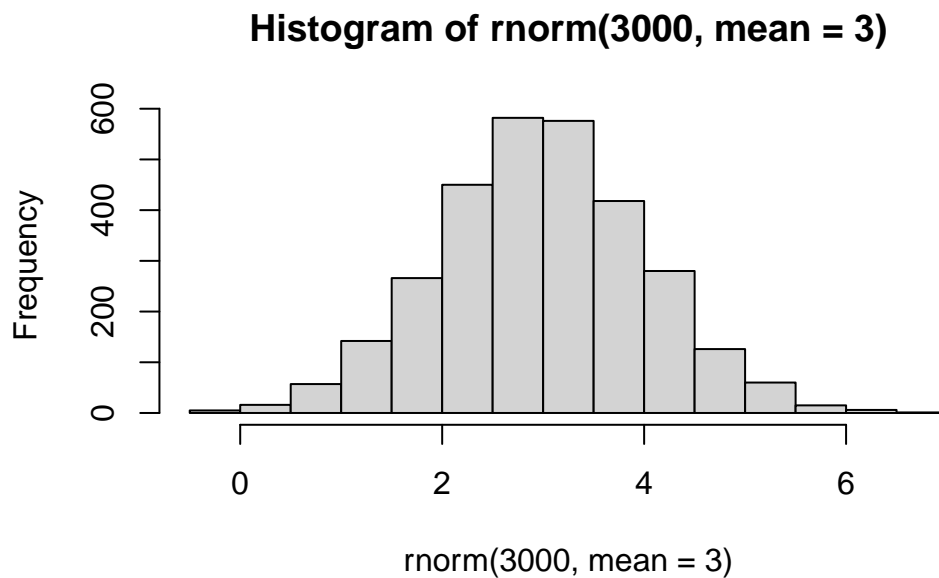# Class7: Machine Learning1

Hanhee Jo (PID: A59017994)

Today we will delve into unsupervised machine learning with a initial focus on clustering and dimensionality reduction.

Let's start by making up some data to cluster: Ther `rnorm()` function can help us here...

```r
hist( rnorm(3000, mean = 3) )
```
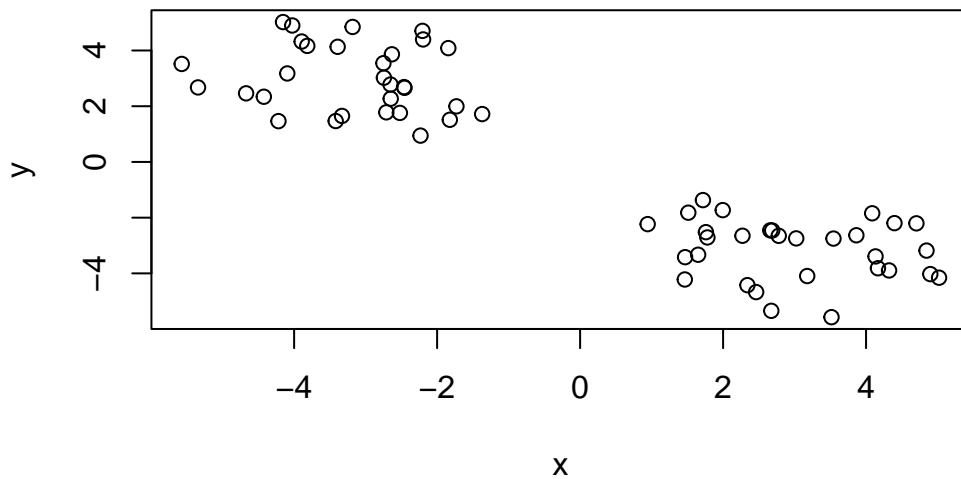
Histogram of rnorm(3000, mean = 3)

Let's get some data centered at (3, -3) (-3, 3)

```r
# Combine 30 +3 values with 30 -3 values
x <- c( rnorm(30, mean = 3), rnorm(30, mean = -3) )
```

```
# Creat the matrix
z <- cbind(x=x, y=rev(x))
```

```
plot(z)
```


K-means

Now we can see how K-means clusters this data. The main function for K-means clustering in "base R" is called `kmeans()`

```
km <- kmeans(z,centers=2)
km
```

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

Cluster means:
          x          y
1 -3.150583  2.995548
2  2.995548 -3.150583
```

```
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 1

Within cluster sum of squares by cluster:
[1] 76.65035 76.65035
 (between_SS / total_SS =  88.1 %)

Available components:

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

```
attributes(km)
```

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

$class
[1] "kmeans"
```

Q. What size is each cluster?

```
km$size
```

```
[1] 30 30
```

Q. The cluster of membership vector (i.e. the answer: cluster to which each point is allocated)

```
km$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 1
```
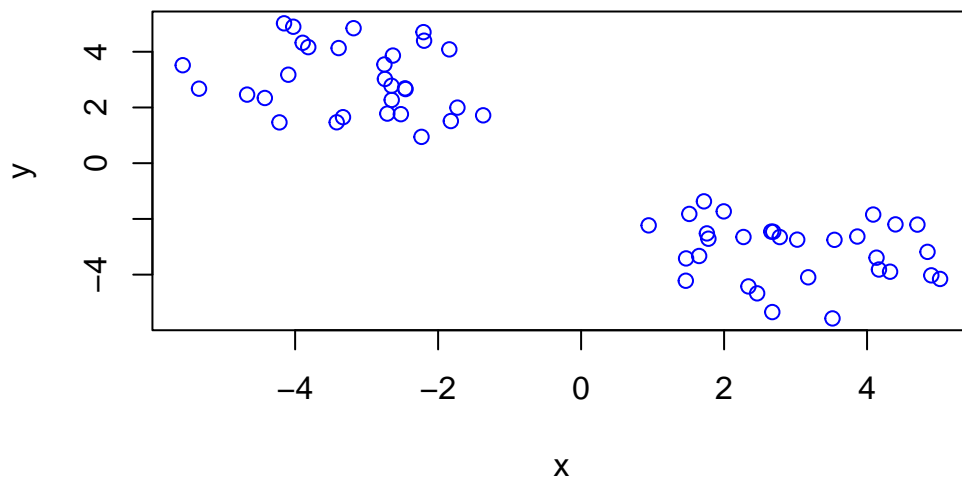
Q. Cluster center

```
km$centers
```

```
           x          y
1 -3.150583   2.995548
2  2.995548  -3.150583
```
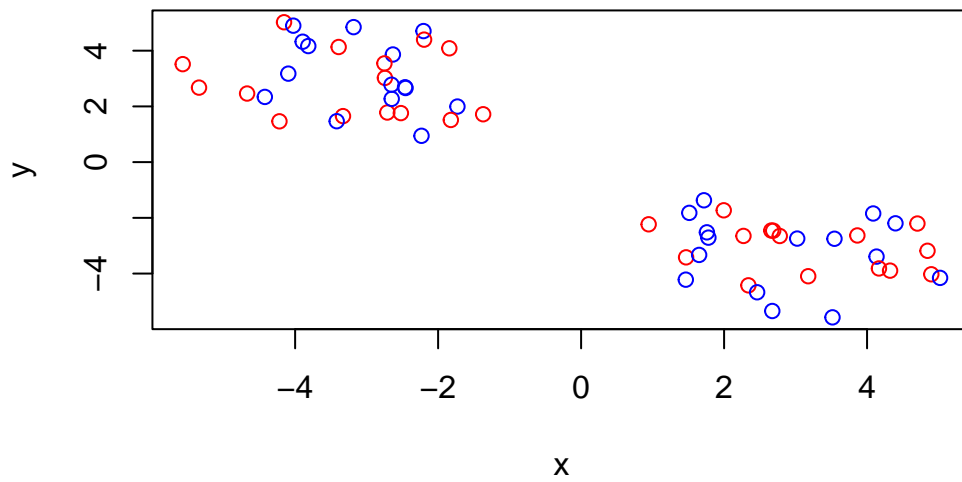
Q. Make a results figure, i.e. plot the data z colored by cluster membership and show the cluster centers.

```
plot(z, col = "blue")
```
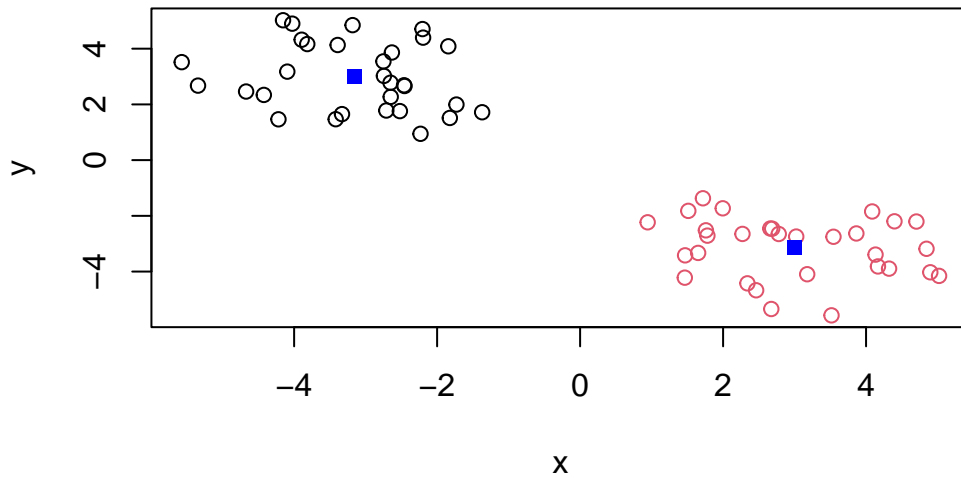


```
plot(z, col = c("red", "blue"))
```

You can specify color based on a number, where 1 is black, 2 is red.

```
plot(z, col = km$cluster)
points(km$centers, col = "blue", pch=15)
```

Q. Re-run your K-means clustering and as for 4 clusters and plot the results as above.

```
km_4 <- kmeans(z,centers=4)
km_4
```

```
K-means clustering with 4 clusters of sizes 14, 14, 16, 16

Cluster means:
          x         y
1 -2.594047  1.979740
2  1.979740 -2.594047
3 -3.637552  3.884380
4  3.884380 -3.637552

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

Within cluster sum of squares by cluster:
[1] 11.77462 11.77462 29.65875 29.65875
 (between_SS / total_SS =  93.6 %)
```

```
Available components:

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

```r
plot(z, col = km_4$cluster)
points(km_4$centers, col = "blue", pch = 15)
```
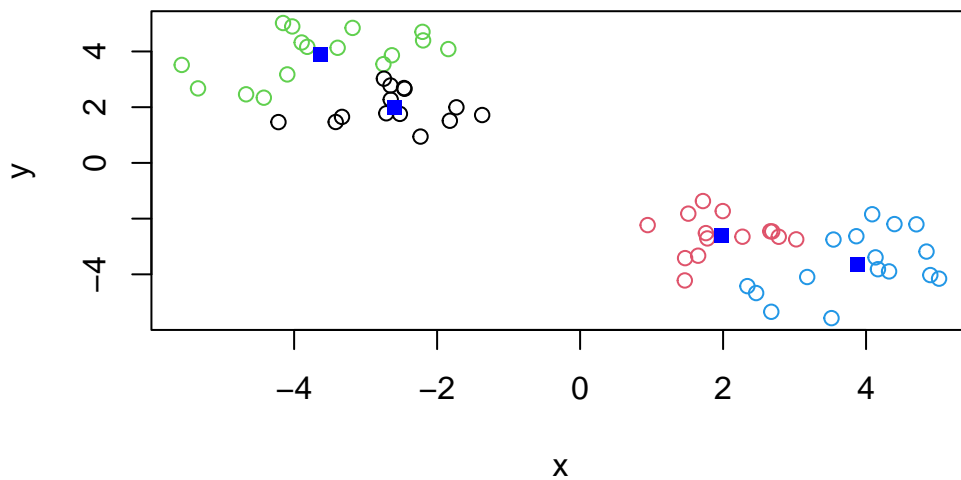

Hierarchical Clustering

The main "base R" finction for this is `hclust()`. Unlike `kmeans()` you can't just give your
dataset as input, you need to provide a distance matrix.

We can use `dist()` function for this

```r
d <- dist(z)
# hclust()
```

```r
dim(z)
```

```
[1] 60  2
```

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

```
Call:
hclust(d = d)

Cluster method   : complete
Distance         : euclidean
Number of objects: 60
```
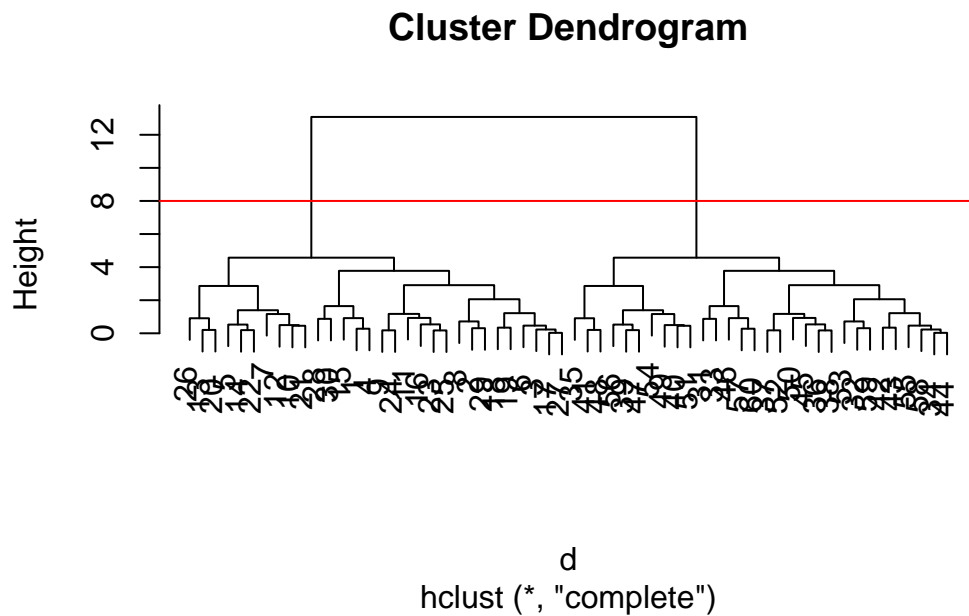
There is a custom plot() for hclust objects, let's see it.

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

Cluster Dendrogram

d
hclust (*, "complete")

The function to extract clusters/grps from a hclust object/tree is called `cutree()`:

```
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 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
```

Q. Plot data with hclust clusters:

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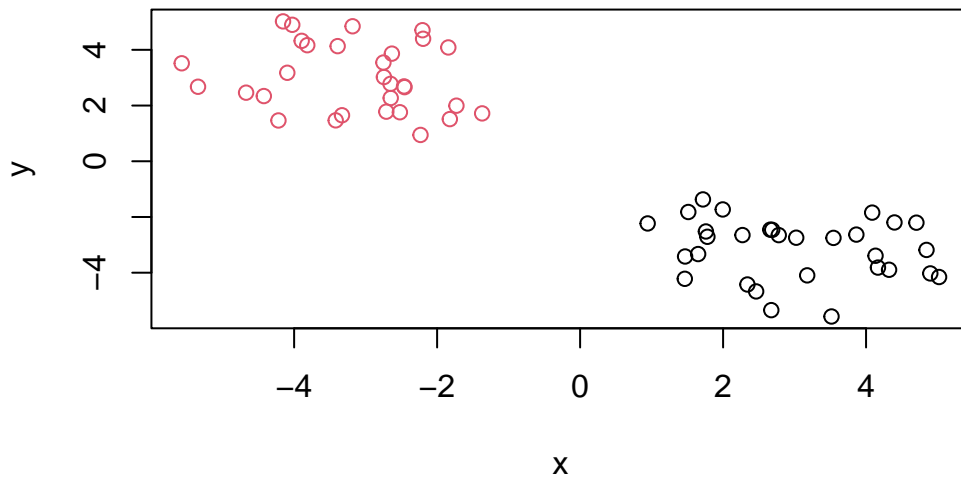

Principal Component Analysis (PCA)

The main function for PCA in base R for PCA is called `prcomp()` There are many add on packages with PCA functions tailored to particular data types (RNAseq, protein structures, metagenomics, etc.)

PCA of UK food data

Read the data into R, it is a CSV file and we can use `read.csv()` to read it:

```
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
```

I would like the food names as row names not their own colum of data (first colum currently).
I can fix this like so:

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

```
                   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
```

A better way to do this is to do it at the time of data import with `read.csv()`

```r
food <- read.csv(url, row.names = 1)
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
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
```
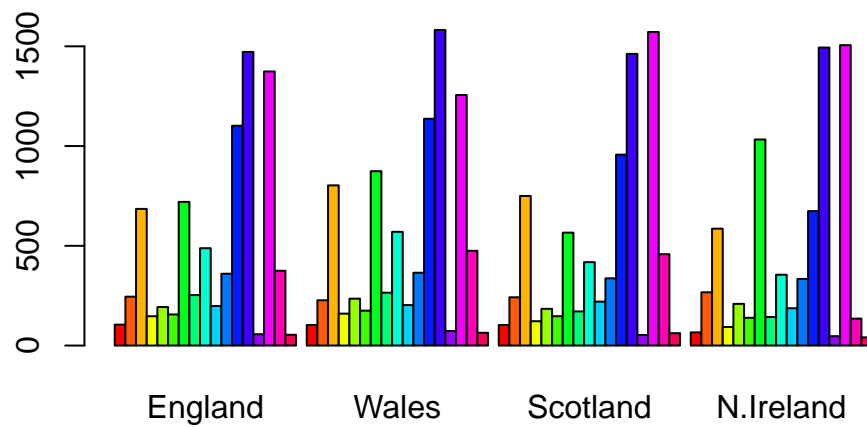
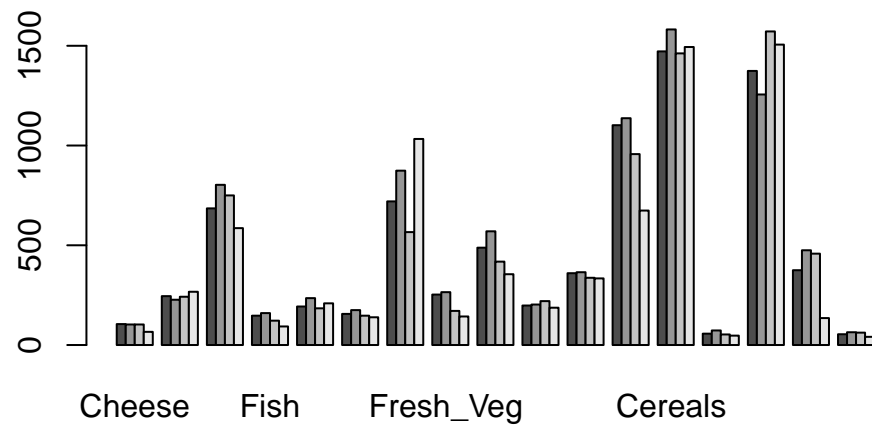Let's make some plots and dig into the data a little.

```r
rainbow(nrow(food))
```

```
 [1] "#FF0000" "#FF5A00" "#FFB400" "#F0FF00" "#96FF00" "#3CFF00" "#00FF1E"
 [8] "#00FF78" "#00FFD2" "#00D2FF" "#0078FF" "#001EFF" "#3C00FF" "#9600FF"
[15] "#F000FF" "#FF00B4" "#FF005A"
```

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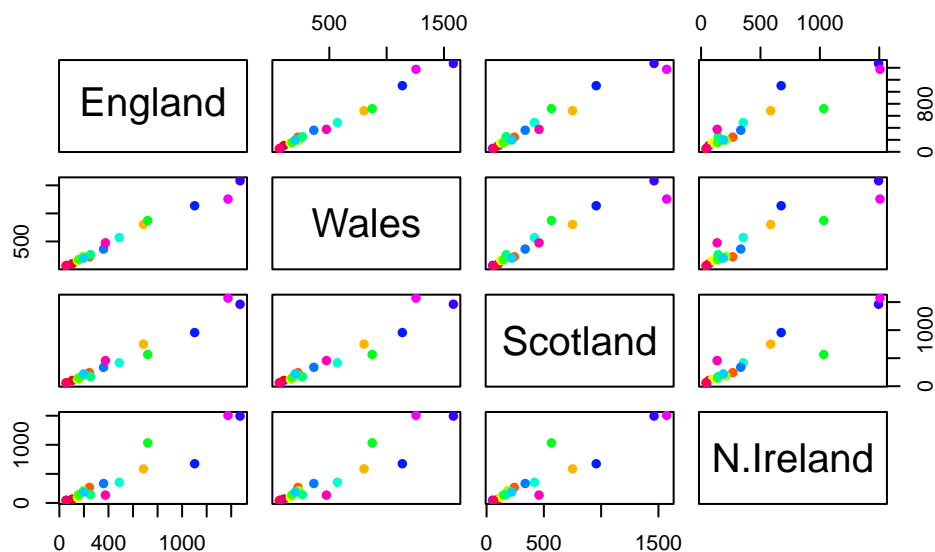


```
barplot(as.matrix(t(food)), beside=T)
```

How about a so-called "pairs" plot where we plot each country against all other contries.

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

Really there has to be a better way...

PCA to the rescue!

We can run a Principal Component Analysis (PCA) for this data with the `prcomp()` function.

```
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

We need to take the transpose of this data to get the foods in the columns and the countries in the rows.

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

What is in my `pca` result object?

```r
attributes(pca)
```

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


$class
[1] "prcomp"
```

```r
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
```

To make my main result figure, sometimes called a PC plot (or score plot, oridenation plot, or
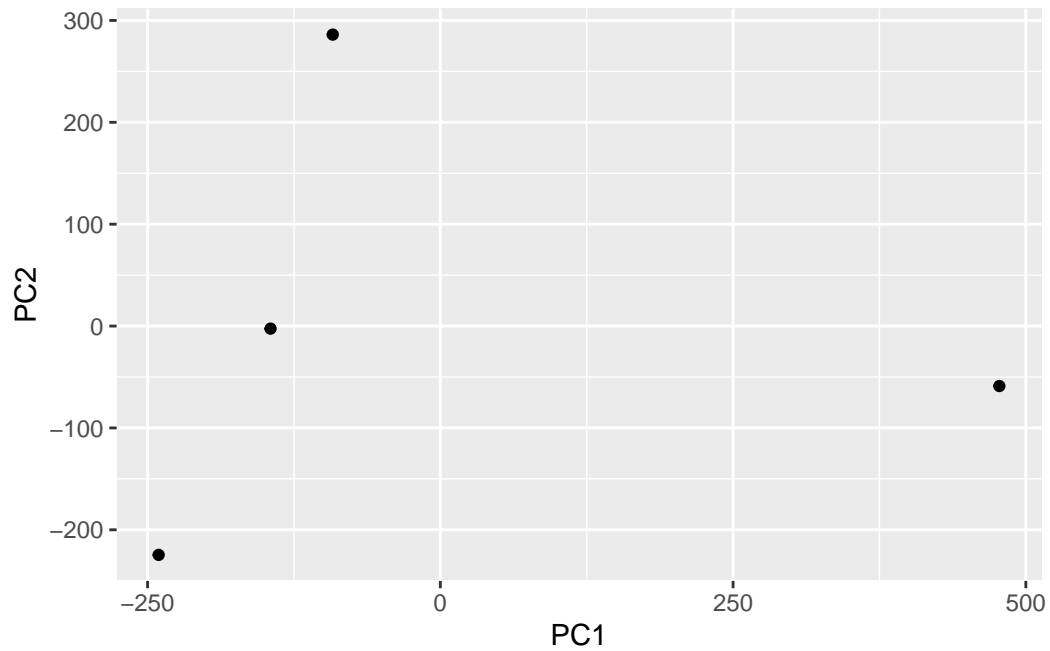PC1 vs PC2 plot, etc.)

```r
plot(pca$x[,1], pca$x[,2], xlab = "PC1", ylab = "PC2", col = c("orange", "red", "blue", "d
```
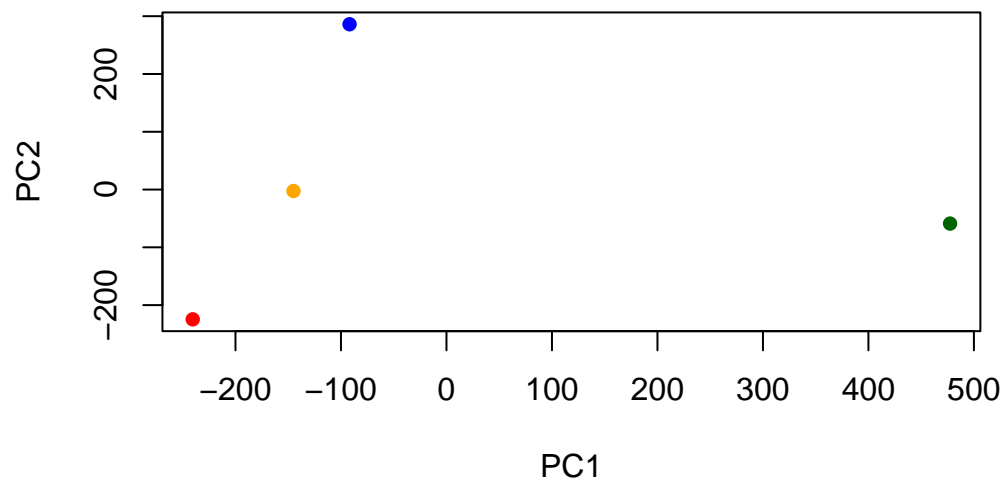
```
library(ggplot2)

data <- as.data.frame(pca$x)

ggplot(data, aes(PC1, PC2)) +
  geom_point()
```
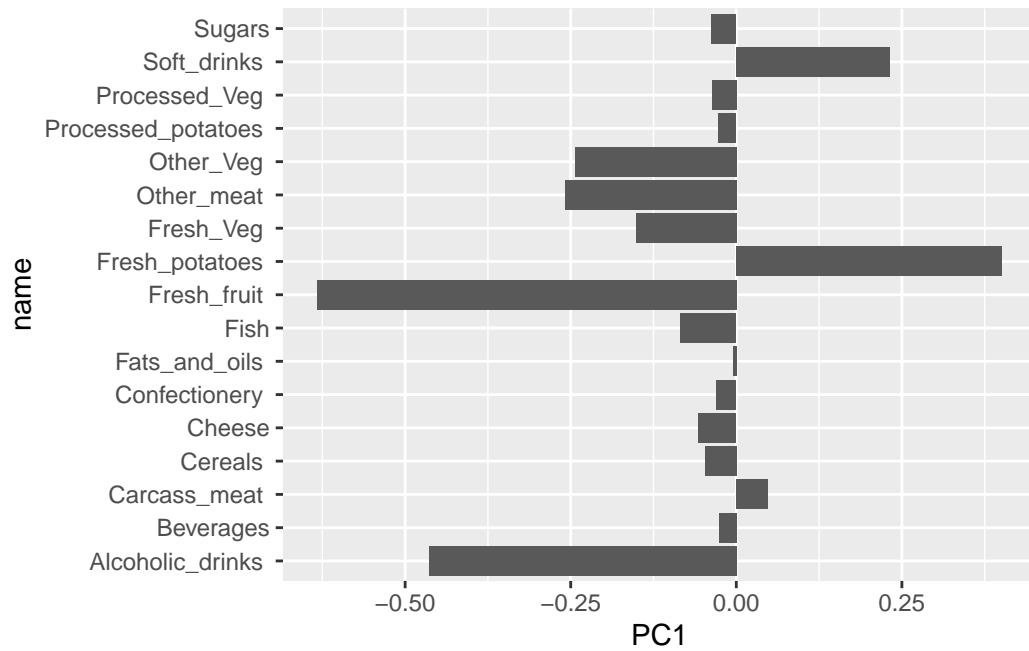
To see the contributions of the original variables (foods) to these new PCs we can look at the
`pca$rotation` component of our results object.

```
loadings <- as.data.frame(pca$rotation)
name <-  rownames(loadings)

ggplot(loadings) +
 aes(PC1, name) +
 geom_col()
```

And PC2

```
ggplot(loadings) +
 aes(PC2, name) +
 geom_col()
```