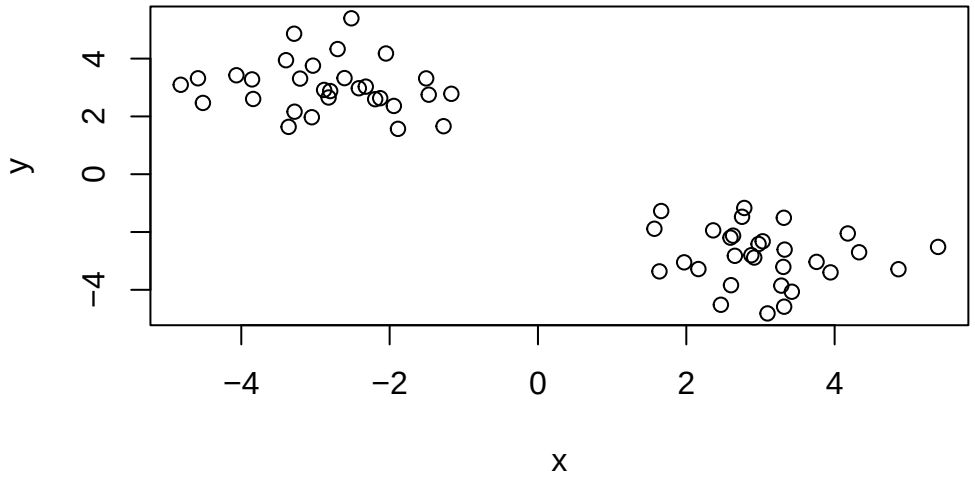
Class 07: Machine Learning 1

Isabel Mejia

K-means clustering

Let's make up some data to cluster.

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



The function to do k-means clustering in base R is called `kmeans()`. We give this our input data for clustering and the number of clusters we want `centers`.

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

```
K-means clustering with 4 clusters of sizes 12, 18, 11, 19

Cluster means:
          x         y
1  3.648355 -3.651841
2  2.633003 -2.287212
3 -1.850903  2.713561
4 -3.401684  3.227639

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

Within cluster sum of squares by cluster:
[1] 14.484890 15.254158  7.083316 24.891021
 (between_SS / total_SS =  94.6 %)

Available components:

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

Cluster size

```r
km$size
```

```
[1] 12 18 11 19
```

Cluster assignment/membership

```r
km$cluster
```

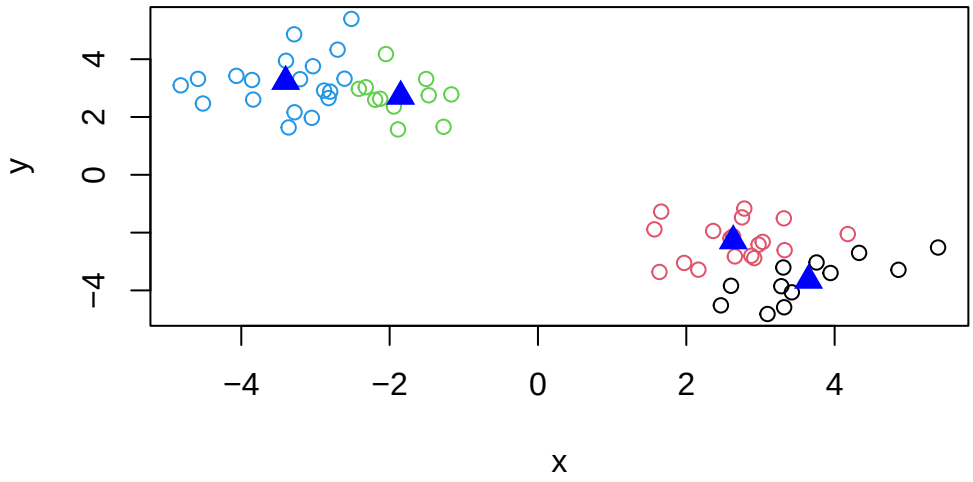
```
 [1] 4 3 4 3 4 4 3 4 4 4 4 4 4 4 3 3 3 4 4 4 3 4 4 4 3 3 3 4 3 4 1 2 2 2 2 2 1 1
[39] 2 2 2 1 2 2 2 2 1 1 2 1 1 1 2 2 1 2 2 1 2 1
```

Cluster center

```
km$centers
```

```
          x         y
1  3.648355 -3.651841
2  2.633003 -2.287212
3 -1.850903  2.713561
4 -3.401684  3.227639
```

```
plot(x, col=km$cluster)
points(km$centers, col="blue", pch=17, cex=1.5)
```



Heirarchial CLustering

The `hclust()` function performs hierarchical clustering. The big advantage here is I don't need to tell it "k" the number of clusters

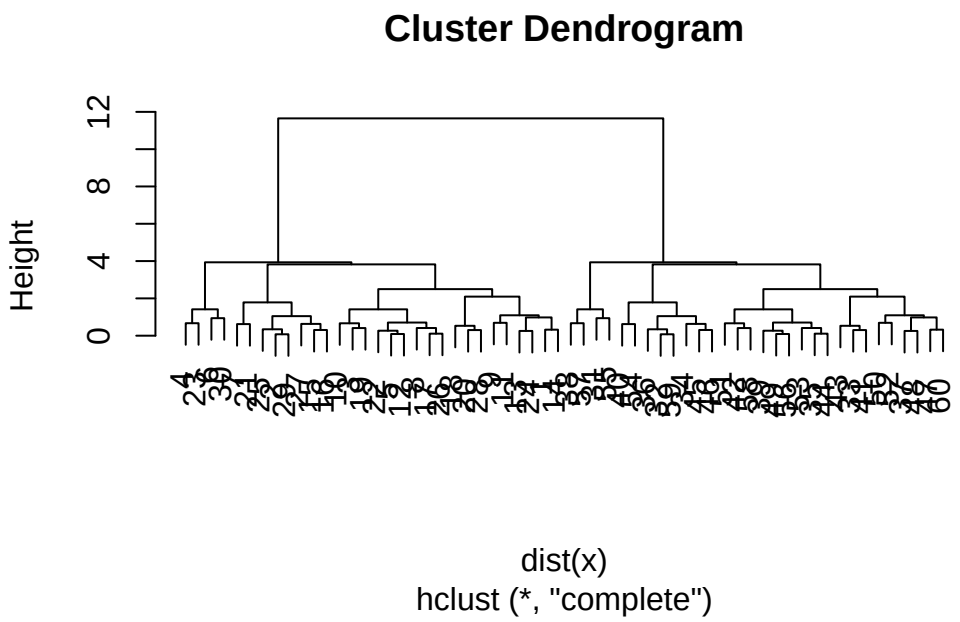
To run `hclust()` I need to provide a distance matrix as input (not the original data)

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

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

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

```
plot(hc)
```



Function to "cut" the tree to get the main result (which is cluster membership). Put branches into groups

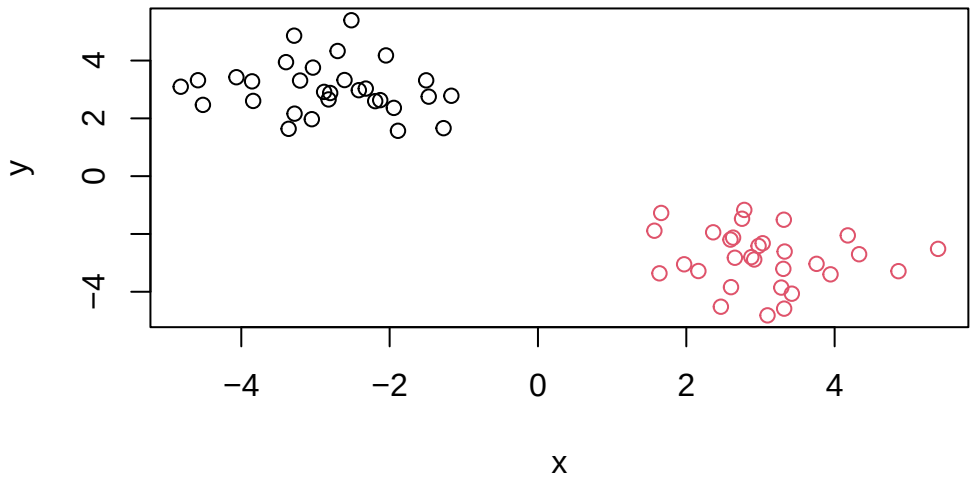
```
cutree(hc, h=8 )
```

```
 [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
```

More often we will use `cutree()` with k=2 for example

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

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



Principal Component Analysis (PCA)

Read data for UK food trends from online

```
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

Check the data:

```
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
```

Fixing the row names

```
#rownames(x) <- x[,1]
#x <- x[,-1]
#head(x)
```

Alternative way of changing the row names

```
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
```

Checking to make sure the rows and columns have changed

```
dim(x)
```

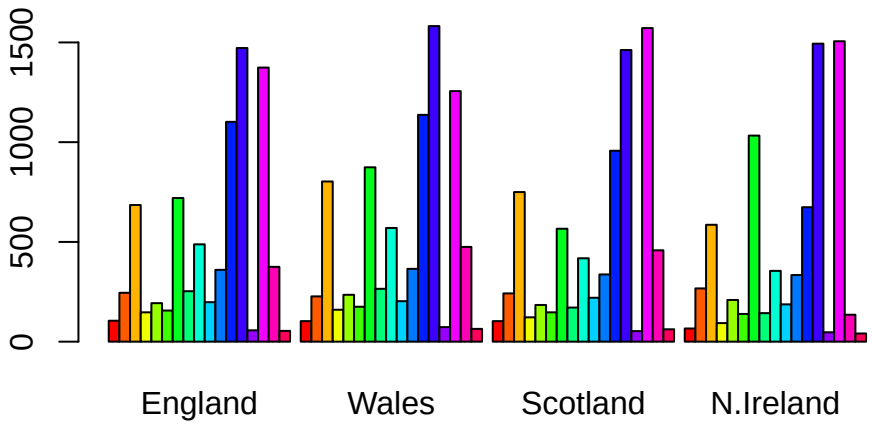
```
[1] 17  4
```

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?

I prefer the method in which you specify the row.names in the in the `read.csv()` function because it's much simpler to specify that the first column is the name. This approach seems to be more robust because if you rerun the code of the first approach then data could be deleted.

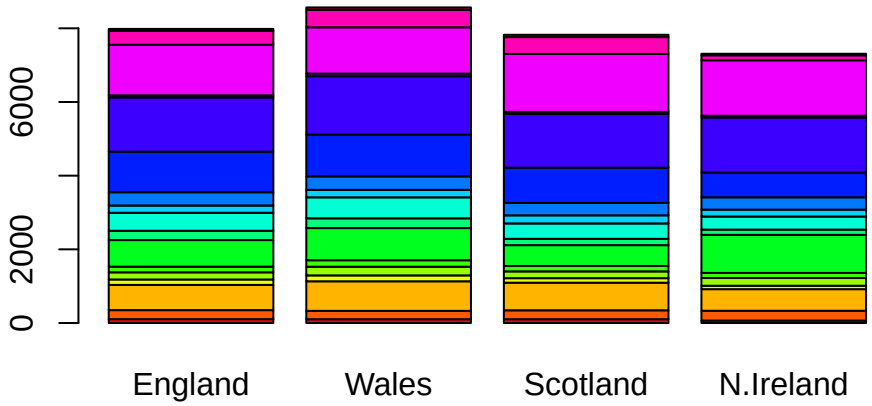
Spotting major differences and trends.

```
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)))
```

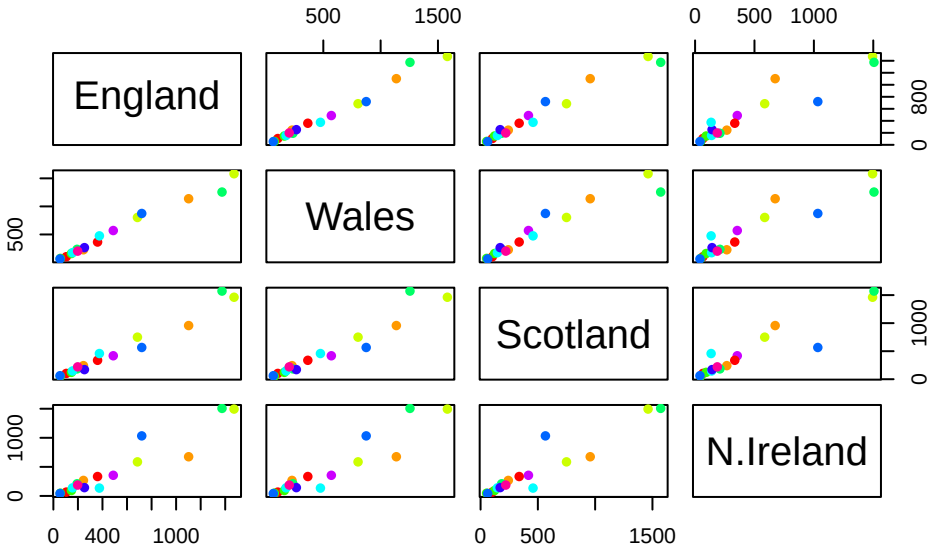


Changing the beside argument to "F" will stack the barplots

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?

A pairs plot is somewhat useful.

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



In this plot it is all possible pairwise plots. if a given point lies on the diagonal for a given plot, that means the numbers between the two countries are similar, if not the same.

PCA to the rescue

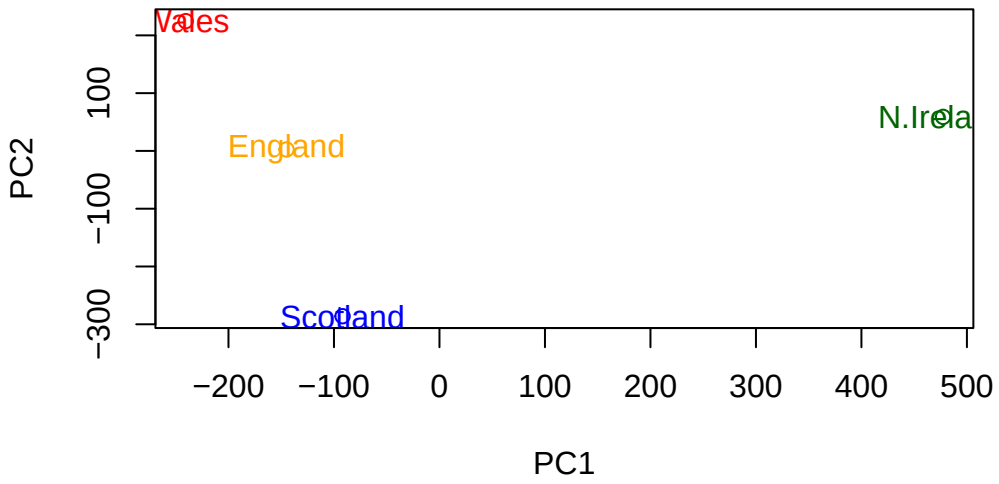
The main function in base R to do PCA is called `prcomp()`. One issue with the `prcomp()` function is that it expect the transpose of our data as input.

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

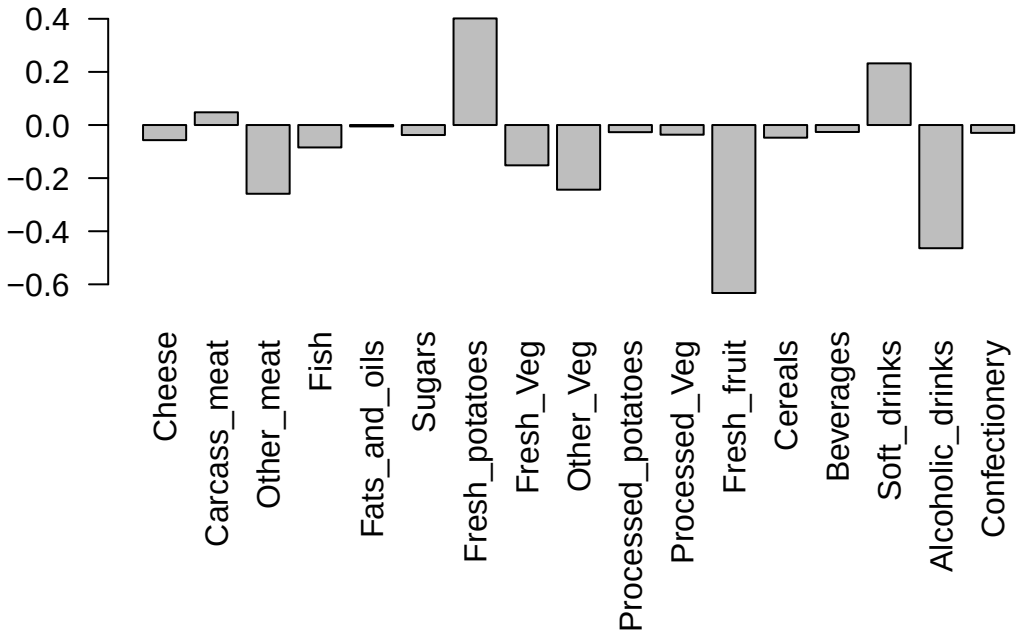
```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 4.189e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

The object returned by `prcomp()` has our results that include a $x component

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




```
par(mar=c(10, 3, 0.35, 0))
barplot (pca$rotation[,1], las=2)
```


0.4
0.2
0.0
−0.2
−0.4
−0.6
Cheese
Carcass_meat
Other_meat
Fish
Fats_and_oils
Sugars
Fresh_potatoes
Fresh_Veg
Other_Veg
Processed_potatoes
Processed_Veg
Fresh_fruit
Cereals
Beverages
Soft_drinks
Alcoholic_drinks
Confectionery