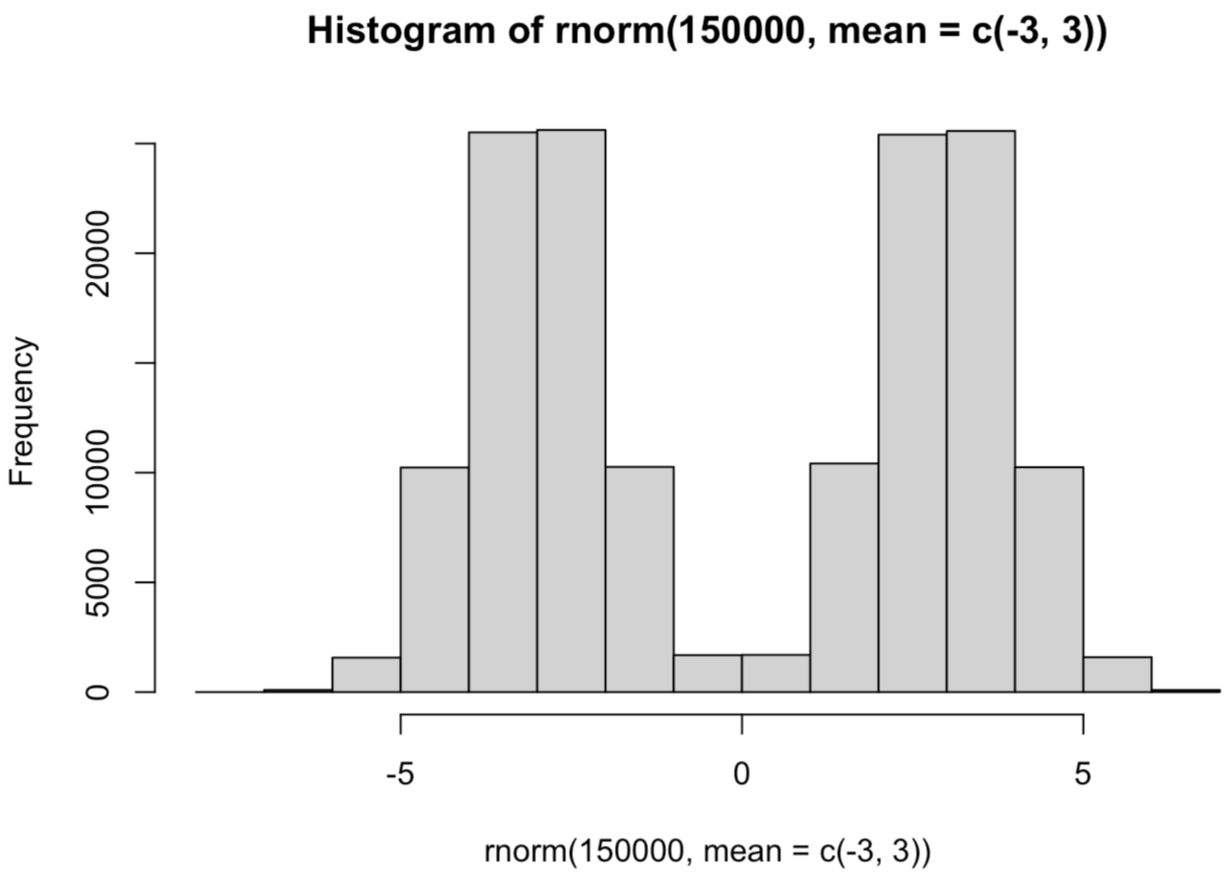
Class 7: Machine Learning 1

AUTHOR
Tiffany Chin PID 15700705

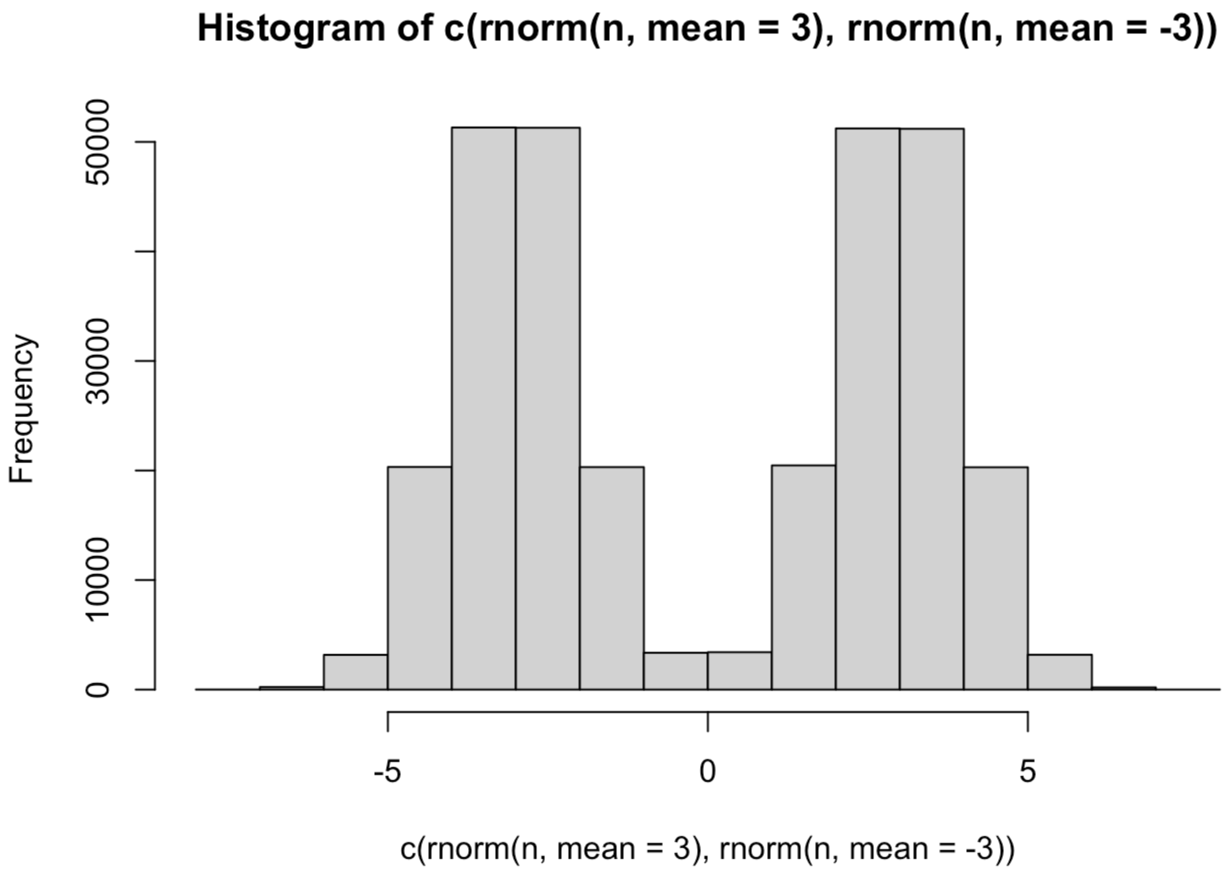
Before we get into clustering methods, let’s make some sample data to cluster where we know what the answer should be.

To help with this, I will use the `rnorm()` function.

```r
hist(rnorm(150000, mean = c(-3,3)))
```



```r
#rnorm(150000, mean = c(-3,3))
#same as
n = 150000
hist(c(rnorm(n, mean =3), rnorm(n, mean = -3)))
```

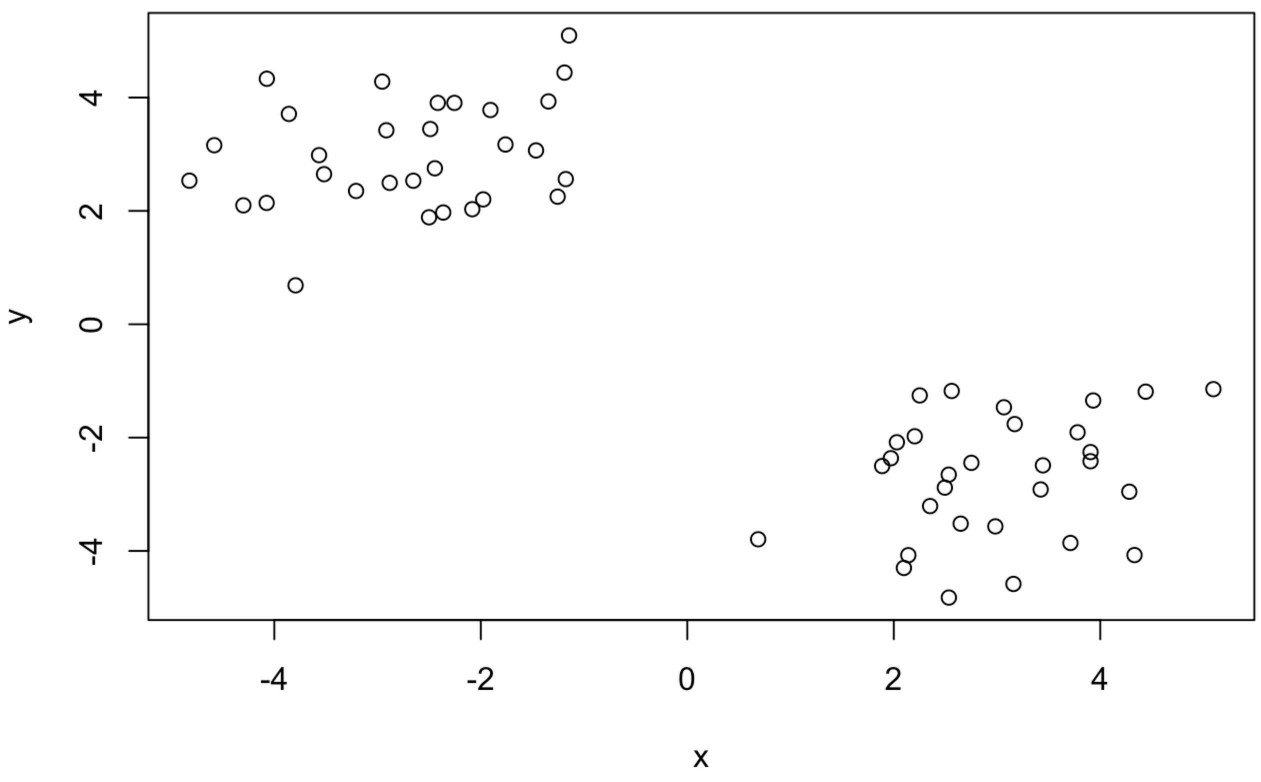


```r
n=30
x <- c(rnorm(n, mean =3), rnorm(n, mean = -3))
#set y as reverse of x
y <- rev(x)
#use cbind to combine x and y together
z <- cbind(x,y)
z
```

```
              x          y
 [1,]  2.2513098 -1.2559354
 [2,]  4.3315399 -4.0723973
 [3,]  2.0298507 -2.0824691
 [4,]  3.1712888 -1.7612314
 [5,]  2.0979060 -4.2997163
 [6,]  3.9315672 -1.3450126
 [7,]  0.6864118 -3.7942137
 [8,]  2.1407705 -4.0745183
 [9,]  3.9055832 -2.2557048
[10,]  2.5321830 -2.6537386
[11,]  3.0666983 -1.4652343
[12,]  2.4945748 -2.8823927
[13,]  1.9717088 -2.3640029
[14,]  2.2029597 -1.9767342
```

```
[15,]  3.4224037 -2.9150514
[16,]  1.8873257 -2.5015661
[17,]  2.9835508 -3.5666858
[18,]  3.9056399 -2.4168939
[19,]  3.7109053 -3.8588876
[20,]  2.6477652 -3.5184310
[21,]  4.2812557 -2.9549149
[22,]  5.0957054 -1.1451779
[23,]  3.4444481 -2.4895826
[24,]  3.1588835 -4.5820229
[25,]  2.3517633 -3.2081814
[26,]  2.5334469 -4.8225964
[27,]  3.7801343 -1.9069593
[28,]  2.7515827 -2.4452184
[29,]  4.4399845 -1.1893702
[30,]  2.5610391 -1.1771377
[31,] -1.1771377  2.5610391
[32,] -1.1893702  4.4399845
[33,] -2.4452184  2.7515827
[34,] -1.9069593  3.7801343
[35,] -4.8225964  2.5334469
[36,] -3.2081814  2.3517633
[37,] -4.5820229  3.1588835
[38,] -2.4895826  3.4444481
[39,] -1.1451779  5.0957054
[40,] -2.9549149  4.2812557
[41,] -3.5184310  2.6477652
[42,] -3.8588876  3.7109053
[43,] -2.4168939  3.9056399
[44,] -3.5666858  2.9835508
[45,] -2.5015661  1.8873257
[46,] -2.9150514  3.4224037
[47,] -1.9767342  2.2029597
[48,] -2.3640029  1.9717088
[49,] -2.8823927  2.4945748
[50,] -1.4652343  3.0666983
[51,] -2.6537386  2.5321830
[52,] -2.2557048  3.9055832
[53,] -4.0745183  2.1407705
[54,] -3.7942137  0.6864118
[55,] -1.3450126  3.9315672
[56,] -4.2997163  2.0979060
[57,] -1.7612314  3.1712888
[58,] -2.0824691  2.0298507
[59,] -4.0723973  4.3315399
[60,] -1.2559354  2.2513098
```

```r
plot(z)
```



K-means clustering

The function in base R for k-means clustering is called `kmeans()`. The two arguments it needs, without defaults, is x (our data) and centers (the number of clusters, *k*).

```r
km <- kmeans(z, 2)
km
```

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

Cluster means:
          x         y
1 -2.699399  2.992340
2  2.992340 -2.699399

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 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] 60.92723 60.92723
 (between_SS / total_SS =  88.9 %)
```

```
Available components:

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

#clustering vector shows which cluster that value has been assigned to. There are 30 points assigned to 1 cluster, and 30 assigned to the 2nd. Since our x and y are just reverse of each other, the first 30 are assigned to 1 and then last 30 are assigned to 2nd, which makes sense based off distance.

```
#to print out the centers of the two clusters
km$centers
```

```
          x         y
1 -2.699399  2.992340
2  2.992340 -2.699399
```

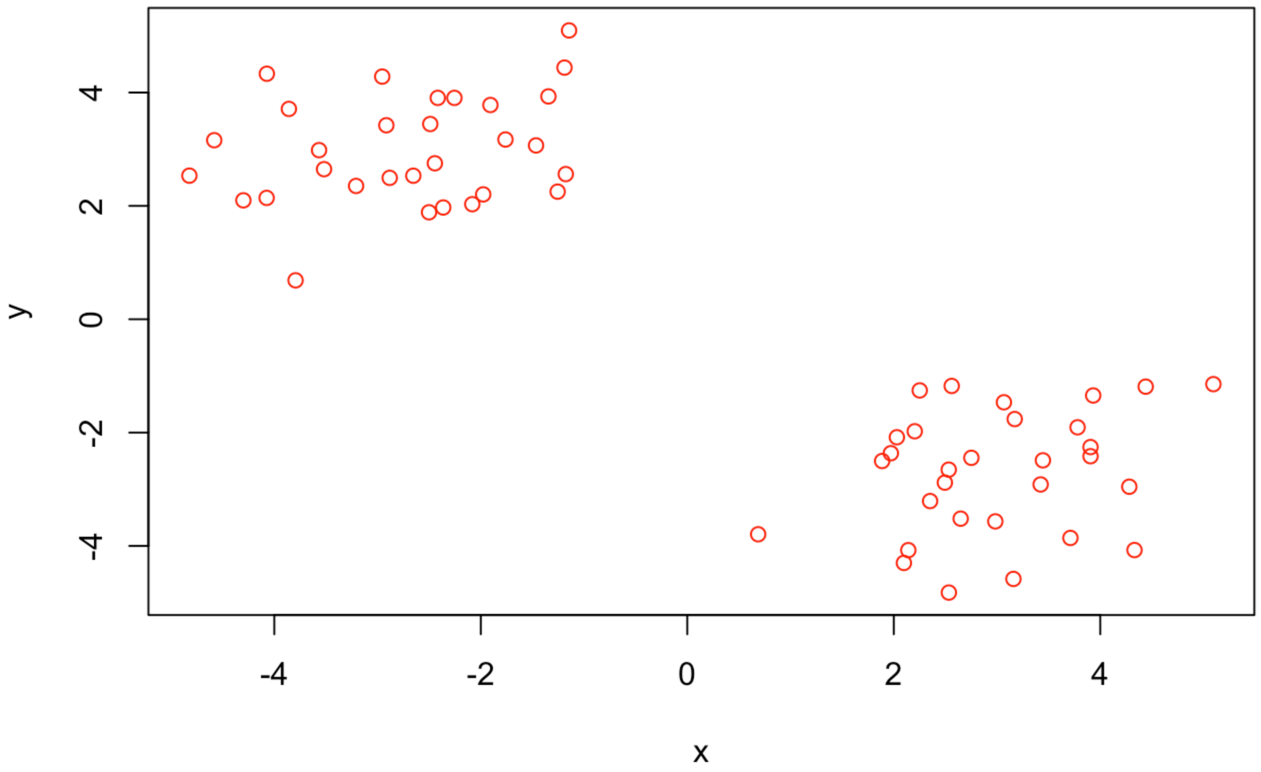
Q. Print out the cluster membership vector (i.e. our main answer). Which cluster is everything in?

```
km$cluster
```

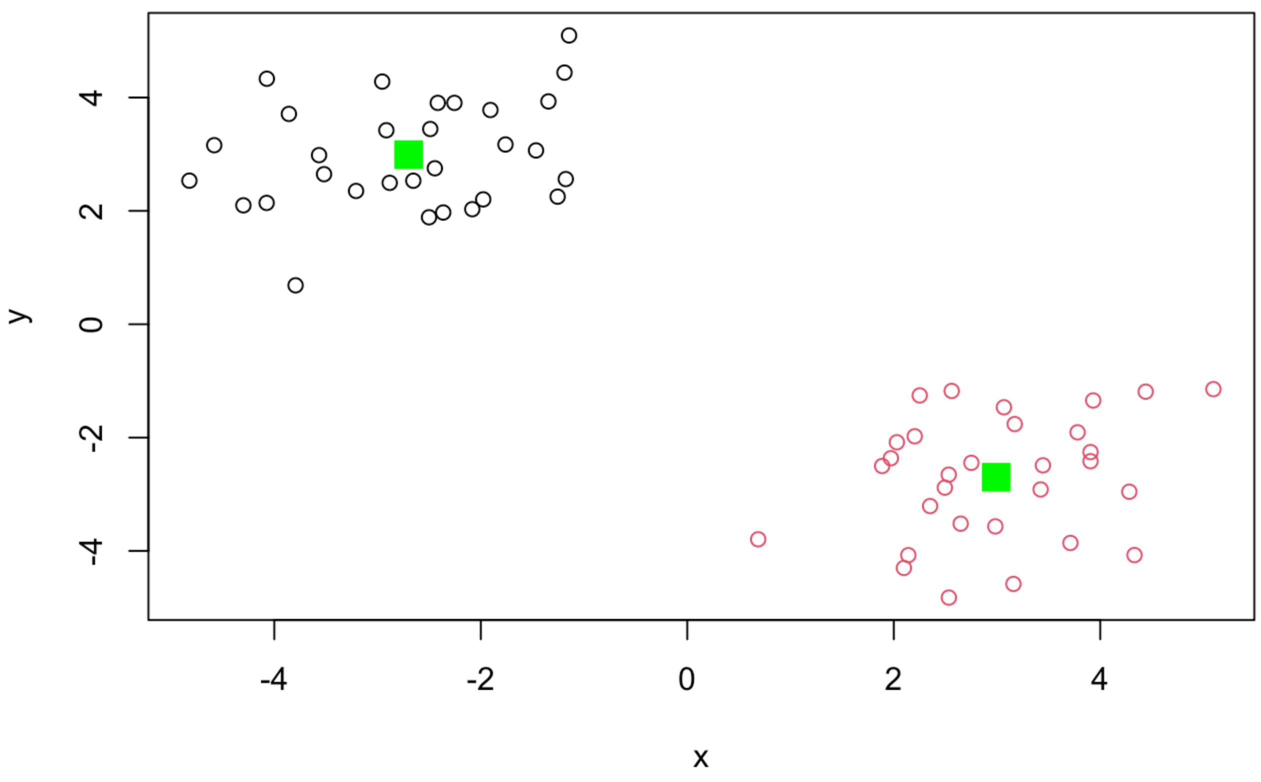
```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Make a plot of this data with the clustering data shown, using base R plot

```
plot(z, col="red")
```



```r
#the above colors all points red, but we want this separated
#below, use the km clustering results to color the plot
plot(z, col = km$cluster)
#to this plot, add center points as green boxes
points(km$centers, col = "green", pch = 15, cex =2)
```



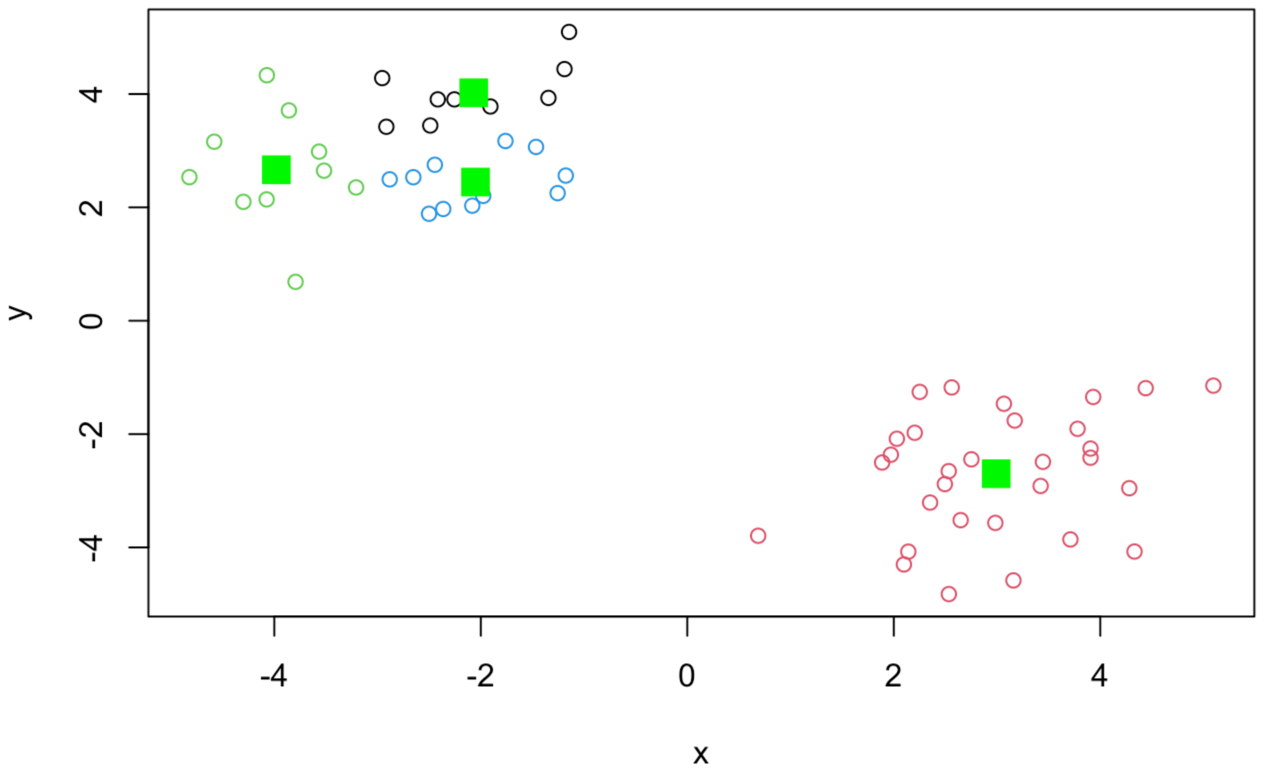
Q. Can you cluster our data in `z` into four clusters?

```r
km4 <- kmeans(z, 4)
km4
```

```
K-means clustering with 4 clusters of sizes 9, 30, 10, 11

Cluster means:
          x         y
1 -2.068741  4.022969
2  2.992340 -2.699399
3 -3.979765  2.664294
4 -2.051424  2.447320

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 4 1 4 1 3 3 3 1
[39] 1 1 3 3 1 3 4 1 4 4 4 4 4 1 3 3 1 3 4 4 3 4

Within cluster sum of squares by cluster:
[1]  6.192648 60.927226 11.064460  5.175166
 (between_SS / total_SS =  92.4 %)

Available components:
```

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

```
plot(z, col = km4$cluster)
points(km4$centers, col = "green", pch = 15, cex = 2)
```



```
#kmeans will always map clusters based on how many k you specify, which is an issue when
#also re plotting this will be different each time because clustering will be different!!
```

Hierarchical Clustering

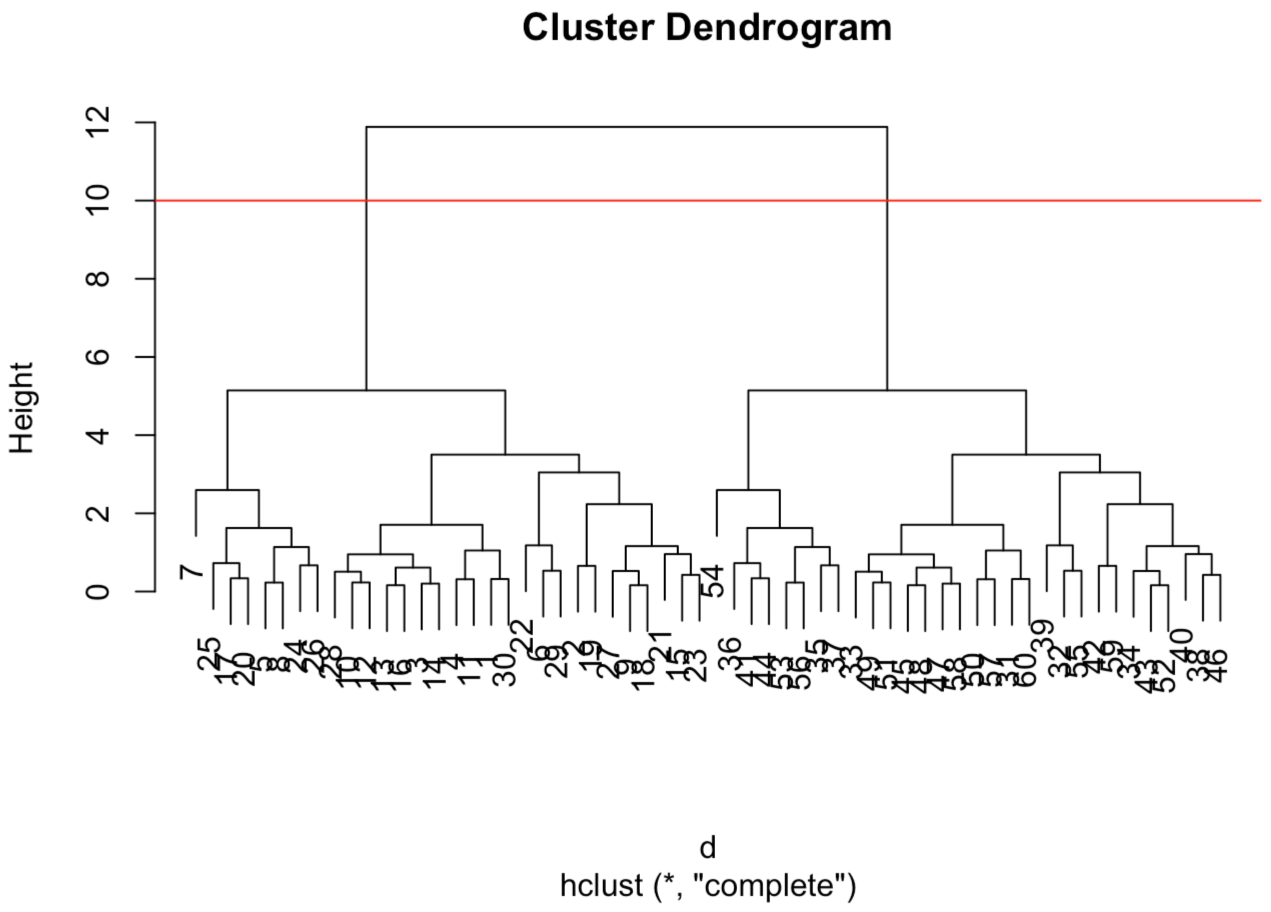
The main function for hierarchical clustering in base R is called `hclust()`. We will cluster step-by-step in a hierarchical cluster, starting from 60 (how many points of data we have) and merging clusters of similarity. More work than k-means clustering but more reliable clustering method than randomly assigning k. Unlike `kmeans()`, I cannot just pass in my data as an input. I first need a **distance matrix** from my data that will measure distance between every point and all other 59 points.

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

```
Call:
hclust(d = d)

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

There is a specific hclust plot() method we can use

```
plot(hc)
abline(h=10, col="red")
```

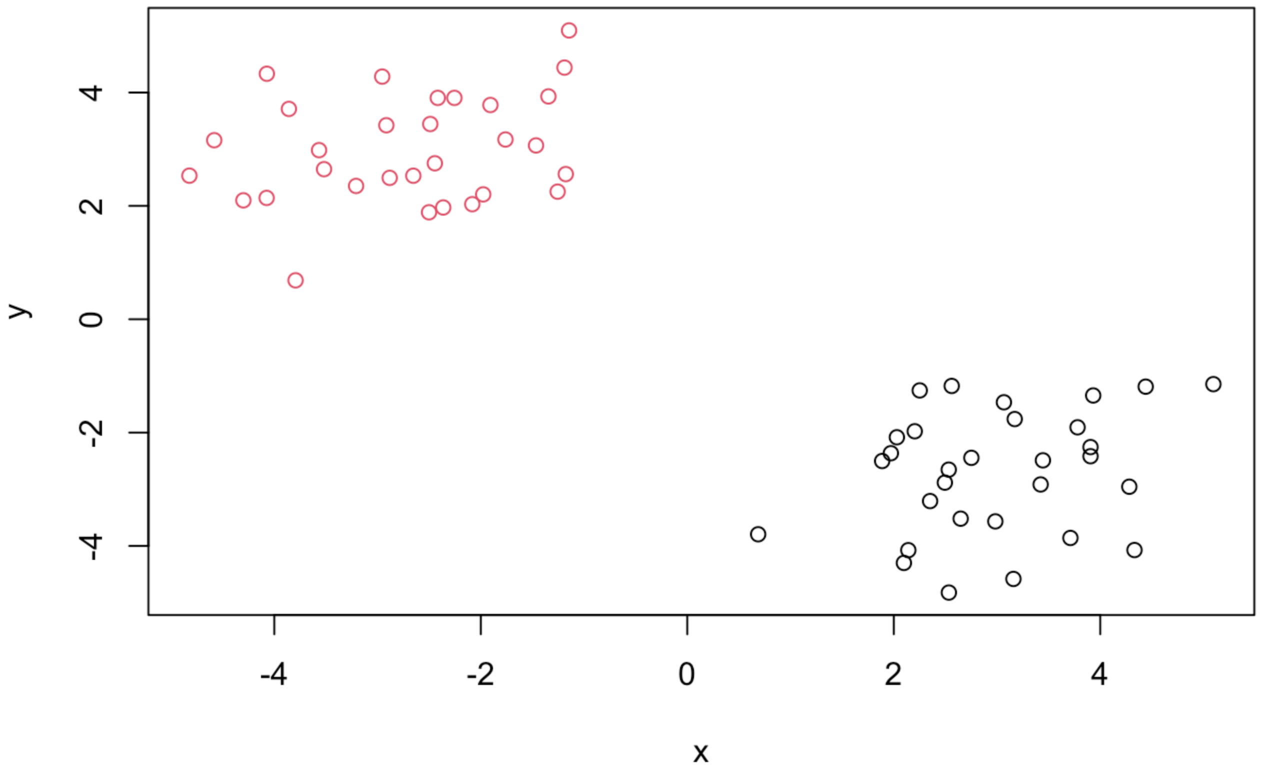


To get my main clustering result (i.e. the membership vector), I can “cut” my tree at a given height. To do this I will use the `cutree()` at the line that we looked at in the above plot.

```
grps <- cutree(hc, h=10)
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 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
```

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



Principal Component Analysis

PCA projects features onto the principal components. The motivation is to reduce the features dimensionality while only losing a small amount of information.

Principal component analysis (PCA) is a well established “multivariate statistical technique” used to reduce the dimensionality of a complex data set to a more manageable number (typically 2D or 3D). This method is particularly useful for highlighting strong paterns and relationships in large datasets (i.e. revealing major similarities and diferences) that are otherwise hard to visualize. As we will see again and again in this course PCA is often used to make all sorts of bioinformatics data easy to explore and visualize.

Lab 7 Worksheet: PCA of UK food data

Import UK_foods.csv

```
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 question?

```
dim(x)
```

```
[1] 17  5
```

```
#View(x)
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
```

Row-names were not set up properly since we only want 4 columns, one for each country. This is a common occurrence where the first column in the x dataframe contains rownames considered as a new column. Fix this below.

```
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
```

Another way to setting correct row-names could be to re-read the data file again and specify a row.names argument of read.csv() to say that the first column contains row.names

```
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
```

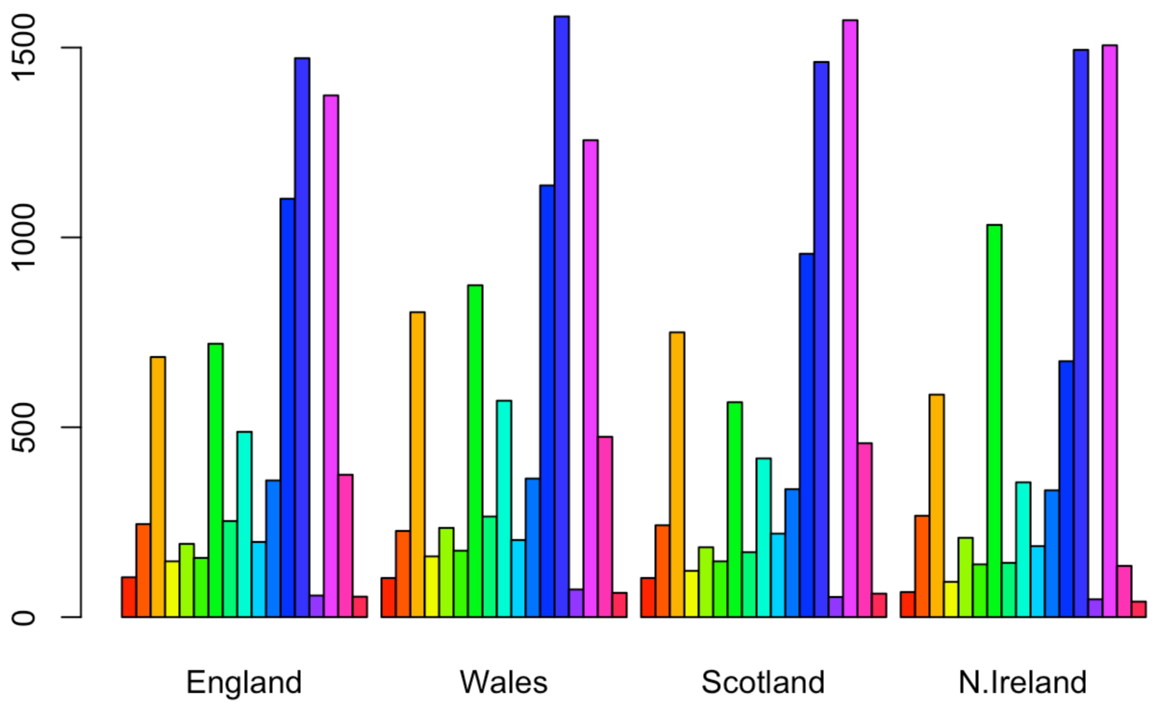
```
dim(x)
```

```
[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?

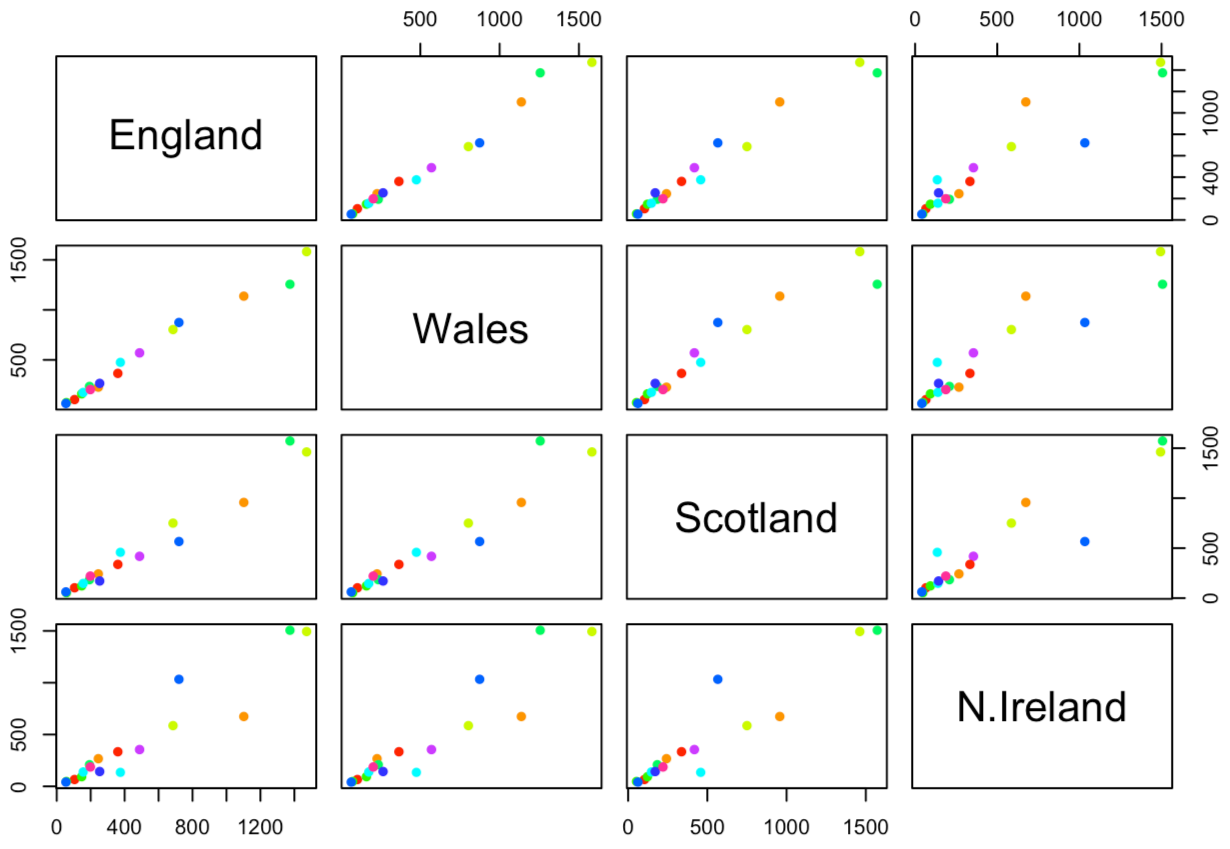
I prefer using the 2nd method of specifying row.names=1 when performing read.csv() because of how you can accidentally remove columns using the first method if you aren’t careful.

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



Generating a pair-wise plot can help somewhat to understanding the differences and similarities in this dataset.

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



We can see with the pairwise plot that N.Ireland differs from England, Wales, and Scotland in that there are a few points, namely the blue and teal, that don't line up with the other countries. Looking at pairwise plots, those that follow the diagonal in the plot means that the x and y (the two countries of comparison) have a similar correlation for that data. To transpose means to sway the x and y data, so swapping the column and rows.

PCA

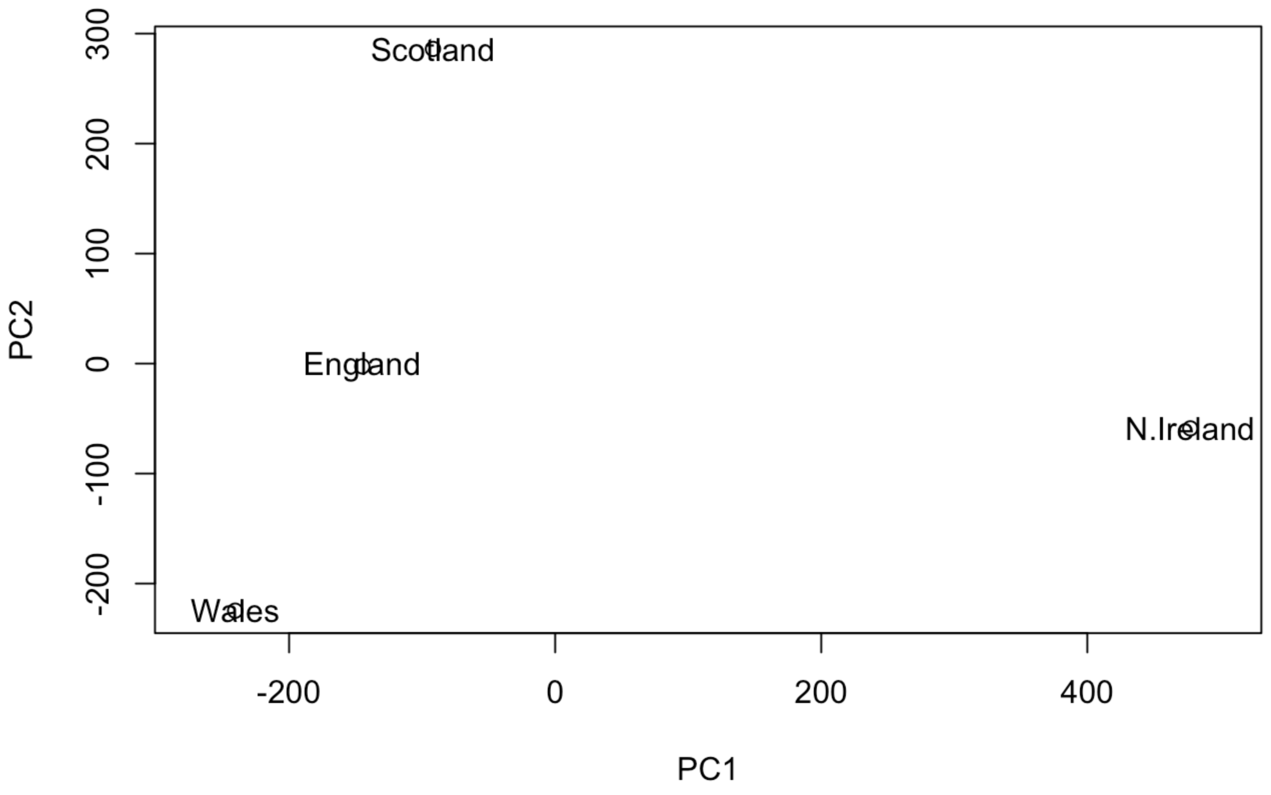
We will perform PCA in R using the base R `prcomp()` function. This function expects observations to be rows and variables to be columns, so we need to transpose our data.frame matrix with the `t()` transpose function.

```r
pca <- prcomp(t(x))
summary(pca)
```

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 2.921e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

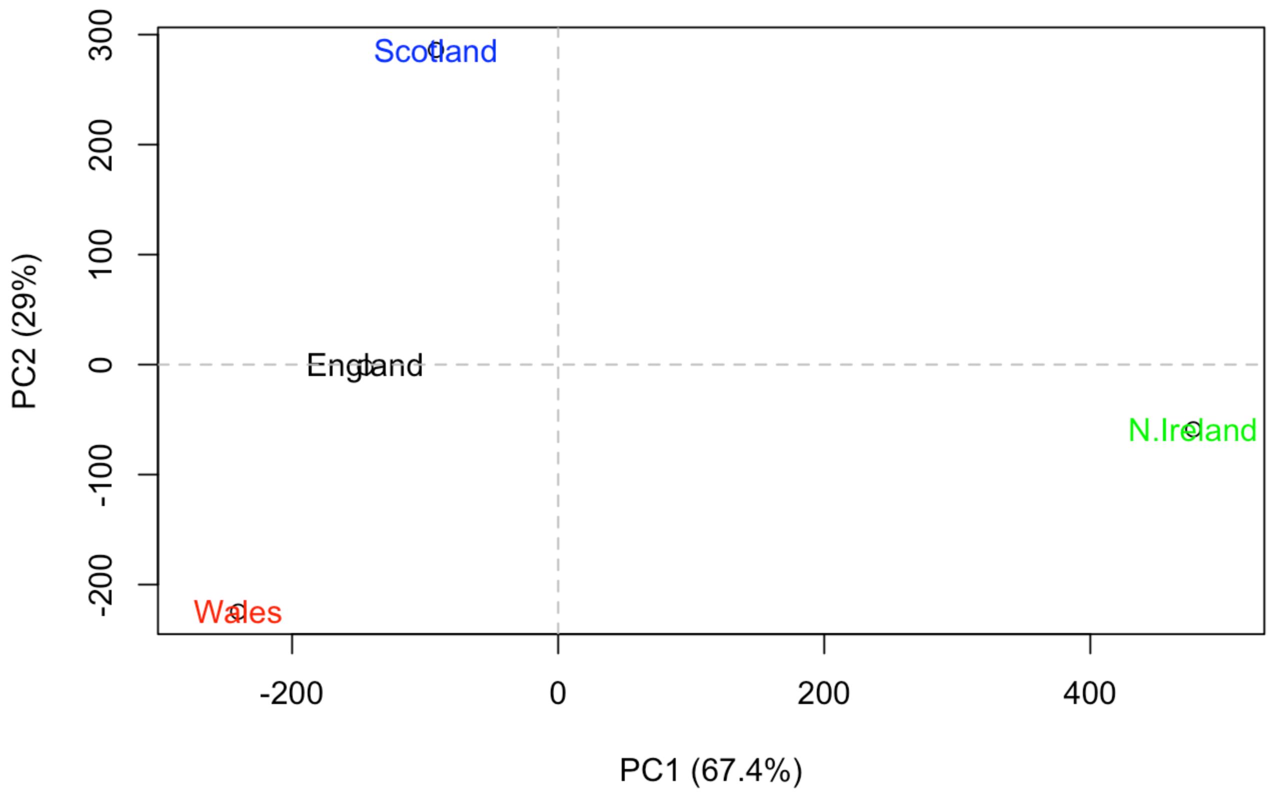
Plot PC1 and PC2 against each other.

```r
#"x" in pca data.frame contains the countries and PC info.
plot(pca$x[,1], pca$x[,2], xlab= "PC1", ylab= "PC2", xlim= c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```



Customize this plot so the colors of the country names match the colors in our UK and Ireland map and table at the start of the document.

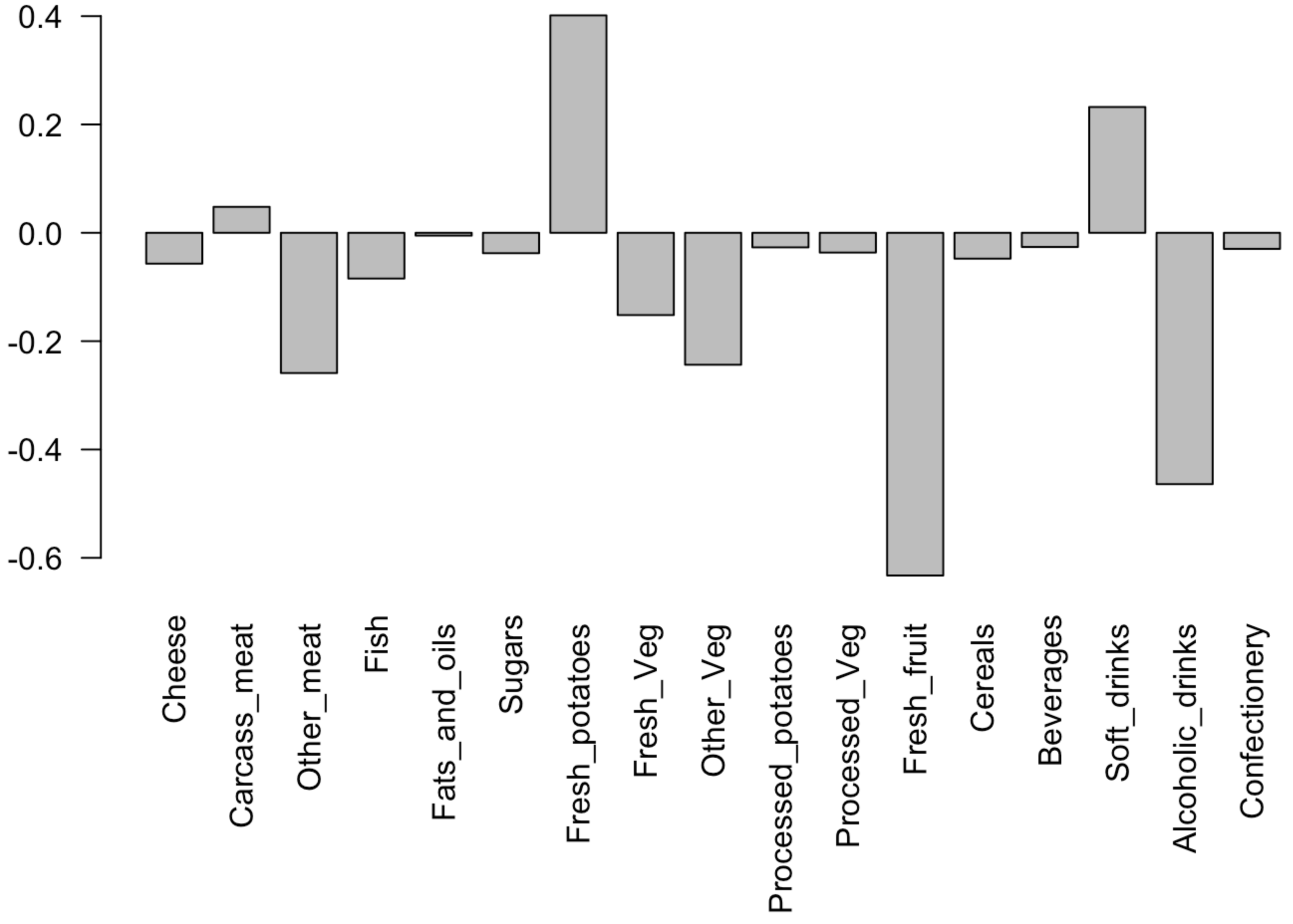
```r
plot(pca$x[,1], pca$x[,2], xlab= "PC1 (67.4%)", ylab= "PC2 (29%)", xlim= c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x), col = c("black", "red", "blue", "green"))
abline(h=0, col="gray", lty="dashed")
abline(v=0, col="gray", lty="dashed")
```



Variable Loadings plot

We can also consider the influence of each of the original variables upon the principal components (typically known as loading scores). This information can be obtained from the prcomp() returned $rotation component. It can also be summarized with a call to biplot(), see below:

```
## 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 )
```



```r
pca$rotation
```

```
                            PC1          PC2         PC3          PC4
Cheese             -0.056955380  0.016012850  0.02394295 -0.409382587
Carcass_meat        0.047927628  0.013915823  0.06367111  0.729481922
Other_meat         -0.258916658 -0.015331138 -0.55384854  0.331001134
Fish               -0.084414983 -0.050754947  0.03906481  0.022375878
Fats_and_oils      -0.005193623 -0.095388656 -0.12522257  0.034512161
Sugars             -0.037620983 -0.043021699 -0.03605745  0.024943337
Fresh_potatoes      0.401402060 -0.715017078 -0.20668248  0.021396007
Fresh_Veg          -0.151849942 -0.144900268  0.21382237  0.001606882
Other_Veg          -0.243593729 -0.225450923 -0.05332841  0.031153231
Processed_potatoes -0.026886233  0.042850761 -0.07364902 -0.017379680
Processed_Veg      -0.036488269 -0.045451802  0.05289191  0.021250980
Fresh_fruit        -0.632640898 -0.177740743  0.40012865  0.227657348
Cereals            -0.047702858 -0.212599678 -0.35884921  0.100043319
Beverages          -0.026187756 -0.030560542 -0.04135860 -0.018382072
Soft_drinks         0.232244140  0.555124311 -0.16942648  0.222319484
Alcoholic_drinks   -0.463968168  0.113536523 -0.49858320 -0.273126013
Confectionery      -0.029650201  0.005949921 -0.05232164  0.001890737
```