Applied Statistical Modelling - Assignment 1

Chiara Torri (1066761)

2023-01-12

The aim of this assignment is to study the relationship between the total number of violent crimes per 100K population and a series of 122 covariates that describe several features of the communities under analysis.

In particular, the regression to study is the following:

$log(y_i) = \mathbf{x}_i^T \beta + \epsilon_i$

where ϵ_i are zero mean i.i.d. variables.

Question 1

In this first section of the assignment, I am going to study the relationship mentioned above by computing the OLS estimates.

First, I load the data and I define the response vector and the covariates matrix. Then, since the response in the model is reported in logarithmic scale, I compute the natural logarithm of y.

```r
# Load the data and define response vector and covariates matrix
Dati <- read.table("crime.txt",header = T,sep=";")
y <- Dati$y
Dati$y <- NULL
X <- Dati
X=as.matrix(X)

# Log transformation of y
log_y=log(y)
```

I now estimate the coefficients of the model, that will be presented in a summary table together with their confidence intervals, test statistics and p-values.

The OLS coefficients are computed by minimizing the residual sum of squares:

$RSS = \sum_{i=1}^{n} e_i^2 = \mathbf{e}^T \mathbf{e}$

where $\mathbf{e} = \mathbf{Y} - \mathbf{X}\hat{\beta}$.

It is possible to prove that the values that minimize the RSS are:

$\hat{\beta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$

```r
# Compute XtX
X=model.matrix(log_y~X)
XtX=t(X)%*%X
# Invert
XtXm1= solve(XtX)
# Apply the formula of the beta
beta_hat=XtXm1%*%t(X)%*%log_y
```

Next, I compute the 95% confidence intervals for β_j, which is computed as:

$[\hat{\beta}_j - t_{\frac{\alpha}{2},n-p}\hat{SE}, \hat{\beta}_j + t_{\frac{\alpha}{2},n-p}\hat{SE}]$

To apply this formula I first need to compute the standard errors of the coefficients:

$SE = \sqrt{\hat{\sigma}^2(\mathbf{X}^T\mathbf{X})^{-1}_{jj}}$

To compute the SE, it is necessary to compute:

$\hat{\sigma}^2 = \frac{1}{n-p}RSS = \frac{1}{n-p}\sum_{i=1}^{N} e_i^2$

Finally, to compute the residuals e, I need to compute first the fitted values:

$\hat{\mathbf{y}} = \mathbf{H}\mathbf{y}$

where: $\mathbf{H} = \mathbf{X}(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$

```r
### 95% confidence intervals
n=length(y)
p=length(X[1,])

# Hat matrix
H=X%*%XtXm1%*%t(X)

# Fitted values and residuals
log_y_hat=H%*%log_y
e_hat=(diag(1,n)-H)%*%log_y

# RSS and sigma2_hat
RSS=sum((e_hat)^2)
sig2_hat=RSS/(n-p)

# Confidence intervals
SE=c()
l=c()
u=c()

for (j in 1:length(beta_hat)){
  SE[j]=sqrt(sig2_hat*(XtXm1[j,j]))
  l[j]=beta_hat[j]-qt(p=1-0.025, df=n-p)*SE[j]
  u[j]=beta_hat[j]+qt(p=1-0.025, df=n-p)*SE[j]
}

CI95=data.frame(l,u)
```

At this point, I compute the p-value for the problem:

$H_0 : \beta_j = 0$ vs $H_1 : \beta_j \neq 0$.

In order to do this, I exploit the fact that under H_0, $\frac{\hat{\beta_j}}{\hat{SE}} \sim t_{n-p}$.

Therefore, basing on this, I first compute the t-statistics and then the associated p-value.

```
### Test and p-values

t=c()
p_val=c()

for (j in 1:length(beta_hat)){
  t[j]=beta_hat[j]/SE[j]
  p_val[j]=2*pt(-abs(t[j]), df=n-p)
}

test=data.frame(t,p_val)
```

Finally, I can present my result in the following table.

```
OLS=data.frame(beta_hat,CI95,test)
OLS
```

```
##                 beta_hat            l           u           t        p_val
## (Intercept)  1.055641851  -2.096940744  4.208224446  0.65671850 5.114427e-01
## Xx.V6        7.683936494   3.709973204 11.657899783  3.79218087 1.540681e-04
## Xx.V7       -0.452992729  -1.536975352  0.630989894 -0.81959262 4.125528e-01
## Xx.V8        0.284012404  -0.226749936  0.794774745  1.09055498 2.756092e-01
## Xx.V9       -0.254666387  -0.928428214  0.419095440 -0.74130060 4.586042e-01
## Xx.V10       0.200626726  -0.136986890  0.538240342  1.16546158 2.439808e-01
## Xx.V11       0.958364910   0.394777538  1.521952282  3.33502257 8.695915e-04
## Xx.V12       0.160360483  -1.045612747  1.366333712  0.26078873 7.942841e-01
## Xx.V13      -0.041638634  -1.597468249  1.514190981 -0.05248843 9.581451e-01
## Xx.V14      -0.775423846  -2.692458301  1.141610610 -0.79330166 4.277027e-01
## Xx.V15       0.846537043  -0.367755999  2.060830084  1.36726151 1.717076e-01
## Xx.V16      -6.553214161 -10.498270349 -2.608157972 -3.25784408 1.142724e-03
## Xx.V17       0.475071457   0.224833336  0.725309579  3.72335627 2.024093e-04
## Xx.V18      -1.502576877  -3.333371133  0.328217378 -1.60963085 1.076472e-01
## Xx.V19      -0.769965633  -1.875262331  0.335331064 -1.36622301 1.720332e-01
## Xx.V20       0.243793234  -0.021749810  0.509336277  1.80059420 7.192786e-02
## Xx.V21      -1.635260183  -2.327122624 -0.943397741 -4.63549625 3.806960e-06
## Xx.V22      -0.272222202  -1.396367051  0.851922648 -0.47493086 6.348917e-01
## Xx.V23      -0.128028116  -0.614443147  0.358386916 -0.51621128 6.057679e-01
## Xx.V24      -0.276259305  -0.670819107  0.118300497 -1.37319813 1.698553e-01
## Xx.V25       3.064015126   1.588152710  4.539877541  4.07168488 4.861951e-05
## Xx.V26      -1.367898864  -3.150529448  0.414731720 -1.50494893 1.325060e-01
## Xx.V27      -0.064642160  -1.442868078  1.313583758 -0.09198660 9.267185e-01
## Xx.V28      -0.283112476  -0.681411330  0.115186378 -1.39405234 1.634673e-01
## Xx.V29      -0.111368241  -0.315667866  0.092931384 -1.06911129 2.851574e-01
## Xx.V30       0.023400539  -0.185339693  0.232140771  0.21986132 8.260031e-01
## Xx.V31      -0.004508886  -0.014488097  0.005470326 -0.88613958 3.756562e-01
## Xx.V32      -0.034954279  -0.261798796  0.191890238 -0.30220468 7.625296e-01
## Xx.V33       0.185956200  -1.053855917  1.425768317  0.29416020 7.686681e-01
## Xx.V34      -0.457175146  -1.164709161  0.250358869 -1.26725617 2.052214e-01
## Xx.V35      -0.316057274  -0.981166445  0.349051897 -0.93197010 3.514722e-01
## Xx.V36       0.164873786  -0.779064391  1.108811963  0.34256042 7.319677e-01
## Xx.V37       -0.533186073  -1.395833930  0.329461783 -1.21220027 2.255888e-01
## Xx.V38        0.131609301  -0.317598018  0.580816620  0.57460432 5.656280e-01
## Xx.V39        0.809770475  -0.108167758  1.727708709  1.73012550 8.377276e-02
## Xx.V40       -0.255249485  -0.554581023  0.044082054 -1.67240525 9.461163e-02
## Xx.V41       -0.175296717  -0.591990466  0.241397032 -0.82506075 4.094423e-01
## Xx.V42        0.208007956  -0.346741192  0.762757103  0.73538104 4.621996e-01
## Xx.V43        1.151373442   0.260695829  2.042051056  2.53527339 1.131679e-02
## Xx.V44        2.135623673  -0.206125401  4.477372748  1.78860111 7.384082e-02
## Xx.V45        0.882488055   0.138925072  1.626051037  2.32766353 2.003614e-02
## Xx.V46        1.888957176  -1.096622306  4.874536657  1.24085948 2.148132e-01
## Xx.V47       -3.508891909  -8.429364272  1.411580454 -1.39859612 1.620999e-01
## Xx.V48       -1.616929587  -3.317314425  0.083455252 -1.86497499 6.234135e-02
## Xx.V49        0.324638331  -1.265862010  1.915138673  0.40030882 6.889748e-01
## Xx.V50       -2.043343780  -3.595283210 -0.491404350 -2.58223522 9.891642e-03
## Xx.V51        0.168274638  -0.366243337  0.702792614  0.61742680 5.370284e-01
## Xx.V52        0.220395747  -0.302725556  0.743517050  0.82628512 4.087477e-01
## Xx.V53        0.344819184  -0.160375350  0.850013717  1.33863416 1.808524e-01
## Xx.V54       -0.499613201  -1.031272954  0.032046551 -1.84301660 6.548468e-02
## Xx.V55       -1.412641097  -2.399831778 -0.425450416 -2.80646685 5.060682e-03
## Xx.V56        0.445558638   0.002676090  0.888441186  1.97308333 4.863305e-02
## Xx.V57       -0.503174652  -1.228326261  0.221976956 -1.36087761 1.737163e-01
## Xx.V58        0.334131009  -0.212200780  0.880462798  1.19947013 2.304971e-01
## Xx.V59       -0.350830912  -1.218587825  0.516926001 -0.79291913 4.279255e-01
## Xx.V60       -0.579129242  -1.573670984  0.415412500 -1.14204076 2.535832e-01
## Xx.V61        0.723781852   0.002244754  1.445318950  1.96733424 4.929222e-02
## Xx.V62       -1.045141373  -2.283490108  0.193207362 -1.65524088 9.804335e-02
## Xx.V63        1.362793670  -0.896082517  3.621669857  1.18322356 2.368709e-01
## Xx.V64        1.078794213  -1.763991739  3.921580164  0.74425811 4.568138e-01
## Xx.V65       -2.124688640  -4.409343326  0.159966046 -1.82391189 6.832484e-02
## Xx.V66       -0.004514460  -0.741077514  0.732048594 -0.01202057 9.904105e-01
## Xx.V67       -0.828574366  -1.525998781 -0.131149952 -2.33004052 1.990988e-02
## Xx.V68       -0.370038211  -2.508985129  1.768908706 -0.33929362 7.344266e-01
## Xx.V69        0.478078744  -1.791056046  2.747213533  0.41320757 6.795019e-01
## Xx.V70        2.546080202  -0.022140255  5.114300659  1.94432521 5.200595e-02
## Xx.V71       -0.514395552  -2.212923441  1.184132336 -0.59395515 5.526139e-01
## Xx.V72       -0.151683570  -1.074072333  0.770705193 -0.32251778 7.470965e-01
## Xx.V73       -0.786523659  -4.201537047  2.628489729 -0.45169835 6.515387e-01
## Xx.V74       -0.004464652  -0.758401267  0.749471964 -0.01161400 9.907348e-01
## Xx.V75        0.362251830  -0.247165007  0.971668666  1.16580326 2.438426e-01
## Xx.V76        0.109761282  -0.075032739  0.294555303  1.16490466 2.442061e-01
## Xx.V77        0.034852201  -0.603460569  0.673164972  0.10708430 9.147336e-01
## Xx.V78       -0.335246187  -0.640973865 -0.029518510 -2.15059295 3.163610e-02
## Xx.V79        0.136817088  -3.444328306  3.717962483  0.07492858 9.402795e-01
## Xx.V80        0.114806553  -0.083599717  0.313212824  1.13485510 2.565814e-01
## Xx.V81       -0.298605508  -0.558617150 -0.038593866 -2.25234103 2.441622e-02
## Xx.V82       -0.130851920  -0.398010825  0.136306984 -0.96059335 3.368808e-01
## Xx.V83        0.050731467  -0.296865712  0.398328647  0.28623999 7.747260e-01
## Xx.V84       -0.195195547  -0.426229195  0.035838100 -1.65700492 9.768614e-02
## Xx.V85       -1.703757443  -3.769622965  0.362108080 -1.61746483 1.059465e-01
## Xx.V86        1.012310740  -1.982213181  4.006834661  0.66300253 5.074106e-01
## Xx.V87        0.271536000  -1.235216226  1.778288226  0.35343919 7.237990e-01
## Xx.V88       -1.096001735  -1.791685028 -0.400318441 -3.08978880 2.032470e-03
## Xx.V89        0.209678239  -1.667149208  2.086505687  0.21910795 8.265899e-01
## Xx.V90        0.621479198  -0.289039990  1.531998386  1.33864870 1.808477e-01
## Xx.V91        0.320988996  -1.027300696  1.669278688  0.46691310 6.406164e-01
## Xx.V92        0.189492776  -0.141131699  0.520117250  1.12405299 2.611347e-01
## Xx.V93       -0.154728929  -0.495767806  0.186309947 -0.88980893 3.736829e-01
## Xx.V94       -0.219479087  -0.461091032  0.022132859 -1.78157401 7.498089e-02
## Xx.V95        0.121617103  -0.438868424  0.682102629  0.42555861 6.704785e-01
## Xx.V96        0.271735992  -0.136034178  0.679506163  1.30695562 1.913884e-01
## Xx.V97        1.039941328   0.104688052  1.975194604  2.18076429 2.932504e-02
## Xx.V98        0.025905016  -0.412672413  0.464482445  0.11584218 9.077901e-01
## Xx.V99        0.073071881  -0.537465658  0.683609420  0.23472916 8.144446e-01
## Xx.V100       0.067310058  -0.352941911  0.487562028  0.31412271 7.534629e-01
## Xx.V101       0.042209023  -0.392886624  0.477304670  0.19026096 8.491253e-01
## Xx.V102      -2.332931154  -5.630836863  0.964974555 -1.38737165 1.654937e-01
## Xx.V103     -12.203295462 -66.954726394 42.548135470 -0.43713017 6.620674e-01
## Xx.V104      -1.494049739  -4.086467525  1.098368048 -1.13028819 2.584997e-01
## Xx.V105      -0.914602601  -2.674957876  0.845752674 -1.01896961 3.083491e-01
## Xx.V106      -0.183930677  -0.854937028  0.487075673 -0.53759679 5.909194e-01
## Xx.V107       0.044302123  -0.577536934  0.666141180  0.13972550 8.888919e-01
## Xx.V108       0.101306710  -0.260439725  0.463053145  0.54924116 5.829055e-01
## Xx.V109      13.175406742 -41.294262649 67.645076133  0.47439316 6.352749e-01
## Xx.V110      -0.179220745  -0.522989769  0.164548279 -1.02247021 3.066906e-01
## Xx.V111       0.116900125  -0.242982166  0.476782415  0.63706482 5.241606e-01
## Xx.V112      -0.457381635  -1.215579560  0.300816290 -1.18311036 2.369157e-01
## Xx.V113      -0.549376165  -1.392258483  0.293506152 -1.27829767 2.013030e-01
## Xx.V114       0.106730945  -0.132592042  0.346053932  0.87465154 3.818758e-01
## Xx.V115       0.687392886  -0.339886895  1.714672667  1.31233714 1.895674e-01
## Xx.V116       0.005734790  -0.790619704  0.802089283  0.01412343 9.887330e-01
## Xx.V117       0.013120360  -0.146304132  0.172544851  0.16140606 8.717910e-01
## Xx.V118       0.008354500  -0.141815336  0.158524337  0.10911059 9.131265e-01
## Xx.V119      -0.177407089  -0.610059676  0.255245498 -0.80419393 4.213871e-01
## Xx.V120       0.012776876  -0.265215624  0.290769376  0.09014066 9.281851e-01
## Xx.V121      -0.275496731  -0.491384323 -0.059609139 -2.50275245 1.240785e-02
## Xx.V122       0.256278772  -0.122838876  0.635396419  1.32576870 1.850782e-01
## Xx.V123       1.159534011  -0.233481543  2.552549566  1.63251302 1.027397e-01
## Xx.V124       0.067002360  -0.250274818  0.384279538  0.41417168 6.787959e-01
## Xx.V125       0.015332844  -0.067157778  0.097823466  0.36454174 7.154947e-01
## Xx.V126       0.094677778  -0.074543431  0.263898987  1.09729245 2.726548e-01
## Xx.V127      -0.277547913  -0.885993677  0.330897851 -0.89463363 3.710980e-01
```

Looking at this table it is possible to select the coefficients β_j for which I cannot reject the null hypothesis $H_0 : \beta_j = 0$. These coefficients are called "not statistically significant".

```
### Parameters for which i cannot reject H0
not_significant=data.frame(OLS[which(test$p_val>0.05),c(1,5)])
cat("The parameters for which I can not reject H0: beta_j=0:")
```

```
## The parameters for which I can not reject H0: beta_j=0:
```

```
not_significant
```

```
##                 beta_hat      p_val
## (Intercept)   1.055641851 0.51144270
## Xx.V7        -0.452992729 0.41255277
## Xx.V8         0.284012404 0.27560917
## Xx.V9        -0.254666387 0.45860423
## Xx.V10        0.200626726 0.24398076
## Xx.V12        0.160360483 0.79428414
## Xx.V13       -0.041638634 0.95814512
## Xx.V14       -0.775423846 0.42770268
## Xx.V15        0.846537043 0.17170758
## Xx.V18       -1.502576877 0.10764716
## Xx.V19       -0.769965633 0.17203315
## Xx.V20        0.243793234 0.07192786
## Xx.V22       -0.272222202 0.63489168
## Xx.V23       -0.128028116 0.60576785
## Xx.V24       -0.276259305 0.16985530
## Xx.V26       -1.367898864 0.13250599
## Xx.V27       -0.064642160 0.92671853
## Xx.V28       -0.283112476 0.16346733
## Xx.V29       -0.111368241 0.28515743
## Xx.V30        0.023400539 0.82600315
## Xx.V31       -0.004508886 0.37565620
## Xx.V32       -0.034954279 0.76252962
## Xx.V33        0.185956200 0.76866812
## Xx.V34       -0.457175146 0.20522141
## Xx.V35       -0.316057274 0.35147222
## Xx.V36        0.164873786 0.73196766
## Xx.V37       -0.533186073 0.22558878
## Xx.V38        0.131609301 0.56562801
## Xx.V39        0.809770475 0.08377276
## Xx.V40       -0.255249485 0.09461163
## Xx.V41       -0.175296717 0.40944228
## Xx.V42        0.208007956 0.46219963
## Xx.V44        2.135623673 0.07384082
## Xx.V46        1.888957176 0.21481323
## Xx.V47       -3.508891909 0.16209986
## Xx.V48       -1.616929587 0.06234135
## Xx.V49        0.324638331 0.68897477
## Xx.V51        0.168274638 0.53702838
## Xx.V52        0.220395747 0.40874773
## Xx.V53        0.344819184 0.18085243
## Xx.V54       -0.499613201 0.06548468
## Xx.V57       -0.503174652 0.17371626
## Xx.V58        0.334131009 0.23049706
## Xx.V59       -0.350830912 0.42792547
## Xx.V60       -0.579129242 0.25358321
## Xx.V62       -1.045141373 0.09804335
## Xx.V63        1.362793670 0.23687088
## Xx.V64        1.078794213 0.45681380
## Xx.V65       -2.124688640 0.06832484
## Xx.V66       -0.004514460 0.99041049
## Xx.V68       -0.370038211 0.73442663
## Xx.V69        0.478078744 0.67950189
## Xx.V70        2.546080202 0.05200595
## Xx.V71       -0.514395552 0.55261392
## Xx.V72       -0.151683570 0.74709650
## Xx.V73       -0.786523659 0.65153866
## Xx.V74       -0.004464652 0.99073481
## Xx.V75         0.362251830 0.24384259
## Xx.V76         0.109761282 0.24420608
## Xx.V77         0.034852201 0.91473358
## Xx.V79         0.136817088 0.94027955
## Xx.V80         0.114806553 0.25658140
## Xx.V82        -0.130851920 0.33688080
## Xx.V83         0.050731467 0.77472601
## Xx.V84        -0.195195547 0.09768614
## Xx.V85        -1.703757443 0.10594655
## Xx.V86         1.012310740 0.50741060
## Xx.V87         0.271536000 0.72379903
## Xx.V89         0.209678239 0.82658985
## Xx.V90         0.621479198 0.18084769
## Xx.V91         0.320988996 0.64061642
## Xx.V92         0.189492776 0.26113474
## Xx.V93        -0.154728929 0.37368288
## Xx.V94        -0.219479087 0.07498089
## Xx.V95         0.121617103 0.67047845
## Xx.V96         0.271735992 0.19138839
## Xx.V98         0.025905016 0.90779005
## Xx.V99         0.073071881 0.81444464
## Xx.V100        0.067310058 0.75346286
## Xx.V101        0.042209023 0.84912528
## Xx.V102       -2.332931154 0.16549369
## Xx.V103      -12.203295462 0.66206740
## Xx.V104       -1.494049739 0.25849968
## Xx.V105       -0.914602601 0.30834913
## Xx.V106       -0.183930677 0.59091938
## Xx.V107        0.044302123 0.88889192
## Xx.V108        0.101306710 0.58290550
## Xx.V109       13.175406742 0.63527493
## Xx.V110       -0.179220745 0.30669059
## Xx.V111        0.116900125 0.52416056
## Xx.V112       -0.457381635 0.23691572
## Xx.V113       -0.549376165 0.20130304
## Xx.V114        0.106730945 0.38187579
## Xx.V115        0.687392886 0.18956737
## Xx.V116        0.005734790 0.98873301
## Xx.V117        0.013120360 0.87179102
## Xx.V118        0.008354500 0.91312648
## Xx.V119       -0.177407089 0.42138715
## Xx.V120        0.012776876 0.92818509
## Xx.V122        0.256278772 0.18507816
## Xx.V123        1.159534011 0.10273973
## Xx.V124        0.067002360 0.67879586
## Xx.V125        0.015332844 0.71549470
## Xx.V126        0.094677778 0.27265477
## Xx.V127       -0.277547913 0.37109804
```

```
dim(not_significant)
```

```
## [1] 105   2
```

There are 105 non significant coefficients.

In this last part of Question 1, I will verify the deviance decomposition for this model and its R^2.

The deviance decomposition shows that the variability of the data can be split between the variability of the fitted values and the variability of the residuals.

- The variability of the data is proportional to: $TSS = \sum_{i=1}^{n} (y_i - \bar{y})^2$
- The variability of the predicted values is given by: $ESS = \sum_{i=1}^{n} (\hat{y}_i - \bar{y})^2$
- The variability of the residuals is given by: $RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$

Therefore, the deviance decomposition can be represented as TSS=ESS+RSS.

```
### Deviance decomposition
bar_log_y=mean(log_y)

TSS=sum((log_y-bar_log_y)^2)
ESS=sum((log_y_hat-bar_log_y)^2)
cat("TSS:",TSS,"ESS:",ESS,"RSS:",RSS)
```

```
## TSS: 1992.389 ESS: 1392.772 RSS: 599.6163
```

```
cat("The sum of ESS and RSS is:", ESS+RSS)
```

```
## The sum of ESS and RSS is: 1992.389
```

As expected, the sum of ESS and RSS is equal to TSS.

This concept can be used to compute a coefficient that assess the goodness of fit of the model, by computing the fraction of the total variance that is explained by the model:
$R^2 = \frac{ESS}{TSS} = 1 - \frac{RSS}{TSS}$

```
# R2
R2=ESS/TSS
cat("R2=",R2)
```

```
## R2= 0.6990465
```

```
rm(list = ls())
```

The $R^2 = 0.6990465$, meaning that approximately the 70% of the variability of the data is explained by this model.

Question 2

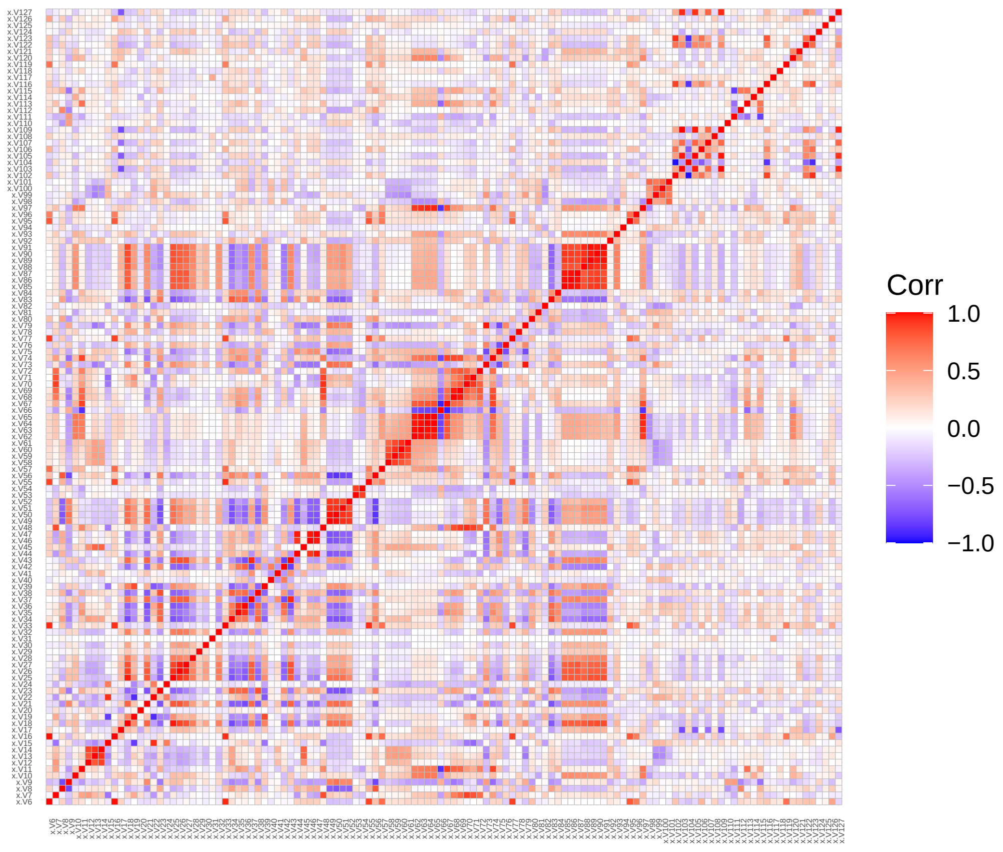
The empirical correlation matrix of the 122 standardized covariates is the following. The darker red or blue cells represent an higher positive or negative correlation between the variables that are considered.

```r
Dati <- read.table("crime.txt",header = T,sep=";")
y <- Dati$y
Dati$y <- NULL
X <- Dati
log_y=log(y)

X<-as.matrix(scale(X))
Xcor=cor(X)

library(ggcorrplot)
```

```
## Warning: il pacchetto 'ggcorrplot' è stato creato con R versione 4.2.2
```

```
## Caricamento del pacchetto richiesto: ggplot2
```

```r
ggcorrplot(Xcor, tl.cex=3, tl.srt=90)
```



```r
rm(list = ls())
```

It is possible to notice that there are some areas of the graph that have a very strong red or blue color, meaning that some of the covariates are very correlated. This situation is called imperfect multicollinearity and it can represent a problem since it affects the OLS estimates. In particular, the OLS estimates in presence of multicollinearity may be biased, because it is difficult to assess the effect of a variable keeping all the other constant, if this variable is highly correlated with the others.

One possible solution that performs well also in case of perfect multicollinearity is to carry out a Ridge regression.

Question 3

Best subset selection

The best subset selection consists in an exhaustive search of the best model (according to cross-validated prediction error, Mallow's Cp, AIC, BIC, or adjusted R^2) among all possible models.

It is possible to show that there are 2^p total subset in a model with p candidate predictors. Therefore, in this case, I should perform a best subset selection among 2^{122} sub-models.

```
# There are 2^p models, so in this case
2^122
```

```
## [1] 5.316912e+36
```

In this case, I should test 5.316912e+36 sub-models.

Stepwise selection

Given this excessive number of models, it is more appropriate to perform a stepwise selection. There are two types of stepwise selection:

- forward: start with a model containing no predictors and add, one at a time, the predictor that improves the model the most;
- backward: start with a model that contains all the predictors and remove, one at a time, the predictor such that the model improves the most.

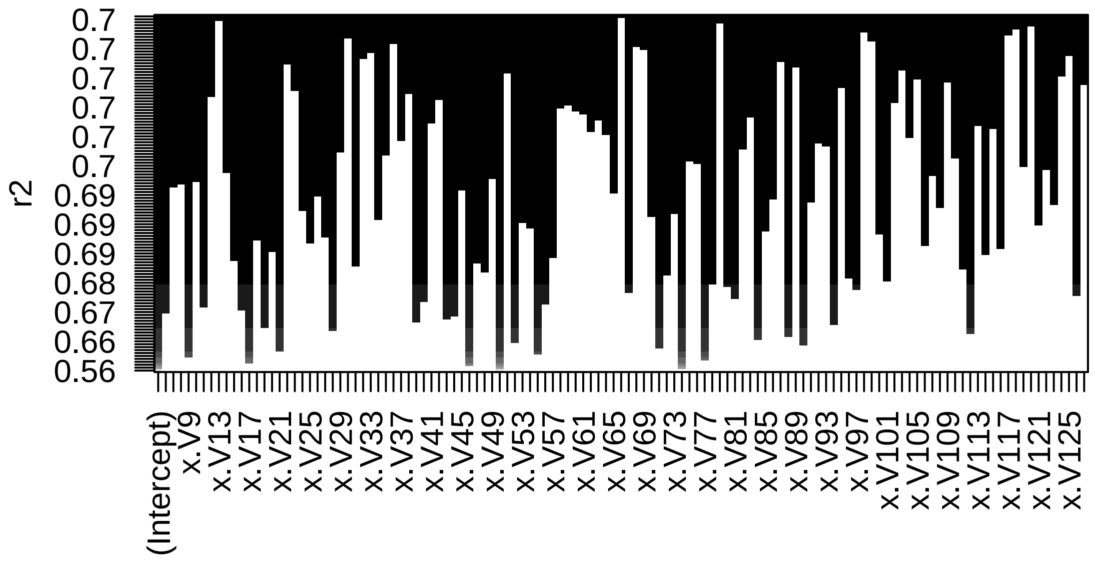
Forward subset selection

I start by performing a forward subset selection. I will compare the results I get using three criteria: the adjusted R^2, the BIC and the Mallow's Cp.

```
# Load the data
Dati <- read.table("crime.txt",header = T,sep=";")
y <- Dati$y
Dati$y <- NULL
X <- Dati
X<-as.matrix(scale(X))
log_y=log(y)

Dati=data.frame(X,log_y)

## Forward selection
library(leaps)
```

```
## Warning: il pacchetto 'leaps' è stato creato con R versione 4.2.2
```

```
regfit.fwd = regsubsets(log_y ~ . , data=Dati, method = "forward", nvmax=122)

reg_summary_fwd = summary(regfit.fwd)

# Adj R2
plot(regfit.fwd, scale = "r2")
```

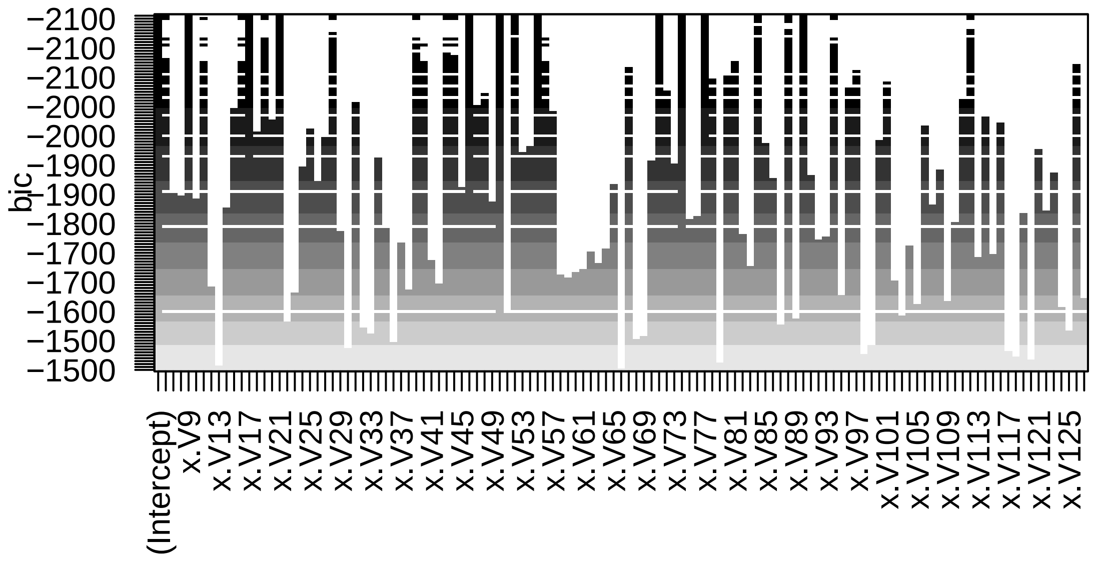


```
adjr2_max=which.max(reg_summary_fwd$adjr2)
cat("The best model using the adjusted R2 criterion is the one with",
    adjr2_max,"covariates")
```

```
## The best model using the adjusted R2 criterion is the one with 71 covariates
```

According to the adjusted R^2 criterion, the best model is the one with 71 covariates.

```
# BIC
plot(regfit.fwd, scale = "bic")
```

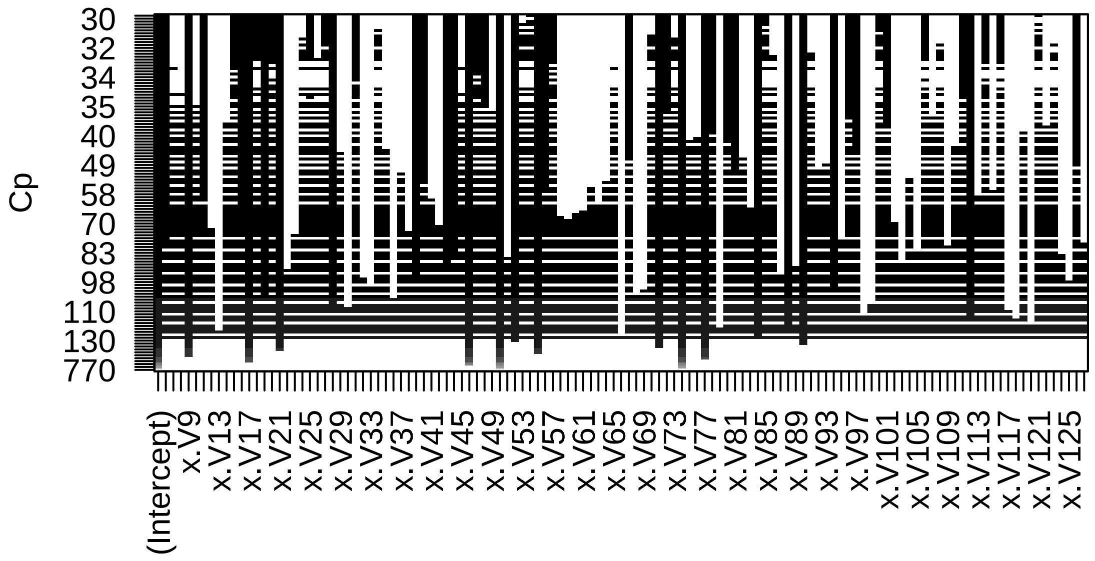


```
bic_min=which.min(reg_summary_fwd$bic)
cat("The best model using the BIC criterion is the one with", bic_min,
    "covariates")
```

```
## The best model using the BIC criterion is the one with 22 covariates
```

According to the BIC criterion, the best model is the one with 22 covariates.

```
# Cp
plot(regfit.fwd, scale = "Cp")
```



```r
Cp_min=which.min(reg_summary_fwd$cp)
cat("The best model using the Mallow's Cp criterion is the one with",
    Cp_min,"covariates")
```

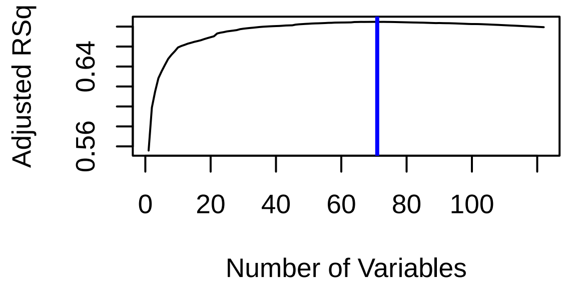
```
## The best model using the Mallow’s Cp criterion is the one with 51 covariates
```

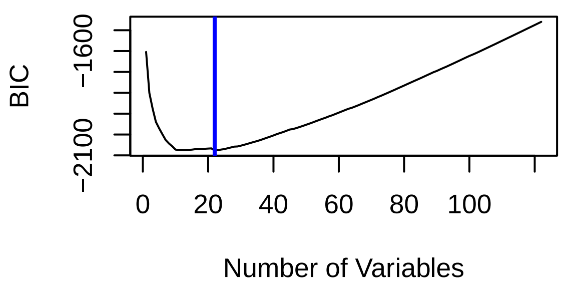
According to the Mallow’s Cp criterion, the best model is the one with 51 covariates.

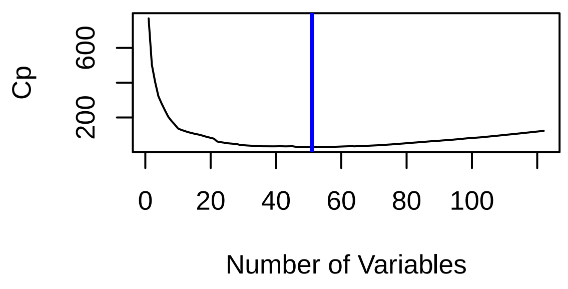
The following plot shows how the three criteria change when a different number of variables is considered. The blue line represent the selected model.

```r
# Plots
x11()
par(mfrow = c(2,2))

# Adj R2
plot(reg_summary_fwd$adjr2, xlab = "Number of Variables",
     ylab = "Adjusted RSq", type = "l")
abline(v=adjr2_max, col ="blue", lwd=2)

# BIC
plot(reg_summary_fwd$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
abline(v=bic_min, col ="blue", lwd=2)
```

```
# Cp
plot(reg_summary_fwd$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
abline(v=Cp_min, col ="blue", lwd=2)
```







I decided to consider, in particular, the model selected using BIC. In the following table I report the β coefficients of this model.

```
# BIC coefficients
fwd_coeff=data.frame(coef(regfit.fwd, 22))
cat("The coefficients of the model selected using BIC are:")
```

```
## The coefficients of the model selected using BIC are:
```

```
fwd_coeff
```

```
##             coef.regfit.fwd..22.
## (Intercept)          -1.85117452
## x.V6                  0.94464052
## x.V9                 -0.21478187
## x.V16                -0.87251523
## x.V17                 0.19960968
## x.V19                -0.17725292
## x.V21                -0.31980925
## x.V28                -0.05382199
## x.V39                  0.12250190
## x.V43                  0.08387119
## x.V44                  0.11250597
## x.V46                 -0.01052818
## x.V50                 -0.43301270
## x.V52                  0.08643770
## x.V55                 -0.14876470
## x.V71                 -0.03303824
## x.V74                  0.07620782
## x.V77                  0.11059006
## x.V84                 -0.04743033
## x.V88                 -0.11827604
## x.V90                  0.23686850
## x.V94                 -0.04073126
## x.V112                -0.05777908
```

Finally, I compute the adjusted R^2 of this model in order to compare it with other models.

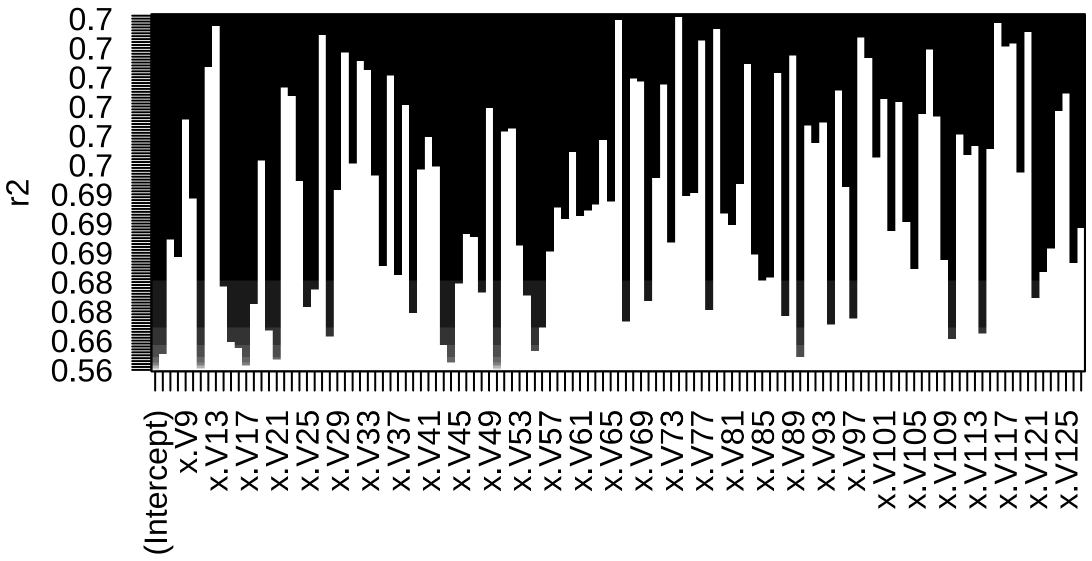
```r
# Adjusted R2 for comparison
cat("The adjusted R2 of the model is:", reg_summary_fwd$adjr2[22])
```

```
## The adjusted R2 of the model is: 0.6731111
```

Backward subset selection

I now perform a backward subset selection, using the same criteria I used for the forward.

```r
# Backward
regfit.back = regsubsets(log_y ~ . , data=Dati, method = "backward", nvmax=122)

reg_summary_back = summary(regfit.back)

par(mfrow=c(1,1))
plot(regfit.back, scale = "r2")
```

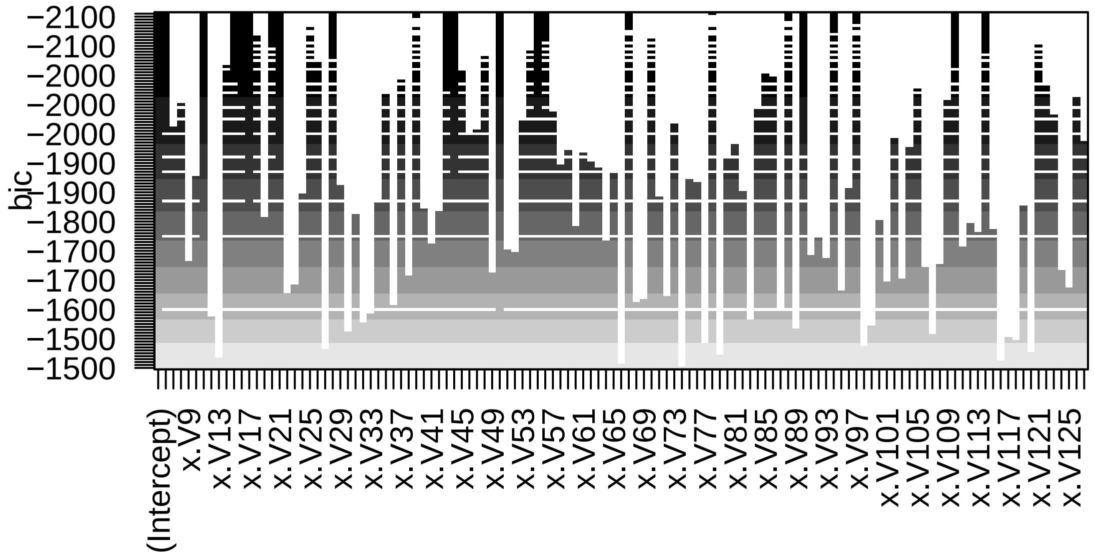


```r
adjr2_max=which.max(reg_summary_back$adjr2)
cat("The best model using the adjusted R2 criterion is the one with",
    adjr2_max,"covariates")
```

```
## The best model using the adjusted R2 criterion is the one with 69 covariates
```

According to the adjusted R^2 criterion, the best model is the one with 69 covariates.

```r
plot(regfit.back, scale = "bic")
```

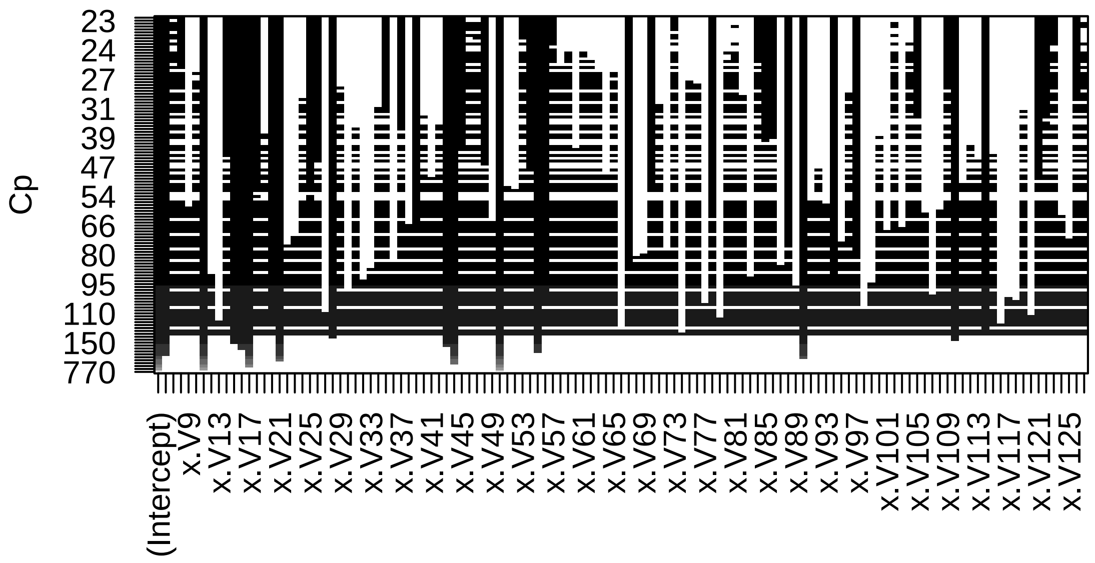


```r
bic_min=which.min(reg_summary_back$bic)
cat("The best model using the BIC criterion is the one with", bic_min,
    "covariates")
```

```
## The best model using the BIC criterion is the one with 22 covariates
```

According to the BIC criterion, the best model is the one with 22 covariates.

```r
plot(regfit.back, scale = "Cp")
```



```
Cp_min=which.min(reg_summary_back$cp)
cat("The best model using the Mallow's Cp criterion is the one with",
    Cp_min,"covariates")
```

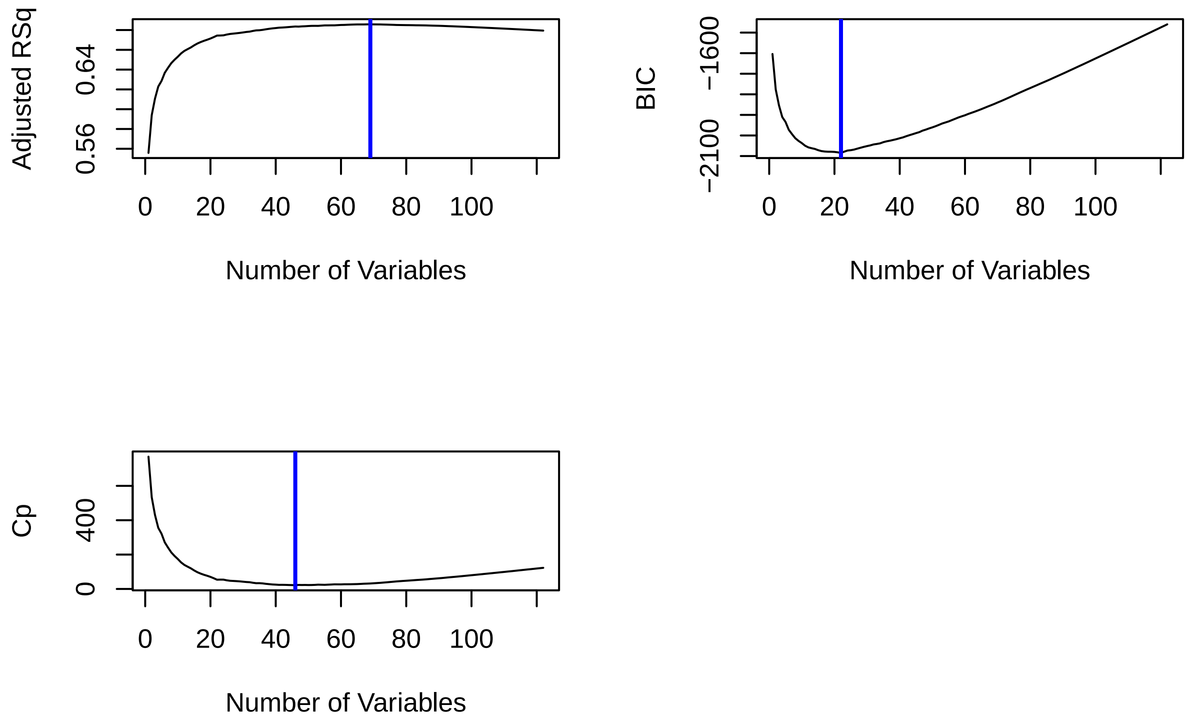
```
## The best model using the Mallow’s Cp criterion is the one with 46 covariates
```

According to the Mallow’s Cp criterion, the best model is the one with 46 covariates.

The following plot shows how the three criteria change when a different number of variables is considered. The blue line represent the selected model.

```
# Plots
x11()
par(mfrow = c(2,2))

# Adj R2
plot(reg_summary_back$adjr2, xlab = "Number of Variables",
     ylab = "Adjusted RSq", type = "l")
abline(v=adjr2_max, col ="blue", lwd=2)

# BIC
plot(reg_summary_back$bic, xlab = "Number of Variables", ylab = "BIC", type = "l")
abline(v=bic_min, col ="blue", lwd=2)
```

```r
# Cp
plot(reg_summary_back$cp, xlab = "Number of Variables", ylab = "Cp", type = "l")
abline(v=Cp_min, col ="blue", lwd=2)
```



In the following table I report the β coefficients of the best model according to BIC.

```r
# BIC coefficients
back_coeff=data.frame(coef(regfit.back, 22))
cat("The coefficients of the model selected using BIC are:")
```

```
## The coefficients of the model selected using BIC are:
```

```r
back_coeff
```

```
##             coef.regfit.back..22.
## (Intercept)           -1.85117452
## x.V6                   1.19654494
## x.V11                  0.18860248
## x.V15                  0.15156898
## x.V16                 -1.03670365
## x.V17                  0.23898404
## x.V20                  0.05384003
## x.V21                 -0.37025281
## x.V28                 -0.06468132
## x.V39                     0.08629684
## x.V43                     0.08005374
## x.V44                     0.10651044
## x.V50                    -0.33223623
## x.V55                    -0.14708435
## x.V56                     0.14511218
## x.V67                    -0.20808005
## x.V78                    -0.05048579
## x.V88                    -0.14317983
## x.V90                     0.22395848
## x.V94                    -0.05743477
## x.V97                     0.13740354
## x.V110                   -0.06976653
## x.V114                    0.06346076
```

Finally, also here I compute the adjusted R^2 of this model in order to compare it with other models.

```
# Adjusted R2 for comparison
cat("The adjusted R2 of the model is:",reg_summary_back$adjr2[22])
```

```
## The adjusted R2 of the model is: 0.6743957
```

```
rm(list = ls())
```

The adjusted R^2 of this model is slightly higher than the one of the forward subset selection model.

My code for stepwise subset selection (forward)

This is my code for the forward stepwise selection. I didn't managed to create a sufficiently efficient code for the backward stepwise selection.

Since I'm implementing a forward stepwise selection, I start from $\mathcal{M}_0$ and I proceed as follows.

First, I define all the M_k models by inserting, one at a time, the variables that most ameliorate the model. In particular, for each of the k iterations:

- I define $\mathcal{M}_k$ as all the possible models that contain M_{k-1} and an additional covariate among the ones that have not been inserted yet ("available").
- For each of the models of $\mathcal{M}_k$, I create the covariates matrix and I compute the adjusted R^2.
- I select the covariate that improves the adjusted R^2 the most and I drop it from the list of "available" covariates.

At the end of this process, I have all the M_k models. I compute the adjusted R^2 and the BIC for each model and I select the models that maximize the adjusted R^2 and minimize the BIC. The resulting models have the same number of covariates of the models I got using the leaps package.

Finally, as I did above, I compute the coefficients of the model chosen using the BIC criterion. Also these correspond to the ones I found with the leaps package.

```r
Dati <- read.table("crime.txt",header = T,sep=";")
y <- Dati$y
Dati$y <- NULL
X <- Dati
X<-as.matrix(scale(X))
log_y=log(y)

Dati=data.frame(X,log_y)

## Forward

p=length(X[1,])

# Available covariates= the ones that have not been inserted yet
var_names=colnames(X)
available=var_names


## First iteration: to find M1
adj_r_k = c() # vector of the adjusted R2

# for all the models in Call_M1 (which are the ones in available)
for(j in 1:length(available)){

  # Select the row of Call_mk, which contains the columns of X
  # to consider in the model
  X_j = subset(X, select = available[j])
  X_j=data.frame(X_j)

  # Compute adj R2
  adj_r_k[j] = summary(lm(log_y ~ ., X_j))$adj.r.squared

}

# Insert the variable that most ameliorates the model
M=c()
M[1]=available[which.max(adj_r_k)]

# Drop the best from available variables
available=available[!available==M]


## Next iterations: find M2,...,Mp
for (k in 2:p){

  # Define Call_Mk
  Mk_matr=matrix(rep(M, length(available)), nrow = length(available), byrow = T)
  Call_Mk= data.frame(Mk_matr,available)

  # Create the vector for the adjusted R2
  adj_r_k = c()

  # for all the models in Call_Mk
  for(j in 1:nrow(Call_Mk)){

    # Select the row of Call_mk, which contains the columns of X
    # to consider in the model
    covar=unlist(Call_Mk[j,])
    X_j = subset(X, select = covar)
    X_j=data.frame(X_j)

    # Compute the adjusted R2
    adj_r_k[j] = summary(lm(log_y ~ ., X_j))$adj.r.squared

  }

  # Insert the variable that most ameliorates the model
  M[k]=available[which.max(adj_r_k)]

  # Drop the best from available
  available=available[!available==M[k]]
}

# Now I have my M_1:p models

# Compute adjusted R2 and BIC for each Mk
bic=c()
adj_r2=c()
cp=c()
for (k in 1:length(M)){
  X_k=subset(X, select=M[1:k])
  X_k=data.frame(X_k)
  mod=lm(log_y ~., X_k)
  bic[k]=BIC(mod)
  adj_r2[k]=summary(mod)$adj.r.squared
}

cat("The best model using the adjusted R2 criterion is the on with",
    which.max(adj_r2),"covariates")
```

```
## The best model using the adjusted R2 criterion is the on with 71 covariates
```

```r
cat("The best model using the BIC criterion is the on with", which.min(bic),
    "covariates")
```

```
## The best model using the BIC criterion is the on with 22 covariates
```

```r
# Coefficients of the model chosen by BIC
X_k=subset(X, select=M[1:which.min(bic)])
X_k=data.frame(X_k)
mod=lm(log_y ~., X_k)
summary(mod)$coeff
```

```
##                 Estimate Std. Error      t value     Pr(>|t|)
## (Intercept) -1.85117452 0.01280180 -144.6027177 0.000000e+00
## x.V50       -0.43301270 0.05308761   -8.1565673 6.067976e-16
## x.V74        0.07620782 0.02721857    2.7998467 5.162519e-03
## x.V46       -0.01052818 0.04413541   -0.2385427 8.114850e-01
## x.V17        0.19960968 0.02625328    7.6032298 4.440571e-14
## x.V77        0.11059006 0.03203236    3.4524475 5.672516e-04
## x.V9        -0.21478187 0.02981370   -7.2041345 8.278141e-13
## x.V55       -0.14876470 0.03262657   -4.5596188 5.439351e-06
## x.V21       -0.31980925 0.03695012   -8.6551616 1.003959e-17
## x.V71       -0.03303824 0.02735926   -1.2075706 2.273574e-01
## x.V90        0.23686850 0.04289571    5.5219630 3.796047e-08
## x.V52        0.08643770 0.03298839    2.6202464 8.854095e-03
## x.V84       -0.04743033 0.01647552   -2.8788361 4.034493e-03
## x.V88       -0.11827604 0.04046993   -2.9225662 3.511144e-03
## x.V112      -0.05777908 0.01934297   -2.9870846 2.851267e-03
## x.V28       -0.05382199 0.01900101   -2.8325852 4.664265e-03
## x.V19       -0.17725292 0.03516525   -5.0405702 5.065845e-07
## x.V94       -0.04073126 0.01551486   -2.6253070 8.724023e-03
## x.V39        0.12250190 0.03339283    3.6685089 2.504439e-04
## x.V43        0.08387119 0.02583980    3.2458136 1.190787e-03
## x.V44        0.11250597 0.04007092    2.8076710 5.039248e-03
## x.V6         0.94464052 0.20260697    4.6624286 3.334510e-06
## x.V16       -0.87251523 0.20579559   -4.2397178 2.341459e-05
```

```
rm(list = ls())
```

Question 4

Ridge regression

In this section I compute the ridge estimate $\hat{\beta}^{ridge}$ for the same model of Question 1 and 3.

As in Question 3, I standardize the covariates, because the ridge regression penalizes the coefficients that are "far form zero" and hence it is necessary to make the definition of "far from zero" the same for all the covariates.

Furthermore, I also center the response vector (log_y) by subtracting off its mean. In this way, the intercept β_0 (which represent the mean of log_y) is always equal to zero and, hence, I do not need to estimate it.

After computing these transformations, I estimate the $\hat{\beta}^{ridge}$ coefficients. The principle behind the ridge regression (and also behind other penalizations) is to penalize the "less realistic coefficients", which are those coefficients that are further away from zero. Consequently, the coefficient estimates are the values that minimize:

$RSS + \lambda \sum_{j=1}^{p} \beta_j^2$, where λ is a tuning parameter and $\sum_{j=1}^{p} \beta_j^2$ is the ridge penalization.

It is possible to show that:

$\hat{\beta}^{ridge} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$

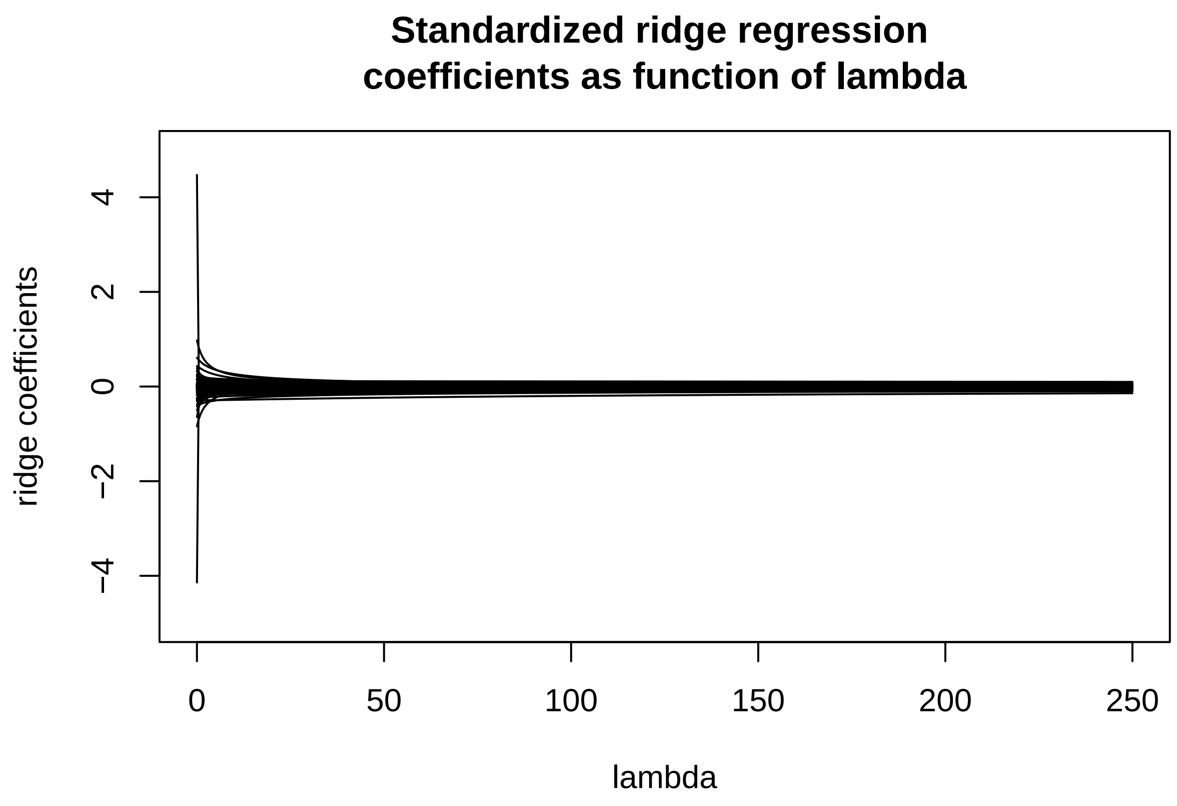
I compute the coefficients for different values of λ and at the end I will choose the best one.

```
### Ridge
Dati <- read.table("crime.txt",header = T,sep=";")
y <- Dati$y
Dati$y <- NULL
```

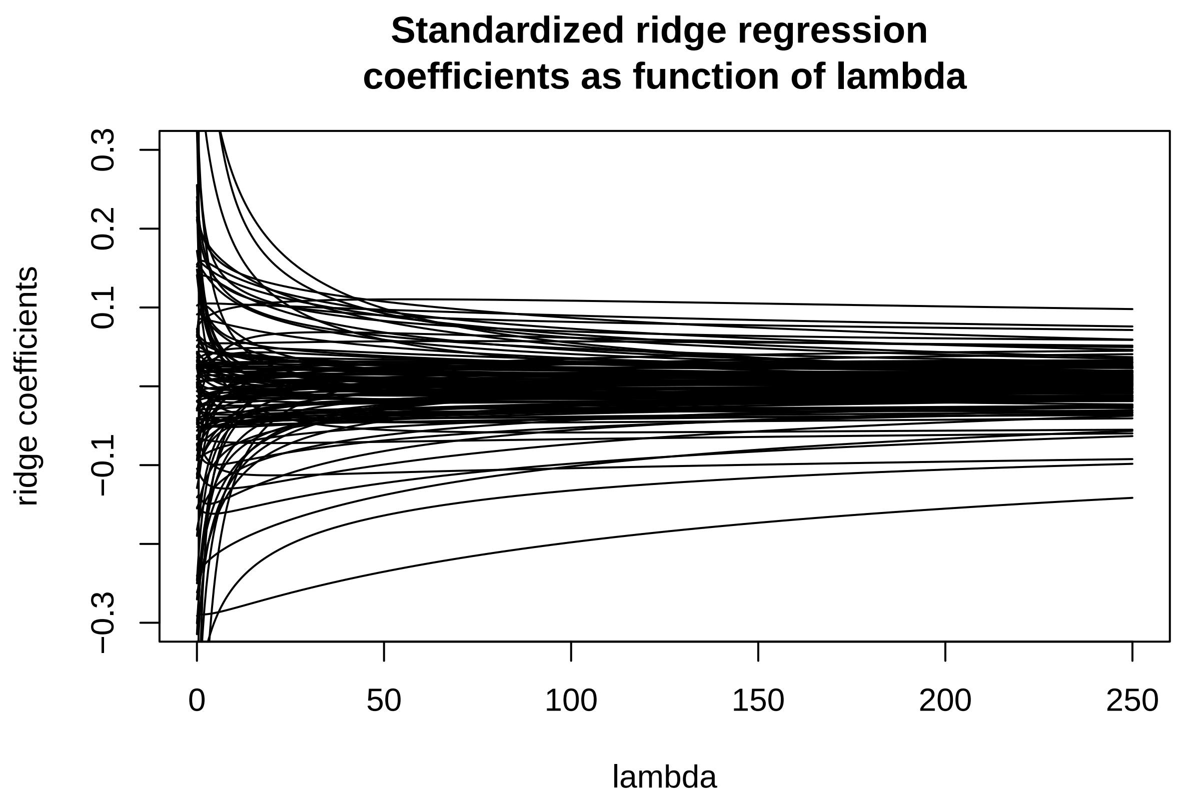
```r
X <- Dati
X=as.matrix(scale(X))
log_y=log(y)
cen_ly=log_y-mean(log_y) # centered to eliminate the intercept

# Beta coefficients
lambda=seq(0,250,by=0.5)
I=diag(x=1, nrow=122, ncol=122)
XtX=t(X)%*%X
n=length(log_y)

beta_ridge=data.frame()

for (i in 1:length(lambda)){
  XtX=t(X)%*%X
  W=(solve(XtX+lambda[i]*I))
  beta_ridge[1:122,i]=W%*%t(X)%*%cen_ly
}
```

In order to visualize the coefficients, I plot how the values of the coefficients change as function of λ. I plot the same graph using two different y-axis scales to make it more understandable.

```r
# Plot
plot(lambda,beta_ridge[1,], type="l", ylim=c(-5,5),
     main="Standardized ridge regression \ncoefficients as function of lambda",
     ylab="ridge coefficients")

for (i in 2:122){
  lines(lambda,beta_ridge[i,], type="l")
}
```



```
# Zoom in
plot(lambda,beta_ridge[1,], type="l", ylim=c(-0.3,0.3),
     main="Standardized ridge regression \ncoefficients as function of lambda",
     ylab="ridge coefficients")

for (i in 2:122){
  lines(lambda,beta_ridge[i,], type="l")
}
```



In the first graph it is possible to see the effect of the tuning parameter λ:

- As $\lambda \to 0$, $\hat{\beta}^{ridge} \to \hat{\beta}^{OLS}$
- As $\lambda \to \infty$, $\hat{\beta}^{ridge} \to 0$

It is also possible to notice that there are (at least) two coefficients that are very far from zero when no penalization is included and that get shrunken a lot when λ increases.

The second graph shows in a clearer way the path of each coefficient, even if, in the final part, most of them overlap due to the convergence to 0.

I now choose the best value for the tuning parameter λ. To do this, I use the generalized cross-validation method and I choose the λ that minimizes it. The generalized cross validation is computed as:

$GCV = \frac{1}{n}\sum_{i=1}^{n}(\frac{y_i - \hat{y}_i}{1 - tr(\mathbf{H})/n})^2$

```
# GCV
GCV=c()
for (i in 1:length(lambda)){
  W=(solve(XtX+lambda[i]*I))
  H=X%*%W%*%t(X)
  cen_ly_hat=H%*%cen_ly
  tr_H=sum(diag(H))
  GCV[i]=1/n*sum(((cen_ly_hat-cen_ly)/(1-(tr_H)/n))^2)
}
```

```
opt_lambda=lambda[which.min(GCV)]
cat("The optimal lambda is:", opt_lambda)
```

```
## The optimal lambda is: 48
```

At this point, it is possible to estimate the ridge coefficients for the model with optimal λ, selecting, from the estimates above, the one with corresponding λ.

```
# Estimation
beta_ridge_opt=beta_ridge[,which.min(GCV)]
cat("The coefficients of the ridge regression using the optimal lambda are:")
```

```
## The coefficients of the ridge regression using the optimal lambda are:
```

```
beta_ridge_opt
```

```
##                [,1]
##    [1,]  9.476159e-02
##    [2,] -6.963180e-02
##    [3,]  1.102362e-01
##    [4,] -1.099511e-01
##    [5,]  3.409939e-02
##    [6,]  1.085902e-01
##    [7,]  1.572252e-03
##    [8,] -8.290981e-03
##    [9,]  1.602761e-02
##   [10,]  9.641765e-02
##   [11,]  2.667986e-03
##   [12,]  9.048620e-02
##   [13,] -2.181147e-03
##   [14,] -8.389856e-02
##   [15,]  2.667040e-02
##   [16,] -2.371296e-01
##   [17,]  4.761190e-03
##   [18,] -2.833353e-04
##   [19,] -2.849393e-02
##   [20,]  1.016971e-01
##   [21,] -4.540197e-02
##   [22,] -2.419138e-02
##   [23,] -3.813720e-02
##   [24,] -1.050828e-02
##   [25,]  9.004944e-03
##   [26,] -2.169632e-02
##   [27,] -5.368149e-04
##   [28,] -1.180681e-03
##   [29,] -1.006322e-01
##   [30,] -3.553507e-02
##   [31,]  3.063518e-02
##   [32,] -3.267921e-02
##   [33,] -8.089162e-03
##   [34,]  6.970117e-02
##   [35,] -3.002668e-02
##  [36,] -1.788743e-02
##  [37,] -1.698151e-03
##  [38,]  8.421903e-02
##  [39,]  8.392301e-02
##  [40,]  6.015103e-02
##  [41,]  1.712559e-02
##  [42,]  2.146669e-02
##  [43,] -2.869573e-02
##  [44,] -5.482685e-02
##  [45,] -1.659279e-01
##  [46,] -2.126374e-03
##  [47,]  2.088576e-02
##  [48,]  2.720693e-02
##  [49,] -5.255615e-02
##  [50,] -1.241078e-01
##  [51,]  9.746047e-02
##  [52,] -4.306333e-02
##  [53,]  1.731149e-03
##  [54,] -1.687199e-02
##  [55,]  6.783897e-03
##  [56,]  6.631145e-03
##  [57,] -1.894684e-02
##  [58,]  3.915491e-03
##  [59,]  8.181224e-03
##  [60,] -2.646756e-02
##  [61,] -1.632123e-02
##  [62,] -6.334472e-02
##  [63,] -8.956834e-03
##  [64,]  2.221474e-02
##  [65,]  6.106967e-02
##  [66,] -6.007863e-05
##  [67,]  6.780425e-02
##  [68,] -4.137678e-02
##  [69,] -6.475229e-04
##  [70,]  1.906035e-02
##  [71,]  9.649355e-03
##  [72,]  2.243546e-02
##  [73,] -7.057741e-02
##  [74,] -1.169531e-02
##  [75,]  2.273384e-02
##  [76,] -4.541532e-02
##  [77,] -9.724466e-03
##  [78,]  2.313178e-02
##  [79,] -4.010025e-02
##  [80,] -3.126749e-02
##  [81,]  2.117082e-03
##  [82,]  2.205528e-02
##  [83,] -1.400597e-01
##  [84,]  6.218476e-03
##  [85,]  9.080926e-02
##  [86,]  5.096074e-02
##  [87,]  3.124016e-02
##  [88,] -2.907637e-02
##  [89,] -4.150515e-02
##  [90,]  2.240036e-02
##  [91,]  2.664620e-02
##  [92,]  6.484117e-02
##  [93,] -2.117239e-02
##  [94,] -6.778324e-04
##  [95,]  1.493244e-02
##  [96,]  3.131705e-02
##  [97,] -1.812064e-02
##  [98,] -4.449725e-03
##  [99,]  1.090594e-02
## [100,] -3.450952e-02
## [101,] -3.752932e-02
## [102,] -5.736518e-03
## [103,]  4.261664e-02
## [104,] -4.766621e-03
## [105,]  1.495509e-02
## [106,]  2.964141e-02
## [107,] -1.977193e-02
## [108,]  2.825645e-02
## [109,]  1.787228e-02
## [110,] -6.058840e-03
## [111,]  3.691087e-02
## [112,]  1.143841e-02
## [113,] -9.408864e-03
## [114,]  3.157320e-03
## [115,]  1.509653e-02
## [116,] -3.451437e-02
## [117,]  5.697135e-02
## [118,]  3.885399e-02
## [119,] -4.549502e-02
## [120,]  1.043276e-02
## [121,]  2.374586e-02
## [122,] -1.627952e-02
```

To conclude, I compute the adjusted R^2 for this model to compare it with the results of Question 3.

```r
# Adjusted R2 for comparison
W=(solve(XtX+opt_lambda*I))
H=X%*%W%*%t(X)
p=sum(diag(H))
cen_ly_hat=H%*%cen_ly
RSS=sum((cen_ly-cen_ly_hat)^2)
TSS=sum((cen_ly-mean(cen_ly))^2)
adj_r2=1-(RSS*(n-p-1))/(TSS*(n-1))
cat("The adjusted R2 of the model is:", adj_r2)
```

```
## The adjusted R2 of the model is: 0.7031478
```

The adjusted R^2 of this model is 0.7031478, which is higher than those of the stepwise subset selection methods in Question 3. This is due to the fact that ridge regression does not perform a subset selection, but it just reduces the variance of the coefficients, by shrinking them toward zero. Hence, all the 122 coefficients are used in this model, whereas only 22 were used in the subset selection models, leading to an higher adjusted R^2.

Lasso regression

The lasso regression, as the ridge, is a penalized regression in which the "less realistic" coefficients are penalized using the following penalty function: $P(\beta) = \sum_{j=1}^{p} |\beta_j|$

Consequently, the lasso estimator minimizes: $\frac{1}{2}RSS + \lambda \sum_{j=1}^{p} |\beta_j|$.

Since this function is not differentiable, optimization algorithms are needed to find the solution.

Also in this case, I considered the standardized covariates matrix and the centered response vector, for the reasons described above.

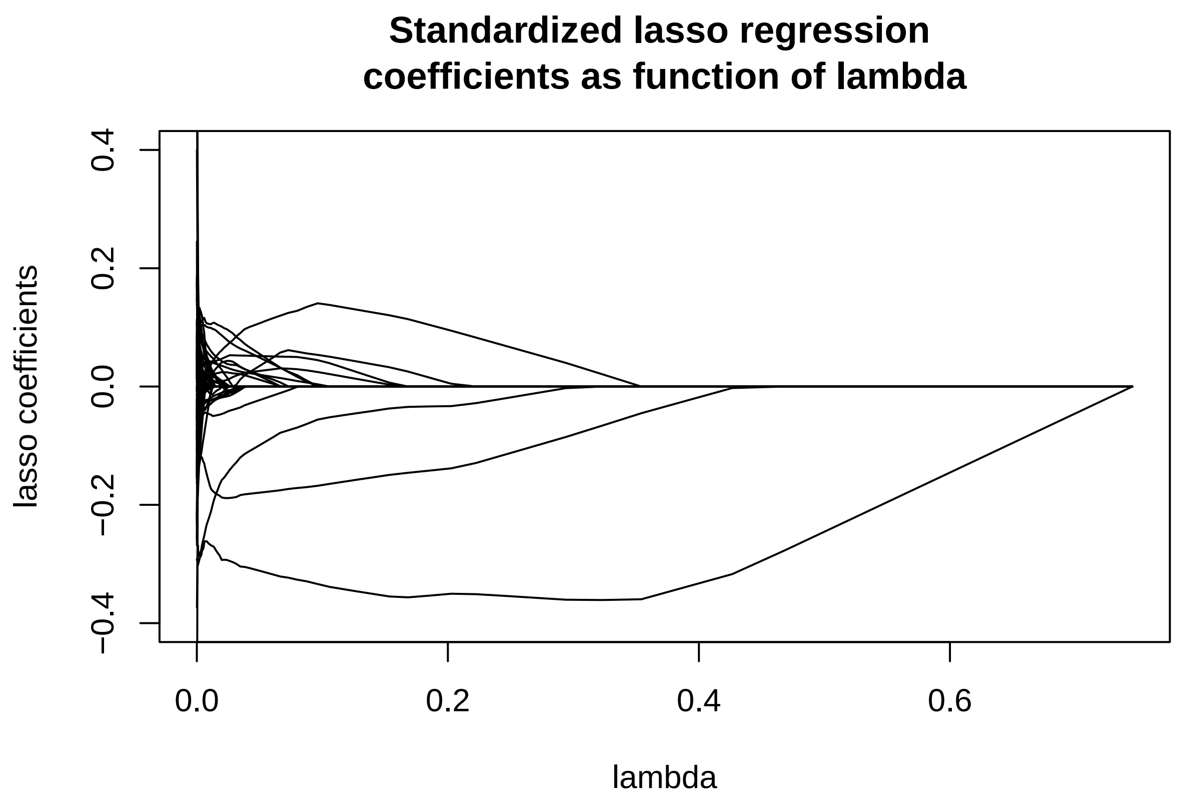
```
### Lasso
library(glmnet)
```

```
## Warning: il pacchetto 'glmnet' è stato creato con R versione 4.2.2

## Caricamento del pacchetto richiesto: Matrix

## Loaded glmnet 4.1-6
```

```
lasso_fit=glmnet(X, cen_ly)
```

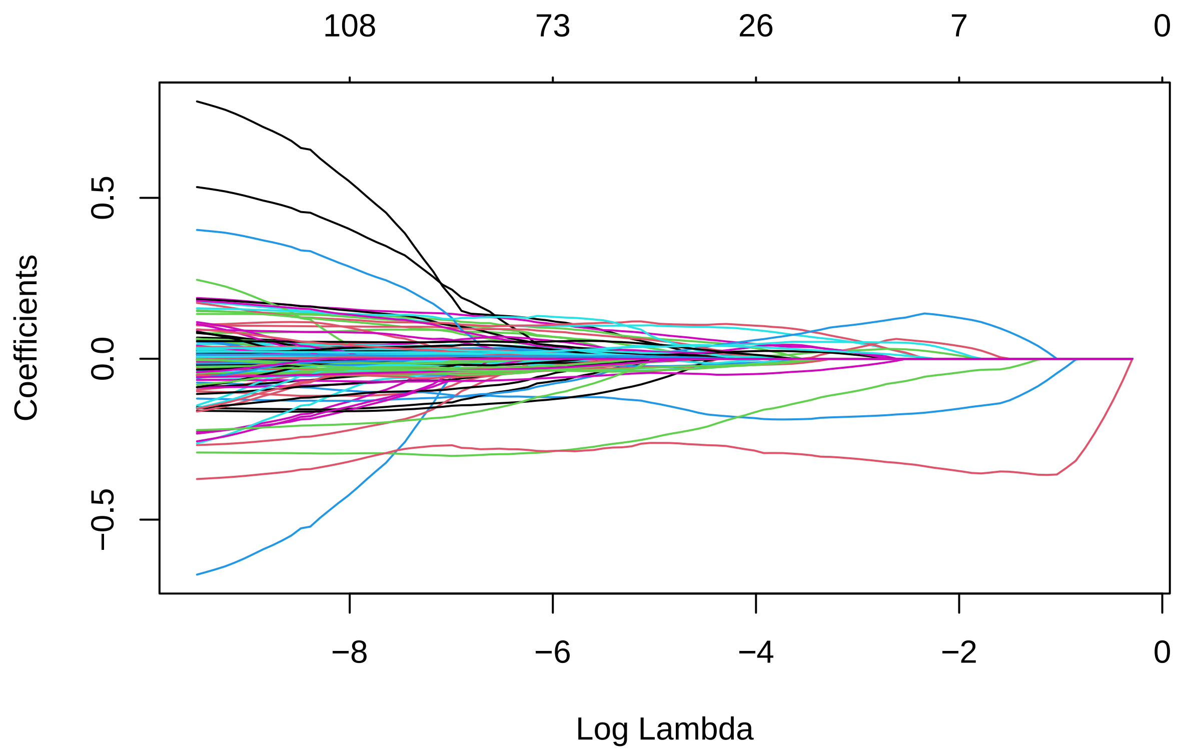
I now replicate the plot I did for the ridge regression, in which the coefficients are represented as function of λ.

```
# Plot with lambda
lasso_coef=as.matrix(coef(lasso_fit))
plot(lasso_fit$lambda, lasso_coef[1,], type="l", ylim=c(-0.4,0.4),
     main="Standardized lasso regression \ncoefficients as function of lambda",
     ylab="lasso coefficients", xlab="lambda")

for (i in 2:122){
  lines(lasso_fit$lambda,lasso_coef[i,], type="l")
}
```



Then I plot the coefficients using a plot provided by the package glmnet, that uses, as x-axis variable, the logarithm of λ.

```
# Plot with log lambda
plot(lasso_fit, xvar="lambda")
```



In both graphs, it appears clearly that, differently form the ridge coefficients, the lasso coefficients are shrunken all the way to zero as the tuning parameter increases. For this reason, the lasso regression also performs subset selection.

Also for the lasso regression it is necessary to identify the optimal value of λ. The glmnet package offers a function that determines the best λ by performing a cross-validation.

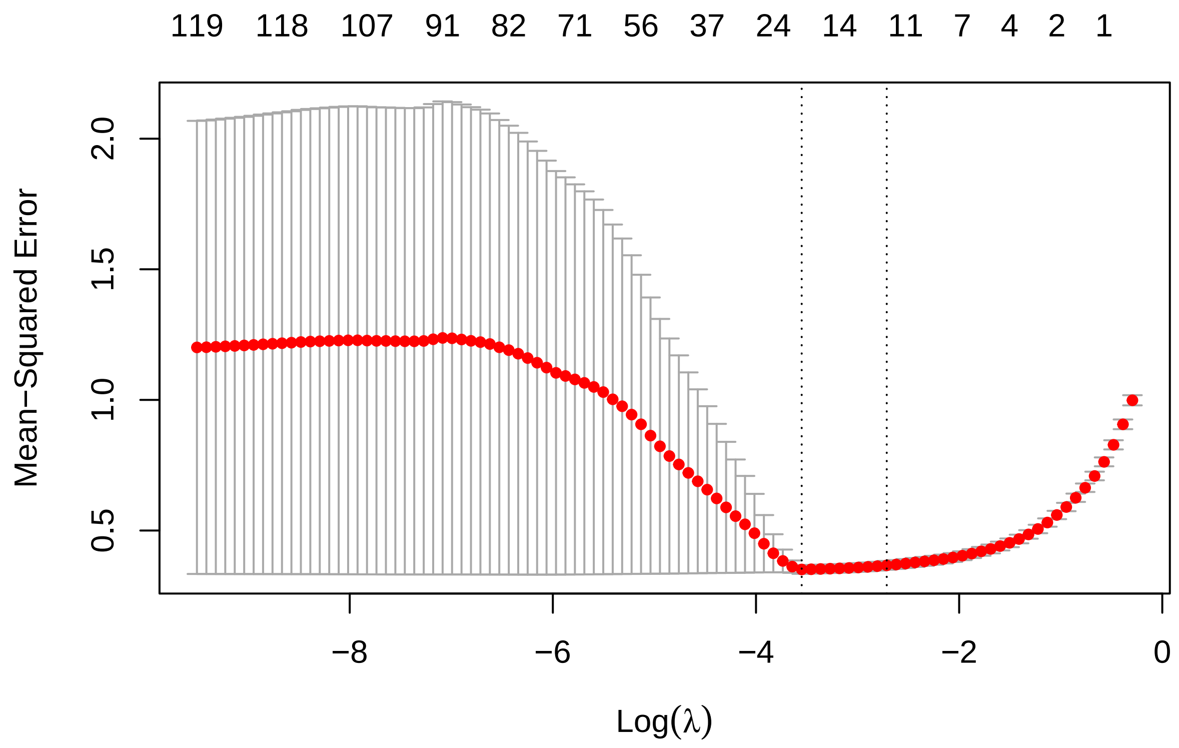
```
# Optimal lambda
set.seed(123)
cv_lasso=cv.glmnet(X, cen_ly)

cat("The optimal lambda is:", cv_lasso$lambda.min)
```

```
## The optimal lambda is: 0.02872345
```

```
cat("The log of the optimal lambda is:", log(cv_lasso$lambda.min))
```

```
## The log of the optimal lambda is: -3.550042
```

```
plot(cv_lasso)
```



The lambda which yields to the minimum CV error is 0.02872345 and its logarithm is -3.550042. The plot shows the CV error as a function of the logarithm of λ and, as expected, the dashed line that indicates the minimum CV error is on the value -3.550042.

Now that I have the optimal λ, I can compute the coefficients of the related model.

```r
# Estimation
cat("The coefficients of the lasso regression with optimal lambda are:")
```

```
## The coefficients of the lasso regression with optimal lambda are:
```

```r
coef(cv_lasso, s = "lambda.min")
```

```
## 123 x 1 sparse Matrix of class "dgCMatrix"
##                          s1
## (Intercept)  5.076261e-17
## x.V6         .
## x.V7         .
## x.V8         .
## x.V9        -1.877921e-01
## x.V10        .
## x.V11        3.661731e-02
## x.V12        .
## x.V13        .
## x.V14        .
## x.V15        2.420961e-05
## x.V16        .
## x.V17        7.145383e-02
## x.V18        .
## x.V19       -1.194948e-02
## x.V20        .
## x.V21       -1.344584e-01
## x.V22        .
## x.V23        .
## x.V24        .
## x.V25        .
## x.V26        .
## x.V27        .
## x.V28        .
## x.V29        .
## x.V30        .
## x.V31        .
## x.V32        .
## x.V33        .
## x.V34        .
## x.V35        .
## x.V36        .
## x.V37        .
## x.V38        .
## x.V39        .
## x.V40       -4.251457e-03
## x.V41        .
## x.V42        .
## x.V43        .
## x.V44        8.974730e-02
## x.V45        .
## x.V46        7.851610e-02
## x.V47        .
## x.V48        .
## x.V49        .
## x.V50       -2.969310e-01
## x.V51        .
## x.V52        .
## x.V53        .
## x.V54        .
## x.V55        .
## x.V56        .
## x.V57        .
## x.V58        .
## x.V59        .
## x.V60        .
## x.V61        .
## x.V62        .
## x.V63        .
## x.V64        .
## x.V65        .
## x.V66        .
## x.V67        .
## x.V68        .
## x.V69          .
## x.V70          .
## x.V71          .
## x.V72          .
## x.V73          .
## x.V74          .
## x.V75          1.600974e-02
## x.V76          .
## x.V77          5.276697e-02
## x.V78         -3.929481e-02
## x.V79          .
## x.V80          .
## x.V81         -1.297518e-02
## x.V82          .
## x.V83          .
## x.V84          .
## x.V85          .
## x.V86          .
## x.V87          .
## x.V88          .
## x.V89          .
## x.V90          .
## x.V91          .
## x.V92          2.225624e-02
## x.V93          .
## x.V94         -7.706240e-03
## x.V95          .
## x.V96          .
## x.V97          4.201262e-02
## x.V98          .
## x.V99          .
## x.V100         .
## x.V101         .
## x.V102         .
## x.V103         .
## x.V104         .
## x.V105         .
## x.V106         .
## x.V107         .
## x.V108         .
## x.V109         .
## x.V110         .
## x.V111         .
## x.V112         .
## x.V113         .
## x.V114         .
## x.V115         .
## x.V116         .
## x.V117         .
## x.V118         .
## x.V119         .
## x.V120         .
## x.V121         .
## x.V122         .
## x.V123       .
## x.V124      -4.301448e-03
## x.V125       .
## x.V126       2.847761e-02
## x.V127       .
```

```
cat("Number of non-zero coefficients:", sum((coef(cv_lasso, s = "lambda.min"))!=0))
```

```
## Number of non-zero coefficients: 20
```

As mentioned above, the lasso regression shrink some of the coefficients all the way to zero. In particular, in this case, 103 coefficients have been shrunken to zero and the number of non zero coefficients is 20.

To conclude, as I did for the previous models, I compute the adjusted R^2 of the lasso regression model with the optimal λ, to compare it with the others.

```
# Adjusted R2 for comparison
p=sum((coef(cv_lasso, s = "lambda.min"))!=0) # number of non-zero coeff
cen_ly_hat= predict(cv_lasso, X, s="lambda.min")
RSS=sum((cen_ly-cen_ly_hat)^2)
TSS=sum((cen_ly-mean(cen_ly))^2)
adj_r2=1-(RSS*(n-p-1))/(TSS*(n-1))
cat("The adjusted R2 of the model is:", adj_r2)
```

```
## The adjusted R2 of the model is: 0.65988
```

The adjusted R^2 of the model is slightly lower than the subset selection models ones. Also in this case, this is probably due to the different number of covariates included in the model, that, in this case is lower.