Class 07

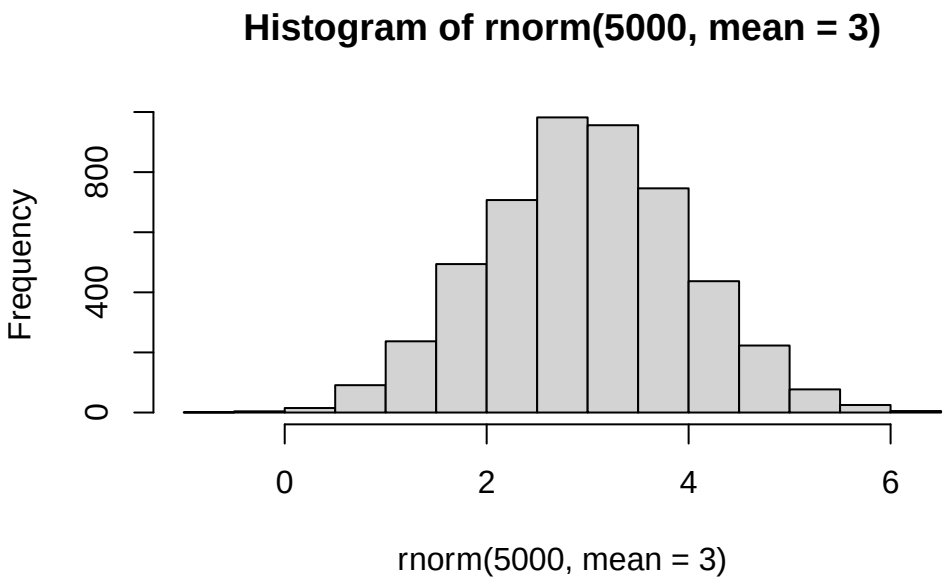
Raidah Anisah Huda A16124317

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

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

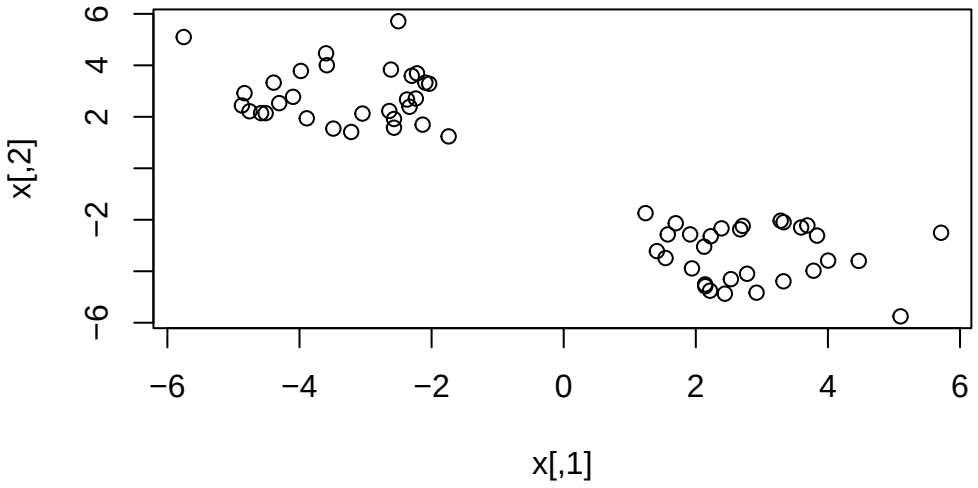
```
 [1]  2.126058  2.439757  3.328727  2.143113  2.919610  1.696100  2.226026
 [8]  3.327221  2.672068  1.239093  5.712966  1.575443  5.099664  1.541645
[15]  3.594217  3.835164  2.388632  2.775934  3.688904  2.141017  2.530940
[22]  2.214902  1.941465  4.002771  3.781818  2.709947  1.411174  4.466843
[29]  3.285112  1.914159 -2.568865 -2.035928 -3.597790 -3.216949 -2.238981
[36] -3.979299 -3.586138 -3.889531 -4.756313 -4.306335 -4.512334 -2.220598
[43] -4.097570 -2.333804 -2.615496 -2.299976 -3.487246 -5.752437 -2.569267
[50] -2.503699 -1.742187 -2.372342 -4.390739 -2.640083 -2.135712 -4.832269
[57] -4.581240 -2.094709 -4.871549 -3.046387
```

```r
x<-cbind(x<-tmp, y<-rev(tmp))
x
```

```
          [,1]      [,2]
 [1,]  2.126058 -3.046387
 [2,]  2.439757 -4.871549
 [3,]  3.328727 -2.094709
 [4,]  2.143113 -4.581240
 [5,]  2.919610 -4.832269
 [6,]  1.696100 -2.135712
 [7,]  2.226026 -2.640083
 [8,]  3.327221 -4.390739
 [9,]  2.672068 -2.372342
[10,]  1.239093 -1.742187
[11,]  5.712966 -2.503699
[12,]  1.575443 -2.569267
[13,]  5.099664 -5.752437
[14,]  1.541645 -3.487246
[15,]  3.594217 -2.299976
[16,]  3.835164 -2.615496
[17,]  2.388632 -2.333804
[18,]  2.775934 -4.097570
[19,]  3.688904 -2.220598
[20,]  2.141017 -4.512334
[21,]  2.530940 -4.306335
[22,]  2.214902 -4.756313
[23,]  1.941465 -3.889531
```

```
[24,]  4.002771 -3.586138
[25,]  3.781818 -3.979299
[26,]  2.709947 -2.238981
[27,]  1.411174 -3.216949
[28,]  4.466843 -3.597790
[29,]  3.285112 -2.035928
[30,]  1.914159 -2.568865
[31,] -2.568865  1.914159
[32,] -2.035928  3.285112
[33,] -3.597790  4.466843
[34,] -3.216949  1.411174
[35,] -2.238981  2.709947
[36,] -3.979299  3.781818
[37,] -3.586138  4.002771
[38,] -3.889531  1.941465
[39,] -4.756313  2.214902
[40,] -4.306335  2.530940
[41,] -4.512334  2.141017
[42,] -2.220598  3.688904
[43,] -4.097570  2.775934
[44,] -2.333804  2.388632
[45,] -2.615496  3.835164
[46,] -2.299976  3.594217
[47,] -3.487246  1.541645
[48,] -5.752437  5.099664
[49,] -2.569267  1.575443
[50,] -2.503699  5.712966
[51,] -1.742187  1.239093
[52,] -2.372342  2.672068
[53,] -4.390739  3.327221
[54,] -2.640083  2.226026
[55,] -2.135712  1.696100
[56,] -4.832269  2.919610
[57,] -4.581240  2.143113
[58,] -2.094709  3.328727
[59,] -4.871549  2.439757
[60,] -3.046387  2.126058
```

```
plot(x)
```



K-means clustering.

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

```
km<-kmeans(x, centers=2)
km
```

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

Cluster means:
       [,1]      [,2]
1  2.824350 -3.309192
2 -3.309192  2.824350

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 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

Within cluster sum of squares by cluster:
[1] 69.33222 69.33222
 (between_SS / total_SS =  89.1 %)
```

```
Available components:

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

```
km$size
```

```
[1] 30 30
```

Q. How many points are in each cluster?

`km$size` determines that there are 30 points in each cluster

```
km$size
```

```
[1] 30 30
```

Q. What component gives:

-cluster size

`size` component gives cluster size

-cluster assignment

```
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 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
```

`cluster` component gives assignment

-cluster centers

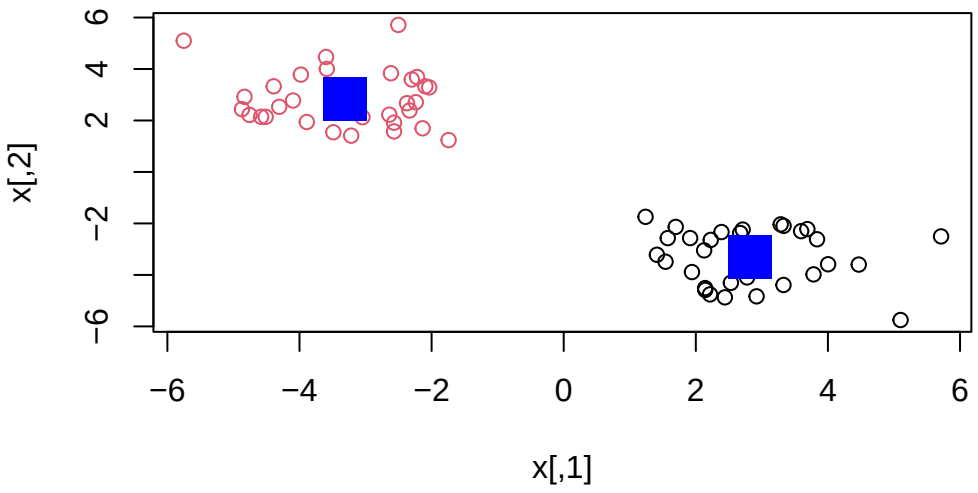
```
km$centers
```

```
       [,1]      [,2]
1  2.824350 -3.309192
2 -3.309192  2.824350
```

`centers` component gives centers

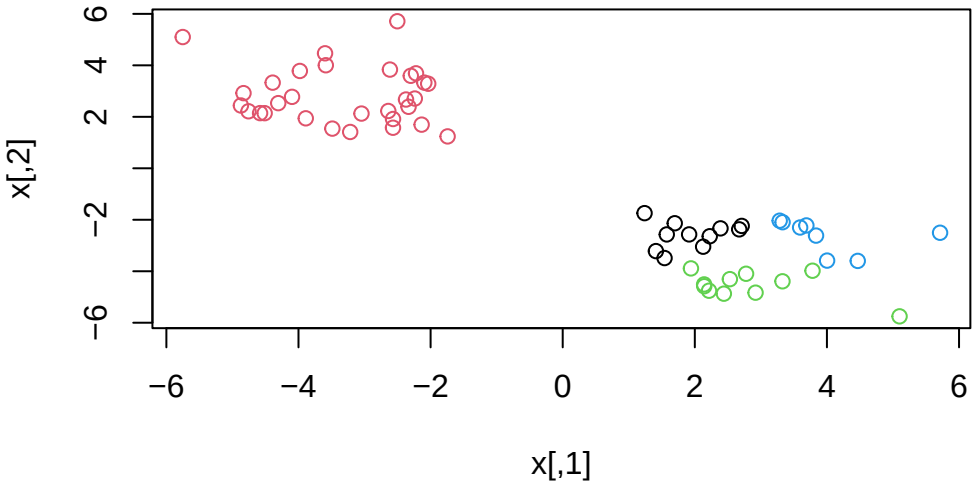
Q: plot x colored by the kmeans clusters assignment and add cluster centers as blue points.

```
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=4)
plot(x,col=km$cluster)
```



Hierarchical Clustering

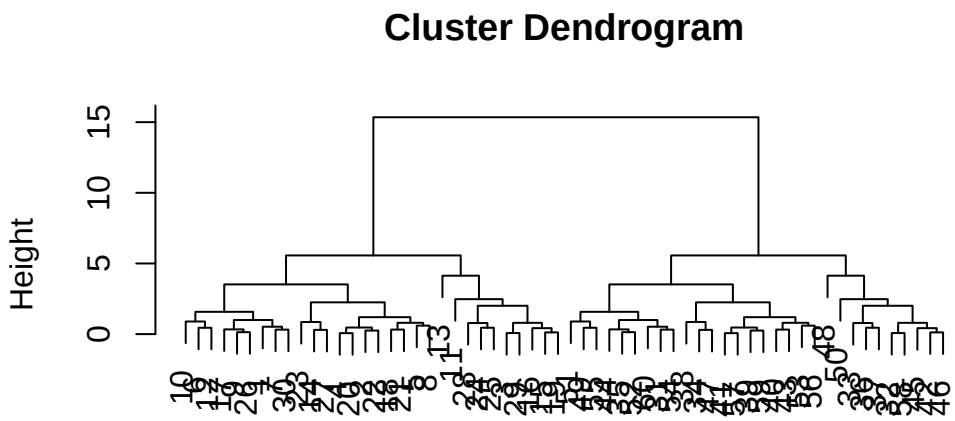
We can use the `hclust()` function for Hierarchical Clustering. Unlike `kmeans()`, where we could just pass in our data as inputs, we need to give `hclust()` a "distance matrix".

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
```

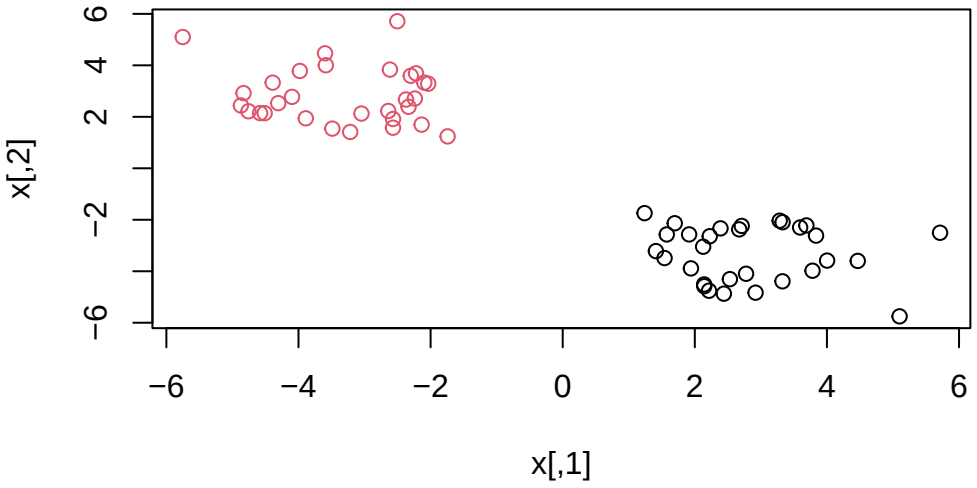
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

You can also tell `cutree()` 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 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)

It aims to reveal that most important structure in the dataset by reducing dimensionality. -PC1: where the data are the most spread -PC2, etc: less spread than previos PCn, etc.

LAB Part 1

```
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
```

```
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
```

There are actually 17 rows and 4 columns

Fixing row names

```
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
```

```
dim(x)
```

```
[1] 17  4
```

vs

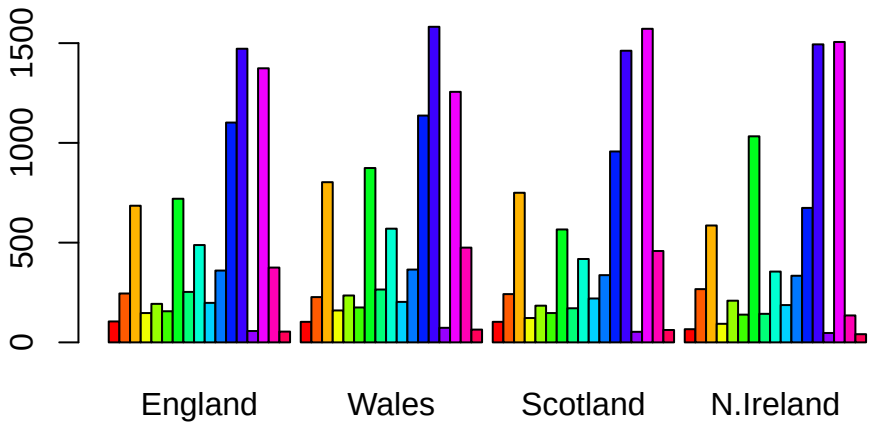
```
x <- read.csv(url, row.names=1)
head(x)
```

```
              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?

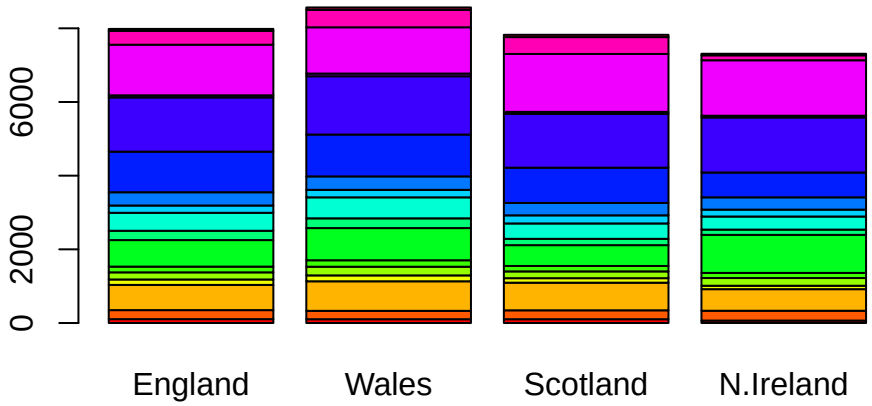
I prefer the second approach to solving the row-names problem because it makes it very clear that the first column should actually be the row name. In contrast, the other approach is more risky because if you rerun `x <- x[,-1]`, you would end up dropping a column each time.

```
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?

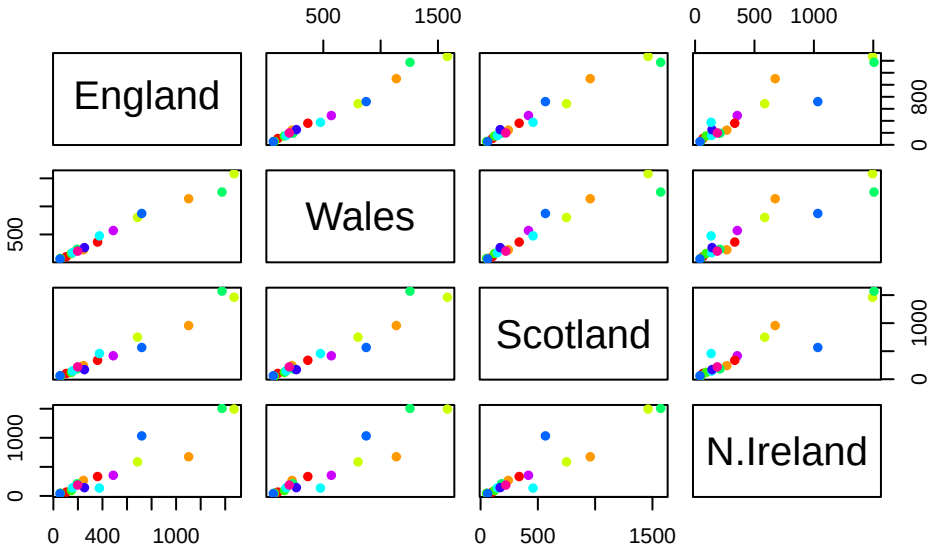
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



By changing the optional argument of `beside=` from `T` to `F`, we are able to make the bars lay ontop of each other instead of beside each other. But it is still hard to discern differences with this plot.

> 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)
```



It appears that this pairwise plot is showing that the straighter lines(points laying on the diagonal) indicate that the data is the same/close between the two countries. Points further from the diagonal indicate that which country has more of the item depending on which country the point is pulled to.

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

Between N.Ireland and the other UK countries, the largest difference appears to be that Northern Ireland has the largest difference in food consumption in contrast to the other countries.

PCA to the rescue

The main PCS function in base R is classed `prcomp()` it expects the transpose of our data.

```
# 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
```

```
attributes(pca)
```

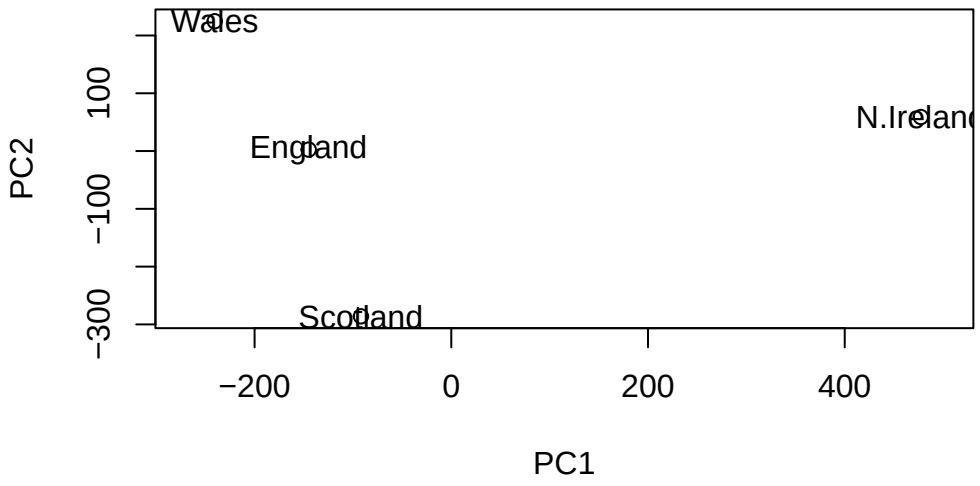
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```
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
```

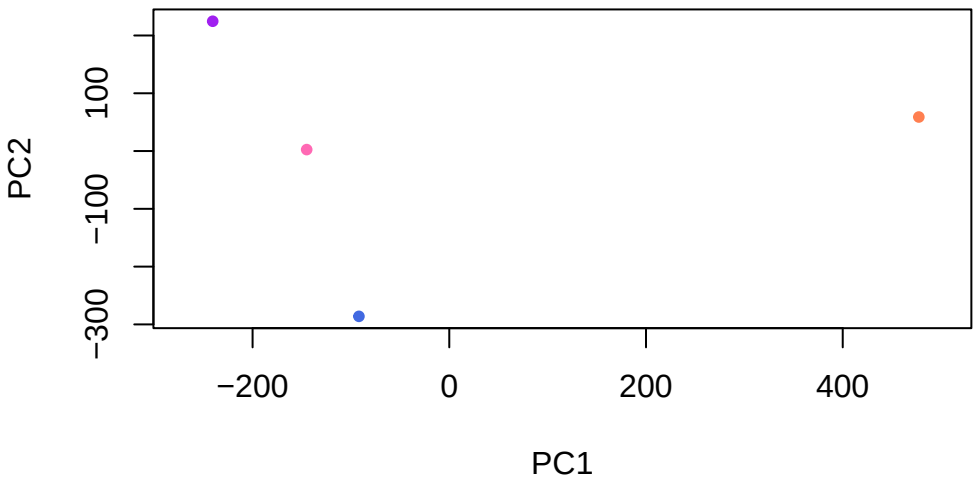
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# 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), col=c("hotpink","purp
```


PC2
100
−100
−300
−200
0
200
400
PC1