STOR 565 Fall 2019 Homework 3

Hunter Finger

Remark. This homework aims to help you further understand the model selection techniques in linear model. Credits for **Theoretical Part** and **Computational Part** are in total 100 pt. For **Computational Part** , please complete your answer in the **RMarkdown** file and summit your printed PDF homework created by it.

Computational Part

Hint. Before starting your work, carefully read Textbook Chapter 6.5-6.7 (Lab 1-3). Mimic the related analyses you learn from it. Also look at the demonstrations I showed you in class (see Sakai/Resources/Lectures and click on each Lecture for demonstrations). Some related packages have been loaded in setup.

1. (Model Selection, Textbook 6.8, *18 pt*) In this exercise, we will generate simulated data, and will then use this data to perform model selection.

(a) Use the `rnorm` function to generate a predictor $\boldsymbol{X}$ of length $n = 100$, as well as a noise vector $\boldsymbol{\epsilon}$ of length $n = 100$. Do not print the entire vector.

```
set.seed(83)
X = rnorm(100)
epsilon = rnorm(100)
```

Hint. Before generating random numbers, fix your favourite random seed by `set.seed` so that your result is reproducible as you carry forward your exploration.

(b) Generate a response vector $\boldsymbol{Y}$ of length $n = 100$ according to the model

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon,$$

where $\beta_0 = 3$, $\beta_1 = 2$, $\beta_2 = -3$, $\beta_3 = 0.3$. Do not print the entire vector.

```
b0 = 3
b1 = 2
b2 = -3
b3 = .3

Y = (b0 + b1*X + b2*(X^2) + b3*(X^3) + epsilon)
Y
```

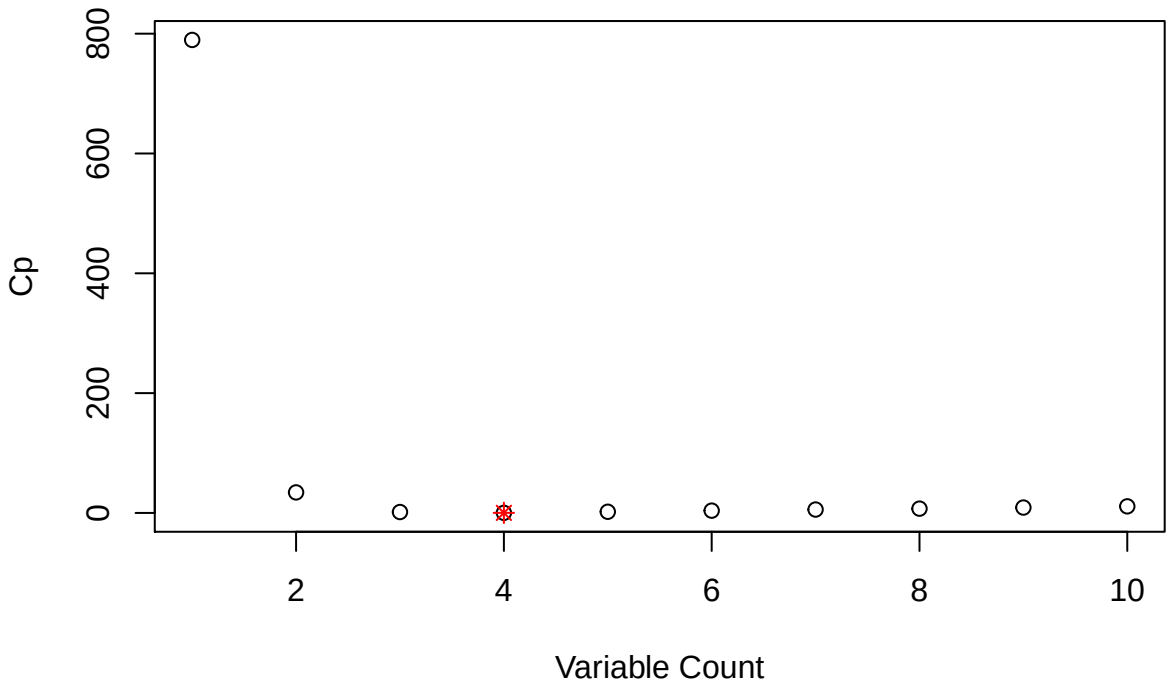
```
##   [1] -21.15695351   2.08731336   3.91353355   2.47193610   3.05994391
##   [6]  -1.24190889  -4.33719146   3.98933354   1.57967056   2.69168694
##  [11]   5.13207946  -1.62051939   2.67813847   4.33235602   0.81224207
##  [16]  -0.21915904  -6.31696604   3.67922212  -4.63460999   4.67751387
##  [21]   2.58913682   1.61653930   2.03991559   3.02838355   2.50314766
##  [26]   3.76867983   4.60268732   1.45102111  -0.77139685 -13.50514893
##  [31]   4.72567725  -9.62398229  -0.49131531   4.64952753   1.97029004
##  [36]   3.17817245  -3.90967159   1.32591893 -10.39968442   3.41982733
##  [41]   3.21334593  -9.72550272   0.26315785   1.98470069  -0.56220220
##  [46]   4.12687555  -4.23788360   2.32010894   3.28138271   3.09572439
##  [51]   3.23993106  -6.23940801   1.22906505   1.43636275   2.69100830
##  [56]  -5.84888300   3.96389068   1.32166709   3.16770068   3.10883561
##  [61]   2.74917676  -6.10157968   3.02909238   3.00756113   3.12154219
##  [66]   2.89252680   3.71528495  -0.59309832   3.22970983 -22.28504483
##  [71]   0.80155939 -12.43490492  -4.94768558   3.20215678   0.19099885
##  [76]  -1.38458033   3.17923065   2.06384047   0.02649022   3.88111228
```

```
##  [81]   2.81040363   1.86026955  -3.34262077   0.11838114   4.01196136
##  [86]   2.08990682   2.87330883   4.14261190   3.70125308  -0.63842861
##  [91]   1.71451438  -1.56482579   0.49495713  -3.71846361  -9.83265055
##  [96]   1.45205006   1.05022106  -2.31334160   0.90222507   3.35419359
```

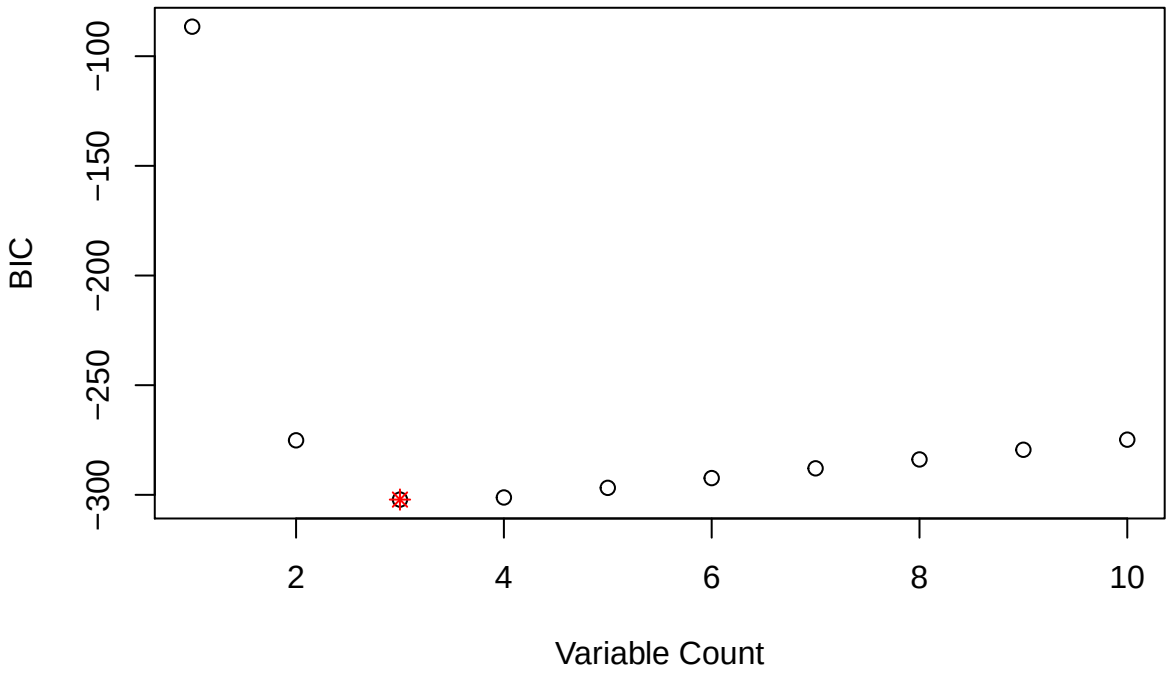
(c) Use the `regsubsets` function from `leaps` package to perform best subset selection in order to choose the best model containing the predictors $(X, X^2, \cdots, X^{10})$.

What is the best model obtained according to C_p, BIC, and adjusted R^2? Show some plots to provide evidence for your answer, and report the coefficients of the best model obtained.

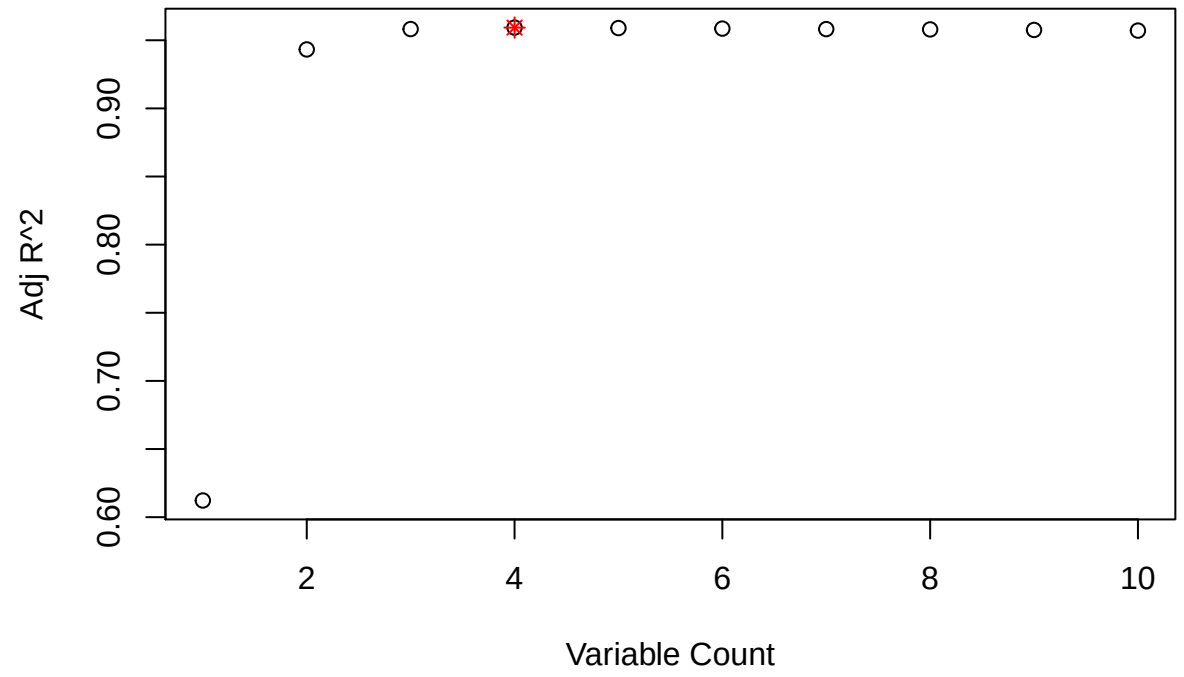
```
full = data.frame(Y, X)
sub = regsubsets(Y ~ X + I(X^2) + I(X^3) + I(X^4) + I(X^5)
                          + I(X^6) + I(X^7) + I(X^8)+ I(X^9)+ I(X^10)
                          , data = full, nvmax=10)
sub.summary = summary(sub)
plot(sub.summary$cp, xlab = "Variable Count", ylab = "Cp")
points(which.min(sub.summary$cp), sub.summary$cp[which.min(sub.summary$cp)],
       col = "red", cex = 1, pch = 8)
```




```
plot(sub.summary$bic, xlab = "Variable Count", ylab = "BIC")
points(which.min(sub.summary$bic), sub.summary$bic[which.min(sub.summary$bic)],
       col = "red", cex = 1, pch = 8)
```




```
plot(sub.summary$adjr2, xlab = "Variable Count", ylab = "Adj R^2")
points(which.max(sub.summary$adjr2), sub.summary$adjr2[which.max(sub.summary$adjr2)],
       col = "red", cex = 1, pch = 8)
```

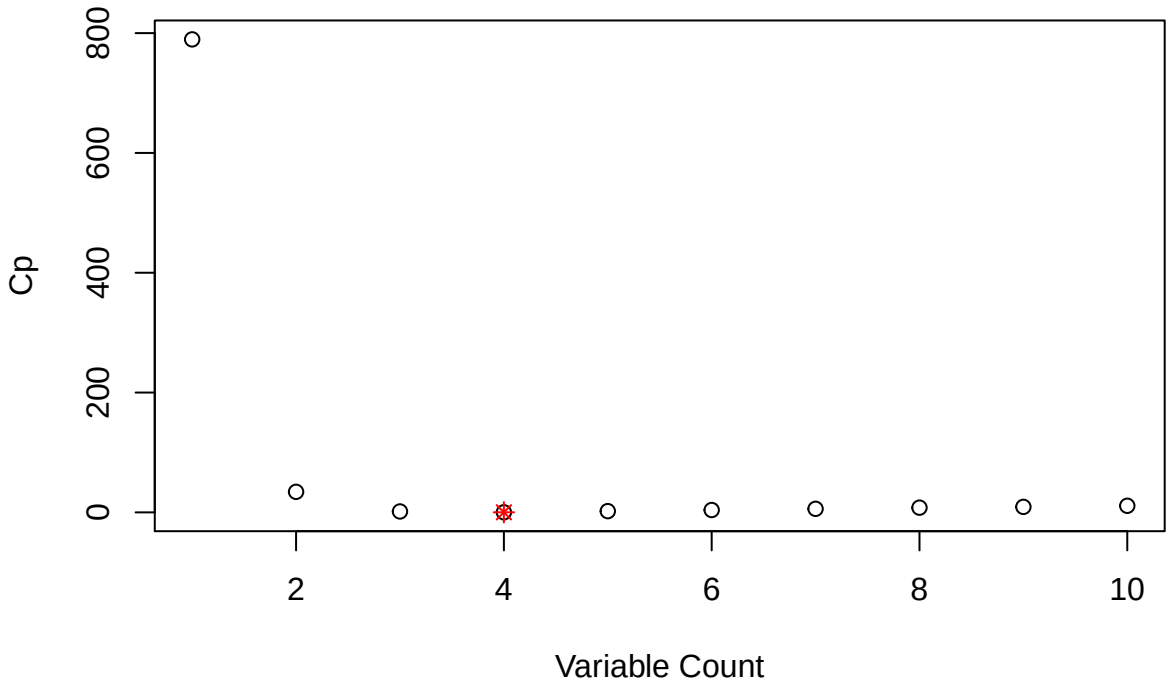



```
With 2 models, the number of variables chosen for the model is 4. Those models are Cp and Adjusted R Sq
```

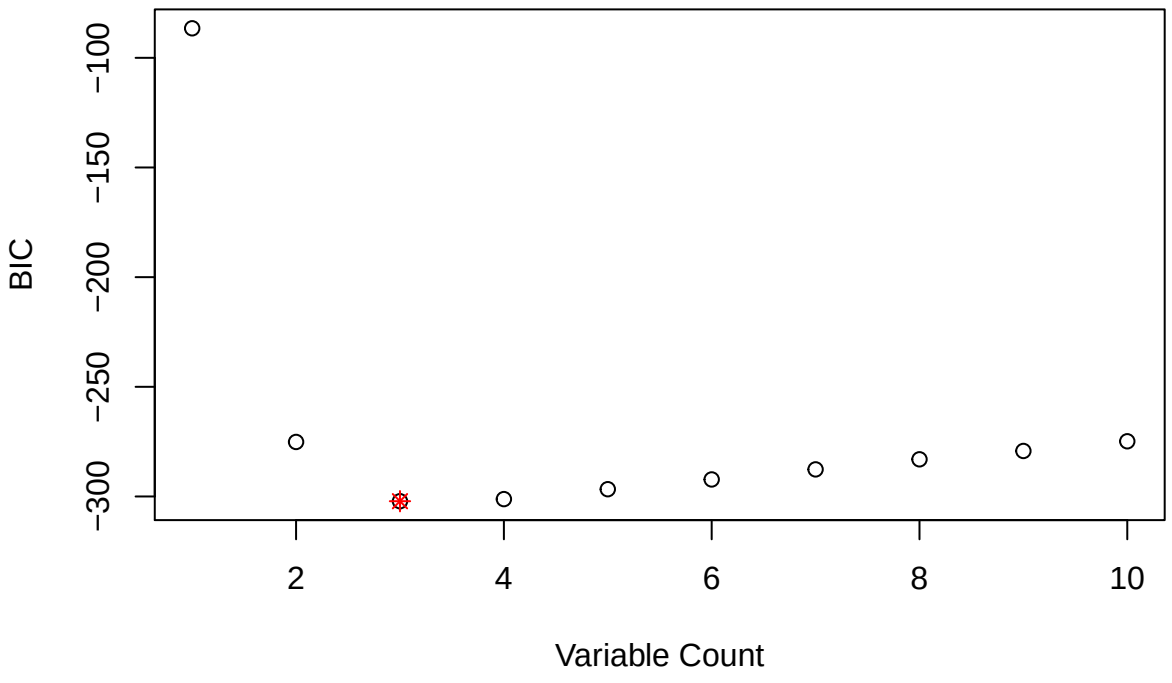
(d) Repeat (c), using forward stepwise selection and also using backwards stepwise selection. How does your answer compare to the results in (c)? You must show a summary of the selected model or other evidence to support your statements.

```
forward = 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(X^10)
```

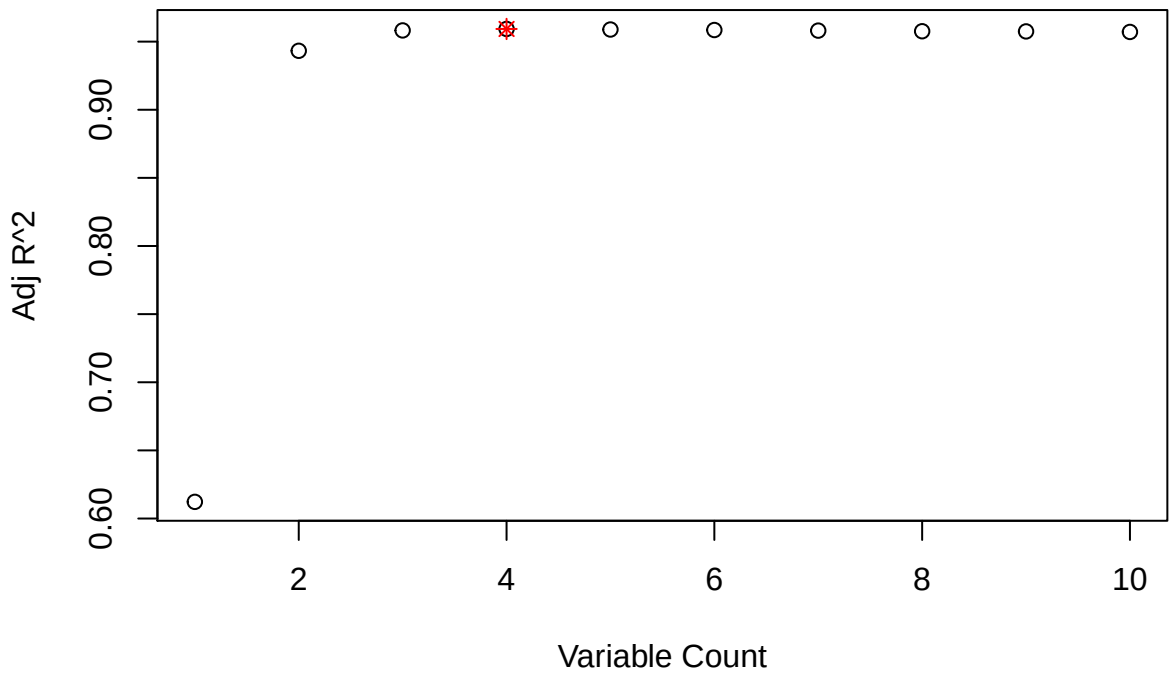
```
                          , data = full, nvmax=10, method = "forward")
forward.summary = summary(forward)
plot(forward.summary$cp, xlab = "Variable Count", ylab = "Cp")
points(which.min(forward.summary$cp), forward.summary$cp[which.min(forward.summary$cp)],
       col = "red", cex = 1, pch = 8)
```



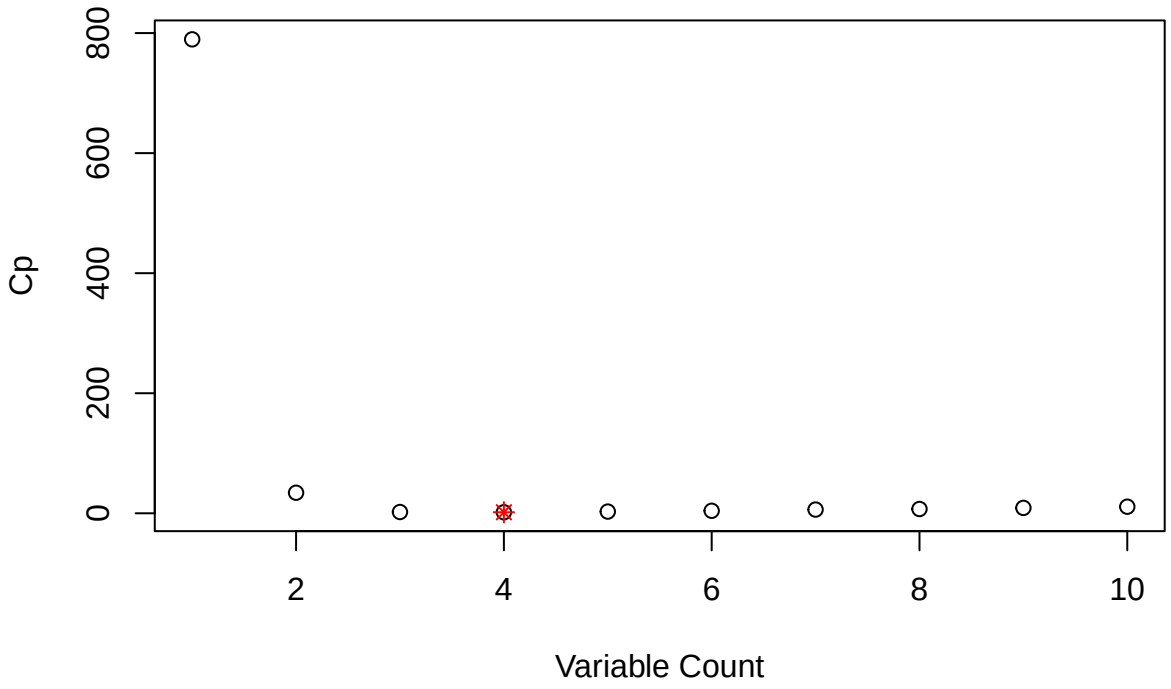

```
plot(forward.summary$bic, xlab = "Variable Count", ylab = "BIC")
points(which.min(forward.summary$bic), forward.summary$bic[which.min(forward.summary$bic)],
       col = "red", cex = 1, pch = 8)
```



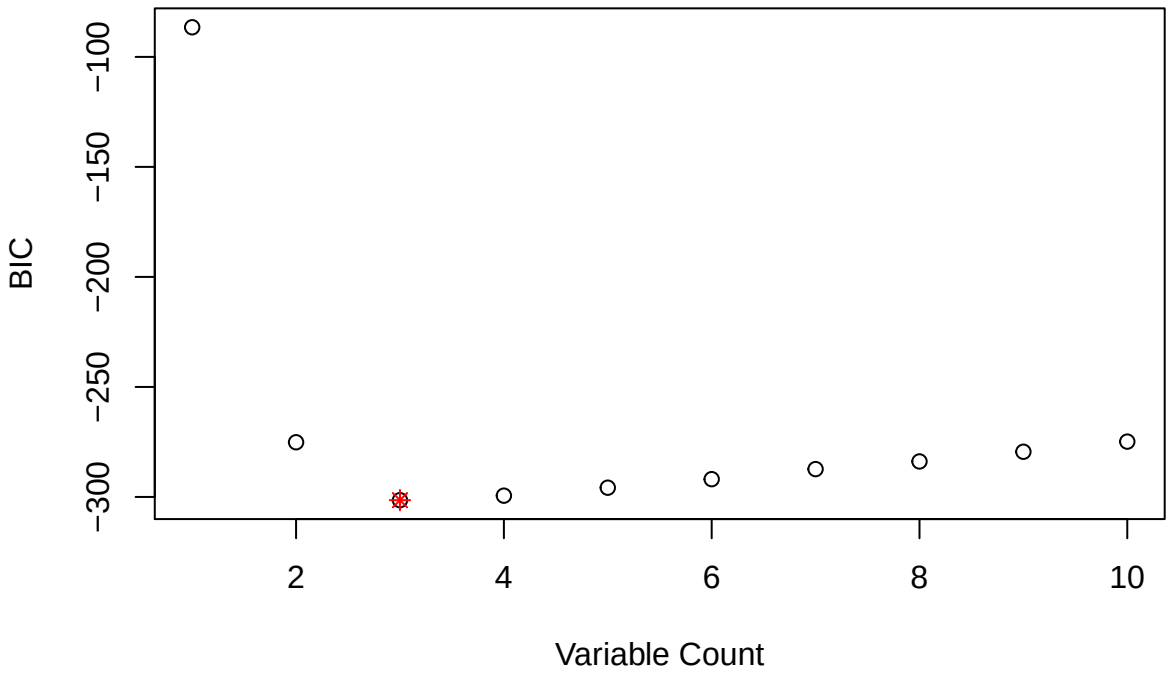
```
plot(forward.summary$adjr2, xlab = "Variable Count", ylab = "Adj R^2")
points(which.max(forward.summary$adjr2), forward.summary$adjr2[which.max(forward.summary$adjr2)],
       col = "red", cex = 1, pch = 8)
```



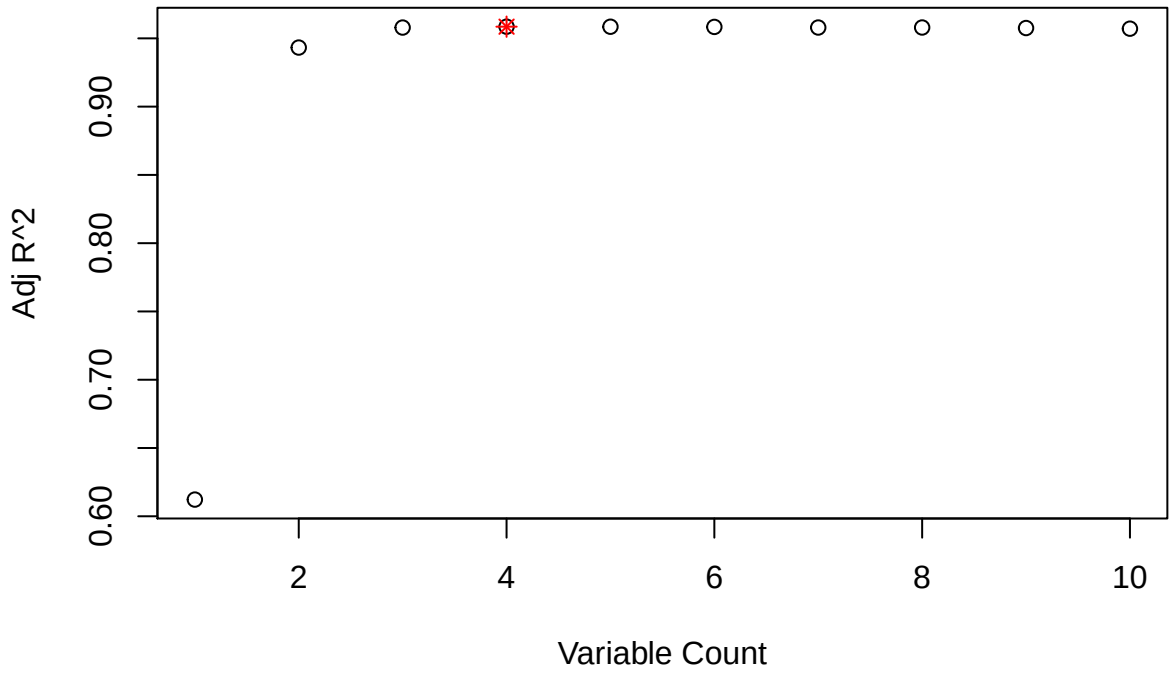

```
backward = regsubsets(Y ~ X + I(X^2) + I(X^3) + I(X^4) + I(X^5)
                          + I(X^6) + I(X^7) + I(X^8)+ I(X^9)+ I(X^10)
                          , data = full, nvmax=10, method = "backward")
backward.summary = summary(backward)
plot(backward.summary$cp, xlab = "Variable Count", ylab = "Cp")
points(which.min(backward.summary$cp), backward.summary$cp[which.min(backward.summary$cp)],
       col = "red", cex = 1, pch = 8)
```




```
plot(backward.summary$bic, xlab = "Variable Count", ylab = "BIC")
points(which.min(backward.summary$bic), backward.summary$bic[which.min(backward.summary$bic)],
       col = "red", cex = 1, pch = 8)
```




```
plot(backward.summary$adjr2, xlab = "Variable Count", ylab = "Adj R^2")
points(which.max(backward.summary$adjr2), backward.summary$adjr2[which.max(backward.summary$adjr2)],
       col = "red", cex = 1, pch = 8)
```




```
summary(sub)
```

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

```
## 1  ( 1 )  " "
## 2  ( 1 )  " "
## 3  ( 1 )  " "
## 4  ( 1 )  "*"
## 5  ( 1 )  " "
## 6  ( 1 )  " "
## 7  ( 1 )  " "
## 8  ( 1 )  "*"
## 9  ( 1 )  "*"
## 10  ( 1 ) "*"
```

```
summary(forward)
```

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

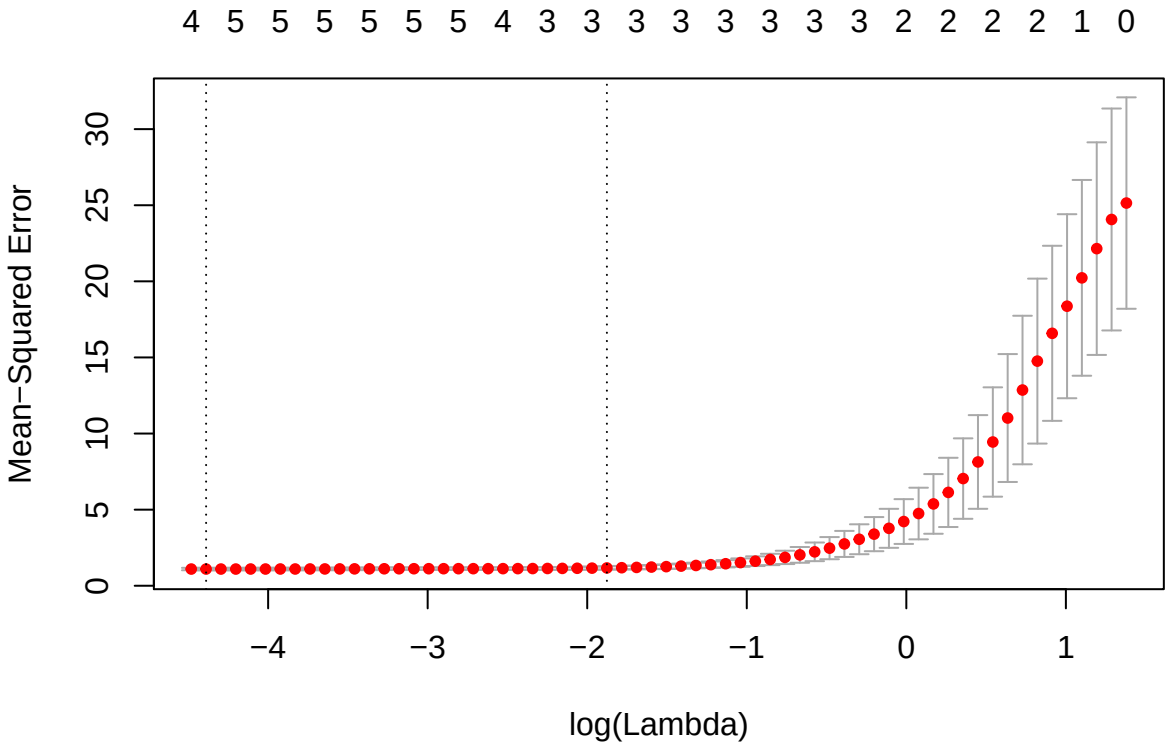
```
summary(backward)
```

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

Backwards subset regression wants to select X, X^2, X^5, and X^8 for the 4 variable model and drops X^8

(e) Now fit a LASSO model with `glmnet` function from `glmnet` package to the simulated data, again using $(X, X^2, \cdots, X^{10})$ as predictors. Use 5-fold 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.

```
x = 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)+ I(X^
lasso = cv.glmnet(x, Y, nfolds = 5)
plot(lasso)
```




```
lambda = lasso$lambda.min
lasso_fit = glmnet(x, Y)
predict(lasso_fit, s=lambda, type = "coefficients")[1:12,]
```

```
##   (Intercept)   (Intercept)             X        I(X^2)        I(X^3)
##  3.081187e+00  0.000000e+00  1.919192e+00 -3.100745e+00  2.704661e-01
##        I(X^4)        I(X^5)        I(X^6)        I(X^7)        I(X^8)
##  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
##        I(X^9)       I(X^10)
##  2.842655e-07  1.657782e-04
```

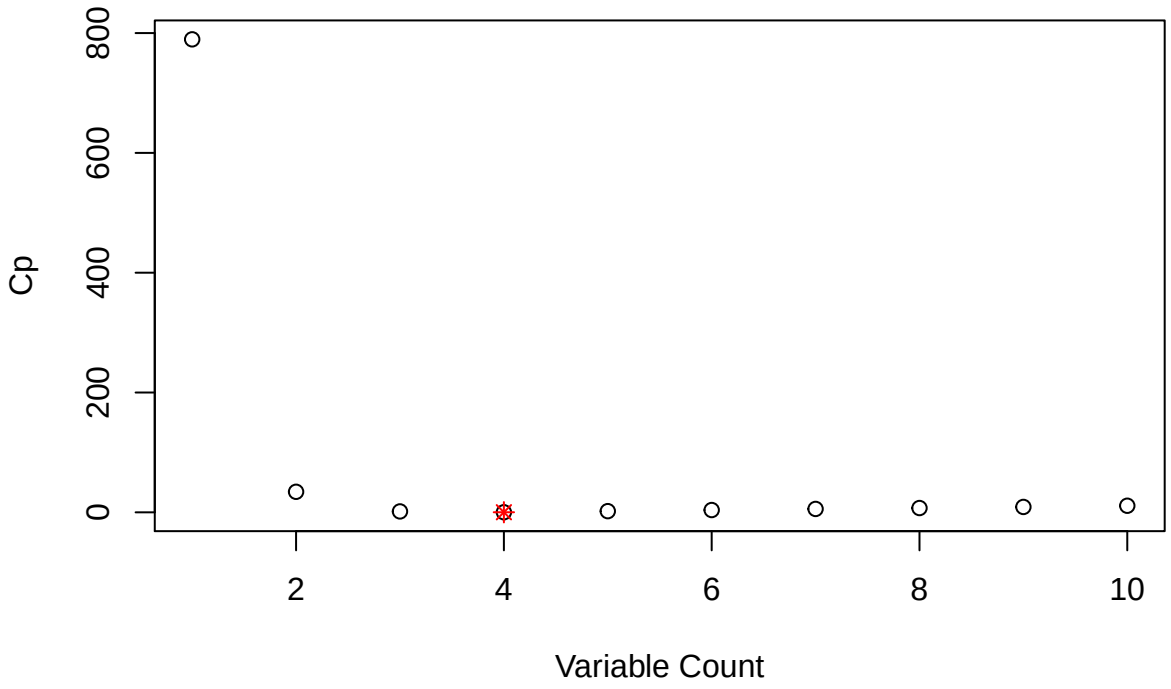
Lasso chose X, X^1, X^2, X^3 and X^9 as the significant predictors.

(f) Now generate a response vector Y according to the model

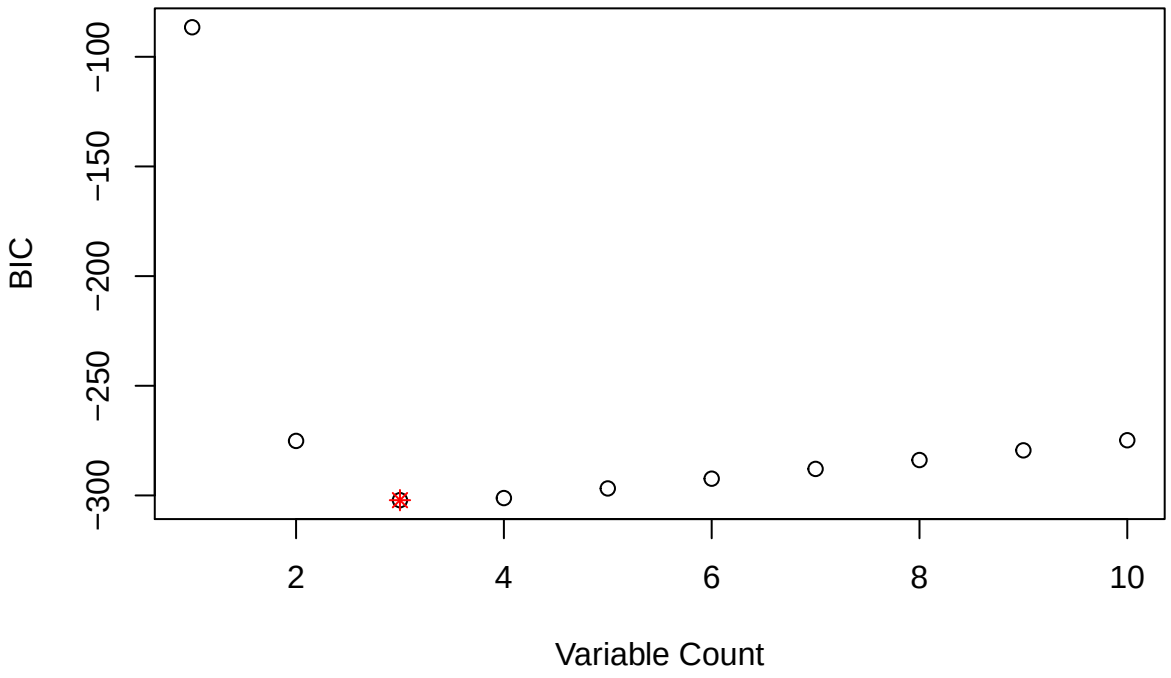
$$Y = \beta_0 + \beta_7 X^7 + \epsilon,$$

where $\beta_7 = 7$, and perform best subset selection and the LASSO. Discuss the results obtained.

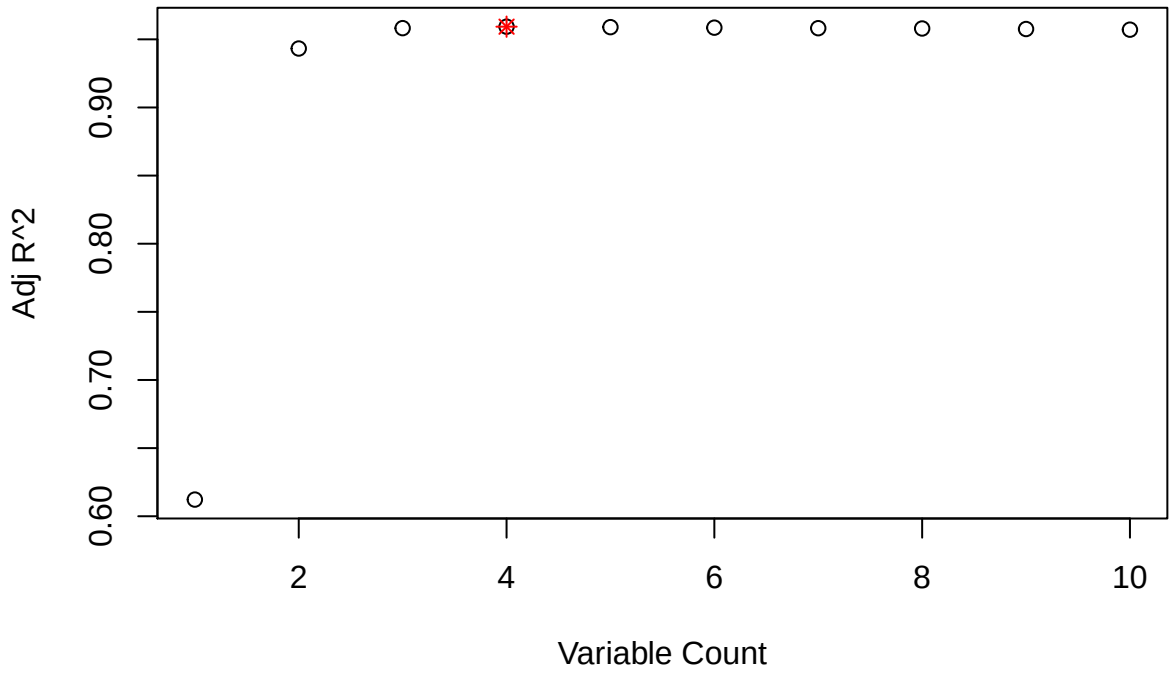
```
b7 = 7
Y = b0 + b7*(X^7)+epsilon

sub = 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(X^10)
                          , data = full, nvmax=10)
sub.summary = summary(sub)
plot(sub.summary$cp, xlab = "Variable Count", ylab = "Cp")
points(which.min(sub.summary$cp), sub.summary$cp[which.min(sub.summary$cp)],
       col = "red", cex = 1, pch = 8)
```



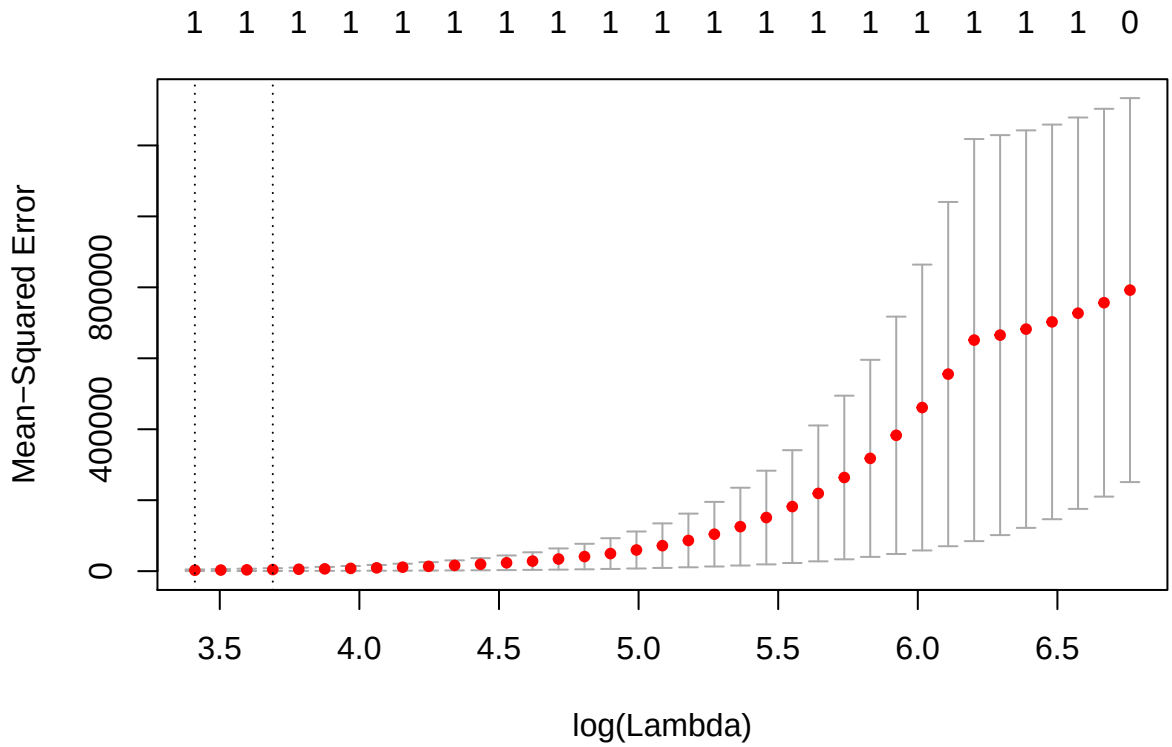

```
plot(sub.summary$bic, xlab = "Variable Count", ylab = "BIC")
points(which.min(sub.summary$bic), sub.summary$bic[which.min(sub.summary$bic)],
       col = "red", cex = 1, pch = 8)
```




```
plot(sub.summary$adjr2, xlab = "Variable Count", ylab = "Adj R^2")
points(which.max(sub.summary$adjr2), sub.summary$adjr2[which.max(sub.summary$adjr2)],
       col = "red", cex = 1, pch = 8)
```




```
    x = 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)+ I(X^
    lasso = cv.glmnet(x, Y, nfolds = 5)
    plot(lasso)
```




```
    lambda = lasso$lambda.min
    lasso_fit = glmnet(x, Y)
    predict(lasso_fit, s=lambda, type = "coefficients")[1:12,]
```

```
## (Intercept) (Intercept)           X      I(X^2)      I(X^3)      I(X^4)
```

```
##    5.822681    0.000000    0.000000    0.000000    0.000000    0.000000
##      I(X^5)      I(X^6)      I(X^7)      I(X^8)      I(X^9)     I(X^10)
##    0.000000    0.000000    6.753863    0.000000    0.000000    0.000000
```

---

2. (Prediction, *20 pt*) In this exercise, we will try to develop a prediction model for wins in a basketball season for a team based on a host of other factors. The starting point is to load the nba-teams-2017 data set (which was scraped by Gaston Sanchez at Berkeley).

(a) Do some exploratory data analysis by picking 6-7 features that you think might be interesting and explore relationship between these features by making a scatterplot matrix like the following (you **do not** have to use the same features I am using!):

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

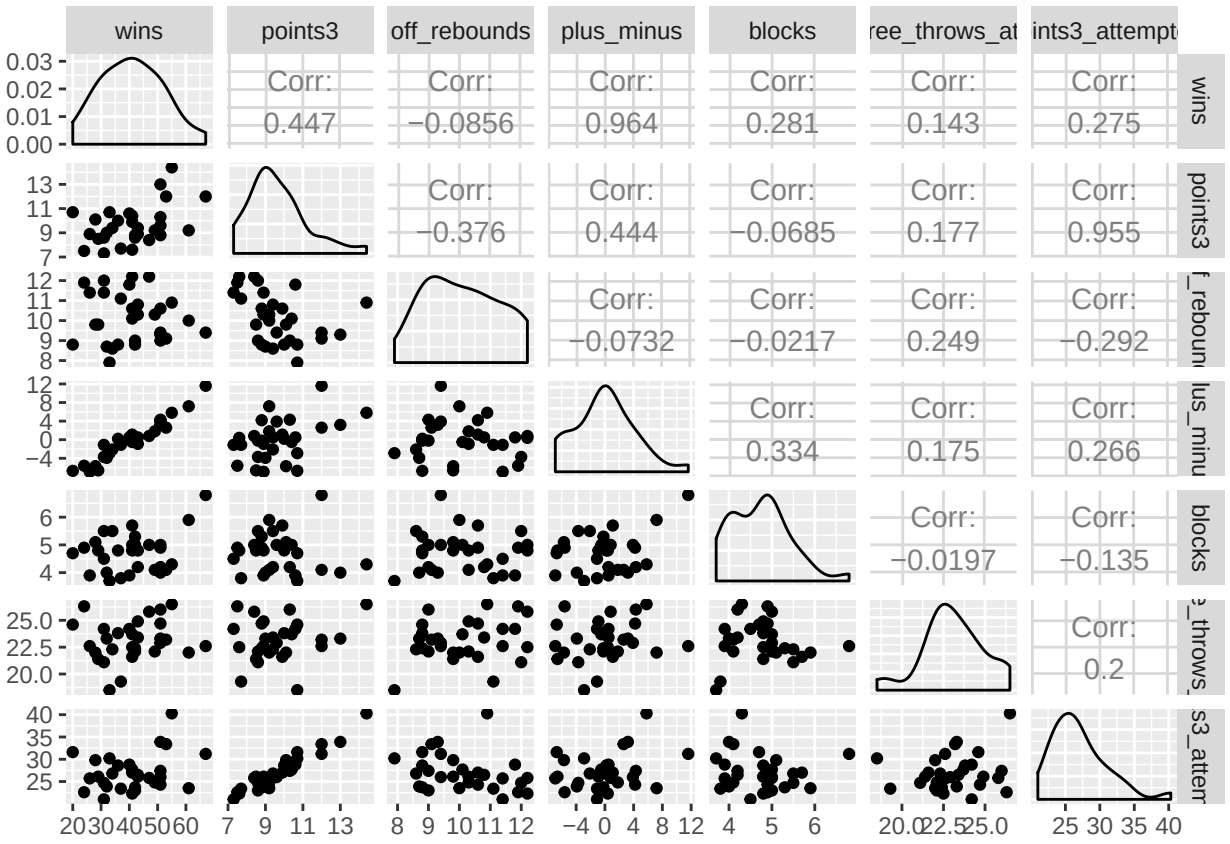
```
library(ggplot2)
library(GGally)
```

```
##
## Attaching package: 'GGally'
```

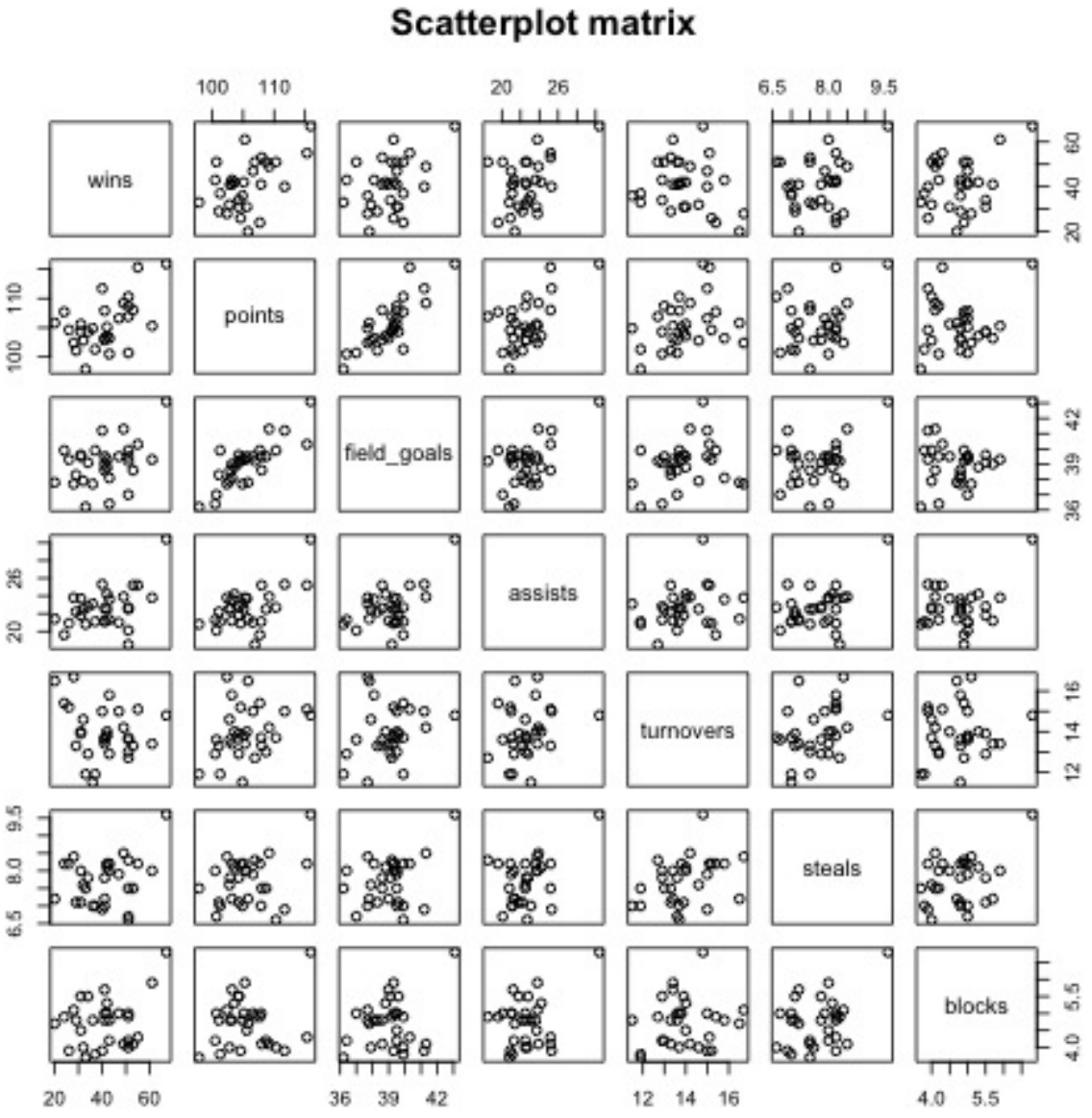
```
## The following object is masked from 'package:dplyr':
##
##     nasa
```

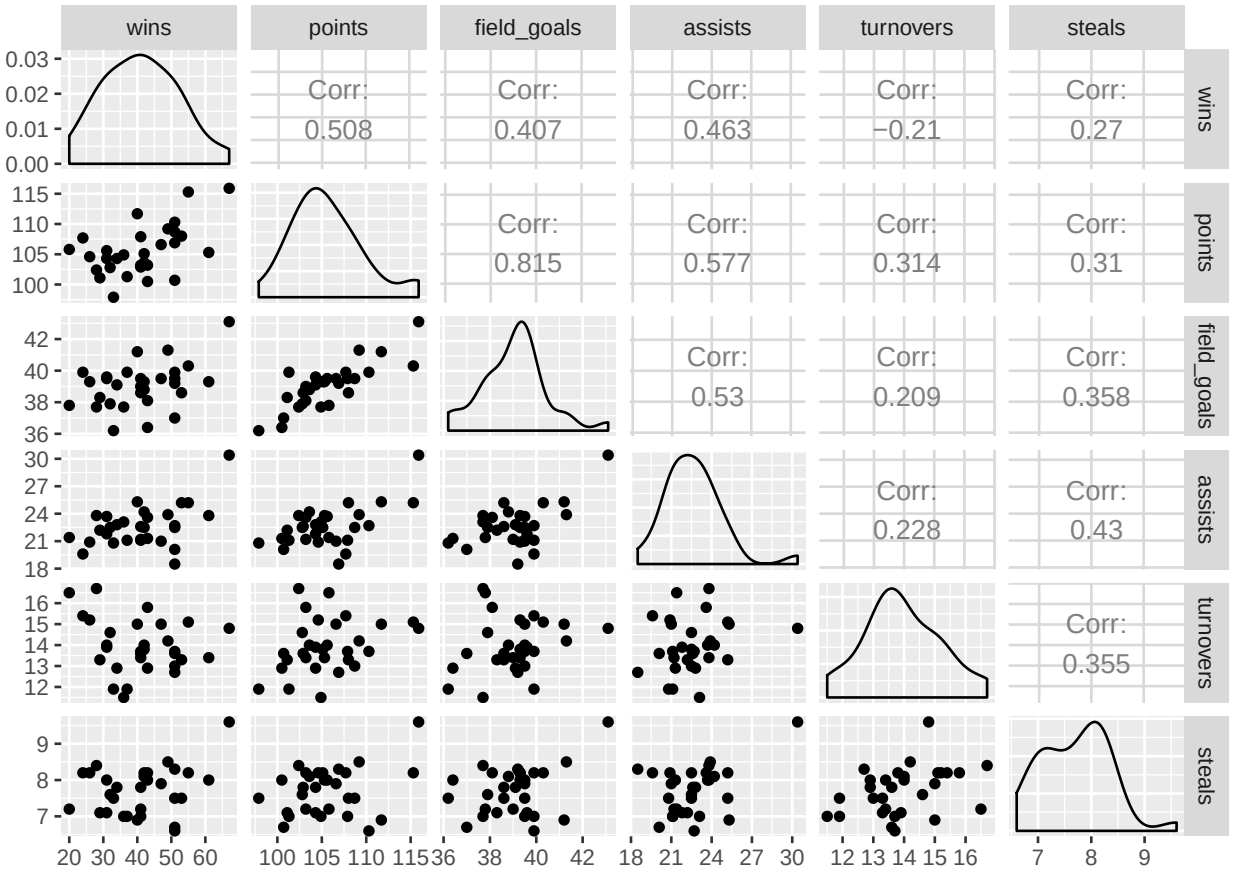
```
nba = read.csv("nba-teams-2017.csv")
nba1 = nba %>% select(wins, points3, off_rebounds, plus_minus, blocks, free_throws_att, points3_attempt
ggpairs(nba1[,1:7])
```

```
## Warning: `new_list_along()` is deprecated as of rlang 0.3.0.
## Please use `rep_along(x, list(NULL))` or `rep_named(nms, list(NULL))` instead.
## This warning is displayed once per session.
```



NOTE: You may remove the includegraphics statements below when knitting your own response, if they are giving you trouble


Scatterplot matrix
wins
points
field_goals
assists
turnovers
steals
blocks



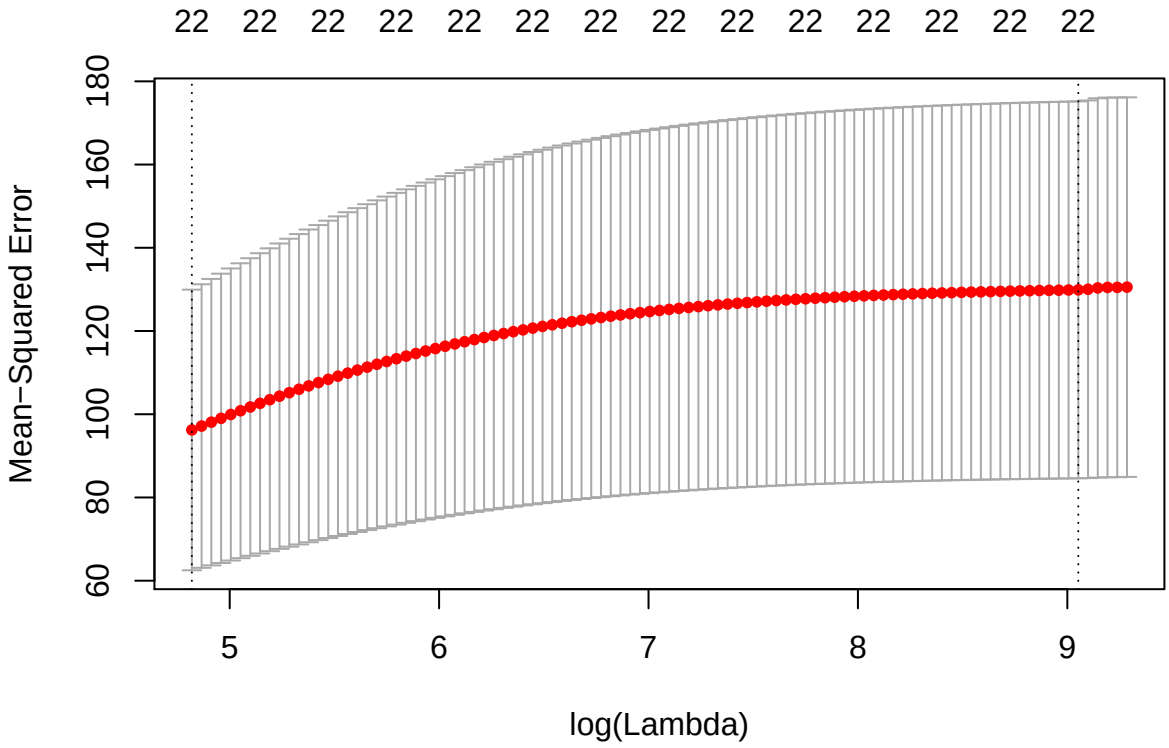
(b) The aim is now to predict *wins* based on the other features. First explain why you would remove the "losses" column from the above data set? Would you necessarily remove any other columns?

```
Losses needs to be removed because it is the opposite of a win. There is a -1 correlation between the tw
```

```
nba = select(nba, -c(losses, win_prop, team))
```

(c) Use ridge regression with 5 fold cross-validation to choose the optimal tuning parameter and report your model along with your test error as found by cross-validation for that choice of λ.

```
set.seed(83)
train <- sample(1:nrow(nba), nrow(nba) / 2)
test <- (-train)
nba.train <- nba[train,]
nba.test <- nba[test,]

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


train.mat <- model.matrix(wins ~ ., data = nba.train)
test.mat <- model.matrix(wins ~ ., data = nba.test)

ridge.model.train <- glmnet(train.mat, nba.train$wins, alpha = 0, lambda = grid.lambda)

cv.out <- cv.glmnet(train.mat, nba.train$wins, alpha = 0, nfolds = 5)
plot(cv.out)
```




```
bestlam.ridge <- cv.out$lambda.min
bestlam.ridge
```

```
## [1] 123.8767
```

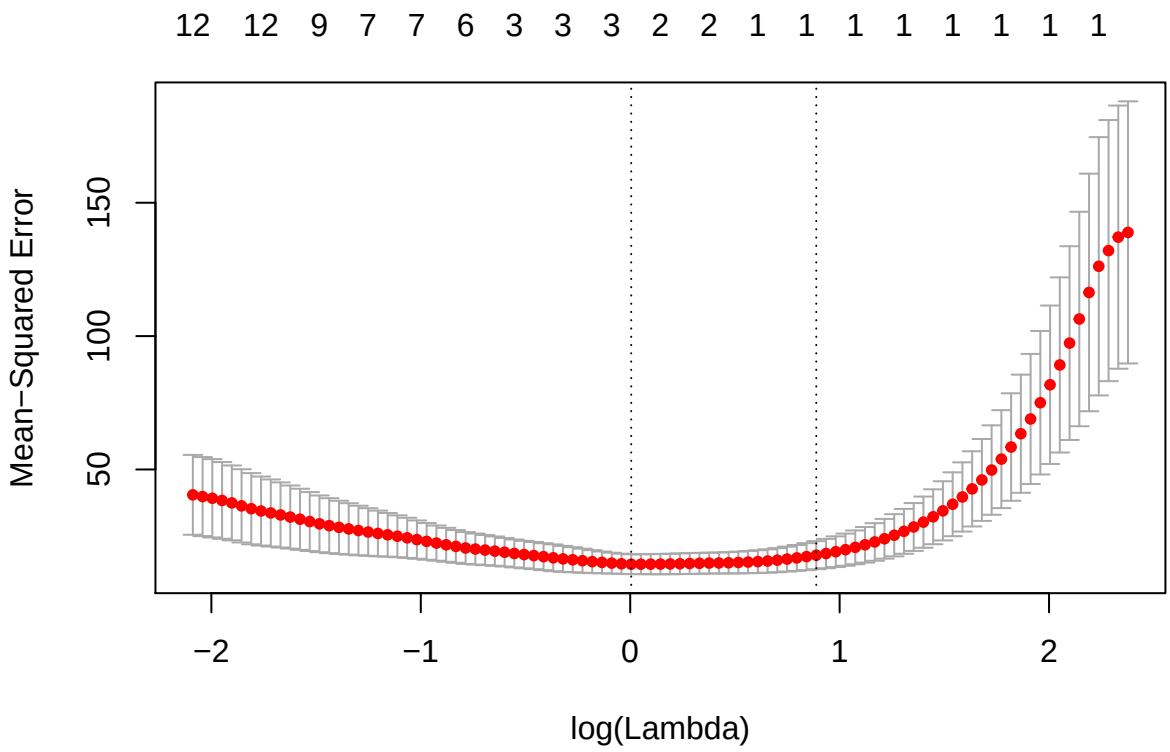
```
pred.ridge <- predict(ridge.model.train, s = bestlam.ridge, newx = test.mat)
mean((pred.ridge - nba.test$wins)^2)
```

```
## [1] 81.99093
```

```
The MSPE for the data was 81.99 using 5 fold ridge regression.
```

(d) Fit a LASSO model on the training set, with λ chosen by 5-fold cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```
lasso.model.train <- glmnet(train.mat, nba.train$wins, alpha = 1, lambda = grid.lambda)

cv.out <- cv.glmnet(train.mat, nba.train$wins, alpha = 1, nfolds = 5)
plot(cv.out)
```




```
bestlam.lasso <- cv.out$lambda.min
bestlam.lasso
```

```
## [1] 1.004802
```

```
pred.lasso <- predict(lasso.model.train, s = bestlam.lasso, newx = test.mat)
mean((pred.lasso - nba.test$wins)^2)
```

```
## [1] 13.29133
```

```
predict(lasso.model.train, s = bestlam.lasso, type = "coefficients")
```

```
## 25 x 1 sparse Matrix of class "dgCMatrix"
##                                    1
## (Intercept)           65.19925859
## (Intercept)            .
## games_played           .
## minutes                .
## points                 .
## field_goals            .
## field_goals_attempted  .
## field_goals_prop       .
## points3                .
## points3_attempted      .
## points3_prop           .
## free_throws            .
## free_throws_att        .
## free_throws_prop      -0.01997032
## off_rebounds           .
## def_rebounds          -0.66216021
## rebounds               .
## assists                .
```

```
## turnovers              .
## steals                 .
## blocks                 .
## block_fga              .
## personal_fouls         .
## personal_fouls_drawn   .
## plus_minus             2.21768908
```

```
The MSPE for lasso was 13.29. This is substaintally less than ridge. As seen, lasso used 3 variables plu
```

---

3. Let us now try to understand the performance of the various techniques on simulated data to get insight.

Suppose your true model is $\mathbf{y} = \mathbf{X}\beta + \epsilon$

where:

- $\beta = (\underbrace{1, ..., 1}_{20}, \underbrace{0, ..., 0}_{1980})^T$
- $p = 2000 > n = 1000$
- Uncorrelated predictors:
  - $\mathbf{X}_i \overset{\text{iid}}{\sim} N(\mathbf{0}, \mathbf{I})$. Precisely for the i-th individual, $\mathbf{X}_i = (X_{i1}, X_{i2}, \ldots, X_{i,2000})$ where $X_{i,j}$ are independent and identically distributed normal random variables with mean zero and variance one.
- $\epsilon \overset{\text{iid}}{\sim} N(\mathbf{0}, \mathbf{I})$. Precisely: $\epsilon = (\epsilon_1, \epsilon_2, \ldots, \epsilon_{1000})$ where ϵ_i are independent and identically distributed normal random variables with mean zero and variance 1.

(a)(2 pt) Generate the above data with seed = 1234

```
set.seed(1234)
X = matrix(rnorm(1000*2000), 1000, 2000)
train = sample(1:nrow(X), nrow(X) / 1)

beta = c(rep(1,20), rep(0, 1980))
epsilon = rnorm(1000)

Y = X*beta + epsilon

X.train = X[train,]
Y.train = Y[train]


train.mat = model.matrix(Y.train ~ X.train)
```

(b) (6 pt) Using glmnet fit Lasso, ridge regression and elastic net with $\alpha = .1, .2, .3, .4, .5, .6, .7, .8, .9$

```
# YOUR CODE HERE (uncomment first of course)
```

What I am looking for: (outputting the entire model for each one of the above is not-trivial so):

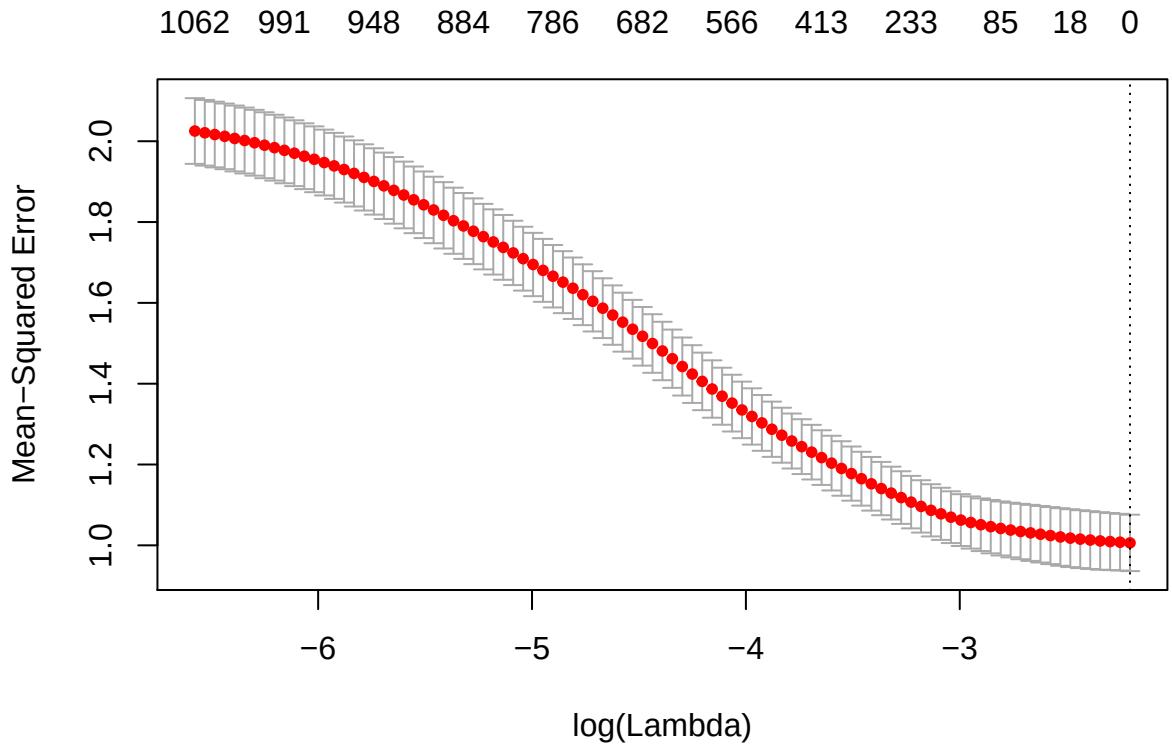
- code showing the fitting of each of the above models

```
lasso.model.train <- glmnet(train.mat, Y.train, alpha = 1, lambda = grid.lambda)
ridge.model.train <- glmnet(train.mat, Y.train, alpha = 0, lambda = grid.lambda)
en.model.train1<- glmnet(train.mat, Y.train, alpha = .1, lambda = grid.lambda)
en.model.train2<- glmnet(train.mat, Y.train, alpha = .2, lambda = grid.lambda)
en.model.train3<- glmnet(train.mat, Y.train, alpha = .3, lambda = grid.lambda)
en.model.train4<- glmnet(train.mat, Y.train, alpha = .4, lambda = grid.lambda)
en.model.train5<- glmnet(train.mat, Y.train, alpha = .5, lambda = grid.lambda)
```

```
en.model.train6<- glmnet(train.mat, Y.train, alpha = .6, lambda = grid.lambda)
en.model.train7<- glmnet(train.mat, Y.train, alpha = .7, lambda = grid.lambda)
en.model.train8<- glmnet(train.mat, Y.train, alpha = .8, lambda = grid.lambda)
en.model.train9<- glmnet(train.mat, Y.train, alpha = .9, lambda = grid.lambda)
```

- For ridge, Lasso and for $\alpha = .2, , .4, .6$ plot the cross-validated (6 fold) MSE versus lambda as well as your optimal value of *lambda* for ridge, Lasso and for $\alpha = .2, , .4, .6$

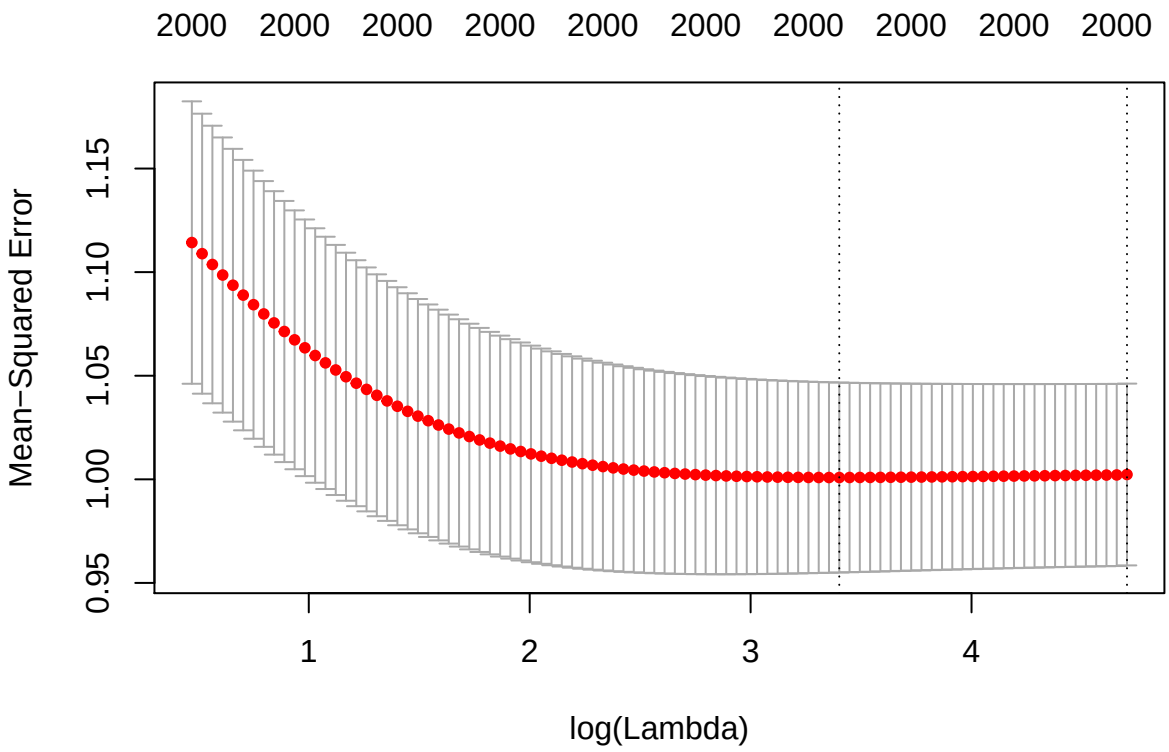
```
cv.out.lasso <- cv.glmnet(train.mat, Y.train, alpha = 1, nfolds = 6)
plot(cv.out.lasso)
```




```
bestlam.lasso <- cv.out$lambda.min
bestlam.lasso
```

```
## [1] 1.004802
```

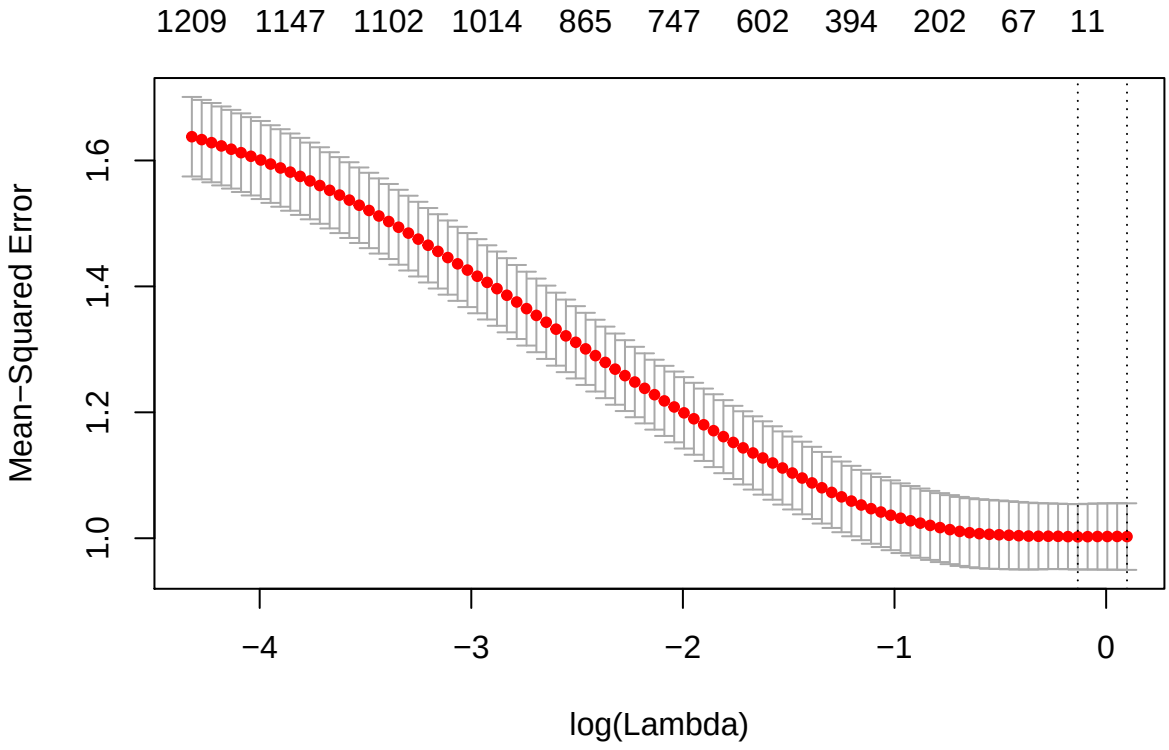
```
cv.out.ridge <- cv.glmnet(train.mat, Y.train, alpha = 0, nfolds = 6)
plot(cv.out.ridge)
```




```
bestlam.ridge <- cv.out$lambda.min
bestlam.ridge
```

```
## [1] 1.004802
```

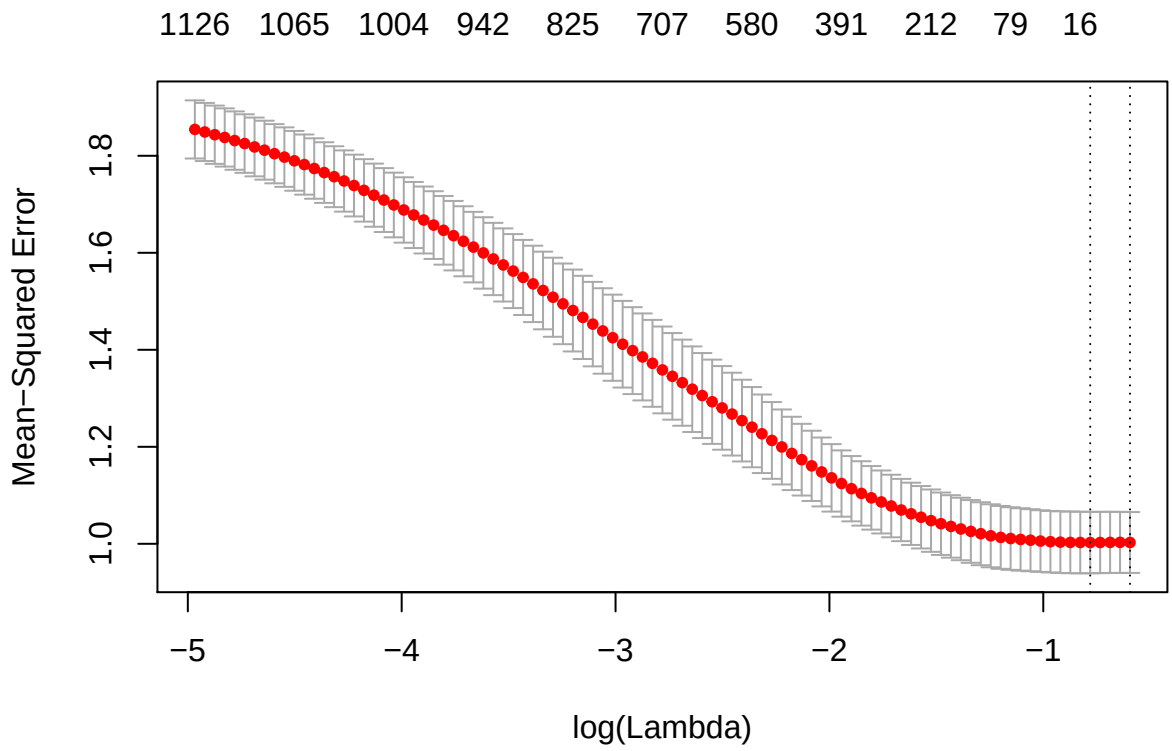
```
cv.out.en1 <- cv.glmnet(train.mat, Y.train, alpha = .1, nfolds = 6)
plot(cv.out.en1)
```



```
bestlam.en1 <- cv.out$lambda.min
bestlam.en1
```

```
## [1] 1.004802
```

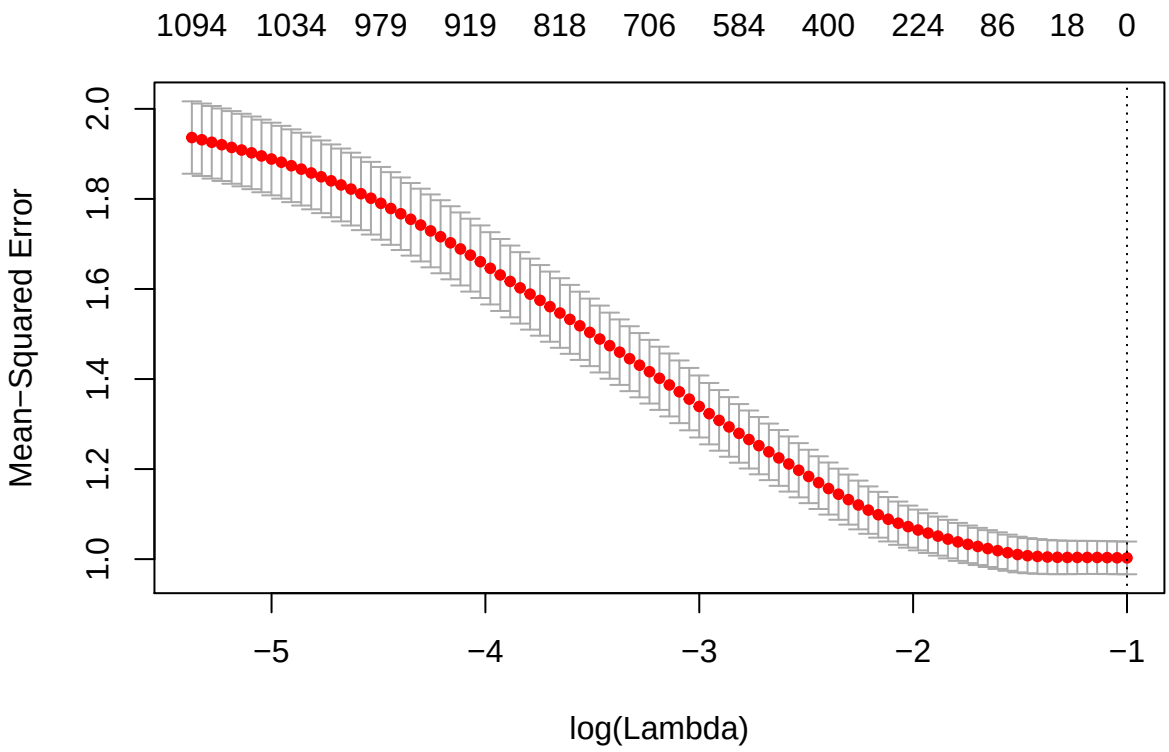
```
cv.out.en2 <- cv.glmnet(train.mat, Y.train, alpha = .2, nfolds = 6)
plot(cv.out.en2)
```




```
bestlam.en2 <- cv.out$lambda.min
bestlam.en2
```

```
## [1] 1.004802
```

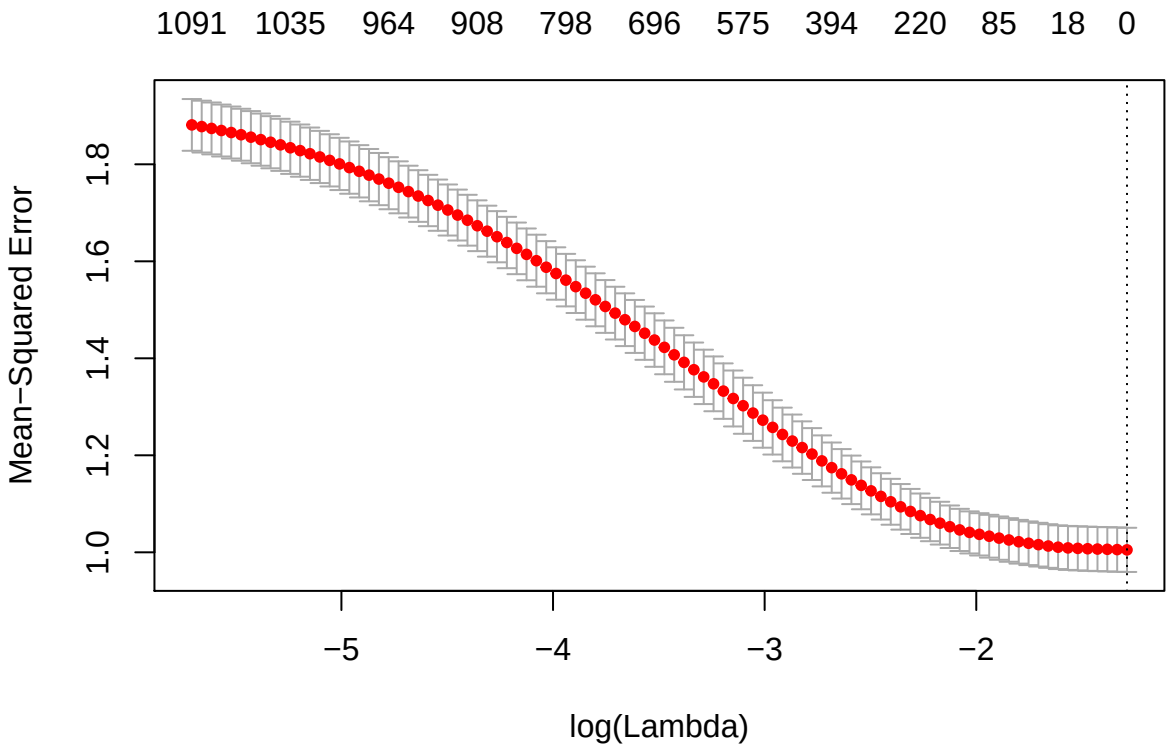
```
cv.out.en3 <- cv.glmnet(train.mat, Y.train, alpha = .3, nfolds = 6)
plot(cv.out.en3)
```




```
bestlam.en3 <- cv.out$lambda.min
bestlam.en3
```

```
## [1] 1.004802
```

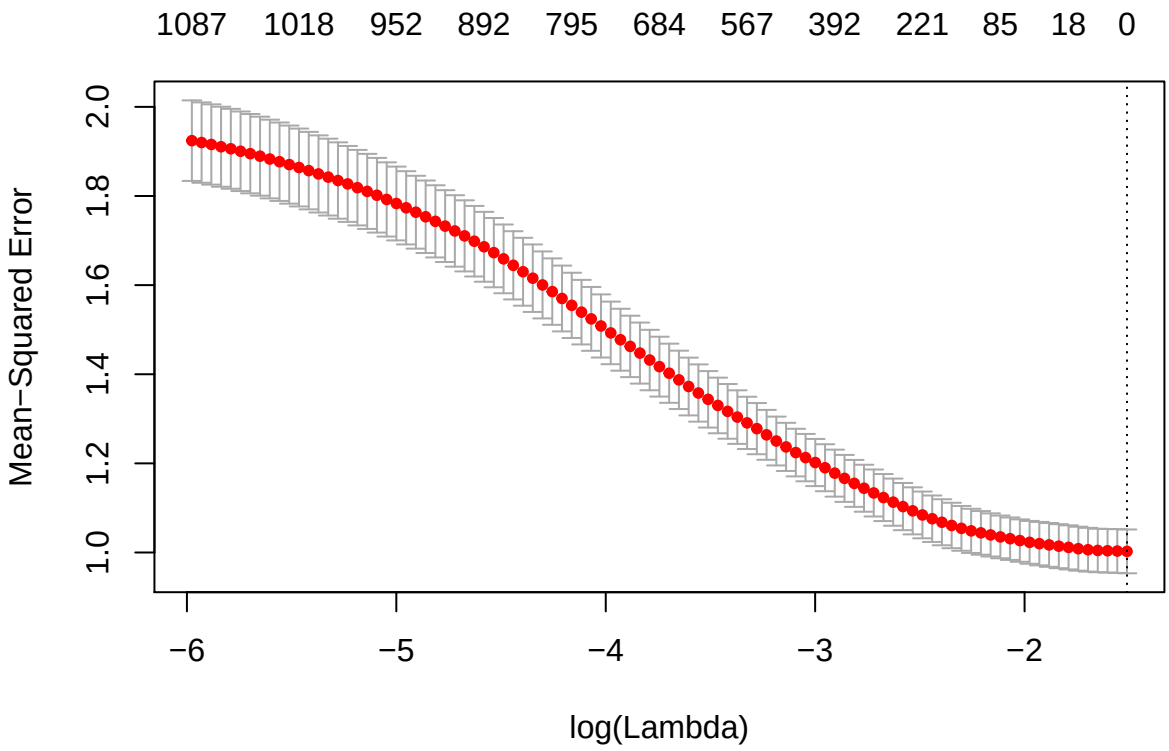
```
cv.out.en4 <- cv.glmnet(train.mat, Y.train, alpha = .4, nfolds = 6)
plot(cv.out.en4)
```



```
bestlam.en4 <- cv.out$lambda.min
bestlam.en4
```

```
## [1] 1.004802
```

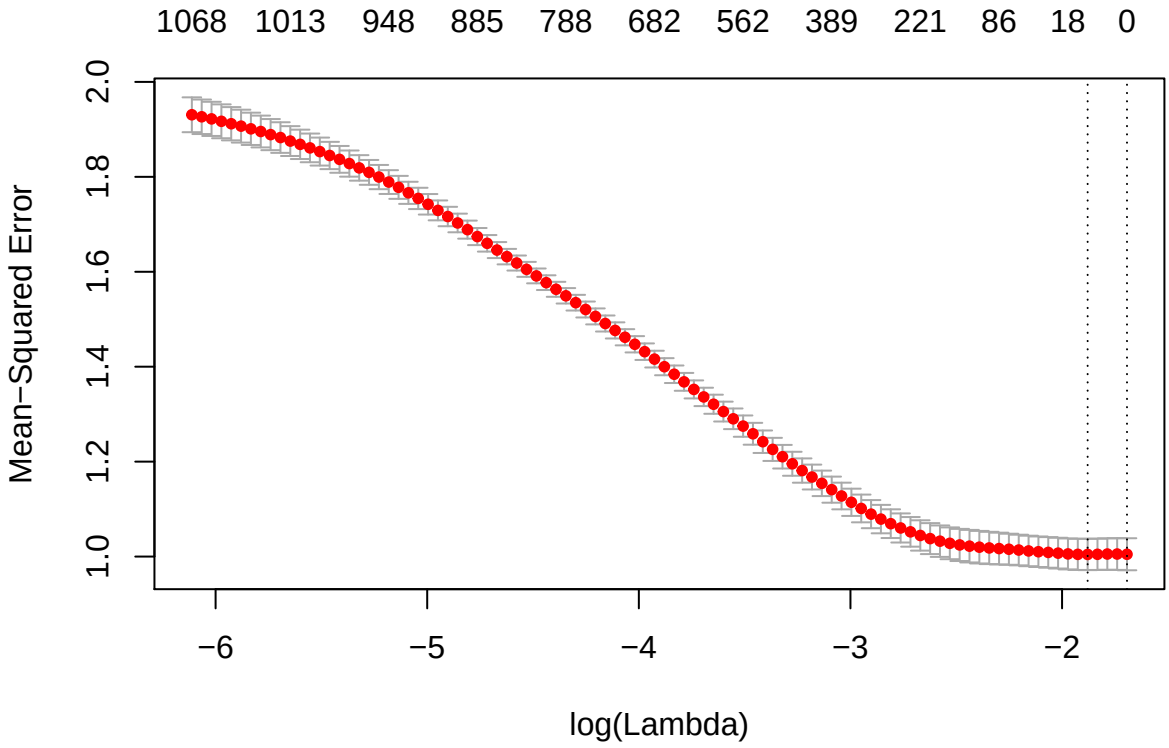
```
cv.out.en5 <- cv.glmnet(train.mat, Y.train, alpha = .5, nfolds = 6)
plot(cv.out.en5)
```




```
bestlam.en5 <- cv.out$lambda.min
bestlam.en5
```

```
## [1] 1.004802
```

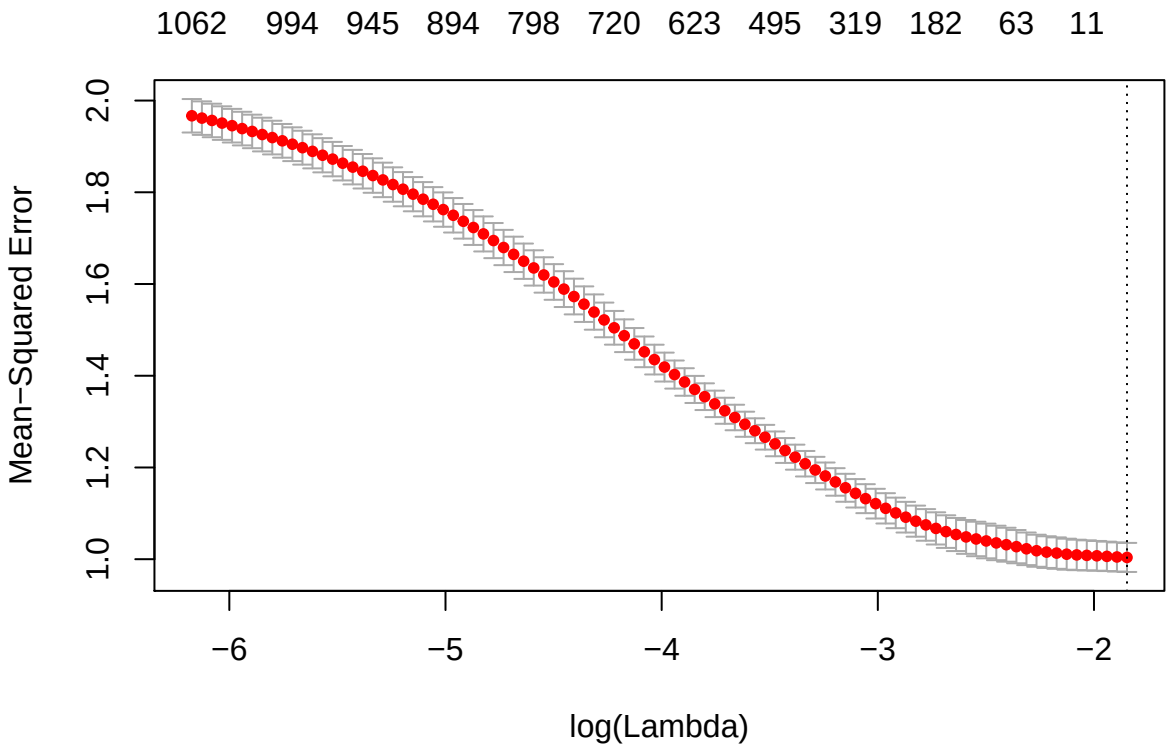
```
cv.out.en6 <- cv.glmnet(train.mat, Y.train, alpha = .6, nfolds = 6)
plot(cv.out.en6)
```




```
bestlam.en6 <- cv.out$lambda.min
bestlam.en6
```

```
## [1] 1.004802
```

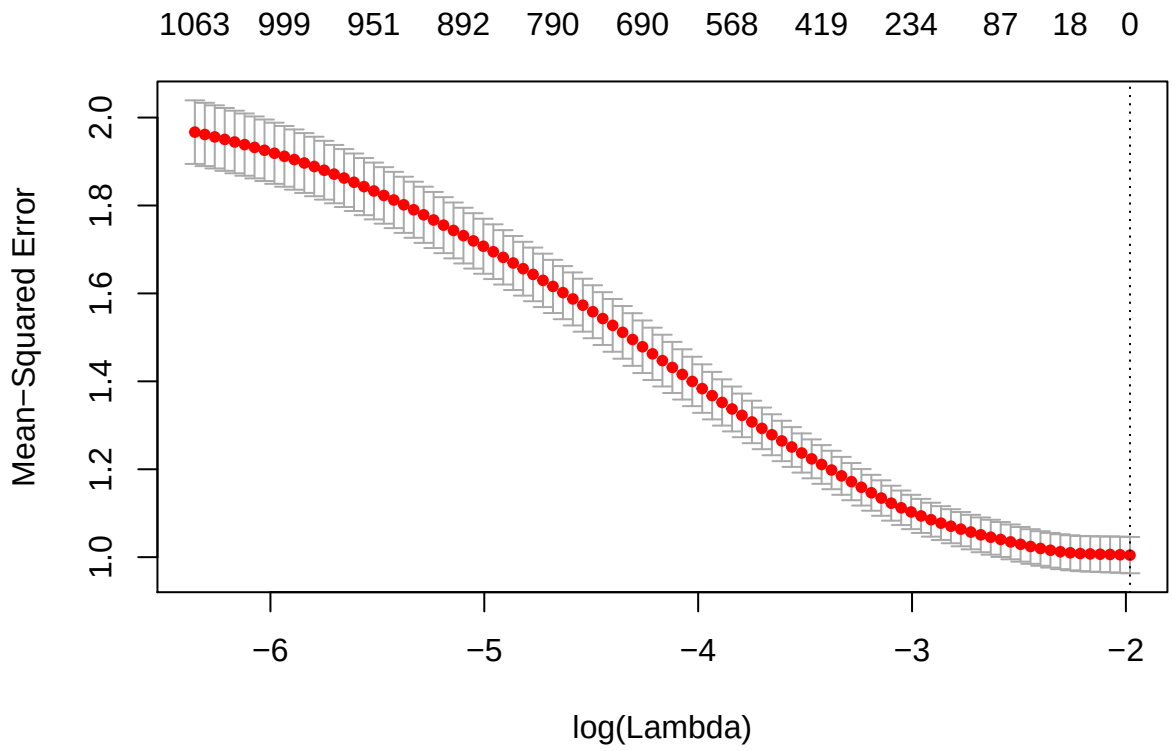
```
cv.out.en7 <- cv.glmnet(train.mat, Y.train, alpha = .7, nfolds = 6)
plot(cv.out.en7)
```



```
bestlam.en7 <- cv.out$lambda.min
bestlam.en7
```

```
## [1] 1.004802
```

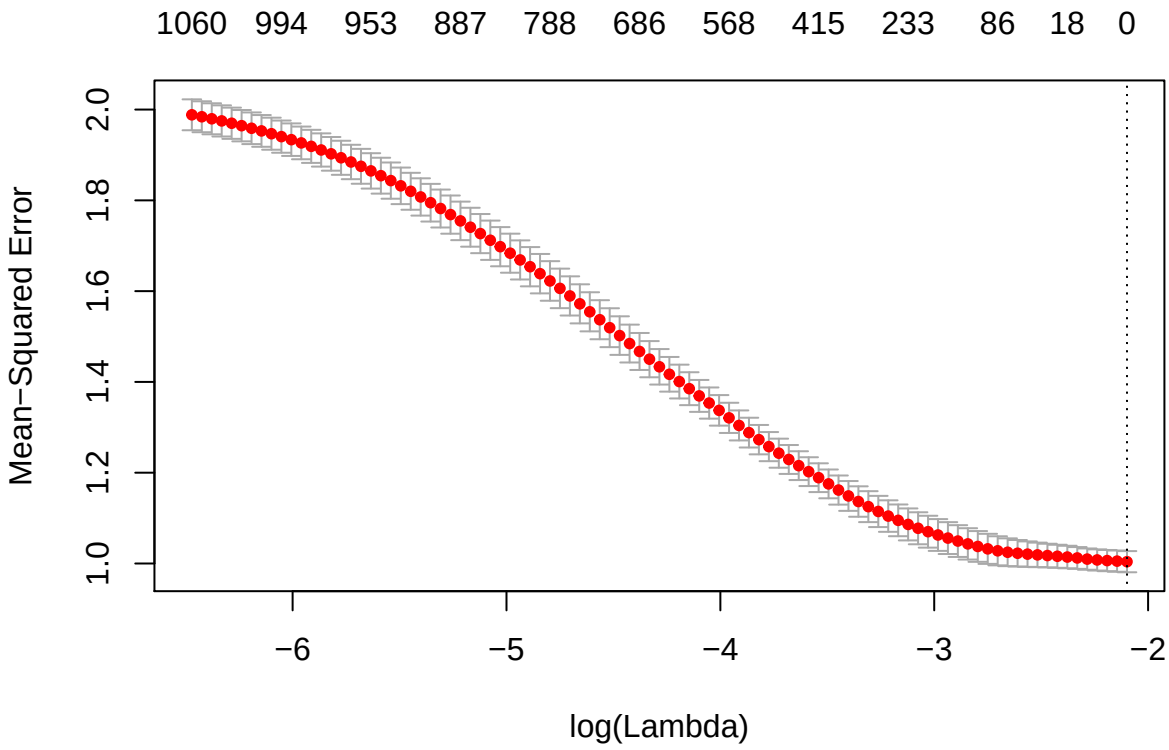
```
cv.out.en8 <- cv.glmnet(train.mat, Y.train, alpha = .8, nfolds = 6)
plot(cv.out.en8)
```




```
bestlam.en8 <- cv.out$lambda.min
bestlam.en8
```

```
## [1] 1.004802
```

```
cv.out.en9 <- cv.glmnet(train.mat, Y.train, alpha = .9, nfolds = 6)
plot(cv.out.en9)
```




```
bestlam.en9 <- cv.out$lambda.min
bestlam.en9
```

```
## [1] 1.004802
```

- The number of non-zero regression coeffecients for each of the above techniques.

```
library(coefplot)
```

```
##
## Attaching package: 'coefplot'
```

```
## The following object is masked from 'package:pls':
##
##     coefplot
```

```
extract.coef(cv.out.lasso)
```

```
##                    Value SE Coefficient
## X.Intercept. 0.06374977 NA (Intercept)
```

```
extract.coef(cv.out.en1)
```

```
##                       Value SE Coefficient
## X.Intercept.  0.0636061158 NA (Intercept)
## X.train95    -0.0016869353 NA   X.train95
## X.train211    0.0001662434 NA  X.train211
## X.train326   -0.0013419652 NA  X.train326
## X.train352    0.0057173180 NA  X.train352
## X.train512   -0.0057926424 NA  X.train512
## X.train678    0.0086975107 NA  X.train678
## X.train719    0.0134528055 NA  X.train719
## X.train988   -0.0041465791 NA  X.train988
## X.train1091  -0.0046208118 NA X.train1091
```

```
## X.train1339  -0.0011203226 NA X.train1339
## X.train1485   0.0025835115 NA X.train1485
## X.train1610  -0.0031937174 NA X.train1610
## X.train1728  -0.0034394181 NA X.train1728
## X.train1881   0.0056552895 NA X.train1881
## X.train1882  -0.0020661221 NA X.train1882
## X.train1942   0.0072337361 NA X.train1942
```

```
extract.coef(cv.out.en2)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0637599176 NA (Intercept)
## X.train352    0.0045261881 NA  X.train352
## X.train512   -0.0048009649 NA  X.train512
## X.train678    0.0083426285 NA  X.train678
## X.train719    0.0144271067 NA  X.train719
## X.train988   -0.0025009949 NA  X.train988
## X.train1091  -0.0030155037 NA X.train1091
## X.train1485   0.0004102417 NA X.train1485
## X.train1610  -0.0013657788 NA X.train1610
## X.train1728  -0.0016164004 NA X.train1728
## X.train1881   0.0044491147 NA X.train1881
## X.train1942   0.0066746951 NA X.train1942
```

```
extract.coef(cv.out.en3)
```

```
##                   Value SE Coefficient
## X.Intercept. 0.06374977 NA (Intercept)
```

```
extract.coef(cv.out.en4)
```

```
##                   Value SE Coefficient
## X.Intercept. 0.06374977 NA (Intercept)
```

```
extract.coef(cv.out.en5)
```

```
##                   Value SE Coefficient
## X.Intercept. 0.06374977 NA (Intercept)
```

```
extract.coef(cv.out.en6)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0637482542 NA (Intercept)
## X.train352    0.0056965204 NA  X.train352
## X.train512   -0.0061661300 NA  X.train512
## X.train678    0.0109545813 NA  X.train678
## X.train719    0.0186965468 NA  X.train719
## X.train988   -0.0031228093 NA  X.train988
## X.train1091  -0.0036243156 NA X.train1091
## X.train1485   0.0004097393 NA X.train1485
## X.train1610  -0.0016494846 NA X.train1610
## X.train1728  -0.0019719028 NA X.train1728
## X.train1881   0.0054887402 NA X.train1881
## X.train1942   0.0088438270 NA X.train1942
```

```
extract.coef(cv.out.en7)
```

```
##                   Value SE Coefficient
## X.Intercept. 0.06374977 NA (Intercept)
```

```
extract.coef(cv.out.en8)
```

```
##                    Value SE Coefficient
## X.Intercept. 0.06374977 NA (Intercept)
```

```
extract.coef(cv.out.en9)
```

```
##                    Value SE Coefficient
## X.Intercept. 0.06374977 NA (Intercept)
```

(c) (2 pt) Simulate an independent {**test**} **data set of the same type as above (response** y **and** 2000 **features per subject) with** $n = 10,000$**. Use seed = 4567.**

```
set.seed(4567)
X = matrix(rnorm(10000*2000),10000,2000)
test = sample(1:nrow(X), nrow(X) / 1)

beta = c(rep(1,20), rep(0, 1980))
epsilon = rnorm(1000)

Y = X*beta + epsilon

X.test = X[test,]
Y.test = Y[test]


test.mat = model.matrix(Y.test ~ X.test)
```

(d) (2 pt) Using the models you obtained above using the training data set and the 11 models above, compute average test error for each of the 11 models. Which one is the "best" model?

```
pred.ridge <- predict(ridge.model.train, s = bestlam.ridge, newx = test.mat)
ridge.MSPE = mean((pred.ridge - Y.test)^2)
ridge.MSPE
```

```
## [1] 1.20207
```

```
pred.lasso <- predict(lasso.model.train, s = bestlam.lasso, newx = test.mat)
lasso.MSPE = mean((pred.lasso -Y.test)^2)
lasso.MSPE
```

```
## [1] 0.9958705
```

```
pred.en1 <- predict(en.model.train1, s = bestlam.en1, newx = test.mat)
en1.MSPE = mean((pred.en1 - Y.test)^2)
en1.MSPE
```

```
## [1] 0.9957768
```

```
pred.en2 <- predict(en.model.train2, s = bestlam.en2, newx = test.mat)
en2.MSPE = mean((pred.en2 - Y.test)^2)
en2.MSPE
```

```
## [1] 0.9958705
```

```
pred.en3 <- predict(en.model.train3, s = bestlam.en3, newx = test.mat)
en3.MSPE = mean((pred.en3 - Y.test)^2)
en3.MSPE
```

```
## [1] 0.9958705
```

```
pred.en4 <- predict(en.model.train4, s = bestlam.en4, newx = test.mat)
en4.MSPE = mean((pred.en4 - Y.test)^2)
en4.MSPE
```

```
## [1] 0.9958705
```

```
pred.en5 <- predict(en.model.train5, s = bestlam.en5, newx = test.mat)
en5.MSPE = mean((pred.en5 - Y.test)^2)
en5.MSPE
```

```
## [1] 0.9958705
```

```
pred.en6 <- predict(en.model.train6, s = bestlam.en6, newx = test.mat)
en6.MSPE = mean((pred.en6 - Y.test)^2)
en6.MSPE
```

```
## [1] 0.9958705
```

```
pred.en7 <- predict(en.model.train7, s = bestlam.en7, newx = test.mat)
en7.MSPE = mean((pred.en7 - Y.test)^2)
en7.MSPE
```

```
## [1] 0.9958705
```

```
pred.en8 <- predict(en.model.train8, s = bestlam.en8, newx = test.mat)
en8.MSPE = mean((pred.en8 - Y.test)^2)
en8.MSPE
```

```
## [1] 0.9958705
```

```
pred.en9 <- predict(en.model.train9, s = bestlam.en9, newx = test.mat)
en9.MSPE = mean((pred.en9 - Y.test)^2)
en9.MSPE
```

```
## [1] 0.9958705
```

```
The best model is elastic net with an alpha of .3. It yields the lowest MSPE.
```

4. Do all the 4 parts of problem 3 but where the underlying model is:

$\beta = (\underbrace{1, ..., 1}_{1000}, \underbrace{0, ..., 0}_{1000})^T$

(a)(2 pt) Generate the above data with seed = 8910

```
set.seed(8910)
X = matrix(rnorm(1000*2000), 1000, 2000)
train = sample(1:nrow(X), nrow(X) / 1)

beta = c(rep(1,1000), rep(0, 1000))
epsilon = rnorm(1000)

Y = X*beta + epsilon

X.train = X[train,]
Y.train = Y[train]


train.mat = model.matrix(Y.train ~ X.train)
```

(b) (6 pt) Using glmnet fit Lasso, ridge regression and elastic net with $\alpha = .1, .2, .3, .4, .5, .6, .7, .8, .9$

```
# YOUR CODE HERE (uncomment first of course)
```

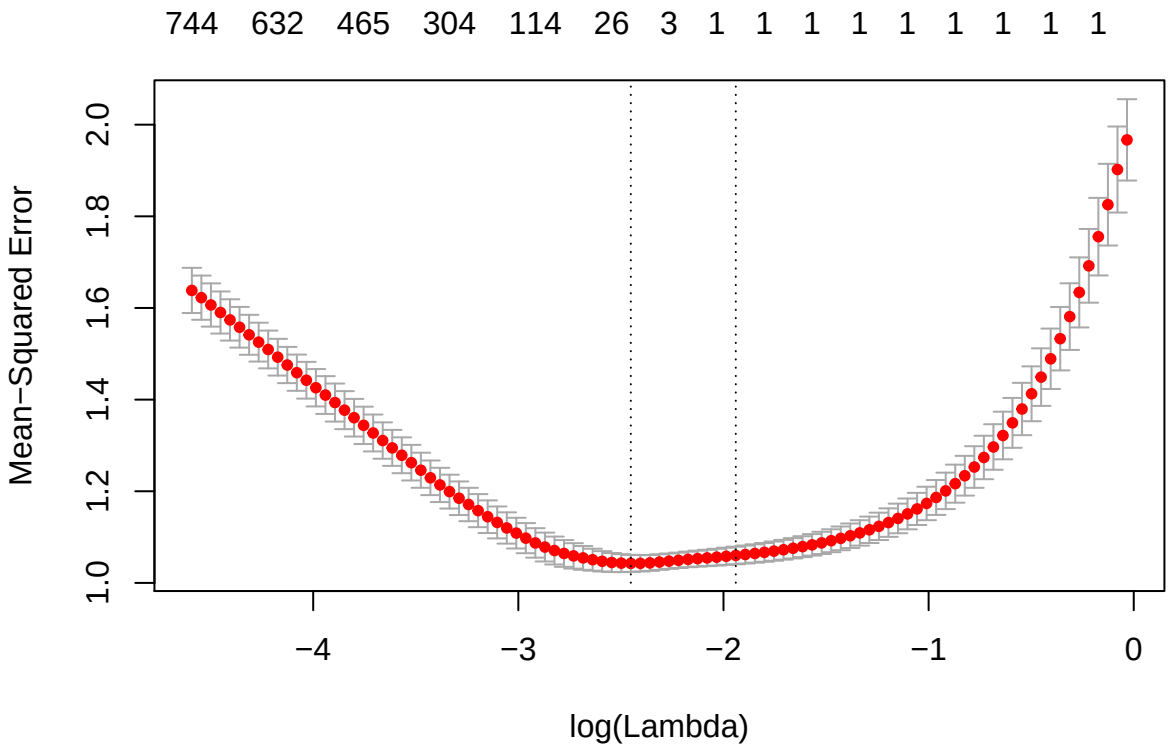
What I am looking for: (outputting the entire model for each one of the above is not-trivial so):

- code showing the fitting of each of the above models

```
lasso.model.train <- glmnet(train.mat, Y.train, alpha = 1, lambda = grid.lambda)
ridge.model.train <- glmnet(train.mat, Y.train, alpha = 0, lambda = grid.lambda)
en.model.train1<- glmnet(train.mat, Y.train, alpha = .1, lambda = grid.lambda)
en.model.train2<- glmnet(train.mat, Y.train, alpha = .2, lambda = grid.lambda)
en.model.train3<- glmnet(train.mat, Y.train, alpha = .3, lambda = grid.lambda)
en.model.train4<- glmnet(train.mat, Y.train, alpha = .4, lambda = grid.lambda)
en.model.train5<- glmnet(train.mat, Y.train, alpha = .5, lambda = grid.lambda)
en.model.train6<- glmnet(train.mat, Y.train, alpha = .6, lambda = grid.lambda)
en.model.train7<- glmnet(train.mat, Y.train, alpha = .7, lambda = grid.lambda)
en.model.train8<- glmnet(train.mat, Y.train, alpha = .8, lambda = grid.lambda)
en.model.train9<- glmnet(train.mat, Y.train, alpha = .9, lambda = grid.lambda)
```

- For ridge, Lasso and for $\alpha = .2, , .4, .6$ plot the cross-validated (6 fold) MSE versus lambda as well as your optimal value of *lambda* for ridge, Lasso and for $\alpha = .2, , .4, .6$

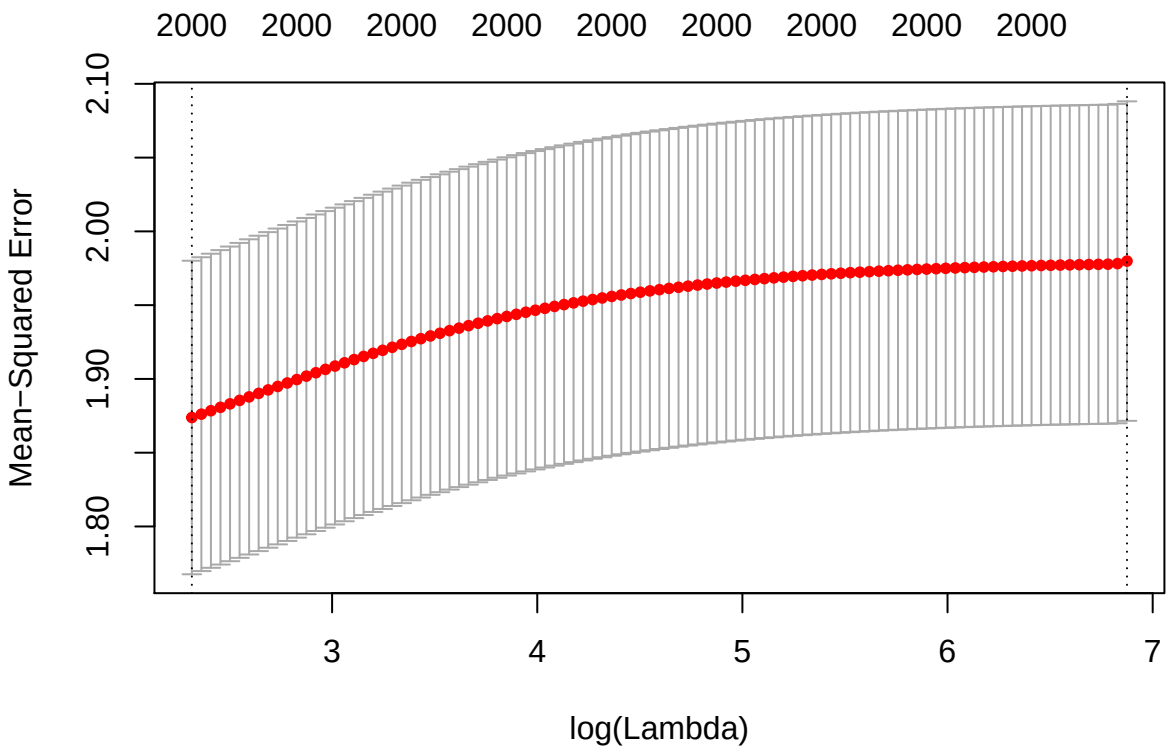
```
cv.out.lasso <- cv.glmnet(train.mat, Y.train, alpha = 1, nfolds = 6)
plot(cv.out.lasso)
```




```
bestlam.lasso <- cv.out$lambda.min
bestlam.lasso
```

```
## [1] 1.004802
```

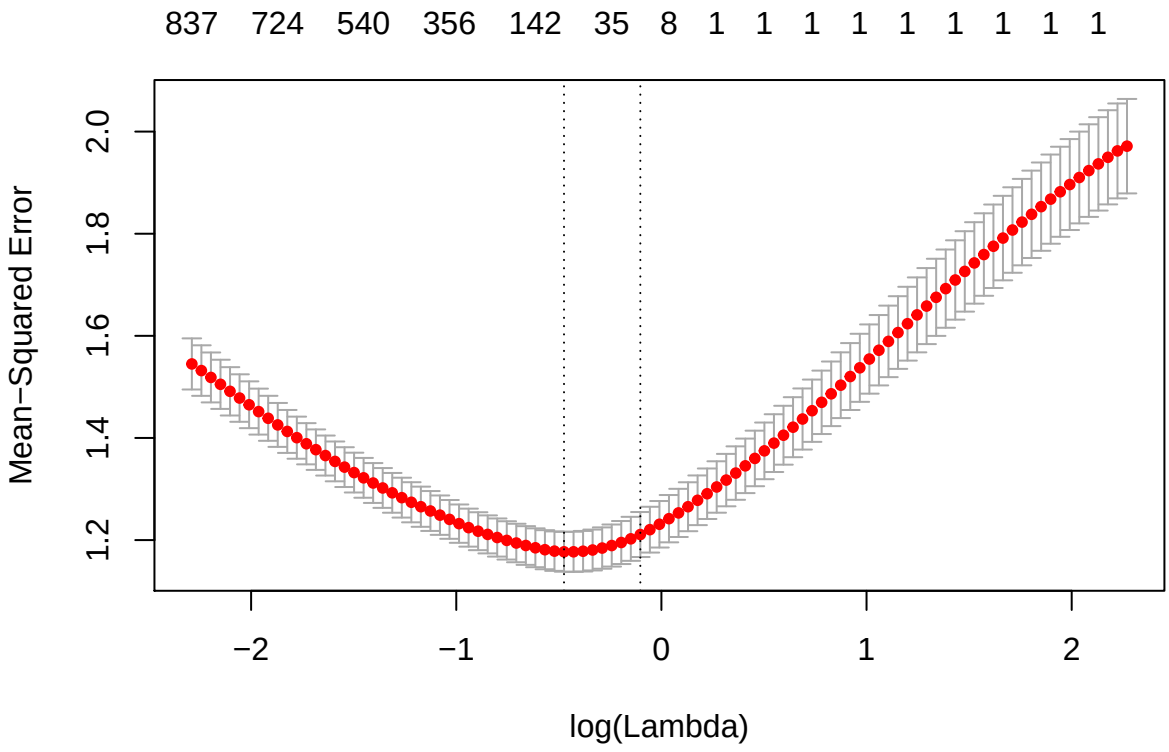
```
cv.out.ridge <- cv.glmnet(train.mat, Y.train, alpha = 0, nfolds = 6)
plot(cv.out.ridge)
```




```
bestlam.ridge <- cv.out$lambda.min
bestlam.ridge
```

```
## [1] 1.004802
```

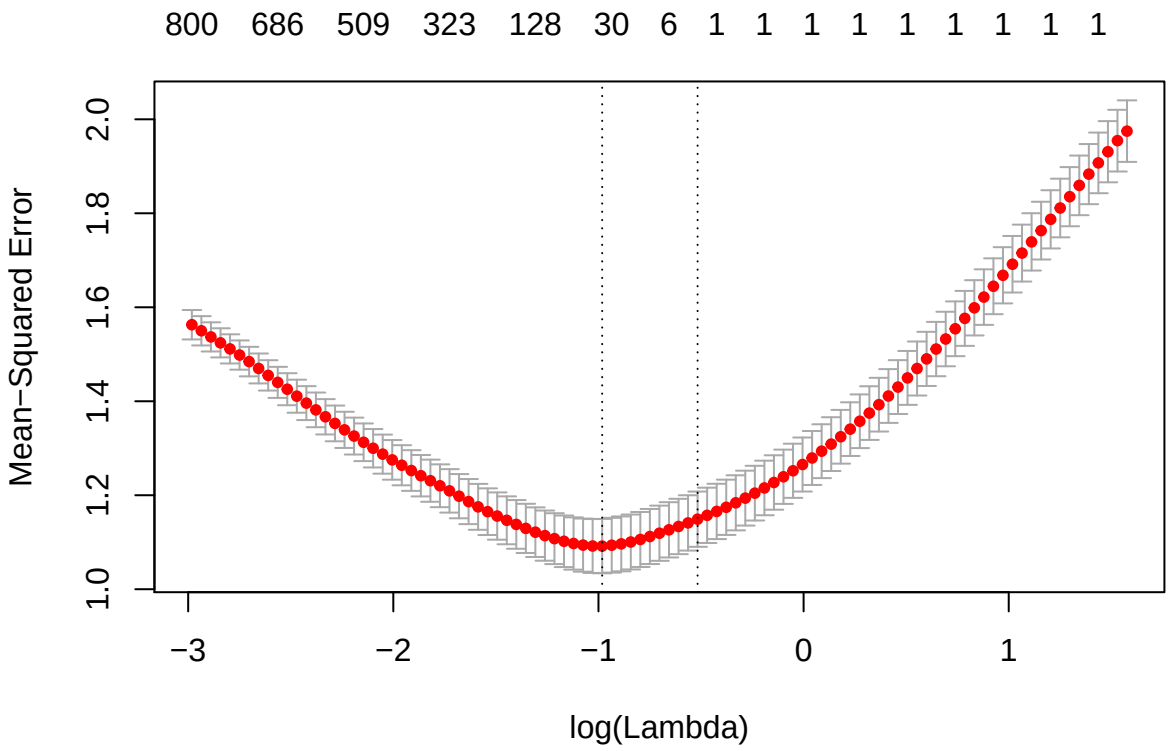
```
cv.out.en1 <- cv.glmnet(train.mat, Y.train, alpha = .1, nfolds = 6)
plot(cv.out.en1)
```



```
bestlam.en1 <- cv.out$lambda.min
bestlam.en1
```

```
## [1] 1.004802
```

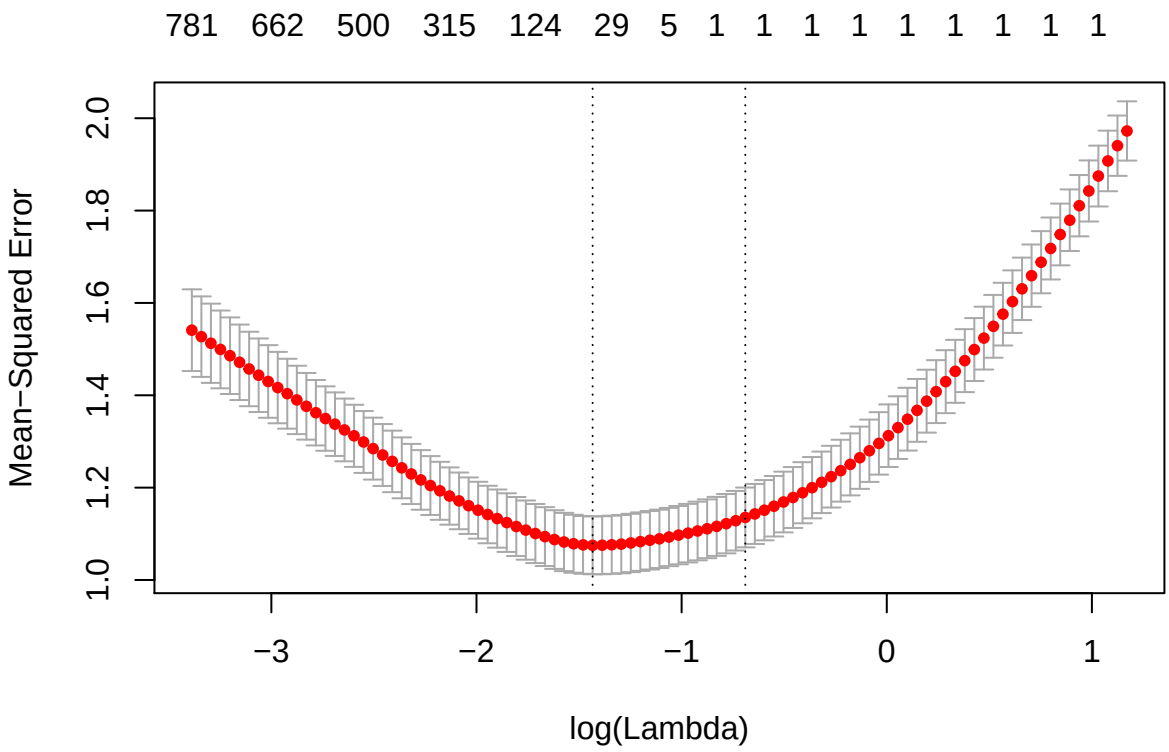
```
cv.out.en2 <- cv.glmnet(train.mat, Y.train, alpha = .2, nfolds = 6)
plot(cv.out.en2)
```




```
bestlam.en2 <- cv.out$lambda.min
bestlam.en2
```

```
## [1] 1.004802
```

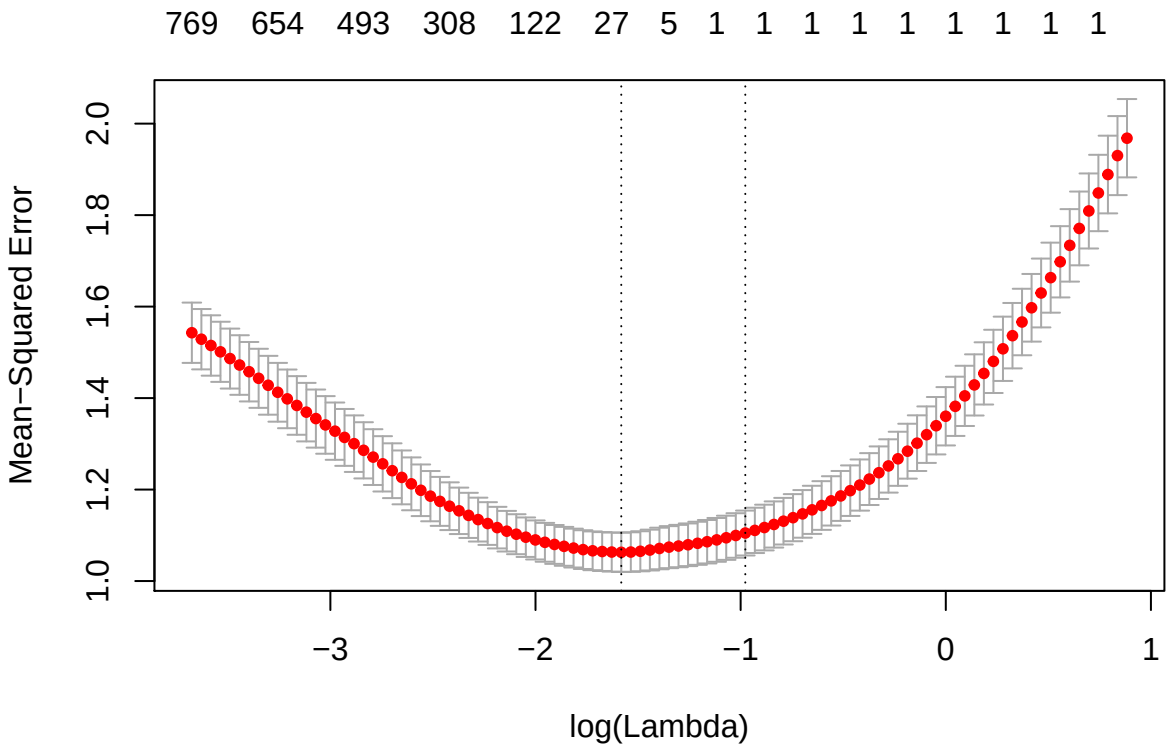
```
cv.out.en3 <- cv.glmnet(train.mat, Y.train, alpha = .3, nfolds = 6)
plot(cv.out.en3)
```




```
bestlam.en3 <- cv.out$lambda.min
bestlam.en3
```

```
## [1] 1.004802
```

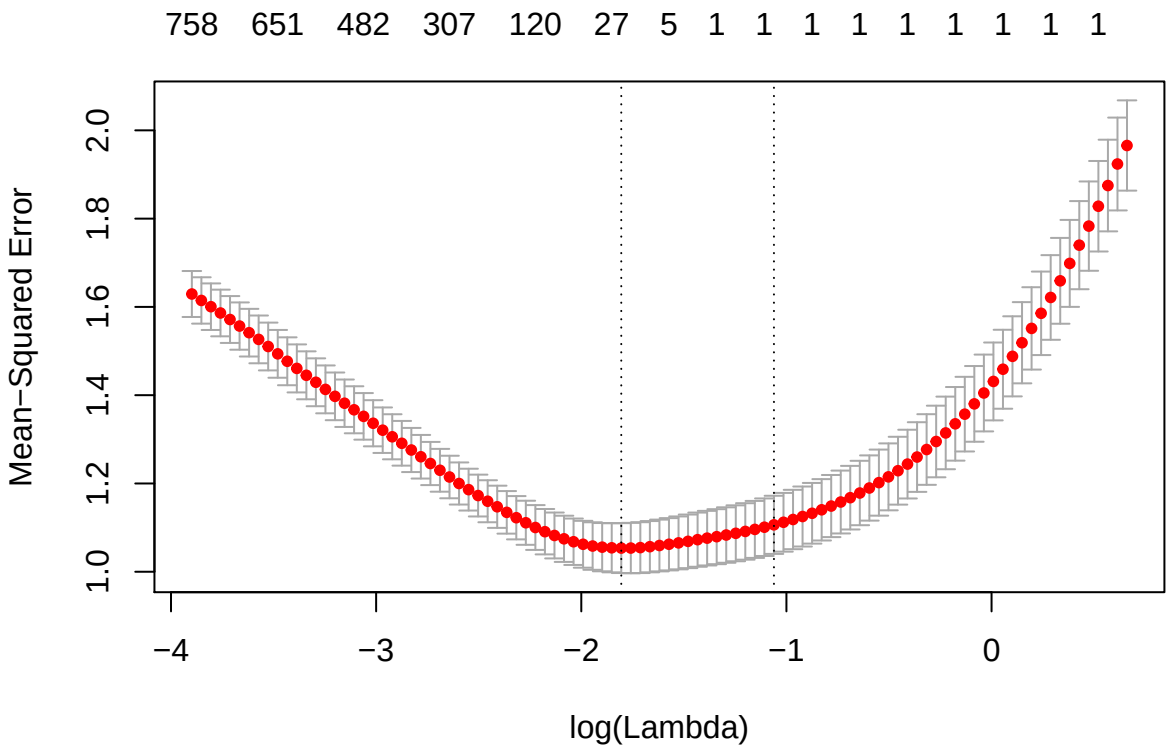
```
cv.out.en4 <- cv.glmnet(train.mat, Y.train, alpha = .4, nfolds = 6)
plot(cv.out.en4)
```



```
bestlam.en4 <- cv.out$lambda.min
bestlam.en4
```

```
## [1] 1.004802
```

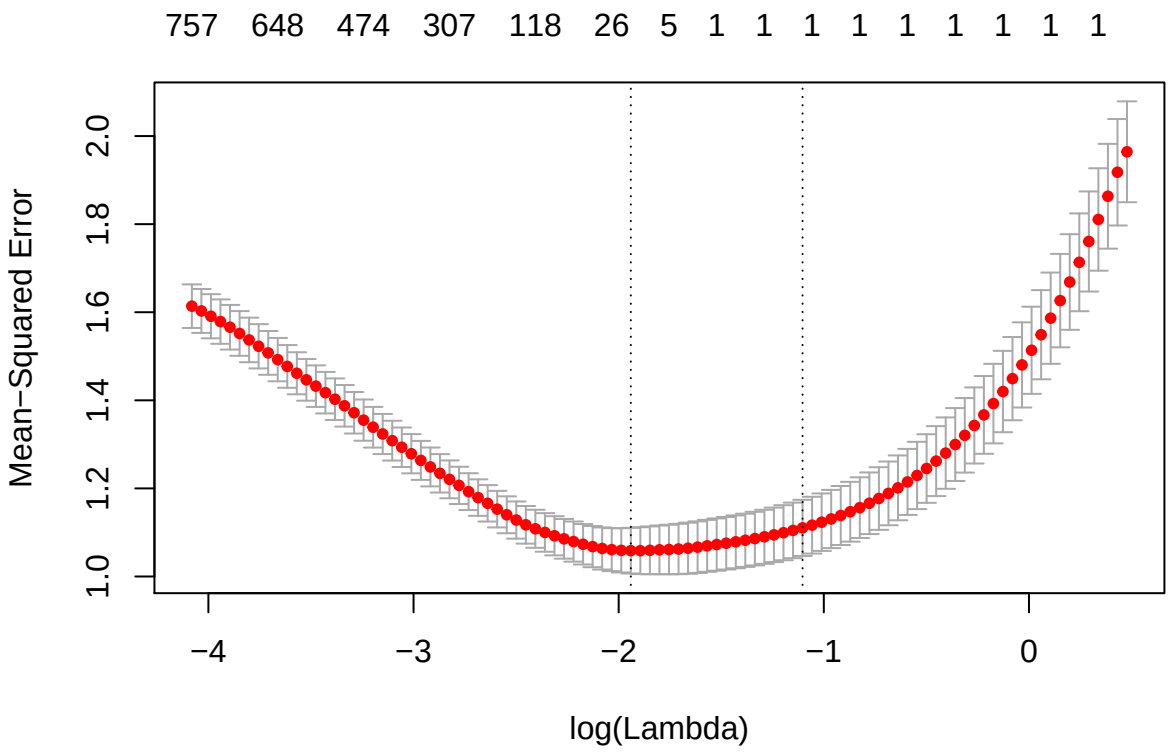
```
cv.out.en5 <- cv.glmnet(train.mat, Y.train, alpha = .5, nfolds = 6)
plot(cv.out.en5)
```




```
bestlam.en5 <- cv.out$lambda.min
bestlam.en5
```

```
## [1] 1.004802
```

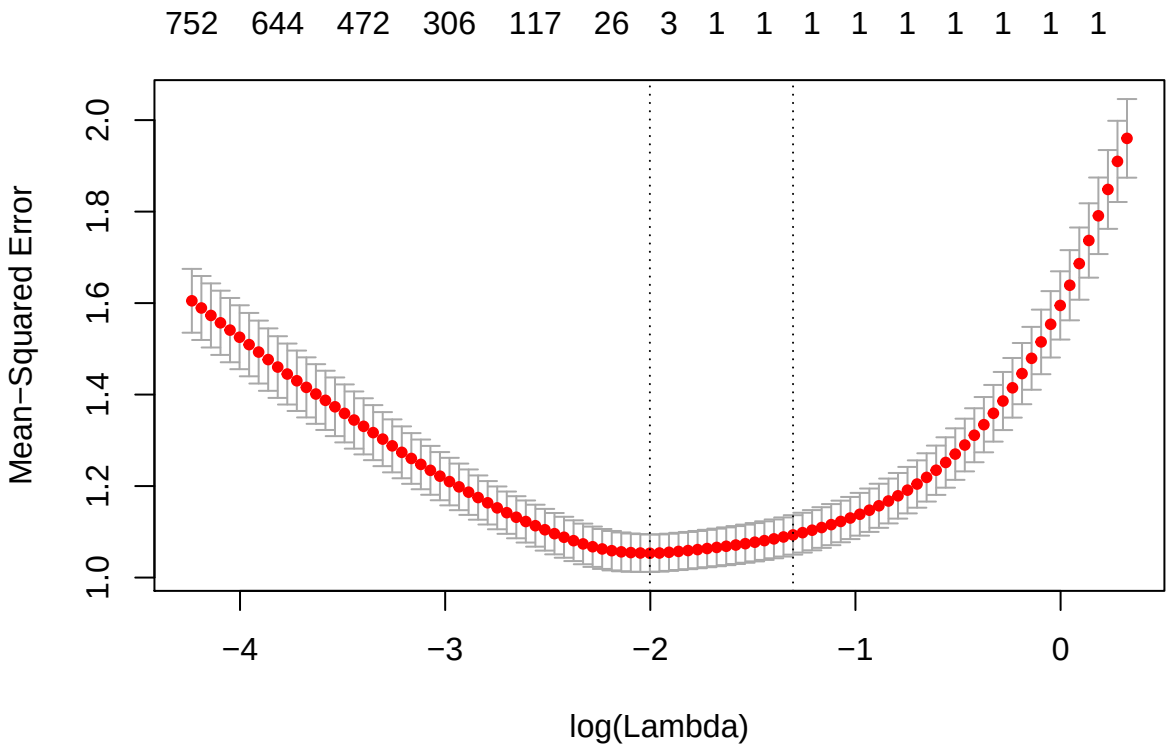
```
cv.out.en6 <- cv.glmnet(train.mat, Y.train, alpha = .6, nfolds = 6)
plot(cv.out.en6)
```




```
bestlam.en6 <- cv.out$lambda.min
bestlam.en6
```

```
## [1] 1.004802
```

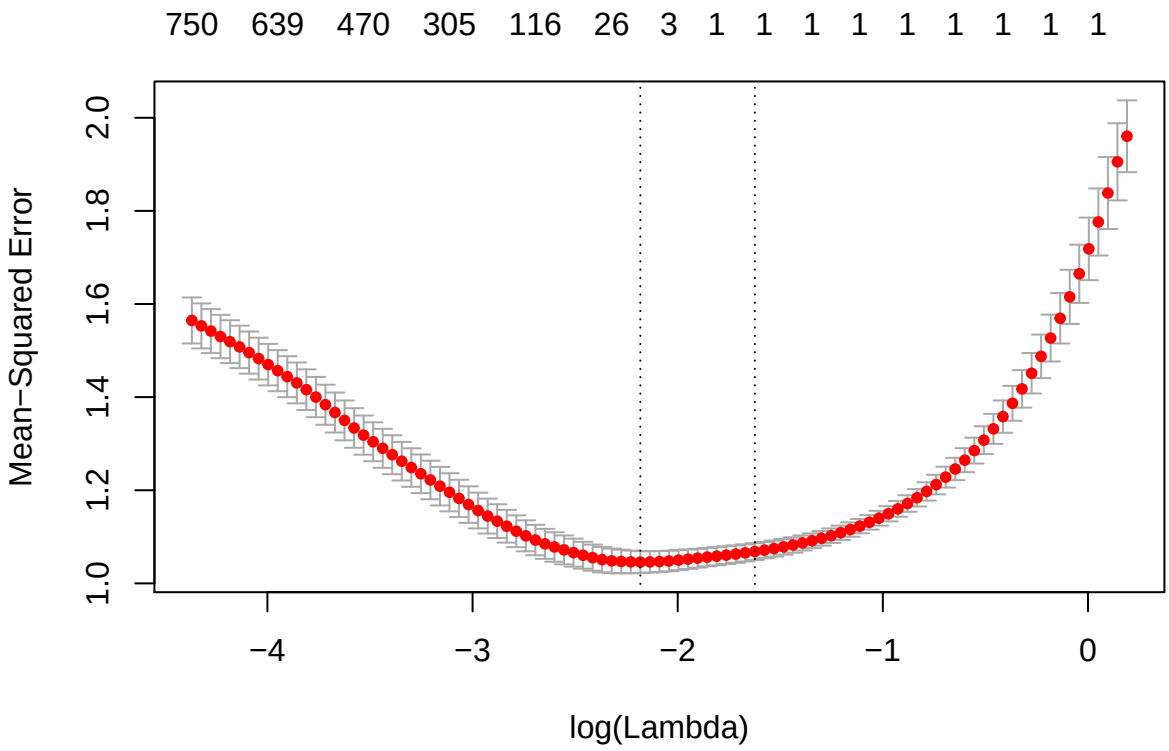
```
cv.out.en7 <- cv.glmnet(train.mat, Y.train, alpha = .7, nfolds = 6)
plot(cv.out.en7)
```



```
bestlam.en7 <- cv.out$lambda.min
bestlam.en7
```

```
## [1] 1.004802
```

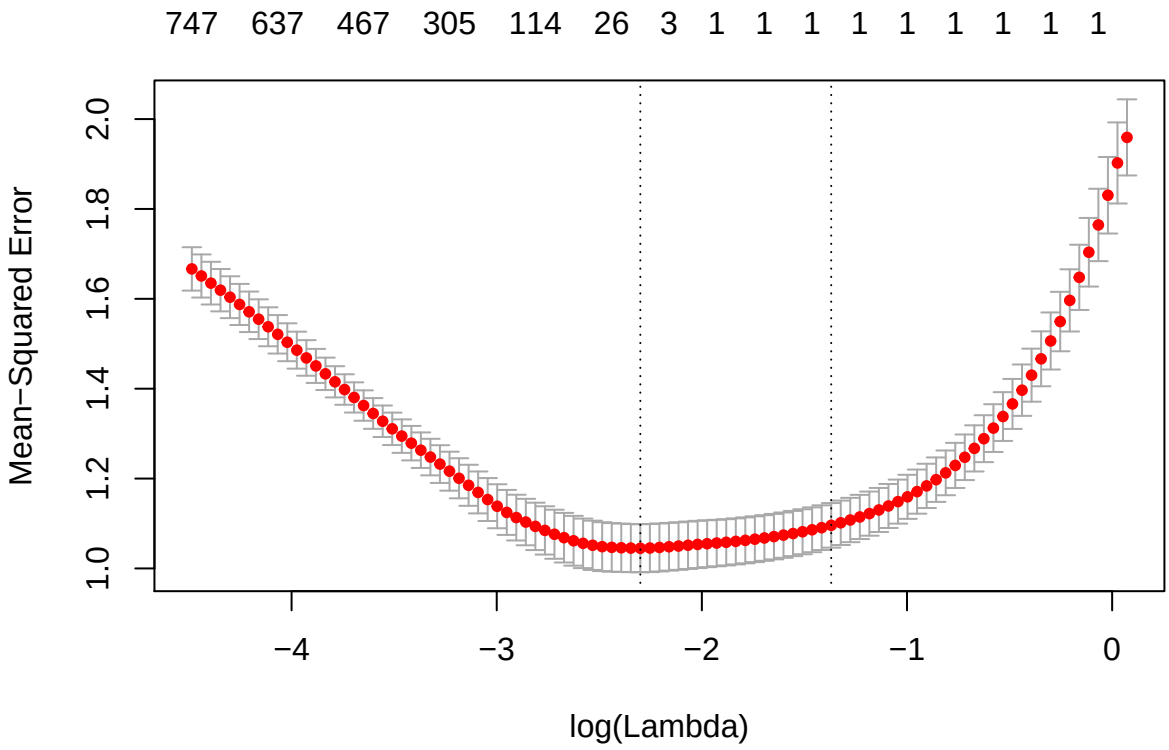
```
cv.out.en8 <- cv.glmnet(train.mat, Y.train, alpha = .8, nfolds = 6)
plot(cv.out.en8)
```




```
bestlam.en8 <- cv.out$lambda.min
bestlam.en8
```

```
## [1] 1.004802
```

```
cv.out.en9 <- cv.glmnet(train.mat, Y.train, alpha = .9, nfolds = 6)
plot(cv.out.en9)
```




```
bestlam.en9 <- cv.out$lambda.min
bestlam.en9
```

```
## [1] 1.004802
```

- The number of non-zero regression coeffecients for each of the above techniques.

```
library(coefplot)
extract.coef(cv.out.lasso)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0054251309 NA (Intercept)
## X.train1      0.8658272578 NA    X.train1
## X.train195   -0.0368069563 NA  X.train195
## X.train479    0.0062791558 NA  X.train479
## X.train577   -0.0032734962 NA  X.train577
## X.train850    0.0149881834 NA  X.train850
## X.train1101   0.0007219154 NA X.train1101
## X.train1119   0.0003832804 NA X.train1119
## X.train1220  -0.0044782758 NA X.train1220
## X.train1262  -0.0284762550 NA X.train1262
## X.train1342  -0.0135797345 NA X.train1342
## X.train1474   0.0029316583 NA X.train1474
## X.train1609  -0.0146457932 NA X.train1609
## X.train1664   0.0030750995 NA X.train1664
## X.train1711  -0.0081120089 NA X.train1711
## X.train1718  -0.0027526963 NA X.train1718
## X.train1730  -0.0129578212 NA X.train1730
## X.train1756  -0.0142264413 NA X.train1756
```

```
extract.coef(cv.out.en1)
```

```
##                      Value SE Coefficient
```

```
## X.Intercept.  1.095352e-02 NA (Intercept)
## X.train1      6.255458e-01 NA    X.train1
## X.train11    -4.535583e-03 NA   X.train11
## X.train38     2.797010e-03 NA   X.train38
## X.train67    -1.000609e-02 NA   X.train67
## X.train90     2.555485e-03 NA   X.train90
## X.train93    -3.427971e-03 NA   X.train93
## X.train120    4.049635e-03 NA  X.train120
## X.train133   -7.450704e-03 NA  X.train133
## X.train139   -1.446105e-02 NA  X.train139
## X.train140   -4.707313e-03 NA  X.train140
## X.train154   -5.306816e-03 NA  X.train154
## X.train177   -1.076933e-04 NA  X.train177
## X.train187   -2.632806e-03 NA  X.train187
## X.train195   -4.127559e-02 NA  X.train195
## X.train243   -3.778953e-05 NA  X.train243
## X.train251    1.691768e-03 NA  X.train251
## X.train253    1.082075e-02 NA  X.train253
## X.train255   -5.607043e-03 NA  X.train255
## X.train263   -4.480800e-03 NA  X.train263
## X.train268    1.145766e-02 NA  X.train268
## X.train361    3.720608e-03 NA  X.train361
## X.train398   -9.833893e-03 NA  X.train398
## X.train409    6.071941e-03 NA  X.train409
## X.train414    1.485167e-02 NA  X.train414
## X.train422    1.152181e-02 NA  X.train422
## X.train445    1.070487e-03 NA  X.train445
## X.train453   -1.816294e-03 NA  X.train453
## X.train479    2.354734e-02 NA  X.train479
## X.train498    1.154706e-02 NA  X.train498
## X.train511    7.216611e-03 NA  X.train511
## X.train527    9.449930e-04 NA  X.train527
## X.train530   -8.598308e-03 NA  X.train530
## X.train536   -5.131550e-04 NA  X.train536
## X.train542   -1.431779e-04 NA  X.train542
## X.train577   -2.215310e-02 NA  X.train577
## X.train592   -3.699418e-03 NA  X.train592
## X.train663    1.116104e-02 NA  X.train663
## X.train697    1.393349e-02 NA  X.train697
## X.train719   -2.217185e-03 NA  X.train719
## X.train747    8.535335e-04 NA  X.train747
## X.train803    8.059935e-04 NA  X.train803
## X.train842   -8.961497e-03 NA  X.train842
## X.train850    3.235351e-02 NA  X.train850
## X.train864    2.937222e-03 NA  X.train864
## X.train880   -3.312435e-03 NA  X.train880
## X.train898    9.843080e-03 NA  X.train898
## X.train922   -9.675211e-03 NA  X.train922
## X.train936   -1.165240e-02 NA  X.train936
## X.train989   -3.276077e-03 NA  X.train989
## X.train1015   4.037549e-03 NA X.train1015
## X.train1024   3.097635e-03 NA X.train1024
## X.train1038   6.831771e-04 NA X.train1038
## X.train1101   2.523787e-02 NA X.train1101
```

```
## X.train1119   9.031247e-03 NA X.train1119
## X.train1146  -1.813447e-02 NA X.train1146
## X.train1154  -1.733446e-03 NA X.train1154
## X.train1176  -8.345832e-04 NA X.train1176
## X.train1220  -2.319111e-02 NA X.train1220
## X.train1224   4.950511e-03 NA X.train1224
## X.train1250   1.244349e-02 NA X.train1250
## X.train1262  -3.928700e-02 NA X.train1262
## X.train1264   2.055154e-03 NA X.train1264
## X.train1312   8.767592e-03 NA X.train1312
## X.train1325   2.640615e-04 NA X.train1325
## X.train1327   3.436688e-03 NA X.train1327
## X.train1342  -2.199494e-02 NA X.train1342
## X.train1354   1.579522e-03 NA X.train1354
## X.train1384   2.766495e-03 NA X.train1384
## X.train1388  -3.276649e-03 NA X.train1388
## X.train1390   3.342817e-03 NA X.train1390
## X.train1418  -4.665979e-03 NA X.train1418
## X.train1424   5.373360e-03 NA X.train1424
## X.train1474   3.008676e-02 NA X.train1474
## X.train1479   1.315935e-03 NA X.train1479
## X.train1496   8.016581e-03 NA X.train1496
## X.train1503  -1.390366e-02 NA X.train1503
## X.train1609  -3.504435e-02 NA X.train1609
## X.train1612   8.377463e-03 NA X.train1612
## X.train1664   1.700470e-02 NA X.train1664
## X.train1710   1.853816e-03 NA X.train1710
## X.train1711  -2.712552e-02 NA X.train1711
## X.train1718  -1.860622e-02 NA X.train1718
## X.train1730  -1.405902e-02 NA X.train1730
## X.train1756  -2.068673e-02 NA X.train1756
## X.train1826   4.289566e-04 NA X.train1826
## X.train1829   4.053685e-03 NA X.train1829
## X.train1894  -3.568199e-03 NA X.train1894
## X.train1898  -7.544604e-03 NA X.train1898
## X.train1945   7.787408e-03 NA X.train1945
## X.train1966  -4.703058e-03 NA X.train1966
```

```
extract.coef(cv.out.en2)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0082965705 NA (Intercept)
## X.train1      0.7183037279 NA    X.train1
## X.train67    -0.0042037333 NA   X.train67
## X.train139   -0.0063836756 NA  X.train139
## X.train195   -0.0388250043 NA  X.train195
## X.train253    0.0045601333 NA  X.train253
## X.train268    0.0027783793 NA  X.train268
## X.train398   -0.0010516031 NA  X.train398
## X.train414    0.0078923192 NA  X.train414
## X.train422    0.0053285399 NA  X.train422
## X.train479    0.0158887606 NA  X.train479
## X.train498    0.0020596555 NA  X.train498
## X.train530   -0.0026670519 NA  X.train530
## X.train577   -0.0151137493 NA  X.train577
```

```
## X.train663    0.0034822781 NA  X.train663
## X.train697    0.0057152492 NA  X.train697
## X.train842   -0.0011653754 NA  X.train842
## X.train850    0.0259587049 NA  X.train850
## X.train880   -0.0012938505 NA  X.train880
## X.train898    0.0011396224 NA  X.train898
## X.train936   -0.0006890662 NA  X.train936
## X.train1101   0.0162631700 NA X.train1101
## X.train1119   0.0052598968 NA X.train1119
## X.train1146  -0.0098397776 NA X.train1146
## X.train1220  -0.0160744279 NA X.train1220
## X.train1250   0.0059907410 NA X.train1250
## X.train1262  -0.0344564975 NA X.train1262
## X.train1312   0.0021231153 NA X.train1312
## X.train1342  -0.0165803927 NA X.train1342
## X.train1474   0.0198648246 NA X.train1474
## X.train1503  -0.0042776969 NA X.train1503
## X.train1609  -0.0270115190 NA X.train1609
## X.train1664   0.0106181406 NA X.train1664
## X.train1711  -0.0193207729 NA X.train1711
## X.train1718  -0.0119977847 NA X.train1718
## X.train1730  -0.0133493596 NA X.train1730
## X.train1756  -0.0172546660 NA X.train1756
## X.train1966  -0.0007458238 NA X.train1966
```

```
extract.coef(cv.out.en3)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0065060107 NA (Intercept)
## X.train1      0.7822619590 NA    X.train1
## X.train67    -0.0107447200 NA   X.train67
## X.train139   -0.0077831328 NA  X.train139
## X.train140   -0.0006801439 NA  X.train140
## X.train195   -0.0446805695 NA  X.train195
## X.train253    0.0083387975 NA  X.train253
## X.train255   -0.0020082308 NA  X.train255
## X.train268    0.0044942491 NA  X.train268
## X.train398   -0.0007701622 NA  X.train398
## X.train414    0.0116468712 NA  X.train414
## X.train422    0.0079540440 NA  X.train422
## X.train479    0.0191640553 NA  X.train479
## X.train498    0.0041769956 NA  X.train498
## X.train511    0.0007834062 NA  X.train511
## X.train530   -0.0056730212 NA  X.train530
## X.train577   -0.0185740496 NA  X.train577
## X.train663    0.0058039417 NA  X.train663
## X.train697    0.0054910297 NA  X.train697
## X.train842   -0.0031355382 NA  X.train842
## X.train850    0.0287136188 NA  X.train850
## X.train880   -0.0059879906 NA  X.train880
## X.train898    0.0016015609 NA  X.train898
## X.train936   -0.0012981577 NA  X.train936
## X.train1101   0.0172413573 NA X.train1101
## X.train1119   0.0089753612 NA X.train1119
## X.train1146  -0.0114454697 NA X.train1146
```

```
## X.train1220  -0.0180479322 NA X.train1220
## X.train1250   0.0073756600 NA X.train1250
## X.train1262  -0.0394948375 NA X.train1262
## X.train1312   0.0021920227 NA X.train1312
## X.train1342  -0.0202890515 NA X.train1342
## X.train1424   0.0028053065 NA X.train1424
## X.train1474   0.0202291382 NA X.train1474
## X.train1503  -0.0059357323 NA X.train1503
## X.train1609  -0.0286318206 NA X.train1609
## X.train1664   0.0148927655 NA X.train1664
## X.train1711  -0.0218873997 NA X.train1711
## X.train1718  -0.0146816548 NA X.train1718
## X.train1730  -0.0188185403 NA X.train1730
## X.train1756  -0.0221735554 NA X.train1756
## X.train1966  -0.0029292343 NA X.train1966
```

```
extract.coef(cv.out.en4)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0067956725 NA (Intercept)
## X.train1      0.7976929953 NA    X.train1
## X.train67    -0.0006293592 NA   X.train67
## X.train195   -0.0370672914 NA  X.train195
## X.train253    0.0003260949 NA  X.train253
## X.train414    0.0025459496 NA  X.train414
## X.train479    0.0100517424 NA  X.train479
## X.train577   -0.0082894546 NA  X.train577
## X.train850    0.0199244183 NA  X.train850
## X.train1101   0.0076409805 NA X.train1101
## X.train1119   0.0021925659 NA X.train1119
## X.train1146  -0.0013206441 NA X.train1146
## X.train1220  -0.0095608209 NA X.train1220
## X.train1262  -0.0305821362 NA X.train1262
## X.train1342  -0.0143895310 NA X.train1342
## X.train1474   0.0106905134 NA X.train1474
## X.train1609  -0.0203613280 NA X.train1609
## X.train1664   0.0058949204 NA X.train1664
## X.train1711  -0.0130210587 NA X.train1711
## X.train1718  -0.0067483300 NA X.train1718
## X.train1730  -0.0122943365 NA X.train1730
## X.train1756  -0.0150153148 NA X.train1756
```

```
extract.coef(cv.out.en5)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0062124726 NA (Intercept)
## X.train1      0.8201470832 NA    X.train1
## X.train67    -0.0015952623 NA   X.train67
## X.train195   -0.0380882110 NA  X.train195
## X.train253    0.0006801971 NA  X.train253
## X.train414    0.0026750040 NA  X.train414
## X.train479    0.0099715798 NA  X.train479
## X.train577   -0.0081973849 NA  X.train577
## X.train850    0.0195904967 NA  X.train850
## X.train880   -0.0005083857 NA  X.train880
## X.train1101   0.0066215483 NA X.train1101
```

```
## X.train1119   0.0027400944 NA X.train1119
## X.train1146  -0.0005636682 NA X.train1146
## X.train1220  -0.0091216934 NA X.train1220
## X.train1262  -0.0311641899 NA X.train1262
## X.train1342  -0.0149116989 NA X.train1342
## X.train1474   0.0094008254 NA X.train1474
## X.train1609  -0.0197523334 NA X.train1609
## X.train1664   0.0062121161 NA X.train1664
## X.train1711  -0.0126947541 NA X.train1711
## X.train1718  -0.0066212503 NA X.train1718
## X.train1730  -0.0134784878 NA X.train1730
## X.train1756  -0.0158178435 NA X.train1756
```

```
extract.coef(cv.out.en6)
```

```
##                     Value SE Coefficient
## X.Intercept.  0.006297560 NA (Intercept)
## X.train1      0.831266944 NA    X.train1
## X.train195   -0.035334198 NA  X.train195
## X.train479    0.006528799 NA  X.train479
## X.train577   -0.003685388 NA  X.train577
## X.train850    0.015753406 NA  X.train850
## X.train1101   0.002640903 NA X.train1101
## X.train1220  -0.005391526 NA X.train1220
## X.train1262  -0.027716110 NA X.train1262
## X.train1342  -0.012732894 NA X.train1342
## X.train1474   0.005312801 NA X.train1474
## X.train1609  -0.015870279 NA X.train1609
## X.train1664   0.002809840 NA X.train1664
## X.train1711  -0.008818566 NA X.train1711
## X.train1718  -0.003235854 NA X.train1718
## X.train1730  -0.011098372 NA X.train1730
## X.train1756  -0.012945712 NA X.train1756
```

```
extract.coef(cv.out.en7)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0071178410 NA (Intercept)
## X.train1      0.8343348413 NA    X.train1
## X.train195   -0.0281806105 NA  X.train195
## X.train850    0.0079157828 NA  X.train850
## X.train1262  -0.0197593507 NA X.train1262
## X.train1342  -0.0064746865 NA X.train1342
## X.train1609  -0.0080663473 NA X.train1609
## X.train1711  -0.0001860238 NA X.train1711
## X.train1730  -0.0039654566 NA X.train1730
## X.train1756  -0.0054754604 NA X.train1756
```

```
extract.coef(cv.out.en8)
```

```
##                      Value SE Coefficient
## X.Intercept.  0.0063264626 NA (Intercept)
## X.train1      0.8487354437 NA    X.train1
## X.train195   -0.0324975737 NA  X.train195
## X.train479    0.0027092328 NA  X.train479
## X.train850    0.0114394211 NA  X.train850
```

```
## X.train1220  -0.0011020210 NA X.train1220
## X.train1262  -0.0241011127 NA X.train1262
## X.train1342  -0.0102704443 NA X.train1342
## X.train1474   0.0004826472 NA X.train1474
## X.train1609  -0.0113658288 NA X.train1609
## X.train1711  -0.0042772510 NA X.train1711
## X.train1730  -0.0085544790 NA X.train1730
## X.train1756  -0.0098547906 NA X.train1756
```

```
extract.coef(cv.out.en9)
```

```
##                       Value SE Coefficient
## X.Intercept.  0.0061478350 NA (Intercept)
## X.train1      0.8563543177 NA    X.train1
## X.train195   -0.0328014257 NA  X.train195
## X.train479    0.0025978435 NA  X.train479
## X.train850    0.0112244623 NA  X.train850
## X.train1220  -0.0008599822 NA X.train1220
## X.train1262  -0.0242364257 NA X.train1262
## X.train1342  -0.0104356852 NA X.train1342
## X.train1609  -0.0110697637 NA X.train1609
## X.train1711  -0.0040866088 NA X.train1711
## X.train1730  -0.0089352793 NA X.train1730
## X.train1756  -0.0100924990 NA X.train1756
```

(c) (2 pt) Simulate an independent {**test**} **data set of the same type as above (response** y **and** 2000 **features per subject) with** $n = 10,000$**. Use seed = 1112.**

```
set.seed(1112)
X = matrix(rnorm(10000*2000),10000,2000)
test = sample(1:nrow(X), nrow(X) / 1)

beta = c(rep(1,1000), rep(0, 1000))
epsilon = rnorm(1000)

Y = X*beta + epsilon

X.test = X[test,]
Y.test = Y[test]


test.mat = model.matrix(Y.test ~ X.test)
```

(d) (2 pt) Using the models you obtained above using the training data set and the 11 models above, compute average test error for each of the 11 models. Which one is the "best" model?

```
pred.ridge <- predict(ridge.model.train, s = bestlam.ridge, newx = test.mat)
ridge.MSPE = mean((pred.ridge - Y.test)^2)
ridge.MSPE
```

```
## [1] 1.673044
```

```
pred.lasso <- predict(lasso.model.train, s = bestlam.lasso, newx = test.mat)
lasso.MSPE = mean((pred.lasso -Y.test)^2)
lasso.MSPE
```

```
## [1] 1.446803
```

```
pred.en1 <- predict(en.model.train1, s = bestlam.en1, newx = test.mat)
en1.MSPE = mean((pred.en1 - Y.test)^2)
en1.MSPE
```

```
## [1] 1.253796
```

```
pred.en2 <- predict(en.model.train2, s = bestlam.en2, newx = test.mat)
en2.MSPE = mean((pred.en2 - Y.test)^2)
en2.MSPE
```

```
## [1] 1.25137
```

```
pred.en3 <- predict(en.model.train3, s = bestlam.en3, newx = test.mat)
en3.MSPE = mean((pred.en3 - Y.test)^2)
en3.MSPE
```

```
## [1] 1.2541
```

```
pred.en4 <- predict(en.model.train4, s = bestlam.en4, newx = test.mat)
en4.MSPE = mean((pred.en4 - Y.test)^2)
en4.MSPE
```

```
## [1] 1.261359
```

```
pred.en5 <- predict(en.model.train5, s = bestlam.en5, newx = test.mat)
en5.MSPE = mean((pred.en5 - Y.test)^2)
en5.MSPE
```

```
## [1] 1.274587
```

```
pred.en6 <- predict(en.model.train6, s = bestlam.en6, newx = test.mat)
en6.MSPE = mean((pred.en6 - Y.test)^2)
en6.MSPE
```

```
## [1] 1.295724
```

```
pred.en7 <- predict(en.model.train7, s = bestlam.en7, newx = test.mat)
en7.MSPE = mean((pred.en7 - Y.test)^2)
en7.MSPE
```

```
## [1] 1.327416
```

```
pred.en8 <- predict(en.model.train8, s = bestlam.en8, newx = test.mat)
en8.MSPE = mean((pred.en8 - Y.test)^2)
en8.MSPE
```

```
## [1] 1.373325
```

```
pred.en9 <- predict(en.model.train9, s = bestlam.en9, newx = test.mat)
en9.MSPE = mean((pred.en9 - Y.test)^2)
en9.MSPE
```

```
## [1] 1.413996
```

For this model, Elastic net with alpha of .7 is best based on MSPE.