class07 Clustering BIMM143 Scott MacLeod

Scott MacLeod

Clustering Methods

The broad goal here is to find groupings (clusters) in your input data.

Kmeans

First, let's make up some data to cluster.

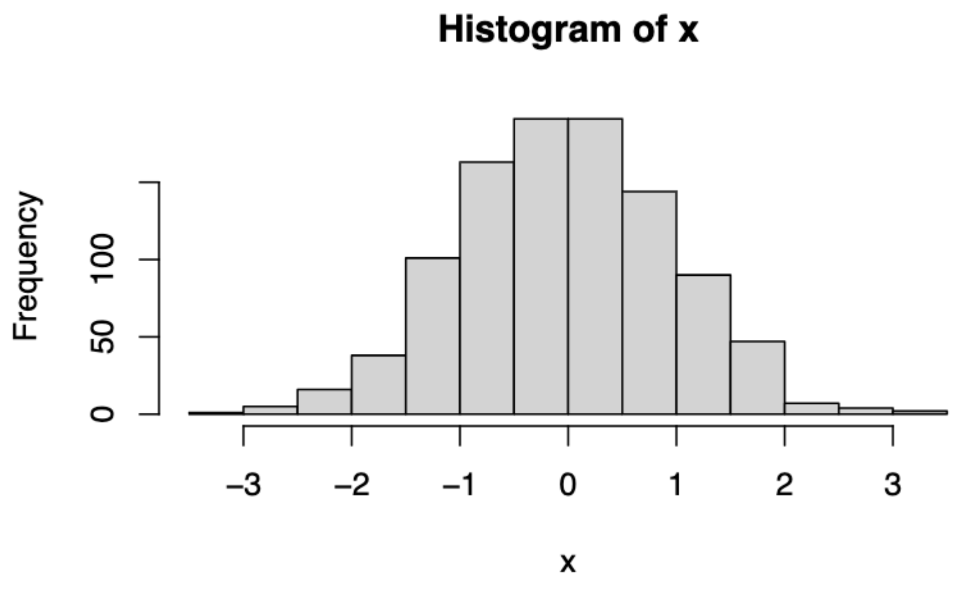
We are going to use `rnorm()` in order to make up some numbers. For example:

```r
rnorm(5)
```

```
[1] 1.287272 1.759522 2.963176 2.253911 1.270339
```

But, we are going to use 1,000 and turn it into a histogram!

```r
x <- rnorm(1000)
hist(x)
```



Make a vector of length 60 with 30 points centered at -3 and 30 points centered at +3

```
tmp <- c(rnorm(30, mean=-3), rnorm(30, mean = 3))
tmp
```

```
 [1] -2.3712600 -2.3881590 -2.4632339 -1.3430033 -2.0695799 -3.2314319
 [7] -1.9028718 -2.3344299 -3.9799338 -2.6991218 -2.7234767 -3.1024425
[13] -3.0829499 -3.5722274 -4.6456327 -3.9673174 -2.4413441 -4.7840010
[19] -4.7045100 -2.9898411 -2.3348882 -4.6237639 -2.9815985 -4.2668254
[25] -3.6732881 -2.7657798 -1.4943215 -1.1313594 -4.3114108 -2.9491492
[31]  3.2385677  0.6859490  3.1219878  1.6210932  2.9014646  3.5444786
[37]  1.5461177  2.4624231  1.8820063  1.9835510  2.8747541  3.7701765
[43]  2.4487741  3.2755139  3.1310713  3.1805380  2.8372498  3.5611826
[49]  0.3714007  4.1804710  1.4335416  3.9345738  4.3826019  2.2796438
[55]  3.3306641  4.4146861  3.0408435  4.3314404  1.4983177  3.1346905
```

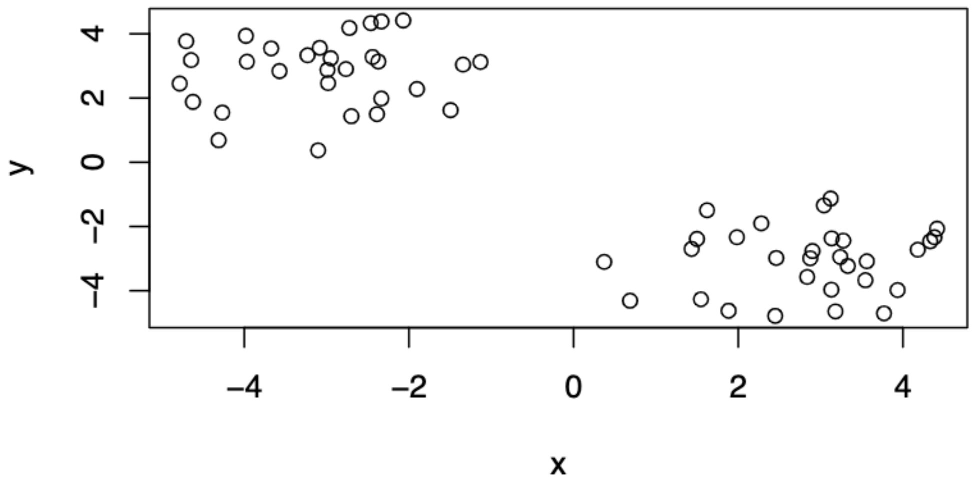
I will now make a small x and y dataset with 2 groups of points. Basically going to take the reverse of it! We are going to use the `rev()` function.

```
x <-cbind(x=tmp, y=rev(tmp))
x
```

```
              x          y
 [1,] -2.3712600  3.1346905
 [2,] -2.3881590  1.4983177
 [3,] -2.4632339  4.3314404
 [4,] -1.3430033  3.0408435
 [5,] -2.0695799  4.4146861
 [6,] -3.2314319  3.3306641
 [7,] -1.9028718  2.2796438
 [8,] -2.3344299  4.3826019
 [9,] -3.9799338  3.9345738
[10,] -2.6991218  1.4335416
[11,] -2.7234767  4.1804710
[12,] -3.1024425  0.3714007
[13,] -3.0829499  3.5611826
[14,] -3.5722274  2.8372498
[15,] -4.6456327  3.1805380
[16,] -3.9673174  3.1310713
[17,] -2.4413441  3.2755139
[18,] -4.7840010  2.4487741
[19,] -4.7045100  3.7701765
[20,] -2.9898411  2.8747541
[21,] -2.3348882  1.9835510
[22,] -4.6237639  1.8820063
[23,] -2.9815985  2.4624231
[24,] -4.2668254  1.5461177
[25,] -3.6732881  3.5444786
[26,] -2.7657798  2.9014646
[27,] -1.4943215  1.6210932
[28,] -1.1313594  3.1219878
[29,] -4.3114108  0.6859490
[30,] -2.9491492  3.2385677
[31,]  3.2385677 -2.9491492
[32,]  0.6859490 -4.3114108
[33,]  3.1219878 -1.1313594
[34,]  1.6210932 -1.4943215
[35,]  2.9014646 -2.7657798
[36,]  3.5444786 -3.6732881
[37,]  1.5461177 -4.2668254
[38,]  2.4624231 -2.9815985
[39,]  1.8820063 -4.6237639
[40,]  1.9835510 -2.3348882
[41,]  2.8747541 -2.9898411
[42,]  3.7701765 -4.7045100
```

```
[43,]  2.4487741 -4.7840010
[44,]  3.2755139 -2.4413441
[45,]  3.1310713 -3.9673174
[46,]  3.1805380 -4.6456327
[47,]  2.8372498 -3.5722274
[48,]  3.5611826 -3.0829499
[49,]  0.3714007 -3.1024425
[50,]  4.1804710 -2.7234767
[51,]  1.4335416 -2.6991218
[52,]  3.9345738 -3.9799338
[53,]  4.3826019 -2.3344299
[54,]  2.2796438 -1.9028718
[55,]  3.3306641 -3.2314319
[56,]  4.4146861 -2.0695799
[57,]  3.0408435 -1.3430033
[58,]  4.3314404 -2.4632339
[59,]  1.4983177 -2.3881590
[60,]  3.1346905 -2.3712600
```

```
plot(x)
```



We are going to run **`kmeans()`**

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

```
K-means clustering with 2 clusters of sizes 30, 30

Cluster means:
          x         y
1  2.813326 -3.044305
2 -3.044305  2.813326

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] 63.37361 63.37361
 (between_SS / total_SS =  89.0 %)

Available components:

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

Q. From your result object `k` how many points are in each cluster?

```r
k$size
```

```
[1] 30 30
```

Q. What "component" of your results object details the cluster membership?

```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
```

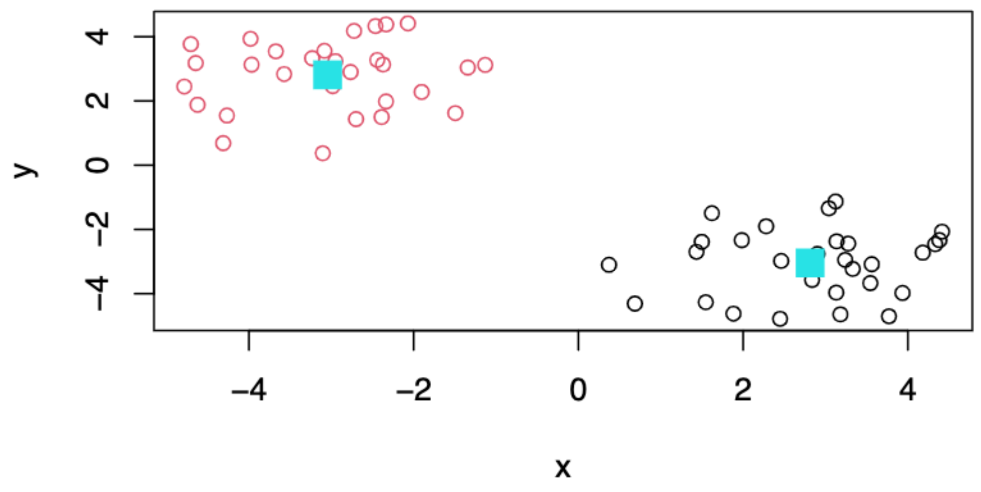
Q. Cluster centers?

```r
k$centers
```

```
          x         y
1  2.813326 -3.044305
2 -3.044305  2.813326
```

Q. Plot of our clustering results?

```
plot(x, col=k$cluster)
points(k$centers, col=5, pch=15, cex=2)
```



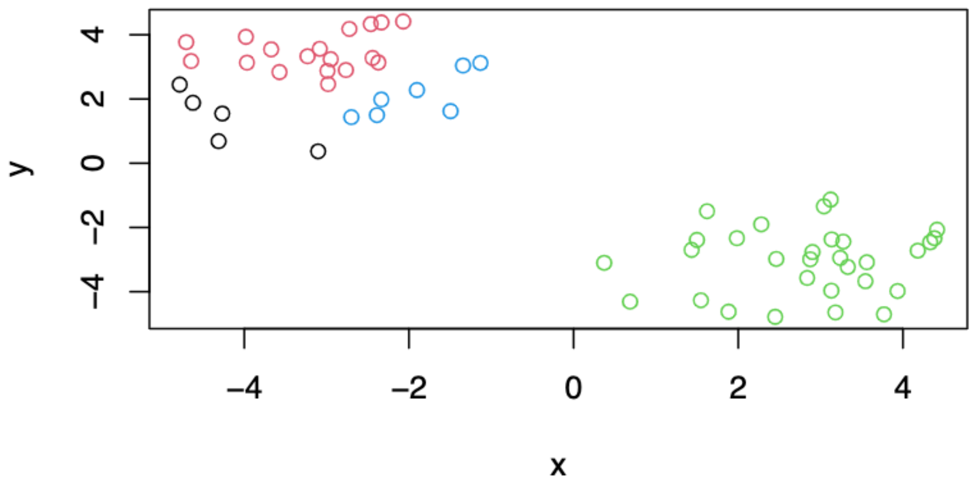
We can also cluster into 4 groups!

```
#kmeans
k4 <- kmeans(x, centers = 4)
k4
```

```
K-means clustering with 4 clusters of sizes 5, 18, 30, 7

Cluster means:
          x        y
1 -4.217689  1.386850
2 -3.163721  3.471475
```

```
3  2.813326 -3.044305
4 -1.899104  2.139854

Clustering vector:
 [1] 2 4 2 4 2 2 4 2 2 4 2 1 2 2 2 2 2 1 2 2 4 1 2 1 2 2 4 4 1 2 3 3 3 3 3 3 3 3
[39] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

Within cluster sum of squares by cluster:
[1]  4.661207 16.113605 63.373607  5.131554
 (between_SS / total_SS =  92.3 %)

Available components:

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

```
#plot results
plot(x, col=k4$cluster)
```



A big limitation of `kmeans()` is that it does wht you ask even if you ask for silly clusters!

Hierarchical Clustering

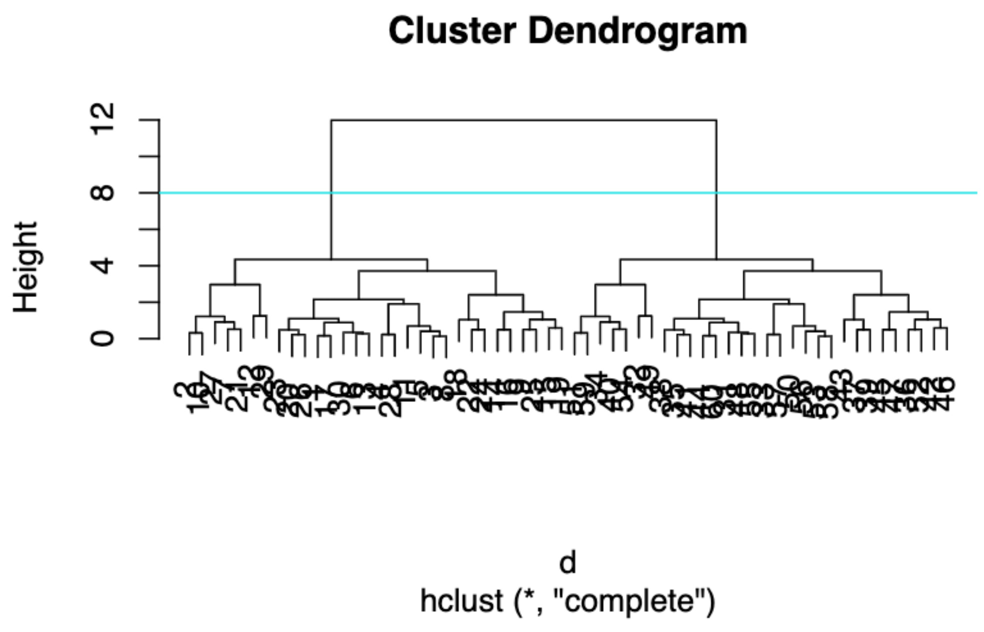
The main base R function for Hierarchical Clustering is `hclust()`. Unlike `kmeans()` you can not just pass your data as input. You first need to calculate a distance matrix.

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

```
Call:
hclust(d = d)

Cluster method   : complete 
Distance         : euclidean 
Number of objects: 60 
```

Use `plot()` to view results.

```
plot(hc)
abline(h=8, col=5)
```



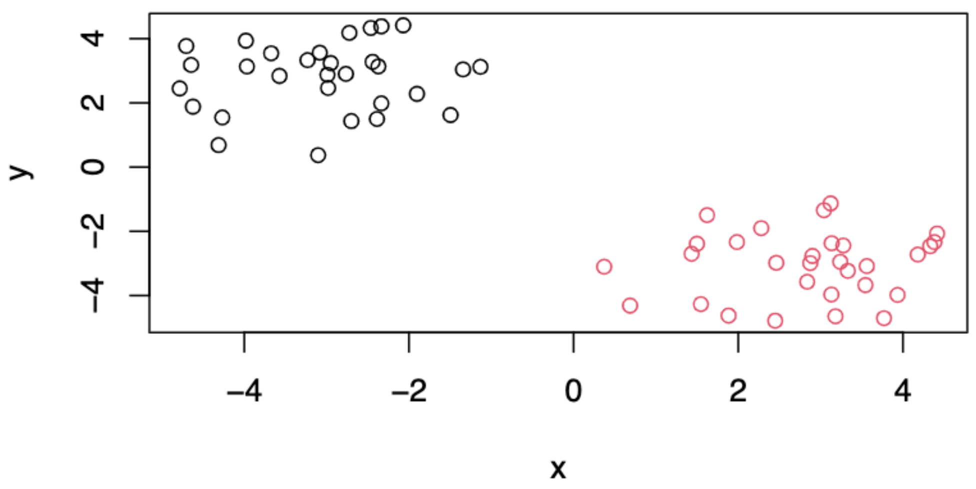
To make the "cut" and get our cluster membership vector we can use the `cutree()` function.

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

```
 [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 2 2
```

Make a plot of our data colored by hclust results.

```r
plot(x, col=grps)
```



LAB 7 Differences in Food in the UK - Principal Component Analysis (PCA)

Here we will do PCA on some food data from the United Kingdom.

```r
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
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
Fresh_potatoes          720   874      566      1033
Fresh_Veg               253   265      171       143
Other_Veg               488   570      418       355
Processed_potatoes      198   203      220       187
Processed_Veg           360   365      337       334
Fresh_fruit            1102  1137      957       674
Cereals                1472  1582     1462      1494
Beverages                57    73       53        47
Soft_drinks            1374  1256     1572      1506
Alcoholic_drinks        375   475      458       135
Confectionery            54    64       62        41
```

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

```r
nrow(x)
```

```
[1] 17
```

```r
ncol(x)
```

```
[1] 4
```

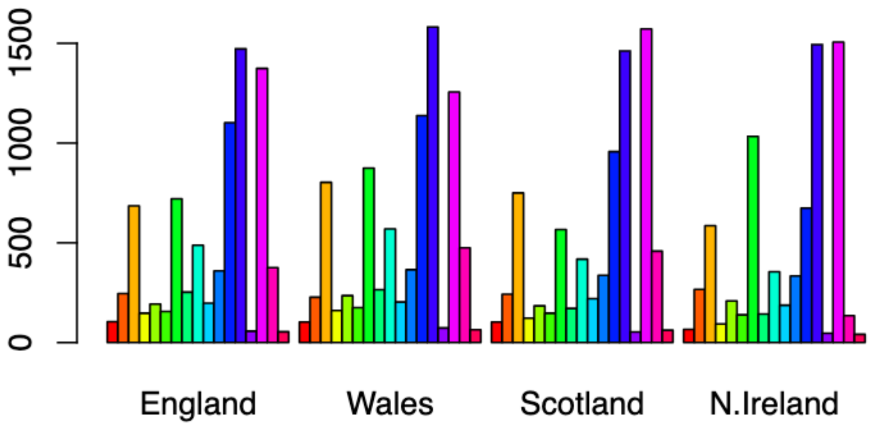
There are 17 rows and 4 columns. You can also look at the "environment" tab in the top right of the window.

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?

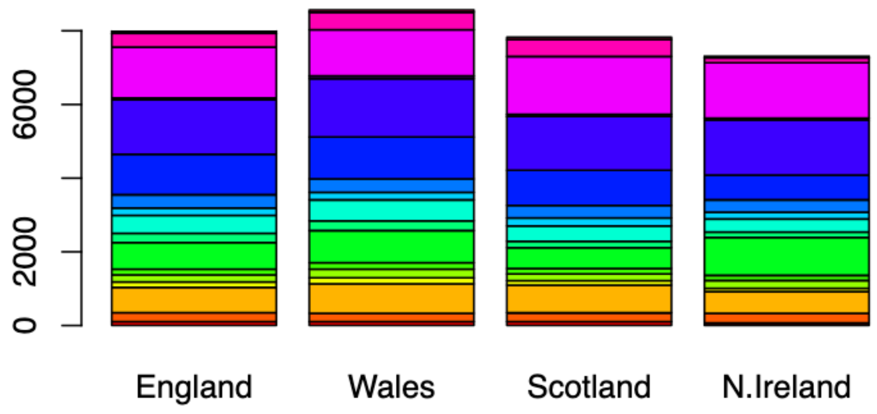
I like using the `row.names()` function versus subtracting the rows. If you kept subtracting the columns, eventually you would run out of columns if you kept running the code. The `row.names()` approach is more robust and leaves less room for accidents in the future.

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

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



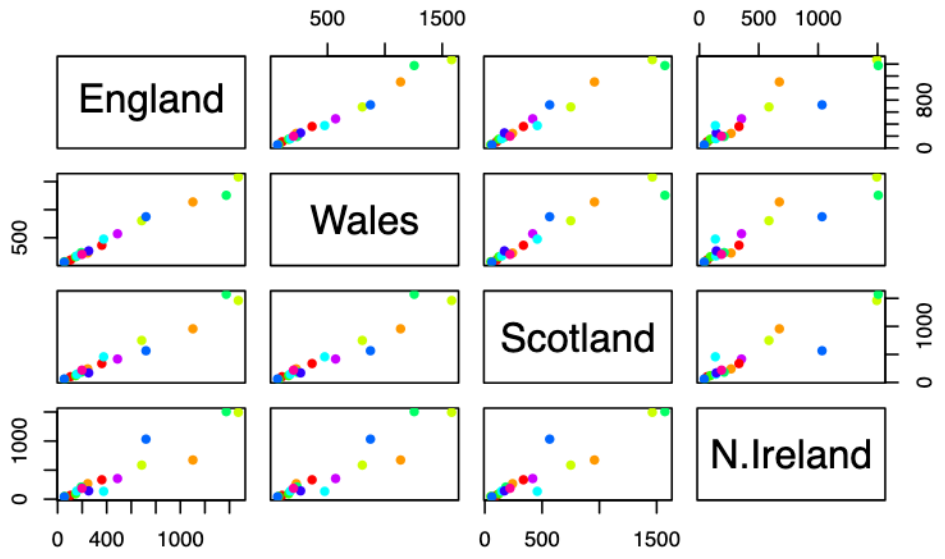
```r
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



By changing the "Beside" from True to False, you can change the plots. If it is false, the columns of heights are portrayed as stacked bars, and if true the columns are portrayed as juxtaposed bars.

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)
```



This would be useful for a small set of Data. This Paris Plot is comparing all of the 4 countries. If we wanted to look further, we could color code the dots and see which foods/bevs are being compared.

##PCA to the rescue

The main "base" R function for PCA is called `prcomp()`. We are going to use `t()` to transpose the data set. Then we took the pcr and then made a summary table of the results.

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

> Q How much variance is captured in two PCs?

96.5% is captured in two PCs. Look at the cumunulative tab in the table above :)

Now to make our main "PC score plot" (a.k.a "PC1 vs. PC2", or "PC plot" or "Ordination plot"). This thing has a lot of different names.

```
attributes(pca)
```

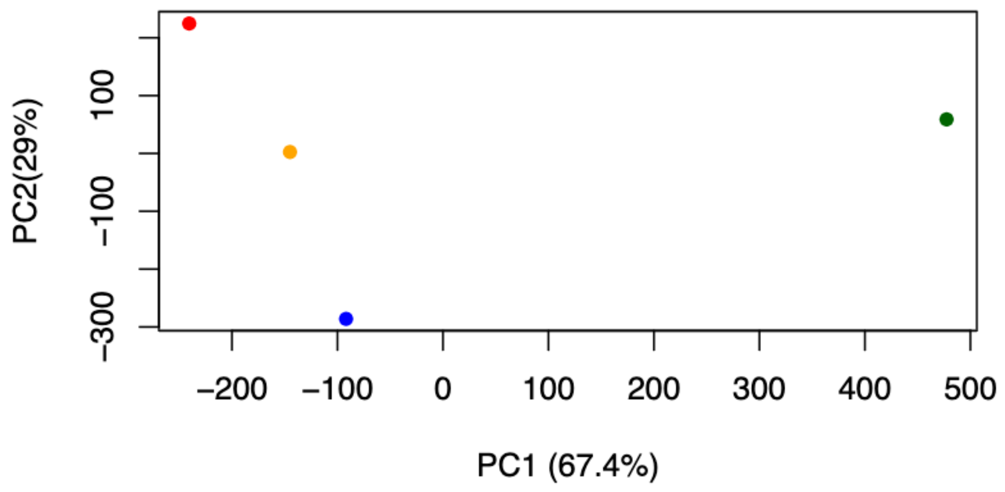
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

We are after the **pca$x** result component to make our main PCA plot.

```
pca$x
```

```
                 PC1        PC2         PC3           PC4
England   -144.99315   2.532999 -105.768945  2.842865e-14
Wales     -240.52915 224.646925   56.475555  7.804382e-13
Scotland   -91.86934 -286.081786  44.415495 -9.614462e-13
N.Ireland  477.39164  58.901862    4.877895  1.448078e-13
```

```
mycols <- c("orange","red","blue","darkgreen")
plot(pca$x[,1], pca$x[,2], col=mycols, pch=16, xlab="PC1 (67.4%)", ylab="PC2(29%)")
```



Another important result from PCA is how the original variables (in this case: the foods) contributed to the PCs.

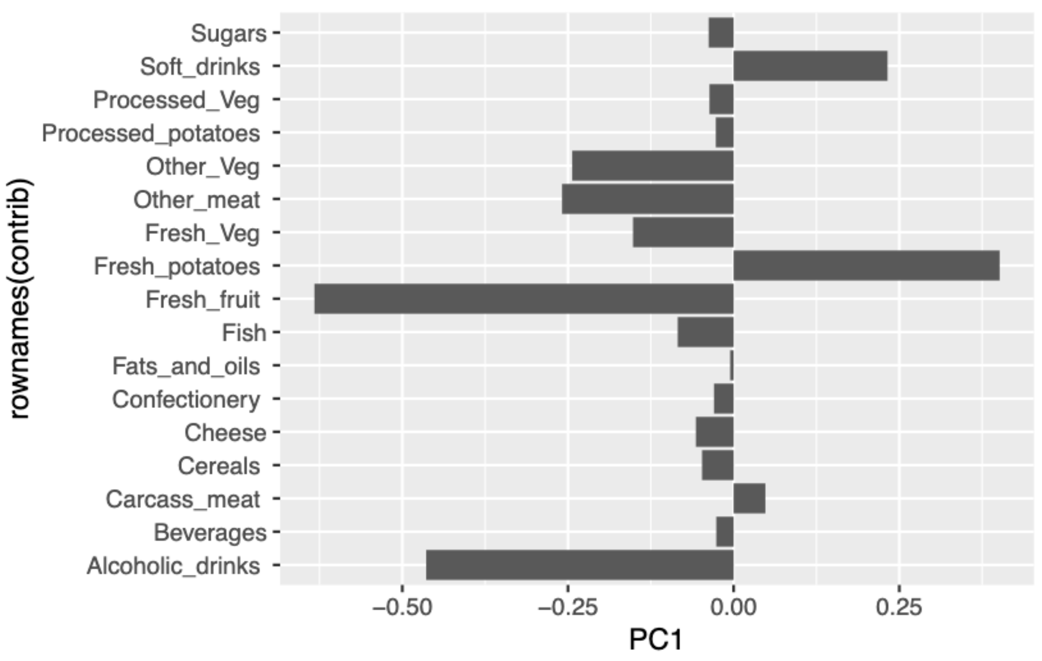
This is contained in the `pca$rotation()` object - people often call this the "loadings" or "contributions" to the PCs.

```
pca$rotation
```

```
                            PC1          PC2         PC3          PC4
Cheese              -0.056955380 -0.016012850 -0.02394295 -0.691718038
Carcass_meat         0.047927628 -0.013915823 -0.06367111  0.635384915
Other_meat          -0.258916658  0.015331138  0.55384854  0.198175921
Fish                -0.084414983  0.050754947 -0.03906481 -0.015824630
Fats_and_oils       -0.005193623  0.095388656  0.12522257  0.052347444
Sugars              -0.037620983  0.043021699  0.03605745  0.014481347
Fresh_potatoes       0.401402060  0.715017078  0.20668248 -0.151706089
Fresh_Veg           -0.151849942  0.144900268 -0.21382237  0.056182433
Other_Veg           -0.243593729  0.225450923  0.05332841 -0.080722623
Processed_potatoes  -0.026886233 -0.042850761  0.07364902 -0.022618707
Processed_Veg       -0.036488269  0.045451802 -0.05289191  0.009235001
Fresh_fruit         -0.632640898  0.177740743 -0.40012865 -0.021899087
Cereals             -0.047702858  0.212599678  0.35884921  0.084667257
Beverages           -0.026187756  0.030560542  0.04135860 -0.011880823
Soft_drinks          0.232244140 -0.555124311  0.16942648 -0.144367046
Alcoholic_drinks    -0.463968168 -0.113536523  0.49858320 -0.115797605
Confectionery       -0.029650201 -0.005949921  0.05232164 -0.003695024
```

We can make a plot along PC1.This one isn't as pretty as it could be, but we have the general idea.

```
library(ggplot2)

contrib <- as.data.frame(pca$rotation)
ggplot(contrib) +
  aes(PC1, rownames(contrib)) +
  geom_col()
```


rownames(contrib)
Sugars
Soft_drinks
Processed_Veg
Processed_potatoes
Other_Veg
Other_meat
Fresh_Veg
Fresh_potatoes
Fresh_fruit
Fish
Fats_and_oils
Confectionery
Cheese
Cereals
Carcass_meat
Beverages
Alcoholic_drinks
−0.50
−0.25
0.00
0.25
PC1