K-means Clustering

Short Intro to k-means

K-means clustering is an unsupervised machine learning algorithm. Unsupervised algorithms make inferences using only input data without referring to known, or labeled outcomes. The goal of the K-means is to group similar observations together and discover underlying patterns.

K-means algorithm has one input parameter - **k** - which refers to the number of groups (clusters) that the observations should be grouped into. The algorithm actually finds centers of those clusters, which are called *centroids*; it allocates each observation to a cluster based the nearest cluster centroid. The objective of the K-means is to minimize the in-cluster sum of squares, that is the sum of the squared distance of every point to its corresponding cluster centroid.

Steps of the K-means algorithm:

1) The initial selection of cluster centroids. Centroids are either randomly generated or selected from the data set, i.e. K random observations are declared as centroids

Repeat:

2) Cluster assignment: For each observation from the data set, identify the nearest centroid (usually based on the Euclidean distance) and assign the observation to the cluster of that centroid.

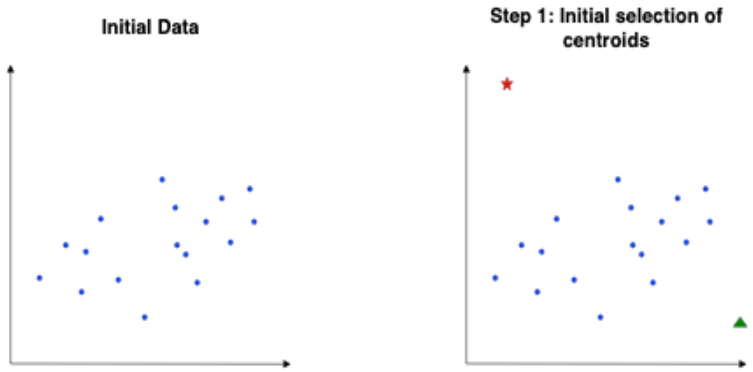
3) Centroid update: for each cluster, calculate a new centroid by taking the mean of all observations assigned to that cluster.

The algorithm iterates between steps 2 and 3 until one of the following conditions is met: i) no observation changes its cluster; ii) the sum of the distances is minimized (that is, falls beneath the threshold); iii) the maximum number of iterations is reached.

The result of the algorithm may be a local optimum that is not necessarily the best possible outcome. Therefore k-means algorithm is typically run several times, each time with a different (randomized) set of initial centroids. Eventually, the result of the best run is consider the final one.

K-means Example

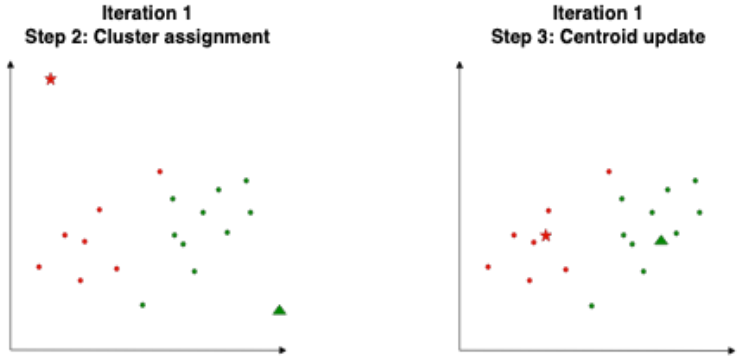
Suppose we have a dataset with several observations and two variables. They can be plotted in the Euclidean space (Fig. 1).



In the **Step 1** of the algorithm, we randomly choose the centroids. Suppose k=2, which means we need to choose two centroids. Although there are other approaches to choosing centroids, we will use the most basic one: choosing randomly. Still, it is advisable to generate the most disperse values possible (maximizing distance between centroids) in order to prevent unwanted localized convergence.

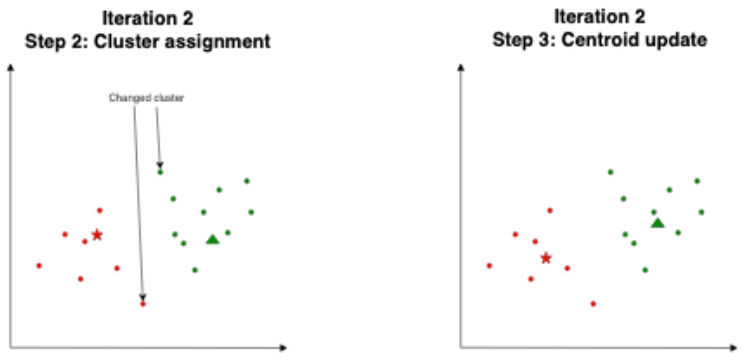
In the **Step 2**, we assign a cluster to each observation (Fig. 2). The assignment is performed based on the Euclidean distance measured from a data point (representing the observation) to centroids. An observation is assigned the cluster of the nearest centroid.

In the **Step 3**, we update the cluster centroids and assign them a new value based on the mean value of all observations that belong to the cluster (Fig. 2).



Cluster assignment and centroid update (Iteration 1)

In the next iteration (Iteration 2), we repeat Steps 2 and 3 (Fig. 3). In **Step 2**, we again assign each data point (observation) to a cluster. We can observe that two data points changed their cluster, i.e. they switched the cluster. After that, in **Step 3**, we again calculate new centroids as the mean value of all the observations that currently belong to the corresponding clusters.



Cluster assignment and centroid update (Iteration 2)

In Iteration 3, none of the observations would change their cluster. This means that the algorithm has converged and that no further iterations should be performed.

Load and prepare data set

Wholesale Customer dataset contains data about clients of a wholesale distributor. It includes the annual spending in monetary units (m.u.) on diverse product categories. The dataset is available from the UCI ML Repository. Note that the dataset used in this script is not the original one; the original data were partially preprocessed, which included the transformation of the *Channel* and *Region* attributes into factors as well as removal of outliers for some of the variables.

The objective is to segment (cluster) clients of the wholesale distributor.

Let's load the dataset.

```
# load the data from "data/wholesale_customers.csv"
customers.data <- read.csv(file = "data/wholesale_customers.csv")

# examine the data structure
str(customers.data)
```

```
## 'data.frame':    440 obs. of  8 variables:
##  $ Channel         : chr  "Retail" "Retail" "Retail" "Horeca" ...
##  $ Region          : chr  "Other_region" "Other_region" "Other_region" ...
##  $ Fresh           : num  12669 7057 6353 13265 22615 ...
##  $ Milk            : num  9656 9810 8808 1196 5410 ...
##  $ Grocery         : int  7561 9568 7684 4221 7198 5126 6975 9426 6192 ...
##  $ Frozen          : num  214 1762 2405 6404 3915 ...
##  $ Detergents_Paper: num  2674 3293 3516 507 1777 ...
##  $ Delicatessen    : num  1338 1776 3456 1788 3456 ...
```

Examining and Preparing the Data

We will first check if there are observations with missing values. If missing values are present, those instances should be either removed or imputed (imputation is the process of replacing missing data with substituted values).

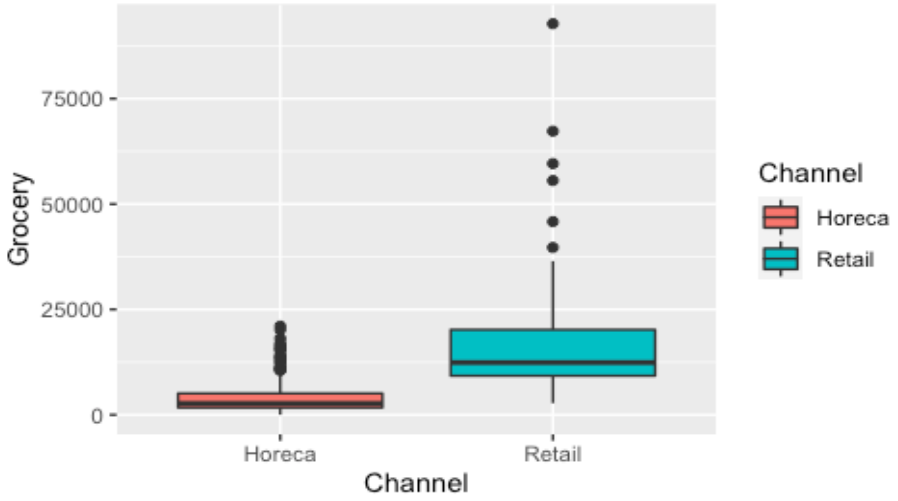
```r
# check for missing values
which(complete.cases(customers.data)==FALSE)

## integer(0)
```

In our dataset, there are no missing values.

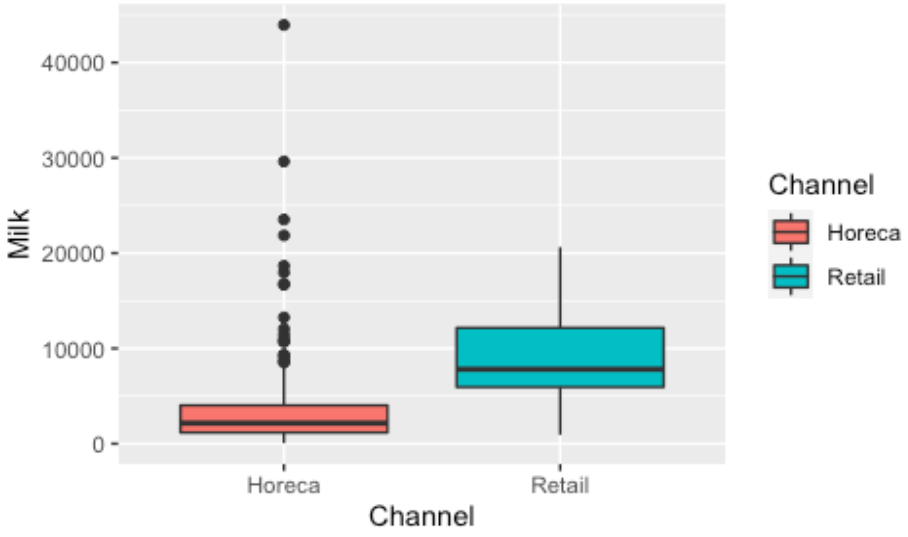
Since the algorithm is based on measuring distances (e.g. Euclidean), this implies that **all variables must be continuous** and the approach can be severely **affected by outliers**. So, we will use only numerical variables and will check if outliers are present.

Box-plots are useful for the detection of outliers. More details on how to analyze box plots can be found here.

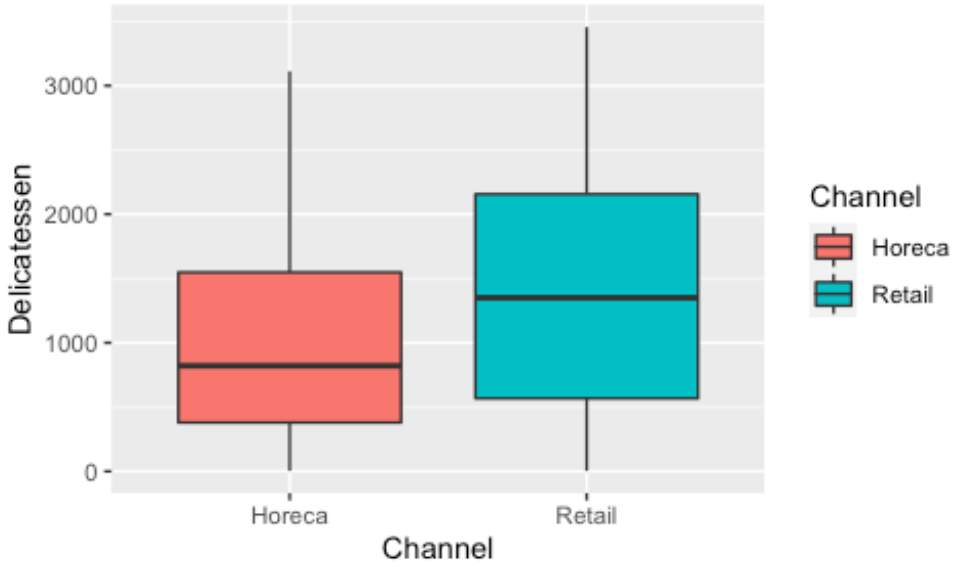
```r
# plot boxplots for all numeric variables
ggplot(customers.data, aes(x=Channel, y=Grocery, fill=Channel)) + geom_boxplot()
```



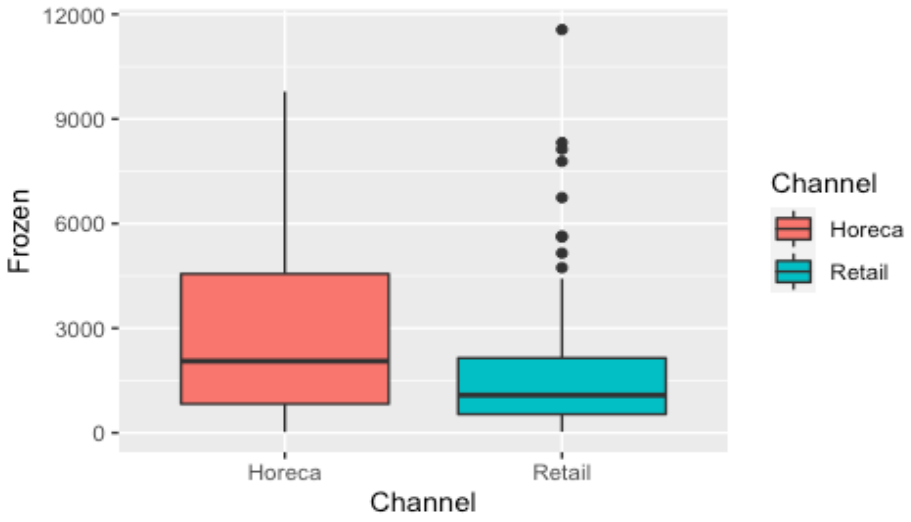

```r
ggplot(customers.data, aes(x=Channel, y=Milk, fill=Channel)) + geom_boxplot()
```



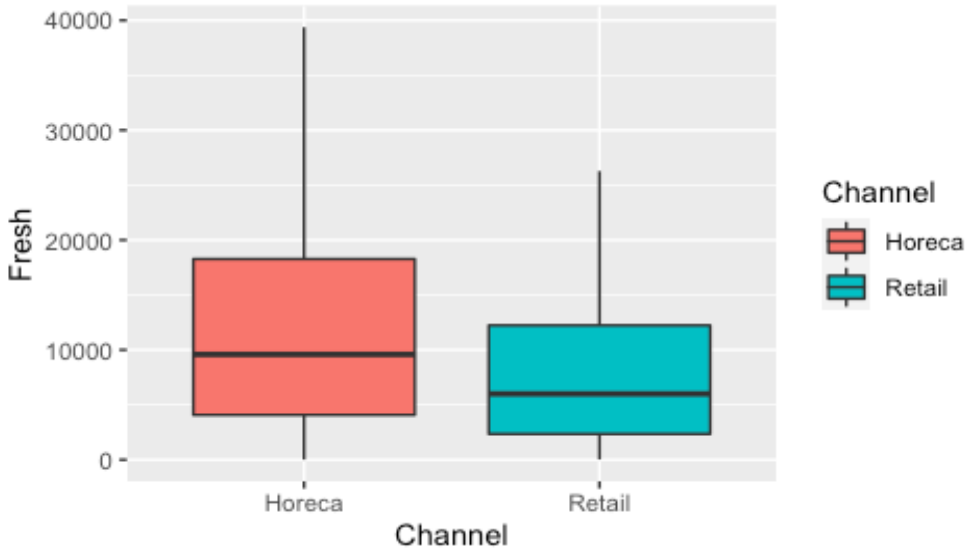
```
ggplot(customers.data, aes(x=Channel, y=Delicatessen, fill=Channel)) + geom_boxplot()
```



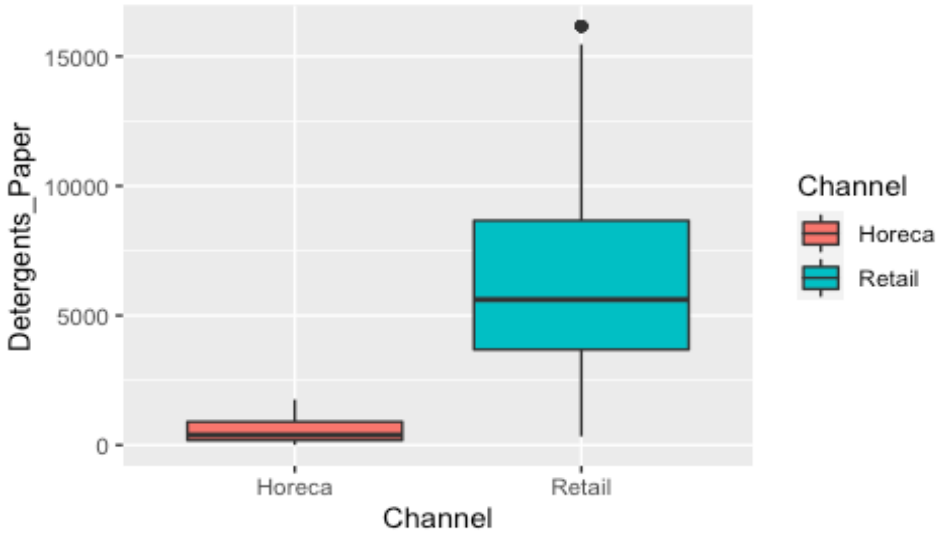

```
ggplot(customers.data, aes(x=Channel, y=Frozen, fill=Channel)) + geom_boxplot()
```




```
ggplot(customers.data, aes(x=Channel, y=Fresh, fill=Channel)) + geom_boxplot()
```




```
ggplot(customers.data, aes(x=Channel, y=Detergents_Paper, fill=Channel)) + geom_
boxplot()
```



A couple of variables have outliers namely Grocery, Milk, and Frozen.

The plots also suggest that there is a considerable difference between the two distribution channels, so it would be better to examine and cluster each of them separately.

```
# split the dataset into two subsets based on the value of the Channel variable
retail.data <- subset(customers.data, Channel == 'Retail')
horeca.data <- subset(customers.data, Channel == 'Horeca')
```

In the following, we will focus on the *retail.data*. **For the homework**: do the same with the *horeca.data*.

```r
# print the summary of the retail.data
summary(retail.data)
```

```
##    Channel            Region             Fresh            Milk
##  Length:142         Length:142         Min.   :   18   Min.   :  928
##  Class :character   Class :character   1st Qu.: 2348   1st Qu.: 5938
##  Mode  :character   Mode  :character   Median : 5994   Median : 7812
##                                        Mean   : 8460   Mean   : 9421
##                                        3rd Qu.:12230   3rd Qu.:12163
##                                        Max.   :26287   Max.   :20638
##     Grocery          Frozen        Detergents_Paper  Delicatessen
##  Min.   : 2743   Min.   :   33.0   Min.   :  332    Min.   :   3.0
##  1st Qu.: 9245   1st Qu.:  534.2   1st Qu.: 3684    1st Qu.: 566.8
##  Median :12390   Median : 1081.0   Median : 5614    Median :1350.0
##  Mean   :16323   Mean   : 1652.6   Mean   : 6650    Mean   :1485.2
##  3rd Qu.:20184   3rd Qu.: 2146.8   3rd Qu.: 8662    3rd Qu.:2156.0
##  Max.   :92780   Max.   :11559.0   Max.   :16171    Max.   :3455.6
```

Remove the *Channel* variable as we now have just one channel.

```r
# remove the Channel variable
retail.data$Channel <- NULL
```

Check which variables have outliers.

```r
# compute the number of outliers for all numeric variables
apply(X = retail.data[,-1], # all variables except Region
      MARGIN = 2,
      FUN = function(x) length(boxplot.stats(x)$out))
```

```
##            Fresh             Milk          Grocery           Frozen
##                0                0                6                9
## Detergents_Paper     Delicatessen
##                0                0
```

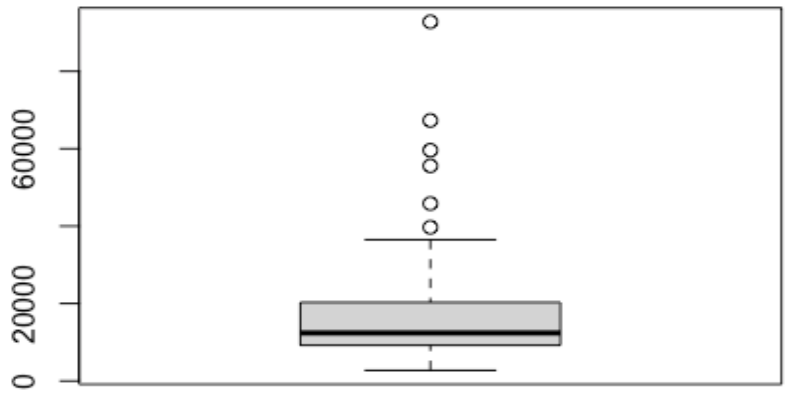
So, *Grocery* and *Frozen* variables have outliers that we need to deal with.

As a way of dealing with outliers, we'll use the Winsorizing technique. Practically, it consists of replacing extreme values with a specific percentile of the data, typically 95th for overly high values and 5th percentile for overly low values. To do that easily, we will use the *Winsorize* function from the *DescTools* package.

```r
# install.packages('DescTools')
library(DescTools)
```

Let's start with the *Grocery* variable.

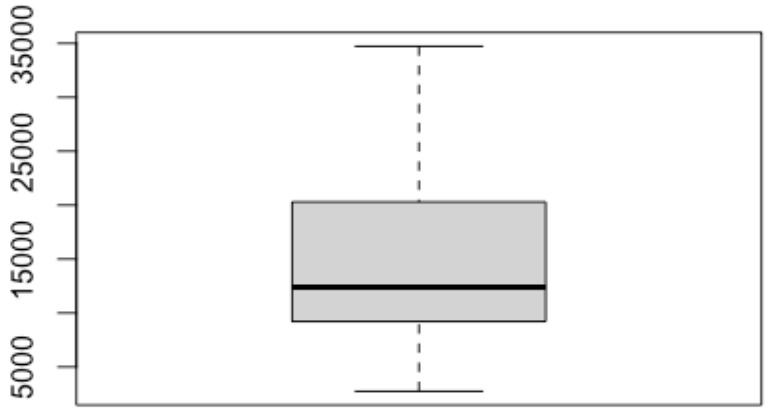
```r
boxplot(retail.data$Grocery, xlab='Grocery')
```



Grocery

All outliers are overly high values. So, we will replace them with the 95th percentile

```
grocery_w <- Winsorize(retail.data$Grocery,
                       probs = c(0,0.95))
# Note: we do not have overly low values, so, we set 0 as the 1st value for prob
s to indicate that no change should be done regarding low values; since we need
to replace overly high values (see the plot above), we set the 2nd value for pro
bs to 0.95 to indicate that we want to replace values above 95th percentile with
the value of the 95th percentile

# Plot the Grocery values after Winsorizing
boxplot(grocery_w, xlab='Grocery winsorized')
```

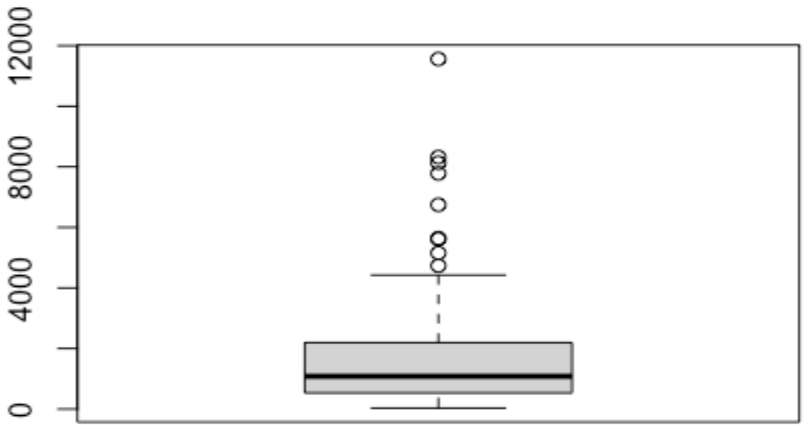


Grocery winsorized

We can see that no outliers are present anymore.

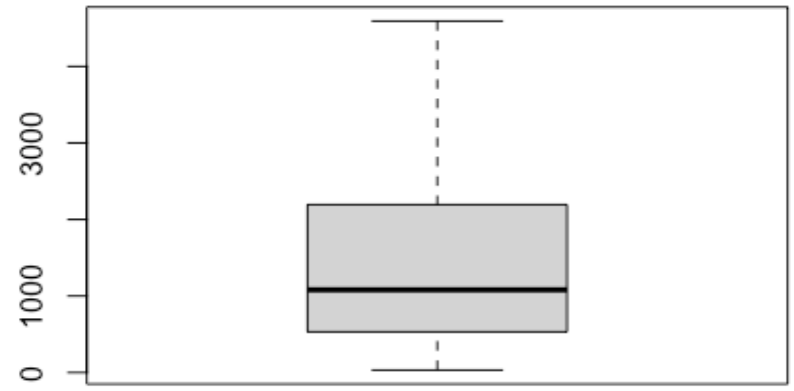
Now, we'll deal, in the same manner, with outliers for the *Frozen* variable.

```
boxplot(retail.data$Frozen, xlab='Frozen')
```



Frozen

```
# Again, we only have overly high values (no overly low values)
frozen_w <- Winsorize(retail.data$Frozen,
                      probs = c(0,0.94))
# Note that here we use 94th percentile, as with the 95th percentile we were not
able to remove all the outliers

# Plot the Grocery values after Winsorizing
boxplot(frozen_w, xlab='Frozen winsorized')
```



Frozen winsorized

Finally, update *retail.data* and examine it after the transformations.

```
retail.data$Grocery <- grocery_w
retail.data$Frozen <- frozen_w

# print the summary of the retail.data dataset
summary(retail.data)
```

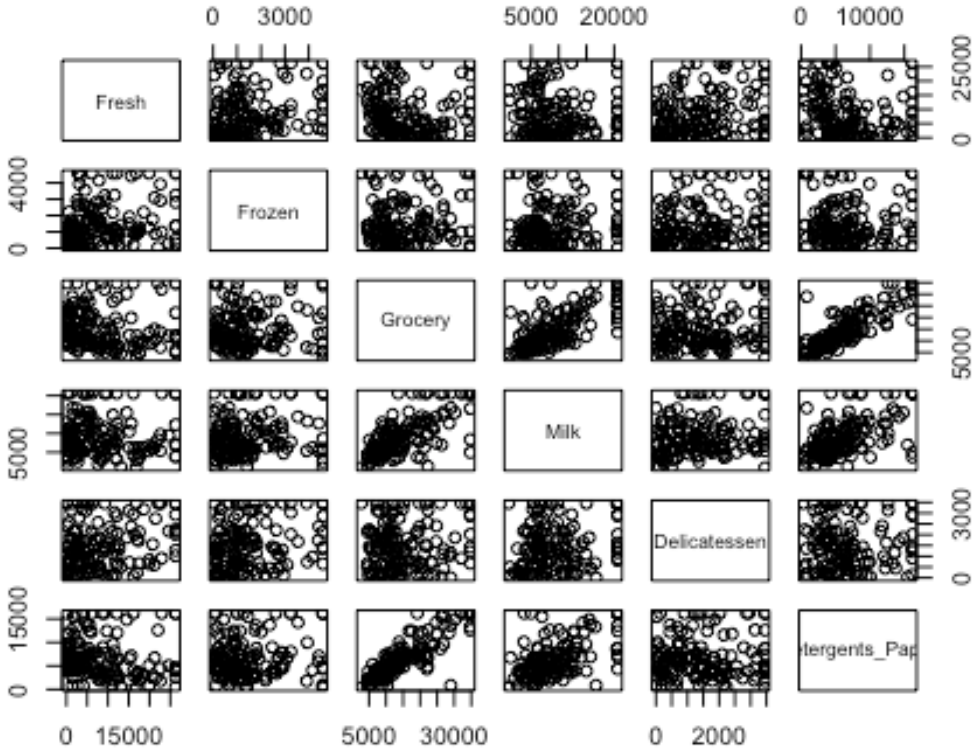
```
##     Region              Fresh             Milk          Grocery
##  Length:142         Min.   :   18   Min.   :  928   Min.   : 2743
##  Class :character   1st Qu.: 2348   1st Qu.: 5938   1st Qu.: 9245
##  Mode  :character   Median : 5994   Median : 7812   Median :12390
##                     Mean   : 8460   Mean   : 9421   Mean   :15237
##                     3rd Qu.:12230   3rd Qu.:12163   3rd Qu.:20184
##                     Max.   :26287   Max.   :20638   Max.   :34732
##      Frozen       Detergents_Paper  Delicatessen
##  Min.   :  33.0   Min.   :  332    Min.   :   3.0
##  1st Qu.: 534.2   1st Qu.: 3684    1st Qu.: 566.8
##  Median :1081.0   Median : 5614    Median :1350.0
##  Mean   :1495.2   Mean   : 6650    Mean   :1485.2
##  3rd Qu.:2146.8   3rd Qu.: 8662    3rd Qu.:2156.0
##  Max.   :4592.9   Max.   :16171    Max.   :3455.6
```

Clustering with 2 Features

Let's choose only two features for this initial, simple clustering.

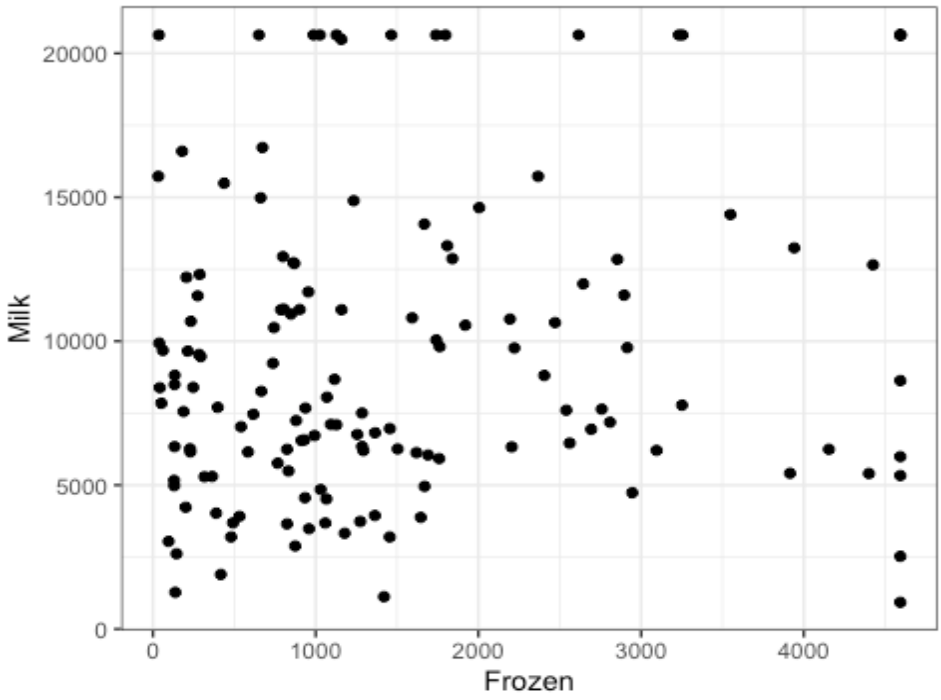
NOTE: Clustering with two features only is used here just for the sake of clearer explanation of the clustering steps and the results; in real-world situations, multiple (numerical) features are used for clustering.

```
# print the matrix of scatterplots for all numeric variables
pairs(~Fresh+Frozen+Grocery+Milk+Delicatessen+Detergents_Paper, data = retail.da
ta)
```



By looking at the plots, no particular pattern can be observed. But for the sake of an example, let's pick *Frozen* and *Milk* variables. Get a closer look at selected variables.

```
# plot the scatterplot for the variables Frozen and Milk
ggplot(data=retail.data, aes(x=Frozen, y=Milk)) +
  geom_point() +
  theme_bw()
```



No clear pattern in the data, but let's run K-means and see if some clusters will emerge.

Create a subset of the original data containing the attributes to be used in the K-means.

```
# create a subset of the data with variables Frozen and Milk
retail.data1 <- retail.data[, c("Frozen", "Milk")]

# print the summary of the new dataset
summary(retail.data1)
```

```
##      Frozen            Milk
##  Min.   :  33.0   Min.   :  928
##  1st Qu.: 534.2   1st Qu.: 5938
##  Median :1081.0   Median : 7812
##  Mean   :1495.2   Mean   : 9421
##  3rd Qu.:2146.8   3rd Qu.:12163
##  Max.   :4592.9   Max.   :20638
```

When variables are in incomparable units and/or the numeric values are on very different scales of magnitude, they should be rescaled. Since our variables Frozen and Milk have different value ranges, we need to rescale the data. To that end, we will use normalization as there are no outliers. Normalization is done using the formula:

$$Z = \frac{X - min(X)}{max(X) - min(X)}$$

Let's create a function for performing the normalization.

```r
# function for performing the normalization
normalize.feature <- function( feature ) {
  if ( sum(feature, na.rm = T) == 0 ) feature
  else ((feature - min(feature, na.rm = T))/(max(feature, na.rm = T) - min(featu
re, na.rm = T)))
}
```

Then, we normalize all variables by calling our custom function *normalize.feature.*

```r
# normalize both variables
retail.data1.norm <- as.data.frame(apply(retail.data1, 2, normalize.feature))

# print the summary
summary(retail.data1.norm)
```

```
##      Frozen            Milk
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.1099   1st Qu.:0.2542
##  Median :0.2298   Median :0.3493
##  Mean   :0.3207   Mean   :0.4309
##  3rd Qu.:0.4635   3rd Qu.:0.5700
##  Max.   :1.0000   Max.   :1.0000
```

Run the K Means algorithm using the *kmeans* function

```r
# open the documentation for the kmeans function
?kmeans
```

Initially, we will try to group data into, for example, 4 clusters (k=4). We use the *iter.max* parameter to define the maximum number of iterations. This overcomes the problem of slow (or no) convergence. K-means tends to be sensitive to the initial selection of centroids. Thus, *nstart* parameter is used to indicate the number of initial configurations to try (and eventually select the best one).

Since the initial cluster centers (centroids) are randomly selected, we need to set the seed to assure replicability of the results.

```r
# set the seed
set.seed(5320)

# run the clustering with 4 clusters, iter.max=20, nstart=1000
simple.4k <- kmeans(x = retail.data1.norm, centers=4, iter.max=20, nstart=1000)

# print the model
simple.4k
```

```
## K-means clustering with 4 clusters of sizes 25, 10, 73, 34
##
## Cluster means:
##      Frozen      Milk
## 1 0.7023575 0.3335116
```

```
## 2 0.8565630 0.8903003
## 3 0.1636317 0.2670127
## 4 0.2195785 0.7192044
##
## Clustering vector:
##   1   2   3   5   6   7   8  10  11  12  13  14  15  17  19  21  24  25  26
29
##   3   3   1   1   3   3   3   4   1   3   4   1   3   3   1   3   2   1   3
4
##  36  38  39  43  44  45  46  47  48  49  50  53  54  57  58  61  62  63  64
66
##   3   4   4   3   4   3   4   4   2   3   4   3   3   2   3   3   2   1   1
4
##  68  74  75  78  82  83  85  86  87  93  95  97 101 102 103 107 108 109 110 1
12
##   3   1   3   4   3   3   3   4   4   2   4   3   1   4   1   3   1   3   4
4
## 124 128 146 156 157 159 160 161 164 165 166 167 171 172 174 176 189 190 194 1
98
##   1   3   3   3   3   3   3   3   4   1   2   3   3   4   3   3   1   4   3
3
## 201 202 206 208 210 212 215 217 219 224 227 231 246 252 265 267 269 280 282 2
94
##   2   2   4   1   4   2   3   4   3   1   3   1   3   2   3   1   3   3   3
4
## 296 298 299 301 302 303 304 305 306 307 310 313 316 320 332 334 335 336 341 3
42
##   3   3   3   3   4   3   3   3   4   4   4   3   4   4   4   3   1   3   3
3
## 344 347 348 350 352 354 358 366 371 374 377 380 397 408 409 416 417 419 422 4
24
##   3   1   1   4   3   3   3   3   3   3   1   3   1   1   3   3   4   3   3
3
## 425 438
##   3   4
##
## Within cluster sum of squares by cluster:
## [1] 1.4103914 0.5165821 1.9633244 1.7969693
##  (between_SS / total_SS =  73.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss
"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

From the output, we can observe the following evaluation metrics:

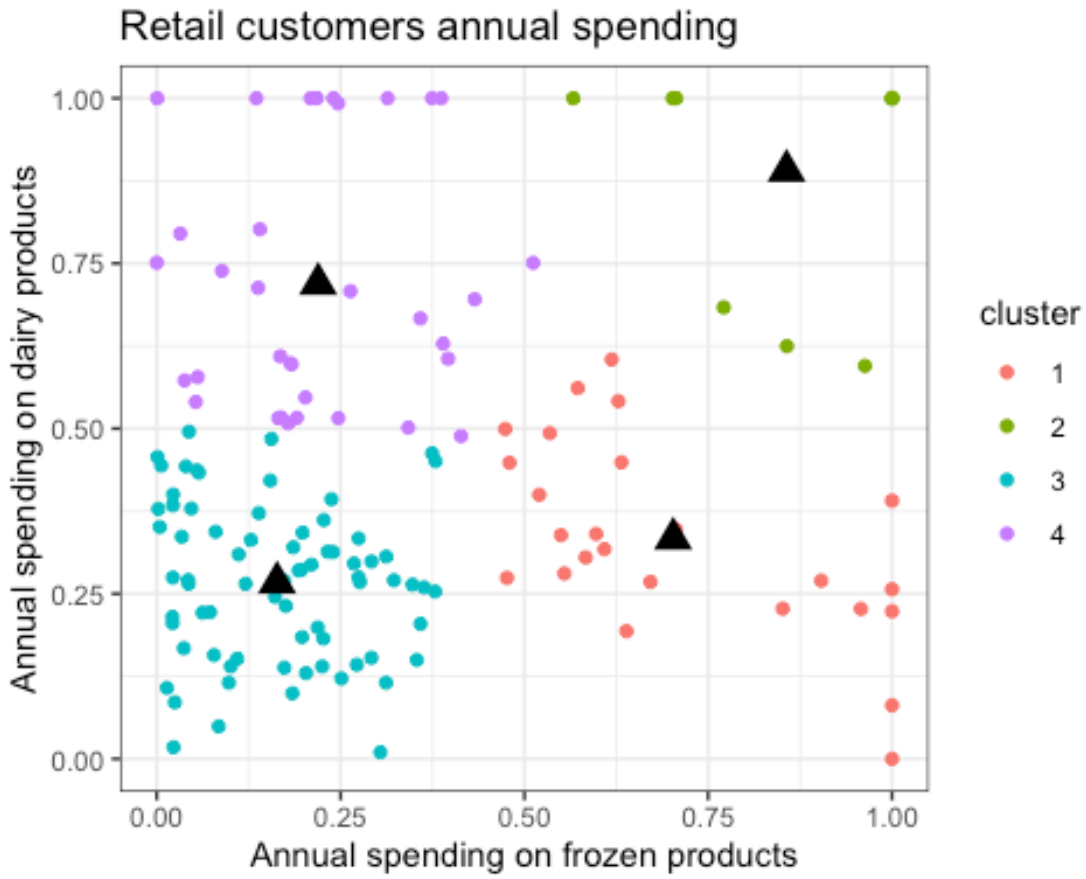
- **within_SS (within-cluster sum of squares)**: The sum of squared differences between individual data points in a cluster and the cluster center (centroid). It is computed for each cluster separately;
- **total_SS (total sum of squares)**: The sum of squared differences of each data point to the global sample mean;
- **between_SS (between cluster sum of squares)**: The sum of squared differences of each centroid to the global sample mean; when computing this value, the squared difference of each cluster center to the global sample mean is multiplied by the number of data points in that cluster;
- **between_SS / total_SS**: This ratio indicates how well the sample splits into clusters; the higher the ratio, the better the clustering. The maximum is 1.

Add the vector of cluster assignments to the data frame as a new variable *cluster* and plot it.

```
# add the cluster as a new variable to the dataset
retail.data1.norm$cluster <- factor(simple.4k$cluster)

# print several instances of the dataset
head(retail.data1.norm)

##       Frozen      Milk cluster
## 1 0.03969350 0.4428231       3
## 2 0.37917166 0.4506365       3
## 3 0.52018228 0.3997991       1
## 5 0.85132699 0.2273984       1
## 6 0.13881762 0.3719451       3
## 7 0.09802761 0.1152213       3
```

Let's plot the data and new centroids to visually inspect the clusters.

```
# plot the clusters along with their centroids and color the points by their res
pective clusters
ggplot(data=retail.data1.norm, aes(x=Frozen, y=Milk, colour=cluster)) +
  geom_point() +
  labs(x = "Annual spending on frozen products",
       y = "Annual spending on dairy products",
       title = "Retail customers annual spending") +
  theme_bw() +
  # add cluster centers
  geom_point(data=as.data.frame(simple.4k$centers), colour="black", size=4, shap
e=17)
```



We have set K to 4 by making a random guess. However, we should adopt a more systematic approach to selecting the value for K.

Selecting the Best Value for K (the Elbow Method)

Instead of guessing the best value for K, we can take a more systematic approach to choosing that value. It is called the **Elbow method**, and is based on the sum of squared differences between data points and cluster centers, that is, the sum of *within_SS* for all the clusters (*tot.withinss*).

Let us run the K-means for different values for K (from 2 to 8). We will compute the metric required for the Elbow method (*tot.withinss*). Along the way, we'll also compute another useful metric: *ratio of between_SS and total_SS*.

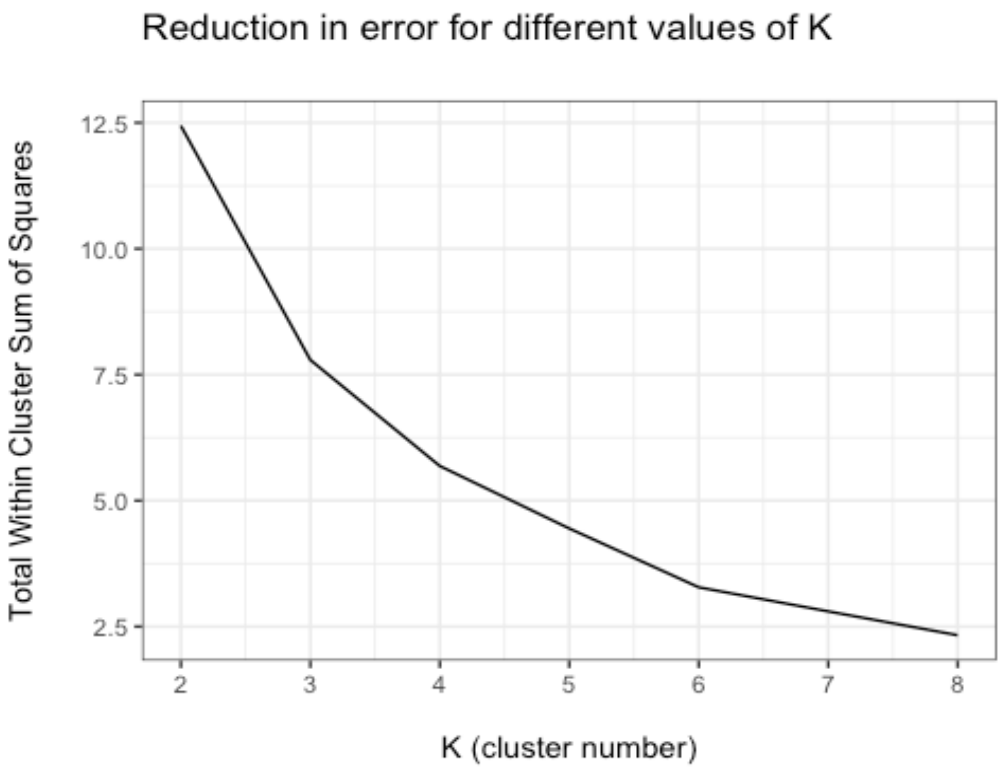
```
# create an empty data frame for storing evaluation measures for different k val
ues
eval.metrics.2var <- data.frame()

# remove the column with cluster assignments
retail.data1.norm$cluster <- NULL

# run kmeans for all K values in the range 2:8
```

```r
for (k in 2:8) {
  set.seed(5320)
  km.res <- kmeans(x=retail.data1.norm, centers=k, iter.max=20, nstart = 1000)
  # combine cluster number and the evaluation measures, write to df
  eval.metrics.2var <- rbind(eval.metrics.2var,
                       c(k, km.res$tot.withinss, km.res$betweenss/km.res$totss))
}

# assign more meaningful column names
names(eval.metrics.2var) <- c("k", "tot.within.ss", "ratio")

# print the metrics
eval.metrics.2var
```

```
##   k tot.within.ss     ratio
## 1 2     12.440575 0.4209188
## 2 3      7.792814 0.6372618
## 3 4      5.687267 0.7352703
## 4 5      4.447272 0.7929894
## 5 6      3.279949 0.8473257
## 6 7      2.801294 0.8696060
## 7 8      2.328407 0.8916178
```

Draw the Elbow plot.

```r
# plot the line chart for K values vs. tot.within.ss
ggplot(data=eval.metrics.2var, aes(x=k, y=tot.within.ss)) +
  geom_line() +
  labs(x = "\nK (cluster number)",
       y = "Total Within Cluster Sum of Squares\n",
       title = "Reduction in error for different values of K\n") +
  theme_bw() +
  scale_x_continuous(breaks=seq(from=0, to=8, by=1))
```



It seems that *K=3* or *K=4* would be the best options for the number of clusters.

If it is not fully clear from the plot where we have a significant decrease in the *tot.within.ss* value, we can compute the difference between every two subsequent values. We will load the script with our utility functions (code is given at the end of the file) and call our custom function *compute.difference*.

```
# load the source code from the Utility.R file
source("Utility.R")

# calculate the difference in tot.within.ss and in ratio for each two consecutiv
e K values (from 2 to 8)
data.frame(K=2:8,
           tot.within.ss.delta=compute.difference(eval.metrics.2var$tot.within.s
s),
           ratio.delta=compute.difference(eval.metrics.2var$ratio))
```

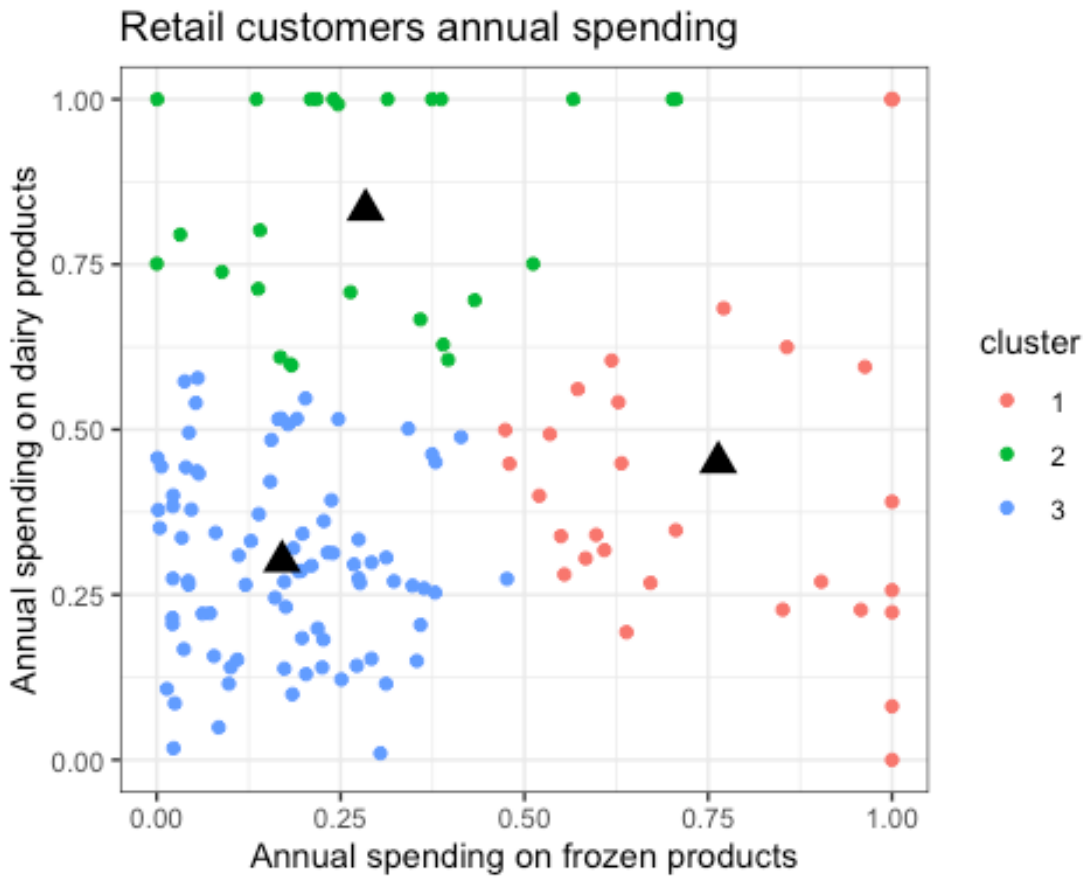
```
##   K tot.within.ss.delta ratio.delta
## 1 2                  NA          NA
## 2 3           4.6477612  0.21634297
## 3 4           2.1055470  0.09800854
## 4 5           1.2399954  0.05771904
## 5 6           1.1673229  0.05433629
## 6 7           0.4786544  0.02228030
## 7 8           0.4728878  0.02201188
```

The results suggest that the largest drop is between k=2 and k=3. So, let's examine the solution with k=3.

```
# set the seed value
set.seed(5320)

# run the clustering for 3 clusters, iter.max=20, nstart=1000
simple.3k <- kmeans(x = retail.data1.norm, centers=3, iter.max=20, nstart=1000)

# store the (factorized) cluster value in a new variable
retail.data1.norm$cluster <- factor(simple.3k$cluster)
```

Plot the 3-cluster solution.

```
# plot the clusters along with their centroids
ggplot(data=retail.data1.norm, aes(x=Frozen, y=Milk, colour=cluster)) +
  geom_point() +
  labs(x = "Annual spending on frozen products",
       y = "Annual spending on dairy products",
       title = "Retail customers annual spending") +
  theme_bw() +
  geom_point(data=as.data.frame(simple.3k$centers), colour="black", size=4, shap
e=17)
```



Next, we examine clusters closer by looking into the cluster centers (mean) and standard deviation from the centers. When examining clusters - in order to interpret them - we typically use 'regular' (not normalized) features. The *summary.stats()* f. used below is a custom function defined in the *Utility.R*.

```
# calculate the mean and sd values for all three clusters
sum.stats <- summary.stats(feature.set = retail.data1,
                           clusters = simple.3k$cluster,
                           cl.num = 3)

sum.stats
```

```
##   attributes         Mean (SD)          Mean (SD)         Mean (SD)
## 1    cluster                 1                  2                 3
## 2       freq                31                 26                85
## 3     Frozen  3515.31 (906.96)   1328.08 (879.37)   809.65 (545.72)
## 4       Milk 9808.18 (5253.35) 17342.03 (3268.35) 6856.39 (2766.19)
```

Considering the high level of dispersion around the cluster centers (observe the high value of SD in all clusters and for both variables), it might be that the solution with K=4 clusters (the inital one) is better after all. To check if that is the case, for K=4, compute the cluster centers (mean) and dispersion (SD).

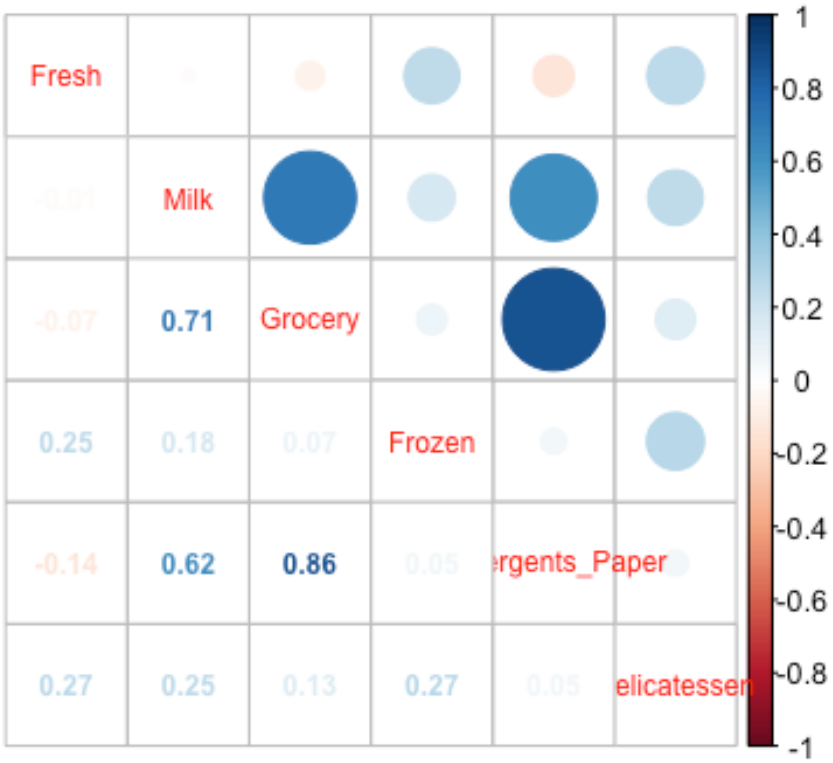
Clustering with several numeric features

First, we need to check if some of the variable are highly correlated. The use of highly correlated variables can negatively affect cluster analysis so that patterns detected in the data may not be the true ones. Therefore, we will first compute and plot correlations for all variable pairs.

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
# compute correlations (avoid the Region variable)
corr.matrix <- cor(retail.data[,-1])
# create correlations plot (as was done in the linear regression lab)
corrplot.mixed(corr.matrix, tl.cex=0.75, number.cex=0.75)
```



Since Grocery is highly correlated with both Detergents_Paper and Milk, we will not use it for clustering.

```
retail.data <- retail.data[,-4]

summary(retail.data)
```

```
##     Region              Fresh             Milk          Frozen
##  Length:142         Min.   :   18   Min.   :  928   Min.   :  33.0
##  Class :character   1st Qu.: 2348   1st Qu.: 5938   1st Qu.: 534.2
##  Mode  :character   Median : 5994   Median : 7812   Median :1081.0
##                     Mean   : 8460   Mean   : 9421   Mean   :1495.2
##                     3rd Qu.:12230   3rd Qu.:12163   3rd Qu.:2146.8
##                     Max.   :26287   Max.   :20638   Max.   :4592.9
##  Detergents_Paper  Delicatessen
##  Min.   :  332    Min.   :   3.0
##  1st Qu.: 3684    1st Qu.: 566.8
##  Median : 5614    Median :1350.0
##  Mean   : 6650    Mean   :1485.2
##  3rd Qu.: 8662    3rd Qu.:2156.0
##  Max.   :16171    Max.   :3455.6
```

Since the 5 remaining numeric attributes differ in their value ranges, we need to normalize them, that is, transform them to the value range [0,1].

```
# normalize all numeric variables from the retail.data dataset
retail.norm <- as.data.frame(apply(retail.data[,c(2:6)], 2, normalize.feature))
```

```r
# print the summary
summary(retail.norm)
```

```
##      Fresh              Milk            Frozen       Detergents_Paper
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.08869   1st Qu.:0.2542   1st Qu.:0.1099   1st Qu.:0.2116
##  Median :0.22747   Median :0.3493   Median :0.2298   Median :0.3335
##  Mean   :0.32138   Mean   :0.4309   Mean   :0.3207   Mean   :0.3989
##  3rd Qu.:0.46487   3rd Qu.:0.5700   3rd Qu.:0.4635   3rd Qu.:0.5260
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##   Delicatessen
##  Min.   :0.0000
##  1st Qu.:0.1633
##  Median :0.3901
##  Mean   :0.4293
##  3rd Qu.:0.6236
##  Max.   :1.0000
```

Instead of guessing K, we'll right away use the Elbow method to find the optimal value for K.

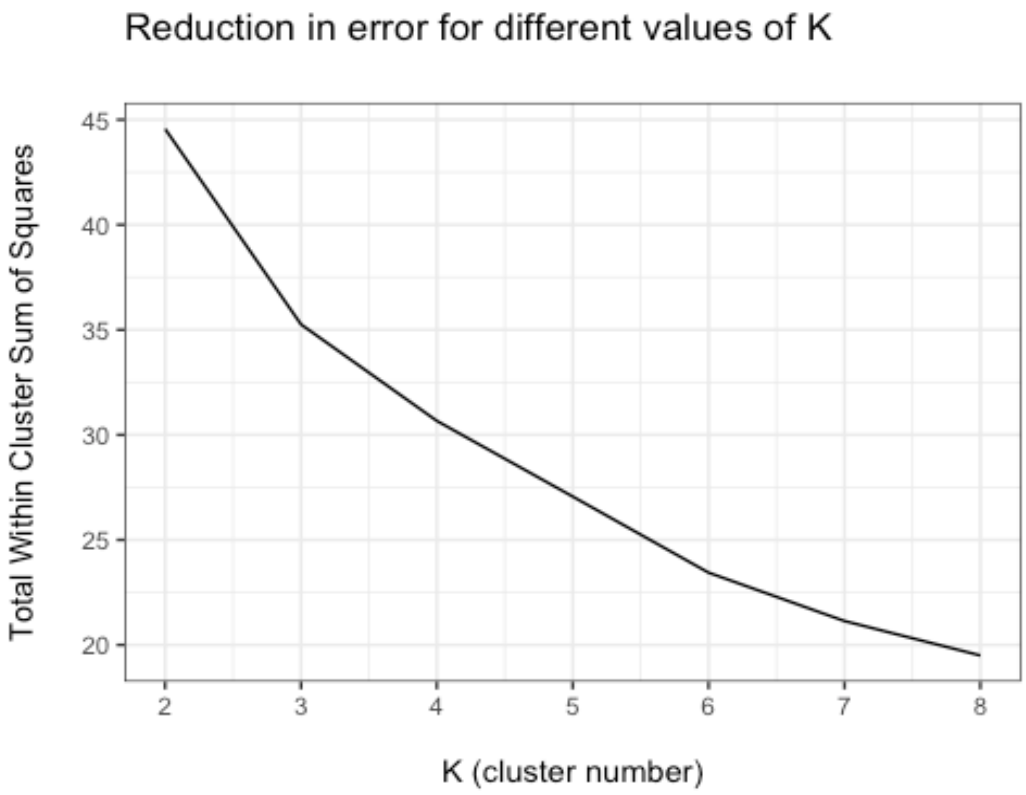
```r
# create an empty data frame to store evaluation measures
eval.metrics.5var <- data.frame()

# run kmeans for K values in the range 2:8
for (k in 2:8) {
  set.seed(5320)
  km.res <- kmeans(x=retail.norm, centers=k, iter.max=20, nstart = 1000)
  # combine the K value and the evaluation measures, write to df
  eval.metrics.5var <- rbind(eval.metrics.5var,
                       c(k, km.res$tot.withinss, km.res$betweenss/km.res$totss))
}

# assign meaningful column names
names(eval.metrics.5var) <- c("k", "tot.within.ss", "ratio")
eval.metrics.5var
```

```
##   k tot.within.ss     ratio
## 1 2      44.55835 0.2252166
## 2 3      35.25739 0.3869423
## 3 4      30.66352 0.4668209
## 4 5      27.06268 0.5294324
## 5 6      23.43998 0.5924242
## 6 7      21.13826 0.6324466
## 7 8      19.50035 0.6609268
```

Draw the Elbow plot.

```r
# plot the line chart for K values vs. tot.within.ss
ggplot(data=eval.metrics.5var, aes(x=k, y=tot.within.ss)) +
  geom_line() +
```

```
  labs(x = "\nK (cluster number)",
       y = "Total Within Cluster Sum of Squares\n",
       title = "Reduction in error for different values of K\n") +
  theme_bw() +
  scale_x_continuous(breaks=seq(from=0, to=8, by=1))
```



This time it is clearer that the solution with 3 clusters might be the best. Still, we'll also examine the difference between each two consecutive values of tot.within.ss and ratio (of *between.SS* and *total.SS*).

```
# calculate the difference in tot.within.ss and in ratio for each two consecutiv
e K values
data.frame(k=c(2:8),
        tot.within.ss.delta=compute.difference(eval.metrics.5var$tot.within.ss),
        ration.delta=compute.difference(eval.metrics.5var$ratio))
```

```
##   k tot.within.ss.delta ration.delta
## 1 2                  NA           NA
## 2 3            9.300962   0.16172571
## 3 4            4.593874   0.07987856
## 4 5            3.600833   0.06261151
## 5 6            3.622707   0.06299186
## 6 7            2.301718   0.04002241
## 7 8            1.637913   0.02848013
```

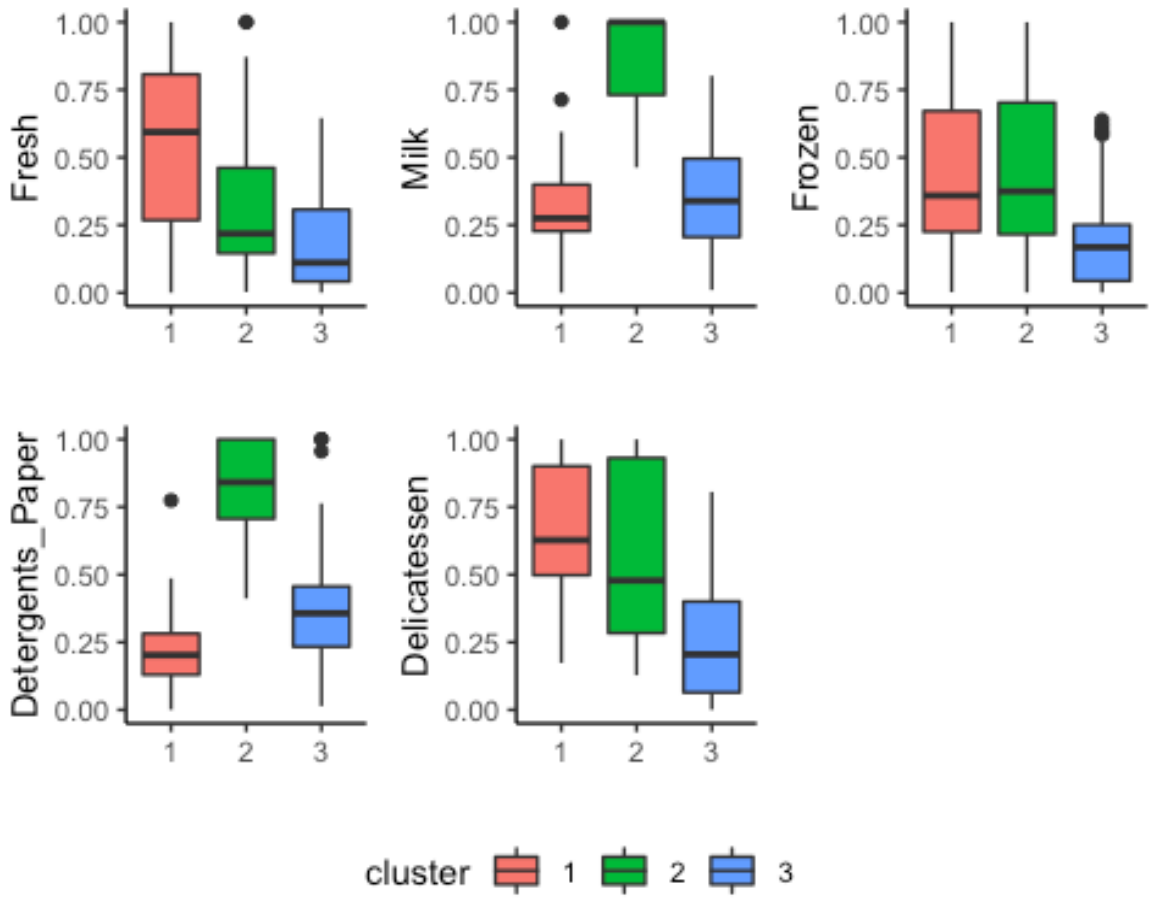
This again suggests 3 clusters as the best option.

Let's examine in detail clustering with K=3.

```
# set the seed
set.seed(5320)

# run the clustering for 3 clusters, iter.max=20, nstart=1000
retail.3k.5var <- kmeans(x=retail.norm, centers=3, iter.max=20, nstart = 1000)

# examine cluster centers
sum.stats <- summary.stats(feature.set = retail.norm, # retail.data[,c(2:6)],
                           clusters = retail.3k.5var$cluster,
                           cl.num = 3)
sum.stats
```

```
##         attributes   Mean (SD)   Mean (SD)   Mean (SD)
## 1          cluster           1           2           3
## 2             freq          45          24          73
## 3            Fresh 0.54 (0.32)  0.33 (0.3) 0.19 (0.17)
## 4             Milk 0.33 (0.17) 0.88 (0.17) 0.35 (0.18)
## 5           Frozen 0.46 (0.32) 0.44 (0.31) 0.19 (0.17)
## 6 Detergents_Paper 0.22 (0.14)  0.81 (0.2)  0.37 (0.2)
## 7     Delicatessen 0.65 (0.26) 0.57 (0.33)  0.25 (0.2)
```

```
# Plot the distribution of the attributes across the 3 clusters
# See the description of the create_comparison_plots() function (in the Utility.
R script)
# to see what arguments the function expects
create_comparison_plots(df = retail.norm,
                        clust = as.factor(retail.3k.5var$cluster))
```



TASK: try to describe the 3 groups (clusters) of customers based on the items they purchased more / less (compared to the other groups).

Appendix: Utility Functions ('Utility.R' file)

```
# The function computes the difference between each two consecutive
# values in the given vector of values
compute.difference <- function(values) {
  dif <- list()
  dif[[1]] <- NA
  for(i in 1:(length(values)-1)) {
    dif[[i+1]] <- abs(values[i+1] - values[i])
  }
  unlist(dif)
}
```

```r
# The function computes summary statistics for individual clusters
summary.stats <- function(feature.set, clusters, cl.num) {
  sum.stats <- aggregate(x = feature.set,
                         by = list(clusters),
                         FUN = function(x) {
                           m <- mean(x, na.rm = T)
                           sd <- sqrt(var(x, na.rm = T))
                           paste(round(m, digits = 2), " (",
                                 round(sd, digits = 2), ")", sep = "")
                         })
  sum.stat.df <- data.frame(cluster = sum.stats[,1],
                            freq = as.vector(table(clusters)),
                            sum.stats[,-1])

  sum.stats.transpose <- t( as.matrix(sum.stat.df) )
  sum.stats.transpose <- as.data.frame(sum.stats.transpose)
  attributes <- rownames(sum.stats.transpose)
  sum.stats.transpose <- as.data.frame( cbind(attributes, sum.stats.transpose) )
  colnames(sum.stats.transpose) <- c( "attributes", rep("Mean (SD)", cl.num) )
  rownames(sum.stats.transpose) <- NULL
  sum.stats.transpose
}


# The following two functions are for creating and arranging a set of
# box plots, one plot for each attribute that was used for clustering.
# The purpose is to visually compare the distribution of the attributes
# across the clusters
#
# The 'main' function is create_comparison_plots() that receives
# 2 input parameters:
# - a data frame with the attributes used for clustering
# - a vector with cluster assignments
# The function creates and arranges box plots for all the attributes.
#
# The create_attr_boxplot() is a helper function for the create_comparison_plots
# f. that creates a box plot for one attribute


create_attr_boxplot <- function(df, attribute, clust_var) {
  ggplot(data = df,
         mapping = aes(x=.data[[clust_var]],
                       y=.data[[attribute]],
                       fill=.data[[clust_var]])) +
    geom_boxplot() +
    labs(y = attribute, x = "") +
    theme_classic()
}
```

```r
create_comparison_plots <- function(df, clust) {
  require(dplyr)
  require(ggpubr)

  df_clust <- df
  df_clust[['cluster']] <- clust
  boxplots <- lapply(colnames(df),
                     function(x) create_attr_boxplot(df_clust, x, 'cluster'))

  ggarrange(plotlist = boxplots,
            ncol = 3, nrow = 2,
            common.legend = TRUE, legend = "bottom",
            vjust = 1, hjust = -1, font.label = list(size=12))

}
```