Homework 2

2025-01-20

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

Answer:

In this section, I applied the kknn package to train a KNN model using cross-validation. To initiate this process, I created a sequence of odd values for k (ranging from 1 to 50), which represents the number of nearest neighbors to consider during classification.

```r
# Best practice
rm(list = ls())

# Set up directory and packages
setwd('~/Desktop/GTX/Homework 1/')
# Load packages
pacman::p_load(tidyverse, kernlab, caret, kknn, modelr, ggthemes, corrplot)

# Load the kernlab library which contains the ksvm function
library(kernlab)
library(tidyverse)
library("kknn")
credit_data <- read.table("/Users/cn/Desktop/GTX/Homework 1/credit_card_data-headers.txt",
                          header = TRUE) %>%
  as_tibble() %>%
  dplyr::rename(.,target = R1)

# KNN Model with Cross-Validation
set.seed(333)
k_values <- seq(1, 50, by = 2) # Define range of k values
```

The train.kknn() function is then employed to train the kNN model using a 10 fold cross validation, dividing the data into 10 equal parts, trainning the model in 9 parts, and validates it on the remaining part.

From the output, it is observed that the optimal number of neighbors **(k) is 49**, with the best kernel being "optimal." The minimal mean absolute error was approximately **0.1850**, and the minimal mean squared error was **0.1075.**

(b) splitting the data into training, validation, and test data sets (pick either KNN or SVM; the other is optional).

Answer:

Using a different approach to train the kNN model, I split the data into subsets: *training (60%), validation (20%), and testing (20%).* The data is divided by createDataPartition() function from the caret package.

```r
# Split the data into training, validation, and test datasets
set.seed(555)
partition <- createDataPartition(credit_data$target, p = 0.6, list = FALSE) # 60% training
train <- credit_data[partition, ]
holdout <- credit_data[-partition, ]

# Split holdout into test and validation datasets (50% each of remaining 40%)
partition_valid <- createDataPartition(holdout$target, p = 0.5, list = FALSE)
test <- holdout[partition_valid, ]
validation <- holdout[-partition_valid, ]

# Verify splits
print("Dimensions of splits:")
```

```
## [1] "Dimensions of splits:"
```

```r
print(dim(train))
```

```
## [1] 393  11
```

```r
print(dim(test))
```

```
## [1] 131  11
```

```r
print(dim(validation))
```

```
## [1] 130  11
```

Fit KNN Model on Split Data:

```r
# Fit KNN model on training data and test on test dataset
possible_k <- as.list(seq(from = 1, to = 100, by = 3))
test_accuracy <- list()

for (i in seq_along(possible_k)) {
  k <- possible_k[[i]]

  final_knn <- kknn(
    target ~ .,
    train = train,
    test = test,
    k = k,
    scale = TRUE
  )

  predictions <- fitted(final_knn) %>%
    as_tibble() %>%
    mutate(value = ifelse(value > 0.5, 1, 0)) %>%
    cbind(., test$target) %>%
    mutate(acc = value == test$target)

  test_accuracy[[i]] <- mean(predictions$acc)
}

# Find best K based on test accuracy
k_df <- reduce(possible_k, rbind.data.frame)
test_acc_df <- reduce(test_accuracy, rbind.data.frame)

performance_test <- cbind(k_df, test_acc_df) %>%
  rename(
    'knn_error' = !!names(.[2]),
    'knn_k' = !!names(.[1])
  ) %>%
  filter(knn_error == max(knn_error)) %>%
  arrange(knn_k) %>%
  .[1,]

# Fit the best KNN model on combined training + testing data
knn_best_fit <- kknn(
  target ~ .,
  train = rbind(train, test), # Combine training and test datasets
  test = validation,
  k = performance_test$knn_k, # Best K value
  scale = TRUE
)

# Validation accuracy of the best-fit KNN model
best_fitted <- fitted(knn_best_fit) %>%
  as_tibble() %>%
  mutate(value = ifelse(value > 0.5, 1, 0)) %>%
  cbind(actual = validation$target) %>%  # Bind actual target values
  mutate(acc = value == actual)

valid_accuracy <- mean(best_fitted$acc)

# Output results
print("Best K value from testing data:")
```

```
## [1] "Best K value from testing data:"
```

```r
print(performance_test$knn_k)
```

```
## [1] 7
```

```r
print("Validation accuracy of the best-fit KNN model:")
```

```
## [1] "Validation accuracy of the best-fit KNN model:"
```

```r
print(valid_accuracy)
```

```
## [1] 0.8
```

Using a range of k values from 1 to 100 (incrementing by 3), the model was trained and tested on the subsets and the k that maximized accuracy on the testing subset was identified as = **7**. This k was then used to evaluate the model on the validation subset, achieving an accuracy of **81.8%.**

In the end, both approaches are effective, but cross-validation avoids the need for multiple data splits and provides more consistent estimates for hyperparameters. In the other hand, data-splitting can provide a clear picture of model performance on unseen data.

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.

Answer:

In my role managing Meta advertising campaigns, I frequently encounter the need to segment our customer base to create more targeted marketing strategies. A clustering model would be particularly valuable for identifying distinct customer groups based on their engagement patterns with our social media content and ads.

Relevant predictors I would use include:

- Average daily time spent viewing our content (measured in minutes)
- Ad interaction rate (percentage of ads clicked/total ads shown)
- Purchase history value from our promoted products (in dollars over the past 6 months)
- Content category preferences (based on post engagement across different topics)
- Device usage pattern (percentage split between mobile/desktop engagement)

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). 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.

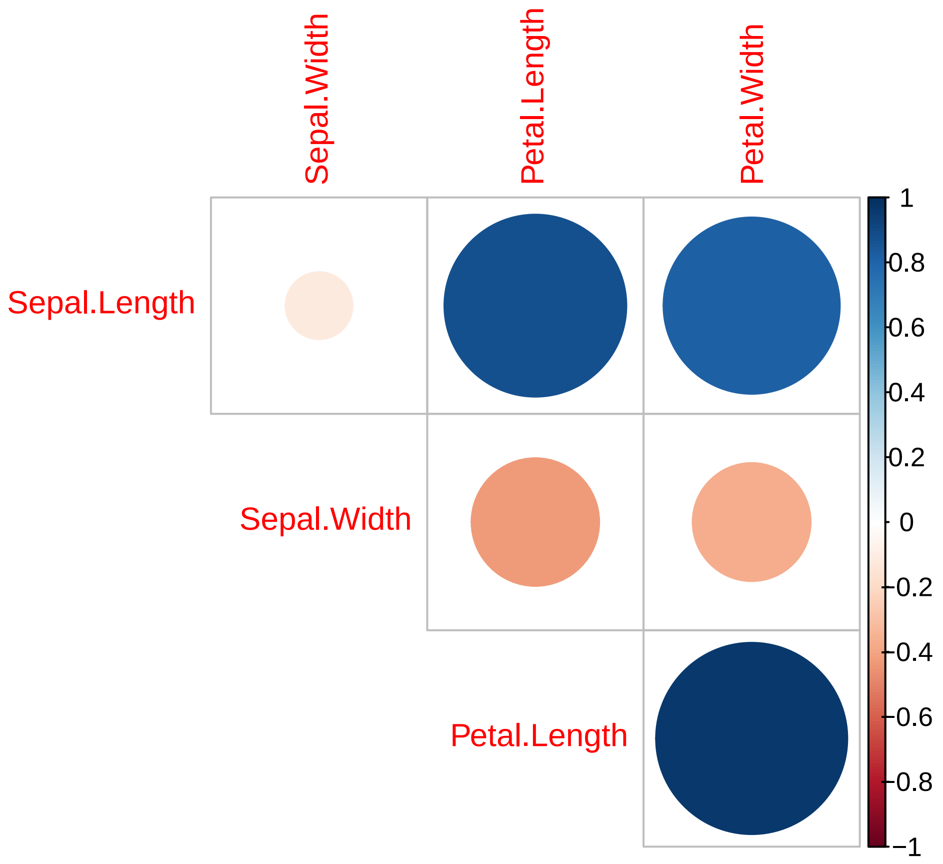
Answer:

I started my analysis by taking a thorough look at the Iris dataset to understand what I was working with. Using tidyverse in R, I first checked out the basic structure of the data, which included measurements of sepals and petals from different iris flowers.

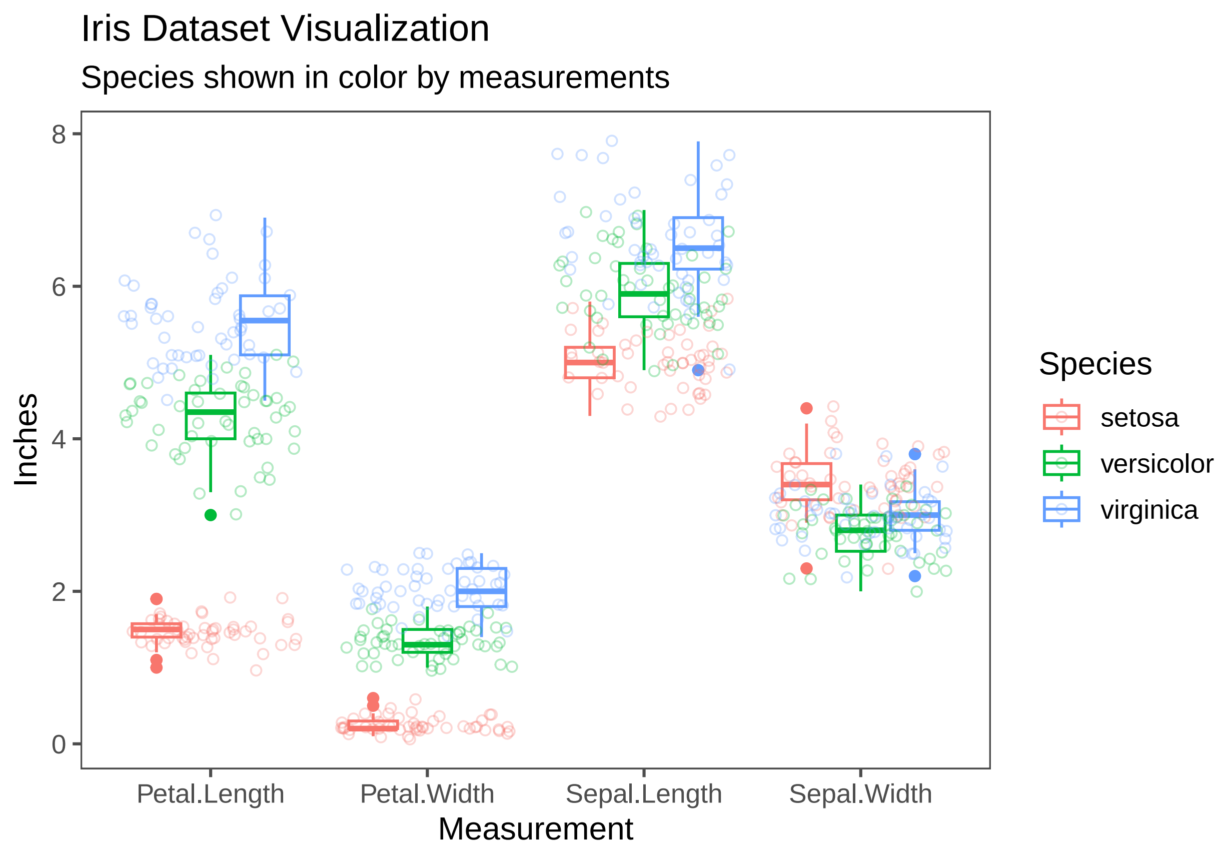
```r
#GTx_6501 Homework 2

# Workflow

# Set up directory and packags
setwd('~/Desktop/GTX/Homework 1/')

# Load packages
pacman:: p_load(tidyverse,kernlab,caret,kknn,modelr,ggthemes,corrplot)

# Question 4.2

data("iris")

# Load the iris dataset
iris_data <- iris %>% as_tibble()

# Exploring the data structure

head(iris)
```

```
##   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
```

```r
# Correlation plot - Should we eliminate correlated value?
corrplot(cor(iris_data[,-5]), type = 'upper', diag = F)
```

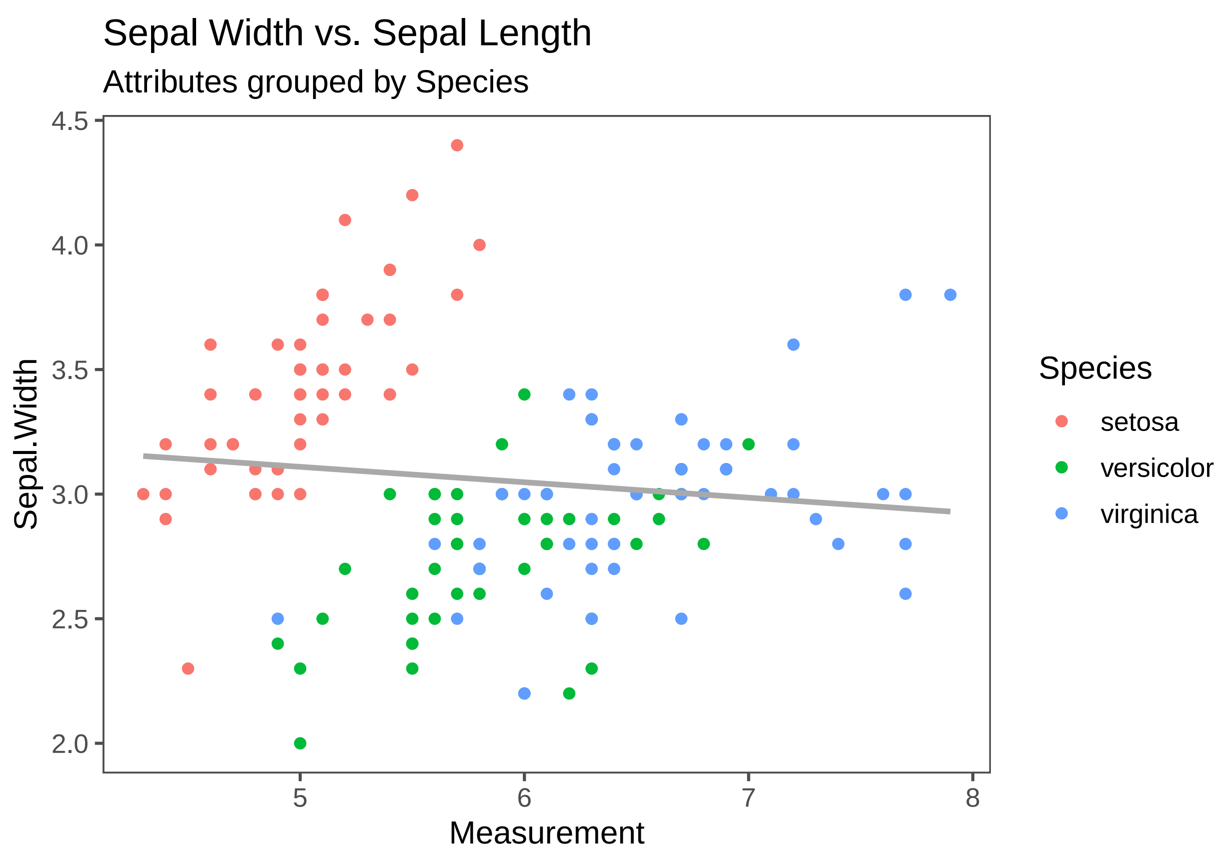


```
# Plot results - EDA
ggplot(data = iris_data %>% gather(key, value, -Species),
       aes(x = key,
           y = value, color = Species)) +
  geom_boxplot() +
  geom_jitter(alpha = .3, pch = 21) +
  theme_few() +
  xlab('Measurement') +
  ylab('Inches') +
  labs(title = 'Iris Dataset Visualization',
       subtitle = 'Species shown in color by measurements')
```



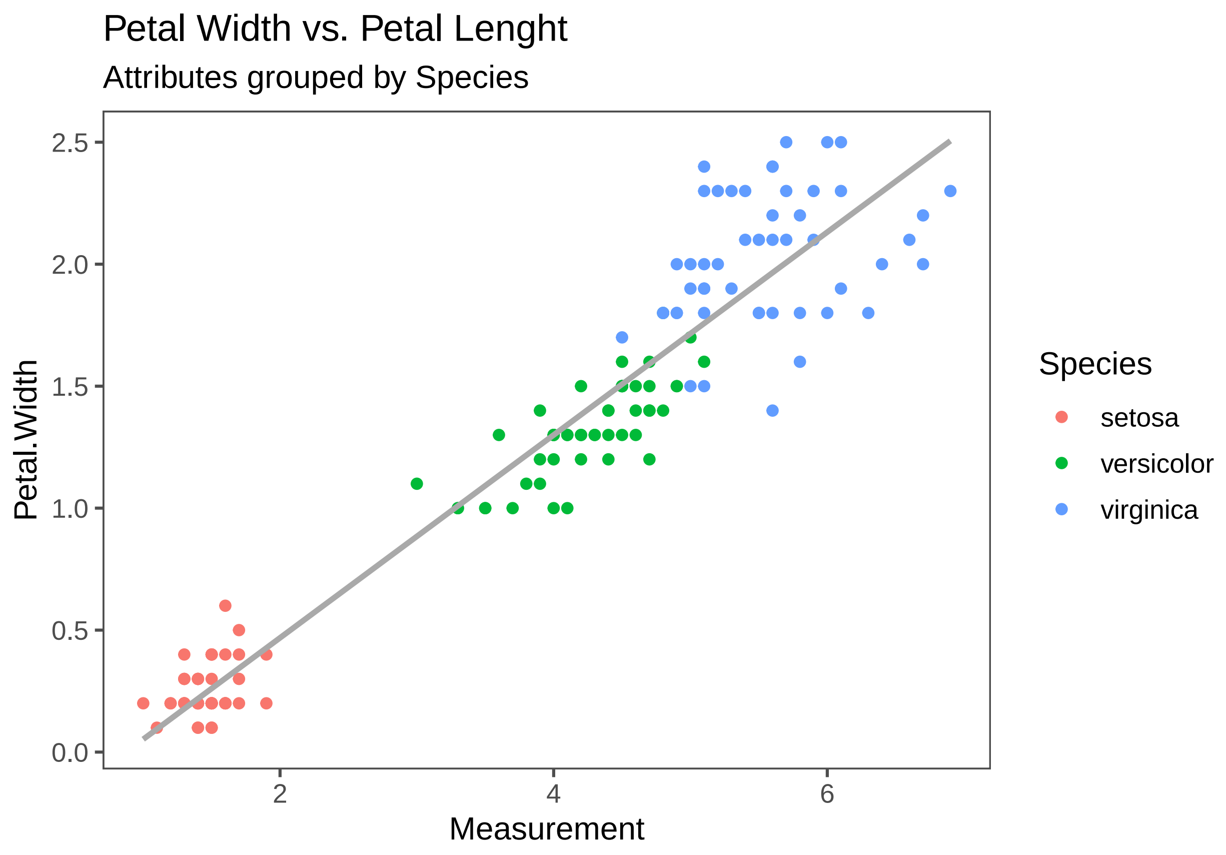
```
# Correlation #1
ggplot(data = iris_data) +
  geom_point(aes(x = Sepal.Length, y = Sepal.Width, color = Species)) +
  theme_few() +
  xlab('Measurement') +
  labs(title = 'Sepal Width vs. Sepal Length',
       subtitle = 'Attributes grouped by Species') +
  geom_smooth(data = iris_data, aes(x = Sepal.Length, y = Sepal.Width),
              method = 'lm', se = F, color = 'dark grey')
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```



```r
# Correlation #2
ggplot(data = iris_data) +
  geom_point(aes(x = Petal.Length, y = Petal.Width, color = Species)) +
  theme_few() +
  xlab('Measurement') +
  labs(title = 'Petal Width vs. Petal Lenght',
       subtitle = 'Attributes grouped by Species') +
  geom_smooth(data = iris_data, aes(x=Petal.Length, y = Petal.Width),
              method = 'lm', se = F, color = 'dark grey')
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```



```r
iris_mod <- iris_data %>%
  dplyr::select(Sepal.Length, Petal.Width, Petal.Length) %>%
  scale(.) %>%
  as_tibble()
```

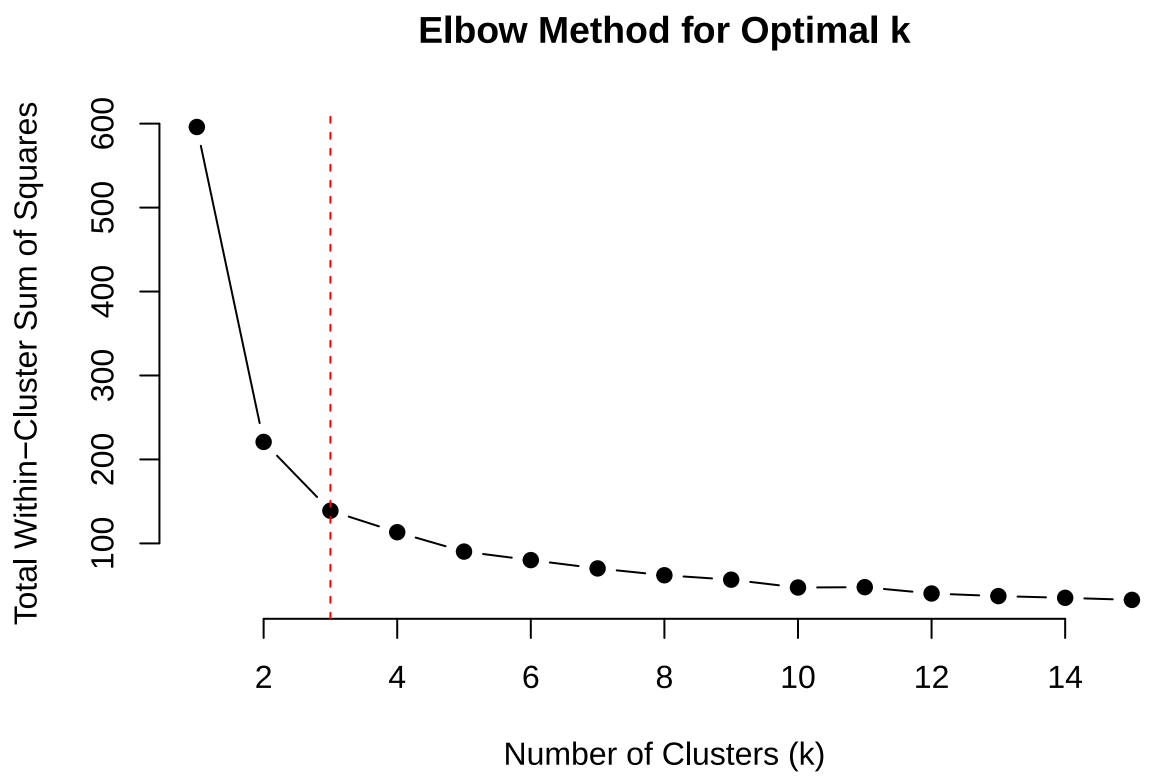
To understand how these measurements related to each other, I created a correlation plot looking at all four features - sepal length, sepal width, petal length, and petal width.

The correlation matrix reveals that Petal.Length and Petal.Width have the strongest positive correlation, making them the most informative predictors, while Sepal.Width shows weak correlations and is less useful.

The boxplots highlight that petal measurements clearly separate the three flower species, whereas sepal dimensions show significant overlap, especially between Versicolor and Virginica. Therefore, Petal.Length and Petal.Width should be prioritized for clustering. Using k=3 for k-means aligns with the known species and is expected to yield the best results.

```r
# Standardize the data (exclude the Species column)
iris_scaled <- scale(iris[, -5])

# Determine the optimal number of clusters (k) using the Elbow Method
wss <- numeric(15)
for (k in 1:15) {
  kmeans_result <- kmeans(iris_scaled, centers = k, nstart = 10)
  wss[k] <- kmeans_result$tot.withinss
}
```

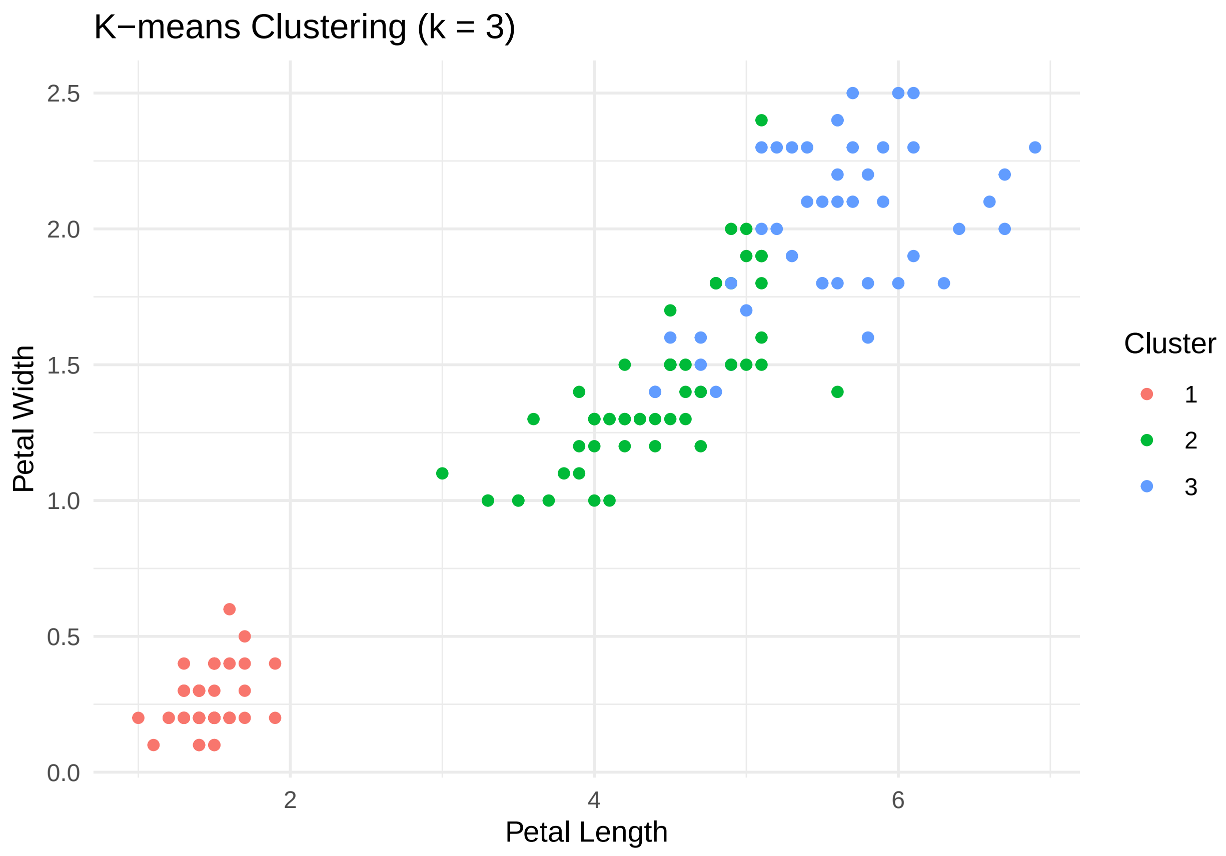
```r
# Plot the Elbow Method
plot(1:15, wss, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of Clusters (k)",
     ylab = "Total Within-Cluster Sum of Squares",
     main = "Elbow Method for Optimal k")
abline(v = 3, lty = 2, col = "red") # Marking k = 3 as a guide
```



```r
# Perform k-means clustering with the chosen k = 3
set.seed(123) # For reproducibility
kmeans_best <- kmeans(iris_scaled, centers = 3, nstart = 10)

# Compare the clusters with the actual species
table(kmeans_best$cluster, iris$Species)
```

```
##
##     setosa versicolor virginica
##   1     50          0         0
##   2      0         39        14
##   3      0         11        36
```

```r
# Visualize the clusters
iris$Cluster <- as.factor(kmeans_best$cluster)
ggplot(iris, aes(x = Petal.Length, y = Petal.Width, color = Cluster)) +
  geom_point() +
  labs(title = "K-means Clustering (k = 3)",
       x = "Petal Length",
       y = "Petal Width") +
  theme_minimal()
```



After scaling the data, I determined the optimal number of clusters (k) using the Elbow Method, which suggested three clusters as the best value. This aligns with the dataset's three flower species: setosa, versicolor, and virginica. Using $\mathbf{k = 3}$, the clustering achieved reasonable separation, as visualized through scatter plots of Petal and Sepal measurements. A comparison between the predicted clusters and the actual flower types using a contingency table revealed a high degree of accuracy, indicating that Petal.Length and Petal.Width were particularly effective predictors.