# class 07

Ruofan Kang (A17236920)

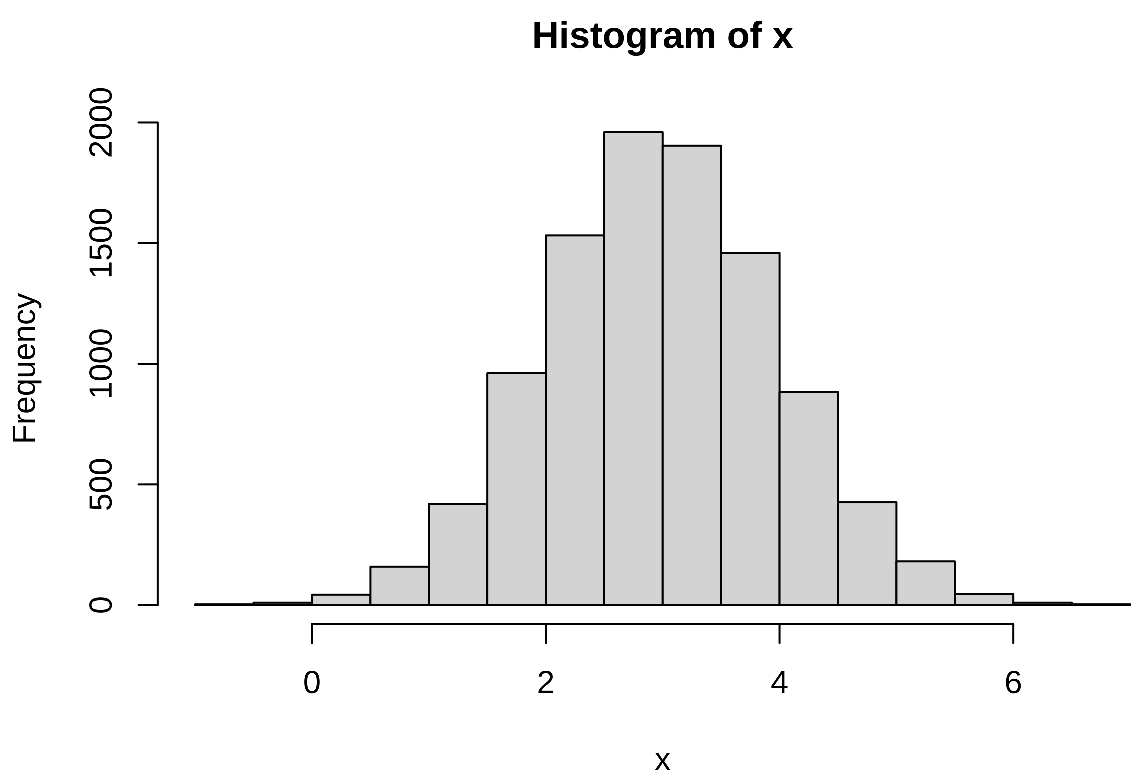
2023-10-24

## Clustering

we will start today's lab with clustering methods, in particular so-called K-means. The main function for this in R is `kmeans()`

Let's try it on some made up data where we know that what the answer should be.

```
x <- rnorm(10000, mean=3)
hist(x)
```




60 points

```
tmp <- c(rnorm(30, mean=3), rnorm(30,mean=-3))
x <- cbind(x=tmp, y=rev(tmp))
x
```

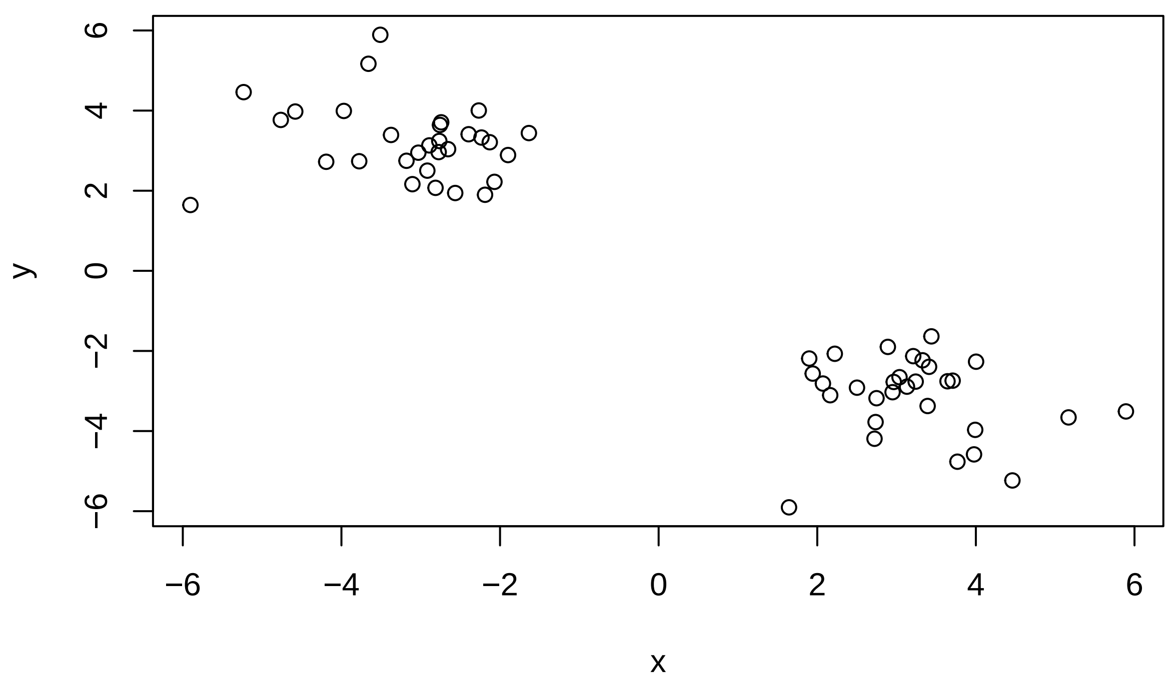
```
##                x          y
```

```
##  [1,]  3.240976 -2.766022
##  [2,]  3.991392 -3.969736
##  [3,]  2.163052 -3.104976
##  [4,]  1.898661 -2.189034
##  [5,]  3.976497 -4.582273
##  [6,]  3.130927 -2.891582
##  [7,]  5.168612 -3.659319
##  [8,]  3.391826 -3.374629
##  [9,]  3.210018 -2.129573
## [10,]  3.707352 -2.741579
## [11,]  3.409804 -2.395693
## [12,]  3.642331 -2.757303
## [13,]  2.735407 -3.775581
## [14,]  3.767412 -4.764453
## [15,]  4.460710 -5.233132
## [16,]  3.037118 -2.655066
## [17,]  3.327671 -2.233722
## [18,]  2.721973 -4.191819
## [19,]  3.439150 -1.636162
## [20,]  2.964130 -2.773621
## [21,]  1.942545 -2.565032
## [22,]  2.949764 -3.030119
## [23,]  2.747292 -3.179566
## [24,]  5.892171 -3.509852
## [25,]  2.890235 -1.899102
## [26,]  1.643839 -5.903929
## [27,]  2.071449 -2.813516
## [28,]  2.500929 -2.916589
## [29,]  4.002452 -2.268015
## [30,]  2.221102 -2.070330
## [31,] -2.070330  2.221102
## [32,] -2.268015  4.002452
## [33,] -2.916589  2.500929
## [34,] -2.813516  2.071449
## [35,] -5.903929  1.643839
## [36,] -1.899102  2.890235
## [37,] -3.509852  5.892171
## [38,] -3.179566  2.747292
## [39,] -3.030119  2.949764
## [40,] -2.565032  1.942545
## [41,] -2.773621  2.964130
## [42,] -1.636162  3.439150
## [43,] -4.191819  2.721973
## [44,] -2.233722  3.327671
## [45,] -2.655066  3.037118
## [46,] -5.233132  4.460710
## [47,] -4.764453  3.767412
## [48,] -3.775581  2.735407
## [49,] -2.757303  3.642331
## [50,] -2.395693  3.409804
## [51,] -2.741579  3.707352
## [52,] -2.129573  3.210018
## [53,] -3.374629  3.391826
## [54,] -3.659319  5.168612
```

```
## [55,] -2.891582  3.130927
## [56,] -4.582273  3.976497
## [57,] -2.189034  1.898661
## [58,] -3.104976  2.163052
## [59,] -3.969736  3.991392
## [60,] -2.766022  3.240976
```

We can pass this to the base R `plot()` function for a quick

```r
plot(x)
```




```r
k <- kmeans(x, centers=2, nstart=20)
k
```

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -3.132711  3.208227
## 2  3.208227 -3.132711
##
## Clustering vector:
##  [1] 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 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##
## Within cluster sum of squares by cluster:
## [1] 55.68467 55.68467
##  (between_SS / total_SS =  91.5 %)
```

```
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q1. How many points are in each cluster

```r
k$size
```

```
## [1] 30 30
```

Q. How we do get to the cluster membership/assignment.

```r
k$cluster
```

```
##  [1] 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 1 1 1 1 1 1 1 1
## [39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

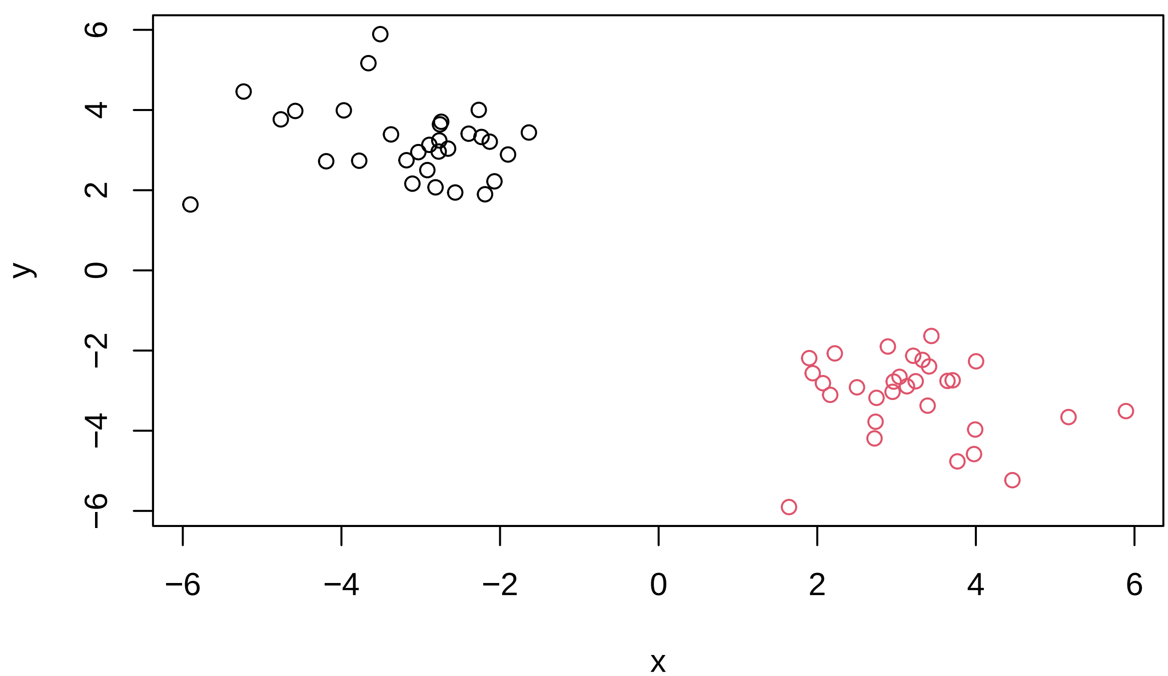
Q3. What about cluster centers?

```r
k$centers
```

```
##           x         y
## 1 -3.132711  3.208227
## 2  3.208227 -3.132711
```

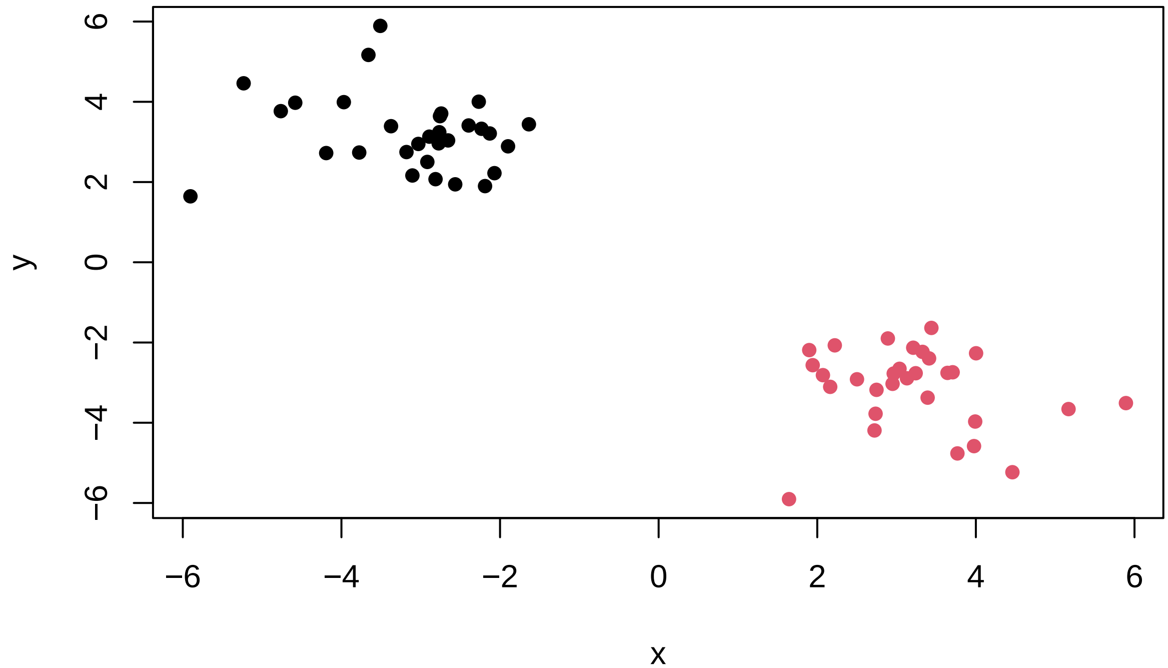
Now we got to the main results let's use them to plot our data with the kmeans result.

```r
plot(x, col=k$cluster)
```



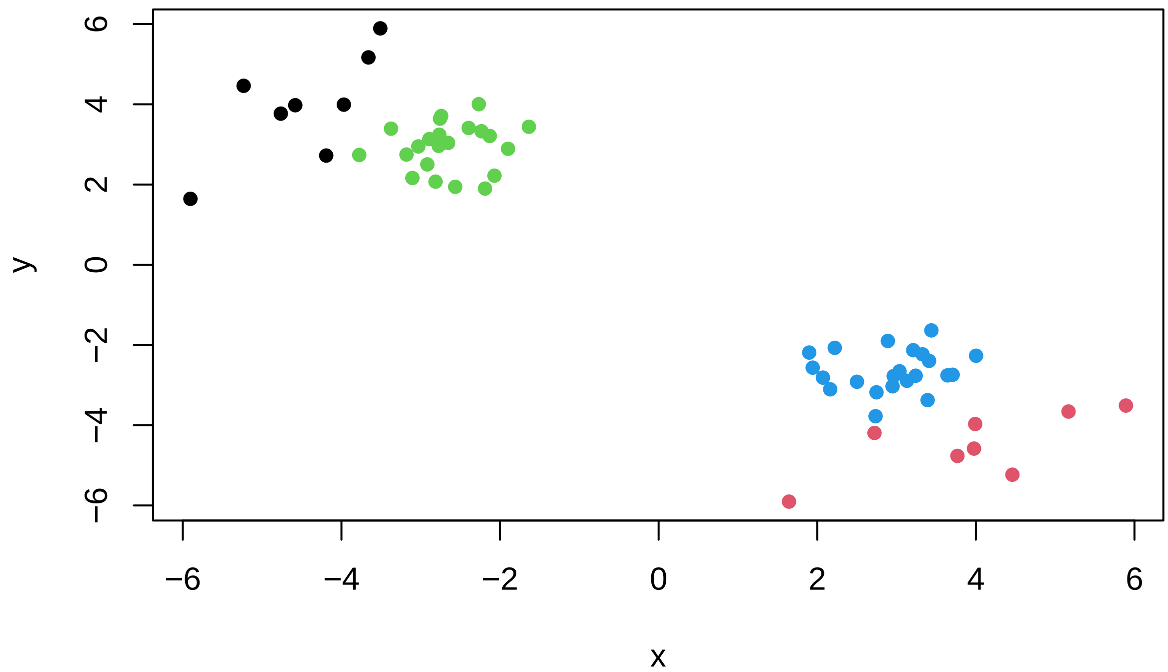

Q4.Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```r
plot(x, col=k$cluster, pch=16)
```




Q5. Cluster the data again with kmeans() into 4 groups and plot the results.

```r
k4 <- kmeans(x, center= 4, nstart=20)
plot(x, col=k4$cluster, pch=16)
```




K-means is very popular mostly because it is fast and relatively straight forward to run and understand. It has a big limitation in that you need to tell it how many groups (k, or centers) you want.

#Hierarchical clustering

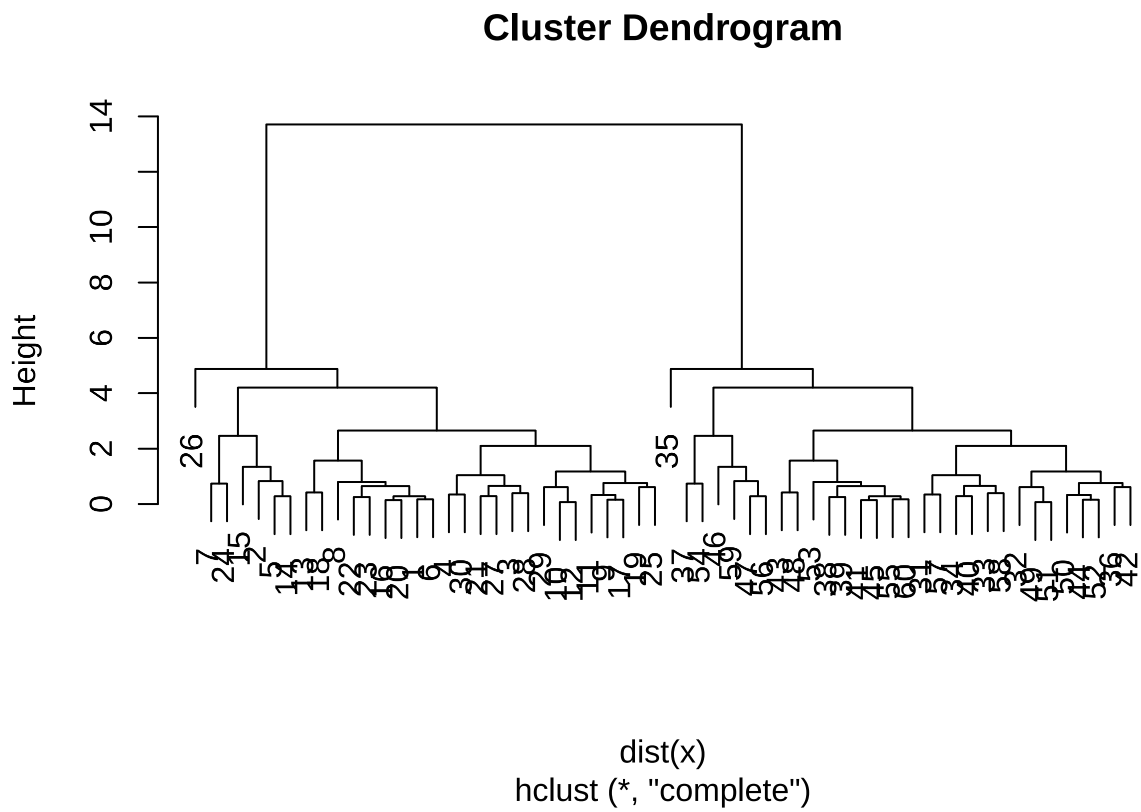
The main function in base R is called 'hclust()'. You have to pass it in a "distance matrix" not just your input data.

you can generate a distance matrix with the "dist()"

```
hc <- hclust( dist(x))
hc
```

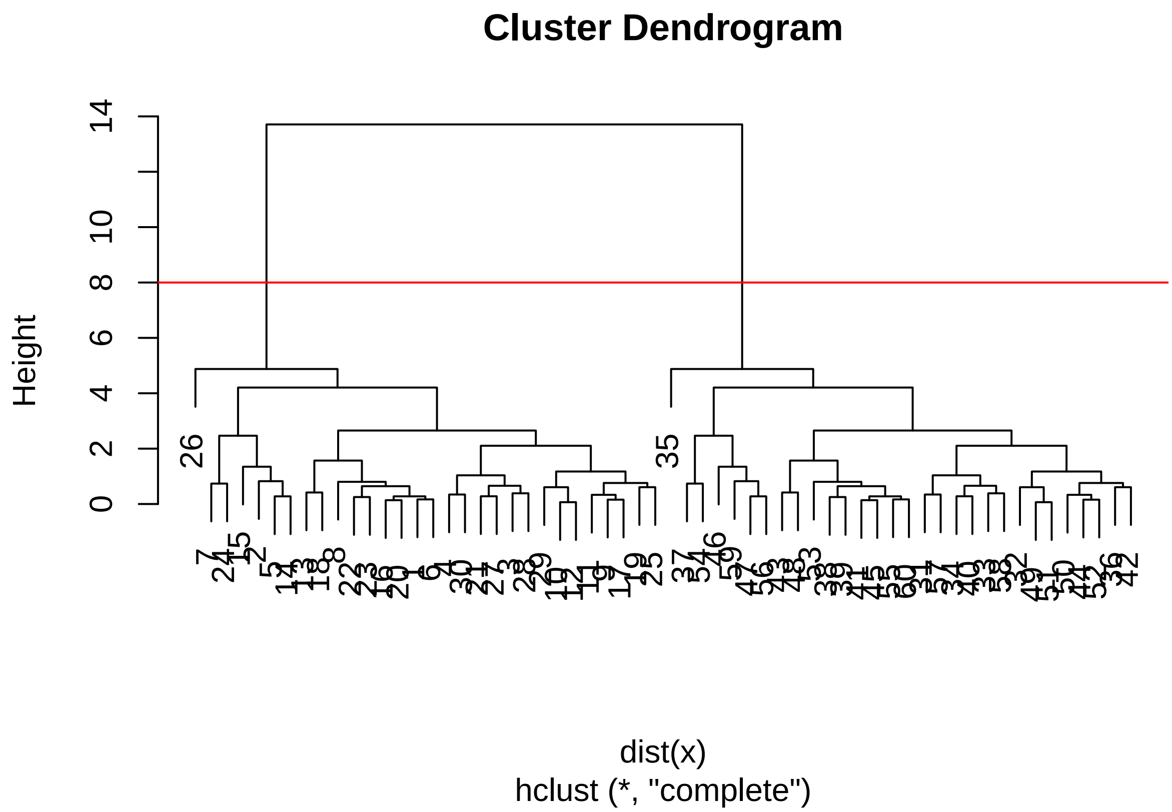
```
##
## Call:
## hclust(d = dist(x))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

```
plot(hc)
```




To find the cluster(cluster membership vector)from a 'hclust()' result we can "cut" the tree at a certain height

```
plot(hc)
abline(h=8,col="red")
```




```
grps <- cutree(hc, h=8)
```

```
table(grps)
```

```
## grps
##  1  2
## 30 30
```

> Q6. Plot our hclust results.

# Principal Component Analysis

## PCA of UK food data

Read data showing the consumption in grams (per person, per week) of 17 different types of food-stuff measured and averaged in the four countries of the United kingdom.

Let's see how PCA can help us but first we can try conventional analysis.

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
x
```

```
##                      X England Wales Scotland N.Ireland
## 1               Cheese     105   103      103        66
## 2        Carcass_meat      245   227      242       267
## 3          Other_meat      685   803      750       586
## 4                 Fish     147   160      122        93
```

```
## 5       Fats_and_oils      193   235       184       209
## 6              Sugars      156   175       147       139
## 7      Fresh_potatoes      720   874       566      1033
## 8           Fresh_Veg      253   265       171       143
## 9           Other_Veg      488   570       418       355
## 10 Processed_potatoes      198   203       220       187
## 11      Processed_Veg      360   365       337       334
## 12        Fresh_fruit     1102  1137       957       674
## 13            Cereals     1472  1582      1462      1494
## 14          Beverages       57    73        53        47
## 15        Soft_drinks     1374  1256      1572      1506
## 16    Alcoholic_drinks     375   475       458       135
## 17      Confectionery       54    64        62        41
```

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
rownames(x) <- x[ ,1]
x <-x[, -1]
head(x)
```

```
##                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
```

> Q1.How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

```
## Complete the following code to find out how many rows and columns are in x?
dim(x)
```

```
## [1] 17  4
```

```
rownames(x) <- x[,1]
x <- x[,-1]
head(x)
```

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

```
dim(x)
```

```
## [1] 17  3
```

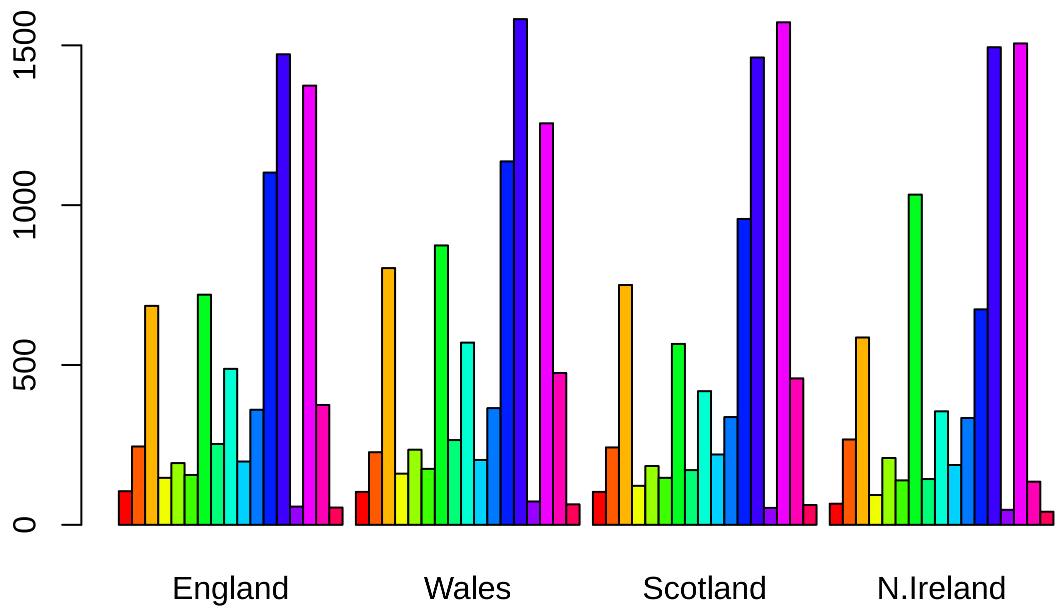
```
x <- read.csv(url, row.names=1)
head(x)
```

```
##                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
```

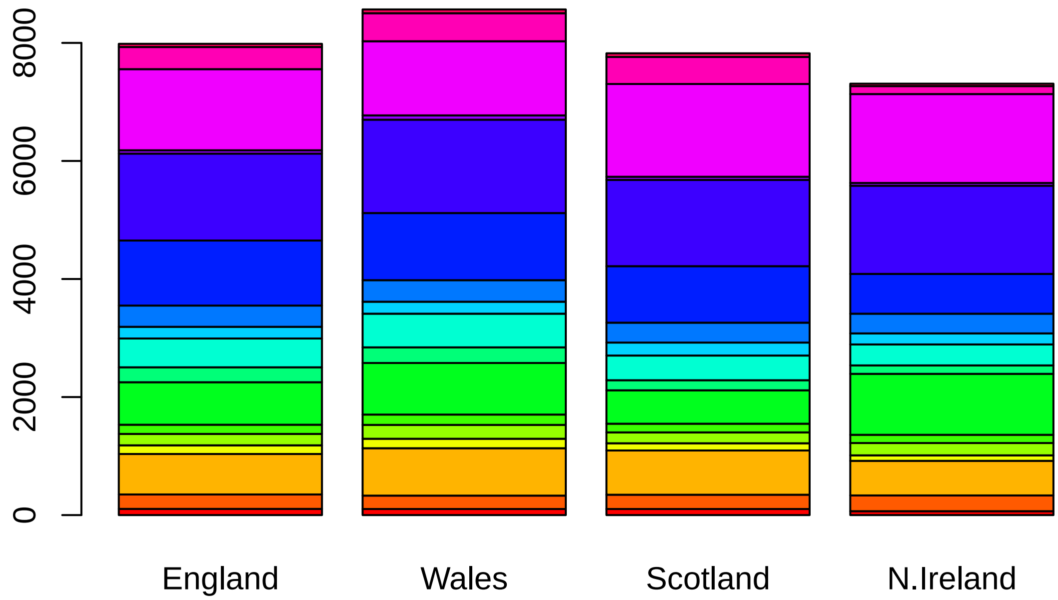
Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



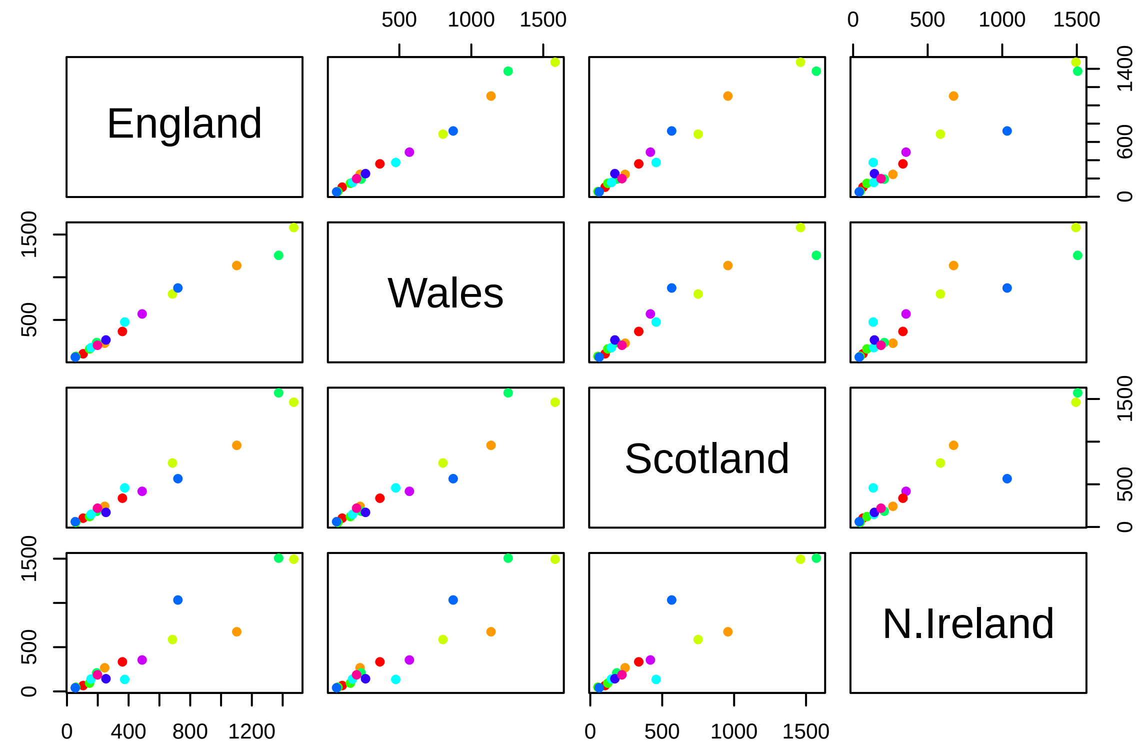

Q3.Changing what optional argument in the above barplot() function results in the following plot?

```
barplot(as.matrix(x), beside=FALSE, col=rainbow(nrow(x)))
```




Q5.Generating all pairwise plots may help somewhat. Can you make sense of the following code and resulting figure? What does it mean if a given point lies on the diagonal for a given plot?

```
pairs(x, col=rainbow(10), pch=16)
```




## Principal Component Analysis(PCA)

PCA can help us make sense of these types of datasets. Let's see how it works.

The main function in "base" R is called 'prcomp()'. In this case we want to first take the teanspose of our input 'x' so the columns are the food types and the countries are the rows.

```
head( t(x) )
```

```
##             Cheese Carcass_meat  Other_meat  Fish Fats_and_oils  Sugars
## England        105           245         685   147           193     156
## Wales          103           227         803   160           235     175
## Scotland       103           242         750   122           184     147
## N.Ireland       66           267         586    93           209     139
##             Fresh_potatoes  Fresh_Veg  Other_Veg  Processed_potatoes
## England                720        253        488                 198
## Wales                  874        265        570                 203
## Scotland               566        171        418                 220
## N.Ireland             1033        143        355                 187
##             Processed_Veg  Fresh_fruit  Cereals  Beverages Soft_drinks
## England               360         1102     1472         57        1374
## Wales                 365         1137     1582         73        1256
## Scotland              337          957     1462         53        1572
## N.Ireland             334          674     1494         47        1506
##             Alcoholic_drinks  Confectionery
## England                  375             54
## Wales                    475             64
```

```
## Scotland                    458              62
## N.Ireland                   135              41
```

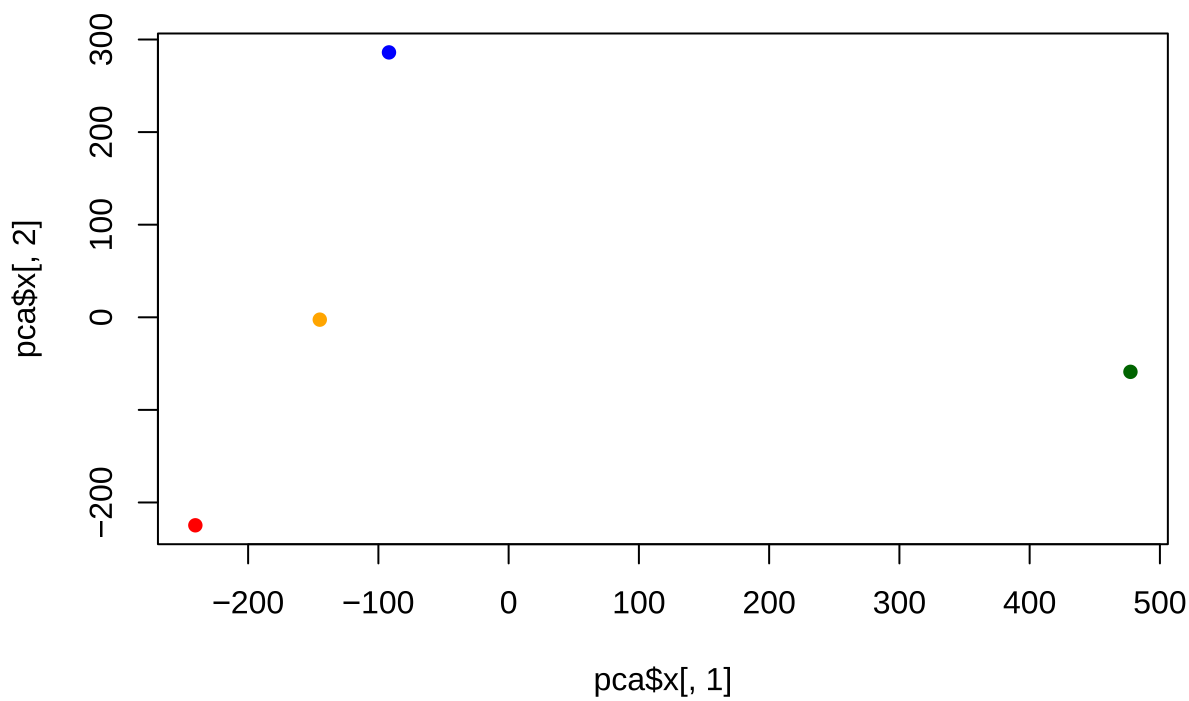
```
pca <- prcomp( t(x) )
summary(pca)
```

```
## Importance of components:
##                              PC1      PC2      PC3       PC4
## Standard deviation     324.1502 212.7478 73.87622 2.921e-14
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```
pca$x
```

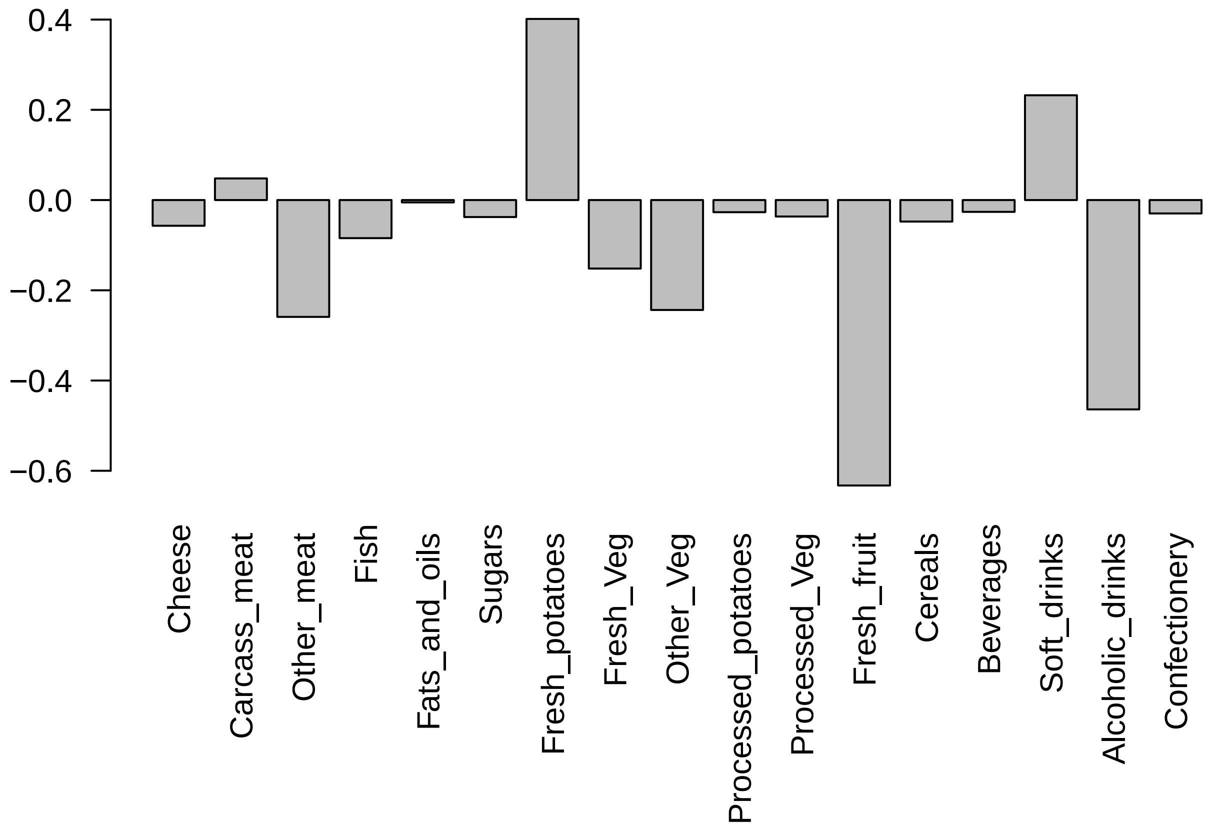
```
##                 PC1         PC2        PC3           PC4
## England   -144.99315   -2.532999 105.768945 -9.152022e-15
## Wales     -240.52915 -224.646925 -56.475555  5.560040e-13
## Scotland   -91.86934  286.081786 -44.415495 -6.638419e-13
## N.Ireland  477.39164  -58.901862  -4.877895  1.329771e-13
```

```
plot( pca$x[,1], pca$x[,2], col=c("orange", "red", "blue", "darkgreen"),pch=16)
```




The "loadings" tells us how much the origional variables (in our case the foods) contribute to the new variables i.e. the PCs.

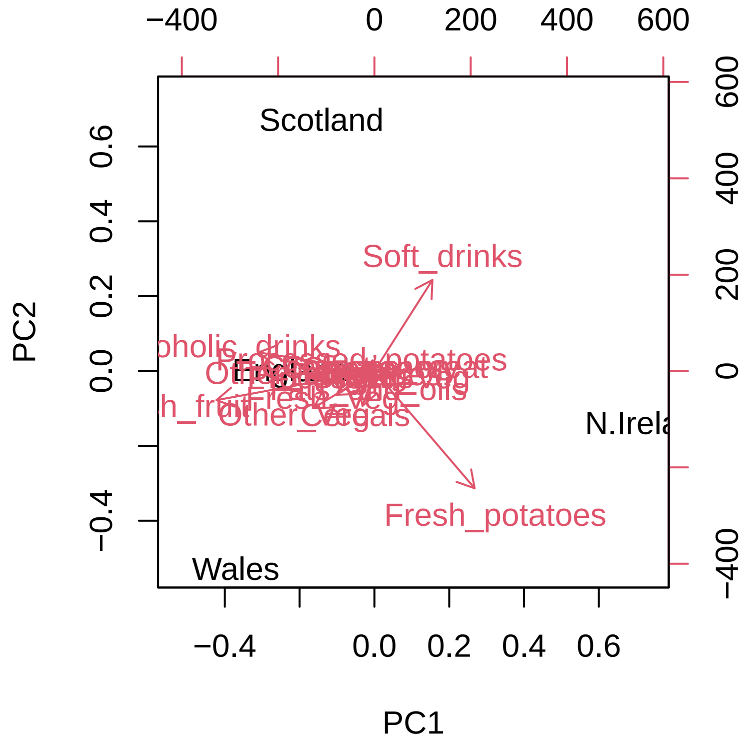
```
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```




```r
pca$rotation
```

```
##                             PC1          PC2         PC3          PC4
## Cheese             -0.056955380  0.016012850  0.02394295 -0.409382587
## Carcass_meat        0.047927628  0.013915823  0.06367111  0.729481922
## Other_meat         -0.258916658 -0.015331138 -0.55384854  0.331001134
## Fish               -0.084414983 -0.050754947  0.03906481  0.022375878
## Fats_and_oils      -0.005193623 -0.095388656 -0.12522257  0.034512161
## Sugars             -0.037620983 -0.043021699 -0.03605745  0.024943337
## Fresh_potatoes      0.401402060 -0.715017078 -0.20668248  0.021396007
## Fresh_Veg          -0.151849942 -0.144900268  0.21382237  0.001606882
## Other_Veg          -0.243593729 -0.225450923 -0.05332841  0.031153231
## Processed_potatoes -0.026886233  0.042850761 -0.07364902 -0.017379680
## Processed_Veg      -0.036488269 -0.045451802  0.05289191  0.021250980
## Fresh_fruit        -0.632640898 -0.177740743  0.40012865  0.227657348
## Cereals            -0.047702858 -0.212599678 -0.35884921  0.100043319
## Beverages          -0.026187756 -0.030560542 -0.04135860 -0.018382072
## Soft_drinks         0.232244140  0.555124311 -0.16942648  0.222319484
## Alcoholic_drinks   -0.463968168  0.113536523 -0.49858320 -0.273126013
## Confectionery      -0.029650201  0.005949921 -0.05232164  0.001890737
```

```r
## The inbuilt biplot() can be useful for small datasets
biplot(pca)
```


−400
0
200
400
600
Scotland
Soft_drinks
N.Irela
Fresh_potatoes
Wales
0.6
0.4
0.2
0.0
−0.4
PC2
−0.4
0.0
0.2
0.4
0.6
PC1