Homework Assignment 3

1. We perform best subset, forward stepwise, and backward stepwise selection on a single data set. For each approach, we obtain p + 1 models, containing 0, 1, 2, . . . , p predictors. Explain your answers:

(a) Which of the three models with k predictors has the smallest training RSS?

Given the condition that a single data set is used to perform the best subset, forward stepwise, and backward stepwise selection and for each approach, we obtain p + 1 models, containing 0, 1, 2, . . . , p predictors then the out of the three models with k predictors best subset selection method will have the smallest training RSS. Though forward stepwise tends to do well in practice, it is not guaranteed to find the best possible model out of all the models containing subsets of the p predictors. Also like forward stepwise selection, backward stepwise selection is not guaranteed to yield the best model containing a subset of the p predictors. Furthermore, Backward selection requires that the number of samples n is larger than the number of variables p (so that the full model can be fit). In contrast, forward stepwise can be used even when $n < p$, and so is the only viable subset method when p is very large.

(b) Which of the three models with k predictors has the smallest test RSS?

It might be possible that the best subset selection method and backward stepwise selection might arrive at the same model as backward stepwise selection iteratively removes the least useful predictor, one at a time and best subset selection tries different combinations of all the predictors while forward selection retains the old predictors when it iteratively add new predictors. So there is no guarantee that which model will come up with the smallest test RSS as either of these methods can pick up a model that best fits the test data and provide the smallest test RSS.

(c) True or False:
i. The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the (k+1)-variable model identified by forward stepwise selection. - **True**
ii. The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the (k + 1) - variable model identified by backward stepwise selection. - **True**
iii. The predictors in the k-variable model identified by back-ward stepwise are a subset of the predictors in the (k + 1) - variable model identified by forward stepwise selection. - **False**
iv. The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the (k+1)-variable model identified by backward stepwise selection. - **False**
v. The predictors in the k-variable model identified by best subset are a subset of the predictors in the (k + 1)-variable model identified by best subset selection. - **False**

2. We will now explore (6.12) and (6.13) further.

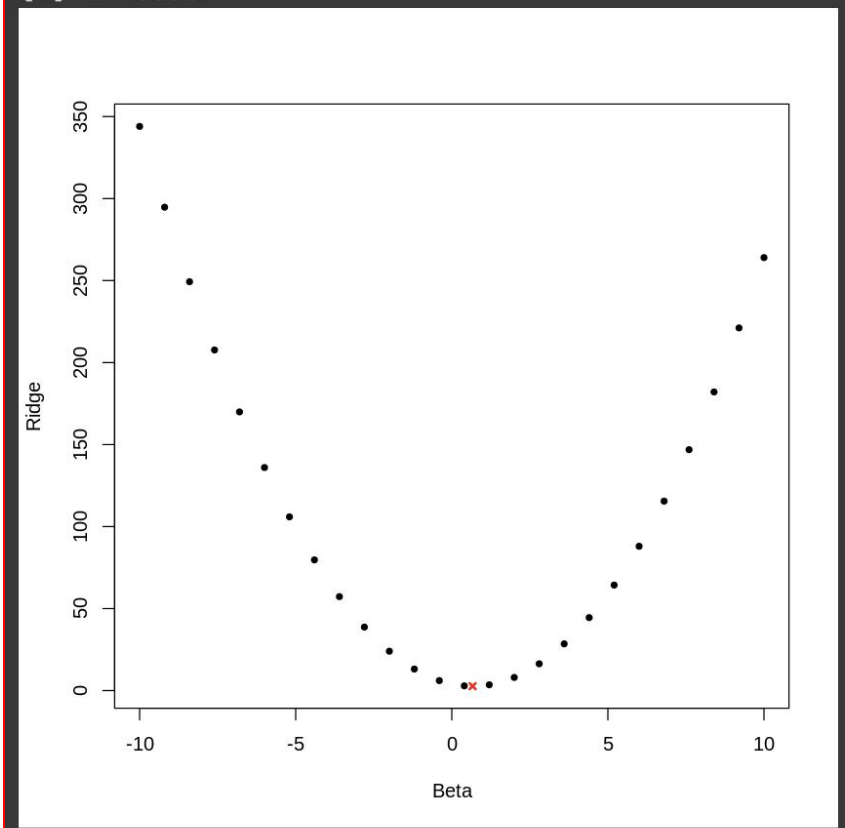
(a) Consider (6.12) with p = 1. For some choice of y1 and λ > 0, plot (6.12) as a function of β1. Your plot should confirm that (6.12) is solved by (6.14).

$$\sum_{j=1}^{p}(y_j - \beta_j)^2 + \lambda\sum_{j=1}^{p}\beta_j^2 \qquad (6.12)$$

$$\hat{\beta}_j^R = y_j/(1+\lambda), \qquad (6.14)$$

```
betas=seq(-10,10,0.8)

# equation 6.12
func1 = ((y - betas)^2) + (lambda * (betas^2))

plot(betas,func1,pch=20,xlab='Beta',ylab='Ridge')

# equation 6.14
estimation_beta=y/(1+lambda)
print(estimation_beta)

estimation_function=((y - estimation_beta)^2) + (lambda * (estimation_beta^2))
print(estimation_function)

points(estimation_beta,estimation_function,col='red',pch=4,lwd=2,cex=estimation_beta)
```

```
[1] 0.6666667
[1] 2.666667
```



The red cross in the figure shows that the function (6.12) is minimized by beta (6.14)

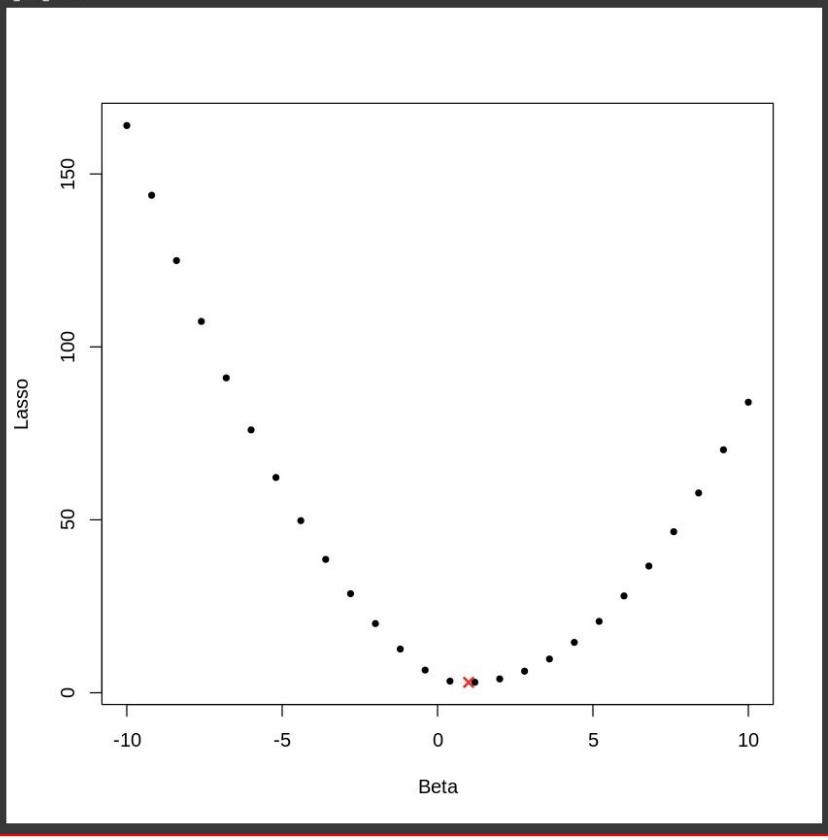
(b) Consider (6.13) with p = 1. For some choice of y1 and λ > 0, plot (6.13) as a function of β1. Your plot should confirm that (6.13) is solved by (6.15).

$$\sum_{j=1}^{p}(y_j - \beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j| \qquad (6.13)$$

$$\hat{\beta}_j^L = \begin{cases} y_j - \lambda/2 & \text{if } y_j > \lambda/2; \\ y_j + \lambda/2 & \text{if } y_j < -\lambda/2; \\ 0 & \text{if } |y_j| \leq \lambda/2. \end{cases} \qquad (6.15)$$

```
betas=seq(-10,10,0.8)

# equation 6.13
func1=((y - betas)^2) + (lambda * abs(betas))

plot(betas,func1,pch=20,xlab='Beta',ylab='Lasso')

# equation 6.15
# since y > (lambda / 2) i.e 2 > (2/2) so
estimation_beta=y-(lambda/2)
print(estimation_beta)

estimation_function=((y - estimation_beta)^2) + (lambda * abs(estimation_beta))
print(estimation_function)

points(estimation_beta,estimation_function,col='red',pch=4,lwd=2,cex=estimation_beta)
```

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



The red cross in the figure shows that the function (6.13) is minimized by beta (6.15)

3. In this exercise, we will generate simulated data, and will then use this data to perform best subset selection.

(a) Use the rnorm() function to generate a predictor X of length n = 100, as well as a noise vector ε of length n = 100.

```
set.seed(123)

x_n=100
e_n=100
x=rnorm(x_n)
print(x)
e_noise=rnorm(e_n)
print(e_noise)
```

```
 [1] -0.560475647 -0.230177489  1.558708314  0.070508391  0.129287735
 [6]  1.715064987  0.460916206 -1.265061235 -0.686852852 -0.445661970
[11]  1.224081797  0.359813827  0.400771451  0.110682716 -0.555841135
[16]  1.786913137  0.497850478 -1.966617157  0.701355902 -0.472791408
[21] -1.067823706 -0.217974915 -1.026004448 -0.728891229 -0.625039268
[26] -1.686693311  0.837787044  0.153373118 -1.138136937  1.253814921
[31]  0.426464221 -0.295071483  0.895125661  0.878133488  0.821581082
[36]  0.688640254  0.553917654 -0.061911711 -0.305962664 -0.380471001
[41] -0.694706979 -0.207917278 -1.265396352  2.168955965  1.207961998
[46] -1.123108583 -0.402884835 -0.466655354  0.779965118 -0.083369066
[51]  0.253318514 -0.028546755 -0.042870457  1.368602284 -0.225770986
[56]  1.516470604 -1.548752804  0.584613750  0.123854244  0.215941569
[61]  0.379639483 -0.502323453 -0.333207384 -1.018575383 -1.071791226
[66]  0.303528641  0.448209779  0.053004227  0.922267468  2.050084686
[71] -0.491031166 -2.309168876  1.005738524 -0.709200763 -0.688008616
[76]  1.025571370 -0.284773007 -1.220717712  0.181303480 -0.138891362
[81]  0.005764186  0.385280401 -0.370660032  0.644376549 -0.220486562
[86]  0.331781964  1.096839013  0.435181491 -0.325931586  1.148807618
[91]  0.993503856  0.548396960  0.238731735 -0.627906076  1.360652449
[96] -0.600259587  2.187332993  1.532610626 -0.235700359 -1.026420900
 [1] -0.71040656  0.25688371 -0.24669188 -0.34754260 -0.95161857 -0.04502772
 [7] -0.78490447 -1.66794194 -0.38022652  0.91899661 -0.57534696  0.60796432
[13] -1.61788271 -0.05556197  0.51940720  0.30115336  0.10567619 -0.64070601
[19] -0.84970435 -1.02412879  0.11764660 -0.94747461 -0.49055744 -0.25609219
[25]  1.84386201 -0.65194990  0.23538657  0.07796085 -0.96185663 -0.07130809
[31]  1.44455086  0.45150405  0.04123292 -0.42249683 -2.05324722  1.13133721
[37] -1.46064007  0.73994751  1.90910357 -1.44389316  0.70178434 -0.26219749
[43] -1.57214416 -1.51466765 -1.60153617 -0.53090652 -1.46175558  0.68791677
[49]  2.10010894 -1.28703048  0.78773885  0.76904224  0.33220258 -1.00837661
[55] -0.11945261 -0.28039534  0.56298953 -0.37243876  0.97697339 -0.37458086
[61]  1.05271147 -1.04917701 -1.26015524  3.24103993 -0.41685759  0.29822759
[67]  0.63656967 -0.48378063  0.51686204  0.36896453 -0.21538051  0.06529303
[73] -0.03406725  2.12845190 -0.74133610 -1.09599627  0.03778840  0.31048075
[79]  0.43652348 -0.45836533 -1.06332613  1.26318518 -0.34965039 -0.86551286
[85] -0.23627957 -0.19717589  1.10992029  0.08473729  0.75405379 -0.49929202
[91]  0.21444531 -0.32468591  0.09458353 -0.89536336 -1.31080153  1.99721338
[97]  0.60070882 -1.25127136 -0.61116592 -1.18548008
```

(b) Generate a response vector Y of length n = 100 according to the model $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \varepsilon$, where β_0, β_1, β_2, and β_3 are constants of your choice.

```
beta_0=1
beta_1=2
beta_2=3
beta_3=4
```

```
y = beta_0 + (beta_1 * x) + (beta_2 * (x^2)) + (beta_3 * (x^3)) + e_noise
```

```
y
```

-0.593214479535489 · 0.906693002891148 · 26.3074134401957 · 0.809790594284573 · 0.365747201112343 · 33.3885432561223 · 2.16593425634838 · -6.49523905058849 · -0.634769301967035 · 1.26945656292057 · 14.7044857305433 · 2.90232456031134 · 0.922997503362667 · 1.20797920407416 · 0.647673734916369 · 37.2770284501246 · 3.33852156425047 · -22.39541184983 · 4.40869591337405 · -0.721851957642085 · -2.46759151777556 · -0.282311877017288 · -2.70474945223353 · -0.669015775289461 · 1.78905914898594 · -13.6846603282348 · 7.36874984033212 · 1.46970840065668 · -4.24923206353372 · 16.0367256213954 · 4.1533416447014 · 1.0197984595414 · 8.10411165252995 · 7.35570504509802 · 4.83315540839494 · 6.23757671894972 · 2.24749215656729 · 1.62667402426127 · 2.46344917964417 · -0.990865777733988 · 0.419111959316267 · 0.415704019583014 · -6.40400498779967 · 58.7506395370841 · 13.2424026059506 · -3.65963989559314 · -1.04215561077853 · 1.00141876844334 · 8.38302928200068 · -0.43523519909908 · 2.55190875334469 · 1.71430042905389 · 1.25166013033574 · 18.6020082240071 · 0.535890555395325 · 24.6012024028037 · -9.19818259985384 · 3.62132978466808 · 2.27830112937003 · 1.23747264188042 · 3.46323272992399 · -0.80384015835976 · -0.741468809209702 · 1.08930582900584 · -3.03905326399369 · 2.29352971598538 · 3.49583229380926 · 0.631251822392155 · 9.05096785256671 · 52.5424464259703 · 0.0523187261096945 · -36.808625622539 · 10.0811979513454 · 1.79213284541305 · -0.999977391458018 · 9.42530643503964 · 0.619153955559616 · -3.936717933768 · 1.92158176546959 · 0.311007066246303 · -0.0516973185744989 · 3.70783455363648 · 0.117497890337623 · 3.73913869390281 · 0.425715058684329 · 1.94271511586668 · 13.1910000151638 · 2.8529128469592 · 1.28238813694678 · 12.8221965708793 · 10.0851543305882 · 3.33402358993131 · 1.79744952206143 · -0.958625558264435 · 18.0409407947742 · 2.01250702461877 · 62.1891797422247 · 24.2604036646214 · 0.0317203303908158 · -3.4032036367233

(c) Use the regsubsets() function to perform best subset selection in order to choose the best model containing the predictors $X, X^2, ..., X^{10}$. What is the best model obtained according to Cp, BIC, and adjusted R^2? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained. Note you will need to use the data.frame() function to create a single data set containing both X and Y.

```
1  library(leaps)
```

```
1  question_3_data=data.frame(x,x^2,x^3,x^4,x^5,x^6,x^7,x^8,x^9,x^10,y)
```

```
1  model_bss=regsubsets(y~.,data=question_3_data,nvmax=10)
```

```
1  model_bss_summary=summary(model_bss)
```

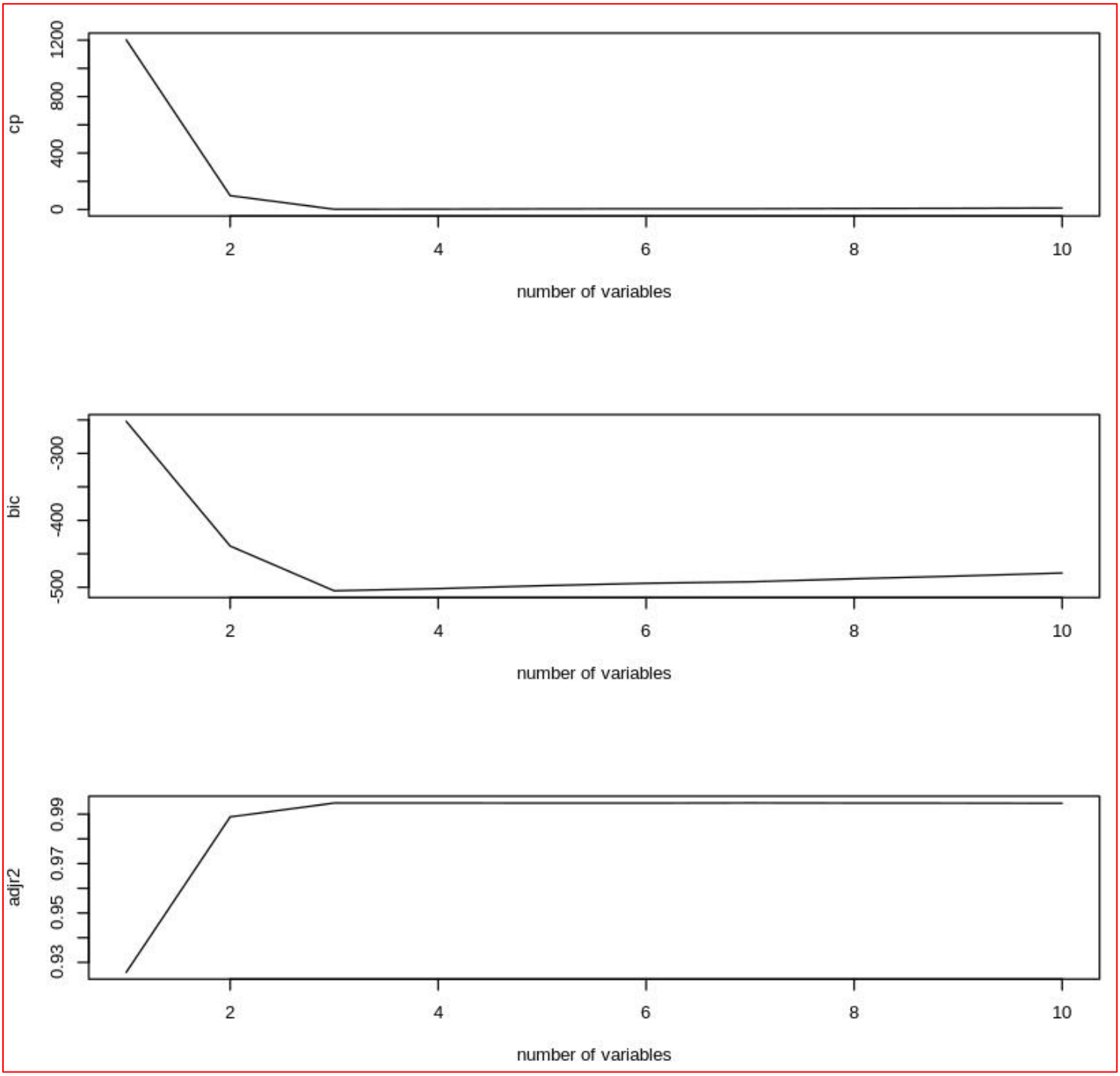
```
1  model_bss_summary
```

```
Subset selection object
Call: regsubsets.formula(y ~ ., data = question_3_data, nvmax = 10)
10 Variables  (and intercept)
     Forced in Forced out
x        FALSE      FALSE
x.2      FALSE      FALSE
x.3      FALSE      FALSE
x.4      FALSE      FALSE
x.5      FALSE      FALSE
x.6      FALSE      FALSE
x.7      FALSE      FALSE
x.8      FALSE      FALSE
x.9      FALSE      FALSE
x.10     FALSE      FALSE
1 subsets of each size up to 10
Selection Algorithm: exhaustive
          x   x.2 x.3 x.4 x.5 x.6 x.7 x.8 x.9 x.10
1  ( 1 )  " " " " "*" " " " " " " " " " " " " "
2  ( 1 )  " " "*" "*" " " " " " " " " " " " " "
3  ( 1 )  "*" "*" "*" " " " " " " " " " " " " "
4  ( 1 )  "*" "*" "*" " " " " "*" " " " " " " "
5  ( 1 )  "*" "*" "*" " " " " " " "*" " " "*" " "
6  ( 1 )  "*" "*" "*" "*" " " "*" " " "*" " " " "
7  ( 1 )  "*" "*" "*" "*" " " "*" " " "*" " " "*"
8  ( 1 )  "*" "*" "*" "*" "*" "*" " " "*" " " "*"
9  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" " " "*"
10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

```
1  names(model_bss)
```

```
'np' · 'nrbar' · 'd' · 'rbar' · 'thetab' · 'first' · 'last' · 'vorder' · 'tol' · 'rss' · 'bound' · 'nvmax' · 'ress' · 'ir' · 'nbest
```

```r
par(mfrow=c(3,1))
plot(model_bss_summary$cp,xlab='number of variables',ylab='cp',type='l')
plot(model_bss_summary$bic,xlab='number of variables',ylab='bic',type='l')
plot(model_bss_summary$adjr2,xlab='number of variables',ylab='adjr2',type='l')
```




```r
c(which.min(model_bss_summary$cp),which.min(model_bss_summary$bic),which.max(model_bss_summary$adjr2))
```

```
3·3·7
```

```r
print(coef(model_bss,3))
```

```
(Intercept)           x         x.2         x.3
  0.9703939   1.9204462   2.9084569   4.0204363
```

```r
print(coef(model_bss,7))
```

```
(Intercept)           x         x.2         x.3         x.4         x.6
  0.7931750   1.8907484   5.2758848   4.0530514  -4.0243410   2.2138377
        x.8        x.10
```

(d) Repeat (c), using forward stepwise selection and also using back-wards stepwise selection. How does your answer compare to the results in (c)?

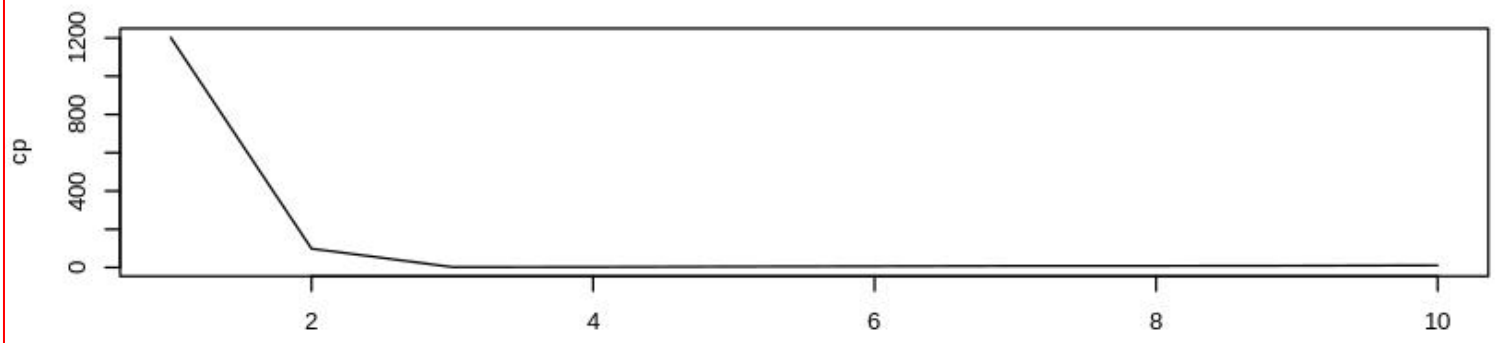
Forward Stepwise Selection

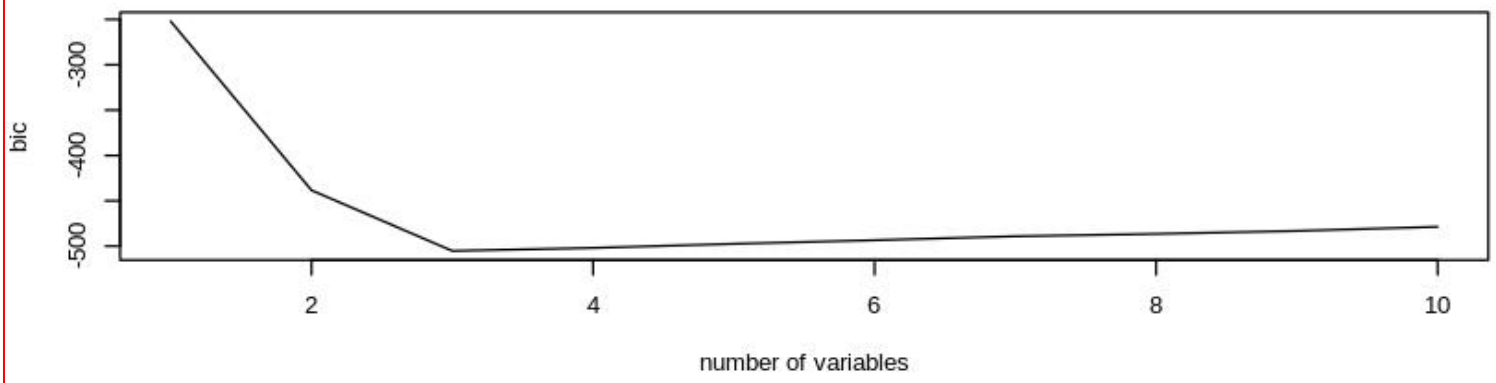
```
# forward stepwise selection
model_fds=regsubsets(y~.,data=question_3_data,nvmax=10,method='forward')
model_fds_summary=summary(model_fds)
```

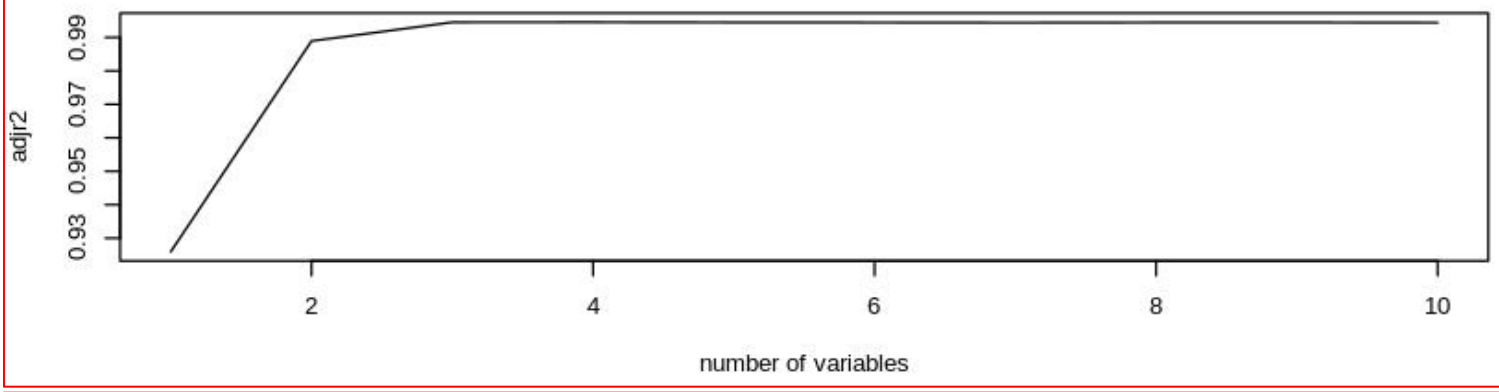
```
model_fds_summary
```

```
Subset selection object
Call: regsubsets.formula(y ~ ., data = question_3_data, nvmax = 10,
    method = "forward")
10 Variables  (and intercept)
     Forced in Forced out
x        FALSE      FALSE
x.2      FALSE      FALSE
x.3      FALSE      FALSE
x.4      FALSE      FALSE
x.5      FALSE      FALSE
x.6      FALSE      FALSE
x.7      FALSE      FALSE
x.8      FALSE      FALSE
x.9      FALSE      FALSE
x.10     FALSE      FALSE
1 subsets of each size up to 10
Selection Algorithm: forward
          x   x.2 x.3 x.4 x.5 x.6 x.7 x.8 x.9 x.10
1  ( 1 )  " " " " "*" " " " " " " " " " " " " "
2  ( 1 )  " " "*" "*" " " " " " " " " " " " " "
3  ( 1 )  "*" "*" "*" " " " " " " " " " " " " "
4  ( 1 )  "*" "*" "*" " " " " "*" " " " " " " "
5  ( 1 )  "*" "*" "*" " " " " " " "*" " " " " "*" " "
6  ( 1 )  "*" "*" "*" " " " " "*" "*" " " " " "*" " "
7  ( 1 )  "*" "*" "*" " " " " "*" "*" " " " " "*" "*"
8  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " " " "*" "*"
9  ( 1 )  "*" "*" "*" "*" "*" "*" " " " " "*" "*" "*"
10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

```
par(mfrow=c(3,1))
plot(model_fds_summary$cp,xlab='number of variables',ylab='cp',type='l')
plot(model_fds_summary$bic,xlab='number of variables',ylab='bic',type='l')
plot(model_fds_summary$adjr2,xlab='number of variables',ylab='adjr2',type='l')
```








```r
indices<-c(which.min(model_fds_summary$cp),which.min(model_fds_summary$bic),which.max(model_fds_summary$adjr2))

for (index in indices) {
  cat("Number of variables:", index, "\n")
  cat("Coefficients:\n")
  print(coef(model_fds, id = index))
  cat("\n")
}
```

```
Number of variables: 3
Coefficients:
(Intercept)           x         x.2         x.3
  0.9703939   1.9204462   2.9084569   4.0204363

Number of variables: 3
Coefficients:
(Intercept)           x         x.2         x.3
  0.9703939   1.9204462   2.9084569   4.0204363

Number of variables: 4
Coefficients:
(Intercept)           x         x.2         x.3         x.6
1.040475559 1.928367169 2.746130084 4.023397035 0.009248115
```

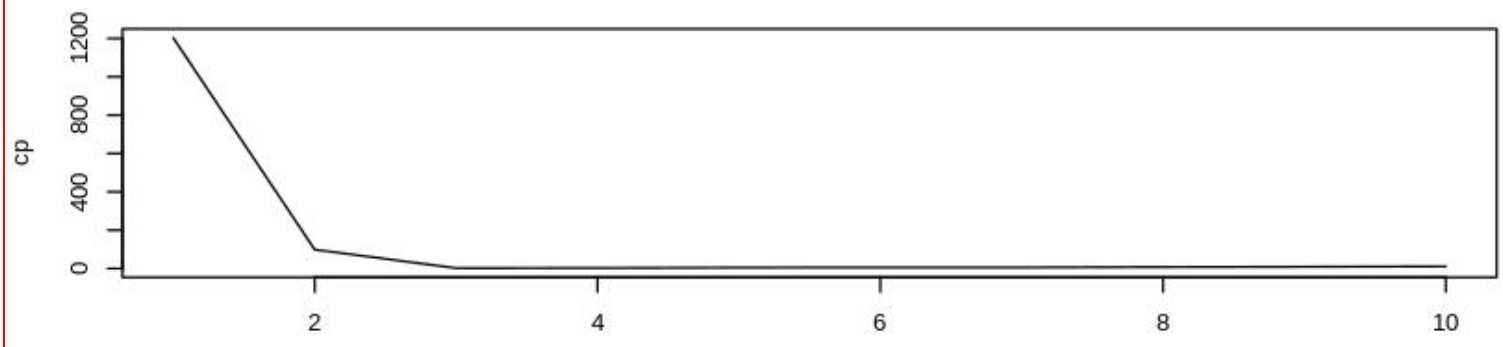
Backward Stepwise Selection

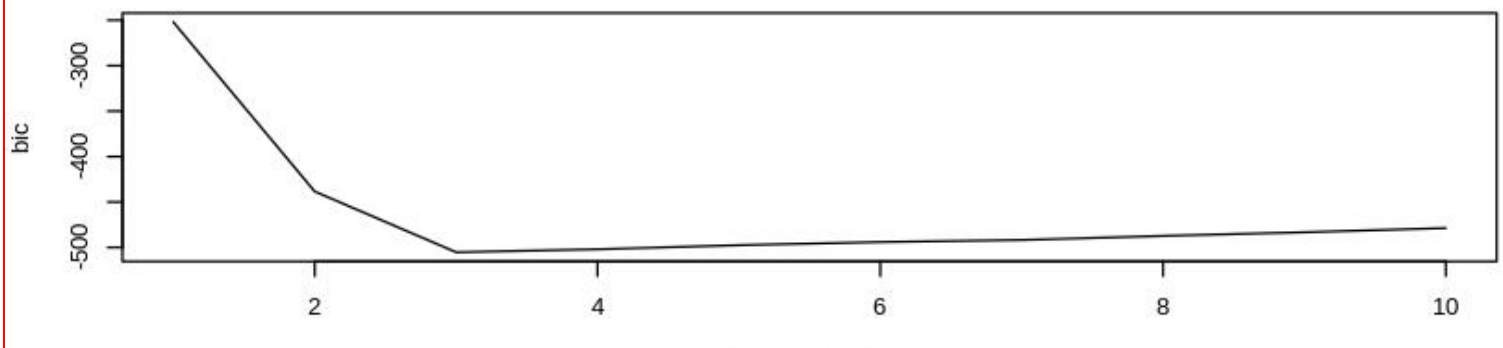
```
# backward stepwise selection
model_bds=regsubsets(y~.,data=question_3_data,nvmax=10,method='backward')
model_bds_summary=summary(model_bds)
```

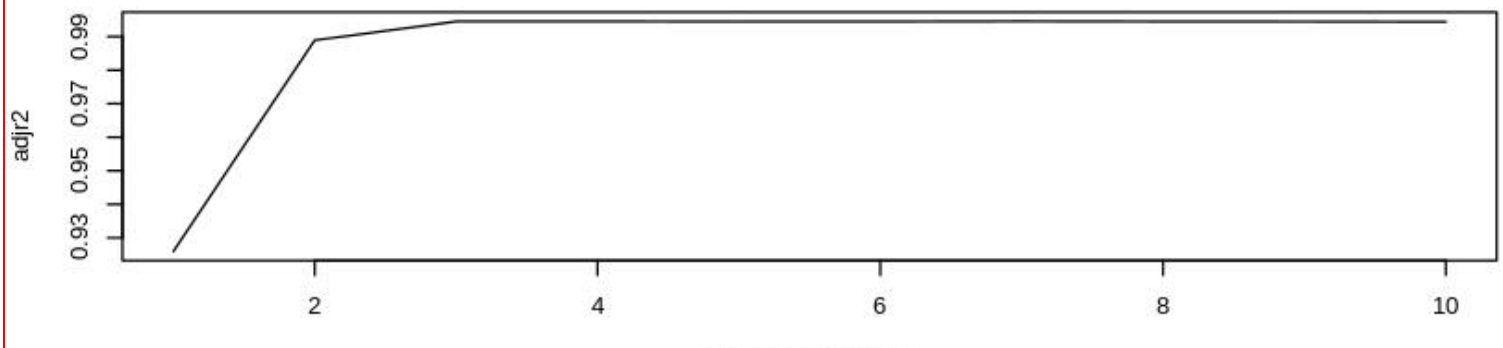
```
model_bds_summary
```

```
Subset selection object
Call: regsubsets.formula(y ~ ., data = question_3_data, nvmax = 10,
    method = "backward")
10 Variables  (and intercept)
     Forced in Forced out
x        FALSE      FALSE
x.2      FALSE      FALSE
x.3      FALSE      FALSE
x.4      FALSE      FALSE
x.5      FALSE      FALSE
x.6      FALSE      FALSE
x.7      FALSE      FALSE
x.8      FALSE      FALSE
x.9      FALSE      FALSE
x.10     FALSE      FALSE
1 subsets of each size up to 10
Selection Algorithm: backward
          x   x.2 x.3 x.4 x.5 x.6 x.7 x.8 x.9 x.10
1  ( 1 )  " " " " "*" " " " " " " " " " " " " "
2  ( 1 )  " " "*" "*" " " " " " " " " " " " " "
3  ( 1 )  "*" "*" "*" " " " " " " " " " " " " "
4  ( 1 )  "*" "*" "*" " " " " "*" " " " " " " "
5  ( 1 )  "*" "*" "*" "*" " " "*" " " " " " " "
6  ( 1 )  "*" "*" "*" "*" " " "*" " " "*" " " "
7  ( 1 )  "*" "*" "*" "*" " " "*" " " "*" " " "*"
8  ( 1 )  "*" "*" "*" "*" "*" "*" " " "*" " " "*"
9  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" " " "*"
10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

```
par(mfrow=c(3,1))
plot(model_bds_summary$cp,xlab='number of variables',ylab='cp',type='l')
plot(model_bds_summary$bic,xlab='number of variables',ylab='bic',type='l')
plot(model_bds_summary$adjr2,xlab='number of variables',ylab='adjr2',type='l')
```








```r
indices<-c(which.min(model_bds_summary$cp),which.min(model_bds_summary$bic),which.max(model_bds_summary$adjr2))

for (index in indices) {
  cat("Number of variables:",index,"\n")
  cat("Coefficients:\n")
  print(coef(model_bds,id=index))
  cat("\n")
}
```

```
Number of variables: 3
Coefficients:
(Intercept)           x         x.2         x.3
  0.9703939   1.9204462   2.9084569   4.0204363

Number of variables: 3
Coefficients:
(Intercept)           x         x.2         x.3
  0.9703939   1.9204462   2.9084569   4.0204363

Number of variables: 7
Coefficients:
(Intercept)           x         x.2         x.3         x.4         x.6
  0.7931750   1.8907484   5.2758848   4.0530514  -4.0243410   2.2138377
        x.8         x.10
 -0.4870719   0.0373538
```

All the models (Best subset, Forward Stepwise & Backward Stepwise selection method) gave the lowest Cp & BIC score for the model containing the predictors X, X^2, and X^3 and only the Best subset and Backward Stepwise selection methods gave the highest value for adjusted R^2 for the model containing the predictors X, X^2, X^3, X^4, X^6, X^8, X^{10} i.e model with 7 out of 10 predictors whereas Forward Stepwise selection method chose the model containing the predictors X, X^2, X^3, X^6 i.e model with only 4 out of 10 predictors.

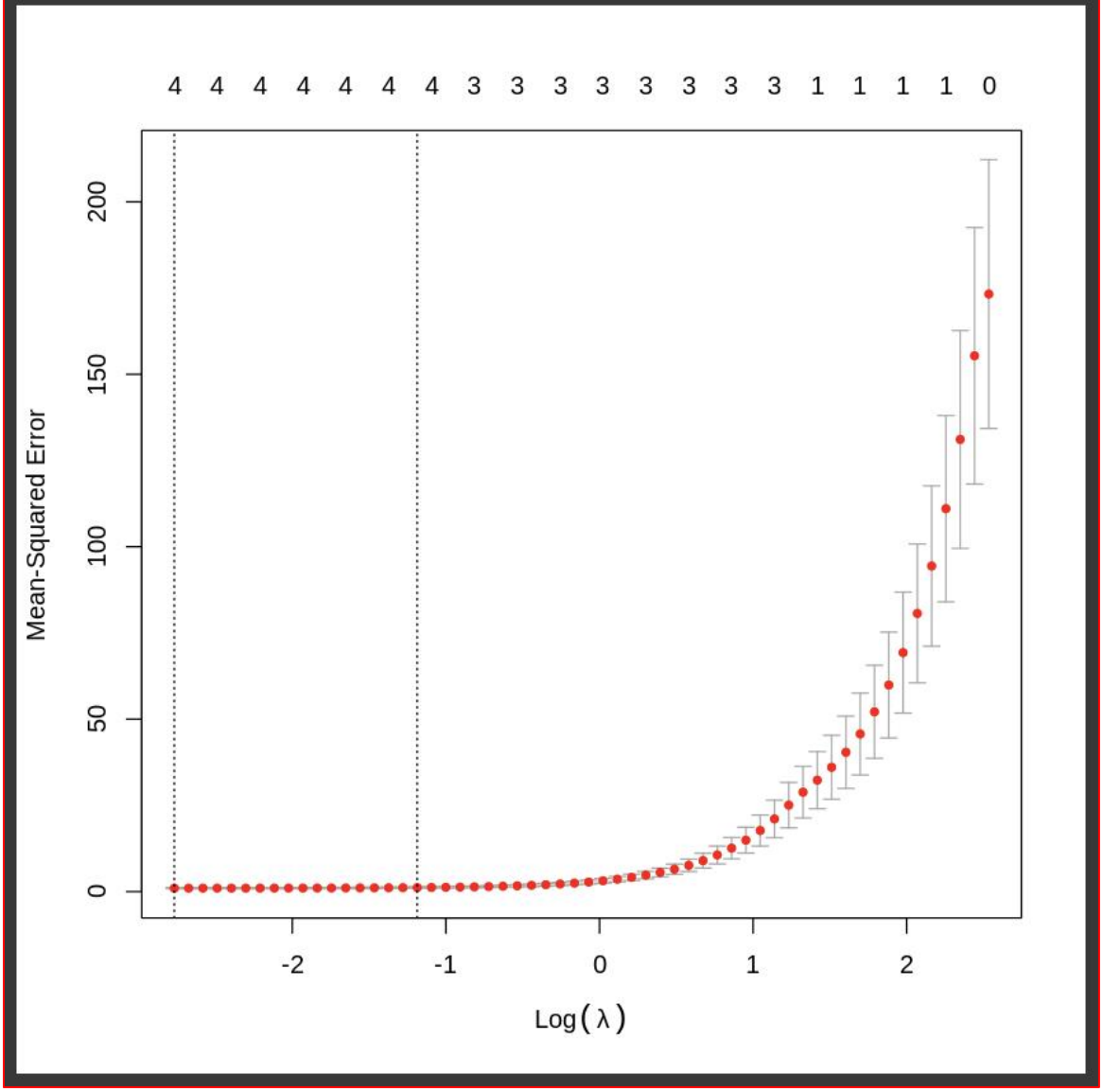
(e) Now fit a lasso model to the simulated data, again using $X, X^2, \ldots, X^{10}$ as predictors. Use cross-validation to select the optimal value of λ. Create plots of the cross-validation error as a function of λ. Report the resulting coefficient estimates, and discuss the results obtained.

```
library(glmnet)
```

```
xmat=model.matrix(y~.,data=question_3_data)[,-1]
model_lasso=cv.glmnet(xmat,y,alpha=1)
model_lasso_best_lambda=model_lasso$lambda.min
print(model_lasso_best_lambda)
```

```
0.06276895
```

```
1  plot(model_lasso)
```



```
1  model_best_lasso=glmnet(xmat,y,alpha=1)
2  predict(model_best_lasso,s=model_lasso_best_lambda,type="coefficients")
```

```
11 x 1 sparse Matrix of class "dgCMatrix"
                    s1
(Intercept) 1.07946949
x           1.89142726
x.2         2.66822407
x.3         4.01088941
x.4         0.04840102
x.5         .
x.6         .
x.7         .
x.8         .
x.9         .
x.10        .
```

After performing the cross-validation using the lasso method we find that lasso puts more weight on X, X^2, X^3, and X^4 predictors stating that other predictors don't contribute much to the prediction of response Y.

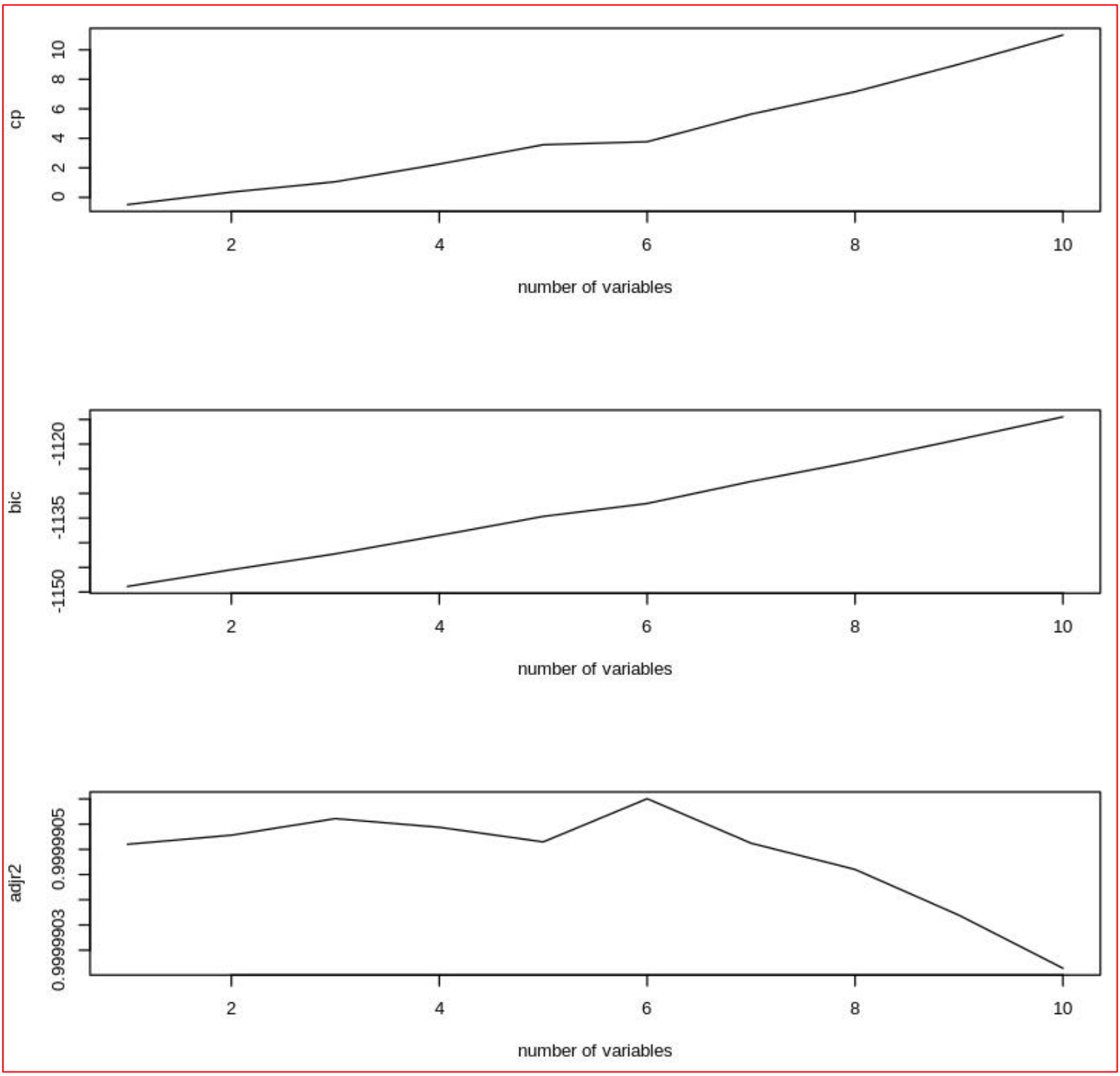
(f) Now generate a response vector Y according to the model $Y = \beta_0 + \beta_7 X^7 + \varepsilon$, and perform best subset selection and the lasso. Discuss the results obtained.

```
beta_7=6

y = beta_0 + (beta_7 * (x^7)) + e_noise

question_3_f_data=data.frame(x,x^2,x^3,x^4,x^5,x^6,x^7,x^8,x^9,x^10,y)
```

```
# best subset selection method
model_bss_q3f=regsubsets(y~.,data=question_3_f_data,nvmax=10)

model_bss_q3f_summary=summary(model_bss_q3f)
model_bss_q3f_summary
```

```
Subset selection object
Call: regsubsets.formula(y ~ ., data = question_3_f_data, nvmax = 10)
10 Variables  (and intercept)
     Forced in Forced out
x        FALSE      FALSE
x.2      FALSE      FALSE
x.3      FALSE      FALSE
x.4      FALSE      FALSE
x.5      FALSE      FALSE
x.6      FALSE      FALSE
x.7      FALSE      FALSE
x.8      FALSE      FALSE
x.9      FALSE      FALSE
x.10     FALSE      FALSE
1 subsets of each size up to 10
Selection Algorithm: exhaustive
          x   x.2 x.3 x.4 x.5 x.6 x.7 x.8 x.9 x.10
1  ( 1 )  " " " " " " " " " " " " "*" " " " " " "
2  ( 1 )  " " "*" " " " " " " " " " " " " " " " "
3  ( 1 )  " " "*" " " " " " " " " "*" "*" " " " " " "
4  ( 1 )  " " " " " " " "*" " " "*" "*" "*" " " " " "
5  ( 1 )  " " "*" " " " " "*" " " "*" "*" "*" " " " " "
6  ( 1 )  " " "*" " " " " "*" " " "*" "*" "*" " " " " "*"
7  ( 1 )  "*" "*" " " " " "*" " " "*" "*" "*" " " " " "*"
8  ( 1 )  " " "*" "*" "*" " " " " "*" "*" "*" "*" "*"
9  ( 1 )  "*" "*" "*" "*" "*" "*" "*" "*" " " " " "*"
10  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"
```

```r
par(mfrow=c(3,1))
plot(model_bss_q3f_summary$cp,xlab='number of variables',ylab='cp',type='l')
plot(model_bss_q3f_summary$bic,xlab='number of variables',ylab='bic',type='l')
plot(model_bss_q3f_summary$adjr2,xlab='number of variables',ylab='adjr2',type='l')
```



```r
indices<-c(which.min(model_bss_q3f_summary$cp),which.min(model_bss_q3f_summary$bic),which.max(model_bss_q3f_summary$adjr2))

for (index in indices) {
  cat("Number of variables:",index,"\n")
  cat("Coefficients:\n")
  print(coef(model_bss_q3f,id=index))
  cat("\n")
}
```

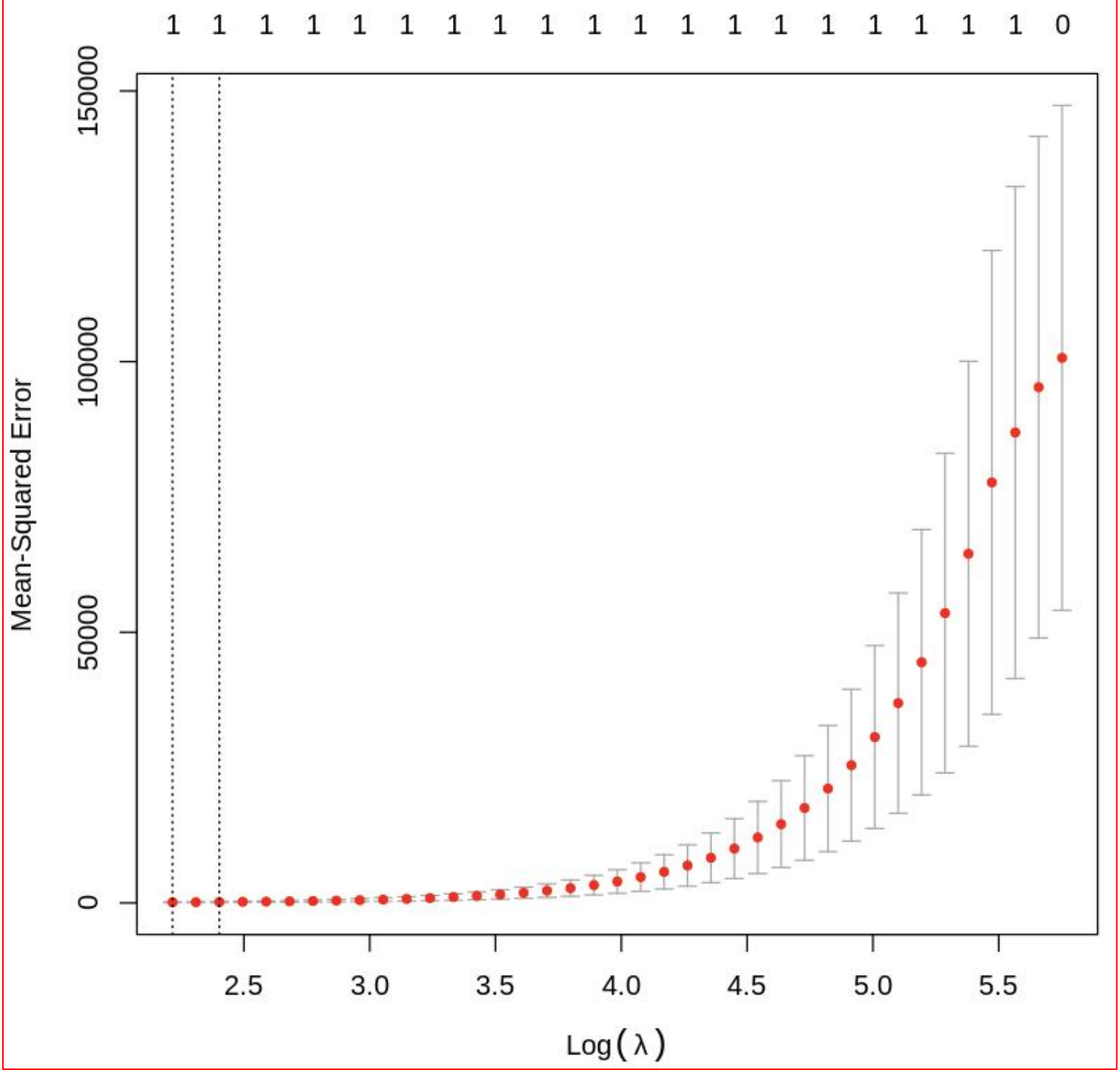
```
Number of variables: 1
Coefficients:
(Intercept)         x.7
  0.8932512   5.9997045

Number of variables: 1
Coefficients:
(Intercept)         x.7
  0.8932512   5.9997045

Number of variables: 6
Coefficients:
(Intercept)         x.2         x.4         x.6         x.7         x.8
 0.79358642  2.30456710 -4.10732161  2.27693175  6.00151236 -0.50477246
       x.10
 0.03899314
```

```r
# lasso selection method
xmat=model.matrix(y~.,data=question_3_f_data)[,-1]
model_lasso=cv.glmnet(xmat,y,alpha=1)
model_lasso_best_lambda=model_lasso$lambda.min
print(model_lasso_best_lambda)
```

```r
model_best_lasso=glmnet(xmat,y,alpha=1)
predict(model_best_lasso,s=model_lasso_best_lambda,type="coefficients")
```

```
11 x 1 sparse Matrix of class "dgCMatrix"
                    s1
(Intercept) 1.365508
x           .
x.2         .
x.3         .
x.4         .
x.5         .
x.6         .
x.7         5.824810
x.8         .
x.9         .
x.10        .
```

```
1   plot(model_lasso)
```



The Best subset selection method found that for Cp and BIC score it was minimum for predictor X^3 and for maximum adjusted R^2 the best model was the one with 6 predictors namely X, X^2, X^3, X^5, X^6 and X^9. However, after performing the cross-validation using the lasso method, we find that lasso puts more weight only on the X^7 predictor stating that other predictors don't contribute much to the prediction of response Y.

4. In this exercise, we will predict the number of applications received using the other variables in the College data set.

```
library(ISLR)

set.seed(123)

sum(is.na(College))
```

(a) Split the data set into a training set and a test set.

```
round((nrow(College)*0.2),0)
```

```
155
```

```
test_index=sample(nrow(College),round((nrow(College)*0.2),0),replace=FALSE)
train_set=College[-test_index,]
test_set=College[test_index,]
```

```
c(nrow(train_set),nrow(test_set),nrow(train_set)+nrow(test_set),nrow(College))
```

```
622 · 155 · 777 · 777
```

```
names(College)
```

```
'Private' · 'Apps' · 'Accept' · 'Enroll' · 'Top10perc' · 'Top25perc' · 'F.Undergrad' · 'P.Undergrad' · 'Outstate' · 'Room.Board' · 'Books' · 'Personal' · 'PhD' · 'Terminal' · 'S.F.Ratio' · 'perc.alumni' · 'Expend' · 'Grad.Rate'
```

(b) Fit a linear model using least squares on the training set, and report the test error obtained.

```
lm_model=lm(Apps~.,data=train_set)
lm_model_pred=predict(lm_model,test_set)
# RSS
print(mean((test_set[,"Apps"]-lm_model_pred)^2))
```

```
[1] 726062.2
```

(c) Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

```
library(glmnet)
```

```
train_xmat=model.matrix(Apps~.,data=train_set)
test_xmat=model.matrix(Apps~.,data=test_set)
```

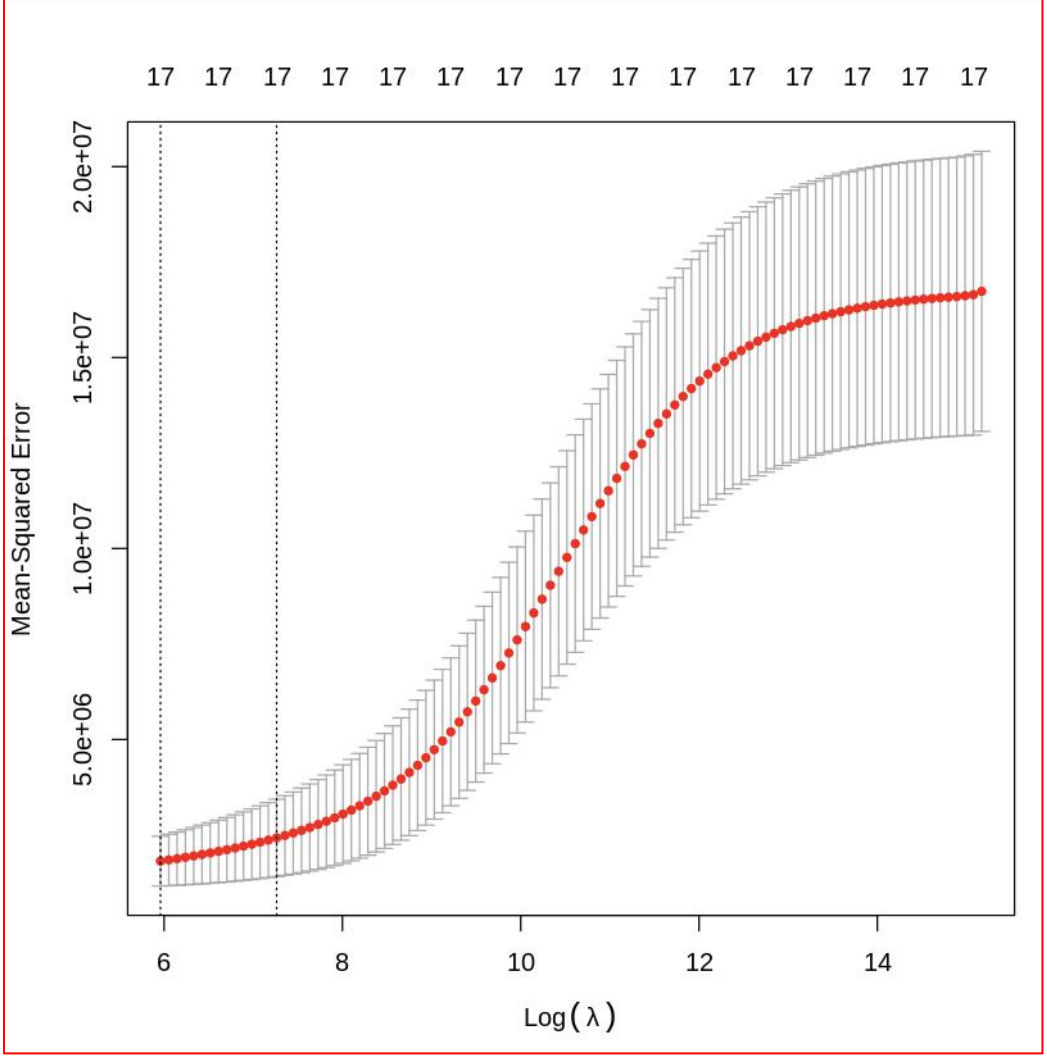
```
# Ridge
model_ridge=cv.glmnet(train_xmat,train_set['Apps'],alpha=0)
model_ridge_best_lambda=model_ridge$lambda.min
print(model_ridge_best_lambda)
```

```
[1] 387.5921
```

```
ridge_model_pred=predict(model_ridge,test_xmat,s=model_ridge_best_lambda)
# RSS for Ridge
print(mean((test_set["Apps"]-ridge_model_pred)^2))
```

```
799031.9
```

```
plot(model_ridge)
```



(d) Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```
# Lasso
model_lasso=cv.glmnet(train_xmat,train_set[,'Apps'],alpha=1)
model_lasso_best_lambda=model_lasso$lambda.min
print(model_lasso_best_lambda)
```

```
12.11503
```

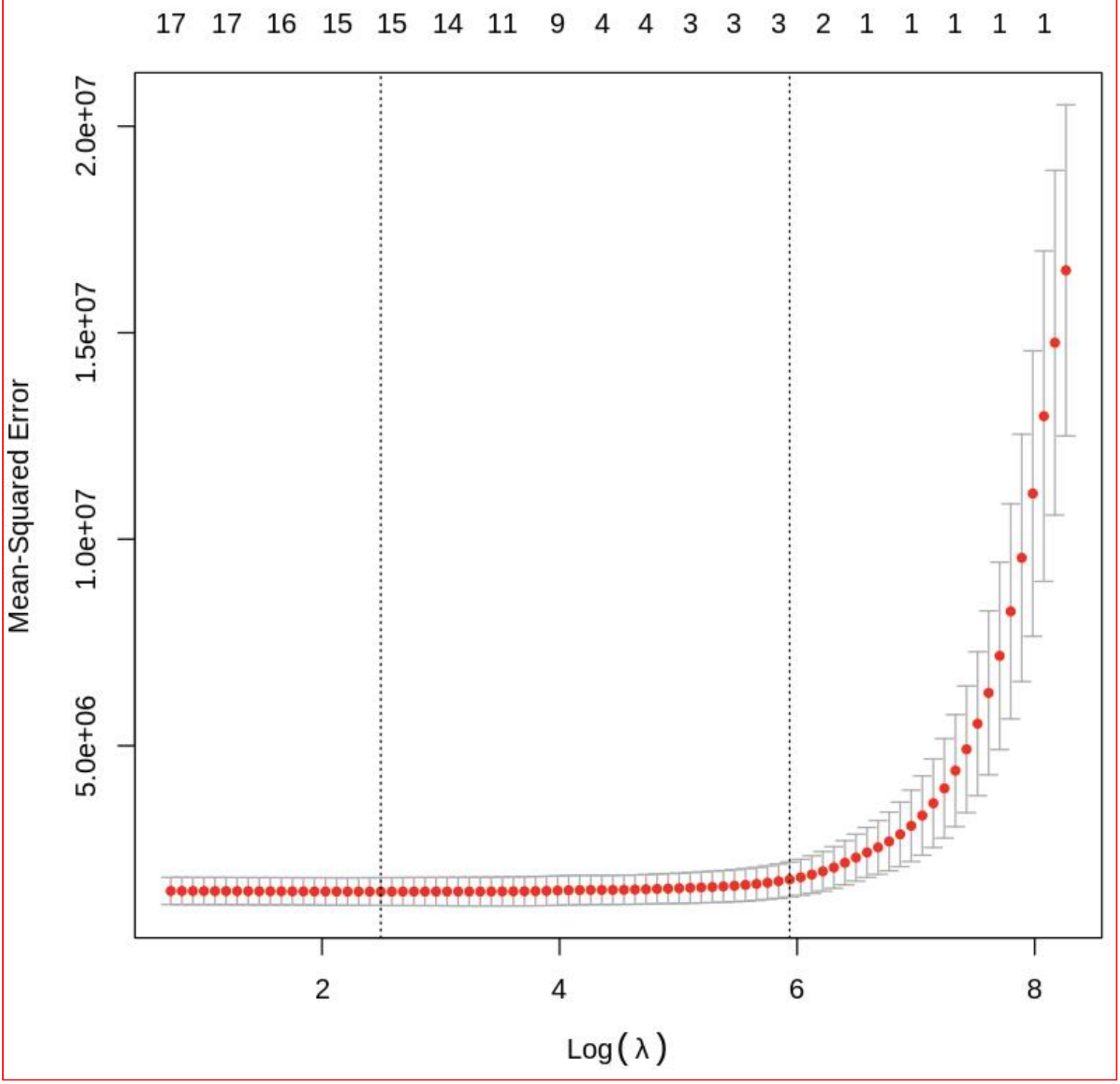
```
lasso_model_pred=predict(model_lasso,test_xmat,s=model_lasso_best_lambda)
# RSS for Lasso
print(mean((test_set[,"Apps"]-lasso_model_pred)^2))
```

```
750801.5
```

```
1  predict(model_lasso,test_xmat,s=model_lasso_best_lambda,type='coefficients')
```

```
19 x 1 sparse Matrix of class "dgCMatrix"
                      s1
(Intercept) -6.183583e+02
(Intercept)  .
PrivateYes  -4.079245e+02
Accept       1.527555e+00
Enroll      -4.085465e-01
Top10perc    3.813867e+01
Top25perc   -5.484379e+00
F.Undergrad  .
P.Undergrad  3.469064e-02
Outstate    -7.896075e-02
Room.Board   1.138274e-01
Books        1.426352e-01
Personal     2.533014e-03
PhD         -6.384711e+00
Terminal    -2.743584e+00
S.F.Ratio    5.759028e+00
perc.alumni  .
Expend       7.859546e-02
Grad.Rate    7.305107e+00
```

```
1   plot(model_lasso)
```



(e) Fit a PCR model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation.

```
1  library(pls)
```

```
Attaching package: 'pls'


The following object is masked from 'package:stats':

    loadings
```

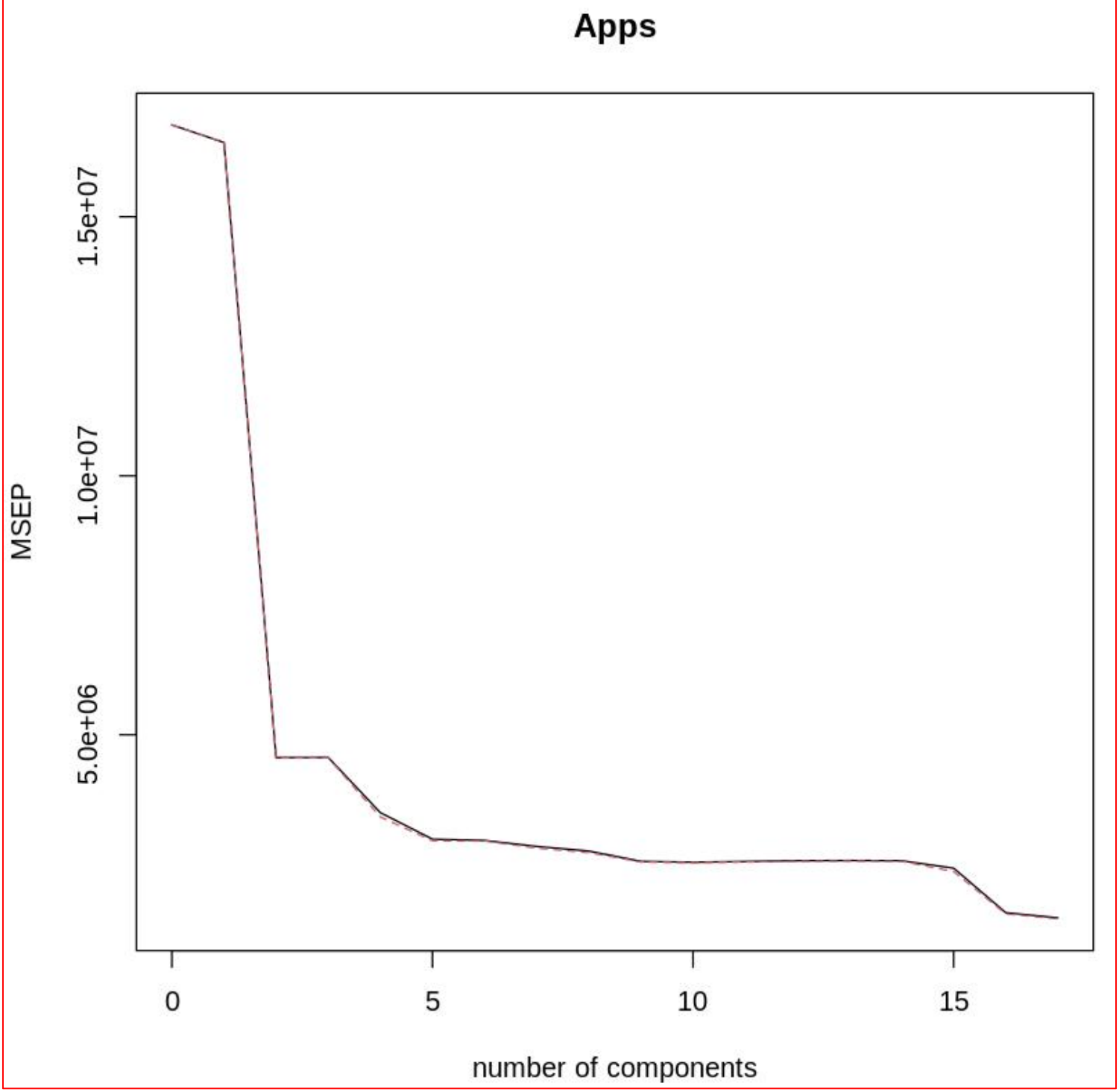
```
1  set.seed(123)
```

```
1  model_pcr=pcr(Apps~.,data=train_set,scale=TRUE,validation="CV")
```

```
1  summary(model_pcr)
```

```
Data:   X dimension: 622 17
        Y dimension: 622 1
Fit method: svdpc
Number of components considered: 17

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV            4096     4054     2135     2136     1870     1728     1720
adjCV         4096     4054     2133     2134     1847     1719     1717
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV        1686     1660     1599      1594      1598      1603      1604
adjCV     1677     1651     1595      1591      1595      1599      1601
       14 comps  15 comps  16 comps  17 comps
CV         1603      1557      1251      1211
adjCV      1599      1536      1242      1202

TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X      30.699    56.57    63.82    69.58    74.97    79.92    83.57    87.14
Apps    2.599    73.20    73.38    80.31    83.28    83.35    84.43    84.84
      9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
X       90.29     92.80     94.90     96.74     97.85     98.73     99.36
Apps    85.60     85.78     85.78     85.80     85.81     85.83     90.92
      16 comps  17 comps
X        99.84    100.00
Apps     92.80     93.15
```

```
1  validationplot(model_pcr,val.type="MSEP")
```




```
1  names(model_pcr)
```

'coefficients' · 'scores' · 'loadings' · 'Yloadings' · 'projection' · 'Xmeans' · 'Ymeans' · 'fitted.values' · 'residuals' · 'Xvar' · 'Xtotvar' · 'fit.time' · 'ncomp' · 'method' · 'center' · 'scale' · 'validation' · 'call' · 'terms' · 'model'

```
model_pcr$validation
```

3529.23722884627 · 7972.00684733083 · 5039.12191363883 · 3308.79482709097 · 16533.1812358161 · 1709.13598804866 · 1007.40992271018 · 542.128685230945 · 1444.10759381807 · 730.509112834031 · 1383.05215739374 · 1686.21727223182 · 3438.82062735123 · 1253.86519252245 · 8883.16886052385 · 12015.326726696 · 1926.0371268543 · 1047.44884471543 · -72.8654593176093 · 325.828909125968 · 4507.19263323483 · 557.642537257667 · 3035.27391799075 · 902.504881038608 · 4115.38347015494 · 4484.77744319732 · 1962.9891202398 · 1655.97537484267 · 3338.42600729071 · 1266.53044352338 · 10513.0239755484 · 1744.66081581826 · 1993.73959707934 · 5166.14105529183 · 2706.9813173143 · 471.218508313124 · 1139.50984368088 · 615.292113451915 · 1723.56342380075 · 2321.53803655795 · 2212.04538431593 · 637.56085540921 · 1496.29465765044 · 1674.24857948719 · 1704.87417025867 · 2174.30812133138 · 499.661515128412 · 345.41194326638 · 3974.31701344962 · -103.826497963051 · 1058.7857493162 · 2692.15176175419 · 19.7390187031174 · 2531.17673590743 · 1888.20181517252 · 3706.19026428798 · 2671.34169932398 · 3183.19725119152 · 2919.75077479317

$coefficients
NULL

$gammas
NULL

$PRESS0
Apps: 10433066442.2424

$PRESS

A matrix: 1 × 17 of type dbl

	1 comps	2 comps	3 comps	4 comps	5 comps	6 comps	7 comps	8 comps	9 comps	10 comps	11 comps	12 comps	13 comps	14 comps	15 comps	16 comps
Apps	10221327994	2836331306	2837640890	2174425480	1856403269	1841060167	1768814640	1713568318	1590008381	1580372354	1588545069	1597583639	1600240595	1597348814	1508141487	973559455

$adj

A matrix: 1 × 17 of type dbl

	1 comps	2 comps	3 comps	4 comps	5 comps	6 comps	7 comps	8 comps	9 comps	10 comps	11 comps	12 comps	13 comps	14 comps	15 comps	16 comps	17 comps
Apps	16279889	4489677	4457159	3374094	2825616	2795139	2635904	2564797	2420648	2386627	2387357	2385472	2382463	2379599	1583036	1227043	1165422

$segments

$V1
471 · 48 · 481 · 8 · 531 · 565 · 376 · 570 · 407 · 321 · 151 · 410 · 350 · 600 · 604 · 144 · 28 · 476 · 422 · 555 · 27 · 461 · 506 · 78 · 115 · 164 · 282 · 230 · 91 · 23 · 548 · 427 · 236 · 606 · 402 · 446 · 95 · 12 · 349 · 553 · 360 · 136 · 47 · 179 · 362 · 283 · 263 · 429 · 547 · 605 · 88 · 273 · 249 · 279 · 552 · 334 · 431 · 259 · 260 · 59 · 254 · 81 · 219

$V2
227 · 482 · 250 · 258 · 488 · 329 · 84 · 200 · 595 · 9 · 622 · 472 · 314 · 142 · 573 · 308 · 157 · 439 · 152 · 480 · 201 · 327 · 504 · 158 · 238 · 382 · 292 · 223 · 89 · 216 · 435 · 515 · 24 · 239 · 45 · 202 · 498 · 128 · 354 · 507 · 1 · 585 · 549 · 578 · 15 · 54 · 13 · 558 · 590 · 160 · 588 · 325 · 31 · 317 · 306 · 309 · 141 · 405 · 120 · 340 · 228 · 196 · 131

$V3
315 · 248 · 597 · 262 · 271 · 339 · 617 · 237 · 391 · 281 · 344 · 572 · 217 · 229 · 86 · 559 · 300 · 372 · 208 · 513 · 211 · 366 · 364 · 156 · 612 · 567 · 448 · 101 · 143 · 277 · 154 · 181 · 270 · 557 · 520 · 37 · 244 · 176 · 501 · 320 · 601 · 621 · 71 · 524 · 177 · 73 · 414 · 603 · 189 · 577 · 359 · 301 · 492 · 430 · 502 · 433 · 452 · 68 · 14 · 499 · 442 · 82

$V4
243 · 105 · 330 · 529 · 398 · 583 · 554 · 591 · 290 · 40 · 168 · 224 · 184 · 375 · 434 · 35 · 310 · 546 · 75 · 100 · 475 · 34 · 215 · 399 · 138 · 335 · 313 · 145 · 123 · 275 · 419 · 16 · 540 · 522 · 367 · 495 · 449 · 358 · 102 · 337 · 512 · 294 · 284 · 107 · 97 · 22 · 134 · 550 · 274 · 246 · 118 · 178 · 21 · 85 · 66 · 108 · 261 · 374 · 121 · 392 · 267 · 497

$V5
265 · 171 · 195 · 206 · 218 · 247 · 194 · 380 · 287 · 148 · 161 · 592 · 608 · 64 · 17 · 438 · 153 · 496 · 87 · 425 · 418 · 381 · 390 · 186 · 462 · 474 · 3 · 133 · 413 · 420 · 569 · 403 · 109 · 467 · 83 · 511 · 491 · 288 · 613 · 543 · 443 · 150 · 336 · 357 · 338 · 20 · 453 · 571 · 599 · 233 · 33 · 464 · 62 · 445 · 296 · 289 · 38 · 245 · 389 · 345 · 485 · 57

$V6
535 · 459 · 99 · 199 · 155 · 370 · 460 · 616 · 466 · 18 · 421 · 400 · 451 · 242 · 137 · 26 · 285 · 556 · 94 · 25 · 191 · 517 · 103 · 286 · 204 · 174 · 386 · 371 · 30 · 328 · 441 · 240 · 190 · 46 · 477 · 6 · 4 · 377 · 394 · 395 · 52 · 415 · 117 · 165 · 519 · 114 · 92 · 416 · 307 · 505 · 125 · 50 · 463 · 104 · 135 · 257 · 276 · 331 · 113 · 412 · 368 · 106

$V7
112 · 516 · 98 · 607 · 209 · 530 · 77 · 5 · 566 · 579 · 444 · 39 · 56 · 280 · 163 · 269 · 483 · 593 · 304 · 437 · 343 · 303 · 80 · 500 · 494 · 426 · 11 · 369 · 393 · 197 · 564 · 582 · 116 · 332 · 298 · 252 · 319 · 188 · 611 · 455 · 7 · 436 · 147 · 423 · 60 · 542 · 450 · 322 · 586 · 212 · 609 · 251 · 378 · 373 · 574 · 232 · 132 · 509 · 563 · 299 · 231 · 544

```
model_pcr_pred=predict(model_pcr,test_set,ncomp=17)
round((mean((test_set['Apps'] - model_pcr_pred)^2)),0)
```

```
726062
```

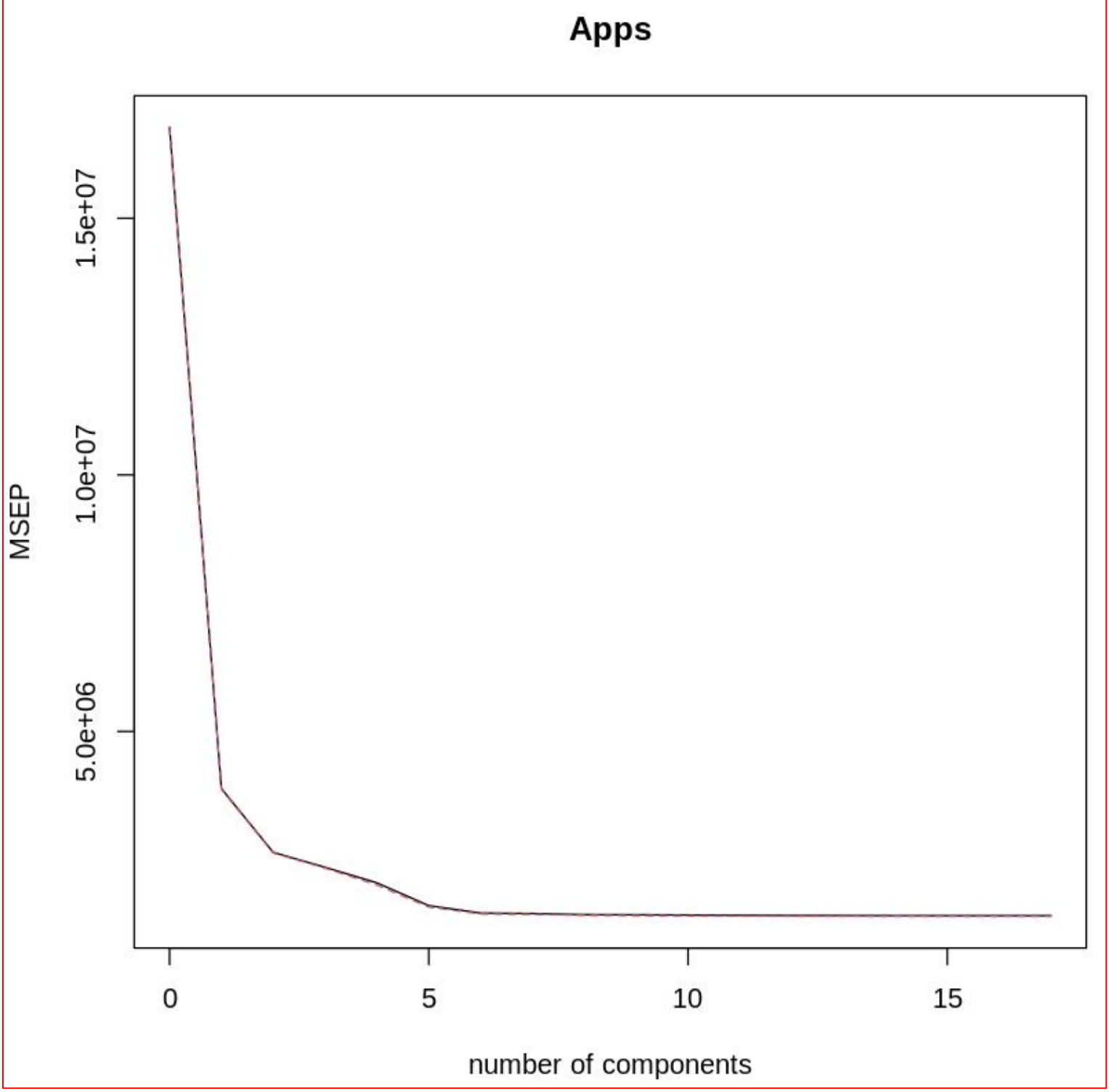
(f) Fit a PLS model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected by cross-validation.

```
1  model_plsr=plsr(Apps~.,data=train_set,scale=TRUE,validation="CV")
```

```
1  summary(model_plsr)
```

```
Data:   X dimension: 622 17
        Y dimension: 622 1
Fit method: kernelpls
Number of components considered: 17

VALIDATION: RMSEP
Cross-validated using 10 random segments.
       (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
CV            4096     1971     1626     1533     1431     1267     1208
adjCV         4096     1968     1622     1528     1416     1255     1201
       7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
CV        1205     1196     1195      1193      1190      1189      1188
adjCV     1198     1189     1187      1186      1184      1182      1181
       14 comps  15 comps  16 comps  17 comps
CV         1187      1187      1187      1187
adjCV      1181      1180      1180      1180

TRAINING: % variance explained
      1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
X       25.96    38.18    61.83    64.80    68.33    73.36    77.46    80.72
Apps    78.02    85.59    87.61    90.65    92.49    92.93    92.98    93.03
      9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
X       82.54     85.20     88.15     90.63     92.25     94.38     96.96
Apps    93.10     93.12     93.13     93.14     93.15     93.15     93.15
      16 comps  17 comps
X        97.95    100.00
Apps     93.15     93.15
```

```
1   validationplot(model_plsr,val.type="MSEP")
```



```
model_plsr$validation
```

$method
'CV'

$pred

2073.59008260573 · 57.2099396276953 · 88.9121827620693 · 2933.66928645074 · 650.294420285835 · 1423.65038223042 · 3060.97696631763 · 1534.30524991411 · -875.050469892874 · 2729.75427195453 · 2536.83701337515 · 60.8953827705304 · 740.808211878373 · 4837.86469011346 · -21.5092947986836 · 66.6255892632748 · 1980.38818134788 · 6764.77162598306 · 13509.5261099642 · 1827.21486086216 · 2862.08058834378 · 10137.3606636982 · -60.2989470563339 · 2299.36400322152 · 571.836093674192 · 2296.09740843603 · 516.171691644397 · 2919.92975120124 · -510.443526366508 · 3666.23017559483 · 4384.42080110117 · 1751.17745739585 · 817.358595742962 · 900.034220741616 · 1680.51344022965 · 1489.5300617588 · -297.830729782615 · -512.758295433986 · 3743.29755333456 · 2426.21650468125 · -1182.15141627074 · -128.474122671984 · 2431.35935540342 · 39.0660159681229 · 3880.31027391842 · -210.01145216567 · 429.527116218209 · 13057.5833142104 · 5651.33523156349 · 8697.24617778204 · 49.7001050536187 · 4915.90776757338 · 344.582457720395 · -616.553209112386 · 11581.5117416502 · 7494.75517368663 · 5727.37513319961 · 3072.99452690043 · -98.4767222994201 · 1261.18540877082 · 890.769990063968 · 7697.32867356495 · 7111.58270848894 · 2252.81409712199 · 1717.32768081584 · -2214.74392880023 · 4059.80706995919 · 6644.03777529298 · 5416.54079568104 · 2112.74666310087 · 776.908543842857 · 2984.13063093042 · -3.53144235937543 · 1050.31130653721 · 346.036944234849 · -450.887950527494 · 1793.1804341345 · 13.936832357821 · 1848.45724460864 · 3730.34987188496 · 4971.0160783294 · 4198.31588107414 · -1175.26405585634 · 3085.78303705047 · 1689.22565838588 · 1764.20322518812 · 1410.84735151323 · 626.234543412661 · -91.8865811094952 · 3842.42292328646 · 9.13340009509375 · 8281.24125434834 · 1103.14976244778 · 259.729481254802 · 3658.3625467936 · 4508.51630816866 · 995.786620085258 · 4882.48319618123 · 1027.57068252986 · 901.178978525155 · 1062.06892587167 · 1710.37562416906 · 817.183318650087 · 892.617927799959 · 1379.6568440783 · -1369.62314348334 · 4510.86742550805 · 6376.06253897075 · 2617.99982990733 · 9367.57363846405 · 901.104516186835 · 6160.53208956952 · -427.924376042134 · -213.34838119414 · 593.453007262012 · 96.9535862133516 · 3826.93860990453 · 1543.04423352658 · -576.617869221361 · 146.714758586201 · -1267.39686586329 · -944.289787753883 · 933.752886744391 · 2854.58075409339 · 1717.48205024486 · 232.437052736363 · -1191.46962378747 · 495.729365861122 · 1054.26503104095 · 4133.09875137173 · 3746.33309473857 · 1697.85104239486 · 7962.10047609445 · 2356.59604477856 · -243.942605759206 · 2640.9875287497 · 4520.32787645597 · -92.0683386913452 · -259.835025601647 · 2365.90097124608 · 1834.38828302682 · 2039.07751922954 · 634.304487034391 · 1909.57324654618 · 1643.96892440449 · 1620.11177474743 · 7740.81524820692 · 3776.05787082977 · 1354.60972518165 · 4551.36415413253 · 709.609608816299 · -498.148860669785 · 121.505921509973 · 3320.672480832 · 8851.91281376503 · 1088.80442523646 · 12178.4427681081 · -775.512136415984 · 4845.14822524977 · 4286.24275060326 · 2646.73246710708 · 910.456610621815 · 1084.87768522354 · 4344.21759202625 · 1083.32855896821 · -275.194344231767 · -303.200444326031 · 7261.77085395001 · 6678.63570754221 · 740.861449035161 · 7228.73038438624 · -372.819236030287 · 3700.78320616759 · -1653.49589015081 · 3006.83135367814 · -253.907919714269 · 2360.49580737221 · -612.326623609297 · 152.294240949243 · 5351.78026933859 · -216.409846488601 · -333.037580766891 · 2101.53331955723 · 1399.36410390799 · 2210.85200639182 · -607.626120477394 · 1684.97072464265 · 3542.7182223324 · 1546.35786964352 · 753.919622891721 · 2327.06646882332 · 8320.06179014564 · 5197.61548418899 · 90.090312085526 · 2597.91442704948 · 1812.96242004521 · 1991.54544060862 · 3328.32954924063 · 6319.50600700604 · 1617.08143587061 · … · 1401.38665678139 · 211.230377086175 · 5512.96874972635 · 905.30720963108 · 1606.99997081257 · -200.21108941412 · 282.282798140372 · 1907.40763094951 · -165.423791492452 · 2871.85019645871 · 756.420740163849 · 1219.02988788222 · 2654.43583527547 · -534.010203163249 · 2470.75207218967 · 2379.35023098967 · 1652.79132228454 · 1814.62577355054 · 397.534496381982 · 1859.33177289075 · 2371.96380531119 · 2688.4046185803 · 14029.3072435639 · 10520.8017668873 · 14382.8917536474 · 10966.7151599875 · 5014.02879804059 · 6442.86197157372 · 4496.32603987551 · 3977.6180111083 · 8496.37613310036 · 5528.85810180688 · 5152.20846853366 · 3439.60440882518 · 1714.85333624014 · 1243.84471510145 · 9966.52775084806 · 26.7279891067597 · 2329.33824297327 · 2203.03032421964 · 14591.1591403507 · 862.906977531669 · 4489.48620648048 · 662.966005103606 · 2412.08724861677 · 337.531897633703 · 2385.78133194427 · 4976.52272143229 · 3335.86550372902 · 344.573579542765 · 269.331376857032 · -155.294419014334 · 2593.58350920433 · -180.235983768056 · 3088.90360026438 · 3957.33060681546 · 19524.0423334083 · 4159.91737554586 · 701.46459416793 · 8220.59906572288 · 10557.445479329 · 1488.81806533399 · 15230.543225621 · 2627.61869952451 · 888.716171317634 · -60.8152721310253 · 14711.8316080412 · 5203.26622058384 · 4567.85649246882 · 16560.6277320492 · 8295.6418402986 · 1854.89686515611 · 7083.16005387531 · 1067.26661896641 · 4279.74976167058 · 1324.36079187922 · 971.767470940036 · 174.504848722436 · 15226.1983054792 · 17672.8789568185 · 3594.9960921061 · 8390.84444438924 · 18692.370626832 · 3726.64632193648 · 2181.95823609631 · 9020.11632741929 · 4992.8719710266 · 6662.14761000228 · 3777.1063094431 · 1927.85587461856 · -5.92081720892111 · 743.2207721564 · 10741.1800479465 · 1606.71051470696 · 10196.0774934666 · 6112.90493111247 · 5606.44776594194 · 4742.15925919449 · 2621.97515141553 · 2673.34645293456 · 5326.59789040273 · 4574.96446398733 · 6632.86755747059 · 9796.63751569144 · 4527.97212974177 · 11350.1028996748 · 4723.67657514433 · 10038.8838594692 · 2398.27927200023 · 483.121926367908 · 4530.72486557457 · 837.667400280941 · 8538.50188691285 · 12958.0582920047 · 1376.51129768868 · 2861.12843874583 · 2314.27149756348 · 912.025795163041 · 7565.43344307537 · 3936.730370639 · 13225.9351472302 · 4373.39824835584 · 1026.47501065884 · 3047.96756049347 · 2207.47345879422 · 2421.12148136544 · 6793.21884765186 · 8158.70363801863 · 9146.23101766974 · 10723.4319256673 · 2343.84724279661 · 790.045786096312 · 4838.08265267316 · 2413.83595133003 · 15537.3417583405 · 4848.29566117498 · -190.11454774273 · 1965.48092049776 · 465.219688557757 · 379.59981414285 · 3189.83240547675 · 3555.53390149075 · 8034.8645320058 · 4932.2041096781 · 3231.45394882419 · 16913.801294608 · 1697.34743665767 · 982.66064301264 · 453.524807502577 · 1545.20732192982 · 811.71335452225 · 1445.25474793952 · 1572.59681092713 · 3154.41729314318 · 1272.06943744317 · 8833.87168898124 · 11760.0640588486 · 2010.54165040329 · 1042.86457592954 · -235.671952889562 · 274.733209550702 · 4114.69539526113 · 509.830346913673 · 2687.79472031583 · 956.041807038659 · 3955.99244391938 · 4511.07248880822 · 2021.68086748147 · 1664.09170741993 · 3241.51552501265 · 1344.02899778168 · 10605.7943031588 · 1714.94474073681 · 2079.77607485227 · 5289.32086313067 · 2732.77191190435 · 499.494661780044 · 1167.78163298595 · 604.953328190718 · 1994.87192561236 · 2410.08539744206 · 1976.54024857554 · 678.498044753921 · 1378.16792475175 · 1679.55064179461 · 1663.81827925231 · 2330.25366016752 · 481.810556227269 · 441.436432282926 · 4243.70200630799 · -123.203612609581 · 1071.26758968883 · 2732.09074364256 · 24.2182114644347 · 2592.03736172935 · 1712.92688752068 · 3759.01818102303 · 2697.19844771165 · 3263.52094856016 · 2901.81286570455

$coefficients
NULL

$gammas
NULL

$PRESS0
Apps: 10433066442.2424

$PRESS

A matrix: 1 × 17 of type dbl

	1 comps	2 comps	3 comps	4 comps	5 comps	6 comps	7 comps	8 comps	9 comps	10 comps	11 comps	12 comps	13 comps	14 comps	15 comps	16 comps	17 comps
Apps	2416784138	1643495381	1462478268	1273898165	997760139	908050267	902532919	889400868	887550310	884723067	881210878	878661551	877937594	876717149	876431156	876408362	876437949

$adj

A matrix: 1 × 17 of type dbl

	1 comps	2 comps	3 comps	4 comps	5 comps	6 comps	7 comps	8 comps	9 comps	10 comps	11 comps	12 comps	13 comps	14 comps	15 comps	16 comps	17 comps
Apps	3689126	2421610	2087662	1606026	1284484	1199941	1189753	1179679	1170930	1167448	1164582	1162699	1161976	1161680	1161616	1161611	1161616

$segments

$V1
469 · 76 · 423 · 251 · 321 · 314 · 331 · 280 · 246 · 511 · 199 · 183 · 266 · 539 · 397 · 569 · 102 · 501 · 463 · 224 · 317 · 345 · 608 · 245 · 322 · 376 · 310 · 464 · 453 · 302 · 621 · 416 · 577 · 609 · 489 · 594 · 176 · 25 · 395 · 494 · 557 · 335 · 410 · 582 · 110 · 6 · 346 · 57 · 466 · 3 · 21 · 563 · 148 · 145 · 193 · 213 · 82 · 236 · 24 · 178 · 479 · 200 · 18

$V2
70 · 579 · 249 · 333 · 214 · 407 · 353 · 46 · 598 · 242 · 443 · 446 · 325 · 533 · 448 · 434 · 104 · 618 · 601 · 522 · 123 · 284 · 480 · 12 · 165 · 26 · 341 · 390 · 543 · 391 · 202 · 263 · 360 · 13 · 34 · 503 · 573 · 499 · 15 · 477 · 572 · 316 · 157 · 197 · 146 · 394 · 117 · 537 · 312 · 265 · 587 · 370 · 339 · 401 · 271 · 133 · 105 · 452 · 439 · 141 · 286 · 357 · 14

$V3
276 · 532 · 59 · 364 · 597 · 505 · 328 · 27 · 546 · 223 · 191 · 551 · 261 · 139 · 185 · 329 · 124 · 383 · 343 · 270 · 351 · 576 · 510 · 371 · 275 · 173 · 382 · 589 · 50 · 11 · 211 · 179 · 330 · 201 · 516 · 393 · 490 · 290 · 571 · 488 · 313 · 109 · 388 · 367 · 228 · 550 · 467 · 91 · 90 · 19 · 384 · 584 · 355 · 459 · 365 · 126 · 125 · 156 · 1 · 159 · 295 · 332

$V4
465 · 256 · 581 · 175 · 474 · 115 · 17 · 396 · 255 · 526 · 30 · 413 · 512 · 320 · 350 · 491 · 450 · 283 · 85 · 362 · 622 · 205 · 406 · 44 · 545 · 568 · 363 · 229 · 68 · 216 · 549 · 192 · 235 · 417 · 361 · 260 · 400 · 239 · 586 · 88 · 529 · 221 · 155 · 298 · 482 · 523 · 131 · 111 · 485 · 262 · 606 · 366 · 348 · 174 · 127 · 240 · 292 · 476 · 500 · 418 · 93 · 531

$V5
75 · 602 · 137 · 234 · 521 · 294 · 380 · 49 · 457 · 232 · 580 · 495 · 619 · 101 · 287 · 496 · 129 · 42 · 300 · 304 · 555 · 438 · 60 · 356 · 103 · 10 · 147 · 293 · 471 · 519 · 592 · 16 · 540 · 441 · 308 · 167 · 427 · 398 · 342 · 535 · 28 · 528 · 305 · 481 · 128 · 267 · 281 · 389 · 552 · 210 · 596 · 2 · 222 · 315 · 599 · 158 · 567 · 378 · 180 · 620 · 231 · 404

$V6
296 · 132 · 198 · 272 · 67 · 542 · 408 · 583 · 354 · 37 · 181 · 51 · 241 · 604 · 507 · 187 · 402 · 451 · 190 · 89 · 392 · 233 · 419 · 422 · 94 · 368 · 403 · 442 · 493 · 254 · 169 · 47 · 437 · 386 · 98 · 277 · 258 · 163 · 252 · 161 · 134 · 570 · 72 · 152 · 359 · 142 · 43 · 227 · 520 · 301 · 561 · 483 · 574 · 209 · 472 · 553 · 610 · 250 · 116 · 544 · 595 · 514

$V7
405 · 4 · 297 · 143 · 425 · 502 · 135 · 225 · 203 · 138 · 189 · 575 · 122 · 447 · 226 · 186 · 71 · 412 · 421 · 259 · 492 · 307 · 373 · 220 · 56 · 38 · 445 · 436 · 429 · 470 · 92 · 61 · 454 · 593 · 324 · 282 · 611 · 326 · 387 · 556 · 562 · 517 · 566 · 461 · 334 · 323 · 600 · 615 · 160 · 119 · 212 · 269 · 291 · 248 · 509 · 435 · 430 · 536 · 285 · 9 · 78 · 112

$V8
58 · 217 · 415 · 347 · 32 · 154 · 195 · 77 · 473 · 603 · 426 · 385 · 114 · 238 · 558 · 86 · 585 · 33 · 96 · 374 · 188 · 31 · 311 · 444 · 144 · 219 · 369 · 45 · 55 · 273 · 460 · 153 · 433 · 432 · 440 · 151 · 22 · 468 · 5 · 497 · 264 · 244 · 372 · 196 · 41 · 80 · 560 · 303 · 253 · 420 · 120 · 54 · 414 · 65 · 171 · 344 · 84 · 588 · 534 · 607 · 377 · 52

$V9
83 · 81 · 547 · 150 · 309 · 39 · 207 · 20 · 113 · 274 · 182 · 162 · 130 · 99 · 559 · 431 · 456 · 379 · 40 · 279 · 455 · 230 · 508 · 177 · 108 · 399 · 64 · 617 · 121 · 74 · 48 · 208 · 484 · 288 · 194 · 358 · 409 · 319 · 206 · 218 · 449 · 614 · 338 · 524 · 613 · 337 · 375 · 62 · 107 · 29 · 35 · 118 · 168 · 268 · 458 · 478 · 257 · 349 · 69 · 166 · 612 · 95

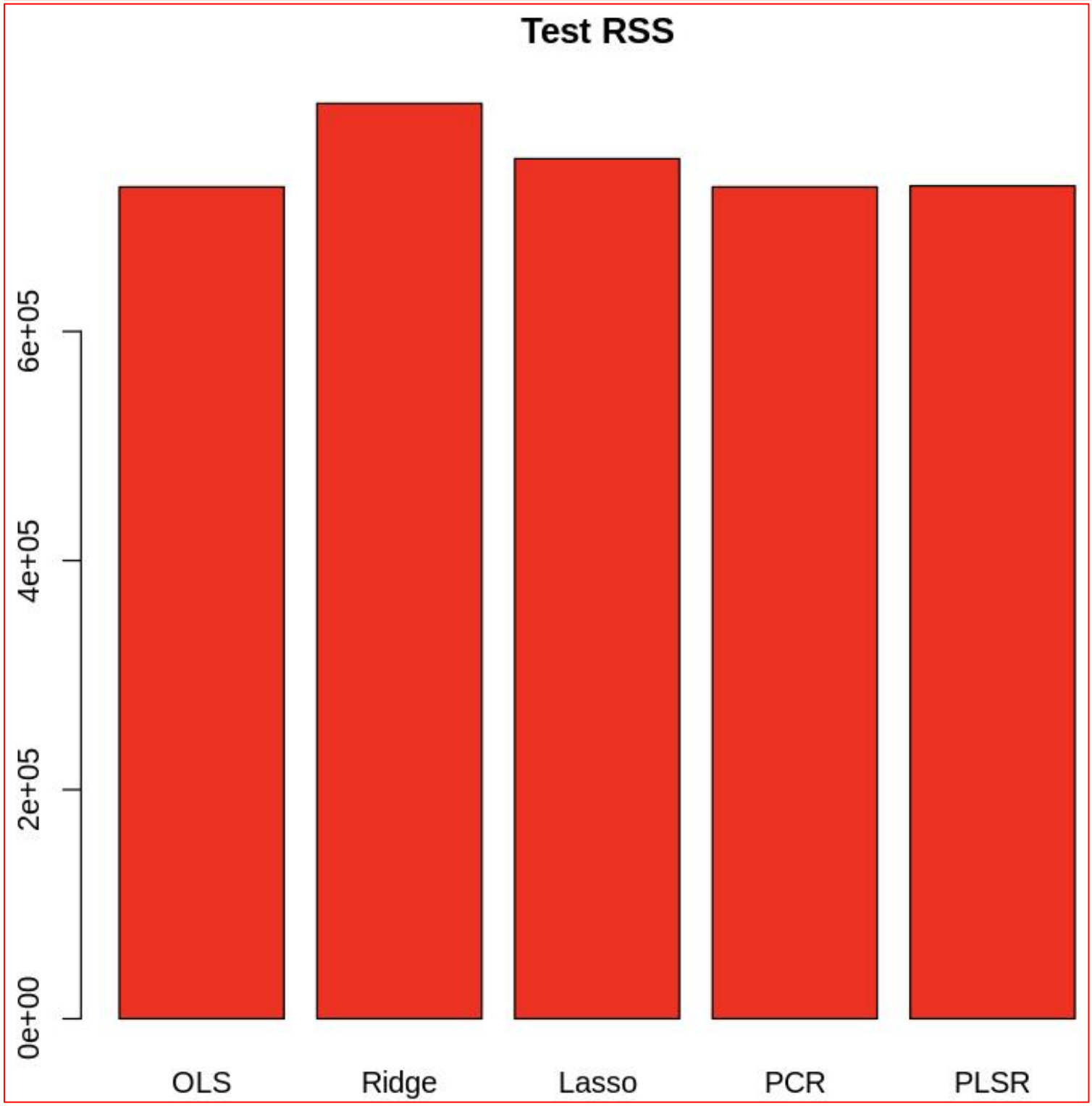
$V10
299 · 548 · 73 · 136 · 23 · 352 · 475 · 184 · 318 · 140 · 515 · 554 · 411 · 513 · 538 · 506 · 36 · 106 · 79 · 486 · 525 · 591 · 278 · 172 · 462 · 215 · 381 · 243 · 164 · 336 · 237 · 424 · 7 · 247 · 149 · 204 · 498 · 170 · 487 · 87 · 8 · 616 · 63 · 428 · 289 · 306 · 541 · 605 · 518 · 564 · 504 · 527 · 530 · 97 · 340 · 53 · 590 · 565 · 327 · 66 · 578 · 100

```
model_plsr_pred=predict(model_plsr,test_set,ncomp=13)
round((mean((test_set[,'Apps'] - model_plsr_pred)^2)),0)
```

727135

(g) Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

```
barplot(c(726062.2,799031.9,750801.5,726062,727135), col="red", names.arg=c("OLS","Ridge","Lasso","PCR","PLSR"),main="Test RSS")
```



Based on the given graph we can see that Ridge has the highest RSS (799031.9) as compared to others which is followed by Lasso (750801.5), then PLS (727135) and lastly Linear Model and PCR gave almost the same results for RSS with 726062 and 726062.2 respectively. Based on this we can Use any of the models except Ridge and Lasso.