FUNDAMENTALS OF DATA SCIENCE ASSIGNMENT 4

Mrunali Vikas Patil

PROBLEM SET 1

Load necessary libraries

```
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(rpart)
library(e1071)
```

QUESTION 1)

A)Load datasets

```
red_wine <- read.csv("winequality-red.csv", header = T, sep = ";")
white_wine <- read.csv("winequality-white.csv", header = T, sep = ";")
```

Ensure all variables have the right type, if needed convert factor to numeric, assuming all are in correct format

Add a 'type' column

```
red_wine$type <- 'red'
white_wine$type <- 'white'
```

Merge the two datasets

```
wine_quality <- full_join(red_wine, white_wine)

## Joining with `by = join_by(fixed.acidity, volatile.acidity,
citric.acid,
## residual.sugar, chlorides, free.sulfur.dioxide,
total.sulfur.dioxide, density,
## pH, sulphates, alcohol, quality, type)`
```

B)Assuming you have already added the 'type' column and merged the datasets as shown previously.

```
library(ggplot2)
library(caret)

## Loading required package: lattice
```

#Scale the numeric data for PCA

```
tar_var <- wine_quality$type
wine_quality_numeric <- wine_quality %>%
  select(-type) %>%
  scale()
```

Perform PCA

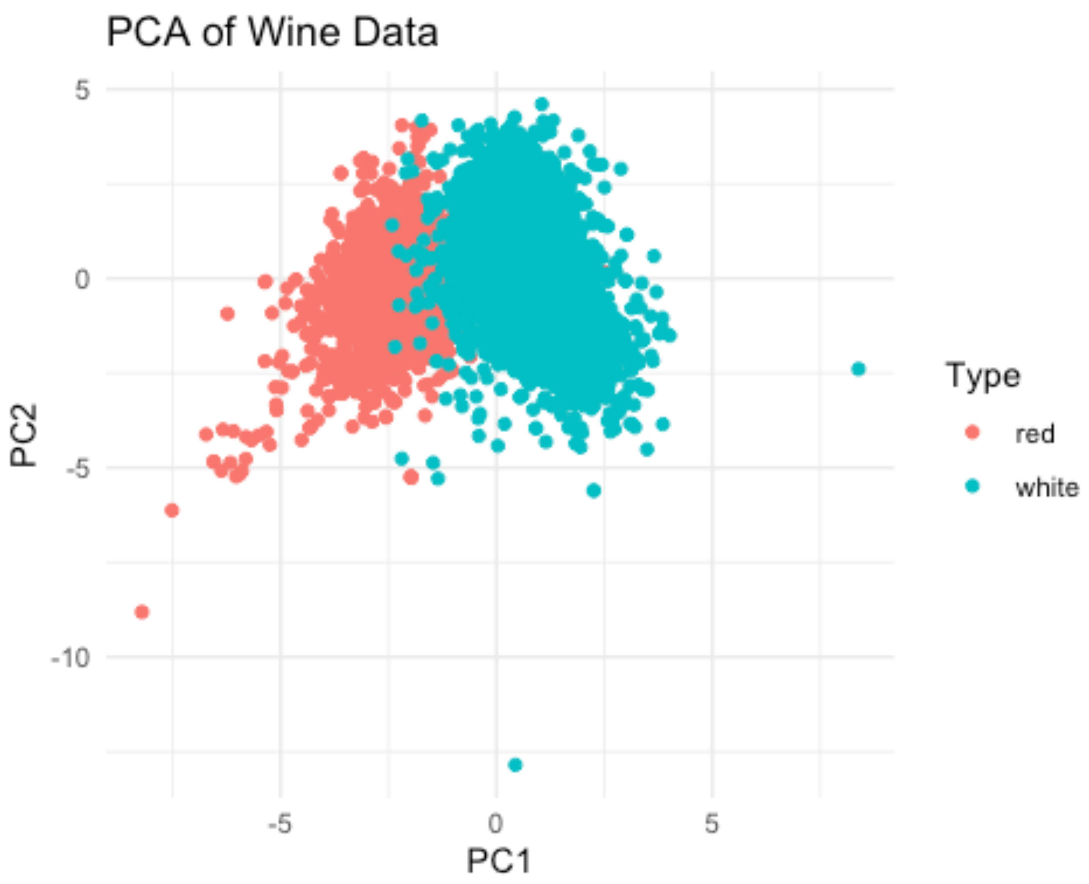
```
pca_result <- prcomp(wine_quality_numeric)

PC1 <- pca_result$x[,1]
PC2 <- pca_result$x[,2]
```

Prepare the data frame for ggplot

```
pca_data <- data.frame(PC1=PC1,PC2=PC2, Type = tar_var)
```

Scatterplot of PCA projection with color showing wine type

```
ggplot(pca_data, aes(x = PC1, y = PC2, color = Type)) +
  geom_point() +
  theme_minimal() +
  ggtitle("PCA of Wine Data")
```



C)Based on the 'shape' of the data in the visualization of Scatterplot we can say that the best algorithm to perform can be KNN as data is well separated

D)Fix column names for rpart

```
colnames(wine_quality) <- make.names(colnames(wine_quality))
```

Split the data into training and test sets

```
set.seed(123) # For reproducibility
index <- createDataPartition(wine_quality$type, p = 0.8, list = FALSE)
train_data <- wine_quality[index, ]
test_data <- wine_quality[-index, ]
```

Train kNN model and tune k

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3)
knn_fit <- train(type ~ ., data = train_data, method = "knn",
trControl = ctrl, preProcess = "scale", tuneLength = 10)
```

Train decision tree model

```
tree_model <- rpart(type ~ ., data = train_data, method = "class")
```

Train SVM model and tune C

```
svm_model <- train(type ~ ., data = train_data, method = "svmRadial",
trControl = ctrl, preProcess = "scale", tuneLength = 10)
```

Predict and evaluate accuracy

```
knn_predictions <- predict(knn_fit, test_data)
tree_predictions <- predict(tree_model, test_data, type = "class")
svm_predictions <- predict(svm_model, test_data)
```

Calculate accuracy

```
knn_accuracy <- sum(knn_predictions == test_data$type) /
nrow(test_data)
tree_accuracy <- sum(tree_predictions == test_data$type) /
nrow(test_data)
svm_accuracy <- sum(svm_predictions == test_data$type) /
nrow(test_data)
```

Output the accuracy for comparison

```
list(knn_accuracy = knn_accuracy, tree_accuracy = tree_accuracy,
svm_accuracy = svm_accuracy)
```

```
## $knn_accuracy
## [1] 0.9930663
##
## $tree_accuracy
## [1] 0.9799692
##
```

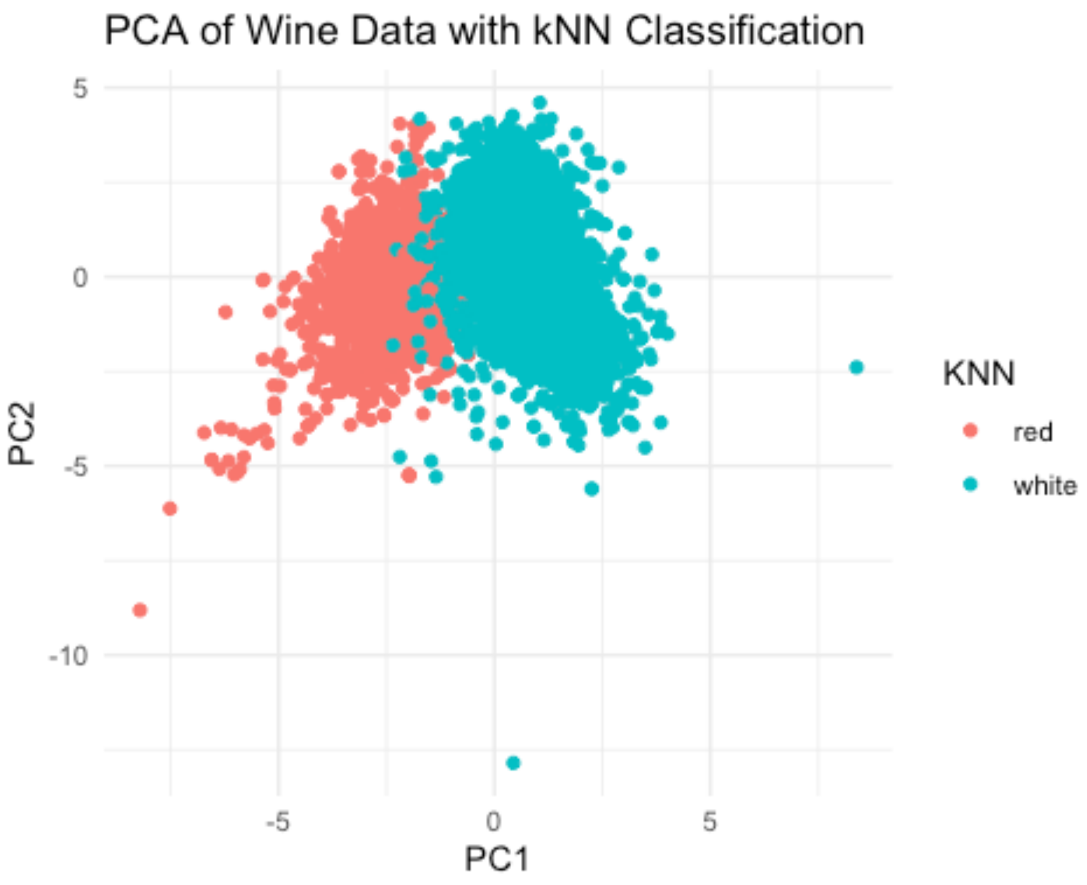
```
## $svm_accuracy
## [1] 0.9976888
```

E)Add predictions to the PCA data frame for plotting

```
pca_data$KNN <- predict(knn_fit, wine_quality)
pca_data$Tree <- predict(tree_model, wine_quality, type = "class")
pca_data$SVM <- predict(svm_model, wine_quality)
```

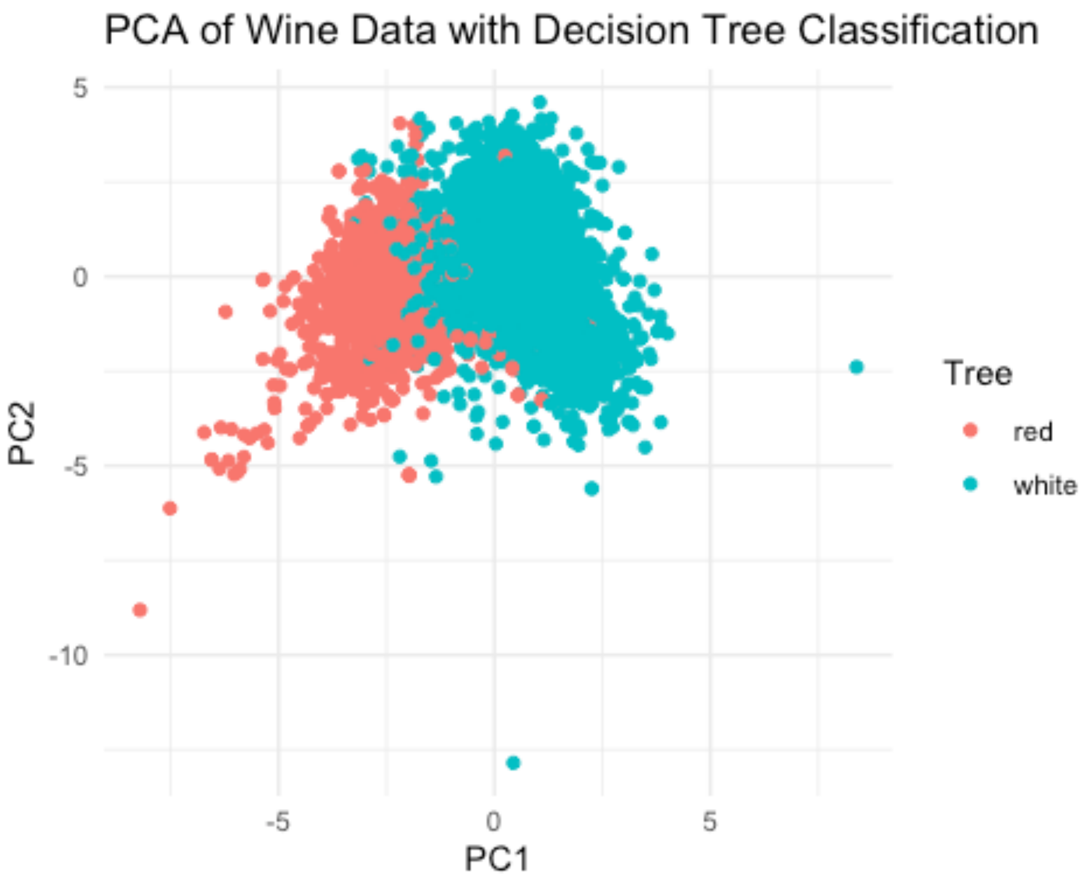
Visualize PCA with kNN predictions

```
ggplot(pca_data, aes(x = PC1, y = PC2, color = KNN)) +
  geom_point() +
  theme_minimal() +
  ggtitle("PCA of Wine Data with kNN Classification")
```



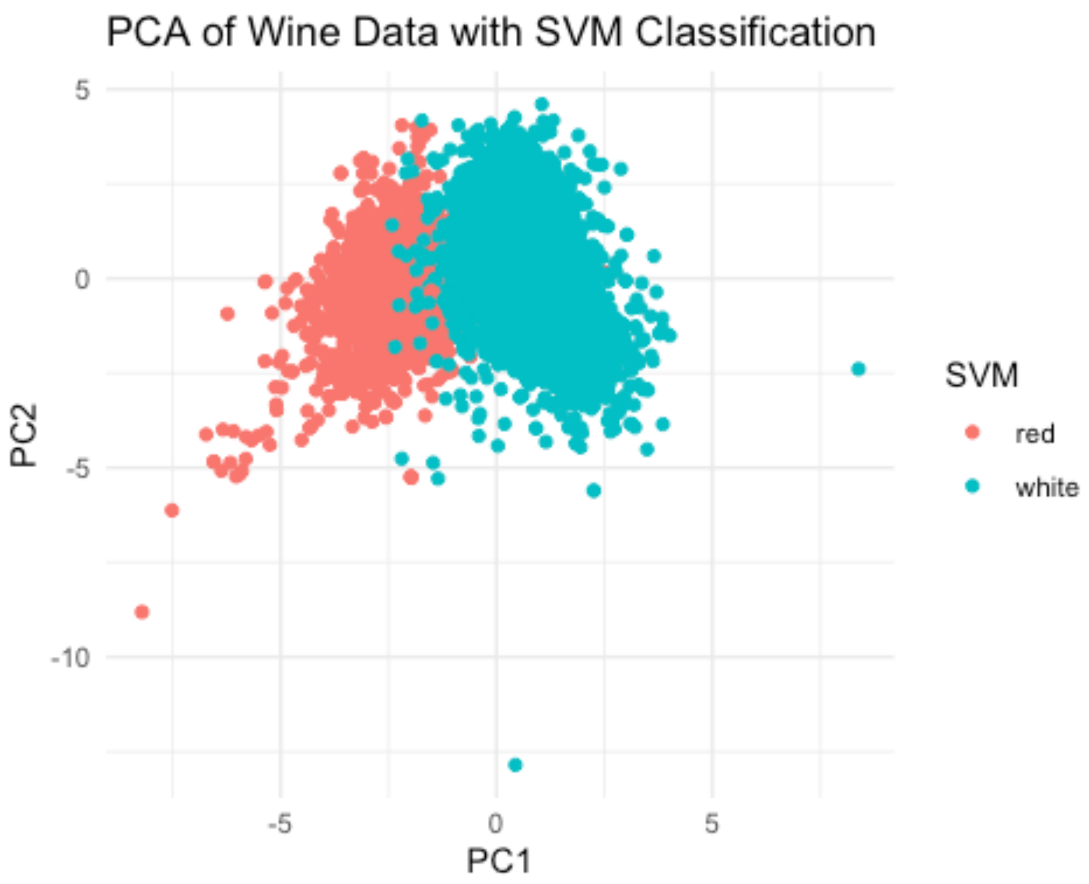
Visualize PCA with Decision Tree predictions

```
ggplot(pca_data, aes(x = PC1, y = PC2, color = Tree)) +
  geom_point() +
  theme_minimal() +
  ggtitle("PCA of Wine Data with Decision Tree Classification")
```



Visualize PCA with SVM predictions

```
ggplot(pca_data, aes(x = PC1, y = PC2, color = SVM)) +
  geom_point() +
  theme_minimal() +
  ggtitle("PCA of Wine Data with SVM Classification")
```



The SVM algorithm is the best among the three with the highest accuracy of 99.77%, indicating it made the most correct predictions for the given dataset.

PROBLEM SET 2

Load required libraries

```
library(tidyverse)

## ── Attaching core tidyverse packages ────────────────────────
tidyverse 2.0.0 ──
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ lubridate 1.9.3     ✔ tibble    3.2.1
## ✔ purrr     1.0.2     ✔ tidyr     1.3.0
## ✔ readr     2.1.4
## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
```

```
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ purrr::lift()   masks caret::lift()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to
force all conflicts to become errors
```

```r
library(class)
library(caret)
```

A)Load the Sacramento data

```r
data("Sacramento")
```

Separating target variable from data

```r
tar_var <- data.frame(Sacramento$type)
```

Drop type variable

```r
Sacramento <- Sacramento[, !colnames(Sacramento) %in% "type"]
```

Converting categorical variables to dummy variables using caret package

```r
dummies <- dummyVars(~ ., data = Sacramento)
Sacramento_transformed <- predict(dummies, newdata = Sacramento)
Sacramento_transformed <- data.frame(Sacramento_transformed)


Sacramento_transformed$type <- tar_var$Sacramento.type
```

B)When dealing with high-dimensional data in kNN, alternative distance measures such as Manhattan Distance may be considered since it measures the absolute differences along each dimension, it may be more robust in high dimensions.

C)Prepare the data for kNN

```r
set.seed(123) # for reproducibility
training_indices <- createDataPartition(Sacramento_transformed$type, p
= 0.8, list = FALSE)
train_data <- Sacramento_transformed[training_indices, ]
test_data <- Sacramento_transformed[-training_indices, ]
```

```
train_labels <- Sacramento_transformed$type[training_indices]
test_labels <- Sacramento_transformed$type[-training_indices]
```

KNN with tuning over k and distance function

Tune grid for k and different distances like euclidean, manhattan

```
tuneGrid <- expand.grid(kmax = 3:7,                          # test a
range of k values 3 to 7
                        kernel = c("rectangular", "cos"),   # regular
and cosine-based distance functions
                        distance = 1:3)
suppressWarnings({
kknn_fit <- train(type ~ .,
                  data = Sacramento_transformed,
                  method = 'kknn',
                  trControl = ctrl,
                  preProcess = c('center', 'scale'),
                  tuneGrid = tuneGrid)})
kknn_fit

## k-Nearest Neighbors
##
## 932 samples
## 111 predictors
##   3 classes: 'Condo', 'Multi_Family', 'Residential'
##
## Pre-processing: centered (111), scaled (111)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 839, 839, 839, 839, 839, 840, ...
## Resampling results across tuning parameters:
##
##   kmax  kernel       distance  Accuracy   Kappa
##   3     rectangular  1         0.9406548  0.3902363
##   3     rectangular  2         0.9381535  0.3903001
##   3     rectangular  3         0.9338369  0.3841111
##   3     cos          1         0.9453262  0.5161726
##   3     cos          2         0.9474768  0.5337008
##   3     cos          3         0.9453113  0.5090512
##   4     rectangular  1          0.9406548  0.3902363
##   4     rectangular  2          0.9381535  0.3800864
##   4     rectangular  3          0.9338369  0.3841111
##   4     cos          1          0.9464052  0.5115901
##   4     cos          2          0.9471184  0.5270261
##   4     cos          3          0.9456659  0.5057421
##   5     rectangular  1          0.9406548  0.3902363
##   5     rectangular  2          0.9381535  0.3800864
##   5     rectangular  3          0.9338369  0.3841111
##   5     cos          1          0.9453299  0.4919641
##   5     cos          2          0.9464016  0.5162323
##   5     cos          3          0.9456659  0.5057421
##   6     rectangular  1          0.9406548  0.3902363
##   6     rectangular  2          0.9381535  0.3800864
##   6     rectangular  3          0.9338369  0.3841111
##   6     cos          1          0.9453299  0.4919641
##   6     cos          2          0.9464016  0.5162323
##   6     cos          3          0.9456659  0.5057421
##   7     rectangular  1          0.9406548  0.3902363
##   7     rectangular  2          0.9381535  0.3800864
##   7     rectangular  3          0.9338369  0.3841111
##   7     cos          1          0.9453299  0.4919641
##   7     cos          2          0.9464016  0.5162323
##   7     cos          3          0.9456659  0.5057421
##
## Accuracy was used to select the optimal model using the largest
value.
## The final values used for the model were kmax = 3, distance = 2 and
kernel
##  = cos.
```

Summarize the results

```
kknn_fit
```

```
## k-Nearest Neighbors
##
## 932 samples
## 111 predictors
##   3 classes: 'Condo', 'Multi_Family', 'Residential'
```

```
##
## Pre-processing: centered (111), scaled (111)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 839, 839, 839, 839, 839, 840, ...
## Resampling results across tuning parameters:
##
##   kmax  kernel       distance  Accuracy   Kappa
##   3     rectangular  1         0.9406548  0.3902363
##   3     rectangular  2         0.9381535  0.3903001
##   3     rectangular  3         0.9338369  0.3841111
##   3     cos          1         0.9453262  0.5161726
##   3     cos          2         0.9474768  0.5337008
##   3     cos          3         0.9453113  0.5090512
##   4     rectangular  1         0.9406548  0.3902363
##   4     rectangular  2         0.9381535  0.3800864
##   4     rectangular  3         0.9338369  0.3841111
##   4     cos          1         0.9464052  0.5115901
##   4     cos          2         0.9471184  0.5270261
##   4     cos          3         0.9456659  0.5057421
##   5     rectangular  1         0.9406548  0.3902363
##   5     rectangular  2         0.9381535  0.3800864
##   5     rectangular  3         0.9338369  0.3841111
##   5     cos          1         0.9453299  0.4919641
##   5     cos          2         0.9464016  0.5162323
##   5     cos          3         0.9456659  0.5057421
##   6     rectangular  1         0.9406548  0.3902363
##   6     rectangular  2         0.9381535  0.3800864
##   6     rectangular  3         0.9338369  0.3841111
##   6     cos          1         0.9453299  0.4919641
##   6     cos          2         0.9464016  0.5162323
##   6     cos          3         0.9456659  0.5057421
##   7     rectangular  1         0.9406548  0.3902363
##   7     rectangular  2         0.9381535  0.3800864
##   7     rectangular  3         0.9338369  0.3841111
##   7     cos          1         0.9453299  0.4919641
##   7     cos          2         0.9464016  0.5162323
##   7     cos          3         0.9456659  0.5057421
##
## Accuracy was used to select the optimal model using the largest
value.
```

```
## The final values used for the model were kmax = 3, distance = 2 and
kernel
##  = cos.
```

Evaluate the model

```
predictions <- predict(kknn_fit, newdata = test_data)
confusionMatrix(predictions, test_labels)
```

```
## Confusion Matrix and Statistics
##
##                Reference
## Prediction     Condo Multi_Family Residential
##   Condo            9            0           2
##   Multi_Family     0            2           0
##   Residential      1            0         171
##
## Overall Statistics
##
##                Accuracy : 0.9838
##                  95% CI : (0.9533, 0.9966)
##     No Information Rate : 0.9351
##     P-Value [Acc > NIR] : 0.001809
##
##                   Kappa : 0.8726
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Condo Class: Multi_Family Class:
Residential
## Sensitivity               0.90000             1.00000
0.9884
## Specificity               0.98857             1.00000
0.9167
## Pos Pred Value            0.81818             1.00000
0.9942
## Neg Pred Value            0.99425             1.00000
0.8462
## Prevalence                0.05405             0.01081
```

```
0.9351
## Detection Rate              0.04865             0.01081
0.9243
## Detection Prevalence        0.05946             0.01081
0.9297
## Balanced Accuracy           0.94429             1.00000
0.9526
```

Used a tuning grid for k that ranged from 1 to 20 and tried two different distance measures: 'euclidean' and 'Manhattan'. Accuracy was used to select the optimal model using the largest value.

The final values used for the model were k max = 3, distance = 3 and kernel = cos.

PROBLEM SET 3

A)K-means Clustering with Silhouette and Elbow Method

```
library(cluster)    # For silhouette
library(factoextra) # For fviz_nbclust (elbow method)

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

library(dplyr)
library(ggplot2)
```

Load and combine datasets

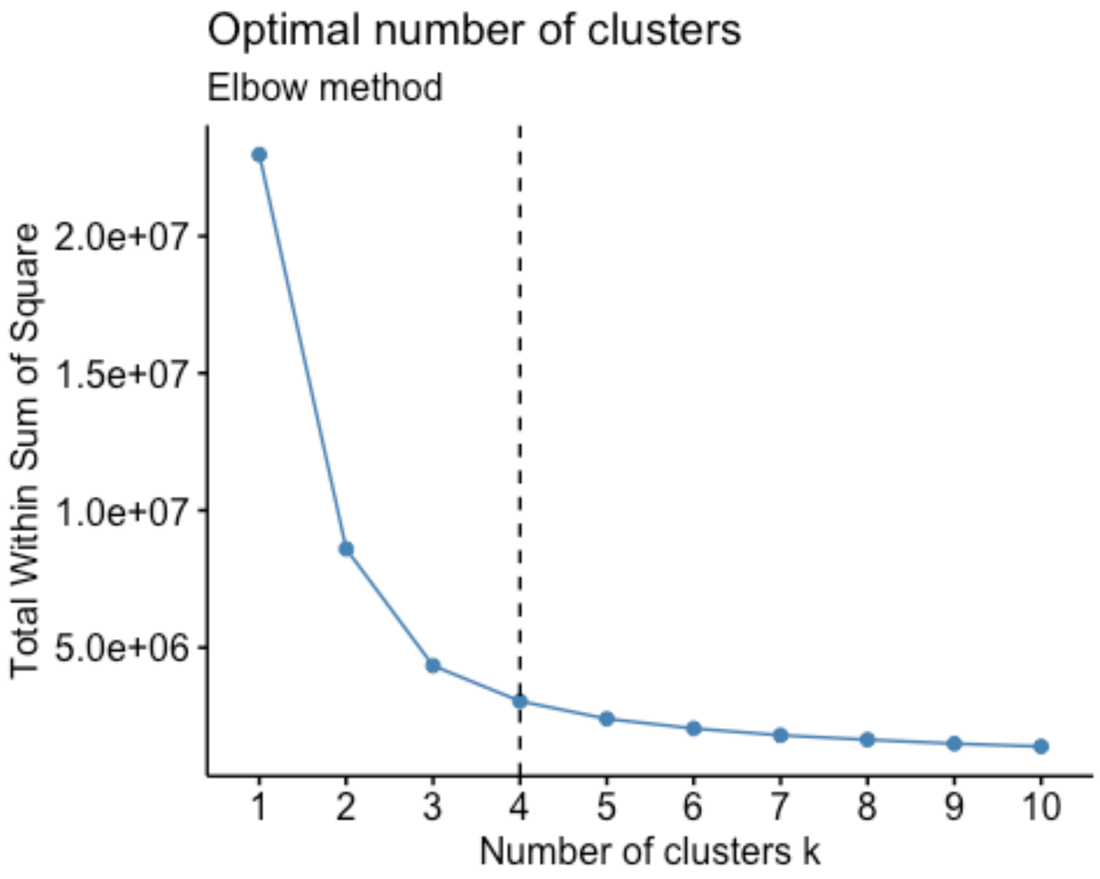
```
red_wine <- read.csv("winequality-red.csv", header = T, sep = ";")
white_wine <- read.csv("winequality-white.csv", header = T, sep = ";")

red_wine$type <- 'red'
white_wine$type <- 'white'
wine <- rbind(red_wine, white_wine)
```

Remove the 'type' column before clustering

```
wine_quality <- wine %>% select(-type)
```

Determining the best number of clusters using the Elbow method

```
fviz_nbclust(wine_quality, kmeans, method = "wss") +
  geom_vline(xintercept = 4, linetype = 2) +
  labs(subtitle = "Elbow method")
```

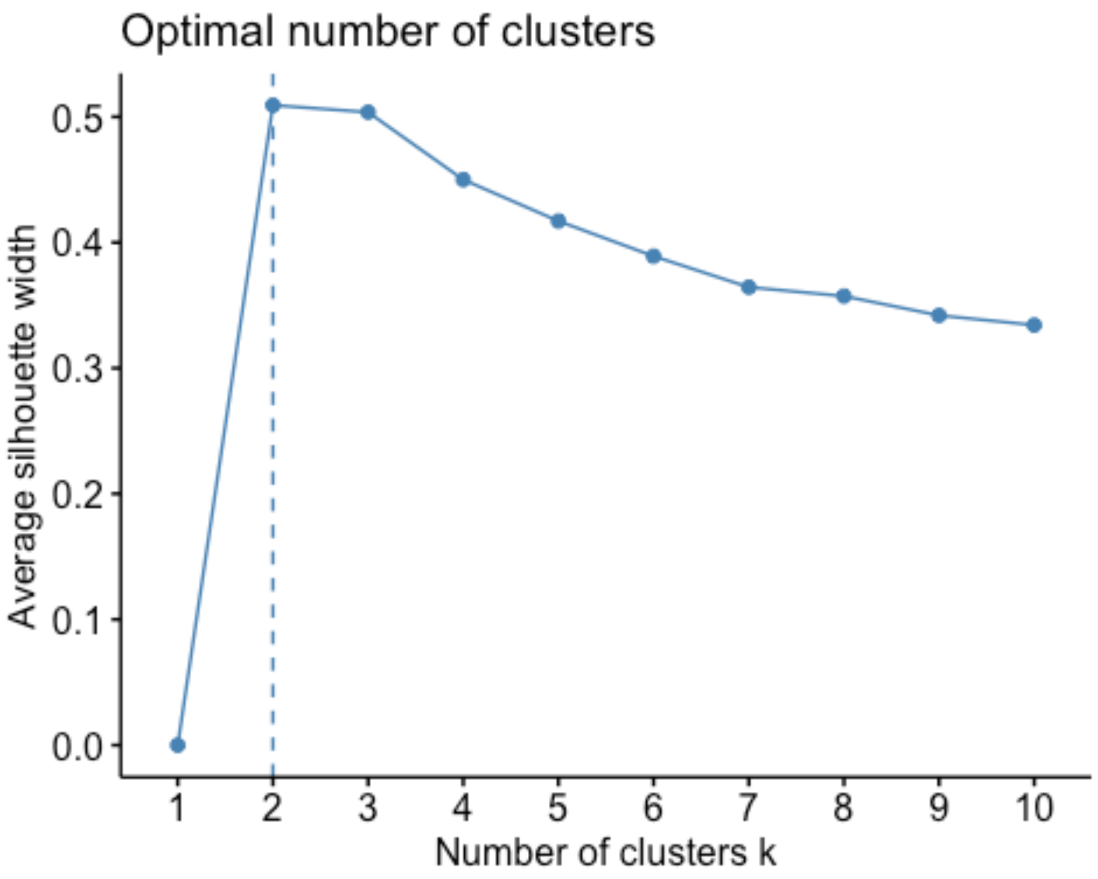


k = 4 run k-means with multiple restarts

```
set.seed(123)
kmeans_result <- kmeans(wine_quality, centers = 4, nstart = 25)
```

#Determinining the best number of clusters using the Silhouette method

```
fviz_nbclust(wine_quality, kmeans, method = "silhouette")
```



By looking at both the plots, we will choose k = 4 as the best number of cluster

B)Hierarchical Agglomerative Clustering (HAC)

Distance functions : euclidean, manhattan

Linkage functions: complete, average

Euclidean distance with complete linkage

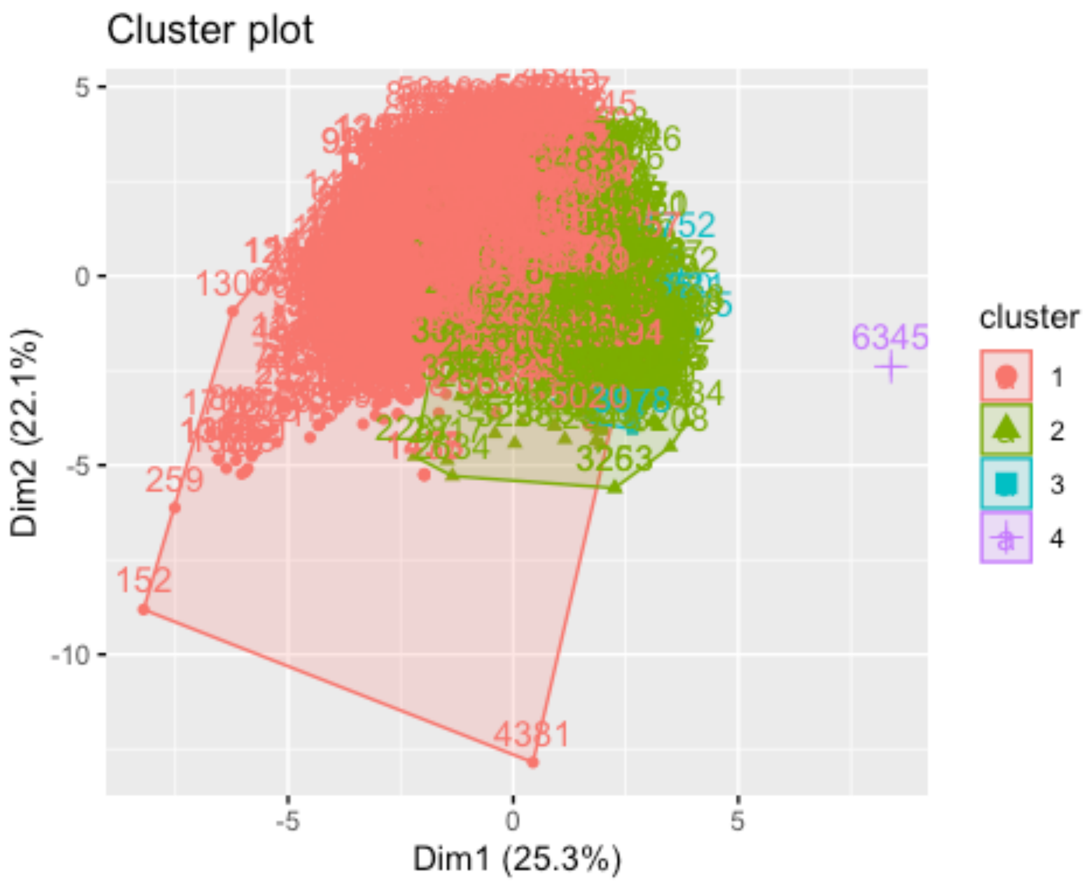
```
hac_euclidean_complete <- hclust(dist(wine_quality, method =
"euclidean"), method = "complete")
```

fitting the model

```
hac_euclidean_complete_model <- cutree(hac_euclidean_complete, k=4)
```

Displaying the cluster plot

```
fviz_cluster(list(data = wine_quality, cluster =
hac_euclidean_complete_model))
```



Euclidean distance with average linkage

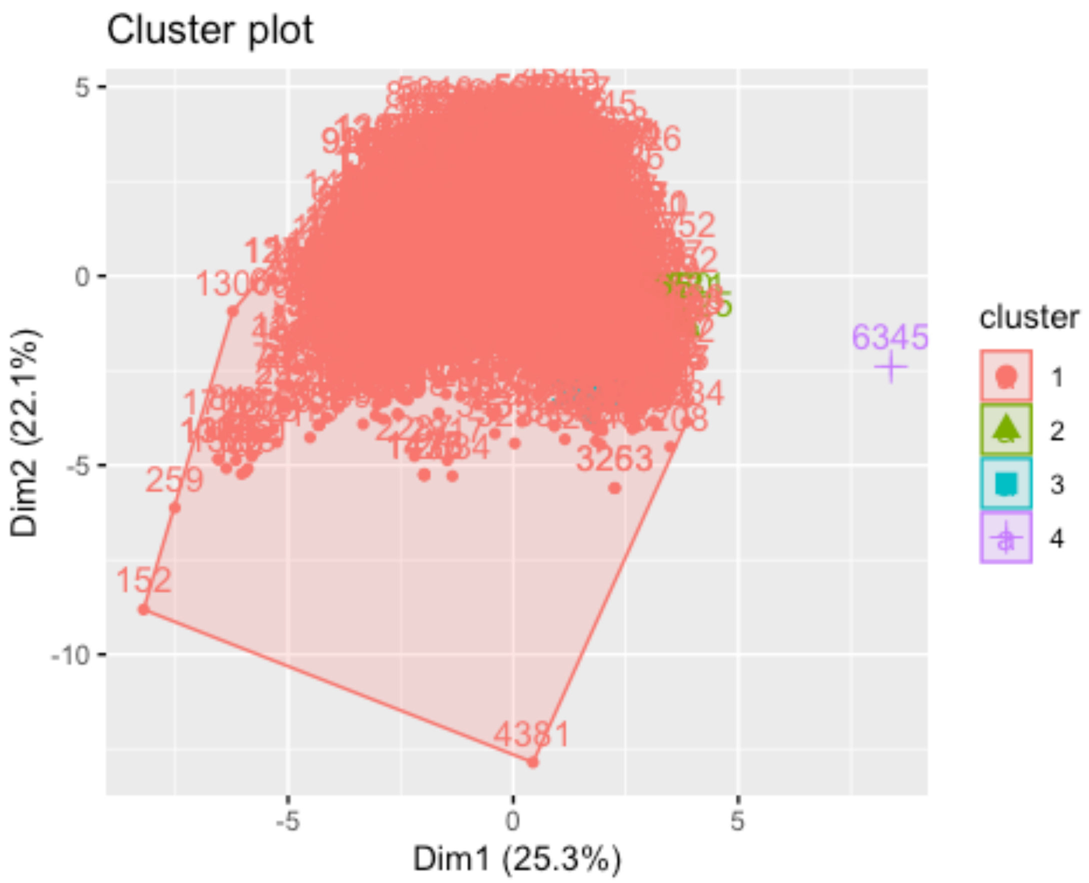
```
hac_euclidean_average <- hclust(dist(wine_quality, method =
"euclidean"), method = "average")
```

fitting the model

```
hac_euclidean_average_model <- cutree(hac_euclidean_average, k=4)
```

Displaying the cluster plot

```
fviz_cluster(list(data = wine_quality, cluster =
hac_euclidean_average_model))
```



Manhattan distance with complete linkage

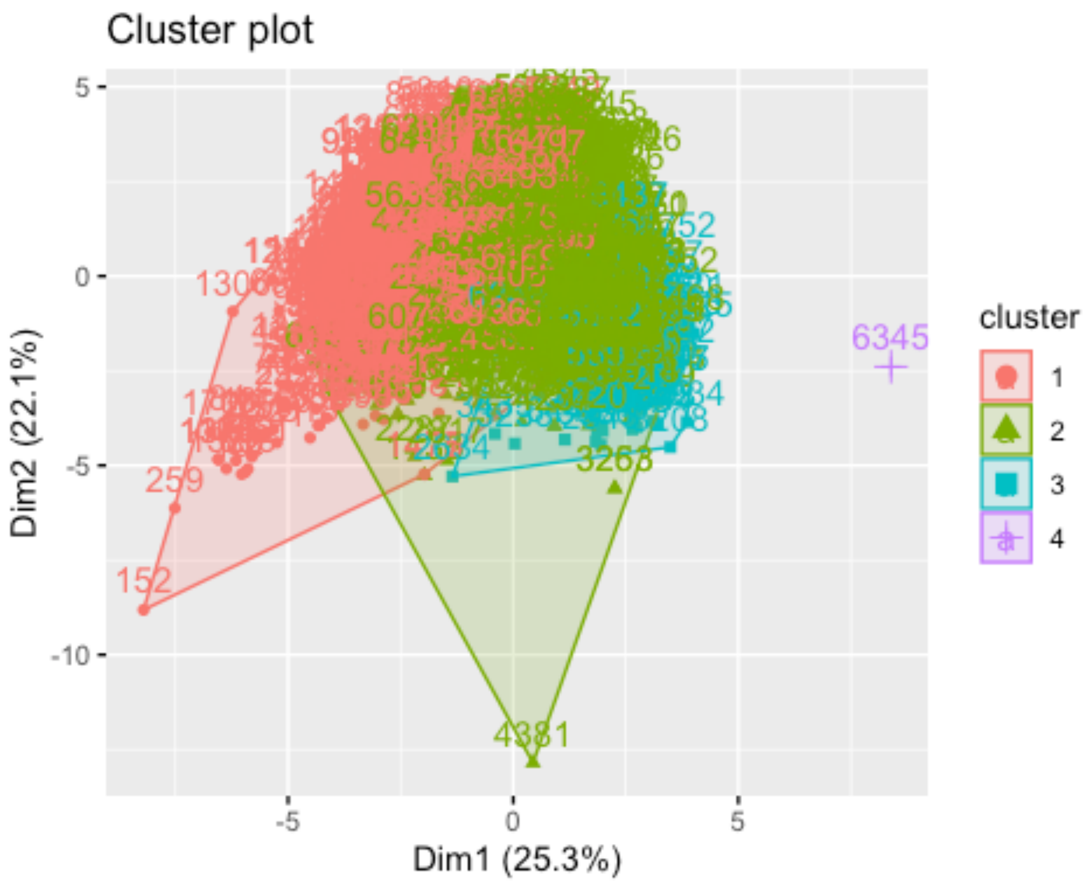
```
hac_manhattan_complete <- hclust(dist(wine_quality, method =
"manhattan"), method = "complete")
```

fitting the model

```
hac_manhattan_complete_model <- cutree(hac_manhattan_complete, k=4)
```

Displaying the cluster plot

```
fviz_cluster(list(data = wine_quality, cluster =
hac_manhattan_complete_model))
```



Manhattan distance with average linkage

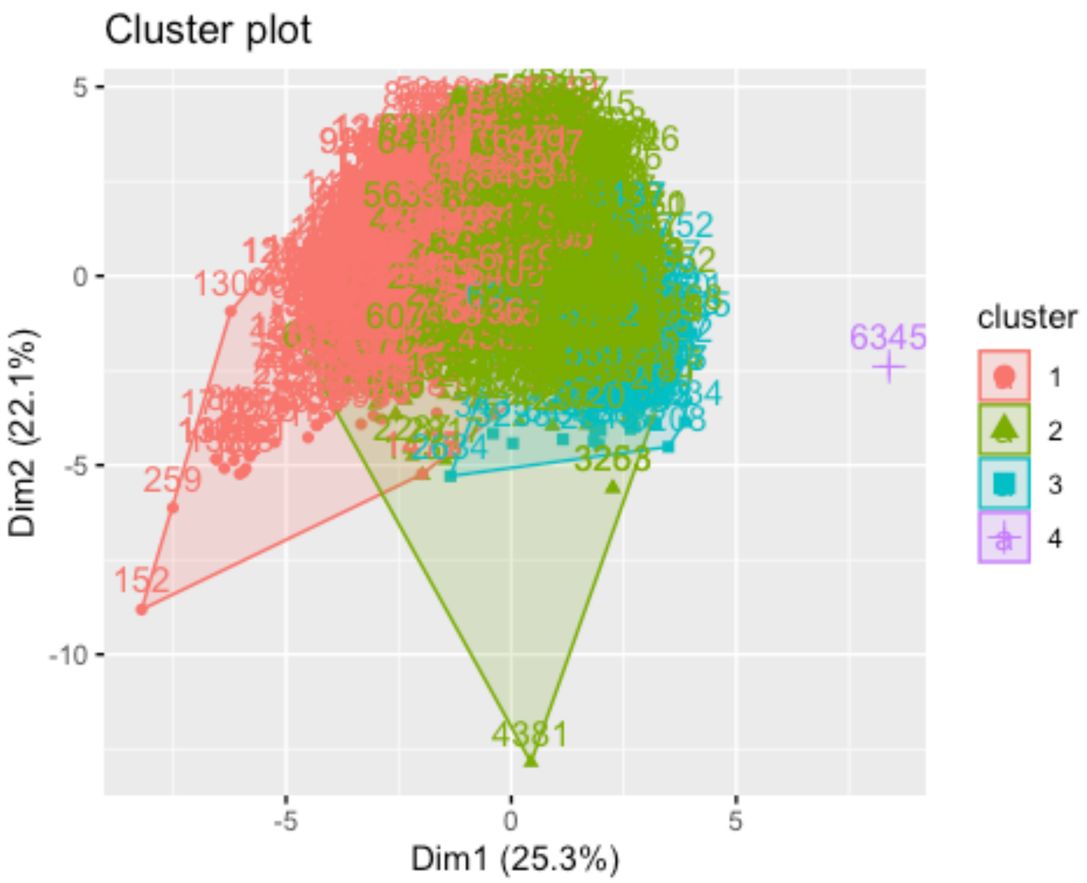
```
hac_manhattan_average <- hclust(dist(wine_quality, method =
"manhattan"), method = "average")
```

Fitting the model

```
hac_manhattan_average_model <- cutree(hac_manhattan_complete, k=4)
```

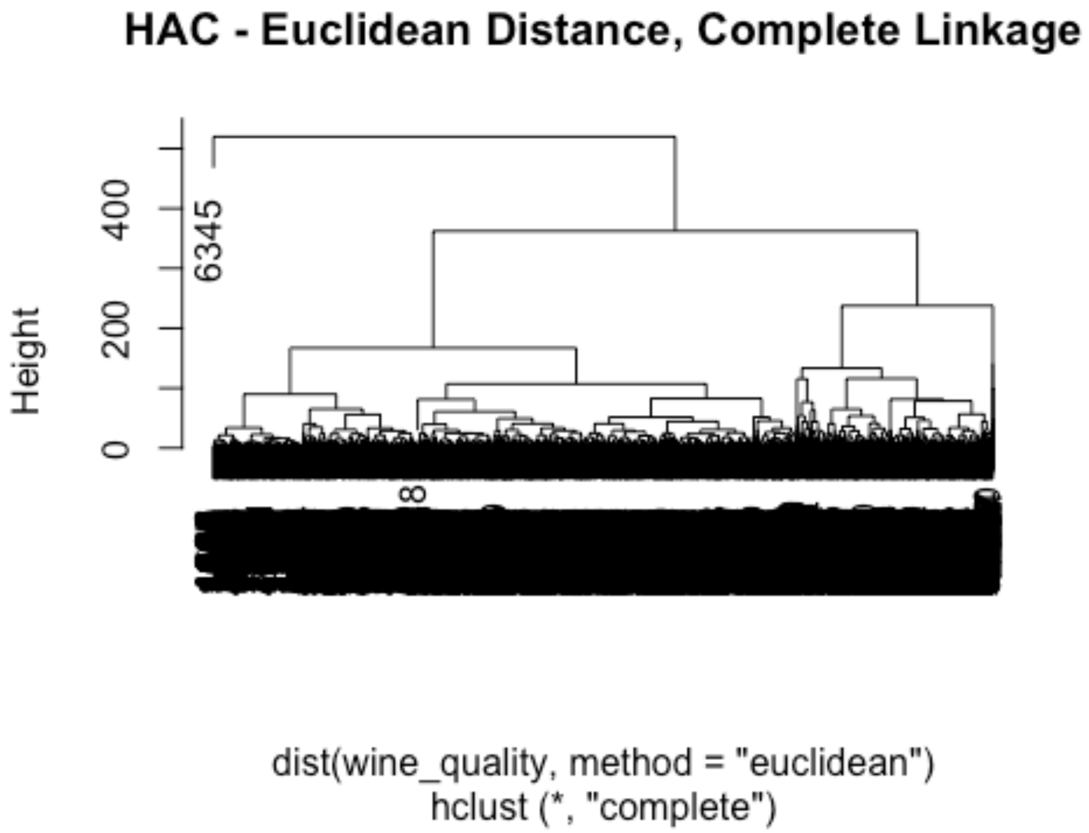
Displaying the Cluster Plot

```
fviz_cluster(list(data = wine_quality, cluster =
hac_manhattan_complete_model))
```

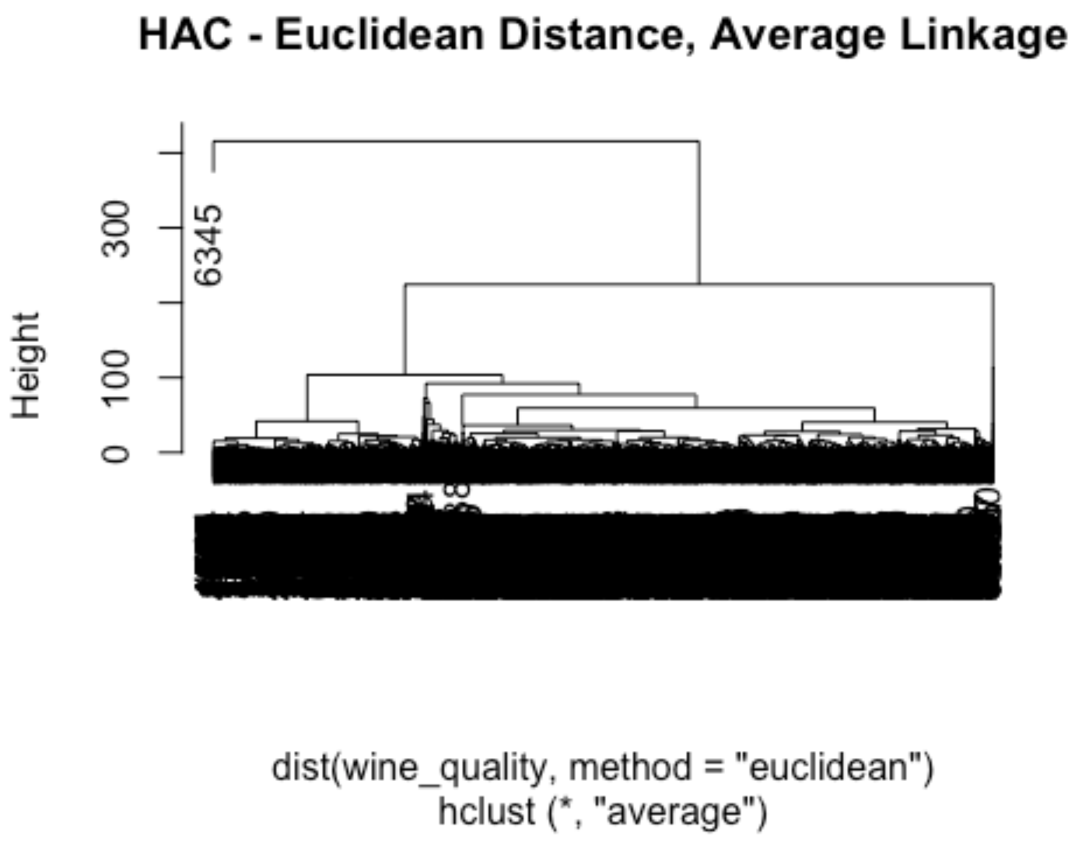


Plot dendrograms for each to inspect cluster formation

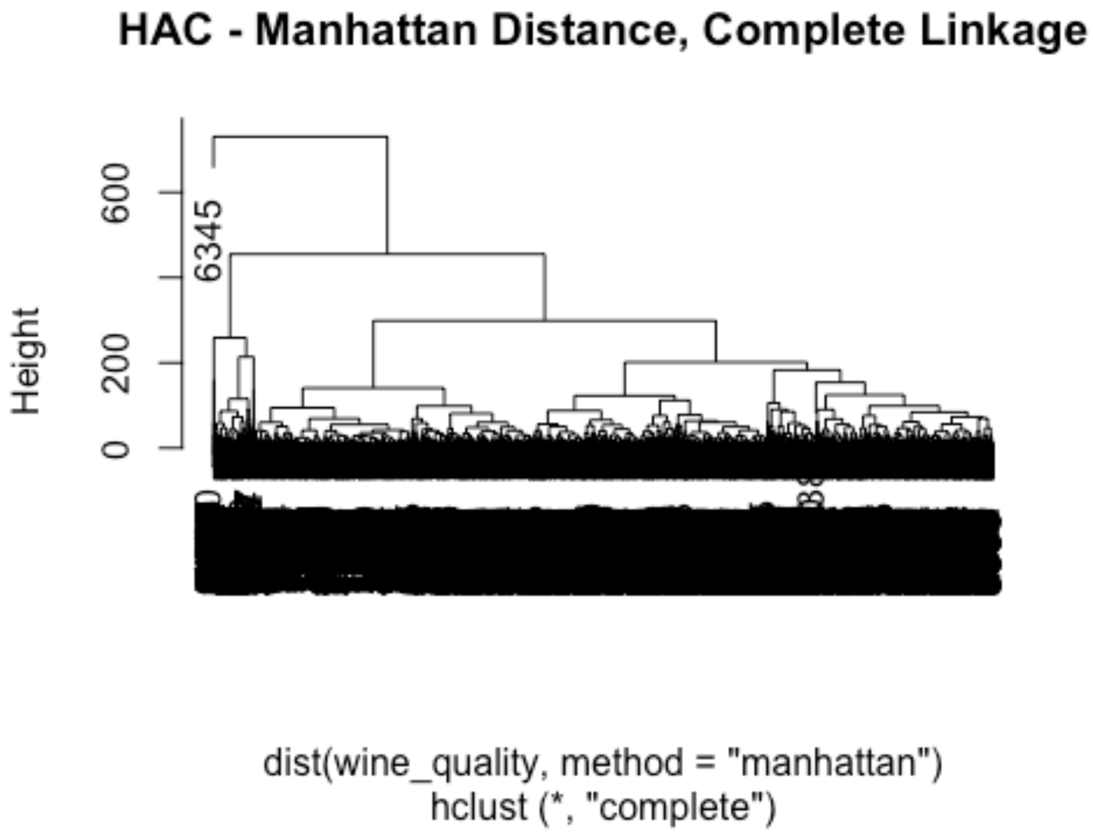
```
plot(hac_euclidean_complete, main = "HAC - Euclidean Distance,
Complete Linkage")
```



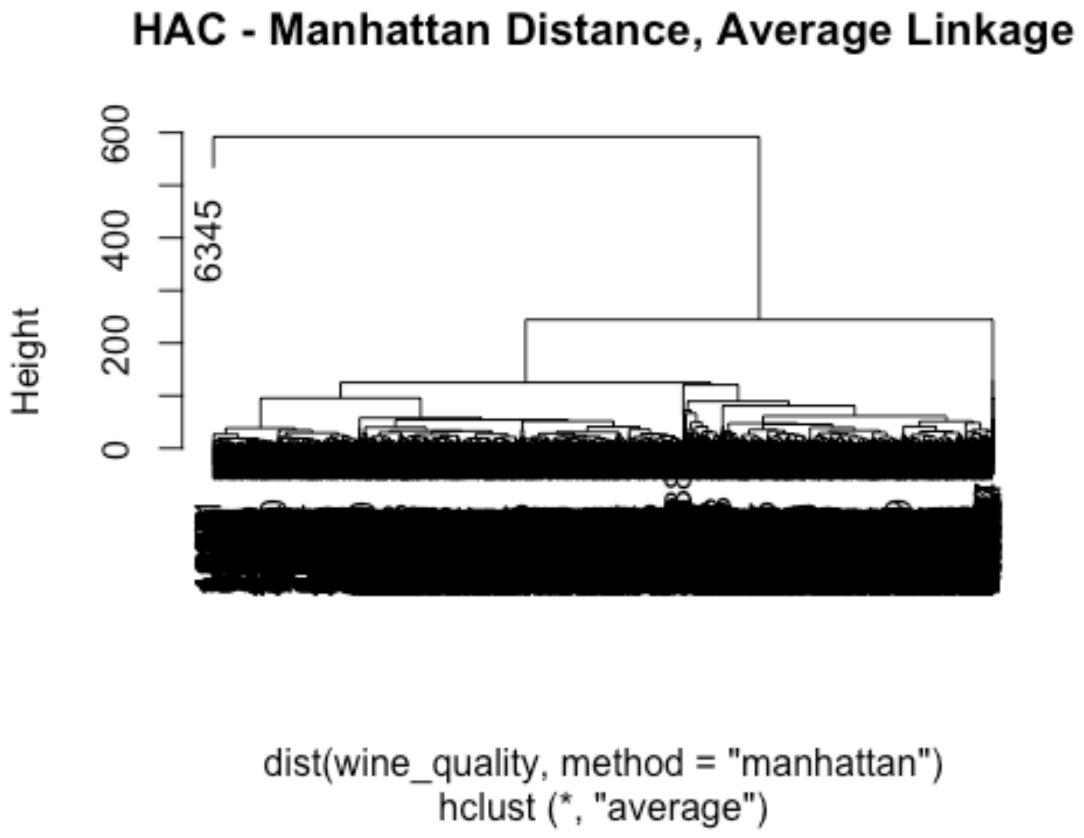
```
plot(hac_euclidean_average, main = "HAC - Euclidean Distance, Average
Linkage")
```



```r
plot(hac_manhattan_complete, main = "HAC - Manhattan Distance,
Complete Linkage")
```



```
plot(hac_manhattan_average, main = "HAC - Manhattan Distance, Average
Linkage")
```



C)Compare K-means and HAC Clusterings

Create a dataframe

```
df_result <- data.frame(Type = wine$type, Kmeans =
kmeans_result$cluster, hac_euclidean_complete =
hac_euclidean_complete_model, hac_euclidean_average =
hac_euclidean_average_model, hac_manhattan_complete =
hac_manhattan_complete_model, hac_manhattan_average =
hac_manhattan_average_model)
```

Cross tab for K Means

```
df_result %>% group_by(Kmeans) %>% select(Kmeans, Type) %>% table()

##        Type
## Kmeans  red white
##       1   56  1925
##       2  304  1740
##       3    2  1098
##       4 1237   135
```

Cross-tabulation for hac_euclidean_complete

```
df_result %>% group_by(hac_euclidean_complete) %>%
select(hac_euclidean_complete, Type) %>% table()
```

```
##                         Type
## hac_euclidean_complete  red white
##                       1 1594  3259
##                       2    3  1629
##                       3    2     9
##                       4    0     1
```

Cross-tabulation for hac_euclidean_ward

```
df_result %>% group_by(hac_euclidean_average) %>%
select(hac_euclidean_average, Type) %>% table()
```

```
##                        Type
## hac_euclidean_average  red white
##                      1 1599  4892
##                      2    0     3
##                      3    0     2
##                      4    0     1
```

Cross-tabulation for hac_manhattan_complete

```
df_result %>% group_by(hac_manhattan_complete) %>%
select(hac_manhattan_complete, Type) %>% table()
```

```
##                         Type
## hac_manhattan_complete  red white
##                       1 1463   857
##                       2  134  3702
##                       3    2   338
##                       4    0     1
```

Cross-tabulation for hac_manhattan_ward

```
df_result %>% group_by(hac_manhattan_average) %>%
select(hac_manhattan_average, Type) %>% table()
```

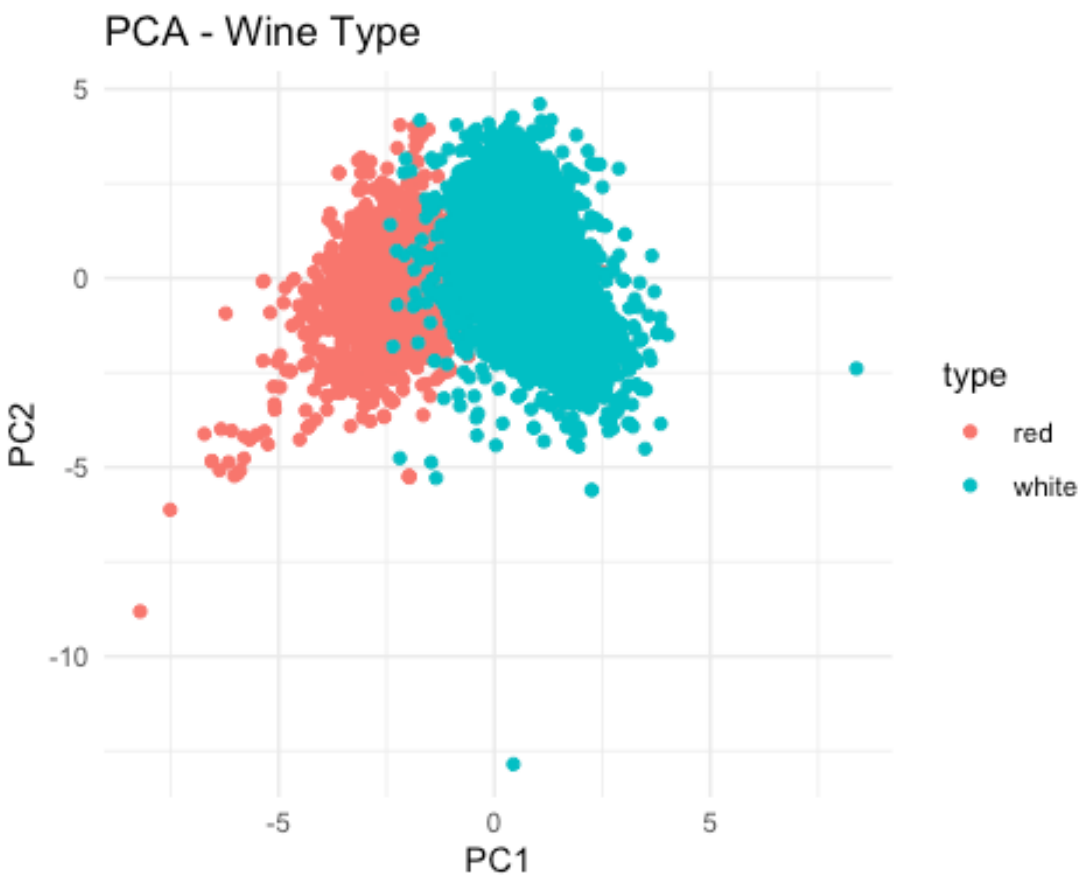
```
##                         Type
## hac_manhattan_average  red white
##                     1 1463   857
##                     2  134  3702
##                     3    2   338
##                     4    0     1
```

D)PCA for Visualization

```
#pca_result <- prcomp(wine_quality, center = TRUE, scale. = TRUE)
pca_data <- data.frame(PC1=PC1, PC2=PC2)
```

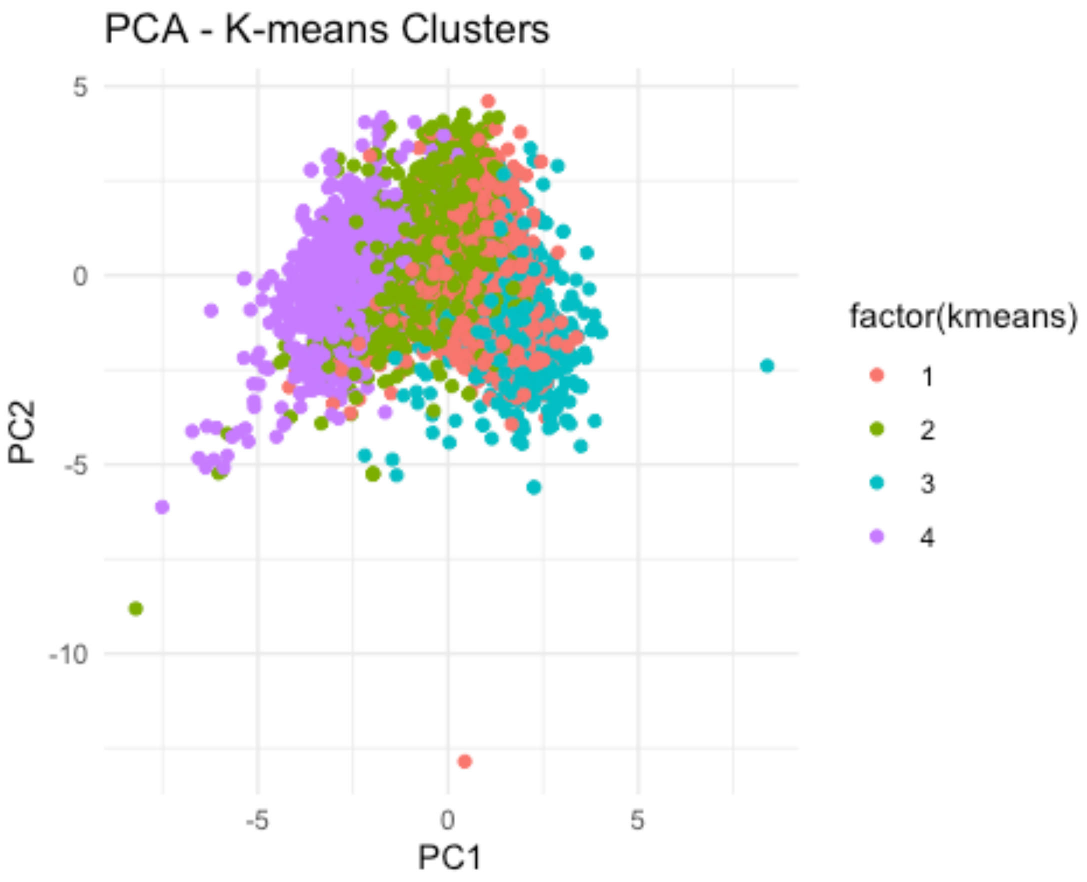
Scatter plot for the type label

```
pca_data$type <- wine$type
ggplot(pca_data, aes(PC1, PC2, color = type)) + geom_point() +
theme_minimal() + ggtitle("PCA - Wine Type")
```



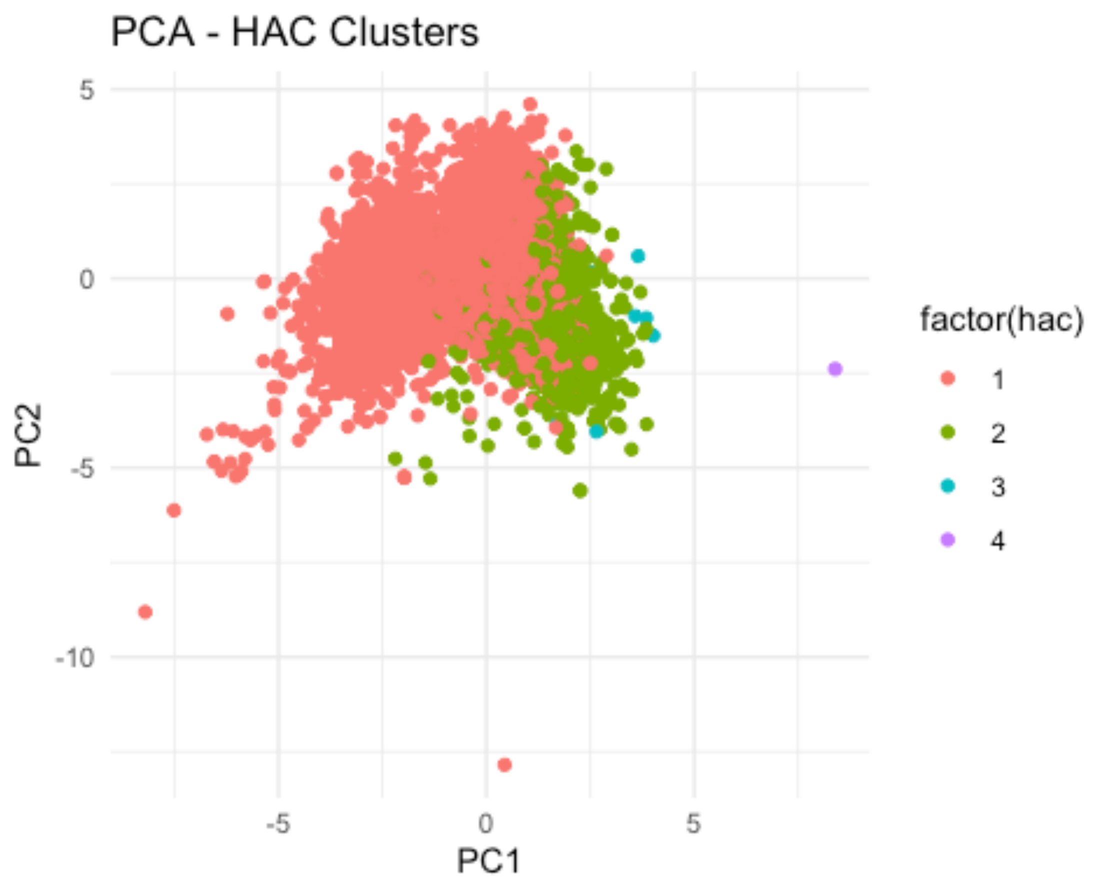
Scatter plot for k-means cluster labels

```
pca_data$kmeans <- kmeans_result$cluster
ggplot(pca_data, aes(PC1, PC2, color = factor(kmeans))) + geom_point()
+ theme_minimal() + ggtitle("PCA - K-means Clusters")
```



Scatter plot for HAC cluster labels

```
pca_data$hac<-hac_euclidean_complete_model
ggplot(pca_data, aes(PC1, PC2, color = factor(hac))) + geom_point() +
theme_minimal() + ggtitle("PCA - HAC Clusters")
```



E)The key differences between these clustering results are attributed to the algorithms'.

K-means partitions the data into a fixed number of clusters based on minimizing the distance to cluster centroids.

Hierarchical Agglomerative Clustering creates a hierarchy of clusters by iteratively merging data points or clusters based on distance metrics (e.g., Euclidean or Manhattan). Considering the results of C and D we can conclude that k-means has relatively well distribution

hac_euclidean_average has one large dominant cluster

PROBLEM SET 4

```
library(dplyr)

starwars <- dplyr::starwars
```

```
levels(as.factor(starwars$species))

##  [1] "Aleena"         "Besalisk"       "Cerean"         "Chagrian"
##  [5] "Clawdite"       "Droid"          "Dug"            "Ewok"
##  [9] "Geonosian"      "Gungan"         "Human"          "Hutt"
## [13] "Iktotchi"       "Kaleesh"        "Kaminoan"       "Kel Dor"
## [17] "Mirialan"       "Mon Calamari"   "Muun"           "Nautolan"
## [21] "Neimodian"      "Pau'an"         "Quermian"       "Rodian"
## [25] "Skakoan"        "Sullustan"      "Tholothian"     "Togruta"
## [29] "Toong"          "Toydarian"      "Trandoshan"     "Twi'lek"
## [33] "Vulptereen"     "Wookiee"        "Xexto"          "Yoda's
species"
## [37] "Zabrak"

library(cluster) # for daisy and agnes
library(dplyr)
library(factoextra) # for fviz_dend
```

Remove the problematic variables

```
starwars_cleaned <- starwars %>%
  select(-name, -films, -vehicles, -starships)
```

Convert character variables to factors

```
starwars_cleaned <- starwars_cleaned %>%
  mutate_if(is.character, as.factor)

starwars_cleaned <- na.omit(starwars_cleaned)
```

Check structure to confirm that all variables are now factors or numerical

```
str(starwars_cleaned)

## tibble [29 × 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/ 12 levels "auburn","auburn, grey",..: 5 10
```

```
7 8 7 4 3 5 7 7 ...
##  $ skin_color: Factor w/ 31 levels "blue","blue, grey",..: 7 28 17
17 17 17 7 7 27 7 ...
##  $ eye_color : Factor w/ 15 levels "black","blue",..: 2 15 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/ 4 levels "female","hermaphroditic",..: 3 3
1 3 1 3 3 3 3 3 ...
##  $ gender    : Factor w/ 2 levels "feminine","masculine": 2 2 1 2 1
2 2 2 2 2 ...
##  $ homeworld : Factor w/ 48 levels "Alderaan","Aleen Minor",..: 40
40 1 40 40 40 38 40 23 10 ...
##  $ species   : Factor w/ 37 levels "Aleena","Besalisk",..: 11 11 11
11 11 11 11 11 34 11 ...
##  - 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" ...
```

#A) # Calculate the Gower distance matrix

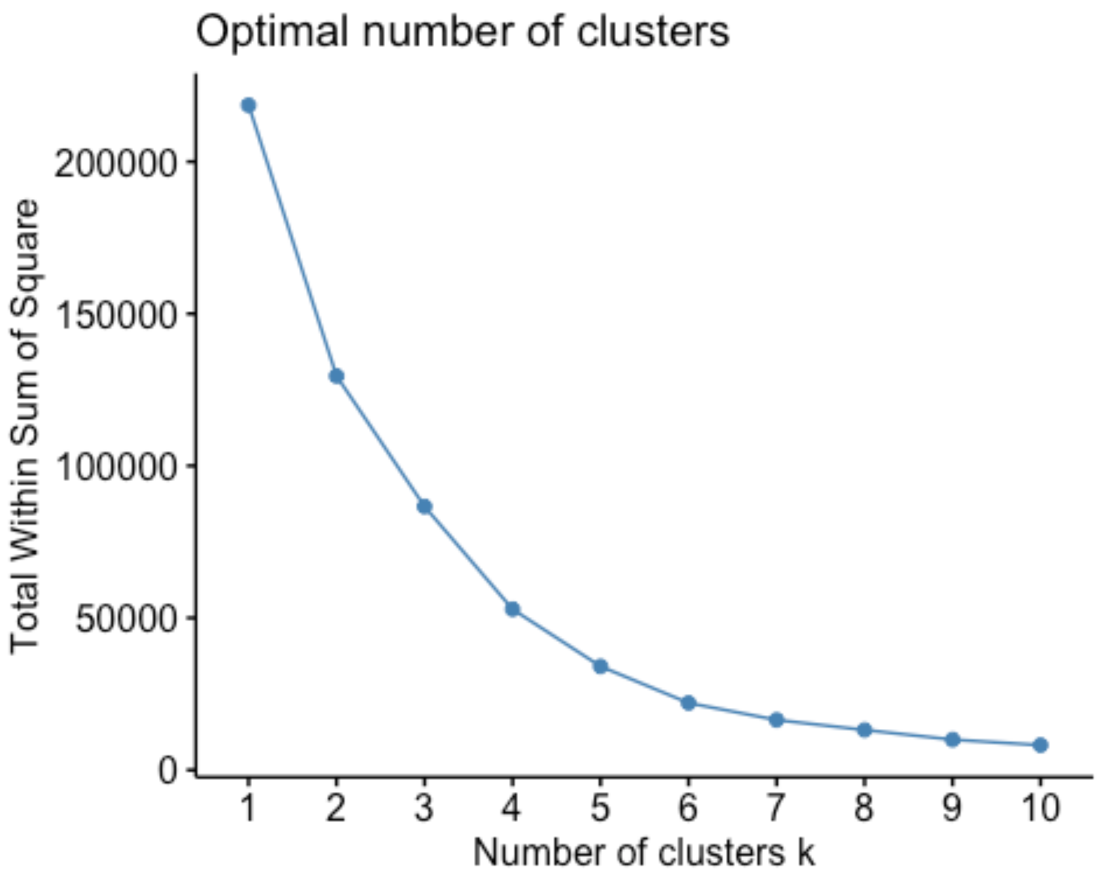
```
gower_dist <- daisy(starwars_cleaned, metric = "gower")
```

Perform hierarchical clustering using average linkage

#hac <- agnes(gower_dist, method = “average”)

```
fviz_nbclust(starwars_cleaned, FUN = hcut , method = "wss")

## Warning in stats::dist(x): NAs introduced by coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion
```

```
## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion

## Warning in stats::dist(x, method = method, ...): NAs introduced by
coercion
```



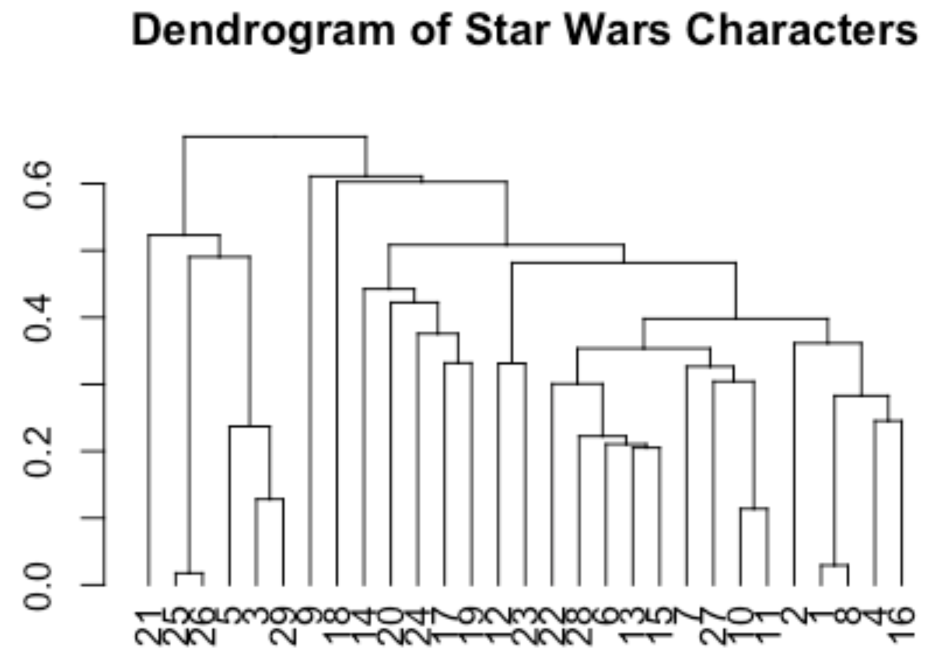
```
hac_1 <- hclust(gower_dist, method='average')
hac_1_average <- cutree(hac_1, k=6)
```

B)Determine the best number of clusters by plotting the dendrogram

```
dendrogram <- as.dendrogram(hac_1)
```

Plot dendrogram

```
plot(dendrogram, main = "Dendrogram of Star Wars Characters")
```



Anomalies would appear as branches that link to the rest of the tree at higher distances (y-axis indicates the distance or dissimilarity).

These are likely to be characters whose attributes are significantly different from the rest of the group, causing them to be joined at a later stage in the clustering process.

There appear to be a few branches that join at a higher level, particularly the single branch extending near the top right of the dendrogram.

This could indicate a potential anomaly - a character that is quite distinct from all others in the dataset.

Advantages of Considering Anomalies in a Dendrogram: Dendrograms provide a visual representation of data relationships, making it easier to identify clusters and anomalies.

They can reveal hierarchical structures within the data, which can help in understanding how anomalies are related to other data points.

Disadvantages of Considering Anomalies in a Dendrogram: Dendrograms can become complex and difficult to interpret, especially in large datasets or when the hierarchy is deep.

Anomalies may not always be apparent in a dendrogram, especially if they are close to the root or buried within the hierarchy.

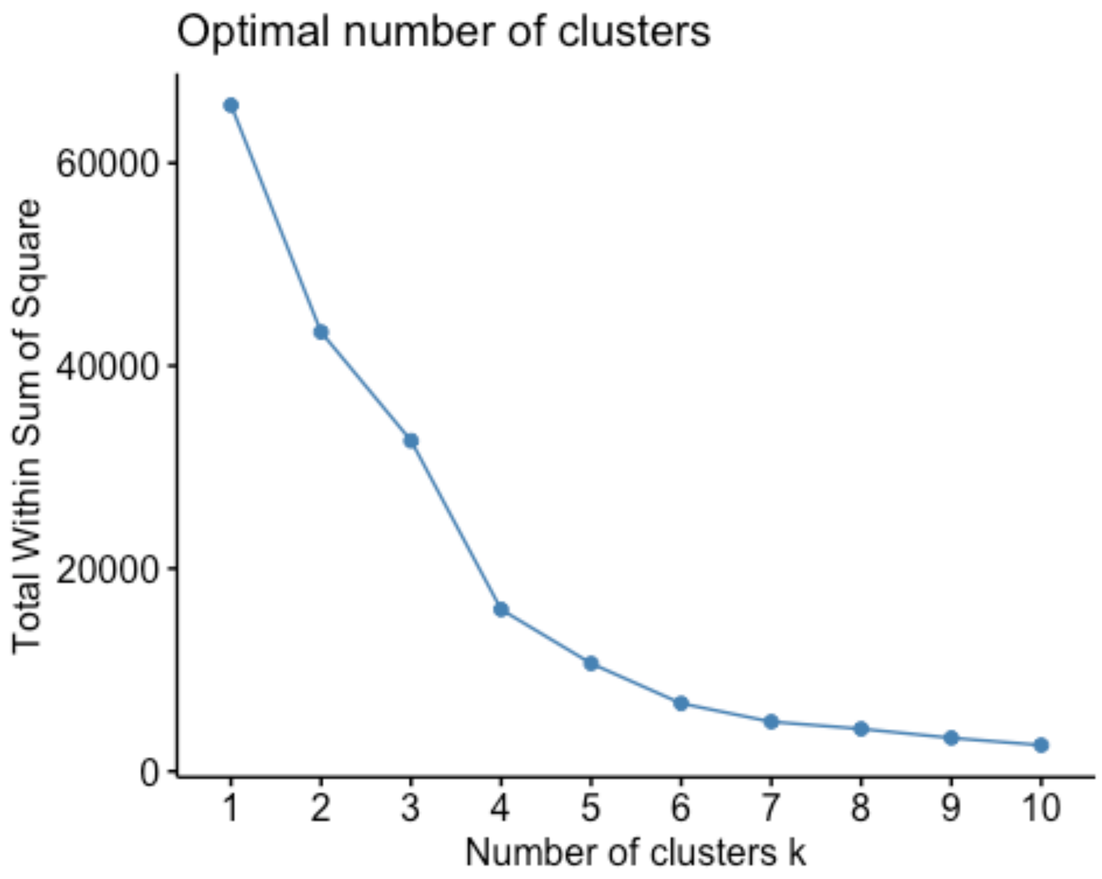
C)Remove columns that are lists or where dummy variables are not applicable

```
starwars_data_dummies <- starwars_cleaned %>%
  model.matrix(~ . - 1, data = .) %>% # Create dummy variables
(excluding intercept)
  as.data.frame() # Convert to a data frame
```

Perform k-means clustering on the dummy variable dataset

We’ll use the ‘fviz_nbclust’ function to find the optimal number of clusters based on several methods

```
fviz_nbclust(starwars_data_dummies, kmeans, method = "wss")  # Elbow
method
```



Let’s assume the best number of clusters chosen is ‘k’

set.seed(123) for reproducibility k <- 3 replace with the number found to be optimal

```
kmeans_result <- kmeans(starwars_data_dummies, centers = 6, nstart =
20)
```

Review the structure of the k-means result

```
str(kmeans_result)

## List of 9
##  $ cluster     : Named int [1:29] 6 4 5 4 6 6 6 6 2 6 ...
##   ..- attr(*, "names")= chr [1:29] "1" "2" "3" "4" ...
##  $ centers     : num [1:6, 1:146] 186 228 88 190 166 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ : chr [1:6] "1" "2" "3" "4" ...
##   .. ..$ : chr [1:146] "height" "mass" "hair_colorauburn"
"hair_colorauburn, grey" ...
##  $ totss       : num 65696
##  $ withinss    : num [1:6] 1375 0 0 649 1356 ...
##  $ tot.withinss: num 6698
##  $ betweenss   : num 58999
##  $ size        : int [1:6] 5 1 1 3 5 14
##  $ iter        : int 3
##  $ ifault      : int 0
##  - attr(*, "class")= chr "kmeans"
```

D)

Create a dataframe

```
results <- data.frame(Species = starwars_cleaned$species, HAC =
hac_1_average, Kmeans = kmeans_result$cluster)

starwars_cleaned$species

##  [1] Human        Human        Human        Human        Human
##  [6] Human        Human        Human        Wookiee      Human
## [11] Human        Human        Human        Trandoshan   Human
## [16] Human        Mon Calamari Ewok         Gungan       Zabrak
## [21] Twi'lek      Human        Cerean       Kel Dor      Mirialan
## [26] Mirialan     Human        Human        Human
## 37 Levels: Aleena Besalisk Cerean Chagrian Clawdite Droid Dug ...
Zabrak

levels(starwars_cleaned$species)

##  [1] "Aleena"       "Besalisk"     "Cerean"       "Chagrian"
##  [5] "Clawdite"     "Droid"        "Dug"          "Ewok"
##  [9] "Geonosian"    "Gungan"       "Human"        "Hutt"
## [13] "Iktotchi"     "Kaleesh"      "Kaminoan"     "Kel Dor"
## [17] "Mirialan"     "Mon Calamari" "Muun"         "Nautolan"
## [21] "Neimodian"    "Pau'an"       "Quermian"     "Rodian"
## [25] "Skakoan"      "Sullustan"    "Tholothian"   "Togruta"
## [29] "Toong"        "Toydarian"    "Trandoshan"   "Twi'lek"
## [33] "Vulptereen"   "Wookiee"      "Xexto"        "Yoda's"
```

```
species"
## [37] "Zabrak"
```

Cross-tabulation for HAC

```
results %>% group_by(HAC) %>% select(HAC, Species) %>% table()

##    Species
## HAC Aleena Besalisk Cerean Chagrian Clawdite Droid Dug Ewok
Geonosian Gungan
##   1      0        0      1        0        0     0   0    0
0      0
##   2      0        0      0        0        0     0   0    0
0      0
##   3      0        0      0        0        0     0   0    0
0      0
##   4      0        0      0        0        0     0   0    0
0      1
##   5      0        0      0        0        0     0   0    1
0      0
##   6      0        0      0        0        0     0   0    0
0      0
##    Species
## HAC Human Hutt Iktotchi Kaleesh Kaminoan Kel Dor Mirialan Mon
Calamari Muun
##   1    15    0        0       0        0       0        0
0    0
##   2     3    0        0       0        0       0        2
0    0
##   3     0    0        0       0        0       0        0
0    0
##   4     0    0        0       0        0       1        0
1    0
##   5     0    0        0       0        0       0        0
0    0
##   6     0    0        0       0        0       0        0
0    0
##    Species
## HAC Nautolan Neimodian Pau'an Quermian Rodian Skakoan Sullustan
Tholothian
##   1        0         0      0        0      0       0         0
```

```
0
##   2         0          0       0         0       0       0         0
0
##   3         0          0       0         0       0       0         0
0
##   4         0          0       0         0       0       0         0
0
##   5         0          0       0         0       0       0         0
0
##   6         0          0       0         0       0       0         0
0
##    Species
## HAC Togruta Toong Toydarian Trandoshan Twi'lek Vulptereen Wookiee
Xexto
##   1       0     0         0          0       0          0       0
0
##   2       0     0         0          0       0          0       0
0
##   3       0     0         0          0       0          0       1
0
##   4       0     0         0          1       0          0       0
0
##   5       0     0         0          0       0          0       0
0
##   6       0     0         0          0       1          0       0
0
##    Species
## HAC Yoda's species Zabrak
##   1              0      0
##   2              0      0
##   3              0      0
##   4              0      1
##   5              0      0
##   6              0      0
```

Crosstab for K Means

```
results %>% group_by(Kmeans) %>% select(Kmeans, Species) %>% table()
```

```
##       Species
## Kmeans Aleena Besalisk Cerean Chagrian Clawdite Droid Dug Ewok
```

```
Geonosian Gungan
##      1       0         0       1         0         0     0    0     0
0      0
##      2       0         0       0         0         0     0    0     0
0      0
##      3       0         0       0         0         0     0    0     1
0      0
##      4       0         0       0         0         0     0    0     0
0      0
##      5       0         0       0         0         0     0    0     0
0      0
##      6       0         0       0         0         0     0    0     0
0      1
##       Species
## Kmeans Human Hutt Iktotchi Kaleesh Kaminoan Kel Dor Mirialan Mon
Calamari Muun
##      1     4    0        0       0        0       0        0
0    0
##      2     0    0        0       0        0       0        0
0    0
##      3     0    0        0       0        0       0        0
0    0
##      4     2    0        0       0        0       0        0
0    0
##      5     2    0        0       0        0       0        2
0    0
##      6    10    0        0       0        0       1        0
1    0
##       Species
## Kmeans Nautolan Neimodian Pau'an Quermian Rodian Skakoan Sullustan
Tholothian
##      1        0         0      0        0      0       0         0
0
##      2        0         0      0        0      0       0         0
0
##      3        0         0      0        0      0       0         0
0
##      4        0         0      0        0      0       0         0
0
##      5        0         0      0        0      0       0         0
0
```

```
##      6         0          0       0         0       0        0          0
0
##        Species
## Kmeans Togruta Toong Toydarian Trandoshan Twi'lek Vulptereen
Wookiee Xexto
##      1       0     0         0          0       0          0
0      0
##      2       0     0         0          0       0          0
1      0
##      3       0     0         0          0       0          0
0      0
##      4       0     0         0          1       0          0
0      0
##      5       0     0         0          0       1          0
0      0
##      6       0     0         0          0       0          0
0      0
##        Species
## Kmeans Yoda's species Zabrak
##      1              0      0
##      2              0      0
##      3              0      0
##      4              0      0
##      5              0      0
##      6              0      1
```