# Clustering Assignment

Blessing Ekereke

3/10/2020

# Food

## Assignment Deliverables

*1. Perform an analysis of the data in order to group the food items into a few interpretable clusters. Assess the robustness of the cluster solution, and discuss your approach for it.*

*2. How many clusters is it optimal to retain? What do the clusters represent? Interpret their meaning.*

*3. Describe your general findings from the cluster analysis and interpret the software output.*

*4. Do you have any other thoughts about further analysis of this data?*

### Required packages

```
library(tidyverse)  # data manipulation
```

```
## Warning: package 'tidyverse' was built under R version 3.6.1
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.2.1 --
```

```
## v ggplot2 3.2.0     v purrr   0.3.2
## v tibble  2.1.1     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## Warning: package 'tidyr' was built under R version 3.6.2
```

```
## Warning: package 'readr' was built under R version 3.6.1
```

```
## Warning: package 'dplyr' was built under R version 3.6.1
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(cluster)    # clustering algorithms
library(factoextra) # clustering visualization
```

```
## Warning: package 'factoextra' was built under R version 3.6.3
```

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

```
library(dendextend) # for comparing two dendrograms
```

```
##
## ---------------------
## Welcome to dendextend version 1.12.0
## Type citation('dendextend') for how to cite the package.
##
## Type browseVignettes(package = 'dendextend') for the package vignette.
## The github page is: https://github.com/talgalili/dendextend/
##
## Suggestions and bug-reports can be submitted at: https://github.com/talgalili/dendextend/issues
## Or contact: <tal.galili@gmail.com>
##
##  To suppress this message use:  suppressPackageStartupMessages(library(dendextend))
## ---------------------
```

```
##
## Attaching package: 'dendextend'
```

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

## Reading in the Data

We shall split the data into two sets, the calibration set which would be used to perform the first phase of the analysis and the validation set which would be used to validate the results.

```
Data <- readxl::read_xlsx('food.xlsx')
Data <- Data[-28,]
set.seed(1)
train <- sample(1:nrow(Data), nrow(Data)/2)
calibration <-Data[-train,] # calibration set containing 14 observations
row.names(calibration) <- calibration$`Food Item`
```

```
## Warning: Setting row names on a tibble is deprecated.
```

```
library(factoextra)
validation <- Data[train,] # Validation set containing 13 observations
str(calibration)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    14 obs. of  6 variables:
##  $ Food Item: chr  "Roast Beef" "Canned beef" "Beef heart" "Roast lamb leg" ...
##  $ Calories : num  420 180 160 265 340 355 185 135 70 135 ...
##  $ Protein  : num  15 22 26 20 19 19 23 22 11 16 ...
##  $ Fat      : num  39 10 5 20 29 30 9 4 1 5 ...
##  $ Calcium  : num  7 17 14 9 9 9 9 25 82 15 ...
##  $ Iron     : num  2 3.7 5.9 2.6 2.5 2.4 2.7 0.6 6 0.5 ...
```

```
psych::describe(calibration)
```

```
## Warning in psych::describe(calibration): NAs introduced by coercion
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to min; returning
## Inf
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf
```

| | vars <int> | n <dbl> | mean <dbl> | sd <dbl> | median <dbl> | trimmed <dbl> | mad <dbl> | min <dbl> | max <dbl> |
|---|---|---|---|---|---|---|---|---|---|
| Food Item* | 1 | 14 | NaN | *NA* | *NA* | NaN | *NA* | Inf | -Inf |
| Calories | 2 | 14 | 205.357143 | 101.705916 | 175.00 | 198.750000 | 59.30400 | 70.0 | 420 |
| Protein | 3 | 14 | 19.500000 | 4.292211 | 19.50 | 19.666667 | 5.18910 | 11.0 | 26 |
| Fat | 4 | 14 | 12.857143 | 11.921907 | 9.00 | 11.666667 | 6.67170 | 1.0 | 39 |
| Calcium | 5 | 14 | 33.714286 | 45.587496 | 14.00 | 25.666667 | 7.41300 | 7.0 | 157 |
| Iron | 6 | 14 | 2.557143 | 1.681444 | 2.45 | 2.441667 | 1.33434 | 0.5 | 6 |

6 rows | 1-10 of 14 columns

## Rescale(Standardize) the Variables

Prior to clustering data, we have to rescale (standardize) variables for comparability. Standardizing a dataset involves rescaling the distribution of variable values in such a way that the mean of the observed values becomes 0 and the standard deviation (also variance) becomes 1.

```
calibration[, 2:6] <- scale(calibration[, 2:6])
psych::describe(calibration)
```

```
## Warning in psych::describe(calibration): NAs introduced by coercion
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to min; returning
## Inf
```

```
## Warning in FUN(newX[, i], ...): no non-missing arguments to max; returning
## -Inf
```

| | vars <int> | n <dbl> | mean <dbl> | sd <dbl> | median <dbl> | trimmed <dbl> | mad <dbl> | min <dbl> | max <dbl> |
|---|---|---|---|---|---|---|---|---|---|
| Food Item* | 1 | 14 | NaN | *NA* | *NA* | NaN | *NA* | Inf | -Inf |
| Calories | 2 | 14 | -3.867504e-17 | 1 | -0.29847962 | -0.06496321 | 0.5830929 | -1.3308679 | 2.110426 |
| Protein | 3 | 14 | 6.938894e-18 | 1 | 0.00000000 | 0.03883002 | 1.2089573 | -1.9803313 | 1.514371 |
| Fat | 4 | 14 | -4.362752e-17 | 1 | -0.32353405 | -0.09985619 | 0.5596168 | -0.9945676 | 2.192842 |
| Calcium | 5 | 14 | -3.569348e-17 | 1 | -0.43244941 | -0.17653128 | 0.1626104 | -0.5860003 | 2.704376 |
| Iron | 6 | 14 | -1.119738e-16 | 1 | -0.06372075 | -0.06867681 | 0.7935681 | -1.2234385 | 2.047560 |

6 rows | 1-10 of 14 columns

now we have a standard deviation of 1 and a mean value which is effectively 0 for every column.

## Execute Cluster Analysis

After preparing the data, the next phase is hierarchical clustering, this involves the following steps:

*1. Compute the distance matrix*

*2. Choose a clustering (linkage) method*

*3. Determine the number of clusters*

*4. Interpret the results*

*5. Repeat if necessary with different clustering methods and numbers of clusters*

## Computing the distance Matrix

A distance matrix shows the distances between pairs of objects. By definition, an object's distance from itself, which is shown in the main diagonal of the table, is 0. The more dissimilar the objects are, the larger the computed distance. Distance matrices are also called dissimilarity matrices.

```
dist_mat <- dist(calibration[,2:6], method= 'euclidean')

as.matrix(dist_mat)[1:10, 1:10]# visualizing the distance among the first 10 observations.
```

```
##           1         2        3        4         5         6         7
## 1  0.000000 3.9006993 5.161359 2.519668 1.5103338 1.3803375 3.9140880
## 2  3.900699 0.0000000 1.673104 1.441503 2.4583285 2.6252584 0.6694984
## 3  5.161359 1.6731044 0.000000 2.909770 3.7343418 3.8830273 2.0725420
## 4  2.519668 1.4415027 2.909770 0.000000 1.0823579 1.2470186 1.4007453
## 5  1.510334 2.4583285 3.734342 1.082358 0.0000000 0.1797895 2.4534656
## 6  1.380338 2.6252584 3.883027 1.247019 0.1797895 0.0000000 2.6070916
## 7  3.914088 0.6694984 2.072542 1.400745 2.4534656 2.6070916 0.0000000
## 8  4.469891 1.9694935 3.305994 2.278164 3.2169221 3.3455693 1.4679431
## 9  5.588954 3.4945038 3.916260 4.155184 4.7889926 4.9387671 3.9951329
## 10 4.106884 2.4391685 3.975271 2.379587 3.1679522 3.2953030 2.1778877
##           8        9       10
## 1  4.469891 5.588954 4.106884
## 2  1.969494 3.494504 2.439169
## 3  3.305994 3.916260 3.975271
## 4  2.278164 4.155184 2.379587
## 5  3.216922 4.788993 3.167952
## 6  3.345569 4.938767 3.295303
## 7  1.467943 3.995133 2.177888
## 8  0.000000 4.349353 1.418718
## 9  4.349353 0.000000 3.838941
## 10 1.418718 3.838941 0.000000
```

The euclidean distance (named after Euclid) is the straight line distance between two points, and it is generally the default choice for cluster analysis. Looking at the first 10 observations, we see, for example, that observations 9 and 1 are the most dissimilar and observations 5 and 6 are

the least disimilar. These distances will be used in this analysis to group similar observations together.

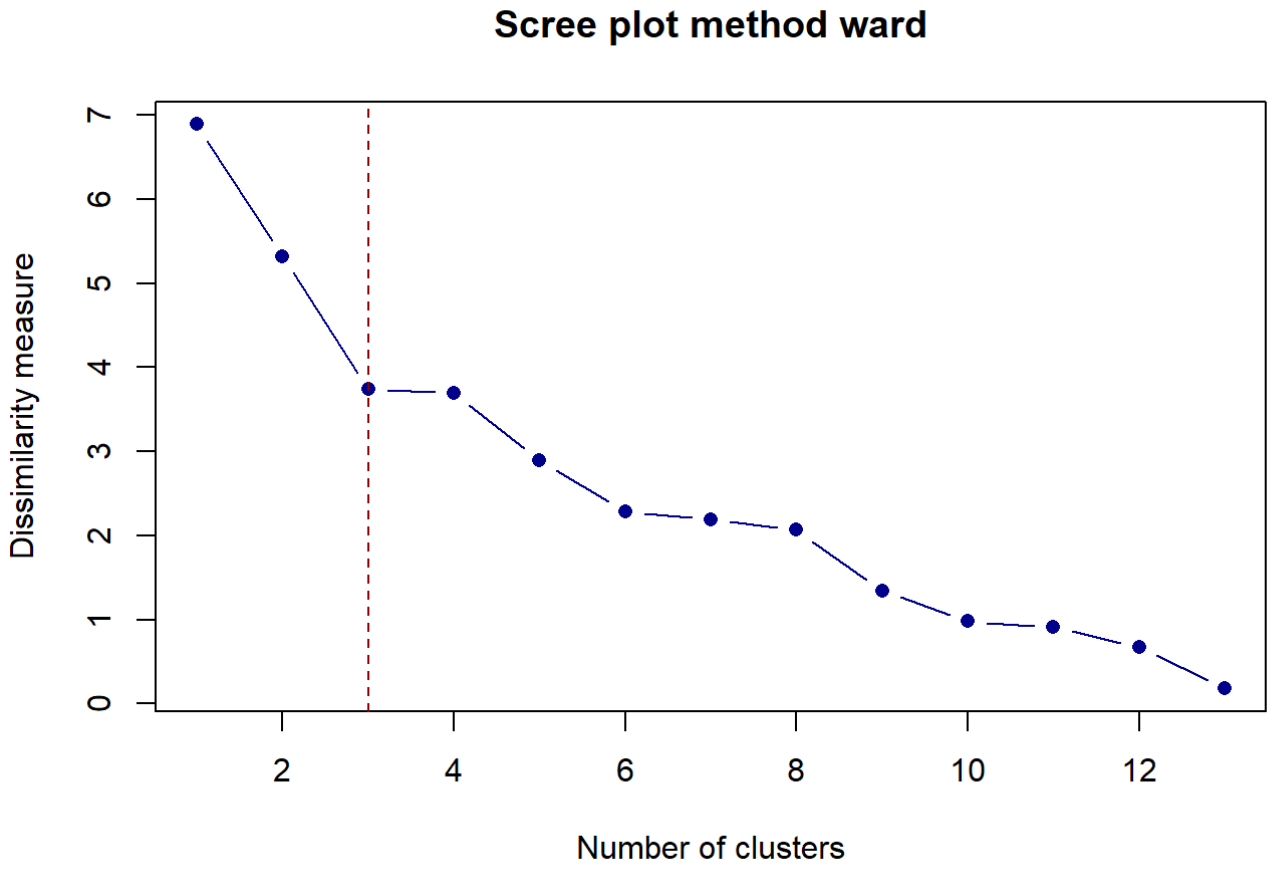
## Choose Clustering Method

We shall work with the Ward's method

```
food_ward <- hclust(dist_mat, method = 'ward.D2')
```

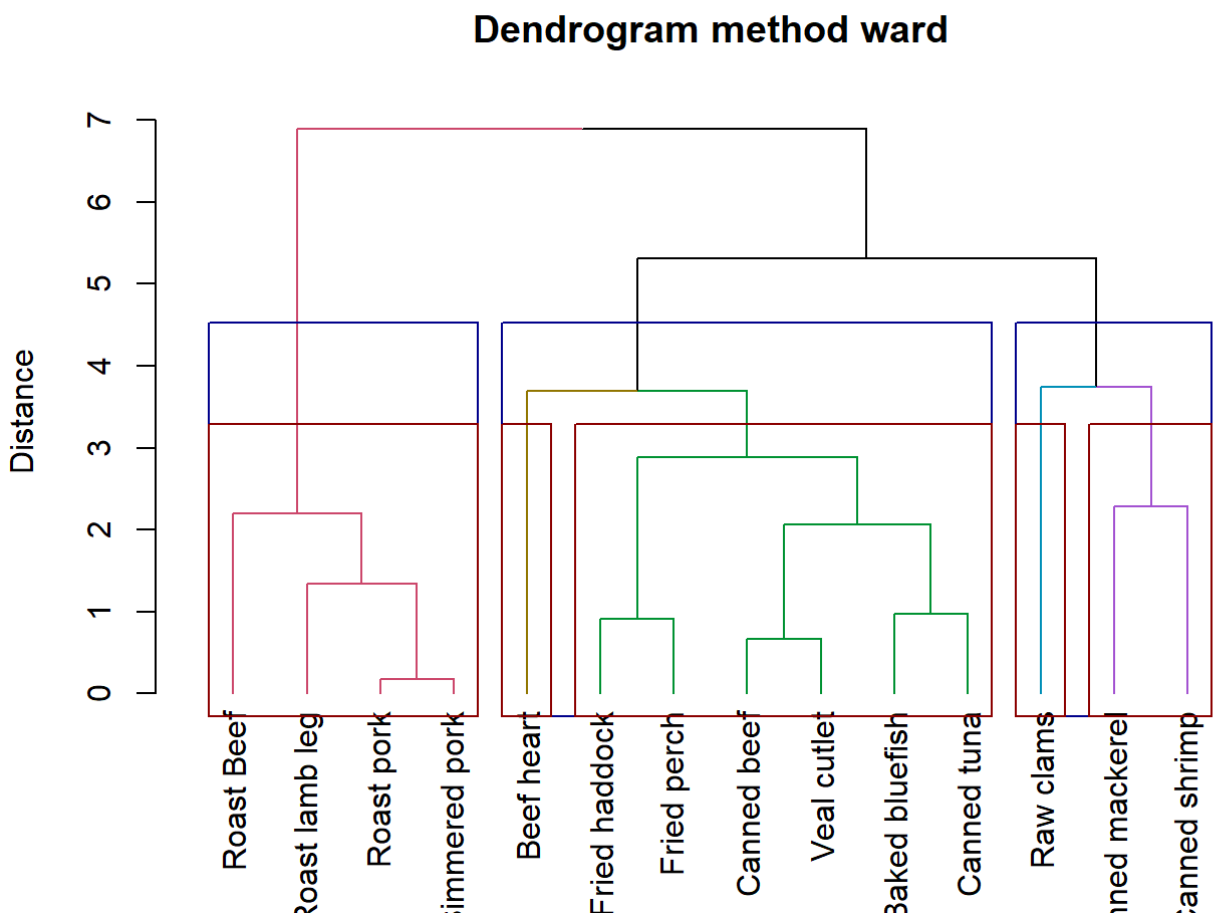
**In order to get an indication of the appropriate number of clusters, we next examine a scree plot and a dendrogram.**

```
# Scree plot
plot(rev(food_ward$height), # rev is used to plot from low to high values on Y axis
     type = "b",           # to display both the points and lines
     ylab = "Dissimilarity measure",
     xlab = "Number of clusters",
     main = "Scree plot method ward",
     col = "darkblue",
     pch = 16)             # specify the plot symbol: 16 = filled circle
abline(v = 3, lty = 2, col = "darkred") # draw a vertical line at v = 5
```




```
# Dendrogram
library(dendextend)
food_ward$labels <- calibration$`Food Item`
plot(set(as.dendrogram(food_ward),
         "branches_k_color", # to highlight the cluster solution with a color
         k = 5),
     ylab = "Distance",
     main = "Dendrogram method ward",
     cex = 0.2)             # Size of labels
```

```
rect.hclust(food_ward, k = 3, border = "darkblue")  # draw red borders around 2 clusters
rect.hclust(food_ward, k = 5, border = "darkred")  # draw red borders around 5 clusters
```




Based on the scree plot, we can choose a 3-cluster solution.Observing the dendrogram, the best choice of the no. clusters is the no. of vertical lines in the dendrogram cut by a horizontal line that transverse the maximum distance vertically without intersecting a cluster. In this case it is either 3 clusters or 2 clusters.

In order to get a second opinion, let try out another method, the complete method.

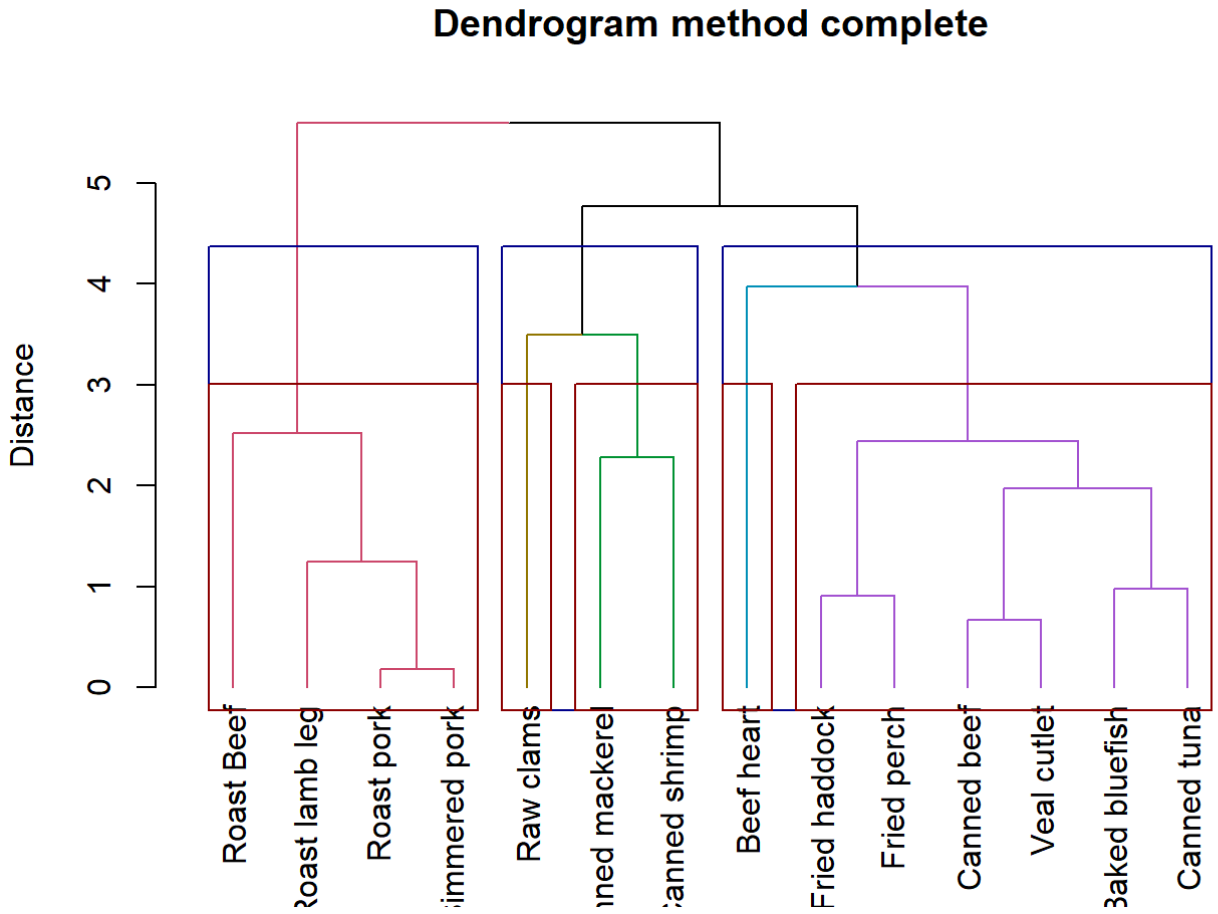
```
food_comp <- hclust(dist_mat, method = 'complete')
```

```
plot(rev(food_comp$height), # rev is used to plot from low to high values on Y axis
     type = "b",           # to display both the points and lines
     ylab = "Dissimilarity measure",
     xlab = "Number of clusters",
     main = "Scree plot method complete",
     col = "darkblue",
     pch = 16)             # specify the plot symbol: 16 = filled circle
abline(v = 5, lty = 2, col = "darkred")
abline(v = 3, lty = 2, col = "darkred")# draw a vertical line at v = 5
```


on the scree plot, we notice an elbow at cluster no.3 and no. 5, we can choose a 3-cluster solution or a five-cluster solution. Based on the findings from the two methods, We shall proceed with the 3-cluster solution.

```
# Dendrogram
library(dendextend)
food_comp$labels <- calibration$`Food Item`
plot(set(as.dendrogram(food_comp),
         "branches_k_color", # to highlight the cluster solution with a color
         k = 5),
     ylab = "Distance",
```

```
     main = "Dendrogram method complete",
     cex = 0.2)             # Size of labels
rect.hclust(food_comp, k = 3, border = "darkblue")  # draw red borders around 2 clusters
rect.hclust(food_comp, k = 5, border = "darkred")  # draw red borders around 5 clusters
```




Using the following commands, we find the sizes of the four clusters:

```
food_ward.cut <- cutree(food_ward, 3)# for ward's method
table(food_ward.cut)
```

```
## food_ward.cut
## 1 2 3
## 4 7 3
```

```
food_comp.cut <- cutree(food_comp, 3)# for complete method
table(food_comp.cut)
```

```
## food_comp.cut
## 1 2 3
## 4 7 3
```

## K-means cluster analysis

As a second stage of the analysis, we proceed by using a partitioning approach. We begin by defining the initial cluster centroids to be submitted to the K-means cluster analysis. The following code is an iterative loop where each row represents one cluster. Every cluster is linked to the previous cluster using the function rbind(). This creates a matrix, where the columns represent the average value of each variable in the respective cluster.

```
cent <- NULL
for(k in 1:3){
  cent <- rbind(cent, colMeans(calibration[,2:6][food_ward.cut == k, , drop = FALSE]))
}
```

The output is given in the following table which represents the average nutritional levels for each of the 5 nutrients in each cluster:

```
round(cent, 3)
```

```
##      Calories Protein    Fat Calcium   Iron
## [1,]    1.373  -0.291  1.396  -0.553 -0.108
## [2,]   -0.390   0.449 -0.467  -0.423 -0.170
## [3,]   -0.921  -0.660 -0.771   1.725  0.541
```

The second stage of the analysis is running the K-means clustering method. Recall that K-means needs two inputs: the number of clusters K and an initial choice (seeds) for the cluster centers - these will then be updated as the algorithm progresses. We shall use the results from the earlier

hierarchical clustering as inputs into K-means. Thus, we are looking for K = 3 clusters, and we shall take as initial cluster centers the values computed in the table above (the centroids of the clusters identified by Ward's method).

```
set.seed(1)
food_kmeans <- kmeans(calibration[, 2:6], centers = cent , iter.max = 10)
food_kmeans
```

```
## K-means clustering with 3 clusters of sizes 4, 8, 2
##
## Cluster means:
##     Calories    Protein        Fat    Calcium       Iron
## 1  1.3730062 -0.2912252  1.3959895 -0.5530965 -0.1083253
## 2 -0.4582540  0.4950828 -0.5332320 -0.1938972 -0.1454957
## 3 -0.9129965 -1.3978809 -0.6590508  1.8817817  0.7986334
##
## Clustering vector:
##  [1] 1 2 2 1 1 1 2 2 3 2 3 2 2 2
##
## Within cluster sum of squares by cluster:
## [1]  3.323511 17.817637  5.725831
##  (between_SS / total_SS =  58.7 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

```
food_kmeans$size
```

```
## [1] 4 8 2
```

```
food_kmeans$centers
```

```
##     Calories    Protein        Fat    Calcium       Iron
## 1  1.3730062 -0.2912252  1.3959895 -0.5530965 -0.1083253
## 2 -0.4582540  0.4950828 -0.5332320 -0.1938972 -0.1454957
## 3 -0.9129965 -1.3978809 -0.6590508  1.8817817  0.7986334
```

We now can calculate the change in each cluster from the K-means clustering - this will be the difference between the original cluster centers (recall that these were earlier computed with the Ward's method and saved in cent and the cluster centers of the final solution:

```
change <- NULL
for (i in 1:3){
  change <- rbind(change, food_kmeans$centers[i,]-cent[i,])
}

round(change, 3)
```

```
##      Calories Protein    Fat Calcium  Iron
## [1,]    0.000   0.000  0.000   0.000 0.000
## [2,]   -0.068   0.046 -0.066   0.229 0.024
## [3,]    0.008  -0.738  0.112   0.157 0.258
```

The change in cluster centers is not always small, thus it was a good idea to run a second (partitioning) method following-up on the first (hierarchical) method in order to arrive at a more robust solution.

In order to gain more insights into the degree of separation between clusters, we let's generate the matrix of pairwise distances between cluster centers using the command:

```
dist(food_kmeans$centers)
```

```
##          1        2
## 2 2.797159
## 3 4.174313 3.000959
```

The table of pairwise distances suggests that clusters 3 and 1 are the furthest away from each other (in terms of nutrientional contents), while clusters 2 and 1 are the closest.

# Interpreting the results

For results interpretation, it is easier to rely on the original data set enriched with the information about the cluster assignment of each observation. Thus, we will store the clustering solution in our dataset, so that each row also records the cluster to which that particular observation has been assigned.

```
# Overall average in the sample
colMeans(calibration[, 2:6])
```

```
##      Calories       Protein           Fat       Calcium          Iron
## -3.865955e-17  6.938894e-18 -4.361590e-17 -3.568574e-17 -1.120136e-16
```

```
round(colMeans(calibration[, 2:6]),3)
```

```
## Calories  Protein      Fat  Calcium     Iron
##        0        0        0        0        0
```

```
calibration <- cbind(calibration, cluster= food_kmeans$cluster)
```

```
# Average for each cluster
aggregate(calibration[, 2:6],
          by = list(cluster = calibration$cluster),
          FUN = mean)
```

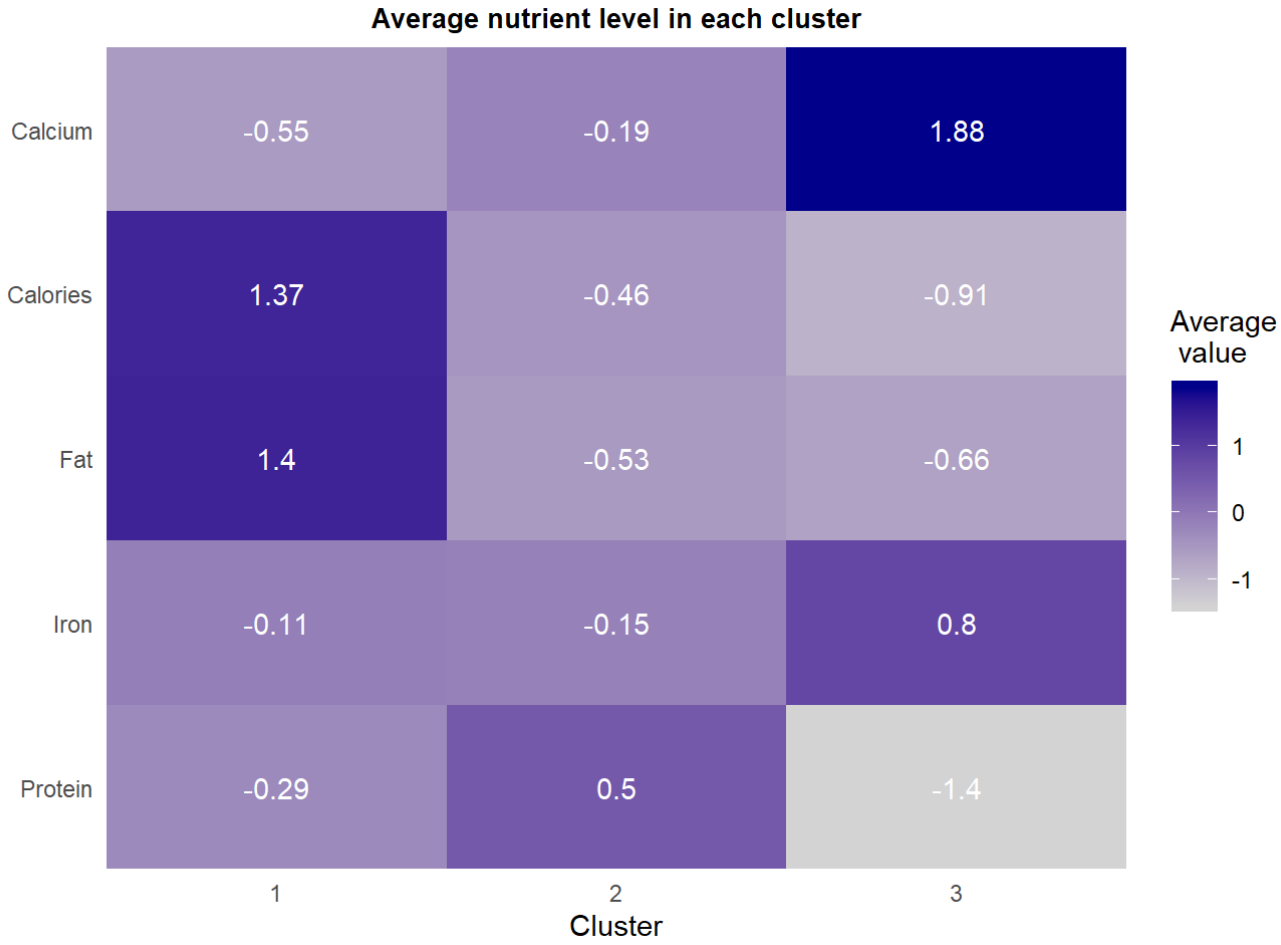
| cluster <int> | Calories <dbl> | Protein <dbl> | Fat <dbl> | Calcium <dbl> | Iron <dbl> |
|---|---|---|---|---|---|
| 1 | 1.3730062 | -0.2912252 | 1.3959895 | -0.5530965 | -0.1083253 |
| 2 | -0.4582540 | 0.4950828 | -0.5332320 | -0.1938972 | -0.1454957 |
| 3 | -0.9129965 | -1.3978809 | -0.6590508 | 1.8817817 | 0.7986334 |

3 rows

Note that the overall averages for our 5 variables are 0 because we standardized the values prior to the cluster analysis. The average value of any standardized variable in the cluster is interpreted as the deviation from the average pattern in the data. For example, the average value for calcium is 1.88 for cluster 3, thus this cluster has a higher than average preference or pattern for this nutrient by almost two standard deviations. On the other hand, cluster 2 shows the lowest value for Cacium, by less than 1/2 standard deviation compared to the sample average.

It is possible to proceed with the interpretation on the basis of the matrix giving cluster averages alone but, as often the case, it is easier to gain more insights if we visualize these values in a graphical form. To this end we can generate a heat map of the matrix using the following code

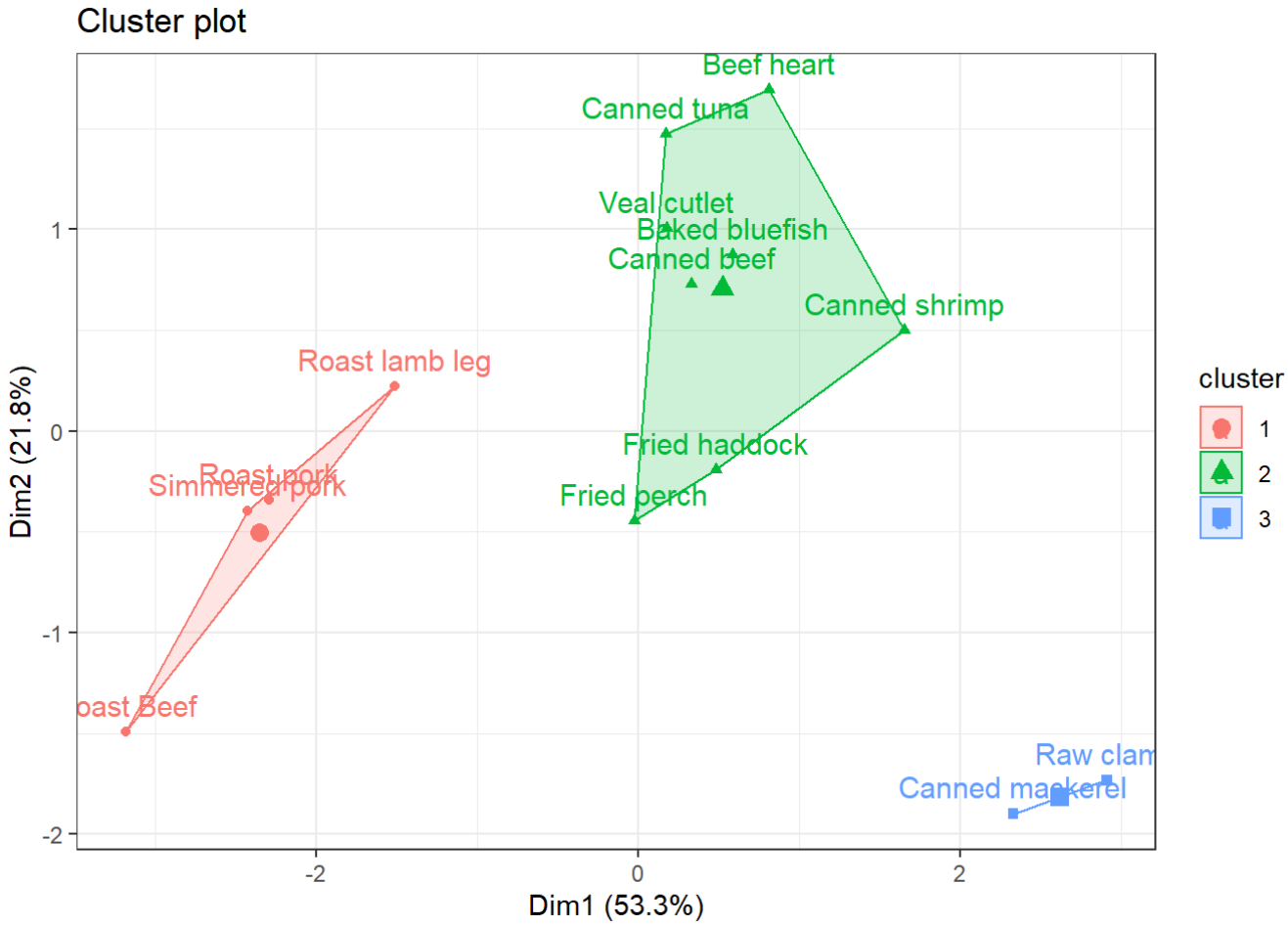
```
library(tidyverse)

dt.cluster = aggregate(calibration[, 2:6],
                       by = list(cluster = calibration$cluster),
                       FUN = mean)
dt.cluster %>%
  gather(carmake, value, -cluster) %>% # to transfrom from wide to long format
  mutate(carmake = fct_rev(factor(carmake))) %>% # to reverse the order of car makes' names on the plot
  ggplot(aes(x = factor(cluster), y = carmake)) +
  geom_tile(aes(fill = round(value, digits = 2))) +
  geom_text(aes(label = round(value, digits = 2)), color="white") +
  scale_x_discrete(expand = c(0,0)) +
  scale_y_discrete(expand = c(0,0)) +
  scale_fill_gradient("Average\n value", low = "lightgrey", high = "darkblue") +
  theme_minimal() +
  labs(title = "Average nutrient level in each cluster",
       x = "Cluster",
       y = " ") +
  theme(legend.position="right",
        plot.title = element_text(size = 10, face = "bold", hjust = 0.5),
        axis.ticks = element_blank())
```




From the average nutrients level we see that the cluster 1 evaluates highly the nutrients fats and calories, based on this evaluation we can call cluster 1 fatty and high calories food, Cluster 2 evaluates the nutrient protein highly in comparison to other clusters hence cluster 2 can be called protenious food, Cluster 3 in comparison to other cluster highly evaluates the nutrients iron and calcium hence Cluster 3 can be decribed as mineral food.

We can also visualize our results by using fviz_cluster(). If there are more than two dimensions (variables), fviz_cluster will perform principal component analysis (PCA) and plot the data points according to the first two principal components that explain the majority of the variance. In our case, we have 5 utrients variables, thus PCA is necessary. Such a plot allows us to visualize the relative separation between the clusters.

```
row.names(calibration) <- calibration$`Food Item`
#install.packages("factoextra")
#library(factoextra)
fviz_cluster(food_kmeans, data = calibration[,-1]) +
  theme_bw()
```




## Validation

The results can be validated by implementing steps 1-5. The data used for validation consists of the holdout sample of 13 observations, contained in the file validation.csv.

## STEPS FOR CLUSTER VALIDATION

*1. Split data into two random samples: calibration(calibration) and validation (validation).*

*2. Use clustering method to cluster calibration data. Determine the appropriate number of clusters and calculate centroids.*

*3. Using the cluster centroids from the calibration data, assign each observation from the validation sample to the closest centroid. We denote this cluster solution KS1*

*4. Use the same clustering method from step 2 to cluster the validation data. Choose the solution with the same number of clusters as determined in step 2. We denote this duster solution KS2.*

*5. Cluster solutions KS1 and KS2 represent different assignments of the same set of observations to clusters. To assess the agreement between the two solutions, we cross-tabulate KS1 versus KS2.*

We have already completed Step 1 and 2 earlier.

### Step 3

```
validation[,2:6] <- scale(validation[, 2:6])
```

Next, we assign each observation from the validation sample to the closest centroid (food_kmeans$centers) that we obtained from the clustering on the calibration sample . Note that this can be accomplished by running the K-means procedure with the centroids as initial cluster centers and with one single iteration - this effectively just assigns each data point in the validation sample to the closest centroid.

```
set.seed(1)
food_kmeans1 <- kmeans(validation[,2:6], centers = food_kmeans$centers, iter.max = 1)
```

```
## Warning: did not converge in 1 iteration
```

```
food_kmeans1
```

```
## K-means clustering with 3 clusters of sizes 6, 6, 1
##
```

```
## Cluster means:
##     Calories    Protein        Fat    Calcium       Iron
## 1  0.8706428  0.2021287  0.9001742 -0.4481796  0.2794633
## 2 -0.6087769  0.2407709 -0.6997448  0.4175486 -0.7175692
## 3 -1.5711951 -2.6573974 -1.2025764  0.1837859  2.6286357
##
## Clustering vector:
##  [1] 2 1 2 1 1 1 1 3 2 2 1 2 2
##
## Within cluster sum of squares by cluster:
## [1]  4.414689 16.691434  0.000000
##  (between_SS / total_SS =  64.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
## Warning: did *not* converge in specified number of iterations
```

The cluster assignment of each observation in the validation sample is saved in food_kmeans1$cluster (called “Clustering vector” in the above output). We store this solution as KS1 into data validation.

```
vali<- cbind(validation, KS1 = food_kmeans1$cluster)
vali[, c("Food Item", "KS1")]
```

| Food Item <chr> | KS1 <int> |
|---|---|
| Canned sardines | 2 |
| Beef steak | 1 |
| Canned chicken | 2 |
| Braised beef | 1 |

| Food Item <chr> | KS1 <int> |
|---|---|
| Hamburger | 1 |
| Smoked ham | 1 |
| Beef tongue | 1 |
| Canned clams | 3 |
| canned crabmeat | 2 |
| Canned salmon | 2 |

1-10 of 13 rows Previous **1** 2 Next

New centers from kmeans on validation sample using centroids from kmeans on calibration sample:

```
round(food_kmeans1$centers, 3)
```

```
##   Calories Protein    Fat Calcium   Iron
## 1    0.871   0.202  0.900  -0.448  0.279
## 2   -0.609   0.241 -0.700   0.418 -0.718
## 3   -1.571  -2.657 -1.203   0.184  2.629
```

### Step 4

using the same clustering method, number of clusters, and centroids cent as for car_pref1 to cluster the validation data directly.

```
food_kmeans2 <- kmeans(validation[,2:6], centers = cent, iter.max = 10)
food_kmeans2
```

```
## K-means clustering with 3 clusters of sizes 6, 6, 1
##
## Cluster means:
##     Calories    Protein        Fat    Calcium       Iron
```

```
## 1  0.8706428  0.2021287  0.9001742 -0.4481796  0.2794633
## 2 -0.6087769  0.2407709 -0.6997448  0.4175486 -0.7175692
## 3 -1.5711951 -2.6573974 -1.2025764  0.1837859  2.6286357
##
## Clustering vector:
##  [1] 2 1 2 1 1 1 1 3 2 2 1 2 2
##
## Within cluster sum of squares by cluster:
## [1]  4.414689 16.691434  0.000000
##  (between_SS / total_SS =  64.8 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "ifault"
```

The new cluster assignment of each observation in the validation sample is saved in food_kmeans2$cluster (called “Clustering vector” in the above output). We store this solution as KS2 into validation data.

```
vali <- cbind(vali, KS2 = food_kmeans2$cluster)
vali[, c("Food Item", "KS1", "KS2")]
```

| **Food Item**<br><chr> | **KS1**<br><int> | **KS2**<br><int> |
|---|---|---|
| Canned sardines | 2 | 2 |
| Beef steak | 1 | 1 |
| Canned chicken | 2 | 2 |
| Braised beef | 1 | 1 |
| Hamburger | 1 | 1 |
| Smoked ham | 1 | 1 |

| Food Item <chr> | KS1 <int> | KS2 <int> |
|---|---|---|
| Beef tongue | 1 | 1 |
| Canned clams | 3 | 3 |
| canned crabmeat | 2 | 2 |
| Canned salmon | 2 | 2 |

1-10 of 13 rows Previous 1 2 Next

### Step 5

Cluster solutions KS1 and KS2 represent different assignments of the same set of observations from the validation sample to clusters. To assess the agreement between the two solutions, we cross-tabulate KS1 versus KS2.

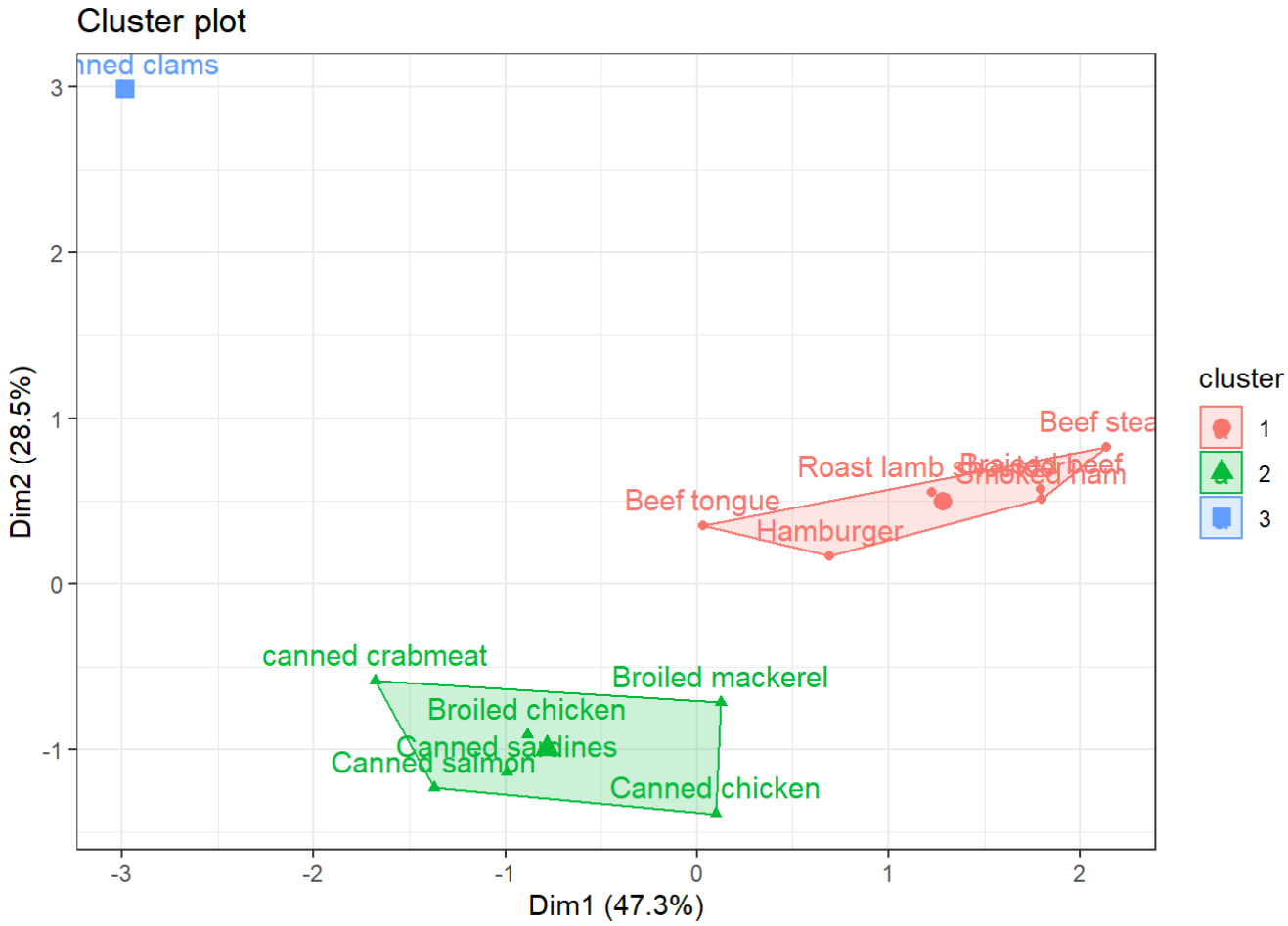
```
table(KS1 = vali$KS1, KS2 = vali$KS2)
```

```
##    KS2
## KS1 1 2 3
##   1 6 0 0
##   2 0 6 0
##   3 0 0 1
```

The confusion matrix makes it visually clear that the two solutions are in broad agreement, with all of the observations falling into just 3 table cells (the digonal cells), this strong agreement can be attributed to the fact that the calibration data set was too small. The resulting clusters may seem relatively robust but further analysis is required.

visualizing the clusters in the validation set.

```
validation <- as.data.frame(validation)
row.names(validation) <- validation$`Food Item`
#install.packages("factoextra")
#library(factoextra)
fviz_cluster(food_kmeans2, data = validation [,-1]) +  theme_bw()
```




Thoughts on further Analysis.

More data gives more accuracy. The data set contains just 27 food items which is quite small for this type of analysis. If a more accurate clustering result is intended, more data should be collected and compiled, preferably above 100 records and then the clustering algorithm should be reruned.