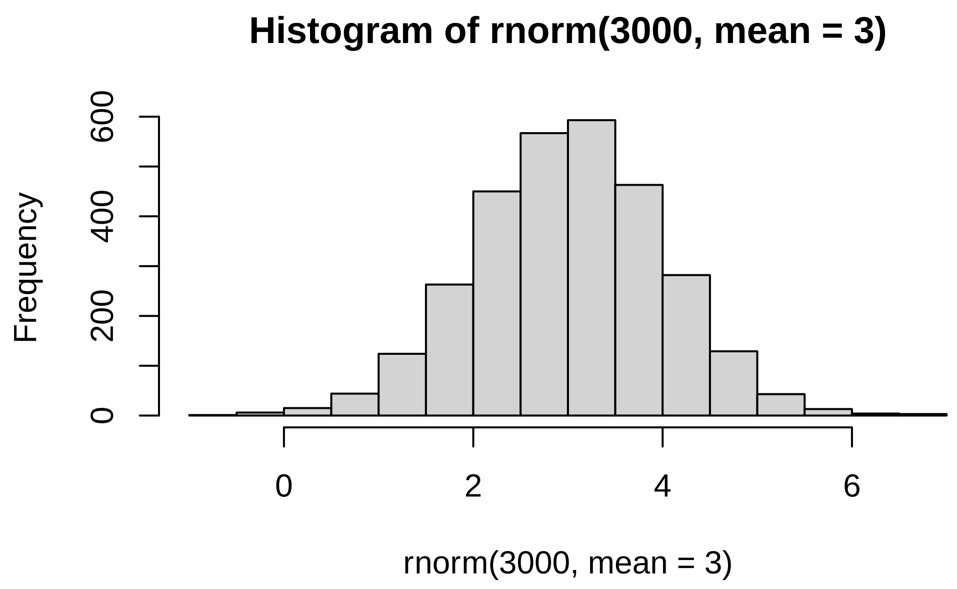
# Class 7: Machine Learning 1

Yiyu

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

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

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



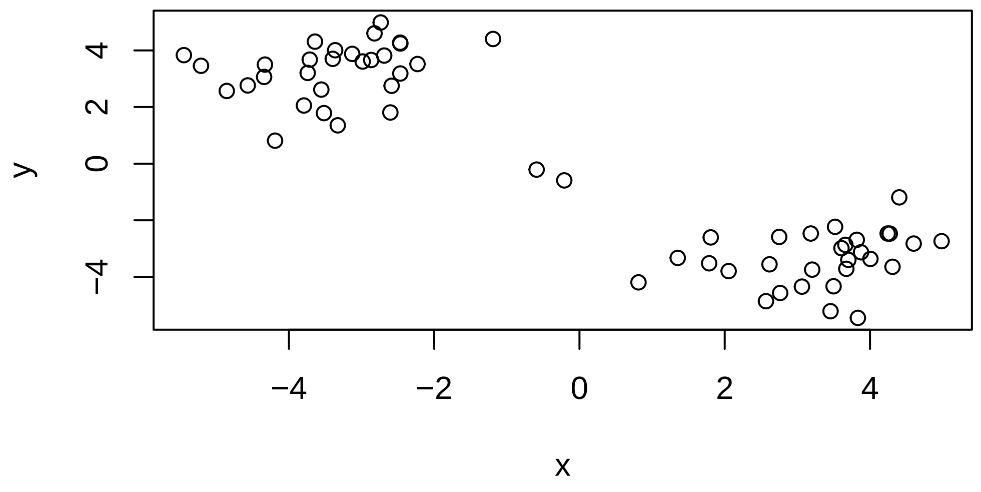
Let's get some data centered at 3,-3 and -3,3

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

#Bind these values together
```

```r
z <- cbind(x=x,y=rev(x))

head(z)
```

```
             x         y
[1,] 2.6163811 -3.554387
[2,] 4.2727540 -2.468921
[3,] 3.1844101 -2.466591
[4,] 0.8127403 -4.190887
[5,] 3.0628106 -4.341727
[6,] 3.2041846 -3.742074
```

```r
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()`.

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

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

```
Cluster means:
         x        y
1 -3.29408  3.17404
2  3.17404 -3.29408

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] 73.52457 73.52457
 (between_SS / total_SS =  89.5 %)

Available components:

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

> Q. What size is each cluster?

```r
km$size
```

```
[1] 30 30
```

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

```r
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
```

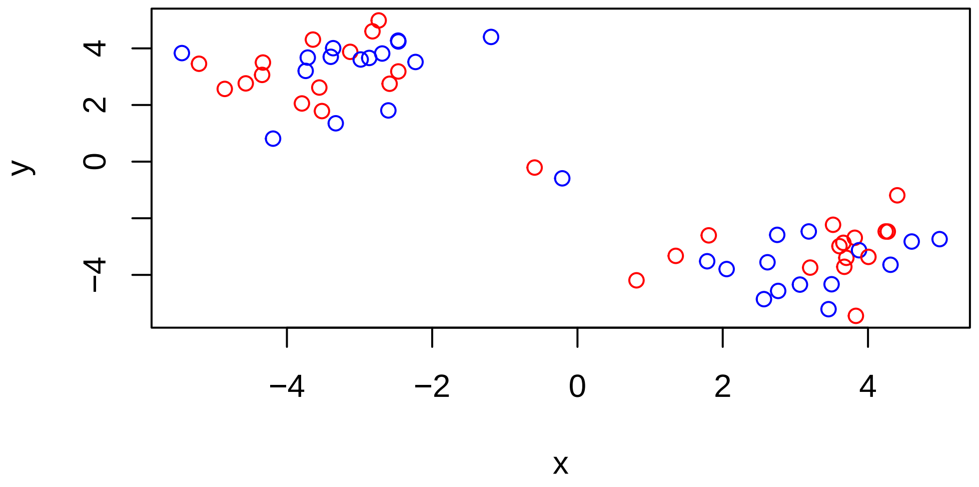
> Q. Cluster centers

```r
km$centers
```

```
         x        y
1 -3.29408  3.17404
2  3.17404 -3.29408
```

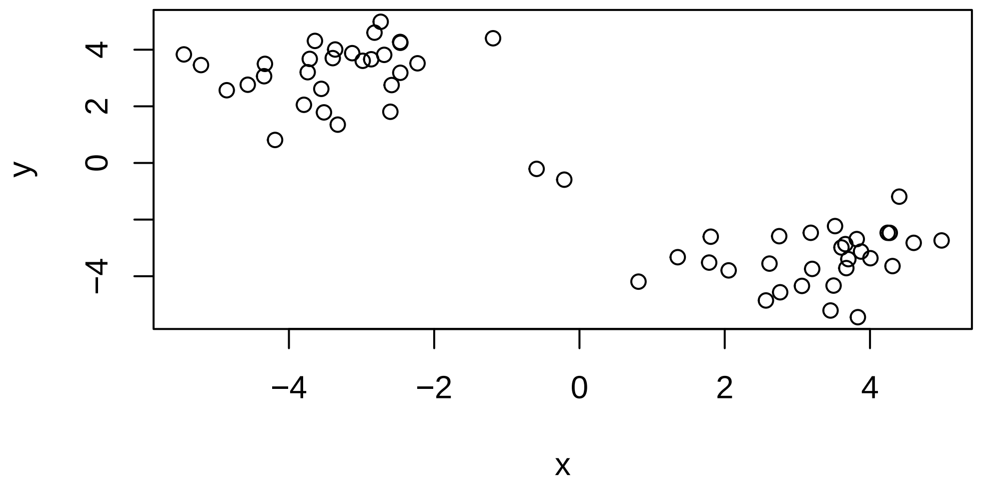
Q. Make a results figure , i.e plot the data `z` colored by cluster membership and show cluster centers.

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



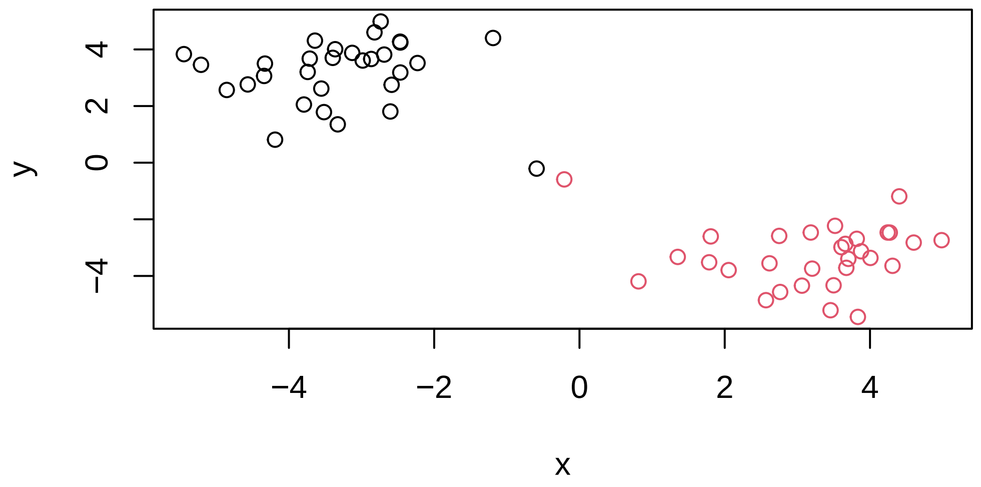
Uses recycling, so it will mix colors within clusters. You can color based on a number, which will be useful here.

```r
plot(z, col = 1)
```



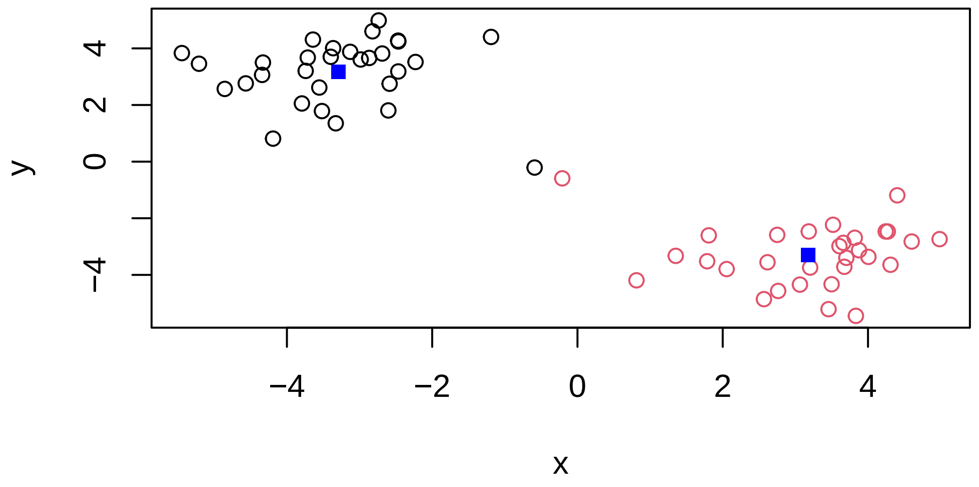
We know `km$cluster` categorizes clusters into 1 or 2, so color by `km$cluster` will color the clusters with separate colors.

```r
plot(z, col = km$cluster)
```



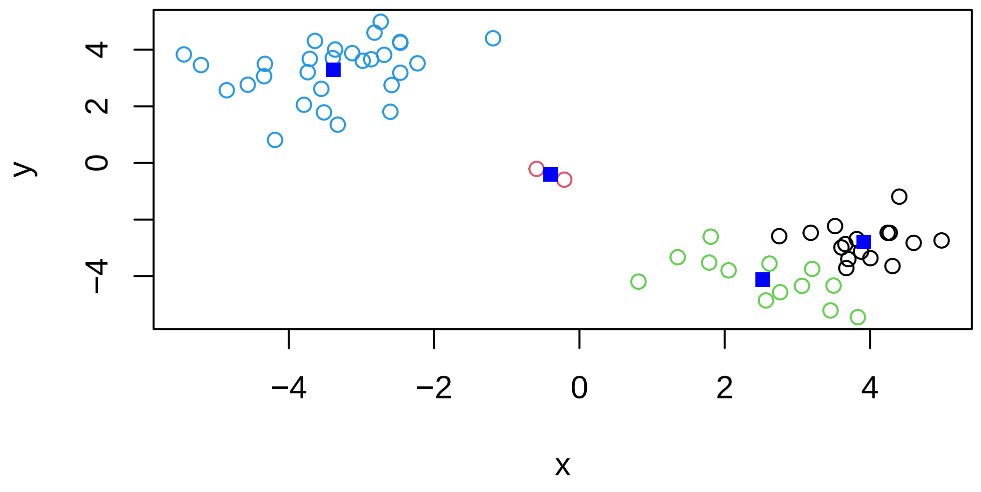
Can also color cluster center

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



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

```r
km4 <- kmeans(z, centers=4)
plot(z, col=km4$cluster)
points(km4$center, col = "blue", pch=15)
```



##Hierarchial Clustering

The main "base R" function 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 the `dist()` function for this

```
d <-dist(z)
```

```
dim(z)
```

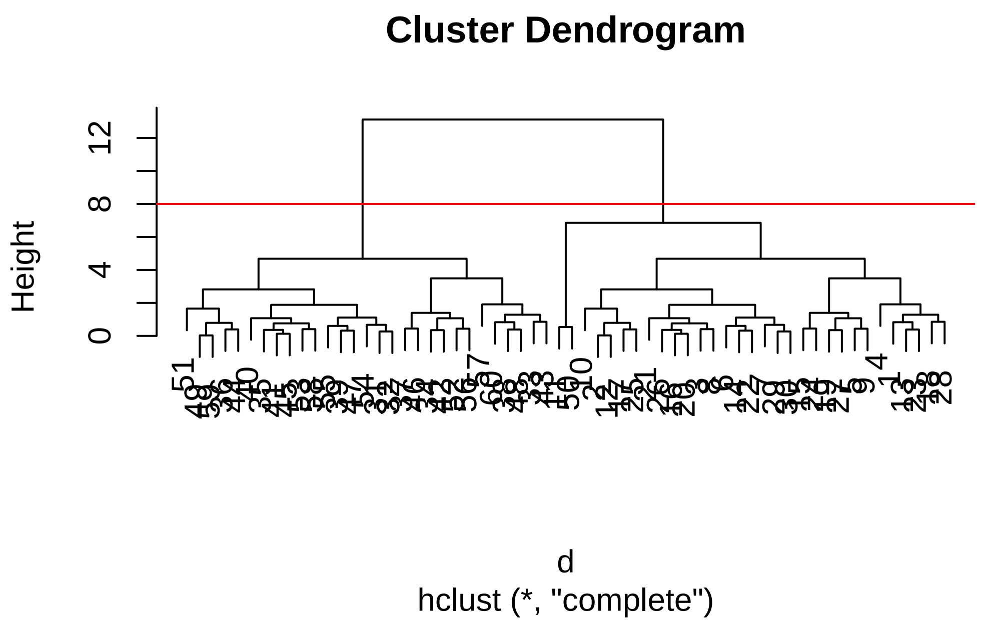
```
[1] 60  2
```

```
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")
```



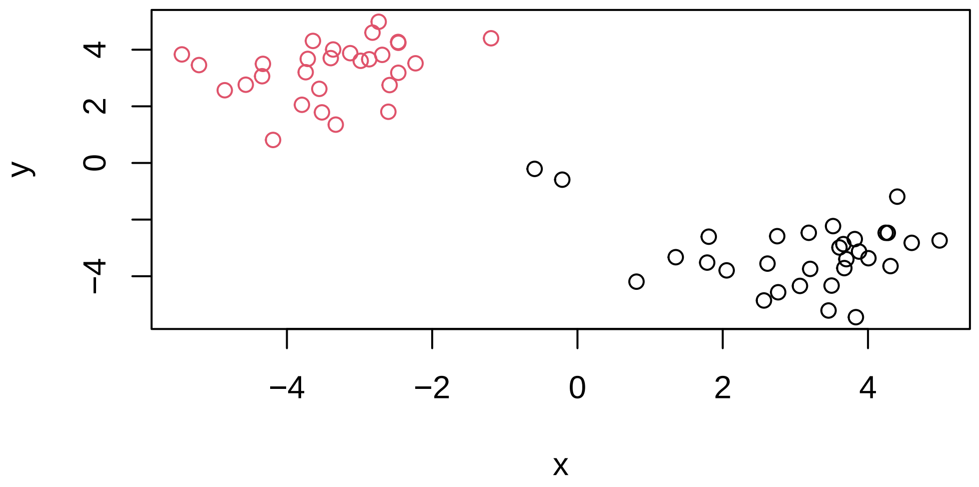
The function to extract clusters/grps from an hclust object/tree is called `cutree()`:

```
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 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 1 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, 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:

```r
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 column of data (first column currently). I can fix this like so:

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

Could be risky, if you run it multiple times you will continue to delete the first column.

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
```

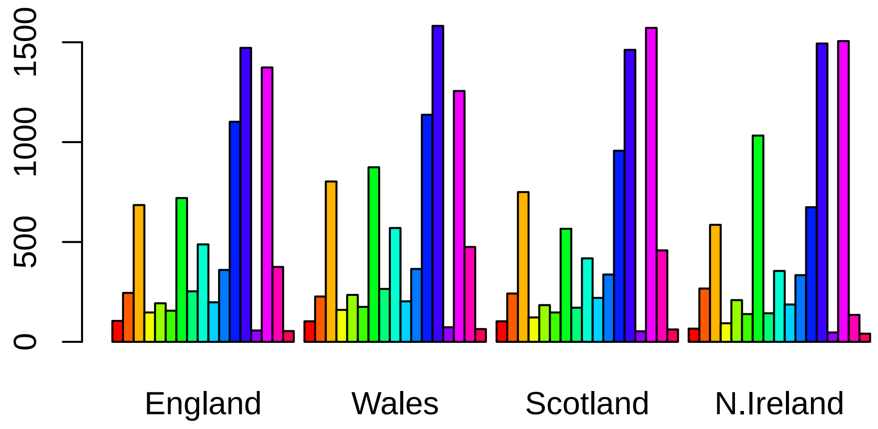
```
                    England Wales Scotland N.Ireland
Cheese                  105   103      103        66
Carcass_meat            245   227      242       267
Other_meat              685   803      750       586
Fish                    147   160      122        93
Fats_and_oils           193   235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

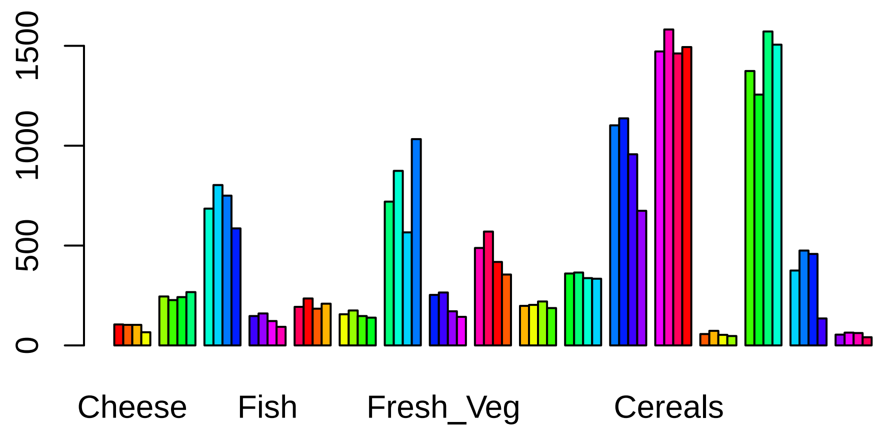
```r
rainbow(nrow(food))
```

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

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

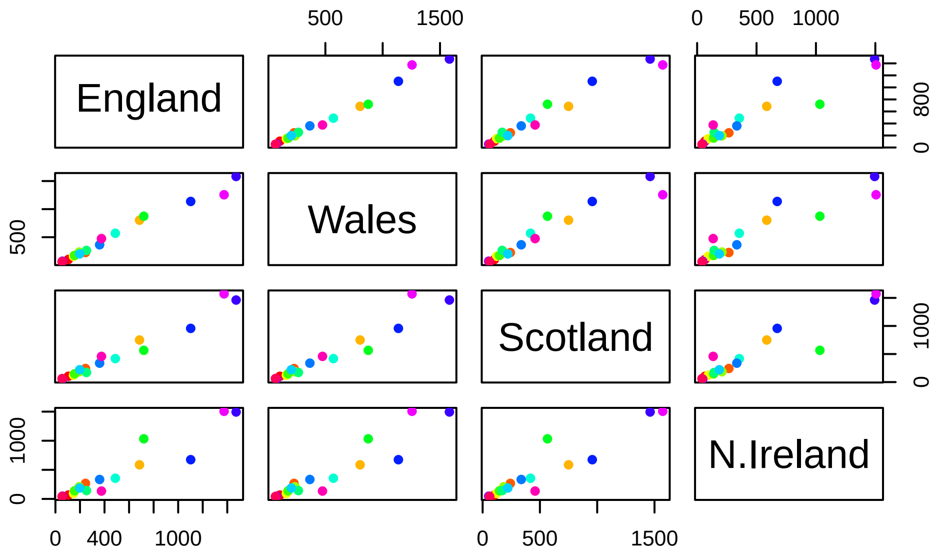


```r
barplot(as.matrix(t(food)), beside=T, col=rainbow(nrow(food)))
```



How about a so-called “pairs” plot where we plot each country against all other countries.

```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.

```r
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 column and the countries in the rows.

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

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 3.176e-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` resultobject?

```r
attributes(pca)
```

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

$class
[1] "prcomp"
```

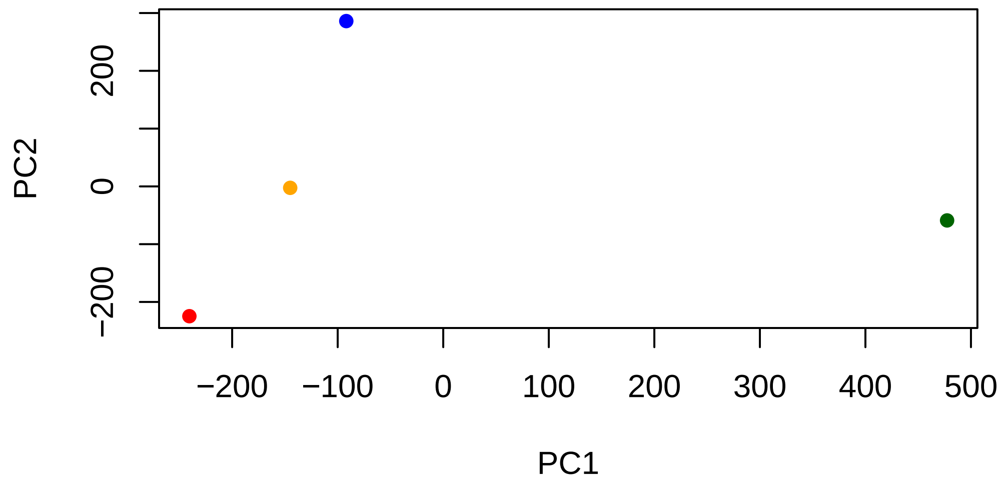
The scores along the new PCs

```r
pca$x
```

```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -4.894696e-14
Wales     -240.52915 -224.646925 -56.475555  5.700024e-13
Scotland   -91.86934  286.081786 -44.415495 -7.460785e-13
N.Ireland  477.39164  -58.901862  -4.877895  2.321303e-13
```

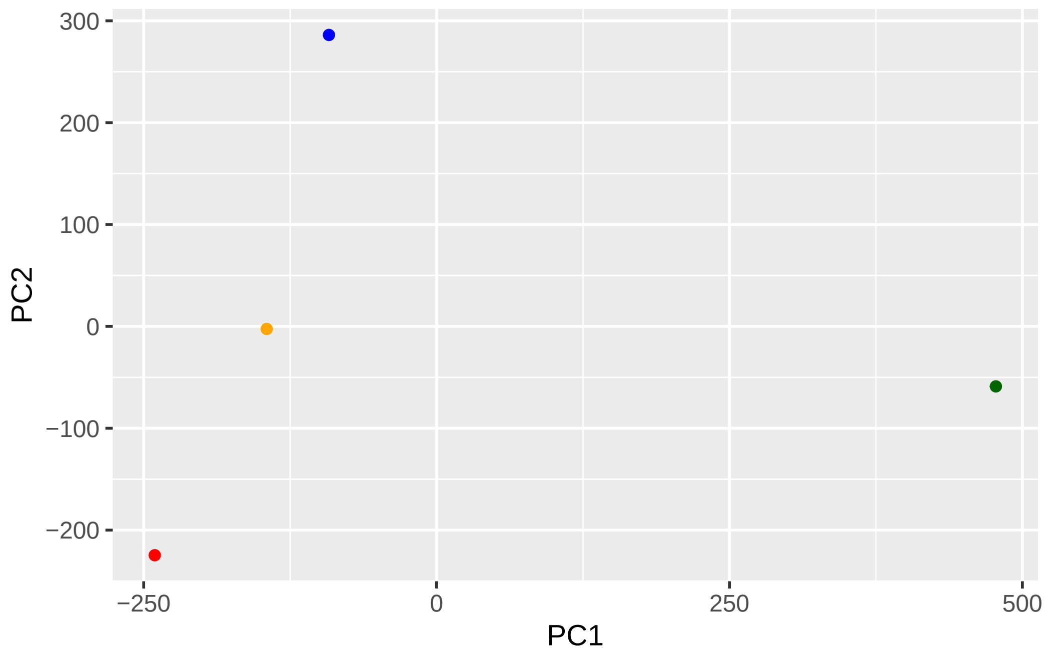
To make my main result figure, often called a PC plot (or score plot, ordenation plot, PC1 vs PC2 plot etc.)

```r
plot(pca$x[,1], pca$x[,2], xlab = "PC1", ylab= "PC2",
     col = c("orange", "red","blue","darkgreen"), pch=16)
```



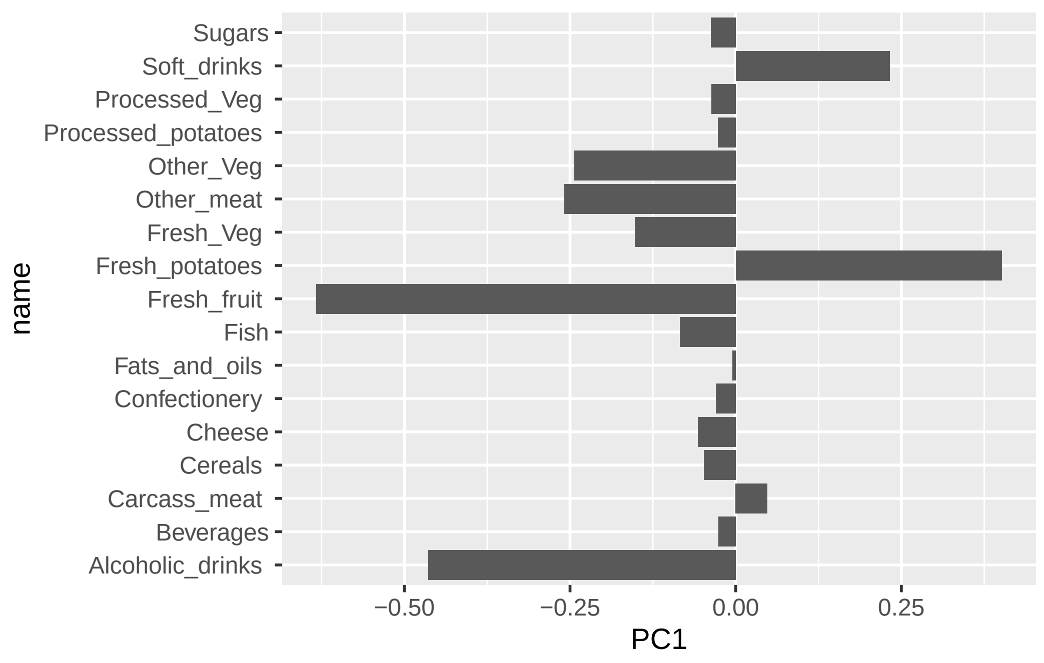
Try with ggplot 2

```r
library(ggplot2)

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

ggplot(data) +
  aes(PC1, PC2) +
  geom_point(col = c("orange", "red","blue","darkgreen"))
```



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$rotation)
loadings$name <- rownames(loadings)

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


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
name
−0.50
−0.25
0.00
0.25
PC1