Class 7: Clustering and PCA

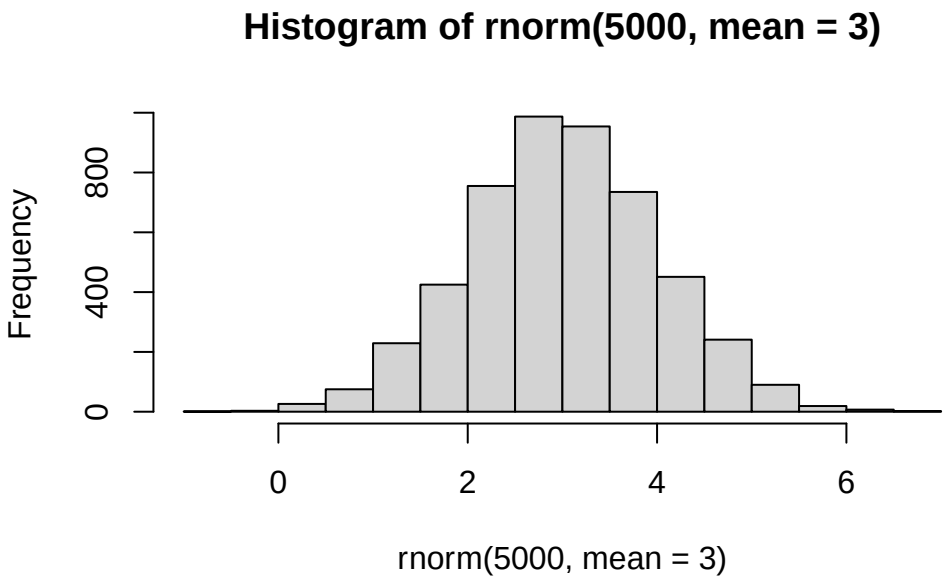
Trinity Leahy

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.

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

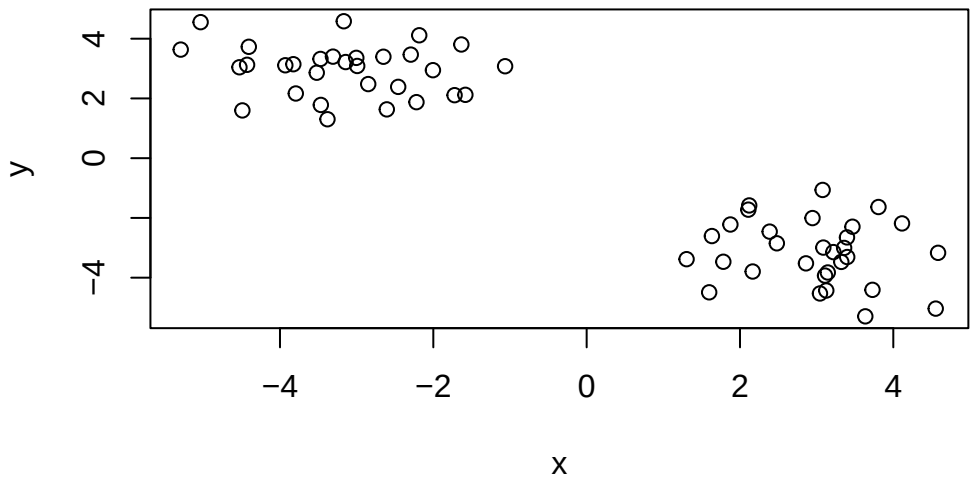
Now we will use `rev` to reverse the order of our vector and then `cbind` in order to put our vector into a column/

```
x <- cbind(x = spread, y = rev(spread))
x
```

```
              x         y
 [1,]  2.480730 -2.849145
 [2,]  1.874934 -2.219749
 [3,]  2.945980 -2.004916
 [4,]  3.806381 -1.634434
 [5,]  2.387235 -2.458239
 [6,]  2.165601 -3.793719
 [7,]  3.043219 -4.528724
 [8,]  3.218125 -3.143696
 [9,]  4.113796 -2.183553
[10,]  3.078065 -1.063731
[11,]  4.583948 -3.167989
[12,]  3.728398 -4.406747
[13,]  1.303456 -3.380669
[14,]  3.125781 -4.429457
[15,]  3.145194 -3.825847
[16,]  3.634839 -5.294785
[17,]  2.860721 -3.520510
[18,]  3.358449 -3.004496
[19,]  3.109431 -3.930680
[20,]  1.633755 -2.606705
[21,]  2.109032 -1.722715
[22,]  4.553269 -5.035247
[23,]  3.467112 -2.294381
[24,]  3.321637 -3.471708
[25,]  1.782648 -3.467376
[26,]  3.398894 -3.308877
[27,]  3.395619 -2.651108
[28,]  2.120444 -1.581810
[29,]  1.598417 -4.490488
[30,]  3.087097 -2.992734
[31,] -2.992734  3.087097
[32,] -4.490488  1.598417
[33,] -1.581810  2.120444
```

```
[34,] -2.651108  3.395619
[35,] -3.308877  3.398894
[36,] -3.467376  1.782648
[37,] -3.471708  3.321637
[38,] -2.294381  3.467112
[39,] -5.035247  4.553269
[40,] -1.722715  2.109032
[41,] -2.606705  1.633755
[42,] -3.930680  3.109431
[43,] -3.004496  3.358449
[44,] -3.520510  2.860721
[45,] -5.294785  3.634839
[46,] -3.825847  3.145194
[47,] -4.429457  3.125781
[48,] -3.380669  1.303456
[49,] -4.406747  3.728398
[50,] -3.167989  4.583948
[51,] -1.063731  3.078065
[52,] -2.183553  4.113796
[53,] -3.143696  3.218125
[54,] -4.528724  3.043219
[55,] -3.793719  2.165601
[56,] -2.458239  2.387235
[57,] -1.634434  3.806381
[58,] -2.004916  2.945980
[59,] -2.219749  1.874934
[60,] -2.849145  2.480730
```

```
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:
          x         y
1  2.947740 -3.148808
2 -3.148808  2.947740

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

Within cluster sum of squares by cluster:
[1] 54.04676 54.04676
 (between_SS / total_SS =  91.2 %)
```

```
Available components:

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

```
km$size
```

```
[1] 30 30
```

```
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
```

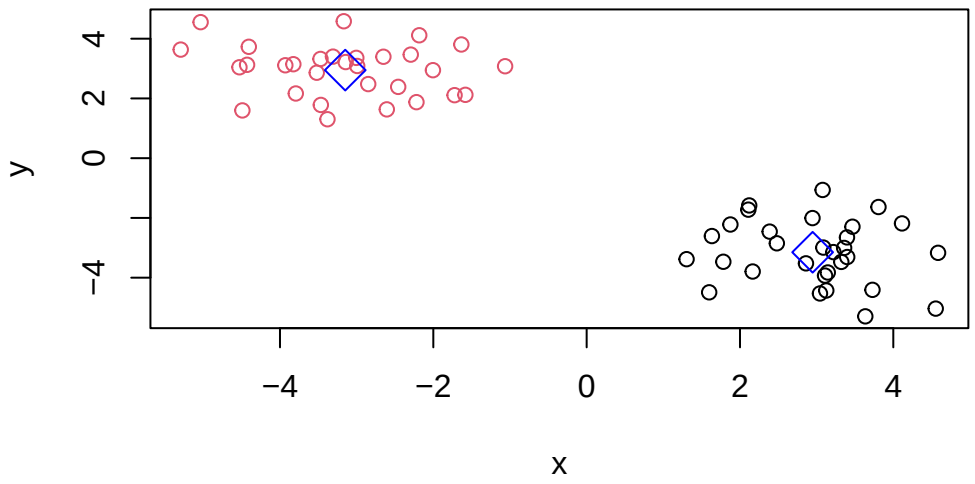
```
km$centers
```

```
          x         y
1  2.947740 -3.148808
2 -3.148808  2.947740
```

There are 30 points in each cluster. - cluster size is `km$size` - cluster assignment/membership is `km$cluster` - cluster center is `km$centers`

> 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 = 5, cex = 2)
```



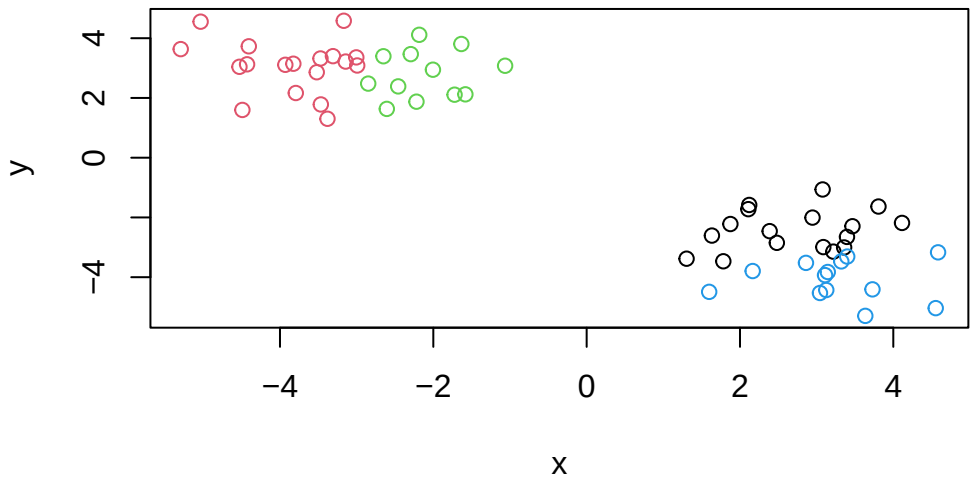
Let's cluster into 3 groups of the same `x` data and make a plot.

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

```
K-means clustering with 4 clusters of sizes 17, 18, 12, 13

Cluster means:
          x         y
1  2.715462 -2.427027
2 -3.844097  3.056618
3 -2.105874  2.784424
4  3.251489 -4.092675

Clustering vector:
 [1] 1 1 1 1 1 4 4 1 1 1 4 4 1 4 4 4 4 1 4 1 1 4 1 4 1 4 1 1 4 1 2 2 3 3 2 2 2 3
[39] 2 3 3 2 2 2 2 2 2 2 2 2 3 3 2 2 2 3 3 3 3 3

Within cluster sum of squares by cluster:
[1] 18.20275 21.67806 10.08102 13.28942
 (between_SS / total_SS =  94.8 %)
```

```
Available components:

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

```
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".

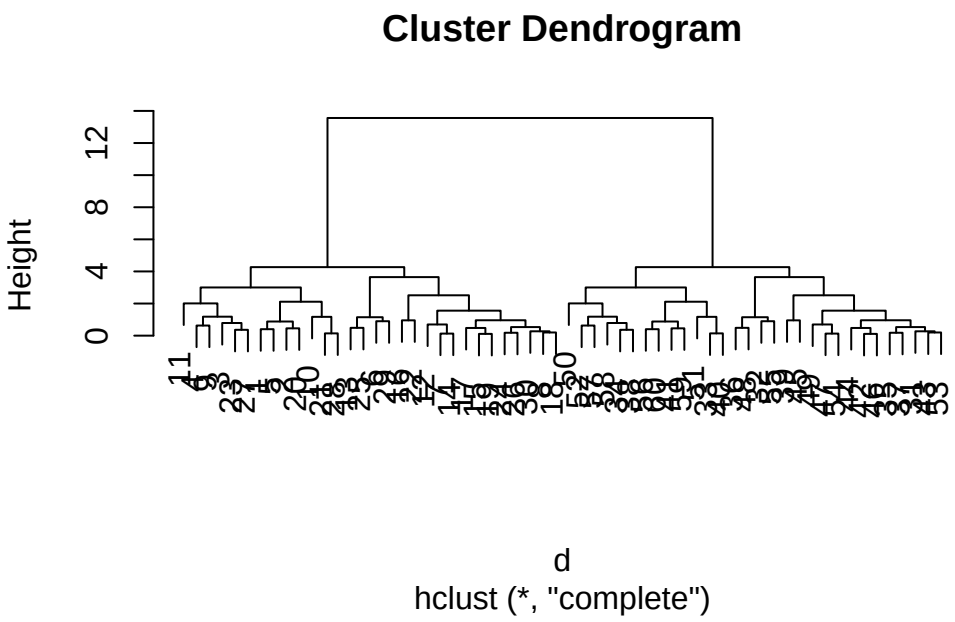
We will use the `dist()` function to start with.

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

```
Call:
hclust(d = d)
```

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

```
plot(hc)
```



I can 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 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
```

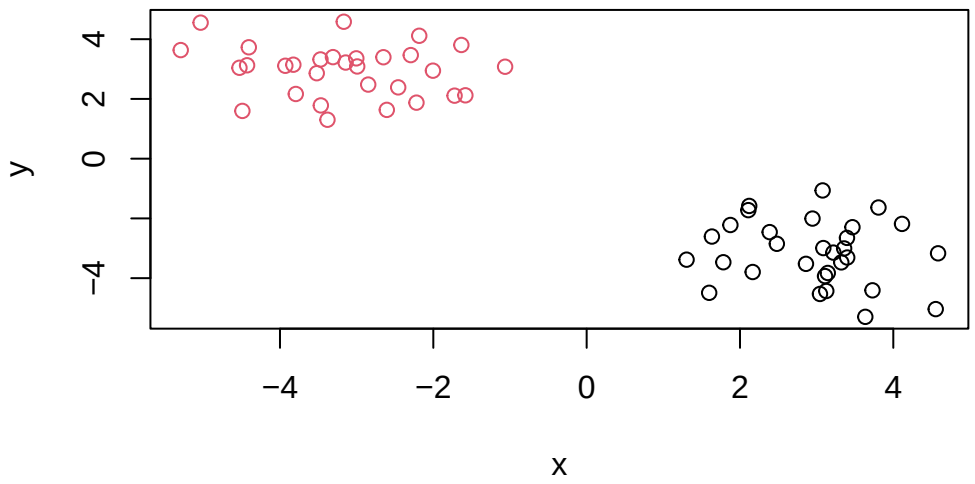
You can also tell `cutree()` to cut where it yeilds "k" groups.

```
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 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
```

Now we will plot our data using hierarchical clustering.

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



Principal Component Analysis (PCA)

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names = 1)
```

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  4
```

There are 17 rows and 4 columns.

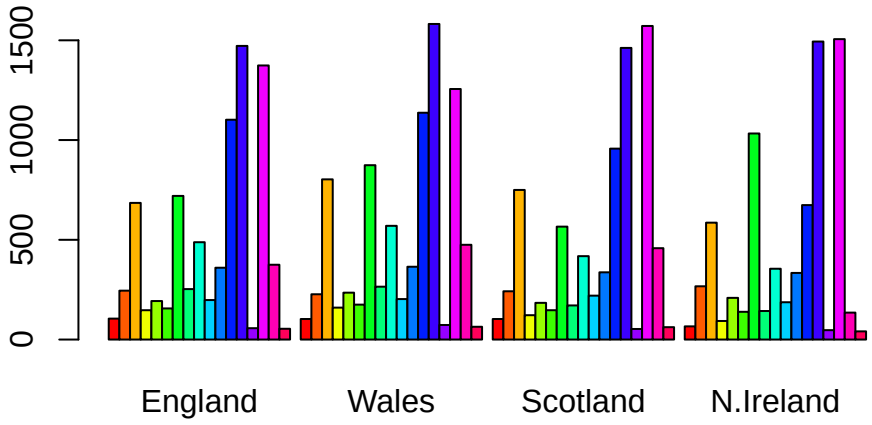
```
# Preview the first 6 rows.
head(x)
```

```
              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?

The second approach listed in the lab material is preferable in which we call the url and designate the categories as row names. The other method, in which we use a negative function can damage our dataset and get rid of columns, which is not helpful or efficient.

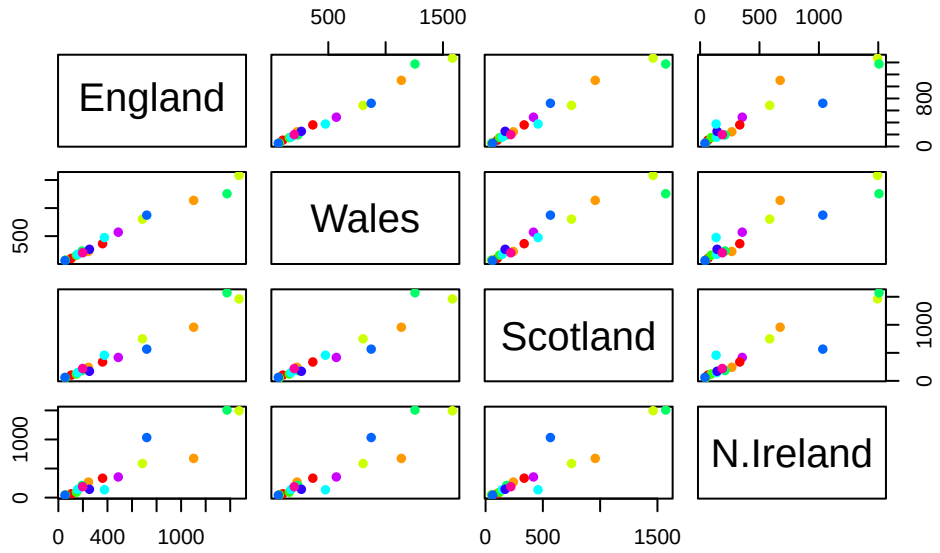
```
barplot(as.matrix(x), beside = T, col = rainbow(nrow(x)))
```



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

Setting the `beside` argument equal to FALSE.

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



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?

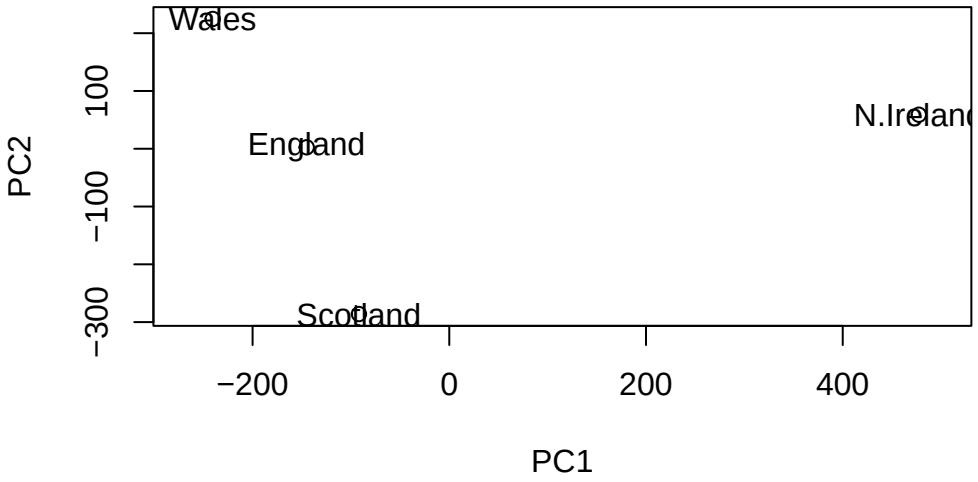
The `pairwise` function assigns two different countries to the x and y axis, plotting the different data points for each category following that x and y scheme. If a given point lies near/on the diagonal for a given plot, it means that the data for that point for the y-axis country and the x-axis country are around the same amount. The farther the point is from the diagonal, the greater the difference between the data points for each country.

```
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
```

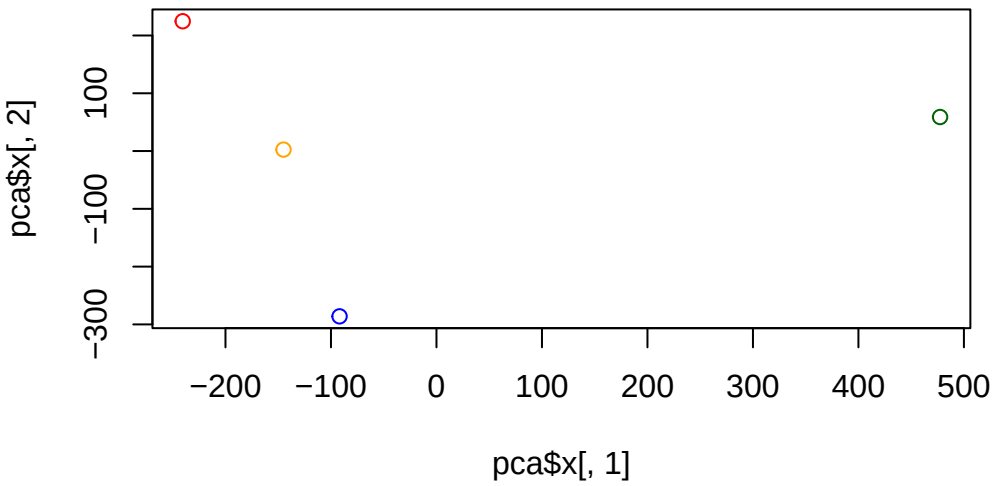
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
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], col=c("orange", "red", "blue", "darkgreen"))
```


pca$x[, 2]
100
−100
−300
−200
−100
0
100
200
300
400
500
pca$x[, 1]