HW3

Peiran Chen

4/30/2022

1.

a.

```r
set.seed(123)
y <- rnorm(100)
x <- matrix(rnorm(10000*100), ncol=10000)

lm_1a <- lm(y ~ x)
```

We can write our expression as:

$$Y_i = \beta_{0,i} + \beta_{1,i}X_{1,i} + \beta_{2,i}X_{2,i} + ... + \beta_{10000,i}X_{10000,i} + \varepsilon_i, \text{ where } \varepsilon \sim N(0,1)$$

In matrix algebra form, we have:

$$Y = \beta X + \varepsilon$$

And Least Squares will give us

$$\hat{Y} = \hat{\beta} X$$

Hence, it's MSE is

$$\begin{aligned} MSE &= E[Y - \hat{Y}]^2 \\ &= Var[\varepsilon] + Bias^2[\hat{f}(x_0)] + Var[\hat{f}(x_0)] \end{aligned}$$

b.

As we can see, that $Var[\varepsilon]$ is irreducible, because it exists in nature, and we know that it is caused by the randomness in our y, which is just 1.

c.

i.

$$Bias = \sqrt{\left(E[\hat{f}(x_0)] - f(x_0)\right)^2} = f(x_0)$$

Because we have $f(x) = y \sim N(0,1)$. Thus, we have the Bias is equal to 0.

```r
fc <- rep(0,100)
variance <- (sd(fc))^2
```

ii.

$$Var[\hat{f}(x)] = Var[0] = 0$$

Thus, the variance for this procedure is 0.

iii. In this case,

$$EPE = Var[\varepsilon] + Bias^2[\hat{f}(x_0)] + Var[\hat{f}(x_0)] = Var[\varepsilon] = 1$$

And our $Bias^2[\hat{f}(x)] = 0$, and $Var[\hat{f}(x)] = 0$. And $Var[\varepsilon] = 1$, we have **EPE** $= 1$.

iv. With Validation Set Approach, we first scramble the data set into train and validation, and then, then, with $\hat{f}_{\text{train}}(\text{validation } x) = 0$. It's

$$CV_{VSA} = \frac{1}{n} \sum_{i \in \text{validation set}} (\hat{f}_{\text{train}}(x_i) - y_i)^2 = E[y_i^2] = Var[y_i] + E[y_i]^2 = 1$$

.

```r
set.seed(123)
train <- sample(100, 50)
train.x <- x[train, ]
val.x <- x[-train, ]

train.y <- y[train]
val.y <- y[-train]
lm_2 <- lm(train.y ~ train.x)
mean((0 - predict.lm(lm_2, data.frame(val.x)))^2)
```

```
## [1] 0.8815132
```

And it is close to our derivation for CV_{VSA}.

v. I found out that they do match with each other. And the reason for that is because we have set that

$$\hat{f}(x) = 0 \ \forall x \in \mathbb{R}$$

. Hence, it will has Variance of 0 and Bias of 0.

d.

```r
set.seed(123)
train <- sample(100, 50)
train.x <- x[train, ]
val.x <- x[-train, ]

train.y <- y[train]
val.y <- y[-train]
lm_2 <- lm(train.y ~ train.x)
mean((val.y - predict.lm(lm_2, data.frame(val.x)))^2)
```

```
## [1] 1.906787
```

e.

(c) has a smaller Estimated Test Error, along with 0 Bias and 0 Variance, which are definitely lower than part (d)'s Bias and Variance. It might seems like that (c) is a better choice. And it all comes down to how we generated our data in the first place. We generated y and x independently with the same $Normal(0, 1)$ distribution, so in a perfect setting, their covariance would be

$$\begin{aligned} Cov[X, Y] &= E[XY] - E[X]E[Y] \\ &= E[X]E[Y] - E[X]E[Y] \\ &= 0 \end{aligned}$$

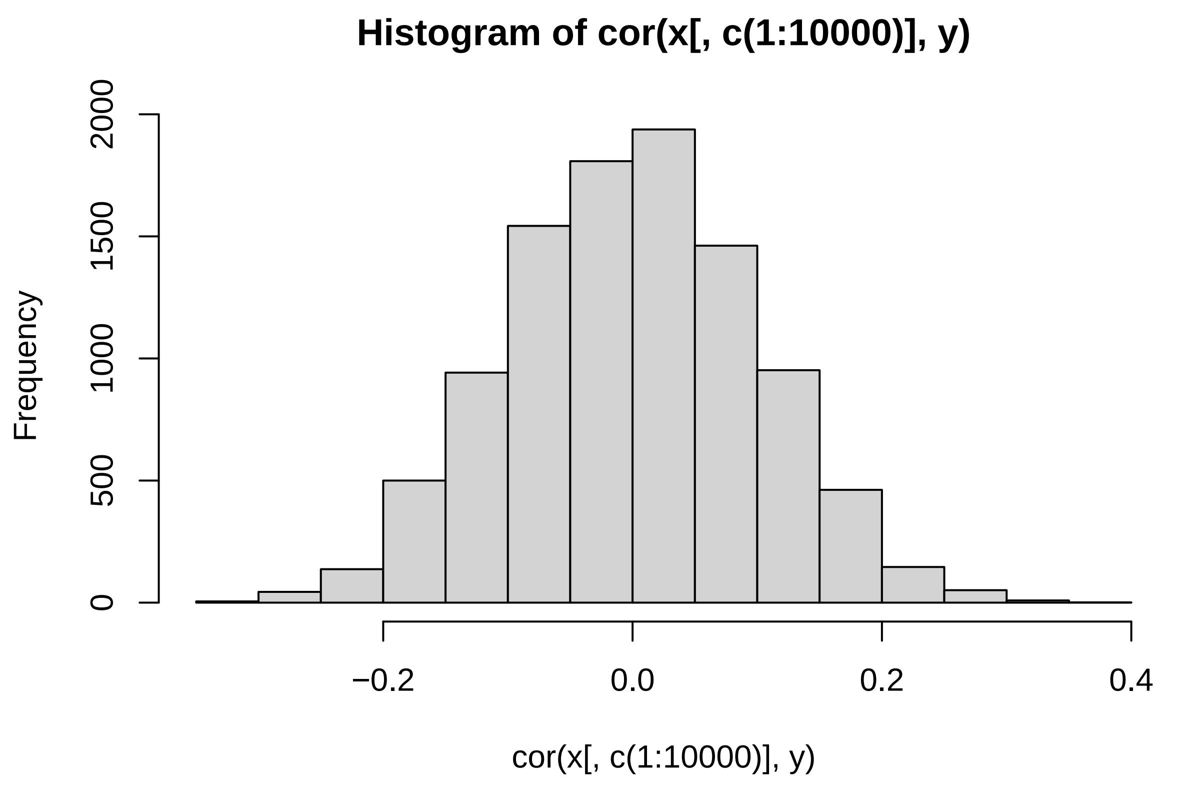
$$\begin{aligned} Cor[X, Y] &= \frac{Cov[X, Y]}{\sqrt{Var[X]Var[Y]}} \\ &= 0 \end{aligned}$$

And we are doing least squares to find relationship on this 0 correlation relationship in (d). Hence, (c)'s choice of $\hat{f}(x) = 0 \forall x$ seems to be a reasonable and good choice. And that answers why (c) gives us a lower estimation test error when comparing to (d). And my asnwer does support this.

2.

a.

```r
hist(cor(x[,c(1:10000)],y))
```



```
df <- tibble("index" = c(1:10000),
             "Correlation" = abs(cor(x[,c(1:10000)],y)))

top_10 <- top_n(df, 10, Correlation)
```

b.

```
set.seed(123)
train <- sample(100, 50)
train.x <- x[train, ]
val.x <- x[-train, ]

train.y <- y[train]
val.y <- y[-train]

lm_3 <- lm(train.y ~ train.x[,top_10$index])
mean((val.y - predict.lm(lm_3, data.frame(val.x[,top_10$index]))) ^ 2)
```

```
## [1] 1.574802
```

c.

```
set.seed(123)
train <- sample(100, 50)
```

```r
train.x <- x[train, ]
val.x <- x[-train, ]

train.y <- y[train]
val.y <- y[-train]

df_new <- tibble("index" = c(1:10000),
             "Correlation" = cor(train.x[,c(1:10000)],train.y))

top_10_new <- top_n(df_new, 10, Correlation)

lm_4 <- lm(train.y ~ train.x[,top_10_new$index])
mean((val.y - predict.lm(lm_4, data.frame(val.x[,top_10_new$index]))) ^ 2)
```

```
## [1] 1.626437
```

d.

In our finding, we found out that with the same $q = 10$ features, "Option 1" gave us a lower estimated test error when compare to "Option 2". And the reason for that is when selecting top 10 predictors, we used all the information/data we have for "Option 1", and that includes validation set, which is not appropriate when performing subset selection. Hence, we shall avoid using this and this way of modeling does not give us meaningful answers. Hence, despite having higher test error, we shall use "Option 2" to perform this task.

3.

a.

I choose a data set that is called "Superconductivty Data Data Set". It has $p = 81$ features. And the goal is to use all the features(extracted from the superconductor's chemical formula) to predict the superconducting critical temperature.

b.

```r
df <- read.csv("train.csv")

set.seed(123)
# We would first split the data into train & val sets
train <- sample(21263, 10632)

df_train <- df[train, ]
df_val <- df[-train, ]

lm_5 <- lm(critical_temp ~.-critical_temp, data = df_train)
test_error <- mean((df_val$critical_temp - predict.lm(lm_5, df_val[1:81])) ^ 2)
```

The test error(Test MSE) is 306.3443086. And I first use Validation Set Approach to split the data into train/validation set about half and half(10632:10631). Then, I perform Least Squares on the training data. And calculate the Test Error using validation set data.

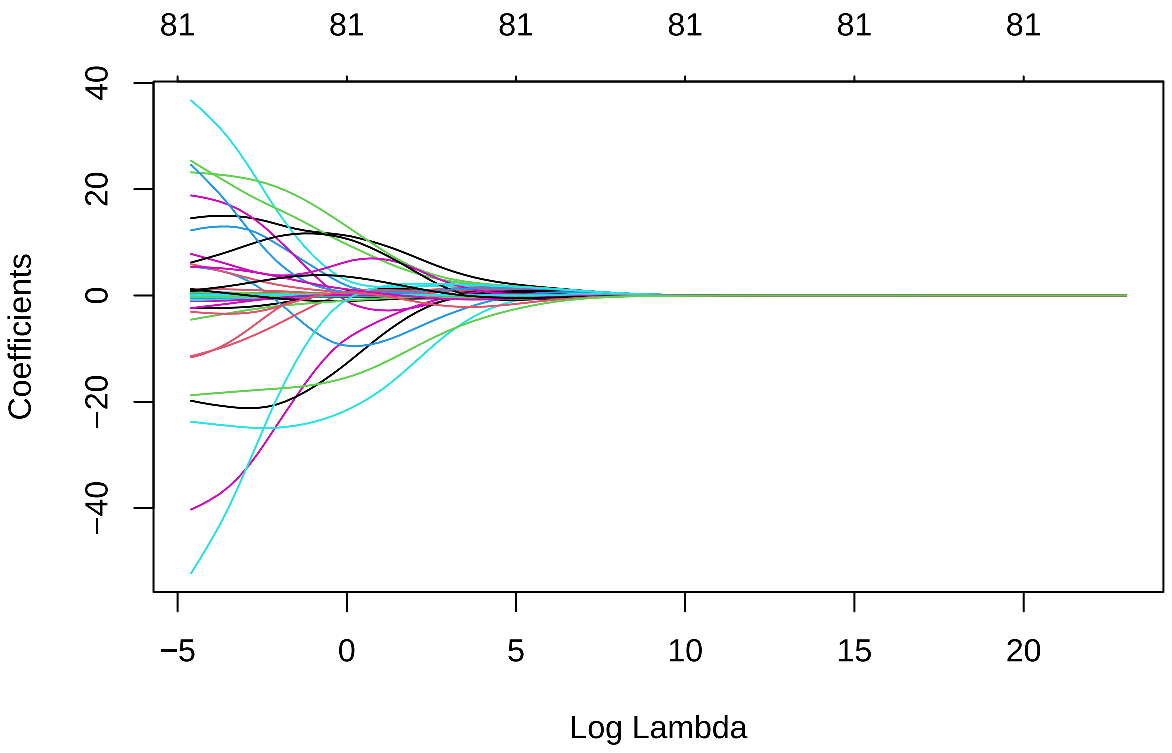
c.

```
lambdas <- 10^seq(10, -2, length=100)
ridge_mod <- glmnet(as.matrix(df_train[1:81]), as.matrix(df_train[82]), alpha = 0, lambda = lambdas)

dim(coef(ridge_mod))
```

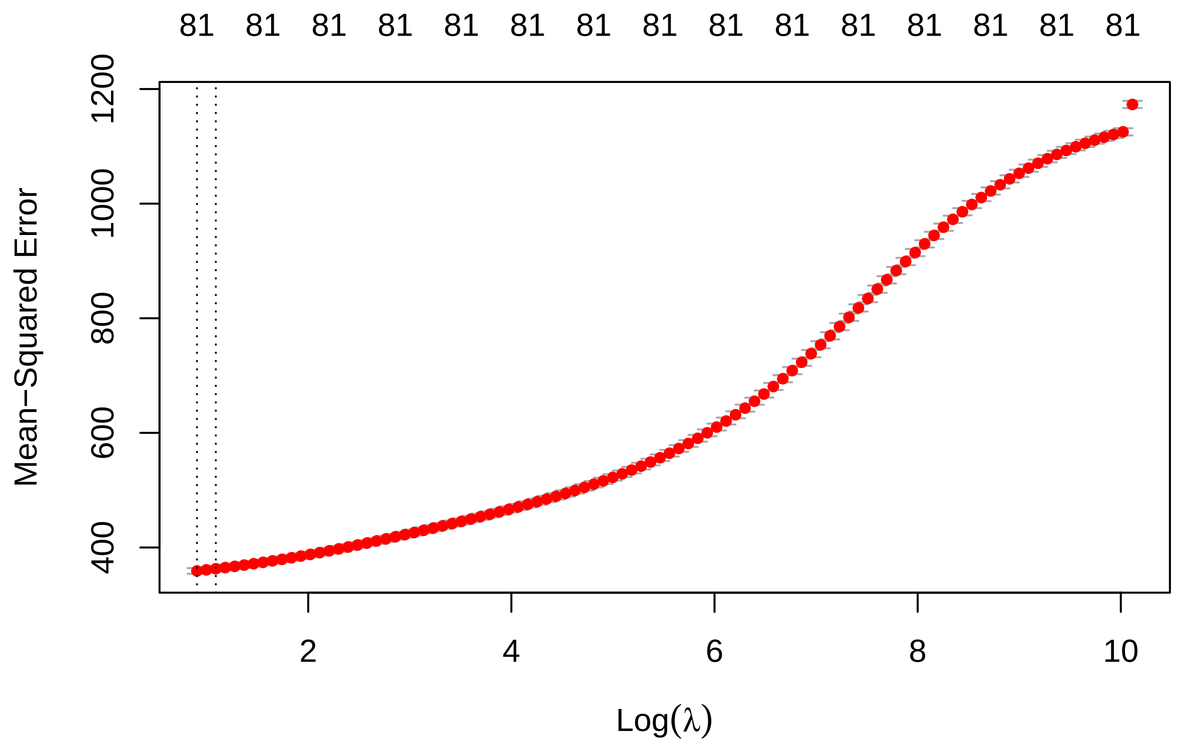
```
## [1]  82 100
```

```
plot(ridge_mod, xvar="lambda")
```



d.

```
cv.out <- cv.glmnet(as.matrix(df[1:81]), as.matrix(df[82]), alpha = 0)
plot(cv.out)
```



```
best_lambda <- cv.out$lambda.min

smallest_test_error <- min(cv.out$cvm)
```

When $\lambda = 2.47061$, we will have the smallest test error of 359.0993455.

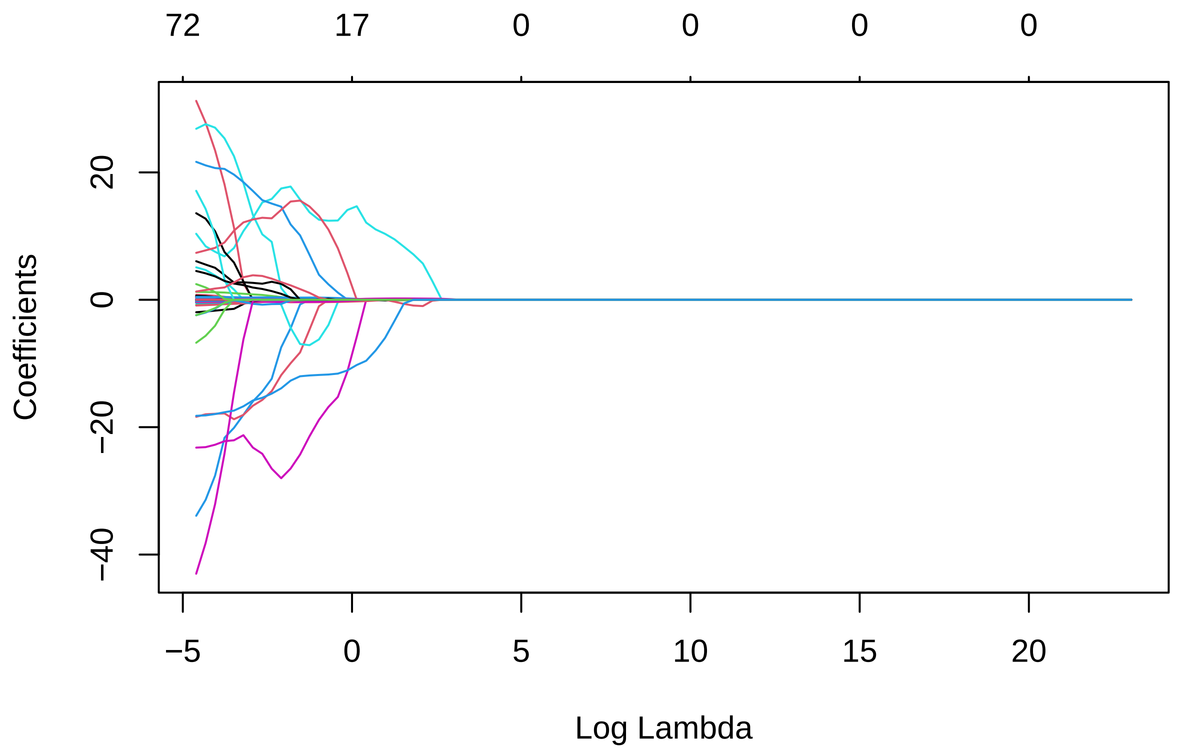
e.

```
lambdas <- 10^seq(10, -2, length=100)
lasso_mod <- glmnet(as.matrix(df_train[1:81]), as.matrix(df_train[82]), alpha = 1, lambda = lambdas)

dim(coef(lasso_mod))
```

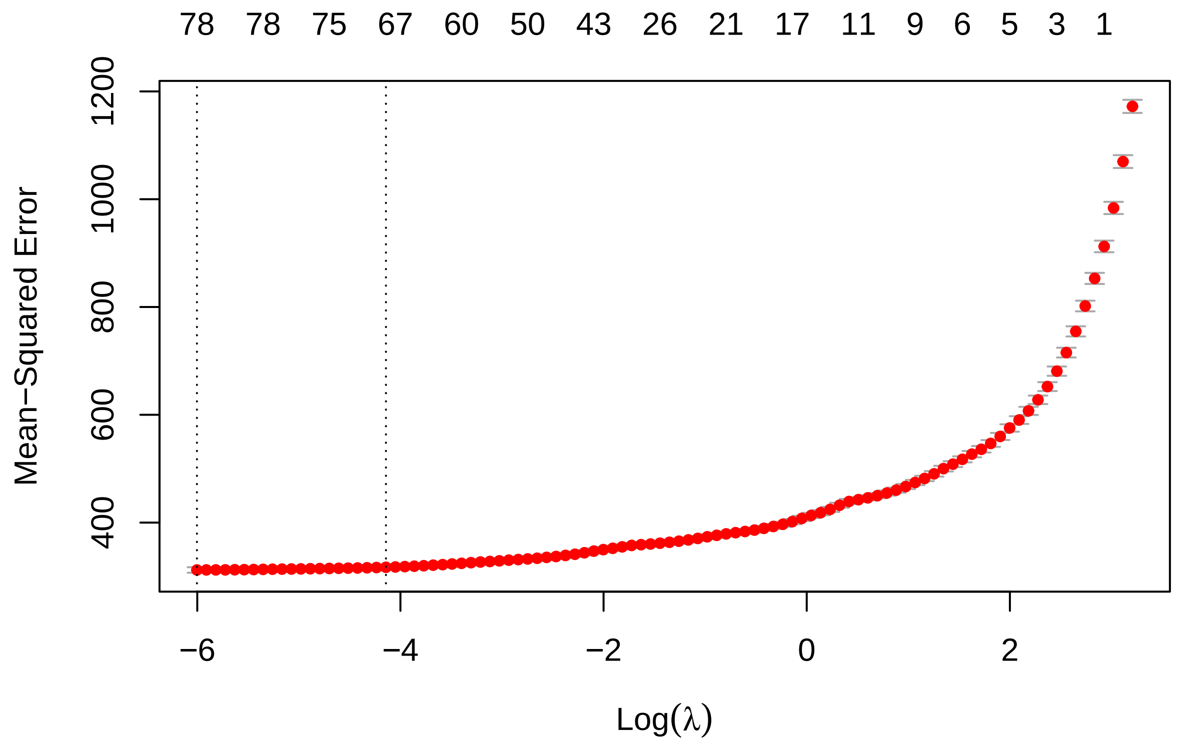
```
## [1]  82 100
```

```
plot(lasso_mod, xvar="lambda")
```



f.

```
cv.out <- cv.glmnet(as.matrix(df[1:81]), as.matrix(df[82]), alpha = 1)
plot(cv.out)
```



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

smallest_test_error <- min(cv.out$cvm)

coef(cv.out, s=best_lambda)
```

```
## 82 x 1 sparse Matrix of class "dgCMatrix"
##                                            s1
## (Intercept)                     -1.863123e+01
## number_of_elements              -3.712175e+00
## mean_atomic_mass                 4.681039e-01
## wtd_mean_atomic_mass            -4.009149e-01
## gmean_atomic_mass               -1.488603e-01
## wtd_gmean_atomic_mass            1.692463e-01
## entropy_atomic_mass             -3.515285e+01
## wtd_entropy_atomic_mass          7.526031e+00
## range_atomic_mass                2.027316e-01
## wtd_range_atomic_mass            1.599796e-02
## std_atomic_mass                 -3.357037e-01
## wtd_std_atomic_mass             -1.519477e-01
## mean_fie                         2.551794e-03
## wtd_mean_fie                     1.454059e-02
## gmean_fie                       -9.377466e-04
## wtd_gmean_fie                    5.784444e-03
## entropy_fie                     -9.077222e+00
## wtd_entropy_fie                  4.009666e+01
## range_fie                        6.454953e-02
## wtd_range_fie                    1.818315e-02
## std_fie                         -1.649075e-01
## wtd_std_fie                     -2.571322e-02
## mean_atomic_radius               7.352046e-01
## wtd_mean_atomic_radius           1.203987e+00
## gmean_atomic_radius             -1.021587e+00
## wtd_gmean_atomic_radius         -8.471990e-01
## entropy_atomic_radius            .
## wtd_entropy_atomic_radius        3.457795e+01
## range_atomic_radius              2.299207e-01
## wtd_range_atomic_radius         -8.925065e-02
## std_atomic_radius               -9.455527e-01
## wtd_std_atomic_radius            2.613079e-01
## mean_Density                    -4.182863e-03
## wtd_mean_Density                -6.588360e-04
## gmean_Density                    5.411907e-04
## wtd_gmean_Density                2.796627e-03
## entropy_Density                  1.417015e+01
## wtd_entropy_Density             -1.572851e+01
## range_Density                   -1.450341e-03
## wtd_range_Density                4.738824e-05
## std_Density                      5.021856e-03
## wtd_std_Density                 -7.272521e-04
## mean_ElectronAffinity           -1.459997e-01
## wtd_mean_ElectronAffinity        5.926216e-01
## gmean_ElectronAffinity           2.096564e-01
## wtd_gmean_ElectronAffinity      -6.346540e-01
## entropy_ElectronAffinity         1.465220e+00
## wtd_entropy_ElectronAffinity    -1.939434e+01
## range_ElectronAffinity          -3.733321e-01
## wtd_range_ElectronAffinity      -1.556939e-01
## std_ElectronAffinity             1.264752e+00
## wtd_std_ElectronAffinity        -5.661835e-01
## mean_FusionHeat                  8.692804e-01
## wtd_mean_FusionHeat             -1.050186e+00
## gmean_FusionHeat                -7.515246e-01
## wtd_gmean_FusionHeat             7.960367e-01
## entropy_FusionHeat              -1.374822e+01
## wtd_entropy_FusionHeat           2.269922e+01
## range_FusionHeat                -3.909656e-01
## wtd_range_FusionHeat             4.796326e-01
## std_FusionHeat                   1.434330e-01
## wtd_std_FusionHeat               1.398599e-01
## mean_ThermalConductivity        -5.544018e-02
## wtd_mean_ThermalConductivity     5.016274e-01
## gmean_ThermalConductivity       -6.446868e-02
## wtd_gmean_ThermalConductivity   -3.066998e-01
## entropy_ThermalConductivity      8.413621e+00
## wtd_entropy_ThermalConductivity  3.054302e+00
## range_ThermalConductivity       -8.039620e-02
## wtd_range_ThermalConductivity   -2.100160e-01
## std_ThermalConductivity          2.334896e-01
## wtd_std_ThermalConductivity      1.556576e-02
## mean_Valence                     .
## wtd_mean_Valence                 2.349723e-04
## gmean_Valence                    4.471047e+00
## wtd_gmean_Valence               -4.455900e+00
## entropy_Valence                  4.594607e+01
## wtd_entropy_Valence             -6.485796e+01
## range_Valence                    4.799753e+00
## wtd_range_Valence                3.775408e-01
## std_Valence                      .
## wtd_std_Valence                 -1.928312e+01
```

Best lambda in this case is 0.0024706, we will have the smallest test error of 312.1169266. As we can see from above, this time, LASSO model uses all features except "entropy_atomic_radius", "mean_Valence", and "std_Valence".

4.

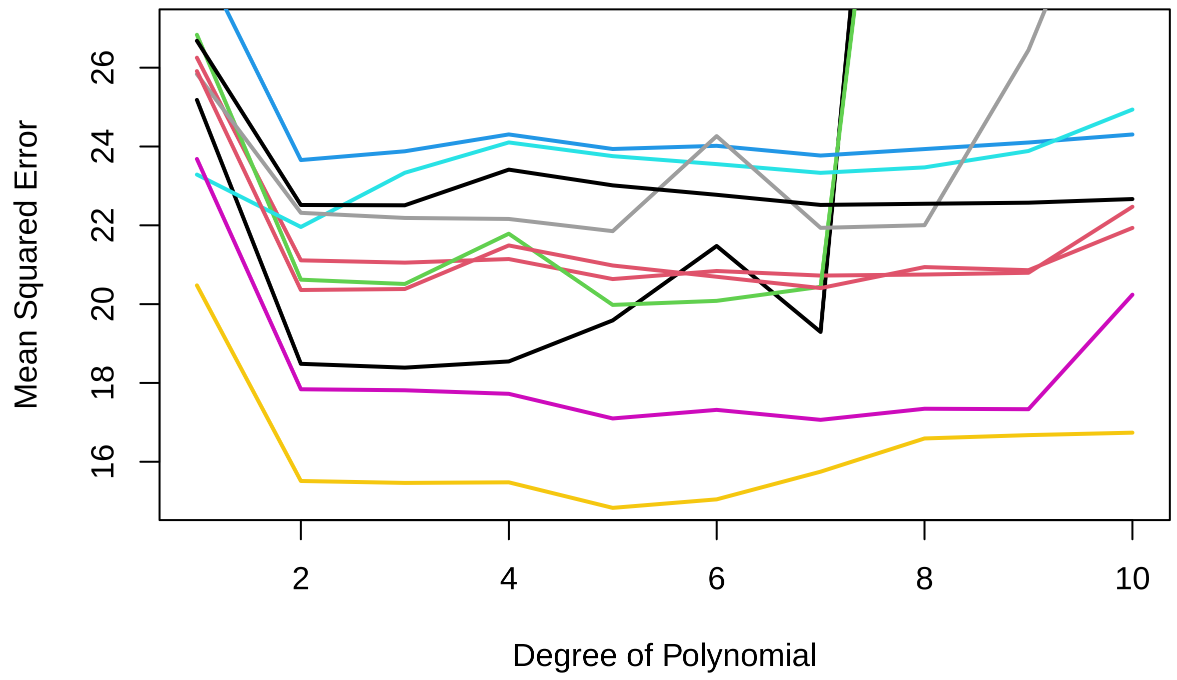
a.

```r
data(Auto)
# Split data
set.seed(434)
degrees <- c(1:10)
train <- sample(392, 196)
test_MSE <- matrix(NA, 10, 10)
```

```r
for(i in degrees){
  lm.fit <- lm(mpg ~ poly(horsepower, i), data = Auto, subset = train)
  test_MSE[1, i] <- mean((Auto$mpg - predict(lm.fit, Auto))[-train] ^ 2)
}

plot(x = degrees, y = test_MSE[1, ],
     lwd = 2,
     type = "l",
     lty = 1,
     ylim = c(15,27),
     xlab = "Degree of Polynomial",
     ylab = "Mean Squared Error")


for (i in 2:10) {
  train <- sample(392, 196)
  for (j in degrees) {
    lm.fit <- lm(mpg ~ poly(horsepower, j),
                 data = Auto,
                 subset = train)
    test_MSE[i, j] <- mean((Auto$mpg - predict(lm.fit, Auto))[-train] ^ 2)
  }
  lines(x = degrees, y = test_MSE[i, ], col = i,
        lty = 1,
        lwd = 2)
}
```



```r
kable(tibble(
  Degree = degrees,
  "Mean Validation MSE" = rowMeans(test_MSE)
))
```

Degree	Mean Validation MSE
1	56.72504
2	21.57787
3	30.38250
4	24.48737
5	23.56148
6	18.34581
7	16.25580
8	24.18641
9	23.12112
10	21.39636

As we can see from the above table, Mean Test MSE is the smallest when degree is at 7, with a value of 16.25580.
I found out that the test error is **not** monotonically decreasing. And after a sharp decrease in Degree 2, we see little to no improvements, and sometimes even worse test error with higher degrees.

b.

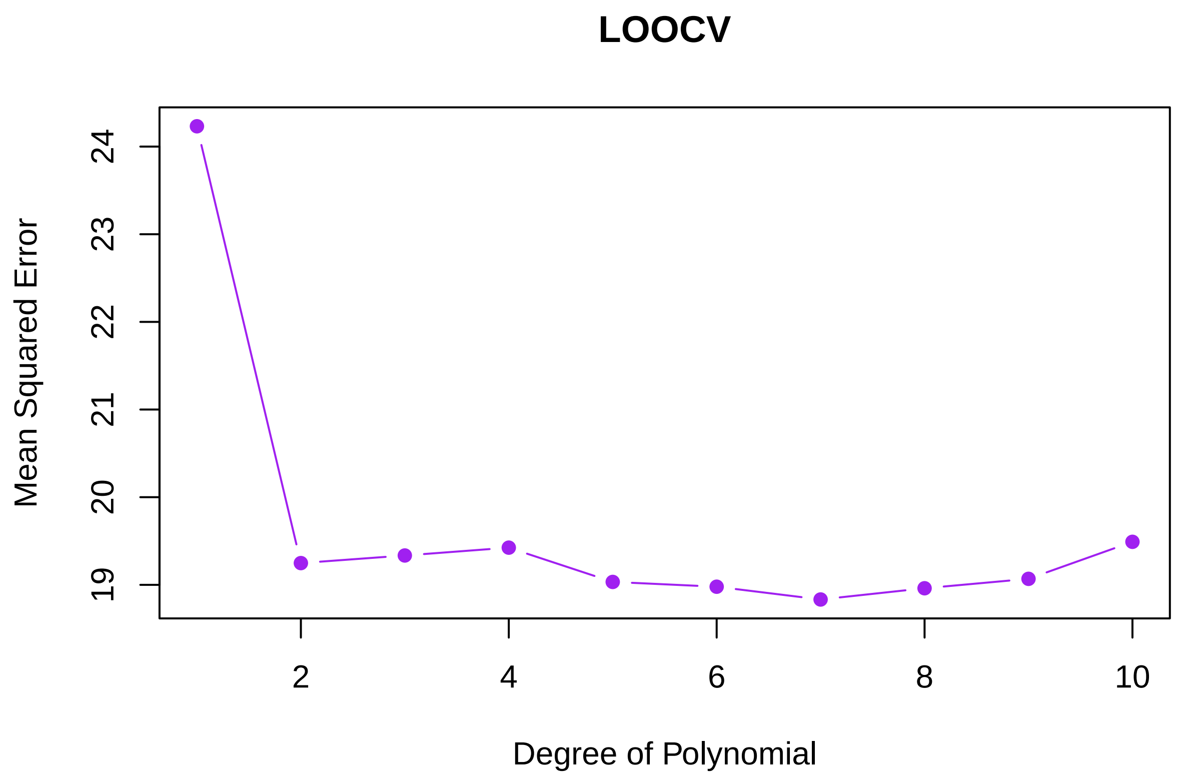
```r
# LOOCV
set.seed(435)
cv_error <- rep(NA, 10)

for( i in 1:10){
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv_error[i] <- cv.glm(Auto, glm.fit)$delta[1]
}

kable(tibble(
  Degree = degrees,
  "LOOCV" = cv_error
))
```

Degree	LOOCV
1	24.23151
2	19.24821
3	19.33498
4	19.42443
5	19.03321
6	18.97864
7	18.83305
8	18.96115
9	19.06863
10	19.49093

```r
plot(x = degrees,
     y = cv_error,
     xlab = "Degree of Polynomial",
     ylab = "Mean Squared Error",
     main = "LOOCV",
     col = "purple",
     type = "b",
     pch = 16)
```



With LOOCV, the lowest error occurs at Degree equals 7, with mean test error of 18.83305.
I found out that the test error is **not** monotonically decreasing. And after a sharp decrease in Degree 2, we see little to no improvements, and sometimes even worse test error with higher degrees.

c.

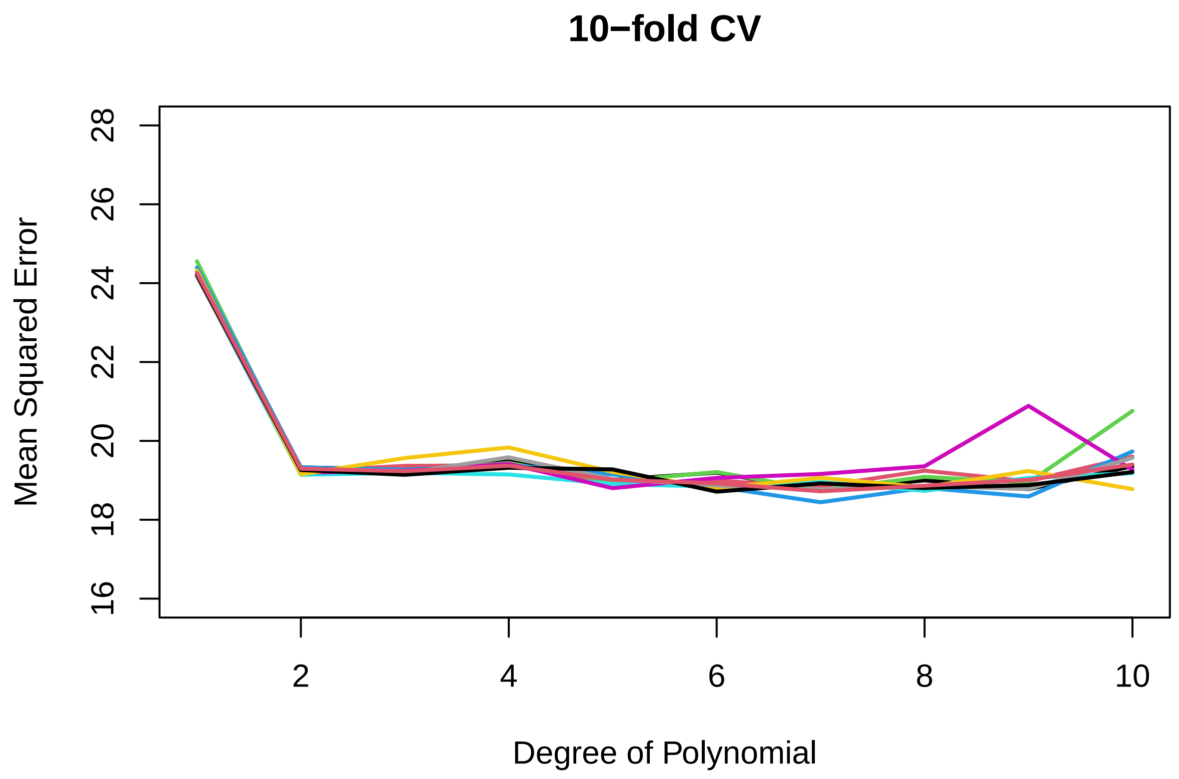
```
set.seed(435)

cv_error_10 <- matrix(0, 10, 10)

for (i in 1:10) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  cv_error_10[1, i] <- cv.glm(Auto, glm.fit, K = 10)$delta[1]
}
plot(
  x = c(1:10),
  y = cv_error_10[1,],
  lty = 1,
  type = "l",
  lwd = 2,
  ylim = c(16, 28),
  xlab = "Degree of Polynomial",
```

```r
  ylab = "Mean Squared Error",
  main = "10-fold CV"
)


for (i in 2:10) {
  for (j in 1:10) {
    glm.fit <- glm(mpg ~ poly(horsepower, j), data = Auto)
    cv_error_10[i, j] <- cv.glm(Auto, glm.fit, K = 10)$delta[1]
  }
  lines(x = c(1:10), y = cv_error_10[i, ],
        col = i,
        lwd = 2)
}
```



```r
kable(tibble(
  Degree = degrees,
  "Mean Validation MSE" = rowMeans(cv_error_10)
))
```

Degree	Mean Validation MSE
1	19.64669
2	19.67251
3	19.81529
4	19.59466
5	19.53472
6	19.85939

Degree	Mean Validation MSE
7	19.67893
8	19.60792
9	19.57428
10	19.60936

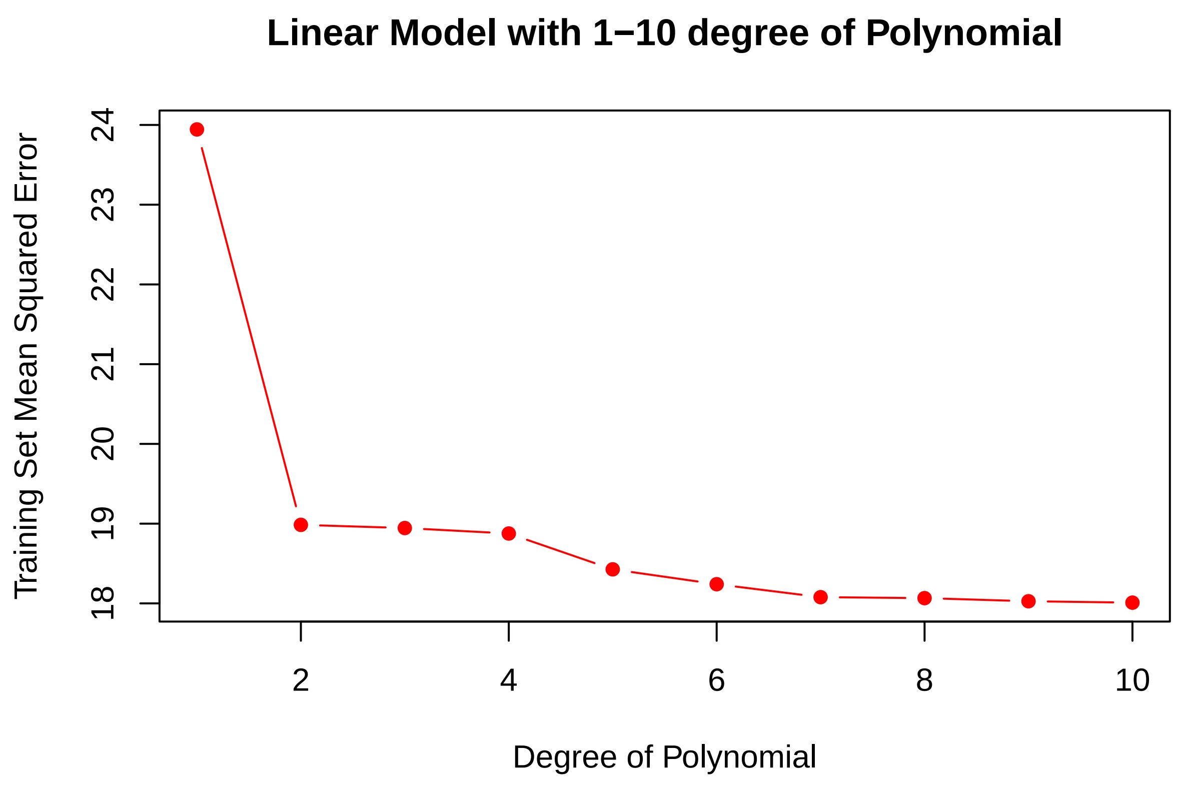
At degree equals to 5, we have the smallest mean 10 fold CV error of 19.53472. I found out that the test error is **not** monotonically decreasing. And after a sharp decrease in Degree 2, we see little to no improvements, and sometimes even worse test error with higher degrees.

d.

```r
set.seed(435)
degrees <- c(1:10)
train_MSE <- rep(NA, 10)

for(i in degrees){
  lm.fit <- lm(mpg ~ poly(horsepower, i), data = Auto)
  train_MSE[i] <- mean(lm.fit$residuals^2)
}

plot(x = degrees,
     y = train_MSE,
     xlab = "Degree of Polynomial",
     ylab = "Training Set Mean Squared Error",
     main = "Linear Model with 1-10 degree of Polynomial",
     col = "red",
     type = "b",
     pch = 16)
```



```
kable(tibble(
  Degree = degrees,
  "Mean Validation MSE" = train_MSE
))
```

Degree	Mean Validation MSE
1	23.94366
2	18.98477
3	18.94499
4	18.87633
5	18.42697
6	18.24065
7	18.07817
8	18.06613
9	18.02697
10	18.00953

With increase in Degree of Polynomial, our Training MSE will monotonically decrease, and the smallest Train MSE occurred at $degree = 10$.

e.

```
set.seed(435)

lm.fit <- lm(mpg ~ poly(horsepower, 10), data = Auto)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = mpg ~ poly(horsepower, 10), data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.7081  -2.5904  -0.1922   2.2859  14.8338
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)              23.4459     0.2174 107.840   <2e-16 ***
## poly(horsepower, 10)1  -120.1377     4.3046 -27.909   <2e-16 ***
## poly(horsepower, 10)2    44.0895     4.3046  10.242   <2e-16 ***
## poly(horsepower, 10)3    -3.9488     4.3046  -0.917   0.3595
## poly(horsepower, 10)4    -5.1878     4.3046  -1.205   0.2289
## poly(horsepower, 10)5    13.2722     4.3046   3.083   0.0022 **
## poly(horsepower, 10)6    -8.5462     4.3046  -1.985   0.0478 *
## poly(horsepower, 10)7     7.9806     4.3046   1.854   0.0645 .
## poly(horsepower, 10)8     2.1727     4.3046   0.505   0.6140
## poly(horsepower, 10)9    -3.9182     4.3046  -0.910   0.3633
## poly(horsepower, 10)10   -2.6146     4.3046  -0.607   0.5440
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.305 on 381 degrees of freedom
## Multiple R-squared:  0.7036, Adjusted R-squared:  0.6958
## F-statistic: 90.45 on 10 and 381 DF,  p-value: < 2.2e-16
```

As we can see from summary table, the intercept, first, and second degree polynomials are statistically significant for having p-value<0.001. And Degree 5, p-value<0.01, Degree 6, p-value < 0.05, Degree 7, p-value < 0.1. However, after Degree 2, all the rest are minor changes/improvement to our test error. Hence I would say using 2 degree of freedom polynomial fit is a good enough practice. And it does match the graph we plotted from (a)-(d), where we saw a sharp decrease in Test/Train MSE at 2 degree of polynomial.

5.

a.

$$\begin{aligned}
\arg\min_{\beta} \sum_{i=1}^{n}(y_i - \beta x_i)^2 &= \frac{\partial}{\partial \beta}\sum_{i=1}^{n}(y_i - \beta x_i)^2 = 0 \\
&-2\sum_{i=1}^{n} x_i(y_i - \beta x_i) = 0 \\
&\sum_{i=1}^{n} x_i y_i - \beta \sum_{i=1}^{n} x_i^2 = 0 \\
&\hat{\beta}^{L.S.} = \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}
\end{aligned}$$

b.

$$\begin{aligned}
\arg\min_{\beta} \sum_{i=1}^{n}(y_i - \beta x_i)^2 + \lambda\beta^2 = \frac{\partial}{\partial\beta}\left[\sum_{i=1}^{n}(y_i - \beta x_i)^2 + \lambda\beta^2\right] &= 0 \\
-2\sum_{i=1}^{n} x_i(y_i - \beta x_i) + 2\lambda\beta &= 0 \\
\sum_{i=1}^{n} x_i y_i - \beta\sum_{i=1}^{n} x_i^2 &= \lambda\beta \\
\beta\left(\sum_{i=1}^{n} x_i^2 + \lambda\right) &= \sum_{i=1}^{n} x_i y_i \\
\hat{\beta}^{L.S.} &= \frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2 + \lambda}
\end{aligned}$$

c.

From part a),

$$\begin{aligned}
E[\hat{\beta}^{L.S.}] &= E\left[\frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2}\right] \\
&= E\left[\frac{\sum_{i=1}^{n} x_i(3x_i + \varepsilon))}{\sum_{i=1}^{n} x_i^2}\right] \\
&= E\left[\frac{\sum_{i=1}^{n} 3x_i^2 + \varepsilon\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} x_i^2}\right]
\end{aligned}$$

Since

$$\begin{aligned}
X \text{ and } \varepsilon \text{ are independent} \\
&= \frac{3E[\sum_{i=1}^{n} x_i^2] + E[\varepsilon] \times E[\sum_{i=1}^{n} x_i]}{E[\sum_{i=1}^{n} x_i^2]} \\
&= \frac{3E[\sum_{i=1}^{n} x_i^2]}{E[\sum_{i=1}^{n} x_i^2]} \\
&= 3 = \beta
\end{aligned}$$

Hence, $\hat{\beta}$ is an unbiased estimator of β.

d.

$$\begin{aligned}
E[\hat{\beta}^{ridge}] &= E\left[\frac{\sum_{i=1}^{n} x_i y_i}{\sum_{i=1}^{n} x_i^2 + \lambda}\right] \\
&= E\left[\frac{\sum_{i=1}^{n} x_i(3x_i + \varepsilon))}{\sum_{i=1}^{n} x_i^2 + \lambda}\right] \\
&= E\left[\frac{\sum_{i=1}^{n} 3x_i^2 + \varepsilon\sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} x_i^2 + \lambda}\right]
\end{aligned}$$

Since

$$X \text{ and } \varepsilon \text{ are independent}$$

$$= \frac{3E[\sum_{i=1}^n x_i^2] + E[\varepsilon] \times E[\sum_{i=1}^n x_i]}{E[\sum_{i=1}^n x_i^2] + E[\lambda]}$$

$$= \frac{3E[\sum_{i=1}^n x_i^2]}{E[\sum_{i=1}^n x_i^2] + E[\lambda]}$$

Since x is fixed, and lambda is a constant

$$= \frac{3\sum_{i=1}^n x_i^2}{\sum_{i=1}^n x_i^2 + \lambda}$$

Hence, we have shown that Ridge regression will give us a biased estimator $\hat{\beta}^{ridge}$ for β. However, as $\lambda \to 0$, we will have an less biased estimator for β. And finally when $E[\lambda] = 0$, $\hat{\beta}^{ridge}$ will be unbiased.

e.

$$\begin{aligned}
Var[\hat{\beta}^{L.S.}] &= E\left[\hat{\beta} - E[\hat{\beta}]\right]^2 \\
&= E[\hat{\beta} - \beta]^2 \\
&= E\left[\frac{\sum_{i=1}^n 3x_i^2 + \varepsilon\sum_{i=1}^n x_i}{\sum_{i=1}^n x_i^2} - 3\right]^2 \\
&= E\left[\frac{\varepsilon\sum_{i=1}^n x_i}{\sum_{i=1}^n x_i^2}\right]^2 \\
&= \frac{1}{\sum_{i=1}^n x_i^4} E\left[\left(\sum_{i=1}^n \varepsilon x_i\right)^2\right] \\
&= \frac{1}{\sum_{i=1}^n x_i^4} E\left[(x_1\varepsilon + x_2\varepsilon + ... + x_n\varepsilon)^2\right] \\
&= \frac{1}{\sum_{i=1}^n x_i^4} E\left[x_1^2\varepsilon^2 + x_2^2\varepsilon^2 + ... + x_n^2\varepsilon^2 + 2x_1x_2\varepsilon^2 + ... + 2x_{n-1}x_n\varepsilon^2\right] \\
&= \frac{E[\varepsilon^2]}{\sum_{i=1}^n x_i^4} E[x_1^2 + x_2^2 + ... + x_n^2 + 2x_1x_2 + ... + 2x_{n-1}x_n]
\end{aligned}$$

Because we are given $Cov(\varepsilon_i, \varepsilon_i') = 0 \ \forall i \neq i'$

$$\begin{aligned}
&= \frac{E[\varepsilon^2]}{\sum_{i=1}^n x_i^4} E[x_1^2 + x_2^2 + ... + x_n^2] \\
&= \frac{Var[\varepsilon] + E[\varepsilon]^2}{\sum_{i=1}^n x_i^4} \sum_{i=1}^n x_i^2 \\
&= \frac{\sigma^2}{\sum_{i=1}^n x_i^2}
\end{aligned}$$

f.

$$
\begin{aligned}
Var[\hat{\beta}^{ridge}] &= E\left[\hat{\beta}^{ridge} - E[\hat{\beta}^{ridge}]\right]^2 \\
&= E\left[\frac{\sum_{i=1}^n 3x_i^2 + \sum_{i=1}^n \varepsilon_i x_i}{\sum_{i=1}^n x_i^2 + \lambda} - \frac{3\sum_{i=1}^n x_i^2}{\sum_{i=1}^n x_i^2 + \lambda}\right]^2 \\
&= E\left[\frac{\sum_{i=1}^n \varepsilon_i x_i}{\sum_{i=1}^n x_i^2 + \lambda}\right]^2 \\
&= \frac{1}{(\sum_{i=1}^n x_i^2 + \lambda)^2} E\left[\left(\sum_{i=1}^n \varepsilon x_i\right)^2\right] \\
&= \frac{1}{(\sum_{i=1}^n x_i^2 + \lambda)^2} E\left[(x_1\varepsilon + x_2\varepsilon + ... + x_n\varepsilon)^2\right] \\
&= \frac{1}{(\sum_{i=1}^n x_i^2 + \lambda)^2} E\left[x_1^2\varepsilon^2 + x_2^2\varepsilon^2 + ... + x_n^2\varepsilon^2 + 2x_1x_2\varepsilon^2 + ... + 2x_{n-1}x_n\varepsilon^2\right] \\
&= \frac{E[\varepsilon^2]}{(\sum_{i=1}^n x_i^2 + \lambda)^2} E[x_1^2 + x_2^2 + ... + x_n^2 + 2x_1x_2 + ... + 2x_{n-1}x_n]
\end{aligned}
$$

Because we are given $Cov(\varepsilon_i, \varepsilon_i') = 0 \; \forall i \neq i'$

$$
\begin{aligned}
&= \frac{E[\varepsilon^2]}{(\sum_{i=1}^n x_i^2 + \lambda)^2} E[x_1^2 + x_2^2 + ... + x_n^2] \\
&= \frac{Var[\varepsilon] + E[\varepsilon]^2}{(\sum_{i=1}^n x_i^2 + \lambda)^2} \sum_{i=1}^n x_i^2 \\
&= \frac{\sigma^2}{(\sum_{i=1}^n x_i^2 + \lambda)^2} \sum_{i=1}^n x_i^2
\end{aligned}
$$

g.

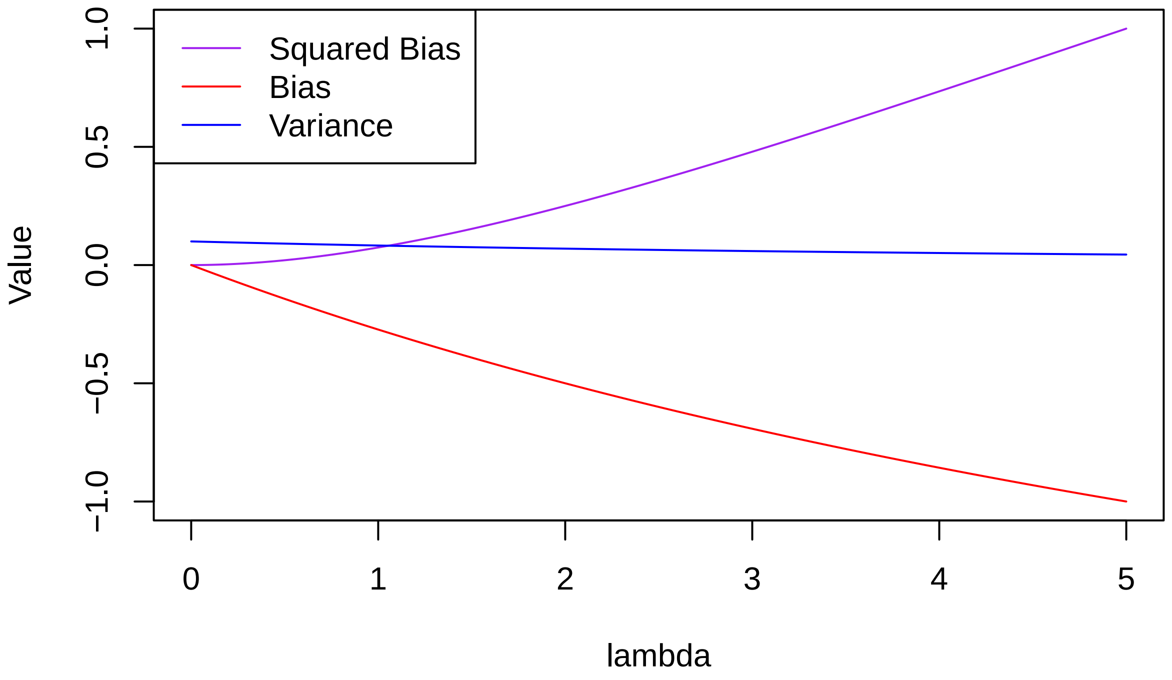
So we have

$$
\begin{aligned}
Bias[\hat{\beta}^{ridge}] &= E[\hat{\beta}^{ridge}] - \beta \\
&= \frac{3\sum_{i=1}^n x_i^2}{\sum_{i=1}^n x_i^2 + \lambda} - 3 \\
&= 3\left(\frac{\sum_{i=1}^n x_i^2}{\sum_{i=1}^n x_i^2 + \lambda} - \frac{\sum_{i=1}^n x_i^2 + \lambda}{\sum_{i=1}^n x_i^2 + \lambda}\right) \\
&= 3\frac{-\lambda}{\sum_{i=1}^n x_i^2 + \lambda} \\
Bias^2[\hat{\beta}^{ridge}] &= 9\frac{\lambda^2}{(\sum_{i=1}^n x_i^2 + \lambda)^2}
\end{aligned}
$$

And suppose we have $\sum_{i=1}^n x_i^2 = 10$, $\sigma = 1$.

```r
x <- 10
lambda <- seq(0, 5, by = 0.001)
sigma <- 1
```

```r
plot(lambda, (-3*lambda/(x + lambda))^2, type = "l", col = "purple",
     ylab = "Value", ylim = c(-1,1))
lines(lambda, -3*lambda/(x + lambda), type = "l", col = "red")
lines(lambda, 10*(sigma^2/(10 + lambda)^2), col = "blue")
legend("topleft",
       c("Squared Bias", "Bias", "Variance"),
       col = c("purple", "red", "blue"),
       lty = 1)
```



As we can see from above, as $lambda$ increases, $Bias^2$ increases and takes the value 0 at 0. And it's not hard to see that $\hat{\beta}^{ridge} = \hat{\beta}^{L.S.}$ if $\lambda = 0$.(This is the reason why Bias/Squared Bias is 0 at 0). Then, as λ goes to 9, since $\lambda \to \infty$ shows that the ridge estimate would be 0 and hence bias would be 3. For variance though, we see that increase λ reduce variances. Because larger penalty that introduced by λ will force the weight to shrink towards zero and reduce the scale and variance. Thus, we will see that larger penalty in ridge-regression increases the Squared Bias for the estimate and reduces the variance. a.k.a, Bias-Variance Trade-off.