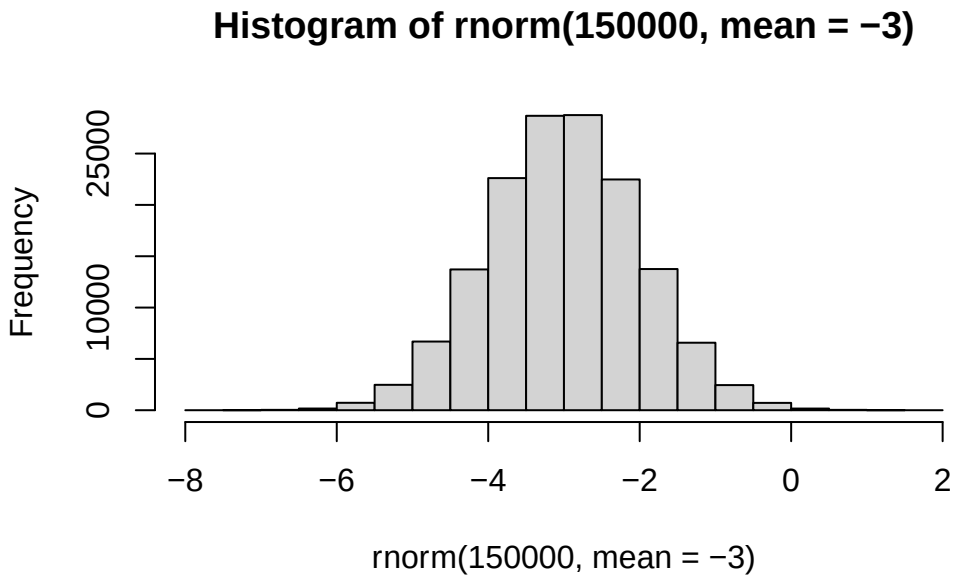
Class07: Machine Learning 1

Caliope Marin (PID: A13912583)

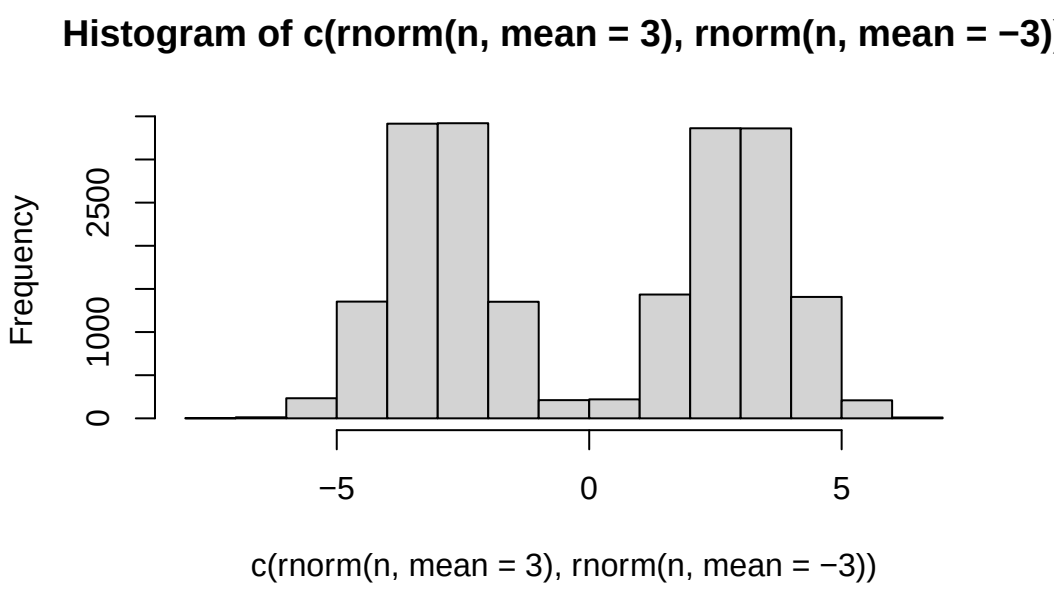
Before we get into clustering methods lets make some sample data to cluster where we know that the answer should be.

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

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




```
n=10000
hist( c(rnorm(n, mean=3), rnorm(n, mean=-3)))
```




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

```
 [1]  3.922253  2.965511  2.469500  3.991739  2.091786  3.039838  1.064168
 [8]  2.296719  2.066480  3.090307  3.429873  2.224289  2.352459  2.583673
[15]  1.099356  4.715308  2.927814  2.873306  3.800530  3.685667  3.147944
[22]  2.105082  2.114693  2.386030  4.373889  2.336201  2.983011  2.827075
[29]  1.793113  3.801964 -2.862887 -5.019054 -2.113812 -2.293015 -4.141340
[36] -3.564913 -1.954786 -2.041445 -3.013351 -1.334665 -3.079644 -2.833279
[43] -4.181212 -3.683765 -3.272183 -4.841788 -1.182331 -3.246872 -3.513540
[50] -3.397939 -4.547985 -2.656886 -2.618565 -2.644152 -3.410258 -4.043236
[57] -4.044239 -3.869654 -2.877528 -2.016994
```

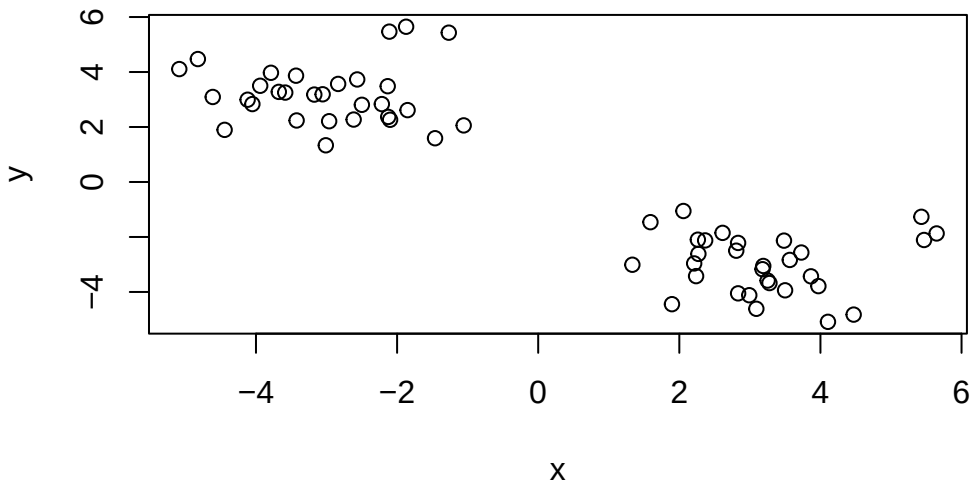
```r
n=30
x <- c(rnorm(n, mean=3), rnorm(n, mean=-3) )
y <- rev(x)

z <- cbind(x, y) #rbind can combine data fram argument by columns and cbind combines by colu
z
```

```
             x         y
 [1,]  3.865529 -3.432437
```

```
 [2,]  2.806934 -2.499096
 [3,]  2.833255 -4.051071
 [4,]  3.277308 -3.676404
 [5,]  5.646820 -1.872162
 [6,]  1.895823 -4.444322
 [7,]  2.057750 -1.056035
 [8,]  4.106484 -5.086992
 [9,]  3.179104 -3.173237
[10,]  3.189735 -3.054988
[11,]  2.237583 -3.423958
[12,]  2.613099 -1.851480
[13,]  2.210526 -2.962924
[14,]  3.251375 -3.585520
[15,]  5.468499 -2.109872
[16,]  4.470871 -4.823418
[17,]  2.270132 -2.615676
[18,]  3.729201 -2.565462
[19,]  5.430649 -1.267484
[20,]  3.498755 -3.940375
[21,]  2.833080 -2.216203
[22,]  3.091458 -4.612671
[23,]  2.366236 -2.125738
[24,]  2.991439 -4.118229
[25,]  2.262598 -2.099114
[26,]  1.590795 -1.462161
[27,]  1.334291 -3.010290
[28,]  3.483346 -2.132984
[29,]  3.565692 -2.835463
[30,]  3.971090 -3.787921
[31,] -3.787921  3.971090
[32,] -2.835463  3.565692
[33,] -2.132984  3.483346
[34,] -3.010290  1.334291
[35,] -1.462161  1.590795
[36,] -2.099114  2.262598
[37,] -4.118229  2.991439
[38,] -2.125738  2.366236
[39,] -4.612671  3.091458
[40,] -2.216203  2.833080
[41,] -3.940375  3.498755
[42,] -1.267484  5.430649
[43,] -2.565462  3.729201
[44,] -2.615676  2.270132
```

```
[45,] -4.823418  4.470871
[46,] -2.109872  5.468499
[47,] -3.585520  3.251375
[48,] -2.962924  2.210526
[49,] -1.851480  2.613099
[50,] -3.423958  2.237583
[51,] -3.054988  3.189735
[52,] -3.173237  3.179104
[53,] -5.086992  4.106484
[54,] -1.056035  2.057750
[55,] -4.444322  1.895823
[56,] -1.872162  5.646820
[57,] -3.676404  3.277308
[58,] -4.051071  2.833255
[59,] -2.499096  2.806934
[60,] -3.432437  3.865529
```

```
plot(z)
```



##K means clustering

The function in base R for k-means clustering is called `k-means()`.

```
km <-kmeans(z, centers=2) # assigns data points to the center data points
#c;uster size is 30 because n is equal to 30,
#cluster vector for each point its measuring which cluster is # closer to the center
```

```
km$centers
```

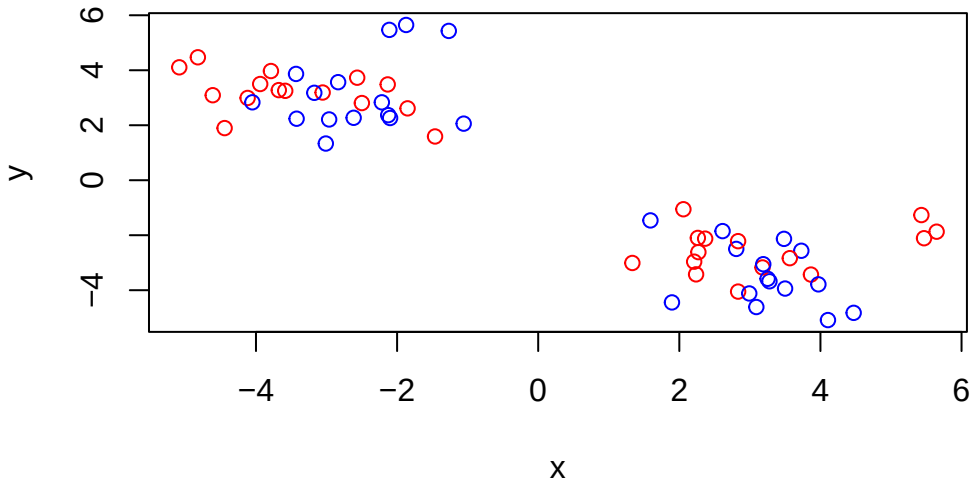
```
          x         y
1 -2.996456  3.184315
2  3.184315 -2.996456
```

Q. Print out the cluster membership vector (i.e our main answer)

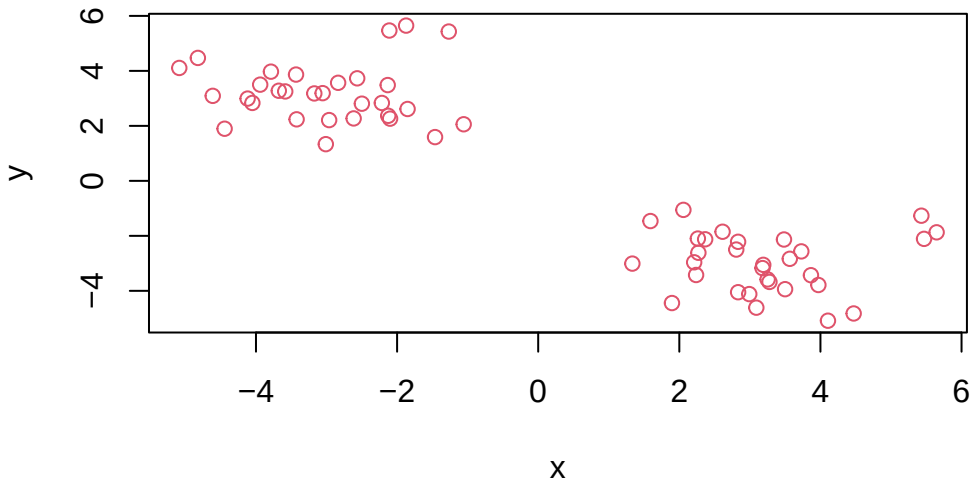
```
km$cluster
```

```
 [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 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
```

```
plot(z, col=c("red", "blue")) #if you do this it will repeat each point
```

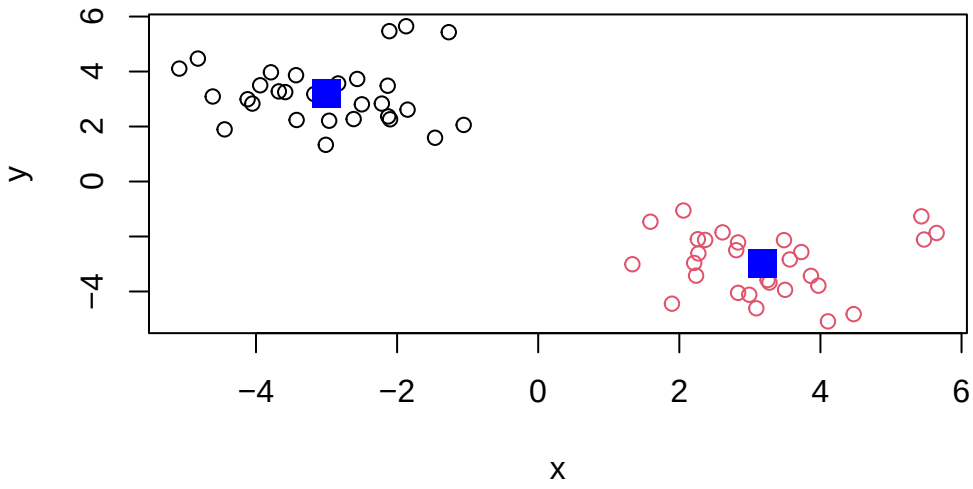


```r
#you can color it by number

plot(z, col=2)
```



Plot with clustering result

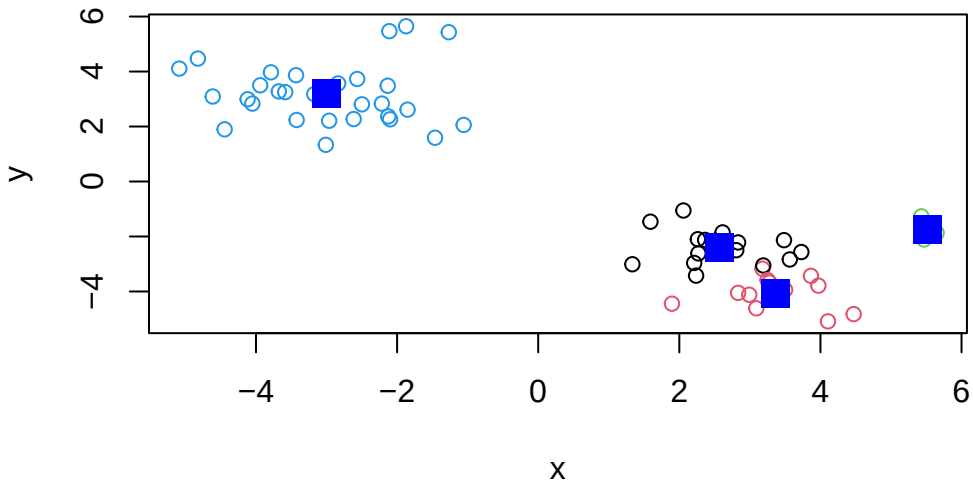
```r
# in order to separate the colors of each cluster
plot(z, col=km$cluster)
points(km$centers, col="blue", pch=15, cex=2) #pch=15 makes a #filled blue square
```




```
#cex stands for character expansion so greater than 1 makes
#it greater thsn the characters
```

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

```
km4 <- kmeans(z, centers = 4)
plot(z, col=km4$cluster)
points(km4$centers, col="blue", pch=15, cex=2)
```



##Hierarchical Clustering

The main function for hierarchical clustering in base R is called `hclust()`

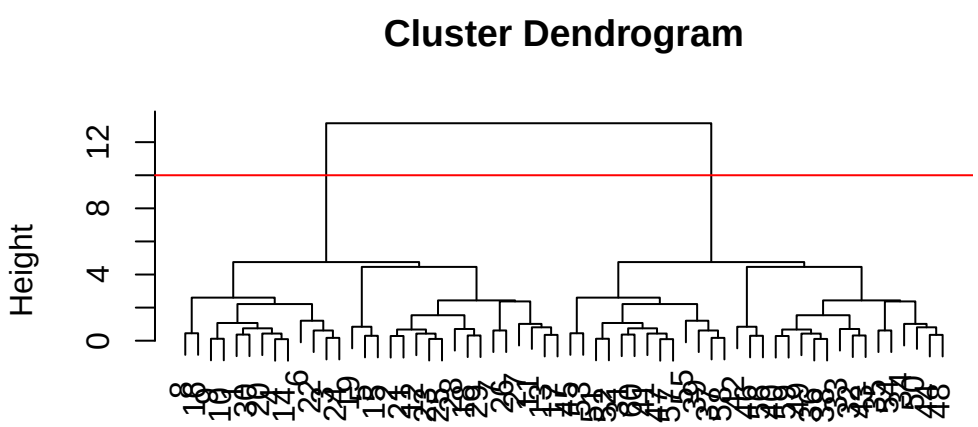
Unlike `kmeans()` I cannot just pass in mt data as input, I first need a distance matrix from my data.

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

```
Call:
hclust(d = d)

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

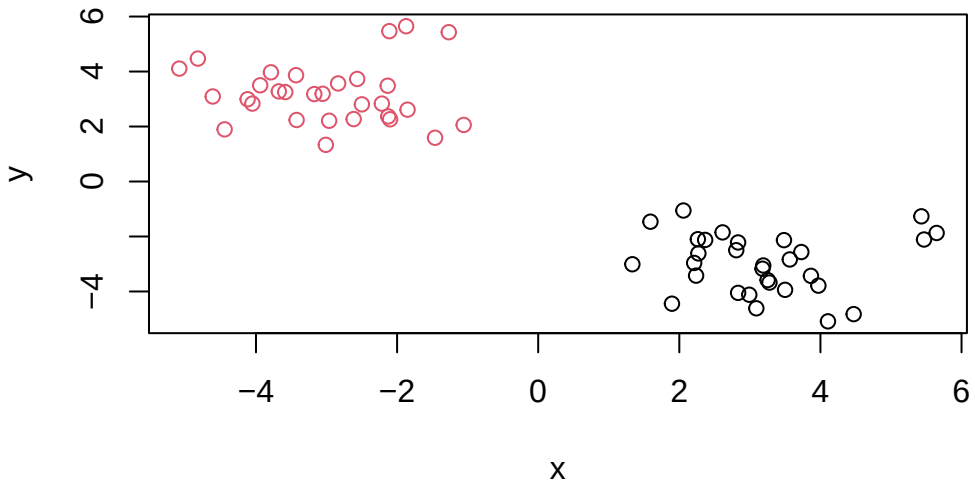
There is a specific hclut plot () method...

```
plot(hc)
#all the numbers from 1-30 are on onside and the other is #above 30
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 `cutreee()`

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



#Principal Component Analysis

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.

PCA of UK food data

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

Q. 1 How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

```
#I can use `nrow()` and `ncol()` to show number of #rows and columns
nrow(x)
```

```
[1] 17
```

```
ncol(x)
```

```
[1] 4
```

##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
```

```
# Note how the minus indexing works
rownames(x) <- x[,1]
x <- x[,-1]
head(x)
```

```
    Wales Scotland N.Ireland
105   103      103        66
245   227      242       267
685   803      750       586
147   160      122        93
193   235      184       209
156   175      147       139
```

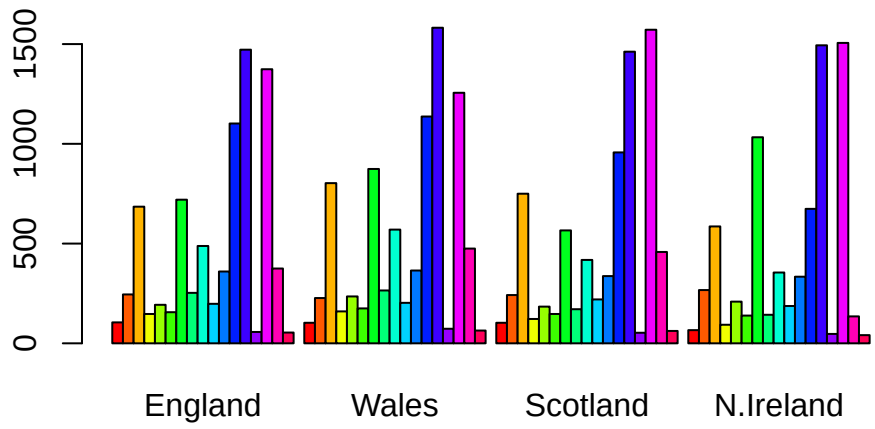
```
dim(x)
```

```
[1] 17  3
```

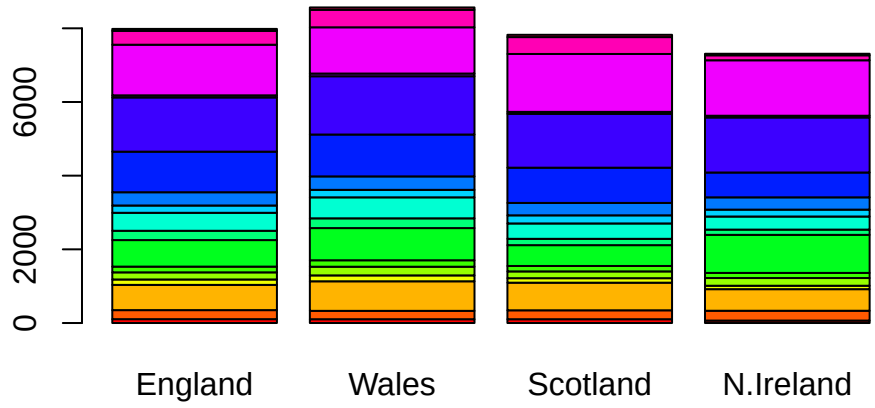
```
x <- read.csv(url, row.names=1)
head(x)
```

```
               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
```

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

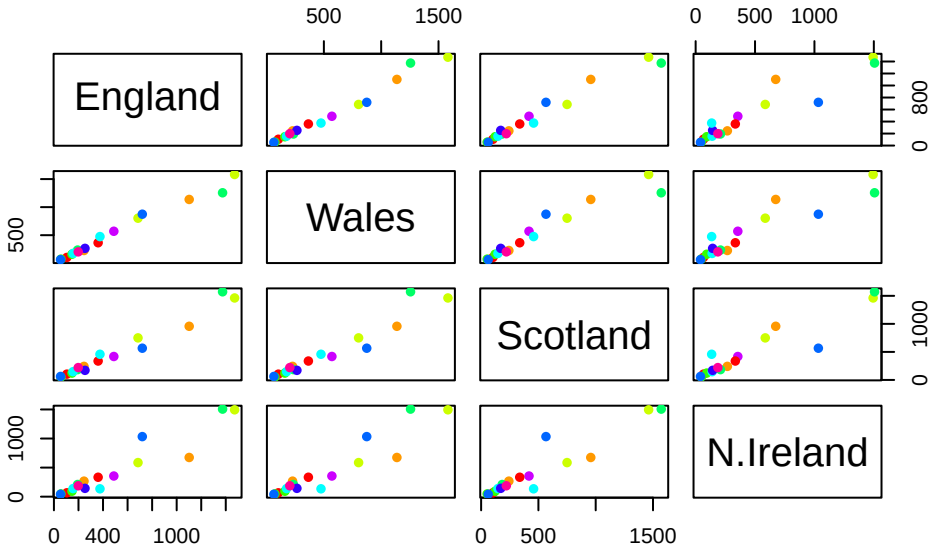



```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x))) #making `beside=F` will
```




```
#make the plot stack all the categories on top of eachother,not pleasant to #look at
```

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



#PCA to the rescue

The main function to do PCA in base R is called `prcomp()`.

Note that I need to take the transpose of this particular data as that is what the `prcomp()` help page was asking for.

```
pca <- prcomp( t(x) ) #t stands for #transpose of 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
```

Whats inside of the object `pca`

```
attributes(pca)
```

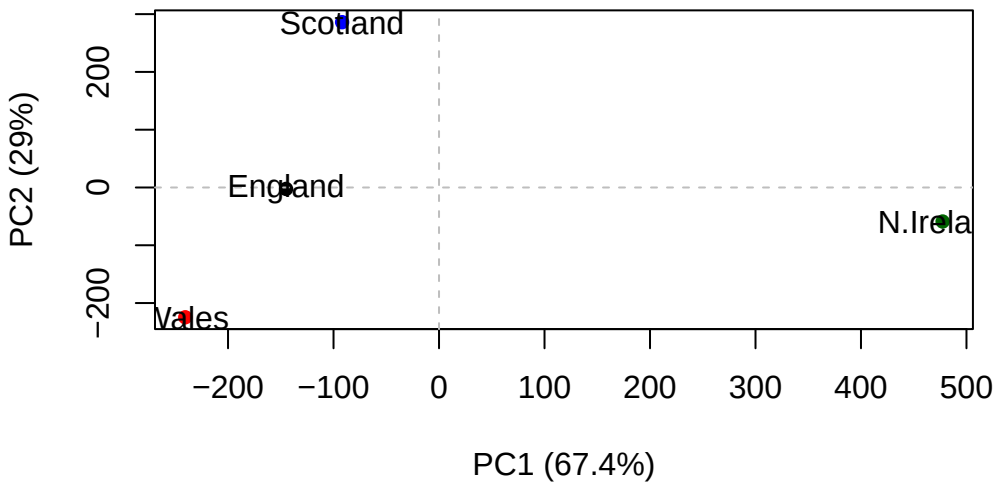
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```
#use x to plot main result figure
pca$x
```

```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -9.152022e-15
Wales     -240.52915 -224.646925 -56.475555  5.560040e-13
Scotland   -91.86934  286.081786 -44.415495 -6.638419e-13
N.Ireland  477.39164  -58.901862  -4.877895  1.329771e-13
```

to make our main result figure, called a "PC plot" (or "score plot", "ordination plot" or "PC1 vs PC2 plot").

```
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2],  col= c( "black", "red", "blue", "darkgreen"), pch=16, xlab="PC1
abline(h=0, col="gray", lty=2)
abline(v=0, col="gray", lty=2)
text(pca$x[,1],pca$x[,2], colnames(x))
```



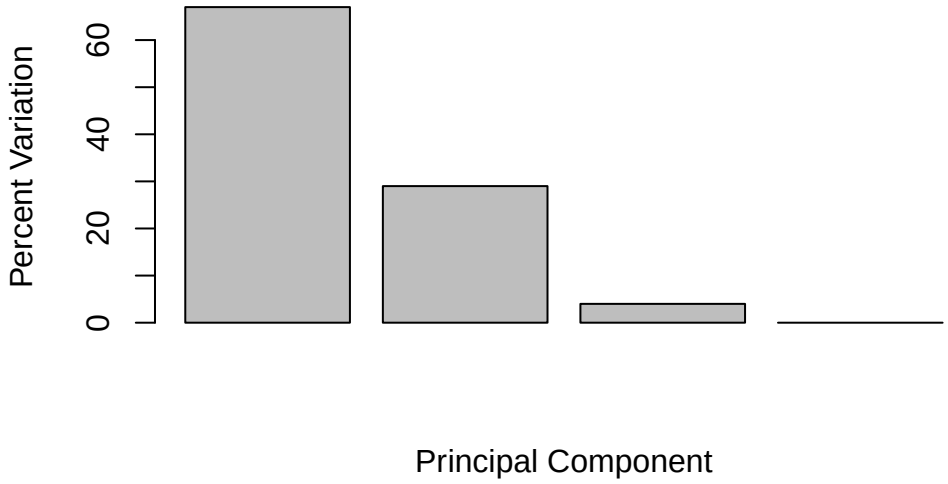

```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

```
[1] 67 29  4  0
```

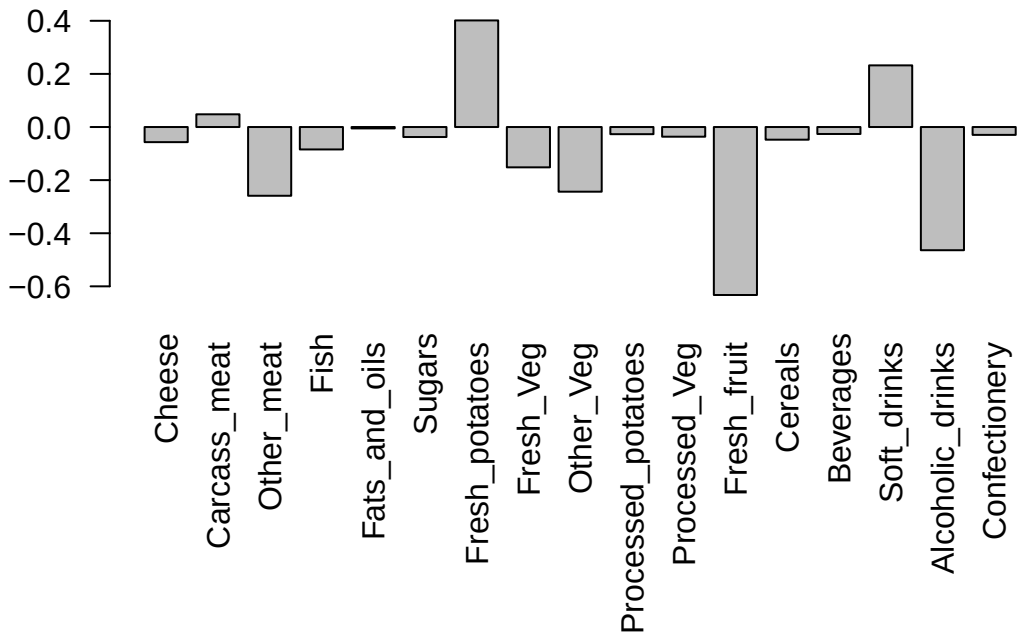
```
## or the second row here...
z <- summary(pca)
z$importance
```

```
                             PC1       PC2      PC3           PC4
Standard deviation     324.15019 212.74780 73.87622 2.921348e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

```r
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



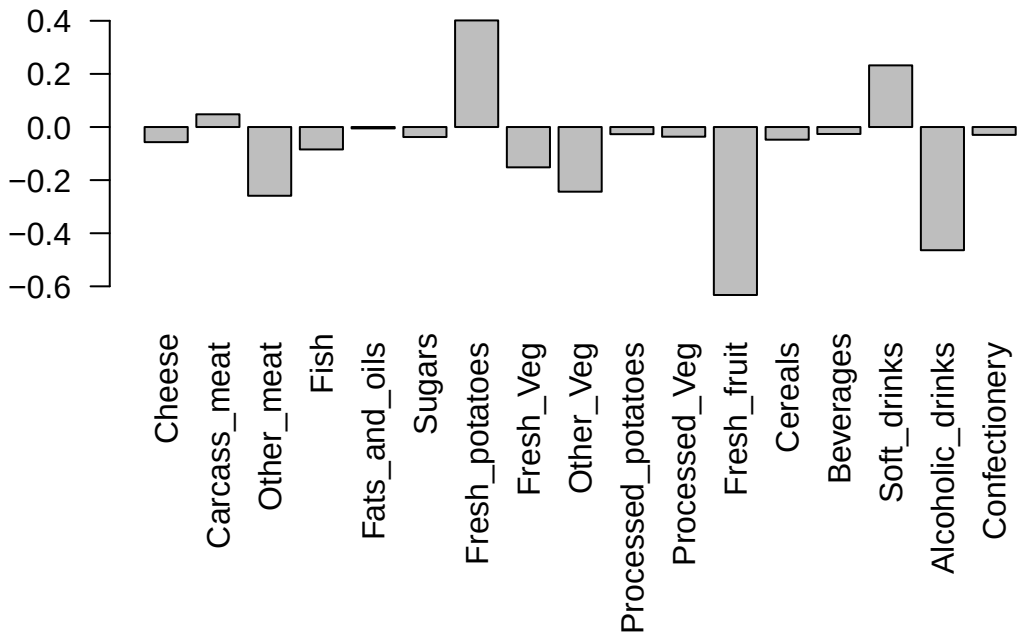

```r
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



#Variable Loadings Plot

Can give us insight on how the original variables (in this case the foods contribute to our new PC axis)

```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```




```
barplot( pca$rotation[,2], las=2)
```