Data 3 HW 3

Sean Duan

9/28/2020

1.

A

```r
train=sample(c(TRUE ,FALSE), nrow(College),rep=TRUE)
test=(!train)
```

B

```r
p1_m1<-lm(Apps~.,data=College, subset = train)
summary(p1_m1)
```

```
##
## Call:
## lm(formula = Apps ~ ., data = College, subset = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2937.0  -478.3    -7.8   376.1  7157.3
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    396.92024  626.52893   0.634 0.526800
## PrivateYes    -532.84007  202.35365  -2.633 0.008828 **
## Accept           1.80677    0.05269  34.290  < 2e-16 ***
## Enroll          -1.31801    0.29783  -4.425 1.28e-05 ***
## Top10perc       58.43136    9.04712   6.459 3.48e-10 ***
## Top25perc      -20.83759    7.12699  -2.924 0.003680 **
## F.Undergrad      0.06578    0.05009   1.313 0.189951
## P.Undergrad      0.04089    0.04103   0.997 0.319582
## Outstate        -0.09561    0.02848  -3.357 0.000872 ***
## Room.Board       0.05981    0.07343   0.815 0.415899
## Books            0.03668    0.36416   0.101 0.919820
## Personal        -0.05762    0.09052  -0.637 0.524790
## PhD            -11.52997    6.92686  -1.665 0.096888 .
## Terminal        -2.27361    7.52405  -0.302 0.762692
## S.F.Ratio        2.81588   22.73975   0.124 0.901519
## perc.alumni     -1.83184    5.82601  -0.314 0.753383
## Expend           0.08903    0.02622   3.396 0.000761 ***
## Grad.Rate        9.51661    4.12105   2.309 0.021502 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1067 on 355 degrees of freedom
## Multiple R-squared:  0.9377, Adjusted R-squared:  0.9347
## F-statistic:   314 on 17 and 355 DF,  p-value: < 2.2e-16
```

```
print("MSE below")
```

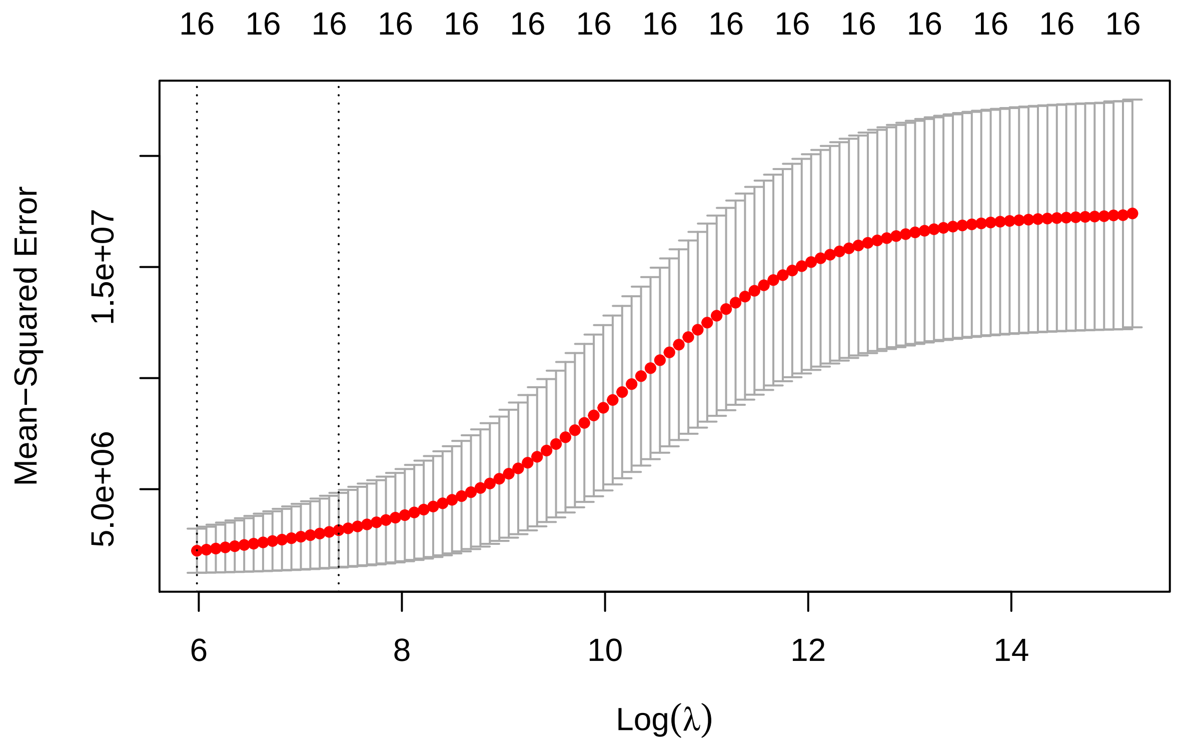
```
## [1] "MSE below"
```

```
mean(p1_m1$residuals^2)
```

```
## [1] 1083677
```

C

```
x=model.matrix(Apps~.,data=College)[,-2]
y=College$Apps
grid=10^seq(10,-2, length =100)

ridge.mod=glmnet (x,y,alpha=0, lambda=grid)
y.test=y[test]

set.seed(1)
cv.out=cv.glmnet(x[train ,],y[ train],alpha=0)
plot(cv.out)
```



```
bestlam =cv.out$lambda.min
print("this is our best lambda estimate")
```

```
## [1] "this is our best lambda estimate"
```

```
bestlam
```

```
## [1] 396.1741
```

```
ridge.pred=predict (ridge.mod ,s=bestlam ,newx=x[test ,])
print("this is our test error for our best lambda estimate")
```

```
## [1] "this is our test error for our best lambda estimate"
```

```
mean((ridge.pred -y.test)^2)
```

```
## [1] 912376.9
```

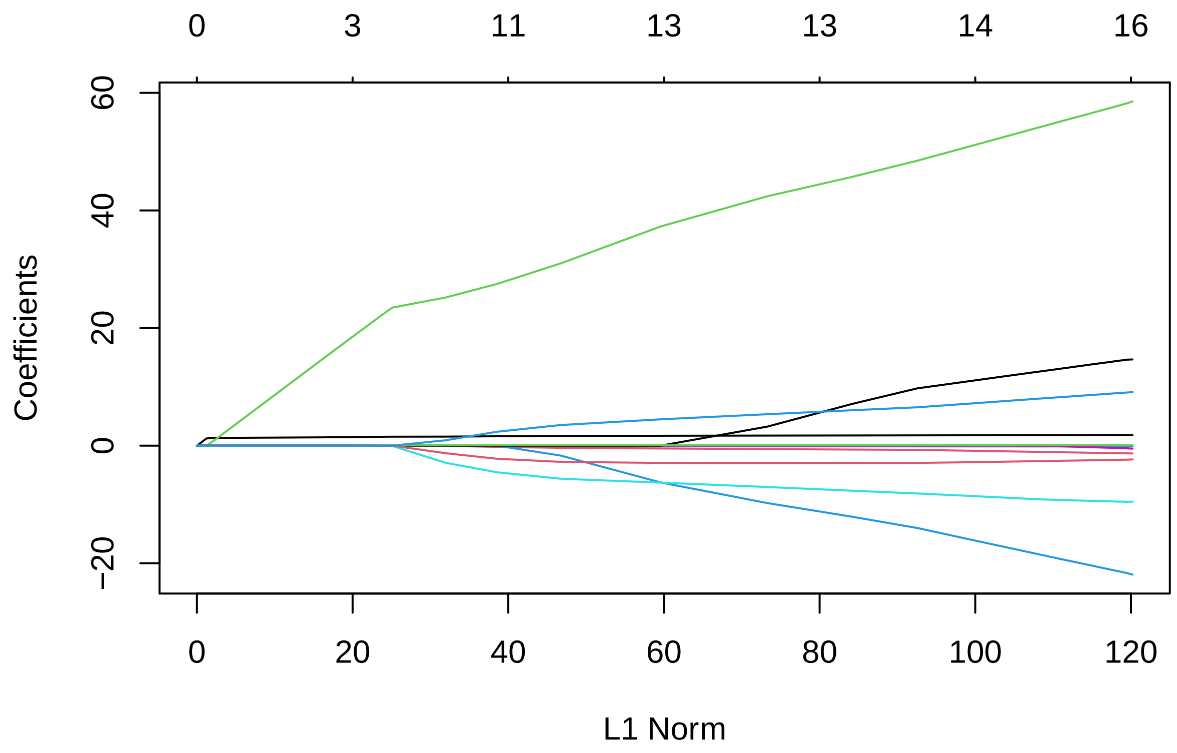
```
out=glmnet(x,y,alpha=0)
predict (out ,type="coefficients",s= bestlam) [1:18,]
```

```
##   (Intercept)   (Intercept)        Accept        Enroll     Top10perc
## -1.980288e+03  0.000000e+00  9.871014e-01  4.873012e-01  2.487770e+01
##     Top25perc   F.Undergrad   P.Undergrad      Outstate    Room.Board
##  9.051487e-01  8.581811e-02  3.320495e-02 -4.129840e-02  1.796283e-01
##         Books      Personal           PhD      Terminal     S.F.Ratio
##  1.065737e-01 -3.837186e-03 -1.973184e+00 -2.810838e+00  2.080287e+01
##   perc.alumni        Expend     Grad.Rate
## -1.043877e+01  7.736496e-02  1.048621e+01
```

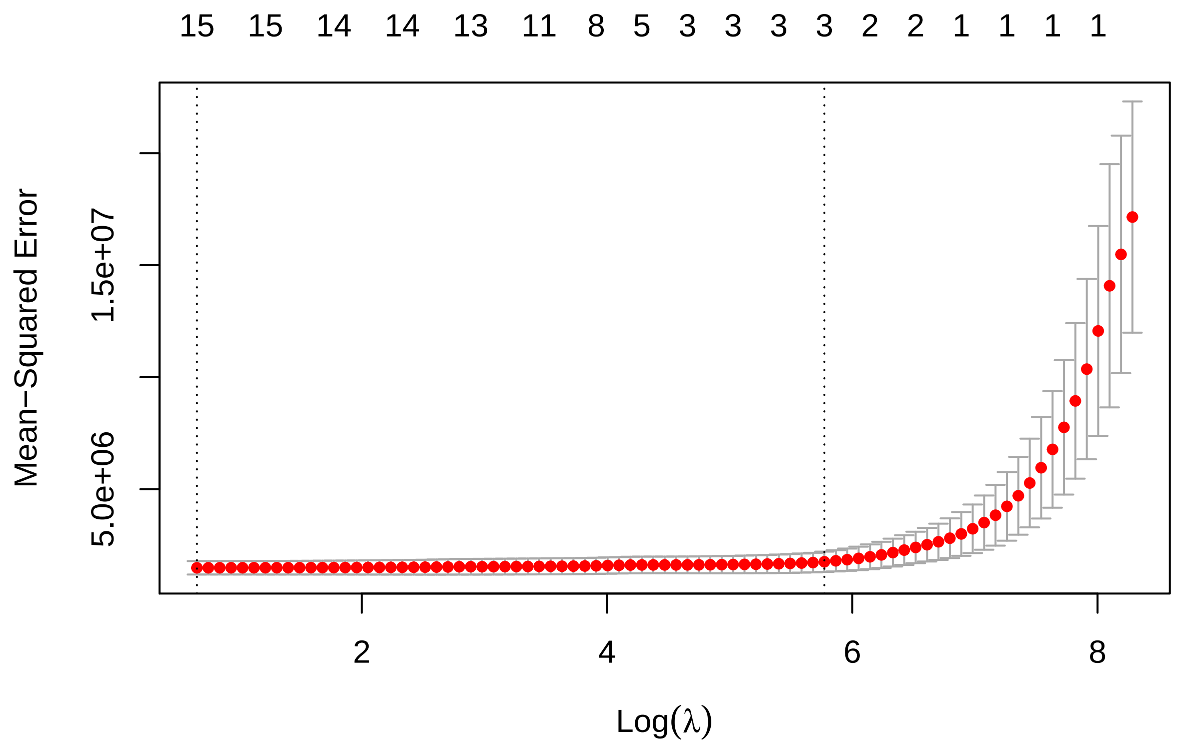
D

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

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



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



```
bestlam =cv.out$lambda.min
print("this is our best lambda estimate")
```

```
## [1] "this is our best lambda estimate"
```

```
bestlam
```

```
## [1] 1.926437
```

```
lasso.pred=predict (lasso.mod ,s=bestlam ,newx=x[test ,])
print("this is our test error for our best lambda estimate")
```

```
## [1] "this is our test error for our best lambda estimate"
```

```
mean((lasso.pred -y.test)^2)
```

```
## [1] 1179560
```

```
out=glmnet (x,y,alpha=1, lambda=grid)
lasso.coef=predict (out ,type="coefficients",s= bestlam) [1:18,]
lasso.coef
```

```
##   (Intercept)   (Intercept)        Accept        Enroll     Top10perc
## -920.20157147    0.00000000    1.57691744   -0.78052922   48.02928703
##     Top25perc   F.Undergrad   P.Undergrad      Outstate    Room.Board
##  -13.12051804    0.05881664    0.04948528   -0.10676680    0.13356042
##         Books      Personal           PhD      Terminal     S.F.Ratio
##    0.00000000    0.02887153   -6.50386286   -1.35898599   22.17736545
##   perc.alumni        Expend     Grad.Rate
##   -1.34400491    0.08119706    7.70337350
```

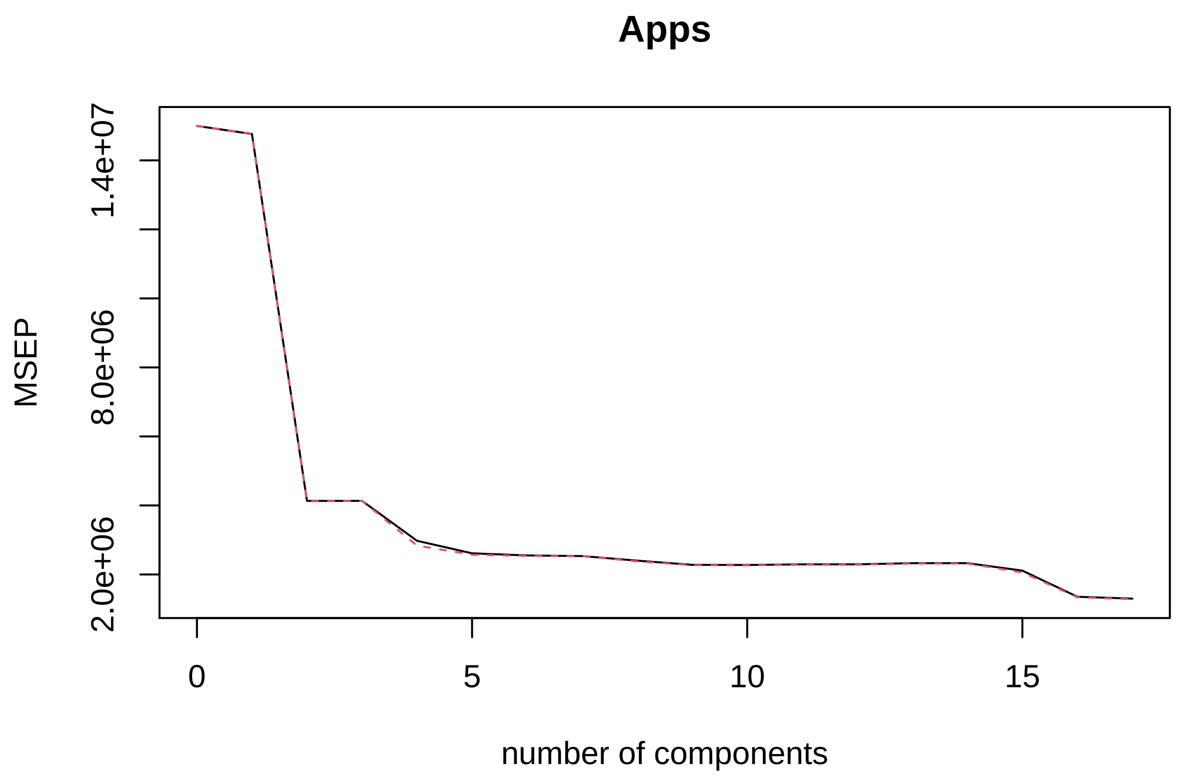
It seems like there are 16 non zero coefficients in our model.

E

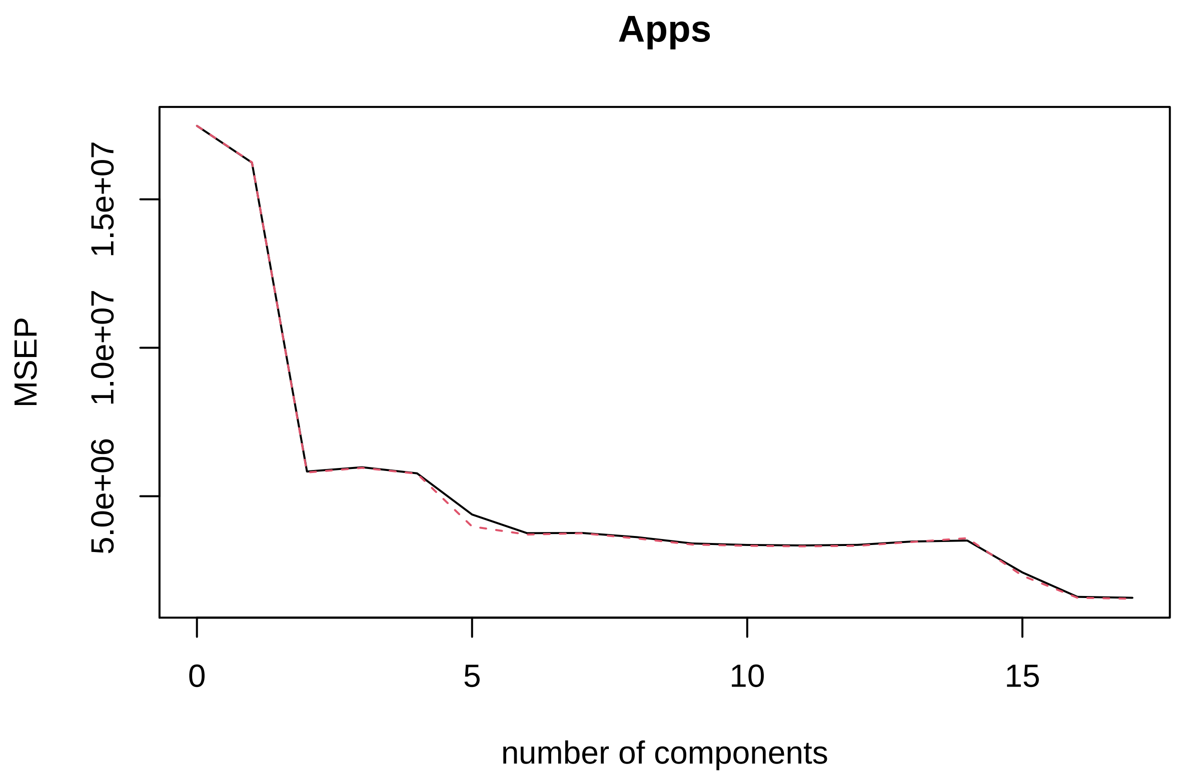
```r
set.seed(1)
pcr.fit=pcr(Apps~., data=College , scale=TRUE ,validation ="CV")

summary (pcr.fit)
```

```
## Data:    X dimension: 777 17
##  Y dimension: 777 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            3873     3842     2033     2033     1725     1617     1597
## adjCV         3873     3844     2031     2033     1684     1604     1593
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1592     1549     1510      1508      1514      1515      1525
## adjCV     1592     1543     1507      1505      1511      1511      1522
##        14 comps  15 comps  16 comps  17 comps
## CV         1526      1453      1163      1140
## adjCV      1522      1435      1157      1134
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X      31.670    57.30    64.30    69.90    75.39    80.38    83.99    87.40
## Apps    2.316    73.06    73.07    82.08    84.08    84.11    84.32    85.18
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       90.50     92.91     95.01     96.81      97.9     98.75     99.36
## Apps    85.88     86.06     86.06     86.10      86.1     86.13     90.32
##       16 comps  17 comps
## X        99.84    100.00
## Apps     92.52     92.92
```

```r
validationplot(pcr.fit ,val.type= "MSEP")
```



```
pcr.fit=pcr(Apps~., data=College , subset=train ,scale=TRUE ,
            validation ="CV")
validationplot(pcr.fit ,val.type="MSEP")
```



```r
pcr.pred=predict (pcr.fit ,x[test ,],ncomp =17)
print("below is our test error")
```

```
## [1] "below is our test error"
```

```r
mean((pcr.pred -y.test)^2)
```

```
## [1] 1192896
```

```r
pcr.fit=pcr(Apps~., data=College ,scale=TRUE ,
            ncomp =17)

summary (pcr.fit)
```

```
## Data:    X dimension: 777 17
##  Y dimension: 777 1
## Fit method: svdpc
## Number of components considered: 17
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X      31.670    57.30    64.30    69.90    75.39    80.38    83.99    87.40
## Apps    2.316    73.06    73.07    82.08    84.08    84.11    84.32    85.18
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       90.50     92.91     95.01     96.81      97.9     98.75     99.36
## Apps    85.88     86.06     86.06     86.10      86.1     86.13     90.32
##       16 comps  17 comps
## X        99.84    100.00
## Apps     92.52     92.92
```

The value of M we obtained using cross validation on our PCR was 17.

F

```
set.seed(1)
pls.fit=plsr(Apps~., data=College , subset=train , scale=TRUE ,
               validation ="CV")
summary (pls.fit)
```

```
## Data:    X dimension: 373 17
##  Y dimension: 373 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            4180     2211     1943     1740     1644     1392     1242
## adjCV         4180     2204     1939     1728     1607     1367     1231
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1227     1212     1204      1202      1203      1201      1200
## adjCV     1217     1203     1194      1193      1193      1191      1190
##        14 comps  15 comps  16 comps  17 comps
## CV         1201      1200      1200      1200
## adjCV      1192      1191      1190      1190
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       26.24    44.87    61.59    63.92    67.96    72.02    75.71    79.81
## Apps    74.40    81.92    86.72    91.59    93.05    93.42    93.51    93.59
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       82.82     86.26     89.25     90.82     92.42     94.71     96.96
## Apps    93.68     93.71     93.73     93.75     93.76     93.76     93.77
##       16 comps  17 comps
## X        98.98    100.00
## Apps     93.77     93.77
```

```
pls.pred=predict (pls.fit ,x[test ,],ncomp =8)
print("below is our test error")
```

```
## [1] "below is our test error"
```

```
mean((pls.pred -y.test)^2)
```

```
## [1] 1209847
```

```
pls.fit=plsr(Apps~., data=College , scale=TRUE , ncomp=8)
summary (pls.fit)
```

```
## Data:    X dimension: 777 17
##  Y dimension: 777 1
## Fit method: kernelpls
## Number of components considered: 8
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       25.76    40.33    62.59    64.97    66.87    71.33    75.39    79.37
## Apps    78.01    85.14    87.67    90.73    92.63    92.72    92.77    92.82
```

The value of M we obtained using cross validation on our pls model was 8.

G

Our results seemed good, as our data was clean and we were able to clearly predict the outcome we were interested in. Looking at the proportion of our variance explained in our outcome across all our models in our output above, we had a R squared of roughly 90%, meaning that we can very accurately predict the number of college applications received. Looking at our test errors, it seems like our test error values were relatively spread out from each other, as our error estimate for the least squares estimate was superior to all of our other models. Additionally, we can see that our errors for the PLS and PCR were significantly better than our errors for the lasso and ridge regression methods, with the ridge regression error being particularly bad.

2.

A

```r
set.seed(1)
train=sample(c(TRUE ,FALSE), nrow(Boston),rep=TRUE)
test=(!train)
Boston$chas<-as.factor(Boston$chas)
```

B

```r
p2_m1<-lm(crim~.,data=Boston, subset = train)
summary(p2_m1)
```

```
##
## Call:
## lm(formula = crim ~ ., data = Boston, subset = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -11.299  -1.860  -0.353   0.963  55.942
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.420104   9.051325   2.035  0.04291 *
## zn           0.047900   0.027055   1.770  0.07787 .
## indus       -0.108191   0.103959  -1.041  0.29902
## chas1       -0.209718   1.507043  -0.139  0.88944
## nox         -7.656603   6.521943  -1.174  0.24153
## rm          -0.322922   0.768164  -0.420  0.67457
## age          0.013498   0.022818   0.592  0.55469
## dis         -0.806539   0.362495  -2.225  0.02698 *
## rad          0.504137   0.109698   4.596 6.87e-06 ***
## tax         -0.001217   0.006451  -0.189  0.85059
## ptratio     -0.191409   0.227237  -0.842  0.40041
## black       -0.014725   0.004521  -3.257  0.00128 **
## lstat        0.109361   0.087650   1.248  0.21331
## medv        -0.122624   0.071805  -1.708  0.08894 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
##
## Residual standard error: 5.798 on 248 degrees of freedom
## Multiple R-squared:  0.5071, Adjusted R-squared:  0.4813
## F-statistic: 19.63 on 13 and 248 DF,  p-value: < 2.2e-16
```

```
mean(p2_m1$residuals^2)
```

```
## [1] 31.81789
```

C

```
regfit.best=regsubsets (crim~.,data=Boston[train ,],
                        nvmax=13)
test.mat=model.matrix(crim~.,data=Boston [test ,])

val.errors =rep(NA ,13)
for(i in 1:13){
  coefi=coef(regfit.best ,id=i)
  pred=test.mat[,names(coefi)]%*%coefi
  val.errors[i]=mean((Boston$crim[test]-pred)^2)
}

which.min(val.errors)
```

```
## [1] 9
```

```
coef(regfit.best ,9)
```

```
## (Intercept)          zn       indus         nox         dis         rad
## 16.49784501  0.04428501 -0.11356470 -6.80041892 -0.87067024  0.48133294
##     ptratio       black       lstat        medv
## -0.17759119 -0.01438142  0.12943566 -0.13215744
```

```
regfit.best=regsubsets (crim~.,data=Boston[test,] ,nvmax=13)
coef(regfit.best ,9)
```

```
##   (Intercept)            zn           nox            rm           dis
##  16.369890929   0.050195174 -14.693411614   1.233082940  -1.150319989
##           rad           tax       ptratio         lstat          medv
##   0.688365438  -0.007773229  -0.374553169   0.151115226  -0.304007811
```

```
p2_m2<-lm(crim~zn+nox+rm+dis+rad+tax+ptratio+lstat+medv, data =Boston[test,])
print("below is test MSE")
```

```
## [1] "below is test MSE"
```

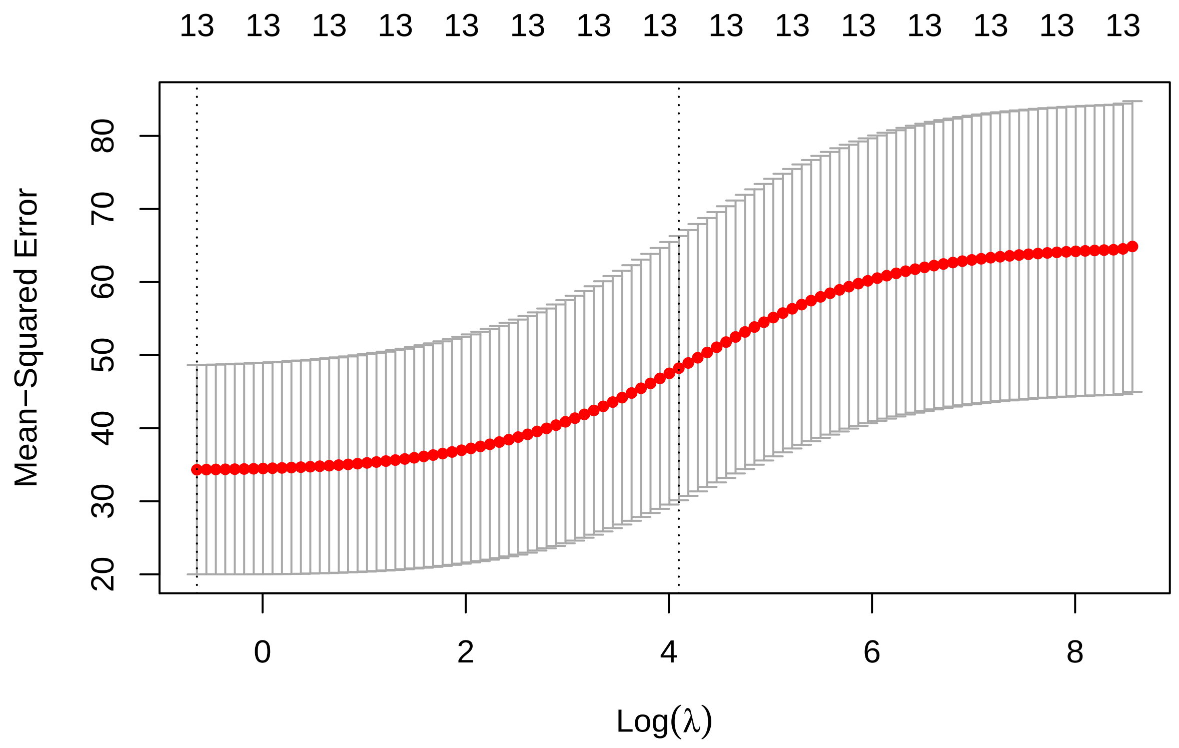
```
mean(p2_m2$residuals^2)
```

```
## [1] 48.27408
```

D

```
x=model.matrix(crim~.,Boston )[,-1]
y=Boston$crim
y.test<-y[test]

set.seed(1)
```

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



```
bestlam =cv.out$lambda.min
print("best value of lambda")
```

```
## [1] "best value of lambda"
```

```
bestlam
```

```
## [1] 0.5240686
```

```
grid=10^seq(10,-2, length =100)

ridge.mod=glmnet (x,y,alpha=0, lambda=grid)

ridge.pred=predict (ridge.mod ,s=bestlam ,newx=x[test ,])

out=glmnet(x,y,alpha=0)

predict (out ,type="coefficients",s= bestlam) [1:14,]
```

```
##  (Intercept)           zn        indus        chas1          nox           rm
##  9.063048626  0.033002416 -0.082046152 -0.737684583 -5.393098481  0.335972073
##          age          dis          rad          tax      ptratio        black
##  0.001962473 -0.702123641  0.422779054  0.003400607 -0.135911587 -0.008483285
##        lstat         medv
##  0.142613436 -0.139604127
```

```
print("test error below")
```

```
## [1] "test error below"
```

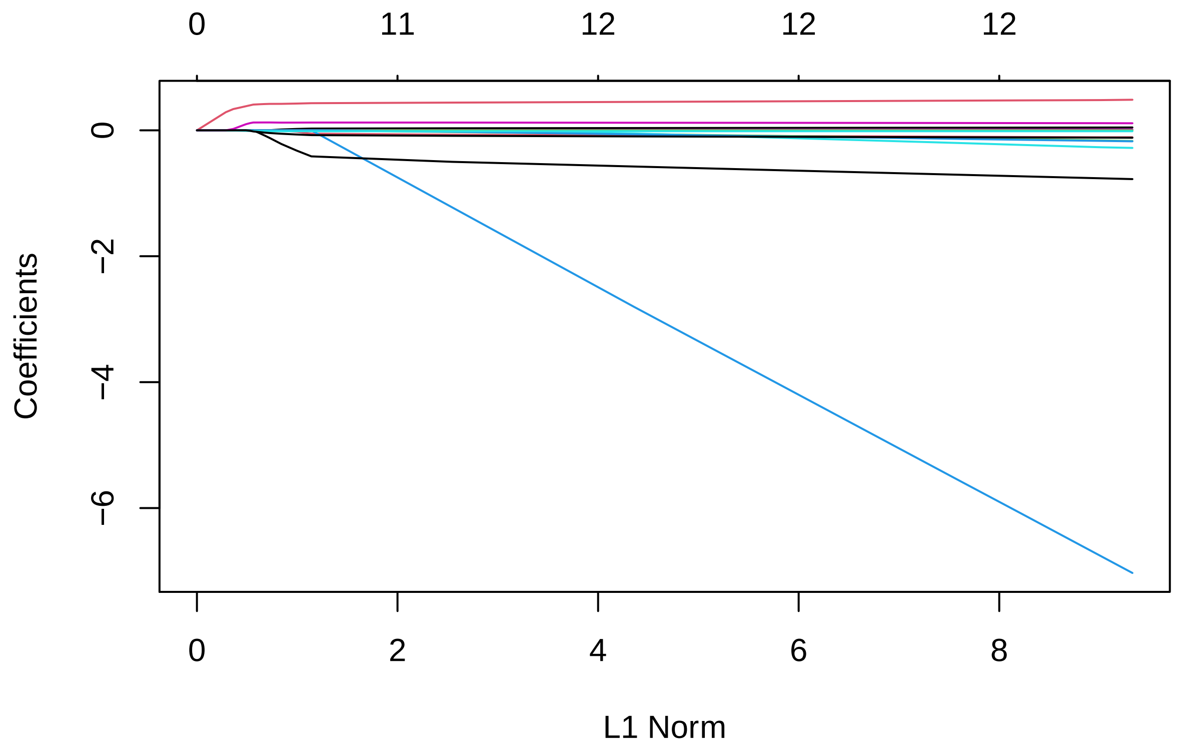
```
mean((ridge.pred-y.test)^2)
```

```
## [1] 49.69485
```

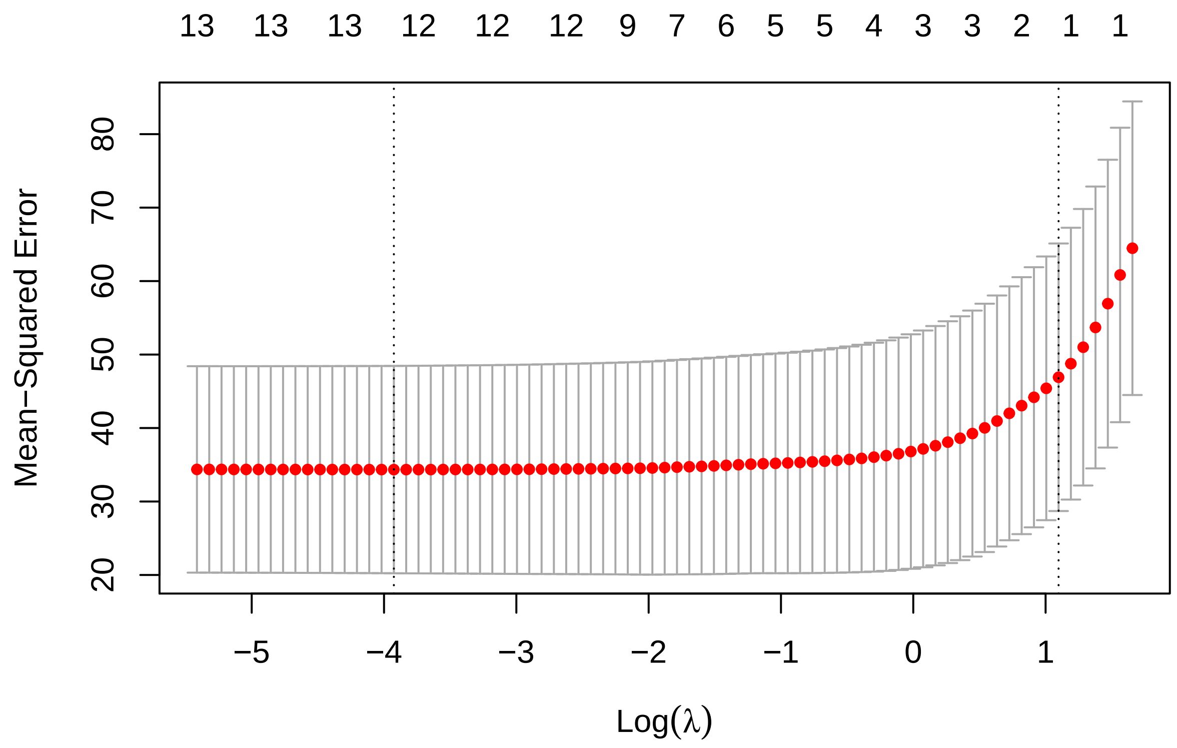
E

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

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



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



```
bestlam =cv.out$lambda.min
print("best value of lambda below")
```

```
## [1] "best value of lambda below"
```

```
bestlam
```

```
## [1] 0.01973085
```

```
lasso.pred=predict (lasso.mod ,s=bestlam ,newx=x[test ,])
print("test error below")
```

```
## [1] "test error below"
```

```
mean((lasso.pred -y.test)^2)
```

```
## [1] 50.73601
```

```
out=glmnet (x,y,alpha=1, lambda=grid)
lasso.coef=predict (out ,type="coefficients",s= bestlam) [1:14,]
lasso.coef
```

```
##  (Intercept)           zn        indus        chas1          nox           rm
## 15.274657338  0.041086972 -0.068744852 -0.674509624 -9.009346033  0.361194922
##          age          dis          rad          tax      ptratio        black
##  0.000000000 -0.914511964  0.554349050 -0.001960195 -0.239265646 -0.007532969
##        lstat         medv
##  0.127257803 -0.182791130
```

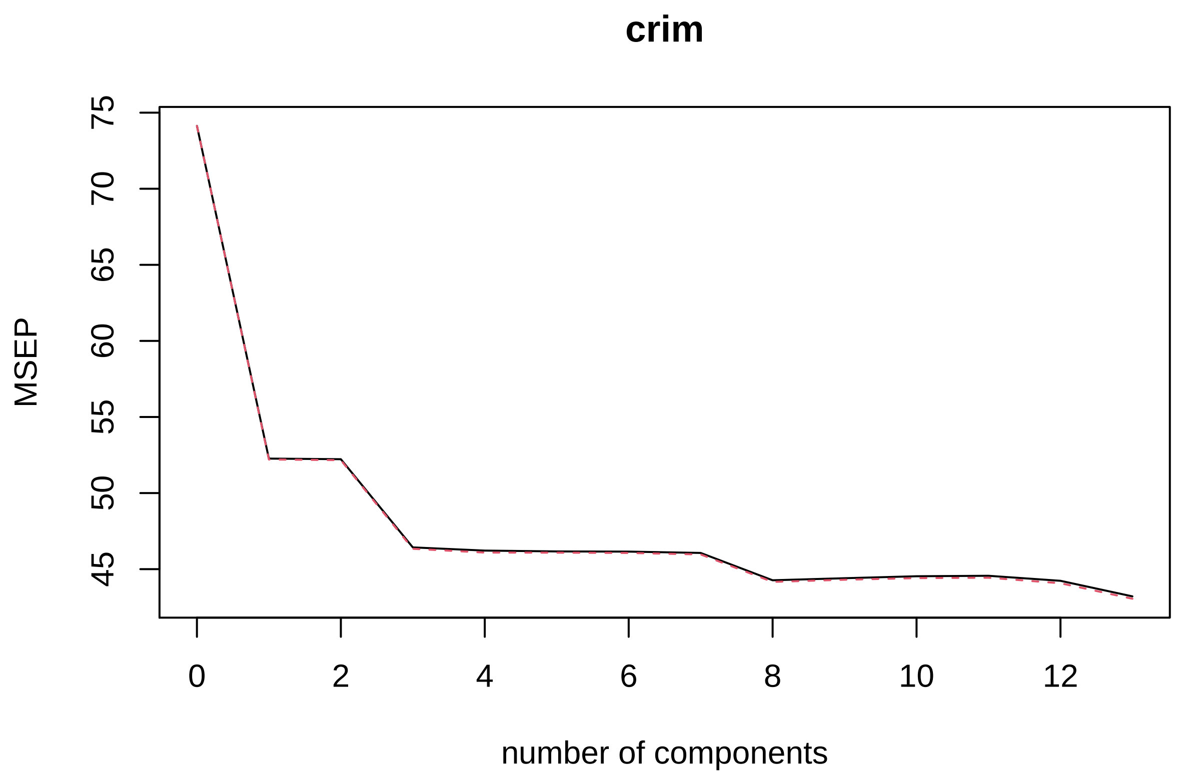
The age parameter has been estimated to be zero.

F

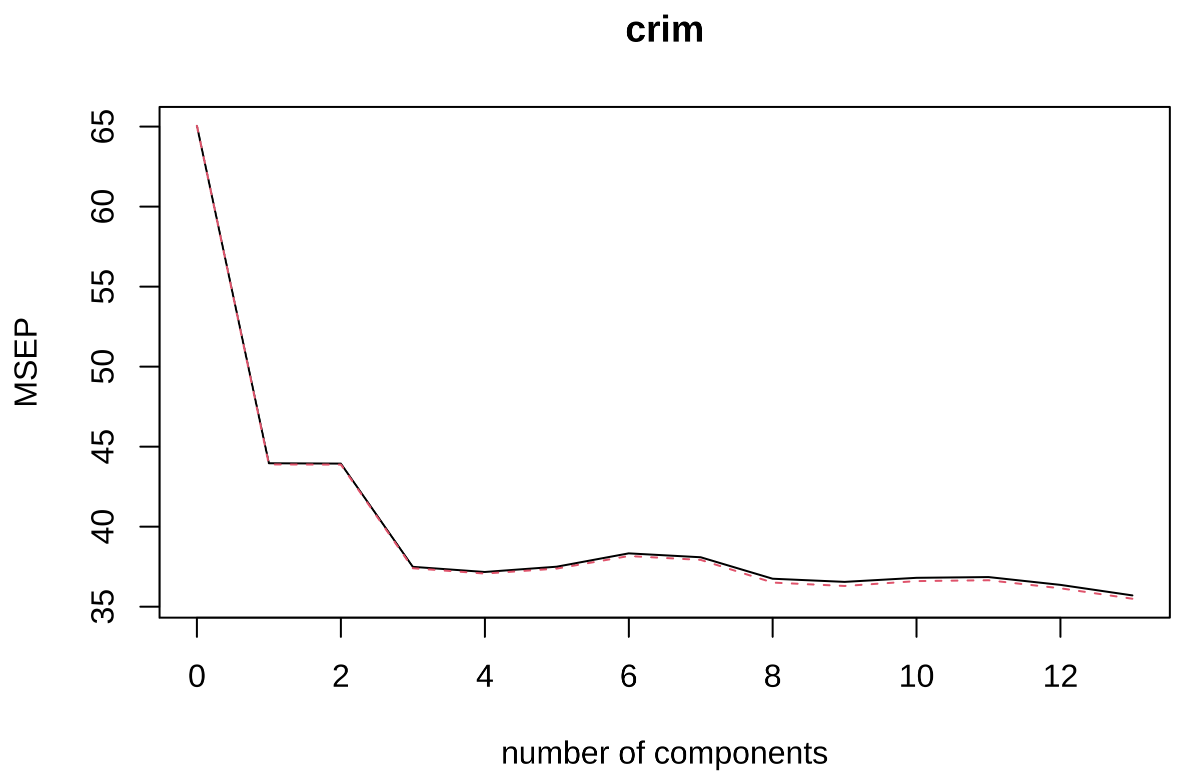
```r
set.seed(2)
pcr.fit=pcr(crim~., data=Boston , scale=TRUE ,validation ="CV")

summary(pcr.fit)
```

```
## Data:    X dimension: 506 13
##  Y dimension: 506 1
## Fit method: svdpc
## Number of components considered: 13
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            8.61    7.229    7.227    6.814    6.799    6.795    6.794
## adjCV         8.61    7.225    7.222    6.807    6.789    6.788    6.787
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       6.787    6.654    6.664     6.673     6.676     6.651     6.573
## adjCV    6.780    6.645    6.656     6.664     6.666     6.639     6.562
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       47.70    60.36    69.67    76.45    82.99    88.00    91.14    93.45
## crim    30.69    30.87    39.27    39.61    39.61    39.86    40.14    42.47
##       9 comps  10 comps  11 comps  12 comps  13 comps
## X       95.40     97.04     98.46     99.52     100.0
## crim    42.55     42.78     43.04     44.13      45.4
```

```r
validationplot(pcr.fit ,val.type= "MSEP")
```



```r
pcr.fit=pcr(crim~., data=Boston , subset=train ,scale=TRUE ,
            validation ="CV")
validationplot(pcr.fit ,val.type="MSEP")
```



```
pcr.pred=predict (pcr.fit ,x[test,],ncomp =13)
mean((pcr.pred -y.test)^2)
```

```
## [1] 50.65678
```

```
pcr.fit=pcr(y~x,scale=TRUE ,ncomp=13)
summary (pcr.fit)
```

```
## Data:    X dimension: 506 13
##  Y dimension: 506 1
## Fit method: svdpc
## Number of components considered: 13
## TRAINING: % variance explained
##    1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X    47.70    60.36    69.67    76.45    82.99    88.00    91.14    93.45
## y    30.69    30.87    39.27    39.61    39.61    39.86    40.14    42.47
##    9 comps  10 comps  11 comps  12 comps  13 comps
## X    95.40     97.04     98.46     99.52     100.0
## y    42.55     42.78     43.04     44.13      45.4
```

```
print("M is 13")
```

```
## [1] "M is 13"
```

```
print("test error below")
```

```
## [1] "test error below"
```

```r
mean((pcr.pred -y.test)^2)
```

```
## [1] 50.65678
```

G

```r
set.seed(1)
pls.fit=plsr(crim~., data=Boston , subset=train , scale=TRUE ,
             validation ="CV")
summary (pls.fit)
```

```
## Data:    X dimension: 262 13
##  Y dimension: 262 1
## Fit method: kernelpls
## Number of components considered: 13
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           8.065    6.357    6.034    6.079    6.062    6.016    5.995
## adjCV        8.065    6.353    6.025    6.057    6.041    5.998    5.977
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       6.025    6.006    5.988     5.998     5.999     6.000     6.000
## adjCV    6.004    5.987    5.970     5.979     5.981     5.981     5.981
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       43.95    54.85    60.29    69.59    76.82    79.48    83.83    87.61
## crim    39.21    47.44    49.46    50.05    50.31    50.56    50.64    50.68
##       9 comps  10 comps  11 comps  12 comps  13 comps
## X       90.24     94.42     96.55     98.20    100.00
## crim    50.70     50.71     50.71     50.71     50.71
```

```r
pls.pred=predict (pls.fit ,x[test ,],ncomp =10)

pls.fit=plsr(crim~., data=Boston , scale=TRUE , ncomp=10)
summary (pls.fit)
```

```
## Data:    X dimension: 506 13
##  Y dimension: 506 1
## Fit method: kernelpls
## Number of components considered: 10
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       47.27    56.79    61.38    71.13    76.41    79.78    83.99    86.27
## crim    34.32    41.81    44.03    44.58    44.94    45.24    45.33    45.38
##       9 comps  10 comps
## X        88.5     91.32
## crim     45.4     45.40
```

```r
print("M is 10")
```

```
## [1] "M is 10"
```

```r
print("test error is below")
```

```
## [1] "test error is below"
```

```r
mean((pls.pred -y.test)^2)
```

```
## [1] 50.63096
```

H

It seems like the test errors for all of our methods are relatively close to each other, thus it seems like any of the above methods would be acceptable to use to predict our outcome. It seems like we can predict per capita crime rate relatively accurately, given that we can account for roughly half of the variance related to per capita crime rate. The variable that seems to be the most important would be Nox, given it's large coefficient. If I were to analyze the data again, I would perhaps use a different training and validation set, to see if there is significant variation in outcomes with a slightly different validation set.

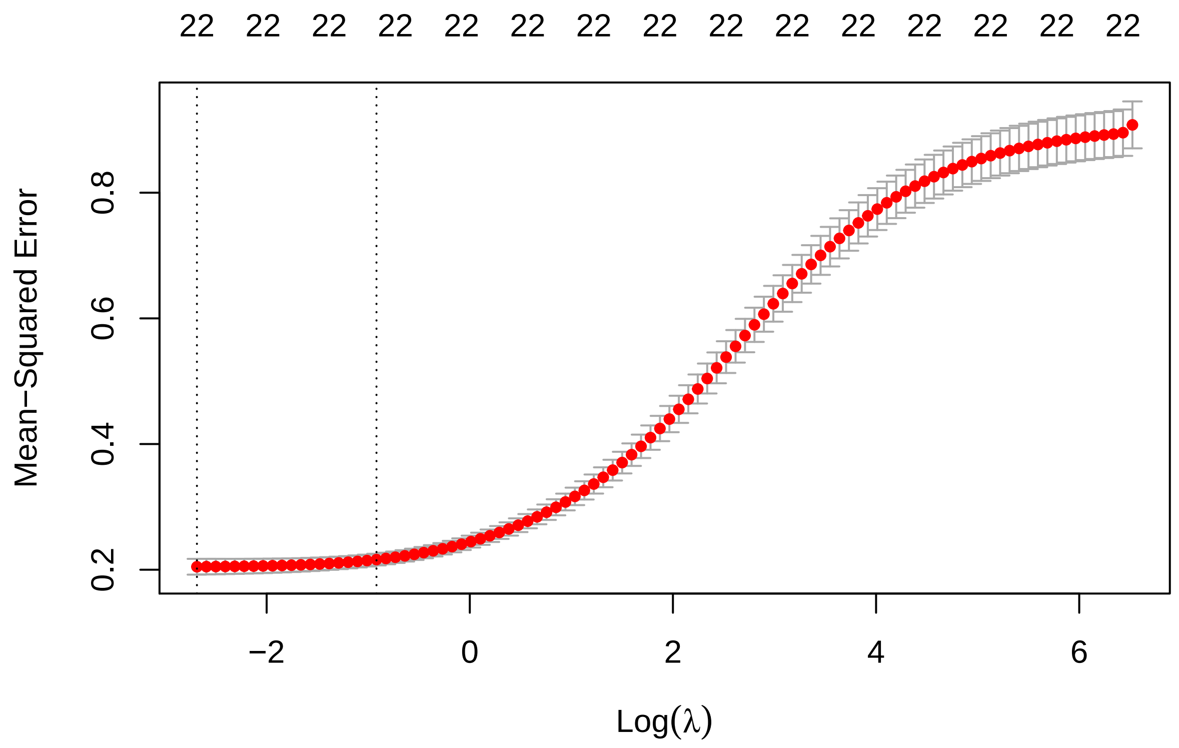
3.

```r
train=sample(c(TRUE ,FALSE), nrow(lakes),rep=TRUE)
test=(!train)
```

Ridge Regression

Below is our code and output for the Ridge Regression

```r
x=model.matrix(lsecchi~.-secchi,lakes )[,-c(4,25)]
y=lakes$lsecchi
y.test<-y[test]
set.seed(1)
cv.out=cv.glmnet(x[train,],y[train],alpha=0)
plot(cv.out)
```



```
bestlam =cv.out$lambda.min
bestlam
```

```
## [1] 0.06812536
```

```
grid=10^seq(10,-2, length =100)
ridge.mod=glmnet (x,y,alpha=0, lambda=grid)
ridge.pred=predict (ridge.mod ,s=bestlam ,newx=x[test ,])
out=glmnet(x,y,alpha=0)
predict (out ,type="coefficients",s= bestlam) [1:24,]
```

```
##        (Intercept)        (Intercept)                 tp                 tn
##      -7.602657e-01       0.000000e+00      -4.554496e-03      -4.748844e-05
##                lat               long          lake_area         mean_depth
##       6.974510e-02       6.692322e-03      -4.246087e-05       3.698620e-02
##          max_depth          iws_urban             iws_ag        iws_pasture
##       1.105345e-02      -4.511235e-02      -2.081887e-01       2.101931e-01
##         iws_forest        iws_wetland   mean_annual_temp   mean_winter_temp
##       2.661107e-01      -5.385826e-01      -1.170359e-02       7.140768e-03
##   mean_spring_temp   mean_summer_temp     mean_fall_temp mean_annual_precip
##      -1.868178e-02      -2.393641e-02      -2.761409e-02       1.461565e-03
## mean_winter_precip mean_spring_precip mean_summer_precip   mean_fall_precip
##       2.654544e-03       3.938596e-03      -6.899772e-03       2.599957e-04
```

```
mean((ridge.pred-y.test)^2)
```

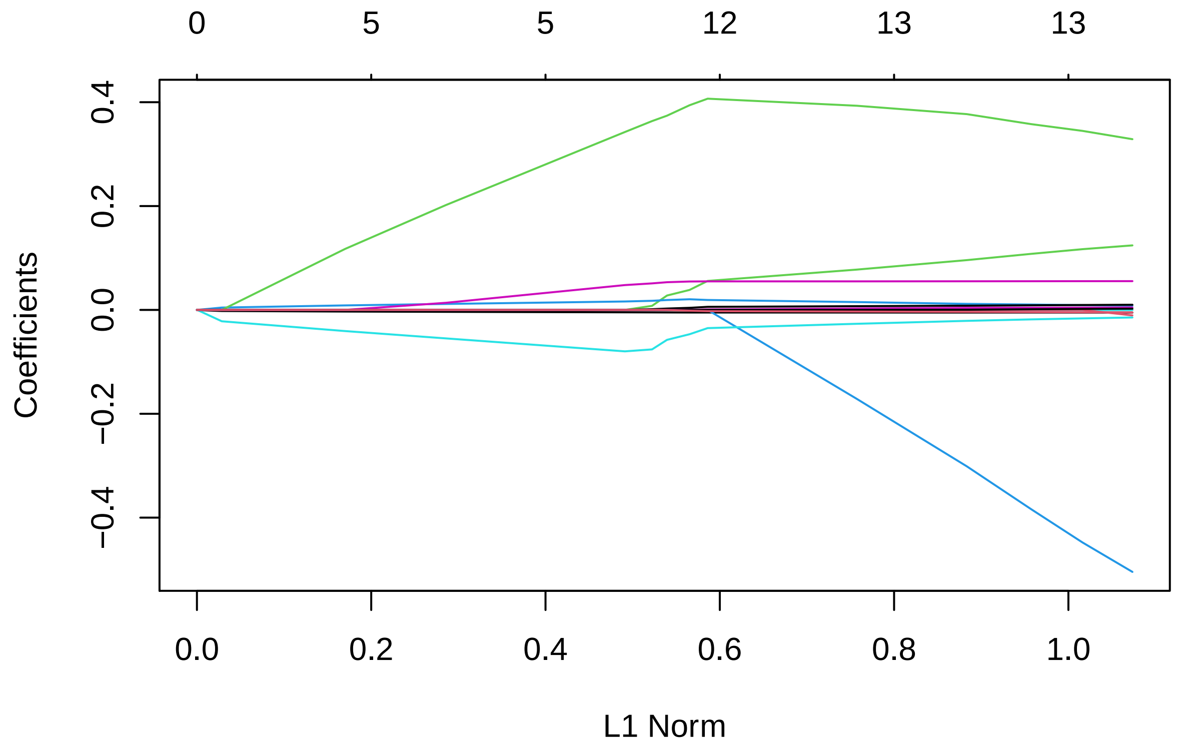
```
## [1] 0.1809818
```

Lasso

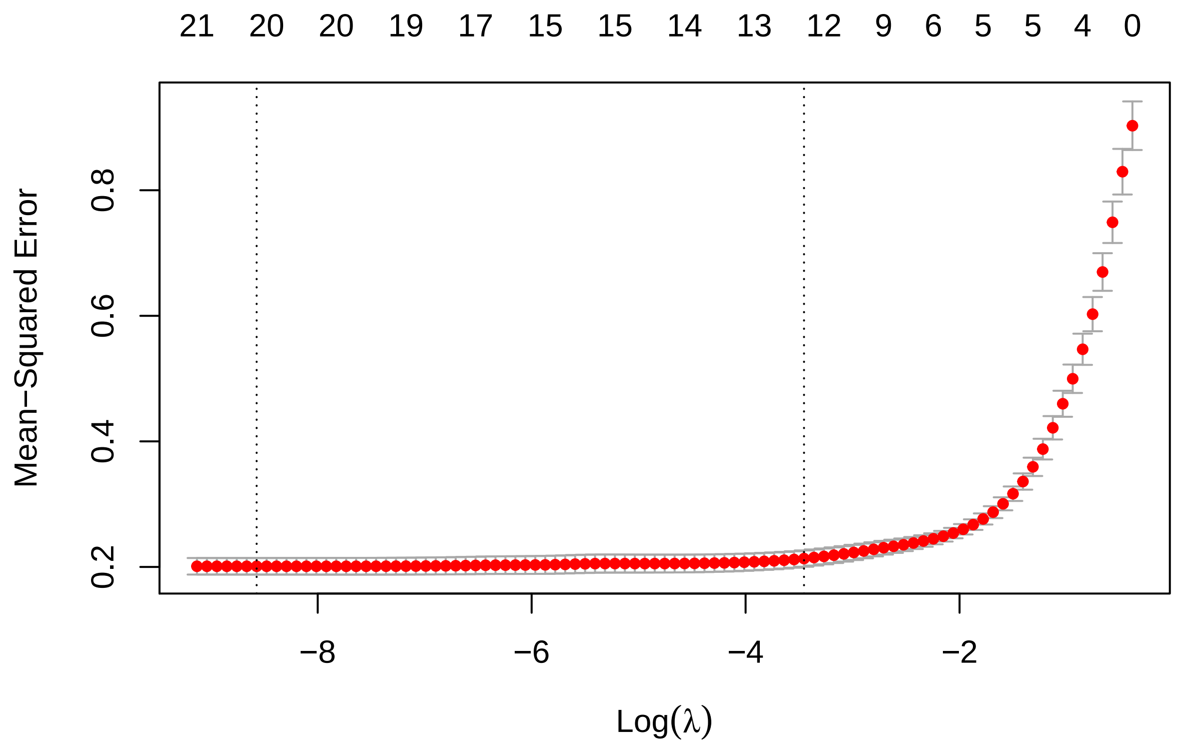
Below is our code and output for the Lasso

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

```
## Warning in regularize.values(x, y, ties, missing(ties), na.rm = na.rm):
## collapsing to unique 'x' values
```



```r
set.seed(1)
cv.out=cv.glmnet(x[train ,],y[ train],alpha=1)
plot(cv.out)
```



```r
bestlam =cv.out$lambda.min
bestlam
```

```
## [1] 0.0001895629
```

```r
lasso.pred=predict (lasso.mod ,s=bestlam ,newx=x[test ,])
mean((lasso.pred -y.test)^2)
```

```
## [1] 0.2203316
```

```r
out=glmnet (x,y,alpha=1, lambda=grid)
lasso.coef=predict (out ,type="coefficients",s= bestlam) [1:24,]
lasso.coef
```

```
##        (Intercept)        (Intercept)                 tp                 tn
##      -3.462498e+00       0.000000e+00      -4.923366e-03      -4.872727e-05
##                lat               long          lake_area         mean_depth
##       1.151524e-01       6.011241e-03      -3.267351e-05       3.975757e-02
##          max_depth          iws_urban             iws_ag        iws_pasture
##       9.374553e-03       0.000000e+00      -3.682796e-02       4.924675e-02
##         iws_forest        iws_wetland   mean_annual_temp   mean_winter_temp
##       4.065587e-01      -3.235198e-01       0.000000e+00       0.000000e+00
##   mean_spring_temp   mean_summer_temp     mean_fall_temp mean_annual_precip
##       0.000000e+00      -2.923733e-02       0.000000e+00       0.000000e+00
## mean_winter_precip mean_spring_precip mean_summer_precip   mean_fall_precip
##       3.488196e-03       4.194002e-03      -4.727267e-03       0.000000e+00
```

```r
cv.out2=cv.glmnet(x,y,alpha=1,nfolds=5)
cv.out2
```

```
##
## Call:  cv.glmnet(x = x, y = y, nfolds = 5, alpha = 1)
##
## Measure: Mean-Squared Error
##
##       Lambda Measure       SE Nonzero
## min 0.000242  0.1990 0.009456      20
## 1se 0.027770  0.2082 0.008477      13
```

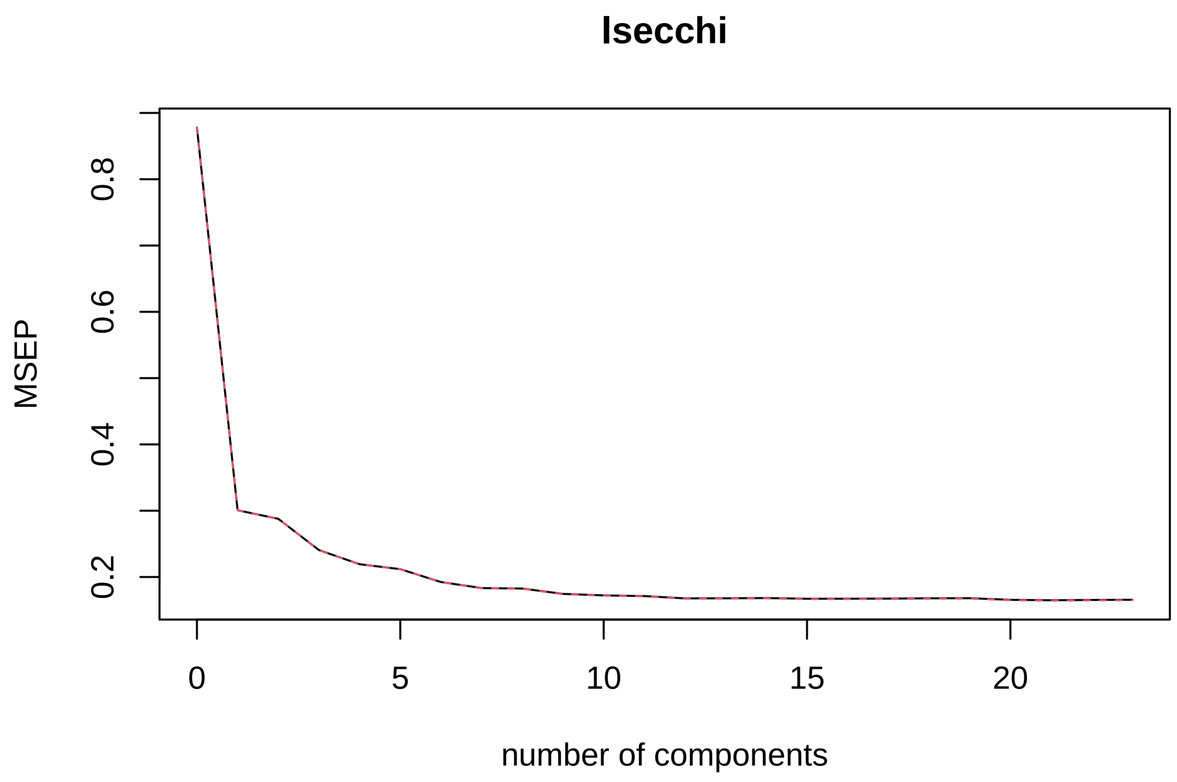
PC regression

Below is our code and output for the Principle Components regression

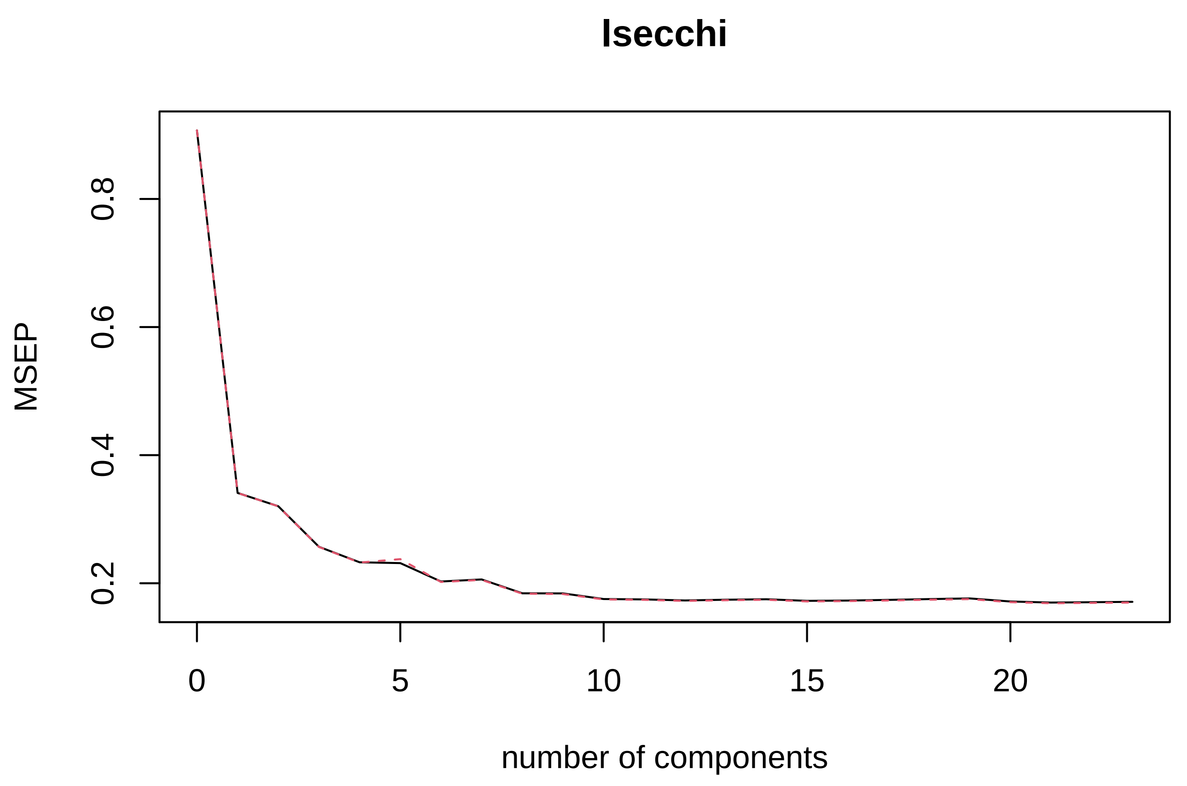
```
set.seed(1)
pcr.fit=pcr(lsecchi~. -secchi, data=lakes , scale=TRUE ,validation ="CV")
summary (pcr.fit)
```

```
## Data:    X dimension: 1188 23
##  Y dimension: 1188 1
## Fit method: svdpc
## Number of components considered: 23
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.9371   0.5483   0.5365   0.4905   0.4682   0.4602   0.4386
## adjCV       0.9371   0.5483   0.5365   0.4903   0.4680   0.4604   0.4383
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.4281   0.4273   0.4177    0.4151    0.4138    0.4095    0.4097
## adjCV   0.4278   0.4270   0.4172    0.4146    0.4133    0.4090    0.4091
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## CV       0.4102    0.4089    0.4090    0.4091    0.4096    0.4099    0.4069
## adjCV    0.4096    0.4083    0.4083    0.4085    0.4090    0.4092    0.4063
##        21 comps  22 comps  23 comps
## CV       0.4061    0.4067    0.4070
## adjCV    0.4054    0.4059    0.4062
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          38.16    58.84    68.03    74.50    79.22    83.79    87.29    90.28
## lsecchi    65.79    67.36    72.90    75.34    76.33    78.51    79.69    79.98
##          9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X          93.03     95.49     96.94     98.01     98.82     99.25     99.55
## lsecchi    80.91     81.05     81.25     81.78     81.79     81.79     81.96
##          16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
## X           99.70     99.81     99.91     99.97     99.99    100.00    100.00
## lsecchi     81.97     81.98     81.98     81.98     82.24     82.38     82.38
##          23 comps
## X          100.00
## lsecchi     82.39
```

```
validationplot(pcr.fit ,val.type= "MSEP")
```



```
pcr.fit=pcr(lsecchi~. -secchi, data=lakes , subset=train ,scale=TRUE ,
          validation ="CV")
validationplot(pcr.fit ,val.type="MSEP")
```



```r
pcr.pred=predict (pcr.fit ,x[test ,],ncomp =12)
mean((pcr.pred -y.test)^2)
```

```
## [1] 188.7026
```

```r
pcr.fit=pcr(lsecchi~. -secchi, data=lakes , scale=TRUE, ncomp=12)
summary (pcr.fit)
```

```
## Data:    X dimension: 1188 23
##  Y dimension: 1188 1
## Fit method: svdpc
## Number of components considered: 12
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          38.16    58.84    68.03    74.50    79.22    83.79    87.29    90.28
## lsecchi    65.79    67.36    72.90    75.34    76.33    78.51    79.69    79.98
##          9 comps  10 comps  11 comps  12 comps
## X          93.03     95.49     96.94     98.01
## lsecchi    80.91     81.05     81.25     81.78
```

PLS

Below is our code and output for the partial least squares regression.

```r
set.seed(1)
pls.fit=plsr(lsecchi~.-secchi, data=lakes , subset=train , scale=TRUE ,
             validation ="CV")
summary (pls.fit)
```

```
## Data:    X dimension: 594 23
##  Y dimension: 594 1
## Fit method: kernelpls
## Number of components considered: 23
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.9524   0.5223   0.4408   0.4225   0.4142   0.4130   0.4127
## adjCV       0.9524   0.5224   0.4404   0.4219   0.4136   0.4123   0.4120
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.4136   0.4137   0.4142    0.4138    0.4127    0.4134    0.4125
## adjCV   0.4127   0.4128   0.4131    0.4128    0.4119    0.4116    0.4107
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps  20 comps
## CV       0.4149    0.4163    0.4137    0.4111    0.4110    0.4107    0.4112
## adjCV    0.4130    0.4143    0.4121    0.4098    0.4097    0.4095    0.4099
##        21 comps  22 comps  23 comps
## CV       0.4114    0.4111    0.4124
## adjCV    0.4101    0.4099    0.4111
##
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          36.60    48.68    65.38    71.61    76.12    79.95    82.69    84.45
## lsecchi    70.11    79.31    81.27    82.17    82.39    82.50    82.59    82.68
##          9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X          86.29     89.87     92.16     93.59     95.61     97.50     98.98
## lsecchi    82.74     82.76     82.81     83.04     83.20     83.26     83.29
##          16 comps  17 comps  18 comps  19 comps  20 comps  21 comps  22 comps
## X           99.57     99.68     99.71     99.78     99.87     99.97    100.00
## lsecchi     83.30     83.31     83.32     83.32     83.32     83.33     83.33
##          23 comps
## X          100.00
## lsecchi     83.35
```

```
pls.pred=predict (pls.fit ,x[test ,],ncomp =19)
mean((pls.pred -y.test)^2)
```

```
## [1] 190.3498
```

```
pls.fit=plsr(lsecchi~.-secchi, data=lakes , scale=TRUE , ncomp=19)
summary (pls.fit)
```

```
## Data:    X dimension: 1188 23
##  Y dimension: 1188 1
## Fit method: kernelpls
## Number of components considered: 19
## TRAINING: % variance explained
##          1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X          37.83    49.23    65.07    71.95    76.50    80.13    83.09    85.32
## lsecchi    71.38    79.12    80.73    81.54    81.77    81.89    81.95    81.99
##          9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X          87.15     90.44     92.85     94.78     95.74     98.08     99.01
## lsecchi    82.04     82.06     82.08     82.14     82.28     82.29     82.32
##          16 comps  17 comps  18 comps  19 comps
## X            99.30    99.68    99.81    99.85
## lsecchi      82.37    82.37    82.38    82.38
```

Looking at our four methods, I believe that the best model amongst our choices is the lasso regression. I chose between the four methods primarily by considering the differences in test error, as well as considering concerns of interpretability. Looking at test error specifically, while the ridge regression had the lowest test error out of all the models, the lasso regression had comparable error, while being significantly more parsimonius, as the final lasso model comprised 7 less predictors than the full ridge regression model.

We selected the variables for our lasso, pls, and pcr models by using a holdout sample to cross validate our results.

5 fold cross validation on our lasso model confirms that a 13 parameter model (our 12 predictors, plus an intercept) is the best model we can be using.

4.

```
set.seed(1)
x1 = rnorm(1000)
x2 = rnorm(1000)
x3 = rnorm(1000)
x4 = rnorm(1000)
x5 = rnorm(1000)
e = .1*rnorm(1000)
beta.truth = c(1.3,.01,-1.2,-.02,.6)
x = cbind(x1,x2,x3,x4,x5)
y = x%*%beta.truth + e
data4<-cbind(y,x1,x2,x3,x4,x5)
data4<-as.data.frame(data4)

myRSSgen <- function(beta,x,y,lam,q){
  sum((y-x%*% beta)^2) + lam*sum((abs(beta))^q)
}
minigrid<-seq(from=0, to=4, by=.2)
```

A

```
optim_list<-matrix(NA, nrow=length(minigrid), ncol=length(minigrid))
for(i in 1:length(minigrid)){
  for(j in 1:length(minigrid)){
    optim_list[i,j]<-as.numeric(optim(rep(0,ncol(x)),myRSSgen,method='CG',x=x,y=y,lam=minigrid[i],q=min
    }
}
print(optim_list)
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
##  [1,]  9.466934  9.466934  9.466934  9.466934  9.466934  9.466934  9.466934
##  [2,] 10.466934 10.246233 10.148964 10.108226 10.094166 10.093468 10.100091
##  [3,] 11.466934 11.024352 10.830139 10.749143 10.721238 10.719905 10.733153
##  [4,] 12.466934 11.801123 11.510395 11.389678 11.348149 11.346245 11.366118
##  [5,] 13.466934 12.576285 12.189653 12.029820 11.974900 11.972487 11.998987
##  [6,] 14.466934 13.349371 12.867806 12.669557 12.601489 12.598633 12.631761
##  [7,] 15.466934 13.834534 13.544704 13.308878 13.227915 13.224682 13.264438
##  [8,] 16.466934 14.565520 14.220105 13.947768 13.854177 13.850634 13.897020
##  [9,] 17.466934 15.452163 14.893512 14.586211 14.480274 14.476489 14.529506
## [10,] 18.466934 16.203317 15.501606 15.224189 15.106206 15.102247 15.161896
## [11,] 19.466934 16.750675 16.163866 15.861680 15.731972 15.727908 15.794191
## [12,] 20.466934 17.547007 16.796647 16.498658 16.357569 16.353473 16.426390
## [13,] 21.466934 18.367684 17.487189 17.135090 16.982998 16.978940 17.058493
## [14,] 22.466934 19.124606 18.159156 17.770936 17.608257 17.604310 17.690500
## [15,] 23.466934 19.731972 18.760046 18.406139 18.233345 18.229583 18.322412
## [16,] 24.466934 20.585402 19.379127 19.040612 18.858260 18.854760 18.954228
## [17,] 25.466934 21.346039 20.104524 19.674204 19.483002 19.479839 19.585948
## [18,] 26.466934 22.005566 20.692726 20.273223 20.107568 20.104822 20.217573
## [19,] 27.466934 22.722969 21.368858 20.904349 20.731958 20.729707 20.849103
## [20,] 28.466934 23.185855 22.024978 21.523010 21.356169 21.354496 21.480537
## [21,] 29.466934 24.156807 22.632069 22.176887 21.980201 21.979187 22.111875
##            [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
##  [1,]  9.466934  9.466934  9.466934  9.466934  9.466934  9.466934  9.466934
##  [2,] 10.111250 10.125639 10.142636 10.161937 10.183393 10.206933 10.232525
##  [3,] 10.755446 10.784186 10.818129 10.856667 10.899501 10.946484 10.997551
##  [4,] 11.399522 11.442575 11.493414 11.551127 11.615261 11.685590 11.762012
##  [5,] 12.043478 12.100806 12.168491 12.245316 12.330670 12.424250 12.525909
##  [6,] 12.687314 12.758878 12.843360 12.939233 13.045731 13.162464 13.289244
##  [7,] 13.331030 13.416793 13.518020 13.632880 13.760443 13.900234 14.052016
##  [8,] 13.974626 14.074549 14.192473 14.326256 14.474806 14.637559 14.814228
##  [9,] 14.618101 14.732148 14.866719 15.019362 15.188822 15.374441 15.575878
## [10,] 15.261457 15.389588 15.540756 15.712197 15.902489 16.110880 16.336969
## [11,] 15.904693 16.046871 16.214586 16.404763 16.615809 16.846875 17.097501
## [12,] 16.547810 16.703996 16.888208 17.097058 17.328781 17.582429 17.857475
## [13,] 17.190806 17.360963 17.561623 17.789083 18.041406 18.317540 18.616891
## [14,] 17.833682 18.017772 18.234831 18.480839 18.753685 19.052210 19.375751
## [15,] 18.476439 18.674423 18.907831 19.172325 19.465617 19.786438 20.134055
## [16,] 19.119076 19.330917 19.580624 19.863542 20.177203 20.520227 20.891803
## [17,] 19.761593 19.987254 20.253210 20.554490 20.888442 21.253575 21.648997
## [18,] 20.403990 20.643432 20.925589 21.245169 21.599336 21.986484 22.405637
## [19,] 21.046268 21.299454 21.597761 21.935578 22.309885 22.718953 23.161725
## [20,] 21.688426 21.955317 22.269726 22.625719 23.020088 23.450984 23.917260
## [21,] 22.330464 22.611024 22.941485 23.315592 23.729947 24.182577 24.672244
##           [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
##  [1,]  9.466934  9.466934  9.466934  9.466934  9.466934  9.466934  9.466934
##  [2,] 10.260164 10.289858 10.321628 10.355507 10.391533 10.429755 10.470227
##  [3,] 11.052683 11.111896 11.175227 11.242732 11.314483 11.390568 11.471089
##  [4,] 11.844495 11.933053 12.027734 12.128614 12.235791 12.349384 12.469532
##  [5,] 12.635599 12.753329 12.879152 13.013158 13.155462 13.306211 13.465570
##  [6,] 13.425998 13.572726 13.729484 13.896367 14.073505 14.261059 14.459217
##  [7,] 14.215691 14.391247 14.578733 14.778246 14.989924 15.213937 15.450485
##  [8,] 15.004682 15.208893 15.426901 15.658800 15.904727 16.164855 16.439388
##  [9,] 15.792970 16.025666 16.273992 16.538032 16.817919 17.113820 17.425938
## [10,] 16.580558 16.841568 17.120008 17.415948 17.729507 18.060843 18.410148
## [11,] 17.367446 17.656601 17.964953 18.292550 18.639496 19.005933 19.392032
## [12,] 18.153635 18.470767 18.808828 19.167844 19.547895 19.949097 20.371601
## [13,] 18.939127 19.284068 19.651638 20.041834 20.454707 20.890346 21.348867
## [14,] 19.723923 20.096505 20.493384 20.914524 21.359940 21.829687 22.323845
## [15,] 20.508024 20.908081 21.334070 21.785917 22.263599 22.767129 23.296544
## [16,] 21.291432 21.718796 22.173699 22.656019 23.165691 23.702681 24.266979
## [17,] 22.074147 22.528654 23.012273 23.524833 24.066221 24.636352 25.235160
## [18,] 22.856170 23.337656 23.849794 24.392364 24.965196 25.568150 26.201099
## [19,] 23.637504 24.145804 24.686267 25.258615 25.862621 26.498083 27.164809
## [20,] 24.418149 24.953099 25.521693 26.123590 26.758502 27.426160 28.126301
## [21,] 25.198106 25.759544 26.356075 26.987294 27.652845 28.352389 29.085587
```

The value of lambda and q we obtained was 0 in both cases. ## B

```
paralist<-rep(NA)
for(i in 1:length(minigrid)){
  paralist[i]<-optim(rep(0,ncol(x)),myRSSgen,method='CG',x=x,y=y,lam=10,q=minigrid[[i]])[1]
  }
print(paralist)
```

```
## [[1]]
## [1]  1.29815716  0.01252772 -1.19747468 -0.02845786  0.59629345
##
## [[2]]
## [1]  1.302030e+00  1.009134e-05 -1.202964e+00 -1.313452e-02  5.975385e-01
##
## [[3]]
## [1]  1.279859e+00  3.630109e-10 -1.186367e+00 -2.351512e-02  5.890859e-01
##
## [[4]]
## [1]  1.303727e+00  1.408903e-04 -1.203006e+00 -7.739613e-08  5.965547e-01
##
## [[5]]
## [1]  1.294172864  0.000212077 -1.193665323 -0.020115037  0.591909120
##
## [[6]]
## [1]  1.292997004  0.007805155 -1.192814458 -0.023671593  0.591384532
##
## [[7]]
## [1]  1.29172787  0.01022891 -1.19166992 -0.02565416  0.59099172
##
## [[8]]
## [1]  1.29033473  0.01136948 -1.19041623 -0.02680112  0.59074839
##
## [[9]]
## [1]  1.28881407  0.01192218 -1.18906536 -0.02745703  0.59062367
##
## [[10]]
## [1]  1.28716171  0.01218090 -1.18762156 -0.02782111  0.59059444
##
## [[11]]
## [1]  1.28537288  0.01229234 -1.18608659 -0.02801360  0.59064206
##
## [[12]]
## [1]  1.28344263  0.01233108 -1.18446125 -0.02810663  0.59075119
##
## [[13]]
## [1]  1.28136618  0.01233434 -1.18274607 -0.02814259  0.59090909
##
## [[14]]
## [1]  1.27913917  0.01232026 -1.18094162 -0.02814578  0.59110509
##
## [[15]]
## [1]  1.27675797  0.01229755 -1.17904882 -0.02812985  0.59133032
##
## [[16]]
## [1]  1.27421985  0.01227031 -1.17706899 -0.02810233  0.59157734
##
## [[17]]
## [1]  1.2715232  0.0122405 -1.1750039 -0.0280673  0.5918399
##
## [[18]]
## [1]  1.26866781  0.01220907 -1.17285606 -0.02802697  0.59211298
##
## [[19]]
## [1]  1.26565471  0.01217649 -1.17062827 -0.02798251  0.59239217
##
## [[20]]
## [1]  1.26248654  0.01214304 -1.16832408 -0.02793458  0.59267402
##
## [[21]]
## [1]  1.25916749  0.01210887 -1.16594757 -0.02788356  0.59295565
```

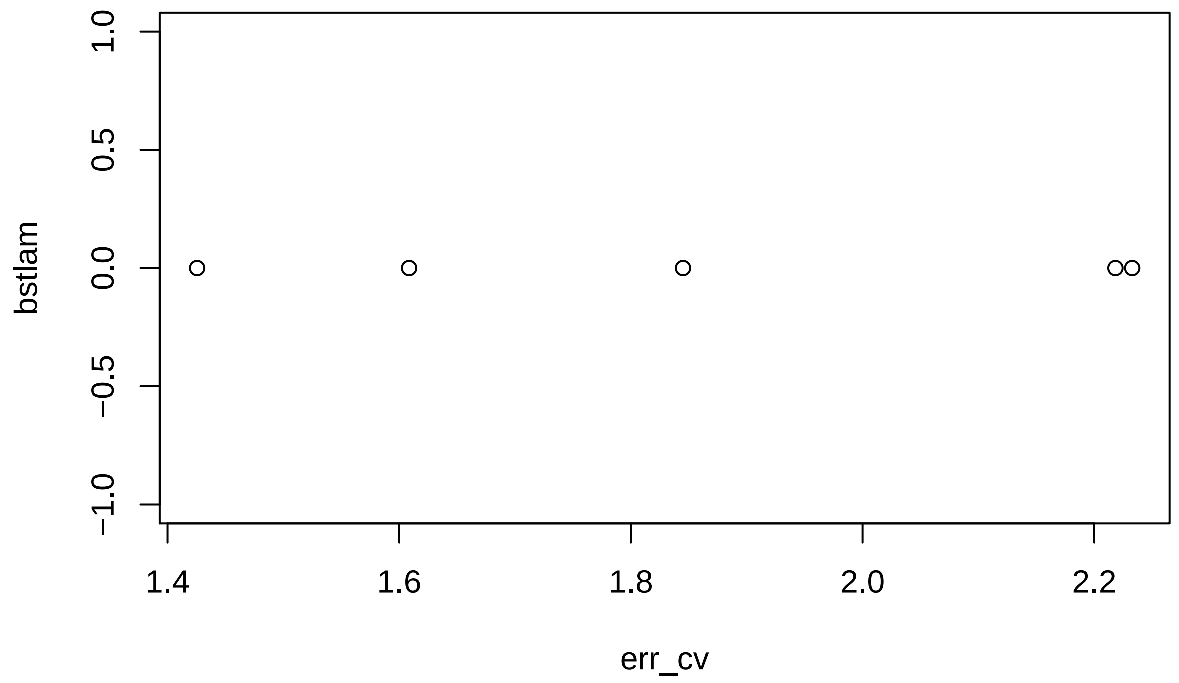
It seems like our first, third, and last beta parameters change very little as a result of us using various values of q, however, it seems like our second and fourth beta parameters change a great deal, especially from q values of 0 to 1.

C

```
K=5
folds = sample(1:K,nrow(data4),replace=T)
optim_list<-rep(NA)
bstlam<-rep(NA)
err_cv<-rep(NA)
betalist<-rep(NA)
for(k in 1:K){
  CV.train = x[folds != k,]
  CV.test = x[folds == k,]
  CV.tr_y = y[folds != k,]
  CV.ts_y = y[folds == k,]
  for(j in 1:length(minigrid)){
    optim_list[j]<-optim(rep(0,ncol(x)),myRSSgen,method='CG',x=CV.train,y=CV.tr_y,lam=minigrid[[j]],q=2
  }
  bstlam[k]<-minigrid[[which.min(optim_list)]]
  betalist[k]<-optim(rep(0,ncol(x)),myRSSgen,method='CG',x=CV.train,y=CV.tr_y,lam=bstlam[[k]],q=2)[1]
    err_cv[k]<-optim(rep(0,ncol(x)),myRSSgen,method='CG',x=CV.test,y=CV.ts_y,lam=bstlam[[k]],q=2)[2]
}
mean(as.numeric(unlist(err_cv)))
```

```
## [1] 1.866015
```

```
plot(x=err_cv, y=bstlam)
```



```
print("best parameter estimates below")
```

```
## [1] "best parameter estimates below"
```

```
optim(rep(0,ncol(x)),myRSSgen,method='CG',x=x,y=y,lam=0,q=2)[1]
```

```
## $par
## [1]  1.29815716  0.01252772 -1.19747468 -0.02845786  0.59629345
```

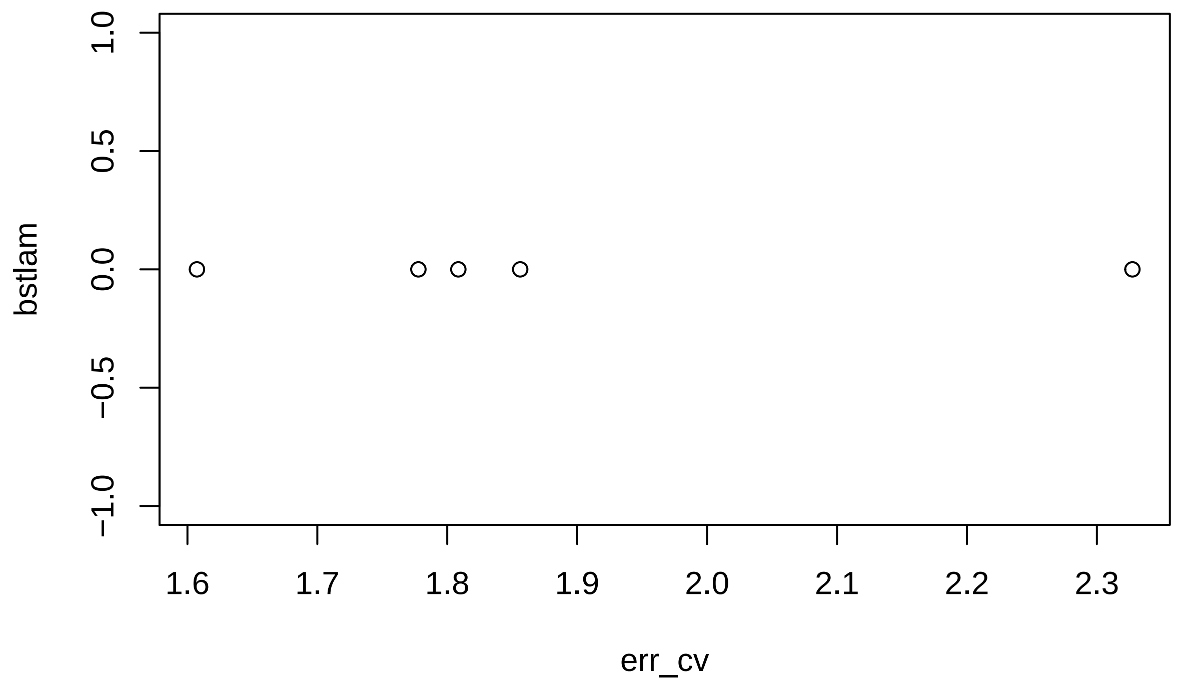
D

```
K=5
folds = sample(1:K,nrow(data4),replace=T)
optim_list<-rep(NA)
bstlam<-rep(NA)
err_cv<-rep(NA)
betalist<-rep(NA)
for(k in 1:K){
  CV.train = x[folds != k,]
  CV.test = x[folds == k,]
  CV.tr_y = y[folds != k,]
  CV.ts_y = y[folds == k,]
  for(j in 1:length(minigrid)){
    optim_list[j]<-optim(rep(0,ncol(x)),myRSSgen,method='CG',x=CV.train,y=CV.tr_y,lam=minigrid[[j]],q=1
  }
  bstlam[k]<-minigrid[[which.min(optim_list)]]
  betalist[k]<-optim(rep(0,ncol(x)),myRSSgen,method='CG',x=CV.train,y=CV.tr_y,lam=bstlam[[k]],q=1)[1]
```

```
    err_cv[k]<-optim(rep(0,ncol(x)),myRSSgen,method='CG',x=CV.test,y=CV.ts_y,lam=bstlam[[k]],q=1)[2]
}
mean(as.numeric(unlist(err_cv)))
```

```
## [1] 1.875424
```

```
plot(x=err_cv, y=bstlam)
```



```
print("best parameter estimates below")
```

```
## [1] "best parameter estimates below"
```

```
optim(rep(0,ncol(x)),myRSSgen,method='CG',x=x,y=y,lam=0,q=1)[1]
```

```
## $par
## [1]  1.29815716  0.01252772 -1.19747468 -0.02845786  0.59629345
```

E

```
K=5
folds = sample(1:K,nrow(data4),replace=T)
optim_list<-rep(NA)
bstlam<-rep(NA)
bstq<-rep(NA)
err_cv<-rep(NA)
betalist<-rep(NA)
optim_list<-matrix(NA, nrow=length(minigrid), ncol=length(minigrid))
```

```
for(k in 1:K){
  CV.train = x[folds != k,]
  CV.test = x[folds == k,]
  CV.tr_y = y[folds != k,]
  CV.ts_y = y[folds == k,]
  for(i in 1:length(minigrid)){
    for(j in 1:length(minigrid)){
      optim_list[i,j]<-as.numeric(optim(rep(0,ncol(x)),myRSSgen,method='CG',x=x,y=y,lam=minigrid[i],q=mi
    }
  }
  obj<-which(optim_list == min(optim_list), arr.ind=TRUE)
  bstlam[k]<-minigrid[[obj[1,1]]]
  bstq[k]<-minigrid[[obj[1,2]]]
  err_cv[k]<-optim(rep(0,ncol(x)),myRSSgen,method='CG',x=CV.test,y=CV.ts_y,lam=bstlam[[k]],q=bstq[[k]])
}
mean(as.numeric(unlist(err_cv)))
```

```
## [1] 1.856078
```

```
#bstlam
#bstq

#contour(x=bstlam, y=bstq, z=err_cv)

print("best parameter estimates below")
```

```
## [1] "best parameter estimates below"
```

```
optim(rep(0,ncol(x)),myRSSgen,method='CG',x=x,y=y,lam=0,q=0)[1]
```

```
## $par
## [1]  1.29815716  0.01252772 -1.19747468 -0.02845786  0.59629345
```

Given that our estimates for best lambda and best q values were 0 across all our folds, the ‘contour’ function was unable to produce a contour plot, as it expected increasing x and y values.

F

Overall, it seems like there is not much point in considering different values of q. The reasoning I have for this is fairly simple, that conceptually, values of q other than 1 or 2 make very little sense from a conceptual standpoint, looking at our formula for lasso and ridge regression as a generalized form of the equation we were presented at the beginning of the problem.

Additionally, from a test error generation standpoint, it seems like values of q wherein q is set to 0 seem to dominate all other options, to the exclusion of the importance of various values of lambda as well. Given that lambda has essentially no effect on test error if q is 0, I would posit that there is no purpose to consider different values of q in this case.