BE3 - apprentissage statitique

Tulio NAVARRO TUTUI, Filipe PENNA CERAVOLO SOARES

07 December, 2022

Exercice 1 : Bitume - approches PLS, PCR , Lasso

1. Lire les données «bitume.train.txt» et «bitume.test.txt».

```
set.seed(23)
library(DiceEval)
```

```
## Loading required package: DiceKriging
```

```
library(car)
```

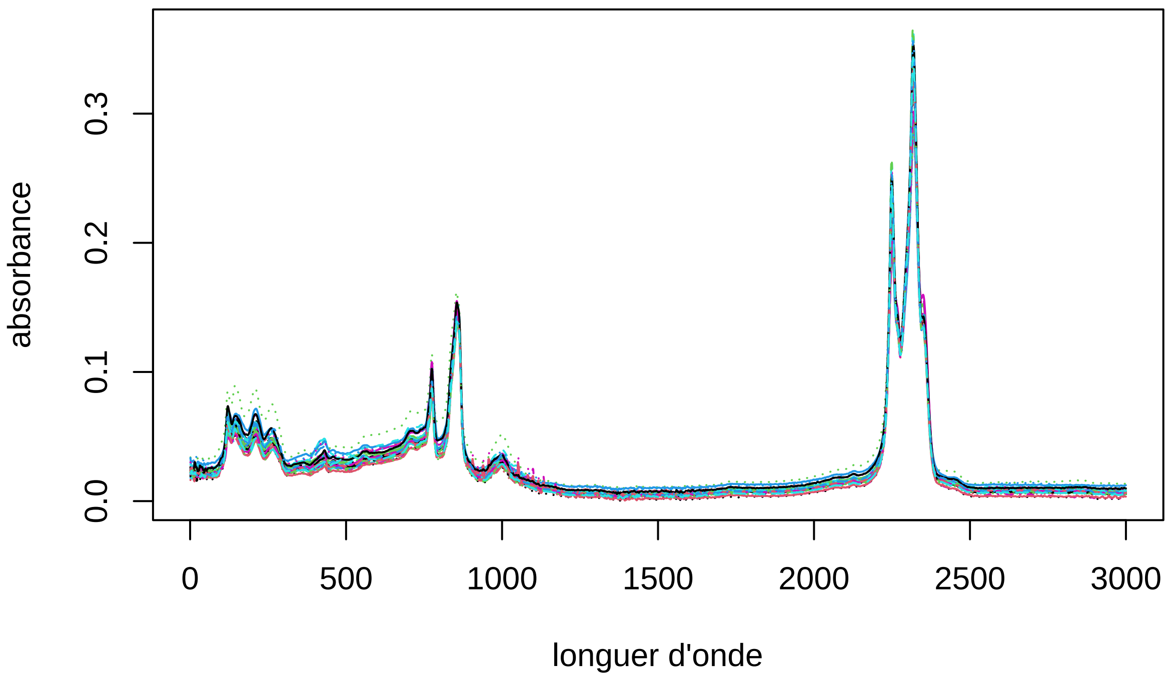
```
## Loading required package: carData
```

```
library(MASS)

bitume_train = read.table("bitume.train.txt", header = T)
bitume_test  = read.table("bitume.test.txt", header = T)

p = ncol(bitume_train)
# head(bitume_train)

summary(bitume_train[, 1])
```

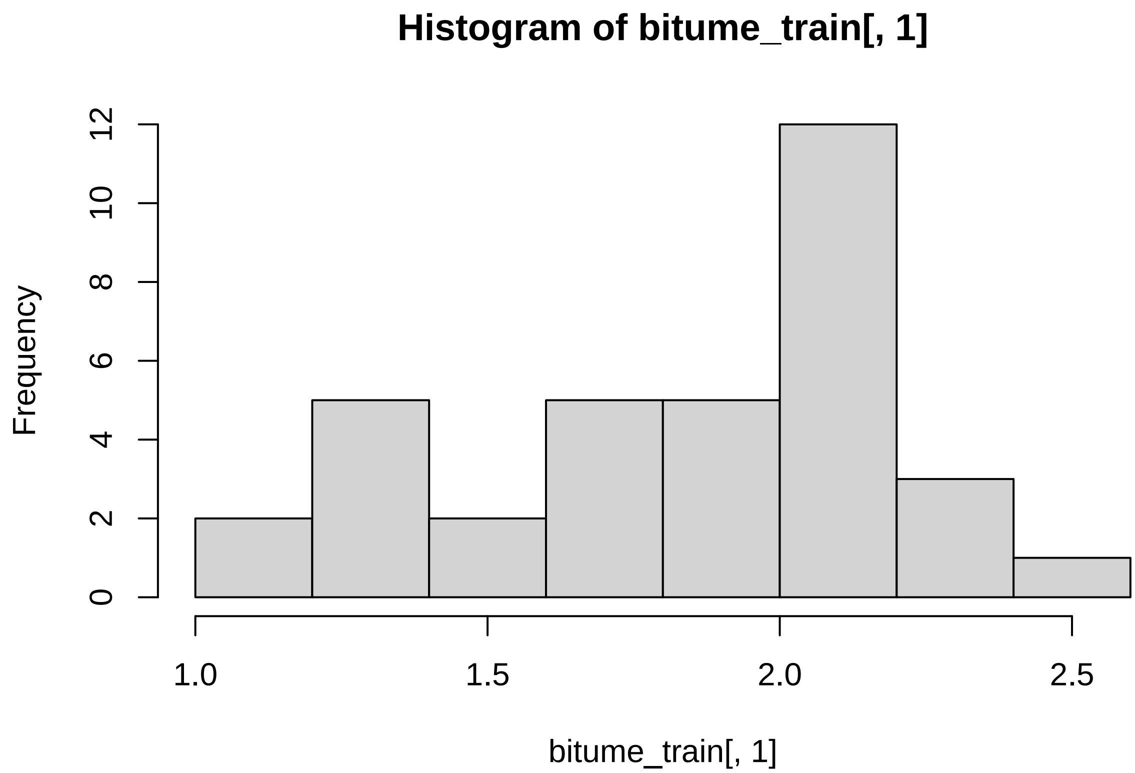
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.079   1.620   1.908   1.842   2.132   2.431
```

2. Visualiser sur le même graphes avec des couleurs différentes les 35 spectres de l'échantillon d'apprentissage. Tracer l'histogramme des pénétrabilités correspondantes.

```
matplot(t(bitume_train[, -1]), type="l", xlab = "longuer d'onde", ylab = "absorbance")
```

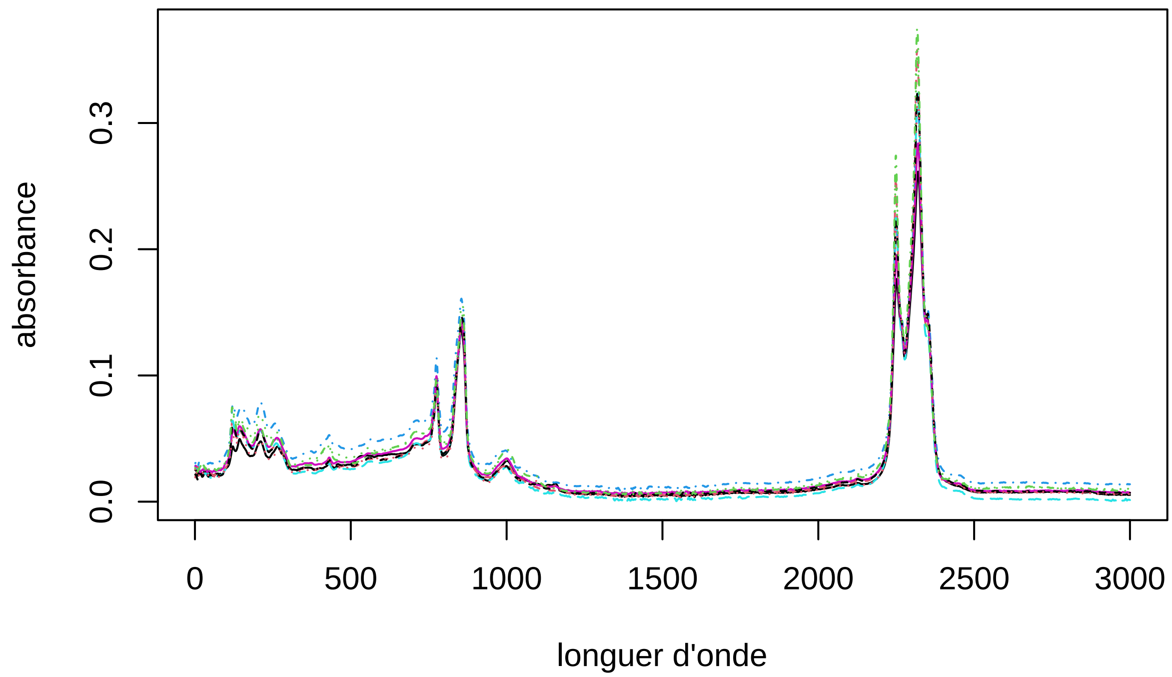



```
hist(bitume_train[, 1])
```

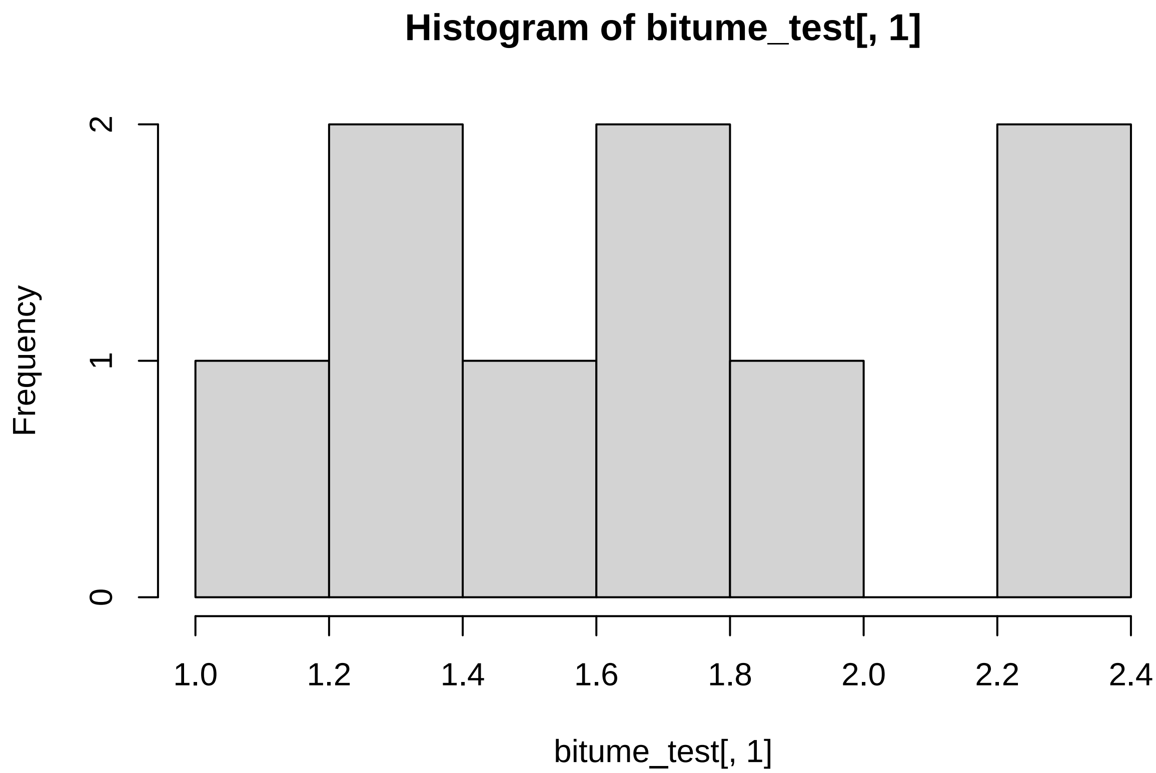


3. Faire de même avec l'échantillon test.

```
matplot(t(bitume_test[, -1]), type="l", xlab = "longuer d'onde", ylab = "absorbance")
```




```
hist(bitume_test[, 1])
```



On observe une allure similaire entre la longuer d'onde de l'échantillon de test et celui d'apprentissage. Par contre, l'histogramme révele un profil different de pénétrabilités.

BONUS Faire une classification pour identifier des typologies différentes de spectres (routines kmeans ou hclust). Tracer les pénétrabilités en fonction du numéro de classe. Y-a-t-il un lien ?

```
bitume_train_normal <- scale(bitume_train)
# resKM <- kmeans(data, centers = .., nstart = 20)
```

4. Ajuster un modèle PCR et PLS (fonction pcr et plsr du package pls) puis un modèle lasso. Expliquer les différentes étapes de la sélection des hyper-paramètres des méthodes. Interpréter les modèles obtenus. Pour les modèles PCR et PLS on visualisera notamment les premières fonctions propres. Comparer la qualité prédictive sur l'ensemble test des modèles ainsi "calibrés".

```
library(pls)
```

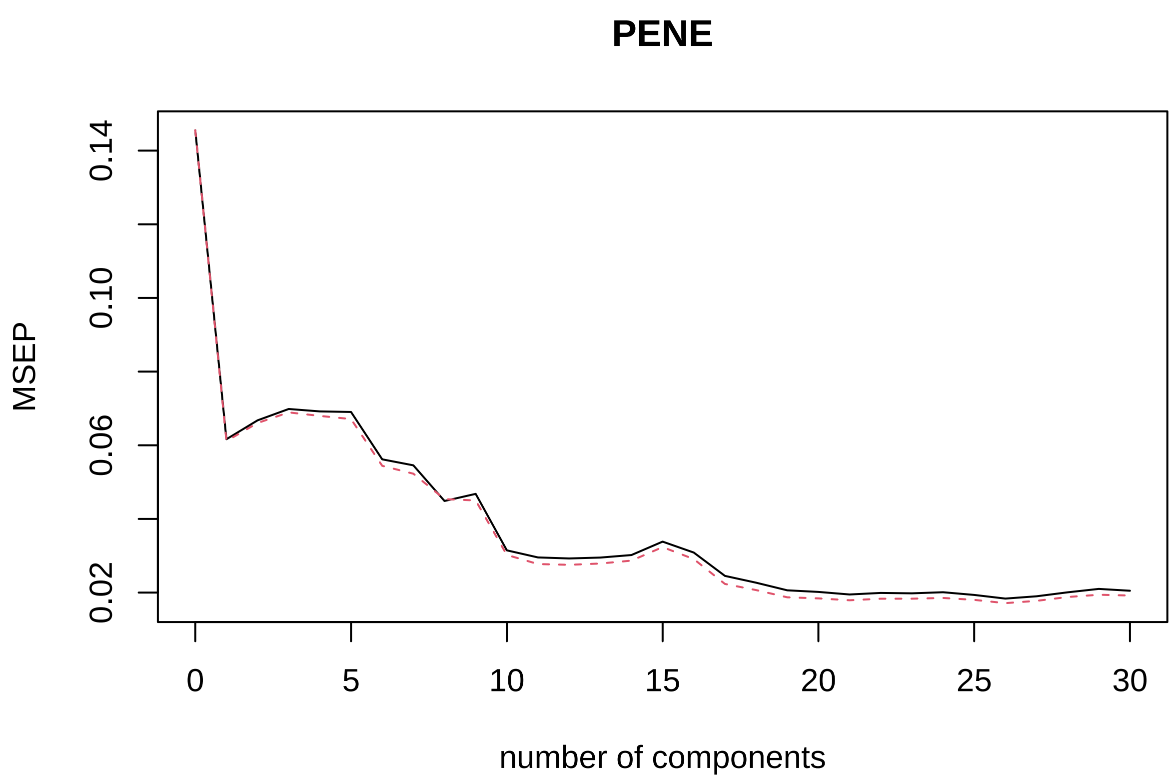
PCR

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:DiceEval':
##
##     R2

## The following object is masked from 'package:stats':
##
##     loadings
```

```
#PCR
bitume_train.pcr <- pcr(PENE ~., 30, data = bitume_train)
```

Faire le choix du nombre de composantes par validation croisée en utilisant la routine crossval.

```
bitume_cv <- crossval(bitume_train.pcr, segments = 10)
plot(MSEP(bitume_cv))
```



Aprés 21 on a un diminuition négligeable d'erreur, donc on choisi ce valeur pour n'avoir un modele trés complexe

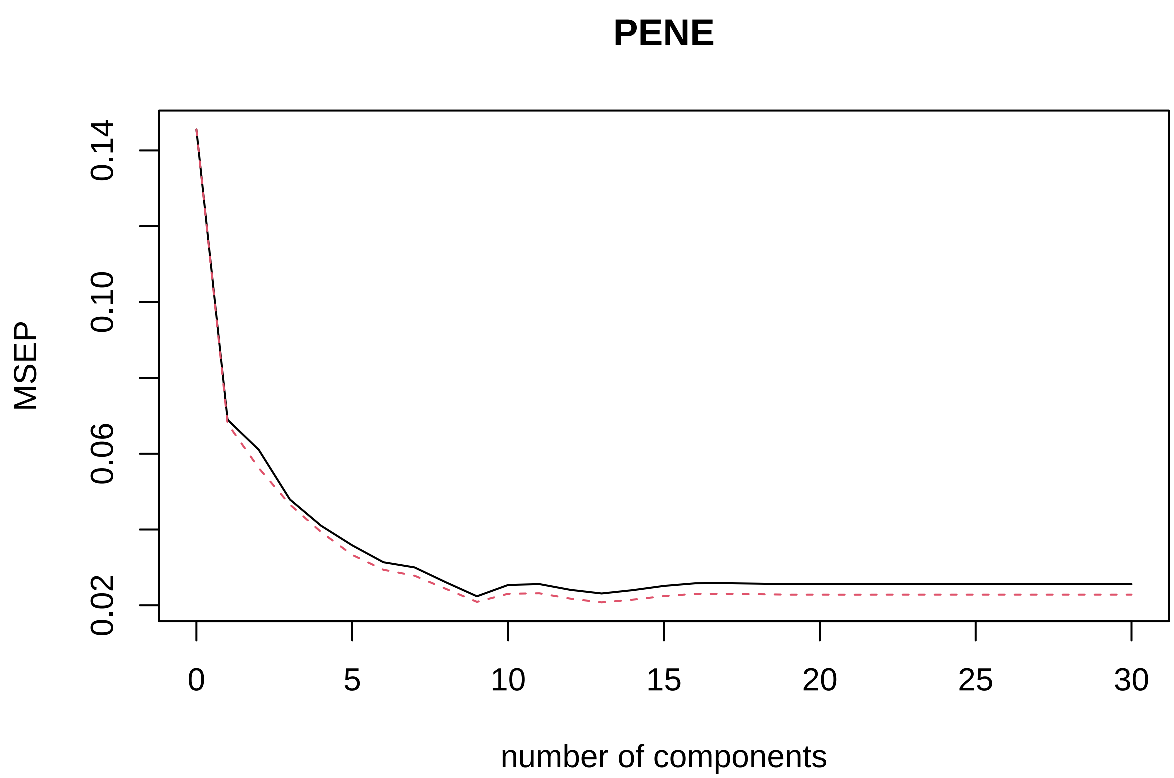
```
nbcomp_pcr = 21
Y_PCR = predict(bitume_train.pcr, newdata = bitume_test, ncomp = nbcomp_pcr, type = "response") # nolin
```

```r
RMSE_PCR = RMSE(bitume_test[, 1], Y_PCR)
RMSE_PCR
```

```
## [1] 0.1887259
```

```r
#tracer des premières fonctions propres
# par(mfrow = c(1,1))
# plot(1:(p-1), bitume_train.pcr$loadings[1:(p_train-1), 1], type ="l",col = 1)
# points(1:(p-1), bitume_train.pcr$loadings[1:(p_train-1), 1], col = 1)
# for (i in 2:4)
# {
#   points(1:(p_train-1),bitume_train.pcr$loadings[1:(p_train-1),i],col = i)
# }
```

```r


bitume_train.pls <- plsr(PENE ~., 30, data = bitume_train)
bitume_train.pls.cv <- crossval(bitume_train.pls, segments = 10)
plot(MSEP(bitume_train.pls.cv))
```



PLS

Dans ce cas, c'est claire que le nombre optimale de composants est 9, un fois que l'erreur est le plus petite et la complexité n'est pas si grande.

```r
nbcomp_pls = 9
Y_PLS = predict(bitume_train.pls, newdata = bitume_test, ncomp = nbcomp_pls, type = "response")

RMSE_PLS = RMSE(bitume_test[, 1], Y_PLS)
RMSE_PLS
```

```
## [1] 0.1965467
```

```r
#-------------------------------------------------------------
# lasso
#-------------------------------------------------------------

library(lars)
```

Lasso

```
## Loaded lars 1.3
```

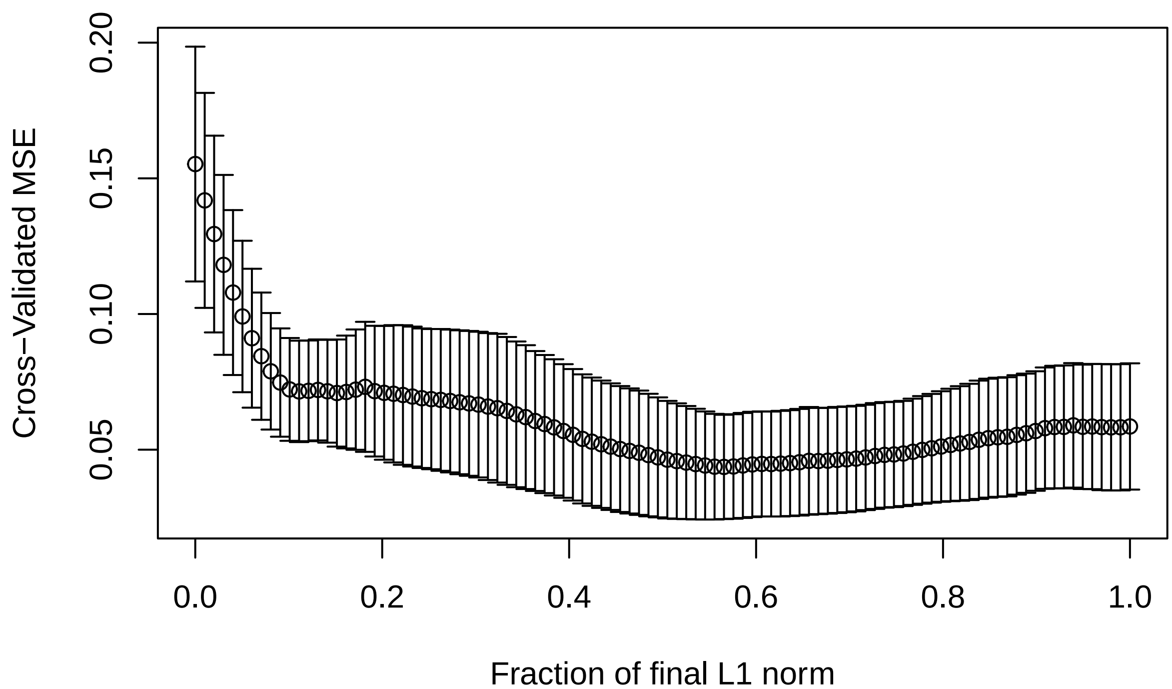
```r
y = as.matrix(bitume_train[, 1])
x = as.matrix(bitume_train[, -1])
# x_extend = x
#
# # makes the binary combinations of all columns
# for (i in 1:(p - 2)) {
#   for (j in 2:(p - 1)) {
#     x_extend = cbind(x_extend, x[, i] * x[, j])
#   }
# }

mod_lasso = lars(x, y, type = "lasso")
```

```
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
```

```r
CV = cv.lars(x, y, K = 10, index=seq(from = 0, to = 1, length = 100),
             trace = FALSE, plot.it = TRUE, se = TRUE, type = "lasso", mode = "fraction")
```

```
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
```

```
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
## There are more than 500 variables and n<m;
## You may wish to restart and set use.Gram=FALSE
```




```
which.min(CV$cv.error)
```

```
## [1] 13
```

```
value = 0.5  # choisir valeur avec plus petite erreur

# newx = as.matrix(bitume_test[, -1])
# for (i in 1:(p - 2))
# {
#   for (j in 2:(p - 1))
#   {
#     newx = cbind(newx, newx[, i] * newx[, j])
#   }
# }
```

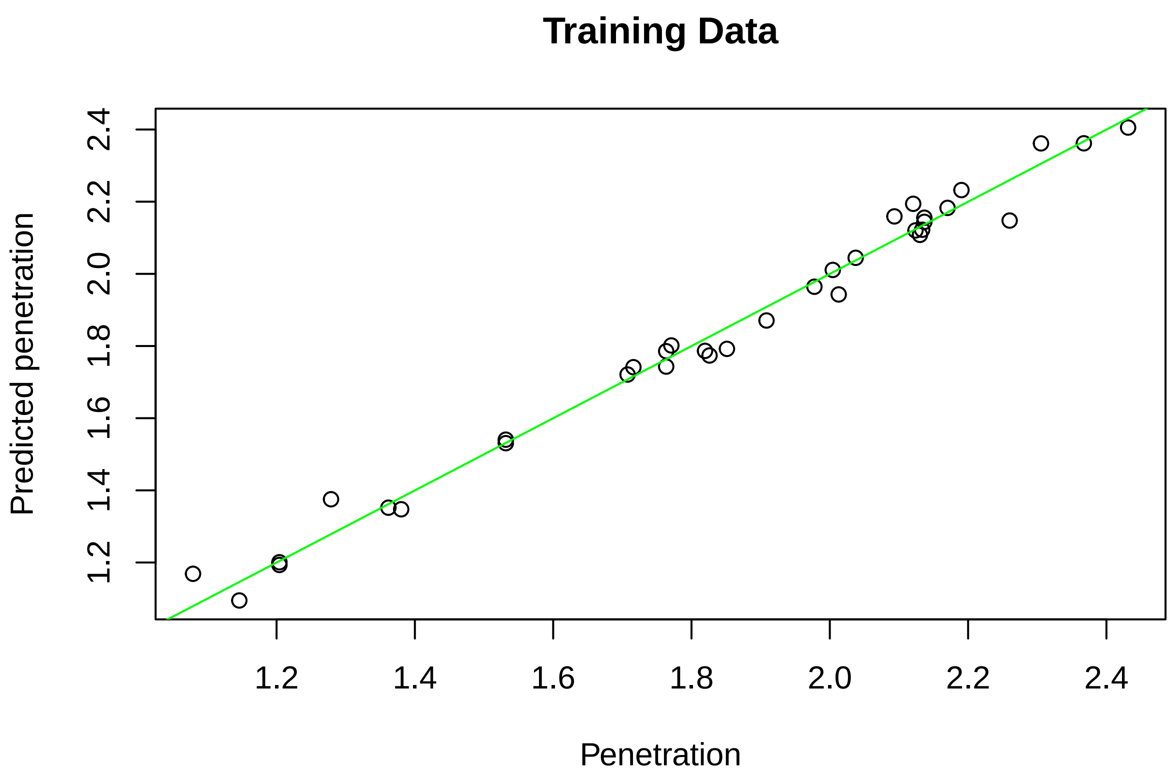
```
fits <- predict.lars(mod_lasso, bitume_test[, -1], s = value, mode = "frac")

RMSE_lasso = RMSE(bitume_test[, 1], fits$fit)
RMSE_lasso
```

```
## [1] 0.2205037
```

5. Pour les 3 méthodes, représenter les pénétrabilités prédites en fonction des pénétrabilités observées sur les 2 ensembles : apprentissage et test.

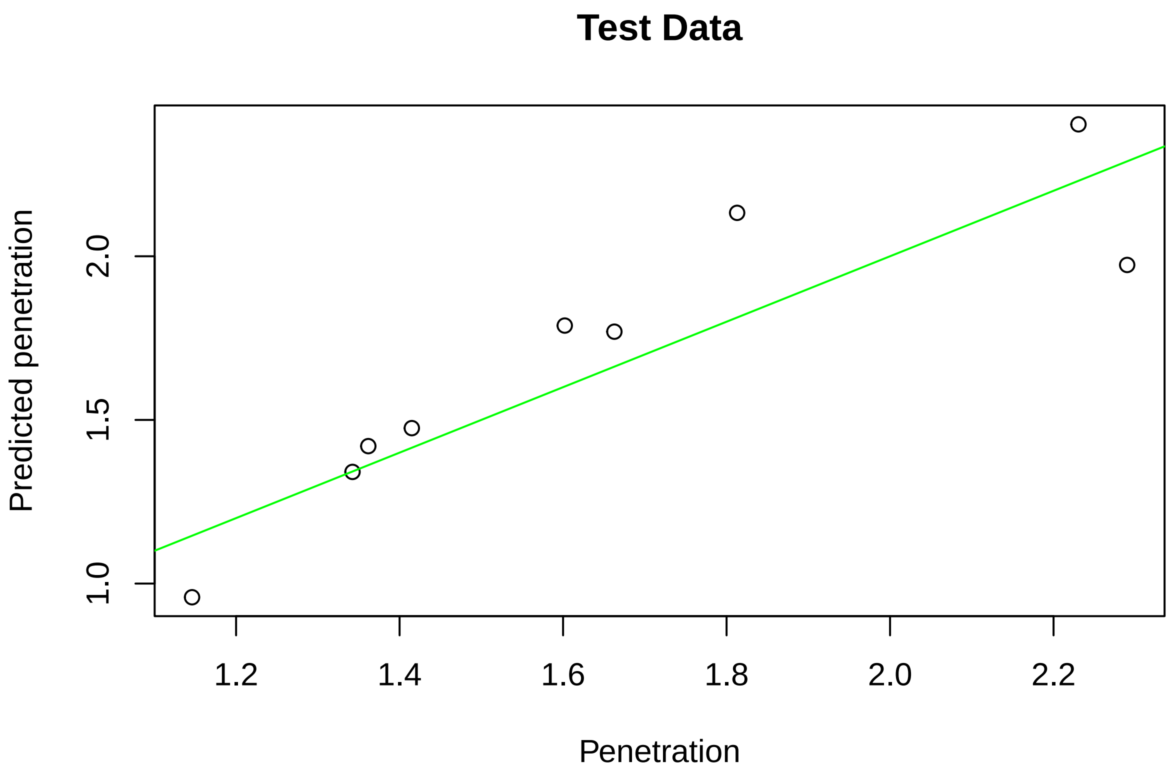
PCR

```
train_predictions = predict(bitume_train.pcr, newdata = bitume_train, ncomp = nbcomp_pcr, type = "respon
plot(bitume_train$PENE, train_predictions, xlab="Penetration", ylab="Predicted penetration", main="Train
abline(a = 0, b = 1, col = "green")
```



Train

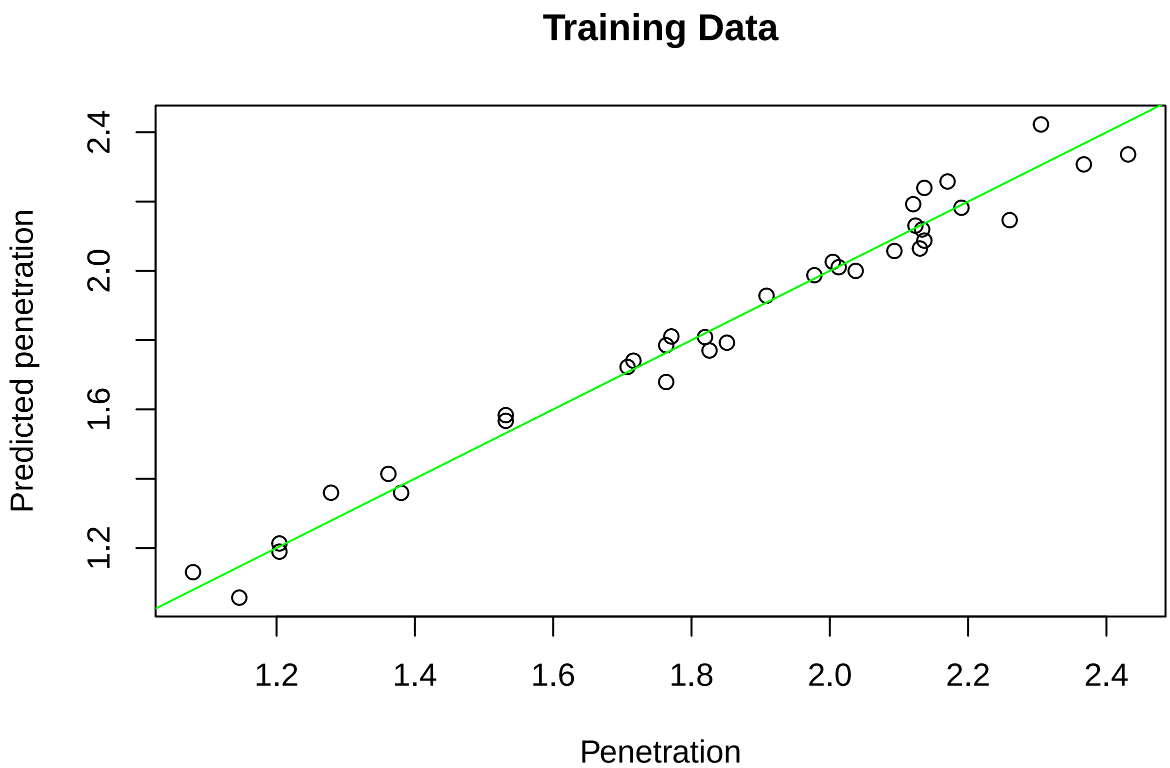
```
plot(bitume_test$PENE, Y_PCR, xlab="Penetration", ylab="Predicted penetration", main="Test Data")
abline(a = 0, b = 1, col = "green")
```



Test

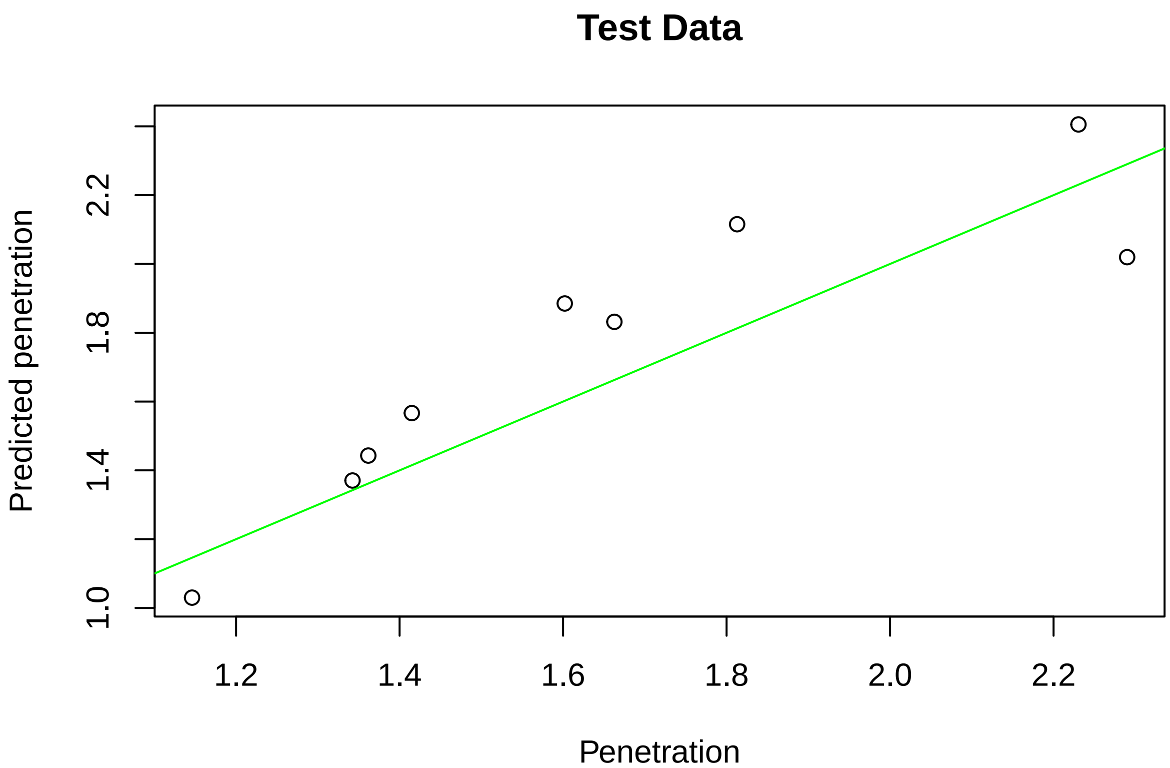
PLS

```
train_predictions = predict(bitume_train.pls, newdata = bitume_train, ncomp = nbcomp_pls, type = "respon
plot(bitume_train$PENE, train_predictions, xlab="Penetration", ylab="Predicted penetration", main="Train
abline(a = 0, b = 1, col = "green")
```



Train

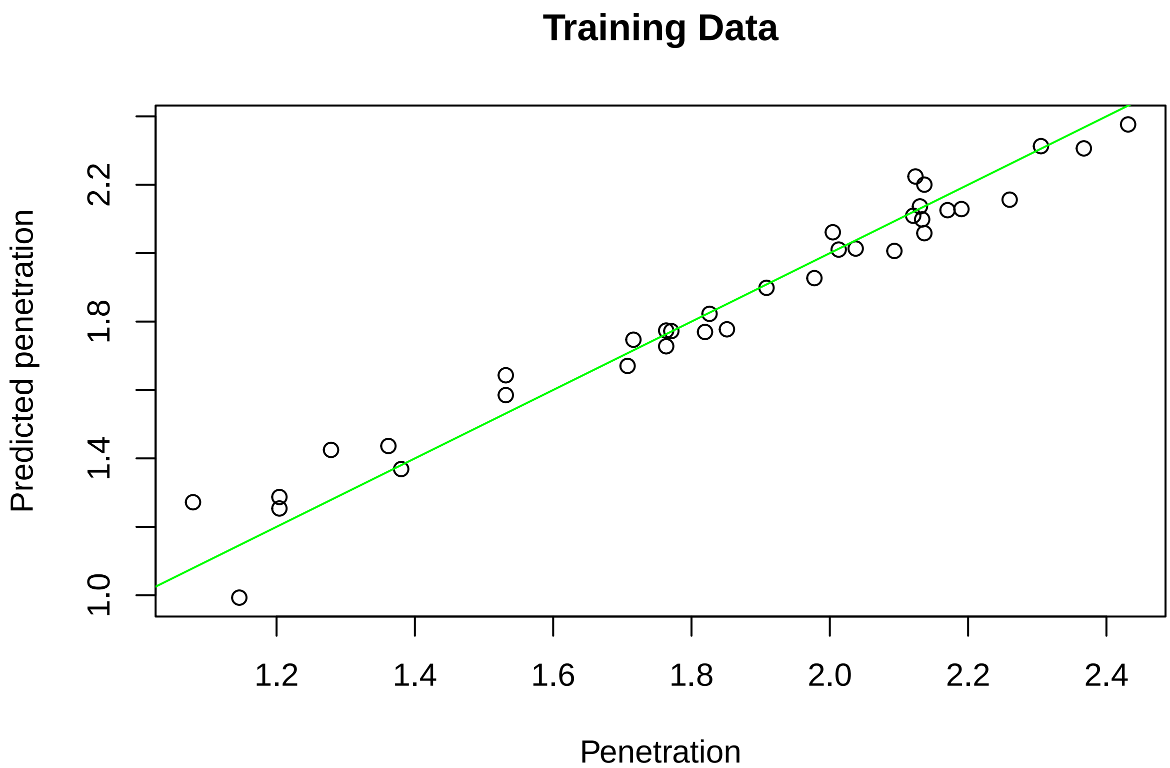
```
plot(bitume_test$PENE, Y_PLS, xlab="Penetration", ylab="Predicted penetration", main="Test Data")
abline(a = 0, b = 1, col = "green")
```



Test

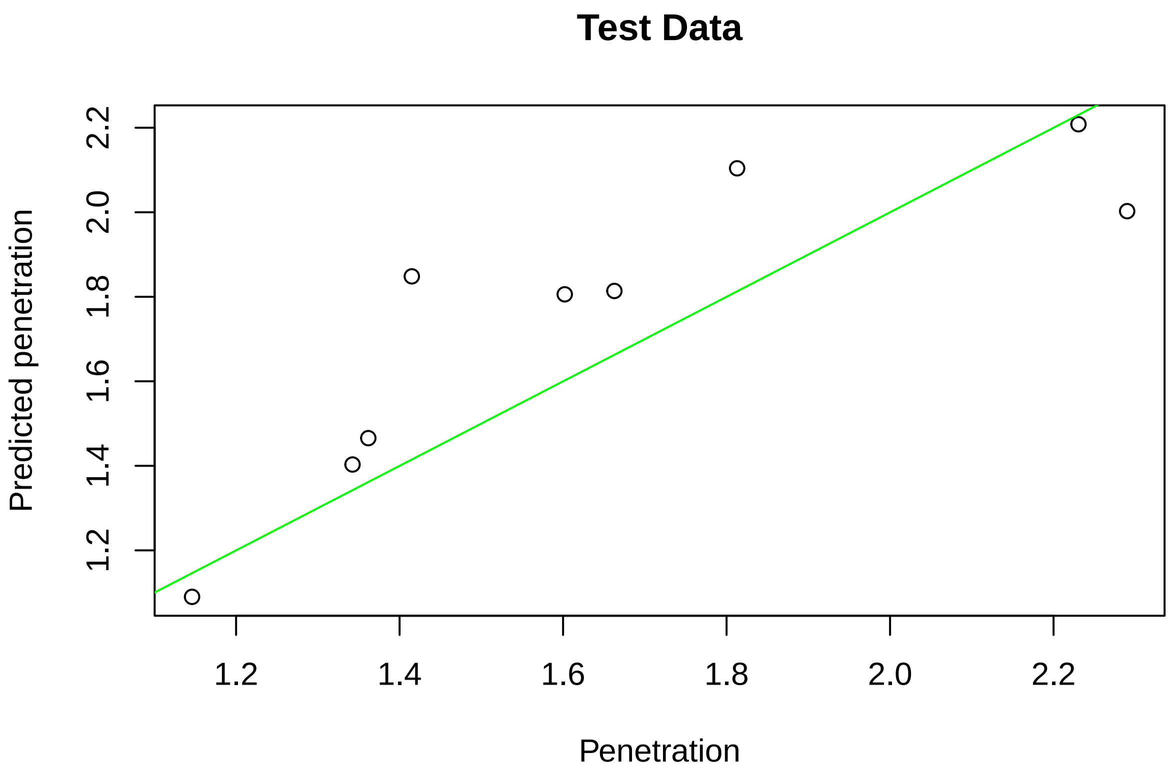
Lasso

```
train_fits <- predict.lars(mod_lasso, bitume_train[, -1], s = value, mode = "frac")
plot(bitume_train$PENE, train_fits$fit, xlab="Penetration", ylab="Predicted penetration", main="Trainin
abline(a = 0, b = 1, col = "green")
```



Train

```
plot(bitume_test$PENE, fits$fit, xlab="Penetration", ylab="Predicted penetration", main="Test Data")
abline(a = 0, b = 1, col = "green")
```



Test

Conclusion En regardant les courbes de test, c'est évident que tous les modèles sont en train de surestimer les valeurs de pénétration dans la base de test. La raison pour cette conclusion est le fait qu'on voit que la plupart des points sont au-dessus de la courbe verte, que représente que la prédiction est parfaite.

6. Pourrait-on essayer d'ajuster un modèle linéaire ?

```
bitume_train.lm <- lm(PENE ~., data = bitume_train)
Y_pred_lm = predict(bitume_train.lm, bitume_test[, -1])
```

```
## Warning in predict.lm(bitume_train.lm, bitume_test[, -1]): prediction from a
## rank-deficient fit may be misleading
```

```
RMSE_lasso = RMSE(bitume_test[, 1], Y_pred_lm)
RMSE_lasso
```

```
## [1] 6.671221
```

```
length(na.omit(bitume_train.lm$coefficients))
```

```
## [1] 35
```

C'est possible d'ajuster un modèle linéaire, mais ce modèle va avoir une performance baisse. La cause de ce fait est la différence de quantité entre colonnes et lignes, de manière que ce n'est pas possible de déterminer tous les coefficients de la régression, parce que n_colonnes > n_lignes.

Exercice 2 : Carseats

```r
carseat = read.table("Carseats.txt", header = T)
summary(carseat)
```

```
##      Sales          CompPrice       Income        Advertising    
##  Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000  
##  1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000  
##  Median : 7.490   Median :125   Median : 69.00   Median : 5.000  
##  Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635  
##  3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000  
##  Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000  
##    Population        Price        ShelveLoc              Age       
##  Min.   : 10.0   Min.   : 24.0   Length:400         Min.   :25.00  
##  1st Qu.:139.0   1st Qu.:100.0   Class :character   1st Qu.:39.75  
##  Median :272.0   Median :117.0   Mode  :character   Median :54.50  
##  Mean   :264.8   Mean   :115.8                      Mean   :53.32  
##  3rd Qu.:398.5   3rd Qu.:131.0                      3rd Qu.:66.00  
##  Max.   :509.0   Max.   :191.0                      Max.   :80.00  
##    Education       Urban                US           
##  Min.   :10.0   Length:400         Length:400        
##  1st Qu.:12.0   Class :character   Class :character  
##  Median :14.0   Mode  :character   Mode  :character  
##  Mean   :13.9                                        
##  3rd Qu.:16.0                                        
##  Max.   :18.0                                        
```

```r
dim(carseat)
```

```
## [1] 400  11
```

```r
p = ncol(carseat)
carseat$ShelveLoc = as.factor(carseat$ShelveLoc)
carseat$Urban = as.factor(carseat$Urban)
carseat$US = as.factor(carseat$US)
summary(carseat)
```

```
##      Sales          CompPrice       Income        Advertising    
##  Min.   : 0.000   Min.   : 77   Min.   : 21.00   Min.   : 0.000  
##  1st Qu.: 5.390   1st Qu.:115   1st Qu.: 42.75   1st Qu.: 0.000  
##  Median : 7.490   Median :125   Median : 69.00   Median : 5.000  
##  Mean   : 7.496   Mean   :125   Mean   : 68.66   Mean   : 6.635  
##  3rd Qu.: 9.320   3rd Qu.:135   3rd Qu.: 91.00   3rd Qu.:12.000  
##  Max.   :16.270   Max.   :175   Max.   :120.00   Max.   :29.000  
##    Population        Price        ShelveLoc        Age          Education   
##  Min.   : 10.0   Min.   : 24.0   Bad   : 96   Min.   :25.00   Min.   :10.0  
##  1st Qu.:139.0   1st Qu.:100.0   Good  : 85   1st Qu.:39.75   1st Qu.:12.0  
##  Median :272.0   Median :117.0   Medium:219   Median :54.50   Median :14.0  
##  Mean   :264.8   Mean   :115.8                Mean   :53.32   Mean   :13.9  
##  3rd Qu.:398.5   3rd Qu.:131.0                3rd Qu.:66.00   3rd Qu.:16.0  
##  Max.   :509.0   Max.   :191.0                Max.   :80.00   Max.   :18.0  
```

```
##  Urban       US
##  No :118   No :142
##  Yes:282   Yes:258
##
##
##
##
```

1. Séparer les données en un ensemble d'apprentissage (70%) et un ensemble de test (30%).

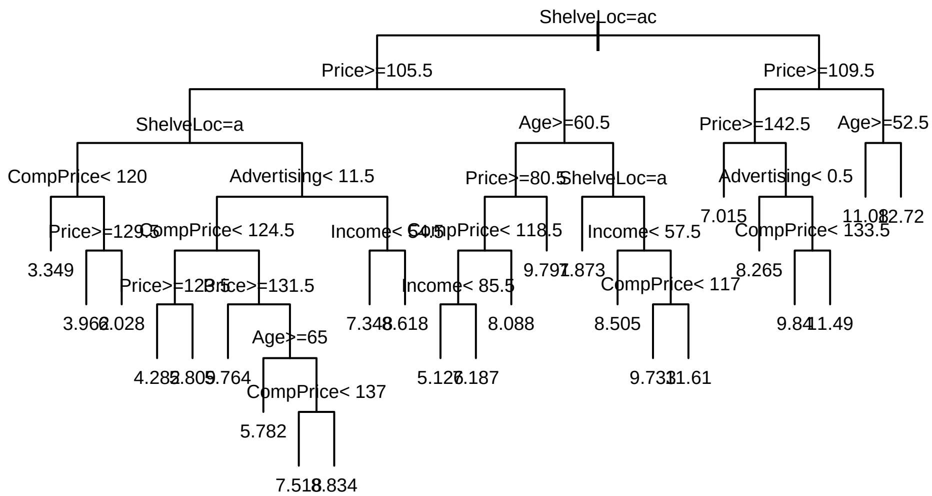
```
set.seed(23)
apprentissage_fraction = as.integer(nrow(carseat) * 0.7)
u = sample(1:nrow(carseat), apprentissage_fraction)
carseat.train = carseat[u,]
carseat.test = carseat[-u,]
```

2. Mettre en place un modèle CART, RF, bagging et boosting. Expliquer précisément les différentes étapes de la mise en oeuvre. Donner les avantages et les inconvénients de chacun de ces modèles. Illustrer cela sur les données à disposition.

CART

Pour faire une modèle CART, il faut choisir un minimum cp (paramètre de complexité). Le modèle crée a chaque itération un nouvele noeud de décision qui minimisera l'erreur sur la base d'apprentissage. Il arrete la création des nouvelles noeuds de décision lorsque l'erreur est moins important que le cp choisi. Ce modèle considère tout les paramètres et échantillons de la base d'apprentissage et donne à la fin seulement une arbre de régression (ou classification).

```
library(rpart)
cont = rpart.control(cp = 0.0001)  # define minimum cp
mod_tree <- rpart(Sales ~., data = carseat.train, control = cont)
par(mfrow = c(1, 1))
plot(mod_tree, uniform = TRUE, margin = 0.05)
text(mod_tree, cex = 0.6)
```



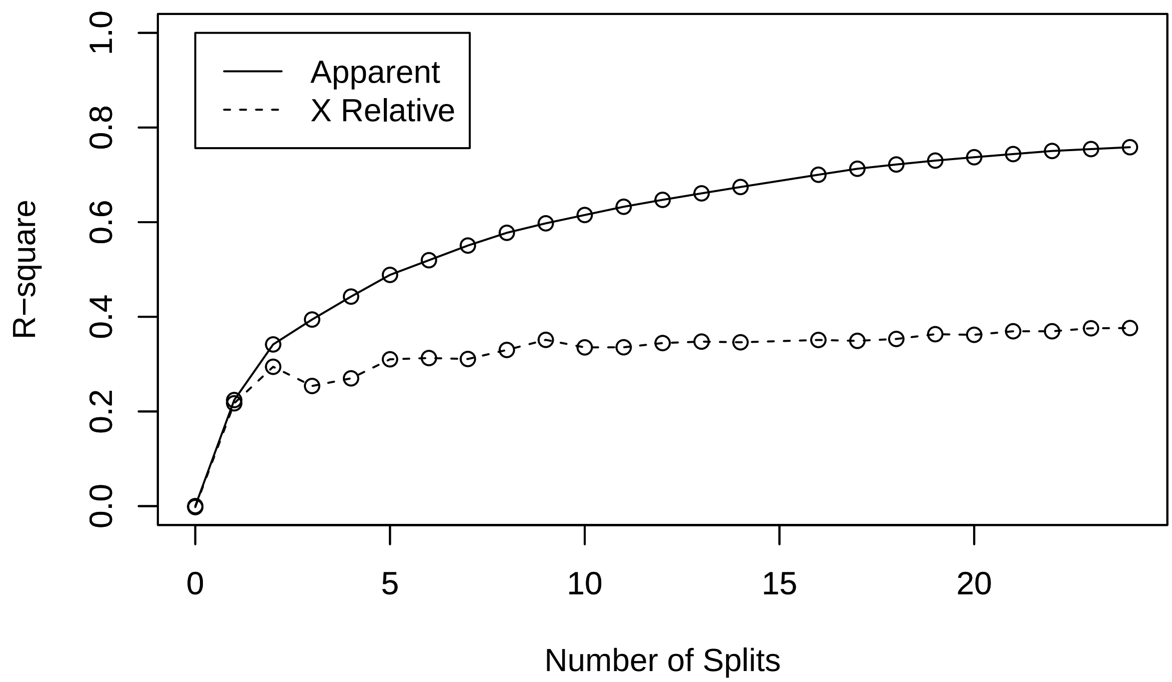

```
mod_tree$cptable
```

```
##              CP nsplit rel error    xerror       xstd
## 1  0.224060850      0 1.0000000 1.0021134 0.07917995
## 2  0.117610132      1 0.7759391 0.7827311 0.06204936
## 3  0.052619954      2 0.6583290 0.7056251 0.05721646
## 4  0.048494078      3 0.6057091 0.7461575 0.05853630
## 5  0.045808395      4 0.5572150 0.7300177 0.05617266
## 6  0.031029715      5 0.5114066 0.6897691 0.05474383
## 7  0.030984877      6 0.4803769 0.6870861 0.05525115
## 8  0.026932469      7 0.4493920 0.6891534 0.05497707
## 9  0.019999420      8 0.4224595 0.6700470 0.05378056
## 10 0.017612488      9 0.4024601 0.6487010 0.05496202
## 11 0.017437922     10 0.3848476 0.6646031 0.05583071
## 12 0.014587415     11 0.3674097 0.6643845 0.05506737
## 13 0.013697434     12 0.3528223 0.6554419 0.05488367
## 14 0.013448122     13 0.3391249 0.6523862 0.05459784
## 15 0.012959115     14 0.3256767 0.6538327 0.05503861
## 16 0.012617739     16 0.2997585 0.6489532 0.05522369
## 17 0.009050297     17 0.2871408 0.6507735 0.05554054
## 18 0.008195227     18 0.2780905 0.6467578 0.05493531
## 19 0.007158189     19 0.2698952 0.6367409 0.05437870
## 20 0.006603075     20 0.2627370 0.6380642 0.05423734
## 21 0.006556505     21 0.2561340 0.6305612 0.05387045
## 22 0.004025029     22 0.2495775 0.6305268 0.05425660
```

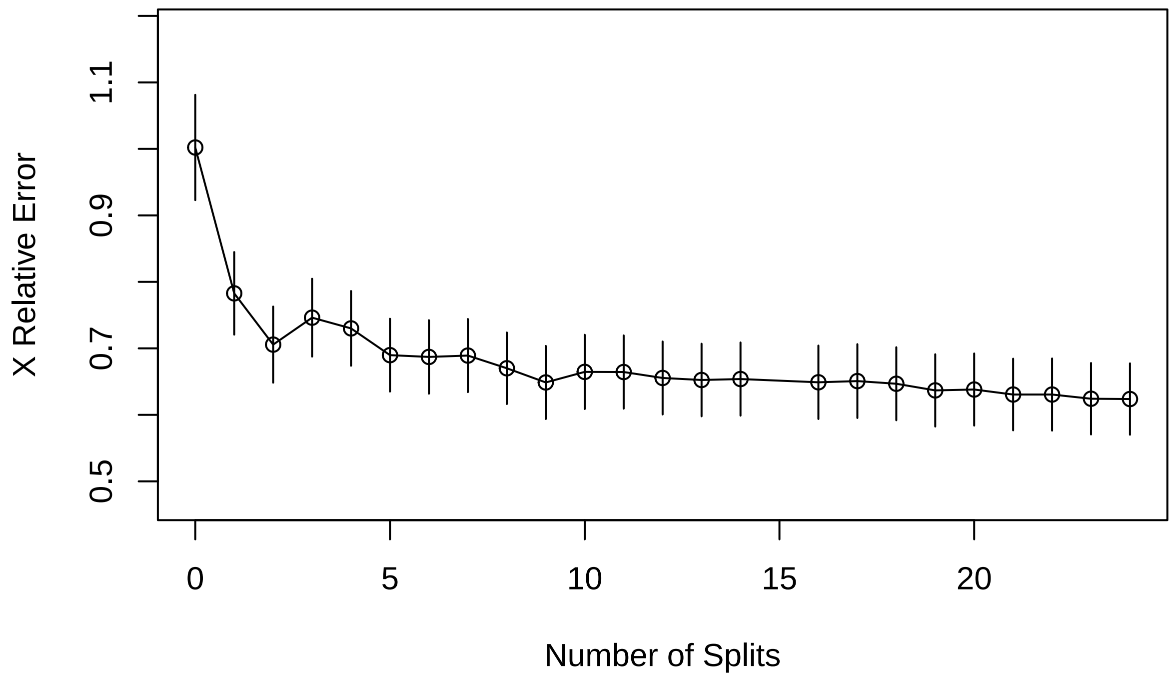
```
## 23 0.003943691     23 0.2455524 0.6242207 0.05371521
## 24 0.000100000     24 0.2416087 0.6237455 0.05370130
```

```
par(mfrow = c(1, 1)) # one plot on one page
rsq.rpart(mod_tree) # visualize cross-validation results
```

```
##
## Regression tree:
## rpart(formula = Sales ~ ., data = carseat.train, control = cont)
##
## Variables actually used in tree construction:
## [1] Advertising Age         CompPrice   Income      Price       ShelveLoc
##
## Root node error: 2135.5/280 = 7.6268
##
## n= 280
##
##           CP nsplit rel error  xerror     xstd
## 1  0.2240609      0   1.00000 1.00211 0.079180
## 2  0.1176101      1   0.77594 0.78273 0.062049
## 3  0.0526200      2   0.65833 0.70563 0.057216
## 4  0.0484941      3   0.60571 0.74616 0.058536
## 5  0.0458084      4   0.55721 0.73002 0.056173
## 6  0.0310297      5   0.51141 0.68977 0.054744
## 7  0.0309849      6   0.48038 0.68709 0.055251
## 8  0.0269325      7   0.44939 0.68915 0.054977
## 9  0.0199994      8   0.42246 0.67005 0.053781
## 10 0.0176125      9   0.40246 0.64870 0.054962
## 11 0.0174379     10   0.38485 0.66460 0.055831
## 12 0.0145874     11   0.36741 0.66438 0.055067
## 13 0.0136974     12   0.35282 0.65544 0.054884
## 14 0.0134481     13   0.33912 0.65239 0.054598
## 15 0.0129591     14   0.32568 0.65383 0.055039
## 16 0.0126177     16   0.29976 0.64895 0.055224
## 17 0.0090503     17   0.28714 0.65077 0.055541
## 18 0.0081952     18   0.27809 0.64676 0.054935
## 19 0.0071582     19   0.26990 0.63674 0.054379
## 20 0.0066031     20   0.26274 0.63806 0.054237
## 21 0.0065565     21   0.25613 0.63056 0.053870
## 22 0.0040250     22   0.24958 0.63053 0.054257
## 23 0.0039437     23   0.24555 0.62422 0.053715
## 24 0.0001000     24   0.24161 0.62375 0.053701
```



Apparent
X Relative
R−square
0.0
0.2
0.4
0.6
0.8
1.0
0
5
10
15
20
Number of Splits

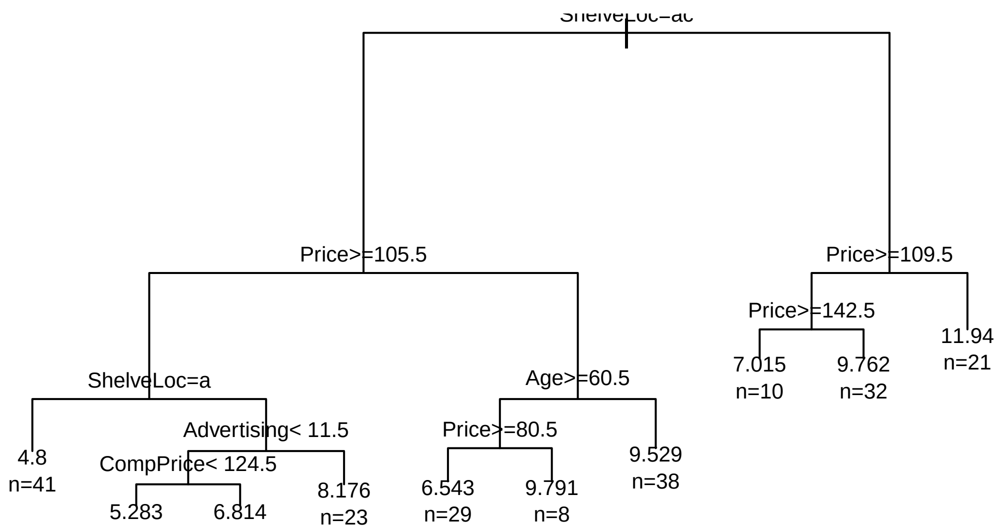


En regardant la table des cps et le résultats de la validation croiséé, on décide d'utiliser le résultat avec nsplits=09 (ligne 10), car le erreur croisée stabilize après cette valeur.

```
mod_tree$cptable[10, 1]
```

```
## [1] 0.01761249
```

```
mod_tree_pruned = prune(mod_tree, cp = 0.0176125)
par(mfrow = c(1, 1))
plot(mod_tree_pruned)
text(mod_tree_pruned, use.n = TRUE, cex = 0.7)
```




```
library(DiceEval)

y_pred_tree <- predict(mod_tree_pruned, carseat.test[, -1])
RMSE_cart = RMSE(carseat.test[, 1], y_pred_tree)
RMSE_cart
```

```
## [1] 2.138056
```

Random forest

Dans la random forest, le modèle choisi aleatoirement des paramètres a être enlevés avant de créer l'arbre de regression. Il crée donc um nombre importante d'arbres avec des différentes paramètres enlevés. À la fin, le résultat final sera donné par une mésure en considérant également tout ces modèles.

```
library(randomForest)
```

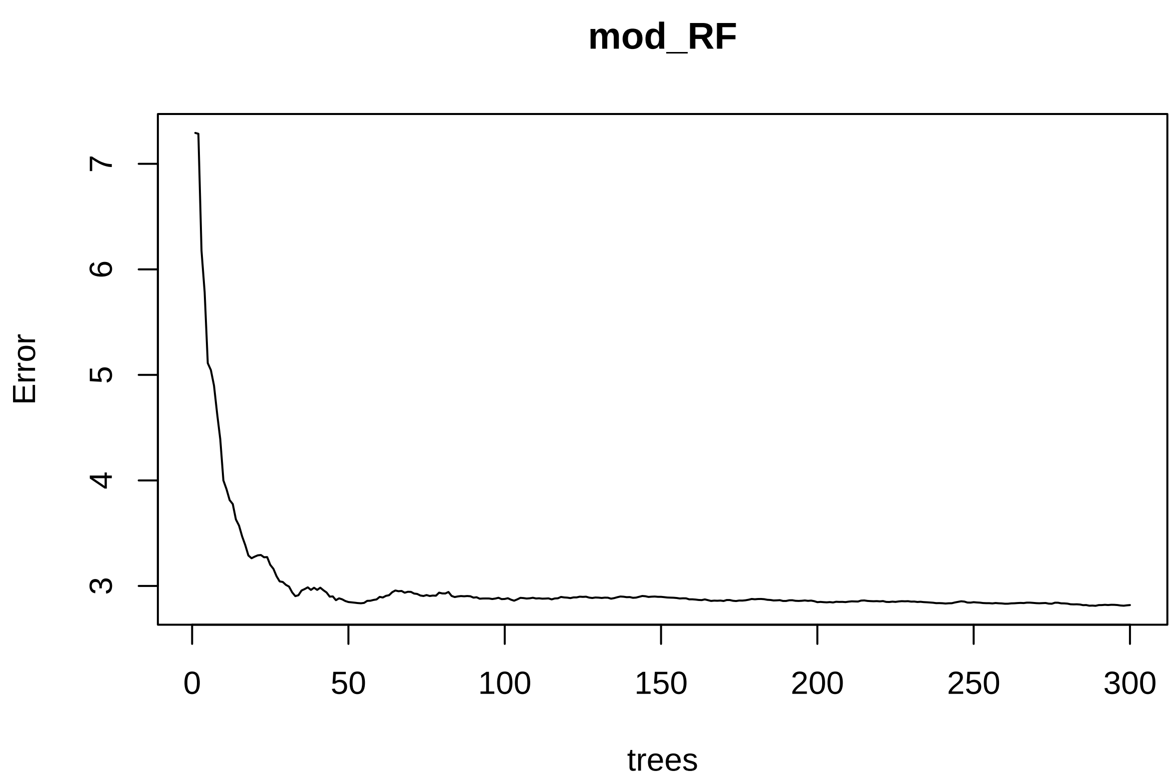
```
## randomForest 4.7-1.1
```

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

```
mod_RF <- randomForest(Sales~., data = carseat.train, ntree = 300, sampsize = nrow(carseat.train))
summary(mod_RF)
```

```
##                 Length Class  Mode
## call              5    -none- call
## type              1    -none- character
## predicted       280    -none- numeric
## mse             300    -none- numeric
## rsq             300    -none- numeric
## oob.times       280    -none- numeric
## importance       10    -none- numeric
## importanceSD      0    -none- NULL
## localImportance   0    -none- NULL
## proximity         0    -none- NULL
## ntree             1    -none- numeric
## mtry              1    -none- numeric
## forest           11    -none- list
## coefs             0    -none- NULL
## y               280    -none- numeric
## test              0    -none- NULL
## inbag             0    -none- NULL
## terms             3    terms  call
```

```
plot(mod_RF)
```



En obserant le plot, on observe que le meilleur valeur à utiliser doit être une valeur ligèrement inferieur à 100. Mais avec le randomForest du R, cette choix est fait automatiquement par le logiciel.

```r
Y_pred_RF <- predict(mod_RF,carseat.test)
RMSE_RF = RMSE(carseat.test[, 1], Y_pred_RF)
RMSE_RF
```

```
## [1] 1.899346
```

Bagging

Le bagging, d'autre part, considère encore une fois tous les paramètres. Mais cette fois-ci, il enlève plusieurs échantillons. Il donc construit différentes bases de entrainement avec échantillons aleatoirs de la base d'apprentissage. Chaque base sera utilisé pour construire une arbre de régression différent. Encore une fois, tous les arbres seront utilisés pour construire lé résultat final.

```r
cont = rpart.control(minsplit = 2, cp = 0.0001)
B = 2000
Y <- matrix(0, nrow(carseat.test), B)
for (i in 1:B) {
  u = sample(1:nrow(carseat.train), nrow(carseat.train), replace = TRUE)
  appren <- carseat.train[u, ]
  mod_bag <- rpart(Sales ~., data = appren, control = cont)
  Y[, i] <- predict(mod_bag, newdata = carseat.test)
}

Y_pred_bag = apply(Y, 1, mean)  # mean of each tree`s result
RMSE_bag = RMSE(carseat.test[, 1], Y_pred_bag)
RMSE_bag
```

```
## [1] 1.605508
```

Boosting

Le boosting est un cas particulier de Bagging. Mais dans ce cas, pour construire des échantillons d'apprentissage, il faut priorizer les données qui ont reçus l'erreur le plus importante avec le dernière modèle construit. Cette idée est important afin de améliorer la précision du modèle.

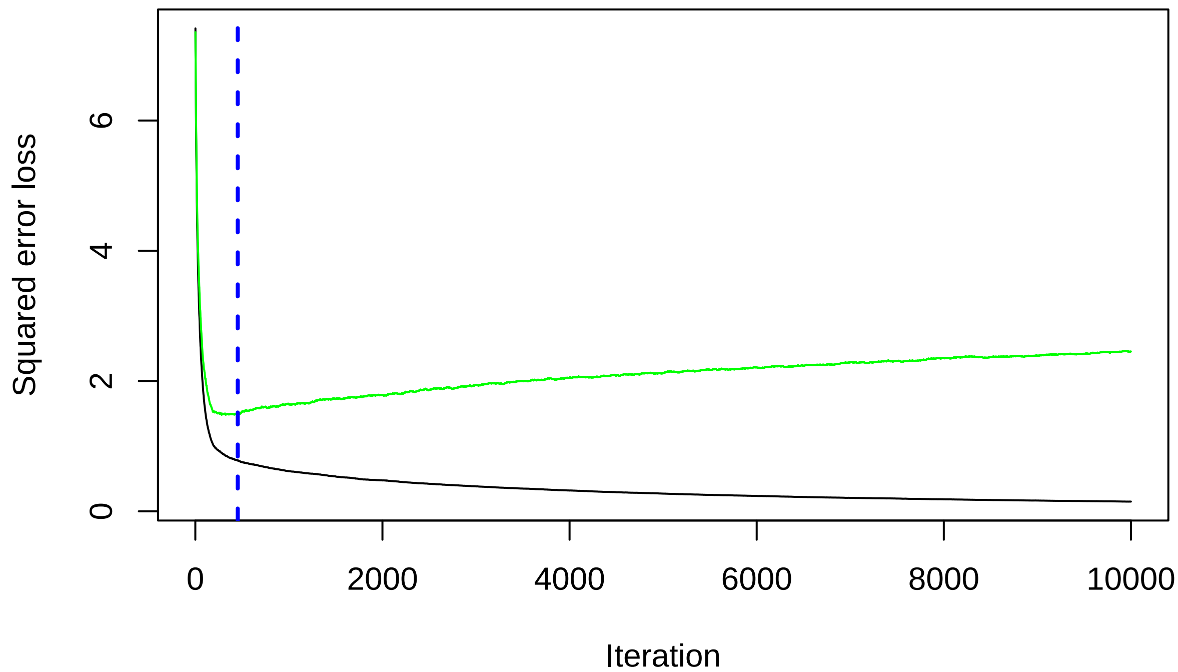
```r
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```r
mod_boosted <- gbm(Sales ~., data = carseat.train, n.trees = 10000, cv.folds = 10)
```

```
## Distribution not specified, assuming gaussian ...
```

```r
par(mfrow = c(1, 1))
best.iter <- gbm.perf(mod_boosted, method = "cv")
```

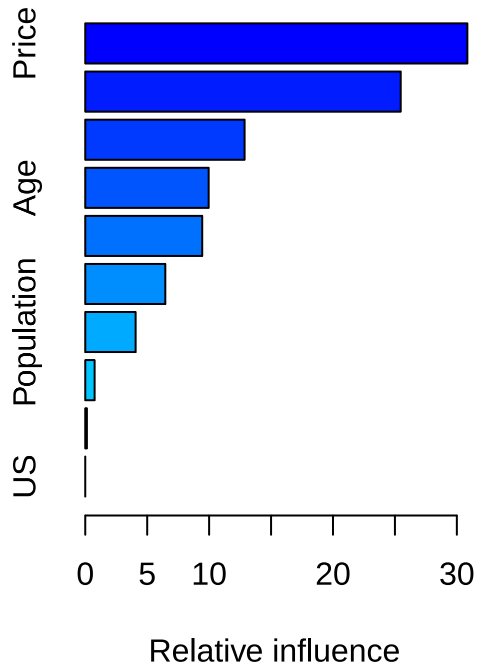



```
par(mfrow = c(1, 2))
summary(mod_boosted, n.trees = best.iter)
```

```
##                     var    rel.inf
## Price             Price 30.8476890
## ShelveLoc     ShelveLoc 25.4678158
## CompPrice     CompPrice 12.8622931
## Age                 Age  9.9633424
## Advertising Advertising  9.4416893
## Income           Income  6.4623071
## Population   Population  4.0665163
## Education     Education  0.7549492
## Urban             Urban  0.1333977
## US                   US  0.0000000
```

```
f.predict <- predict(mod_boosted, carseat.test[, -1], best.iter)
RMSE_bos = RMSE(carseat.test[, 1], f.predict)
RMSE_bos
```

```
## [1] 1.268251
```



3. Quelle est l'approche qui donne les meilleurs résultats sur l'ensemble test ?

```r
c(RMSE_cart, RMSE_RF, RMSE_bag, RMSE_bos)
```

```
## [1] 2.138056 1.899346 1.605508 1.268251
```

L'approche qui donne les meilleurs résultats sur l'ensemble test, c'est le boosting.

4. Si on décide de mettre en place un modèle linéaire avec une sélection backward sur les tests d'influence des variables, comment faut-il procéder ?

Pour identifier les variables significatives, on peut procèder par deux stratégies: AIC et BIC. La différence entre eux, c'est que le critére pour éliminer des variables sont differents. Pour l'AIC, c'est le critère de Akaike, tandis que pour le BIC, c'est le critère Bayesienne. Avec AIC, la pénalité est de 2k, alors qu'avec BIC, la pénalité est de ln(n)k.

```r
mod_lineire = lm(Sales~., data = carseat)
summary(mod_lineire)
```

```
##
## Call:
```

```
## lm(formula = Sales ~ ., data = carseat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.8692 -0.6908  0.0211  0.6636  3.4115
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      5.6606231  0.6034487   9.380  < 2e-16 ***
## CompPrice        0.0928153  0.0041477  22.378  < 2e-16 ***
## Income           0.0158028  0.0018451   8.565 2.58e-16 ***
## Advertising      0.1230951  0.0111237  11.066  < 2e-16 ***
## Population       0.0002079  0.0003705   0.561    0.575
## Price           -0.0953579  0.0026711 -35.700  < 2e-16 ***
## ShelveLocGood    4.8501827  0.1531100  31.678  < 2e-16 ***
## ShelveLocMedium  1.9567148  0.1261056  15.516  < 2e-16 ***
## Age             -0.0460452  0.0031817 -14.472  < 2e-16 ***
## Education       -0.0211018  0.0197205  -1.070    0.285
## UrbanYes         0.1228864  0.1129761   1.088    0.277
## USYes           -0.1840928  0.1498423  -1.229    0.220
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.019 on 388 degrees of freedom
## Multiple R-squared:  0.8734, Adjusted R-squared:  0.8698
## F-statistic: 243.4 on 11 and 388 DF,  p-value: < 2.2e-16
```

```r
slm_AIC = step(mod_lineire, direction="backward", k = 2)
```

```
## Start:  AIC=26.82
## Sales ~ CompPrice + Income + Advertising + Population + Price +
##     ShelveLoc + Age + Education + Urban + US
##
##               Df Sum of Sq     RSS    AIC
## - Population   1      0.33  403.16  25.15
## - Education    1      1.19  404.02  26.00
## - Urban        1      1.23  404.06  26.04
## - US           1      1.57  404.40  26.38
## <none>                      402.83  26.82
## - Income       1     76.16  478.99  94.09
## - Advertising  1    127.14  529.97 134.54
## - Age          1    217.44  620.27 197.48
## - CompPrice    1    519.91  922.74 356.35
## - ShelveLoc    2   1053.20 1456.03 536.80
## - Price        1   1323.23 1726.06 606.85
##
## Step:  AIC=25.15
## Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc +
##     Age + Education + Urban + US
##
##               Df Sum of Sq     RSS    AIC
## - Urban        1      1.15  404.31  24.29
## - Education    1      1.36  404.52  24.49
## - US           1      1.89  405.05  25.02
```

```
## <none>                       403.16  25.15
## - Income       1     75.94  479.10  92.18
## - Advertising  1    145.38  548.54 146.32
## - Age          1    218.52  621.68 196.38
## - CompPrice    1    521.69  924.85 355.27
## - ShelveLoc    2   1053.18 1456.34 534.89
## - Price        1   1323.51 1726.67 605.00
##
## Step:  AIC=24.29
## Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc +
##     Age + Education + US
##
##               Df Sum of Sq     RSS    AIC
## - Education    1      1.44  405.76  23.72
## - US           1      1.85  406.16  24.12
## <none>                      404.31  24.29
## - Income       1     76.64  480.96  91.73
## - Advertising  1    146.03  550.34 145.63
## - Age          1    217.59  621.91 194.53
## - CompPrice    1    526.17  930.48 355.69
## - ShelveLoc    2   1053.93 1458.25 533.41
## - Price        1   1322.80 1727.11 603.10
##
## Step:  AIC=23.72
## Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc +
##     Age + US
##
##               Df Sum of Sq     RSS    AIC
## - US           1      1.63  407.39  23.32
## <none>                      405.76  23.72
## - Income       1     77.87  483.62  91.94
## - Advertising  1    145.30  551.06 144.15
## - Age          1    217.97  623.73 193.70
## - CompPrice    1    525.25  931.00 353.92
## - ShelveLoc    2   1056.88 1462.64 532.61
## - Price        1   1322.83 1728.58 601.44
##
## Step:  AIC=23.32
## Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc +
##     Age
##
##               Df Sum of Sq     RSS    AIC
## <none>                      407.39  23.32
## - Income       1     76.68  484.07  90.30
## - Age          1    219.12  626.51 193.48
## - Advertising  1    234.03  641.42 202.89
## - CompPrice    1    523.83  931.22 352.01
## - ShelveLoc    2   1055.51 1462.90 530.68
## - Price        1   1324.42 1731.81 600.18
```

Avec le modèle AIC, on arrive aux variables explicatives Income, Age, Advertising, CompPrice, ShelveLoc et Price.

```r
slm_BIC = step(mod_lineire, direction="backward", k = log(nrow(carseat)))
```

```
## Start:  AIC=74.72
## Sales ~ CompPrice + Income + Advertising + Population + Price +
##     ShelveLoc + Age + Education + Urban + US
##
##               Df Sum of Sq     RSS    AIC
## - Population   1      0.33  403.16  69.05
## - Education    1      1.19  404.02  69.91
## - Urban        1      1.23  404.06  69.95
## - US           1      1.57  404.40  70.28
## <none>                      402.83  74.72
## - Income       1     76.16  478.99 137.99
## - Advertising  1    127.14  529.97 178.45
## - Age          1    217.44  620.27 241.38
## - CompPrice    1    519.91  922.74 400.26
## - ShelveLoc    2   1053.20 1456.03 576.72
## - Price        1   1323.23 1726.06 650.76
##
## Step:  AIC=69.05
## Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc +
##     Age + Education + Urban + US
##
##               Df Sum of Sq     RSS    AIC
## - Urban        1      1.15  404.31  64.21
## - Education    1      1.36  404.52  64.41
## - US           1      1.89  405.05  64.94
## <none>                      403.16  69.05
## - Income       1     75.94  479.10 132.09
## - Advertising  1    145.38  548.54 186.23
## - Age          1    218.52  621.68 236.30
## - CompPrice    1    521.69  924.85 395.18
## - ShelveLoc    2   1053.18 1456.34 570.81
## - Price        1   1323.51 1726.67 644.91
##
## Step:  AIC=64.21
## Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc +
##     Age + Education + US
##
##               Df Sum of Sq     RSS    AIC
## - Education    1      1.44  405.76  59.64
## - US           1      1.85  406.16  60.04
## <none>                      404.31  64.21
## - Income       1     76.64  480.96 127.65
## - Advertising  1    146.03  550.34 181.55
## - Age          1    217.59  621.91 230.45
## - CompPrice    1    526.17  930.48 391.62
## - ShelveLoc    2   1053.93 1458.25 565.34
## - Price        1   1322.80 1727.11 639.02
##
## Step:  AIC=59.64
## Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc +
##     Age + US
```

```
##
##               Df Sum of Sq     RSS    AIC
## - US           1      1.63  407.39  55.25
## <none>                      405.76  59.64
## - Income       1     77.87  483.62 123.87
## - Advertising  1    145.30  551.06 176.08
## - Age          1    217.97  623.73 225.63
## - CompPrice    1    525.25  931.00 385.85
## - ShelveLoc    2   1056.88 1462.64 560.55
## - Price        1   1322.83 1728.58 633.37
##
## Step:  AIC=55.25
## Sales ~ CompPrice + Income + Advertising + Price + ShelveLoc +
##     Age
##
##               Df Sum of Sq     RSS    AIC
## <none>                      407.39  55.25
## - Income       1     76.68  484.07 118.24
## - Age          1    219.12  626.51 221.42
## - Advertising  1    234.03  641.42 230.83
## - CompPrice    1    523.83  931.22 379.95
## - ShelveLoc    2   1055.51 1462.90 554.63
## - Price        1   1324.42 1731.81 628.12
```

Avec le BIC, on arrive aux mêmes variables.