HW3_435

```r
set.seed(435)
y <- rnorm(100)
x <- matrix(rnorm(10000*100), ncol=10000)

# Part 1civ)
set.seed(435)
train <- sample(100, 50)
validation_y <- y[-train]
(test_error <- mean(validation_y^2))
```

```
## [1] 0.8158012
```

```r
# Part 1d)
train_y <- y[train]
train_x <- x[train, ]
validation_x <- x[-train, ]

fit <- lm(train_y ~ train_x)
values <- predict(fit, data.frame(validation_x))
```

```
## Warning in predict.lm(fit, data.frame(validation_x)): prediction from a rank-
## deficient fit may be misleading
```

```r
(test_error1 <- mean((values - validation_y)^2))
```

```
## [1] 1.660356
```

1a) $Y = \varepsilon$

1b) irreducible error $= Var(\varepsilon) = Y$

1c i) Bias $= E(\hat{f}(x)) - E(f(x))$
$= 0 - E(f(x))$
$= -E(f(x))$
$= 0$

ii) $Var = 0$ Since the procedure always predicts 0 for each observation

iii) Expected prediction error $= E(y_0 - \hat{f}(x_0))^2 = Var(\hat{f}(x_0)) + [Bias(\hat{f}(x_0))]^2 + Var(\varepsilon)$
$= 0 + 0 + 1$
$= 1$

1civ) The expected test error 0.8158012

1cv) The expected prediction error for this procedure is 1, while the estimated test error of this procedure is 0.8158012 which is close to 1.

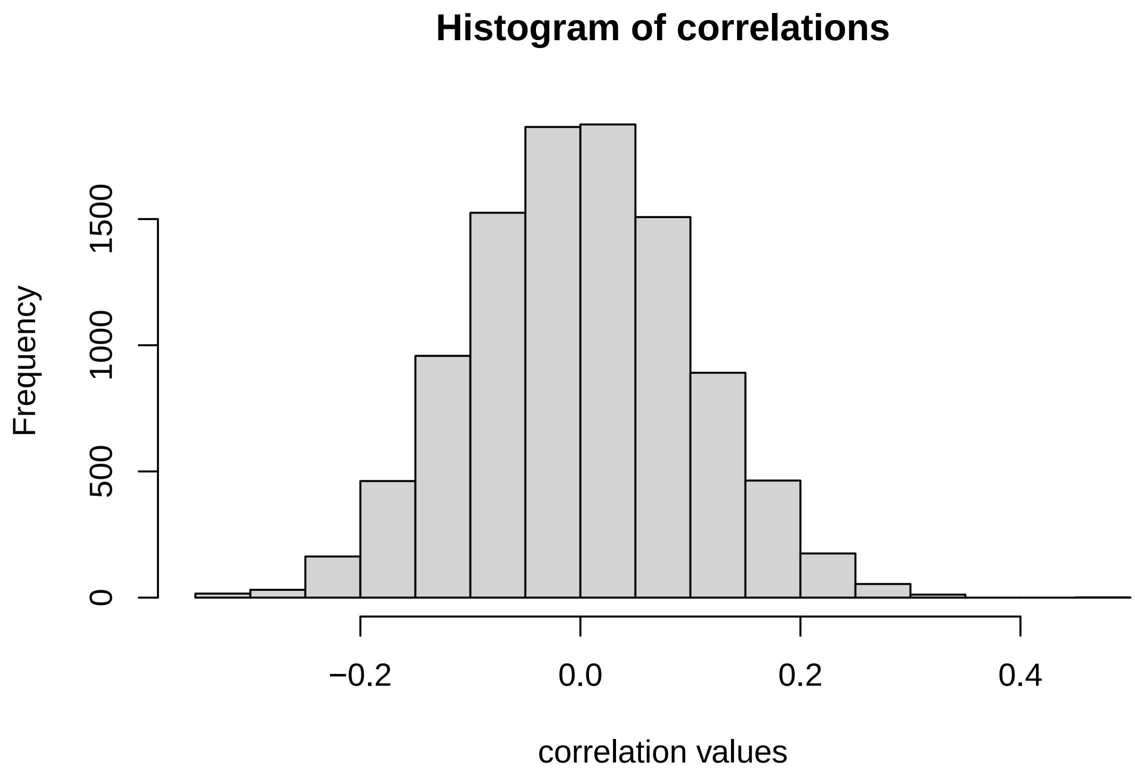
1d) The expected test error is 1.660356.

1e) part c) has a smaller estimated test error. Part c) also has a smaller bias and smaller variance because they are both zero since the procedure just predicts 0 for every observation. Therefore, it also makes sense that part c) has a smaller estimated test error.

```
# Part 2a
cor_values <- data.frame(cor_values = cor(x, y), index = seq(1, 10000, 1))
cor_values_sort <- cor_values %>% arrange(desc(abs(cor_values)))
(largest <- abs(cor_values_sort[0:10, ]))
```

```
##    cor_values index
## 1   0.4581637  1298
## 2   0.3476051  3673
## 3   0.3458173  5512
## 4   0.3457527  2455
## 5   0.3399391   491
## 6   0.3387277  7542
## 7   0.3365027  2036
## 8   0.3355233  9240
## 9   0.3345952  5388
## 10  0.3286617  3447
```

```
hist(cor_values[,1],
     main = "Histogram of correlations", xlab = "correlation values")
```




```r
# Part 2b
indices <- largest[, 2]
(fit1 <- lm(train_y ~ train_x[, indices]))
```

```
##
## Call:
## lm(formula = train_y ~ train_x[, indices])
##
## Coefficients:
##         (Intercept)   train_x[, indices]1   train_x[, indices]2
##            0.274093              0.160772              0.246239
##  train_x[, indices]3   train_x[, indices]4   train_x[, indices]5
##           -0.255160             -0.001556              0.203308
##  train_x[, indices]6   train_x[, indices]7   train_x[, indices]8
##            0.207197              0.116454              0.115369
##  train_x[, indices]9  train_x[, indices]10
##           -0.233546             -0.193885
```

```r
values1 <- predict(fit1, data.frame(validation_x))
(test_error2 <- mean((values1 - validation_y)^2))
```

```
## [1] 1.556062
```

```r
# Part 2c
cor_values1 <- data.frame(cor_values1 = cor(train_x, train_y),
                          index = seq(1, 10000, 1))
cor_values_sort1 <- cor_values1 %>% arrange(desc(abs(cor_values1)))
(largest1 <- cor_values_sort1[0:10, ])
```

```
##    cor_values1 index
## 1   -0.5417270  5157
## 2    0.4786239  9457
## 3   -0.4775034  9017
## 4    0.4709132  9715
## 5   -0.4654072  9492
## 6    0.4624135  2023
## 7    0.4614542  3673
## 8   -0.4605599  6960
## 9   -0.4594087  3283
## 10   0.4582407  6812
```

```r
indices1 <- largest1[, 2]
(fit2 <- lm(train_y ~ train_x[, indices1]))
```

```
##
## Call:
## lm(formula = train_y ~ train_x[, indices1])
##
## Coefficients:
##          (Intercept)   train_x[, indices1]1   train_x[, indices1]2
##              0.19011               -0.25205                0.26962
##  train_x[, indices1]3   train_x[, indices1]4   train_x[, indices1]5
##             -0.30675               -0.10118               -0.21047
##  train_x[, indices1]6   train_x[, indices1]7   train_x[, indices1]8
##              0.18218                0.27933               -0.09753
##  train_x[, indices1]9  train_x[, indices1]10
##             -0.03380                0.29935
```

```r
values2 <- predict(fit2, data.frame(validation_x))
(test_error3 <- mean((values2 - validation_y)^2))
```

```
## [1] 1.41439
```

2d) We can see here from part 2b and 2c that option 2 has smaller estimated test error, which gives option 2 a better approach than option 1. The validation set should be independent of the selection of feature, and therefore, using it to select our features ruins our model. So, in option 2, we do not use our validation set for our training model.

```r
# Part 3b
players_22 <- read_csv("players_22.csv")
```

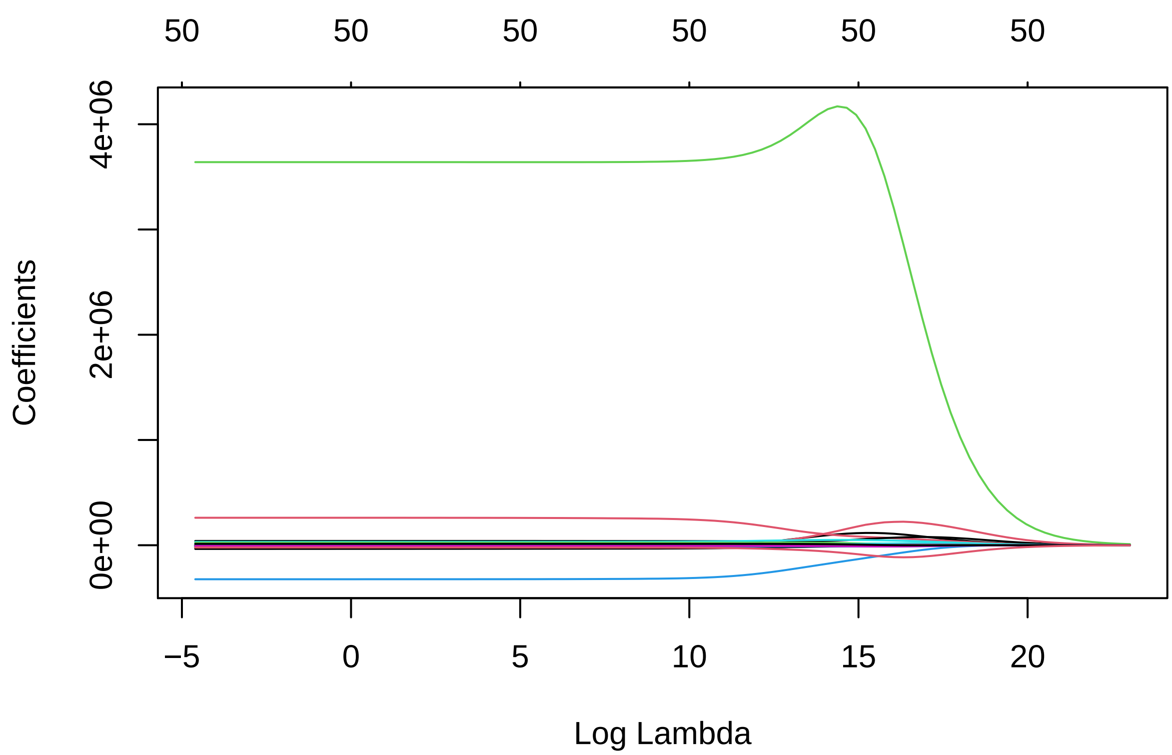
```
## Rows: 19238 Columns: 53
## -- Column specification --------------------------------------------------------
## Delimiter: ","
```

```
## chr  (2): short_name, long_name
## dbl (51): value_eur, potential, overall, wage_eur, age, height_cm, weight_kg...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

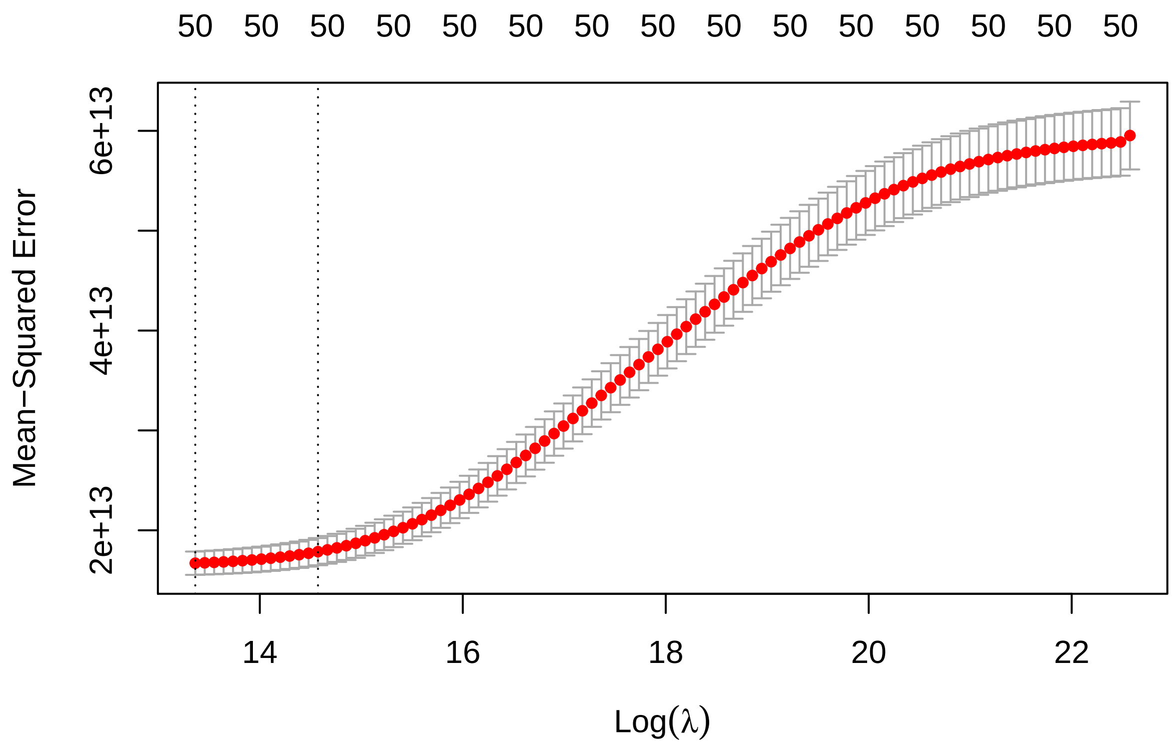
```r
players_22 <- na.omit(players_22)
y <- as.matrix(players_22[3])
x <- as.matrix(players_22[4:53])

train <- sample(17040, 8520)
validation_y <- y[-train]
train_y <- y[train]
train_x <- x[train, ]
validation_x <- x[-train, ]

fit_data <- lm(train_y~train_x)
values <- predict(fit_data, data.frame(validation_x))
(test_error4 <- mean((values - validation_y)^2))
```

```
## [1] 1.057054e+14
```

```r
# Part 3c
grid <- 10^seq(10, -2, length=100)
ridge.mod <- glmnet(x, y, alpha=0, lambda=grid)
plot(ridge.mod, xvar="lambda")
```



```r
# Part 3d
set.seed(1)
cv.out <- cv.glmnet(x, y, alpha=0)
plot(cv.out)
```



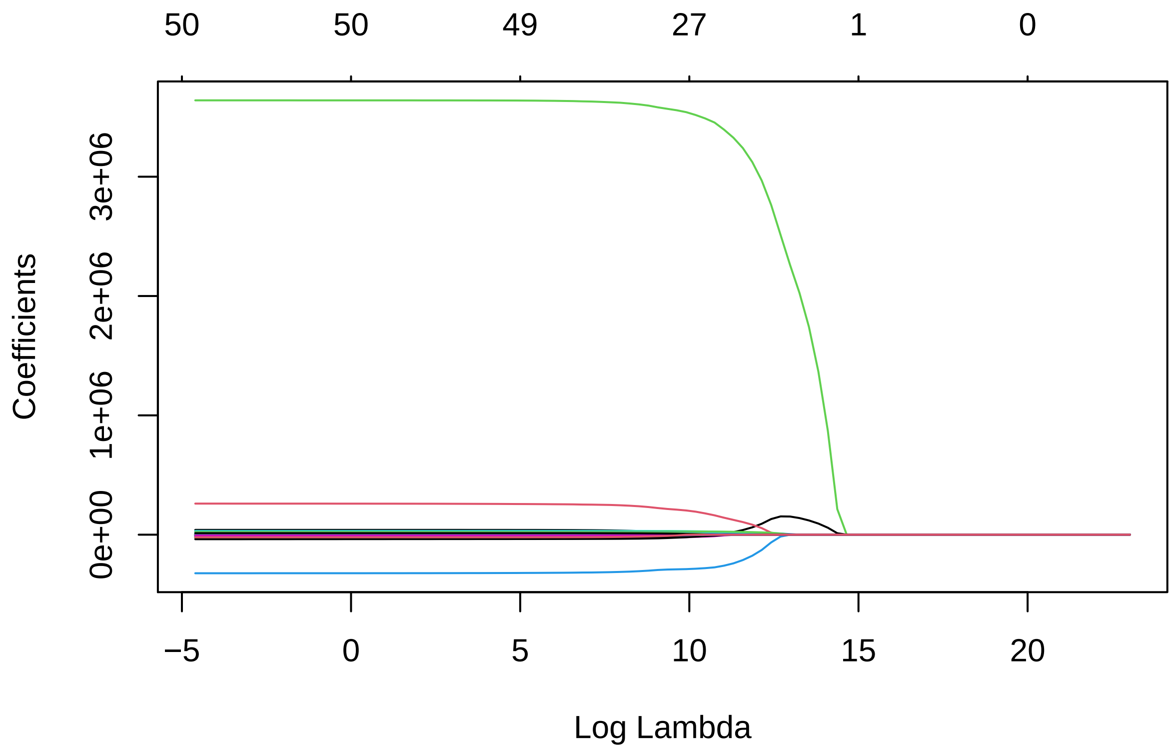

```r
(bestlam <- cv.out$lambda.min)
```

```
## [1] 636643.6
```

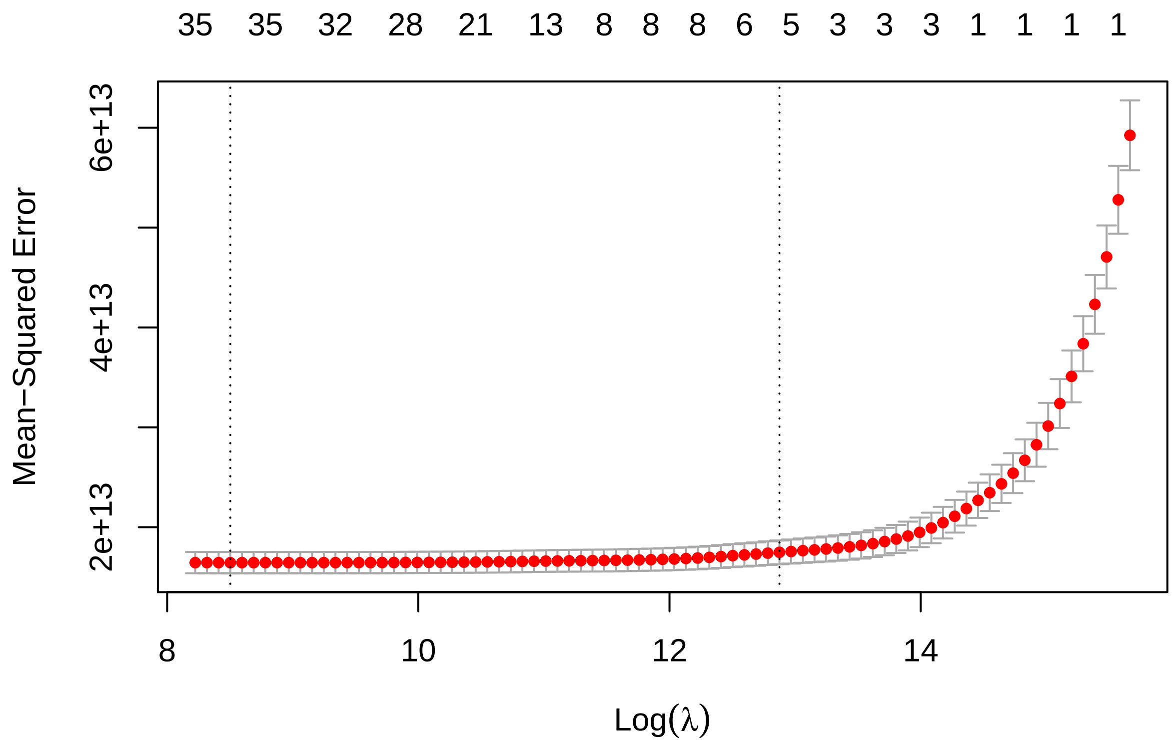
```r
min(cv.out$cvm)
```

```
## [1] 1.67049e+13
```

```r
# Part 3e
lasso.mod <- glmnet(x, y, alpha=1, lambda=grid)
plot(lasso.mod, xvar="lambda")
```




```r
# Part 3f
set.seed(1)
cv.out <- cv.glmnet(x, y, alpha=1)
plot(cv.out)
```




```r
(bestlam <- cv.out$lambda.min)
```

```
## [1] 4929.301
```

```r
min(cv.out$cvm)
```

```
## [1] 1.644897e+13
```

```r
coef(cv.out, s=bestlam)
```

```
## 51 x 1 sparse Matrix of class "dgCMatrix"
##                                    s1
## (Intercept)              -1.010488e+07
## potential                -1.102053e+04
## overall                   2.369276e+05
## wage_eur                  2.423559e+02
## age                      -3.059835e+05
## height_cm                 .
## weight_kg                -1.252694e+04
## weak_foot                 2.986567e+04
## skill_moves               .
## international_reputation  3.605657e+06
## pace                      1.432056e+04
## shooting                  .
```

```
## passing                          .
## dribbling                        .
## defending                        .
## physic                           .
## attacking_crossing          -9.010350e+03
## attacking_finishing          2.170369e+04
## attacking_heading_accuracy  -2.497318e+04
## attacking_short_passing          .
## attacking_volleys                .
## skill_dribbling             -2.521498e+04
## skill_curve                 -1.499645e+03
## skill_fk_accuracy            2.979962e+03
## skill_long_passing               .
## skill_ball_control          -3.272458e+04
## movement_acceleration        1.946115e+03
## movement_sprint_speed            .
## movement_agility            -9.910176e+03
## movement_reactions           3.153321e+04
## movement_balance             1.313725e+04
## power_shot_power            -1.957412e+04
## power_jumping               -6.132188e+03
## power_stamina                2.631103e+04
## power_strength               2.170723e+04
## power_long_shots            -2.341379e+03
## mentality_aggression        -4.798604e+03
## mentality_interceptions          .
## mentality_positioning        3.355849e+03
## mentality_vision             1.849199e+04
## mentality_penalties         -9.158218e+03
## mentality_composure         -2.151701e+03
## defending_marking_awareness      .
## defending_standing_tackle   -5.569143e+03
## defending_sliding_tackle         .
## goalkeeping_diving               .
## goalkeeping_handling        -6.983702e+03
## goalkeeping_kicking          7.331828e+03
## goalkeeping_positioning     -4.356605e+03
## goalkeeping_reflexes         1.107176e+04
## league_level                -1.681288e+04
```

3a) This dataset was found from Kaggle. This dataset is about the FIFA 2022 players where it shows all the stats of the players. The response that I chose is the value of the player in Euros, and the predictors are the other quantitative data that relates to each players, such as his/her dribbling skills, height, weight, wage, etc.

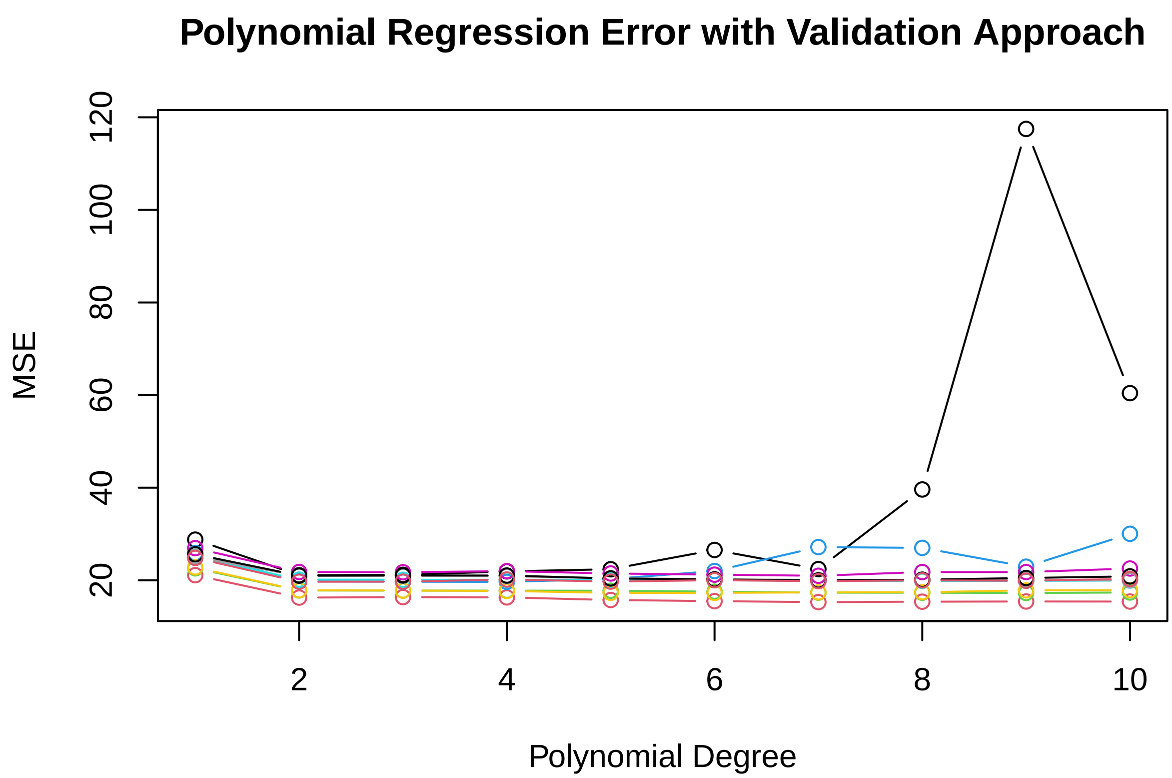
3b) We are using the validation set approach and then calculate the MSE on the validation set to estimate the test error. The test error is expected to be 1.057054e+14.

3d) We are using the 10-fold cross validation. Our value of lambda is 636643.6 and the expected test error is 1.67049e+13.

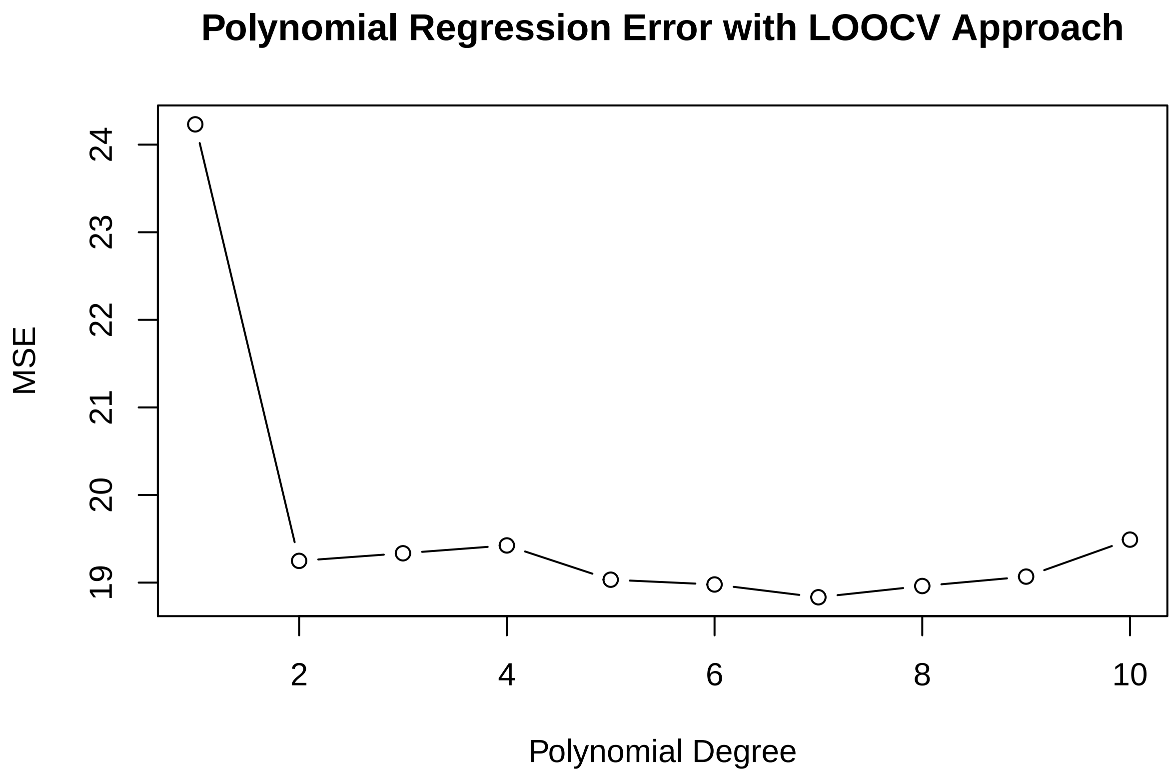
3e) We are using the 10-fold cross validation. Our value of lambda is 4929.301 and the expected test error is 1.644897e+13.

3f) The features that are not included are the height in cm, moves skill, shooting, passing, dribbling, defending, physic, attacking short passing, attacking volleys, skill long passing, movement sprint speed, mentality interceptions, defending sliding tackle and goalkeeping diving.

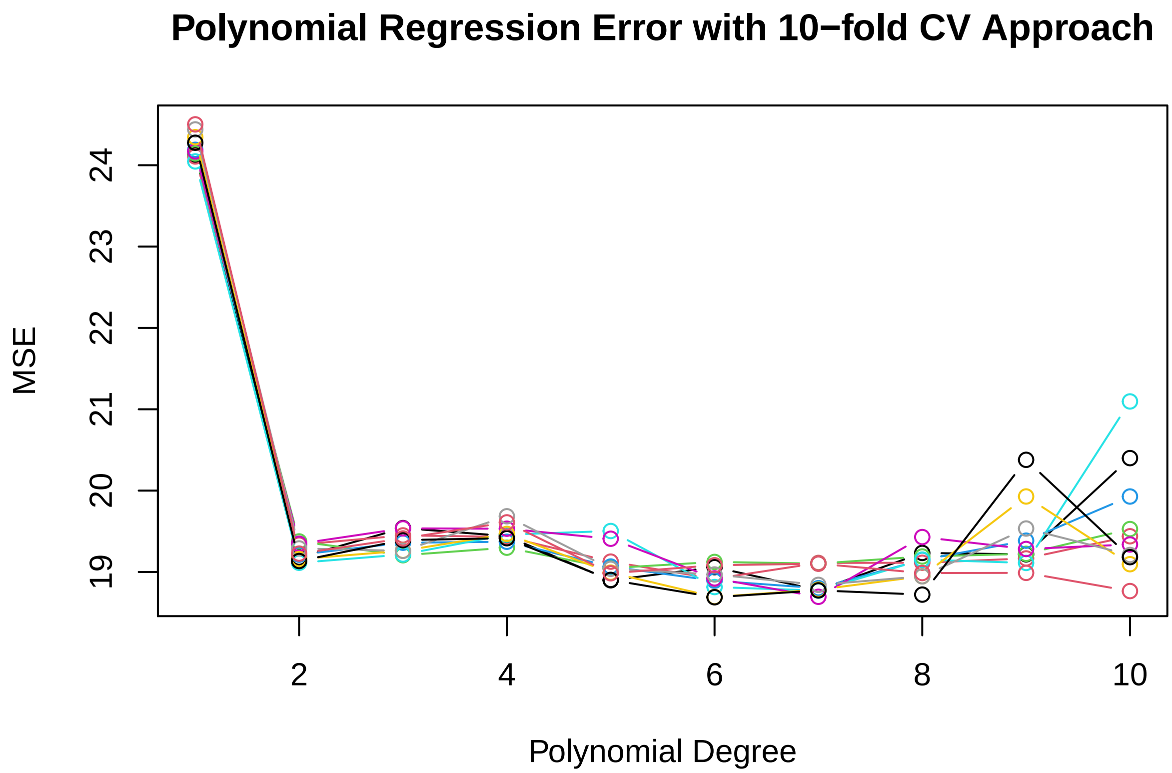
```r
# Part 4a
x <- Auto$horsepower
y <- Auto$mpg
n <- length(y)
mat <- data.frame(matrix(rep(0, 110), nrow=10))
colnames(mat)[11] <- "polydegree"

for(i in 1:10) {
  train <- sample(n, n/2)
  mat[i, 11] <- i
  for(j in 1:10) {
    model <- lm(y ~ poly(x, j), subset = train)
    pred <- predict(model, as.data.frame(y))
    mat[j,i] <- mean((pred-y)[-train]^2)
  }
}

plot(mat$polydegree, mat[,1], col=1, type="b",
  ylim=range(min(mat[,1:10]), max(mat)),
  xlab="Polynomial Degree", ylab="MSE",
  main="Polynomial Regression Error with Validation Approach")

for(i in 2:10) {
  points(mat$polydegree, mat[,i], col=i, type="b")
}
```



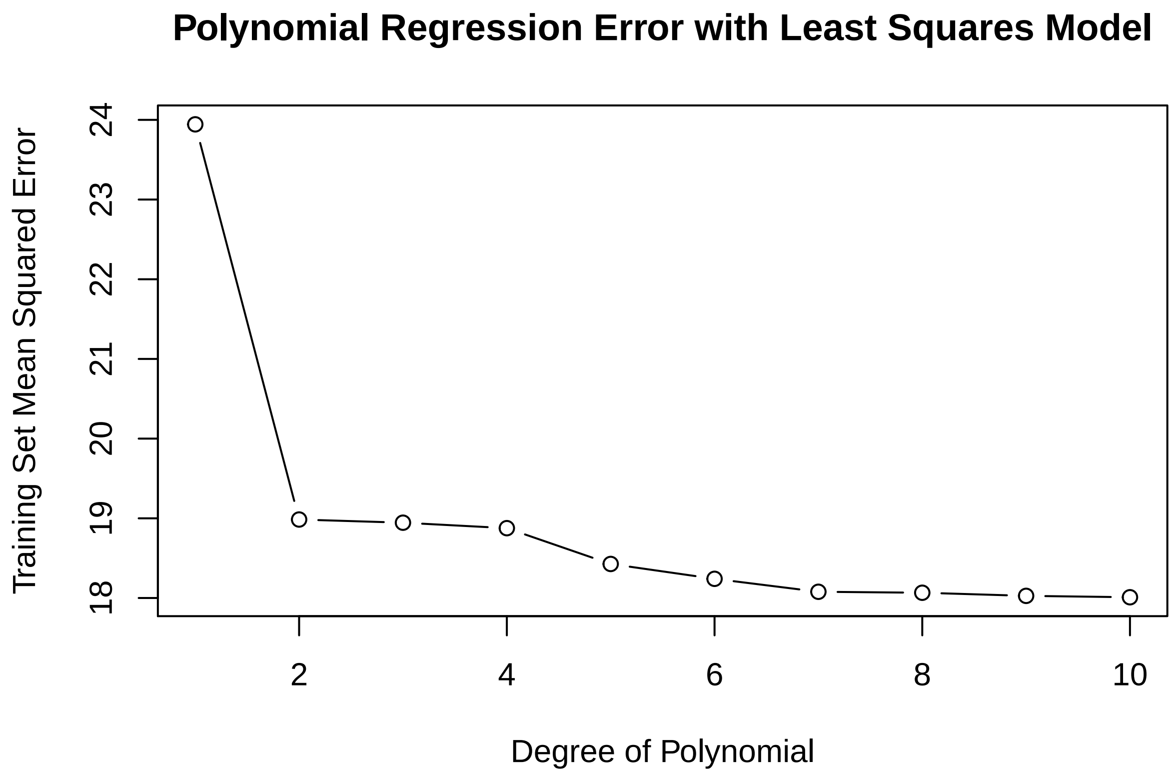

```r
# Part 4b
mat1 <- data.frame(matrix(rep(0, 20), nrow=10))
colnames(mat1)[2] <- "polydegree"

for (i in 1:10) {
  glm.fit <- glm(mpg ~ poly(horsepower, i), data = Auto)
  mat1[i, 1] <- cv.glm(Auto, glm.fit)$delta[1]
  mat1[i, 2] <- i
}

plot(mat1$polydegree, mat1[,1], col=1, type="b",
  xlab="Polynomial Degree", ylab="MSE",
  main="Polynomial Regression Error with LOOCV Approach")
```




```r
# Part 4c
mat2 <- data.frame(matrix(rep(0, 110), nrow=10))
colnames(mat2)[11] <- "polydegree"

for(i in 1:10) {
  mat2[i, 11] <- i
  for(j in 1:10) {
    glm.fit <- glm(mpg ~ poly(horsepower, j), data = Auto)
    mat2[j,i] <- cv.glm(Auto, glm.fit, K = 10)$delta[1]
  }
}

plot(mat2$polydegree, mat2[,1], col=1, type="b",
  ylim=range(min(mat2[,1:10]), max(mat2)),
  xlab="Polynomial Degree", ylab="MSE",
  main="Polynomial Regression Error with 10-fold CV Approach")

for(i in 2:10) {
  points(mat2$polydegree, mat2[,i], col=i, type="b")
}
```




```
# Part 4d
mat3 <- data.frame(matrix(rep(0, 20), nrow=10))
colnames(mat3)[2] <- "polydegree"

for (i in 1:10) {
  fit_lm <- lm(y ~ poly(x, i), data = Auto)
  value <- predict(fit_lm, as.data.frame(y))
  mat3[i, 2] <- i
  mat3[i, 1] <- mean((value-y)^2)
}

plot(mat3$polydegree, mat3[,1], col=1, type="b",
  xlab="Degree of Polynomial", ylab="Training Set Mean Squared Error",
  main="Polynomial Regression Error with Least Squares Model")
```




```r
# Part 4e
fit3 <- lm(y ~ poly(x, 10))
summary(fit3)
```

```
##
## Call:
## lm(formula = y ~ poly(x, 10))
##
## 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(x, 10)1   -120.1377     4.3046 -27.909   <2e-16 ***
## poly(x, 10)2     44.0895     4.3046  10.242   <2e-16 ***
## poly(x, 10)3     -3.9488     4.3046  -0.917   0.3595
## poly(x, 10)4     -5.1878     4.3046  -1.205   0.2289
## poly(x, 10)5     13.2722     4.3046   3.083   0.0022 **
## poly(x, 10)6     -8.5462     4.3046  -1.985   0.0478 *
## poly(x, 10)7      7.9806     4.3046   1.854   0.0645 .
## poly(x, 10)8      2.1727     4.3046   0.505   0.6140
## poly(x, 10)9     -3.9182     4.3046  -0.910   0.3633
## poly(x, 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
```

4a) We can see that there is a significant decrease from degree one to two. After that, some of the models increases but some continues to decrease until it reaches around a degree of 6 or 7. However, I would say that the best degree for most of these models is around 2 because of the significant decrease in MSE.

4b) As we can see from the graph, the lowest MSE is at degree of 7, and after that, it continues to increase. However, I still believe we should stick with a degree of 2 so that we will not overfit our data.

4c) We can see from the plot, the a degree of around 7 has the lowest MSE. However, there is a significant decrease on a degree of 2, and I still believe we should stick with a degree of 2 so that we will not overfit our data.

4d) We can see from the plot that as the degree of polynomial increases, the training error will get smaller and smaller because our model will better fit the points, which ended up creating a model that overfits but has small errors.

4e) As we can see the p values are really small, which indicates that it is unlikely we will observe a relationship between the predictor (horsepower) and the response (mpg) due to chance, which means that there is actually a relationship between the predictor and the response. As we can see, as the degree of polynomials get larger, the p-value also increases. This means that it is more likely that the relationship between the predictor and the response is by chance.

5a) $\sum_{i=1}^{n} (y_i - \beta x_i)^2 = \sum_{i=1}^{n} y_i^2 - 2\beta y_i x_i + \beta^2 x_i^2$

$$= \sum_{i=1}^{n} y_i^2 - 2\beta \sum_{i=1}^{n} y_i x_i + \beta^2 \sum_{i=1}^{n} x_i^2$$

$$\frac{\partial}{\partial \beta} \left(\sum_{i=1}^{n} y_i^2 - 2\beta \sum_{i=1}^{n} y_i x_i + \beta^2 \sum_{i=1}^{n} x_i^2 \right) = 0$$

$$-2\sum_{i=1}^{n} y_i x_i + 2\beta \sum_{i=1}^{n} x_i^2 = 0$$

$$2\beta \sum_{i=1}^{n} x_i^2 = 2\sum_{i=1}^{n} y_i x_i$$

$$\beta = \frac{\sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} x_i^2} \quad //$$

5b) $\sum_{i=1}^{n} (y_i - \beta x_i)^2 + \lambda\beta^2 = \sum_{i=1}^{n} y_i^2 - 2\beta \sum_{i=1}^{n} y_i x_i + \beta^2 \sum_{i=1}^{n} x_i^2 + \lambda\beta^2$

$$\frac{\partial}{\partial \beta} \left(\sum_{i=1}^{n} y_i^2 - 2\beta \sum_{i=1}^{n} y_i x_i + \beta^2 \sum_{i=1}^{n} x_i^2 + \lambda\beta^2 \right) = 0$$

$$-2\sum_{i=1}^{n} y_i x_i + 2\beta \sum_{i=1}^{n} x_i^2 + 2\beta\lambda = 0$$

$$\beta\left(\sum_{i=1}^{n} x_i^2 + \lambda \right) = \sum_{i=1}^{n} y_i x_i$$

$$\beta = \frac{\sum_{i=1}^{n} y_i x_i}{\lambda + \sum_{i=1}^{n} x_i^2} \quad //$$

5c) $Y = 3X + \varepsilon$

$y_i = 3x_i + \varepsilon_i$

$$E[\beta] = E\left[\frac{\sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} x_i^2} \right]$$

$$= E\left[\frac{\sum_{i=1}^{n} (3x_i + \varepsilon_i) x_i}{\sum_{i=1}^{n} x_i^2} \right]$$

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

$$= E\left[\frac{\sum_{i=1}^{n} 3x_i^2}{\sum_{i=1}^{n} x_i^2} \right] + E\left[\frac{\sum_{i=1}^{n} x_i \varepsilon_i}{\sum_{i=1}^{n} x_i^2} \right] = 3E\left[\frac{\sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n} x_i^2} \right] + E\left[\frac{1}{\sum_{i=1}^{n} x_i^2} \right] \sum_{j=1}^{n} E[x_j] \underset{0}{E[\varepsilon_j]}$$

$$= 3 \quad //$$

This is unbiased.

5d) $Y = 3X + \varepsilon$

$$E[\beta] = E\left[\frac{\sum_{i=1}^{n} y_i x_i}{\lambda + \sum_{i=1}^{n} x_i^2}\right]$$

$$= E\left[\frac{\sum_{i=1}^{n} (3x_i + \varepsilon_i) x_i}{\lambda + \sum_{i=1}^{n} x_i^2}\right]$$

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

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

$$= 3E\left[\frac{\sum_{i=1}^{n} x_i^2}{\lambda + \sum_{i=1}^{n} x_i^2}\right] + E\left[\frac{1}{\lambda + \sum_{i=1}^{n} x_i^2}\right] \sum_{j=1}^{n} E[x_j]\, \overset{0}{E[\varepsilon_j]}$$

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

This is biased because as λ goes up, β decreases. Therefore, the bias increases.

5e) $Y = 3X + \varepsilon$

$$Var(\beta) = Var\left(\frac{\sum_{i=1}^{n} y_i x_i}{\sum_{i=1}^{n} x_i^2}\right)$$

$$= Var\left(\frac{\sum_{i=1}^{n} (3x_i + \varepsilon_i) x_i}{\sum_{i=1}^{n} x_i^2}\right)$$

$$= Var\left(\frac{\sum_{i=1}^{n} 3x_i^2 + x_i \varepsilon_i}{\sum_{i=1}^{n} x_i^2}\right)$$

$$= Var\left(\underbrace{\frac{\sum_{i=1}^{n} 3x_i^2}{\sum_{i=1}^{n} x_i^2}}_{} + \frac{\sum_{i=1}^{n} x_i \varepsilon_i}{\sum_{i=1}^{n} x_i^2}\right) = Var\left(\frac{\sum_{i=1}^{n} x_i \varepsilon_i}{\sum_{i=1}^{n} x_i^2}\right)$$

since x is constant then, $Var(a + w) = Var(w)$

$$= \left(\frac{1}{\sum_{i=1}^{n} x_i^2}\right)^2 Var\left(\sum_{i=1}^{n} x_i \varepsilon_i\right)$$

$$= \frac{1}{\left(\sum_{i=1}^{n} x_i^2\right)^2} \cdot \left(x_1^2 Var(\varepsilon_1) + x_2^2 Var(\varepsilon_i) \ldots\right)$$

$$= \frac{1}{\left(\sum_{i=1}^{n} x_i^2\right)^2} \cdot \sum_{i=1}^{n} x_i^2\, Var(\varepsilon_i)$$

$$= \frac{1}{\left(\sum_{i=1}^{n} x_i^2\right)^2} \cdot \sigma^2 \left(\sum_{i=1}^{n} x_i^2\right) = \frac{\sigma^2}{\sum_{i=1}^{n} x_i^2} \quad //$$

5f) $$\mathrm{Var}(\beta) = \mathrm{Var}\left(\frac{\sum_{i=1}^{n} y_i x_i}{\lambda + \sum_{i=1}^{n} x_i^2}\right)$$

$$= \mathrm{Var}\left(\frac{\sum_{i=1}^{n} (3x_i + \varepsilon_i)x_i}{\lambda + \sum_{i=1}^{n} x_i^2}\right)$$

$$= \mathrm{Var}\left(\frac{\sum_{i=1}^{n} 3x_i^2 + x_i\varepsilon_i}{\lambda + \sum_{i=1}^{n} x_i^2}\right)$$

$$= \mathrm{Var}\left(\underbrace{\frac{\sum_{i=1}^{n} 3x_i^2}{\lambda + \sum_{i=1}^{n} x_i^2}}_{\text{since } x \text{ is constant then, } \mathrm{Var}(a+w) = \mathrm{Var}(w)} + \frac{\sum_{i=1}^{n} x_i\varepsilon_i}{\lambda + \sum_{i=1}^{n} x_i^2}\right)$$

$$= \mathrm{Var}\left(\frac{\sum_{i=1}^{n} x_i\varepsilon_i}{\lambda + \sum_{i=1}^{n} x_i^2}\right) = \frac{1}{\left(\lambda + \sum_{i=1}^{n} x_i^2\right)^2} \mathrm{Var}\left(\sum_{i=1}^{n} x_i\varepsilon_i\right)$$

$$= \frac{1}{\left(\lambda + \sum_{i=1}^{n} x_i^2\right)^2} \cdot \left(x_1^2 \mathrm{Var}(\varepsilon_1) + x_2^2 \mathrm{Var}(\varepsilon_2) \ldots\right)$$

$$= \frac{1}{\left(\lambda + \sum_{i=1}^{n} x_i^2\right)^2} \cdot \sum_{i=1}^{n} x_i^2 \, \sigma^2$$

$$= \frac{\sigma^2 \cdot \sum_{i=1}^{n} x_i^2}{\left(\lambda + \sum_{i=1}^{n} x_i^2\right)^2}$$

This means that as λ increases, $\mathrm{Var}(\beta)$ decreases.

5g) As we can see from part d), when we increase λ, we are increasing the bias of our model. However, in part f), as we increase λ, we are also decreasing the variance of β. In conclusion, increasing λ, allows us to increase the bias of our model and also decrease the variance. Therefore, we will be able to optimize our expected prediction error