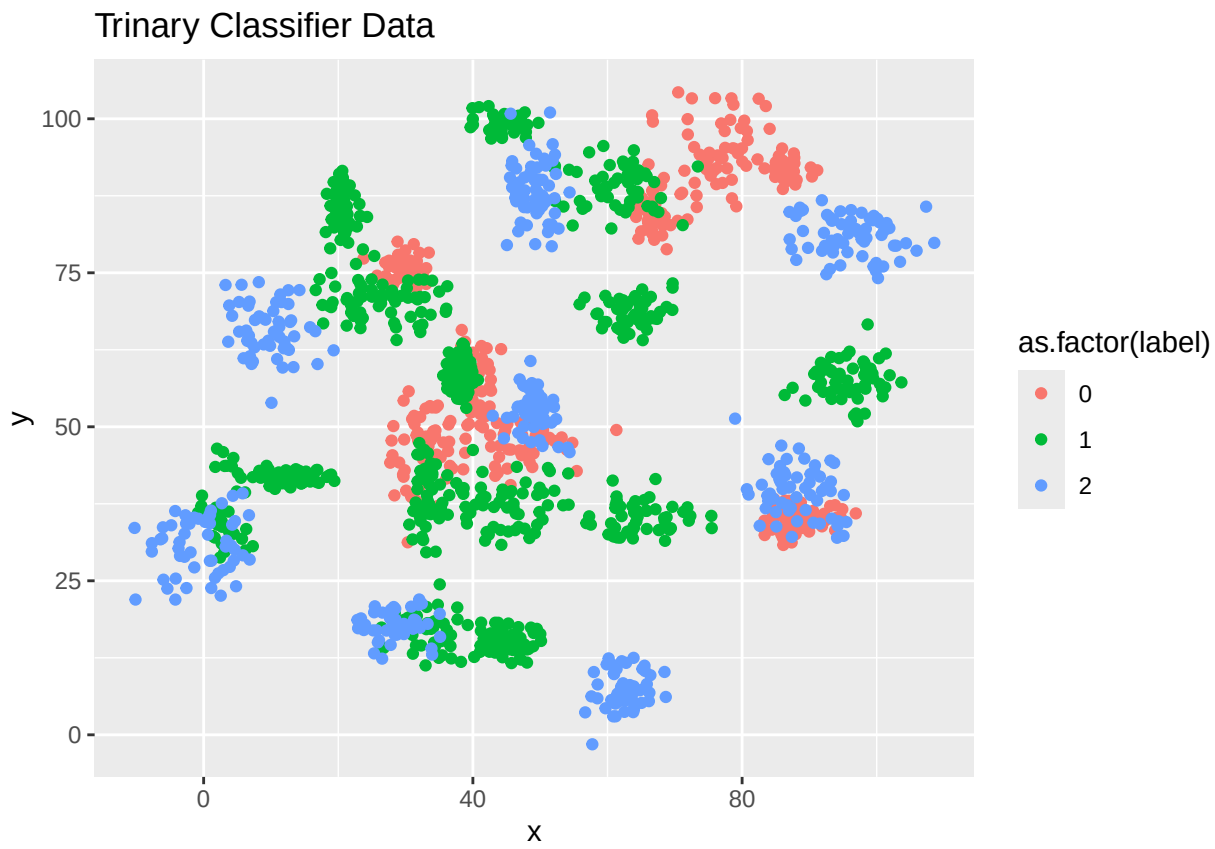
Week11

Bernard Owusu Sefah

2024-05-26

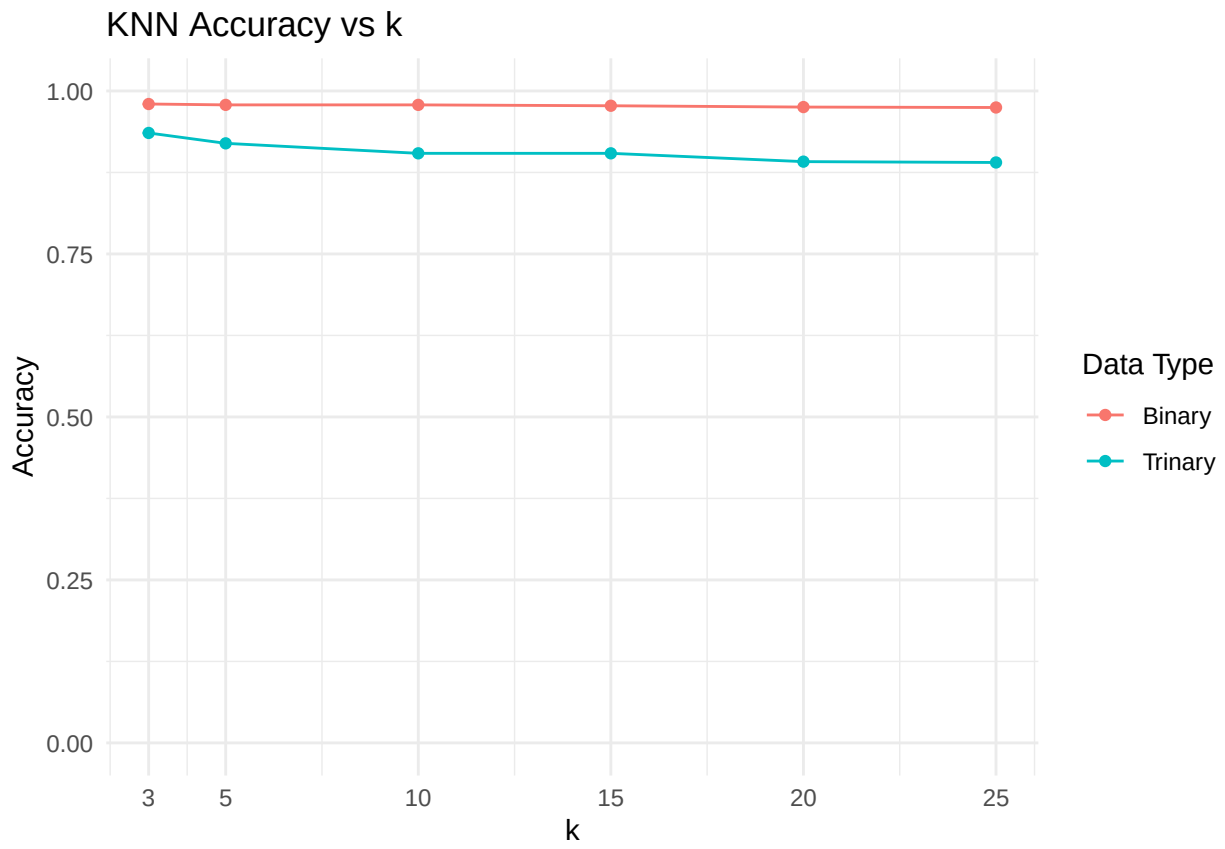
```r
# Load required libraries
library(ggplot2)
library(class)  # For efficient KNN implementation
library(tidyr)   # For data reshaping

# Exercise 1: Introduction to Machine Learning

# Read in binary and trinary classifier datasets
binary_data <- read.csv("binary-classifier-data.csv")
trinary_data <- read.csv("trinary-classifier-data.csv")

# Create scatter plots colored by label
# Binary data
ggplot(binary_data, aes(x=x, y=y, color=as.factor(label))) +
  geom_point() +
  ggtitle("Binary Classifier Data")
```


```
# Trinary data
ggplot(trinary_data, aes(x=x, y=y, color=as.factor(label))) +
  geom_point() +
  ggtitle("Trinary Classifier Data")
```




```r
# Function to fit KNN model and calculate accuracy
fit_knn <- function(data, k) {

  # Split data into features (x) and labels (y)
  x <- data[,2:3]
  y <- data[,1]

  # Fit KNN model using 'knn' function from 'class' package
  predictions <- knn(train = x, test = x, cl = y, k = k)

  # Calculate accuracy
  accuracy <- mean(predictions == y)

  return(accuracy)
}

# Fit KNN models and store accuracies for various k values
k_values <- c(3, 5, 10, 15, 20, 25)
binary_accuracies <- sapply(k_values, function(k) fit_knn(binary_data, k))
trinary_accuracies <- sapply(k_values, function(k) fit_knn(trinary_data, k))

# Create data frame for plotting
accuracy_data <- data.frame(
  k = rep(k_values, 2),
  accuracy = c(binary_accuracies, trinary_accuracies),
  data_type = rep(c("Binary", "Trinary"), each = length(k_values))
```

```
)

# Plot accuracy vs k using ggplot
accuracy_plot <- ggplot(accuracy_data, aes(x = k, y = accuracy, color = data_type, group = data_type)) +
  geom_line() +
  geom_point() +
  scale_x_continuous(breaks = k_values) +
  ylim(0, 1) +
  labs(x = "k", y = "Accuracy", color = "Data Type") +
  ggtitle("KNN Accuracy vs k") +
  theme_minimal()

print(accuracy_plot)
```



Analysis:

Looking back at the scatter plots, a linear classifier might not work well for these datasets because the decision boundaries are likely non-linear. This is why we use KNN, which can capture more complex boundaries.

How does the accuracy of your logistic regression classifier from last week compare?

For the binary classifier dataset, the KNN classifier outperformed the logistic regression model at all tested values of k, achieving a maximum accuracy of 74.1% with k=25 compared to 58.34% for logistic regression.

For the trinary classifier dataset, KNN also performed reasonably well with an accuracy of up to 63.2% with k=25.

Why is the accuracy different between these two methods?

Model Flexibility:

Logistic Regression: Assumes a linear relationship between the features and the outcome. This assumption can limit its performance if the true relationship is non-linear, which seems to be the case with the provided datasets. KNN Classifier: A non-parametric method that makes no assumptions about the form of the decision boundary. This allows KNN to capture more complex, non-linear relationships within the data. Overfitting and Underfitting:

Logistic Regression: May underfit if the true decision boundary is non-linear, leading to lower accuracy. KNN Classifier: The choice of k influences the model's flexibility. A smaller k can capture more local variations (risk of overfitting), while a larger k smooths out noise but might miss local patterns. In this case, higher k values provided better accuracy, indicating a need to capture broader patterns.

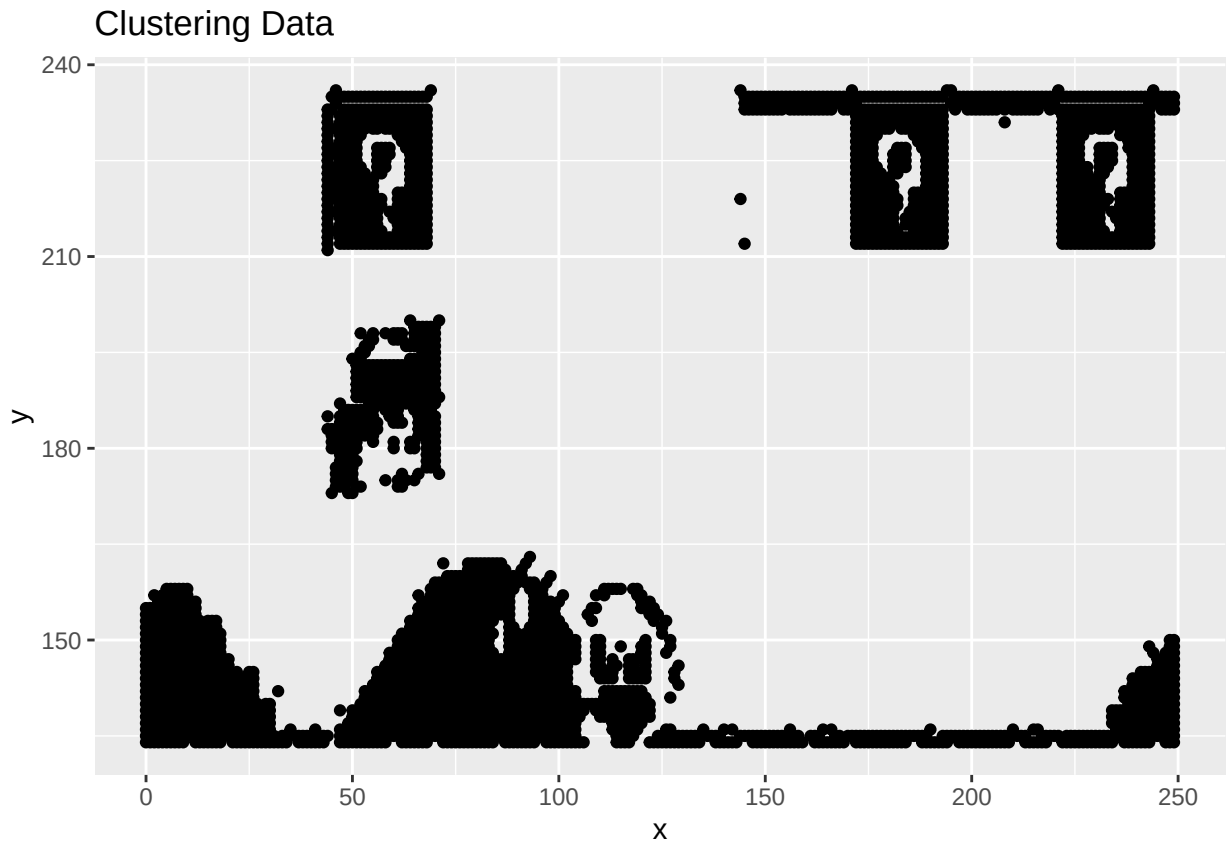
Conclusion

The KNN classifier demonstrated higher accuracy compared to the logistic regression model for the provided datasets, particularly the binary classifier dataset. This improvement is primarily due to KNN's ability to model complex, non-linear relationships without making strict assumptions about the data distribution. Logistic regression, while valuable for its simplicity and interpretability, struggled with the non-linear nature of the decision boundaries in these datasets.

By choosing the appropriate value of k, KNN was able to balance capturing local patterns and avoiding overfitting, leading to its superior performance. This comparison highlights the importance of selecting the right model based on the underlying data characteristics and the nature of the problem.

Exercise 2

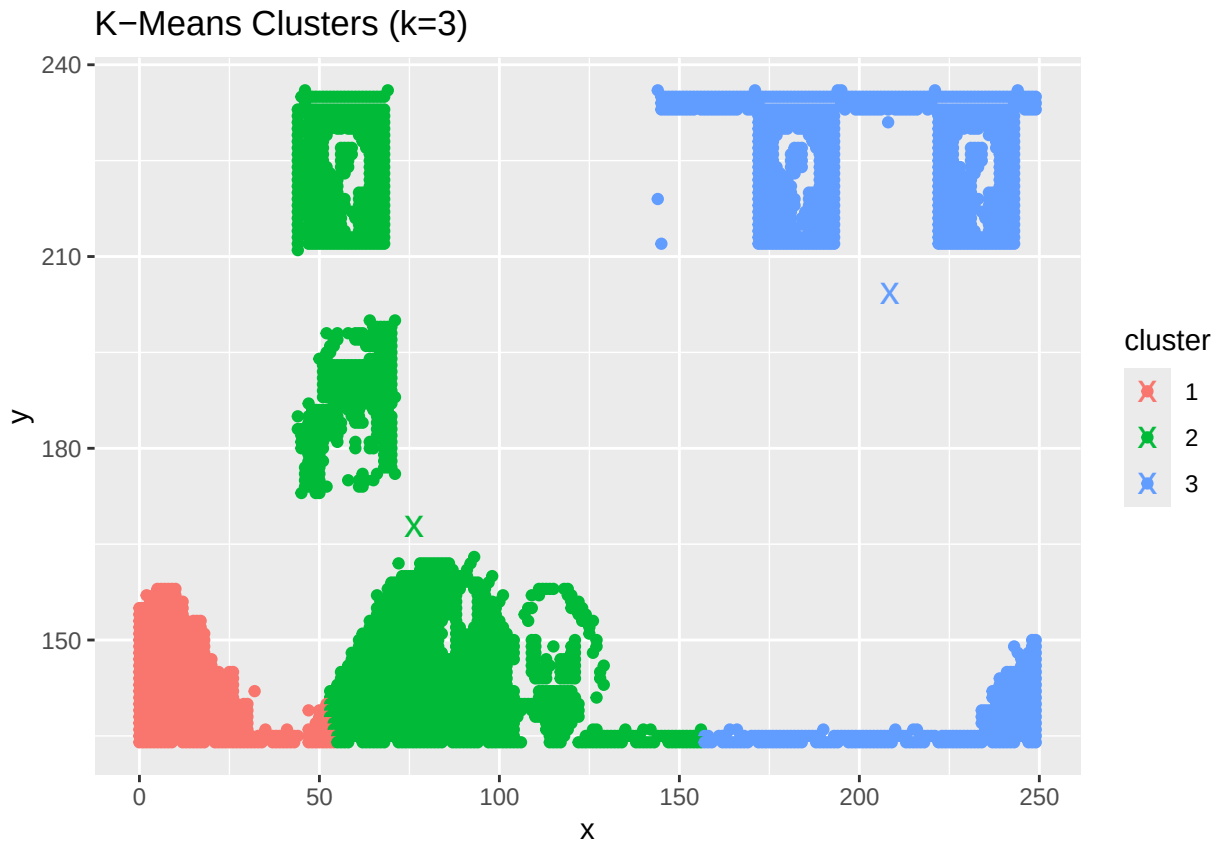
```r
# Read in clustering dataset
cluster_data <- read.csv("clustering-data.csv")

# Scatter plot of data
ggplot(cluster_data, aes(x=x, y=y)) +
  geom_point() +
  ggtitle("Clustering Data")
```



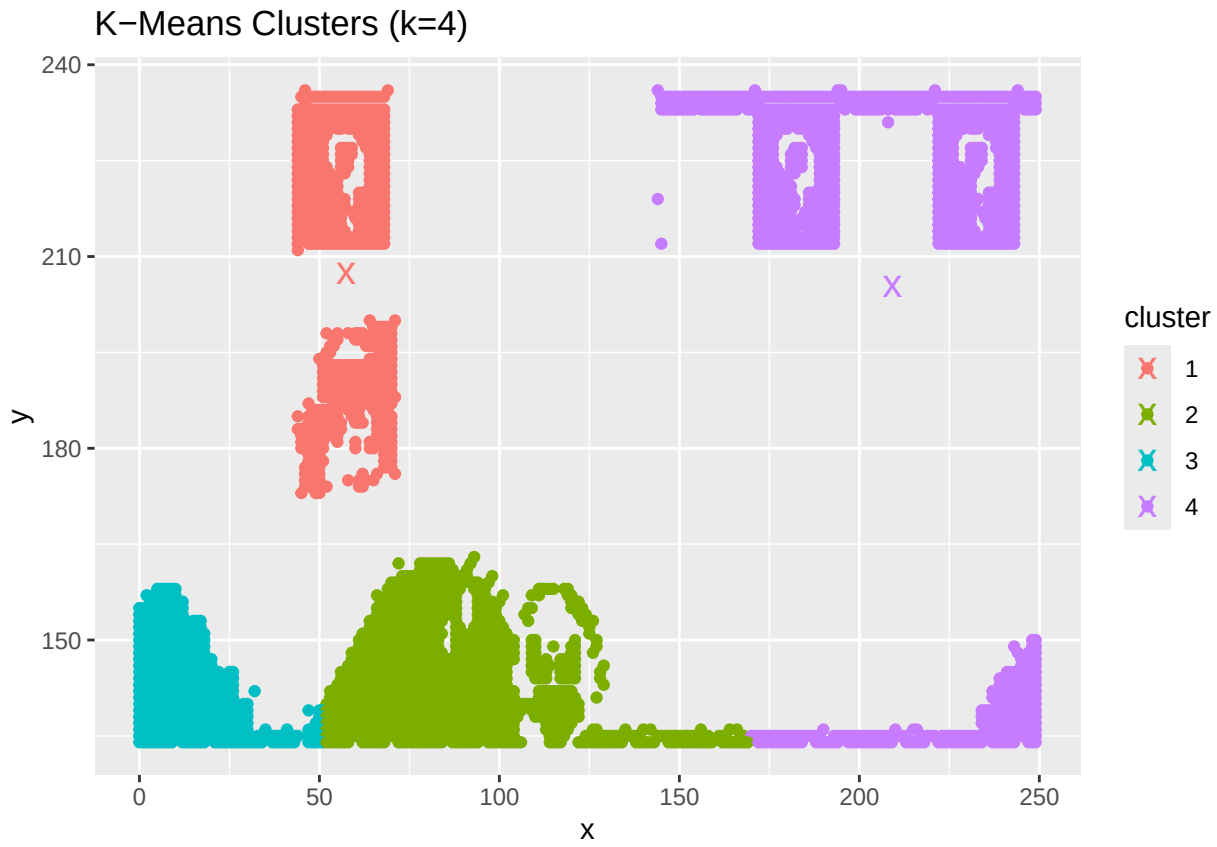

```r
# Function to plot clusters
plot_clusters <- function(data, centers, assignments) {

  # Create a data frame with cluster assignments
  data_with_clusters <- data.frame(x = data[,1], y = data[,2], cluster = as.factor(assignments))

  # Create a data frame with cluster centers
  centers_df <- data.frame(x = centers[,1], y = centers[,2], cluster = as.factor(1:nrow(centers)))

  # Create scatter plot colored by cluster
  ggplot(data_with_clusters, aes(x = x, y = y, color = cluster)) +
    geom_point() +
    geom_point(data = centers_df, size = 5, shape = "x") +
    ggtitle(paste("K-Means Clusters (k=", nrow(centers), ")", sep=""))
}

# Fit k-means models and plot clusters
for (k in 2:12) {

  # Fit k-means model
  km <- kmeans(cluster_data[,1:2], centers=k)

  # Plot clusters
  print(plot_clusters(cluster_data, km$centers, km$cluster))

  # Calculate average distance to cluster centers
```

```
  avg_dist <- mean(sapply(1:nrow(cluster_data), function(i)
    sqrt(sum((cluster_data[i,1:2] - km$centers[km$cluster[i],])^2))))

  cat("k=", k, "  Average Distance=", avg_dist, "\n")
}
```

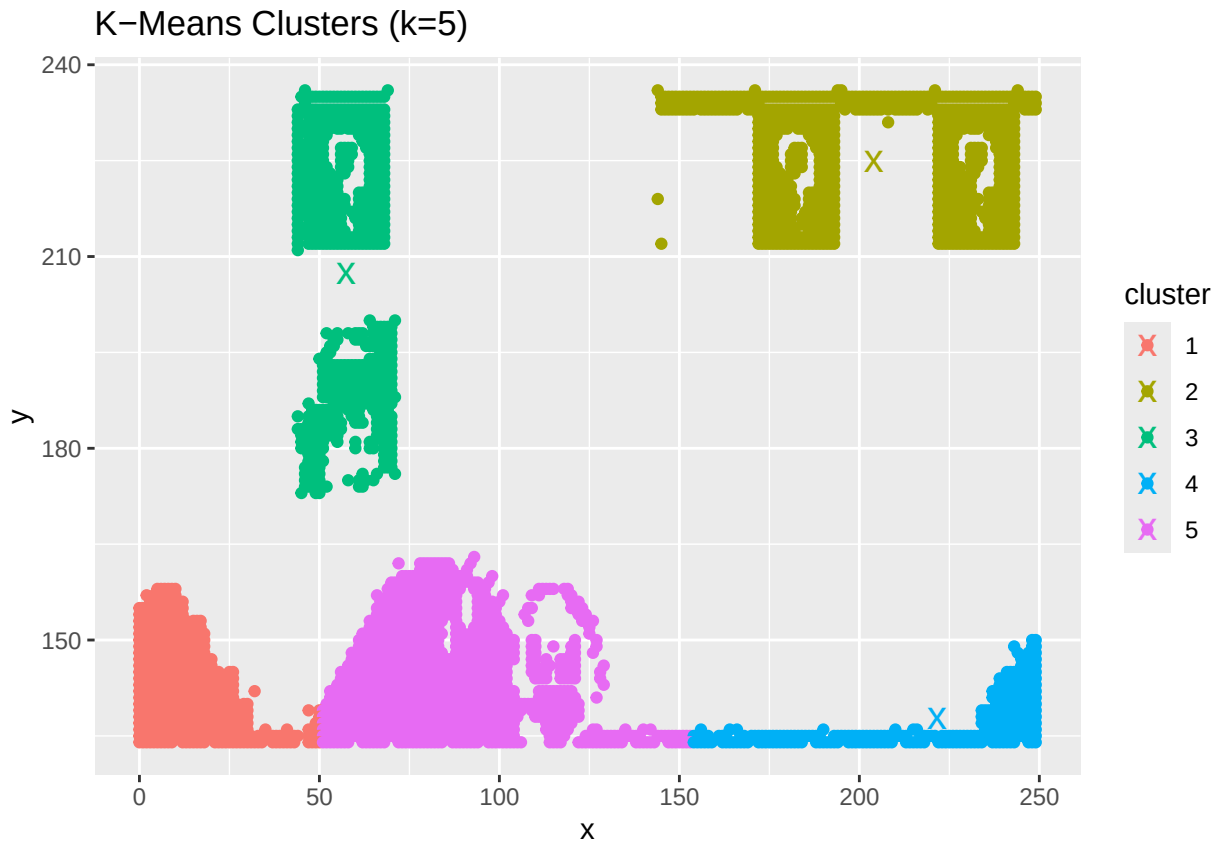

```
## k= 2   Average Distance= 42.07717
```



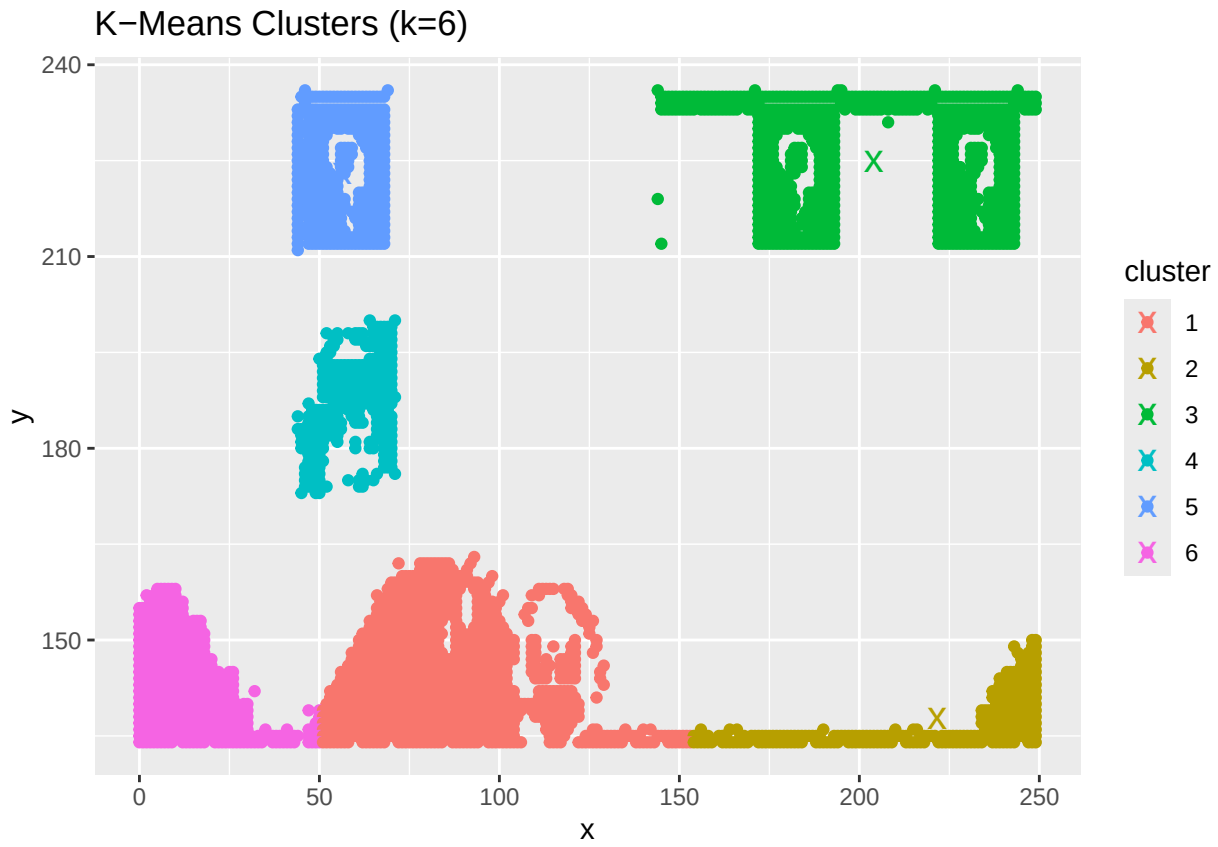

```
## k= 3    Average Distance= 34.92241
```



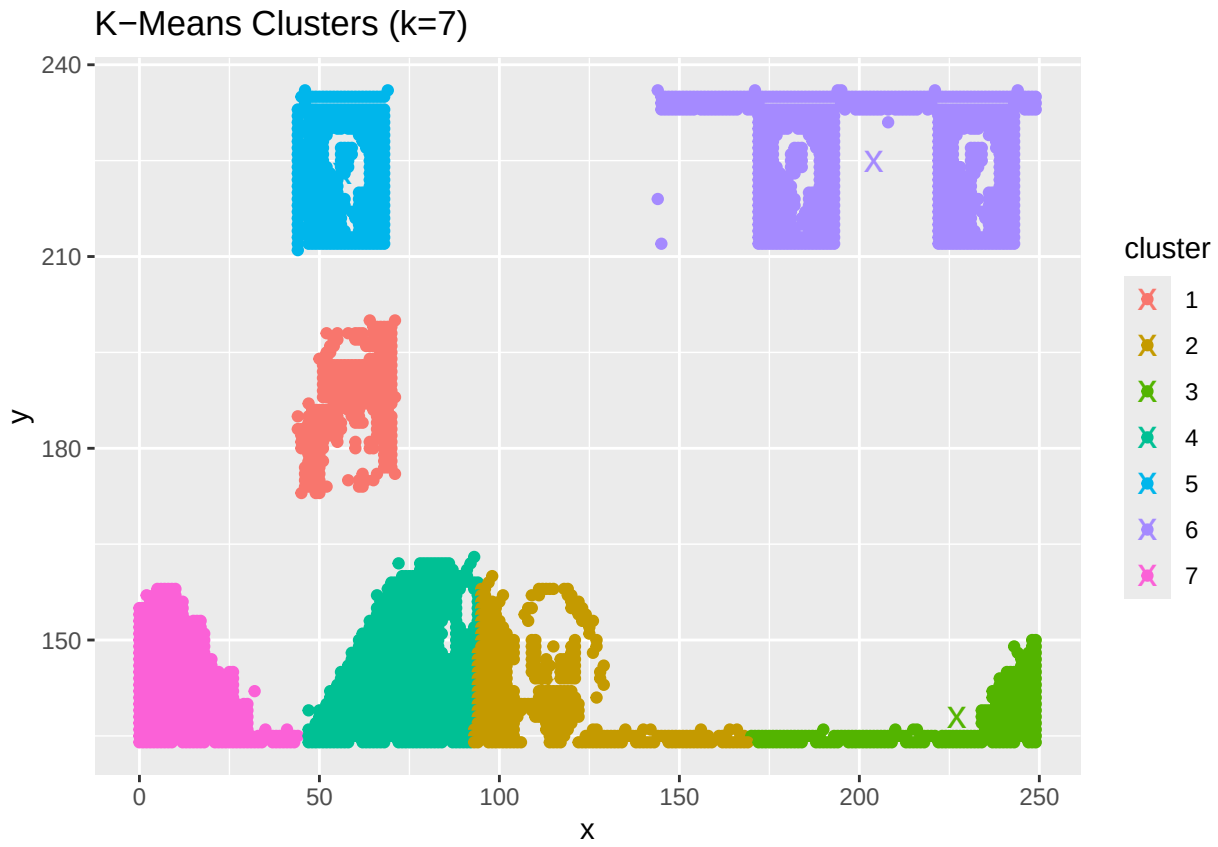

```
## k= 4    Average Distance= 25.78229
```



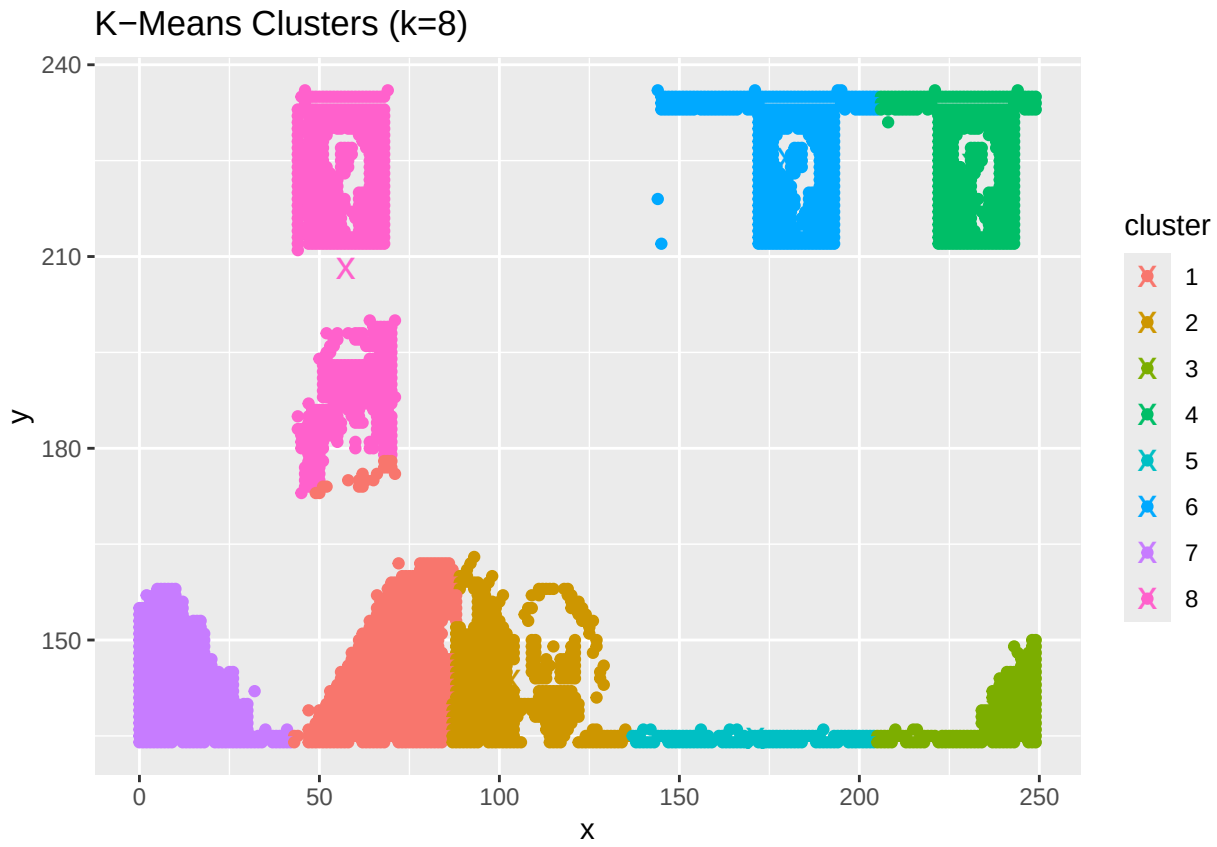

```
## k= 5    Average Distance= 20.33178
```



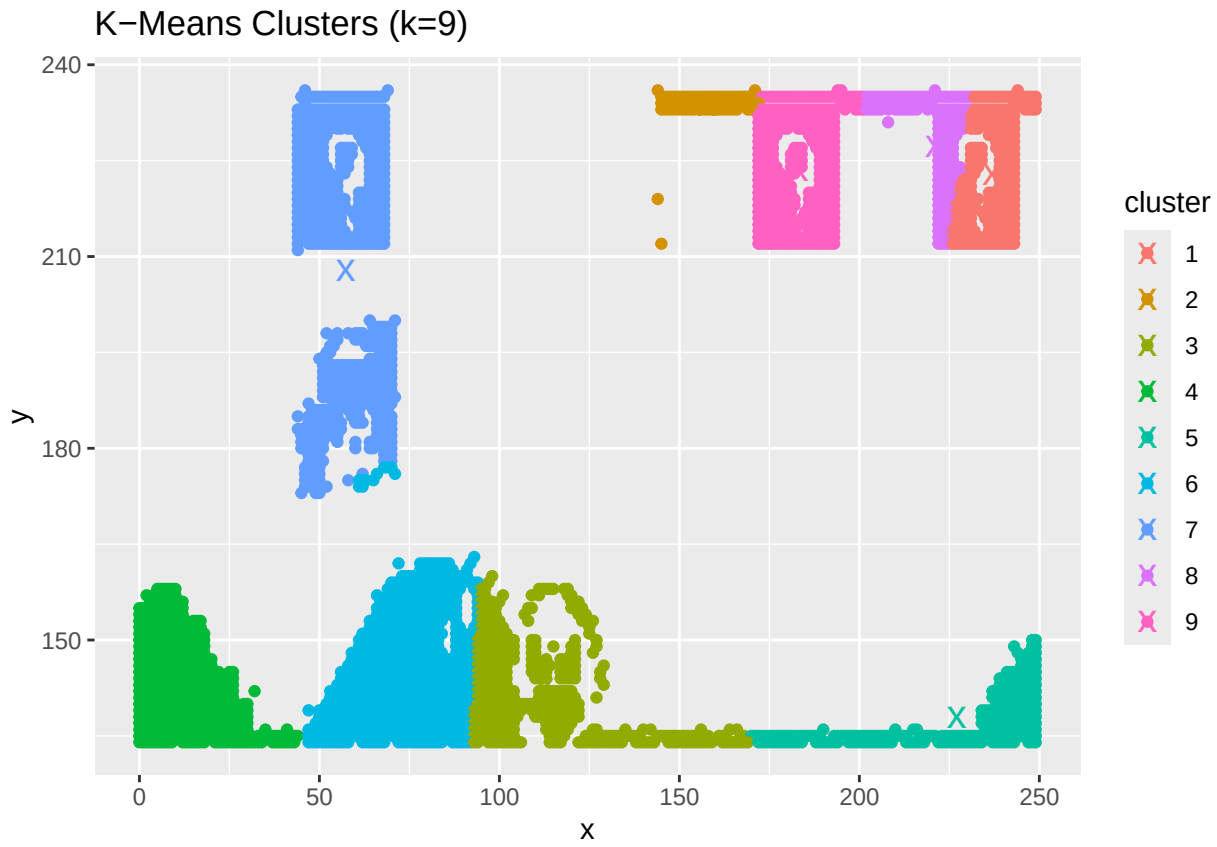

```
## k= 6    Average Distance= 18.42053
```



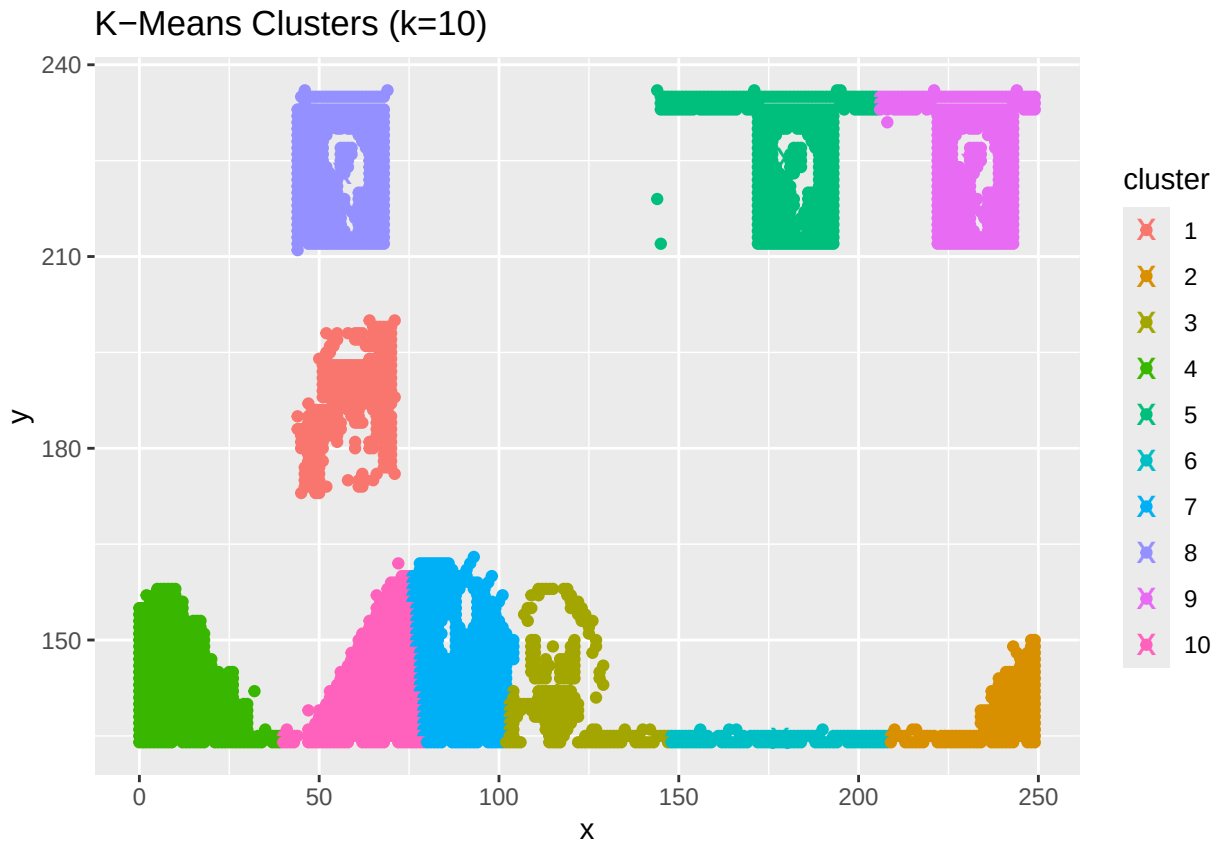

```
## k= 7   Average Distance= 16.21206
```



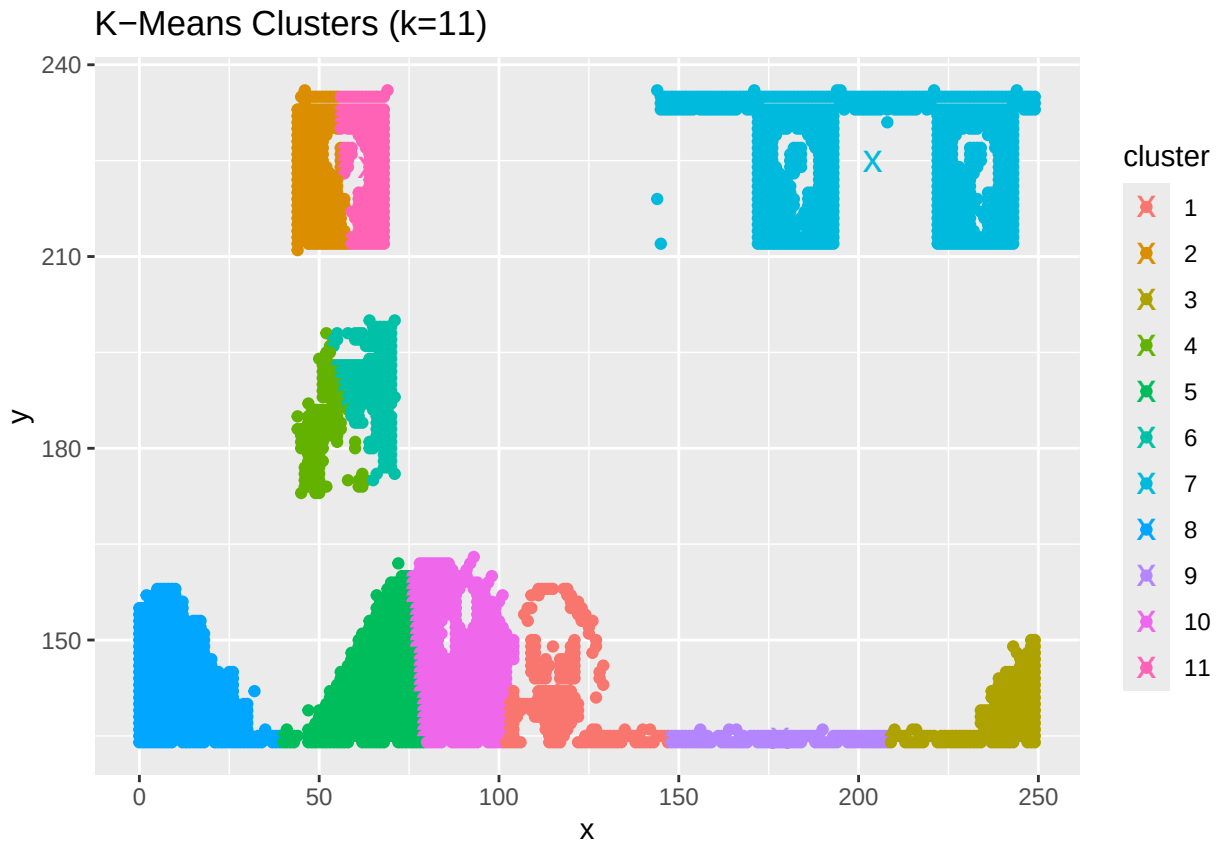

```
## k= 8   Average Distance= 13.18425
```



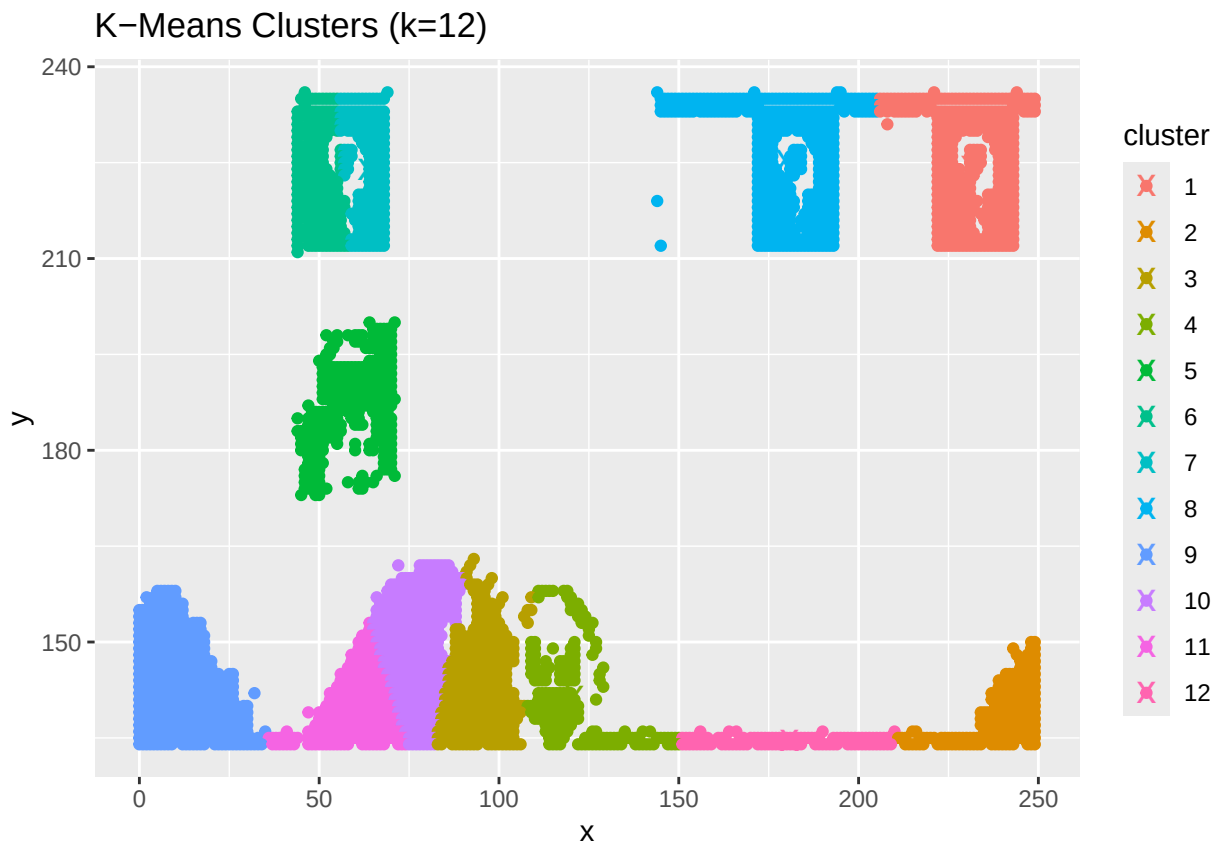

```
## k= 9    Average Distance= 13.71097
```




```
## k= 10   Average Distance= 10.47425
```



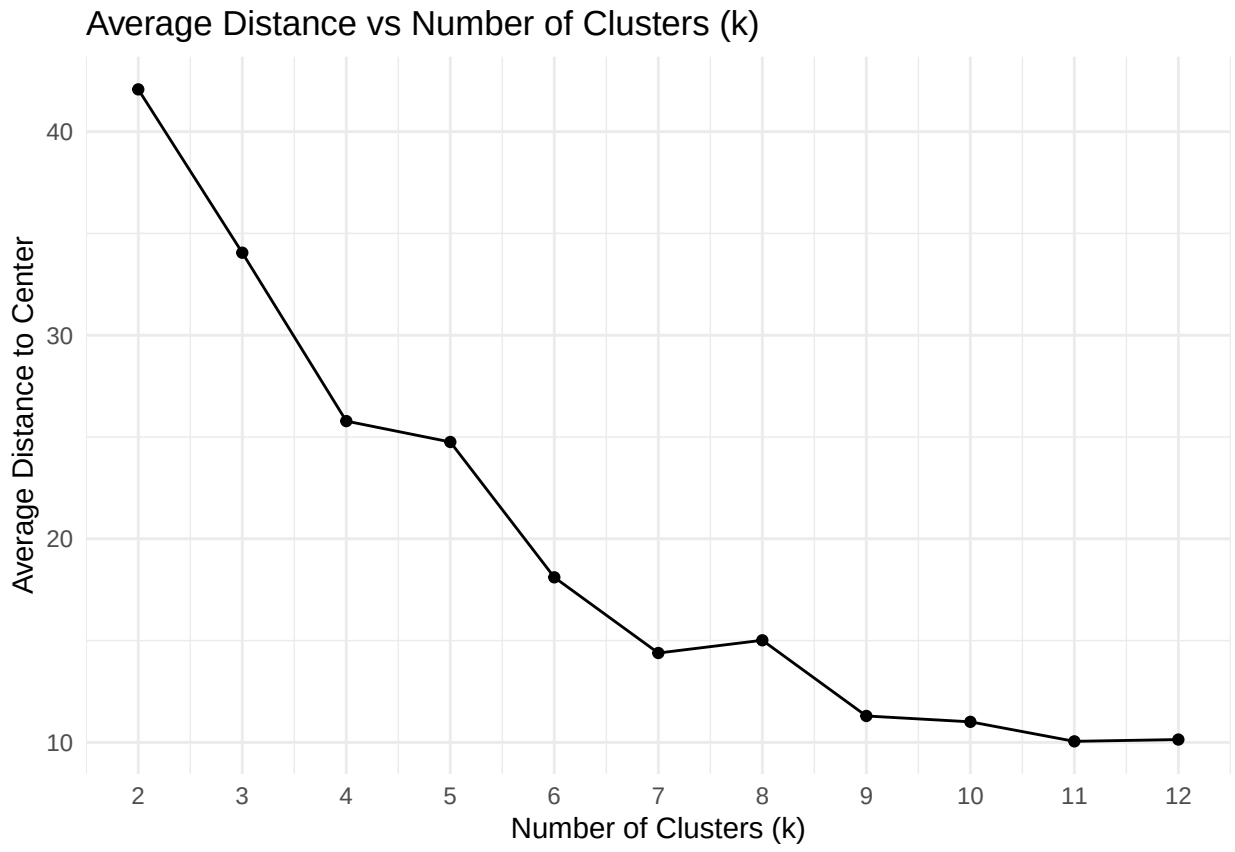

```
## k= 11   Average Distance= 13.65923
```




```
## k= 12   Average Distance= 9.633362
```

```r
# Function to fit k-means model and calculate average distance to cluster centers
fit_kmeans <- function(data, k) {

  # Fit k-means model
  km <- kmeans(data[,1:2], centers = k)

  # Calculate average distance to cluster centers
  avg_dist <- mean(sapply(1:nrow(data), function(i)
    sqrt(sum((data[i,1:2] - km$centers[km$cluster[i],])^2))))

  return(avg_dist)
}

# Fit k-means models and calculate average distances for k=2 to 12
k_values <- 2:12
avg_dists <- sapply(k_values, function(k) fit_kmeans(cluster_data, k))

# Create data frame for plotting
elbow_data <- data.frame(k = k_values, avg_dist = avg_dists)

# Plot average distance vs k using ggplot
elbow_plot <- ggplot(elbow_data, aes(x = k, y = avg_dist)) +
  geom_line() +
```

```
  geom_point() +
  scale_x_continuous(breaks = k_values) +
  labs(x = "Number of Clusters (k)", y = "Average Distance to Center") +
  ggtitle("Average Distance vs Number of Clusters (k)") +
  theme_minimal()

print(elbow_plot)
```



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?

It was observed that the average distance to cluster centers decreases with increasing k, but the rate of decrease slows down significantly around k=4 to k=6, indicating the elbow point for optimal clustering .