K-means Clustering

1 Short Intro

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.

An input to the K-means algorithm is a number **k**, which refers to the number of groups (clusters) that the observations will be grouped into. The algorithm actually finds centers of those clusters (called *centroids*) and allocates each observation to the nearest cluster. 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 instance from the data set, identify the nearest centroid (usually based on the Euclidean distance) and assign the instance to the cluster of that centroid.
3) Centroid update: for each cluster, calculate a new centroid by taking the mean of all data points assigned to that centroid's cluster.

The algorithm iterates between steps 2 and 3 until no data points change clusters, the sum of the distances is minimized, or a maximum number of iterations is reached.

The result may be a local optimum (which may not necessarily be the best possible outcome). This means that running the algorithm with randomized starting centroids may give better results.

1.1 K-means Example

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

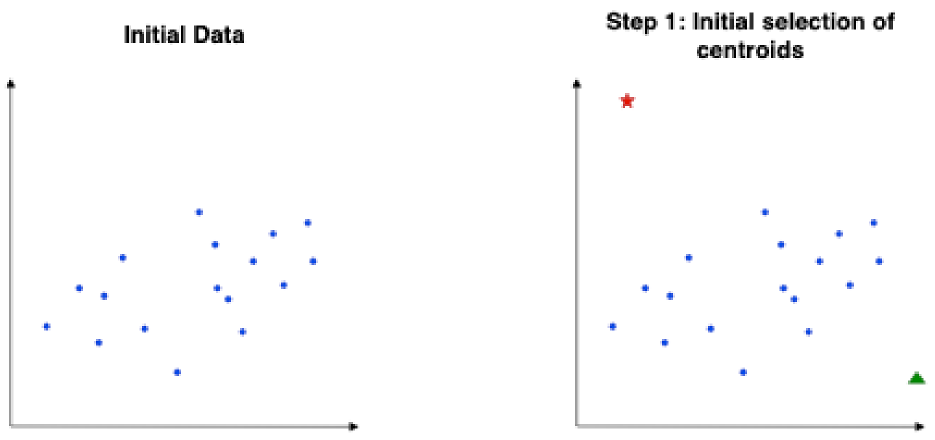


Figure 1: Initial data and Initial selection of centroids

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).

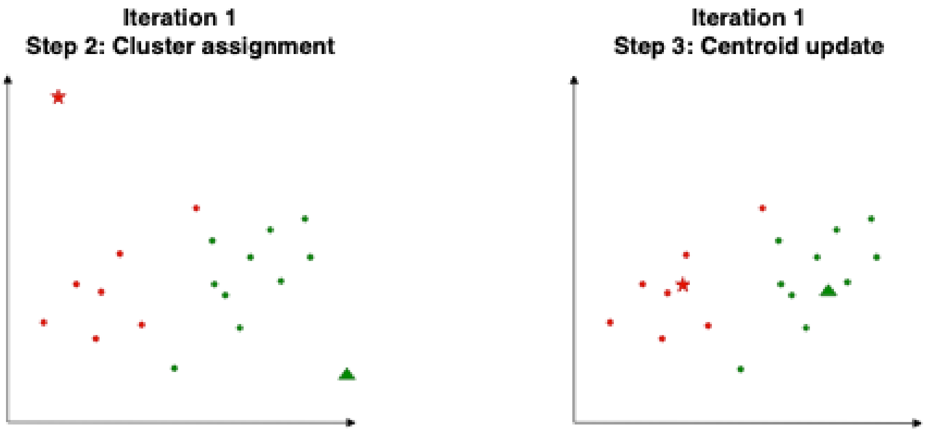


Figure 2: Cluster assignment and centroid update (Iteration 1)

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

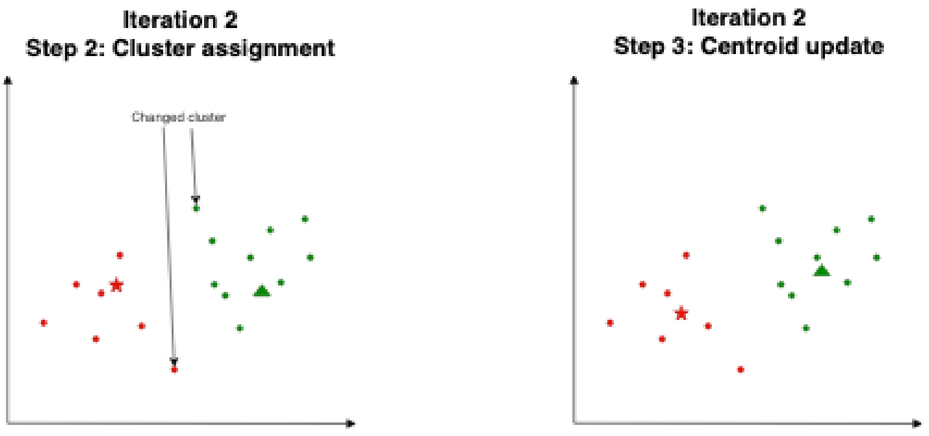


Figure 3: Cluster assignment and centroid update (Iteration 2)

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.

2 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 (the dataset used in this script is partially preprocessed, where *Channel* and *Region* attributes are factorized and outliers were removed for some variables).

The objective is to segment (cluster) customers.

Let's load the dataset.

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

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

```
## 'data.frame':    440 obs. of  8 variables:
##  $ Channel         : Factor w/ 2 levels "Horeca","Retail": 2 2 2 1 2 2 2 2 1 2 ...
##  $ Region          : Factor w/ 3 levels "Lisbon","Oporto",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ 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 18881 ...
##  $ Frozen          : num  214 1762 2405 6404 3915 ...
##  $ Detergents_Paper: num  2674 3293 3516 507 1777 ...
##  $ Delicatessen    : num  1338 1776 3456 1788 3456 ...
```

2.1 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).

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

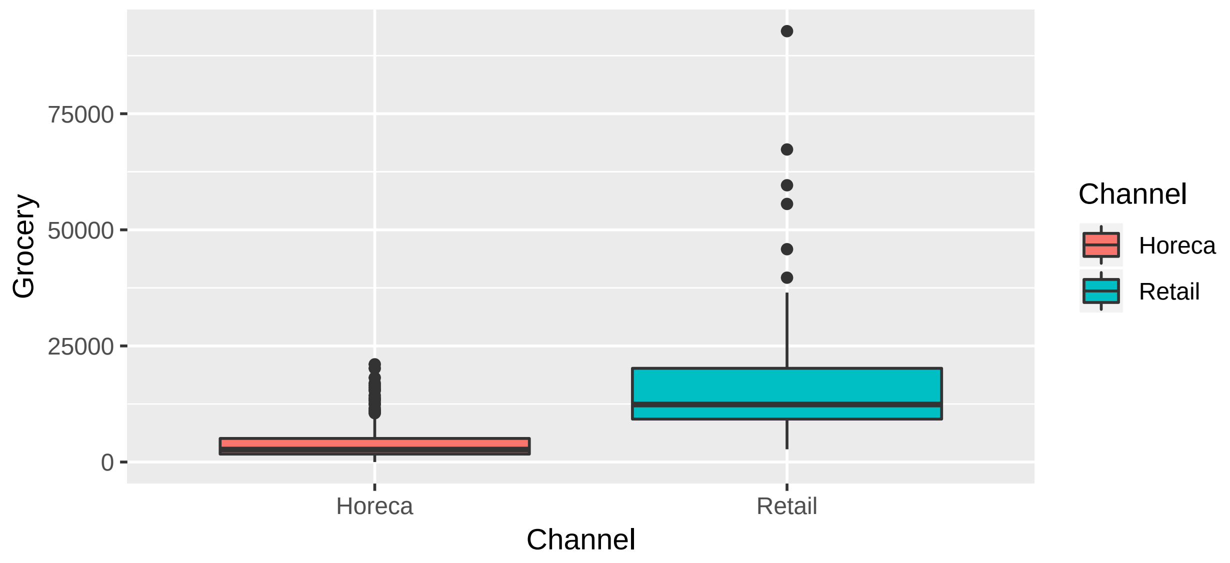
```
## integer(0)
```

In our dataset, there are no missing values.

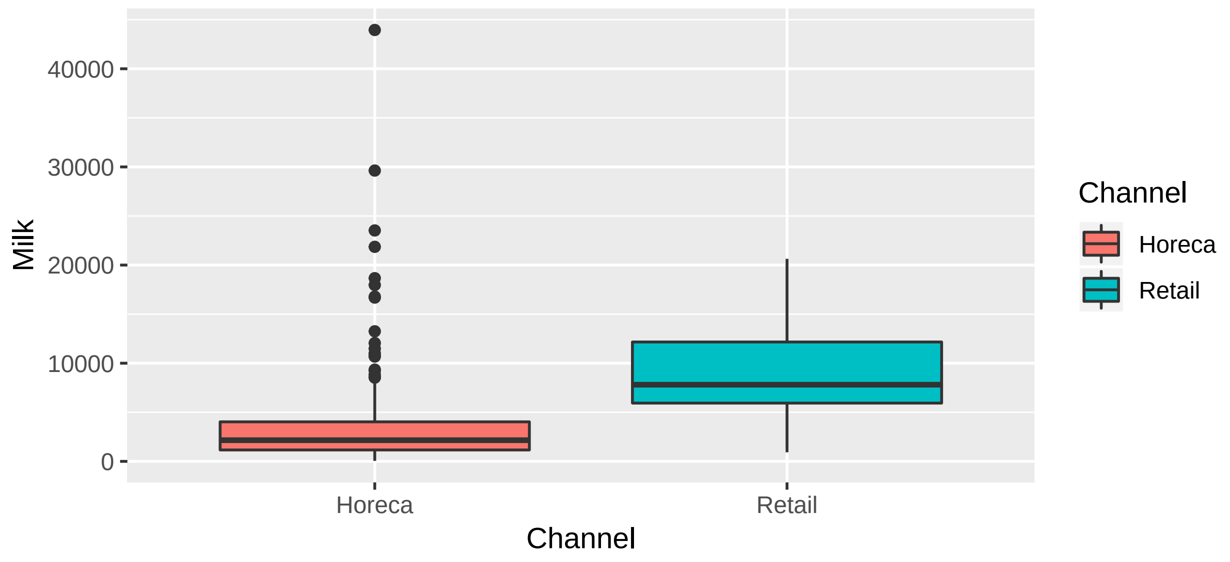
Since the algorithm is based on measuring distances (e.g. Euclidean), this implies that **all variables must be continuous** and the approach can be severely **affected by outliers**. So, we should check if outliers are present. We will check only numerical variables that will be used for clustering.

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

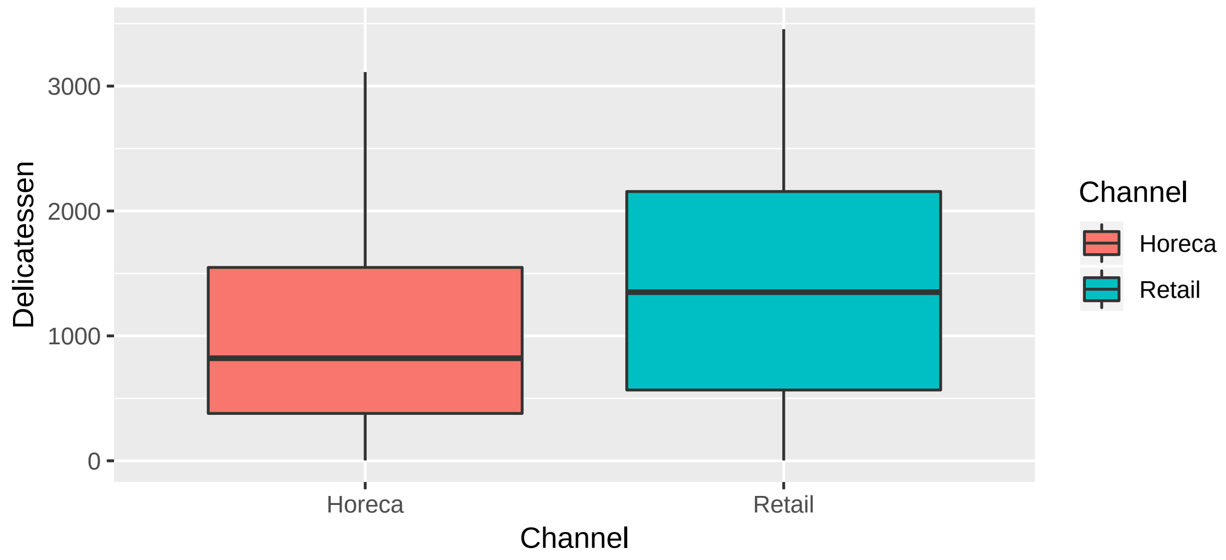
```
# plot boxplots for all numeric variables
ggplot(customers.data, aes(x=Channel, y=Grocery, fill=Channel)) + geom_boxplot()
```



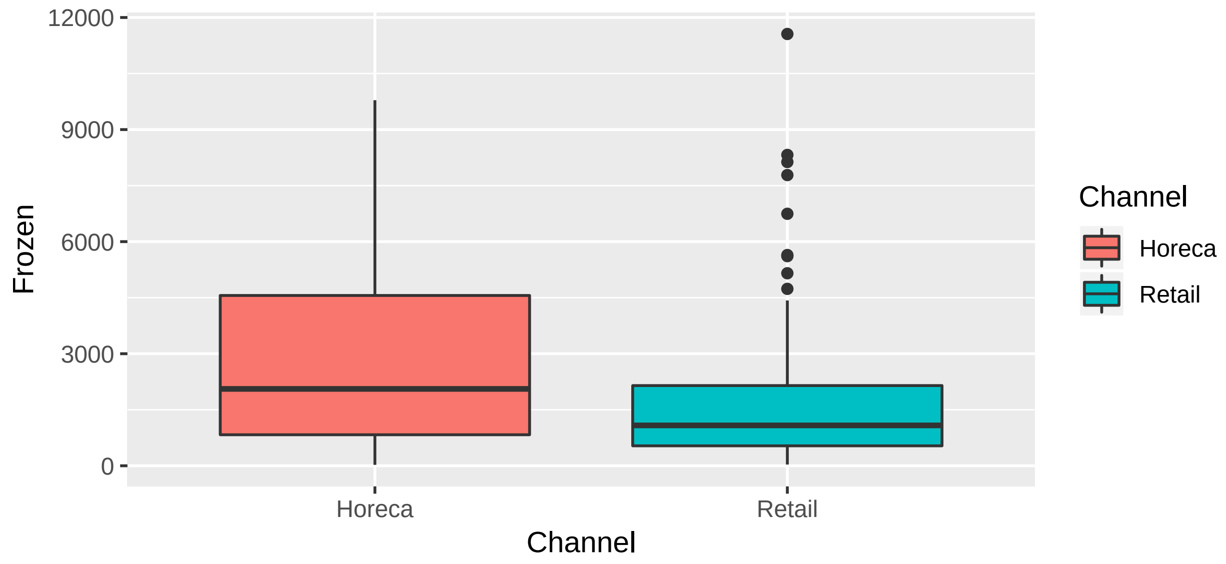
```r
ggplot(customers.data, aes(x=Channel, y=Milk, fill=Channel)) + geom_boxplot()
```



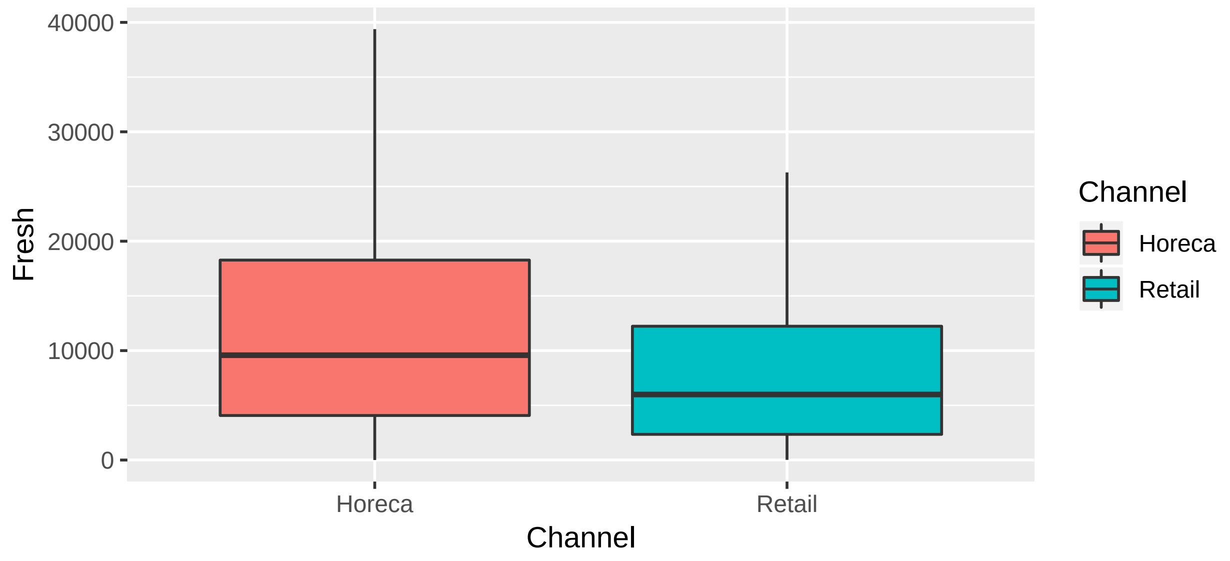
```r
ggplot(customers.data, aes(x=Channel, y=Delicatessen, fill=Channel)) + geom_boxplot()
```



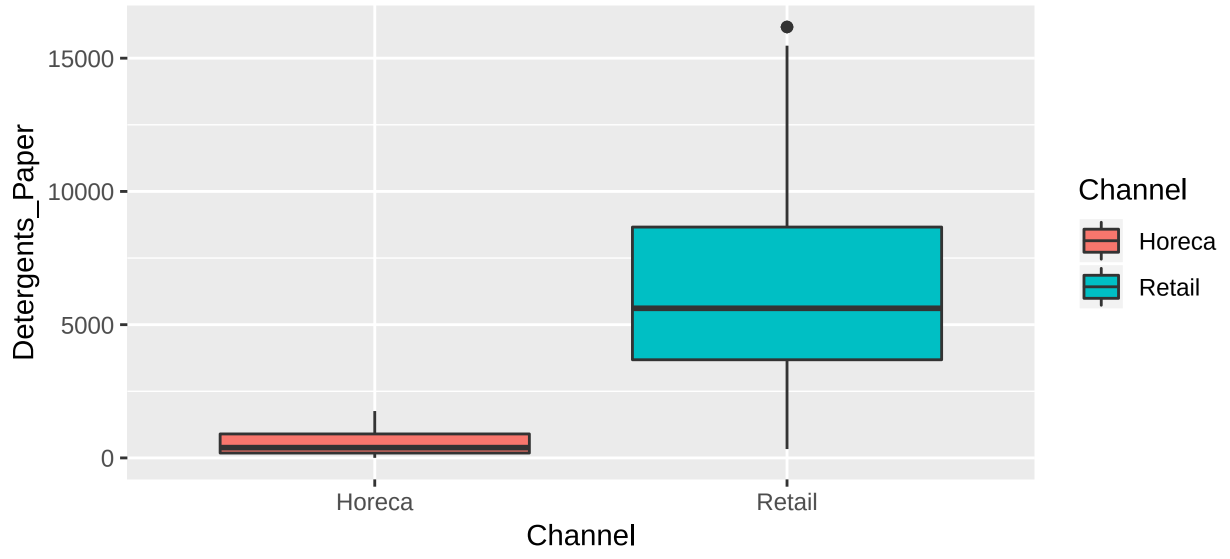
```
ggplot(customers.data, aes(x=Channel, y=Frozen, fill=Channel)) + geom_boxplot()
```



```
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()
```



It seems that only 2 variables have outliers.

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')
```

Let's focus first on the *retail.data* (**for the homework**: do the same with the *horeca.data*).

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

```
##    Channel            Region        Fresh             Milk
##  Horeca:  0   Lisbon     : 18   Min.   :   18   Min.   :  928
##  Retail:142   Oporto     : 19   1st Qu.: 2348   1st Qu.: 5938
##               Other_region:105   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.

```
# remove the Channel variable
retail.data <- retail.data[,-1]
```

Check which variables have outliers.

```
# check for 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 90th or 95th.

Let's start with the *Grocery* variable. We will extract the outliers and sort them by their values.

```
# sort all outliers of the Grocery variable
sort(boxplot.stats(retail.data$Grocery)$out)
```

```
## [1] 39694 45828 55571 59598 67298 92780
```

Now, we examine the 90th, 95th, ... percentile.

```
# examine percentiles of the Grocery variable higher than the 90th percentile
quantile(retail.data$Grocery, probs = seq(from = 0.9, to = 1, by = 0.025))
```

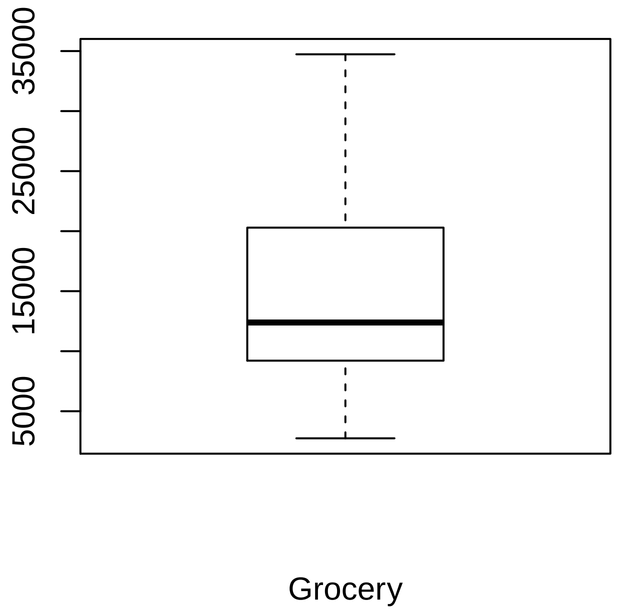
```
##      90%    92.5%      95%    97.5%     100%
## 28373.00 31004.18 34731.70 50455.93 92780.00
```

The 95th percentile seems to be a good cutting point.

```
# store the 95th percentile of the Grocery variable to a new variable
new.max <- as.numeric(quantile(retail.data$Grocery, probs = 0.95))

# to all outliers of the Grocery variable assing the value of the 95th percentile
retail.data$Grocery[retail.data$Grocery > new.max] <- new.max
```

By drawing the box plot for the *Grocery* variable again we will see that there are no outliers present anymore.

```
# print the boxplot for the Grocery variable
boxplot(retail.data$Grocery, xlab='Grocery')
```



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

```r
# sort all outliers of the Frozen variable
sort(boxplot.stats(retail.data$Frozen)$out)
```

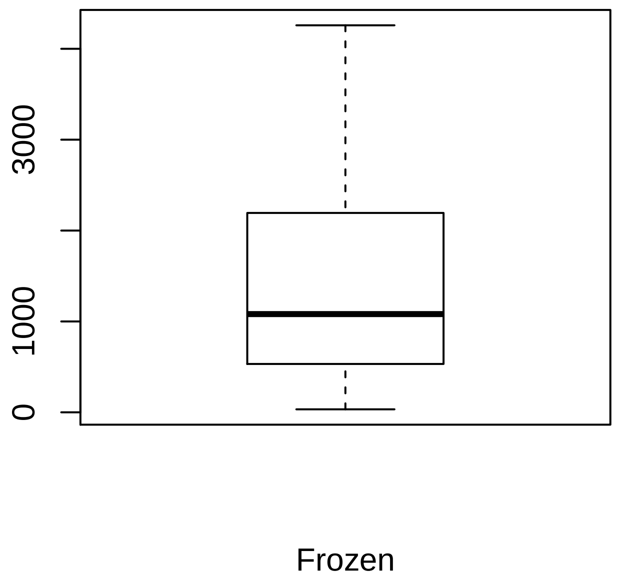
```
## [1]  4736  5154  5612  5641  6746  7782  8132  8321 11559
```

```r
# examine percentiles of the Frozen variable higher than the 90th percentile
quantile(retail.data$Frozen, probs = c(seq(0.9, 1, 0.025)))
```

```
##      90%     92.5%      95%     97.5%      100%
##  3519.50  4258.55  5133.10  7238.10 11559.00
```

Setting values to the 92.5th percentile seem to be a good approach.

```r
# store the 92.5th percentile of the Frozen variable to a new variable
new.max <- as.numeric(quantile(retail.data$Frozen, probs = 0.925))

# to all outliers of the Frozen variable assing the value of the 92.5th percentile
retail.data$Frozen[retail.data$Frozen > new.max] <- new.max

# print a boxplot for the Frozen variable
boxplot(retail.data$Frozen, xlab='Frozen')
```



Finally, examine the *retail.data* after the transformations.

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

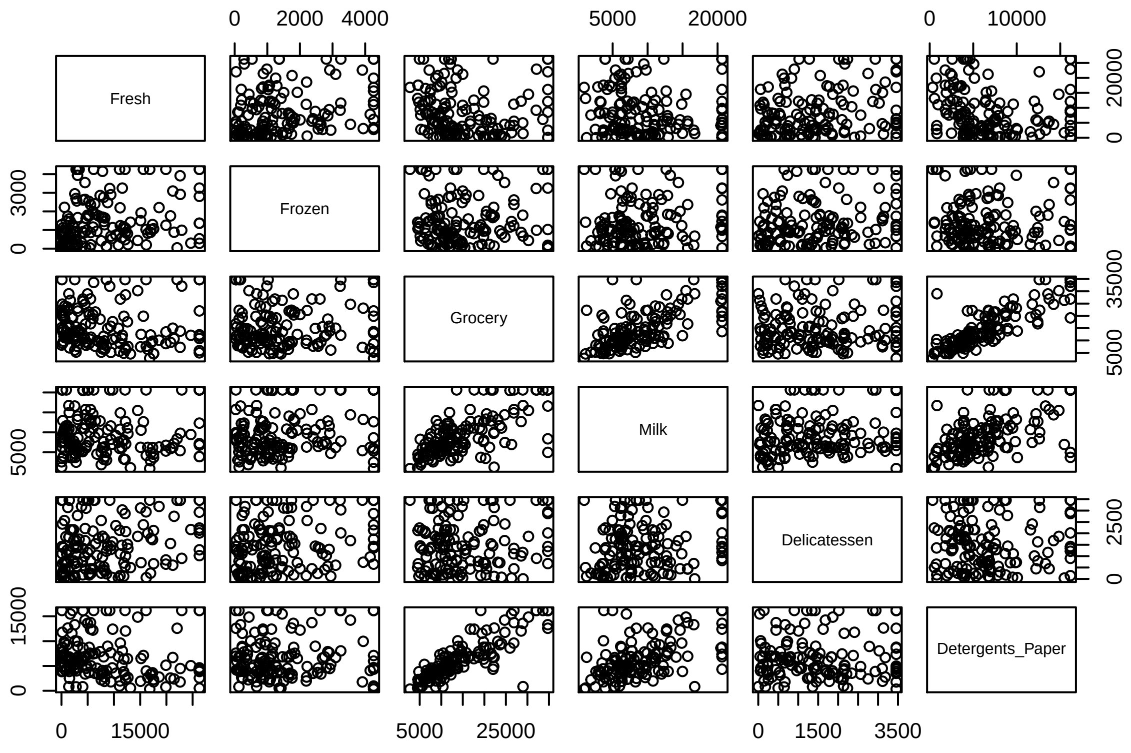
```
##          Region       Fresh            Milk          Grocery
##  Lisbon      : 18   Min.   :   18   Min.   :  928   Min.   : 2743
##  Oporto      : 19   1st Qu.: 2348   1st Qu.: 5938   1st Qu.: 9245
##  Other_region:105   Median : 5994   Median : 7812   Median :12390
##                     Mean   : 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   :1471.9   Mean   : 6650    Mean   :1485.2
##  3rd Qu.:2146.8   3rd Qu.: 8662    3rd Qu.:2156.0
##  Max.   :4258.6   Max.   :16171    Max.   :3455.6
```

3 Clustering with 2 Features

Let's choose only two features for this initial, simple clustering. For the sake of an example, let's pick Frozen and Milk variables.

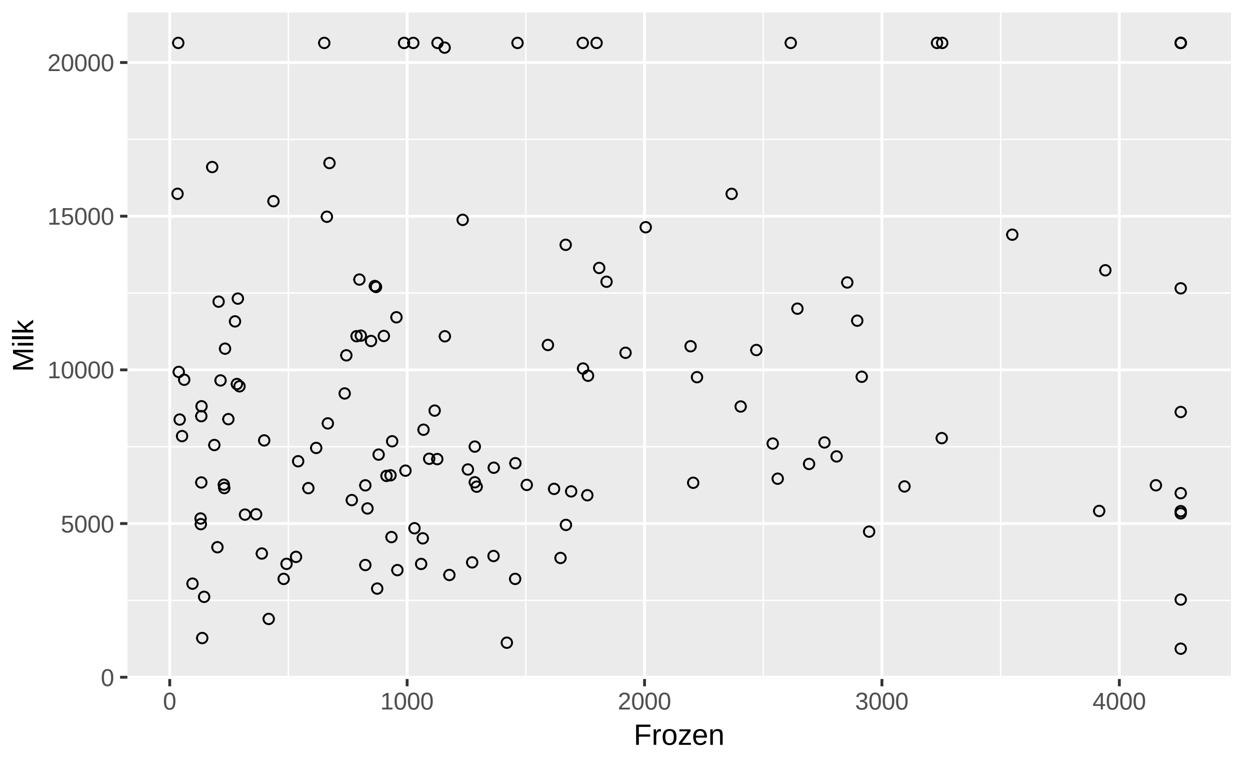
NOTE: We will choose only two features in order to demonstrate the algorithm. But in reality, all numerical variables (that make sense) should be taken into consideration.

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



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.

```r
# plot the scatterplot for the variablesFrozen and Milk
ggplot(data=retail.data, aes(x=Frozen, y=Milk)) +
  geom_point(shape=1)
```



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.

```r
# 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   :1471.9   Mean   : 9421
##  3rd Qu.:2146.8   3rd Qu.:12163
##  Max.   :4258.6   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 q 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(feature, na.rm = T)))
}
```

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

```
# 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.:0.1186   1st Qu.:0.2542
##  Median :0.2480   Median :0.3493
##  Mean   :0.3405   Mean   :0.4309
##  3rd Qu.:0.5002   3rd Qu.:0.5700
##  Max.   :1.0000   Max.   :1.0000
```

Run the K Means algorithm, specifying, for example, 4 centers. *'iter.max'* defines the maximum number of iterations. This overcomes the problem of situations where there is a slow convergence. This clustering approach can be sensitive to the initial selection of centroids. *'nstart'* option attempts multiple initial configurations and reports on the best one. Afterward, we inspect the results.

```
# set the seed
set.seed(3108)

# 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 10, 35, 72, 25
##
## Cluster means:
##      Frozen      Milk
## 1 0.8887541 0.8903003
## 2 0.2417335 0.7118702
## 3 0.1734192 0.2642975
## 4 0.7407727 0.3335116
##
## Clustering vector:
##   1   2   3   5   6   7   8  10  11  12  13  14  15  17  19  21  24  25
##   3   3   4   4   3   3   3   2   4   3   2   4   3   3   4   3   1   4
##  26  29  36  38  39  43  44  45  46  47  48  49  50  53  54  57  58  61
##   3   2   3   2   2   3   2   3   2   2   1   3   2   3   3   1   3   3
##  62  63  64  66  68  74  75  78  82  83  85  86  87  93  95  97 101 102
##   1   4   4   2   3   4   3   2   3   3   3   2   2   1   2   3   4   2
## 103 107 108 109 110 112 124 128 146 156 157 159 160 161 164 165 166 167
##   4   3   4   3   2   2   4   3   3   3   3   3   3   3   2   4   1   3
## 171 172 174 176 189 190 194 198 201 202 206 208 210 212 215 217 219 224
##   3   2   3   3   4   2   3   3   1   1   2   4   2   1   3   2   3   4
## 227 231 246 252 265 267 269 280 282 294 296 298 299 301 302 303 304 305
##   3   4   3   1   3   4   3   3   3   2   3   3   3   3   2   3   3   3
## 306 307 310 313 316 320 332 334 335 336 341 342 344 347 348 350 352 354
##   2   2   2   3   2   2   2   3   4   3   3   3   3   4   4   2   2   3
## 358 366 371 374 377 380 397 408 409 416 417 419 422 424 425 438
##   3   3   3   3   4   3   4   4   3   3   2   3   3   3   3   2
##
## Within cluster sum of squares by cluster:
## [1] 0.4615838 1.9864552 2.0309120 1.3114720
##  (between_SS / total_SS =  74.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
## [5] "tot.withinss" "betweenss"    "size"         "iter"
## [9] "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;
- **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.

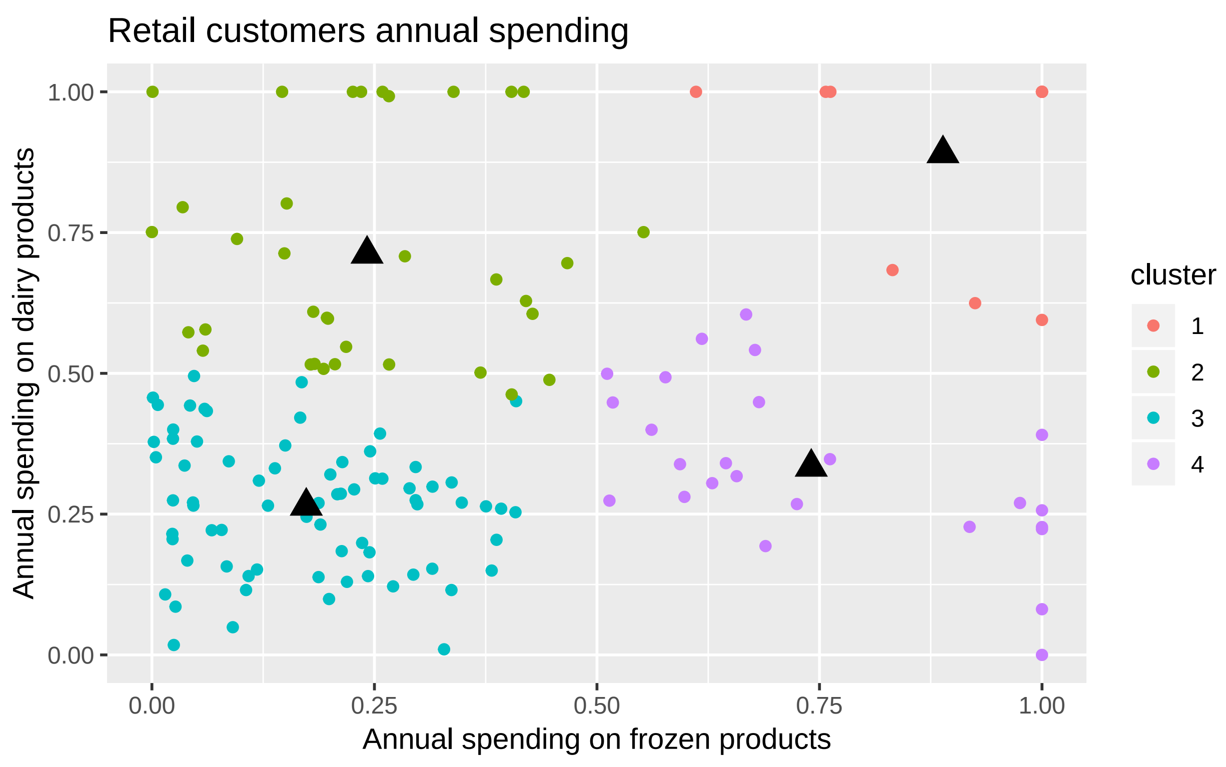
Add the vector of clusters to the data frame as a new variable *'cluster'* and plot it.

```r
# 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 0.04283466 0.4428231       3
## 2 0.40917750 0.4506365       3
## 3 0.56134704 0.3997991       4
## 5 0.91869697 0.2273984       4
## 6 0.14980298 0.3719451       3
## 7 0.10578505 0.1152213       3
```

Let's plot the data and new centroids.

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



The plot allows us to visually inspect the clusters, but it is better to have a more systematic approach to judging the quality of clustering and selecting the best value for K.

4 Selecting the Best Value for K

4.1 Selection based on the Elbow Method

Instead of guessing the correct value for K, we can take a more systematic approach to choose the 'right' K. 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).

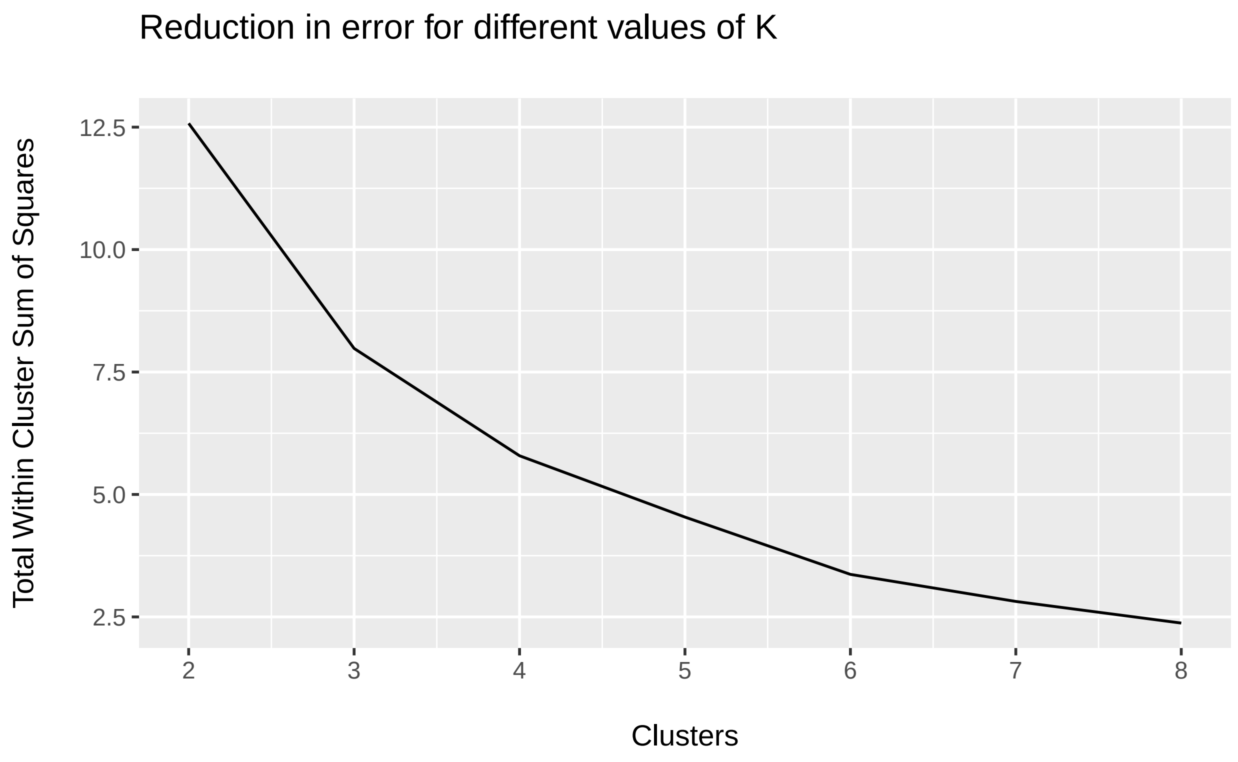
Let us run the K-means for different values for K (from 2 to 8). We will compute the metric required for the Elbow method (tot.withinss). Along the way, we'll also compute the other metric: *ratio of between_SS and total_SS*.

```
# create an empty data frame
eval.metrics.2var <- data.frame()

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

# run kmeans for all K values in the range 2:8
for (k in 2:8) {
  set.seed(3108)
  km.res <- kmeans(x=retail.data1.norm, centers=k, iter.max=20, nstart = 1000)
  # combine cluster number and the error measure, write to df
  eval.metrics.2var <- rbind(eval.metrics.2var,
                             c(k, km.res$tot.withinss, km.res$betweenss/km.res$totss))
}

# update the column names
names(eval.metrics.2var) <- c("cluster", "tot.within.ss", "ratio")

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

```
##   cluster tot.within.ss     ratio
## 1       2     12.577765 0.4350165
## 2       3      7.981614 0.6414721
## 3       4      5.790423 0.7398987
## 4       5      4.538501 0.7961340
## 5       6      3.368527 0.8486884
## 6       7      2.815509 0.8735295
## 7       8      2.374140 0.8933555
```

Draw the Elbow plot.

```
# plot the line chart for cluster vs. tot.within.ss
ggplot(data=eval.metrics.2var, aes(x=cluster, y=tot.within.ss)) +
  geom_line() +
  ggtitle("Reduction in error for different values of K\n") +
  xlab("\nClusters") +
  ylab("Total Within Cluster Sum of Squares\n") +
  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 file with our utility functions (code is given at the end of the script) and call our custom function *compute.difference*.

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

# calculate the ratio difference for different K values (from 2 to 8)
data.frame(K=2:8,
           tot.within.ss.delta=compute.difference(eval.metrics.2var$tot.within.ss),
           ratio.delta=compute.difference(eval.metrics.2var$ratio))
```

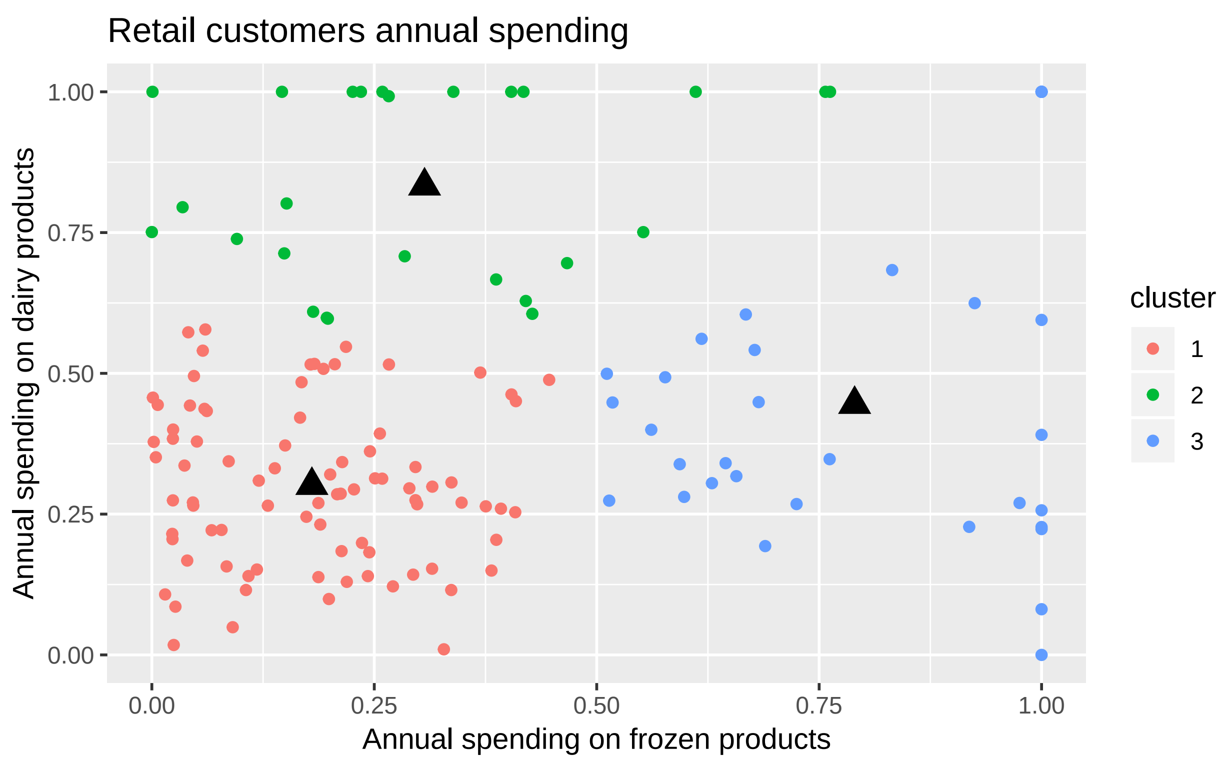
```
##   K tot.within.ss.delta ratio.delta
## 1 2                  NA          NA
## 2 3           4.5961507  0.20645554
## 3 4           2.1911907  0.09842659
## 4 5           1.2519218  0.05623536
## 5 6           1.1699738  0.05255432
## 6 7           0.5530184  0.02484116
## 7 8           0.4413690  0.01982595
```

As we've already examined the solution with 4 clusters, let's also examine the solution with K=3.

```r
# set the seed value
set.seed(3108)

# 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.

```r
# plot the clusters along with their centroids
ggplot(data=retail.data1.norm, aes(x=Frozen, y=Milk, colour=cluster)) +
    geom_point() +
    xlab("Annual spending on frozen products") +
    ylab("Annual spending on dairy products") +
    ggtitle("Retail customers annual spending") +
    geom_point(data=as.data.frame(simple.3k$centers),
                     colour="black",size=4, shape=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 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                84                 26                32
## 3     Frozen   793.04 (526.94)   1328.08 (879.37)  3370.69 (797.97)
## 4       Milk 6862.69 (2782.19) 17342.03 (3268.35) 9699.39 (5204.43)
```

The solution with **K=4** seems to be better as features are less dispersed than here (e.g. the *Milk* variable in cluster 3, and *Frozen* variable in cluster 2 have very high dispersion). The solution with **K=4** resolves this (check the plot and examine cluster centers).

5 Clustering with All Numeric Features

Since the 6 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(2:7)], 2, normalize.feature))
```

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

```
##      Fresh               Milk           Grocery           Frozen
##  Min.   :0.00000   Min.   :0.0000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.08869   1st Qu.:0.2542   1st Qu.:0.2033   1st Qu.:0.1186
##  Median :0.22747   Median :0.3493   Median :0.3016   Median :0.2480
##  Mean   :0.32138   Mean   :0.4309   Mean   :0.3906   Mean   :0.3405
##  3rd Qu.:0.46487   3rd Qu.:0.5700   3rd Qu.:0.5452   3rd Qu.:0.5002
##  Max.   :1.00000   Max.   :1.0000   Max.   :1.0000   Max.   :1.0000
##  Detergents_Paper  Delicatessen
##  Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.2116   1st Qu.:0.1633
##  Median :0.3335   Median :0.3901
##  Mean   :0.3989   Mean   :0.4293
##  3rd Qu.:0.5260   3rd Qu.:0.6236
##  Max.   :1.0000   Max.   :1.0000
```

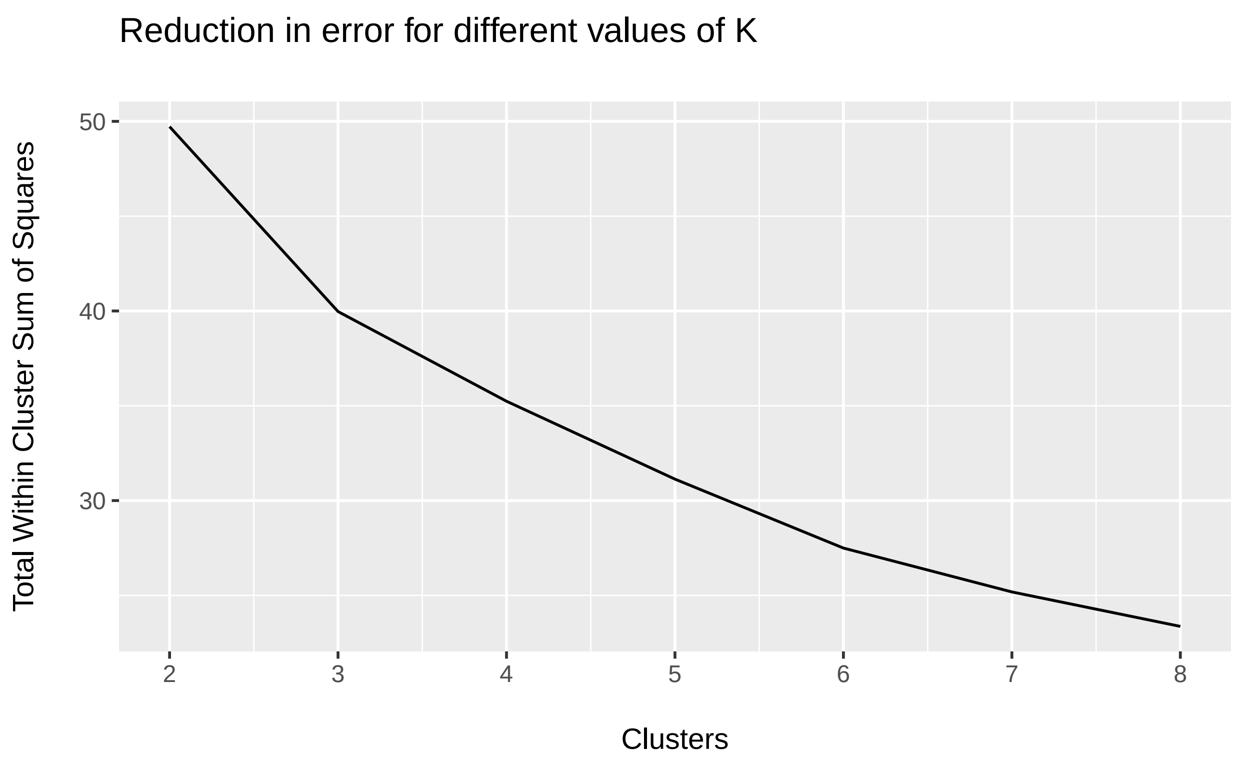
Instead of guessing K, we'll right away use the Elbow method to find the optimal value for K.

```r
# create an empty data frame
eval.metrics.6var <- data.frame()

# run kmeans for all K values in the range 2:8
for (k in 2:8) {
  set.seed(3108)
  km.res <- kmeans(x=retail.norm, centers=k, iter.max=20, nstart = 1000)
  # combine cluster number and the error measure, write to df
  eval.metrics.6var <- rbind(eval.metrics.6var,
                             c(k, km.res$tot.withinss, km.res$betweenss/km.res$totss))
}

# update the column names
names(eval.metrics.6var) <- c("cluster", "tot.within.ss", "ratio")
eval.metrics.6var
```

```
##   cluster tot.within.ss     ratio
## 1       2      49.72071 0.2667494
## 2       3      39.96778 0.4105797
## 3       4      35.23843 0.4803252
## 4       5      31.13015 0.5409116
## 5       6      27.49447 0.5945284
## 6       7      25.18087 0.6286479
## 7       8      23.36426 0.6554381
```

Draw the Elbow plot.

```r
# plot the clusters along with their centroids
ggplot(data=eval.metrics.6var, aes(x=cluster, y=tot.within.ss)) +
  geom_line() +
  ggtitle("Reduction in error for different values of K\n") +
  xlab("\nClusters") +
  ylab("Total Within Cluster Sum of Squares\n") +
  scale_x_continuous(breaks=seq(from=0, to=8, by=1))
```



This time it seems that the solution with 3 clusters might be the best.

As previously done, we'll also look for the best value for K by examining the difference between every two subsequent values of tot.within.ss and ratio (of *between.SS* and *total.SS*).

```
# calculate the ratio difference for different K values (from 2 to 8)
data.frame(k=c(2:8),
           tot.within.ss.delta=compute.difference(eval.metrics.6var$tot.within.ss),
           ration.delta=compute.difference(eval.metrics.6var$ratio))
```

```
##   k tot.within.ss.delta ration.delta
## 1 2                  NA           NA
## 2 3            9.752931   0.14383025
## 3 4            4.729349   0.06974554
## 4 5            4.108280   0.06058640
## 5 6            3.635680   0.05361679
## 6 7            2.313597   0.03411951
## 7 8            1.816606   0.02679019
```

This again suggests 3 clusters as the best option.

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

```
# set the seed
set.seed(3108)

# run the clustering for 3 clusters, iter.max=20, nstart=1000
retail.3k.6var <- 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(2:7)],
                           clusters = retail.3k.6var$cluster,
                           cl.num = 3)
sum.stats
```

```
##        attributes   Mean (SD)   Mean (SD)   Mean (SD)
## 1         cluster           1           2           3
## 2            freq          43          25          74
## 3           Fresh 0.55 (0.31)  0.36 (0.3) 0.18 (0.17)
## 4            Milk 0.32 (0.18) 0.83 (0.23) 0.36 (0.19)
## 5         Grocery 0.22 (0.13) 0.81 (0.18) 0.34 (0.18)
## 6          Frozen  0.5 (0.32) 0.44 (0.34) 0.21 (0.19)
## 7 Detergents_Paper  0.2 (0.12) 0.83 (0.18) 0.37 (0.19)
## 8     Delicatessen 0.63 (0.27) 0.59 (0.33) 0.26 (0.22)
```

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

```
## function that computes the difference between two consecutive values
compute.difference <- function(values) {
  dif <- vector(mode = "numeric", length = length(values))
  dif[1] <- NA
  for(i in 1:(length(values)-1)) {
    dif[i+1] <- abs(values[i+1] - values[i])
  }
  dif
}

## function that provides summary statistics about 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
}
```