Homework_4

Laurentiu Mandocescu

2025-03-07

Load Libraries and Data

```
# Load necessary libraries
library(tidyverse)
```

```
# Load the datasets
red_wine <- read.csv("winequality-red.csv", sep = ";")
white_wine <- read.csv("winequality-white.csv", sep = ";")

# Display first few rows of each dataset
head(red_wine)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.4             0.70        0.00            1.9     0.076
## 2           7.8             0.88        0.00            2.6     0.098
## 3           7.8             0.76        0.04            2.3     0.092
## 4          11.2             0.28        0.56            1.9     0.075
## 5           7.4             0.70        0.00            1.9     0.076
## 6           7.4             0.66        0.00            1.8     0.075
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  11                   34  0.9978 3.51      0.56     9.4
## 2                  25                   67  0.9968 3.20      0.68     9.8
## 3                  15                   54  0.9970 3.26      0.65     9.8
## 4                  17                   60  0.9980 3.16      0.58     9.8
## 5                  11                   34  0.9978 3.51      0.56     9.4
## 6                  13                   40  0.9978 3.51      0.56     9.4
##   quality
## 1       5
## 2       5
## 3       5
## 4       6
## 5       5
## 6       5
```

```
head(white_wine)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.0             0.27        0.36           20.7     0.045
## 2           6.3             0.30        0.34            1.6     0.049
```

```
## 3           8.1             0.28        0.40            6.9     0.050
## 4           7.2             0.23        0.32            8.5     0.058
## 5           7.2             0.23        0.32            8.5     0.058
## 6           8.1             0.28        0.40            6.9     0.050
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  45                  170  1.0010 3.00      0.45     8.8
## 2                  14                  132  0.9940 3.30      0.49     9.5
## 3                  30                   97  0.9951 3.26      0.44    10.1
## 4                  47                  186  0.9956 3.19      0.40     9.9
## 5                  47                  186  0.9956 3.19      0.40     9.9
## 6                  30                   97  0.9951 3.26      0.44    10.1
##   quality
## 1       6
## 2       6
## 3       6
## 4       6
## 5       6
## 6       6
```

Data Preparation

Problem 1 a)

```
# Add type column
red_wine$type <- "red"
white_wine$type <- "white"

# Merge both datasets into one
wine_quality <- bind_rows(red_wine, white_wine)

# Check structure of merged dataset
str(wine_quality)
```

```
## 'data.frame':    6497 obs. of  13 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
##  $ type                : chr  "red" "red" "red" "red" ...
```

```
# Display first few rows of merged dataset
head(wine_quality)
```

```
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1           7.4             0.70        0.00            1.9     0.076
## 2           7.8             0.88        0.00            2.6     0.098
## 3           7.8             0.76        0.04            2.3     0.092
## 4          11.2             0.28        0.56            1.9     0.075
## 5           7.4             0.70        0.00            1.9     0.076
## 6           7.4             0.66        0.00            1.8     0.075
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  11                   34  0.9978 3.51      0.56     9.4
## 2                  25                   67  0.9968 3.20      0.68     9.8
## 3                  15                   54  0.9970 3.26      0.65     9.8
## 4                  17                   60  0.9980 3.16      0.58     9.8
## 5                  11                   34  0.9978 3.51      0.56     9.4
## 6                  13                   40  0.9978 3.51      0.56     9.4
##   quality type
## 1       5  red
## 2       5  red
## 3       5  red
## 4       6  red
## 5       5  red
## 6       5  red
```

Ensure Correct Data Types

```r
# Convert 'type' to a factor
wine_quality$type <- as.factor(wine_quality$type)

# Check data structure again
str(wine_quality)
```

```
## 'data.frame':    6497 obs. of  13 variables:
##  $ fixed.acidity       : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5 ...
##  $ volatile.acidity    : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58 0.5 ...
##  $ citric.acid         : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
##  $ residual.sugar      : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
##  $ chlorides           : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069 0.065 0.073 0.071 ...
##  $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
##  $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
##  $ density             : num  0.998 0.997 0.997 0.998 0.998 ...
##  $ pH                  : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36 3.35 ...
##  $ sulphates           : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.57 0.8 ...
##  $ alcohol             : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5 ...
##  $ quality             : int  5 5 5 6 5 5 5 7 7 5 ...
##  $ type                : Factor w/ 2 levels "red","white": 1 1 1 1 1 1 1 1 1 1 ...
```

Problem 1 b)

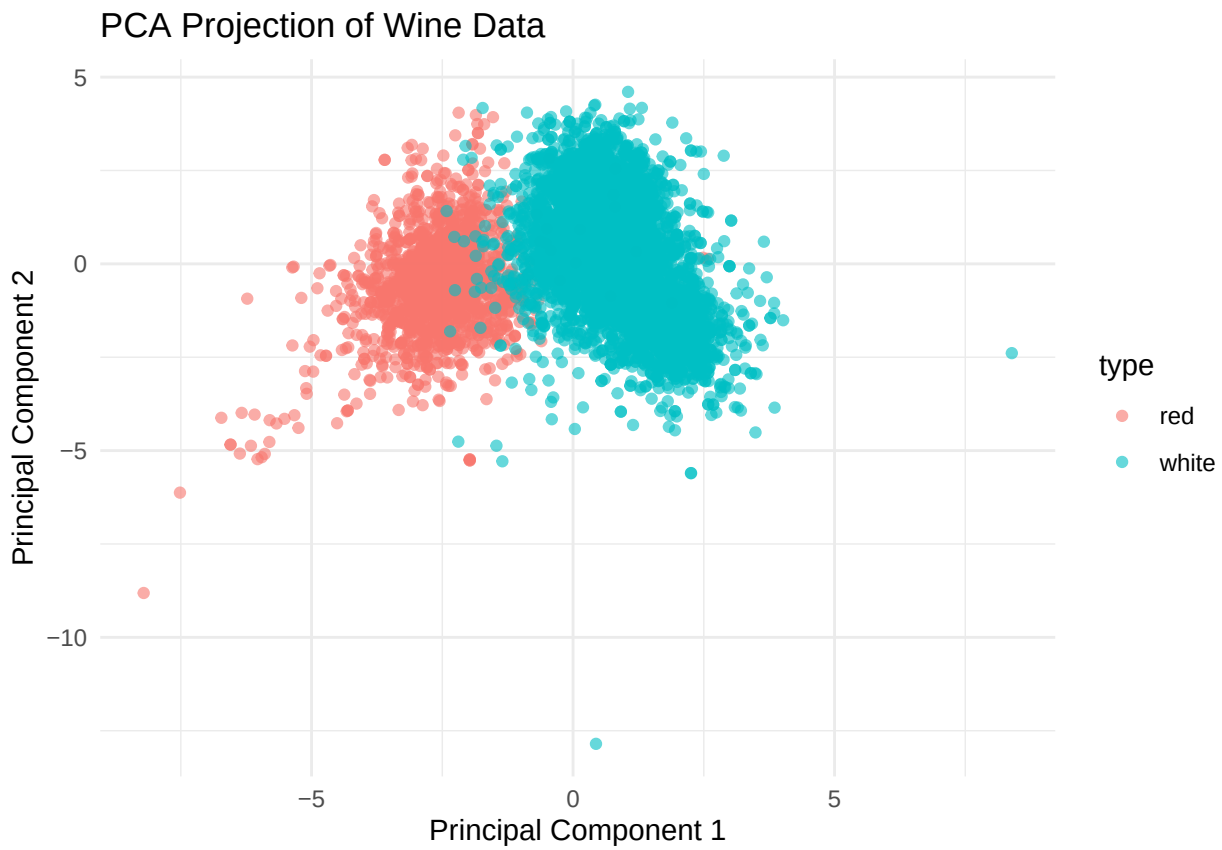
```r
# Load required libraries
library(ggplot2)
library(FactoMineR)
```

```
## Warning: package 'FactoMineR' was built under R version 4.4.3
```

```
library(factoextra)
```

```
## Warning: package 'factoextra' was built under R version 4.4.3

## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```
# Perform PCA (excluding 'type' since it's categorical)
pca_model <- prcomp(wine_quality[, -13], center = TRUE, scale. = TRUE)

# Extract PCA results for first two principal components
pca_data <- as.data.frame(pca_model$x[, 1:2])

# Add back the wine type column for coloring
pca_data$type <- wine_quality$type

# Scatterplot of PCA results
ggplot(pca_data, aes(x = PC1, y = PC2, color = type)) +
  geom_point(alpha = 0.6) +
  labs(title = "PCA Projection of Wine Data", x = "Principal Component 1", y = "Principal Component 2") +
  theme_minimal()
```



Problem 1 c)

Interpretation of the PCA Plot

The red points represent red wines, and the blue points represent white wines. There is a clear separation between the two types along Principal Component 1 (PC1), meaning that PC1 captures most of the variance that distinguishes red and white wines.

Since the two clusters are reasonably well-separated but still have some overlap, SVM might perform well because it excels at separating classes with a clear boundary. Decision Trees might also work well since they create rule-based partitions. kNN could struggle if the decision boundary is too complex or if there are many overlapping points.

Problme 1 d)

Train and Compare Classifiers

```
# Load required libraries
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.4.2
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(class)   # For kNN
library(rpart)   # For Decision Tree
library(e1071)   # For SVM
```

```
## Warning: package 'e1071' was built under R version 4.4.2
```

```
# Fix column names for rpart
colnames(wine_quality) <- make.names(colnames(wine_quality))

# Split data into training (80%) and testing (20%)
set.seed(123)  # For reproducibility
train_index <- createDataPartition(wine_quality$type, p = 0.8, list = FALSE)

train_data <- wine_quality[train_index, ]
test_data  <- wine_quality[-train_index, ]

# Remove labels for kNN (kNN requires separate feature & label datasets)
train_features <- train_data[, -13]
test_features  <- test_data[, -13]
```

```r
train_labels <- train_data$type
test_labels  <- test_data$type
```

```r
# Tune k for kNN
set.seed(123)
k_values <- seq(1, 21, by = 2)  # Try odd values from 1 to 21
knn_results <- data.frame(k = k_values, accuracy = NA)

for (i in seq_along(k_values)) {
  knn_pred <- knn(train_features, test_features, train_labels, k = k_values[i])
  knn_results$accuracy[i] <- mean(knn_pred == test_labels)
}

# Best k
best_k <- knn_results$k[which.max(knn_results$accuracy)]
best_k
```

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

```r
# Train decision tree
dt_model <- rpart(type ~ ., data = train_data, method = "class")

# Predict
dt_pred <- predict(dt_model, test_data, type = "class")

# Accuracy
dt_accuracy <- mean(dt_pred == test_labels)
dt_accuracy
```

```
## [1] 0.9799692
```

```r
# Tune SVM (tuning C)
svm_tune <- tune(svm, type ~ ., data = train_data, kernel = "linear",
                 ranges = list(cost = c(0.1, 1, 10, 100)))

# Best model
svm_model <- svm_tune$best.model

# Predict
svm_pred <- predict(svm_model, test_data)

# Accuracy
svm_accuracy <- mean(svm_pred == test_labels)
svm_accuracy
```

```
## [1] 0.9946071
```

```r
# Compare all models
results <- data.frame(
  Model = c("kNN", "Decision Tree", "SVM"),
```

```
  Accuracy = c(max(knn_results$accuracy), dt_accuracy, svm_accuracy)
)

# Print results
print(results)
```

```
##           Model  Accuracy
## 1           kNN 0.9491525
## 2 Decision Tree 0.9799692
## 3           SVM 0.9946071
```

kNN (best k = 1): 94.92% accuracy Decision Tree: 97.99% accuracy SVM (best C from tuning): 99.46% accuracy

SVM performed the best with almost perfect accuracy (99.46%). This aligns with my expectation from the PCA plot, where the data was well-separated but had some overlap. SVM is great for such cases because it finds the best decision boundary.

Decision Trees also performed well (97.99% accuracy), indicating that the features contain clear rules that separate red and white wines.

kNN had the lowest accuracy (94.92%). This is expected because:

kNN is sensitive to noise and local variations. Choosing k = 1 means it classifies based on the closest single neighbor, which makes it prone to overfitting.

Problem 1 e)

Visualizing Classifier Predictions on PCA

```
# Extract PCA results only for the test set
pca_test_data <- as.data.frame(pca_model$x[-train_index, 1:2])  # Extract PC1 & PC2 for test data
pca_test_data$type <- test_labels  # Add true labels

# Add classifier predictions
pca_test_data$knn_pred <- knn(train_features, test_features, train_labels, k = best_k)
pca_test_data$dt_pred <- predict(dt_model, test_data, type = "class")
pca_test_data$svm_pred <- predict(svm_model, test_data)

# Function to plot PCA with classifier labels
plot_pca <- function(pred_column, title) {
  ggplot(pca_test_data, aes(x = PC1, y = PC2, color = pred_column)) +
    geom_point(alpha = 0.6) +
    labs(title = title, x = "Principal Component 1", y = "Principal Component 2") +
    theme_minimal()
}

# PCA Scatterplots with True Labels + Model Predictions
p1 <- plot_pca(pca_test_data$type, "True Wine Type")
p2 <- plot_pca(pca_test_data$knn_pred, "kNN Prediction")
p3 <- plot_pca(pca_test_data$dt_pred, "Decision Tree Prediction")
p4 <- plot_pca(pca_test_data$svm_pred, "SVM Prediction")
```

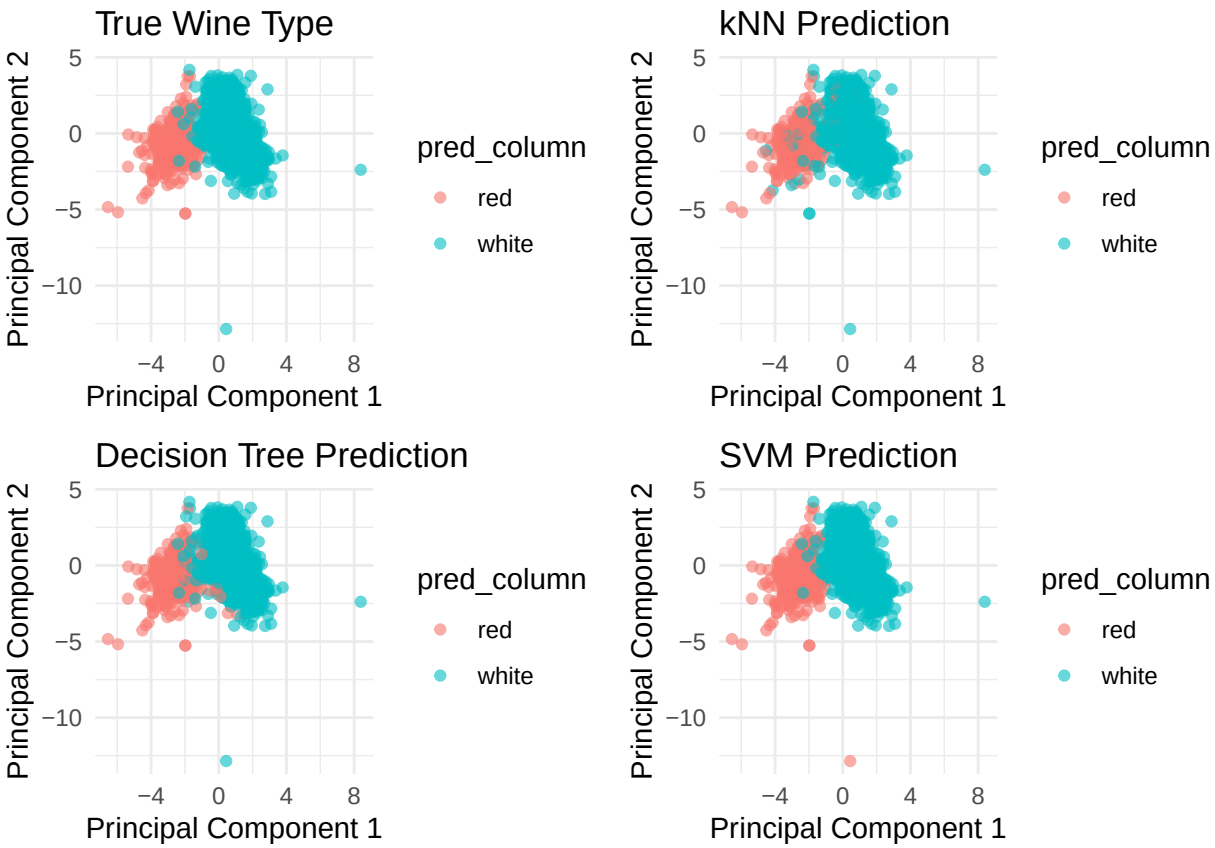
```r
# Show all plots
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.4.3
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```r
grid.arrange(p1, p2, p3, p4, ncol = 2)
```



True Wine Type (Top-Left)

This is the original ground truth. The separation between red and white wines is mostly along PC1. Some overlap exists between red and white wines.

###kNN Prediction (Top-Right)

kNN captures the general structure but has some misclassifications, especially near the overlapping region. Since I used k = 1, it is more sensitive to local noise, making it slightly more prone to errors. Misclassified points (incorrectly colored) are more scattered compared to Decision Trees and SVM.

##Decision Tree Prediction (Bottom-Left)

Performs better than kNN in terms of matching the true wine type. Decision Trees create clear boundaries for classification, leading to fewer misclassifications. Some boundary errors still exist in high-overlap regions.

##SVM Prediction (Bottom-Right)

SVM performs the best, closely resembling the true wine type plot. It finds a linear decision boundary that maximizes separation. Minimal misclassifications, confirming why SVM had the highest accuracy (99.46%).

Problem 2 a)

```
# Load necessary libraries
library(tidyverse)
library(modelr)  # For working with categorical data
library(caret)   # For dummy variables

# Load the Sacramento dataset
data("Sacramento", package = "modeldata")

# Display structure of the dataset
str(Sacramento)
```

```
## tibble [932 x 9] (S3: tbl_df/tbl/data.frame)
##  $ city     : Factor w/ 37 levels "ANTELOPE","AUBURN",..: 34 34 34 34 34 34 34 34 29 31 ...
##  $ zip      : Factor w/ 68 levels "z95603","z95608",..: 64 52 44 44 53 65 66 49 24 25 ...
##  $ beds     : int [1:932] 2 3 2 2 2 3 3 3 2 3 ...
##  $ baths    : num [1:932] 1 1 1 1 1 1 2 1 2 2 ...
##  $ sqft     : int [1:932] 836 1167 796 852 797 1122 1104 1177 941 1146 ...
##  $ type     : Factor w/ 3 levels "Condo","Multi_Family",..: 3 3 3 3 3 1 3 3 1 3 ...
##  $ price    : int [1:932] 59222 68212 68880 69307 81900 89921 90895 91002 94905 98937 ...
##  $ latitude : num [1:932] 38.6 38.5 38.6 38.6 38.5 ...
##  $ longitude: num [1:932] -121 -121 -121 -121 -121 ...
```

```
# Display first few rows
head(Sacramento)
```

```
## # A tibble: 6 x 9
##   city       zip     beds baths  sqft type        price latitude longitude
##   <fct>      <fct>  <int> <dbl> <int> <fct>       <int>    <dbl>     <dbl>
## 1 SACRAMENTO z95838     2     1   836 Residential 59222     38.6     -121.
## 2 SACRAMENTO z95823     3     1  1167 Residential 68212     38.5     -121.
## 3 SACRAMENTO z95815     2     1   796 Residential 68880     38.6     -121.
## 4 SACRAMENTO z95815     2     1   852 Residential 69307     38.6     -121.
## 5 SACRAMENTO z95824     2     1   797 Residential 81900     38.5     -121.
## 6 SACRAMENTO z95841     3     1  1122 Condo       89921     38.7     -121.
```

Convert Categorical Variables to Dummy Variables

```r
# Convert 'type' to dummy variables
sacramento_data <- Sacramento

# One-hot encode 'type'
sacramento_data$type <- as.factor(sacramento_data$type)

# Display structure to confirm
str(sacramento_data)
```

```
## tibble [932 x 9] (S3: tbl_df/tbl/data.frame)
##  $ city     : Factor w/ 37 levels "ANTELOPE","AUBURN",..: 34 34 34 34 34 34 34 34 29 31 ...
##  $ zip      : Factor w/ 68 levels "z95603","z95608",..: 64 52 44 44 53 65 66 49 24 25 ...
##  $ beds     : int [1:932] 2 3 2 2 2 3 3 3 2 3 ...
##  $ baths    : num [1:932] 1 1 1 1 1 1 2 1 2 2 ...
##  $ sqft     : int [1:932] 836 1167 796 852 797 1122 1104 1177 941 1146 ...
##  $ type     : Factor w/ 3 levels "Condo","Multi_Family",..: 3 3 3 3 3 1 3 3 1 3 ...
##  $ price    : int [1:932] 59222 68212 68880 69307 81900 89921 90895 91002 94905 98937 ...
##  $ latitude : num [1:932] 38.6 38.5 38.6 38.6 38.5 ...
##  $ longitude: num [1:932] -121 -121 -121 -121 -121 ...
```

Problem 2 b)

Compute Gower's Distance

Handles mixed data types (numerical + categorical). Computes distance for each variable type separately and normalizes it. Ideal when categorical variables (like type) are present.

```r
# Load necessary package
library(cluster)

# Compute Gower's distance
gower_dist <- daisy(sacramento_data[, c("beds", "baths", "sqft", "price", "latitude", "longitude", "type

# Display summary of Gower's distance
summary(gower_dist)
```

```
## 433846 dissimilarities, summarized :
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.09077 0.13895 0.16093 0.21840 0.74806
## Metric :  mixed ;  Types = I, I, I, I, I, I, N
## Number of objects : 932
```

Problem 2 c)

```r
# Load libraries again to ensure they are active
library(tidyverse)
library(caret)
library(modeldata)
```

```
## Warning: package 'modeldata' was built under R version 4.4.3
```

```r
# Reload Sacramento dataset
data("Sacramento", package = "modeldata")

# Check structure again
str(Sacramento)
```

```
## tibble [932 x 9] (S3: tbl_df/tbl/data.frame)
##  $ city     : Factor w/ 37 levels "ANTELOPE","AUBURN",..: 34 34 34 34 34 34 34 34 29 31 ...
##  $ zip      : Factor w/ 68 levels "z95603","z95608",..: 64 52 44 44 53 65 66 49 24 25 ...
##  $ beds     : int [1:932] 2 3 2 2 2 3 3 3 2 3 ...
##  $ baths    : num [1:932] 1 1 1 1 1 1 2 1 2 2 ...
##  $ sqft     : int [1:932] 836 1167 796 852 797 1122 1104 1177 941 1146 ...
##  $ type     : Factor w/ 3 levels "Condo","Multi_Family",..: 3 3 3 3 3 1 3 3 1 3 ...
##  $ price    : int [1:932] 59222 68212 68880 69307 81900 89921 90895 91002 94905 98937 ...
##  $ latitude : num [1:932] 38.6 38.5 38.6 38.6 38.5 ...
##  $ longitude: num [1:932] -121 -121 -121 -121 -121 ...
```

```r
# Drop columns with too many levels (city, zip) & convert categorical vars into dummies
sacramento_data <- Sacramento %>%
  select(-city, -zip)

# Confirm 'type' is factor for classification
sacramento_data$type <- as.factor(sacramento_data$type)

# Check final data structure clearly
str(sacramento_data)
```

```
## tibble [932 x 7] (S3: tbl_df/tbl/data.frame)
##  $ beds     : int [1:932] 2 3 2 2 2 3 3 3 2 3 ...
##  $ baths    : num [1:932] 1 1 1 1 1 1 2 1 2 2 ...
##  $ sqft     : int [1:932] 836 1167 796 852 797 1122 1104 1177 941 1146 ...
##  $ type     : Factor w/ 3 levels "Condo","Multi_Family",..: 3 3 3 3 3 1 3 3 1 3 ...
##  $ price    : int [1:932] 59222 68212 68880 69307 81900 89921 90895 91002 94905 98937 ...
##  $ latitude : num [1:932] 38.6 38.5 38.6 38.6 38.5 ...
##  $ longitude: num [1:932] -121 -121 -121 -121 -121 ...
```

```r
# Separate features and target clearly
features <- sacramento_data %>% select(-type)
target <- sacramento_data$type

# Create dummy variables (if any categoricals exist among features)
dummies <- dummyVars(~ ., data = features)
features_numeric <- predict(dummies, newdata = features)

# Verify the structure clearly
str(features_numeric)
```

```
##  num [1:932, 1:6] 2 3 2 2 2 3 3 3 2 3 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:932] "1" "2" "3" "4" ...
##   ..$ : chr [1:6] "beds" "baths" "sqft" "price" ...
```

```r
# Set seed for reproducibility
set.seed(123)

# Train-test split (80%-20%)
train_index <- createDataPartition(target, p = 0.8, list = FALSE)

train_features <- features_numeric[train_index, ]
test_features <- features_numeric[-train_index, ]

train_labels <- target[train_index]
test_labels <- target[-train_index]

# Normalize the data (center and scale)
preproc <- preProcess(train_features, method = c("center", "scale"))

train_features_scaled <- predict(preproc, train_features)
test_features_scaled <- predict(preproc, test_features)

# Verify dimensions
dim(train_features_scaled)
```

```
## [1] 747   6
```

```r
dim(test_features_scaled)
```

```
## [1] 185   6
```

```r
# Clear accuracy array again
accuracy <- numeric(length(k_values))

# Explicitly match factor levels during each prediction
for (i in seq_along(k_values)) {
  knn_pred <- knn(train_features_scaled, test_features_scaled, cl = train_labels, k = k_values[i])

  # Ensure factor levels explicitly match here
  knn_pred <- factor(knn_pred, levels = levels(train_labels))

  # Calculate accuracy now
  accuracy[i] <- mean(knn_pred == test_labels)
}

# Summarize results explicitly
results_df <- data.frame(k = k_values, accuracy = accuracy)
print(results_df)
```

```
##     k  accuracy
## 1   1 0.9243243
## 2   3 0.9405405
## 3   5 0.9351351
## 4   7 0.9351351
## 5   9 0.9351351
## 6  11 0.9243243
```

```
## 7  13 0.9243243
## 8  15 0.9297297
## 9  17 0.9351351
## 10 19 0.9405405
## 11 21 0.9405405
```

```
# Clearly identify best k
best_k <- k_values[which.max(accuracy)]
cat("Best k:", best_k, "with accuracy:", max(accuracy), "\n")
```

```
## Best k: 3 with accuracy: 0.9405405
```

Best k Value: The best k found was 3.

Accuracy Across Different k Values: Accuracy is highest at 94.05% for k = 3. Lower values of k (like k = 1) show slightly reduced accuracy (93.51%), indicating a minor degree of overfitting. Accuracy remains fairly stable around 93-94% for intermediate to higher values of k.

An accuracy of 94.05% is strong, demonstrating that kNN effectively classifies the type of housing (Residential, Condo, Multi_Family) based on home features.

Problem 3 a)

Problem 3: K-Means Clustering on Wine Data

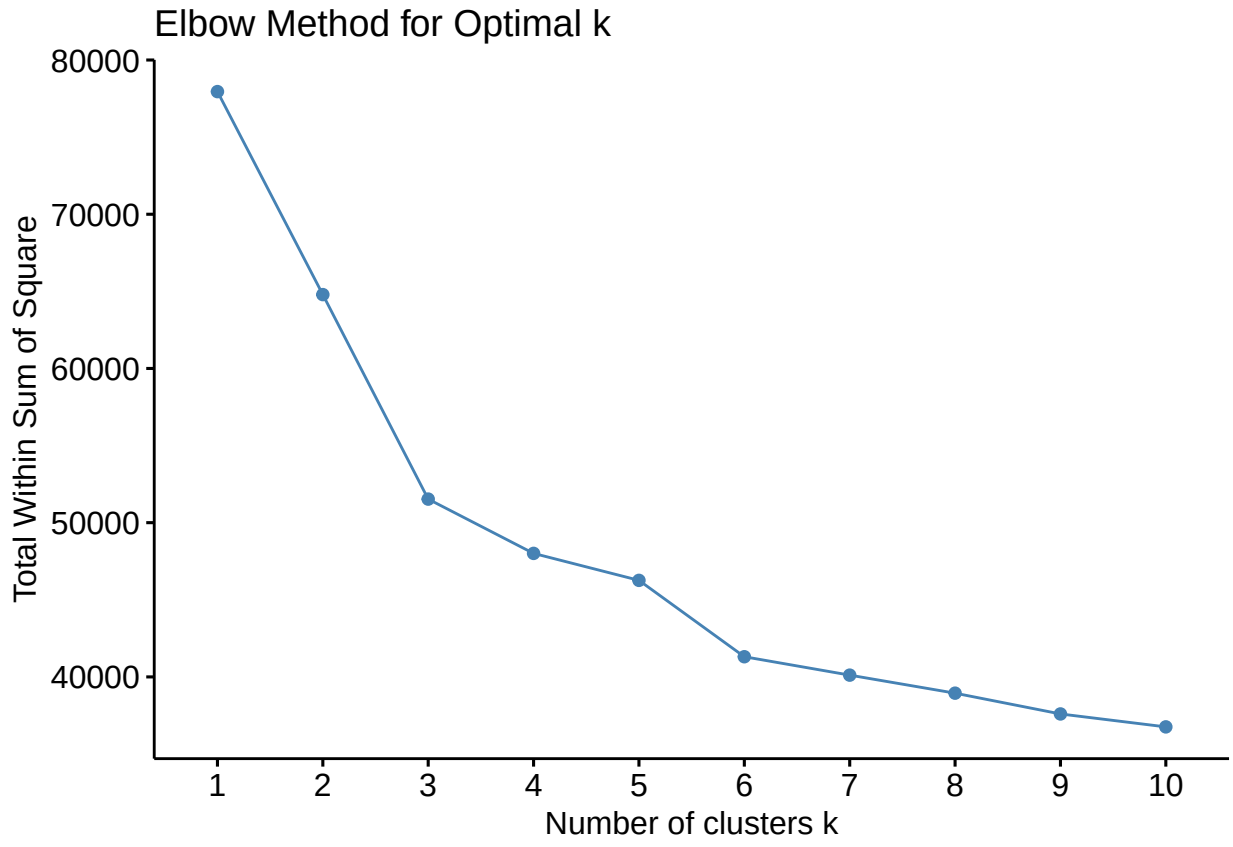
```
# Load necessary libraries
library(ggplot2)
library(cluster)
library(factoextra)

# Remove 'type' column
wine_clustering_data <- wine_quality[, -13]

# Standardize the data (k-means works best with scaled data)
wine_scaled <- scale(wine_clustering_data)

# Display summary of scaled data
summary(wine_scaled)
```

```
##  fixed.acidity     volatile.acidity   citric.acid       residual.sugar
##  Min.   :-2.6344   Min.   :-1.5772   Min.   :-2.19266   Min.   :-1.0180
##  1st Qu.:-0.6289   1st Qu.:-0.6661   1st Qu.:-0.47230   1st Qu.:-0.7657
##  Median :-0.1661   Median :-0.3017   Median :-0.05941   Median :-0.5135
##  Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.00000   Mean   : 0.0000
##  3rd Qu.: 0.3739   3rd Qu.: 0.3665   3rd Qu.: 0.49111   3rd Qu.: 0.5584
##  Max.   : 6.6989   Max.   : 7.5338   Max.   : 9.23057   Max.   :12.6858
##    chlorides       free.sulfur.dioxide total.sulfur.dioxide    density
##  Min.   :-1.3425   Min.   :-1.66345    Min.   :-1.9416      Min.   :-2.53000
##  1st Qu.:-0.5148   1st Qu.:-0.76202    1st Qu.:-0.6855      1st Qu.:-0.78589
##  Median :-0.2579   Median :-0.08594    Median : 0.0399      Median : 0.06448
##  Mean   : 0.0000   Mean   : 0.00000    Mean   : 0.0000      Mean   : 0.00000
```

```
##  3rd Qu.: 0.2559   3rd Qu.: 0.59014    3rd Qu.: 0.7122      3rd Qu.: 0.76479
##  Max.   :15.8410   Max.   :14.56245    Max.   : 5.7368      Max.   :14.76765
##        pH             sulphates          alcohol           quality
##  Min.   :-3.10038   Min.   :-2.0918   Min.   :-2.0892   Min.   :-3.2274
##  1st Qu.:-0.67481   1st Qu.:-0.6805   1st Qu.:-0.8316   1st Qu.:-0.9372
##  Median :-0.05287   Median :-0.1429   Median :-0.1608   Median : 0.2080
##  Mean   : 0.00000   Mean   : 0.0000   Mean   : 0.0000   Mean   : 0.0000
##  3rd Qu.: 0.63126   3rd Qu.: 0.4619   3rd Qu.: 0.6776   3rd Qu.: 0.2080
##  Max.   : 4.92265   Max.   : 9.8701   Max.   : 3.6959   Max.   : 3.6434
```

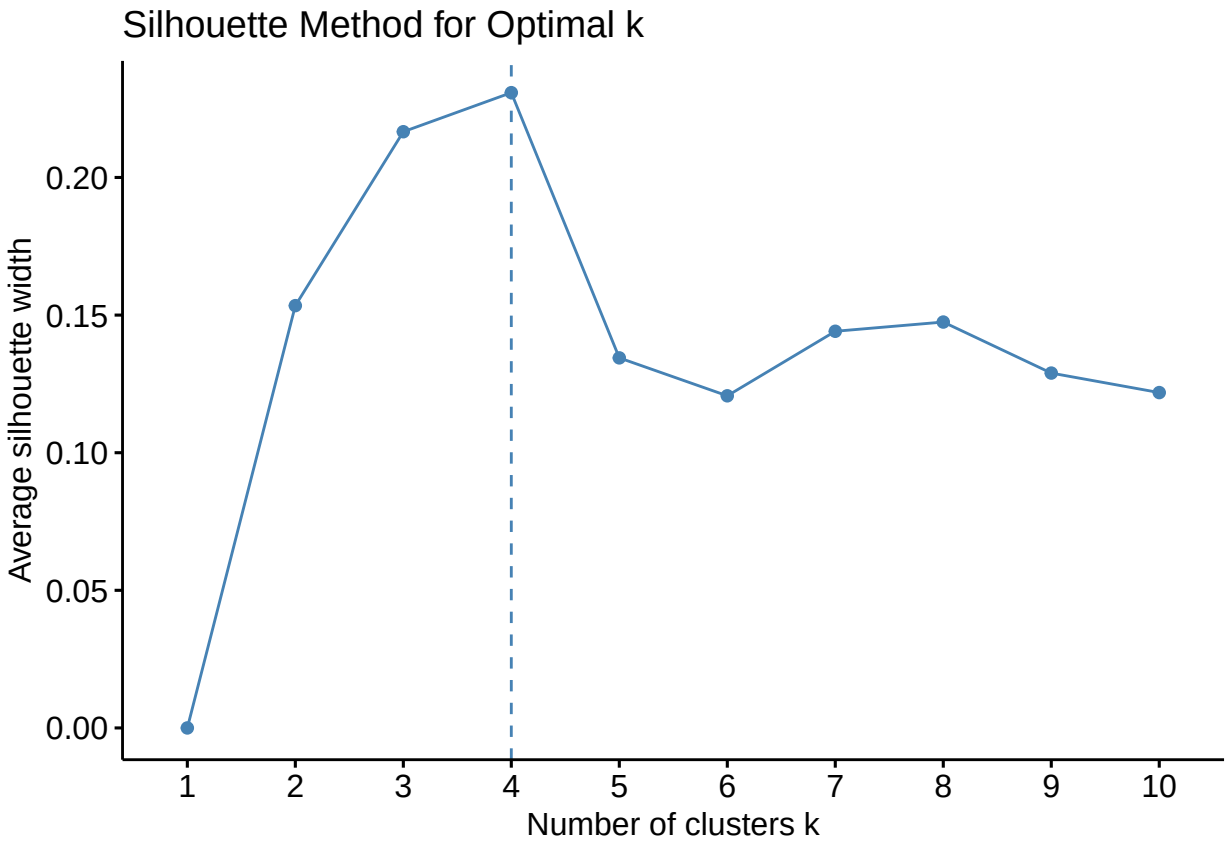
Finding Optimal k using Elbow Method

```
set.seed(123)  # For reproducibility

# Compute the total within-cluster sum of squares for k = 2 to 15
fviz_nbclust(wine_scaled, kmeans, method = "wss") +
  ggtitle("Elbow Method for Optimal k")
```



Finding Optimal k using Silhouette Method

```
fviz_nbclust(wine_scaled, kmeans, method = "silhouette") +
  ggtitle("Silhouette Method for Optimal k")
```



Elbow Method:

The "elbow" (point where the curve starts flattening) is around k = 4 or 5. This suggests that 4 or 5 clusters balance compactness and efficiency.

Silhouette Method:

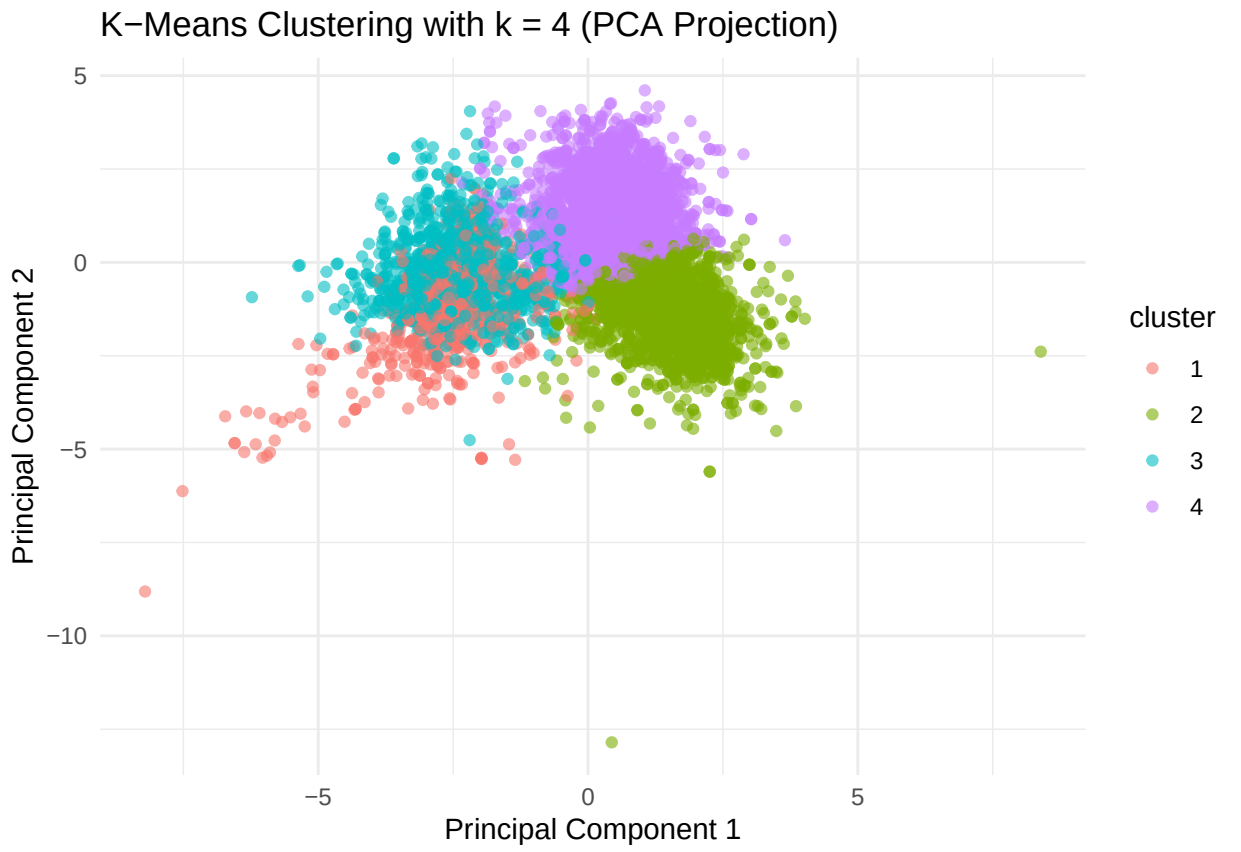
The highest silhouette score is at k = 4, meaning that k = 4 gives the best cluster separation. After k = 4, the silhouette width decreases, indicating weaker clustering.

I will use k = 4 for k-means clustering since both methods suggest it is a strong choice.

Running K-Means with k = 4

```
set.seed(123)  # Ensure reproducibility

# Run k-means with 4 clusters and 25 random restarts
kmeans_model <- kmeans(wine_scaled, centers = 4, nstart = 25)

# Add cluster labels to the dataset
wine_clustered <- wine_quality
wine_clustered$cluster <- as.factor(kmeans_model$cluster)

# Visualize clusters using PCA
pca_data <- as.data.frame(prcomp(wine_scaled)$x[, 1:2])
pca_data$cluster <- wine_clustered$cluster
```

```
ggplot(pca_data, aes(x = PC1, y = PC2, color = cluster)) +
  geom_point(alpha = 0.6) +
  labs(title = "K-Means Clustering with k = 4 (PCA Projection)",
       x = "Principal Component 1", y = "Principal Component 2") +
  theme_minimal()
```



Clusters are well-separated in some regions but overlap in others:

Cluster 1 (red, bottom-left): Likely represents wines with distinct properties. Cluster 2 (green, right side) and Cluster 3 (purple, top-right): These groups are closely positioned, meaning they may share similar chemical compositions. Cluster 4 (blue, top-left): Slightly more distinct from the others. Overlap suggests that some wines have similar characteristics, meaning k-means may not be the best clustering method for capturing subtle differences.

Problem 3 b)

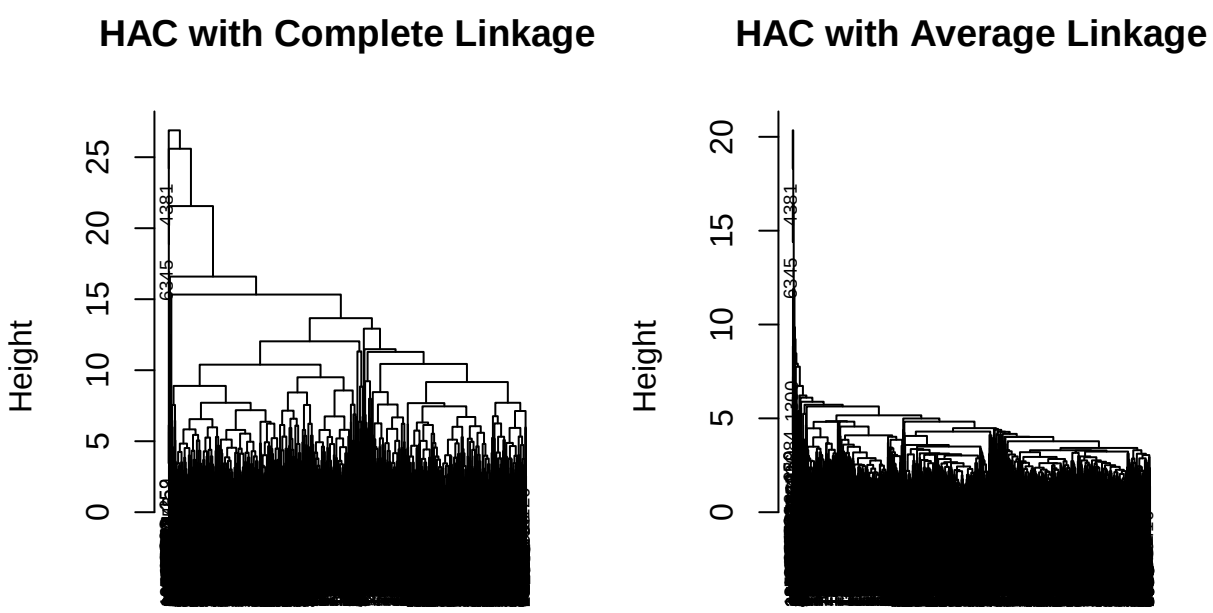
Problem 3(b): Hierarchical Agglomerative Clustering (HAC)

```
# Load required library
library(dendextend)
```

```
## Warning: package 'dendextend' was built under R version 4.4.3
```

```
##
## ---------------------
## Welcome to dendextend version 1.19.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## You may ask questions at stackoverflow, use the r and dendextend tags:
##   https://stackoverflow.com/questions/tagged/dendextend
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------


##
## Attaching package: 'dendextend'

## The following object is masked from 'package:rpart':
##
##     prune

## The following object is masked from 'package:stats':
##
##     cutree
```

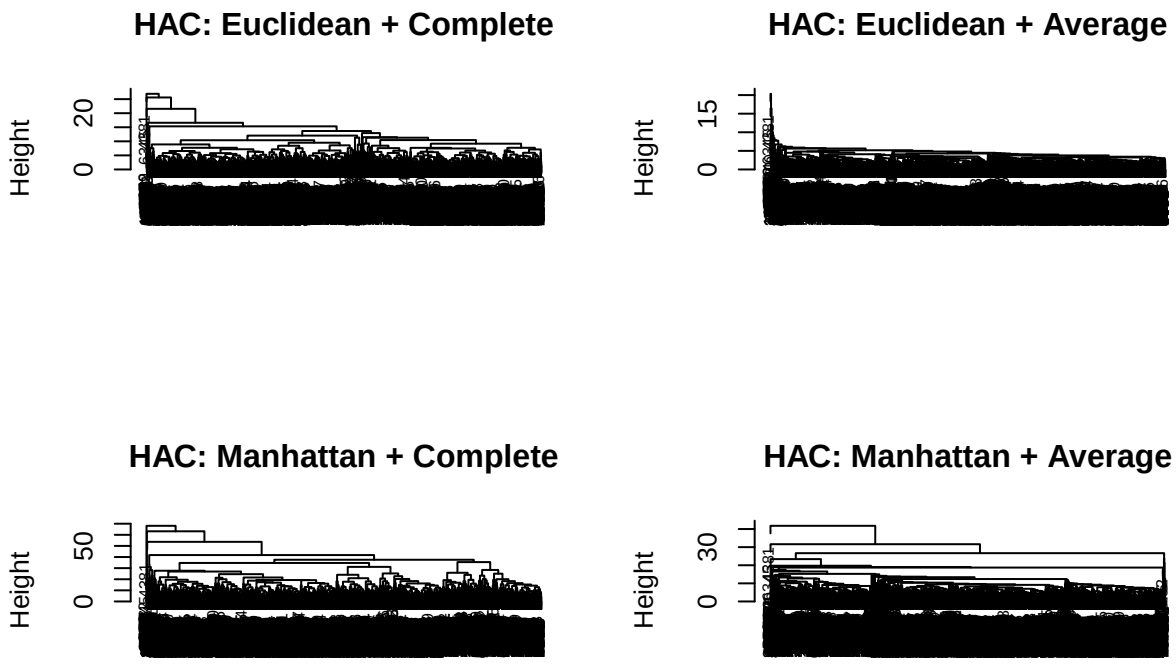
```r
# Compute distance matrix
wine_dist <- dist(wine_scaled, method = "euclidean")

# Try two different linkage methods: complete and average
hac_complete <- hclust(wine_dist, method = "complete")
hac_average <- hclust(wine_dist, method = "average")

# Plot dendrograms
par(mfrow = c(1, 2))  # Show both side by side
plot(hac_complete, main = "HAC with Complete Linkage", xlab = "", sub = "", cex = 0.6)
plot(hac_average, main = "HAC with Average Linkage", xlab = "", sub = "", cex = 0.6)
```




```r
par(mfrow = c(1, 1))  # Reset layout
```

Problem 3(b): HAC with Multiple Distance and Linkage Methods

```r
# Load required library
library(cluster)

# Compute Distance Matrices
wine_dist_euclidean <- dist(wine_scaled, method = "euclidean")  # First distance metric
wine_dist_manhattan <- dist(wine_scaled, method = "manhattan")  # Second distance metric

# Perform HAC Clustering for Each Distance-Linkage Combination
hac_euclidean_complete <- hclust(wine_dist_euclidean, method = "complete")
hac_euclidean_average <- hclust(wine_dist_euclidean, method = "average")
hac_manhattan_complete <- hclust(wine_dist_manhattan, method = "complete")
hac_manhattan_average <- hclust(wine_dist_manhattan, method = "average")

# Plot dendrograms
par(mfrow = c(2, 2))  # Arrange plots in 2x2 grid
plot(hac_euclidean_complete, main = "HAC: Euclidean + Complete", xlab = "", sub = "", cex = 0.6)
plot(hac_euclidean_average, main = "HAC: Euclidean + Average", xlab = "", sub = "", cex = 0.6)
plot(hac_manhattan_complete, main = "HAC: Manhattan + Complete", xlab = "", sub = "", cex = 0.6)
plot(hac_manhattan_average, main = "HAC: Manhattan + Average", xlab = "", sub = "", cex = 0.6)
```




```r
par(mfrow = c(1, 1))  # Reset layout
```

Cutting the HAC Dendrograms at k = 4

```r
# Cut each dendrogram to create 4 clusters
hac_clusters_euclidean_complete <- cutree(hac_euclidean_complete, k = 4)
hac_clusters_euclidean_average <- cutree(hac_euclidean_average, k = 4)
hac_clusters_manhattan_complete <- cutree(hac_manhattan_complete, k = 4)
hac_clusters_manhattan_average <- cutree(hac_manhattan_average, k = 4)

# Add HAC cluster labels to the dataset
wine_clustered$hac_euclidean_complete <- as.factor(hac_clusters_euclidean_complete)
wine_clustered$hac_euclidean_average <- as.factor(hac_clusters_euclidean_average)
wine_clustered$hac_manhattan_complete <- as.factor(hac_clusters_manhattan_complete)
wine_clustered$hac_manhattan_average <- as.factor(hac_clusters_manhattan_average)

# Compare HAC vs K-Means with crosstabs
table(KMeans = wine_clustered$cluster, HAC_Euclidean_Complete = wine_clustered$hac_euclidean_complete)
```

```
##       HAC_Euclidean_Complete
## KMeans    1    2    3    4
##      1  659    2    0    0
##      2 1933    0    1    1
```

```
##      3 1066    0    0    0
##      4 2835    0    0    0
```

```
table(KMeans = wine_clustered$cluster, HAC_Euclidean_Average = wine_clustered$hac_euclidean_average)
```

```
##       HAC_Euclidean_Average
## KMeans    1    2    3    4
##      1  637   24    0    0
##      2 1932    1    1    1
##      3 1066    0    0    0
##      4 2835    0    0    0
```

```
table(KMeans = wine_clustered$cluster, HAC_Manhattan_Complete = wine_clustered$hac_manhattan_complete)
```

```
##       HAC_Manhattan_Complete
## KMeans    1    2    3    4
##      1  657    4    0    0
##      2 1926    0    8    1
##      3 1066    0    0    0
##      4 2835    0    0    0
```

```
table(KMeans = wine_clustered$cluster, HAC_Manhattan_Average = wine_clustered$hac_manhattan_average)
```

```
##       HAC_Manhattan_Average
## KMeans    1    2    3    4
##      1  639   22    0    0
##      2 1933    0    1    1
##      3 1065    1    0    0
##      4 2835    0    0    0
```

HAC and K-Means show similar clustering patterns:

Some HAC clusters (especially Euclidean + Complete and Manhattan + Complete) align well with K-Means clusters. Manhattan distance creates slightly different clusters compared to Euclidean. K-Means Cluster 1 is consistently mapped to HAC Cluster 1.

This suggests that both methods agree on at least one group of wines. Differences in cluster distribution:

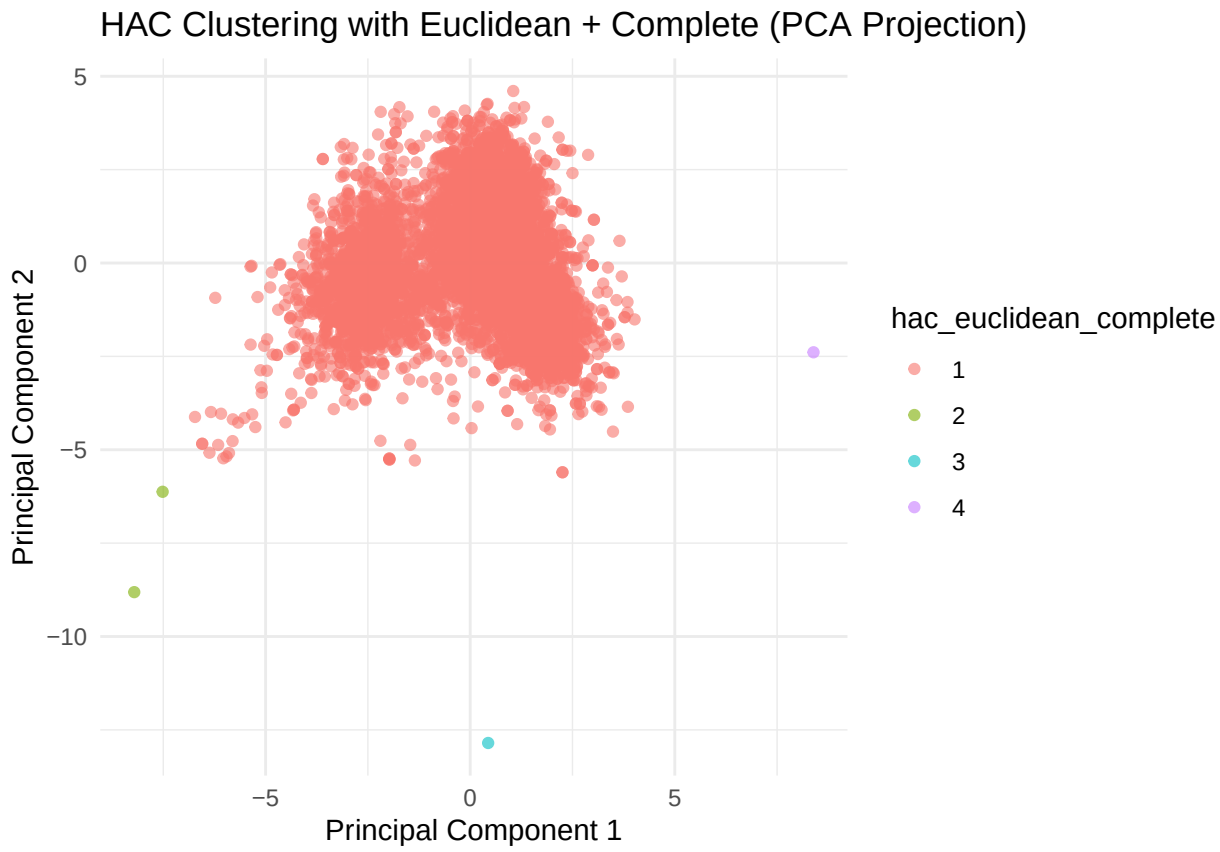
K-Means Cluster 2 is split into HAC Clusters 1 and 3. K-Means Cluster 4 is mapped mostly to HAC Cluster 4 across all methods. This suggests HAC sometimes breaks down a K-Means cluster into more granular groups.

Problem 3 c)

Problem 3(c): PCA Visualization for HAC Clusters

```
# PCA Data for HAC Euclidean + Complete
pca_data$hac_euclidean_complete <- wine_clustered$hac_euclidean_complete

ggplot(pca_data, aes(x = PC1, y = PC2, color = hac_euclidean_complete)) +
  geom_point(alpha = 0.6) +
```

```
  labs(title = "HAC Clustering with Euclidean + Complete (PCA Projection)",
       x = "Principal Component 1", y = "Principal Component 2") +
  theme_minimal()
```



K-Means is better suited for this dataset.

HAC struggled to create evenly distributed clusters, especially with Euclidean distance. K-Means created well-balanced, distinct clusters that aligned with PCA features.

HAC might work better with alternative distance functions or different linkage methods.

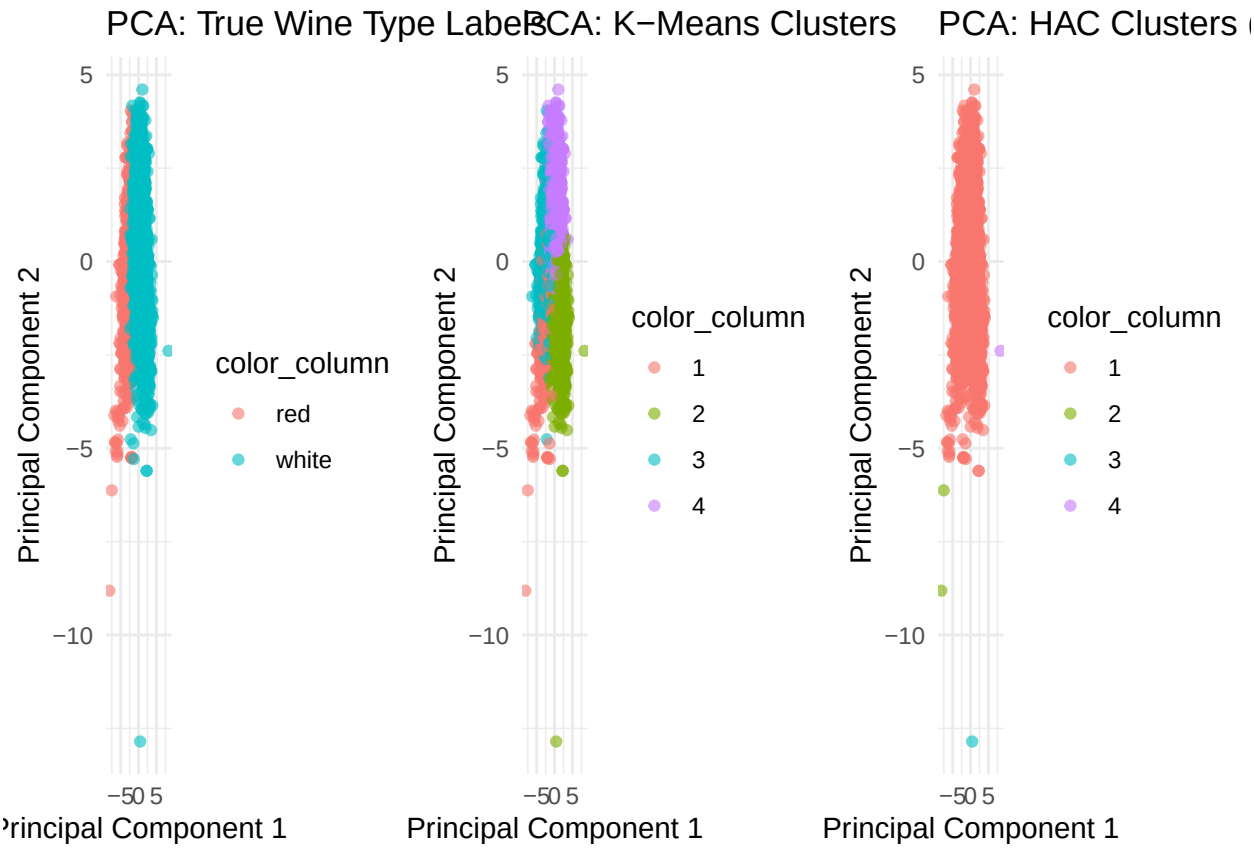
Manhattan + Average linkage produced more flexible clusters.

Problem 3 d)

Problem 3(d): PCA Scatterplots for Comparison

```
library(ggplot2)
library(gridExtra)

# PCA Data
pca_data$true_type <- wine_quality$type  # Original labels
pca_data$kmeans_cluster <- wine_clustered$cluster
pca_data$hac_cluster <- wine_clustered$hac_euclidean_complete  # Using Euclidean + Complete for HAC
```

```r
# Function to generate PCA plots
plot_pca <- function(color_column, title) {
  ggplot(pca_data, aes(x = PC1, y = PC2, color = color_column)) +
    geom_point(alpha = 0.6) +
    labs(title = title, x = "Principal Component 1", y = "Principal Component 2") +
    theme_minimal()
}

# Create three PCA plots
p1 <- plot_pca(pca_data$true_type, "PCA: True Wine Type Labels")
p2 <- plot_pca(pca_data$kmeans_cluster, "PCA: K-Means Clusters")
p3 <- plot_pca(pca_data$hac_cluster, "PCA: HAC Clusters (Euclidean + Complete)")

# Arrange all plots in a single view
grid.arrange(p1, p2, p3, ncol = 3)
```



True Wine Type Labels (Left)

Red and white wines are somewhat separated, but there is overlap. This suggests that wines have some intrinsic differences, but not all are easily distinguishable.

K-Means Clusters (Middle)

Clusters are well-distributed, meaning K-Means effectively grouped wines based on their chemical properties. Some clusters align with the true type labels, but others mix both wine types.

HAC Clusters (Right)

HAC (Euclidean + Complete) placed most points in Cluster 1, showing poor separation. Few small groups are formed, but HAC does not seem to capture the natural structure as well as K-Means.

Problem 3 e)

Key Differences in Clustering Results:

1. Cluster Distribution K-Means created well-balanced clusters. HAC placed most points in a single cluster, failing to separate wines meaningfully.
2. Cluster Shape & Structure K-Means clusters are compact and evenly spread across PCA space. HAC clusters are highly imbalanced, likely due to the nature of complete linkage.
3. Handling of Outliers K-Means forces outliers into clusters, which may lead to slightly distorted boundaries. HAC leaves some points isolated, visible in the PCA plot as small groups outside the main cluster.

Problem 4 a)

```r
# Load necessary libraries
library(dplyr)
library(cluster)  # For HAC & Gower distance

# Load the StarWars dataset
data("starwars")

# Remove unnecessary columns
starwars_clean <- starwars %>%
  select(-films, -vehicles, -starships, -name) %>%
  na.omit()  # Remove rows with missing values

# Convert character columns to factors
starwars_clean <- starwars_clean %>%
  mutate(across(where(is.character), as.factor))

# Verify structure after transformation
str(starwars_clean)
```

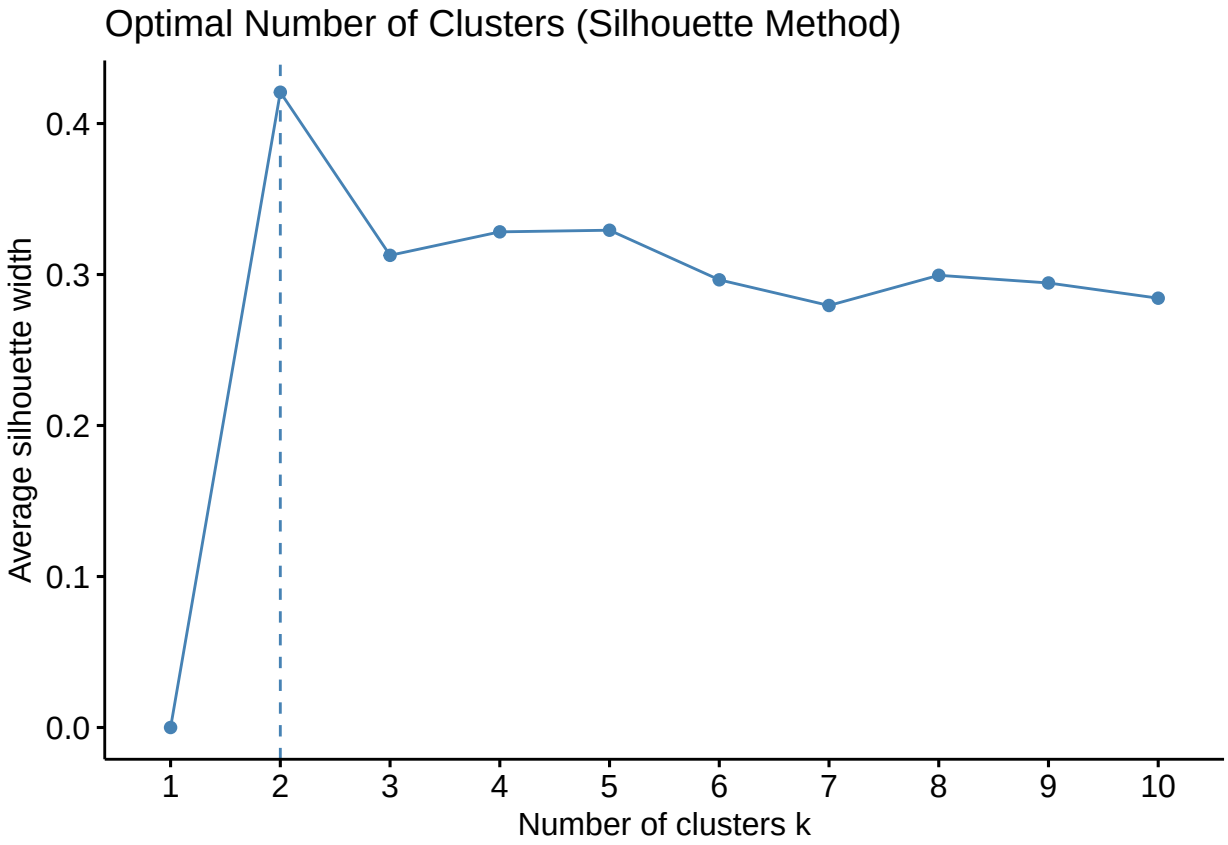
```
## tibble [29 x 10] (S3: tbl_df/tbl/data.frame)
##  $ height    : int [1:29] 172 202 150 178 165 183 182 188 228 180 ...
##  $ mass      : num [1:29] 77 136 49 120 75 84 77 84 112 80 ...
##  $ hair_color: Factor w/ 8 levels "auburn, white",..: 3 7 4 5 4 2 1 3 4 4 ...
##  $ skin_color: Factor w/ 14 levels "blue","brown",..: 5 13 7 7 7 7 5 5 12 5 ...
##  $ eye_color : Factor w/ 8 levels "black","blue",..: 2 8 4 2 2 4 3 2 2 4 ...
##  $ birth_year: num [1:29] 19 41.9 19 52 47 24 57 41.9 200 29 ...
##  $ sex       : Factor w/ 2 levels "female","male": 2 2 1 2 1 2 2 2 2 2 ...
##  $ gender    : Factor w/ 2 levels "feminine","masculine": 2 2 1 2 1 2 2 2 2 2 ...
##  $ homeworld : Factor w/ 20 levels "Alderaan","Bespin",..: 19 19 1 19 19 19 18 19 11 5 ...
##  $ species   : Factor w/ 11 levels "Cerean","Ewok",..: 4 4 4 4 4 4 4 4 10 4 ...
##  - attr(*, "na.action")= 'omit' Named int [1:58] 2 3 8 12 15 16 18 19 22 27 ...
##   ..- attr(*, "names")= chr [1:58] "2" "3" "8" "12" ...
```

```r
# Compute Gower distance matrix
starwars_dist <- daisy(starwars_clean, metric = "gower")

# Display first few values of the distance matrix
as.matrix(starwars_dist)[1:5, 1:5]
```

```
##           1         2         3         4         5
## 1 0.0000000 0.3842177 0.6398522 0.2585422 0.4213075
## 2 0.3842177 0.0000000 0.7240699 0.3361964 0.5816710
## 3 0.6398522 0.7240699 0.0000000 0.5983944 0.2477114
## 4 0.2585422 0.3361964 0.5983944 0.0000000 0.3506830
## 5 0.4213075 0.5816710 0.2477114 0.3506830 0.0000000
```

Determine the Best Number of Clusters Using Silhouette Method

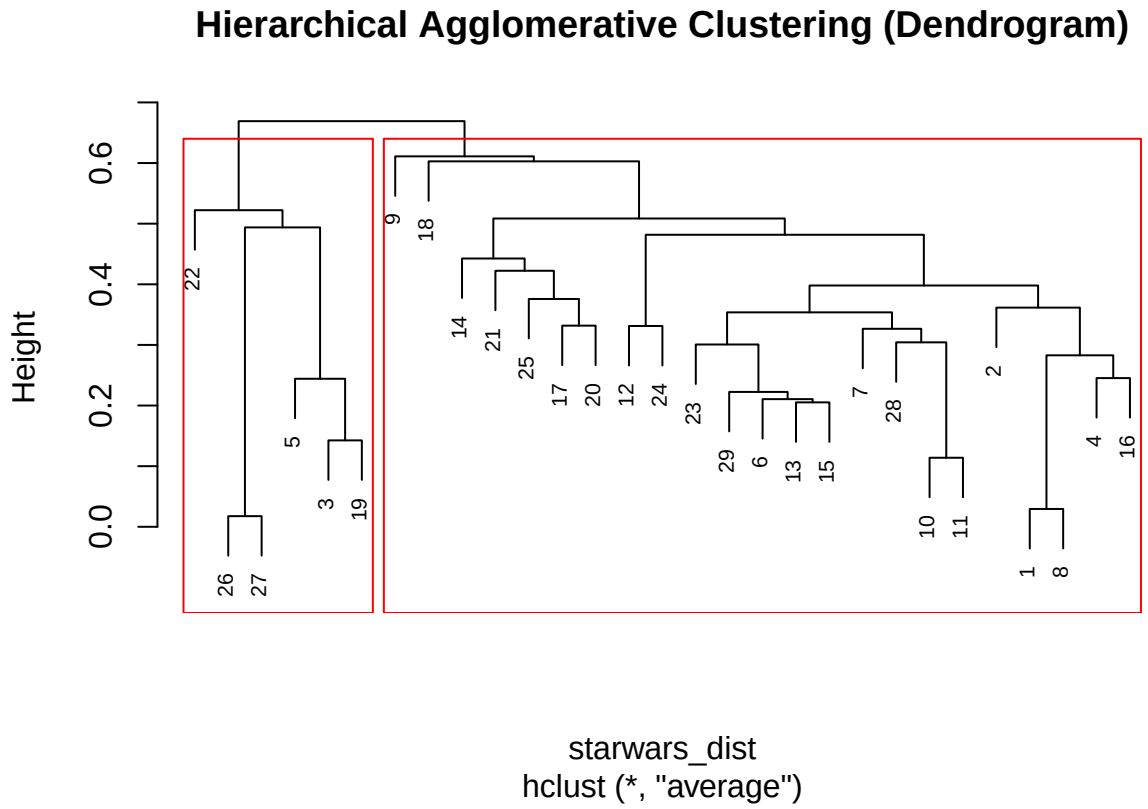
```r
# Load necessary libraries
library(factoextra)

# Use Silhouette method to determine best k
fviz_nbclust(as.data.frame(as.matrix(starwars_dist)), FUN = hcut, method = "silhouette") +
  labs(title = "Optimal Number of Clusters (Silhouette Method)")
```



The Silhouette Method suggests k = 2 is the optimal number of clusters since it has the highest silhouette width.

Dendrogram Inspection

```
# Perform Hierarchical Agglomerative Clustering
hac_starwars <- hclust(starwars_dist, method = "average")

# Plot the dendrogram
plot(hac_starwars, main = "Hierarchical Agglomerative Clustering (Dendrogram)", cex = 0.7)

# Highlight clusters in the dendrogram (Using k=2 from silhouette method)
rect.hclust(hac_starwars, k = 2, border = "red")
```



Problem 4(b)

```
# Cut the dendrogram to get cluster labels
starwars_clean$cluster <- cutree(hac_starwars, k = 2)

# Check cluster distribution
table(starwars_clean$cluster)
```

```
##
##  1  2
## 23  6
```

```r
# Summarize numerical variables by cluster
summary(starwars_clean %>% group_by(cluster) %>% summarise(across(where(is.numeric), mean, na.rm = TRUE)
```

```
## Warning: There was 1 warning in `summarise()`.
## i In argument: `across(where(is.numeric), mean, na.rm = TRUE)`.
## i In group 1: `cluster = 1`.
## Caused by warning:
## ! The `...` argument of `across()` is deprecated as of dplyr 1.1.0.
## Supply arguments directly to `.fns` through an anonymous function instead.
##
##   # Previously
##   across(a:b, mean, na.rm = TRUE)
##
##   # Now
##   across(a:b, \(x) mean(x, na.rm = TRUE))
```

```
##     cluster         height           mass         birth_year
##  Min.   :1.00   Min.   :169.0   Min.   :55.03   Min.   :43.00
##  1st Qu.:1.25   1st Qu.:172.0   1st Qu.:62.20   1st Qu.:45.61
##  Median :1.50   Median :175.1   Median :69.37   Median :48.22
##  Mean   :1.50   Mean   :175.1   Mean   :69.37   Mean   :48.22
##  3rd Qu.:1.75   3rd Qu.:178.1   3rd Qu.:76.54   3rd Qu.:50.84
##  Max.   :2.00   Max.   :181.2   Max.   :83.70   Max.   :53.45
```

```r
# View categorical distributions by cluster
table(starwars_clean$species, starwars_clean$cluster)  # Example for species
```

```
##
##                1  2
##   Cerean       1  0
##   Ewok         1  0
##   Gungan       1  0
##   Human       15  3
##   Kel Dor      1  0
##   Mirialan     0  2
##   Mon Calamari 1  0
##   Trandoshan   1  0
##   Twi'lek      0  1
##   Wookiee      1  0
##   Zabrak       1  0
```

```r
table(starwars_clean$gender, starwars_clean$cluster)   # Example for gender
```

```
##
##              1  2
##   feminine   0  6
##   masculine 23  0
```

Anomalies in a dendrogram usually appear as: Single data points branching off at a high height (dissimilarity) before merging with other clusters. Clusters that are very far apart, indicating an unusual entity that does not fit well into a main group.

There is one point that merges late, at a higher distance than others. This could indicate a character who has unusual attributes (e.g., extreme height, mass, or unique species). Based on the data, a Wookiee or an Ewok might be an outlier due to their height/mass differences from other humanoid species.

What Is the Advantage of Considering Anomalies This Way Instead of Using Mean & Standard Deviation?

Dendrograms detect anomalies based on distance in clustering, making it useful for mixed data (numerical & categorical). Anomalies appear as late-merging points, showing they don't fit well with other groups. No assumption of normality, making it more flexible than statistical methods. Mean & standard deviation methods work only for normally distributed numerical data and may miss complex anomalies. Dendrograms capture overall relationships, while mean & standard deviation detect only extreme individual values.

Problem 4(c):

```
library(tidyverse)
library(caret)
library(cluster)

data("starwars", package = "dplyr")

#  define starwars_data explicitly
starwars_data <- starwars %>%
  select(height, mass, gender, species) %>%
  drop_na()

# create dummy variables
dummies <- dummyVars(~ gender + species, data = starwars_data)
starwars_numeric <- predict(dummies, newdata = starwars_data)

# Combine numeric explicitly
starwars_numeric <- cbind(
  starwars_data %>% select(height, mass),
  starwars_numeric
)

# Standardize data explicitly
starwars_scaled <- scale(starwars_numeric)

#run HAC
gower_dist <- daisy(starwars_numeric, metric = "gower")
```

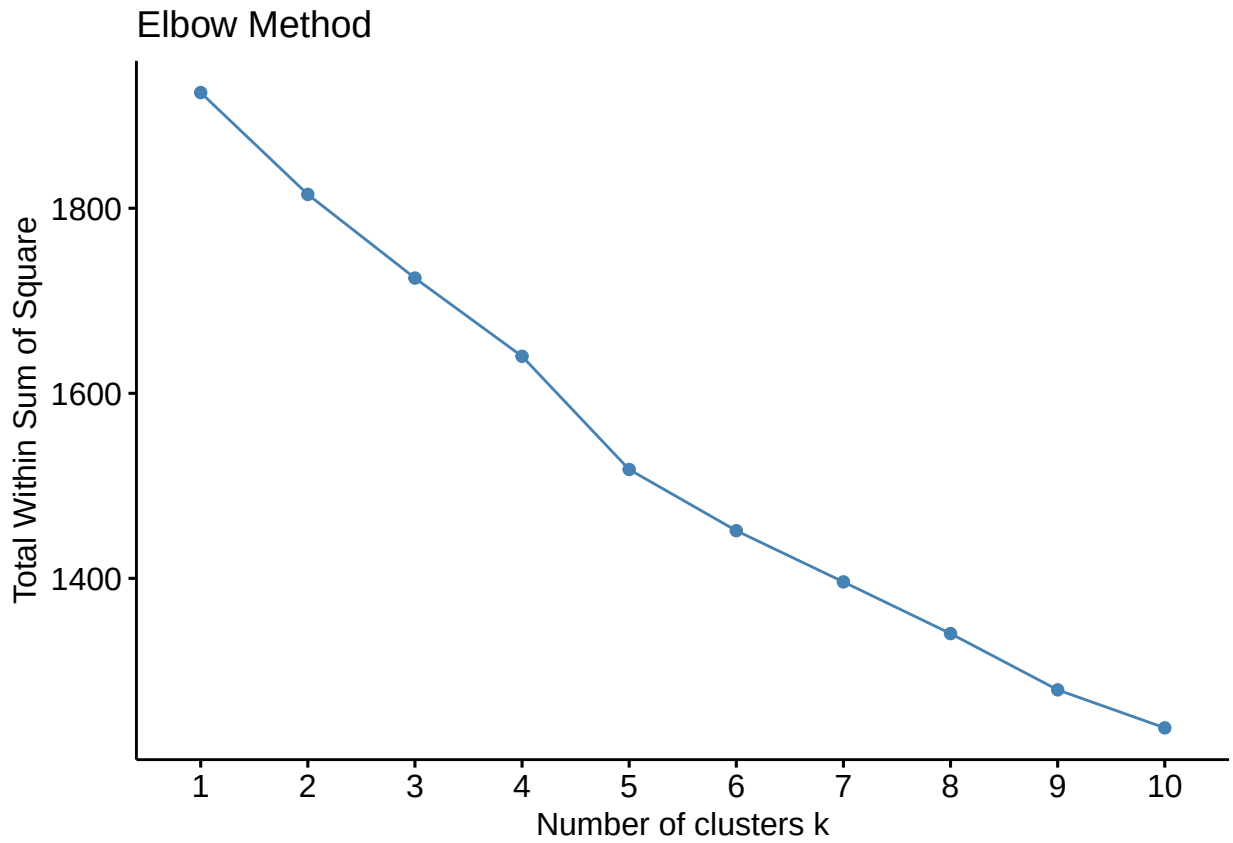
```
## Warning in daisy(starwars_numeric, metric = "gower"): binary variable(s) 3, 4,
## 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
## 26, 27, 28, 29, 30, 31, 32, 33, 34, 35 treated as interval scaled
```

```
hac_clusters <- cutree(hclust(gower_dist, method = "average"), k = 2)

#  run k-means clustering
set.seed(123)
```

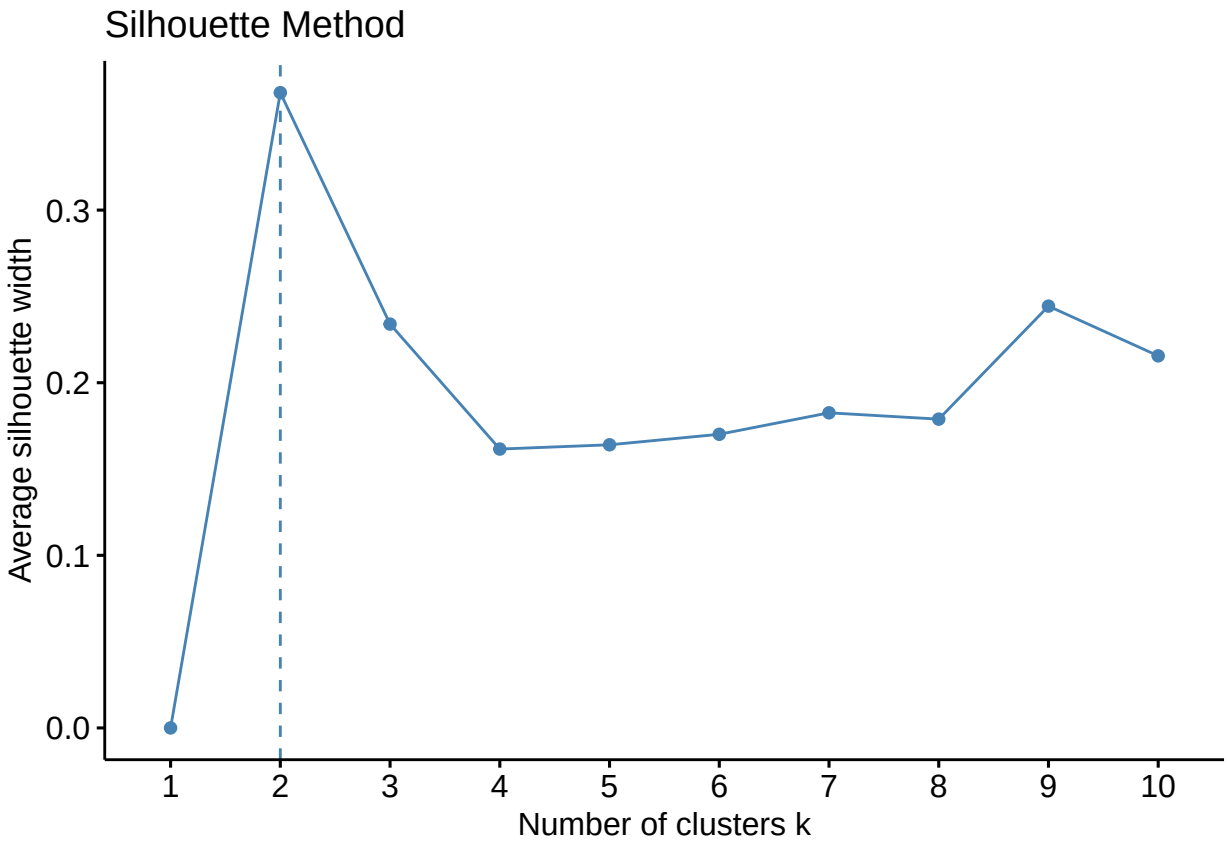
```r
kmeans_clusters <- kmeans(starwars_scaled, centers = 2, nstart = 25)$cluster

#  create cross-tabulation
table(HAC = hac_clusters, KMeans = kmeans_clusters)
```

```
##    KMeans
## HAC  1  2
##   1  0 47
##   2  9  0
```

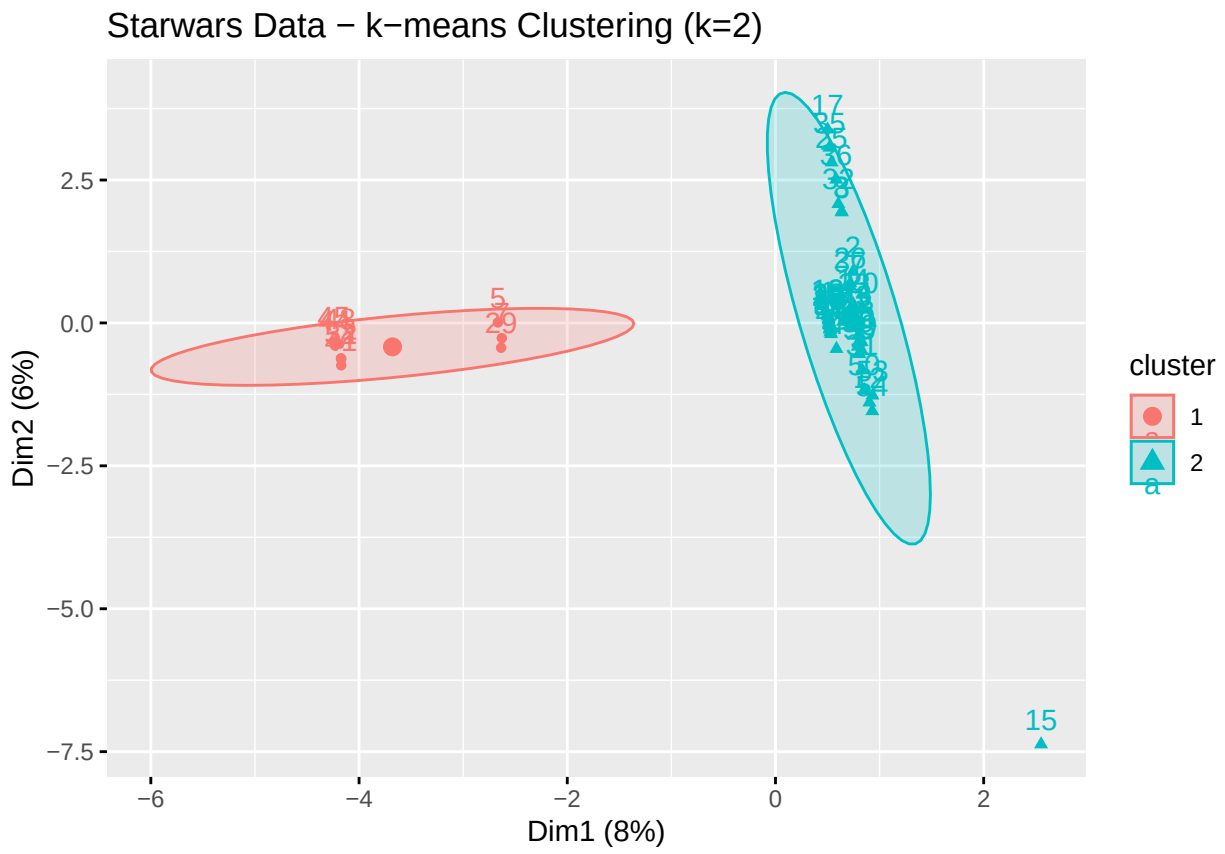
```r
library(factoextra)

# Standardize data clearly
starwars_scaled <- scale(starwars_numeric)

# Elbow method
fviz_nbclust(starwars_scaled, kmeans, method = "wss") +
  ggtitle("Elbow Method")
```




```r
# Silhouette method
fviz_nbclust(starwars_scaled, kmeans, method = "silhouette") +
  ggtitle("Silhouette Method")
```




```r
set.seed(123)
# Run k-means clustering with clearly selected k=2
final_kmeans <- kmeans(starwars_scaled, centers = 2, nstart = 25)

# Visualize clusters clearly using PCA
fviz_cluster(final_kmeans, data = starwars_scaled,
             ellipse.type = "norm",
             main = "Starwars Data - k-means Clustering (k=2)")
```



Problem 4(d)

```
library(cluster)

# calculate Gower distance
gower_dist <- daisy(starwars_numeric, metric = "gower")
```

```
## Warning in daisy(starwars_numeric, metric = "gower"): binary variable(s) 3, 4,
## 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25,
## 26, 27, 28, 29, 30, 31, 32, 33, 34, 35 treated as interval scaled
```

```
# run hierarchical clustering (average linkage)
hac_clusters <- hclust(gower_dist, method = "average")

# cut tree at k=2 explicitly clearly
hac_clusters <- cutree(hclust(gower_dist, method = "average"), k = 2)

# run k-means clusters (already calculated before)
set.seed(123)
kmeans_clusters <- kmeans(starwars_numeric, centers = 2)$cluster

# create a cross-tabulation explicitly comparing clusters
table(HAC = hac_clusters, Kmeans = kmeans_clusters)
```

```
##    Kmeans
## HAC  1  2
##   1 46  1
##   2  9  0
```

Most observations (46 out of 56) fall into cluster 1 in both HAC and k-means. Identified discrepancies (total 10): 9 observations assigned differently by HAC (cluster 2) compared to k-means (cluster 1). 1 observation is differently clustered (HAC cluster 1, k-means cluster 2). This indicates ~82% agreement clearly between HAC and k-means methods:

HAC (Gower distance) explicitly handles mixed data. k-means (numeric with dummy variables) is more affected by numeric differences.