplot 3

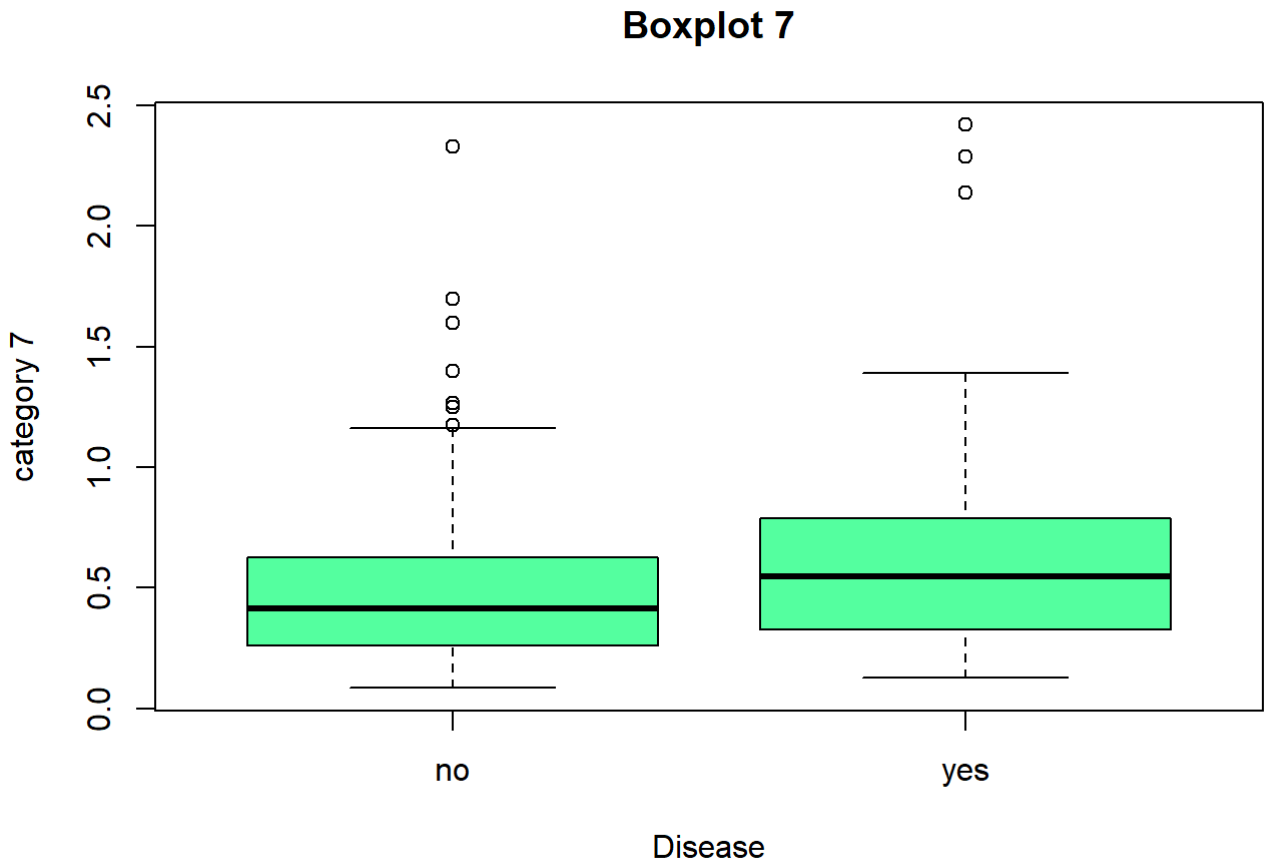

category 3
40
60
80
100
no
yes
Disease

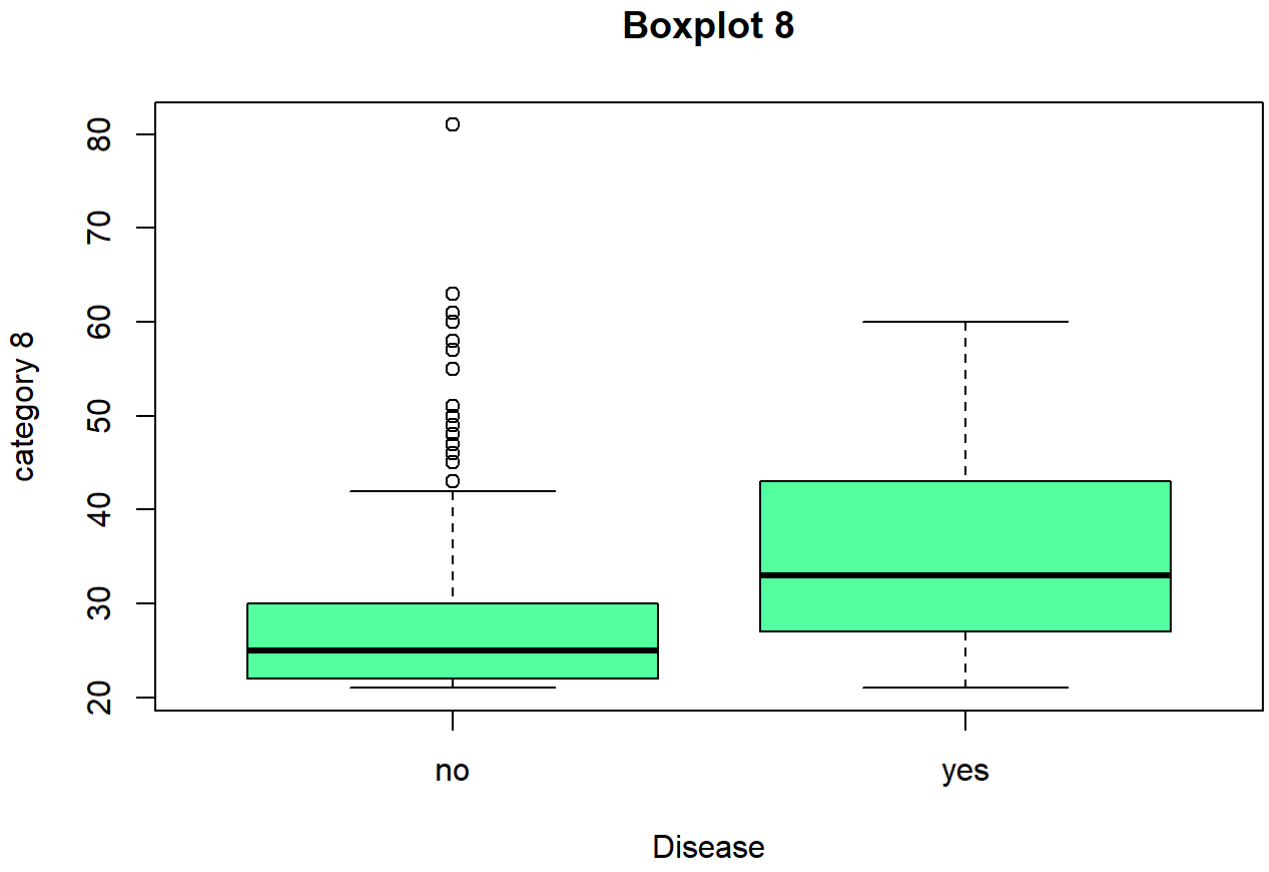
Boxplot 4


category 4
10
20
30
40
50
60
no
yes
Disease


Boxplot 5
category 5
0
200
400
600
800
no
yes
Disease


Boxplot 6
category 6
20
30
40
50
60
no
yes
Disease





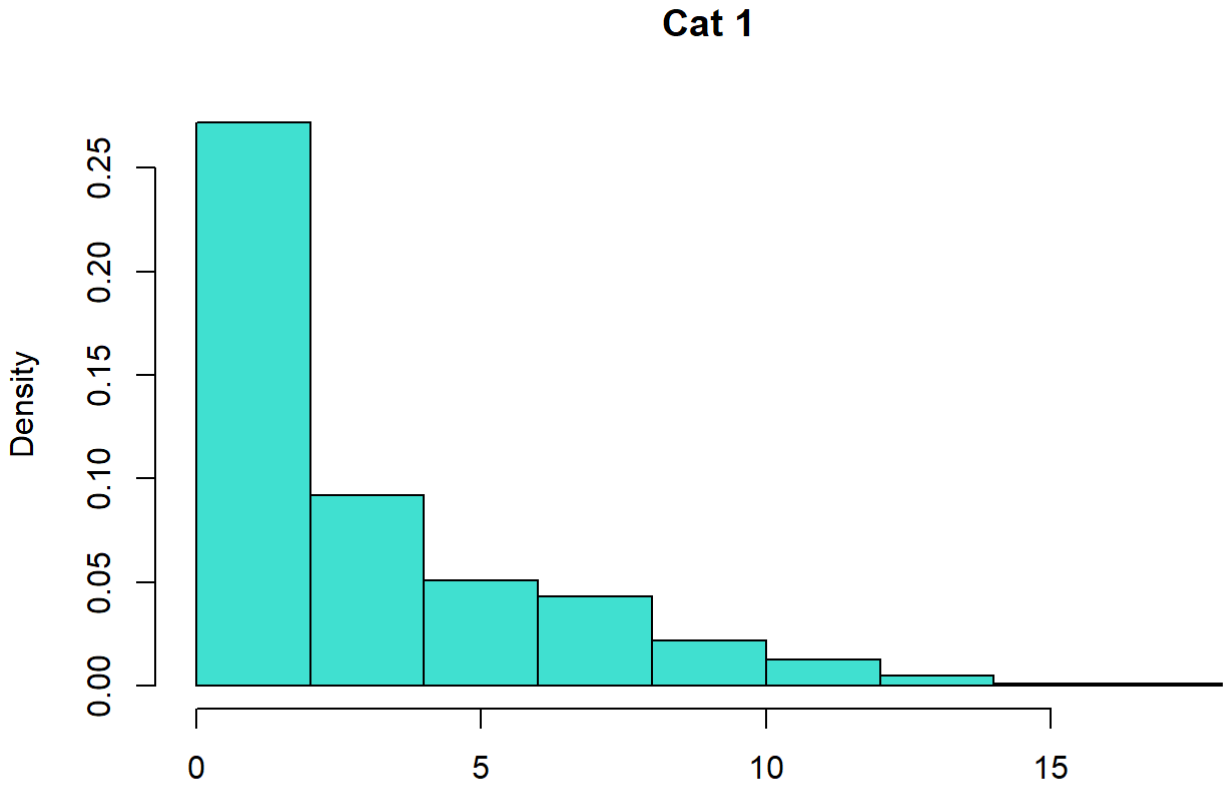
In these box-plots we can see the the differences of the values for each of the numerical categories (*C1* to *C8*) according the condition of having or not the disease. I think that it is interesting considering the second box-plots, here there is a not irrelevant difference between the values of the yes condition compared to the one of

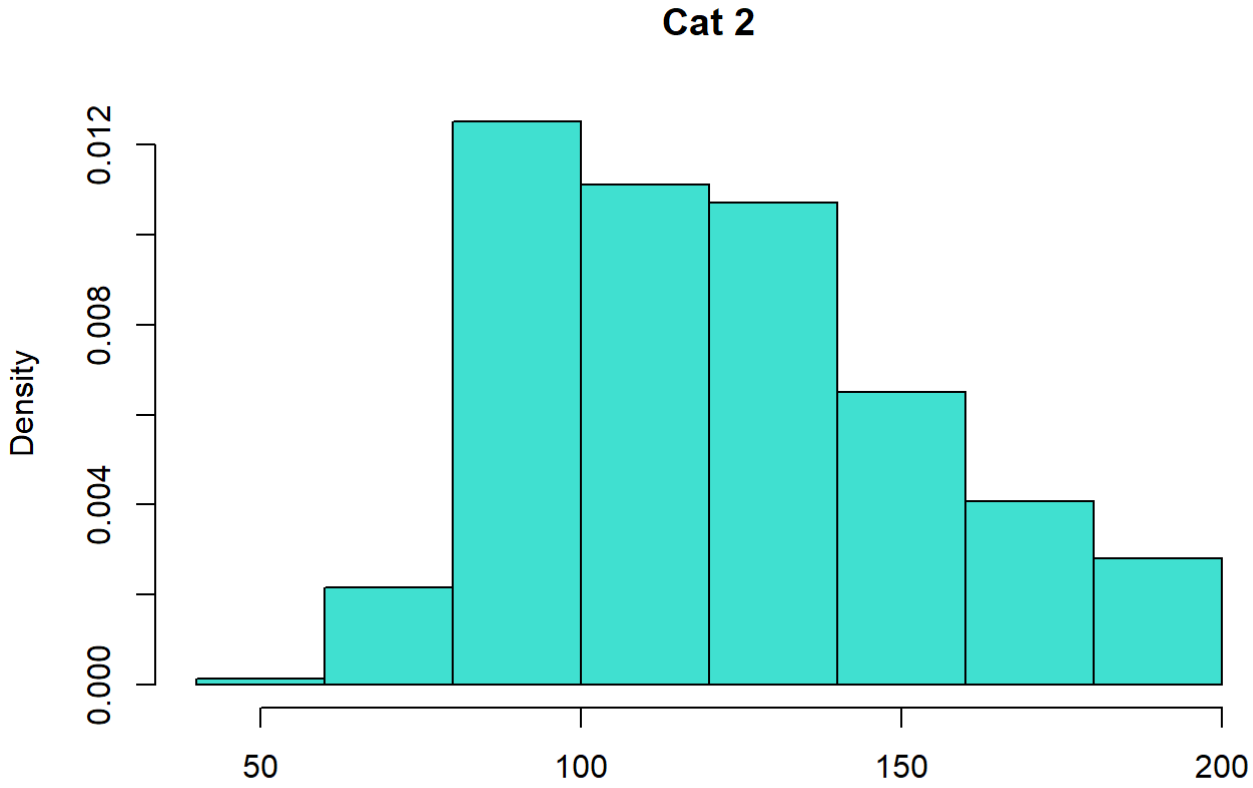
the no condition. Hence, *C2* has discriminative power on the disease because the median between the two categories are quite different. This could suggest that *C2* is an important factor in determining the presence of the disease. A similar assumption could also be made for *C1* and *C8*. However, this are just assumption and it is still difficult to assess the impact of the *C-variables* on the disease.

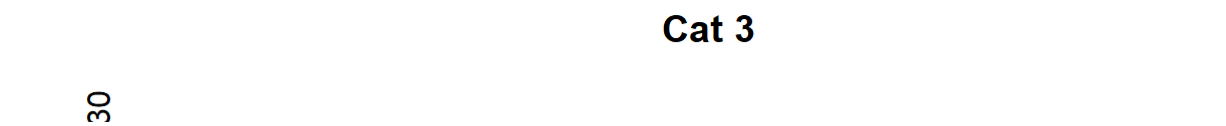
Check the distribution of attributes

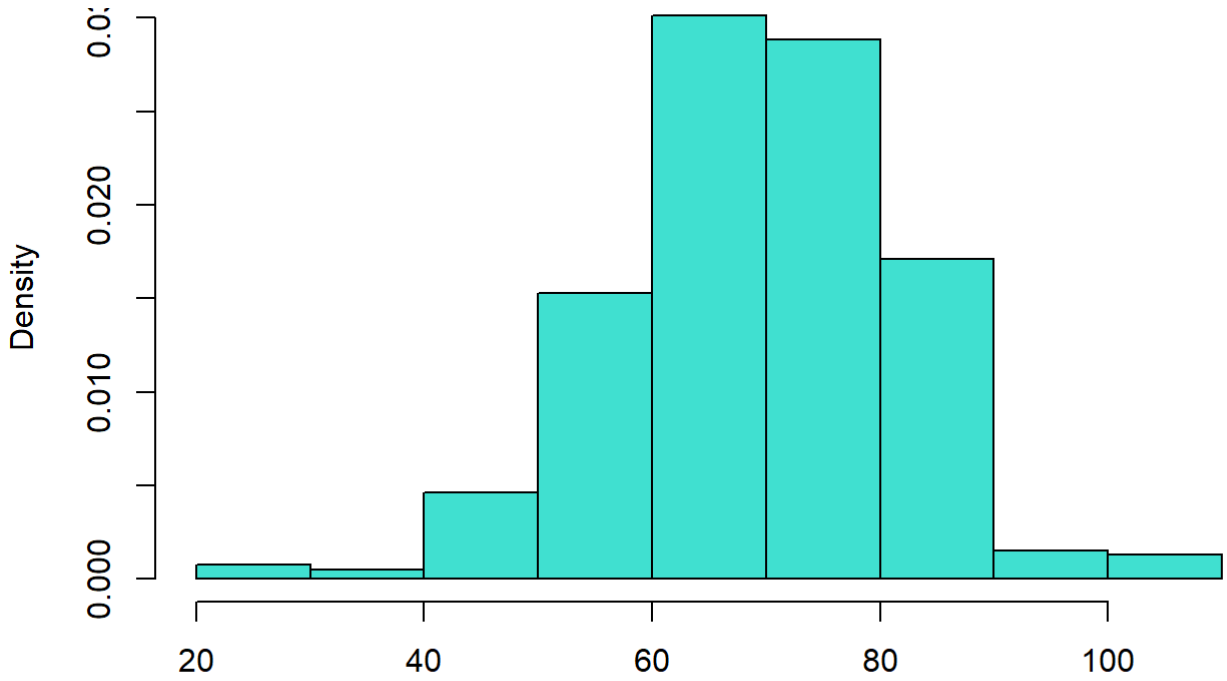
Now we want to understand the distribution of the values for each numerical variable (*C1* - *C8*). To do so we simply use the function `hist()` .

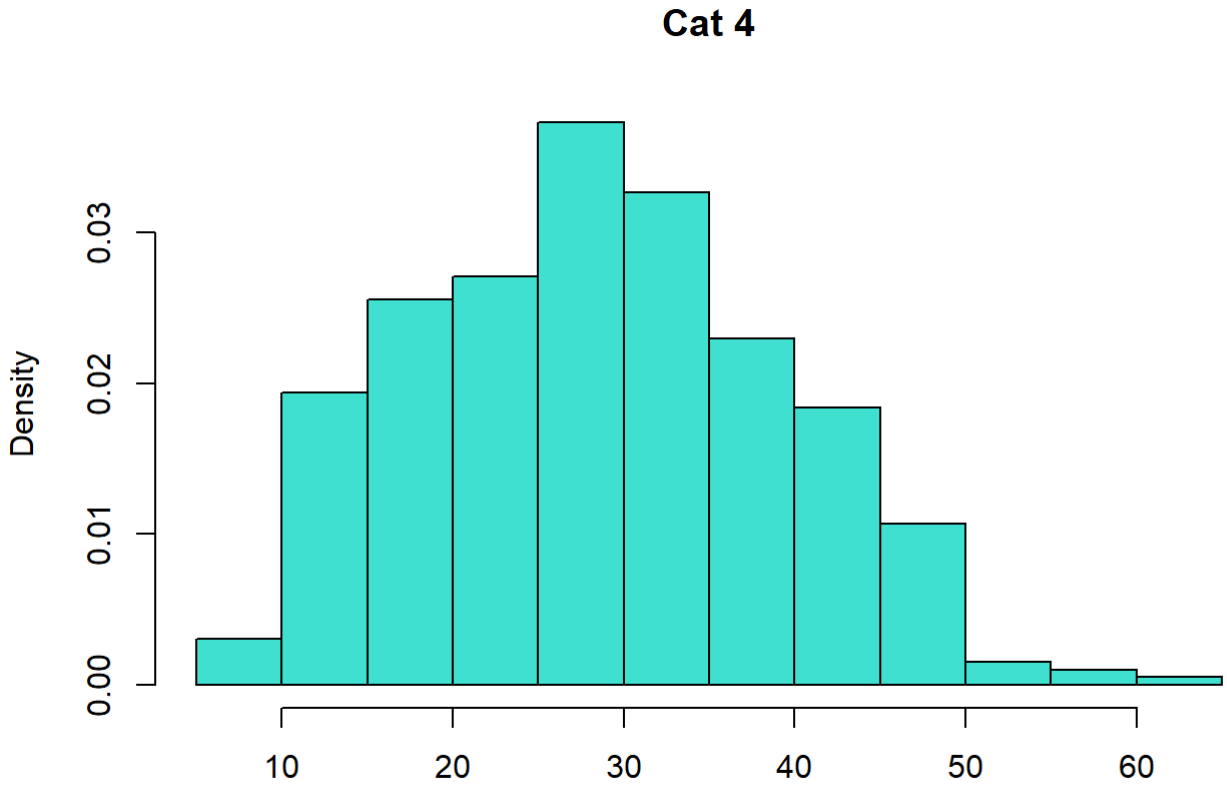
```
for (i in 1:8){
  title <- sprintf("Cat %d",i)
  hist(df[,i], main = title, freq = F, col='turquoise', xlab = '')
}
```

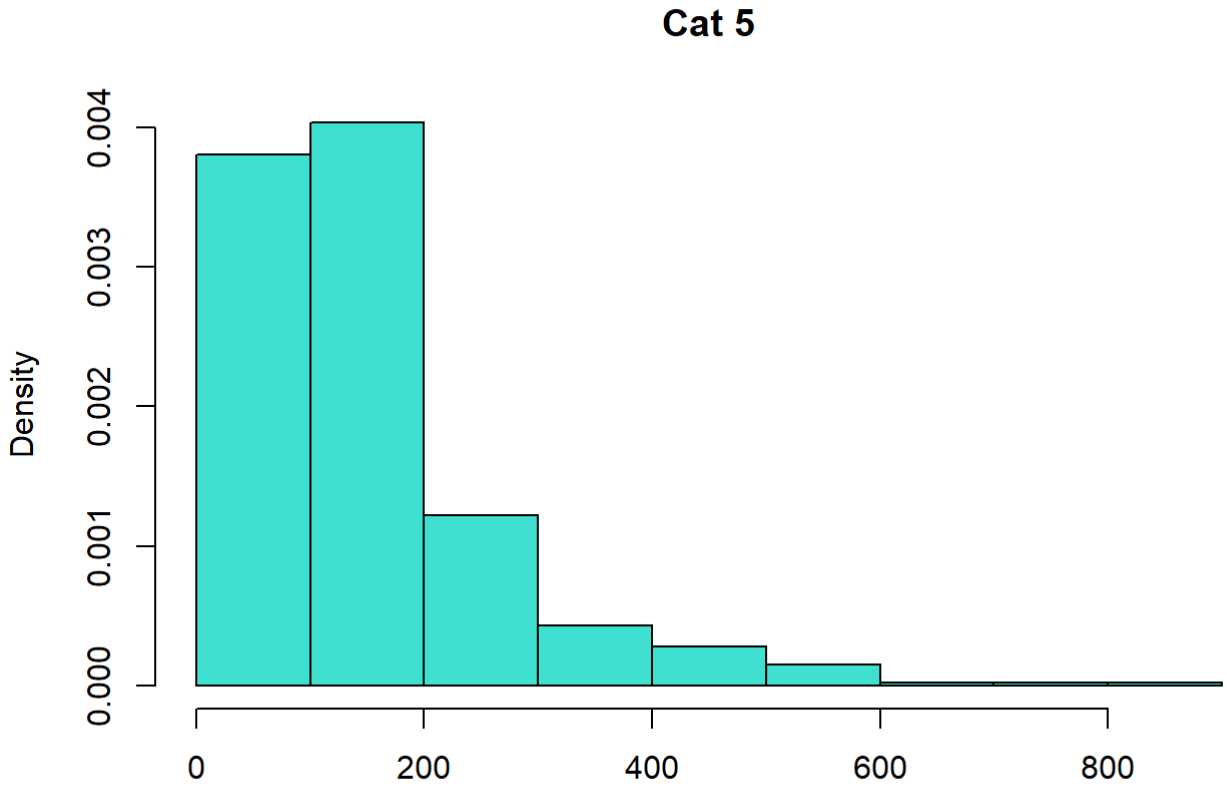

Cat 1
Density
0.00
0.05
0.10
0.15
0.20
0.25
0
5
10
15

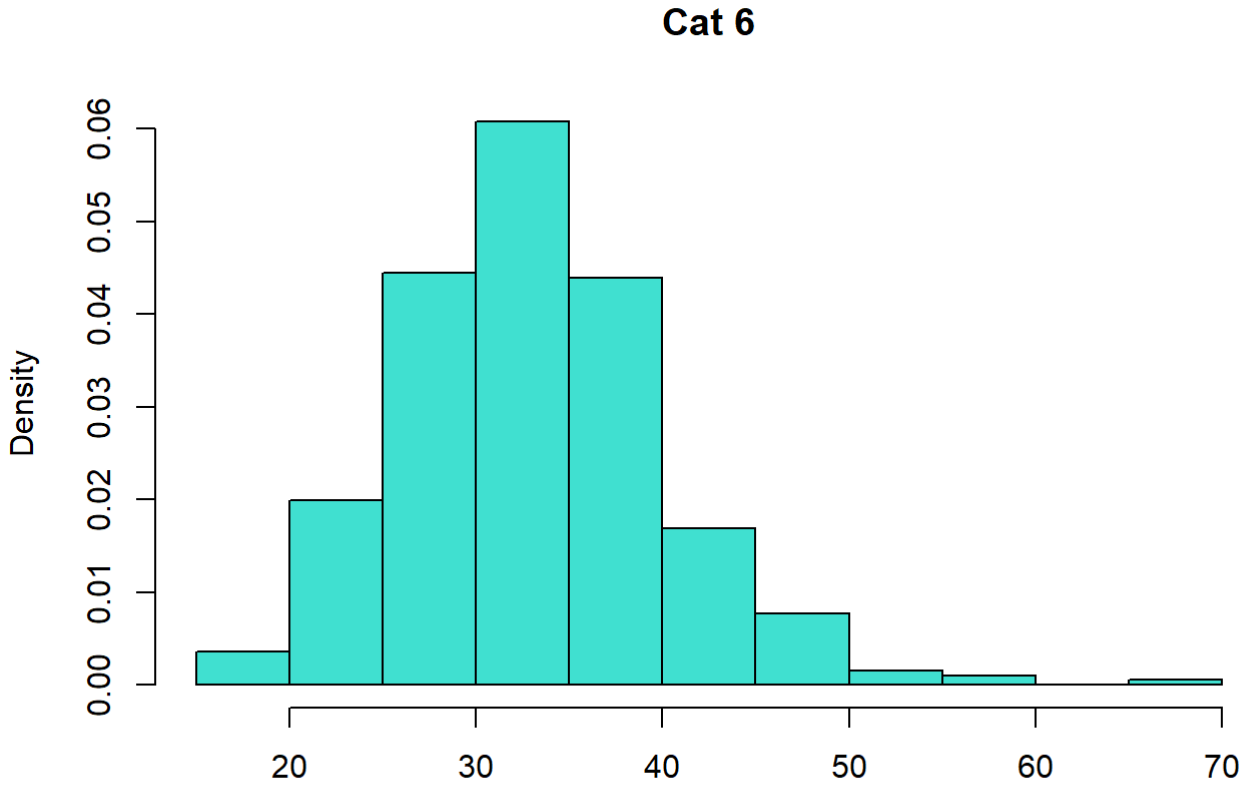

Cat 2
Density
0.000
0.004
0.008
0.012
50
100
150
200

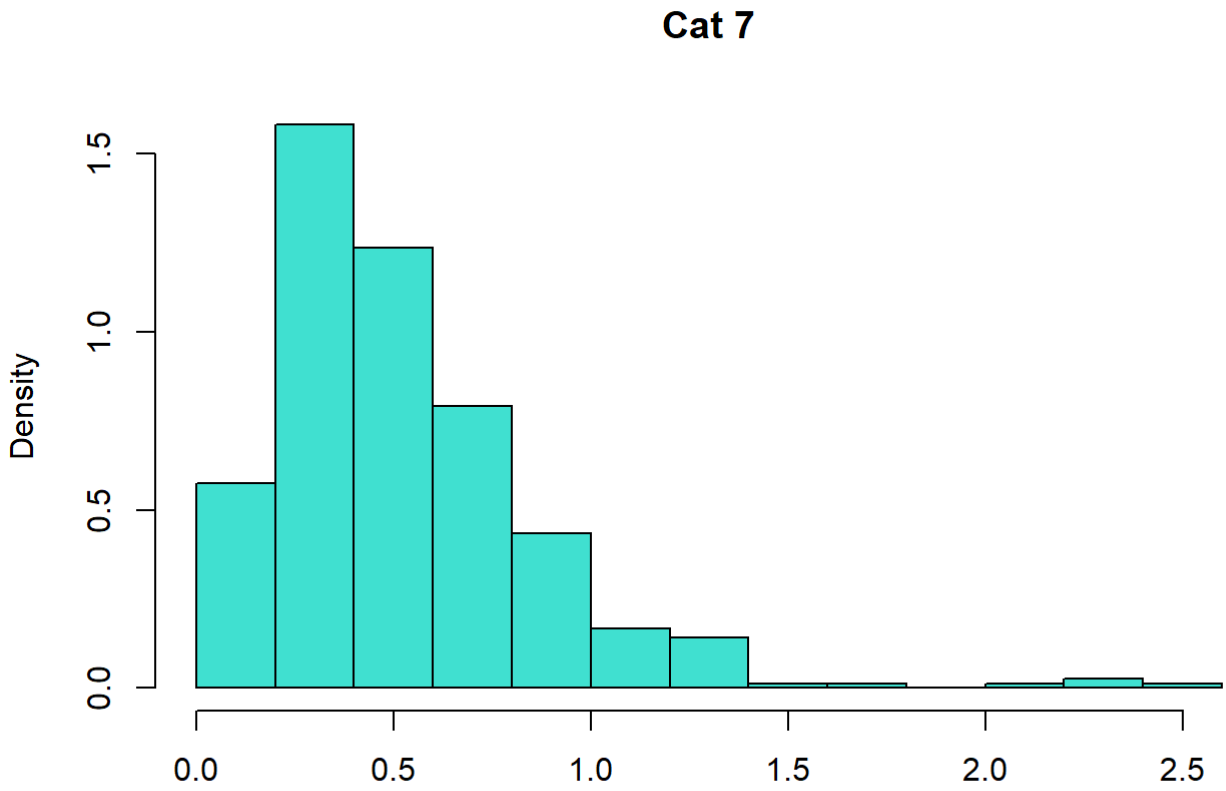

Cat 3
30

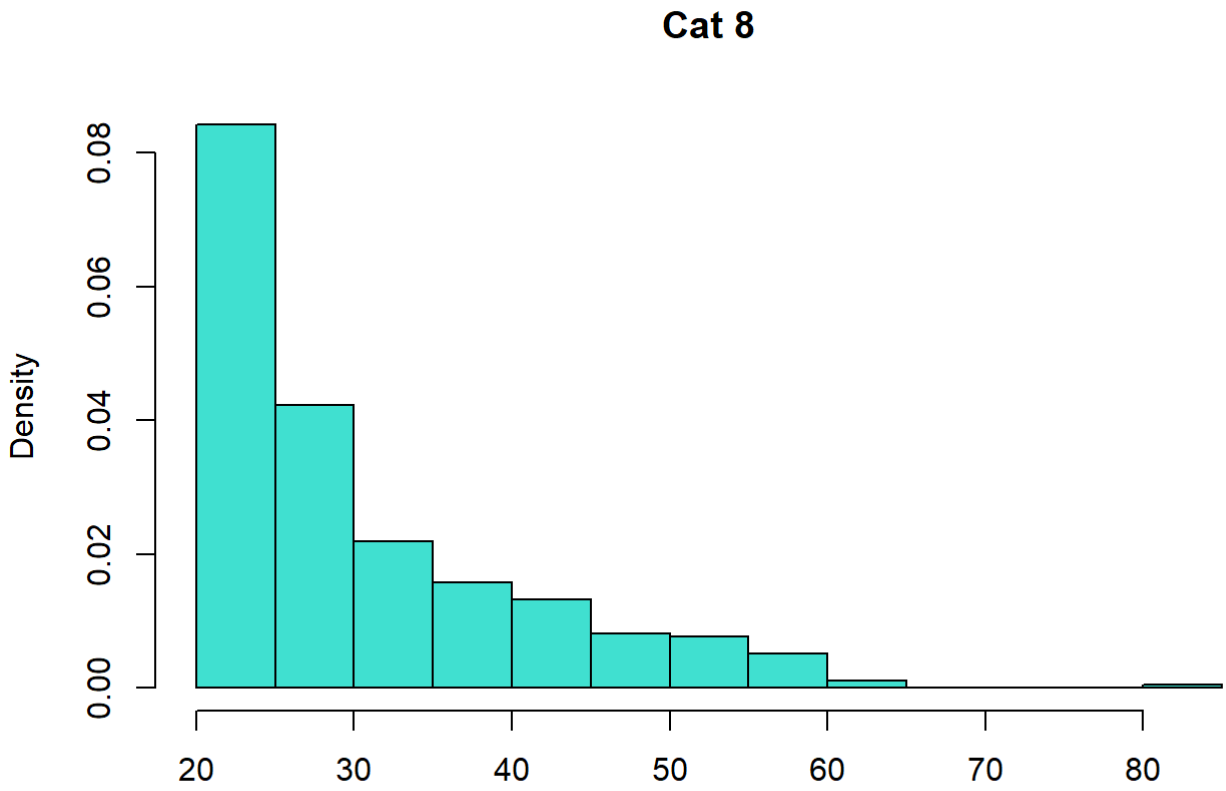

Density
0.000
0.010
0.020
20
40
60
80
100


Cat 4
Density
0.00
0.01
0.02
0.03
10
20
30
40
50
60


Cat 5
Density
0.000
0.001
0.002
0.003
0.004
0
200
400
600
800


Cat 6
Density
0.00
0.01
0.02
0.03
0.04
0.05
0.06
20
30
40
50
60
70





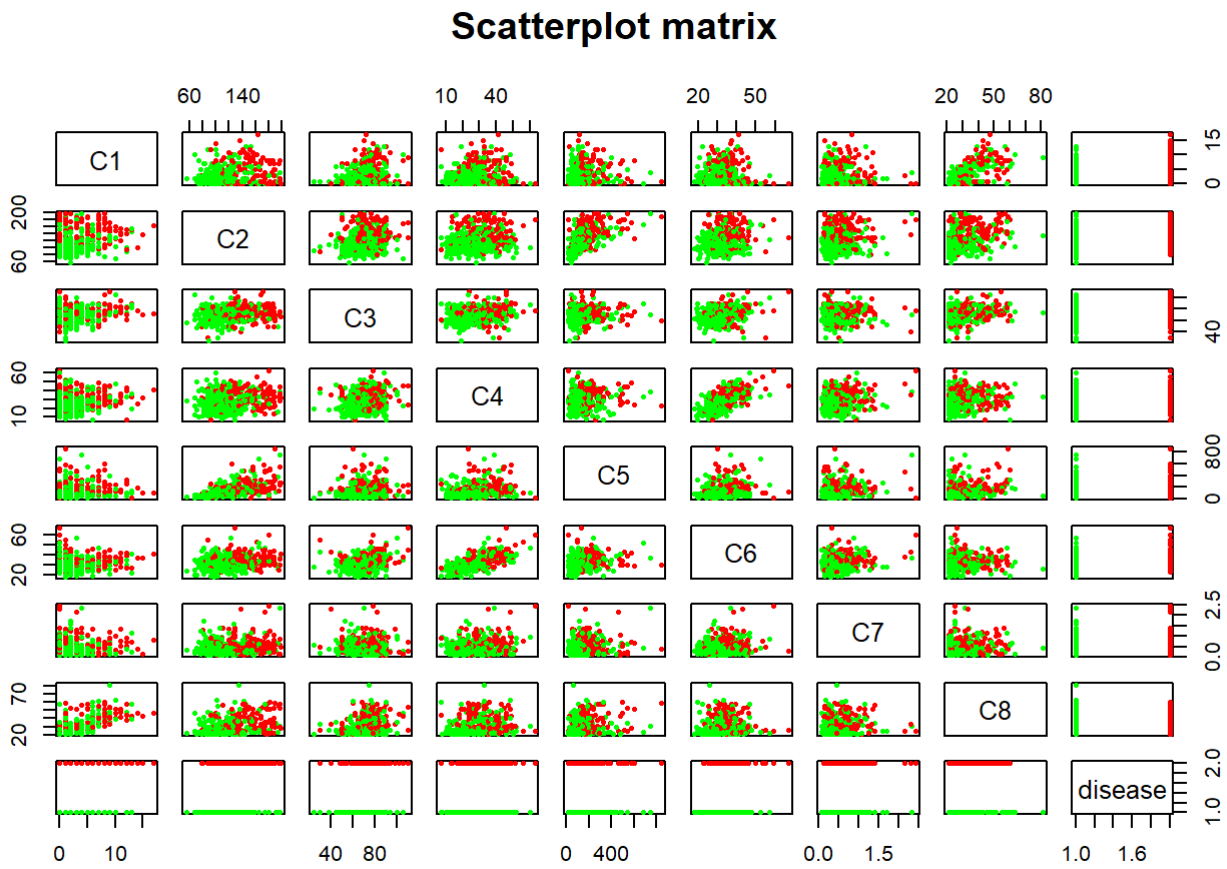
Looking at the different histograms and their shapes we can say that: the values of the first, the fifth and the eighth categories follow an exponential distribution; the fourth and the sixth follows a normal distribution; for all the others is more complicated to precisely state which type of distribution exactly fits. However, we could say

that they belong to the gamma/shifted-gamma family. Furthermore, since we have many observation we could also approximate the remaining distributions to a normal.

Plot a scatterplot matrix between all the independent variables, coloring the data by disease status

Now we want to create a scatter-plot matrix where the red observation refers to the individuals with the disease and the green observation refers to the people without the disease. With this scatter-plot matrix we want to investigate the interaction between the different variables among each other. We also call the function `cor()` which produces a matrix that contains all the pairwise correlations among the predictors in the data set. However, this function works only on numerical variables hence, we have to consider just the first eight columns of our data set.

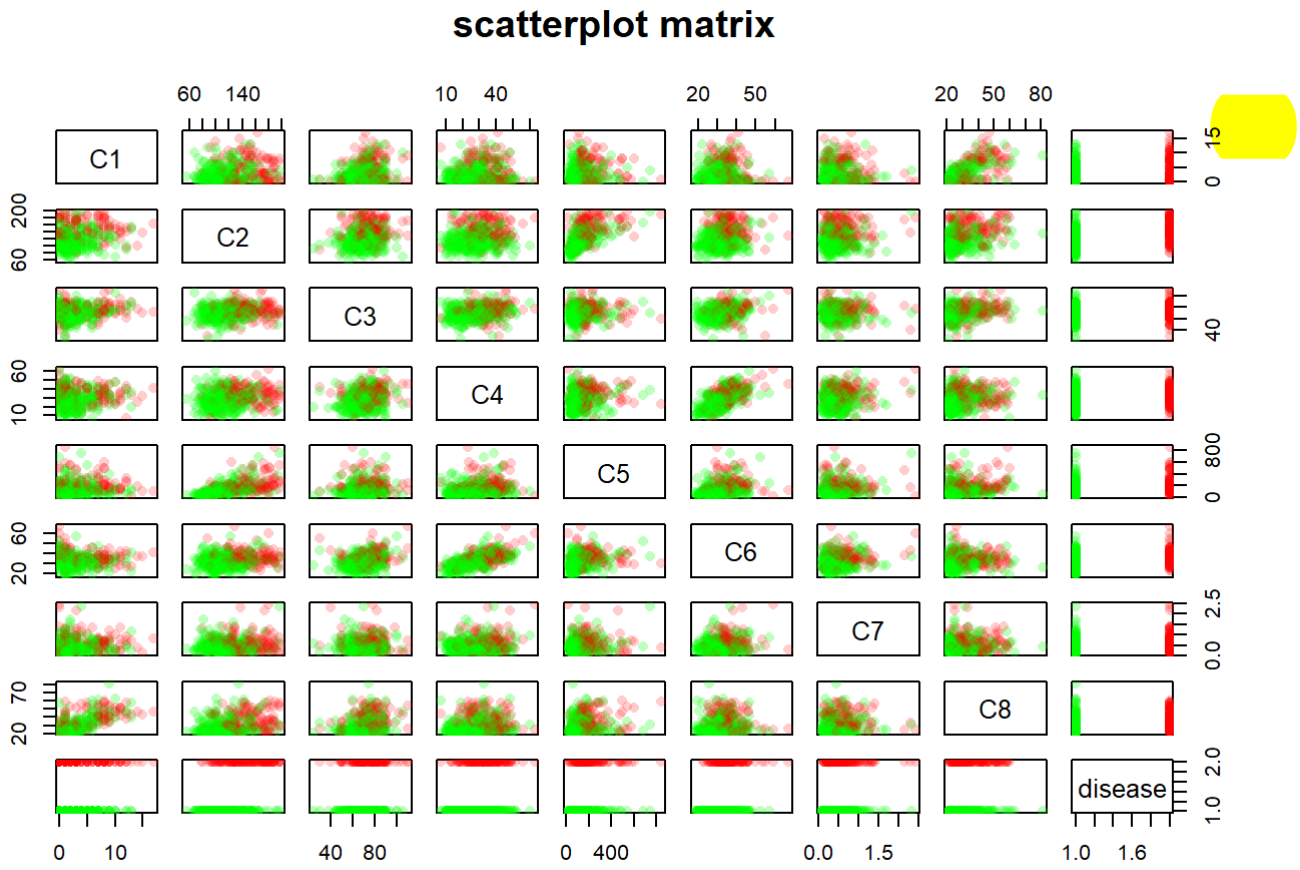
```
colorss <- c('green','red')
pairs(df, pch=16, cex = 0.5, col = ifelse(df$disease =='yes', colorss[2], colorss[1]), main =
'Scatterplot matrix' )
```




```
cor(df[,-9])
```

```
##              C1         C2         C3         C4          C5          C6
## C1  1.000000000 0.1982910  0.2133548 0.0932094 0.07898363 -0.02534728
## C2  0.198291043 1.0000000  0.2100266 0.1988558 0.58122301  0.20951592
## C3  0.213354775 0.2100266  1.0000000 0.2325712 0.09851150  0.30440337
## C4  0.093209397 0.1988558  0.2325712 1.0000000 0.18219906  0.66435487
## C5  0.078983625 0.5812230  0.0985115 0.1821991 1.00000000  0.22639652
## C6 -0.025347276 0.2095159  0.3044034 0.6643549 0.22639652  1.00000000
## C7  0.007562116 0.1401802 -0.0159711 0.1604985 0.13590578  0.15877104
## C8  0.679608470 0.3436415  0.3000389 0.1677611 0.21708199  0.06981380
##              C7         C8
## C1  0.007562116 0.67960847
## C2  0.140180180 0.34364150
## C3 -0.015971104 0.30003895
## C4  0.160498526 0.16776114
## C5  0.135905781 0.21708199
## C6  0.158771043 0.06981380
## C7  1.000000000 0.08502911
## C8  0.085029106 1.00000000
```

```
column_1 <- c(rgb(green=255, blue = 0, red = 0,   alpha=65, maxColorValue=255),
rgb(red=255,   green=0, blue = 0, alpha=50, maxColorValue=255) )
column_1 <- ifelse(df$disease=='no', column_1[1], column_1[2])
set.seed(99)
jittdf <-  apply(df[,1:8], 2, FUN=function(x){ jitter(x, amount=.56) })
jittdf <-  cbind(jittdf, class=df[,1])
pairs(df, col=column_1, pch=16, main = 'scatterplot matrix')
```



Through plotting, try to understand which attributes are most related to the outcome

In the previous point we create a scatter-plot matrix however, to avoid loosing information I have also create a new scatter-plot matrix were it is possible to see the overlapping points. Said so, we can see that for *C1*, *C3*, *C4*, it is difficult to see any shape or something that can leads us to suggest that there is a relation to the disease status. On the other side, we see that *C2* could be instead a relevant factor. Indeed, we see that the values are split in two main areas, on the left there are the values of not infected individuals while on the right there are the values of the infected ones. *C6* and *C7* do not show any particular trend so we can say that these two variables are not that related to the outcome. On the other hand, *C8* looks to be more related to the outcome because we can see the the data regarding the diseased individuals tend to be up and on the right compared to the one of not infected individuals. More difficult is to establish if *C5* is significantly related to the outcome or not.

For all the subsequent exercises:

Split the data randomly into reasonably sized train and test sets.

In order to make our analysis we need to split the data into a training set and a test set. Before we begin, we use the `set.seed()` function in order to set a seed for R's random number generator, so that we will obtain precisely the same results overtime. To build our train set and test set we decide that the 60% of the observation will be used for the training set while the residual 40% will constitute our test set. Although this is not a very common choice, we did so in order to have a bigger training set but also a consistent test set where we will test the accuracy of our models. To build the two sets we use the `sample()` function which selects randomly two third of the rows. These will constitute the training set and all the other rows the test set. We could not have divided the data set considering just the first two third of the rows and the last third because that could have led to a loss of some information due to the method through which the data has been collected.

Evaluate whether the re-scaling of predictors has an effect on your analyses/conclusions

```
set.seed(99)
size_tt <- floor(0.6 * nrow(df))
train_ind <- sample(seq_len(nrow(df)), size = size_tt)
train <-  df[train_ind,]
test <-  df[-train_ind,]
```

Exercise 2

Perform a classification of the data into the two classes using a logistic regression model on all the predictors.

Next, we will fit a logistic regression model in order to predict the presence of the *disease* using *C1* through *C8*. The `glm()` function fits *generalized linear models*, a class of models that includes logistic regression. The syntax used is the same that we use to that of `lm()` but we have to pass the argument `family = binomial` in order to tell R to run a logistic regression rather than other type of generalized linear model.

```
glm.fits <-  glm(disease ~ ., data = train, family = 'binomial')
summary(glm.fits)
```

```
##
## Call:
## glm(formula = disease ~ ., family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0386  -0.6252  -0.3187   0.5662   2.1763
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.037e+01  1.642e+00  -6.315 2.70e-10 ***
## C1           2.044e-01  7.935e-02   2.575 0.010015 *
## C2           4.552e-02  8.423e-03   5.404 6.52e-08 ***
## C3           4.846e-03  1.729e-02   0.280 0.779262
## C4           3.886e-02  2.221e-02   1.749 0.080216 .
## C5           6.332e-04  1.817e-03   0.348 0.727501
## C6           3.007e-02  3.510e-02   0.857 0.391592
## C7           1.931e+00  5.726e-01   3.371 0.000748 ***
## C8          -1.740e-02  2.475e-02  -0.703 0.482034
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 198.64  on 226  degrees of freedom
## AIC: 216.64
##
## Number of Fisher Scoring iterations: 5
```

Evaluate the output to identify the most significant predictors.

Let's examine the summary of the regression model: The smallest p-values here is associated with *C2* and it is very low this implies that there is a clear evidence of a real association between *C2* and the disease. The other statistically relevant predictors are *C1* and *C7*; their p-values is below the ordinary threshold 0,05 but, we notice also that compared to *C2* these other two predictors have a lower weight. (In the first attempt of doing the homework I set another seed and the only statistically relevant predictor was *C2*). Looking at the residual deviance and the null deviance we can say that there is no over-dispersion and we can reject the null hypothesis.
Evaluate the model performance in terms of confusion matrix and train/test accuracy The `predict()` function can be used to predict the probability that an individual has the disease, given values of the predictors. The *type* = *response* option tells R to output probabilities of the form $P(Y = 1|X)$, as opposed to other information such as the *logit*. If no data set is supplied to the *predict()* function, then the probabilities are

computed for the training data that was used to fit the logistic regression model. Here we have printed the only the first ten probabilities. We know that these values correspond to the probability of having the disease, because the `contrast()` function indicates that R has created a dummy variable with a 1 for disease yes.

```
contrasts(df$disease)
```

```
##     yes
## no    0
## yes   1
```

```
glm.probs <-predict(glm.fits, test,  type = 'response')
glm.probs[1:10]
```

```
##          2          4          5          6         10         12         16
## 0.04663520 0.78277224 0.07115560 0.18918465 0.89502338 0.01245535 0.01973135
##         17         18         19
## 0.93629443 0.16689586 0.12783702
```

Now, in order to make prediction as to whether an individual has the disease or not, we must covert these predicted probabilities into class labels *yes* and *no*. To do so, we firstly create a vector of class prediction based on whether the predicted probability of having the disease is greater than or less than a fixed threshold that we decided to be 50%.

```
glm.pred <-  rep('no', nrow(test))

glm.pred[glm.probs > 0.5] <- 'yes'
```

The first command creates a vector of 157 *no*. With the second command we transform all of the elements for which the predicted probability of being infected exceeds the 50%.
Now, that we have *glm.pred* we can build the confusion matrix with the function `table()` in order to determine how many observation were correctly or incorrectly classified.

```
table(glm.pred, test$disease)
```

```
##
## glm.pred no yes
##      no  94  22
##      yes 12  29
```

```
mean(glm.pred == test$disease)
```

```
## [1] 0.7834395
```

```
mean(glm.pred != test$disease)
```

```
## [1] 0.2165605
```

```
glm.acc <-  mean(glm.pred == test$disease)
```

The diagonal elements of the confusion matrix indicate correct prediction, while the off diagonals represent incorrect prediction. Hence, our model correctly predicted that 94 individuals do not have the disease while 29 do have it. The `mean()` function can be used to compute the number of individuals for which the prediction was correct. In this case, logistic regression correctly predicted the presence of the disease for 78% of the individuals. Our predictions looks quite satisfying since the model performs much better than random guessing. The accuracy of the model is high and consequently the error is low. However, since we are dealing with a disease, this result should be confirmed with further analysis in order to make a more accurate prediction that is not influenced by random noise or from other source of uncertainty.

Using the fitted model, predict the probability of having the disease for someone who has C2=31 and all other predictors set to their average value.

Now we want to use the previous fitted model to predict the probability of having the disease given some specific attributes. To do so, we create a data set formed by just one observation where the all the variables have mean values except *C2* which as a fixed values equals to 31.

```
new_df <-  data.frame(C1 = mean(df$C1), C2 = 31, C3 = mean(df$C3), C4 = mean(df$C4),C5 = mean
(df$C5), C6 = mean(df$C6), C7 = mean(df$C7), C8 = mean(df$C8))
```

Here we have this new data set that satisfies the request of the exercise and we can make prediction on this data set and compute the probability of having the disease.

```
prob.individuals <-  predict(glm.fits, new_df, type = 'response')
prob.individuals
```

```
##         1
## 0.0052569
```

It seems very unlikely for un individual to have the disease if his values for C2 are equal to 31.

Let’s examine if rescaling the predictors has effect on our analysis/conclusion.

Now, we want to rescale the predictors and verify if this has some effect on our analysis. To rescale the our dataframe we simply have to call the function `scale()` . However, we need to remember that in order to scale the dataframe we have to exclude the categorical variable *disease*.

```
standardized.X <-  scale(df[,-9])
var(standardized.X[,1])
```

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

```
var(standardized.X[,2])
```

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

```
mean(standardized.X[,2])
```

```
## [1] 1.586531e-17
```

```
mean(standardized.X[,1])
```

```
## [1] 3.242055e-17
```

In this way we have created a standardized dataframe which has variance 1 and mean (almost) 0. Now that we have created this new dataframe we add the categorical variable *disease*.

```
standardized.X <-  as.data.frame(standardized.X)
standardized.X$disease <-  df$disease
train.x <-  standardized.X[train_ind,]
test.x <-  standardized.X[-train_ind,]
```

Now we have created a new standardized training set and a new standardized test set both created with the same methods as before. Now we will repeat the same computation to find the value of the accuracy of the test with the standardized sets.

```
glm.fitx <-  glm(disease~., data = train.x, family = 'binomial')
summary(glm.fitx)
```

```
##
## Call:
## glm(formula = disease ~ ., family = "binomial", data = train.x)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0386  -0.6252  -0.3187   0.5662   2.1763
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.07238    0.19490  -5.502 3.75e-08 ***
## C1           0.65626    0.25483   2.575 0.010015 *
## C2           1.40467    0.25993   5.404 6.52e-08 ***
## C3           0.06056    0.21607   0.280 0.779262
## C4           0.40865    0.23359   1.749 0.080216 .
## C5           0.07525    0.21596   0.348 0.727501
## C6           0.21131    0.24665   0.857 0.391592
## C7           0.66699    0.19784   3.371 0.000748 ***
## C8          -0.17750    0.25248  -0.703 0.482034
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 198.64  on 226  degrees of freedom
## AIC: 216.64
##
## Number of Fisher Scoring iterations: 5
```

The result of this model are identical to the one computed above: *C2* is the most statistically significant predictor detecting the presence of the disease then follow *C7* and *C1*. Once again we use the predict function to make our predictions and computing the probability of having the disease according to our model.

```
glm.probx <-predict(glm.fitx, test.x, type = 'response')
glm.probx[1:10]
```

```
##          2          4          5          6         10         12         16
## 0.04663520 0.78277224 0.07115560 0.18918465 0.89502338 0.01245535 0.01973135
##         17         18         19
## 0.93629443 0.16689586 0.12783702
```

Now we use the probabilities just computed to create a vector of class predictions as before.

```
glm.predx <- rep('no', nrow(test.x))
glm.predx[glm.probx>0.5] <-  'yes'
```

Lastly, we create a confusion matrix to investigate the accuracy of our new predictions.

```
table(glm.predx, test.x$disease)
```

```
##
## glm.predx no yes
##       no  94  22
##       yes 12  29
```

```
mean(glm.predx == test.x$disease)
```

```
## [1] 0.7834395
```

```
mean(glm.predx != test.x$disease)
```

```
## [1] 0.2165605
```

As we can see from the results scaling the results does not have any effect on the analysis neither on the conclusion.

Exercise 3

Perform a classification via a k-nn model using all of the available variables and exploring different values of k;

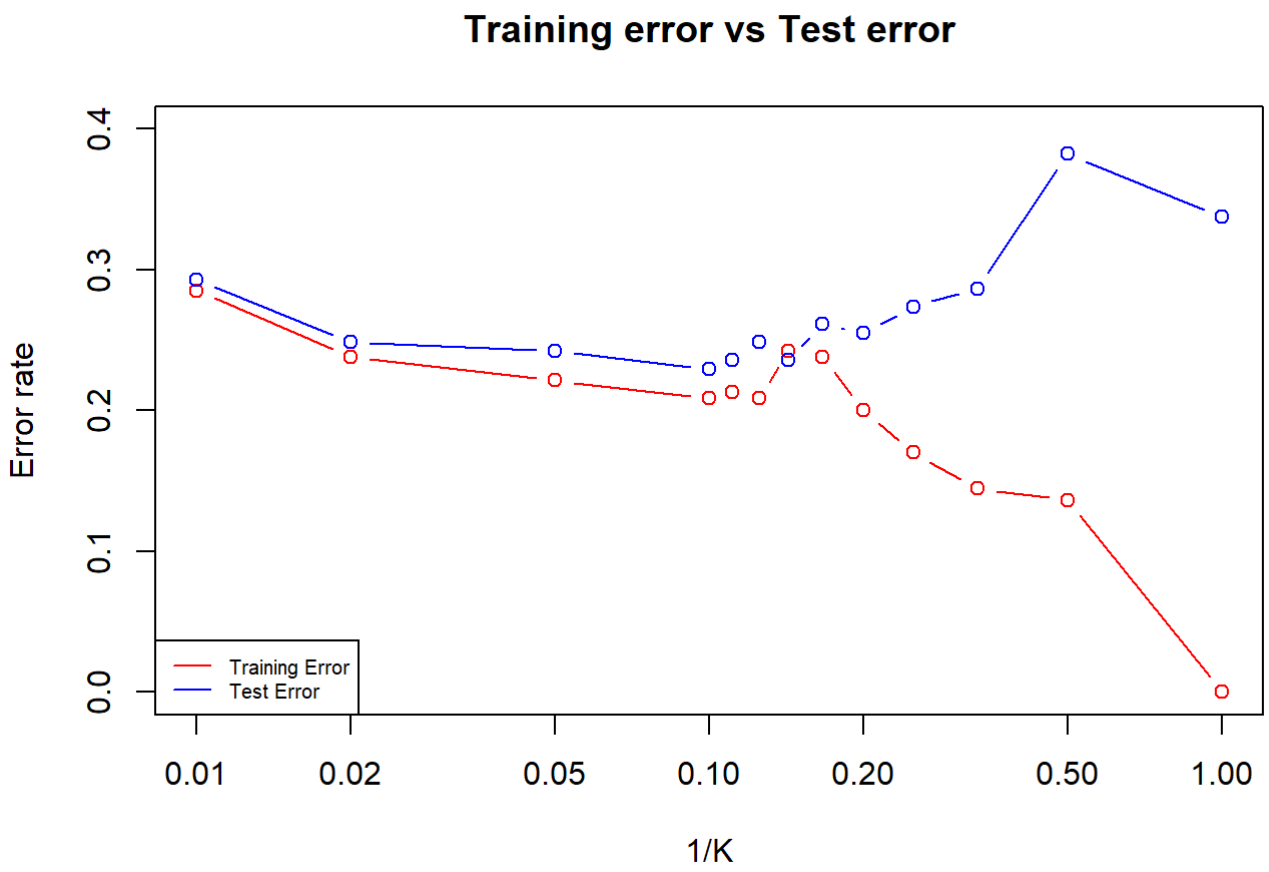
Now, we will use a different approach the **the K-Nearest Neighbor (KNN) model**. To perform the K-Nearest Neighbors we use the function `knn()` , which is part of the *class* library. This function works rather differently from the other model-fitting functions. Rather than a two-step approach in which we first fit the model and then we use the model to make prediction, the `knn()` forms predictions using a single command. For the KNN we need at least 4 ingredients: * a matrix with predictors for training (`train`); * a matrix with predictors for testing

(`test`); * a vector with training labels (`cl`) * the number of neighbor (`k`) To perform a classification via a k-nn model we create 4 data set in order to apply the function `knn()` we also create a function to easily compute the error rate and we also use a for loop to easily compute and compare the accuracy of the knn models at different k levels.

```
library(class)
x_tr <-  train[,1:8]
x_ts <-  test[, 1:8]

y_tr <-  train[,9]
y_ts <-  test[,9]


calc_error_rate <-  function(predicted.value, true.value){
  mean(true.value != predicted.value)
}

set.seed(99)
errors_tr <-  errors_ts <- c()
kvec <-  c(seq(1:10), 20, 50, 100)
for (k in kvec) {
  pred_tr <- knn(x_tr, x_tr, y_tr, k = k, prob = TRUE)
  pred_ts <- knn(x_tr, x_ts, y_tr, k = k, prob = TRUE)
  err_tr <-  calc_error_rate(pred_tr, y_tr)
  err_ts <- calc_error_rate(pred_ts, y_ts)
  errors_tr <- append(errors_tr, err_tr)
  errors_ts <-  append(errors_ts, err_ts)
  print(table(pred_ts, test$disease))
  errors_ts
}
```

```
##
## pred_ts no yes
##     no  77  24
##     yes 29  27
##
## pred_ts no yes
##     no  75  29
##     yes 31  22
##
## pred_ts no yes
##     no  82  21
##     yes 24  30
##
## pred_ts no yes
##     no  84  21
##     yes 22  30
##
## pred_ts no yes
##     no  88  22
##     yes 18  29
##
## pred_ts no yes
##     no  87  22
##     yes 19  29
##
## pred_ts no yes
##     no  91  22
##     yes 15  29
##
## pred_ts no yes
##     no  88  21
##     yes 18  30
##
## pred_ts no yes
##     no  89  20
##     yes 17  31
##
## pred_ts no yes
##     no  90  20
##     yes 16  31
##
## pred_ts no yes
##     no  91  23
##     yes 15  28
##
## pred_ts no yes
##     no  97  30
##     yes  9  21
##
## pred_ts no yes
##     no  98  38
##     yes  8  13
```

```
plot(1,type = 'n', xlim = c(0.01,1), ylim = c(0, 0.4), log = 'x', xlab = '1/K', ylab = 'Error
rate', main = 'Training error vs Test error')
 lines(1/kvec, errors_tr, type = 'b', col = 'red')
lines(1/kvec, errors_ts, type ='b', col ='blue')
legend('bottomleft', legend = c('Training Error', 'Test Error'), col=c('red','blue'), lty = 1
, cex = .64)
```



Discuss the results and reach a conclusion on the optimal value for k.

A crucial step when we apply the K-NN is to chose the right value for k. This is not an easy task since we have to face a trade-off: a bias-variance trade-off. Indeed, with a big value of K we have a simpler model, high bias and low variance; with a small value of k we have a more complex model, low bias and high variance. In other words, with a small value of k we tend to have an accurate model on average but we will obtain different results for different data set. On the other hand, with a big value of k the answer will be similar for different data set but the answer will be also less accurate. In the plot shown above we see how the training and the test error change according to different values of k. Obviously, the training error is always lower than the test error since in the first case we train and test the data on the same data set. However, looking at the data and at the graph we can say that a reasonably good data for k is 10. Indeed, with this value of k we have a model accuracy around the 77% moreover, we have also a low value for the false negative which is also a key factor since we are dealing with a disease.

Now we repeat the same operations after having rescaled the predictors.

Because the KNN classifier predict the class of a given set of observation by identifying the observations that are nearest to it, the scale of the variables matters. Any variables that are on a large scale will have much larger effect on the distance between the observations, and hence on the KNN classifier, than variables that are on a small scale. (Just to make it clearer let's imagine a data set where we have two predictors *salary* and *age*. As far as KNN is concerned, a difference of 1000$ in *salary* is enormous compared to a variation of 50 years in *age*. Consequently, *salary* will drive the KNN classification results, and *age* will have almost no effect. This is contrary to the intuition that a salary difference of 1000$ is quite small compared to an age difference of 50 years.) So, a good way to handle this problem is to standardize the data so that all variables are given a mean zero and standard deviation of one. Then all the variables will be on a comparable scale. To do so we use the already known function `scale()`. This being said, we do not have much information on our data set, we do not know what the predictors represent so we do not know if they use different scales. At a first look we could say yes because some variables have quite high values for example *C2* while other have quite small numbers e.g. *C7*. Hence, we expect that there will be some differences between the un-scaled result and the scaled one.

```
x_trx <- train.x[1:8]
x_tsx <- test.x[1:8]

y_trx <-  train.x[9]$disease
y_tsx <-  train.x[9]$disease
knn_acc <-  c()
errors_trx <-  errors_tsx <-  c()
for(k in kvec) {
  pred_trx <-  knn(x_trx, x_trx, y_trx , k = k, prob = T)
  pred_tsx <-  knn(x_trx, x_tsx, y_trx, k = k, prob = T)
  err_trx <-  calc_error_rate(pred_trx, y_trx)
  err_tsx <-  calc_error_rate(pred_tsx, y_tsx)
  errors_trx <-  append(errors_trx, err_trx)
  errors_tsx <-  append(errors_tsx, err_tsx)
  print(table(pred_tsx, test$disease))
  knn_acc <-  append(knn_acc, mean(pred_tsx == y_tsx))
}
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  87  20
##      yes 19  31
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  87  23
##      yes 19  28
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  88  23
##      yes 18  28
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  89  20
##      yes 17  31
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  92  21
##      yes 14  30
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  94  22
##      yes 12  29
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  95  26
##      yes 11  25
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  92  24
##      yes 14  27
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  96  25
##      yes 10  26
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  96  25
##      yes 10  26
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx no yes
##      no  97  25
##      yes  9  26
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx  no yes
##      no  100  24
##      yes   6  27
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

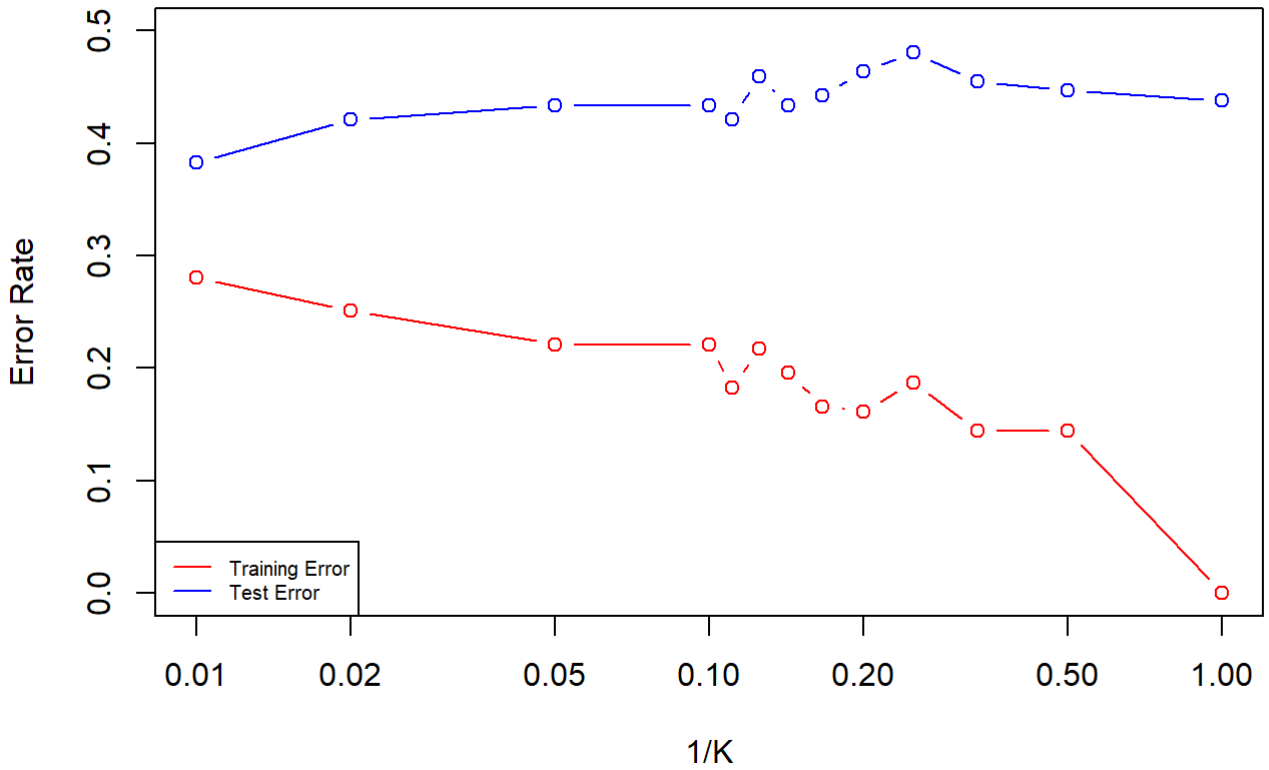
```
## Warning in `!=.default`(true.value, predicted.value): longer object length is
## not a multiple of shorter object length
```

```
## Warning in is.na(e1) | is.na(e2): longer object length is not a multiple of
## shorter object length
```

```
##
## pred_tsx  no yes
##      no  102  36
##      yes   4  15
```

```
## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length

## Warning in `==.default`(pred_tsx, y_tsx): longer object length is not a multiple
## of shorter object length
```

```
plot(1, type = 'n', xlim = c(0.01, 1), ylim = c(0,0.5), log = 'x', xlab= '1/K', ylab = 'Error
Rate')
lines(1/kvec, errors_trx, type = 'b', col = 'red')
lines(1/kvec, errors_tsx, type = 'b', col='blue')
legend('bottomleft', legend = c('Training Error', 'Test Error'), col=c('red','blue'), lty = 1
, cex = .64)
```




```
mean(knn_acc)
```

```
## [1] 0.5603928
```

As we expected, after having standardized the training set and the test set we have different results. As before, we have to choose the right value for k, to do so we look again to the accuracy of the model. Looking at the data we notice that the accuracy of the model reaches its maximum with two different values of k: k = 8 and k = 50. However, looking at the confusion matrix we could prefer the option with k = 50 because in this case the number of false negative is lower than in the case of k = 8 and as mentioned before since we are dealing with a disease we think that it is preferable having a test which is more able to detect the presence of the disease instead of its absence.

Exercise 4

Explore the use of LDA, QDA, and Naive Bayes to predict disease onset using all the predictors. For each method:

* Train the model on the training data

* Apply the fitted model to the test set; compute the confusion matrix and prediction accuracy

Now we will perform **LDA** on the training set using the `lda()` function, which is part of the MASS library. The syntax for the `lda()` function is identical to the one of the `lm()`.

LDA

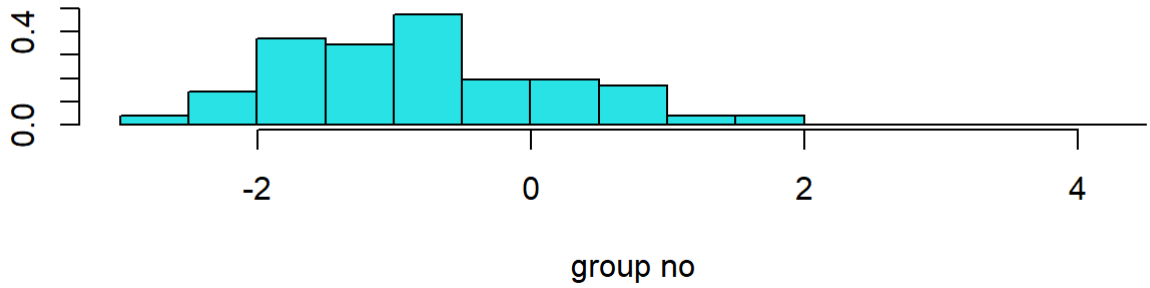
```
if (!require(MASS)) {
    install.packages(MASS)
    library(MASS)
}
```

```
## Loading required package: MASS
```

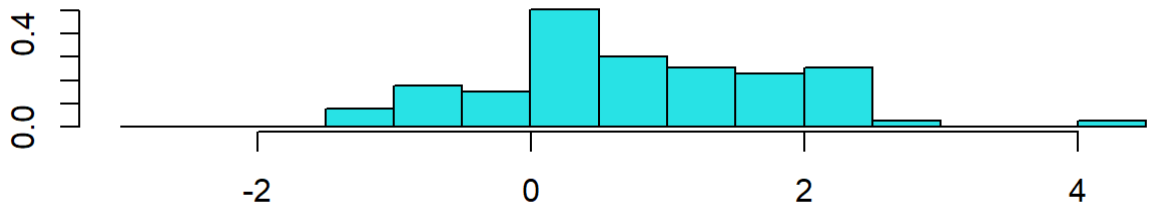
```
lda.fit <-  lda(disease~., data = train)
lda.fit
```

```
## Call:
## lda(disease ~ ., data = train)
##
## Prior probabilities of groups:
##        no       yes
## 0.6638298 0.3361702
##
## Group means:
##           C1       C2       C3       C4       C5       C6        C7       C8
## no  2.782051 112.1538 69.35256 27.86538 127.7372 32.22692 0.4601603 28.49359
## yes 4.341772 144.7342 73.45570 33.63291 207.0380 35.64810 0.6398987 34.88608
##
## Coefficients of linear discriminants:
##              LD1
## C1  0.1325431364
## C2  0.0301822740
## C3  0.0062865838
## C4  0.0233367618
## C5  0.0009209734
## C6  0.0105412914
## C7  1.1580491590
## C8 -0.0088817132
```

```
plot(lda.fit)
```



group no



group yes

The *LDA* provides the group means; these are the average of each predictor within each class, and are used by *LDA* as estimates of μk. The coefficients of linear discriminant output provides the linear combination of the predictors that are used to form the LDA decision rule.

The `predict()` function applied to the model and the test set returns a list with three elements. The first

element *class*, contains LDA's prediction about the presence of the disease. The second element *posterior*, is a matrix whose kth column contains the posterior probability that the corresponding observation belongs to the kth class, computed from $Pr(Y = k|X = x) = \frac{\pi f_k(x)}{\sum_{i=1}^{10} \pi_l f_l(x)}$. Finally, *x* contains the linear discriminant described earlier.

```
lda.pred <- predict(lda.fit, test)
names(lda.pred)
```

```
## [1] "class"     "posterior" "x"
```

```
lda.class <-  lda.pred$class
table(lda.class,test$disease)
```

```
##
## lda.class no yes
##       no  92  21
##       yes 14  30
```

```
mean(lda.class == test$disease)
```

```
## [1] 0.7770701
```

```
lda.acc <- mean(lda.class == test$disease)
sum(lda.pred$posterior[,2] >= .5)
```

```
## [1] 44
```

```
sum(lda.pred$posterior[,2] < .5)
```

```
## [1] 113
```

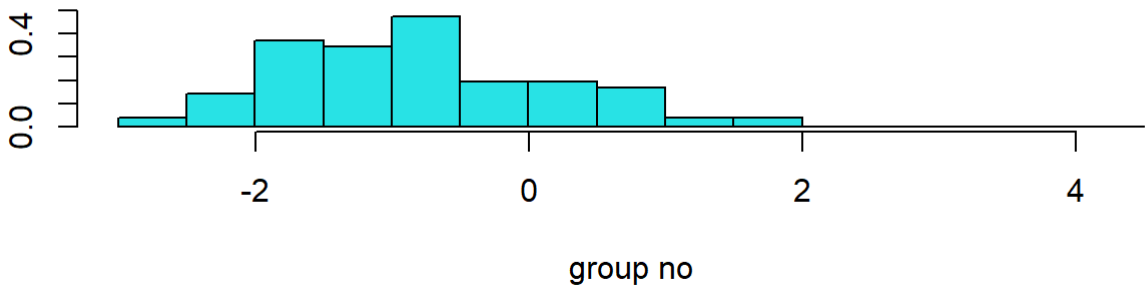
Applying a 50% threshold to the posterior probabilities allows us to recreate the predictions contained in *lda.class*.

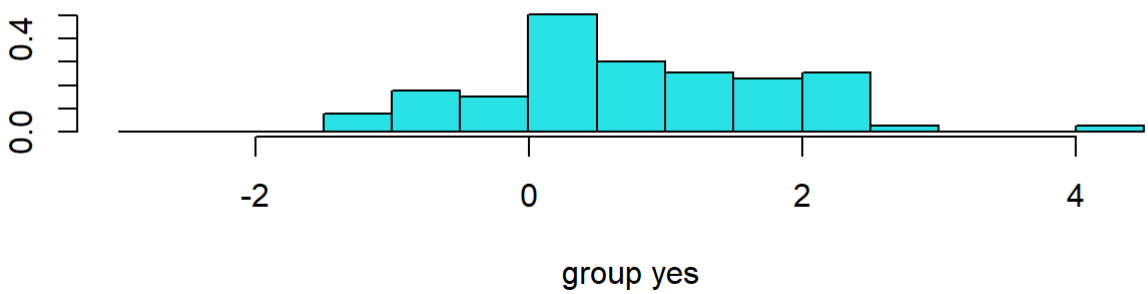
Looking at the results we notice that LDA and logistic regression predictions are almost identical. But at a first glance, we might prefer the logistic regression because the accuracy is slightly higher and the number of false negative is lower in the logistic regression and this could be useful since we are dealing with a disease and it is reasonable to believe that false negative individuals could be a danger. ## Now we will re compute the same operation with the standardized training set and test set.

```
lda.fitx <-  lda(disease~., data = train.x)
lda.fitx
```

```
## Call:
## lda(disease ~ ., data = train.x)
##
## Prior probabilities of groups:
##        no       yes
## 0.6638298 0.3361702
##
## Group means:
##             C1         C2         C3         C4         C5         C6
## no  -0.1616009 -0.3393856 -0.1048889 -0.1217166 -0.2382913 -0.1222742
## yes  0.3240779  0.7163340  0.2234643  0.4267138  0.4289896  0.3645420
##             C7         C8
## no  -0.1820198 -0.2324535
## yes  0.3382254  0.3942131
##
## Coefficients of linear discriminants:
##            LD1
## C1  0.42565227
## C2  0.93144854
## C3  0.07855773
## C4  0.24541928
## C5  0.10945003
## C6  0.07408060
## C7  0.40009214
## C8 -0.09060037
```

```
plot(lda.fitx)
```





```
lda.predx <- predict(lda.fitx, test.x)
names(lda.predx)
```

```
## [1] "class"     "posterior" "x"
```

```
lda.classx <-  lda.predx$class
table(lda.classx,test.x$disease)
```

```
##
## lda.classx no yes
##        no  92  21
##        yes 14  30
```

```
mean(lda.classx == test.x$disease)
```

```
## [1] 0.7770701
```

```
sum(lda.predx$posterior[,2] >= .5)
```

```
## [1] 44
```

```
sum(lda.predx$posterior[,2] < .5)
```

```
## [1] 113
```

As we can see from the results there is no change when we standardize the training and the test set with the LDA.

QDA

We will now fit a *QDA* model to our training set. For what concerns the syntax of this function we know that it is pretty identical to the one of the `lda()` function.

```
qda.fit <-  qda(disease~., data = train)
qda.fit
```

```
## Call:
## qda(disease ~ ., data = train)
##
## Prior probabilities of groups:
##        no       yes
## 0.6638298 0.3361702
##
## Group means:
##           C1       C2       C3       C4       C5       C6        C7       C8
## no  2.782051 112.1538 69.35256 27.86538 127.7372 32.22692 0.4601603 28.49359
## yes 4.341772 144.7342 73.45570 33.63291 207.0380 35.64810 0.6398987 34.88608
```

The output contains the group means. But it does not contain the coefficients of the linear discriminants, because the *QDA* classifier involves a quadratic rather than a linear function of predictors. The `predict()` function works in exactly the same fashion as for *LDA*.

```
qda.pred <-  predict(qda.fit, test)
qda.class <-  qda.pred$class
table(qda.class, test$disease)
```

```
##
## qda.class no yes
##       no  91  19
##       yes 15  32
```

```
mean(qda.class == test$disease)
```

```
## [1] 0.7834395
```

```
qda.acc <-  mean(qda.class == test$disease)
```

The *QDA* predictions are slightly more accurate than the ones of the *LDA*. The accuracy of the model is almost identical to the one of the logistic regression. However, according to the reasoning used for the selection of k in the KNN, the data suggest that we should prefer the QDA since the number of false negative is lower with the *QDA*.

Now we will re compute the same operation with the standardized training set and test set.

```
qda.fitx <-  qda(disease~., data = train.x)
qda.fitx
```

```
## Call:
## qda(disease ~ ., data = train.x)
##
## Prior probabilities of groups:
##        no       yes
## 0.6638298 0.3361702
##
## Group means:
##             C1         C2         C3         C4         C5         C6
## no  -0.1616009 -0.3393856 -0.1048889 -0.1217166 -0.2382913 -0.1222742
## yes  0.3240779  0.7163340  0.2234643  0.4267138  0.4289896  0.3645420
##             C7         C8
## no  -0.1820198 -0.2324535
## yes  0.3382254  0.3942131
```

```
qda.predx <- predict(qda.fitx, test.x)
names(qda.predx)
```

```
## [1] "class"     "posterior"
```

```
qda.classx <-  qda.predx$class
table(qda.classx,test.x$disease)
```

```
##
## qda.classx no yes
##        no  91  19
##        yes 15  32
```

```
mean(qda.classx == test.x$disease)
```

```
## [1] 0.7834395
```

```
sum(qda.predx$posterior[,2] >= .5)
```

```
## [1] 47
```

```
sum(qda.predx$posterior[,2] < .5)
```

```
## [1] 110
```

As for the LDA, we observe no change in the results when we apply the `qda()` function on a standardized data set.

Naive Bayes

Naive Bayes (NB) is highly recommended when we our input data is composed only by categorical variables. However, NB can handle continuous variables too at the price of assuming they are normally distributed ('Gaussian naive Bayes): the probabilities of likelihood are computed using the Gaussian probability density function. Another way to deal with continuous variables is to discretize them into discrete values beforehand (for example, re-coding the values into quartiles). In R, the Naive Bayes (NB) classifier is included in the package `e1071` : if it is not already there, we will install it. To fit a NB model and to compute the predictions the syntax is almost the same that we used for the other parametric models.

```
if(!require("e1071")) {
    install.packages("e1071")
    library(e1071)
}
```

```
## Loading required package: e1071
```

```
## Warning: package 'e1071' was built under R version 4.0.4
```

```
nb.fit <-  naiveBayes(disease ~. ,data = train)
nb.fit
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        no       yes
## 0.6638298 0.3361702
##
## Conditional probabilities:
##      C1
## Y         [,1]     [,2]
##   no  2.782051 2.588548
##   yes 4.341772 3.658100
##
##      C2
## Y         [,1]     [,2]
##   no  112.1538 23.56081
##   yes 144.7342 27.54144
##
##      C3
## Y         [,1]     [,2]
##   no  69.35256 10.95669
##   yes 73.45570 12.52566
##
##      C4
## Y         [,1]      [,2]
##   no  27.86538 10.424590
##   yes 33.63291  9.908786
##
##      C5
## Y         [,1]     [,2]
##   no  127.7372  93.5733
##   yes 207.0380 124.9361
##
##      C6
## Y         [,1]     [,2]
##   no  32.22692 6.957329
##   yes 35.64810 7.154770
##
##      C7
## Y          [,1]      [,2]
##   no  0.4601603 0.2740381
##   yes 0.6398987 0.4364440
##
##      C8
## Y         [,1]     [,2]
##   no  28.49359 9.589306
##   yes 34.88608 9.754667
```

```
nb.preds <- predict(nb.fit, test)
table(nb.preds, test$disease)
```

```
##
## nb.preds no yes
##      no  87  17
##      yes 19  34
```

```
sum(nb.preds==test$disease)/nrow(test)
```

```
## [1] 0.7707006
```

```
prop.table(table(nb.preds, test$disease)[1,])
```

```
##        no       yes
## 0.8365385 0.1634615
```

```
prop.table(table(nb.preds, test$disease)[2,])
```

```
##        no       yes
## 0.3584906 0.6415094
```

```
nb.probs <-  predict(nb.fit, test, type = 'raw')
head(nb.probs)
```

```
##               no         yes
## [1,] 0.985907220 0.014092780
## [2,] 0.005140088 0.994859912
## [3,] 0.961526058 0.038473942
## [4,] 0.893180132 0.106819868
## [5,] 0.005336028 0.994663972
## [6,] 0.997819786 0.002180214
```

```
nb_acc <-  c(sum(nb.preds == test$disease)/nrow(test))
print(nb_acc)
```

```
## [1] 0.7707006
```

The *Naive Bayes* fitted object contains the priori probabilities of being infected and the means and standard deviations of the predictors (If the predictors were categorical variables the nb would have shown the conditional probabilities.) Looking at the result of the prediction did on the basis of the nb model we can say that the result is quite satisfying, we have an accuracy of the 77% similar to the obtained with the lda's model. However, we believe that the nb is not the best option in this case due to its assumption of independence between the different predictors because it is reasonable to think that there is a certain level of correlation between medical predictors. ## Now we will re compute the same operation with the standardized training set and test set.

```
nb.fitx <-  naiveBayes(disease ~. ,data = train.x)
nb.fitx
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##        no       yes
## 0.6638298 0.3361702
##
## Conditional probabilities:
##      C1
## Y           [,1]      [,2]
##   no  -0.1616009 0.8060435
##   yes  0.3240779 1.1390895
##
##      C2
## Y           [,1]      [,2]
##   no  -0.3393856 0.7634547
##   yes  0.7163340 0.8924415
##
##      C3
## Y           [,1]      [,2]
##   no  -0.1048889 0.8768095
##   yes  0.2234643 1.0023665
##
##      C4
## Y           [,1]      [,2]
##   no  -0.1217166 0.9912676
##   yes  0.4267138 0.9422201
##
##      C5
## Y           [,1]      [,2]
##   no  -0.2382913 0.7873777
##   yes  0.4289896 1.0512820
##
##      C6
## Y           [,1]      [,2]
##   no  -0.1222742 0.9899924
##   yes  0.3645420 1.0180872
##
##      C7
## Y           [,1]      [,2]
##   no  -0.1820198 0.7931912
##   yes  0.3382254 1.2632680
##
##      C8
## Y           [,1]      [,2]
##   no  -0.2324535 0.9400565
##   yes  0.3942131 0.9562671
```

```
nb.predsx <- predict(nb.fitx, test.x)
table(nb.predsx, test.x$disease)
```

```
##
## nb.predsx no yes
##       no  87  17
##       yes 19  34
```

```
sum(nb.predsx==test.x$disease)/nrow(test.x)
```

```
## [1] 0.7707006
```

```
prop.table(table(nb.predsx, test.x$disease)[1,])
```

```
##        no       yes
## 0.8365385 0.1634615
```

```
prop.table(table(nb.predsx, test.x$disease)[2,])
```

```
##        no       yes
## 0.3584906 0.6415094
```

```
nb.probsx <-  predict(nb.fitx, test.x, type = 'raw')
head(nb.probsx)
```

```
##               no         yes
## [1,] 0.985907220 0.014092780
## [2,] 0.005140088 0.994859912
## [3,] 0.961526058 0.038473942
## [4,] 0.893180132 0.106819868
## [5,] 0.005336028 0.994663972
## [6,] 0.997819786 0.002180214
```

As we can see from the results we notice that there is no substantial difference after having standardized the training set and the test set.

Exercise 5

Compare all the methods considered so far on the test data (logistic regression, k-nn, LDA, QDA, NB).

```
Models <-  c('Logistic regression','LDA','QDA','KNN','Naive Bayes')
Accuracy <-  c(glm.acc, lda.acc, qda.acc, mean(knn_acc), nb_acc)
recap_table <-  cbind(Models,Accuracy)
print(recap_table)
```

```
##      Models                Accuracy
## [1,] "Logistic regression" "0.78343949044586"
## [2,] "LDA"                 "0.777070063694268"
## [3,] "QDA"                 "0.78343949044586"
## [4,] "KNN"                 "0.560392798690671"
## [5,] "Naive Bayes"         "0.770700636942675"
```

In our analysis we have considered five different classification approaches: Logistic regression; Linear discriminant analysis, Quadratic discriminant analysis, K-nearest neighbor and the Naive Bayes.
The logistic regression and the LDA methods are closely connected because they both produce linear decision boundaries; the only difference between the two approaches lies in the fact the parameters are estimated using the maximum likelihood while in the LDA they are computed using the estimated mean and variance from a normal distribution. Since LDA and logistic regression differ only in their fitting procedures the fact that we got similar results do not surprise us. However, we could also do the following observation: since LDA assumes that the observation are drawn from a Gaussian distribution with a common covariance matrix in each class and we have noticed in our first inspection of the data set that these assumptions do not hold for all the predictors so, we could prefer the logistic regression. Moving to the KNN we recall that this model completely differs from the other since it is a non-parametric approach. Hence, no assumptions are made about the shape of the decision boundary. Therefore, we expect this approach to dominate the LDA and logistic regression when the decision boundary is highly non linear. On the other hand, KNN does not tell us which predictors are important; we do not get a table of coefficients. Moreover, regarding the KNN we should prefer the prediction made on the standardized data sets because scaling the data is useful to avoid biases caused by the different scale of the numerical variables. The QDA is a compromise between the non-parametric KNN method and the linear LDA and logistic regression approaches. Since QDA assumes a quadratic decision boundary, it can accurately model a wider range of problems than can be a linear methods. Though not as flexible KNN, QDA can perform better in the presence of a limited number of training observation, as in our case, because it does make some assumption about the form of the decision boundary. Finally, we have the Naive Bayes which is a classifier based on the application of the Bayes'theorem with the strong (naive) independence assumption between the features. The Bayes classifier requires the knowledge of the priori probabilities and the condition relative to the problem; these information are typically unknown but can be easily estimated. If these estimates of the probabilities are reliable the Bayes classifier is generally good. However, since we are dealing with medical variables it seems reasonable to think that the assumption of independence is not supported (We are not expert of medicine but typically some variables are highly correlated in this field such as the blood pressure and the weight or the percentage of glucose in the blood and again the weight). To conclude, we have to say that there is not a clear right answer to determine which is the absolute best model. Indeed, we have to face a trade off between the accuracy level and the interpretability of the results. However, our intuition will lead us to choose the QDA as the best model among the others due to the higher level of accuracy and its property of being a sort of compromise between all the other methods.

Draw the ROC curve, combining all of the ROC curves in a single plot. Compute also the AUC for each method. Discuss whether there is any method clearly outperforming the rest.

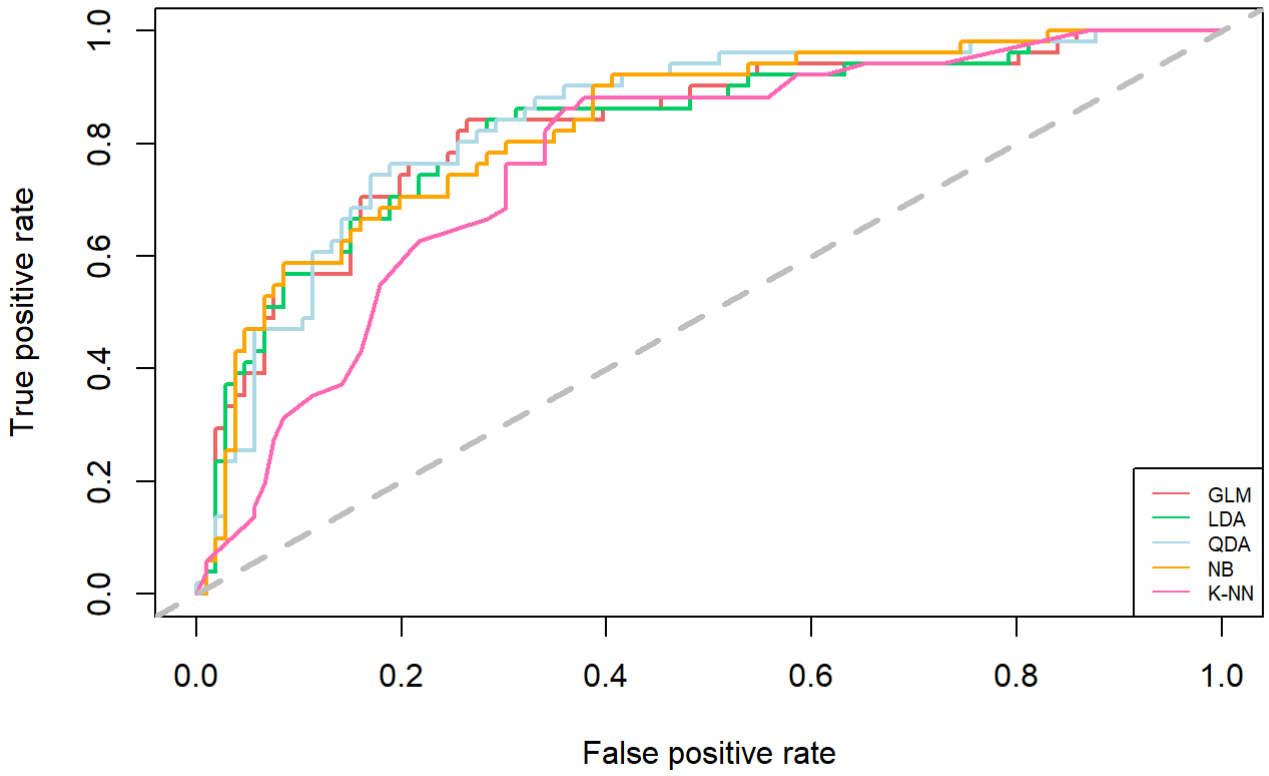
Now the next task is to plot the ROC curve and to compute the area under the curve.
The *ROC curve* is a popular graphic for simultaneously displaying the two types of errors for all the possible threshold. The overall performance of a classifier, summarized over all the possible thresholds, is given by the

area under the (ROC) curve (AUC). An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier.
ROC curves are useful to compare different classifiers, since they can take into account both possible classification error (FP and FN) for every possible threshold

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 4.0.4
```

```
predrglm <-  ROCR::prediction(glm.probs, test$disease)
perfglm <-  ROCR::performance(predrglm, "tpr", "fpr")
plot(perfglm, col = "indianred2", lwd = 2)
auc.glm <-  ROCR::performance(predrglm, measure = "auc", main = "ROC Curve")


predrlda <-  ROCR::prediction(lda.pred$posterior[,2], test$disease)
perflda <-  ROCR::performance(predrlda, "tpr", "fpr")
plot(perflda, col = "springgreen3", lwd = 2, add = T)
auc.lda <-  ROCR::performance(predrlda, measure = "auc")


predrqda <-  ROCR::prediction(qda.pred$posterior[,2], test$disease)
perfqda <-  ROCR::performance(predrqda, "tpr", "fpr")
plot(perfqda, col = "lightblue", lwd = 2, add = T)
auc.qda <-  ROCR::performance(predrqda, measure = "auc")


predrnb <-  ROCR::prediction(nb.probs[,2], test$disease)
perfnb <-  ROCR::performance(predrnb, "tpr", "fpr")
plot(perfnb, col = "orange", lwd = 2, add = T)
auc.nb <-  ROCR::performance(predrnb, measure = "auc")

# Since there is no predict for KNN method, I will have to extract the probability of predict
ion in the KNN testing with prob = True, otherwise we will get the predicted classes ( the pr
oportions are needed)
pred_ts_prob <- attr(pred_ts, "prob")
pred_ts_prob[pred_ts == "no"] <-  1 - pred_ts_prob[pred_ts == "no"]
# KNN outputs the posterior probability of the winning class, we need to re-scale it.
predrknn <-  ROCR::prediction(pred_ts_prob, test$disease)
perfknn <-  ROCR::performance(predrknn, "tpr", "fpr")
plot(perfknn, col = "hotpink", lwd = 2, add = T)
auc.knn <-  ROCR::performance(predrknn, measure = "auc")
abline(0,1, col = "grey", lwd = 3, lty = 2)
legend('bottomright', legend = c('GLM','LDA', 'QDA','NB', 'K-NN'), col=c('indianred2','spring
green3','lightblue','orange','hotpink'), lty = 1, cex = .64)
```




```
auc_table <-  data.frame(Glm = auc.glm@y.values[1], Lda = auc.lda@y.values[1], Qda = auc.qda@
y.values[1], NB = auc.nb@y.values[1], Knn = auc.knn@y.values[1])
auc_table
```

```
##   X0.829078801331854 X0.826489086200518 X0.842212356640769 X0.833888272290048
## 1          0.8290788          0.8264891          0.8422124          0.8338883
##   X0.77219755826859
## 1         0.7721976
```

```
max_auc <-  max(auc_table)
min_auc <- min(auc_table)
```

```
auc_vector <-  c(auc.glm@y.values[1], auc.lda@y.values[1], auc.qda@y.values[1],auc.knn@y.valu
es[1], auc.nb@y.values[1])
auc_summ_table <-  cbind(Models, auc_vector)
print(auc_summ_table)
```

```
##      Models                auc_vector
## [1,] "Logistic regression" 0.8290788
## [2,] "LDA"                 0.8264891
## [3,] "QDA"                 0.8422124
## [4,] "KNN"                 0.7721976
## [5,] "Naive Bayes"         0.8338883
```

Analyzing the results we see that the model with the highest value for the AUC is the QDA. However, the difference is really small indeed, looking at the graph we see how the different ROC curves are all pretty close one with each other. The model with the lowest ROC curve and consequently with the smallest AUC is the KNN.

Reflect on what is not ideal on this comparative analysis, which could bias the results in favor of one of the methods (which one?)? Do you see this on the results?

As mentioned before, it looks difficult to determine which is the best methods also when we compare the the different ROC curves and the different areas under the curves. Indeed, we can easily notice that all the curves overlaps over each other. This create a bias in our analysis and makes difficult to clearly state which model we should prefer also because the difference between the values of the AUC is pretty small. It is also difficult to assess which model could be favored due to a bias. A reasonable hypothesis is that the naive Bayes is favored because comparing both the accuracy and the AUC we notice that the NB has a much higher value of AUC compared to the LDA and the logistic regression while their accuracy is almost identical. Our final thought is that a linear model such as the logistic regression could lead us to a model which is easier to interpret but since the classes are not well separated the assumption of having a linear decision boundary may not hold so once again our preferences go in favor to the QDA.

Exercise 6

Since the predictors are continuous, there could be also the option of including polynomial terms. In the context of logistic regression.

Consider only the predictor C2 and fit a model which includes higher orders of this variable in the model;

Explore different degrees and check the performance on test data to reach a conclusion on the optimal degree;

Since the predictors are continuous we could include polynomial terms in the context of logistic regression. The introduction of polynomial terms is helpful in the case the true relationship between the response and the predictors is non linear. In this case we also want to investigate the performance of the polynomial regression with different degrees. To do so we simply create a for cycle where we will perform a logistic regression with the addition of polynomial terms up to the twelfth degree and then we simply make the predictions as before.

```
my_seq <-  c(seq(1:12))
v_test <-  c()
v_train <-  c()
poly_acc_train <-  c()
poly_acc_test <- c()
poly_err_train <-  c()
poly_err_test <- c()
true_neg <-  c()
true_pos <-  c()
for (i in my_seq){
  model <-  glm(disease ~ poly(C2,i), data = train, family = 'binomial')
  print(summary(model))
  poly_prob_test <-  predict(model, test, type = 'response')
  poly_pred_test <-  rep('no', length(test))
  poly_pred_test <-  ifelse(poly_prob_test > 0.5, 'yes', 'no')
  poly_prob_train <-  predict(model, train, type = 'response')
  poly_pred_train <- rep('no', length(train))
  poly_pred_train <- ifelse(poly_prob_train > 0.5, 'yes', 'no')

  # confusion matrix
  table(poly_pred_test, test$disease)
  # TN & TP
  true_pos <-  c(true_pos, table(poly_pred_test, test$disease)[2,2]/sum(table(poly_pred_test,
test$disease)[,2]))
  true_neg <-  c(true_neg, table(poly_pred_test, test$disease)[1,1]/sum(table(poly_pred_test,
test$disease)[,1]))
  # accuracy
  poly_acc_test <-  append(poly_acc_test, mean(poly_pred_test==test$disease))
  poly_err_test <-  append(poly_err_test, mean(poly_pred_test!=test$disease))
  poly_acc_train <-  append(poly_acc_train, mean(poly_pred_train==train$disease))
  print(mean(poly_pred_train ==  train$disease))
   poly_err_train <-  append(poly_err_train, mean(poly_pred_train!=train$disease))
  print(mean(poly_pred_train ==  train$disease))
}
```

```
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1997  -0.7740  -0.4462   0.7207   2.4375
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.8816     0.1691  -5.214 1.85e-07 ***
## poly(C2, i)  21.1750     3.0575   6.926 4.34e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 229.24  on 233  degrees of freedom
## AIC: 233.24
##
## Number of Fisher Scoring iterations: 4
##
## [1] 0.7574468
## [1] 0.7574468
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0611  -0.7899  -0.4209   0.7422   2.5462
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.9285     0.1909  -4.863 1.16e-06 ***
## poly(C2, i)1  21.5705     3.1716   6.801 1.04e-11 ***
## poly(C2, i)2  -1.8821     3.2471  -0.580    0.562
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 228.90  on 232  degrees of freedom
## AIC: 234.9
##
## Number of Fisher Scoring iterations: 5
##
## [1] 0.7531915
## [1] 0.7531915
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
```

```
## -2.1303  -0.8011  -0.4179   0.7560   2.6071
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.9387     0.1979  -4.744 2.09e-06 ***
## poly(C2, i)1  22.0568     3.7052   5.953 2.63e-09 ***
## poly(C2, i)2  -2.1956     3.5506  -0.618    0.536
## poly(C2, i)3   0.9718     3.4577   0.281    0.779
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 228.82  on 231  degrees of freedom
## AIC: 236.82
##
## Number of Fisher Scoring iterations: 5
##
## [1] 0.7531915
## [1] 0.7531915
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1154  -0.8008  -0.4203   0.7496   2.6233
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.9417     0.2023  -4.655 3.24e-06 ***
## poly(C2, i)1  22.1275     3.8364   5.768 8.03e-09 ***
## poly(C2, i)2  -2.3774     4.2080  -0.565    0.572
## poly(C2, i)3   1.0988     3.8115   0.288    0.773
## poly(C2, i)4  -0.3060     3.6743  -0.083    0.934
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 228.82  on 230  degrees of freedom
## AIC: 238.82
##
## Number of Fisher Scoring iterations: 6
##
## [1] 0.7574468
## [1] 0.7574468
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0983  -0.8166  -0.4027   0.7361   2.6037
##
## Coefficients:
```

```
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.9372     0.1979  -4.735 2.19e-06 ***
## poly(C2, i)1  21.9661     3.6682   5.988 2.12e-09 ***
## poly(C2, i)2  -2.0409     3.8742  -0.527    0.598
## poly(C2, i)3   0.3010     3.8589   0.078    0.938
## poly(C2, i)4   0.2581     3.5744   0.072    0.942
## poly(C2, i)5  -1.3474     3.2557  -0.414    0.679
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 228.65  on 229  degrees of freedom
## AIC: 240.65
##
## Number of Fisher Scoring iterations: 6
##
## [1] 0.7617021
## [1] 0.7617021
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0938  -0.8171  -0.4018   0.7341   2.6095
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.9378     0.1983  -4.730 2.25e-06 ***
## poly(C2, i)1  21.9676     3.6600   6.002 1.95e-09 ***
## poly(C2, i)2  -2.0420     3.8485  -0.531    0.596
## poly(C2, i)3   0.2781     3.8508   0.072    0.942
## poly(C2, i)4   0.3263     3.8538   0.085    0.933
## poly(C2, i)5  -1.3986     3.4328  -0.407    0.684
## poly(C2, i)6   0.1438     3.2533   0.044    0.965
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 228.65  on 228  degrees of freedom
## AIC: 242.65
##
## Number of Fisher Scoring iterations: 6
##
## [1] 0.7617021
## [1] 0.7617021
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9952  -0.8342  -0.3917   0.6930   2.5163
##
```

```
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.93820    0.19909  -4.713 2.45e-06 ***
## poly(C2, i)1 21.99072    3.75344   5.859 4.66e-09 ***
## poly(C2, i)2 -2.10908    4.13858  -0.510    0.610
## poly(C2, i)3  0.46668    4.50054   0.104    0.917
## poly(C2, i)4 -0.03216    4.69579  -0.007    0.995
## poly(C2, i)5 -0.57938    4.41941  -0.131    0.896
## poly(C2, i)6 -0.48172    3.84854  -0.125    0.900
## poly(C2, i)7  1.51818    3.38042   0.449    0.653
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 228.44  on 227  degrees of freedom
## AIC: 244.44
##
## Number of Fisher Scoring iterations: 6
##
## [1] 0.7659574
## [1] 0.7659574
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9749  -0.8211  -0.4140   0.6686   2.6220
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -0.9348     0.1988  -4.701 2.58e-06 ***
## poly(C2, i)1  22.0504     3.7293   5.913 3.36e-09 ***
## poly(C2, i)2  -2.0116     3.9774  -0.506    0.613
## poly(C2, i)3   0.4487     4.0850   0.110    0.913
## poly(C2, i)4   0.1763     4.0794   0.043    0.966
## poly(C2, i)5  -0.8960     3.8603  -0.232    0.816
## poly(C2, i)6   0.3435     3.7971   0.090    0.928
## poly(C2, i)7   0.9105     3.4044   0.267    0.789
## poly(C2, i)8   1.5482     3.2840   0.471    0.637
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 228.22  on 226  degrees of freedom
## AIC: 246.22
##
## Number of Fisher Scoring iterations: 6
##
## [1] 0.7659574
## [1] 0.7659574
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2938  -0.7770  -0.3704   0.7502   2.5047
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.0068     0.2655  -3.792 0.000149 ***
## poly(C2, i)1  25.9350     6.7875   3.821 0.000133 ***
## poly(C2, i)2  -4.5729     8.2939  -0.551 0.581387
## poly(C2, i)3   6.4342    11.2864   0.570 0.568624
## poly(C2, i)4  -4.3141    11.7405  -0.367 0.713280
## poly(C2, i)5   5.2451    11.6607   0.450 0.652846
## poly(C2, i)6  -4.9188    10.1361  -0.485 0.627482
## poly(C2, i)7   7.9515     8.5616   0.929 0.353021
## poly(C2, i)8  -3.1570     6.0806  -0.519 0.603632
## poly(C2, i)9   7.6726     4.7984   1.599 0.109818
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 224.84  on 225  degrees of freedom
## AIC: 244.84
##
## Number of Fisher Scoring iterations: 8
##
## [1] 0.7489362
## [1] 0.7489362
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1657  -0.7346  -0.3644   0.7088   2.6264
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -1.2844     0.5613  -2.288   0.0221 *
## poly(C2, i)1   32.2962    14.6635   2.202   0.0276 *
## poly(C2, i)2  -15.6231    21.7347  -0.719   0.4723
## poly(C2, i)3   18.4288    27.1685   0.678   0.4976
## poly(C2, i)4  -19.1696    30.3463  -0.632   0.5276
## poly(C2, i)5   17.8557    27.6765   0.645   0.5188
## poly(C2, i)6  -17.4937    24.9058  -0.702   0.4824
## poly(C2, i)7   17.7764    19.3097   0.921   0.3573
## poly(C2, i)8  -12.3428    14.9352  -0.826   0.4086
## poly(C2, i)9   13.0542     9.1512   1.426   0.1537
## poly(C2, i)10  -5.2758     5.8840  -0.897   0.3699
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 223.93  on 224  degrees of freedom
## AIC: 245.93
##
## Number of Fisher Scoring iterations: 10
##
## [1] 0.7574468
## [1] 0.7574468
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

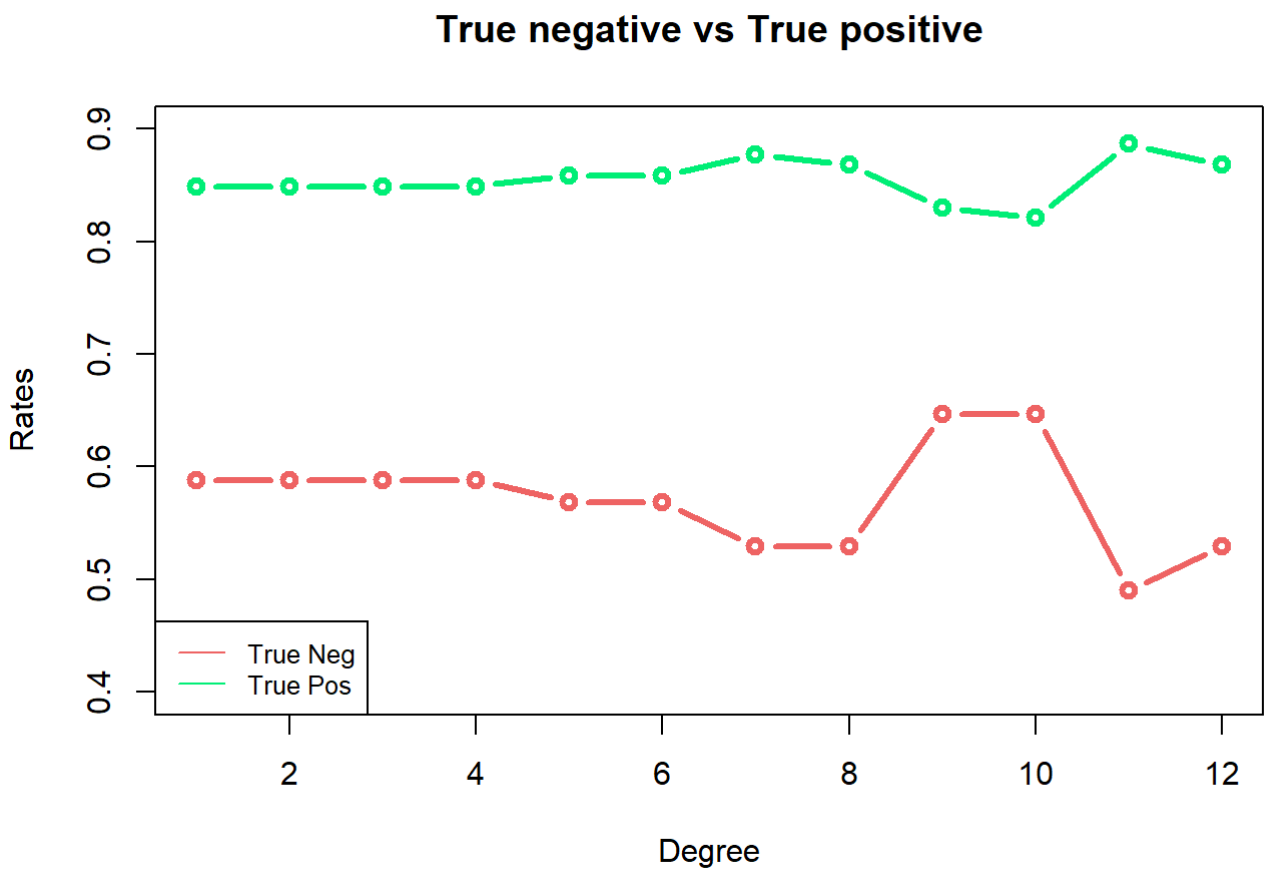
```
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2853  -0.7494  -0.3241   0.7436   3.0495
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -3.611      1.653  -2.185   0.0289 *
## poly(C2, i)1    98.940     46.154   2.144   0.0321 *
## poly(C2, i)2  -108.524     65.568  -1.655   0.0979 .
## poly(C2, i)3   143.060     85.568   1.672   0.0945 .
## poly(C2, i)4  -148.809     90.586  -1.643   0.1004
## poly(C2, i)5   143.635     85.397   1.682   0.0926 .
## poly(C2, i)6  -126.637     74.155  -1.708   0.0877 .
## poly(C2, i)7   108.704     59.553   1.825   0.0679 .
## poly(C2, i)8   -82.450     45.043  -1.830   0.0672 .
## poly(C2, i)9    64.781     31.143   2.080   0.0375 *
## poly(C2, i)10  -33.705     16.973  -1.986   0.0470 *
## poly(C2, i)11   18.103      9.182   1.972   0.0487 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 219.19  on 223  degrees of freedom
## AIC: 243.19
##
## Number of Fisher Scoring iterations: 12
##
## [1] 0.7744681
## [1] 0.7744681
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

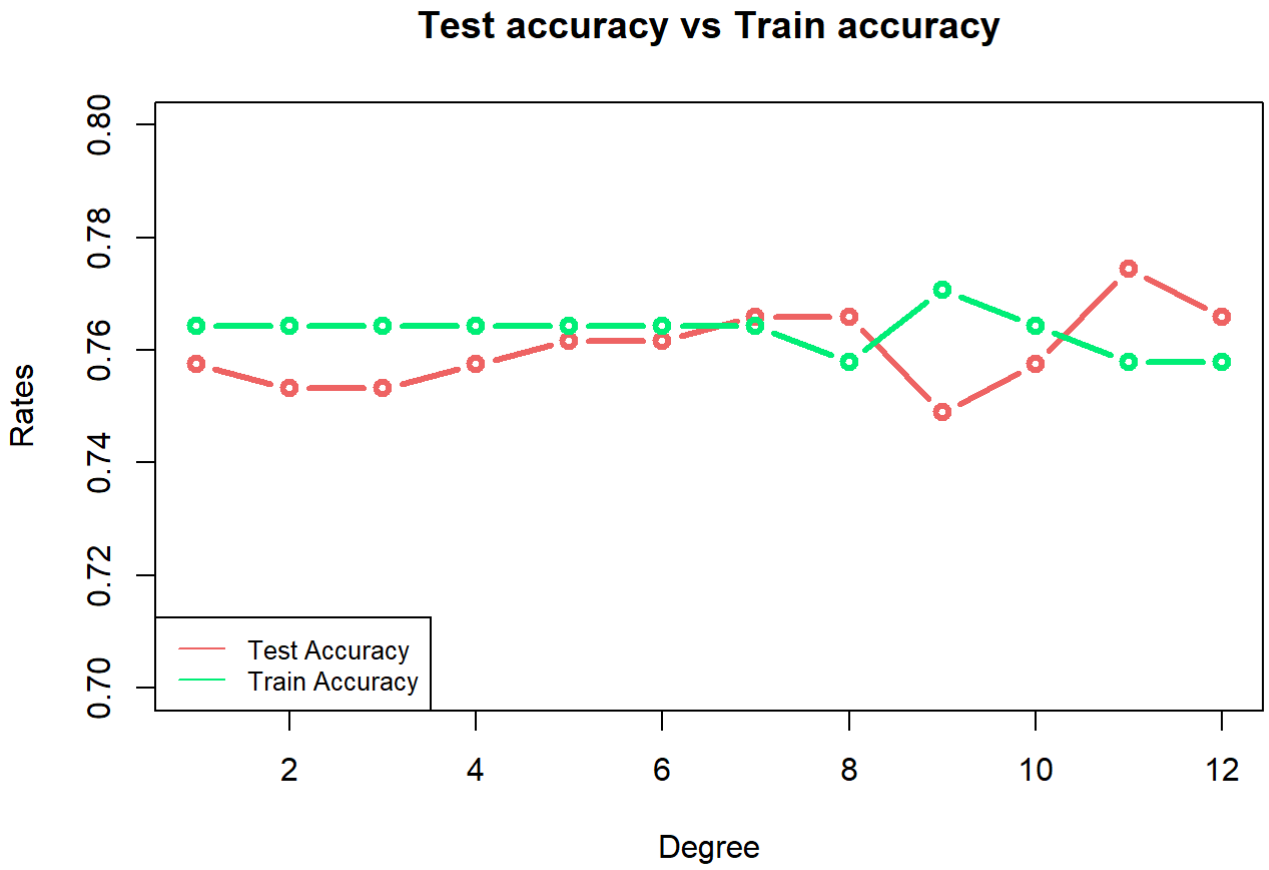
```
##
## Call:
## glm(formula = disease ~ poly(C2, i), family = "binomial", data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.2805  -0.7554  -0.3112   0.7265   2.9284
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -1.698      2.166  -0.784    0.433
## poly(C2, i)1    49.056     57.750   0.849    0.396
## poly(C2, i)2   -28.817     89.067  -0.324    0.746
## poly(C2, i)3    46.472    110.473   0.421    0.674
## poly(C2, i)4   -36.506    125.143  -0.292    0.771
## poly(C2, i)5    43.972    113.277   0.388    0.698
## poly(C2, i)6   -31.556    105.523  -0.299    0.765
## poly(C2, i)7    34.791     83.078   0.419    0.675
## poly(C2, i)8   -19.047     69.401  -0.274    0.784
## poly(C2, i)9    21.818     47.568   0.459    0.646
## poly(C2, i)10   -2.305     33.223  -0.069    0.945
## poly(C2, i)11    3.016     16.560   0.182    0.855
## poly(C2, i)12    9.255      9.443   0.980    0.327
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 300.08  on 234  degrees of freedom
## Residual deviance: 218.27  on 222  degrees of freedom
## AIC: 244.27
##
## Number of Fisher Scoring iterations: 11
##
## [1] 0.7659574
## [1] 0.7659574
```

Now we simply want to plot the results just obtained: so we decide to create three different graphs where we plot the true positive rate and the true negative rate; the test and train accuracy; test and train error.

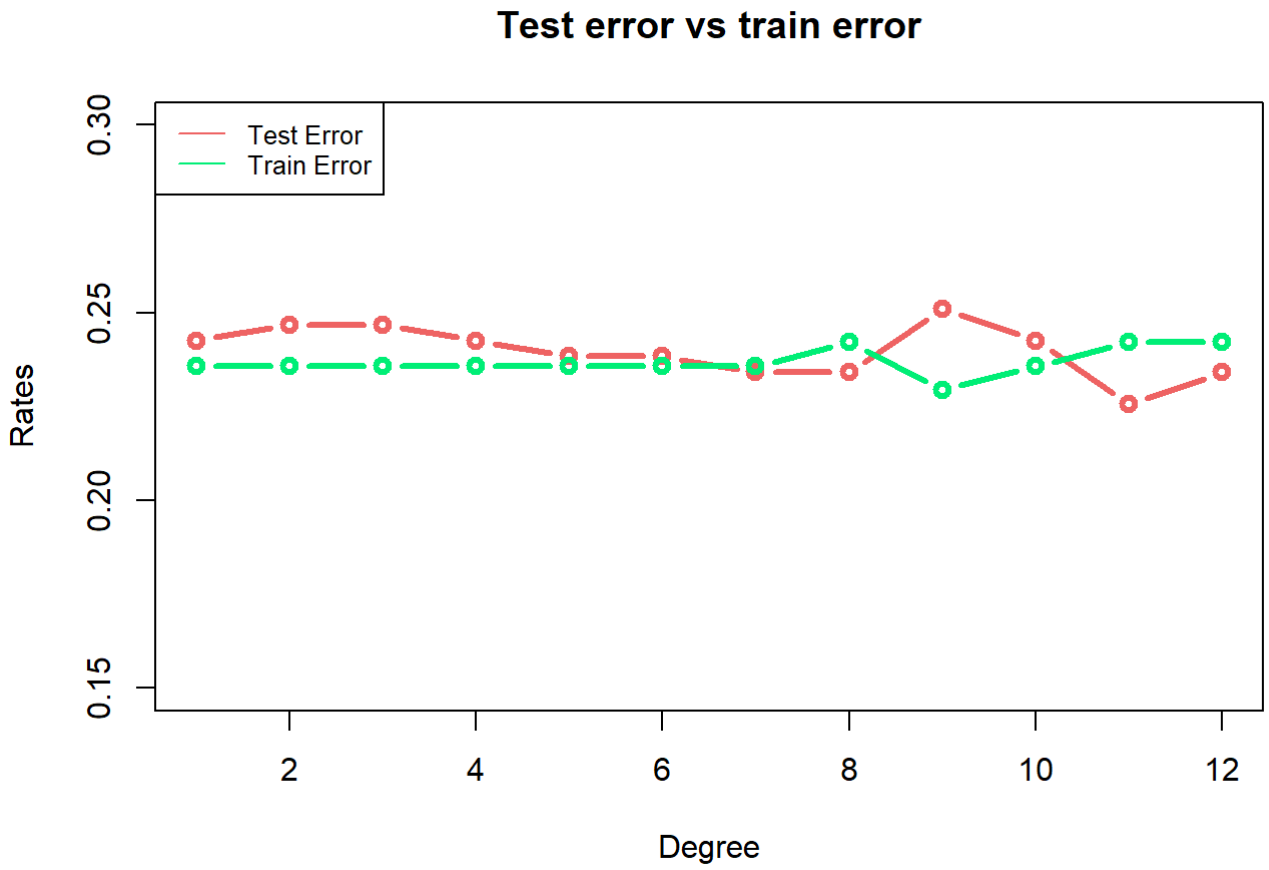
```
# creating a blank plot to fill with poly lines afterwards
plot(my_seq, type = 'n', xlab = 'Degree', ylab = 'Rates', ylim = c(0.4,0.9), xlim = c(1,12),
 main = 'True negative vs True positive')
# add lines for MSE on train and test lines
lines(my_seq, true_pos, col = 'indianred2', lwd = 3, type = 'b')
lines( my_seq, true_neg, col = 'springgreen2', lwd = 3, type = 'b' )
legend('bottomleft', legend = c('True Neg', 'True Pos'), col = c('indianred2', 'springgreen2'
), lty = 1.5, cex = .80)
```




```
# creating a blank plot to fill with poly lines afterwards
plot(my_seq, type = 'n', xlab = 'Degree', ylab = 'Rates', ylim = c(0.7,0.8), main = 'Test acc
uracy vs Train accuracy')
lines(my_seq, poly_acc_train, col = 'indianred2', lwd = 3, type = 'b')
lines( my_seq, poly_acc_test, col = 'springgreen2', lwd = 3, type = 'b' )
legend('bottomleft', legend = c('Test Accuracy', 'Train Accuracy'), col = c('indianred2', 'sp
ringgreen2'), lty = 1.5, cex = .80)
```




```
# creating a blank plot to fill with poly lines afterwards
plot(my_seq, type = 'n', xlab = 'Degree', ylab = 'Rates', ylim = c(0.15,0.3), main = 'Test er
ror vs train error')
lines(my_seq, poly_err_train, col = 'indianred2', lwd = 3, type = 'b')
lines( my_seq, poly_err_test, col = 'springgreen2', lwd = 3, type = 'b' )
legend('topleft', legend = c('Test Error', 'Train Error'), col = c('indianred2', 'springgreen
2'), lty = 1.5, cex = .80)
```



The first thing we can spot is that there is not a significant difference between the model created considering all the predictors and the one created considering only *C2* with the polynomial terms. According to the results of the analysis and their respective plots it seems reasonable to think that in this case the best option could be consider the polynomial of degree one since there is not a big difference in the accuracy-error rate and the true positive-true negative rate between the different degree of the polynomials. Moreover, a regression model with a lower degree is always easier to interpret compared to the ones with an higher degree.

Write the formula of your optimal model.

$log(p(1|x)/(1-p(1|x))) = \beta^t \cdot x = \beta 0 + \beta 1 \cdot x1 = -0.8816 + 21.1750 \cdot C2$