Exercice 4

1. Let prove that under the constraint $R\beta = r$ we have:

$$\hat{\beta}_c = \hat{\beta} + (X'X)^{-1} R'(R(X'X)^{-1}R')^{-1}(r - R\hat{\beta})$$

to do so, we will minimise the sum of squares $(Y - X\beta)'(Y - X\beta)$ subject to the condition that $R\beta = r$. This lead us to the following Lagrangean function:

$$(1) \Rightarrow L = (Y - X\beta)'(Y - X\beta) + 2\lambda'(R\beta - r)$$

* The factor "2" is introduced to simplify the later steps of optimization process. It does not fundamentally change the nature of the optimization problem but helps on mathematical derivations and calculation.

Let's apply the first order condition $\frac{\partial L}{\partial \beta} = 0$ and $\frac{\partial L}{\partial \lambda} = 0$

$$(1) \Rightarrow L = Y'Y - Y'X\beta - \beta'X'Y - \beta'X'X\beta + 2\lambda'R\beta - 2\lambda'r$$

$$\Rightarrow L = Y'Y - 2Y'X\beta + \beta'X'X\beta + 2\lambda'R\beta - 2\lambda'r \text{ ; with } (\beta'X'y)' = Y'X\beta$$

$$\frac{\partial L}{\partial \beta} = 0 \Leftrightarrow 0 - 2Y'X + 2\beta'X'X + 2\lambda'R + 0 = 0$$

$$\Leftrightarrow X'X\beta + R'\lambda = X'Y$$

$$\Leftrightarrow X'X\beta = X'Y - R'\lambda$$

* We assume that $X'X$ is invertible, so we have:

$$\Leftrightarrow \hat{\beta}_c = (X'X)^{-1}X'Y - (X'X)^{-1}R'\lambda$$

$$\underline{\hat{\beta}_c = \hat{\beta} - (X'X)^{-1}R'\lambda \quad (2)}$$

$\frac{\partial L}{\partial \lambda} = 0 \Leftrightarrow 2R\hat{\beta} - 2r = 0$

$\Leftrightarrow R\hat{\beta} = r.$

Let's found λ.

$R\hat{\beta}_c = r \Leftrightarrow R(\hat{\beta} - (x'x)^{-1}R'\lambda) = r$

$\Leftrightarrow R\hat{\beta} - R(x'x)^{-1}R'\lambda = r$

$\Leftrightarrow -R(x'x)^{-1}R'\lambda = r - R\hat{\beta}$

$\Leftrightarrow R(x'x)^{-1}R'\lambda = R\hat{\beta} - r$

We suppose that $R(x'x)^{-1}R'$ is invertible. We have :

$$R\hat{\beta}_c = r \Leftrightarrow \lambda = \left(R(x'x)^{-1}R'\right)^{-1}(R\hat{\beta} - r) \quad (3)$$

We substitute (3) in (2) and we have :

$$(2) \Rightarrow \hat{\beta}_c = \hat{\beta} - (x'x)^{-1}R'\left(R(x'x)^{-1}R'\right)^{-1}(R\hat{\beta} - r)$$

$$\boxed{\hat{\beta}_c = \hat{\beta} + (x'x)^{-1}.R'\left(R.(x'x)^{-1}.R'\right)^{-1}.(r - R\hat{\beta})}$$

0

2.

Let consider ANOVA with one factor, we denote by $a_1, a_2 \dots, a_I$ the different modalities.

$\forall i \in \{1, \dots, I\}$, $Y_{i,s} = \alpha_0 + \alpha_1 \mathbb{1}_{F_i = a_1} + \alpha_2 \mathbb{1}_{F_i = a_2} + \dots + \alpha_I \mathbb{1}_{F_i = a_I} + \varepsilon_{i,s}$

the matrix form is given by: $Y = X\beta + \varepsilon$, with:

$$\beta = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_I \end{pmatrix}, \quad X = \begin{pmatrix} 1 & 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 1 & 0 & \dots & \vdots \\ | & | & 0 & 1 & & \vdots \\ | & | & | & 0 & \ddots & \vdots \\ 1 & 0 & 0 & 0 & \dots & 1 \end{pmatrix} ; \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_I \end{pmatrix} ; Y = \begin{pmatrix} y_{1,1} \\ y_{2,1} \\ y_{3,1} \\ \vdots \\ y_{I,1} \end{pmatrix}$$

Remark: The form of X is an example among many possible. the representation will depend of the data. for example if in the dataset we have two observations of each modality, we will have

$$X = \begin{pmatrix} 1 & 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & 0 & & \vdots \\ \vdots & 0 & 1 & 0 & & \vdots \\ \vdots & | & 1 & 0 & & \vdots \\ \vdots & | & | & 1 & & \vdots \\ 1 & 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

We have to keep in mind that in the one factor ANOVA with m columns, the sum of the last m-1 columns is equal to the first one and it is always True.

Given that one column can be written as a linear combination of the others columns, we can deduce that $X'X$ is not inversible because the rank is not maximal.

the shape of the considered matrix are:

β :

X :

ε :

Y :

R :

In order to compute the estimate for the model, we consider :

a) the intercept is null.

If the intercept is null, $R = (0\ 1\ 1\ \dots\ 1)$

the model with this constraint becomes :

$$Y_{i,k} = \alpha_1 \mathbb{1}_{F_i = a_1} + \alpha_2 \mathbb{1}_{F_i = a_2} + \dots + \alpha_I \mathbb{1}_{F_i = a_I} + \varepsilon_{i,k}.$$

We deduce from that :

$$\tilde{X} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & \vdots \\ \vdots & \vdots & 1 & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix} \quad \text{and} \quad \tilde{\beta} = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_I \end{pmatrix}$$

Let's find $\tilde{\beta}$.

We can see that the columns of $\tilde{X}$ are orthogonal 2 by 2 :

We have :

$$\tilde{X}'\tilde{X} = \begin{pmatrix} n_1 & 0 & 0 & \dots & 0 \\ 0 & n_2 & 0 & \dots & \vdots \\ \vdots & 0 & n_3 & 0 & \vdots \\ \vdots & \vdots & 0 & \ddots & \vdots \\ 0 & 0 & 0 & \dots & n_I \end{pmatrix}$$

, where n_i is the total number of times the modality i appear in our dataset.

We obtain a diagonal matrix because the columns of $\tilde{X}$ are orthogonal 2 by 2 and given that $\tilde{X}$ is composed of 0 and 1 the elements on the diagonal are equal to total number of times there is one in a given columns.

* We also have :

$$\tilde{X}'Y = \begin{pmatrix} \sum_{k=1}^{n_1} y_{1,k} \\ \sum_{k=1}^{n_2} y_{2,k} \\ \vdots \\ \sum_{k=1}^{n_I} y_{3,k} \end{pmatrix}$$

, with k the number of appearance of a modality in the dataset.

So we have:

$$\tilde{\hat{\beta}} = (\tilde{X}'\tilde{X})^{-1}\tilde{X}'Y$$

$$\tilde{\hat{\beta}} = \begin{pmatrix} \frac{1}{n_1}\sum_{i=1}^{n_1} y_{i,B} \\ \frac{1}{n_2}\sum_{i=1}^{n_2} y_{i,B} \\ \vdots \\ \frac{1}{n_I}\sum_{i=1}^{n_I} y_{i,B} \end{pmatrix} = \begin{pmatrix} \tilde{\alpha}_1 \\ \tilde{\alpha}_2 \\ \vdots \\ \tilde{\alpha}_I \end{pmatrix}$$

b) the coefficient in the model associated to the first modality is null.

the expression of R is : $R = (0\ 1\ 0\ 0\ \cdots\ 0)$

the model becomes:

$$\forall i \in \{1, \dots, n\},\ Y_i = \alpha_0 + \alpha_2 \mathbb{1}_{F_i=a_2} + \alpha_3 \mathbb{1}_{F_i=a_3} + \dots + \alpha_I \mathbb{1}_{F_i=a_I} + \varepsilon_i$$

in this case, we have:

$$\tilde{X} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & 1 & 0 & \cdots & 0 \\ 1 & 0 & 1 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \cdots & 1 \end{pmatrix} \quad \text{and} \quad \beta = \begin{pmatrix} \alpha_0 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \alpha_I \end{pmatrix}$$

Let's find $\tilde{\hat{\beta}}$.

if $F=a_1$, $y_{1B} = \alpha_0 + \varepsilon_1 \Rightarrow E(y_{1B}) = \alpha_0$. with $E(\varepsilon_1) = 0$

by applying the method of moment, we know that an estimate of α_0 is solution of $E(y_{1B}) = \tilde{\alpha}_0 \Rightarrow$

$$\boxed{\tilde{\alpha}_0 = \frac{1}{n_1}\sum_{B=1}^{n_1} y_{1,B}}$$

if $F = a_2$, $y_{2,k} = \alpha_0 + \alpha_2 + \varepsilon_2 \Rightarrow E(y_{2,k}) = \alpha_0 + \alpha_2$

again by applying the moment method we have:

$$E(y_{2,k}) = \hat{\alpha}_0 + \hat{\alpha}_2 \Rightarrow \hat{\alpha}_2 = \frac{1}{n_2} \sum_{k=1}^{n_2} y_{2,k} - \hat{\alpha}_0$$

if $F = a_I$, $y_{I,k} = \alpha_0 + \alpha_I + \varepsilon_I \Rightarrow E(y_{I,k}) = \alpha_0 + \alpha_I$

with the method of moments we have:

$$E(y_{I,k}) = \hat{\alpha}_0 + \hat{\alpha}_I \Rightarrow \boxed{\hat{\alpha}_I = \frac{1}{n_I} \sum_{k=1}^{n_I} y_{I,k} - \hat{\alpha}_0}$$

So in that case we have:

$$\hat{\beta} = \begin{pmatrix} \hat{\alpha}_0 \\ \alpha_2 \\ \alpha_3 \\ \vdots \\ \hat{\alpha}_I \end{pmatrix} = \begin{pmatrix} \frac{1}{n_1} \sum_{k=1}^{n_1} y_{1,k} \\ \frac{1}{n_2} \sum_{k=1}^{n_2} y_{2,k} - \hat{\alpha}_0 \\ \vdots \\ \frac{1}{n_I} \sum_{k=1}^{n_I} y_{I,k} - \hat{\alpha}_0 \end{pmatrix}$$

EXAM ASML PART 2 Exercice 2 - Question 1

Jean-Luc BOA THIEMELE

2024-02-15

```
#Loading dataset
load("/Users/jlbt/Downloads/data_advanced.RData")
```

```
##Installing and loading library
#install.packages("rpart")
#install.packages("VSURF")
#install.packages("randomForest")
library("randomForest")
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library("VSURF")
library(rpart)
```

MODEL 1: Classification using CART model

```
#Let's define explanatories and predicted variable
X = A$X # Set of explanatory variables
Y = data.frame(Y = A$Y) # The variable to predict
data = cbind(X,Y)

set.seed(42)
#STEP 1 : Construction of the maximal tree
max_tree = rpart(Y ~ .,data=data,minsplit=2,cp=10^(-9))

#STEP 2 : Construction of the best tree
best_cart_model=function (T)
{
  table_of_cp = T$cptable
  #We extract the cross validation error columns
  cross_validation_error=table_of_cp[, 4]

  #We get the index of the minimum of the CV error
  index_min_cv = which(cross_validation_error==min(cross_validation_error))

  # We compute the threshold as min(cv_error) + its standard deviation
  threshold = min(table_of_cp[index_min_cv, 4] + table_of_cp[index_min_cv, 5])

  #We extract the index of all the cross validationerror less or equal to the threshold
  index_cv_inf_thres = which(cross_validation_error<=threshold)
  best_cp_index =index_cv_inf_thres[1] #Get the first index of the index of cv inferior to threshold =>
  Tf=prune(T, cp=table_of_cp[best_cp_index, 1]) # We prune the maximal tree by using the best cp value
  best_cart_model=Tf
```

```
}

cart_model = best_cart_model(max_tree) #Best model using Cart Algo
print(cart_model)
```

```
## n= 77
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 77 37 -1 (0.51948052 0.48051948)
##   2) V2< 0.2506549 43  6 -1 (0.86046512 0.13953488)
##     4) V6< 0.6835541 38  1 -1 (0.97368421 0.02631579) *
##     5) V6>=0.6835541 5  0 1 (0.00000000 1.00000000) *
##   3) V2>=0.2506549 34  3 1 (0.08823529 0.91176471) *
```

Model 2 : Using VSURF for variables selection and CART for generating the best tree after the interpret step and the prediction step

```
# Custom column names
set.seed(42)
#We perform variable selection with VSURF
variable_selection = VSURF(data$Y ~.,data=data)
```

```
## Thresholding step
## Estimated computational time (on one core): 2.7 sec.
##   |                                                                                                  |
## Interpretation step (on 16 variables)
## Estimated computational time (on one core): between 0.3 sec. and  0.6 sec.
##   |                                                                                                  |
## Prediction step (on 5 variables)
## Maximum estimated computational time (on one core): 0.1 sec.
##   |                                                                                                  |
## Warning in VSURF.formula(data$Y ~ ., data = data): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data
```

```
# variables selected after interpretation step
selection_interp_step = variable_selection$varselect.interp

# variables selected after prediction step
selection_pred_step = variable_selection$varselect.pred

#Reordering indices to get indices of the original data
selection_interp_step = data[,c(colnames(data[selection_interp_step]),"Y")]
selection_pred_step = data[,c(colnames(data[selection_pred_step]),"Y")]

#We contruct the maximal tree using CART
max_tree_interp = rpart(Y ~ .,data=selection_interp_step,minsplit=2,cp=10^(-9))
max_tree_pred = rpart(Y ~ .,data=selection_pred_step,minsplit=2,cp=10^(-9))

# we determine the best model
best_tree_interp = best_cart_model(max_tree_interp)
best_tree_pred = best_cart_model(max_tree_pred)
```

```r
print(best_tree_interp)
```

```
## n= 77
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 77 37 -1 (0.51948052 0.48051948)
##   2) V2< 0.2506549 43  6 -1 (0.86046512 0.13953488)
##     4) V6< 0.6835541 38  1 -1 (0.97368421 0.02631579)
##       8) V5< 1.408609 37  0 -1 (1.00000000 0.00000000) *
##       9) V5>=1.408609 1  0 1 (0.00000000 1.00000000) *
##     5) V6>=0.6835541 5  0 1 (0.00000000 1.00000000) *
##   3) V2>=0.2506549 34  3 1 (0.08823529 0.91176471)
##     6) V3< -0.3936446 2  0 -1 (1.00000000 0.00000000) *
##     7) V3>=-0.3936446 32  1 1 (0.03125000 0.96875000) *
```

```r
print(best_tree_pred)
```

```
## n= 77
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 77 37 -1 (0.51948052 0.48051948)
##   2) V3< 0.3527536 48  9 -1 (0.81250000 0.18750000)
##     4) V6< 0.8046648 40  1 -1 (0.97500000 0.02500000) *
##     5) V6>=0.8046648 8  0 1 (0.00000000 1.00000000) *
##   3) V3>=0.3527536 29  1 1 (0.03448276 0.96551724) *
```

Model 3 : Random forest

```r
set.seed(42)
### Using random forest
## We split the data
rf = randomForest(Y ~.,data=data)
print(rf)
```

```
##
## Call:
##  randomForest(formula = Y ~ ., data = data)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 14
##
##         OOB estimate of  error rate: 7.79%
## Confusion matrix:
##    -1  1 class.error
## -1 39  1   0.0250000
## 1   5 32   0.1351351
```

EVALUATE MODELS USING CROSS VALIDATION

```r
#Splitting function for cross validation
splitting=function(u,V,seed) #function to split a vector in V parts
{
  n=length(u)
```

```
  k=floor(n/V) #number of elements in mean in each part
  L=list() #initialization
  for (i in 1:(V-1))  #loop to create the V subsamples
  {
    set.seed(seed)
    u1=sample(u,k,replace=FALSE) #individuals in the subsample number i
    L=c(L,list(u1))  #to save the different subsamples
    b=c()

    #to know the position of the individuals in the vector u1 inside the vector u
    for (j in (1:k))
      b=c(b,which(u==u1[j]))
    u=u[-b] #we suppress the individuals that constitute the subsample number i
  }
  L=c(L,list(u))  #the last vector u is the last subsample
  splitting=L
}

crosserror=function(SL) # function to compute the cross validation error
{
  #Initialisation of test error variables
  testerr_interp=c()
  testerr_pred=c()
  testerr_rf=c()
  testerr_cart=c()

  V=length(SL)
  for (k in 1:V)
  {
    #step k
    Learning=data[-SL[[k]],] #learning sample at step k
    Test=data[SL[[k]],] #test sample at step k

    ###Random forest
    model_rf = randomForest(Learning$Y ~., data=Learning)

    ### Cart model
    max_tree = rpart(Y ~ .,data=Learning,minsplit=2,cp=10^(-9))
    model_cart = best_cart_model(max_tree)

    ### Cart model after interp step
    variable_selection = VSURF(Y ~.,data=Learning)
    selection_interp_step = variable_selection$varselect.interp
    selection_interp_step = Learning[,c(colnames(Learning[selection_interp_step]),"Y")]
    max_tree_interp = rpart(Y ~ .,data=selection_interp_step,minsplit=2,cp=10^(-9))
    model_interp = best_cart_model(max_tree_interp)

    ### Cart model after prediction step
    selection_pred_step = variable_selection$varselect.pred
    selection_pred_step = Learning[,c(colnames(Learning[selection_pred_step]),"Y")]
    max_tree_pred = rpart(Y ~ .,data=selection_pred_step,minsplit=2,cp=10^(-9))
    model_pred = best_cart_model(max_tree_pred)
```

```
    #Prediction with each model
    pred_rf=predict(model_rf,newdata=Test,type='class')
    pred_pred=predict(model_pred,newdata=Test,type='class')
    pred_interp=predict(model_interp,newdata=Test,type='class')
    pred_cart=predict(model_cart,newdata=Test,type='class')

    testerr_rf = c(testerr_rf, mean(Test$Y != pred_rf))
    testerr_pred = c(testerr_pred, mean(Test$Y != pred_pred))
    testerr_interp = c(testerr_interp, mean(Test$Y != pred_interp))
    testerr_cart = c(testerr_cart, mean(Test$Y != pred_cart))
  }
  crosserror=list( testerr_rf,testerr_cart, testerr_pred, testerr_interp) #a list with a first element
  #the second element is the same for the forward model
}
```

```
n = nrow(data) #number of individuals
u = 1:n #the number of all the individuals
V = 3 #number of split

means_per_row=c()
for (i in list(42,107,142)) #Prediction with different seed
{
  SL=splitting(u,V,seed=i)

  errors = crosserror(SL)

  # Convert the list to a matrix
  data_matrix <- do.call(rbind, errors)

  # Compute the mean for each row
  means_per_row <- c(means_per_row,rowMeans(data_matrix))
}
```

```
## Thresholding step
## Estimated computational time (on one core): 1.6 sec.
##   |                                                                      |
## Interpretation step (on 18 variables)
## Estimated computational time (on one core): between 0.2 sec. and  0.5 sec.
##   |                                                                      |
## Prediction step (on 5 variables)
## Maximum estimated computational time (on one core): 0.1 sec.
##   |                                                                      |
## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data
## Thresholding step
## Estimated computational time (on one core): 1.7 sec.
##   |                                                                      |
## Interpretation step (on 18 variables)
## Estimated computational time (on one core): between 0.2 sec. and  0.5 sec.
##   |                                                                      |
## Prediction step (on 7 variables)
## Maximum estimated computational time (on one core): 0.1 sec.
```

```
##   |                                                                      |
## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data
## Thresholding step
## Estimated computational time (on one core): 1.6 sec.
##   |                                                                      |
## Interpretation step (on 21 variables)
## Estimated computational time (on one core): between 0.4 sec. and  0.6 sec.
##   |                                                                      |
## Prediction step (on 6 variables)
## Maximum estimated computational time (on one core): 0.1 sec.
##   |                                                                      |
## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data
## Thresholding step
## Estimated computational time (on one core): 1.7 sec.
##   |                                                                      |
## Interpretation step (on 16 variables)
## Estimated computational time (on one core): between 0.3 sec. and  0.5 sec.
##   |                                                                      |
## Prediction step (on 4 variables)
## Maximum estimated computational time (on one core): 0 sec.
##   |                                                                      |
## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data
## Thresholding step
## Estimated computational time (on one core): 1.7 sec.
##   |                                                                      |
## Interpretation step (on 23 variables)
## Estimated computational time (on one core): between 0.2 sec. and  0.7 sec.
##   |                                                                      |
## Prediction step (on 6 variables)
## Maximum estimated computational time (on one core): 0.1 sec.
##   |                                                                      |
## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data
## Thresholding step
## Estimated computational time (on one core): 1.6 sec.
##   |                                                                      |
## Interpretation step (on 19 variables)
## Estimated computational time (on one core): between 0.4 sec. and  0.6 sec.
##   |                                                                      |
## Prediction step (on 5 variables)
## Maximum estimated computational time (on one core): 0.1 sec.
##   |                                                                      |
```

```
## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data

## Thresholding step
## Estimated computational time (on one core): 1.7 sec.
##   |                                                                      |
## Interpretation step (on 28 variables)
## Estimated computational time (on one core): between 0.3 sec. and  0.8 sec.
##   |                                                                      |
## Prediction step (on 8 variables)
## Maximum estimated computational time (on one core): 0.2 sec.
##   |                                                                      |

## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data

## Thresholding step
## Estimated computational time (on one core): 1.8 sec.
##   |                                                                      |
## Interpretation step (on 17 variables)
## Estimated computational time (on one core): between 0.2 sec. and  0.5 sec.
##   |                                                                      |
## Prediction step (on 5 variables)
## Maximum estimated computational time (on one core): 0.1 sec.
##   |                                                                      |

## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data

## Thresholding step
## Estimated computational time (on one core): 1.6 sec.
##   |                                                                      |
## Interpretation step (on 19 variables)
## Estimated computational time (on one core): between 0.2 sec. and  0.6 sec.
##   |                                                                      |
## Prediction step (on 6 variables)
## Maximum estimated computational time (on one core): 0.1 sec.
##   |                                                                      |

## Warning in VSURF.formula(Y ~ ., data = Learning): VSURF with a formula-type call outputs selected va
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data
```

```r
row_names <- c("Errors_rf","Errors_cart_model", "Erros_best_tree_pred", "Errors_best_tree_interp")
names(means_per_row) <- row_names


# Create sample data for demonstration
results <- matrix(means_per_row, nrow = 4, ncol = 3)

# Create a data frame with row names as seeds and column names as model names
results_df <- as.data.frame(results)
rownames(results_df) <- row_names
colnames(results_df) <- c(42,107,142)
```

```r
# Print the data frame
print(results_df)
```

```
##                                42        107       142
## Errors_rf                0.1190123 0.08839506 0.1027160
## Errors_cart_model        0.1817284 0.16839506 0.1540741
## Erros_best_tree_pred     0.1170370 0.12839506 0.0400000
## Errors_best_tree_interp 0.1170370 0.11506173 0.1407407
```

```r
"

We can see that depending of the choice of seed, the best model changes.
-for a seed of 42 the best models are those obtain with the selected
variables after the interpretation and prediction step.
-for a seed of 107, the best model is the random forest
-for a seed of 142, the best model is the model constructed
with the selected variables after the prediction step

From the results, the key observation is that the variability
introduced by randomness and the choice of seed during model
training, can significantly impact model stability, generalizability,
and interpretability. Indeed :

-The varying misclassification errors across different seeds
reflect the sensitivity of models to initial randomizations,
leading to different model structures and performance metrics.
This variability poses challenges in interpreting model results
and comparing the effectiveness of different algorithms.

-The observed differences in misclassification errors highlight
concerns regarding the models' ability to generalize well to unseen data.

-The fluctuating performance makes it difficult to select the
best model after obtaining cross-validation metrics.

###Solution:
In order to resolve these issues, we must increase the dataset
size to reduce the influence of randomness."
```

```
## [1] "\n\nWe can see that depending of the choice of seed, the best model changes. \n-for a seed of 4
```

EXAM ASML PART 2-Exercise 2 - question 2

Jean-Luc BOA THIEMELE

2024-01-15

```
# Install and load required packages
#install.packages("VSURF")
#install.packages("randomForest")
#install.packages("rpart")
```

```
library(rpart)
library("randomForest")
```

```
## randomForest 4.7-1.1
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(VSURF)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
# Load the PM10 dataset from the VSURF package.
data("jus")
```

```
# Custom column names
jus = jus[,c("NO","NO2","SO2","T.min","T.max","T.moy","DV.maxvv","DV.dom"
             ,"VV.max","VV.moy","PL.som","HR.min","HR.max"
             ,"HR.moy","PA.moy","GTrouen","GTlehavre","PM10")]
```

```
# We supress the na from the dataset --> 64
jus <- na.omit(jus)
rownames(jus) <- NULL
```

```
set.seed(123)
#Splitting data
n <- nrow(jus)
inTrain <- sample.int(n, size = 0.8 * n, replace = FALSE)
training <- jus[inTrain,]
test = jus[-inTrain,]
rownames(test) <- NULL
rownames(training) <- NULL
```

Function that construct the best tree after determining the maximal tree

```
best_cart_model=function (T)
{
  table_of_cp = T$cptable
  cross_validation_error=table_of_cp[, 4] #We extract the cross validation error columns
```

```
  #We get the index of the minimum of the CV error
  index_min_cv = which(cross_validation_error==min(cross_validation_error))

  # We compute the threshold as min(cv_error) + its standard deviation
  threshold = min(table_of_cp[index_min_cv, 4] + table_of_cp[index_min_cv, 5])

  #threshold=min(threshold)
  #We extract the index of all the cross validation error less or equal to the threshold
  index_cv_inf_thres = which(cross_validation_error<=threshold)

  #Get the first index of the index of cv inferior to threshold => best cp
  best_cp_index =index_cv_inf_thres[1]
  # We prune the maximal tree by using the best cp value to get the best cart model
  Tf=prune(T, cp=table_of_cp[best_cp_index, 1])

  best_cart_model=Tf
}
```

Performing variables selection

```
#We perform variable selection with VSURF
set.seed(123)
variable_selection = VSURF(PM10 ~.,data=training)
```

```
## Thresholding step
## Estimated computational time (on one core): 35.8 sec.
##   |                                                                              |
## Interpretation step (on 17 variables)
## Estimated computational time (on one core): between 5.6 sec. and  33.8 sec.
##   |                                                                              |
## Prediction step (on 12 variables)
## Maximum estimated computational time (on one core): 19.9 sec.
##   |                                                                              |
```

```
## Warning in VSURF.formula(PM10 ~ ., data = training): VSURF with a formula-type call outputs selected
## which are indices of the input matrix based on the formula:
## you may reorder these to get indices of the original data
```

MODEL 1: Training of the model using the variable selected after the interpret step

```
set.seed(123)
#The varaibles selected after the interpret step are : NO, NO2, GTrouen,
#VV.moy, GTlehavre, SO2, T.max, T.moy, PL.som, VV.max, HR.moy, DV.maxvv
selection_interp_step = variable_selection$varselect.interp

#re-sepecifying the columns in the dataset
dataset_interp_step = training[,c(colnames(training[selection_interp_step]),"PM10")]

#We contruct the maximal tree using CART
maximal_tree_interpret_variables = rpart(PM10 ~ .,data=dataset_interp_step,minsplit=2,cp=10^(-9))

#We determine the best tree using the variables selected
model_interp = best_cart_model(maximal_tree_interpret_variables)
print(model_interp)
```

```
## n= 825
```

```
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 825 61335.450 21.26182
##   2) NO< 57.5 797 36550.880 20.44542
##     4) GTrouen< 2.35 560 21737.000 18.62679
##       8) PL.som>=0.5 238  5056.996 15.99580 *
##       9) PL.som< 0.5 322 13814.860 20.57143 *
##     5) GTrouen>=2.35 237  8585.300 24.74262 *
##   3) NO>=57.5 28  9133.000 44.50000
##     6) VV.moy>=1.604167 24  2979.333 38.66667 *
##     7) VV.moy< 1.604167 4   437.000 79.50000 *
```

Model 2: Training of the model using the variable selected after the prediction step

```
set.seed(123)
#The variables selected after the prediction step are : NO, GTrouen, VV.moy, GTlehavre,
#SO2, T.max, PL.som
selection_pred_step = variable_selection$varselect.pred

#re-specifying the columns in the dataset
dataset_pred_step = training[,c(colnames(training[selection_pred_step]),"PM10")]
maximal_tree_predict_variables = rpart(PM10 ~ .,data=dataset_pred_step,minsplit=2,cp=10^(-9))

#We determine the best tree using the variables selected
model_predict = best_cart_model(maximal_tree_predict_variables)
print(model_predict)
```

```
## n= 825
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
## 1) root 825 61335.450 21.26182
##   2) NO< 57.5 797 36550.880 20.44542
##     4) GTrouen< 2.35 560 21737.000 18.62679 *
##     5) GTrouen>=2.35 237  8585.300 24.74262 *
##   3) NO>=57.5 28  9133.000 44.50000
##     6) VV.moy>=1.604167 24  2979.333 38.66667 *
##     7) VV.moy< 1.604167 4   437.000 79.50000 *
```

MODEL 3 : Training of the model using CART algorithm

```
set.seed(123)
# Construction of the maximal tree
max_tree = rpart(PM10 ~ .,data=training,minsplit=2,cp=10^(-9))

#Construction of the best
cart_model = best_cart_model(max_tree)
print(cart_model)
```

```
## n= 825
##
## node), split, n, deviance, yval
##       * denotes terminal node
```

```
##
## 1) root 825 61335.450 21.26182
##   2) NO< 57.5 797 36550.880 20.44542
##     4) GTrouen< 2.35 560 21737.000 18.62679 *
##     5) GTrouen>=2.35 237  8585.300 24.74262 *
##   3) NO>=57.5 28  9133.000 44.50000
##     6) VV.moy>=1.604167 24  2979.333 38.66667 *
##     7) VV.moy< 1.604167 4   437.000 79.50000 *
```

MODEL 4 : Training of the model using Random Forest

```
set.seed(123)
## Training of random forest
rf = randomForest(PM10 ~.,data=training)
rf
```

```
##
## Call:
##  randomForest(formula = PM10 ~ ., data = training)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
##           Mean of squared residuals: 31.45672
##                     % Var explained: 57.69
```

MODEL 5 : Training of the model using MCO

```
set.seed(123)
#We perform a regression
lr = lm(PM10 ~.,data=training)
summary(lr)
```

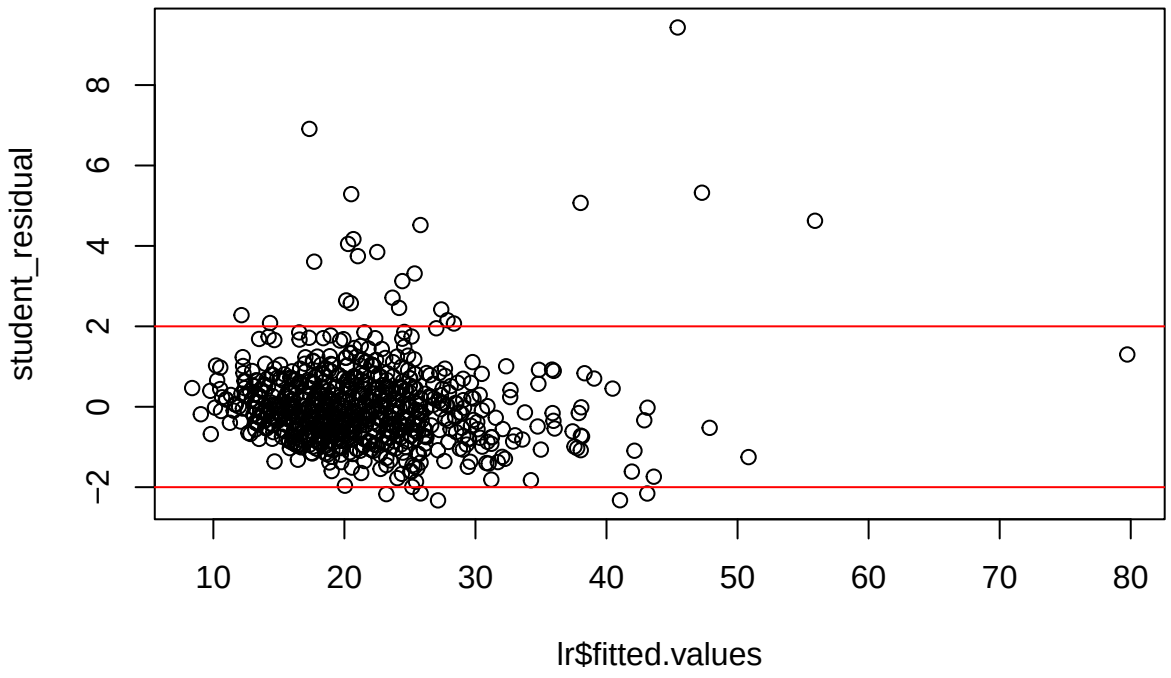
```
##
## Call:
## lm(formula = PM10 ~ ., data = training)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -13.143  -3.413  -0.595   2.582  49.569
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -54.985624  29.323743  -1.875  0.06114 .
## NO            0.190395   0.016683  11.413  < 2e-16 ***
## NO2           0.104999   0.025003   4.199 2.97e-05 ***
## SO2           0.291225   0.044860   6.492 1.48e-10 ***
## T.min         0.635420   0.210966   3.012  0.00268 **
## T.max         0.802648   0.261192   3.073  0.00219 **
## T.moy        -1.122538   0.404562  -2.775  0.00565 **
## DV.maxvv     -0.003968   0.002562  -1.549  0.12189
## DV.dom       -0.005314   0.002654  -2.002  0.04557 *
## VV.max       -0.180625   0.230430  -0.784  0.43335
## VV.moy       -0.052712   0.337197  -0.156  0.87582
## PL.som       -0.140676   0.056576  -2.486  0.01310 *
## HR.min        0.090114   0.046529   1.937  0.05313 .
## HR.max       -0.128956   0.084172  -1.532  0.12590
```

```
## HR.moy       -0.160595   0.090873  -1.767  0.07756 .
## PA.moy        0.084640   0.027671   3.059  0.00230 **
## GTrouen       0.092643   0.186744   0.496  0.61996
## GTlehavre     0.849605   0.271149   3.133  0.00179 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.705 on 807 degrees of freedom
## Multiple R-squared:  0.5717, Adjusted R-squared:  0.5627
## F-statistic: 63.37 on 17 and 807 DF,  p-value: < 2.2e-16
```

```
#The only part we can interpret are the adjusted R2 (0.5494) and the coefficient estimate.
#Before analyzing the other part we need to know if the residuals follows N(0,constant)
#If not, we can't use the results of tests, the residual standard error
#, the std of each estimate because they have all been calculated assuming
#the redisudal follows N(0,constant)
```

```
set.seed(123)
#The residuals are not identically distributed, so we need
#to perform some transformation so that they become idd.
#Computation of studendized residual which distribution is a student(N-3)
student_residual = rstudent(lr)

#Detect outliers
plot(lr$fitted.values,student_residual)
abline(h=2,col='red')
abline(h=-2,col='red')
```



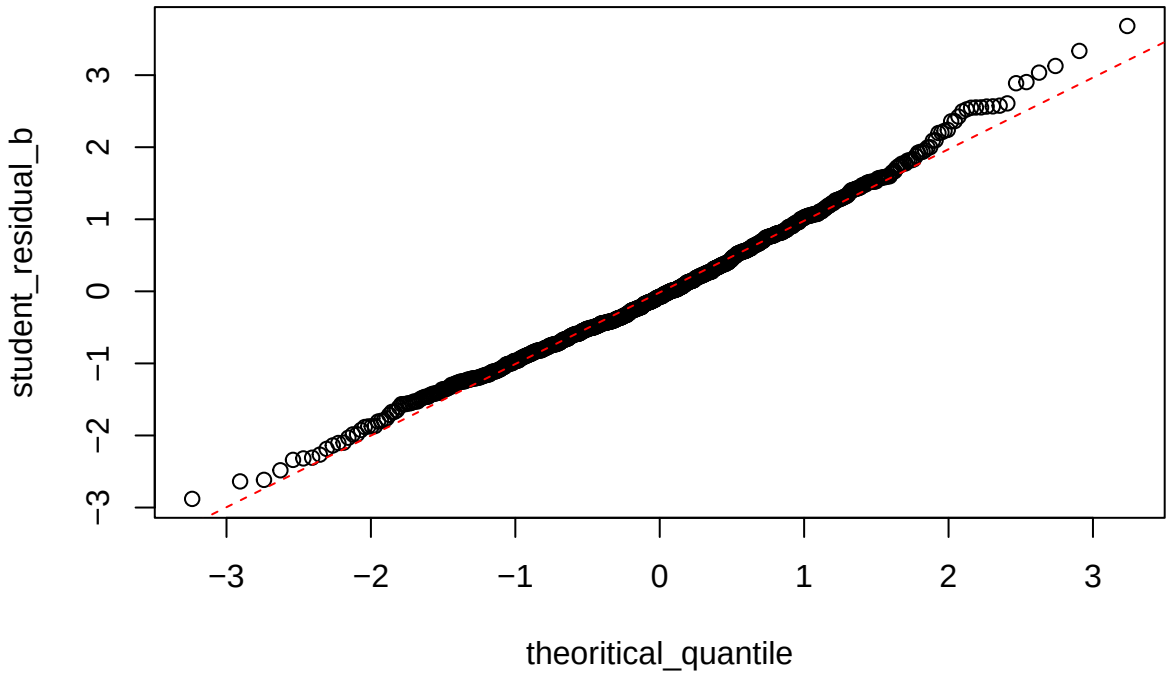

```
#Get the index of the observation that are consider as outliers
a=which(student_residual>2)
b=which(student_residual<(-2))
```

```
#We are going to delete all the outliers
set.seed(123)
```

```r
training_v2=training[-a,]
training_v2=training_v2[-b,]

lr_b=lm(PM10~.,data=training_v2)
student_residual_b = rstudent(lr_b) # studentized residuals

#we draw the qqplot for the residuals--> #We can see
#that the residuals follow a normal distribution with mean 0 and constant standard deviation.
theoritical_quantile = qt(ppoints(797), df = 794)
qqplot(theoritical_quantile, student_residual_b) #student_residual are empirical quantile
qqline(student_residual_b,distribution=function(p){qt(p,df=794)},col='red',lty=2)
```




```r
#We perform a Kolmogorov-Smirnov test to have an analytic proof
#We obtain a big p-value, so we can not reject the null hypothesis which is :
#we can not reject the fact that the residuals follows a student law with 794 degree of liberty
ks.test(student_residual_b,'pt',794) # --> p-value :0.1792
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  student_residual_b
## D = 0.038896, p-value = 0.1792
## alternative hypothesis: two-sided
```

```r
summary(lr_b)
```

```
##
## Call:
## lm(formula = PM10 ~ ., data = training_v2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.5434  -2.7489  -0.2974   2.6390  13.2889
```

```
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -63.008300  21.163425  -2.977  0.00300 **
## NO            0.172528   0.012394  13.920  < 2e-16 ***
## NO2           0.124439   0.018308   6.797 2.13e-11 ***
## SO2           0.154496   0.033312   4.638 4.13e-06 ***
## T.min         0.551225   0.151770   3.632  0.00030 ***
## T.max         0.620790   0.188938   3.286  0.00106 **
## T.moy        -0.865693   0.291882  -2.966  0.00311 **
## DV.maxvv     -0.004749   0.001854  -2.561  0.01062 *
## DV.dom       -0.002246   0.001917  -1.171  0.24187
## VV.max        0.009744   0.166392   0.059  0.95332
## VV.moy       -0.184091   0.244230  -0.754  0.45122
## PL.som       -0.115551   0.042336  -2.729  0.00649 **
## HR.min        0.060571   0.033482   1.809  0.07083 .
## HR.max       -0.052049   0.060506  -0.860  0.38992
## HR.moy       -0.167806   0.065255  -2.572  0.01031 *
## PA.moy        0.087139   0.019952   4.368 1.43e-05 ***
## GTrouen       0.283780   0.134753   2.106  0.03553 *
## GTlehavre     0.548995   0.198409   2.767  0.00579 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.06 on 779 degrees of freedom
## Multiple R-squared:  0.6674, Adjusted R-squared:  0.6602
## F-statistic: 91.97 on 17 and 779 DF,  p-value: < 2.2e-16
```

```r
set.seed(123)
#Variables selection using glmnet
Las=glmnet(training_v2[,-18],training_v2[,18])

#we create a collection of models and we are going to select the best one using cross validation
sLas=cv.glmnet(as.matrix(training_v2[,-18]),training_v2[,18])
#We select lambda that is going to give us the best model
lambdaf=sLas$lambda.1se

#We construct the model with the value lambdaf : the variable selected are those wihtout a dot in s0.
Lasf=glmnet(training_v2[,-18],training_v2[,18],lambda=lambdaf)
print(Lasf$beta)
```

```
## 17 x 1 sparse Matrix of class "dgCMatrix"
##                       s0
## NO         0.1650879827
## NO2        0.0921898706
## SO2        0.0778602923
## T.min      .
## T.max      0.1780595017
## T.moy      .
## DV.maxvv  -0.0028749500
## DV.dom    -0.0004056753
## VV.max    -0.0542302276
## VV.moy    -0.0609292265
## PL.som    -0.0734824661
## HR.min     .
```

```
## HR.max    -0.0960476362
## HR.moy    -0.0247946122
## PA.moy     0.0686833159
## GTrouen    0.3015535983
## GTlehavre  0.6527149501
```

```
# Subset the training_v2 dataset
new_training <- training_v2[, c("NO", "NO2", "SO2",
                                "T.max", "DV.maxvv", "DV.dom","VV.max",
                                "VV.moy", "PL.som", "HR.max", "HR.moy" ,"PA.moy",
                                "GTrouen", "GTlehavre", "PM10")]
```

```
set.seed(123)
lr_final = lm(PM10 ~.,data=new_training )
```

```
#Results of the final linear regression
summary(lr_final)
```

```
##
## Call:
## lm(formula = PM10 ~ ., data = new_training)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.0561  -2.8158  -0.3703   2.5510  12.5456
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -63.647238  21.161994  -3.008  0.00272 **
## NO            0.174428   0.012431  14.031  < 2e-16 ***
## NO2           0.125450   0.018299   6.855 1.44e-11 ***
## SO2           0.152775   0.033151   4.608 4.74e-06 ***
## T.max         0.298800   0.031084   9.613  < 2e-16 ***
## DV.maxvv     -0.004759   0.001865  -2.551  0.01092 *
## DV.dom       -0.002600   0.001918  -1.355  0.17572
## VV.max       -0.010017   0.164943  -0.061  0.95159
## VV.moy       -0.148399   0.239353  -0.620  0.53544
## PL.som       -0.108868   0.042599  -2.556  0.01079 *
## HR.max       -0.146946   0.051032  -2.880  0.00409 **
## HR.moy       -0.018551   0.028462  -0.652  0.51474
## PA.moy        0.087749   0.019925   4.404 1.21e-05 ***
## GTrouen       0.184752   0.128031   1.443  0.14941
## GTlehavre     0.548318   0.198924   2.756  0.00598 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.089 on 782 degrees of freedom
## Multiple R-squared:  0.6613, Adjusted R-squared:  0.6552
## F-statistic:   109 on 14 and 782 DF,  p-value: < 2.2e-16
```

EVALUATION OF MODELS

```
  eval=function(test,model_interp,model_predict,rf,cart_model,lr_final) # function to evaluate models
{

  # Prediction with each model
```

```
  pred_interp = predict(model_interp,newdata=test)
  pred_prediction = predict(model_predict,newdata=test) #the same with the second model
  pred_rf = predict(rf,newdata=test)
  pred_cart = predict(cart_model,newdata=test)
  pred_lr_final = predict(lr_final,newdata=test)

  # Computing errors fpr each model
  testerr_interp=1/(nrow(test))*sum((test$PM10-pred_interp)^2)
  testerr_prediction=1/(nrow(test))*sum((test$PM10-pred_prediction)^2)
  testerr_rf=1/(nrow(test))*sum((test$PM10-pred_rf)^2)
  testerr_cart=1/(nrow(test))*sum((test$PM10-pred_cart)^2)
  testerr_lr_final=1/(nrow(test))*sum((test$PM10-pred_lr_final)^2)

  # Storing results in a list
  eval=list(testerr_interp, testerr_prediction, testerr_rf, testerr_cart,testerr_lr_final)

}
```

```
set.seed(123)
results = eval(test, model_interp,model_predict,rf,cart_model,lr_final)
```

```
results <- matrix(results, nrow = 5, ncol = 1)
row_names <- c("errors_interp","Erros_pred","Errors_rf","Errors_cart_model","errors_lr_final")
rownames(results) <- row_names
colnames(results) <- "model_errors"
print(results)
```

```
##                   model_errors
## errors_interp     39.70477
## Erros_pred        43.79505
## Errors_rf         26.06769
## Errors_cart_model 43.79505
## errors_lr_final   29.26239
```

```
"Random forest (Errors_rf) achieves the lowest error (26.06769 ) ,
indicating superior predictive performance compared to others models.
However, interpreting random forest results can be challenging
due to the complexity of the model. Random forests operate by constructing
multiple decision trees during training and outputting the mean prediction
of the individual trees for regression.
So, despite its superior performance, the Linear regression model
(errors_lr_final) might be preferred in practice due to their ease
of interpretation. We also notice that model_cart and the model
with the variables selected after the prediction step retained
the same variable for the construction of the best tree and that's
why they give the same errors values.
"
```

```
## [1] "Random forest (Errors_rf) achieves the lowest error (26.06769 ) , \nindicating superior predicti
```