Class 7: Clustering and PCA

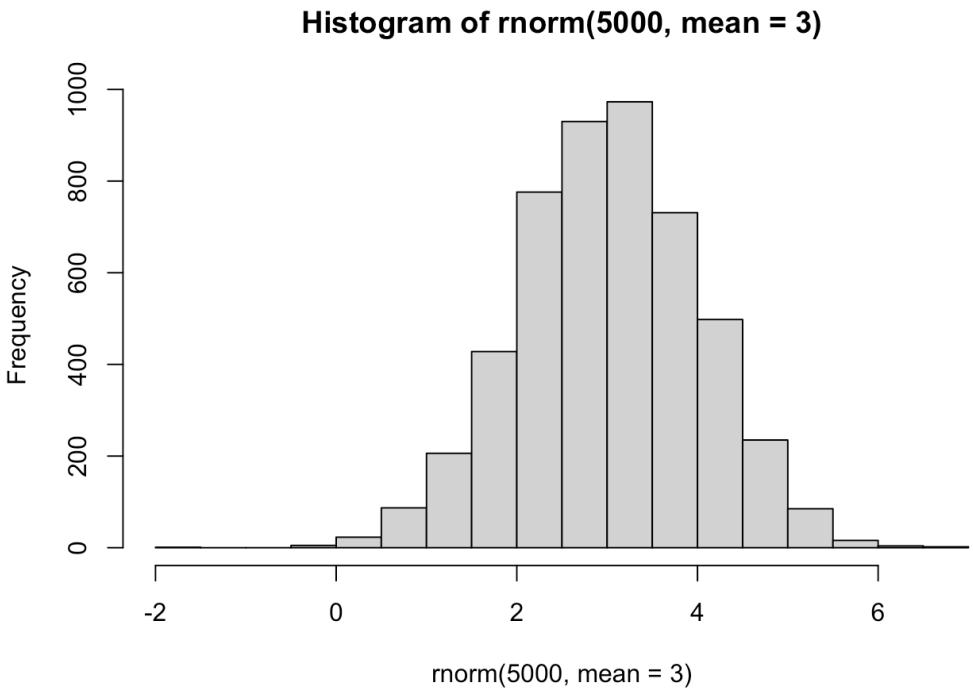
AUTHOR
Loreen A17059289

Clustering

First, let's make up some data to cluster so we can get a feel for these methods and how to work with them.

We can use the `rnorm()` function to get random numbers from a normal distribution around a given `mean`.

```
hist(rnorm(5000, mean = 3))
```



Let's get 30 points with a mean of 3.

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

```
 [1]  2.956726  2.564887  1.719090  3.030295  3.728052
```

```
3.125835  2.567224
 [8]  3.592008  4.620241  1.240388  1.910028  4.173773
2.203513  3.365460
[15]  4.392148  4.724032  1.922579  3.716244  4.120152
2.887242  2.840388
[22]  4.812284  3.749495  4.604146  2.515375  3.683936
2.617974  3.609495
[29]  3.264616  3.252820 -2.718359 -3.221281 -2.021248
-2.515707 -2.892956
[36] -3.176945 -3.600650 -2.934144 -3.119099 -2.002449
-1.044093 -1.916917
[43] -3.569002 -3.015509 -3.029055 -4.253974 -2.946550
-2.017523 -3.229771
[50] -2.767928 -3.861492 -2.567927 -3.562828 -3.452052
-3.771082 -3.692328
[57] -2.569388 -2.392296 -4.627242 -2.294861
```

Trying `rev()`:

```
rev(c(1, 2, 3, 4, 5))
```

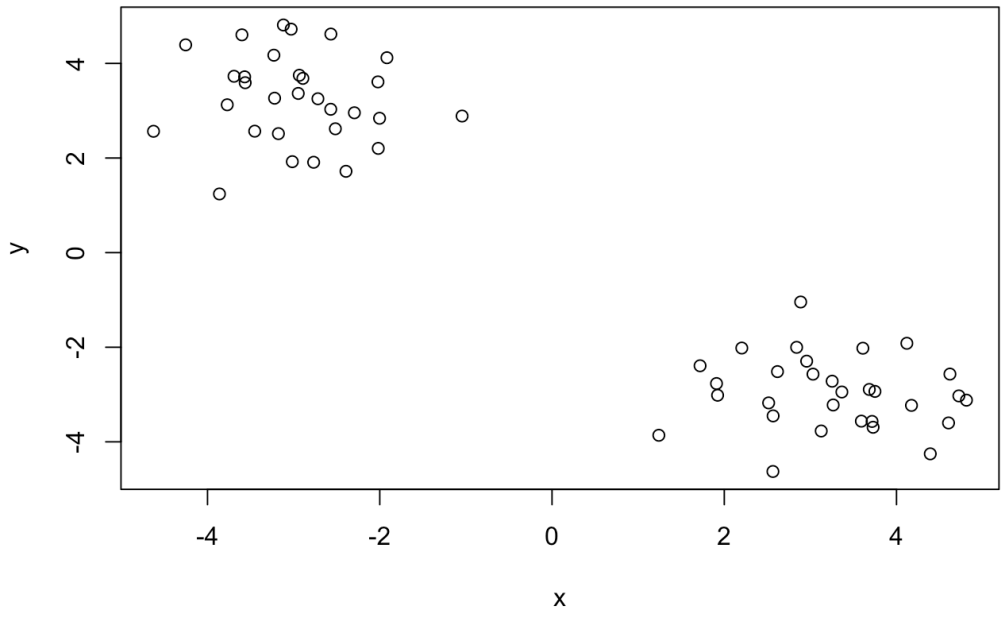
```
[1] 5 4 3 2 1
```

```
cbind(c(1, 2, 3, 4, 5), rev(c(1,2,3,4,5)))
```

```
     [,1] [,2]
[1,]    1    5
[2,]    2    4
[3,]    3    3
[4,]    4    2
[5,]    5    1
```

Putting two together(code above the one above this):

```
x <- cbind(x=tmp, y=rev(tmp))
plot(x)
```



K-means clustering.

Very popular clustering method that we can use with the `kmeans()` function in base R.

```
# 2 clusters:
# Cluster vector says which cluster they belong to.
# Available components is the stuff needed to work with this ar
km <- kmeans(x, centers = 2)
km
```

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

Cluster means:
          x         y
1 -2.959489  3.250348
2  3.250348 -2.959489

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 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] 42.62513 42.62513
 (between_SS / total_SS =  93.1 %)

Available components:

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

```
# Questions:
#Cluster size:
km$size
```

```
[1] 30 30
```

```
#Cluster assignment/membership:
km$cluster
```

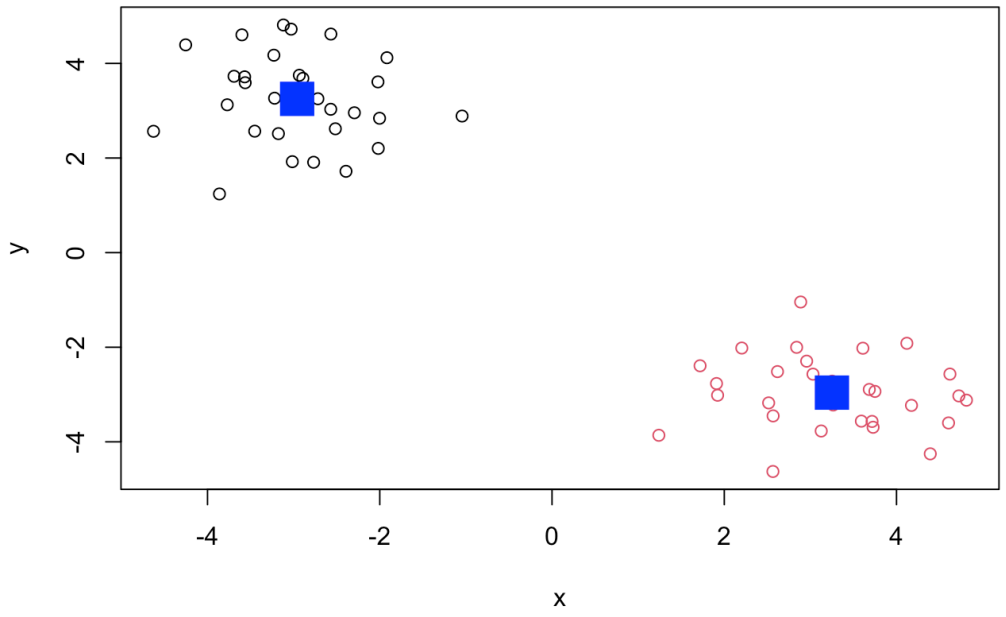
```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
km$centers
```

```
          x         y
1 -2.959489  3.250348
2  3.250348 -2.959489
```

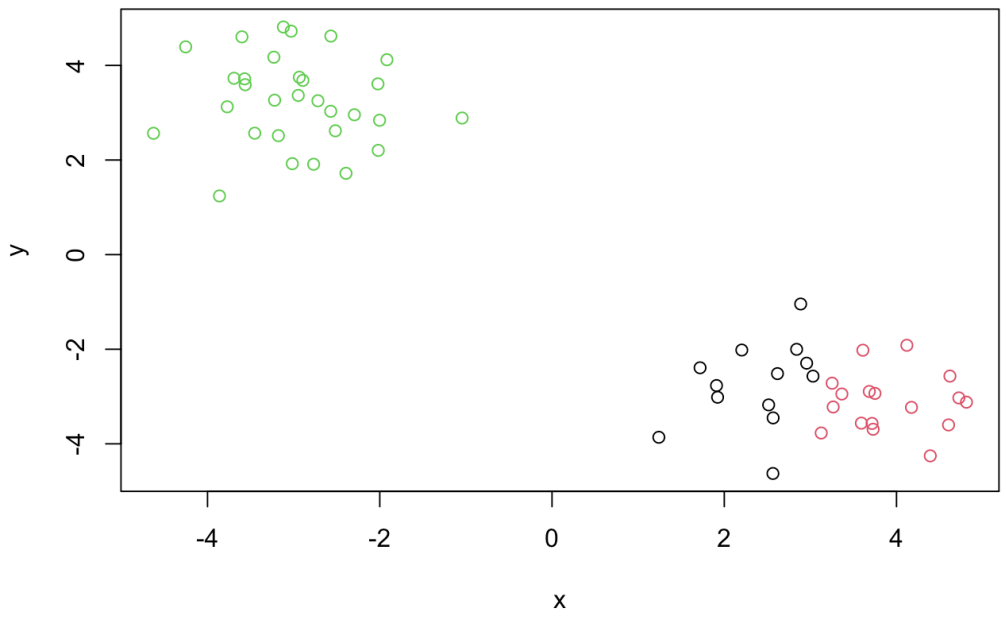
> Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```
mycols <- c(1, 5)
# col=km$cluster will split them into two colors
plot(x, col=km$cluster)
points(km$centers, col="blue", pch=15, cex=3)
```



> Q. Let's cluster into 3 groups or some `x` data and make a plot.

```
km <- kmeans(x, centers = 3)
plot(x, col=km$cluster)
```



Hierarchical Clustering

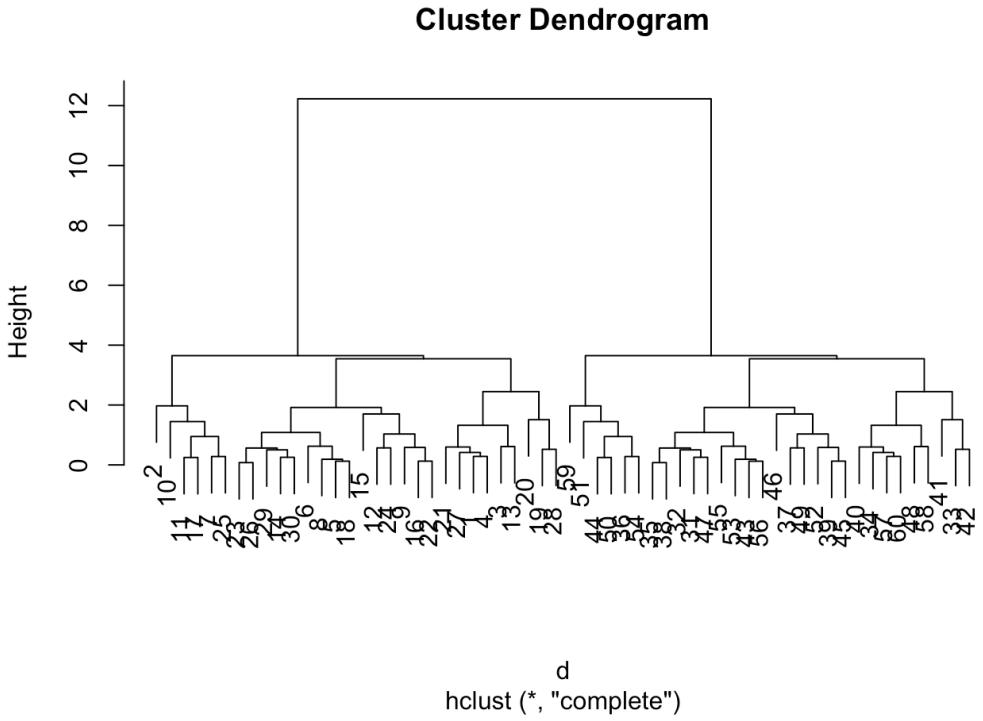
We can use the `hcluster()` function for Hierarchical Clustering. Unlike `kmeans()`, where we could just pass in our data as input, we need to give `hclust()` a "distance matrix".

We will use the `dist()` function to start with:

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

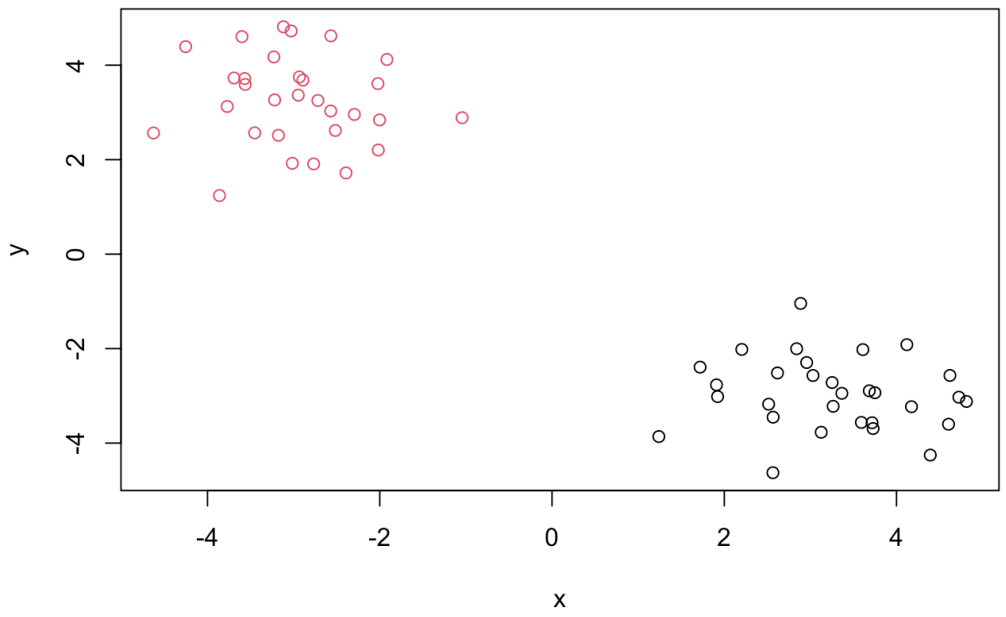
```
Call:
hclust(d = d)

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

```
plot(hc)
```



I can now "cut" my tree with the `cutree()` to yield a cluster membership vector.

```
grps <- cutree(hc, h=12)
plot(x, col=grps)
```



You can also tell `cutree()` to cut where it yields "k" groups.

```
cutree(hc, 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 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)

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
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
7	Fresh_potatoes	720	874	566	1033
8	Fresh_Veg	253	265	171	143
9	Other_Veg	488	570	418	355
10	Processed_potatoes	198	203	220	187
11	Processed_Veg	360	365	337	334
12	Fresh_fruit	1102	1137	957	674
13	Cereals	1472	1582	1462	1494
14	Beverages	57	73	53	47
15	Soft_drinks	1374	1256	1572	1506
16	Alcoholic_drinks	375	475	458	135
17	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?

```
# Or you can use `dim()`:
dim(x)
```

```
[1] 17  5
```

Using `head()`:

```
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 it to get 4 cols, not 5:

```
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
```

Checking if this worked: It did!

```
nrow(x)
```

```
[1] 17
```

```
ncol(x)
```

```
[1] 4
```

2nd Approach to get 4 cols:

```
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
```

```
nrow(x)
```

```
[1] 17
```

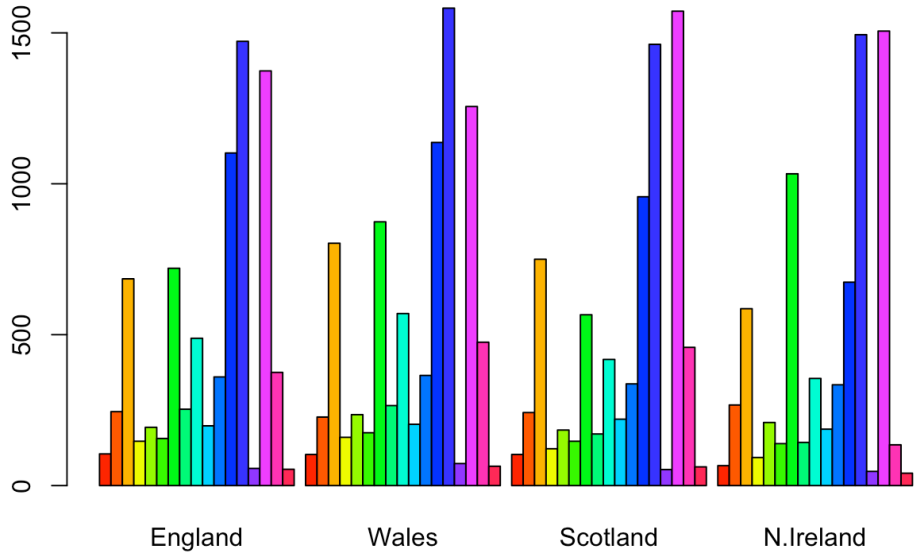
```
ncol(x)
```

```
[1] 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?

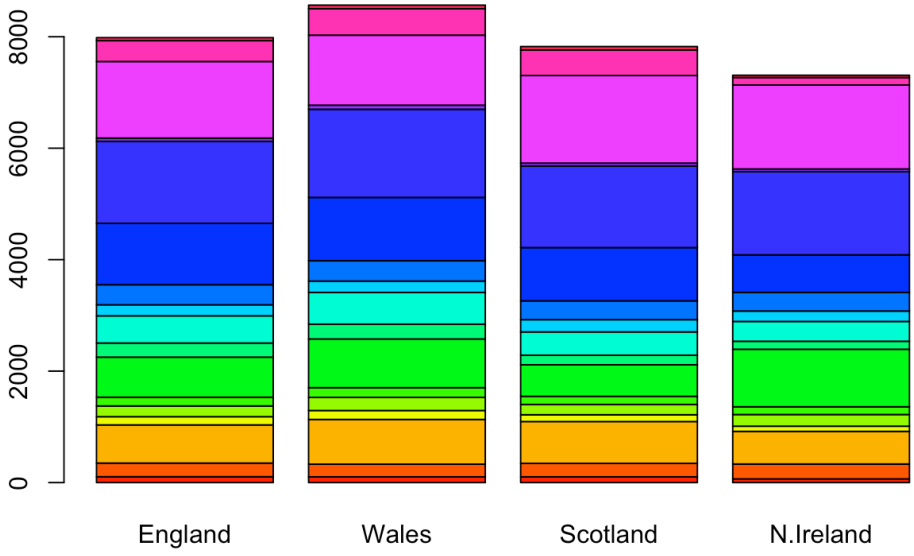
I prefer the 2nd approach using (x <- read.csv(url, row.names=1)) because the first approach keeps deleting each column after every time you run it. We don't want to keep losing the columns, we just wanted to remove 1 of them.

```
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?

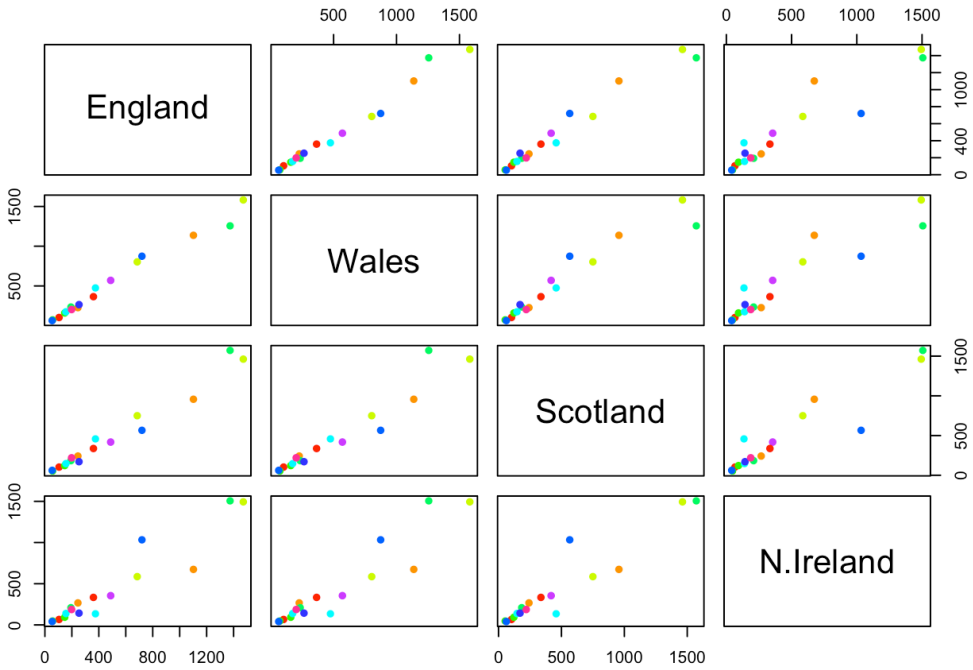
```
barplot(as.matrix(x), beside=FALSE, col=rainbow(nrow(x)))
```



> 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?

The values are equal/similar to one another if they lie on the diagonal for a given plot.

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



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

The other countries form a nicer line while N.Ireland has more wiggle. The blue point is below the diagonal so it is consumed more in N.Ireland.

PCA

```
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
```

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

The main PCA function in base R is called `prcomp()` it expects the

transpose of our data.

> Q8.

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