Machine Learning 1

Anu Chaparala PID: A15484707

10/21/2021

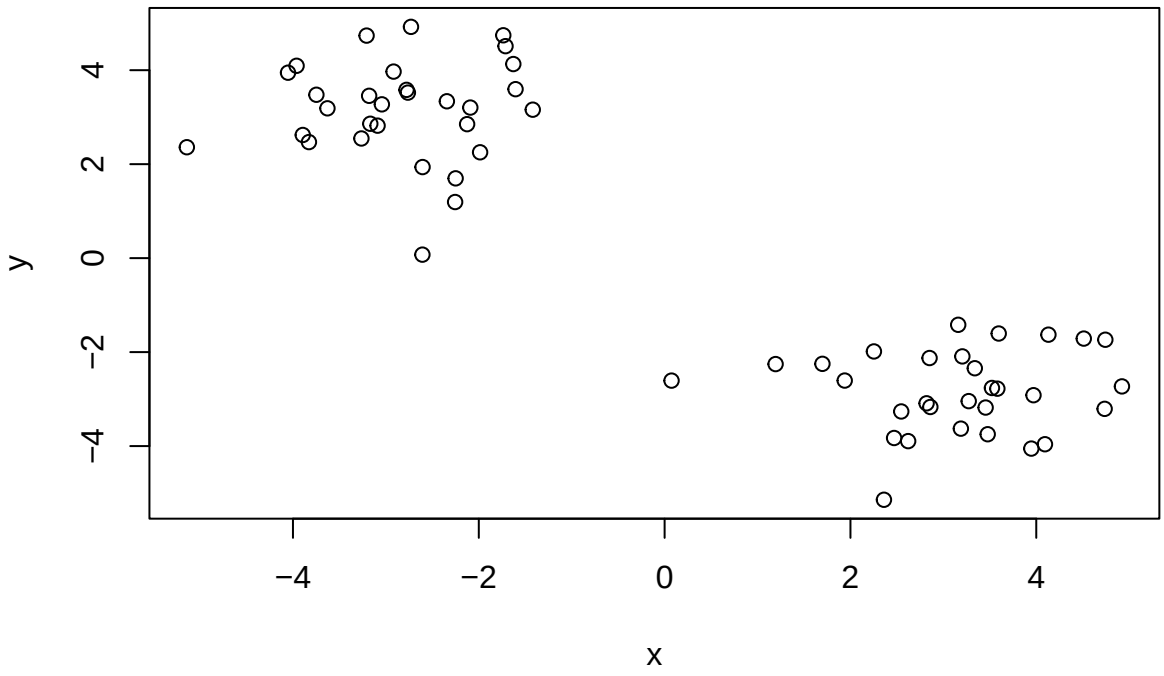
First up is clustering methods

#Kmeans clustering

The function in base R to do Kmeans clustering is called 'kmeans()'.

First make up some data where we know what the answer should be:

```
#rnorm creates random normal distributed data around a set center, in this case -3.
tmp <- c(rnorm(30, -3), rnorm(30, 3))
x <- cbind(x=tmp, y=rev(tmp))
plot(x)
```



Q. Can we use kmeans() to cluster this data setting k to 2 and nstart to 20?

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1  3.151290 -2.825216
## 2 -2.825216  3.151290
##
## 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 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] 56.23634 56.23634
##  (between_SS / total_SS =  90.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How many points are in each cluster?

```
#Answer: 30 points each
km$size
```

```
## [1] 30 30
```

Q. What 'component' of your result object details cluster assignment/membership?

```
#Answer: the cluster component
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 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
```

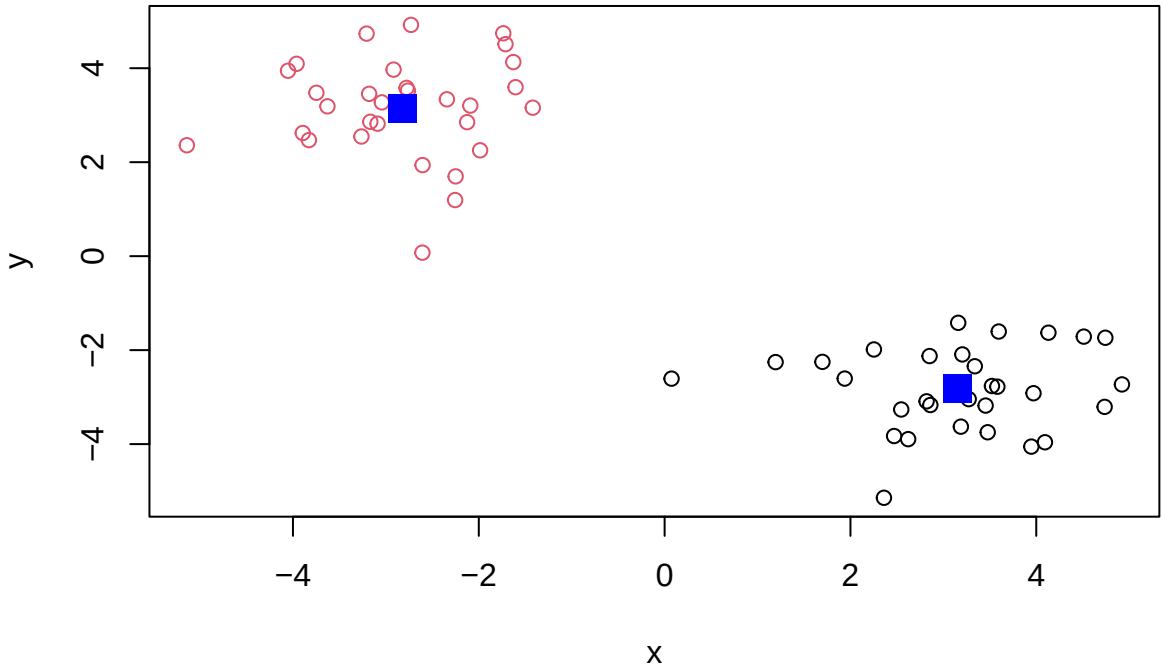
Q. What 'componenet of your result object details cluster center?

```
#Answer: the centers component
km$centers
```

```
##           x         y
## 1  3.151290 -2.825216
## 2 -2.825216  3.151290
```

Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points.

```
plot(x, col=km$cluster)
#points() function can help us highlight specific points, where col = color, pch = dot type, cex = poin
points(km$centers, col="blue", pch=15, cex=2)
```



hclust

Now let's move onto hclust() function, which stands for 'Hierarchal Cluster.' A big limitation with kmeans is that we have to tell it K (the number of clusters we want).

Analyze some data with hclust()

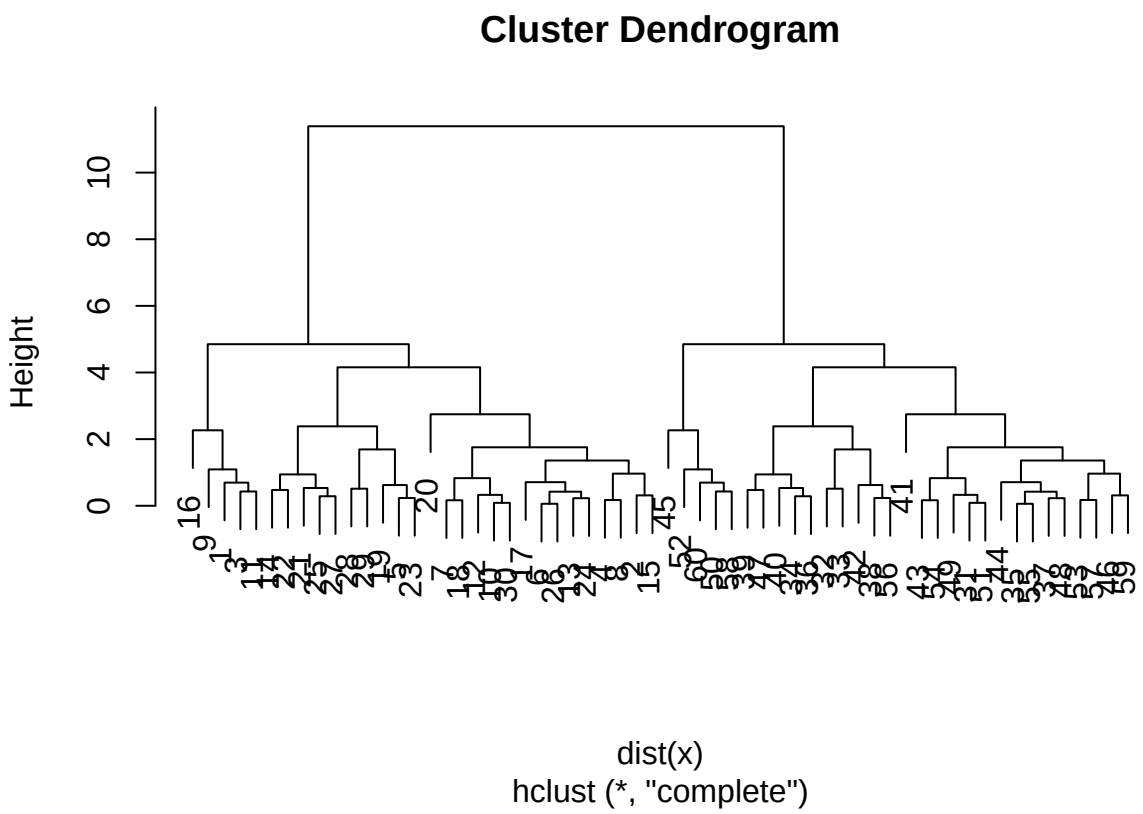
Demonstrate the use of dist(), hclust(), plot(), and cutree() functions to do clustering. Generate **dendrograms** and return cluster assignment/membership vector.

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

```
##
## Call:
## hclust(d = dist(x))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

There is a plot method for hclust result objects. Let's see it. (dendrogram)

```
plot(hc)
```



To get our cluster membership vector we have to do a wee bit more work. We have to "cut" the tree/dendrogram where we think it makes sense. For this we will use the 'cuttree()' function.

```
cutree(hc, h=6)
```

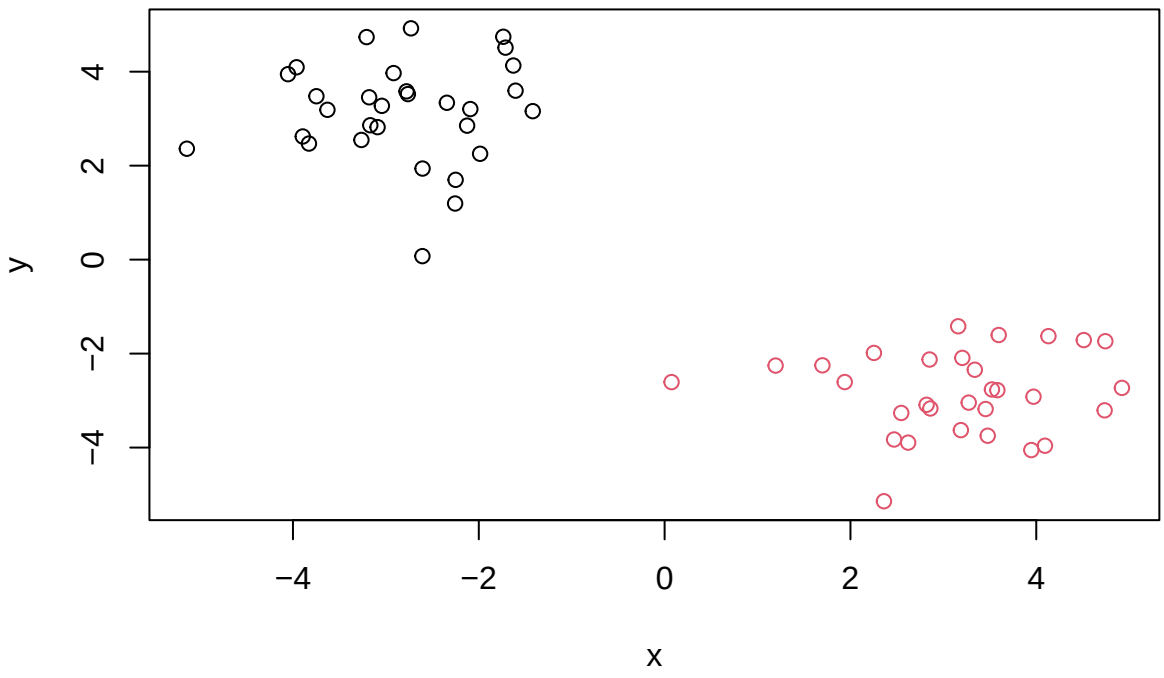
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

You can also call 'cutree()' setting k=the number of grps/clusters you want.

```
grps <- cutree(hc, k=2)
```

Make our results plot.

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




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

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

```
dim(x)
```

```
## [1] 17  5
```

```
#There are 17 rows and 5 columns.
#Note, it should be 17x4, not 17x5.
```

```
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
```

Preview first 6 rows.

```
#viewing first 6 rows, can also use View() or tail() functions.
#Notice row names are incorrectly set as onother column and not rownames.
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
```

One way to fix this is...

```
# 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
```

```
#Now there are only 4 columns, which is what we want.
```

However, this is not the best way to fix the dataset, since this overwrites x and will continue to subtract 1 column, each time this chunk is run.

```
#dimension check
dim(x)
```

```
## [1] 17  4
```

Here is a better way...

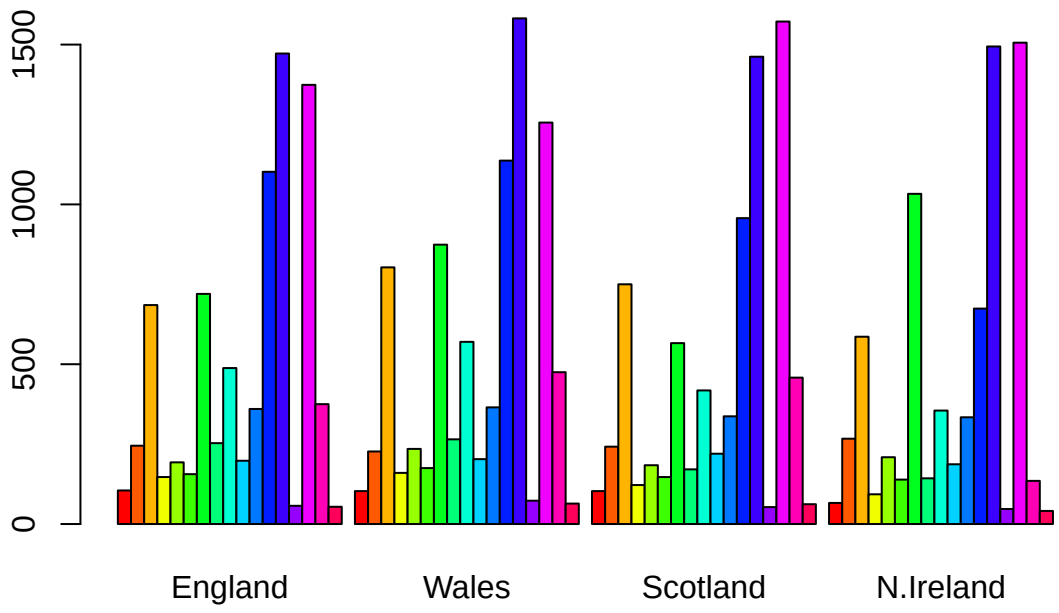
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
head(x)
```

```
##                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?

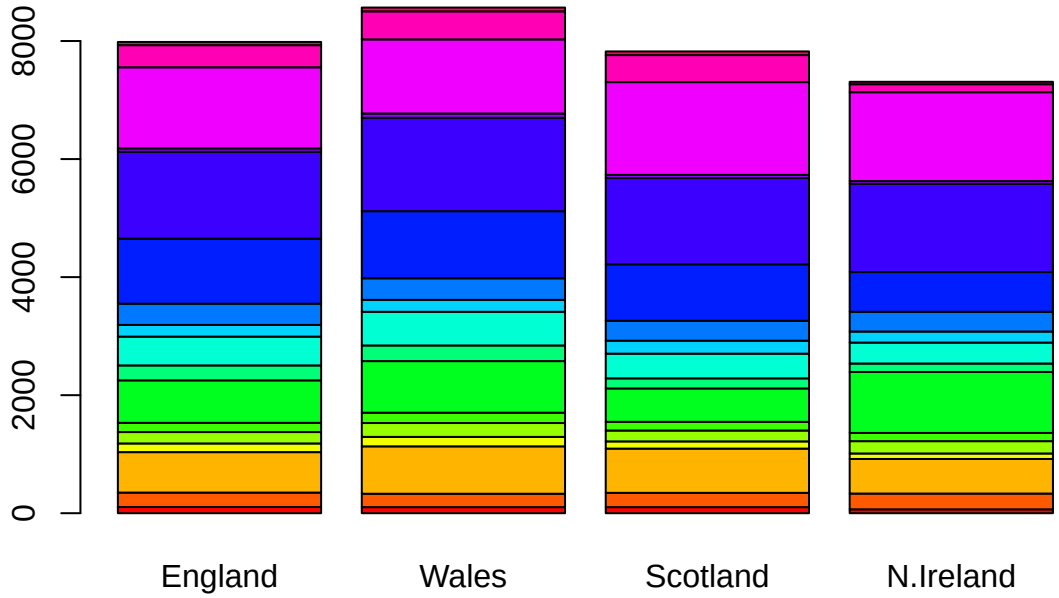
I prefer the second solution, which uses row.names argument in the initial reading of the CSV file. This way, we do not risk losing any data by repeatedly (multiple times) running this file over and over, like in the first solution.

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



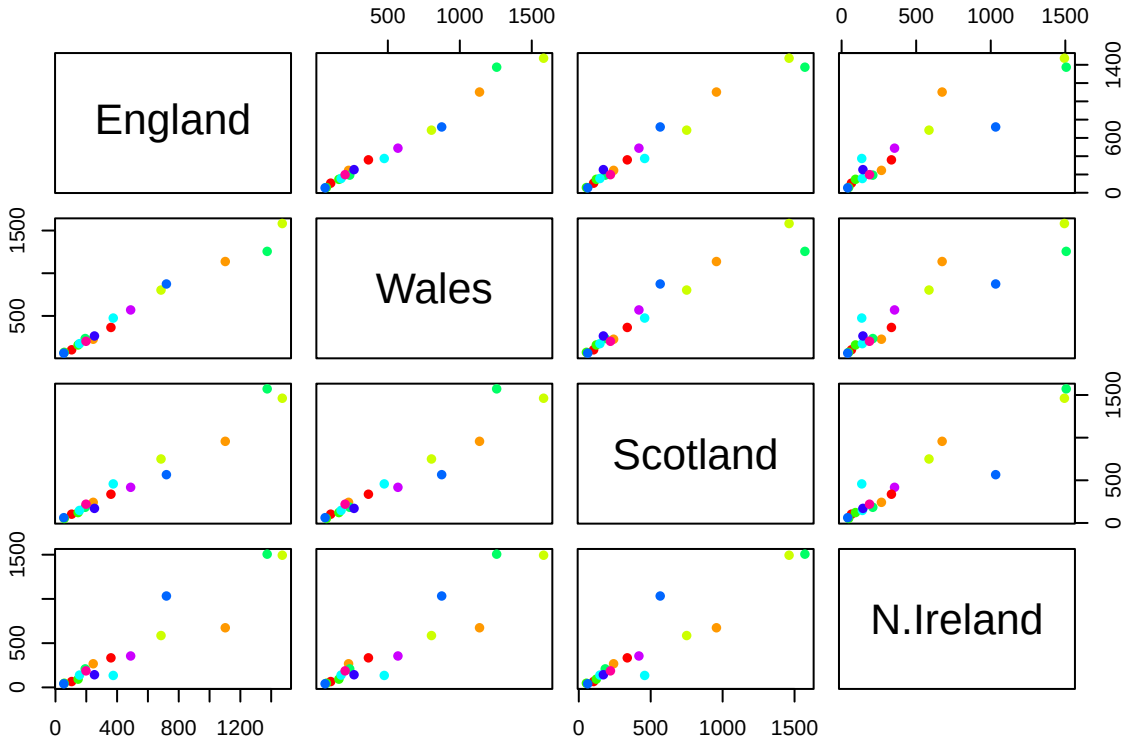
Q3: Changing what optional argument in the above barplot() function results in the following plot?

```
#setting the beside argument to False, enables the plot to stack the data sets on top of one another in
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)
```




```
#All of the plots show us all the permutations of the countries against one another. This is useful bec
```

Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

We can see that N. Ireland appears to have more differences with the other countries than the others do with each other (i.e. not everything is on the straight line in that last column and row of plots, implying some deviance from the others). However, it is very hard for us to get specific differences from just our pairwise plots.

PCA to the rescue!

The main function in base R for PCA is 'prcomp' This wan't the tranpose of our data using t() function.

```
# Use the prcomp() PCA function
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
```

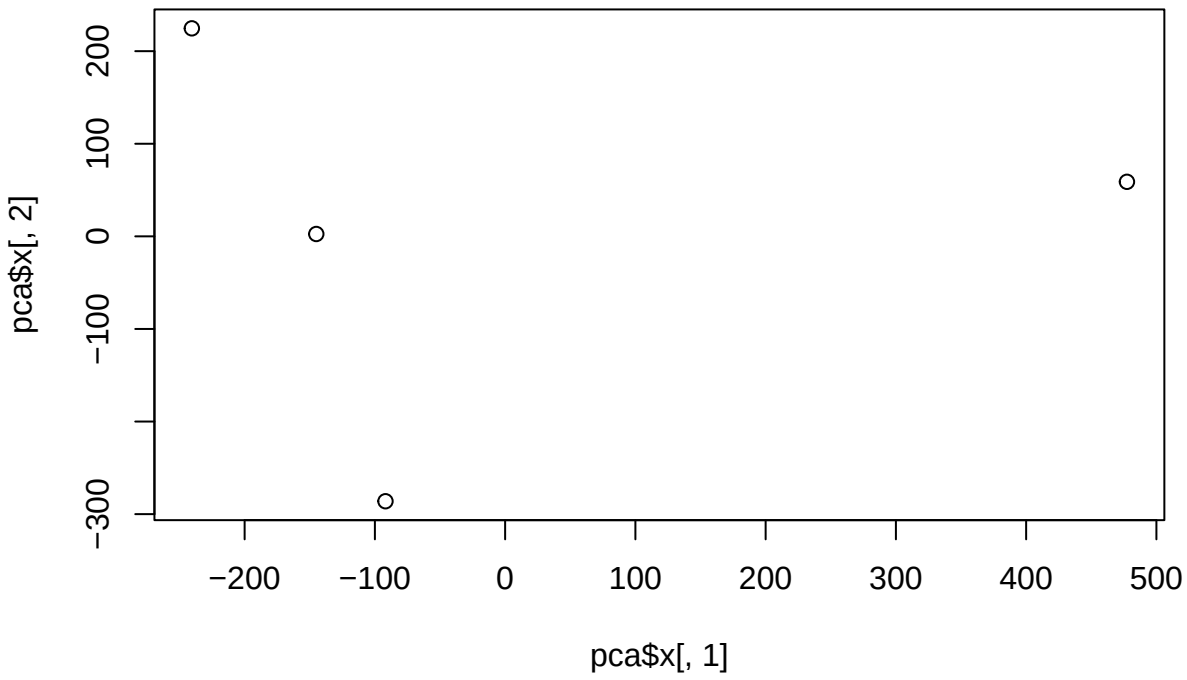
```
t(x)
```

```
##           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
```

```
attributes(pca)
```

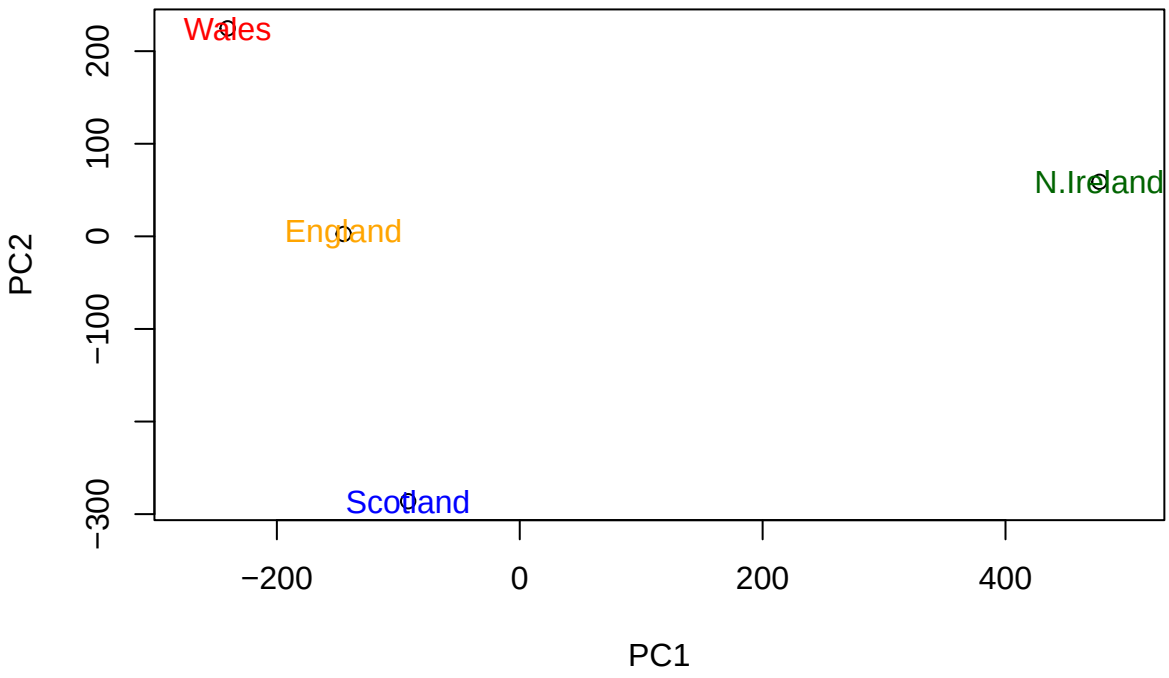
```
## $names
## [1] "sdev"     "rotation" "center"   "scale"    "x"
##
## $class
## [1] "prcomp"
```

```
plot(pca$x[,1], pca$x[,2])
```



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
#with edited color scheme
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
color <- c("orange", "red", "blue", "darkgreen")
text(pca$x[,1], pca$x[,2], colnames(x), col= color)
```



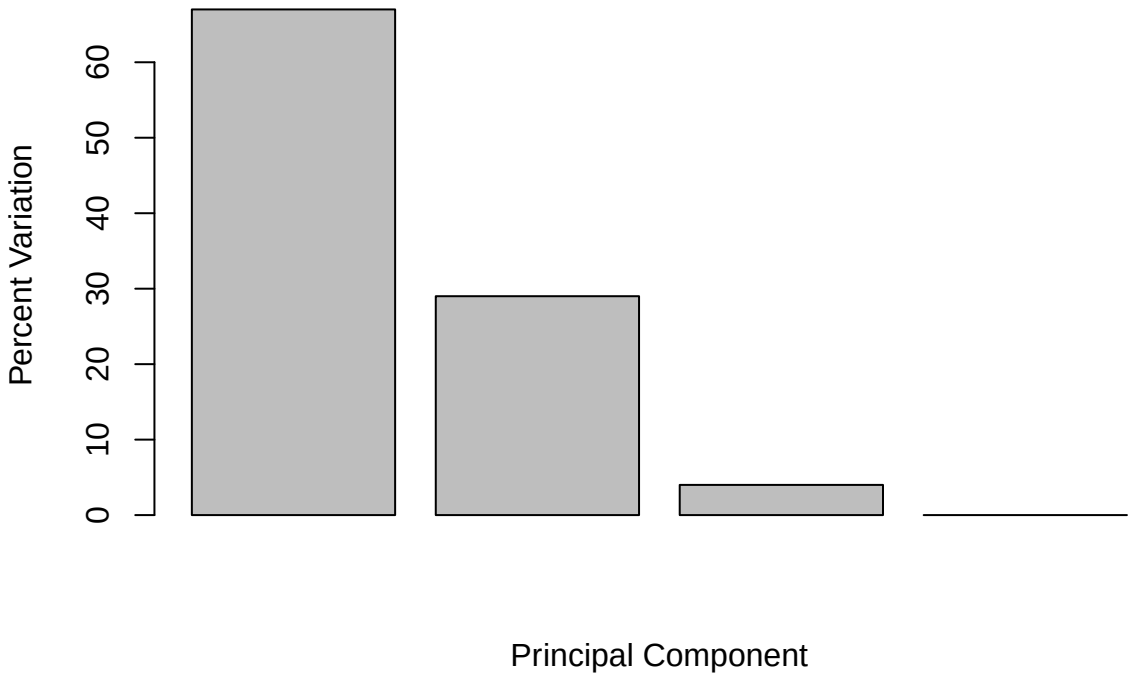

```
#calculates variation in original data that each PC accounts for.
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

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

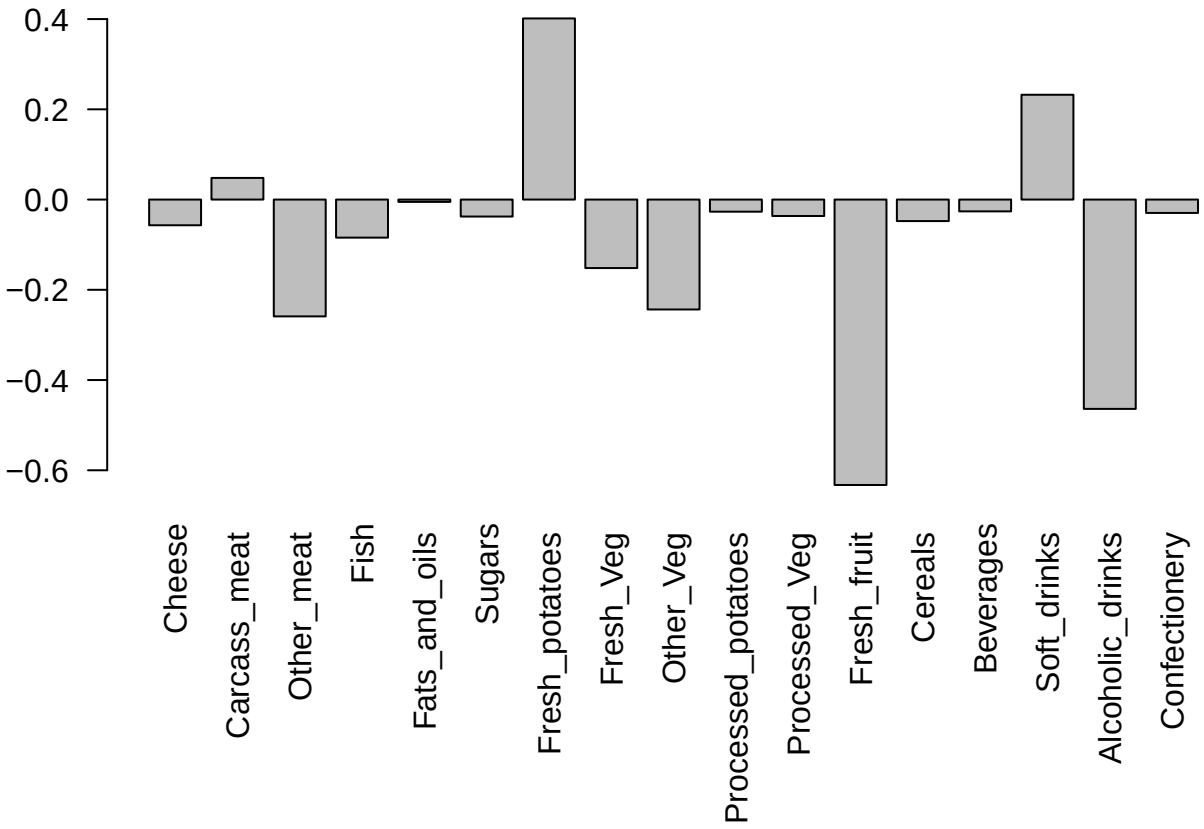
```
## or the second row here...
z <- summary(pca)
z$importance
```

```
##                             PC1       PC2      PC3          PC4
## Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
## Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
## Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

```
#summary plot of the variances (eigenvalues) with respect to the principal component number (eigenvecto
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```

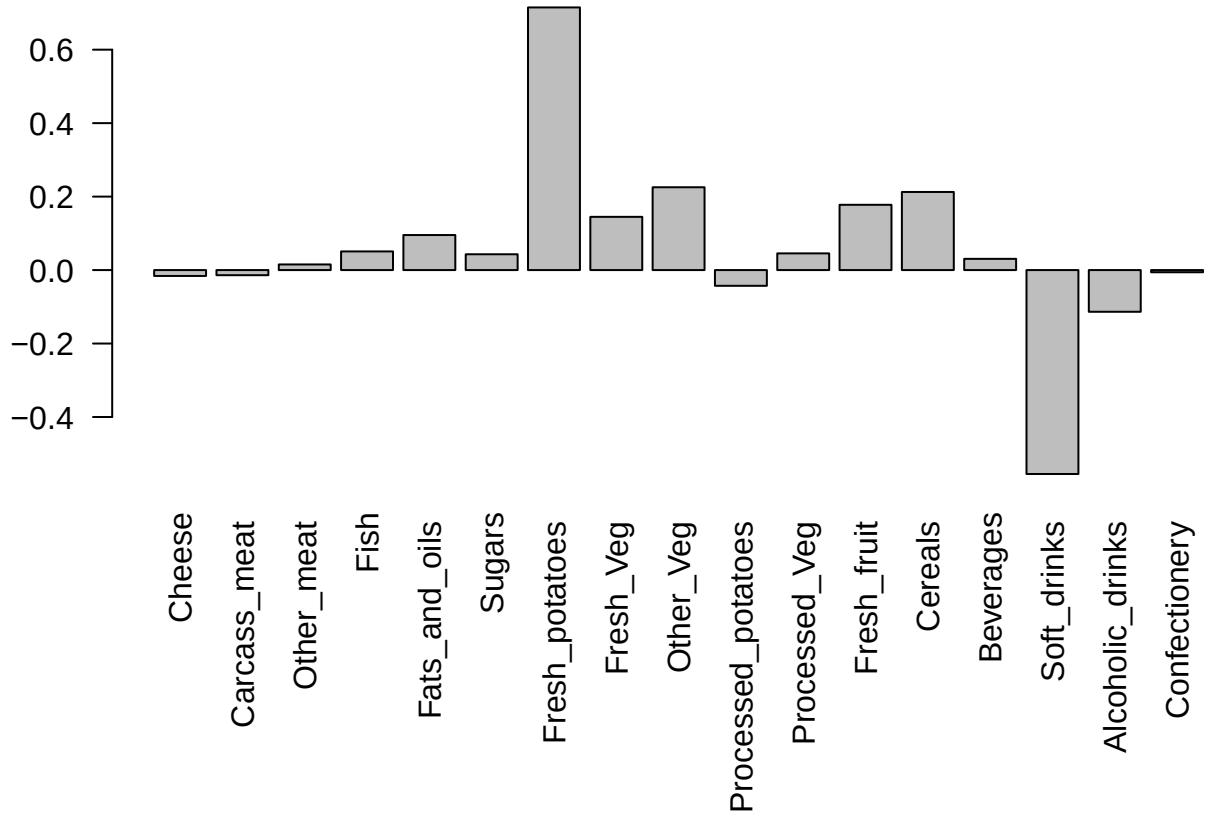



```
#Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



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 )
```




```
#PC2 tells us about the variation between the countries that PC1 did not show us. The two prominent foo
```

```
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <- read.csv(url2, row.names=1)
head(rna.data)
```

```
##        wt1 wt2  wt3  wt4 wt5 ko1 ko2 ko3 ko4 ko5
## gene1  439 458  408  429 420  90  88  86  90  93
## gene2  219 200  204  210 187 427 423 434 433 426
## gene3 1006 989 1030 1017 973 252 237 238 226 210
## gene4  783 792  829  856 760 849 856 835 885 894
## gene5  181 249  204  244 225 277 305 272 270 279
## gene6  460 502  491  491 493 612 594 577 618 638
```

Q10: How many genes and samples are in this data set?

```
dim(rna.data)
```

```
## [1] 100  10
```

```
#there are 100 genes and 10 samples.
```