K-Nearest and K-Means Assignment

David Berberena

03-02-2024

K-Nearest Neighbors and K-Means Clustering Machine Learning Assignment

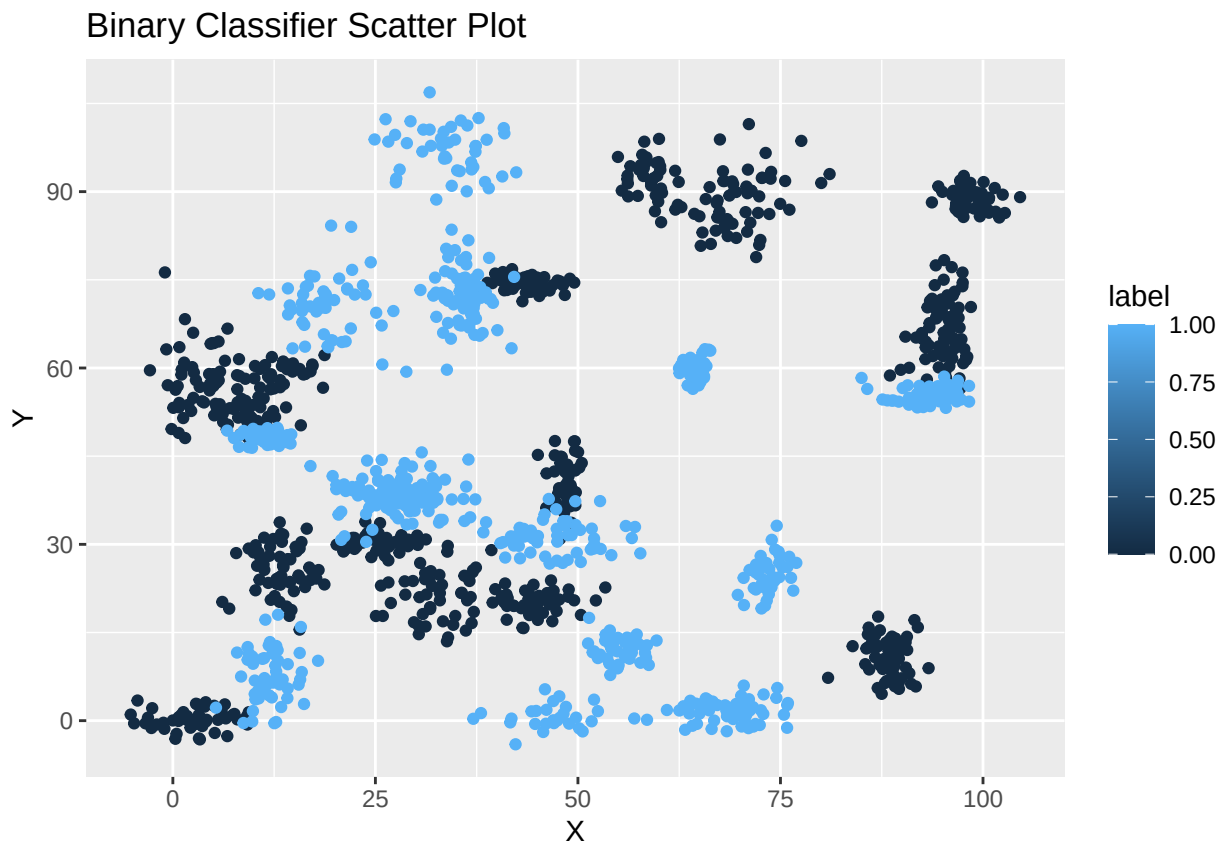
K-Nearest Neighbor Classifier Dataset Scatter Plots

Binary Classifier Dataset

```
# Assignment Start

# Upload readr library for file importing

library(readr)

# set working directory for smooth file importing

setwd("C:/Users/dbzda/Documents/School/DSC 520 Statistics for Data Science")

# Import the converted Housing CSV file to view its properties

binary <- read_csv("binary_classifier_data.csv", show_col_types = FALSE)

trinary <- read_csv("trinary_classifier_data.csv", show_col_types = FALSE)
```

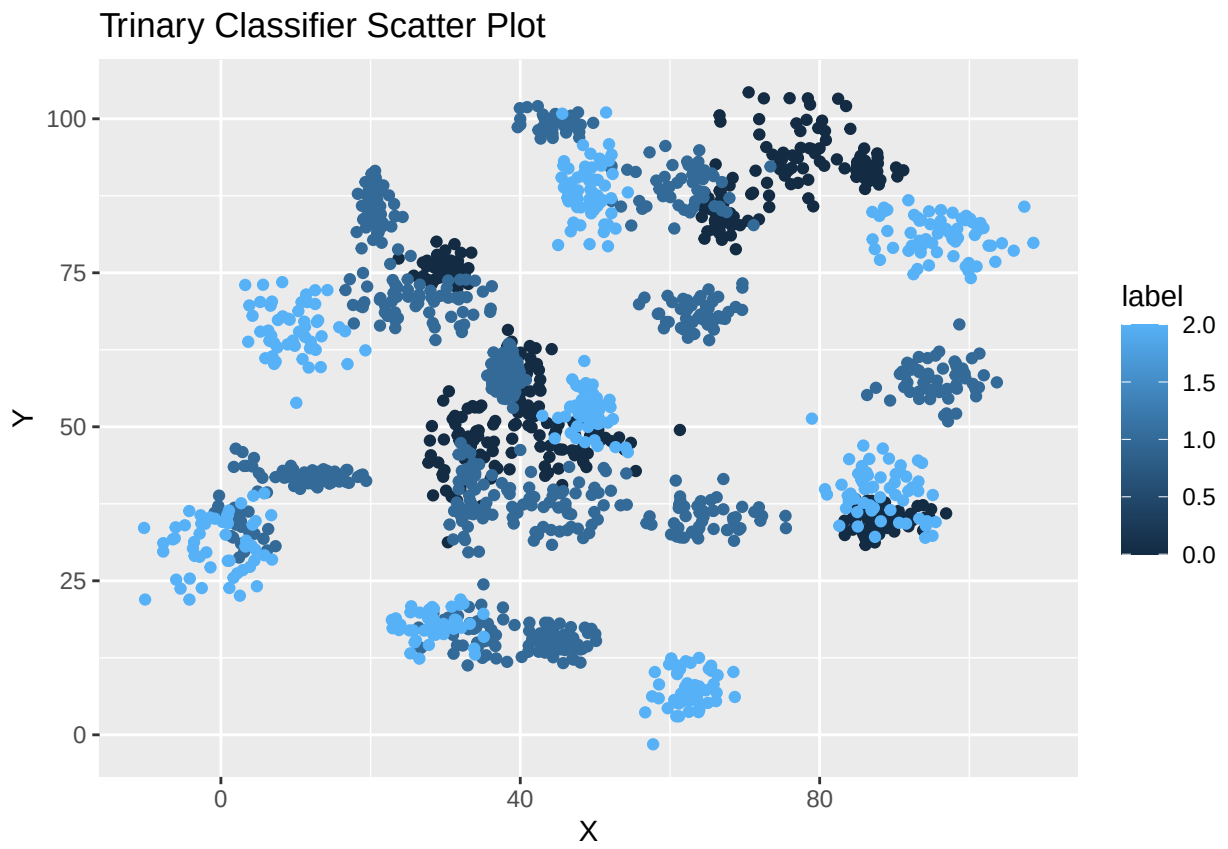
1A. Plot the data from each dataset using a scatter plot.

```
# Upload ggplot2 library to create scatter plots with increased functionality

library(ggplot2)

# Binary Dataset Scatter Plot

# Scatter plot of x variable and y variable with the label denoted as the color

ggplot(binary, aes(x = x, y = y, color = label)) +
  geom_point() +
  labs(title = "Binary Classifier Scatter Plot",
       x = "X",
       y = "Y")
```



Trinary Classifier Dataset

```
# Trinary Dataset Scatter Plot

# Scatter plot of x variable and y variable with the label denoted as the color

ggplot(trinary, aes(x = x, y = y, color = label)) +
  geom_point() +
  labs(title = "Trinary Classifier Scatter Plot",
       x = "X",
       y = "Y")
```



K-Nearest Neighbor Models

1B. 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 Classifier Dataset Models

```
# k=3 binary k-nearest neighbor model

# Load the caTools library to aid in splitting the dataset by creating a
# training dataset and a test dataset

library(caTools)

# A seed must be set so that the data being split will be split the same each
# time the code is run and subsequent values will be consistent

set.seed(123)

# Load the caret library for the creation of k-nearest neighbor models

library(caret)
```

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

```
# Split the data into a training dataset and a test dataset using the
# createDataPartition() function in the caret library, which functions similarly
# to the sample.split() function in caTools and the subset() function, with 80%
# of the dataset being the training dataset (notated as p = 0.8) and the other
# 20% being the test dataset

binary_split <- createDataPartition(binary$label, p = .8,
                                    list = FALSE,
                                    times = 1)
binary_train_data <- binary[binary_split,]
binary_test_data <- binary[-binary_split,]

# The outcome variable (label in this scenario) needs to be factored to make
# sure that the model being generated works on the same levels when being tested

binary_train_data$label <- factor(binary_train_data$label)
binary_test_data$label <- factor(binary_test_data$label)

# Create and train the k-nearest neighbor model with the train() function in the
# caret library with the method parameter set to "knn" and the tuneLength
# parameter set to whatever value of k is needed (k=3, k=5, k=10, k=15, k=20,
# and k=25)

binary_knn_model1 <- train(label ~ x + y,
                   data = binary_train_data,
                   method = "knn",
                   trControl = trainControl(method = "cv"),
                   tuneLength = 3)

# With the k=3 model built, we can now run the test data through the model to
# generate predictions for our confusion matrix needed to calculate accuracy

predictions1 <- predict(binary_knn_model1, newdata = binary_test_data)

# The confusionMatrix() function that exists within the caret package creates
# the necessary matrix and computes the accuracy automatically, which can be
# called with the $ operator

binary_confusion_matrix1 <- confusionMatrix(predictions1, binary_test_data$label)

binary_model_accuracy1 <- binary_confusion_matrix1$overall["Accuracy"]

binary_model_accuracy1
```

```
##  Accuracy
## 0.9866221
```

```
# k=5 binary k-nearest neighbor model

# To craft each additional model and compute the accuracy for each, we simply
# repeat the same steps from model generation to the accuracy output and change
```

```r
# the value of k

set.seed(123)

binary_knn_model2 <- train(label ~ x + y,
                    data = binary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 5)

predictions2 <- predict(binary_knn_model2, newdata = binary_test_data)

binary_confusion_matrix2 <- confusionMatrix(predictions2, binary_test_data$label)

binary_model_accuracy2 <- binary_confusion_matrix2$overall["Accuracy"]

binary_model_accuracy2
```

```
##  Accuracy
## 0.9832776
```

```r
# k=10 binary k-nearest neighbor model

set.seed(123)

binary_knn_model3 <- train(label ~ x + y,
                    data = binary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 10)

predictions3 <- predict(binary_knn_model3, newdata = binary_test_data)

binary_confusion_matrix3 <- confusionMatrix(predictions3, binary_test_data$label)

binary_model_accuracy3 <- binary_confusion_matrix3$overall["Accuracy"]

binary_model_accuracy3
```

```
##  Accuracy
## 0.9832776
```

```r
# k=15 binary k-nearest neighbor model

set.seed(123)

binary_knn_model4 <- train(label ~ x + y,
                    data = binary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 15)
```

```r
predictions4 <- predict(binary_knn_model4, newdata = binary_test_data)

binary_confusion_matrix4 <- confusionMatrix(predictions4, binary_test_data$label)

binary_model_accuracy4 <- binary_confusion_matrix4$overall["Accuracy"]

binary_model_accuracy4
```

```
##  Accuracy
## 0.9832776
```

```r
# k=20 binary k-nearest neighbor model

set.seed(123)

binary_knn_model5 <- train(label ~ x + y,
                   data = binary_train_data,
                   method = "knn",
                   trControl = trainControl(method = "cv"),
                   tuneLength = 20)

predictions5 <- predict(binary_knn_model5, newdata = binary_test_data)

binary_confusion_matrix5 <- confusionMatrix(predictions5, binary_test_data$label)

binary_model_accuracy5 <- binary_confusion_matrix5$overall["Accuracy"]

binary_model_accuracy5
```

```
##  Accuracy
## 0.9832776
```

```r
# k=25 binary k-nearest neighbor model

set.seed(123)

binary_knn_model6 <- train(label ~ x + y,
                   data = binary_train_data,
                   method = "knn",
                   trControl = trainControl(method = "cv"),
                   tuneLength = 25)

predictions6 <- predict(binary_knn_model6, newdata = binary_test_data)

binary_confusion_matrix6 <- confusionMatrix(predictions6, binary_test_data$label)

binary_model_accuracy6 <- binary_confusion_matrix6$overall["Accuracy"]

binary_model_accuracy6
```

```
##  Accuracy
## 0.9832776
```

Trinary Classifier Dataset Models

```
# k=3 trinary k-nearest neighbor model

set.seed(123)

trinary_split <- createDataPartition(trinary$label, p = .8,
                                     list = FALSE,
                                     times = 1)
trinary_train_data <- trinary[trinary_split,]
trinary_test_data <- trinary[-trinary_split,]

trinary_train_data$label <- factor(trinary_train_data$label)
trinary_test_data$label <- factor(trinary_test_data$label)

trinary_knn_model1 <- train(label ~ x + y,
                    data = trinary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 3)

predict1 <- predict(trinary_knn_model1, newdata = trinary_test_data)

trinary_confusion_matrix1 <- confusionMatrix(predict1, trinary_test_data$label)

trinary_model_accuracy1 <- trinary_confusion_matrix1$overall["Accuracy"]

trinary_model_accuracy1
```

```
##  Accuracy
## 0.8945687
```

```
# k=5 trinary k-nearest neighbor model

set.seed(123)

trinary_knn_model2 <- train(label ~ x + y,
                    data = trinary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 5)

predict2 <- predict(trinary_knn_model2, newdata = trinary_test_data)

trinary_confusion_matrix2 <- confusionMatrix(predict2, trinary_test_data$label)

trinary_model_accuracy2 <- trinary_confusion_matrix2$overall["Accuracy"]

trinary_model_accuracy2
```

```
##  Accuracy
## 0.8881789
```

```r
# k=10 trinary k-nearest neighbor model

set.seed(123)

trinary_knn_model3 <- train(label ~ x + y,
                    data = trinary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 10)

predict3 <- predict(trinary_knn_model3, newdata = trinary_test_data)

trinary_confusion_matrix3 <- confusionMatrix(predict3, trinary_test_data$label)

trinary_model_accuracy3 <- trinary_confusion_matrix3$overall["Accuracy"]

trinary_model_accuracy3
```

```
##  Accuracy
## 0.8881789
```

```r
# k=15 trinary k-nearest neighbor model

set.seed(123)

trinary_knn_model4 <- train(label ~ x + y,
                    data = trinary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 15)

predict4 <- predict(trinary_knn_model4, newdata = trinary_test_data)

trinary_confusion_matrix4 <- confusionMatrix(predict4, trinary_test_data$label)

trinary_model_accuracy4 <- trinary_confusion_matrix4$overall["Accuracy"]

trinary_model_accuracy4
```

```
##  Accuracy
## 0.8881789
```

```r
# k=20 trinary k-nearest neighbor model

set.seed(123)

trinary_knn_model5 <- train(label ~ x + y,
                    data = trinary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 20)
```

```r
predict5 <- predict(trinary_knn_model5, newdata = trinary_test_data)

trinary_confusion_matrix5 <- confusionMatrix(predict5, trinary_test_data$label)

trinary_model_accuracy5 <- trinary_confusion_matrix5$overall["Accuracy"]

trinary_model_accuracy5
```

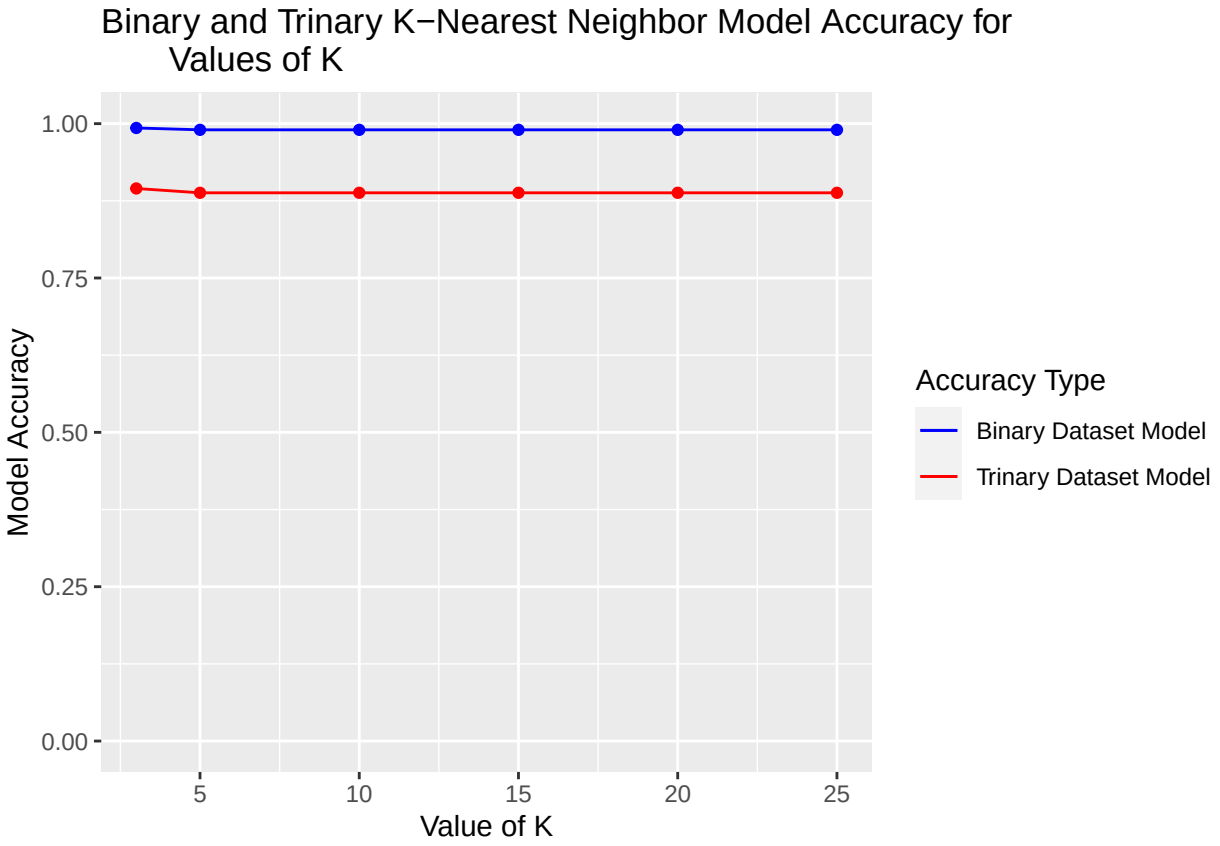
```
##  Accuracy
## 0.8881789
```

```r
# k=25 trinary k-nearest neighbor model

set.seed(123)

trinary_knn_model6 <- train(label ~ x + y,
                    data = trinary_train_data,
                    method = "knn",
                    trControl = trainControl(method = "cv"),
                    tuneLength = 25)

predict6 <- predict(trinary_knn_model6, newdata = trinary_test_data)

trinary_confusion_matrix6 <- confusionMatrix(predict6, trinary_test_data$label)

trinary_model_accuracy6 <- trinary_confusion_matrix6$overall["Accuracy"]

trinary_model_accuracy6
```

```
##  Accuracy
## 0.8881789
```

```r
# To create a graph of the k values and their respective accuracy metrics, I
# will create three vectors to hold those values, create a data frame with them,
# and create a scatter plot to visualize the relationship

k_values <- c(3, 5, 10, 15, 20, 25)
b_accuracy_values <- c(0.993, 0.99, 0.99, 0.99, 0.99, 0.99)
t_accuracy_values <- c(0.895, 0.888, 0.888, 0.888, 0.888, 0.888)

k_accuracy_metrics <- data.frame(k = k_values,
                    b_accuracy = b_accuracy_values,
                    t_accuracy = t_accuracy_values)

ggplot(k_accuracy_metrics, aes(x = k, y = b_accuracy)) +
  geom_point(color = "blue") +
  geom_point(aes(y = t_accuracy), color = "red") +
  geom_line(aes(y = b_accuracy, color = "Blue")) +
  geom_line(aes(y = t_accuracy, color = "Red")) +
  scale_color_manual(name = "Accuracy Type",
                     values = c("Blue" = "blue", "Red" = "red"),
                     labels = c("Binary Dataset Model",
```

```
                                "Trinary Dataset Model")) +
  labs(title = "Binary and Trinary K-Nearest Neighbor Model Accuracy for
       Values of K",
       x = "Value of K",
       y = "Model Accuracy") +
  ylim(0, 1)
```



1C. Looking back at the plots of the data, do you think a linear classifier would work well on these datasets?

I would say that a linear classifier would not work well on these datasets as per the scatter plots, the data points of each dataset are spread far and wide and do not congregate around an area of the plot that could be categorized as having a linear relationship. Linear classifiers work best for variables that are continuous and not categorical in nature. Even though the label variable in each dataset contains numbers, the meaning of those values is categorical, and looking at the scatter plots, the conditions to be categorized into one label or another do not show a linear relationship as the classification clusters are sporadic across the plots.

1D. How does the accuracy of your logistic regression classifier from last week compare? Why is the accuracy different between these two methods?

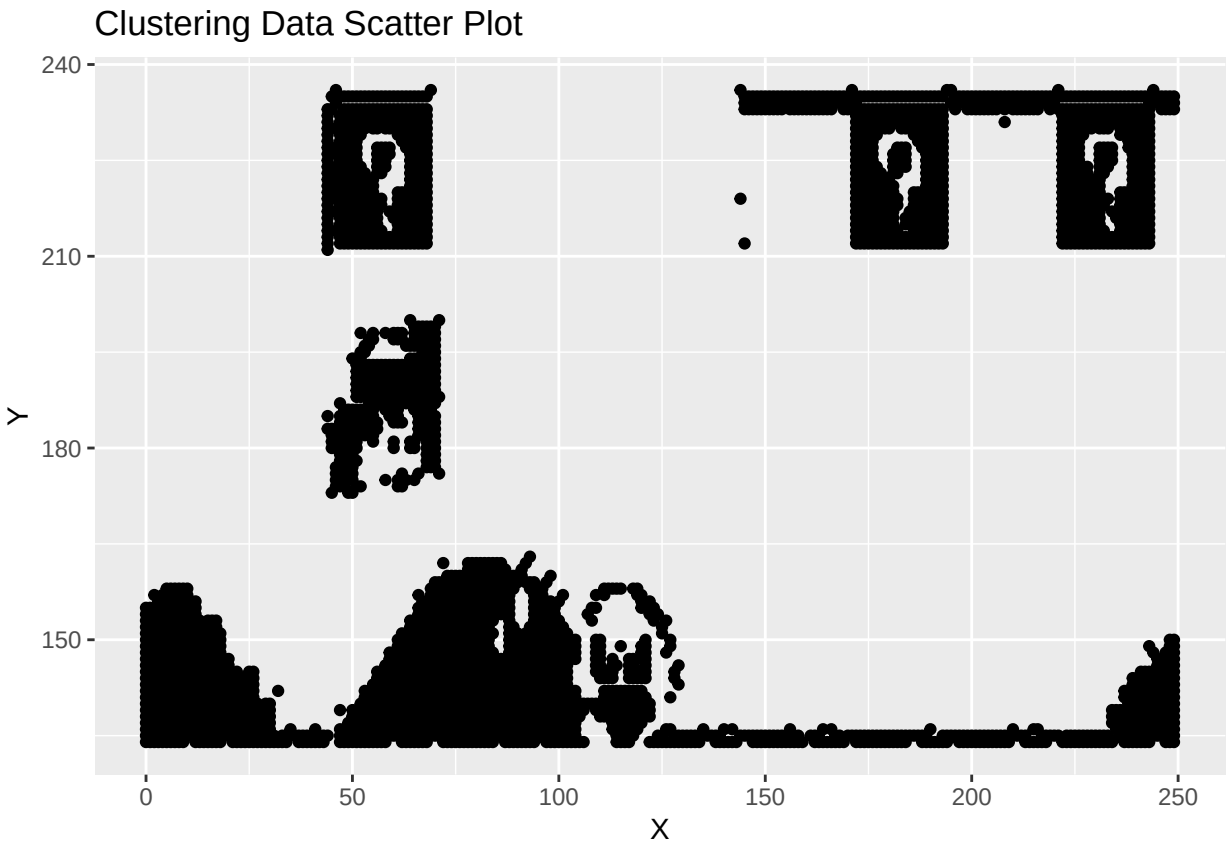
The accuracy of the logistic regression model on the binary classifier dataset was 59.13%, a far cry from the k-nearest neighbor model accuracy of 98.7% with k being equal to 3. The accuracy is different between these two methods due to there not being a significant linear relationship between the predictor and outcome variables. This nonlinearity causes logistic regression models to perform poorly if we are solely basing model performance on accuracy. K-nearest neighbor models can account for a large amount of spread between data points and does not assume linearity.

K-Means Clustering Dataset Scatter Plot

```
# Import the Clustering Data CSV file to work with its contents

cluster_data <- read_csv("clustering_data.csv", show_col_types = FALSE)
```

2A. Plot the dataset using a scatter plot.

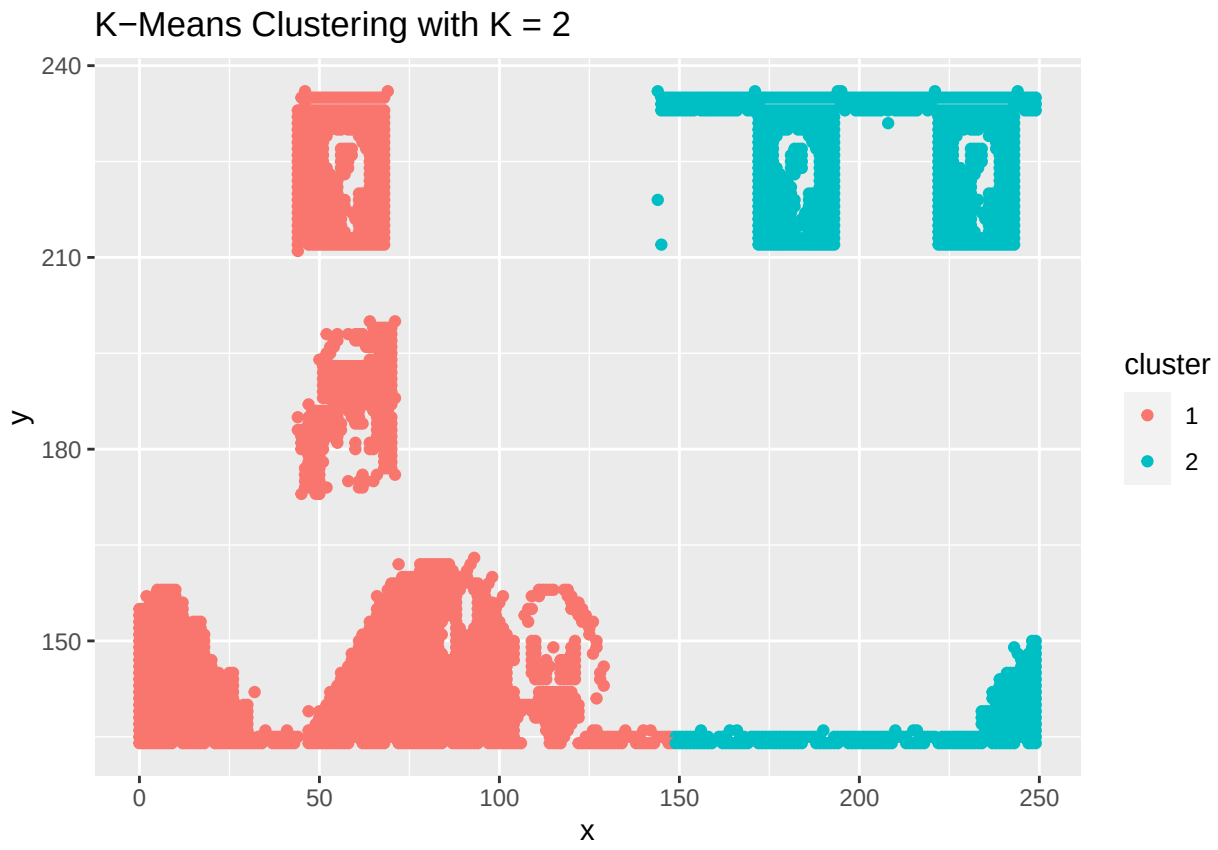
```
ggplot(cluster_data, aes(x = x, y = y)) +
  geom_point() +
  labs(title = "Clustering Data Scatter Plot",
       x = "X",
       y = "Y")
```

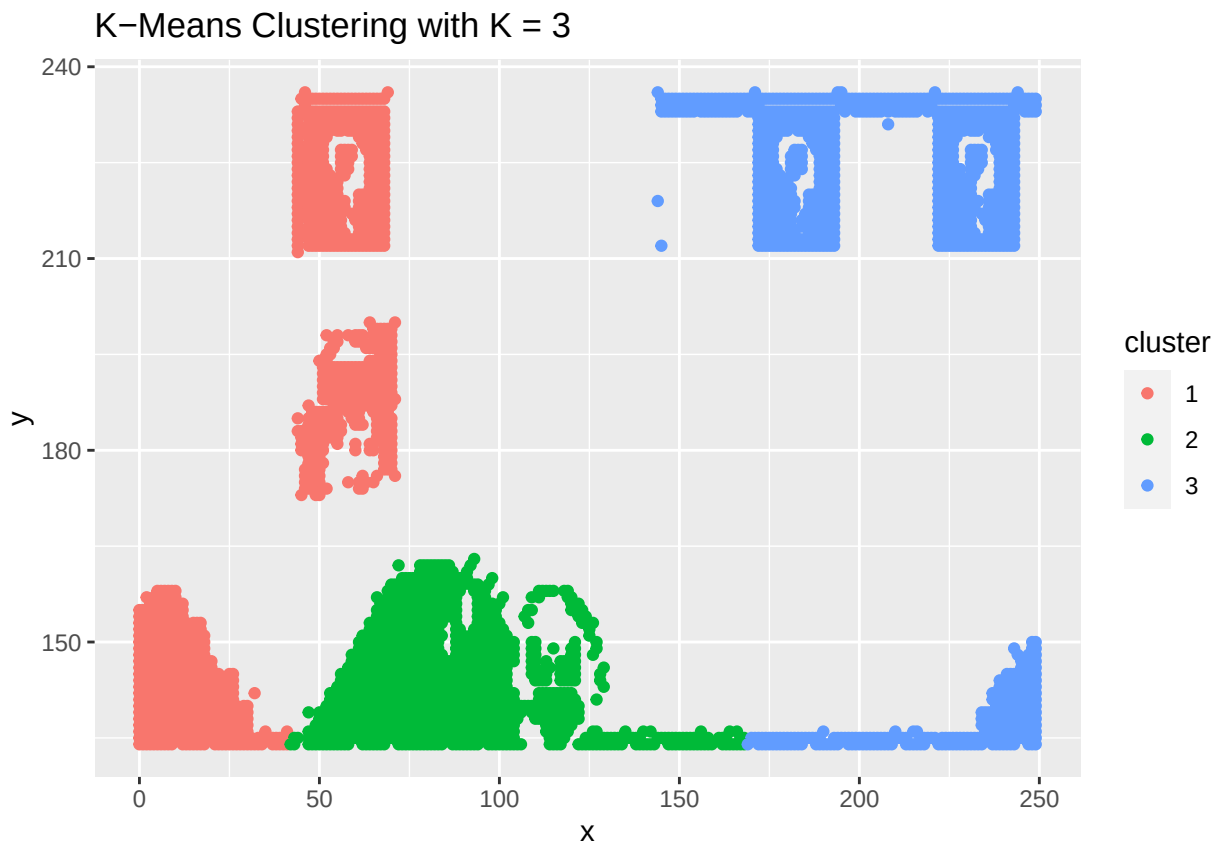


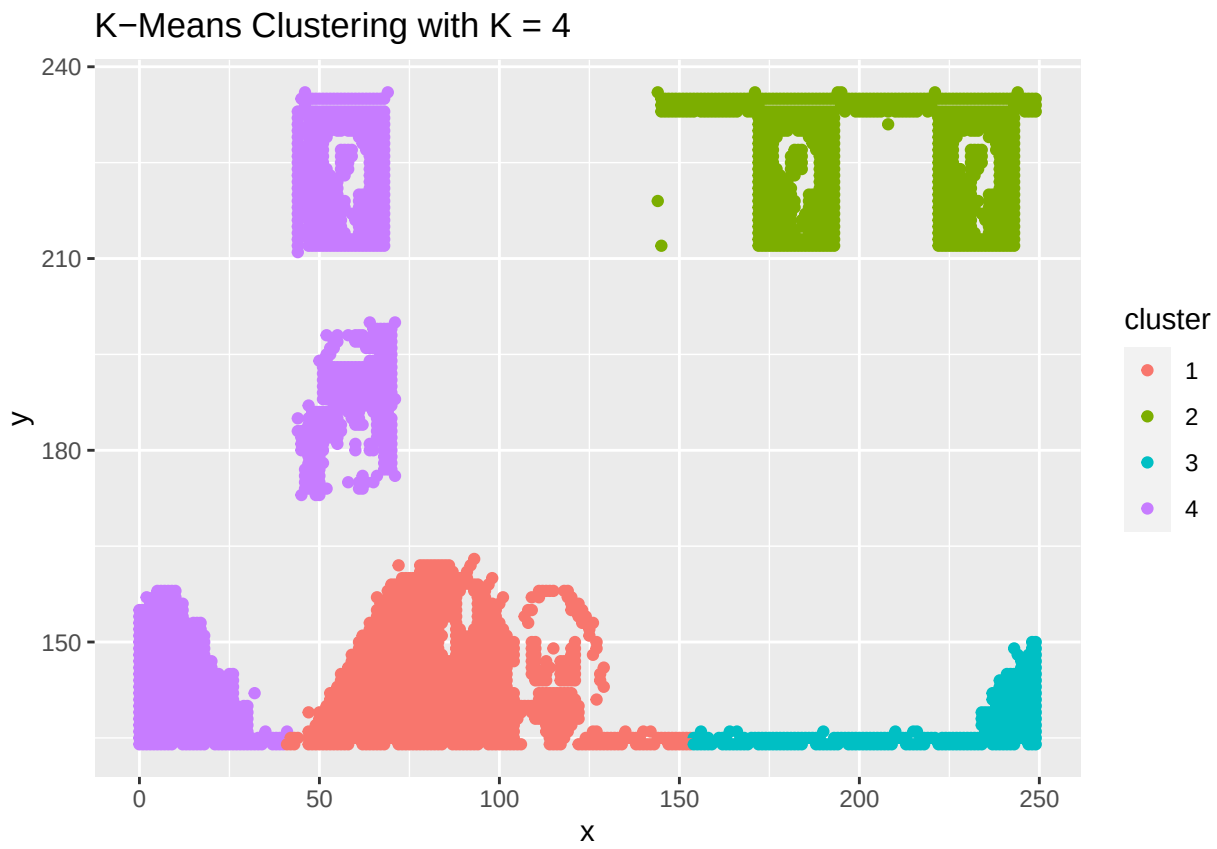
2B. Fit the dataset using the k-means algorithm from k=2 to k=12. Create a scatter plot of the resultant clusters for each value of k.

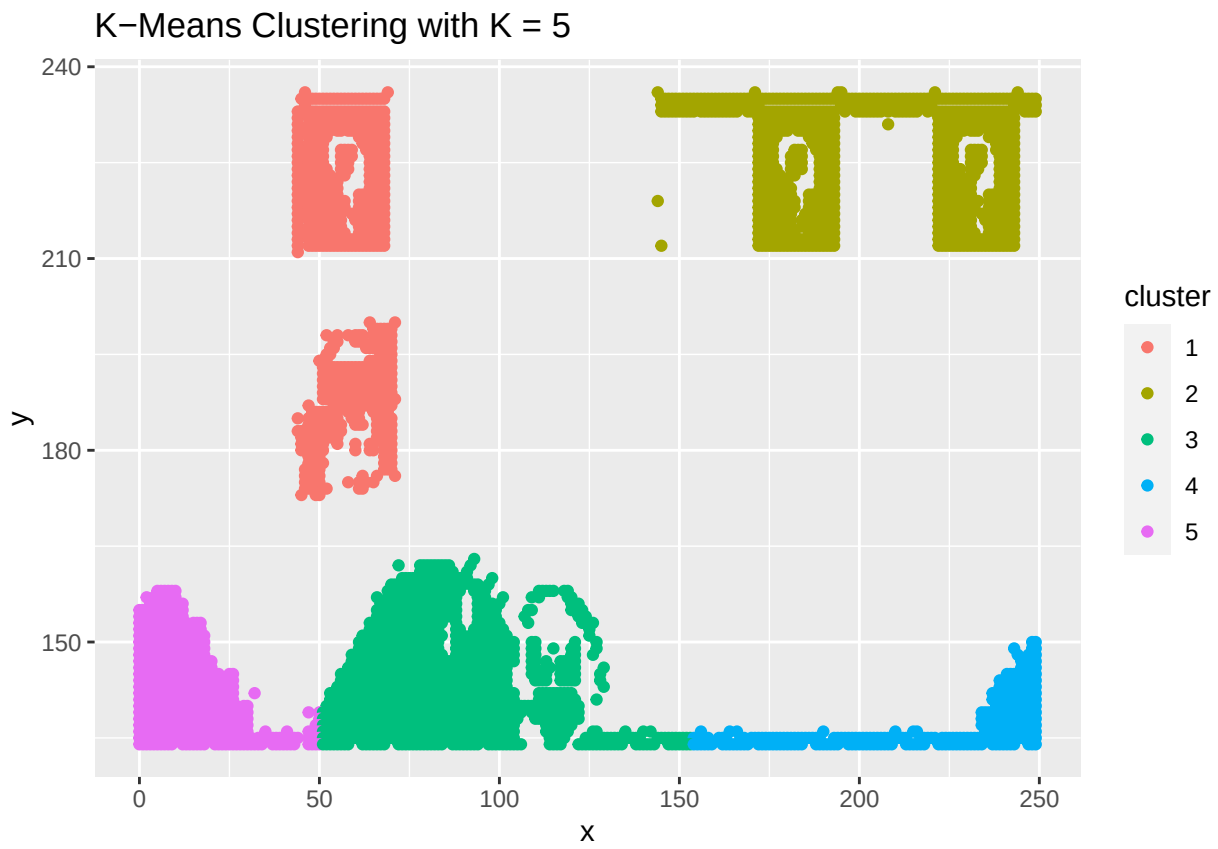
```
# K-means clustering can be performed using the kmeans() function

# I have used a for loop to run the k-means clustering algorithm for each
# value of k from 2 through 12, create the cluster variable within the dataset
# as a factor of the k-means clusters obtained by the algorithm, and plot the
# scatter plots of each algorithm with the title of each scatter plot being
# changed to match the number of clusters with the paste() function
```

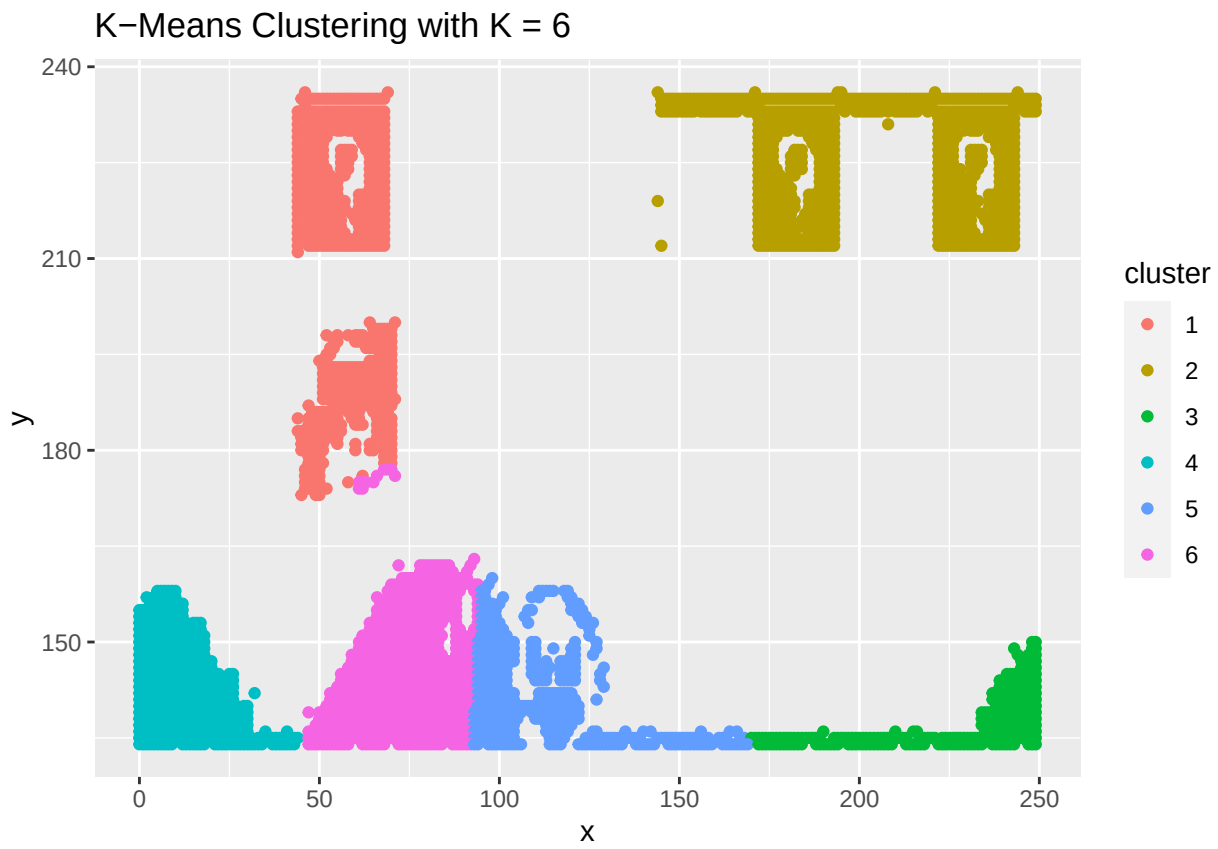
```r
for(k in 2:12) {
  kmeans_result <- kmeans(cluster_data, centers = k)

  cluster_data$cluster <- as.factor(kmeans_result$cluster)

  print(
    ggplot(cluster_data, aes(x = x, y = y, color = cluster)) +
      geom_point() +
      ggtitle(paste("K-Means Clustering with K =", k))
  )
}
```

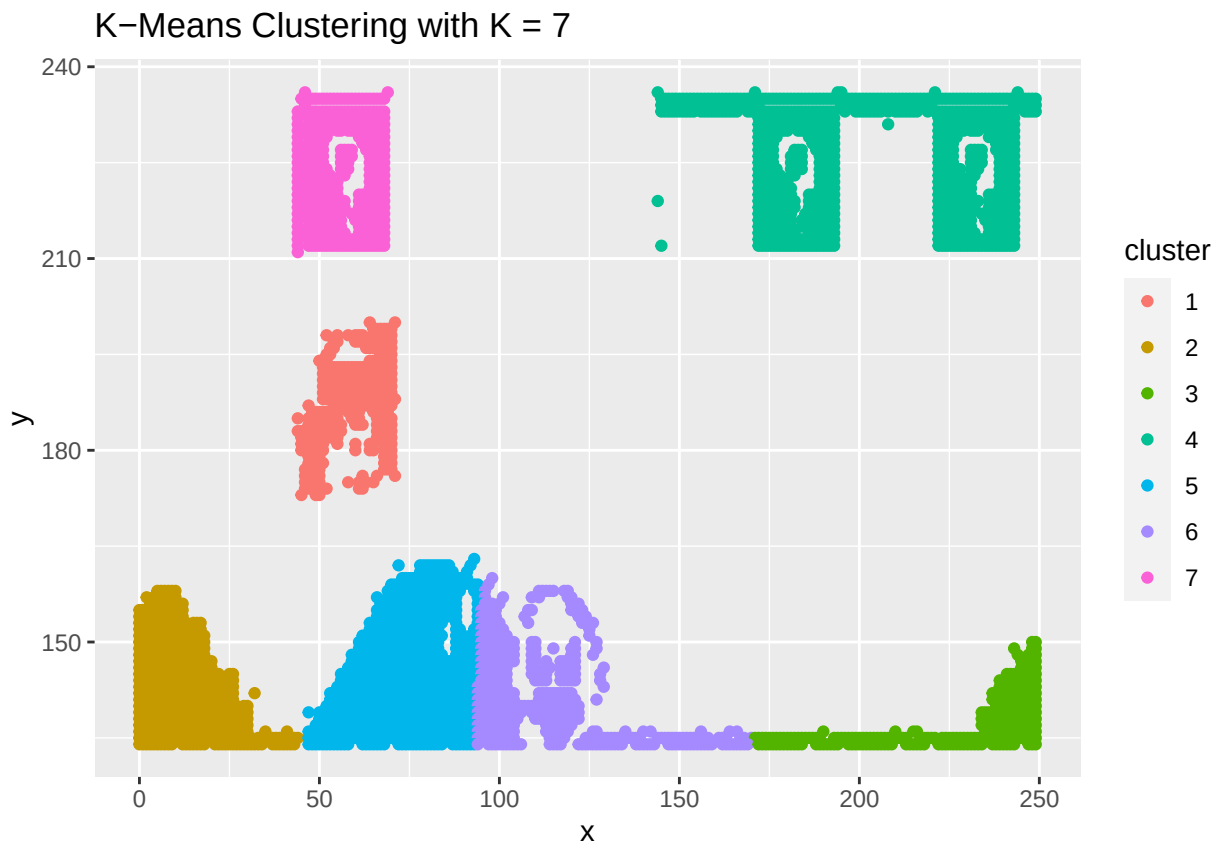


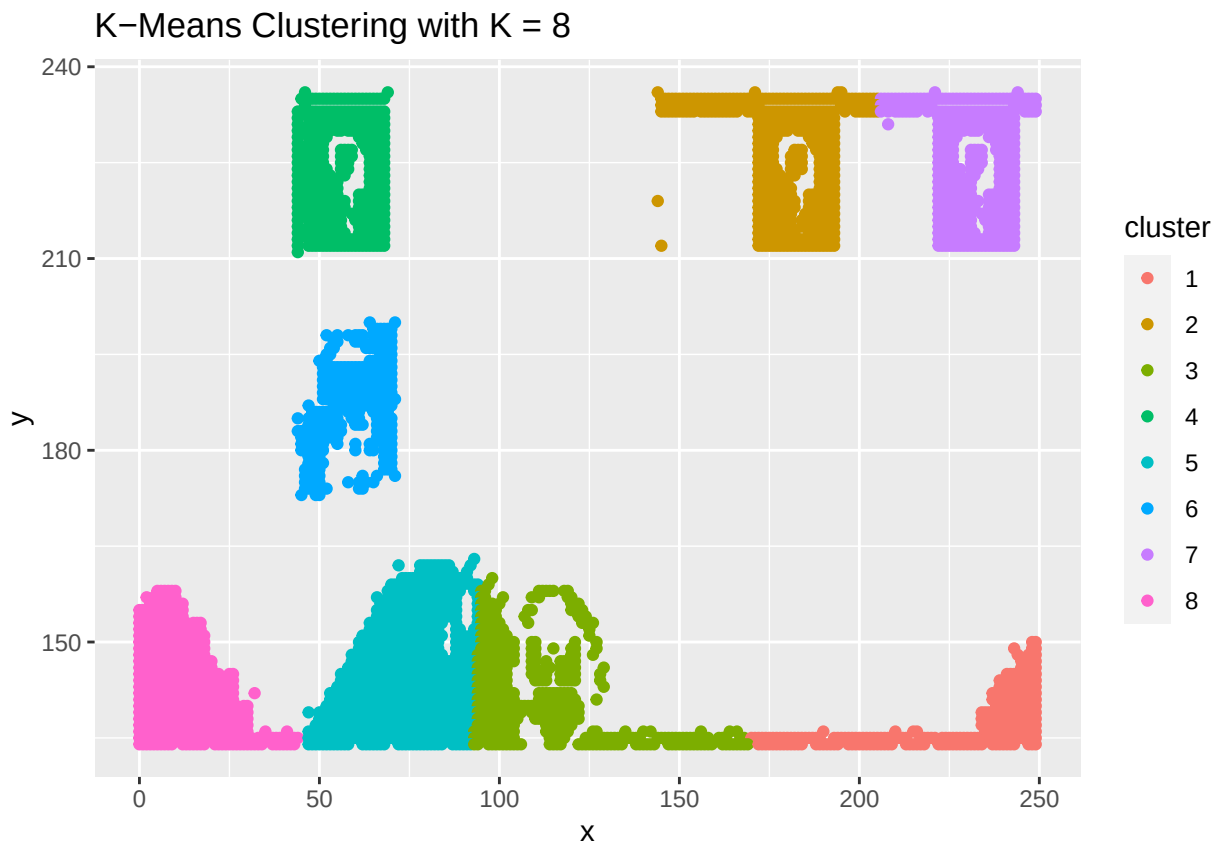

K−Means Clustering with K = 3
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3

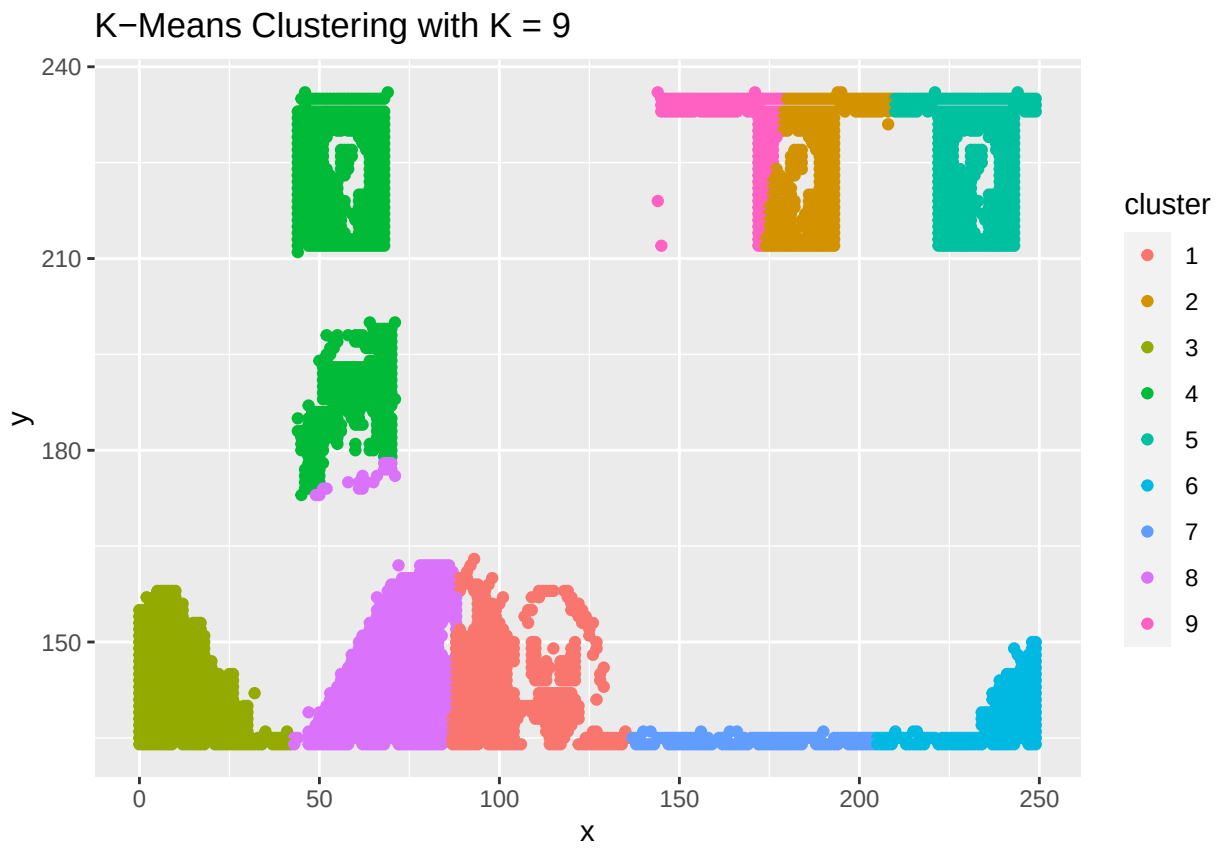

K−Means Clustering with K = 4
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4

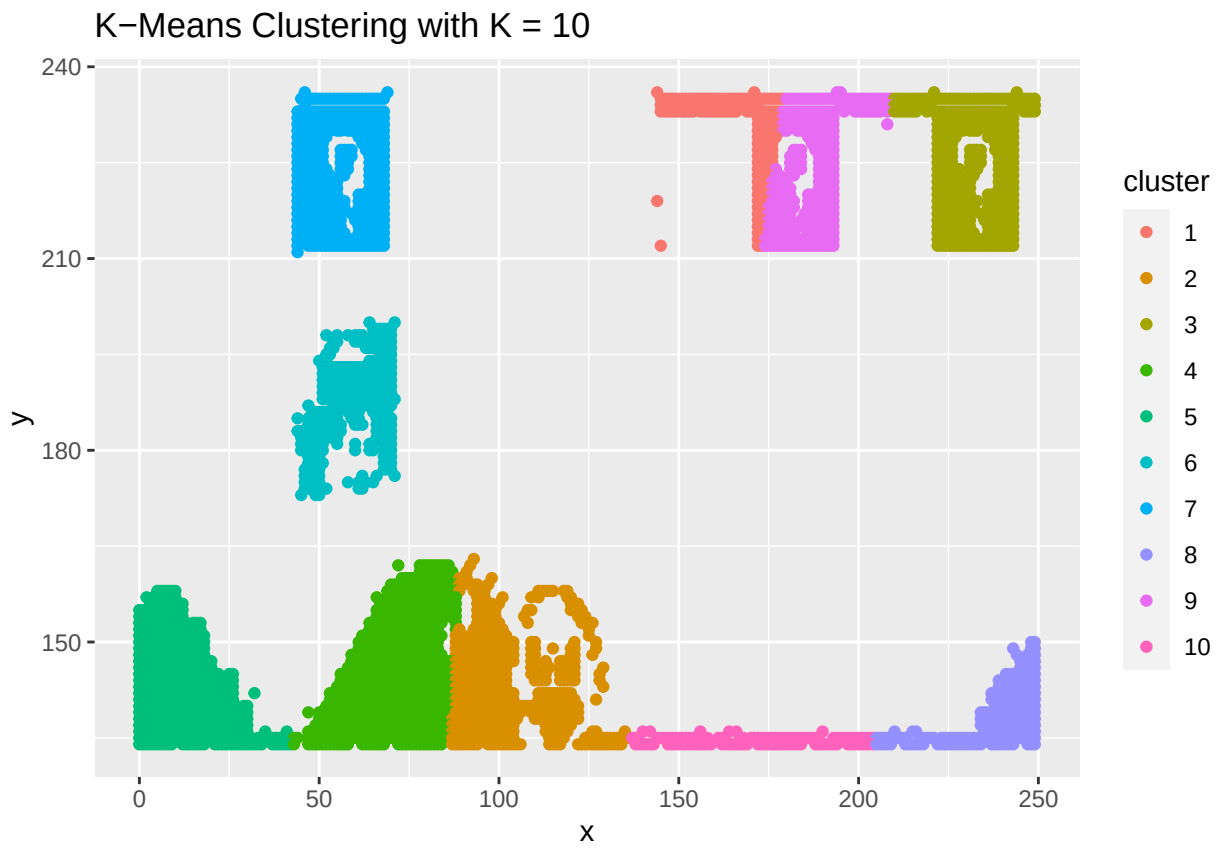

K−Means Clustering with K = 5
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5

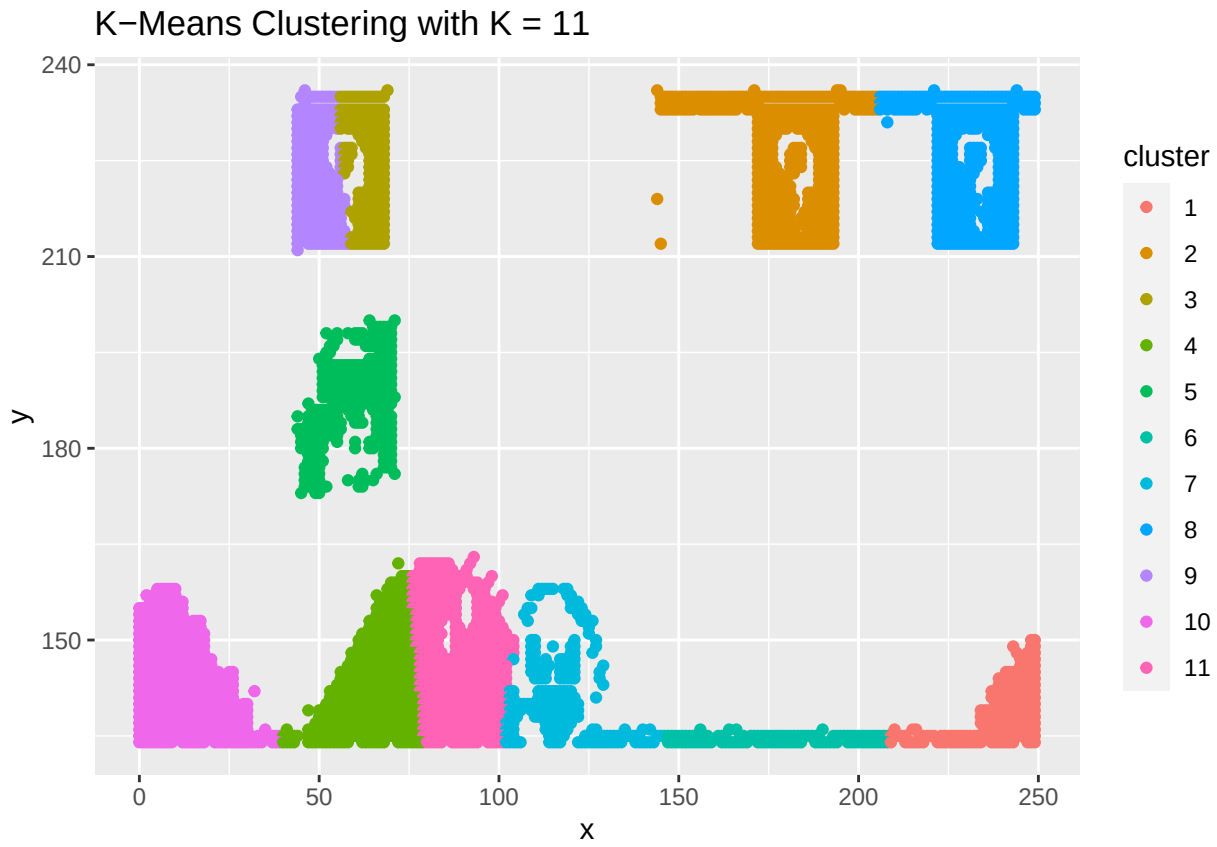

K−Means Clustering with K = 6
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6

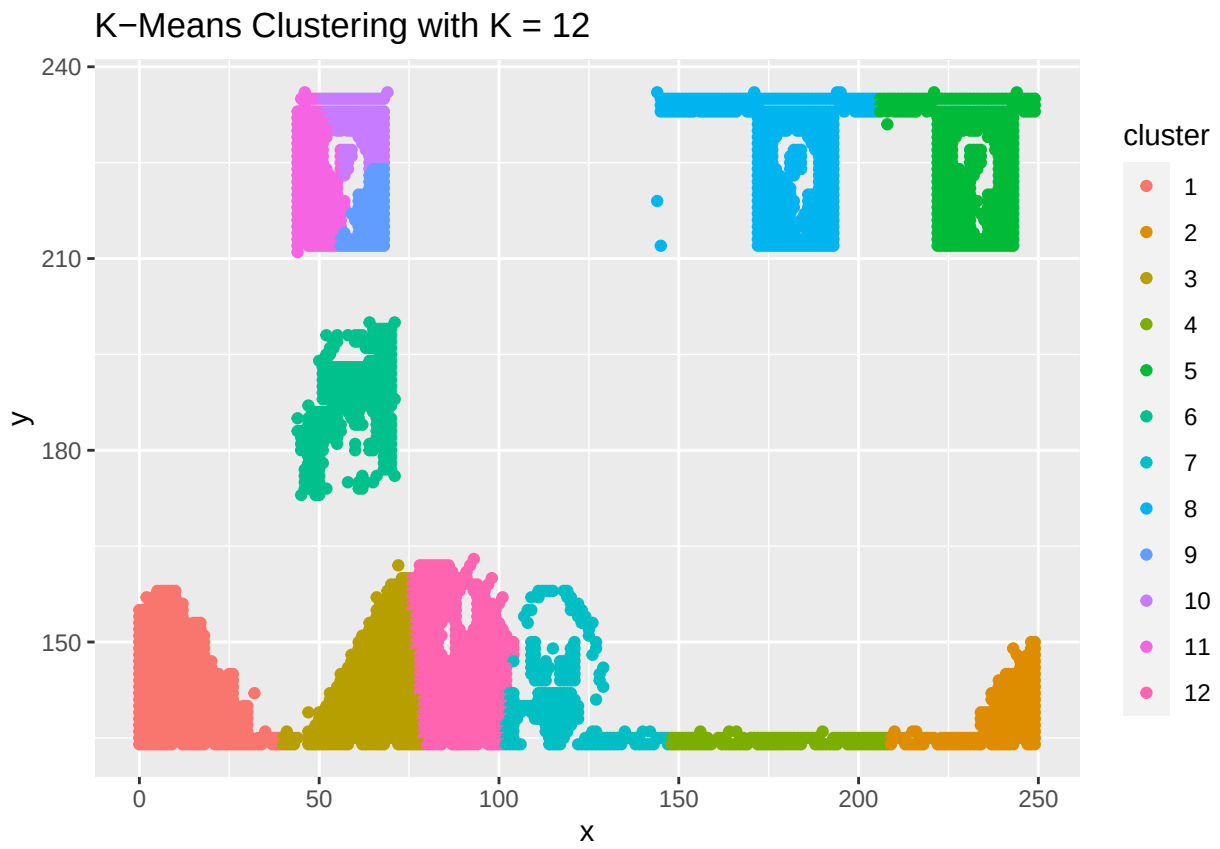

K−Means Clustering with K = 7
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7


K−Means Clustering with K = 8
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7
8


K−Means Clustering with K = 9
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7
8
9


K−Means Clustering with K = 10
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7
8
9
10


K−Means Clustering with K = 11
240
210
180
150
y
0
50
100
150
200
250
x
cluster
1
2
3
4
5
6
7
8
9
10
11



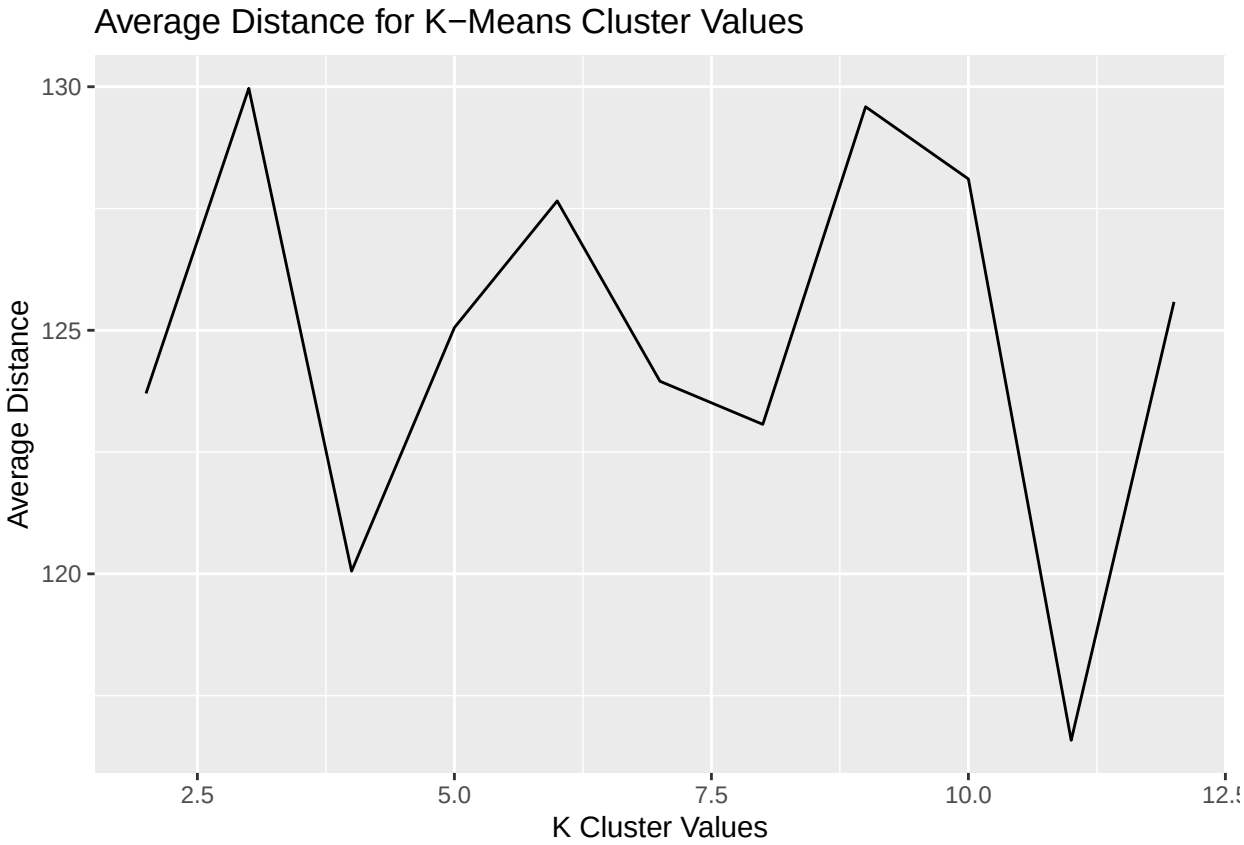
2C. Calculate this average distance from the center of each cluster for each value of k and plot it as a line chart where k is the x-axis and the average distance is the y-axis.

```
# After running this code chunk multiplt times, I found that I needed to set
# the seed for reproducibility

set.seed(123)

# An empty numerical vector needs to be created that will hold the average
# distance for each value of k=2 through k=12 using the numeric() function

avg_distances <- numeric(11)

# I am repeating the for loop for the k-means clustering algorithm as I have
# created a nested for loop which computes the average distance and stores each
# average distance into the empty vector

for (k in 2:12) {
  kmeans_result <- kmeans(cluster_data, centers = k)

# Here the distance variable captures every single x/y pair in the dataset and
# stores those row pairs in a numerical vector using the nrow() function

  distances <- numeric(nrow(cluster_data))

  # In order to place each of the data points in the dataset with each cluster,
```

```
  # define the center point of each cluster, and measure all of the distances
  # of the data points within each cluster using the Euclidean distance formula,
  # the inner for loop below is needed

  for (i in 1:k) {
    cluster_points <- cluster_data[kmeans_result$cluster == i, c("x", "y")]
    center <- kmeans_result$centers[i, ]
    distances[kmeans_result$cluster == i] <- sqrt(rowSums((cluster_points - center)^2))
  }

  # The average distance vector created above now is filled with the average
  # distance value for each k value of clusters using the mean() function

  avg_distances[k - 1] <- mean(distances)
}

print(avg_distances)
```

```
##  [1] 123.7034 129.9676 120.0548 125.0540 127.6559 123.9530 123.0692 129.5889
##  [9] 128.1069 116.5820 125.5829
```

```
# Creating the line chart requires the creation of a second vector of k-means
# cluster values, turn the two vectors into a data frame, and then plot the data

k_cluster_values <- c(2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)

avg_dist_data <- data.frame(x = k_cluster_values, y = avg_distances)

ggplot(avg_dist_data, aes(x = x, y = y)) +
  geom_line() +
  labs(
    title = "Average Distance for K-Means Cluster Values",
    x = "K Cluster Values",
    y = "Average Distance"
  )
```



2D. Looking at the graph you generated in the previous example, what is the elbow point for this dataset?

Visually looking at the line chart, the elbow point of this dataset is where k is equal to 9, as the rate of decrease in distance is the lowest throughout the chart.