Homework_2

Claire Kraft

2024-09-03

Question 3.1

Using the same data set (credit_card_data.txt or credit_card_data-headers.txt) as in Question 2.2, use the ksvm or kknn function to find a good classifier: (a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional); and (b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

3.1(a) using cross-validation (do this for the k-nearest-neighbors model; SVM is optional); and It seems k=1 is the best with an accuracy of 81.50% which means a high bias as it prefers itself. But Looking at the ggplot we can see K=5 is the tipping point of the “elbow” graph for prediction power.

```
#------------------------- Libraries
library(kknn)
library(ggplot2)

#------------------------- Clean workspace
rm(list = ls())

#------------------------- Load and explore data
# Read the credit card data from the specified file
credit_df <- read.table("C://Users//Clair//OneDrive//Documents//GitHub//omsa//ISYE 6501//Homewor
k 2//credit_card_data.txt", sep = "\t", header = FALSE)
head(credit_df)
```

```
##   V1    V2    V3   V4 V5 V6 V7 V8  V9 V10 V11
## 1  1 30.83 0.000 1.25  1  0  1  1 202   0   1
## 2  0 58.67 4.460 3.04  1  0  6  1  43 560   1
## 3  0 24.50 0.500 1.50  1  1  0  1 280 824   1
## 4  1 27.83 1.540 3.75  1  0  5  0 100   3   1
## 5  1 20.17 5.625 1.71  1  1  0  1 120   0   1
## 6  1 32.08 4.000 2.50  1  1  0  0 360   0   1
```

```
# Display a summary of the dataset, showing basic statistics for each column
summary(credit_df)
```

```
##        V1               V2              V3                V4
##  Min.   :0.0000   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:0.0000   1st Qu.:22.58   1st Qu.: 1.040   1st Qu.: 0.165
##  Median :1.0000   Median :28.46   Median : 2.855   Median : 1.000
##  Mean   :0.6896   Mean   :31.58   Mean   : 4.831   Mean   : 2.242
##  3rd Qu.:1.0000   3rd Qu.:38.25   3rd Qu.: 7.438   3rd Qu.: 2.615
##  Max.   :1.0000   Max.   :80.25   Max.   :28.000   Max.   :28.500
##        V5               V6               V7                V8
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median : 0.000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   : 2.498   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :67.000   Max.   :1.0000
##        V9                V10               V11
##  Min.   :   0.00   Min.   :     0   Min.   :0.0000
##  1st Qu.:  70.75   1st Qu.:     0   1st Qu.:0.0000
##  Median : 160.00   Median :     5   Median :0.0000
##  Mean   : 180.08   Mean   :  1013   Mean   :0.4526
##  3rd Qu.: 271.00   3rd Qu.:   399   3rd Qu.:1.0000
##  Max.   :2000.00   Max.   :100000   Max.   :1.0000
```

```
# Get the dimensions of the dataset (rows x columns)
dim(credit_df)
```

```
## [1] 654  11
```

```
# Check for missing values in the dataset
sum(is.na(credit_df))
```

```
## [1] 0
```

```
#------------------------- Cross-validation for different K values
# borrowing my homework 1 code snippets for the cross validation and hypertuning of the model
# Initialize a vector to store accuracies for each K value
accuracies <- c()

# Loop over different K values (1 to 10)
for (K_value in 1:10) {
  # Count correct predictions
  correct_predictions <- 0

  # look at [1] of the references
  # Random selection of the data: 80% for training and 20% for testing
  ## 80% of 654 = 523
  ## 20% of 654 = 131
  random_sample <- sample(1:nrow(credit_df), 523, replace = FALSE)
  train_data <- credit_df[random_sample, ]
  test_data <- credit_df[-random_sample, ]

  # Fit the kknn model with internal scaling
  hypertuned_model <- kknn(V11 ~ ., train = train_data, test = test_data, k = K_value, scale = T
RUE)

  # Get the predicted labels for the test data
  preds <- fitted(hypertuned_model)

  # Compare predictions with true labels for the test data
  correct_predictions <- sum(as.character(preds) == as.character(test_data$V11))

  # Calculate accuracy for this K value
  accuracy <- correct_predictions / nrow(test_data)

  # Print the K value and corresponding accuracy
  print(paste("K =", K_value, "-> Accuracy:", accuracy))

  # Store the accuracy for each model
  accuracies <- c(accuracies, accuracy)
}
```

```
## [1] "K = 1 -> Accuracy: 0.847328244274809"
## [1] "K = 2 -> Accuracy: 0.641221374045801"
## [1] "K = 3 -> Accuracy: 0.595419847328244"
## [1] "K = 4 -> Accuracy: 0.610687022900763"
## [1] "K = 5 -> Accuracy: 0.595419847328244"
## [1] "K = 6 -> Accuracy: 0.488549618320611"
## [1] "K = 7 -> Accuracy: 0.450381679389313"
## [1] "K = 8 -> Accuracy: 0.419847328244275"
## [1] "K = 9 -> Accuracy: 0.412213740458015"
## [1] "K = 10 -> Accuracy: 0.366412213740458"
```

```
# Find the best K for the scaled model
best_k_scaled <- which.max(accuracies)
best_accuracy_scaled <- max(accuracies)

print(paste("Best K for scaled data:", best_k_scaled))
```

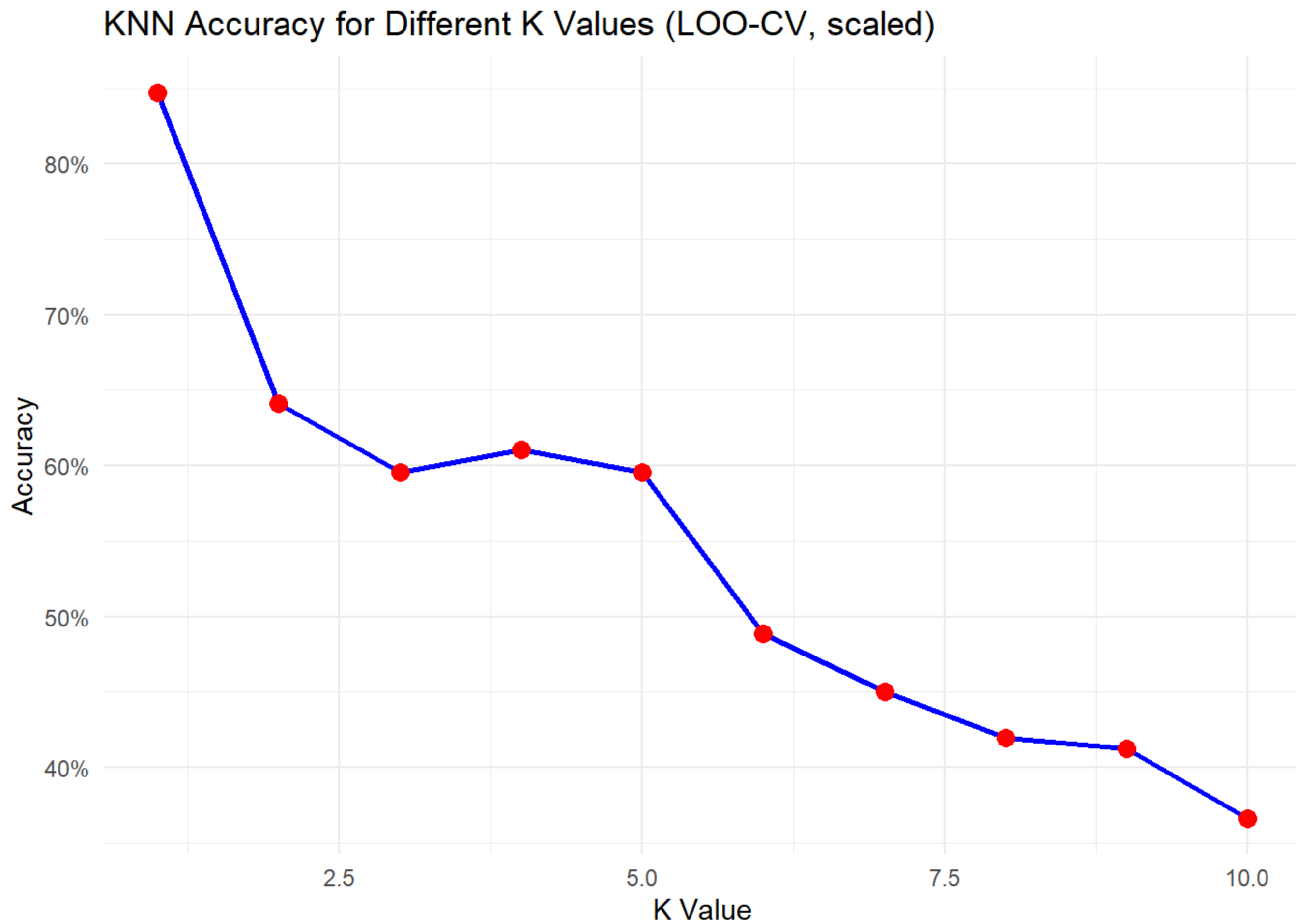
```
## [1] "Best K for scaled data: 1"
```

```
print(paste("Best accuracy for scaled data:", best_accuracy_scaled))
```

```
## [1] "Best accuracy for scaled data: 0.847328244274809"
```

```
#------------------------- viz
# Plotting the accuracy vs K value
ggplot(data.frame(K = 1:10, Accuracy = accuracies), aes(x = K, y = Accuracy)) +
  geom_line(color = "blue", size = 1) +
  geom_point(color = "red", size = 3) +
  labs(title = "KNN Accuracy for Different K Values (LOO-CV, scaled)",
       x = "K Value",
       y = "Accuracy") +
  theme_minimal() +
  scale_y_continuous(labels = scales::percent)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



3.1(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional). In the lectures Dr. Sokol explained that data can be split randomly or in rotations. In this exercise I chose to randomly split my data (3 ways) because rotational data can be organized in such a way that may make the data have bias. The splitting can be 60-20-20 or 70-15-15, 80-10-10. I chose the 80-10-10 configuration. The best k still remains to be k=1. However the accuracy was decreased by a little compared to when the prediction was pitted against testing data in 3.1a. I even shuffled all indices for the train, validate, and test data to eliminate any potential biases. The better performance in 3.1a compared to 3.1b may be due to the test data being lucky [3]. I also wonder if the 20% split was halved making the pool of data to match against even smaller which could mean less likelyhood of matching which could mean lower accuracy.

```
#------------------------- Libraries
library(kknn)
library(ggplot2)

#------------------------- Clean workspace
rm(list = ls())

#------------------------- Load and explore data
# Read the credit card data from the specified file
credit_df <- read.table("C://Users//Clair//OneDrive//Documents//GitHub//omsa//ISYE 6501//Homewor
k 2//credit_card_data.txt", sep = "\t", header = FALSE)
head(credit_df)
```

```
##   V1    V2    V3   V4 V5 V6 V7 V8  V9 V10 V11
## 1  1 30.83 0.000 1.25  1  0  1  1 202   0   1
## 2  0 58.67 4.460 3.04  1  0  6  1  43 560   1
## 3  0 24.50 0.500 1.50  1  1  0  1 280 824   1
## 4  1 27.83 1.540 3.75  1  0  5  0 100   3   1
## 5  1 20.17 5.625 1.71  1  1  0  1 120   0   1
## 6  1 32.08 4.000 2.50  1  1  0  0 360   0   1
```

```
# Display a summary of the dataset, showing basic statistics for each column
summary(credit_df)
```

```
##        V1               V2              V3               V4
##  Min.   :0.0000   Min.   :13.75   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:0.0000   1st Qu.:22.58   1st Qu.: 1.040   1st Qu.: 0.165
##  Median :1.0000   Median :28.46   Median : 2.855   Median : 1.000
##  Mean   :0.6896   Mean   :31.58   Mean   : 4.831   Mean   : 2.242
##  3rd Qu.:1.0000   3rd Qu.:38.25   3rd Qu.: 7.438   3rd Qu.: 2.615
##  Max.   :1.0000   Max.   :80.25   Max.   :28.000   Max.   :28.500
##        V5               V6               V7               V8
##  Min.   :0.0000   Min.   :0.0000   Min.   : 0.000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.: 0.000   1st Qu.:0.0000
##  Median :1.0000   Median :1.0000   Median : 0.000   Median :1.0000
##  Mean   :0.5352   Mean   :0.5612   Mean   : 2.498   Mean   :0.5382
##  3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.: 3.000   3rd Qu.:1.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :67.000   Max.   :1.0000
##        V9               V10              V11
##  Min.   :   0.00   Min.   :     0   Min.   :0.0000
##  1st Qu.:  70.75   1st Qu.:     0   1st Qu.:0.0000
##  Median : 160.00   Median :     5   Median :0.0000
##  Mean   : 180.08   Mean   :  1013   Mean   :0.4526
##  3rd Qu.: 271.00   3rd Qu.:   399   3rd Qu.:1.0000
##  Max.   :2000.00   Max.   :100000   Max.   :1.0000
```

```
# Get the dimensions of the dataset (rows x columns)
dim(credit_df)
```

```
## [1] 654  11
```

```
# Check for missing values in the dataset
sum(is.na(credit_df))
```

```
## [1] 0
```

```
#------------------------- Split Data
set.seed(1)  # For reproducibility

# Shuffle the data
shuffled_indices <- sample(1:nrow(credit_df))

# Calculate the number of rows for each set
# look at [2] of the references
total_rows <- nrow(credit_df)
train_rows <- floor(0.8 * total_rows)
validate_rows <- floor(0.1 * total_rows)
test_rows <- total_rows - train_rows - validate_rows

# Split indices for training (80%), validation (10%), and testing (10%)
train_indices <- shuffled_indices[1:train_rows]
validate_indices <- shuffled_indices[(train_rows + 1):(train_rows + validate_rows)]
test_indices <- shuffled_indices[(train_rows + validate_rows + 1):total_rows]

# Create datasets
train_data <- credit_df[train_indices, ]
validate_data <- credit_df[validate_indices, ]
test_data <- credit_df[test_indices, ]

#------------------------- Cross-validation for different K values
# Initialize a vector to store accuracies for each K value
accuracies <- c()

# Loop over different K values (1 to 30)
for (K_value in 1:10) {
  correct_predictions <- 0  # Count correct predictions

  # Fit the kknn model with internal scaling on the training data
  hypertuned_model <- kknn(V11 ~ ., train = train_data, test = validate_data, k = K_value, scale
= TRUE)

  # Get the predicted labels for the validation data
  preds <- fitted(hypertuned_model)

  # Compare predictions with true labels for the validation data
  correct_predictions <- sum(as.character(preds) == as.character(validate_data$V11))

  # Calculate accuracy for this K value on validation data
  accuracy <- correct_predictions / nrow(validate_data)

  # Print the K value and corresponding accuracy
  print(paste("K =", K_value, "-> Accuracy on validation data:", accuracy))

  # Store the accuracy for each model
  accuracies <- c(accuracies, accuracy)
}
```

```
## [1] "K = 1 -> Accuracy on validation data: 0.8"
## [1] "K = 2 -> Accuracy on validation data: 0.661538461538462"
## [1] "K = 3 -> Accuracy on validation data: 0.584615384615385"
## [1] "K = 4 -> Accuracy on validation data: 0.507692307692308"
## [1] "K = 5 -> Accuracy on validation data: 0.507692307692308"
## [1] "K = 6 -> Accuracy on validation data: 0.507692307692308"
## [1] "K = 7 -> Accuracy on validation data: 0.430769230769231"
## [1] "K = 8 -> Accuracy on validation data: 0.4"
## [1] "K = 9 -> Accuracy on validation data: 0.353846153846154"
## [1] "K = 10 -> Accuracy on validation data: 0.307692307692308"
```

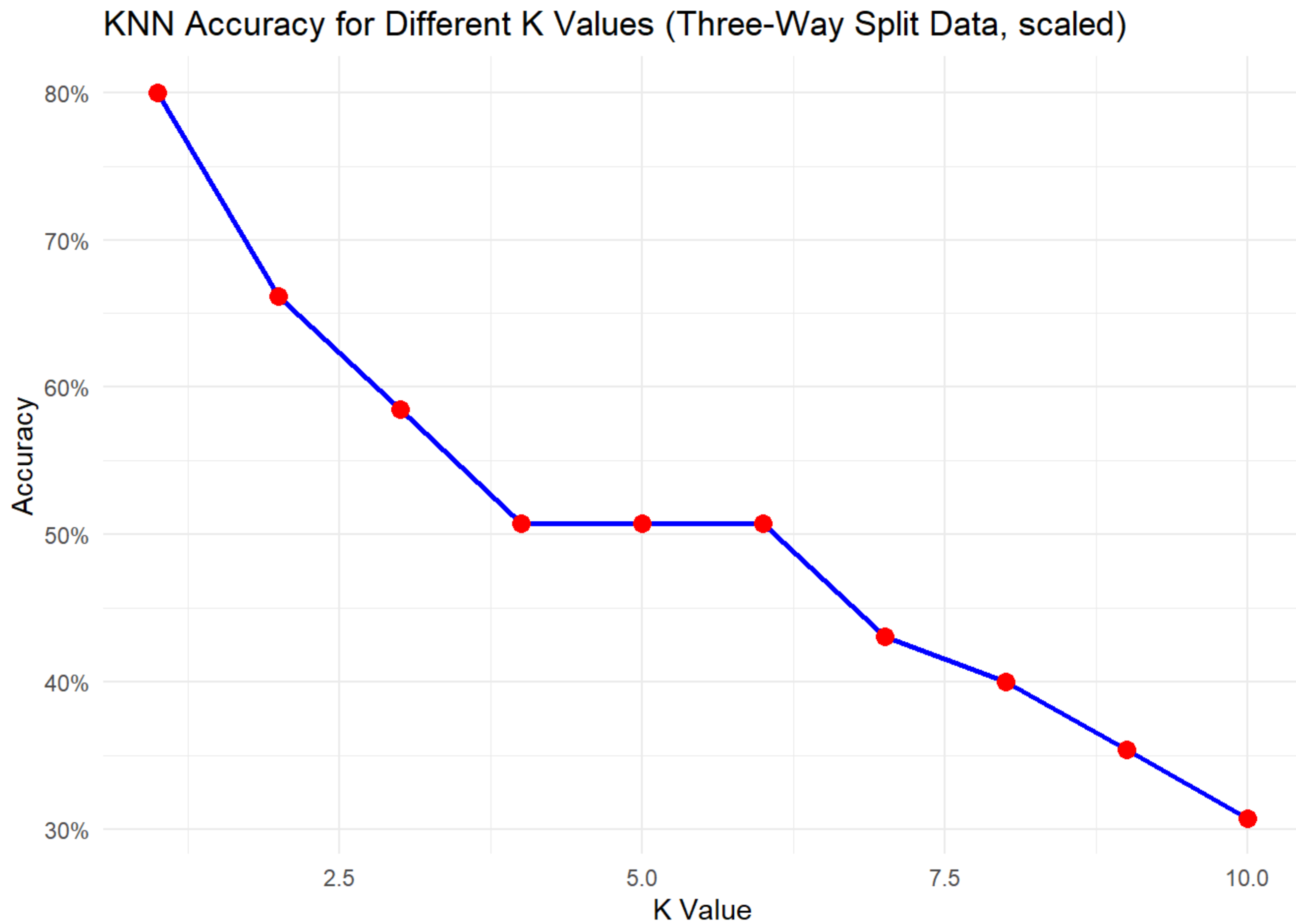
```
# Find the best K for the scaled model
best_k_scaled <- which.max(accuracies)
best_accuracy_scaled <- max(accuracies)

print(paste("Best K for scaled data:", best_k_scaled))
```

```
## [1] "Best K for scaled data: 1"
```

```
print(paste("Best accuracy for scaled data:", best_accuracy_scaled))
```

```
## [1] "Best accuracy for scaled data: 0.8"
```

```
#------------------------- Visualization
# Plotting the accuracy vs K value
ggplot(data.frame(K = 1:10, Accuracy = accuracies), aes(x = K, y = Accuracy)) +
  geom_line(color = "blue", size = 1) +
  geom_point(color = "red", size = 3) +
  labs(title = "KNN Accuracy for Different K Values (Three-Way Split Data, scaled)",
       x = "K Value",
       y = "Accuracy") +
  theme_minimal() +
  scale_y_continuous(labels = scales::percent)
```




```
#------------------------- Final Model Evaluation on Test Data
# Train final model with the best K value on the full training data
final_model <- kknn(V11 ~ ., train = train_data, test = test_data, k = best_k_scaled, scale = TR
UE)

# Get the predicted labels for the test data
final_preds <- fitted(final_model)

# Calculate and print the final accuracy on the test data
final_accuracy <- sum(as.character(final_preds) == as.character(test_data$V11)) / nrow(test_dat
a)
print(paste("Final accuracy on test data:", final_accuracy))
```

```
## [1] "Final accuracy on test data: 0.863636363636364"
```

Question 4.1

Describe a situation or problem from your job, everyday life, current events, etc., for which a clustering model would be appropriate. List some (up to 5) predictors that you might use.

1. I have been tracking my spending habits for a few years now. I log monthly expenses such as rent/mortgage, insurance, utility bills, subscription fees, etc. on an excel sheet and create visualizations for myself to analyze. As of right now the clustering process is manual. It'd be neat if a cluster model can identify the purchase types by where the transactions were made without my intervention.

2. Adult friendships are hard to initiate and maintain. Similar to the dating app concept, but for friendships, it'd be neat if there is a social networking app that links you and other young adults based on a few factors. The cluster model can group the app profiles by hobbies, geography, age, and (time) availability.

Question 4.2

The iris data set iris.txt contains 150 data points, each with four predictor variables and one categorical response. The predictors are the width and length of the sepal and petal of flowers and the response is the type of flower. The data is available from the R library datasets and can be accessed with iris once the library is loaded. It is also available at the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Iris (https://archive.ics.uci.edu/ml/datasets/Iris)). The response values are only given to see how well a specific method performed and should not be used to build the model.

Use the R function kmeans to cluster the points as well as possible. Report the best combination of predictors, your suggested value of k, and how well your best clustering predicts flower type.

It looks like the most optimal amount of clusters to have is 3 before the efficacy falls off.

```
#------------------------- Libraries
library(ggplot2)  # Load the ggplot2 library for data visualization

#------------------------- Clean workspace
rm(list = ls())  # Remove all objects from the workspace to ensure a clean start

#------------------------- Load and explore data
# Read the iris data from the specified file
iris_df <- read.table("C://Users//Clair//OneDrive//Documents//GitHub//omsa//ISYE 6501//Homework 2//iris.txt", header = TRUE)

# Display the first few rows of the dataset to understand its structure
head(iris_df)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# Display a summary of the dataset, showing basic statistics for each column
summary(iris_df)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300  
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
##    Species         
##  Length:150        
##  Class :character  
##  Mode  :character  
##                    
##                    
## 
```

```
# Get the dimensions of the dataset (rows x columns)
dim(iris_df)
```

```
## [1] 150   5
```

```
# Print the column names
names(iris_df)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

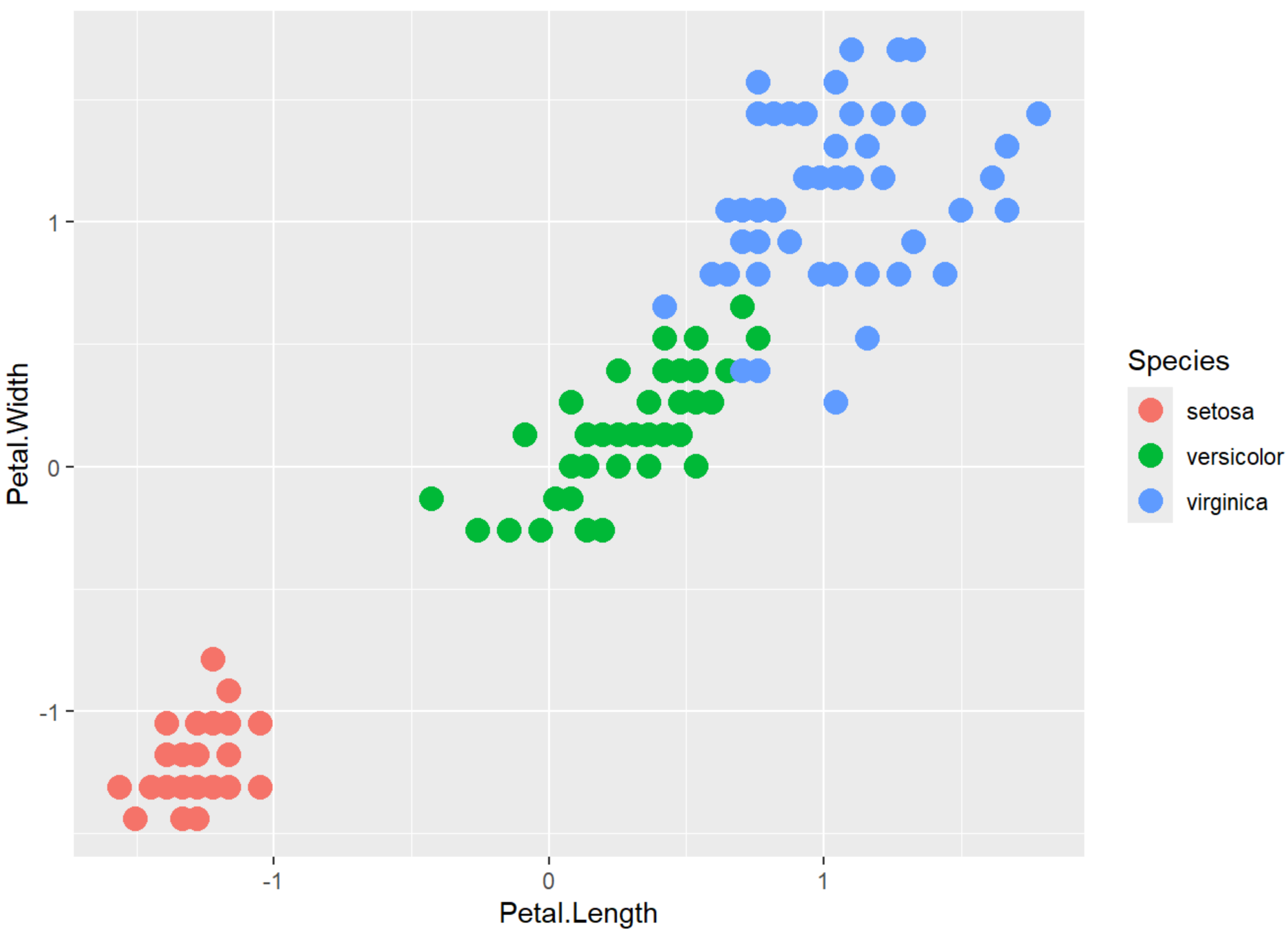
```
# Check for missing values in the dataset
sum(is.na(iris_df))
```

```
## [1] 0
```

```
# Exclude the first column which is an index, not relevant to model
iris_clean_df <- iris_df[,2:5]

# Check the data types of the columns
str(iris_clean_df)
```

```
## 'data.frame':    150 obs. of  4 variables:
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : chr  "setosa" "setosa" "setosa" "setosa" ...
```

```
# Exclude non-numeric columns (ensure you only include numeric columns for scaling)
iris_numeric_df <- iris_clean_df[, sapply(iris_clean_df, is.numeric)]

# Scale the numeric columns
iris_scaled <- scale(iris_numeric_df)

# Convert the scaled data to a data frame and add the Species column back
# look at [4] of the references
iris_scaled_df <- as.data.frame(iris_scaled)
iris_scaled_df$Species <- iris_clean_df$Species

# Plotting the scaled data
# look at [5] of the references
ggplot(iris_scaled_df, aes(Petal.Length, Petal.Width)) + geom_point(aes(col=Species), size=4)
```




```
# Display the frequency table for the Species column
table(iris_scaled_df$Species)
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

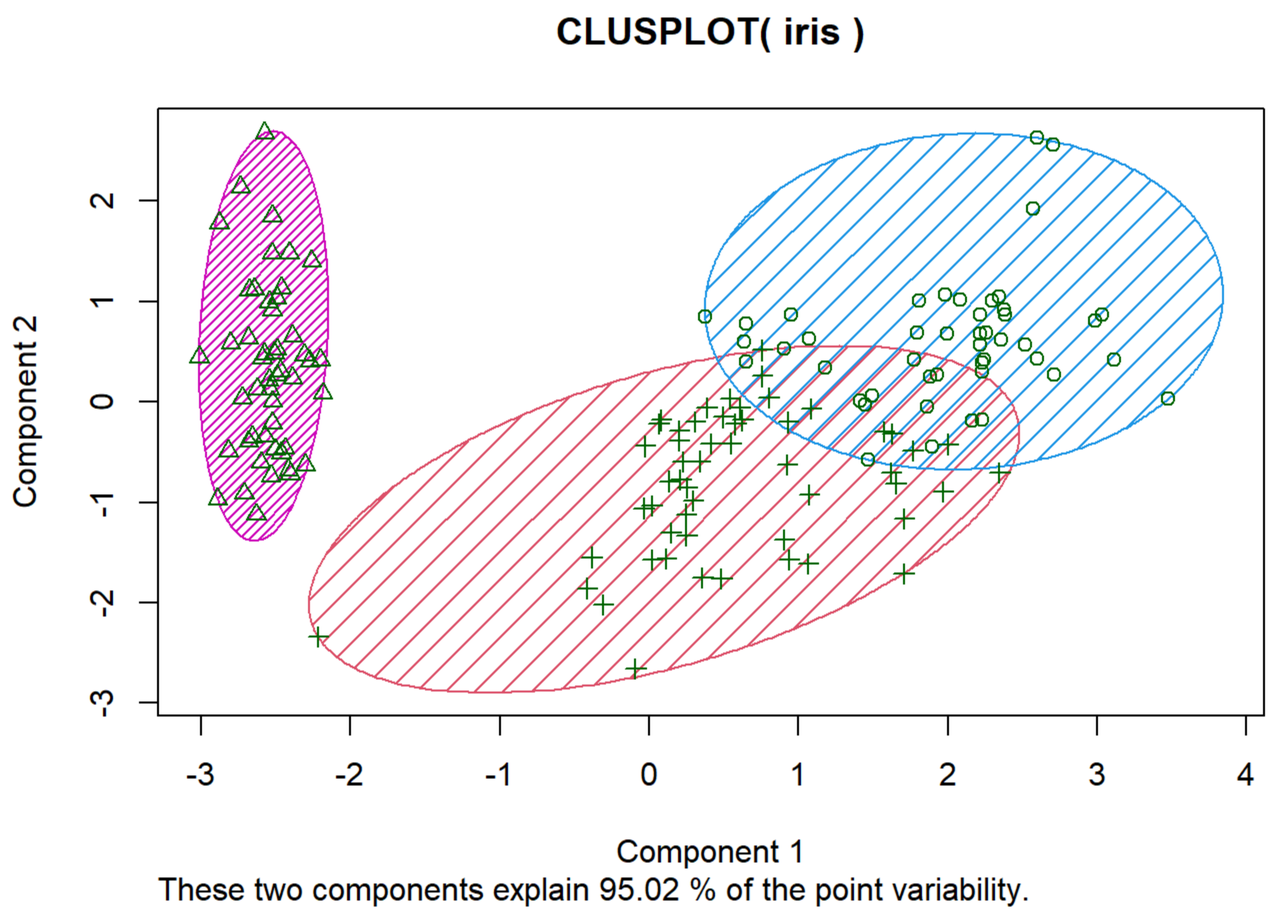
```
#------------------------- Modeling for singular k

# Setting the seed for reproducibility
# look at [5] of the references
set.seed(101)
iris_cluster <- kmeans(iris_scaled, centers=3, nstart=20)
print(iris_cluster)
```

```
## K-means clustering with 3 clusters of sizes 46, 49, 55
##
## Cluster means:
##   Sepal.Width Petal.Length Petal.Width
## 1   0.1178396    1.0015797   1.0732321
## 2   0.9032290   -1.2987572  -1.2521493
## 3  -0.9032517    0.3193898   0.2179389
##
## Clustering vector:
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
##   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2   2
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   2   3   2   2   2   2   2   2   2   2   1   1   1   3   3   3   1   3   3   3
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##   3   3   3   3   3   3   3   3   3   3   1   3   3   3   3   3   3   1   3   3
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
##   3   3   3   3   3   1   1   3   3   3   3   3   3   3   3   3   3   3   3   3
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   1   3   1   1   1   1   3   1   3   1   1   3   1   3   1   1   1   1   1   3
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
##   1   1   1   3   1   1   3   1   1   1   1   1   1   3   3   1   1   1   1   1
## 141 142 143 144 145 146 147 148 149 150
##   1   1   3   1   1   1   3   1   1   1
##
## Within cluster sum of squares by cluster:
## [1] 26.97503 31.62385 36.08438
##  (between_SS / total_SS =  78.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

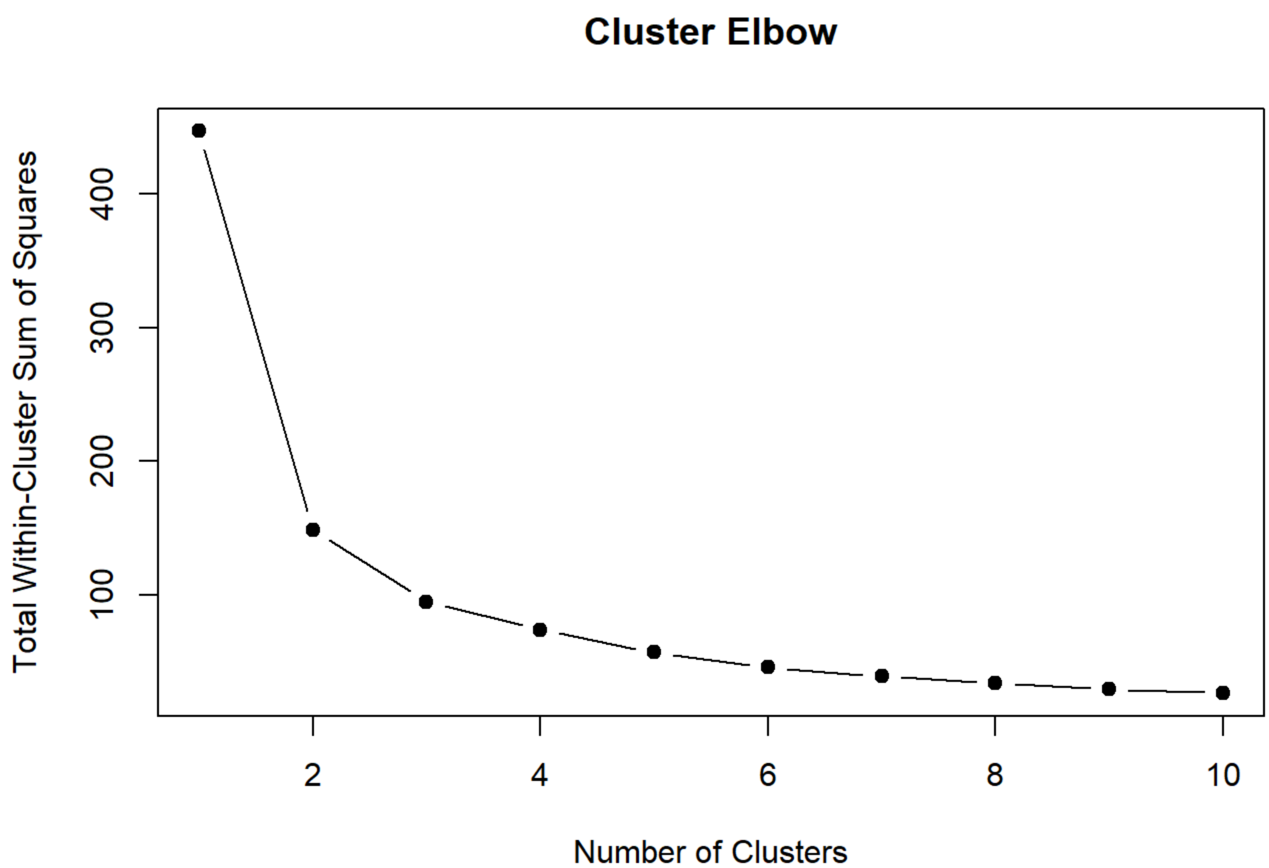
```
#------------------------- Modeling for multiple ks
# Setting the seed for reproducibility
# look at [5] of the references
set.seed(101)

k2 <- kmeans(iris_scaled, centers=2, nstart=20)
k3 <- kmeans(iris_scaled, centers=3, nstart=20)
k4 <- kmeans(iris_scaled, centers=4, nstart=20)
k5 <- kmeans(iris_scaled, centers=5, nstart=20)
k6 <- kmeans(iris_scaled, centers=6, nstart=20)
k7 <- kmeans(iris_scaled, centers=6, nstart=20)
k8 <- kmeans(iris_scaled, centers=6, nstart=20)
k9 <- kmeans(iris_scaled, centers=6, nstart=20)
k10 <- kmeans(iris_scaled, centers=6, nstart=20)


# Table output of predicted clusters with the original data
# look at [5] of the references
table(iris_cluster$cluster, iris_scaled_df$Species)
```

```
##
##     setosa versicolor virginica
##   1      0          8        38
##   2     49          0         0
##   3      1         42        12
```

```
#------------------------- Visualization
# look at [5] of the references
library(cluster)
clusplot(iris, iris_cluster$cluster, color=T, shade=T, labels=0, lines=0)
```




```
# initialize vector
# look at [5] of the references
tot.withinss <- numeric(length=10)

# Loop thorugh number of Ks
# look at [5] of the references
for (i in 1:10) {
  iris_cluster <- iris_cluster <- kmeans(iris_scaled, centers=i, nstart=20)
  tot.withinss[i] <- iris_cluster$tot.withinss
}

# Ensure the first element is initialized
# look at [5] of the references
tot.withinss[1] <- kmeans(iris_scaled, centers=1, nstart=5)$tot.withinss

# Graph elbow plot
# look at [5] of the references
plot(1:10, tot.withinss, type="b", pch=19, xlab="Number of Clusters", ylab="Total Within-Cluster
Sum of Squares", main="Cluster Elbow")
```



References: [1] in. (2019, May 22). Randomly Sampling Rows in R. Learningtree.com. https://www.learningtree.com/blog/randomly-sampling-rows-r/ (https://www.learningtree.com/blog/randomly-sampling-rows-r/) [2] Calculate the Floor and Ceiling values in R Programming - floor() and ceiling() Function. (2020, May 30). GeeksforGeeks. https://www.geeksforgeeks.org/calculate-the-floor-and-ceiling-values-in-r-programming-floor-and-ceiling-function/ (https://www.geeksforgeeks.org/calculate-the-floor-and-ceiling-values-in-r-programming-floor-and-ceiling-function/) [3] Why my test accuracy higher than validation accuracy? (2023). Mathworks.com. https://www.mathworks.com/matlabcentral/answers/1954939-why-my-test-accuracy-higher-than-validation-accuracy/?s_tid=ans_lp_feed_leaf (https://www.mathworks.com/matlabcentral/answers/1954939-why-my-test-accuracy-higher-than-validation-accuracy/?s_tid=ans_lp_feed_leaf) [4] Zach. (2021, January 27). How to Add a Column to a Data Frame in R (With Examples). Statology. https://www.statology.org/r-add-a-column-to-dataframe/ (https://www.statology.org/r-add-a-column-to-dataframe/) [5] Ramos Lorenzo, C. (2019, June 14). RPubs - K-means clustering with iris dataset in R. Rpubs.com. https://rpubs.com/MrCristianrl/504935 (https://rpubs.com/MrCristianrl/504935)