summer_student_kmeans

2024-07-02

```
data(iris) #The dataset that we will be using for this exercise
library(ggplot2) #A package used to make plots
library(ggstar)
set.seed(5) #A random seed, used so that every "random" iteration is the same
```

1. What does our data look like?

We will be trying to separate 3 types of flowers based on their characteristics: petal and sepal width and length

```
head(iris) #The head() function prints the first few lines of a dataset
```

```
##   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
```

```
summary(iris) # The summary() function gives us some statistics on all the columns in the dataset
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

2. Functions

In maths, a function takes an input, applies an operation and gives an output. For instance:

$$f(x) = x^2 + 2$$

takes an input number (x), applies an operation ($x^2 + 2$) and outputs a new value ($f(x)$).

In computer science, functions work the exact same way, they have:

An input

Some code that will manipulate the input

An output

In a k-means classifier, we have the following functions:

euclidean_distance:

initialize_centroids:

assign_clusters:

update_centroids:

kmeans_algorithm:

Let's have a look at how each one of them works:

Euclidean distance

The Euclidean distance function returns the Euclidean distance between two points in a multidimensional space. For instance, in 2 dimensions the distance between points 1 and 2 is:

$$d = \sqrt{(x_q - x_1)^2 + (y_2 - y_1)^2}$$

in 3 dimensions:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

and in n dimensions:

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Inputs:

point1 - numerical vector of any dimension (x, y, z, ...)

point2 - numerical vector of any dimension (x, y, z, ...)

Outputs: the Euclidean distance, a numebr representing the distance between the 2 points.

```
euclidean_distance <- function(point1, point2) {
  distance = sqrt(sum((point1 - point2)^2))
  return(distance)
}
```

Initialising the centroids

The centroids will be chosen randomly at first (k random points from the dataset), before being assigned by their mathematical definition for the enxt iterations. Input: the dataset Output: k randomly selected rows from the dataset, which are now the centroids.

```
initialize_centroids <- function(data, k) {
  data[sample(1:nrow(data), k), ]
}
```

Assigning clusters

Each point in the dataset will be assigned to a cluster, according to the centroid they are closest to. Inputs:

The dataset

The coordinates of the centroids

Output: A list of numbers 1-k, representing the centroid that each datapoint belongs to.

```
assign_clusters <- function(data, centroids) {
  clusters <- vector("numeric", nrow(data))
  for (i in 1:nrow(data)) {
    distances <- apply(centroids, 1, function(centroid) euclidean_distance(data[i, ], centroid))
    clusters[i] <- which.min(distances)
  }
  clusters
}
```

Updating the centroids

The non-random way to assign centroids, which we do after the 1st iteration, is thanks to their mathematical definition. The centroids are assigned as the point which is in the middle of all the points in a cluster. Inputs:

The dataset

The cluster assignments

Output: The coordinates for the new centroids

```
update_centroids <- function(data, clusters, k) {
  centroids <- matrix(NA, nrow = k, ncol = ncol(data))
  for (i in 1:k) {
    cluster_points <- data[clusters == i, ]
    if (nrow(cluster_points) > 0) {
      centroids[i, ] <- colMeans(cluster_points)
    } else {
      centroids[i, ] <- data[sample(1:nrow(data), 1), ]
    }
  }
  centroids
}
```

Putting everything together: the k-means clustering algorithm

Inputs:

The dataset

k

The tolerance: the minimum difference between the previous and next centroids for us to consider that the difference was big enough, and that we should iterate again

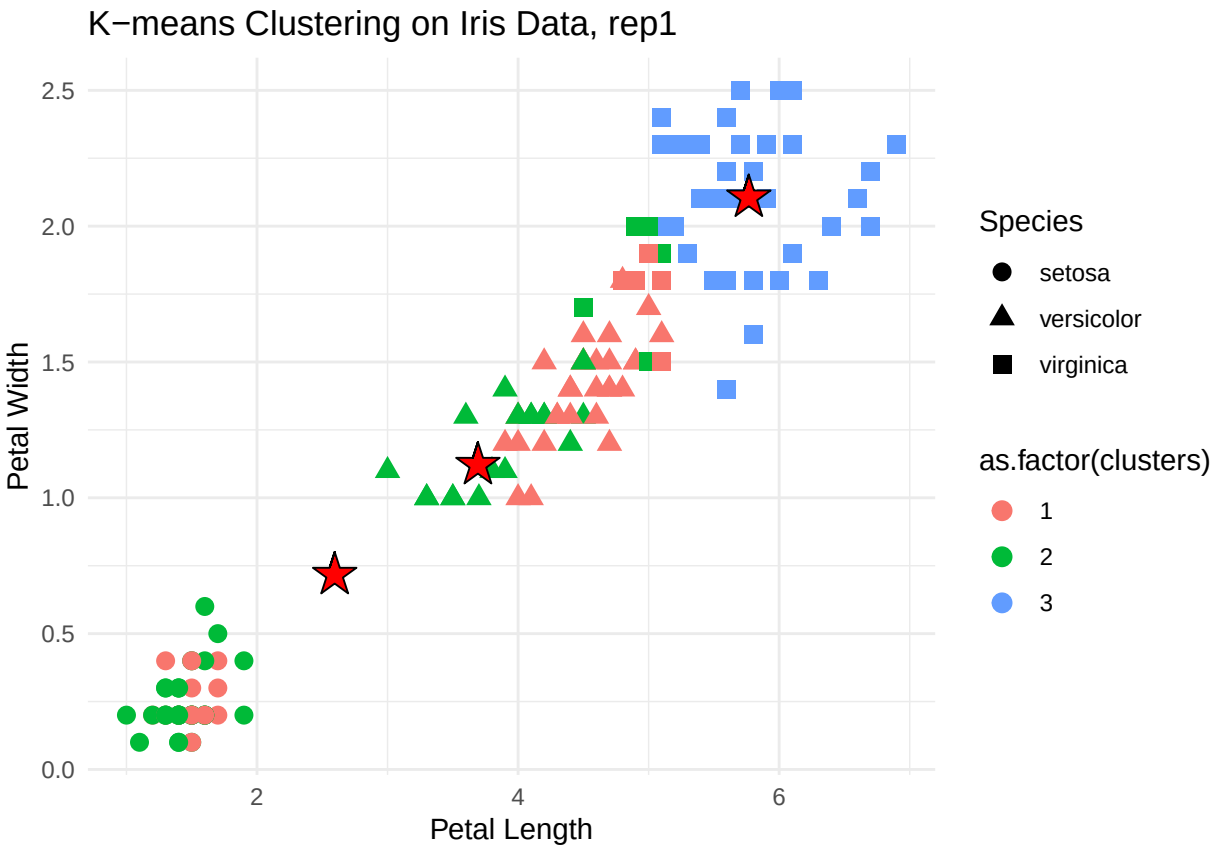
The maximum number of iterations: the number of repeats after which we will stop reassigning centroids, even if the threshold has not been reached

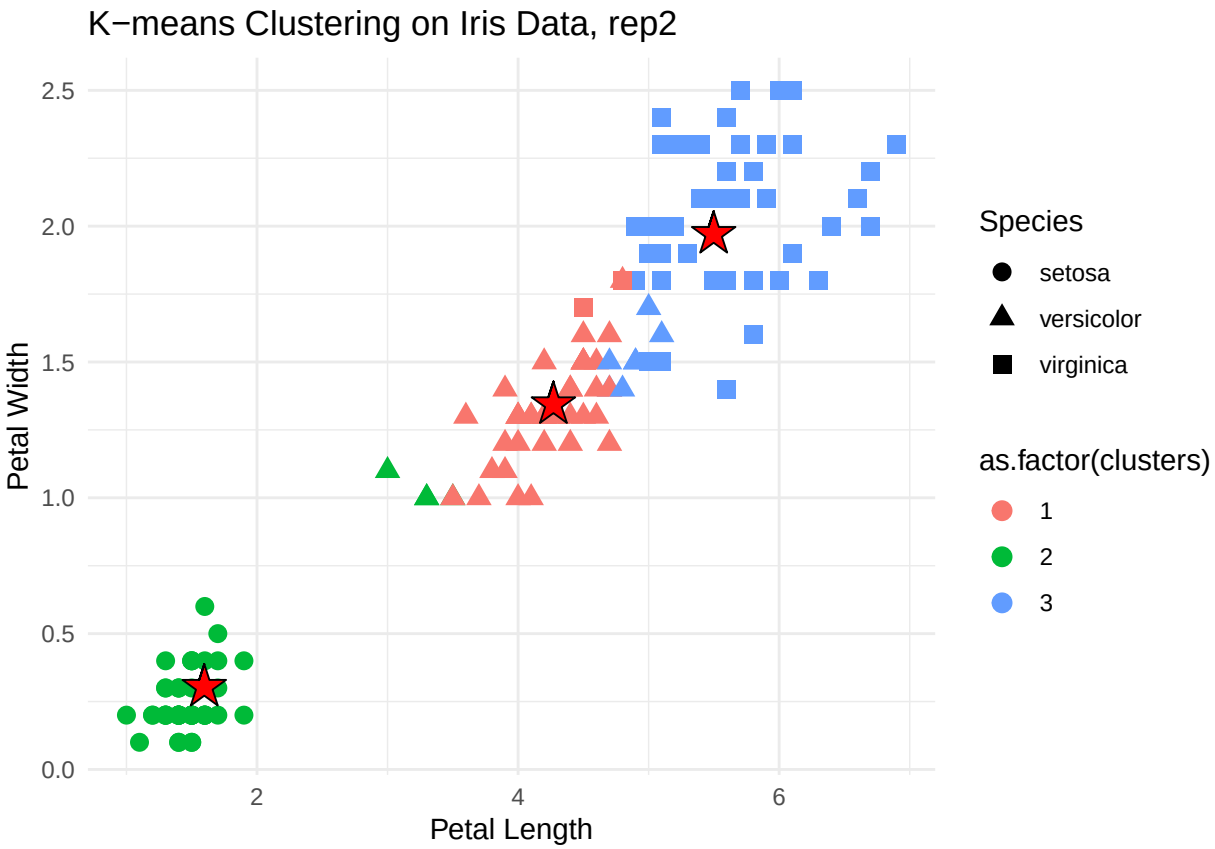
Output: The clusters assigned to all the data points.

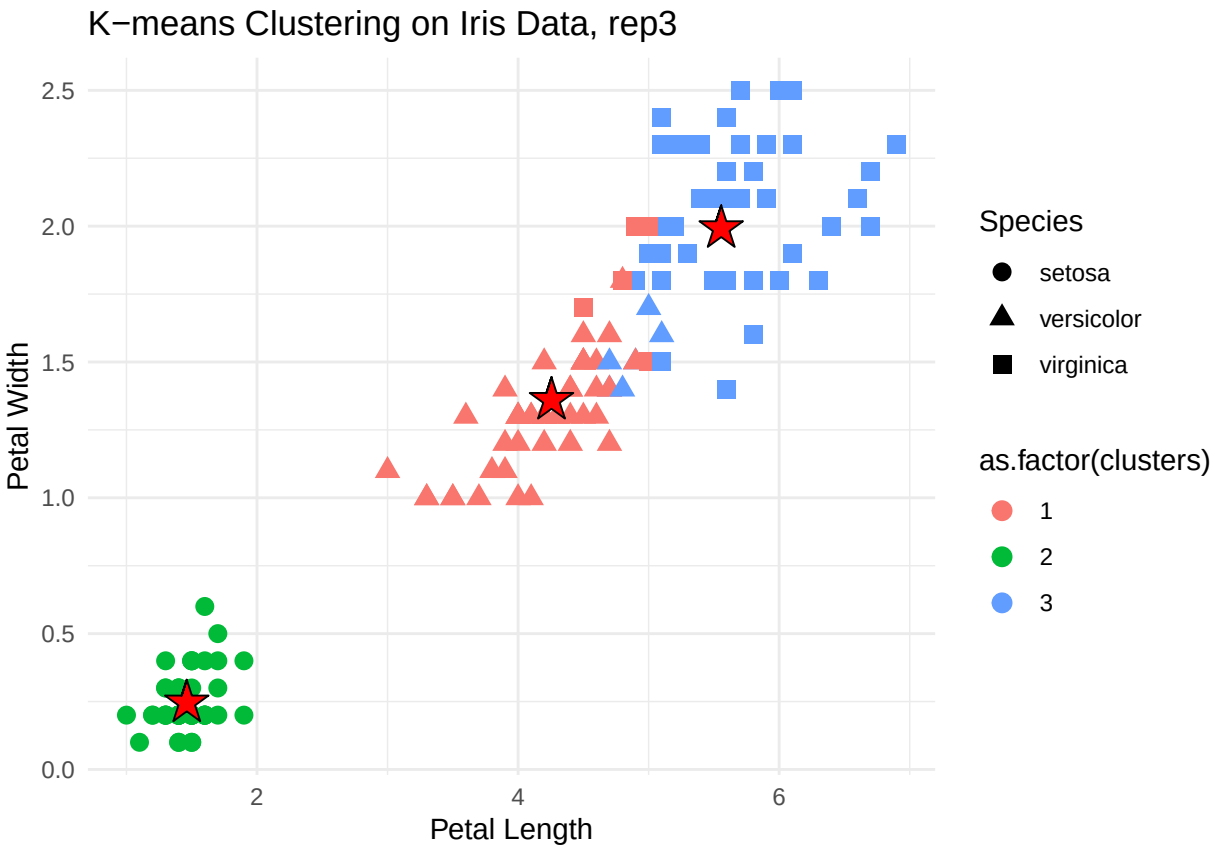
```
kmeans_algorithm <- function(data, k, max_iter = 100, tol = 1e-4) {

  centroids <- initialize_centroids(data, k)
  previous_centroids <- centroids
  clusters <- NULL

  for (iteration in 1:max_iter) {
    clusters <- assign_clusters(data, centroids)
    centroids <- update_centroids(data, clusters, k)

    if (sum((centroids - previous_centroids)^2) < tol) {
      cat("Converged in", iteration, "iterations\n")
      break
    }
    previous_centroids <- centroids
    centroids_df <- as.data.frame(centroids)

    plot <- ggplot() +
      geom_point(iris, mapping=aes(Petal.Length, Petal.Width, color = as.factor(clusters), shape = Spec
      geom_star(centroids_df, mapping=aes(V3, V4), color = "black", fill="red", size =5)+
      labs(title = paste0("K-means Clustering on Iris Data, rep", iteration),
           x = "Petal Length",
           y = "Petal Width") +
      theme_minimal()
    print(plot)
  }

  list(centroids = centroids, clusters = clusters)
}
```

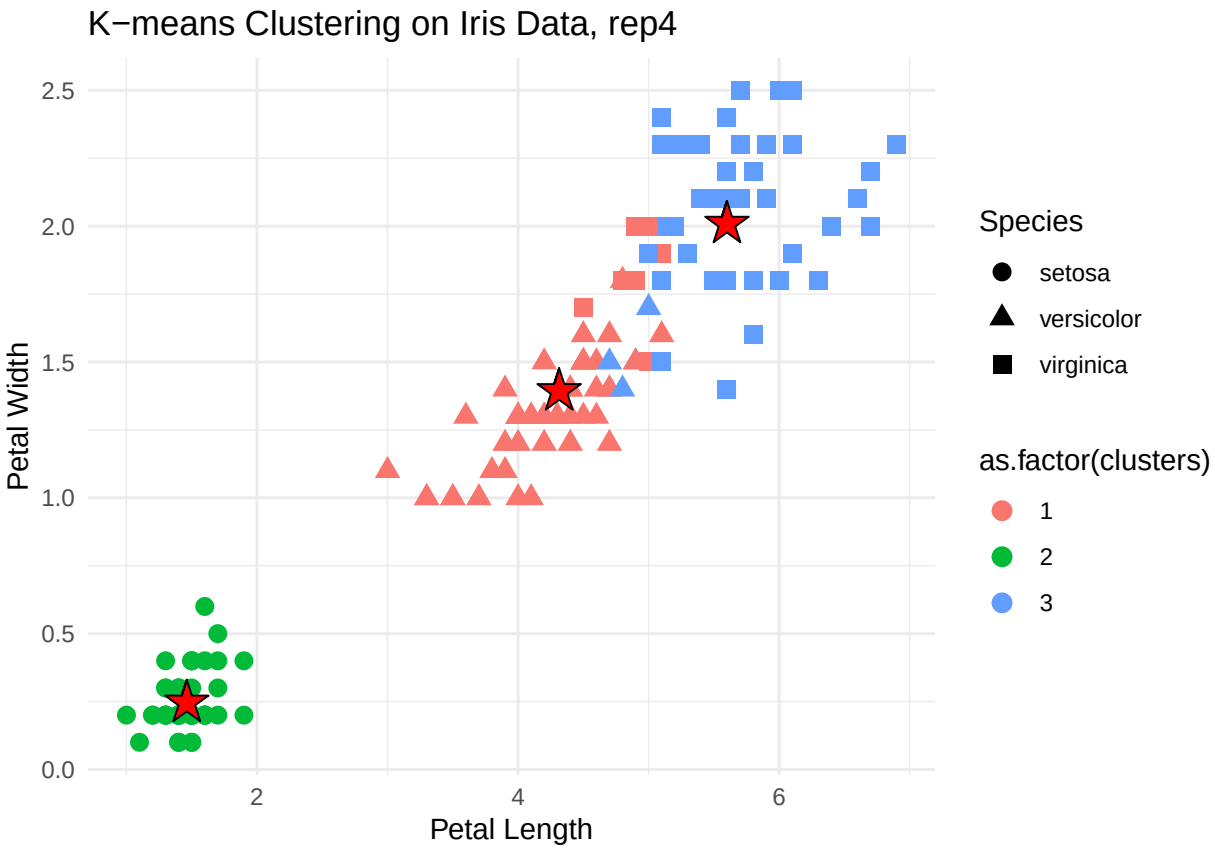
Running everything

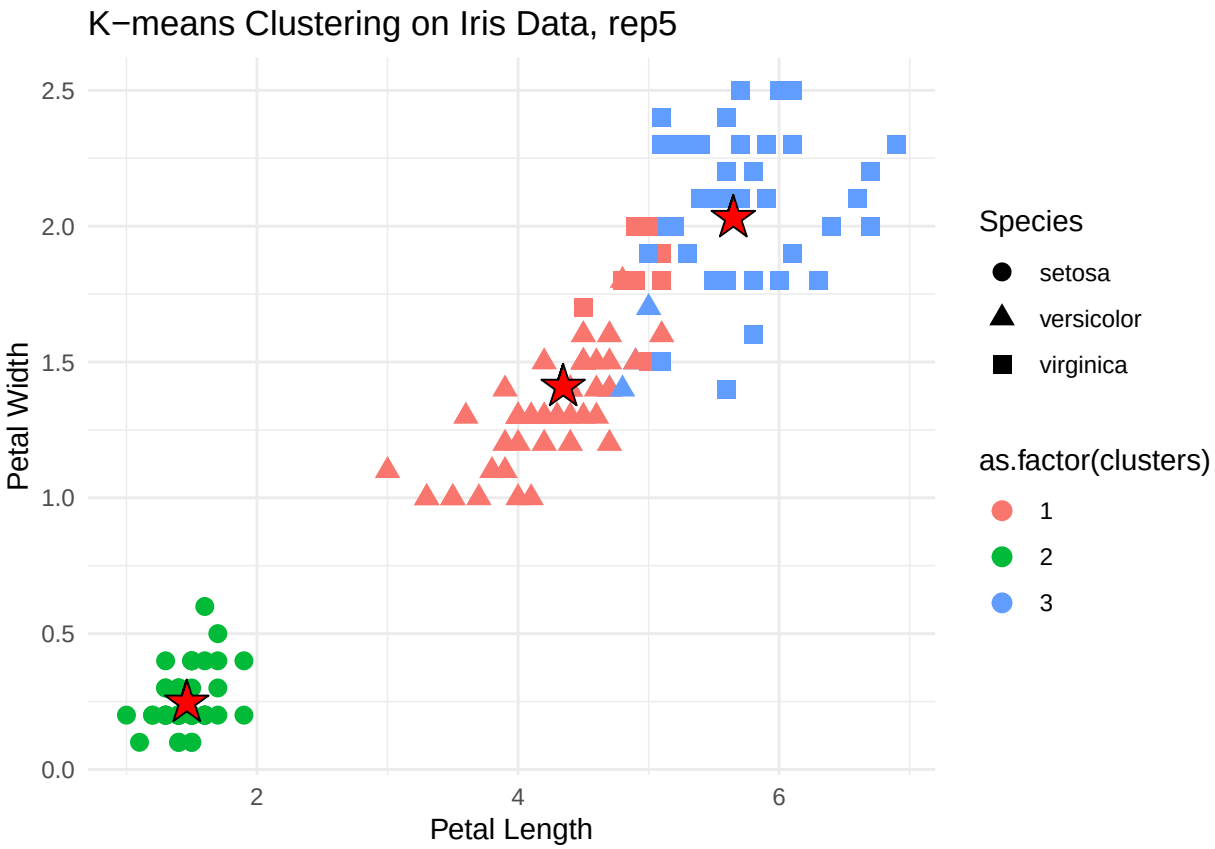
```
iris_data <- iris[, -5]

k <- 3
set.seed(5)
result <- kmeans_algorithm(iris_data, k)
```

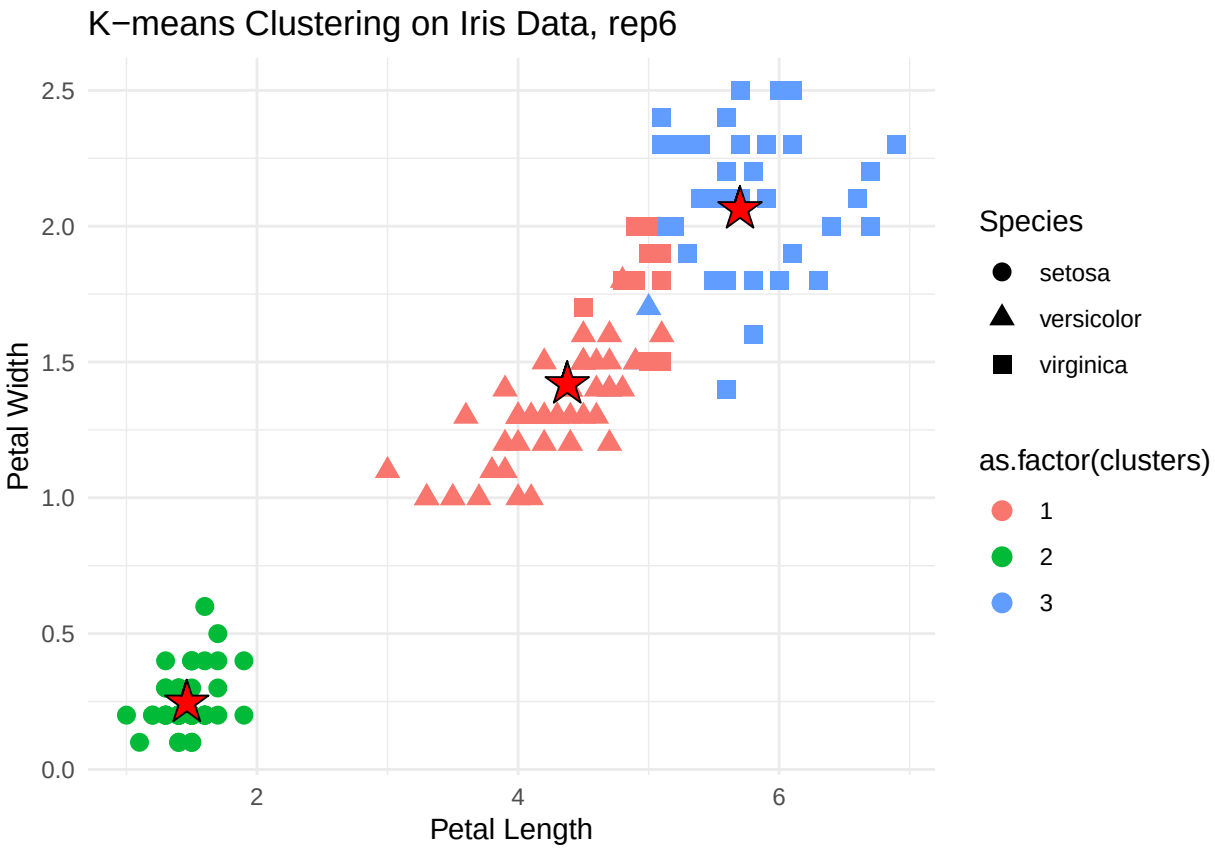

K−means Clustering on Iris Data, rep1
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
Species
setosa
versicolor
virginica
as.factor(clusters)
1
2
3

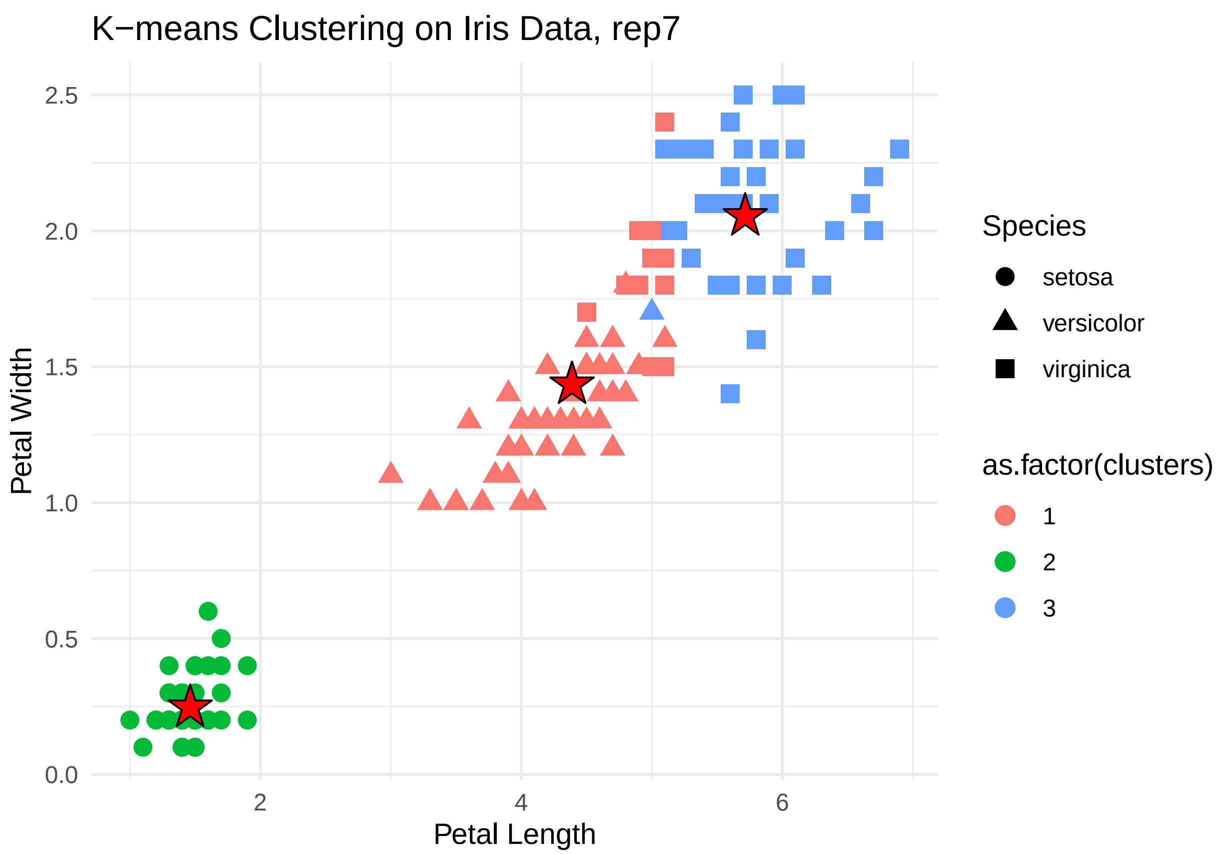

K−means Clustering on Iris Data, rep2
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
Species
setosa
versicolor
virginica
as.factor(clusters)
1
2
3


K−means Clustering on Iris Data, rep3
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
Species
setosa
versicolor
virginica
as.factor(clusters)
1
2
3


K−means Clustering on Iris Data, rep4
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
Species
setosa
versicolor
virginica
as.factor(clusters)
1
2
3


K−means Clustering on Iris Data, rep5
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
Species
setosa
versicolor
virginica
as.factor(clusters)
1
2
3


K−means Clustering on Iris Data, rep6
Petal Width
Petal Length
2.5
2.0
1.5
1.0
0.5
0.0
2
4
6
Species
setosa
versicolor
virginica
as.factor(clusters)
1
2
3




```
## Converged in 8 iterations
```

```
print(result)
```

```
## $centroids
##          [,1]     [,2]     [,3]     [,4]
## [1,] 5.883607 2.740984 4.388525 1.434426
## [2,] 5.006000 3.428000 1.462000 0.246000
## [3,] 6.853846 3.076923 5.715385 2.053846
##
## $clusters
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3 3 3 3 1 3 3 3 3
## [112] 3 3 1 1 3 3 3 3 1 3 1 3 1 3 3 1 1 3 3 3 3 3 1 3 3 3 3 1 3 3 3 1 3 3 3 1 3
## [149] 3 1
```