# More on K-Means Clustering

Scott Thatcher

6/6/2021

## Preliminaries

We'll first load packages, then create a version of the iris data set with only the sepal length and width (so that we can makes graphs without hidden variables affecting the clustering).

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.2     v dplyr   1.0.6
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
my_iris <- iris %>% select(contains("Sepal"))
head(my_iris)
```

```
##   Sepal.Length Sepal.Width
## 1          5.1         3.5
## 2          4.9         3.0
## 3          4.7         3.2
## 4          4.6         3.1
## 5          5.0         3.6
## 6          5.4         3.9
```

## The Importance of Scaling Variables when Clustering

If you use a clustering method that measures distances using a Euclidean distance metric (or any distance metric that depends on numeric values of the various measured variables), you might run into a problem due to the fact that different measured quantities can have different units. Choice of units will affect the numeric magnitude of the values in the data set, and thus the choice of units for different variables will affect which contribute most strongly to the overall distance measure between two data points. In other words, choice of measurement units affects how data points are clustered.

### An Iris Example

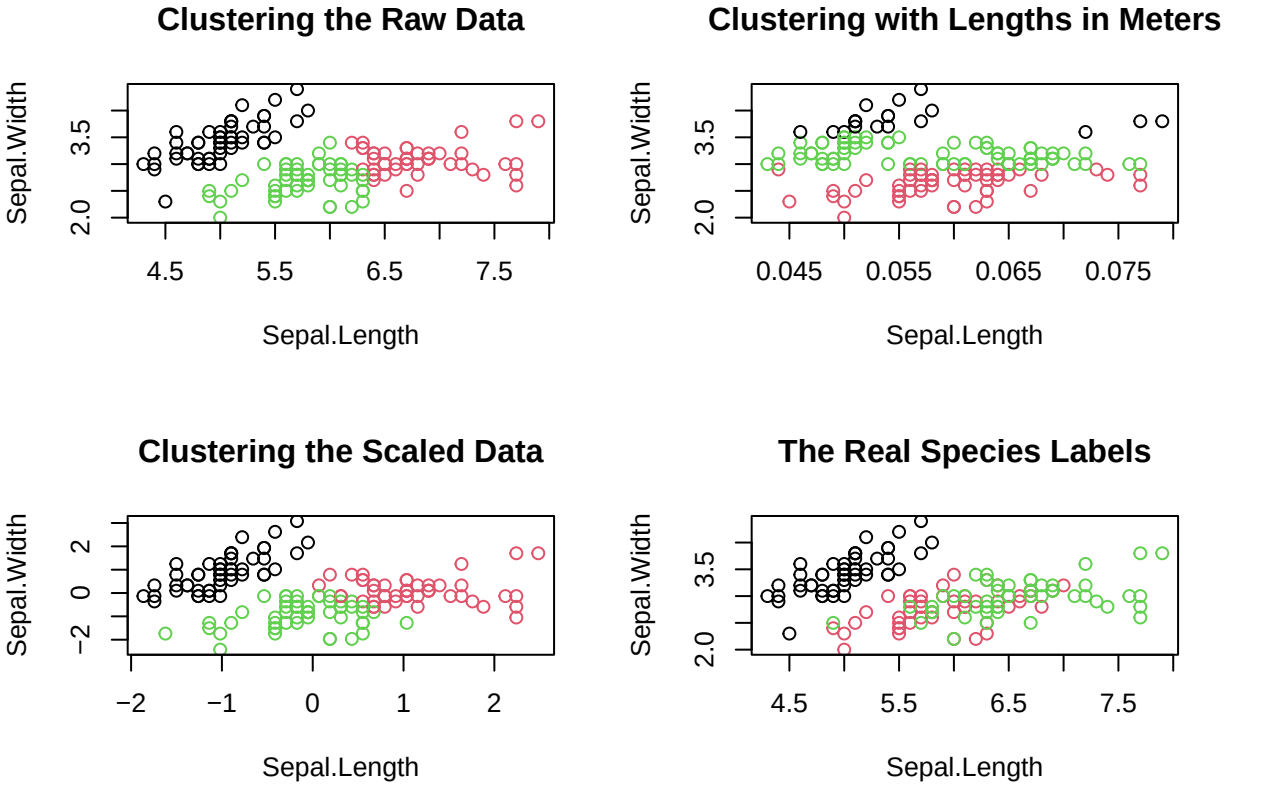
Here's an example. The graphs below show k-means clustering of the iris data in

- its raw form (all measurements in cm),

- with sepal length measured in meters and sepal width measured in cm (this is a bit silly, but illustrates the point), and
- with all variables standardized to have mean 0 and standard deviation 1 (in other words, standardized by calculating z-scores).

The example also shows the true species classification.

```
# Here's we'll set the seed before each k-means run to show that we're getting
# different results, even from the same starting point.
seed <- 31
set.seed(seed)
km_raw <- kmeans(my_iris, centers=3)
set.seed(seed)
my_iris_meters <- my_iris %>% mutate(Sepal.Length = Sepal.Length/100)
km_meters <- kmeans(my_iris_meters, centers=3)
set.seed(seed)
my_iris_scaled <- scale(my_iris)
km_scaled <- kmeans(my_iris_scaled, centers=3)
par(mfrow=c(2,2))
plot(my_iris[, 1:2], col=km_raw$cluster, main="Clustering the Raw Data")
plot(my_iris_meters[, 1:2], col=km_meters$cluster, main="Clustering with Lengths in Meters")
plot(my_iris_scaled[, 1:2], col=km_scaled$cluster, main="Clustering the Scaled Data")
plot(my_iris[, 1:2], col=iris$Species, main="The Real Species Labels")
```




## Which Scaling is Best?

If units of each variable are the same, and furthermore if each *should* have equal weight in determining clusters, then it might make sense to leave variables in their original form. For example, if you're clustering by physical location, and x and y are both measured in kilometers from on origin point.

On the other hand, it's often best to standardize variables if they are measuring different quantities with different units, or even if they share the same units, but the scale of variation is different for the different variables.

For example, all iris measurements are in centimeters, but you might argue that both mean and variance in sepal width are on a different scale from mean and variance of sepal length. Therefore, the two are not directly comparable for the purposes of classification, and they *should* be standardized.

## The Effects of Iteration and Randomization

The k-means algorithm starts by choosing *random* centers for each of its clusters. Therefore, it's possible that the final clustering will depend on the initial random seed.

Furthermore, the clustering is refined at each step of the algorithm. Therefore, final clusters can depend on the number of iterations the algorithm uses.

Let's look at these one-at-a-time.

### Iterations

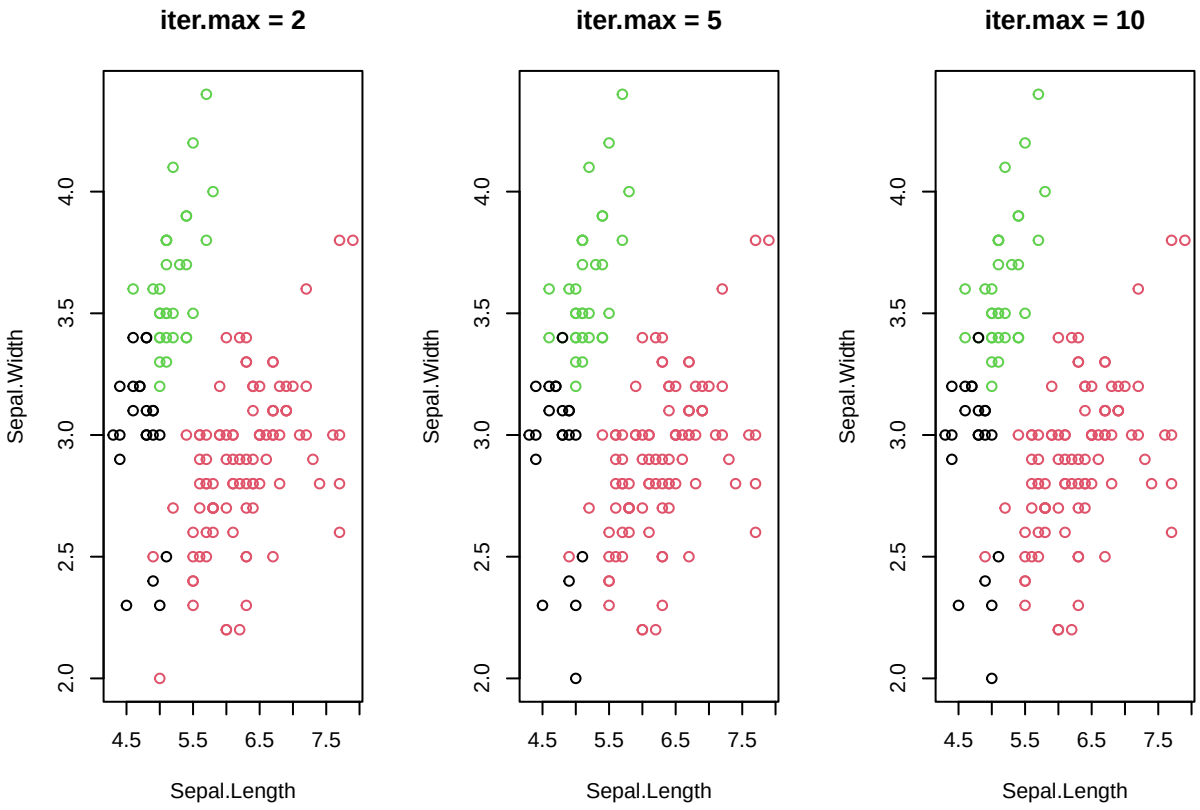
The algorithm will exit when no further changes to clusters are detected, but that is not guaranteed to happen. Therefore, the `kmeans` command also allows a `iter.max` option (by default, set to 10) will will make sure the algorithm does terminate after a certain number of iterations.

Let's see the effect of changing the size of `iter.max`.

```
# Note: It was actually hard to get an example where `iter.max=2` didn't give
# the same results as the others. The default algorithm ("Hartigan-Wong") is
# actually a bit more involved than what was described in the lecture, and
# pretty good with this size data set.
seed <- 87
set.seed(seed)
km2 <- kmeans(iris[, 1:4], centers=3, iter.max=2)
```

```
## Warning: did not converge in 2 iterations
```

```
set.seed(seed)
km5 <- kmeans(iris[, 1:4], centers=3, iter.max=5)
set.seed(seed)
km10 <- kmeans(iris[, 1:4], centers=3, iter.max=10)
par(mfrow=c(1,3))
plot(iris[, 1:2], col=km2$cluster, main="iter.max = 2")
plot(iris[, 1:2], col=km5$cluster, main="iter.max = 5")
plot(iris[, 1:2], col=km10$cluster, main="iter.max = 10")
```




```
# This code will confirm whether any cluster assignments have changed.
# any(km2$cluster != km5$cluster)
# any(km5$cluster != km10$cluster)
```

So, for many situations, the default value of `iter.max` might be OK, but be aware of any warnings that the algorithm didn't converge!

## Effects of the Random Seed

Even when the k-means algorithm converges, the results can be different in two ways, both of which depend on the random seed:

- Clusters might be the same, but labeled differently. (Cluster 1 the first time is cluster 3 the second time, etc.)
- Clusters might simply turn out differently.

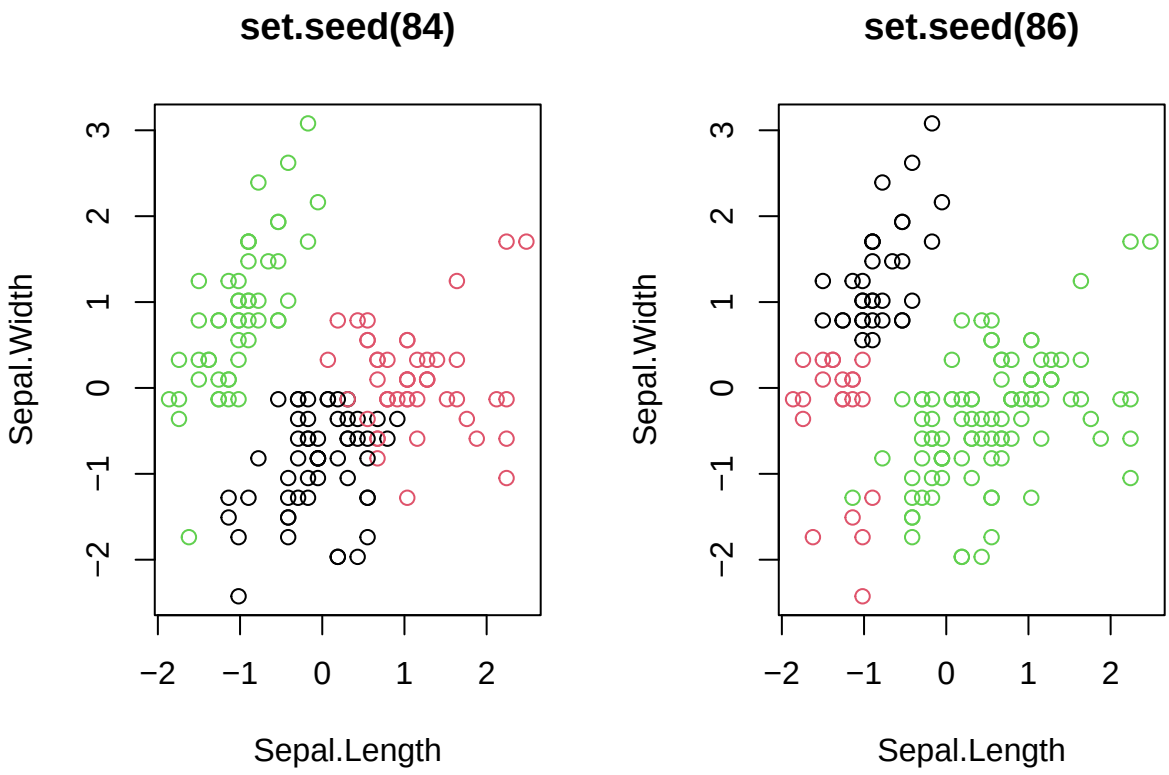
The first issue is not a big problem, it's it's good to be aware of. The second issue is more important–a user needs to be aware that a k-means clustering is not guaranteed to always give the same clusters.

### Different Seeds can Give Different Clusterings

Here's an example of the issue. We'll go back to using the entire four- variable iris data set here. Note that the cluster differ in more than color assignment. In the first, the upper-left group is one cluster, while in the second, the upper-left group of points in sorted into two clusters.

```
# We'll standardize all variables.
my_iris <- scale(iris[, 1:4])
set.seed(84)
km1 <- kmeans(my_iris, centers=3)
```

```r
set.seed(86)
km2 <- kmeans(my_iris, centers=3)
par(mfrow=c(1,2))
plot(my_iris[, 1:2], col=km1$cluster, main="set.seed(84)")
plot(my_iris[, 1:2], col=km2$cluster, main="set.seed(86)")
```




### Which Clustering is "Best"?

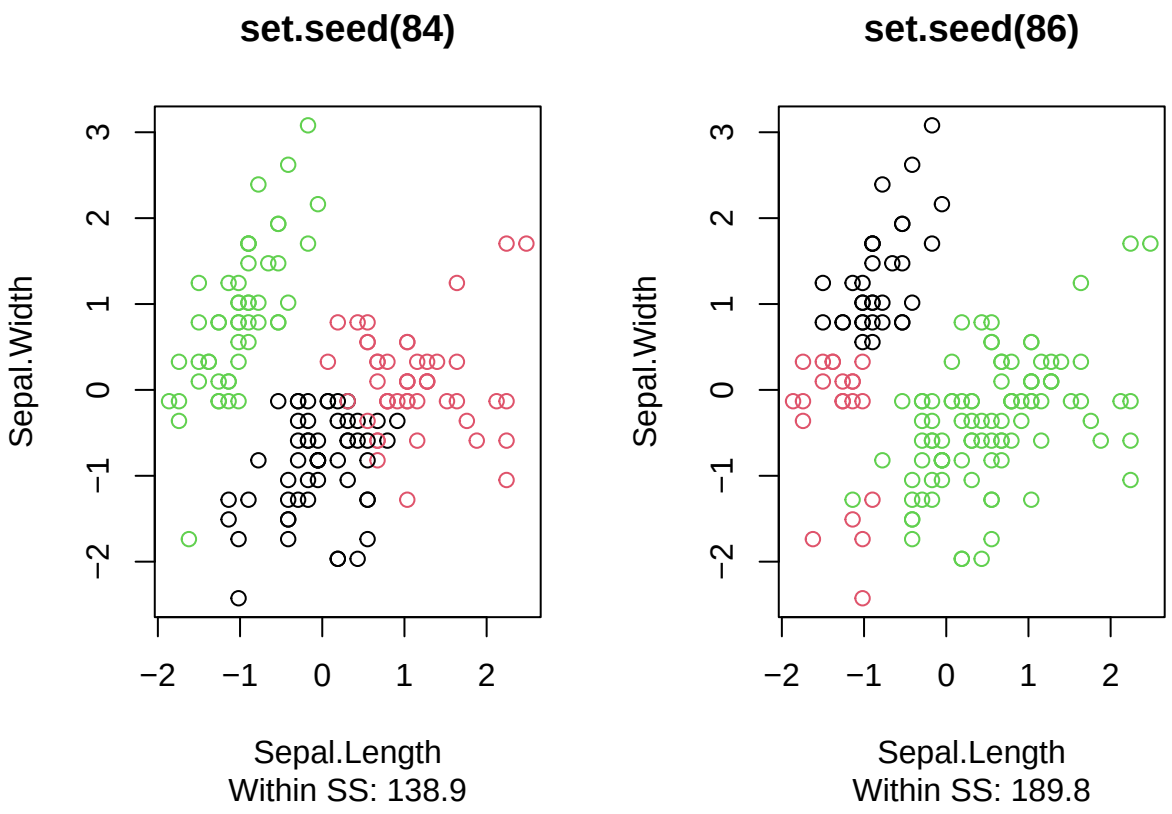
There can be many definitions of what makes a good clustering. A common choice of criterion, however, is to look for the clustering that has the minimum *within-cluster sum of squares*. In other words, we want to minimize

$$\sum_{i=1}^{k} \sum_{x \in C_k} d(x, c_k)^2,$$

the sum of squared distances of each data point to the center of its cluster. A clustering that minimized the within-cluster sum of squares has clusters that are more compact that a clustering with a larger sum of squares.

Here are our two clusters, and their within-cluster sum of squares:

```r
par(mfrow=c(1,2))
plot(my_iris[, 1:2], col=km1$cluster, main="set.seed(84)",
     sub=paste("Within SS:", round(km1$tot.withinss, 1)))
plot(my_iris[, 1:2], col=km2$cluster, main="set.seed(86)",
     sub=paste("Within SS:", round(km2$tot.withinss, 1)))
```



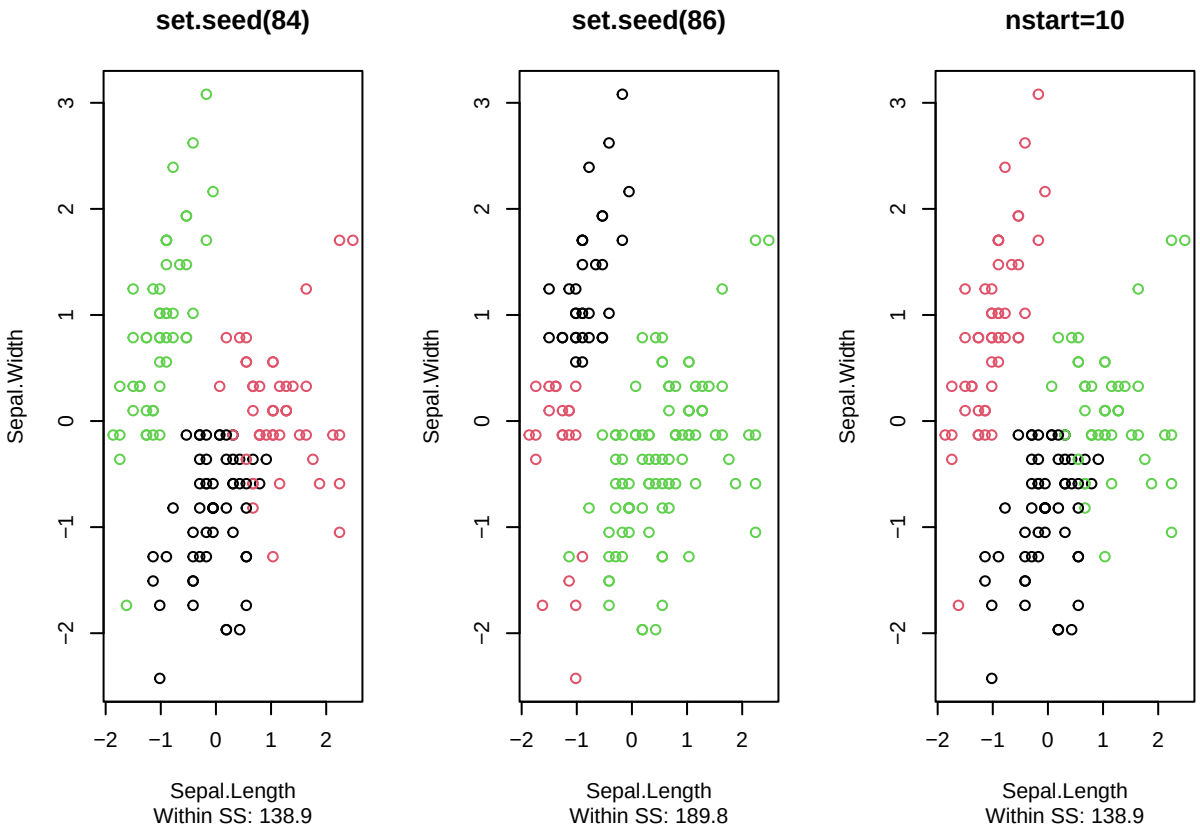

The left-hand clustering is clearly preferred by this criterion.

### Automating with `nstarts`

Rather than checking many random seeds by hand, and choosing the lowest within- cluster sum of squares, you can use the `nstarts` option of the `kmeans` command to automate the process. This option specifies that the entire algorithm will be run several times with different random starts, and the best clustering, according to within-cluster sum of squares, will be chosen as the final result.

For a simple data set, setting a small value for `nstarts` may be enough, but if the data set is larger, you may have to experiment with larger values of `nstarts`. Here's what it looks like for our iris data:

```
km_n <- kmeans(my_iris, centers=3, nstart=10)
par(mfrow=c(1,3))
plot(my_iris[, 1:2], col=km1$cluster, main="set.seed(84)",
     sub=paste("Within SS:", round(km1$tot.withinss, 1)))
plot(my_iris[, 1:2], col=km2$cluster, main="set.seed(86)",
     sub=paste("Within SS:", round(km2$tot.withinss, 1)))
plot(my_iris[, 1:2], col=km_n$cluster, main="nstart=10",
     sub=paste("Within SS:", round(km_n$tot.withinss, 1)))
```




## Hierarchical Clustering

Hierarchical clustering doesn't depend on a random start, so that part of this discussion doesn't apply to hierarchical clustering. However, hierarchical clustering does depend on a distance measure between data points, and the discussion of standardization does apply to doing hierarchical clustering. It's usually a good idea to standardize variables with the `scale` command before doing a hierarchical clustering method.