class 7: Clustering and PCA

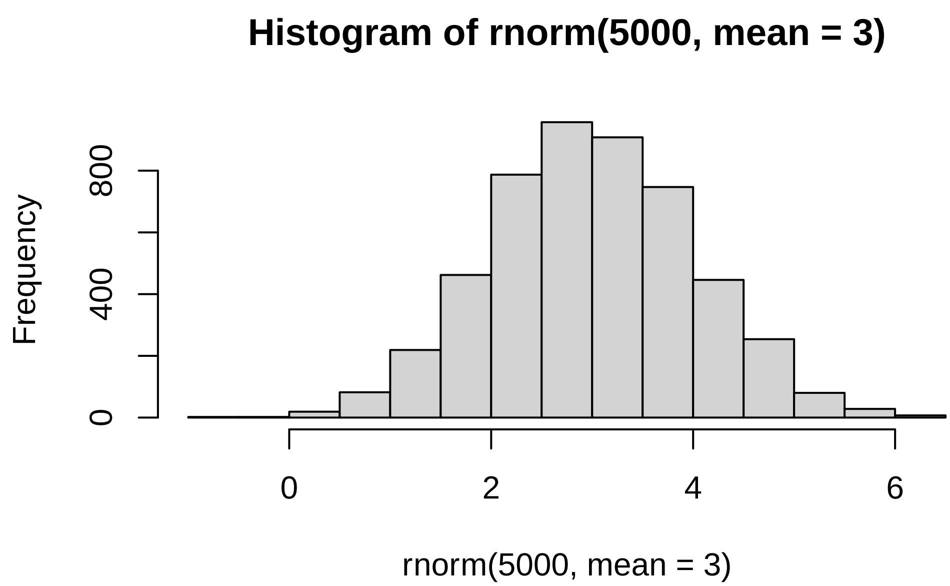
Peter Shamasha (A15857589)

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 and another 30 points with a mean of -3.

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

```
 [1]  3.5617549  3.6935195  2.6382110  1.5768841  4.3655170  2.4991049
 [7]  4.0905280  3.1994779  2.0924390  3.1631631  2.0096473  4.0840262
[13]  5.8514993  3.7235685  2.5908186  2.5172583  4.5596933  3.3868127
[19]  4.0320453  3.4794301  3.0544710  2.2554072  2.5884035  3.6314284
[25]  2.2225646  4.3867547  4.2803475  3.9433191  3.0988989  3.4537905
[31] -2.2003711 -2.4677845 -1.5232410 -1.9211791 -2.9242754 -2.6974960
[37] -0.7407271 -2.8752775 -3.6092241 -2.2874363 -2.0842908 -1.5648450
[43] -3.0788967 -2.3649978 -4.1283373 -4.3511755 -4.2594955 -3.9212639
[49] -3.6163633 -1.9657994 -2.6503601 -2.7880934 -1.2683491 -3.0386432
[55] -1.7253671 -4.0280694 -4.0617006 -2.9251508 -3.2116063 -3.8038745
```

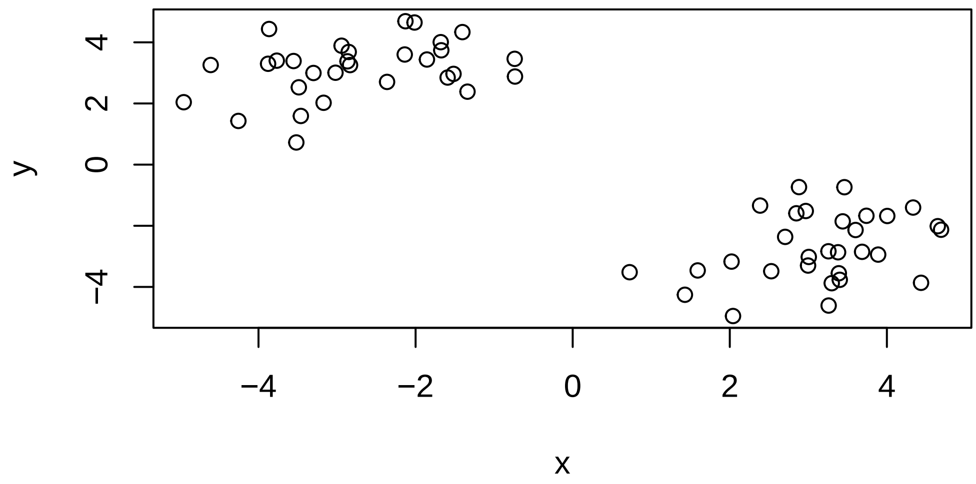
make a temporary variable of this cluster.

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

```
 [1]  3.8890609  2.9966417  1.5913740  0.7249137  2.7048372  3.4374723
 [7]  3.4003594  3.2979833  4.0049838  4.6900138  3.0057363  4.3338751
[13]  3.2585852  3.6844297  4.4348843  3.2558181  2.5281159  2.8474972
[19]  3.7393233  3.3787163  2.0225599  3.6012814  3.3883243  1.4286879
[25]  2.8810152  2.9672298  2.0411254  4.6477897  2.3861486  3.4593901
[31] -0.7384297 -1.3393308 -2.0138071 -4.9519277 -1.5169901 -0.7344876
[37] -4.2559978 -3.5535860 -2.1381285 -3.1712388 -2.8670861 -1.6730475
[43] -1.5921442 -3.4866079 -2.8343238 -3.8650414 -2.8515407 -4.6088043
[49] -1.4039963 -3.0199418 -2.1293566 -1.6798288 -3.8780626 -3.7672410
[55] -1.8557891 -2.3631356 -3.5184950 -3.4608738 -3.3002876 -2.9425536
```

Put two of these together

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



K-means Clusturing

Very popular clustering method that we can use 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  3.134272 -2.717069
2 -2.717069  3.134272

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 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] 61.67553 61.67553
 (between_SS / total_SS =  89.3 %)
```

```
Available components:

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

Q. What 'component' of your result object details

- cluster size?

```
km$size
```

```
[1] 30 30
```

- cluster assignment/membership?

```
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 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
```

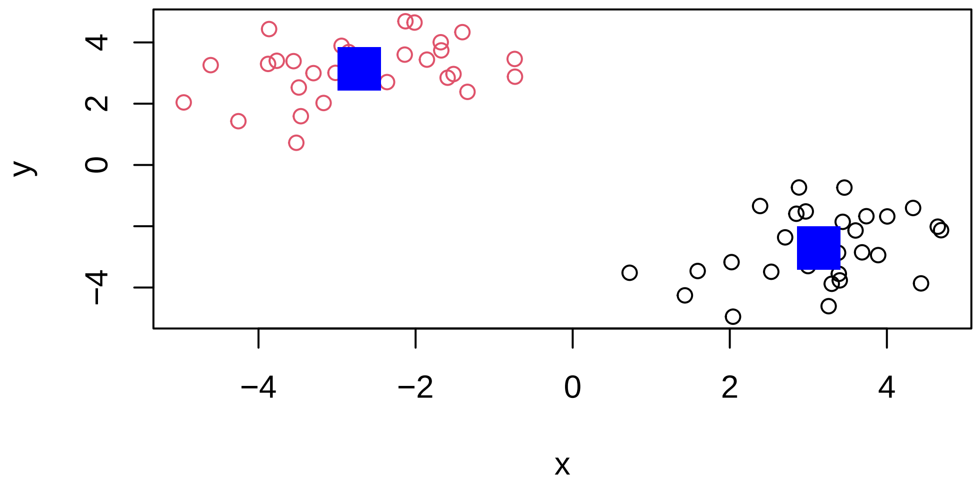
- cluster center?

```
km$centers
```

```
          x         y
1  3.134272 -2.717069
2 -2.717069  3.134272
```

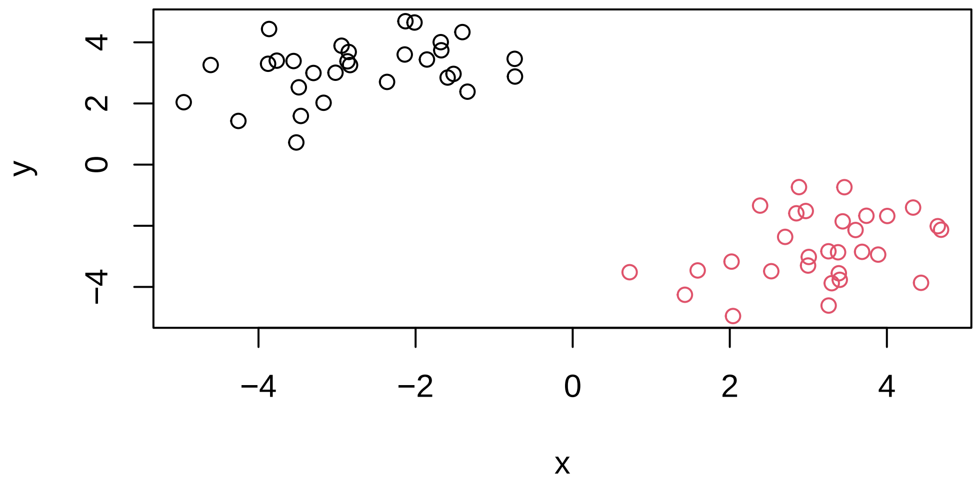
> Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```
plot(x, col= km$cluster)
points(km$centers, col="blue", pch=15, cex=3)
```



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

```
km3 <- kmeans(x, centers = 2)
plot(x, col=km3$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.

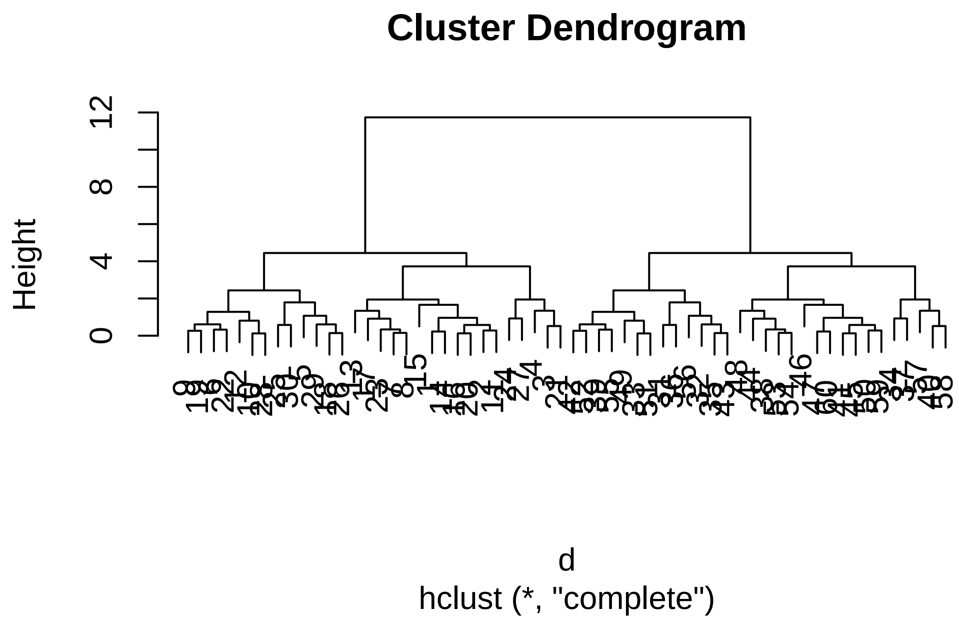
```
d <- dist(x)
hc <- hclust(d)
hc
```

```

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=8)
grps
```

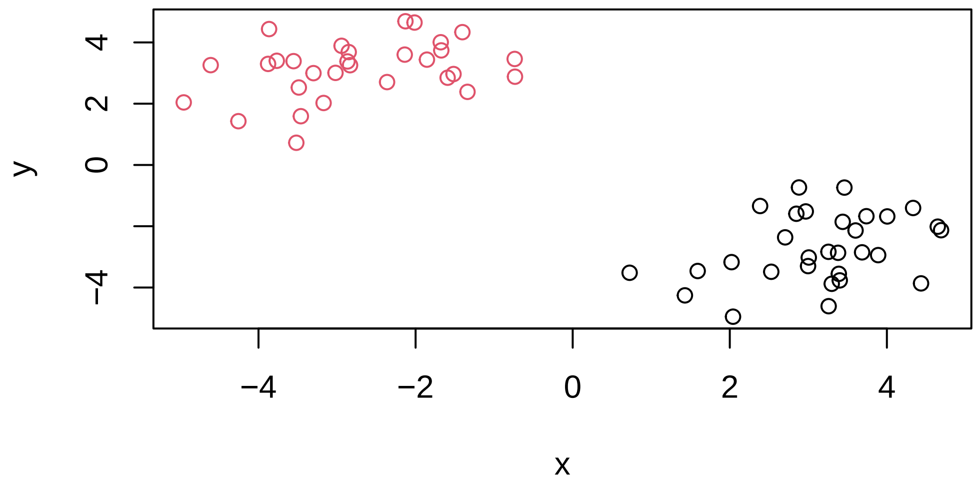
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

You can also tell ‘cutree()’ to cut where it 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 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
```

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



Principal Component Analysis (PCA)

```r
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
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
Fresh_potatoes          720   874      566      1033
Fresh_Veg               253   265      171       143
Other_Veg               488   570      418       355
Processed_potatoes      198   203      220       187
Processed_Veg           360   365      337       334
Fresh_fruit            1102  1137      957       674
Cereals                1472  1582     1462      1494
```

```
Beverages                 57    73      53       47
Soft_drinks             1374  1256    1572     1506
Alcoholic_drinks         375   475     458      135
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?

```r
## Complete the following code to find out how many rows and columns are in x?
dim(x)
```

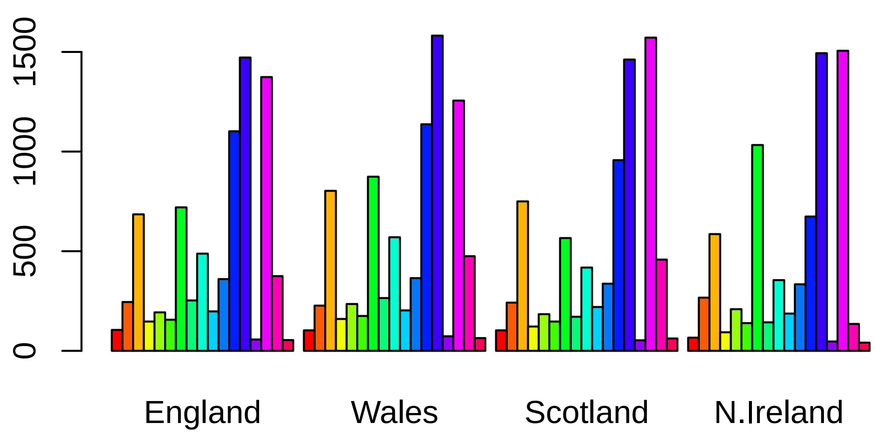
```
[1] 17  4
```

> 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 the 'row.names' approach to solving the 'row-names problem because it is more robust and it will not lead to problems with the data set if the code is run multiple times.

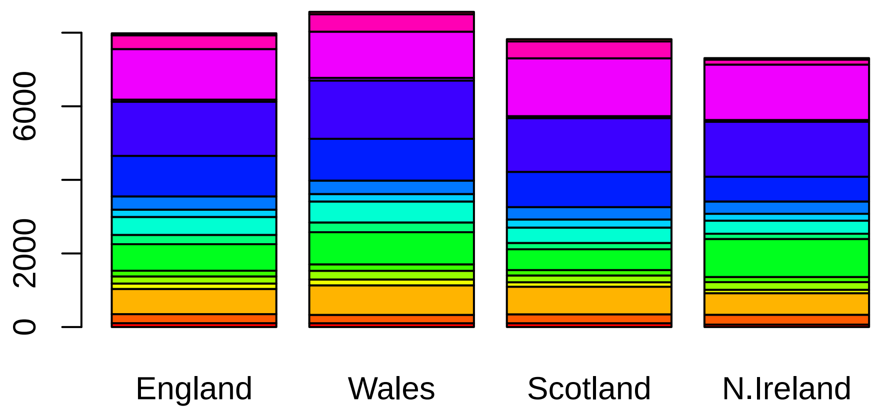
Plots of the UK foods

```r
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?

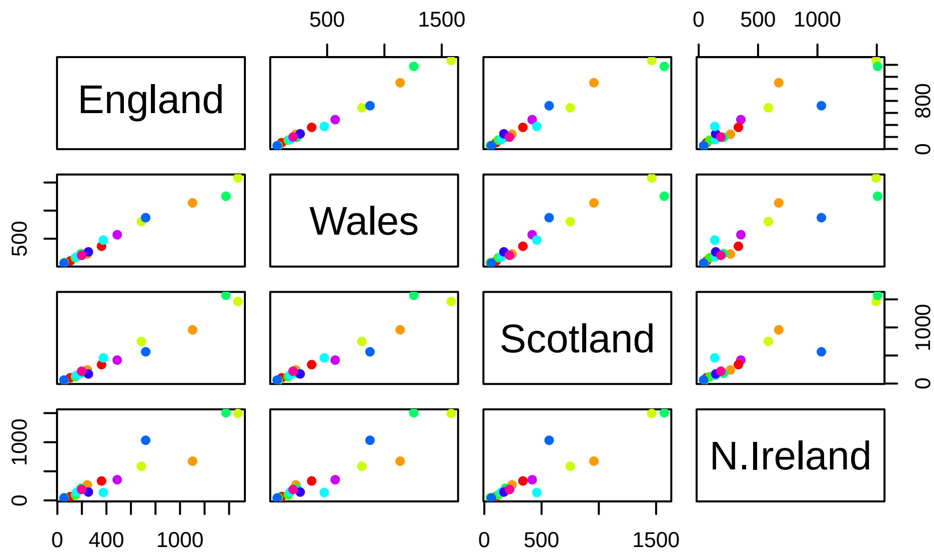
```r
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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?

If a given point lies on the diagonal line for a given plot, it means that both countries have the same value for that category.

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



The main PCA function in base R is called 'prcomp()' it expects the transpose of our data.

```
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"
```

```r
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
```

```r
plot(pca$x[,1], pca$x[,2],
     col=c("orange", "red", "blue", "darkgreen"),
     pch=16)
```