Kunal Mehta
50352960
kunalkam

Homework 3

Problem 1

a. This problem has 5 main parts:
   - OLS estimates
   - Forwards selection
   - Backwards selection
   - LASSO regression
   - Ridge Regression

b. The first step was to import the required data into the environment, which I did by using the read.delim() function. A brief overview of the data:

```
> names(train)
 [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
[12] "V12" "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22"
[23] "V23" "V24" "V25" "V26" "V27" "V28" "V29" "V30" "V31" "V32" "V33"
[34] "V34" "V35" "V36" "V37" "V38" "V39" "V40" "V41" "V42" "V43" "V44"
[45] "V45" "V46" "V47" "V48" "V49" "V50" "V51" "V52" "V53" "V54" "V55"
[56] "V56" "V57" "V58" "V59" "V60" "V61" "V62" "V63" "V64" "V65" "V66"
[67] "V67" "V68" "V69" "V70" "V71" "V72" "V73" "V74" "V75" "V76" "V77"
[78] "V78" "V79" "V80" "V81" "V82" "V83" "V84" "V85" "V86"
```

```
> names(test)
 [1] "V1"  "V2"  "V3"  "V4"  "V5"  "V6"  "V7"  "V8"  "V9"  "V10" "V11"
[12] "V12" "V13" "V14" "V15" "V16" "V17" "V18" "V19" "V20" "V21" "V22"
[23] "V23" "V24" "V25" "V26" "V27" "V28" "V29" "V30" "V31" "V32" "V33"
[34] "V34" "V35" "V36" "V37" "V38" "V39" "V40" "V41" "V42" "V43" "V44"
[45] "V45" "V46" "V47" "V48" "V49" "V50" "V51" "V52" "V53" "V54" "V55"
[56] "V56" "V57" "V58" "V59" "V60" "V61" "V62" "V63" "V64" "V65" "V66"
[67] "V67" "V68" "V69" "V70" "V71" "V72" "V73" "V74" "V75" "V76" "V77"
[78] "V78" "V79" "V80" "V81" "V82" "V83" "V84" "V85"
```

As can be observed, V86 is missing from the test data, an indication that this last column is the one that indicates whether or not a caravan insurance policy was bought. A further examination of this column reveals:

```
> table(train[,86])

   0    1
5474  348
```

There are very sparse instances of a caravan policy being bought (indicated by 1), only an approximate 6.35% of the cases to be precise. This is not something that I initially explored, but the importance of this observation became more important as I progressed with the problem. I'll explain as I come to it.

c. The next step was to check the OLS estimates. This was simple enough, the results of which are displayed below:

```
Residual standard error: 0.23 on 5736 degrees of freedom
Multiple R-squared:  0.0729,    Adjusted R-squared:  0.05916
F-statistic: 5.306 on 85 and 5736 DF,  p-value: < 2.2e-16
```

The summaries of the predictions:

Kunal Mehta
50352960
kunalkam

```
> pred <- predict(fit, test)
> summary(pred)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.12796  0.01631  0.05435  0.05929  0.09379  0.80824
> predtrain <- predict(fit, train)
> summary(predtrain)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-0.17130  0.01668  0.05396  0.05977  0.09501  0.83797
>
```

d. The more insightful part of these estimates lies in their accuracy, which I calculated both for the training and testing dataset. This was done simply by rounding off the prediction values so they are either 1 or 0 and comparing them to the true values. The training and testing accuracies are:

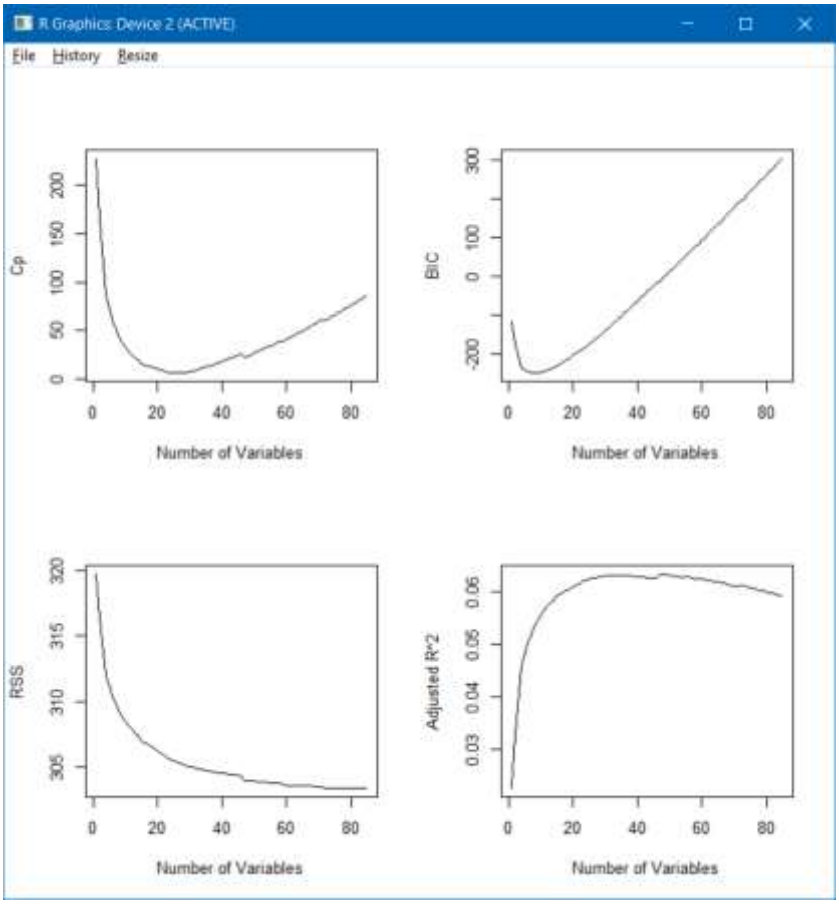
```
> accuracyOLStrain
[1] 94.03985
> accuracyOLStest
[1] 94.025
>
```

OLS Training accuracy: 94.03985%
OLS Testing accuracy: 94.025%

e. The next step in the problem was to perform forward subset selection and calculate the accuracy of this method. For this I used the regsubsets() function with the nvmax parameter set to 85 so as to not limit the maximum size of subsets. A small snip of the result of this selection:

The variation of cp, bic, rss and adjusted r^2 with the number of variables is shown in the graphs below:

Kunal Mehta
50352960
kunalkam



The subsets that show minimum cp and bic are:

```
> which(fwdsumm$cp == min(fwdsumm$cp))
[1] 23
> which(fwdsumm$bic == min(fwdsumm$bic))
[1] 8
```

I decided to compare the accuracies of all the different subsets and pick the one with the best testing accuracy as the one to compare other methods with. Therefore for all the values from 1 to 85, I calculated the training and testing errors for the corresponding subsets. This was done by multiplying the data matrices into the coefficients of that particular subset size.
At the end of this process, the subset with the minimum error was:

```
> which(store_error_test == min(store_error_test))
[1] 9
>
```

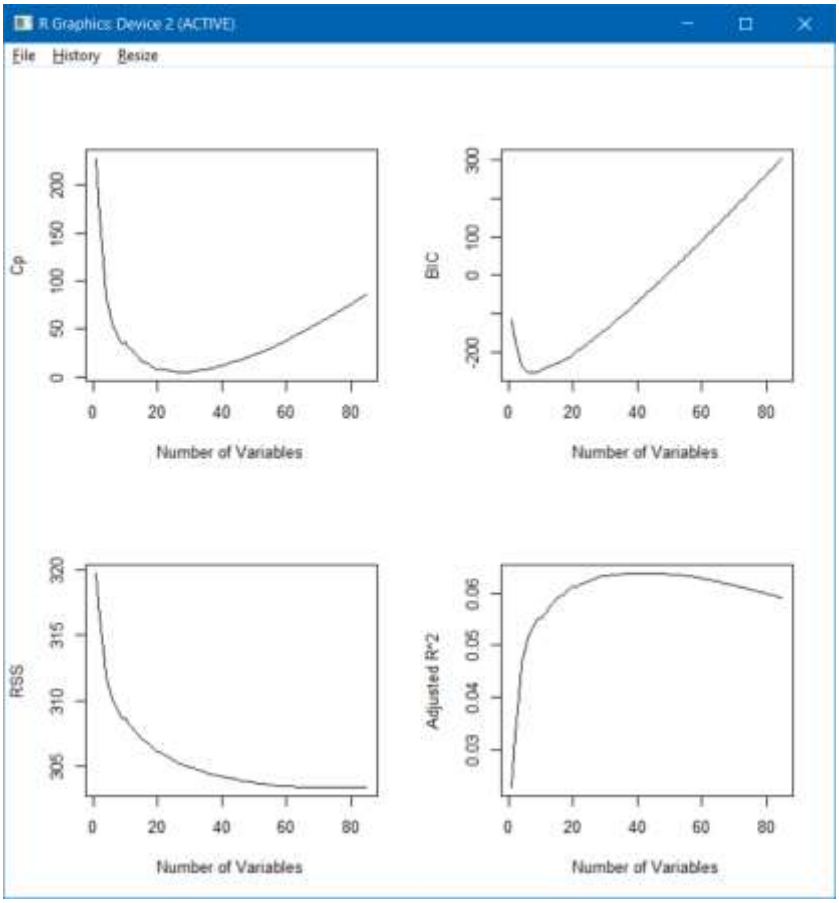
Thereby, I accessed the train and test errors corresponding to 9, and calculated the accuracy by subtracting them from 1 and converting them into a percentage. The accuracies for forward subset selection:

Kunal Mehta
50352960
kunalkam

```
> accuracyfwdtrain
[1] 94.66028
> accuracyfwdtest
[1] 94.56521
>
```

FSS Training Accuracy: 94.66028%
FSS Testing Accuracy: 94.56521%

f. In much the same way, I performed backward subset selection on the data and calculated the accuracies. The results are shown below:



These graphs are nearly identical to those of the forward selection method. The subsets that show minimum cp and bic values are:

```
> which(bwdsumm$cp == min(bwdsumm$cp))
[1] 29
> which(bwdsumm$bic == min(bwdsumm$bic))
[1] 8
```

Subset with minimum test error:

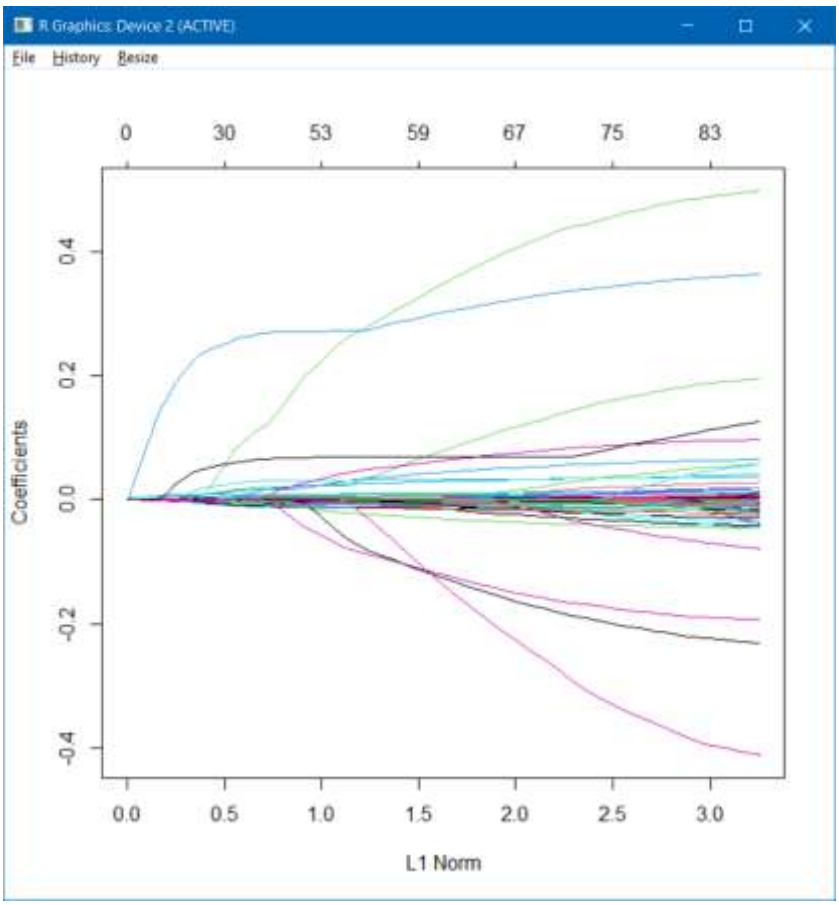
```
> which(store_error_test == min(store_error_test))
[1] 23
```

The training and testing accuracies corresponding to 23:

```
> accuracybwdtrain
[1] 94.73423
> accuracybwdtest
[1] 94.60059
> |
```

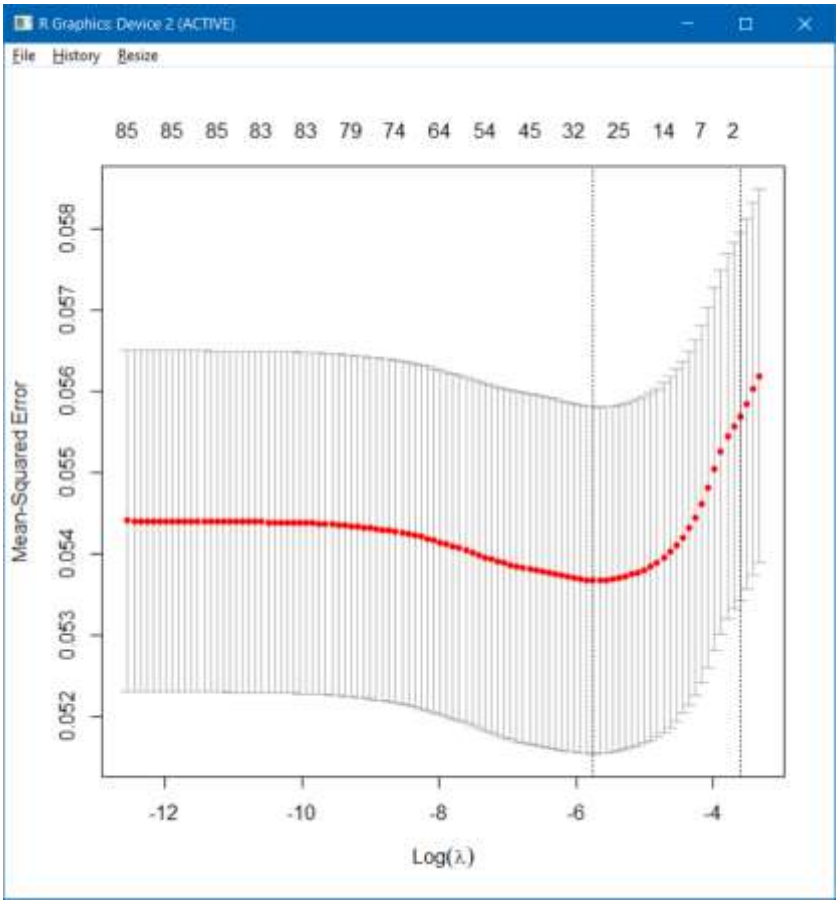
BSS Training Accuracy: 94.73423%
BSS Testing Accuracy: 94.60059%

g. The next step was to perform lasso method on the data. For this, I converted the training data to a matrix and removed the V86 values from it. This is essential for the use of the glmnet() function, to which I passed the alpha value of 1. The plot of this lasso model is shown below:



To calculate the best λ, I used cv.glmnet() function with the same parameters, and the graph below shows the MSE values with increasing λ:

Kunal Mehta
50352960
kunalkam



The best value of λ: 0.003184799

```
> lambest <- cv.out$lambda.min
> lambest
[1] 0.003184799
```

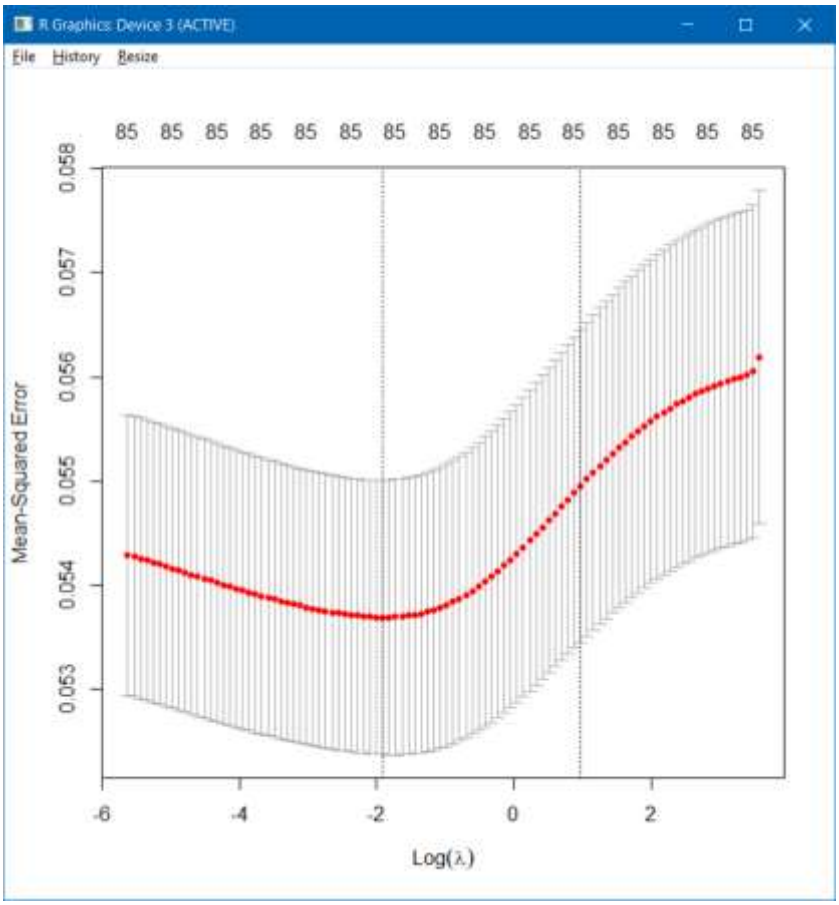
The next step was to predict the values of the response based on the lasso model we made earlier, and this was done by using the predict() function with the s value set to the best value of λ. I also rounded the prediction values so as to be comparable to the true values of the training and testing data. The error values were calculated in the same manner as the previous methods, number of wrong classifications divided by total number of classifications.
The results of the same are:

```
> accuracylassotrain
[1] 94.02267
> accuracylassotest
[1] 94.025
>
```

LASSO Training Accuracy: 94.02267%
LASSO Testing Accuracy: 94.025%

Kunal Mehta
50352960
kunalkam

h. The last step was to perform ridge regression on the data. For this step, I passed the same matrix form of the training data as in LASSO method, except that this time the alpha parameter was passed a value of 0. To find the best λ as well, the same method was followed as in LASSO, cv.glmnet() function. The variation of MSE with increase in λ:



The best value of λ: 0.1478253

```
> lambest <- cv.out$lambda.min
> lambest
[1] 0.1478253
>
```

The training and testing accuracy for ridge regression with the best value of λ:

```
> accuracyridgetest <- 100 - err
> accuracyridgetrain
[1] 94.03985
> accuracyridgetest
[1] 94.05
>
```

Ridge Training Accuracy: 94.03985%
Ridge Testing Accuracy: 94.05%

i. A comparison of the Training and Testing Accuracies of the different methods:

Kunal Mehta
50352960
kunalkam

Method	Testing Accuracy	Testing 1s	Training Accuracy	Training 1s
OLS	94.025%	3	94.03985%	3
Forward	94.56521%	3	94.66028%	2
Backward	94.60059%	3	94.73423%	2
LASSO	94.025%	3	94.02267%	2
Ridge	94.05%	0	94.03985%	1

From these results, it is easy to be misled into believing that all of these methods are very good at classifying. However, the fact that there is just about 6.35% of the observations have a positive result in terms of the caravan insurance policy being bought is the reason for such a high accuracy. Most of the predictions are 0 as well due to this and thus the accuracy is high.
Now to get a better picture, I have listed the number of 1s reported in the predictions for each method. The total number of true 1s in the training and testing data are given below:
True Training 1s: 348
True Testing 1s: 238
Therefore we see that an incredibly small number of true 1s are being predicted as such in each method. Therefore I do not believe that any of these methods is really capable of predicting whether or not a caravan insurance policy will be purchased due to the nature of the data that we have.

Problem 2

a. As per the question, I have set the p and n values as 20 and 1000. To form the data X matrix, I have used the rnorm function as suggested in the question. For ß as well, I used the rnorm function, and to make sure that some of these are 0, I forcibly set 5 random ß values as 0 to make sure that the requirements of the question are followed. The ß values are:

```
> B
 [1] -0.528173317  1.250119626  0.000000000  0.000000000 -0.074730276
 [6]  1.862882417 -0.003903067  0.245684654  0.000000000  0.000000000
[11]  0.912607396  0.490218589 -2.625629481 -0.640989889  1.768741130
[16]  1.608662378  0.421746606  0.791765998  0.000000000  0.608962814
>
```

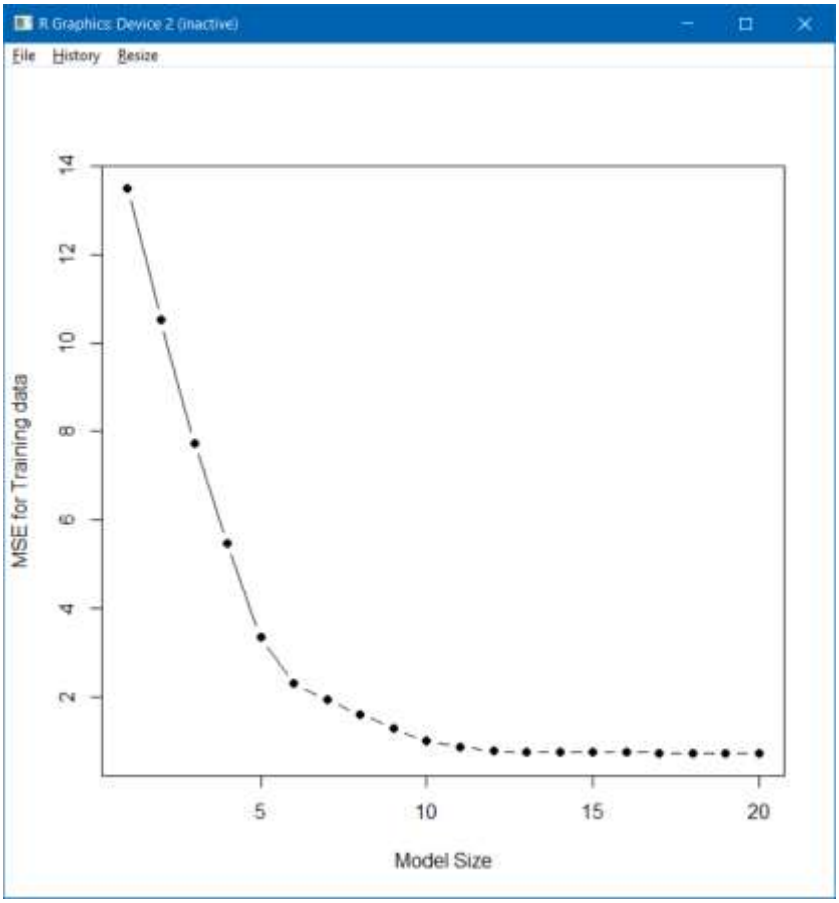
The y was calculated as $Y = X*ß + \epsilon$.

b. The next step was to compute forward subset selection. This was straightforward.

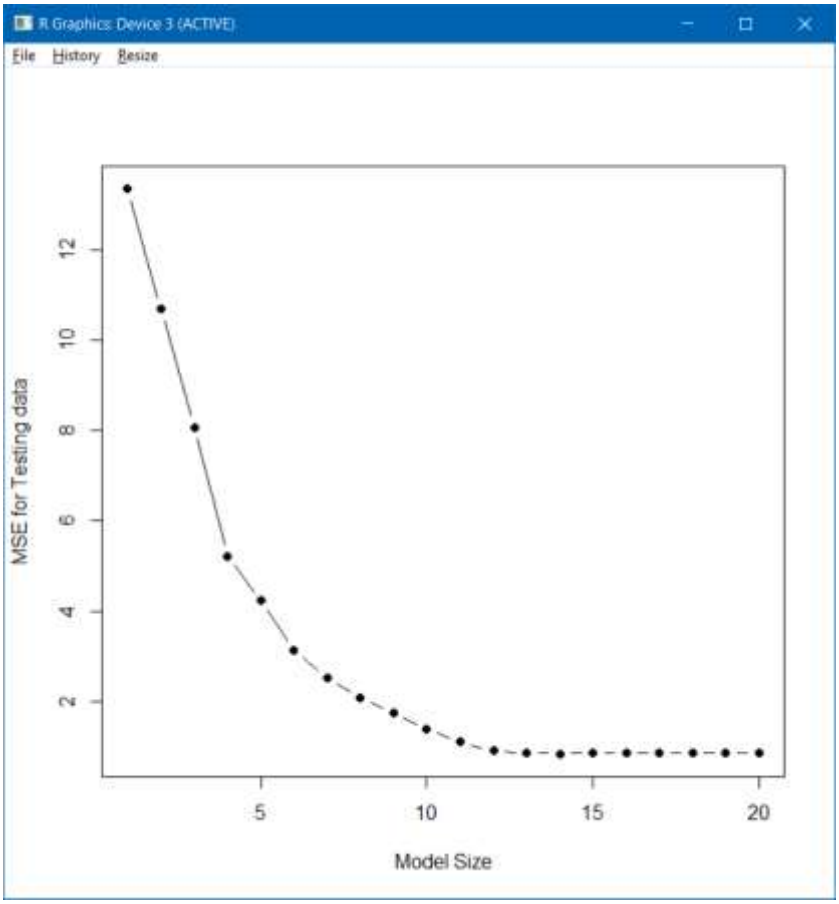
Kunal Mehta
50352960
kunalkam

```
1 subsets of each size up to 20
Selection Algorithm: forward
          x.1 x.2 x.3 x.4 x.5 x.6 x.7 x.8 x.9 x.10 x.11 x.12 x.13 x.14 x.15 x.16 x.17 x.18 x.19 x.20
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 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"  "*"  "*"  "*"  "*"  "*"  "*"  "*"  "*"  " "  "*"
20  ( 1 ) "*" "*" "*" "*" "*" "*" "*" "*" "*" "*"  "*"  "*"  "*"  "*"  "*"  "*"  "*"  "*"  "*"  "*"
> |
```

The next step was to calculate the training and testing MSEs associated with the best model of each size and plot them.



Kunal Mehta
50352960
kunalkam



c. Model size for which testing MSE is minimum: 14
There isn't much of an increase after 14 for higher values of model size. There is a sharp decrease in MSE for increasing model size from 1-5, followed by less sharp decreases that eventually plateau at about 12.

d. The coefficient values for the model size as 14 are:

```
> coefi
(Intercept)         x.1         x.2         x.5         x.6         x.8
-0.29333738 -0.57183989  1.26648007 -0.09168602  1.90195883  0.15552213
        x.11        x.12        x.13        x.14        x.15        x.16
 0.91786010  0.37366720 -2.72092434 -0.55517205  1.74175282  1.50970147
        x.17        x.18        x.20
 0.30056176  0.73899249  0.71846037
```

When compared to the true coefficients, these values are quite similar. The 0 values of the original coefficients at positions 3, 4, 9, 10 and 19 have been eradicated, as well as the value at position 7 which is very small in magnitude and close to 0. The coefficient values also are very close to the original values of coefficients I had at the beginning.

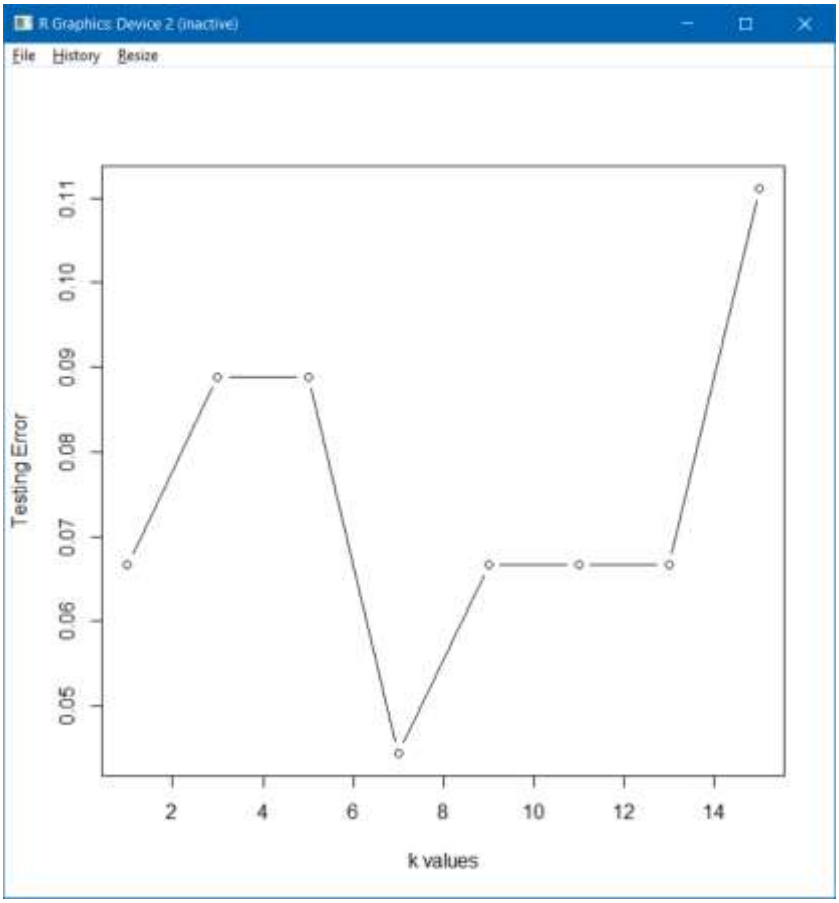
Problem 3

a. This problem required me to load the iris data and perform knn on the data for a range of k values. I took all the values from 1 to 15 as my range of k values to perform knn on. Then I simply looped over all these values, using the knn() function on the training and testing data and then storing the corresponding errors in two separate

lists. I am aware that only the testing error is relevant, but I calculated and plotted training error just to see what it was like.

(*It has been my general practice to use 100 as the seed for all the assignments I have done. In this case as well, I initially used the seed as 100 while working with my code, but I found that the error values are the same for quite a few of the k values. Also, my k values only include the odd values from 1-15. Upon thinking a bit about the code, it just occurred to me that with such a small dataset, the choice of rows which are part of the training and testing dataset would significantly impact any methods I use to analyse it, including knn. Therefore, I used a bunch of different seed values to find one that could lend me a k value that alone produces a lower testing error than any of the other k values. I did get such a result with seed 1234, which gave me 7 as the k with lowest error. I know that this sort of gerrymandering isn't really the motive of the question, and that even with the seed value of 100, my best k would be the smallest k which gave me the minimum error. Yet, I have stuck to using 1234 as my seed for the problem. Also, the hilarity of the situation if it so happens that I have solved this problem incorrectly is not lost on me. Would be anticlimactic to say the least.*)

The plots for the same:



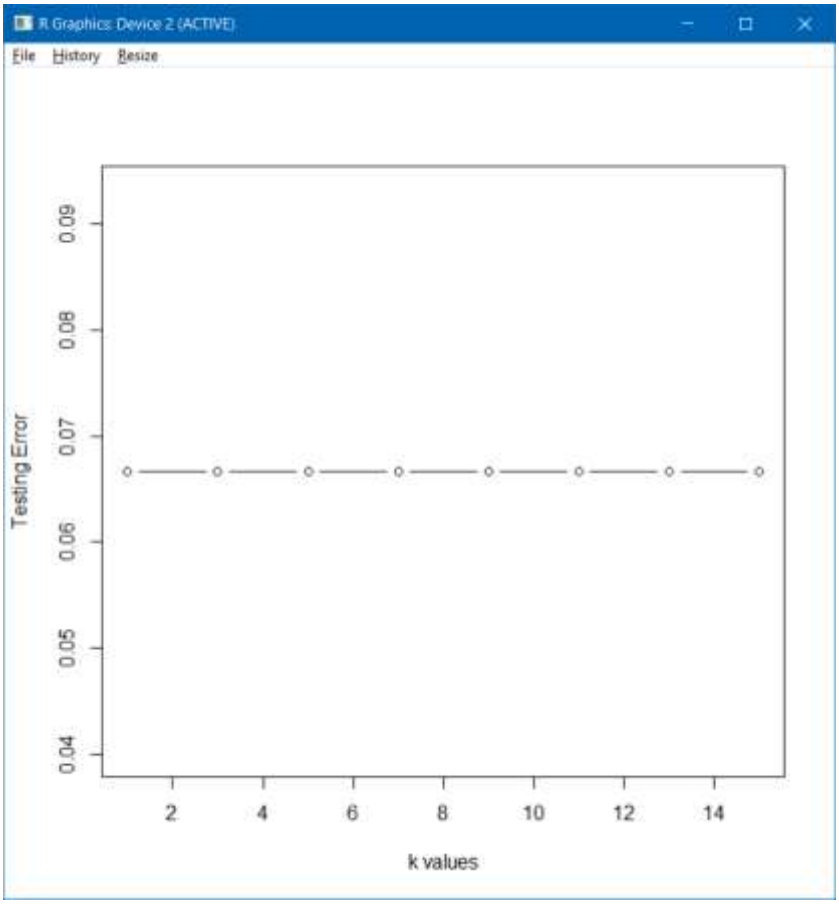
k-value for optimal model: 7
Confusion Matrix:

Kunal Mehta
50352960
kunalkam

```
> conf
Confusion Matrix and Statistics

            Reference
Prediction   setosa versicolor virginica
  setosa         16          0         0
  versicolor      0         16         2
  virginica       0          0        11

Overall Statistics

               Accuracy : 0.9556
                 95% CI : (0.8485, 0.9946)
    No Information Rate : 0.3556
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9327

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: setosa Class: versicolor Class: virginica
Sensitivity                 1.0000            1.0000           0.8462
Specificity                 1.0000            0.9310           1.0000
Pos Pred Value              1.0000            0.8889           1.0000
Neg Pred Value              1.0000            1.0000           0.9412
Prevalence                  0.3556            0.3556           0.2889
Detection Rate              0.3556            0.3556           0.2444
Detection Prevalence        0.3556            0.4000           0.2444
Balanced Accuracy           1.0000            0.9655           0.9231
> |
```

b. The next step was to do the same for the first 2 principal components. I first performed PCA on the data and isolated the first 2 principal components. Then I divided the data (PC1, PC2, Labels) into training and testing data.
The error plot for the same:

Kunal Mehta
50352960
kunalkam



The error for all values of k is the same, therefore I used the same value of k as the previous part, 7, as the optimal model.

The confusion matrix:

Kunal Mehta
50352960
kunalkam

```
> conf
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3
         1 14  0  0
         2  0 13  0
         3  0  3 15

Overall Statistics

               Accuracy : 0.9333
                 95% CI : (0.8173, 0.986)
    No Information Rate : 0.3556
    P-Value [Acc > NIR] : 5.426e-16

                  Kappa : 0.9001

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            1.0000   0.8125   1.0000
Specificity            1.0000   1.0000   0.9000
Pos Pred Value         1.0000   1.0000   0.8333
Neg Pred Value         1.0000   0.9062   1.0000
Prevalence             0.3111   0.3556   0.3333
Detection Rate         0.3111   0.2889   0.3333
Detection Prevalence   0.3111   0.2889   0.4000
Balanced Accuracy      1.0000   0.9062   0.9500
> |
```

The score plot for the principal components and sample coloration:

Kunal Mehta
50352960
kunalkam