Class 07

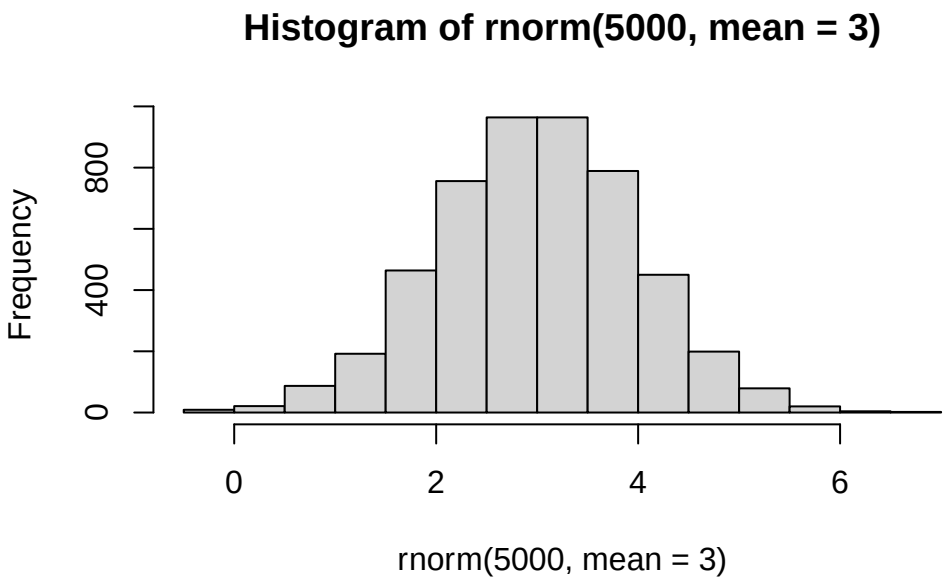
Loretta Cheng

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 give `mean`.

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



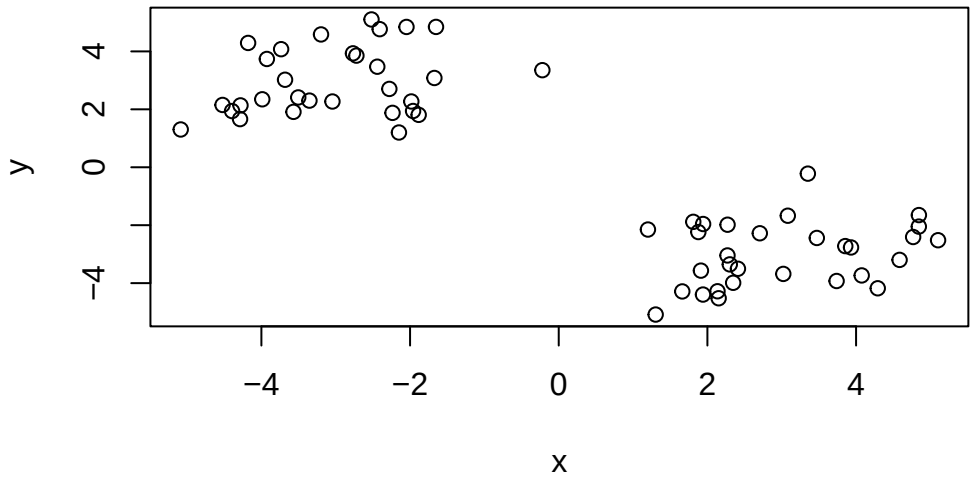
Lets get 30 points with a mean of 3 and another 3 with the means of -3.

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

```
 [1]  1.8100011  5.1032741  3.4720342  1.9422943  1.1999180  3.7381344
 [7]  2.3465482  3.8547306  2.3013702  3.0209520  2.4110056  4.5854422
[13]  1.9407033  2.7050567  1.9128765  4.7678502  3.0817099  2.2712916
[19]  2.2701894  3.9324372  4.2916276  1.3039783  1.8782885  2.1359072
[25]  2.1516195  1.6622771  4.8451943  4.8442221  4.0759935  3.3515693
[31] -0.2202102 -3.7346233 -2.0481525 -1.6508460 -4.2868979 -4.5227716
[37] -4.2811149 -2.2360442 -5.0852085 -4.1784851 -2.7680960 -3.0444616
[43] -1.9831771 -1.6729428 -2.4080646 -3.5684492 -2.2785551 -4.3946739
[49] -3.1977029 -3.5041484 -3.6816886 -3.3535187 -2.7219275 -3.9880887
[55] -3.9262875 -2.1497208 -1.9595805 -2.4410144 -2.5188388 -1.8842247
```

Put these two together:

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



K-means clustering.

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

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

```
K-means clustering with 4 clusters of sizes 30, 11, 11, 8

Cluster means:
          x         y
1  2.973617 -2.989651
2 -2.872185  4.319176
3 -3.973729  2.132493
4 -1.798057  2.280016

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 4 2 2 2 3 3 3 4
[39] 3 2 2 3 4 4 2 3 4 3 2 3 3 3 2 3 2 4 4 2 2 4

Within cluster sum of squares by cluster:
[1] 75.359997  9.317410  5.578200  6.754059
 (between_SS / total_SS =  92.0 %)

Available components:

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

- Km size

```
km$size
```

```
[1] 30 11 11  8
```

-km cluster

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

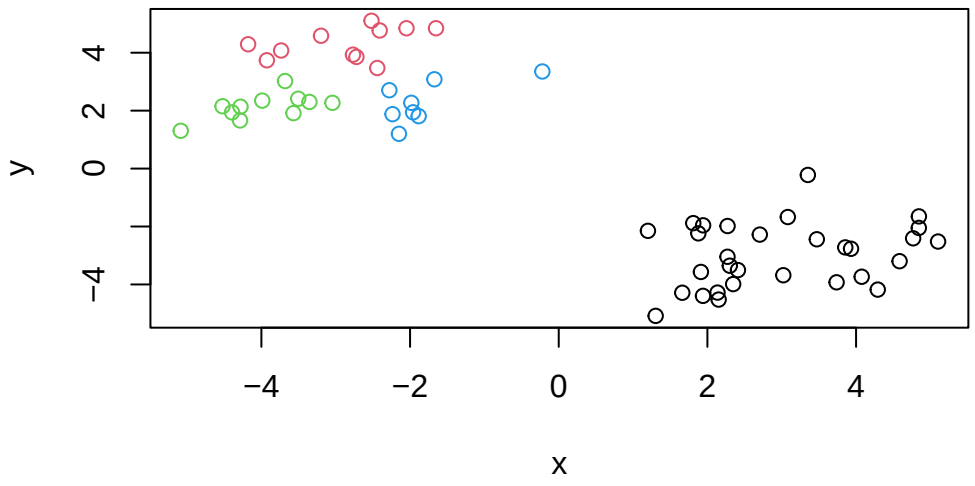
-km center

```
km$centers
```

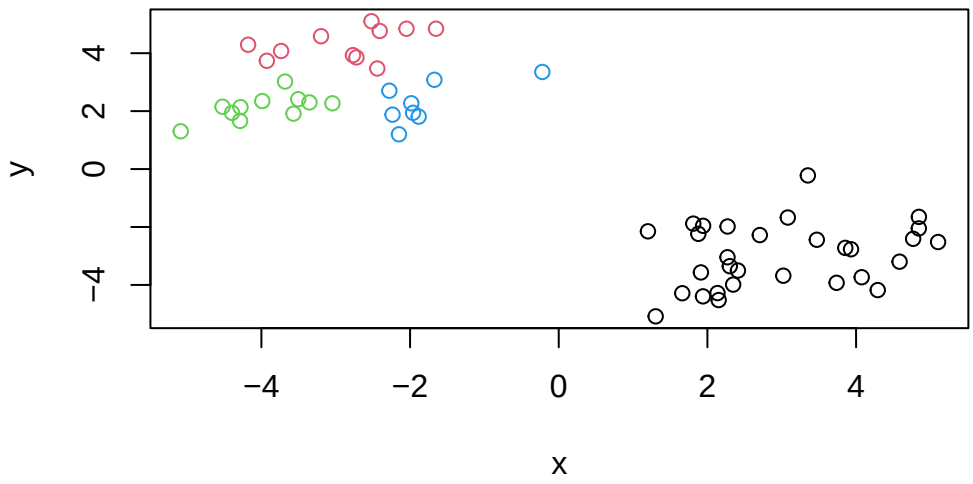
```
          x         y
1  2.973617 -2.989651
2 -2.872185  4.319176
3 -3.973729  2.132493
4 -1.798057  2.280016
```

Q. Plot x colored by the kmeans cluster assignment, adding cluster centers as blue prints

```
mycols <- c(km$cluster)
plot(x, col=mycols)
```

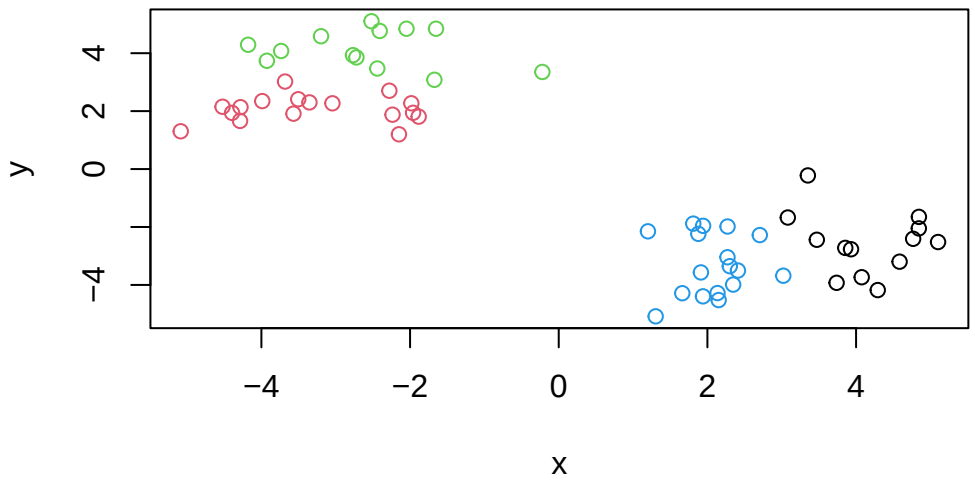



```
plot (x, col=km$cluster)
```



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

```
km <- kmeans(x, centers=4)
plot (x, col=km$cluster)
```



Hierarchical clustering

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

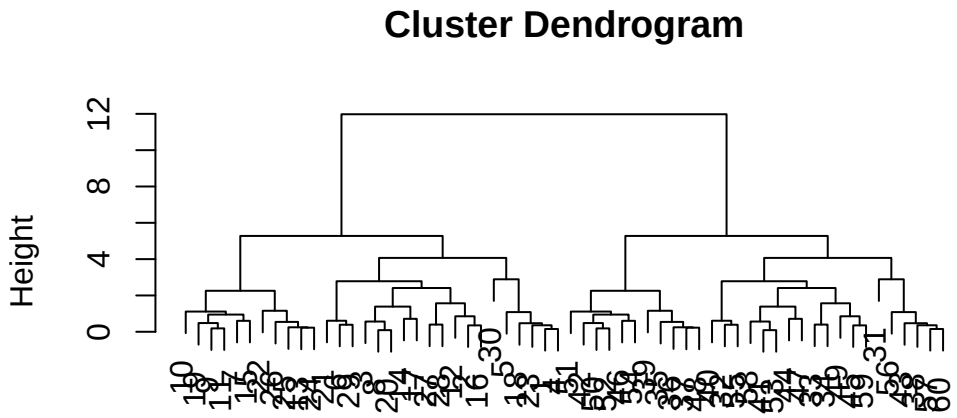
we will use the `dist()` function to start with.

```
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=8)
grps
```

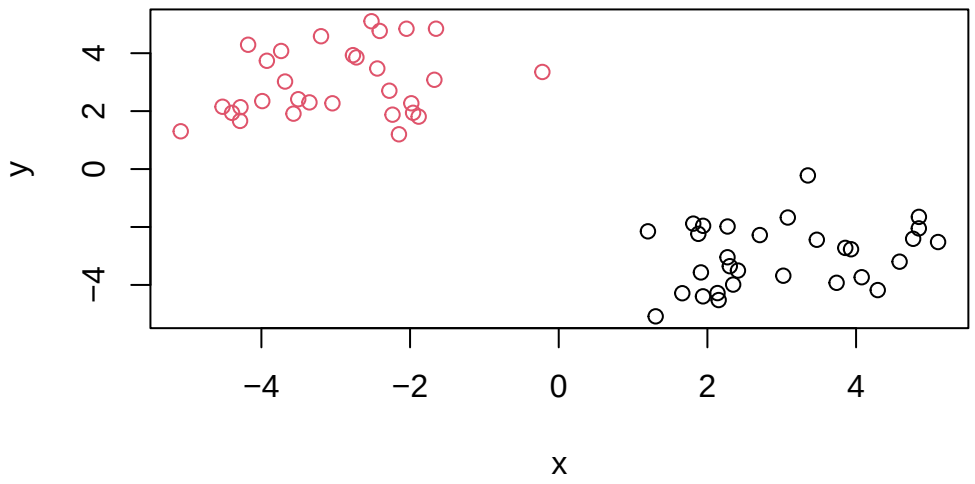
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

You can also tell `cutree()` ro 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 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
```

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



Principal Component Analysis (PCA)

```
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
```

Checking Our Data

Making sure no odd things have happened during importing phase. Using the `view()` allows us to display all data

we will be using the `rownames()` function to fix the row-names as they were set incorrectly in the first column of our x data.

```
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
```

Checkin dimensons again:

```
dim(x)
```

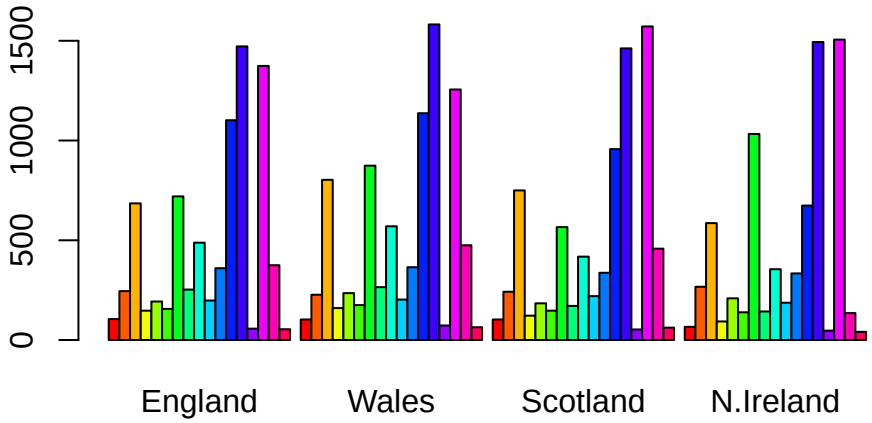
```
[1] 17  4
```

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?

2. I prefer the x <- read.csv(url, row.anmes=1) function as it is more accurate and the data won't change compared to the other one.

spoting major differences and trends

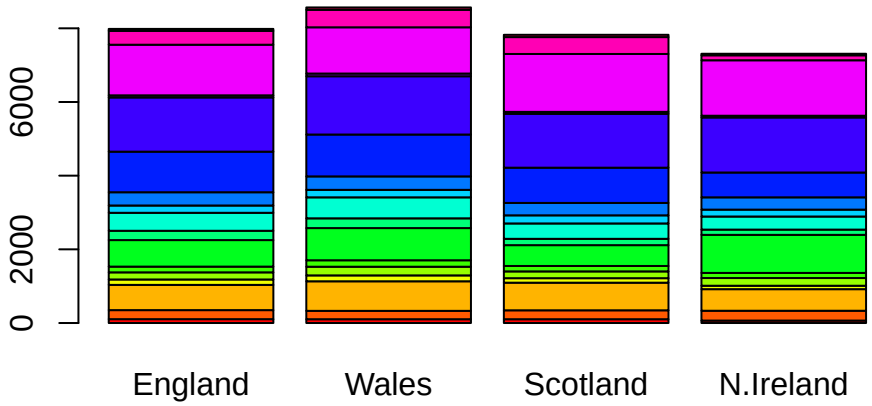
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

Changing the beside=T to beside=F will give us the following graph:

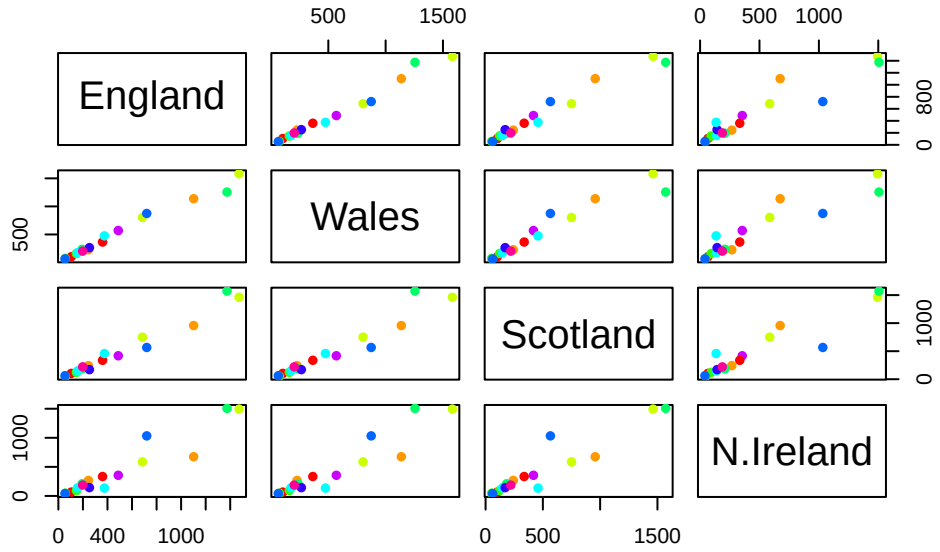
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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?

The code below generates a scatterplot matrix of pairwise plots for the columns of data frame for UK food data. If the points lies on the diagnoal of a plot, it means that two variables are the same variable.

```
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?

N. Ireland is different compared to the rest of the other Uk countries because less variables are similar to one another. Also the blue and orange dots in N.Ireland are different compared to the rest of the UK countries.

PCA to the rescue

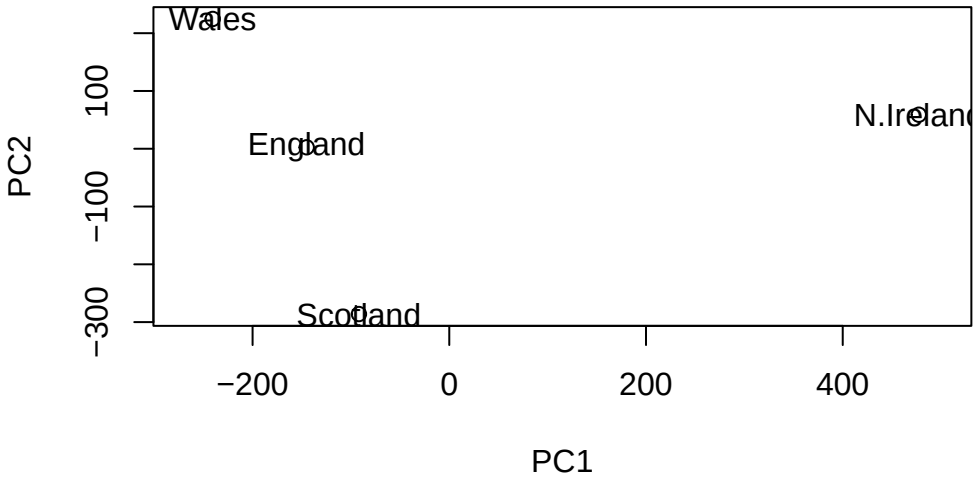
We will be using the `prcomp()` funcion for a slick graphing approach. However, we first need to transpose our data.frame matrix with `t()` transpose function.

```
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
```

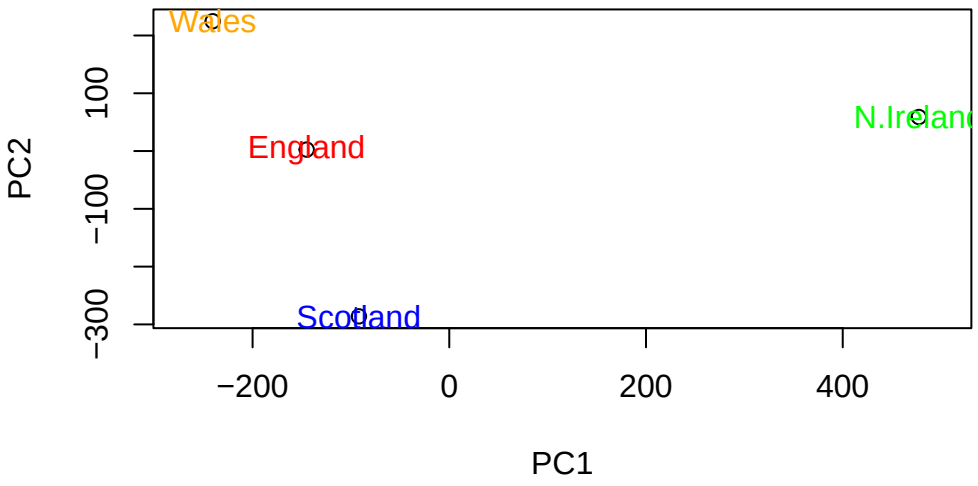
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
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], colnames(x), col= c("red", "orange", "blue", "green"))
```


Wales
England
N.Ireland
Scotland
PC2
100
−100
−300
−200
0
200
400
PC1