Class 7: Machine Learning

Michelle Woo

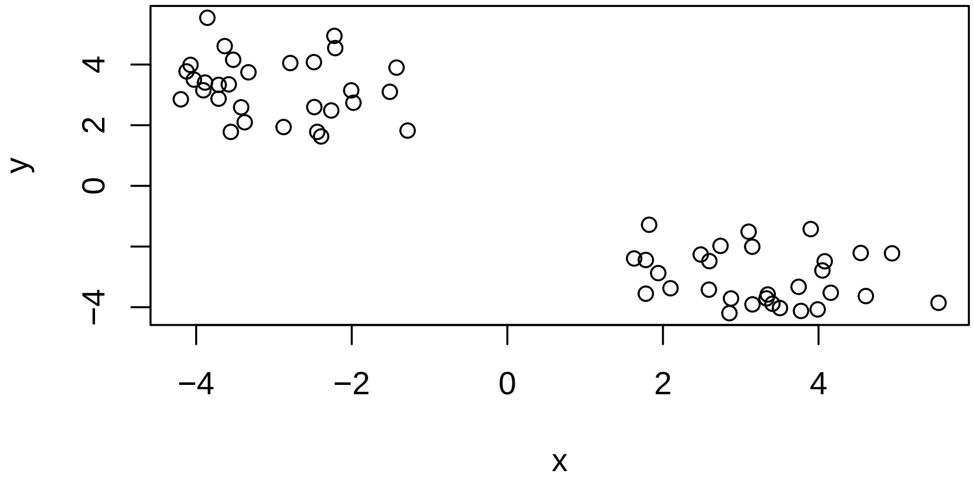
Example of K-means clustering

First step is to make up some data with a known structure, so we know what the answer should be

```
# generating random data
tmp <- c(rnorm(30, mean=-3), rnorm(30, mean=3))
tmp
```

```
 [1] -3.421605 -2.263818 -4.072744 -2.006747 -2.223410 -4.197326 -1.424704
 [8] -3.580889 -3.633132 -1.978511 -2.212551 -2.790069 -2.393998 -2.481646
[15] -3.328053 -4.029252 -3.886351 -3.376245 -2.443714 -3.524116 -1.510642
[22] -1.282096 -3.554522 -2.486291 -2.876201 -3.712021 -3.907400 -4.123471
[29] -3.710735 -3.856055  5.542449  3.330610  3.774362  3.150696  2.874897
[36]  1.939550  4.079208  1.779662  1.822530  3.101260  4.157091  1.780063
[43]  2.096811  3.407007  3.503618  3.743822  2.596999  1.629745  4.049740
[50]  4.542132  2.739943  4.607806  3.346997  3.899235  2.854375  4.944517
[57]  3.147905  3.988761  2.485760  2.590351
```

```
# visualizing in 3D
x <- cbind(x=tmp, y=rev(tmp))
plot(x)
```



Now we have some structured data in `x`. Let's see if k-means is able to identify the two groups

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

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

Cluster means:
          x         y
1 -3.009611  3.250263
2  3.250263 -3.009611

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 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

Within cluster sum of squares by cluster:
[1] 52.17816 52.17816
 (between_SS / total_SS =  91.8 %)

Available components:
```

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

Let’s explore and better understand k:

```
# how many elements are in each group?
k$size
```

```
[1] 30 30
```

```
k$centers
```

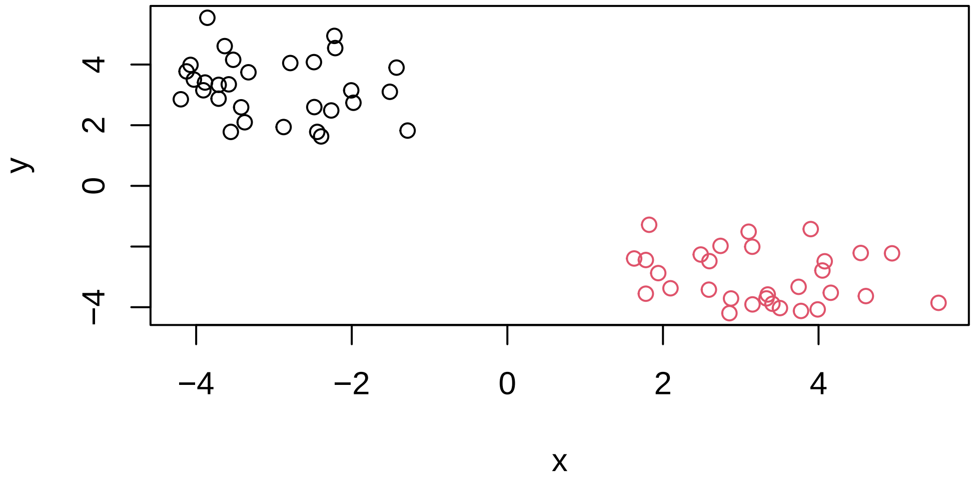
```
          x         y
1 -3.009611  3.250263
2  3.250263 -3.009611
```

```
# able to use this to color the plot
k$cluster
```

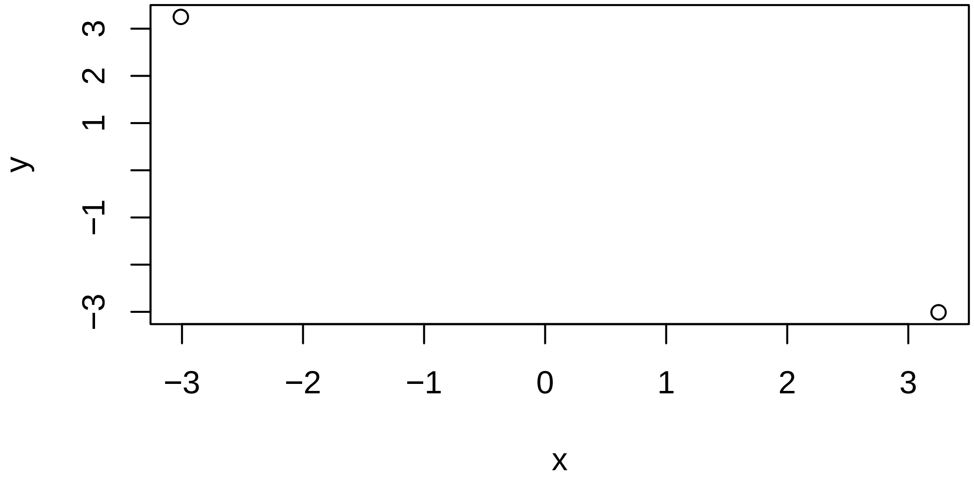
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Refining the plot:

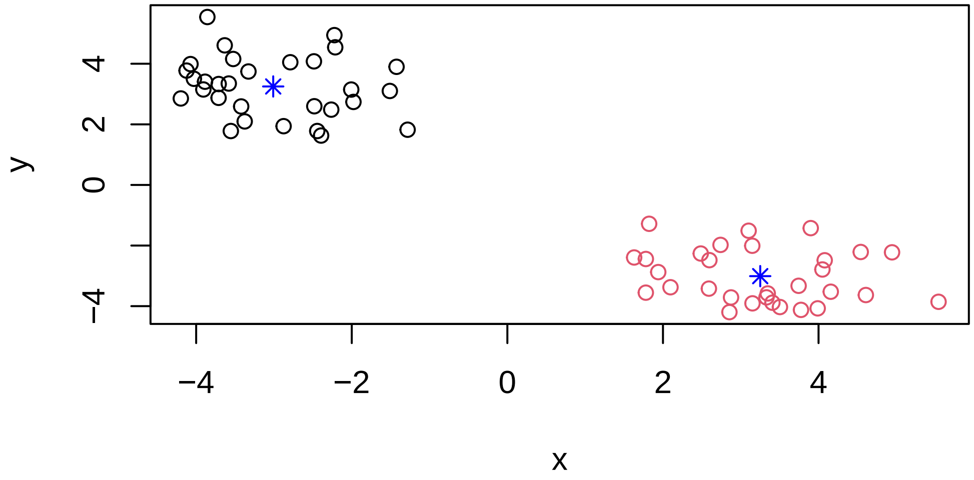
```
# coloring the different groups
plot(x, col=k$cluster)
```



```
# adding in cluster centers, plot(x, col=k$cluster)
plot(k$centers)
```



```
# want to overlap the two above
plot(x, col=k$cluster)
points(k$centers, col = 'blue', pch = 8)
```



Example of Hierarchical Clustering

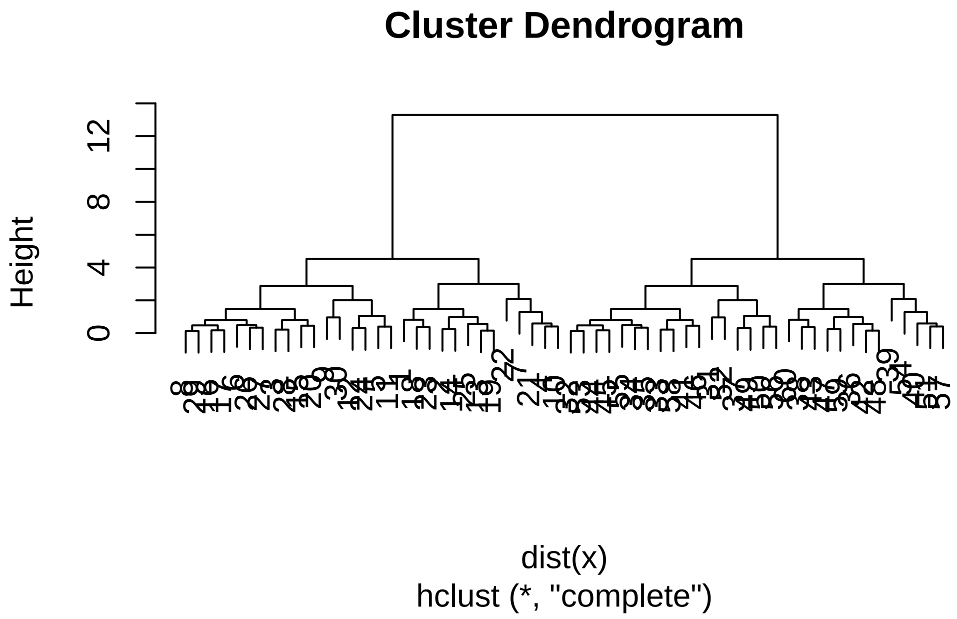
Let's use the same data as before, which we stored in `x`. We will use the `hclust()` function

dist(x) calculates the distance between all the points, this is input required for clustering

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

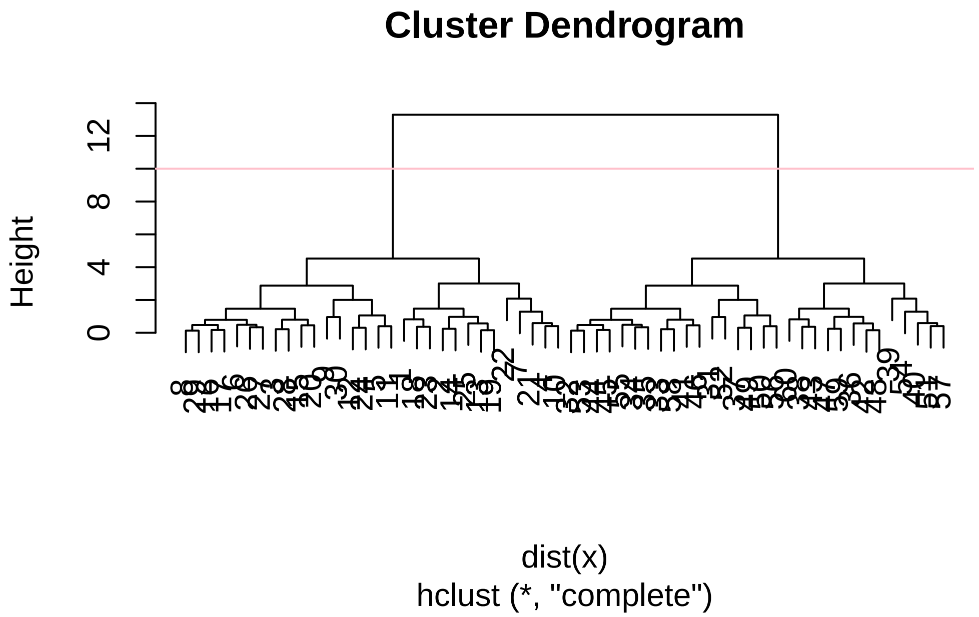
```

Call:
hclust(d = dist(x))

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

```
# results in tree, plot function gives something different
plot(clustering)
```



Lets add a horizontal line

```
plot(clustering)
abline(h=10, col='pink') # results in 6 classifications
```



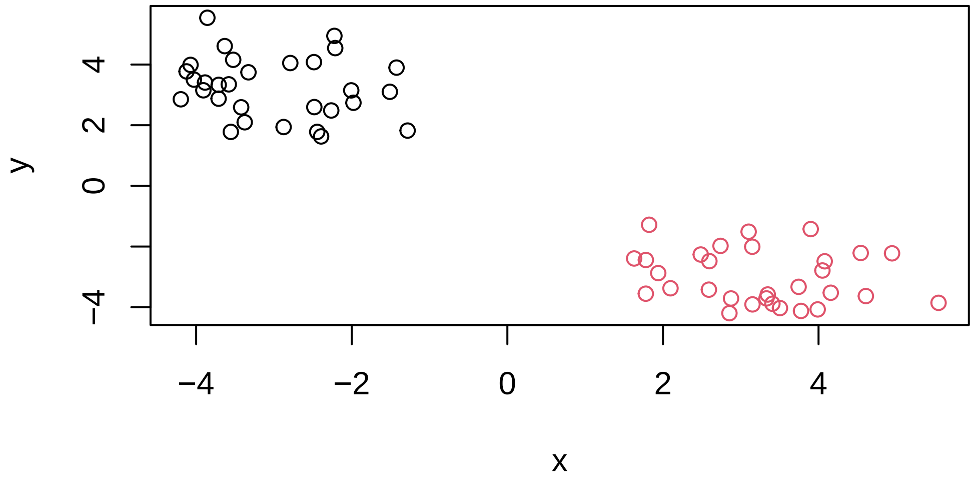
To get our results (i.e. membership vector) we need to 'cut' the tree at the chosen height. The function for doing that is `cutree()`

```
# able to get membership clustering
subgroups <- cutree(clustering, h=10)
subgroups
```

```
 [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
```

Plotting this:

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



You can cut your tree with the number of clusters you want:

```
cutree(clustering, k=2)
```

```
 [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
```

Principal Component Analysis (PCA)

PCA of UK food

First, we need to read the data

```
url <- "https://tinyurl.com/UK-foods"

# making sure foods are the first column and for our rows
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
```

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

```r
nrow(x)
```

```
[1] 17
```

```r
ncol(x)
```

```
[1] 4
```

```r
# using dim() which provides both information
dim(x)
```

```
[1] 17  4
```

There are 17 rows and 5 columns in the data frame (after setting the correct row names)

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?

```r
x <- read.csv(url, row.names=1)
```

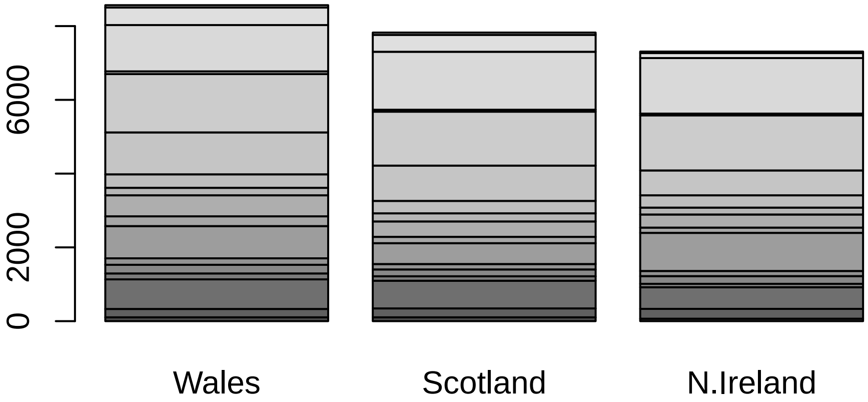
```r
rownames(x) <- x[,1]
x <- x[,-1]
head(x)
```

```
    Wales Scotland N.Ireland
105   103      103        66
245   227      242       267
685   803      750       586
147   160      122        93
193   235      184       209
156   175      147       139
```

I prefer the first method where it is just one line of code to correct the row names. It is more seamless and effortless. And if the second code block is run multiple times, it becomes an error as it removes each column with each run.

Now we can generate some basic visualizations. We need to make x as a matrix to be able to plot it

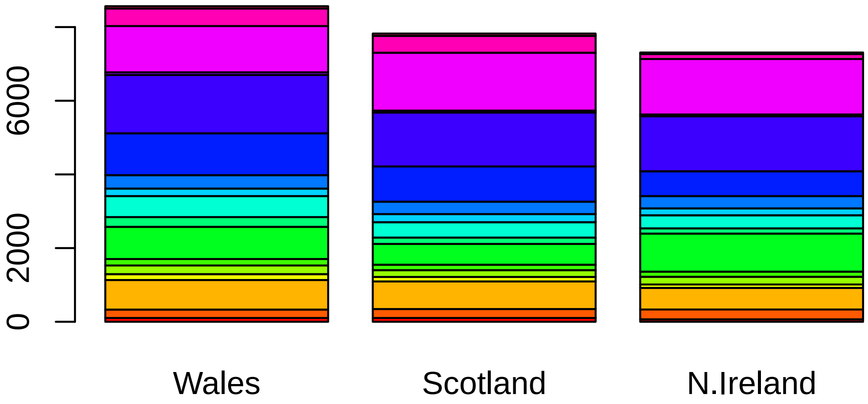
```
barplot(as.matrix(x))
```



```
rainbow(nrow(x))
```

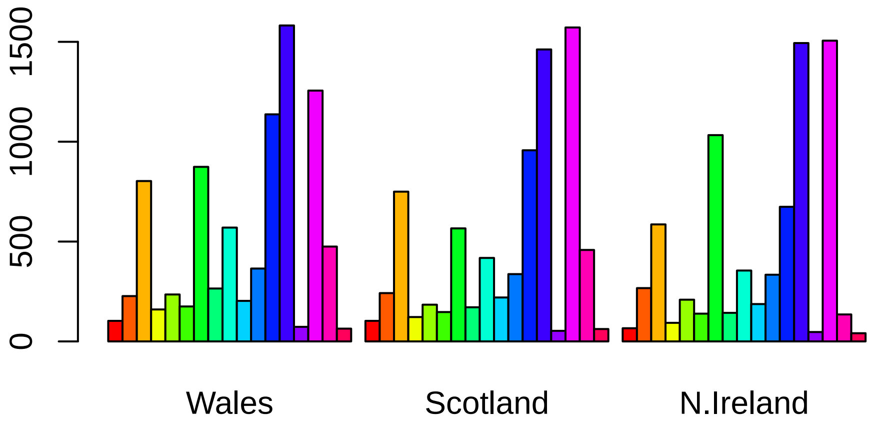
```
 [1] "#FF0000" "#FF5A00" "#FFB400" "#F0FF00" "#96FF00" "#3CFF00" "#00FF1E"
 [8] "#00FF78" "#00FFD2" "#00D2FF" "#0078FF" "#001EFF" "#3C00FF" "#9600FF"
[15] "#F000FF" "#FF00B4" "#FF005A"
```

```
# combining - giving color to the plot
barplot(as.matrix(x), col=rainbow(nrow(x)))
```



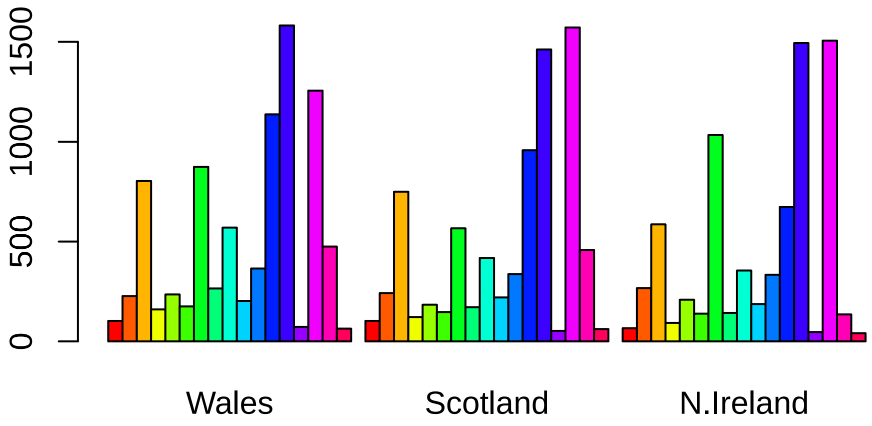
Lets refine our barplot

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

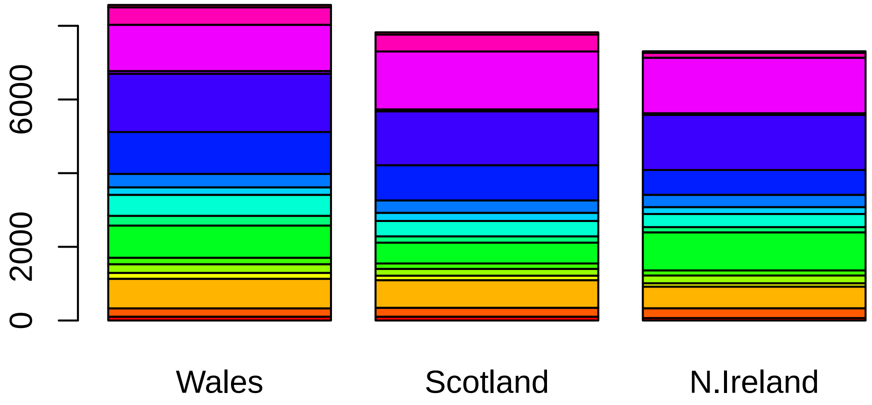


Q3. Changing what optional argument in the above `barplot()` function results in the following plot?

```
# beside to either T or F
barplot(as.matrix(x), col=rainbow(nrow(x)), beside = T)
```



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



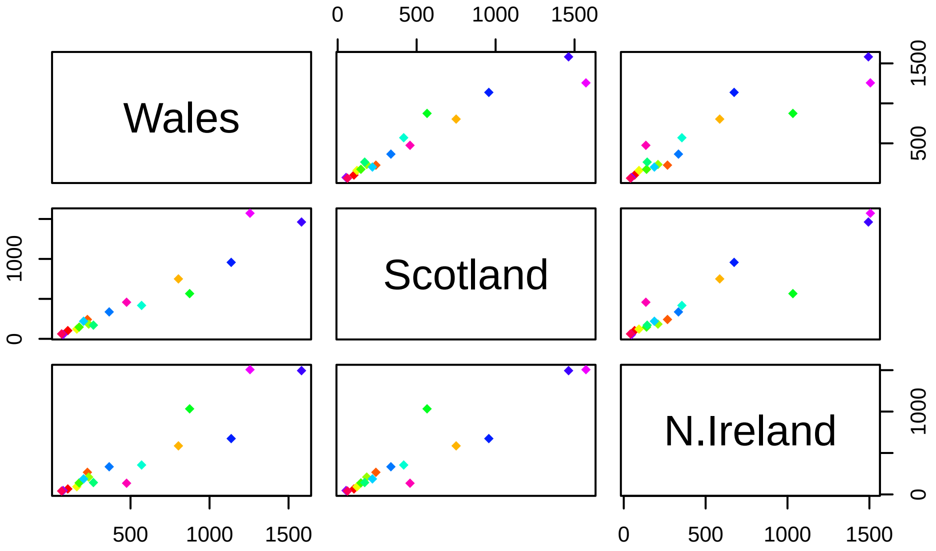
Adding the argument (`beside = T`) to `barplot(as.matrix(x), col=rainbow(nrow(x)))` will result in the data to be stacked 'beside' each other. Removing that would result in the barplot with stacked data.

Q4. 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?

If a point were to be shifted more towards one of the locations, that means that data point belongs to that area. Though the figure is difficult to discern and analyze.

Other visualizations that can be useful:

```
pairs(x, col=rainbow(nrow(x)), pch=18)
```



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

N. Ireland consumes more fresh potatoes and soft drinks and less of the other foods hence its location at the bottom of the figure and the subsequent data points near it.

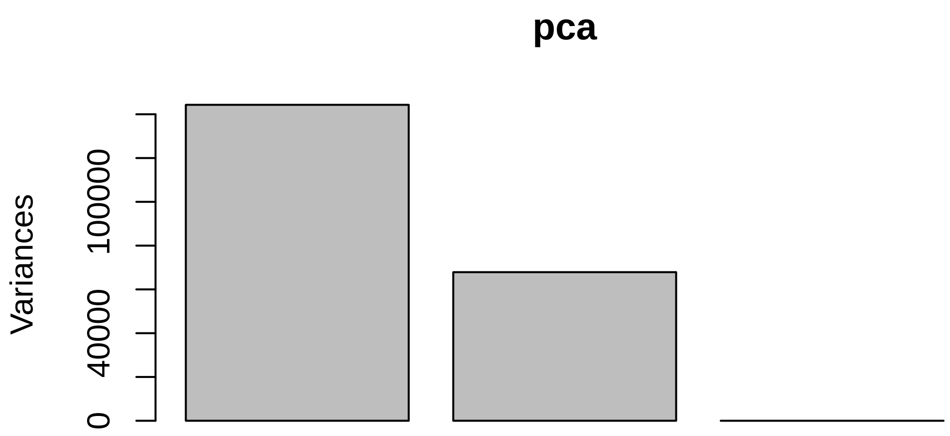
Lets apply PCA. For that, we need to use the command `prcomp()`. This function expects the transpose of our data

```
# t flips the rows and columns
# transpose_matrix <- t(x)
# pca <- prcomp(transpose_matrix)
pca <- prcomp(t(x))

summary(pca)
```

```
Importance of components:
                            PC1      PC2       PC3
Standard deviation     379.8991 260.5533 1.515e-13
Proportion of Variance   0.6801   0.3199 0.000e+00
Cumulative Proportion    0.6801   1.0000 1.000e+00
```

Lets plot the PCA results:

```
plot(pca)
```



We need to access the results of the PCA analysis

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

We can explore the `pca$x` dataframe:

(all 4 components, we can now place 2 in x axis and 2 in y axis)

```
pca$x
```

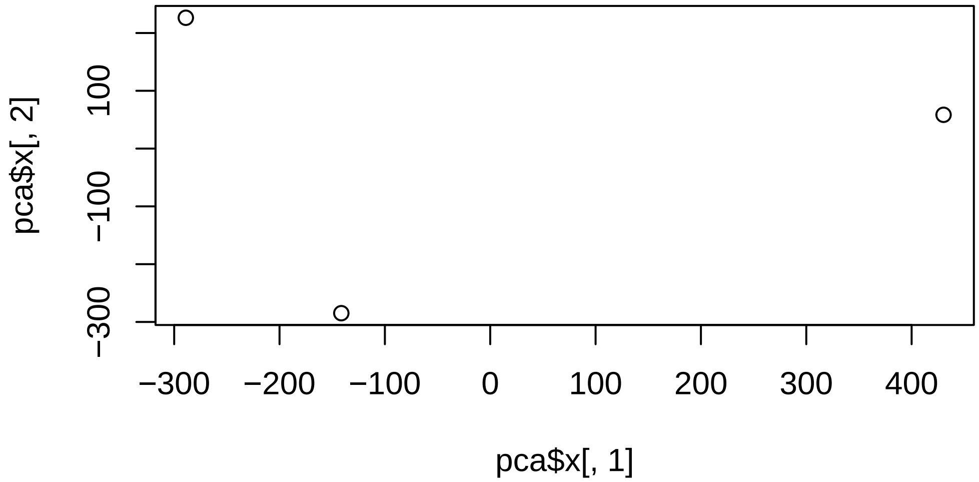
```
                PC1         PC2           PC3
Wales     -288.9534  226.36855  2.296774e-14
Scotland  -141.3603 -284.81172  4.517428e-13
N.Ireland  430.3137   58.44317 -1.407069e-13
```

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

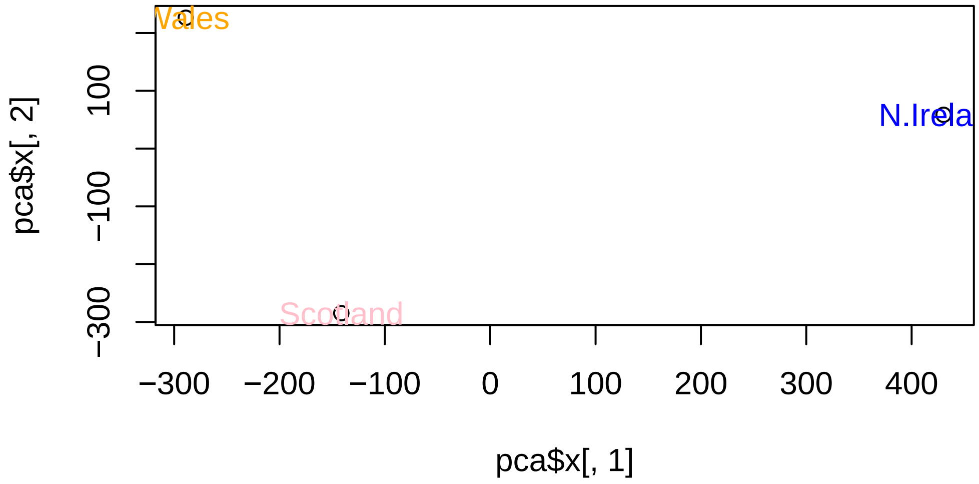
Q8. Customize your plot so that the colors of the country names match the colors in our UK and Ireland map and table at start of this document.

Plotting:

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



```
# overlay country names and adding colors
plot(pca$x[,1], pca$x[,2] )
colors_countries <- c('orange', 'pink', 'blue', 'green')
text(x=pca$x[,1], y=pca$x[,2], colnames(x), col=colors_countries)
```



Plot of variances and loading scores

```
# calculating how much variation each PC accounts for
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

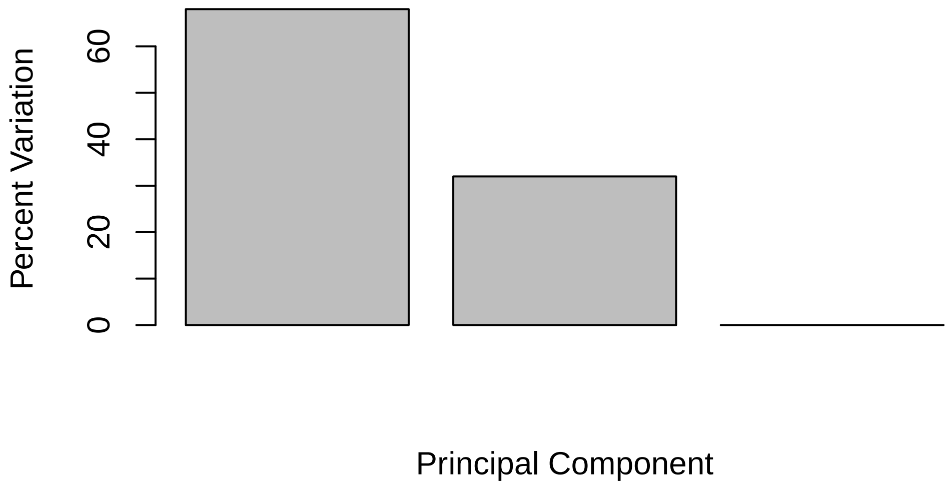
```
[1] 68 32  0
```

```
z <- summary(pca)
z$importance
```

```
                             PC1       PC2          PC3
Standard deviation     379.89908 260.55330 1.515317e-13
Proportion of Variance   0.68009   0.31991 0.000000e+00
Cumulative Proportion    0.68009   1.00000 1.000000e+00
```

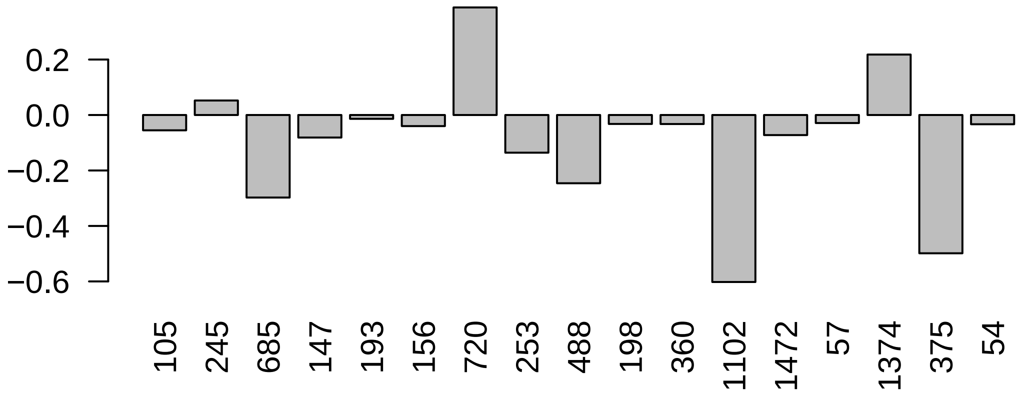
Plotting:

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



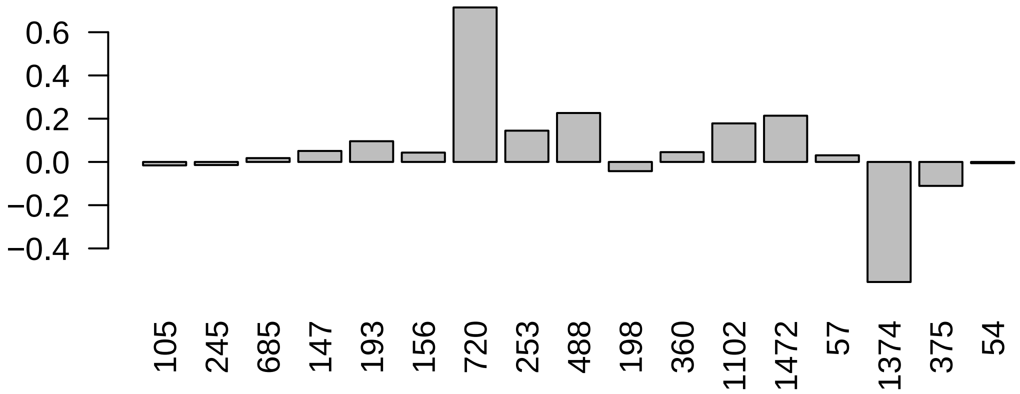
Making it into a biplot:

```
# Focusing 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 prominently and what does PC2 mainly tell us about?

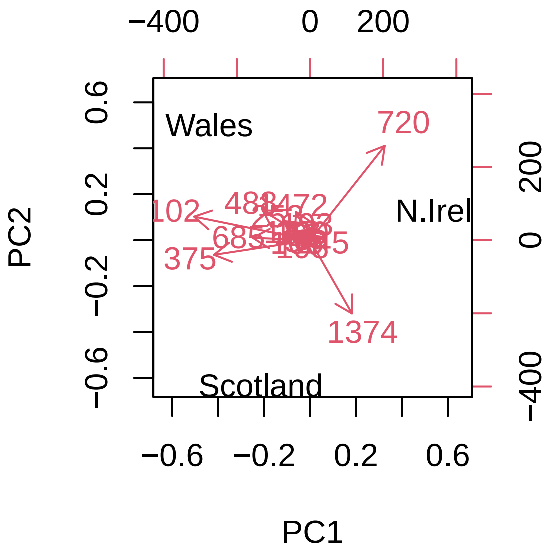
```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )

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



Fresh potatoes and soft drinks are the two main food groups featured prominently. PC2 would mainly tell us about the push of N. Ireland to the right positive side of the plot and the push of Scotland to the left negative side for soft drinks.

Biplots:

```
biplot(pca)
```



PCA of RNA-seq dataset

First step as always is to load the data:

```r
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?

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

```
[1] 100  10
```

There are 100 genes, and 10 samples.

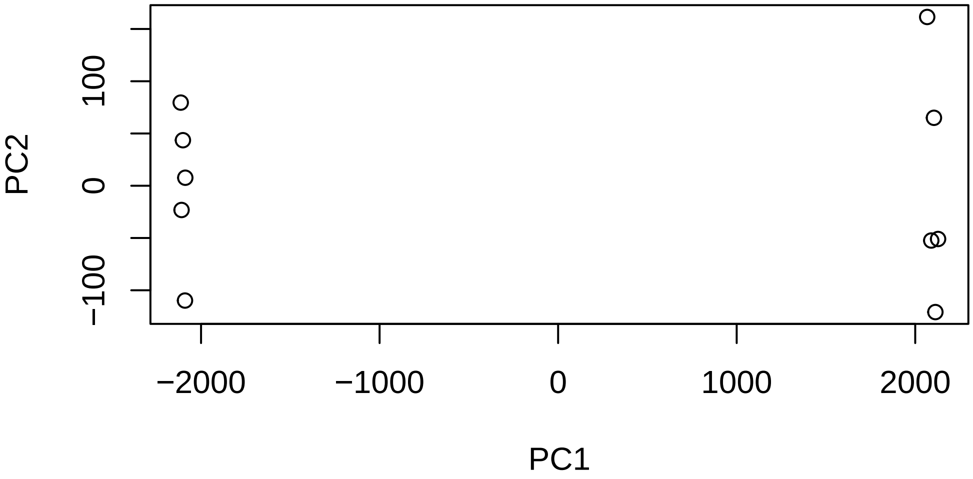
Now lets apply PCA:

```
pca_rna <- prcomp(t(rna.data))
summary(pca_rna)
```

```
Importance of components:
                             PC1     PC2      PC3      PC4      PC5      PC6
Standard deviation     2214.2633 88.9209 84.33908 77.74094 69.66341 67.78516
Proportion of Variance    0.9917  0.0016  0.00144  0.00122  0.00098  0.00093
Cumulative Proportion     0.9917  0.9933  0.99471  0.99593  0.99691  0.99784
                            PC7      PC8      PC9      PC10
Standard deviation     65.29428 59.90981 53.20803 3.142e-13
Proportion of Variance  0.00086  0.00073  0.00057 0.000e+00
Cumulative Proportion   0.99870  0.99943  1.00000 1.000e+00
```

Lets plot the principal components 1 and 2

```
plot(pca_rna$x[,1], pca_rna$x[,2], xlab='PC1', ylab='PC2')
```



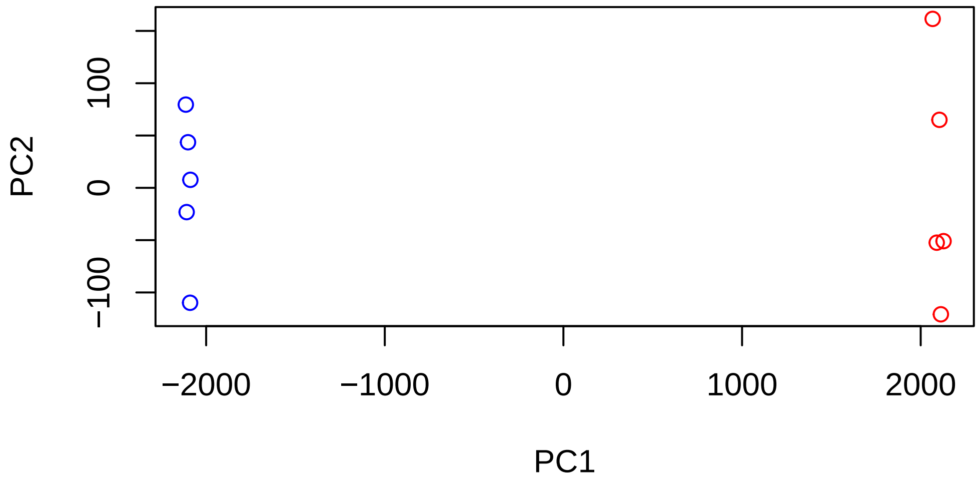
```
# checking the names
colnames(rna.data)
```

```
[1] "wt1" "wt2" "wt3" "wt4" "wt5" "ko1" "ko2" "ko3" "ko4" "ko5"
```

```
# generating a vector that will color the sample
cols_samples <- c(rep('blue', 5), rep('red', 5))
cols_samples
```

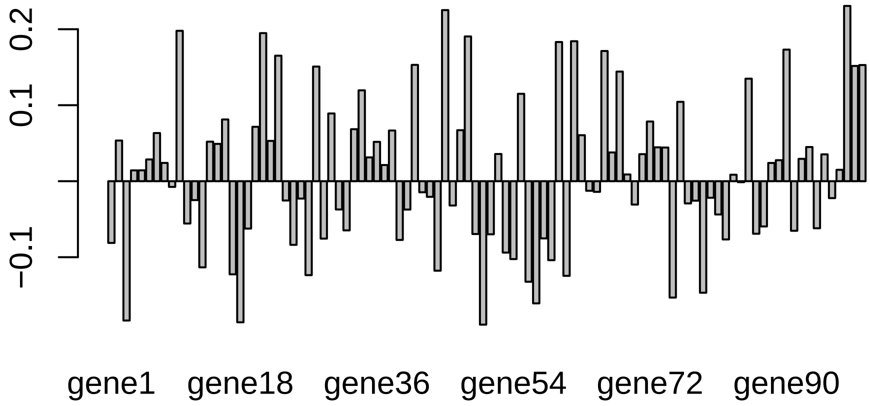
```
[1] "blue" "blue" "blue" "blue" "blue" "red"  "red"  "red"  "red"  "red"
```

```
# applying color to the plot
plot(pca_rna$x[,1], pca_rna$x[,2], xlab='PC1', ylab='PC2', col=cols_samples)
```



Identifying which gene is contributing the most

```
barplot(pca_rna$rotation[,1])
```



```r
# identifying under- or overexpression
sort(pca_rna$rotation[,1])
```

```
      gene50       gene18        gene3       gene57       gene75       gene79 
-0.188796985 -0.185668500 -0.183374164 -0.160771014 -0.153164404 -0.146803635 
      gene56       gene61       gene27       gene17       gene44       gene13 
-0.132330117 -0.124572881 -0.123615228 -0.122536548 -0.117808971 -0.113357525 
      gene59       gene54       gene53       gene25        gene1       gene39 
-0.103935563 -0.102503320 -0.093979884 -0.083761992 -0.081247810 -0.077306742 
      gene82       gene29       gene58       gene51       gene49       gene86 
-0.076658760 -0.075605635 -0.075274651 -0.069855142 -0.069530208 -0.069165267 
      gene91       gene32       gene19       gene94       gene87       gene11 
-0.065288752 -0.064721235 -0.062411218 -0.061938300 -0.059547317 -0.055698801 
      gene81       gene40       gene31       gene46       gene70       gene77 
-0.043780416 -0.037323670 -0.037219970 -0.031990529 -0.030784982 -0.029225446 
      gene78       gene24       gene12       gene26       gene96       gene80 
-0.025639741 -0.025407507 -0.024870802 -0.022868107 -0.022293151 -0.021824860 
      gene43       gene42       gene65       gene64        gene9       gene84 
-0.020617052 -0.014550791 -0.014052839 -0.012639567 -0.007495075 -0.001289937 
      gene83       gene69        gene4        gene5       gene97       gene37 
 0.008504287  0.008871890  0.014242602  0.014303808  0.014994546  0.021280555 
      gene88        gene8       gene89        gene6       gene92       gene35 
```

```
0.024015925  0.024026657  0.027652967  0.028634131  0.029394259  0.031349942
     gene95       gene71       gene52       gene67       gene74       gene73
0.035342407  0.035589259  0.035802086  0.037840851  0.044286948  0.044581700
     gene93       gene15       gene36       gene14       gene22        gene2
0.044940861  0.049090676  0.051765605  0.052004194  0.053013523  0.053465569
     gene63        gene7       gene38       gene47       gene33       gene20
0.060529157  0.063389255  0.066665407  0.067141911  0.068437703  0.071571203
     gene72       gene16       gene30       gene76       gene55       gene34
0.078551648  0.081254592  0.089150461  0.104435777  0.114988217  0.119604059
     gene85       gene68       gene28       gene99      gene100       gene41
0.134907896  0.144227333  0.150812015  0.151678253  0.152877246  0.153077075
     gene23       gene66       gene90       gene60       gene62       gene48
0.165155192  0.171311307  0.173156806  0.183139926  0.184203008  0.190495289
     gene21       gene10       gene45       gene98
0.194884023  0.197905454  0.225149201  0.230633225
```