Class 07: Machine Learning 1

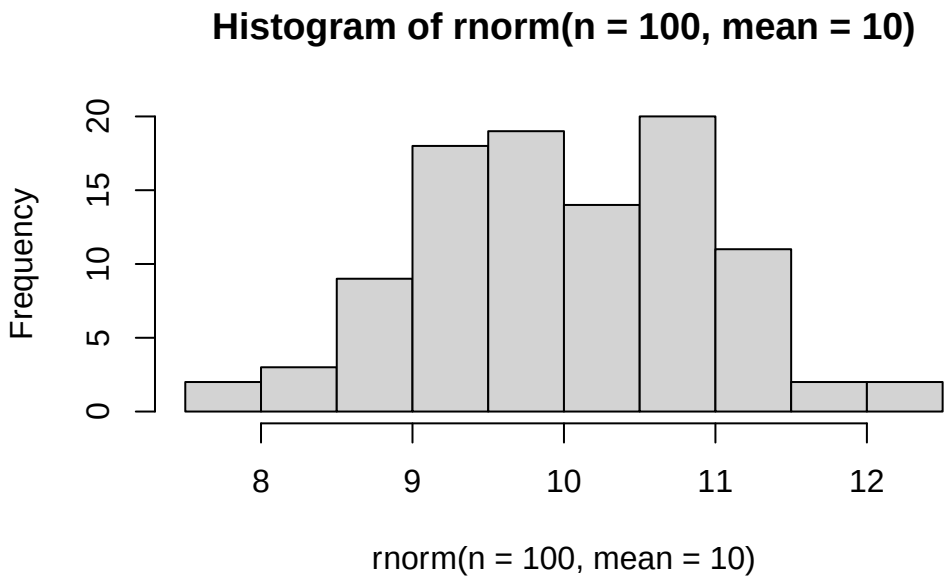
Aaron (PID: A17544470)

Today we will explore unsupervised machinelearning methods including clustering and dimensionality reduction methods.

Let's start by making up some data (where we know that 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=100, mean = 10))
```



Make data `z` with two "clusters".

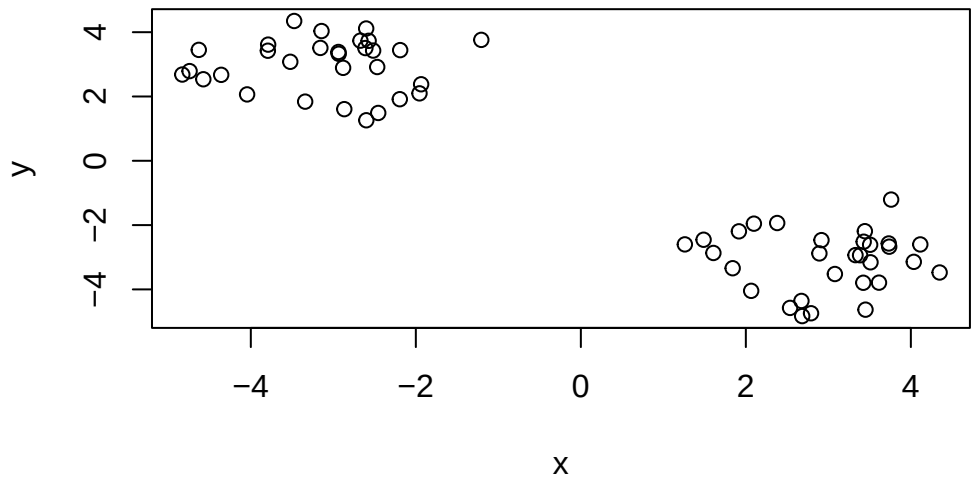
```r
x <- c(rnorm(30, mean = -3),
  rnorm(30, mean = 3)
)

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

head(z)
```

```
             x        y
[1,] -2.517374 3.428300
[2,] -2.600177 1.260736
[3,] -4.359256 2.673686
[4,] -4.830704 2.682922
[5,] -4.575557 2.535801
[6,] -2.934680 3.328967
```

```r
plot(z)
```



How big is `z`

```
nrow(z)
```

```
[1] 60
```

```
ncol(z)
```

```
[1] 2
```

K-means clustering

The main function in "base" in 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  2.968001 -3.100195
2 -3.100195  2.968001

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 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] 44.44869 44.44869
 (between_SS / total_SS =  92.6 %)

Available components:

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

finding what are the components of something

```
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 lies in which cluster)?

```
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 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
```

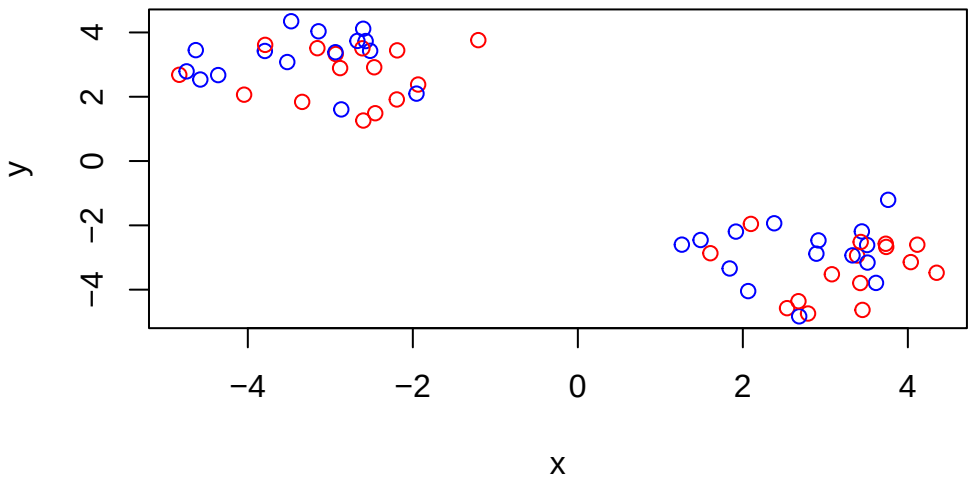
Q. Center of each cluster?

```
k$centers
```

```
          x         y
1  2.968001 -3.100195
2 -3.100195  2.968001
```

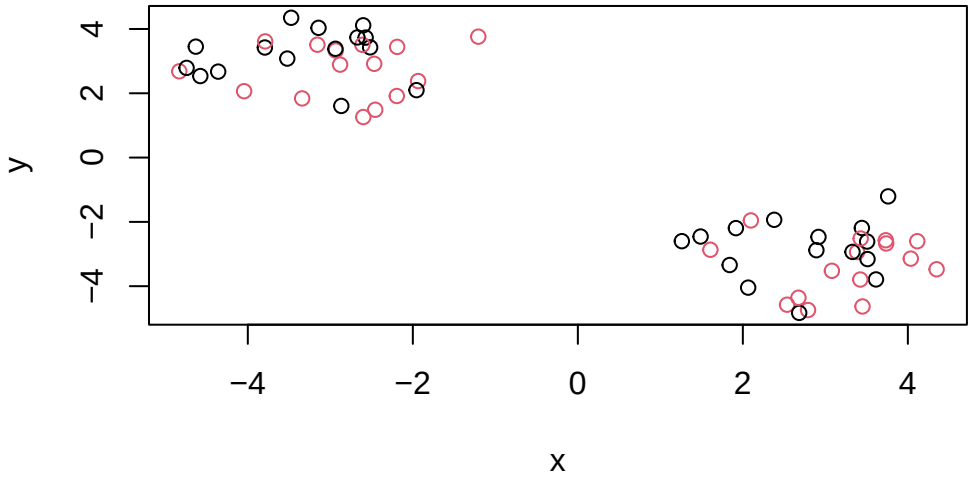
Q. Put this result info together and make a little "base R" plot of our clustering result. Also add the cluster center points to this plot.

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



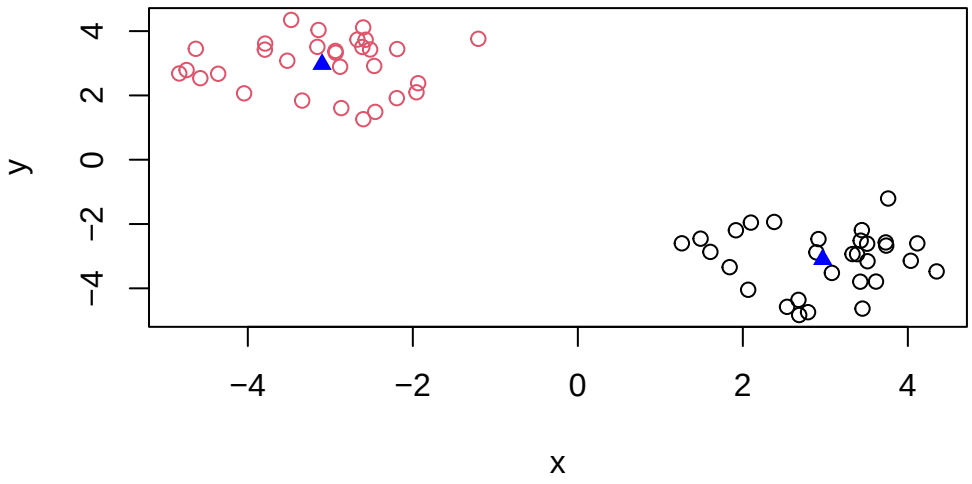
You can also color by number, such as (1,2,3…)

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



You can also plot by cluster membership

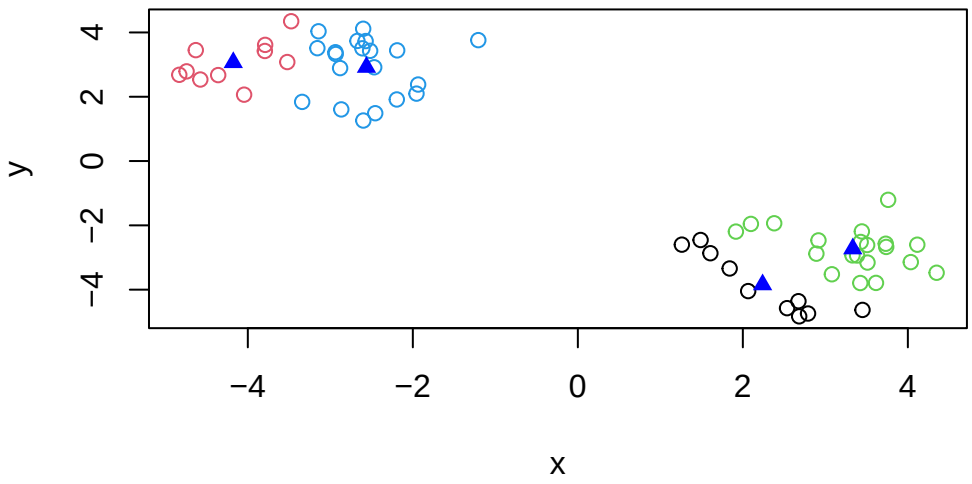
```
plot(z, col = k$cluster)
points(k$centers, col = "blue", pch = 17)
```



Q. Run kmeans on our input `z` and define 4 clusters, making the same result visualization plot as above (plot of ze colored by cluster membership).

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

plot(z, col = k4$cluster)
points(k4$centers, col = "blue", pch = 17)
```




```
k4$totss
```

```
[1] 1193.588
```

```
k4$tot.withinss
```

```
[1] 55.10067
```

Hierarchical Clustering

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

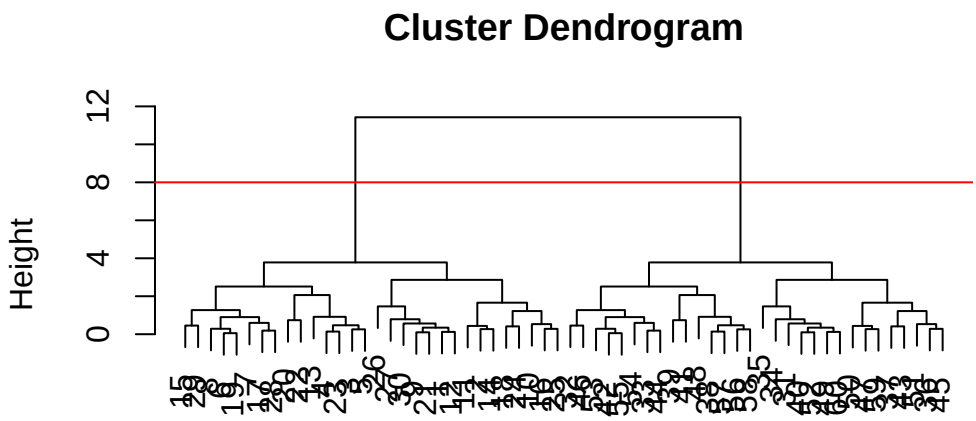
This calculates every distance between each point to another.

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

```
Call:
hclust(d = d)

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

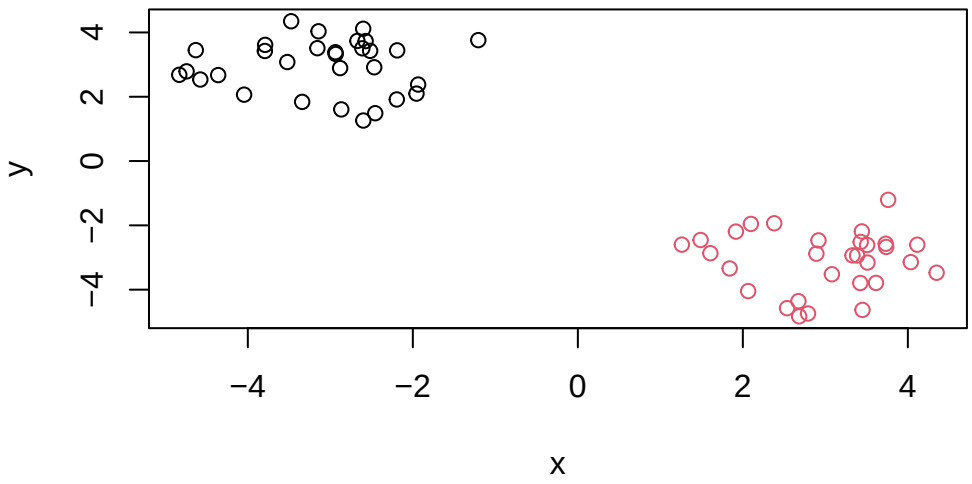
```
plot(hc)
abline(h=8, col = "red")
```



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

```
groups <- cutree(hc, h = 8)
```

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



Hands on with Principal Component Analysis (PCA)

Let's examine a 17-dimensional data detailing food consumption in the UK(England, Scotland, N. Ireland, Wales)

```
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
```

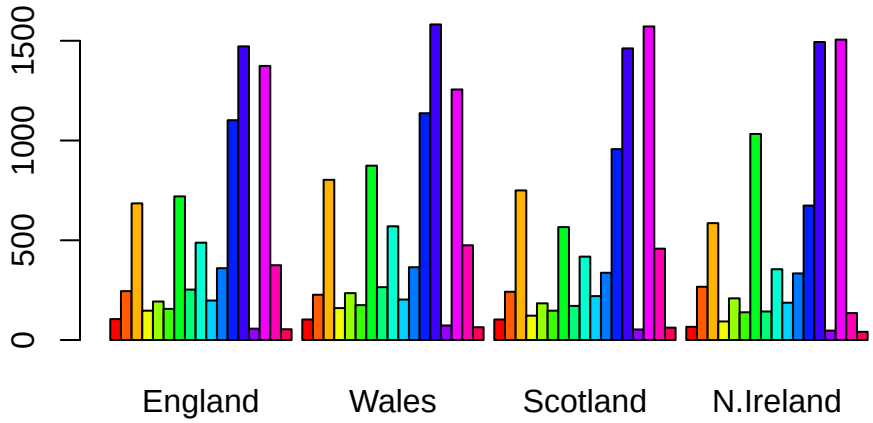
```
dim(x)
```

```
[1] 17  4
```

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 x <- read.csv(url, row.names=1) approach is the one I prefer more because if it is ran multiple times it may return an error with an incorrect number of dimensions

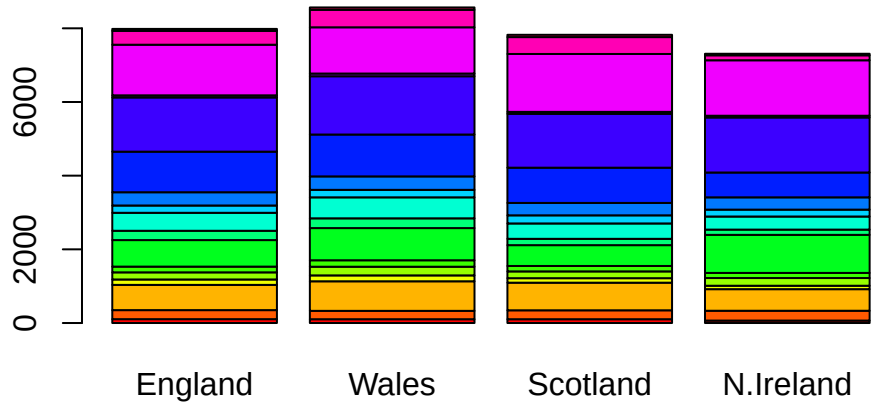
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

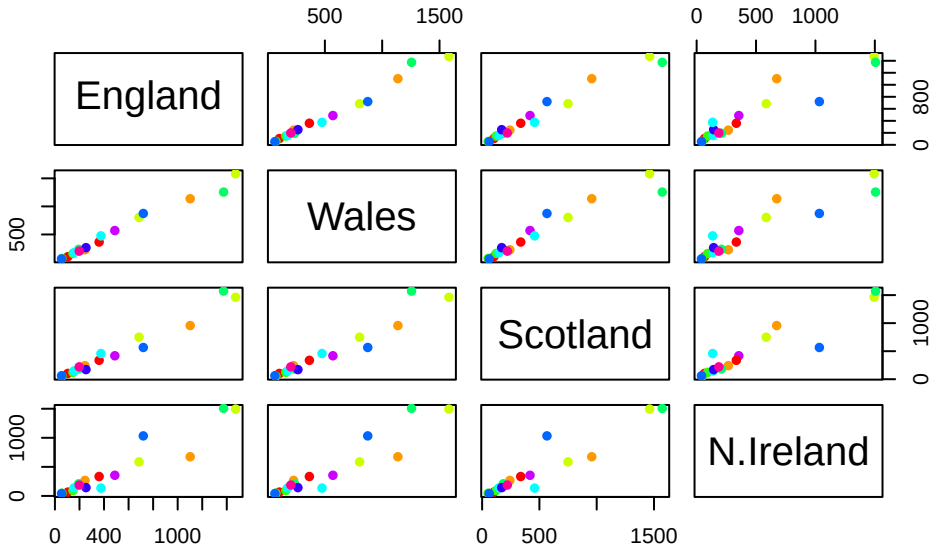
Changing the beside argument to be false results in the following plot.

```
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)
```



Looking at these types of "pairwise plots" can be helpful but does not scale well and kind of sucks(time consuming and error prone)! There must be a better way…

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

N. Ireland is more different compared to the other countries of UK as it varies more than other countries.

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. important food categories as column titles 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 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
```

Proportion of variance shows how much of the action is being done in that principal component.

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

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

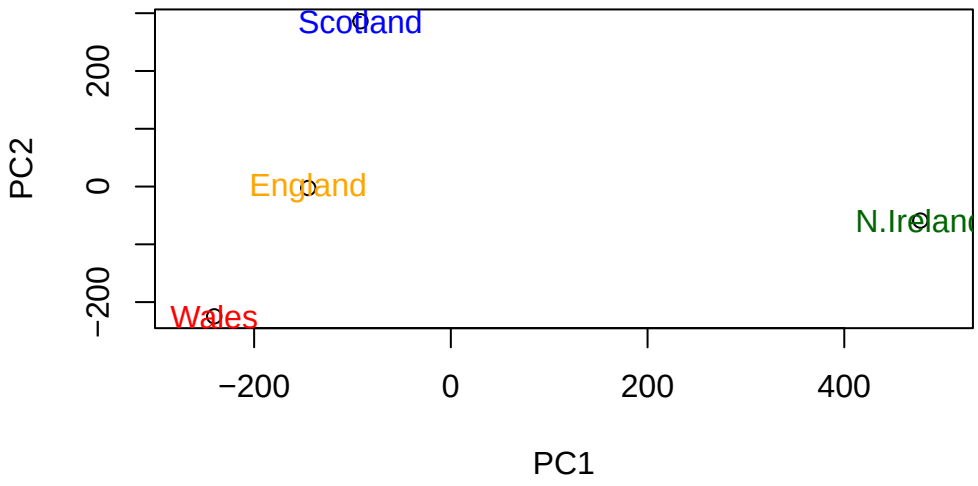
The `pca$x` result object is where we will focus first as this details how the countries are related to each other in terms of our new "axis" (aka "PCs", "eigenvectors", etc.)

```
head(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
```

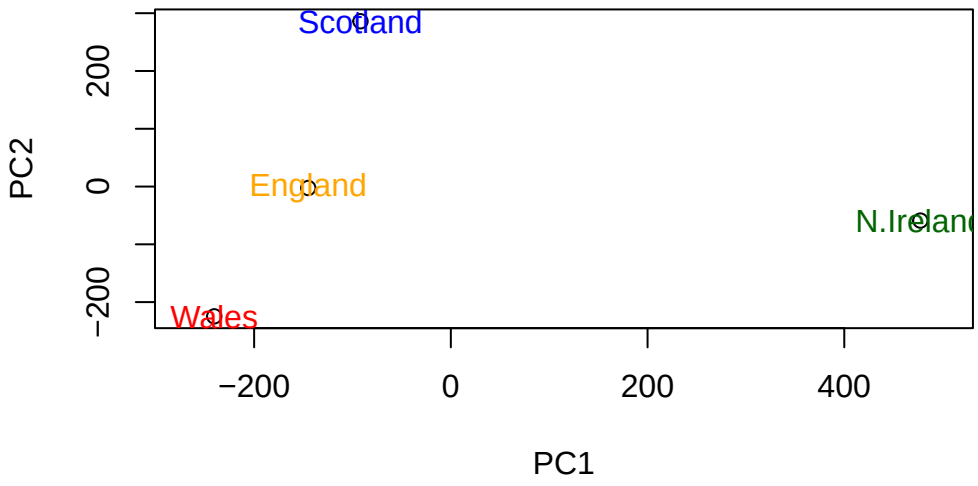
> Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

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



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

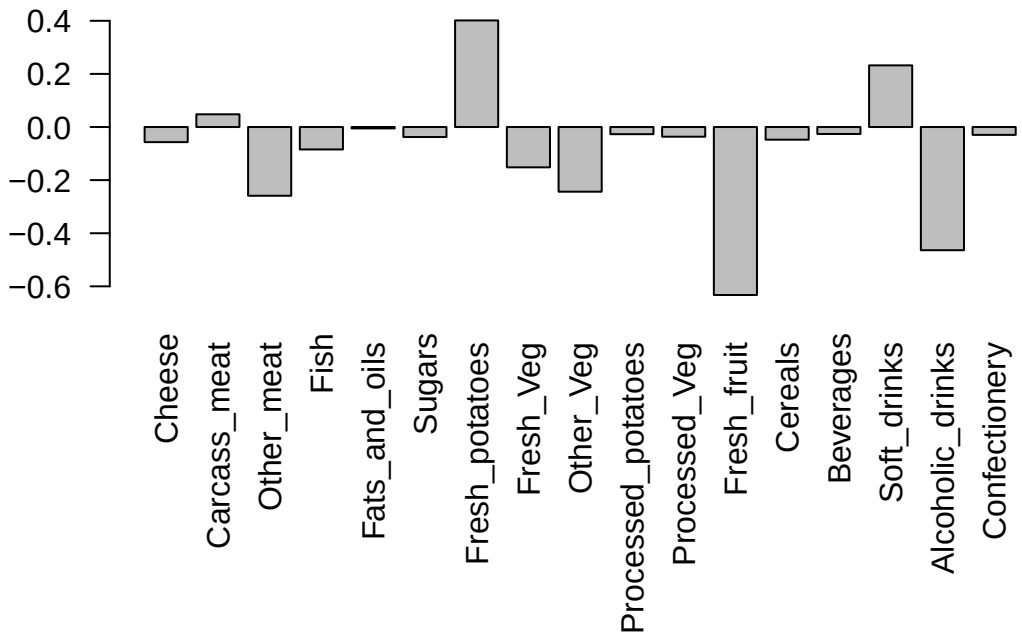


We can look at the so called PC "loadings" result object to see how the original foods contribute to our new PCs (i.e. how the original variables contribute to our new better PC variables).

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

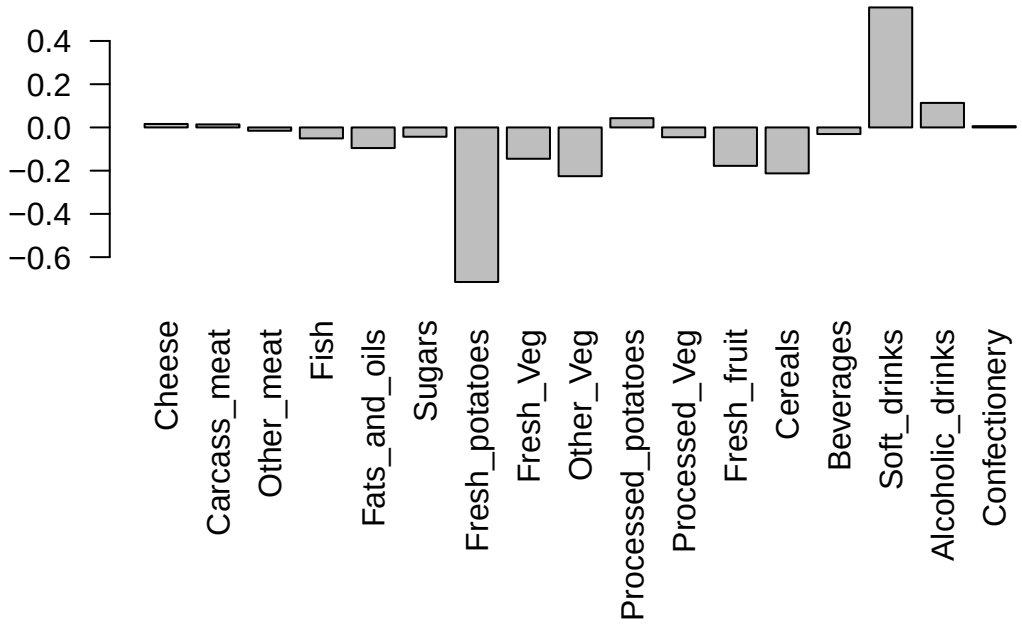
```
          Cheese       Carcass_meat         Other_meat                Fish
    -0.056955380         0.047927628       -0.258916658       -0.084414983
Fats_and_oils              Sugars     Fresh_potatoes          Fresh_Veg
    -0.005193623       -0.037620983         0.401402060       -0.151849942
   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
```

```
## 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?

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



PCA of RNA-seq data

```
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
```

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

```
[1] 100  10
```

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

There are 100 genes and 10 samples.