Clustering in R

David Culhane

2024-05-24

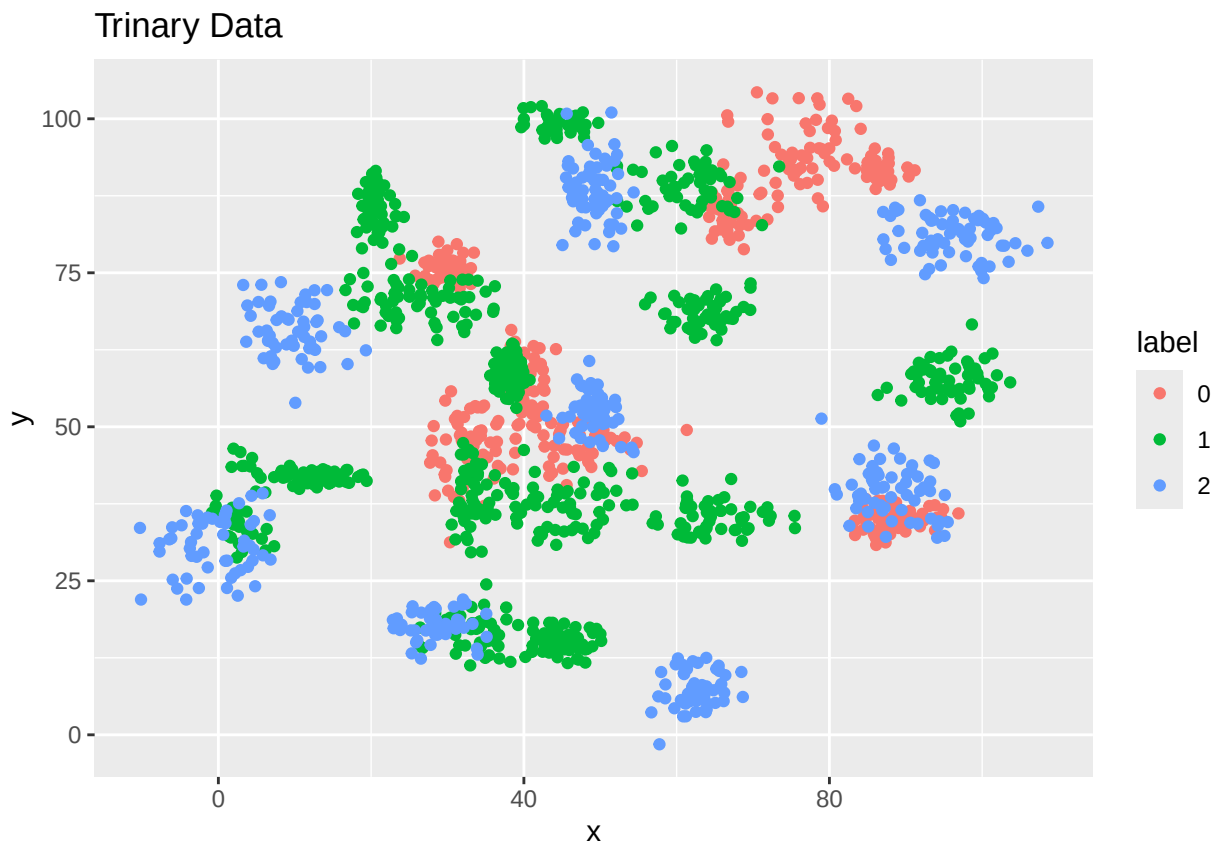
Part 1: KNN

Loading and Plotting the Data

The first part of this assignment will be using the binary and trinary classification CSVs to classify the datapoints in them using k-nearest neighbors algorithms. The first thing to do would be to load the datasets and plot them individually in order to get an idea of what we're looking at.

```
# Loading the files
library(readxl)
binary <- read.csv("binary-classifier-data.csv")
trinary <- read.csv("trinary-classifier-data.csv")
# Since the Label data will be read as numerics, it would be wise to convert
# them into factors. This will make it easier to identify points when plotted.
binary$label <- as.factor(binary$label)
trinary$label <- as.factor(trinary$label)

# Plotting the data
library(ggplot2)
binary_plot <- ggplot(binary, aes(x=x, y=y)) +
  geom_point(aes(color=label)) + labs(title='Binary Data')
binary_plot
```


```
trinary_plot <- ggplot(trinary, aes(x=x, y=y)) + geom_point(
  aes(color=label)) + labs(title='Trinary Data')
trinary_plot
```



In each of the plots, we can see that there are distinct, identifiable clusters spread throughout the data. Some clusters overlap while others do not. These plots will help with comparing the results from our clustering models generated in the next section.

Nearest Neighbors Clustering

The class library has access to K-Nearest Neighbors classification with its knn function. To use the knn function, data from each set will need to be randomly divided into training and test sets. The caTools library has an easy to use tool for splitting data between training and test sets, the sample.split funciton.

```
library(caTools)
set.seed(42)
# Splitting the binary dataset
binary_split <- sample.split(binary$label, SplitRatio = 0.75)
binary_train <- subset(binary, binary_split==TRUE)
binary_test <- subset(binary, binary_split == FALSE)

# Splitting the trinary dataset
trinary_split <- sample.split(trinary$label, SplitRatio= 0.75)
trinary_train <- subset(trinary, trinary_split==TRUE)
trinary_test <- subset(trinary, trinary_split==FALSE)
```

Now that the datasets have been split, KNN can be run using the datasets. We are looking for KNN models for k = 3, 5, 10, 15, 20, and 25 for each dataset. Given that we're working with two datasets and 6 groupings, it would be wise to write a function to do it all at once for us.

```
library(class)
knn_modeler <- function(training_set, testing_set, training_variable, k_values)
{
  # Create an empty dataframe to index into and replace values with
  predictions <- data.frame(matrix(NA, nrow = length(testing_set[,1]), ncol = length(k_values)))
  column_names <- c()
  name_base <- "Pred k ="
  # While loop to run the knn function for each value of k_values, change the
  # column name, and place the values into the dataframe.
  i <- 1
  while (i <= length(k_values))
  {
    prediction <- knn(training_set, testing_set, training_variable, k_values[i])
    predictions[,i] <- prediction
    column_names[i] <- paste(name_base, k_values[i]) #
    i <- i + 1
  }
  colnames(predictions) <- column_names # Names the columns in Predictions
  return(predictions)
}
```

The knn_modeler function takes inputs for training set, testing set, training variable, and k_values. The training_variable must be present in the training_set input. A for loop is used to run the knn function for the desired k value and then adds the results to the complete dataset by indexing it as a new column

```
k_values <- c(3, 5, 10, 15, 20, 25)
column_labels <- c('Actual', 'Pred k=3', 'Pred k=5', 'Pred k=10',
                           'Pred k=15', 'Pred k=20', 'Pred k=25')
# Beginning for the Biniary dataset
binary_preds <- knn_modeler(binary_train, binary_test, binary_train$label, k_values)
binary_preds <- cbind(binary_test$label, binary_preds)
colnames(binary_preds) <- column_labels
# Beginning for the Trinary dataset
trinary_preds <- knn_modeler(trinary_train, trinary_test, trinary_train$label, k_values)
trinary_preds <- cbind(trinary_test$label, trinary_preds)
colnames(trinary_preds) <- column_labels
```

Checking and Plotting the Accuracies

We will now want to check the accuracy for each of the six k values specified. Since running typing that 12 times would be tedious, writitng a function to do it for us would be better. We will have it return a vector, acc_vector. This will have the accuracy values of each k-value

```
accuracy_checker <- function(check_set, k_values)
{
  # Create an empty dataframe to index into and replace values with
  accuracies<- data.frame(matrix(NA, nrow = length(check_set[,1]), ncol = length(k_values)))
  column_names <- c()
  name_base <- 'Acc k ='
  acc_vector <- c()
  # I'll use another while loop to go through the k-values and test the
  # accuaries vs the test set.
```

```
  i <- 1
  while (i <= length(k_values))
  {
    accuracies[,i] <- check_set[,1] == check_set[,(i+1)]
    column_names[i] <- paste(name_base, k_values[i])
    acc_vector[i] <- sum(accuracies[,i]) / length(check_set[,1])
    i <- i+1
  }
  colnames(accuracies) <- column_names
  return(acc_vector)
}
```

We can now run accuracy_checker for each dataset and each k-value.

```
binary_accuracies <- accuracy_checker(binary_preds, k_values) * 100
binary_accuracies
```

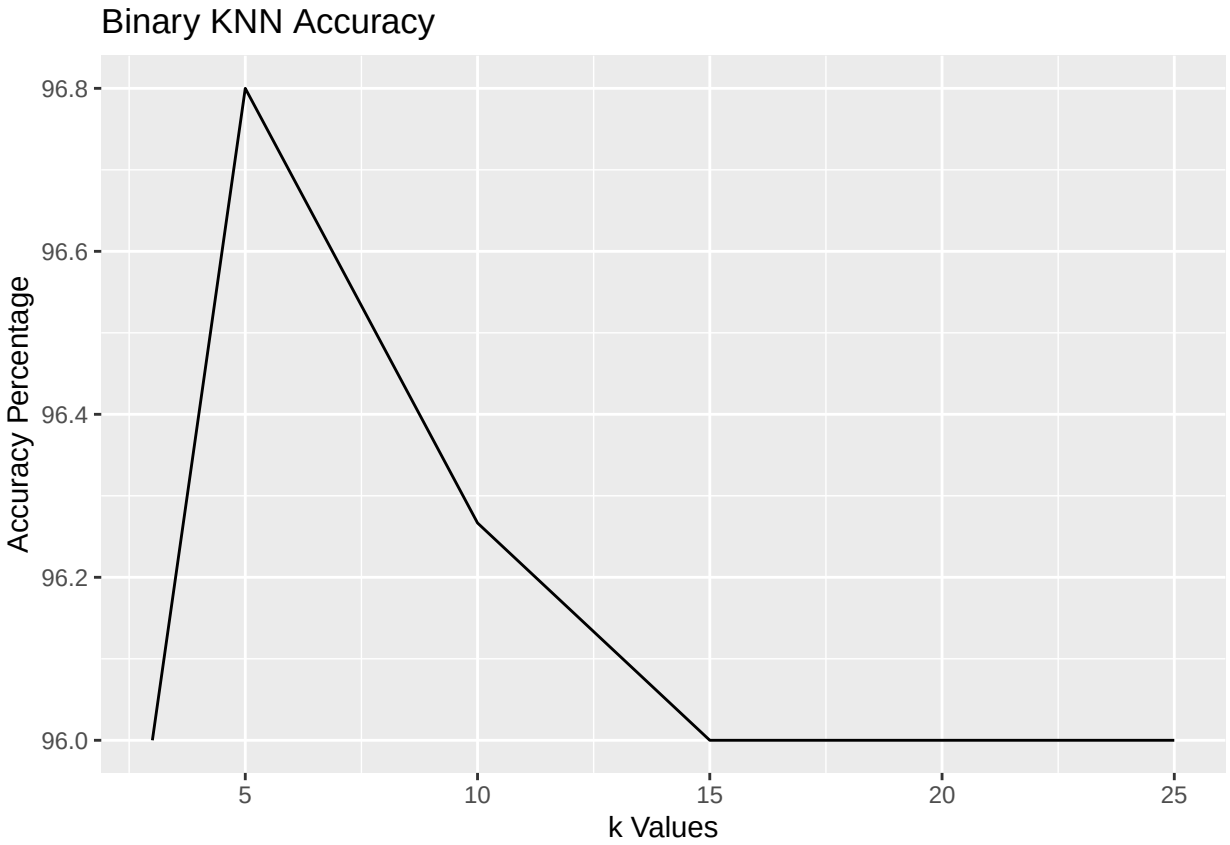
```
## [1] 96.00000 96.80000 96.26667 96.00000 96.00000 96.00000
```

```
trinary_accuracies <- accuracy_checker(trinary_preds, k_values) * 100
trinary_accuracies
```

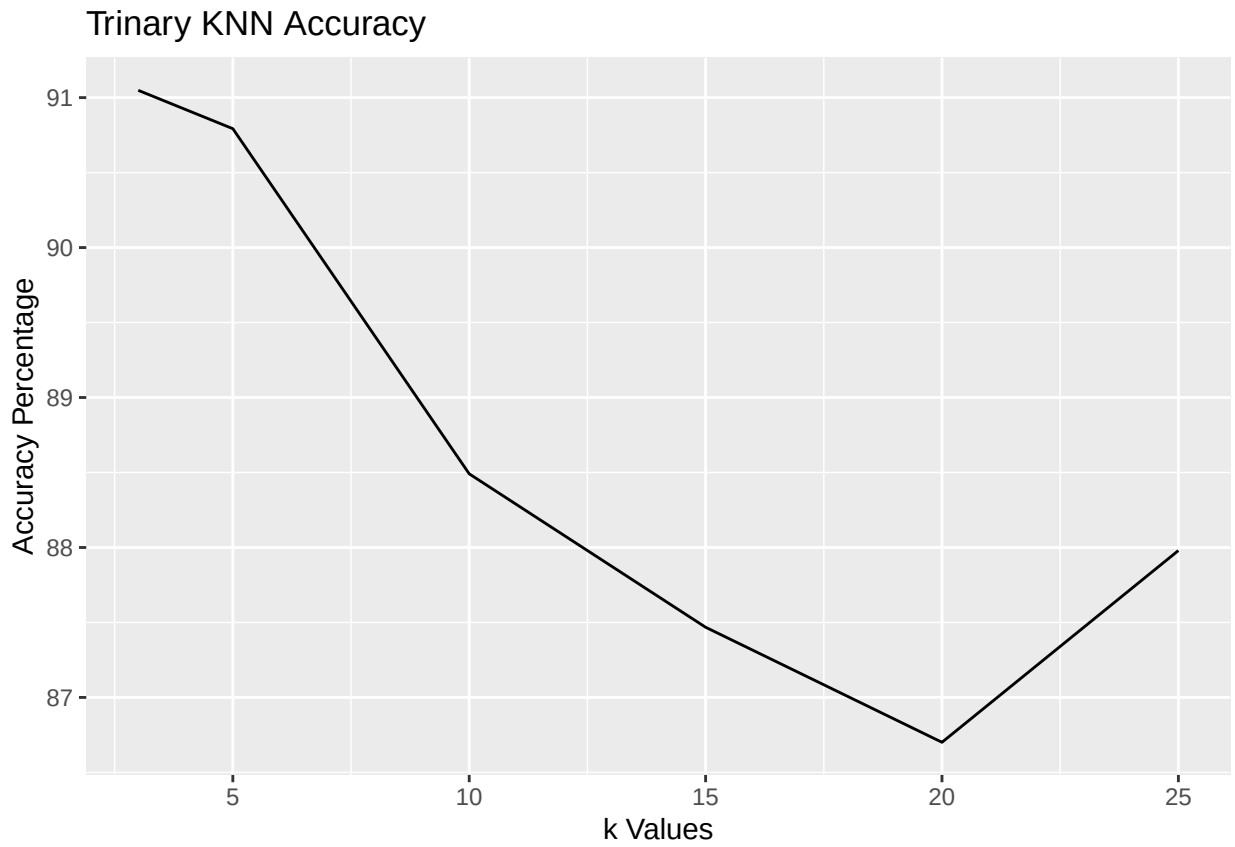
```
## [1] 91.04859 90.79284 88.49105 87.46803 86.70077 87.97954
```

Now that we have accuracy values for each k-value, we can plot them for ease of understanding. Using ggplot2 will require getting the accuraices and k-values into a dataframe. geom_line can let us see what happens to the accuracy values as k increases.

```
binary_df <- data.frame(k_values, binary_accuracies)
trinary_df <- data.frame(k_values, trinary_accuracies)

binary_bar <- ggplot(binary_df) + geom_line(aes(x=k_values, y=binary_accuracies)) + labs(x="k Values",
                                                                                        y= "Accuracy Pe
                                                                                        title="Binary K

binary_bar
```




```
trinary_bar <- ggplot(trinary_df) + geom_line(aes(x=k_values, y=trinary_accuracies)) + labs(x="k Values
                                                                                            y= "Accuracy
                                                                                            title="Trina

trinary_bar
```



We see a small decrease in overall accuracy when the number of possible outcomes increases. The binary accuracy stays around 96% for all k-values used. For the same k-values in the trinary dataset, we see accuracy decrease as the value of k increases.

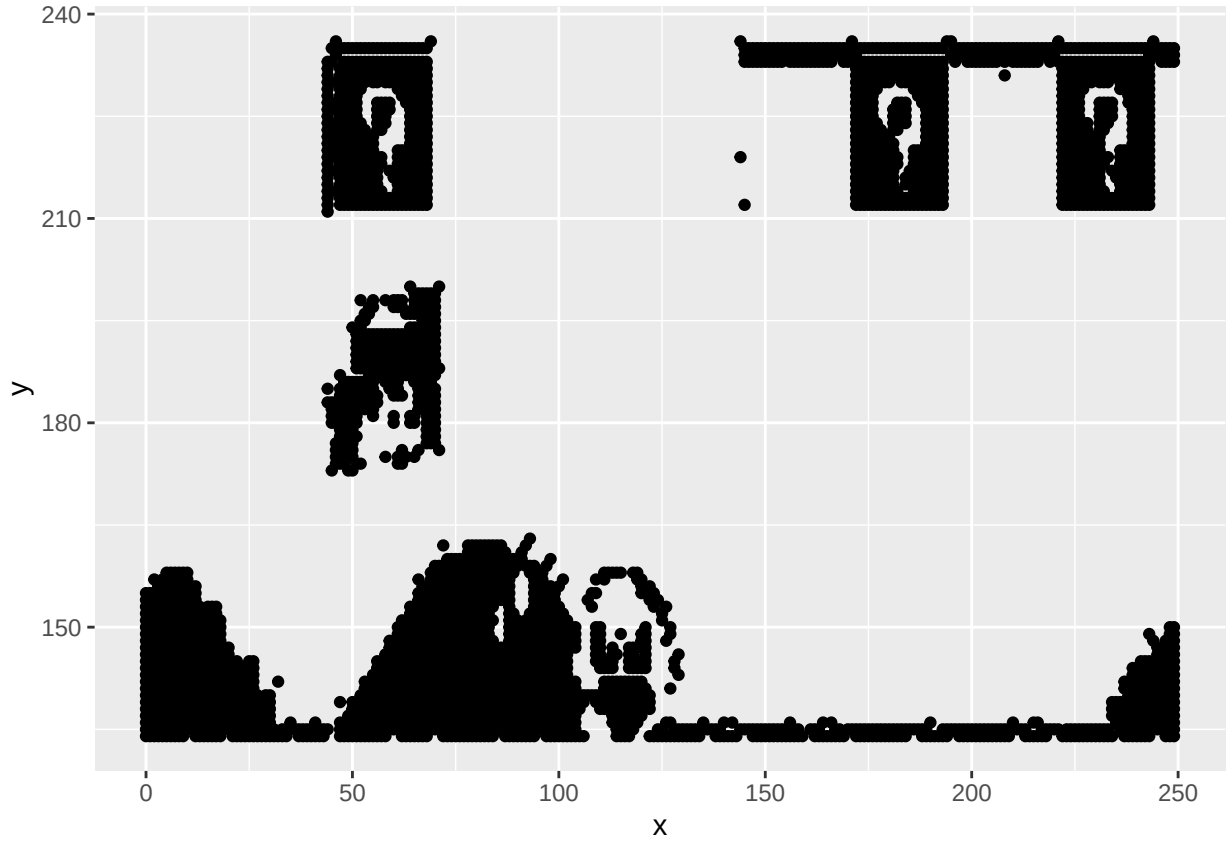
Part 2: Clustering

This section will deal with clustering as an unsupervised (unlabeled) analysis technique. We'll be loading a new dataset, the clustering-data CSV for this section.

```
clustering <- read.csv("clustering-data.csv")
```

The first thing to do is to check out the data. We can do this with a simple scatterplot.

```
ggplot(clustering) + geom_point(aes(x=x, y=y))
```



This obviously looks like a level from the original Super Mario Bros. Ideally, we can have our clusters be identifiable here.

Clustering here will be done using k-means, with k-values ranging from 2 to 12. For each k-value, we will want to perform the clustering and plot the clusters. Unfortunately, I couldn't think of a way to write a loop to do it that would also be able to create the variable name for each loop.

```
set.seed(1985)
library(useful)

mario2 <- kmeans(x=clustering, centers=2)
mario3 <- kmeans(x=clustering, centers=3)
mario4 <- kmeans(x=clustering, centers=4)
mario5 <- kmeans(x=clustering, centers=5)
mario6 <- kmeans(x=clustering, centers=6)
mario7 <- kmeans(x=clustering, centers=7)
mario8 <- kmeans(x=clustering, centers=8)
mario9 <- kmeans(x=clustering, centers=9)
mario10 <- kmeans(x=clustering, centers=10)
mario11 <- kmeans(x=clustering, centers=11)
mario12 <- kmeans(x=clustering, centers=12)
```

With the clusters now calculated, we can find mean distance to the cluster's center for each k-value. Doing this will require us to create a series of datasets with the original point information, which center each point belongs to, and the location of each center.

```r
# Creating the individual datasets for each k-means clustering run
k2 <- cbind(clustering, mario2[["cluster"]])
k3 <- cbind(clustering, mario3[["cluster"]])
k4 <- cbind(clustering, mario4[["cluster"]])
k5 <- cbind(clustering, mario5[["cluster"]])
k6 <- cbind(clustering, mario6[["cluster"]])
k7 <- cbind(clustering, mario7[["cluster"]])
k8 <- cbind(clustering, mario8[["cluster"]])
k9 <- cbind(clustering, mario9[["cluster"]])
k10 <- cbind(clustering, mario10[["cluster"]])
k11 <- cbind(clustering, mario11[["cluster"]])
k12 <- cbind(clustering, mario12[["cluster"]])
# Correcting the column name for the cluster column
colnames(k2) <- c("x", "y", "Cluster")
colnames(k3) <- c("x", "y", "Cluster")
colnames(k4) <- c("x", "y", "Cluster")
colnames(k5) <- c("x", "y", "Cluster")
colnames(k6) <- c("x", "y", "Cluster")
colnames(k7) <- c("x", "y", "Cluster")
colnames(k8) <- c("x", "y", "Cluster")
colnames(k9) <- c("x", "y", "Cluster")
colnames(k10) <- c("x", "y", "Cluster")
colnames(k11) <- c("x", "y", "Cluster")
colnames(k12) <- c("x", "y", "Cluster")
# Adding the cluster center coordinates as fields
k2$`Cluster x` <- mario2[["centers"]][k2$Cluster,1]
k2$`Cluster y` <- mario2[["centers"]][k2$Cluster,2]
k3$`Cluster x` <- mario3[["centers"]][k3$Cluster,1]
k3$`Cluster y` <- mario3[["centers"]][k3$Cluster,2]
k4$`Cluster x` <- mario4[["centers"]][k4$Cluster,1]
k4$`Cluster y` <- mario4[["centers"]][k4$Cluster,2]
k5$`Cluster x` <- mario5[["centers"]][k5$Cluster,1]
k5$`Cluster y` <- mario5[["centers"]][k5$Cluster,2]
k6$`Cluster x` <- mario6[["centers"]][k6$Cluster,1]
k6$`Cluster y` <- mario6[["centers"]][k6$Cluster,2]
k7$`Cluster x` <- mario7[["centers"]][k7$Cluster,1]
k7$`Cluster y` <- mario7[["centers"]][k7$Cluster,2]
k8$`Cluster x` <- mario8[["centers"]][k8$Cluster,1]
k8$`Cluster y` <- mario8[["centers"]][k8$Cluster,2]
k9$`Cluster x` <- mario9[["centers"]][k9$Cluster,1]
k9$`Cluster y` <- mario9[["centers"]][k9$Cluster,2]
k10$`Cluster x` <- mario10[["centers"]][k10$Cluster,1]
k10$`Cluster y` <- mario10[["centers"]][k10$Cluster,2]
k11$`Cluster x` <- mario11[["centers"]][k11$Cluster,1]
k11$`Cluster y` <- mario11[["centers"]][k11$Cluster,2]
k12$`Cluster x` <- mario12[["centers"]][k12$Cluster,1]
k12$`Cluster y` <- mario12[["centers"]][k12$Cluster,2]
# Now we can calculate the distances from center for each point in each k-means clustering
k2$Distance <- sqrt((k2$x - k2$`Cluster x`)**2 + (k2$y - k2$`Cluster y`)**2)
k3$Distance <- sqrt((k3$x - k3$`Cluster x`)**2 + (k3$y - k3$`Cluster y`)**2)
k4$Distance <- sqrt((k4$x - k4$`Cluster x`)**2 + (k4$y - k4$`Cluster y`)**2)
k5$Distance <- sqrt((k5$x - k5$`Cluster x`)**2 + (k5$y - k5$`Cluster y`)**2)
k6$Distance <- sqrt((k6$x - k6$`Cluster x`)**2 + (k6$y - k6$`Cluster y`)**2)
```

```
k7$Distance <- sqrt((k7$x - k7$`Cluster x`)**2 + (k7$y - k7$`Cluster y`)**2)
k8$Distance <- sqrt((k8$x - k8$`Cluster x`)**2 + (k8$y - k8$`Cluster y`)**2)
k9$Distance <- sqrt((k9$x - k9$`Cluster x`)**2 + (k9$y - k9$`Cluster y`)**2)
k10$Distance <- sqrt((k10$x - k10$`Cluster x`)**2 + (k10$y - k10$`Cluster y`)**2)
k11$Distance <- sqrt((k11$x - k11$`Cluster x`)**2 + (k11$y - k11$`Cluster y`)**2)
k12$Distance <- sqrt((k12$x - k12$`Cluster x`)**2 + (k12$y - k12$`Cluster y`)**2)
```

Now that our datasets are constructed, we can use dplyr's summarize function to get us the mean distance from each cluster center to its associated points for each k-means clustering run.

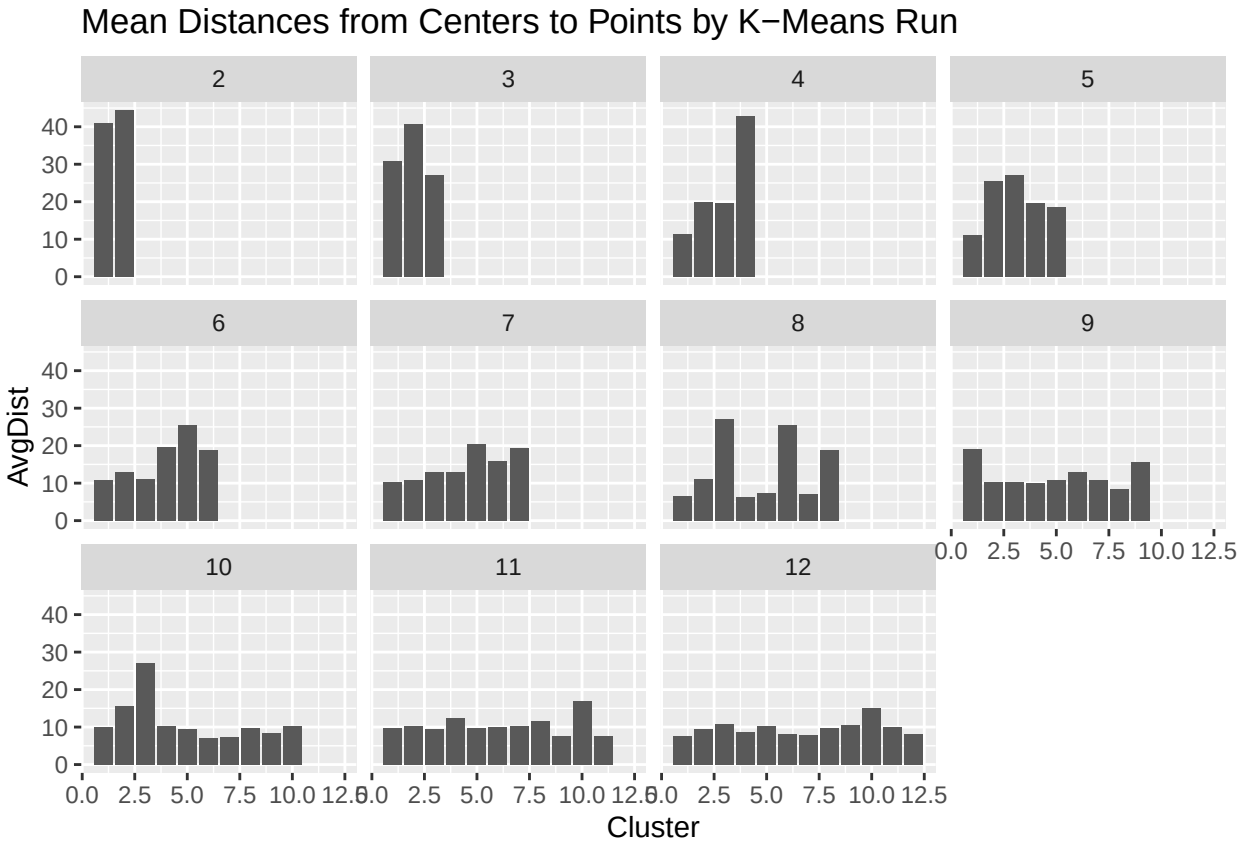
```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(magrittr)
k2means <- k2 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k3means <- k3 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k4means <- k4 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k5means <- k5 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k6means <- k6 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k7means <- k7 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k8means <- k8 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k9means <- k9 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k10means <- k10 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k11means <- k11 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
k12means <- k12 %>% group_by(Cluster) %>% summarize(AvgDist=mean(Distance))
```

If we want to plot these, using geom_col bar graphs would work to see how the distances change. To make that easier, we will want to bring all of the means datasets into one dataframe and bring the k values in as a column too.

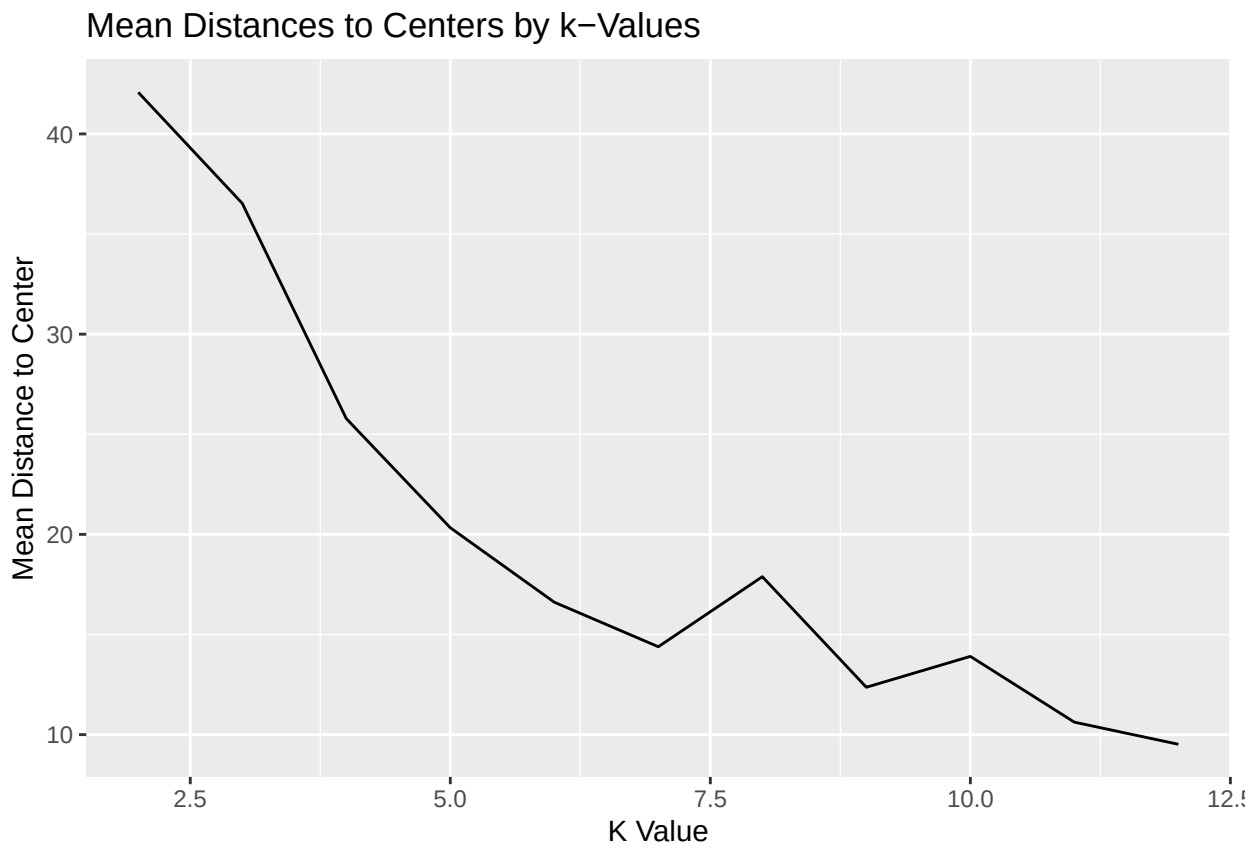
```
means_by_k_run <- rbind(k2means, k3means, k4means, k5means, k6means, k7means, k8means,
                        k9means, k10means, k11means, k12means)
`K Value` <- as.factor(c(2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6,
                         7, 7, 7, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8, 8, 9, 9, 9, 9, 9,
                         9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 11, 11, 11,
                         11, 11, 11, 11, 11, 11, 11, 12, 12, 12, 12, 12, 12, 12, 12,
                         12, 12, 12, 12))
means_by_k_run <- cbind(`K Value`, means_by_k_run)

ggplot(means_by_k_run, aes(x=Cluster, y=AvgDist)) + geom_col() + facet_wrap(~`K Value`) +
  labs(title="Mean Distances from Centers to Points by K-Means Run")
```



We can see from the plot that as the number of centers employed increases, the average distance from each center to the points in the cluster decreases. It is worth noting that there are some irregularities though, where some clusters have large distances and others have small distances, like when k = 8 (8 clusters used).

The total mean distance to centers can also be used to evaluate each k-means run.

```
k2mean <- mean(k2$Distance)
k3mean <- mean(k3$Distance)
k4mean <- mean(k4$Distance)
k5mean <- mean(k5$Distance)
k6mean <- mean(k6$Distance)
k7mean <- mean(k7$Distance)
k8mean <- mean(k8$Distance)
k9mean <- mean(k9$Distance)
k10mean <- mean(k10$Distance)
k11mean <- mean(k11$Distance)
k12mean <- mean(k12$Distance)
ks <- c(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)
all_means <- c(k2mean, k3mean, k4mean, k5mean, k6mean, k7mean, k8mean, k9mean,
               k10mean, k11mean, k12mean)
means_df <- data.frame(ks, all_means)

ggplot(means_df) + geom_line(aes(x=ks, y=all_means)) +
  labs(x="K Value", y="Mean Distance to Center", title="Mean Distances to Centers by k-Values")
```

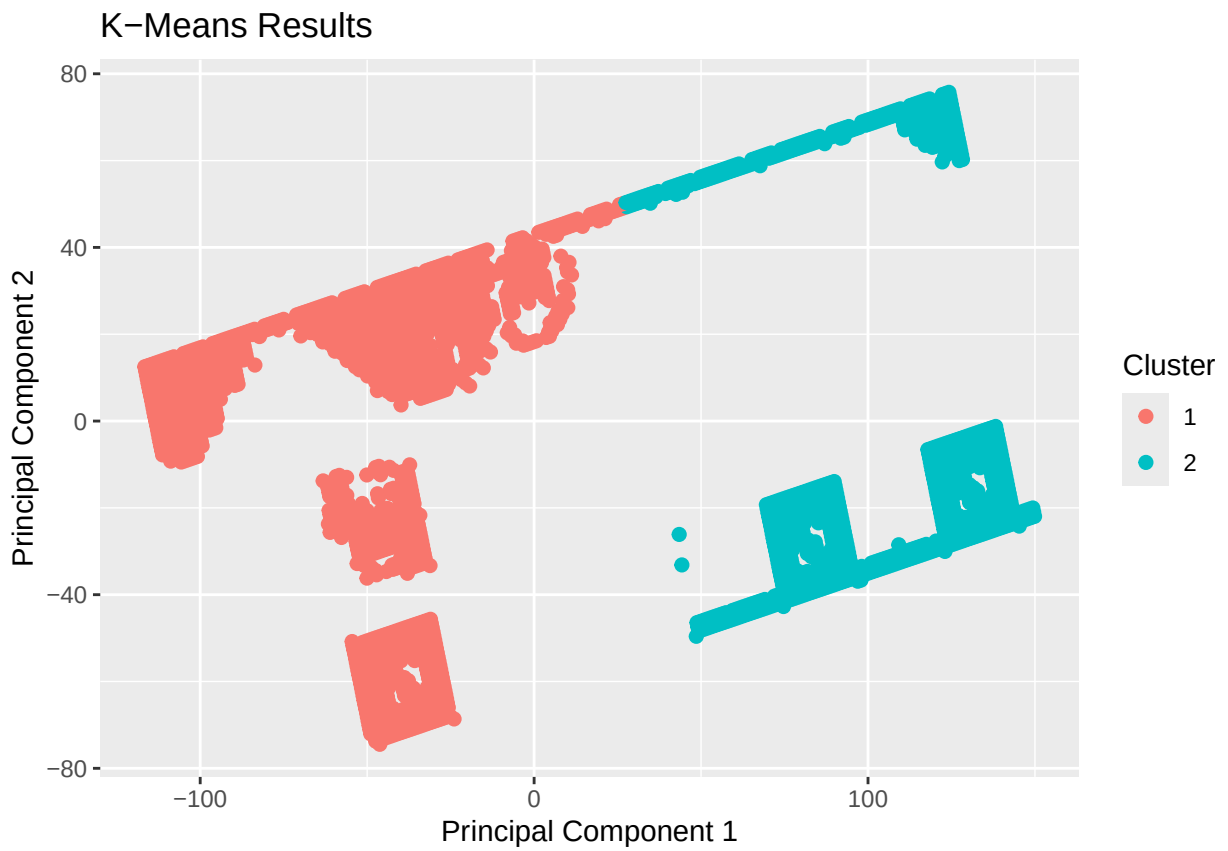


From this graph we can see that, generally, as the value of k (the number of clusters employed) increases, the mean distance from any point in the plot to the center of its cluster tends to decrease. Though we do see a couple of aberrations at k = 8 and k = 10. Their distances actually increased compared to the k-value that came before them.

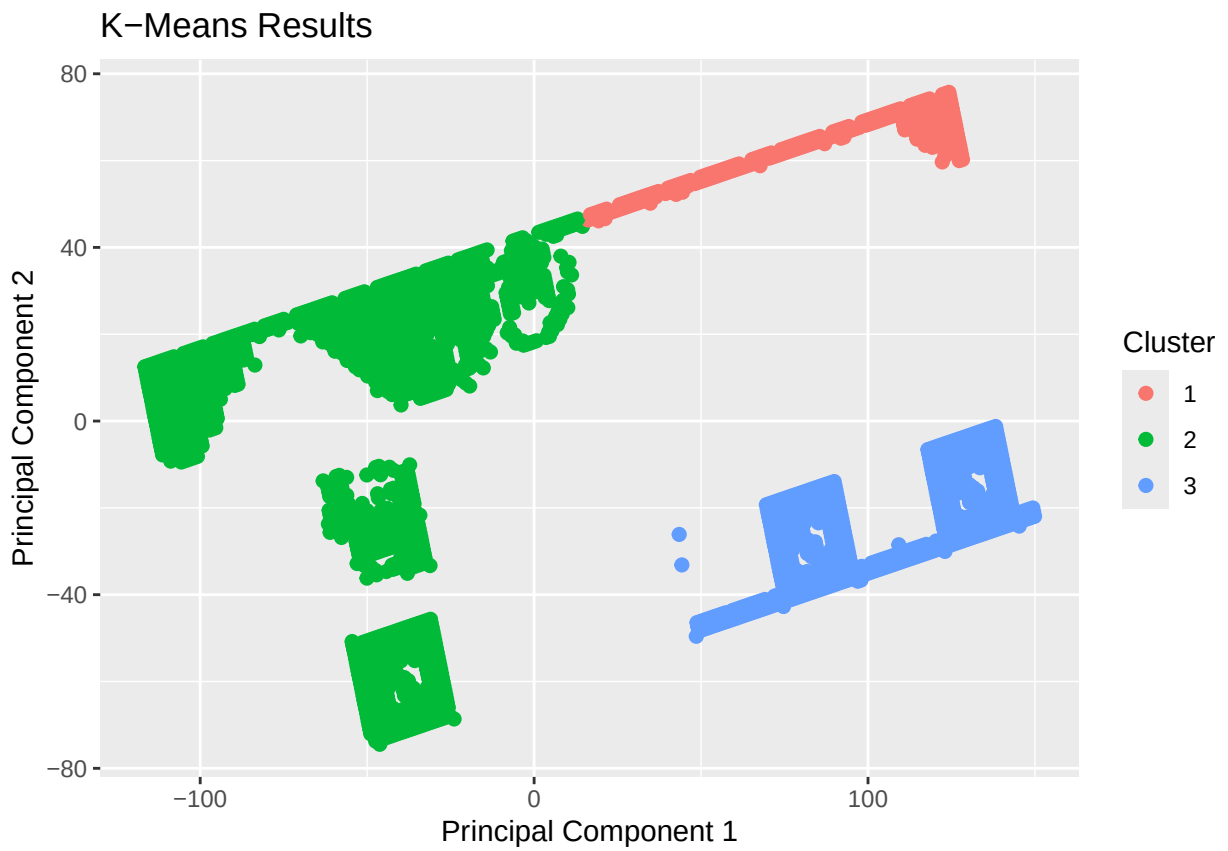
Choosing the best number of clusters to use depends on the "elbow point" in the graph above. In the graph here, two elbows can be seen. One is at k = 7 and the other at k = 9. In the interest of computational efficiency, k = 7 may be a better choice than k = 9 or 11 or 12. The distances for 11 and 12 may both be lower than those for 7 and 9, but nore is not always better.

Below, we can see what the clusters themselves looked like. Since plotting the clusters is computationally expensive, I decided to do that last.

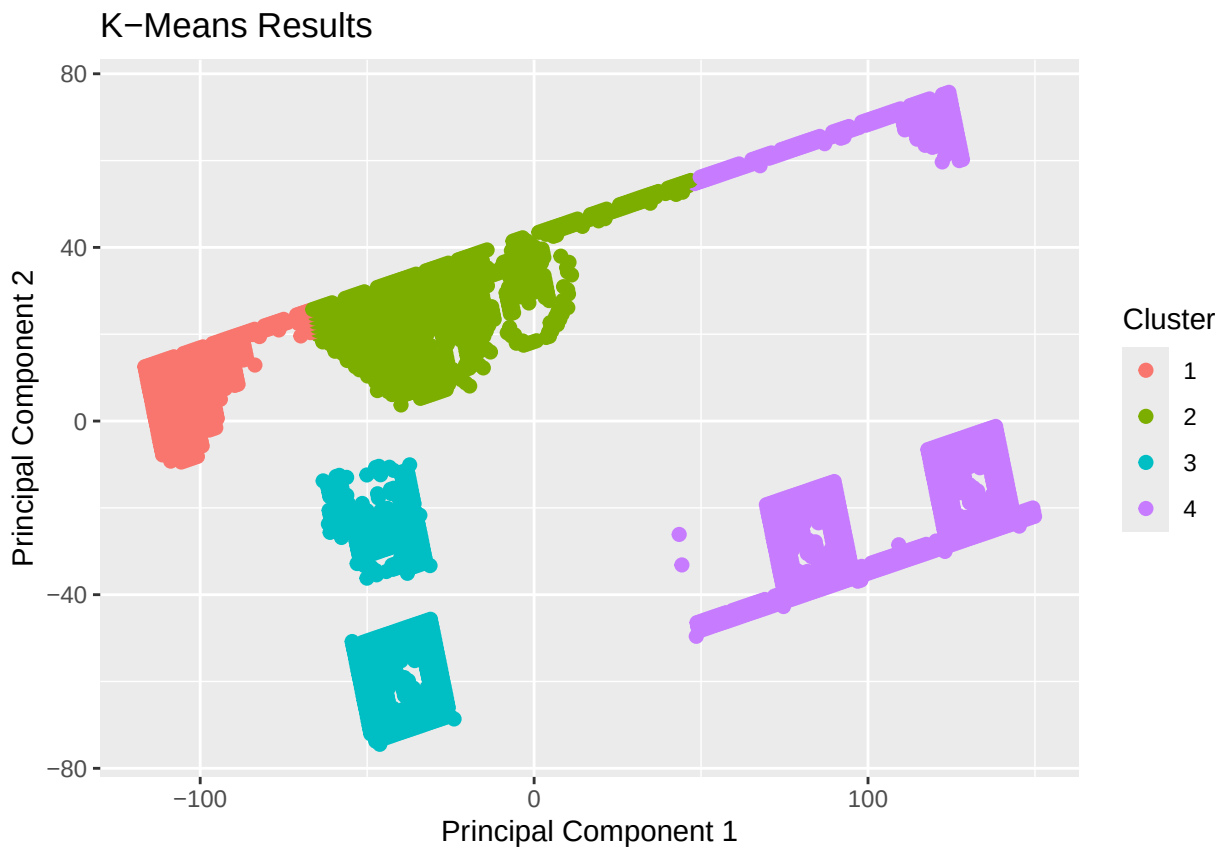
```
plot(mario2, data=clustering)
```



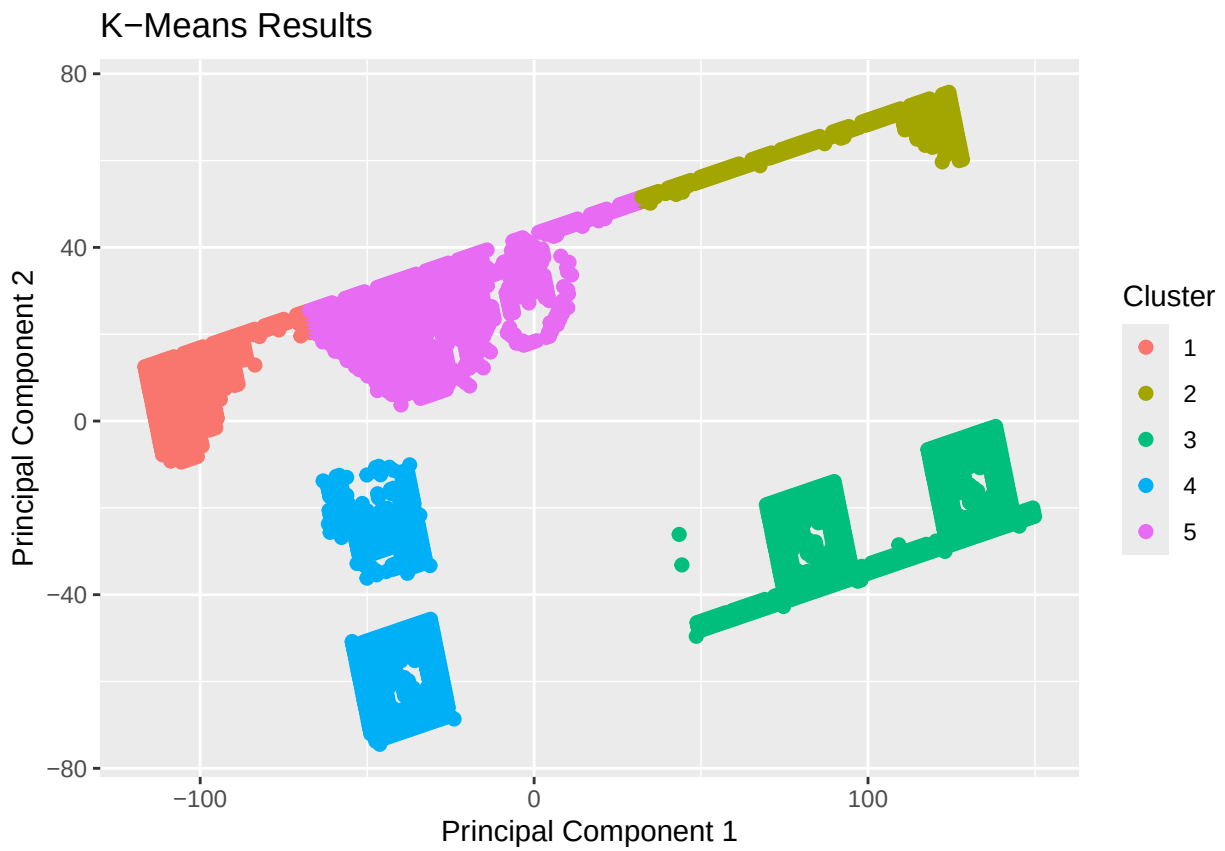

```
plot(mario3, data=clustering)
```



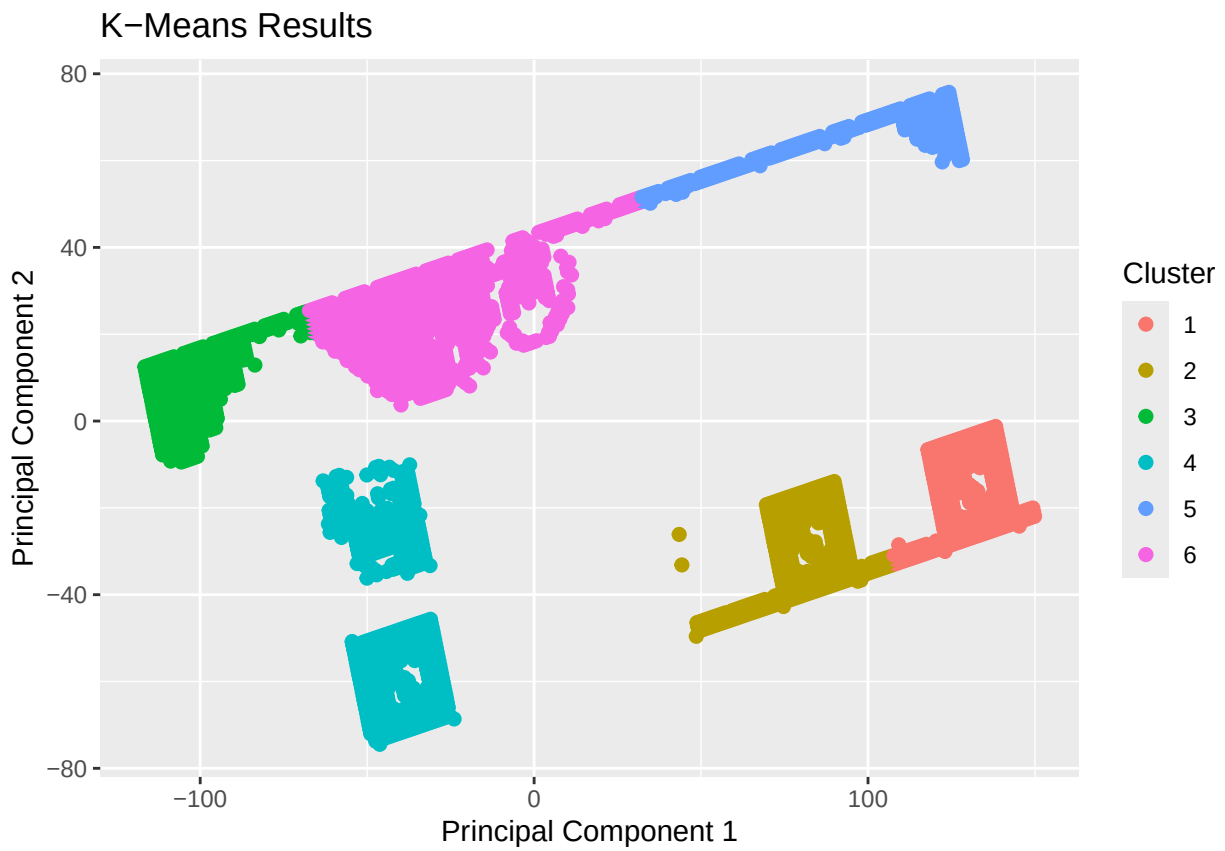

```
plot(mario4, data=clustering)
```



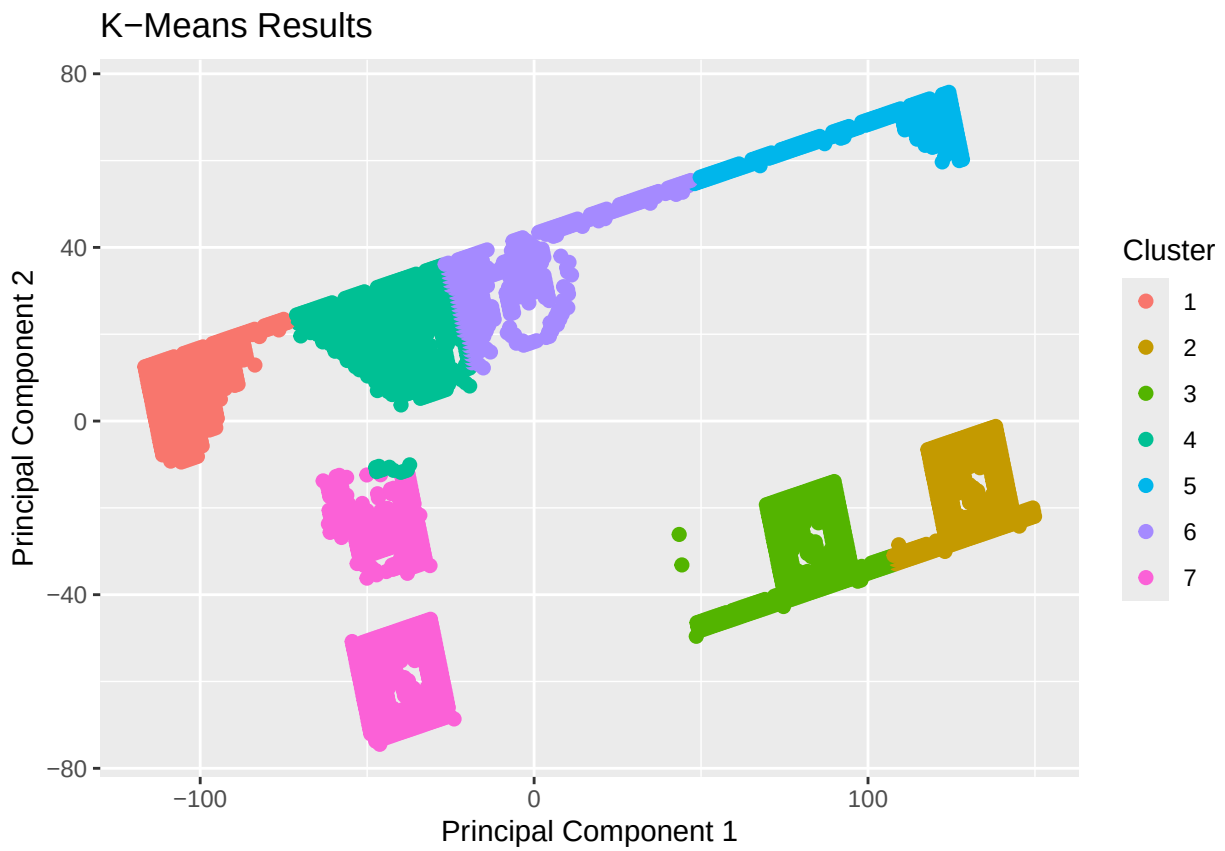

```
plot(mario5, data=clustering)
```



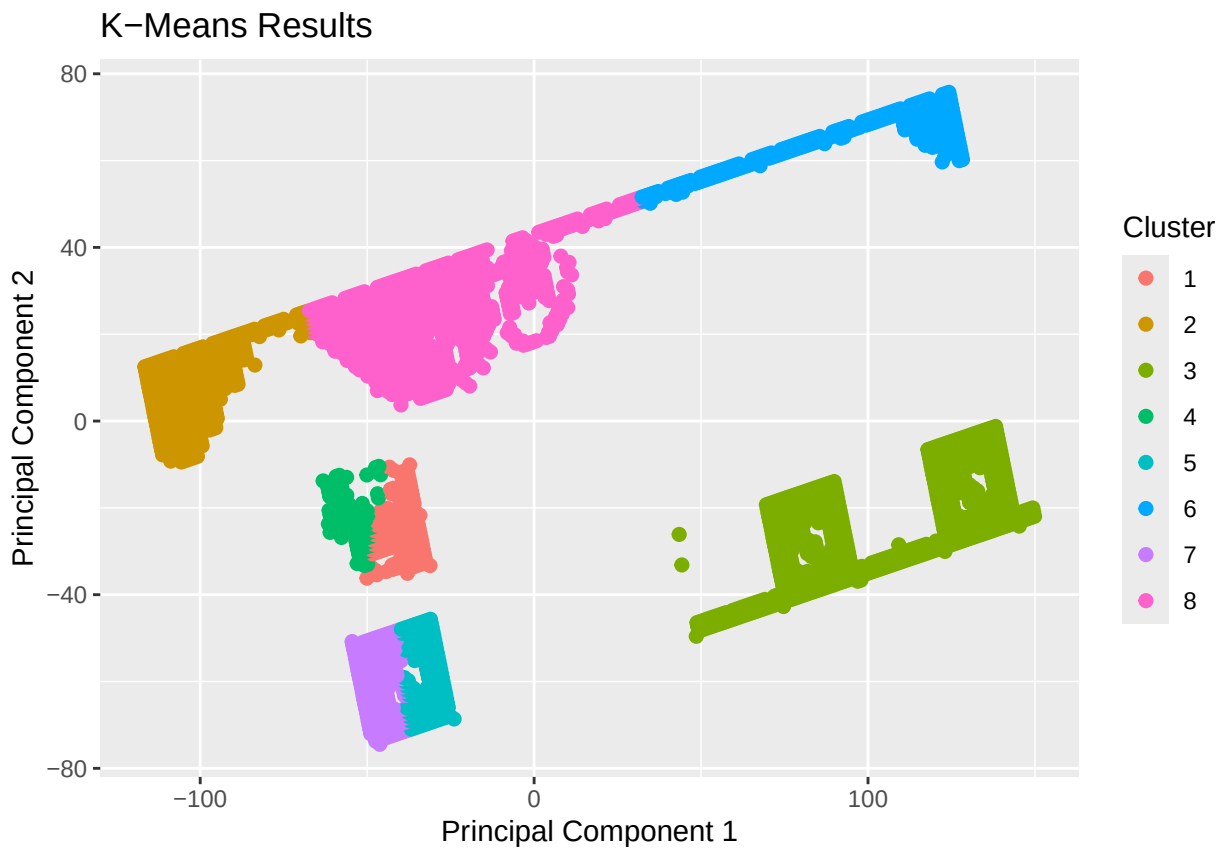

```
plot(mario6, data=clustering)
```



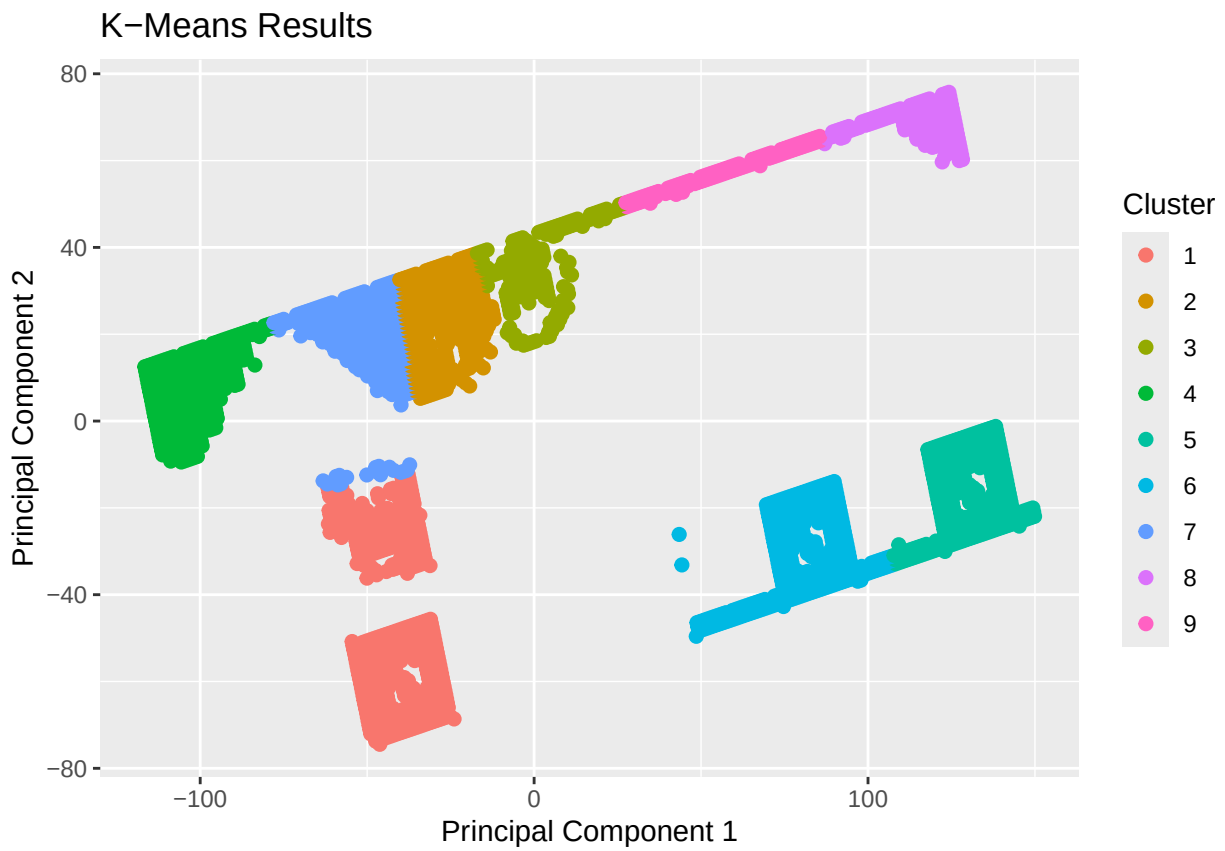

```
plot(mario7, data=clustering)
```



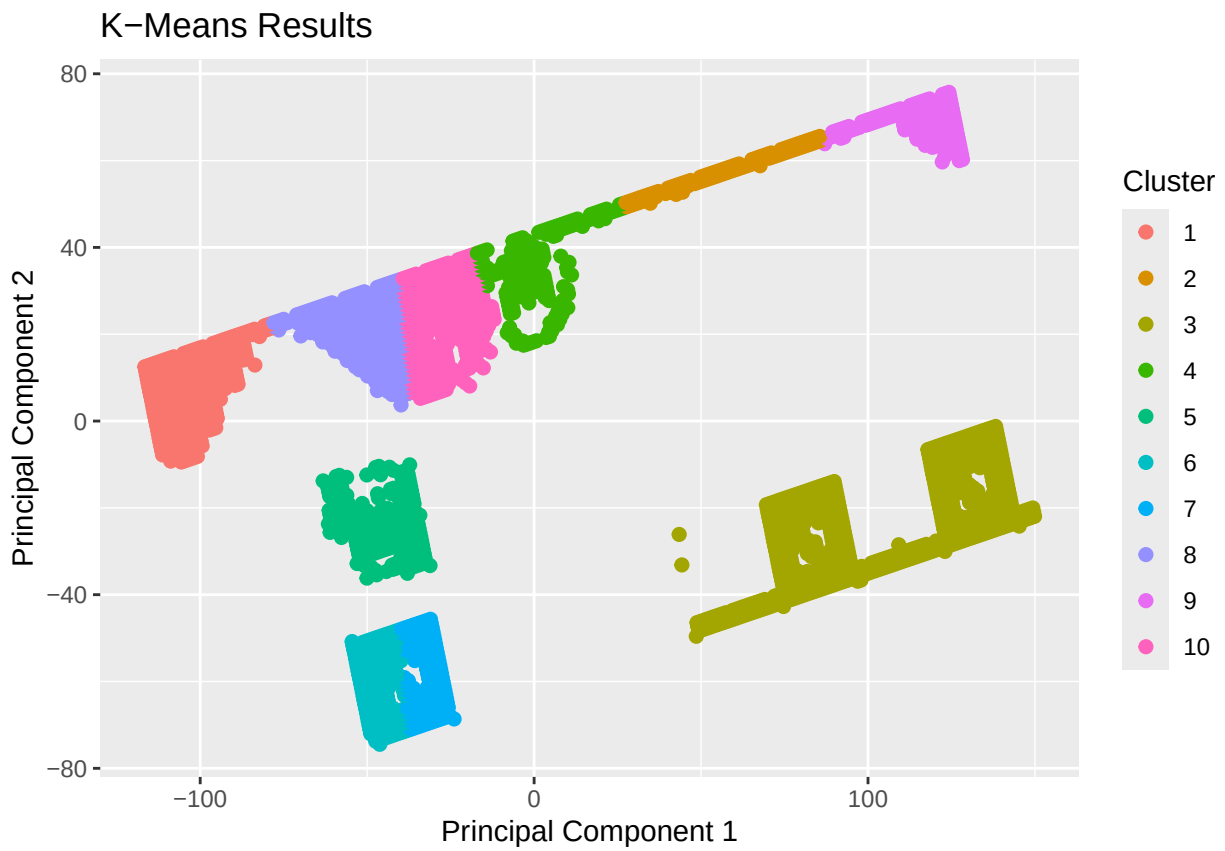

```
plot(mario8, data=clustering)
```



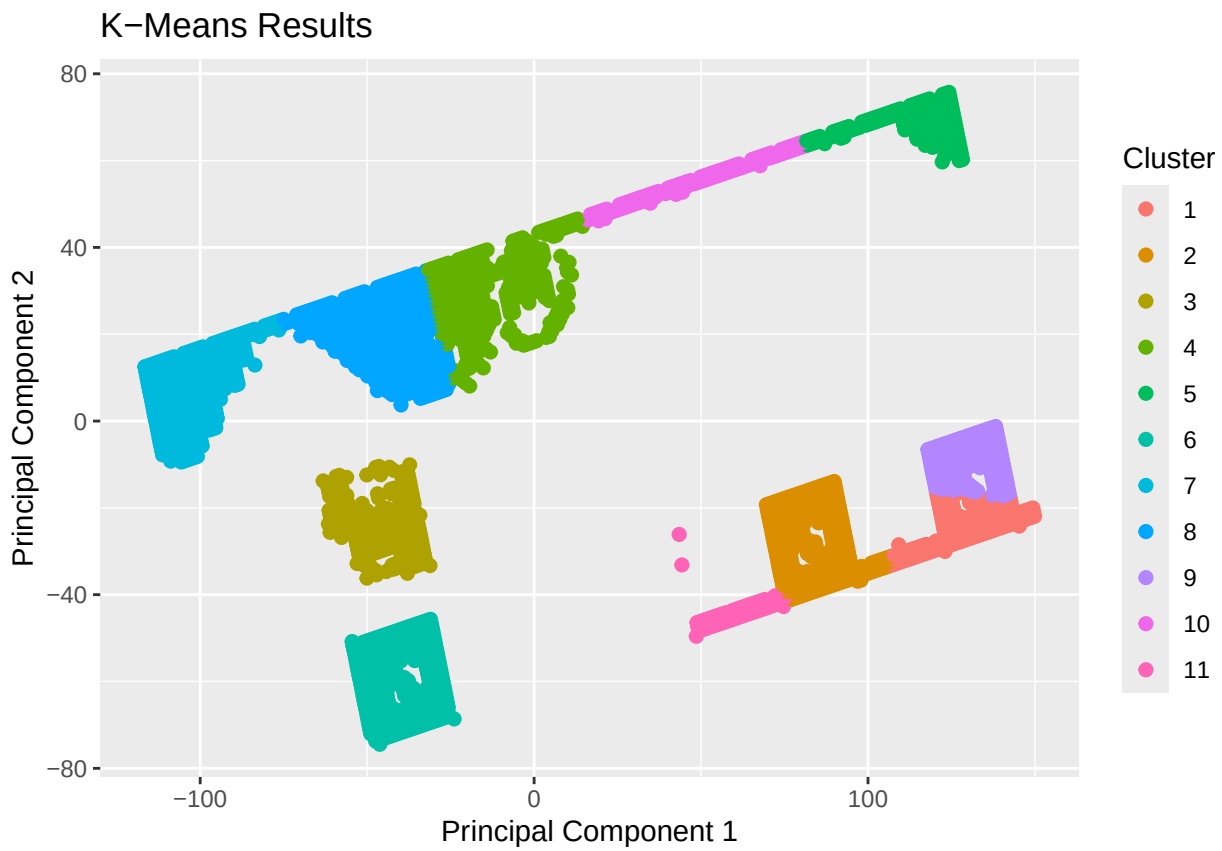

```
plot(mario9, data=clustering)
```



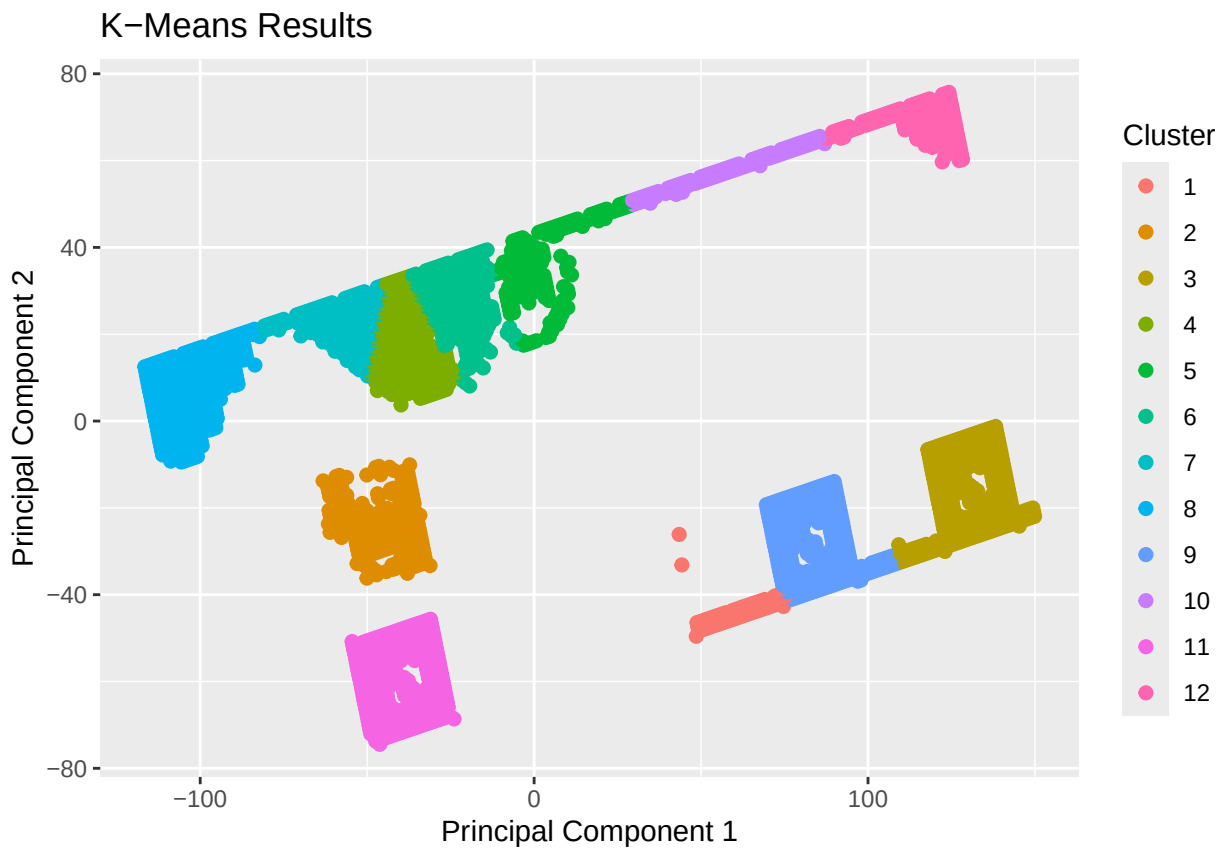

```
plot(mario10, data=clustering)
```




```
plot(mario11, data=clustering)
```




```
plot(mario12, data=clustering)
```


K−Means Results
Principal Component 2
80
40
0
−40
−80
−100
0
100
Principal Component 1
Cluster
1
2
3
4
5
6
7
8
9
10
11
12