class 07: Machine learning 1

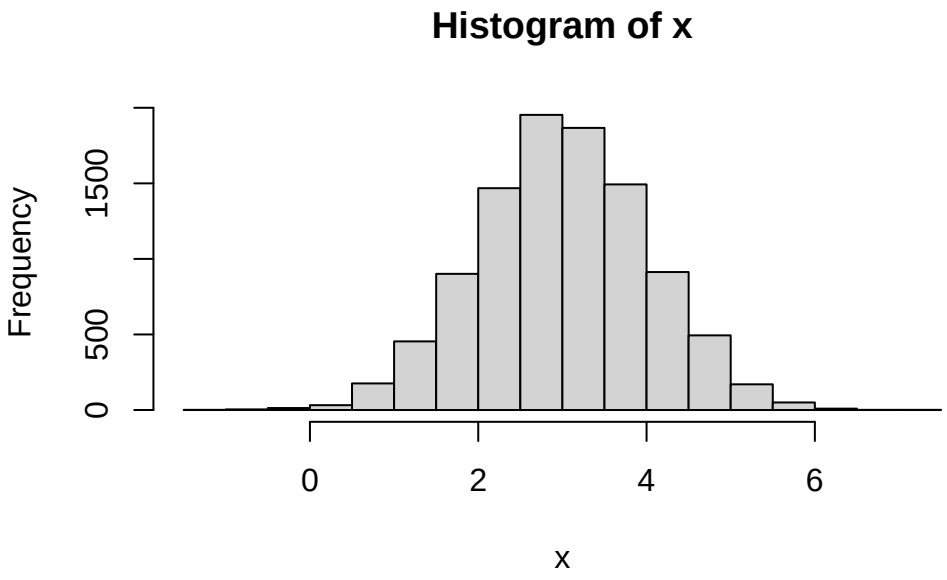
Trinity Lee A16639698

#clustering

We will start today's lab with clustering methods, in particular so-called K-means. The main function for this is in R is `kmeans()`.

Lets try it on some made up data where we know what the answer should be.

```
x<-rnorm(10000,mean=3)
hist(x)
```



60 points We can pass the function to `plot()`

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

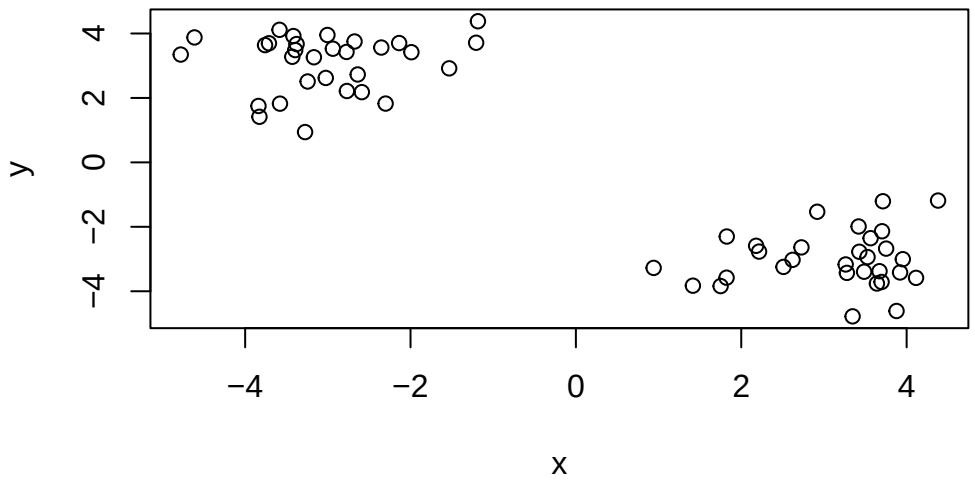
```
              x          y
 [1,]  3.7521348 -2.6768698
 [2,]  3.5274651 -2.9392110
 [3,]  0.9402528 -3.2753738
 [4,]  2.5106483 -3.2444738
 [5,]  1.4151658 -3.8269727
 [6,]  3.7029144 -2.1374934
 [7,]  2.1798670 -2.5891148
 [8,]  3.9539282 -3.0051297
 [9,]  1.8223841 -3.5788155
[10,]  2.6204334 -3.0250348
[11,]  3.4278350 -2.7739875
[12,]  3.4191140 -1.9893891
[13,]  2.7277342 -2.6387420
[14,]  4.3803821 -1.1848370
[15,]  4.1140120 -3.5844066
[16,]  3.2775672 -3.4300224
[17,]  1.7506709 -3.8391432
[18,]  1.8248036 -2.3012990
[19,]  3.4856530 -3.3942753
[20,]  3.5644250 -2.3532672
[21,]  3.8778454 -4.6126922
[22,]  2.2151803 -2.7695832
[23,]  3.9206566 -3.4166090
[24,]  2.9188138 -1.5324979
[25,]  3.6968538 -3.7123009
[26,]  3.2628197 -3.1684774
[27,]  3.6405467 -3.7592933
[28,]  3.6720885 -3.3779871
[29,]  3.3465937 -4.7791904
[30,]  3.7131531 -1.2063464
[31,] -1.2063464  3.7131531
[32,] -4.7791904  3.3465937
[33,] -3.3779871  3.6720885
[34,] -3.7592933  3.6405467
[35,] -3.1684774  3.2628197
[36,] -3.7123009  3.6968538
[37,] -1.5324979  2.9188138
```

```
[38,] -3.4166090  3.9206566
[39,] -2.7695832  2.2151803
[40,] -4.6126922  3.8778454
[41,] -2.3532672  3.5644250
[42,] -3.3942753  3.4856530
[43,] -2.3012990  1.8248036
[44,] -3.8391432  1.7506709
[45,] -3.4300224  3.2775672
[46,] -3.5844066  4.1140120
[47,] -1.1848370  4.3803821
[48,] -2.6387420  2.7277342
[49,] -1.9893891  3.4191140
[50,] -2.7739875  3.4278350
[51,] -3.0250348  2.6204334
[52,] -3.5788155  1.8223841
[53,] -3.0051297  3.9539282
[54,] -2.5891148  2.1798670
[55,] -2.1374934  3.7029144
[56,] -3.8269727  1.4151658
[57,] -3.2444738  2.5106483
[58,] -3.2753738  0.9402528
[59,] -2.9392110  3.5274651
[60,] -2.6768698  3.7521348
```

```
head(x)
```

```
            x         y
[1,] 3.7521348 -2.676870
[2,] 3.5274651 -2.939211
[3,] 0.9402528 -3.275374
[4,] 2.5106483 -3.244474
[5,] 1.4151658 -3.826973
[6,] 3.7029144 -2.137493
```

```
plot(x)
```




```r
k<-kmeans(x,centers=2,nstart=20)
k
```

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

Cluster means:
          x         y
1  3.088731 -3.004095
2 -3.004095  3.088731

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 2 2

Within cluster sum of squares by cluster:
[1] 44.23246 44.23246
 (between_SS / total_SS =  92.6 %)

Available components:

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

Q1. How many points are in each cluster?

```
k$size
```

```
[1] 30 30
```

Q2. CLuster membership?

```
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 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
```

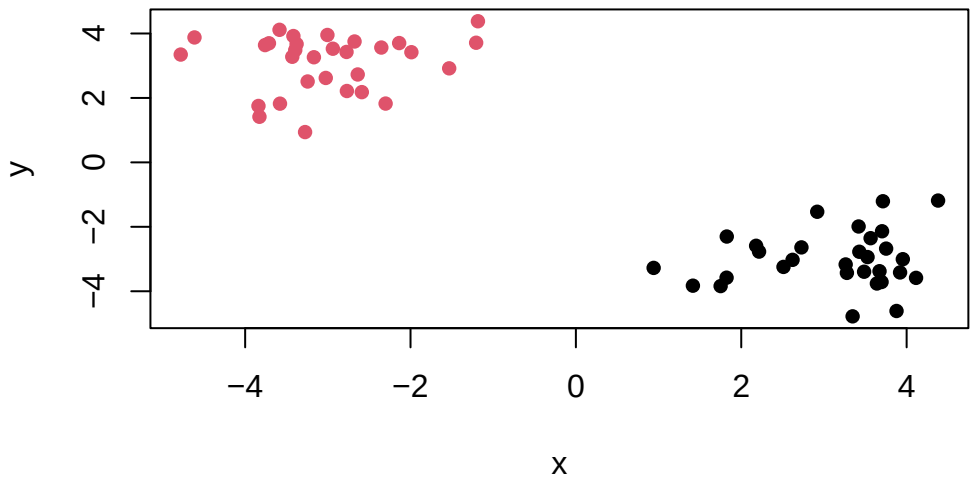
Q3. CLuster centers?

```
k$centers
```

```
          x         y
1  3.088731 -3.004095
2 -3.004095  3.088731
```

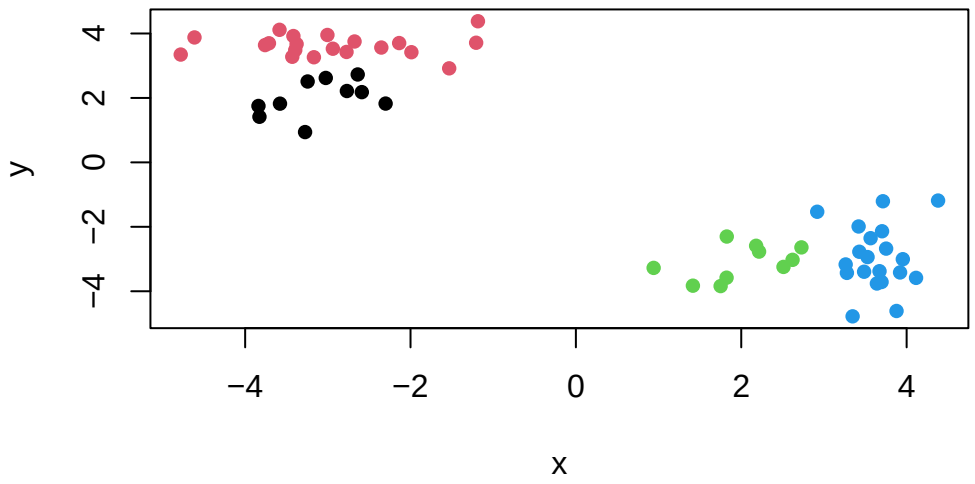
Q4. plot my clustering results.

```
plot(x, col=k$cluster,pch=16)
```



Q5. cluster the data again with `kmeans()` into 4 groups and plot the results.

```
k4<-kmeans(x,centers=4,nstart=20)
plot(x, col=k4$cluster,pch=16)
```



k-means is popular mostly because it is fast and relatively straightforward to run and understand. It has a big limitation in that you need to tell it how many groups (k, or centers) you want.

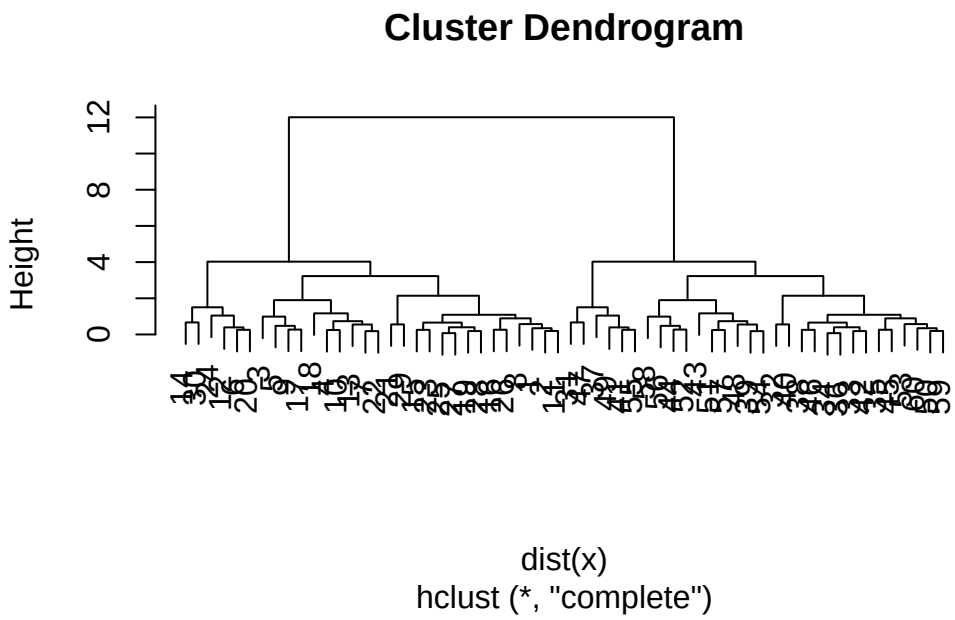
#Hierarchial clustering the main fucntion in base R is called `hclust()`. You have to pass it in a "distance matrix" nto just your input data.

You can generate a distance matrix with the `dist()` fucntion.

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

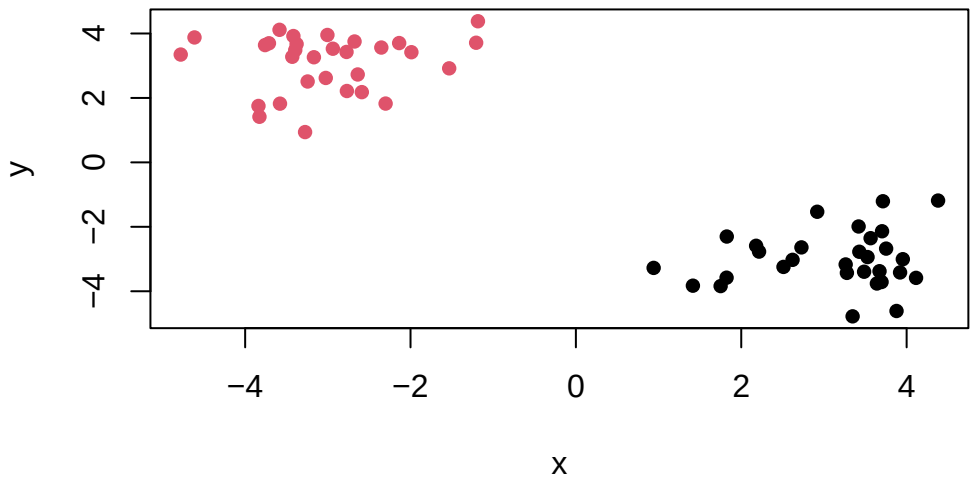
```
Call:
hclust(d = dist(x))

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

```
plot(hc)
```



To find clusters (cluster membership vector) from a `hclust()` reuslt we can "cut" the tree at a certain height that we like.

```
grps<-cutree(hc,h=8)
plot(x, col=grps,pch=16)
abline(h=8,col="red")
```




```
table(grps)
```

```
grps
 1  2
30 30
```

#Principal component analysis

##PCA of UK food data Read data showing the comsumption of food in grams (per person, per week) of 17 different types of food-stuff measured and averaged in the four countries of the United Kingdom in 1997.

```
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
```

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

```
ans<-dim(x)
ans
```

```
[1] 17  5
```

Preview the first 6 rows

```
head(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
```

The row names seem to be incorrect with X as the first one. Let's try to fix this and get the correct amount of rows.

```
# Note how the minus indexing works
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
```

```
ans1<-dim(x)
ans1
```

```
[1] 17  4
```

To fix the issue we have when rerunning the data file again we will use `read.csv()`

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
head(x)
```

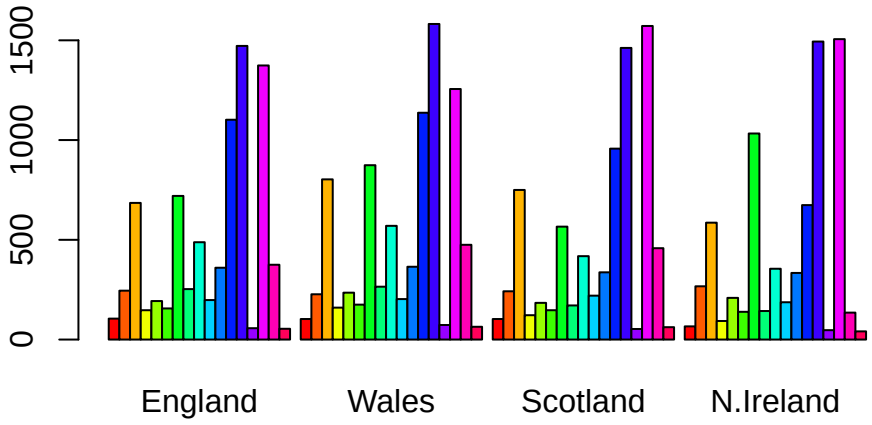
```
              England Wales Scotland N.Ireland
Cheese            105   103      103        66
Carcass_meat      245   227      242       267
Other_meat        685   803      750       586
Fish              147   160      122        93
Fats_and_oils     193   235      184       209
Sugars        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

I prefer using the row.names argument set to read.csv() due to it fixing the issue while allowing us to rerun the data without messing it up. Therefore it is more robust and less prone to error.

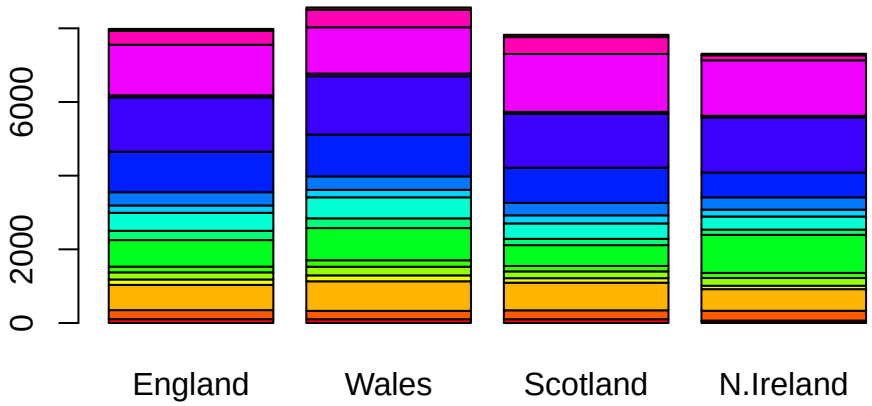
#Spotting major differences and trends A cursory glance over the numbers in this table does not reveal much of anything. Indeed in general it is difficult to extract meaning in regard to major differences and trends from any given array of numbers. Generating regular bar-plots and various pairwise plots does not help too much either

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



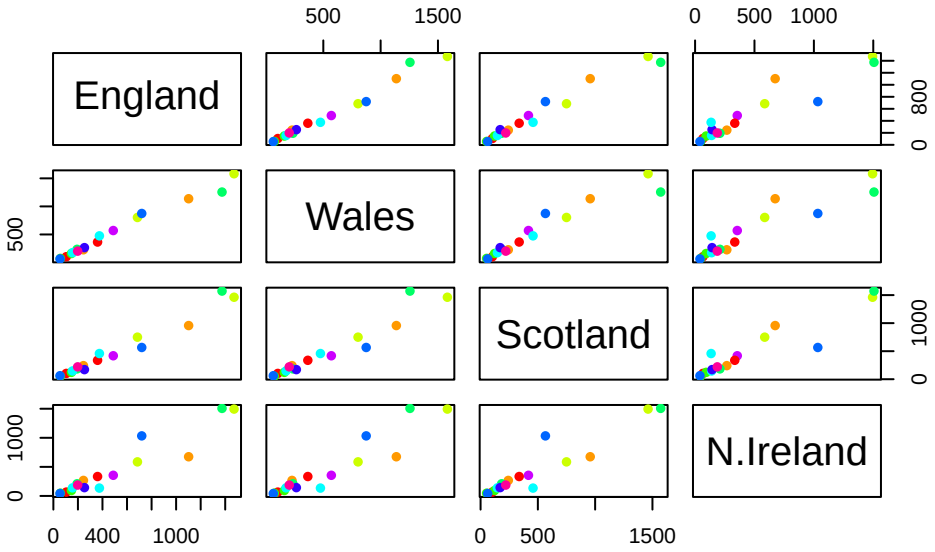
Q3: Changing what optional argument in the above barplot() function results in the following plot?

```
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)
```



The figure shows graphs of the comparison of the amounts of each food consumed by two countries. If a point lies on the diagonal for a graph it means that exactly the same amount or similar amounts of that food are consumed in the two countires being compared. Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

There seems to be two major food categories that are different or deviate from the diagonal between N.IReland and other UK countries but we cannot determine which ones exactly from just looking at the graph.

##PCA

PCA can help us make sense of these types of datasets. Let's see how it works.

The main function in "base R" is called `prcomp()`. In this case we want to first take the transpose of our input `x` so the columns are the food types and the countries are the rows.

```
head(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
```

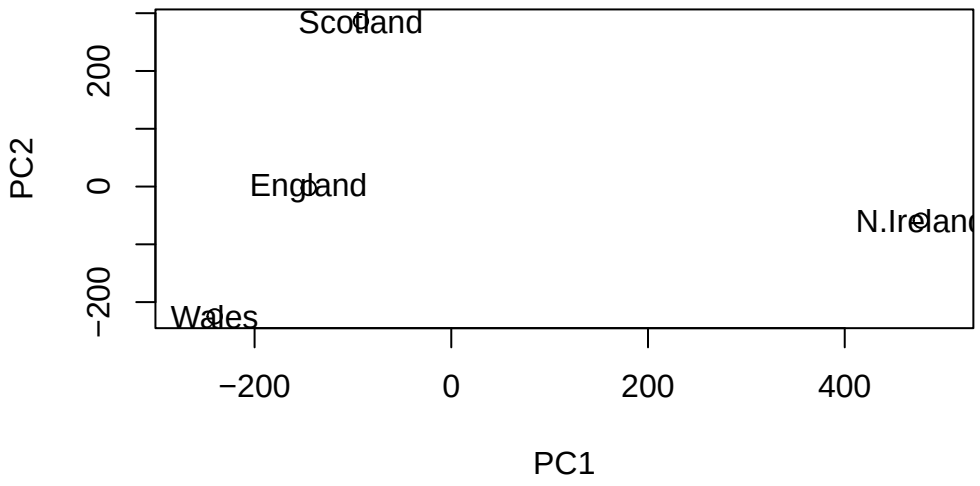
```
           Fresh_potatoes  Fresh_Veg  Other_Veg  Processed_potatoes
England               720        253        488                 198
Wales                 874        265        570                 203
Scotland              566        171        418                 220
N.Ireland            1033        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
```

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

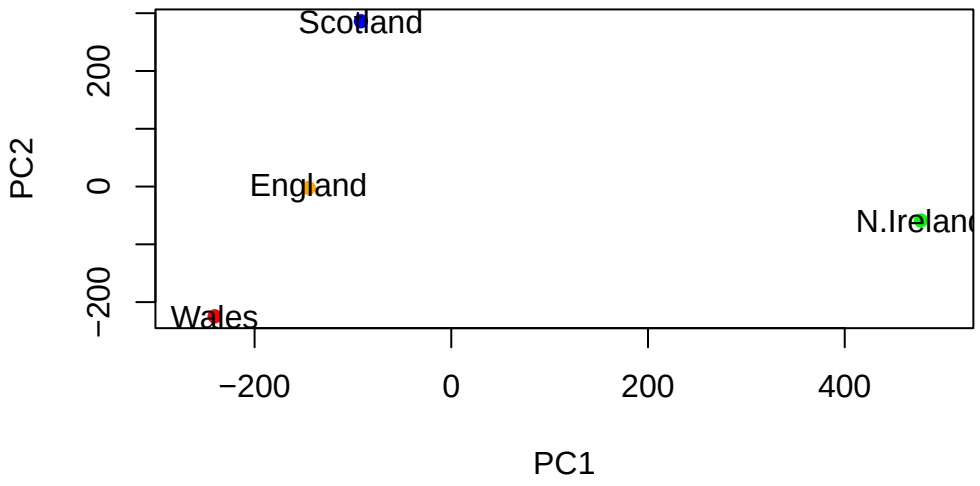
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```



Q8. Customize your plot so that the colors of the country names match the colors in our UK and Ireland map and table at start of this document.

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



The “loadings” tell us how much of the original variables -foods contribute to the new variables -PCs

```
head(pca$rotation)
```

```
                    PC1         PC2         PC3          PC4
Cheese       -0.056955380  0.01601285  0.02394295 -0.694538519
Carcass_meat  0.047927628  0.01391582  0.06367111  0.489884628
Other_meat   -0.258916658 -0.01533114 -0.55384854  0.279023718
Fish         -0.084414983 -0.05075495  0.03906481 -0.008483145
Fats_and_oils -0.005193623 -0.09538866 -0.12522257  0.076097502
Sugars       -0.037620983 -0.04302170 -0.03605745  0.034101334
```

we can use the square of pca$sdev , which stands for “standard deviation”, to calculate how much variation in the original data each PC accounts for.

```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

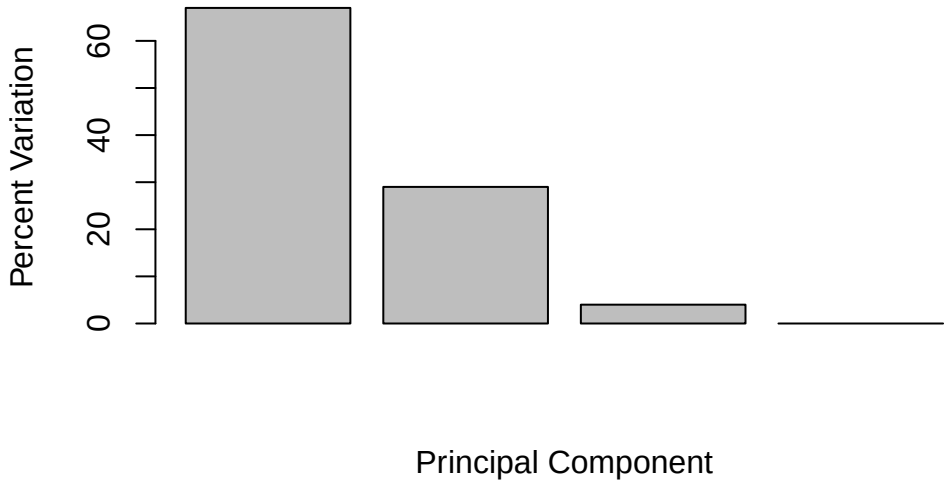
```
[1] 67 29  4  0
```

```
## or the second row here...
z <- summary(pca)
z$importance
```

```
                            PC1       PC2      PC3           PC4
Standard deviation     324.15019 212.74780 73.87622 3.175833e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

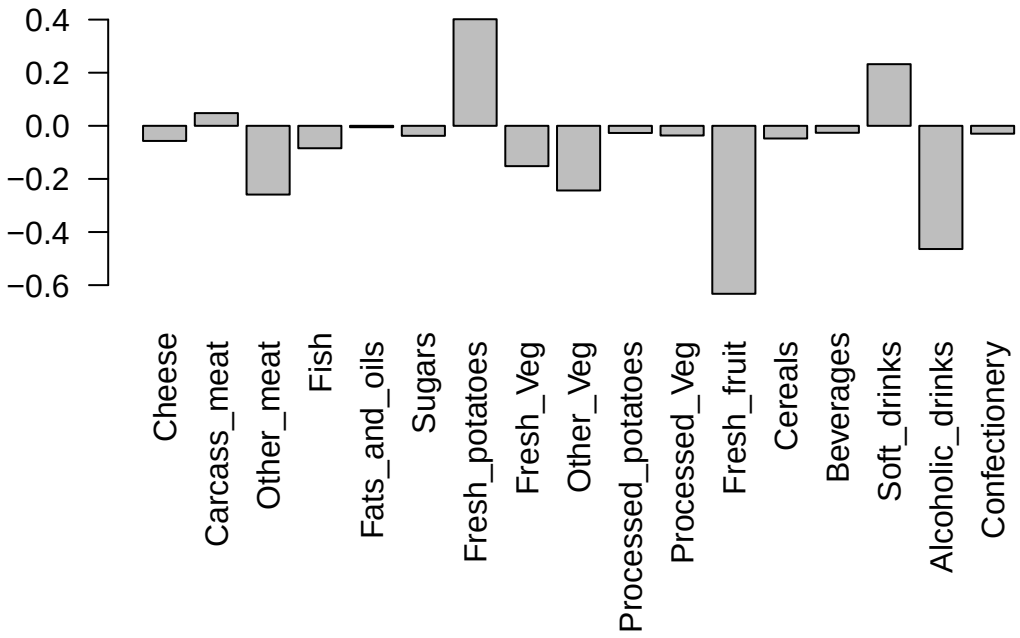
This information can be summarized in a plot of the variances (eigenvalues) with respect to the principal component number (eigenvector number), which is given below.

```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



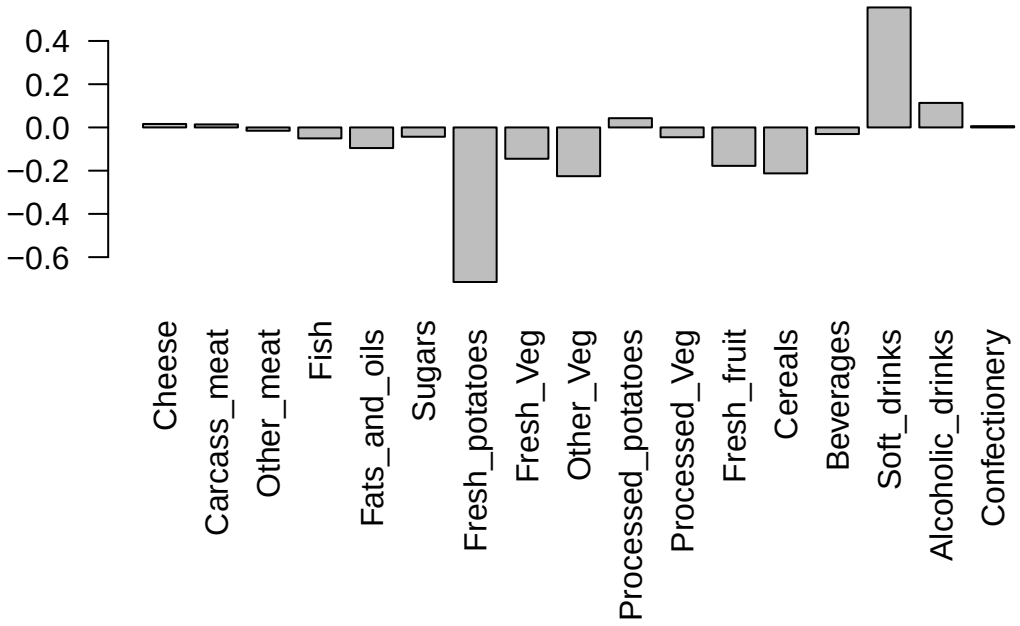
#Digging deeper We can also consider the influence of each of the original variables upon the principal components (typically known as loading scores). This information can be obtained from the `prcomp()` returned `$rotation` component. It can also be summarized with a call to `biplot()`

```
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



The two food groups are fresh potatoes and soft drinks. PC2 mainly tells us about the left over varience that PC1 does not cover.