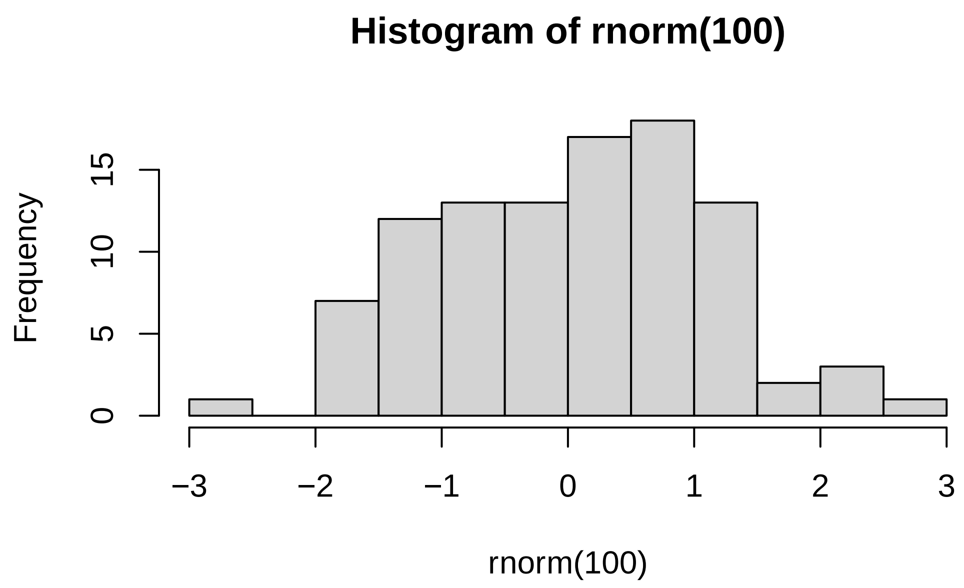
Class07: Machine learning

Bobbie Morales A15443382

Today we will begin our exploration of some "classical" machine learning approaches. We will start with clustering:

Let's first make up some data to cluster where we know what the answer should be.

```
hist(rnorm(100))
```

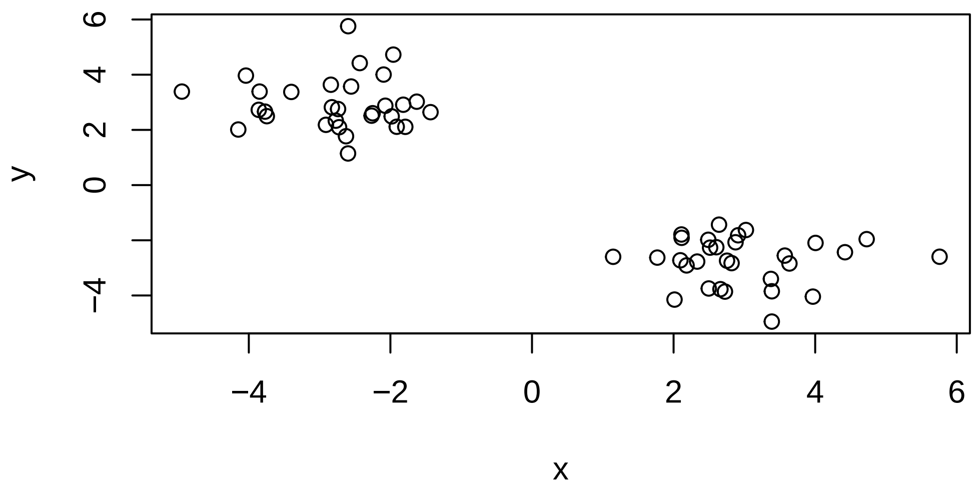


```
x <- c(rnorm(30, mean=-3), rnorm(30, mean=3))
y <- rev(x)

 x<- cbind(x, y)
 head(x)
```

```
             x        y
[1,] -4.148901 2.013092
[2,] -2.725030 2.095612
[3,] -2.071963 2.875000
[4,] -3.848006 3.387797
[5,] -2.598403 1.146123
[6,] -2.266136 2.515443
```

a wee peak at x with `plot()`

```r
plot(x)
```



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

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

> Q. How big are the clusters (i.e their size)

```r
k$size
```

```
[1] 30 30
```

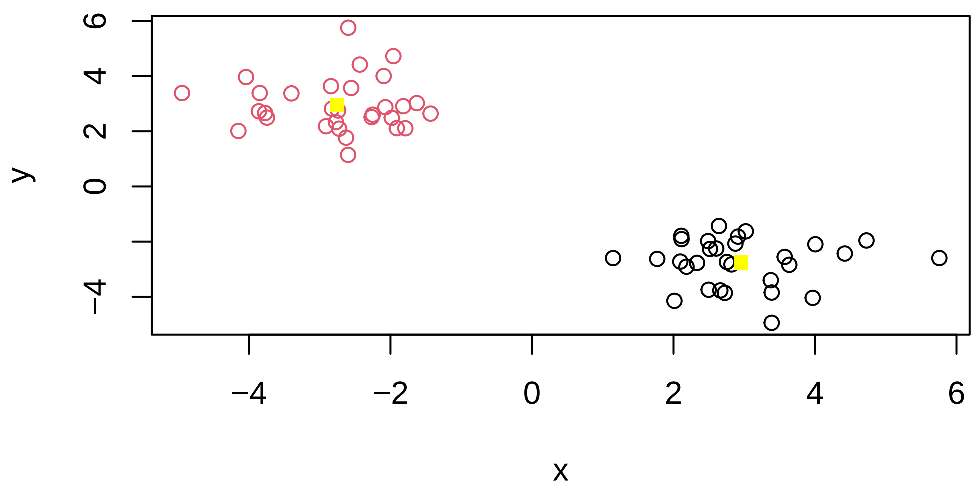
Q. What clusters do my data points reside in?

```
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 1 1
```

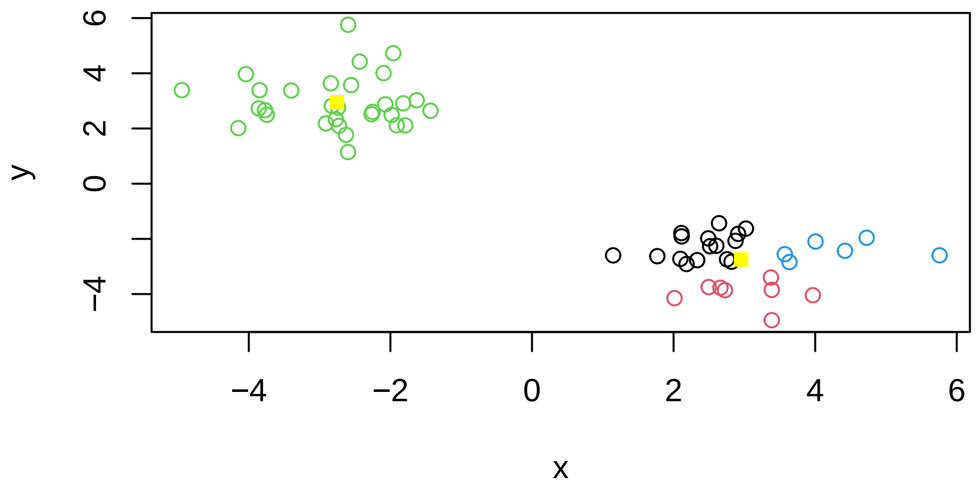
Q. Make a plot of our data coloredby cluster assignment - ie. make a result figure

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



Q Cluster with k-means into 4 clusters and plot your results above.

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

plot(x,col=k4$cluster)
points(k$centers, col = "yellow", pch=15)
```

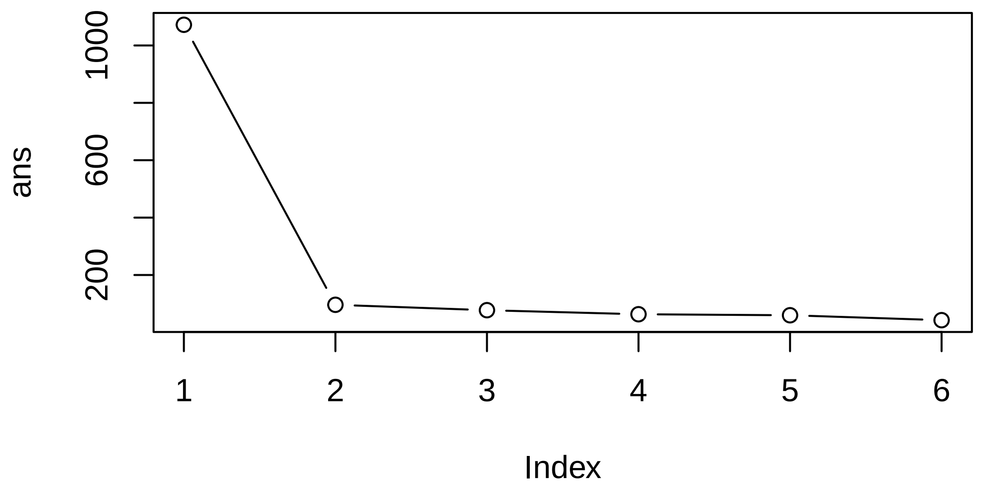


Q. Run kmeans with values center(i.e values of k) equal to 1 to 6.

```r
k1 <- kmeans(x, centers=1)$tot.withinss
k2 <- kmeans(x, centers=2)$tot.withinss
k3 <- kmeans(x, centers=3)$tot.withinss
k4 <- kmeans(x, centers=4)$tot.withinss
k5 <- kmeans(x, centers=5)$tot.withinss
k6 <- kmeans(x, centers=6)$tot.withinss

ans <- c(k1, k2, k3, k4, k5, k6)
```

Or use a for loop

```r
ans <- NULL
for(i in 1:6) {
  ans <- c(ans, kmeans(x, centers=i)$tot.withinss)
}
plot(ans, type = "b")
```



Hierarchical Clustering

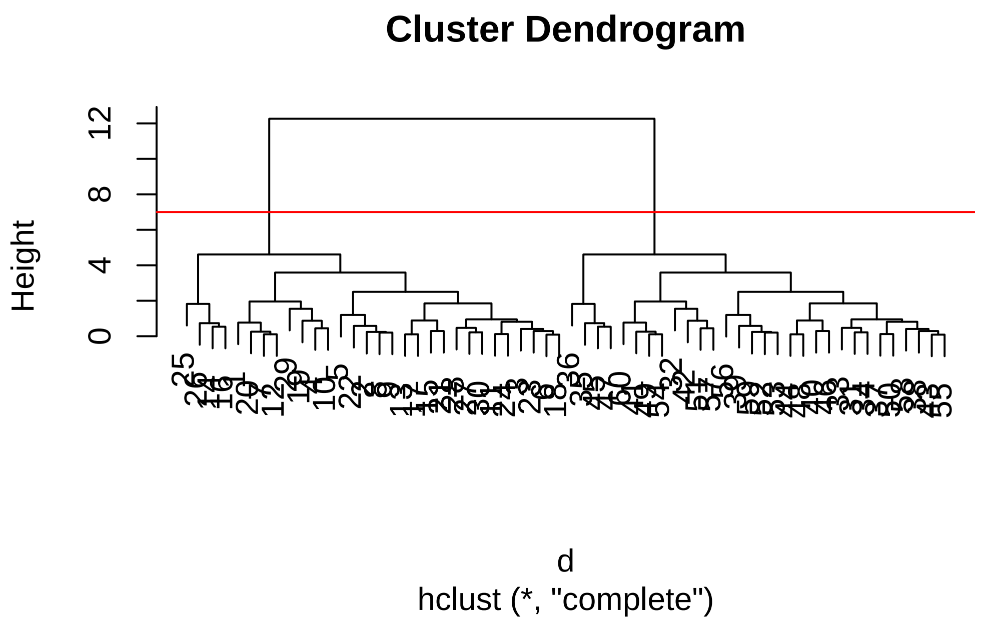
The main function in "base" R for this is called `hclust()`

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

```
Call:
hclust(d = d)

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

```r
#hclust(x)
```

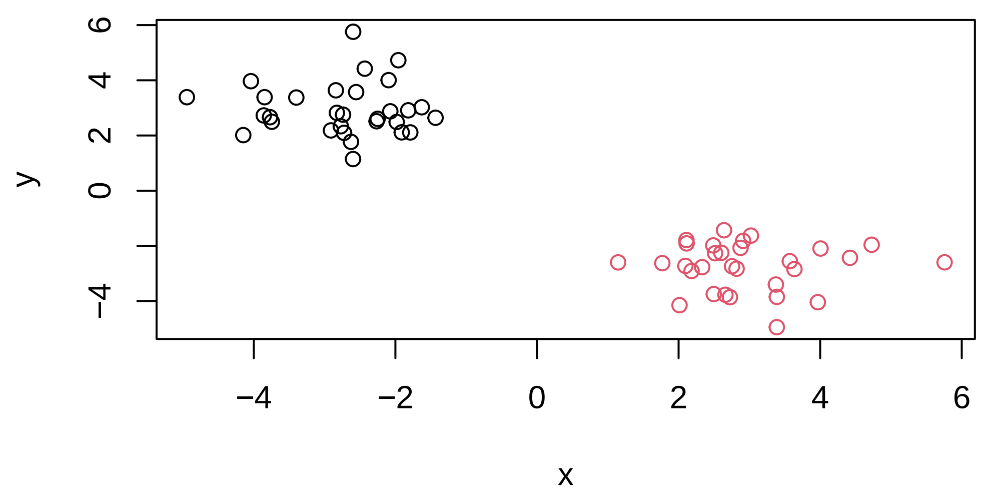
```r
plot(hc)
abline(h=7, col="red")
```



obtain clusters from `hclust` result object **hc** we “cut” the tree to yield different sub branches. For this we use the `cutree()` function

```
grps <- cutree(hc,h=7)
```

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



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

```r
dim(x)
```

```
[1] 17  5
```

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

```
dim(x)
```

```
[1] 17  4
```

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

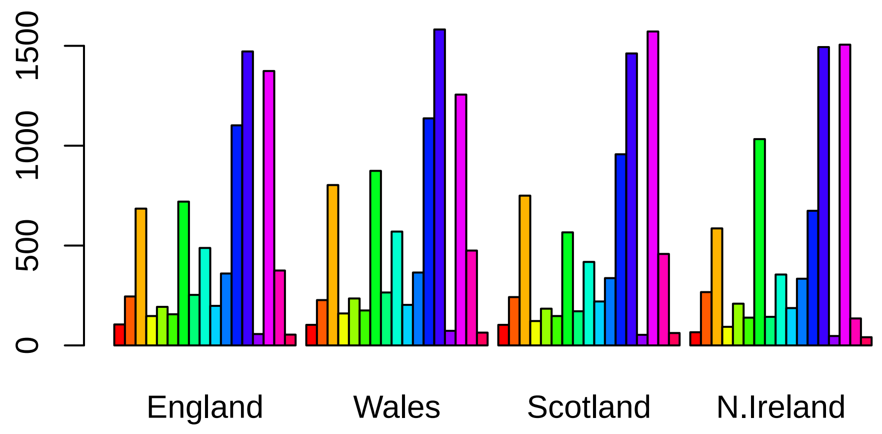
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars             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?

The second approach is more robust. It reads the row names correctly

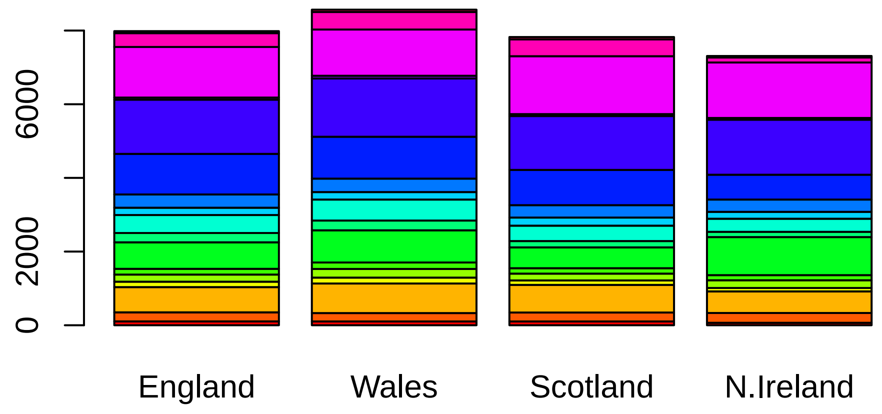
using base R

```
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),col=rainbow(nrow(x)))
```



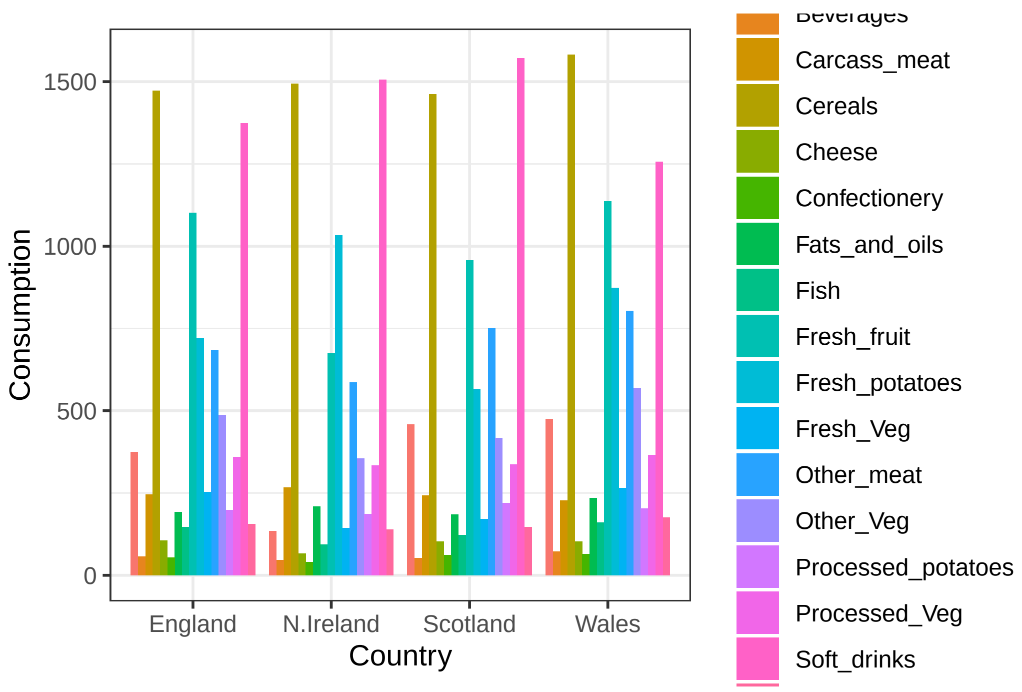
Removing the beside resutled in the plot.

> Q4: Changing what optional argument in the above ggplot() code results in a stacked barplot figure?

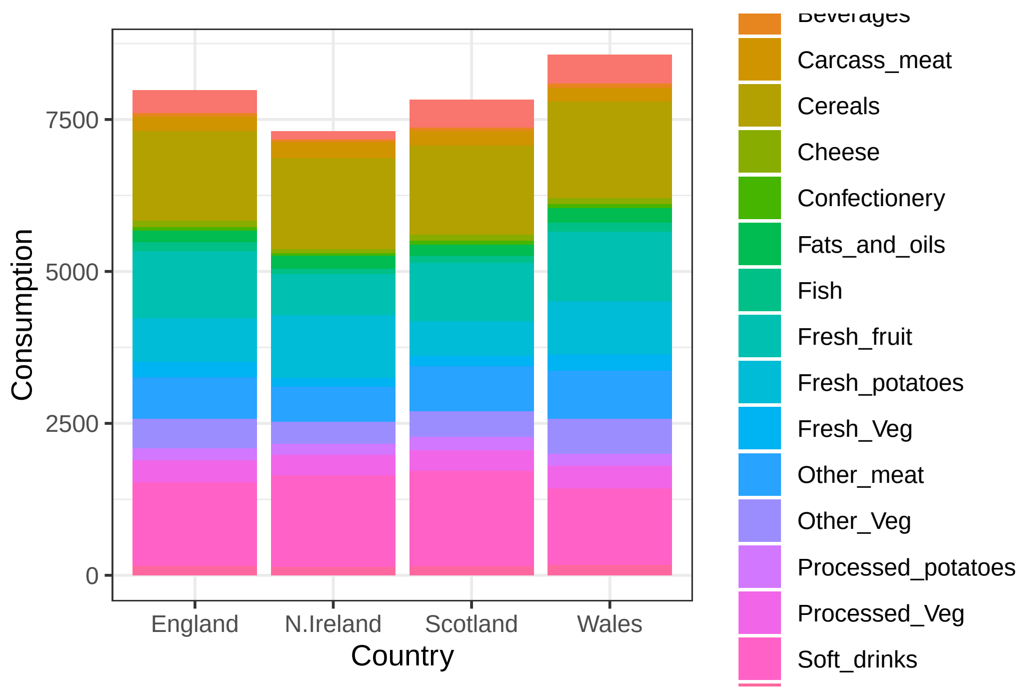
```r
library(tidyr)

# Convert data to long format for ggplot with `pivot_longer()`
x_long <- x |>
          tibble::rownames_to_column("Food") |>
          pivot_longer(cols = -Food,
                       names_to = "Country",
                       values_to = "Consumption")

dim(x_long)
```

```
[1] 68  3
```

```r
library(ggplot2)

ggplot(x_long) +
  aes(x = Country, y = Consumption, fill = Food) +
  geom_col(position = "dodge") +
  theme_bw()
```



```r
ggplot(x_long) +
  aes(x = Country, y = Consumption, fill = Food) +
  geom_col() +
  theme_bw()
```

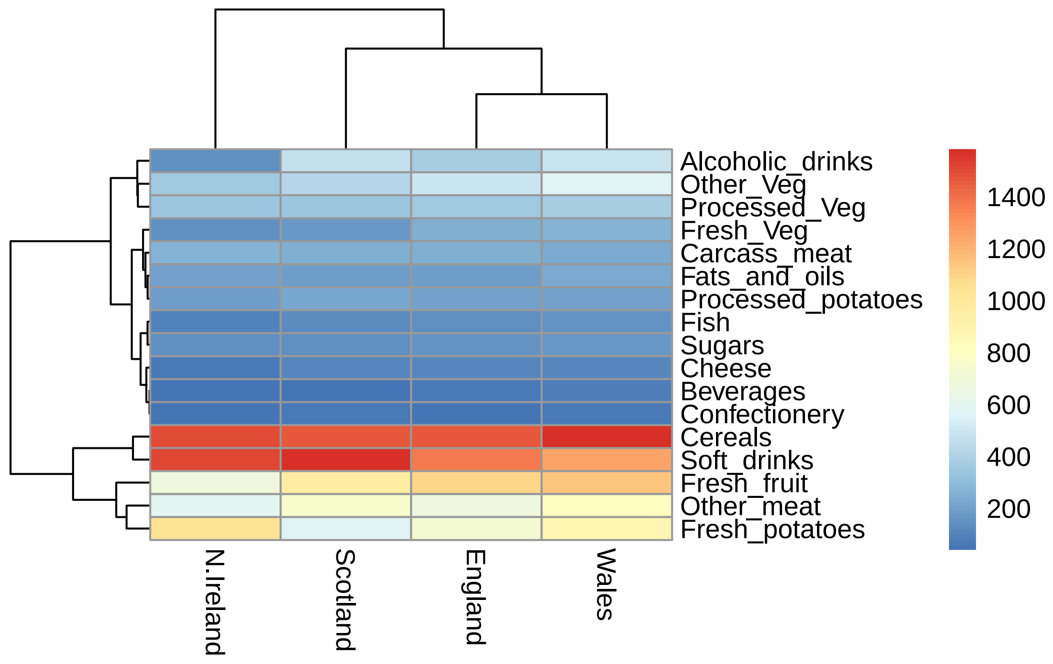


Removing the position argument made the plot.

> Q5: We can use the pairs() function to generate all pairwise plots for our countries. 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?

It is difficult to interpret the plot because the axes are not clear to me. It is taking the four countries we are examining and putting them in a display. They plot each country against another country for comparison. Each dot corresponds to the foods we are looking at. Foods along the diagonal mean there are similar quantities of that food consumed in those 2 countries.

```r
library(pheatmap)

pheatmap( as.matrix(x) )
```



> Q6. Based on the pairs and heatmap figures, which countries cluster together and what does this suggest about their food consumption patterns? Can you easily tell what the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

Wales and England look the most similar in their food consumption. It is difficult to read this plot.

PCA to the rescue

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

As we want to do PCA on the food data for the different countires we will want the foods in the columns (taking the transpose of the data)

```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
```

Our result object is called `pca` and it has a. `$x` component that we will look at first

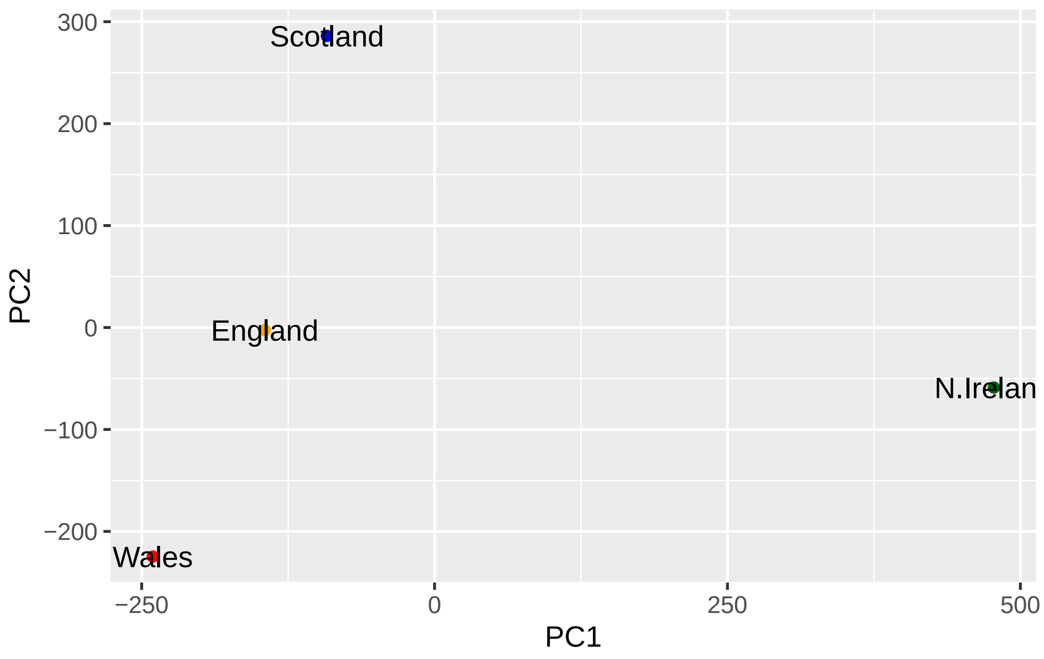
```
pca$x
```

```
                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
```

```
pca$x
```

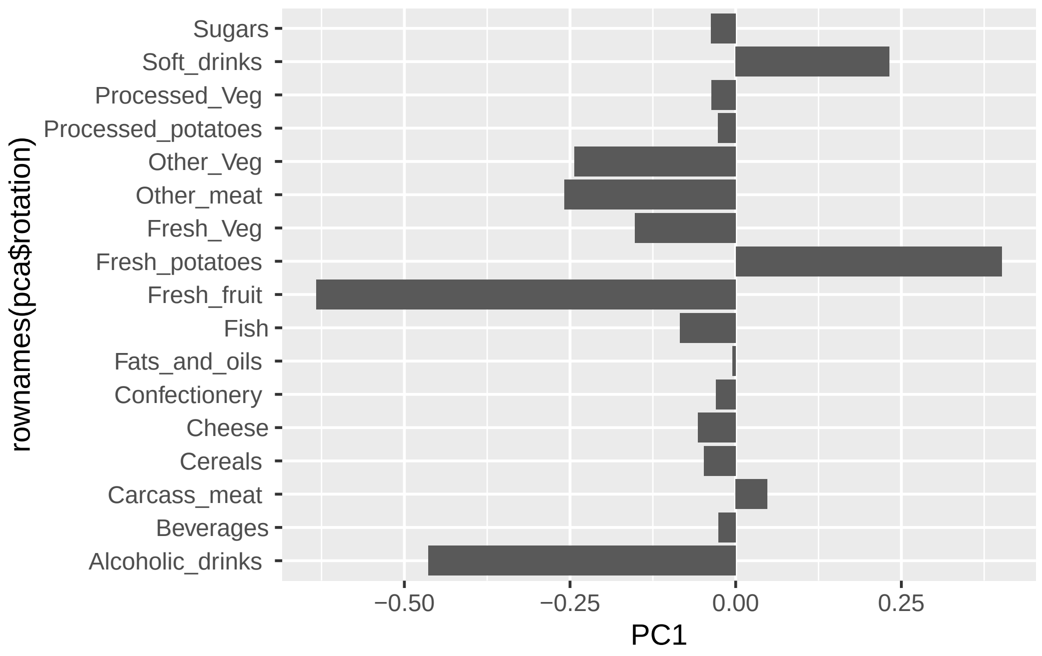
```
                PC1         PC2        PC3           PC4
England  -144.99315   -2.532999 105.768945 -9.152022e-15
Wales    -240.52915 -224.646925 -56.475555  5.560040e-13
Scotland  -91.86934  286.081786 -44.415495 -6.638419e-13
N.Ireland 477.39164  -58.901862  -4.877895  1.329771e-13
```

```
cols <- c("orange", "red", "blue", "darkgreen")
ggplot(pca$x) +
  aes(PC1, PC2, label=rownames(pca$x)) +
  geom_point(col = cols) +
  geom_text()
```



Another major result out of PCA is the so called "variable loadings" or `$rotation` that tells us how the original variablees (foods) contribute to PCs (the new axis), 0 is the mean the right and left side correspond to ggplot(pca$x)

```r
ggplot(pca$rotation) +
  aes(PC1, rownames(pca$rotation)) +
  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
rownames(pca$rotation)
PC1