Week 11 - 12

Kimberly Cable

06-04-2022

Introduction to Machine Learning

a. These assignments are here to provide you with an introduction to the "Data Science" use for these tools. This is your future. It may seem confusing and weird right now but it hopefully seems far less so than earlier in the semester. Attempt these homework assignments. You will not be graded on your answer but on your approach. This should be a, "Where am I on learning this stuff" check. If you can't get it done, please explain why.

b. Include all of your answers in a R Markdown report.

c. Regression algorithms are used to predict numeric quantity while classification algorithms predict categorical outcomes. A spam filter is an example use case for a classification algorithm. The input dataset is emails labeled as either spam (i.e. junk emails) or ham (i.e. good emails). The classification algorithm uses features extracted from the emails to learn which emails fall into which category.

d. In this problem, you will use the nearest neighbors algorithm to fit a model on two simplified datasets. The first dataset (found in binary-classifier-data.csv) contains three variables; label, x, and y. The label variable is either 0 or 1 and is the output we want to predict using the x and y variables (You worked with this dataset last week!). The second dataset (found in trinary-classifier-data.csv) is similar to the first dataset except that the label variable can be 0, 1, or 2.

```r
## Load the binary classifier data
binary_df <- read.csv("data/binary-classifier-data.csv",
    header = TRUE,
    stringsAsFactors = FALSE)

head(binary_df)
```

```
##   label        x        y
## 1     0 70.88469 83.17702
## 2     0 74.97176 87.92922
## 3     0 73.78333 92.20325
## 4     0 66.40747 81.10617
```

```
## 5     0 69.07399 84.53739
## 6     0 72.23616 86.38403
```

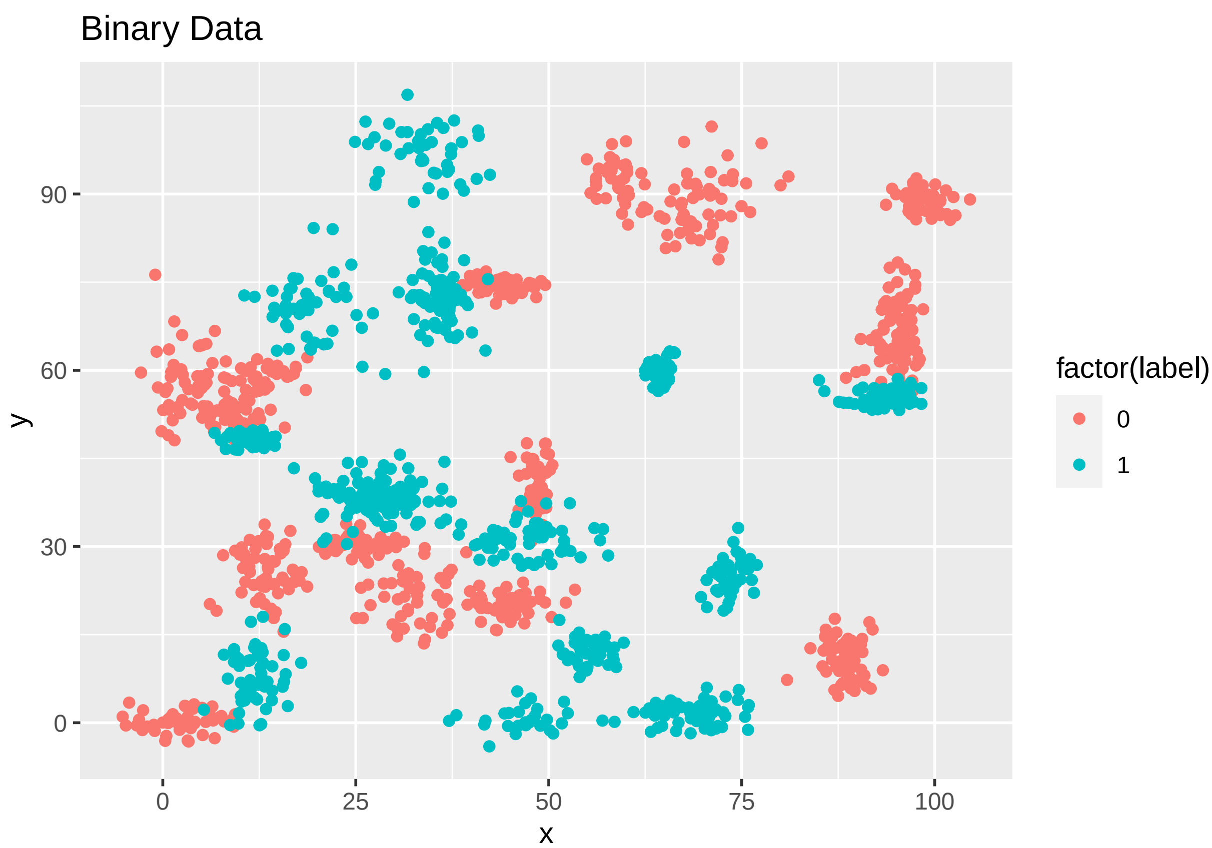
```r
## Load the trinary classifier data
trinary_df <- read.csv("data/trinary-classifier-data.csv",
    header = TRUE,
    stringsAsFactors = FALSE)

head(trinary_df)
```

```
##   label        x        y
## 1     0 30.08387 39.63094
## 2     0 31.27613 51.77511
## 3     0 34.12138 49.27575
## 4     0 32.58222 41.23300
## 5     0 34.65069 45.47956
## 6     0 33.80513 44.24656
```

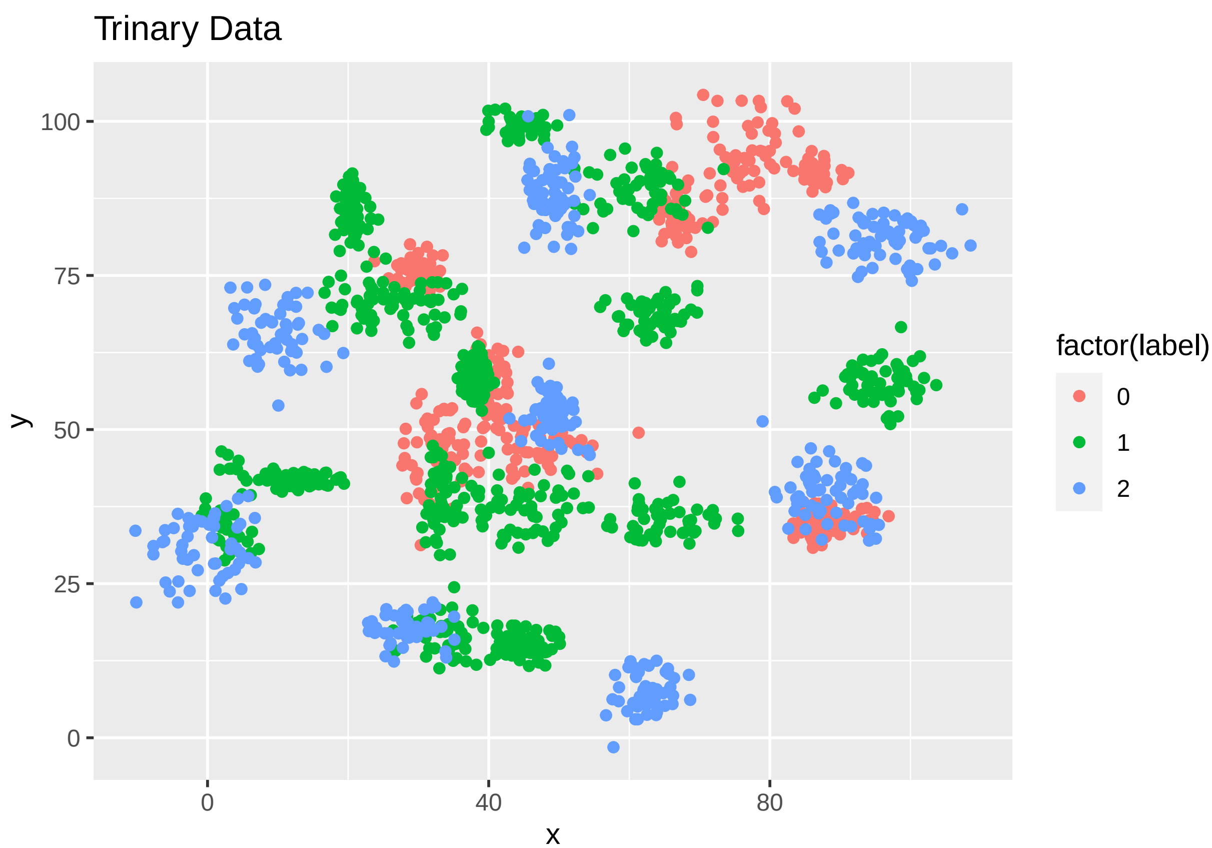
e. Note that in real-world datasets, your labels are usually not numbers, but text-based descriptions of the categories (e.g. spam or ham). In practice, you will encode categorical variables into numeric values.

i. Plot the data from each dataset using a scatter plot.

```r
ggplot(binary_df, aes(x = x, y = y)) +
  geom_point(aes(color = factor(label))) +
  ggtitle("Binary Data")
```




```r
ggplot(trinary_df, aes(x = x, y = y)) +
  geom_point(aes(color = factor(label))) +
  ggtitle("Trinary Data")
```



ii. The k nearest neighbors algorithm categorizes an input value by looking at the labels for the k nearest points and assigning a category based on the most common label. In this problem, you will determine which points are nearest by calculating the Euclidean distance between two points. As a refresher, the Euclidean distance between two points:

$p1 = (x1, y1)$ and $p2 = (x2, y2)$ is $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

```
binary_df$dist <- as.matrix(dist(binary_df))[nrow(binary_df), ]

head(binary_df)
```

```
##   label        x        y     dist
## 1     0 70.88469 83.17702 39.43945
## 2     0 74.97176 87.92922 42.21840
## 3     0 73.78333 92.20325 40.46914
## 4     0 66.40747 81.10617 36.00396
## 5     0 69.07399 84.53739 37.29978
## 6     0 72.23616 86.38403 39.86160
```

```
trinary_df$dist <- as.matrix(dist(trinary_df))[nrow(trinary_df), ]

head(trinary_df)
```

```
##   label        x        y     dist
## 1     0 30.08387 39.63094 79.20989
## 2     0 31.27613 51.77511 72.33329
```

```
## 3      0 34.12138 49.27575 70.88413
## 4      0 32.58222 41.23300 76.24301
## 5      0 34.65069 45.47956 72.26367
## 6      0 33.80513 44.24656 73.61988
```

iii. Fitting a model is when you use the input data to create a predictive model. There are various metrics you can use to determine how well your model fits the data. For this problem, you will focus on a single metric, accuracy. Accuracy is simply the percentage of how often the model predicts the correct result. If the model always predicts the correct result, it is 100% accurate. If the model always predicts the incorrect result, it is 0% accurate.

iv. Fit a k nearest neighbors' model for each dataset for k=3, k=5, k=10, k=15, k=20, and k=25. Compute the accuracy of the resulting models for each value of k. Plot the results in a graph where the x-axis is the different values of k and the y-axis is the accuracy of the model.

Binary Data

```
split_binary <- sample.split(binary_df, SplitRatio = 0.8)

train_cl <- subset(binary_df, split_binary == "TRUE")
test_cl <- subset(binary_df, split_binary == "FALSE")
```

```
# Feature Scaling
train_scale <- scale(train_cl[, 2:4])
test_scale <- scale(test_cl[, 2:4])
```

```
classifier_k01 <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$label,
                      k = 1)
classifier_k01
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [112] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [149] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [186] 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 0 1 1 0 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [297] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
## [334] 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [371] 1 1 1 1
## Levels: 0 1
```

```
# Confusion Matrix
confmatrix <- table(test_cl$label, classifier_k01)
confmatrix
```

```
##    classifier_k01
##       0   1
##   0 183   8
##   1  10 173
```

```r
# Calculate out of Sample error
misClassError_k01 <- mean(classifier_k01 != test_cl$label)

accuracy_k01 <- 1 - misClassError_k01

print(paste('Accuracy (k=1) =', accuracy_k01))
```

```
## [1] "Accuracy (k=1) = 0.951871657754011"
```

```r
# K = 3
classifier_k03 <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$label,
                      k = 3)
misClassError_k03 <- mean(classifier_k03 != test_cl$label)

accuracy_k03 <- 1 - misClassError_k01

print(paste('Accuracy (k=3) =', accuracy_k03))
```

```
## [1] "Accuracy (k=3) = 0.951871657754011"
```

```r
# K = 5
classifier_k05 <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$label,
                      k = 5)
misClassError_k05 <- mean(classifier_k05 != test_cl$label)

accuracy_k05 <- 1 - misClassError_k05

print(paste('Accuracy (k=5) =', accuracy_k05))
```

```
## [1] "Accuracy (k=5) = 0.967914438502674"
```

```r
# K = 10
classifier_k10 <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$label,
                      k = 10)
misClassError_k10 <- mean(classifier_k10 != test_cl$label)

accuracy_k10 <- 1 - misClassError_k10

print(paste('Accuracy (k=10) =', accuracy_k10))
```

```
## [1] "Accuracy (k=10) = 0.967914438502674"
```

```r
# K = 15
classifier_k15 <- knn(train = train_scale,
                      test = test_scale,
```

```r
                      cl = train_cl$label,
                      k = 15)
misClassError_k15 <- mean(classifier_k15 != test_cl$label)

accuracy_k15 <- 1 - misClassError_k15

print(paste('Accuracy (k=15) =', accuracy_k15))
```

```
## [1] "Accuracy (k=15) = 0.967914438502674"
```

```r
# K = 20
classifier_k20 <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$label,
                      k = 20)
misClassError_k20 <- mean(classifier_k20 != test_cl$label)

accuracy_k20 <- 1 - misClassError_k20

print(paste('Accuracy (k=20) =', accuracy_k20))
```

```
## [1] "Accuracy (k=20) = 0.967914438502674"
```

```r
# K = 20
classifier_k25 <- knn(train = train_scale,
                      test = test_scale,
                      cl = train_cl$label,
                      k = 25)
misClassError_k25 <- mean(classifier_k25 != test_cl$label)

accuracy_k25 <- 1 - misClassError_k25

print(paste('Accuracy (k=25) =', accuracy_k25))
```

```
## [1] "Accuracy (k=25) = 0.967914438502674"
```

```r
clusters = c(1, 3, 5, 10, 15, 20, 25)
accuracies = c(accuracy_k01, accuracy_k03, accuracy_k05, accuracy_k10,
  accuracy_k15, accuracy_k20, accuracy_k25)

binary_knn = data.frame(clusters, accuracies)
binary_knn
```

```
##   clusters accuracies
## 1        1  0.9518717
## 2        3  0.9518717
## 3        5  0.9679144
## 4       10  0.9679144
## 5       15  0.9679144
## 6       20  0.9679144
## 7       25  0.9679144
```

Trinary Data

```r
split_trinary <- sample.split(trinary_df, SplitRatio = 0.8)

train_cl_tri <- subset(trinary_df, split_trinary == "TRUE")
test_cl_tri <- subset(trinary_df, split_trinary == "FALSE")
```

```r
# Feature Scaling
train_scale_tri <- scale(train_cl_tri[, 2:4])
test_scale_tri <- scale(test_cl_tri[, 2:4])
```

```r
classifier_tri_k01 <- knn(train = train_scale_tri,
                       test = test_scale_tri,
                       cl = train_cl_tri$label,
                       k = 1)
classifier_tri_k01
```

```
##   [1] 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 2 0 2 0 2 2 0 2 2 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [75] 0 0 1 1 0 0 0 0 0 0 0 0 0 1 2 0 2 0 0 0 0 0 1 0 0 1 1 1 0 0 1 1 1 1 1 1 0
## [112] 0 1 1 2 2 1 1 1 2 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [186] 1 1 1 1 2 1 1 0 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0 1 1 1 1 1
## [223] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [260] 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 0 1 2 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 2
## [297] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 0 2 2 0 2 2 2 2 2
## [334] 2 1 1 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [371] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## Levels: 0 1 2
```

```r
# Confusion Matrix
confmatrix_tri <- table(test_cl_tri$label, classifier_tri_k01)
confmatrix_tri
```

```
##    classifier_tri_k01
##       0   1   2
##   0  80  11   8
##   1  17 156   7
##   2   3   8 102
```

```r
# Calculate out of Sample error
misClassError_tri_k01 <- mean(classifier_tri_k01 != test_cl_tri$label)

accuracy_tri_k01 <- 1 - misClassError_tri_k01

print(paste('Accuracy (k=1) =', accuracy_tri_k01))
```

```
## [1] "Accuracy (k=1) = 0.862244897959184"
```

```r
# K = 3
classifier_tri_k03 <- knn(train = train_scale_tri,
                          test = test_scale_tri,
                          cl = train_cl_tri$label,
                          k = 3)
misClassError_tri_k03 <- mean(classifier_tri_k03 != test_cl_tri$label)

accuracy_tri_k03 <- 1 - misClassError_tri_k01

print(paste('Accuracy (k=3) =', accuracy_tri_k03))
```

```
## [1] "Accuracy (k=3) = 0.862244897959184"
```

```r
# K = 5
classifier_tri_k05 <- knn(train = train_scale_tri,
                          test = test_scale_tri,
                          cl = train_cl_tri$label,
                          k = 5)
misClassError_tri_k05 <- mean(classifier_tri_k05 != test_cl_tri$label)

accuracy_tri_k05 <- 1 - misClassError_tri_k05

print(paste('Accuracy (k=5) =', accuracy_tri_k05))
```

```
## [1] "Accuracy (k=5) = 0.89030612244898"
```

```r
# K = 10
classifier_tri_k10 <- knn(train = train_scale_tri,
                          test = test_scale_tri,
                          cl = train_cl_tri$label,
                          k = 10)
misClassError_tri_k10 <- mean(classifier_tri_k10 != test_cl_tri$label)

accuracy_tri_k10 <- 1 - misClassError_tri_k10

print(paste('Accuracy (k=10) =', accuracy_tri_k10))
```

```
## [1] "Accuracy (k=10) = 0.885204081632653"
```

```r
# K = 15
classifier_tri_k15 <- knn(train = train_scale_tri,
                          test = test_scale_tri,
                          cl = train_cl_tri$label,
                          k = 15)
misClassError_tri_k15 <- mean(classifier_tri_k15 != test_cl_tri$label)

accuracy_tri_k15 <- 1 - misClassError_tri_k15

print(paste('Accuracy (k=15) =', accuracy_tri_k15))
```

```
## [1] "Accuracy (k=15) = 0.862244897959184"
```

```r
# K = 20
classifier_tri_k20 <- knn(train = train_scale_tri,
                          test = test_scale_tri,
                          cl = train_cl_tri$label,
                          k = 20)
misClassError_tri_k20 <- mean(classifier_tri_k20 != test_cl_tri$label)

accuracy_tri_k20 <- 1 - misClassError_tri_k20

print(paste('Accuracy (k=20) =', accuracy_tri_k20))
```

```
## [1] "Accuracy (k=20) = 0.869897959183674"
```

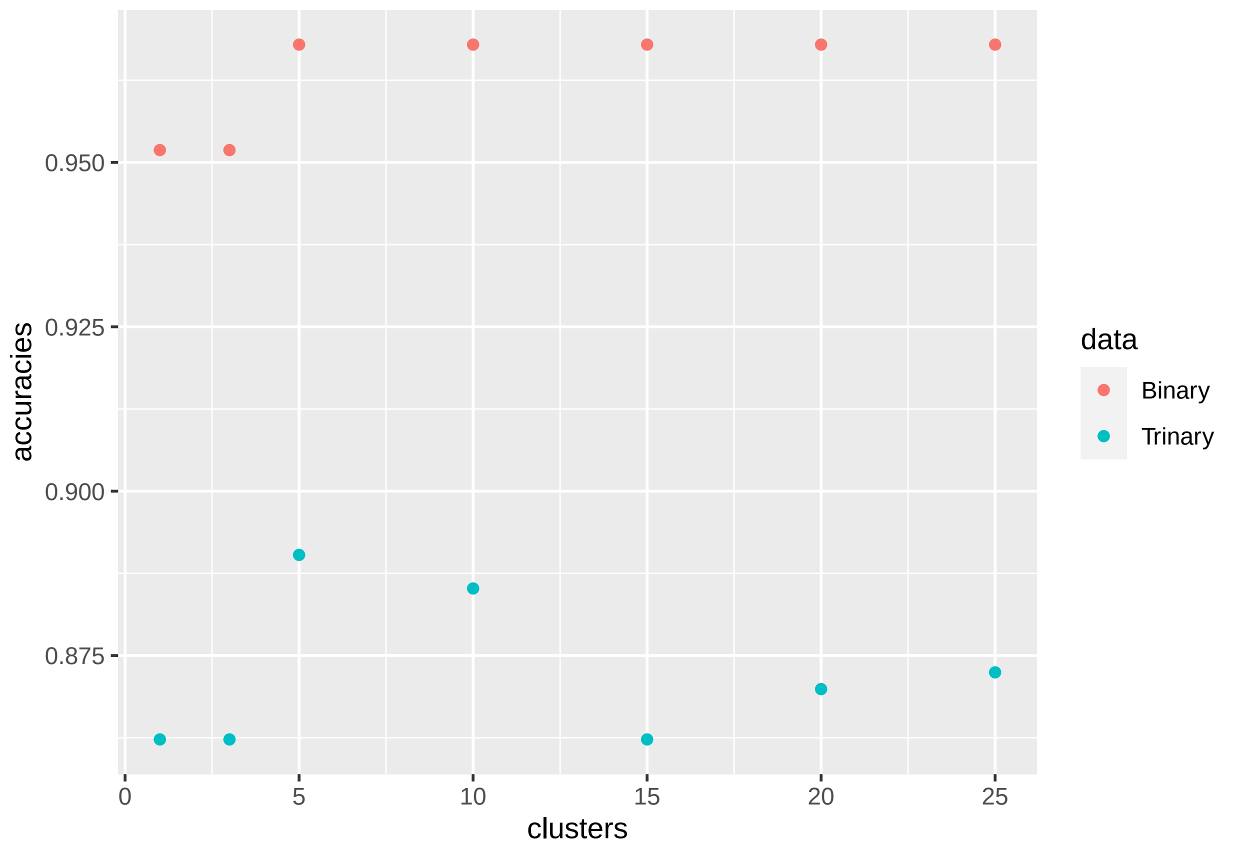
```r
# K = 25
classifier_tri_k25 <- knn(train = train_scale_tri,
                          test = test_scale_tri,
                          cl = train_cl_tri$label,
                          k = 25)
misClassError_tri_k25 <- mean(classifier_tri_k25 != test_cl_tri$label)

accuracy_tri_k25 <- 1 - misClassError_tri_k25

print(paste('Accuracy (k=25) =', accuracy_tri_k25))
```

```
## [1] "Accuracy (k=25) = 0.872448979591837"
```

```r
clusters = c(1, 3, 5, 10, 15, 20, 25)
data <- c('Binary', 'Binary', 'Binary', 'Binary', 'Binary', 'Binary', 'Binary',
        'Trinary',  'Trinary', 'Trinary', 'Trinary', 'Trinary', 'Trinary', 'Trinary')
accuracies = c(accuracy_k01, accuracy_k03, accuracy_k05, accuracy_k10,
  accuracy_k15, accuracy_k20, accuracy_k25, accuracy_tri_k01, accuracy_tri_k03,
  accuracy_tri_k05,   accuracy_tri_k10, accuracy_tri_k15, accuracy_tri_k20,
  accuracy_tri_k25)

binary_knn = data.frame(clusters, data, accuracies)
binary_knn
```

```
##    clusters    data accuracies
## 1         1  Binary  0.9518717
## 2         3  Binary  0.9518717
## 3         5  Binary  0.9679144
## 4        10  Binary  0.9679144
## 5        15  Binary  0.9679144
## 6        20  Binary  0.9679144
## 7        25  Binary  0.9679144
## 8         1 Trinary  0.8622449
## 9         3 Trinary  0.8622449
## 10        5 Trinary  0.8903061
## 11       10 Trinary  0.8852041
## 12       15 Trinary  0.8622449
## 13       20 Trinary  0.8698980
## 14       25 Trinary  0.8724490
```

```
ggplot(data = binary_knn, aes(x = clusters, y = accuracies, color = data)) + geom_point()
```



v. Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?

No, the data visually looks to be in clusters not a linear path so the linear classification would not predict the label very accurately

vi. How does the accuracy of your logistic regression classifier from last week compare? Why is the accuracy different between these two methods?

With the linear regression the accuracy was only 58.4% whereas even with only 3 clusters the accuracy went up to 95% and with 25 clusters 97%.

2. Clustering

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

a. These assignments are here to provide you with an introduction to the "Data Science" use for these tools. This is your future. It may seem confusing and weird right now but it hopefully seems far less so than earlier in the semester. Attempt these homework assignments. You will not be graded on your answer but on your approach. This should be a, "Where am I on learning this stuff" check. If you can't get it done, please explain why.

b. Remember to submit this assignment in an R Markdown report.

c. Labeled data is not always available. For these types of datasets, you can use unsupervised algorithms to extract structure. The k-means clustering algorithm and the k nearest neighbor algorithm both use the Euclidean distance between points to group data points. The difference is the k-means clustering algorithm does not use labeled data.

d. In this problem, you will use the k-means clustering algorithm to look for patterns in an unlabeled dataset. The dataset for this problem is found at data/clustering-data.csv.

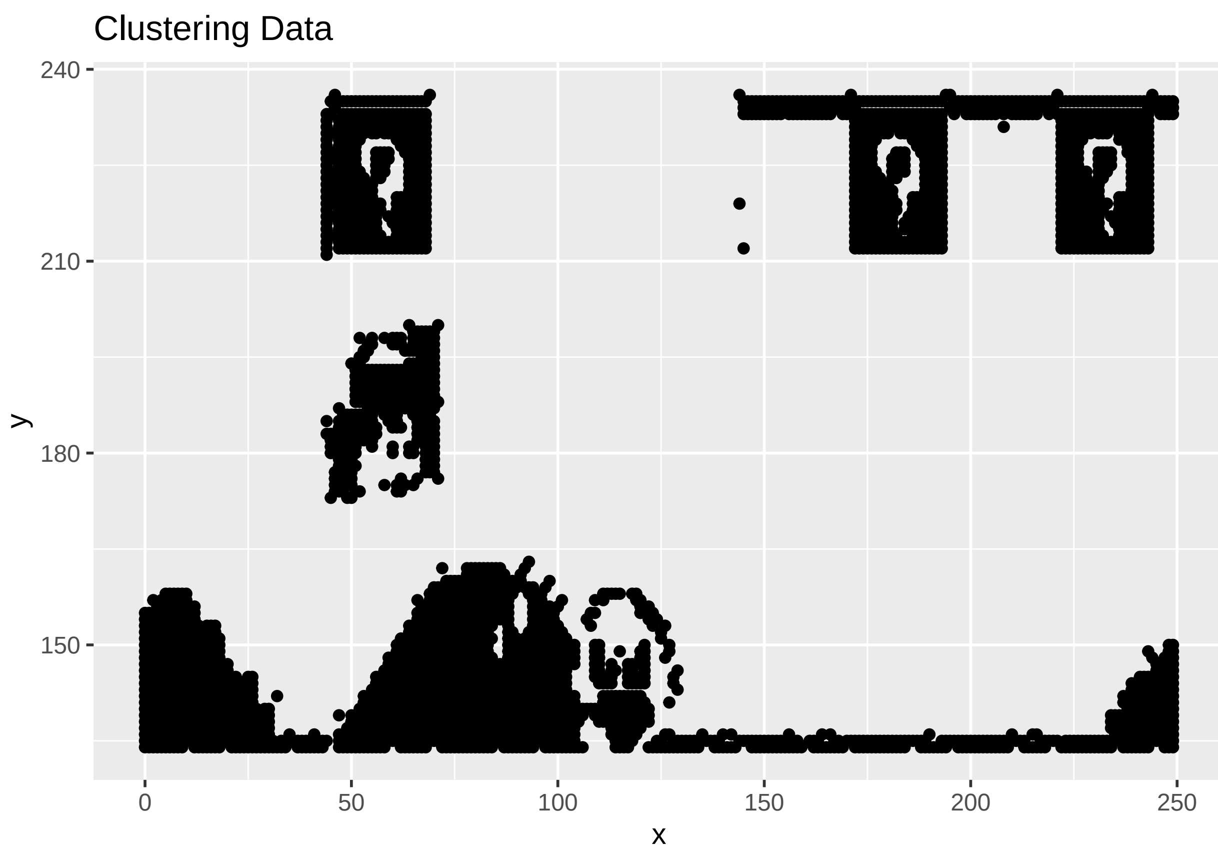
```r
## Load the binary classifier data
clustering_df <- read.csv("data/clustering-data.csv",
    header = TRUE,
    stringsAsFactors = FALSE)

head(clustering_df)
```

```
##     x   y
## 1  46 236
## 2  69 236
## 3 144 236
## 4 171 236
## 5 194 236
## 6 195 236
```

```r
set.seed(10)
```

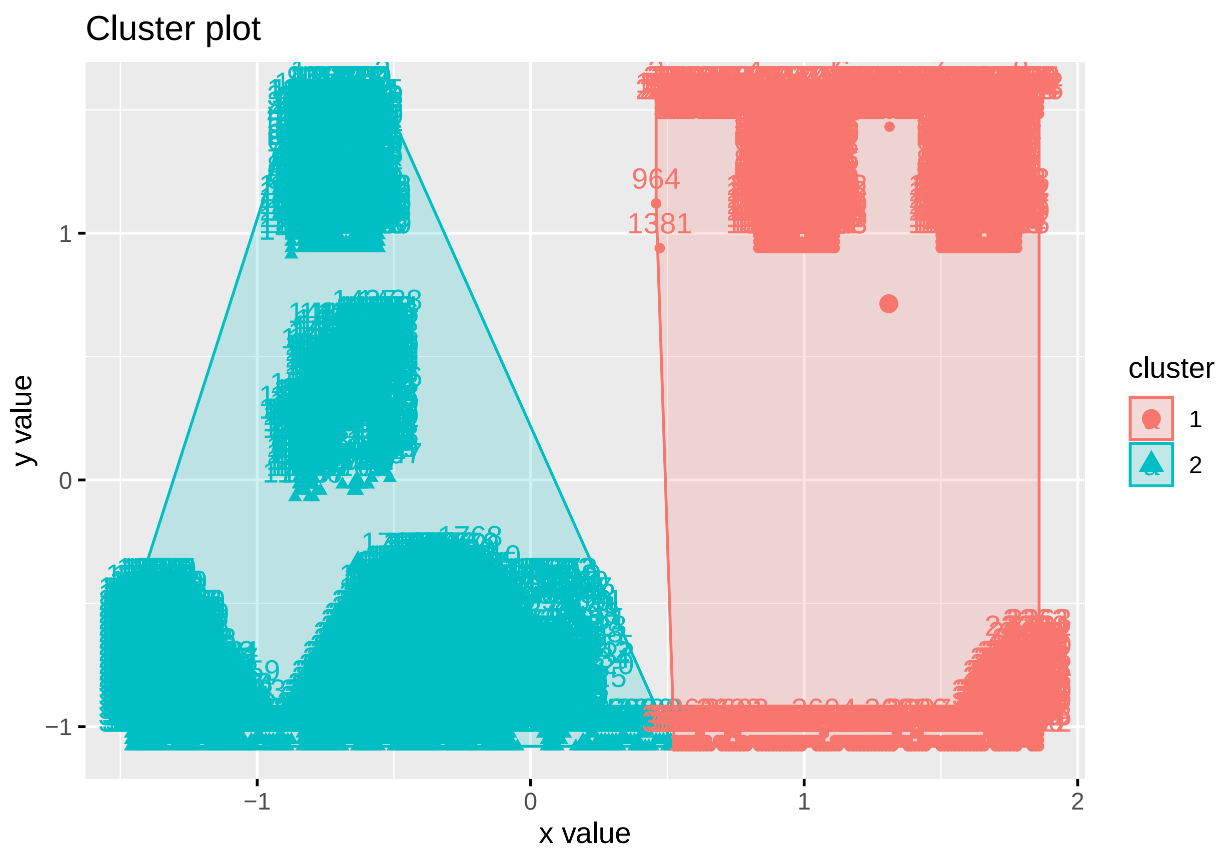
```r
ggplot(clustering_df, aes(x = x, y = y)) +
  geom_point() +
  ggtitle("Clustering Data")
```

i. Plot the dataset using a scatter plot.

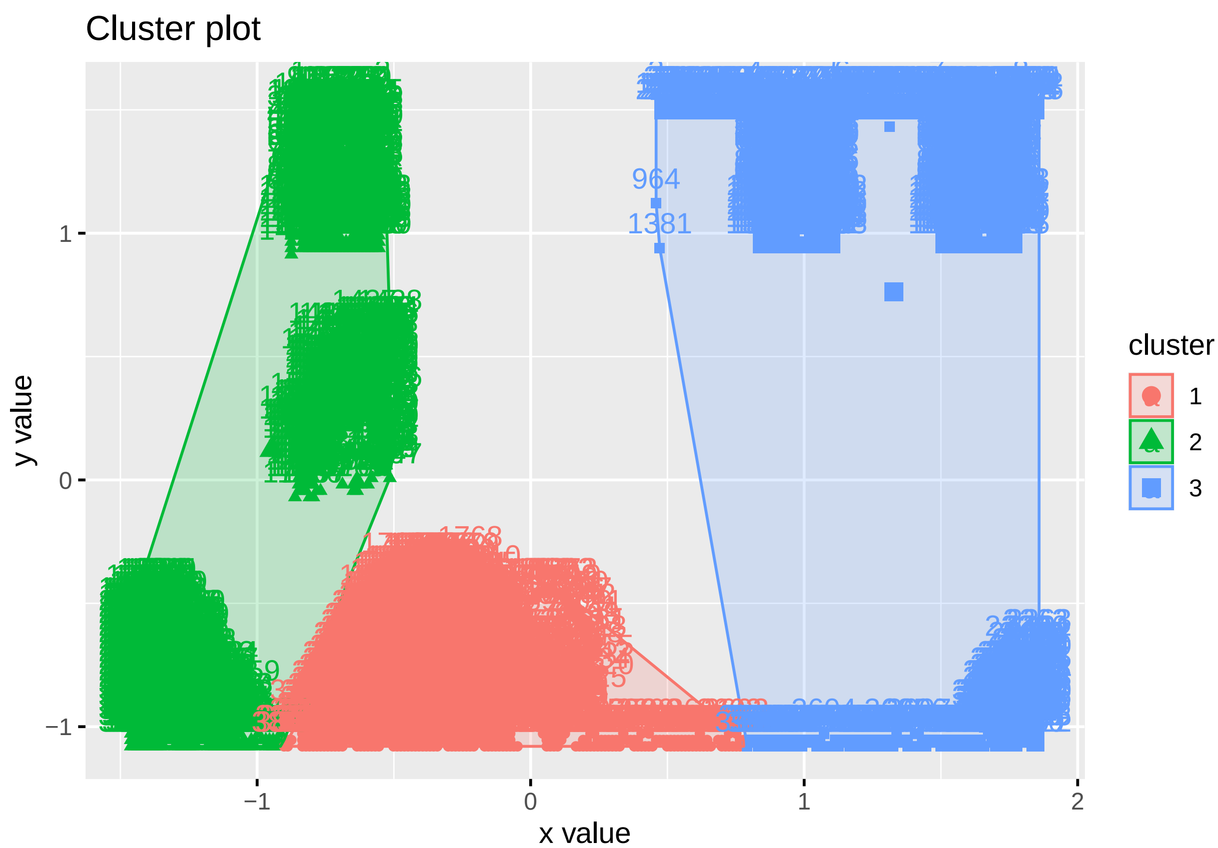



```r
# k = 2
clustering.k2 <- kmeans(clustering_df, centers = 2, nstart = 20)

fviz_cluster(clustering.k2, data = clustering_df)
```

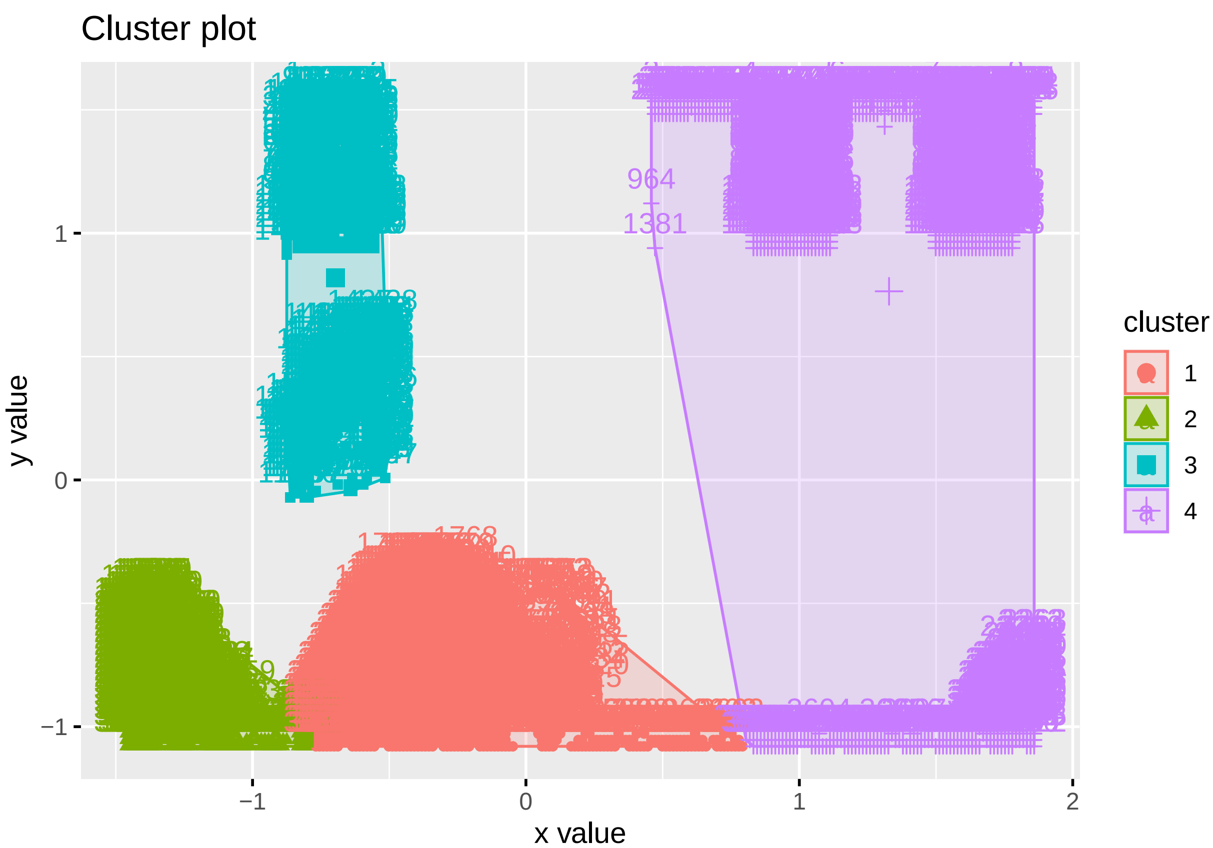
ii. Fit the dataset using the k-means algorithm from k=2 to k=12. Create a scatter plot of the resultant clusters for each value of k.



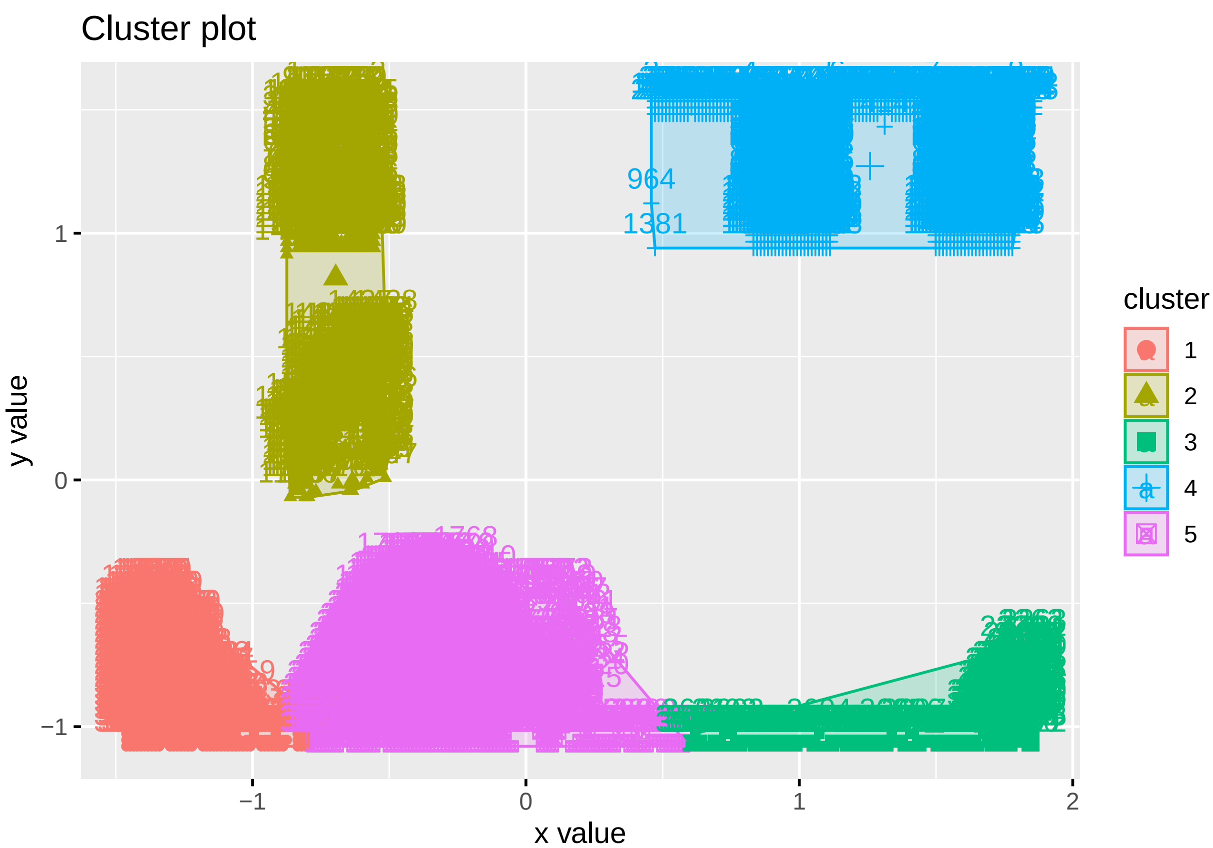

```r
# k = 3
clustering.k3 <- kmeans(clustering_df, centers = 3, nstart = 20)

fviz_cluster(clustering.k3, data = clustering_df)
```



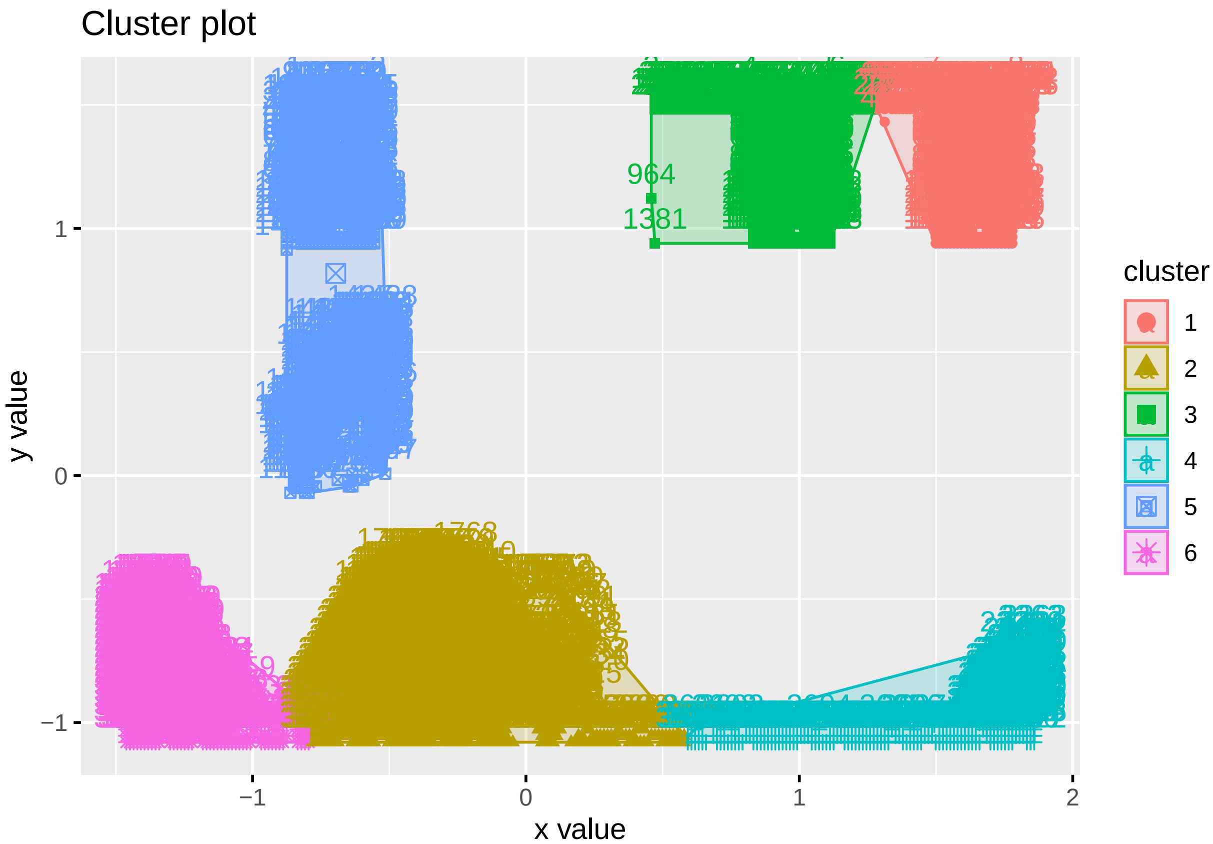

```r
# k = 4
clustering.k4 <- kmeans(clustering_df, centers = 4, nstart = 20)

fviz_cluster(clustering.k4, data = clustering_df)
```



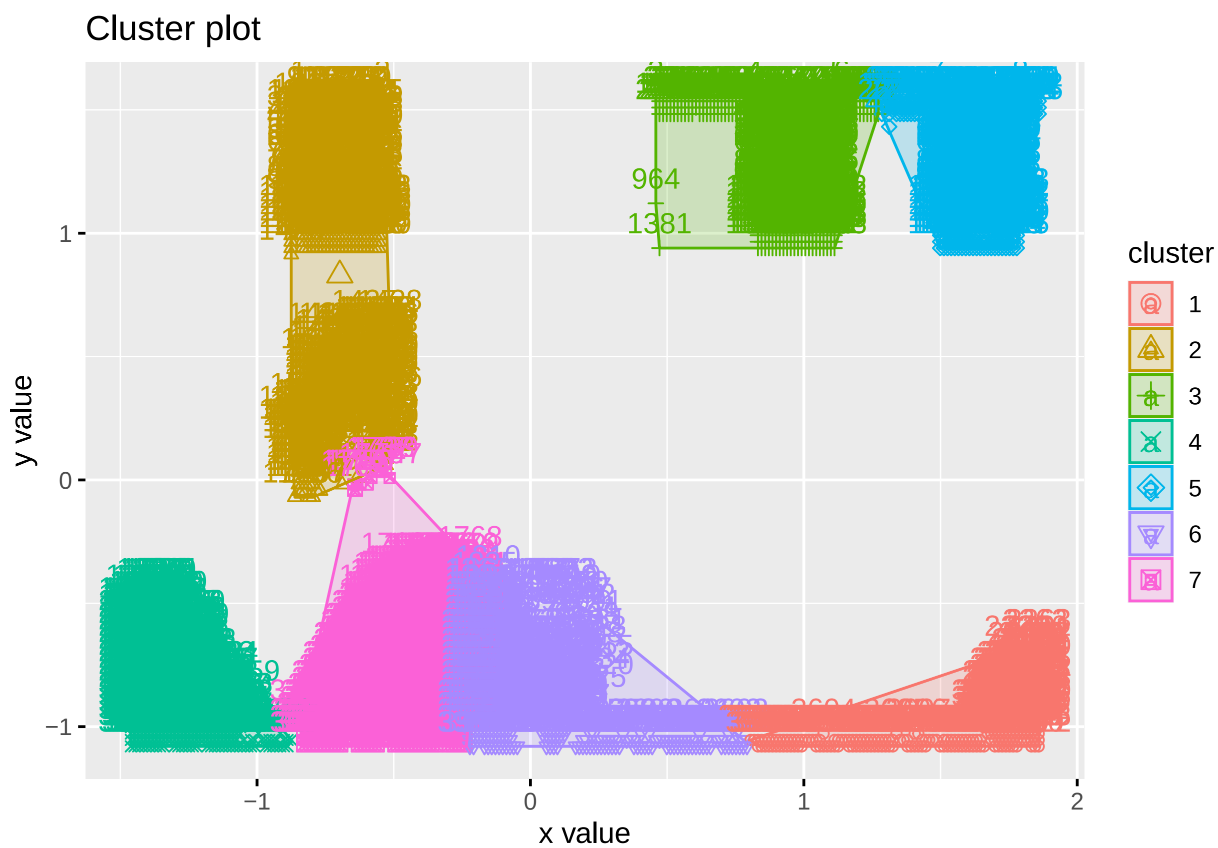

```r
# k = 5
clustering.k5 <- kmeans(clustering_df, centers = 5, nstart = 20)

fviz_cluster(clustering.k5, data = clustering_df)
```



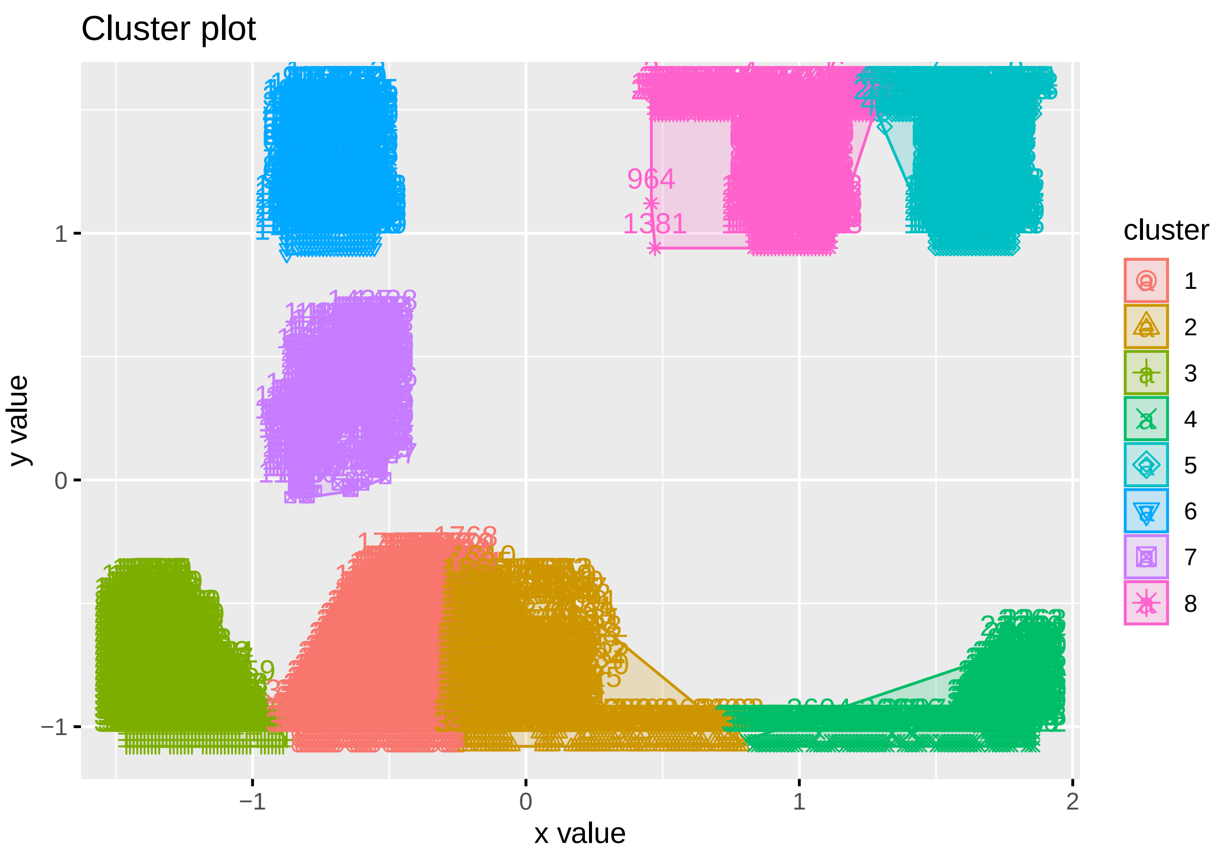

```
# k = 6
clustering.k6 <- kmeans(clustering_df, centers = 6, nstart = 20)

fviz_cluster(clustering.k6, data = clustering_df)
```



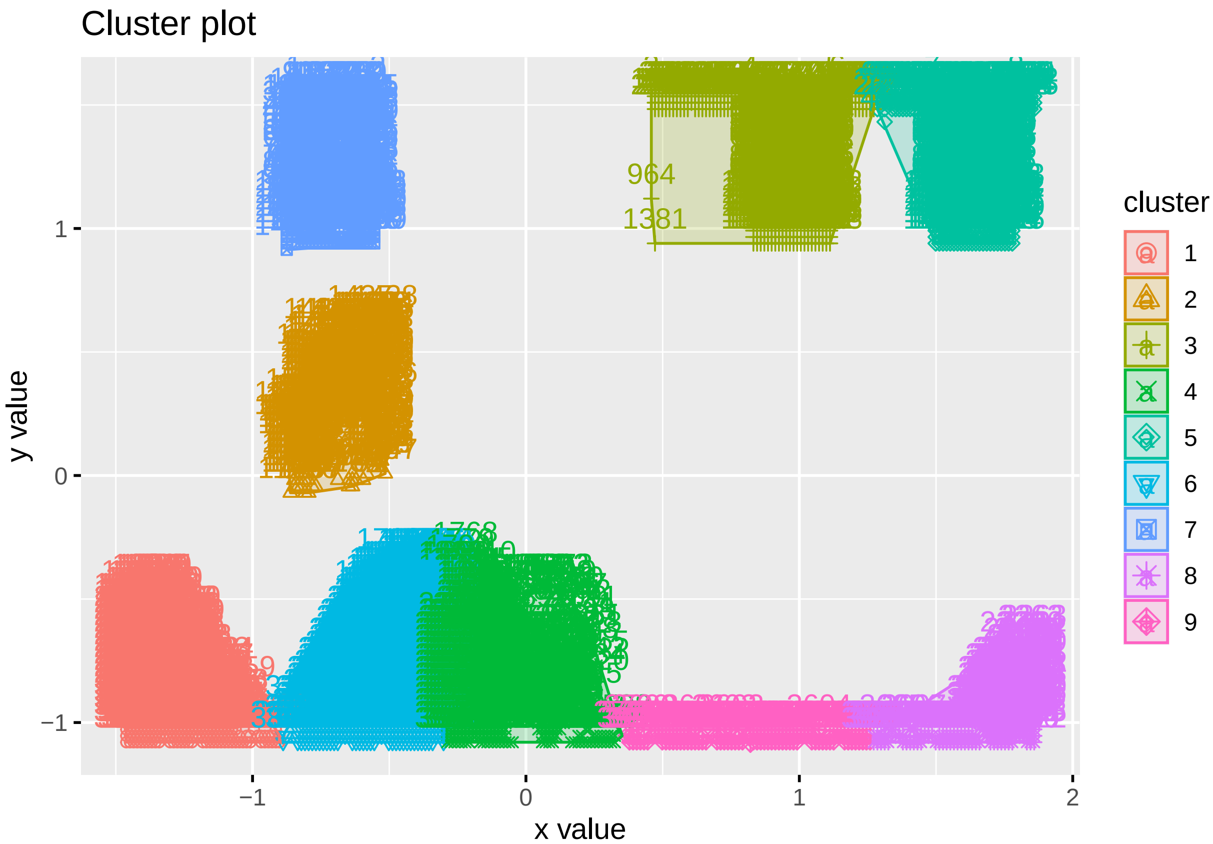

```r
# k = 7
clustering.k7 <- kmeans(clustering_df, centers = 7, nstart = 20)

fviz_cluster(clustering.k7, data = clustering_df)
```



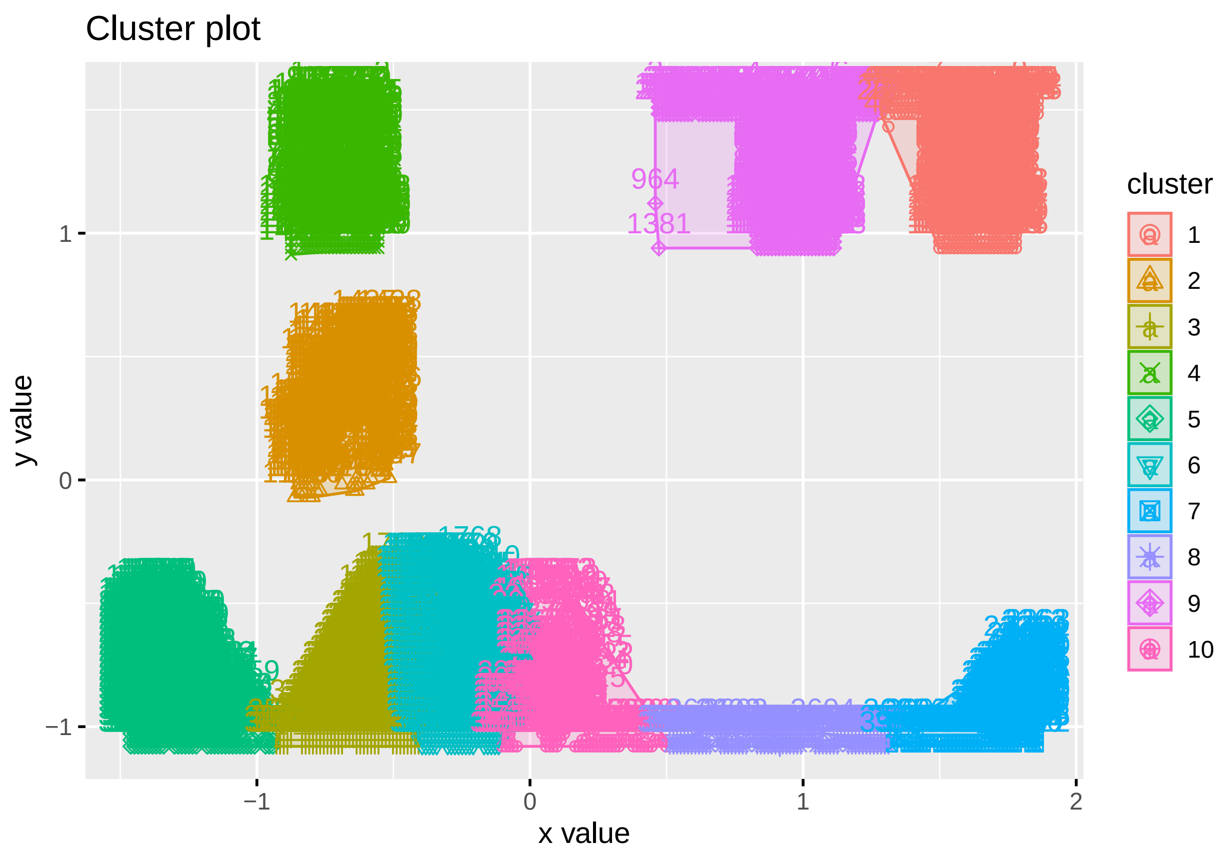

```r
# k = 8
clustering.k8 <- kmeans(clustering_df, centers = 8, nstart = 20)

fviz_cluster(clustering.k8, data = clustering_df)
```



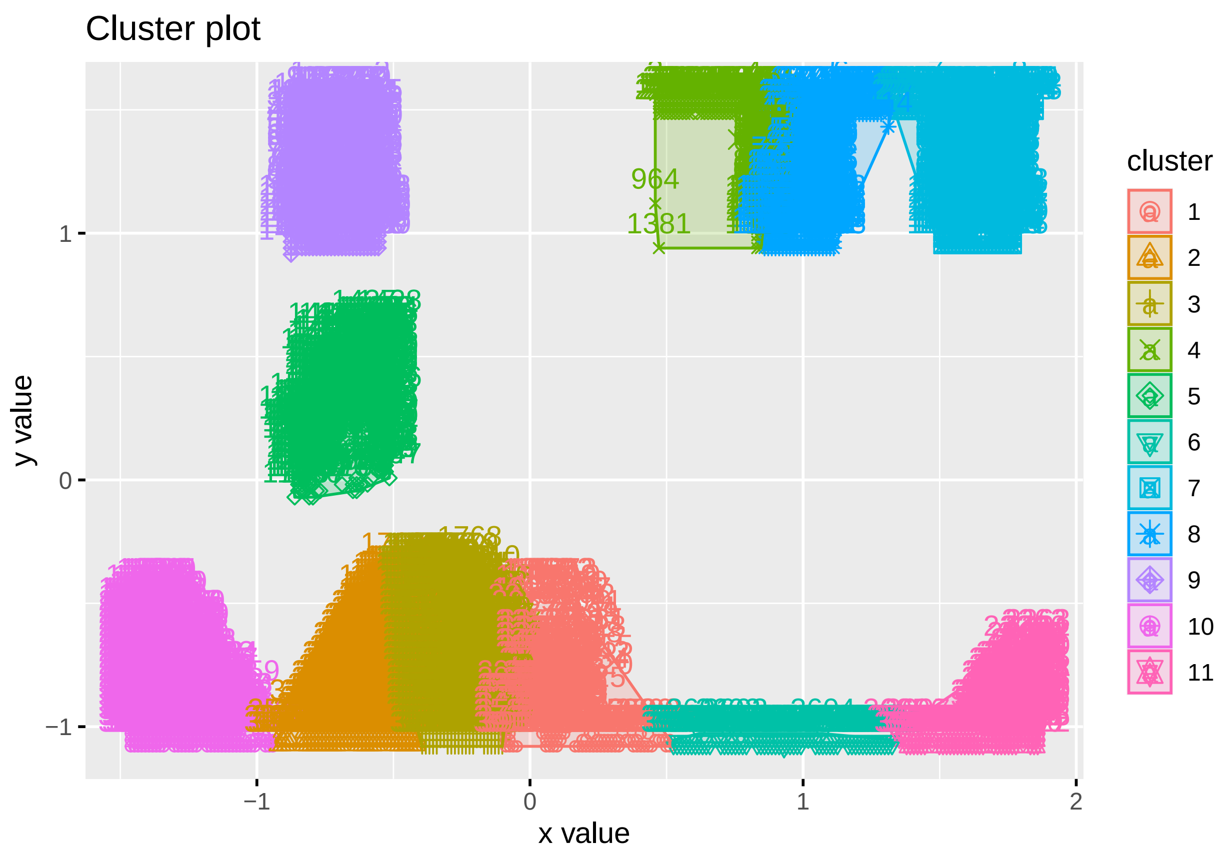

```r
# k = 9
clustering.k9 <- kmeans(clustering_df, centers = 9, nstart = 20)

fviz_cluster(clustering.k9, data = clustering_df)
```



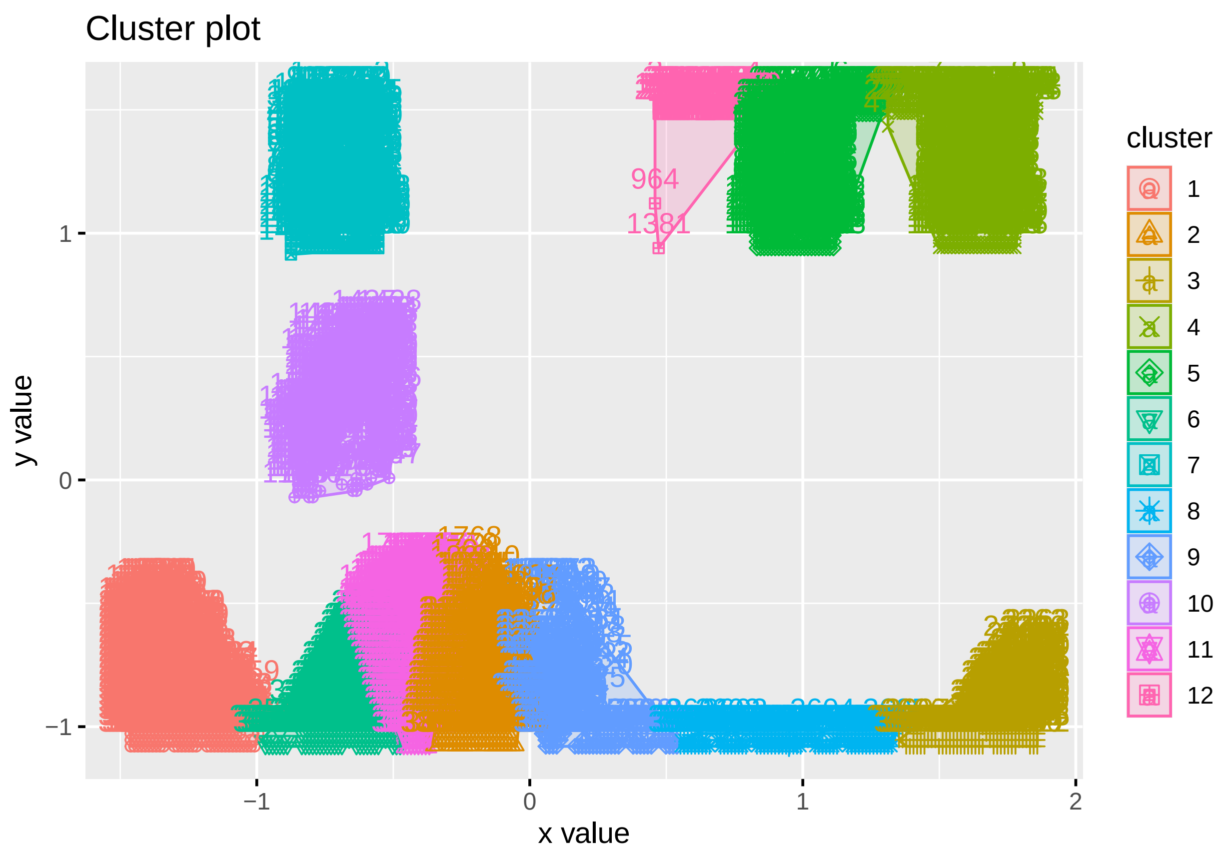

```
# k = 10
clustering.k10 <- kmeans(clustering_df, centers = 10, nstart = 20)

fviz_cluster(clustering.k10, data = clustering_df)
```




```
# k = 11
clustering.k11 <- kmeans(clustering_df, centers = 11, nstart = 20)

fviz_cluster(clustering.k11, data = clustering_df)
```




```r
# k = 12
clustering.k12 <- kmeans(clustering_df, centers = 12, nstart = 20)

fviz_cluster(clustering.k12, data = clustering_df)
```

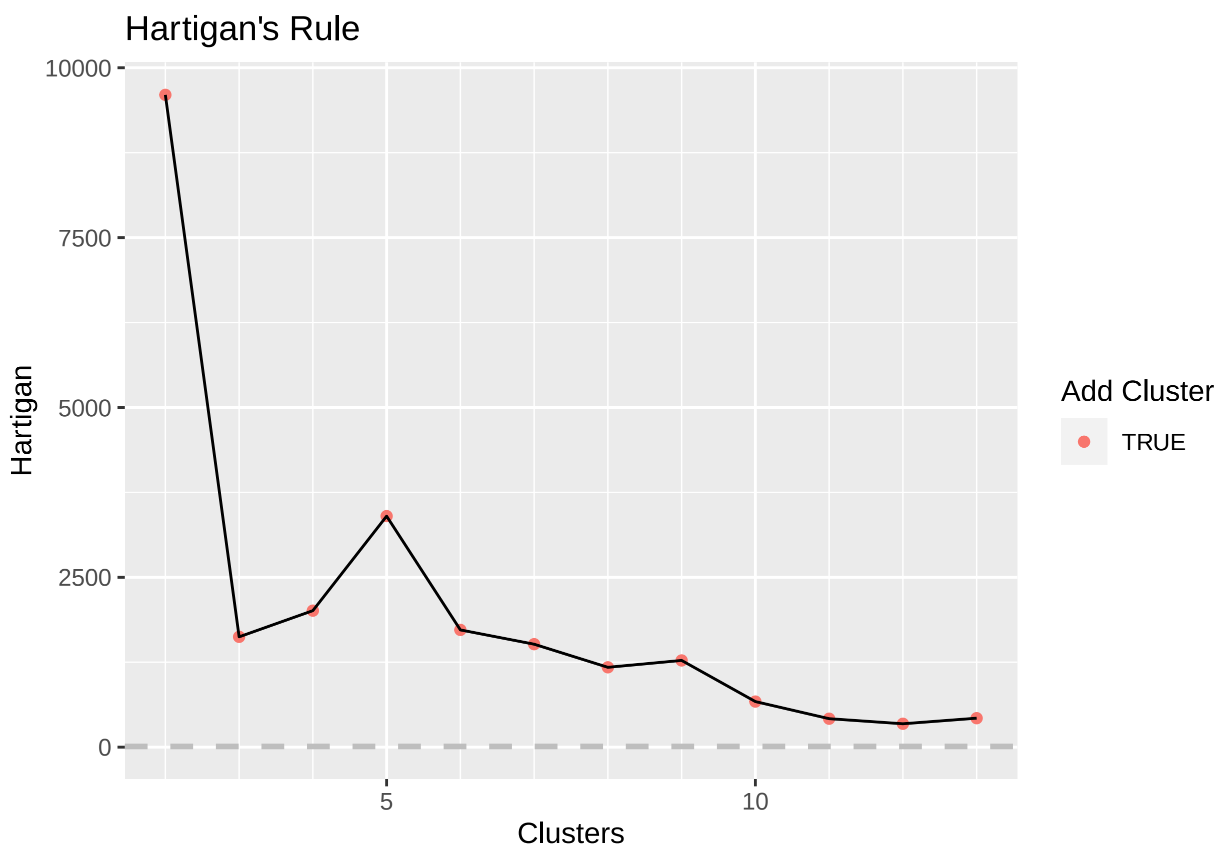



```
clustering_best <- FitKMeans(clustering_df, max.clusters = 13, nstart = 20)

clustering_best
```

iii. As k-means is an unsupervised algorithm, you cannot compute the accuracy as there are no correct values to compare the output to. Instead, you will use the average distance from the center of each cluster as a measure of how well the model fits the data. To calculate this metric, simply compute the distance of each data point to the center of the cluster it is assigned to and take the average value of all of those distances.

```
##    Clusters  Hartigan AddCluster
## 1         2 9600.6138       TRUE
## 2         3 1623.3382       TRUE
## 3         4 2008.8600       TRUE
## 4         5 3400.0118       TRUE
## 5         6 1725.2430       TRUE
## 6         7 1515.0802       TRUE
## 7         8 1174.8501       TRUE
## 8         9 1275.9895       TRUE
## 9        10  670.5540       TRUE
## 10       11  418.3654       TRUE
## 11       12  343.7137       TRUE
## 12       13  425.9384       TRUE
```

```
PlotHartigan(clustering_best)
```



e. Calculate this average distance from the center of each cluster for each value of k and plot it as a line chart where k is the x-axis and the average distance is the y-axis.

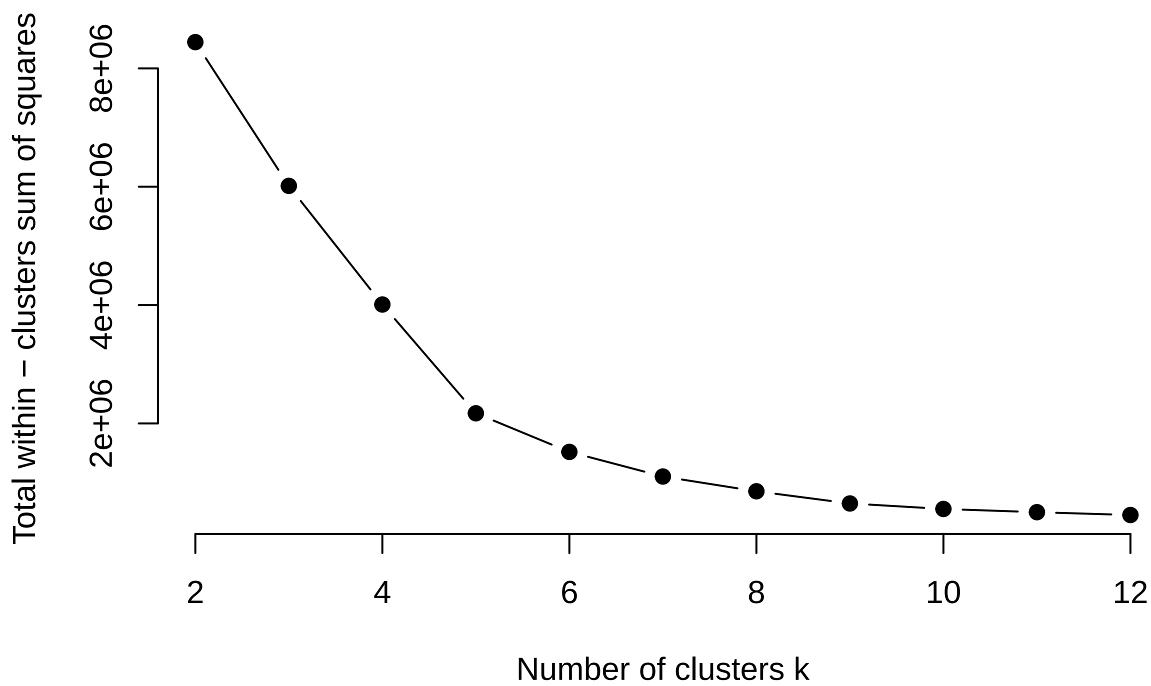
```
k.max <- 12

clustering_best
```

```
##    Clusters  Hartigan AddCluster
## 1         2 9600.6138       TRUE
## 2         3 1623.3382       TRUE
## 3         4 2008.8600       TRUE
## 4         5 3400.0118       TRUE
## 5         6 1725.2430       TRUE
## 6         7 1515.0802       TRUE
## 7         8 1174.8501       TRUE
## 8         9 1275.9895       TRUE
## 9        10  670.5540       TRUE
## 10       11  418.3654       TRUE
## 11       12  343.7137       TRUE
## 12       13  425.9384       TRUE
```

```r
wss <- sapply(2:k.max, function(k){kmeans(clustering_df, k, nstart = 50, iter.max = 12)$tot.withinss})
wss
```

```
##  [1] 8443681.1 6014377.9 4009678.4 2171612.8 1519043.4 1102869.9  853160.1
##  [8]  647331.9  554632.2  500038.6  452352.3
```

```r
plot(2:k.max, wss,
     type = 'b', pch = 19, frame = FALSE,
     xlab = 'Number of clusters k',
     ylab = 'Total within - clusters sum of squares')
```



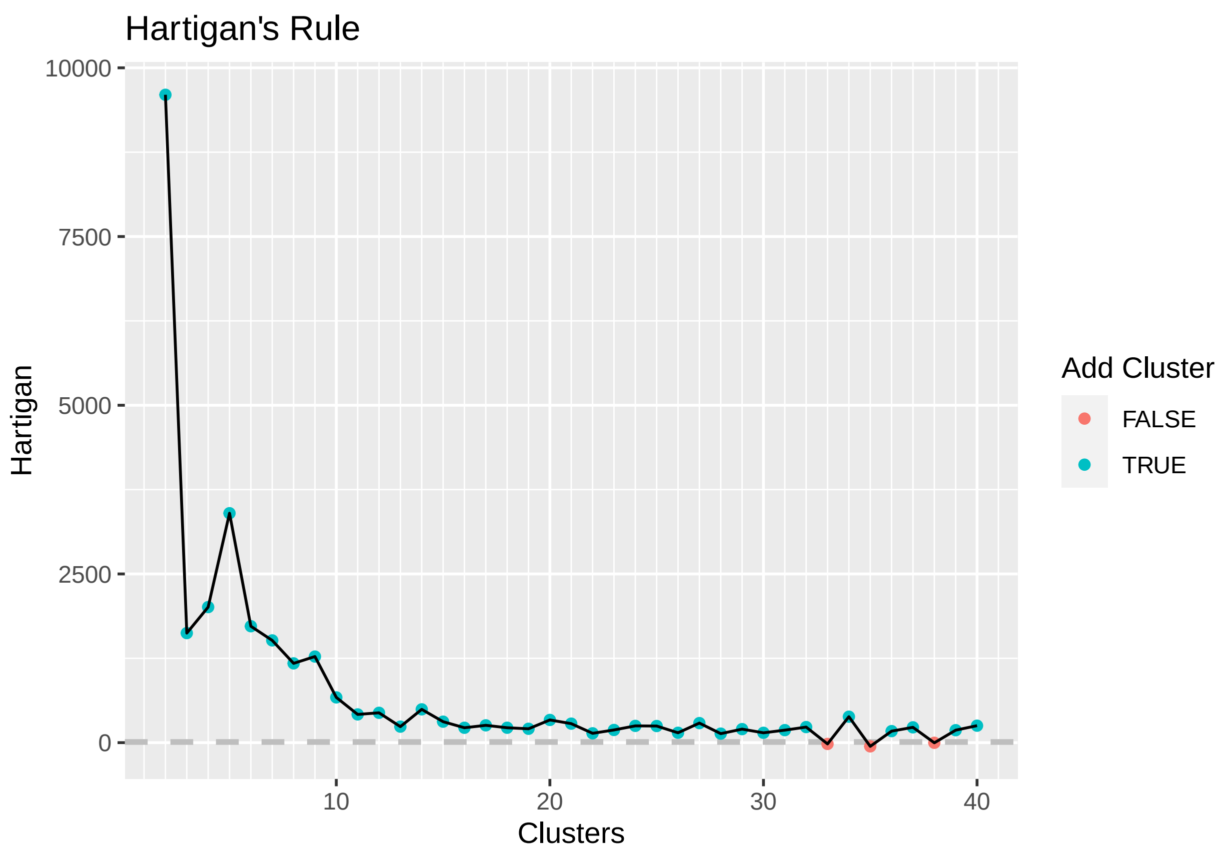

```r
clustersBest <- FitKMeans(clustering_df, max.clusters = 40, nstart = 20, seed = 10)
```

```
## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations

## Warning: did not converge in 10 iterations
```

```r
clustersBest
```

```
##    Clusters   Hartigan AddCluster
## 1         2 9600.61378       TRUE
## 2         3 1623.33823       TRUE
## 3         4 2008.86000       TRUE
## 4         5 3400.01180       TRUE
## 5         6 1725.24304       TRUE
## 6         7 1515.08024       TRUE
## 7         8 1174.85010       TRUE
## 8         9 1276.01546       TRUE
## 9        10  670.53106       TRUE
## 10       11  418.36544       TRUE
## 11       12  442.29321       TRUE
## 12       13  237.15764       TRUE
## 13       14  493.40182       TRUE
## 14       15  310.92926       TRUE
## 15       16  220.73979       TRUE
## 16       17  256.73282       TRUE
## 17       18  220.82498       TRUE
## 18       19  205.44392       TRUE
## 19       20  336.40611       TRUE
## 20       21  281.64398       TRUE
## 21       22  138.30725       TRUE
## 22       23  188.00035       TRUE
## 23       24  248.44427       TRUE
## 24       25  246.31578       TRUE
## 25       26  146.28557       TRUE
## 26       27  289.74534       TRUE
## 27       28  133.55867       TRUE
## 28       29  200.35379       TRUE
## 29       30  145.77929       TRUE
## 30       31  185.29727       TRUE
## 31       32  230.87478       TRUE
## 32       33  -19.03329      FALSE
## 33       34  384.67158       TRUE
## 34       35  -55.05502      FALSE
## 35       36  170.82466       TRUE
## 36       37  226.92808       TRUE
## 37       38   -2.13646      FALSE
## 38       39  184.74102       TRUE
## 39       40  251.45425       TRUE
```

```
PlotHartigan(clustersBest)
```



f. One way of determining the "right" number of clusters is to look at the graph of k versus average distance and finding the "elbow point". Looking at the graph you generated in the previous example, what is the elbow point for this dataset?

Looking at 2 through 12 clusters, 5 clusters is the best. Looking at clusters up to 40, 33 clusters is the best.