HW2

Question 3.1
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); and
(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

```
#loading libraries necessary for my problem
install.packages("kernlab",repos = "http://cran.us.r-project.org")
install.packages("kknn",repos = "http://cran.us.r-project.org")
#CARET  = classification and regression training, contains functions to strea
mline the model training process. Added the https link because of a mirror er
ror I was getting.
install.packages("caret",repos = "http://cran.us.r-project.org")
library(kernlab)
library(kknn)
library(caret)
```

```
## Loading required package: ggplot2
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:kernlab':
##
##     alpha
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

```
#Reading the credit card data in tab delimited text
cc_data<-read.delim("credit_card_data-headers.txt", header=TRUE)
#Checking our  data and the dimension
head(cc_data)
```

```
##   A1    A2    A3   A8 A9 A10 A11 A12 A14 A15 R1
## 1  1 30.83 0.000 1.25  1   0   1   1 202   0  1
## 2  0 58.67 4.460 3.04  1   0   6   1  43 560  1
## 3  0 24.50 0.500 1.50  1   1   0   1 280 824  1
## 4  1 27.83 1.540 3.75  1   0   5   0 100   3  1
## 5  1 20.17 5.625 1.71  1   1   0   1 120   0  1
## 6  1 32.08 4.000 2.50  1   1   0   0 360   0  1
```

```
dim(cc_data)
```

```
## [1] 654  11
```

```
#Initializing a pseudorandom number generator, to ensure we get the smae resu
lts if we start with the same seed
set.seed(123)
#Create a set of test/training data partition with the last column of our dat
a, and 80% of our data will be training and validayion as recommended in the
video lectures.We do not want the result to be in a list setting list to fals
e
split_data<-createDataPartition(y=cc_data$R1,p=0.8,list=FALSE)
#Our training data takes the portion of our cc_data, our split data is a list
of matrix of row positions integers corresponding to the training data
train_data<-cc_data[split_data, ]
head(train_data)
```

```
##   A1    A2     A3   A8 A9 A10 A11 A12 A14   A15 R1
## 2  0 58.67  4.460 3.04  1   0   6   1  43   560  1
## 4  1 27.83  1.540 3.75  1   0   5   0 100     3  1
## 5  1 20.17  5.625 1.71  1   1   0   1 120     0  1
## 6  1 32.08  4.000 2.50  1   1   0   0 360     0  1
## 7  1 33.17  1.040 6.50  1   1   0   0 164 31285  1
## 8  0 22.92 11.585 0.04  1   1   0   1  80  1349  1
```

```
#Verifying dimension  of train_data to make sure it is accurate
dim(train_data)
```

```
## [1] 524  11
```

```
#Our test data is the rest of the rows of cc data that are split data
test_data<-cc_data[-split_data,]
head(test_data)
```

```
##    A1    A2     A3    A8 A9 A10 A11 A12 A14 A15 R1
## 1   1 30.83  0.000 1.250  1   0   1   1 202   0  1
## 3   0 24.50  0.500 1.500  1   1   0   1 280 824  1
## 9   1 54.42  0.500 3.960  1   1   0   1 180 314  1
## 15  0 45.83 10.500 5.000  1   0   7   0   0   0  1
## 18  0 23.25  5.875 3.170  1   0  10   1 120 245  1
## 22  1 23.25  1.000 0.835  1   1   0   1 300   0  1
```

```
#Verifying dimension of test_data
dim(test_data)
```

```
## [1] 130  11

#Performing a repeated K-fold cross validation. The number 10 states that the
given dataset will be split into 10 folds(or subsets). Among the 10 folds, th
e model is trained on the 9 subsets and the remaining subset will be used to
evaluate the model's performance. These steps will be repeated up to a 5 time
s( according to our repeats parameter)
trainControl<-trainControl(method = "repeatedcv", number = 10, repeats=5)
#In train() method, "trControl()" method should be passed with results from t
rainControl() method.
#The "preProcess" parameter is used to preprocess our training data. We are p
assing "center" and "scale" parameter for centering and scaling the data.Here
we are converting our training data with mean value as 0 and standard deviati
on as 1. The "tuneLength" parameter holds an integer value, which is used for
tuning our algorithm.
knn_model_train<-train(as.factor(R1)~.,data = train_data,method="knn",trContr
ol=trainControl,preProcess= c("center","scale"),tuneLength=15)
knn_model_train

## k-Nearest Neighbors
##
## 524 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 470, 472, 471, 472, 472, 472, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8498777  0.6998562
##    7  0.8483538  0.6971119
##    9  0.8445659  0.6894465
##   11  0.8403857  0.6806746
##   13  0.8407921  0.6812250
##   15  0.8373881  0.6742275
##   17  0.8388832  0.6771452
##   19  0.8289193  0.6565058
##   21  0.8296880  0.6578685
##   23  0.8312409  0.6603403
##   25  0.8335056  0.6643771
##   27  0.8389047  0.6749974
##   29  0.8377651  0.6726059
##   31  0.8381422  0.6732609
##   33  0.8385053  0.6739479
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```r
#Repeating the same thing for 2-3 different values of number of fold and repe
ats just to see how much they will differ.
trainControl2<-trainControl(method = "repeatedcv", number = 15, repeats=5)
knn_model_train2<-train(as.factor(R1)~.,data = train_data,method="knn",trCont
rol=trainControl2,preProcess= c("center","scale"),tuneLength=15)
knn_model_train2
```

```
## k-Nearest Neighbors
##
## 524 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (15 fold, repeated 5 times)
## Summary of sample sizes: 489, 489, 488, 490, 490, 488, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8506567  0.7008939
##    7  0.8521488  0.7044802
##    9  0.8468042  0.6936688
##   11  0.8438531  0.6876598
##   13  0.8416103  0.6826324
##   15  0.8423063  0.6842393
##   17  0.8430582  0.6854418
##   19  0.8335294  0.6660383
##   21  0.8350875  0.6688922
##   23  0.8304824  0.6591012
##   25  0.8381612  0.6738672
##   27  0.8381183  0.6734805
##   29  0.8404027  0.6780642
##   31  0.8415251  0.6799790
##   33  0.8392275  0.6751707
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```r
trainControl3<-trainControl(method = "repeatedcv", number =20, repeats=10)
knn_model_train3<-train(as.factor(R1)~.,data = train_data,method="knn",trCont
rol=trainControl3,preProcess= c("center","scale"),tuneLength=15)
knn_model_train3
```

```
## k-Nearest Neighbors
##
## 524 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
```

```
## Resampling: Cross-Validated (20 fold, repeated 10 times)
## Summary of sample sizes: 497, 498, 498, 498, 498, 498, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.8480983  0.6958065
##    7  0.8472792  0.6946509
##    9  0.8425356  0.6852238
##   11  0.8400570  0.6800997
##   13  0.8354986  0.6704442
##   15  0.8355057  0.6705611
##   17  0.8387678  0.6770726
##   19  0.8364957  0.6722622
##   21  0.8303561  0.6595136
##   23  0.8292165  0.6566221
##   25  0.8331909  0.6640831
##   27  0.8343234  0.6658823
##   29  0.8356838  0.6681048
##   31  0.8375926  0.6719041
##   33  0.8397080  0.6760007
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
#I decided to test my model with my first two values of k-fold and repeats : 10 and 5
knn_model_test<-train(as.factor(R1)~.,data = test_data,method="knn",trControl=trainControl,preProcess= c("center","scale"),tuneLength=15)
knn_model_test
```

```
## k-Nearest Neighbors
##
## 130 samples
##  10 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered (10), scaled (10)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 116, 116, 118, 116, 117, 117, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.7877656  0.5465126
##    7  0.8288462  0.6269446
##    9  0.8274359  0.6233324
##   11  0.8244505  0.6160561
##   13  0.8321612  0.6360786
##   15  0.8228938  0.6143440
##   17  0.8231136  0.6151843
##   19  0.8231319  0.6137643
```

```
##   21  0.8081685  0.5806408
##   23  0.8049267  0.5735399
##   25  0.7969963  0.5539006
##   27  0.7971245  0.5502742
##   29  0.7984432  0.5508733
##   31  0.7987729  0.5519478
##   33  0.7924725  0.5388864
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 13.
```

#The 10-fold cross validation ensures that k=5 gives us the highest accuracy for our KNN training model(84.98%) while k=13 yields the highest accuracy for our KNN test model(83.21%).

#Testing the k fold validation with KVSM, split our data as follow: 70% to tr aining, and the rest 30% to testing and validation

```
set.seed(123)
split_dt<-createDataPartition(y=cc_data$R1,p=0.7, list=FALSE)
train_dt<-cc_data[split_dt,]
dim(train_dt)
```

```
## [1] 458  11
```

```
rest_dt<-cc_data[-split_dt, ]
dim(rest_dt)
```

```
## [1] 196  11
```

```
split_dt2<-createDataPartition(y=rest_dt$R1, p =0.5, list=FALSE)
validation_dt<-rest_dt[split_dt2, ]
dim(validation_dt)
```

```
## [1] 98 11
```

```
test_dt<-rest_dt[-split_dt2, ]
dim(test_dt)
```

```
## [1] 98 11
```

 # Reusing the function from HW1 to test the accuracy of each model
#

```
check_accuracy = function(X){
  predicted <- rep(0,(nrow(train_dt)))

  for (i in 1:nrow(train_dt)){
 model=kknn(R1~.,train_dt[-i,],train_dt[i,],k=X, scale = TRUE)
 predicted[i] <- as.integer(fitted(model)+0.5) # round off to 0 or 1
  }
```

```
  accuracy = sum(predicted == train_dt[,11]) / nrow(train_dt)
  return(accuracy)
}

acc <- rep(0,20) # set up a vector of 20 zeros to start
for (X in 1:20){
  acc[X] = check_accuracy(X) # test knn with X neighbors
}

acc
```

```
##  [1] 0.8078603 0.8078603 0.8078603 0.8078603 0.8427948 0.8362445 0.8406114
##  [8] 0.8427948 0.8493450 0.8558952 0.8537118 0.8558952 0.8558952 0.8537118
## [15] 0.8515284 0.8515284 0.8537118 0.8537118 0.8537118 0.8493450
```

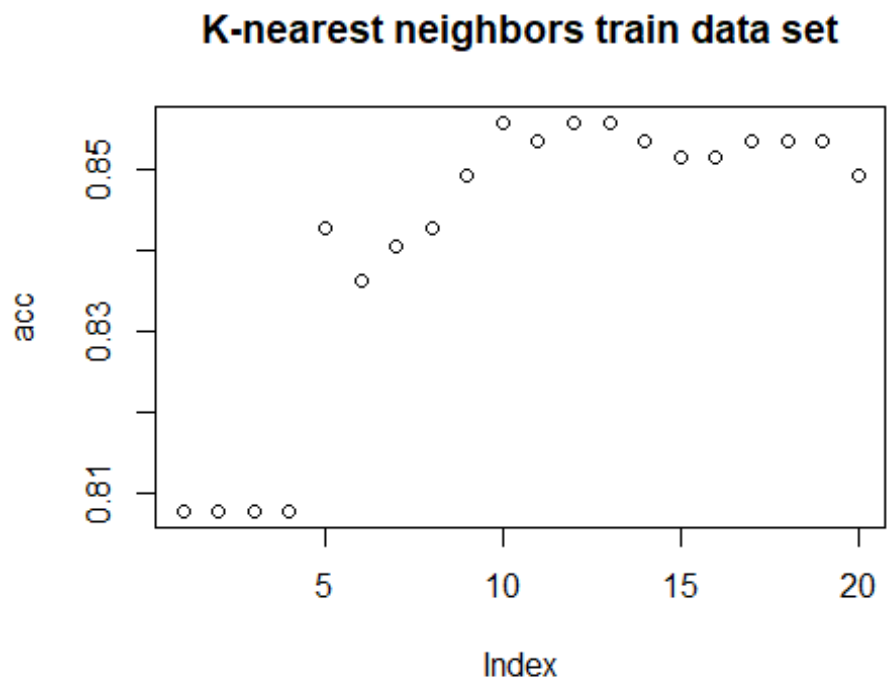
```
max(acc)
```

```
## [1] 0.8558952
```

```
which.max(acc)
```

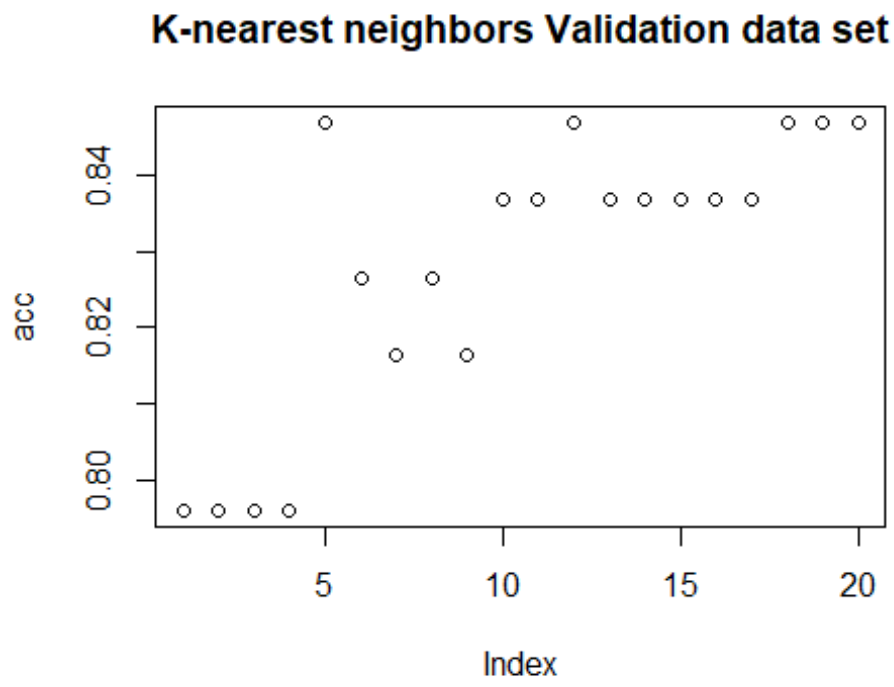
```
## [1] 10
```

```
plot(acc)
title("K-nearest neighbors train data set")
```




```
check_accuracy = function(X){
  predicted <- rep(0,(nrow(validation_dt))) # predictions: start with a vecto
```

```
r of all zeros

  # for each row, estimate its response based on the other rows

  for (i in 1:nrow(validation_dt)){

    # data[-i] means we remove row i of the data when finding nearest neighbo
rs...
    #...otherwise, it'll be its own nearest neighbor!

    model=kknn(R1~.,validation_dt[-i,],validation_dt[i,],k=X, scale = TRUE) #
use scaled data

    # record whether the prediction is at least 0.5 (round to one) or less th
an 0.5 (round to zero)

    predicted[i] <- as.integer(fitted(model)+0.5) # round off to 0 or 1
  }

  accuracy = sum(predicted == validation_dt[,11]) / nrow(validation_dt)
  return(accuracy)
}

acc <- rep(0,20) # set up a vector of 20 zeros to start
for (X in 1:20){
  acc[X] = check_accuracy(X) # test knn with X neighbors
}

acc

##  [1] 0.7959184 0.7959184 0.7959184 0.7959184 0.8469388 0.8265306 0.8163265
##  [8] 0.8265306 0.8163265 0.8367347 0.8367347 0.8469388 0.8367347 0.8367347
## [15] 0.8367347 0.8367347 0.8367347 0.8469388 0.8469388 0.8469388

max(acc)

## [1] 0.8469388

which.max(acc)

## [1] 5

plot(acc)
title("K-nearest neighbors Validation data set")
```



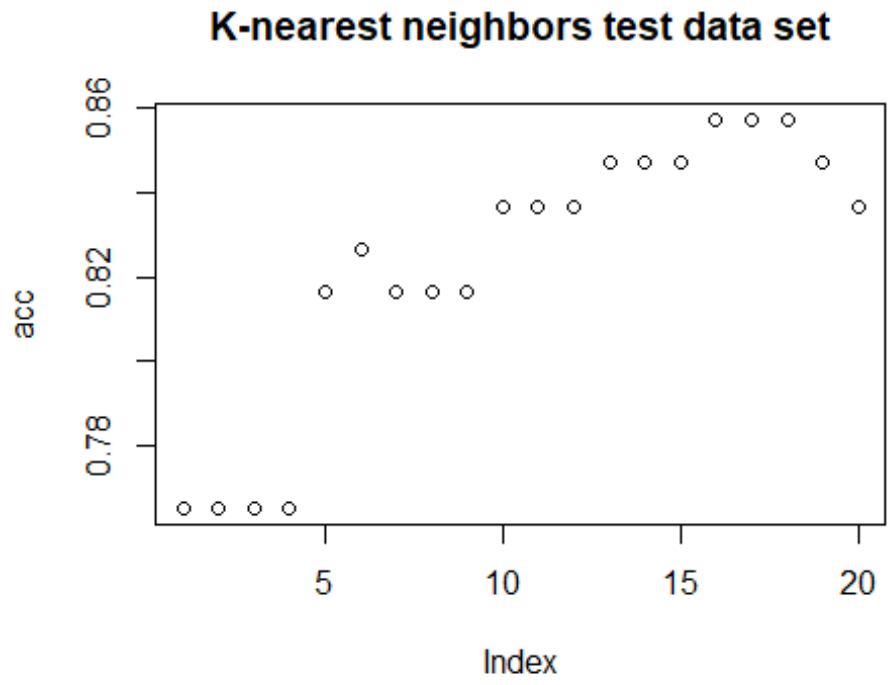

```
check_accuracy = function(X){
  predicted <- rep(0,(nrow(test_dt)))

  for (i in 1:nrow(test_dt)){

    model=kknn(R1~.,test_dt[-i,],test_dt[i,],k=X, scale = TRUE)    predicted[
i] <- as.integer(fitted(model)+0.5) # round off to 0 or 1
  }

  accuracy = sum(predicted == test_dt[,11]) / nrow(test_dt)
  return(accuracy)
}

acc <- rep(0,20) # set up a vector of 20 zeros to start
for (X in 1:20){
  acc[X] = check_accuracy(X) # test knn with X neighbors
}
acc
```

```
##  [1] 0.7653061 0.7653061 0.7653061 0.7653061 0.8163265 0.8265306 0.8163265
##  [8] 0.8163265 0.8163265 0.8367347 0.8367347 0.8367347 0.8469388 0.8469388
## [15] 0.8469388 0.8571429 0.8571429 0.8571429 0.8469388 0.8367347
```

```
max(acc)
```

```
## [1] 0.8571429
```

```
which.max(acc)

## [1] 16

plot(acc)
title("K-nearest neighbors test data set")
```




```
#Observing the graphs we notice the following:
#K=10, gives an accuracy of 85.58% for the training data
#K=5, gives an accuracy of 84.69% for the validation data
#K=16, gives an accuracy of 85.71% for the test data
```

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.

What city to open a luxury restaurant in?(Specifically what area). I would use the following classifiers:

- Average income of the neighborhood/zip code
- Average percentage of pescatarian (Assuming it is a sea food restaurant)
- Average numbers of luxury amenities around the area
- Average number of 3-star restaurants and above
- Average number of restaurant visits per day in the area

Question 4.2

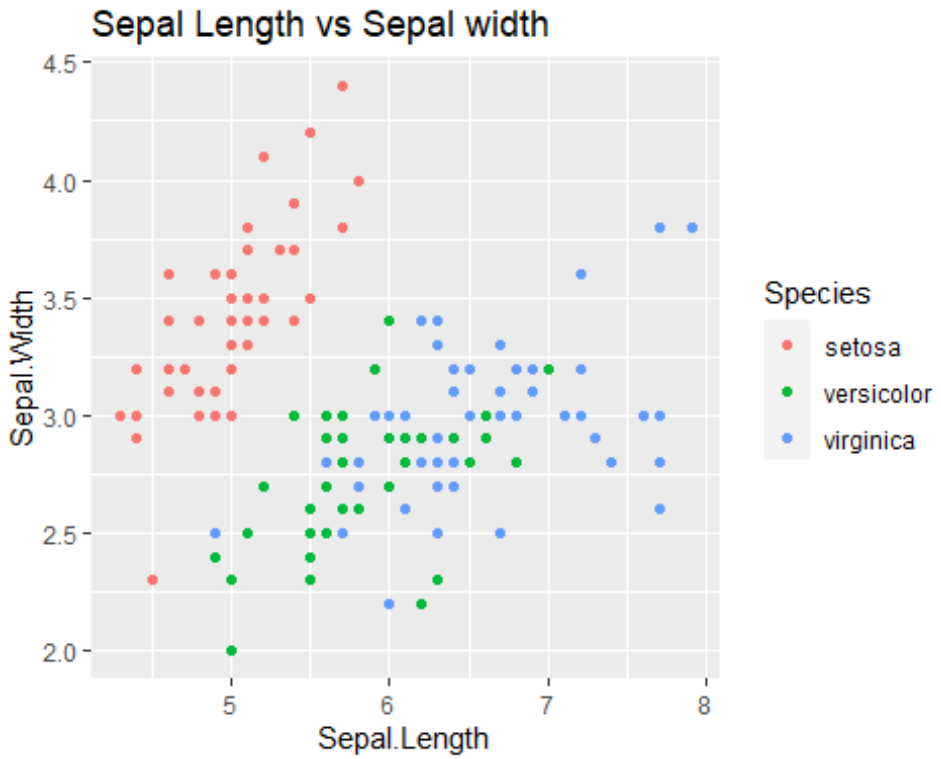
The *iris* data set `iris.txt` contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed 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.

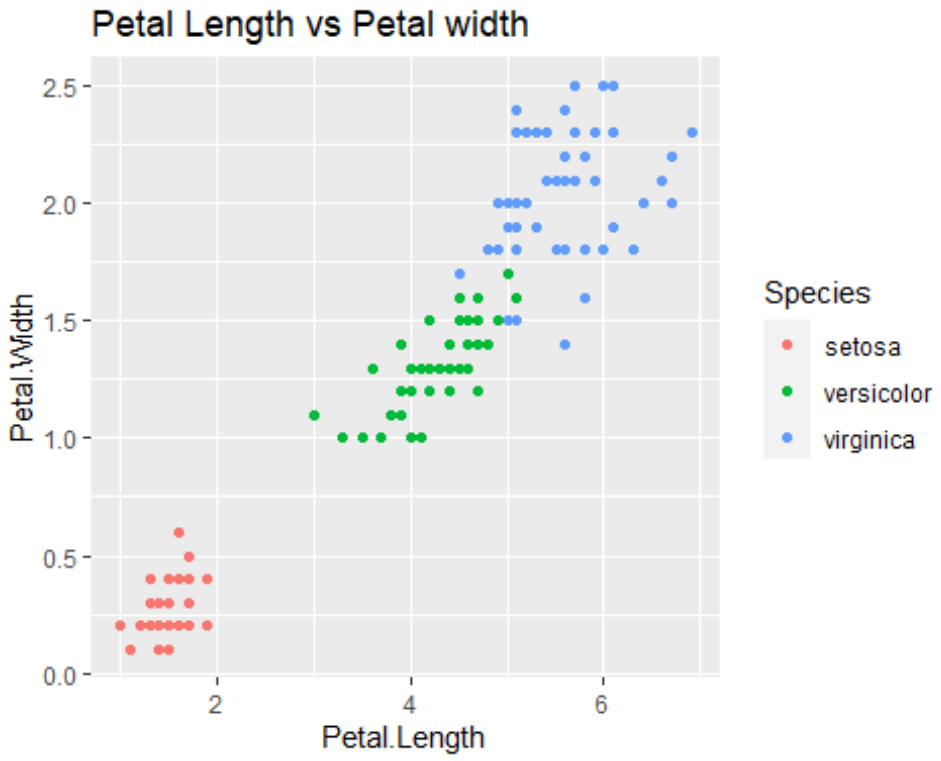
```
install.packages("tidyverse")
library(tidyverse)
library(ggplot2)
#adding a separator in my read.csv to separate the files columns
iris_data<-read.csv("iris.txt",sep="")
dim(iris_data)
## [1] 150   5

#View first six data points
head(iris_data)

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa

#View internal structure of data
str(iris_data)

## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : chr  "setosa" "setosa" "setosa" "setosa" ...

#View statistical distribution of variables in iris_data
summary(iris_data)

##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
```

```
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##    Species
##  Length:150
##  Class :character
##  Mode  :character
##
##
##
```

```
#The iris  data has 150 points and  4 numerical predictors and a response cal
led "Species". Plotting the Sepal.length vs Sepal.width to visualize the clus
ters
ggplot(iris_data,aes(x = Sepal.Length, y = Sepal.Width, col =Species))+geom_p
oint()+ggtitle("Sepal Length vs Sepal width")
```




```
#Plotting petal length vs petal wideh to visualize the clusters
ggplot(iris_data,aes(x=Petal.Length, y = Petal.Width, col =Species))+geom_poi
nt()+ggtitle("Petal Length vs Petal width")
```




```
#From the two plots above, we can see the Petal.length vs Petal.width  plot g
ives better clusters separation compared to the Sepal.length vs Sepal.width p
lot
#I am going to verify this by creating different combination and calculating
the total within-cluster sum of squares of each combination.
#I am picking a k=3 , since we have 3 different species of flowers
set.seed(200)
#Sepal length and sepal width
m1<-kmeans(iris_data[,1:2],3,iter.max = 10,nstart=1)
#Sepal length and Petal length
m2<-kmeans(iris_data[,1:3],3,iter.max = 10,nstart=1)
#Sepal length and Petal width
m3<-kmeans(iris_data[,1:4],3,iter.max = 10,nstart=1)
#Sepal width and Petal length
m4<-kmeans(iris_data[,2:3],3,iter.max = 10,nstart=1)
#Sepal width and Petal width
m5<-kmeans(iris_data[,2:4],3,iter.max = 10,nstart=1)
#Petal length and Petal width
m6<-kmeans(iris_data[,3:4],3,iter.max = 10,nstart=1)
#Sepal length,Sepal width and Petal length
m7<-kmeans(iris_data[,c(1,2,3)],3,iter.max = 10,nstart=1)
#Sepal length,Petal length and Petal width
m8<-kmeans(iris_data[,c(1:3,4)],3,iter.max = 10,nstart=1)
#Sepal length,Sepal width and Petal width
m9<-kmeans(iris_data[,c(1,2,4)],3,iter.max = 10,nstart=1)
#Sepal width,petal width and Petal length
m10<-kmeans(iris_data[,c(3,2,4)],3,iter.max = 10,nstart=1)
```

```r
#Sepal length,Sepal width , petal width and Petal length
m11<-kmeans(iris_data[,c(1,2,3,4)],3,iter.max = 10,nstart=1)


#Vector with the different within-clusters sum of squares, then we will deter
mine which model has the minimum.
total.distance<-c(m1$tot.withinss,m2$tot.withinss,m3$tot.withinss,m4$tot.with
inss,m5$tot.withinss,m6$tot.withinss,m7$tot.withinss,m8$tot.withinss,m9$tot.w
ithinss,m10$tot.withinss,m11$tot.withinss)
bestcombo<-which.min(total.distance)
minimumpercentage<-min(total.distance)
bestcombo
```

```
## [1] 6
```

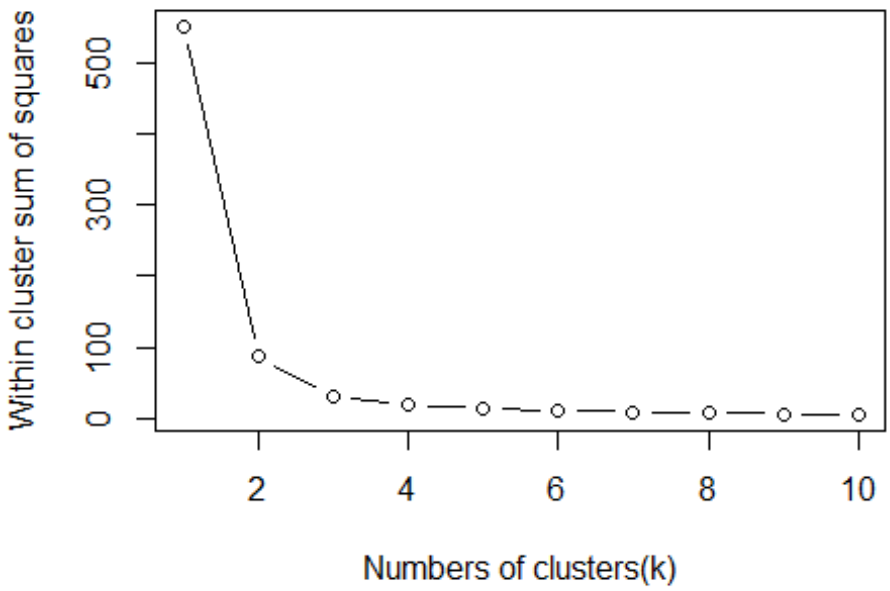
```r
minimumpercentage
```

```
## [1] 31.41289
```

```r
#Our verification confirmed that Petal.length vs Petal Width combination had
the lowest within-cluster sum of squares, so it gives us better clusters sepa
ration.
#Plotting number of clusters(k) vs within sum of squares
set.seed(456)
kmax<-10
cluster<- sapply(1:kmax,function(k){kmeans(iris[,3:4],k,nstart = 20,iter.max
= 20)$tot.withinss})
cluster
```

```
##  [1] 550.895333  86.390220  31.371359  19.465989  13.916909  11.025145
##  [7]   9.236596   7.816945   6.456495   5.550520
```

```r
plot(1:kmax,cluster,type="b", xlab="Numbers of clusters(k)", ylab="Within clu
ster sum of squares")
```




```
#From the plot, we notice that the optimal value of k is 3 (elbow point)
# Let's see how well our best clustering predicts our flower type. A value of
20 for n-start means we will try 20 different random starting assignments and
select the one with the lowest within cluster variation.
fcluster<-kmeans(iris_data[,3:4],3,nstart = 20)
table(fcluster$cluster,iris_data$Species)

##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0          2        46
##   3      0         48         4
```

Best combination- petal width and petal length

Optimal K: k=3

How best does it predict our flower type: please see table above