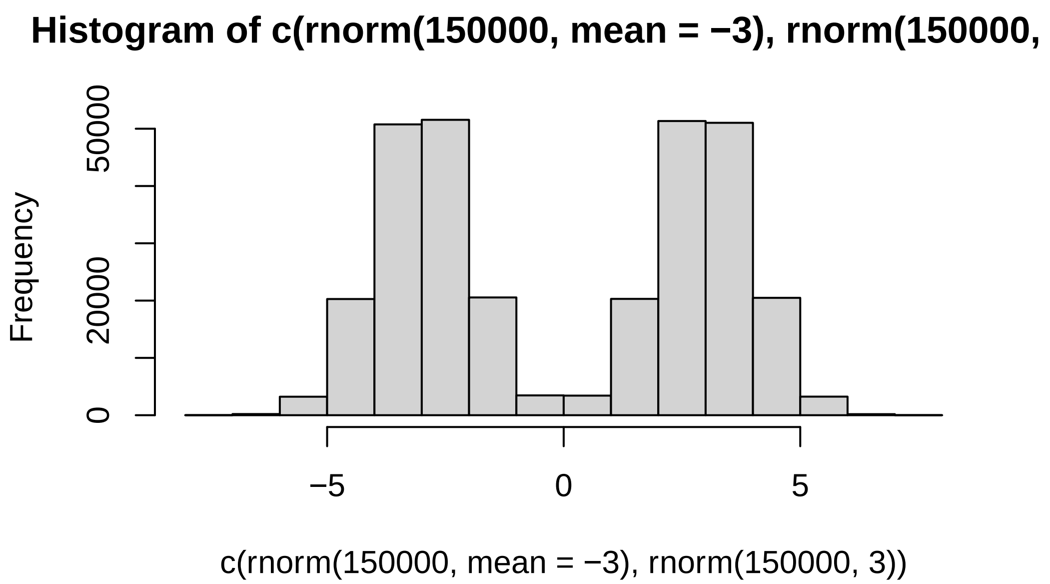
Class 7: Machine Learning 1

Matthew White

Before we get into clustering methods lets make up some sample data to cluster in which we know what the answer should be.

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

```
#combine plots of two different random sets of values clustered around a set mean by vectori
hist(c(rnorm(150000, mean=-3), rnorm(150000, 3)))
```




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

```
 [1] -2.830317262 -1.230943331 -2.057713351 -4.288282629 -2.028794871
 [6] -2.483461772 -1.984912921 -3.635901352 -3.818423960 -3.471964117
[11] -3.339836283 -1.332549748 -2.807758639 -4.366408294 -3.191455667
[16] -1.448907045 -4.456975800 -2.228883602 -2.917929116 -3.918793570
[21] -4.007872984 -2.973662274 -1.430706419 -0.008211708 -4.116393984
[26] -2.762597625 -3.009595277 -4.137147082 -3.203578855 -3.527280965
[31]  2.585187817  2.315797879  2.677233881  3.537877934  4.623835453
[36]  3.393928509  2.820906764  3.711640693  1.806906469  3.038622320
[41]  4.030281310  3.779987974  4.802598073  2.580011734  3.472510547
[46]  2.864600136  3.782767307  1.305641830  3.814645974  3.505728220
[51]  4.123810862  1.288252813  3.804564601  0.860694764  2.471153016
[56]  2.313307466  3.626851866  4.812380746  3.150191096  2.211991687
```

```
#to build up our second sample data cluster, could either rewrite the above code but flipped
y <- rev(x)
y
```

```
 [1]  2.211991687  3.150191096  4.812380746  3.626851866  2.313307466
 [6]  2.471153016  0.860694764  3.804564601  1.288252813  4.123810862
[11]  3.505728220  3.814645974  1.305641830  3.782767307  2.864600136
[16]  3.472510547  2.580011734  4.802598073  3.779987974  4.030281310
[21]  3.038622320  1.806906469  3.711640693  2.820906764  3.393928509
[26]  4.623835453  3.537877934  2.677233881  2.315797879  2.585187817
[31] -3.527280965 -3.203578855 -4.137147082 -3.009595277 -2.762597625
[36] -4.116393984 -0.008211708 -1.430706419 -2.973662274 -4.007872984
[41] -3.918793570 -2.917929116 -2.228883602 -4.456975800 -1.448907045
[46] -3.191455667 -4.366408294 -2.807758639 -1.332549748 -3.339836283
[51] -3.471964117 -3.818423960 -3.635901352 -1.984912921 -2.483461772
[56] -2.028794871 -4.288282629 -2.057713351 -1.230943331 -2.830317262
```

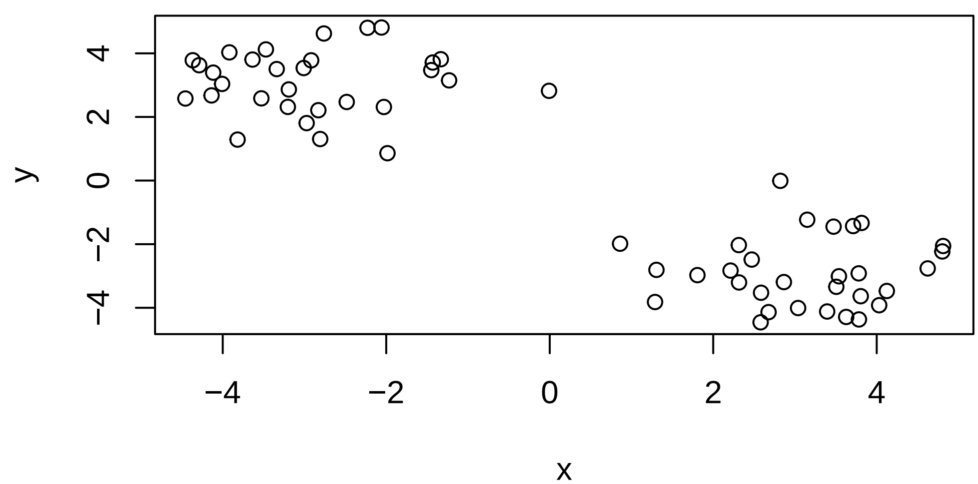
```
#cbind will combine columns of two different vectors/matrices/data frames. basically making t
z <- cbind(x,y)
z
```

```
                x           y
 [1,] -2.830317262  2.211991687
 [2,] -1.230943331  3.150191096
 [3,] -2.057713351  4.812380746
 [4,] -4.288282629  3.626851866
 [5,] -2.028794871  2.313307466
 [6,] -2.483461772  2.471153016
```

```
 [7,] -1.984912921  0.860694764
 [8,] -3.635901352  3.804564601
 [9,] -3.818423960  1.288252813
[10,] -3.471964117  4.123810862
[11,] -3.339836283  3.505728220
[12,] -1.332549748  3.814645974
[13,] -2.807758639  1.305641830
[14,] -4.366408294  3.782767307
[15,] -3.191455667  2.864600136
[16,] -1.448907045  3.472510547
[17,] -4.456975800  2.580011734
[18,] -2.228883602  4.802598073
[19,] -2.917929116  3.779987974
[20,] -3.918793570  4.030281310
[21,] -4.007872984  3.038622320
[22,] -2.973662274  1.806906469
[23,] -1.430706419  3.711640693
[24,] -0.008211708  2.820906764
[25,] -4.116393984  3.393928509
[26,] -2.762597625  4.623835453
[27,] -3.009595277  3.537877934
[28,] -4.137147082  2.677233881
[29,] -3.203578855  2.315797879
[30,] -3.527280965  2.585187817
[31,]  2.585187817 -3.527280965
[32,]  2.315797879 -3.203578855
[33,]  2.677233881 -4.137147082
[34,]  3.537877934 -3.009595277
[35,]  4.623835453 -2.762597625
[36,]  3.393928509 -4.116393984
[37,]  2.820906764 -0.008211708
[38,]  3.711640693 -1.430706419
[39,]  1.806906469 -2.973662274
[40,]  3.038622320 -4.007872984
[41,]  4.030281310 -3.918793570
[42,]  3.779987974 -2.917929116
[43,]  4.802598073 -2.228883602
[44,]  2.580011734 -4.456975800
[45,]  3.472510547 -1.448907045
[46,]  2.864600136 -3.191455667
[47,]  3.782767307 -4.366408294
[48,]  1.305641830 -2.807758639
[49,]  3.814645974 -1.332549748
```

```
[50,]  3.505728220 -3.339836283
[51,]  4.123810862 -3.471964117
[52,]  1.288252813 -3.818423960
[53,]  3.804564601 -3.635901352
[54,]  0.860694764 -1.984912921
[55,]  2.471153016 -2.483461772
[56,]  2.313307466 -2.028794871
[57,]  3.626851866 -4.288282629
[58,]  4.812380746 -2.057713351
[59,]  3.150191096 -1.230943331
[60,]  2.211991687 -2.830317262
```

```r
plot(z)
```



##K-means clustering

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

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

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

```
Cluster means:
          x         y
1 -2.900575  3.103797
2  3.103797 -2.900575

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] 64.79842 64.79842
 (between_SS / total_SS =  89.3 %)

Available components:

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

```
km$centers
```

```
          x         y
1 -2.900575  3.103797
2  3.103797 -2.900575
```

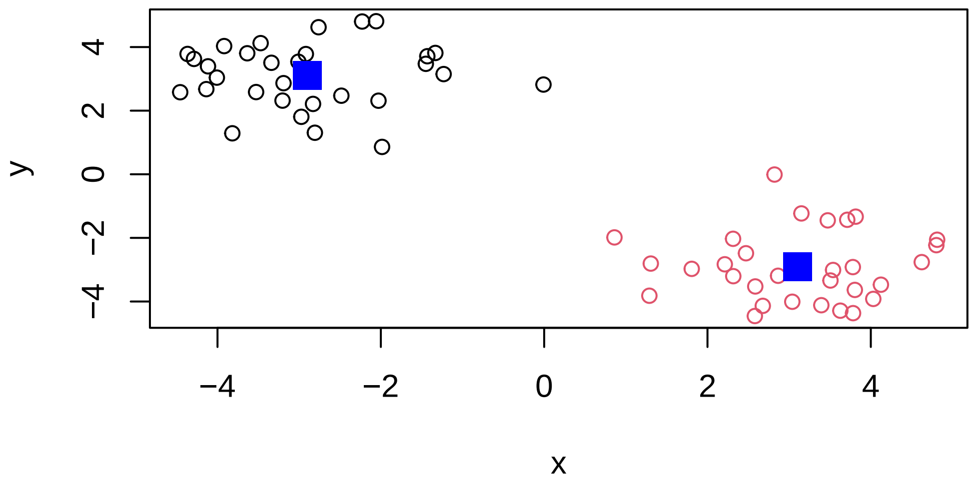
Q: Print out the cluster membership vector (i.e. our main answer)

```
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
```

Plot of data with the clustering result

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



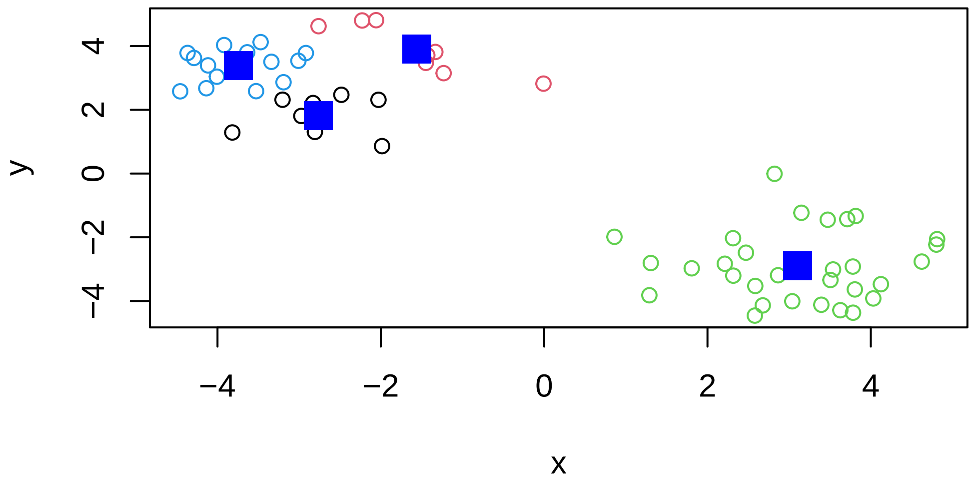
Can you cluster our data in `z` into four clusters?

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

```
K-means clustering with 4 clusters of sizes 8, 8, 30, 14

Cluster means:
          x         y
1 -2.766364  1.821718
2 -1.562564  3.901089
3  3.103797 -2.900575
4 -3.741846  3.380818

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

Within cluster sum of squares by cluster:
[1]  5.116030  8.861647 64.798425  7.136586
 (between_SS / total_SS =  92.9 %)

Available components:
```

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

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




```r
##must be careful with kmeans clustering, as it will return what you ask for. You can force
```

Hierarchical Clustering

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

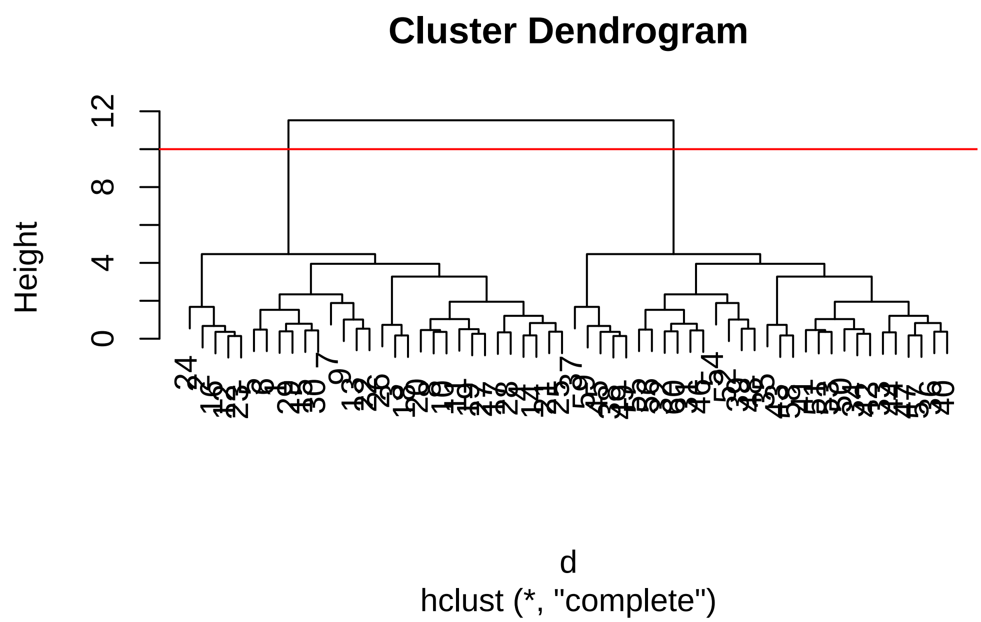
Unlike `kmeans()` I can not just pass in my data as input. I first need a distance matrix from my data.

```r
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

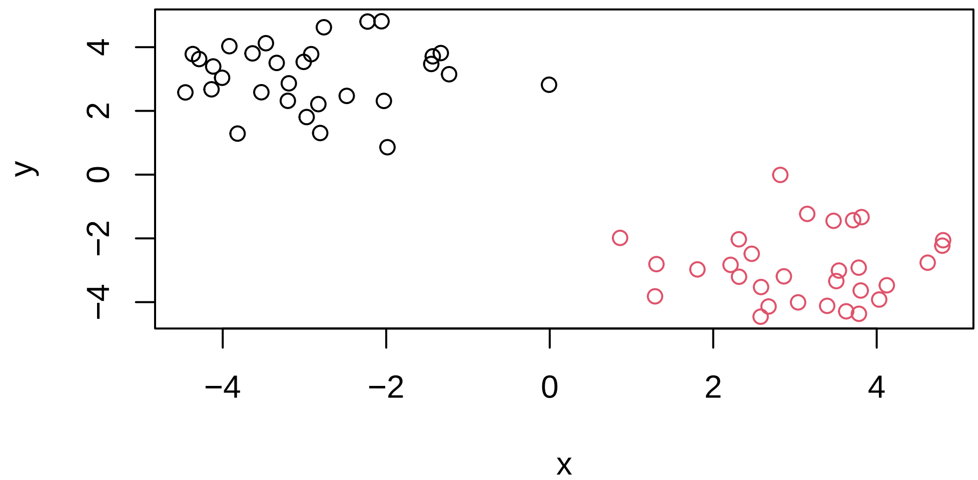
```r
plot(hc)
#To get main clustering result (membership vector), give a height at which to cut the tree a
abline(h=10, col = "red")
```




```r
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
```

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



#Principal Component Analysis

##PCA of UK food data

```
url <- "https://tinyurl.com/UK-foods"
#reading dataset with `row.names = 1` argument to not conisder row names as a column value
x <- read.csv(url, row.names = 1)
dim(x)
```

```
[1] 17  4
```

```
head(x)
```

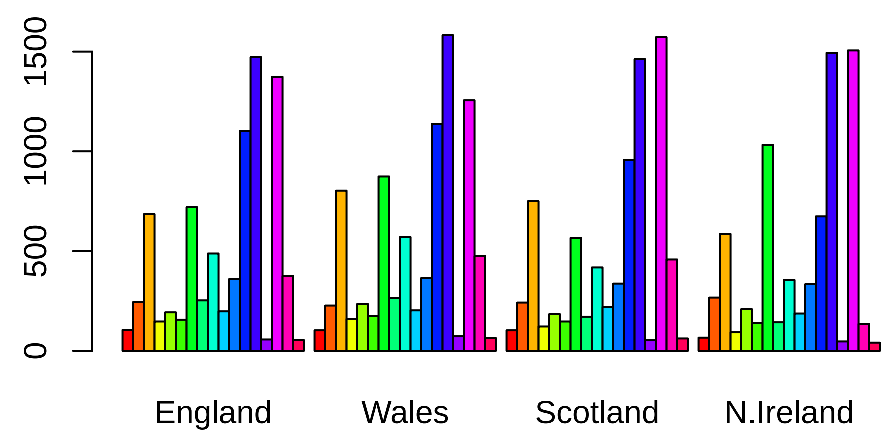
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars  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

```r
tail(x)
```

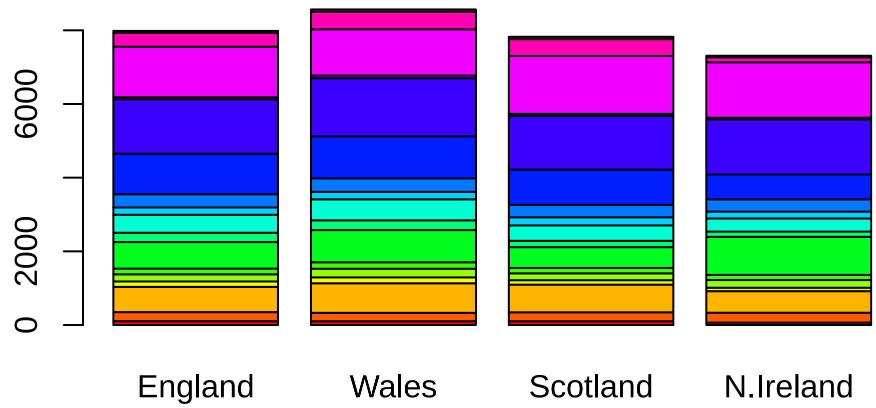
```
                  England Wales Scotland N.Ireland
Fresh_fruit          1102  1137      957       674
Cereals              1472  1582     1462      1494
Beverages              57    73       53        47
Soft_drinks          1374  1256     1572      1506
Alcoholic_drinks      375   475      458       135
Confectionery          54    64       62        41
```

Now some plots of our data (not very useful ones for visualization)

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




```r
#beside argument can change to stacked bars. default is False
barplot(as.matrix(x), beside = F, col = rainbow(nrow(x)))
```



##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 because that is what `prcomp()` help page was asking for

```
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
```

```
#get transposed version of x data frame with t() function
t(x)
```

```
          Cheese Carcass_meat  Other_meat  Fish Fats_and_oils  Sugars
England      105          245         685   147           193     156
Wales        103          227         803   160           235     175
Scotland     103          242         750   122           184     147
N.Ireland     66          267         586    93           209     139
          Fresh_potatoes  Fresh_Veg  Other_Veg  Processed_potatoes
England              720        253        488                 198
Wales                874        265        570                 203
Scotland             566        171        418                 220
N.Ireland           1033        143        355                 187
          Processed_Veg  Fresh_fruit  Cereals  Beverages Soft_drinks
England             360         1102     1472         57        1374
Wales               365         1137     1582         73        1256
Scotland            337          957     1462         53        1572
N.Ireland           334          674     1494         47        1506
          Alcoholic_drinks  Confectionery
England                375             54
Wales                  475             64
Scotland               458             62
N.Ireland              135             41
```

```
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
```

Lets see what is inside our result object `pca` that we just calculated

```r
attributes(pca)
```

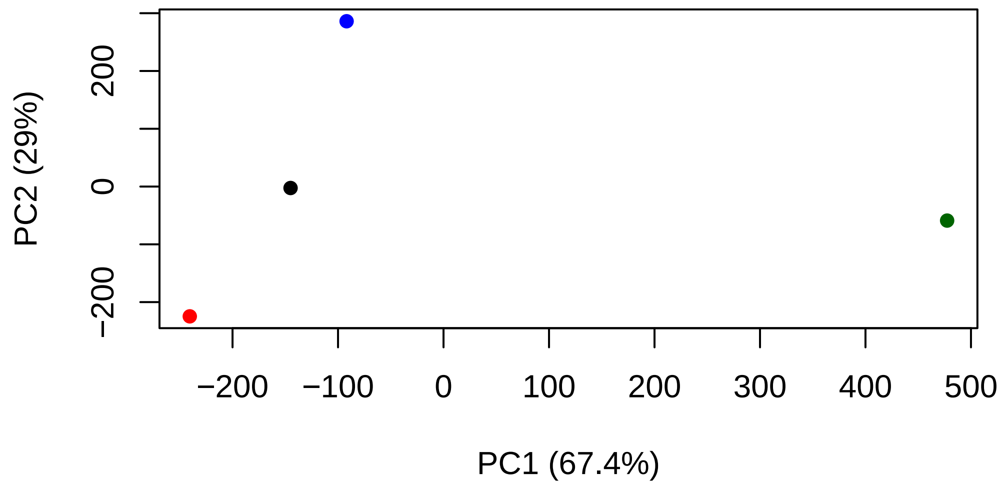
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```r
pca$x
```

```
                 PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -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, or"ordination plot" or "PC1 vs PC2 plot".

```r
plot(pca$x[,1], pca$x[,2], col=c("black", "red", "blue", "darkgreen"),
     pch=16,
     xlab = "PC1 (67.4%)", ylab = "PC2 (29%)"
     )
```

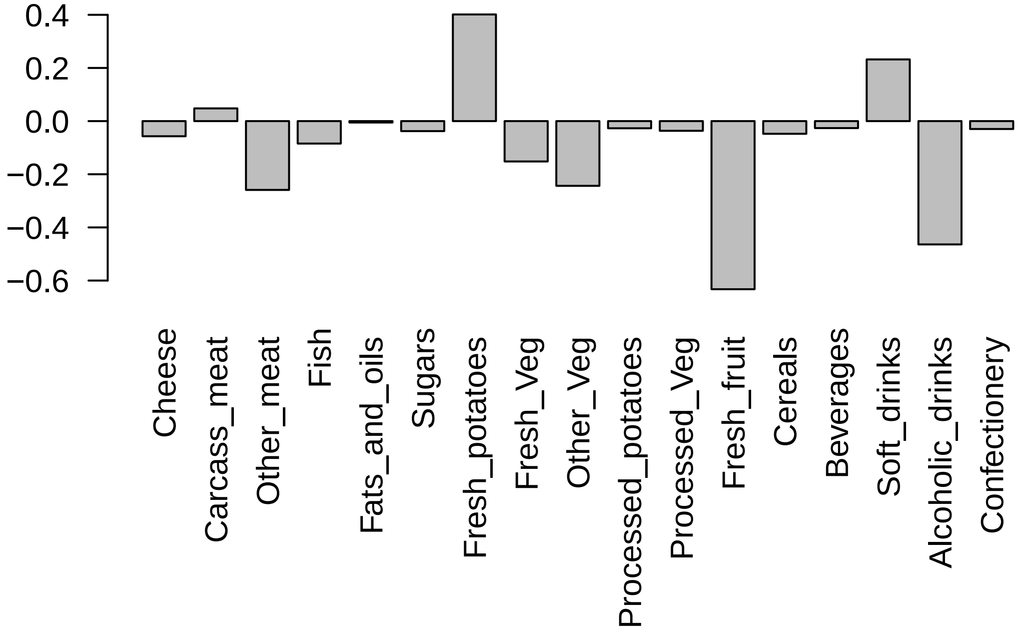



```
#now understand why had to transpose the dataset. we can see a point for each of the countrie
```

##Variable Loadings plot

Can give us insight into 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 )
```




```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
```