Class 07- Machine Learning

Renee Zuhars (PID: A17329856)

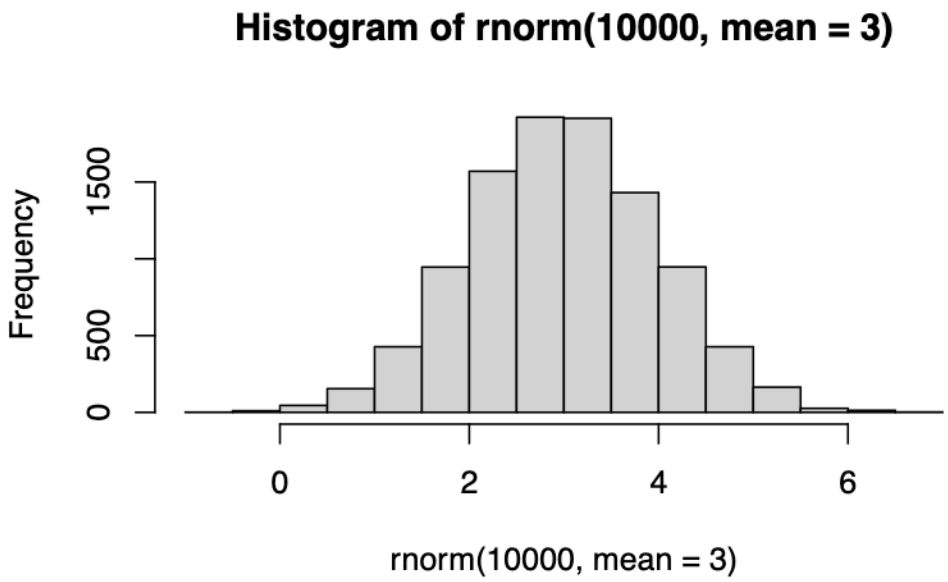
Table of contents

Today we will explore unsupervised machine learning methods starting with clustering and dimensionality reduction.

Clustering

To start let's make up some data to cluster where we know what the answer should be. The `rnorm()` function will help us here.

```
hist( rnorm(10000, mean=3) )
```



Return 30 numbers (a vector of 30 elements) centered on -3. Then do the same with positive 3.

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

x <- cbind(x=tmp, y=rev(tmp))

x
```

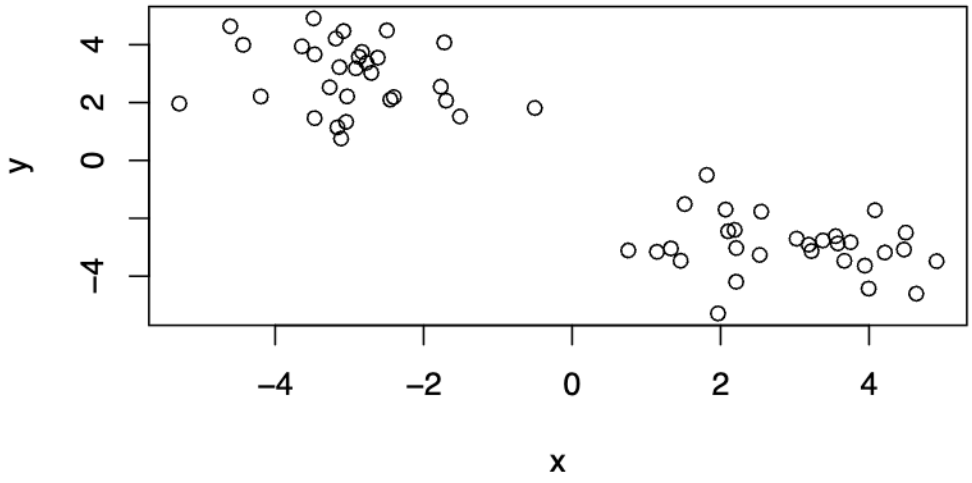
```
                x          y
 [1,] -1.6981285  2.0664976
 [2,] -3.0437124  1.3298806
 [3,] -1.5111845  1.5169687
 [4,] -2.4947773  4.4945683
 [5,] -3.0302841  2.2118881
 [6,] -2.7049707  3.0247849
 [7,] -3.1113827  0.7559237
 [8,] -2.9124431  3.1866345
 [9,] -2.7687072  3.3775433
[10,] -3.1842921  4.2118093
[11,] -2.4484406  2.1009738
[12,] -3.0801428  4.4701086
```

```
[13,] -3.4822823  4.9098622
[14,] -5.2915783  1.9649429
[15,] -2.8296930  3.7512184
[16,] -3.2662457  2.5272984
[17,] -1.7214677  4.0782580
[18,] -2.6183079  3.5496954
[19,] -4.6046246  4.6357346
[20,] -3.1604839  1.1428491
[21,] -2.4012385  2.1902371
[22,] -3.4694786  3.6667724
[23,] -1.7692965  2.5472309
[24,] -3.4689491  1.4617259
[25,] -4.4302804  3.9942563
[26,] -3.1334542  3.2242124
[27,] -2.8718356  3.5789886
[28,] -4.1944781  2.2093556
[29,] -0.5033586  1.8139809
[30,] -3.6395110  3.9428218
[31,]  3.9428218 -3.6395110
[32,]  1.8139809 -0.5033586
[33,]  2.2093556 -4.1944781
[34,]  3.5789886 -2.8718356
[35,]  3.2242124 -3.1334542
[36,]  3.9942563 -4.4302804
[37,]  1.4617259 -3.4689491
[38,]  2.5472309 -1.7692965
[39,]  3.6667724 -3.4694786
[40,]  2.1902371 -2.4012385
[41,]  1.1428491 -3.1604839
[42,]  4.6357346 -4.6046246
[43,]  3.5496954 -2.6183079
[44,]  4.0782580 -1.7214677
[45,]  2.5272984 -3.2662457
[46,]  3.7512184 -2.8296930
[47,]  1.9649429 -5.2915783
[48,]  4.9098622 -3.4822823
[49,]  4.4701086 -3.0801428
[50,]  2.1009738 -2.4484406
[51,]  4.2118093 -3.1842921
[52,]  3.3775433 -2.7687072
[53,]  3.1866345 -2.9124431
[54,]  0.7559237 -3.1113827
[55,]  3.0247849 -2.7049707
```

```
[56,]  2.2118881 -3.0302841
[57,]  4.4945683 -2.4947773
[58,]  1.5169687 -1.5111845
[59,]  1.3298806 -3.0437124
[60,]  2.0664976 -1.6981285
```

Make a plot of `x`.

```
plot(x)
```



K-means

The main function in "base" R for K-means clustering is called `kmeans()`.

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

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

Cluster means:
          x         y
```

```
1  2.931234 -2.961501
2 -2.961501  2.931234

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 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] 66.34473 66.34473
 (between_SS / total_SS =  88.7 %)

Available components:

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

The `kmeans()` function return "list" with 9 components.

```
attributes(km)
```

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

$class
[1] "kmeans"
```

Q. How many points are in each cluster?

```
km$size
```

```
[1] 30 30
```

Q. Cluster assignment/membership vector?

```
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
```

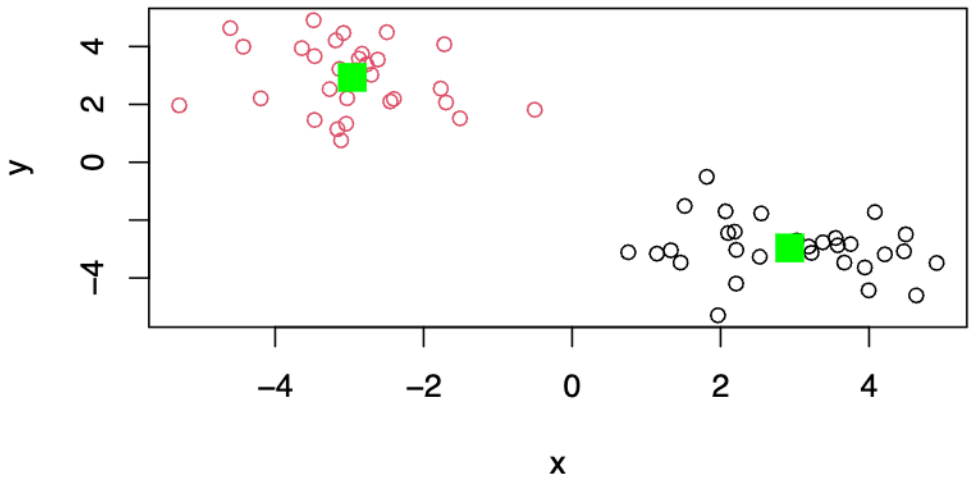
Q. Cluster centers?

```
km$centers
```

```
          x         y
1  2.931234 -2.961501
2 -2.961501  2.931234
```

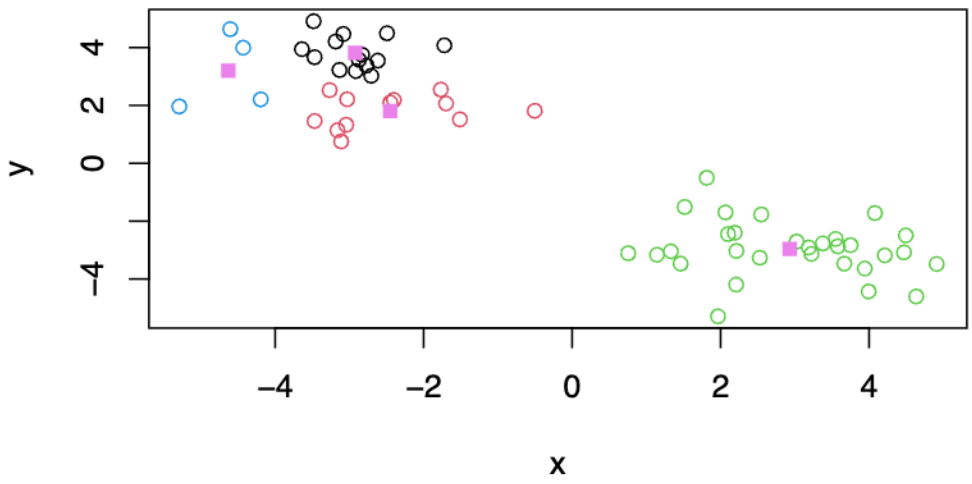
Q. Make a plot of our **kmeans()** results showing cluster assignment using different colors for each cluster/group of points and cluster centers.

```
plot(x, col=km$cluster) #color by cluster
points(km$centers, col="green", pch=15, cex=2)
```



Q. Run **kmeans** again on **x** and create 4 groups/clusters. Plot the same result figure as above.

```
km4 <- kmeans(x, centers=4)
plot(x, col=km4$cluster)
points(km4$centers, col="violet", pch=15)
```



key point: K means clustering is super popular but can be misused. One big limitation is that it can impose a clustering pattern on your data even if clear natural grouping doesn't exist- i.e. it does what you tell it to do in terms of `centers`.

Hierarchical Clustering

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

You can't just pass our dataset as is into `hclust()` you must give "distance matrix" as input. We can get this from the `dist()` function in R.

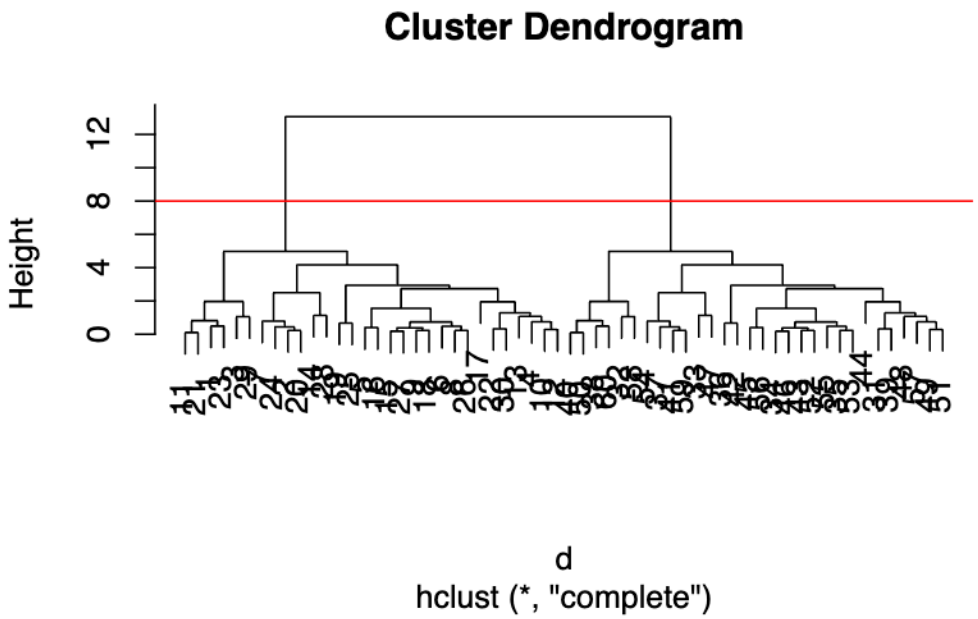
```
d <- dist(x)
hc <- hclust(d)
hc
```

```
Call:
hclust(d = d)

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

The results of `hclust()` don't have a useful `print()` method but do have a special `plot()` method.

```
plot(hc)
abline(h=8, col='red')
```



To get our main cluster assignment (membership vector) we need to "cut" the tree at the big goalposts

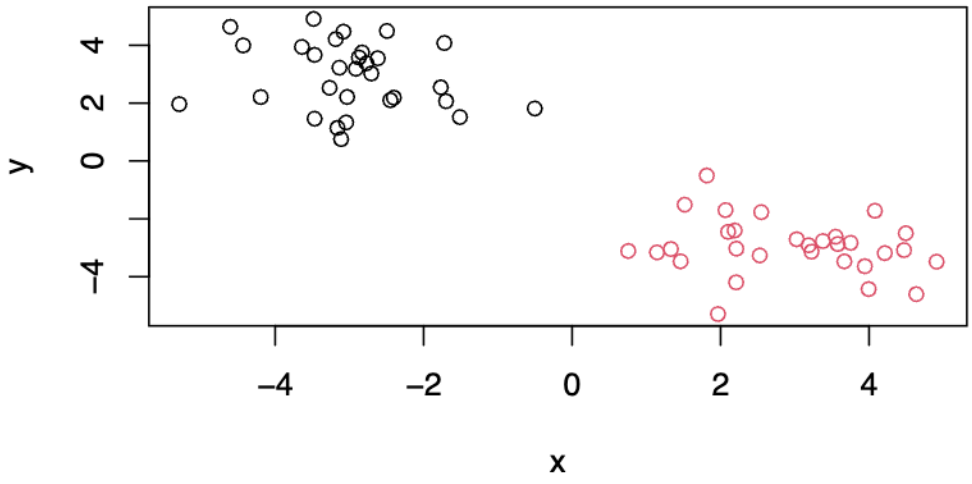
```
grps <- (cutree(hc, h=8))
grps
```

```
 [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 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
```

```
table(grps)
```

```
grps
 1  2
30 30
```

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



Hierarchical Clustering is distinct in that the dendrogram (tree figure) can reveal the potential grouping in your data

Principal Component Analysis (PCA)

PCA is a common and highly useful dimensionality reduction technique used in many fields- particularly bioinformatics.

Here we will analyze some data from the UK on food consumption.

Data import

```
url <- 'https://tinyurl.com/UK-foods'
x <- read.csv(url)
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
```

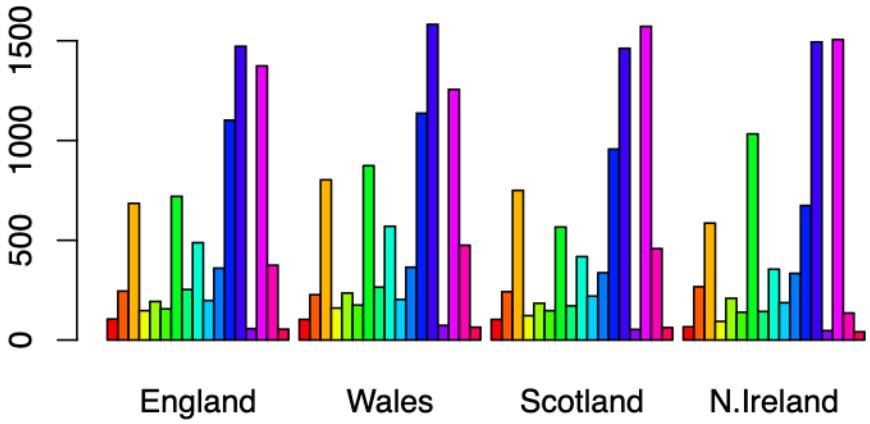
```
rownames(x) <- x[,1]
x <- x[,-1] #destructive, removes each time
head(x)
```

```
              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
```

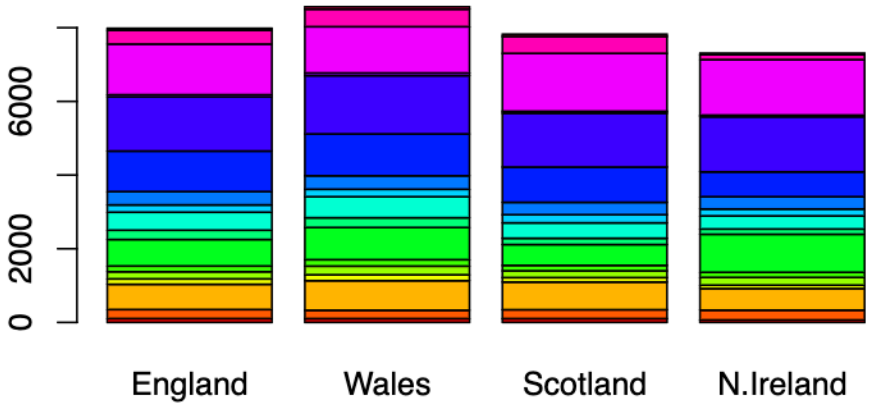
```
x <- read.csv(url, row.names = 1)
head(x)
```

```
              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)))
```

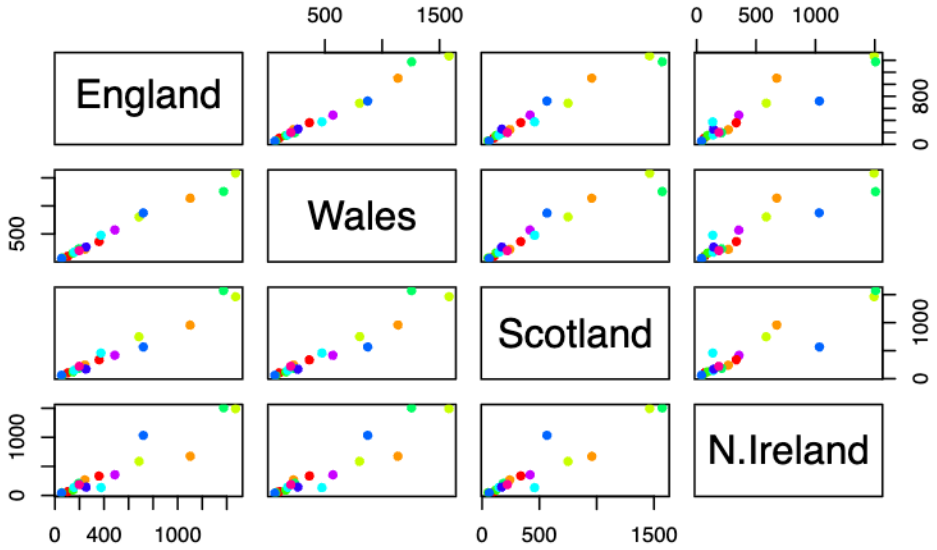



```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



One conventional plot that can be useful is called a "pairs" plot (plot of all pairwise combinations of countries next to each other)

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



This pairwise plot produces 12 graphs directly comparing each country to each other country.

PCA to the rescue

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

```
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
```

PC1 vs PC2 results in 96.5% coverage (see cumulative proportion)

The `prcomp()` function returns a list object of our results with

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

The two main "results" in here are `pca$x` and `pca$rotation`. The first of these (`pca$x`) contains the scores of the data on the new PC axis- we use these to make our "PCA plot".

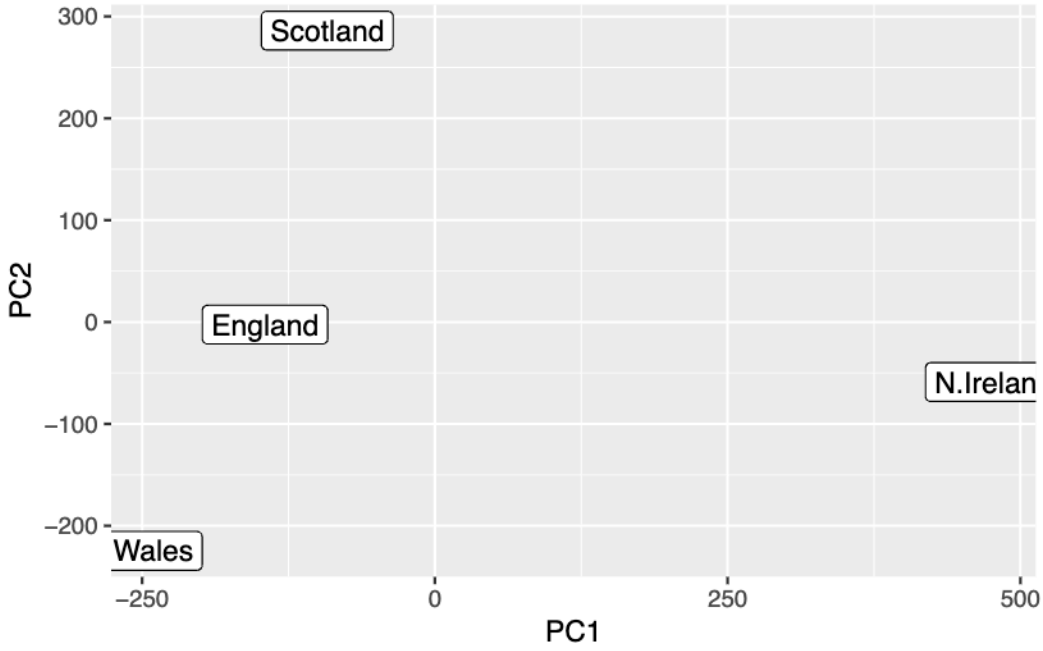
```
pca$x
```

```
                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
```

Q. Make a plot of pca$x with PC1 vs PC2

```
library(ggplot2)

ggplot(pca$x) +
  aes(PC1, PC2, label=rownames(pca$x)) +
  geom_point() +
  geom_label()
```



This plot shows the distribution spread of the four countries.

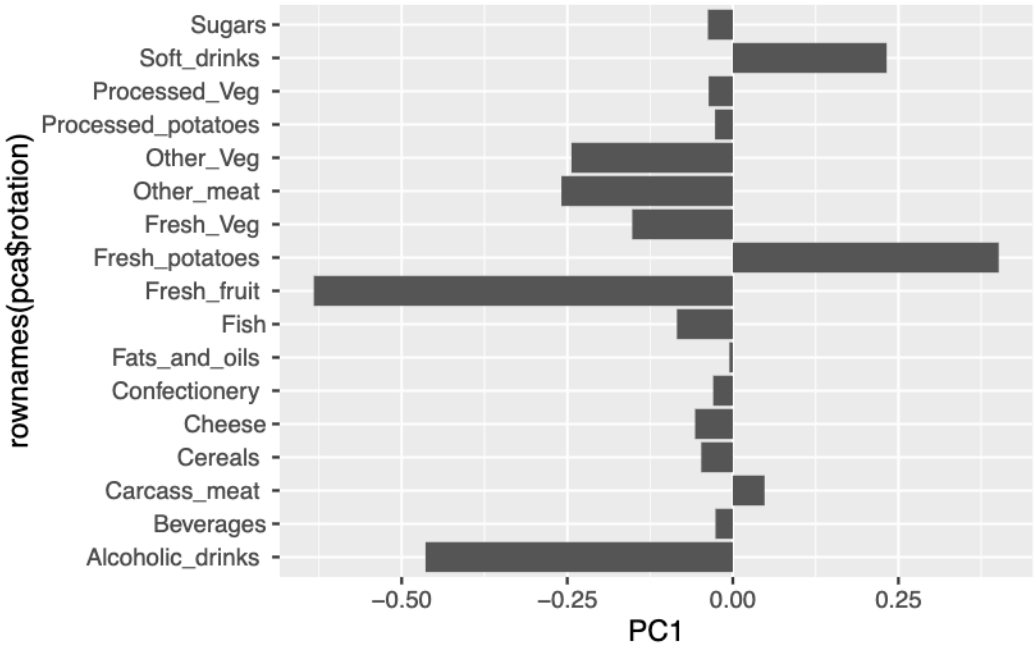
The second major result is contained in the `pca$rotation` object or component. Let's plot this to see what PCA is picking up...

```
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
```

```
ggplot(pca$rotation) +
  aes(PC1, rownames(pca$rotation)) +
  geom_col()
```



This plot compares the PC1 to our original dataset.