James Yang

DATA 558

Homework #3

1. In this problem, we'll see a (very!!) simple simulated example where a least squares linear model is “too flexible”.
    a. Y = 2.307-1.695x1 + 4.272x2 + ε

(Intercept)	x1	x2	x3	x4
2.306505732	1.695296056	4.272167040	-2.729463188	0.191738690
x5	x6	x7	x8	x9
1.163680601	1.198196997	0.435457211	-1.644682703	-3.937359649
x10	x11	x12	x13	x14
-0.207480864	-2.376594443	-0.056167061	0.163069435	-2.264984679
x15	x16	x17	x18	x19
2.011100606	0.101943833	0.778552231	-0.254901914	-3.288098436
x20	x21	x22	x23	x24
-0.545638399	-3.783029595	0.344860887	-0.002848443	0.876689458
x25	x26	x27	x28	x29
-0.692520489	-3.018049979	-1.835271183	-1.427209621	2.699155478
x30	x31	x32	x33	x34
1.274103646	1.001134036	3.311894873	-0.921074319	2.172241679
x35	x36	x37	x38	x39
-0.017989285	0.379203043	-3.279170658	-0.710303300	0.606287987
x40	x41	x42	x43	x44
2.031515285	0.025238751	3.498058874	1.114141053	1.704281591
x45	x46	x47	x48	x49
-4.075822475	-0.917221018	2.787521000	3.287218175	-0.671398258
x50	x51	x52	x53	x54
0.244274207	-1.096655350	-1.715377190	0.041685861	-0.478248540
x55	x56	x57	x58	x59
4.881243203	-2.884971293	1.195074758	-0.570711390	2.839530066
x60	x61	x62	x63	x64
1.008976546	-0.287398063	-1.098899507	1.949370161	4.637028047
x65	x66	x67	x68	x69
4.351198008	0.518134695	-1.022685526	1.028914731	1.659871768
x70	x71	x72	x73	x74
-0.262161014	-0.710983470	-1.811265938	2.627507027	0.020545013
x75	x76	x77	x78	x79
0.511877548	0.929658687	0.776608636	2.177290676	-2.690797762
x80	x81	x82	x83	x84
1.176125648	-2.193357552	-1.767313551	0.715603510	0.303634622
x85	x86	x87	x88	x89
3.060074437	0.290342757	-0.087358453	0.524794582	3.017789481
x90	x91	x92	x93	x94
-0.176484467	-1.624221384	1.713890416	-0.637834555	0.840173900
x95	x96	x97	x98	x99
-1.466366015	1.627936205	-2.833944456	0.408007314	2.913748674

    b. ε = 0.395
    c.
        i. Bias: 0.0246
        ii. Variance = 0
        iii. EPE = Test MSE = E((y – f(x)^2) = 1.2363
        iv. EPE = Test MSE = E((testing)^2) / 20 = 0.86109
        v. The expected predicted error was higher than the validation set test error but they seem to be very similar to each other. Since the model predicts 0, it should be very similar for all values.
    d. EPE 1d = sum((testing_predicted – training)^2) = **64.13**

Code used:

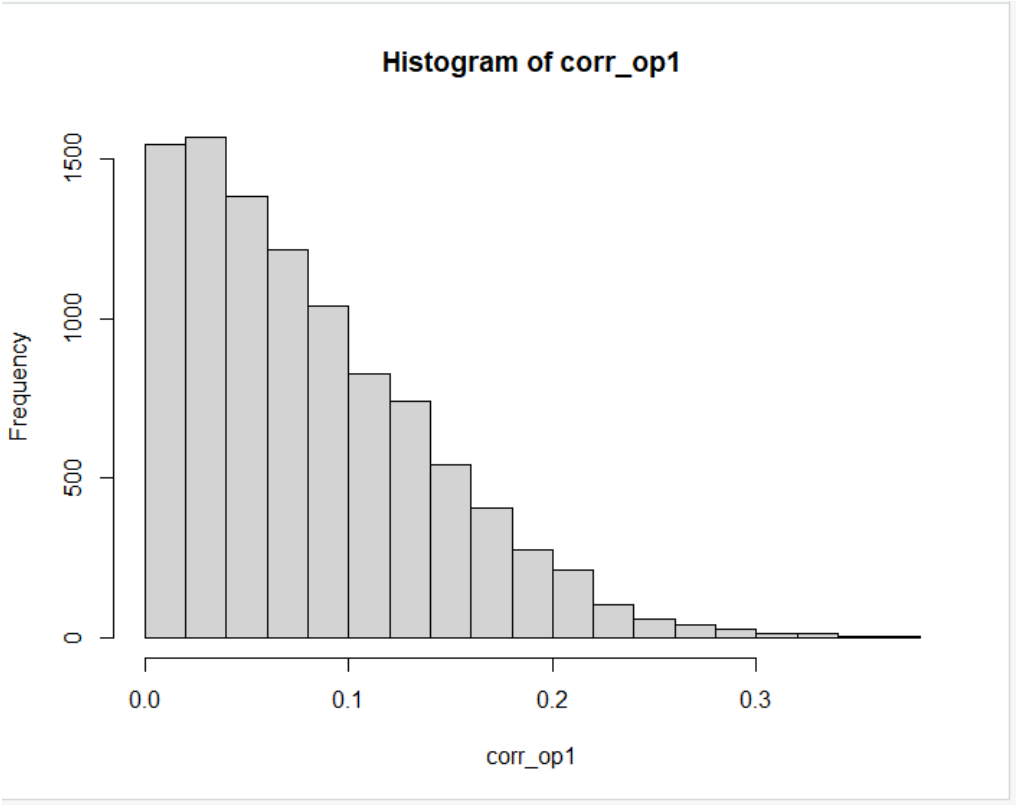
```
### Generate data ###
y <- rnorm(100)
x <- matrix(rnorm(10000*100), ncol = 10000)
error <- rnorm(100, mean = 0, sd = 1)
#####################

#####Predict Model with F(x) = 0 #####
model_1a <- lm(y~x)
zeros <- rep(0, 100)
# matrix_of_zeros <- data.frame(matrix(0, nrow = 1, 100))
# model_1d <- data.frame(predict(model_1a, newdata=matrix_of_zeros))

####Bias Function ####
get_bias = function(estimate, truth) {
  return (mean(estimate) - truth)

}
#### Get bias with estimate as all 0s and model predicts ####
#1c(i)
bias <- get_bias(zeros, y)
bias <- sum(bias)/100

#1c(ii)
variance <- var(y)

#1c (iii) Test MSE = E((y -f(x))^2)
EPE <- sum((y)^2)/100

# 1c(iv)
testing <- y[80:100]
EPE_validation <- sum(testing^2)/20

#1d
training <- y[0:80]
model_1d <- lm(y[0:80] ~ training)
testing_predicted <- predict(model_1d, newdata = data.frame(testing))
EPE_1d <- sum((testing_predicted - training)^2) / 20
```

e. The linear model had a higher variance and test error but a lower bias because of the overfitting in the model. This caused it to perform poorly when trying to predict new data points, but the model that predicted all 0's was performing well because the output was already generated with the intention of having an expected value of 0. This meant that predicting 0 when X and Y were uncorrelated performed better with a more optimal testing error.

2.

a.

	V1
8623	0.3831140
9536	0.3661025
7175	0.3547790
1972	0.3340687
5908	0.3295812
7654	0.3275334
619	0.3193427
4855	0.3187251
2101	0.3172948
3681	0.3167858
8272	0.3136242
8570	0.3135560
1411	0.3134681



b. Validation error 2b = 0.9298
c. Validation error 2c = 0.9336
d. The estimated error of 2b was lower than the estimated error of 2c because we were fitting data that had correlation/relation to each other. If we could imagine data points that were fitted closer together in a more predictable trend line, then it would give you a more properly fitted regression line that has fewer outlying

points. If we rerun this test multiple times, we can see that the validation (random correlation values) fluctuates a lot whereas the highest correlation values in 2c produce a more consistent error value. This explains why option 1 essentially gave us a meaningless estimate of the test error because there were always going to be random values that had little to no correlation with each other, thus making them almost completely random and misleading. Not only that, sometimes they can make the test error less than what its true value is because they might accidentally stumble in line with each other very closely by complete coincidence. Basically, they are phony points that disrupt any significance when it comes to the true error/model.

Using this code:

```
################## Q2

#2a
corr_op1 <- abs(cor(x, y))
hist(corr_op1)

#2b
model_2 <- lm(y[1:10]~training[1:10], drop.unused.levels = TRUE)
model_2option1 <- predict(model_2, newdata = data.frame(training[1:10]))
EPE_2b <- sum((model_2option1 - testing[1:10])^2)/10

#2c
indexIncreasing <- sort(corr_op1, decreasing = TRUE)
model_2c <- lm(y[1:10] ~ indexIncreasing[1:10], drop.unused.levels = TRUE)
model_2option2 <- predict(model_2c, newdata = data.frame(largest10Q))
EPE_2c <- sum((model_2option2 - testing[1:10])^2)/10
```

3. .

a. I got my dataset from UCI ML Repo (https://archive.ics.uci.edu/ml/datasets/Sports+articles+for+objectivity+analysis). It represents attributes from 1000 sports articles with their link, objectivity, word count, number of adverbs, number of nouns, etc. It essentially analyzes 59 of attributes regarding the articles and how they are structured. In my case, I'm trying to see the response of the total word count based on how many adverbs, nouns, commas, semicolons, and the 54 other attributes that put together a sports article. These predictors are all small pieces of what puts together a sports article. How many quotes it contains, numbers/statistics found, and many other features help put together predictors for how many words are found within the article. Words are conditionalized as a string with a space. For example, "1 2 hello" would be 3 words in the case of this dataset.

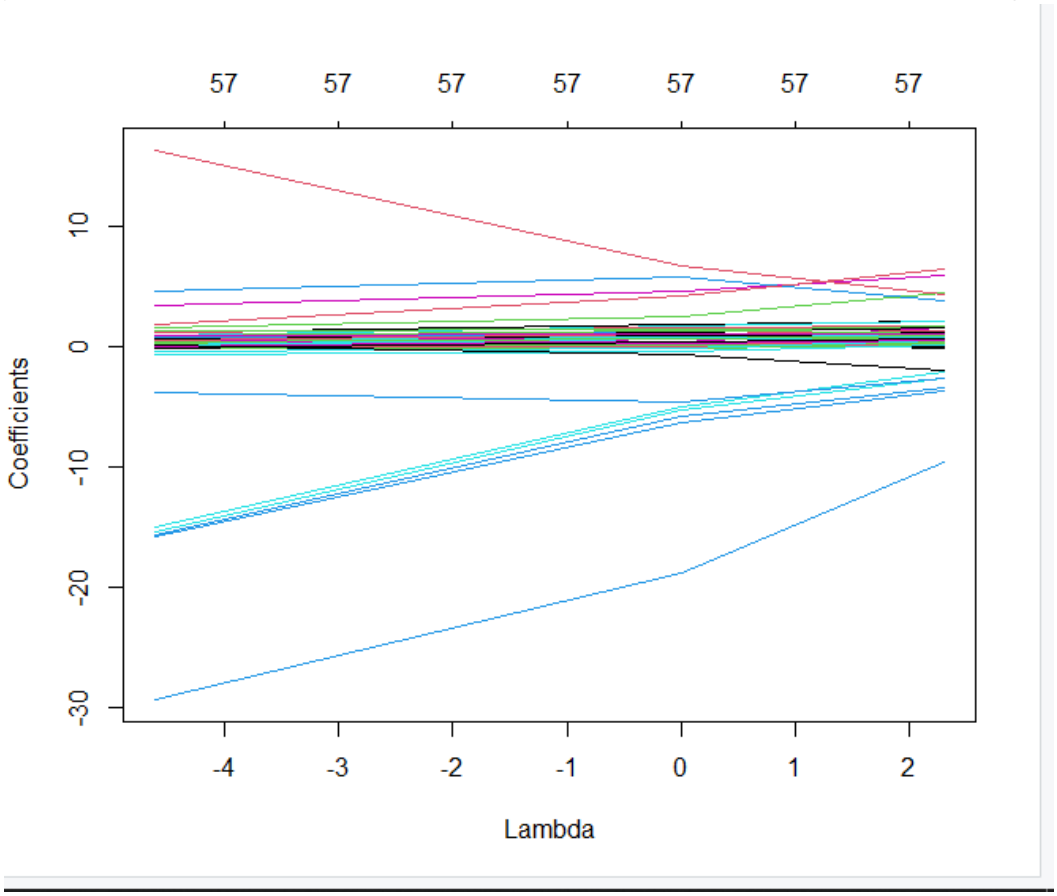
b. Using the same method as option 2 from the previous question, I split the observations into training and validation sets. Then on the training sets I found the correlations for the largest values for the correlation. I fit an L.S model to predict y using 59 features. My estimate came out to be **120.77** which means that the

dataset may contain data with notable irregularities. Furthermore if I reduced the number of features to 30, my error came out to be a very high **1207.72.**

```
################ Q3

#3b
df_3 <- read.csv("features.csv")
model_3b <- lm(df_3$totalWordsCount ~ ., data = df_3)
x <- df_3 %>% select(4:62)
correlations <- abs(cor(x, df_3$totalWordsCount))
index <- order(correlations, decreasing = TRUE)
largest <- c()
for (i in 1:300) {
  largest <- append(largest, x[index[i]])
}
training3b <- df_3$totalWordsCount[1:700]
testing3b <- df_3$totalWordsCount[700:1000]


model_3bLargest <- lm(training3b ~ ., data=largest, drop.unused.levels = TRUE)
model_3boption2 <- predict(model_3bLargest, newdata = data.frame(largest))
EPE_3b <- sum((model_3boption2[700:1000] - testing3b)^2)/300
```

c. Using this code:

```
#3c
library(glmnet)
lambdas = c(0.01, 1, 10)
ridge3c <- glmnet(data.matrix(x), df_3$totalWordsCount, alpha = 0, lambda = lambdas)
plot(ridge3c, xvar = "lambda", xlab="Lambda")
```



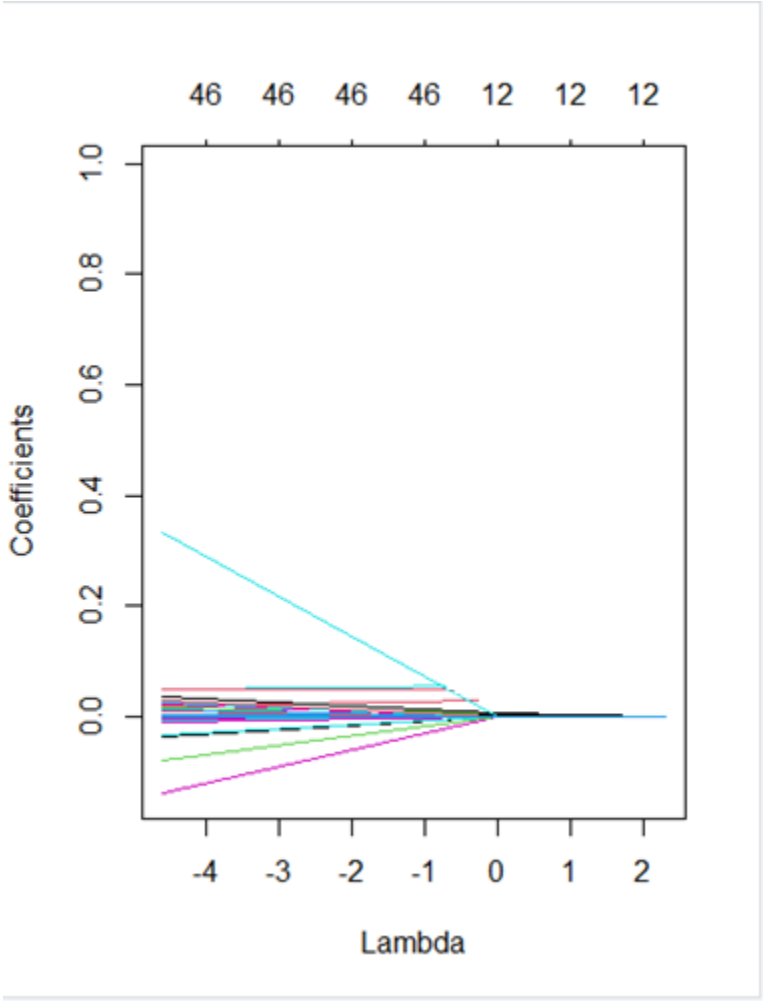
d. Using this code:

```
#3d
ridge3c <- cv.glmnet(data.matrix(x), df_3$totalWordsCount, alpha = 0)
best_lambda <- ridge3c$lambda.min
ridge3d <- glmnet(data.matrix(x), df_3$totalWordsCount, alpha = 0, lambda = 52.307)
model_3d <- predict(ridge3d, newdata = data.frame(training), newx = data.matrix(x))
EPE_3d <- sum((model_3d[700:1000]-testing3b)^2)/300
```

The best lambda calculated was **52.308.**

The test error for ridge was **349.450.** I estimated the error the same way I have been doing in previous sections. I first created a ridge model using all the features and the total words count as the y and setting the lambda as 52.308. Then I predicted the ridge values on training data and calculated the testing error by summing the predicted – testing (same values from 3b) and squared them then divided by 300.

e. Using practically the same code as part c and d, except changing alpha = 1 we get these plots.



f. The best lambda calculated for Lasso was **15.248** and the test error received was **310.957**. I estimated the test error the same as I did in part d. I created a Lasso model which is the same as the Ridge model except with alpha = 1. Then I predicted the lasso based on training data and then calculated the error by

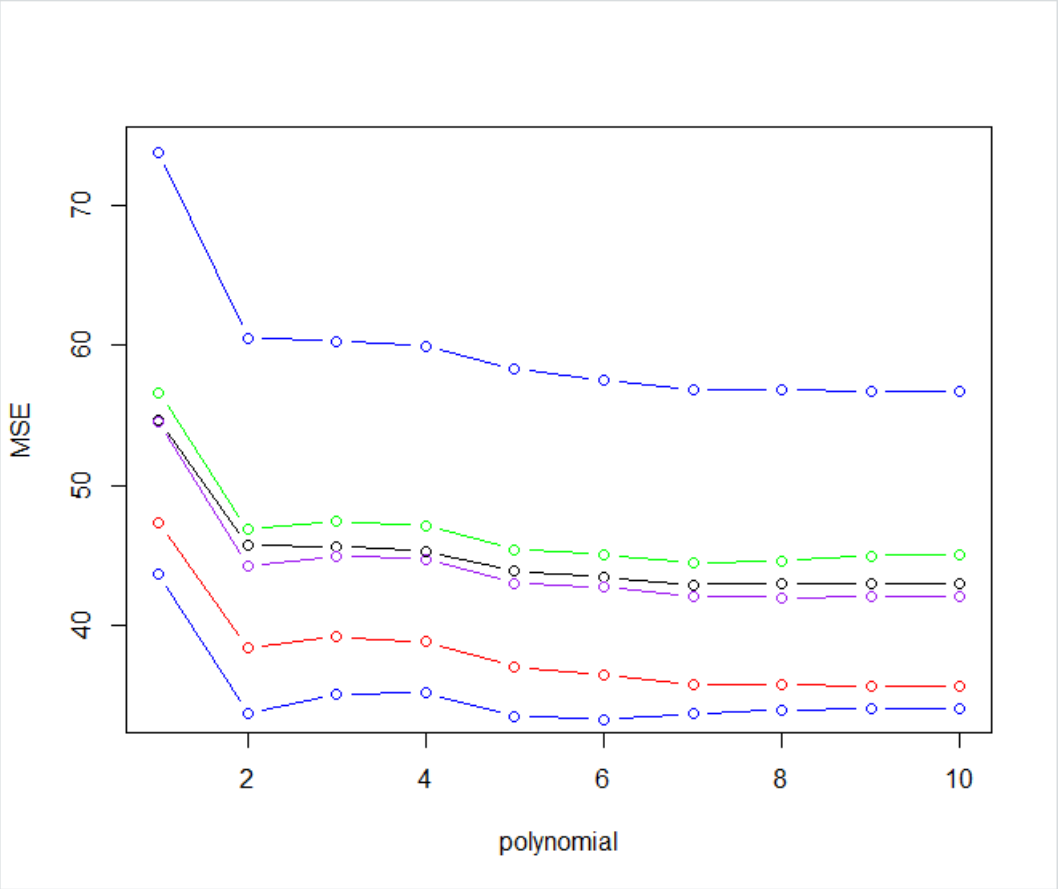
summing the predicted model minus the testing model squared, and dividing by the number of elements 300.

```
#3e
lambdas = c(0.01, 1, 10)
lasso3e <- glmnet(data.matrix(x), df_3$totalWordsCount, alpha = 1, lambda = lambdas)
plot(lasso3e, xvar = "lambda", xlab="Lambda")

#3f

lasso3f <- cv.glmnet(data.matrix(x), df_3$totalWordsCount, alpha = 1)
best_lambda_3f <- lasso3f$lambda.min
lasso3f <- glmnet(data.matrix(x), df_3$totalWordsCount, alpha = 1, lambda = best_lambda
model_3f <- predict(ridge3f, newdata = data.frame(training), newx = data.matrix(x))
EPE_3f <- sum((model_3f[700:1000] - testing3b)^2)/300
```

4.

a. Using this code:

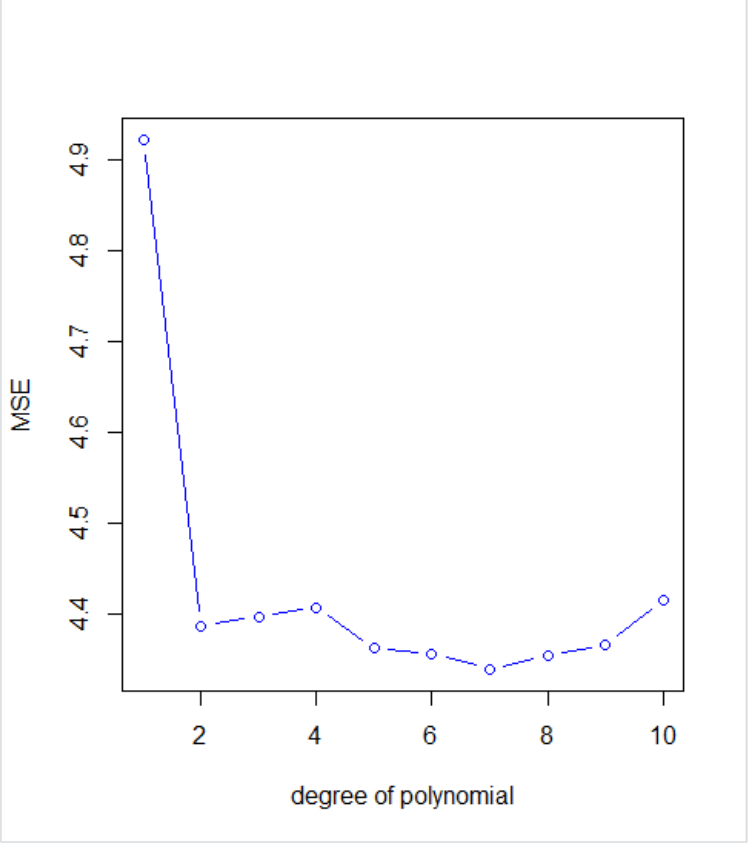
```
generate_errors <- function(n) {
  errors <- c()
  for(i in 1:10){
    training_auto <- Auto[1:n,]
    validation_auto <- Auto[(n+1):392,]
    model_4a <- lm(training_auto$mpg ~ poly(horsepower, i), data = training_auto)
    model_4aPredict <- predict(model_4a, newdata = data.frame(validation_auto))
    MSE_4a <- sum((validation_auto$mpg - model_4aPredict)^2)/length(model_4aPredict)
    errors <- append(errors, MSE_4a)
  }
  return (errors)
}

error1 <- generate_errors(100)
error2 <- generate_errors(150)
error3 <- generate_errors(200)
error4 <- generate_errors(250)
error5 <- generate_errors(175)
error6 <- generate_errors(300)


plot(1:10, errors, type = "b", ylab = "MSE", xlab = "polynomial", ylim = c(34, 74))
points(error1, col = "blue", type ="b")
points(error2, col = "red", type ="b")
points(error3, col = "green", type ="b")
points(error4, col = "black", type ="b")
points(error5, col = "purple", type ="b")
points(error6, col = "blue", type ="b")
```



After running this procedure a few times I have noticed that the best polynomial finding is approximately around 6. What this means is that the prediction that fits the regression best is not a simple linear model but rather a model that fits well with a 6^{th} degree polynomial. The reason this may be the case is because the data is more complex and has a more curvy trajectory.

b. Using this code:

```
lasso_errors <- c()

library(caret)
ctrl <- trainControl(method = "LOOCV")
model_4b <- train(mpg ~ poly(horsepower,1), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])

model_4b <- train(mpg ~ poly(horsepower,2), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])

model_4b <- train(mpg ~ poly(horsepower,3), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])

model_4b <- train(mpg ~ poly(horsepower,4), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])

model_4b <- train(mpg ~ poly(horsepower,5), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])

model_4b <- train(mpg ~ poly(horsepower,6), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])

model_4b <- train(mpg ~ poly(horsepower,7), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])

model_4b <- train(mpg ~ poly(horsepower,8), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])

model_4b <- train(mpg ~ poly(horsepower,9), data = Auto, method = "lm", trControl = ctrl)
lasso_errors <- append(lasso_errors, model_4b$results[2])
```

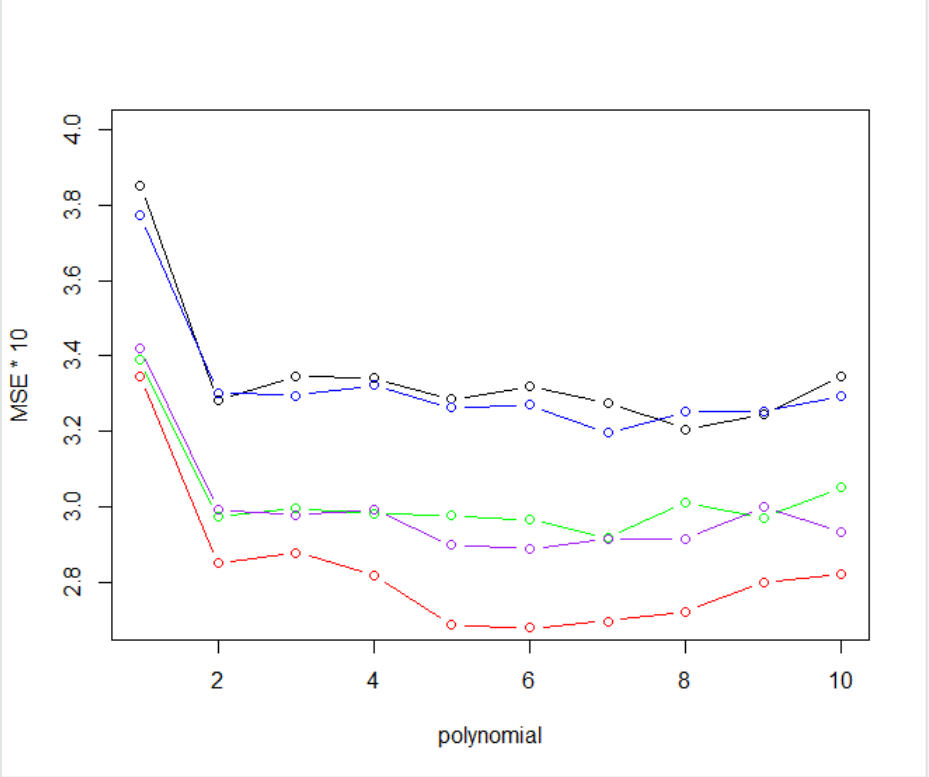


I was able to produce this graph that looks very similar to the one found in figure 5.4. It appears that the best degree of polynomial is 7 which correlates well with the previous question using the validation set approach because they agree with each other. They are both saying that the model is complex and has a curvy

trajectory. This model would not fit well with something as straight as a linear regression.

c. After using the caret library to generate 10-fold CVs, it produced a plot like this. Using this code:

```
generateKfold_errors <- function(n) {
  K_errors <- c()
  ctrl <- trainControl(method = "cv")
  training_auto <- Auto[1:n,]
  validation_auto <- Auto[(n+1):392,]
  model_4c <- train(mpg ~ poly(horsepower,1), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,2), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,3), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,4), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,5), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,6), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,7), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,8), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,9), data = training_auto, method = "lm", trControl = ctrl)
  K_errors <- append(K_errors, model_4c$results[2])

  model_4c <- train(mpg ~ poly(horsepower,10), data = training_auto, method = "lm", trControl = ctrl)
  lasso_errors <- append(lasso_errors, model_4c$results[2])

  return (lasso_errors)
}

error1 <- generateKfold_errors(100)
error2 <- generateKfold_errors(150)
error3 <- generateKfold_errors(200)
error4 <- generateKfold_errors(250)
error5 <- generateKfold_errors(175)
error6 <- generateKfold_errors(300)



plot(1:10, error1, type = "b", ylab = "MSE * 10", xlab = "polynomial")
points(1:10, error2, col = "red", type ="b")
points(1:10,error3, col = "green", type ="b")
points(1:10,error4, col = "black", type ="b")
points(1:10,error5, col = "purple", type ="b")
points(1:10,error6, col = "blue", type ="b")
```

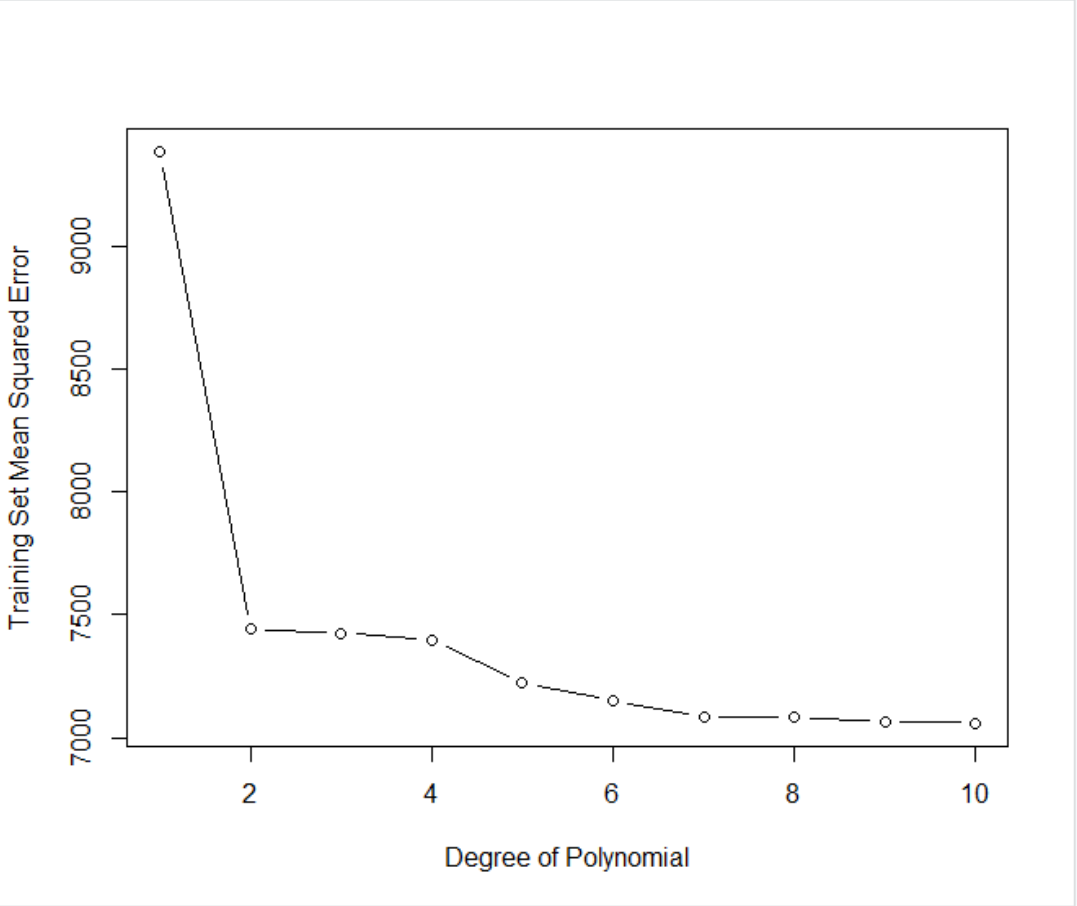


The best degree polynomial it found to be was approximately 5. It aligns fairly well with the validation set and leave one out methods because it the dataset can not be fitted with a linear regression, so it needs more curvature in its regression. In this case it predicted that it only needed a 5th polynomial as opposed to the 6th and 7th polynomial predicted in the previous sections.

d. Using this code: (Code was too long to screenshot)

```
#4d
error_4d <- c()
for(i in 1:10) {
  model_4d <- lm(mpg ~ poly(Auto$horsepower, i), data=Auto)
  predict_4d <- predict(model_4d, newdata = data.frame(Auto$horsepower))
  error_4d <- append(error_4d, sum(model_4d$residuals^2))
}
plot(1:10, error_4d, type = "b", xlab = "Degree of Polynomial", ylab = "Training Set Mean Squared Error")
```

I was able to produce this plot:



These values look awfully like both the textbook and what I have found in the previous sections (b and c). It makes sense that this linear regression has errors up to 4000+ though because as exhibited from the previous parts of the question, it needs to fit a curvature model with exponents up to 5-7 to reduce the error of the model.

e. Using the 10^{th} polynomial,

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

It will give you a very long report about each feature and their estimate, standard error, t value and p value. Something to note here is that the p-value is somewhat high. As we go higher in polynomial, we can see that 8-10 produce somewhat high p-values. This would indicate that they aren’t significant in the dataset in predicting MPG. It makes sense when looking at more of the features too because features at higher polynomial values are expected to be less statistically significant because they need to be more specific to the model.