Machine Learning Exercise 11.2

Maxim Bilenkin

2025-02-19

1. e.
   f. Plot the data from each dataset using a scatter plot.

Loading libraries and datasets.

```
# Loading the necessary libraries and both datasets.
library(readr)
library(class)
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```
# Setting the working directory
setwd("C:/Users/maxim/OneDrive/Desktop/BU/DSC 520")

# Loading datasets
binary_classifier_data <- read_csv("binary-classifier-data.csv",
                                   show_col_types = FALSE)
trinary_classifier_data <- read_csv("trinary-classifier-data.csv",
                                    show_col_types = FALSE)

# Displaying the first few rows of each dataset
head(binary_classifier_data)
```

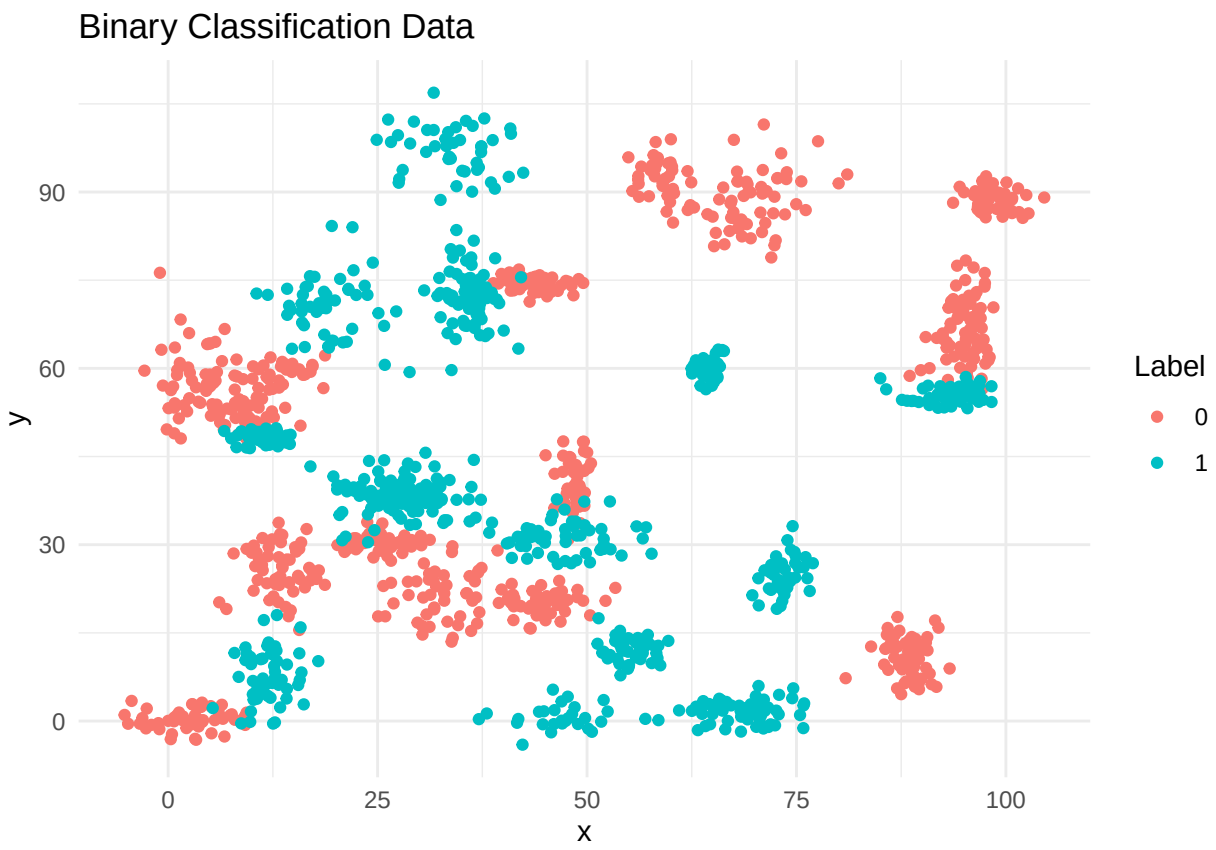
```
## # A tibble: 6 x 3
##   label     x     y
##   <dbl> <dbl> <dbl>
## 1     0  70.9  83.2
## 2     0  75.0  87.9
## 3     0  73.8  92.2
## 4     0  66.4  81.1
## 5     0  69.1  84.5
## 6     0  72.2  86.4
```

```
head(trinary_classifier_data)
```

```
## # A tibble: 6 x 3
##   label     x     y
##   <dbl> <dbl> <dbl>
## 1     0  30.1  39.6
## 2     0  31.3  51.8
## 3     0  34.1  49.3
## 4     0  32.6  41.2
## 5     0  34.7  45.5
## 6     0  33.8  44.2
```

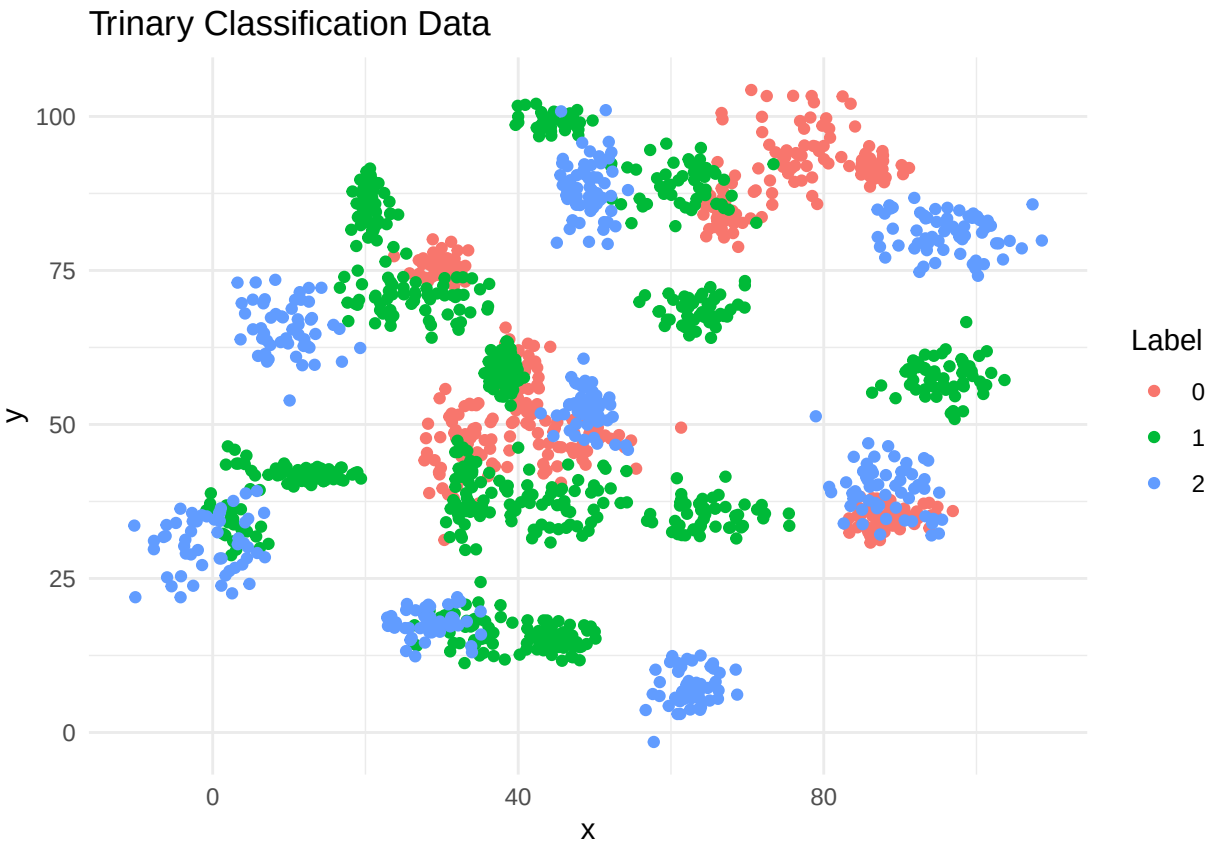
Binary classification dataset.

```
# Plotting the binary classification dataset using a scatter plot.
ggplot(binary_classifier_data, aes(x = x, y = y, color = as.factor(label))) +
  geom_point() +
  labs(title = "Binary Classification Data", color = "Label") +
  theme_minimal()
```



Trinary classification datasets.

```
# Plotting the trinary classification dataset using a scatter plot.
ggplot(trinary_classifier_data, aes(x = x, y = y, color = as.factor(label))) +
  geom_point() +
  labs(title = "Trinary Classification Data", color = "Label") +
  theme_minimal()
```



ii. Determine which points are nearest by calculating the Euclidean distance between two points.

```
# Function calculating the Euclidean distance between two points
euclidean_distance <- function(point1, point2) {
  sqrt(sum((point1 - point2)^2))
}

# Example usage
point_1 <- c(1, 2)
point_2 <- c(4, 6)
distance <- euclidean_distance(point_1, point_2)
print(paste("Euclidean Distance:", distance))
```

```
## [1] "Euclidean Distance: 5"
```

Binary Classification and model fitting.

Splitting binary dataset into training and test sets, and fitting a k-nearest neighbors model.

```
set.seed(123)
train_indices <- sample(seq_len(nrow(binary_classifier_data)), size = 0.7 *
                          nrow(binary_classifier_data))
train_binary <- binary_classifier_data[train_indices, ]
test_binary <- binary_classifier_data[-train_indices, ]

# Apply k-NN algorithm
k <- 3
predicted_binary <- knn(train_binary[, c("x", "y")], test_binary[, c("x", "y")],
                        train_binary$label, k)
```

```r
# Evaluate model performance
confusionMatrix(predicted_binary, as.factor(test_binary$label))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 225   7
##          1   6 212
##
##                Accuracy : 0.9711
##                  95% CI : (0.9511, 0.9845)
##     No Information Rate : 0.5133
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9422
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9740
##             Specificity : 0.9680
##          Pos Pred Value : 0.9698
##          Neg Pred Value : 0.9725
##              Prevalence : 0.5133
##          Detection Rate : 0.5000
##    Detection Prevalence : 0.5156
##       Balanced Accuracy : 0.9710
##
##        'Positive' Class : 0
##
```

Trinary Classification and model fitting.

Model Fitting

Splitting the trinary dataset into training and test tests, and fitting a k-nearest neighbors model.

```r
set.seed(123)
train_indices <- sample(seq_len(nrow(trinary_classifier_data)), size = 0.7 *
                          nrow(trinary_classifier_data))
train_trinary <- trinary_classifier_data[train_indices, ]
test_trinary <- trinary_classifier_data[-train_indices, ]

# Apply k-NN algorithm
predicted_trinary <- knn(train_trinary[, c("x", "y")],
                         test_trinary[, c("x", "y")], train_trinary$label, k)

# Evaluate model performance
confusionMatrix(predicted_trinary, as.factor(test_trinary$label))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1   2
##          0 113  13  11
```

```
##          1  14 174  12
##          2   7   7 120
##
## Overall Statistics
##
##                Accuracy : 0.8641
##                  95% CI : (0.8298, 0.8938)
##     No Information Rate : 0.4119
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.7929
##
##  Mcnemar's Test P-Value : 0.5238
##
## Statistics by Class:
##
##                      Class: 0 Class: 1 Class: 2
## Sensitivity            0.8433   0.8969   0.8392
## Specificity            0.9288   0.9061   0.9573
## Pos Pred Value         0.8248   0.8700   0.8955
## Neg Pred Value         0.9371   0.9262   0.9318
## Prevalence             0.2845   0.4119   0.3036
## Detection Rate         0.2399   0.3694   0.2548
## Detection Prevalence   0.2909   0.4246   0.2845
## Balanced Accuracy      0.8860   0.9015   0.8982
```

Accuracy

Calculating Accuracy for Binary Classification

```
# Calculate accuracy for binary classification
conf_matrix_binary <- confusionMatrix(predicted_binary,
                                      as.factor(test_binary$label))
accuracy_binary <- conf_matrix_binary$overall['Accuracy']
print(paste("Accuracy for Binary Classification:", accuracy_binary))
```

```
## [1] "Accuracy for Binary Classification: 0.971111111111111"
```

Calculating Accuracy for Trinary Classification

```
# Calculate accuracy for trinary classification
conf_matrix_trinary <- confusionMatrix(predicted_trinary,
                                       as.factor(test_trinary$label))
accuracy_trinary <- conf_matrix_trinary$overall['Accuracy']
print(paste("Accuracy for Trinary Classification:", accuracy_trinary))
```

```
## [1] "Accuracy for Trinary Classification: 0.86411889596603"
```

ii. 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.

Defining the values for k

```
k_values <- c(3, 5, 10, 15, 20, 25)
```

Computing the accuracy for binary classification

```r
# Splitting binary dataset into training and test sets
set.seed(123)
train_indices <- sample(seq_len(nrow(binary_classifier_data)),
                        size = 0.7 * nrow(binary_classifier_data))
train_binary <- binary_classifier_data[train_indices, ]
test_binary <- binary_classifier_data[-train_indices, ]

# Calculate accuracies for different values of k
binary_accuracies <- numeric(length(k_values))

for (i in seq_along(k_values)) {
  k <- k_values[i]
  predicted_binary <- knn(train_binary[, c("x", "y")],
                          test_binary[, c("x", "y")], train_binary$label, k)
  conf_matrix_binary <- confusionMatrix(predicted_binary,
                                        as.factor(test_binary$label))
  binary_accuracies[i] <- conf_matrix_binary$overall['Accuracy']
}

binary_accuracies
```

```
## [1] 0.9711111 0.9711111 0.9800000 0.9733333 0.9822222 0.9822222
```

Computing the accuracy for trinary classification

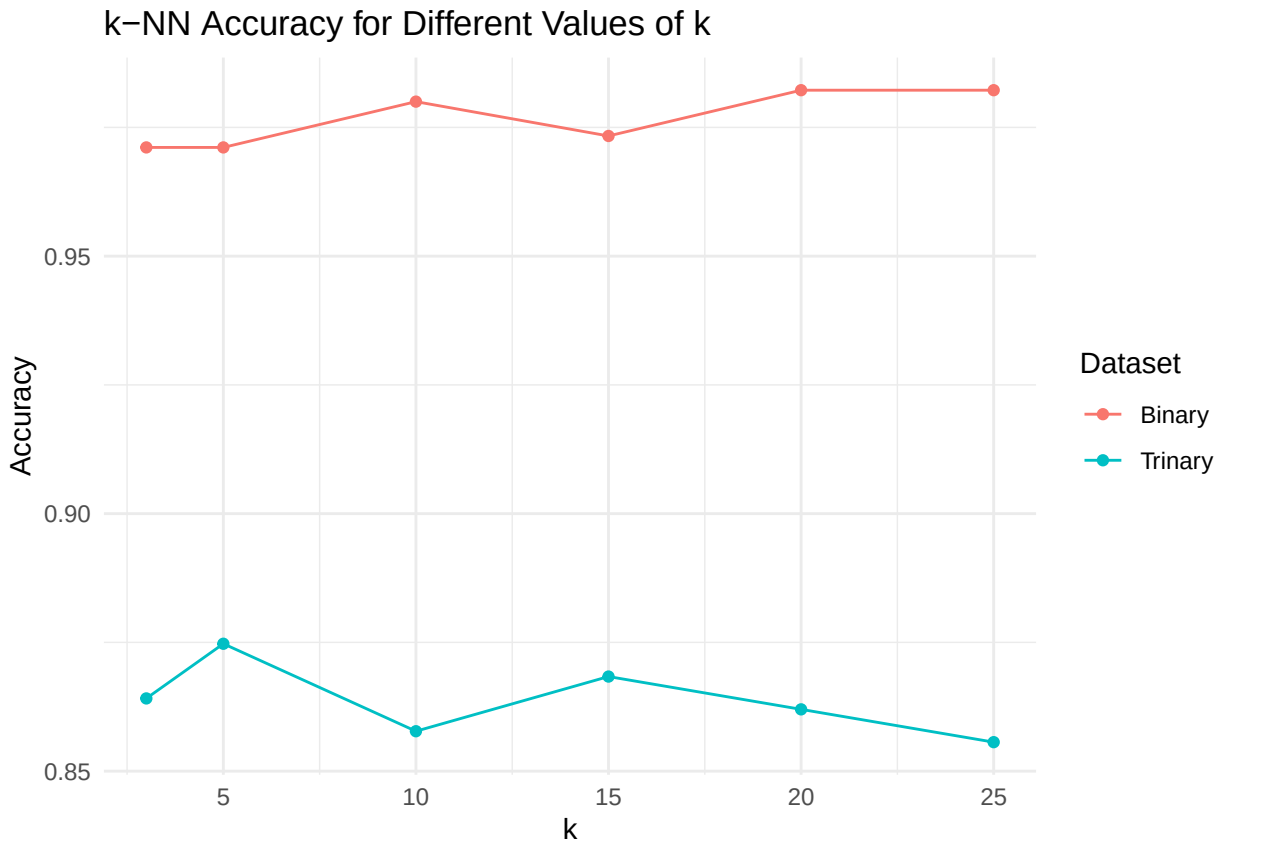
```r
# Splitting trinary dataset into training and test sets
set.seed(123)
train_indices <- sample(seq_len(nrow(trinary_classifier_data)),
                        size = 0.7 * nrow(trinary_classifier_data))
train_trinary <- trinary_classifier_data[train_indices, ]
test_trinary <- trinary_classifier_data[-train_indices, ]

# Calculate accuracies for different values of k
trinary_accuracies <- numeric(length(k_values))

for (i in seq_along(k_values)) {
  k <- k_values[i]
  predicted_trinary <- knn(train_trinary[, c("x", "y")],
                           test_trinary[, c("x", "y")], train_trinary$label, k)
  conf_matrix_trinary <- confusionMatrix(predicted_trinary,
                                         as.factor(test_trinary$label))
  trinary_accuracies[i] <- conf_matrix_trinary$overall['Accuracy']
}

trinary_accuracies
```

```
## [1] 0.8641189 0.8747346 0.8577495 0.8683652 0.8619958 0.8556263
```

Plotting the results for both classifications.

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

```r
# Plot the results
ggplot(accuracy_data, aes(x = k, y = accuracy, color = dataset,
                          group = dataset)) +
  geom_line() +
  geom_point() +
  labs(title = "k-NN Accuracy for Different Values of k",
       x = "k",
       y = "Accuracy",
       color = "Dataset") +
  theme_minimal()
```



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

Answer:

Looking at the Binary Classification Dataset, we can clearly see that both 0 and 1 labels are almost distinctly separated from each other in different clusters. The points on the graph for each label are almost clearly separated. This suggests that a linear classifier would likely work well on this dataset.

Looking at the Trinary Classification Dataset scatter plot, the three different clusters for the labels 0, 1, and 2 are shown to be less separated. Many points for different labels overlap with each other. This indicates that a linear classifier might not be a good model to use because the accuracy would probably be low.

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

Answer:

The logistic regression classifier model from last week for the Thoracic Surgery Binary Dataset had an accuracy rate of 82.98%, indicating that a linear model works well for this dataset.

However, the logistic regression model's accuracy rate for the binary classification dataset is much lower at 53.51%. This is significantly lower than the k-NN models' accuracy rates, which range from 97.11% to 98.22%. This suggests that k-NN models have a greater ability to capture non-linear relationships, leading to much better performance on this dataset.

2. Clustering

i. Plot the dataset using a scatter plot.

Loading the data from the dataset.

```
# Loading necessary libraries
library(ggplot2)

# Loading the dataset
data <- read.csv("C:/Users/maxim/OneDrive/Desktop/BU/DSC 520/clustering-data.csv")
```

Exploring the data in the dataset to get a better insight.

```
# Explore the data
str(data)
```

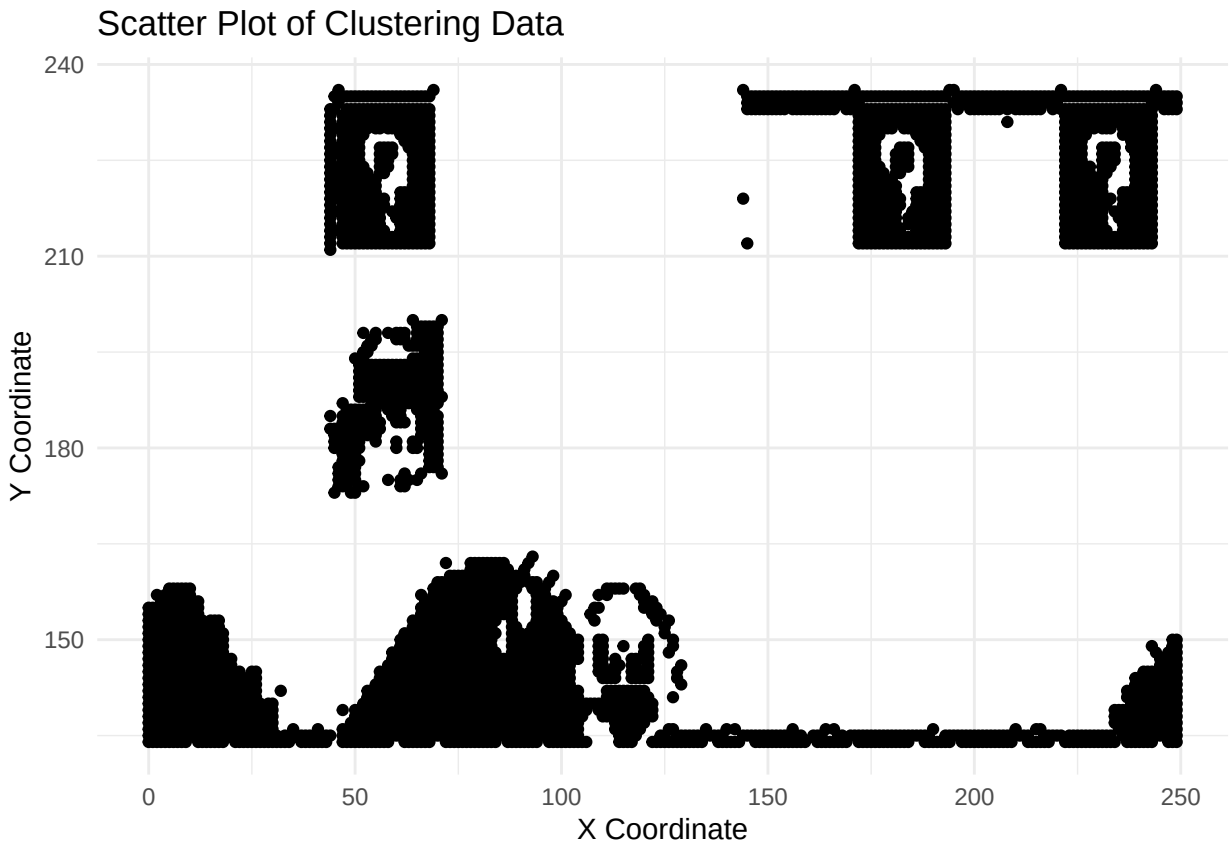
```
## 'data.frame':    4022 obs. of  2 variables:
##  $ x: int  46 69 144 171 194 195 221 244 45 47 ...
##  $ y: int  236 236 236 236 236 236 236 236 235 235 ...
```

```
summary(data)
```

```
##        x                y
##  Min.   :  0.0   Min.   :134.0
##  1st Qu.: 56.0   1st Qu.:141.0
##  Median : 82.0   Median :154.0
##  Mean   :109.6   Mean   :175.7
##  3rd Qu.:180.0   3rd Qu.:218.0
##  Max.   :249.0   Max.   :236.0
```

Plotting the data from the dataset using a scatter plot.

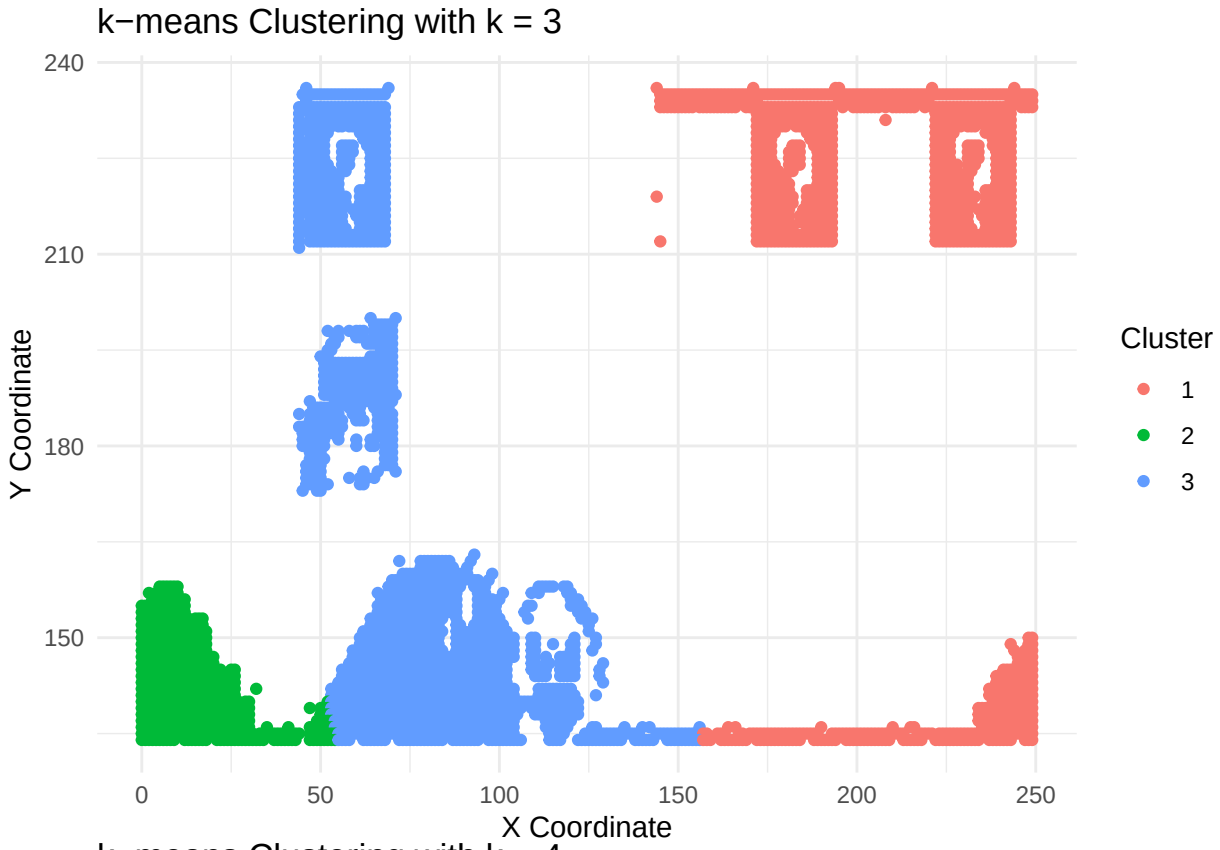
```
# Plotting the data from the dataset using a scatter plot
ggplot(data, aes(x = x, y = y)) +
  geom_point() +
  labs(title = "Scatter Plot of Clustering Data", x = "X Coordinate",
       y = "Y Coordinate") +
  theme_minimal()
```



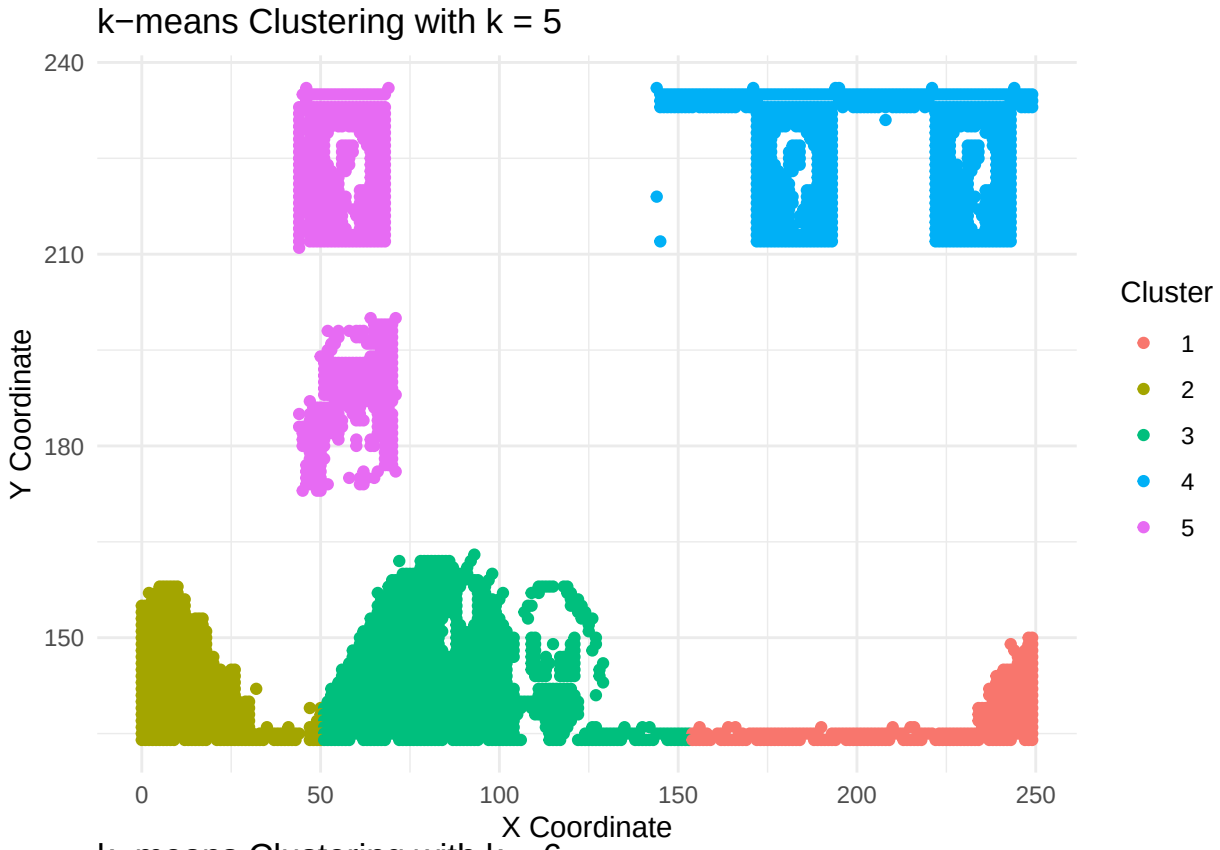
ii. Fit the dataset using the k-means algorithm from k=2 to k=12. Create a scatter plot of the resultant clusters for each value of k.

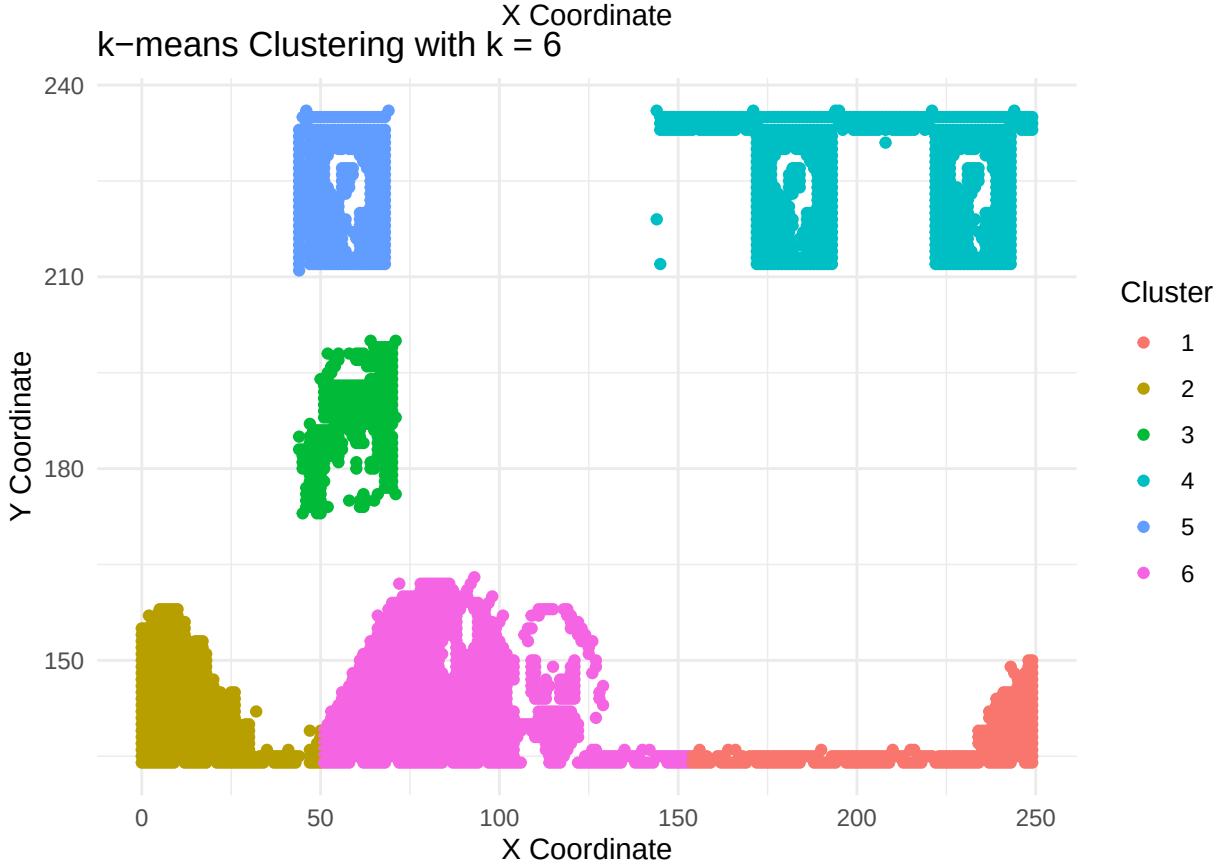
```
# Loading necessary libraries
library(ggplot2)
library(dplyr)

# Loading the data from the dataset
data <- read.csv("C:/Users/maxim/OneDrive/Desktop/BU/DSC 520/clustering-data.csv")

# Function to plot k-means clusters
plot_kmeans <- function(data, k) {
  set.seed(123)  # For reproducibility
  kmeans_result <- kmeans(data, centers = k)
  data$cluster <- as.factor(kmeans_result$cluster)

  ggplot(data, aes(x = x, y = y, color = cluster)) +
    geom_point() +
    labs(title = paste("k-means Clustering with k =", k), x = "X Coordinate",
         y = "Y Coordinate") +
    theme_minimal() +
    scale_color_discrete(name = "Cluster")
}

# Looping through k values from 2 to 12 and create plots
plots <- lapply(2:12, function(k) plot_kmeans(data, k))

# Displaying the plots
```

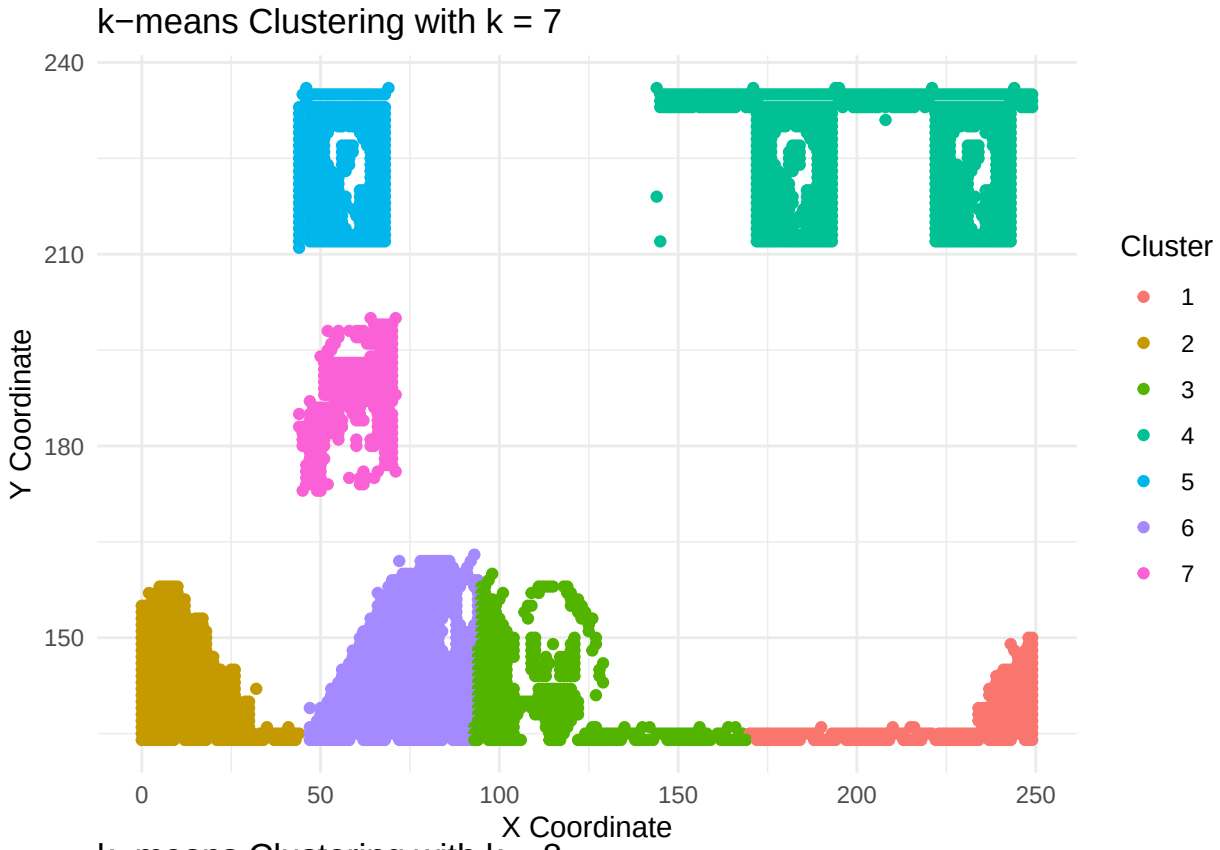
```
for (plot in plots) {
  print(plot)
}
```


k−means Clustering with k = 3
240
210
180
150
Y Coordinate
0
50
100
150
200
250
X Coordinate
Cluster
1
2
3

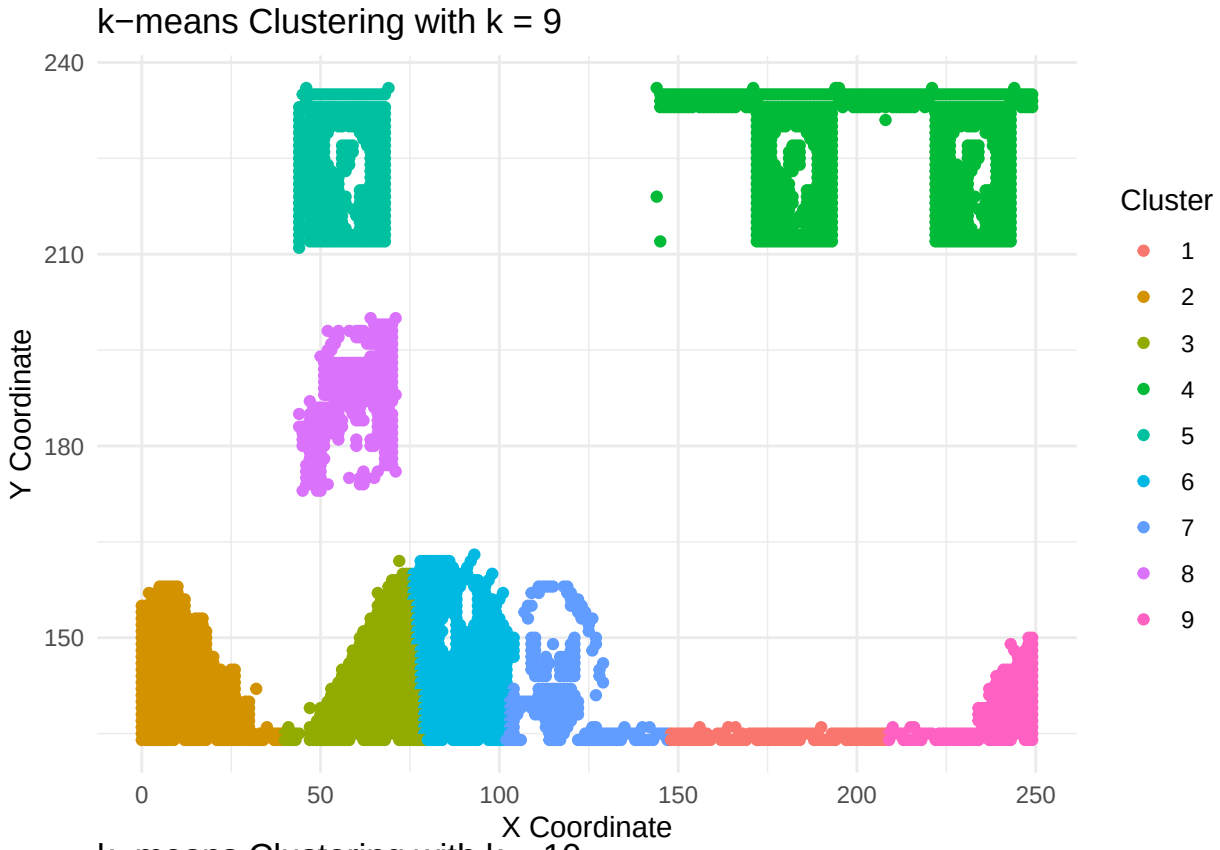

k−means Clustering with k = 4
240
210
180
150
Y Coordinate
0
50
100
150
200
250
X Coordinate
Cluster
1
2
3
4

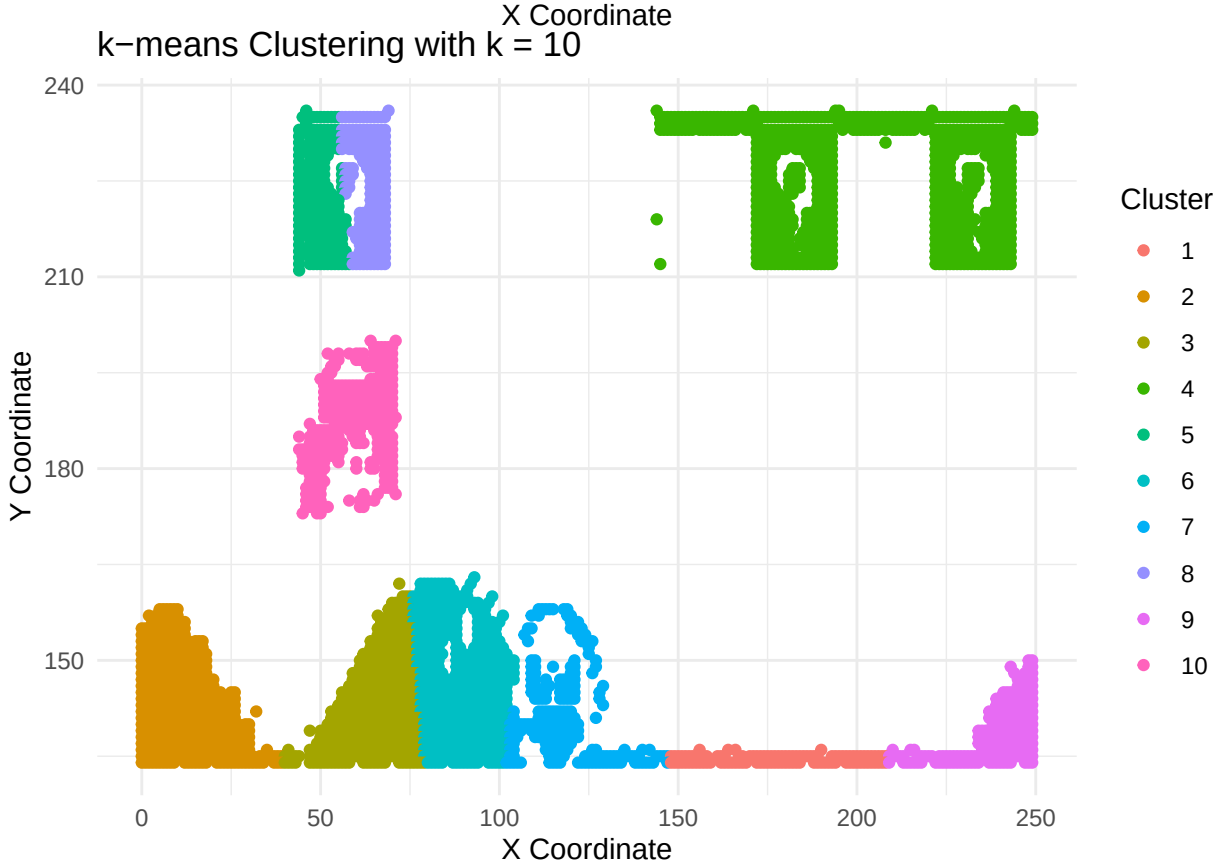

k−means Clustering with k = 5
240
210
180
150
Y Coordinate
0
50
100
150
200
250
X Coordinate
Cluster
1
2
3
4
5

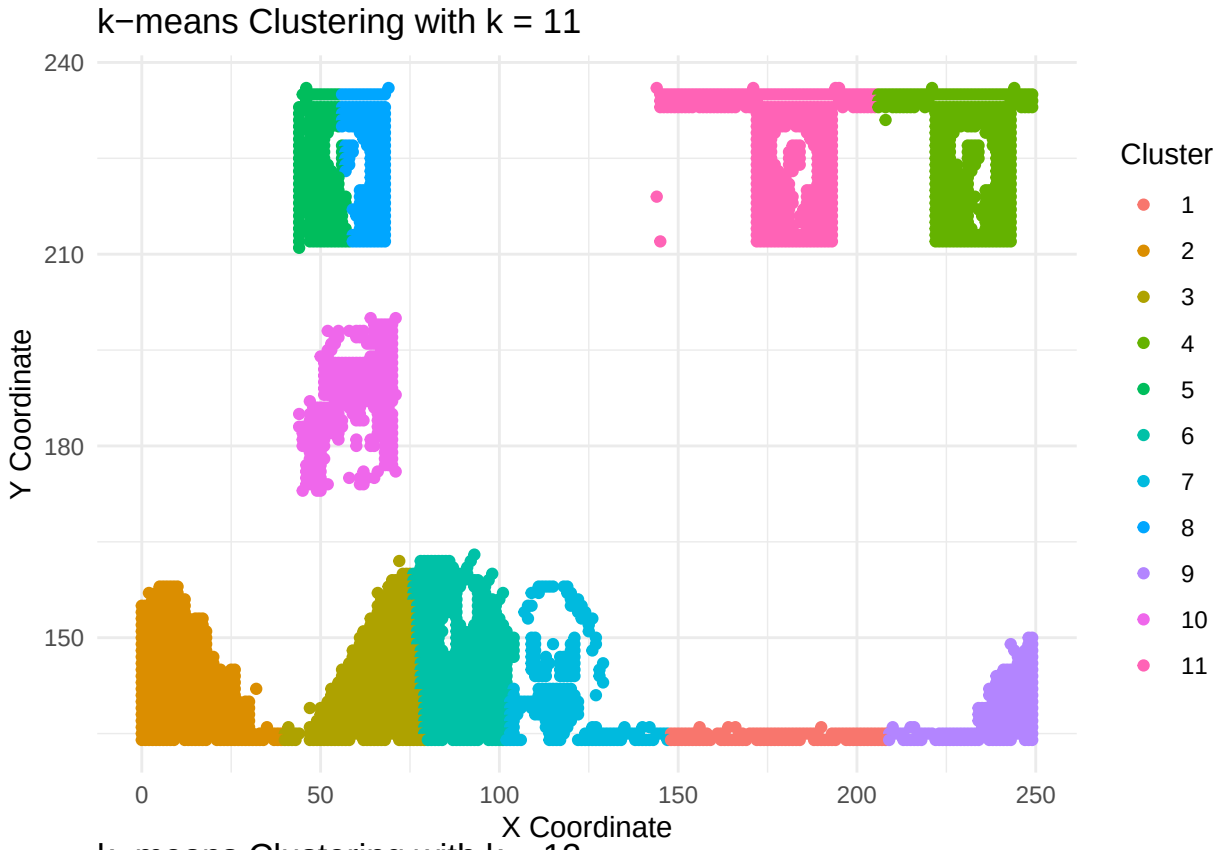

k−means Clustering with k = 6
240
210
180
150
Y Coordinate
0
50
100
150
200
250
X Coordinate
Cluster
1
2
3
4
5
6

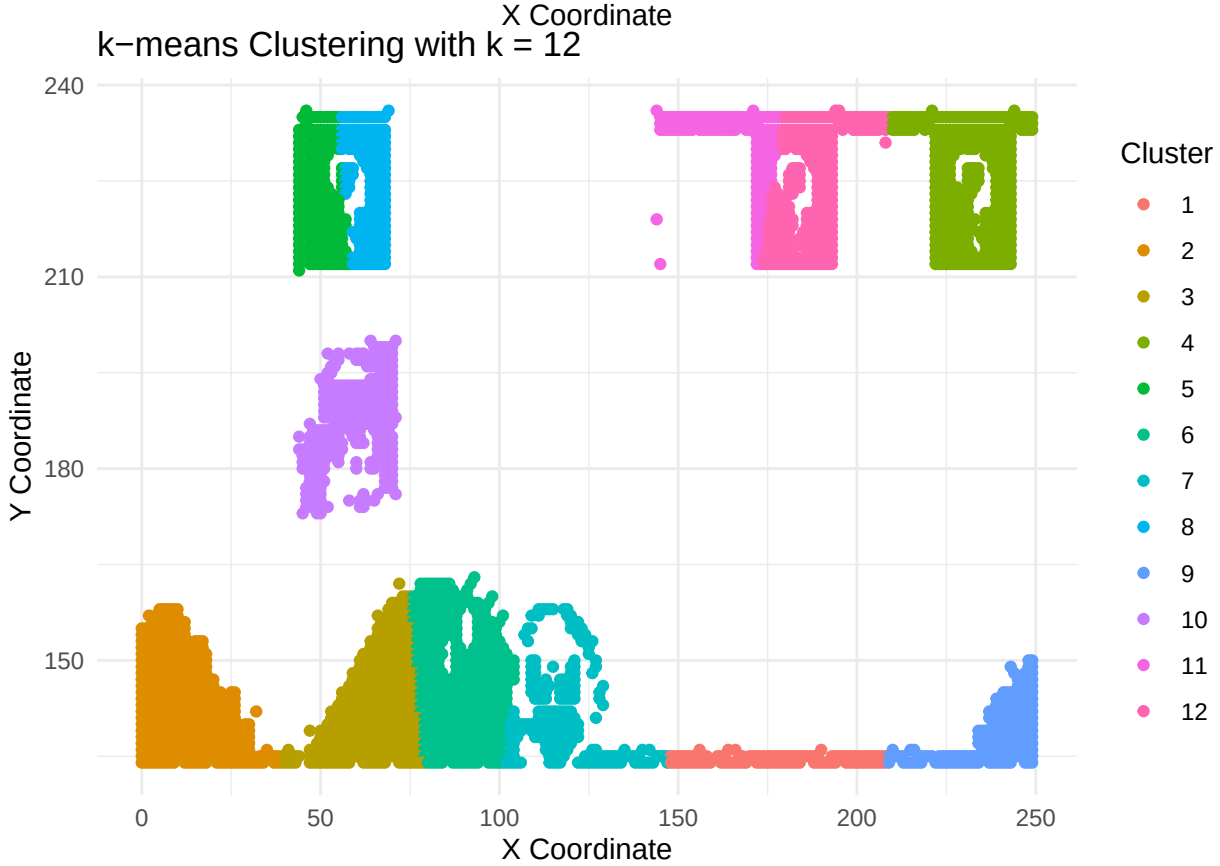

k−means Clustering with k = 7
240
210
180
150
Y Coordinate
0
50
100
150
200
250
X Coordinate
Cluster
1
2
3
4
5
6
7


k−means Clustering with k = 8
240
210
180
150
Y Coordinate
0
50
100
150
200
250
X Coordinate
Cluster
1
2
3
4
5
6
7
8


k−means Clustering with k = 9
Y Coordinate
X Coordinate
240
210
180
150
0
50
100
150
200
250
Cluster
1
2
3
4
5
6
7
8
9


k−means Clustering with k = 10
Y Coordinate
X Coordinate
240
210
180
150
0
50
100
150
200
250
Cluster
1
2
3
4
5
6
7
8
9
10





iii. 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.

```
# Computing averages of distance to the cluster centers
compute_avg_distance <- function(data, k) {
  set.seed(123)
  kmeans_result <- kmeans(data, centers = k)
  centers <- kmeans_result$centers
  clusters <- kmeans_result$cluster
  distances <- sqrt(rowSums((data - centers[clusters, ])^2))
  avg_distance <- mean(distances)
  return(avg_distance)
}

# Computing and printing average distances for k values from 2 to 12
avg_distances <- sapply(2:12, function(k) compute_avg_distance(data, k))
avg_distances_df <- data.frame(k = 2:12, avg_distance = avg_distances)

# Printing the results
avg_distances_df
```

```
##     k avg_distance
## 1   2    42.077169
## 2   3    34.922407
## 3   4    25.782290
## 4   5    20.331782
## 5   6    18.420531
## 6   7    16.212063
## 7   8    15.015130
## 8   9    14.192327
## 9  10    13.908049
## 10 11    10.189973
## 11 12     9.816107
```

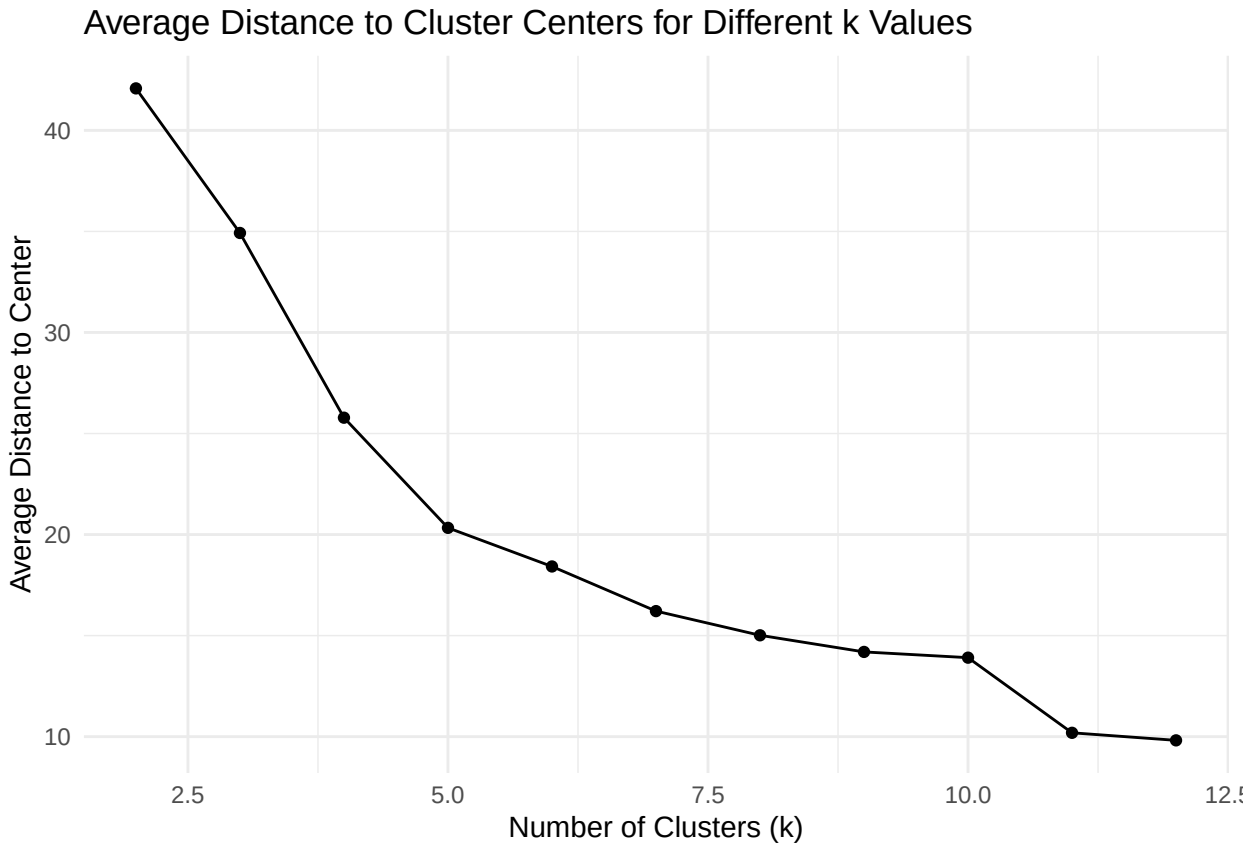
e. Calculate this average distance from the center of each cluster for each value of k and plot it as a line chart where k is the x-axis and the average distance is the y-axis.

```
# Computing average distance to cluster centers
compute_avg_distance <- function(data, k) {
  set.seed(123)  # For reproducibility
  kmeans_result <- kmeans(data, centers = k)
  centers <- kmeans_result$centers
  clusters <- kmeans_result$cluster
  distances <- sqrt(rowSums((data - centers[clusters, ])^2))
  avg_distance <- mean(distances)
  return(avg_distance)
}

# Computing and printing average distances for k values from 2 to 12
avg_distances <- sapply(2:12, function(k) compute_avg_distance(data, k))
avg_distances_df <- data.frame(k = 2:12, avg_distance = avg_distances)

# Printing the results
avg_distances_df
```

```
##     k avg_distance
## 1   2    42.077169
```

```
## 2   3    34.922407
## 3   4    25.782290
## 4   5    20.331782
## 5   6    18.420531
## 6   7    16.212063
## 7   8    15.015130
## 8   9    14.192327
## 9  10    13.908049
## 10 11    10.189973
## 11 12     9.816107
```

```r
# Plotting the average distance for each k value
ggplot(avg_distances_df, aes(x = k, y = avg_distance)) +
  geom_line() +
  geom_point() +
  labs(title = "Average Distance to Cluster Centers for Different k Values",
       x = "Number of Clusters (k)", y = "Average Distance to Center") +
  theme_minimal()
```



f. Looking at the graph you generated in the previous example, what is the elbow point for this dataset?

Answer:

Based on the generated graph in the previous example, the elbow point for this dataset appears at k=5. This is the point where the decrease in average distance significantly slows down, creating the elbow point. Additionally, we can state that the optimal number of clusters is likely k=5. Therefore, 5 clusters probably represent the optimal number for this dataset.