Cluster Analysis - Iris Dataset

C. Perez

Project Description

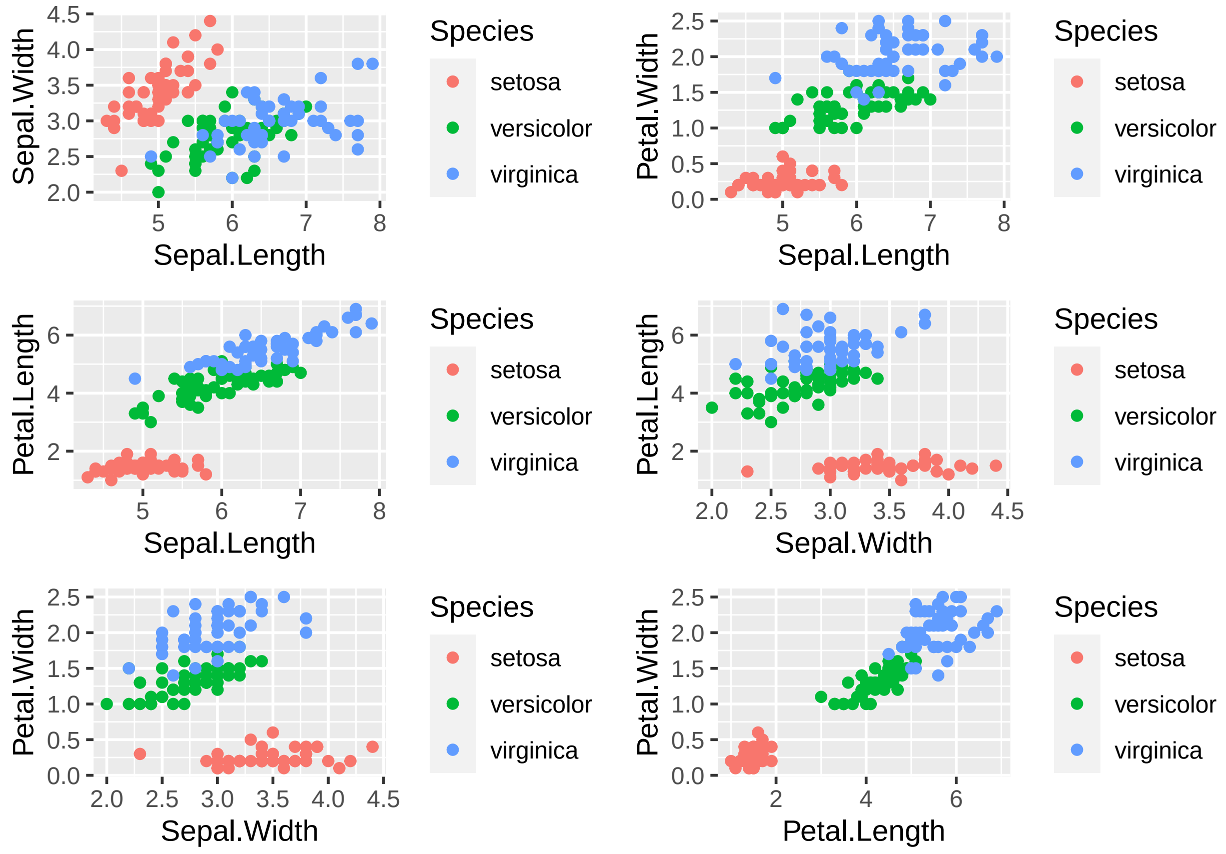
This can be considered a supervised learning project with a commonly used dataset (iris) to explore distinctions between three techniques, **hierarchical clustering**, **kmeans** and **density based spatial cluster applications w/ Noise (DBSCAN)**. Although this is not supervised learning in the traditional sense (i.e. building a model or classification algorithm) this project explores applying multiple methods in a supervised setting as the data has the correct cluster classification available (species) to check the results against. This is important for identifying the effectiveness of say "single linkage" vs "complete linkage", or subtle disadvantages between methods. This specific dataset is used as there are only quantitative variables present and thus removes the trouble of creating a proper distance or dissimilarity matrix for mixed datasets (**explored in later projects for more interesting data**). All code will be included.

Data Discussion

The iris dataset is made up 150 observations of 5 variables. The "species" variable shows the group each observation belongs to and will serve as the cluster identifier for later comparison. Typically this type of analysis is performed on large datasets for which it would not be feasible to plot every pairwise comparison. However, here there are only 6 (4choose2) pairwise comparisons and as such each is plotted below to get an idea of what the results will look like.

```r
# Cluster Analysis on the Iris Dataset

# clear global environment
rm(list = ls(all.names = TRUE))


# libraries----
library(tidyverse)
library(dendextend)
library(factoextra)
library(cluster)
library(gridExtra)
library(knitr)
library(dbscan)


# load data----
attach(iris)
iris_c <- select(iris, -Species)


# Preliminary view of the data
# possibilities 4x3/2 = 6
```

```
c1 <- ggplot(iris, aes(Sepal.Length, Sepal.Width, color = Species))+
  geom_point()

c2 <- ggplot(iris, aes(Sepal.Length, Petal.Width, color = Species))+
  geom_point()

c3 <- ggplot(iris, aes(Sepal.Length, Petal.Length, color = Species))+
  geom_point()

c4 <- ggplot(iris, aes(Sepal.Width, Petal.Length, color = Species))+
  geom_point()

c5 <- ggplot(iris, aes(Sepal.Width, Petal.Width, color = Species))+
  geom_point()

c6 <- ggplot(iris, aes(Petal.Length, Petal.Width, color = Species))+
  geom_point()

# create the grid and plot
grid.arrange(c1,c2,c3,c4,c5,c6, nrow = 3)
```

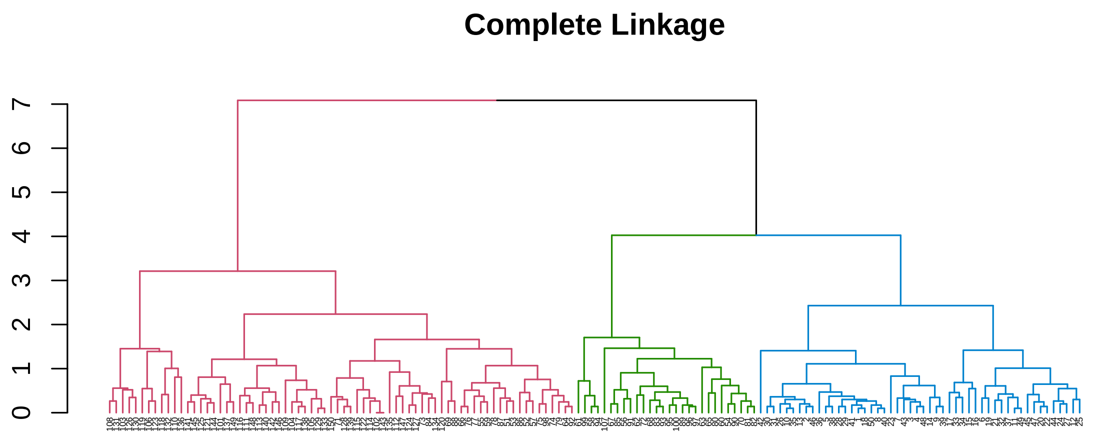


The pairwise plots reveal that one group is easily distinguishable from the other two, while the remaining two groups can easily be mistaken for one group in the common situation where the group is not known beforehand. This leads me to believe that neither method will be able to correctly classify the 150 observations into their corresponding groups, however one might offer increased accuracy.

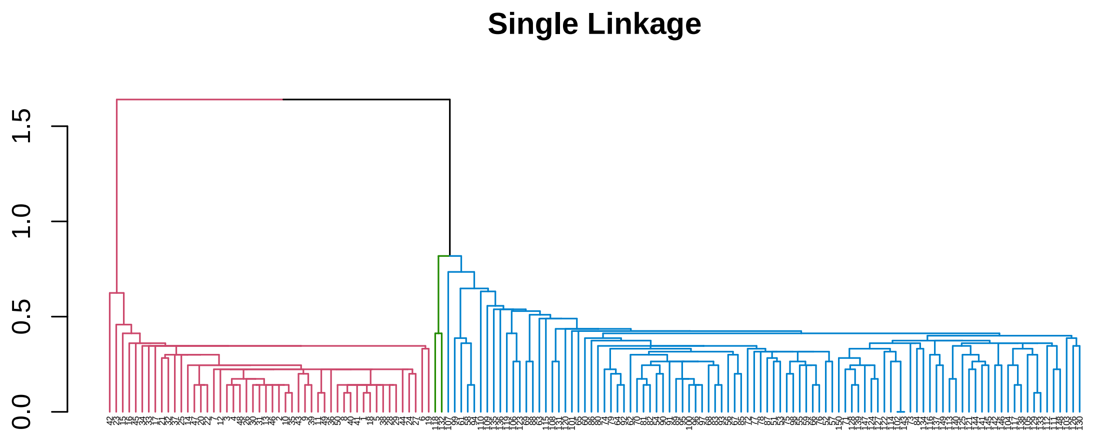
Hierarchical Clustering

The following code implements a hierarchical approach to clustering these observations, and the steps are identified within the code. Three commonly used methods of forming these clusters are **complete linkage**, **single linkage**, and **average linkage**. The method chosen directly relates to the measure of distance used in grouping the observations. Hierarchical clustering starts with calculating the specified distance (euclidean here) in p-dimensional space between all observations and forms initial clusters for observations that share the smallest distance between each other. From there, remaining observations are measured from the initial clusters and classified into the next layer of clusters according to the method chosen. For complete linkage, the maximum distance is used, for single linkage, the minimum distance is used, and for average linkage, the average is used. All three methods were used and the associated dendrogram was created to visually explore how well each method performed.This is provided below:

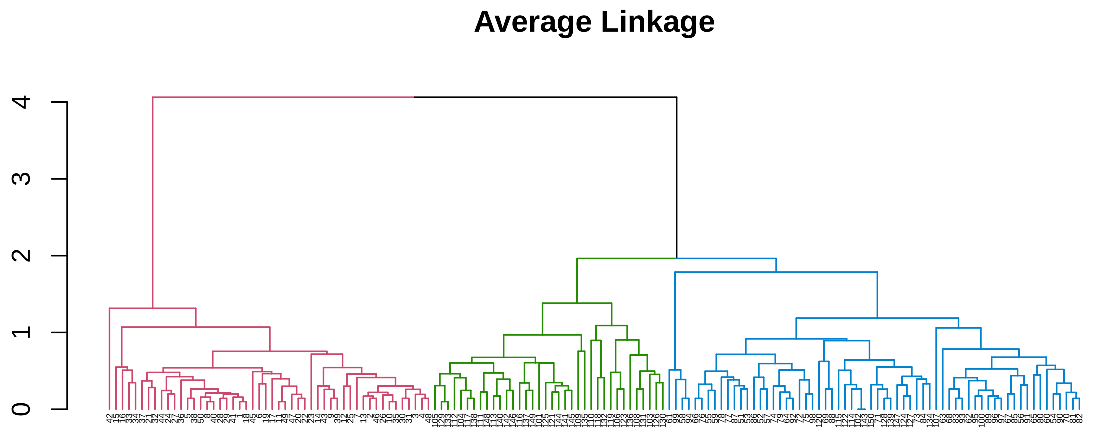
```r
# hierarchical clustering----
# step 1: calculate distance first
dist_iris <- dist(iris_c, method = "euclidean")

# step 2: cluster by 3 main linkage methods
clustobject_c <- hclust(dist_iris, method = "complete")
clustobject_s <- hclust(dist_iris, method = "single")
clustobject_a <- hclust(dist_iris, method = "average")

# step 3: dendrogram coloring and plotting
dend_obj_c <- color_branches(as.dendrogram(clustobject_c), k = 3) %>% #k=3 supervised example
  set("labels_cex",.3) #set text size at node
plot(dend_obj_c, main = "Complete Linkage")
```



```r
dend_obj_s <- color_branches(as.dendrogram(clustobject_s), k = 3) %>% #k=3 supervised example
  set("labels_cex",.3) #set text size at node
plot(dend_obj_s, main = "Single Linkage", cex = .6)
```



```
dend_obj_a <- color_branches(as.dendrogram(clustobject_a), k = 3) %>%  #k=3 supervised example
  set("labels_cex",.3) #set text size at node
plot(dend_obj_a, main = "Average Linkage")
```



Because the data has 50 observations in each group, we would expect the dendrogram to show three clusters with a roughly equivalent quantity of observations. However, based on the clusters present for each method (shown by the coloring based on k=3) it is clear that complete linkage and average linkage do a much better job of identifying 3 clusters with a good amount of observations in each, while single linkage, even colored according to 3 clusters, does not do a great job at all. Based on the results thus far, it would appear that single linkage is not as good as the other two methods when clustering according to k = 3 for the variables present. This can be seen from the following tables.

```r
# step 4: cut trees to receive cluster assignments
clusters_c <- cutree(clustobject_c, k = 3)
clusters_s <- cutree(clustobject_s, k = 3)
clusters_a <- cutree(clustobject_a, k = 3)

# step 5: assign clusters back to data frame and check for inconsistencies
linkage_results_c <- iris %>% mutate(cluster = clusters_c)
linkage_results_s <- iris %>% mutate(cluster = clusters_s)
linkage_results_a <- iris %>% mutate(cluster = clusters_a)

# step 6: check errors in cluster assignment based on methods
table(clusters_c)
```

```
clusters_c
 1  2  3
50 72 28
```

```r
table(clusters_s)
```

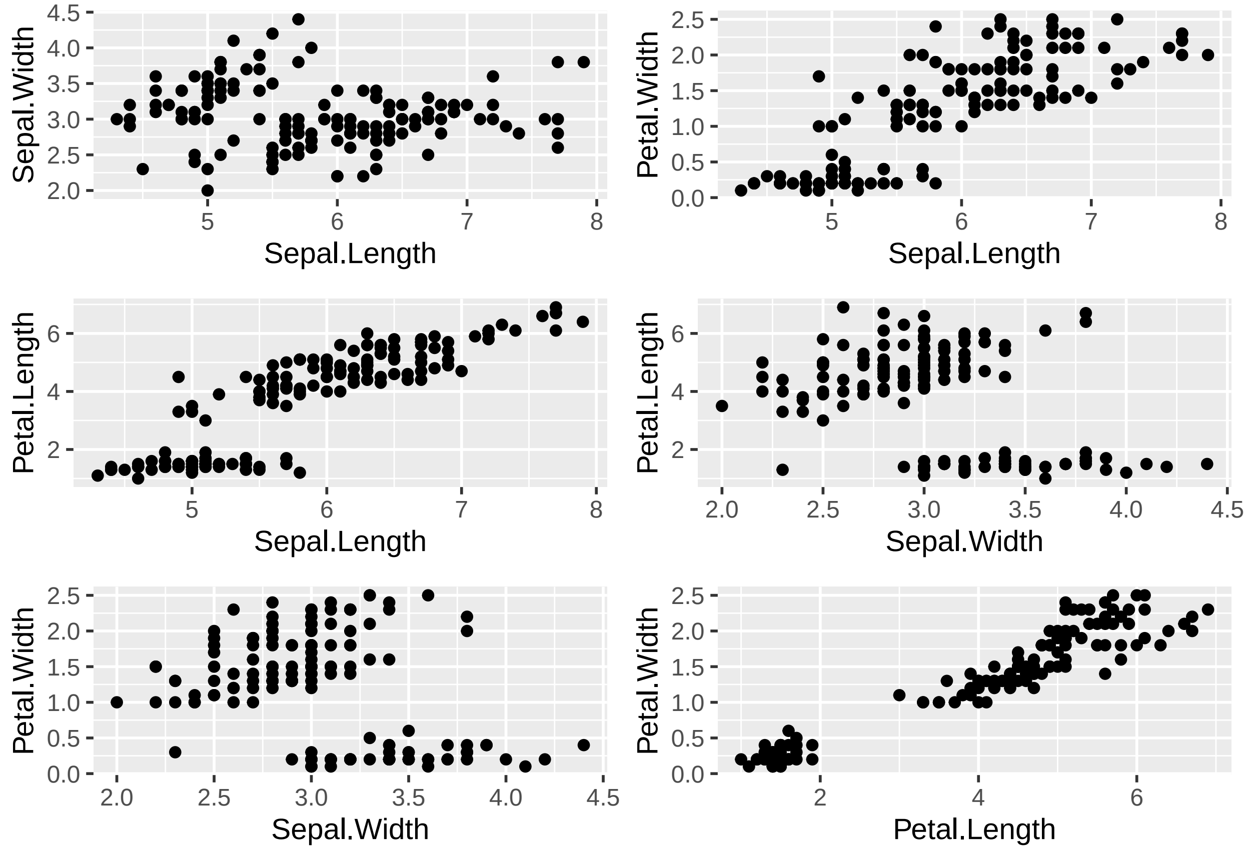
```
clusters_s
 1  2  3
50 98  2
```

```r
table(clusters_a)
```

```
clusters_a
 1  2  3
50 64 36
```

This demonstrates how vital it is to leverage any prior information about the data. Because we know there are three groups that are equivalent in size we can see that single linkage is not performing well in correctly identifying the group (cluster) each observation belongs to. However, if the species (groups) had not been provided, the plots would lead us to believe that two clusters is correct according to each pairwise comparison. This would then lead us to conclude that single linkage is actually outperforming the other 2 methods. It can be seen from the plots below (original pairwise plots with color removed) that there appears to be two distinct groups without prior knowledge of k:

```r
# removing knowledge of k
c11 <- ggplot(iris, aes(Sepal.Length, Sepal.Width))+
  geom_point()

c22 <- ggplot(iris, aes(Sepal.Length, Petal.Width))+
  geom_point()

c33 <- ggplot(iris, aes(Sepal.Length, Petal.Length))+
  geom_point()

c44 <- ggplot(iris, aes(Sepal.Width, Petal.Length))+
  geom_point()

c55 <- ggplot(iris, aes(Sepal.Width, Petal.Width))+
  geom_point()

c66 <- ggplot(iris, aes(Petal.Length, Petal.Width))+
  geom_point()

# create the grid and plot
grid.arrange(c11,c22,c33,c44,c55,c66, nrow = 3)
```



I actually believe that single linkage is still outperforming the other two methods even with prior knowledge of "k" because:

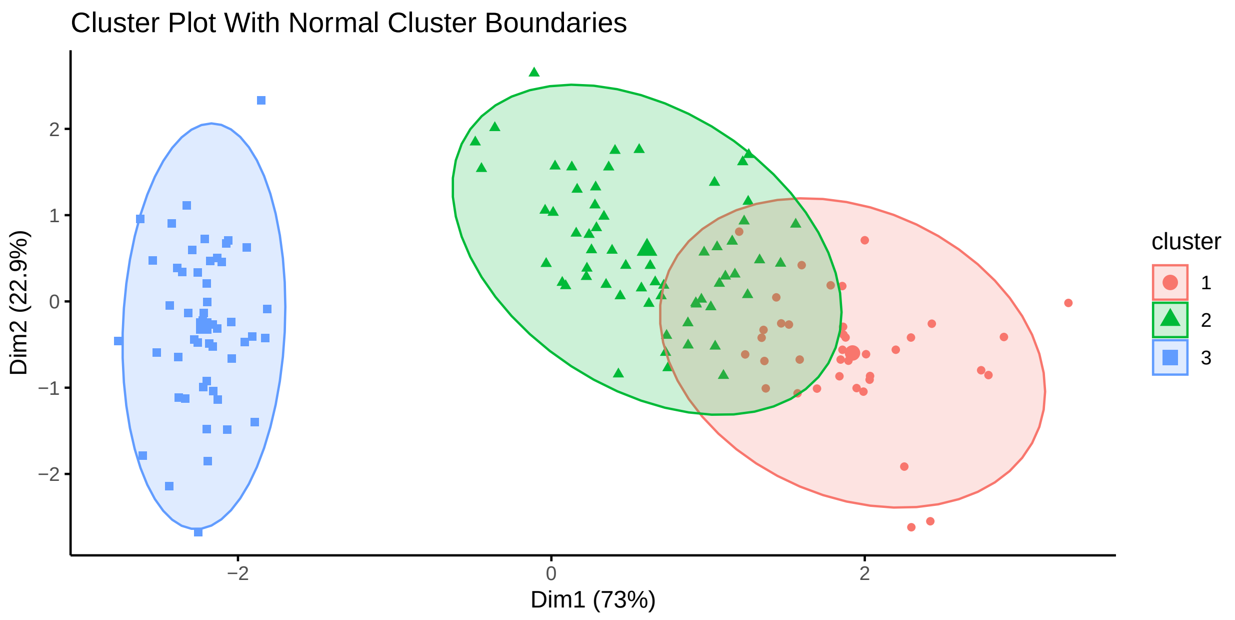
1. The usual method of choosing "k" (the quantity of clusters) from a dendrogram is to cut the dendrogram between cluster layers that show the largest difference in distance. Here that would be between the last layer and the previous 2 which leads to 2 clusters (k = 2).
2. The exploratory pairwise comparison plots reveal two groups.
3. Additional widely used and accepted methods explored later in this project reveal k=2 as the optimal number of clusters as well.

Again, the above 3 reasons are purely a reflection of how well the variables are distinct enough, by group, to identify the number of clusters correctly. For example, if the age, hair color, IQ, and favorite color of 50 men, 50 women, and 50 children were recorded, we can construct a similar dataset to that of the iris dataset, with the groups replaced by the observation belonging to either a man, woman, or child. It would be easy to classify children based on any pairwise comparison including age, however, all 4 variables do nothing to distinguish gender. Even though we know there are 3 distinct groups, a cluster analysis based on the 4 chosen features would probably reveal 2 distinct groups. This shows that the ability for the algorithms to correctly classify observations is directly correlated to the statistical difference of variables chosen by group.

Kmeans Clustering

The following code implements a kmeans approach to clustering these observations, and the steps are identified within the code. The kmeans method requires a specified number of clusters as an argument because the method essentially starts with "k" points as centers, adds observations based on euclidean distance from these centers, then moves the centers via minimizing within sum of squares distance between centers and neighboring observations. This process repeats until all observations are classified within a group and the centers become stable. Because it is not practical to plot pairwise comparisons, Principal Component Analysis (PCA) is used to plot the clusters and the observations belonging to each cluster. The dimensions are the two components that explain a majority of the variance (the significance of these components are explored within PCA analysis projects). The visualization of these clusters can be seen below:

```
# kmeans clustering----
# create a model
model_k <- kmeans(iris_c, centers = 3) #k=3 supervised
error_kmeans <- iris %>% mutate(cluster = model_k$cluster) %>%
  group_by(Species,cluster) %>% summarise(count = n())

# visualization model via PCA for kmeans clustering technique
fviz_cluster(model_k, iris_c, geom = "point", ggtheme = theme_classic(),
             ellipse.type = "norm",
             main = str_to_title("cluster plot with normal cluster boundaries"))
```



The above visualization shows poor separation between groups 2 and 3. This further corroborates the idea that the variables are not statistically different between groups 2 and 3 and hence the algorithm fails to correctly classify them. The error for the kmeans approach can be seen visually via the following table:

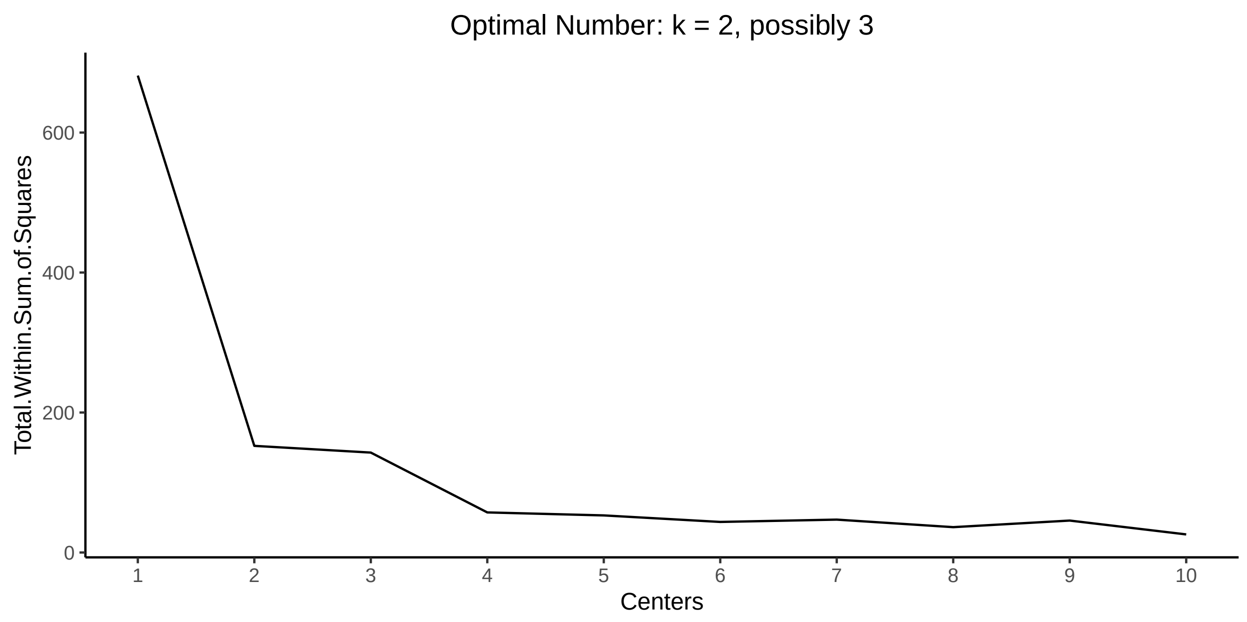
```
# print table
kable(error_kmeans)
```

Species	cluster	count
setosa	3	50
versicolor	1	2
versicolor	2	48
virginica	1	36
virginica	2	14

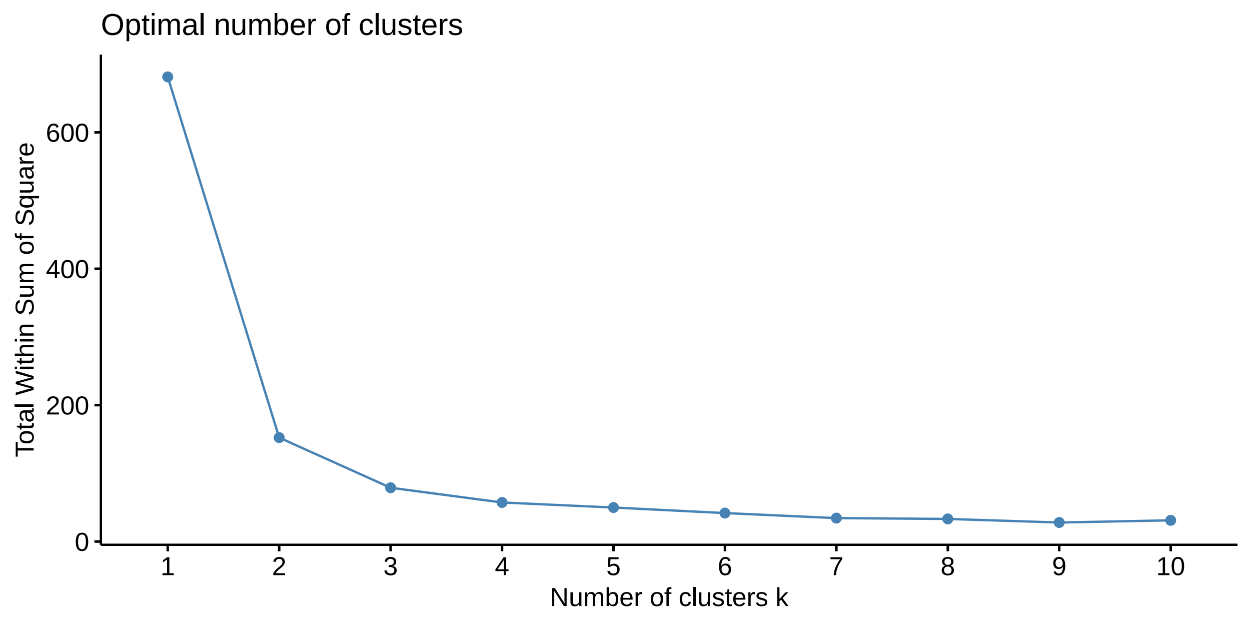
The kmeans approach yields similar results to the average linkage method from hierarchical clustering when using prior information (k =3). When prior knowledge of k is not known, typically two methods are used to determine the optimal number for k. Interestingly enough, both of these methods show 2 clusters as optimal based on the features available. The two methods are as follows:

Elbow Method: The elbow method requires simulation over a specified range of values for k. The total within sum of squares distance is recorded for the stabilized "k" centers and plotted against the associated value of k. Based on where the curve starts to level out (the elbow) is where the optimal value of K should be chosen. The following code leads to this plot, or the use of the fviz_nbclust() function.

```r
# run multiple models for k, create a data frame, check elbow method
# set k values
k_elbow_method <- c(1:10)

# calculate total within ss across k values
total_within_ss <- map_dbl(k_elbow_method,  function(k){
  model <- kmeans(x = iris_c, centers = k)
  model$tot.withinss
})

# create data frame
elbow_method_df <- data.frame("Centers" = k_elbow_method ,
                              "Total Within Sum of Squares" = total_within_ss)

# construct plot
ggplot(data = elbow_method_df, mapping = aes( x= Centers,y = Total.Within.Sum.of.Squares))+
  geom_line()+scale_x_continuous(breaks = 1:10)+theme_classic()+
  labs(title = "Optimal Number: k = 2, possibly 3")+
  theme(plot.title = element_text(hjust = .5))
```

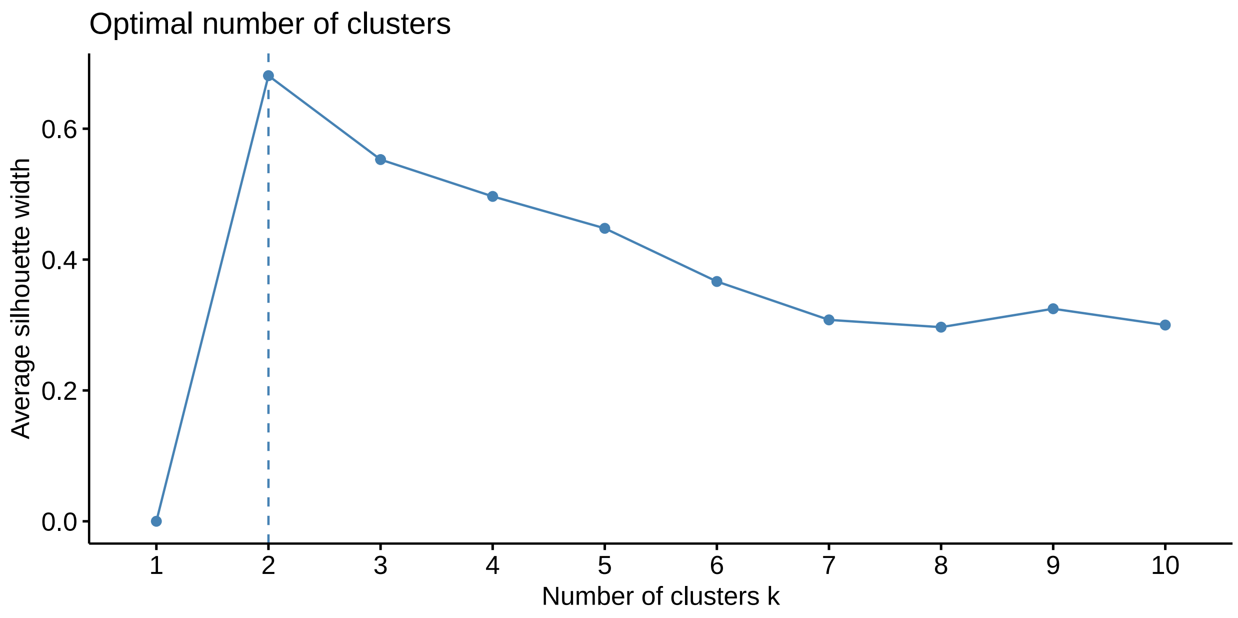


```r
# function plot
fviz_nbclust(iris_c, kmeans, method = "wss")
```



Silhouette Method: The silhouette method also requires simulation over a specified range of values for k (minimum 2). The basic idea is that it captures a ratio of the distances from each observation to the observations of the closest clusters, for each observation. The closer that distance to 1, the better the fit, the closer to 0, the observation lies almost evenly between two clusters. Anything else is evidence the observation belongs to another cluster. The commented code can create the necessary graph however the fviz_nbclust() function is used here for simplicity. Here the optimal number of clusters is where the silhouette width is highest and it is important to note this plot is the average silhouette width of all observations for the specified value of k.

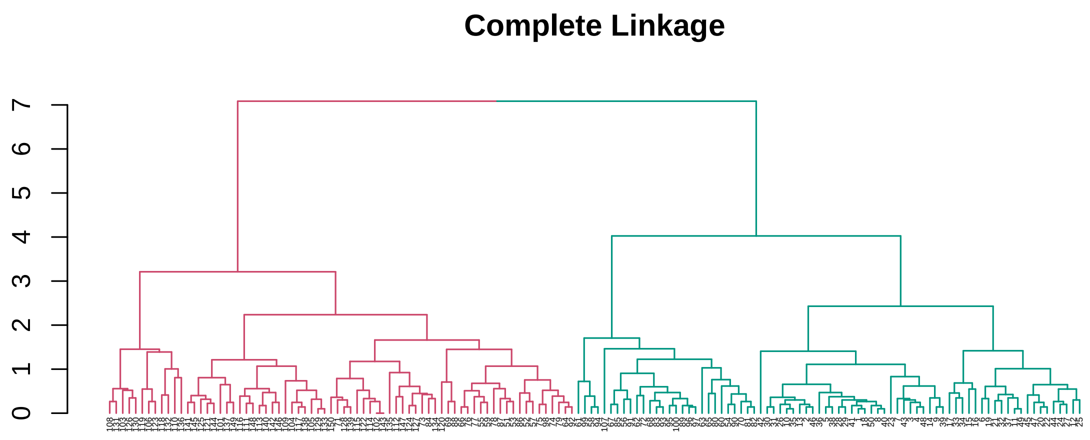
```r
# run multiple models for k, create a data frame, check silhouette method
# # set k values
# k_sil_method <- c(2:10) # need at least 2 clusters to compare
#
# # calculate total within ss across k values
# sil_width <- map_dbl(2:10,  function(k){
#   model <- pam(x = iris_c, k = k) # pam used to pull avg. silhouette width
#   model$silinfo$avg.width
# })
#
# # create data frame
# sil_method_df <- data.frame("Centers" = 2:10,"Average Silhouette Width" = sil_width)
#
# # construct plot
# ggplot(data = sil_method_df, aes(x = Centers, y = Average.Silhouette.Width)) +
#   geom_line() +
#   scale_x_continuous(breaks = 2:10)

# function plot
fviz_nbclust(iris_c, kmeans, method = "silhouette")
```



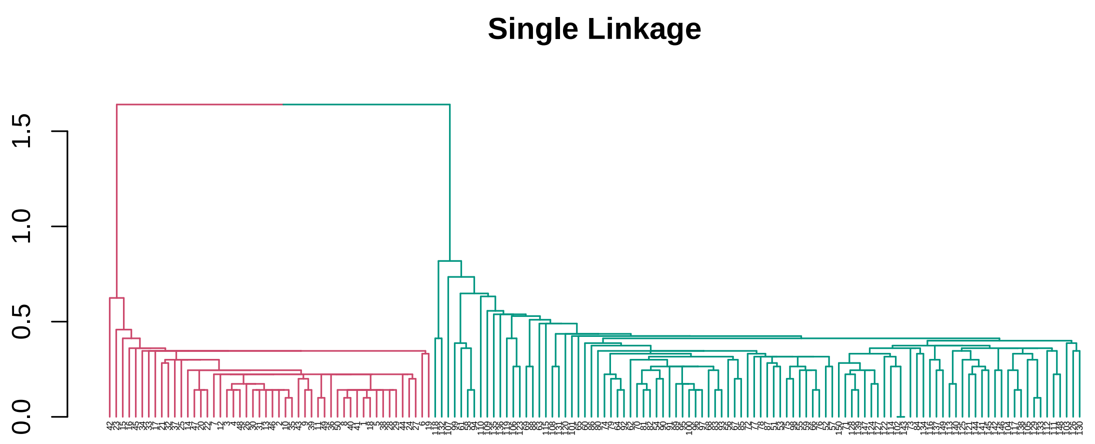
Both methods reveal k=2 as the optimal number of clusters. This is in agreement with the exploratory pairwise plots (without color) shown earlier, and the results provided by single linkage hierarchical clustering (k=2 or k=3). It is important to note that I feel single linkage is outperforming the other two, not because it is a better method, but more so because I feel that single linkage is more resistant to classify in 3 groups when there is not a statistical difference between 2 of those groups for the variables chosen. The following shows the dendrograms colored for k=2 and the table results. It is easy to see that using the optimal k, single and average linkage methods perform very well for a value of k derived from the data, numerically and graphically. This is vital seeing as we typically do not know the number of groups prior to.

```r
# step 1: calculate distance first
dist_iris <- dist(iris_c, method = "euclidean")
```

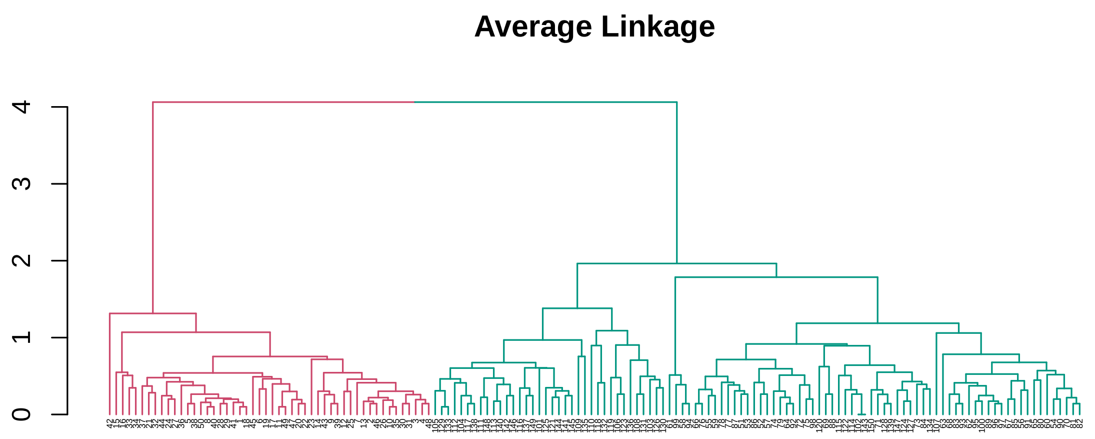
```r
# step 2: cluster by 3 main linkage methods
clustobject_c <- hclust(dist_iris, method = "complete")
clustobject_s <- hclust(dist_iris, method = "single")
clustobject_a <- hclust(dist_iris, method = "average")

# step 3: dendrogram coloring and plotting
dend_obj_c <- color_branches(as.dendrogram(clustobject_c), k = 2) %>% #k=3 supervised example
  set("labels_cex",.3) #set text size at node
plot(dend_obj_c, main = "Complete Linkage")
```



```r
dend_obj_s <- color_branches(as.dendrogram(clustobject_s), k = 2) %>% #k=3 supervised example
  set("labels_cex",.3) #set text size at node
plot(dend_obj_s, main = "Single Linkage", cex = .6)
```



```
dend_obj_a <- color_branches(as.dendrogram(clustobject_a), k = 2) %>%  #k=3 supervised example
  set("labels_cex",.3) #set text size at node
plot(dend_obj_a, main = "Average Linkage")
```



```
# step 4: cut trees to receive cluster assignments
clusters_c <- cutree(clustobject_c, k = 2)
clusters_s <- cutree(clustobject_s, k = 2)
clusters_a <- cutree(clustobject_a, k = 2)

# step 5: assign clusters back to data frame and check for inconsistencies
linkage_results_c <- iris %>% mutate(cluster = clusters_c)
linkage_results_s <- iris %>% mutate(cluster = clusters_s)
```

```r
linkage_results_a <- iris %>% mutate(cluster = clusters_a)

# step 6: check errors in cluster assignment based on methods
table(clusters_c)
```

```
clusters_c
 1  2
78 72
```

```r
table(clusters_s)
```

```
clusters_s
  1   2
 50 100
```

```r
table(clusters_a)
```

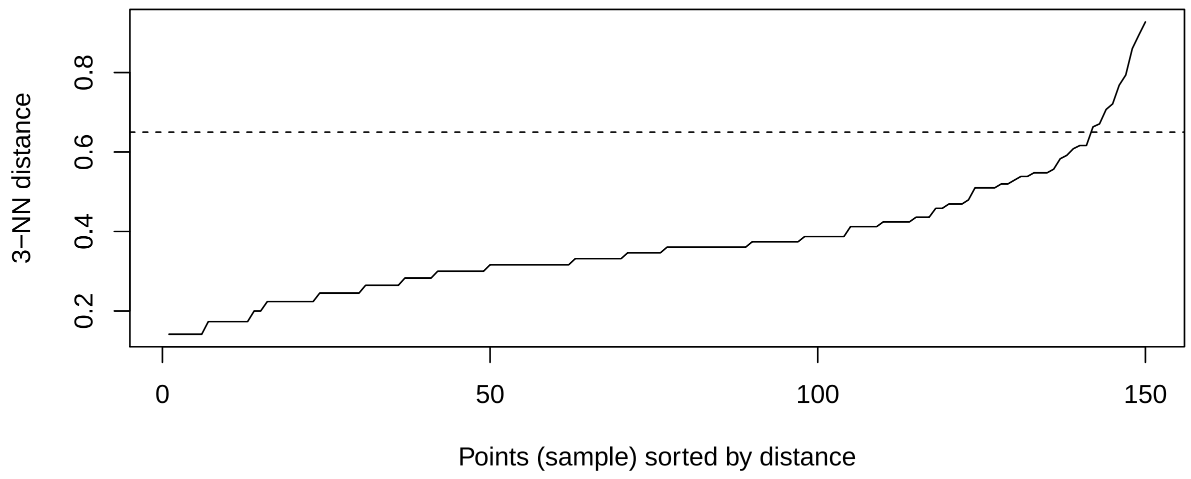
```
clusters_a
  1   2
 50 100
```

DBSCAN

The following code implements a density based approach to clustering these observations, and the steps are identified within the code. DBSCAN requires a distance matrix opposed to the original data and has two main arguments that are central to optimizing the results, "eps" and "minPts". Density Based clustering essentially tries to capture what we can do visually which is to identify dense regions from sparse regions and classify those dense regions as clusters. This is achieved by taking a user specified radius and capturing a user specified minimum number of points in that radius until that minimum number can no longer be achieved. This is a very strong technique when the dense regions (clusters) do not take a spherical or linear shape (i.e. donut or helix with volume). The algorithm basically takes a point and checks the quantity of points in the radius specified. Should that quantity be greater than or equal to the minimum specified, then those points become new central points, and the process repeats. When the minimum number of points can no longer be achieved those last point selected become border points that seal off the dense region.

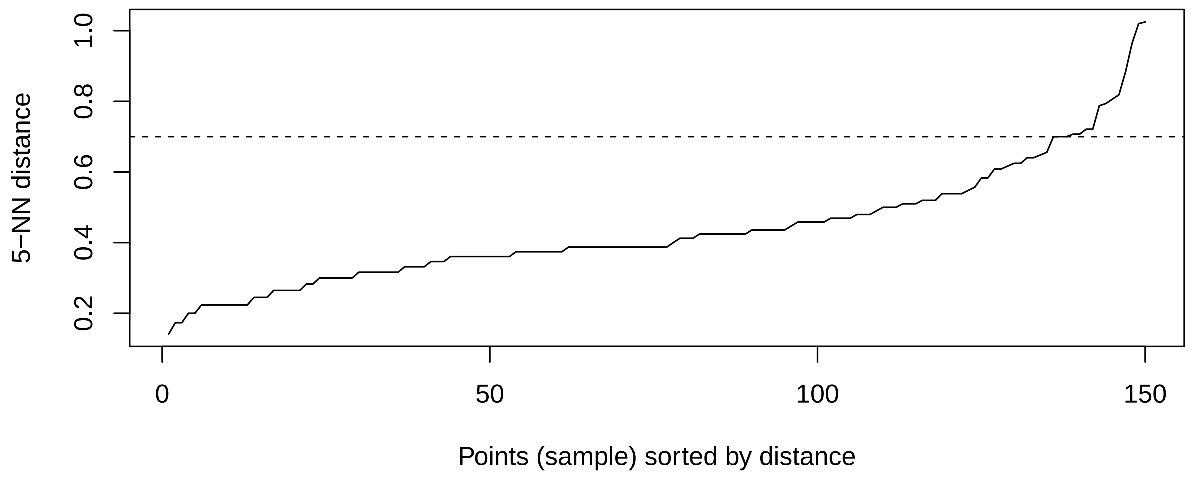
DBSCAN requires a radius and as seen with previous methods there is a way to achieve the optimal radius (eps), here using kNN. This method calculates the average distance between each point and it's "k" nearest neighbors, then plots this distance. Similar to the elbow method, we look for a spike in average distance where it starts to increase drastically and choose an "eps" from the horizontal line that marks that location as follows:

```r
# DBSCAN clustering----
# step 1: find the optimal eps
kNNdistplot(x = dist_iris, k = 3)
abline(h = 0.65, lty = 2)
```

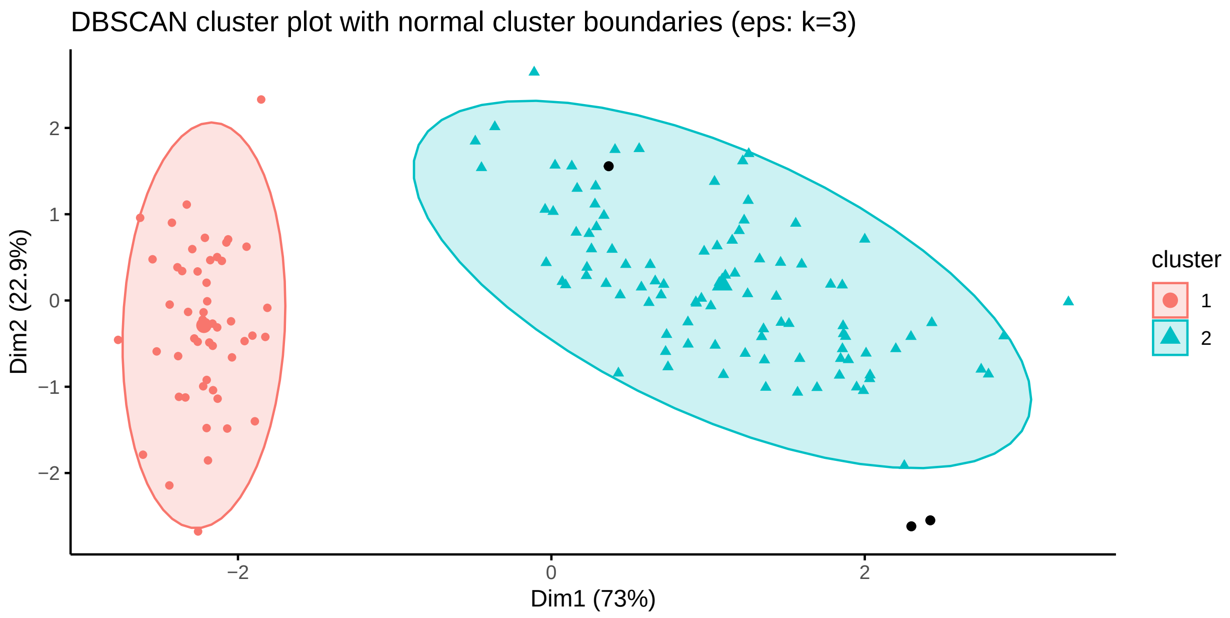


I chose k=3 for the number of neighbors based on inspection of the pairwise plots. This is arbitrary and guided by intuition more so than there being an optimal value. It is shown below that changing the value of “k” (within margin) will not drastically affect the margin of “eps” or the overall cluster results:

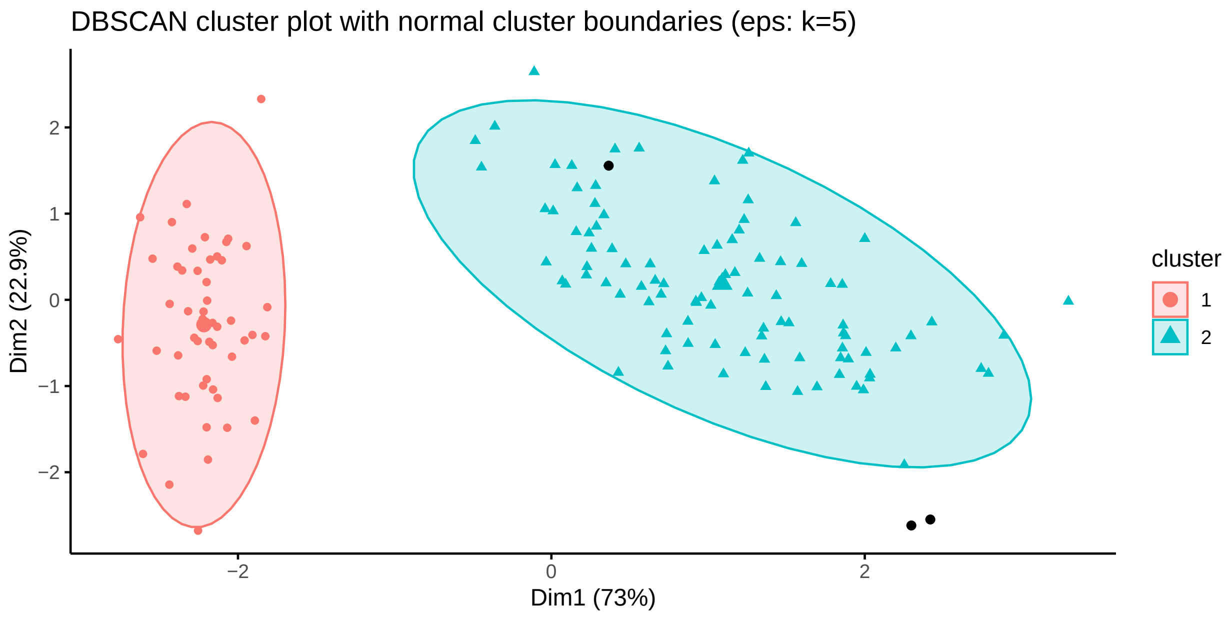
```
kNNdistplot(x = dist_iris, k = 5)
abline(h = 0.7, lty = 2)
```



```
# step 2: create a model
model_db_5 <- dbscan(x = dist_iris, eps = .65, minPts = 5) #.65 model

# step 3: visualization modelfor DBSCAN clustering technique
fviz_cluster(model_db_5, iris_c, geom = "point", ggtheme = theme_classic(),
             ellipse.type = "norm",
             main = "DBSCAN cluster plot with normal cluster boundaries (eps: k=3)")
```

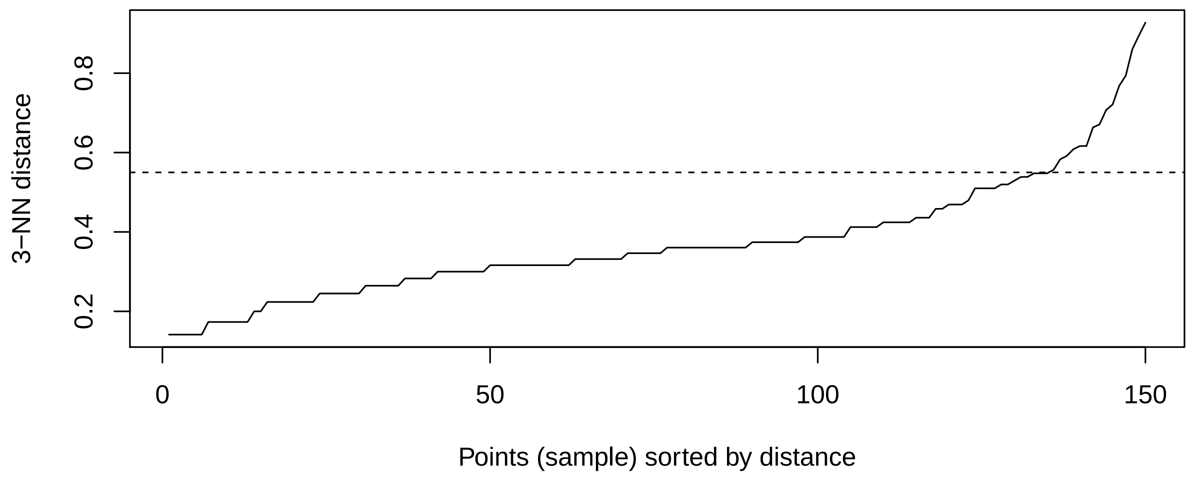


```
# step 2: create a model
model_db_6 <- dbscan(x = dist_iris, eps = .7, minPts = 5) #.7 model

# step 3: visualization model for DBSCAN clustering technique
fviz_cluster(model_db_6, iris_c, geom = "point", ggtheme = theme_classic(),
            ellipse.type = "norm",
            main = "DBSCAN cluster plot with normal cluster boundaries (eps: k=5)")
```

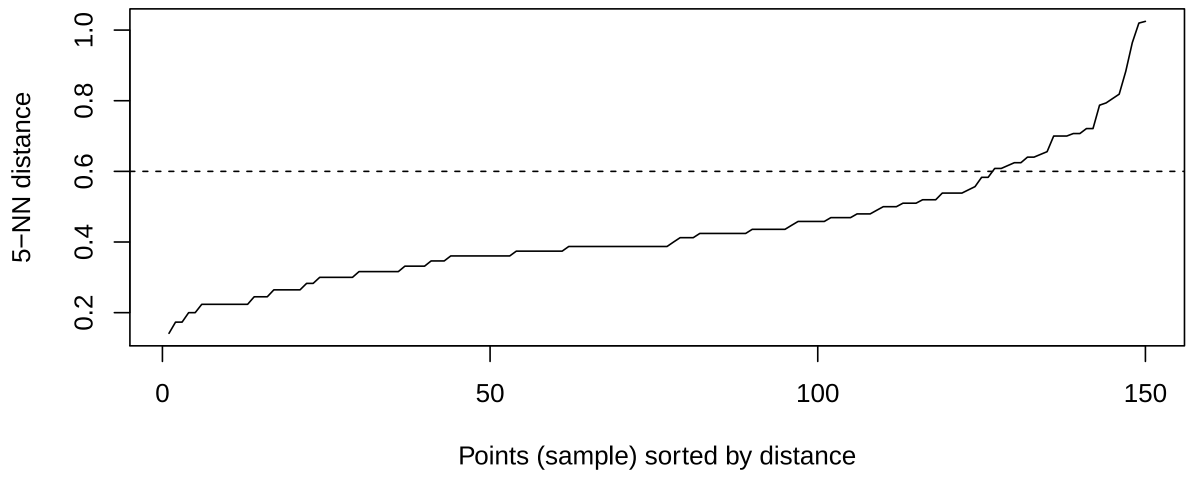


The above cluster plots for both kNN optimal "eps" values under different "k" values (3 and 5) are equivalent. Should the intuition be that the bend in the kNN curve is actually lower, both values of "eps" would change and hence the resulting plots will still be similar. This is shown below:

```r
kNNdistplot(x = dist_iris, k = 3)
abline(h = 0.55, lty = 2)
```

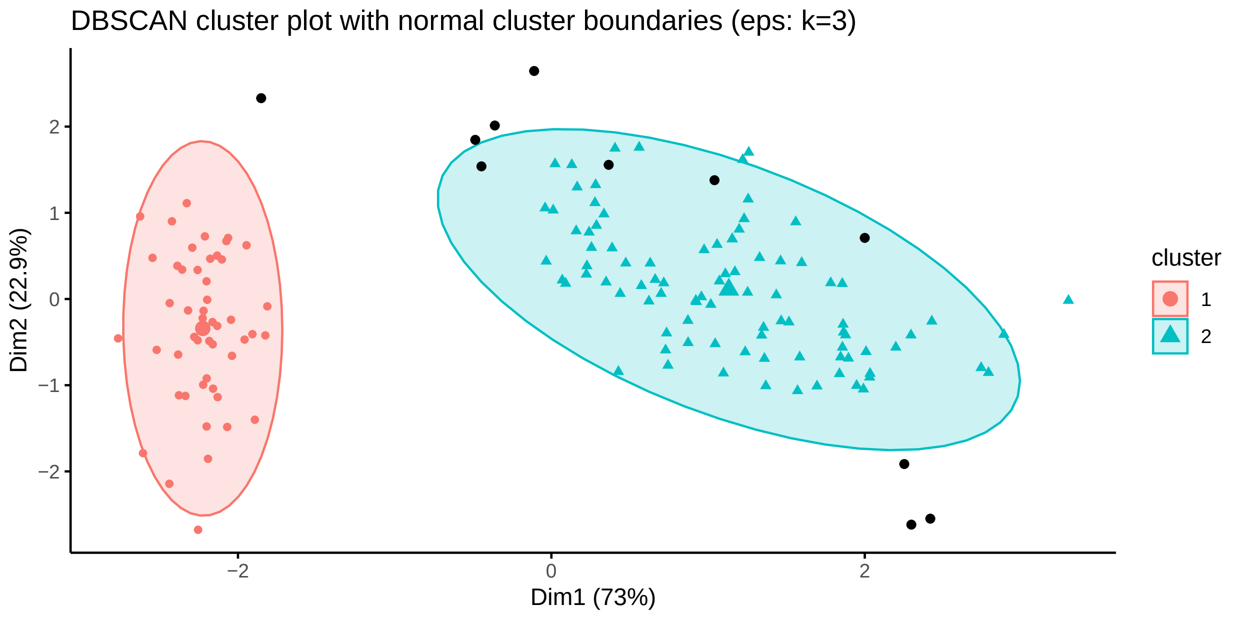


```r
kNNdistplot(x = dist_iris, k = 5)
abline(h = 0.6, lty = 2)
```

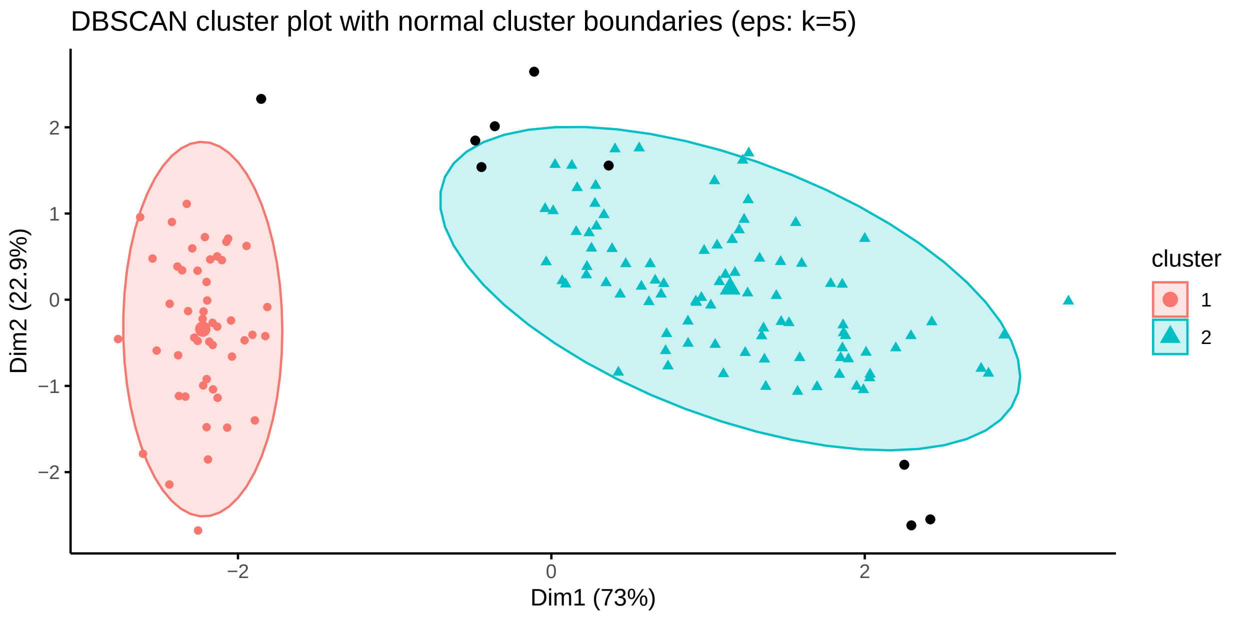


```r
# step 2: create a model
model_db_5 <- dbscan(x = dist_iris, eps = .55, minPts = 5) #.55 model
```

```
# step 3: visualization model for DBSCAN clustering technique
fviz_cluster(model_db_5, iris_c, geom = "point", ggtheme = theme_classic(),
             ellipse.type = "norm",
             main = "DBSCAN cluster plot with normal cluster boundaries (eps: k=3)")
```



```
# step 2: create a model
model_db_6 <- dbscan(x = dist_iris, eps = .6, minPts = 5) #.6 model

# step 3: visualization model for DBSCAN clustering technique
fviz_cluster(model_db_6, iris_c, geom = "point", ggtheme = theme_classic(),ellipse.type = "norm",
             main = "DBSCAN cluster plot with normal cluster boundaries (eps: k=5)")
```


DBSCAN cluster plot with normal cluster boundaries (eps: k=5)
Dim2 (22.9%)
Dim1 (73%)
cluster