ISLR CH6 Exercises

```r
library(tidyverse)
```

```
## -- Attaching packages -----------------------------------------------------------------------
## v ggplot2 3.3.0     v purrr   0.3.3
## v tibble  3.0.1     v dplyr   0.8.5
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## -- Conflicts -----------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(leaps)
hitters.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Hitters.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

# note that Salary is od type string and some of them are NA
sum(hitters.df$Salary=="NA")
```

```
## [1] 59
```

```r
# first remove character NAs
hitters.df <- hitters.df[hitters.df$Salary != "NA",]

# now convert Salary into numeric
hitters.df$Salary <- as.numeric(as.character(hitters.df$Salary))

# Now convert it to tibble
hitters.df <- tibble(hitters.df)

# regsubsets() part of lepas library chooses best subset using RSS


regfit.full <- regsubsets(Salary ~ ., hitters.df, nvmax = 19)

#The summary shows the result of step 2 of algorithm 6.1 page 205 of the book
summary <- summary(regfit.full)

names(summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```r
summary
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., hitters.df, nvmax = 19)
## 19 Variables  (and intercept)
##            Forced in Forced out
```

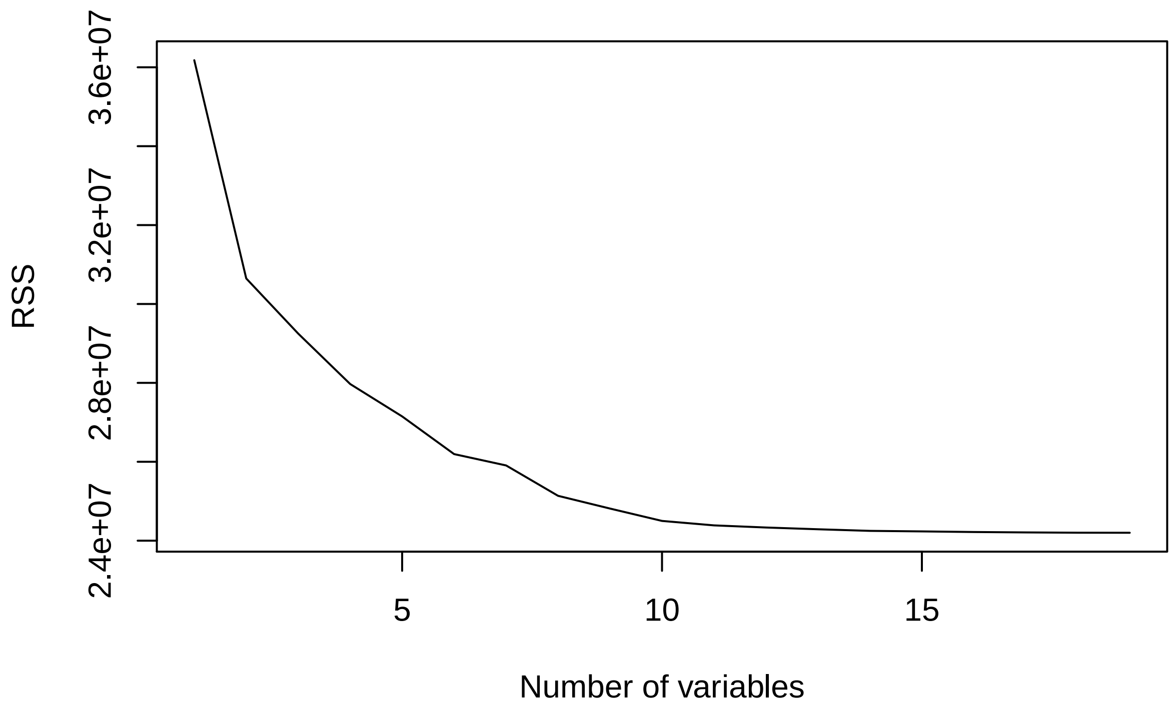
```
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: exhaustive
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 4  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*"
## 6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*"
## 7  ( 1 )  " "   "*"  " "   " "  " " "*"   " "   "*"    "*"   "*"    " "   " "
## 8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   "*"    "*"   " "
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*"
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"   "*"
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*"
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"   "*"
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*"
##           CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 2  ( 1 )  " "    " "     " "       " "     " "     " "    " "
## 3  ( 1 )  " "    " "     " "       "*"     " "     " "    " "
## 4  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 5  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 6  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 7  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "
## 8  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "
## 10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "
## 11  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
## 12  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "
```

```
## 13  ( 1 ) "*"    "*"     "*"        "*"     "*"     "*"    " "
## 14  ( 1 ) "*"    "*"     "*"        "*"     "*"     "*"    " "
## 15  ( 1 ) "*"    "*"     "*"        "*"     "*"     "*"    " "
## 16  ( 1 ) "*"    "*"     "*"        "*"     "*"     "*"    " "
## 17  ( 1 ) "*"    "*"     "*"        "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"    "*"     "*"        "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"    "*"     "*"        "*"     "*"     "*"    "*"
```

```r
# coef(, n) returns coefficient estimates associated with best n variable model
coef(regfit.full,4)
```

```
##  (Intercept)         Hits         CRBI    DivisionW      PutOuts
##   13.9231044    2.6757978    0.6817790 -139.9538855    0.2735002
```

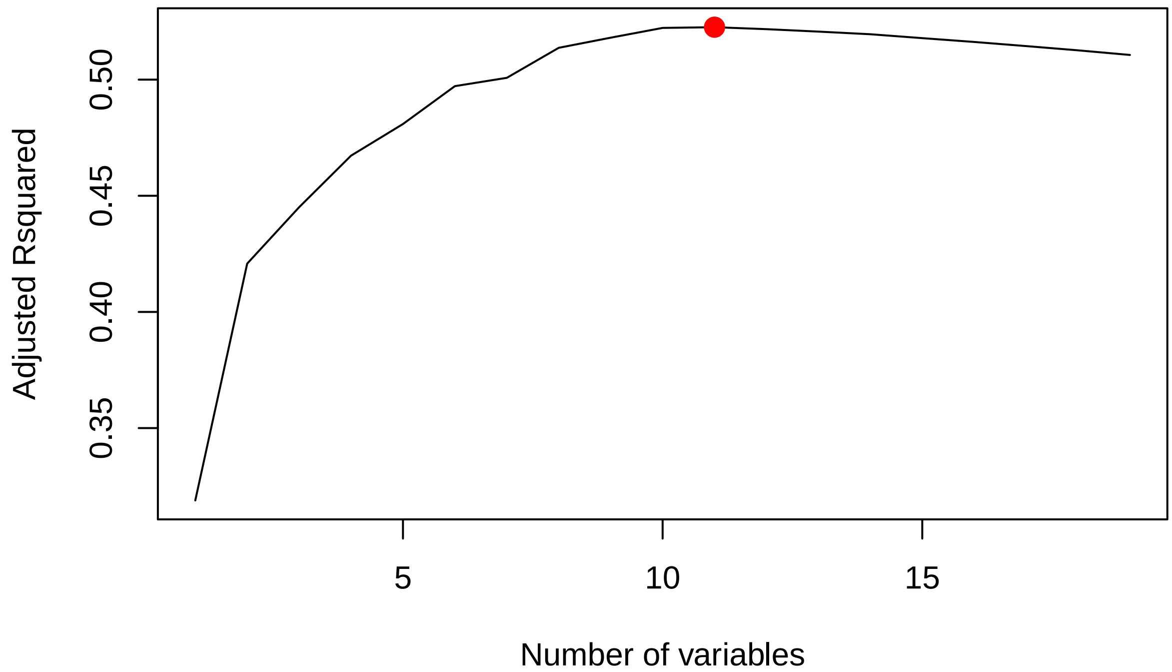
```r
# plot Rsq , Cp and BIC
# par(mfrow=c(1,1))
plot(summary$rss, xlab = "Number of variables", ylab="RSS",type = "l")
```




```r
plot(summary$adjr2,xlab = "Number of variables", ylab="Adjusted Rsquared",
     type = "l")

# which.max() returns location maximum point of the vector
(index <- which.max(summary$adjr2))
```

```
## [1] 11
```

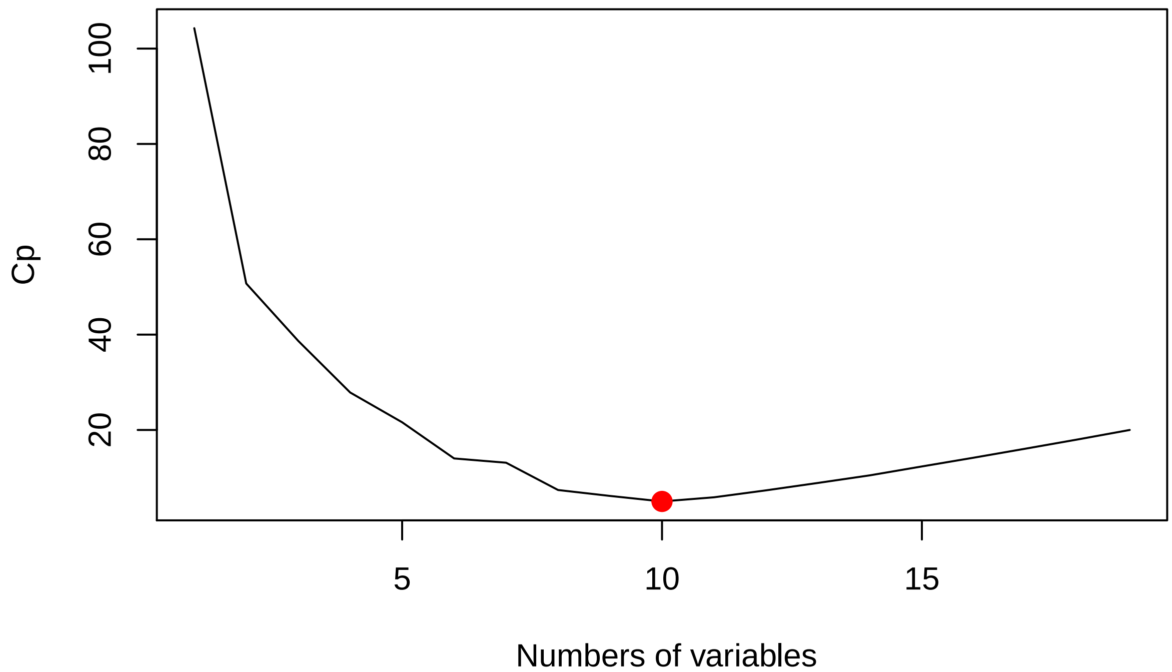
```r
points(index, summary$adjr2[index], col="red", cex=2, pch=20)
```




```r
# which.min() returns location minimum point of the vector
plot(summary$cp, xlab =" Numbers of variables", ylab="Cp", type="l")
(index <- which.min(summary$cp))
```

```
## [1] 10
```

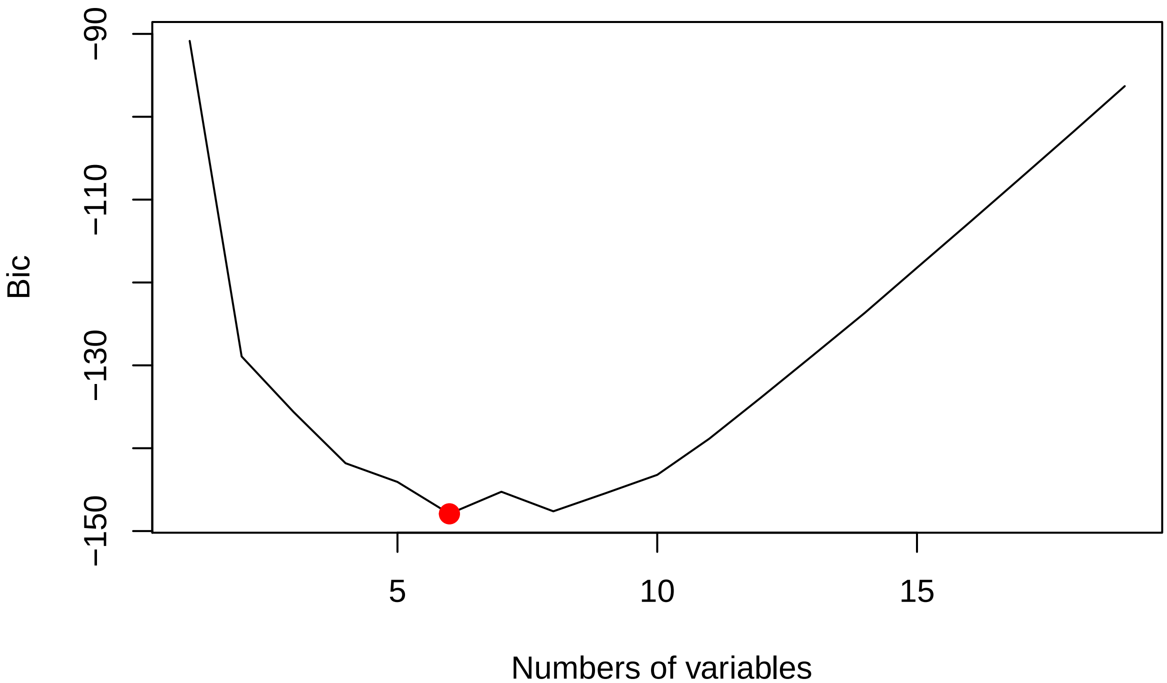
```r
points(index, summary$cp[index], col="red", cex=2, pch=20)
```



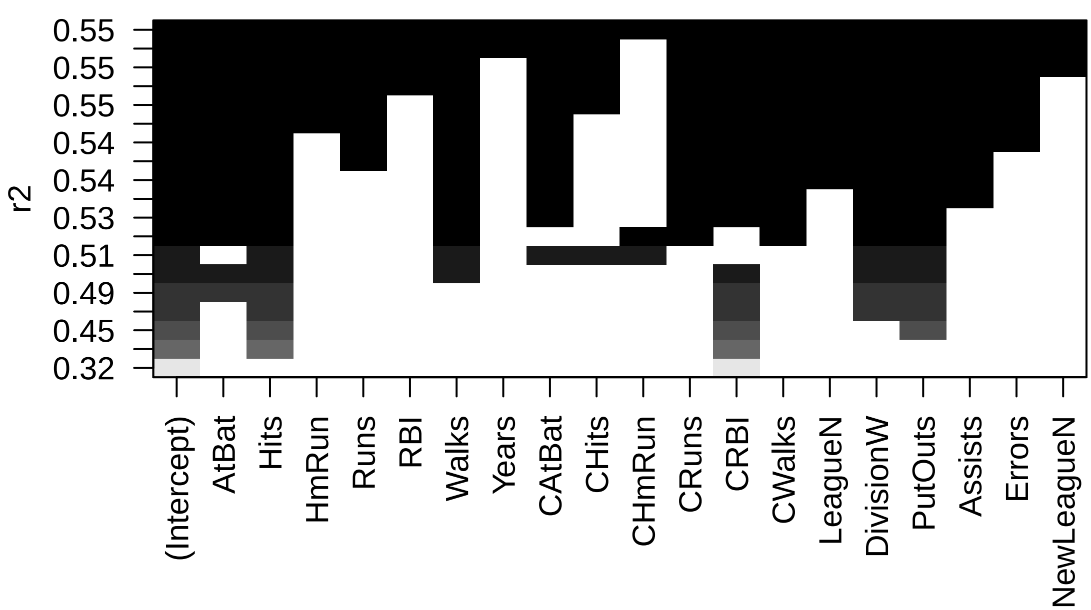

```r
# same for bic
plot(summary$bic, xlab =" Numbers of variables", ylab="Bic", type="l")
(index <- which.min(summary$bic))
```

```
## [1] 6
```

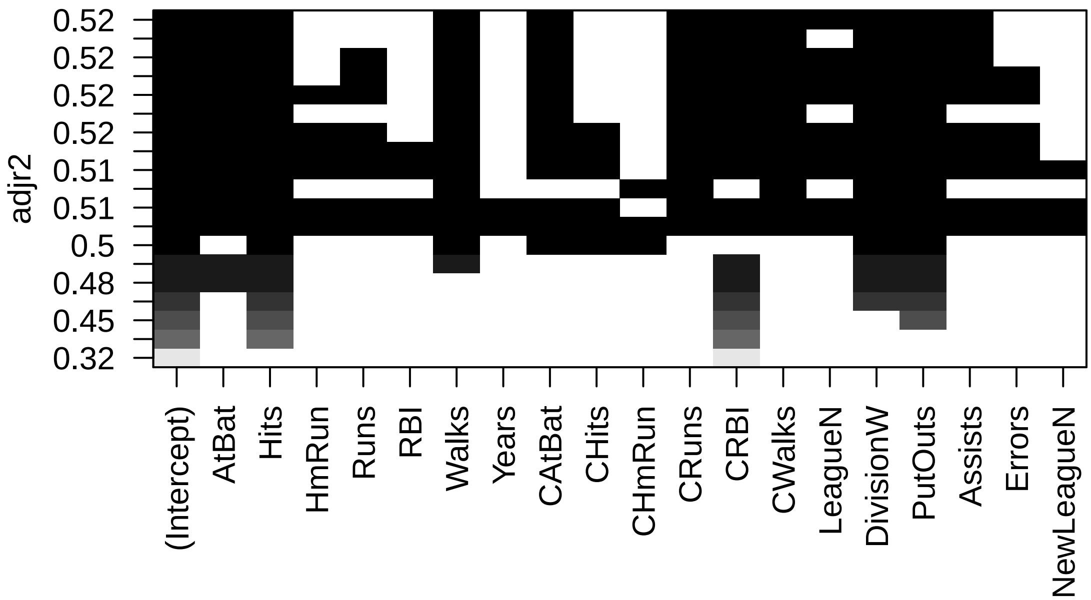
```r
points(index, summary$bic[index], col="red", cex=2, pch=20)
```



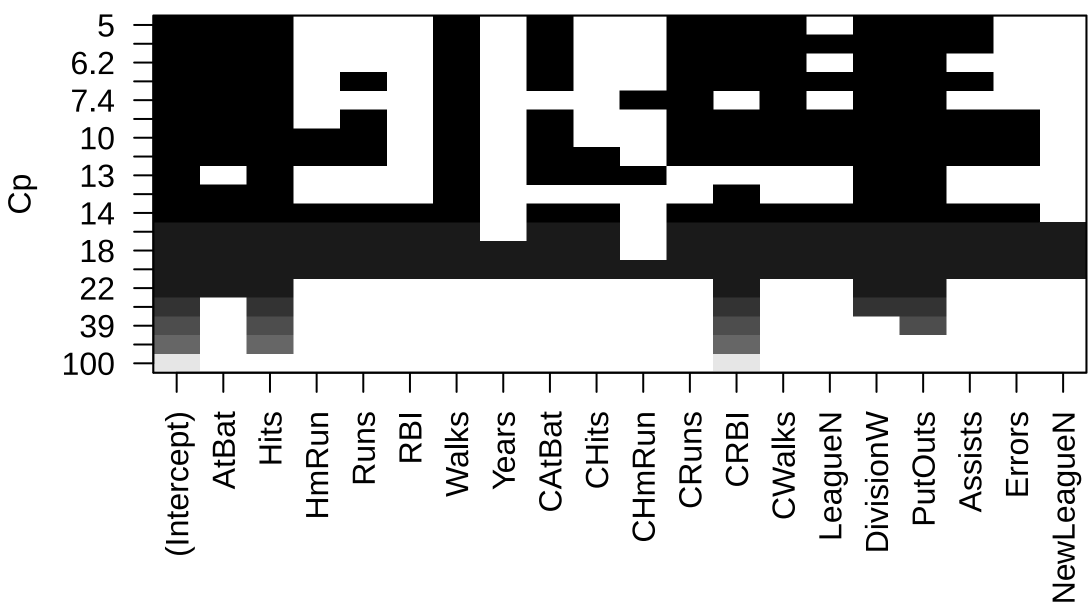

```r
# regsubsets() has builtin plot() command that displays selected variables for
# best model with a given number of predictors

plot(regfit.full,scale="r2")
```



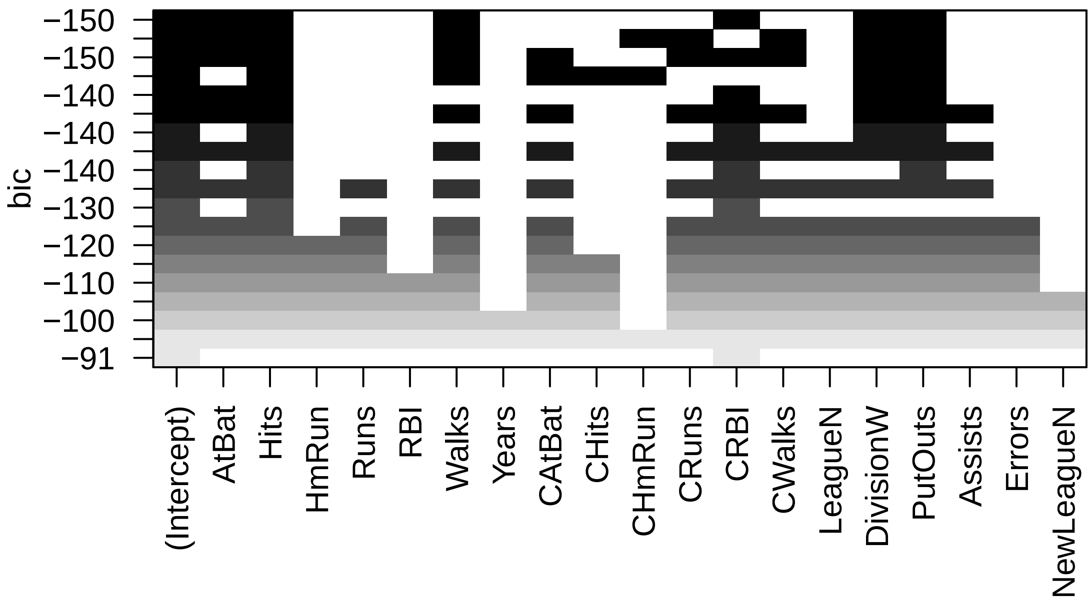

```r
plot(regfit.full,scale="adjr2")
```




```
plot(regfit.full,scale="Cp")
```




```
plot(regfit.full,scale="bic")
```




```
library(tidyverse)
library(leaps)
hitters.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Hitters.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

# note that Salary is od type string and some of them are NA
sum(hitters.df$Salary=="NA")
```

```
## [1] 59
```

```
# first remove character NAs
hitters.df <- hitters.df[hitters.df$Salary != "NA",]

# now convert Salary into numeric
hitters.df$Salary <- as.numeric(as.character(hitters.df$Salary))

# Now convert it to tibble
hitters.df <- tibble(hitters.df)

# we can use regsubsets() to perform forward / backward stepwise selection

regfit.fwd <- regsubsets(Salary ~ ., data = hitters.df, nvmax=ncol(hitters.df),
                         method = "forward")

summary(regfit.fwd)
```

```
## Subset selection object
## Call: regsubsets.formula(Salary ~ ., data = hitters.df, nvmax = ncol(hitters.df),
##     method = "forward")
## 19 Variables  (and intercept)
##            Forced in Forced out
## AtBat          FALSE      FALSE
## Hits           FALSE      FALSE
## HmRun          FALSE      FALSE
## Runs           FALSE      FALSE
```

```
## RBI            FALSE      FALSE
## Walks          FALSE      FALSE
## Years          FALSE      FALSE
## CAtBat         FALSE      FALSE
## CHits          FALSE      FALSE
## CHmRun         FALSE      FALSE
## CRuns          FALSE      FALSE
## CRBI           FALSE      FALSE
## CWalks         FALSE      FALSE
## LeagueN        FALSE      FALSE
## DivisionW      FALSE      FALSE
## PutOuts        FALSE      FALSE
## Assists        FALSE      FALSE
## Errors         FALSE      FALSE
## NewLeagueN     FALSE      FALSE
## 1 subsets of each size up to 19
## Selection Algorithm: forward
##           AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns CRBI
## 1  ( 1 )  " "   " "  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 2  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 3  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 4  ( 1 )  " "   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 5  ( 1 )  "*"   "*"  " "   " "  " " " "   " "   " "    " "   " "    " "   "*" 
## 6  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*" 
## 7  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    " "   "*" 
## 8  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   " "    " "   " "    "*"   "*" 
## 9  ( 1 )  "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*" 
## 10  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*" 
## 11  ( 1 ) "*"   "*"  " "   " "  " " "*"   " "   "*"    " "   " "    "*"   "*" 
## 12  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*" 
## 13  ( 1 ) "*"   "*"  " "   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*" 
## 14  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    " "   " "    "*"   "*" 
## 15  ( 1 ) "*"   "*"  "*"   "*"  " " "*"   " "   "*"    "*"   " "    "*"   "*" 
## 16  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*" 
## 17  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   " "   "*"    "*"   " "    "*"   "*" 
## 18  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   " "    "*"   "*" 
## 19  ( 1 ) "*"   "*"  "*"   "*"  "*" "*"   "*"   "*"    "*"   "*"    "*"   "*" 
##           CWalks LeagueN DivisionW PutOuts Assists Errors NewLeagueN
## 1  ( 1 )  " "    " "     " "       " "     " "     " "    " "       
## 2  ( 1 )  " "    " "     " "       " "     " "     " "    " "       
## 3  ( 1 )  " "    " "     " "       "*"     " "     " "    " "       
## 4  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "       
## 5  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "       
## 6  ( 1 )  " "    " "     "*"       "*"     " "     " "    " "       
## 7  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "       
## 8  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "       
## 9  ( 1 )  "*"    " "     "*"       "*"     " "     " "    " "       
## 10  ( 1 ) "*"    " "     "*"       "*"     "*"     " "    " "       
## 11  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "       
## 12  ( 1 ) "*"    "*"     "*"       "*"     "*"     " "    " "       
## 13  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "       
## 14  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "       
## 15  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "       
## 16  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    " "       
```

```
## 17  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 18  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
## 19  ( 1 ) "*"    "*"     "*"       "*"     "*"     "*"    "*"
```

```
# coefficient for the best model with 3 coefficients
(coefs <- coef(regfit.fwd,3))
```

```
## (Intercept)        Hits        CRBI     PutOuts
## -71.4592204   2.8038162   0.6825275   0.2735814
```

```
names(coefs)
```

```
## [1] "(Intercept)" "Hits"        "CRBI"        "PutOuts"
```

```
library(tidyverse)
library(leaps)
hitters.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Hitters.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

# note that Salary is od type string and some of them are NA
sum(hitters.df$Salary=="NA")
```

```
## [1] 59
```

```
# first remove character NAs
hitters.df <- hitters.df[hitters.df$Salary != "NA",]

# now convert Salary into numeric
hitters.df$Salary <- as.numeric(as.character(hitters.df$Salary))

# Now convert it to tibble
hitters.df <- tibble(hitters.df)
# first we create a vector that allocates each observation to one of K = 10 folds
k = 10
set.seed(1)

# create k folds
folds <- sample(1:k, nrow(hitters.df), replace = T)
# table(folds)
#  1  2  3  4  5  6  7  8  9 10
# 35 25 33 31 34 31 32 29 39 33

# number of features
noOfFeatures <- ncol(hitters.df) -1

# an empty accumulator to store MSE for each fold and each predictor
cv.errors <- matrix(NA, k, noOfFeatures,
                    dimnames = list(NULL, paste(1:noOfFeatures)))

# perform a cross validation on a for loop
for (j in 1:k){
  # step# 2 of algorithm is evaluated on all folds except one of them each time
  # it chooses best models with number of features 1,2,..., noOfFeatures
  # on k-1 training folds
  best.fit <- regsubsets(Salary ~ ., data = hitters.df[folds != j, ],
                         nvmax = noOfFeatures)
```

```r
  # Now build X matrix from test data
  test.mat <- model.matrix(Salary ~ ., data = hitters.df[folds == j, ])

#model.matrix:
# ------------
#(Intercept) AtBat Hits HmRun Runs RBI Walks Years CAtBat CHits CHmRun CRuns ...
#          1   202   53     4   31  26    27     9   1876   467     15   192 ...
#          1   239   60     0   30  11    22     6   1941   510      4   309 ...
#          1   472  116    16   60  62    74     6   1924   489     67   242 ...

  # now compute CV test error for each of  models that have best number of
  # predictors on test fold # j
  for(i in 1:noOfFeatures){
    # extract coefficients for model # i
    coefi <- coef(best.fit, id = i)

    # claculate cv test error for each row in test matrix()
    predicted_values <- test.mat [, names(coefi)] %*% coefi
    cv.errors[j,i] <- mean((predicted_values - hitters.df[folds ==j, ]$Salary)^2)
  }
}

# finally calculate mean of CV MSE error for each model
cv.error.means <- rep(NA, ncol(cv.errors))
for (l in 1:ncol(cv.errors)){
  cv.error.means[l] <- mean(cv.errors[ ,l])
}

# find the minimum of all cv-MSE means and corresponding coefficients
print("No of selected Features is the one with minimum of all cv-MSE means: ")
```

```
## [1] "No of selected Features is the one with minimum of all cv-MSE means: "
```

```r
(no.of.selected.features <- which.min(cv.error.means))
```

```
## [1] 10
```

```r
# finally get the slected columns (remeber it includes (Intercept) that has to be removed)
reg.best <- regsubsets(Salary ~ ., data = hitters.df,
                       nvmax = no.of.selected.features)
names(coef(best.fit, id = which.min(cv.error.means)))
```

```
##  [1] "(Intercept)" "AtBat"       "Hits"        "Walks"       "CAtBat"
##  [6] "CRuns"       "CRBI"        "CWalks"      "DivisionW"   "PutOuts"
## [11] "Assists"
```

```r
library(tidyverse)
library(class)
library(boot)
library(leaps)

set.seed(17)
weekly.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Weekly.csv",
                     header=T, stringsAsFactors = F, na.strings = "?")
weekly.df = tibble(weekly.df)
```

```r
#-------- Some usual cleaning on character columns ---------------- #

# First remove all recods with spaces in character column Direction
weekly.df$Direction <- gsub('\\s+', '', weekly.df$Direction)

# Second remove all leading and trailing spaces from a character column "Direction"
weekly.df$Direction <- trimws(weekly.df$Direction, which = c("both"))

# Remove all records with "NA" or empty string in character column "Direction"
weekly.df <- weekly.df[!(tolower(weekly.df$Direction) == "na" |
                           weekly.df$Direction == ""), ]

# convert all character fields
weekly.df[sapply(weekly.df, is.character)] <-
  lapply(weekly.df[sapply(weekly.df, is.character)], as.factor)

#----------- Find and remove NA in all columns ------------- #

weekly.df <- na.omit(weekly.df)

# ---------- stepwise forward feature selectin with CV -----------#

# create k-fold
k <- 10
threshold <- 0.5

set.seed(1)

# create k folds
folds <- sample(1:k, nrow(weekly.df), replace = T)

# For folds with same size do:
# sameSizefolds <- sample(rep(1:k, length.out = nrow(weekly.df)),
#                                         size = nrow(weekly.df), replace = F)
# table(sameSizefolds)


# number of features
noOfFeatures <- ncol(weekly.df) -1

# an empty accumulator to store MSE for each fold and each predictor
cv.errors <- matrix(NA, k, noOfFeatures,
                    dimnames = list(NULL, paste(1:noOfFeatures)))


# perform a cross validation on a for loop
for (j in 1:k){
  # step# 2 of algorithm 6.3 page 209 is evaluated on all folds except one of
  # them each time it chooses best models with number of features
  # 1,2,..., noOfFeatures on k-1 training folds.
  best.fit <- regsubsets(Direction ~ ., data = weekly.df[folds != j, ],
                         nvmax = noOfFeatures, method = "backward")
```

```r
  # Compute CV test error for each of  models that have best number of
  # predictors on test fold # j
  for(i in 1:noOfFeatures){

    # extract coefficients for model # i
    coefi <- coef(best.fit, id = i)

    # For classification we are interested in name
    # of features for model # i (not their coefficients)
    # except intercept
    predictorsOfModel <- names(coefi)[-1]
    # fit the model on k-1 trainimng portion
    lda.fit <- MASS::lda(as.formula(paste("Direction~", paste(predictorsOfModel, collapse="+"))),
                   data = weekly.df, family = binomial, subset = (folds != j))

    # predict on single validation fold
    lda.pred <- predict(lda.fit, weekly.df[folds == j, ], type =  "response")
    # since contrasts(weekly.df$Direction) shows dummy variable 1 asigned to 'Up'
    # and since P(y=1|x) is glm.probs what we get is prosterior of probability of 'Up' case
    stopifnot(length (lda.pred$class) == length(weekly.df[folds == j, ]$Direction))
    cv.errors[j,i] <- mean(lda.pred$class == weekly.df[folds == j, ]$Direction)
  }
}

# finally calculate mean of CV MSE error for each model
cv.error.means <- rep(NA, ncol(cv.errors))
for (i in 1:ncol(cv.errors)){
  cv.error.means[i] <- mean(cv.errors[, i])
}

# find the minimum of all cv-MSE means and corresponding coefficients
print("minimum of all cv-MSE means and corresponding coefficients: ")
```

```
## [1] "minimum of all cv-MSE means and corresponding coefficients: "
```

```r
(selected.no.of.features <- which.min(cv.error.means))
```

```
## [1] 4
```

```r
lda.best <- regsubsets(Direction ~ ., data = weekly.df[folds != j, ],
                       nvmax = selected.no.of.features, method = "backward")

# seems like model with following 5 predictors has least MSE error on test data
names(coef(lda.best, id = which.min(cv.error.means) ))
```

```
## [1] "(Intercept)" "Lag2"        "Lag3"        "Lag5"        "Today"
```

```r
library(tidyverse)
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
```

```
##     expand, pack, unpack
## Loaded glmnet 3.0-2
```

```r
hitters.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Hitters.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

# -------------- Usual clean up first ------------------#

# First remove all recods with spaces in character column Salary
hitters.df$Salary <- gsub('\\s+', '', hitters.df$Salary)

# Second remove all leading and trailing spaces from a character column "Salary"
hitters.df$Salary <- trimws(hitters.df$Salary, which = c("both"))

# Remove all records with "NA" or empty string in character column "Salary"
hitters.df <- hitters.df[!(tolower(hitters.df$Salary) == "na" |
                           hitters.df$Salary == ""), ]

# convert Salary column to numberic
hitters.df$Salary <- as.numeric(as.character(hitters.df$Salary))

# convert all character fields
hitters.df[sapply(hitters.df, is.character)] <-
  lapply(hitters.df[sapply(hitters.df, is.character)], as.factor)

# Find and remove NA in all columns
hitters.df <- na.omit(hitters.df)

# Use glmnet () for Ridge (glmnet only take numarical values)
# glmnet automatically standardize predictors unless we set standardize = F

# first create a matrix of all predictors
# model.matrix automatically transforms any qualitative variable to factor

x <- model.matrix(Salary~., hitters.df)[, -1]
y <- hitters.df$Salary

# apply Ridge
grid <- 10 ^ seq(10, -2, length = 100)
# alpha = 0 causes Ridge to be applied
ridge.model <- glmnet(x, y, alpha=0, lambda = grid)

# there are 100 values for lambda and associated to each we have
# number of ncol(hitters.df) coefficients

ridge.model$lambda[50]
```

```
## [1] 11497.57
```

```r
# get the coefficients corresponding to 50th lambda:
rownames(coef(ridge.model))
```

```
##  [1] "(Intercept)" "AtBat"       "Hits"        "HmRun"       "Runs"
##  [6] "RBI"         "Walks"       "Years"       "CAtBat"      "CHits"
```

```
## [11] "CHmRun"      "CRuns"       "CRBI"        "CWalks"      "LeagueN"
## [16] "DivisionW"   "PutOuts"     "Assists"     "Errors"      "NewLeagueN"
```

```r
coef(ridge.model)[,50]
```

```
##   (Intercept)         AtBat          Hits         HmRun          Runs
## 407.356050200   0.036957182   0.138180344   0.524629976   0.230701523
##           RBI         Walks         Years        CAtBat         CHits
##   0.239841459   0.289618741   1.107702929   0.003131815   0.011653637
##        CHmRun         CRuns          CRBI        CWalks       LeagueN
##   0.087545670   0.023379882   0.024138320   0.025015421   0.085028114
##     DivisionW       PutOuts       Assists        Errors    NewLeagueN
##  -6.215440973   0.016482577   0.002612988  -0.020502690   0.301433531
```

```r
# foe some reason there are two intercept coefficients at the begining
# we drop first one to calculate L2 norm of the coefficints
sqrt(sum(coef(ridge.model)[-1,50]^2))
```

```
## [1] 6.360612
```

```r
# we use predict to get a new value for coefficients for any given value of lambda
predict(ridge.model, s = 51, type="coefficients")[1:20]
```

```
##  [1]  4.784128e+01 -3.496519e-01  1.949106e+00 -1.267814e+00  1.147840e+00
##  [6]  8.055626e-01  2.698472e+00 -6.123000e+00  5.606739e-03  1.056868e-01
## [11]  6.221438e-01  2.195339e-01  2.174176e-01 -1.464445e-01  4.567755e+01
## [16] -1.180038e+02  2.497163e-01  1.201684e-01 -3.262943e+00 -9.218087e+00
```

```r
# now split the samples into test and training:
set.seed(10)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
y.test <- y[test]

# Fit ridge regression model to trainig data
ridge.model <- glmnet(x[train, ], y[train], alpha=0,
                      lambda = ( 10 ^ seq(10, -2, length = 100)),
                      thresh = 1e-12)

# Evaluate MSE of the model on on the test set for lambda = 4
# to do the prediction we set newx argument to test set
ridge.predict <- predict(ridge.model, s=1000, newx = x[test,])

print ("now find the MSE corresponding to lambda = 4")
```

```
## [1] "now find the MSE corresponding to lambda = 4"
```

```r
mean((ridge.predict-y.test)^2)
```

```
## [1] 139172.7
```

```r
print ("just for comarison we use intercept to predict and calculate the MSE
       (lambda is set to a very large value)")
```

```
## [1] "just for comarison we use intercept to predict and calculate the MSE\n       (lambda is set to a
```

```r
ridge.predict <- predict(ridge.model, s=1e+10, newx = x[test,])

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

```
## [1] 202640.1
```

```r
print ("now compare Ridge with usuall regression (i.e when lambda = 0)")
```

```
## [1] "now compare Ridge with usuall regression (i.e when lambda = 0)"
```

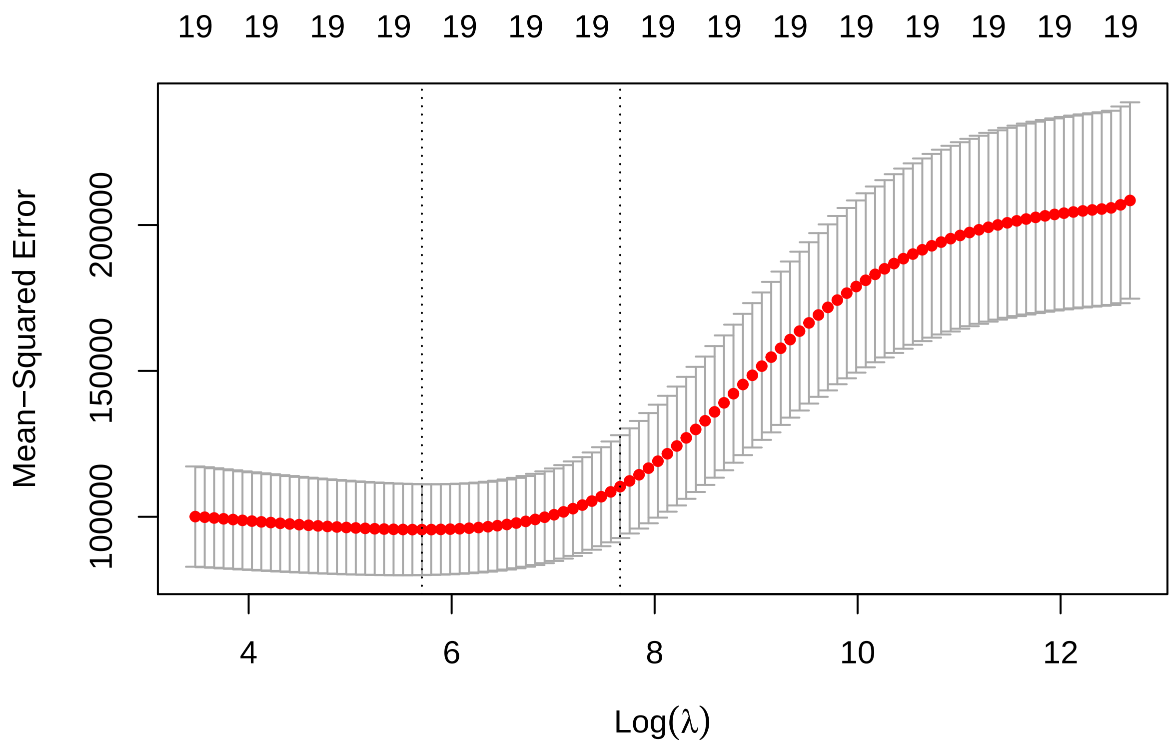
```r
# we set exact to T otherwise predict() function will
# interpolate over the grid of lambda values
ridge.predict <- predict(ridge.model, s=0, newx = x[test,],
                         exact = T, x = x[train,], y=y[train])

# calculate MSE
mean((ridge.predict - y.test)^2)
```

```
## [1] 145023.6
```

```r
# if we want to fit a (unpenalized) least squares model, then we should use the
# lm() function, since that function provides more useful outputs, such as
# standard errors and p-values for the coefficients.
```

```r
library(tidyverse)
library(glmnet)

hitters.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Hitters.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

# -------------- Usual clean up first ------------------#

# First remove all recods with spaces in character column Salary
hitters.df$Salary <- gsub('\\s+', '', hitters.df$Salary)

# Second remove all leading and trailing spaces from a character column "Salary"
hitters.df$Salary <- trimws(hitters.df$Salary, which = c("both"))

# Remove all records with "NA" or empty string in character column "Salary"
hitters.df <- hitters.df[!(tolower(hitters.df$Salary) == "na" |
                           hitters.df$Salary == ""), ]

# convert Salary column to numeric
hitters.df$Salary <- as.numeric(as.character(hitters.df$Salary))

# convert all character fields to factor
hitters.df[sapply(hitters.df, is.character)] <-
  lapply(hitters.df[sapply(hitters.df, is.character)], as.factor)

# Find and remove NA in all columns
hitters.df <- na.omit(hitters.df)


x <- model.matrix(Salary~., hitters.df)[, -1] # -1 is to drop the Intercept
y <- hitters.df$Salary

set.seed(10)
train <- sample(1:nrow(x), nrow(x)/2)
test <- (-train)
```

```r
# Built-in cross-validation function, cv.glmnet().
# By default, the function performs ten-fold cross-validation,
# though this can be changed using the argument nfolds.
# we aaply ot on training portion of the data to find the lambda
# then we run the final model with the lamda on test data to get MSE

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



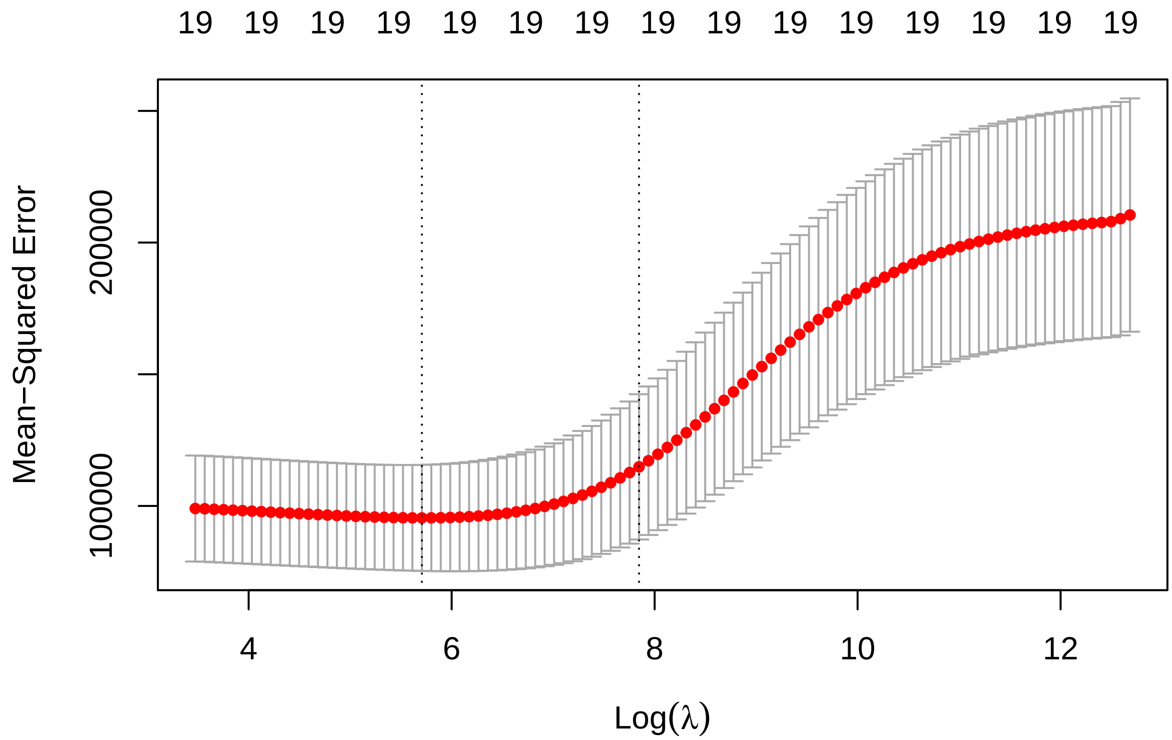

```r
(bestlam=cv.out$lambda.min)
```

```
## [1] 300.8959
```

```r
# What is the test MSE associated with this value of lambda?

# Fit ridge regression model to trainig data
ridge.model <- glmnet(x[train, ], y[train], alpha=0,
                      lambda = ( 10 ^ seq(10, -2, length = 100)),
                      thresh = 1e-12)

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

```
## [1] 143253.2
```

```r
# now let's split the data into train / test and train
# the model on train data , get the best lambda and then run it on test portion
# with the best lambda we got to see the testMSE:

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




```
(best.lambda <- cv.out$lambda.min)
```

```
## [1] 300.8959
```

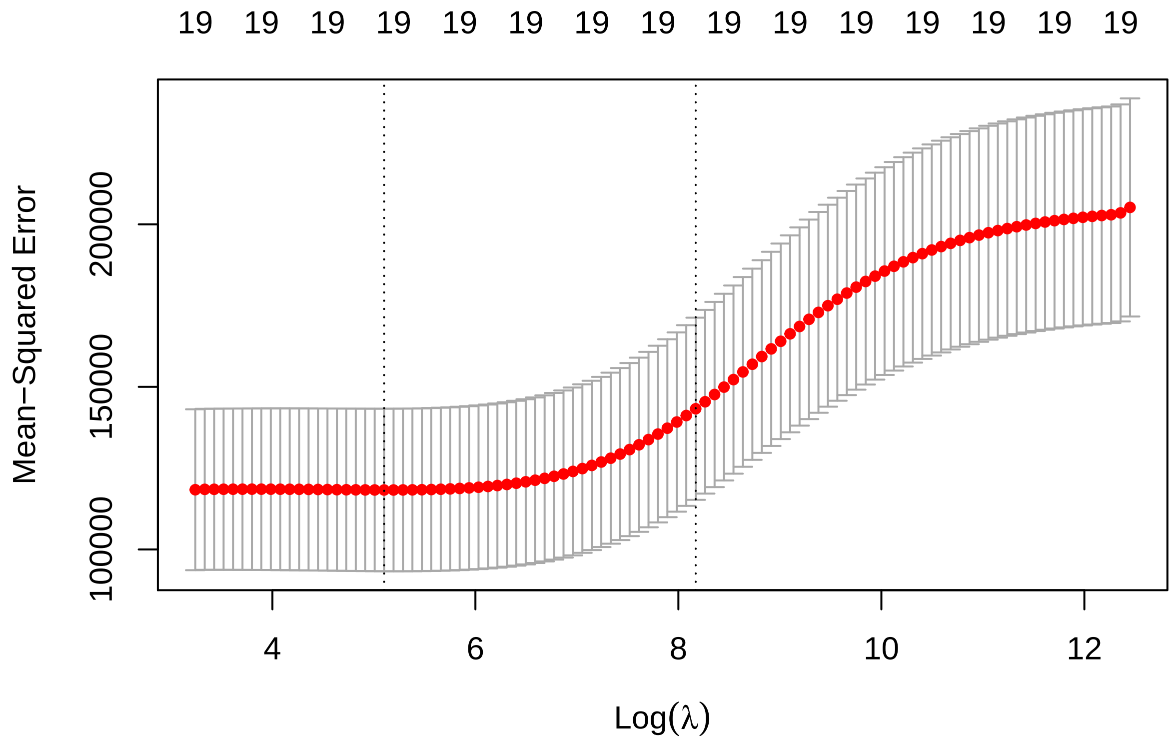
```
ridge.pred <- predict(cv.out, s=bestlam, newx = x[test, ])
sprintf("mean error on test data using %s is %s", bestlam, mean((ridge.pred - y[test])^ 2))
```

```
## [1] "mean error on test data using 300.895860794707 is 143257.455545951"
```

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

```
## [1] 143257.5
```

```
# finally we refit the model on the whole data and use the best lambda calculated
# in CV
cv.out=cv.glmnet(x,y,alpha=0)
plot(cv.out)
```




```r
predict(cv.out, type="coefficients" ,s=bestlam )
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  13.60555790
## AtBat         0.07035200
## Hits          0.88449492
## HmRun         0.52082363
## Runs          1.07391429
## RBI           0.87905669
## Walks         1.65668620
## Years         1.15648560
## CAtBat        0.01133745
## CHits         0.05871840
## CHmRun        0.41453959
## CRuns         0.11702038
## CRBI          0.12398388
## CWalks        0.04940713
## LeagueN      23.01892904
## DivisionW   -81.49794763
## PutOuts       0.17107771
## Assists       0.03150566
## Errors       -1.44191373
## NewLeagueN    8.90014313
```

```r
library(tidyverse)
library(glmnet)

weekly.df =
  read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Weekly.csv",
                     header=T, stringsAsFactors = F, na.strings = "?")
```

```r
str(weekly.df)
```

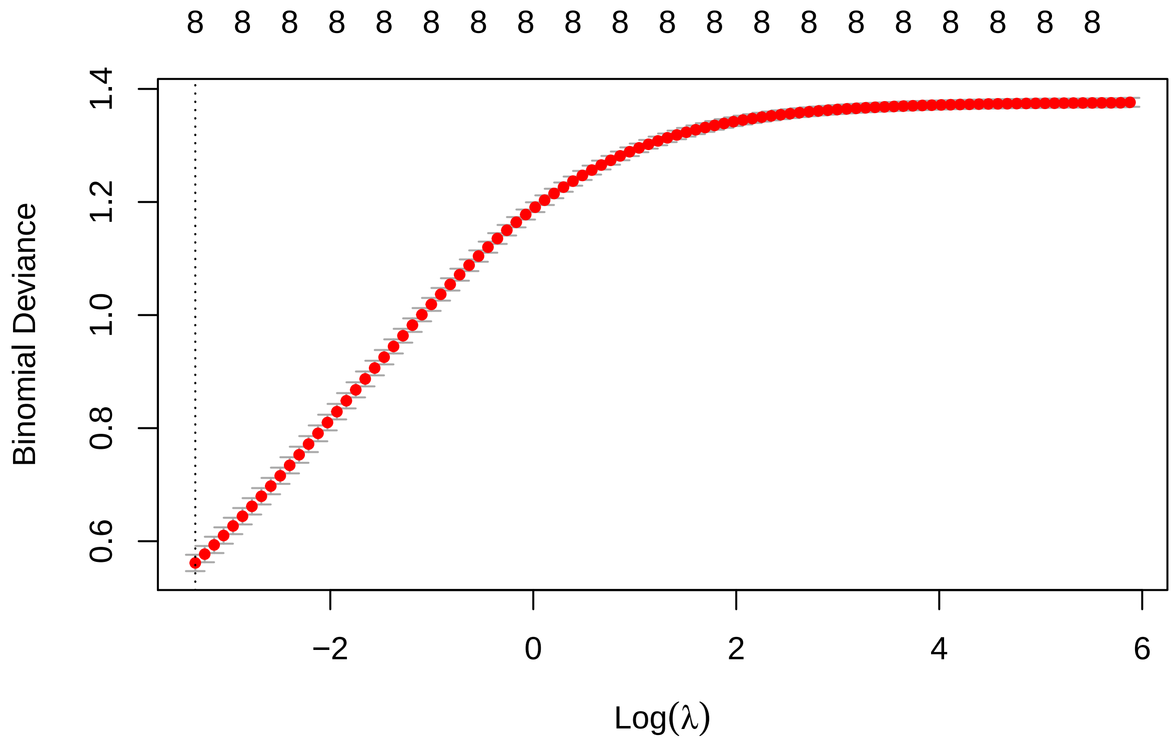
```
## 'data.frame':    1089 obs. of  9 variables:
##  $ Year     : int  1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
##  $ Lag1     : num  0.816 -0.27 -2.576 3.514 0.712 ...
##  $ Lag2     : num  1.572 0.816 -0.27 -2.576 3.514 ...
##  $ Lag3     : num  -3.936 1.572 0.816 -0.27 -2.576 ...
##  $ Lag4     : num  -0.229 -3.936 1.572 0.816 -0.27 ...
##  $ Lag5     : num  -3.484 -0.229 -3.936 1.572 0.816 ...
##  $ Volume   : num  0.155 0.149 0.16 0.162 0.154 ...
##  $ Today    : num  -0.27 -2.576 3.514 0.712 1.178 ...
##  $ Direction: chr  "Down" "Down" "Up" "Up" ...
```

```r
# -------------- Usual clean up first ------------------#

# First remove all recods with spaces in character column Direction
weekly.df$Direction <- gsub('\\s+', '', weekly.df$Direction)

# Second remove all leading and trailing spaces from a character column "Direction"
weekly.df$Direction <- trimws(weekly.df$Direction, which = c("both"))

# Remove all records with "NA" or empty string in character column "Direction"
weekly.df <- weekly.df[!(tolower(weekly.df$Direction) == "na" |
                           weekly.df$Direction == ""), ]

# convert all character fields to factor
weekly.df[sapply(weekly.df, is.character)] <-
 lapply(weekly.df[sapply(weekly.df, is.character)], as.factor)

# Find and remove NA in all columns
weekly.df <- na.omit(weekly.df)

str(weekly.df)
```

```
## 'data.frame':    1089 obs. of  9 variables:
##  $ Year     : int  1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
##  $ Lag1     : num  0.816 -0.27 -2.576 3.514 0.712 ...
##  $ Lag2     : num  1.572 0.816 -0.27 -2.576 3.514 ...
##  $ Lag3     : num  -3.936 1.572 0.816 -0.27 -2.576 ...
##  $ Lag4     : num  -0.229 -3.936 1.572 0.816 -0.27 ...
##  $ Lag5     : num  -3.484 -0.229 -3.936 1.572 0.816 ...
##  $ Volume   : num  0.155 0.149 0.16 0.162 0.154 ...
##  $ Today    : num  -0.27 -2.576 3.514 0.712 1.178 ...
##  $ Direction: Factor w/ 2 levels "Down","Up": 1 1 2 2 2 1 2 2 2 1 ...
```

```r
# contrasts(weekly.df$Direction)

# now we use cross validation to find the best lambda and corresponding coeffs

# First construct matrix from dataframe (and drop intercept column)
x <- model.matrix(Direction~., weekly.df)[,-1]
y <- ifelse(weekly.df$Direction == "Up", 1, 0)

set.seed(10)
```

```r
# We'll use the R function glmnet() [glmnet package] for computing penalized logistic regression.

# Built-in cross-validation function, cv.glmnet().
# By default, the function performs ten-fold cross-validation,
# though this can be changed using the argument nfolds.

cv.out=cv.glmnet(x, y, family = "binomial", alpha=0, lambda = NULL)
plot(cv.out)
```




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

```
## [1] 0.03577832
```

```r
# get the coefficients:
predict(cv.out, type="coefficients" ,s=bestlam )
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  9.238107856
## Year        -0.004539565
## Lag1        -0.011377731
## Lag2         0.036049566
## Lag3        -0.007947419
## Lag4        -0.021976657
## Lag5        -0.012921371
## Volume       0.012663021
## Today        0.966650674
```

```r
print("Here is value of lambda for which the MSE is minimum")
```

```
## [1] "Here is value of lambda for which the MSE is minimum"
```

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

```
## [1] 0.03577832
```

```r
print("Here is one standard error value of lambda for which the MSE is minimum")
```

```
## [1] "Here is one standard error value of lambda for which the MSE is minimum"
```

```r
cv.out$lambda.1se
```

```
## [1] 0.03577832
```

```r
library(tidyverse)
library(glmnet)

weekly.df =
  read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Weekly.csv",
                     header=T, stringsAsFactors = F, na.strings = "?")

str(weekly.df)
```

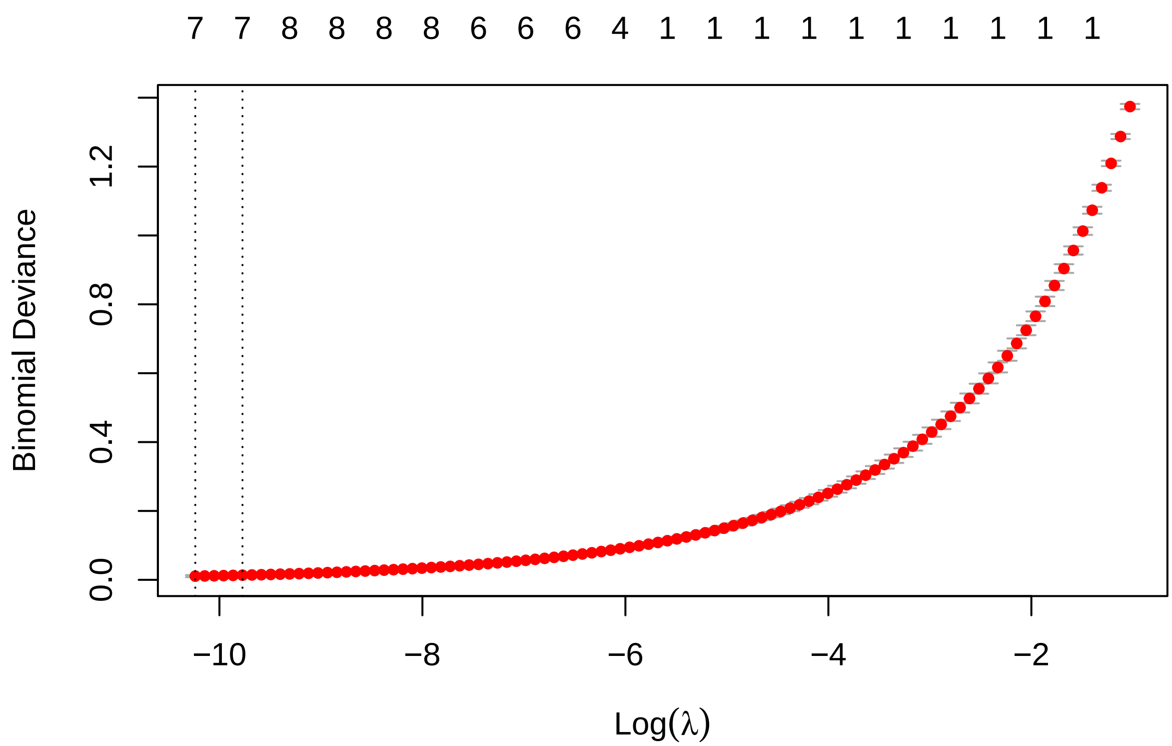
```
## 'data.frame':    1089 obs. of  9 variables:
##  $ Year     : int  1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
##  $ Lag1     : num  0.816 -0.27 -2.576 3.514 0.712 ...
##  $ Lag2     : num  1.572 0.816 -0.27 -2.576 3.514 ...
##  $ Lag3     : num  -3.936 1.572 0.816 -0.27 -2.576 ...
##  $ Lag4     : num  -0.229 -3.936 1.572 0.816 -0.27 ...
##  $ Lag5     : num  -3.484 -0.229 -3.936 1.572 0.816 ...
##  $ Volume   : num  0.155 0.149 0.16 0.162 0.154 ...
##  $ Today    : num  -0.27 -2.576 3.514 0.712 1.178 ...
##  $ Direction: chr  "Down" "Down" "Up" "Up" ...
```

```r
# -------------- Usual clean up first ------------------#

# First remove all recods with spaces in character column Direction
weekly.df$Direction <- gsub('\\s+', '', weekly.df$Direction)

# Second remove all leading and trailing spaces from a character column "Direction"
weekly.df$Direction <- trimws(weekly.df$Direction, which = c("both"))

# Remove all records with "NA" or empty string in character column "Direction"
weekly.df <- weekly.df[!(tolower(weekly.df$Direction) == "na" |
                           weekly.df$Direction == ""), ]

# convert all character fields to factor
weekly.df[sapply(weekly.df, is.character)] <-
 lapply(weekly.df[sapply(weekly.df, is.character)], as.factor)

# Find and remove NA in all columns
weekly.df <- na.omit(weekly.df)

str(weekly.df)
```

```
## 'data.frame':    1089 obs. of  9 variables:
##  $ Year     : int  1990 1990 1990 1990 1990 1990 1990 1990 1990 1990 ...
##  $ Lag1     : num  0.816 -0.27 -2.576 3.514 0.712 ...
##  $ Lag2     : num  1.572 0.816 -0.27 -2.576 3.514 ...
```

```
##  $ Lag3     : num  -3.936 1.572 0.816 -0.27 -2.576 ...
##  $ Lag4     : num  -0.229 -3.936 1.572 0.816 -0.27 ...
##  $ Lag5     : num  -3.484 -0.229 -3.936 1.572 0.816 ...
##  $ Volume   : num  0.155 0.149 0.16 0.162 0.154 ...
##  $ Today    : num  -0.27 -2.576 3.514 0.712 1.178 ...
##  $ Direction: Factor w/ 2 levels "Down","Up": 1 1 2 2 2 1 2 2 2 1 ...
```

```
# contrasts(weekly.df$Direction)

# now we use cross validation to find the best lambda and corresponding coeffs

# First construct matrix from dataframe (and drop intercept column)
x <- model.matrix(Direction~., weekly.df)[,-1]
y <- ifelse(weekly.df$Direction == "Up", 1, 0)

set.seed(10)

# We'll use the R function glmnet() [glmnet package]
# for computing penalized logistic regression.

# Built-in cross-validation function, cv.glmnet().
# By default, the function performs ten-fold cross-validation,
# though this can be changed using the argument nfolds.

cv.out=cv.glmnet(x, y, family = "binomial", alpha=1, lambda = NULL)
plot(cv.out)
```




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

```
## [1] 3.577832e-05
```

```r
# gget the coefficients:
predict(cv.out, type="coefficients" ,s=bestlam )
```

```
## 9 x 1 sparse Matrix of class "dgCMatrix"
##                         1
## (Intercept) -57.08275304
## Year          0.02827835
## Lag1         -0.61384069
## Lag2          0.06008024
## Lag3          0.28917265
## Lag4          .
## Lag5          0.47365956
## Volume        0.22529278
## Today        61.60602428
```

```r
names(cv.out)
```

```
##  [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"
##  [6] "nzero"      "call"       "name"       "glmnet.fit" "lambda.min"
## [11] "lambda.1se"
```

```r
print("Here is value of lambda for which the MSE is minimum")
```

```
## [1] "Here is value of lambda for which the MSE is minimum"
```

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

```
## [1] 3.577832e-05
```

```r
print("Here is one standard error value of lambda for which the MSE is minimum")
```

```
## [1] "Here is one standard error value of lambda for which the MSE is minimum"
```

```r
cv.out$lambda.1se
```

```
## [1] 5.69692e-05
```

```r
library(tidyverse)
library(pls)
```

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

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

```r
hitters.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Hitters.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

# -------------- Usual clean up first ------------------#

# First remove all recods with spaces in character column Salary
hitters.df$Salary <- gsub('\\s+', '', hitters.df$Salary)

# Second remove all leading and trailing spaces from a character column "Salary"
hitters.df$Salary <- trimws(hitters.df$Salary, which = c("both"))
```

```r
# Remove all records with "NA" or empty string in character column "Salary"
hitters.df <- hitters.df[!(tolower(hitters.df$Salary) == "na" |
                           hitters.df$Salary == ""), ]

# convert Salary column to numberic
hitters.df$Salary <- as.numeric(as.character(hitters.df$Salary))

# convert all character fields
hitters.df[sapply(hitters.df, is.character)] <-
  lapply(hitters.df[sapply(hitters.df, is.character)], as.factor)

# Find and remove NA in all columns
hitters.df <- na.omit(hitters.df)

set.seed(2)

# fit the model using pcr
pcr.fit <- pcr(Salary ~ ., data = hitters.df, scale = T, validation="CV")

print ("Summary:")
```

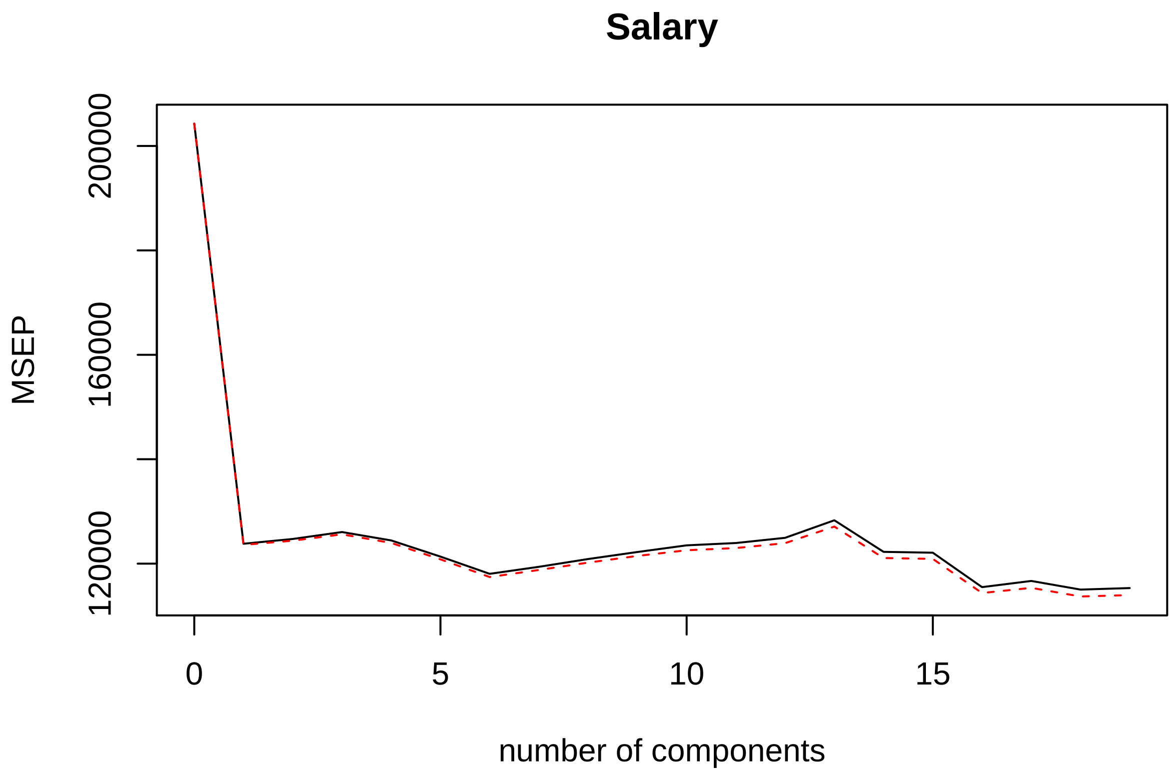
```
## [1] "Summary:"
```

```r
summary(pcr.fit)
```

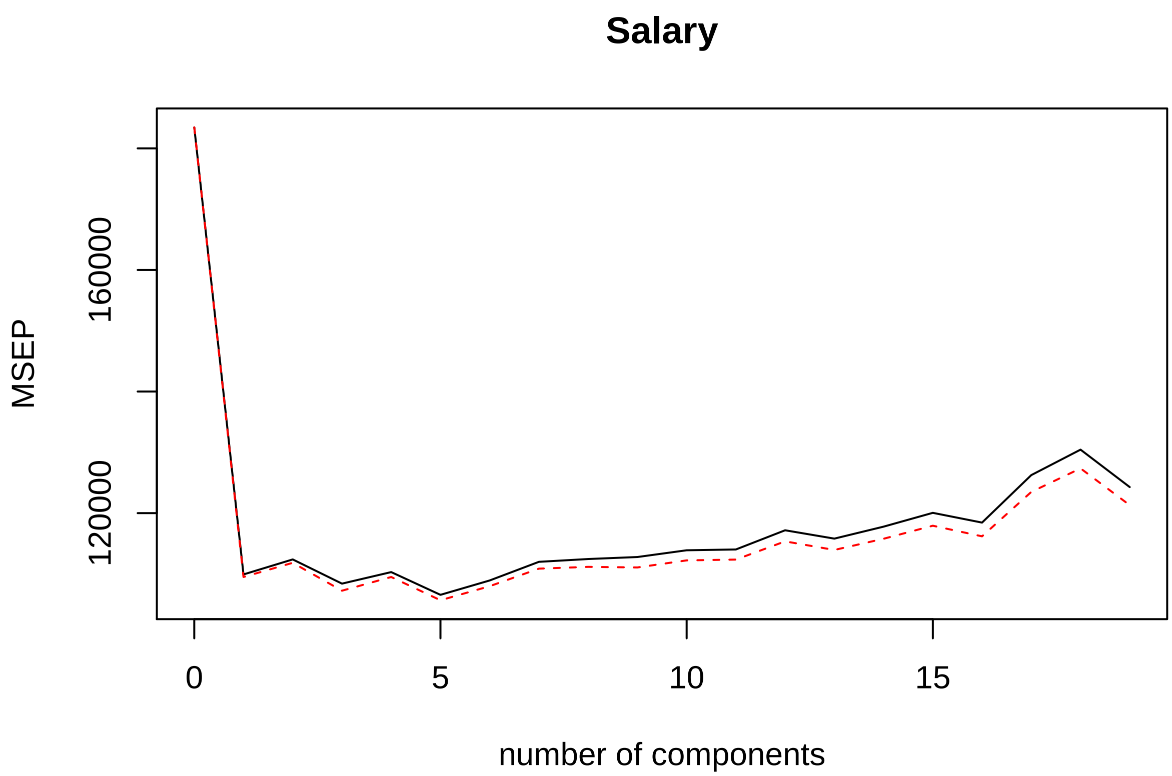
```
## Data:    X dimension: 263 19
##  Y dimension: 263 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV             452    351.9    353.2    355.0    352.8    348.4    343.6
## adjCV          452    351.6    352.7    354.4    352.1    347.6    342.7
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       345.5    347.7    349.6     351.4     352.1     353.5     358.2
## adjCV    344.7    346.7    348.5     350.1     350.7     352.0     356.5
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        349.7     349.4     339.9     341.6     339.2     339.6
## adjCV     348.0     347.7     338.2     339.7     337.2     337.6
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         38.31    60.16    70.84    79.03    84.29    88.63    92.26    94.96
## Salary    40.63    41.58    42.17    43.22    44.90    46.48    46.69    46.75
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         96.28     97.26     97.98     98.65     99.15     99.47     99.75
## Salary    46.86     47.76     47.82     47.85     48.10     50.40     50.55
##         16 comps  17 comps  18 comps  19 comps
## X          99.89     99.97     99.99    100.00
## Salary     53.01     53.85     54.61     54.61
```

```r
# Note that although the minimum value of RSME is associated with
# M = 16 (which is very close to 19) but for M = 7 we get a drastic
```

```r
# decrease in RMSE which is very close to that of M = 17
# This suggsts M=7 gives us good enough model
validationplot(pcr.fit, val.type = "MSEP")
```




```r
# Now to see how model with 7 works on test data an compare it with
# model with M = 17 we split data into test and train and fit the model
# on train

set.seed(1)
train <- train <- sample(1:nrow(hitters.df), nrow(hitters.df)/2)
test <- (-train)
pcr.fit <- pcr(Salary ~ ., data=hitters.df, subset=train, scale=T, validation="CV")
validationplot(pcr.fit, val.type = "MSEP")
```




```
# now let's find the lowest cross validation error occurs M = 7 on the  model
test.y <- hitters.df[test,]$Salary
pcr.pred <- predict(pcr.fit, hitters.df[test,], ncomp = 7)

sprintf("lowest MSE corresponding to M = 7 is %s (Ridge was 143257.45)",mean((pcr.pred-test.y)^2))
```

```
## [1] "lowest MSE corresponding to M = 7 is 140751.276313081 (Ridge was 143257.45)"
```

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

```
## [1] 140751.3
```

```
library(tidyverse)
library(pls)

weekly.df =
  read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Weekly.csv",
                     header=T, stringsAsFactors = F, na.strings = "?")
# -------------- Usual clean up first ------------------#

# First remove all recods with spaces in character column Direction
weekly.df$Direction <- gsub('\\s+', '', weekly.df$Direction)

# Second remove all leading and trailing spaces from a character column "Direction"
weekly.df$Direction <- trimws(weekly.df$Direction, which = c("both"))

# Remove all records with "NA" or empty string in character column "Direction"
weekly.df <- weekly.df[!(tolower(weekly.df$Direction) == "na" |
                           weekly.df$Direction == ""), ]

# convert all character fields to factor
```

```r
weekly.df[sapply(weekly.df, is.character)] <-
 lapply(weekly.df[sapply(weekly.df, is.character)], as.factor)

# Find and remove NA in all columns
weekly.df <- na.omit(weekly.df)

# -------------------  outmost loop must be CV loop ---------------------
# Note that as per "Tibshirani" cross validation must always be before
# dimension reduction or feature selection
# ----------------------------------------------------------------------
set.seed(1)
k <- 10
threshold <- 0.5

folds <- sample(1:k, size = nrow(weekly.df), replace = T)
table(folds)
```

```
## folds
##   1   2   3   4   5   6   7   8   9  10
## 104  86 106 112 116 107 121 102 117 118
```

```r
# folds with same size
# sameSizefolds <- sample(rep(1:k, length.out = nrow(weekly.df)),
#   size = nrow(weekly.df), replace = F)
# table(sameSizefolds)

results <- lapply (1:k, function(x) {  # x is the index of test portion, the rest are for training

  #------------- calculate all PCA scores on k-1 trining fold -----------------
  # First create predictor matrix and response vector on test and train folds
  predictor.train.X <- model.matrix(Direction~., weekly.df[folds != x,])[, -1]
  response.train.Y <- weekly.df[folds != x,]$Direction

  # use PCA to find principal components on k-1 trining fold:
  pca.train <- princomp(predictor.train.X, cor=T) # PCA using correlation matrix

  # make a tibble from PCAs
  weekly.pca.train <- as_tibble(pca.train$scores[,]*-1) %>%
    add_column(Direction=response.train.Y)

  # Do exact same thing for test fold:
  predictor.test.X <- model.matrix(Direction~., weekly.df[folds == x,])[, -1]
  response.test.Y <- weekly.df[folds == x,]$Direction

  # use PCA to find principal components on test fold:
  pca.test <- princomp(predictor.test.X, cor=T) # PCA using correlation matrix

  # make a tibble from PCAs
  weekly.pca.test <- as_tibble(pca.test$scores[,]*-1) %>%
    add_column(Direction=response.test.Y)

  # get list of all pca column names
  pca.cols <- colnames(weekly.pca.train[ , !(names(weekly.pca.train) %in% c("Direction"))])
```

```
  # An empty tibble to collect all result of running LDA on given test fold
  pca.results <- tibble (no.of.pcas = NULL,
                    posterior.up = NULL,
                    posterior.down = NULL,
                    predicted = NULL,
                    real = NULL)

  for (pc.col.idx in 1:length(pca.cols)){

    pca.chosen.cols <- pca.cols[1:pc.col.idx]
    partial.weekly.pca.train <- weekly.pca.train[ ,pca.chosen.cols] %>%
      add_column(Direction=response.train.Y)

    # fit on train fold
    lda.fit <-
      MASS::lda(as.formula(paste("Direction~",
                                 paste(pca.chosen.cols, collapse="+"))),
                data = partial.weekly.pca.train, family = binomial)

    # predict on test fold
    lda.pred <- predict(lda.fit, weekly.pca.test, type =  "response")
    stopifnot(length (lda.pred$class) == length(weekly.pca.test$Direction))
    pca.results <- rbind (pca.results, tibble (no.of.pcas = pc.col.idx,
                    posterior.up = lda.pred$posterior[, "Up"],
                    posterior.down = lda.pred$posterior[, "Down"],
                    predicted = lda.pred$class,
                    real = weekly.pca.test$Direction))
  }
  return (pca.results)
})

# We have to find average of missclassification rate for each number of PCs
# cross all test folds

# first calculte missclassification rate per each number of PCAs


rates <- lapply(results, function (result) {
  return(result %>%
         group_by(no.of.pcas) %>%
         summarise(MSE = mean(predicted != real),
          FP_rates = table(predicted, real)[2,1]/(table(predicted, real)[2,1]+ table(predicted, real)
          TP_rates = table(predicted, real)[2,2]/(table(predicted, real)[2,2]+ table(predicted, real)
          precisions = table(predicted, real)[2,2]/(table(predicted, real)[2,2]+ table(predicted, rea
          specificities = table(predicted, real)[2,1]/(table(predicted, real)[2,1]+ table(predicted, 
          nullClassifier = max( ( table(predicted, real)[1,1] + table(predicted, real)[2,1])/
                                  (table(predicted, real)[1,1] + table(predicted, real)[2,1] +
                                     table(predicted, real)[1,2] + table(predicted, real)[2,2]),
                                table(predicted, real)[1,2] + table(predicted, real)[2,2])/
                                  (table(predicted, real)[1,1] + table(predicted, real)[2,1] +
                                     table(predicted, real)[1,2] + table(predicted, real)[2,2])
          ))
})
```

```r
# Place rates for all folds in one df
(all.rates = do.call(rbind, rates))
```

```
## # A tibble: 80 x 7
##    no.of.pcas   MSE FP_rates TP_rates precisions specificities nullClassifier
##         <int> <dbl>    <dbl>    <dbl>      <dbl>         <dbl>          <dbl>
##  1          1 0.481    1        0.947      0.535         1              0.548
##  2          2 0.606    0.872    0.614      0.461         0.872          0.548
##  3          3 0.740    0.915    0.404      0.348         0.915          0.548
##  4          4 0.760    0.872    0.333      0.317         0.872          0.548
##  5          5 0.827    0.936    0.263      0.254         0.936          0.548
##  6          6 0.817    0.936    0.281      0.267         0.936          0.548
##  7          7 0.904    1        0.175      0.175         1              0.548
##  8          8 0.904    1        0.175      0.175         1              0.548
##  9          1 0.395    0.939    0.943      0.617         0.939          0.616
## 10          2 0.558    0.818    0.604      0.542         0.818          0.616
## # ... with 70 more rows
```

```r
# get mean of rates cross all folds per each mumber of pcs (1, 2, ..., 8)
(all.rates %>%
  group_by(no.of.pcas) %>%
  summarise(MSE = mean(MSE),
            FP_rates = mean(FP_rates),
            TP_rates = mean(TP_rates),
            precisions = mean(precisions),
            specificities = mean(specificities),
            nullClassifier = mean(nullClassifier))
)
```

```
## # A tibble: 8 x 7
##   no.of.pcas   MSE FP_rates TP_rates precisions specificities nullClassifier
##        <int> <dbl>    <dbl>    <dbl>      <dbl>         <dbl>          <dbl>
## 1          1 0.454    0.968    0.956      0.554         0.968          0.558
## 2          2 0.503    0.740    0.688      0.542         0.740          0.558
## 3          3 0.513    0.648    0.597      0.538         0.648          0.558
## 4          4 0.505    0.611    0.582      0.547         0.611          0.558
## 5          5 0.504    0.614    0.587      0.546         0.614          0.558
## 6          6 0.517    0.617    0.564      0.535         0.617          0.558
## 7          7 0.530    0.622    0.543      0.523         0.622          0.558
## 8          8 0.530    0.623    0.545      0.523         0.623          0.558
```

```r
# result shows 1 pca is the best , 2 or 3 number of PCAS areacceptable
```

```r
library(tidyverse)
library(pls)


hitters.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/Hitters.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

# -------------- Usual clean up first -------------------#

# First remove all recods with spaces in character column Salary
hitters.df$Salary <- gsub('\\s+', '', hitters.df$Salary)
```

```r
# Second remove all leading and trailing spaces from a character column "Salary"
hitters.df$Salary <- trimws(hitters.df$Salary, which = c("both"))

# Remove all records with "NA" or empty string in character column "Salary"
hitters.df <- hitters.df[!(tolower(hitters.df$Salary) == "na" |
                           hitters.df$Salary == ""), ]

# convert Salary column to numberic
hitters.df$Salary <- as.numeric(as.character(hitters.df$Salary))

# convert all character fields
hitters.df[sapply(hitters.df, is.character)] <-
  lapply(hitters.df[sapply(hitters.df, is.character)], as.factor)

# Find and remove NA in all columns
hitters.df <- na.omit(hitters.df)

set.seed(2)

# fit the model using pls
pls.fit <- plsr(Salary ~ ., data = hitters.df, scale = T, validation="CV")

print ("Summary:")
```

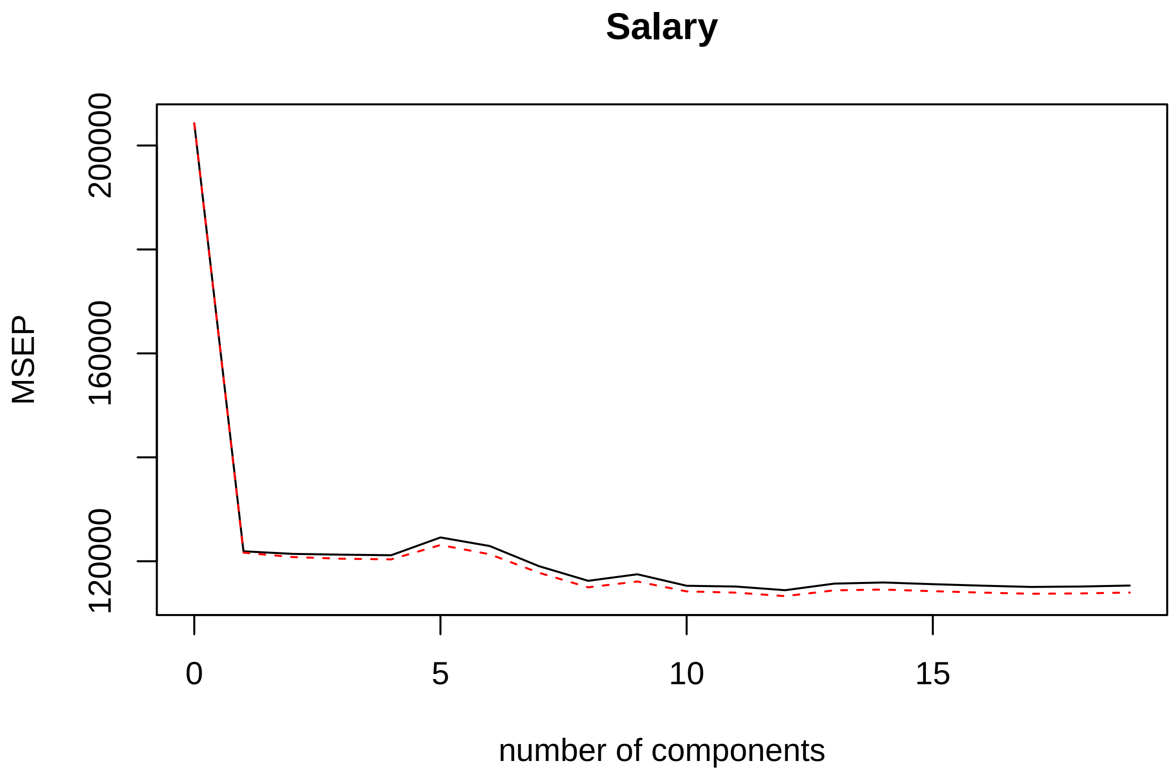
```
## [1] "Summary:"
```

```r
summary(pcr.fit)
```

```
## Data:    X dimension: 131 19
##  Y dimension: 131 1
## Fit method: svdpc
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           428.3    331.5    335.2    329.2    332.1    326.4    330.0
## adjCV        428.3    330.9    334.4    327.5    330.9    325.1    328.6
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       334.6    335.3    335.8     337.5     337.7     342.3     340.3
## adjCV    333.0    333.4    333.3     335.0     335.2     339.6     337.6
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        343.2     346.5     344.2     355.3     361.2     352.5
## adjCV     340.3     343.4     340.9     351.4     356.9     348.3
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         39.32    61.57    71.96    80.83    85.95    89.99    93.25    95.34
## Salary    43.87    43.93    47.36    47.37    49.52    49.55    49.63    50.98
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         96.55     97.61     98.28     98.85     99.22     99.53     99.79
## Salary    53.00     53.00     53.02     53.05     53.80     53.85     54.03
##         16 comps  17 comps  18 comps  19 comps
## X          99.91     99.97     99.99    100.00
```

```
## Salary      55.85      55.89      56.21      58.62
```

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

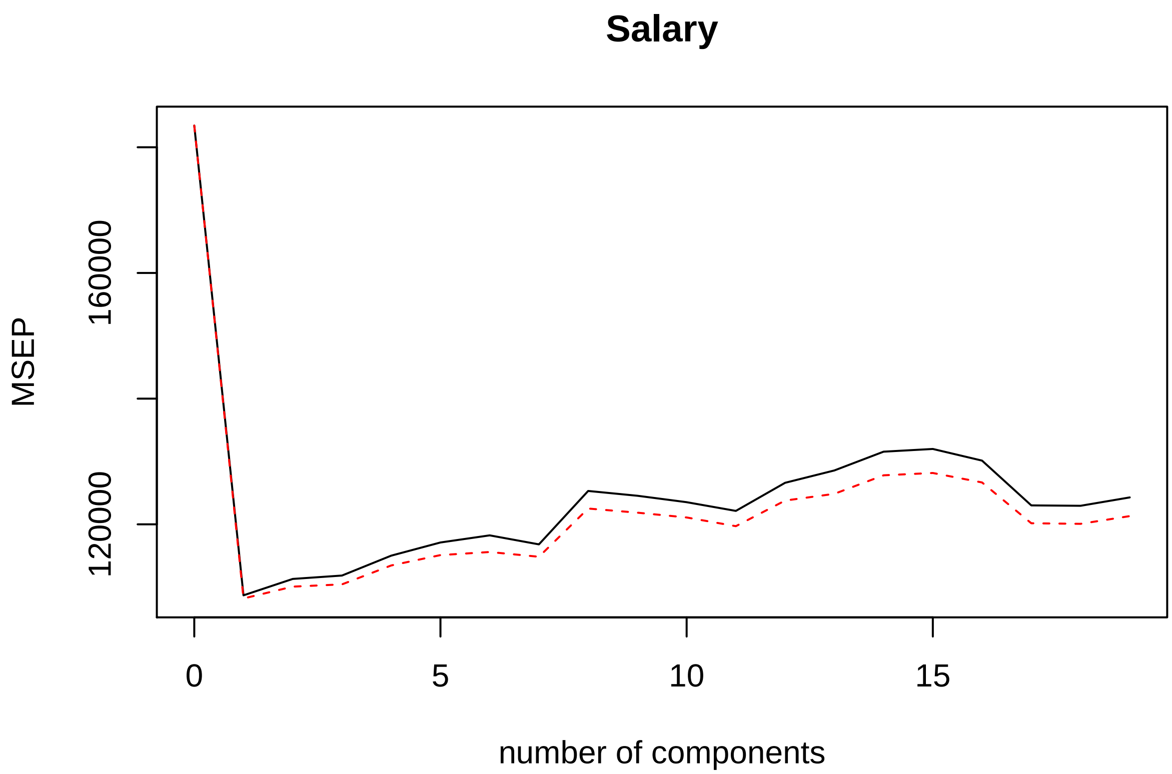



```r
# Now to see how model with 7 works on test data an compare it with
# model with M = 17 we split data into test and train and fit the model
# on train

set.seed(1)
train <- sample(1:nrow(hitters.df), nrow(hitters.df)/2)
test <- (-train)
pls.fit <- plsr(Salary ~ ., data=hitters.df, subset=train, scale=T, validation="CV")
summary(pls.fit)
```

```
## Data:    X dimension: 131 19
##  Y dimension: 131 1
## Fit method: kernelpls
## Number of components considered: 19
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           428.3    329.7    333.6    334.4    339.1    342.2    343.9
## adjCV        428.3    328.9    331.8    332.3    336.8    339.3    340.0
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV       341.8      354    352.9     351.5     349.5     355.8     358.6
## adjCV    338.9      350    349.1     348.0     346.0     351.8     353.3
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV        362.7     363.3     360.8     350.7     350.6     352.5
## adjCV     357.5     358.0     355.9     346.7     346.5     348.3
```

```
##
## TRAINING: % variance explained
##         1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X         39.13    48.80    60.09    75.07    78.58    81.12    88.21    90.71
## Salary    46.36    50.72    52.23    53.03    54.07    54.77    55.05    55.66
##         9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X         93.17     96.05     97.08     97.61     97.97     98.70     99.12
## Salary    55.95     56.12     56.47     56.68     57.37     57.76     58.08
##         16 comps  17 comps  18 comps  19 comps
## X          99.61     99.70     99.95    100.00
## Salary     58.17     58.49     58.56     58.62
```

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




```
# now let's find the lowest cross validation error occurs M = 7 on the  model
test.y <- hitters.df[test,]$Salary
pls.pred <- predict(pls.fit, hitters.df[test,] %>% select(-Salary), ncomp = 7)

sprintf("lowest MSE corresponding to M = 7 is %s (Ridge was 143257.45)",mean((pls.pred-test.y)^2))
```

```
## [1] "lowest MSE corresponding to M = 7 is 143971.58395204 (Ridge was 143257.45)"
```

```
# PLSR really does not add that much value to PCR
mean((pls.pred-test.y)^2)
```

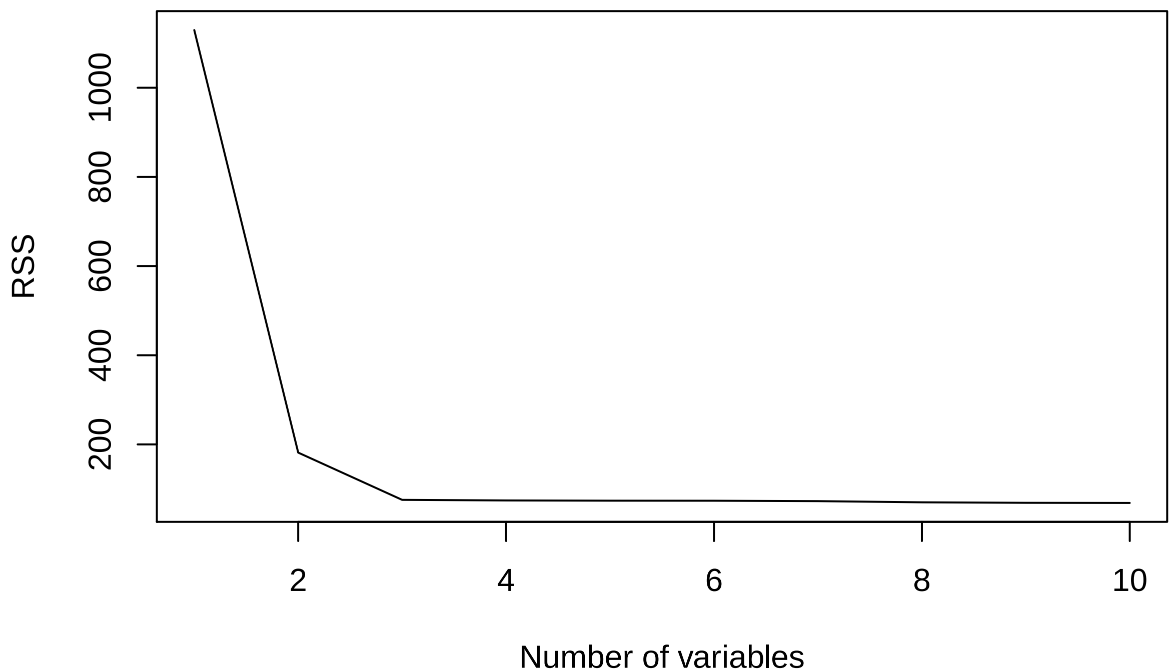
```
## [1] 143971.6
```

```
library(tidyverse)
library(leaps)


X <- rnorm(100)
```

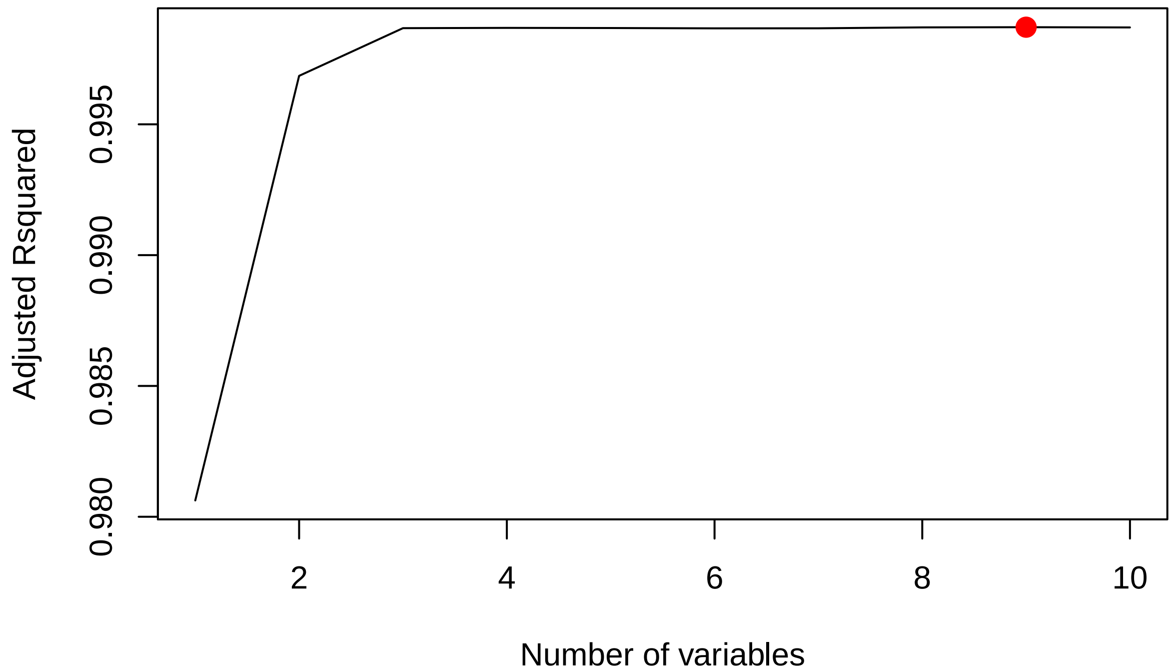
```r
e <- rnorm(100)


Y <- .02 + 1.6*X -2.2 * X^2 + 5.9*X^3 + e

print("Part c:-------------------------------------------------------")
```

```
## [1] "Part c:-------------------------------------------------------"
```

```r
df <- tibble(x1 = X, x2=X^2, x3=X^3, x4=X^4, x5=X^5, x6=X^6, x7=X^7,
             x8=X^8, x9=X^9, x10=X^10, y = Y)

regfit.full <- regsubsets(y ~ ., df, nvmax = 10)

#The summary shows the result of step 2 of algorithm 6.1 page 205 of the book
summary <- summary(regfit.full)

plot(summary$rss, xlab = "Number of variables", ylab="RSS",type = "l")
```




```r
# which.max() returns location maximum point of the vector
index <- which.max(summary$adjr2)
plot(summary$adjr2,xlab = "Number of variables", ylab="Adjusted Rsquared",
     type = "l")
points(index, summary$adjr2[index], col="red", cex=2, pch=20)
```



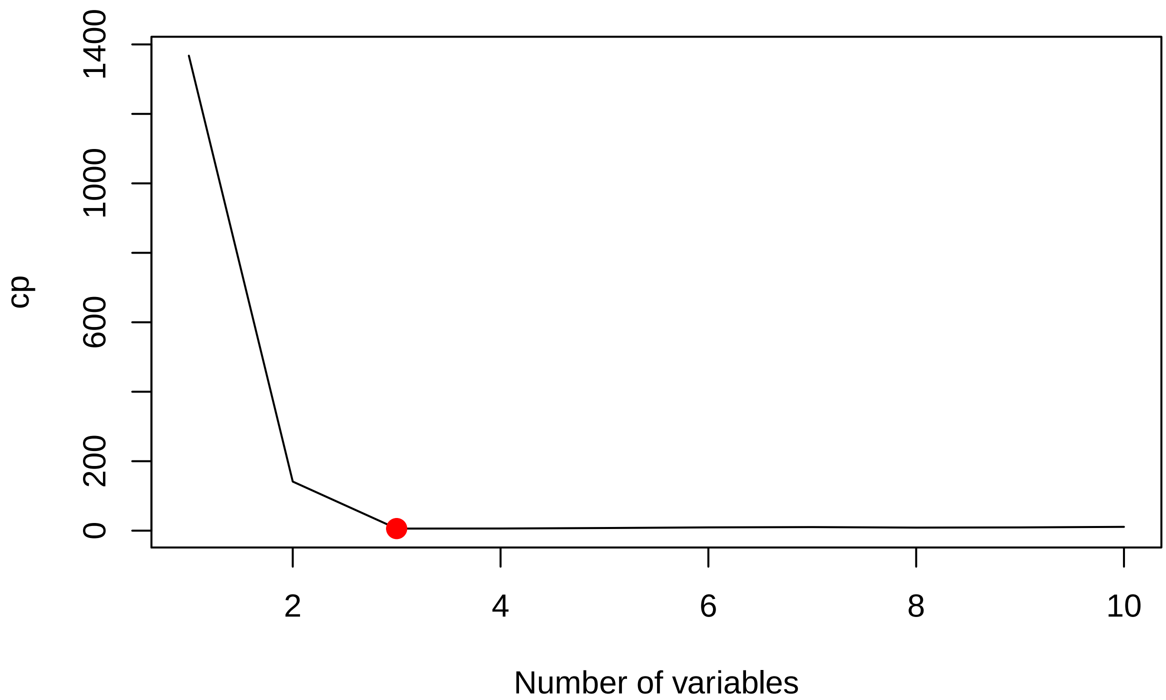

```
print("coefficients of the best model (adjr2) : ")
```

```
## [1] "coefficients of the best model (adjr2) : "
```

```
coef(regfit.full,index)
```

```
## (Intercept)          x1          x2          x3          x4          x5
##  0.25366462  2.03868008 -4.74449117  5.16091299  4.13326339  0.37779502
##          x6          x7          x8         x10
## -2.15279669 -0.05923650  0.42975731 -0.02806608
```

```
# which.min() returns location minimum point of the vector
index <- which.min(summary$cp)
plot(summary$cp,xlab = "Number of variables", ylab="cp", type = "l")
points(index, summary$cp[index], col="red", cex=2, pch=20)
```




```r
print("coefficients of the best model (cp) : ")
```

```
## [1] "coefficients of the best model (cp) : "
```

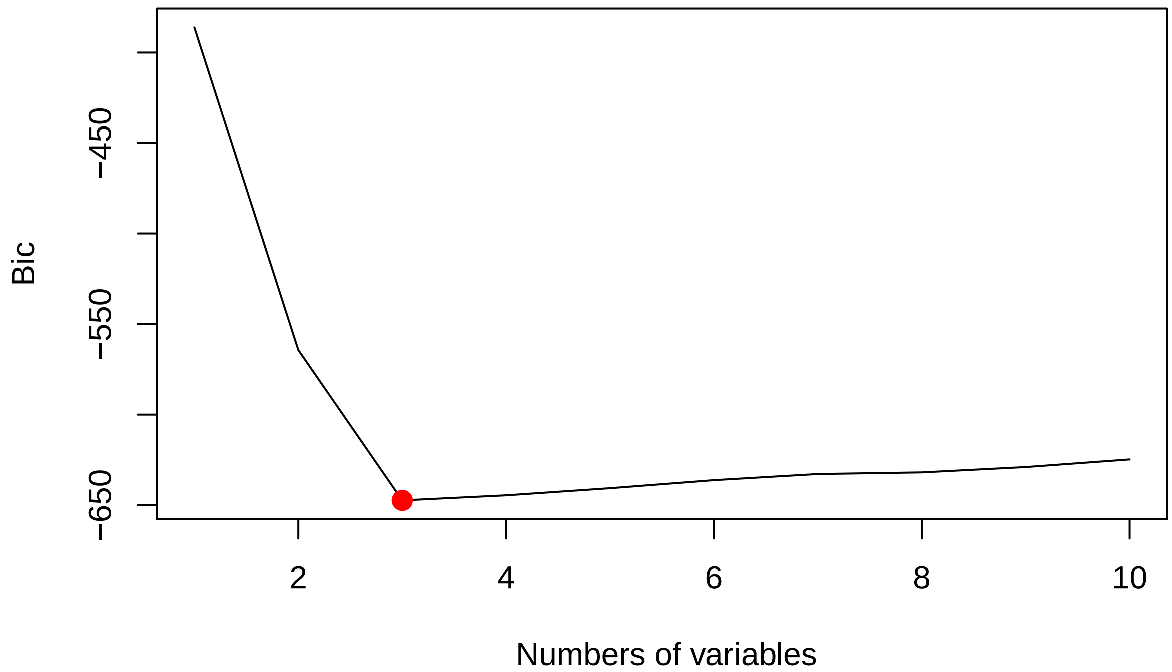
```r
coef(regfit.full,index)
```

```
## (Intercept)          x1          x2          x3
##   0.1304897   1.6922530  -2.2822247   5.8583460
```

```r
# same for bic
plot(summary$bic, xlab =" Numbers of variables", ylab="Bic", type="l")
(index <- which.min(summary$bic))
```

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

```r
points(index, summary$bic[index], col="red", cex=2, pch=20)
```




```r
print("coefficients of the best model (bic) : ")
```

```
## [1] "coefficients of the best model (bic) : "
```

```r
coef(regfit.full,index)
```

```
## (Intercept)          x1          x2          x3
##   0.1304897   1.6922530  -2.2822247   5.8583460
```

```r
# coef(, n) returns coefficient estimates associated with best n variable model



print ("------------- use CV with best subset selection ------------------" )
```

```
## [1] "------------- use CV with best subset selection ------------------"
```

```r
set.seed(1)
k <- 10

folds <- sample(1:k, size = nrow(df), replace = T)
table(folds)
```

```
## folds
##  1  2  3  4  5  6  7  8  9 10
##  9  7  7  6  9 14 14  9 11 14
```

```r
# folds with same size
# sameSizefolds <- sample(rep(1:k, length.out = nrow(weekly.df)),
#   size = nrow(weekly.df), replace = F)
# table(sameSizefolds)

results <- lapply (1:k, function(j) {  # x is the index of test portion, the rest are for training

  # this is to collect the MSEs for each test fold
```

```r
  mses <- tibble(no.of.coefs = NULL, MSE = NULL)
  df.train <- df[folds != j, ]
  df.test <- df[folds == j, ]
  df.train.X <- df.train %>% select (-y)
  df.test.X <- df.test %>% select (-y)
  mat.test.X <- model.matrix(y~., data=df.test)
  df.test.Y <- df.test$y

  # step 2 of algorithm 6.1 page 205 of the book
  regfit.full.train <- regsubsets(y ~ ., df.train, nvmax = ncol(df.train.X))

  # apply the model with selected subsets on test set
  # one at a time and cacluate the MSE
  for (i in 1:ncol(df.test.X)){
    (coefi <- coef(regfit.full.train, id = i))
    (pred <- mat.test.X[, names(coefi)] %*% coefi)
    (mse <- mean((pred - df.test.Y)^2))
     mses <- rbind (mses, tibble(no.of.coefs = i, MSE = mse))
  }
  return(mses)
})

allResults <- results[[1]]
for (i in 2 : length(results)){
  allResults <- rbind(allResults , results[[i]])
}

(allMse <- (allResults %>%
  group_by(no.of.coefs) %>%
  summarise(mse.mean = mean(MSE))) )
```

```
## # A tibble: 10 x 2
##    no.of.coefs  mse.mean
##          <int>     <dbl>
##  1           1    21.2
##  2           2     2.83
##  3           3     0.920
##  4           4     0.913
##  5           5     6.25
##  6           6    30.8
##  7           7    86.9
##  8           8  9814.
##  9           9 17003.
## 10          10 36230.
```

```r
(noOfFeatures <- which.min(allMse$mse.mean))
```

```
## [1] 4
```

```r
print ("The best subset of features selected correspnd to minimum CV_MSE")
```

```
## [1] "The best subset of features selected correspnd to minimum CV_MSE"
```

```r
# train on the whole training set now

regfit.full.train <- regsubsets(y ~ ., df , nvmax = ncol(df %>% select (-y)))
```

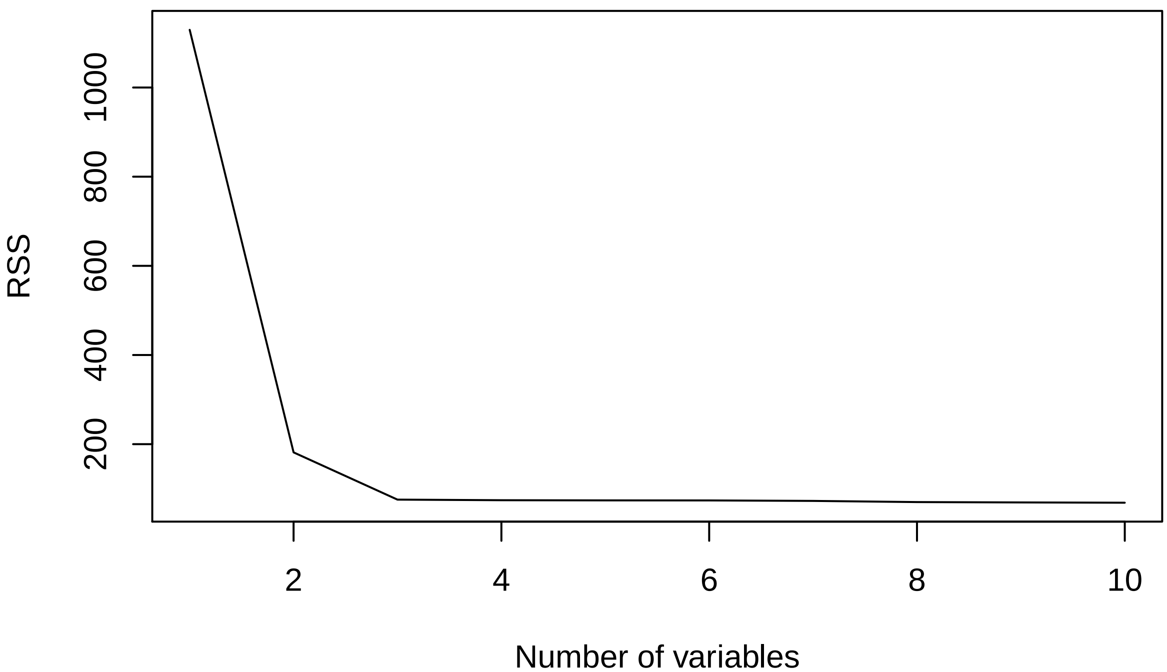
```r
coef(regfit.full.train, id = noOfFeatures)
```

```
## (Intercept)          x1          x2          x3          x4
##  0.04468671  1.74391662 -2.10645119  5.83194832 -0.03328162
```

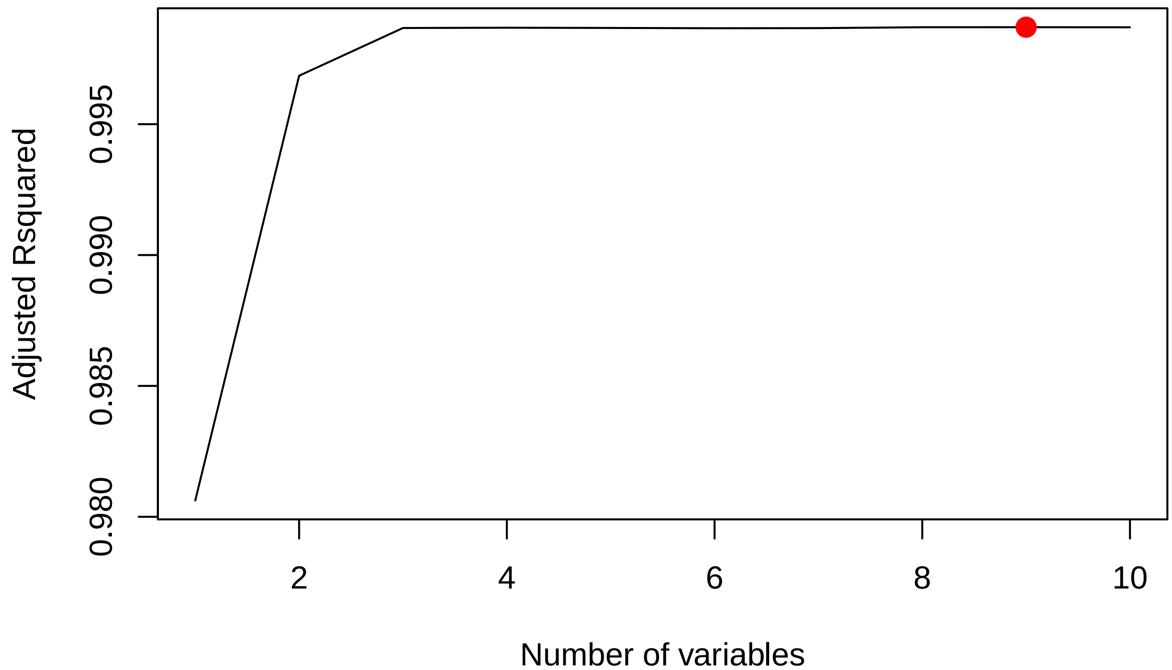
```r
#- -------------------------------------------------------------------------
print ("part d : Use forward stepwise selection to find the best selected subsets  -------------------" )
```

```
## [1] "part d : Use forward stepwise selection to find the best selected subsets  -------------------"
```

```r
regfit.fwd <- regsubsets(y ~ ., df, nvmax = 10, method="forward")

#The summary shows the result of step 2 of algorithm 6.2 page 207 of the book
summary <- summary(regfit.fwd)


plot(summary$rss, xlab = "Number of variables", ylab="RSS",type = "l")
```




```r
# which.max() returns location maximum point of the vector
index <- which.max(summary$adjr2)
plot(summary$adjr2,xlab = "Number of variables", ylab="Adjusted Rsquared",
     type = "l")
points(index, summary$adjr2[index], col="red", cex=2, pch=20)
```



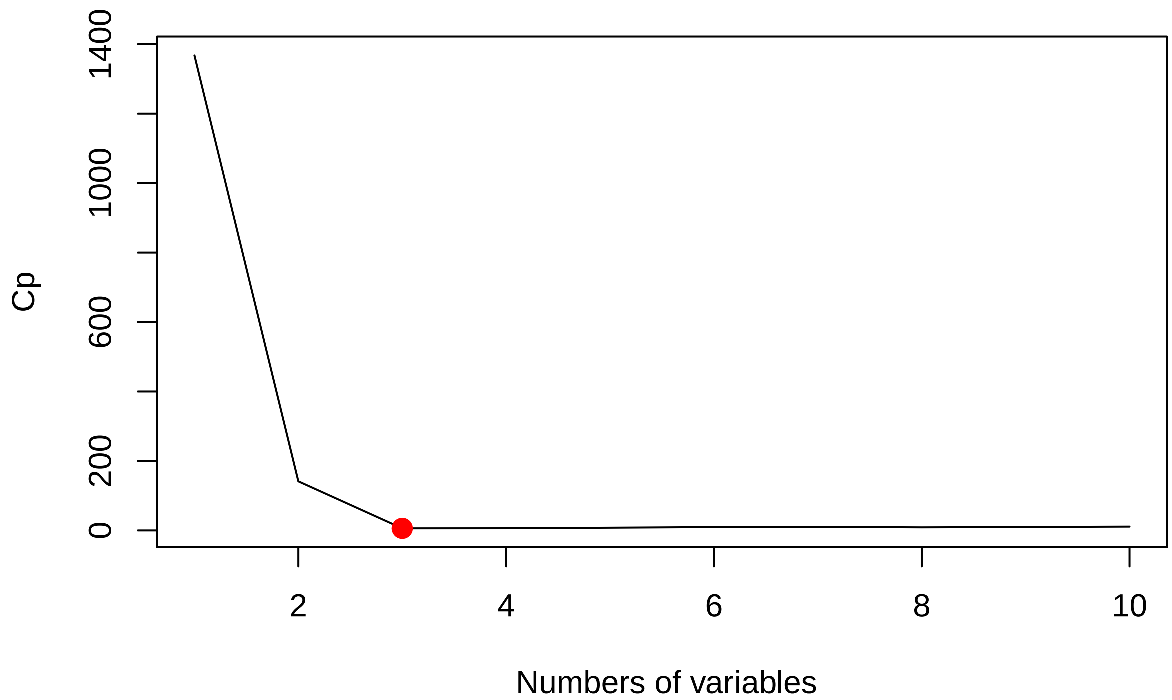

```r
print("coefficients of the best model (adjr2) : ")
```

```
## [1] "coefficients of the best model (adjr2) : "
```

```r
coef(regfit.fwd,index)
```

```
##  (Intercept)           x1           x2           x3           x4           x5 
##  0.241806111  1.900480339 -4.590763438  5.525996094  3.871308892  0.127746525 
##           x6           x8           x9          x10 
## -1.999182860  0.395212947 -0.003996494 -0.025692057
```

```r
# which.min() returns location minimum point of the vector
index <- which.min(summary$cp)
plot(summary$cp, xlab =" Numbers of variables", ylab="Cp", type="l")
points(index, summary$cp[index], col="red", cex=2, pch=20)
```



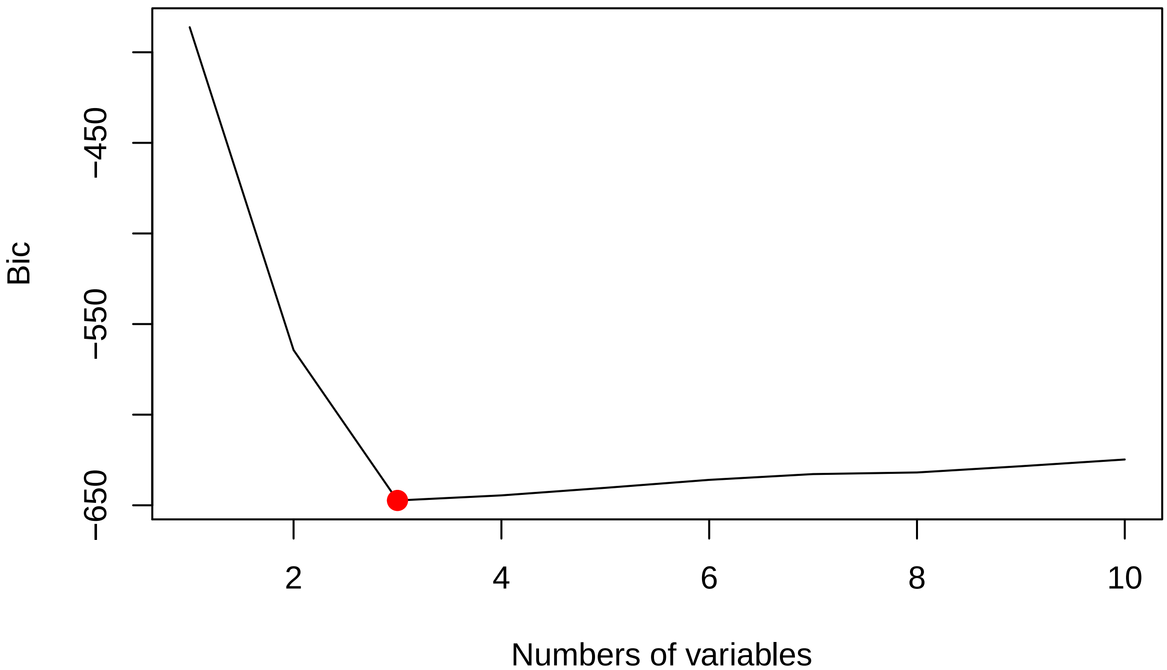

```
print("coefficients of the best model (cp) : ")
```

```
## [1] "coefficients of the best model (cp) : "
```

```
coef(regfit.fwd,index)
```

```
## (Intercept)          x1          x2          x3
##   0.1304897   1.6922530  -2.2822247   5.8583460
```

```
# same for bic
index <- which.min(summary$bic)
plot(summary$bic, xlab =" Numbers of variables", ylab="Bic", type="l")
points(index, summary$bic[index], col="red", cex=2, pch=20)
```




```
print("coefficients of the best model (bic) : ")
```

```
## [1] "coefficients of the best model (bic) : "
```

```
coef(regfit.fwd,index)
```

```
## (Intercept)          x1          x2          x3
##   0.1304897   1.6922530  -2.2822247   5.8583460
```

```
print ("------------- use CV to find best forward stepwise selected model ------------------" )
```

```
## [1] "------------- use CV to find best forward stepwise selected model ------------------"
```

```
set.seed(1)
k <- 10

folds <- sample(1:k, size = nrow(df), replace = T)
table(folds)
```

```
## folds
##  1  2  3  4  5  6  7  8  9 10
##  9  7  7  6  9 14 14  9 11 14
```

```
# folds with same size
# sameSizefolds <- sample(rep(1:k, length.out = nrow(weekly.df)),
#   size = nrow(weekly.df), replace = F)
# table(sameSizefolds)

results <- lapply (1:k, function(x) {  # x is the index of test portion, the rest are for training

  # this is to collect the MSEs for each test fold
  mses <- tibble(no.of.coefs = NULL, MSE = NULL)

  df.train <- df[folds != x, ]
  df.test <- df[folds == x, ]
```

```r
  (df.train.X <- df.train %>% select (-y))
  (df.test.X <- df.test %>% select (-y))
  (mat.test.X <- model.matrix(y~., data=df.test))
  (df.test.Y <- df.test$y)

  # step 2 of algorithm 6.2 page 207 of the book
  regfit.fwd.train <- regsubsets(y ~ ., df.train,
                                 nvmax = ncol(df.train.X), method="forward")

  # apply the model with selected subsets on test set
  # one at a time and cacluate the MSE
  for (i in 1:ncol(df.test.X)){
    (coefi <- coef(regfit.fwd.train, id = i))
    (pred <- mat.test.X[, names(coefi)] %*% coefi)
    (mse <- mean((pred - df.test.Y)^2))
     mses <- rbind (mses, tibble(no.of.coefs = i, MSE = mse))
  }
  return(mses)
})

allResults <- results[[1]]
for (i in 2 : length(results)){
  allResults <- rbind(allResults , results[[i]])
}

(allMse <- (allResults %>%
  group_by(no.of.coefs) %>%
  summarise(mse.mean = mean(MSE))) )
```

```
## # A tibble: 10 x 2
##    no.of.coefs  mse.mean
##          <int>     <dbl>
##  1           1    21.2
##  2           2     2.83
##  3           3     0.920
##  4           4     0.913
##  5           5     1.50
##  6           6     8.00
##  7           7     7.05
##  8           8    96.1
##  9           9  1159.
## 10          10 36230.
```

```r
(no.of.selected.features <- which.min(allMse$mse.mean))
```

```
## [1] 4
```

```r
print ("The forward features selected correspnd to minimum CV_MSE")
```

```
## [1] "The forward features selected correspnd to minimum CV_MSE"
```

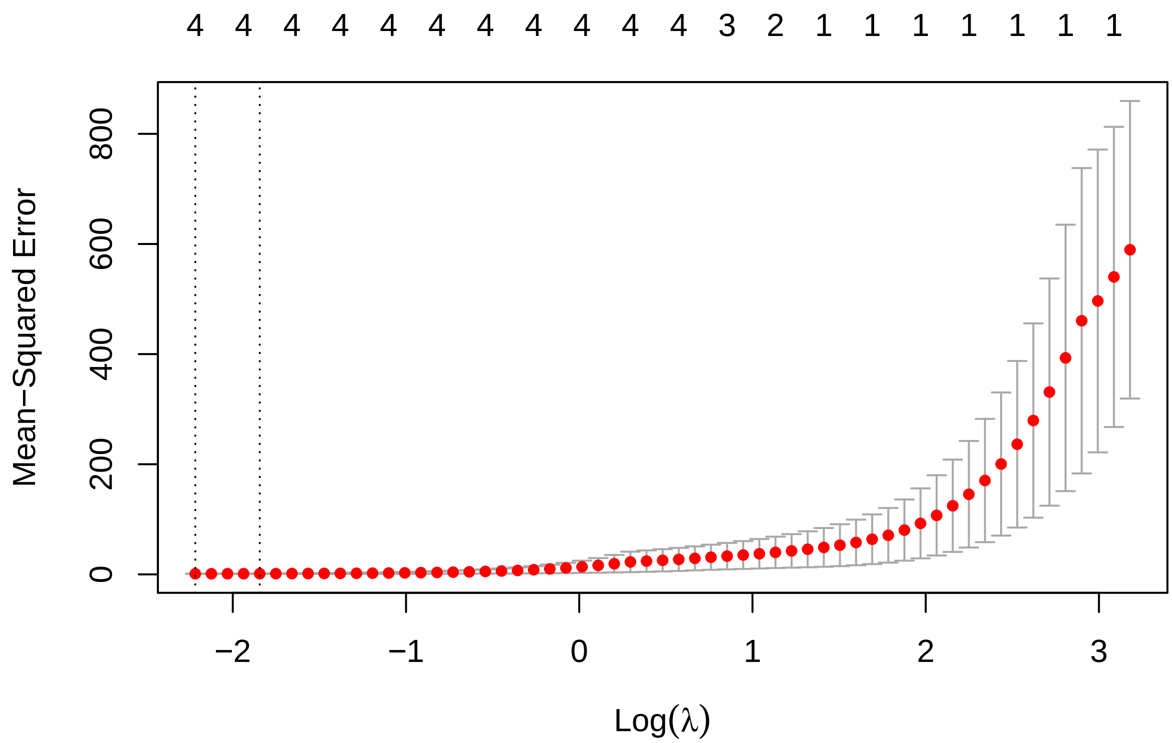
```r
# train on the whole training set now
regfit.fwd.train <- regsubsets(y ~ ., df , nvmax = ncol(df %>% select (-y)),
                               method = "forward")
coef(regfit.fwd.train, id = no.of.selected.features)
```

```
## (Intercept)          x1          x2          x3          x4
##  0.04468671  1.74391662 -2.10645119  5.83194832 -0.03328162
```

```r
#---------------------------------------------------------------------------
print("Part e: Fit lasso model and use cv to find the best value for lamda --------------------")
```

```
## [1] "Part e: Fit lasso model and use cv to find the best value for lamda --------------------"
```

```r
#-----------------------------------------------------------------------------

library(glmnet)
set.seed(1)

# First construct matrix from dataframe (and drop intercept column)
x <- model.matrix(y~., df)[,-1]
y <- df$y

cv.out=cv.glmnet(x, y, alpha=1, lambda = NULL)
plot(cv.out)
```




```r
print("Here is value of lambda for which the MSE is minimum")
```

```
## [1] "Here is value of lambda for which the MSE is minimum"
```

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

```
## [1] 0.108985
```

```r
print("Here are the coefficients corresponding to best value of lambda:")
```

```
## [1] "Here are the coefficients corresponding to best value of lambda:"
```

```r
predict(cv.out, type="coefficients" ,s=bestlam )
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
```

```
##                        1
## (Intercept) -0.05681726
## x1           1.65284155
## x2          -1.99192279
## x3           5.82137607
## x4          -0.03838223
## x5           .
## x6           .
## x7           .
## x8           .
## x9           .
## x10          .
```

```
print ( "Lasso coefficients are not as close as that of all feature selections !!")
```

```
## [1] "Lasso coefficients are not as close as that of all feature selections !!"
```

```
print("Here is one standard error value of lambda for which the MSE is minimum")
```

```
## [1] "Here is one standard error value of lambda for which the MSE is minimum"
```

```
one.SE.lam <- cv.out$lambda.1se

print("Here are the coefficients corresponding to one standard error value of lambda:")
```

```
## [1] "Here are the coefficients corresponding to one standard error value of lambda:"
```

```
predict(cv.out, type="coefficients" ,s=one.SE.lam )
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept) -0.09769231
## x1           1.61472612
## x2          -1.94823793
## x3           5.81658758
## x4          -0.03952492
## x5           .
## x6           .
## x7           .
## x8           .
## x9           .
## x10          .
```

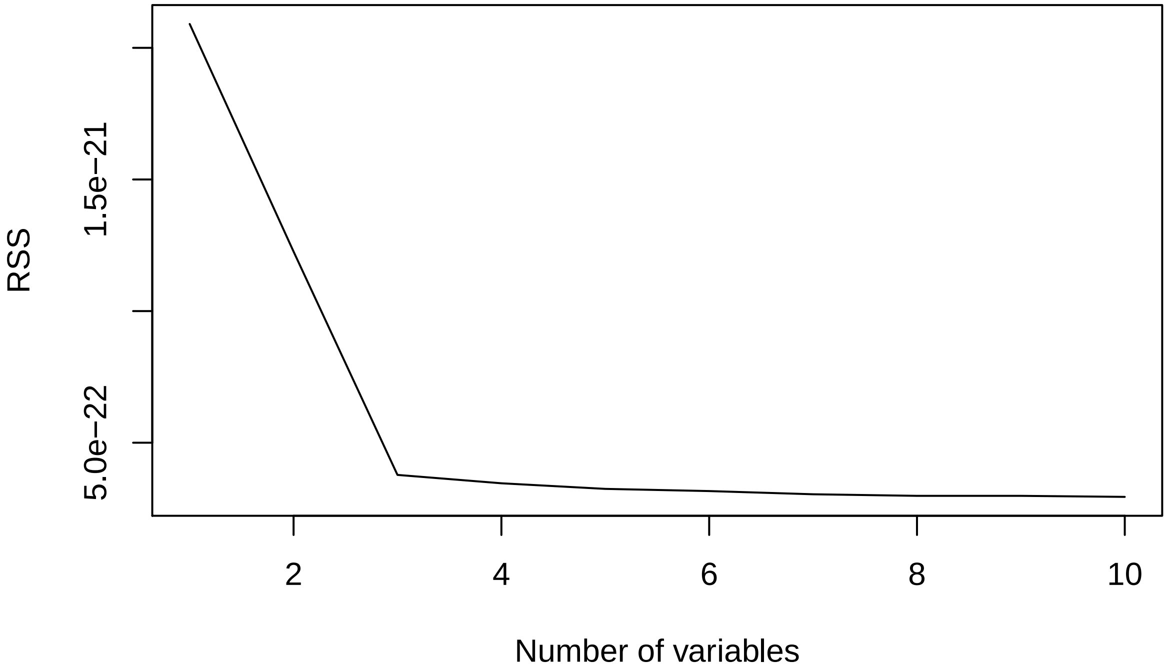
```
#-------------------------------------------------------------------
print("Part f: Perform Best subset selection and lasso on new data ------------")
```

```
## [1] "Part f: Perform Best subset selection and lasso on new data ------------"
```

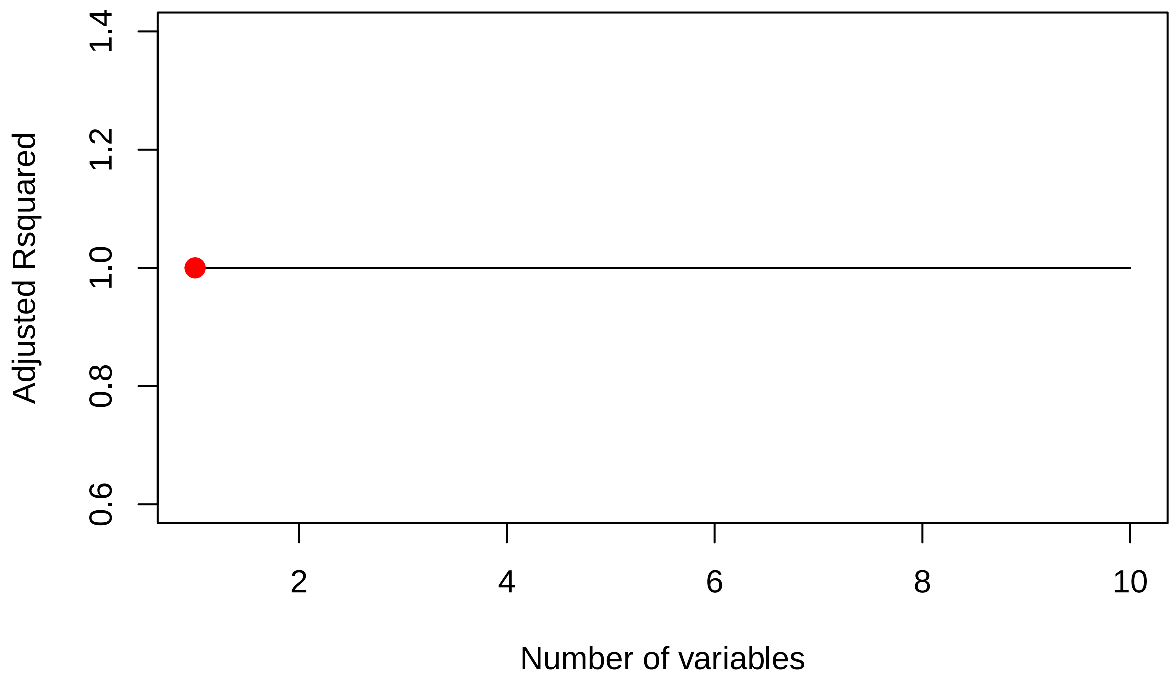
```
#--------------------------------------------------------------------

Y <- 12 - 45.3 * X^7
df <- tibble(x1 = X, x2=X^2, x3=X^3, x4=X^4, x5=X^5, x6=X^6, x7=X^7,
             x8=X^8, x9=X^9, x10=X^10, y = Y)
library(glmnet)
set.seed(1)

print ("Perform best subset selection: ")
```

```
## [1] "Perform best subset selection: "
```

```r
regfit.full <- regsubsets(y ~ ., df, nvmax = 10)

#The summary shows the result of step 2 of algorithm 6.1 page 205 of the book
summary <- summary(regfit.full)

plot(summary$rss, xlab = "Number of variables", ylab="RSS",type = "l")
```




```r
# which.max() returns location maximum point of the vector
index <- which.max(summary$adjr2)
plot(summary$adjr2,xlab = "Number of variables", ylab="Adjusted Rsquared",
     type = "l")
points(index, summary$adjr2[index], col="red", cex=2, pch=20)
```



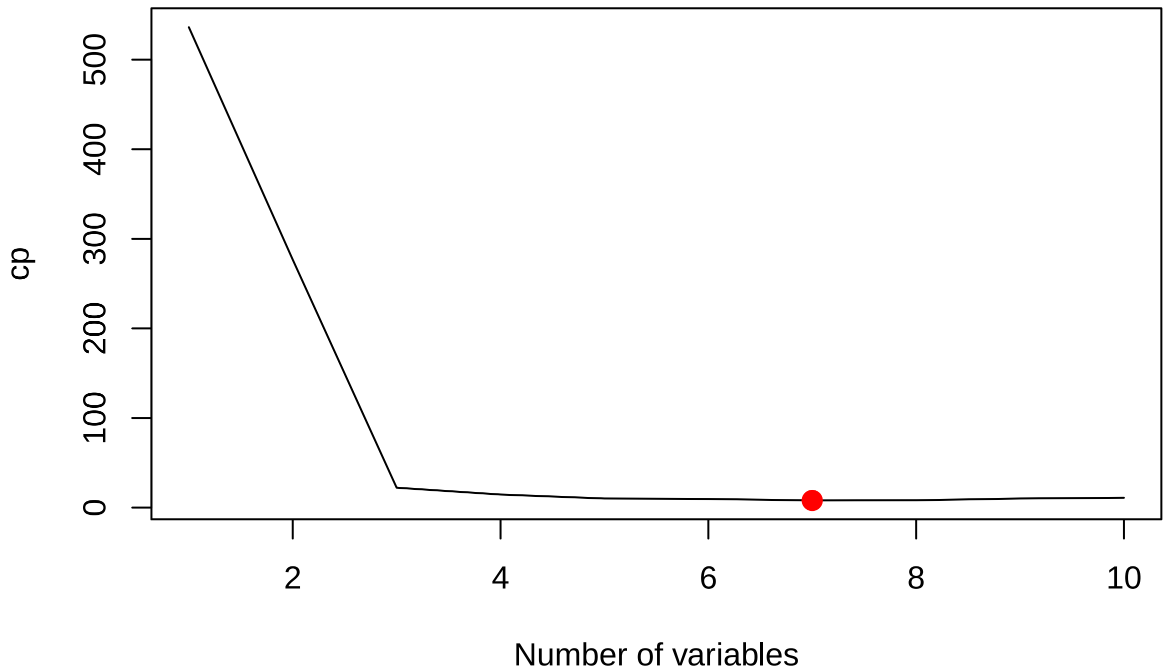

```r
print("coefficients of the best model (adjr2) : ")
```

```
## [1] "coefficients of the best model (adjr2) : "
```

```r
coef(regfit.full,index)
```

```
## (Intercept)          x7
##        12.0       -45.3
```

```r
# which.min() returns location minimum point of the vector
index <- which.min(summary$cp)
plot(summary$cp,xlab = "Number of variables", ylab="cp", type = "l")
points(index, summary$cp[index], col="red", cex=2, pch=20)
```




```r
print("coefficients of the best model (cp) : ")
```

```
## [1] "coefficients of the best model (cp) : "
```

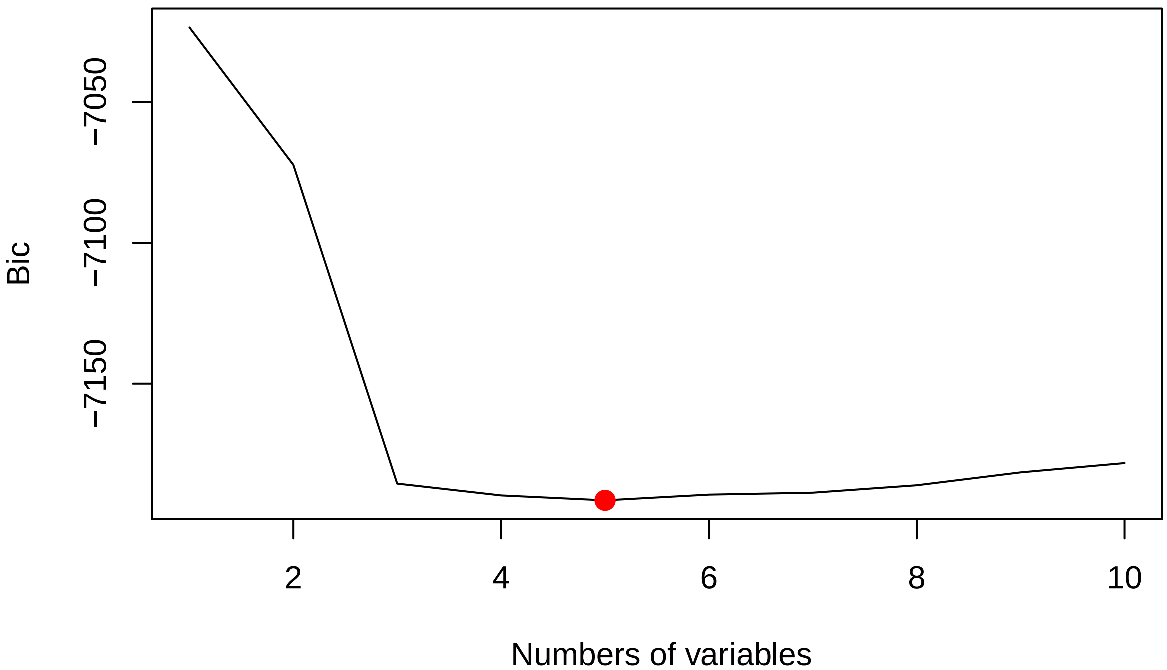
```r
coef(regfit.full,index)
```

```
##   (Intercept)            x1            x2            x3            x5
##  1.200000e+01  1.949149e-12  7.458195e-13  5.604306e-12 -3.916634e-12
##            x7            x9           x10
## -4.530000e+01 -6.896989e-14 -2.863481e-15
```

```r
# same for bic
plot(summary$bic, xlab =" Numbers of variables", ylab="Bic", type="l")
(index <- which.min(summary$bic))
```

```
## [1] 5
```

```r
points(index, summary$bic[index], col="red", cex=2, pch=20)
```




```
print("coefficients of the best model (bic) : ")
```

```
## [1] "coefficients of the best model (bic) : "
```

```
coef(regfit.full,index)
```

```
##   (Intercept)            x1            x2            x5            x7
##  1.200000e+01  4.010558e-12  5.727175e-13 -4.867231e-14 -4.530000e+01
##           x10
## -1.519573e-15
```

```
# coef(, n) returns coefficient estimates associated with best n variable model



print ("------ use CV with best subset selection for new data --------------" )
```

```
## [1] "------ use CV with best subset selection for new data --------------"
```

```
set.seed(1)
k <- 10

folds <- sample(1:k, size = nrow(df), replace = T)
table(folds)
```

```
## folds
##  1  2  3  4  5  6  7  8  9 10
##  9  7  7  6  9 14 14  9 11 14
```

```
# folds with same size
# sameSizefolds <- sample(rep(1:k, length.out = nrow(weekly.df)),
#   size = nrow(weekly.df), replace = F)
# table(sameSizefolds)

results <- lapply (1:k, function(x) {  # x is the index of test portion, the rest are for training
```

```r
  # this is to collect the MSEs for each test fold
  mses <- tibble(no.of.coefs = NULL, MSE = NULL)
  df.train <- df[folds != x, ]
  df.test <- df[folds == x, ]
  df.train.X <- df.train %>% select (-y)
  df.test.X <- df.test %>% select (-y)
  (mat.test.X <- model.matrix(y~., data=df.test))
  (df.test.Y <- df[folds == x, ]$y)

  # step 2 of algorithm 6.1 page 205 of the book
  regfit.full.train <- regsubsets(y ~ ., df.train, nvmax = ncol(df.train.X))

  # apply the model with selected subsets on test set
  # one at a time and cacluate the MSE
  for (i in 1:ncol(df.test.X)){
    (coefi <- coef(regfit.full.train, id = i))
    (pred <- mat.test.X[, names(coefi)] %*% coefi)
    (mse <- mean((pred - df.test.Y)^2))
     mses <- rbind (mses, tibble(no.of.coefs = i, MSE = mse))
  }
  return(mses)
})

allResults <- results[[1]]
for (i in 2 : length(results)){
  allResults <- rbind(allResults , results[[i]])
}

(allMse <- (allResults %>%
  group_by(no.of.coefs) %>%
  summarise(mse.mean = mean(MSE))) )
```

```
## # A tibble: 10 x 2
##    no.of.coefs mse.mean
##          <int>    <dbl>
##  1           1 1.22e-23
##  2           2 2.64e-23
##  3           3 1.96e-23
##  4           4 4.62e-22
##  5           5 1.69e-22
##  6           6 2.23e-21
##  7           7 9.39e-21
##  8           8 1.13e-20
##  9           9 2.67e-20
## 10          10 2.61e-20
```

```r
(no.of.selected.features <- which.min(allMse$mse.mean))
```

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

```r
print ("The best subset of features selected correspnd to minimum CV_MSE")
```

```
## [1] "The best subset of features selected correspnd to minimum CV_MSE"
```

```r
# train on the whole training set now
regfit.full.train <- regsubsets(y ~ ., df, nvmax = ncol(df %>% select (-y)))
coef(regfit.full.train, id = no.of.selected.features)
```

```
## (Intercept)          x7
##        12.0       -45.3
```

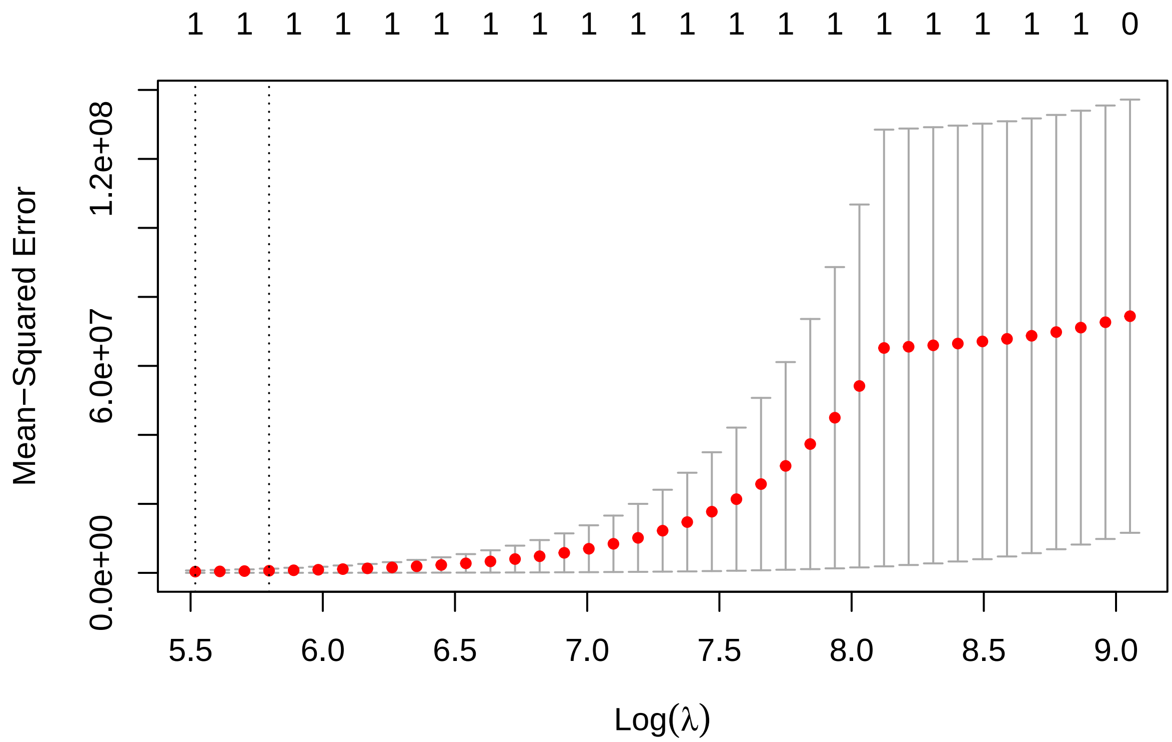
```r
print(" Clearly applying CV on Best subset selection provides nonsensical result.")
```

```
## [1] " Clearly applying CV on Best subset selection provides nonsensical result."
```

```r
print ("Apply Lasso  cross validation to find the best lambda and corresponding coeffs")
```

```
## [1] "Apply Lasso  cross validation to find the best lambda and corresponding coeffs"
```

```r
library(glmnet)
set.seed(1)

# First construct matrix from dataframe (and drop intercept column)
x <- model.matrix(y~., df)[,-1]
y <- df$y

cv.out=cv.glmnet(x, y, alpha=1, lambda = NULL)
plot(cv.out)
```




```r
print("Here is value of lambda for which the MSE is minimum")
```

```
## [1] "Here is value of lambda for which the MSE is minimum"
```

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

```
## [1] 249.0647
```

```r
print("Here are the coefficients corresponding to best value of lambda:")
```

```
## [1] "Here are the coefficients corresponding to best value of lambda:"
```

```r
predict(cv.out, type="coefficients" ,s=bestlam )
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     1
## (Intercept)  19.32170
## x1            .
## x2            .
## x3            .
## x4            .
## x5            .
## x6            .
## x7          -43.97948
## x8            .
## x9            .
## x10           .
```

```r
print("Here is one standard error value of lambda for which the MSE is minimum")
```

```
## [1] "Here is one standard error value of lambda for which the MSE is minimum"
```

```r
one.SE.lam <- cv.out$lambda.1se

print("Here are the coefficients corresponding to one standard error value of lambda:")
```

```
## [1] "Here are the coefficients corresponding to one standard error value of lambda:"
```

```r
predict(cv.out, type="coefficients" ,s=one.SE.lam )
```

```
## 11 x 1 sparse Matrix of class "dgCMatrix"
##                     1
## (Intercept)  21.67886
## x1            .
## x2            .
## x3            .
## x4            .
## x5            .
## x6            .
## x7          -43.55435
## x8            .
## x9            .
## x10           .
```

```r
library(tidyverse)
library(glmnet)
library(pls)
library(leaps)

set.seed(1)
college.df = read.csv("/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/College.csv",
                      header=T, stringsAsFactors = F, na.strings = "?")
college.df = tibble(college.df)

# str(college.df)
```

```r
#-------- Some usual cleaning on character columns ---------------- #

# First remove all recods with spaces in character column Private
college.df$Private <- gsub('\\s+', '', college.df$Private)

# Second remove all leading and trailing spaces from a character column "Private"
college.df$Private <- trimws(college.df$Private, which = c("both"))

# Remove all records with "NA" or empty string in character column "Private"
college.df <- college.df[!(tolower(college.df$Private) == "na" |
                           college.df$Private == ""), ]

# convert all character fields
college.df[sapply(college.df, is.character)] <-
  lapply(college.df[sapply(college.df, is.character)], as.factor)

#----------- Find and remove NA in all columns ------------- #
college.df <- na.omit(college.df)

# str(college.df)
set.seed(1)
print("a: Split into train / test data sets ---------------------")
```

```
## [1] "a: Split into train / test data sets ---------------------"
```

```r
train <- sample(1:nrow(college.df), nrow(college.df)/2)
test <- (-train)

train.df <- college.df[train, ]
test.df <- college.df[test, ]
y.train <- train.df$Apps
y.test <- test.df$Apps

print("b: fit a linear model ------------------------------------")
```

```
## [1] "b: fit a linear model ------------------------------------"
```

```r
# fit a model
df.lm <- lm (Apps ~ ., data = train.df)

# Now predict  Apps for test data
pred.lm <- predict(df.lm, test.df)

print("Linear model test MSE:")
```

```
## [1] "Linear model test MSE:"
```

```r
(lm.MSE <- mean( (pred.lm - y.test)^2))
```

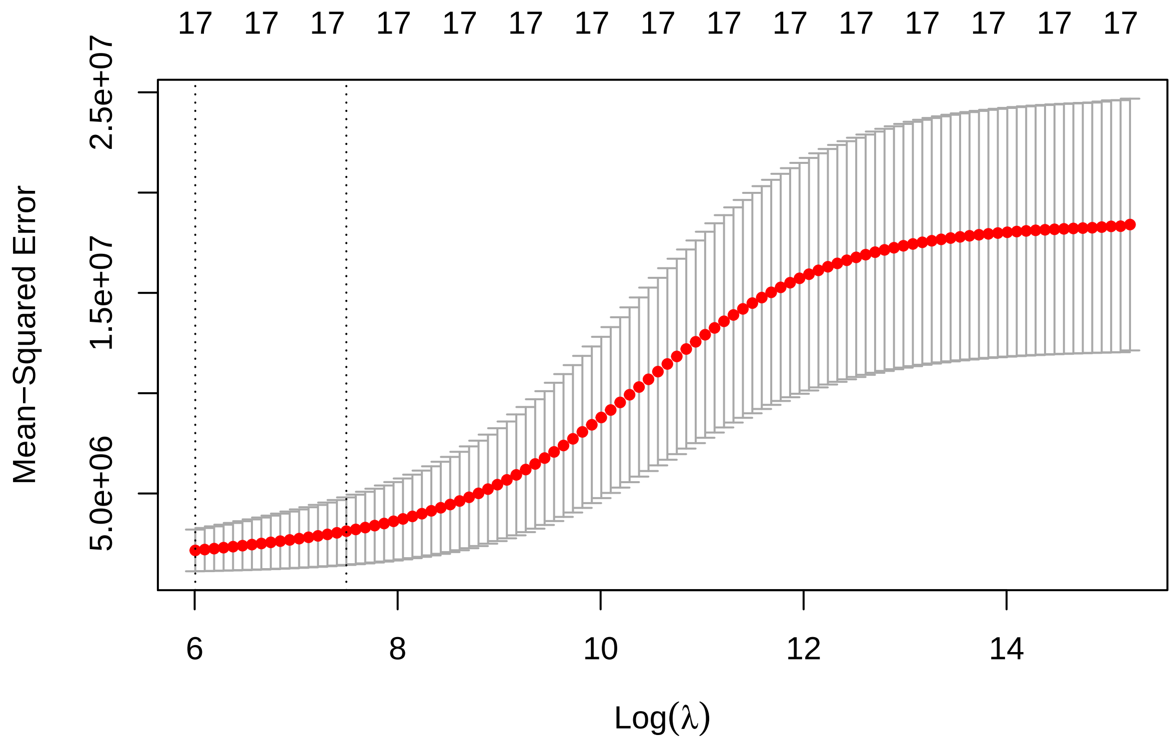
```
## [1] 1135758
```

```r
print("c: fit Ridge regression model on training set and get test.mse-------------")
```

```
## [1] "c: fit Ridge regression model on training set and get test.mse-------------"
```

```r
# use magic model.matrix to convert dataframe into a matrix for Ridge and Lasso
x.train <- model.matrix(Apps~., train.df)[, -1]
```

```r
x.test <- model.matrix(Apps~., test.df)[, -1]


cv.out <- cv.glmnet(x.train, y.train, alpha=0) # Ridge
plot(cv.out)
```




```r
best.lambda <- cv.out$lambda.min

# predict the model on test
pred.ridge <- predict(cv.out, s=best.lambda, newx=x.test)

sprintf("Ridge test mse for best lambda: %s", best.lambda)
```

```
## [1] "Ridge test mse for best lambda: 405.840359582873"
```

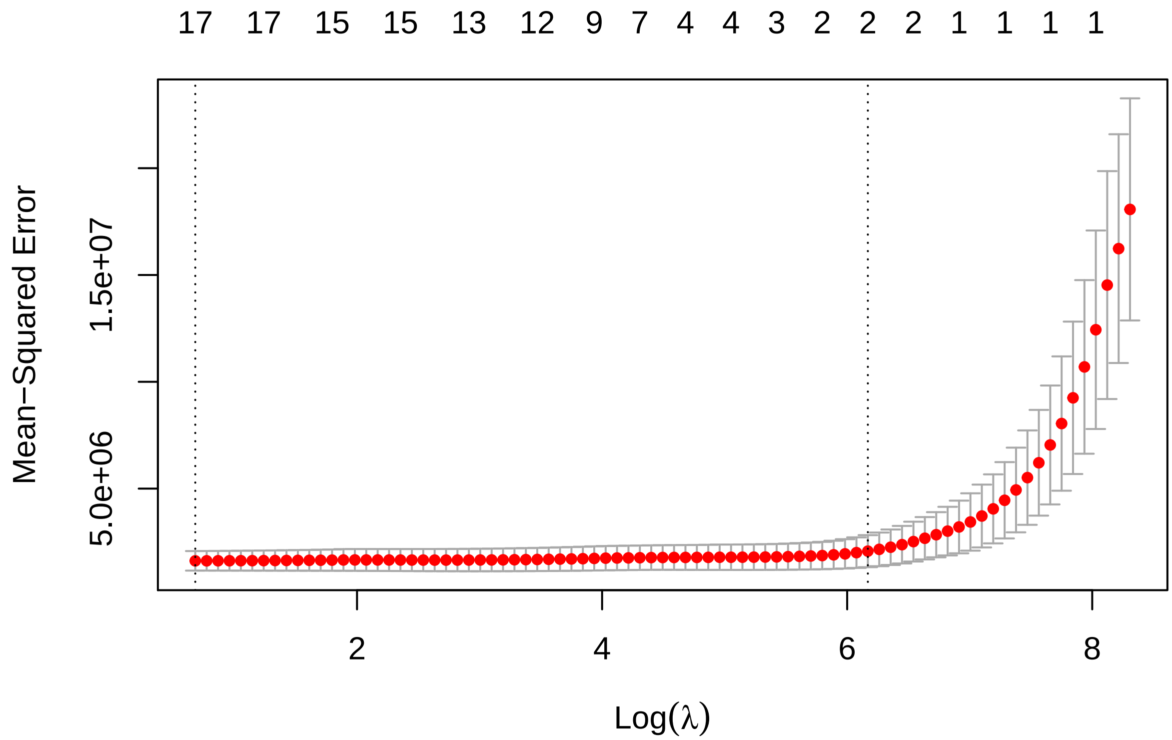
```r
(ridge.mse <- mean((pred.ridge - y.test)^2))
```

```
## [1] 976261.5
```

```r
print("d: fit Lasso regression model on training set and get test.mse and coeffs")
```

```
## [1] "d: fit Lasso regression model on training set and get test.mse and coeffs"
```

```r
cv.out <- cv.glmnet(x.train, y.train, alpha=1) # Lasso
plot(cv.out)
```




```r
best.lambda <- cv.out$lambda.min

# predict the model on test
pred.lasso <- predict(cv.out, s=best.lambda, newx=x.test)

sprintf("Lasso test mse for best lambda: %s", best.lambda)
```

```
## [1] "Lasso test mse for best lambda: 1.97343997085518"
```

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

```
## [1] 1115901
```

```r
print("Coeffs for Lasso: ")
```

```
## [1] "Coeffs for Lasso: "
```

```r
(coefs.ridge <- predict (cv.out, type = "coefficients", s = best.lambda))
```

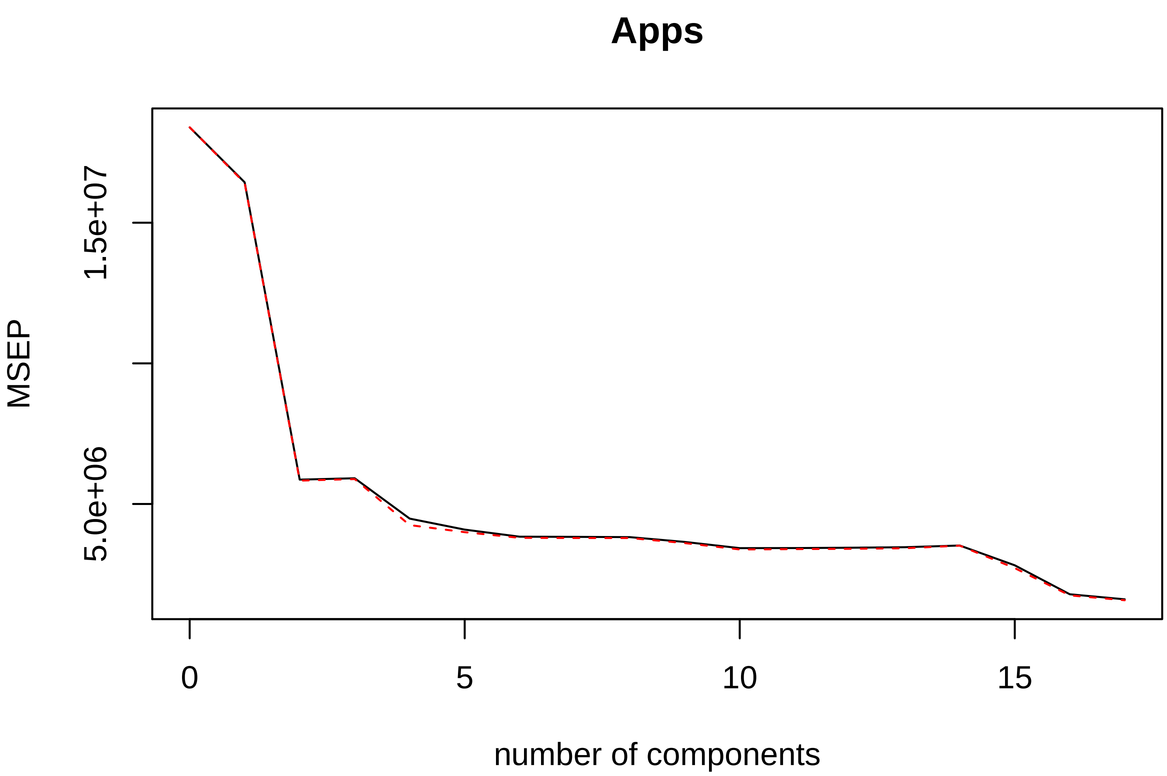
```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                          1
## (Intercept) -7.688896e+02
## PrivateYes  -3.127034e+02
## Accept       1.762718e+00
## Enroll      -1.318195e+00
## Top10perc    6.482356e+01
## Top25perc   -2.081406e+01
## F.Undergrad  7.119149e-02
## P.Undergrad  1.246161e-02
## Outstate    -1.049091e-01
## Room.Board   2.088305e-01
## Books        2.926466e-01
## Personal     3.955068e-03
```

```
## PhD          -1.455463e+01
## Terminal      5.395858e+00
## S.F.Ratio     2.171398e+01
## perc.alumni   5.088260e-01
## Expend        4.824455e-02
## Grad.Rate     7.036148e+00
```

```r
print("e: Fit a PCR model on training set with M chosen by CS and get test.mse")
```

```
## [1] "e: Fit a PCR model on training set with M chosen by CS and get test.mse"
```

```r
pcr.fit <- pcr (Apps~., data=train.df, scale=T, validation="CV")


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




```r
print("Minimum CV Root MSE is for M=17 components which is 100 so CV MSE is 10000")
```

```
## [1] "Minimum CV Root MSE is for M=17 components which is 100 so CV MSE is 10000"
```

```r
print("Looking at validation plot we see that M = 15 or 16  should suffice")
```

```
## [1] "Looking at validation plot we see that M = 15 or 16  should suffice"
```

```r
# Now apply model 1ith M=17 on test data and calculate MSE'
M = 17
pcr.pred <- predict(pcr.fit, x.test, ncomp = M)
sprintf("pcr test cv_mse for when best number of component is: %s", M)
```

```
## [1] "pcr test cv_mse for when best number of component is: 17"
```

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

```
## [1] 1135758
```

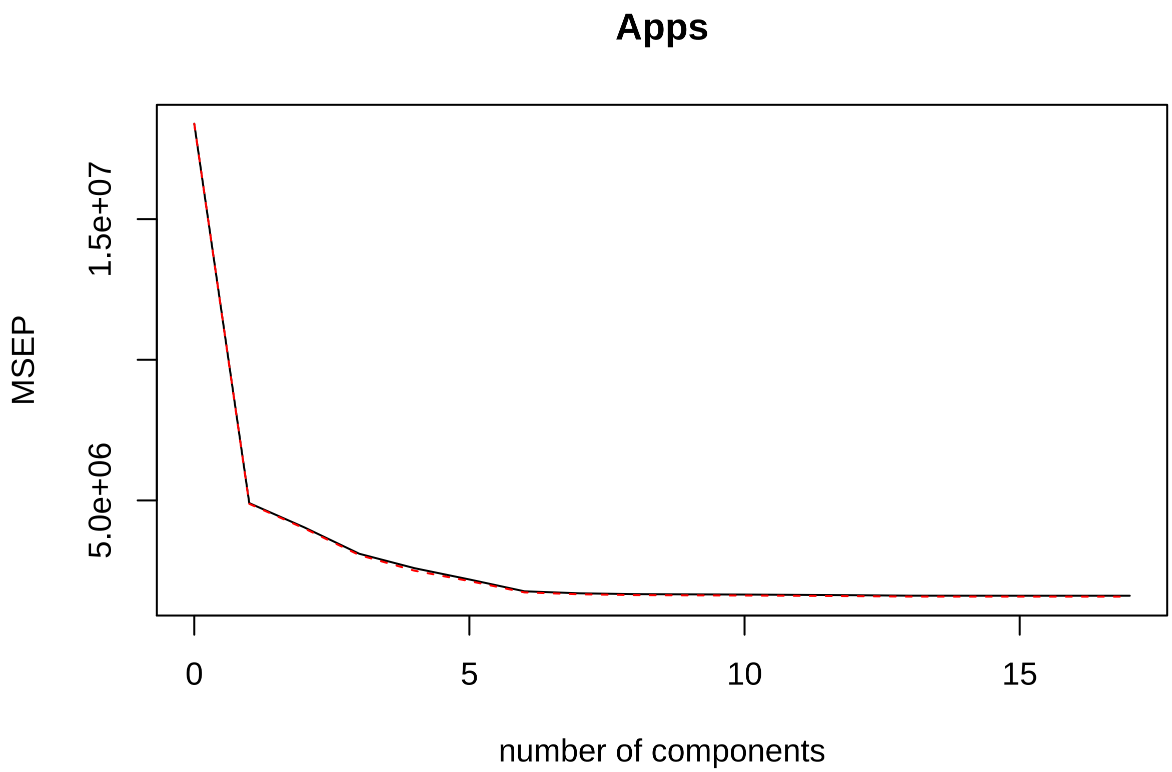
```r
sprintf("pcr test mse  for best number of component: %s is %s:", M, pcr.mse)
```

```
## [1] "pcr test mse  for best number of component: 17 is 1135758.31783053:"
```

```r
print("f: Fit a PLS model on training set with M chosen by CS and get test.mse")
```

```
## [1] "f: Fit a PLS model on training set with M chosen by CS and get test.mse"
```

```r
pls.fit <- plsr(Apps ~ ., data = college.df, subset = train, scale=T, validation = "CV")
validationplot(pls.fit, val.type="MSEP")
```




```r
print("Minimum CV Root MSE is for M=13 components which is 1118")
```

```
## [1] "Minimum CV Root MSE is for M=13 components which is 1118"
```

```r
# Now apply model 1ith M=13 on test data and calculate MSE'
M = 13
pls.pred <- predict(pls.fit, x.test, ncomp = M)

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

```
## [1] 1140255
```

```r
sprintf("pls test mse for for best number of component: %s is %s:", M, pls.mse)
```

```
## [1] "pls test mse for for best number of component: 13 is 1140255.01397807:"
```

```r
library(tidyverse)
library(glmnet)
library(pls)
library(leaps)
```

```r
print("a) Generate a data set with p = 20 features, n = 1,000")
```

```
## [1] "a) Generate a data set with p = 20 features, n = 1,000"
```

```r
set.seed(10)
X <- matrix(rep(NA, 1000 * 20), c(1000,20))
for (i in 1: 20)
  X[, i] <- rnorm(1000)

e <- rnorm(1000)

beta <- c(12,0,2.6, -123,0,11.2,56,-7,0,0,0,13,-41,2.2,0,8.7, -18,0,19,0.03)

# name the features
feature.names <- c("x1", "x2", "x3", "x4", "x5", "x6", "x7", "x8", "x9", "x10",
    "x11", "x12", "x13", "x14", "x15", "x16", "x17", "x18", "x19", "x20" )
colnames(X) <- feature.names
names(beta) <- feature.names
beta
```

```
##      x1      x2      x3      x4      x5      x6      x7      x8      x9     x10
##   12.00    0.00    2.60 -123.00    0.00   11.20   56.00   -7.00    0.00    0.00
##     x11     x12     x13     x14     x15     x16     x17     x18     x19     x20
##    0.00   13.00  -41.00    2.20    0.00    8.70  -18.00    0.00   19.00    0.03
```

```r
Y <- rep(NA, 1000)

for (i in 1 : 1000)
  Y[[i]] <- beta %*% X[i, ] + e[[i]]

# now split the samples into test and training:
print("b) Split the data into train setcontaining 100 and test set containing 900")
```

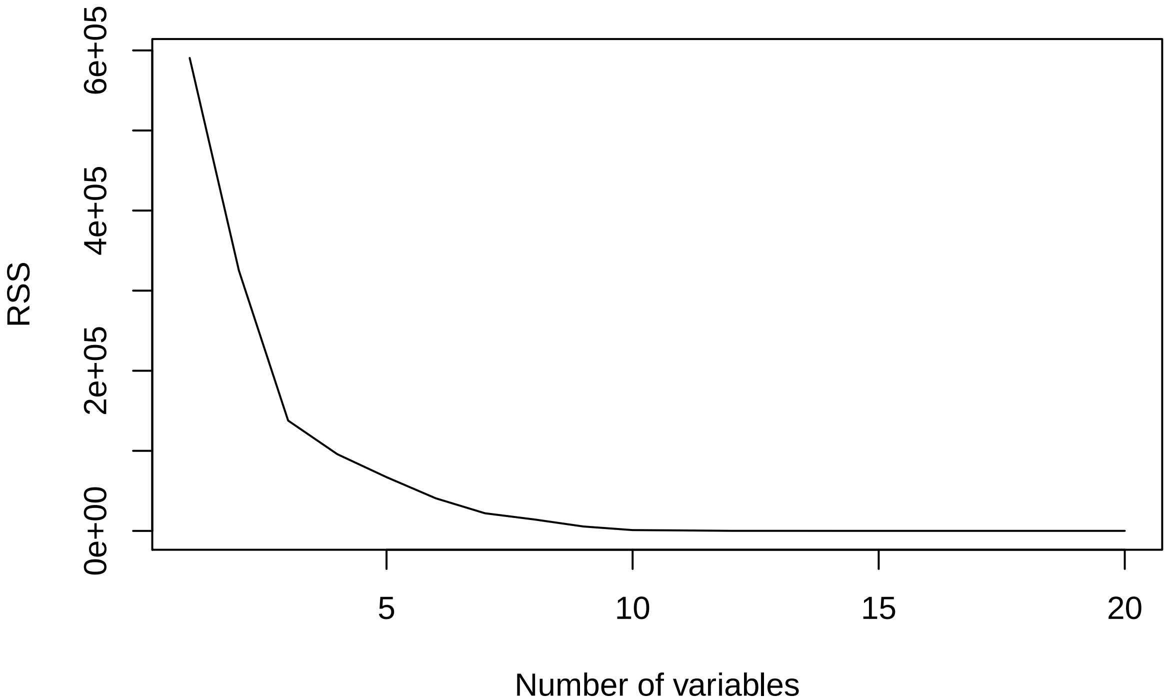
```
## [1] "b) Split the data into train setcontaining 100 and test set containing 900"
```

```r
train <- sample(1:nrow(X), nrow(X)/10)
test <- (-train)

train.x <- X[train, ]
train.y <- Y[train]

test.x <- X[test, ]
test.y <- Y[test]

df.train.X <- as_tibble(train.x)
df.train <- df.train.X %>% add_column(y = train.y, .before = "x1")
df.test.X <- as_tibble(test.x)
df.test <- df.test.X %>% add_column(y = test.y, .before = "x1")

print("c) Perform best subset selection on training set:")
```

```
## [1] "c) Perform best subset selection on training set:"
```

```r
regfit.full <- regsubsets(y ~ ., df.train, nvmax = 20)

#The summary shows the result of step 2 of algorithm 6.1 page 205 of the book
```

```r
summary <- summary(regfit.full)

plot(summary$rss, xlab = "Number of variables", ylab="RSS",type = "l")
```




```r
print("d) Plot test MSE for model of each size")
```

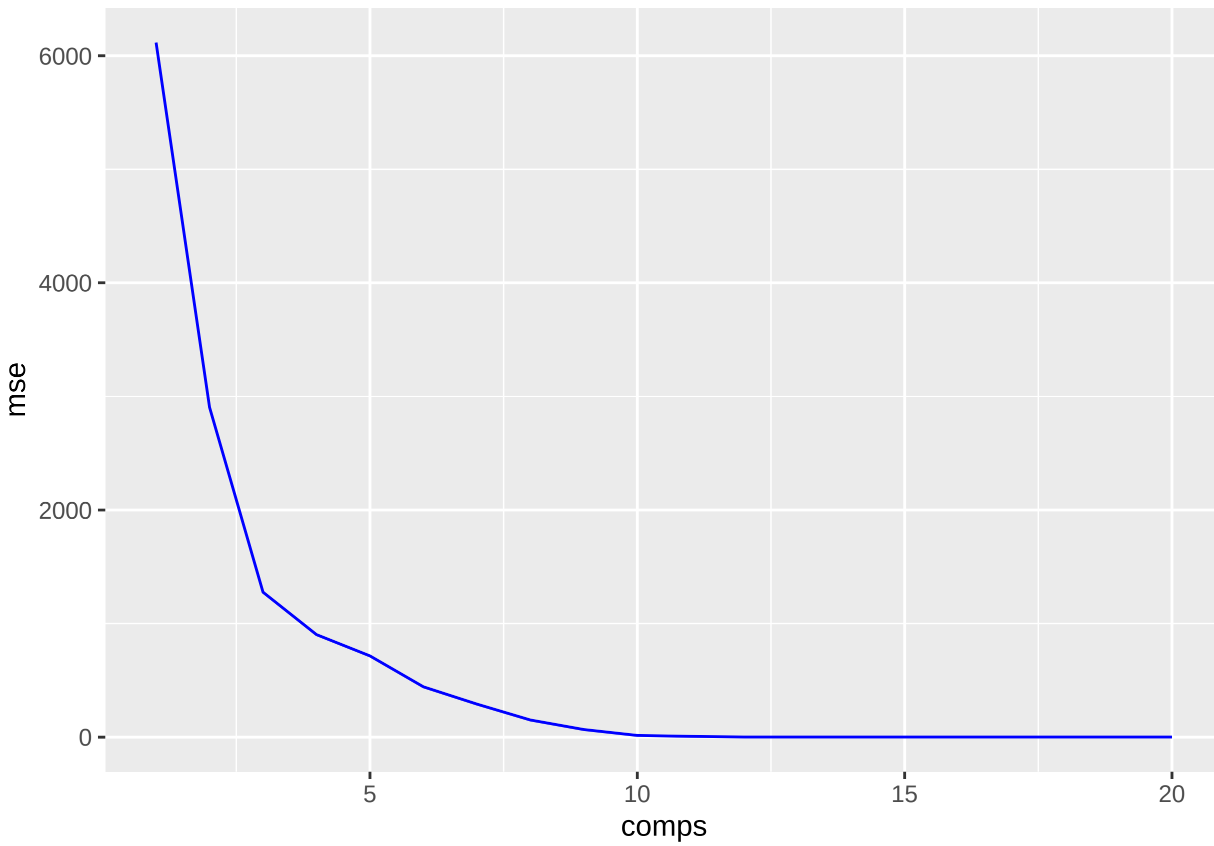
```
## [1] "d) Plot test MSE for model of each size"
```

```r
# First prepare test data as matrix, this is only for convinience
# when using dot product %*%: coefficients  %*% test.X rows
df.test.mat <- model.matrix(y~., data = df.test)

#  loop through all models , get the coefficients and calculate MSE
mses <- rep(NA, 20)

for (i in 1:20){
  coeffs <- coef(regfit.full, i)
  yhat <- df.test.mat[, names(coeffs)] %*% coeffs
  mses[i]  <- mean((test.y - yhat)^2)
}

(mse_data <- tibble(comps = 1:20, mse=mses))
```

```
## # A tibble: 20 x 2
##    comps    mse
##    <int>  <dbl>
##  1     1 6116.
##  2     2 2904.
##  3     3 1276.
##  4     4  903.
##  5     5  715.
##  6     6  443.
```

```
## 7       7  291.
## 8       8  151.
## 9       9   66.7
## 10     10   14.9
## 11     11    6.75
## 12     12    1.23
## 13     13    1.28
## 14     14    1.30
## 15     15    1.32
## 16     16    1.36
## 17     17    1.34
## 18     18    1.34
## 19     19    1.35
## 20     20    1.36
```

```r
ggplot(mse_data, aes(comps, mse))+
  geom_line(color="blue")
```




```r
print("e) For model sise with 12 components MSE is minimized: 12 --> 1.234475")
```

```
## [1] "e) For model sise with 12 components MSE is minimized: 12 --> 1.234475"
```

```r
print("f) How does the model at which the test MSE is moinimized? compare the coeffs")
```

```
## [1] "f) How does the model at which the test MSE is moinimized? compare the coeffs"
```

```r
(coeffs <- coef(regfit.full, 12))
```

```
##  (Intercept)           x1           x3           x4           x6           x7
```

```
##    0.1211867   11.9871248    2.6070986 -123.0928886   10.9639610   56.0180802
##          x8          x12          x13          x14          x16          x17
##   -6.9507468   13.1406636  -40.8673157    2.3919952    8.6684323  -17.9409391
##         x19
##   18.9294841
```

```
beta
```

```
##      x1      x2      x3      x4      x5      x6      x7      x8      x9     x10
##   12.00    0.00    2.60 -123.00    0.00   11.20   56.00   -7.00    0.00    0.00
##     x11     x12     x13     x14     x15     x16     x17     x18     x19     x20
##    0.00   13.00  -41.00    2.20    0.00    8.70  -18.00    0.00   19.00    0.03
```

```
print("comparision shows all zero coefficients successfully predicted and non zero
      coefficients are close approximation to real ones, only the 20th
      coefficient is missing ")
```

```
## [1] "comparision shows all zero coefficients successfully predicted and non zero\n      coefficients
```

```
print("
 (Intercept)        x1           x3           x4
  0.1211867  11.9871248    2.6070986 -123.0928886
                 12.00        2.60      -123.00

                 x6           x7           x8
             10.9639610   56.0180802   -6.9507468
                11.20       56.00         -7.00

                 x12          x13          x14
             13.1406636  -40.8673157    2.3919952
                 13.00      -41.00         2.20


                x16          x17          x19
             8.6684323  -17.9409391   18.9294841
               8.70       -18.00       19.00        0.03
")
```

```
## [1] "\n (Intercept)        x1           x3           x4         \n  0.1211867  11.9871248    2.60709
```

```
print("g) create a plot displaying ...")
```

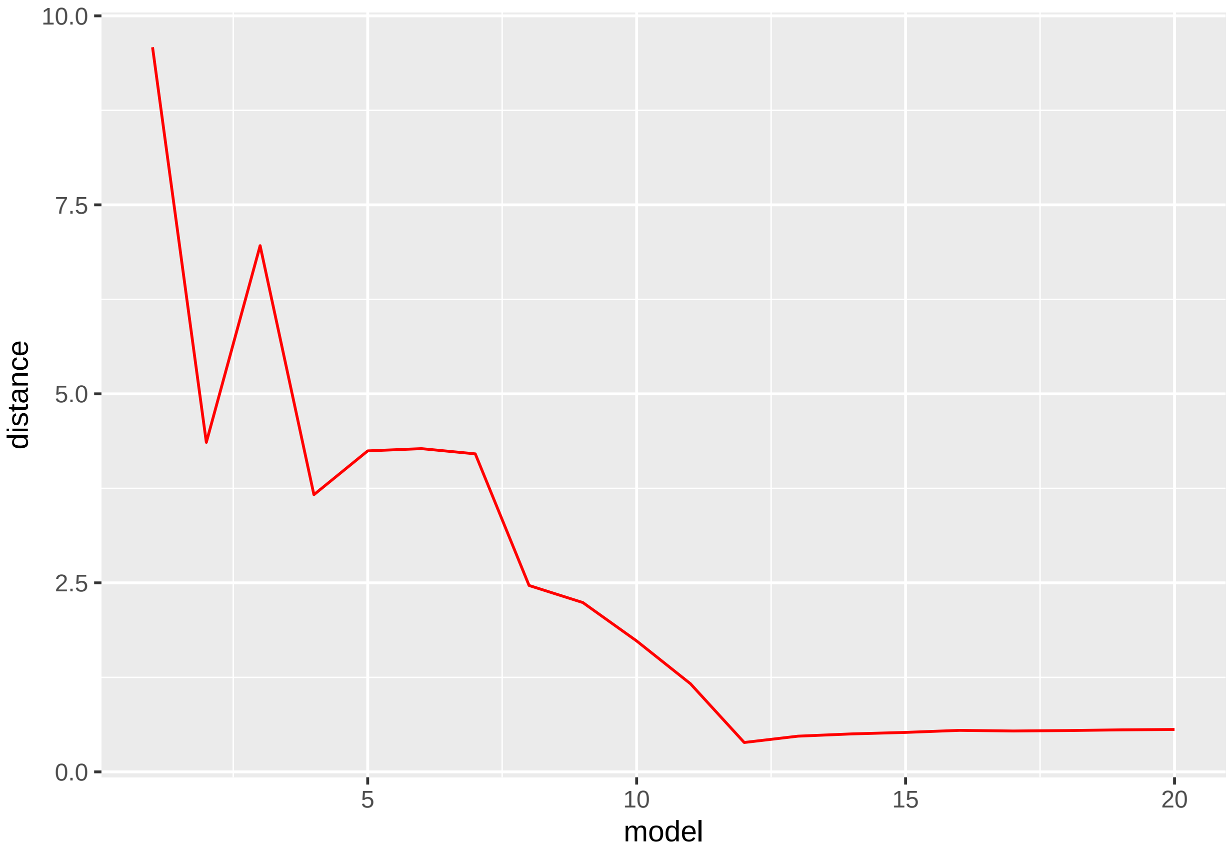
```
## [1] "g) create a plot displaying ..."
```

```
# loop over all models
distance.squared <- rep(0, 20)

for (i in 1:20){
  (coeffs <- coef(regfit.full, i)[-1]) # drop the intercept
  for (j in 1:i)
    (distance.squared[i] <- distance.squared[i] + (coeffs [j] - beta[names(coeffs [j])])^2 )
}
dist <- sqrt(distance.squared)
(distance.vs.model <- tibble(model = 1:20, distance = dist))
```

```
## # A tibble: 20 x 2
##    model distance
##    <int>    <dbl>
```

```
## 1    1    9.58
## 2    2    4.36
## 3    3    6.96
## 4    4    3.67
## 5    5    4.25
## 6    6    4.28
## 7    7    4.21
## 8    8    2.47
## 9    9    2.24
## 10  10    1.73
## 11  11    1.17
## 12  12    0.389
## 13  13    0.472
## 14  14    0.502
## 15  15    0.522
## 16  16    0.549
## 17  17    0.541
## 18  18    0.547
## 19  19    0.556
## 20  20    0.561
```

```r
ggplot(distance.vs.model, aes(model, distance))+
  geom_line(color="red")
```




```r
print("As graph shows on model with 12 components (the model we found in part d)
the distance between two sets of coefficients are minimum")
```

```
## [1] "As graph shows on model with 12 components (the model we found in part d) \nthe distance betwee
```

```r
library(tidyverse)
library(glmnet)
library(pls)
library(leaps)


boston.df = read.csv(
  "/Users/shahrdadshadab/env/my-R-project/ISLR/Data/datasets/BostonHousing.csv",
  header=T, stringsAsFactors = F, na.strings = "?")

str(boston.df)
```

```
## 'data.frame':    506 obs. of  14 variables:
##  $ crim   : num  0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  $ zn     : num  18 0 0 0 0 0 12.5 12.5 12.5 12.5 ...
##  $ indus  : num  2.31 7.07 7.07 2.18 2.18 2.18 7.87 7.87 7.87 7.87 ...
##  $ chas   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ nox    : num  0.538 0.469 0.469 0.458 0.458 0.458 0.524 0.524 0.524 0.524 ...
##  $ rm     : num  6.58 6.42 7.18 7 7.15 ...
##  $ age    : num  65.2 78.9 61.1 45.8 54.2 58.7 66.6 96.1 100 85.9 ...
##  $ dis    : num  4.09 4.97 4.97 6.06 6.06 ...
##  $ rad    : int  1 2 2 3 3 3 5 5 5 5 ...
##  $ tax    : int  296 242 242 222 222 222 311 311 311 311 ...
##  $ ptratio: num  15.3 17.8 17.8 18.7 18.7 18.7 15.2 15.2 15.2 15.2 ...
##  $ b      : num  397 397 393 395 397 ...
##  $ lstat  : num  4.98 9.14 4.03 2.94 5.33 ...
##  $ medv   : num  24 21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 ...
```

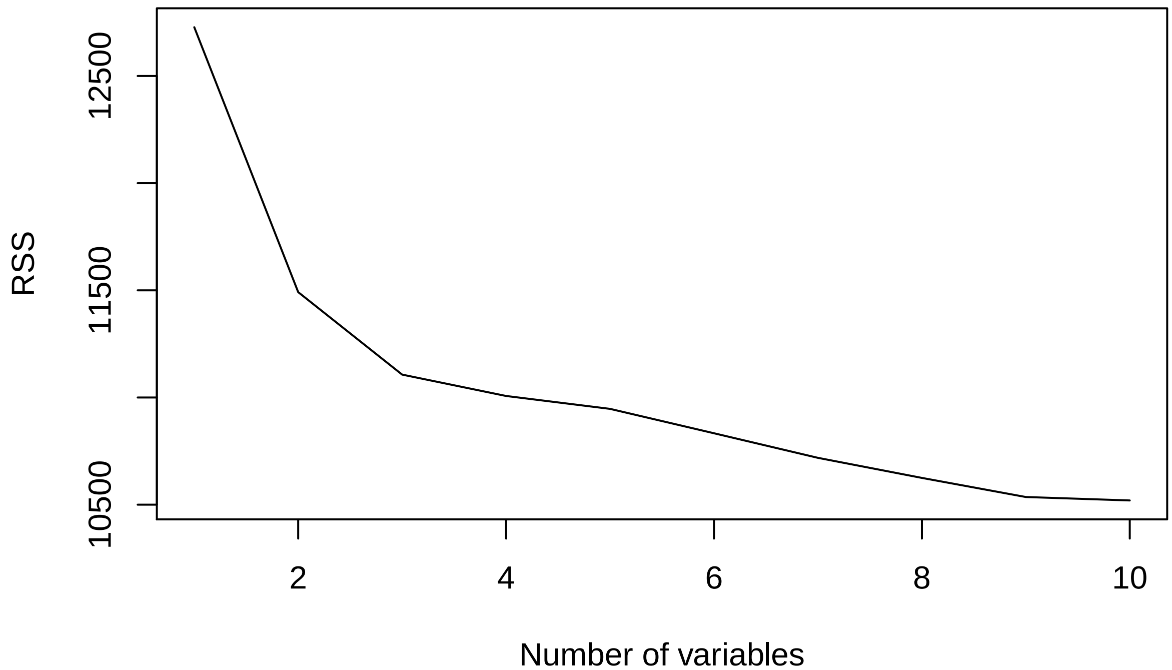
```r
# Find and remove NA in all columns
boston.df <- na.omit(boston.df)
set.seed(1)
print("Split into train / test data sets ---------------------")
```

```
## [1] "Split into train / test data sets ---------------------"
```

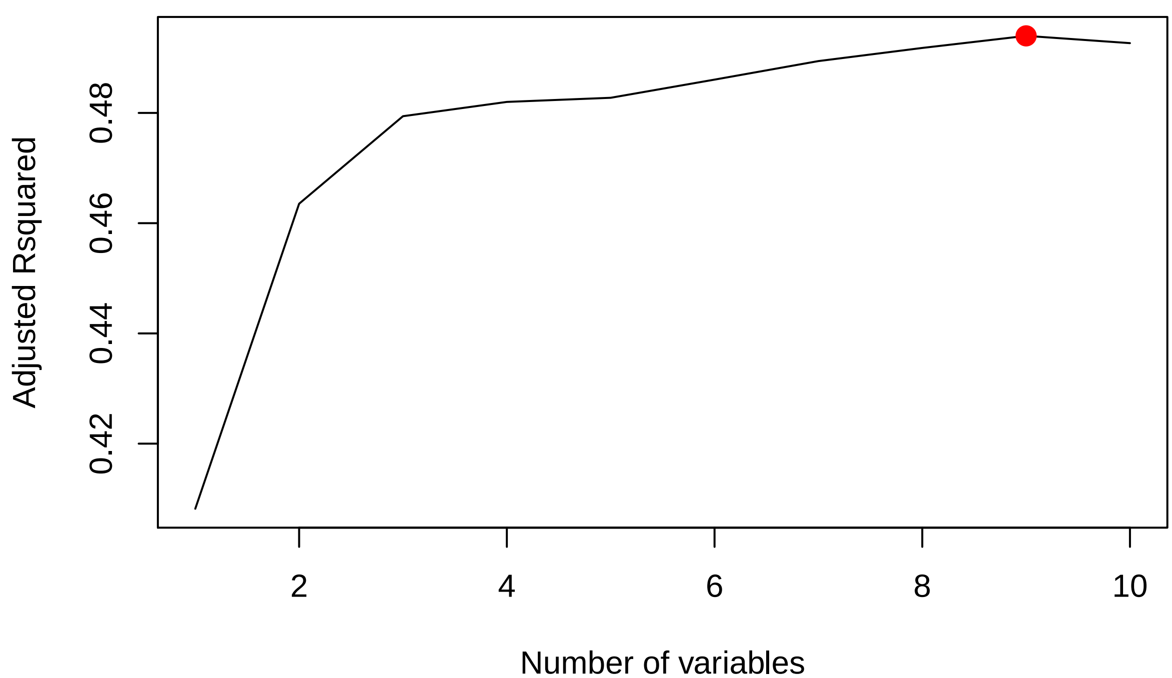
```r
train <- sample(1:nrow(boston.df), nrow(boston.df)/2)
test <- (-train)

df.train <- boston.df[train, ]
df.test <- boston.df[test, ]
train.y <- df.train$crim
test.y <- df.test$crim



print ("Perform best subset selection on train data: ")
```

```
## [1] "Perform best subset selection on train data: "
```

```r
regfit.full <- regsubsets(crim ~ ., df.train, nvmax = 10)

#The summary shows the result of step 2 of algorithm 6.1 page 205 of the book
summary <- summary(regfit.full)

plot(summary$rss, xlab = "Number of variables", ylab="RSS",type = "l")
```




```r
# which.max() returns location maximum point of the vector
index <- which.max(summary$adjr2)
plot(summary$adjr2,xlab = "Number of variables", ylab="Adjusted Rsquared",
     type = "l")
points(index, summary$adjr2[index], col="red", cex=2, pch=20)
```



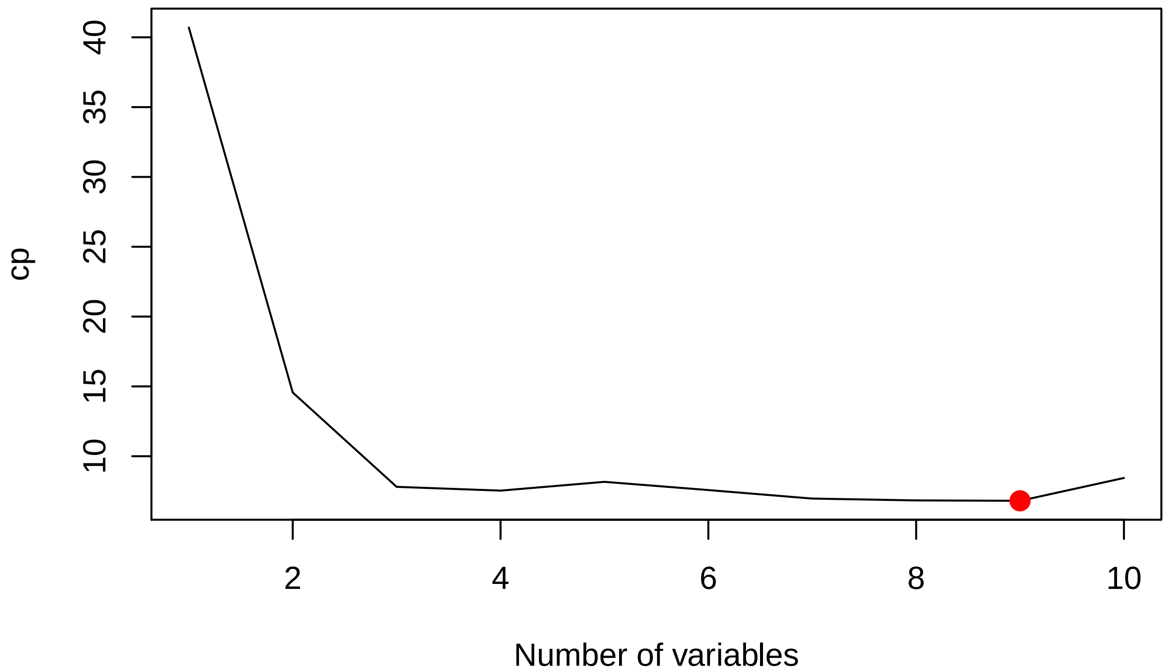

```r
print("coefficients of the best model (adjr2) : ")
```

```
## [1] "coefficients of the best model (adjr2) : "
```

```r
coef(regfit.full,index)
```

```
##  (Intercept)           zn        indus          nox          dis          rad
##  24.83928145   0.04131571  -0.15408884 -12.05853934  -1.05123802   0.55802734
##      ptratio            b        lstat         medv
##  -0.47520557  -0.01309050   0.25525000  -0.18608243
```

```r
# which.min() returns location minimum point of the vector
index <- which.min(summary$cp)
plot(summary$cp,xlab = "Number of variables", ylab="cp", type = "l")
points(index, summary$cp[index], col="red", cex=2, pch=20)
```




```r
print("coefficients of the best model (cp) : ")
```

```
## [1] "coefficients of the best model (cp) : "
```

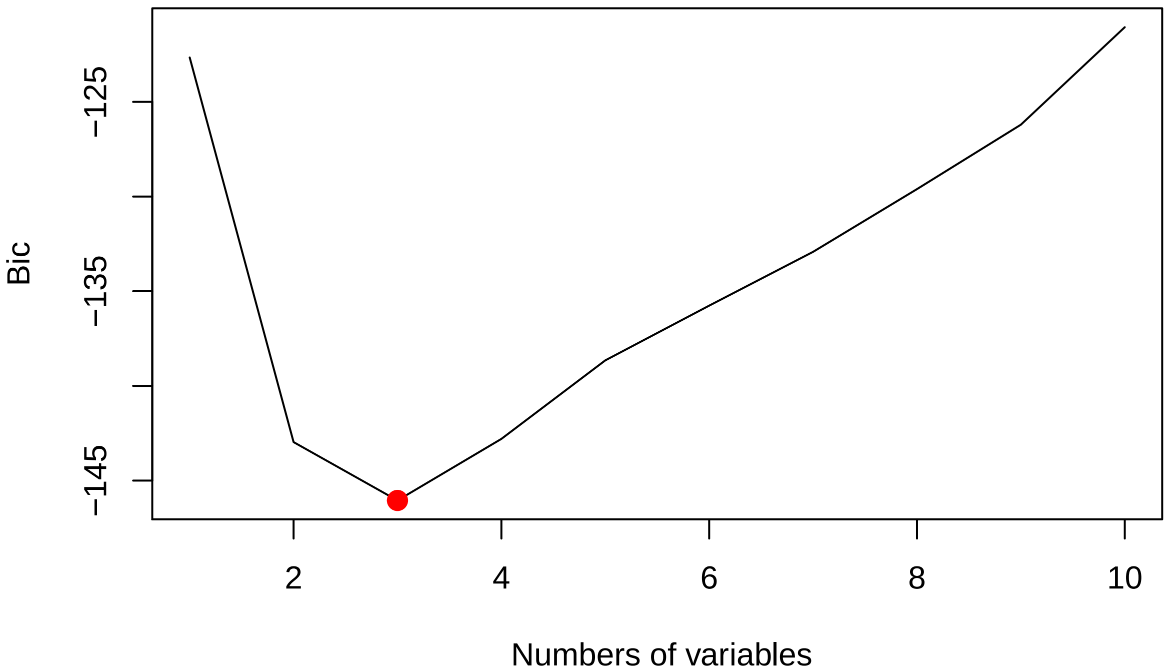
```r
coef(regfit.full,index)
```

```
##  (Intercept)           zn        indus          nox          dis          rad
##  24.83928145   0.04131571  -0.15408884 -12.05853934  -1.05123802   0.55802734
##      ptratio            b        lstat         medv
##  -0.47520557  -0.01309050   0.25525000  -0.18608243
```

```r
# same for bic
plot(summary$bic, xlab =" Numbers of variables", ylab="Bic", type="l")
(index <- which.min(summary$bic))
```

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

```r
points(index, summary$bic[index], col="red", cex=2, pch=20)
```




```r
print("coefficients of the best model (bic) : ")
```

```
## [1] "coefficients of the best model (bic) : "
```

```r
coef(regfit.full,index)
```

```
## (Intercept)         rad           b       lstat
##  0.72163825  0.46965678 -0.01530067  0.31662147
```

```r
print("looks a subset of three variables provides a model with acceptable performance")
```

```
## [1] "looks a subset of three variables provides a model with acceptable performance"
```

```r
print ("------ use CV with best subset selection  --------------" )
```

```
## [1] "------ use CV with best subset selection  --------------"
```

```r
k <- 10

folds <- sample(1:k, size = nrow(df), replace = T)
table(folds)
```

```
## folds
##  1  2  3  4  5  6  7  8  9 10
## 13  8 14 11  9  7 13  7 11  7
```

```r
# folds with same size
# sameSizefolds <- sample(rep(1:k, length.out = nrow(weekly.df)),
#   size = nrow(weekly.df), replace = F)
# table(sameSizefolds)

results <- lapply (1:k, function(x) {  # x is the index of test portion, the rest are for training

  # this is to collect the MSEs for each test fold
  mses <- tibble(no.of.coefs = NULL, MSE = NULL)
  df.train.cv <- df.train[folds != x, ]
```

```r
  df.test.cv <- df.train[folds == x, ]
  df.train.cv.X <- df.train.cv %>% select (-crim)
  df.test.cv.X <- df.test.cv %>% select (-crim)
  (mat.test.cv.X <- model.matrix(crim~., data=df.test.cv))
  (df.test.cv.Y <- df.test.cv$crim)

  # step 2 of algorithm 6.1 page 205 of the book
  regfit.full.train <- regsubsets(crim ~ ., df.train.cv,
                                  nvmax = ncol(df.train.cv.X))

  # apply the model with selected subsets on test set
  # one at a time and cacluate the MSE on test fold
  for (i in 1:ncol(df.test.cv.X)){
    (coefi <- coef(regfit.full.train, id = i))
    (pred <- mat.test.cv.X[, names(coefi)] %*% coefi)
    (mse <- mean((pred - df.test.cv.Y)^2))
     mses <- rbind (mses, tibble(no.of.coefs = i, MSE = mse))
  }
  return(mses)
})

allResults <- results[[1]]
for (i in 2 : length(results)){
  allResults <- rbind(allResults , results[[i]])
}

(allMse <- (allResults %>%
  group_by(no.of.coefs) %>%
  summarise(mse.mean = mean(MSE))) )
```

```
## # A tibble: 13 x 2
##    no.of.coefs mse.mean
##          <int>    <dbl>
##  1           1     51.0
##  2           2     49.2
##  3           3     47.9
##  4           4     47.8
##  5           5     48.4
##  6           6     47.5
##  7           7     47.2
##  8           8     46.6
##  9           9     45.9
## 10          10     46.0
## 11          11     45.6
## 12          12     45.6
## 13          13     45.5
```

```r
print ("The best subset of features selected correspnd to minimum CV_MSE:")
```

```
## [1] "The best subset of features selected correspnd to minimum CV_MSE:"
```

```r
(idx <- which.min(allMse$mse.mean))
```

```
## [1] 13
```

```r
print("However looking at the data shows sebset of features with 9 components
      is very close to the minimum we found so we use it")
```

```
## [1] "However looking at the data shows sebset of features with 9 components \n      is very close to
```

```r
M=9

regfit.full.train <- regsubsets(crim ~ ., df.train,
                                nvmax = ncol(df.train) - 1)

sprintf(" Now apply the the model with %s No. of components to test data:", idx)
```

```
## [1] " Now apply the the model with 13 No. of components to test data:"
```

```r
(coefi <- coef(regfit.full.train, id = M))
```

```
##  (Intercept)           zn        indus          nox          dis          rad
##  24.83928145   0.04131571  -0.15408884 -12.05853934  -1.05123802   0.55802734
##      ptratio            b        lstat         medv
##  -0.47520557  -0.01309050   0.25525000  -0.18608243
```

```r
(mat.df.test <- model.matrix(crim~., df.test))
```

```
##     (Intercept)    zn indus chas    nox    rm   age     dis rad tax ptratio
## 2             1   0.0  7.07    0 0.4690 6.421  78.9  4.9671   2 242    17.8
## 3             1   0.0  7.07    0 0.4690 7.185  61.1  4.9671   2 242    17.8
## 4             1   0.0  2.18    0 0.4580 6.998  45.8  6.0622   3 222    18.7
## 5             1   0.0  2.18    0 0.4580 7.147  54.2  6.0622   3 222    18.7
## 6             1   0.0  2.18    0 0.4580 6.430  58.7  6.0622   3 222    18.7
## 7             1  12.5  7.87    0 0.5240 6.012  66.6  5.5605   5 311    15.2
## 8             1  12.5  7.87    0 0.5240 6.172  96.1  5.9505   5 311    15.2
## 9             1  12.5  7.87    0 0.5240 5.631 100.0  6.0821   5 311    15.2
## 10            1  12.5  7.87    0 0.5240 6.004  85.9  6.5921   5 311    15.2
## 11            1  12.5  7.87    0 0.5240 6.377  94.3  6.3467   5 311    15.2
## 12            1  12.5  7.87    0 0.5240 6.009  82.9  6.2267   5 311    15.2
## 17            1   0.0  8.14    0 0.5380 5.935  29.3  4.4986   4 307    21.0
## 18            1   0.0  8.14    0 0.5380 5.990  81.7  4.2579   4 307    21.0
## 21            1   0.0  8.14    0 0.5380 5.570  98.1  3.7979   4 307    21.0
## 23            1   0.0  8.14    0 0.5380 6.142  91.7  3.9769   4 307    21.0
## 26            1   0.0  8.14    0 0.5380 5.599  85.7  4.4546   4 307    21.0
## 30            1   0.0  8.14    0 0.5380 6.674  87.3  4.2390   4 307    21.0
## 32            1   0.0  8.14    0 0.5380 6.072 100.0  4.1750   4 307    21.0
## 34            1   0.0  8.14    0 0.5380 5.701  95.0  3.7872   4 307    21.0
## 38            1   0.0  5.96    0 0.4990 5.850  41.5  3.9342   5 279    19.2
## 46            1   0.0  6.91    0 0.4480 5.682  33.8  5.1004   3 233    17.9
## 47            1   0.0  6.91    0 0.4480 5.786  33.3  5.1004   3 233    17.9
## 52            1  21.0  5.64    0 0.4390 6.115  63.0  6.8147   4 243    16.8
## 54            1  21.0  5.64    0 0.4390 5.998  21.4  6.8147   4 243    16.8
## 55            1  75.0  4.00    0 0.4100 5.888  47.6  7.3197   3 469    21.1
## 56            1  90.0  1.22    0 0.4030 7.249  21.9  8.6966   5 226    17.9
## 57            1  85.0  0.74    0 0.4100 6.383  35.7  9.1876   2 313    17.3
## 58            1 100.0  1.32    0 0.4110 6.816  40.5  8.3248   5 256    15.1
## 59            1  25.0  5.13    0 0.4530 6.145  29.2  7.8148   8 284    19.7
## 63            1  25.0  5.13    0 0.4530 6.456  67.8  7.2255   8 284    19.7
## 66            1  80.0  3.37    0 0.3980 6.290  17.8  6.6115   4 337    16.1
## 67            1  80.0  3.37    0 0.3980 5.787  31.1  6.6115   4 337    16.1
```

```
## 68          1  12.5  6.07    0 0.4090 5.878  21.4  6.4980   4 345    18.9
## 69          1  12.5  6.07    0 0.4090 5.594  36.8  6.4980   4 345    18.9
## 71          1   0.0 10.81    0 0.4130 6.417   6.6  5.2873   4 305    19.2
## 74          1   0.0 10.81    0 0.4130 6.245   6.2  5.2873   4 305    19.2
## 76          1   0.0 12.83    0 0.4370 6.286  45.0  4.5026   5 398    18.7
## 80          1   0.0 12.83    0 0.4370 5.874  36.6  4.5026   5 398    18.7
## 81          1  25.0  4.86    0 0.4260 6.727  33.5  5.4007   4 281    19.0
## 82          1  25.0  4.86    0 0.4260 6.619  70.4  5.4007   4 281    19.0
## 83          1  25.0  4.86    0 0.4260 6.302  32.2  5.4007   4 281    19.0
## 87          1   0.0  4.49    0 0.4490 6.015  45.1  4.4272   3 247    18.5
## 88          1   0.0  4.49    0 0.4490 6.121  56.8  3.7476   3 247    18.5
## 90          1   0.0  3.41    0 0.4890 7.079  63.1  3.4145   2 270    17.8
## 91          1   0.0  3.41    0 0.4890 6.417  66.1  3.0923   2 270    17.8
## 94          1  28.0 15.04    0 0.4640 6.211  28.9  3.6659   4 270    18.2
## 95          1  28.0 15.04    0 0.4640 6.249  77.3  3.6150   4 270    18.2
## 96          1   0.0  2.89    0 0.4450 6.625  57.8  3.4952   2 276    18.0
## 97          1   0.0  2.89    0 0.4450 6.163  69.6  3.4952   2 276    18.0
## 99          1   0.0  2.89    0 0.4450 7.820  36.9  3.4952   2 276    18.0
## 100         1   0.0  2.89    0 0.4450 7.416  62.5  3.4952   2 276    18.0
## 101         1   0.0  8.56    0 0.5200 6.727  79.9  2.7778   5 384    20.9
## 106         1   0.0  8.56    0 0.5200 5.851  96.7  2.1069   5 384    20.9
## 109         1   0.0  8.56    0 0.5200 6.474  97.1  2.4329   5 384    20.9
## 112         1   0.0 10.01    0 0.5470 6.715  81.6  2.6775   6 432    17.8
## 114         1   0.0 10.01    0 0.5470 6.092  95.4  2.5480   6 432    17.8
## 115         1   0.0 10.01    0 0.5470 6.254  84.2  2.2565   6 432    17.8
## 119         1   0.0 10.01    0 0.5470 5.872  73.1  2.4775   6 432    17.8
## 120         1   0.0 10.01    0 0.5470 5.731  65.2  2.7592   6 432    17.8
## 123         1   0.0 25.65    0 0.5810 5.961  92.9  2.0869   2 188    19.1
## 125         1   0.0 25.65    0 0.5810 5.879  95.8  2.0063   2 188    19.1
## 128         1   0.0 21.89    0 0.6240 5.693  96.0  1.7883   4 437    21.2
## 131         1   0.0 21.89    0 0.6240 6.458  98.9  2.1185   4 437    21.2
## 134         1   0.0 21.89    0 0.6240 5.822  95.4  2.4699   4 437    21.2
## 136         1   0.0 21.89    0 0.6240 6.335  98.2  2.1107   4 437    21.2
## 137         1   0.0 21.89    0 0.6240 5.942  93.5  1.9669   4 437    21.2
## 139         1   0.0 21.89    0 0.6240 5.857  98.2  1.6686   4 437    21.2
## 142         1   0.0 21.89    0 0.6240 5.019 100.0  1.4394   4 437    21.2
## 144         1   0.0 19.58    0 0.8710 5.468 100.0  1.4118   5 403    14.7
## 146         1   0.0 19.58    0 0.8710 6.130 100.0  1.4191   5 403    14.7
## 147         1   0.0 19.58    0 0.8710 5.628 100.0  1.5166   5 403    14.7
## 150         1   0.0 19.58    0 0.8710 5.597  94.9  1.5257   5 403    14.7
## 151         1   0.0 19.58    0 0.8710 6.122  97.3  1.6180   5 403    14.7
## 154         1   0.0 19.58    0 0.8710 5.709  98.5  1.6232   5 403    14.7
## 155         1   0.0 19.58    1 0.8710 6.129  96.0  1.7494   5 403    14.7
## 156         1   0.0 19.58    1 0.8710 6.152  82.6  1.7455   5 403    14.7
## 157         1   0.0 19.58    0 0.8710 5.272  94.0  1.7364   5 403    14.7
## 158         1   0.0 19.58    0 0.6050 6.943  97.4  1.8773   5 403    14.7
## 159         1   0.0 19.58    0 0.6050 6.066 100.0  1.7573   5 403    14.7
## 161         1   0.0 19.58    1 0.6050 6.250  92.6  1.7984   5 403    14.7
## 162         1   0.0 19.58    0 0.6050 7.489  90.8  1.9709   5 403    14.7
## 164         1   0.0 19.58    1 0.6050 8.375  93.9  2.1620   5 403    14.7
## 166         1   0.0 19.58    0 0.6050 6.101  93.0  2.2834   5 403    14.7
## 169         1   0.0 19.58    0 0.6050 6.319  96.1  2.1000   5 403    14.7
## 171         1   0.0 19.58    0 0.6050 5.875  94.6  2.4259   5 403    14.7
## 173         1   0.0  4.05    0 0.5100 5.572  88.5  2.5961   5 296    16.6
```

```
## 175            1   0.0  4.05    0 0.5100 5.859  68.7  2.7019    5 296     16.6
## 177            1   0.0  4.05    0 0.5100 6.020  47.2  3.5549    5 296     16.6
## 178            1   0.0  4.05    0 0.5100 6.315  73.4  3.3175    5 296     16.6
## 179            1   0.0  4.05    0 0.5100 6.860  74.4  2.9153    5 296     16.6
## 180            1   0.0  2.46    0 0.4880 6.980  58.4  2.8290    3 193     17.8
## 182            1   0.0  2.46    0 0.4880 6.144  62.2  2.5979    3 193     17.8
## 183            1   0.0  2.46    0 0.4880 7.155  92.2  2.7006    3 193     17.8
## 184            1   0.0  2.46    0 0.4880 6.563  95.6  2.8470    3 193     17.8
## 186            1   0.0  2.46    0 0.4880 6.153  68.8  3.2797    3 193     17.8
## 188            1  45.0  3.44    0 0.4370 6.782  41.1  3.7886    5 398     15.2
## 189            1  45.0  3.44    0 0.4370 6.556  29.1  4.5667    5 398     15.2
## 190            1  45.0  3.44    0 0.4370 7.185  38.9  4.5667    5 398     15.2
## 191            1  45.0  3.44    0 0.4370 6.951  21.5  6.4798    5 398     15.2
## 192            1  45.0  3.44    0 0.4370 6.739  30.8  6.4798    5 398     15.2
## 195            1  60.0  2.93    0 0.4010 6.604  18.8  6.2196    1 265     15.6
## 196            1  80.0  0.46    0 0.4220 7.875  32.0  5.6484    4 255     14.4
## 197            1  80.0  1.52    0 0.4040 7.287  34.1  7.3090    2 329     12.6
## 199            1  80.0  1.52    0 0.4040 7.274  38.3  7.3090    2 329     12.6
## 200            1  95.0  1.47    0 0.4030 6.975  15.3  7.6534    3 402     17.0
## 201            1  95.0  1.47    0 0.4030 7.135  13.9  7.6534    3 402     17.0
## 203            1  82.5  2.03    0 0.4150 7.610  15.7  6.2700    2 348     14.7
## 205            1  95.0  2.68    0 0.4161 8.034  31.9  5.1180    4 224     14.7
## 208            1   0.0 10.59    0 0.4890 5.783  72.7  4.3549    4 277     18.6
## 209            1   0.0 10.59    1 0.4890 6.064  59.1  4.2392    4 277     18.6
## 210            1   0.0 10.59    1 0.4890 5.344 100.0  3.8750    4 277     18.6
## 211            1   0.0 10.59    1 0.4890 5.960  92.1  3.8771    4 277     18.6
## 215            1   0.0 10.59    0 0.4890 5.412   9.8  3.5875    4 277     18.6
## 216            1   0.0 10.59    0 0.4890 6.182  42.4  3.9454    4 277     18.6
## 220            1   0.0 13.89    1 0.5500 6.373  92.4  3.3633    5 276     16.4
## 222            1   0.0  6.20    1 0.5070 6.164  91.3  3.0480    8 307     17.4
## 223            1   0.0  6.20    1 0.5070 6.879  77.7  3.2721    8 307     17.4
## 226            1   0.0  6.20    0 0.5040 8.725  83.0  2.8944    8 307     17.4
## 227            1   0.0  6.20    0 0.5040 8.040  86.5  3.2157    8 307     17.4
## 228            1   0.0  6.20    0 0.5040 7.163  79.9  3.2157    8 307     17.4
## 232            1   0.0  6.20    0 0.5040 7.412  76.9  3.6715    8 307     17.4
## 235            1   0.0  6.20    1 0.5070 6.726  66.5  3.6519    8 307     17.4
## 236            1   0.0  6.20    0 0.5070 6.086  61.5  3.6519    8 307     17.4
## 238            1   0.0  6.20    0 0.5070 7.358  71.6  4.1480    8 307     17.4
## 240            1  30.0  4.93    0 0.4280 6.606  42.2  6.1899    6 300     16.6
## 243            1  30.0  4.93    0 0.4280 6.358  52.9  7.0355    6 300     16.6
## 244            1  30.0  4.93    0 0.4280 6.393   7.8  7.0355    6 300     16.6
## 245            1  22.0  5.86    0 0.4310 5.593  76.5  7.9549    7 330     19.1
## 250            1  22.0  5.86    0 0.4310 6.718  17.5  7.8265    7 330     19.1
## 251            1  22.0  5.86    0 0.4310 6.487  13.0  7.3967    7 330     19.1
## 253            1  22.0  5.86    0 0.4310 6.957   6.8  8.9067    7 330     19.1
## 256            1  80.0  3.64    0 0.3920 5.876  19.1  9.2203    1 315     16.4
## 257            1  90.0  3.75    0 0.3940 7.454  34.2  6.3361    3 244     15.9
## 258            1  20.0  3.97    0 0.6470 8.704  86.9  1.8010    5 264     13.0
## 259            1  20.0  3.97    0 0.6470 7.333 100.0  1.8946    5 264     13.0
## 260            1  20.0  3.97    0 0.6470 6.842 100.0  2.0107    5 264     13.0
## 261            1  20.0  3.97    0 0.6470 7.203  81.8  2.1121    5 264     13.0
## 262            1  20.0  3.97    0 0.6470 7.520  89.4  2.1398    5 264     13.0
## 264            1  20.0  3.97    0 0.6470 7.327  94.5  2.0788    5 264     13.0
## 266            1  20.0  3.97    0 0.6470 5.560  62.8  1.9865    5 264     13.0
```

```
## 267            1 20.0  3.97    0 0.6470 7.014  84.6  2.1329   5 264     13.0
## 268            1 20.0  3.97    0 0.5750 8.297  67.0  2.4216   5 264     13.0
## 269            1 20.0  3.97    0 0.5750 7.470  52.6  2.8720   5 264     13.0
## 272            1 20.0  6.96    0 0.4640 6.240  16.3  4.4290   3 223     18.6
## 276            1 40.0  6.41    0 0.4470 6.854  42.8  4.2673   4 254     17.6
## 278            1 40.0  6.41    1 0.4470 6.826  27.6  4.8628   4 254     17.6
## 283            1 20.0  3.33    1 0.4429 7.645  49.7  5.2119   5 216     14.9
## 288            1 52.5  5.32    0 0.4050 6.209  31.3  7.3172   6 293     16.6
## 291            1 80.0  4.95    0 0.4110 6.861  27.9  5.1167   4 245     19.2
## 292            1 80.0  4.95    0 0.4110 7.148  27.7  5.1167   4 245     19.2
## 294            1  0.0 13.92    0 0.4370 6.127  18.4  5.5027   4 289     16.0
## 301            1 70.0  2.24    0 0.4000 6.871  47.4  7.8278   5 358     14.8
## 302            1 34.0  6.09    0 0.4330 6.590  40.4  5.4917   7 329     16.1
## 303            1 34.0  6.09    0 0.4330 6.495  18.4  5.4917   7 329     16.1
## 308            1 33.0  2.18    0 0.4720 6.849  70.3  3.1827   7 222     18.4
## 309            1  0.0  9.90    0 0.5440 6.635  82.5  3.3175   4 304     18.4
## 310            1  0.0  9.90    0 0.5440 5.972  76.7  3.1025   4 304     18.4
## 311            1  0.0  9.90    0 0.5440 4.973  37.8  2.5194   4 304     18.4
## 312            1  0.0  9.90    0 0.5440 6.122  52.8  2.6403   4 304     18.4
## 314            1  0.0  9.90    0 0.5440 6.266  82.8  3.2628   4 304     18.4
## 315            1  0.0  9.90    0 0.5440 6.567  87.3  3.6023   4 304     18.4
## 317            1  0.0  9.90    0 0.5440 5.914  83.2  3.9986   4 304     18.4
## 318            1  0.0  9.90    0 0.5440 5.782  71.7  4.0317   4 304     18.4
## 319            1  0.0  9.90    0 0.5440 6.382  67.2  3.5325   4 304     18.4
## 320            1  0.0  9.90    0 0.5440 6.113  58.8  4.0019   4 304     18.4
## 321            1  0.0  7.38    0 0.4930 6.426  52.3  4.5404   5 287     19.6
## 322            1  0.0  7.38    0 0.4930 6.376  54.3  4.5404   5 287     19.6
## 323            1  0.0  7.38    0 0.4930 6.041  49.9  4.7211   5 287     19.6
## 334            1  0.0  5.19    0 0.5150 6.316  38.1  6.4584   5 224     20.2
## 335            1  0.0  5.19    0 0.5150 6.310  38.5  6.4584   5 224     20.2
## 337            1  0.0  5.19    0 0.5150 5.869  46.3  5.2311   5 224     20.2
## 341            1  0.0  5.19    0 0.5150 5.968  58.5  4.8122   5 224     20.2
## 347            1  0.0  4.39    0 0.4420 5.898  52.3  8.0136   3 352     18.8
## 348            1 85.0  4.15    0 0.4290 6.516  27.7  8.5353   4 351     17.9
## 349            1 80.0  2.01    0 0.4350 6.635  29.7  8.3440   4 280     17.0
## 350            1 40.0  1.25    0 0.4290 6.939  34.5  8.7921   1 335     19.7
## 351            1 40.0  1.25    0 0.4290 6.490  44.4  8.7921   1 335     19.7
## 352            1 60.0  1.69    0 0.4110 6.579  35.9 10.7103   4 411     18.3
## 354            1 90.0  2.02    0 0.4100 6.728  36.1 12.1265   5 187     17.0
## 356            1 80.0  1.91    0 0.4130 5.936  19.5 10.5857   4 334     22.0
## 357            1  0.0 18.10    1 0.7700 6.212  97.4  2.1222  24 666     20.2
## 358            1  0.0 18.10    1 0.7700 6.395  91.0  2.5052  24 666     20.2
## 361            1  0.0 18.10    0 0.7700 6.398  88.0  2.5182  24 666     20.2
## 363            1  0.0 18.10    0 0.7700 5.362  96.2  2.1036  24 666     20.2
## 365            1  0.0 18.10    1 0.7180 8.780  82.9  1.9047  24 666     20.2
## 367            1  0.0 18.10    0 0.7180 4.963  91.4  1.7523  24 666     20.2
## 369            1  0.0 18.10    0 0.6310 4.970 100.0  1.3325  24 666     20.2
## 370            1  0.0 18.10    1 0.6310 6.683  96.8  1.3567  24 666     20.2
## 372            1  0.0 18.10    0 0.6310 6.216 100.0  1.1691  24 666     20.2
## 373            1  0.0 18.10    1 0.6680 5.875  89.6  1.1296  24 666     20.2
## 374            1  0.0 18.10    0 0.6680 4.906 100.0  1.1742  24 666     20.2
## 376            1  0.0 18.10    0 0.6710 7.313  97.9  1.3163  24 666     20.2
## 379            1  0.0 18.10    0 0.6710 6.380  96.2  1.3861  24 666     20.2
## 380            1  0.0 18.10    0 0.6710 6.223 100.0  1.3861  24 666     20.2
```

```
## 381            1   0.0 18.10    0 0.6710 6.968  91.9  1.4165  24 666     20.2
## 383            1   0.0 18.10    0 0.7000 5.536 100.0  1.5804  24 666     20.2
## 384            1   0.0 18.10    0 0.7000 5.520 100.0  1.5331  24 666     20.2
## 385            1   0.0 18.10    0 0.7000 4.368  91.2  1.4395  24 666     20.2
## 386            1   0.0 18.10    0 0.7000 5.277  98.1  1.4261  24 666     20.2
## 387            1   0.0 18.10    0 0.7000 4.652 100.0  1.4672  24 666     20.2
## 388            1   0.0 18.10    0 0.7000 5.000  89.5  1.5184  24 666     20.2
## 393            1   0.0 18.10    0 0.7000 5.036  97.0  1.7700  24 666     20.2
## 394            1   0.0 18.10    0 0.6930 6.193  92.6  1.7912  24 666     20.2
## 397            1   0.0 18.10    0 0.6930 6.405  96.0  1.6768  24 666     20.2
## 398            1   0.0 18.10    0 0.6930 5.747  98.9  1.6334  24 666     20.2
## 400            1   0.0 18.10    0 0.6930 5.852  77.8  1.5004  24 666     20.2
## 403            1   0.0 18.10    0 0.6930 6.404 100.0  1.6390  24 666     20.2
## 407            1   0.0 18.10    0 0.6590 4.138 100.0  1.1781  24 666     20.2
## 409            1   0.0 18.10    0 0.5970 5.617  97.9  1.4547  24 666     20.2
## 410            1   0.0 18.10    0 0.5970 6.852 100.0  1.4655  24 666     20.2
## 411            1   0.0 18.10    0 0.5970 5.757 100.0  1.4130  24 666     20.2
## 417            1   0.0 18.10    0 0.6790 6.782  90.8  1.8195  24 666     20.2
## 424            1   0.0 18.10    0 0.6140 6.103  85.1  2.0218  24 666     20.2
## 425            1   0.0 18.10    0 0.5840 5.565  70.6  2.0635  24 666     20.2
## 426            1   0.0 18.10    0 0.6790 5.896  95.4  1.9096  24 666     20.2
## 429            1   0.0 18.10    0 0.6790 6.193  78.1  1.9356  24 666     20.2
## 430            1   0.0 18.10    0 0.6790 6.380  95.6  1.9682  24 666     20.2
## 432            1   0.0 18.10    0 0.5840 6.833  94.3  2.0882  24 666     20.2
## 439            1   0.0 18.10    0 0.7400 5.935  87.9  1.8206  24 666     20.2
## 440            1   0.0 18.10    0 0.7400 5.627  93.9  1.8172  24 666     20.2
## 441            1   0.0 18.10    0 0.7400 5.818  92.4  1.8662  24 666     20.2
## 447            1   0.0 18.10    0 0.7400 6.341  96.4  2.0720  24 666     20.2
## 448            1   0.0 18.10    0 0.7400 6.251  96.6  2.1980  24 666     20.2
## 451            1   0.0 18.10    0 0.7130 6.749  92.6  2.3236  24 666     20.2
## 452            1   0.0 18.10    0 0.7130 6.655  98.2  2.3552  24 666     20.2
## 453            1   0.0 18.10    0 0.7130 6.297  91.8  2.3682  24 666     20.2
## 454            1   0.0 18.10    0 0.7130 7.393  99.3  2.4527  24 666     20.2
## 455            1   0.0 18.10    0 0.7130 6.728  94.1  2.4961  24 666     20.2
## 456            1   0.0 18.10    0 0.7130 6.525  86.5  2.4358  24 666     20.2
## 457            1   0.0 18.10    0 0.7130 5.976  87.9  2.5806  24 666     20.2
## 460            1   0.0 18.10    0 0.7130 6.081  84.4  2.7175  24 666     20.2
## 462            1   0.0 18.10    0 0.7130 6.376  88.4  2.5671  24 666     20.2
## 468            1   0.0 18.10    0 0.5840 6.003  94.5  2.5403  24 666     20.2
## 469            1   0.0 18.10    0 0.5800 5.926  71.0  2.9084  24 666     20.2
## 470            1   0.0 18.10    0 0.5800 5.713  56.7  2.8237  24 666     20.2
## 472            1   0.0 18.10    0 0.5320 6.229  90.7  3.0993  24 666     20.2
## 474            1   0.0 18.10    0 0.6140 6.980  67.6  2.5329  24 666     20.2
## 475            1   0.0 18.10    0 0.5840 5.427  95.4  2.4298  24 666     20.2
## 477            1   0.0 18.10    0 0.6140 6.484  93.6  2.3053  24 666     20.2
## 479            1   0.0 18.10    0 0.6140 6.185  96.7  2.1705  24 666     20.2
## 482            1   0.0 18.10    0 0.5320 6.750  74.9  3.3317  24 666     20.2
## 486            1   0.0 18.10    0 0.5830 6.312  51.9  3.9917  24 666     20.2
## 487            1   0.0 18.10    0 0.5830 6.114  79.8  3.5459  24 666     20.2
## 489            1   0.0 27.74    0 0.6090 5.454  92.7  1.8209   4 711     20.1
## 491            1   0.0 27.74    0 0.6090 5.093  98.0  1.8226   4 711     20.1
## 493            1   0.0 27.74    0 0.6090 5.983  83.5  2.1099   4 711     20.1
## 494            1   0.0  9.69    0 0.5850 5.707  54.0  2.3817   6 391     19.2
## 495            1   0.0  9.69    0 0.5850 5.926  42.6  2.3817   6 391     19.2
```

```
## 497           1   0.0  9.69    0 0.5850 5.390  72.9  2.7986   6 391    19.2
## 499           1   0.0  9.69    0 0.5850 6.019  65.3  2.4091   6 391    19.2
## 502           1   0.0 11.93    0 0.5730 6.593  69.1  2.4786   1 273    21.0
## 504           1   0.0 11.93    0 0.5730 6.976  91.0  2.1675   1 273    21.0
## 506           1   0.0 11.93    0 0.5730 6.030  80.8  2.5050   1 273    21.0
##          b lstat medv
## 2   396.90  9.14 21.6
## 3   392.83  4.03 34.7
## 4   394.63  2.94 33.4
## 5   396.90  5.33 36.2
## 6   394.12  5.21 28.7
## 7   395.60 12.43 22.9
## 8   396.90 19.15 27.1
## 9   386.63 29.93 16.5
## 10  386.71 17.10 18.9
## 11  392.52 20.45 15.0
## 12  396.90 13.27 18.9
## 17  386.85  6.58 23.1
## 18  386.75 14.67 17.5
## 21  376.57 21.02 13.6
## 23  396.90 18.72 15.2
## 26  303.42 16.51 13.9
## 30  380.23 11.98 21.0
## 32  376.73 13.04 14.5
## 34  358.77 18.35 13.1
## 38  396.90  8.77 21.0
## 46  396.90 10.21 19.3
## 47  396.90 14.15 20.0
## 52  393.97  9.43 20.5
## 54  396.90  8.43 23.4
## 55  396.90 14.80 18.9
## 56  395.93  4.81 35.4
## 57  396.90  5.77 24.7
## 58  392.90  3.95 31.6
## 59  390.68  6.86 23.3
## 63  396.90  6.73 22.2
## 66  396.90  4.67 23.5
## 67  396.90 10.24 19.4
## 68  396.21  8.10 22.0
## 69  396.90 13.09 17.4
## 71  383.73  6.72 24.2
## 74  377.17  7.54 23.4
## 76  383.23  8.94 21.4
## 80  396.06  9.10 20.3
## 81  396.90  5.29 28.0
## 82  395.63  7.22 23.9
## 83  396.90  6.72 24.8
## 87  395.99 12.86 22.5
## 88  395.15  8.44 22.2
## 90  396.06  5.70 28.7
## 91  392.18  8.81 22.6
## 94  396.33  6.21 25.0
## 95  396.90 10.59 20.6
## 96  357.98  6.65 28.4
```

```
## 97  391.83 11.34 21.4
## 99  393.53  3.57 43.8
## 100 396.90  6.19 33.2
## 101 394.76  9.42 27.5
## 106 394.05 16.47 19.5
## 109 395.24 12.27 19.8
## 112 395.59 10.16 22.8
## 114 396.90 17.09 18.7
## 115 388.74 10.45 18.5
## 119 338.63 15.37 20.4
## 120 391.50 13.61 19.3
## 123 378.09 17.93 20.5
## 125 379.38 17.58 18.8
## 128 392.11 17.19 16.2
## 131 395.04 12.60 19.2
## 134 388.69 15.03 18.4
## 136 394.67 16.96 18.1
## 137 378.25 16.90 17.4
## 139 392.04 21.32 13.3
## 142 396.90 34.41 14.4
## 144 396.90 26.42 15.6
## 146 172.91 27.80 13.8
## 147 169.27 16.65 15.6
## 150 351.85 21.45 15.4
## 151 372.80 14.10 21.5
## 154 261.95 15.79 19.4
## 155 321.02 15.12 17.0
## 156  88.01 15.02 15.6
## 157  88.63 16.14 13.1
## 158 363.43  4.59 41.3
## 159 353.89  6.43 24.3
## 161 338.92  5.50 27.0
## 162 374.43  1.73 50.0
## 164 388.45  3.32 50.0
## 166 240.16  9.81 25.0
## 169 297.09 11.10 23.8
## 171 292.29 14.43 17.4
## 173 396.90 14.69 23.1
## 175 393.23  9.64 22.6
## 177 393.23 10.11 23.2
## 178 395.60  6.29 24.6
## 179 391.27  6.92 29.9
## 180 396.90  5.04 37.2
## 182 396.90  9.45 36.2
## 183 394.12  4.82 37.9
## 184 396.90  5.68 32.5
## 186 387.11 13.15 29.6
## 188 393.87  6.68 32.0
## 189 382.84  4.56 29.8
## 190 396.90  5.39 34.9
## 191 377.68  5.10 37.0
## 192 389.71  4.69 30.5
## 195 376.70  4.38 29.1
## 196 394.23  2.97 50.0
```

```
## 197 396.90  4.08 33.3
## 199 392.20  6.62 34.6
## 200 396.90  4.56 34.9
## 201 384.30  4.45 32.9
## 203 395.38  3.11 42.3
## 205 390.55  2.88 50.0
## 208 389.43 18.06 22.5
## 209 381.32 14.66 24.4
## 210 396.90 23.09 20.0
## 211 393.25 17.27 21.7
## 215 348.93 29.55 23.7
## 216 393.63  9.47 25.0
## 220 393.74 10.50 23.0
## 222 395.24 21.46 21.7
## 223 390.39  9.93 27.5
## 226 382.00  4.63 50.0
## 227 387.38  3.13 37.6
## 228 372.08  6.36 31.6
## 232 376.14  5.25 31.7
## 235 360.20  8.05 29.0
## 236 376.75 10.88 24.0
## 238 390.07  4.73 31.5
## 240 383.78  7.37 23.3
## 243 372.75 11.22 22.2
## 244 374.71  5.19 23.7
## 245 372.49 12.50 17.6
## 250 393.74  6.56 26.2
## 251 396.28  5.90 24.4
## 253 386.09  3.53 29.6
## 256 395.18  9.25 20.9
## 257 386.34  3.11 44.0
## 258 389.70  5.12 50.0
## 259 383.29  7.79 36.0
## 260 391.93  6.90 30.1
## 261 392.80  9.59 33.8
## 262 388.37  7.26 43.1
## 264 393.42 11.25 31.0
## 266 392.40 10.45 22.8
## 267 384.07 14.79 30.7
## 268 384.54  7.44 50.0
## 269 390.30  3.16 43.5
## 272 396.90  6.59 25.2
## 276 396.90  2.98 32.0
## 278 393.45  4.16 33.1
## 283 377.07  3.01 46.0
## 288 396.90  7.14 23.2
## 291 396.90  3.33 28.5
## 292 396.90  3.56 37.3
## 294 396.90  8.58 23.9
## 301 390.86  6.07 24.8
## 302 395.75  9.50 22.0
## 303 383.61  8.67 26.4
## 308 396.90  7.53 28.2
## 309 396.90  4.54 22.8
```

```
## 310 396.24  9.97 20.3
## 311 350.45 12.64 16.1
## 312 396.90  5.98 22.1
## 314 393.39  7.90 21.6
## 315 395.69  9.28 23.8
## 317 390.70 18.33 17.8
## 318 396.90 15.94 19.8
## 319 395.21 10.36 23.1
## 320 396.23 12.73 21.0
## 321 396.90  7.20 23.8
## 322 396.90  6.87 23.1
## 323 396.90  7.70 20.4
## 334 389.71  5.68 22.2
## 335 389.40  6.75 20.7
## 337 396.90  9.80 19.5
## 341 396.90  9.29 18.7
## 347 364.61 12.67 17.2
## 348 392.43  6.36 23.1
## 349 390.94  5.99 24.5
## 350 389.85  5.89 26.6
## 351 396.90  5.98 22.9
## 352 370.78  5.49 24.1
## 354 384.46  4.50 30.1
## 356 376.04  5.57 20.6
## 357 377.73 17.60 17.8
## 358 391.34 13.27 21.7
## 361 374.56  7.79 25.0
## 363 380.79 10.19 20.8
## 365 354.55  5.29 21.9
## 367 316.03 14.00 21.9
## 369 375.52  3.26 50.0
## 370 375.33  3.73 50.0
## 372 366.15  9.53 50.0
## 373 347.88  8.88 50.0
## 374 396.90 34.77 13.8
## 376 396.90 13.44 15.0
## 379 396.90 23.69 13.1
## 380 393.74 21.78 10.2
## 381 396.90 17.21 10.4
## 383 396.90 23.60 11.3
## 384 396.90 24.56 12.3
## 385 285.83 30.63  8.8
## 386 396.90 30.81  7.2
## 387 396.90 28.28 10.5
## 388 396.90 31.99  7.4
## 393 396.90 25.68  9.7
## 394 396.90 15.17 13.8
## 397 396.90 19.37 12.5
## 398 393.10 19.92  8.5
## 400 338.16 29.97  6.3
## 403 376.11 20.31 12.1
## 407 370.22 23.34 11.9
## 409 314.64 26.40 17.2
## 410 179.36 19.78 27.5
```

```
## 411   2.60 10.11 15.0
## 417  21.57 25.79  7.5
## 424   2.52 23.29 13.4
## 425   3.65 17.16 11.7
## 426   7.68 24.39  8.3
## 429  96.73 21.52 11.0
## 430  60.72 24.08  9.5
## 432  81.33 19.69 14.1
## 439  68.95 34.02  8.4
## 440 396.90 22.88 12.8
## 441 391.45 22.11 10.5
## 447 318.01 17.79 14.9
## 448 388.52 16.44 12.6
## 451   0.32 17.44 13.4
## 452 355.29 17.73 15.2
## 453 385.09 17.27 16.1
## 454 375.87 16.74 17.8
## 455   6.68 18.71 14.9
## 456  50.92 18.13 14.1
## 457  10.48 19.01 12.7
## 460 396.90 14.70 20.0
## 462 391.43 14.65 17.7
## 468 331.29 21.32 19.1
## 469 368.74 18.13 19.1
## 470 396.90 14.76 20.1
## 472 395.33 12.87 19.6
## 474 374.68 11.66 29.8
## 475 352.58 18.14 13.8
## 477 396.21 18.68 16.7
## 479 379.70 18.03 14.6
## 482 393.07  7.74 23.7
## 486 388.62 10.58 21.2
## 487 392.68 14.98 19.1
## 489 395.09 18.06 15.2
## 491 318.43 29.68  8.1
## 493 396.90 13.35 20.1
## 494 396.90 12.01 21.8
## 495 396.90 13.59 24.5
## 497 396.90 21.14 19.7
## 499 396.90 12.92 21.2
## 502 391.99  9.67 22.4
## 504 396.90  5.64 23.9
## 506 396.90  7.88 11.9
## attr(,"assign")
##  [1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13
```

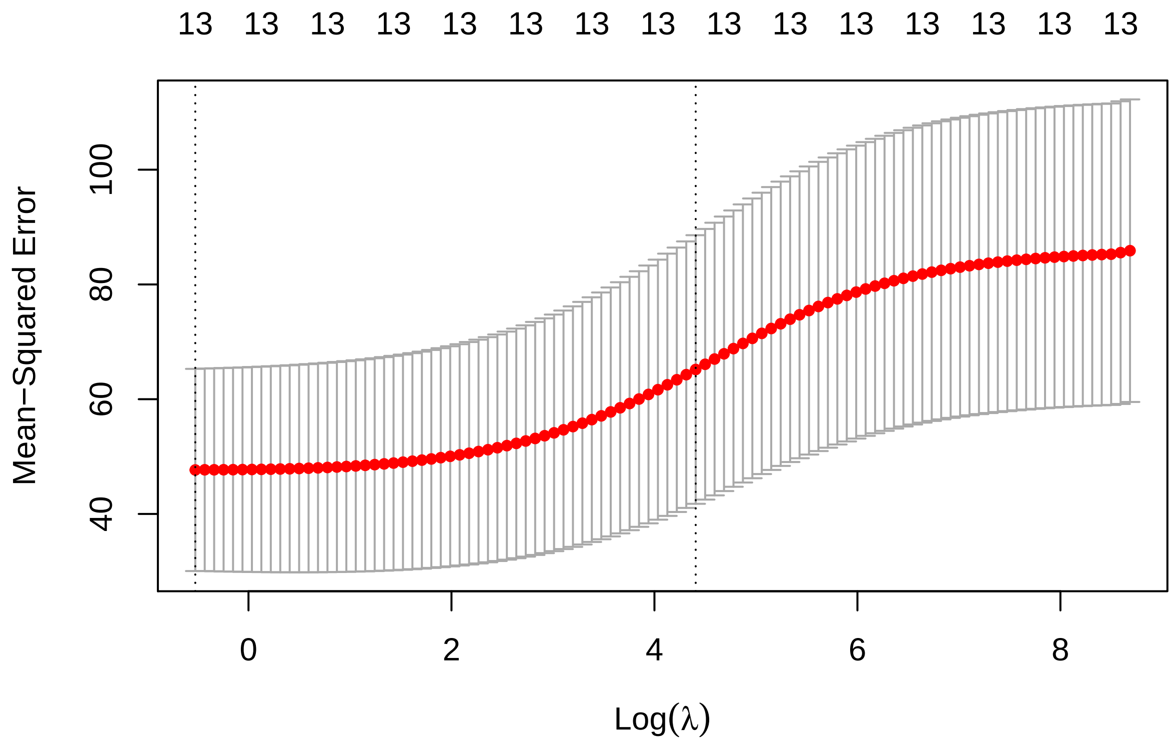
```r
yhat <- mat.df.test[, names(coefi)] %*% coefi

mse.best.subset <- mean ((test.y - yhat)^2)
sprintf( "------------ MSE on test data for best model with 8 component: %s",
         mse.best.subset)
```

```
## [1] "------------ MSE on test data for best model with 8 component: 41.4927142070165"
```

```r
print("c: fit Ridge regression model on training set and get test.mse-------------")
```

```
## [1] "c: fit Ridge regression model on training set and get test.mse-------------"
```

```r
# use magic model.matrix to convert dataframe into a matrix for Ridge and Lasso
x.train <- model.matrix(crim~., df.train)[, -1]
y.train <- (df.train)$crim
x.test <- model.matrix(crim~., df.test)[, -1]
y.test <- df.test$crim

cv.out <- cv.glmnet(x.train, y.train, alpha=0) # Ridge
plot(cv.out)
```




```r
best.lambda <- cv.out$lambda.min

# predict the model on test
pred.ridge <- predict(cv.out, s=best.lambda, newx=x.test)

sprintf("Ridge test cv_mse for best lambda: %s", best.lambda)
```

```
## [1] "Ridge test cv_mse for best lambda: 0.591915933409596"
```

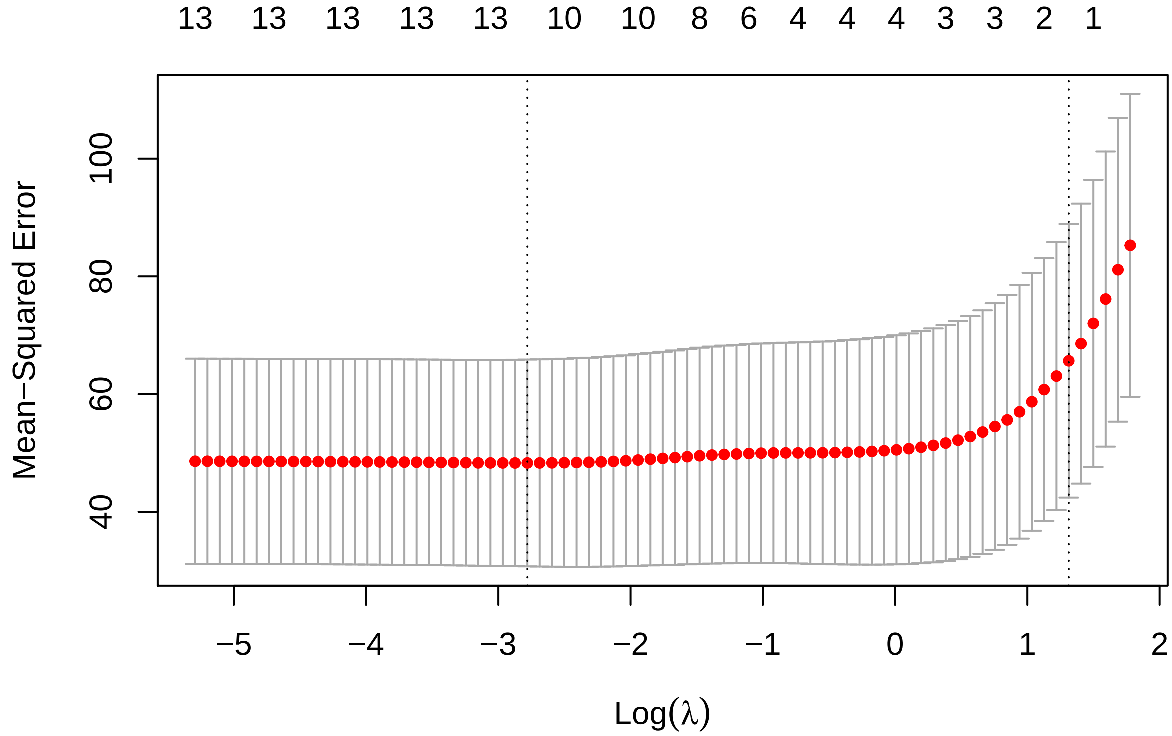
```r
ridge.mse <- mean((pred.ridge - y.test)^2)

sprintf( "-----------Ridge MSE on test data for model with best lambda %s is %s",
         best.lambda, ridge.mse)
```

```
## [1] "-----------Ridge MSE on test data for model with best lambda 0.591915933409596 is 40.92776646235
```

```r
print("fit Lasso regression model on training set and get test.mse ")
```

```
## [1] "fit Lasso regression model on training set and get test.mse "
```

```r
cv.out <- cv.glmnet(x.train, y.train, alpha=1) # Lasso
plot(cv.out)
```




```r
best.lambda <- cv.out$lambda.min

# predict the model on test
pred.lasso <- predict(cv.out, s=best.lambda, newx=x.test)

sprintf("Lasso test cv_mse for best lambda: %s", best.lambda)
```

```
## [1] "Lasso test cv_mse for best lambda: 0.0620100453480678"
```

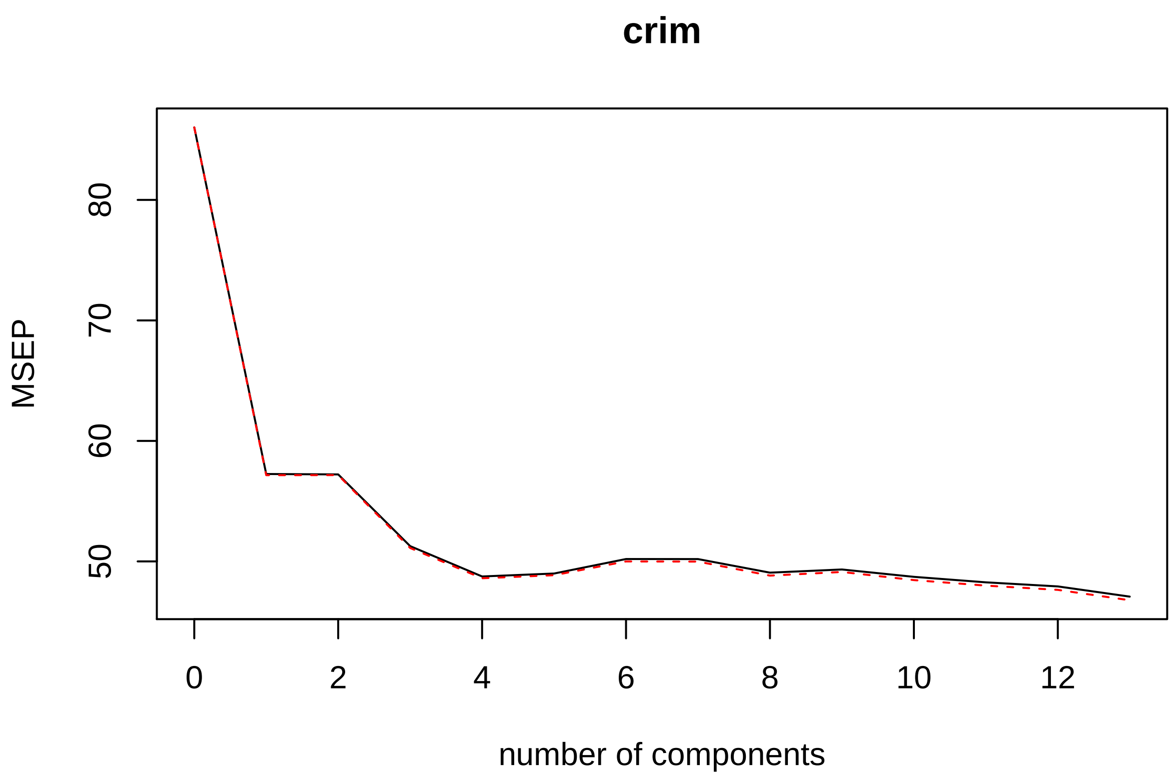
```r
lasso.mse <- mean((pred.lasso - y.test)^2)

sprintf( "-----------Lasso MSE on test data for model with best lambda %s is %s",
         best.lambda, lasso.mse)
```

```
## [1] "-----------Lasso MSE on test data for model with best lambda 0.0620100453480678 is 40.931404807
```

```r
print("Fit a PCR model on training set with M chosen by CS and get test.mse")
```

```
## [1] "Fit a PCR model on training set with M chosen by CS and get test.mse"
```

```r
pcr.fit <- pcr (crim~., data=df.train, scale=T, validation="CV")

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




```r
print("Minimum CV Root MSE is for M=4 components which is 50 so CV MSE is 2500")
```

```
## [1] "Minimum CV Root MSE is for M=4 components which is 50 so CV MSE is 2500"
```

```r
# Now apply model 1ith M=17 on test data and calculate MSE'
M = 4
pcr.pred <- predict(pcr.fit, x.test, ncomp = M)
pcr.mse <- mean((pcr.pred - y.test)^2)
sprintf("pcr test mse  for best number of component: %s is %s:", M, pcr.mse)
```

```
## [1] "pcr test mse  for best number of component: 4 is 43.9110729287546:"
```

```r
sprintf("b) best subset selection has the best test MSE: %s
        comparing with Ridge test MSE %s, Lasso test MSE %s
        and PCR test %s MSEtherefore we prefer it to other three",
        mse.best.subset, ridge.mse, lasso.mse, pcr.mse)
```

```
## [1] "b) best subset selection has the best test MSE: 41.4927142070165 \n        comparing with Ridge
```

```r
sprintf("c) Best  subset selection only includes %s number of components
        probably because other features add noise",M)
```

```
## [1] "c) Best  subset selection only includes 4 number of components\n        probably because other
```

```r
# createDataPartition(
#   y,
#   times = 1,
#   p = 0.5,
#   list = TRUE,
#   groups = min(5, length(y))
# )
# createFolds(y, k = 10, list = TRUE, returnTrain = FALSE)
```

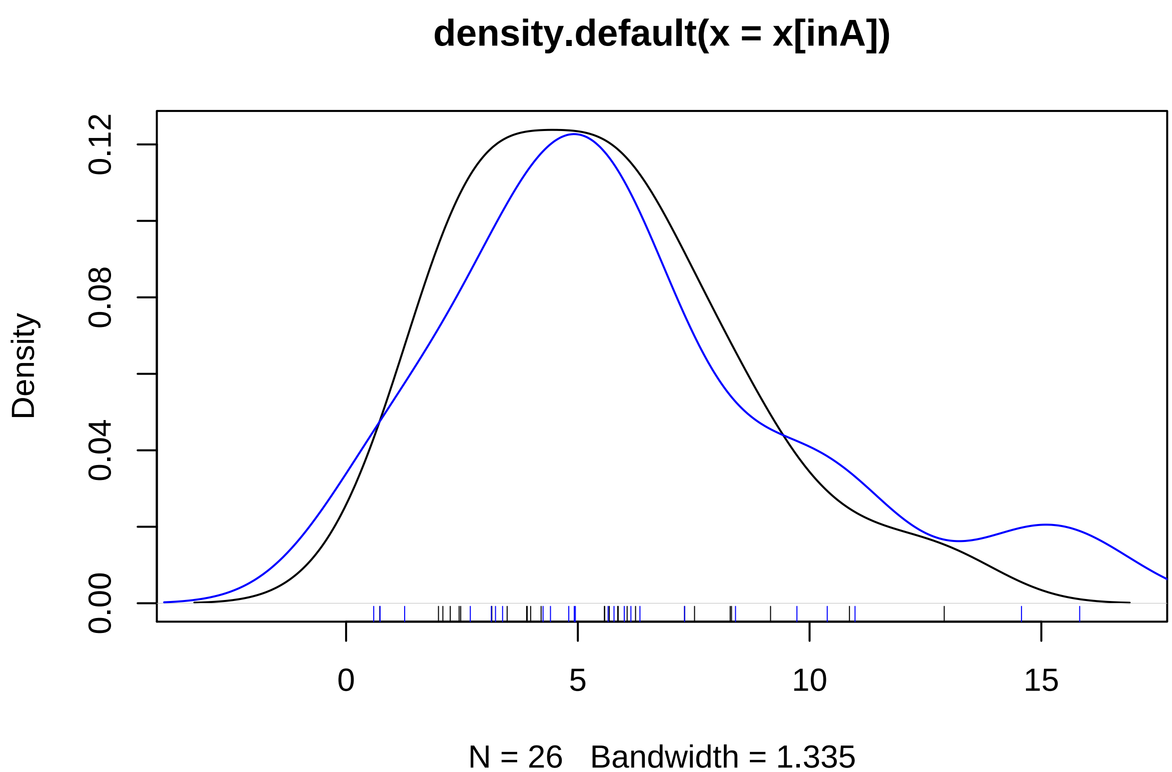
```r
#
# createMultiFolds(y, k = 10, times = 5)
#
# createTimeSlices(y, initialWindow, horizon = 1, fixedWindow = TRUE, skip = 0)
#
# groupKFold(group, k = length(unique(group)))
#
# createResample(y, times = 10, list = TRUE)
# ============================================================================
# Arguments
# y: a vector of outcomes. For createTimeSlices, these should be in
#    chronological order.
# times: the number of partitions to create
# p : the percentage of data that goes to training
# list : logical - should the results be in a list (TRUE) or a matrix with the
#       number of rows equal to floor(p * length(y)) and times columns.
# groups: for numeric y, the number of breaks in the quantiles (see below)
# k: an integer for the number of folds.
# returnTrain : a logical. When true, the values returned are the sample
#               positions corresponding to the data used during training.
#               This argument only works in conjunction with list = TRUE
# initialWindow: Initial number of consecutive values in each training set sample
# horizon: Number of consecutive values in test set sample
# fixedWindow: logical, if FALSE, all training samples start at 1
# skip: integer, how many (if any) resamples to skip to thin the total amount
# group: a vector of groups whose length matches the number of rows in the
#        overall data set.


library(tidyverse)
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'lattice'

## The following object is masked from 'package:boot':
##
##     melanoma

##
## Attaching package: 'caret'

## The following object is masked from 'package:pls':
##
##     R2

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(glmnet)


data(oil)
createDataPartition(oilType, 2)
```

```
## $Resample1
##  [1]  3  8  9 10 11 13 14 15 16 21 22 23 25 26 27 28 29 31 34 36 40 41 42 43 45
## [26] 49 51 52 54 58 59 62 63 64 67 69 71 74 75 76 77 78 81 82 83 86 88 90 95 96
##
## $Resample2
##  [1]  5  7  9 10 11 12 14 15 17 19 21 22 24 25 29 30 35 38 40 41 42 43 46 47 48
## [26] 51 53 55 57 59 60 61 67 68 69 70 73 74 75 76 78 79 82 86 87 88 92 93 95 96
```

```
x <- rgamma(50, 3, .5)
inA <- createDataPartition(x, list = FALSE)

plot(density(x[inA]))
rug(x[inA])

points(density(x[-inA]), type = "l", col = 4)
rug(x[-inA], col = 4)
```




```
createResample(oilType, 2)
```

```
## $Resample1
##  [1]  2  2  5  5  6  7  8  9  9 10 10 11 13 15 15 16 16 17 20 21 22 22 25 25 27
## [26] 30 33 34 34 36 36 37 38 39 39 40 40 41 42 43 43 43 44 45 46 46 46 47 47 47
## [51] 49 50 50 54 55 59 59 60 60 62 62 63 64 64 66 67 67 68 68 70 72 75 77 78 79
## [76] 79 79 80 80 82 85 85 86 87 87 87 88 88 89 90 90 91 92 95 95 96
##
## $Resample2
##  [1]  2  3  4  4  4  5  5  8  9 11 12 14 15 15 17 18 18 18 18 19 19 21 24 24 24
## [26] 25 26 27 27 29 29 30 34 34 35 38 38 41 41 41 42 45 46 46 47 47 47 47 50 51
## [51] 52 52 53 57 58 59 59 59 60 62 63 63 67 68 68 70 72 73 74 74 75 77 78 79 79
```

```
## [76] 80 80 80 82 83 84 85 85 85 86 87 88 89 89 90 90 92 94 94 96 96
```

```r
createFolds(oilType, 10)
```

```
## $Fold01
##  [1]  3 15 20 23 49 53 61 72 74 84 94
##
## $Fold02
##  [1]  7 10 21 27 46 54 60 69 75 78 86
##
## $Fold03
## [1]  2  5 16 17 22 30 45 57 77
##
## $Fold04
##  [1] 14 37 38 43 51 64 81 82 90 92
##
## $Fold05
##  [1]  4 29 50 56 65 66 68 76 83 89 91
##
## $Fold06
## [1]  6 12 31 33 39 40 42 47 59
##
## $Fold07
## [1]  1  8 13 28 44 52 85 87
##
## $Fold08
##  [1] 19 34 41 55 63 67 73 80 93 96
##
## $Fold09
## [1]  9 11 24 26 32 70 79 95
##
## $Fold10
## [1] 18 25 35 36 48 58 62 71 88
```

```r
createFolds(oilType, 5, FALSE)
```

```
##  [1] 3 3 2 2 1 5 4 2 3 5 5 3 4 3 4 1 4 5 5 1 2 4 3 5 2 4 3 4 3 4 3 5 1 4 5 1 5 1
## [39] 2 1 4 4 1 2 1 4 1 3 1 5 3 3 1 4 5 2 3 4 3 1 4 2 2 4 2 3 4 5 1 3 5 3 2 4 1 2
## [77] 3 3 1 3 1 2 2 4 2 3 4 5 1 5 2 5 5 1 2 5
```

```r
createFolds(rnorm(21))
```

```
## $Fold01
## [1]  2 21
##
## $Fold02
## [1]  6 11
##
## $Fold03
## [1] 1 5 8
##
## $Fold04
## [1] 17 19
##
## $Fold05
## [1]  7 10
```

```
##
## $Fold06
## [1]  3 16
##
## $Fold07
## [1] 15 20
##
## $Fold08
## [1]  4 12
##
## $Fold09
## [1] 14 18
##
## $Fold10
## [1]  9 13
```

```r
createTimeSlices(1:9, 5, 1, fixedWindow = FALSE)
```

```
## $train
## $train$Training5
## [1] 1 2 3 4 5
##
## $train$Training6
## [1] 1 2 3 4 5 6
##
## $train$Training7
## [1] 1 2 3 4 5 6 7
##
## $train$Training8
## [1] 1 2 3 4 5 6 7 8
##
##
## $test
## $test$Testing5
## [1] 6
##
## $test$Testing6
## [1] 7
##
## $test$Testing7
## [1] 8
##
## $test$Testing8
## [1] 9
```

```r
createTimeSlices(1:9, 5, 1, fixedWindow = TRUE)
```

```
## $train
## $train$Training5
## [1] 1 2 3 4 5
##
## $train$Training6
## [1] 2 3 4 5 6
##
## $train$Training7
```

```
## [1] 3 4 5 6 7
##
## $train$Training8
## [1] 4 5 6 7 8
##
##
## $test
## $test$Testing5
## [1] 6
##
## $test$Testing6
## [1] 7
##
## $test$Testing7
## [1] 8
##
## $test$Testing8
## [1] 9
```

```r
createTimeSlices(1:9, 5, 3, fixedWindow = TRUE)
```

```
## $train
## $train$Training5
## [1] 1 2 3 4 5
##
## $train$Training6
## [1] 2 3 4 5 6
##
##
## $test
## $test$Testing5
## [1] 6 7 8
##
## $test$Testing6
## [1] 7 8 9
```

```r
createTimeSlices(1:9, 5, 3, fixedWindow = FALSE)
```

```
## $train
## $train$Training5
## [1] 1 2 3 4 5
##
## $train$Training6
## [1] 1 2 3 4 5 6
##
##
## $test
## $test$Testing5
## [1] 6 7 8
##
## $test$Testing6
## [1] 7 8 9
```

```r
createTimeSlices(1:15, 5, 3)
```

```
## $train
```

```
## $train$Training05
## [1] 1 2 3 4 5
##
## $train$Training06
## [1] 2 3 4 5 6
##
## $train$Training07
## [1] 3 4 5 6 7
##
## $train$Training08
## [1] 4 5 6 7 8
##
## $train$Training09
## [1] 5 6 7 8 9
##
## $train$Training10
## [1]  6  7  8  9 10
##
## $train$Training11
## [1]  7  8  9 10 11
##
## $train$Training12
## [1]  8  9 10 11 12
##
##
## $test
## $test$Testing05
## [1] 6 7 8
##
## $test$Testing06
## [1] 7 8 9
##
## $test$Testing07
## [1]  8  9 10
##
## $test$Testing08
## [1]  9 10 11
##
## $test$Testing09
## [1] 10 11 12
##
## $test$Testing10
## [1] 11 12 13
##
## $test$Testing11
## [1] 12 13 14
##
## $test$Testing12
## [1] 13 14 15
```

```r
createTimeSlices(1:15, 5, 3, skip = 2)
```

```
## $train
## $train$Training05
## [1] 1 2 3 4 5
```

```
##
## $train$Training08
## [1] 4 5 6 7 8
##
## $train$Training11
## [1]  7  8  9 10 11
##
##
## $test
## $test$Testing05
## [1] 6 7 8
##
## $test$Testing08
## [1]  9 10 11
##
## $test$Testing11
## [1] 12 13 14
```

```r
createTimeSlices(1:15, 5, 3, skip = 3)
```

```
## $train
## $train$Training5
## [1] 1 2 3 4 5
##
## $train$Training9
## [1] 5 6 7 8 9
##
##
## $test
## $test$Testing5
## [1] 6 7 8
##
## $test$Testing9
## [1] 10 11 12
```

```r
set.seed(131)
groups <- sort(sample(letters[1:4], size = 20, replace = TRUE))
table(groups)
```

```
## groups
## a b c d
## 6 5 4 5
```

```r
folds <- groupKFold(groups)
lapply(folds, function(x, y) table(y[x]), y = groups)
```

```
## $Fold1
##
## b c d
## 5 4 5
##
## $Fold2
##
## a c
## 6 4
##
```

```
## $Fold3
##
## a b d
## 6 5 5
```