Lab: K-Means

Rose Hoover

Getting Started

1. Download the `.qmd` file from Moodle and any needed `.xlsx` or `.csv` data files. Save these in the *same folder/directory*.
2. Open the Quarto file in RStudio: `File > Open File... >`. If you're working on the MHC RStudio server, you need to upload the files first: go to the `Files` panel, then click `Upload`. Upload the `.qmd` file and any data files. You will need to upload each file one at a time.
3. Update the author and date in the YAML header of this file.
4. Click the `Render` button. If successful, you should have a new window pop up with a nice looking HTML document.
5. For this lab, you **may need** to still the package `glmnet`.

Ask for help if you encounter issues on any of the steps above. Once you've successfully made it through these steps, you can continue.

Load Packages

You likely will need to install some these packages before you can run the code chunk below successfully.

```r
library(tidyverse)
library(palmerpenguins)
library(factoextra)
library(amap)
```

Load Penguin Data

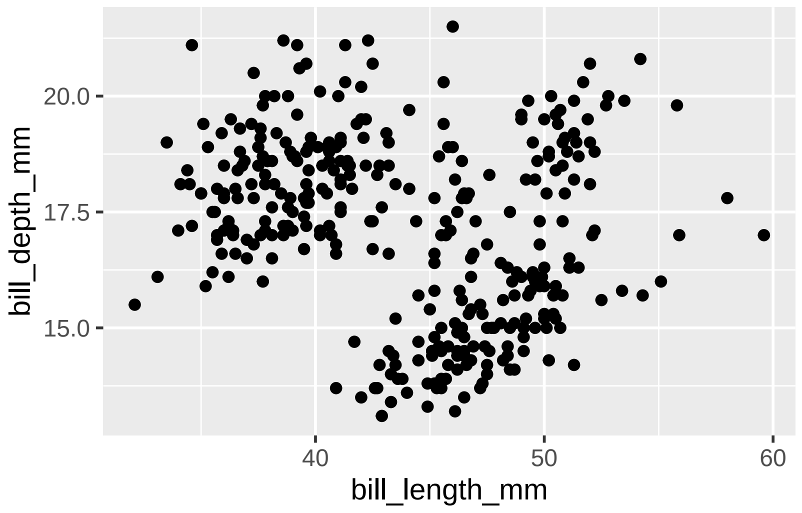
```
data(penguins)
```

Data Cleaning

```
# Remove missing values
# YOUR CODE HERE
penguins = penguins %>%
  filter(!is.na(bill_length_mm) & !is.na(bill_depth_mm) & !is.na(species))
# Make data table (named penguins_reduced) that only has
# bill_length_mm and bill_depth_mm columns
penguins_reduced <- penguins %>% select(bill_length_mm,bill_depth_mm)
```

Initial Visualization

```
ggplot(penguins, aes(x = bill_length_mm, y = bill_depth_mm)) +
  geom_point()
```

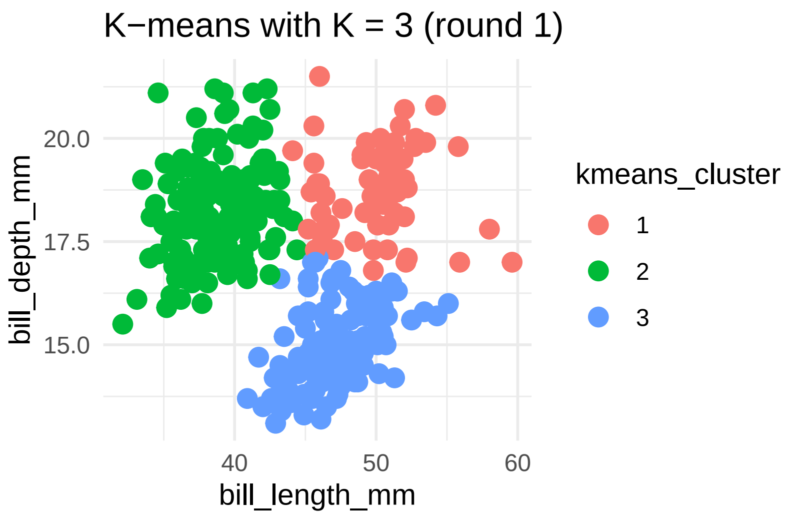


We'll cluster these penguins based on their bill lengths and depths:

Implement K-Means

Complete the code below to run the K-means algorithm using K = 3.

```r
set.seed(244)
# Run the K-means algorithm
kmeans_3_round_1 <- kmeans(scale(penguins_reduced), centers = 3)

# Plot the cluster assignments
penguins_reduced %>%
  mutate(kmeans_cluster = as.factor(kmeans_3_round_1$cluster)) %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = kmeans_cluster))
   ↪  +
  geom_point(size = 3) +
  theme(legend.position = "none") +
  labs(title = "K-means with K = 3 (round 1)") +
  theme_minimal()
```

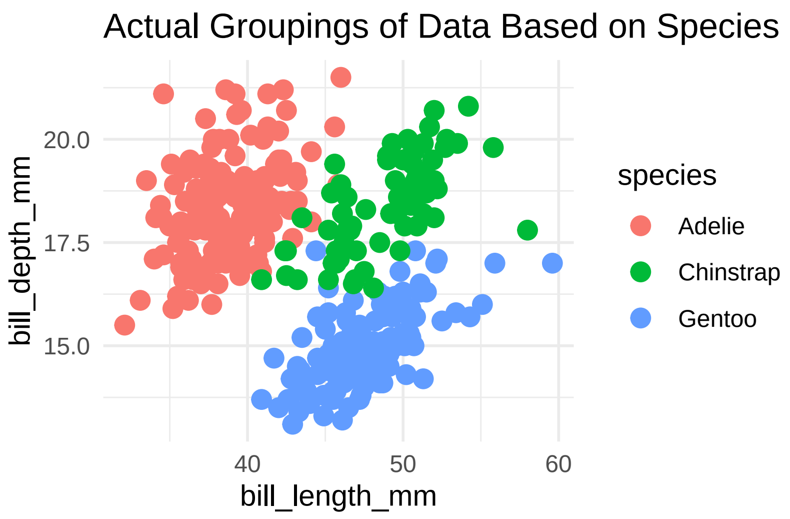


- Why do we have to set the seed for K-means? In practice, why should we try out a variety of seeds?

Answer. K means is a greedy algorithm and its possible so its possible for some random locations to have better/different results each time. It is also possible for it to get stuck at a local solution

K-Means Clusters Versus Known Species Groupings

```r
ggplot(data = penguins, aes(x = bill_length_mm, y = bill_depth_mm, color =
 ↪  species)) +
geom_point(size = 3) +
theme(legend.position = "none") +
labs(title = "Actual Groupings of Data Based on Species") +
theme_minimal()
```



- Visually, how well do you think K-means captured the underlying species structure of the data?

Answer. It seems like it found roughly the data points corresponding to each of the species groups. It is identifying that as a way to structure the data.

Tuning K

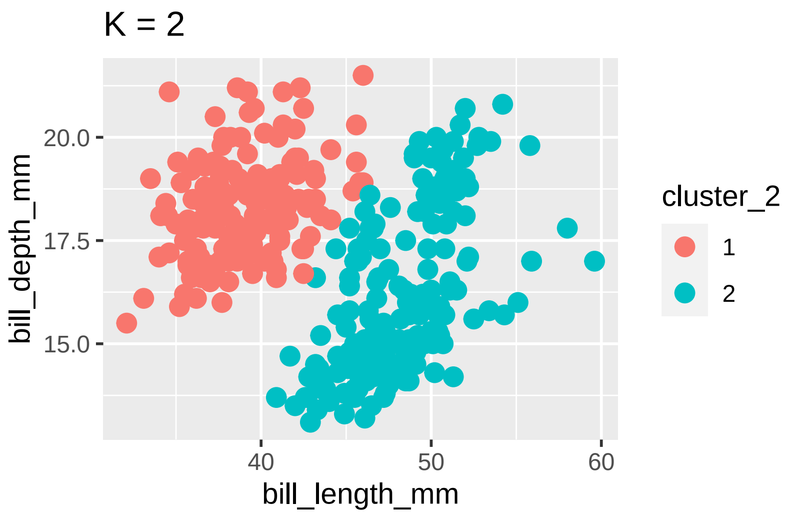
- To implement K-means clustering we must choose an appropriate K! Use the following example to see the two different extreme situations. Typically, the ideal K is somewhere between the two extremes.
- **Minimum:** $K = 2$ groups/clusters
- **Maximum:** $K = n$ groups/clusters (one observation per cluster)

What happens in the K-means algorithm if $K = n$?

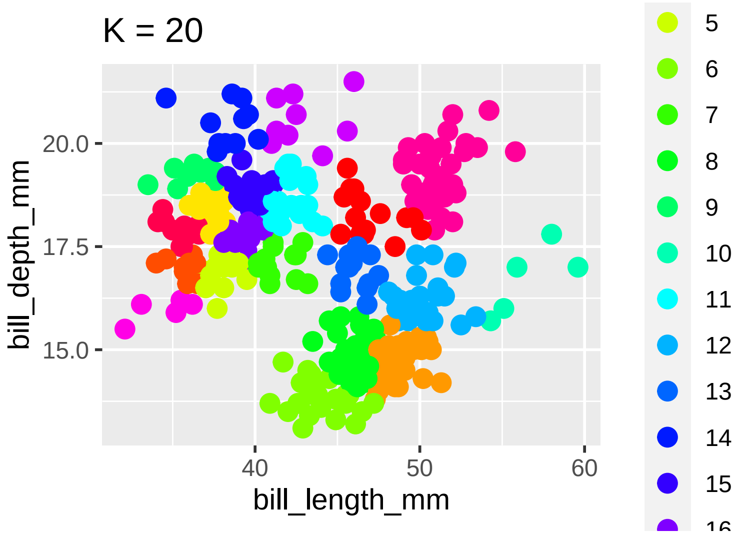
Answer. YOUR ANSWER HERE

Let's consider anywhere from $K = 2$ to $K = 20$ clusters.

```r
set.seed(244)

k_2  <- kmeans(scale(penguins_reduced), centers = 2)
k_20 <- kmeans(scale(penguins_reduced), centers = 20)

penguins_reduced %>%
  mutate(cluster_2 = as.factor(k_2$cluster)) %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = cluster_2)) +
    geom_point(size = 3) +
    labs(title = "K = 2")
```



```r
penguins_reduced %>%
  mutate(cluster_20 = as.factor(k_20$cluster)) %>%
  ggplot(aes(x = bill_length_mm, y = bill_depth_mm, color = cluster_20)) +
    geom_point(size = 3) +
    labs(title = "K = 20") +
    scale_color_manual(values = rainbow(20))
```



What are your general impressions?

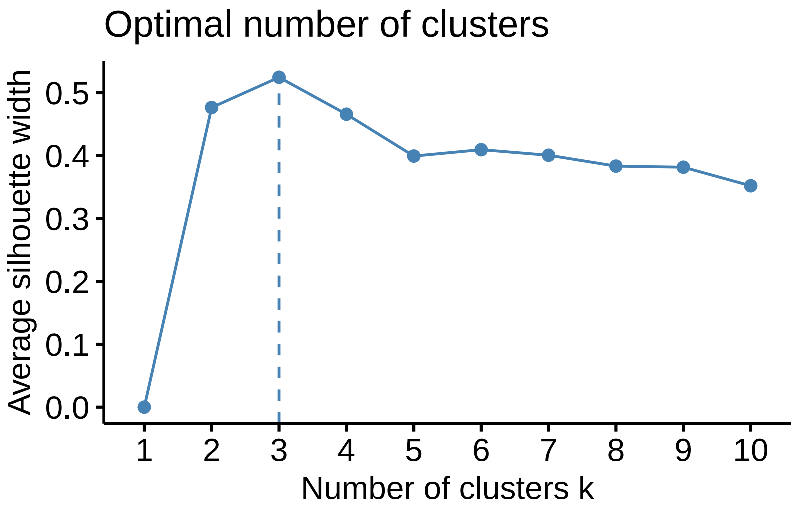
Answer. YOUR ANSWER HERE

Finding Ideal K Value: Silhoutte

- The *average silhouette approach* measures the quality of a clustering. That is, it determines how well each object lies within its cluster.
  - To do so, it maximizes the distance between clusters and minimizes distance within clusters.
- A high average silhouette indicates a good clustering.
- Given a range of possible K values, the optimal number of clusters (K) is the one that maximizes the average silhouette.

We can use a built-in silhouette method in the `fviz_nbclust` function to compute the average silhouette for various K values.

```
fviz_nbclust(scale(penguins_reduced), kmeans, method='silhouette')
```



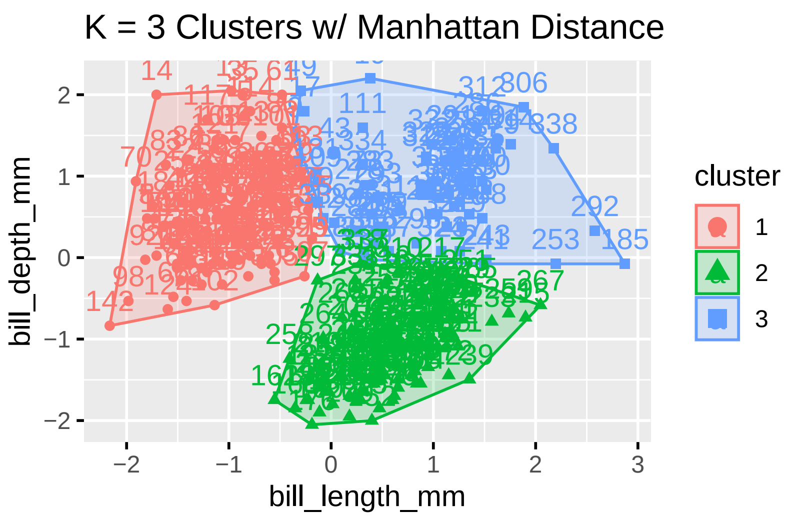
Based on the *average silhouette approach*, what is the optimal K value?

Answer. optimal value is about 3!

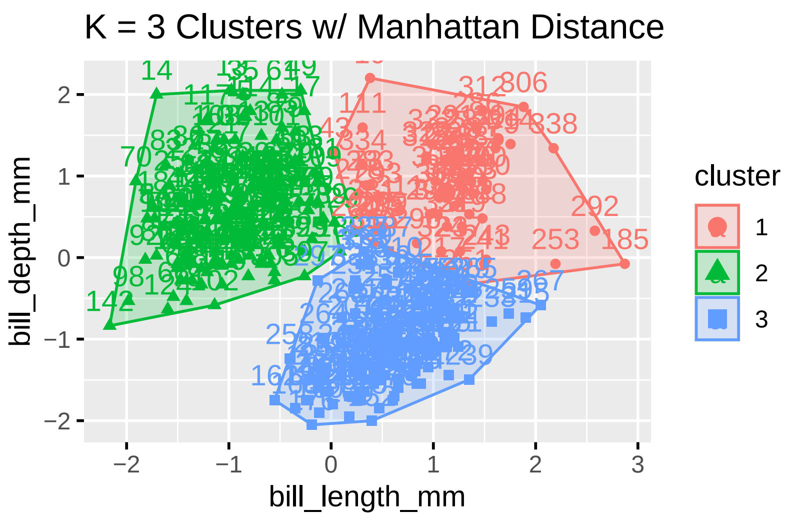
Experimenting with Distance Metrics

We can use the `Kmeans` method (notice the "K" is capitalized in this function name) from the `amap` library to specify how we are measuring distance in the K-means algorithm.

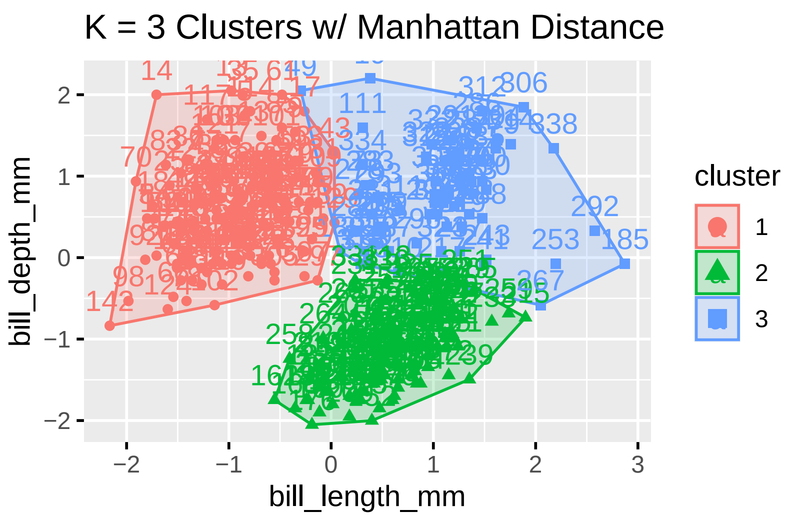
```
set.seed(244)
k_2_manattan = Kmeans(scale(penguins_reduced), centers = 3,
                      method = "manhattan")
k_2_euclid = Kmeans(scale(penguins_reduced), centers = 3,
                    method = "euclidean")
k_2_maxnorm = Kmeans(scale(penguins_reduced), centers = 3,
                     method = "maximum")



fviz_cluster(k_2_euclid, data = scale(penguins_reduced),
             main = sprintf("K = %d Clusters w/ Manhattan Distance", 3))
```



```r
fviz_cluster(k_2_manattan, data = scale(penguins_reduced),
             main = sprintf("K = %d Clusters w/ Manhattan Distance", 3))
```



```r
fviz_cluster(k_2_maxnorm, data = scale(penguins_reduced),
             main = sprintf("K = %d Clusters w/ Manhattan Distance", 3))
```



Try changing K to equal 3$ in the code chunk above. How do the clusterings using the 3 distance metrics compare? What do you generally observe?

Answer. YOUR ANSWER HERE

Modify the code in the chunk above so that we can easily change the value of K (rather than making sure to change K manually in every line). In general coding practices, is called *extracting out a constant.*