Class 7: Machine Learning (Part I)

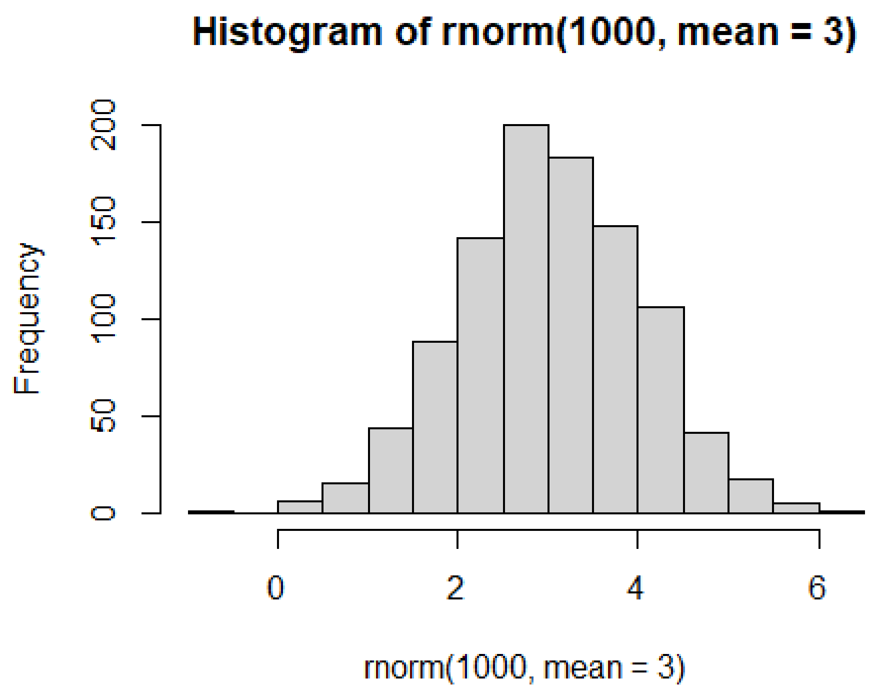
Mirte Ciz Marieke Kuijpers

09/02/2022

R norm function

```
# Investigate the rnorm() function using the help pages.
#?rnorm

# Test the rnorm() function
rnorm(10)

##  [1]  0.6320498  0.5083814  0.4491256  0.3789141  0.9588942  0.5821017
##  [7]  0.9119206  0.1474662 -0.4209862 -0.1795772

# Use some of the arguments with a default set value
rnorm(100, mean = 3)

##   [1] 3.0262402 2.6299905 3.2622461 2.4022800 3.5386113 1.6395142
3.2949997
##   [8] 3.4012802 4.3559156 0.6049153 3.3600162 2.5665774 3.0781990
2.9127685
##  [15] 2.9013585 3.7502652 1.5279054 4.4588865 4.0197098 3.0441384
3.0922703
##  [22] 0.9805644 3.6809898 2.5264373 1.8704351 2.4913367 2.9789281
3.2007072
##  [29] 1.8154686 1.8703356 3.5268392 2.5544735 4.1056631 2.8156858
2.8800292
##  [36] 3.1972025 2.9033934 4.5016124 4.4345655 2.7010116 3.0784997
3.6179135
##  [43] 3.4852368 2.7462454 3.3365119 0.7203355 4.3549133 4.4219001
2.4885448
##  [50] 3.8063984 1.8488879 3.6554320 3.5825168 1.7183210 4.4553130
2.1283181
##  [57] 4.5323385 2.1935225 3.3282061 4.1126624 3.2251278 3.0837277
1.6541894
##  [64] 1.6838106 2.2769864 3.0699750 2.3770338 3.0572761 2.7988761
3.0655744
##  [71] 3.2124347 3.0932181 4.9577385 4.8312037 3.5127439 1.4774555
2.4118400
##  [78] 3.1249519 3.3575152 2.8235404 3.4467317 3.3098436 2.3999610
3.5286238
##  [85] 2.8675116 5.4932577 3.0770147 2.0481685 2.2954452 2.2141610
3.3345663
##  [92] 4.3857712 2.2712687 1.9726711 2.5609732 2.6480138 2.1644163
```

```
1.3125086
##  [99] 1.7599988 1.4430903

# Plot some of these results

hist(rnorm(1000, mean = 3))
```



Clustering methods

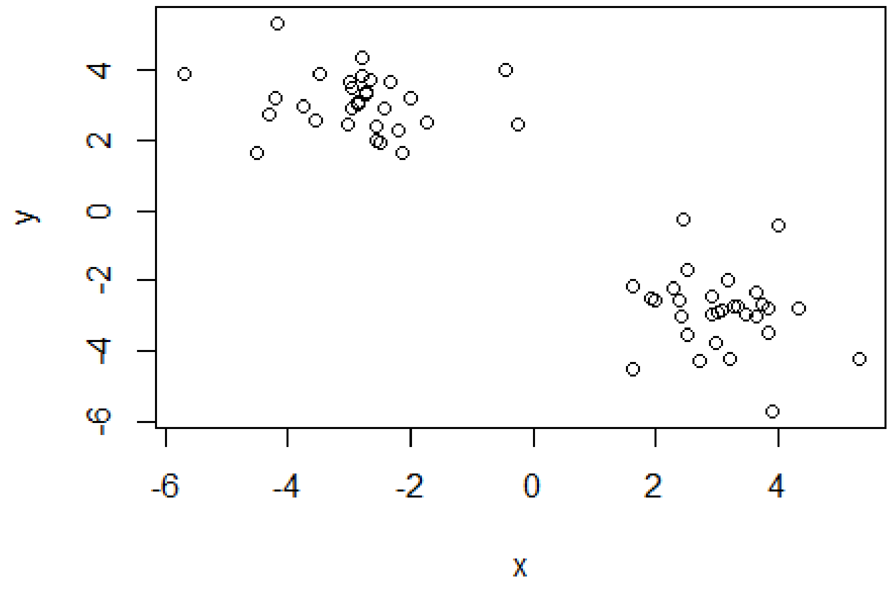
Using the kmeans() function

Now that we understand rnorm()'s function, we can proceed to the example usage of kmeans(). The first step is to generate some example data to test the kmeans() function with.

```
# Generate some random data
tmp <- c(rnorm(30, -3), rnorm(30, 3))
tmp

##  [1] -3.7383801 -2.8003140 -2.9886336 -4.3003321 -4.1929166 -1.9993546
##  [7] -2.5673048 -2.7329298 -4.5019548 -2.9601035 -2.1283271 -2.5672405
## [13] -0.4471940 -1.7191247 -2.8732282 -3.0149166 -2.9725277 -5.7098859
## [19] -4.1966651 -2.4798919 -2.7376195 -3.4982036 -2.6593251 -2.7932877
## [25] -3.5358799 -2.8704816 -2.3201715 -2.2025285 -2.4444723 -0.2572751
## [31]  2.4684921  2.9315847  2.2891036  3.6618893  3.0895716  2.5382916
## [37]  4.3544182  3.7370859  3.8584310  3.2821297  1.9264906  3.2177560
## [43]  3.8987588  2.9137047  2.4345433  3.0158711  2.5097729  4.0051310
## [49]  2.3954122  1.6249559  3.4951928  1.6295588  3.3609248  2.0042986
## [55]  3.1851958  5.3235959  2.7090940  3.6548452  3.8324667  2.9841088
```

```
# Format the random data to be used by kmeans()
x <- cbind(x=tmp, y=rev(tmp))
x
```

```
##                x          y
##  [1,] -3.7383801  2.9841088
##  [2,] -2.8003140  3.8324667
##  [3,] -2.9886336  3.6548452
##  [4,] -4.3003321  2.7090940
##  [5,] -4.1929166  5.3235959
##  [6,] -1.9993546  3.1851958
##  [7,] -2.5673048  2.0042986
##  [8,] -2.7329298  3.3609248
##  [9,] -4.5019548  1.6295588
## [10,] -2.9601035  3.4951928
## [11,] -2.1283271  1.6249559
## [12,] -2.5672405  2.3954122
## [13,] -0.4471940  4.0051310
## [14,] -1.7191247  2.5097729
## [15,] -2.8732282  3.0158711
## [16,] -3.0149166  2.4345433
## [17,] -2.9725277  2.9137047
## [18,] -5.7098859  3.8987588
## [19,] -4.1966651  3.2177560
## [20,] -2.4798919  1.9264906
## [21,] -2.7376195  3.2821297
## [22,] -3.4982036  3.8584310
## [23,] -2.6593251  3.7370859
## [24,] -2.7932877  4.3544182
## [25,] -3.5358799  2.5382916
## [26,] -2.8704816  3.0895716
## [27,] -2.3201715  3.6618893
## [28,] -2.2025285  2.2891036
## [29,] -2.4444723  2.9315847
## [30,] -0.2572751  2.4684921
## [31,]  2.4684921 -0.2572751
## [32,]  2.9315847 -2.4444723
## [33,]  2.2891036 -2.2025285
## [34,]  3.6618893 -2.3201715
## [35,]  3.0895716 -2.8704816
## [36,]  2.5382916 -3.5358799
## [37,]  4.3544182 -2.7932877
## [38,]  3.7370859 -2.6593251
## [39,]  3.8584310 -3.4982036
## [40,]  3.2821297 -2.7376195
## [41,]  1.9264906 -2.4798919
## [42,]  3.2177560 -4.1966651
## [43,]  3.8987588 -5.7098859
## [44,]  2.9137047 -2.9725277
## [45,]  2.4345433 -3.0149166
## [46,]  3.0158711 -2.8732282
## [47,]  2.5097729 -1.7191247
## [48,]  4.0051310 -0.4471940
## [49,]  2.3954122 -2.5672405
## [50,]  1.6249559 -2.1283271
## [51,]  3.4951928 -2.9601035
## [52,]  1.6295588 -4.5019548
## [53,]  3.3609248 -2.7329298
## [54,]  2.0042986 -2.5673048
## [55,]  3.1851958 -1.9993546
## [56,]  5.3235959 -4.1929166
## [57,]  2.7090940 -4.3003321
## [58,]  3.6548452 -2.9886336
## [59,]  3.8324667 -2.8003140
## [60,]  2.9841088 -3.7383801

# Plot data to get a feel for the distribution
plot(x)
```

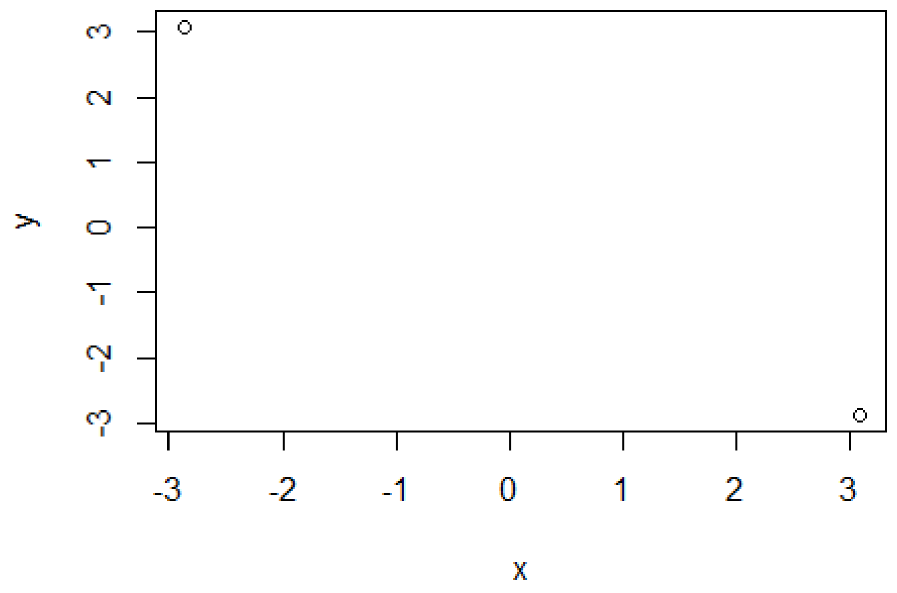


Note that the data above is randomly generated each time the code chunk is run. Therefore, the plot and all answers relying on this data will vary slightly each time the code is re-initialised. Now we can use the kmeans() function to cluster this random data.

```
# Cluster the data using the kmeans method
k <- kmeans(x, centers = 2, nstart = 10) # nstarts represents number of new
starts (i.e. iterations) to run
k

## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -2.873682  3.077756
## 2  3.077756 -2.873682
##
## Clustering vector:
##  [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 2 2 2 2 2
2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 54.80616 54.80616
##  (between_SS / total_SS =  90.6 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"
"tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

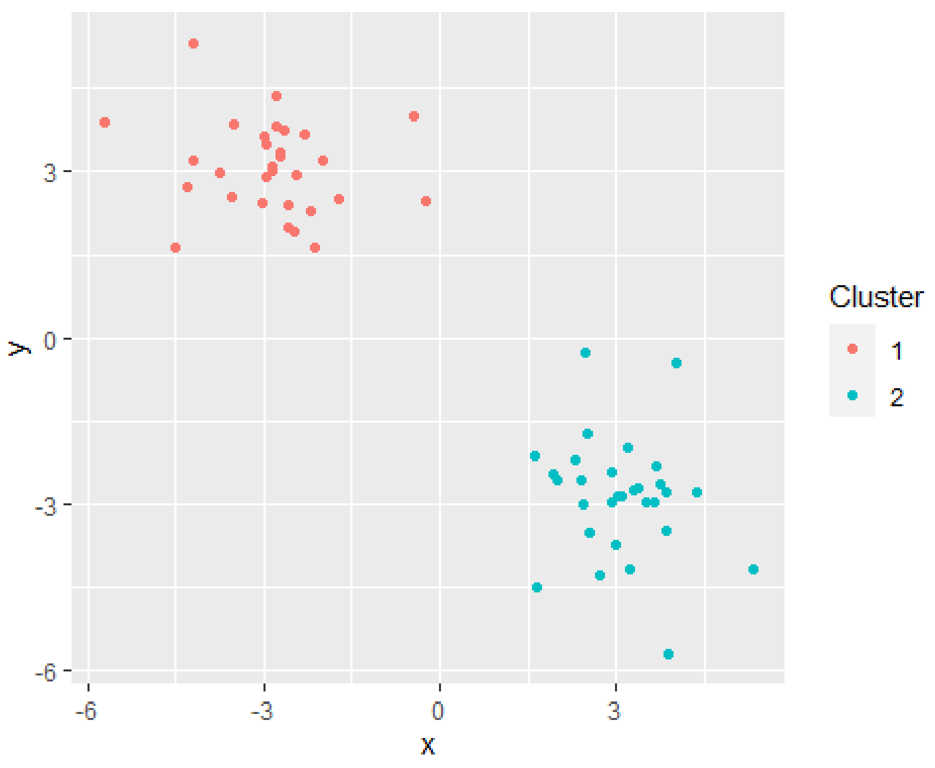
If we want to know how many points are in each cluster use the size values within k. Other interesting data can also be extracted from the variable containing the kmeans() results.

```
# Find the number of points in each group
k$size

## [1] 30 30

# To find the centroids use:
k$centers

##           x         y
## 1 -2.873682  3.077756
## 2  3.077756 -2.873682

# You can also plot them
plot(k$centers)
```

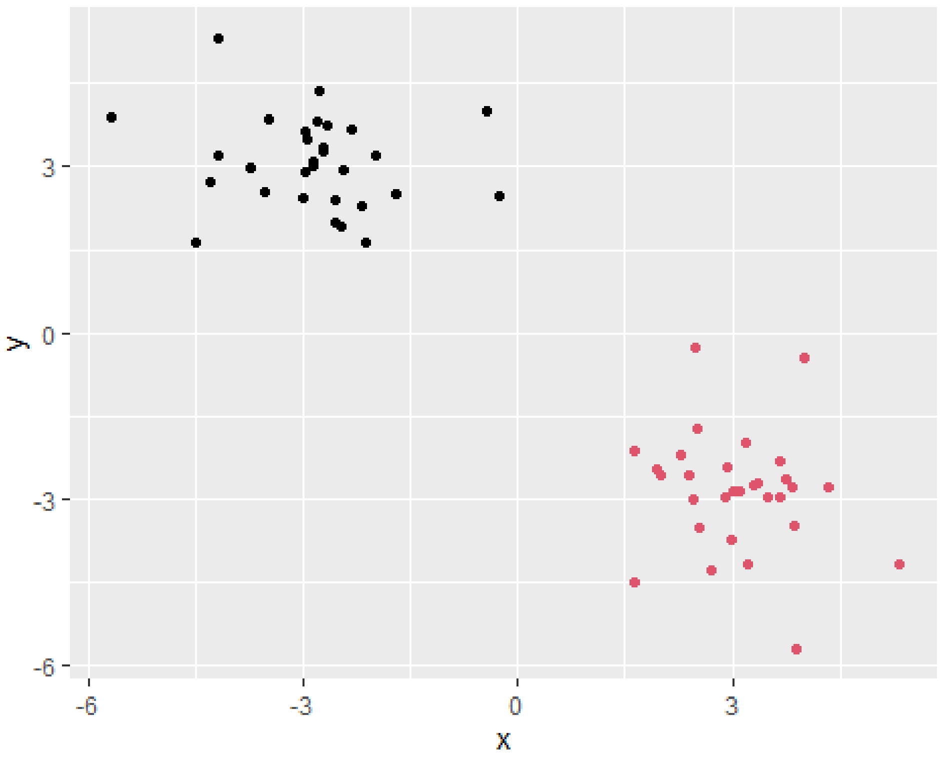


The membership vector, assigning each point to a cluster, is perhaps the most important results, as it allows you to analyse how your points are clustered.

```
# Print the cluster vector
k$cluster

##  [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 2 2 2 2 2
2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

# Plot the points coloured by these clusters

# Load ggplot2 library
library(ggplot2)

# Put data into a format ggplot can use
dat <- as.data.frame(cbind(x, k$cluster))
colnames(dat) <- c("x", "y", "cluster")

# Plot the data
ggplot(dat, aes(x, y, colour = as.factor(cluster))) +
  geom_point() +
  labs(colour = "Cluster")
```



```r
# In actual fact, one does not need to consolidate the data, one can simply
use k$cluster outside the aes(), provided that k$cluster and x stay in the
same order, the data will remain correctly correlated with the colours
ggplot(as.data.frame(x), aes(x,y)) +
  geom_point(col = k$cluster)
```



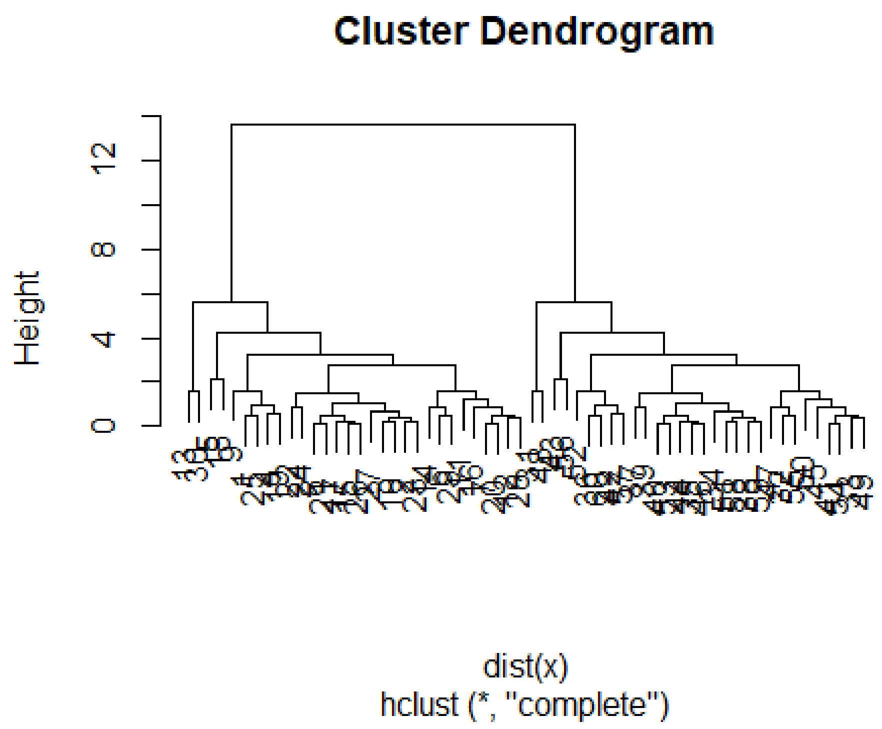
hclust() function

kmeans() uses Euclidean distance, hclust can use various similarity/difference measures, which is useful, but requires extra input. hclust() requires data already as distances between points (a dissimilarity/similarity matrix), not the raw data. One way to do so is to use the dist() function.

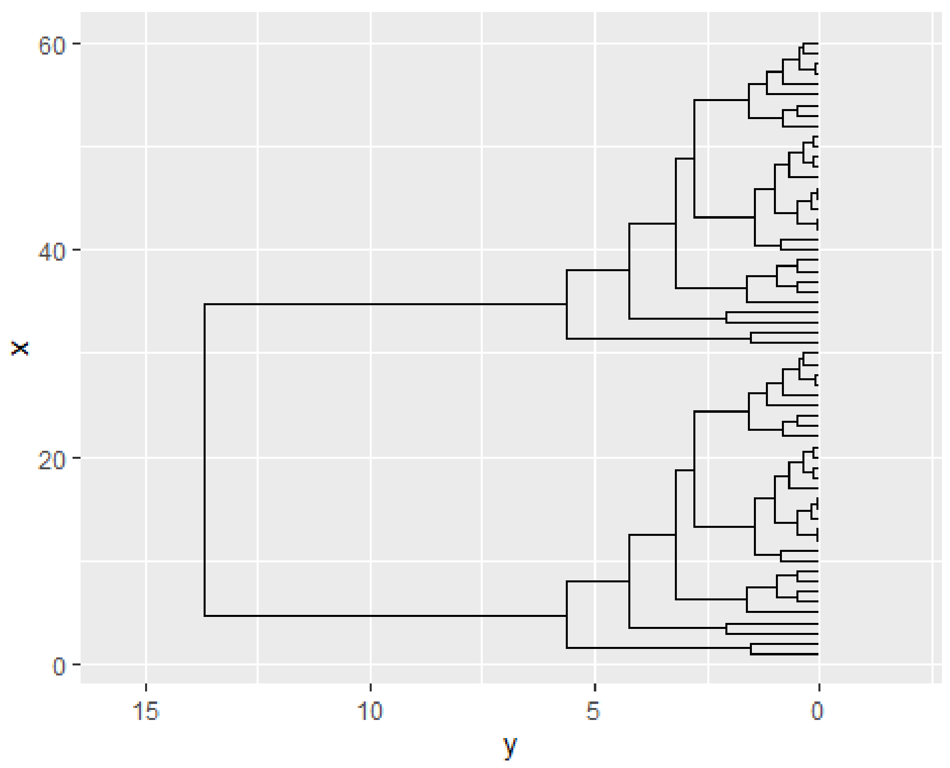
```
# Use the hclust() function on the same data (x) used for investigating the
kmeans() function
hc <- hclust(dist(x))
hc

##
## Call:
## hclust(d = dist(x))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

While the output for hclust(), when printed, is not as useful as for kmeans(), it has a custom plot method which helps interpret the data. In the dendrogram produced, the height of the crossbars represents (or is proportional to) the distance between the two groups joined by said crossbar.

```
# Plot hclust() data, as printing the output is not as useful as for kmeans()
plot(hc)
```

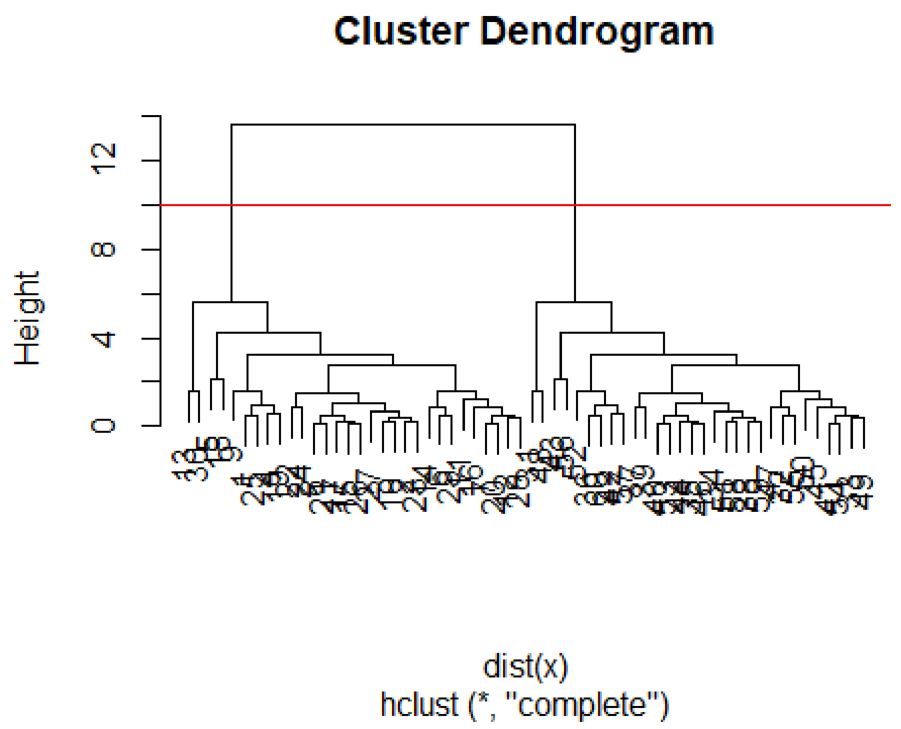


```
# One can also plot with ggplot, if a wrapper for dendrograms is also loaded:
library(ggdendro)

# Convert the data into a format ggplot can use - code borrowed from the
following tutorial https://cran.r-
project.org/web/packages/ggdendro/vignettes/ggdendro.html
ghc <- as.dendrogram(hc)

ghc.dat <- dendro_data(ghc, type = "rectangle")

ggplot(segment(ghc.dat)) +
  geom_segment(aes(x = x, y = y, xend = xend, yend = yend)) +
  coord_flip() +
  scale_y_reverse(expand = c(0.2, 0))
```



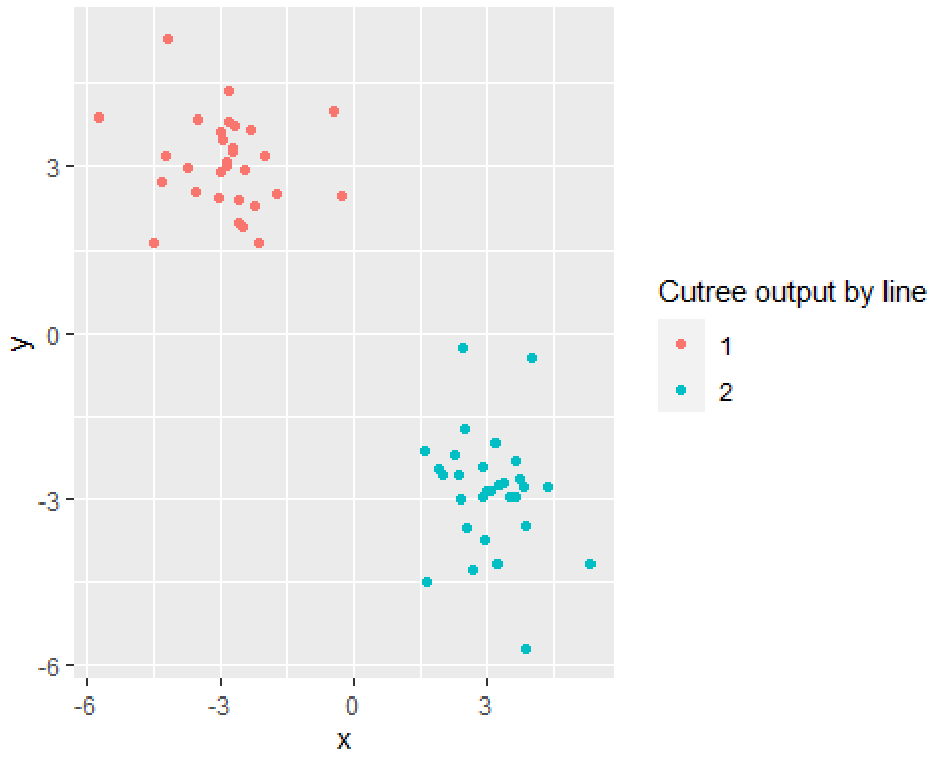
To determine groups, one can set a horizontal cut-off line, which separates points connected below that cut-off into clusters.

```
plot(hc)
abline(h = 10, col = "Red")
```

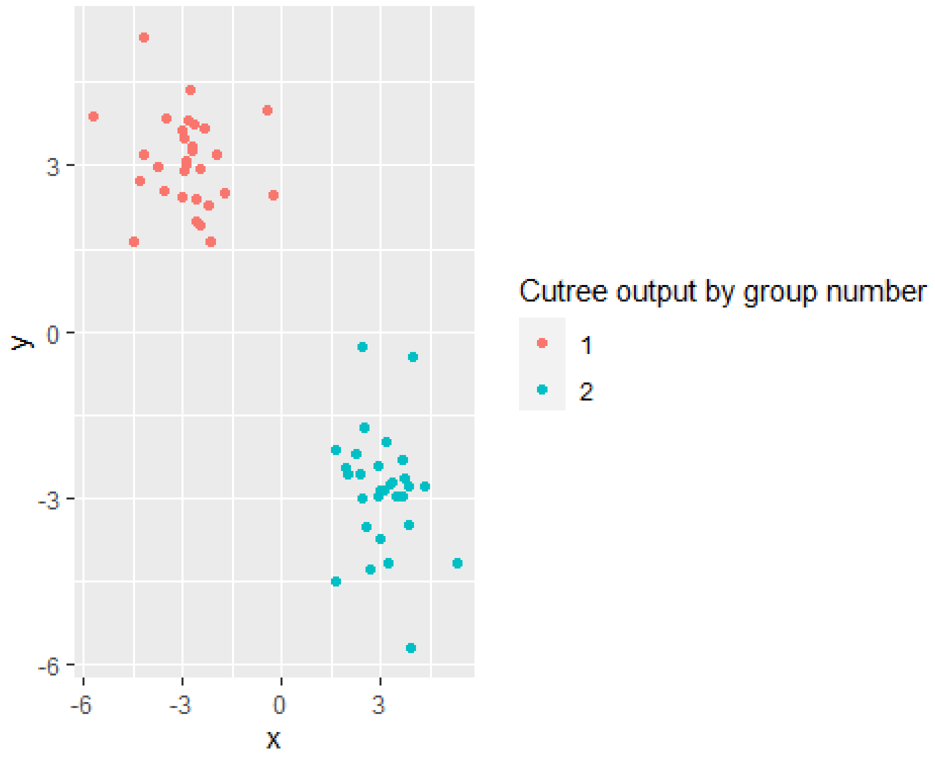


Instead of adding this cut-off line, we can use `cutree()`, a function specifically for this.

```
cth <- cutree(hc, h=10)
#or specify the number of groups
ctk <- cutree(hc, k=2)

# Put data into a format ggplot can use
dat <- as.data.frame(cbind(x, cth, ctk))
colnames(dat) <- c("x", "y", "cth", "ctk")

# Plot the data
ggplot(dat, aes(x, y, colour = as.factor(cth))) +
  geom_point() +
  labs(colour = "Cutree output by line")
```



```
ggplot(dat, aes(x, y, colour = as.factor(ctk))) +
  geom_point() +
  labs(colour = "Cutree output by group number")
```



Principle Component Analysis

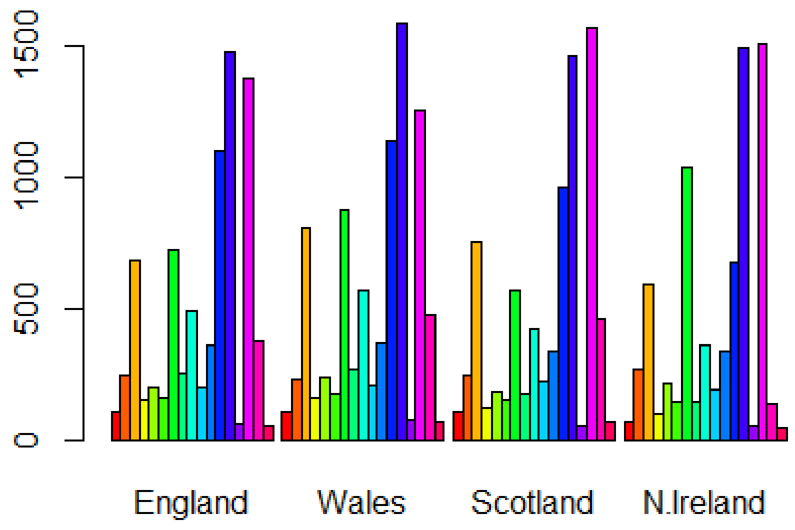
For this we are going to use data on the food consumption of citezins of different countries within the United Kingdom. First download the data:

```
# Download and assign data to an r object
url <- "https://tinyurl.com/UK-foods"
f.dat <- read.csv(url)

# Inspect the data
str(f.dat)

## 'data.frame':    17 obs. of  5 variables:
##  $ X        : chr  "Cheese" "Carcass_meat " "Other_meat " "Fish" ...
##  $ England  : int  105 245 685 147 193 156 720 253 488 198 ...
##  $ Wales    : int  103 227 803 160 235 175 874 265 570 203 ...
##  $ Scotland : int  103 242 750 122 184 147 566 171 418 220 ...
##  $ N.Ireland: int  66 267 586 93 209 139 1033 143 355 187 ...

# The first column would be better set as rownames than its own column
rownames(f.dat) <- f.dat[,1]
f.dat <- f.dat[,-1]

# Check this has worked
head(f.dat)

##                  England Wales Scotland N.Ireland
## Cheese               105   103      103        66
## Carcass_meat         245   227      242       267
## Other_meat           685   803      750       586
## Fish                 147   160      122        93
## Fats_and_oils        193   235      184       209
## Sugars               156   175      147       139

dim(f.dat)

## [1] 17  4

#N.B. a better way to have dealt with this problem is to set rownames = 1 in
the original read.csv
food <- read.csv(url, row.names = 1)
str(food)

## 'data.frame':    17 obs. of  4 variables:
##  $ England  : int  105 245 685 147 193 156 720 253 488 198 ...
##  $ Wales    : int  103 227 803 160 235 175 874 265 570 203 ...
##  $ Scotland : int  103 242 750 122 184 147 566 171 418 220 ...
##  $ N.Ireland: int  66 267 586 93 209 139 1033 143 355 187 ...

head(food)
```

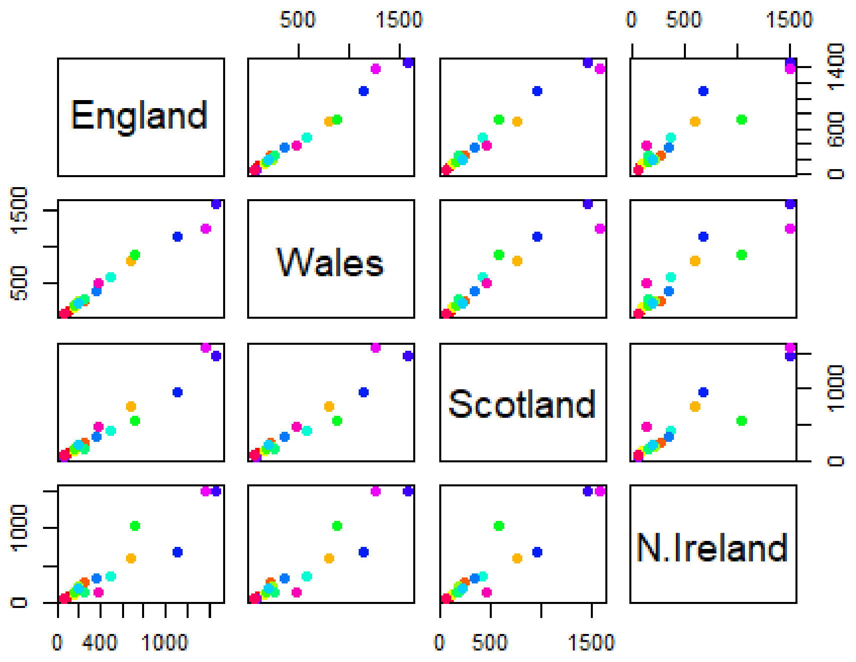
```
##                England Wales Scotland N.Ireland
## Cheese             105   103      103        66
## Carcass_meat       245   227      242       267
## Other_meat         685   803      750       586
## Fish               147   160      122        93
## Fats_and_oils      193   235      184       209
## Sugars             156   175      147       139
```

The data (food), is now in an appropriate form for analysis.

```
# Plot the data
barplot(as.matrix(food), beside = TRUE, col = rainbow(nrow(food)))
```



```
pairs(food, col=rainbow(nrow(food)), pch=16)
```

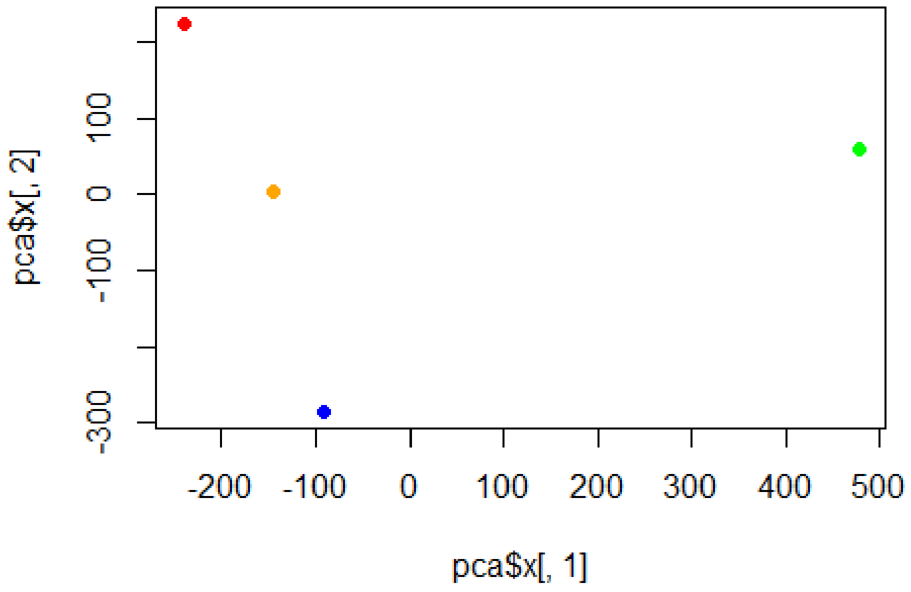


Analysis through plotting is not very helpful, thus we move onto PCA. We will begin with the basic PCA from base r, packages with other PCA functions exist, but they are often specialized PCA functions for specific data types or circumstances. The base r PCA function is prcomp().

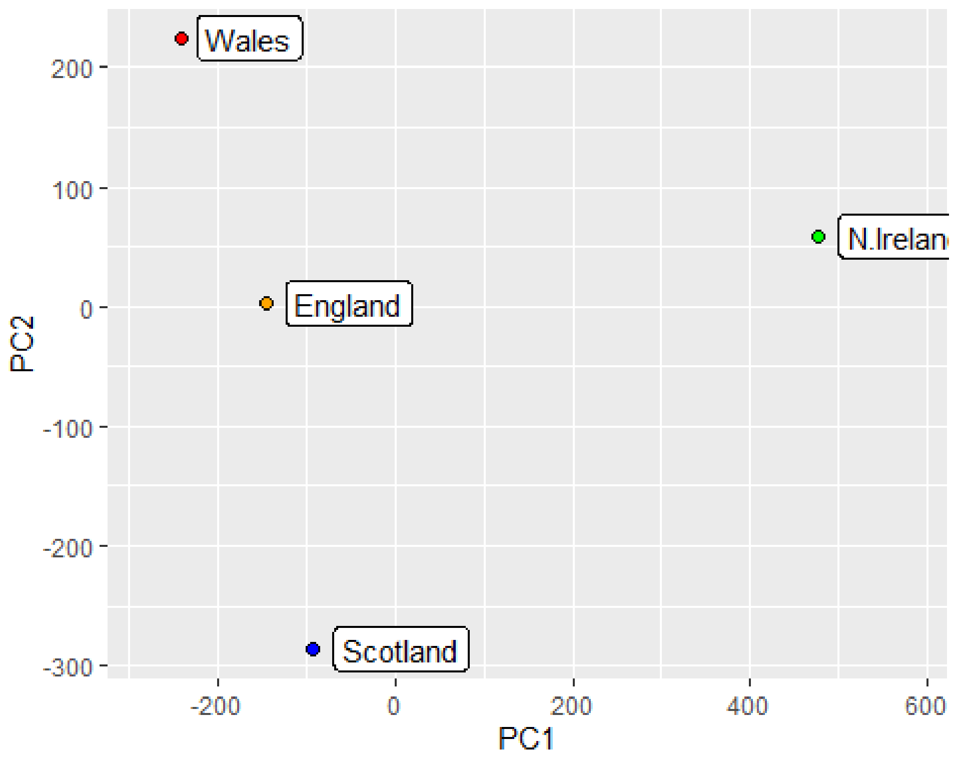
```
# prcomp() expects the transpose of the way our data currently is, with the
columns as the categories we are interested in
pca <- prcomp(t(food))

# Printing pca gives the PC values for each of our food categories, it is
thus easier to look at a summary of the data
summary(pca)

## Importance of components:
##                              PC1      PC2      PC3       PC4
## Standard deviation     324.1502 212.7478 73.87622 4.189e-14
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00

attributes(pca)

## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

We can now view a plot of the data along the two most important axes as found through PCA. A plot of PC1 vs PC2 is often called a PCA plot or “score plot”. The x component of the prcomp() output gives the data for such a score plot.

```
plot(pca$x[,1], pca$x[,2], col = c("Orange", "Red", "Blue", "Green"), pch =
16) #Note order is England, Wales, Scotland, N.Ireland
```

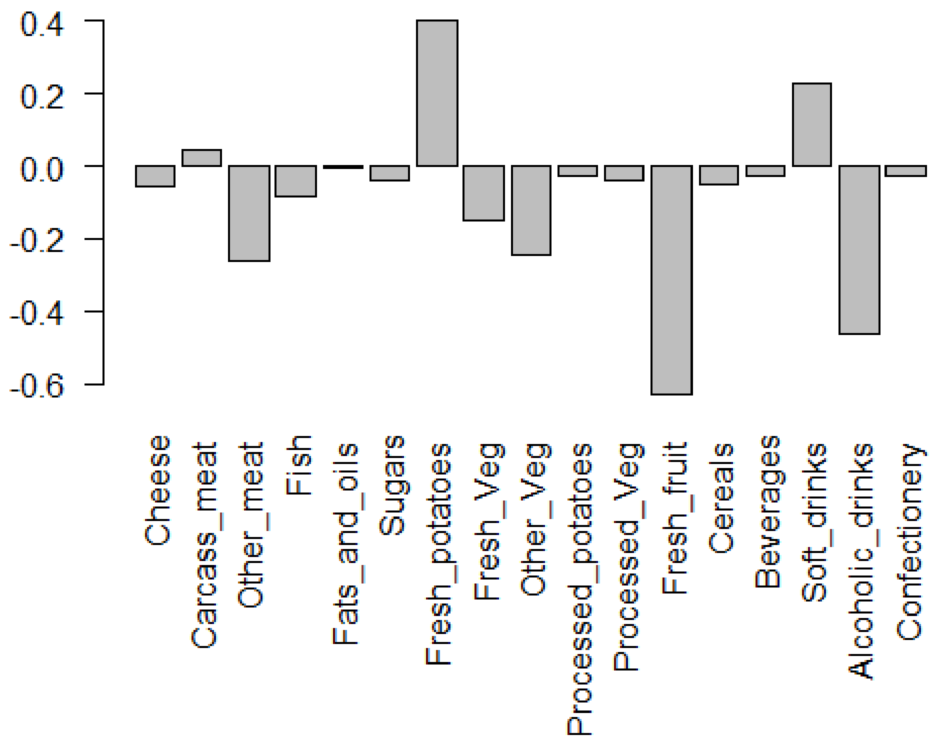


```
##Need to try plot with ggplot...
p <- as.data.frame(pca$x)

ggplot(p, aes(PC1, PC2)) +
  geom_point(fill = c("Orange", "Red", "Blue", "Green"), pch = 21, size = 2)
+
  geom_label(label = c("England", "Wales", "Scotland", "N.Ireland"), hjust =
-0.15) +
  scale_x_continuous(limits = c(-280, 580))
```



The four points are the four countries. Note the the large distance between one point (N.Ireland) versus the other three (England, Wales and Scotland) on the PC1 axis, which explains the most variation. Note that another important point is to show how much these axes actually contribute to the variation (the loadings), these are found in the component rotation. As PC1 explains most of the variation, we will focus on this.

```
par(mar=c(10, 3, 0.35, 0)) #change positioning of the plot
barplot(pca$rotation[,1], las=2)
```


0.4
0.2
0.0
-0.2
-0.4
-0.6
Cheese
Carcass_meat
Other_meat
Fish
Fats_and_oils
Sugars
Fresh_potatoes
Fresh_Veg
Other_Veg
Processed_potatoes
Processed_Veg
Fresh_fruit
Cereals
Beverages
Soft_drinks
Alcoholic_drinks
Confectionery