Class 7: Machine Learning I

Audrey Nguyen

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))
x
```

```
              x         y
 [1,] -1.8528805  2.7783032
 [2,] -2.9269715  2.5506729
 [3,] -2.8032023  5.8059108
 [4,] -1.1424205  2.8209750
 [5,] -3.0843831  1.9859521
 [6,] -2.5135990  1.6171640
 [7,] -3.4733960  1.8619089
 [8,] -2.4456156  4.0791860
 [9,] -3.8008968  1.6618588
[10,] -2.2051226  3.6959530
[11,] -4.1405486  3.9055803
[12,] -3.6978199  2.1948239
[13,] -3.6109422  3.6094984
[14,] -2.8417313  3.6791965
[15,] -2.7114965  1.6806124
[16,] -3.7585458  4.0160263
[17,] -2.0322426  0.3978394
[18,] -3.6192585  2.9452341
[19,] -2.8804125  2.2707954
[20,] -2.7260425  1.3306288
[21,] -2.5583816  2.3755967
```

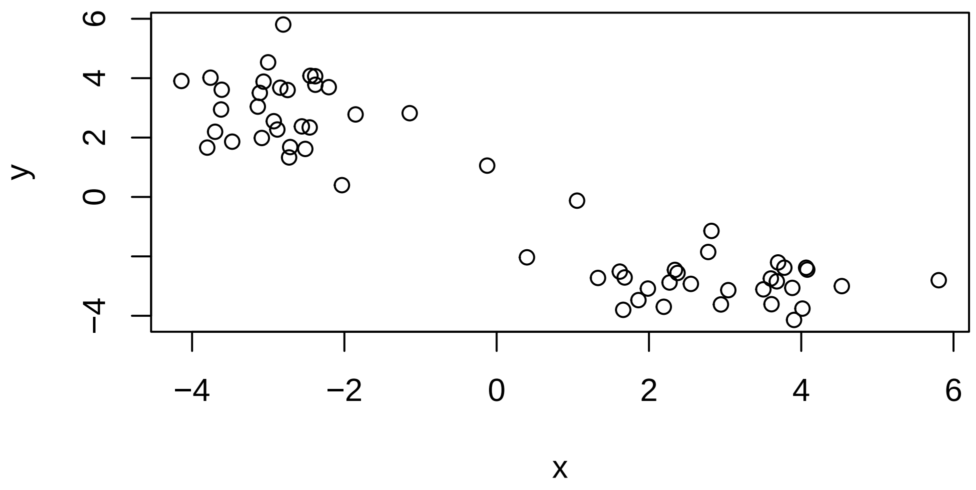
```
[22,] -0.1245820  1.0559112
[23,] -2.7457598  3.6009875
[24,] -2.3835482  4.0643486
[25,] -2.3811922  3.7782588
[26,] -3.0613761  3.8827984
[27,] -3.0016676  4.5327120
[28,] -3.1104324  3.5029603
[29,] -3.1372732  3.0418920
[30,] -2.4555094  2.3412013
[31,]  2.3412013 -2.4555094
[32,]  3.0418920 -3.1372732
[33,]  3.5029603 -3.1104324
[34,]  4.5327120 -3.0016676
[35,]  3.8827984 -3.0613761
[36,]  3.7782588 -2.3811922
[37,]  4.0643486 -2.3835482
[38,]  3.6009875 -2.7457598
[39,]  1.0559112 -0.1245820
[40,]  2.3755967 -2.5583816
[41,]  1.3306288 -2.7260425
[42,]  2.2707954 -2.8804125
[43,]  2.9452341 -3.6192585
[44,]  0.3978394 -2.0322426
[45,]  4.0160263 -3.7585458
[46,]  1.6806124 -2.7114965
[47,]  3.6791965 -2.8417313
[48,]  3.6094984 -3.6109422
[49,]  2.1948239 -3.6978199
[50,]  3.9055803 -4.1405486
[51,]  3.6959530 -2.2051226
[52,]  1.6618588 -3.8008968
[53,]  4.0791860 -2.4456156
[54,]  1.8619089 -3.4733960
[55,]  1.6171640 -2.5135990
[56,]  1.9859521 -3.0843831
[57,]  2.8209750 -1.1424205
[58,]  5.8059108 -2.8032023
[59,]  2.5506729 -2.9269715
[60,]  2.7783032 -1.8528805
```

```
head(x)
```

```
             x        y
[1,] -1.852881 2.778303
[2,] -2.926972 2.550673
[3,] -2.803202 5.805911
[4,] -1.142420 2.820975
[5,] -3.084383 1.985952
[6,] -2.513599 1.617164
```

Quick plot of x to see the two graphs 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 -2.774242  2.902160
2  2.902160 -2.774242

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] 60.00383 60.00383
 (between_SS / total_SS =  89.0 %)

Available components:

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

> Q. How many points are in each cluster?

```
km$size
```

```
[1] 30 30
```

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

```
km$cluster
```

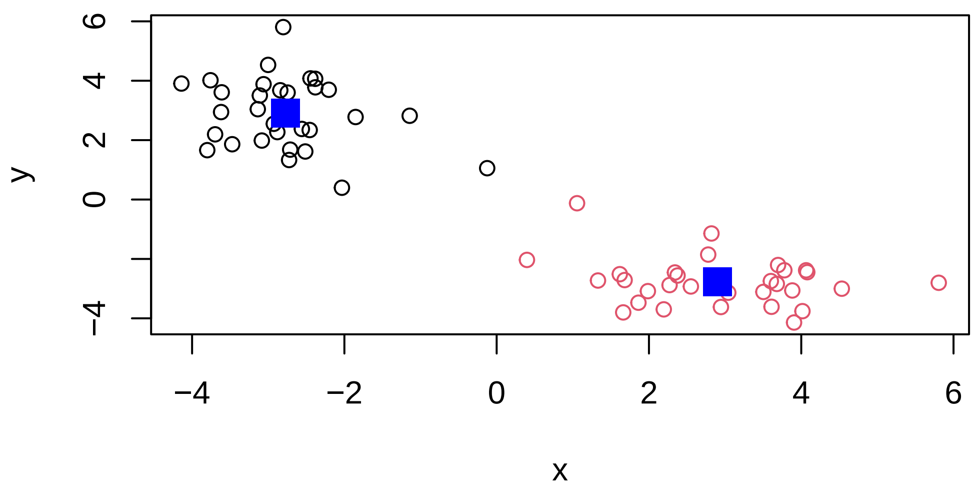
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
km$centers
```

```
          x         y
1 -2.774242  2.902160
2  2.902160 -2.774242
```

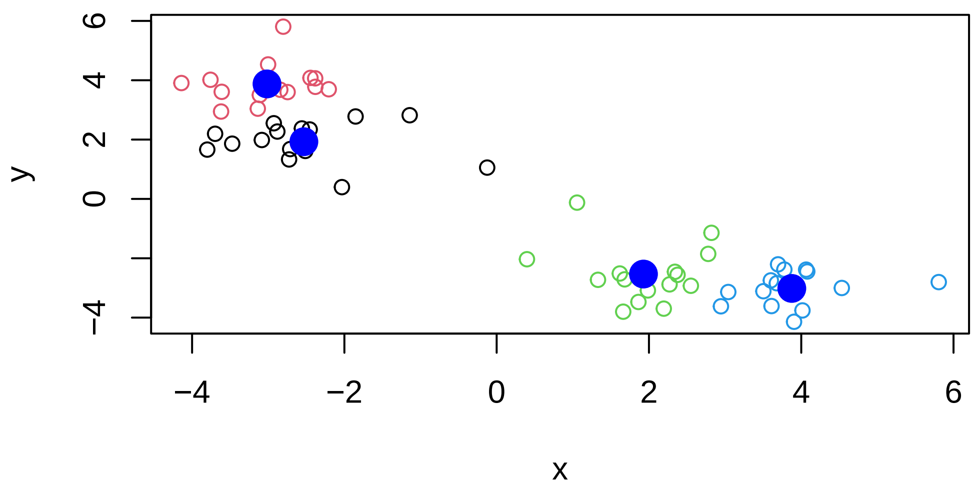
> Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

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



Play with kmeans and ask for different number of clusters

```r
km <- kmeans(x, centers = 4, nstart = 20)
plot(x, col = km$cluster)
points(km$centers, col = "blue", pch = 16, cex = 2)
```



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.

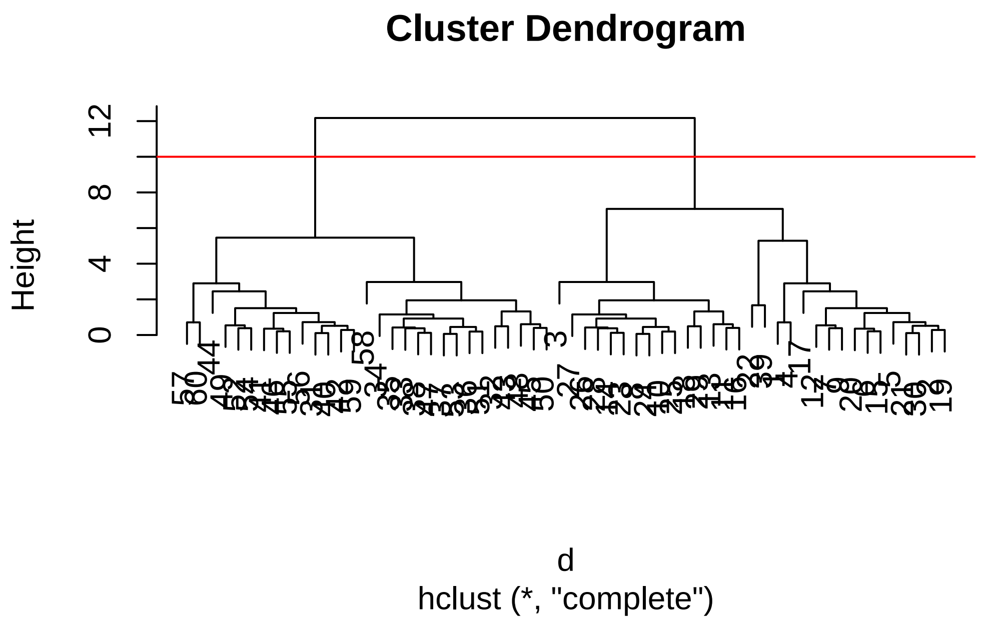
```
d <- dist(x)
hc <- hclust(d)
hc
```

```
Call:
hclust(d = d)

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

There is a plot message 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 or branches with all the members of a given cluster residing on the same cut branch. The function to do this is `cutree()`.

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

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

It is often helpful to use the `k =` argument rather than the `h =` height of cutting with `cutree()`. This will cut the tree to yield the number of clusters you want.

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

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

#Principal Component Analysis (PCA)

The base R function for PCA is called `prcomp()`. Let's play with some 17D data (a very small dataset) and see how PCA can help.

#PCA of UK food data

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

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

You can use the `dim()` function to return the number of rows and columns or `nrow()` to get rows and `ncol()` to get columns separately.

```r
dim(x)
```

```
[1] 17  5
```

```r
nrow(x)
```

```
[1] 17
```

```r
ncol(x)
```

```
[1] 5
```

```r
## Preview the first 6 rows
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
```

It appears that the row-names are incorrectly set as the first column of our **x** data frame (rather than set as proper row-names). We want 4 columns for the 4 countries instead. We can fix this with the function `rownames()` to the first column and then remove the troublesome first column (with the -1 column index):

```r
# 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
```

```r
# checking the dimensions again
dim(x)
```

```
[1] 17  4
```

An alternative approach to setting the correct row-names in this case would be to read the data file again and set the `row.names` argument of `read.csv()` to be the 1st column.

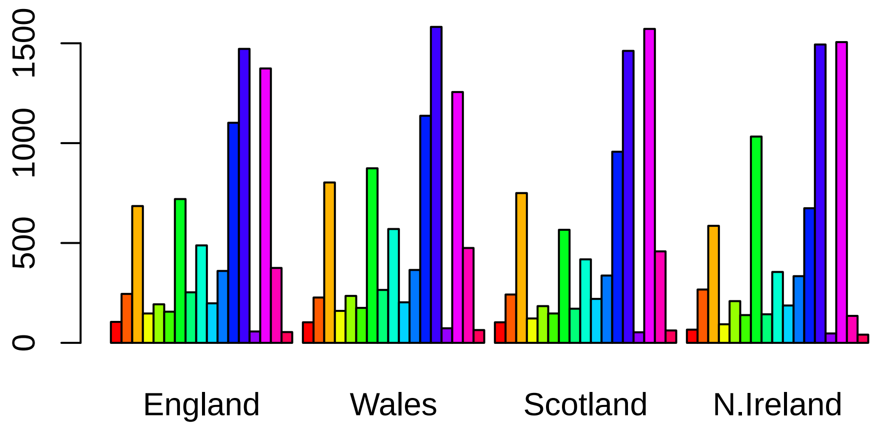
```r
x <- read.csv(url, row.names = 1)
head(x)
```

```
               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 2nd approach, because it's more concise. The 2nd approach is more robust, because it can be run multiple times without messing up the dimensions.

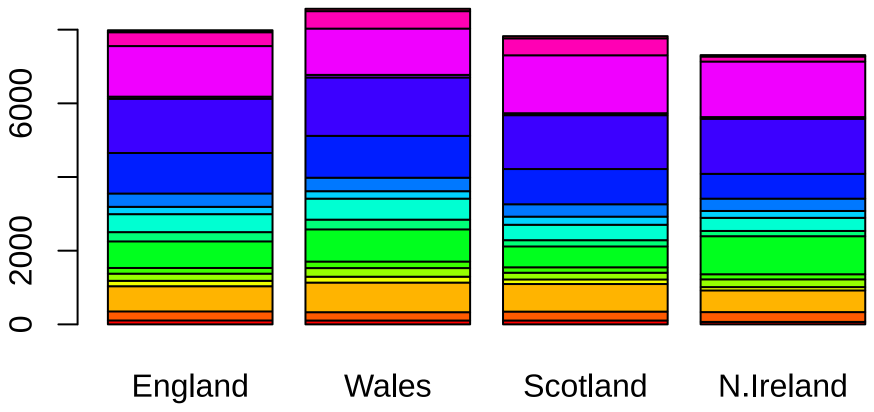
```r
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



Q3. Changing what optional argument in the above barplot() function results in the following plot (bars are stacked on each other)?

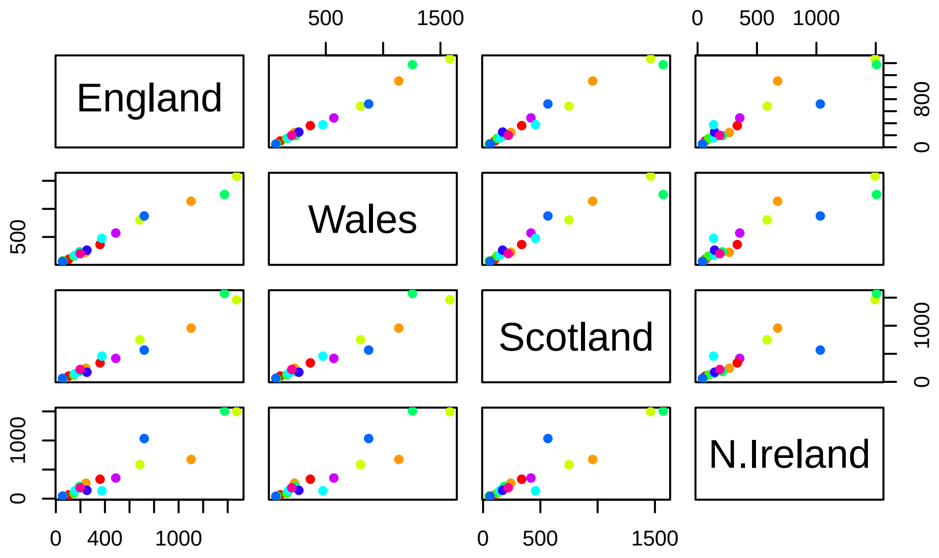
Setting `beside=FALSE` in the `barplot()` code would stack the bars.

```
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)
```



You can compare the countries from switching the axes based on which pair you're looking at.

Given points on the diagonal means that they're the same value as the other food categories from other countries.

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

Comparing to other countries, there are less points on the diagonal, which means that N. Ireland has more distinct values for the food categories.

```r
# Use the prcomp() PCA function
pca <- prcomp(t(x))
summary(pca)
```

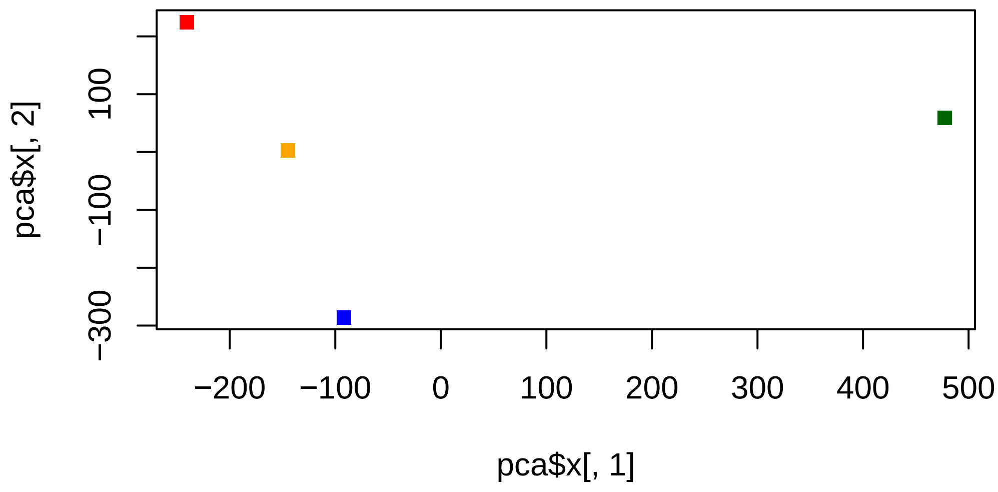
```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 4.189e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

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

```r
pca$x
```

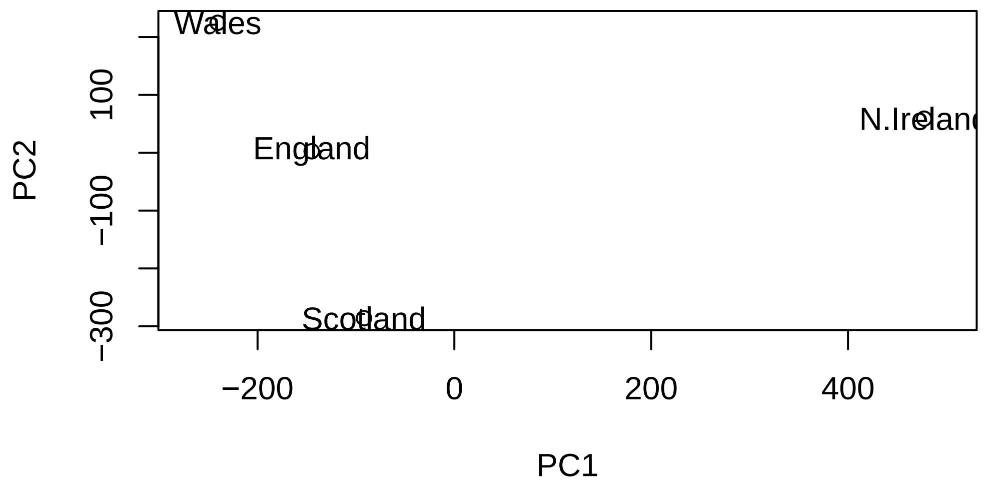
```
                PC1         PC2         PC3           PC4
England   -144.99315    2.532999 -105.768945  2.842865e-14
Wales     -240.52915  224.646925   56.475555  7.804382e-13
Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

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



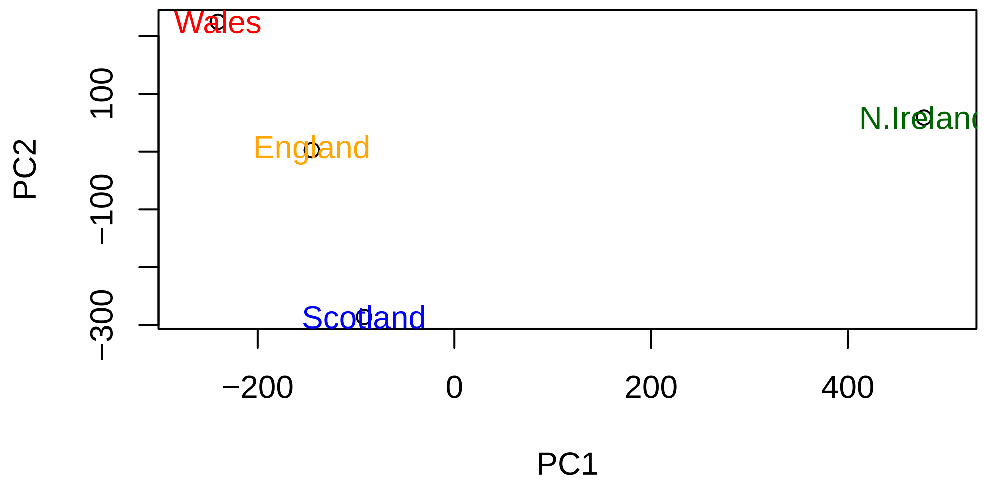
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# Plot PC1 vs PC2
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. 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], col = c("orange", "red", "blue", "darkgreen"), colnames(x))
```



Below we can use the square of pca$sdev , which stands for “standard deviation”, to calculate how much variation in the original data each PC accounts for.

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

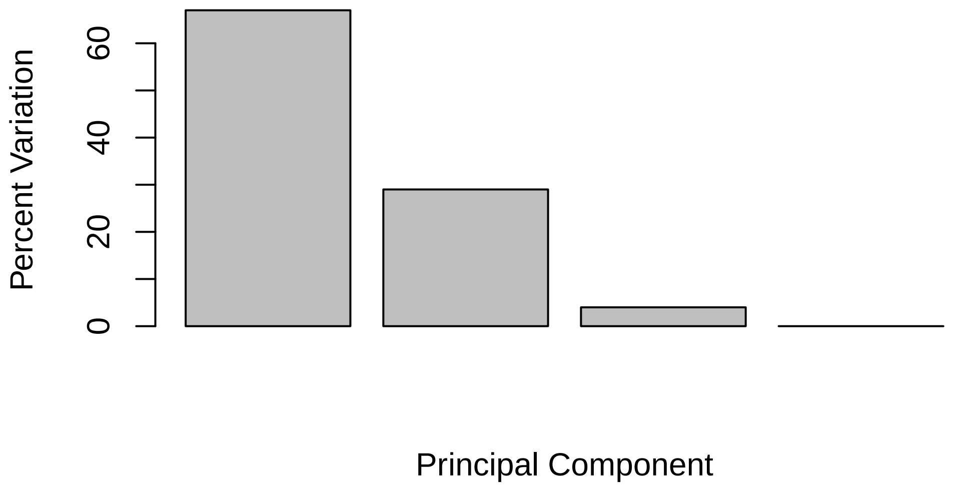
```
[1] 67 29  4  0
```

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

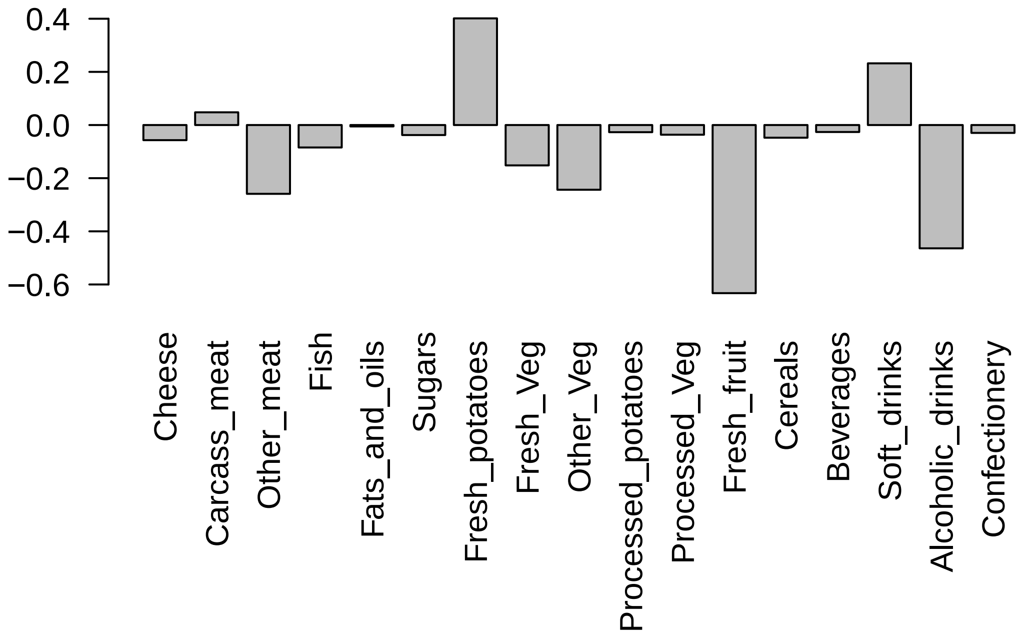
```
                             PC1       PC2      PC3          PC4
Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

This information can be summarized in a plot of the variances (eigenvalues) with respect to the principal component number (eigenvector number), which is given below.

```
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 )
```


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