Class 7:Clustering and PCA

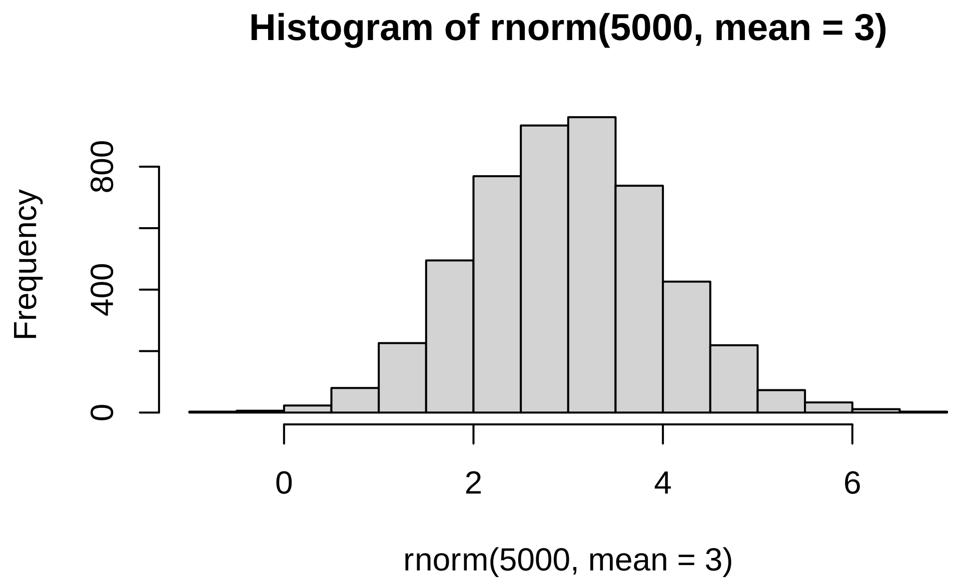
Moises (A17579866)

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 `nnorm()` function to get random numbers from a normal distribution around a given `mean`.

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



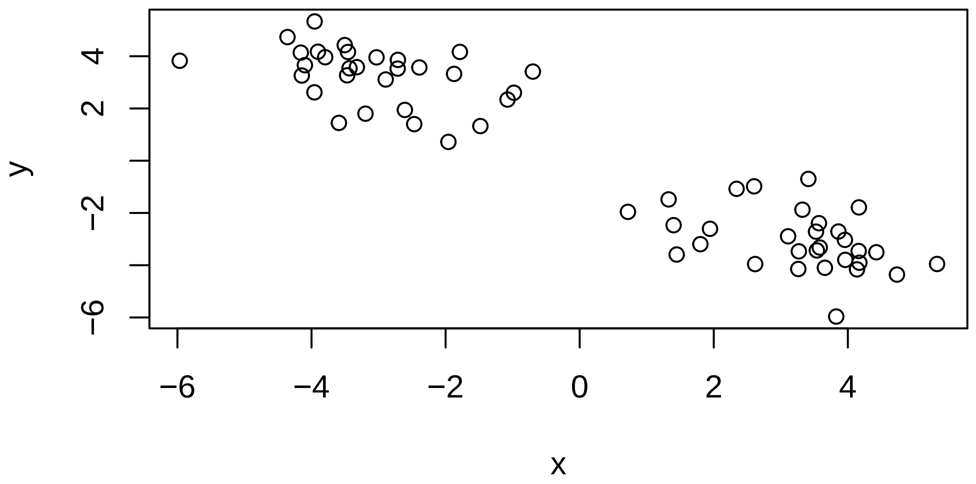
Let's get 30 points with a mean of 3.

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

```
 [1]  3.5258216  3.8598773  0.7200426  1.8006055  4.1619704  1.4009352
 [7]  4.7327565  5.3311541  2.3406185  4.4253643  3.5693945  4.1719440
[13]  4.1651466  2.6170181  4.1365289  3.9619569  3.4113958  3.5376606
[19]  3.1092622  3.3228166  1.3265592  3.6576219  3.2689356  3.9563749
[25]  3.5809772  1.9439144  2.6024144  1.4473374  3.2610374  3.8266300
[31] -5.9658009 -4.1448670 -3.5913908 -0.9806953 -2.6074540 -3.3234603
[37] -3.0302575 -3.4689362 -4.0987446 -1.4812325 -1.8735499 -2.8933524
[43] -3.4335849 -0.6978061 -3.7961236 -4.1597202 -3.9567422 -1.7867942
[49] -3.9039461 -2.3912749 -3.5040650 -1.0763579 -3.9534726 -4.3582337
[55] -2.4678032 -3.4563289 -3.1951988 -1.9579912 -2.7115182 -2.7149464
```

Put two of these together:

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



K-means clusterning.

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

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

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

Cluster means:
          x         y
1 -3.032722  3.239136
2  3.239136 -3.032722

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] 75.03941 75.03941
 (between_SS / total_SS =  88.7 %)

Available components:

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

> Q. What 'component' of your result object details

- cluster size?

```r
km$size
```

```
[1] 30 30
```

- cluster assignment/membership?

```r
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
```

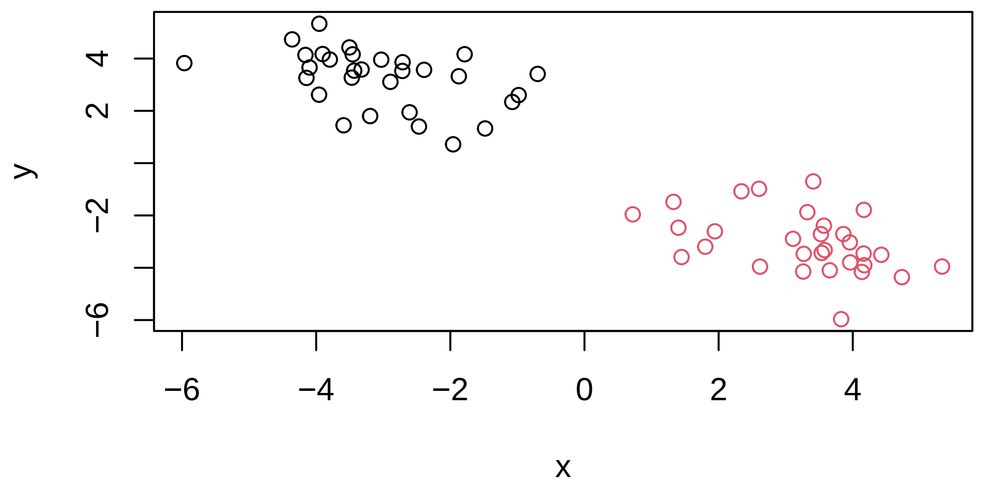
- cluster center?

```
km$centers
```

```
          x         y
1 -3.032722  3.239136
2  3.239136 -3.032722
```

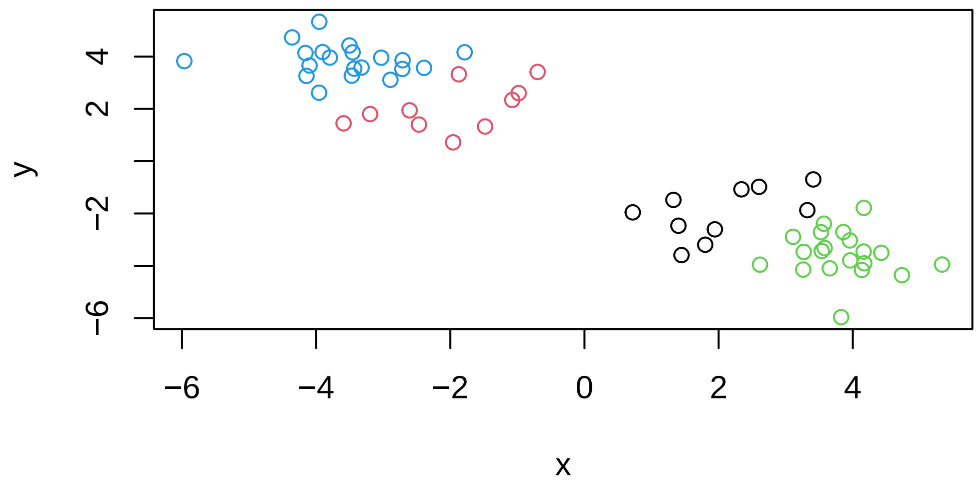
Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

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



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

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



Hierarchical Clustering

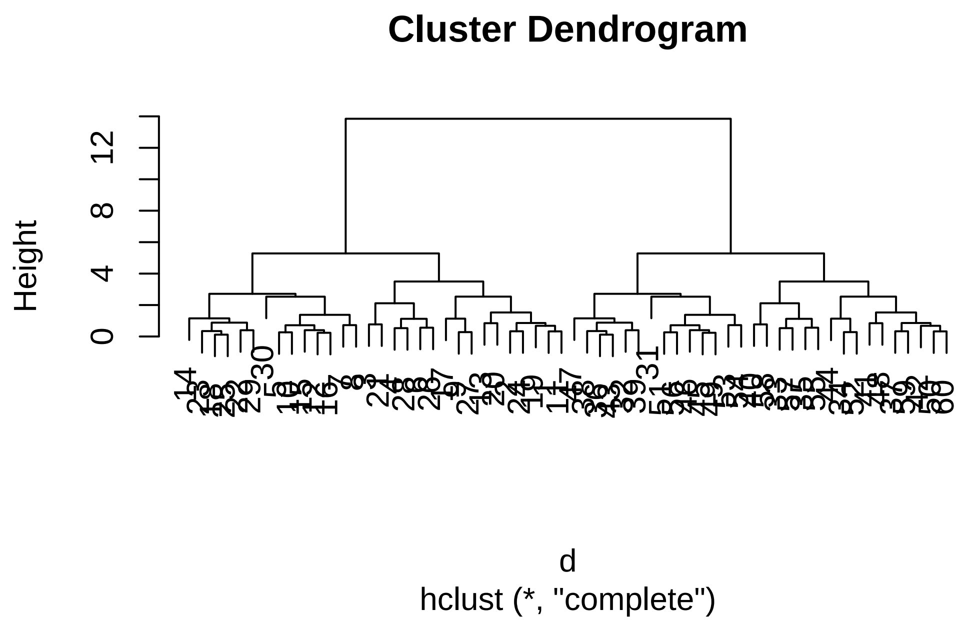
We can use the `hclust()` function for Hierarchical Clustering. Unlike `kmeans()`, where we could just pass in our data as input, we need to give `hclust()` a "distance matrix".

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)
```

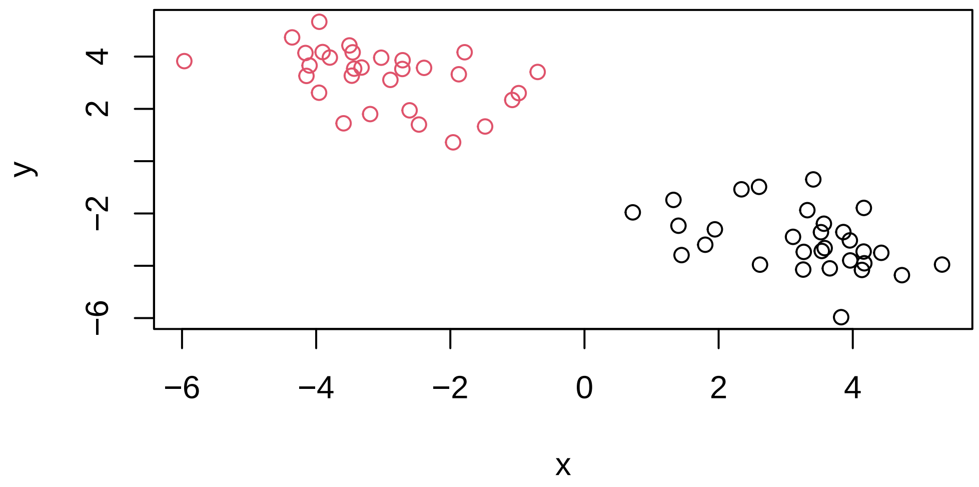


I can now “cut” my tree with the `cutree()` to yield a cluster membership vector.

```
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 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)
```



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

```r
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
```

1. PCA of UK food data

Data import

First we will read the provided `UK_foods.csv` input file (note we can read this directly from the following tinyurl short link: "https://tinyurl.com/UK-foods".

```r
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?

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

```
[1] 17
```

```r
ncol(x)
```

```
[1] 4
```

```r
# You can use the **dim()** function, which returns the number of rows and columns or the
dim(x)
```

```
[1] 17  4
```

Checking your data

It is always a good idea to examine your imported data to make sure it meets your expectations. At this stage we want to make sure that no odd things have happened during the importing phase that will come back to haunt us later.

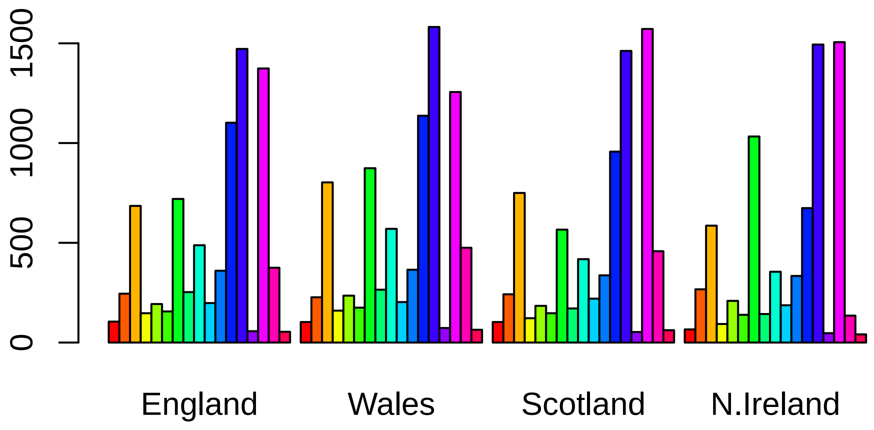
For this task we can use the `View()` function to display all the data (in a new tab in RStudio) or the `head()` and `tail()` functions to print only a portion of the data (by default 6 rows from either the top or bottom of the dataset respectively).

```r
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
```

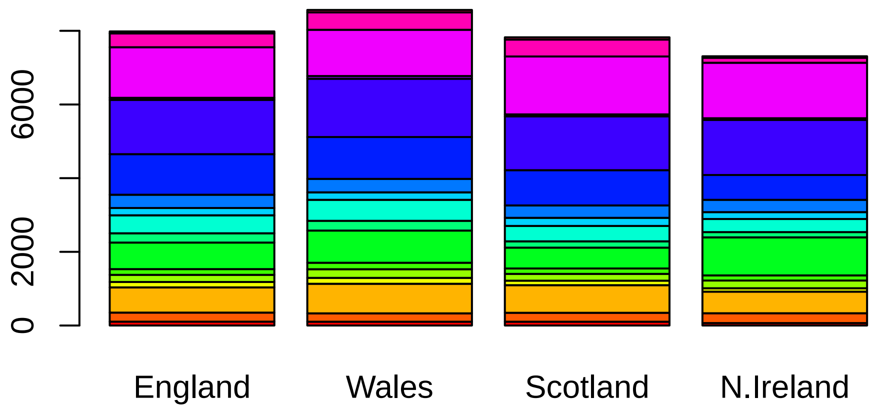
> Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

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



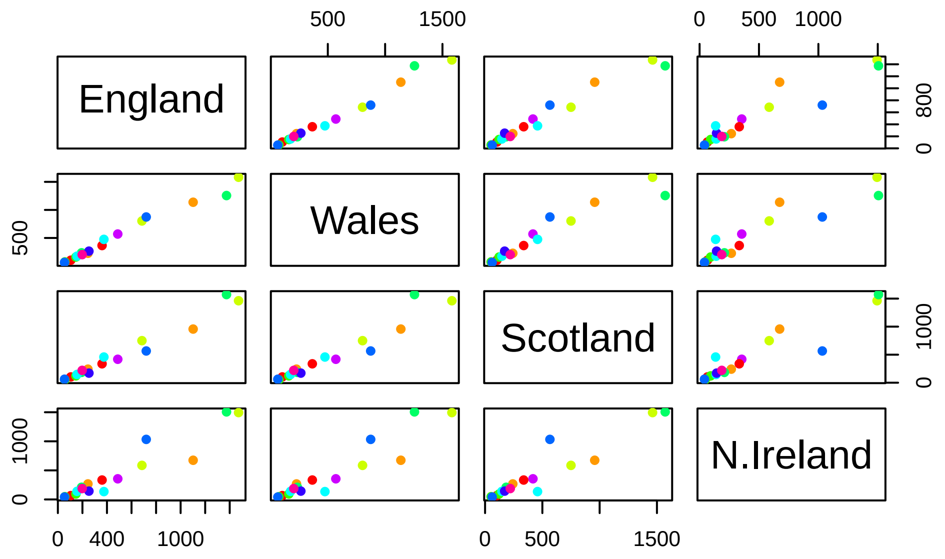
Q3: Changing what optional argument in the above barplot() function results in the following plot?

```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?

```
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?

The higher dots on the diagonal line is representing the y-axis and the dots ranging left and right is representing the x-axis.

The main PCA function in base R is called `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 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
```

```
attributes(pca)
```

```
$names
```

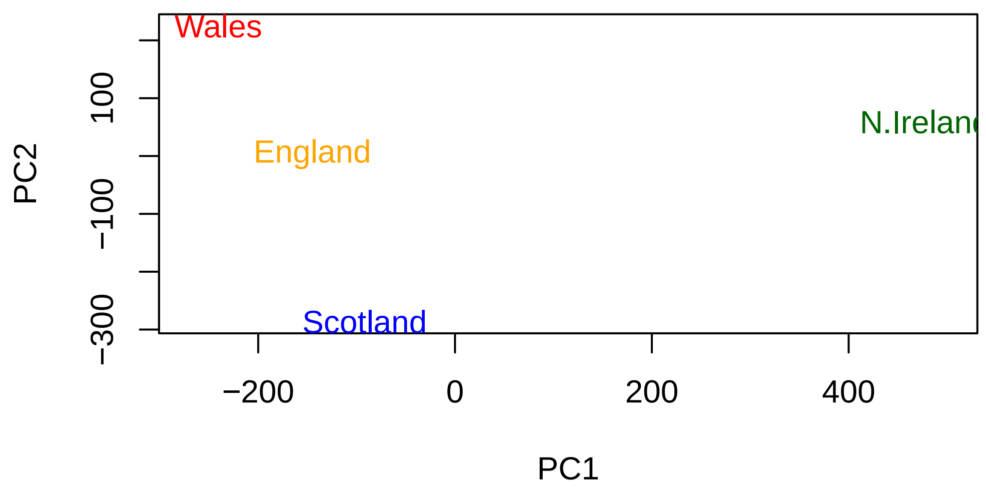
```
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```r
pca$x
```

```
                PC1         PC2         PC3           PC4
England   -144.99315    2.532999 -105.768945  1.042460e-14
Wales     -240.52915  224.646925   56.475555  9.556806e-13
Scotland   -91.86934 -286.081786   44.415495 -1.257152e-12
N.Ireland  477.39164   58.901862    4.877895  2.872787e-13
```

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

```r
plot(pca$x[,1], pca$x[,2],xlab="PC1", ylab="PC2", xlim=c(-270,500), col="transparent",
     pch=16)
text(pca$x[,1], pca$x[,2], colnames(x), col=c("orange", "red", "blue", "darkgreen"))
```


Wales
England
N.Ireland
Scotland
PC2
PC1