HW #6

Bilal Gilani

10/26/2020

1.

a.

Lasso compared to least square regression

iii.: The method is less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

b.

Ridge regression compared to least square regression

iii.:The method is less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.

c.

Fitting non-linear trends compared to least square regression

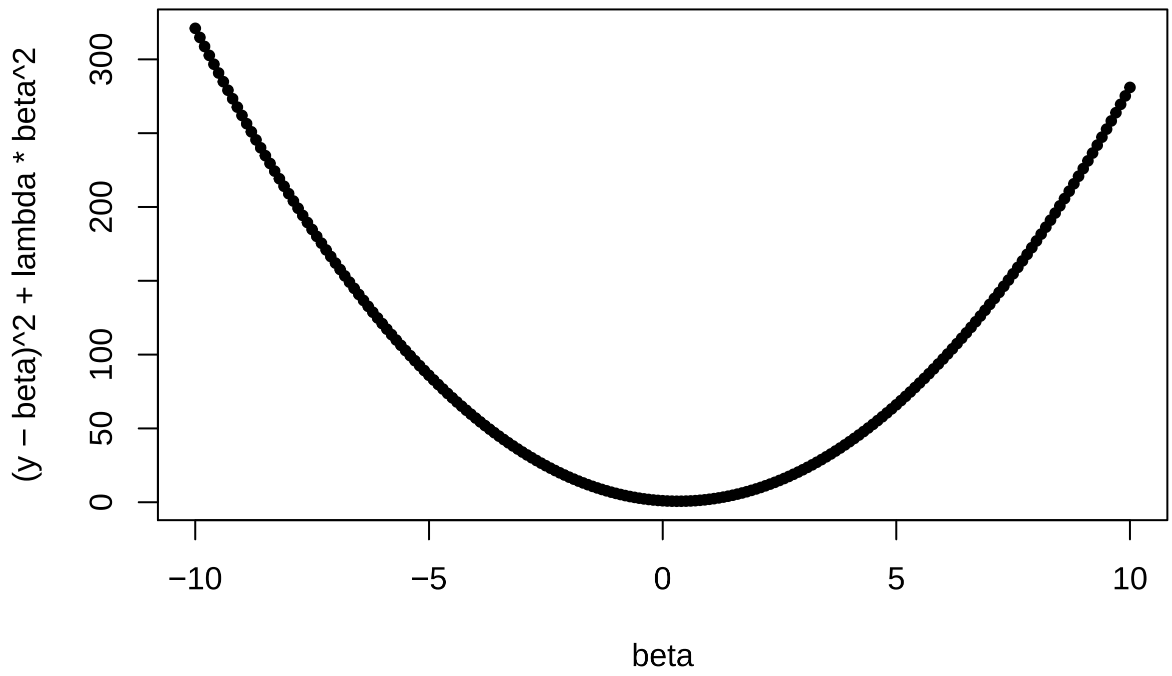
ii.: The method is more flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.

2.

a.

```r
y <- 1
lambda <- 2
beta <- seq(-10, 10, 0.1)

plot(beta, (y - beta)^2 + lambda * beta^2, pch = 20)
```



Ridge Minima:

```
y / (1 - lambda)
```

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

Lasso Minima:

```
lambda / 2
```

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

```
-lambda /2
```

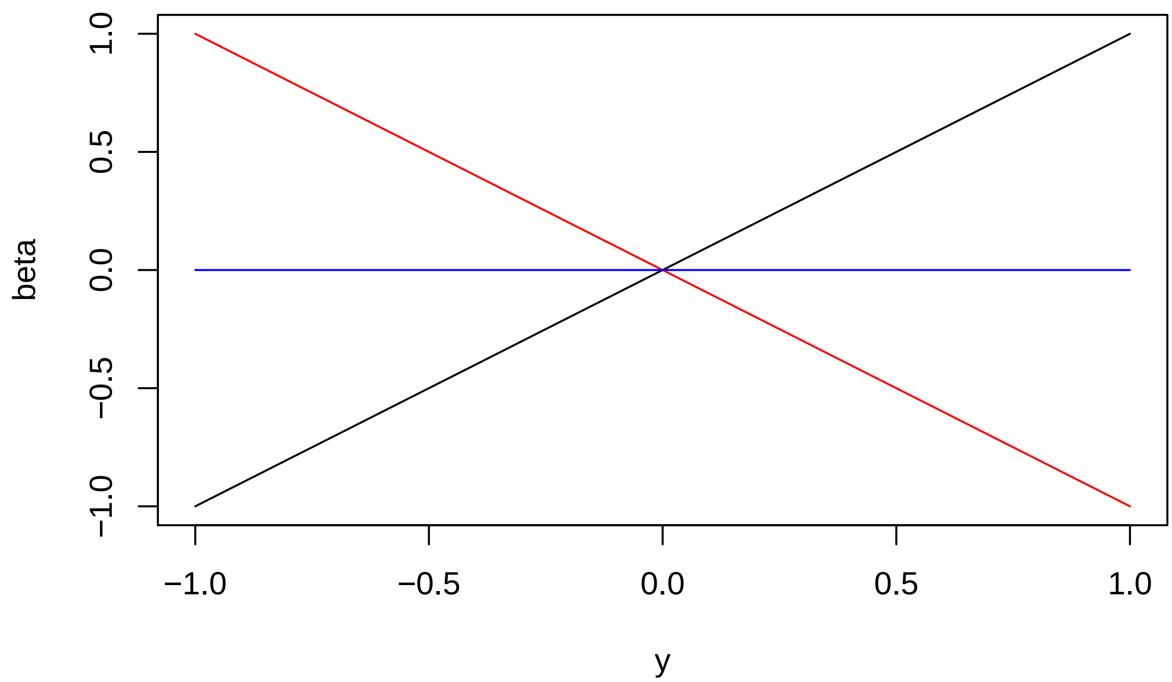
```
## [1] -1
```

Since the absolute value of Y is less than or equal to lamba divided by 2, the lasso minima must be 0.

b.

```
y <- seq(-1, 1, 0.1)
beta <- seq(-1, 1, 0.1)
```

```r
plot(y, beta, "l")
lines(y, beta * -1, pch = 20, col = "red")
lines(y, beta * 0, pch = 20, col = "blue") ## not sure here
```



3.

a.

```r
X <- rnorm(100)
noise <- rnorm(100)
```

b.

```r
b0 <- 2
b1 <- 4
b2 <- 6
b3 <- 0.8

y <- b0 + b1 * X + b2 * X^2 + b3 * X^3 + noise

y
```

```
##   [1]  1.4635846  3.2502449 19.5142870  1.1288948  1.4086986  3.2156547
##   [7] 10.7634061 14.0986999  1.7049302  1.1236787  3.8571032  6.6079479
##  [13] 15.7724161  3.6357741  7.6294564  2.8463619  1.2877445  1.0372165
##  [19]  8.6081368  8.0399489  8.1008851  2.8278590  4.4721273  1.9126597
##  [25]  1.9837081  2.1867059 36.1863401  1.7473950  5.0672877  3.0120499
##  [31]  2.8604728 10.0605421  2.1828396 20.6537523 17.6033750 34.7773385
##  [37]  0.5626263  8.1149128 11.1711692 62.9160384  1.5064440  4.0488997
##  [43]  2.6068003  1.7712095  0.8183634  5.6178606  1.2170076  8.6091862
##  [49]  2.2435415  1.6398006  9.2448592  1.0446455  7.0772592 17.5957204
##  [55] 13.9824343 33.8397171 22.5574469  6.5302452  2.9126693  1.9407470
##  [61]  0.4269583  2.3783346  4.9217684 18.5000779  2.5924047 12.7072724
##  [67] 16.0662150  1.9844535  1.4690360 14.9392726  2.9294760  1.4463776
##  [73]  3.9883046  1.0078639  1.8224136 13.0033884  3.2567705 -0.2351393
##  [79]  4.2620047  2.3634691  3.0385362  7.3332733  6.1330150  2.7196524
##  [85]  3.3584210 17.9919733  7.9321792  2.1133594  2.7886191  3.2149062
##  [91] 22.7372947 13.1045854  3.4264856 12.6926479 20.6164275 12.2530162
##  [97]  3.0276179 48.4054842  2.6533816 18.2004260
```

c.

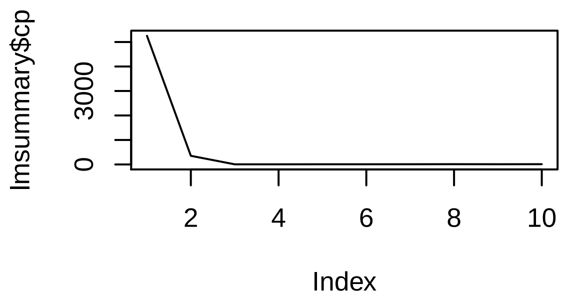
```r
library(leaps)
```

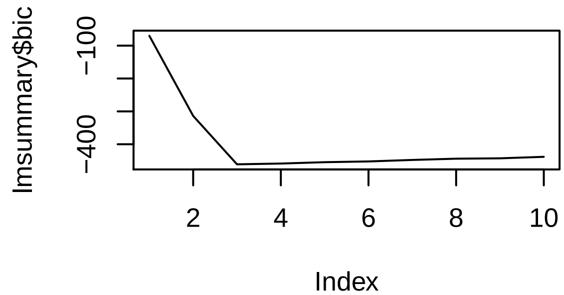
```
## Warning: package 'leaps' was built under R version 3.6.3
```

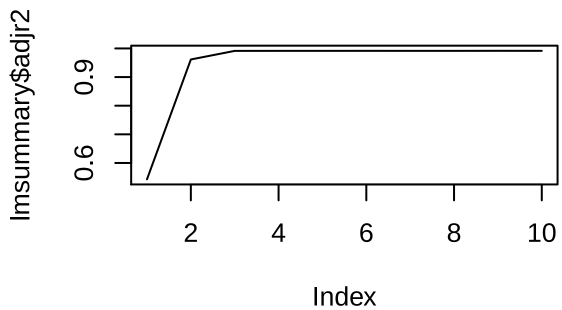
```r
df <- data.frame(y = y, x = X)
lm.full <- regsubsets(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I

lmsummary <- summary(lm.full)

par(mfrow = c(2, 2))
plot(lmsummary$cp, type = "l")
plot(lmsummary$bic, type = "l")
plot(lmsummary$adjr2, type = "l")
```







It seems as though the values we should select is about 3, maybe 4. The next step would be to see the where CP and BIC are at their lowest and ADJR2 is at its peak.

```
coef(lm.full, which.max(lmsummary$adjr2))
```

```
##  (Intercept)            x       I(x^2)       I(x^3)       I(x^5)
##  1.925770660  3.109943141  6.367904268  3.165136525 -1.533712965
##       I(x^6)       I(x^7)       I(x^8)       I(x^9)      I(x^10)
## -0.210221410  0.360518186  0.074699417 -0.027442298 -0.006751482
```

```
coef(lm.full, which.min(lmsummary$bic))
```

```
## (Intercept)           x      I(x^2)      I(x^3)
##   1.9766799   4.1544570   6.1023541   0.7670445
```

```
coef(lm.full, which.min(lmsummary$cp))
```

```
##   (Intercept)             x        I(x^2)        I(x^3)        I(x^9)
##  2.0057531124  3.9929513018  6.0579659621  0.8716846279 -0.0002848474
```

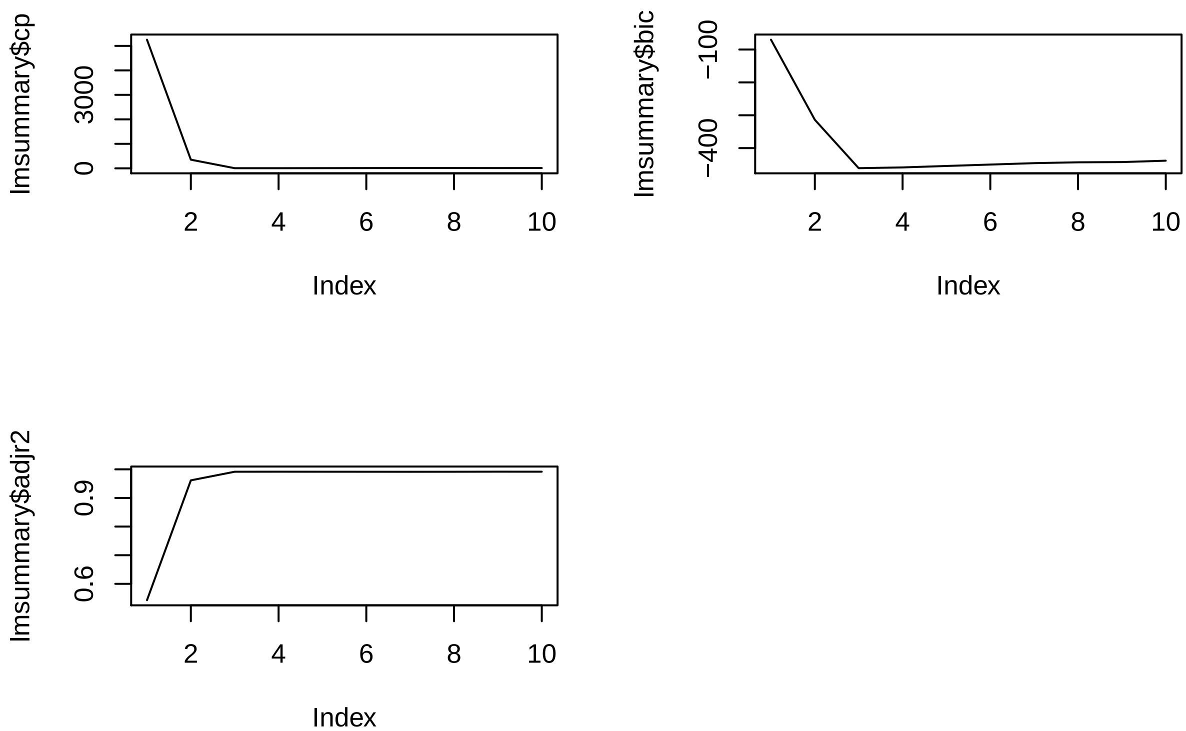
The output shows that ADJR2 suggest 7 variables, BIC suggests 3 variables, and CP suggests 7 variables.

d.

```
lm.full <- regsubsets(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I

lmsummary <- summary(lm.full)

par(mfrow = c(2, 2))
plot(lmsummary$cp, type = "l")
plot(lmsummary$bic, type = "l")
plot(lmsummary$adjr2, type = "l")
```




```
coef(lm.full, which.max(lmsummary$adjr2))
```

```
##  (Intercept)            x       I(x^2)       I(x^3)       I(x^4)
##  1.895564477  3.155662065  6.675270857  3.009930138 -0.445763214
##       I(x^5)       I(x^7)       I(x^8)       I(x^9)      I(x^10)
## -1.399975110  0.323088956  0.035666487 -0.024282610 -0.004258255
```

```
coef(lm.full, which.min(lmsummary$bic))
```

```
## (Intercept)           x      I(x^2)      I(x^3)
##   1.9766799   4.1544570   6.1023541   0.7670445
```

```r
coef(lm.full, which.min(lmsummary$cp))
```

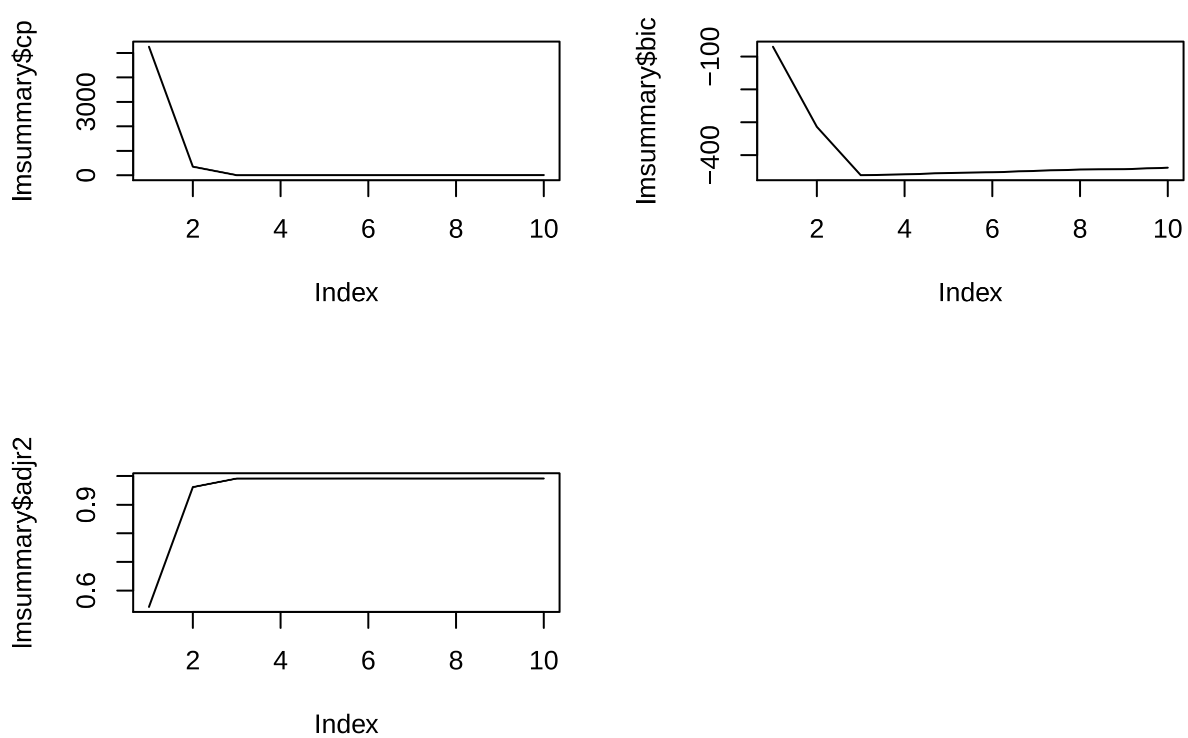
```
##   (Intercept)             x        I(x^2)        I(x^3)        I(x^9)
##  2.0057531124  3.9929513018  6.0579659621  0.8716846279 -0.0002848474
```

When using forward selection, ADJR2 suggests 10 variables, BIC suggests 3 variables, and CP suggests 6 variables.

```r
lm.full <- regsubsets(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I

lmsummary <- summary(lm.full)

par(mfrow = c(2, 2))
plot(lmsummary$cp, type = "l")
plot(lmsummary$bic, type = "l")
plot(lmsummary$adjr2, type = "l")
```




```r
coef(lm.full, which.max(lmsummary$adjr2))
```

```
##  (Intercept)            x       I(x^2)       I(x^3)       I(x^5)
##  1.925770660  3.109943141  6.367904268  3.165136525 -1.533712965
##       I(x^6)       I(x^7)       I(x^8)       I(x^9)      I(x^10)
## -0.210221410  0.360518186  0.074699417 -0.027442298 -0.006751482
```

```r
coef(lm.full, which.min(lmsummary$bic))
```

```
## (Intercept)           x      I(x^2)      I(x^3)
##   1.9766799   4.1544570   6.1023541   0.7670445
```

```r
coef(lm.full, which.min(lmsummary$cp))
```

```
##   (Intercept)             x        I(x^2)        I(x^3)        I(x^9)
##  2.0057531124  3.9929513018  6.0579659621  0.8716846279 -0.0002848474
```

When using backward selection, all three suggests 7 variables.

e.

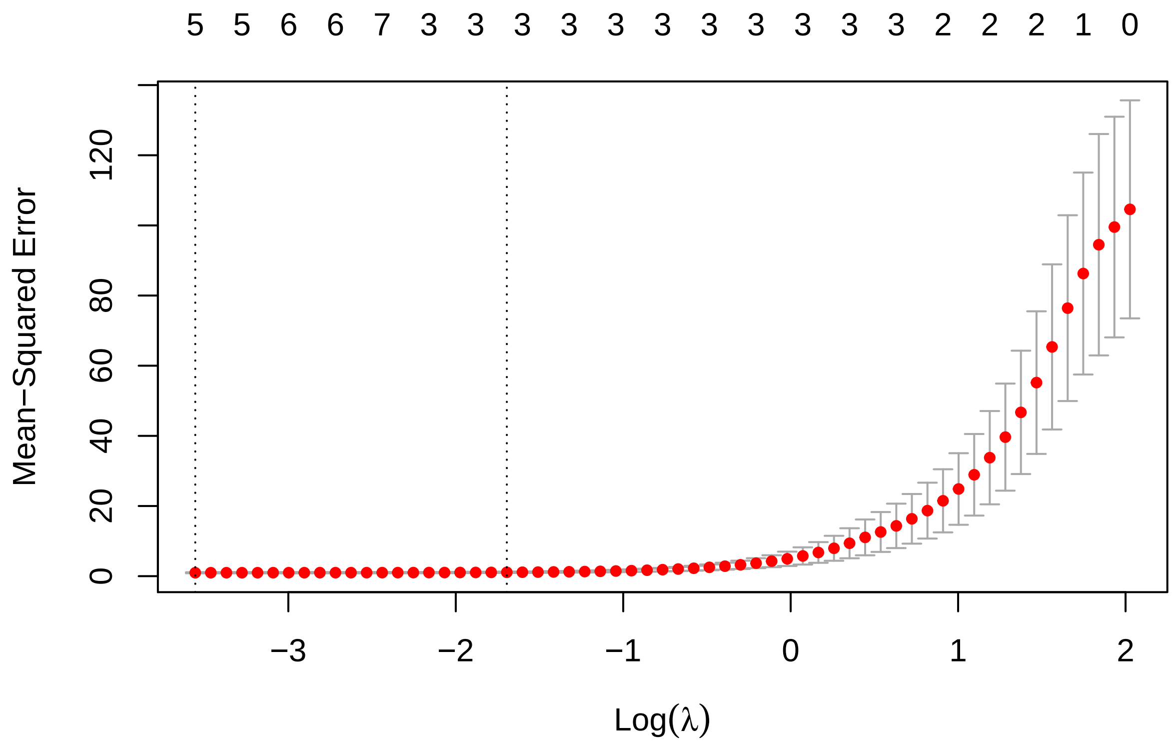
```r
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.6.3
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.0-2
```

```r
Xmatrix <- model.matrix(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) +
cv.lasso <- cv.glmnet(Xmatrix, y, alpha = 1)
plot(cv.lasso)
```




```r
lambdamin <- cv.lasso$lambda.min
lambdamin
```

```
## [1] 0.0285612
```

```r
lasso <- glmnet(Xmatrix, y, alpha = 1)
predict(lasso, s = lambdamin, type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) 2.046361e+00
## x           4.124532e+00
## I(x^2)      6.002840e+00
## I(x^3)      7.782363e-01
## I(x^4)      .
## I(x^5)      .
## I(x^6)      .
## I(x^7)      .
## I(x^8)      1.351220e-04
## I(x^9)      .
## I(x^10)     3.091923e-05
```

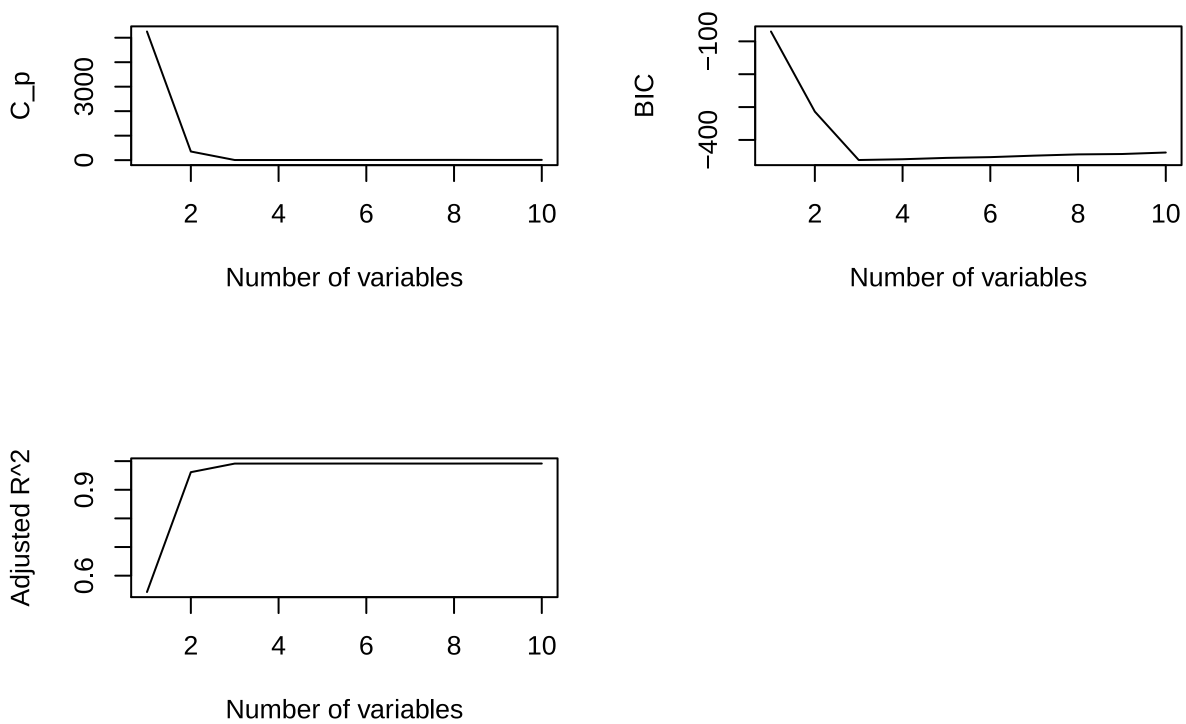
The results of the Lasso model coefficients above show that our model chose variables x, x^2, x^3, x^9, and x^10 for the final model predictors (in addition to the intercept).

f.

```r
b7 <- 7
y <- b0 + b7 * X^7 + noise
regfit <- regsubsets(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I(

regsummary <- summary(regfit)

par(mfrow = c(2, 2))
plot(regsummary$cp, xlab = "Number of variables", ylab = "C_p", type = "l")
plot(regsummary$bic, xlab = "Number of variables", ylab = "BIC", type = "l")
plot(regsummary$adjr2, xlab = "Number of variables", ylab = "Adjusted R^2", type = "l")
```




```r
coef(regfit, which.max(regsummary$adjr2))
```

```
##  (Intercept)            x       I(x^2)       I(x^3)       I(x^5)
##  1.925770660  3.109943141  6.367904268  3.165136525 -1.533712965
##       I(x^6)       I(x^7)       I(x^8)       I(x^9)      I(x^10)
## -0.210221410  0.360518186  0.074699417 -0.027442298 -0.006751482
```

```r
coef(regfit, which.min(regsummary$bic))
```

```
## (Intercept)           x      I(x^2)      I(x^3)
##   1.9766799   4.1544570   6.1023541   0.7670445
```

```r
coef(regfit, which.min(regsummary$cp))
```

```
##   (Intercept)             x        I(x^2)        I(x^3)        I(x^9)
##  2.0057531124  3.9929513018  6.0579659621  0.8716846279 -0.0002848474
```

Both ADJR2 and CP suggest a 7 variable model, whereas BIC suggests a 3 variable.

```r
cv.lasso <- cv.glmnet(Xmatrix, y, alpha = 1)
labmdamin <- cv.lasso$lambda.min
lambdamin
```

```
## [1] 0.0285612
```

```r
lasso <- glmnet(Xmatrix, y, alpha = 1)
predict(lasso, s = labmdamin, type = "coefficients")
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     1
## (Intercept) -2.005345
## x            .
## I(x^2)       .
## I(x^3)       .
## I(x^4)       .
## I(x^5)       .
## I(x^6)       .
## I(x^7)      6.795160
## I(x^8)       .
## I(x^9)       .
## I(x^10)      .
```

The results of Lasso suggest that our model picks x^5, x^7, x^9 and the intercept as predictors.

4.

```r
library(ISLR)
```

```
## Warning: package 'ISLR' was built under R version 3.6.3
```

```r
data("College")
```

```r
attach(College)
set.seed(100)
n <- length(Apps)
Z <- sample(n,n/2)

train <- College[Z, ]
test <- College[-Z, ]
```

a.

LEAST SQUARES REGRESSION

```
lm.train <- lm(Apps ~ ., data = train)
lm.test <- predict(lm.train, newdata = test)

mse <- mean((lm.test - test$Apps)^2)

rmse1 <- sqrt(mse)

rmse1
```

```
## [1] 1226.581
```

The RMSE when it comes to LEAST SQUARES Regression is **1226.581**. This amount is the variance we can expect in terms of applications. So we can expect our prediction to be incorrect by about 1,188 applications.

b.

RIDGE REGRESSION

```
train2 <- model.matrix(Apps ~ ., data = train)
test2 <- model.matrix(Apps ~ ., data = test)

grid <- 10^seq(4,-2,length = 100)

library(glmnet)
ridge <- glmnet(train2, train$Apps, alpha = 0, lambda = grid)
cv.ridge <- cv.glmnet(train2, train$Apps, alpha = 0, lambda = grid) ## cross-validation

lambdamin <- cv.ridge$lambda.min

lambdamin
```

```
## [1] 14.17474
```

```
pred.ridge <- predict(ridge, s = lambdamin, newx = test2)

mse <- mean((test$Apps - pred.ridge)^2)

rmse2 <- sqrt(mse)

rmse2
```

```
## [1] 1253.288
```

The RMSE when it comes to RIDGE regression is **1253.288**. This is a larger value than that of the least squares regression, as a result, ridge regression performs worse than least squares regression.

c.

LASSO

```
lasso <- glmnet(train2, train$Apps, alpha = 1, lambda = grid)
cv.lasso <- cv.glmnet(train2, train$Apps, alpha = 1, lambda = grid) ## cross validation, note the diffe

lambdamin <- cv.lasso$lambda.min

pred.lasso <- predict(lasso, s = lambdamin, newx = test2)

mse <- mean((test$Apps - pred.lasso)^2)

rmse3 <- sqrt(mse)

rmse3
```

```
## [1] 1230.749
```

The RMSE when it comes to LASSO regression is **1230.749**. This is a larger value than that of the least squares regression, but lower than that of regression. Therefore, lasso regression performs worse than least squares regression (not by much) but better than ridge regression.

d.

PCR model

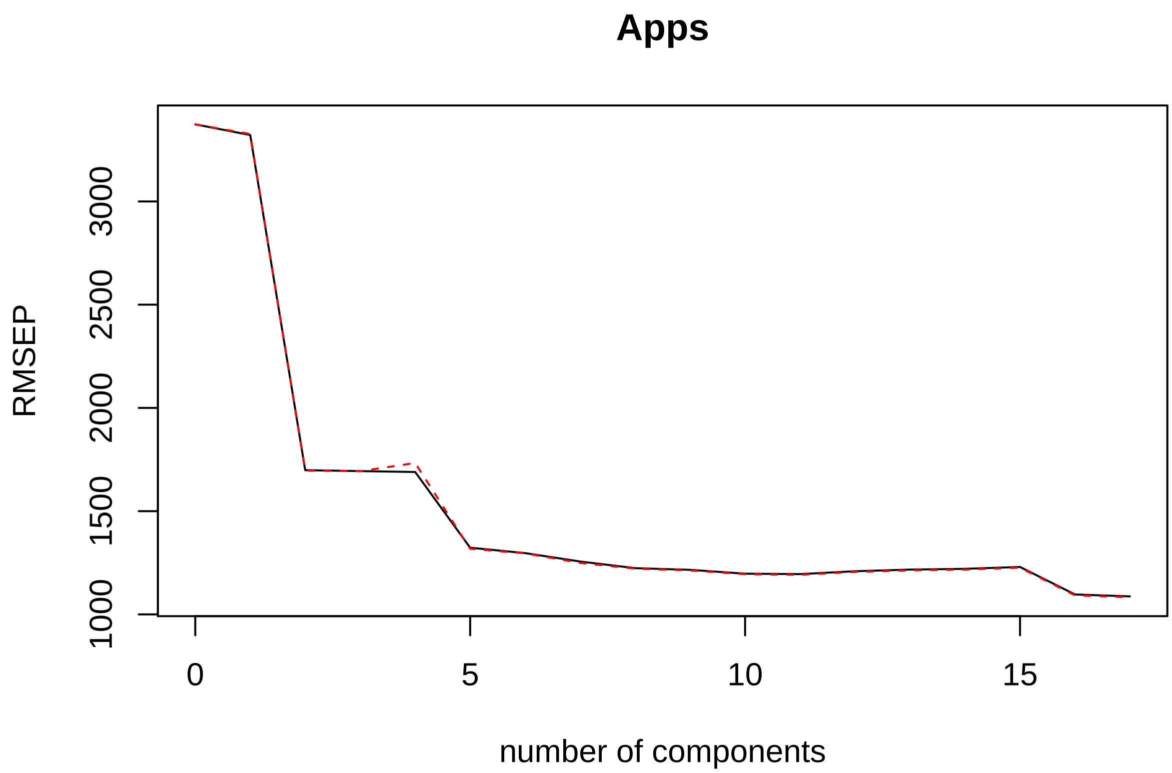
```
library(pls)
```

```
## Warning: package 'pls' was built under R version 3.6.3
```

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

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

```
pcrmodel <- pcr(Apps ~ . - Private + as.factor(Private), data = train, scale = TRUE, validation = "CV")
validationplot(pcrmodel)
```




```r
summary(pcrmodel)
```

```
## Data:    X dimension: 388 17
##  Y dimension: 388 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            3373     3321     1699     1694     1690     1323     1297
## adjCV         3373     3327     1696     1694     1733     1318     1296
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1256     1224     1216      1197      1195      1209      1217
## adjCV     1249     1221     1213      1195      1192      1206      1213
##        14 comps  15 comps  16 comps  17 comps
## CV         1221      1230      1096      1087
## adjCV      1217      1226      1092      1083
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X      30.926    57.63    64.72    70.66    76.23    80.63    84.43
## Apps    4.911    75.42    75.68    75.89    85.82    86.28    87.22
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       87.76    90.78     93.23     95.17     97.01     98.10     98.88
```

```
## Apps     87.79    87.93     88.26     88.35     88.35     88.38     88.38
##        15 comps  16 comps  17 comps
## X         99.45     99.88    100.00
## Apps      88.39     90.82     90.98
```

```
## based off the plot and summary, seems like 10 is the appropriate value for M

pred.pcr <- predict(pcrmodel, test, ncomp = 10)

mse <- mean((pred.pcr - test$Apps)^2)

rmse4 <- sqrt(mse)

rmse4
```

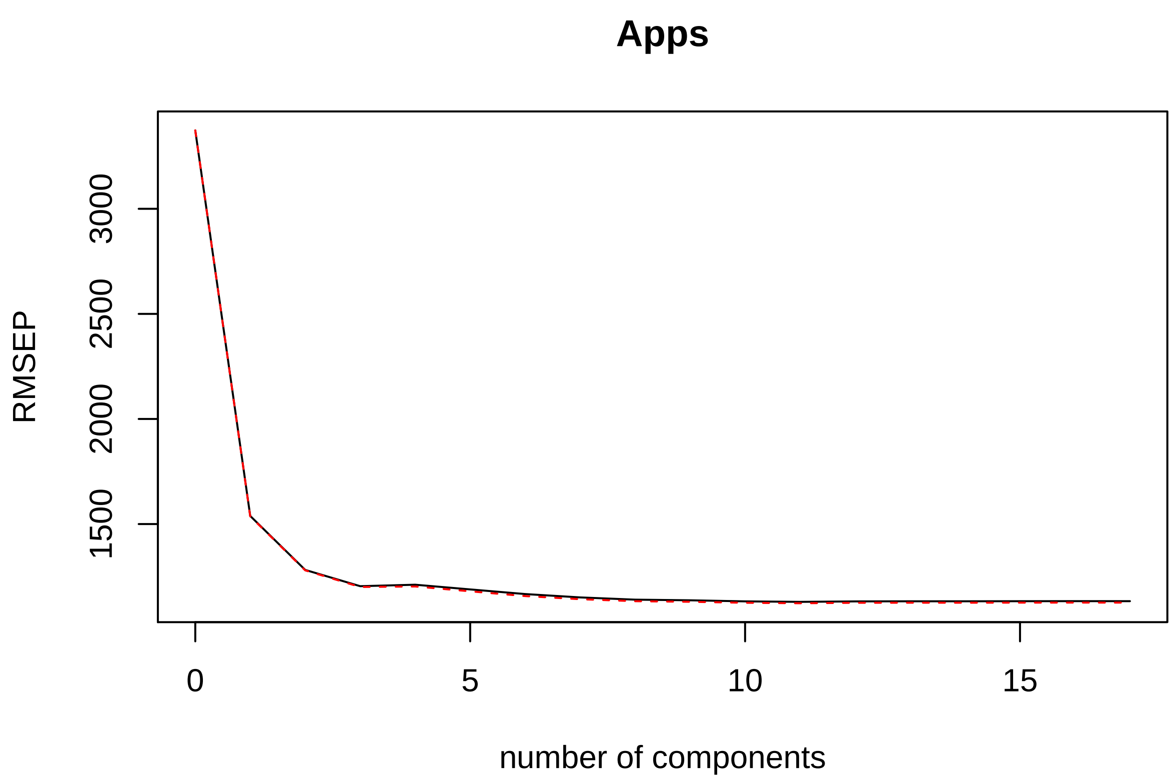
```
## [1] 1737.057
```

The RMSE when it comes to PCR is **1737.057**. This is a larger value than least squares, ridge, and lasso regression. It has so far performed the worst out of all the models we have ran thus far.

e.

PLS model

```
plsmodel <- plsr(Apps ~ . - Private + as.factor(Private), data = train, scale = TRUE, validation = "CV")
validationplot(plsmodel)
```



```r
summary(plsmodel)## lowest CV is at 10 again
```

```
## Data:    X dimension: 388 17
##  Y dimension: 388 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            3373     1539     1282     1204     1211     1189     1167
## adjCV         3373     1536     1280     1200     1203     1180     1157
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1151     1140     1137      1132      1130      1132      1132
## adjCV     1142     1133     1130      1125      1123      1125      1126
##        14 comps  15 comps  16 comps  17 comps
## CV         1132      1133      1133      1133
## adjCV      1126      1126      1126      1126
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X       26.93     40.4    62.97    66.63    70.60    74.23    77.32
## Apps    79.93     86.7    88.65    89.51    90.27    90.81    90.91
##       8 comps  9 comps  10 comps  11 comps  12 comps  13 comps  14 comps
## X       81.12    84.81     87.58     89.79     91.85     93.80     96.33
## Apps    90.92    90.94     90.95     90.97     90.98     90.98     90.98
##       15 comps  16 comps  17 comps
## X        97.45     98.25    100.00
## Apps     90.98     90.98     90.98
```

```r
pred.pls <- predict(plsmodel, test, ncomp = 10)

mse <- mean((pred.pls - test$Apps)^2)

rmse5 <- sqrt(mse)

rmse5
```

```
## [1] 1228.742
```

The RMSE when it comes to PLS is **1228.742**. This is a the lowest value of the five models that were tested, it performs better than least squares, ridge, lasso, and PCR regression.

So what do the values mean?

```r
## LEAST SQUARES
rmse1
```

```
## [1] 1226.581
```

```r
## RIDGE
rmse2
```

```
## [1] 1253.288
```

```r
## LASSO
rmse3
```

```
## [1] 1230.749
```

```r
## PCR
rmse4
```

```
## [1] 1737.057
```

```r
## PLS
rmse5
```

```
## [1] 1228.742
```

Because PLS had the lowest value, it can be inferred that it would be the most accurate choice for a model.