HW2_WK2

Anon Skywalker

8/31/2020

Question 3.1.a

For the first question, we utilize the credit card data from before to perform cross validation while using the KNN model. However, the train.kknn function is a great fit for a basic take on cross validation, and is what I have used for part A of the first question. In the code below, I utilized the sample function in R to sample out a quarter of the credit card data. I then use this sample to store the removed quarter of the data into a data frame named test.data and the remaining data into train.data, as the train data should contain the majority of the data set. This represents a 75:25 split between train and test data.

By using the train.kknn model, it becomes easy to set a max value of neighbors for K as well as test all optional kernels against the train data. This provided a value of *43 for K-Neighbors and a best kernel choice of triangular. The achieved accuracy was 86.58%*. The only downside to this model is that it is locked in terms of how it cross validates the data. Train.KKNN is stuck with using LOOCV (Leave One Out Cross Validation), where a single point of the data is pulled out and then tested against the other points before replacing that point and moving to the next point (or row).The remaining segments of code are used for comparing the train.kknn model against the test.data set that was sampled out earlier in order to get the accuracy of the model.

```
   #set seed for randomness
   set.seed(1)

   #selects 1/4 of the data points as a sample
   sample.data <- sample(1:nrow(data), size = round(nrow(data) / 4), replace = FALSE)
   # 3/4 of the data
   train.data <- data[-sample.data, ]
   # 1/4 of the data
   test.data <- data[sample.data, ]


   #train.kknn Utilizies LOOCV to find the best value of K.
   knn.train.model <- train.kknn(R1~., data = train.data, kmax = 100, kernel = c("optimal", "epa
nechnikov" ,"inv",
                                                                                  "cos", "gaussia
n", "rank","triangular",
                                                                                  "rectangular",
"biweight"), scale = TRUE)

   knn.train.pred <- predict(knn.train.model, test.data)

   pred.round <- round(knn.train.pred)
   #compare that value with the result column from the data for ith point.
   knn.train.accuracy <- pred.round == test.data[, 11]
   #sum the knn.success values up for later comparison
   accuracy.knn  <- sum(knn.train.accuracy) / nrow(test.data)
```

Question 3.1.b

For the second part of the first question, the challange of using k-fold cross validation becomes a real threat. To make this happen, I divided up the credit data into 4 folds while using sample to sample out 10% of the original data set before creating the folds. K-Fold validation is great, because it allows us to alternate the test and valid portions of the data so that all of the data gets used against the model to get a better representation of how good the model is!

The next trick was to use a couple for loops to test the folds out for each value of k-neighbors. This helped me find that the best value of K was

1. "The best K value is: 30. The highest accuracy is: 84.19%. using kerel triangular.
2. "The best K value is: 24. The highest accuracy is: 83.68%. using kernel rectangular.
3. "The best K value is: 7. The highest accuracy is: 85.72%. using kernel inv.
4. "The best K value is: 7. The highest accuracy is: 85.72% using kernel cos.
5. "The best K value is: 7. The highest accuracy is: 85.72%. using kernel rank.
6. "The best K value is: 7. The highest accuracy is: 85.72%. using kernel biweight.
7. "The best K value is: 79. The highest accuracy is: 85.21% using kernel epanechnikov.
8. *"The best K value is: 7. The highest accuracy is: 85.89% using kernel gaussian.*
9. "The best K value is: 26. The highest accuracy is: 83.68%. using kernel optimal

After running through all the kernel possibilities using the k-folds, it was then time to take the test data that was sampled out and put it up against the best model and k-value. In this case, that was a k of 7 using 4 folds of data with kernel gaussian. *This ended up being a final accuracy of 86.15%!*

```
sample.data <- sample(1:nrow(data), size = round(nrow(data) / 10), replace = FALSE)
   # 9/10 of the data, ALL BUT TEST
   BigTrain.data <- data[-sample.data, ]


   test.data <- data[sample.data, ]


   A <- BigTrain.data[1: round(nrow(BigTrain.data)/4), ]

   B <- BigTrain.data[(round(nrow(BigTrain.data)/4) + 1) : round(nrow(BigTrain.data)/2), ]

   C <- BigTrain.data[(round(nrow(BigTrain.data)/2) + 1): round(nrow(BigTrain.data) * 0.75), ]

   D <- BigTrain.data[(round(nrow(BigTrain.data) * 0.75) + 1): nrow(BigTrain.data), ]


   numKFolds <- 4
   avgAccuracyk.list <- c()
   accuracyK.list <- c()

   for (kneigh in 1:100) {

      for (i in 1:numKFolds) {

         if (i == 1) {
            Train.Data <- rbind(B,C,D)
            Valid.Data <- A
         }

         else if (i == 2) {
            Train.Data <- rbind(A,C,D)
            Valid.Data <- B
         }

         else if (i == 3) {
            Train.Data <- rbind(A,B,D)
            Valid.Data <- C
         }

         else if (i == 4) {
            Train.Data <- rbind(A,B,C)
            Valid.Data <- D
         }

         KNN.Model <- kknn(R1~., Train.Data , Valid.Data , k = kneigh, kernel = "optimal",  scal
e=TRUE)
         KNN.pred <- as.integer(predict(KNN.Model) + 0.5)
         KNN.Success <- KNN.pred == Valid.Data[, 11]
         avgAccuracyk.list[i] <- sum(KNN.Success) / nrow(Valid.Data)
      }

      accuracyK.list[kneigh] <- sum(avgAccuracyk.list)/4 ## avg of all avgAccuracyk.list[]
```

```
}

max.accuracy <- max(accuracyK.list) # max accuracy
best.K <- which.max(accuracyK.list)
result <- paste("The best K value is:  ", best.K, ". ",
                "The highest accuracy is: ",  round(max.accuracy * 100, digits = 2), "%. ",
                 sep = "")

#test the best model against the test.data that was sampled out earlier!
KNN.Model <- kknn(R1~., BigTrain.data, test.data , k = 7, kernel = "gaussian",  scale=TRUE)
KNN.pred <- as.integer(predict(KNN.Model) + 0.5)
KNN.Success <- KNN.pred == test.data[, 11]
final.accuracy <- sum(KNN.Success) / nrow(test.data)
```

Question 4.1

As a clinical data analyst, one area that unspervised learning techniques such as clustering could greatly help in prediction is with diabetes.Being able to predict someones liklieness of diabetes is not an easy task, but by using clustering, we could try to find some risk-factor attributes that could assist in clustering off primary care patients by whether or not they are likely to have diabetes in their life or not.

Some of the risk-factor attributes that would greatly help in clustering patients correctly are: 1. BMI 2. Heredity 3. age (over 35) 4. family history (presence of diabetes) 5. High blood pressure

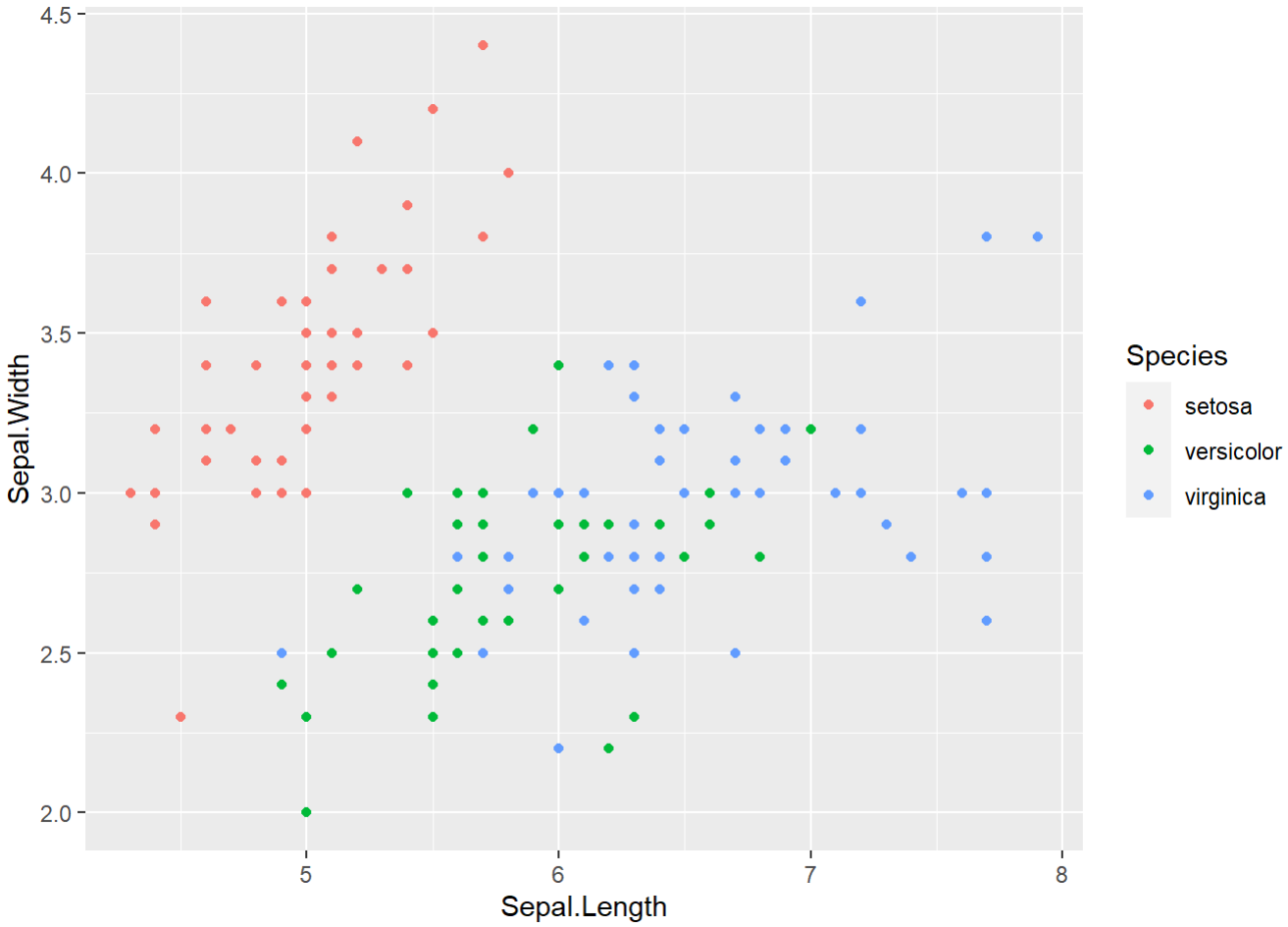
Question 4.2

For this question, we are using the IRIS data set of which contains 3 different species of flowers that are composed of the 4 predictors; sepal length/width, petal length/width. With the goal in mind that we want to use KMeans algorithm for the clustering of the data, it is important that we assess what the data set contains visually. To do this, we can utilize the ggplot library in order to plot out the data. However, we quickly run into an issue due to the number of dimensions. Yet, we can still plot petal length against petal width, and sepal width against sepal length to get a good visual of a portion of the data. Looking at the later two predictors plotted out, we can see that the data is not as easily clustered as it is in the petal length vs width plot (assuming that 3 clusters is the best way to cluster the data).

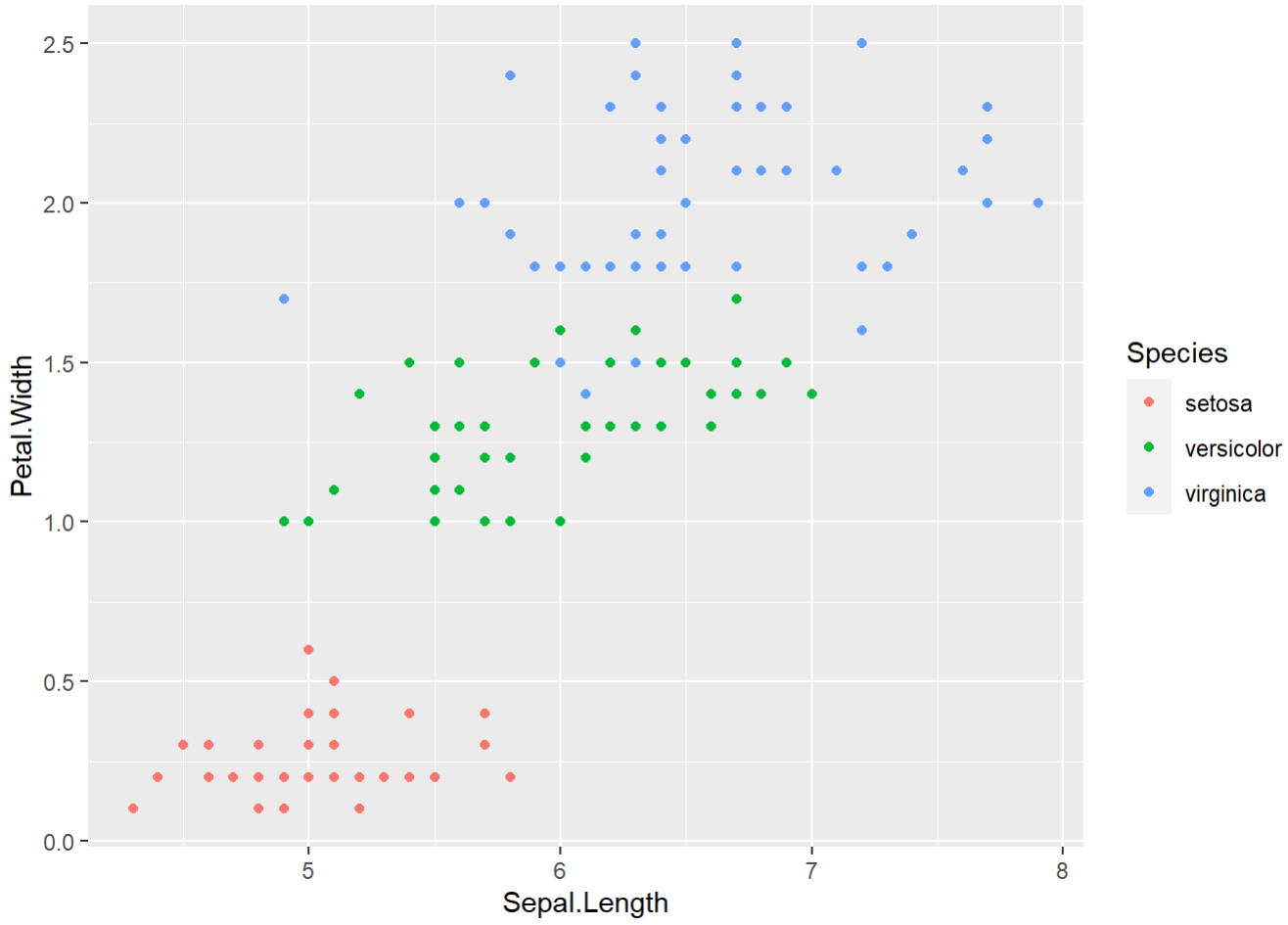
As there are 3 species of flowers, it is clear that the best way to cluster the data and is to find the best way to divide up the points into 3 clusters in order to create a model with kmeans that will accurately classify the different species of flowers. However, the way that we choose to use the data can be useful in getting a better cluster accuracy. For instance, the first ggplot shows sepal length against sepal width, in which the points are overlapped and would be difficult for the model to cluster. After testing multiple different breakdowns of the data, *the best set is petal length against petal width for use with the kmeans model. This produced an accuracy of 94%!*

In order to validate the best number of clusters for this data set, we can utilize fviz_cluster. This library provides a nice illustration of the clusters in a elbow plot that makes it easy to visualize the answer.

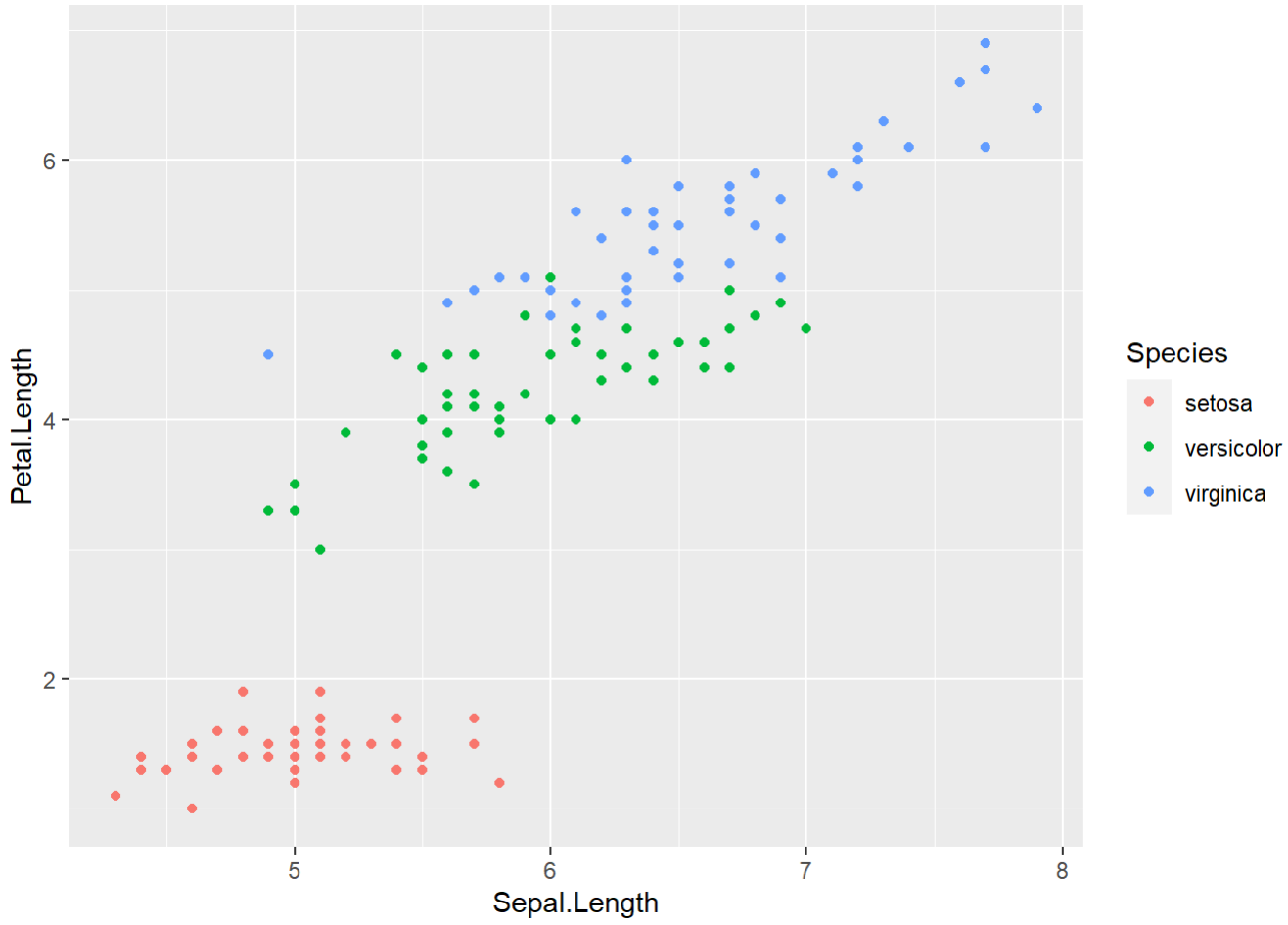
```
#more grouped
ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species)) + geom_point() #semi defined cl
usters
```



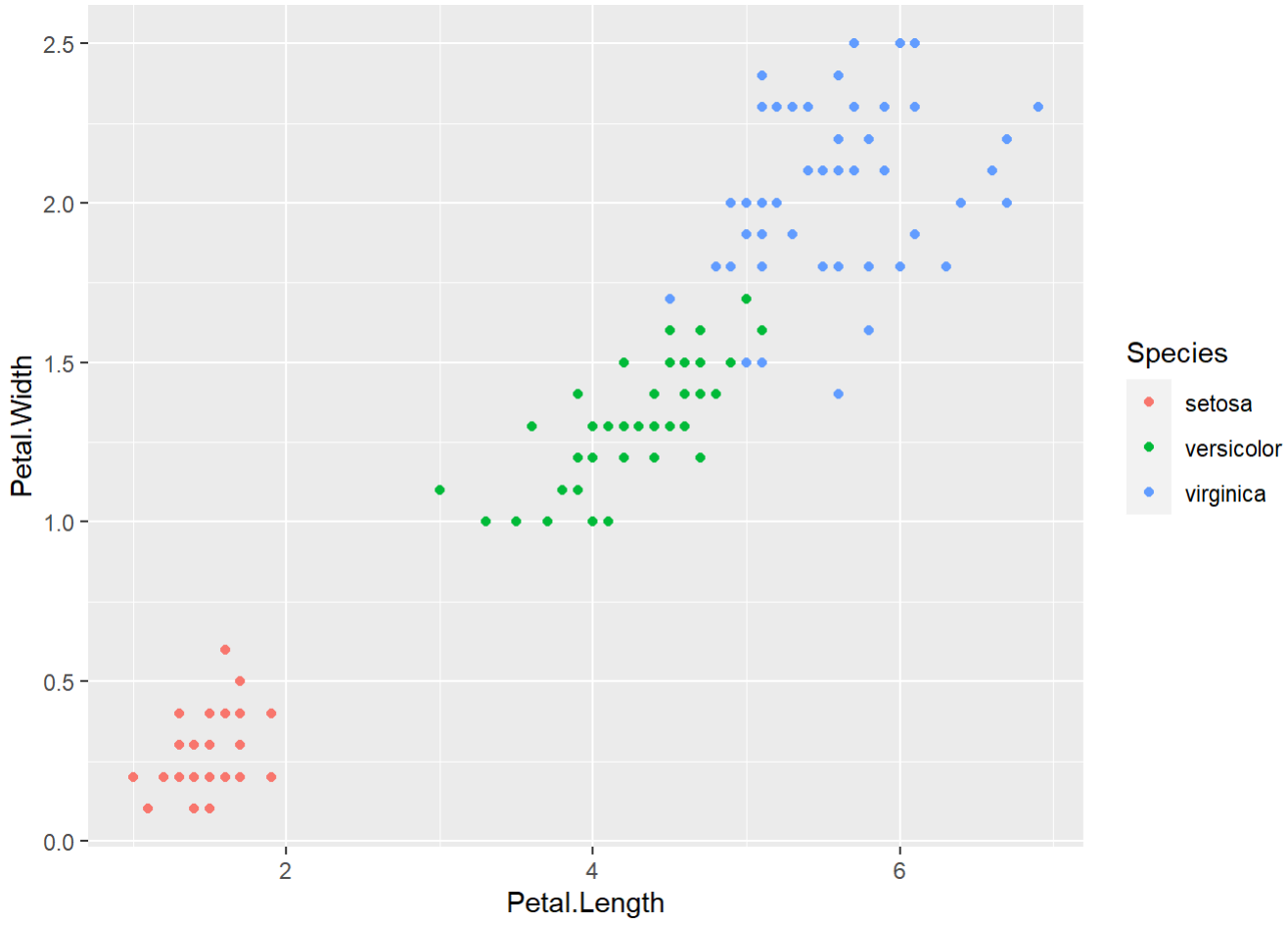

```
ggplot(iris, aes(Sepal.Length, Petal.Width, color = Species)) + geom_point()
```



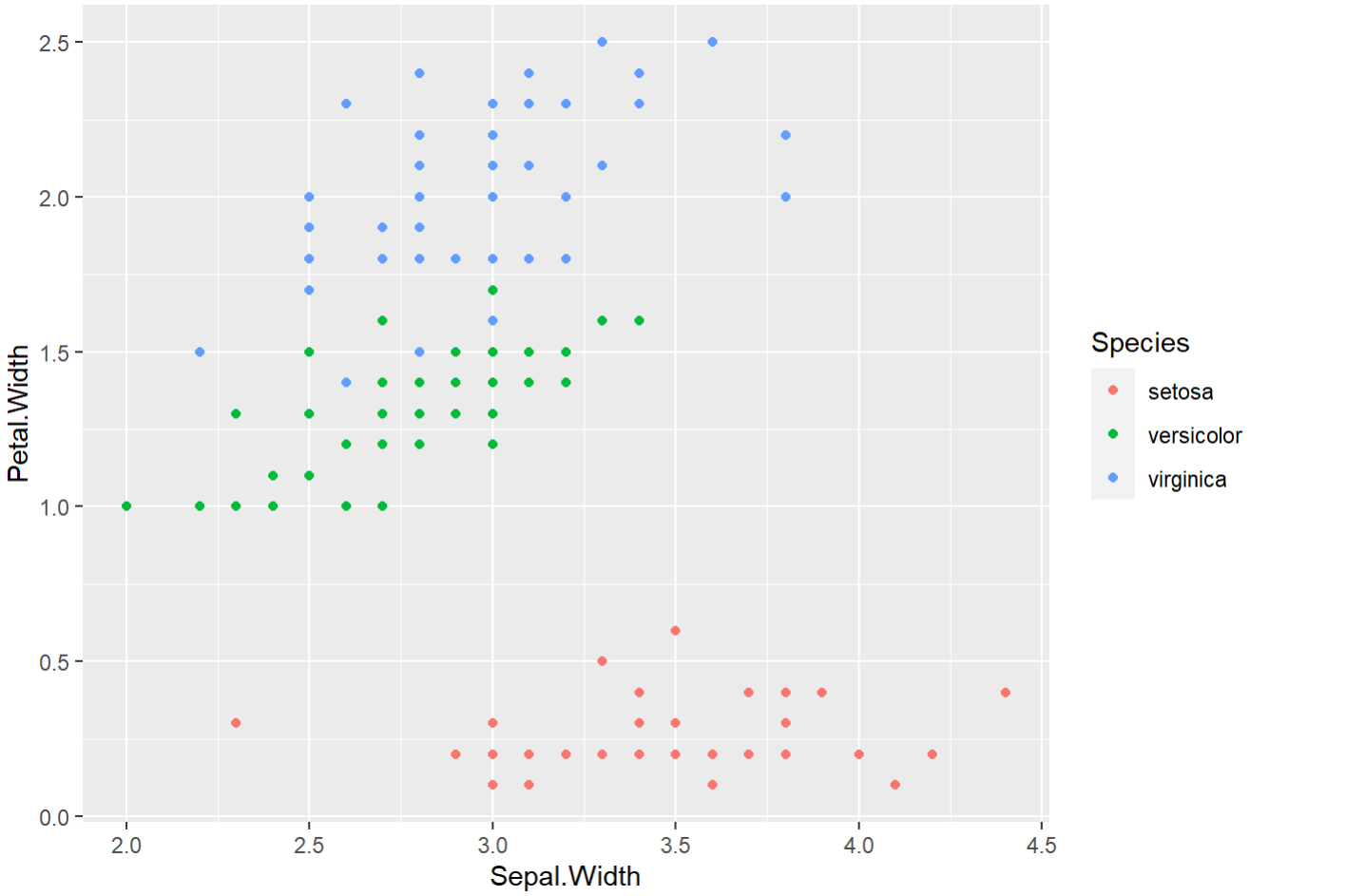

```
ggplot(iris, aes(Sepal.Length, Petal.Length, color = Species)) + geom_point()
```




```
   ggplot(iris, aes(Petal.Length, Petal.Width, color = Species)) + geom_point() #very defined cl
usters!
```




```
ggplot(iris, aes(Sepal.Width, Petal.Width, color = Species)) + geom_point()
```




```
#removes results column from data and scales the data
data.iris <- iris[, -5]

sepalL.PetalL <- data.iris[, -c(2,4)]

sepalW.PetalW <- data.iris[, c(2, 4)]

sepalW.PetalL <- data.iris[, c(2,3)]

petal.iris <- scale(data.iris[, 3:4])

sepal.iris <- scale(data.iris[, 1:2])


kmeans.petal <- kmeans(petal.iris, 3, nstart = 20)
kmeans.sepal <- kmeans(sepal.iris, 3, nstart = 20)
kmeans.sepalL.PetalL <- kmeans(sepalL.PetalL, 3, nstart = 20)
kmeans.sepalW.PetalW <- kmeans(sepalW.PetalW, 3, nstart = 20)
kmeans.sepalW.PetalL<- kmeans(sepalW.PetalL, 3, nstart = 20)


kmeans.table <- table(kmeans.petal$cluster, iris$Species)


paste("accuracy: ", round(((kmeans.petal$betweenss / kmeans.petal$totss) * 100), digits = 2),
"%", sep = "")
```

```
## [1] "accuracy: 93.99%"
```

```
  #elbow plot for best number of clusters!
  fviz_nbclust(data.iris, kmeans, method = "wss") + geom_vline(xintercept = 3, linetype = 2)
```