Exercise1

Riccardo Petrella

1

We upload and explore the data

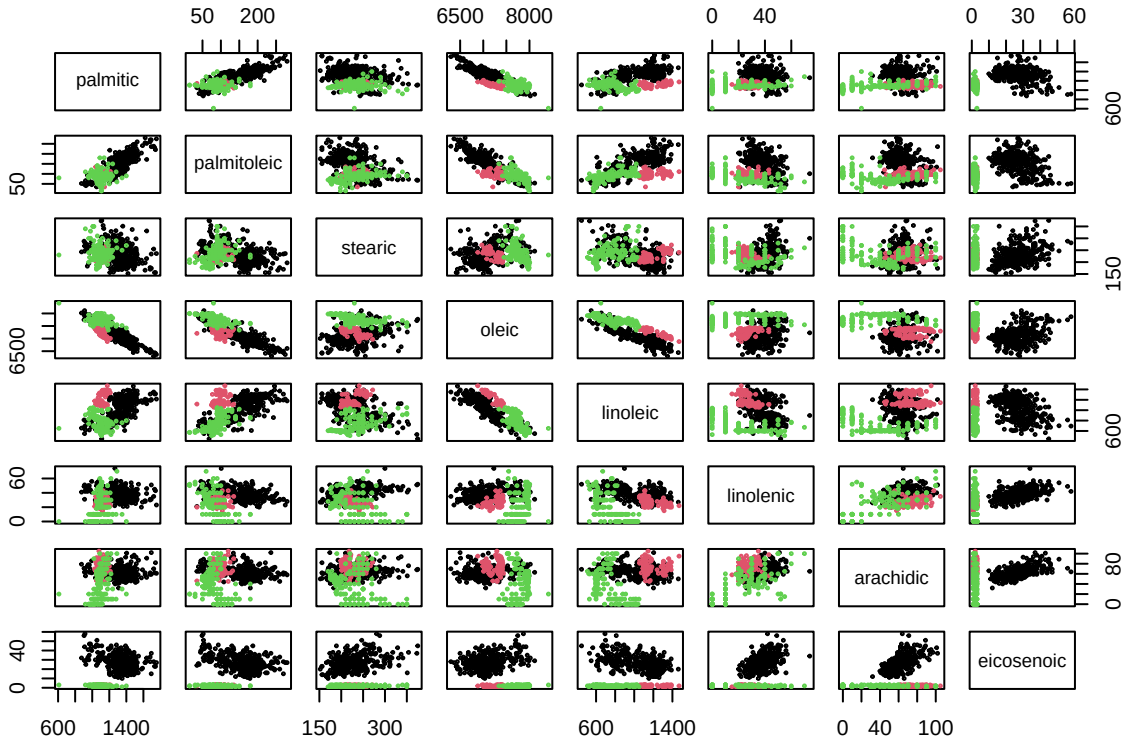
```
library(pdfCluster)
```

```
## pdfCluster 1.0-4
```

```
data(oliveoil)

str(oliveoil)
```

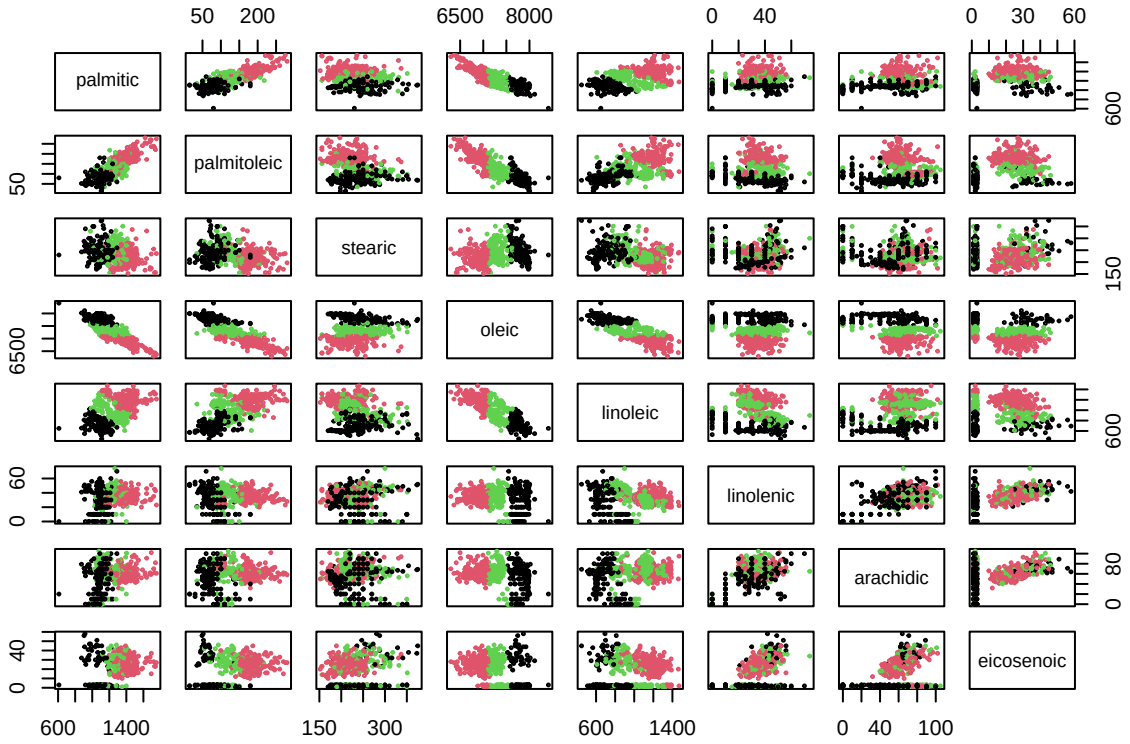
```
## 'data.frame':    572 obs. of  10 variables:
##  $ macro.area : Factor w/ 3 levels "South","Sardinia",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ region     : Factor w/ 9 levels "Apulia.north",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ palmitic   : int  1075 1088 911 966 1051 911 922 1100 1082 1037 ...
##  $ palmitoleic: int  75 73 54 57 67 49 66 61 60 55 ...
##  $ stearic    : int  226 224 246 240 259 268 264 235 239 213 ...
##  $ oleic      : int  7823 7709 8113 7952 7771 7924 7990 7728 7745 7944 ...
##  $ linoleic   : int  672 781 549 619 672 678 618 734 709 633 ...
##  $ linolenic  : int  36 31 31 50 50 51 49 39 46 26 ...
##  $ arachidic  : int  60 61 63 78 80 70 56 64 83 52 ...
##  $ eicosenoic : int  29 29 29 35 46 44 29 35 33 30 ...
```

```
# The chemical variables:
olive <- oliveoil[,3:10]

# Plot with the "real" clusters :
pairs(olive,cex=0.3,col=oliveoil[,1])
```



k-means with k=3 (unscaled)

```
library(cluster)

set.seed(1234)
k_olive <- kmeans(olive, 3, nstart = 100)
plot(olive, cex = 0.3, col = k_olive$cluster)
```




```
oliveoil$macro.area <- as.numeric(oliveoil$macro.area)
```

```
k_olive$tot.withinss
```

```
## [1] 30493566
```

Comparison with true clusters

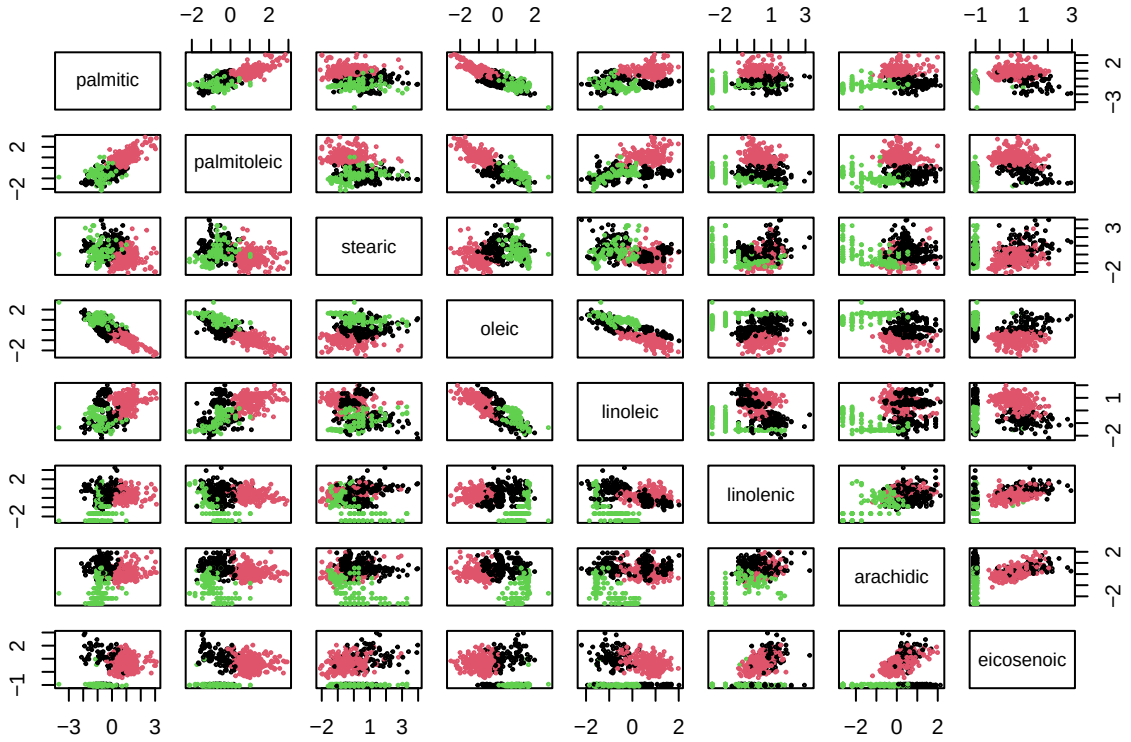
```
table(oliveoil$macro.area, k_olive$cluster)
```

```
##
##        1   2   3
##   1   42 190  91
##   2    0  22  76
##   3  134   0  17
```

k means with k=3 (scaled)

```
set.seed(1234)
solive <- scale(olive)
```

```r
solive <- as.data.frame(solive)

k_solive <- kmeans(solive, 3, nstart = 100)
plot(solive, cex = 0.3, col = k_solive$cluster)
```




```r
k_solive$tot.withinss
```

```
## [1] 2320.024
```

Comparision with true clusters

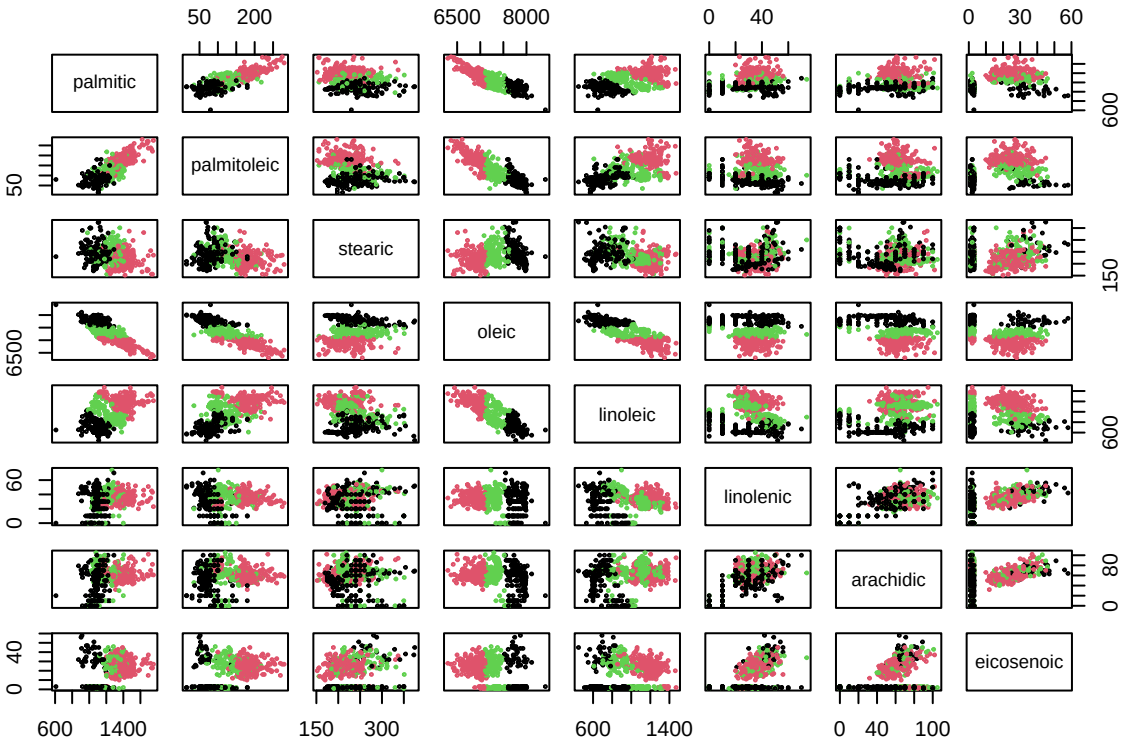
```r
table(oliveoil$macro.area, k_solive$cluster)
```

```
##
##        1   2   3
##   1  102 219   2
##   2   98   0   0
##   3   30   0 121
```

There is a huge difference in terms of the values of S. Scaling the data yields a dramatic reduction in the within-clusters variance. Also K-means may not be working correctly since as shown in the scatter-plots, the clusters are higly helliptical.

k means with k=9 (unscaled)

```
set.seed(1234)
k_olive9 <- kmeans(olive, 9, nstart = 100)
plot(olive, cex = 0.3, col = k_olive$cluster)
```




```
oliveoil$region <- as.numeric(oliveoil$region)

k_olive9$tot.withinss
```

```
## [1] 9336608
```

Comparision with true clusters

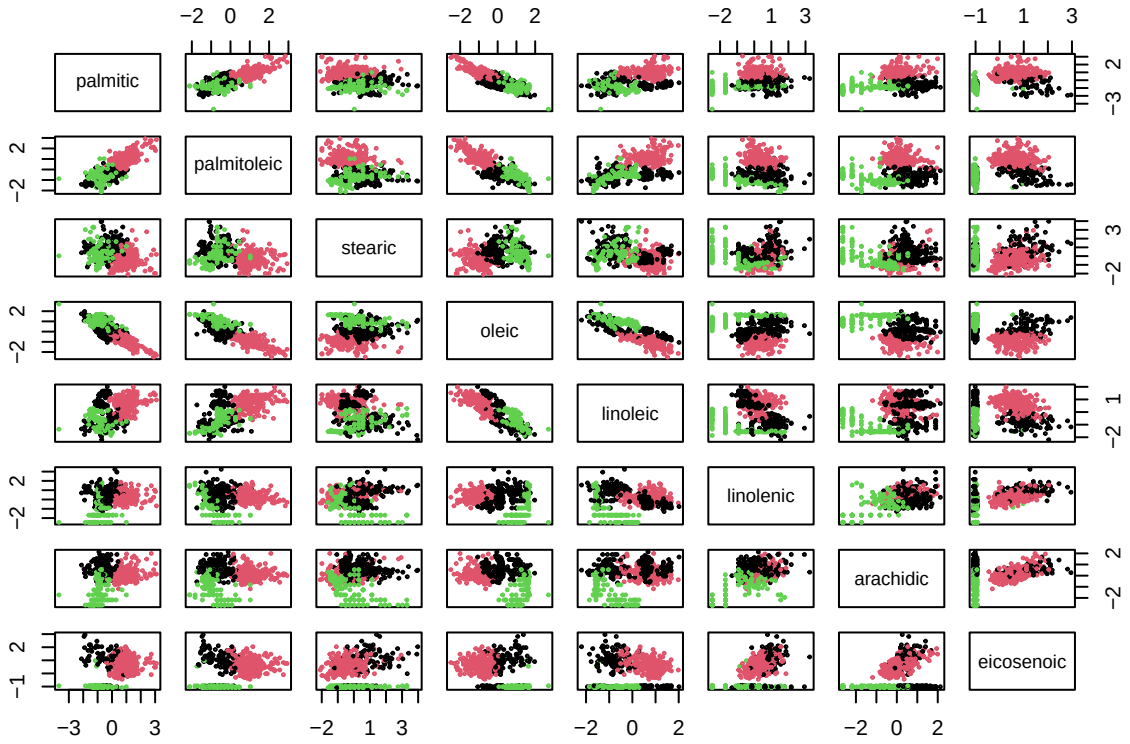
```
table(oliveoil$region, k_olive$cluster)
```

```
##
##        1   2   3
##   1   24   0   1
##   2    4   1  51
##   3    0 180  26
```

```
##    4  14   9  13
##    5   0   0  65
##    6   0  22  11
##    7  46   0   4
##    8  37   0  13
##    9  51   0   0
```

k means with k=9 (scaled)

```
set.seed(1234)

solive <- scale(olive)
solive <- as.data.frame(solive)

k_solive9 <- kmeans(solive, 9, nstart = 100)
plot(solive, cex = 0.3, col = k_solive$cluster)
```




```
k_solive9$tot.withinss
```

```
## [1] 1020.332
```

Comparision with true clusters

```
table(oliveoil$region, k_solive9$cluster)
```

```
##
##        1   2   3   4   5   6   7   8   9
##   1    0   1  22   0   2   0   0   0   0
##   2    0   0   0  23  32   0   1   0   0
##   3    0   0   0   1  12   0   0  49 144
##   4    0   0   6  12  16   0   0   2   0
##   5    0   0   0   0   0  65   0   0   0
##   6    0   0   0   0   0  33   0   0   0
##   7   10   7   0   0   0   0  33   0   0
##   8   50   0   0   0   0   0   0   0   0
##   9    0  50   0   0   0   0   1   0   0
```

We can clearly see that the best solution is the k-means on scaled variables with k = 9. The objective function reaches its smallest value with this settings. This outcome was easy to predict since we certainly know that S decreases if we increase k. Also by scaling data, we reduce the huge differences in the variables' ranges, leading to much smaller values of S.

2

From the probabilistic theory behind K-Means, we know that: when we maximize the log-likelihood of the joint density $(X_1, X_2, ..., X_n)$ for the centroids $a_1, ..., a_k$, the clusters $\gamma(1), ..., \gamma(n)$ do not depend on q because:

$$\log f\left(X_1, X_2, \ldots, X_n\right) = \sum_{i=1}^{n} \left(-\log\left(\sqrt{2\pi \det\left(qI_p\right)}\right) - \frac{1}{2q}\left(X_i - \mathrm{a}_{\gamma(i)}\right)^{\top}\left(X_i - \mathrm{a}_{\gamma(i)}\right)\right)$$

is equivalent to minimize

$$\sum_{i=1}^{n}\left(X_i - \mathrm{a}_{(i)}\right)^{\top}\left(X_i - \mathrm{a}_{\gamma(i)}\right) = \sum_{i=1}^{n}\left\|X_i - \mathrm{a}_{\gamma(i)}\right\|^2.$$

In the last equation, q does not appear. Thus,

$$C(i) = \arg\min \|X_i - m_j\| = C^*(i), \quad i \in N_n$$
$$j \in \{1, \ldots, k\}$$

Now, by multiplaying each X_i by q, we change the scale of the variables. From the theory we know that K-Means is not scale-invariant. Hence, if $q \neq 1$:

$$\hat{m}_j^{k_m *} = \frac{1}{n_j}\sum_{C(i)=j} q \cdot x_i \neq \hat{m}_j^{k_m} = \frac{1}{n_j}\sum_{(i)=j} x_i$$

Therefore, q makes these centroids differ.

3

We upload the dataset and explore its structure

```
library(fpc)
boston <- read.table("Boston.dat",header=TRUE)

mboston <- as.matrix(boston)

sum(is.na(mboston))
```

```
## [1] 0
```

```
str(mboston)
```

```
##  num [1:506, 1:14] 0.00632 0.02731 0.02729 0.03237 0.06905 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:14] "crim" "zn" "indus" "chas" ...
```

Before applying the k-means, it is a good practice to remove factorial variables.

```
#removal of binary variable and factorial variables
mboston <- mboston[,-4]

mboston <- mboston[,-8]
```

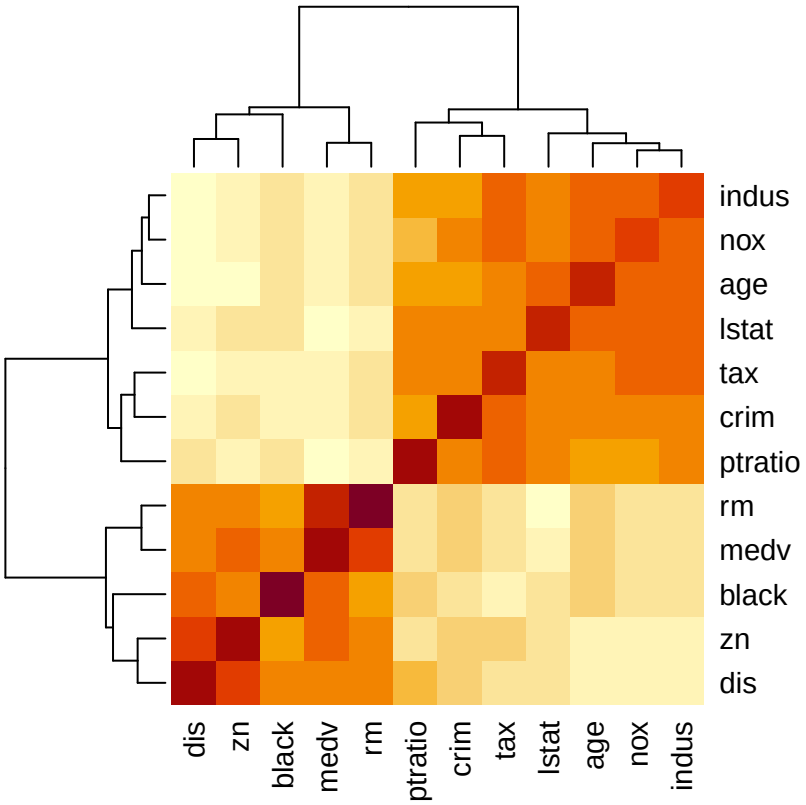
We now perform some data visualization

```
# correlation
cor(mboston)
```

```
##              crim         zn      indus        nox         rm        age
## crim     1.0000000 -0.2004692  0.4065834  0.4209717 -0.2192467  0.3527343
## zn      -0.2004692  1.0000000 -0.5338282 -0.5166037  0.3119906 -0.5695373
## indus    0.4065834 -0.5338282  1.0000000  0.7636514 -0.3916759  0.6447785
## nox      0.4209717 -0.5166037  0.7636514  1.0000000 -0.3021882  0.7314701
## rm      -0.2192467  0.3119906 -0.3916759 -0.3021882  1.0000000 -0.2402649
## age      0.3527343 -0.5695373  0.6447785  0.7314701 -0.2402649  1.0000000
## dis     -0.3796701  0.6644082 -0.7080270 -0.7692301  0.2052462 -0.7478805
## tax      0.5827643 -0.3145633  0.7207602  0.6680232 -0.2920478  0.5064556
## ptratio  0.2899456 -0.3916785  0.3832476  0.1889327 -0.3555015  0.2615150
## black   -0.3850639  0.1755203 -0.3569765 -0.3800506  0.1280686 -0.2735340
## lstat    0.4556215 -0.4129946  0.6037997  0.5908789 -0.6138083  0.6023385
## medv    -0.3883046  0.3604453 -0.4837252 -0.4273208  0.6953599 -0.3769546
##                dis        tax    ptratio      black      lstat       medv
## crim    -0.3796701  0.5827643  0.2899456 -0.3850639  0.4556215 -0.3883046
## zn       0.6644082 -0.3145633 -0.3916785  0.1755203 -0.4129946  0.3604453
## indus   -0.7080270  0.7207602  0.3832476 -0.3569765  0.6037997 -0.4837252
## nox     -0.7692301  0.6680232  0.1889327 -0.3800506  0.5908789 -0.4273208
## rm       0.2052462 -0.2920478 -0.3555015  0.1280686 -0.6138083  0.6953599
```

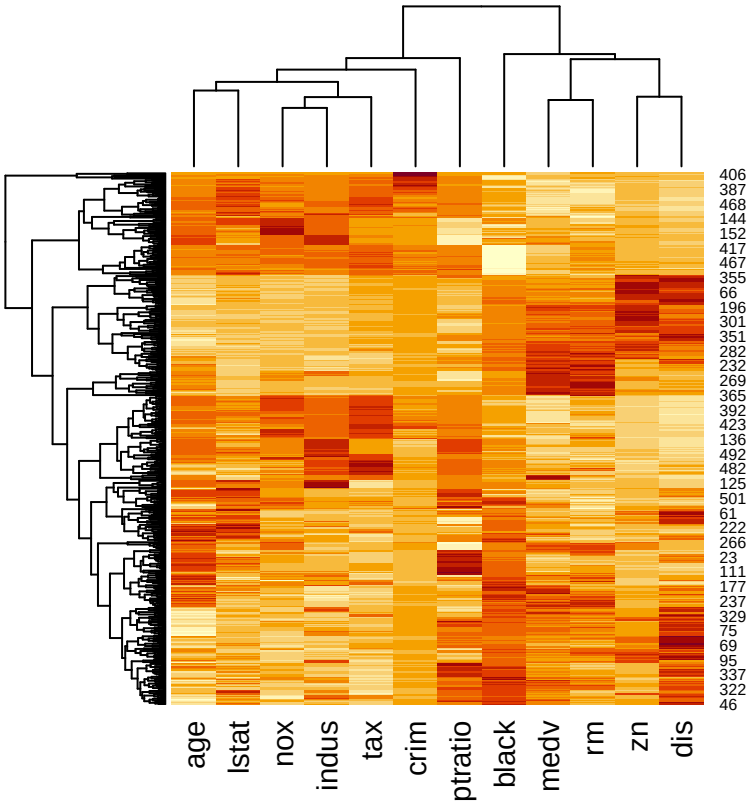
```
## age     -0.7478805  0.5064556  0.2615150 -0.2735340  0.6023385 -0.3769546
## dis      1.0000000 -0.5344316 -0.2324705  0.2915117 -0.4969958  0.2499287
## tax     -0.5344316  1.0000000  0.4608530 -0.4418080  0.5439934 -0.4685359
## ptratio -0.2324705  0.4608530  1.0000000 -0.1773833  0.3740443 -0.5077867
## black    0.2915117 -0.4418080 -0.1773833  1.0000000 -0.3660869  0.3334608
## lstat   -0.4969958  0.5439934  0.3740443 -0.3660869  1.0000000 -0.7376627
## medv     0.2499287 -0.4685359 -0.5077867  0.3334608 -0.7376627  1.0000000
```

```r
heatmap(cor(mboston))
```



Let us scale the variables to reduce variances.

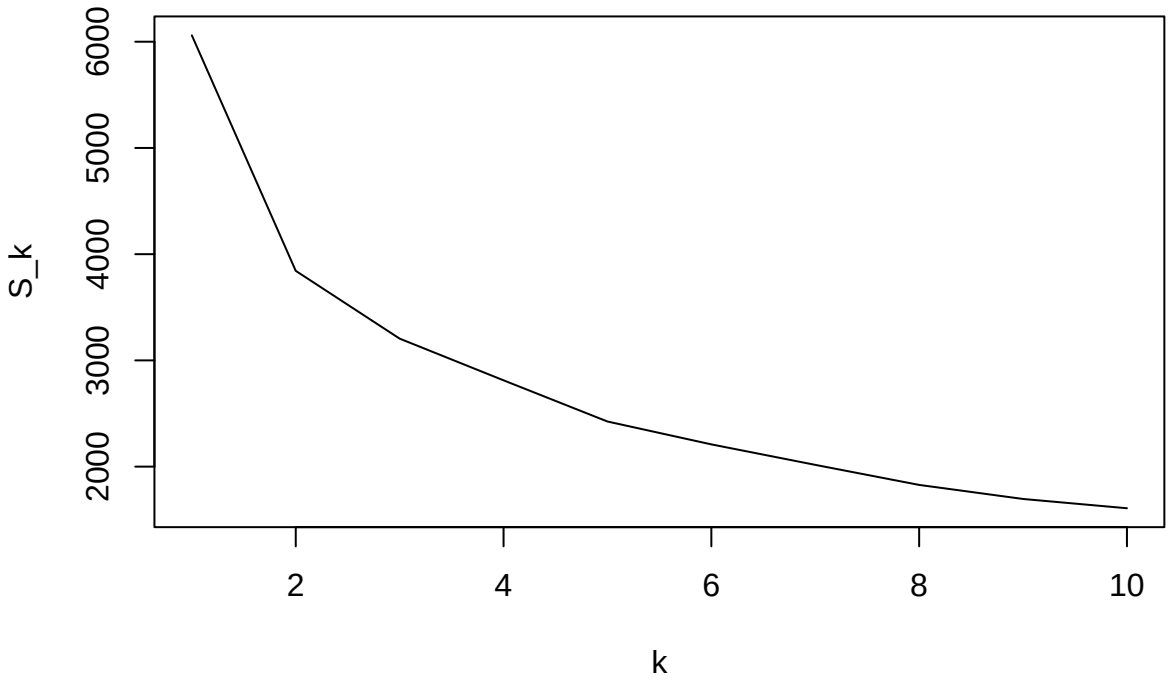
```r
# standardization
s_mboston <- scale(mboston)
heatmap(s_mboston)
```



K-Means

We apply the k-means with k as a list of 1,2,3,4,5,6,7,8,9,10 to better choose k

```
mboston_km <- kmeans(s_mboston, 4, nstart = 50 )

sk <- numeric(0)
kclusterings <- list()
for (k in 1:10){
  kclusterings[[k]] <- kmeans(s_mboston, k, nstart=100)
  sk[k] <- kclusterings[[k]]$tot.withinss
}

plot(1:10, sk, xlab="k",ylab="S_k",type="l")
```

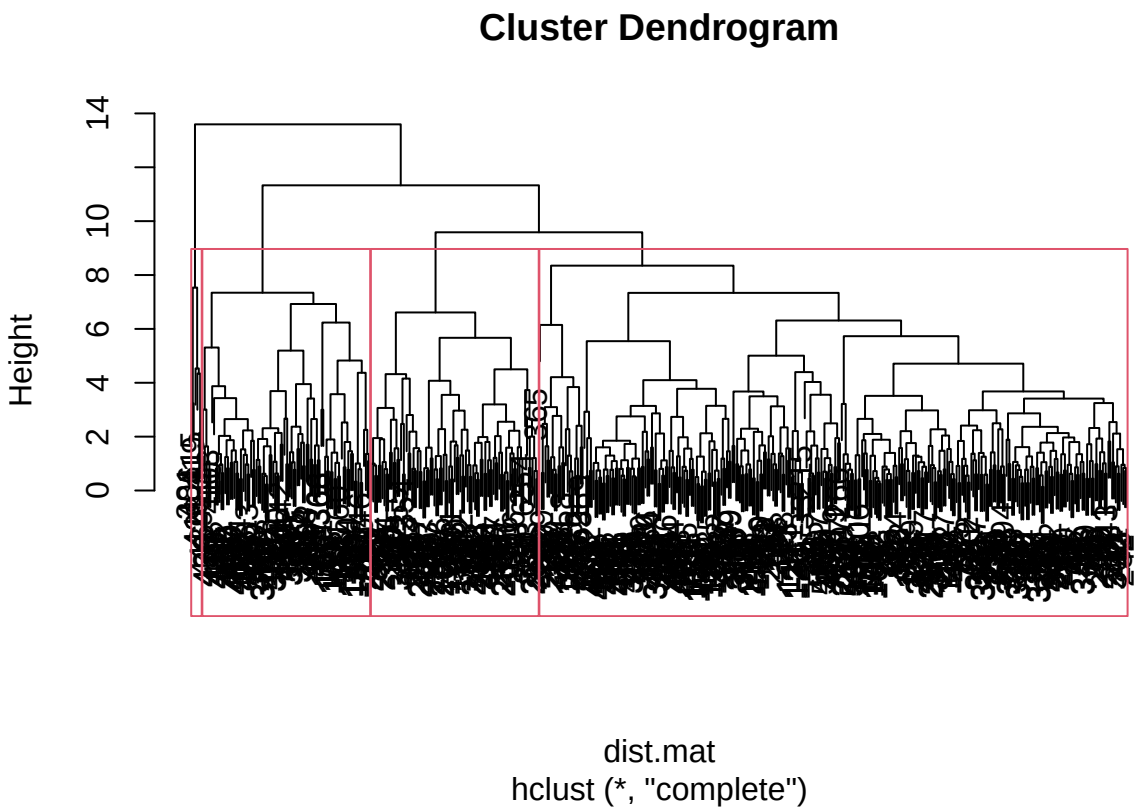


As shown in this "elbow" plot, it may be usefull to choose k=2 as a good trade-off between the complexity given by a big k and a fairly small total within sum of squares of the objective function.

Hierarchical Clustering

For the sake of illustration, we show how to implement hierarchical clustering by computing the distances matrix with the Euclidean distance and the "complete linkage".

```
dist.mat <- dist(s_mboston, method = "euclidean")

hc <- hclust(dist.mat, method = "complete")

plot(hc)   #dendrogram
rect.hclust(hc, k = 4)  #shows the number of desired groups on the dendrogram
```




```
groups <- cutree( hc, k = 4)
groups
```

```
##   [1] 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 1 1 1 1 1 2 2 2 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 1 2 1 1 3 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3
## [149] 3 3 3 3 3 3 3 3 3 1 3 3 3 1 1 1 3 3 1 3 3 3 3 3 1 1 1 1 1 1 1 2 1 1 1 1 1
## [186] 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [223] 1 1 1 1 1 2 1 1 1 2 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 2 2 2 2 2 2 2 2 1 1
## [260] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 1 1
## [297] 1 1 2 2 2 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
## [334] 1 1 1 1 1 1 1 1 2 1 2 2 1 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 3 1 3 1 1
## [371] 1 1 1 3 3 1 3 1 3 3 4 3 3 3 3 3 3 3 3 3 1 1 3 1 1 1 1 1 3 3 3 3 1 3 3 4 3
## [408] 1 3 3 4 3 3 3 4 3 3 3 4 3 1 1 1 3 3 3 3 4 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1
## [445] 3 3 1 1 1 1 3 1 1 1 3 3 3 3 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 3 3 1 3 1 1 1
## [482] 1 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

4

Since in the standard K-means algorithm the initial centroids are chosen as random data points, it might happen that no point is associated to it and at the same time, it could be that more than one cluster is linked to the same centroid. Furthermore, if these centroids are far from the optimal solution, many steps are required to reach the local optimum (i.e a poor initialization of centroids resulted in poor clustering)

K-means++ can be used to overcome these difficulties by ensuring a better initialization of the centroids. The rest of the algorithm is the same as the standard K-means. The steps of K-Means++ are:

- 1. From the data points, randomly select the first centroid.
- 2. For each data point compute its distance from the nearest, previously chosen centroid.
- 3. Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid. (i.e. the point having maximum distance from the nearest centroid is most likely to be selected next as a centroid)
- 4. Repeat steps 2 and 3 until k centroids have been sampled To sum up, by following the above procedure for initialization, we pick up centroids that are far away from one another. This increases the chances of initially picking up centroids that lie in different clusters. Also, since centroids are picked up from the data points, and not from the hyperspace, each centroid has some data points associated with it at the end.

To implement K-Means++ and to compare it to the standard K-Means, we arbitrary choose k=3 and the "solive" dataset.

```
library(pracma)
kmpp1 <- function(X, k) {
  n <- nrow(X)
  C <- numeric(k)
  C[1] <- sample(1:n, 1)
  for (i in 2:k) {
    dm <- distmat(X, X[C, ])
    pr <- apply(dm, 1, min); pr[C] <- 0
    C[i] <- sample(1:n, 1, prob = pr)
  }
  kmeans(X, X[C, ], nstart = 1)
}


kmpp2 <- function(X, k) {
  n <- nrow(X)
  C <- numeric(k)
  C[1] <- sample(1:n, 1)
  for (i in 2:k) {
    dm <- distmat(X, X[C, ])
    pr <- apply(dm, 1, min); pr[C] <- 0
    C[i] <- sample(1:n, 1, prob = pr)
  }
  kmeans(X, X[C, ], nstart = 100)
}
```

Results

```
res_kmpp1 <- kmpp1(as.matrix(solive), 3)
res_kmpp2 <- kmpp2(as.matrix(solive), 3)
```

```r
res_km1 <- kmeans(solive, 3, nstart = 1)
res_km2 <- kmeans(solive, 3, nstart = 100)

res_kmpp1$tot.withinss
```

```
## [1] 2352.049
```

```r
res_kmpp2$tot.withinss
```

```
## [1] 2320.024
```

```r
res_km1$tot.withinss
```

```
## [1] 2473.273
```

```r
res_km2$tot.withinss
```

```
## [1] 2320.024
```

```r
results <- c(res_kmpp1$tot.withinss, res_kmpp2$tot.withinss,
         res_km1$tot.withinss, res_km2$tot.withinss)

which.min(results)
```

```
## [1] 2
```

The best result is achieved by using the K-Means++ with nstart = 100