HW4

Kihyun Han

11/17/2021

Q1

##(a) By taking `floor(0.8*n)` observations, we obtain the training set, and the rest is assigned as the test set.

```
library(ISLR2)
College = ISLR2::College
x = model.matrix(Apps~.,College)[,-1]
y = College$Apps
n = length(College[[1]])
set.seed(12)
training_index = sample(n,floor(0.8*n))
training_set = College[training_index,]
test_set = College[-training_index,]
```

(b)

The test error for the linear model is 1970987.

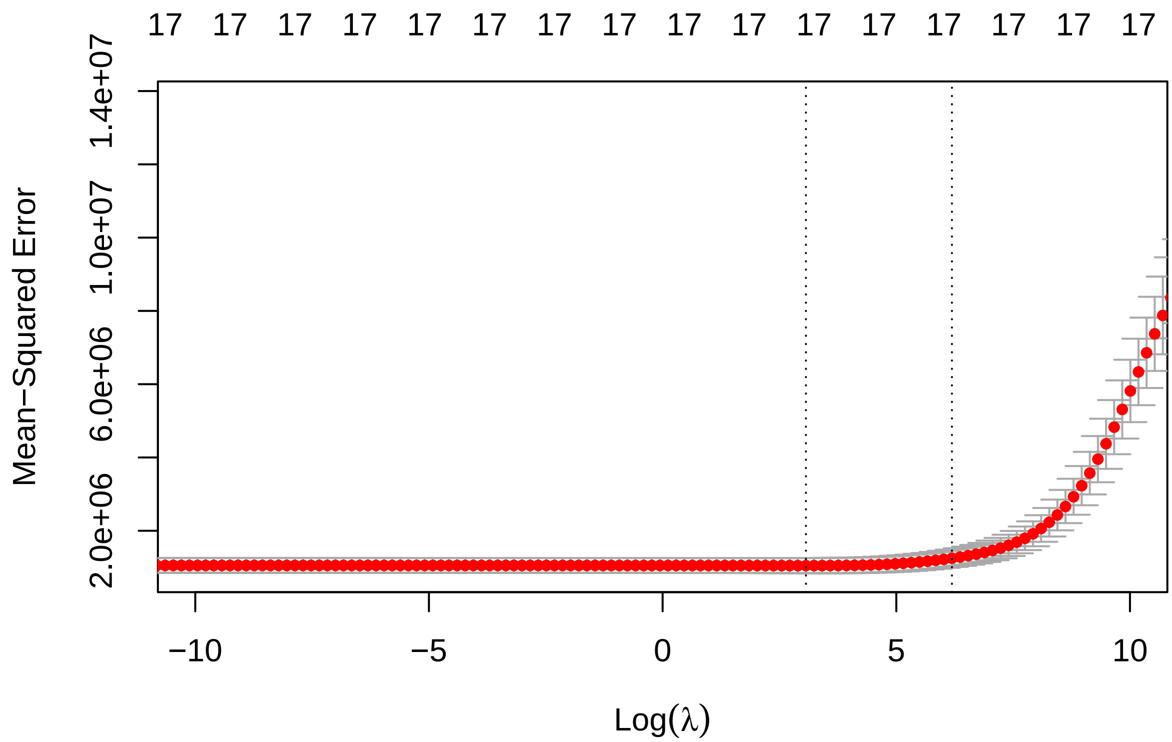
```
linear_fit = lm(Apps ~ ., data = training_set)
test_prediction = predict(linear_fit, test_set)
mse_lin = mean((test_set$Apps - test_prediction)^2)
mse_lin
```

```
## [1] 1970987
```

(c)

The best λ with 10-fold (default) cross-validation error is 21.461.

```
library(glmnet)
set.seed(12)
grid = 10^seq(10,-5,length = 200)
cv_ridge = cv.glmnet(x[training_index,], y[training_index], alpha = 0, lambda = grid)
ridge_mod = glmnet(x[training_index,], y[training_index], alpha = 0, lambda = grid)
plot(cv_ridge, xlim = c(-10,10))
```




```r
best_lam = cv_ridge$lambda.min
best_lam
```

```
## [1] 21.46141
```

Then the test error is 2107210.

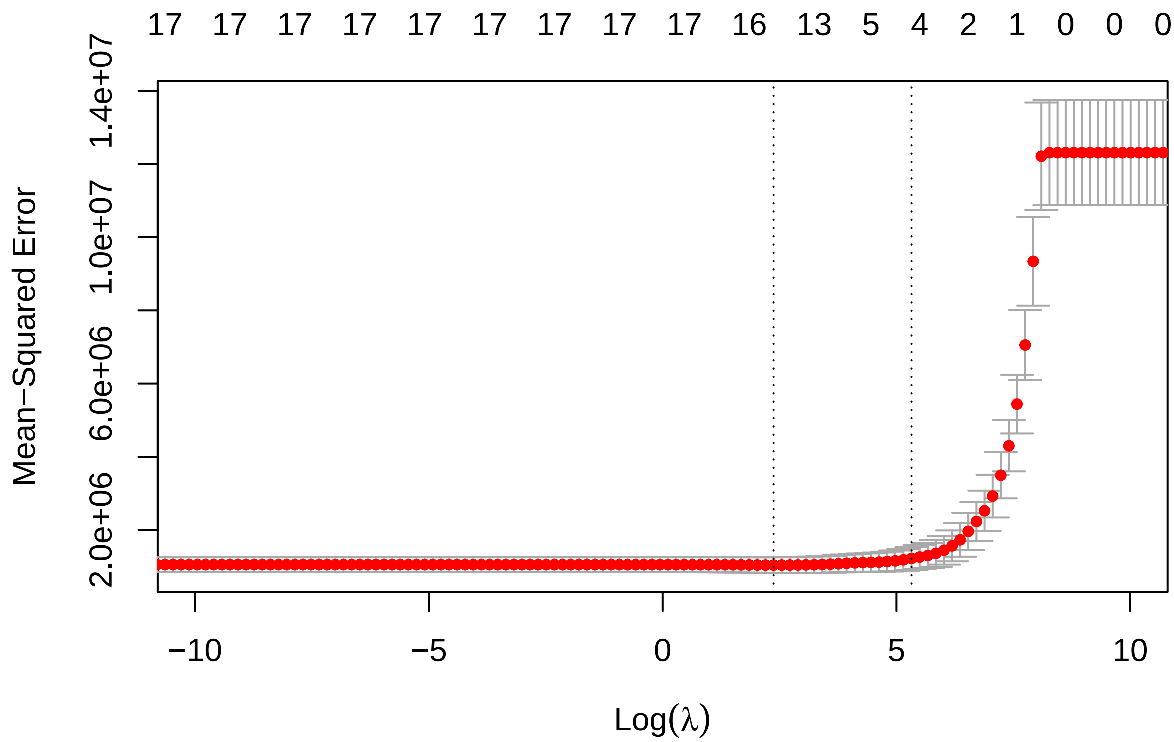
```r
predict_ridge = predict(ridge_mod, x[-training_index,], s=best_lam)
mse_ridge = mean((predict_ridge - y[-training_index])^2)
mse_ridge
```

```
## [1] 2107210
```

(d)

The best lambda for lasso is 10.718.

```r
set.seed(12)
grid = 10^seq(10,-5,length = 200)
cv_lasso = cv.glmnet(x[training_index,], y[training_index], alpha = 1, lambda = grid)
lasso_mod = glmnet(x[training_index,], y[training_index], alpha = 1, lambda = grid)
plot(cv_lasso, xlim = c(-10,10))
```




```r
best_lam_lasso = cv_lasso$lambda.min
best_lam_lasso
```

```
## [1] 10.71891
```

The cross-validation error is 2053068 for lasso, and there are 16 nonzero predictors out of 18 including intercept. If we exclude the intercept, then there are 15 nonzero predictors out of 17.

```r
predict_lasso = predict(lasso_mod, x[-training_index,], s=best_lam_lasso)
mse_lasso = mean((predict_lasso - y[-training_index])^2)
mse_lasso
```

```
## [1] 2053068
```

```r
predict(lasso_mod, x[-training_index,], s=best_lam_lasso, type = 'coefficients')
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                        s1
## (Intercept) -576.37982979
## PrivateYes  -569.37623793
## Accept         1.34449637
## Enroll         .
## Top10perc     35.68110102
## Top25perc     -5.40173667
```

```
## F.Undergrad    .
## P.Undergrad    0.03648229
## Outstate      -0.05927299
## Room.Board     0.09796867
## Books         -0.01249590
## Personal      -0.02437991
## PhD           -6.02279932
## Terminal      -6.20211041
## S.F.Ratio     15.62135812
## perc.alumni   -5.78615965
## Expend         0.11351607
## Grad.Rate      8.68358039
```

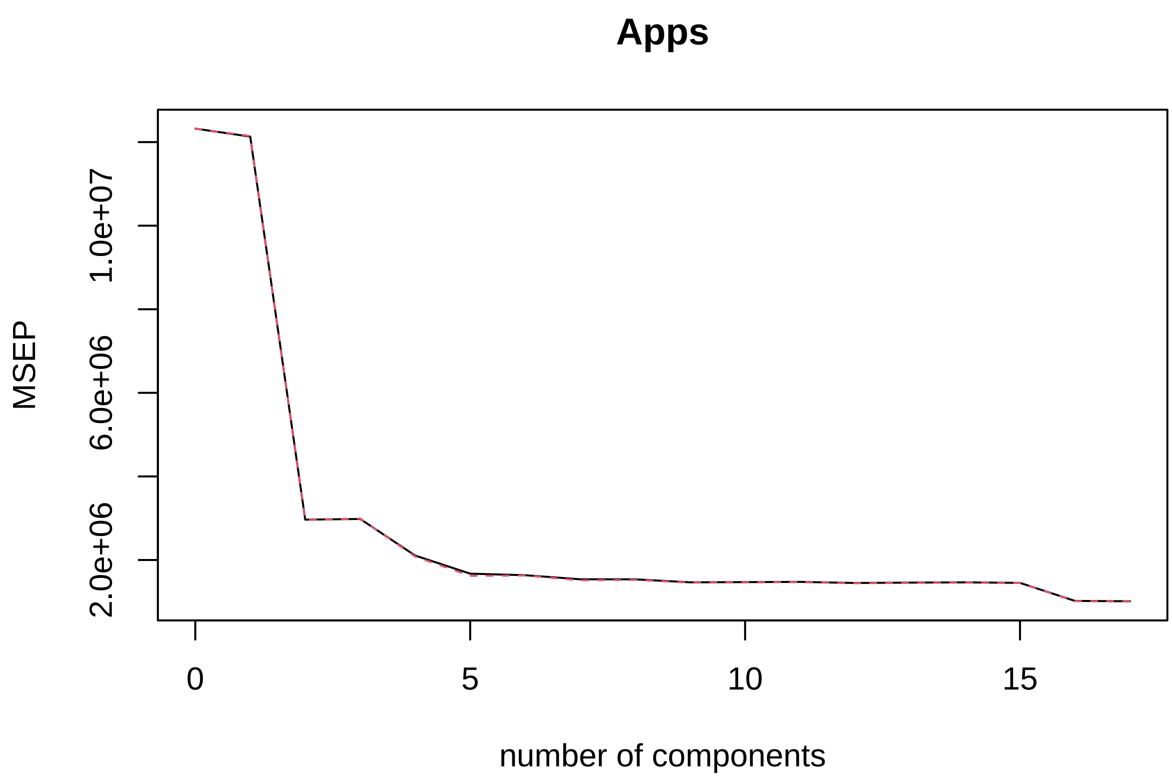
(e)

Note that the smallest value of MSE is attained when M is 17, the full model. Since the MSE value seems unchanged when M is larger than 10, we may choose the optimal value of M as 10.

```
library(pls)
set.seed(12)
pcr.fit = pcr(Apps ~ ., data = College, subset= training_index, scale = TRUE, validation = 'CV')
summary(pcr.fit)
```

```
## Data:    X dimension: 621 17
##  Y dimension: 621 1
## Fit method: svdpc
## Number of components considered: 17
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            3510     3483     1722     1727     1451     1294     1279
## adjCV         3510     3485     1720     1730     1443     1273     1276
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1240     1240     1211      1213      1215      1204      1208
## adjCV     1230     1235     1208      1210      1212      1201      1205
##        14 comps  15 comps  16 comps  17 comps
## CV         1210      1205      1010      1007
## adjCV      1207      1204      1007      1003
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X      31.805    57.96    64.74    70.48    75.97    81.08    84.60    87.97
## Apps    3.044    76.48    76.58    84.31    87.46    87.54    88.44    88.44
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       90.96     93.40     95.44     97.07     98.13     98.97     99.46
## Apps    88.91     88.96     88.99     89.20     89.21     89.22     89.55
##       16 comps  17 comps
## X        99.84    100.00
## Apps     92.51     92.71
```

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



Then the test error is 4988288.

```
pcr.pred = predict(pcr.fit, x[-training_index,], ncomp = 10)
mse_pcr = mean((pcr.pred - y[-training_index])^2)
mse_pcr
```

```
## [1] 4988288
```

(f)

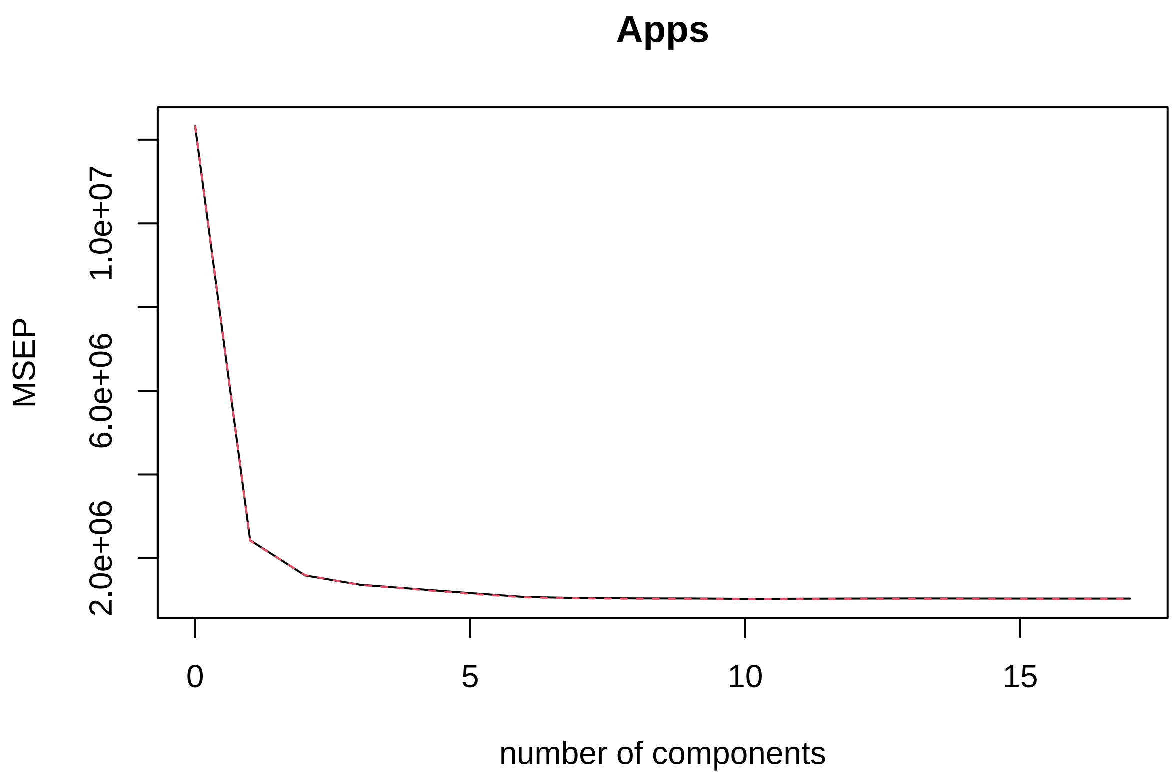
Here, the CV error is minimized at $M = 10$ components. However, we will take $M = 7$ since the value seems unchanged for large M.

```
library(pls)
pls.fit = plsr(Apps ~ ., data = College, subset = training_index, scale = T, validation = 'CV')
summary(pls.fit)
```

```
## Data:    X dimension: 621 17
##  Y dimension: 621 1
## Fit method: kernelpls
## Number of components considered: 17
```

```
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV            3510     1559     1260     1168     1125     1080     1036
## adjCV         3510     1555     1263     1166     1121     1071     1032
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV        1024     1020     1019      1015      1016      1018      1019
## adjCV     1020     1016     1015      1011      1012      1014      1015
##        14 comps  15 comps  16 comps  17 comps
## CV         1018      1018      1018      1018
## adjCV      1015      1014      1014      1014
##
## TRAINING: % variance explained
##       1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps  8 comps
## X       26.34    41.50    63.44    66.37    69.04    73.69    76.87    81.14
## Apps    81.10    87.41    89.61    90.87    92.12    92.51    92.63    92.66
##       9 comps  10 comps  11 comps  12 comps  13 comps  14 comps  15 comps
## X       84.01      85.6     89.36     91.71     92.97     94.31     95.77
## Apps    92.67      92.7     92.70     92.71     92.71     92.71     92.71
##       16 comps  17 comps
## X        97.74    100.00
## Apps     92.71     92.71
```

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



Then the test MSE value is 2113527.

```
pls.pred <- predict(pls.fit, x[-training_index, ], ncomp = 7)
mse_pls = mean((pls.pred - y[-training_index])^2)
mse_pls
```

```
## [1] 2113527
```

(g)

We compare all the results above. We may deduce that the linear model has the smallest test MSE, whereas PCR model has a considerably higher MSE compared to others. Since PCR model for all components is identical to linear model, we should choose higher M value in order to get smaller test error for PCR.

```
c(mse_lin, mse_ridge, mse_lasso, mse_pcr, mse_pls)
```

```
## [1] 1970987 2107210 2053068 4988288 2113527
```

HW3

Kihyun Han

11/17/2021

Q1

(a)

The following is the fitted logistic regression model.

```
library(ISLR)
glm.fits = glm(default ~ income + balance, data= Default, family = binomial)
glm.fits
```

```
##
## Call:  glm(formula = default ~ income + balance, family = binomial,
##     data = Default)
##
## Coefficients:
## (Intercept)       income      balance
##  -1.154e+01    2.081e-05    5.647e-03
##
## Degrees of Freedom: 9999 Total (i.e. Null);  9997 Residual
## Null Deviance:       2921
## Residual Deviance: 1579  AIC: 1585
```

(b)

Step 1: Split the data into two halves.

```
set.seed(2)
n = length(Default[[1]])#10000
training_index = sample(n,0.5 * n)
test_index = (1:n) [-training_index]
Default_training = Default[training_index,]
Default_test = Default[test_index,]
```

Step 2:

```
glm.training = glm(default ~ income + balance, data = Default_training, family= binomial)
glm.training
```

```
##
## Call:  glm(formula = default ~ income + balance, family = binomial,
##     data = Default_training)
##
## Coefficients:
## (Intercept)       income      balance
##  -1.090e+01    1.622e-05    5.365e-03
##
## Degrees of Freedom: 4999 Total (i.e. Null);  4997 Residual
## Null Deviance:       1484
## Residual Deviance: 854.5     AIC: 860.5
```

Step 3:

```
posterior = predict(glm.training, Default_test, type = 'response')
predicted = posterior>0.5
observed = Default_test$default=='Yes'
```

Step 4: the validatin set error is 0.119.

```
sum(observed!=predicted) / 1000
```

```
## [1] 0.119
```

(c)

We develop a function `cv_process`, which we made in (b), with an input seed number `k`.

```
cv_process = function(k){
  set.seed(k)
  n = length(Default[[1]])#10000
  training_index = sample(n,0.5 * n)
  test_index = (1:n) [-training_index]
  Default_training = Default[training_index,]
  Default_test = Default[test_index,]
  glm.training = glm(default ~ income + balance, data = Default_training, family= binomial)
  posterior = predict(glm.training, Default_test, type = 'response')
  predicted = posterior>0.5
  observed = Default_test$default=='Yes'
  return(sum(observed!=predicted) / 1000)
}
```

By setting different validation sets, the following shows that the validation error is 0.127, 0.135, and 0.138. This implies that the validation error is around 0.13.

```
result = rep(0,3)
result[1] = cv_process(1)
result[2] = cv_process(12)
result[3] = cv_process(123)
result
```

```
## [1] 0.127 0.135 0.138
```

(d)

The logistic regression The result for the full model with the same data split is 0.130, 0.134, and 0.136. Thus, we may conclude that the dummy variable did not reduce the test error rate.

```
cv_process_full = function(k){
  set.seed(k)
  n = length(Default[[1]])#10000
  training_index = sample(n,0.5 * n)
  test_index = (1:n) [-training_index]
  Default_training = Default[training_index,]
  Default_test = Default[test_index,]
  glm.training = glm(default ~ income + balance + student, data = Default_training, family= binomial)
  posterior = predict(glm.training, Default_test, type = 'response')
  predicted = posterior>0.5
  observed = Default_test$default=='Yes'
  return(sum(observed!=predicted) / 1000)
}
result_full = rep(0,3)
result_full[1] = cv_process_full(1)
result_full[2] = cv_process_full(12)
result_full[3] = cv_process_full(123)
result_full
```

```
## [1] 0.130 0.134 0.136
```

Q2

(a)

The estimated standard errors of (intercept, income, balance) is (4.348e-01, 4.985e-06, 2.273e-04).

```
fit_all = summary(glm(default ~ income + balance, data = Default, family = binomial))
fit_all$coefficients[,2]
```

```
##  (Intercept)       income      balance
## 4.347564e-01 4.985167e-06 2.273731e-04
```

(b)

```
boot.fn = function(Default, index){
  data_boot = Default[index,]
  fit = glm(default ~ income + balance, data = data_boot, family = binomial)
  return(summary(fit)$coefficients[,1])
}
```

(c)

The standard error for the parameters (intercept, income, balance) is (4.348e-01, 4.874e-06, 2.315e-04).

```
library(boot)
set.seed(2)
boot_result = boot(data = Default, boot.fn, R = 100)
boot_result
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Default, statistic = boot.fn, R = 100)
##
##
## Bootstrap Statistics :
##          original        bias     std. error
## t1* -1.154047e+01  1.803508e-02 4.348178e-01
## t2*  2.080898e-05 -1.843687e-07 4.874022e-06
## t3*  5.647103e-03 -9.175093e-06 2.315819e-04
```

(d)

The estimated standard errors from two different methods are similar to each other. It is because the sample variance of the bootstrap distribution can be approximated as the variance of the bootstrap distribution, and the bootstrap distribution of the estimate can be approximated again as the original sample distribution of the estimate, which implies that the sample standard errors are approximately the same.

HW2

Kihyun Han

10/16/2021

(a)

```
set.seed(1)
x = rnorm(100)
head(x)
```

```
## [1] -0.6264538  0.1836433 -0.8356286  1.5952808  0.3295078 -0.8204684
```

(b)

Since the variance is 0.25, the standard deviation of each value of eps is 0.5.

```
eps = rnorm(100, sd = 0.5)
head(eps)
```

```
## [1] -0.31018334  0.02105794 -0.45546082  0.07901439 -0.32729232  0.88364363
```

(c)

The length of y is 100. In this model, $\beta_0 = -1$ and $\beta_1 = 0.5$

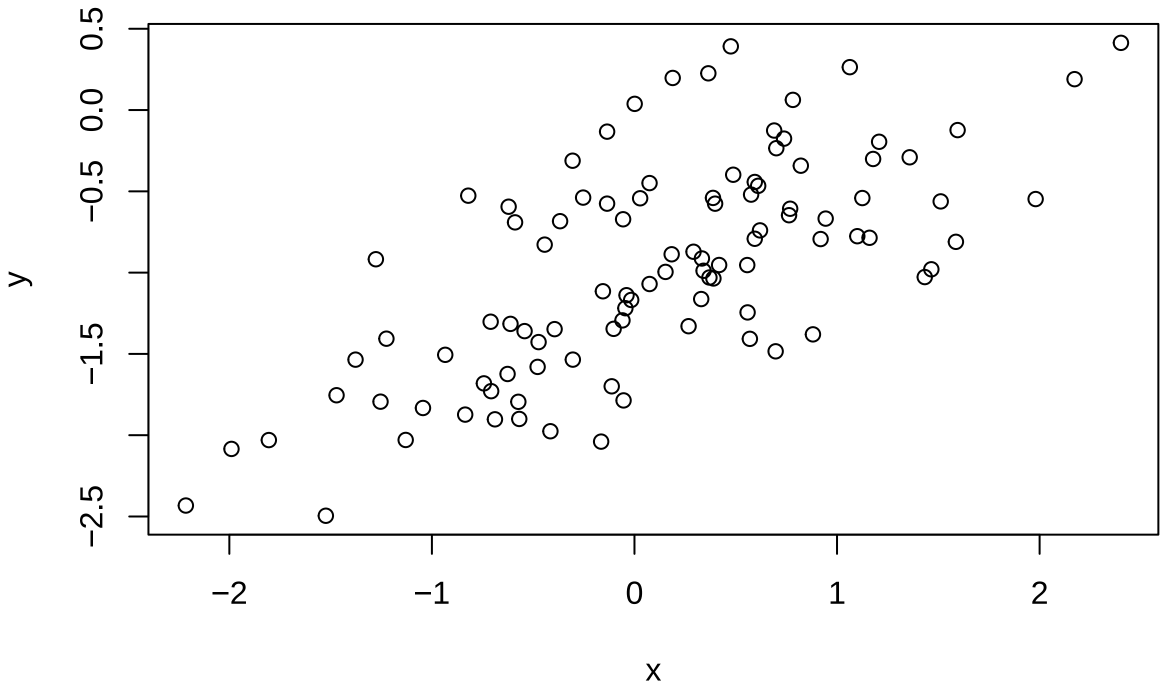
```
y = -1 + 0.5 * x + eps
length(y)
```

```
## [1] 100
```

(d)

In the following plot, x and y follow the trend $y = -1 + 0.5x$ with a little error.

```
plot(x,y)
```



(e)

The least square linear model for y and x is $y = -1.0188 + 0.4995x$. Thus, $\hat{\beta}_0 = -1.0188$ and $\hat{\beta}_1 = 0.4995$ is similar to $\beta_0 = -1$ and $\beta_1 = 0.5$.

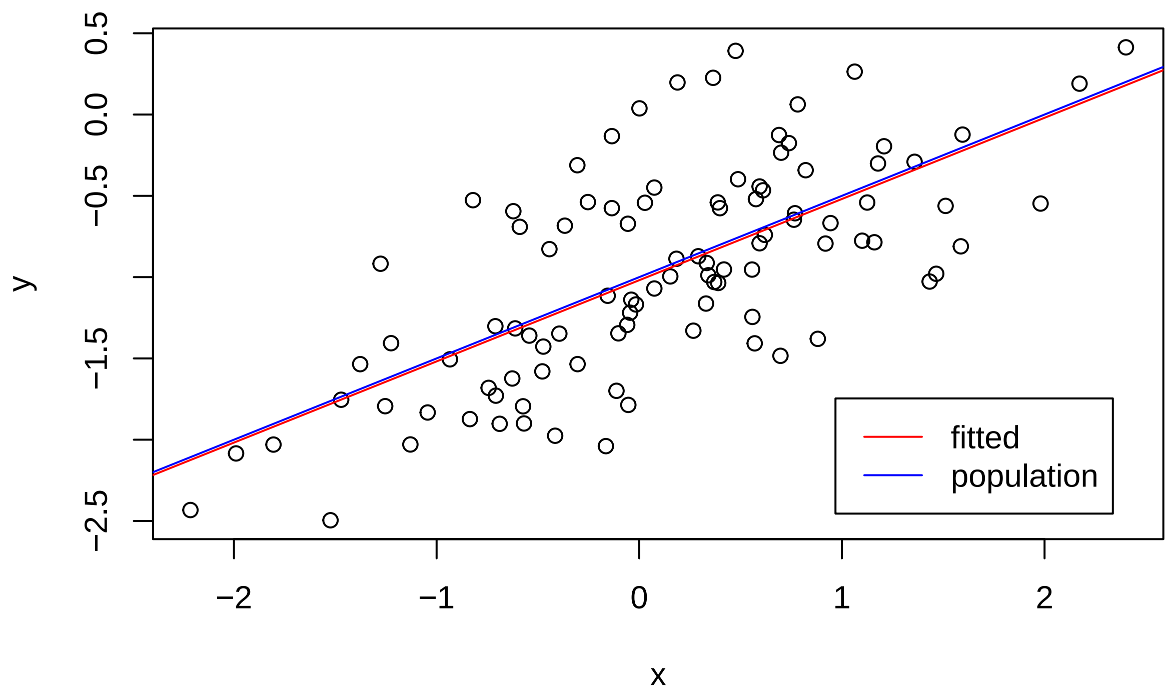
```
fit = lm(y ~ x)
fit$coefficients
```

```
## (Intercept)           x
##  -1.0188463   0.4994698
```

(f)

In the following plot, the color of the least square fitted line and the population regression line is red and blue, respectively.

```
plot(x,y)
abline(fit$coefficients[1], fit$coefficients[2], col = 'red')
abline(-1, 0.5, col = 'blue')
legend(x = 'bottomright', legend = c("fitted", "population"), col = c('red', 'blue'), lwd = 1, inset = 
```



(g)

By applying anova function to the linear fit and the quadratic fit, we get that the quadratic model does not improve compared to the linear fit for the significance level $\alpha = 0.05$ since the p-value of the F test is 0.1638, which is larger than 0.05.

```
fit2 = lm(y ~ x + I(x^2))
anova(fit,fit2)
```

```
## Analysis of Variance Table
##
## Model 1: y ~ x
## Model 2: y ~ x + I(x^2)
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     98 22.709
## 2     97 22.257  1   0.45163 1.9682 0.1638
```

(h)

For this problem, we generate eps_small by reducing its standard deviation to 0.1. The length of y and the values of β_0 and β_1 remains the same. We have that $\hat{\beta}_0 = -1.0037$ and $\hat{\beta}_1 = 0.499894$. Since the values are more similar to $\beta_0 = -1$ and $\beta_1 = 0.5$, we observe that the data lies closer to the population regression line and that the less noisier fitted line is almost the same as the population regression line.

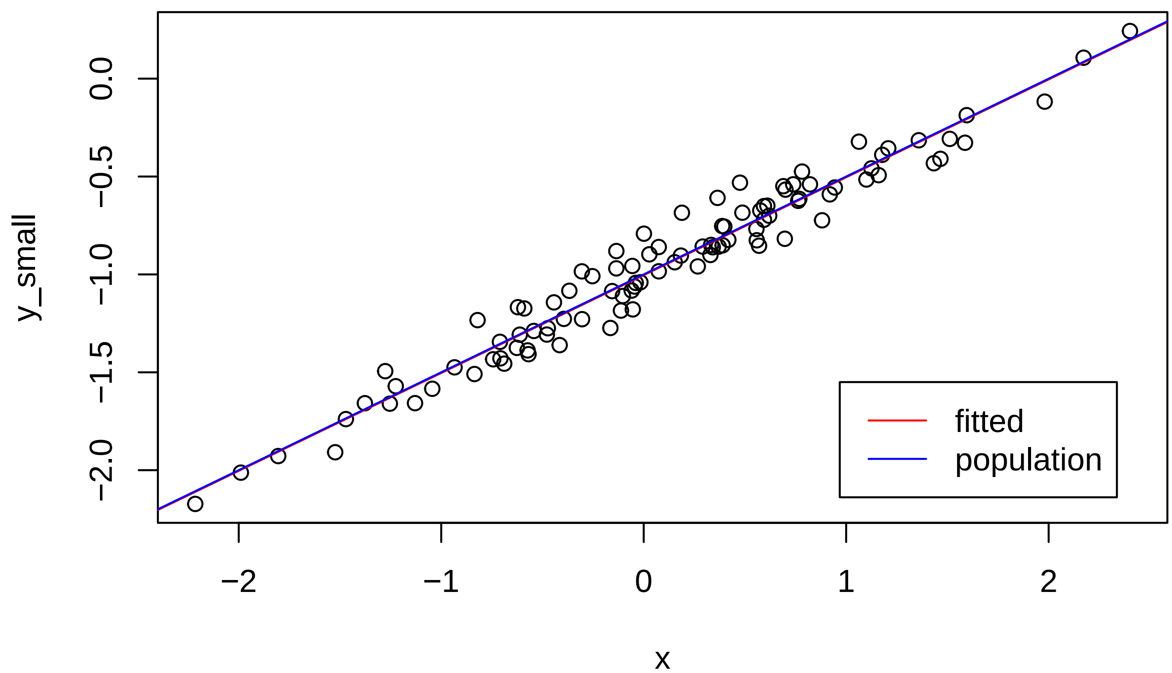
```r
set.seed(1)
x = rnorm(100)
eps_small = rnorm(100, sd = 0.1)

y_small = -1 + 0.5 * x + eps_small

fit_small = lm(y_small ~ x)
fit_small$coefficients
```

```
## (Intercept)           x
##   -1.003769    0.499894
```

```r
plot(x,y_small)
abline(fit_small$coefficients[1], fit_small$coefficients[2], col = 'red')
abline(-1, 0.5, col = 'blue')
legend(x = 'bottomright', legend = c("fitted", "population"), col = c('red', 'blue'), lwd = 1, inset =
```



(i)

In this case, the standard deviation of eps_large is 2. The length of y and the values of β_0 and β_1 remains the same. The coefficients are $\hat{\beta}_0 = -1.07538$ and $\hat{\beta}_1 = -0.49787$. The difference from β_0 and β_1 increased compared to those of the original data. Also, the data lies far from both the population regression line and the fitted line.

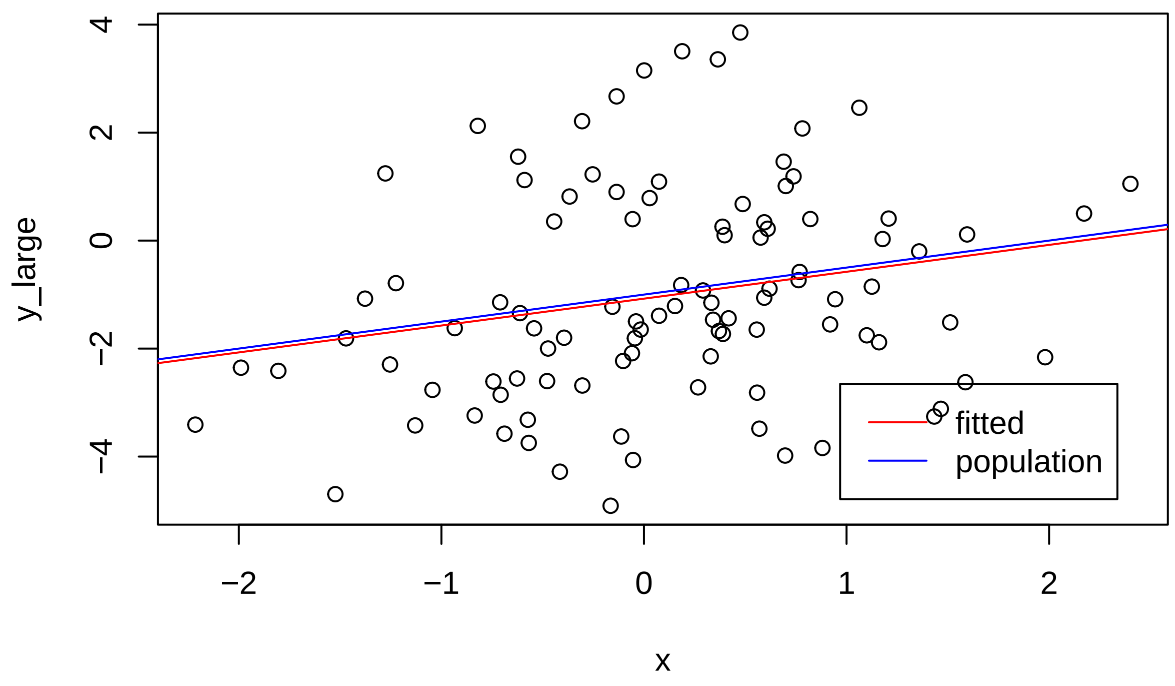
```
set.seed(1)
x = rnorm(100)
eps_large = rnorm(100, sd = 2)

y_large = -1 + 0.5 * x + eps_large

fit_large = lm(y_large ~ x)
fit_large$coefficients
```

```
## (Intercept)           x
##  -1.0753852   0.4978792
```

```
plot(x,y_large)
abline(fit_large$coefficients[1], fit_large$coefficients[2], col = 'red')
abline(-1, 0.5, col = 'blue')
legend(x = 'bottomright', legend = c("fitted", "population"), col = c('red', 'blue'), lwd = 1, inset = (
```



(j)

The 95% confidence intervals for β_0 and β_1 based on the original data are (-1.115, 0.922) and (0.392, 0.606)

```
confint(fit)
```

```
##                   2.5 %     97.5 %
## (Intercept) -1.1150804 -0.9226122
## x            0.3925794  0.6063602
```

The 95% confidence intervals for β_0 and β_1 based on the noisier data are (-1.460, 0.690) and (0.070, 0.925)

```
confint(fit_large)
```

```
##                    2.5 %     97.5 %
## (Intercept) -1.46032149 -0.6904490
## x            0.07031765  0.9254408
```

The 95% confidence intervals for β_0 and β_1 based on the less noisier data are (-1.023, 0.984) and (0.478, 0.521)

```
confint(fit_small)
```

```
##                   2.5 %     97.5 %
## (Intercept) -1.0230161 -0.9845224
## x            0.4785159  0.5212720
```

HW1

Kihyun Han

9/18/2021

#1.

The following code shows the 9×9 centering matrix. This is computed as $I - \frac{1}{n}J$, where I is the identity matrix and $J = 1\ 1^T$ is the square matrix with all entry 1.

```
n = 9
ones = rep(1, n)
I = diag(ones)
J = ones %*% t(ones)
I - J / n
```

```
##             [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
##  [1,]  0.8888889 -0.1111111 -0.1111111 -0.1111111 -0.1111111 -0.1111111
##  [2,] -0.1111111  0.8888889 -0.1111111 -0.1111111 -0.1111111 -0.1111111
##  [3,] -0.1111111 -0.1111111  0.8888889 -0.1111111 -0.1111111 -0.1111111
##  [4,] -0.1111111 -0.1111111 -0.1111111  0.8888889 -0.1111111 -0.1111111
##  [5,] -0.1111111 -0.1111111 -0.1111111 -0.1111111  0.8888889 -0.1111111
##  [6,] -0.1111111 -0.1111111 -0.1111111 -0.1111111 -0.1111111  0.8888889
##  [7,] -0.1111111 -0.1111111 -0.1111111 -0.1111111 -0.1111111 -0.1111111
##  [8,] -0.1111111 -0.1111111 -0.1111111 -0.1111111 -0.1111111 -0.1111111
##  [9,] -0.1111111 -0.1111111 -0.1111111 -0.1111111 -0.1111111 -0.1111111
##             [,7]       [,8]       [,9]
##  [1,] -0.1111111 -0.1111111 -0.1111111
##  [2,] -0.1111111 -0.1111111 -0.1111111
##  [3,] -0.1111111 -0.1111111 -0.1111111
##  [4,] -0.1111111 -0.1111111 -0.1111111
##  [5,] -0.1111111 -0.1111111 -0.1111111
##  [6,] -0.1111111 -0.1111111 -0.1111111
##  [7,]  0.8888889 -0.1111111 -0.1111111
##  [8,] -0.1111111  0.8888889 -0.1111111
##  [9,] -0.1111111 -0.1111111  0.8888889
```

#2.

These are the initial values of the multiple linear regression $Y = X\beta + \epsilon$, given in the problem.

```
Y <- c(8, 12, 16, 18, 28)
one <- rep(1, 5)
x1 <- c(12, 10, 4, 5, 3)
x2 <- c(1, 2, 1, 4, 2)
X <- cbind(one, x1, x2)
```

Since it is well known that the least-squares estimator of β is $\hat{\beta}^{OLS} = (X^TX)^{-1}X^TY$, the value is (26.59,

-1.61, 0.39).

```
beta_hat = solve(t(X) %*% X) %*% t(X) %*% Y
beta_hat
```

```
##           [,1]
## one 26.5915493
## x1  -1.6126761
## x2   0.3873239
```

#3.

First, we import the flight data.

```
# install.packages("nycflights13")
# install.packages("dplyr")
library(nycflights13)
library(dplyr)
flight_data <- flights
```

(1)

The following two methods both show that the unique elements at origin variable are "EWR", "LGA", and "JFK".

```
unique(flight_data$origin)
```

```
## [1] "EWR" "LGA" "JFK"
```

```
distinct(flight_data, origin)
```

```
## # A tibble: 3 x 1
##   origin
##   <chr>
## 1 EWR
## 2 LGA
## 3 JFK
```

(2)

The mean distance of the flights with the origin "EWR", "JFK", and "LGA" is 1057, 1266, and 780, respectively.

```
flight_data %>%
  group_by(origin) %>%
  summarise(mean_distance = mean(distance, na.rm = TRUE))
```

```
## # A tibble: 3 x 2
##   origin mean_distance
## * <chr>          <dbl>
## 1 EWR            1057.
## 2 JFK            1266.
## 3 LGA             780.
```