Class 7: Machine Learning 1

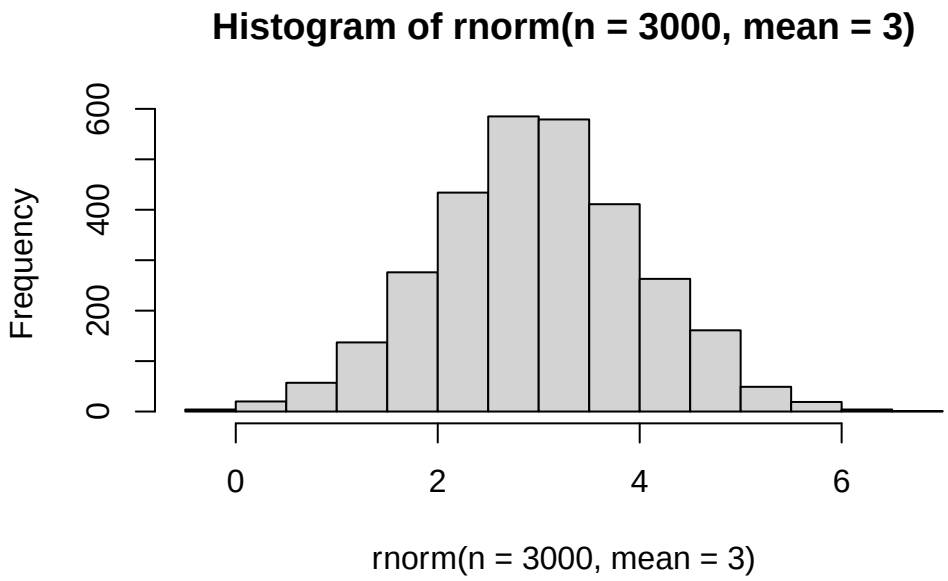
Barry (PID 911)

Today we will explore unsuperivised machine learning methods including clustering and dimensionallity reduction methods.

Let's start by making up some data (where we know there are clear groups) that we can use to test out different clustering methods.

We can use the `rnorm()` function to help us here:

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

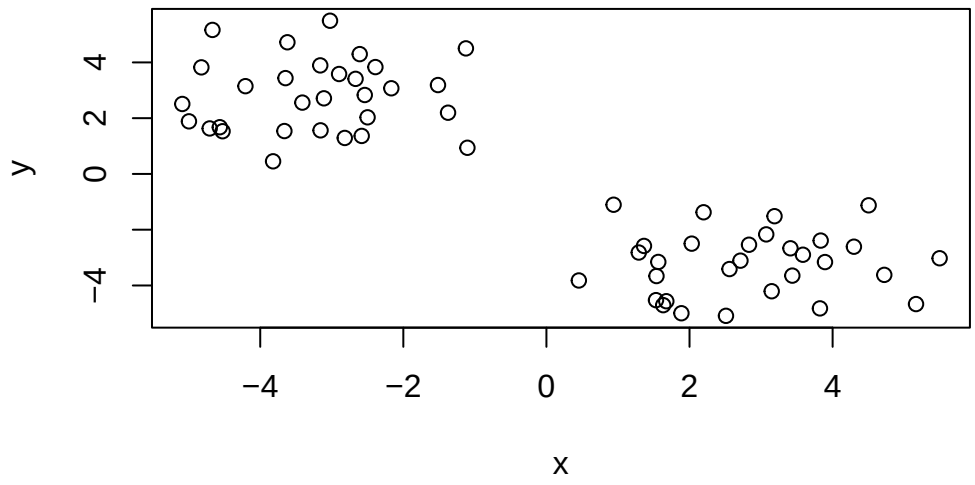


Make data `z` with two "clusters"

```r
x <- c( rnorm(30, mean=-3),
        rnorm(30, mean=+3) )

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

```
             x        y
[1,] -3.109838 2.711750
[2,] -2.609291 4.297772
[3,] -1.125207 4.503473
[4,] -4.564685 1.675743
[5,] -4.667459 5.165766
[6,] -4.526165 1.531527
```

```r
plot(z)
```



How big is `z`

```r
nrow(z)
```

```
[1] 60
```

```
ncol(z)
```

```
[1] 2
```

K-means clustering

The main function in "base" R for K-means clustering is called `kmeans()`

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

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

Cluster means:
          x         y
1 -3.216026  2.809656
2  2.809656 -3.216026

Clustering vector:
 [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

Within cluster sum of squares by cluster:
[1] 87.66716 87.66716
 (between_SS / total_SS =  86.1 %)

Available components:

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

```
attributes(k)
```

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

$class
[1] "kmeans"
```

Q. How many points lie in each cluster?

```
k$size
```

```
[1] 30 30
```

Q. What component of our results tells us about the cluster membership (i.e. which point likes in which cluster)?

```
k$cluster
```

```
 [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
```

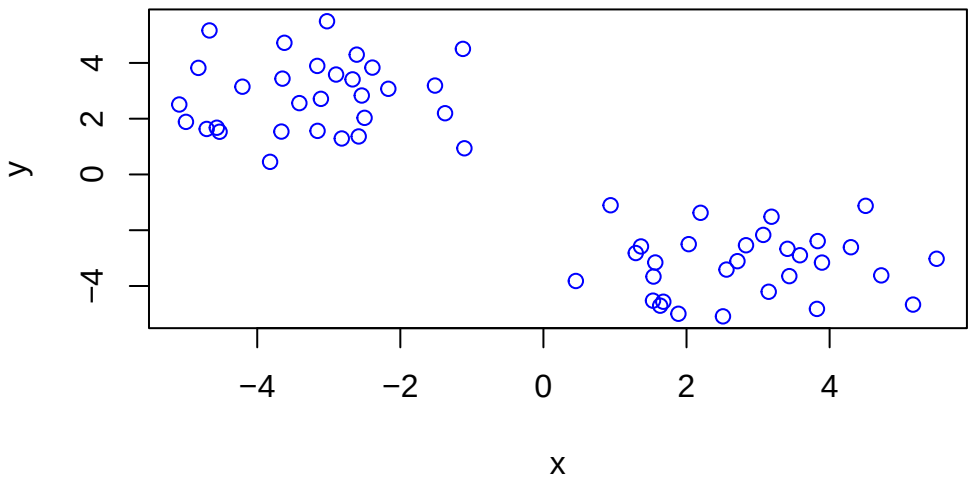
Q. Center of each cluster?

```
k$centers
```

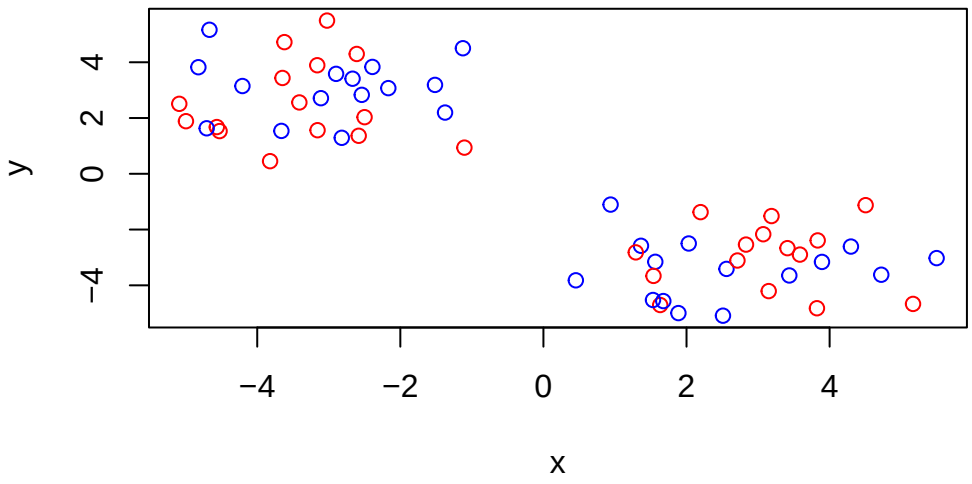
```
          x         y
1 -3.216026  2.809656
2  2.809656 -3.216026
```

Q. Put this result info together and make a little "base R" plot of our clustering result. Also add the cluster center points to this plot.

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

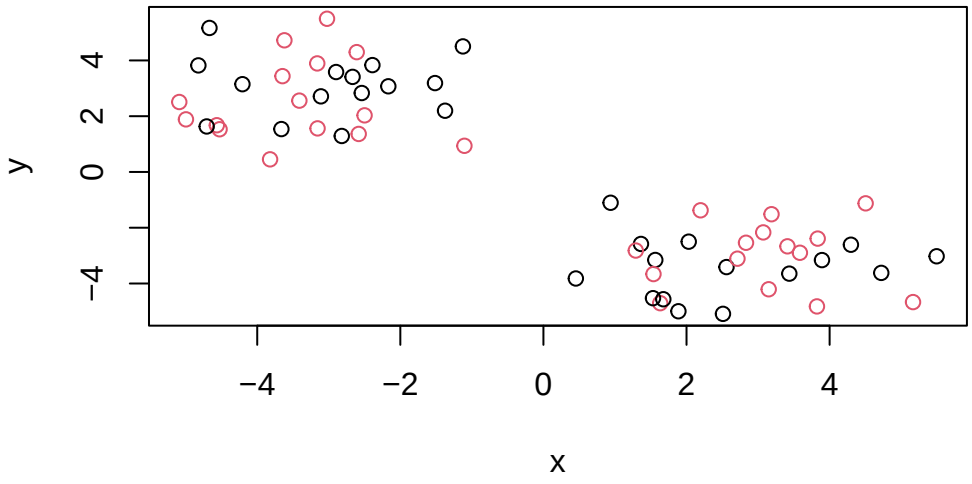



```
plot(z, col=c("blue","red"))
```



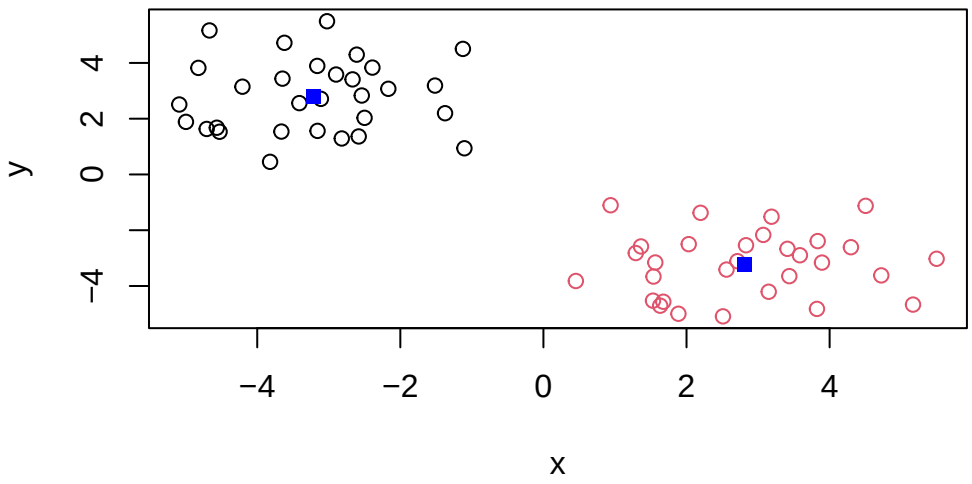
You can color by number.

```
plot(z, col=c(1,2))
```



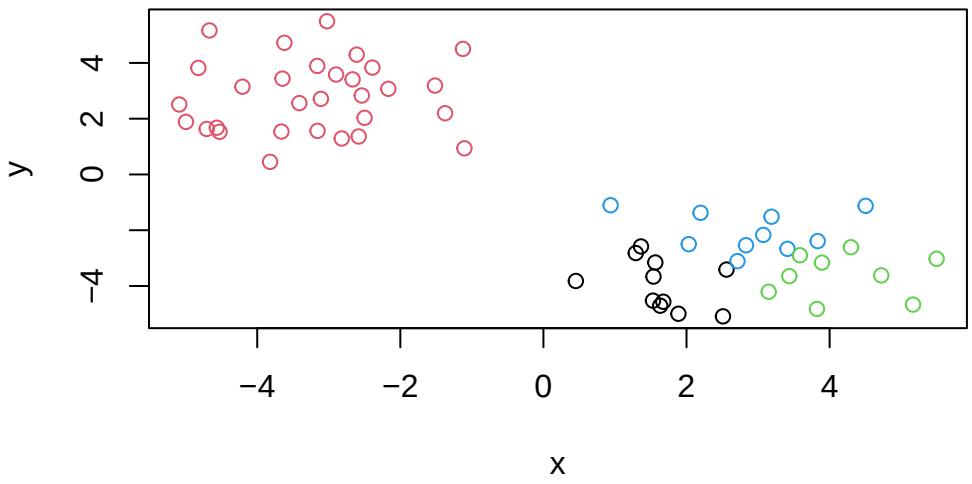
Plot colored by cluster membership:

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



Q. Run kmeans on our input **z** and define 4 clusters making the same result vizualization plot as above (plot of z collored by cluster memnbership).

```
k4 <- kmeans(z, centers = 4)
plot(z, col=k4$cluster)
```



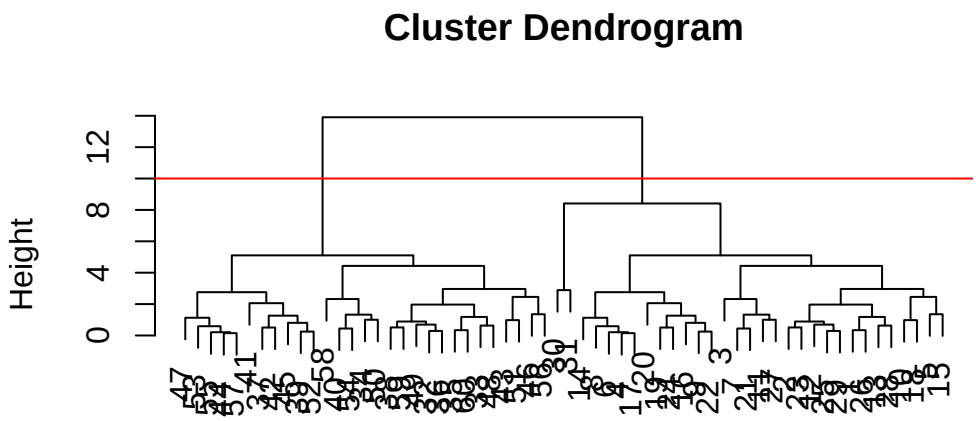
Hierarchical Clustering

The main function in base R for this called `hclust()` it will take as input a distance matrix (key point is that you can't just give your "raw" data as input - you have to first calculate a distance matrix from your data).

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

```
Call:
hclust(d = d)

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

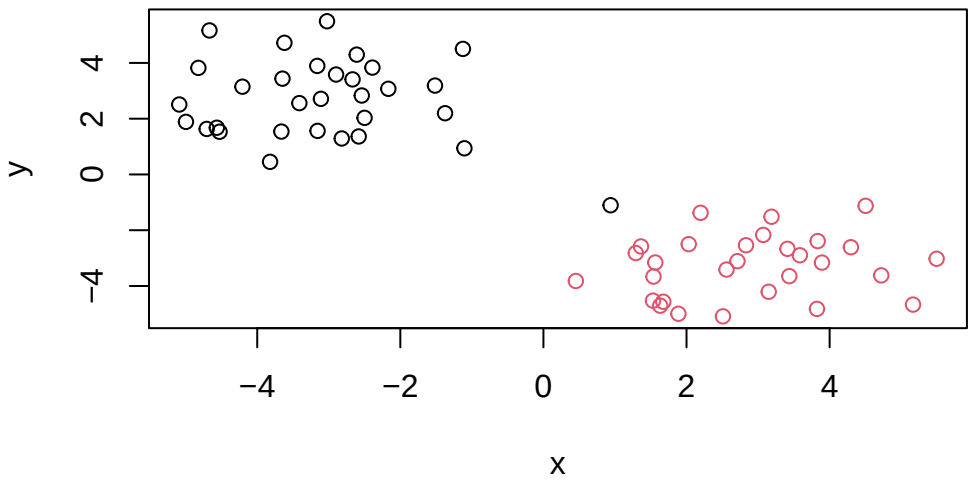
```
plot(hc)
abline(h=10, col="red")
```



Once I inspect the "tree" I can "cut" the tree to yield my groupings or clusters. The function to this is called `cutree()`

```
grps <- cutree(hc, h=10)
```

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



Hands on with Principal Component Analysis (PCA)

Let's examine some silly 17-dimensional data detailing food consumption in the UK (England, Scotland, Wales and N. Ireland). Are these countries eating habits different or similar and if so how?

Data import

```
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?

```
nrow(x)
```

```
[1] 17
```

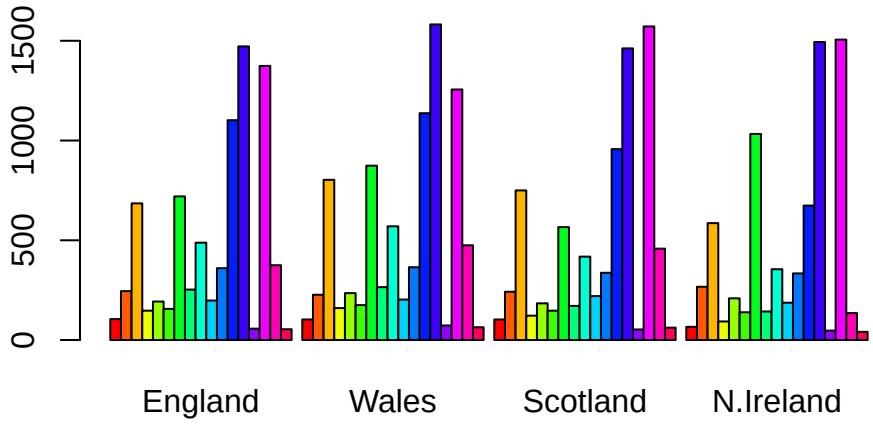
```
ncol(x)
```

```
[1] 4
```

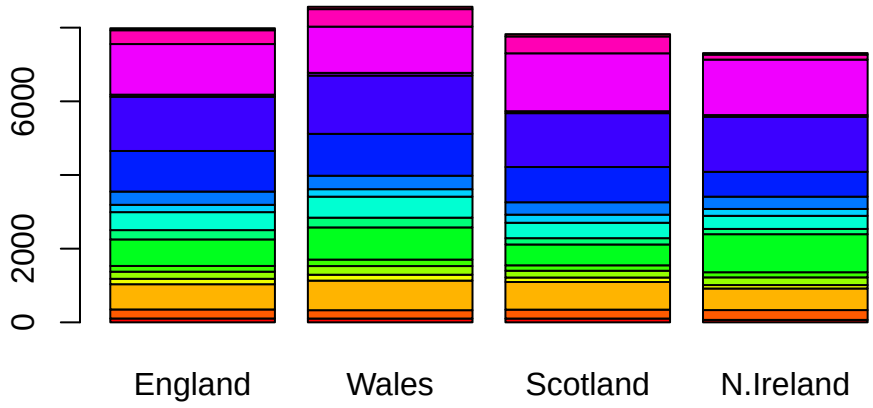
```
dim(x)
```

```
[1] 17  4
```

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



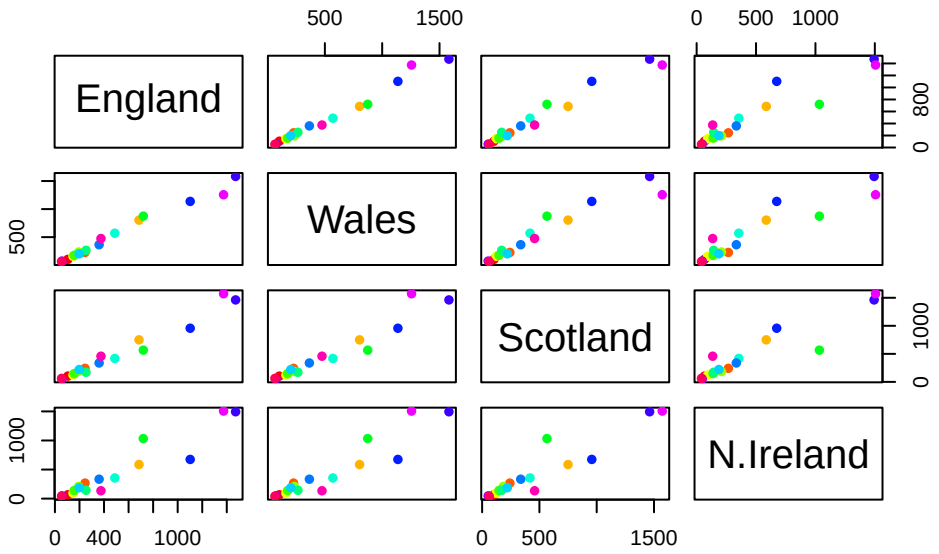

```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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(nrow(x)), pch=16)
```



Looking at these types of "pairwise plots" can be helpful but it does not scale well and kind of sucks! There must be a better way…

PCA to the rescue!

The main function for PCA in base R is called `prcomp()`. This function wants the transpose of our input data - i.e. the important foods in as columns and the countries as rows.

```
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
```

Let's see what is in our PCA result object `pca`

```
attributes(pca)
```

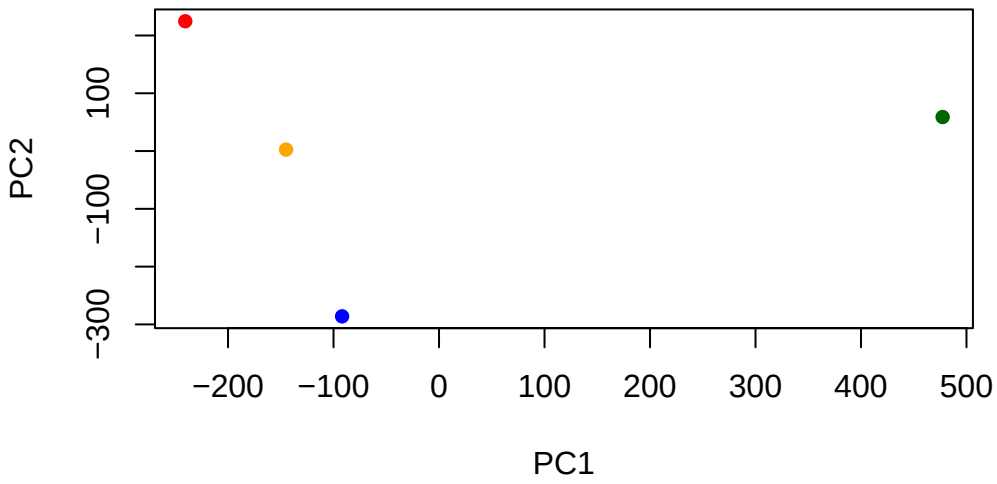
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

The `pca$x` result object is where we will focus first as this details how the countries are related to each other in terms of our new "axis" (a.k.a. "PCs", "eigenvectors", etc.)

```
head(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
```

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



We can look at the so-called PC "loadings" result object to see how the original foods contribute to our new PCs (i.e. how the original variables contribute to our new better PC variables).

```
pca$rotation[,1]
```

```
          Cheese       Carcass_meat         Other_meat               Fish
-0.056955380         0.047927628        -0.258916658        -0.084414983
Fats_and_oils              Sugars     Fresh_potatoes          Fresh_Veg
-0.005193623        -0.037620983         0.401402060        -0.151849942
   Other_Veg  Processed_potatoes       Processed_Veg         Fresh_fruit
-0.243593729        -0.026886233        -0.036488269        -0.632640898
     Cereals            Beverages       Soft_drinks    Alcoholic_drinks
-0.047702858        -0.026187756         0.232244140        -0.463968168
Confectionery
-0.029650201
```