Moore_Week11_12

Heather

2025-02-20

Introduction to Machine Learning

Importing the data

```
binary = read.csv("C:/Users/14027/Documents/Graduate_Schoolish/DSC_520/Code_Hmwk/binary-classifier-data

trinary = read.csv("C:/Users/14027/Documents/Graduate_Schoolish/DSC_520/Code_Hmwk/trinary-classifier-dat
```

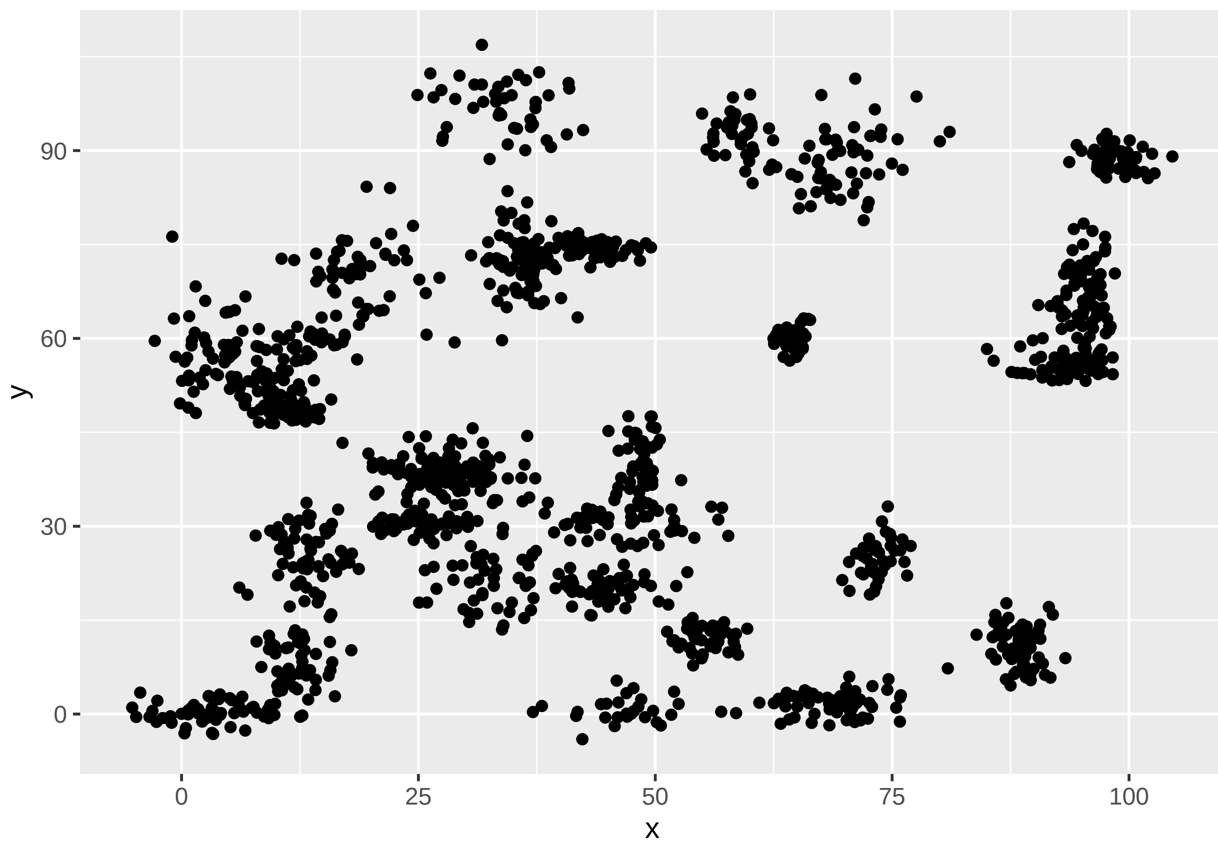
Plot the data from each data set using a scatter plot

```
library(GGally)
```

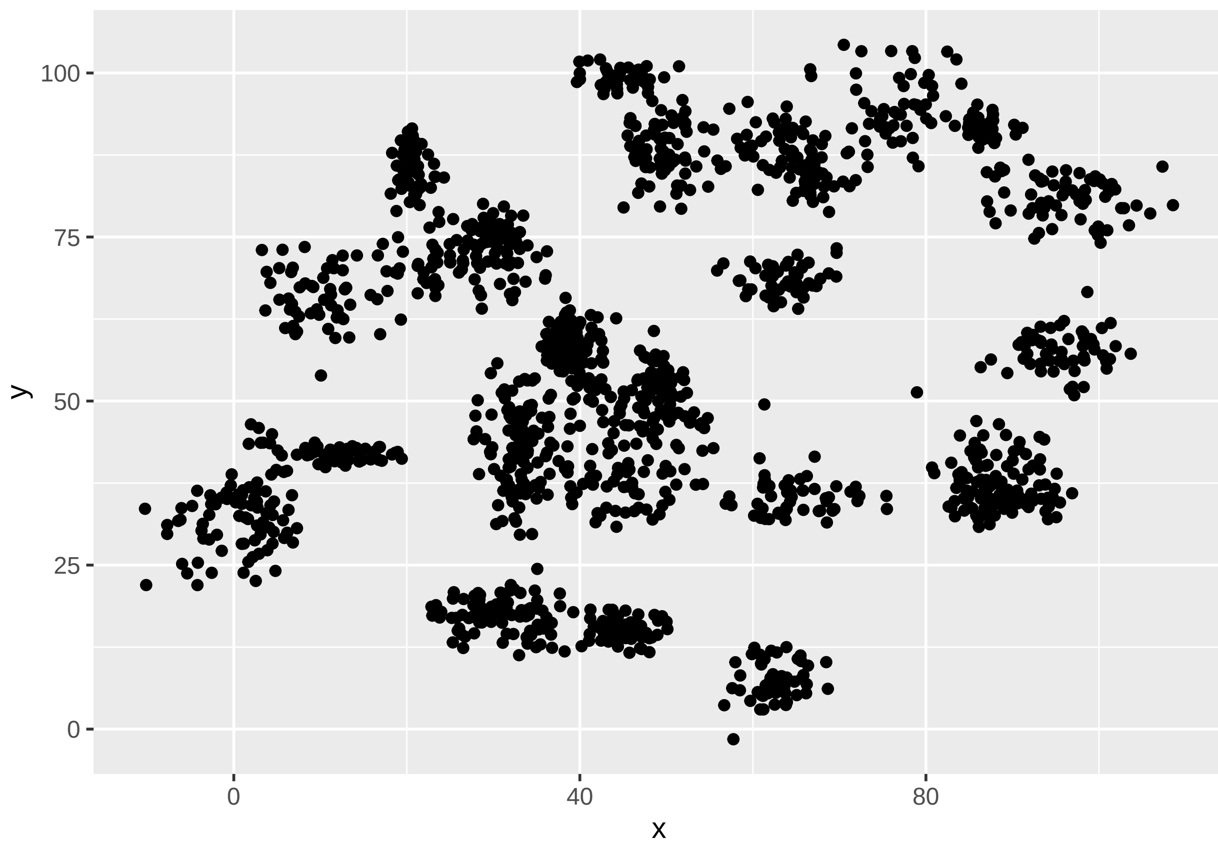
```
## Loading required package: ggplot2
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
library(ggplot2)
#Correlation of binary
#bin = ggpairs(binary)
ggplot(binary, aes(x,y)) + geom_point()
```




```r
#Correlation og trinary
ggplot(trinary, aes(x,y)) +geom_point()
```



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

```r
library(stats)
#binary
vect1 = binary$x
vect2 = binary$y
CalculateEuclideanDistance <- function(vect1, vect2) sqrt(sum((vect1 - vect2)^2))

print("Euclidean distance between vect1 and vect2 from the binary dataset is: ")
```

```
## [1] "Euclidean distance between vect1 and vect2 from the binary dataset is: "
```

```r
# Calling CalculateEuclideanDistance function
CalculateEuclideanDistance(vect1, vect2)
```

```
## [1] 1411.959
```

```r
#trinary
vect1 = trinary$x
vect2 = trinary$y
```

```r
CalculateEuclideanDistance <- function(vect1, vect2) sqrt(sum((vect1 - vect2)^2))

print("Euclidean distance between vect1 and vect2 from the trinary dataset is: ")
```

```
## [1] "Euclidean distance between vect1 and vect2 from the trinary dataset is: "
```

```r
# Calling CalculateEuclideanDistance function
CalculateEuclideanDistance(vect1, vect2)
```

```
## [1] 1357.734
```

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

```r
#Split into test and train sets in each data set
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
```

```r
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.2.2
```

```r
library(class)
train_index = sample(nrow(binary), size = floor(nrow(binary) * 0.75))
## Binary
bin_train = binary[train_index, ]
bin_test = binary[-train_index,]

## Trinary
tri_train = trinary[train_index,]
tri_test = trinary[-train_index,]
```

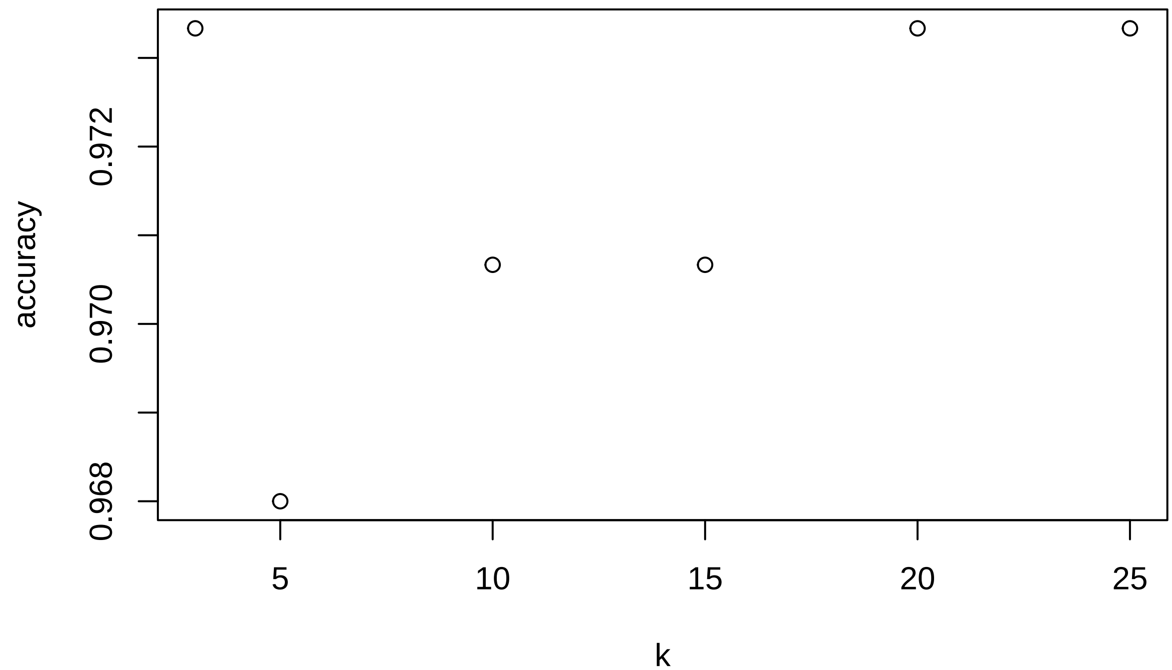
Fit the Binary Model

```r
#For loop for Binary
k_vals = c(3,5,10,15,20,25)
bin_accuracy = data.frame(k = k_vals, accuracy = NA)

# Loop through each k value
for (i in 1:length(k_vals)) {
  k = k_vals[i]

  # Perform k-NN classification
  bin = knn(train = bin_train, test = bin_test, cl = bin_train$label, k = k)

  # Get true labels
  labels = bin_test$label

  # Calculate accuracy
  bin_acc = mean(labels == bin)

  # Store the accuracy in the data frame
  bin_accuracy$accuracy[i] = bin_acc
}

# Print the final data frame
print(bin_accuracy)
```

```
##    k  accuracy
## 1  3 0.9733333
## 2  5 0.9680000
## 3 10 0.9706667
## 4 15 0.9706667
## 5 20 0.9733333
## 6 25 0.9733333
```

Model Graph

```r
plot(bin_accuracy)
```



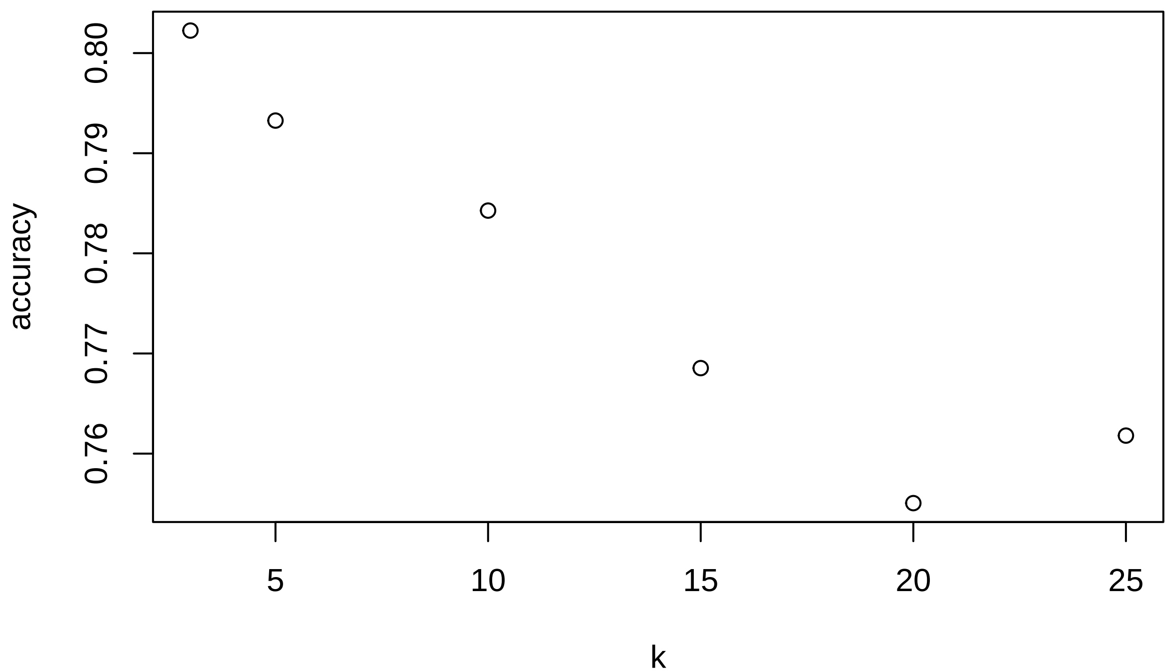
Creation of the Trinary Model

```r
#For loop for Trinary
k_vals = c(3,5,10,15,20,25)
tri_accuracy = data.frame(k = k_vals, accuracy = NA)
# Loop through each k value
for (i in 1:length(k_vals)) {
  k = k_vals[i]

  # Perform k-NN classification
  bin = knn(train = tri_train, test = tri_test, cl = tri_train$label, k = k)

  # Get true labels
  labels = tri_test$label

  # Calculate accuracy
  bin_acc = mean(labels == bin)

  # Store the accuracy in the data frame
  tri_accuracy$accuracy[i] = bin_acc
}

# Print the final data frame
print(tri_accuracy)
```

```
##    k  accuracy
## 1  3 0.8022472
## 2  5 0.7932584
## 3 10 0.7842697
## 4 15 0.7685393
## 5 20 0.7550562
## 6 25 0.7617978
```

Creating the Plot

```r
plot(tri_accuracy)
```



Do you think a linear classifier would work well on these data sets?

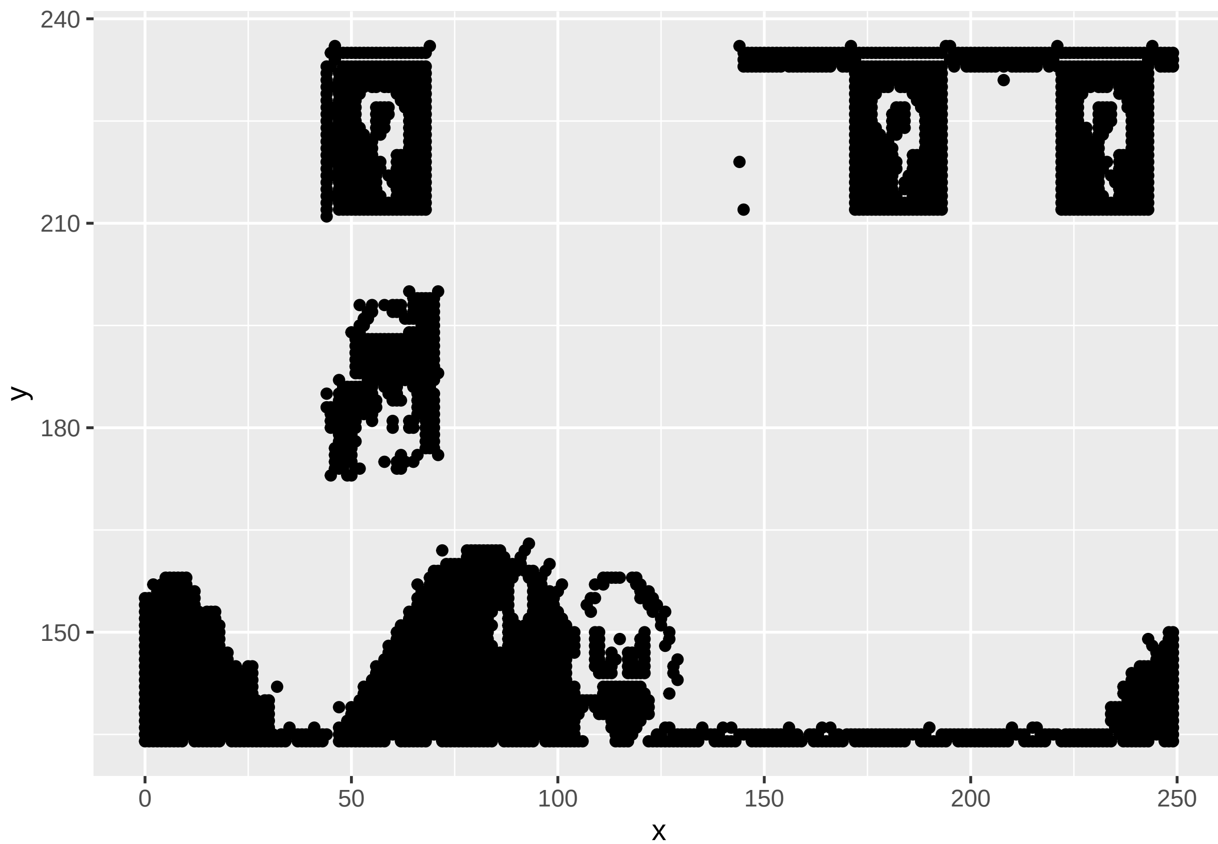
- Based on the the scatter plot from above it shows that there is no obvious linear relationship in the data in either data set.So I would say that the linear classifiers would not work well.

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

The two classifiers last week had an accuracy of 54% and 83% which are smaller than what the accuracies are from this week.The logistic regression was supposed to predict a linear model but, the data isn't linear while the KNN predicts the closest data points in the training set which worked a lot better for both data set this time around.

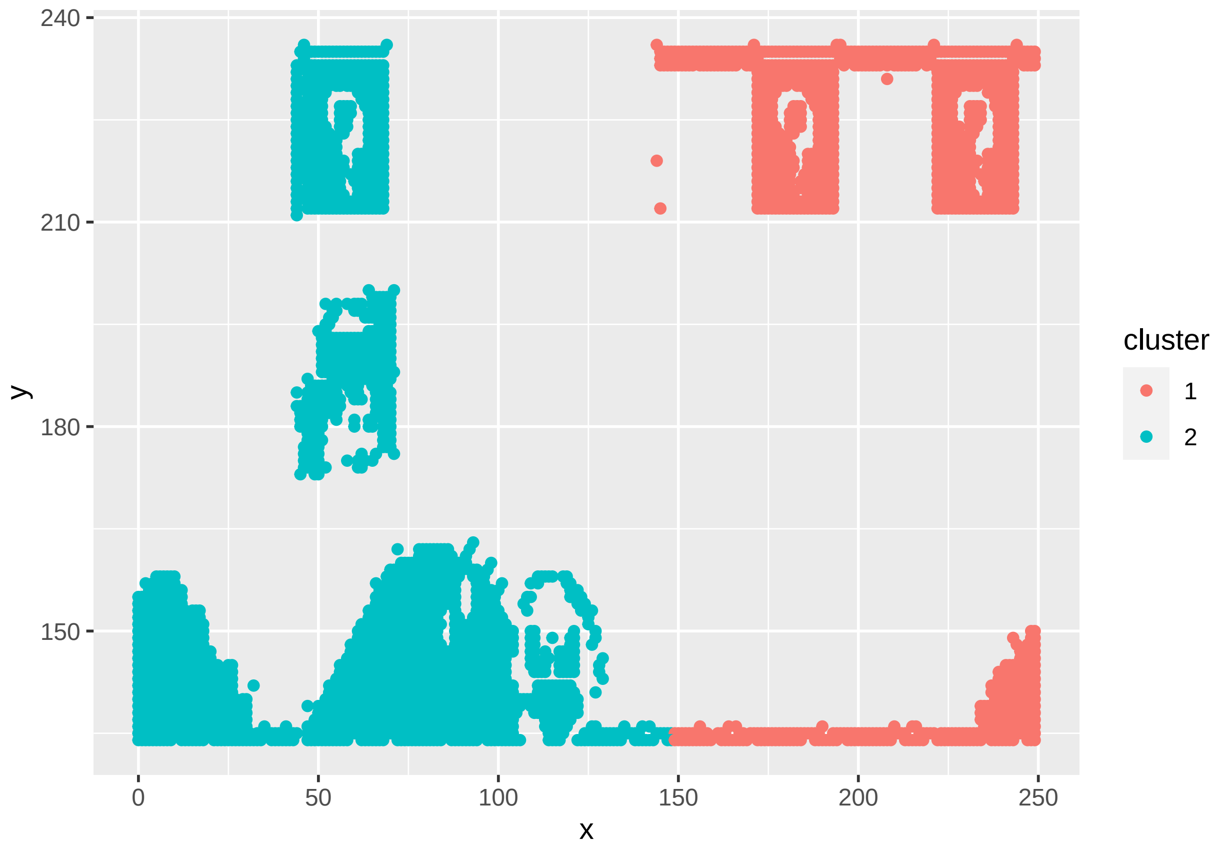
Clustering

```
library(stats)
# Importing the Data
cluster = read.csv("C:/Users/14027/Documents/Graduate_Schoolish/DSC_520/Code_Hmwk/clustering-data.csv")
#Plot the data set in a scatter plot
ggplot(cluster, aes(x,y)) +geom_point()
```

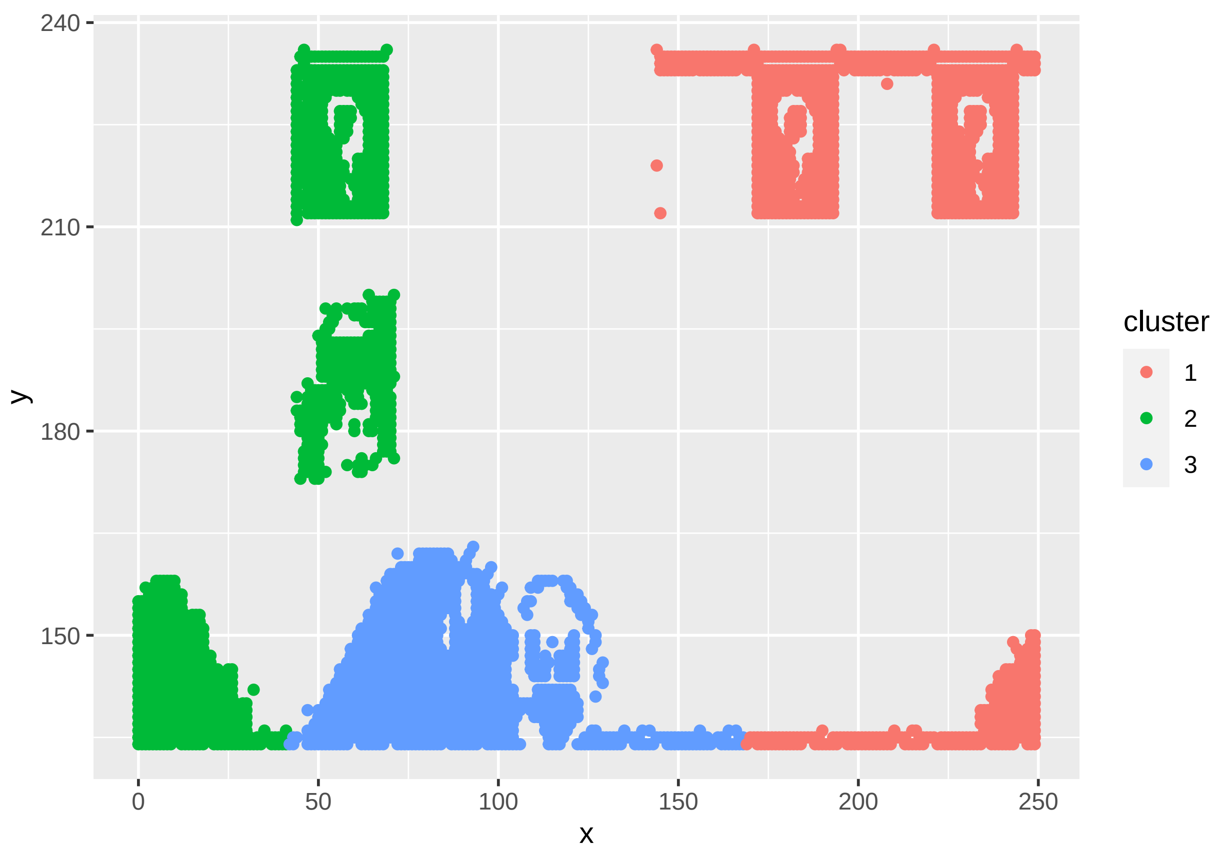


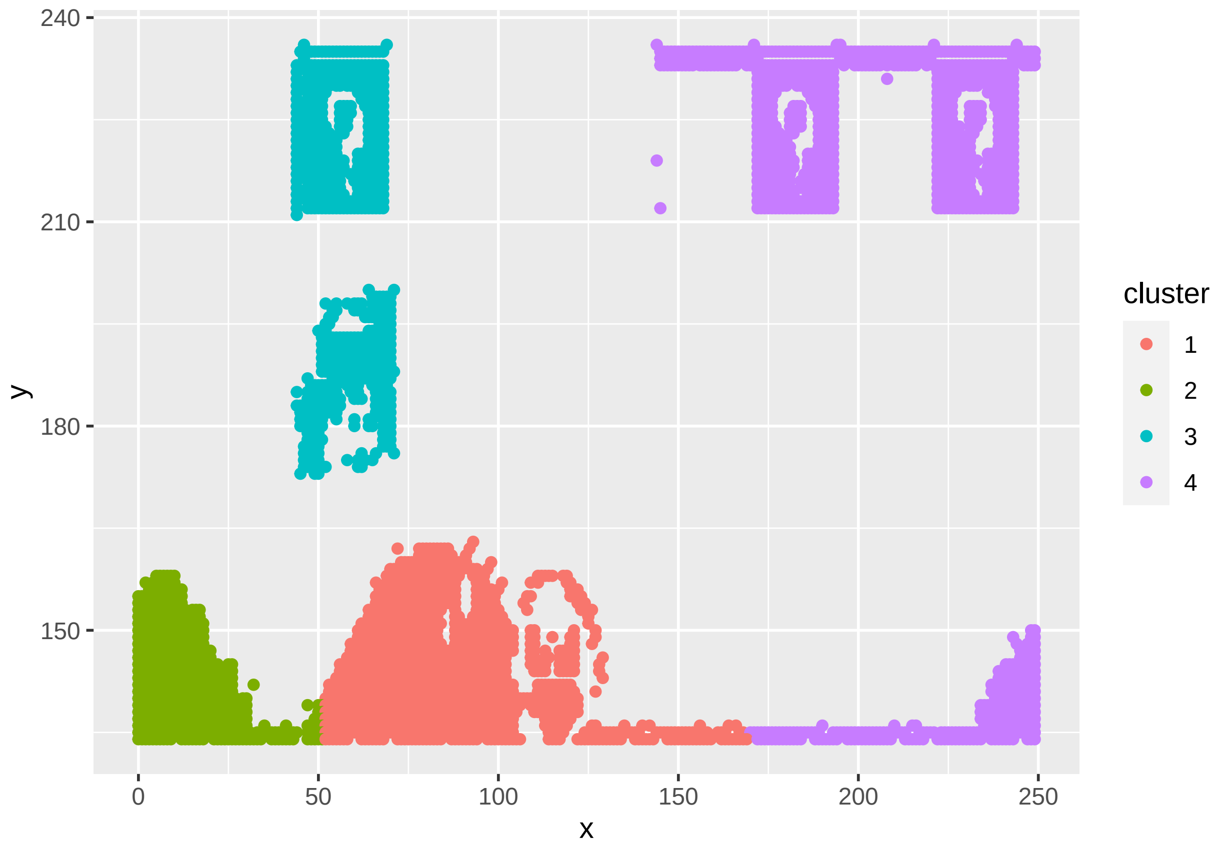
Fit the dataset using the k-means algorithm from k=2 to k=12. Create a scatter plot of the resultant clusters for each value of k.

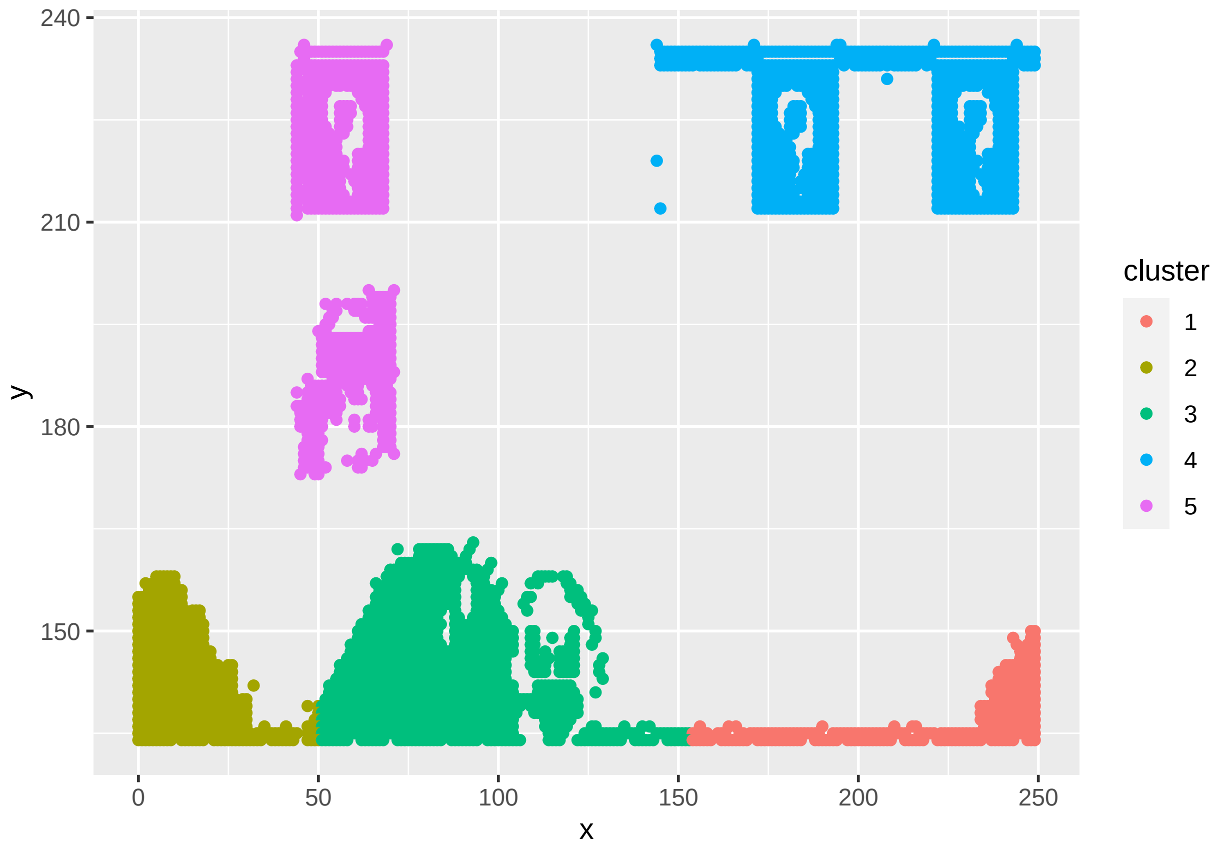
```
k_group = c(2:12)
k_dist = data.frame()
for(x in k_group) {
  set.seed(123)
  cluster_model = kmeans_results = kmeans(cluster, x, nstart = 10)
  cluster$cluster = as.factor(cluster_model$cluster)
  plot = ggplot(cluster, aes(x,y, color = cluster)) + geom_point()

  print(plot)
}
```

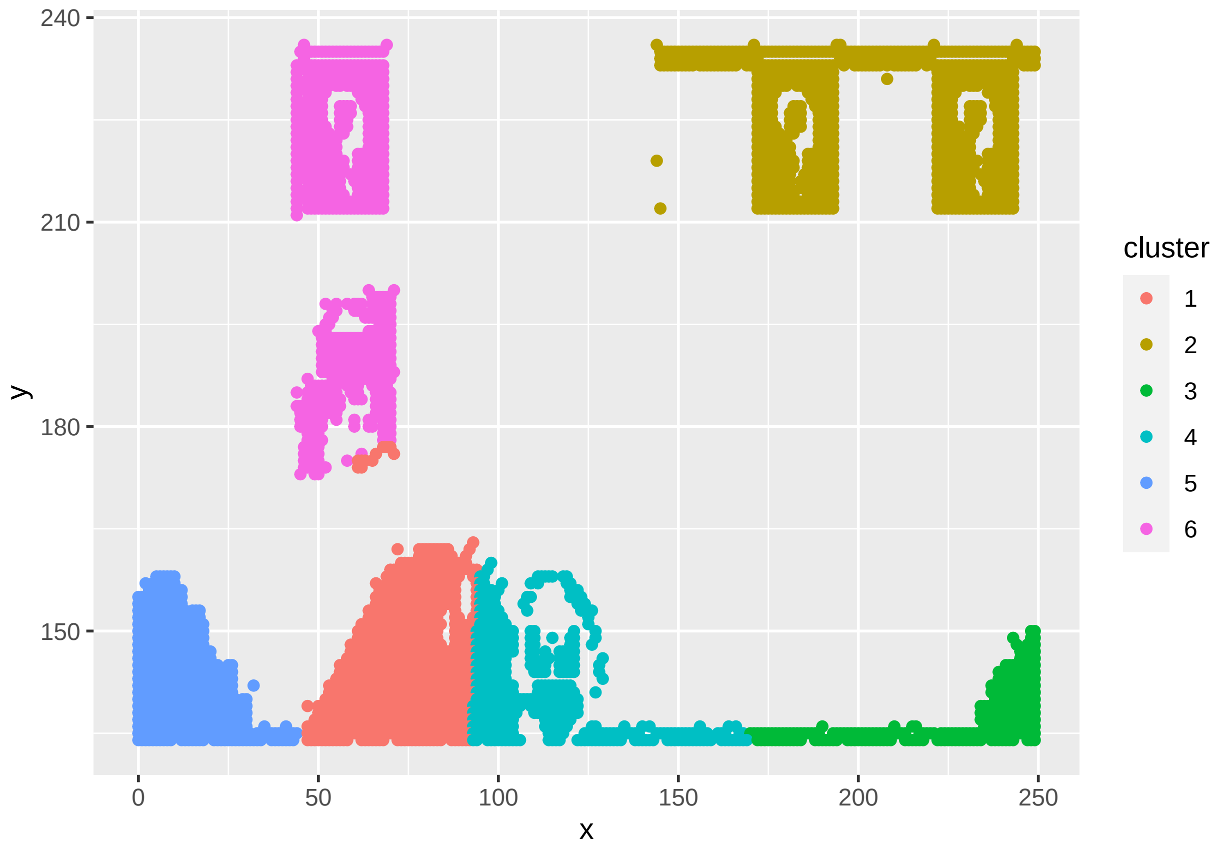


240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2

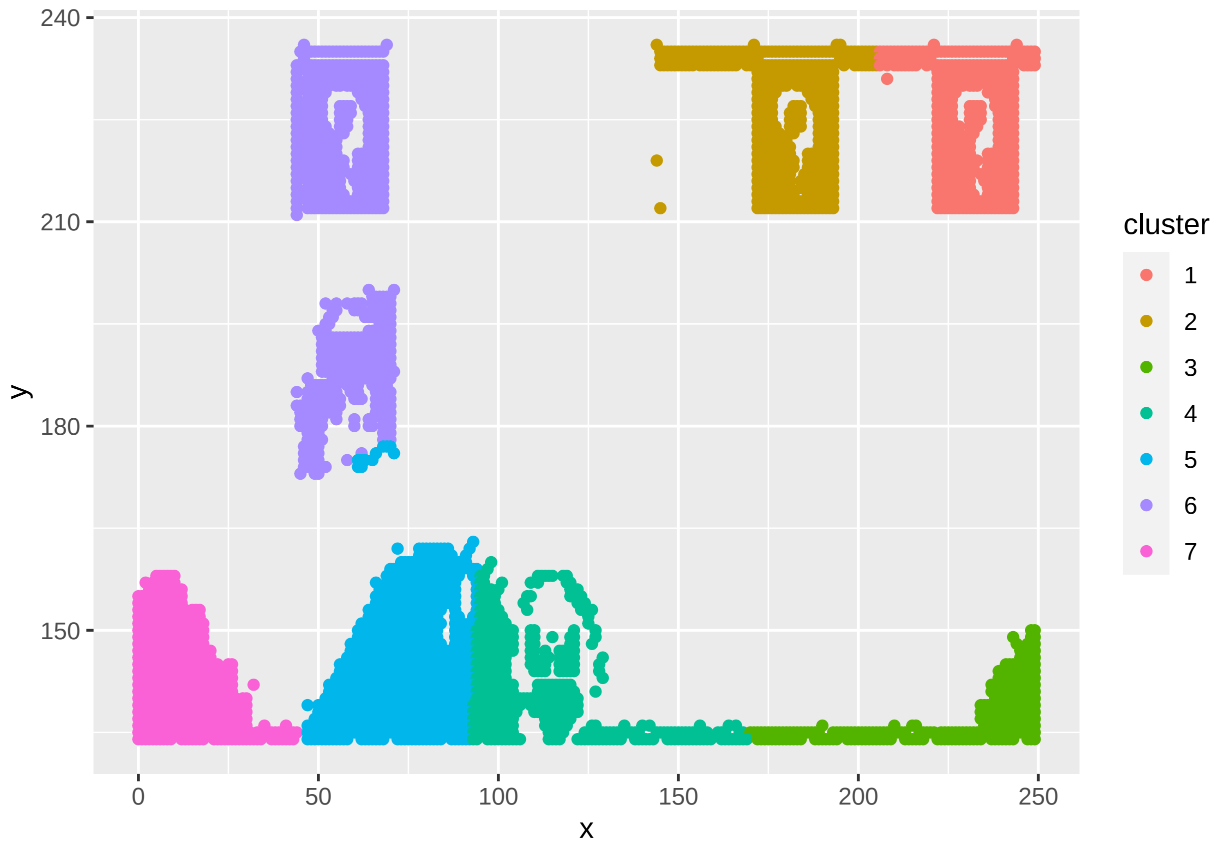

240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3

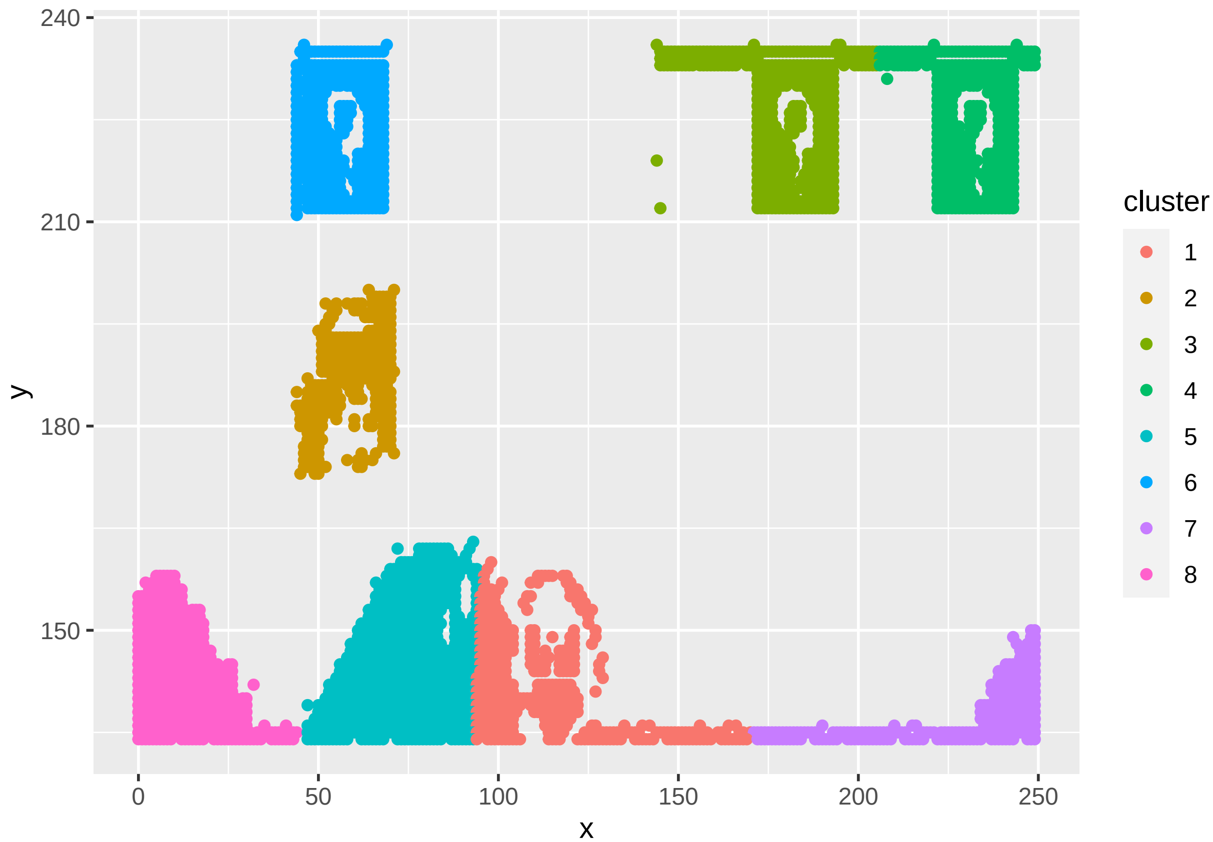

240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4


240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5

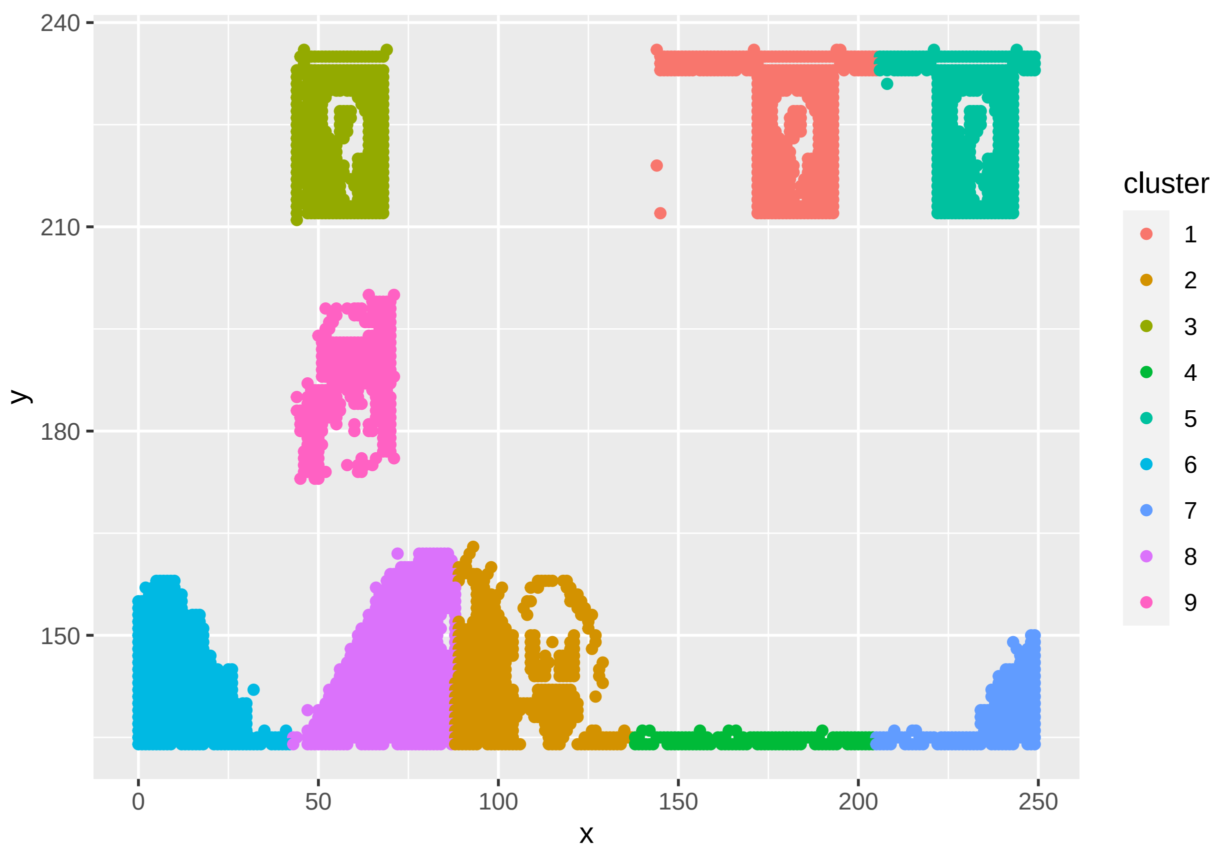


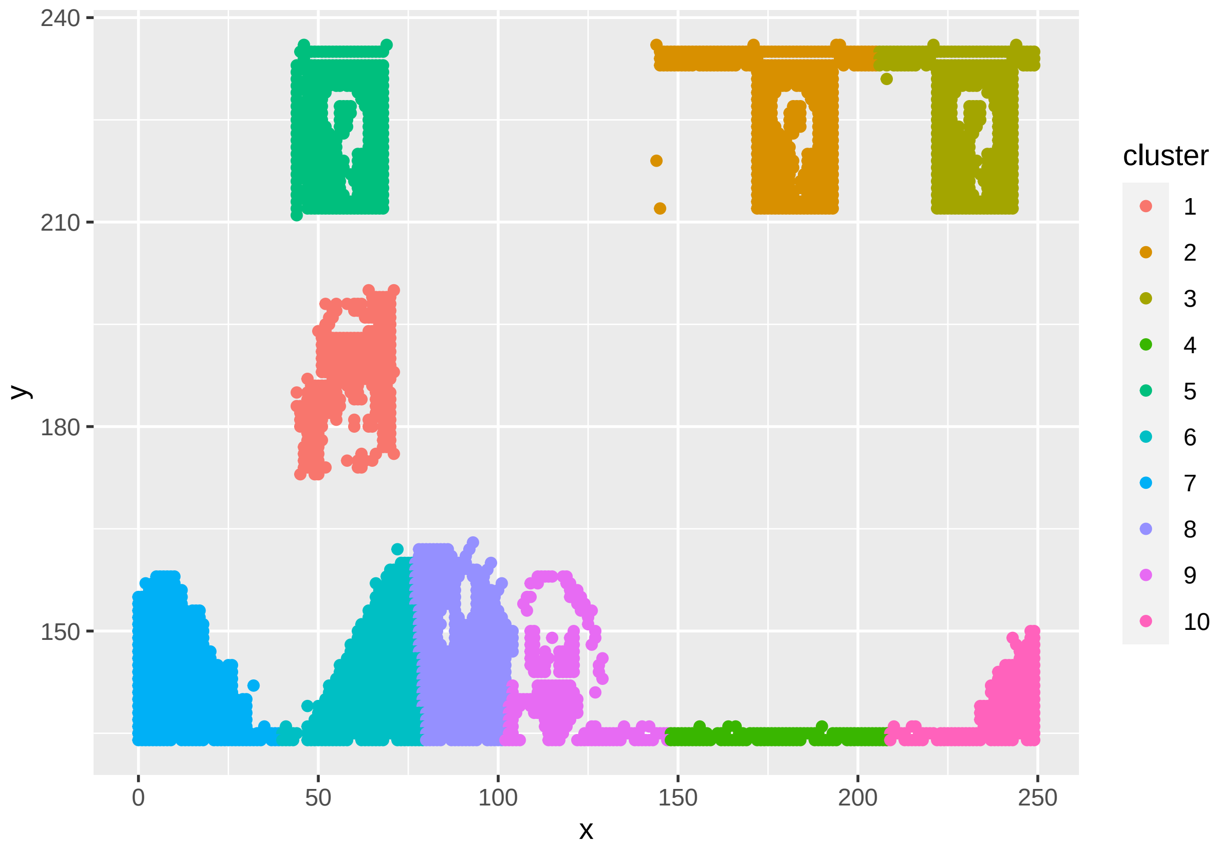
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6

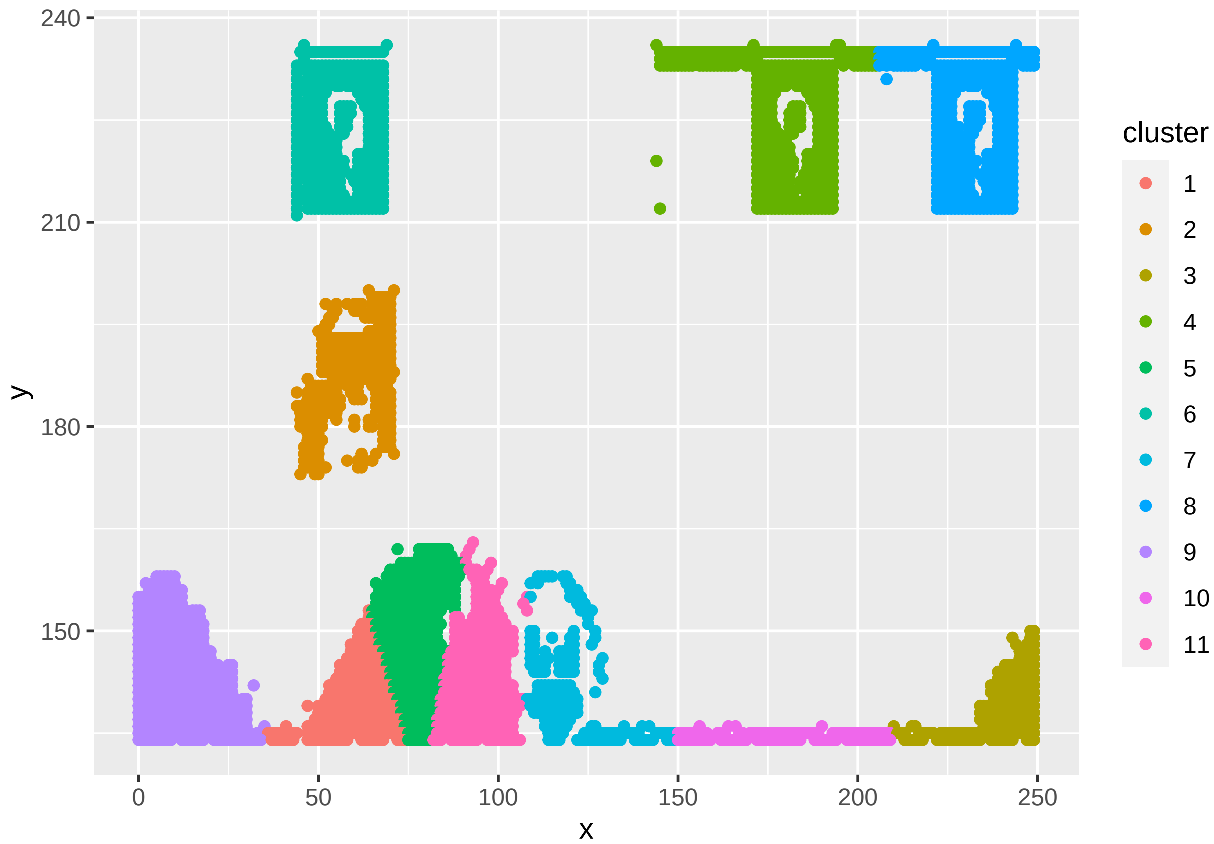

240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7



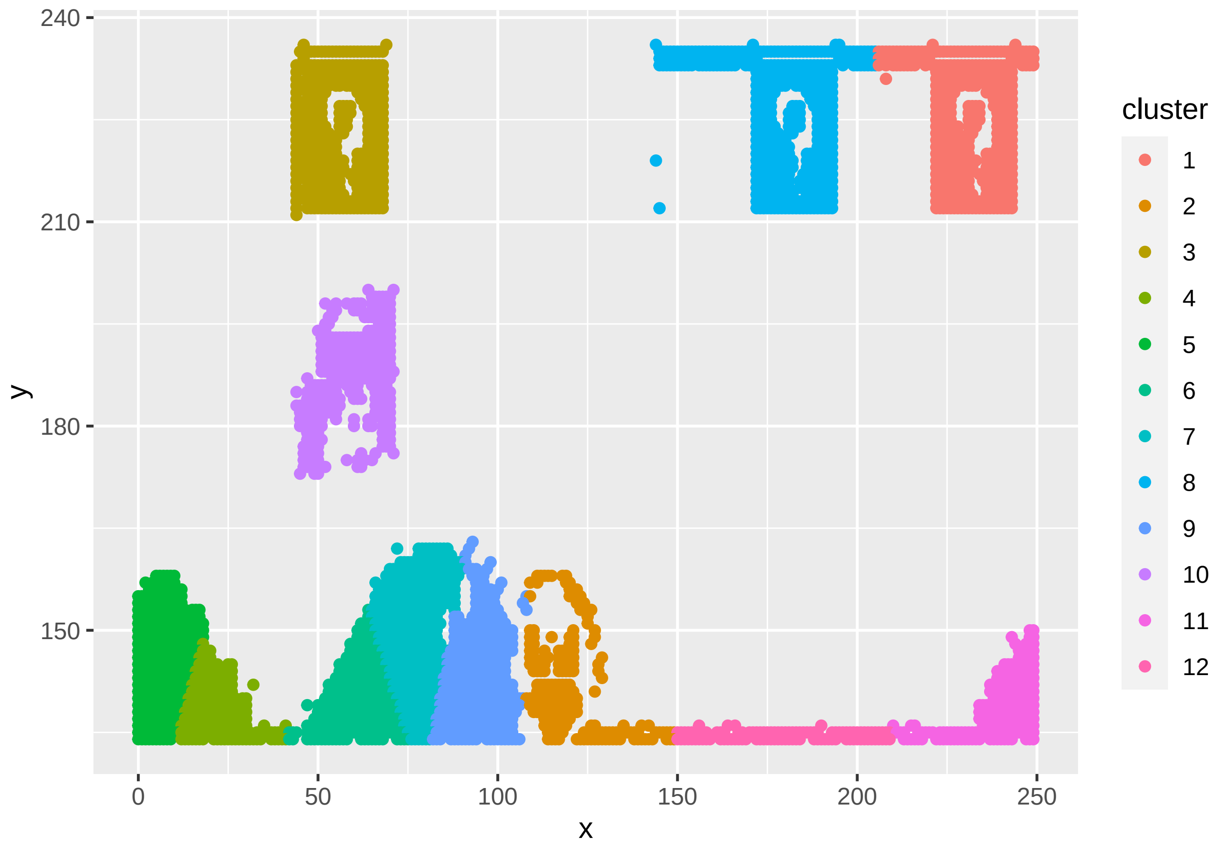
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7
8


240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7
8
9


240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7
8
9
10

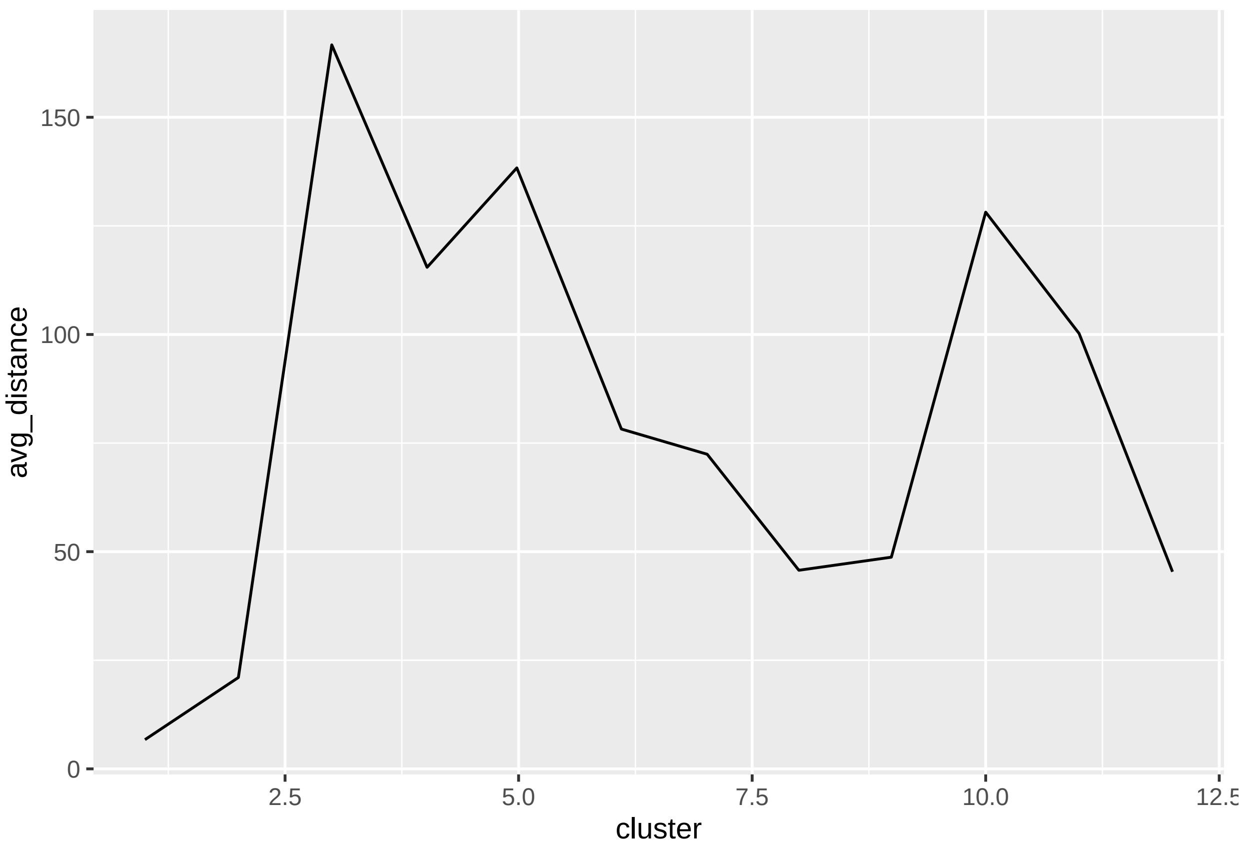


240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7
8
9
10
11



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

```r
# Loop through each k value
set.seed(123)   # Set seed for reproducibility

# Run k-means clustering
cluster_model = kmeans(cluster, centers = 12, nstart = 10)

# Store the centers (convert to list to store in a data frame)
k_center = list(cluster_model$centers)
k_center = as.data.frame(k_center)
k_center$avg_distance = sqrt((k_center$y - k_center$x)^2)

ggplot(k_center, aes(x = cluster, y = avg_distance)) +
  geom_line()
```



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?

- My graph does not follow the proper curve of a elbow curve. However after doing some research the elbow point is where k has reached the point of diminishing returns. The closest point that matches that definition in my opinion would be k=8.