Class 7: Machine Learning 1

Kianna

In this class we will explore clustering and dimensionality reduction methods.

K-means

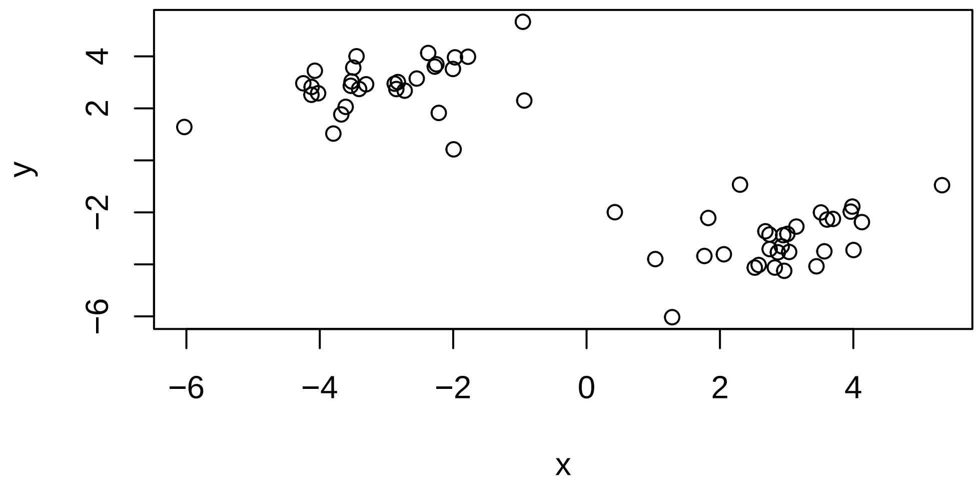
Make up some input data where we know what the answer should be

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

```
              x        y
[1,] -2.8766831 2.948859
[2,] -2.5462347 3.145958
[3,] -1.9726699 3.961154
[4,] -1.7777871 3.984962
[5,] -0.9336613 2.301225
[6,] -4.1227201 2.822851
```

Quick plot of x to see the two groups at -3, +3, and +3, -3

```
plot(x)
```



Use the 'kmeans()' function setting k to 2 and nstart = 20

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

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

Cluster means:
          x         y
1  2.897362 -3.033972
2 -3.033972  2.897362

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 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] 63.00134 63.00134
 (between_SS / total_SS =  89.3 %)

Available components:
```

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

Q. How many points are in each cluster?

```r
km$size
```

```
[1] 30 30
```

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

```r
km$cluster
```

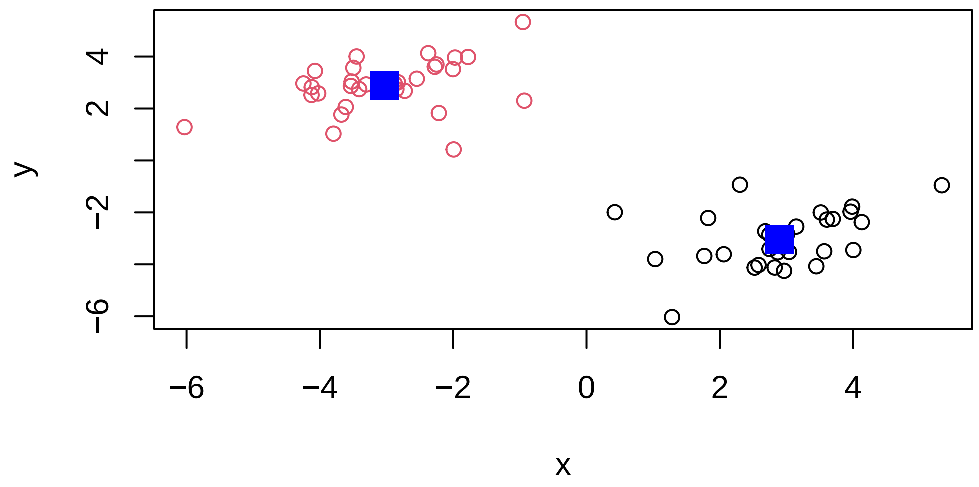
```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```r
km$centers
```

```
          x         y
1  2.897362 -3.033972
2 -3.033972  2.897362
```

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

```r
plot(x, col = km$cluster)
points(km$centers, col="blue", pch=15, cex=2)
```



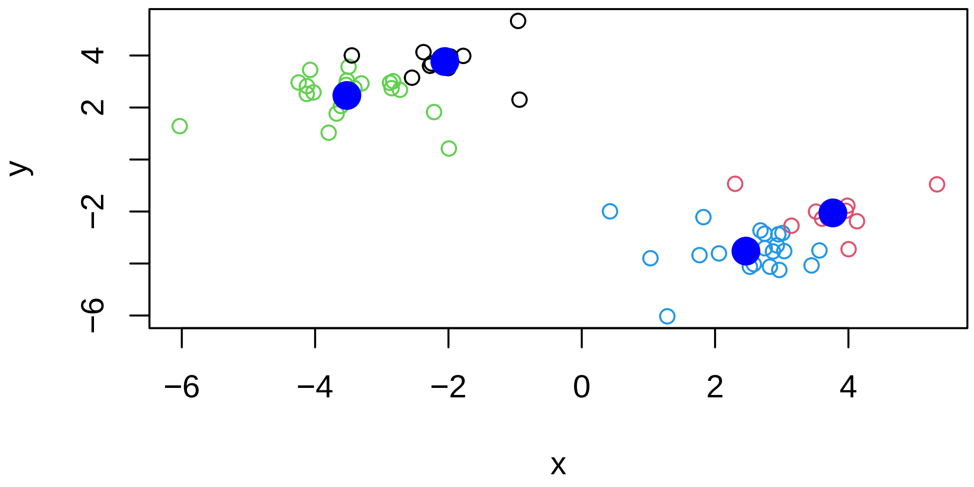
Play with kmeans and ask for different number of clusters

```
km <- kmeans(x, centers = 4, nstart=20)
km
```

```
K-means clustering with 4 clusters of sizes 10, 10, 20, 20

Cluster means:
          x         y
1 -2.054085  3.766947
2  3.766947 -2.054085
3 -3.523915  2.462569
4  2.462569 -3.523915

Clustering vector:
 [1] 3 1 1 1 1 3 3 3 1 3 3 1 3 3 1 3 1 3 3 3 3 3 3 3 3 1 3 3 3 1 2 4 4 4 2 4 4 4
[39] 4 4 4 4 4 2 4 2 4 4 2 4 4 2 4 4 4 2 2 2 2 4

Within cluster sum of squares by cluster:
[1] 10.27317 10.27317 26.98284 26.98284
 (between_SS / total_SS =  93.7 %)
```

```
Available components:

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

```
plot(x, col = km$cluster)
points(km$centers, col="blue", pch=16, cex=2)
```



```
1:6 + c(100, 1)
```

```
[1] 101   3 103   5 105   7
```

```
c(100,1)
```

```
[1] 100   1
```

Hierarchical Clustering

This is another very useful and widely employed clustering method which has the advantage over k-means in that it can help reveal the something of the true grouping in your data.

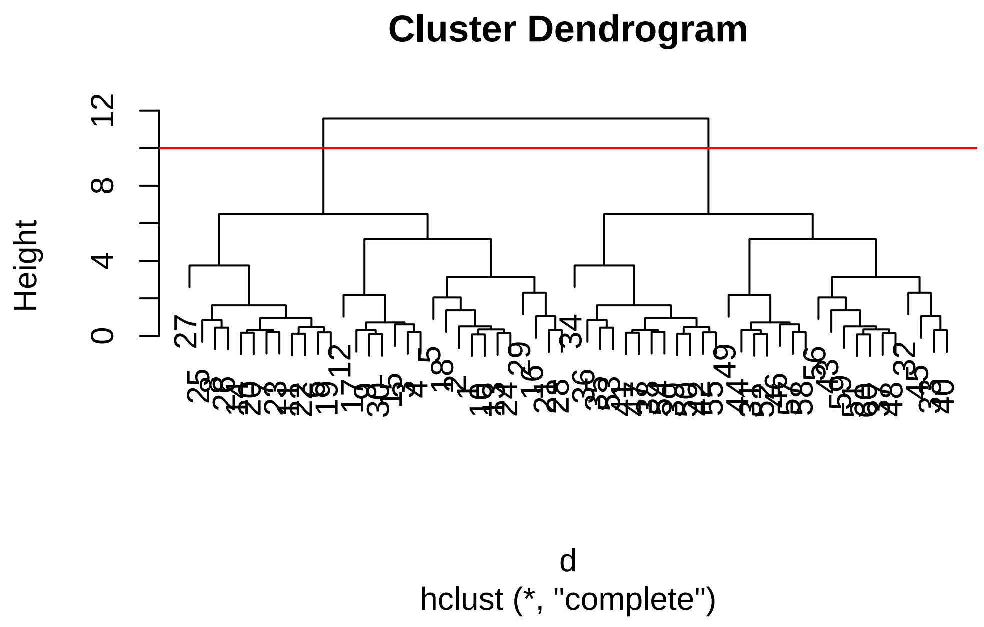
The 'hclust()' function wants a distance matrix as input. We can get this from the 'dist()' function.

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

```

Call:
hclust(d = d)

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

There is a plot method for hclust results:

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

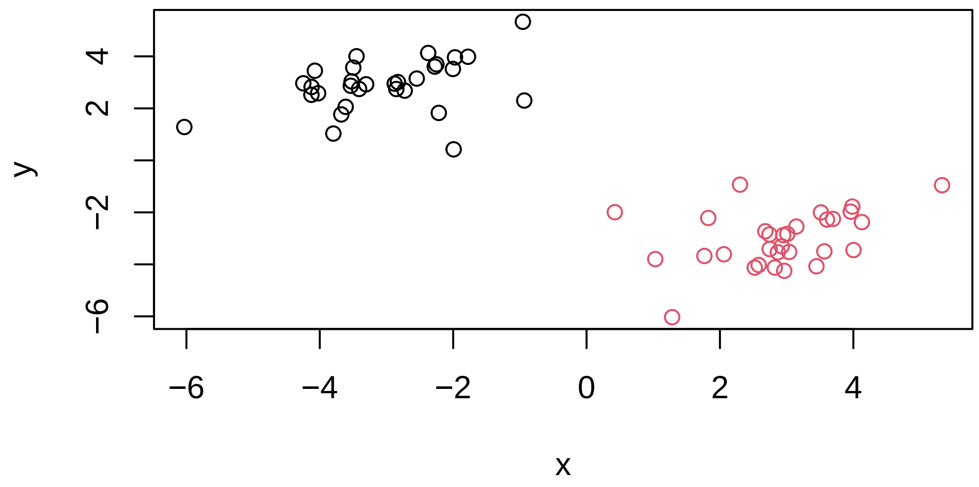


To get my cluster membership vector, I need to "cut" my tree to yield sub-trees of branches with all the members of a given cluster residing on the same cut branch. The function to do this is called 'cutree()'

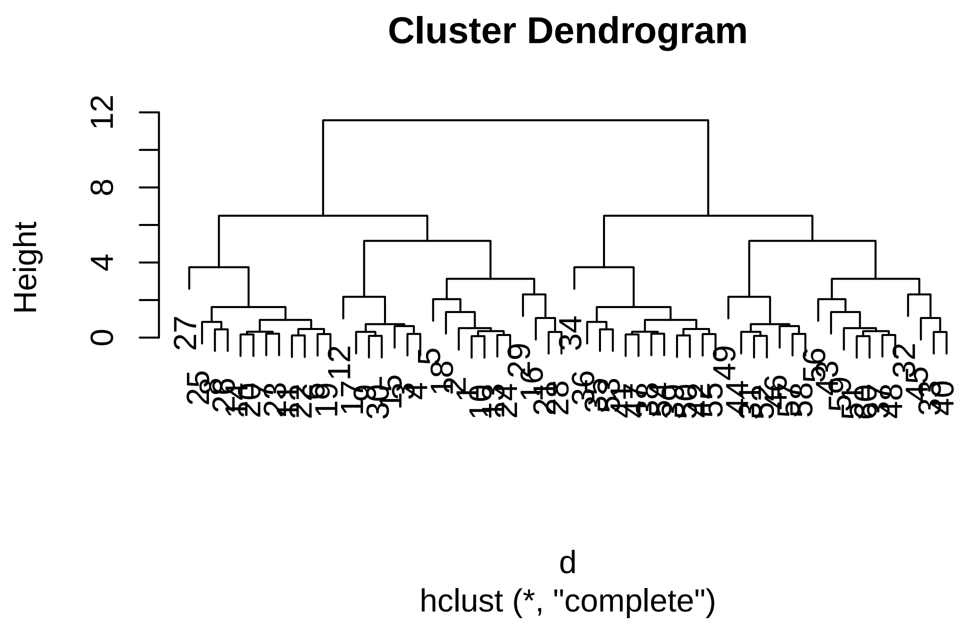
```r
grps <- cutree(hc, h=10)
grps
```

```
 [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 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
```

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



```
plot(hc)
```



It is often helpful to use the 'k=' argument to cutree rather than the 'h=' height of cutting with 'cutree()'. This will cut the tree to yield the number of clusters you want. 'k=4' would give use a cut that yields 4 clusters.

```r
cutree(hc, k=4)
```

```
 [1] 1 1 1 1 1 2 2 2 1 1 2 1 1 2 1 1 1 1 2 2 1 2 2 1 2 2 2 1 1 1 3 3 3 4 4 4 3 4
[39] 4 3 4 4 3 3 3 3 4 3 3 4 3 3 4 4 4 3 3 3 3 3
```

Principal Component Analysis (PCA)

The base R function for PCA is called 'prcomp()'

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

```r
nrow(x)
```

```
[1] 17
```

```r
ncol(x)
```

```
[1] 5
```

```r
dim(x)
```

```
[1] 17  5
```

Q1. There are 17 rows and 5 columns in data frame x.

```r
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
```

```
# 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
```

```
nrow(x)
```

```
[1] 17
```

```
ncol(x)
```

```
[1] 4
```

```
dim(x)
```

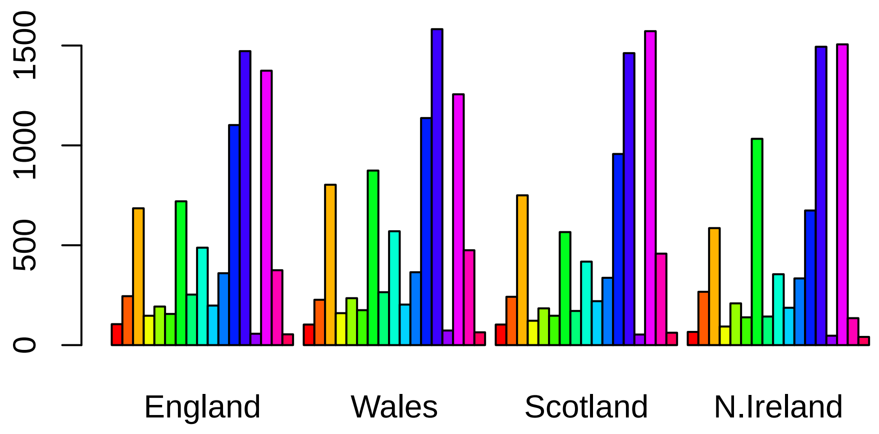
```
[1] 17  4
```

```
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
```

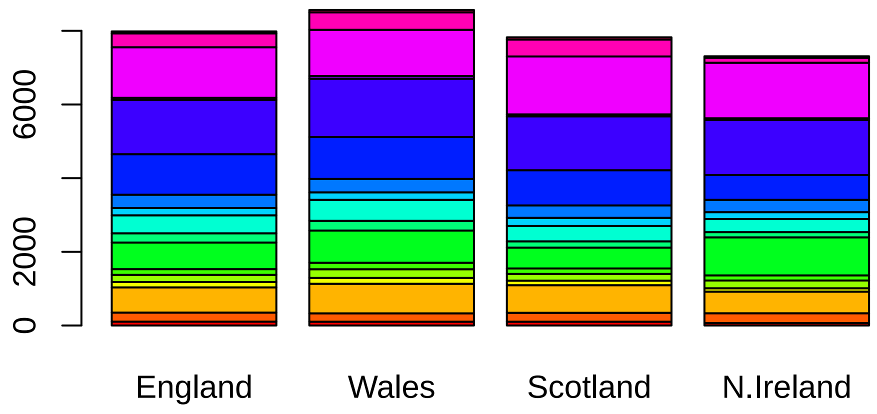
Q2. I prefer the second approach because it is more simple than the first approach as there is one less line of code. It requires defining one less variable which I like better. The second approach is more robust than the first because every time it reruns, it will only affect the first column and treat it as the row names. With the first approach, each time you play it the first column will get deleted over and over.

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



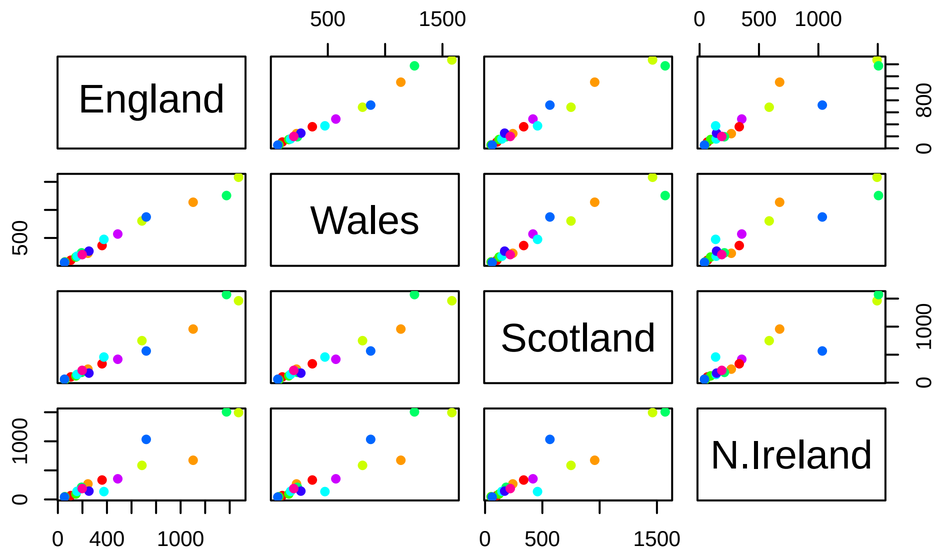
Q3. By changing beside=T to beside=F, the bars within each country will be on top of each other instead of next to ecah other.

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



Q5. The 'pairs()' function gives a matrix of scatter plots comparing variables to each other within the data frame. If a given point lies on the diagonal for a given plot, that means the two countries are similar to each other in regard to their eating habits for that food group.

```
pairs(x, col=rainbow(10), pch=16)
```



Q6. The main differences between N. Ireland and the other countries are in terms of the food group that is the dark blue and orange dots. The blue dot is above the diagonal with N. Ireland on the y-axis meaning Irish people eat more of that food group. The orange dot is below the diagonal with N. Ireland on the y-axis meaning Irish people eat less of that food group.

```
# 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 5.552e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

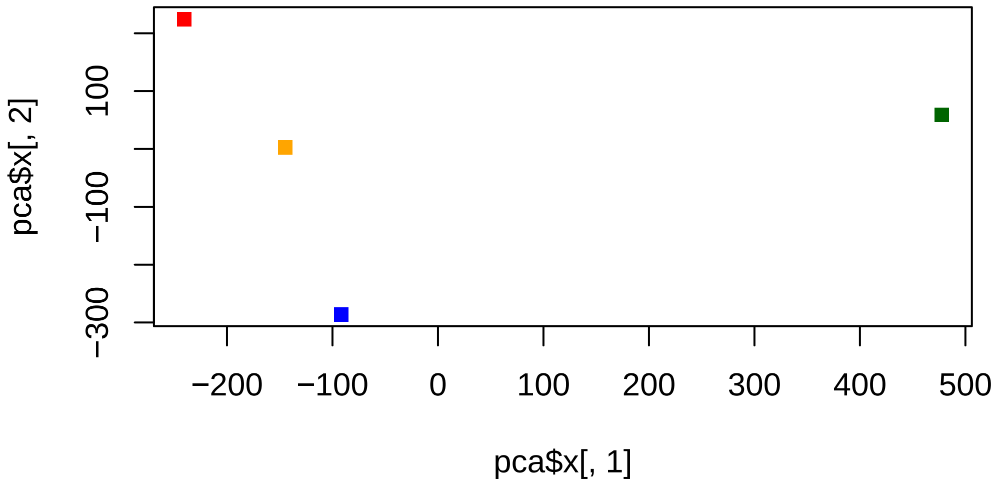
A “PCA plot” (a.k.a “Score plot”, PC1vsPC2 plot, etc.)

```
pca$x
```

```
                 PC1         PC2         PC3           PC4
```

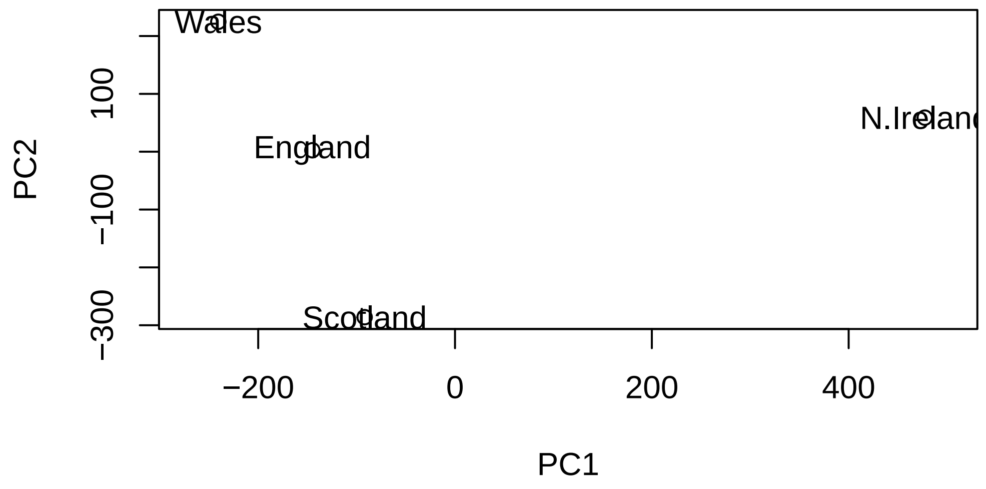
```
England   -144.99315    2.532999 -105.768945  1.042460e-14
Wales     -240.52915  224.646925   56.475555  9.556806e-13
Scotland   -91.86934 -286.081786   44.415495 -1.257152e-12
N.Ireland  477.39164   58.901862    4.877895  2.872787e-13
```

```
plot(pca$x[,1], pca$x[,2],
    col=c("orange", "red", "blue", "darkgreen"), pch=15)
```



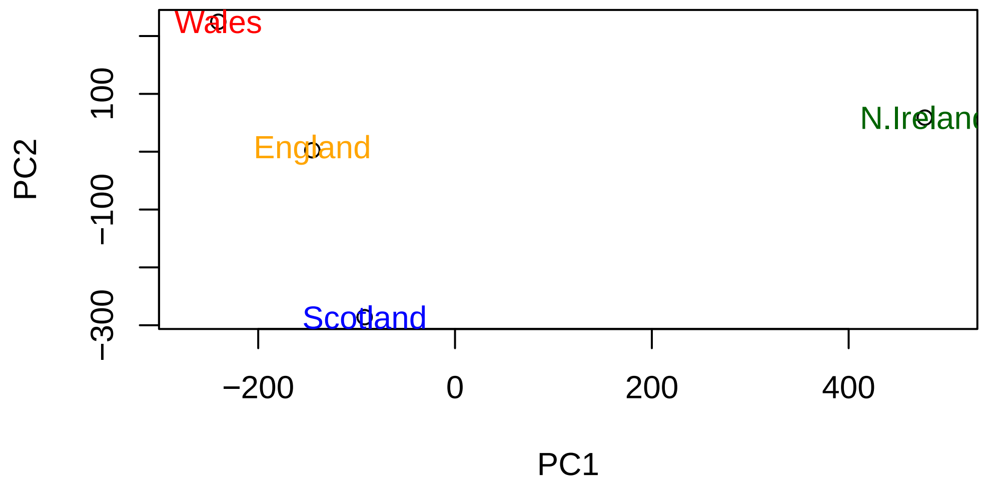
Q7.

```
# 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.

```
# 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), col=c("orange", "red", "blue", "darkgreen"))
```



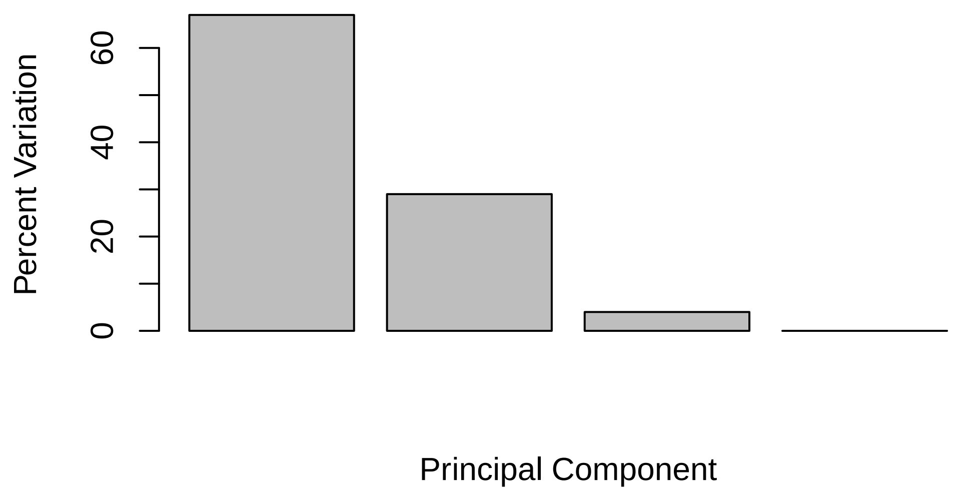
```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

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

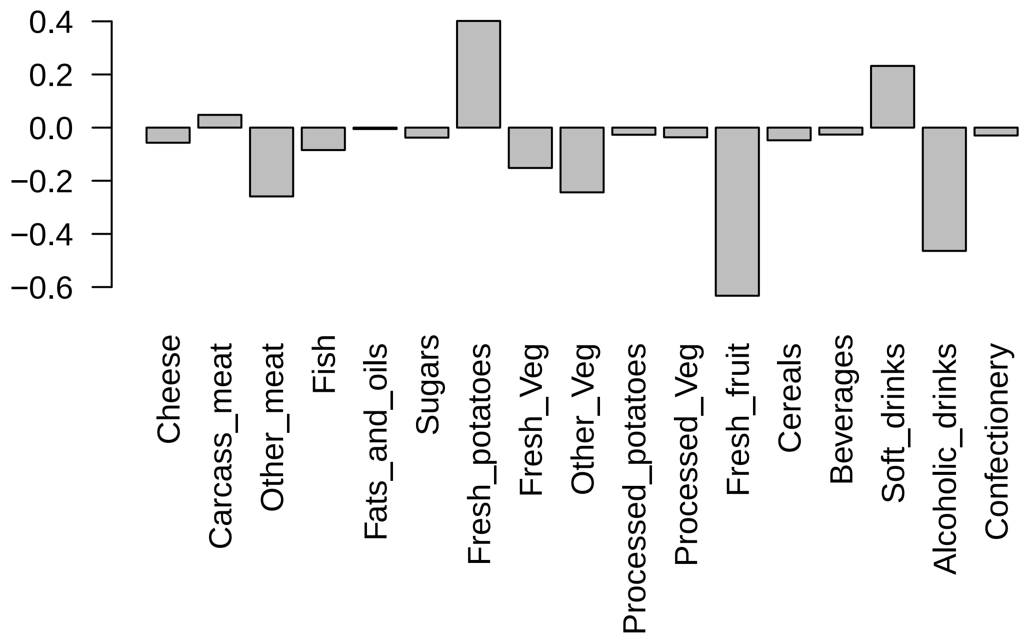
```
## or the second row here...
z <- summary(pca)
z$importance
```

```
                             PC1       PC2      PC3          PC4
Standard deviation     324.15019 212.74780 73.87622 5.551558e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

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

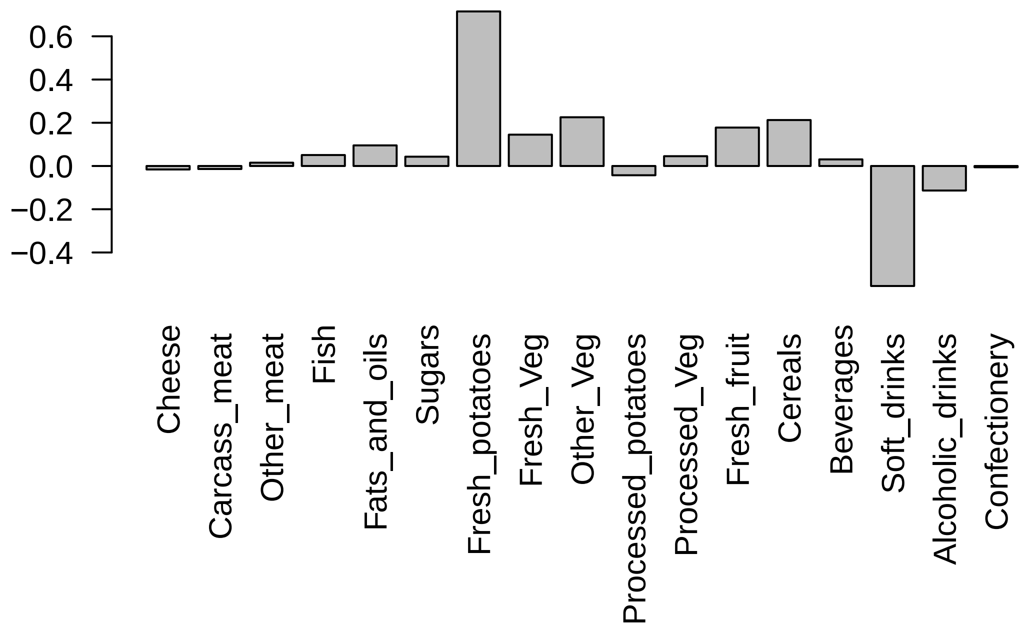


```r
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



Q9.

```r
## 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 )
```



The food with the largest positive value is fresh potatoes which pushes N. Ireland to the right side of the plot. N. Ireland consumes more of this than other countries. The food with the largest negative value is soft drinks which pushes the other countries to the left side of the plot. N. Ireland consumes less of this than other countries.