Unsupervised Learning with K-means Clustering and Hierarchical Clustering

Types of machine learning:

● Unsupervised learning :

This refers to finding structure in unlabeled data so this does not require labeled observations.

● Supervised learning :

This refers to making predictions for a target variable for unseen observations based on labeled data/observations. This involves regression for continuous values and classification for discrete values.

Unsupervised learning:

In unsupervised learning, the input data does not have labeled responses. There are only input variables and no output variables. It can be done in two ways:

● Clustering: Finding homogeneous subgroups within a larger group

● Dimensionality reduction: Finding patterns in the features of the data

Clustering

Cluster analysis consists of finding homogeneous subgroups within a larger group. Hidden patterns or groups are found in unlabeled data. K-means clustering and hierarchical clustering are going to be looked at today. Clustering ensures that:

● The data should be similar within a cluster.

● The data should be dissimilar between clusters.

Can you think of some applications of clustering?

K-means clustering

K-Means clustering partitions observations into a pre-defined number (k) of disjoint subsets based on their similarity. The input variables must be numerical.

The number of clusters (k) has to be specified and this may not always be possible.

"nstart" in the “kmeans” function refers to the number of initial configurations that are tried before the one that gives the lowest within-cluster variation (where the observations in a cluster are as similar as possible to each other) is chosen.

Example of k-means clustering with the “iris” dataset :

To introduce the concept of k-means clustering, we will now look at the “iris” datset from the “datasets” package. kmeans in this case is set to 3 as it is already known that the “iris” dataset has only three species.

```
install.packages("datasets")

library(datasets)

head(iris)
```

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

```
#set a seed to get the same result every time the code is run

set.seed(20)

# k-means clustering using all columns except “Species”

i1 <- kmeans(iris[, 1:4], 3, nstart = 20)

i1
```

```
> i1
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [29] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2
 [57] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2
 [85] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 2 3 3 3 3 3
[113] 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3
[141] 3 3 2 3 3 3 2 3 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"
[4] "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

To find how the observations were classified, the “table” function is used.

table(i1$cluster, iris$Species)

```
> table(i1$cluster, iris$Species)

    setosa versicolor virginica
  1     50          0         0
  2      0         48        14
  3      0          2        36
```

The output of the kmeans() function consists primarily of:

1. cluster - This is the cluster to which each point/observation is allocated.
2. centers - This is a matrix of cluster centers or means.

3. totss: This is the total sum of squares (TSS) or the total variance in the data.
4. withinss - This is the within-cluster sum of squares.
5. tot.withinss - This is the total of within-cluster sum of squares. It is equal to sum(withinss).
6. betweenss: This is the between-cluster sum of squares.
7. size - This is the size of each cluster.

How is the optimal number of clusters determined?

There are several methods, some of which are explained below:

- Elbow method
- Average Silhouette method
- NbClust package
- From the context of the problem

1. Elbow Method:

This minimizes the total within-cluster variation or total within-cluster sum of squares.

Steps:

1. Do k-means clustering for different values of *k*.
2. Calculate the total within-cluster sum of squares (wss) for each *k*.
3. Plot the total within-cluster sum of squares against *k*.
4. Note the “elbow” in the plot where the quality of the clustering stops improving substantially (or the total within-cluster sum of squares stops decreasing substantially) as the number of clusters or the complexity of clustering increases.

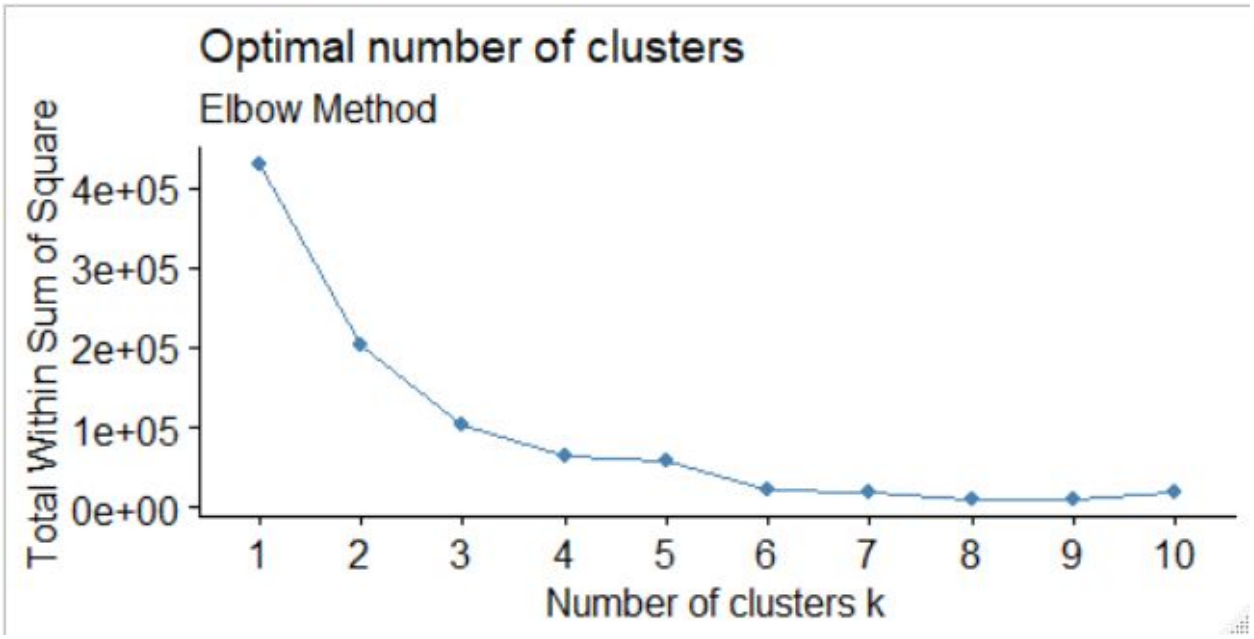
We will now look at how to use the elbow method to find the optimal number of k-means clusters for the “nutrients.meat.fish.fowl.1959” dataset. This is a table with the nutrient levels in meat, fish and fowl. More details can be found at [nutrients.meat.fish.fowl.1959 function | R Documentation]

Example of the elbow method with the factoextra:fviz_nbclust function :

```
#Elbow Method
set.seed(20)
install.packages("cluster.datasets")
library("cluster.datasets")
data(nutrients.meat.fish.fowl.1959)
nutrients<- na.omit(nutrients.meat.fish.fowl.1959)
head(nutrients)
```

```
> head(nutrients)
              name energy protein fat calcium iron
1    Braised beef    340      20  28       9  2.6
2       Hamburger    245      21  17       9  2.7
3      Roast beef    420      15  39       7  2.0
4       Beefsteak    375      19  32       9  2.6
5     Canned beef    180      22  10      17  3.7
6 Broiled chicken    115      20   3       8  1.4
```

```
install.packages("factoextra")
library("factoextra")
# only the columns 2 to 6 are used for k-means clustering
fviz_nbclust(nutrients[,2:6], kmeans, method = "wss")+labs(subtitle = "Elbow Method")
```



2. Average Silhouette method:

This method measures how good the clustering is - that is how similar an object is to its cluster neighbours. This value needs to be high for the clustering to be considered good. The average silhouette method works by looking at the average silhouette width over a range of values of k for all the observations and choosing the value of k that gives the highest average silhouette width as the optimal number of clusters.

The silhouette width for each observation can be:

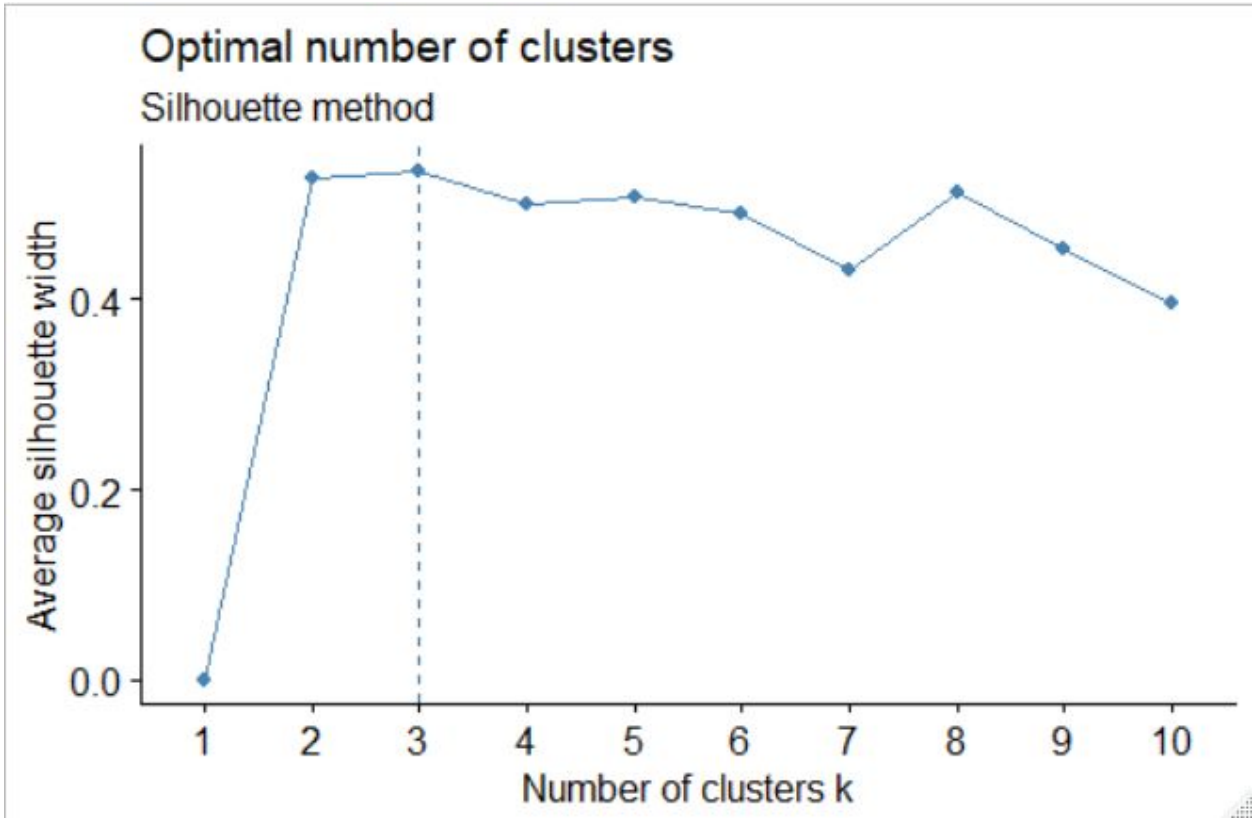
Positive: In this case, the observation is in the right cluster. The higher this value, the better the clustering is.

Negative : In this case, the observation is in the wrong cluster.

Zero: The observation is placed between two clusters.

Example of the Average Silhouette method with the cluster:fviz_nbclust function :

```
#Average Silhouette Method

set.seed(20)

install.packages("cluster")

library(cluster)

fviz_nbclust(nutrients[,2:6], kmeans, method = "silhouette")+

  labs(subtitle = "Silhouette method")
```



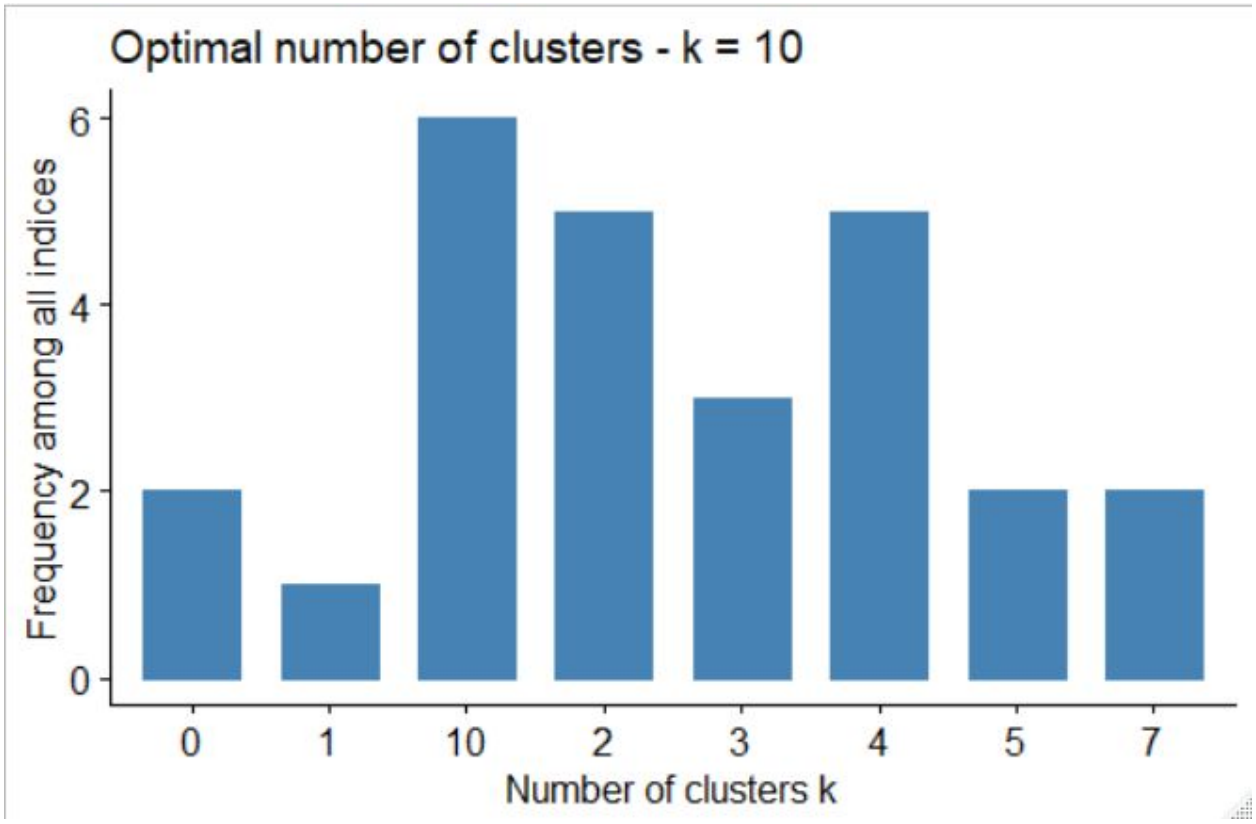
3. NbClust package

The NbClust package can be used to determine the best number of clusters.

Example of using the NbClust: NbClust function:

```
set.seed(20)

install.packages("NbClust")

library("NbClust")

install.packages("dplyr")

library(dplyr)

n1 <- nutrients[,2:6]%>%
```

```
  NbClust(distance = "euclidean",
          min.nc = 2, max.nc = 10,  method = "complete", index ="all")
# Visualize
install.packages("factoextra")
library(factoextra)
fviz_nbclust(n1)
```



4. Use the context of the problem to determine the optimal number of clusters.

For instance, if we wanted to cluster the customers of a grocery chain according to whether they spent more money shopping in person or online, the dataset of customers should be clustered with k = 2 clusters.

The final k-means clustering model

In this case we are going to go with 10 clusters as recommended by the NbClust: NbClust function.

Compute k-means clustering with k = 10

set.seed(20)

c10 <- kmeans(nutrients[,2:6], 10, nstart = 50)

c10

```
> c10
K-means clustering with 10 clusters of sizes 2, 3, 1, 1, 4, 5,
 1, 3, 2, 5

Cluster means:
   energy  protein      fat    calcium     iron
1   57.50  9.00000  1.00000  78.000000 5.700000
2  270.00 19.66667 20.66667   9.000000 2.533333
3  180.00 22.00000  9.00000 367.000000 2.500000
4  420.00 15.00000 39.00000   7.000000 2.000000
5  118.75 18.00000  3.50000  21.500000 0.825000
6  173.00 24.20000  7.60000  11.800000 3.000000
7  110.00 23.00000  1.00000  98.000000 2.600000
8  200.00 17.66667 12.66667   8.666667 1.600000
9  137.50 16.50000  7.00000 158.000000 1.250000
10 350.00 19.40000 29.40000   9.000000 2.520000

Clustering vector:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
10  2  4 10  6  5  6  6  2  2 10 10 10  8  6  5  1  1  5  5
21 22 23 24 25 26 27
 8  9  8  9  3  6  7

Within cluster sum of squares by cluster:
 [1]  352.680 1587.420    0.000    0.000 1923.237  483.280
 [7]    0.000  105.260  623.605  962.428
 (between_SS / total_SS =  98.6 %)
```

```
Available components:

[1] "cluster"      "centers"      "totss"
[4] "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
```

Visualize the results:

visualize the clusters using fviz_cluster

install.packages("factoextra")

library(factoextra)

fviz_cluster(c10, data = nutrients[,2:6])

In case there are more than two variables, Principal Component Analysis (PCA) is performed and the first two principal components that explain most of the variation in the data are used to plot the cluster plot.

Group the data by cluster and summarize :

Here, the very useful pipe operator from the “dplyr” package is used to get an idea of the observations in each cluster.

#use the “dplyr” package to summarize the data by cluster

```
install.packages("tibble")

library(tibble)

install.packages("dplyr")

library(dplyr)

nutrients[,2:6] %>%

 as_tibble() %>%

 mutate(Cluster = c10$cluster) %>%

 group_by(Cluster) %>%

 summarise_all("mean")
```

```
# A tibble: 10 x 6
   Cluster energy protein   fat calcium  iron
     <int>  <dbl>   <dbl> <dbl>   <dbl> <dbl>
 1       1   57.5     9     1     78    5.7
 2       2  270      19.7  20.7    9    2.53
 3       3  180      22     9    367    2.5
 4       4  420      15    39      7    2
 5       5  119.     18     3.5   21.5  0.825
 6       6  173      24.2   7.6   11.8  3
 7       7  110      23     1     98    2.6
 8       8  200      17.7  12.7    8.67 1.6
 9       9  138.     16.5   7    158    1.25
10      10  350      19.4  29.4    9    2.52
```

The output above gives the mean values of the variables “energy”, “protein”, “fat”, “calcium” and “iron” for all the observations in each cluster.

How can the quality of clustering be assessed?

The quality of clustering can be assessed using the “silhouette width” method and the “Dunn index” method.

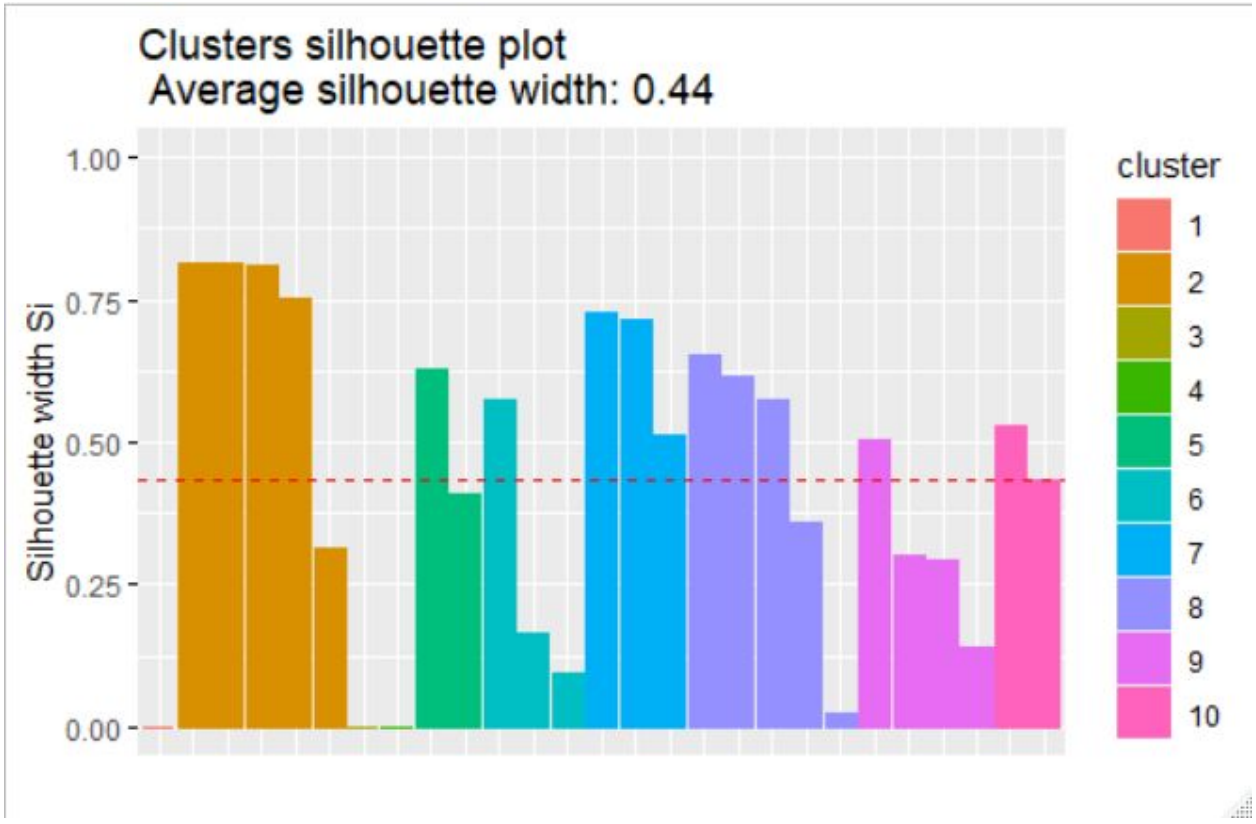
1. The silhouette width method

```
 # silhouette width

set.seed(20)
install.packages("cluster")
library(cluster)
install.packages("factoextra")
library(factoextra)

sil1<- silhouette(c10$cluster, dist(nutrients[,2:6]))
rownames(sil1) <- nutrients[,1]
head(sil1[, 1:3], 10)
```

```
> head(sil1[, 1:3], 10)
                  cluster neighbor  sil_width
Braised beef            2        6 0.81562583
Hamburger               6        7 0.16629406
Roast beef              3        2 0.00000000
Beefsteak               2        3 0.31325432
Canned beef             8        7 0.36005024
Broiled chicken         9        8 0.50255137
Canned chicken          8        7 0.65413800
Beef heart              8        7 0.57434017
Roast lamb leg          6        7 0.57568112
Roast lamb shoulder     6        2 0.09327443
```

```
fviz_silhouette(sil1)
```

```
> fviz_silhouette(sil1)
   cluster size ave.sil.width
1        1    1          0.00
2        2    5          0.70
3        3    1          0.00
4        4    1          0.00
5        5    2          0.52
6        6    3          0.28
7        7    3          0.65
8        8    5          0.45
9        9    4          0.31
10      10    2          0.48
```



The following code is used to determine if there are any negative silhouette widths and to list them out.

```
negindex <- which(sil1[, "sil_width"] < 0)
sil1[negindex, , drop = FALSE]
```

```
> sil1[negindex, , drop = FALSE]
     cluster neighbor sil_width
```

2. The Dunn Index Method

A higher Dunn index means that the data is clustered well or that each observation fits perfectly in its cluster.

#Dunn index

#At first, k-means clustering is done using using the factoextra:eclust() function

install.packages("factoextra")

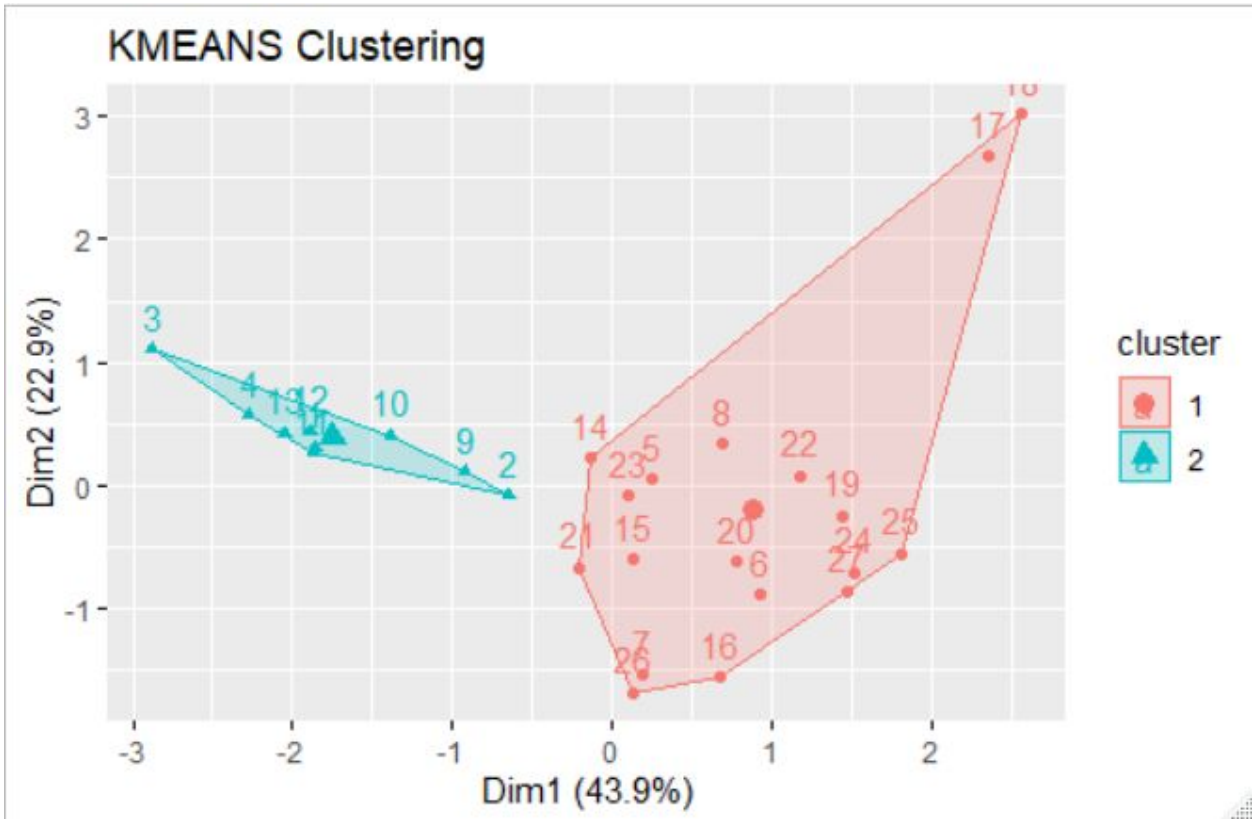
library(factoextra)

km1 <- eclust(nutrients[,2:6], "kmeans", nstart = 25)

km1

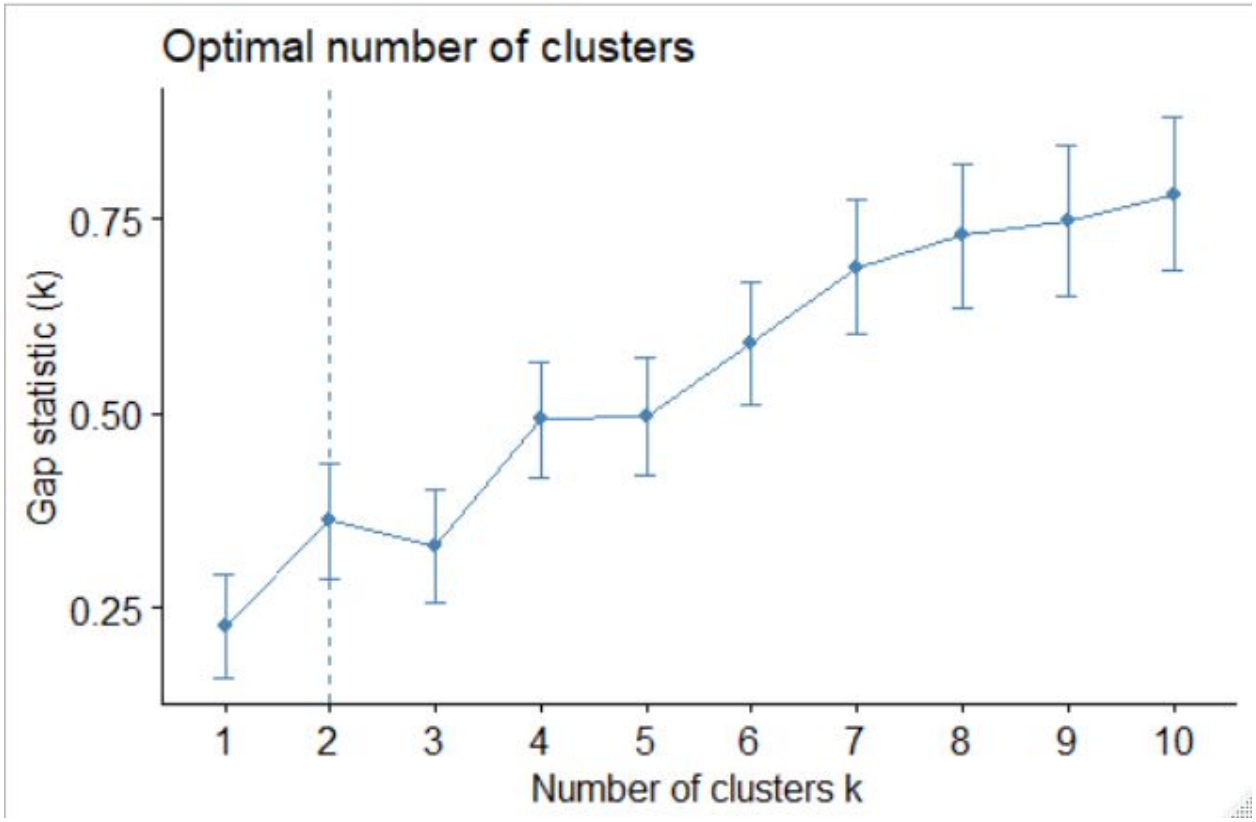
```
> km1 <- eclust(nutrients[,2:6], "kmeans", nstart = 25)
Clustering k = 1,2,..., K.max (= 10): .. done
Bootstrapping, b = 1,2,..., B (= 100)  [one "." per sample]:
.................................................. 50
.................................................. 100
> km1
K-means clustering with 2 clusters of sizes 18, 9

Cluster means:
    energy protein       fat   calcium     iron
1 145.5556      19  6.444444 61.555556 2.338889
2 331.1111      19 27.555556  8.777778 2.466667

Clustering vector:
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
 2  2  2  2  1  1  1  1  2  2  2  2  2  1  1  1  1  1  1  1
21 22 23 24 25 26 27
 1  1  1  1  1  1  1
```

```
Within cluster sum of squares by cluster:
[1] 178738.40  23751.03
 (between_SS / total_SS =  52.7 %)

Available components:

 [1] "cluster"      "centers"      "totss"
 [4] "withinss"     "tot.withinss" "betweenss"
 [7] "size"         "iter"         "ifault"
[10] "clust_plot"   "silinfo"      "nbclust"
[13] "data"         "gap_stat"
```



Gap statistic plot

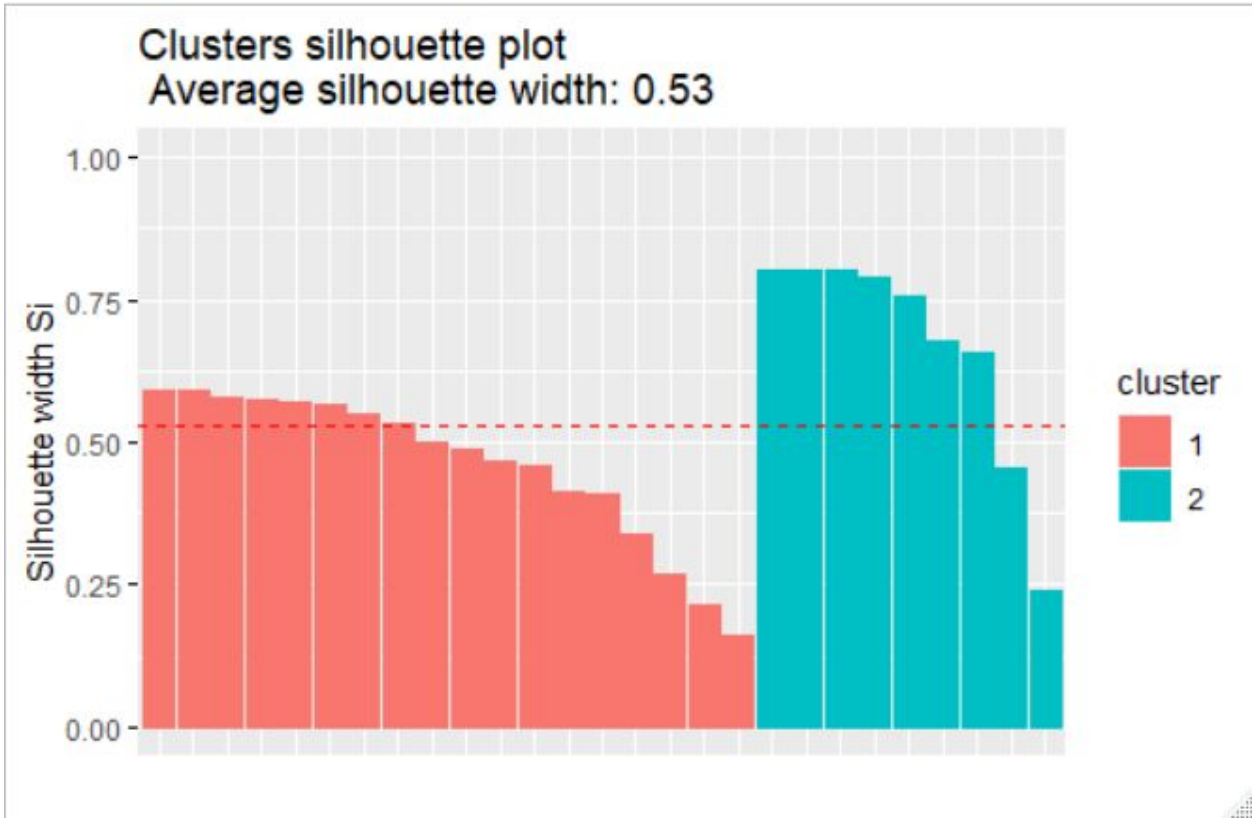
fviz_gap_stat(km1$gap_stat)



Silhouette plot

fviz_silhouette(km1)

```
> fviz_silhouette(km1)
  cluster size ave.sil.width
1       1   18          0.46
2       2    9          0.66
```



install.packages("fpc")

library(fpc)

stats1 <- cluster.stats(dist(nutrients[,2:6]), km1$cluster)

stats1$dunn

```
> stats1$dunn
[1] 0.1103735
```

Hierarchical Clustering

Hierarchical clustering is an unsupervised learning algorithm that identifies groups in a dataset.

Advantages of Hierarchical Clustering over K-means clustering:

The number of clusters to be generated does not have to be specified.
It produces a tree-based dendrogram.

Hierarchical Clustering Algorithms :

There are two main types: agglomerative and divisive.

Agglomerative clustering / AGNES (Agglomerative Nesting): This is better at identifying small clusters. This can be done with the “hclust” and “agnes” functions.

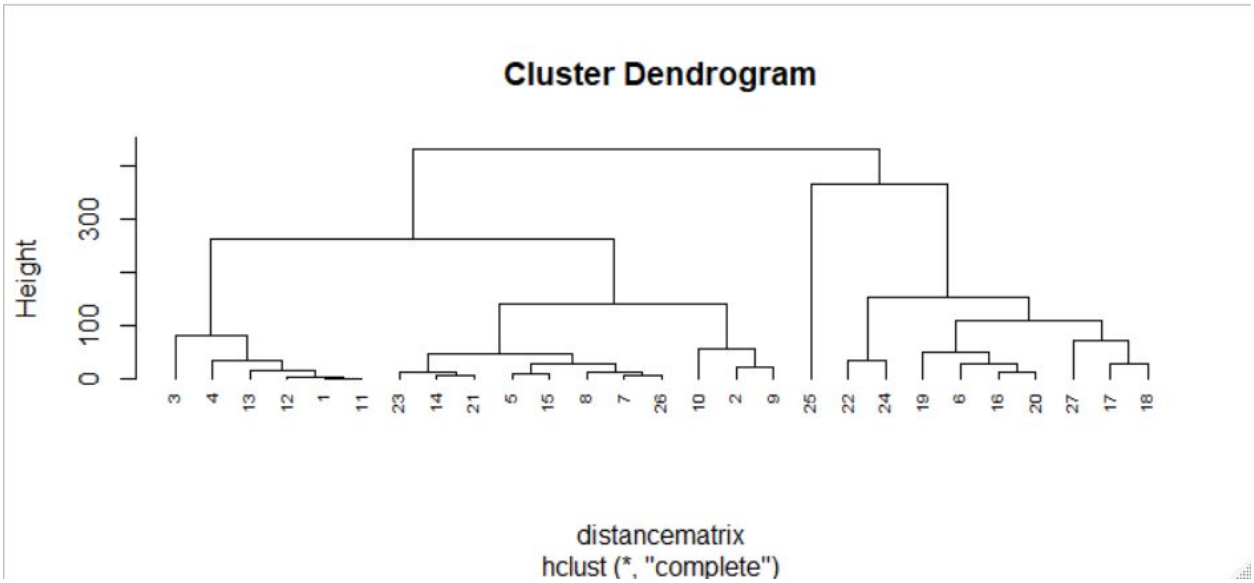
Divisive hierarchical clustering / DIANA (Divise Analysis) : This is better at identifying large clusters. This can be done with the “diana” function.

Agglomerative Hierarchical Clustering with the “hclust” function :

```
set.seed(20)

install.packages("tidyverse")

library(tidyverse)

install.packages("cluster")

library(cluster)

install.packages("factoextra")

library(factoextra)

install.packages("cluster.datasets")

library(cluster.datasets)

data(nutrients.meat.fish.fowl.1959)

# remove missing values

nutrients2 <- na.omit(nutrients.meat.fish.fowl.1959)

head(nutrients2)

distancematrix <- dist(nutrients2[,2:6], method = "euclidean")

# Hierarchical clustering

hierclus <- hclust(distancematrix , method = "complete" )
```

Plot the dendrogram

plot(hierclus, cex = 0.6, hang = -1)



Agglomerative Hierarchical Clustering with the “agnes” function :

If the “agnes” function is used, it also gives the “agglomerative coefficient” which measures the strength of the clustering structure.

```
 #  the agnes function with method = "complete"
 hierclusagnes <- agnes(nutrients2, method = "complete")

 # Agglomerative coefficient
  hierclusagnes$ac
[1] 0.909685
```

How do we find which hierarchical clustering method for “agnes” results in the strongest clustering structure?

```
methodstoassess <- c( "average", "single", "complete", "ward")
names(methodstoassess) <- c( "average", "single", "complete", "ward")
f <- function(x) {
  agnes(nutrients2, method = x)$ac
}
library(purrr)
```

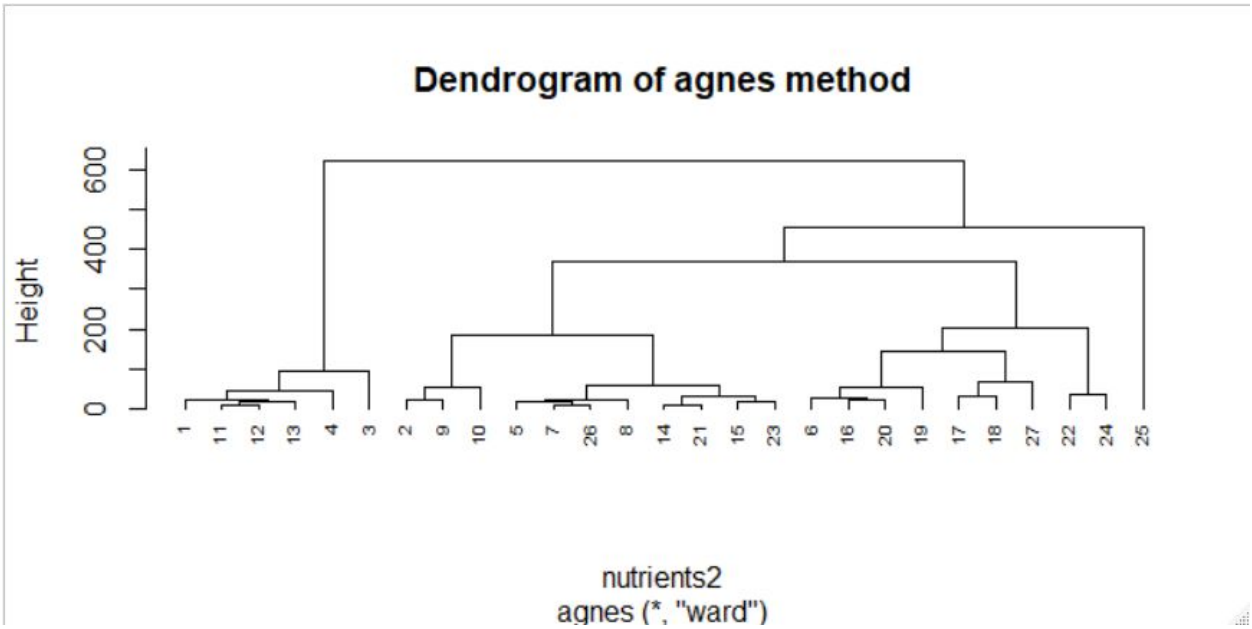
map_dbl(methodstoassess , f)

```
> map_dbl(methodstoassess, f)
  average    single  complete      ward
0.8956966 0.8645779 0.9096850 0.9316394
```

Thus we see that “Ward’s method” results in the strongest clustering structure.
visualize the dendrogram

hcward <- agnes(nutrients2, method = "ward")

pltree(hcward , cex = 0.6, hang = -1, main = "Dendrogram of agnes method")



Divisive Hierarchical Clustering

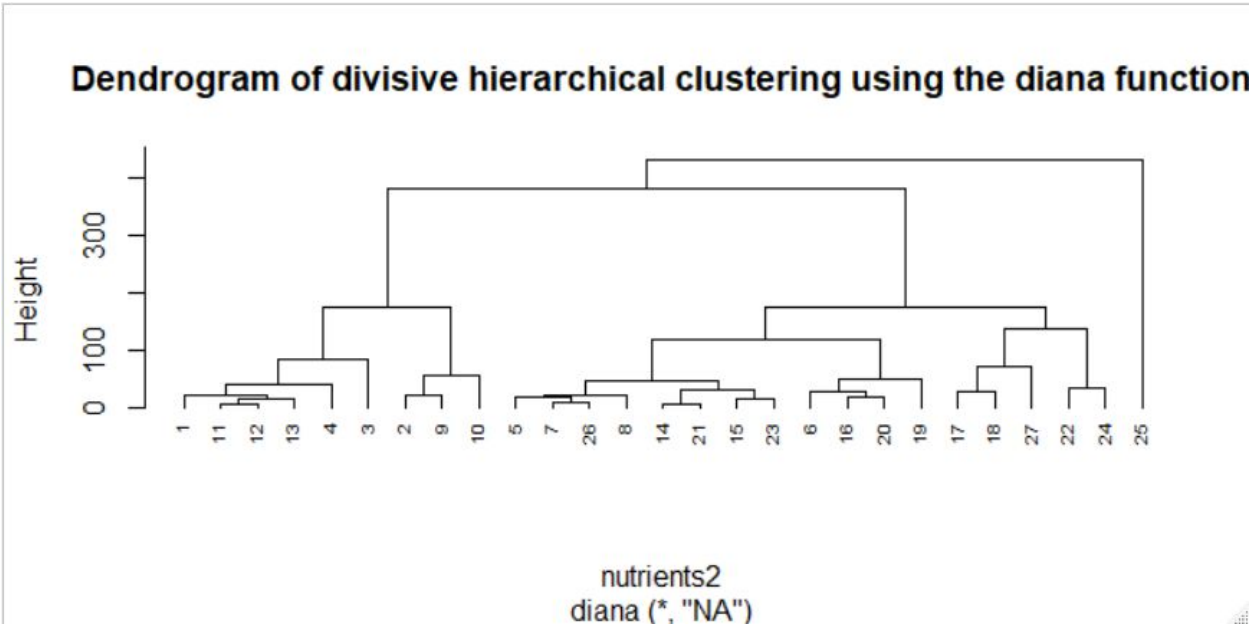
This is done with the “diana” function. This also gives the “divisive coefficient” which measures the strength of the clustering structure.

# Divisive Hierarchical Clustering
hierclusdivisive <- diana(nutrients2)

Divisive coefficient
> hierclusdivisive$dc

[1] 0.9038054

# visualize the dendrogram
pltree(hierclusdivisive, cex = 0.6, hang = -1, main = "Dendrogram of divisive hierarchical clustering using the diana function")



Cut hclust() tree into 4 groups

h6 <- hclust(distancematrix, method = "ward.D2")

tree1 <- cutree(h6, k = 4)

table(tree1)

```
> table(tree1)
tree1
 1  2  3  4
 6 11  9  1
```

Add the cluster of each observation to the data

```
library(dplyr)
nutrients2[,2:6] %>%
  mutate(cluster = tree1) %>%
  head
```

	energy	protein	fat	calcium	iron	cluster
1	340	20	28	9	2.6	1
2	245	21	17	9	2.7	2
3	420	15	39	7	2.0	1
4	375	19	32	9	2.6	1
5	180	22	10	17	3.7	2
6	115	20	3	8	1.4	3

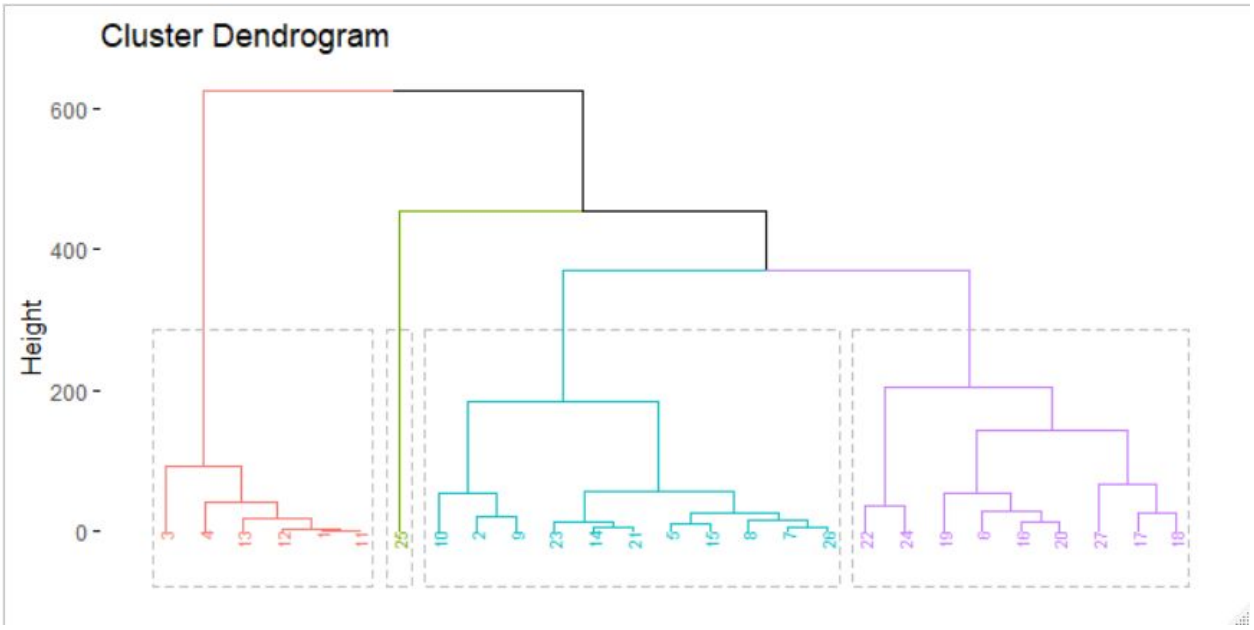
Add the cluster of each observation to the data

```
nutrients2%>%
  mutate(cluster = tree1) %>%
  head
```

	name	energy	protein	fat	calcium	iron	cluster
1	Braised beef	340	20	28	9	2.6	1
2	Hamburger	245	21	17	9	2.7	2
3	Roast beef	420	15	39	7	2.0	1
4	Beefsteak	375	19	32	9	2.6	1
5	Canned beef	180	22	10	17	3.7	2
6	Broiled chicken	115	20	3	8	1.4	3

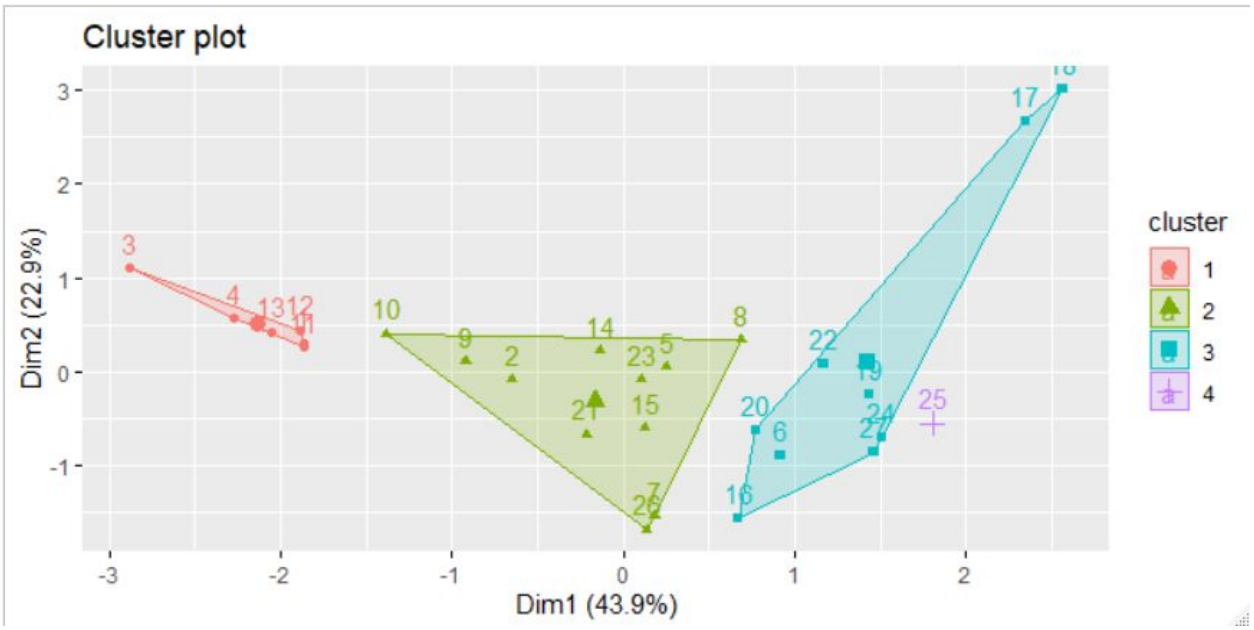
Draw borders around the clusters

```
fviz_dend(  h6, k = 4,rect = TRUE,cex = 0.5)
```



Visualize the clusters

```
nutrients3=nutrients2[,2:6]
fviz_cluster(list(data = nutrients3, cluster = tree1))
```




```
nutrients4=nutrients2%>%
  mutate(cluster = tree1)
count(nutrients4, cluster)
```

```
> count(nutrients4, cluster)
  cluster  n
1       1  6
2       2 11
3       3  9
4       4  1
```

 # Cut agnes() tree into 4 groups

```
hagnes <- agnes(nutrients2, method = "ward")
cutree(as.hclust(hagnes), k = 4)
```

 # Cut diana() tree into 4 groups

```
hdiana<- diana(nutrients2)
cutree(as.hclust(hdiana), k = 4)
```

How is the optimal number of clusters determined?

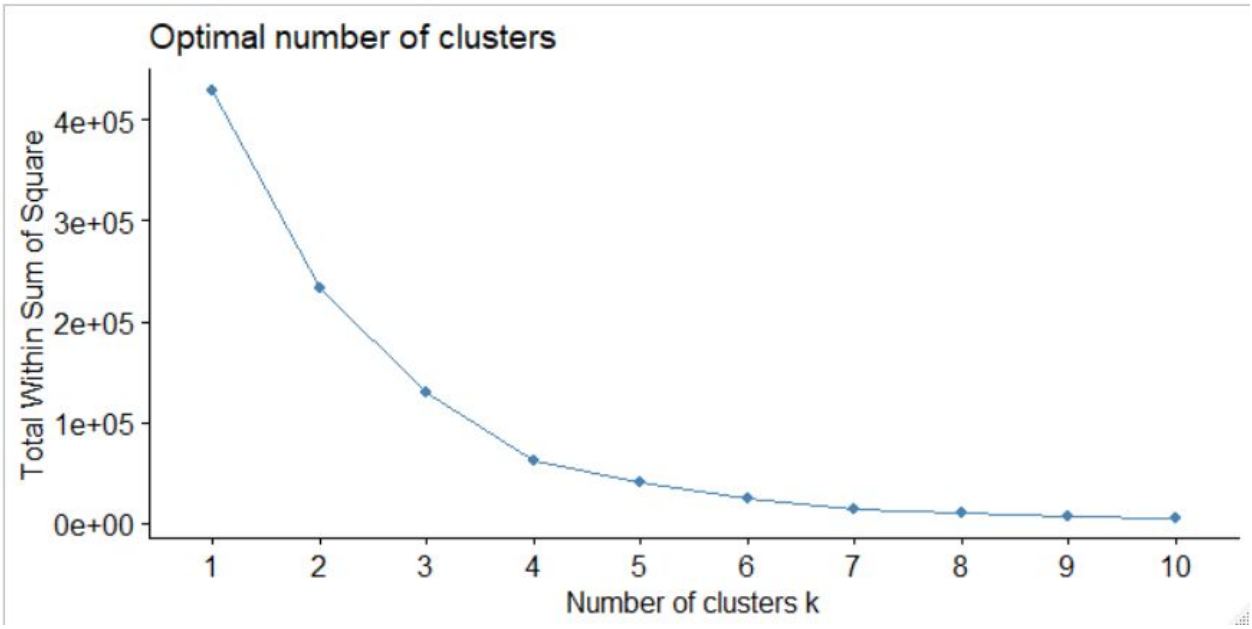
Here, we look at two methods:

- Elbow method
- Average Silhouette method

1. Elbow Method:

Elbow Method with the factoextra:fviz_nbclust function

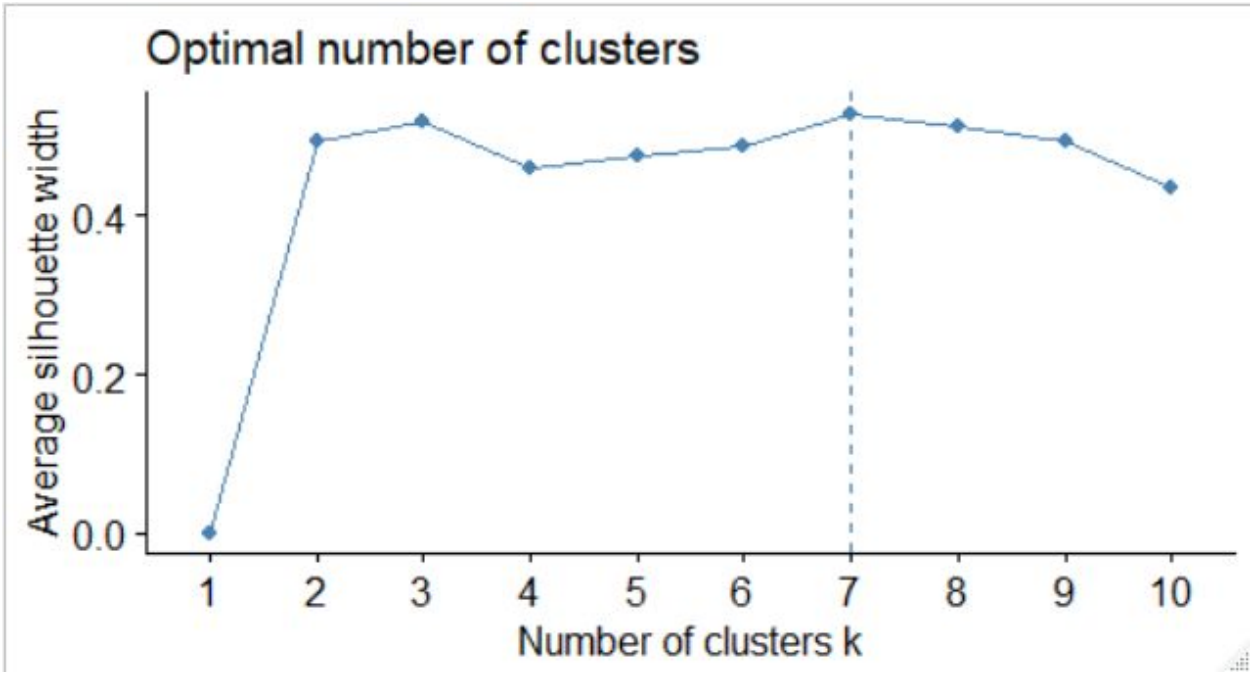
fviz_nbclust(nutrients2[,2:6], FUN = hcut, method = "wss")



2. Average Silhouette Method:

Average Silhouette Method with the cluster:fviz_nbclust function

fviz_nbclust(nutrients2[,2:6], FUN = hcut, method = "silhouette")




```
tree7 <- cutree(h6, k = 7)
table(tree7)
```

```
tree7
1 2 3 4 5 6 7
6 3 8 4 3 2 1
```

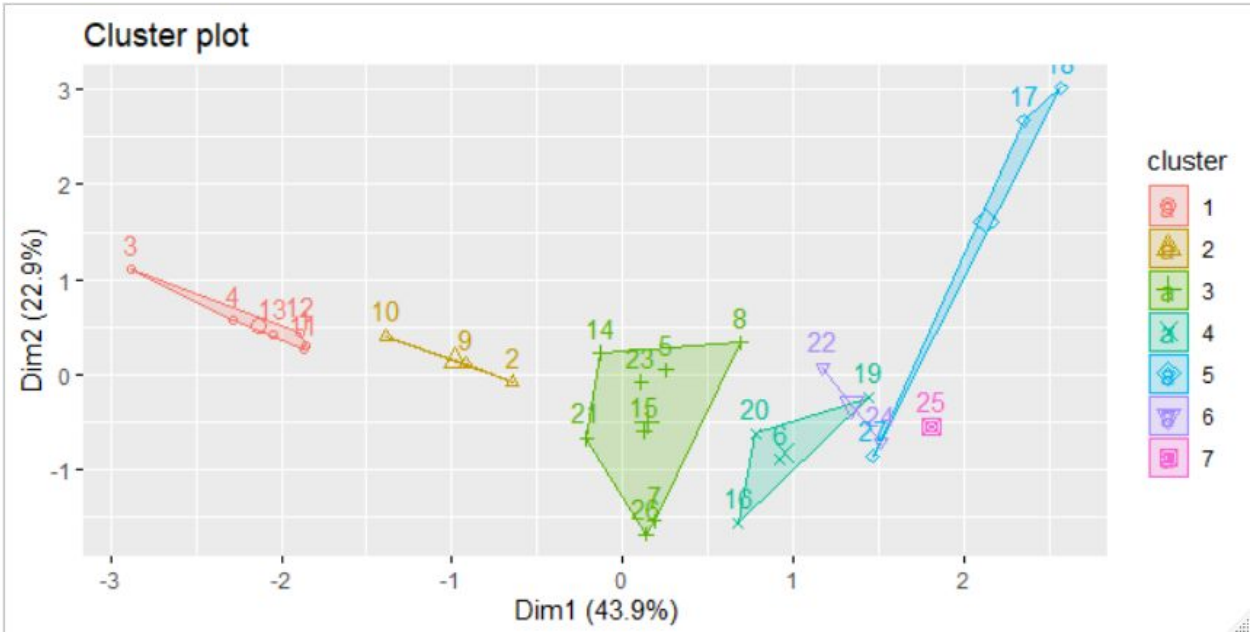
```
nutrients3 %>%
  mutate(cluster = tree7) %>%
  head
```

```
> nutrients3 %>%
+   mutate(cluster = tree7) %>%
+   head
  energy protein fat calcium iron cluster
1    340      20  28       9  2.6       1
2    245      21  17       9  2.7       2
3    420      15  39       7  2.0       1
4    375      19  32       9  2.6       1
5    180      22  10      17  3.7       3
6    115      20   3       8  1.4       4
```

```
nutrients2 %>%
 mutate(cluster = tree7) %>%
 head
```

	name	energy	protein	fat	calcium	iron	cluster
1	Braised beef	340	20	28	9	2.6	1
2	Hamburger	245	21	17	9	2.7	2
3	Roast beef	420	15	39	7	2.0	1
4	Beefsteak	375	19	32	9	2.6	1
5	Canned beef	180	22	10	17	3.7	3
6	Broiled chicken	115	20	3	8	1.4	4

```
fviz_cluster(list(data = nutrients3, cluster = tree7))
```



How can the quality of clustering be assessed?

The quality of clustering can be assessed using the “silhouette width” method and the “Dunn index” method.

1. The silhouette width method

```
install.packages("fpc")
```

```
library(fpc)

distancematrix <- dist(nutrients2[,2:6], method = "euclidean")

enhier <- eclust(nutrients2, "hclust", k = 7,

                 method = "complete", graph = FALSE)

head(enhier$cluster, 15)
```

```
> head(enhier$cluster, 15)
 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
 1  2  1  1  3  4  3  3  2  2  1  1  1  3  3
```

```
hier1 <- cluster.stats(distancematrix,  enhier$cluster)

# within clusters sum of squares

hier1$within.cluster.ss
```

```
> hier1$within.cluster.ss

[1] 13976.1
```

```
# cluster average silhouette widths

hier1$clus.avg.silwidths
```

```
> hier1$clus.avg.silwidths
        1         2         3         4         5         6         7
0.6276384 0.4409888 0.6325980 0.4107376 0.3906555 0.6407663 0.0000000
```

2. The Dunn Index Method

```
install.packages("clValid")

library("clValid")

# hierarchical clustering

distancematrix <- dist(nutrients2[,2:6], method = "euclidean")

clusterObj <- hclust(distancematrix, method="average")
```

```
nc <- 7 # number of clusters

cluster <- cutree(clusterObj,nc)

dunn(distancematrix, cluster)
```

```
> dunn(distancematrix, cluster)
[1] 0.3352845
```