Week 12

Allison Forte

2022-06-04

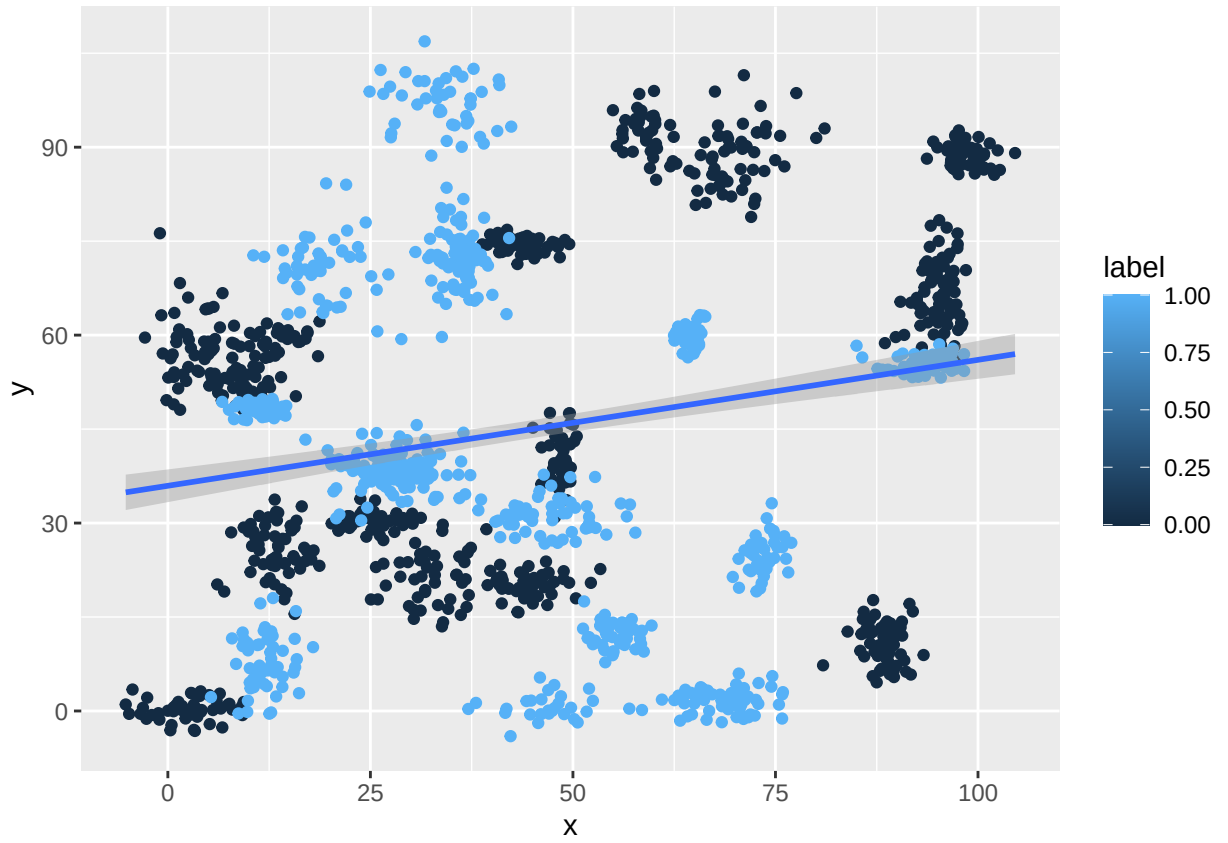
Introduction to Machine Learning

Regression algorithms: used to predict numeric quantity while classification algorithms predict categorical outcomes.

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

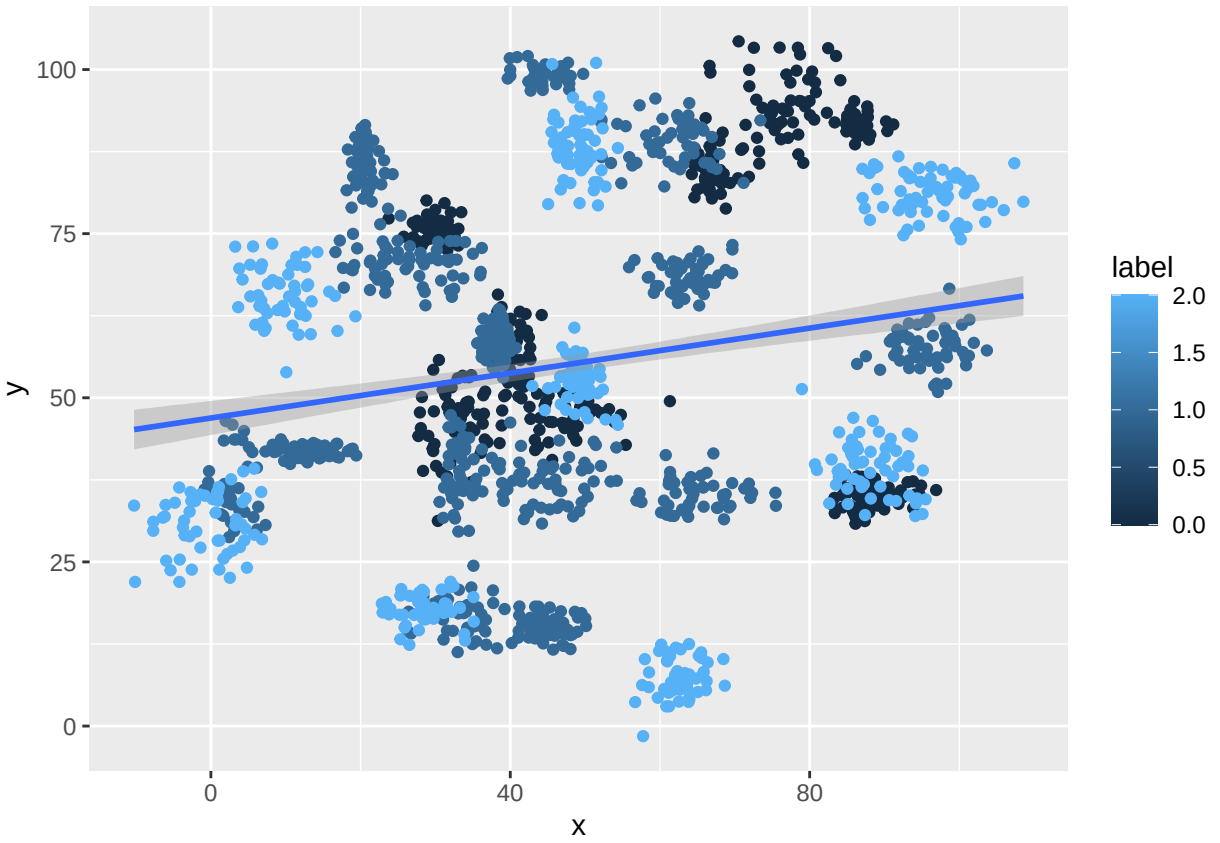
```
library(ggplot2)
ggplot(binary_df, aes(x=x, y=y, color=label)) + geom_point() + geom_smooth(method=lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```




```
library(ggplot2)
ggplot(tertiary_df, aes(x=x, y=y, color = label)) + geom_point() + geom_smooth(method=lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



- 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.
- 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.
- 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 knn calculations

```
library(caTools)
library(class)

# split data: binary
split <- sample.split(binary_df, SplitRatio = 0.8)

XB_train <- subset(binary_df, split == "TRUE")
XB_validate <- subset(binary_df, split == "FALSE")
```

```
YB_target_category <- XB_train[,'label']
YB_test_category <- XB_validate[,'label']

# run knn function: binary
knnB3 <- knn(XB_train,XB_validate,cl=YB_target_category,k=3)

# create confusion matrix: binary
matrixB3 <- table(knnB3,YB_test_category)

# calculate accuracy
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
accuracyB3 <- accuracy(matrixB3)

# adjust for different k values
knnB5 <- knn(XB_train,XB_validate,cl=YB_target_category,k=5)
matrixB5 <- table(knnB5,YB_test_category)
accuracyB5 <- accuracy(matrixB5)

knnB10 <- knn(XB_train,XB_validate,cl=YB_target_category,k=10)
matrixB10 <- table(knnB10,YB_test_category)
accuracyB10 <- accuracy(matrixB10)

knnB15 <- knn(XB_train,XB_validate,cl=YB_target_category,k=15)
matrixB15 <- table(knnB15,YB_test_category)
accuracyB15 <- accuracy(matrixB15)

knnB20 <- knn(XB_train,XB_validate,cl=YB_target_category,k=20)
matrixB20 <- table(knnB20,YB_test_category)
accuracyB20 <- accuracy(matrixB20)

knnB25 <- knn(XB_train,XB_validate,cl=YB_target_category,k=25)
matrixB25 <- table(knnB25,YB_test_category)
accuracyB25 <- accuracy(matrixB25)
```

Tertiary knn calculations

```
# Tertiary

# split data: tertiary
split <- sample.split(tertiary_df, SplitRatio = 0.8)

XT_train <- subset(tertiary_df, split == "TRUE")
XT_validate <- subset(tertiary_df, split == "FALSE")

YT_target_category <- XT_train[,'label']
YT_test_category <- XT_validate[,'label']

# run knn function: tertiary
knnT3 <- knn(XT_train,XT_validate,cl=YT_target_category,k=3)

# create confusion matrix: tertiary
matrixT3 <- table(knnT3,YT_test_category)
```

```
# calculate accuracy: tertiary
accuracy <- function(x){sum(diag(x)/(sum(rowSums(x)))) * 100}
accuracyT3 <- accuracy(matrixT3)

# adjust for different k values
knnT5 <- knn(XT_train,XT_validate,cl=YT_target_category,k=5)
matrixT5 <- table(knnT5,YT_test_category)
accuracyT5 <- accuracy(matrixT5)

knnT10 <- knn(XT_train,XT_validate,cl=YT_target_category,k=10)
matrixT10 <- table(knnT10,YT_test_category)
accuracyT10 <- accuracy(matrixT10)

knnT15 <- knn(XT_train,XT_validate,cl=YT_target_category,k=15)
matrixT15 <- table(knnT15,YT_test_category)
accuracyT15 <- accuracy(matrixT15)

knnT20 <- knn(XT_train,XT_validate,cl=YT_target_category,k=20)
matrixT20 <- table(knnT20,YT_test_category)
accuracyT20 <- accuracy(matrixT20)

knnT25 <- knn(XT_train,XT_validate,cl=YT_target_category,k=25)
matrixT25 <- table(knnT25,YT_test_category)
accuracyT25 <- accuracy(matrixT25)
```

- 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.

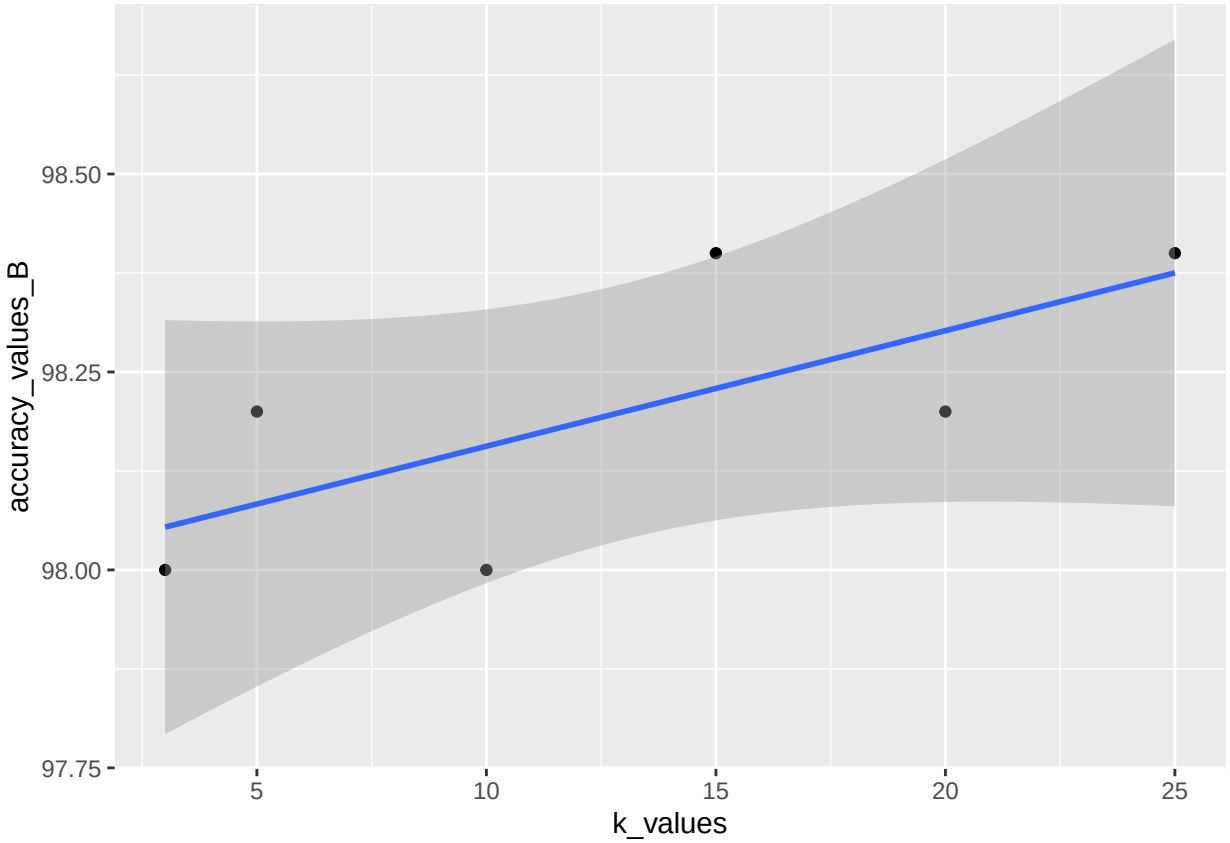
```
# values of k
k_values = c(3, 5, 10, 15, 20, 25)

# accuracy for binary data
accuracy_values_B = c(accuracyB3, accuracyB5, accuracyB10, accuracyB15, accuracyB20, accuracyB25)

# dataframe for k values and accuracy for binary
dfB <- data.frame(k_values, accuracy_values_B)

# plot for binary data
ggplot(dfB, aes(k_values, accuracy_values_B)) + geom_point() + geom_smooth(method=lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

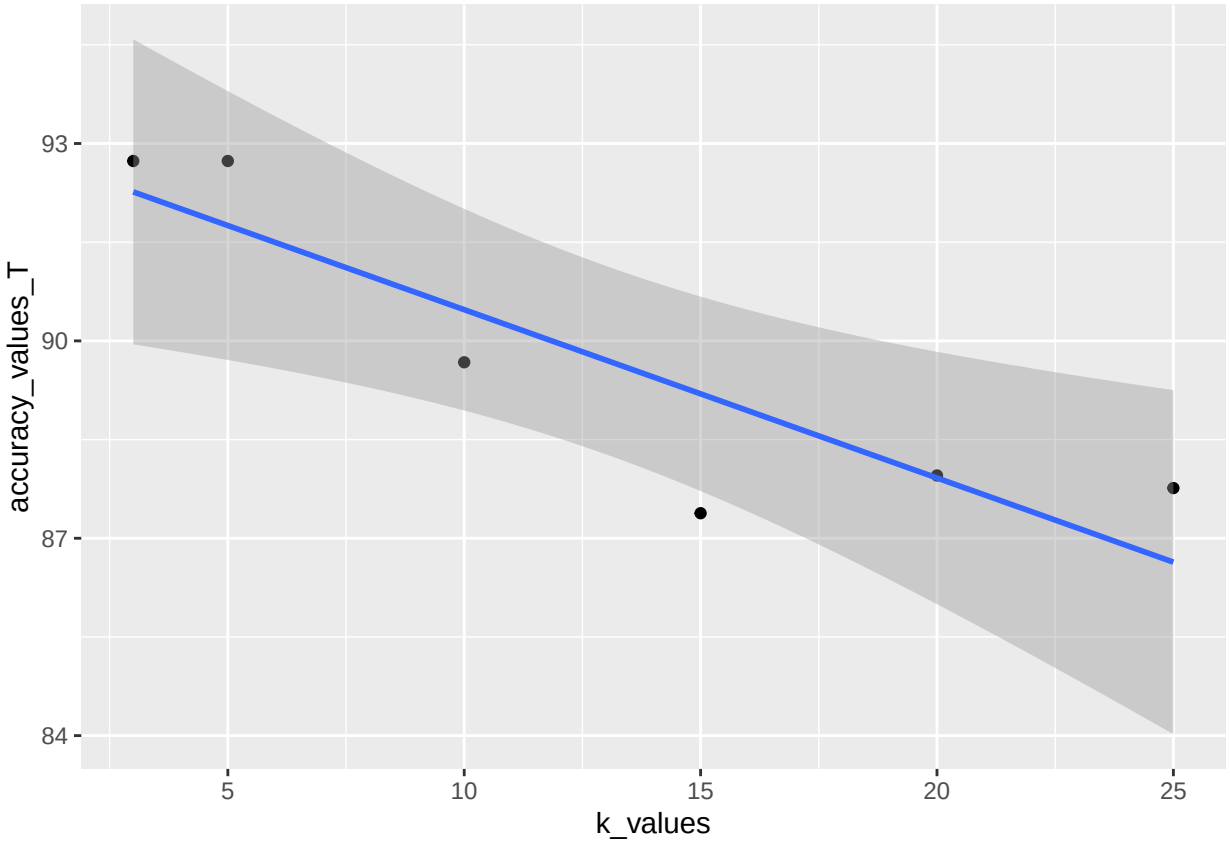



```
# accuracy for tertiary data
accuracy_values_T = c(accuracyT3, accuracyT5, accuracyT10, accuracyT15, accuracyT20, accuracyT25)

# dataframe for k values and accuracy for binary
dfT <- data.frame(k_values, accuracy_values_T)

# plot for binary data
ggplot(dfB, aes(k_values, accuracy_values_T)) + geom_point() + geom_smooth(method=lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



- Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?
  - Based on the plots it appears that a linear classifier may not be the best model for these data sets. From the plots, it looks like a linear classifier would work better on the binary data set than on the tertiary data set given the data points are closer to the line for the binary set. The points for the tertiary data set seem to have a curve to them which leads me to believe a linear model is not the best choice.
- How does the accuracy of your logistic regression classifier from last week compare? Why is the accuracy different between these two methods?
  - The accuracy of the model from last week was 86%. The accuracy of the models from this week ranged from 97.2 to 98.4% for the binary dataset and 87.8% to 94.4% for the tertiary dataset
  - While I do not fully understand why the accuracy is diferent it seems that by using machine learning and asking the program to analyze each data point individually to see what it is near, we can have a more accurate model. The linear regression model is simply looking at the relationship between variables overall and using a singlular equation to calculate any point. When using machine learning, the calculation is different for each point because it is based on its neighbors rather than the overall relationship.

Clustering

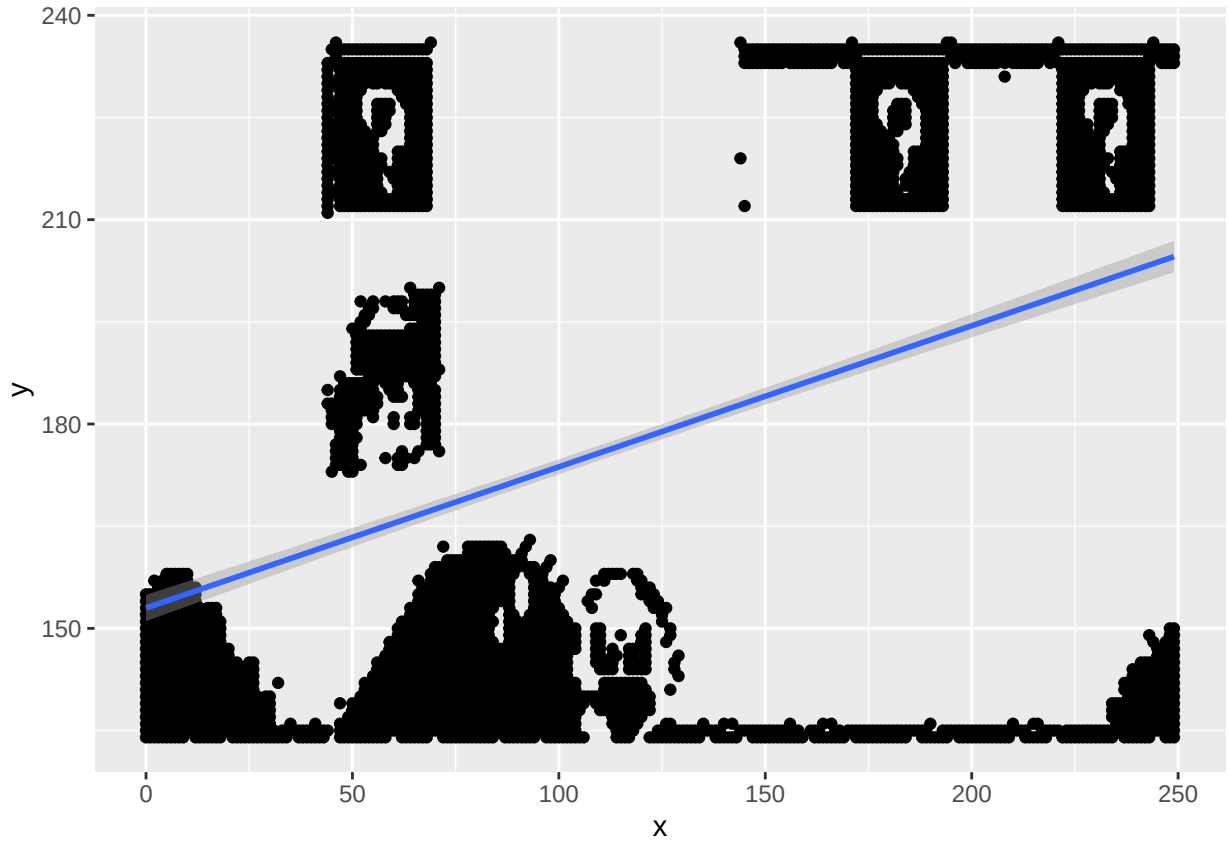
Labeled data is not always available. For these types of datasets, you can use unsupervised algorithms to extract structure. The k-means clustering algorithm and the k nearest neighbor algorithm both use the Euclidean distance between points to group data points. The difference is the k-means clustering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

does not use labeled data. 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.

Plot the dataset using a scatter plot.

```
ggplot(clustering_df, aes(x=x, y=y)) + geom_point() + geom_smooth(method=lm)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Fit the dataset using the k-means algorithm from k=2 to k=12.

```
# kmeans k = 2
km2 <- kmeans(clustering_df, 2, iter.max = 10, nstart = 25)
```

```
# kmeans k = 3
km3 <- kmeans(clustering_df, 3, iter.max = 10, nstart = 25)
```

```
# kmeans k = 4
km4 <- kmeans(clustering_df, 4, iter.max = 10, nstart = 25)
```

```
# kmeans k = 5
km5 <- kmeans(clustering_df, 5, iter.max = 10, nstart = 25)
```

```r
# kmeans k = 6
km6 <- kmeans(clustering_df, 6, iter.max = 10, nstart = 25)
```

```r
# kmeans k = 7
km7 <- kmeans(clustering_df, 7, iter.max = 10, nstart = 25)
```

```r
# kmeans k = 8
km8 <- kmeans(clustering_df, 8, iter.max = 10, nstart = 25)
```

```r
# kmeans k = 9
km9 <- kmeans(clustering_df, 9, iter.max = 10, nstart = 25)
```

```r
# kmeans k = 10
km10 <- kmeans(clustering_df, 10, iter.max = 10, nstart = 25)
```

```r
# kmeans k = 11
km11 <- kmeans(clustering_df, 11, iter.max = 10, nstart = 25)
```

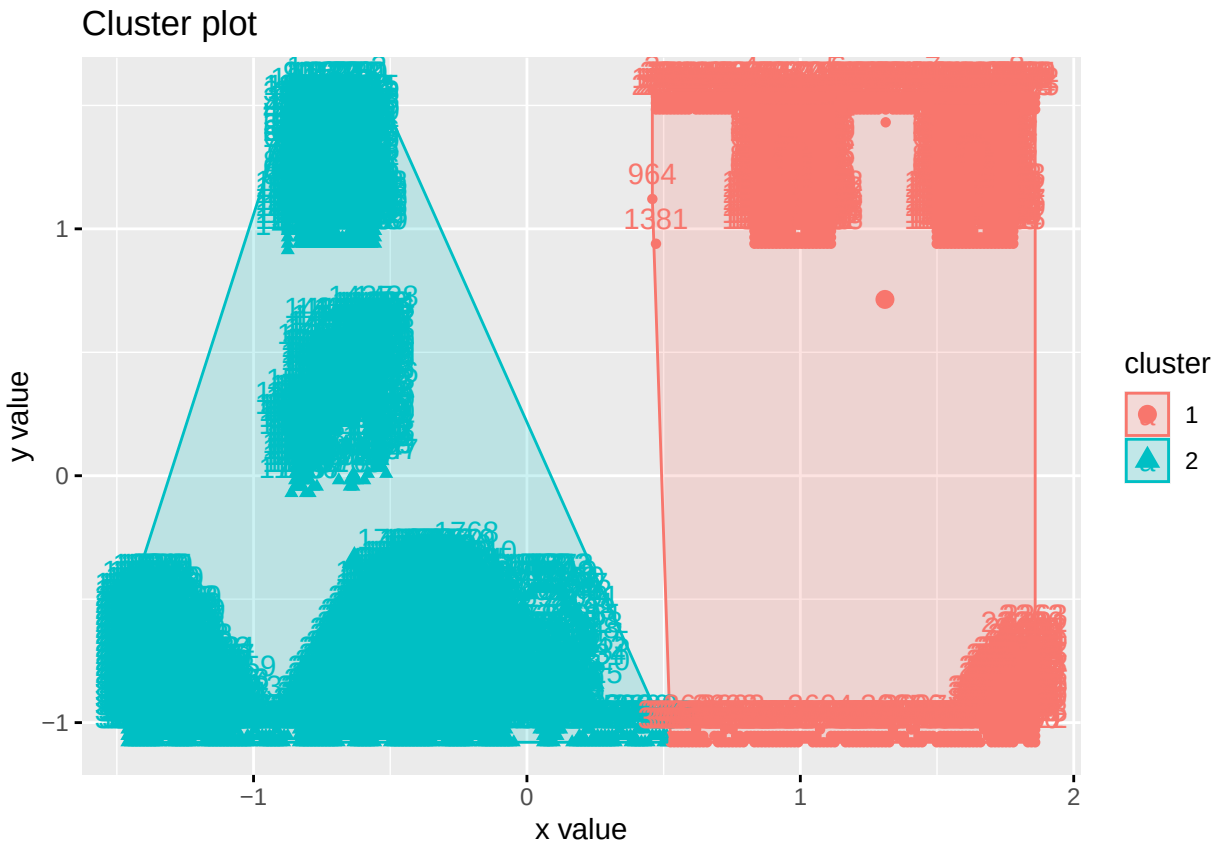
```r
# kmeans k = 12
km12 <- kmeans(clustering_df, 12, iter.max = 10, nstart = 25)
```

Create a scatter plot of the resultant clusters for each value of k.

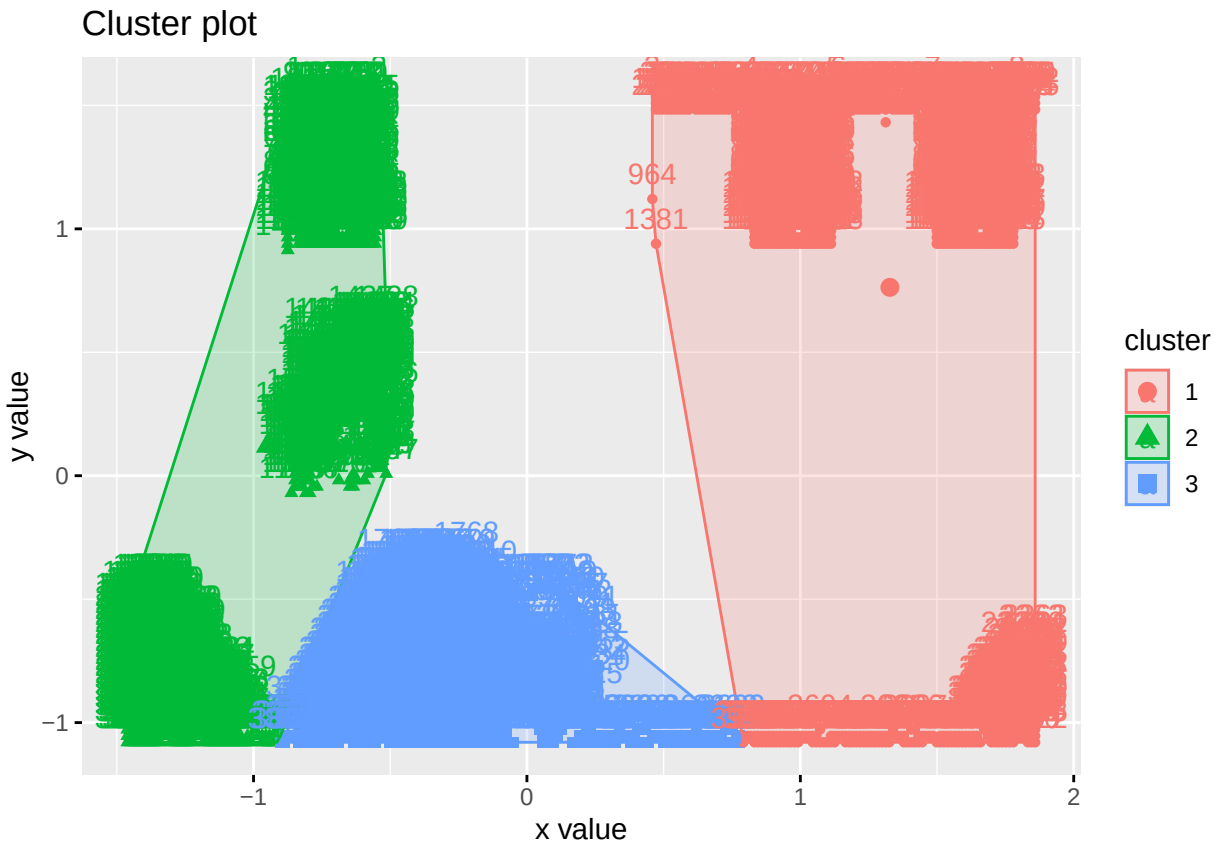
```r
library(factoextra)
```

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

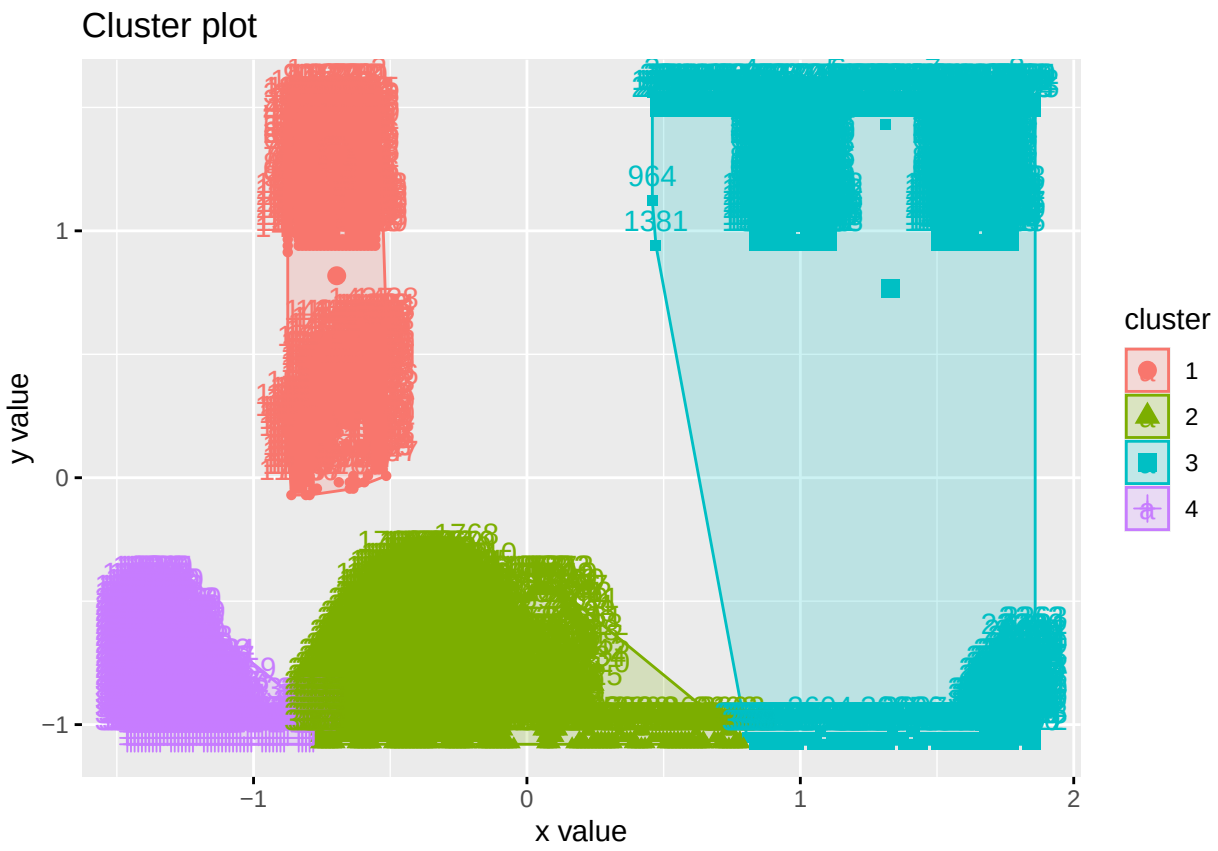
```r
fviz_cluster(km2, clustering_df)
```



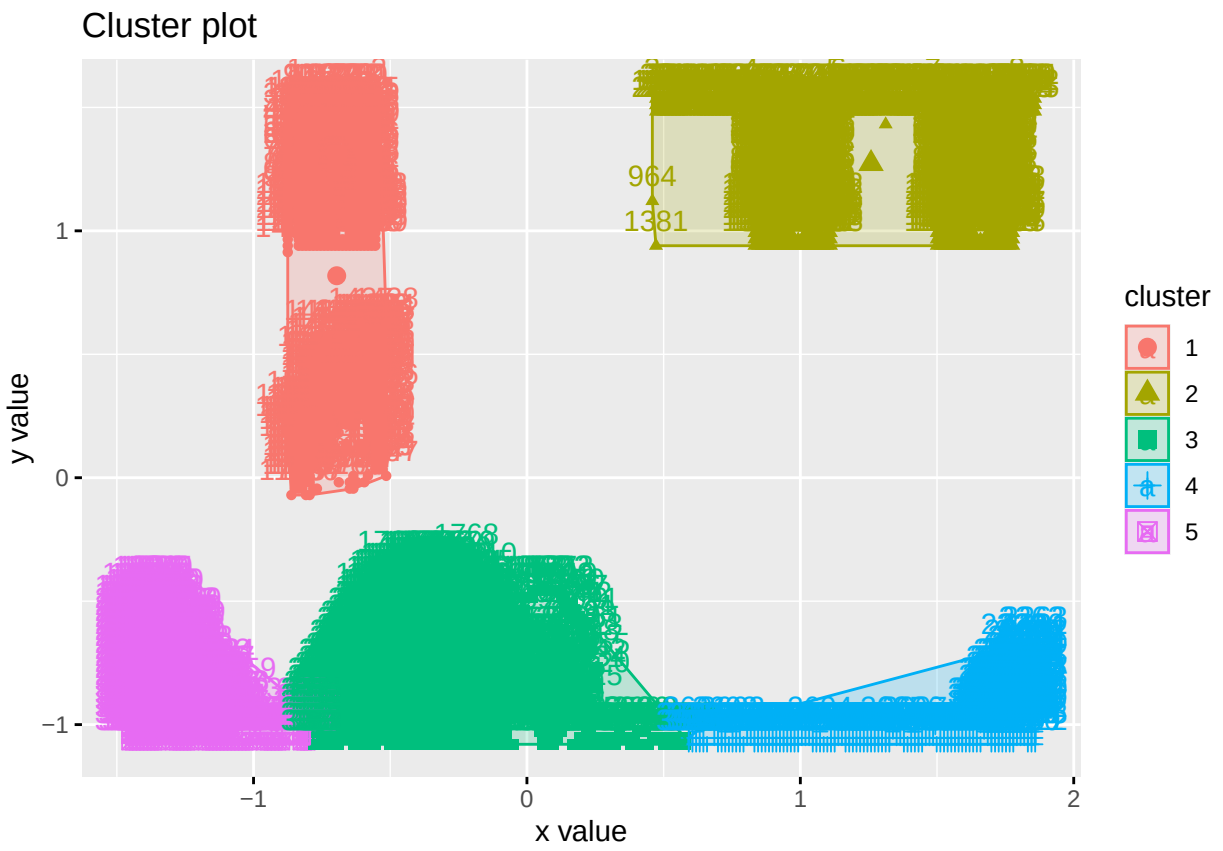

```
fviz_cluster(km3, clustering_df)
```



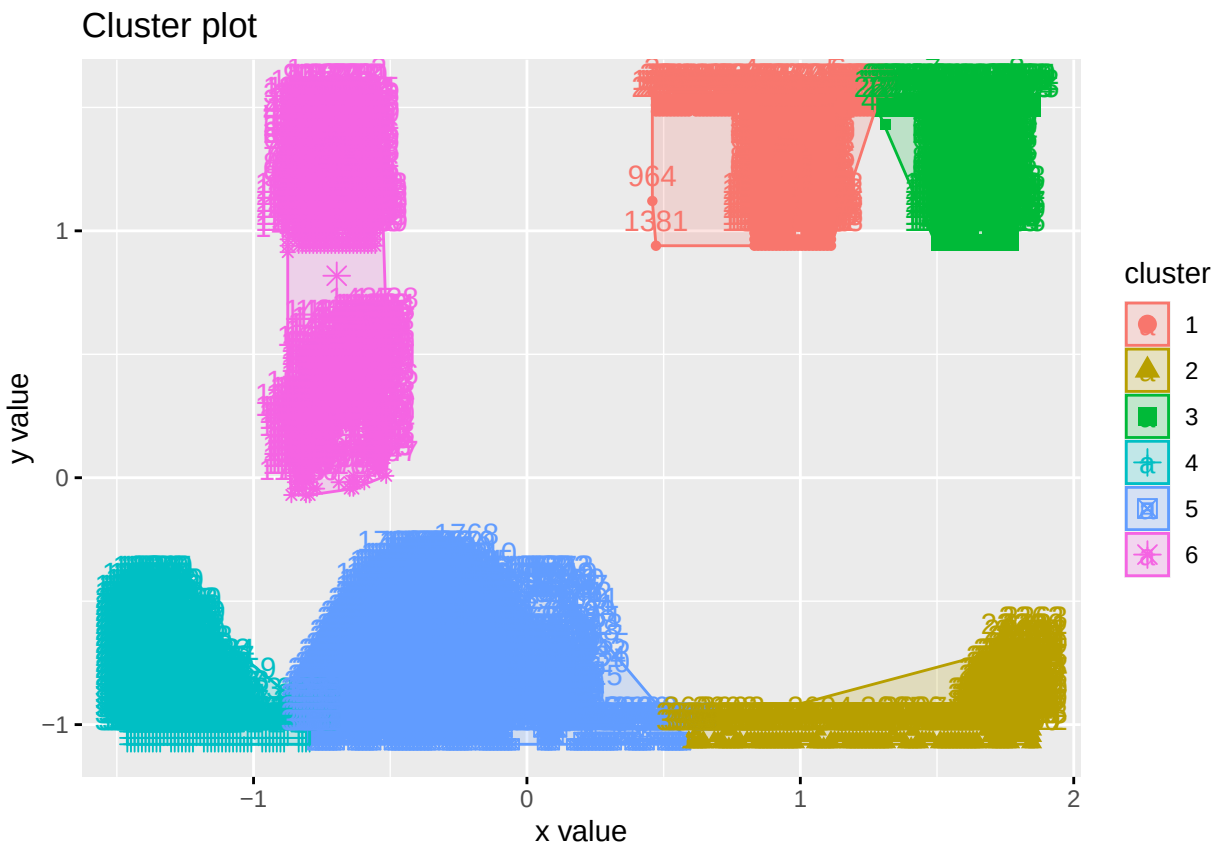

```
fviz_cluster(km4, clustering_df)
```



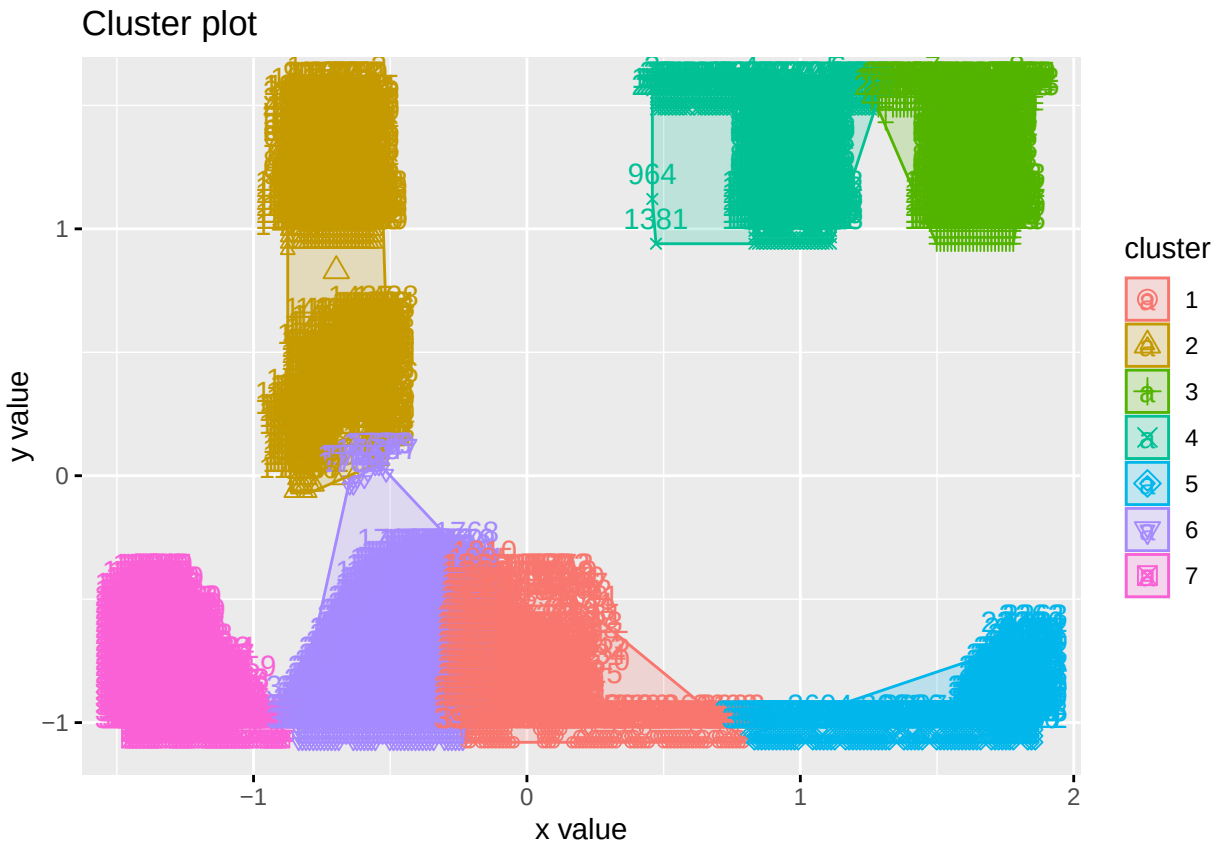

```
fviz_cluster(km5, clustering_df)
```



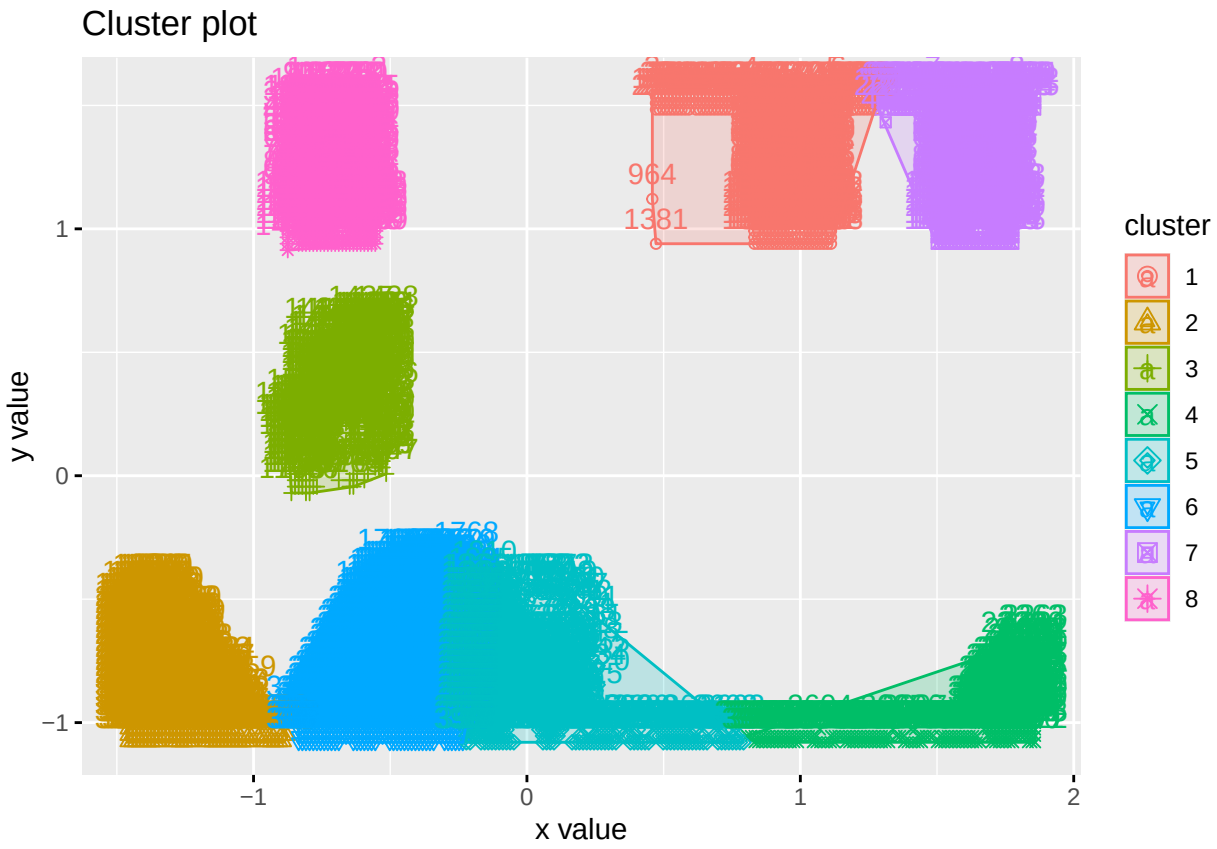

```
fviz_cluster(km6, clustering_df)
```



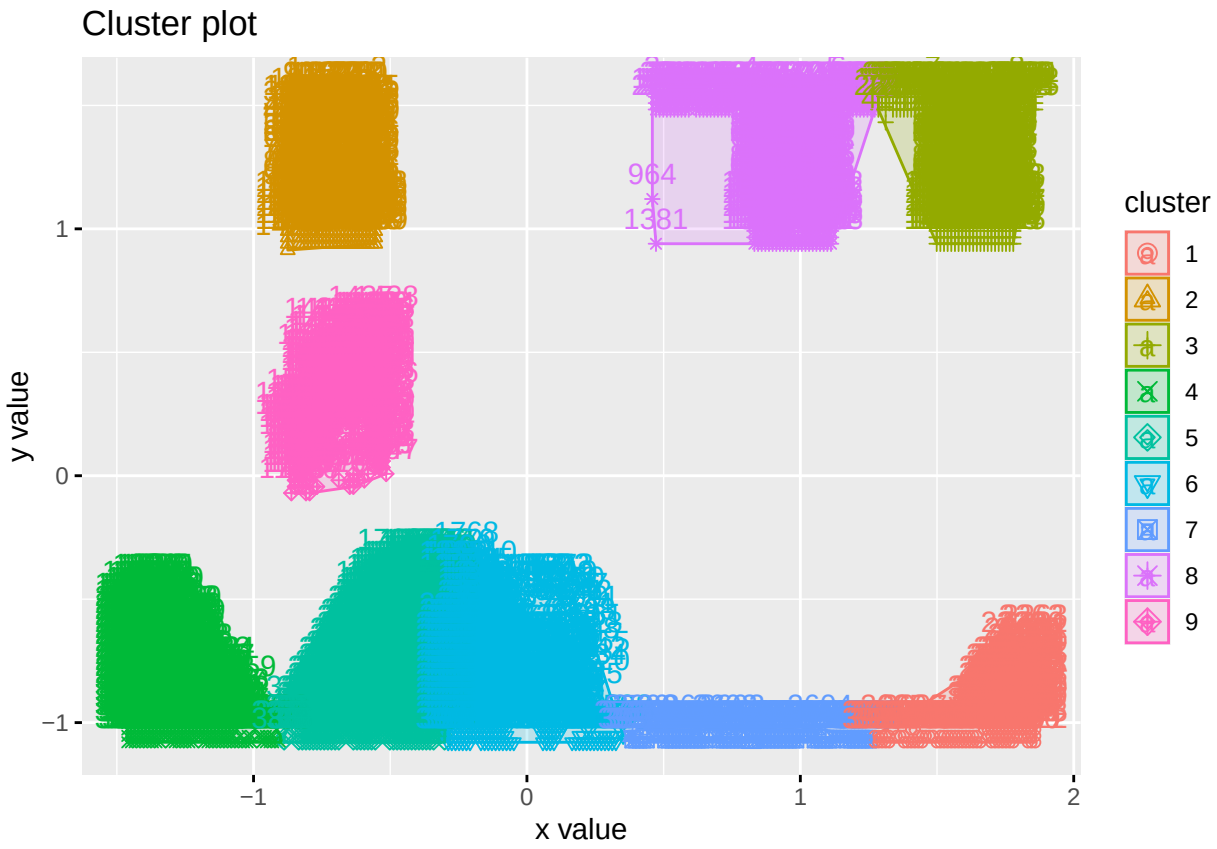

```
fviz_cluster(km7, clustering_df)
```



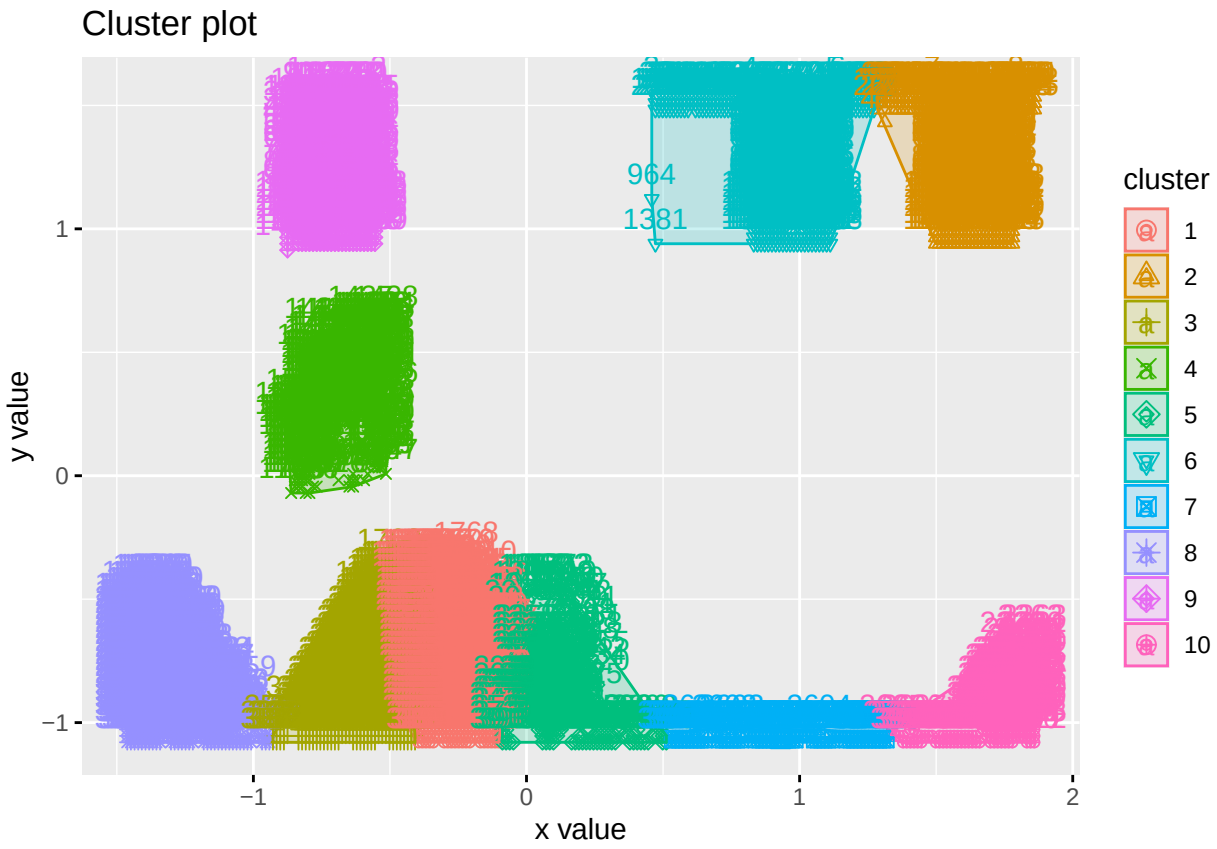

```
fviz_cluster(km8, clustering_df)
```



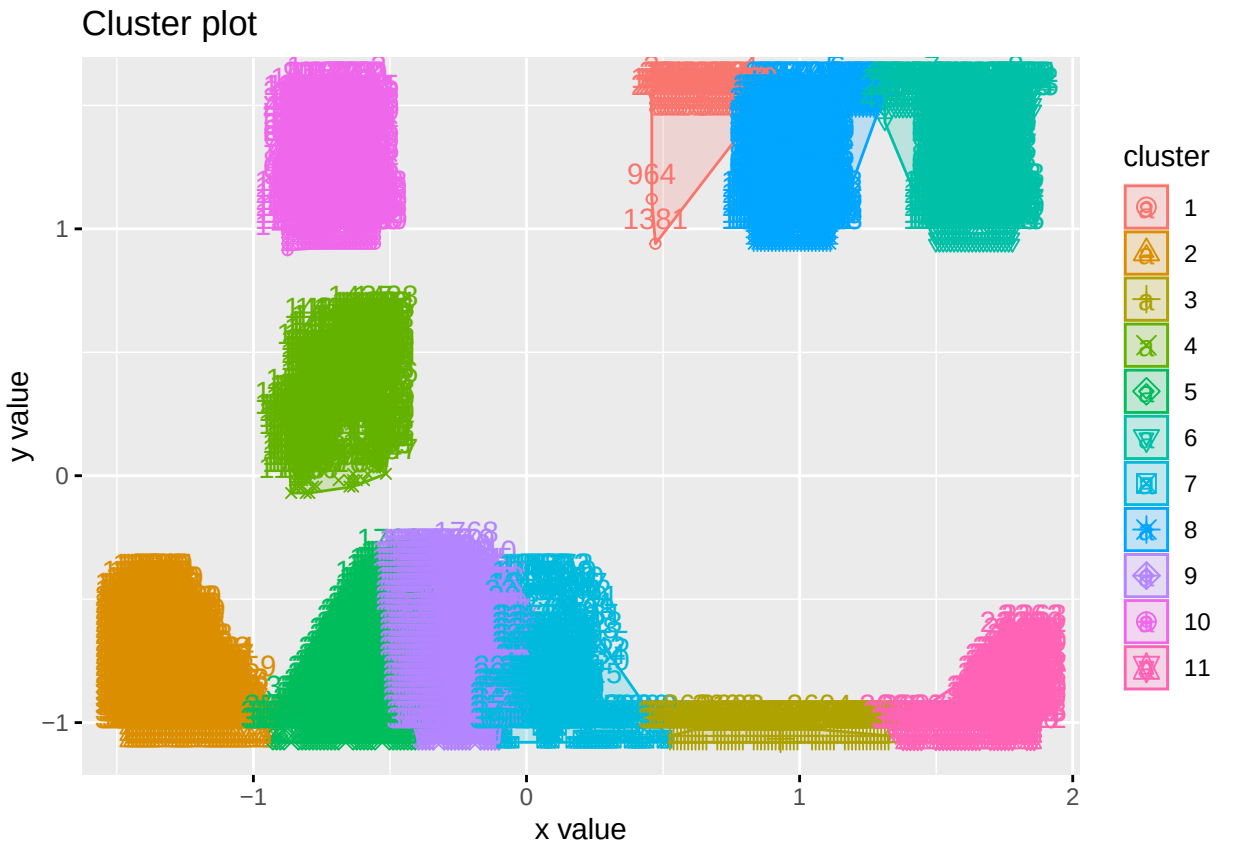

```
fviz_cluster(km9, clustering_df)
```



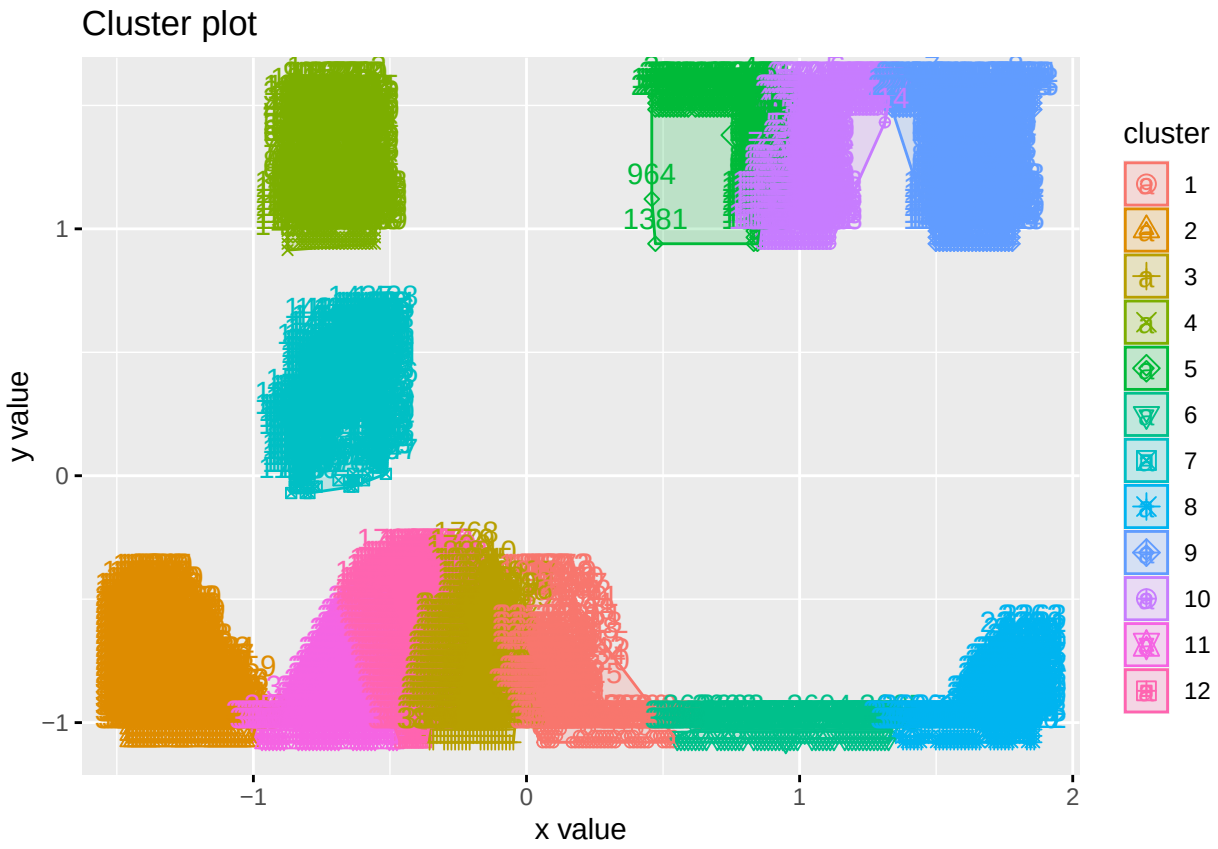

```
fviz_cluster(km10, clustering_df)
```




```
fviz_cluster(km11, clustering_df)
```

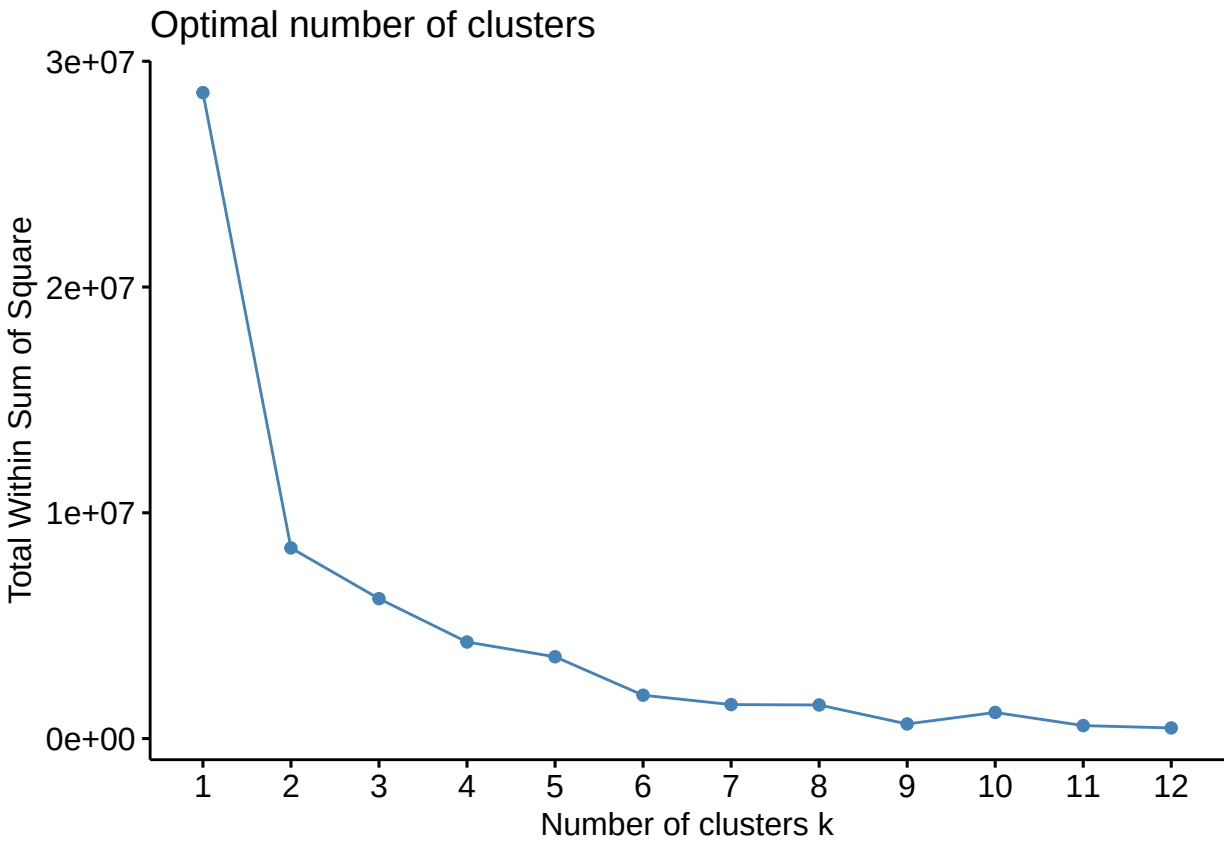



```
fviz_cluster(km12, clustering_df)
```



- 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.
- 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.

```
fviz_nbclust(clustering_df,
  FUNcluster = kmeans,
  method = c("wss"),
  k.max = 12,
  verbose = TRUE,
  print.summary = TRUE)
```



- 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?

  - Based on the graph generated above, the ideal number of clusters (k) appears to be 6. The graph of the total within-cluster sum of squares appears to bend at 6 which allows us to identify this as the ideal number of clusters for this data set.
  - After looking at this graph and looking back at the visualiizations of the clusters, I found it interesting to note that the 7 clusters were not what I would have expected or how I would have divided the data by looking at it. Looking at the visualization of 5, 6, and 7 clusters suppoorts the determination that 6 is the correct number of clusters for this data.
  - Looking at the visualization line graph with gap statistics rather than within-cluster sum of squares, the ideal number of clusters is 5. When looking at the silhouette method, 2 is indicated as the ideal number of clusters. Currently, I am not exactly sure how one would chose the best measurement for determining k, but based on the within-cluster sum of squares and the visual of the clusters, I would choose 6 clusters.