Assignment_4

Barini Simhadri

31-10-2023

Problem 1
Problem 2
Problem 3
Problem 4

Importing Libraries

```
library(dplyr)
library(caret)
library(ggplot2)
library(factoextra)
library(tidyverse)
library(e1071)
library(rpart)
library(kknn)
library(cluster)
```

Problem 1

For this problem, you will tune and apply kNN and compare it to other classifiers. We will use the wine quality data, which has a number of measurements about chemical components in wine, plus a quality rating. There are separate files for red and white wines, so the first step is some data preparation.

a. Load the two provided wine quality datasets and prepare them by (1) ensuring that all the variables have the right type (e.g., what is numeric vs. factor), (2) adding a type column to each that indicates if it is red or white wine and (2) merging the two tables together into one table (hint: try full_join()). You now have one table that contains the data on red and white wine, with a column that tells if the wine was from the red or white set (the type column you made).

```
white =
read.csv("C:/Users/bunty/Desktop/funda/week_7/winequality-
white.csv"
,header = T,sep = ";") red =
read.csv("C:/Users/bunty/Desktop/funda/week_7/winequality-
red.csv",header = T,sep = ";")
```

There are not NA's in both the data set.

```
white[rowSums(is.na(white)) > 0, ]
```

```
## [1] fixed.acidity volatile.acidity citric.acid
```

```
##  [4] residual.sugar       chlorides           free.sulfur.dioxide
##  [7] total.sulfur.dioxide density             pH
## [10] sulphates            alcohol             quality
## <0 rows> (or 0-length row.names)
```

```
red[rowSums(is.na(red)) > 0, ]
```

```
##  [1] fixed.acidity        volatile.acidity     citric.acid
##  [4] residual.sugar       chlorides            free.sulfur.dioxide
##  [7] total.sulfur.dioxide density              pH
## [10] sulphates            alcohol              quality
## <0 rows> (or 0-length row.names)
```

Here we can see, all the variables has right type.

```
str(white)
```

```
## 'data.frame':    4898 obs. of  12 variables:
##  $ fixed.acidity       : num  7 6.3 8.1 7.2 7.2 8.1 6.2 7 6.3 8.1 ...
##  $ volatile.acidity    : num  0.27 0.3 0.28 0.23 0.23 0.28 0.32 0.27 0.3 0.22 ...
##  $ citric.acid         : num  0.36 0.34 0.4 0.32 0.32 0.4 0.16 0.36 0.34 0.43 ...
##  $ residual.sugar      : num  20.7 1.6 6.9 8.5 8.5 6.9 7 20.7 1.6 1.5 ...
##  $ chlorides           : num  0.045 0.049 0.05 0.058 0.058 0.05 0.045 0.045 0.049 0.044 ...
##  $ free.sulfur.dioxide : num  45 14 30 47 47 30 30 45 14 28 ...
##  $  total.sulfur.dioxide: num  170 132 97 186 186 97 136 170 132 129 ...
##  $ density             : num  1.001 0.994 0.995 0.996 0.996 ...
##  $ pH                  : num  3 3.3 3.26 3.19 3.19 3.26 3.18 3 3.3 3.22 ...
##  $ sulphates           : num  0.45 0.49 0.44 0.4 0.4 0.44 0.47 0.45 0.49 0.45 ...
##  $ alcohol             : num  8.8 9.5 10.1 9.9 9.9 10.1 9.6 8.8 9.5 11 ...
##  $ quality             : int  6 6 6 6 6 6 6 6 6 6 ...
```

```
str(red)
```

```
## 'data.frame':    1599 obs. of  12 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 ...
```

Adding type column to both the data set.

```
red$type <- "red"
white$type <- "white"
```

```
wines <- full_join(red, white)
head(wines)
```

```
##   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
```

```
str(wines)
```

```
## '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" ...
```

b. Use PCA to create a projection of the data to 2D and show a scatterplot with color showing the wine type.

```
# Create dummy variables
dummy <- dummyVars(type ~ ., data = wines)
dummies <- as.data.frame(predict(dummy, newdata = wines))
head(dummies)
```

```
##   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
```

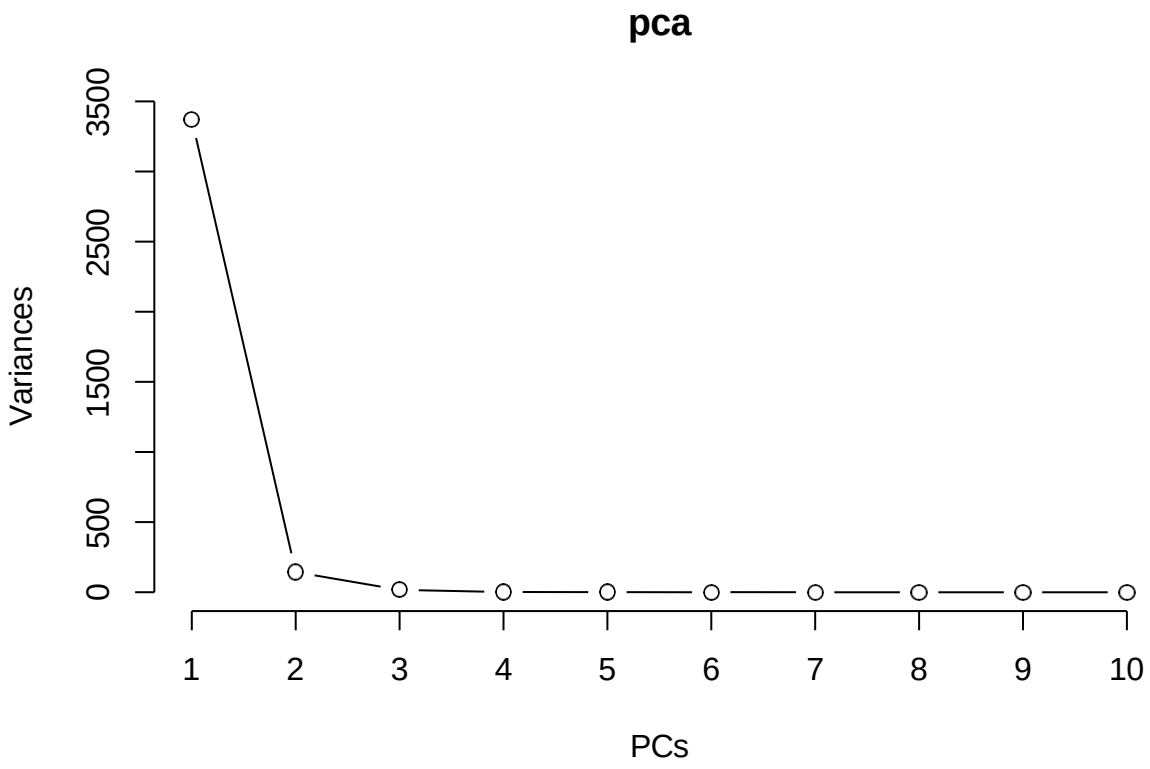
```
## 6            7.4             0.66        0.00            1.8      0.075
##   free.sulfur.dioxide total.sulfur.dioxide density   pH sulphates alcohol
## 1                  11                  34  0.9978 3.51      0.56     9.4
## 2                  25                  67  0.9968 3.20      0.68     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
```

```
set.seed(123)
pca = prcomp(dummies)
```

```
summary(pca)
```

```
## Importance of components:
##                             PC1      PC2      PC3     PC4     PC5     PC6     PC7
## Standard deviation     58.0698 11.98563 4.13097 1.32231 1.10726 0.69455 0.17520
## Proportion of Variance  0.9536  0.04062 0.00483 0.00049 0.00035 0.00014 0.00001
## Cumulative Proportion   0.9536  0.99418 0.99900 0.99950 0.99984 0.99998 0.99999
##                             PC8     PC9    PC10    PC11      PC12
## Standard deviation      0.14193 0.12090 0.10190 0.02786 0.0007505
## Proportion of Variance  0.00001 0.00000 0.00000 0.00000 0.0000000
## Cumulative Proportion   0.99999 1.00000 1.00000 1.00000 1.0000000
```

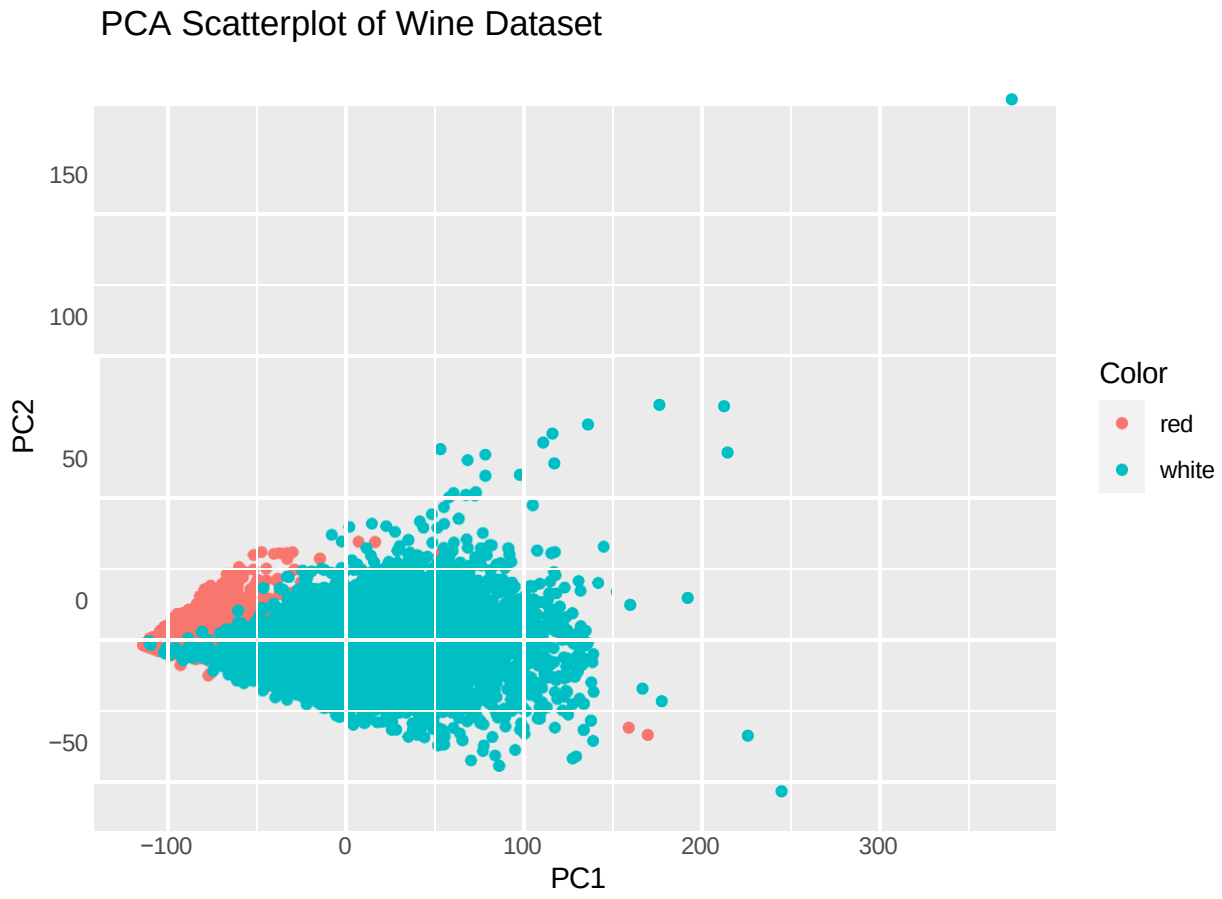
```
screeplot(pca, type = "l") + title(xlab = "PCs")
```




```
## integer(0)
```

```
rotated_data = as.data.frame(pca$x)
rotated_data$Color <- wines$type
```

```
ggplot(data = rotated_data, aes(x = PC1, y = PC2, color = Color)) + geom_point()+
  labs(title = "PCA Scatterplot of Wine Dataset")
```



c. We are going to try kNN, SVM and decision trees on this data. Based on the 'shape' of the data in the visualization from (b), which do you think will do best and why?

Based on the visualization from (b), I think KNN will do best as it can easily find its neighbor and the data above seems to be bifurcated in red and white wine respectively. Let's apply all the three models and observe which gives the higher accuracy.

```
wines$type = as.factor(wines$type)
str(wines$type)
```

```
##  Factor w/ 2 levels "red","white": 1 1 1 1 1 1 1 1 1 1 ...
```

Using KNN :

```
train_control <- trainControl(method="cv", number = 10)
knnFit <- train(type ~ ., data = wines, method = "knn", trControl = train_control,
                preProcess = c("center","scale"))
knnFit
```

```
## k-Nearest Neighbors
##
## 6497 samples
##    12 predictor
##     2 classes: 'red', 'white'
##
## Pre-processing: centered (12), scaled (12)
```

```
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5847, 5847, 5848, 5847, 5847, 5847, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.9924589  0.9796351
##   7  0.9932289  0.9817216
##   9  0.9929205  0.9809196
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

Using SVM:

```
svmFit <- train(type ~., data = wines, method = "svmLinear",trControl = train_control)
svmFit
```

```
## Support Vector Machines with Linear Kernel
##
## 6497 samples
##   12 predictor
##    2 classes: 'red', 'white'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5848, 5847, 5847, 5848, 5847, 5848, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9950743  0.9867007
##
## Tuning parameter 'C' was held constant at a value of 1
```

Using Decision Tree

```
treeFit <- train(type ~., data = wines, method = "rpart", trControl = train_control)
treeFit
```

```
## CART
##
## 6497 samples
##   12 predictor
##    2 classes: 'red', 'white'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5847, 5848, 5847, 5847, 5848, 5847, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.06253909  0.9505924  0.8615560
##   0.06754221  0.9315107  0.8070147
##   0.70043777  0.8196051  0.3073034
```

```
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.06253909.
```

d. Use kNN (tune k), use decision trees (basic rpart method is fine), and SVM (tune C) to predict type from the rest of the variables. Compare the accuracy values – is this what you expected? Can you explain it?

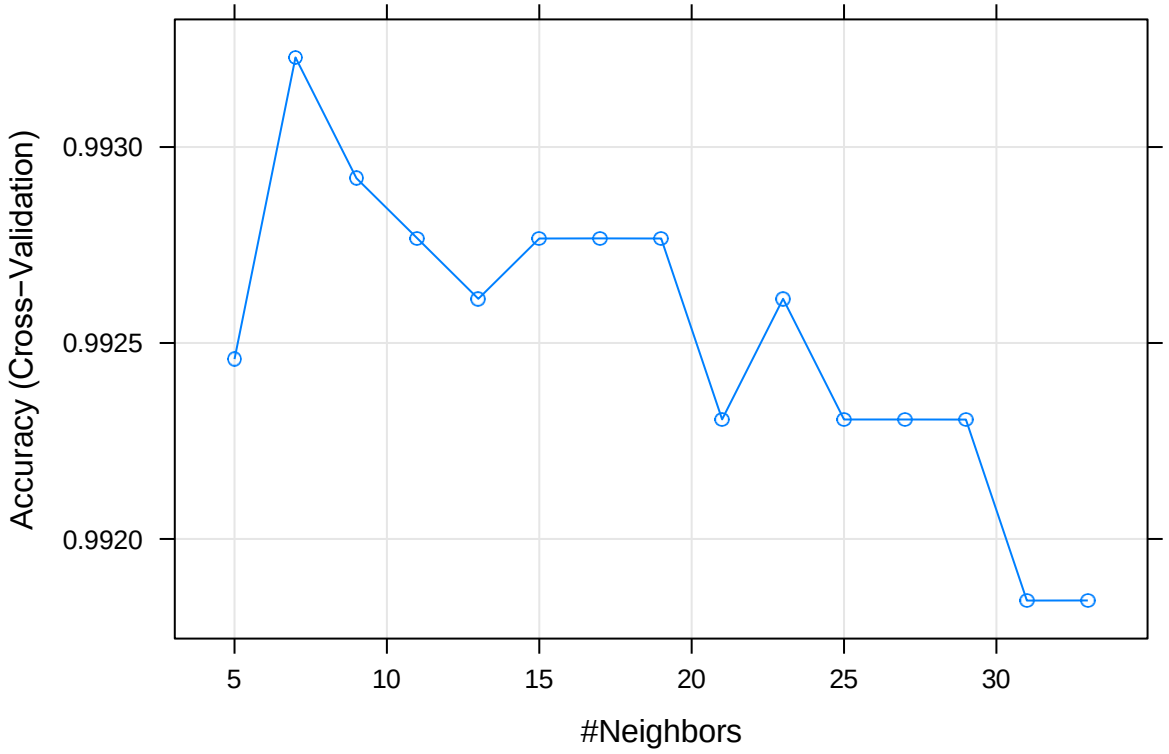
```
colnames(wines) <- make.names(colnames(wines))
```

tuning KNN:

```
set.seed(123)
knnTuneFit <- train(type ~ ., data = wines,method = "knn",trControl = train_control,
                    preProcess = c("center","scale"),tuneLength = 15)
knnTuneFit
```

```
## k-Nearest Neighbors
##
## 6497 samples
##    12 predictor
##     2 classes: 'red', 'white'
##
## Pre-processing: centered (12), scaled (12)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5847, 5847, 5848, 5847, 5847, 5847, ...
## Resampling results across tuning parameters:
##
##    k   Accuracy   Kappa
##    5   0.9924589  0.9796299
##    7   0.9932286  0.9817231
##    9   0.9929205  0.9809196
##   11   0.9927669  0.9805185
##   13   0.9926128  0.9801045
##   15   0.9927666  0.9805096
##   17   0.9927669  0.9805306
##   19   0.9927666  0.9805149
##   21   0.9923051  0.9792825
##   23   0.9926128  0.9801080
##   25   0.9923048  0.9792909
##   27   0.9923048  0.9792805
##   29   0.9923046  0.9792785
##   31   0.9918428  0.9780443
##   33   0.9918431  0.9780359
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 7.
```

```
#Show a plot of accuracy vs k
plot(knnTuneFit)
```



Decision Tree

```
tree <- train(type ~., data = wines, method = "rpart", trControl =train_control)
tree
```

```
## CART
##
## 6497 samples
##    12 predictor
##     2 classes: 'red', 'white'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5847, 5847, 5848, 5849, 5847, 5847, ...
## Resampling results across tuning parameters:
##
##    cp           Accuracy   Kappa
##    0.06253909   0.9432061  0.8379748
##    0.06754221   0.9313599  0.8050301
##    0.70043777   0.8345020  0.3791987
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.06253909.
```

Tuning SVM

```
grid <- expand.grid(C = 10^seq(-5,2,0.5))
svm_grid <- train(type ~., data = wines, method = "svmLinear",
                  trControl = train_control, tuneGrid = grid)
svm_grid
```

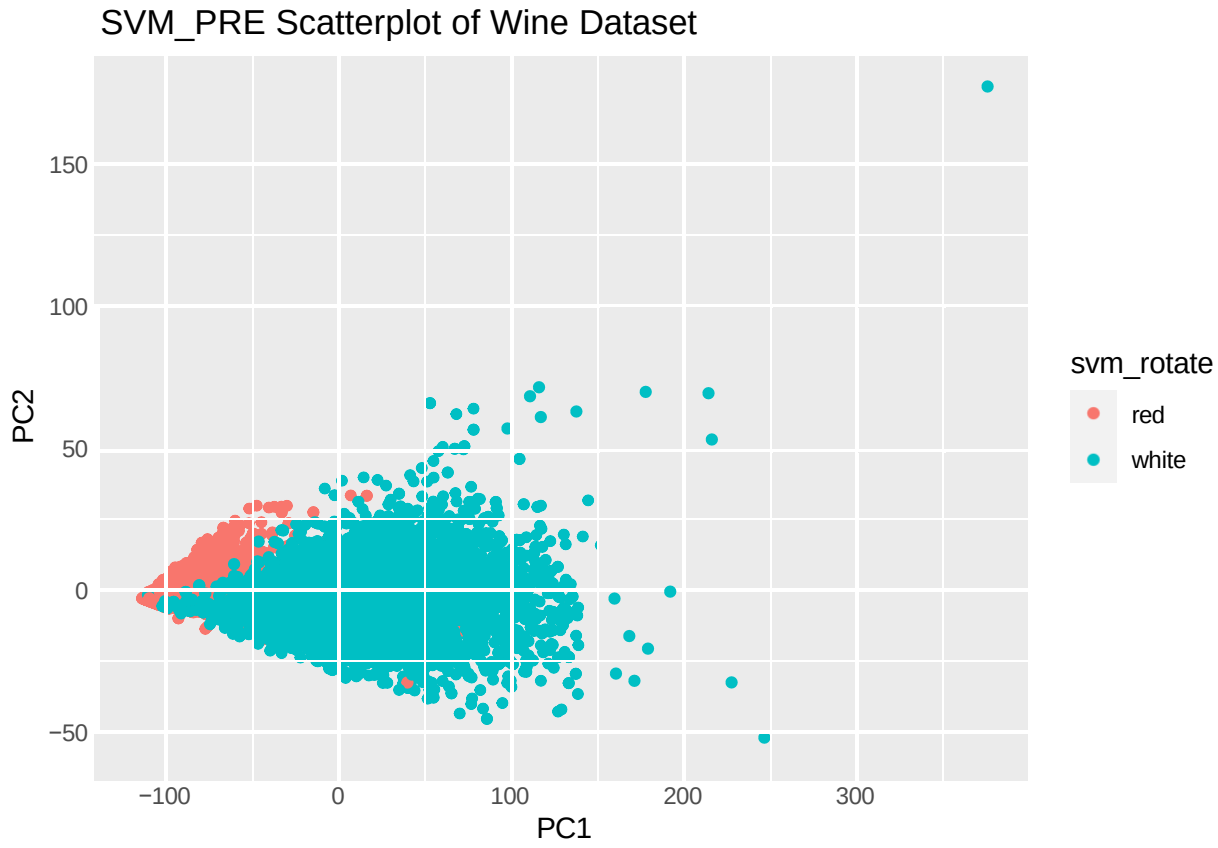
```
## Support Vector Machines with Linear Kernel
##
## 6497 samples
##    12 predictor
##     2 classes: 'red', 'white'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5847, 5847, 5847, 5847, 5847, 5848, ...
## Resampling results across tuning parameters:
##
##    C             Accuracy   Kappa
##    1.000000e-05  0.7538865  0.000000000
##    3.162278e-05  0.7541944  0.001878186
##    1.000000e-04  0.9299663  0.791348391
##    3.162278e-04  0.9836852  0.955439014
##    1.000000e-03  0.9904558  0.974205137
##    3.162278e-03  0.9916871  0.977595825
##    1.000000e-02  0.9926111  0.980094427
##    3.162278e-02  0.9935349  0.982566763
##    1.000000e-01  0.9944580  0.985055742
##    3.162278e-01  0.9950736  0.986705288
##    1.000000e+00  0.9950738  0.986707253
##    3.162278e+00  0.9950738  0.986707253
##    1.000000e+01  0.9953815  0.987532826
##    3.162278e+01  0.9955354  0.987946497
##    1.000000e+02  0.9955354  0.987946497
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 31.62278.
```

From the above, the accuracy are - KNN(k=7) : 0.9932286, decision tree(cp=0.06253909) :0.9313599, SVM(C=31.62278):0.9950738. SVM has the highest accuracy followed by KNN and decision tree. These results are nearly expected as we know KNN and SVM are good classification algorithm and gives higher accuracy. However, the outcome seems to inclined with our expectation , still we should consider the behavior of dataset and other factors before deciding the appropriate algorithm and do some try out methods in order to incline with the specific algorithm.

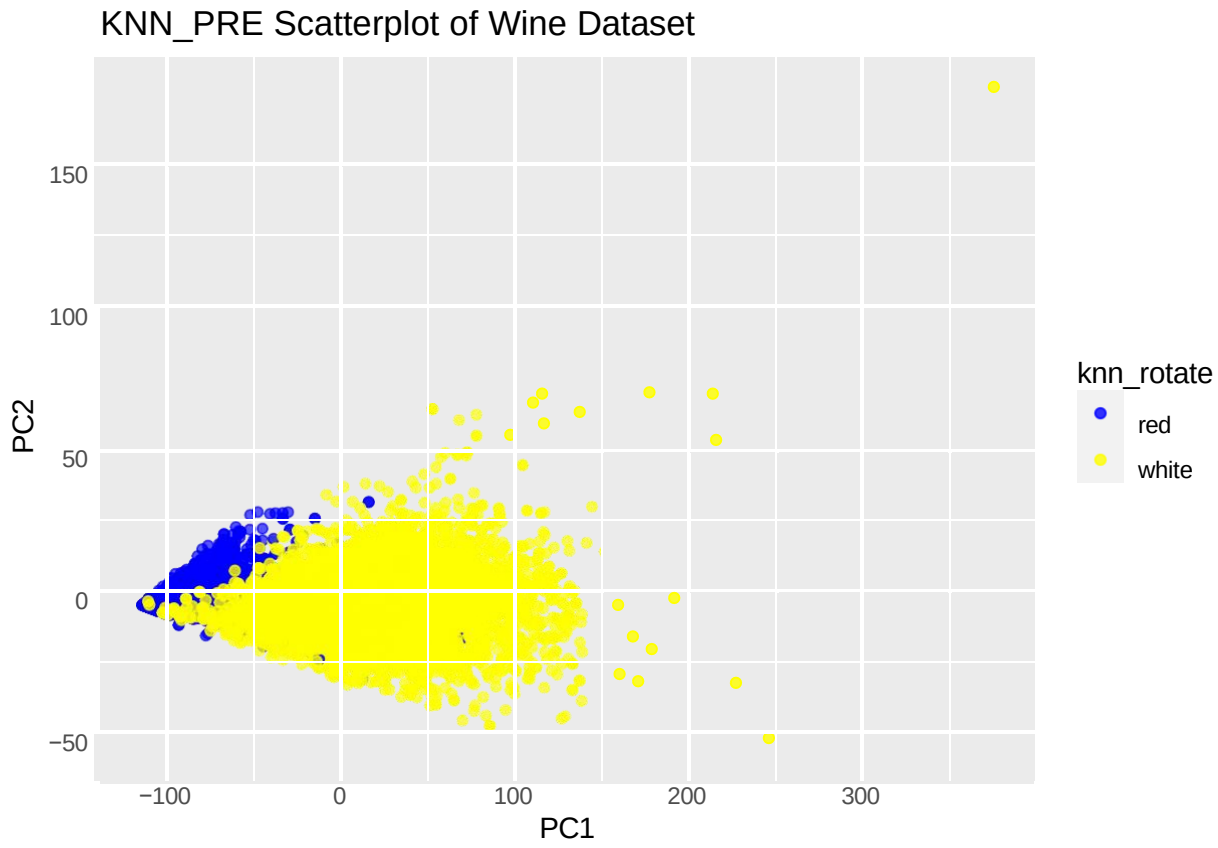
e. Use the same already computed PCA again to show a scatter plot of the data and to visualize the labels for kNN, decision tree and SVM. Note that you do not need to recreate the PCA projection, you have already done this in 1b. Here, you just make a new visualization for each classifier using its labels for color (same points but change the color). Map the color results to the classifier, that is use the "predict" function to predict the class of your data, add it to your data frame and use it as a color. This is done for KNN in the tutorial, it should be similar for the others. Consider and explain the differences in how these classifiers performed.

```
svm_pre = predict(svm_grid,wines)
rotated_data$svm_rotate = svm_pre
```

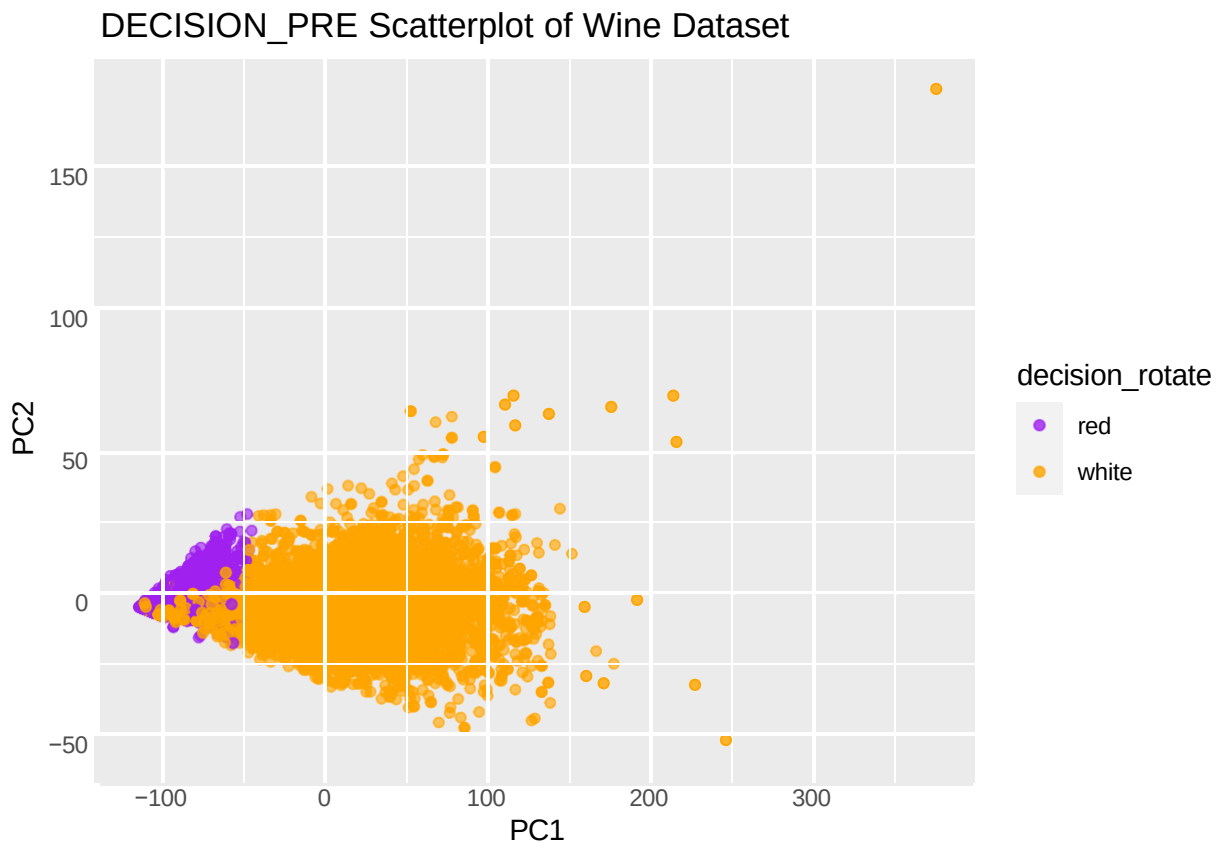
```
ggplot(data = rotated_data, aes(x = PC1, y = PC2, color = svm_rotate)) + geom_point()+
  labs(title = "SVM_PRE Scatterplot of Wine Dataset")
```




```
knn_pre = predict(knnTuneFit,wines)
rotated_data$knn_rotate = knn_pre
ggplot(data = rotated_data, aes(x = PC1, y = PC2, color = knn_rotate)) + geom_point(alpha = 0.8)+
  labs(title = "KNN_PRE Scatterplot of Wine Dataset") + scale_color_manual(values=c('Blue','Yellow'))
```




```
decision_pre = predict(tree,wines)
rotated_data$decision_rotate = decision_pre
ggplot(data = rotated_data, aes(x = PC1, y = PC2, color = decision_rotate)) + geom_point(alpha = 0.8)+
  labs(title = "DECISION_PRE Scatterplot of Wine Dataset") + scale_color_manual(values=c('purple','oran
```



Problem 2

In this question we will use the Sacramento data, which covers available housing in the region of that city. The variables include numerical information about the size of the housing and its price, as well as categorical information like zip code (there are a large but limited number in the area), and the type of unit (condo vs house (coded as residential)).

a. Load the data from the tidyverse library with the data("Sacramento") command and you should have a variable Sacramento. Because we have categoricals, convert them to dummy variables.

```
data("Sacramento")
df_sac = select(Sacramento, -c("latitude", "longitude", "zip"))
head(df_sac)
```

```
##           city beds baths sqft        type price
## 1 SACRAMENTO 2    1   836 Residential 59222
## 2 SACRAMENTO 3    1  1167 Residential 68212
## 3 SACRAMENTO 2    1   796 Residential 68880
## 4 SACRAMENTO 2    1   852 Residential 69307
## 5 SACRAMENTO 2    1   797 Residential 81900
## 6 SACRAMENTO 3    1  1122       Condo 89921
```

creating dummies.

```
dummy_sac <- dummyVars(type ~ ., data = df_sac)
dummies_sac <- as.data.frame(predict(dummy_sac, newdata = df_sac))
```

b. With kNN, because of the high dimensionality, which might be a good choice for the distance function? when working with high dimensionality data set, it always become a challenge to pick up a good choice for the distance. We have to try out different functions and check for the metrics we get and select the appropriate one. Apart from this, Minkowski distance, a general distance function as for which (h=1)-Manhattan distance and (h=2)-Euclidean distance is chosen. So, depending on the data, Minkowski can give flexibility by setting its parameter h. Therefore, using Minkowski is fairly common in high dimensional data.

c. Use kNN to classify this data with type as the label. Tune the choice of k plus the type of distance function. Report your results – what values for these parameters were tried, which were chosen, and how did they perform with accuracy?

```
sacramento_dummies <- dummies_sac
sacramento_dummies$type <- Sacramento$type
head(sacramento_dummies)
```

```
##   city.ANTELOPE city.AUBURN city.CAMERON_PARK city.CARMICHAEL
## 1             0           0                 0               0
## 2             0           0                 0               0
## 3             0           0                 0               0
## 4             0           0                 0               0
## 5             0           0                 0               0
## 6             0           0                 0               0
##   city.CITRUS_HEIGHTS city.COOL city.DIAMOND_SPRINGS city.EL_DORADO
## 1                   0         0                    0              0
## 2                   0         0                    0              0
## 3                   0         0                    0              0
## 4                   0         0                    0              0
## 5                   0         0                    0              0
## 6                   0         0                    0              0
##   city.EL_DORADO_HILLS city.ELK_GROVE city.ELVERTA city.FAIR_OAKS city.FOLSOM
## 1                    0              0            0              0           0
## 2                    0              0            0              0           0
## 3                    0              0            0              0           0
## 4                    0              0            0              0           0
## 5                    0              0            0              0           0
## 6                    0              0            0              0           0
##   city.FORESTHILL city.GALT city.GARDEN_VALLEY city.GOLD_RIVER city.GRANITE_BAY
## 1               0         0                  0               0               0
## 2               0         0                  0               0               0
## 3               0         0                  0               0               0
## 4               0         0                  0               0               0
## 5               0         0                  0               0               0
## 6               0         0                  0               0               0
##   city.GREENWOOD city.LINCOLN city.LOOMIS city.MATHER city.MEADOW_VISTA
## 1              0            0           0           0                 0
## 2              0            0           0           0                 0
## 3              0            0           0           0                 0
## 4              0            0           0           0                 0
```

```
## 5                    0             0             0             0                    0
## 6                    0             0             0             0                    0
##   city.NORTH_HIGHLANDS city.ORANGEVALE city.PENRYN city.PLACERVILLE
## 1                    0               0           0                0
## 2                    0               0           0                0
## 3                    0               0           0                0
## 4                    0               0           0                0
## 5                    0               0           0                0
## 6                    0               0           0                0
##   city.POLLOCK_PINES city.RANCHO_CORDOVA city.RANCHO_MURIETA city.RIO_LINDA
## 1                  0                   0                   0              0
## 2                  0                   0                   0              0
## 3                  0                   0                   0              0
## 4                  0                   0                   0              0
## 5                  0                   0                   0              0
## 6                  0                   0                   0              0
##   city.ROCKLIN city.ROSEVILLE city.SACRAMENTO city.WALNUT_GROVE
## 1            0              0                1                 0
## 2            0              0                1                 0
## 3            0              0                1                 0
## 4            0              0                1                 0
## 5            0              0                1                 0
## 6            0              0                1                 0
##   city.WEST_SACRAMENTO city.WILTON beds baths sqft price        type
## 1                    0           0    2     1  836 59222 Residential
## 2                    0           0    3     1 1167 68212 Residential
## 3                    0           0    2     1  796 68880 Residential
## 4                    0           0    2     1  852 69307 Residential
## 5                    0           0    2     1  797 81900 Residential
## 6                    0           0    3     1 1122 89921       Condo
```

```
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)                # powers of Minkowski 1 to 3


# tune and fit the model with 10-fold cross validation,
# standardization, and our specialized tune grid
kknn_fit <- train(type ~ .,
                  data = sacramento_dummies,
                  method = 'kknn',
                  trControl = train_control,
                  preProcess = c('center', 'scale'),
                  tuneGrid = tuneGrid)

kknn_fit
```

```
## k-Nearest Neighbors
##
## 932 samples
##  41 predictor
```

```
##    3 classes: 'Condo', 'Multi_Family', 'Residential'
##
## Pre-processing: centered (41), scaled (41)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 838, 840, 838, 837, 840, 840, ...
## Resampling results across tuning parameters:
##
##   kmax  kernel       distance  Accuracy   Kappa
##   3     rectangular  1         0.9346429  0.3608857
##   3     rectangular  2         0.9346312  0.3361014
##   3     rectangular  3         0.9357065  0.3529235
##   3     cos          1         0.9356610  0.4144825
##   3     cos          2         0.9367251  0.4159254
##   3     cos          3         0.9324237  0.3965141
##   4     rectangular  1         0.9346429  0.3608857
##   4     rectangular  2         0.9346312  0.3361014
##   4     rectangular  3         0.9357065  0.3529235
##   4     cos          1         0.9399740  0.4296174
##   4     cos          2         0.9410495  0.4369539
##   4     cos          3         0.9410378  0.4271684
##   5     rectangular  1         0.9346429  0.2988499
##   5     rectangular  2         0.9335674  0.2795822
##   5     rectangular  3         0.9357065  0.3008426
##   5     cos          1         0.9388870  0.4103091
##   5     cos          2         0.9388756  0.4014612
##   5     cos          3         0.9388756  0.3967424
##   6     rectangular  1         0.9346429  0.2988499
##   6     rectangular  2         0.9335674  0.2795822
##   6     rectangular  3         0.9357065  0.3008426
##   6     cos          1         0.9388870  0.4103091
##   6     cos          2         0.9388756  0.4014612
##   6     cos          3         0.9388756  0.3819342
##   7     rectangular  1         0.9346429  0.2988499
##   7     rectangular  2         0.9324921  0.2217049
##   7     rectangular  3         0.9335560  0.2777082
##   7     cos          1         0.9388870  0.4103091
##   7     cos          2         0.9399394  0.3964103
##   7     cos          3         0.9388756  0.3657584
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were kmax = 4, distance = 2 and kernel
##  = cos.
```

From the above model, the sacramento data was trained and fit using KNN algorithm. The range of K values tested were 3 to 7 with (rectangular and cosine) kernels and powers of Minkowski distance 1 to 3. After fitting the model, The final values used for the model were kmax = 4, distance = 1 and kernel = cos where the accuracy is 0.9420114 which shows how well the model performed on different sets of values of kernels and finding the best optimal accuracy.

Problem 3

In this problem we will continue with the wine quality data from Problem 1, but this time we will use clustering. Do not forget to remove the type variable before clustering because that would be cheating by using the label to perform clustering.

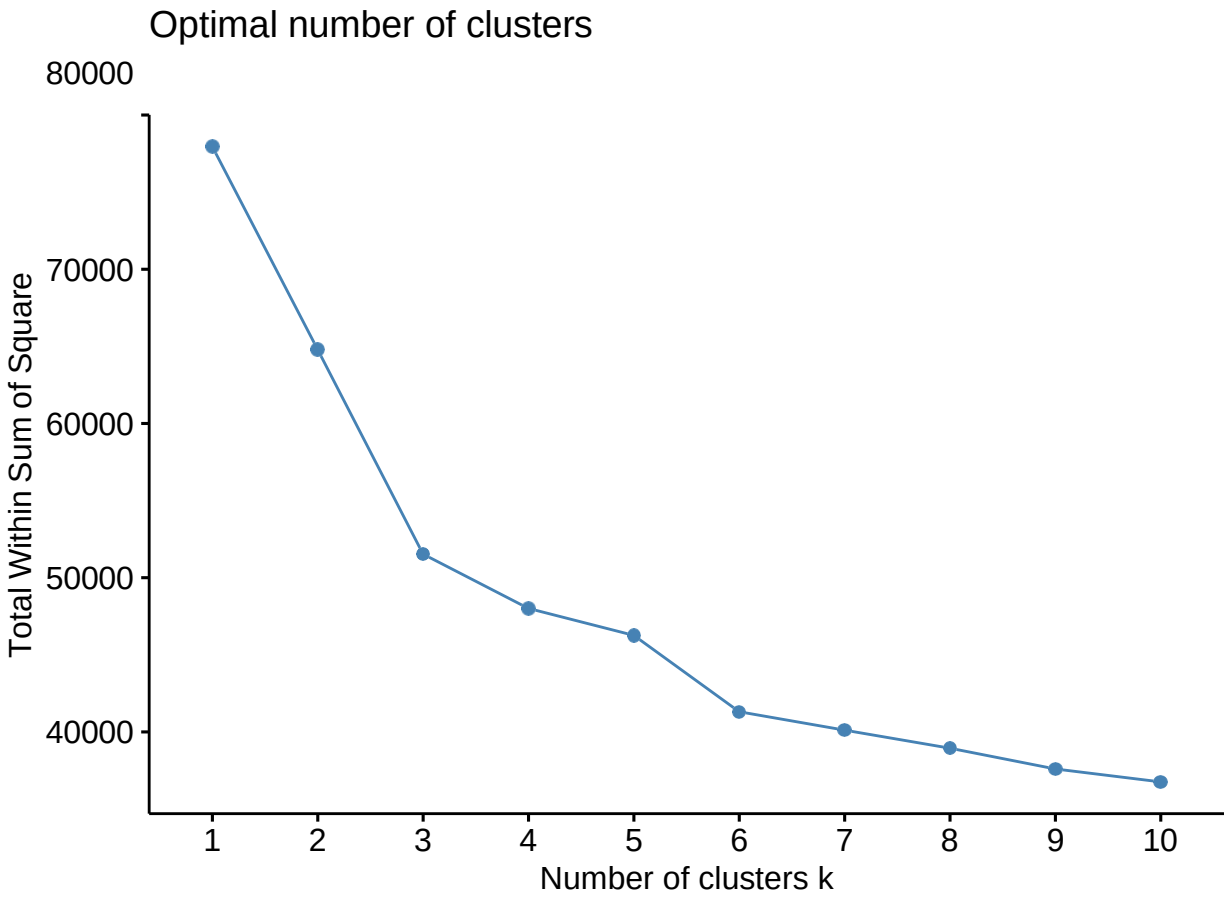
a. Use k-means to cluster the data. Show your usage of silhouette and the elbow method to pick the best number of clusters. Make sure it is using multiple restarts.

```
df_wines = wines
df_wines = select(df_wines,-c("type"))
head(df_wines)
```

```
##   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
```

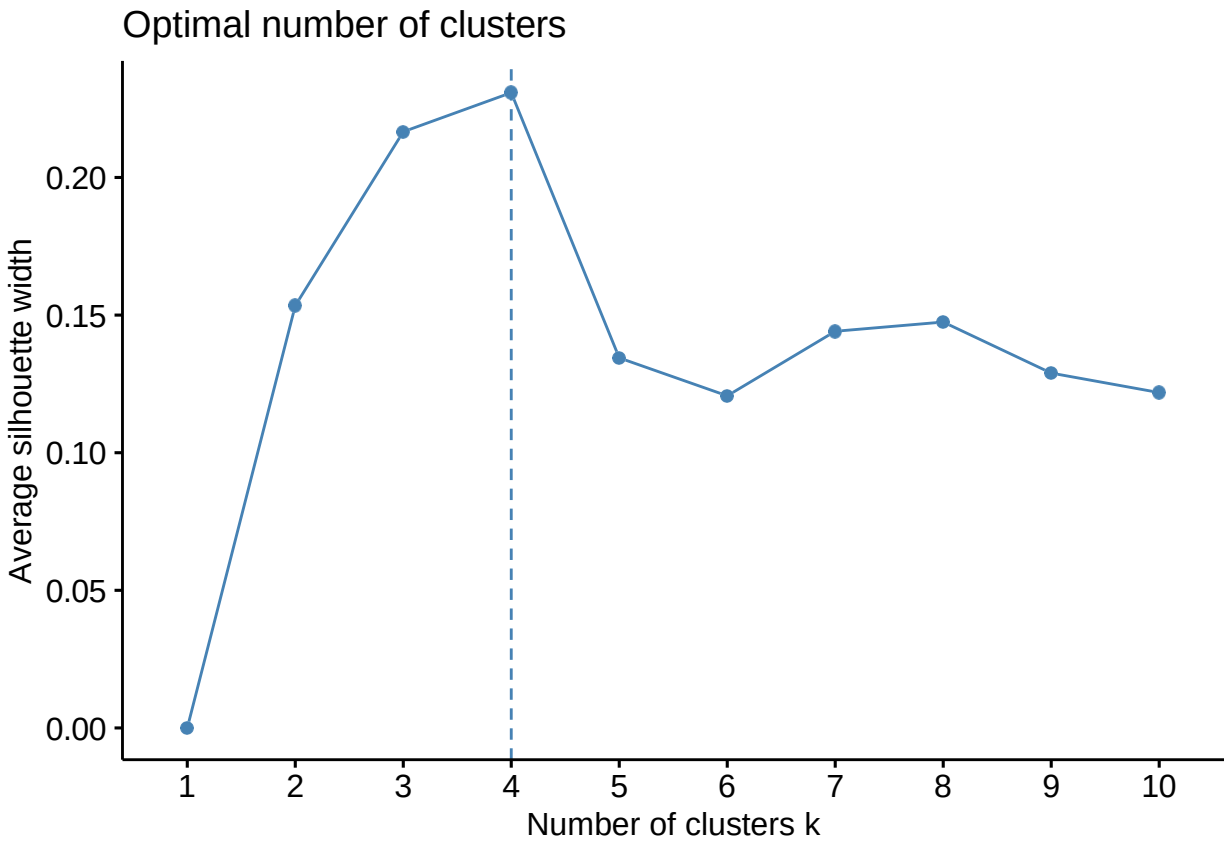
```
set.seed(123)
preproc <- preProcess(df_wines, method=c("center", "scale"))
predictors <- predict(preproc, df_wines)
```

```
fviz_nbclust(predictors, kmeans, method = "wss")
```



elbow method suggest k = 4.

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



Silhouette method suggest k = 4.

```
# Fit the data
fit <- kmeans(predictors, centers = 4, nstart = 25)
fit
```

```
## K-means clustering with 4 clusters of sizes 2835, 661, 1935, 1066
##
## Cluster means:
##   fixed.acidity volatile.acidity citric.acid residual.sugar chlorides
## 1   -0.34563560       -0.4526420  0.04668919     -0.4253161 -0.4572783
## 2    1.95495218        0.4048784  1.02423958     -0.5697395  1.2673641
## 3   -0.19278454       -0.3500900  0.24718059      1.1525186 -0.1081800
## 4    0.05693398        1.5882171 -1.20795561     -0.6076496  0.6266271
##   free.sulfur.dioxide total.sulfur.dioxide    density         pH  sulphates
## 1         -0.06538772           0.03010265 -0.8825064 -0.04277873 -0.2700050
## 2         -0.89472554          -1.24231768  0.9050474 -0.11205327  1.3614349
## 3          0.82522737           0.95020983  0.7315214 -0.38723928 -0.2693766
## 4         -0.76925628          -1.03454506  0.4579506  0.88616595  0.3628512
##       alcohol     quality
## 1  0.64611743  0.40873607
## 2  0.04704102  0.03647183
## 3 -0.80429691 -0.29268305
## 4 -0.28754458 -0.57836111
##
## Clustering vector:
##    [1] 4 4 4 2 4 4 4 4 4 4 4 4 4 2 2 2 2 2 4 2 2 4 2 4 4 4 4 2 4 4 4 4 4 4 4 4 4
```

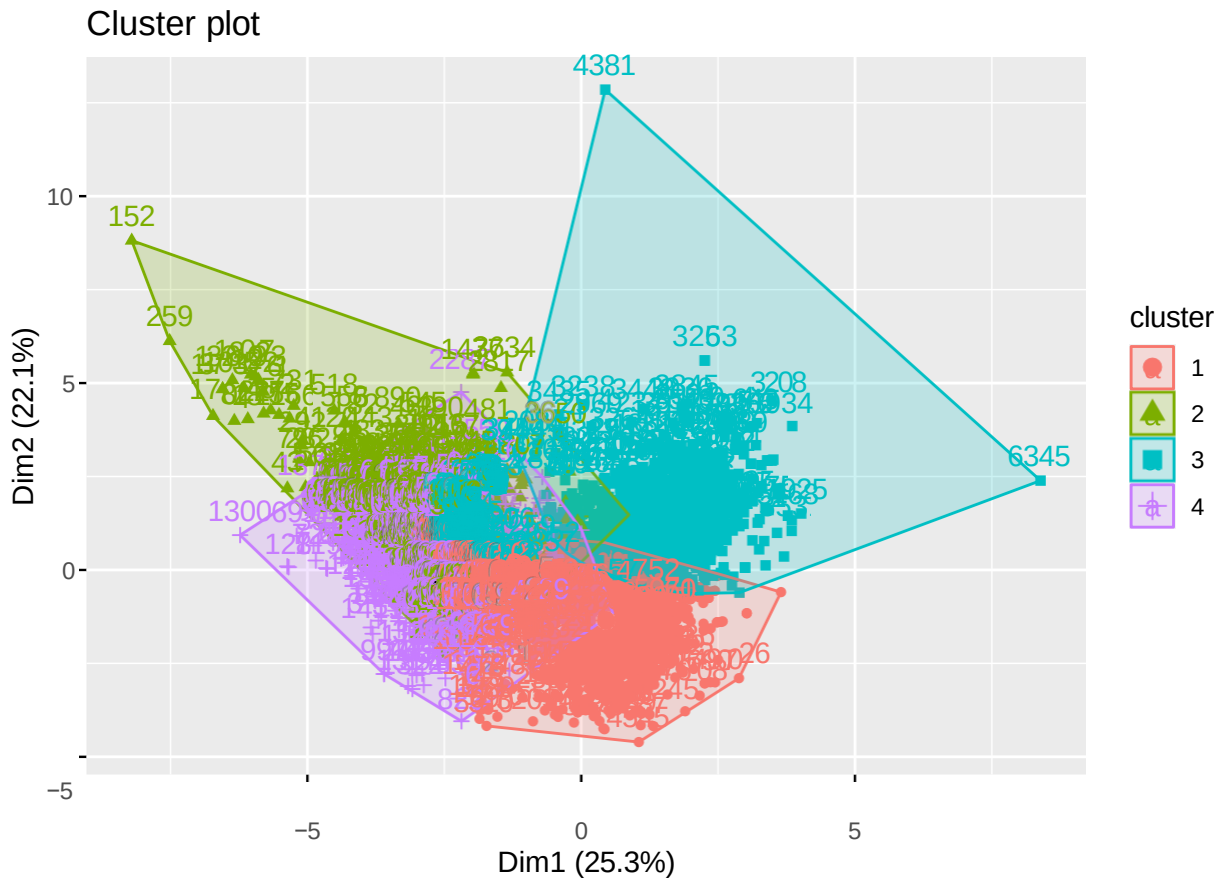
```
##   [38] 2 4 4 4 4 2 4 4 4 4 2 4 4 4 4 4 4 2 4 4 2 4 4 4 4 2 2 4 4 4 4 4 4 4 2 4 4 4 4 4 4
##   [75] 2 2 2 4 4 4 4 4 2 4 2 2 4 2 4 2 4 4 4 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 2 2 4
##  [112] 4 4 2 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 1 4 1 2 4 2
##  [149] 4 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4
##  [186] 2 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 2 2 4 4 4 4 2 2 4 4 4 2 2 4 2 4 4 4 4 4 4 4 4 2 4
##  [223] 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2 4 2 2 4 4 4 4 4 4 2 4 2 4 4 4 2 4 2
##  [260] 2 4 4 4 4 4 2 2 4 2 4 2 4 2 2 4 4 4 4 2 2 2 2 2 4 2 4 4 2 4 4 2 4 4 2 4 2 2 4 2 2 2
##  [297] 2 4 4 4 4 4 2 4 4 2 2 4 2 2 4 2 4 2 4 4 4 4 4 4 2 4 2 4 4 4 2 2 2 2 2 2 2 2 2 2 2 4
##  [334] 4 4 2 2 4 4 2 2 2 2 2 2 2 4 4 2 2 4 2 4 4 4 2 1 4 2 2 2 2 4 2 2 2 2 2 2 2 2 2 2 2 2
##  [371] 4 2 2 4 4 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 2 4 4 2 2 2 2 2 4 4 2 2 4 4 1 2 2 4 2 2
##  [408] 2 2 2 2 2 4 4 2 4 4 2 2 4 4 2 4 4 2 4 4 4 4 4 2 2 2 4 2 2 2 2 4 2 2 4 4 2 2 2 2 2 2
##  [445] 4 4 2 2 4 4 2 2 2 4 4 2 1 2 2 4 4 2 2 2 4 4 2 2 2 2 2 2 2 2 4 4 2 2 2 2 2 2 4 4 2 2
##  [482] 2 2 2 2 2 2 2 2 2 2 4 4 2 2 4 1 2 4 4 4 2 4 4 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [519] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 4 2 4 4 2 2 2 4 4 2 2 2 4 4 2 2 4 2
##  [556] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 4 4 2 2 2 2 4 2 2 2 2 2 2 2 2 2 4 4 2 4 1 2 2 1 2
##  [593] 2 2 4 4 2 2 2 4 2 4 2 4 2 4 4 4 2 2 2 4 4 2 2 4 2 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4 4 4
##  [630] 4 4 2 4 4 4 4 4 4 4 4 2 2 2 2 2 2 2 4 4 4 4 2 1 2 4 4 2 2 2 4 4 2 2 4 4 4 4 4 4 2 2
##  [667] 4 2 2 2 2 4 4 4 4 4 2 2 2 4 4 4 2 2 4 4 4 4 4 4 4 4 4 4 2 4 4 4 2 4 4 4 4 4 4 4 2 2
##  [704] 4 2 4 4 4 4 4 4 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4
##  [741] 4 4 4 4 2 2 4 4 2 2 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2
##  [778] 4 4 4 4 4 4 4 4 4 2 2 2 2 4 2 4 4 4 2 2 2 2 2 2 2 4 4 4 4 4 2 2 2 4 4 4 2 2 4 4 4 4
##  [815] 2 2 2 2 4 4 4 4 4 4 4 4 4 1 4 4 4 4 4 4 2 2 4 4 1 1 2 4 2 4 2 4 2 4 4 4 4 4 4 4 4 2
##  [852] 2 2 2 2 4 4 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2 2 4 4 4 4 4 4 4 2 4 4 4 4 2
##  [889] 4 2 4 4 2 4 4 4 4 2 4 2 4 2 4 4 4 4 4 4 4 4 4 1 2 2 2 2 2 1 2 4 4 2 4 4 2 2 4 4 2 2
##  [926] 2 2 4 4 2 2 4 4 4 4 4 4 2 2 2 1 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 2 2 2 4 4 2 4
##  [963] 4 2 2 2 2 4 4 2 4 4 2 2 2 2 2 4 4 4 4 1 2 2 4 4 1 2 2 4 4 2 4 4 4 2 4 4 4 4 4 2 4 4 4 4
## [1000] 4 4 2 2 1 4 1 2 2 2 2 2 2 2 4 4 4 4 2 2 1 1 4 2 2 4 4 2 4 4 4 1 4 4 4 4 4 4 4 4 4 2
## [1037] 1 4 2 2 4 4 4 2 2 1 4 4 4 4 2 2 4 4 2 4 2 4 4 4 2 4 4 2 2 2 2 2 2 2 4 4 4 4 2 2 4 4 2 4 4
## [1074] 4 4 2 2 2 2 2 1 2 1 4 2 4 4 2 1 2 2 2 2 4 2 4 2 4 4 2 2 2 2 4 4 4 2 4 1 2 4 4 2
## [1111] 4 4 2 2 1 4 4 4 4 1 4 2 4 4 2 4 1 1 4 2 2 4 4 1 1 4 2 1 2 2 4 4 4 4 2 4 4 4 4 2 4 4
## [1148] 2 2 2 2 4 4 4 2 4 4 4 2 1 2 2 2 2 2 2 4 4 2 2 2 2 1 4 2 4 2 4 4 4 4 4 4 4 2 2 2 2 4
## [1185] 4 4 4 4 4 4 4 2 4 1 4 4 4 4 4 4 1 4 4 1 2 4 4 2 2 2 2 2 2 2 4 4 4 4 2 2 2 4 4 1 2 2 2
## [1222] 2 4 4 2 2 4 4 4 4 1 4 4 1 4 4 4 2 4 3 4 4 4 4 4 4 2 2 4 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4
## [1259] 4 4 2 4 4 2 4 2 4 4 4 2 4 4 1 1 1 4 4 4 4 4 2 4 4 4 2 4 4 4 4 4 4 4 2 1 4 4 4 4 4 4 4 4 4
## [1296] 4 4 4 4 4 4 4 4 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 2 2 4 4 2 2 4 4 4 4 4 4 4 4 4
## [1333] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2 4 2 4 4 4 4 4 2 4
## [1370] 4 2 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2 4 2
## [1407] 2 4 2 4 4 4 4 2 2 2 4 2 1 4 4 4 4 4 4 4 2 2 1 4 4 2 4 4 4 4 4 2 2 2 4 4 4 1 4 4
## [1444] 4 4 4 4 4 4 4 1 1 2 4 4 2 4 4 4 4 2 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 2 1 2 1 4 2
## [1481] 4 2 4 2 4 4 4 4 4 4 4 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 2 2 2 4 4 4 4 4 4 4 4
## [1518] 4 2 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 1 4 2 2 4 4 4 2 1 4 4 4 4
## [1555] 4 4 4 4 4 4 4 4 4 4 4 4 4 1 4 4 4 4 2 4 4 4 3 4 2 4 4 4 4 4 4 4 1 2 2 4 4 4 4
## [1592] 4 4 4 4 4 4 4 4 1 3 1 1 3 3 1 3 3 1 1 1 1 1 1 3 1 4 1 1 3 1 1 1 4 1 3 1 3 1
## [1629] 1 3 1 1 1 3 1 1 1 3 3 3 3 3 1 1 1 3 3 3 3 1 1 1 1 3 1 3 3 1 1 3 3 1 1 1 4
## [1666] 1 3 1 3 3 3 3 1 1 3 1 1 3 4 1 3 3 3 3 3 3 3 3 3 3 3 1 1 1 3 3 1 3 3 3 3 3
## [1703] 3 3 3 3 3 3 1 3 3 3 3 3 4 1 1 3 3 4 3 3 1 1 1 1 3 1 1 1 3 3 3 3 3 1 3 1 1
## [1740] 1 3 1 1 1 1 1 4 1 1 1 3 1 1 2 3 3 1 1 1 1 3 1 3 3 3 3 1 3 3 1 1 1 1 3 1 1
## [1777] 3 4 3 3 3 3 3 3 3 3 3 1 1 3 3 3 1 1 3 3 3 3 3 3 3 3 3 1 1 3 3 2 4 1 1 1 3 1
## [1814] 1 3 3 3 3 3 3 3 3 1 1 1 3 1 3 1 3 4 3 3 3 3 3 3 3 3 1 3 3 1 1 3 3 1 1 1 1 4 3
## [1851] 3 3 1 1 1 1 1 1 1 1 1 3 1 3 1 3 3 4 4 4 3 4 3 4 3 3 3 1 1 1 1 1 3 3 3 3 3 3
## [1888] 3 3 3 1 3 1 3 1 3 1 3 1 4 1 4 1 3 3 3 3 1 1 1 1 1 3 3 3 1 1 1 1 1 1 1 3 1
## [1925] 3 3 1 3 1 1 1 1 1 3 1 1 1 3 1 1 3 1 3 1 1 1 1 3 3 3 1 1 1 1 3 3 3 1 1 1 3 4
## [1962] 1 3 1 1 1 1 1 1 1 1 1 4 1 1 1 1 1 1 1 1 3 3 1 1 1 1 3 1 3 3 1 4 1 3 3 1 1 3
## [1999] 1 1 3 4 3 1 3 1 4 1 1 3 3 1 1 3 3 1 3 1 1 1 3 3 3 3 3 3 3 1 3 3 1 3 4 1 1
```

```
## [2036] 1 1 3 4 1 1 1 3 3 1 1 3 3 1 3 1 1 1 1 3 1 3 1 3 3 3 3 1 3 3 3 1 3 3 3 3 1
## [2073] 1 1 3 1 1 4 1 3 1 3 3 3 1 1 1 1 3 1 1 3 1 1 1 3 1 1 3 3 3 1 1 3 3 4 1 1 1
## [2110] 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 1 3 3 3 1 3 3 1 3
## [2147] 1 1 3 3 1 1 3 1 1 3 1 3 1 3 1 3 1 3 3 1 3 1 3 3 1 3 1 1 1 1 1 1 1 3 1 3 3
## [2184] 1 3 3 1 1 1 3 1 3 1 1 1 3 1 1 1 3 1 1 1 1 1 3 3 3 4 1 1 3 3 3 3 1 1 1 1 3
## [2221] 3 1 1 4 1 4 3 3 1 3 3 1 3 1 1 1 3 3 3 1 3 3 3 3 3 1 3 3 3 3 3 3 3 1 1 1 1
## [2258] 1 3 1 1 4 1 3 1 1 3 1 3 3 1 1 1 3 3 3 1 1 1 1 1 3 3 1 3 1 4 3 1 1 3 3 3 3
## [2295] 3 1 3 3 3 3 1 4 1 1 1 3 1 3 4 3 1 1 3 1 1 3 3 1 1 3 1 1 1 1 1 1 3 1 4 3 3
## [2332] 1 3 3 1 3 3 1 1 1 1 1 3 3 1 3 1 3 1 1 3 3 3 1 1 3 3 1 1 3 3 3 3 3 1 3 1 3
## [2369] 3 1 1 1 3 1 1 1 3 3 3 4 3 3 3 3 3 3 3 1 3 3 1 3 1 3 3 3 3 1 1 3 3 3 3 1 3
## [2406] 3 3 3 3 3 4 1 3 3 1 1 3 1 3 1 3 3 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3
## [2443] 1 1 1 3 1 3 1 1 3 1 3 1 1 3 3 3 3 1 3 1 1 1 1 1 3 3 1 3 3 1 2 1 1 1 1 1 1
## [2480] 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 3 1 1 3 1 3 3 3 1 1 2 1 3 3 1 1 1 4 4 4 1
## [2517] 1 1 1 3 3 3 3 1 1 4 1 3 3 1 3 3 3 3 3 1 3 3 3 3 3 1 1 3 1 3 1 4 1 1 3 1 1
## [2554] 3 1 1 1 1 3 3 1 3 1 3 1 1 3 1 1 1 1 3 1 1 3 1 3 1 2 2 1 1 1 1 1 3 3 1 1 1
## [2591] 2 3 4 1 1 1 3 3 1 1 3 3 1 1 1 1 4 1 1 1 1 1 3 3 1 3 1 1 3 1 1 1 3 1 1 1 4
## [2628] 1 3 1 3 3 3 2 1 2 4 1 1 4 1 4 3 1 1 1 1 1 1 3 3 1 2 1 1 1 3 1 3 1 3 3 3 1
## [2665] 3 3 1 1 1 1 3 1 3 3 1 3 1 3 3 1 3 3 3 1 3 1 1 3 1 3 3 1 1 3 1 3 1 3 1 3 1
## [2702] 1 1 3 1 1 1 1 3 1 1 3 1 4 1 4 3 4 3 3 1 1 2 1 3 1 1 1 1 1 1 3 1 1 3 1 1 3
## [2739] 1 1 3 3 1 1 3 1 1 1 3 3 3 4 1 3 1 3 3 3 3 1 3 1 3 1 1 1 1 1 1 1 3 1 1 3 3
## [2776] 3 1 3 3 3 3 1 1 1 3 1 1 1 1 1 1 3 3 3 3 1 3 3 1 1 1 3 1 1 3 3 3 1 1 1 3 1
## [2813] 1 1 3 1 2 1 1 1 1 1 3 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 2 3 1 1 1 1 3 1 1 1 3
## [2850] 1 1 1 1 3 1 1 3 3 3 3 1 1 3 1 1 1 3 3 3 3 1 3 1 3 1 3 3 1 1 1 1 3 1 1 1 1
## [2887] 1 1 1 1 3 1 1 1 3 1 1 1 1 3 3 3 3 3 1 1 2 1 2 3 1 1 3 3 3 1 1 1 3 1 1 1 1
## [2924] 1 1 3 1 1 1 1 3 3 1 1 1 1 3 3 3 1 1 3 3 1 1 1 1 1 1 1 1 1 3 3 1 3 3 1 1 1
## [2961] 1 1 4 1 1 1 3 3 3 1 1 3 3 1 1 1 1 1 1 1 1 4 1 1 4 1 1 1 3 1 1 1 1 1 1 1 3
## [2998] 3 1 3 3 1 1 1 4 1 1 3 1 1 1 1 1 1 4 1 3 1 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 1
## [3035] 3 3 1 3 3 3 1 1 1 3 1 1 1 1 1 3 1 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3
## [3072] 1 1 1 1 3 1 1 1 1 1 1 1 4 1 1 3 3 3 1 1 3 1 1 1 4 1 1 1 1 1 1 1 1 1 3 3 3
## [3109] 1 1 1 1 3 1 1 3 3 1 1 1 1 1 3 3 3 2 3 1 3 3 1 3 1 3 1 1 1 1 1 3 4 1 1 1 1
## [3146] 1 3 1 1 1 3 1 1 1 1 3 1 3 1 1 2 1 1 2 1 3 1 3 3 3 3 3 1 3 3 1 4 3 3 3 1 1
## [3183] 3 3 3 3 3 3 1 1 3 1 2 3 3 1 3 1 1 3 3 1 1 4 1 1 1 3 3 1 1 1 1 3 1 3 1 3 1
## [3220] 1 3 3 1 1 3 3 3 1 3 1 1 1 1 3 1 3 1 3 1 3 3 3 3 3 3 3 1 1 3 1 3 1 3 3 1 1
## [3257] 3 3 3 3 3 1 3 3 1 1 1 1 1 3 1 3 3 3 3 1 1 1 1 3 3 3 3 3 3 3 3 3 1 4 3 1 3
## [3294] 3 1 3 1 1 1 1 3 4 3 3 1 1 3 4 3 1 1 1 1 3 1 1 1 3 1 1 1 3 1 1 3 1 3 3 1 1
## [3331] 3 1 1 3 1 1 3 1 1 3 1 3 3 3 3 1 3 1 1 3 1 3 1 1 3 3 3 1 3 3 1 1 3 3 3 3 3
## [3368] 1 1 3 1 3 3 1 3 3 1 1 1 3 1 3 2 1 3 1 1 1 1 3 4 1 3 4 3 1 1 3 3 3 1 3 3 3
## [3405] 3 4 3 3 3 1 4 1 1 1 1 1 4 1 1 1 1 1 3 3 1 3 3 3 3 3 3 3 3 1 3 3 1 3 3 1 1
## [3442] 3 2 1 3 3 3 3 1 1 1 1 1 1 3 4 1 1 3 1 3 1 3 1 2 3 3 1 3 1 1 3 3 3 3 1 1 3
## [3479] 3 3 3 3 3 3 3 3 4 3 3 1 3 3 3 3 3 3 3 3 3 1 3 1 3 1 3 3 1 1 1 3 3 1 1 1 1 1
## [3516] 3 1 3 3 3 3 3 3 1 1 2 2 1 1 3 3 3 4 3 1 3 3 1 1 3 3 3 3 3 3 3 3 3 1 3 3 4 1
## [3553] 3 3 3 1 3 3 1 1 2 1 3 3 3 1 3 1 1 1 1 3 3 3 3 3 3 1 3 3 3 3 3 3 3 1 3 3 1
## [3590] 4 3 1 1 3 3 3 3 3 3 1 1 1 1 1 3 3 1 1 1 1 1 1 3 1 1 1 1 1 3 1 1 3 1 3 3 2
## [3627] 3 3 3 1 3 1 3 1 1 1 3 1 1 3 1 1 1 3 1 3 3 3 3 2 3 3 1 1 1 3 1 3 3 3 1 1 1
## [3664] 3 1 1 1 1 1 1 1 3 3 3 1 4 1 1 4 4 4 3 3 1 1 1 3 1 1 3 3 4 3 3 3 3 3 3 1 3
## [3701] 1 3 1 1 3 3 3 3 3 3 3 3 4 3 3 1 1 1 1 3 1 3 3 1 3 3 3 4 1 1 3 3 1 1 3 1 3
## [3738] 1 3 3 3 3 1 3 3 1 3 3 1 1 3 1 1 3 3 1 1 1 1 1 1 2 1 3 1 1 3 3 3 3 3 3 3 3
## [3775] 3 3 1 3 1 3 1 3 3 3 3 2 1 3 3 1 3 3 3 3 1 1 1 3 3 3 1 3 1 1 3 3 1 1 1 1 1
## [3812] 1 1 1 3 1 1 1 1 1 3 1 3 1 3 3 3 3 1 3 1 1 1 2 3 3 3 1 3 3 3 3 3 3 1 3 1 1
## [3849] 3 3 1 3 3 1 1 1 2 3 2 3 1 1 3 3 3 3 3 3 3 1 3 1 4 1 3 3 1 3 3 1 1 3 3 3 3
## [3886] 3 3 3 1 1 1 1 1 1 3 1 3 1 1 1 3 3 3 1 1 1 3 3 1 3 1 3 1 1 3 1 3 3 1 1 4 1
## [3923] 1 3 1 3 1 3 3 3 1 3 1 3 3 3 3 3 1 1 3 1 1 1 3 3 1 1 3 3 3 1 1 1 1 1 1 3 4
## [3960] 3 1 1 3 3 1 3 3 3 1 1 3 1 4 1 1 1 1 3 4 4 1 1 1 1 3 1 1 1 1 1 3 1 1 3 3 3
## [3997] 1 1 1 2 2 3 1 2 3 1 1 3 1 3 3 3 1 4 1 3 3 1 3 1 3 4 1 4 1 1 3 1 3 3 3 3 3
```

```
## [4034] 3 3 3 1 3 3 3 1 3 3 3 3 3 3 3 1 1 3 3 1 1 3 3 3 3 3 3 3 1 1 3 1 3 3 1 1 1 3
## [4071] 1 1 3 1 4 3 1 3 3 1 3 3 3 3 3 3 1 1 3 2 3 3 3 3 1 3 3 3 1 3 3 4 4 3 3 3 1
## [4108] 3 3 3 3 3 1 1 1 1 1 3 1 3 3 1 1 1 3 1 1 1 1 3 1 3 3 3 1 1 3 3 3 1 3 1 3 3 1
## [4145] 1 3 3 3 3 3 3 3 3 1 1 1 3 1 1 1 1 3 1 1 1 3 1 1 3 1 1 1 1 3 1 3 3 3 1 3 3 3
## [4182] 3 3 3 3 3 3 1 3 4 3 1 1 3 1 1 3 1 3 3 1 3 1 1 1 1 1 3 3 1 3 3 1 3 1 3 3 3 1
## [4219] 3 3 3 1 3 1 3 1 1 3 4 1 1 3 3 3 1 3 3 1 1 1 3 2 1 3 1 1 1 3 3 3 4 1 3 3 3
## [4256] 3 1 3 1 3 1 1 1 1 3 1 1 2 3 1 1 3 1 1 1 1 1 1 3 1 3 1 1 1 1 1 3 3 3 1 1 1
## [4293] 1 1 1 3 1 1 1 1 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 1 1 1 3 1 1 1 3 1 1 3 1
## [4330] 3 3 1 1 3 3 1 1 3 1 3 3 3 1 1 1 1 3 1 3 1 1 1 1 1 3 3 1 1 1 1 3 1 1 3 1 1
## [4367] 3 1 1 3 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 1 3 3 3 3 3 3 1 3 1 1 3 1 1 3 3 1 1
## [4404] 1 1 3 3 3 1 1 1 1 1 1 1 1 1 3 1 3 3 3 1 3 1 3 1 1 3 3 3 1 1 1 1 3 3 1 1 1
## [4441] 1 1 1 1 1 1 1 1 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3
## [4478] 1 1 1 1 3 1 1 1 1 1 1 1 1 1 3 1 1 4 3 3 1 3 1 1 3 1 1 3 3 1 1 1 3 3 1 3 1
## [4515] 1 1 1 1 3 3 1 1 1 3 3 1 1 1 3 1 1 3 1 1 1 1 3 1 1 1 1 1 3 1 1 1 1 1 3 3 1 1
## [4552] 1 1 1 1 1 1 1 1 3 1 3 1 1 1 3 3 1 1 1 1 1 3 3 1 1 3 3 1 1 1 3 1 1 1 1 1 1
## [4589] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1 1 3 3 1 3 1 1 1 1 1 4 3 3 1
## [4626] 1 3 1 1 1 3 1 3 3 1 1 3 1 3 3 3 3 2 3 1 1 1 1 3 3 3 3 1 1 1 1 1 3 1 3 3 1
## [4663] 3 3 3 3 1 1 1 1 1 3 1 3 1 3 3 1 1 1 3 1 1 1 1 1 3 1 1 1 3 1 1 2 2 1 1 1 1
## [4700] 1 1 1 1 1 1 1 1 3 3 1 1 1 3 1 1 1 1 1 1 3 1 1 1 3 3 1 1 1 1 1 3 1 1 1 3 1
## [4737] 1 1 3 3 1 1 1 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1
## [4774] 1 3 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 3 3
## [4811] 3 1 3 1 1 1 3 4 1 2 1 1 3 1 1 1 3 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [4848] 1 1 1 3 1 1 1 3 3 3 3 3 3 1 3 1 3 3 1 1 1 3 1 1 1 1 1 4 1 1 3 1 1 1 1 3 1
## [4885] 1 1 1 3 3 1 1 3 1 1 3 3 3 1 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 3 1 1 1 1 1 3 1
## [4922] 1 1 1 1 1 1 1 1 3 3 1 1 1 3 3 3 1 1 1 1 1 3 3 3 3 1 1 3 1 1 1 1 1 1 3 1 1
## [4959] 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 3 1 1 1 1 3 1 1 3 1 1 1 1 1 1 3 3
## [4996] 3 1 3 1 1 1 1 1 1 1 1 3 3 1 1 1 3 3 1 1 3 1 3 1 3 1 1 3 1 3 3 3 1 3 3 3 1
## [5033] 1 1 1 1 3 3 3 1 1 1 1 3 1 3 3 1 1 1 1 3 1 1 1 1 1 1 3 3 1 1 1 3 1 1 3 1 3
## [5070] 1 3 1 1 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 3 1 1 1 1
## [5107] 1 3 3 3 1 1 1 1 1 1 1 1 1 3 1 3 3 1 1 1 3 4 1 3 3 1 3 3 3 1 1 1 1 1 1 3 1
## [5144] 1 3 3 3 1 1 1 3 3 3 1 1 1 1 3 1 1 3 1 3 1 1 1 1 1 1 1 4 1 1 1 1 3 1 1 1 1
## [5181] 1 1 1 1 1 1 3 3 3 1 1 1 1 1 1 1 3 1 3 3 1 1 1 1 3 3 1 3 3 3 1 1 3 3 1 3 1
## [5218] 3 3 3 3 1 3 1 3 1 3 3 3 1 1 3 1 3 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 1 3 1 3 1
## [5255] 1 1 3 1 1 1 1 4 1 1 1 3 1 1 1 3 1 1 1 4 1 1 1 3 1 3 1 1 3 1 3 3 3 1 1 1 1
## [5292] 1 1 3 1 1 3 1 3 1 3 3 3 3 3 1 1 3 1 1 3 3 3 1 3 1 1 3 3 1 1 1 1 1 1 1 1 1
## [5329] 1 3 3 1 1 3 1 1 3 1 3 3 3 3 3 3 3 1 3 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3
## [5366] 3 3 3 3 1 3 3 1 3 1 1 1 1 1 3 1 1 3 3 1 3 3 3 3 3 3 3 3 1 3 3 3 1 1 1 1 1
## [5403] 1 1 1 3 1 1 1 3 1 1 1 3 1 3 1 1 1 1 3 1 3 1 1 1 1 1 1 1 3 3 3 3 3 1 1 3 3
## [5440] 1 3 1 1 1 3 1 3 2 2 1 1 1 1 1 1 1 1 3 3 3 3 3 1 1 1 3 3 3 3 3 3 3 3 3 1 3
## [5477] 1 3 4 3 3 1 1 3 1 1 1 3 3 1 1 3 1 1 1 3 3 1 3 1 4 1 1 1 1 1 1 1 1 1 3 1 1
## [5514] 1 1 1 3 1 1 1 3 1 1 1 1 3 1 3 3 1 1 1 1 1 3 1 3 1 1 3 1 1 1 1 1 1 1 1 3 1
## [5551] 3 1 1 3 1 1 1 1 1 3 3 1 1 3 1 1 3 3 1 3 1 2 3 3 1 1 1 1 1 3 1 1 1 1 3 3 3
## [5588] 1 3 3 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 3 1 1 1 1 3 3 4 3 1 3 4 4 1 3 3 1
## [5625] 1 3 1 1 1 1 1 1 3 1 3 1 1 1 1 4 3 3 3 3 3 3 3 3 3 1 3 1 3 3 3 1 3 1 3 1 4 4
## [5662] 1 1 1 1 1 3 1 3 3 1 3 4 3 3 3 1 1 1 3 3 1 1 1 1 1 1 1 1 3 1 1 3 1 3 1 1 1
## [5699] 1 3 3 3 1 1 1 1 3 1 1 1 1 1 1 1 1 3 3 1 1 3 3 3 1 1 3 3 3 3 1 1 3 3 1 1 1
## [5736] 1 3 1 1 3 3 1 3 3 3 3 1 3 1 3 1 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 3 1 1
## [5773] 2 3 1 3 1 1 3 3 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 3 1 3 1 1 1 1
## [5810] 1 3 3 2 3 3 3 3 1 3 1 1 3 4 1 3 3 1 3 1 1 1 1 1 1 3 3 1 3 1 3 1 1 1 3 3 1
## [5847] 1 1 1 3 1 3 4 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 1 3 1 3 3 3 3 3 1
## [5884] 1 1 1 1 1 3 3 3 1 3 1 3 1 1 1 3 3 3 3 1 1 1 3 1 1 1 3 1 1 1 1 1 4 1 1 1 3
## [5921] 3 1 1 1 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 1 3 3 1 1 2 1 2 1 3 3 1 1 1 1 1 1 1 1
## [5958] 3 3 1 1 1 3 1 3 1 3 1 1 1 1 1 1 1 3 1 1 1 3 1 1 1 1 1 3 3 1 1 3 3 3 3 3 3 3
## [5995] 3 3 3 3 3 3 1 1 3 3 3 1 1 3 1 3 1 1 1 1 3 3 1 3 3 3 3 1 1 3 1 1 3 1 1 3 1
```

```
## [6032] 1 3 1 1 1 1 1 4 3 1 3 4 1 1 1 1 1 3 3 3 3 1 3 3 3 3 1 1 3 1 1 1 3 3 1 3 1
## [6069] 1 1 1 1 2 3 1 1 3 3 4 3 3 1 4 4 1 1 1 1 1 1 1 1 3 1 3 1 3 1 1 1 3 1 1 1 3
## [6106] 1 1 1 1 1 1 1 1 3 1 1 1 1 3 3 3 1 3 3 3 3 1 1 3 3 3 1 3 1 3 3 1 1 1 1 1 1
## [6143] 1 1 1 1 1 3 1 1 1 1 1 1 3 1 3 1 1 1 1 1 1 1 3 3 3 1 1 1 1 1 1 1 1 3 1 3
## [6180] 3 1 1 3 1 1 1 1 1 1 1 3 1 3 1 1 1 4 3 1 1 1 3 1 1 1 1 3 1 4 1 1 3 3 3 1 1
## [6217] 1 1 4 1 1 1 1 1 1 3 1 1 1 3 1 3 1 3 3 1 1 1 3 1 1 1 3 1 1 1 1 3 4 4 1 3 1
## [6254] 3 3 3 3 1 1 1 1 1 1 1 1 3 1 1 1 3 1 1 1 1 1 3 1 3 1 4 1 1 1 1 3 4 3 3 3 3
## [6291] 3 1 3 3 1 1 1 2 3 3 4 4 1 3 1 1 3 1 1 1 3 1 1 1 1 1 1 1 1 1 1 3 3 1 3 1 3
## [6328] 3 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 4 3 3 1 3 3 1 3 1 1 3 1 1 1 1 3 3 3 1 1 1
## [6365] 1 1 1 3 3 3 3 3 1 1 1 1 1 1 3 4 3 3 3 3 3 1 1 1 3 1 1 3 4 3 3 1 1 1 3 1 1 1
## [6402] 1 1 1 1 3 1 1 1 3 1 1 3 1 4 3 1 1 1 3 1 1 1 3 1 1 1 3 1 1 1 1 1 1 1 3 1 1
## [6439] 4 1 1 1 1 1 4 1 1 1 3 1 3 1 1 1 3 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1
## [6476] 1 4 4 3 3 3 1 1 3 3 1 1 1 3 1 1 1 1 3 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 17841.974 8421.783 13176.504 7011.810
##  (between_SS / total_SS = 40.4 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

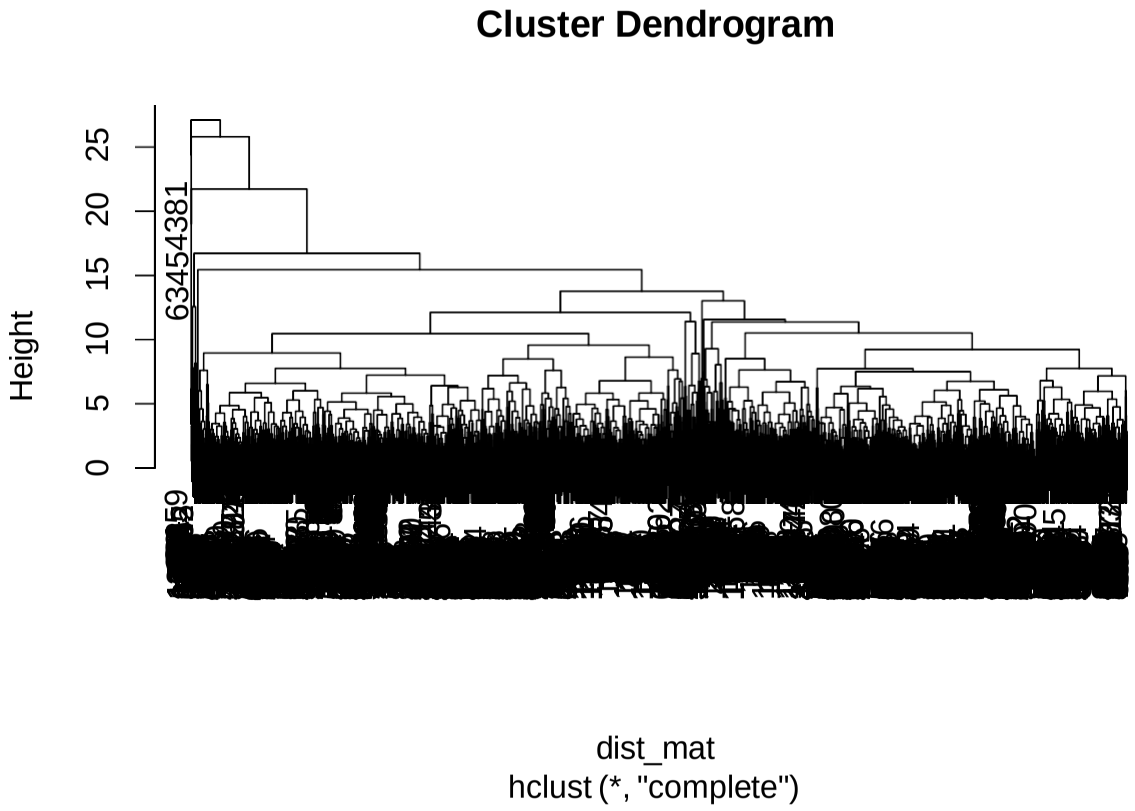
```
fviz_cluster(fit, data = predictors)
```



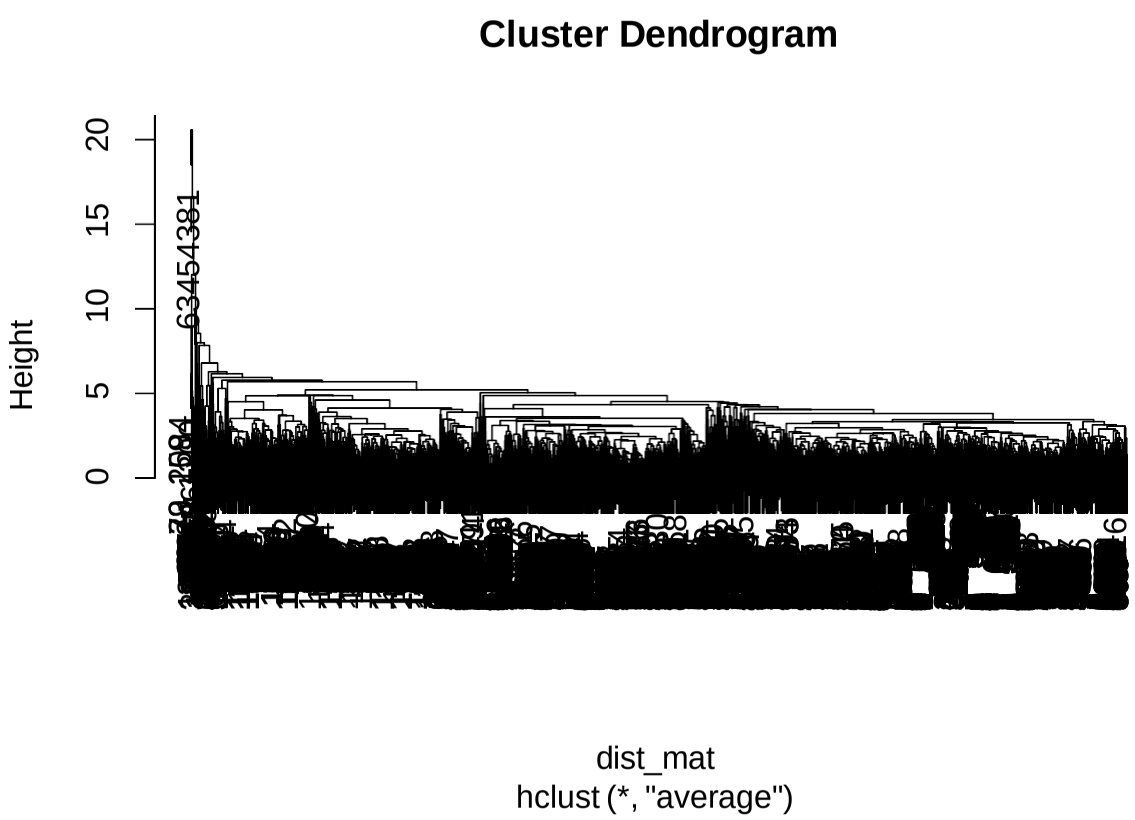
b. Use hierarchical agglomerative clustering (HAC) to cluster the data. Try at least 2 distance functions

and at least 2 linkage functions (cluster distance functions), for a total of 4 parameter combinations. For each parameter combination, perform the clustering.

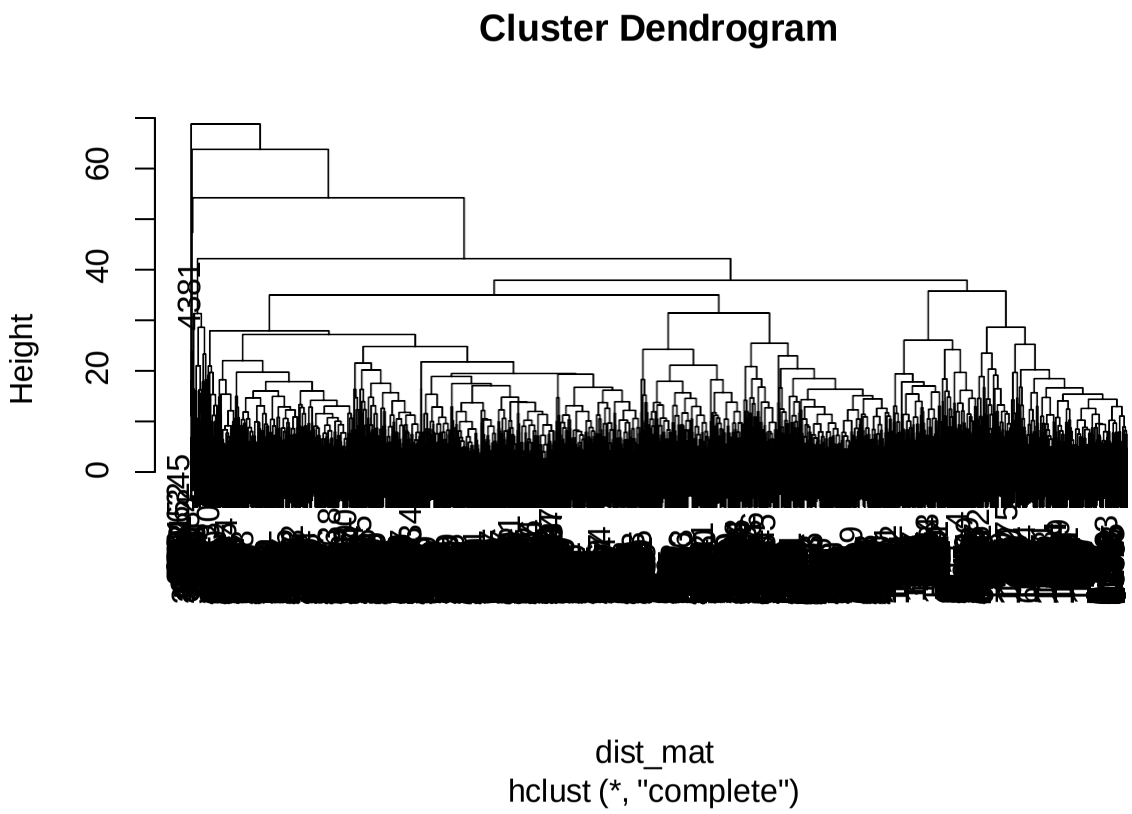
```
#Euclidean and complete linkage:
dist_mat <- dist(predictors, method = 'euclidean')
# Determine assembly/agglomeration method and run hclust
hfit1 <- hclust(dist_mat, method = 'complete')
plot(hfit1)
```



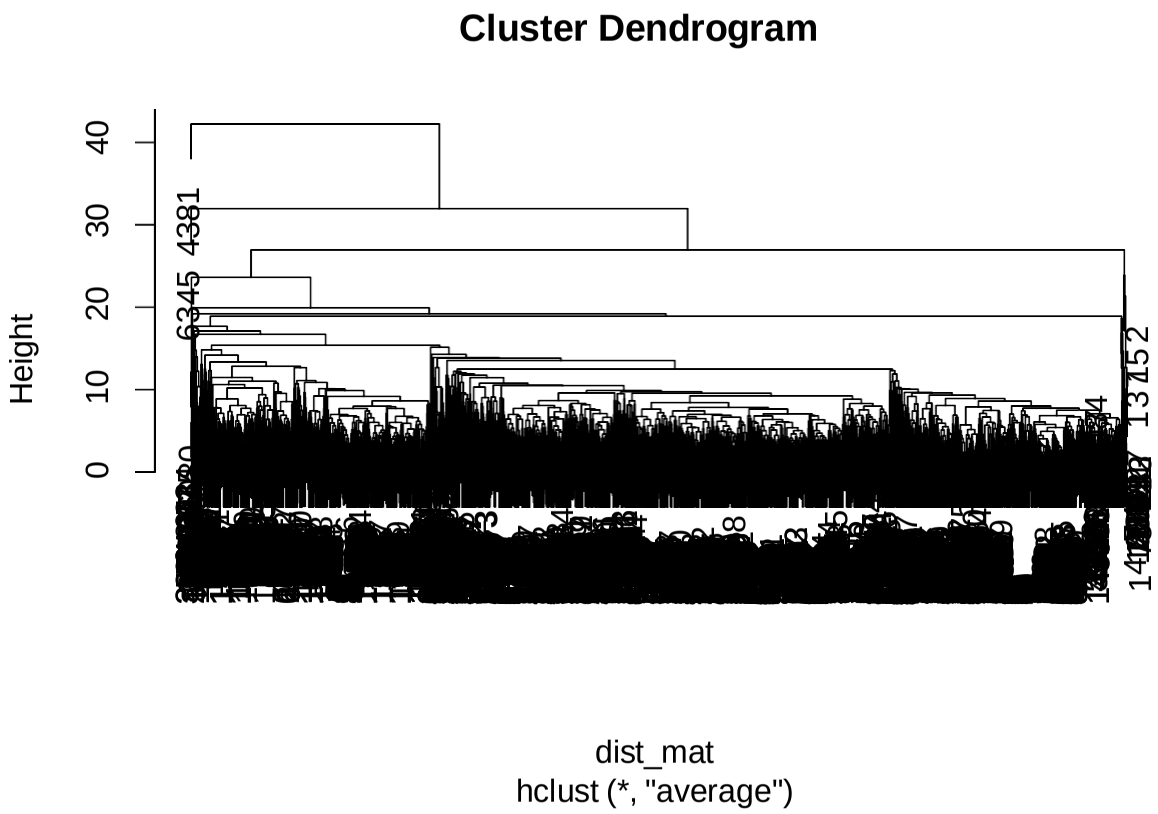

```
#Euclidean and average linkage:
dist_mat <- dist(predictors, method = 'euclidean')
# Determine assembly/agglomeration method and run hclust
hfit2 <- hclust(dist_mat, method = 'average')
plot(hfit2)
```




```
#Manhattan and complete linkage:
dist_mat <- dist(predictors, method = 'manhattan')
# Determine assembly/agglomeration method and run hclust (average uses mean)
hfit3 <- hclust(dist_mat, method = 'complete')
plot(hfit3)
```




```
#Manhattan and average linkage:
dist_mat <- dist(predictors, method = 'manhattan')
# Determine assembly/agglomeration method and run hclust (average uses mean)
hfit4 <- hclust(dist_mat, method = 'average')
plot(hfit4)
```




```
#built the new model
h1 <- cutree(hfit1, k=4)
h2 <- cutree(hfit2, k=4)
h3 <- cutree(hfit3, k=4)
h4 <- cutree(hfit4, k=4)
```

c. Compare the k-means and HAC clusterings by creating a crosstabulation between their labels.

```
# Create a dataframe for results of 4 HAC
result1 <- data.frame(Type = wines$type, HAC1 = h1, Kmeans = fit$cluster)
result2 <- data.frame(Type = wines$type, HAC2 = h2, Kmeans = fit$cluster)
result3 <- data.frame(Type = wines$type, HAC3 = h3, Kmeans = fit$cluster)
result4 <- data.frame(Type = wines$type, HAC4 = h4, Kmeans = fit$cluster)
```

```
# Crosstab for HAC
result1 %>% group_by(HAC1) %>% select(HAC1, Type) %>% table()
```

```
##        Type
## HAC1 red white
##    1 1597  4896
##    2    2     0
##    3    0     1
##    4    0     1
```

```
result2 %>% group_by(HAC2) %>% select(HAC2, Type) %>% table()
```

```
##         Type
## HAC2  red white
##    1 1575  4895
##    2   24     1
##    3    0     1
##    4    0     1
```

```
result3 %>% group_by(HAC3) %>% select(HAC3, Type) %>% table()
```

```
##         Type
## HAC3  red white
##    1 1595  4889
##    2    4     0
##    3    0     8
##    4    0     1
```

```
result4 %>% group_by(HAC4) %>% select(HAC4, Type) %>% table()
```

```
##         Type
## HAC4  red white
##    1 1576  4896
##    2   23     0
##    3    0     1
##    4    0     1
```

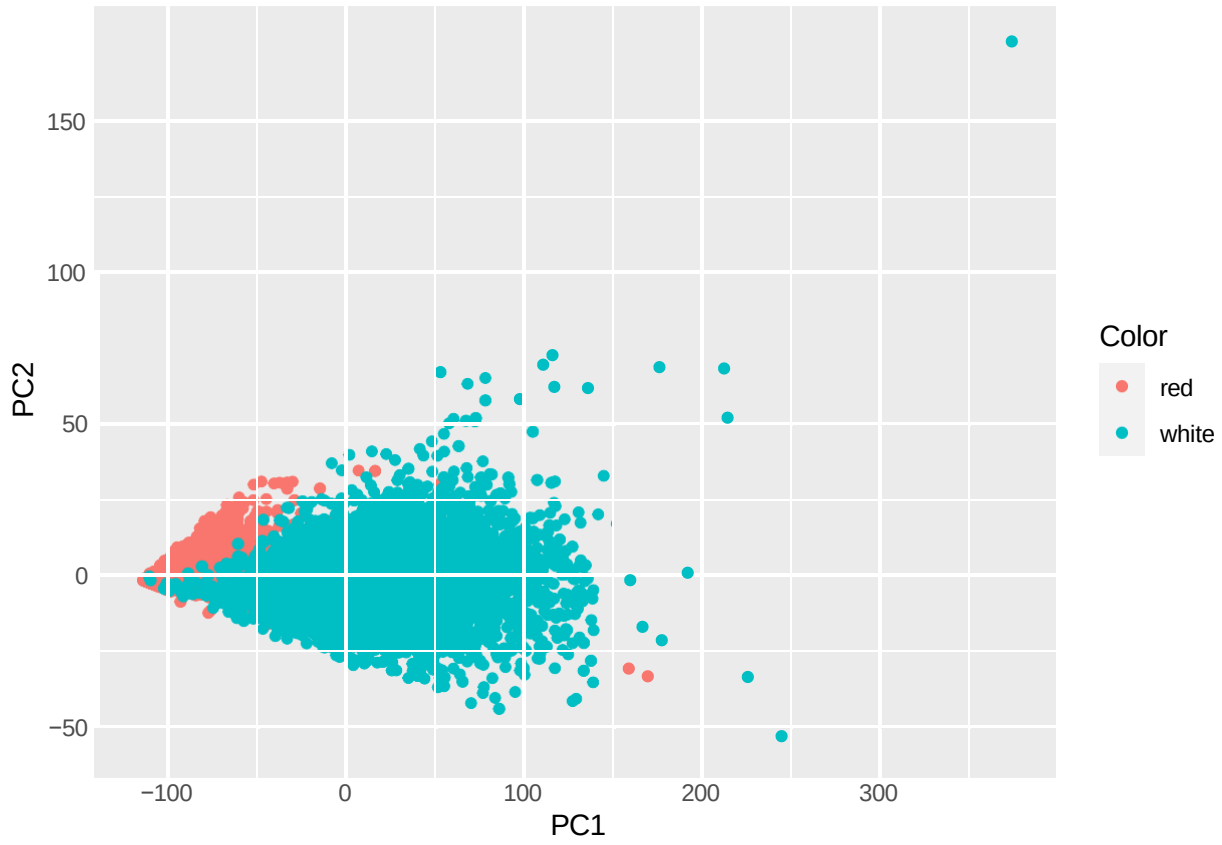
```
# Crosstab for K Means
result1 %>% group_by(Kmeans) %>% select(Kmeans, Type) %>% table()
```

```
##            Type
##  Kmeans  red  white
##       1    60  2775
##       2  609    52
##       3     3  1932
##       4  927   139
```

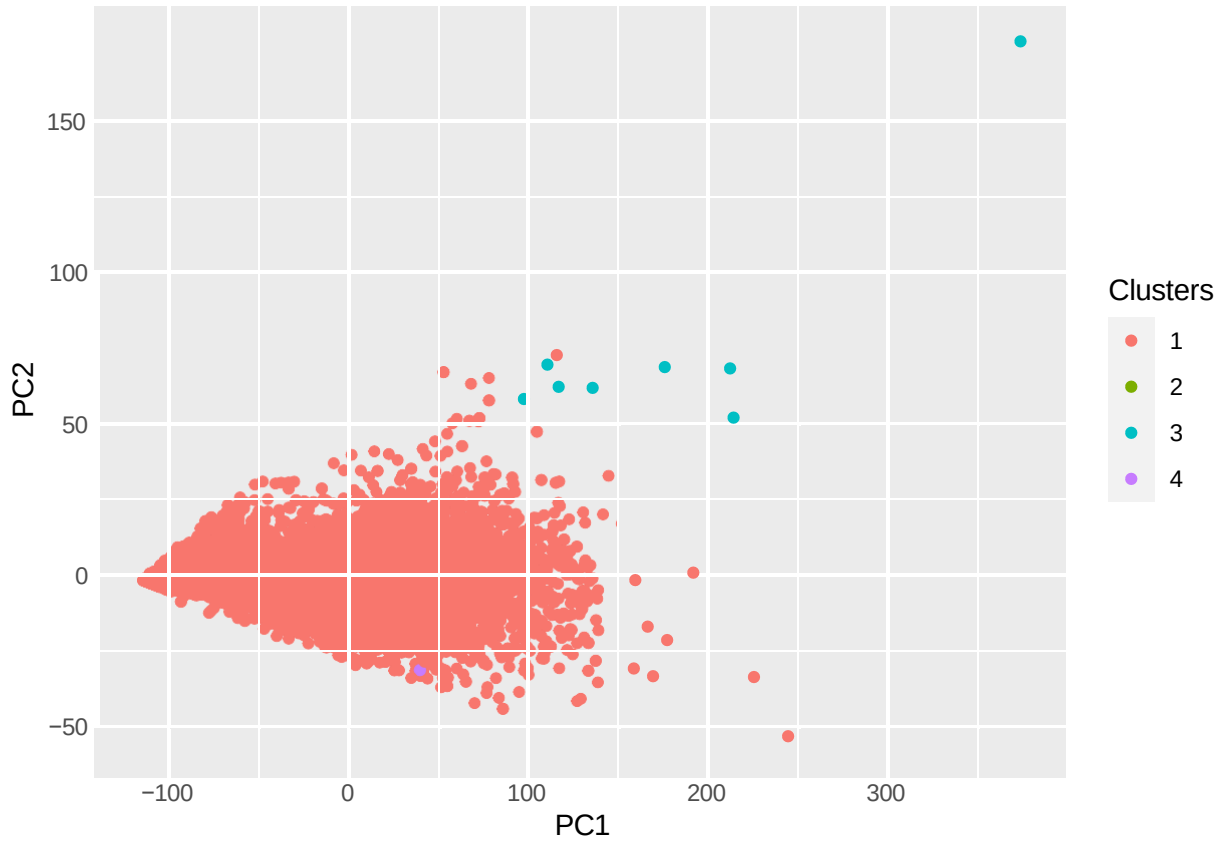
d. For comparison – use PCA to visualize the data in a scatterplot. Create 3 separate plots: use the color of the points to show (1) the type label, (2) the k-means cluster labels and (3) the HAC cluster labels.

```
set.seed(123)
pca = prcomp(dummies)
rotated_data_cluster = as.data.frame(pca$x)
```

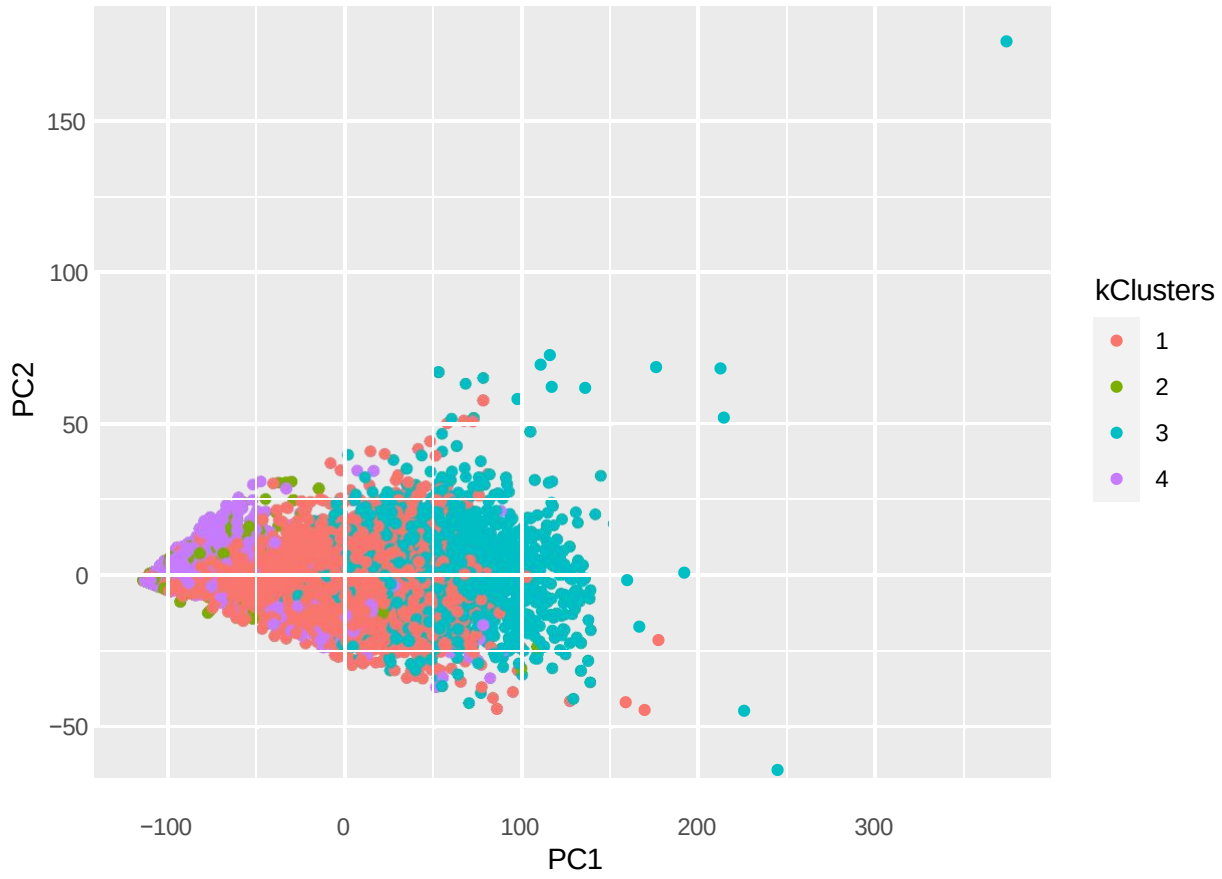
```
# add the type to the rotated data
rotated_data_cluster$Color <- wines$type
ggplot(data = rotated_data_cluster, aes(x = PC1, y = PC2, col = Color)) + geom_point()
```




```
rotated_data_cluster$Clusters = as.factor(h3)
# Plot and color by labels
ggplot(data = rotated_data_cluster, aes(x = PC1, y = PC2, col = Clusters)) + geom_point()
```




```
rotated_data_cluster$kClusters = as.factor(fit$cluster)
# Plot and color by labels
ggplot(data = rotated_data_cluster, aes(x = PC1, y = PC2, col = kClusters)) + geom_point()
```



e. Consider the results of C and D and explain the differences between the clustering results in terms of how the algorithms work.

K-means uses a pre-specified K value, while HAC doesn't. We can see that the clustering method seems to be less random in the K-means method.

Problem 4

Back to the Starwars data from a previous assignment! Remember that the variable that lists the actual names and the variables that are actually lists will be a problem, so remove them (name, films, vehicles, starships). Make sure to double check the types of the variables, i.e., that they are numerical or factors as you expect.

```
df_starwars = starwars
df_starwars = select(df_starwars,-c("name", "vehicles", "starships", "films"))
str(df_starwars)
```

```
## tibble [87 x 10] (S3: tbl_df/tbl/data.frame)
##  $ height    : int [1:87] 172 167 96 202 150 178 165 97 183 182 ...
##  $ mass      : num [1:87] 77 75 32 136 49 120 75 32 84 77 ...
##  $ hair_color: chr [1:87] "blond" NA NA "none" ...
##  $ skin_color: chr [1:87] "fair" "gold" "white, blue" "white" ...
```

```
##  $ eye_color : chr [1:87] "blue" "yellow" "red" "yellow" ...
##  $ birth_year: num [1:87] 19 112 33 41.9 19 52 47 NA 24 57 ...
##  $ sex       : chr [1:87] "male" "none" "none" "male" ...
##  $ gender    : chr [1:87] "masculine" "masculine" "masculine" "masculine" ...
##  $ homeworld : chr [1:87] "Tatooine" "Tatooine" "Naboo" "Tatooine" ...
##  $ species   : chr [1:87] "Human" "Droid" "Droid" "Human" ...
```

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

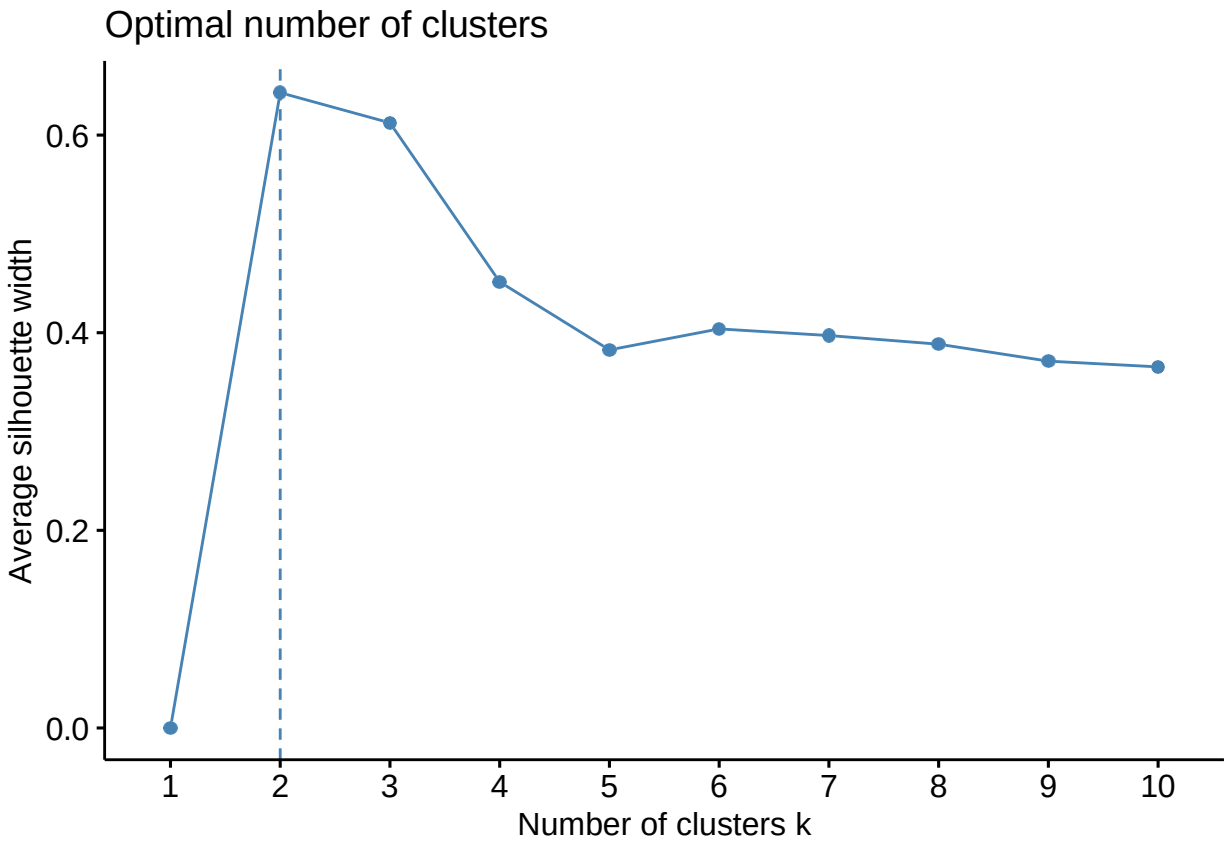
a. Use hierarchical agglomerative clustering to cluster the Starwars data. This time we can leave the categorical variables in place, because we will use the gower metric from daisy in the cluster library to get the distances. Use average linkage. Determine the best number of clusters.

```
df_starwars <- as.data.frame(unclass(df_starwars),     # Convert all columns to factor
                             stringsAsFactors = TRUE)
str(df_starwars)
```

```
## 'data.frame':    29 obs. of  10 variables:
##  $ height    : int  172 202 150 178 165 183 182 188 228 180 ...
##  $ mass      : num  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  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 ...
```

```
dist_mat <- daisy(df_starwars, metric = "gower")
preproc <- preProcess(df_starwars, method=c("center", "scale"))
predictors <- predict(preproc, df_starwars)
```

```
fviz_nbclust(predictors, FUN = hcut, method = "silhouette")
```

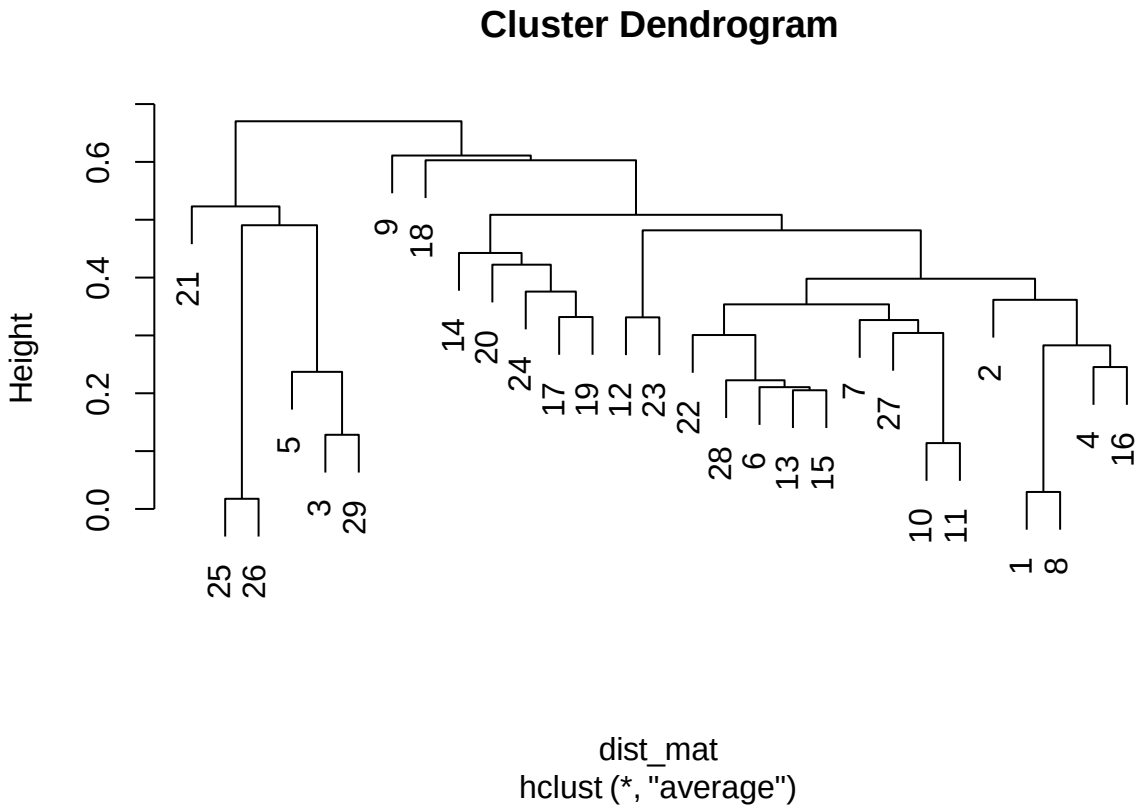



```
# Determine assembly/agglomeration method and run hclust
hfit <- hclust(dist_mat, method = 'average')
# Build the new model
h2 <- cutree(hfit, k=2)
summary(h2)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   1.000   1.207   1.000   2.000
```

b. Produce the dendogram for (a). How might an anomaly show up in a dendogram? Do you see a Starwars character who does not seem to fit in easily? What is the advantage of considering anomalies this way as opposed to looking for unusual values relative to the mean and standard deviations, as we considered earlier in the course? Disadvantages?

```
hfit <- hclust(dist_mat, method = 'average')
plot(hfit)
```



c. Use dummy variables to make this data fully numeric and then use k-means to cluster. Choose the best number of clusters.

```
dummy <- dummyVars(gender ~ ., data = df_starwars)
dummies <- as.data.frame(predict(dummy, newdata = df_starwars))
head(dummies)
```

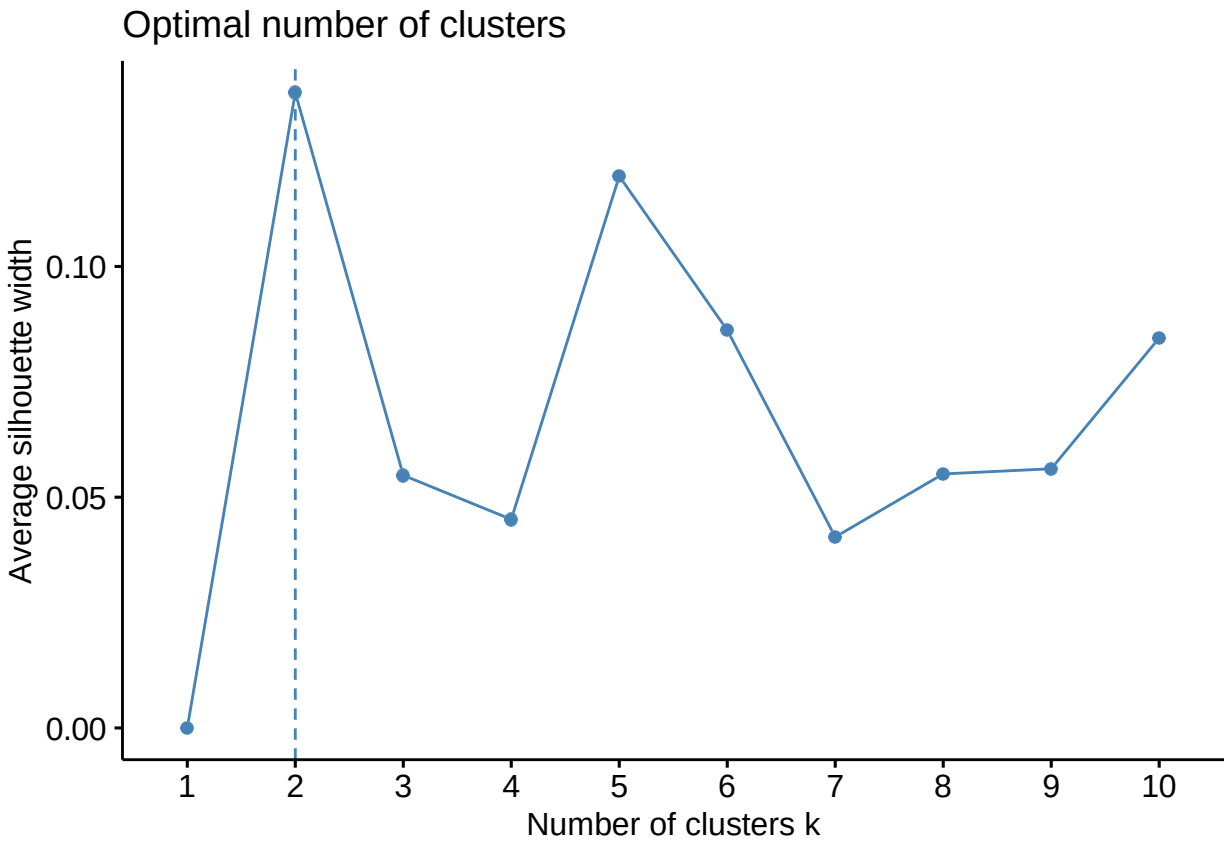
```
##   height  mass  hair_color.auburn,  white  hair_color.black  hair_color.blond
## 1    172    77                    0                     0                 1
## 2    202   136                    0                     0                 0
## 3    150    49                    0                     0                 0
## 4    178   120                    0                     0                 0
## 5    165    75                    0                     0                 0
## 6    183    84                    0                     1                 0
##   hair_color.brown  hair_color.brown,  grey  hair_color.grey  hair_color.none
## 1                0                     0                 0                0
## 2                0                     0                 0                1
## 3                1                     0                 0                0
## 4                0                     1                 0                0
## 5                1                     0                 0                0
## 6                0                     0                 0                0
##   hair_color.white  skin_color.blue  skin_color.brown  skin_color.brown  mottle
## 1                0                0                 0                 0
## 2                0                0                 0                 0
## 3                0                0                 0                 0
## 4                0                0                 0                 0
```

```
## 5                    0                  0                  0                          0
## 6                    0                  0                  0                          0
##   skin_color.dark skin_color.fair skin_color.green skin_color.light
## 1               0               1                0                0
## 2               0               0                0                0
## 3               0               0                0                1
## 4               0               0                0                1
## 5               0               0                0                1
## 6               0               0                0                1
##   skin_color.orange skin_color.pale skin_color.red skin_color.tan
## 1                 0                0              0              0
## 2                 0                0              0              0
## 3                 0                0              0              0
## 4                 0                0              0              0
## 5                 0                0              0              0
## 6                 0                0              0              0
##   skin_color.unknown skin_color.white skin_color.yellow eye_color.black
## 1                  0                0                 0               0
## 2                  0                1                 0               0
## 3                  0                0                 0               0
## 4                  0                0                 0               0
## 5                  0                0                 0               0
## 6                  0                0                 0               0
##   eye_color.blue eye_color.blue-gray eye_color.brown eye_color.hazel
## 1              1                   0                0              0
## 2              0                   0                0              0
## 3              0                   0                1              0
## 4              1                   0                0              0
## 5              1                   0                0              0
## 6              0                   0                1              0
##   eye_color.orange eye_color.red eye_color.yellow birth_year sex.female
## 1                0             0                0       19.0          0
## 2                0             0                1       41.9          0
## 3                0             0                0       19.0          1
## 4                0             0                0       52.0          0
## 5                0             0                0       47.0          1
## 6                0             0                0       24.0          0
##   sex.male homeworld.Alderaan homeworld.Bespin homeworld.Cerea
## 1        1                  0                0               0
## 2        1                  0                0               0
## 3        0                  1                0               0
## 4        1                  0                0               0
## 5        0                  0                0               0
## 6        1                  0                0               0
##   homeworld.Concord Dawn homeworld.Corellia homeworld.Dathomir homeworld.Dorin
## 1                      0                  0                 0                0
## 2                      0                  0                 0                0
## 3                      0                  0                 0                0
## 4                      0                  0                 0                0
## 5                      0                  0                 0                0
## 6                      0                  0                 0                0
##   homeworld.Endor homeworld.Haruun Kal homeworld.Kamino homeworld.Kashyyyk
## 1               0                    0                0                 0
## 2               0                    0                0                 0
```

```
## 3                    0                     0                0                    0
## 4                    0                     0                0                    0
## 5                    0                     0                0                    0
## 6                    0                     0                0                    0
##   homeworld.Mirial homeworld.Mon Cala  homeworld.Naboo homeworld.Ryloth
## 1                0                   0               0                0
## 2                0                   0               0                0
## 3                0                   0               0                0
## 4                0                   0               0                0
## 5                0                   0               0                0
## 6                0                   0               0                0
##   homeworld.Serenno homeworld.Socorro  homeworld.Stewjon homeworld.Tatooine
## 1                 0                  0                 0                 1
## 2                 0                  0                 0                 1
## 3                 0                  0                 0                 0
## 4                 0                  0                 0                 1
## 5                 0                  0                 0                 1
## 6                 0                  0                 0                 1
##   homeworld.Trandosha species.Cerean species.Ewok species.Gungan species.Human
## 1                   0              0             0              0             1
## 2                   0              0             0              0             1
## 3                   0              0             0              0             1
## 4                   0              0             0              0             1
## 5                   0              0             0              0             1
## 6                   0              0             0              0             1
##   species.Kel  Dor  species.Mirialan  species.Mon  Calamari  species.Trandoshan
## 1                 0                 0                  0                   0
## 2                 0                 0                  0                   0
## 3                 0                 0                  0                   0
## 4                 0                 0                  0                   0
## 5                 0                 0                  0                   0
## 6                 0                 0                  0                   0
##    species.Twi'lek  species.Wookiee  species.Zabrak
## 1                0                0               0
## 2                0                0               0
## 3                0                0               0
## 4                0                0               0
## 5                0                0               0
## 6                0                0               0
```

```
predictors <- dummies
# Center scale allows us to standardize the data
preproc <- preProcess(predictors, method=c("center", "scale"))
# We have to call predict to fit our data based on preprocessing
predictors <- predict(preproc, predictors)
```

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




```
fit <- kmeans(predictors, centers = 2, nstart = 25)
fit
```

```
## K-means clustering with 2 clusters of sizes 20, 9
##
## Cluster means:
##         height       mass hair_color.auburn, white hair_color.black
## 1 -0.2784089 -0.2751649                0.0835629          0.2258418
## 2  0.6186863  0.6114774               -0.1856953         -0.5018706
##   hair_color.blond hair_color.brown hair_color.brown, grey hair_color.grey
## 1        0.1203443        0.1346073              0.0835629       -0.1856953
## 2       -0.2674319       -0.2991273             -0.1856953        0.4126563
##   hair_color.none hair_color.white skin_color.blue skin_color.brown
## 1      -0.3405624      -0.07354376       0.0835629        0.0835629
## 2       0.7568054       0.16343058      -0.1856953       -0.1856953
##   skin_color.brown mottle skin_color.dark skin_color.fair skin_color.green
## 1                -0.1856953       0.1203443       0.2494194       -0.1856953
## 2                 0.4126563      -0.2674319      -0.5542653        0.4126563
##   skin_color.light skin_color.orange skin_color.pale skin_color.red
## 1        0.2258418        -0.2674319      -0.2674319      -0.1856953
## 2       -0.5018706         0.5942930       0.5942930       0.4126563
##   skin_color.tan skin_color.unknown skin_color.white skin_color.yellow
## 1      0.0835629           -0.1856953       -0.1856953         0.1203443
## 2     -0.1856953            0.4126563        0.4126563        -0.2674319
##   eye_color.black eye_color.blue eye_color.blue-gray eye_color.brown
## 1      -0.1856953       0.1629911            0.0835629       0.3207862
```

```
## 2          0.4126563       -0.3622024              -0.1856953       -0.7128583
##   eye_color.hazel eye_color.orange eye_color.red eye_color.yellow birth_year
## 1       0.1203443       -0.2674319    -0.1856953       -0.3930429 -0.2442778
## 2      -0.2674319        0.5942930     0.4126563        0.8734288  0.5428396
##   sex.female   sex.male homeworld.Alderaan homeworld.Bespin homeworld.Cerea
## 1  0.2258418 -0.2258418         0.0835629       0.0835629      -0.1856953
## 2 -0.5018706  0.5018706        -0.1856953      -0.1856953       0.4126563
##   homeworld.Concord Dawn homeworld.Corellia homeworld.Dathomir homeworld.Dorin
## 1              0.0835629         0.1203443        -0.1856953      -0.1856953
## 2             -0.1856953        -0.2674319         0.4126563       0.4126563
##   homeworld.Endor homeworld.Haruun Kal homeworld.Kamino homeworld.Kashyyyk
## 1       0.0835629             0.0835629       0.0835629       -0.1856953
## 2      -0.1856953            -0.1856953      -0.1856953        0.4126563
##   homeworld.Mirial homeworld.Mon Cala homeworld.Naboo homeworld.Ryloth
## 1        0.1203443         -0.1856953      -0.1724505       0.0835629
## 2       -0.2674319          0.4126563       0.3832233      -0.1856953
##   homeworld.Serenno homeworld.Socorro homeworld.Stewjon homeworld.Tatooine
## 1         0.0835629         0.0835629       0.0835629        0.1045564
## 2        -0.1856953        -0.1856953      -0.1856953       -0.2323475
##   homeworld.Trandosha species.Cerean species.Ewok species.Gungan species.Human
## 1          -0.1856953     -0.1856953    0.0835629     -0.1856953      0.3631205
## 2           0.4126563      0.4126563   -0.1856953      0.4126563     -0.8069344
##   species.Kel Dor species.Mirialan species.Mon Calamari species.Trandoshan
## 1      -0.1856953        0.1203443           -0.1856953          -0.1856953
## 2       0.4126563       -0.2674319            0.4126563           0.4126563
##   species.Twi'lek species.Wookiee species.Zabrak
## 1       0.0835629      -0.1856953     -0.1856953
## 2      -0.1856953       0.4126563      0.4126563
##
## Clustering vector:
##  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
##  1  2  1  1  1  1  1  1  2  1  1  2  1  2  1  1  2  1  2  2  1  1  2  2  1  1
## 27 28 29
##  1  1  1
##
## Within cluster sum of squares by cluster:
## [1] 985.1128 718.4747
##  (between_SS / total_SS =  7.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

d. Compare the HAC and k-means clusterings with a crosstabulation.

Below results shows the comparison of cluters with a crosstabulation.

```
result <- data.frame(Gender = df_starwars$gender, HAC2 = h2, Kmeans = fit$cluster)
#create a cross tab for HAC
```

```
#create a cross tab for HAC
result %>% group_by(HAC2) %>% select(HAC2, Gender) %>% table()
```

```
##         Gender
## HAC2 feminine masculine
##    1        0         23
##    2        6          0
```

```
#create a cross tab for k-means
result %>% group_by(Kmeans) %>% select(Kmeans, Gender) %>% table()
```

```
##            Gender
## Kmeans feminine masculine
##      1        6         14
##      2        0          9
```