Project Machine Learning

Cesare Cottonaro

2022-05-18

In this exercise I apply 3 types of subset selection: Best Subset Selection, Forward Step-Wise Selection and Backward Step-Wise Selection. So, I start to compute the Best Subset Selection (BSS), Forward Step-Wise Selection (FWS) and Backward Step-Wise Selection (BWS) loading the data from the library ISLR2, included in the R memory and remove the 9th column, because it has a structure that R calls factors, that is a mix of numbers and strings that R is not able to read, and it is useless for the purpose of our analysis:

Question 6

point a)

```
library(ISLR2)
auto = Auto
auto=auto[,-9]
p = ncol(auto)
attach(auto)
## best subset selection

library(leaps)
reg.full=regsubsets(mpg~.,data=auto, nvmax = p-1)
summary(reg.full)
```

```
## Subset selection object
## Call: regsubsets.formula(mpg ~ ., data = auto, nvmax = p - 1)
## 7 Variables  (and intercept)
##              Forced in Forced out
## cylinders        FALSE      FALSE
## displacement     FALSE      FALSE
## horsepower       FALSE      FALSE
## weight           FALSE      FALSE
## acceleration     FALSE      FALSE
## year             FALSE      FALSE
## origin           FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: exhaustive
##          cylinders displacement horsepower weight acceleration year origin
## 1  ( 1 ) " "       " "          " "        "*"    " "          " "  " "
## 2  ( 1 ) " "       " "          " "        "*"    " "          "*"  " "
## 3  ( 1 ) " "       " "          " "        "*"    " "          "*"  "*"
```

```
## 4  ( 1 ) " "       "*"          " "         "*"    " "          "*"  "*"
## 5  ( 1 ) " "       "*"          "*"         "*"    " "          "*"  "*"
## 6  ( 1 ) "*"       "*"          "*"         "*"    " "          "*"  "*"
## 7  ( 1 ) "*"       "*"          "*"         "*"    "*"          "*"  "*"
```

```r
## forward step-wise selection

reg.fwd = regsubsets(mpg~., data=auto, nvmax = p-1, method = "forward")
summary(reg.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(mpg ~ ., data = auto, nvmax = p - 1, method = "forward")
## 7 Variables  (and intercept)
##              Forced in Forced out
## cylinders        FALSE      FALSE
## displacement     FALSE      FALSE
## horsepower       FALSE      FALSE
## weight           FALSE      FALSE
## acceleration     FALSE      FALSE
## year             FALSE      FALSE
## origin           FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: forward
##          cylinders displacement horsepower weight acceleration year origin
## 1  ( 1 ) " "       " "          " "         "*"    " "          " "  " "
## 2  ( 1 ) " "       " "          " "         "*"    " "          "*"  " "
## 3  ( 1 ) " "       " "          " "         "*"    " "          "*"  "*"
## 4  ( 1 ) " "       "*"          " "         "*"    " "          "*"  "*"
## 5  ( 1 ) " "       "*"          "*"         "*"    " "          "*"  "*"
## 6  ( 1 ) "*"       "*"          "*"         "*"    " "          "*"  "*"
## 7  ( 1 ) "*"       "*"          "*"         "*"    "*"          "*"  "*"
```

```r
## backward step-wise selection

reg.bwd=regsubsets(mpg~., data = auto, nvmax = p-1, method = "backward")
summary(reg.bwd)
```

```
## Subset selection object
## Call: regsubsets.formula(mpg ~ ., data = auto, nvmax = p - 1, method = "backward")
## 7 Variables  (and intercept)
##              Forced in Forced out
## cylinders        FALSE      FALSE
## displacement     FALSE      FALSE
## horsepower       FALSE      FALSE
## weight           FALSE      FALSE
## acceleration     FALSE      FALSE
## year             FALSE      FALSE
## origin           FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: backward
##          cylinders displacement horsepower weight acceleration year origin
## 1  ( 1 ) " "       " "          " "         "*"    " "          " "  " "
## 2  ( 1 ) " "       " "          " "         "*"    " "          "*"  " "
```

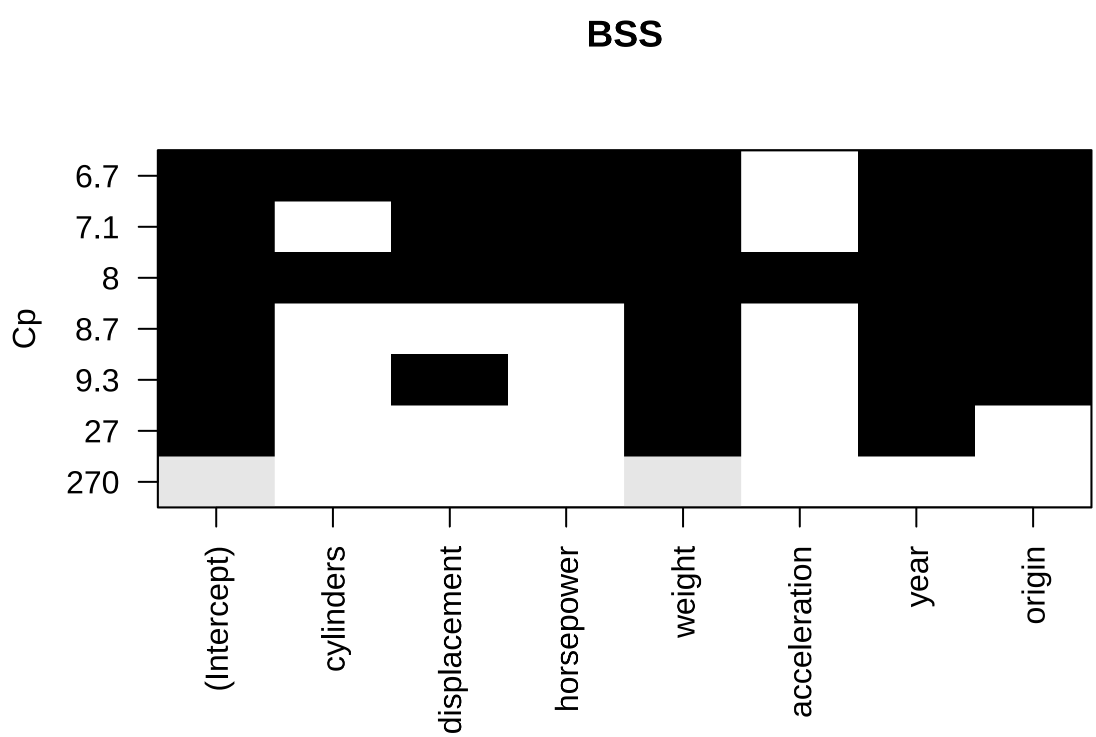
```
## 3  ( 1 ) " "        " "           " "         "*"     " "           "*"   "*"
## 4  ( 1 ) " "        "*"           " "         "*"     " "           "*"   "*"
## 5  ( 1 ) " "        "*"           "*"         "*"     " "           "*"   "*"
## 6  ( 1 ) "*"        "*"           "*"         "*"     " "           "*"   "*"
## 7  ( 1 ) "*"        "*"           "*"         "*"     "*"           "*"   "*"
```

I used the library *leaps* to perform the subset selection, and with the command regsubsets I perform the Best Subset Selection by default, adding *method = forward/backward* we perform the other two methods. With the command *summary* we can see which are the best models with the highest R square.

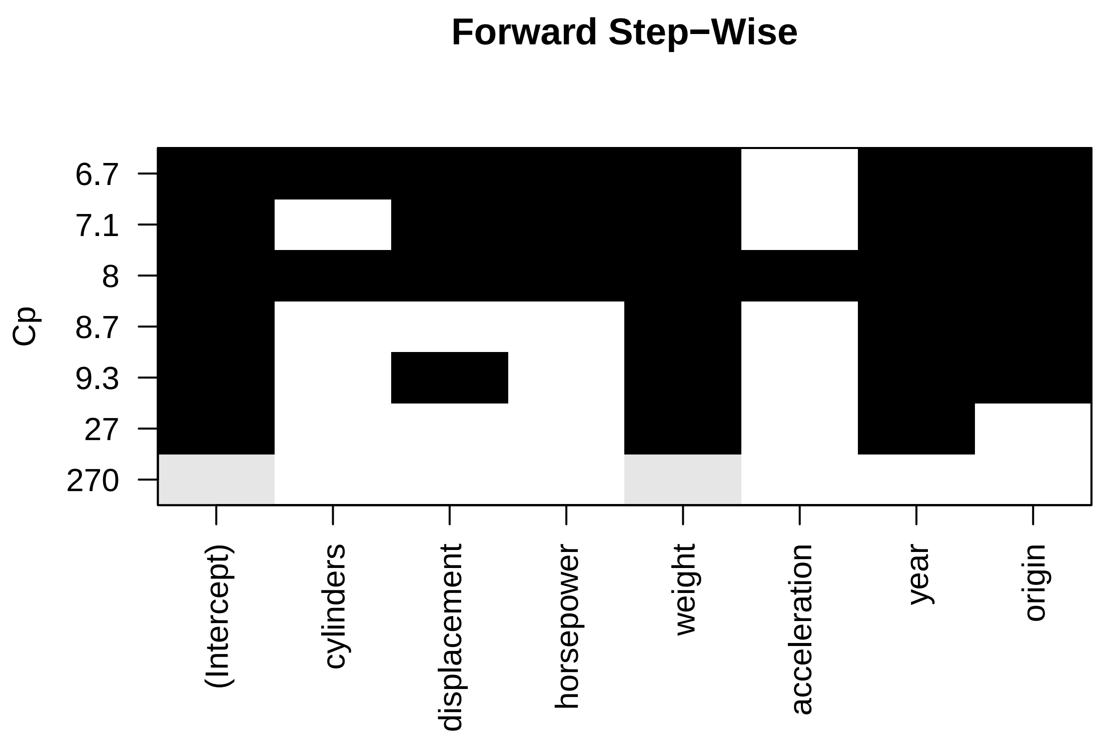
point b)

Then, I calculate the Mallow's Cp, that is an information criterion that allows us to choose the best subset of regressors that minimize the Mean Square Error. The lowest Mallow's Cp is chosen:

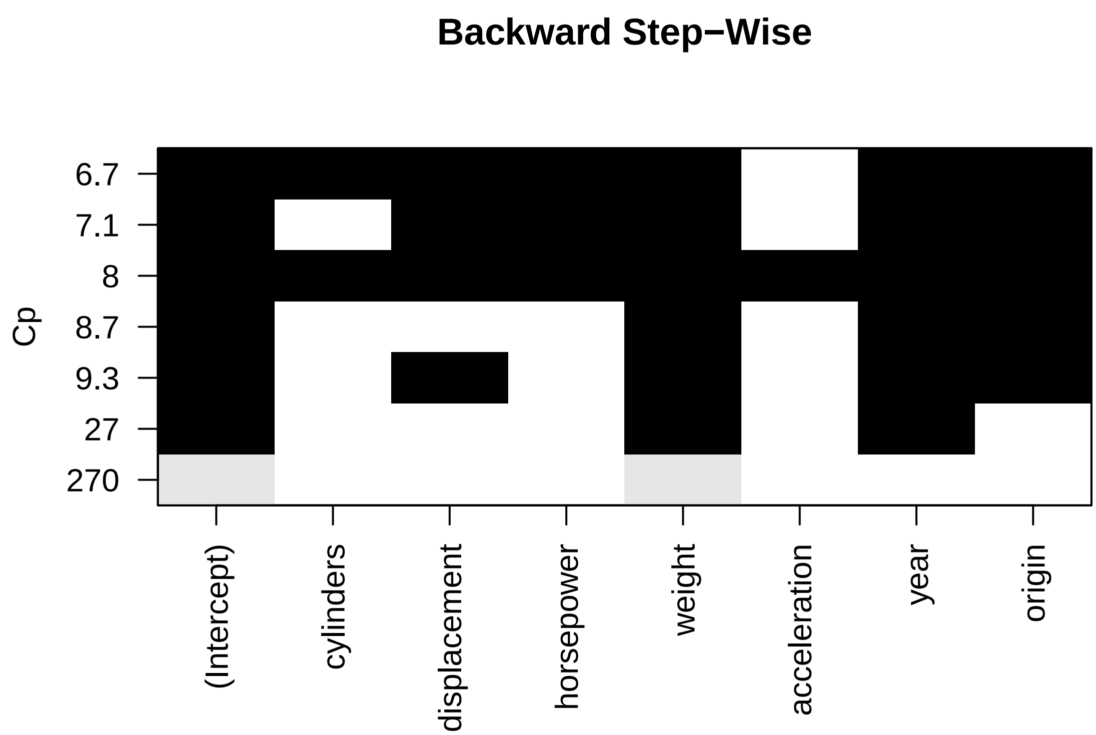
```
summary.reg.full=summary(reg.full)
summary.reg.fwd=summary(reg.fwd)
summary.reg.bwd=summary(reg.bwd)

## plot Mallow's Cp Plots

plot(reg.full, scale = "Cp", main = "BSS")
```



```r
plot(reg.fwd, scale = "Cp", main = "Forward Step-Wise")
```




```r
plot(reg.bwd, scale = "Cp", main="Backward Step-Wise")
```




```r
par(mfrow = c(1,3))

## Best model according to BSS

which.min(summary.reg.full$cp)
```

```
## [1] 6
```

```r
summary.reg.full$cp[6]
```

```
## [1] 6.664509
```

```r
plot(summary.reg.full$cp, xlab ="Number of regressors", ylab = "Cp", type = "b", main = "Cp Plot Best")
points(which.min(summary.reg.full$cp),summary.reg.full$cp[6], col = "red", cex = 2, pch = 16)

## Best model according to FWS

which.min(summary.reg.fwd$cp)
```

```
## [1] 6
```

```r
summary.reg.fwd$cp[6]
```

```
## [1] 6.664509
```

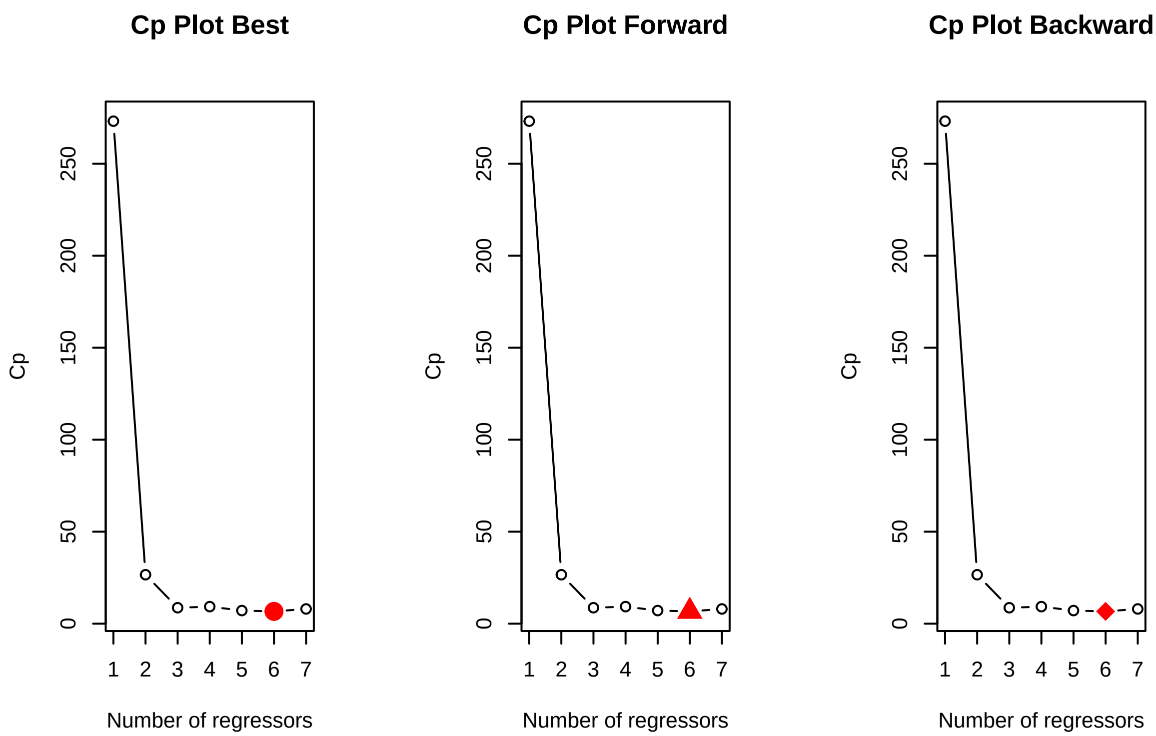
```r
plot(summary.reg.fwd$cp, xlab = "Number of regressors", ylab = "Cp", main = "Cp Plot Forward", type = "
points(which.min(summary.reg.fwd$cp),summary.reg.fwd$cp[6], col = "red", cex = 2, pch = 17)

## Best model according to BWS

which.min(summary.reg.bwd$cp)
```

```
## [1] 6
```

```r
summary.reg.bwd$cp[6]
```

```
## [1] 6.664509
```

```r
plot(summary.reg.bwd$cp, xlab = "Number of regressors", ylab = "Cp", main = "Cp Plot Backward", type =
points(which.min(summary.reg.bwd$cp),summary.reg.bwd$cp[6], col = "red", cex = 2, pch = 18)
```



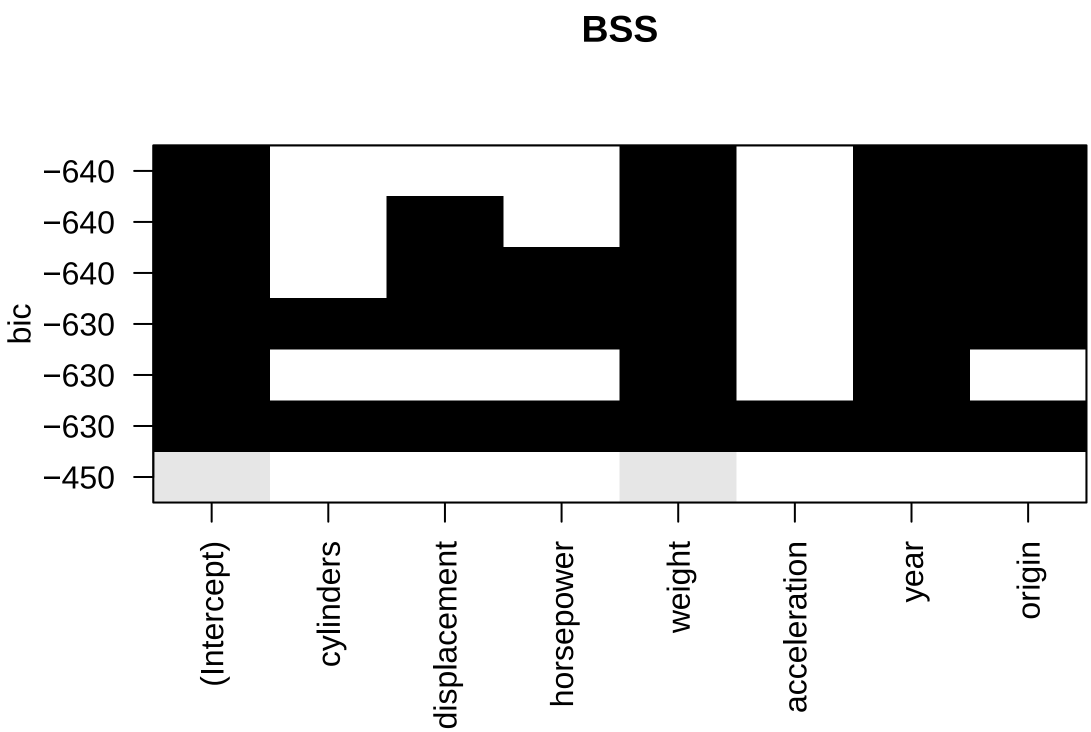

```r
par(mfrow = c(1,1))
```

The result is that, according to Mallow's Cp, the best models count 6 regressors, its value is worth 6.664509.

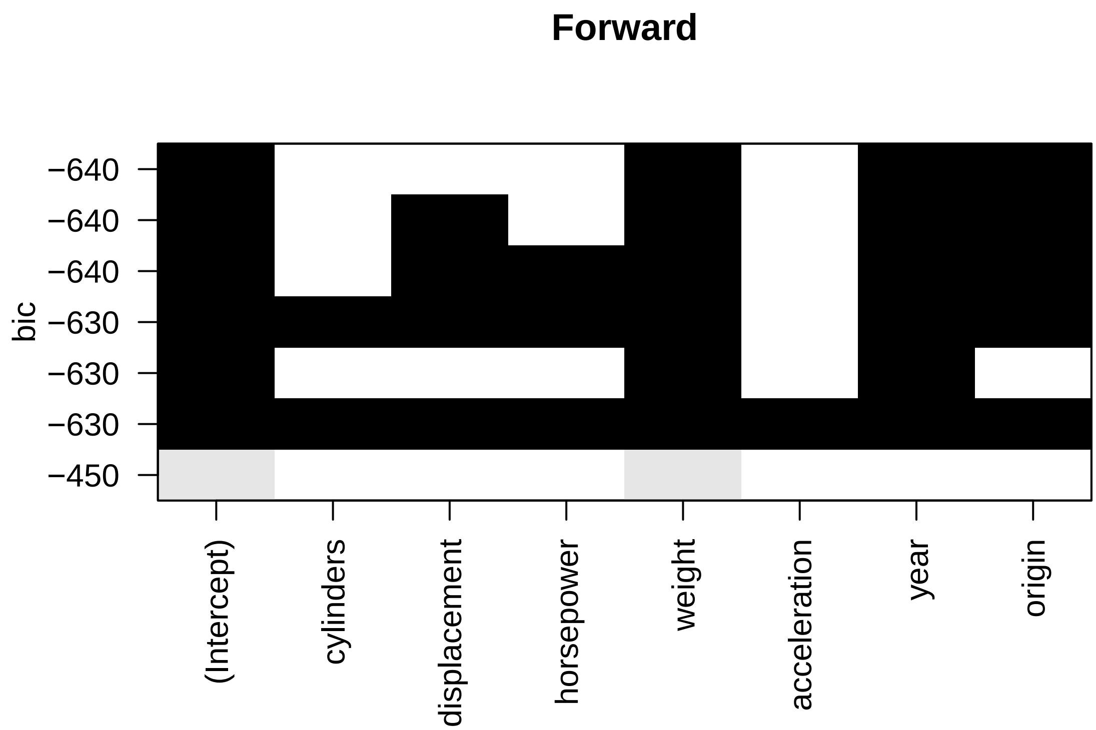
point c)

Now, for the 3 methods computed before we compute the BIC, that is another information criterion like the Mallow's Cp but more parsimonious, because it has an higher penalization for models with an high number of regressors. What we get is that before with the Mallow's Cp I have as best subset 6 regressors, with BIC I get as best subset 3 regressors. I report the plots of the BIC and the plots of Mallow's Cp one against the other for all the methods computed before, in order to catch the different results and to compare the output

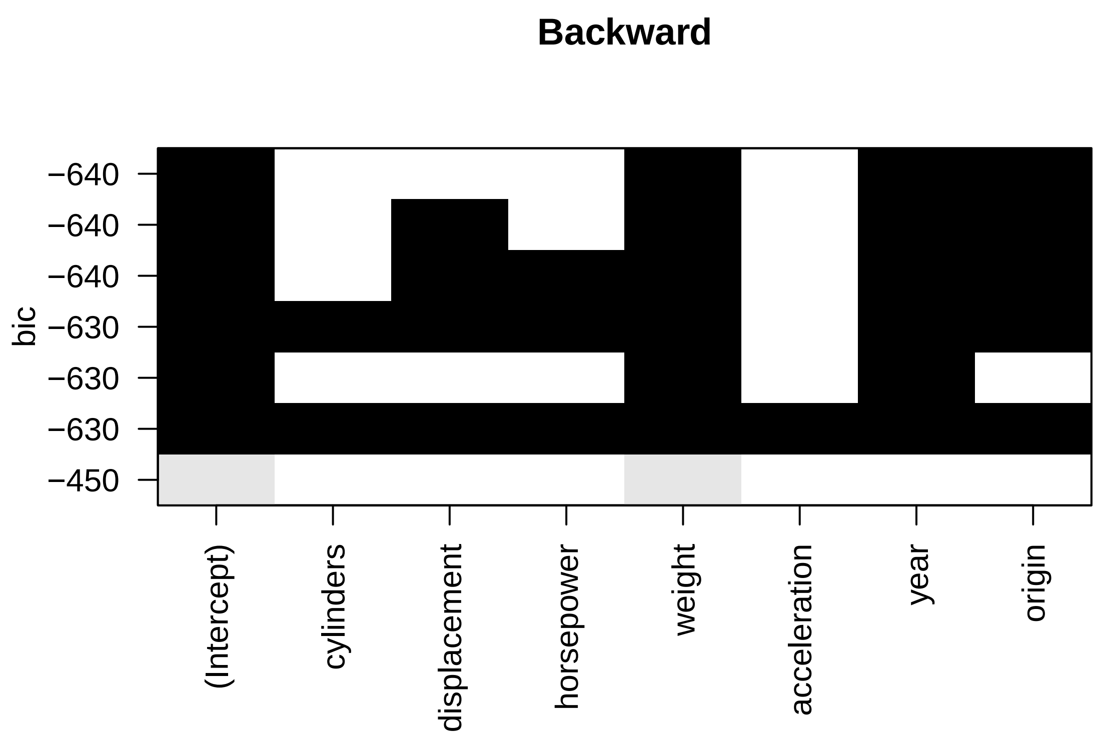
```
plot(reg.full, scale = "bic", main = "BSS")
```




```
plot(reg.fwd, scale = "bic", main = "Forward")
```




```
plot(reg.bwd, scale = "bic", main = "Backward")
```




```
which.min(summary.reg.full$bic)
```

```
## [1] 3
```

```
which.min(summary.reg.fwd$bic)
```

```
## [1] 3
```

```
which.min(summary.reg.bwd$bic)
```

```
## [1] 3
```

```
summary.reg.full$bic[3]
```

```
## [1] -642.8063
```

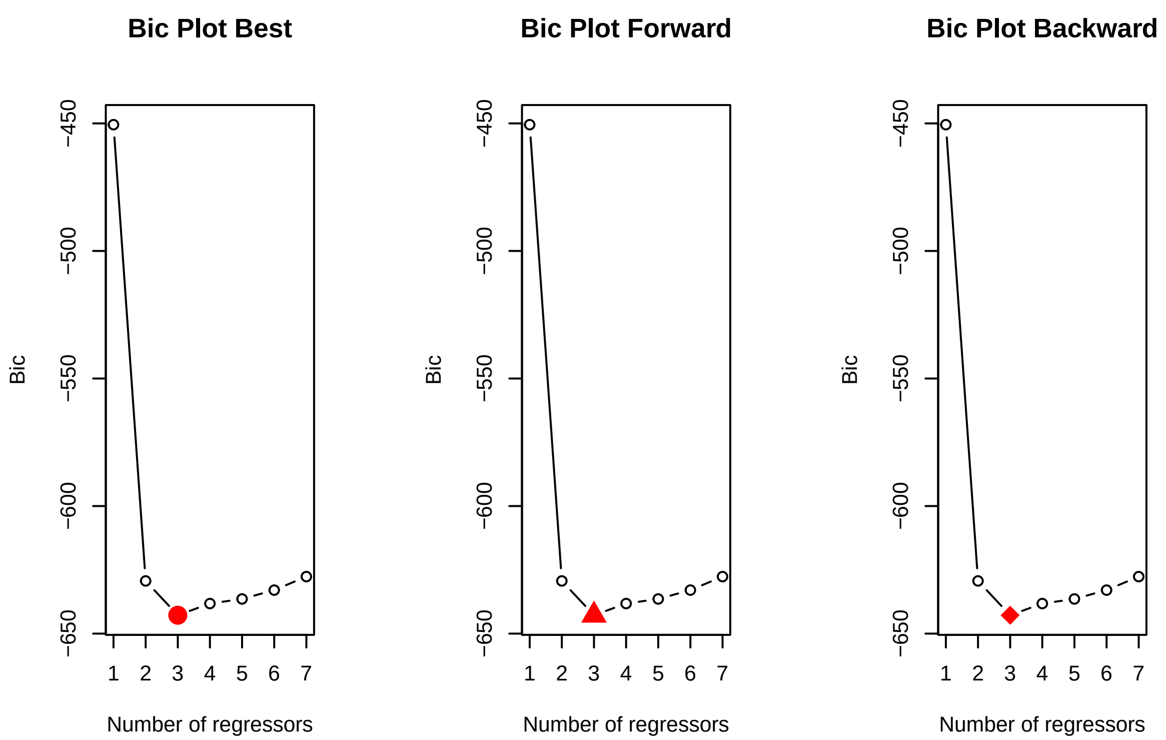
```
summary.reg.fwd$bic[3]
```

```
## [1] -642.8063
```

```r
summary.reg.bwd$bic[3]
```

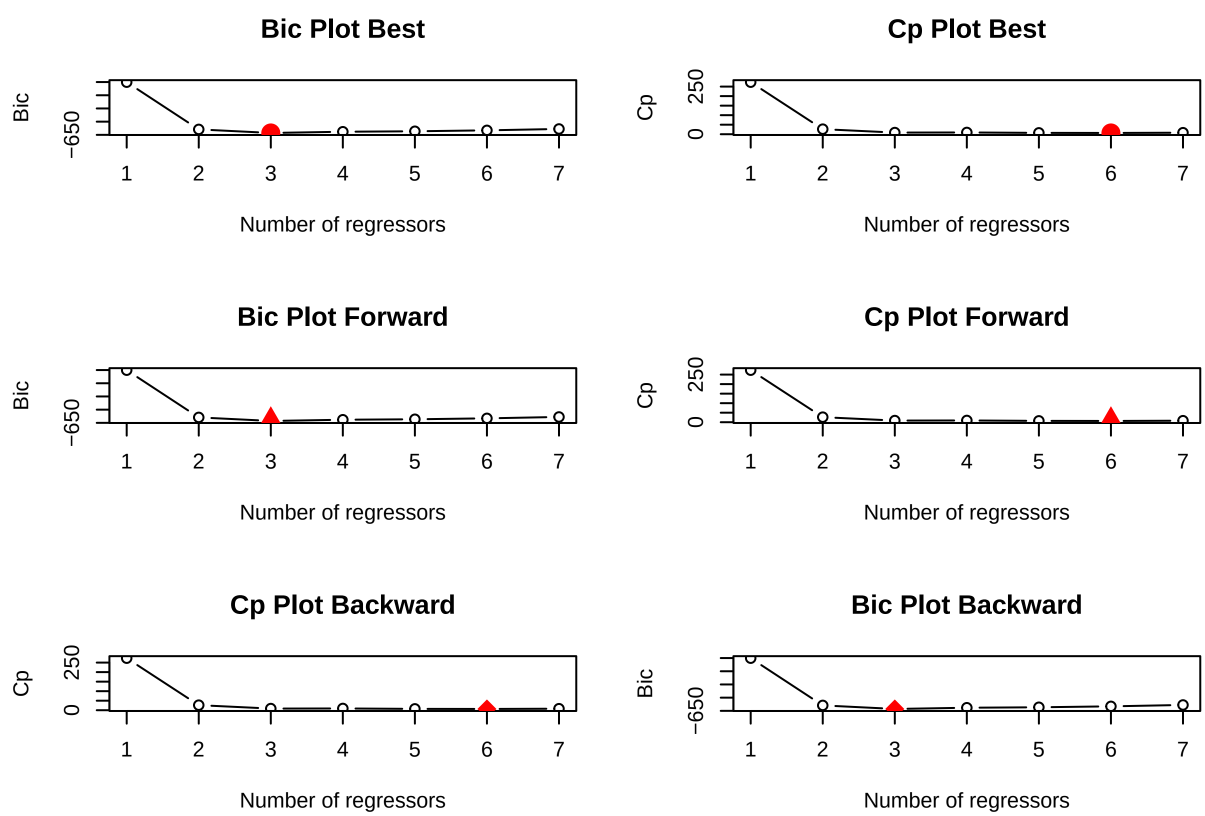
```
## [1] -642.8063
```

```r
par(mfrow = c(1,3))

plot(summary.reg.full$bic, xlab = "Number of regressors", ylab = "Bic", main = "Bic Plot Best", type =
points(which.min(summary.reg.full$bic),summary.reg.full$bic[3], col = "red", cex = 2, pch = 16)

plot(summary.reg.fwd$bic, xlab = "Number of regressors", ylab = "Bic", main = "Bic Plot Forward", type =
points(which.min(summary.reg.fwd$bic),summary.reg.fwd$bic[3], col = "red", cex = 2, pch = 17)

plot(summary.reg.bwd$bic, xlab = "Number of regressors", ylab = "Bic", main = "Bic Plot Backward", type
points(which.min(summary.reg.bwd$bic),summary.reg.bwd$bic[3], col = "red", cex = 2, pch = 18)
```




```r
win.graph()
par(mfrow = c(3,2))


plot(summary.reg.full$bic, xlab = "Number of regressors", ylab = "Bic", main = "Bic Plot Best", type =
points(which.min(summary.reg.full$bic),summary.reg.full$bic[3], col = "red", cex = 2, pch = 16)
plot(summary.reg.full$cp, xlab ="Number of regressors", ylab = "Cp", type = "b", main = "Cp Plot Best")
points(which.min(summary.reg.full$cp),summary.reg.full$cp[6], col = "red", cex = 2, pch = 16)
```

```
plot(summary.reg.fwd$bic, xlab = "Number of regressors", ylab = "Bic", main = "Bic Plot Forward", type =
points(which.min(summary.reg.fwd$bic),summary.reg.fwd$bic[3], col = "red", cex = 2, pch = 17)
plot(summary.reg.fwd$cp, xlab = "Number of regressors", ylab = "Cp", main = "Cp Plot Forward", type = "
points(which.min(summary.reg.fwd$cp),summary.reg.fwd$cp[6], col = "red", cex = 2, pch = 17)

plot(summary.reg.bwd$cp, xlab = "Number of regressors", ylab = "Cp", main = "Cp Plot Backward", type =
points(which.min(summary.reg.bwd$cp),summary.reg.bwd$cp[6], col = "red", cex = 2, pch = 18)
plot(summary.reg.bwd$bic, xlab = "Number of regressors", ylab = "Bic", main = "Bic Plot Backward", type
points(which.min(summary.reg.bwd$bic),summary.reg.bwd$bic[3], col = "red", cex = 2, pch = 18)
```




```
par(mfrow = c(1,1))
```

The value of the BIC corresponding to the model that minimize the MSE is -642.8063.

point d)

The K-Fold Cross Validation method is a method to estimate the total Mean Square Error, in this setting we split the set in k groups, or folds, with approximately the same size. The first is used as validation set, and the method is used in the remaining k-1 folds. The MSE_1 is computed on the observations in the held-out fold. Then the procedure is repeated k times, each time a different groups of observations is treated as a validation set. So we average all of the k MSE. In this case we use *k=10* because, from the empirical evidence, it is a good compromise for the Variance-Bias Trade-Off.

```r
set.seed(1)
win.graph()
par(mfrow = c(1,3))

k = 10
n = nrow(auto)
p = ncol(auto)
folds = sample(rep(1:k, length = n))
cv10.errors.best = matrix( NA, k, (p-1), dimnames = list(NULL, paste(1:(p-1))))

predict.regsubsets = function(object, newdata, id, ...){
  form = as.formula(object$call[[2]])
  mat = model.matrix(form, newdata)
  coefi = coef(object, id = id)
  xvars = names(coefi)
  mat[,xvars]%*%coefi
}

for (i in 1:k){
  best.fit = regsubsets(mpg~., data = auto[folds!= i,], nvmax = (p-1))
  for(j in 1:(p-1)){
    pred = predict.regsubsets(best.fit, auto[folds == i,], id = j)
    cv10.errors.best[i,j] = mean((auto$mpg[folds == i]-pred)^2)
  }
}

dim(cv10.errors.best)
```

```
## [1] 10  7
```

```r
mean.cv10.errors.best = apply(cv10.errors.best, 2, mean)
mean.cv10.errors.best
```

```
##        1        2        3        4        5        6        7
## 18.85951 11.83614 11.29446 11.46775 11.47221 11.37871 11.29750
```

```r
hBSS = which.min(mean.cv10.errors.best)
hBSS
```

```
## 3
## 3
```

```r
plot(mean.cv10.errors.best, xlab = "Number of regresors", main = "10 Fold Cross Validation BSS Plot", ty
points(hBSS,mean.cv10.errors.best[hBSS], col = "red", cex = 2, pch = 23 )

## K-Folds Cross Validation for Forward Step-Wise

cv10.errors.fwd = matrix(NA, k, p-1, dimnames = list(NULL, paste(1:(p-1))))

for (i in 1:k){
  best.fit = regsubsets(mpg~., data = auto[folds!= i,], nvmax = (p-1), method = "forward")
  for(j in 1:(p-1)){
```

```
    pred = predict.regsubsets(best.fit, auto[folds == i,], id= j)
    cv10.errors.fwd[i,j] = mean((auto$mpg[folds == i]-pred)^2)
  }
}

dim(cv10.errors.fwd)
```

```
## [1] 10  7
```

```
mean.cv10.errors.fwd = apply(cv10.errors.fwd, 2, mean)
mean.cv10.errors.fwd
```

```
##        1        2        3        4        5        6        7
## 18.85951 11.83614 11.29446 11.46775 11.47221 11.38985 11.29750
```

```
hFWD = which.min(mean.cv10.errors.fwd)
hFWD
```

```
## 3
## 3
```

```
plot(mean.cv10.errors.fwd, xlab = "Number of regresors", main = "10 Fold Cross Validation FWD Plot", typ
points(hFWD, mean.cv10.errors.best[hFWD], col = "red", cex = 2, pch = 24 )

## K-Folds Cross Validation for Backward Step-Wise

cv10.errors.bwd = matrix(NA, k, p-1, dimnames = list(NULL, paste(1:(p-1))))

for (i in 1:k){
  best.fit = regsubsets(mpg~., data = auto[folds!= i,], nvmax = (p-1), method = "backward")
  for(j in 1:(p-1)){
    pred = predict.regsubsets(best.fit, auto[folds == i,], id= j)
    cv10.errors.bwd[i,j] = mean((auto$mpg[folds == i]-pred)^2)
  }
}

dim(cv10.errors.bwd)
```

```
## [1] 10  7
```

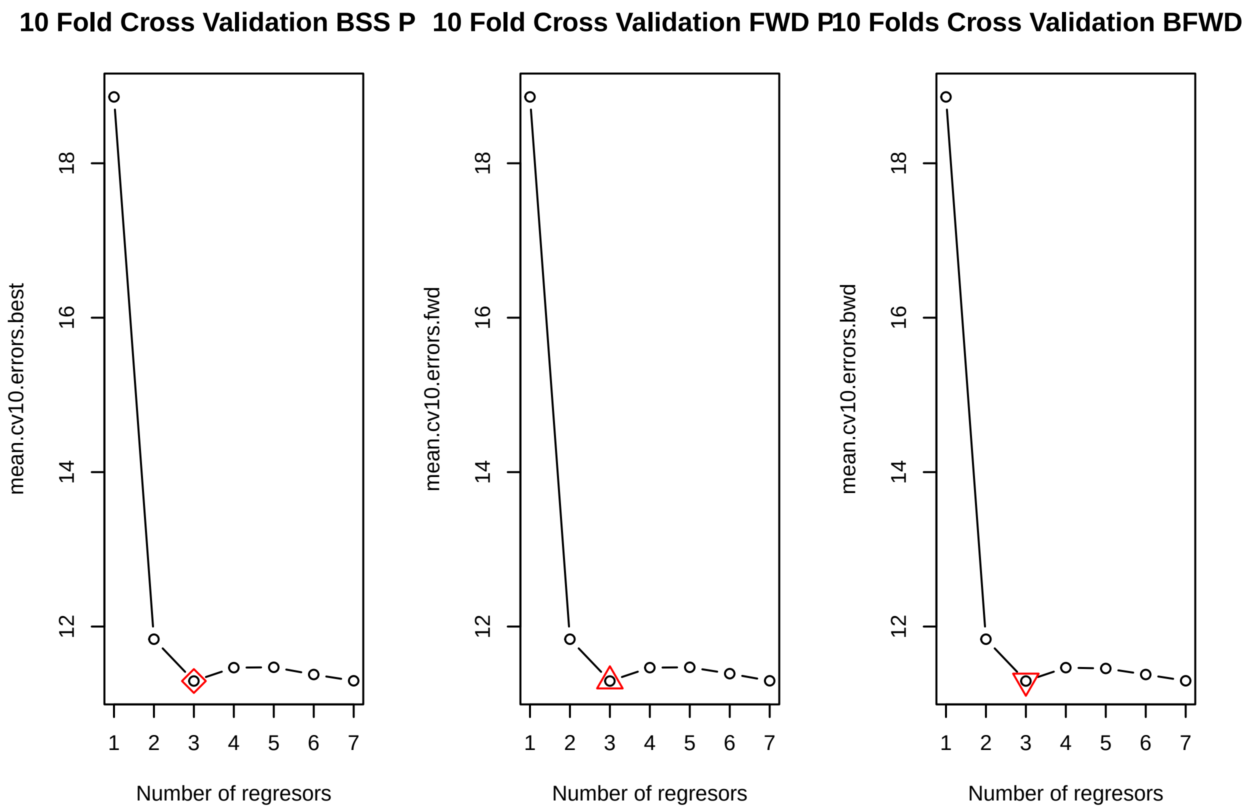
```
mean.cv10.errors.bwd = apply(cv10.errors.bwd, 2, mean)
mean.cv10.errors.bwd
```

```
##        1        2        3        4        5        6        7
## 18.85951 11.83614 11.29446 11.46775 11.45709 11.37871 11.29750
```

```
hBWD = which.min(mean.cv10.errors.bwd)
hBWD
```

```
## 3
## 3
```

```r
plot(mean.cv10.errors.bwd, xlab = "Number of regresors", main = "10 Folds Cross Validation BFWD Plot", 
points(hFWD, mean.cv10.errors.bwd[hBWD], col = "red", cex = 2, pch = 25 )
```




```r
par(mfrow = c(1,1))
```

For computing the 10 Folds Cross Validation method, firstly I set the seed in order to make the results reproducible, I split the set with the function *sample()* to obtain a random permutation of the element of the vector, I created a matrix to store the MSEs, the rows indicate the number of folds, the columns indicate the number of model dimension. In order to get the MSEs, I created a function because the syntax *predict()* does not apply to the syntax *regsubsets*, this function works in general for every selection method we want to apply. To fill the matrix, I used a nested for cycle. The 10 Folds Cross validation method was computed for each of the selection methods just computed in the previous points. Then, by the function *apply()*, I computed the mean along the columns in order to obtain the *Average Test Error Rate* for each model with different numbers of regressors, to choose the best number of regressors to include in the model, the optimal number that gets the lowest MSE. I reported also the plots for the BSS, FWS, BWS, all the three methods deliver the same results, that is, the best number of regressors is 3.

Question 7

In this exercise I want to predict the number of applications received using the remaining variables in the College data set.

point a)

I set the seed and randomly split the data set with the function *sample*, creating a vector of **TRUE** and **FALSE** in order to obtain the train and test (or validation) set.

```
library(ISLR2)
attach(College)
set.seed(1)
train = sample(c(TRUE, FALSE), nrow(College), replace = TRUE)
test = !train
```

point b)

I fit a linear model based only on the train set, using the classical syntax for fitting the linear model, using *data = College[train,]*, to get only the training data, than with model.matrix we create a matrix with all the regressors from test set. I store the betas from the regression on the train set and multiply them by the matrix created before to obtain the prediction on the test set, and at the end I take the mean of the difference of the y of the test set minus the prediction squared, obtaining the validation error:

```
library(ISLR2)
attach(College)
```

```
## I seguenti oggetti sono mascherati da College (pos = 3):
##
##     Accept, Apps, Books, Enroll, Expend, F.Undergrad, Grad.Rate,
##     Outstate, P.Undergrad, perc.alumni, Personal, PhD, Private,
##     Room.Board, S.F.Ratio, Terminal, Top10perc, Top25perc
```

```
set.seed(1)
train = sample(c(TRUE, FALSE), nrow(College), replace = TRUE)
test = !train
reg.lm = lm(Apps~., data = College[train,])
test.mat = model.matrix(Apps~., data = College[test,])
coefi = coef(reg.lm)
pred = test.mat[,names(coefi)]%*%coefi
val.error=mean((College$Apps[test]-pred)^2)
val.error
```

```
## [1] 984743.1
```

point c)

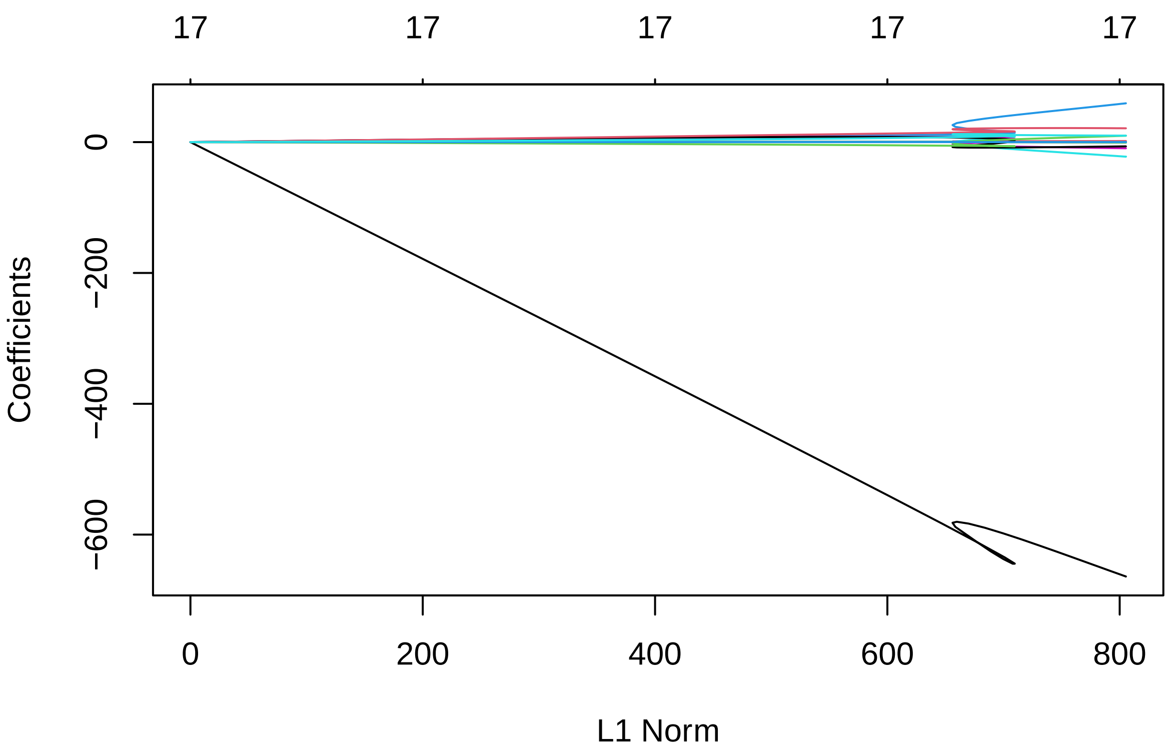
To obtain the rigde regression, first I need to load the library glmnet, to create the regressor matrix without the intercept, and the grid for the possible values of lambda for the shrinkage term in the minimization problem. Then, we are ready to fit the model on the training set:

```
set.seed(1)
library(glmnet)
```

```
## Caricamento del pacchetto richiesto: Matrix
```

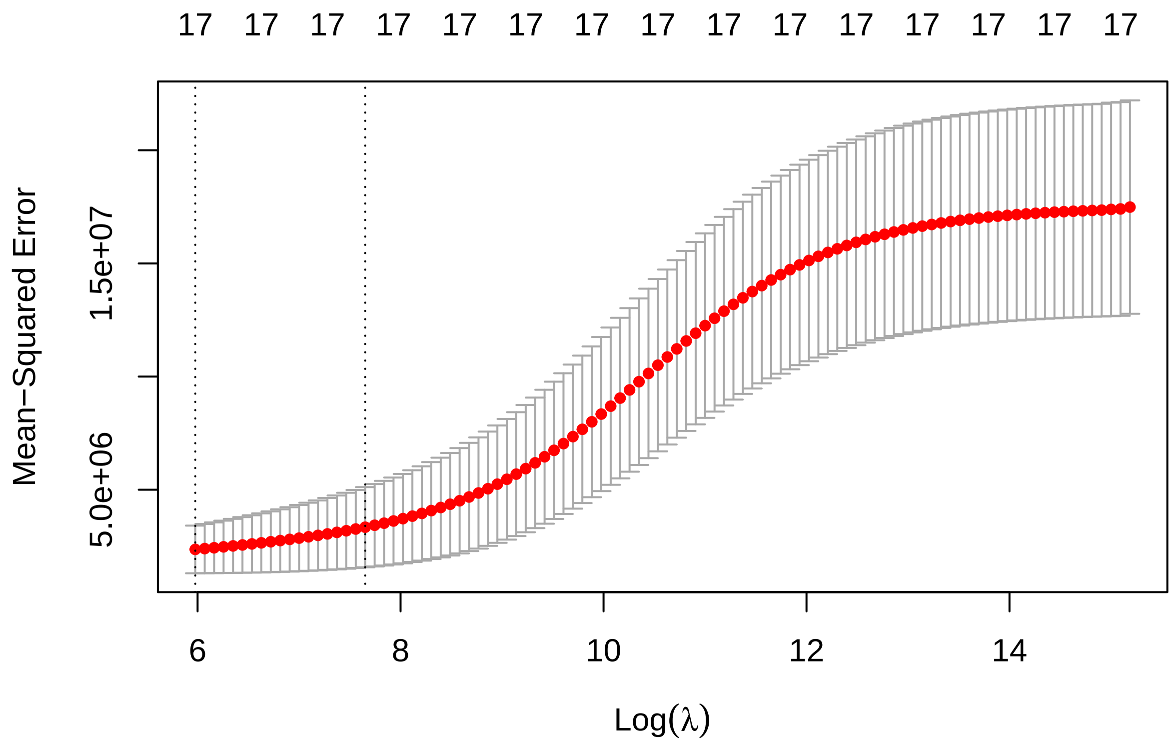
```
## Loaded glmnet 4.1-4
```

```
x = model.matrix(Apps~., data = College)[,-1]
y=College$Apps
train = sample(c(TRUE, FALSE), nrow(College), replace = TRUE)
test = !train
y.test=y[test]
grid = 10^seq(10, -2, length = 100)
ridge.mod = glmnet(x[train,], y[train], alpha = 0,lambda = grid, thresh = 1e-12)
plot(ridge.mod)
```



By the Cross Validation Approach we can estimate the best value of lambda for minimizing the MSE with the function cv.glmnet:

```
cv.outRR = cv.glmnet(x[train,], y[train], alpha = 0)
plot(cv.outRR)
```




```
bestlambdaRR = cv.outRR$lambda.min
bestlambdaRR
```

```
## [1] 394.2365
```

```
ridge.pred = predict(ridge.mod, s = bestlambdaRR, alpha = 0, newx = x[test,])
RRtestMSE = mean((ridge.pred-y.test)^2)
RRtestMSE
```

```
## [1] 941129.7
```

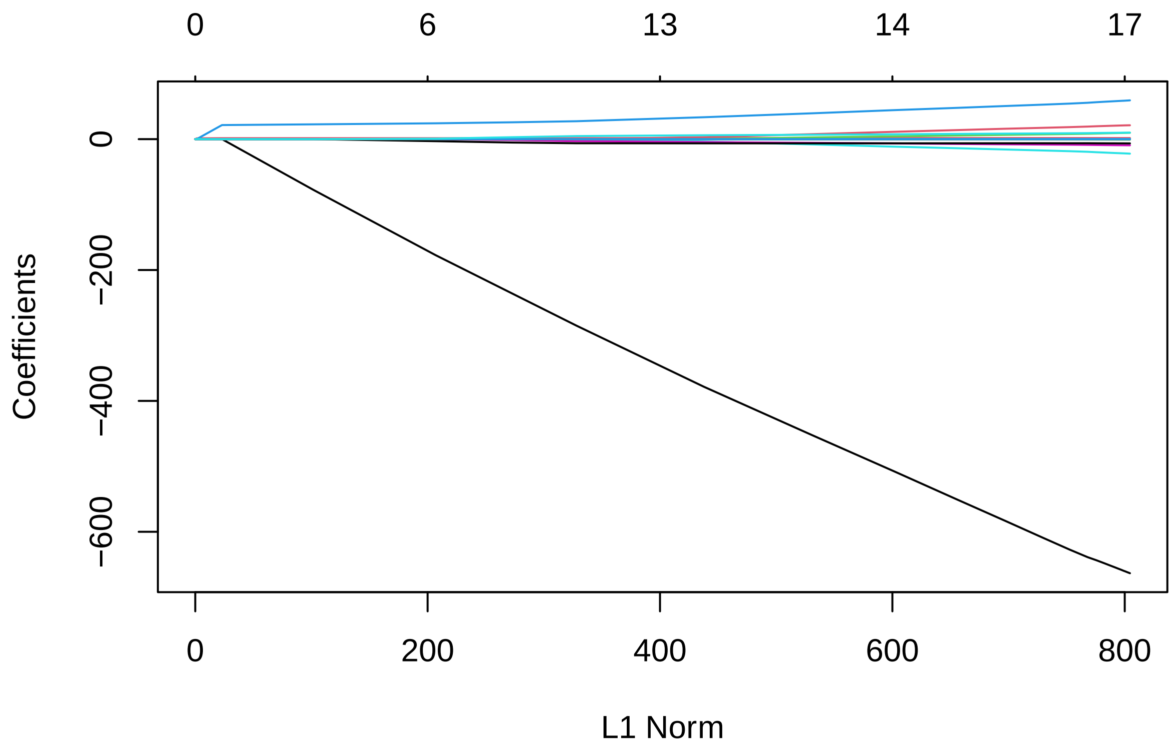
The test error obtained is worth 941129.7

point d)

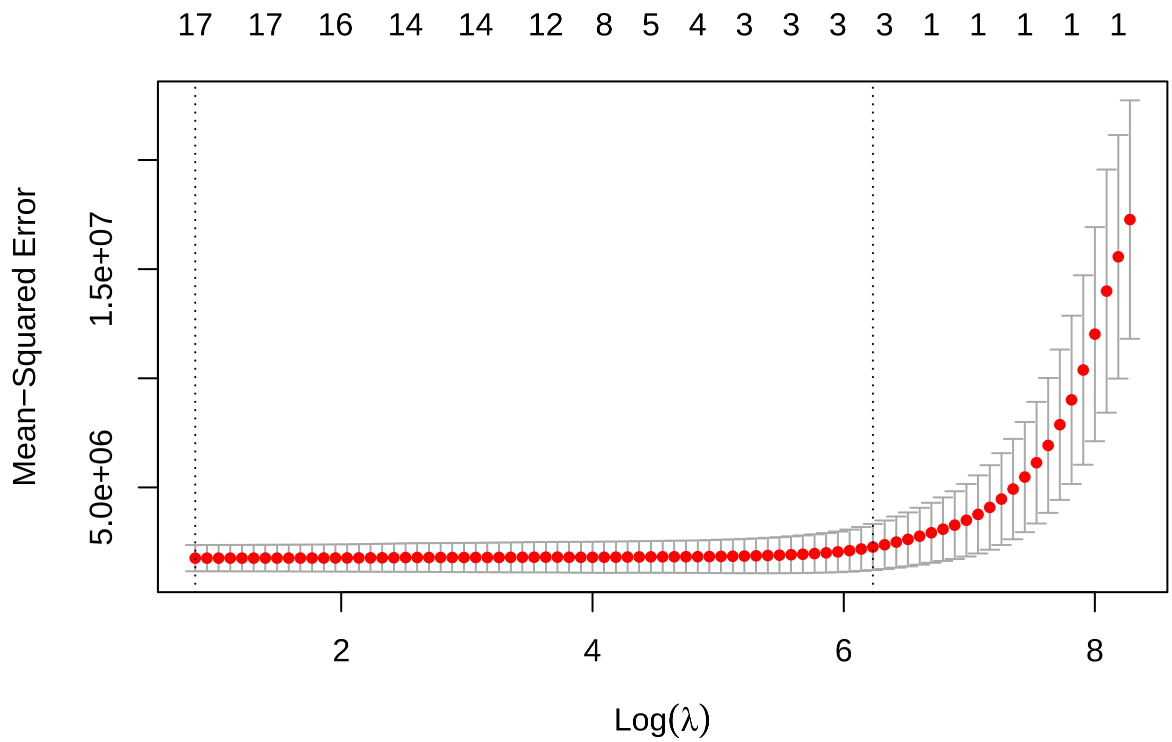
We proceed exactly like in the previous point, but now I set *alpha = 1* to perform the Lasso. But now I report the names and the number of the non zero values:

```
lasso.mod = glmnet(x[train,], y[train], alpha = 1, lambda = grid)
plot(lasso.mod)
```

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm): si
## riduce a valori unici di 'x'
```




```
cv.outLA = cv.glmnet(x[train,], y[train], alpha = 1)
plot(cv.outLA)
```




```
bestlamnbdaLA = cv.outLA$lambda.min
bestlamnbdaLA
```

```
## [1] 2.309051
```

```
lasso.pred = predict(lasso.mod, s = bestlamnbdaLA, alpha = 1, newx = x[test,])
LAtestMSE = mean((lasso.pred-y.test)^2)
LAtestMSE
```

```
## [1] 978962.6
```

```
outLA = glmnet(x, y, alpha = 1, lambda = grid)
lasso.coef = predict(outLA, type = "coefficients", s = bestlamnbdaLA)
lasso.coef
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) -471.62236610
## PrivateYes  -491.05286308
## Accept         1.56921353
## Enroll        -0.75280819
## Top10perc     48.00777736
## Top25perc    -12.75431948
## F.Undergrad    0.04085270
```

```
## P.Undergrad    0.04400650
## Outstate      -0.08302818
## Room.Board     0.14928961
## Books          0.01464639
## Personal       0.02882039
## PhD           -8.38320965
## Terminal      -3.25388115
## S.F.Ratio     14.46488381
## perc.alumni   -0.04955865
## Expend         0.07705012
## Grad.Rate      8.25692656
```

```
lasso.coef[lasso.coef!=0]
```

```
## <sparse>[ <logic> ] : .M.sub.i.logical() potrebbe essere inefficiente
```

```
##  [1] -471.62236610 -491.05286308    1.56921353   -0.75280819   48.00777736
##  [6]  -12.75431948    0.04085270    0.04400650   -0.08302818    0.14928961
## [11]    0.01464639    0.02882039   -8.38320965   -3.25388115   14.46488381
## [16]   -0.04955865    0.07705012    8.25692656
```

So, the names of the coefficients the are different from zero are: Enroll, Top25perc, Undergrad, P.Undergrad, Books, Personal, S.F.Ratio, perc.alumni. The MSE associated at the optimal lambda is worth 987912.6

point e)

Since the MSE for the linear model is worth 984743.1, the one for the Ridge regression is worth 941129.7 and 987912.6 for the Lasso, I state that Ridge regression is the preferred model, since the predicted values are the closest ones to the observed ones.

Question 8

point a)

I fit a logistic regression to predict default using income and balance, splitting the sample with the Validation Approach method.

i)

```
library(ISLR2)
n = nrow(Default)
attach(Default)
set.seed(1)
train = sample(c(TRUE, FALSE), n, replace = T)
test = !train
default.test = default[test]
```

ii)

Now we fit the model only on the training set, using the function glm (general linear model):

```
glm.fit = glm(default ~ income+ balance, data = Default[train,], family = binomial(link = "logit"))
```

iii)

Now I take the predicted probabilities with the function *predict()*, and create a vector with the same length of the test set, where I'll store *No* and *Yes* according to the predicted probabilities: if the predicted probability is <0.5 *No* remains *No*, while if it is > 0.5. I substitute *No* with *Yes*. At the end I create the confusion matrix, that is the matrix where are reported the posterior probabilities, classified according the previous probability 0.5:

```
glm.probs = predict(glm.fit, Default[test,], type = "response")
glm.pred = rep("No", nrow(Default[test,]))
glm.pred[glm.probs > 0.5] = "Yes"
confmat = table(glm.pred, default.test)
confmat
```

```
##          default.test
## glm.pred   No  Yes
##      No  4840  111
##      Yes   22   43
```

iv)

I compute the validation set error:

```
mean(glm.pred != default.test)
```

```
## [1] 0.02651515
```

The validation error set is worth 0.02651515.

point b)

I set other three seeds and repeat this process three times:

```
set.seed(2)
n = nrow(Default)
train = sample(c(TRUE, FALSE), n, replace = T)
test = !train
default.test = default[test]
glm.fit = glm(default ~ income+ balance, data = Default[train,], family = binomial(link = "logit"))
glm.probs = predict(glm.fit, Default[test,], type = "response")
glm.pred = rep("No", nrow(Default[test,]))
glm.pred[glm.probs > 0.5] = "Yes"
confmat = table(glm.pred, default.test)
confmat
```

```
##         default.test
## glm.pred   No  Yes
##      No  4780  145
##      Yes   10   42
```

```r
mean(glm.pred != default.test)
```

```
## [1] 0.03114326
```

```r
set.seed(3)
n = nrow(Default)
train = sample(c(TRUE, FALSE), n, replace = T)
test = !train
default.test = default[test]
glm.fit = glm(default ~ income+ balance, data = Default[train,], family = binomial(link = "logit"))
glm.probs = predict(glm.fit, Default[test,], type = "response")
glm.pred = rep("No", nrow(Default[test,]))
glm.pred[glm.probs > 0.5] = "Yes"
confmat = table(glm.pred, default.test)
confmat
```

```
##         default.test
## glm.pred   No  Yes
##      No  4887   94
##      Yes   21   45
```

```r
mean(glm.pred != default.test)
```

```
## [1] 0.02278581
```

```r
set.seed(4)
n = nrow(Default)
train = sample(c(TRUE, FALSE), n, replace = T)
test = !train
default.test = default[test]
glm.fit = glm(default ~ income+ balance, data = Default[train,], family = binomial(link = "logit"))
glm.probs = predict(glm.fit, Default[test,], type = "response")
glm.pred = rep("No", nrow(Default[test,]))
glm.pred[glm.probs > 0.5] = "Yes"
confmat = table(glm.pred, default.test)
confmat
```

```
##         default.test
## glm.pred   No  Yes
##      No  4847  107
##      Yes   25   58
```

```r
mean(glm.pred != default.test)
```

```
## [1] 0.02620608
```

As we can see, the validation set error is very low in each re sampling over different seeds, so our results are very coherent each other, concluding that the quality of the model in predicting the correct realization is very high independently the different random split.