Homework-3.1_FINAL_v2.r

nhirata

2020-01-22

```
# 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 kk
nn function to find a good classifier:

# Question 3.1(a) Using cross-validation (do this for the k-nearest-neighbors model; SVM is optional)

rm(list = ls()) # Clear the list
library(kernlab)
library(kknn)
#Load kknn and kernlab libraries
set.seed(25) # per TA Lecture #2

#Read my credit card data
df <- read.table("C:/Users/nhirata/Desktop/Georgia Tech/OneDrive - Georgia Institute of Technology/Georgia Tech/ISYE_6501/We
ek_2/data 3.1/credit_card_data-headers.txt", header=TRUE, stringsAsFactors = FALSE)

#Look at the first rows of my data to see if the data comes out right based on the parameters set in read.table.
head(df)
```

```
##   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
```

```
# Use train.kknn to train the whole data set.
model_loocv <- train.kknn(R1~.,data = df, kmax = 100, scale = TRUE ) #iterates over all rows
model_loocv
```

```
##
## Call:
## train.kknn(formula = R1 ~ ., data = df, kmax = 100, scale = TRUE)
##
## Type of response variable: continuous
## minimal mean absolute error: 0.1850153
## Minimal mean squared error: 0.1073792
## Best kernel: optimal
## Best k: 58
```

```
#Loops over the number of desired folds to see which fold is the best for a certain K Nearest Neighbor
k=50
model_loocv_acc=rep(0,k)
for (k in 1:k) {
  pred <- as.integer(fitted(model_loocv)[[k]][1:nrow(df)] +.5)
  model_loocv_acc[k] <- sum(pred == df$R1) / nrow(df) *100
}
#When we perform a cross fold loop, the results show that folding 12 times provides the best accuracy of 85.32%
model_loocv_acc
```

```
##  [1] 81.49847 81.49847 81.49847 81.49847 85.16820 84.55657 84.70948 84.86239
##  [9] 84.70948 85.16820 85.16820 85.32110 85.16820 85.16820 85.32110 85.32110
## [17] 85.32110 85.16820 85.01529 85.01529 84.86239 84.70948 84.40367 84.55657
## [25] 84.55657 84.40367 84.09786 83.79205 83.94495 84.09786 83.79205 83.63914
## [33] 83.63914 83.33333 83.18043 83.18043 83.02752 83.18043 83.18043 83.18043
## [41] 83.18043 83.63914 83.63914 83.63914 83.94495 84.09786 83.79205 83.94495
## [49] 84.09786 83.94495
```

```
which.max(model_loocv_acc)
```

```
## [1] 12
```

```
max(model_loocv_acc)
```

```
## [1] 85.3211
```

```
#Question 3.1(a)(optional for ksvm)

#Per Professor Sokol (Lecture 3.3), "Rule of thumb is 50-70% Training and then split the rest equally" so I decided to meet
 it in the middle with 60-20-20 (60% train, 20% validate & test)
spec = c(train = .6, validate = .2, test = .2) #spec means specifications on how i want my data split up.

#Lecture 3.3 - I used the Random approach so i understand that there are possibilites of the data not being separated equall
y (more early or late data). But i rather used this approach rather than the Rotation approach because I did not want to inc
lude bias's (ex. 5 data point rotation with days).
g = sample(cut( # g means grouping based on taking random rows from the total population of rows
  seq(nrow(df)),
  nrow(df)*cumsum(c(0,spec)),
  labels = names(spec)
))
#split my data tables so that each one lives on its own.
res = split(df, g)
sapply(res, nrow)/nrow(df)
```

```
##     train  validate      test
## 0.5993884 0.2003058 0.2003058
```

```
addmargins(prop.table(table(g)))
```

```
## g
##     train  validate      test       Sum
## 0.5993884 0.2003058 0.2003058 1.0000000
```

```
#Renaming my dataframes
train <- res$train
validate <- res$validate
test <-res$test

#Evidence of distribution of # of rows
nrow(train)
```

```
## [1] 392
```

```
nrow(validate)
```

```
## [1] 131
```

```
nrow(test)
```

```
## [1] 131
```

```
#I used train data for my model and then predicted using my validation data per the TA.
#Once I know that the validation set provides a good accuracy, I will continue with my test data.
c_values= c( 10 ^(- 6 ), 10 ^(- 4 ), 10 ^(- 2 ), 1 , 10 , 10 ^ 2 , 10 ^ 4 , 10 ^ 6 )
SVM_Accuracy = c()
for(i in 1:length(c_values)){
  model_SVM <- ksvm(as.matrix(train[,1:10]),as.factor(train[,11]),type = "C-svc", kernel = "vanilladot", C =c_values[i], sca
led =TRUE)

  SVM_pred <-predict(model_SVM, validate[,1:10])
  SVM_Accuracy[i] <-sum(SVM_pred == validate[,11]) / nrow(validate) * 100
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
#Validation Data Analysis
SVM_Accuracy #List of accuracy per C value
```

```
## [1] 57.25191 57.25191 81.67939 81.67939 81.67939 81.67939 81.67939 84.73282
```

```
c_for_test <- c_values[which.max(SVM_Accuracy)]
c_for_test
```

```
## [1] 1e+06
```

```
max(SVM_Accuracy) #Semi-Final accuracy percentage was good at 84.7%
```

```
## [1] 84.73282
```

```
#Now that the validation data has been proven to work, I will run this again using the "test set". This will remove any chan
ce that the final % accuracy was inflated by luck (like the "validation set"). Since the "validation set" and "test set" rep
ort consistent results compared to each other, i can report the accuracy based on the "test set" to my stakeholders.
c_values= c( 10 ^(- 6 ), 10 ^(- 4 ), 10 ^(- 2 ), 1 , 10 , 10 ^ 2 , 10 ^ 4 , 10 ^ 6 )
SVM_Accuracy = c()
for(i in 1:length(c_values)){
  model_SVM <- ksvm(as.matrix(train[,1:10]),as.factor(train[,11]),type = "C-svc", kernel = "vanilladot", C =c_values[i], sca
led =TRUE)

  SVM_pred <-predict(model_SVM, test[,1:10])
  SVM_Accuracy[i] <-sum(SVM_pred == test[,11]) / nrow(test) * 100
}
```

```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

```
#Test Data Analysis
SVM_Accuracy #List of accuracy per C value
```

```
## [1] 48.09160 48.09160 83.20611 83.20611 83.20611 83.20611 83.20611 78.62595
```

```
c_for_test <- c_values[which.max(SVM_Accuracy)]
c_for_test
```

```
## [1] 0.01
```

```
max(SVM_Accuracy) #Final accuracy percentage
```

```
## [1] 83.20611
```

```
# Question 3.1(b) Split the data into training, validation, and test data sets for kknn.
# I listed different values of k to use for the most optimal k while splitting the data
# By training on the "train set" and testing on the "validation set", we can see if this model provides good accuracy %'s ba
sed on each value of k and the quality of the "validation set". If the model provided bad accuracies, I would revisit fixin
g/replacing a new model and then check again. (iterating process)
k_values = c(5,10,20,25,50,100)
kknn_accuracy=c()
for(n in 1:length(k_values)){
  model <- kknn(R1~., train = train,test=validate, k = k_values[n], scale = TRUE)
  predicted <-as.integer(fitted(model)+.5)
  kknn_accuracy[n] <-sum(predicted == validate$R1)/nrow(validate)*100
}

#Validation Analysis
kknn_accuracy
```

```
## [1] 79.38931 77.86260 79.38931 79.38931 81.67939 84.73282
```

```
max(kknn_accuracy)
```

```
## [1] 84.73282
```

```
k_for_test <- k_values[which.max(kknn_accuracy)]
k_for_test
```

```
## [1] 100
```

```
# If the validation model above is consistently good, I can start testing with the "test set". This will remove any chance t
hat the final % accuracy was inflated by luck (like the "validation set"). Since the "validation set" and "test set" report
 consistent results compared to each other, i can report the accuracy based on the "test set" to my stakeholders.
for(n in 1:length(k_values)){
  model <- kknn(R1~., train = train,test=test, k = k_values[n], scale = TRUE)
  predicted <-as.integer(fitted(model)+.5)
  kknn_accuracy[n] <-sum(predicted == test$R1)/nrow(test)*100
}

#Test Analysis
kknn_accuracy
```

```
## [1] 82.44275 81.67939 82.44275 82.44275 82.44275 83.20611
```

```
max(kknn_accuracy)
```

```
## [1] 83.20611
```

```
k_for_test <- k_values[which.max(kknn_accuracy)]
k_for_test
```

```
## [1] 100
```

```
#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.

#At the Jet Propulsion Laboratory (JPL), I work as a Business Analyst for the Mission Systems Engineering Section. Our Techi
nical Group Supervisors inherently use clustering when they performance coach their groups. Predictors such as education lev
el, years of experience, current position, current level of the employee (1-6), and current level of performance (1-9) based
on JPL's matrix of roles & responsibilities, helps the Group Supervisors identify how to speak with each class of employee f
or efficient/effective clear communication. Clustering the data into 3 groups (Early Career, Mid Career, and Senior hires) c
an provide strategic input to business managment on whether we need to hire a certain group of employee for balanced diversi
fication. We are currently hiring more Early Career hires now due to the large proportion of Senior hires that are getting r
eady to retire in the next 5-10 years.


#Question 4.2 Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictor
s, your suggested value of k, and how well your best clustering predicts flower type.

rm(list = ls())
iris <- read.table("C:/Users/nhirata/Desktop/Georgia Tech/OneDrive - Georgia Institute of Technology/Georgia Tech/ISYE_6501/
Week_2/data 4.2/iris.txt", header=TRUE, stringsAsFactors = FALSE)
iris <- iris
head(iris)
```

```
##   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
```

```
table(iris$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

```
library(ggplot2)
```

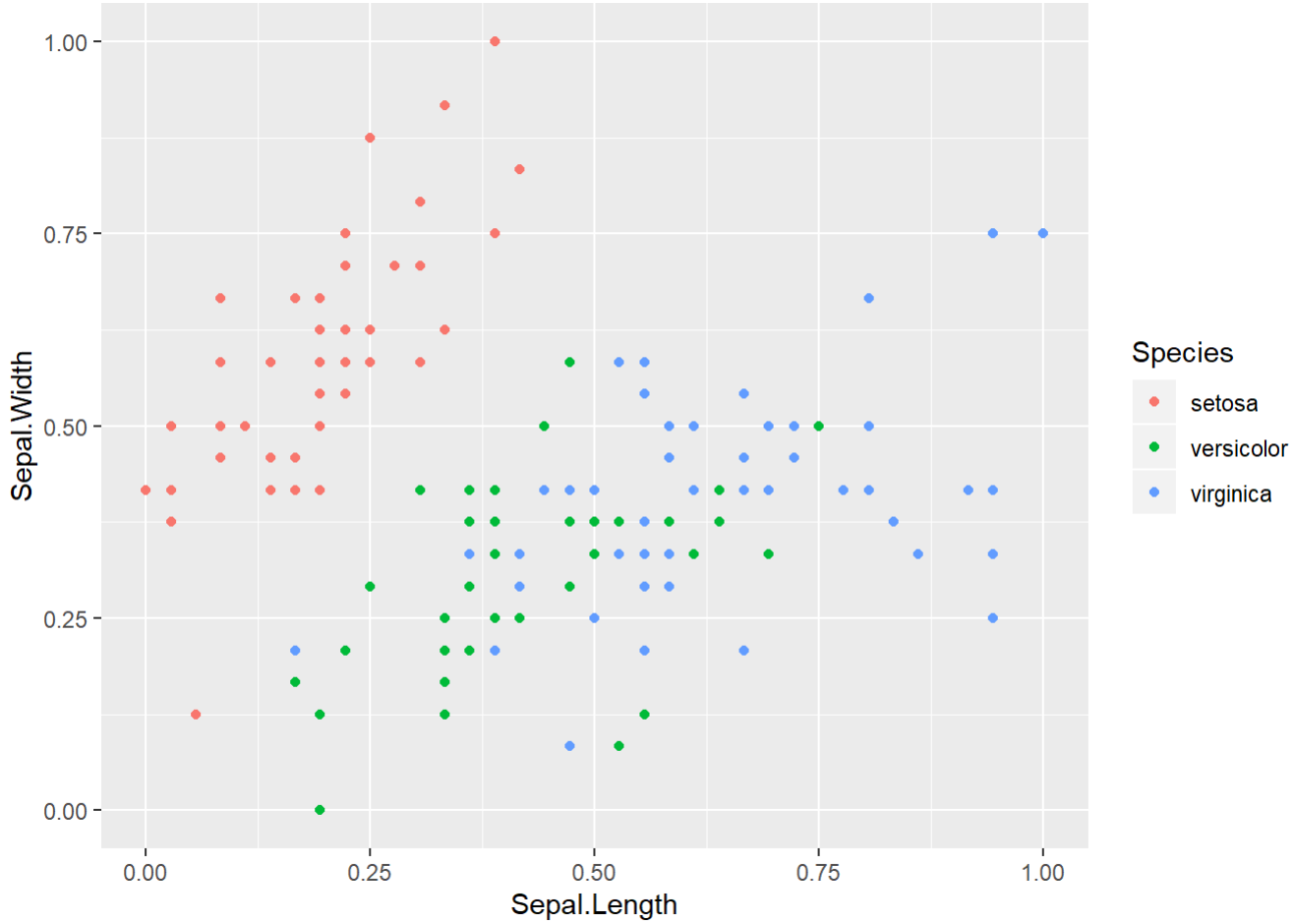
```
##
## Attaching package: 'ggplot2'
```

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

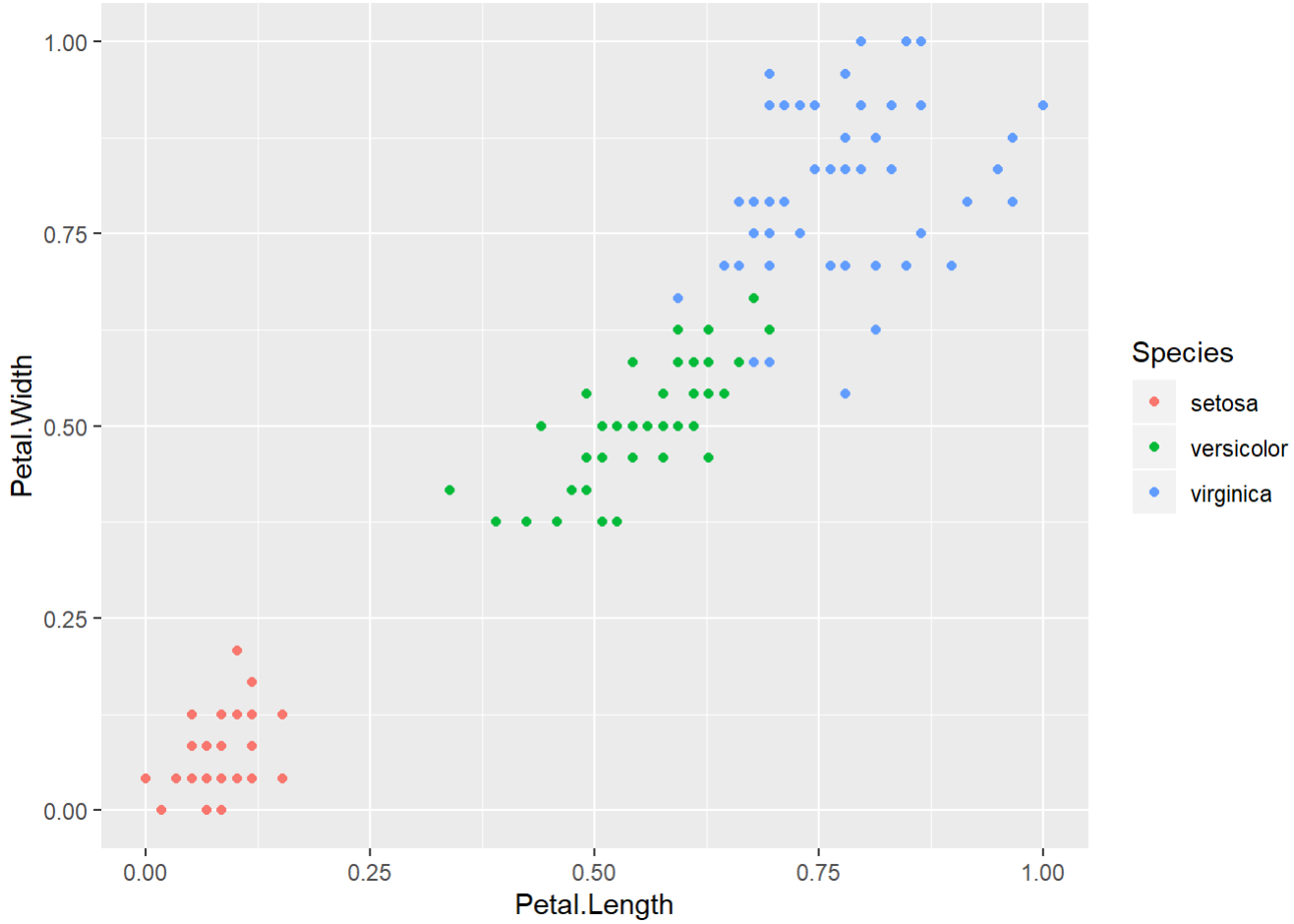
```
library(factoextra)
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

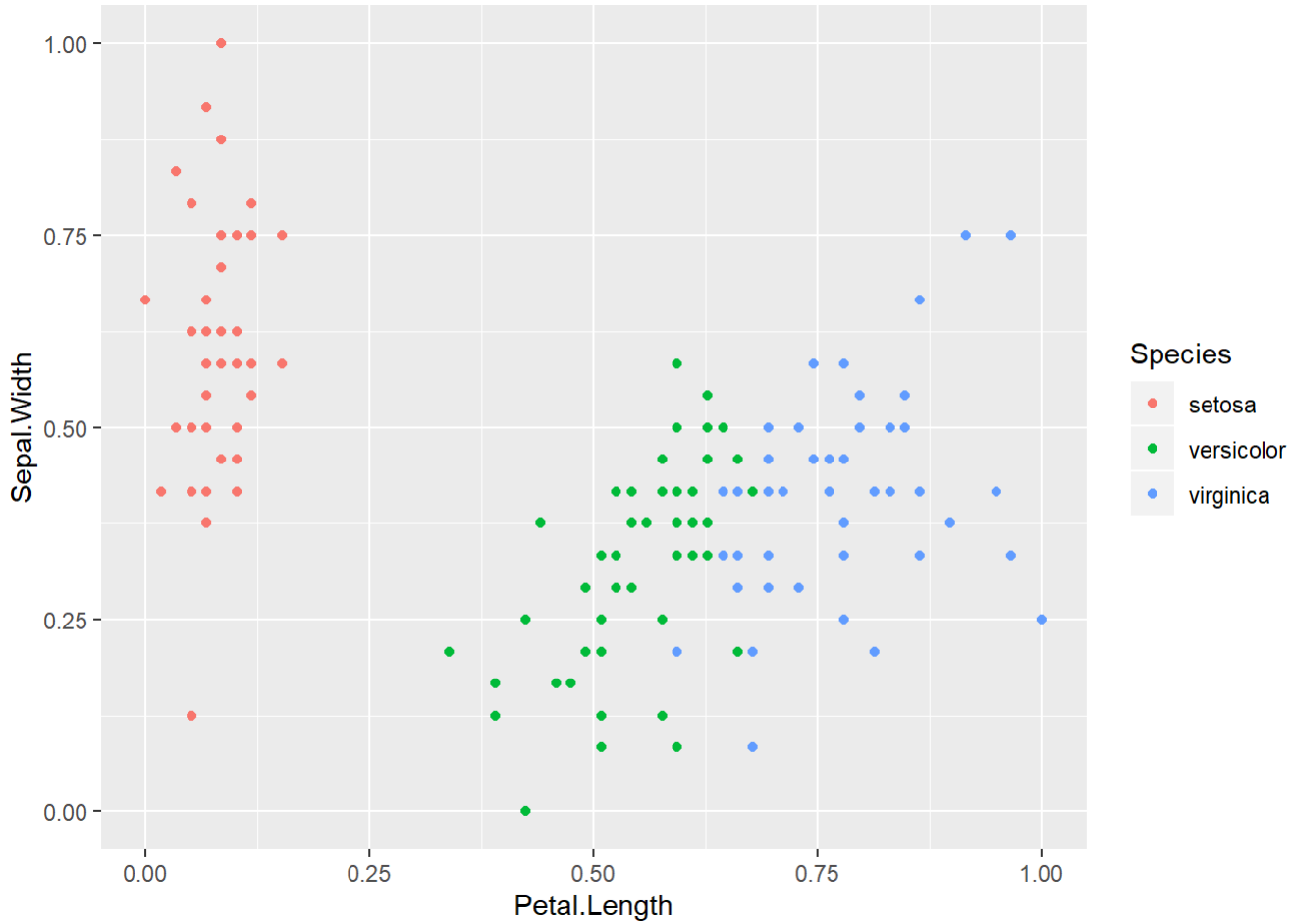
```
#Run the loop to retrieve SCALED Data
iris_scaled_df <- iris
for (i in 1:4) { iris_scaled_df[,i] <- (iris[,i]-min(iris[,i]))/(max(iris[,i])-min(iris[,i])) }

#Identify which predictors are best suited for further analysis.
ggplot(iris_scaled_df,aes(x = Sepal.Length, y = Sepal.Width, col= Species)) + geom_point()
```



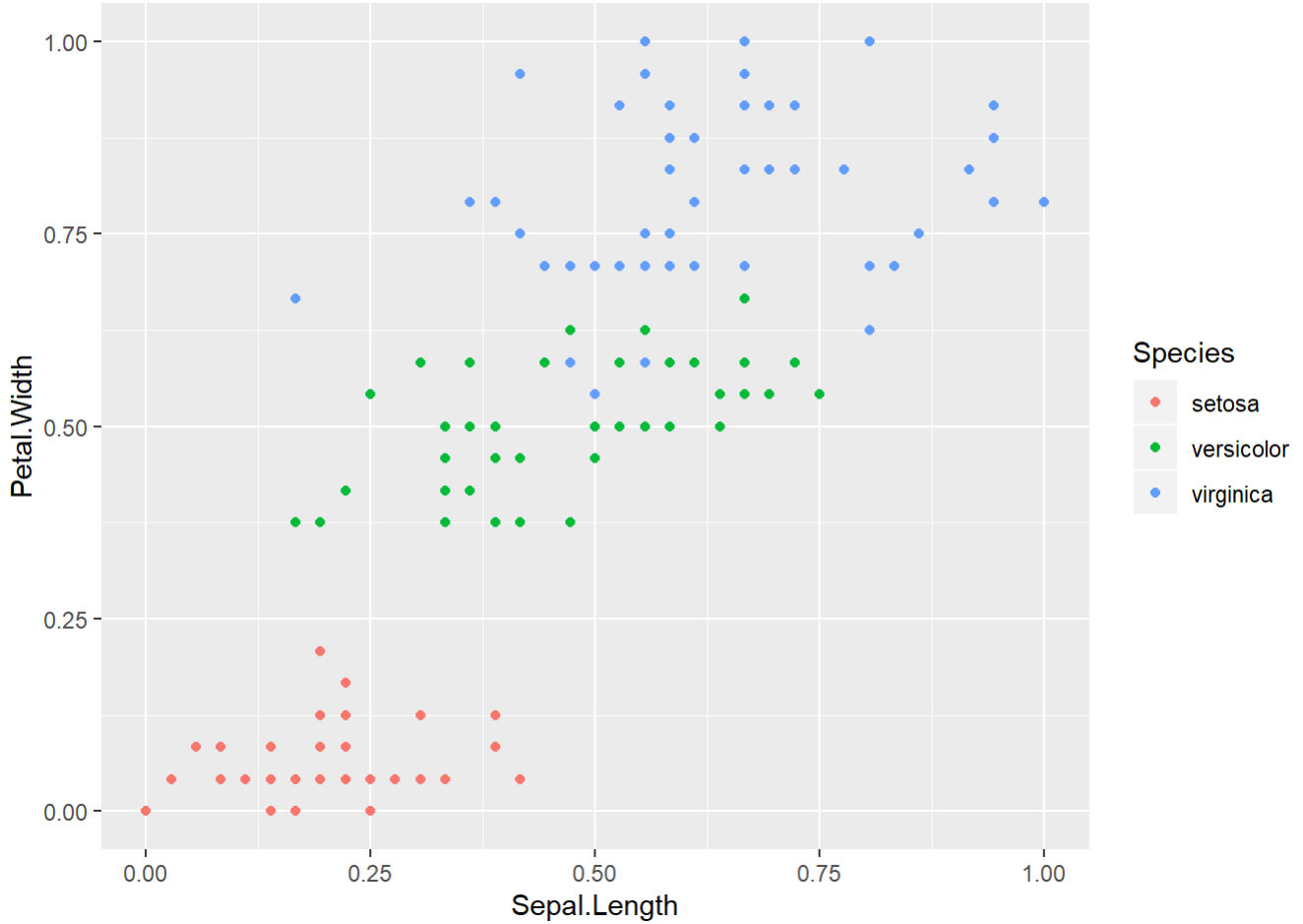

```
ggplot(iris_scaled_df,aes(x = Petal.Length, y = Petal.Width, col= Species)) + geom_point()
```




```
ggplot(iris_scaled_df,aes(x = Petal.Length, y = Sepal.Width, col= Species)) + geom_point()
```




```
ggplot(iris_scaled_df,aes(x = Sepal.Length, y = Petal.Width, col= Species)) + geom_point()
```




```
#Best combination of predictors (Petal.Length and Width)
#Based on the charts above, it's apparent that the predictors (Petal.Length and Width) are more distinguishable to identify
 the flower type. So we will run further analysis based on these columns (,3:4).
#However, if this was Unsupervised learning, I wouldn't have the response values (type of flower) and I would instead run fu
rther analysis based on the predictors that provided the smallest total.withinss
center=5 #I ran this with multiple center values and columns (,3:4) always came out the smallest
kmeans(iris_scaled_df[,1:2],nstart = 20,iter.max = 20, centers=center)$tot.withinss
```

```
## [1] 2.558137
```

```
kmeans(iris_scaled_df[,1:3],nstart = 20,iter.max = 20, centers=center)$tot.withinss
```

```
## [1] 3.368577
```

```
kmeans(iris_scaled_df[,1:4],nstart = 20,iter.max = 20, centers=center)$tot.withinss
```

```
## [1] 4.580323
```

```
kmeans(iris_scaled_df[,2:1],nstart = 20,iter.max = 20, centers=center)$tot.withinss
```

```
## [1] 2.554317
```

```
kmeans(iris_scaled_df[,2:3],nstart = 20,iter.max = 20, centers=center)$tot.withinss
```

```
## [1] 1.944901
```

```
kmeans(iris_scaled_df[,2:4],nstart = 20,iter.max = 20, centers=center)$tot.withinss
```

```
## [1] 2.885804
```

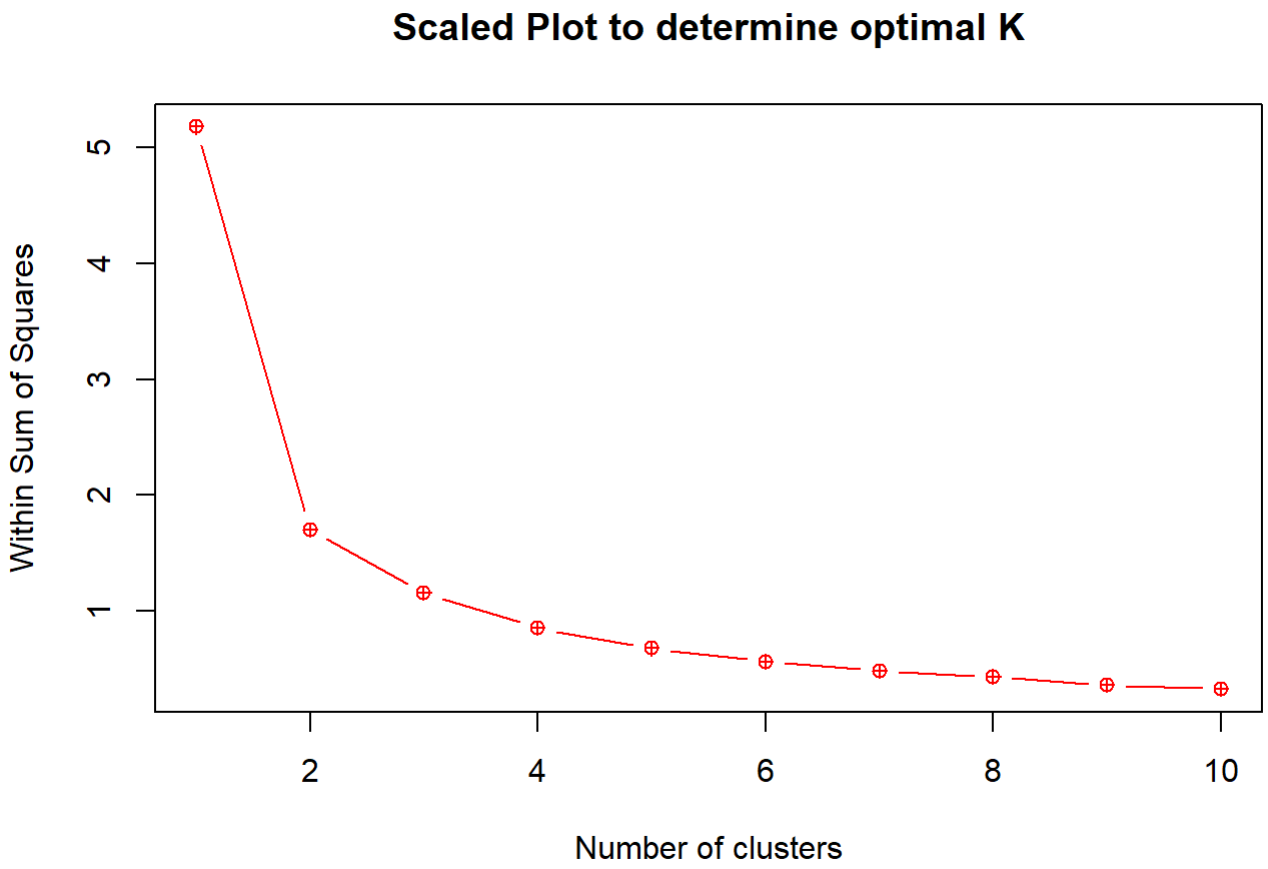
```
kmeans(iris_scaled_df[,3:4],nstart = 20,iter.max = 20, centers=center)$tot.withinss #Lowest tot.withinss so I will deep-dive into this kmeans for further analysis.(Same conclusion as analyzing the ggplots)
```

```
## [1] 0.8535683
```

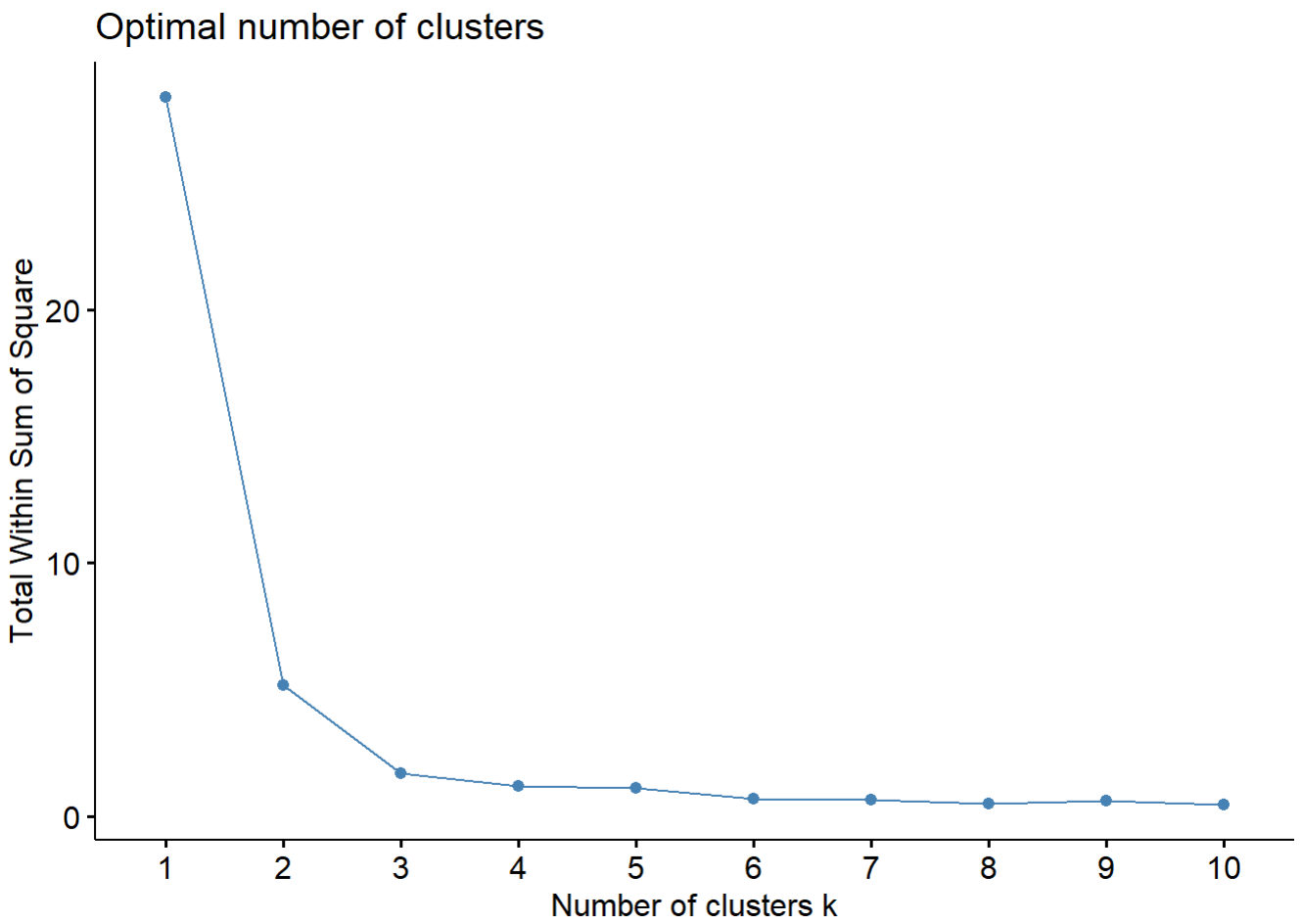
```
#The tot.withinss predicts the number of k required. So by looping through different k values, I constructed the Elbow plot
 to identify the kink in the curve for the most optimal k value based on the "Total Within Sum of Squares" (tot.withinss).
#Elbow Plot preparation
scaled_cut<- rep(0,10)
for (k in 1:10)
{
  scaled_cut[k] <- kmeans(iris_scaled_df[,3:4],nstart = 20,iter.max = 20, centers=k+1)$tot.withinss
}
#tot.withinss sum of squares results
scaled_cut
```

```
##  [1] 5.1764636 1.7018747 1.1588793 0.8535683 0.6795298 0.5651273 0.4858085
##  [8] 0.4362326 0.3636909 0.3268504
```

```
#ggplot
plot(1:10, scaled_cut,type= "b", xlab = "Number of clusters",ylab="Within Sum of Squares", col="red", pch=10, main ="Scaled
 Plot to determine optimal K")
```




```
# It is initially hard to see the optimal k value using the ggplot but using fviz_nbclust makes it more distinguishable base
d on the same method of "within cluster sums of squares" (wss).
fviz_nbclust(x = iris_scaled_df[,3:4],FUNcluster = kmeans, method = 'wss' )
```



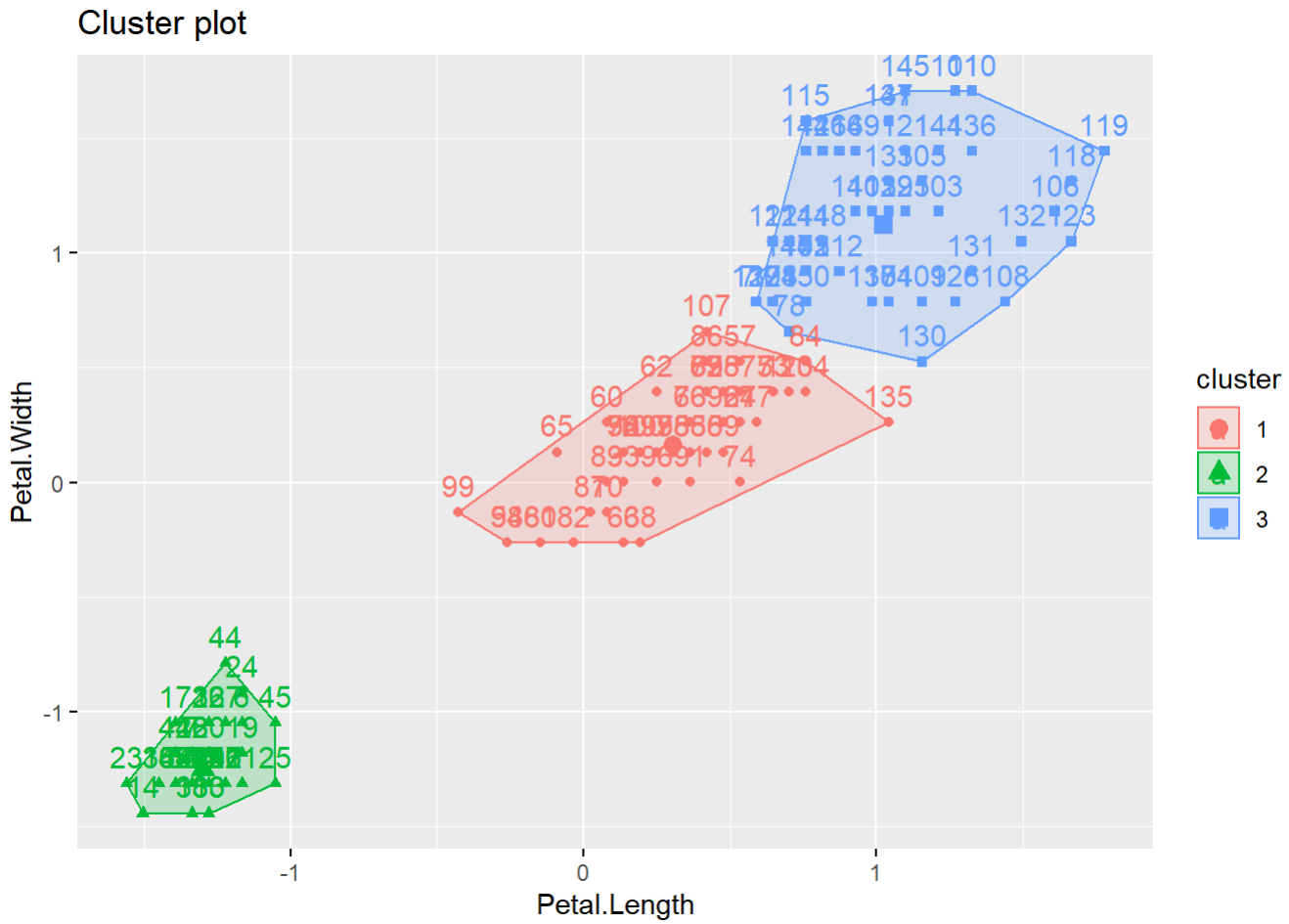

```
# The most optimal k shown above was 3 so now i'm using 3 centers to construct my kmeans model.
model <- kmeans(iris_scaled_df[,3:4],3,nstart = 20)
table(model$cluster,iris_scaled_df$Species)
```

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

```
model
```

```
## K-means clustering with 3 clusters of sizes 52, 50, 48
##
## Cluster means:
##   Petal.Length Petal.Width
## 1   0.55867014  0.51041667
## 2   0.07830508  0.06083333
## 3   0.77401130  0.81510417
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   2   2   2   2   2   2   2   2   2   2   1   1   1   1   1   1   1   1   1   1
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##   1   1   1   1   1   1   1   1   1   1   3   1   1   1   1   1   1   3   1   1
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
##   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   3   3   3   3   3   3   1   3   3   3   3   3   3   3   3   3   3   3   3   1
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
##   3   3   3   3   3   3   3   3   3   3   3   3   3   1   1   3   3   3   3   3
## 141 142 143 144 145 146 147 148 149 150
##   3   3   3   3   3   3   3   3   3   3
##
## Within cluster sum of squares by cluster:
## [1] 0.6791299 0.1369325 0.8858123
##  (between_SS / total_SS =  94.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
fviz_cluster(model, data = iris_scaled_df[,3:4])
```




```
#Analysis of Pedal.Length and Pedal. Width Clustering
# (Look at the table values for the numbers below)
# Total Correct points: 50 + 48 + 46= 144
# Total Incorrect points: 2 from versicolor and 4 from virginica = 6
# The model therefore shows a Percentage Accuracy = between_SS / total_SS is 94% and therefore the model is pretty accurate
 and indicates a good fit.

#Thank you for taking the time to read over my HW2.
```