Question 3.1.a

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier:

(a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional);

Answer:

To start with Cross Validation, we need perform the following steps.

Step 1: Activate the libraries

CODE:

```
setwd("D:/MS Georgia Tech/Introduction to Analytics/HW2")
library(kernlab)
library(kknn)
install.packages("dplyr")
library(dplyr)
set.seed(1)
```

Step 2: Load the data in R data frame.

CODE:

```
ccdata <- read.table("credit_card_data-headers.txt", stringsAsFactors = FALSE, header = TRUE)
glimpse(ccdata)
```

OUTPUT:

```
> glimpse(ccdata)
Observations: 654
Variables: 11
$ A1  <int> 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,...
$ A2  <dbl> 30.83, 58.67, 24.50, 27.83, 20.17, 32.08, 33.17, 22.92, 54.42, 42.50, 22.08, 29.92, 38.25, 48.08, 45....
$ A3  <dbl> 0.000, 4.460, 0.500, 1.540, 5.625, 4.000, 1.040, 11.585, 0.500, 4.915, 0.830, 1.835, 6.000, 6.040, 10...
$ A8  <dbl> 1.250, 3.040, 1.500, 3.750, 1.710, 2.500, 6.500, 0.040, 3.960, 3.165, 2.165, 4.335, 1.000, 0.040, 5.0...
$ A9  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
$ A10 <int> 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,...
$ A11 <int> 1, 6, 0, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 10, 3, 10, 0, 7, 17, 0, 6, 1, 3, 2, 9, 17, 3, 6, 5, 8, 1...
$ A12 <int> 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,...
$ A14 <int> 202, 43, 280, 100, 120, 360, 164, 80, 180, 52, 128, 260, 0, 0, 0, 320, 396, 120, 0, 96, 200, 300, 0, ...
$ A15 <int> 0, 560, 824, 3, 0, 0, 31285, 1349, 314, 1442, 0, 200, 0, 2690, 0, 0, 0, 245, 0, 0, 1208, 0, 1260, 11,...
$ R1  <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
>
```

Step 3: Using `train.kknn` to perform leave-one-out cross-validation

CODE:

```
# kernel choices
kern <- c("rectangular","triangular","epanechnikov","biweight","triweight","cos","inv","gaussian","optimal")
# pick a maximum k (number of neighbors) of 100
k_max <- 100
# distances to try
dist<-c(1,2,3,4,5,6,7,8,9,10)
model <-for (distances in dist) {
  mod<-train.kknn(R1~.,ccdata,kmax=k_max,distance=distances,kernel=kern,scale=TRUE)
  print(distances)
  print(mod)
}
```

OUTPUT:

```
[1] 1

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1880734
Minimal mean squared error: 0.1048715
Best kernel: triweight
Best k: 74
[1] 2

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1850153
Minimal mean squared error: 0.1061155
Best kernel: inv
Best k: 22
[1] 3

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1896024
Minimal mean squared error: 0.107919
Best kernel: biweight
Best k: 91
[1] 4

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1880734
Minimal mean squared error: 0.1084085
Best kernel: biweight
Best k: 98
[1] 5

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)
```

```
Type of response variable: continuous
minimal mean absolute error: 0.1865443
Minimal mean squared error: 0.1087676
Best kernel: biweight
Best k: 88
[1] 6

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1850153
Minimal mean squared error: 0.1090389
Best kernel: biweight
Best k: 92
[1] 7

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1850153
Minimal mean squared error: 0.109183
Best kernel: biweight
Best k: 83
[1] 8

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1880734
Minimal mean squared error: 0.1093045
Best kernel: biweight
Best k: 82
[1] 9

Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1865443
Minimal mean squared error: 0.1094212
Best kernel: biweight
Best k: 85
[1] 10
```

```
Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = k_max, distance = distance
s,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1880734
Minimal mean squared error: 0.1095125
Best kernel: biweight
Best k: 85
distance        7            8            9            10
call            Expression   Expression   Expression   Expression
terms           Expression   Expression   Expression   Expression
data            List,11      List,11      List,11      List,11
```

ANALYSIS: The best model is the model with lowest Minimal Mean Squared Error. If you look at the complete output above, it appears that lowest Minimal Mean Squared Error is 0.1048715 for distance = 1. So for the output with distance = 1, K = 74 which would be the best value of K.

CODE:

```
# The best model set is distance=1 with k=74 and kernel=triweight (lowest minimal mean squared error)
model1 <- train.kknn(R1~., ccdata, kmax=100, distance=1,scale = TRUE, kernel=kern)
model1
```

OUTPUT:

```
Call:
train.kknn(formula = R1 ~ ., data = ccdata, kmax = 100, distance = 1,     ker
nel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1880734
Minimal mean squared error: 0.1048715
Best kernel: triweight
Best k: 74
```

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier:
(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

Answer:

In order to split the data with ration of 60%:20%:20% where 60% belongs to training set, 20% belongs to testing set and 20% belongs to validation.

Step 1: This step is for calculating Testing data

CODE:

```
set.seed(1)

# pulling 20% indexes of random data points in the data set
test.indices = sample(1:nrow(ccdata), round(.2*nrow(ccdata)), replace=FALSE)
# pulling data for those indexes
test = ccdata[test.indices,]
nrow(test)
```

The test dataset generates 131 records

Step 2: This step is for calculating Validation data

CODE:

```
# the remainder of the data besides that which was set aside for testing
validfortraining = ccdata[-test.indices,]
# 20% for validation (25% of 80% is 20%)
validation.indices = sample(1:nrow(validfortraining), round(.25*nrow(validfortraining)), replace=FALSE)
validation = validfortraining [validation.indices,]
nrow(validation)
```

The validation dataset generates 131 records

Step 3: This step is for calculating Training data

CODE:

```
# The remainder of the data -- 80% -- is for training
train = as.data.frame(validfortraining[-validation.indices,])
nrow(train)
```

The train dataset generates 392 records

Step 4: Find the best value of K

CODE:

```
# kernel choices
 kern <- c("rectangular","triangular","epanechnikov","biweight","triweight","cos","inv","gaussian","optimal")
 # pick a maximum k (number of neighbors) of 100
```

```
k_max <- 100
# distances to try
dist<-c(1,2,3,4,5,6,7,8,9,10)
model <-for (distances in dist) {
 mod<-train.kknn(R1~.,train,kmax=k_max,distance=distances,kernel=kern,scale=TRUE)
 print(distances)
 print(mod)
}
```

OUTPUT:

```
[1] 1

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1989796
Minimal mean squared error: 0.1114249
Best kernel: rectangular
Best k: 14
[1] 2

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1938776
Minimal mean squared error: 0.1146717
Best kernel: inv
Best k: 17
[1] 3

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.188792
Minimal mean squared error: 0.1167837
Best kernel: inv
Best k: 17
[1] 4

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1960027
```

```
Minimal mean squared error: 0.1166515
Best kernel: inv
Best k: 18
[1] 5

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1964286
Minimal mean squared error: 0.1177008
Best kernel: inv
Best k: 17
[1] 6

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1938776
Minimal mean squared error: 0.117866
Best kernel: inv
Best k: 19
[1] 7

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1938776
Minimal mean squared error: 0.1188797
Best kernel: inv
Best k: 18
[1] 8

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1938776
Minimal mean squared error: 0.1192697
Best kernel: inv
Best k: 19
[1] 9

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)
```

```
Type of response variable: continuous
minimal mean absolute error: 0.1938776
Minimal mean squared error: 0.120359
Best kernel: triangular
Best k: 41
[1] 10

Call:
train.kknn(formula = R1 ~ ., data = train, kmax = k_max, distance = distances
,     kernel = kern, scale = TRUE)

Type of response variable: continuous
minimal mean absolute error: 0.1870234
Minimal mean squared error: 0.1205148
Best kernel: triangular
Best k: 41
```

ANALYSIS: The output with distance = 1 has the lowest minimal mean squared error. So the best Kernel = rectangular and best K = 14

Given below is the code to calculate accuracy. We first run the model on the test and validation data.

CODE:

```
model2 <-
kknn(train$R1~train$A1+train$A2+train$A3+train$A8+train$A9+train$A10+train$A11+train$A12+train$A14+tra
in$A15,
        validation, test, distance=1, kernel="rectangular",k=14,scale=TRUE)
# how well does the model predict the response variable?
predknn = model2$fitted.values
sum(predknn == train$R1) / nrow(test)
```

OUTPUT:

```
0.740458
```

The model is 74.04% accurate.

Question 4.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

Answer:

I was once working on a project and the data set given to us was a dataset of sales in the apparel industry. It had data for companies like, GAP, Old Nay, Banana Republic, Abercrombie and Fitch, Hollister etc. We were supposed to give some business recommendation after analysis of the data. As a part of the project we applied association rules algorithm to the data using a tool called as XL Miner and we found out that Gap was Highly associated to Old Navy. Though they were sister brands, GAP was losing all its customers to Old Navy.

In a situation like this we used clustering to raise the sales of GAP so that it would not lose its customers to Old Navy or other brands. We categorized the customers that visit GAP into 3 clusters. Given below are the details of the clusters.

Cluster 1 was named as GRAZERS.

1. Education is low may be have just completed school.
2. Age is moderate to high compared to other clusters.
3. Income is Moderate due to profession or they are retired.
4. As the income is moderate their expenditure is moderate.
5. House hold size is relatively high as they have a close-knit family.

Cluster 2 was named as BARGAIN HUNTERS.

1. Education is moderate as they are either in College or Graduating.
2. Age is lowest among all clusters.
3. Income is low as they are either doing an On-Campus job or doing an Internship.
4. Expenditure is low as their income is low; they always hunt for Best Deals.
5. House hold size is lowest as they are teenagers.

Cluster 3 was named as HIGH ROLLERS.

1. Education is highest among all clusters.
2. Age is quite moderate.
3. Income is highest among all clusters which can be derived from their profession.
4. Expenditure is highest as their income is very high.
5. House hold size is relatively high as they have a close-knit family.

Question 4.2

The iris data set iris.txt contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris). The response values are only given to see how well a specific method performed and should not be used to build the model.
Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

Answer:

Given below are the steps performed to eventually find the best value of k.

Step 1: Load all libraries.

CODE:

```
rm(list = ls())
set.seed(1)
library(datasets)
library(ggplot2)
install.packages("corrplot")
library(corrplot)
library(gridExtra)
```

Step 2: Load the IRIS dataset into the dataframe

CODE:

```
iris = read.delim("iris.txt")
```

Step 3: Analysis for best value of K

We can plot the variables in four different ways using color to represent the species in the data set.

The 4 ways are

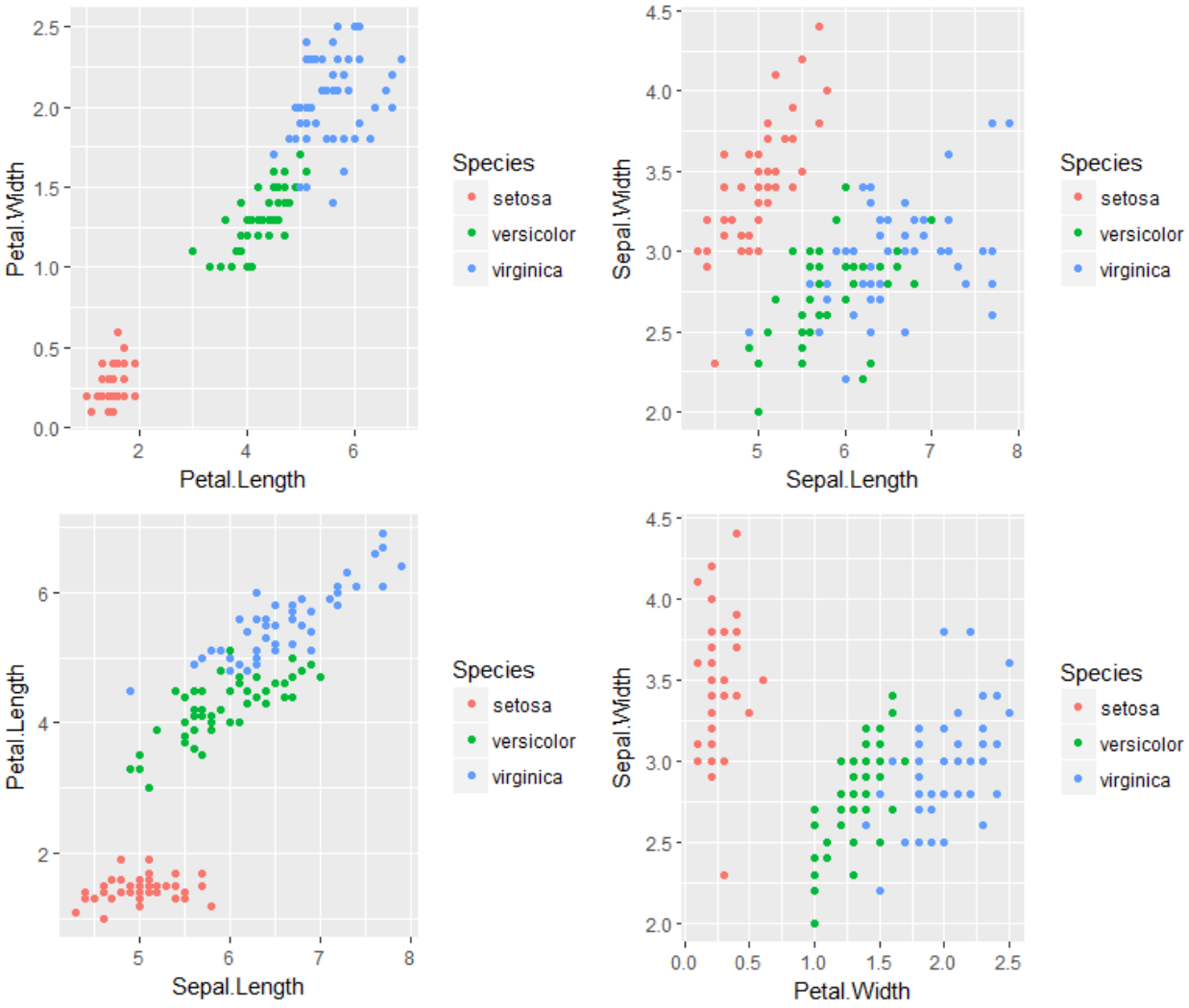
1. Petal Length vs Petal Width
2. Sepal Length vs Sepal Width
3. Sepal Length vs Petal Length
4. Petal Width vs Sepal Width

CODE:

```
# visualize the data
plot1 <- ggplot(iris, aes(Petal.Length, Petal.Width, color=Species)) + geom_point()
plot2 <- ggplot(iris, aes(Sepal.Length, Sepal.Width, color=Species)) + geom_point()
plot3 <- ggplot(iris, aes(Sepal.Length, Petal.Length, color=Species)) + geom_point()
plot4 <- ggplot(iris, aes(Petal.Width, Sepal.Width, color=Species)) + geom_point()
```

grid.arrange(plot1,plot2,plot3,plot4,nrow=2)

OUTPUT:



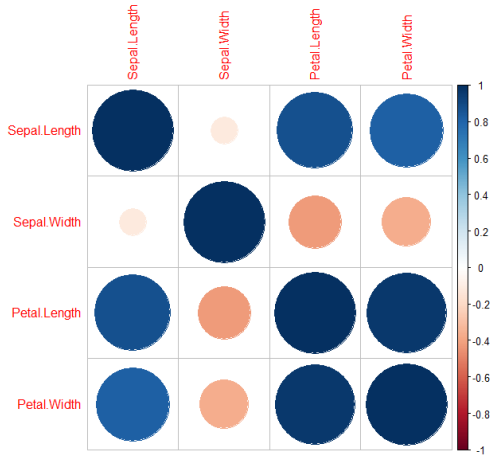
Looking at the plot above, it suggests that the plots with sepal width and Sepal Length (right plot above) are not so helpful for clustering of data.

We can also plot the correlations between each variable and see if we can find any relationships.

CODE:

 # correlation between any two variables
corrplot(cor(iris[,1:4]))

OUTPUT:



Looking at the plot above Sepal Width does not appear to be helpful as it does not have a good correlation with the other fields. So we can stop considering Sepal Width.

So we use the Kmeans function in R using Sepal Length, Petal Length and Petal Width. We will iterate through multiple values of K and store the data in a vector named km

CODE:

```
km <- c()
for (k in 1:10){
m <-kmeans(cbind(iris[,1],iris[,3:4]),k,nstart=20,iter.max=20)
km[k] = (m$tot.withinss)
}
km
```

OUTPUT:

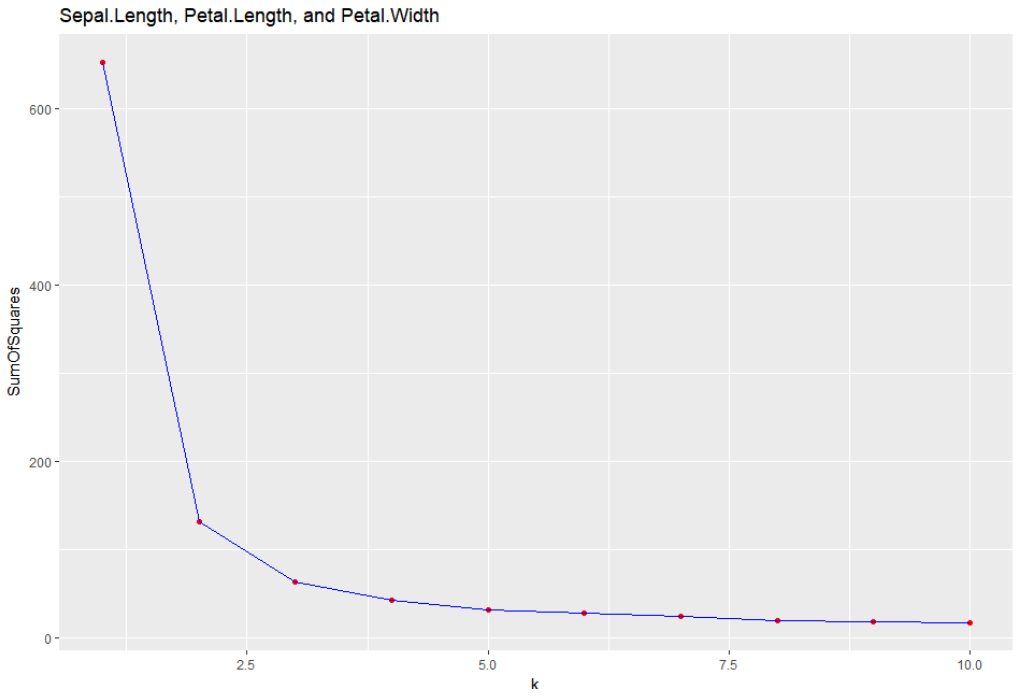
```
km
 [1] 653.06367 131.88039  63.34212  42.19179  31.64576  27.31261  23.39318  1
9.61958  17.94542  16.35759
```

Now we can plot our diagram and see what value of k would work best for our k means clustering.

CODE:

```
plot_best <- as.data.frame(cbind(k=1:10,SumOfSquares=km))
gg_best <- ggplot(plot_best,aes(x=k,y=SumOfSquares)) + geom_point(color="red") + geom_line(color="red") +
ggtitle("Sepal.Length, Petal.Length, and Petal.Width")
plot(gg_best)
```

OUTPUT:



Analysis: Looking at the elbow diagram above, it appears that K = 3 is the best choice.

So choosing K = 3 from elbow diagram

CODE:

```
# choosing k=3 from the elbow diagram
km_3 <- kmeans(cbind(iris[,1],iris[,3:4]),3,nstart=20)
table(km_3$cluster,iris$Species)
```

OUTPUT:

```
  setosa versicolor virginica
1      0          2        36
2     50          0         0
3      0         48        14
```