Csoportosítási megoldások (klaszterezés)

Contents


```
knitr::opts_chunk$set(echo=TRUE)
library('MASS')
library('cluster')
library('clv')
```

```
## Loading required package: class
```

```
library('fpc')
library('compositions')
```

```
## Welcome to compositions, a package for compositional data analysis.
## Find an intro with "? compositions"
```

```
##
## Attaching package: 'compositions'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, cov, dist, var
```

```
## The following objects are masked from 'package:base':
##
##     %*%, norm, scale, scale.default
```

1 Bevezetés

1.1 Csoportosítás fogalma

A csoportosítás (klaszterezés) egy felügyelet nélküli gépi tanulási módszer, melynek célja, hogy valamilyen szempont szerint az adatokban összetartozó csoportokat, osztályokat (klasztereket) keressen. Az osztályozást tekinthetjük egyfajta reprezentáció tanulási folyamatnak. Felmerül a kérdés, hogy mit jelent hogy összetartozó

osztályok? A klaszterezés során a távolság fogalom kulcsfontosságú jelentéssel bír, például a legegyszerűbb esetben euklideszi távolság.

1.2 Csoportosítás típusai

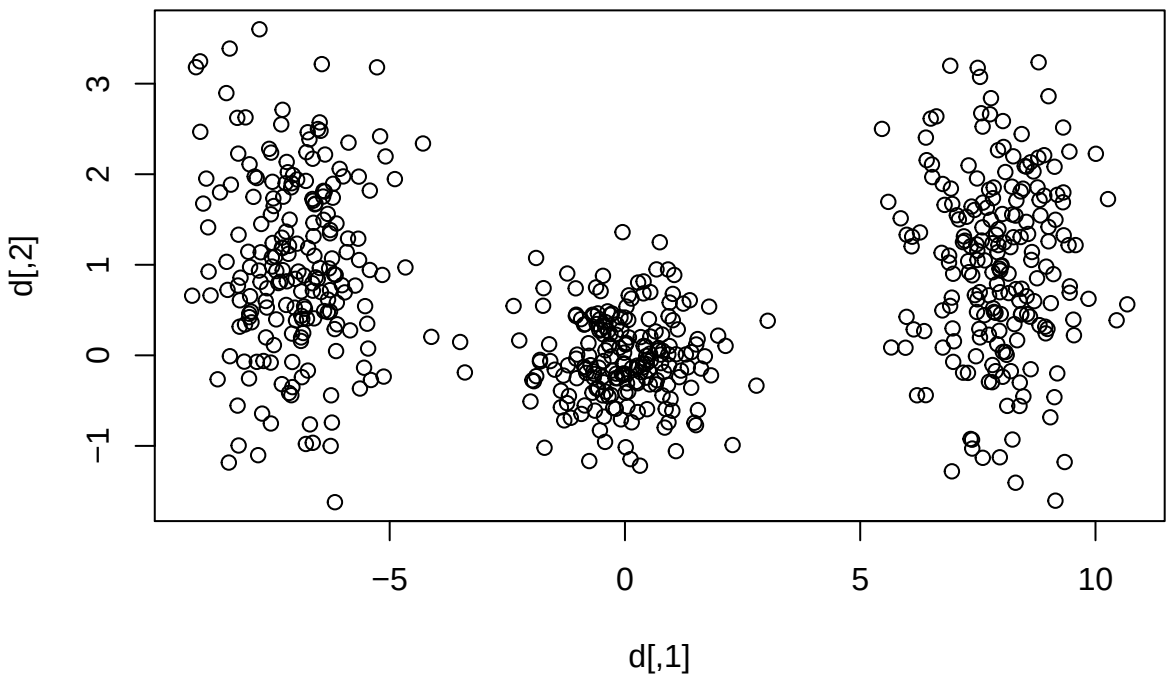
A csoportosításnak rengeteg formája létezik, ezeket sokféle szempont alapján kategorizálhatjuk. A tipikus megvalósítások a következők szerint jellemezhetők:

- Középpont alapú csoportosítás
  - Minden csoportot elsősorban a csoport középpontjával azonosít
  - pl. k-közép csoportosítás
- Kapcsolat alapú csoportosítás
  - Az egyes közeli pontok "összekapcsolásával" képez csoportokat
  - pl. hierarchikus csoportosítás
- Eloszlás alapú csoportosítás
  - Feltételezi, hogy az egyes csoportok valamilyen eloszlást követnek
  - pl. expectation-maximalization algoritmusok
- Sűrűség alapú csoportosítás
  - A sűrűbben elhelyezkedő pontokból képez csoportokat
  - pl. DBSCAN

1.3 Egyszerű példa

Tekintsük az alábbi generált, két dimenziós adathalmazt:

```
set.seed(5)
d<-rbind(
  mvrnorm(200,c(-7,1),Sigma=diag(c(1,1))),
  mvrnorm(200,c(0,0),Sigma=diag(c(1,.2))),
  mvrnorm(200,c(8,1),Sigma=diag(c(1,1)))
)
plot(d)
```



Az ábrán láthatjuk, hogy az adathalmazban három elkülöníthető pontcsoportot találhatunk, amelyeket

adatelemzési terminológiával klasztereknek hívjuk. Természetesen nem minden adathalmazban látható ennyire egyértelműen a klaszterek léte, ugyanakkor jelen példánk az áttekinthetőség kedvéért jól elkülöníti őket. Egy C ponthalmazt feloszhatunk k darab klaszterre, azaz $C = \{C_1, C_2, \ldots, C_k\}$. Egy C_i klasztert a klaszterhez tartozó pontokkal jellemzünk, tehát $x \in C_i$. Számos klaszterezési eljárást ismerünk, ugyanakkor a klaszterezés minőségének mérésére számos, az egyes algoritmusoktól független mértéket használnuk.

A teljes ponthalmaz eloszlásáról ad képet a ponthalmaz teljes négyzetösszege (tulajdonképpen szórása, TOTSS - Total Sum of Squares), ahol $\mu = \frac{1}{N}\sum_{x \in C} x$:

$$TOTSS = \sum_{x \in C} \|x - \mu\|^2$$

A klaszterek egyik kulcsfontosságú jellemzője a klaszter **középpontja**, azaz $\mu_i = \frac{1}{N_i}\sum_{x \in C_i} x$. Az egyik legfontosabb jellemzője a csoportosításnak az egyes klasztereken belül eső pontok négyzetösszege (**WCSS**, within-cluster sum of squares):

$$WCSS_i = \sum_{x \in C_i} \|x - \mu_i\|^2$$

A WCSS egyben a klaszter pontjainak szórásával egyenértékű, hiszen $\sum_{x \in C_i} \|x - \mu_i\|^2 = N_i \mathrm{var}(C_i)$ A klaszterek közötti hiba négyzetösszege (BSS - Between Clusters Sum of Squares) az egyes klaszterközéppontok a teljes ponthalmaz középpontjától vett távolságainak négyzetösszege (ahol N_i az i-dik klaszter pontjainak száma):

$$BSS = \sum_{i=1}^{n} N_i \|\mu_i - \mu\|^2$$

A bemutatott adathalmazra például a TOTSS értéke:

```
totss<-sum(scale(d,scale=F)^2)
print(totss)
```

```
## [1] 23443.13
```

2 K-közép csoportosítás (K-means clustering)

A K-közép csoportosítás célja, hogy egy N pontból álló adathalmazra megkeresse azt a $C = \{C_1, C_2, \ldots, C_k\}$ csoportosítást, amely minimalizálja a teljes WCSS (TWCSS) értékét, azaz

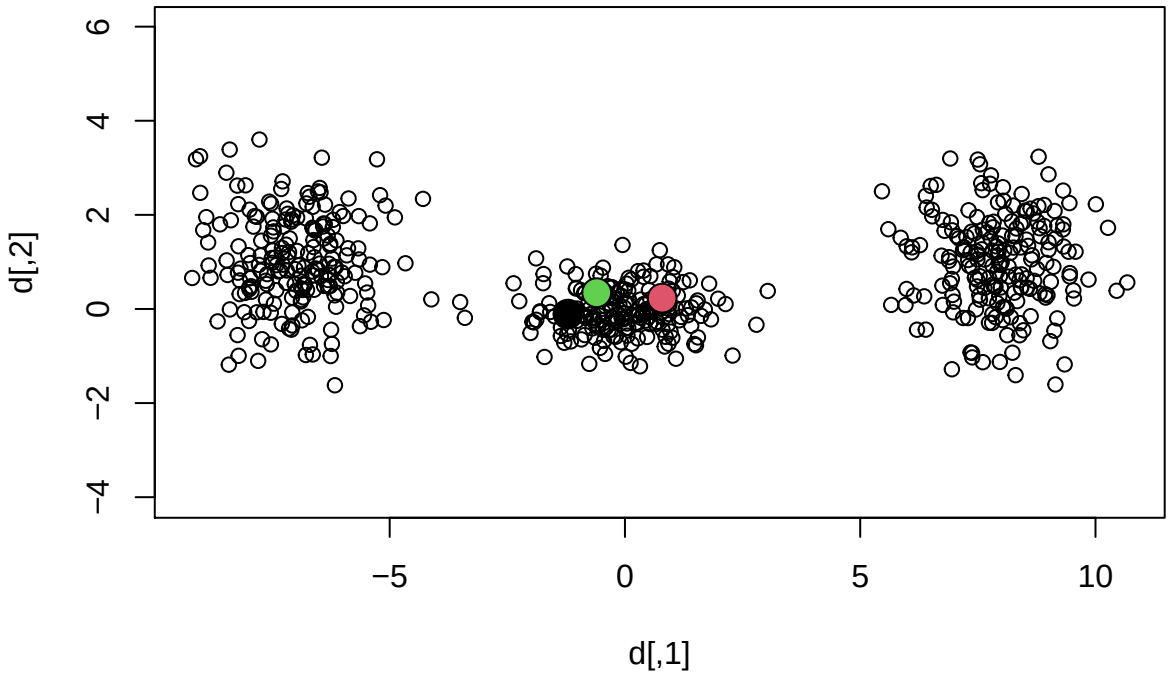
$$\arg\min_{C} \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_i\|^2$$

A K-közép csoportosítás megvalósítására számos módszer ismert, ezek közül a legismertebb a naiv K-közép algoritmus (vagy Lloyd-algoritmus), amely egy véletlenszerűen választott kezdőhelyzetből iteratívan próbálja megtalálni az optimális csoportosítást. Az R nyelv a más metrikával dolgozó Hartigan-Wong algoritmust használja alapértelmezetten, amely gyorsabb konvergenciával rendelkezik.

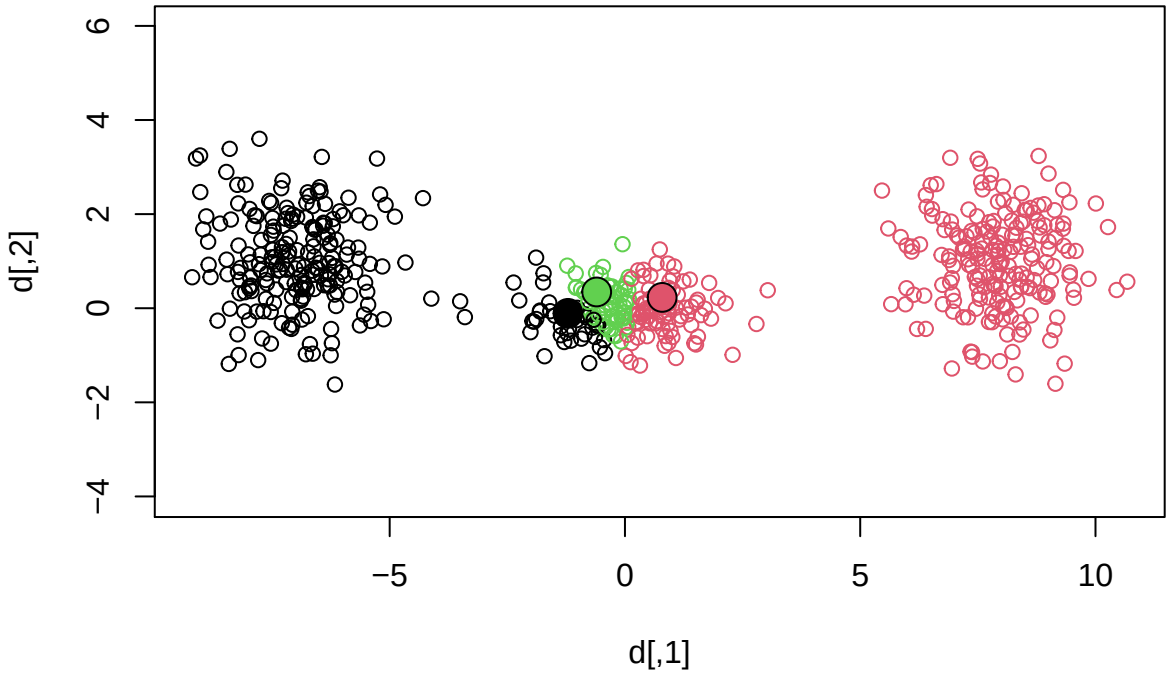
Fontos kiemelni, hogy a K-közép algoritmusok esetén semmi nem garantálja a globális minimumhoz történő konvergenciát, azaz az egyes algoritmusok nem feltétlenül az optimális eredményt adják, megragadhatnak lokális minimumokban.

A K-közép csoportosítás egy iteratív folyamat, az ún. Expectation Maximalization algoritmusok egy leegyszerűsített, speciális esete. A következő ábrák mutatják a példa adathalmazra az algoritmus tipikus futását:

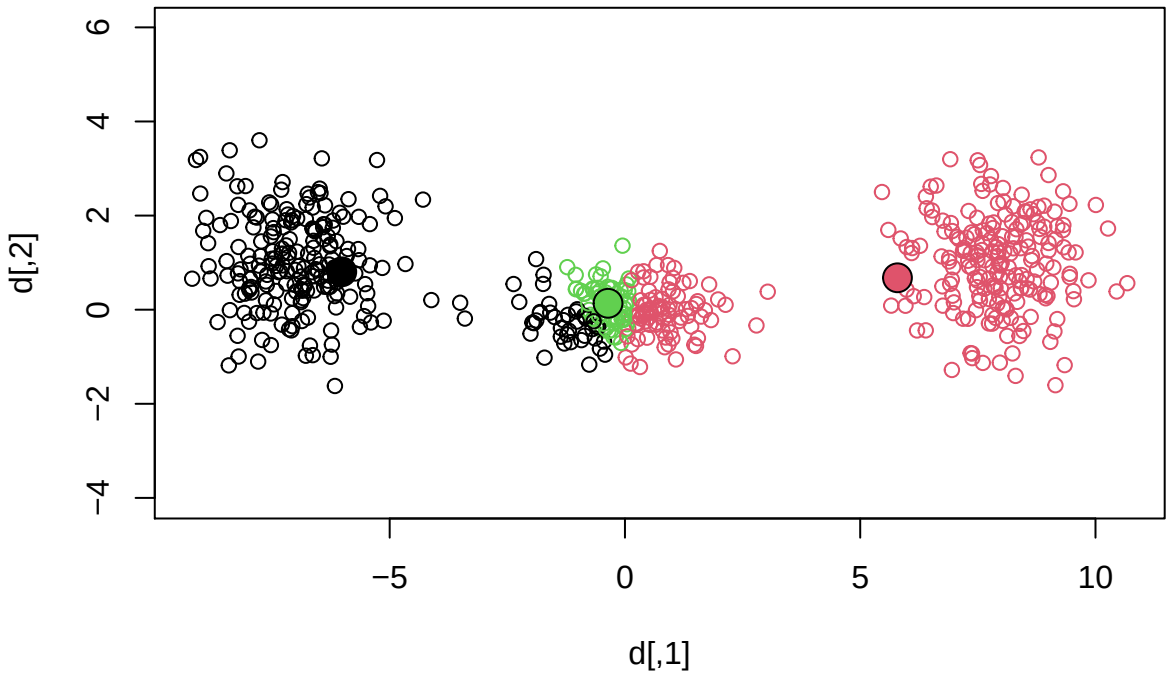
```
centers=d[c(350,370,360),]
plot(d,asp=1)
points(centers,asp=1,pch=21,cex=2,bg=1:3)
```



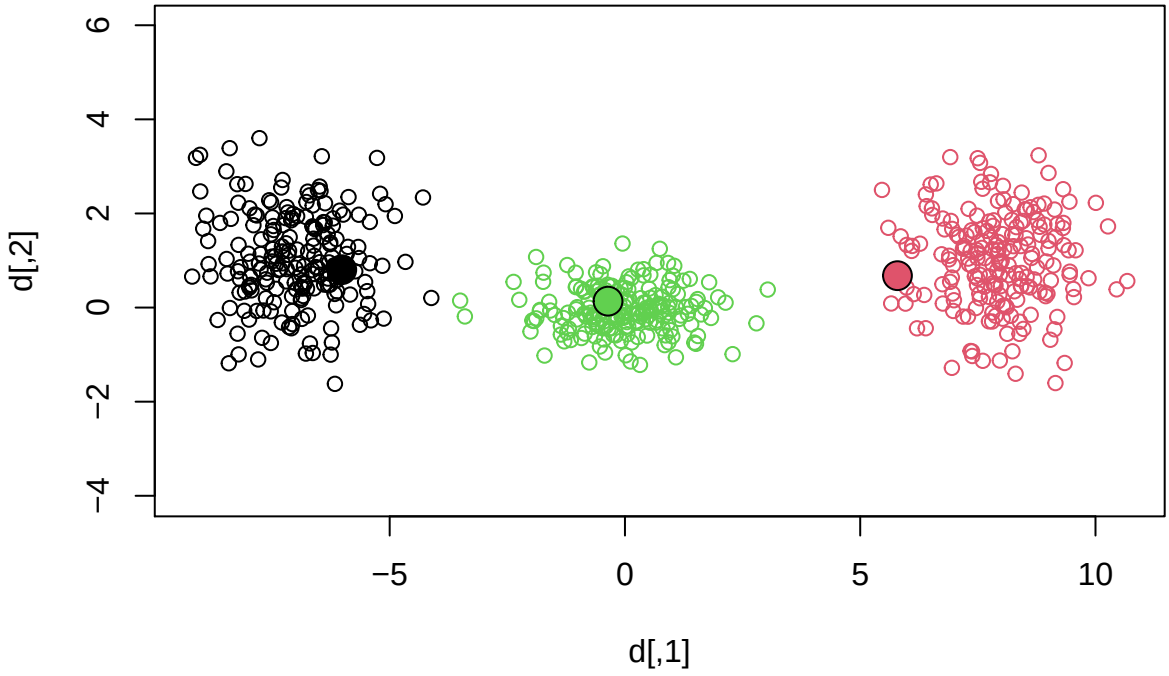

```
suppressWarnings(m<-kmeans(d, iter.max=1, centers=centers, algorithm="Lloyd"))
plot(d,col=m$cluster,asp=1)
points(centers,asp=1,pch=21,cex=2,bg=1:3)
```



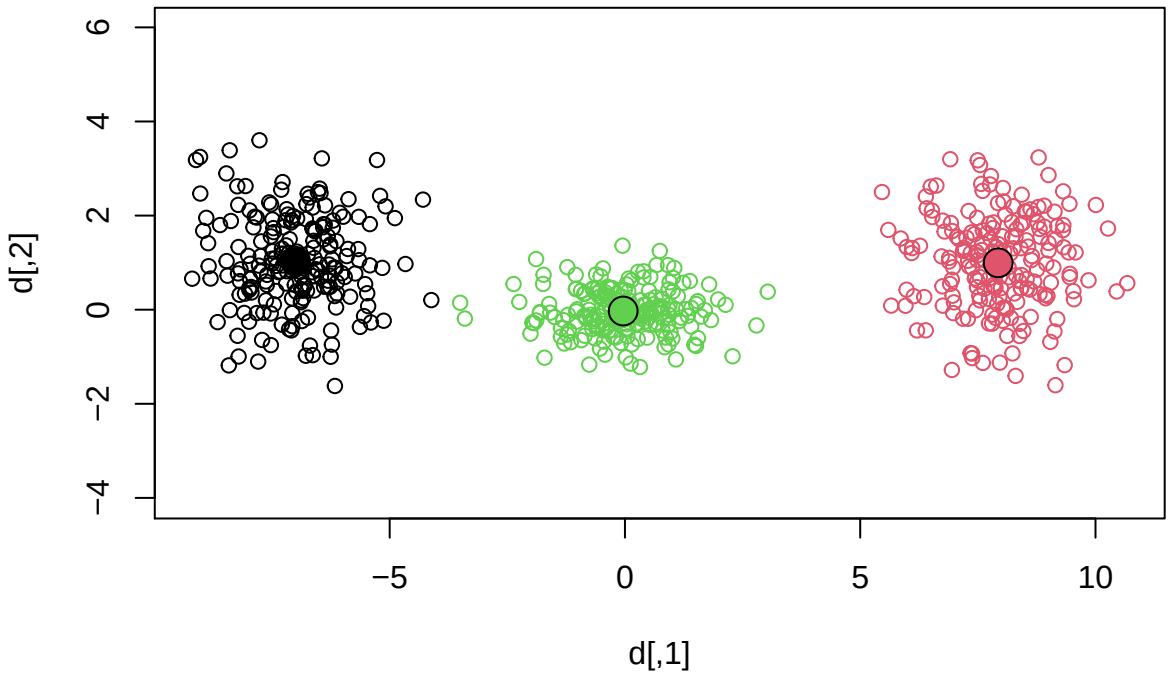

```
plot(d,col=m$cluster,asp=1)
points(m$centers,asp=1,pch=21,cex=2,bg=1:3)
```




```
centers=m$centers
suppressWarnings(m<-kmeans(d, iter.max=2, centers=centers, algorithm="Lloyd"))
plot(d,col=m$cluster,asp=1)
points(centers,asp=1,pch=21,cex=2,bg=1:3)
```




```
plot(d,col=m$cluster,asp=1)
points(m$centers,asp=1,pch=21,cex=2,bg=1:3)
```



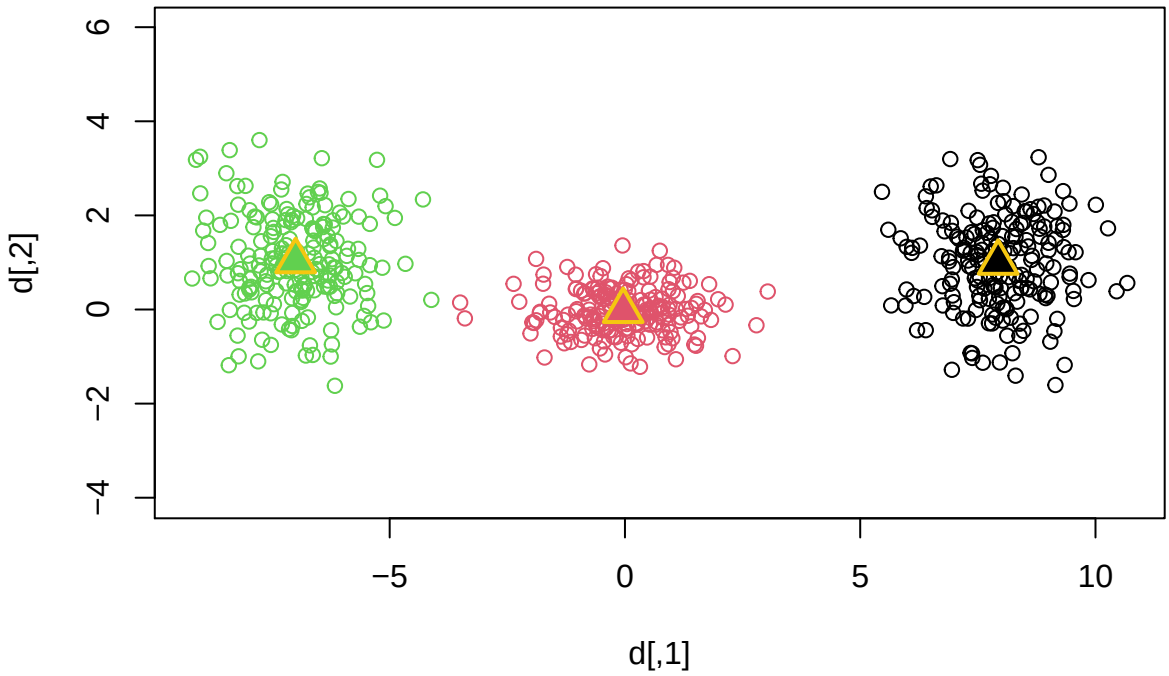
Az R nyelvben a K-közép algoritmust a `kmeans` függvény valósítja meg, amely az adathalmazt várja bemeneti adatként, és a keresendő klaszterek számát. Mivel ez utóbbit szemmel meg tudjuk állapítani, ezért 3 klasztert fogunk keresni:

```
m<-kmeans(d,3)
print(m)
```

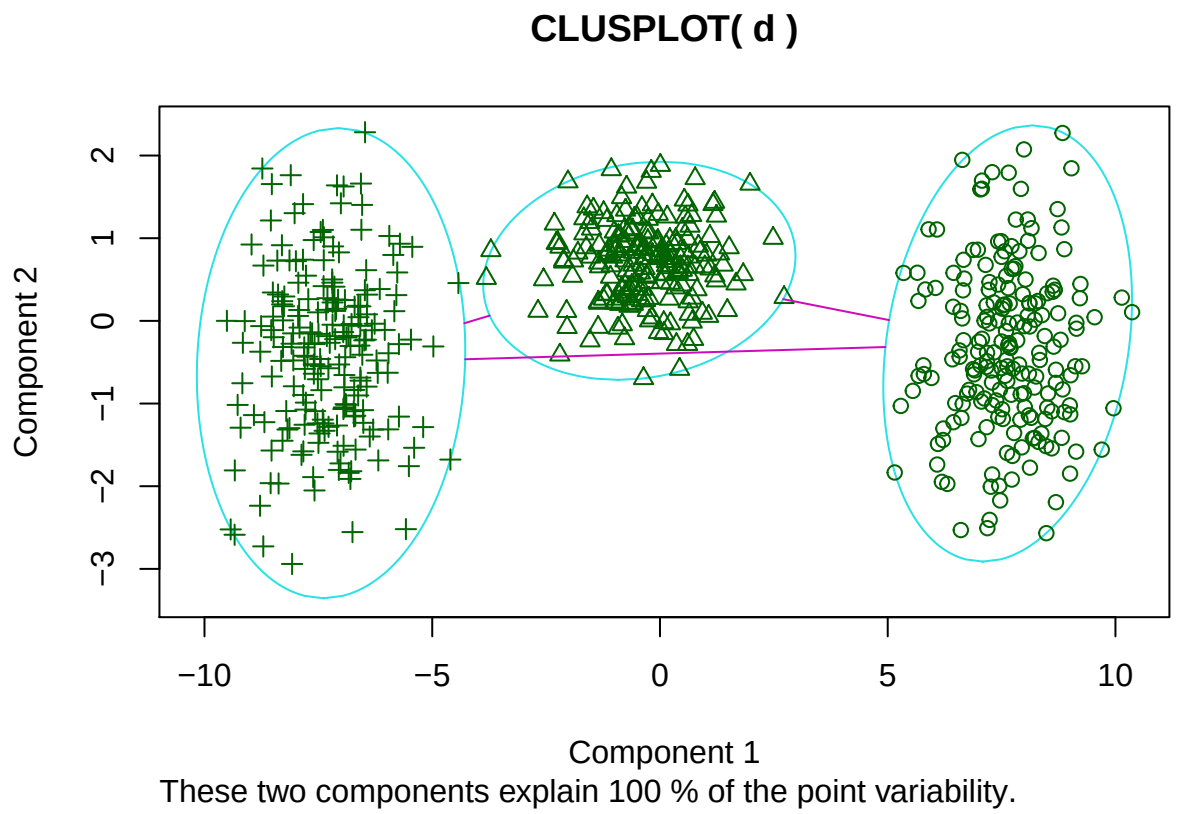
```
## K-means clustering with 3 clusters of sizes 200, 201, 199
##
## Cluster means:
##          [,1]        [,2]
## 1  7.93158237  0.99610911
## 2 -0.03707825 -0.02847592
## 3 -6.99890808  1.02847911
##
## Clustering vector:
##   [1] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3
##  [38] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [186] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [223] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [260] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [297] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [334] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [371] 2 2 2 2 2 2 2 2 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
## [408] 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 1 1 1 1 1 1
## [445] 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 1 1 1 1 1 1
## [482] 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 1 1 1 1 1 1
## [519] 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 1 1 1 1 1 1
## [556] 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 1 1 1 1 1 1
## [593] 1 1 1 1 1 1 1 1
```

```
##
## Within cluster sum of squares by cluster:
## [1] 381.1800 257.9507 386.5948
##  (between_SS / total_SS =  95.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

```
plot.clusters=function(d,m) {
  plot(d,col=m$cluster,asp=1)
  points(m$centers,bg=(1:nrow(m$centers)),cex=2,col=7,pch=24,lwd=2)
}
plot.clusters(d,m)
```

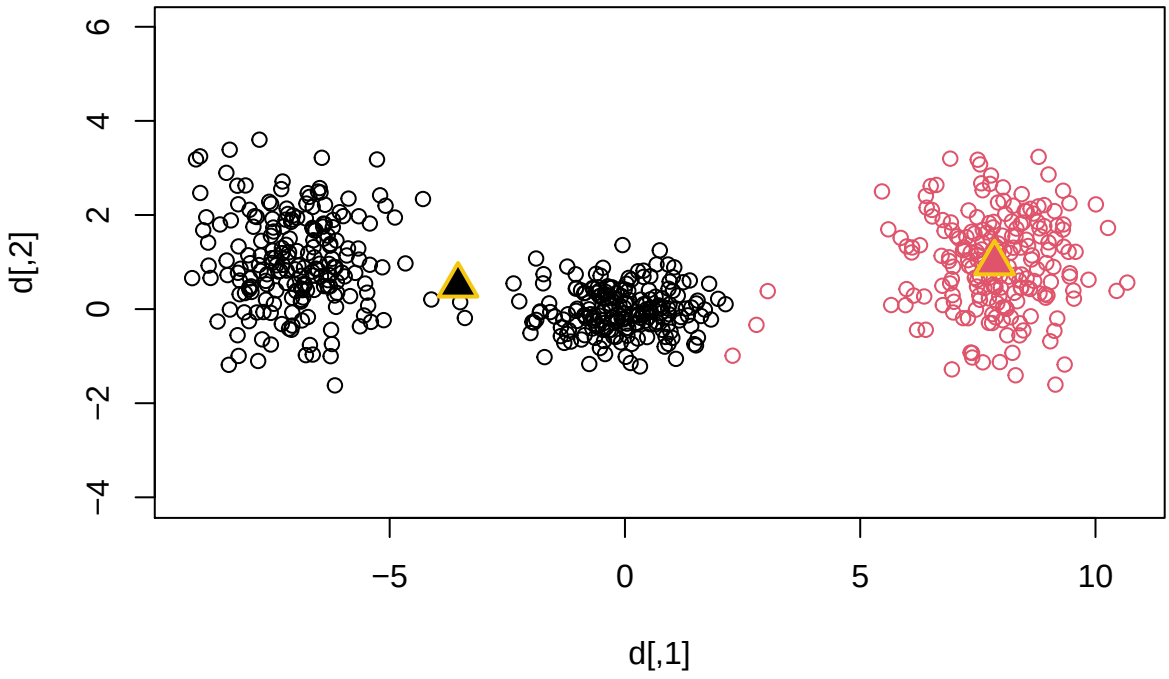



```
clusplot(d,m$cluster)
```

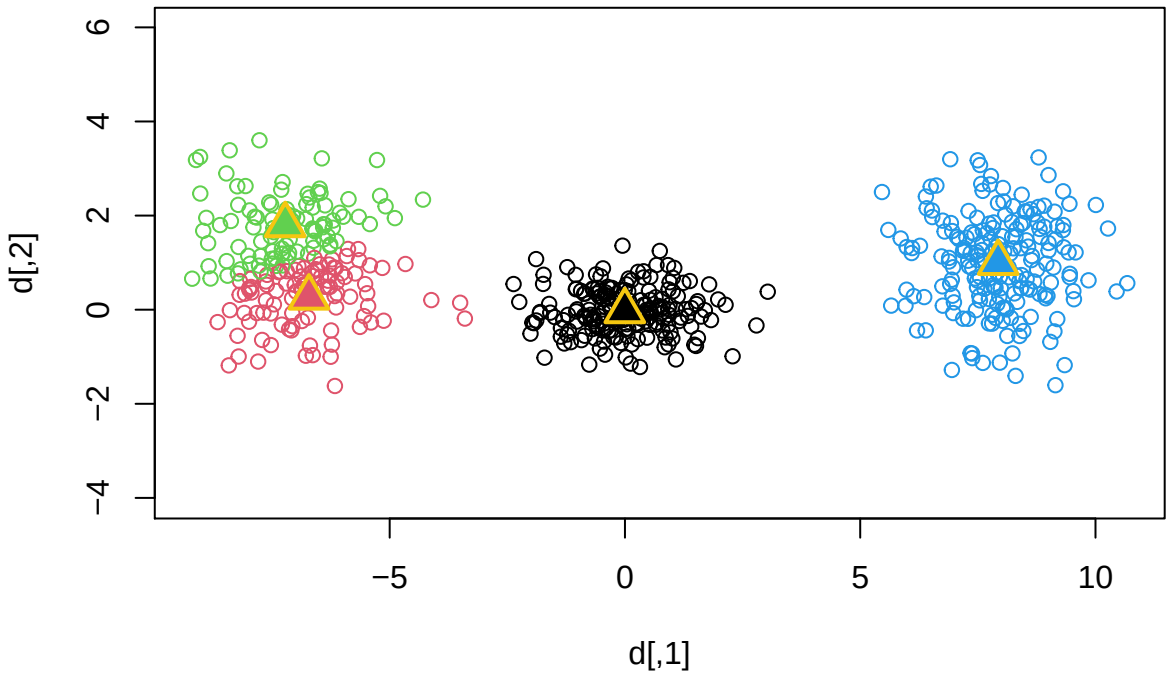


Próbáljuk meg más klaszterszámra is az algoritmust, és jelenítsük meg az eredményt:

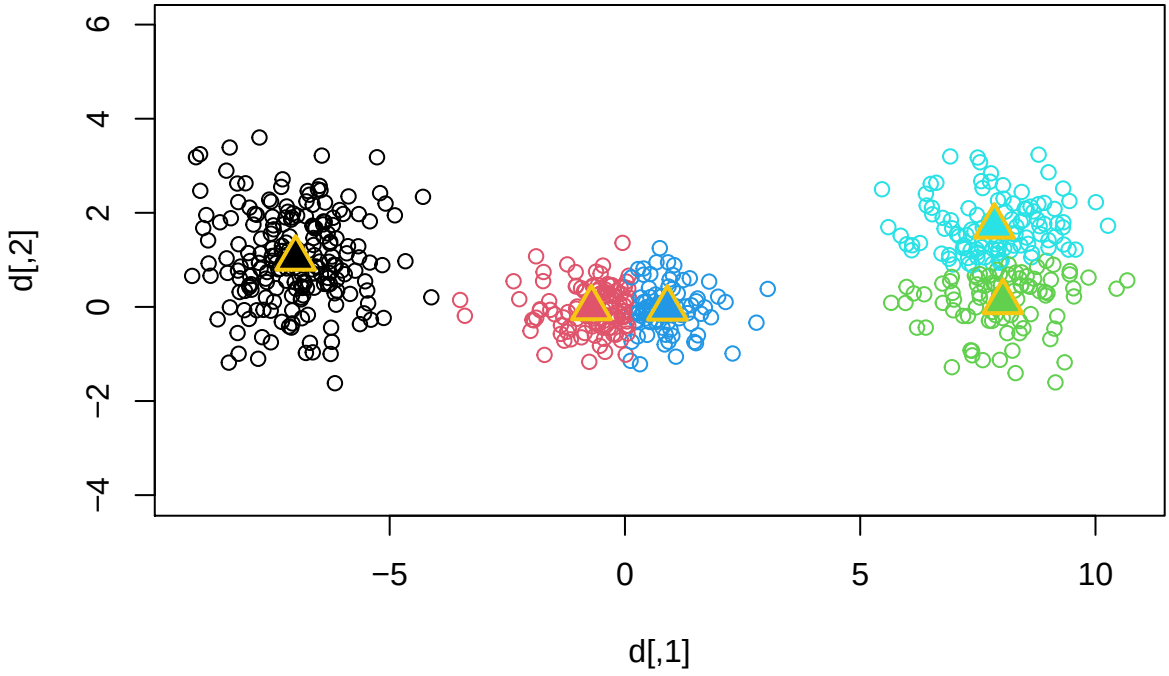
```
m<-kmeans(d,2)
plot.clusters(d,m)
```



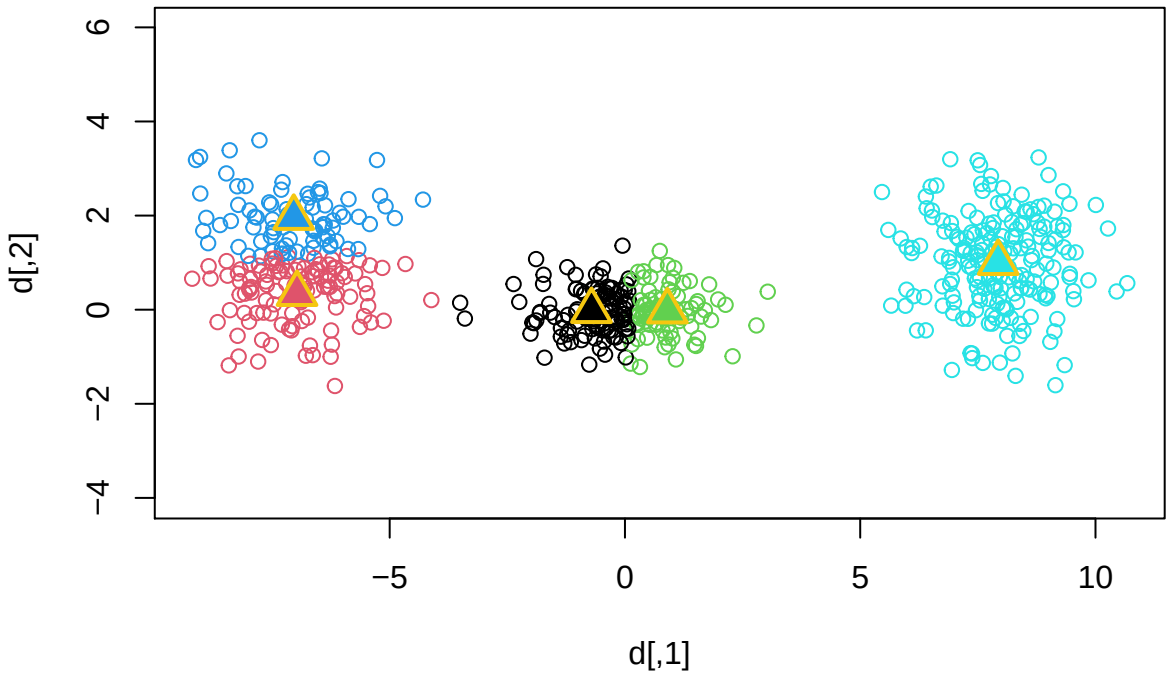

```
m<-kmeans(d,4)
plot.clusters(d,m)
```




```
m<-kmeans(d,5)
plot.clusters(d,m)
```

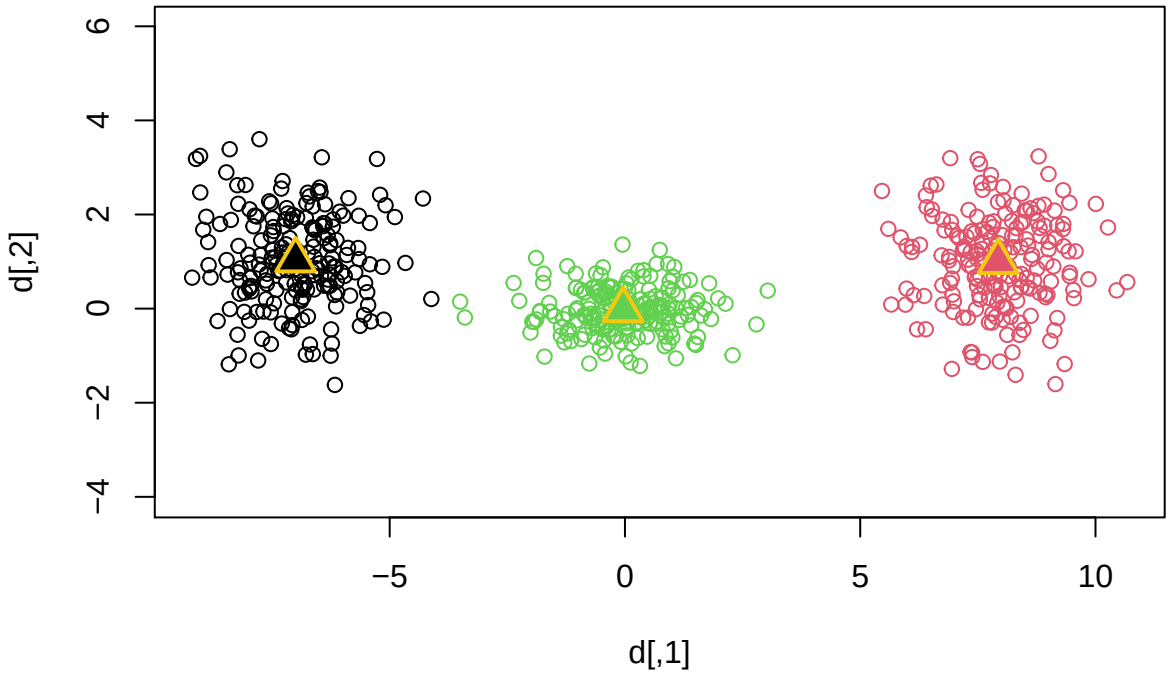



```
m<-kmeans(d,5)
plot.clusters(d,m)
```



A K-közép algoritmus futását jelentősen befolyásolja az iterációk száma és a kezdeti klaszter középpontok választása. Ezek a `iter.max` és `nstart` paraméterekkel szabályozhatók:

```
m<-kmeans(d,3,nstart=10,iter.max=100)
plot.clusters(d,m)
```

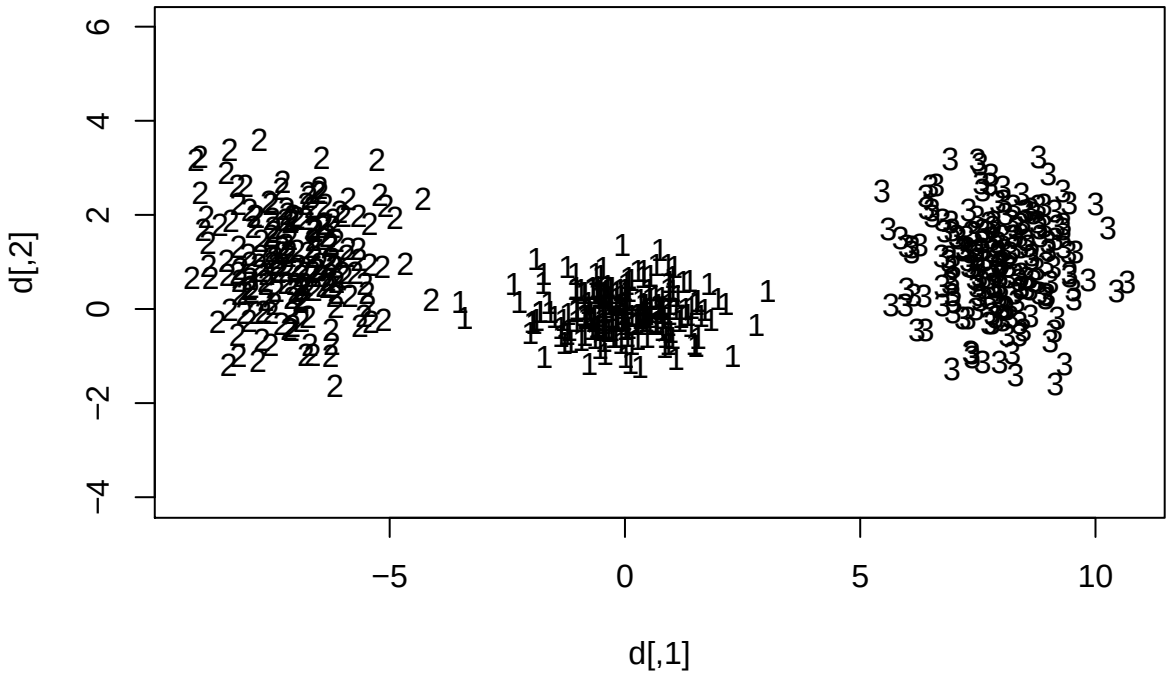


2.1 Klaszterjellemzők mérése

Tekintsük az alábbi klasztereket:

```
m<-kmeans(d,3)
plot(d,type="n",asp=1)
```

```
text(d,labels=m$cluster)
```



A klaszterezésről a legfontosabb mértékeket (klaszterek távolsága, klaszterek mérete) a `clv` csomagban található `cls.scatt.data` függvénnyel számíthatjuk ki (az egyes távolságok definícióit a `?cls.scatt.data` oldalon találjuk meg):

```
cls.scatt.data(d,m$cluster)
```

```
## $intracls.complete
##           c1       c2      c3
## [1,] 6.54109 5.820329 5.56113
##
## $intracls.average
##            c1       c2   c3
## [1,] 1.369407 1.749594 1.74
##
## $intracls.centroid
##             c1       c2       c3
## [1,] 0.9536134 1.229914 1.219765
##
## $intercls.single
##           c1        c2       c3
## c1 0.0000000 0.6157433 2.640255
## c2 0.6157433 0.0000000 9.756335
## c3 2.6402547 9.7563354 0.000000
##
## $intercls.complete
##          c1       c2       c3
## c1  0.00000 12.46943 14.18393
## c2 12.46943  0.00000 19.96437
## c3 14.18393 19.96437  0.00000
##
## $intercls.average
```

```
##          c1        c2        c3
## c1 0.000000  7.129882  8.111771
## c2 7.129882  0.000000 14.995879
## c3 8.111771 14.995879  0.000000
##
## $intercls.centroid
##          c1        c2       c3
## c1 0.000000  7.041607  8.03426
## c2 7.041607  0.000000 14.93053
## c3 8.034260 14.930526  0.00000
##
## $intercls.ave_to_cent
##          c1        c2        c3
## c1 0.000000  7.084883  8.072492
## c2 7.084883  0.000000 14.963168
## c3 8.072492 14.963168  0.000000
##
## $intercls.hausdorff
##          c1        c2        c3
## c1 0.000000  7.152211  9.160805
## c2 6.381695  0.000000 14.775511
## c3 7.643099 14.795308  0.000000
##
## $cluster.center
##           [,1]        [,2]
## c1 -0.03707825 -0.02847592
## c2 -6.99890808  1.02847911
## c3  7.93158237  0.99610911
##
## $cluster.size
## [1] 201 199 200
##
## attr(,"class")
## [1] "cls.list"
```

2.2 A klaszterek számának meghatározása

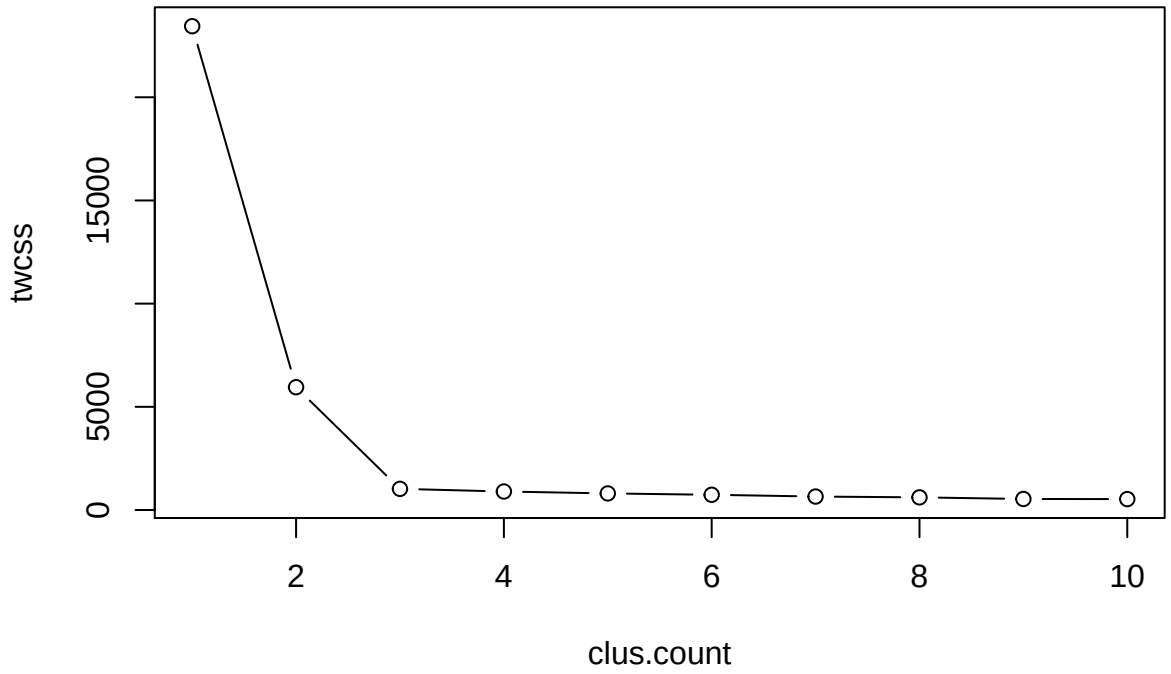
Általában a csoportok száma az adathalmazban ismeretlen, így annak a meghatározása is ránk marad. Nagy segítség, ha van valamilyen előzetes elvárásunk, amely alapján meg tudjuk határozni a klaszterek számát. Amennyiben ilyen nincs, számos módszer a rendelkezésünkre áll a klaszterek számának meghatározásához. Itt most két egyszerű módszert mutatunk be, a TWCSS alapján és a sziluett együttható alapján történő csoportszám meghatározást. Érdemes megemlíteni a TWCSS módszer formalizált változatát, a Gap Statistics módszert, amelyet itt nem tárgyalunk.

2.2.1 A TWCSS alapján történő csoportszám meghatározás

Láttuk, hogy a K-közép algoritmus célja a TWCSS érték minimalizálása. Éppen ezért, az ötlet az, hogy az értékét minimalizáljuk a klaszterek számának függvényében is. Végezzük el a klaszterezést 1-től 10 klaszterre, és ábrázoljuk a TWCSS értékének változását a klaszterszám függvényében.

```
clus.count<-1:10
twcss<-sapply(clus.count,function(k) {
  m<-kmeans(d,k)
  m$tot.withinss
})
```

```r
plot(clus.count,twcss,type="b")
```



Jól látható, hogy az ábrának a 3 klaszterszámnál van egy jellegzetes törése ("könyöke"), innentől a hiba mértéke már nem csökken jelentősen a klaszterek számával. Ez a töréspont, "könyök" a klaszterezés során ökölszabályként az optimális klaszterszám.

2.2.2 Sziluett alapú csoportszám meghatározása (haladó)

Hasznos jelzőszám a klaszterek minőségére az ún. sziluett együttható. A sziluett együttható lényege a következő. Minden egyes pontra kiszámítjuk az

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i,j)$$

és

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i,j)$$

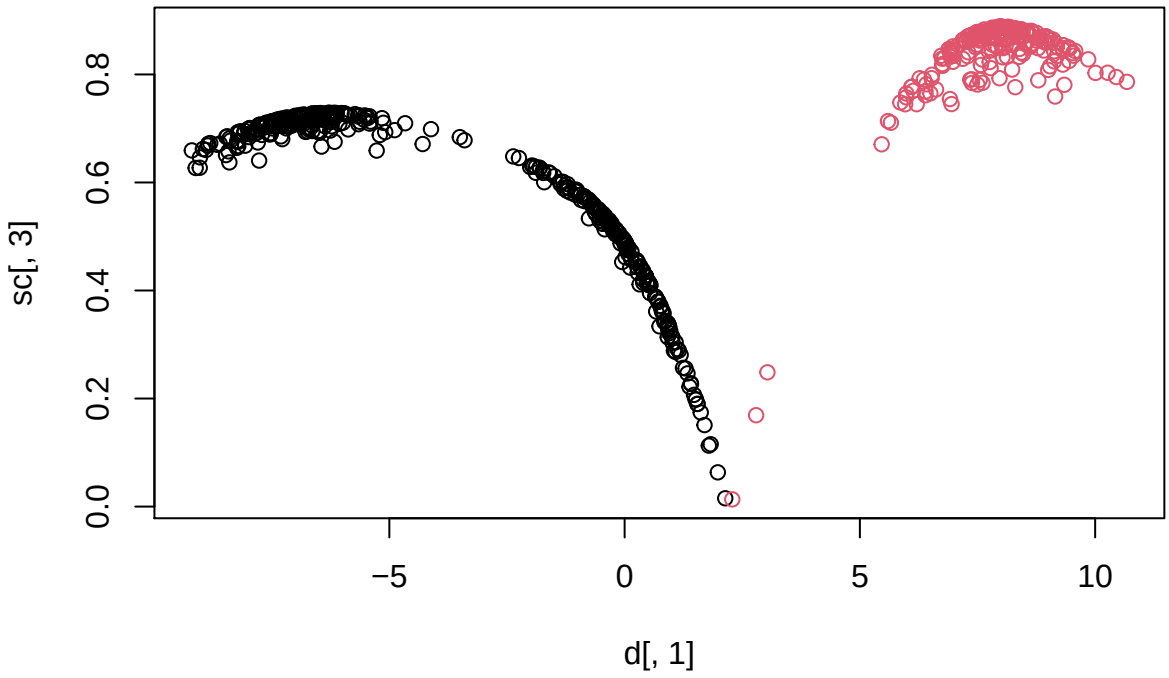
mértékeket. Itt $d(i,j)$ az i és j pont közötti távolság, C_i az i ponthoz tartozó klaszter. Ha megfigyeljük, $a(i)$ az i-dik pont átlagos távolsága a saját klaszteréhez tartozó pontoktól, míg $b(i)$ az i pontot nem tartalmazó klaszterek i ponttól való átlagos távolságainak a minimuma. Ebből következően a sziluett együttható a következő mérték:

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{if} |C_i| > 1$$

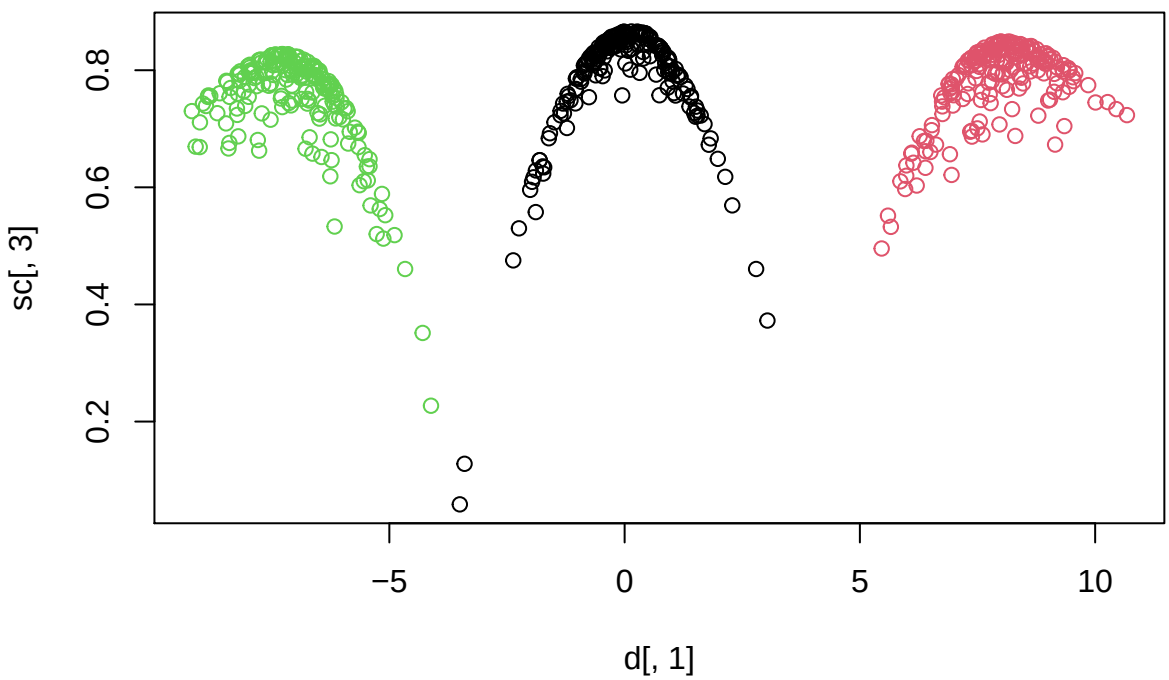
Megjegyezzük, hogy $s(i) = 0$, ha $|C_i| = 1$. Az $s(i)$ sziluett együtthatóra igaz, hogy $-1 \leq s(i) \leq 1$. A jelentése meglehetősen egyszerű: ha az $s(i)$ értéke $+1$-hez közeli, akkor az adott pont elhelyezkedése a saját klaszterében megfelelő, hiszen a saját klaszteréhez közel van, míg a többitől távol. Amennyiben az $s(i)$ érték alacsony, a pont klaszterben való helye nem optimális.

Szerencsére, a sziluett együttható kiszámítása megtalálható a `cluster` csomagban, így az implementációval nem kell törődnünk. Tekintsük meg például pár klaszterméretre a sziluett együttható értékeit:

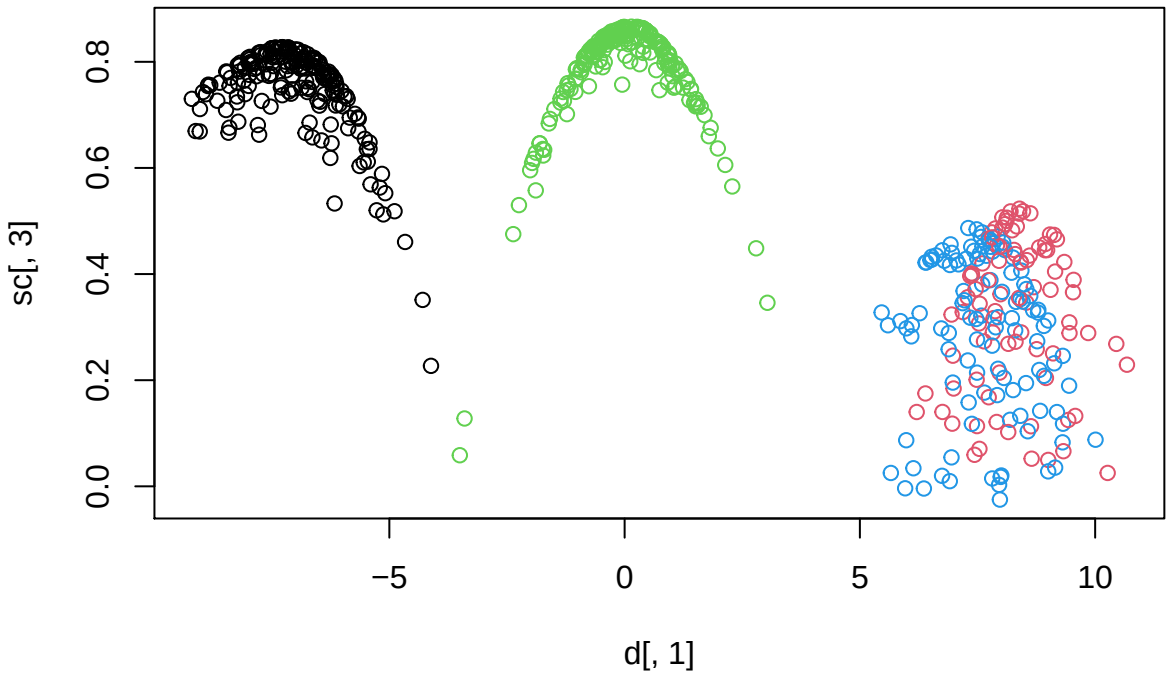
```
silh.plot<-function(k) {
  m<-kmeans(d,k)
  sc<-silhouette(m$cluster, dist(d))
  plot(d[,1],sc[,3],col=m$cluster)
}
silh.plot(2)
```



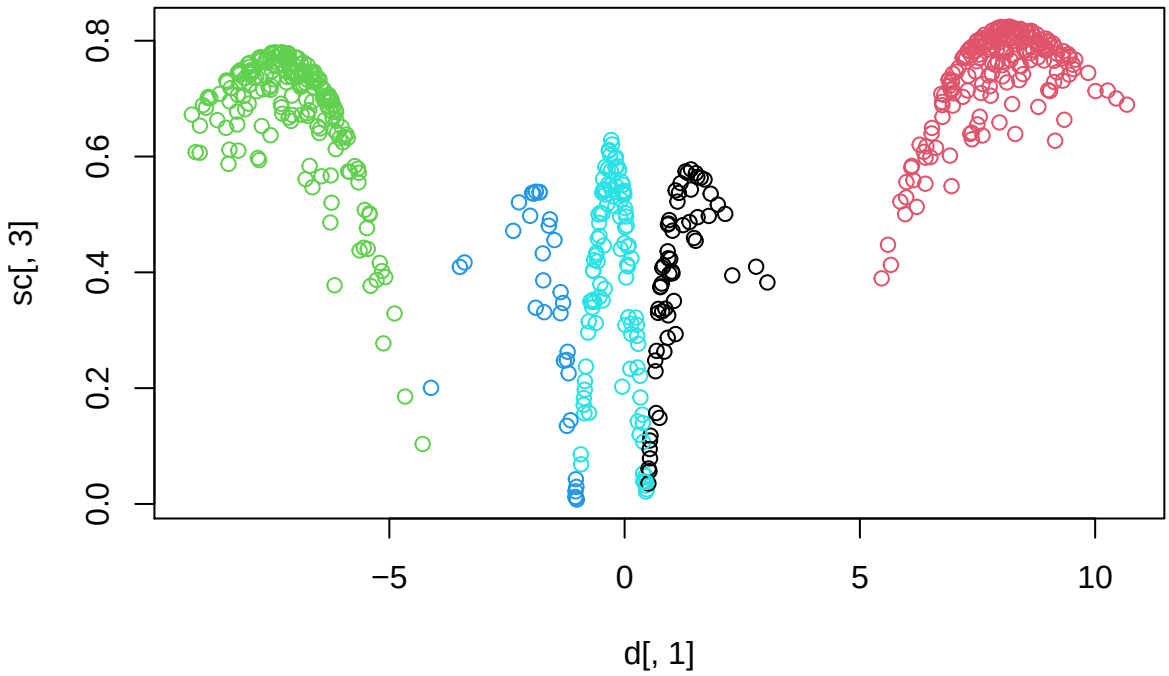

```
silh.plot(3)
```




```
silh.plot(4)
```

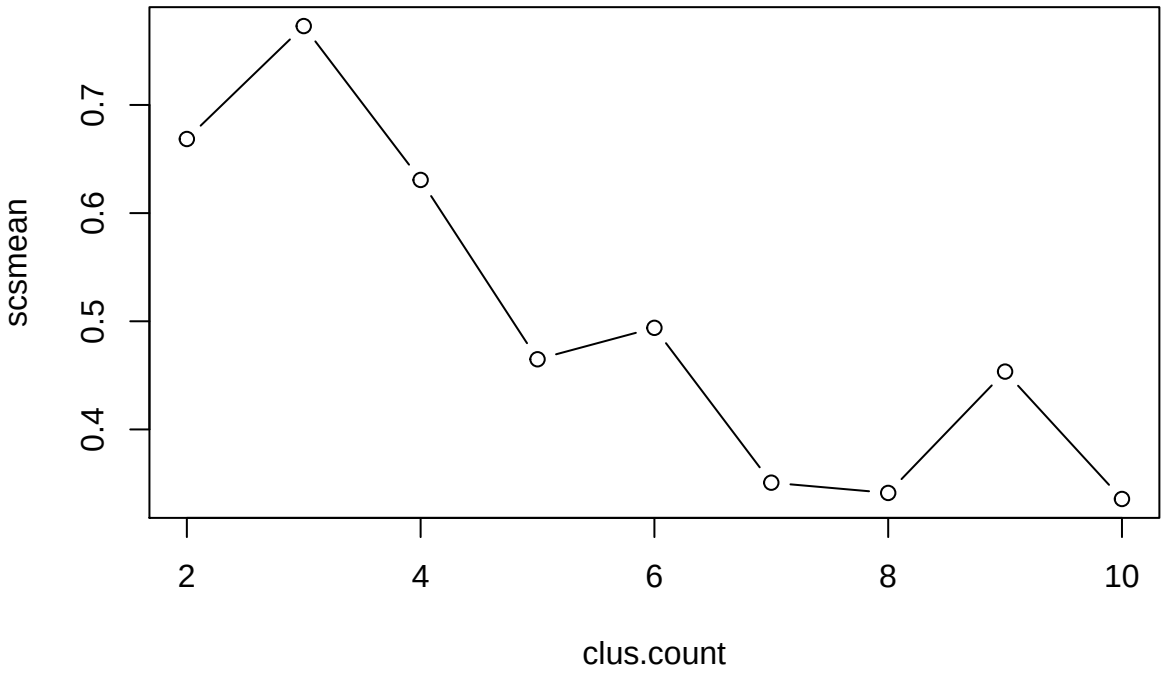



```
silh.plot(5)
```

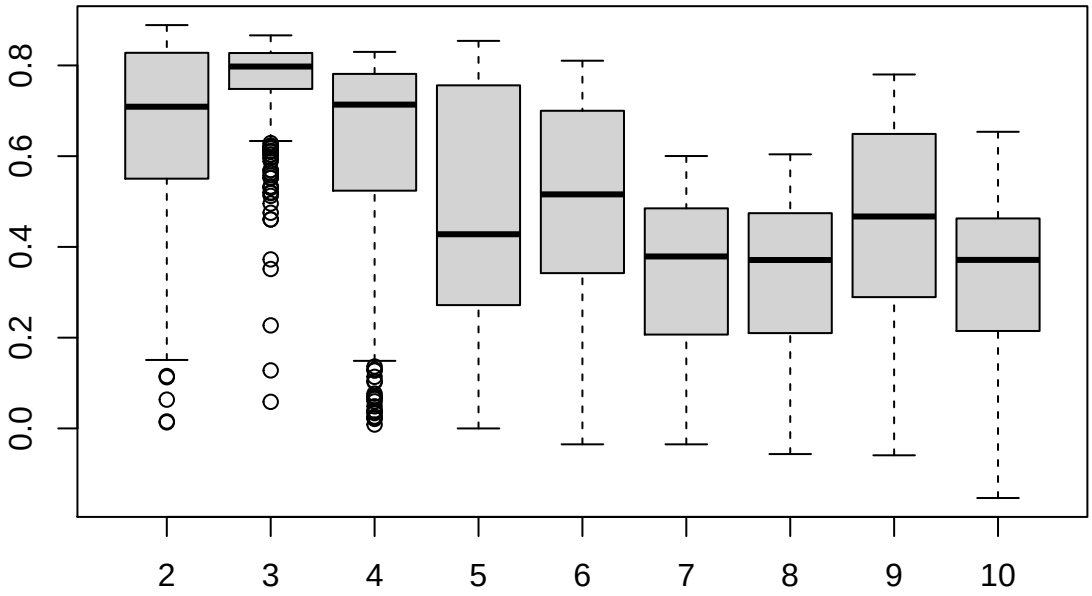



```
f<-function(k)
{
  m<-kmeans(d,k)
  sc<-silhouette(m$cluster,dist(d))
  sc[,3]
}

clus.count<-2:10
scstat<-sapply(clus.count,f)
```

```
scsmean<-apply(scstat,2,mean)
plot(clus.count,scsmean,type="b")
```




```
boxplot(scstat,names=clus.count)
```



3 Hierarchikus csoportosítás

A hierarchikus csoportosítás lényege, hogy a klaszterek képzése több lépésben, az adathalmaz egyesítésével vagy megosztásával történik. Ebből következően egy fát kapunk, amelynek csomópontjaiban az egyes megosztások/egyesítések találhatóak, a leveleken pedig az egyes adatentitások (adatsorok). A hierarchikus csoportosításnak alapvetően kétféle megvalósítása ismert:

- *agglomeratív*: az algoritmusok az egyes adatsorokból indulnak ki, és minden lépésben két adatsort vagy

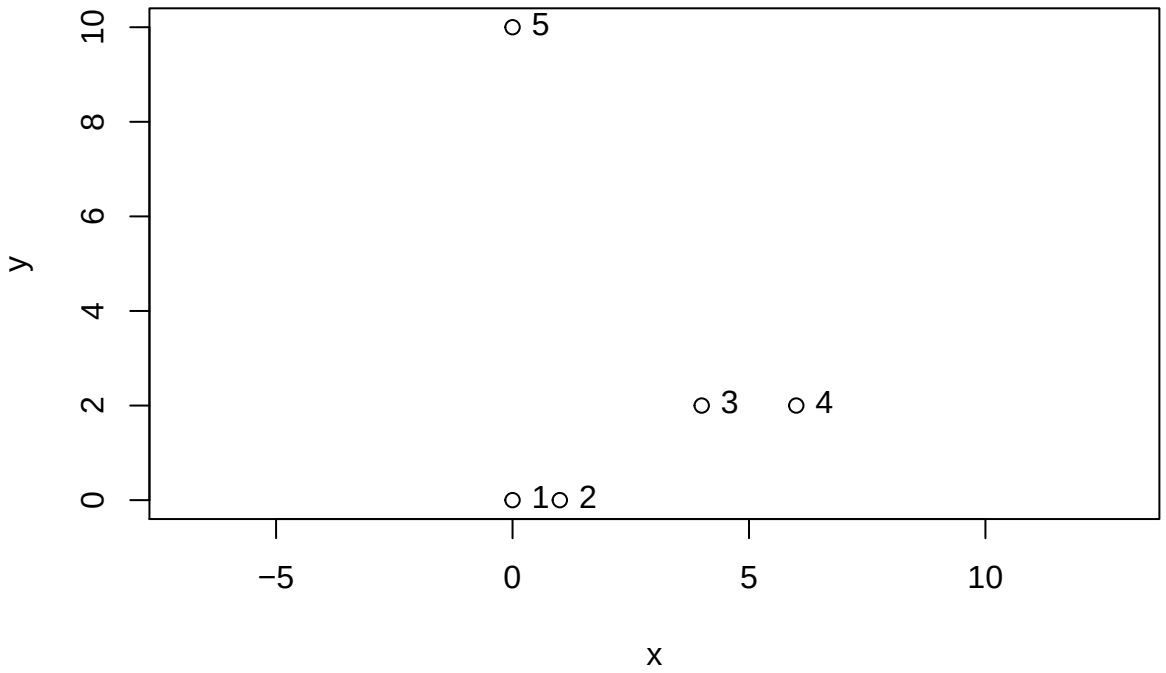
klasztert egyesítenek
- *diverzív*: az algoritmusok a teljes adathalmazból indulnak ki, azt egyetlen klaszterként kezelve, majd minden lépésben egy adott klasztert kettéosztanak (vagy több részre osztanak)

Itt most csak az agglomeratív megoldással fogunk foglalkozni. A hierarchikus csoportosítás előnye, hogy tetszőleges távolságdefinícióval működőképes. Az egyes klaszterek távolságára számos mércét ismerünk (ún. linkage módszer):

- complete: $\max\{\, d(a,b) : a \in A,\, b \in B \,\}$
- single: $\min\{\, d(a,b) : a \in A,\, b \in B \,\}$
- centroid: $\|c_s - c_t\|$
- average: $\frac{1}{|A|\cdot|B|} \sum_{a\in A} \sum_{b\in B} d(a,b)$
- Ward: minden egyes összeolvasztásnál cél, hogy a WCSS a lehető legkevesebbet növekedjen

Tekintsük a korábbi adathalmazunkat, és végezzük el rajta a hierachikus klaszterezést!

```
df<-data.frame(x=c(0,1,4,6,0),y=c(0,0,2,2,10))
plot(df,asp=1)
text(df,pos=4)
```

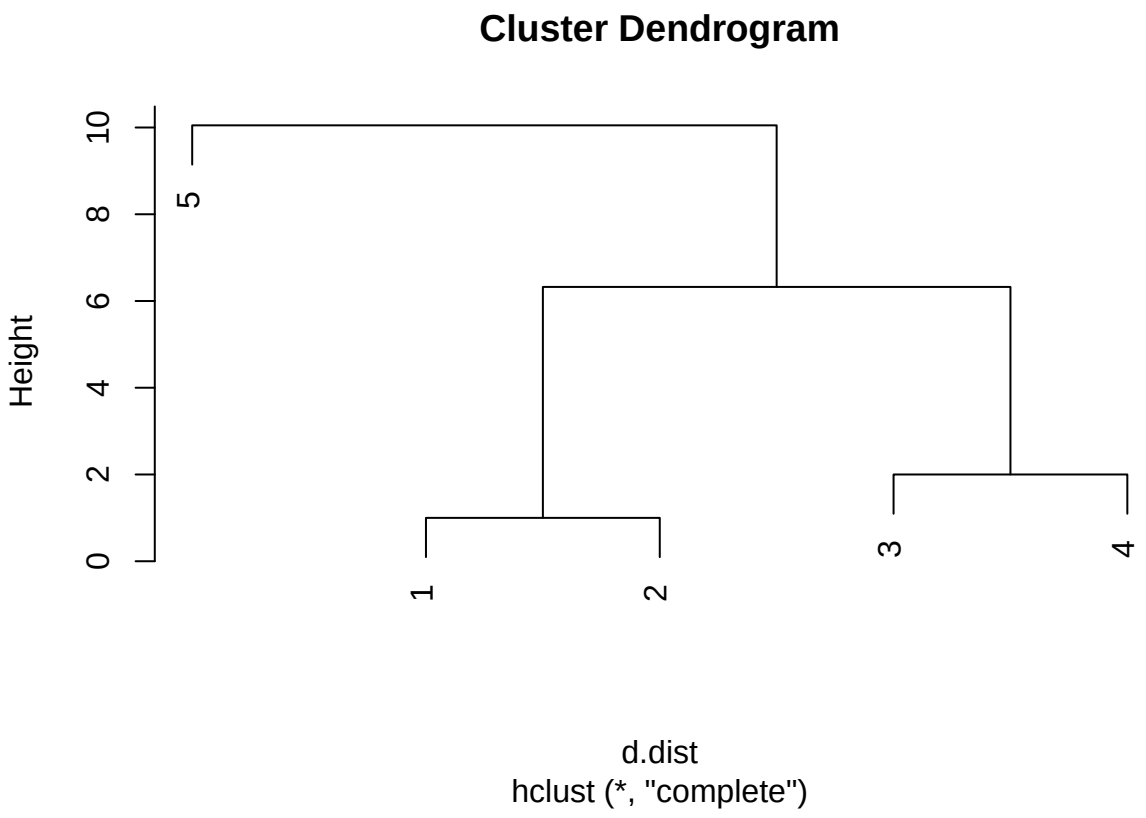


A hierarchikus klaszterezés megvalósításához szükség van az adatok távolság mátrixszára, amelyben az egyes sorok és oszlopok az adott adatsorok közötti távolságot adják meg. A távolság számításának módszere megválasztható, mi itt most az euklideszi távolsággal dolgozunk:

```
d.dist<-dist(df)
```

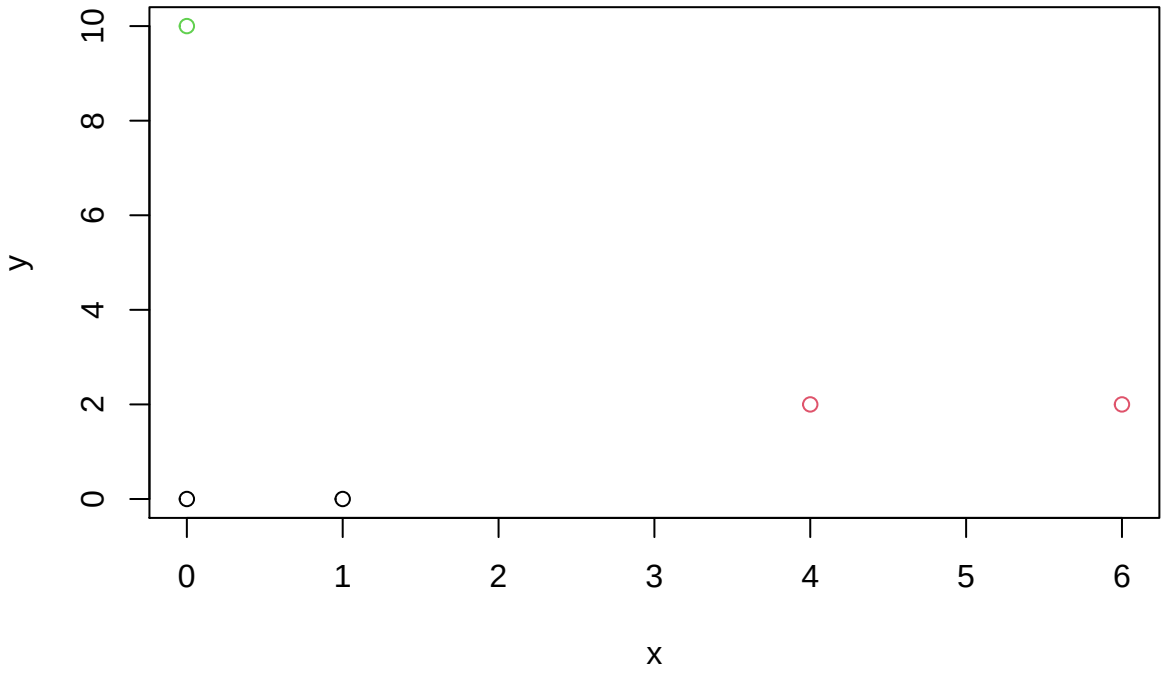
A távolságmátrix ismeretében elvégezhetjük a hierarchikus csoportosítást, valamint az eredmény megjelenítését dendogram formájában:

```
m<-hclust(d.dist)
plot(m)
```



A faszerkezetből a `cutree` paranccsal tudunk klasztereket képezni, megadva, hogy hány klasztert szeretnénk:

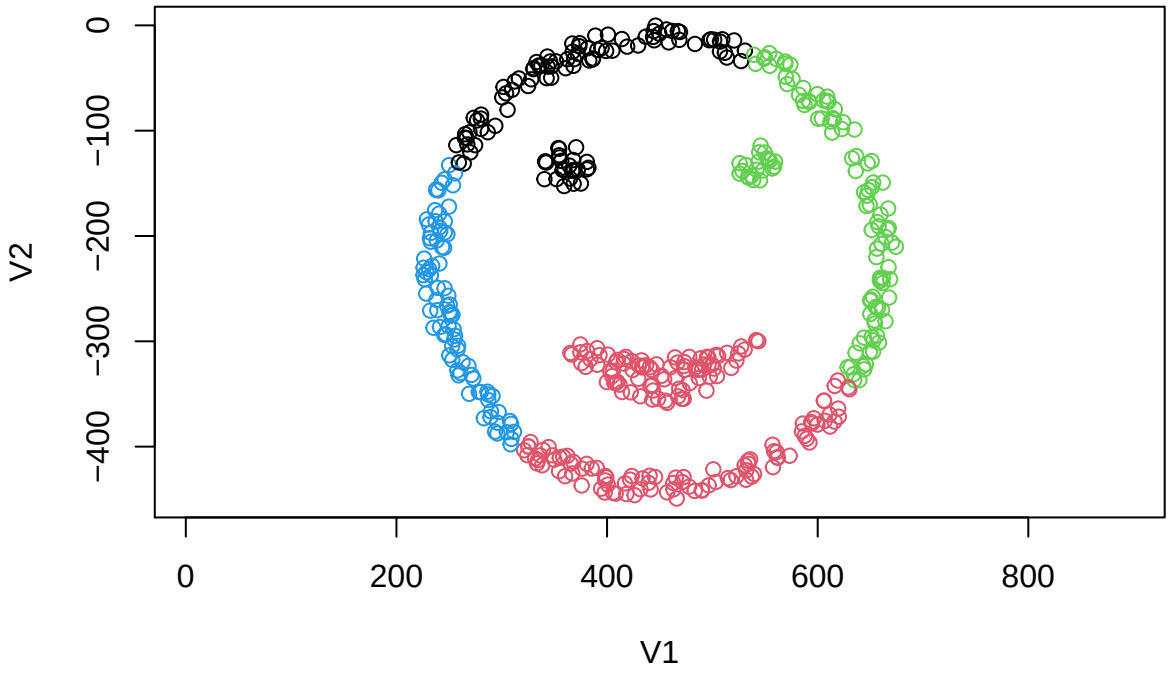
```
cl<-cutree(m,3)
plot(df,col=cl)
```



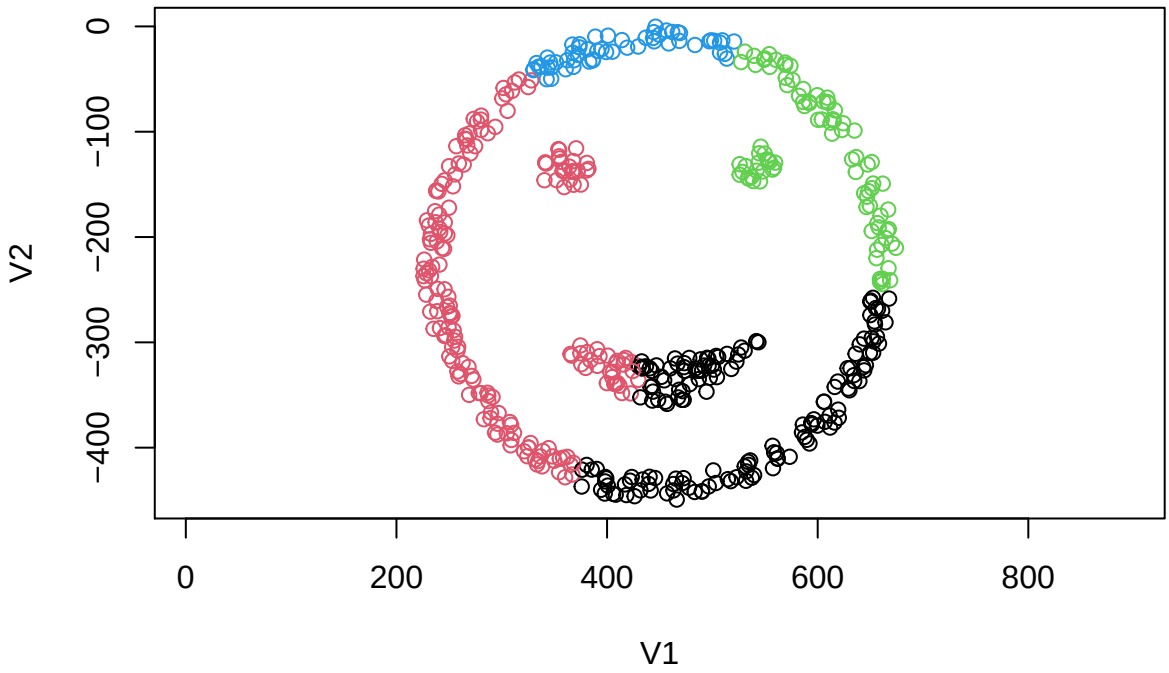
3.1 Metrikák

Érdekes összehasonlítást kapunk a speciális *smiley* adathalmazon az egyes metrikák alkalmazásával:

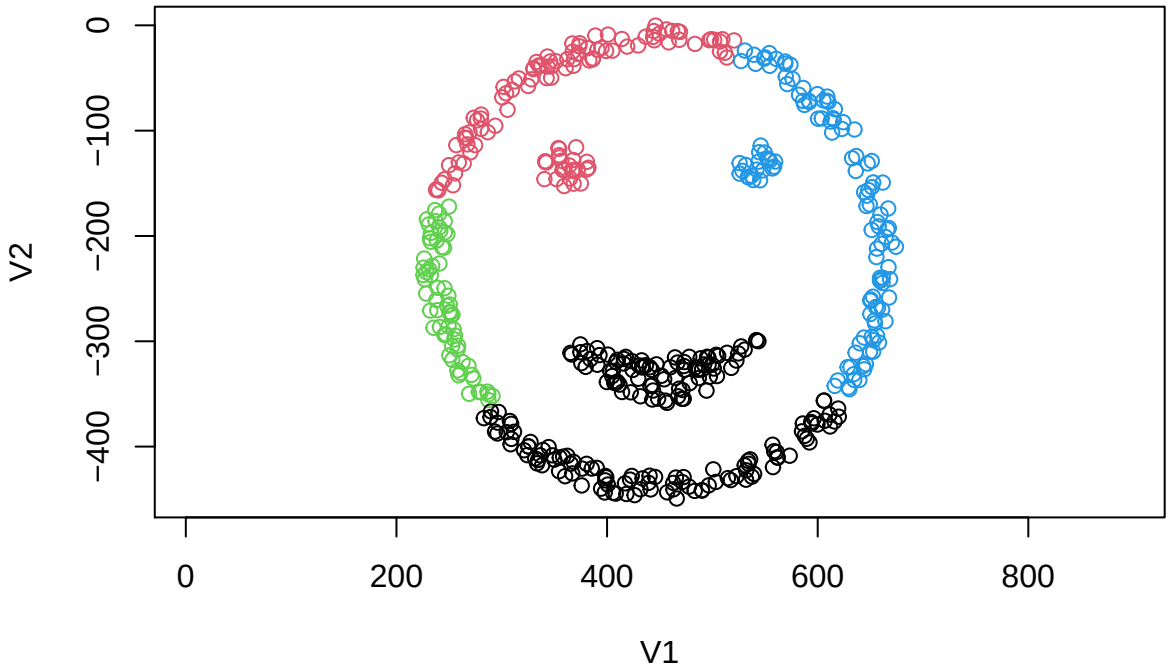
```
df<-read.csv('/opt/datasets/smiley.csv',header=F)
df$V2<-df$V2 * -1

m<-kmeans(df,4)
plot(df,col=m$cluster,asp=1)
```



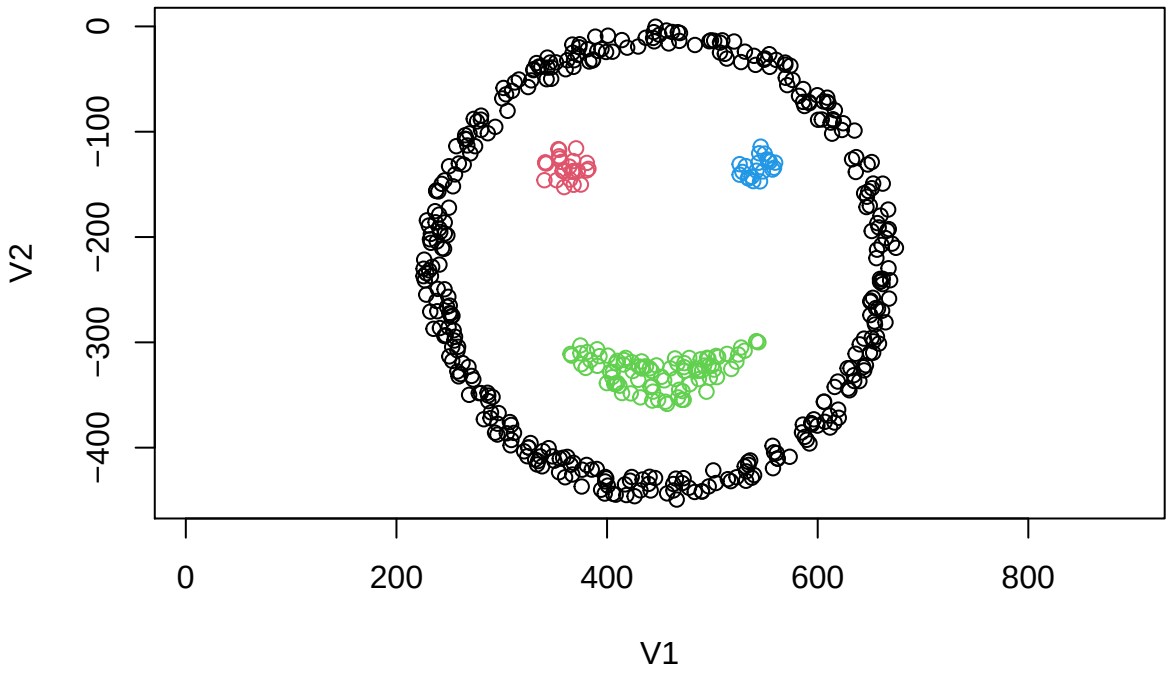

```
d<-dist(df)
h<-hclust(d,method='complete')
cl<-cutree(h,4)
plot(df,col=cl,asp=1)
```




```
h<-hclust(d,method='ward.D2')
cl<-cutree(h,4)
plot(df,col=cl,asp=1)
```




```
h<-hclust(d,method='single')
cl<-cutree(h,4)
plot(df,col=cl,asp=1)
```



4 Példák

A következőkben pár példát tekintünk át a klaszterezési eljárások bemutatására.

4.1 Fehérjefogyasztás

Az adathalmaz az egyes európai országok fehérjefogyasztásának lebontását tartalmazza az egyes fehérjeforrásokra. Töltsük be az adathalmazt!

```
data.file <- read.table("protein.txt", sep="\t", header = TRUE, row.names = 1)
head(data.file)
```

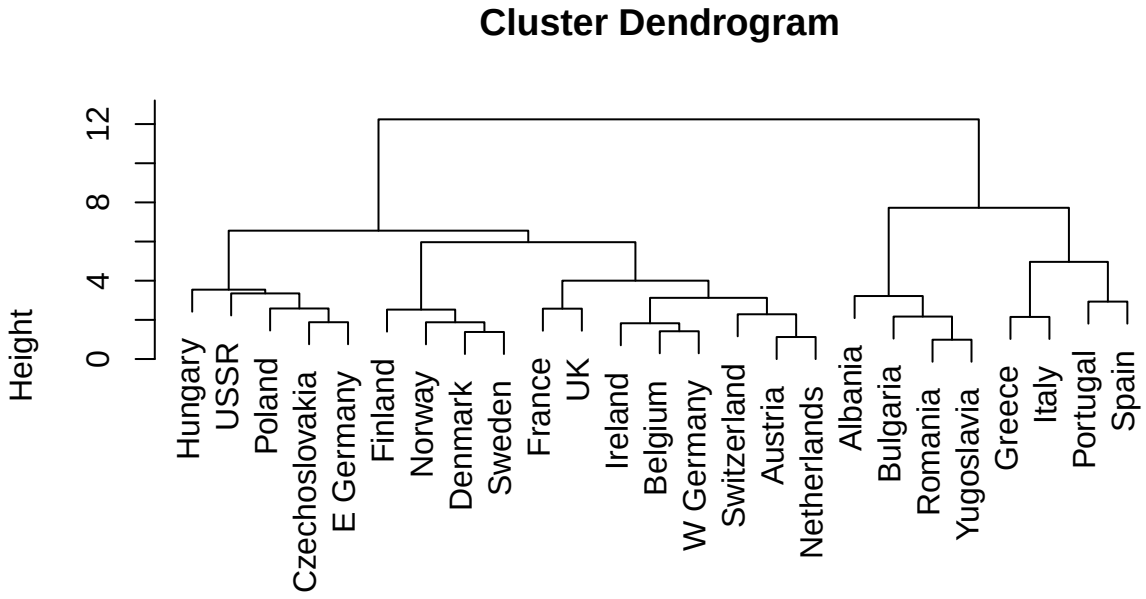
```
##                RedMeat WhiteMeat Eggs Milk Fish Cereals Starch Nuts Fr.Veg
## Albania           10.1       1.4  0.5  8.9  0.2    42.3    0.6  5.5    1.7
## Austria            8.9      14.0  4.3 19.9  2.1    28.0    3.6  1.3    4.3
## Belgium           13.5       9.3  4.1 17.5  4.5    26.6    5.7  2.1    4.0
## Bulgaria           7.8       6.0  1.6  8.3  1.2    56.7    1.1  3.7    4.2
## Czechoslovakia     9.7      11.4  2.8 12.5  2.0    34.3    5.0  1.1    4.0
## Denmark           10.6      10.8  3.7 25.0  9.9    21.9    4.8  0.7    2.4
```

Célszerű az adathalmazt normalizálni, azaz a zéróba tolni és a szórást minden tengely mentén standardizálni:

```
data <- scale(data.file)
```

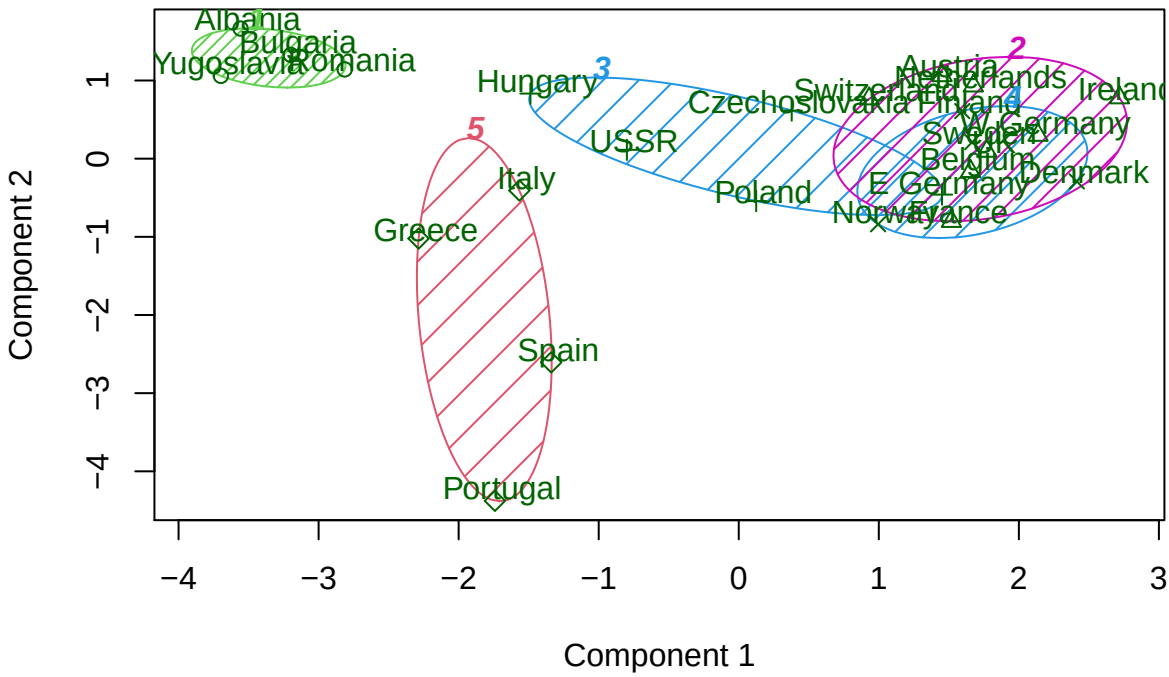
Ezután kirajzolhatjuk az egyes klasztereket, hierarchikus klaszterezést használva. A fát 5 mélységűre vágjuk, és az ehhez tartozó szintet klaszterábrán ábrázoljuk.

```
data.dist<-dist(data)
m<-hclust(data.dist,method='ward.D2')
plot(m)
```



data.dist
hclust (*, "ward.D2")

```
cl<-cutree(m,k=5)

clusplot(data,cl,color=TRUE, shade=TRUE,
   labels=2, lines=0)
```



These two components explain 62.68 % of the point variability.

```
cl[cl==1]
```

```
##    Albania   Bulgaria    Romania Yugoslavia
##          1          1          1          1
```

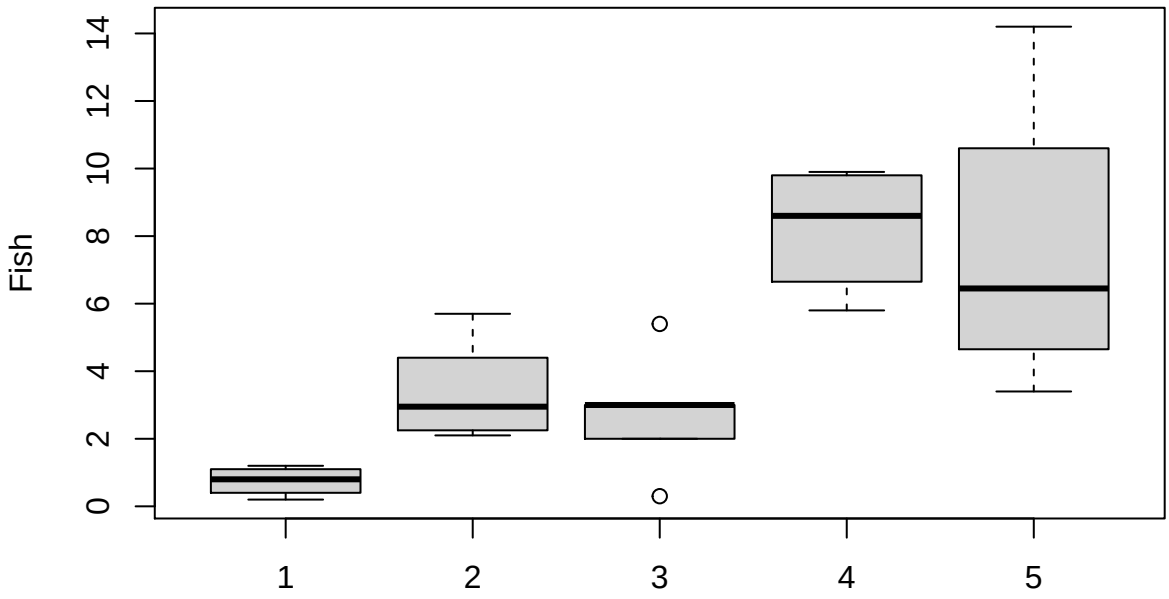
```
tapply(data.file$RedMeat,cl,mean)
```

```
##       1       2       3       4       5
##  7.1250 13.2125  7.9200  9.8500  8.1250
```

```
aggregate(data.file,list(C=cl),mean)
```

```
##   C RedMeat WhiteMeat   Eggs    Milk  Fish Cereals Starch   Nuts Fr.Veg
## 1 1  7.1250    4.6750 1.2000  9.4500 0.750  51.125  1.950 5.0500  2.975
## 2 2 13.2125   10.6375 3.9875 21.1625 3.375  24.700  4.650 2.0625  4.175
## 3 3  7.9200   10.0400 2.8400 13.8400 2.740  35.740  5.560 2.5400  4.260
## 4 4  9.8500    7.0500 3.1500 26.6750 8.225  22.675  4.550 1.1750  2.125
## 5 5  8.1250    3.8000 2.4750 11.2000 7.625  33.675  3.975 5.6750  7.075
```

```
boxplot(split(data.file$Fish,cl),ylab="Fish")
```

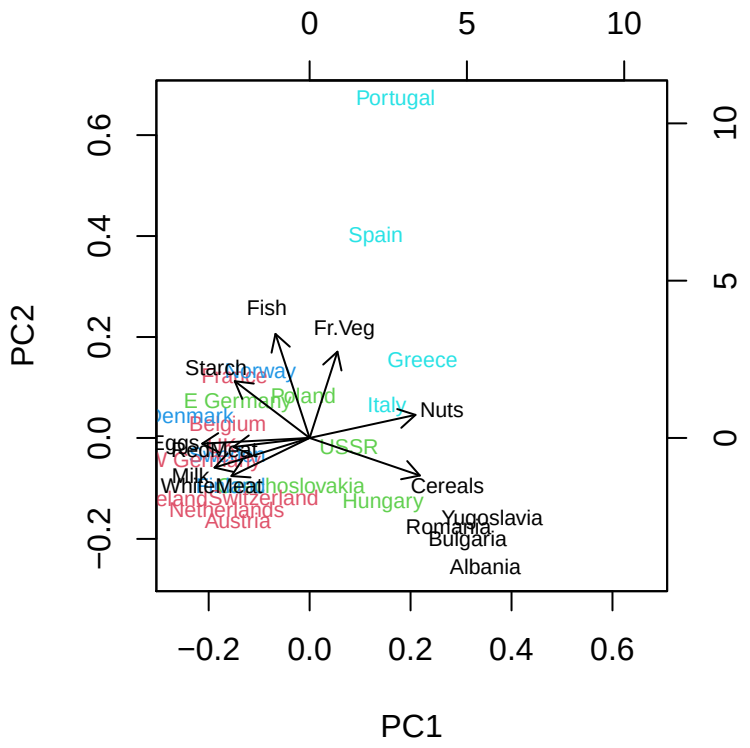


Végül, megtekintjük az adathalmaz PCA felbontását is.

```
p<-prcomp(data)
summary(p)
```

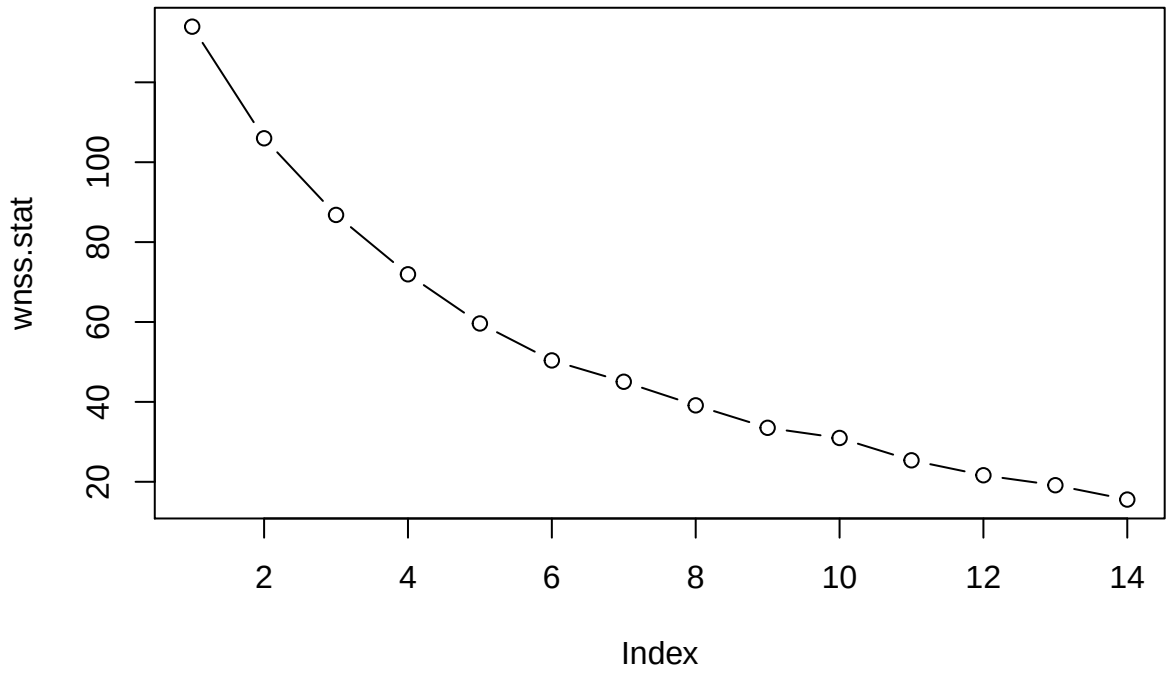
```
## Importance of components:
##                           PC1    PC2    PC3    PC4     PC5     PC6     PC7
## Standard deviation     2.0016 1.2787 1.0620 0.9771 0.68106 0.57020 0.52116
## Proportion of Variance 0.4452 0.1817 0.1253 0.1061 0.05154 0.03613 0.03018
## Cumulative Proportion  0.4452 0.6268 0.7521 0.8582 0.90976 0.94589 0.97607
##                           PC8     PC9
## Standard deviation     0.34102 0.31482
## Proportion of Variance 0.01292 0.01101
## Cumulative Proportion  0.98899 1.00000
```

```
coloredBiplot(p,col=1,cex=0.7,
              xlabs.col=cl)
```



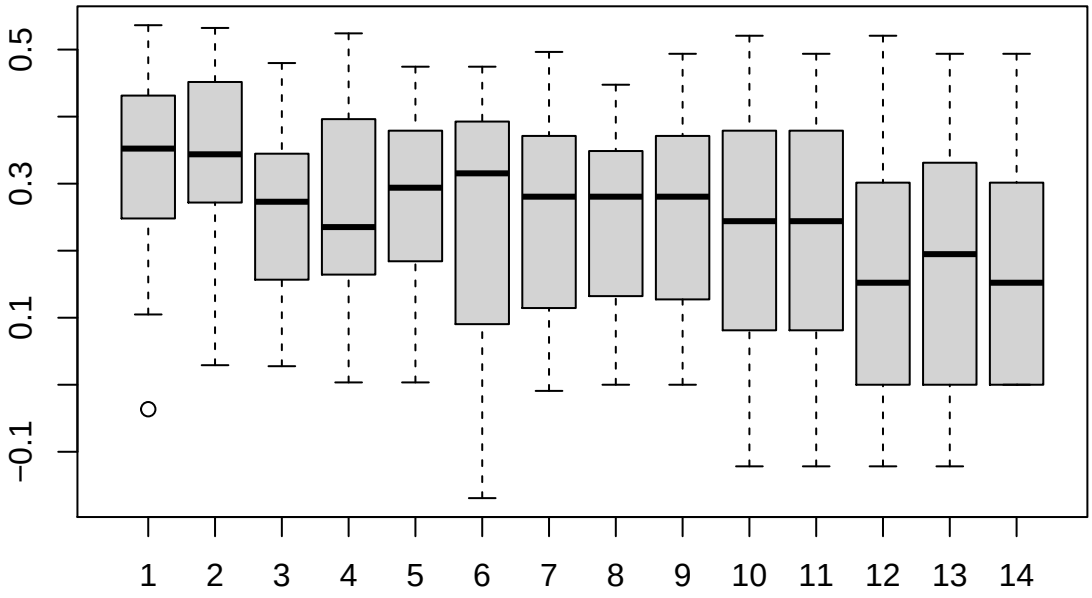
A kmeans algoritmus használatához tekintsük a Total Within NSS értékeket a klaszterszámtól függően, 2-től 14-ig:

```
f<-function(k) {
  m<-kmeans(data,k,nstart=10,iter.max=50)
  m$tot.withinss
}
wnss.stat<-sapply(2:15,f)
plot(wnss.stat,type='b')
```



Sajnos, ahogy az ábrán is látjuk, tipikus törés nem található, a metrika egyenletesen csökken.

```
f<-function(k) {
  m<-kmeans(data,k,nstart=10,iter.max=50)
  sc<-silhouette(m$cluster,dist(data))
  sc[,3]
}
sc.stat<-sapply(2:15,f)
boxplot(sc.stat)
```



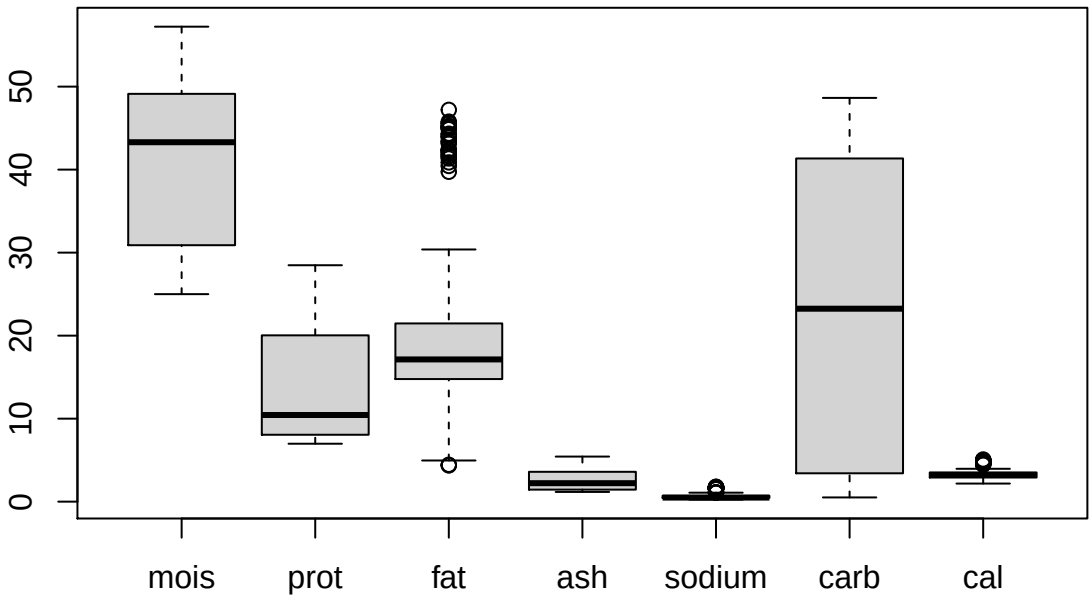
4.2 Pizza adatbázis

Az adatbázis az egyes pizzatípusok tápanyagösszetételét mutatja. Töltsük be az adathalmazt:

```
df<-read.csv("Pizza.csv",stringsAsFactors = T)
summary(df)
```

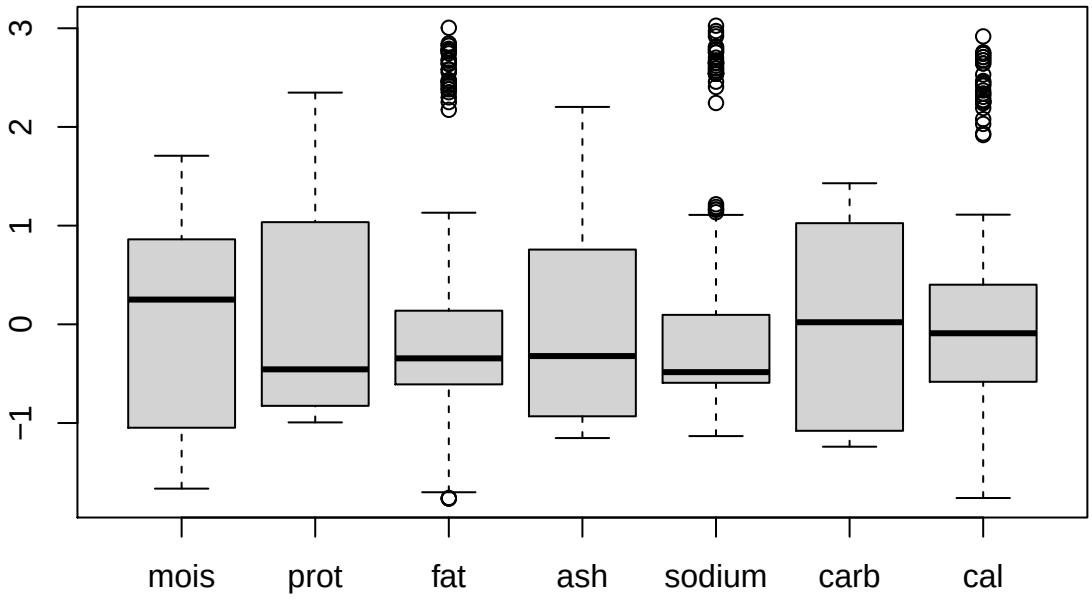
```
##      brand           id             mois            prot            fat
##  H      : 33   Min.   :14003   Min.   :25.00   Min.   : 6.98   Min.   : 4.38
##  D      : 32   1st Qu.:14094   1st Qu.:30.90   1st Qu.: 8.06   1st Qu.:14.77
##  J      : 32   Median :24020   Median :43.30   Median :10.44   Median :17.14
##  B      : 31   Mean   :20841   Mean   :40.90   Mean   :13.37   Mean   :20.23
##  F      : 30   3rd Qu.:24110   3rd Qu.:49.12   3rd Qu.:20.02   3rd Qu.:21.43
##  A      : 29   Max.   :34045   Max.   :57.22   Max.   :28.48   Max.   :47.20
##  (Other):113
##       ash             sodium            carb             cal
##  Min.   :1.170   Min.   :0.2500   Min.   : 0.510   Min.   :2.180
##  1st Qu.:1.450   1st Qu.:0.4500   1st Qu.: 3.467   1st Qu.:2.910
##  Median :2.225   Median :0.4900   Median :23.245   Median :3.215
##  Mean   :2.633   Mean   :0.6694   Mean   :22.865   Mean   :3.271
##  3rd Qu.:3.592   3rd Qu.:0.7025   3rd Qu.:41.337   3rd Qu.:3.520
##  Max.   :5.430   Max.   :1.7900   Max.   :48.640   Max.   :5.080
##
```

```
boxplot(df[-(1:2)])
```



Célszerű boxplot segítségével ábrázolni az adathalmazt, valamint PCA felbontás után biplot segítségével is ábrázolni azt.

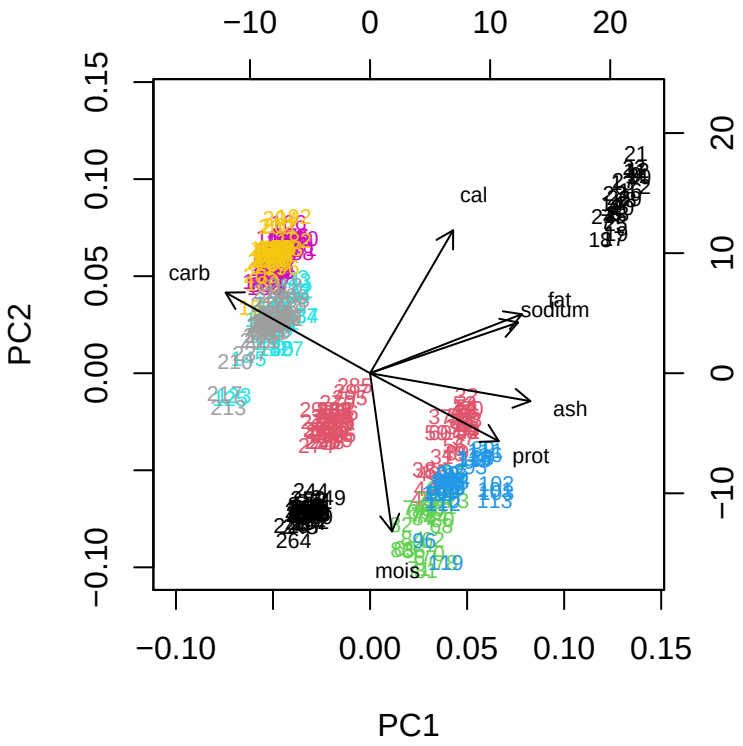
```
data<-df[-(1:2)]
data<-scale(data)

boxplot(data)
```




```
p<-prcomp(data)
summary(p)
```

```
## Importance of components:
##                           PC1    PC2     PC3    PC4     PC5     PC6      PC7
## Standard deviation     2.042 1.5134 0.64387 0.3085 0.16636 0.01837 0.003085
## Proportion of Variance 0.596 0.3272 0.05922 0.0136 0.00395 0.00005 0.000000
## Cumulative Proportion  0.596 0.9232 0.98240 0.9960 0.99995 1.00000 1.000000
```

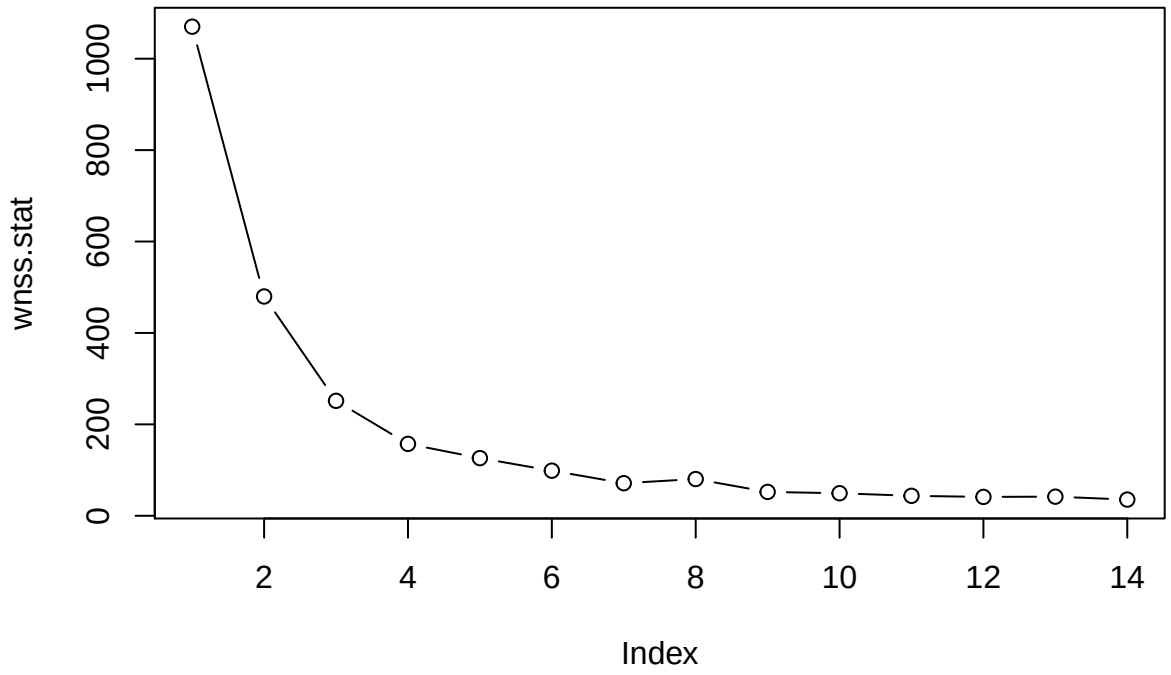
```
coloredBiplot(p,col=1,cex=0.7,
              xlabs.col=as.numeric(df$brand))
```



Jól látható, hogy az egyes pizzák tipikus csoportokra bomlanak tápanyagösszetétel szerint. Azt is érdemes észrevenni, hogy a legerősebb korreláció a zsírtartalom és a nátriumtartalom között van.

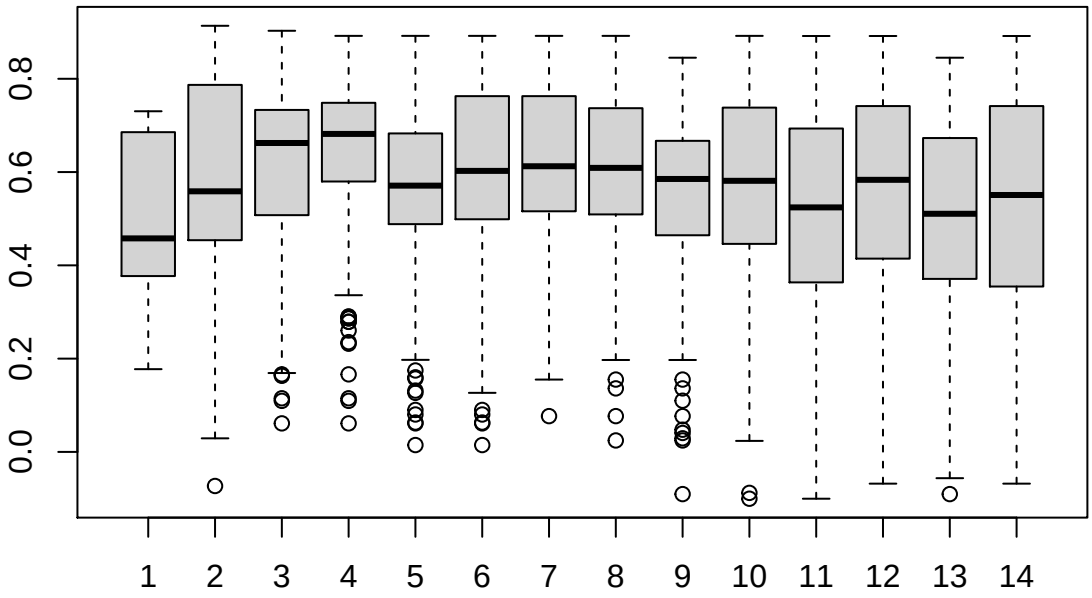
K-közép csoportosításhoz tekintsük a Total Within SS értéket 2-től 14 klaszterig:

```
f<-function(k) {
  m<-kmeans(data,k,nstart=10,iter.max=50)
  m$tot.withinss
}
wnss.stat<-sapply(2:15,f)
plot(wnss.stat,type="b")
```



Ugyanígy, a sziluett együtthatót is tekinthetjük:

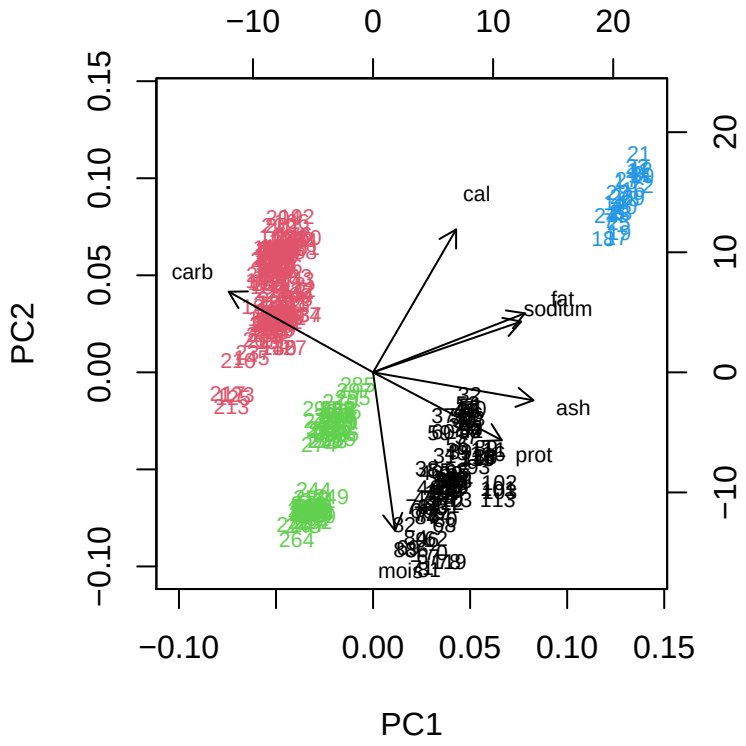
```
f<-function(k) {
  m<-kmeans(data,k,nstart=10,iter.max=50)
  sc<-silhouette(m$cluster,dist(data))
  sc[,3]
}
sc.stat<-sapply(2:15,f)
boxplot(sc.stat)
```



A tipikus klaszterszám a két mérce alapján 3 és 5 közé tehető, ábrázoljuk 4 klaszterre:

```
m<-kmeans(data,4)

coloredBiplot(p,col=1,cex=0.7,
```

```
              xlabs.col=m$cluster)
```



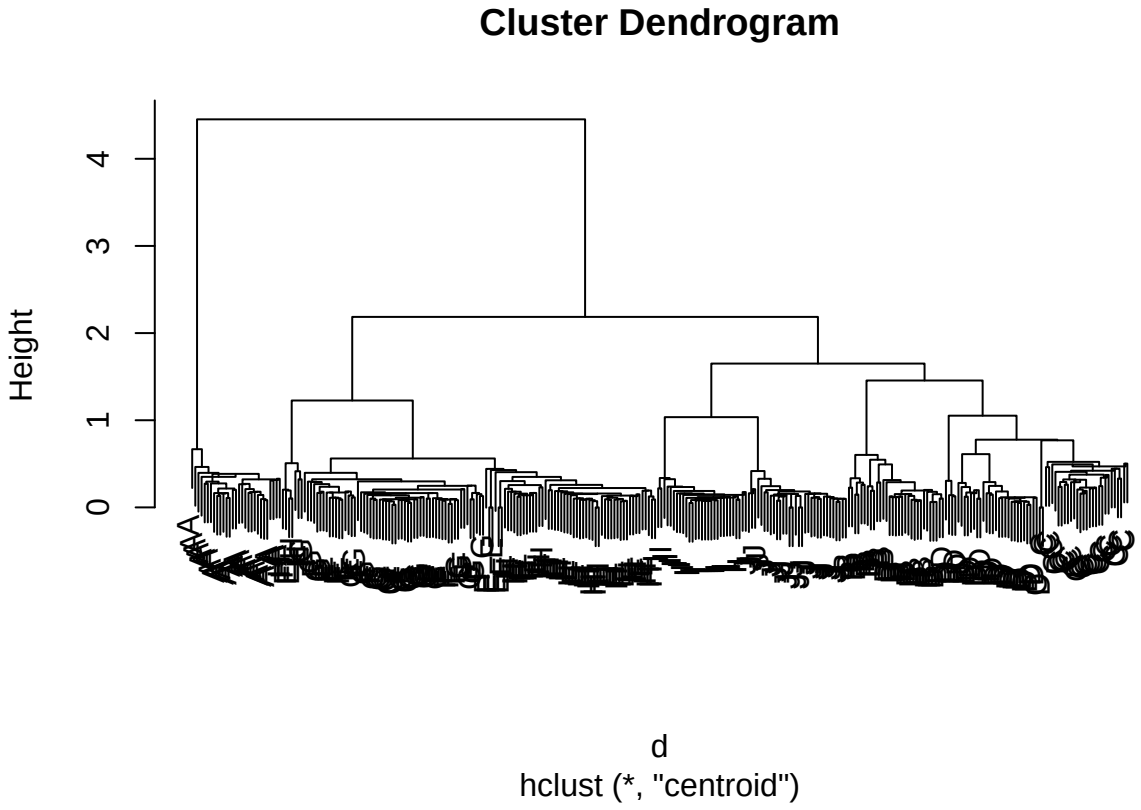
Melyik márkák vannak egy csoportban?

```
tapply(df$brand,m$cluster,unique)
```

```
## $`1`
## [1] B C D
## Levels: A B C D E F G H I J
##
## $`2`
## [1] E F G H
## Levels: A B C D E F G H I J
##
## $`3`
## [1] I J
## Levels: A B C D E F G H I J
##
## $`4`
## [1] A
## Levels: A B C D E F G H I J
```

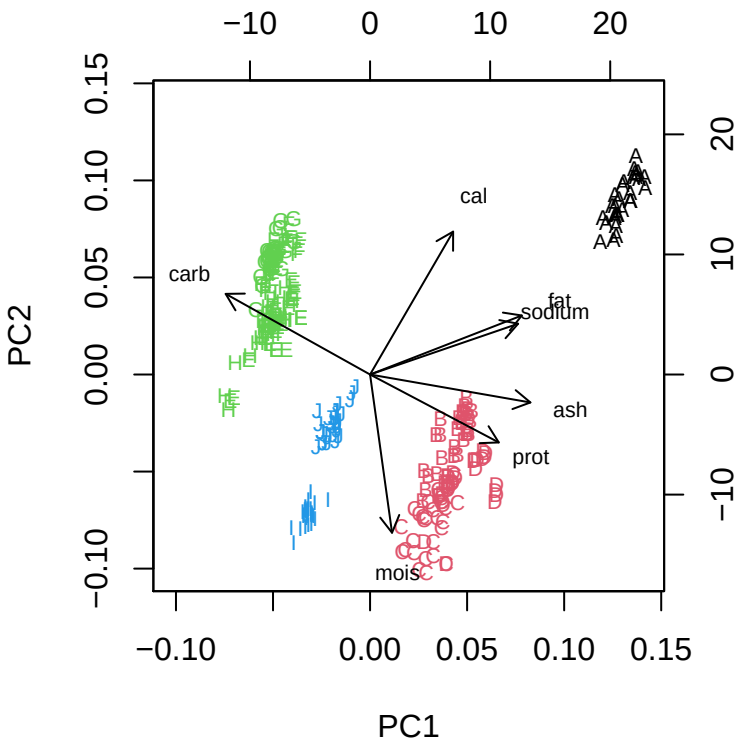
Készítsünk hierarchikus csoportosítást! A dendogramot ábrázoljuk úgy, hogy a leveleken a pizza márkája szerepeljen!

```
rownames(data)<-df$brand
d<-dist(data)
h<-hclust(d,method='centroid')

plot(h)
```



Csináljunk négy csoportot, és ábrázoljuk őket a biplot segítségével!

```
cl<-cutree(h,4)

coloredBiplot(p,col=1,cex=0.7,
              xlabs.col=cl,xlabs=df$brand)
```


−10
0
10
20
0.15
0.10
0.05
0.00
−0.10
20
10
0
−10
PC2
cal
carb
fat
sodium
ash
prot
mois
−0.10
0.00
0.05
0.10
0.15
PC1