lab07

Laurie Chang, A16891192

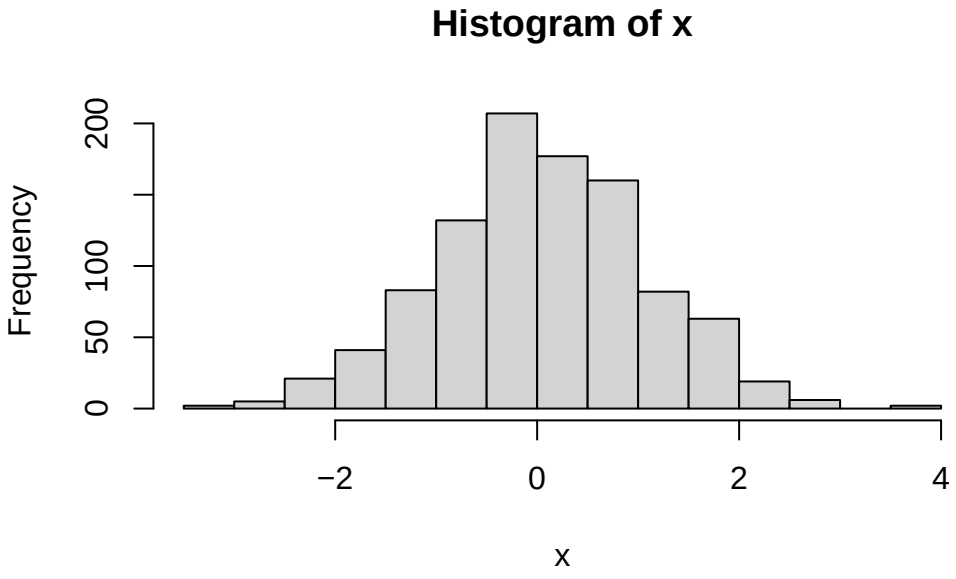
Clustering Methods

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

Kmeans

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

```
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))
```

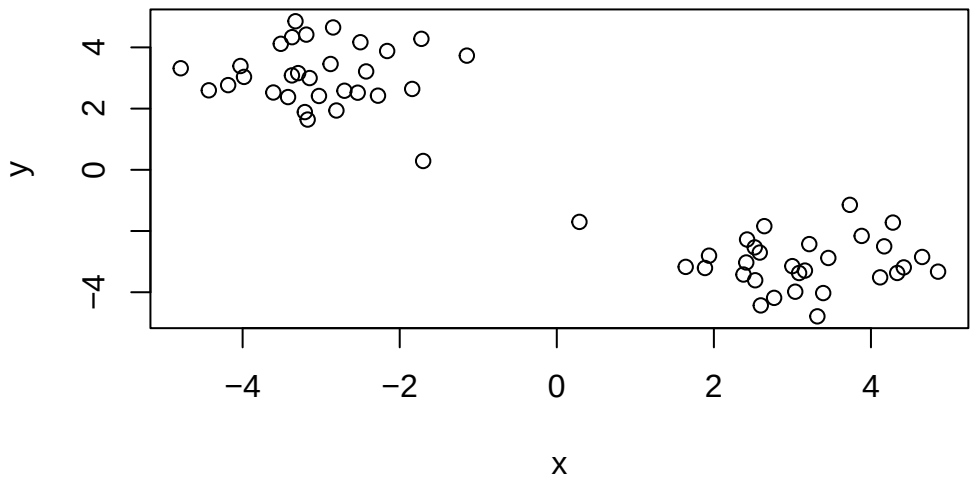
I will now make a wee x and y dataset with 2 groups of points.

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

```
                x          y
 [1,] -3.1719387  1.6412503
 [2,] -1.1452827  3.7309864
 [3,] -3.3705475  4.3339575
 [4,] -3.0279495  2.4123429
 [5,] -2.2765821  2.4226846
 [6,] -4.1844835  2.7671463
 [7,] -3.1472175  2.9986493
 [8,] -2.1599941  3.8815881
 [9,] -4.7881680  3.3177423
[10,] -3.3272624  4.8544846
[11,] -2.4275548  3.2143964
[12,] -2.8482194  4.6514508
[13,] -3.4217925  2.3767309
[14,] -2.7037762  2.5839407
[15,] -3.5141819  4.1168031
[16,] -1.8409373  2.6421002
[17,] -3.2930350  3.1590583
[18,] -3.3723660  3.0847630
[19,] -4.4306331  2.5972224
[20,] -3.2078482  1.8866419
[21,] -2.5004072  4.1683155
[22,] -1.7238755  4.2790337
[23,] -1.7013843  0.2880522
[24,] -2.5345717  2.5201419
[25,] -2.8802792  3.4570833
[26,] -2.8060389  1.9382663
[27,] -4.0275471  3.3918489
[28,] -3.1891791  4.4178222
[29,] -3.6092385  2.5260364
[30,] -3.9824171  3.0359280
[31,]  3.0359280 -3.9824171
[32,]  2.5260364 -3.6092385
```

```
[33,]  4.4178222 -3.1891791
[34,]  3.3918489 -4.0275471
[35,]  1.9382663 -2.8060389
[36,]  3.4570833 -2.8802792
[37,]  2.5201419 -2.5345717
[38,]  0.2880522 -1.7013843
[39,]  4.2790337 -1.7238755
[40,]  4.1683155 -2.5004072
[41,]  1.8866419 -3.2078482
[42,]  2.5972224 -4.4306331
[43,]  3.0847630 -3.3723660
[44,]  3.1590583 -3.2930350
[45,]  2.6421002 -1.8409373
[46,]  4.1168031 -3.5141819
[47,]  2.5839407 -2.7037762
[48,]  2.3767309 -3.4217925
[49,]  4.6514508 -2.8482194
[50,]  3.2143964 -2.4275548
[51,]  4.8544846 -3.3272624
[52,]  3.3177423 -4.7881680
[53,]  3.8815881 -2.1599941
[54,]  2.9986493 -3.1472175
[55,]  2.7671463 -4.1844835
[56,]  2.4226846 -2.2765821
[57,]  2.4123429 -3.0279495
[58,]  4.3339575 -3.3705475
[59,]  3.7309864 -1.1452827
[60,]  1.6412503 -3.1719387
```

```
plot(x)
```




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

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

Cluster means:
          x         y
1  3.089882 -3.020490
2 -3.020490  3.089882

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 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] 49.77686 49.77686
 (between_SS / total_SS =  91.8 %)

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?

```
k$size
```

```
[1] 30 30
```

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

```
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 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
```

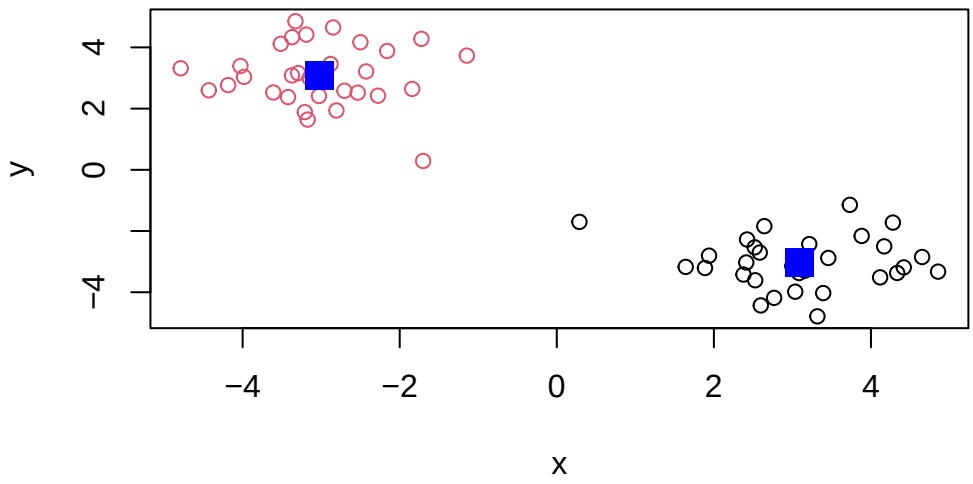
Q. Cluster centers?

```
k$centers
```

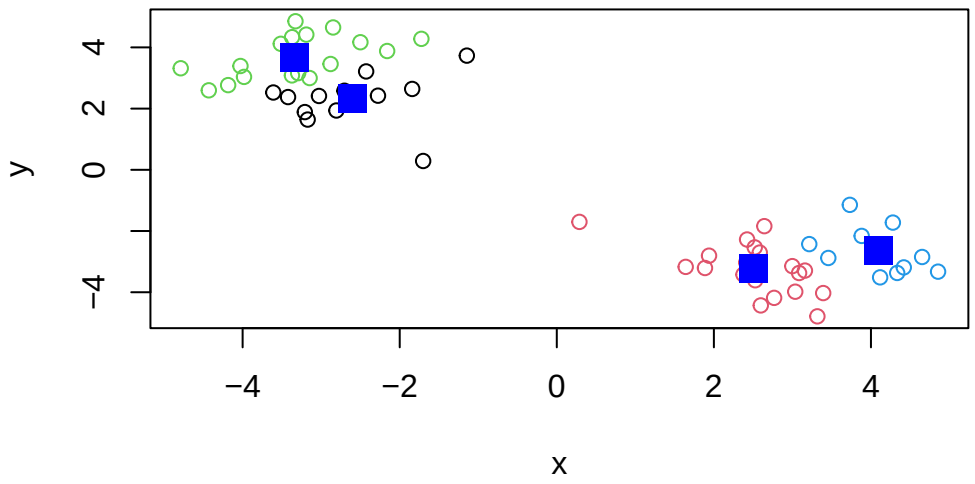
```
          x         y
1  3.089882 -3.020490
2 -3.020490  3.089882
```

Q. Plot of our clustering results

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




```
# kmeans
m <- kmeans(x, centers = 4)
# plot results
plot(x, col = m$cluster,)
points(m$centers, col = "blue", pch = 15, cex = 2)
```



A big limitation of kmeans is that it does what you ask even if you ask for silly clusters.

Hierarchial Clustering

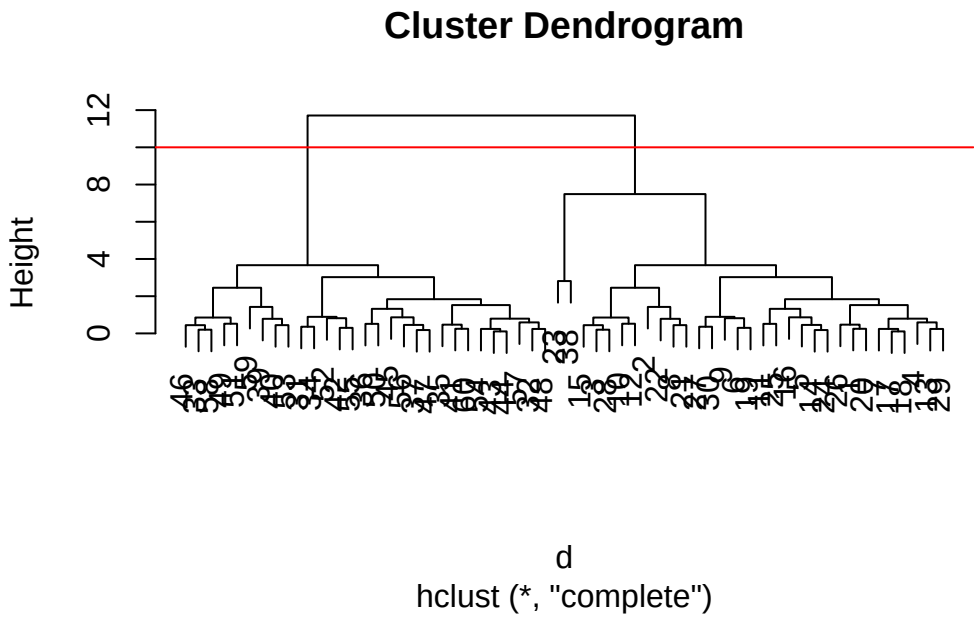
The main base R function for Hierarchial Clustering is `hclust()`. Unlike `kmeans()`, you cannot just pass it your data as input. You first need to calulate a distance matrix.

```
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=10, col = "red")
```



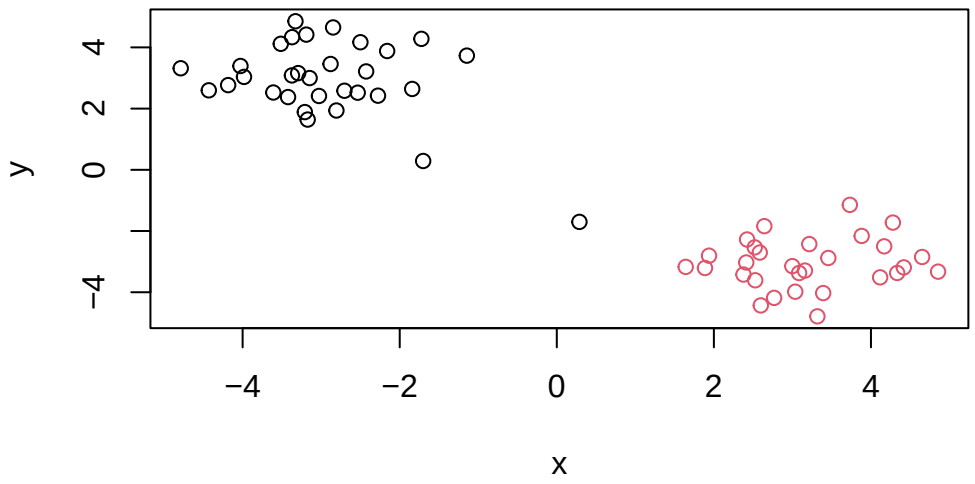
To make the "cut" and get our cluster membership vector, we can use the `cutree()` function.

```
grps <- cutree(hc, h = 10)
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 2 2 2 2 2 2 2 1
[39] 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

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



Principal Component Analysis (PCA)

Here we will do Principal Component Analysis (PCA) on some food data from the UK.

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

Rename it so the first column turns into the row name!

*what not to do:

```
rownames(x)
```

```
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12" "13" "14" "15"
[16] "16" "17"
```

what you should do:

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

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

```
dim(x)
```

```
[1] 17  4
```

preview the first 6 columns

```
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
```

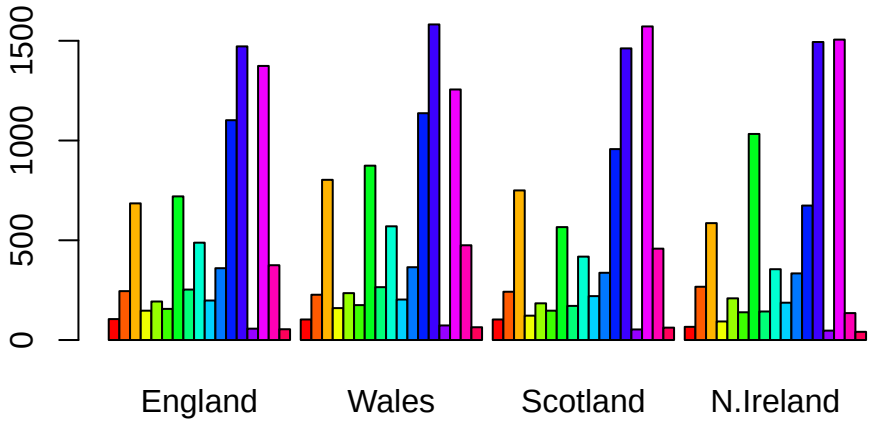
```
x <- read.csv(url, row.names = 1)
head(x)
```

```
              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?

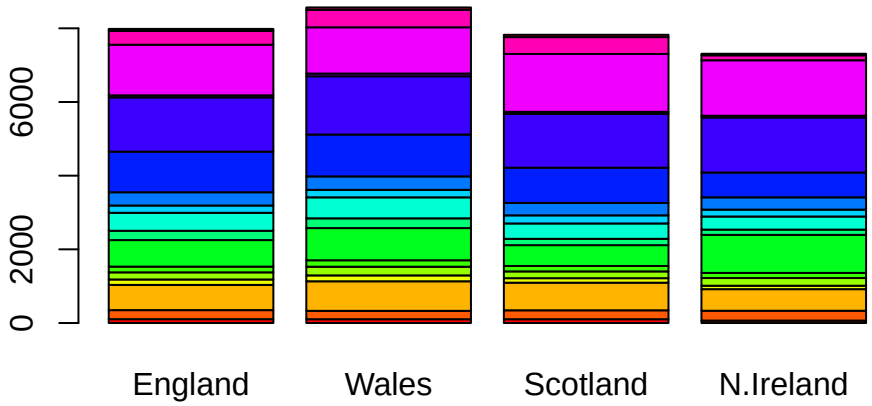
putting the rownames argument into the `read.csv()` function is a better approach as the other approach, if run many times, can delete data.

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



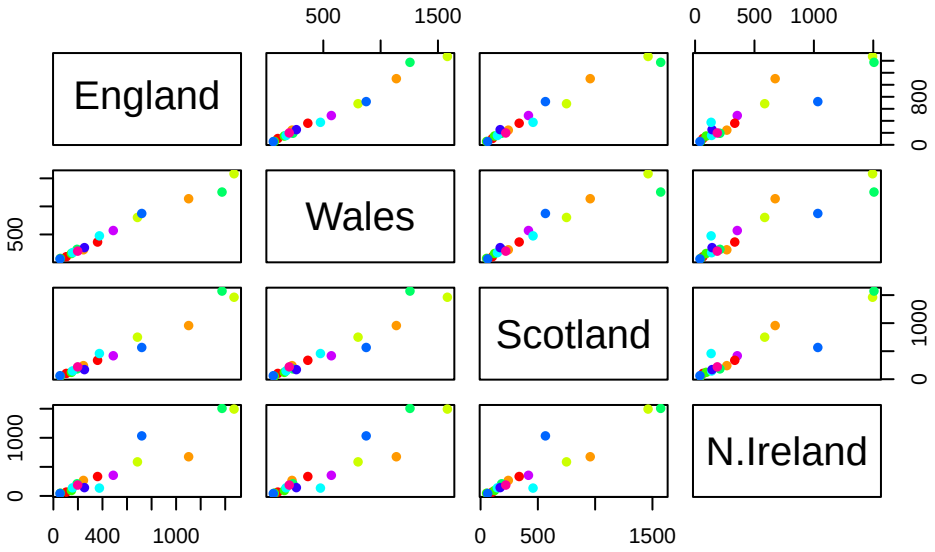
Q3: Changing what optional argument in the above barplot() function results in the following plot?

```
barplot(as.matrix(x), beside=F, 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)
```



This is plotting each country against one another. If a given point lies on the diagnal for the given plot if the points are exactly the same for each country.

> Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

There is a clear difference between N. Ireland and the other countries as there are many dots that are not on the diagonal line (orange, dark blue, etc. are all pretty obviously not on the line).

PCA to the rescue!

The main "base" R function for PCA is called `prcomp()`.

need to transpose the dataset so the countries are the rows and the food catgeories as the rows

Here we need to take the transpose of our input as we want the countries in the rows and the food in the columns.

```
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
```

```r
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
```

This tells us that PC1 captures 67.44% of variance of the dataset, and so on. PC2 should capture less, etc. If you made a graph of PC1 and PC2, you would captue 96.5% of the spread. We can plot this data with good confidence that we captured the majority of the data. This is how we know our PCA is doing a good job.

Q. How much variance is captured in 2 PCs?

96.5%

To make our main "PC score plot" (a.k.a. "PC1 vs. PC2 plot", "PC plot", or "ordination plot").

```
attributes(pca)
```

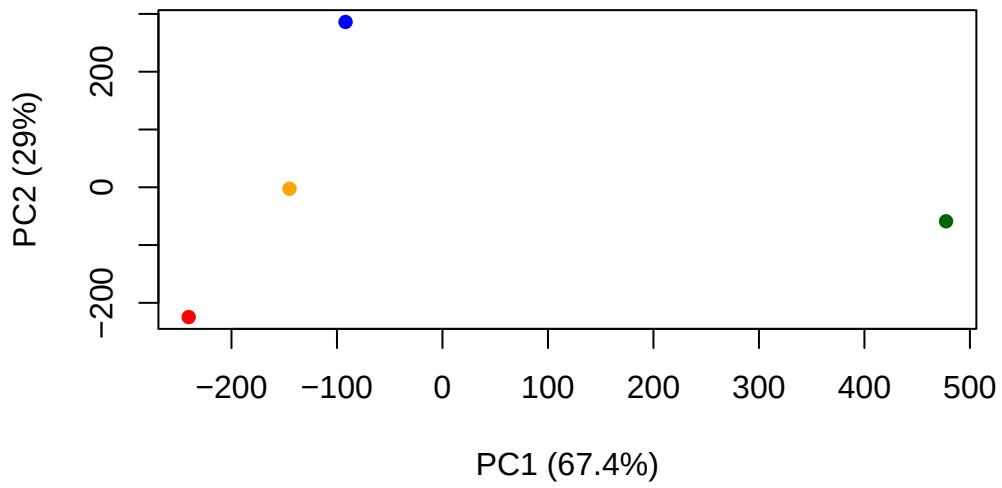
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$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 -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
```

```
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 is how the original variables (in this case, the foods) contribute to the PCs.

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

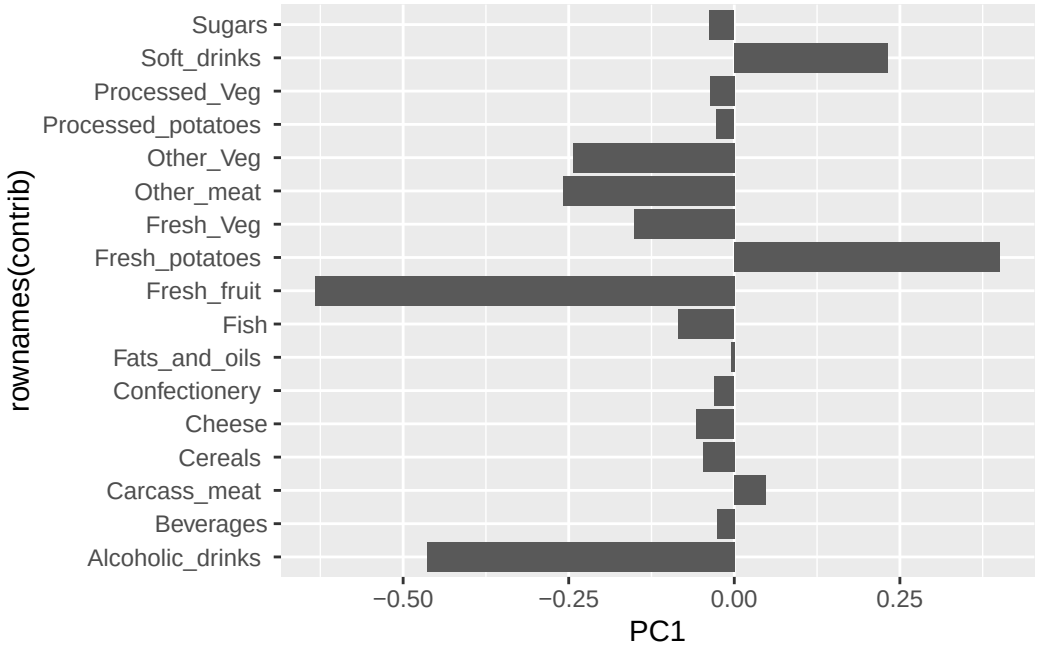
```
pca$rotation
```

```
                          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
```

This shows numbers of all original categories of all PC columns. The magnitude of the volumes represent the contribution to the PCA.

We can make a plot along PC1.

```
library(ggplot2)

contrib <- as.data.frame(pca$rotation)

ggplot(contrib) +
  aes(PC1, rownames(contrib)) +
  geom_col()
```



The more positive the value, the more Ireland consumes, the more negative the value, the more the other countries consumes.