class07

Cienna Santos (PID: A17581026)
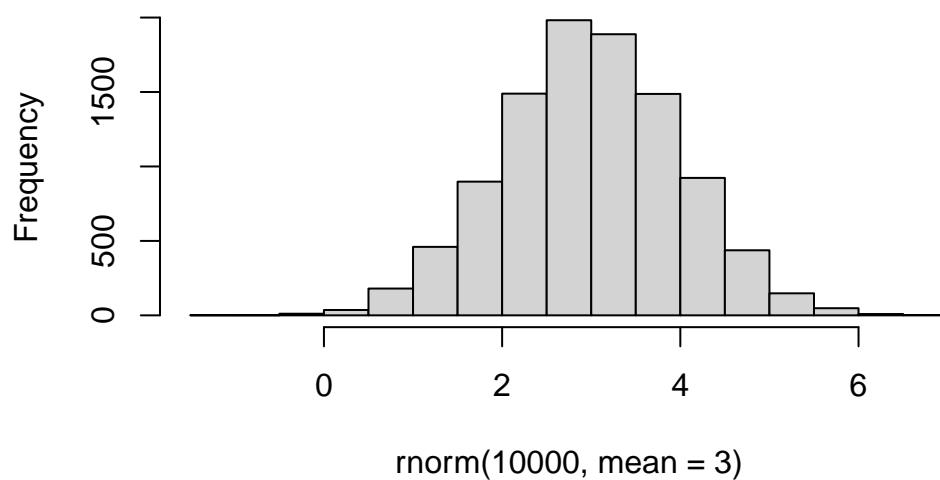
Table of contents

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 up here.

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

Histogram of rnorm(10000, mean = 3)

Return 30 numbers centered on -3

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

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

x
```

```
              x          y
 [1,] -3.5120142  5.0657285
 [2,] -4.5375474  2.5155505
 [3,] -3.9682205  3.7148522
 [4,] -1.5101805  2.2980059
 [5,] -2.0278096  3.4094984
 [6,] -2.3546095  4.7700220
 [7,] -3.4126945  4.1156150
 [8,] -3.7193845  2.0042758
 [9,] -1.4809052  3.0239442
[10,] -2.0489329  4.2928168
[11,] -3.5435075  2.9703155
[12,] -3.7257022  3.4108425
[13,] -2.2799337  5.4017213
```

```
[14,] -3.0632260  2.6753363
[15,] -3.9727662  3.4140783
[16,] -1.4918494  3.2847843
[17,] -2.0727262  3.1325302
[18,] -3.8975415  1.2597132
[19,] -3.1642025  2.0866263
[20,] -2.7874188  3.6625157
[21,] -2.2999997  3.9035741
[22,] -2.3968085  3.2026995
[23,] -1.4274769  1.9335800
[24,] -3.5842320  0.4733858
[25,] -1.6399405  2.3920911
[26,] -2.5586226  1.6294757
[27,] -2.5763201  3.2238168
[28,] -1.9499127  2.6972431
[29,] -2.8206205  3.7831818
[30,] -2.4000128  3.6187393
[31,]  3.6187393 -2.4000128
[32,]  3.7831818 -2.8206205
[33,]  2.6972431 -1.9499127
[34,]  3.2238168 -2.5763201
[35,]  1.6294757 -2.5586226
[36,]  2.3920911 -1.6399405
[37,]  0.4733858 -3.5842320
[38,]  1.9335800 -1.4274769
[39,]  3.2026995 -2.3968085
[40,]  3.9035741 -2.2999997
[41,]  3.6625157 -2.7874188
[42,]  2.0866263 -3.1642025
[43,]  1.2597132 -3.8975415
[44,]  3.1325302 -2.0727262
[45,]  3.2847843 -1.4918494
[46,]  3.4140783 -3.9727662
[47,]  2.6753363 -3.0632260
[48,]  5.4017213 -2.2799337
[49,]  3.4108425 -3.7257022
[50,]  2.9703155 -3.5435075
[51,]  4.2928168 -2.0489329
[52,]  3.0239442 -1.4809052
[53,]  2.0042758 -3.7193845
[54,]  4.1156150 -3.4126945
[55,]  4.7700220 -2.3546095
[56,]  3.4094984 -2.0278096
```

```
[57,]   2.2980059 -1.5101805
[58,]   3.7148522 -3.9682205
[59,]   2.5155505 -4.5375474
[60,]   5.0657285 -3.5120142
```

Make a plot of x

```
plot(x)
```
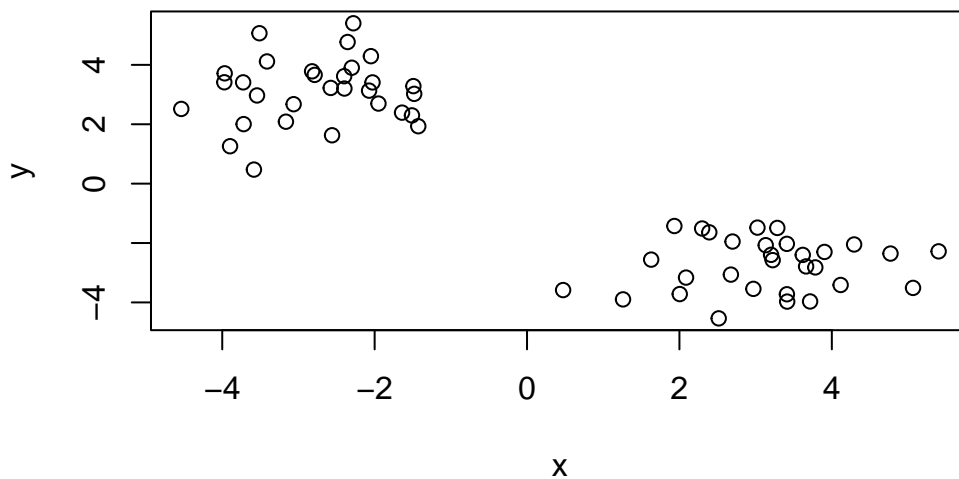

K-means

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 -2.740837  3.112219
```

```
2  3.112219 -2.740837

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 2 2

Within cluster sum of squares by cluster:
[1] 57.43607 57.43607
 (between_SS / total_SS =  89.9 %)

Available components:

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

The `kmeans()` function returns a "list with 9 components. You can see the named components of any list with the `attributes()` function.

```
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] 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
```
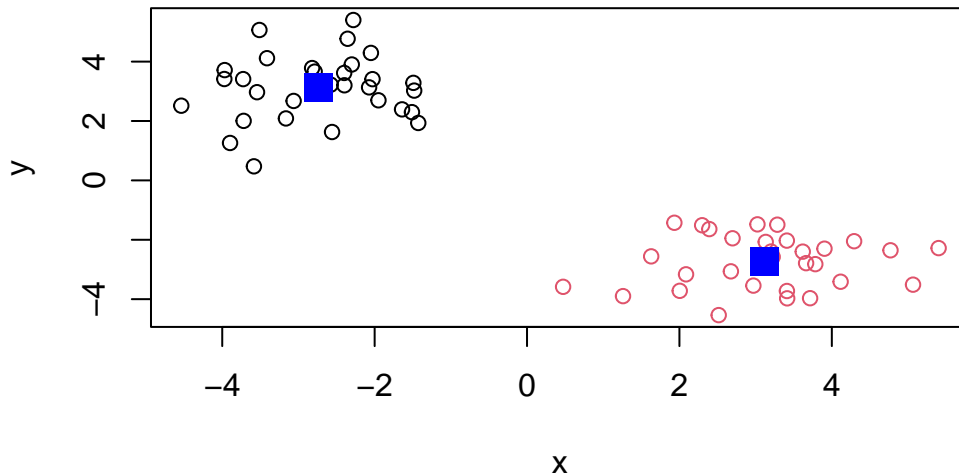
Q. Cluster centers?

```
km$centers
```

```
          x          y
1 -2.740837  3.112219
2  3.112219 -2.740837
```

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

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


Q. Run `kmeans()` again on `x` and this cluster into 4 groups/clusters and plot the same result figure as above.

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

6
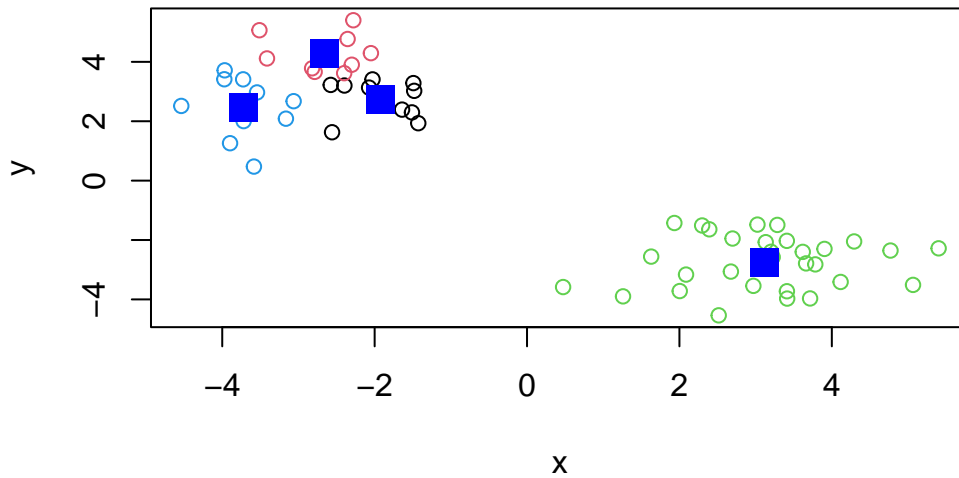
key-point: K-means clustering is super populat but can be misused. One big limitation is that it can impose a clustering pattern on your data even if clear natural grouping doesn't exist – i.e. it does what you tell it to do in terms of `centers`.

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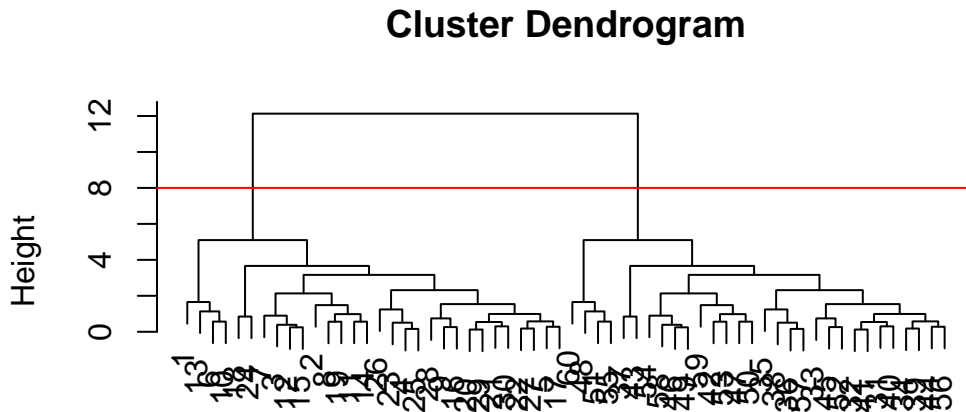
```
d <- dist(x)
hc <- hclust(d)
hc
```

```
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")
```

Cluster Dendrogram

d
hclust (*, "complete")

To get our main cluster assignment (membership vector), we need to "cut" the tree at the big goal posts...
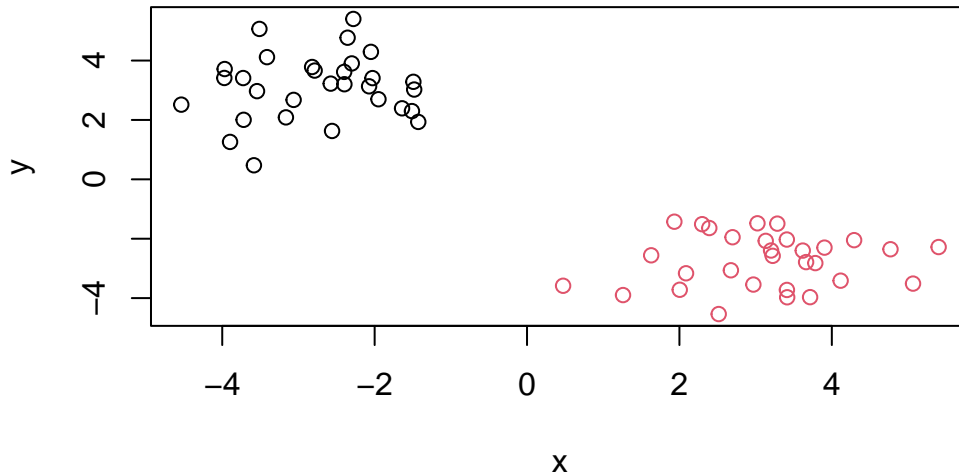
```
grps <- cutree(hc, h=8)
grps
```

```
 [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
```

```
table(grps)
```

```
grps
 1  2
30 30
```

8

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


Hierarchical clustering is distinct in that the dendrogram (tree figure) can reveal the potential grouping in your data (unlike K-means)

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
```

```r
rownames(x) <- (x[,1])
x <- x[,-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
```
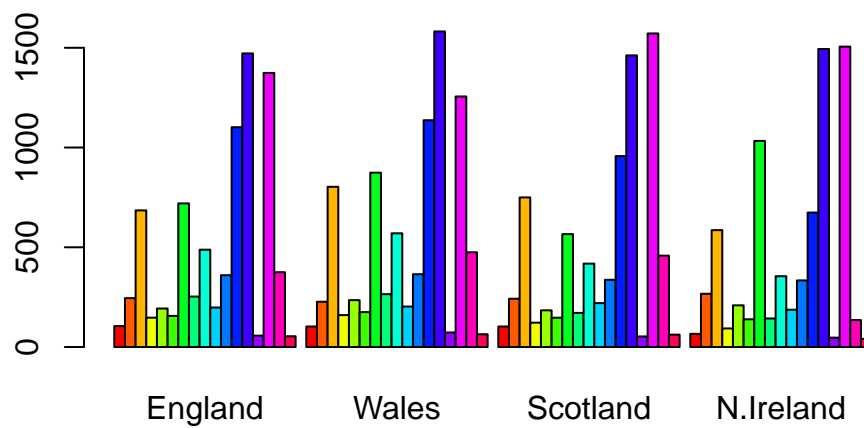
Reassigning x like this keeps erasing the columns as you re-run head() over and over. Instead...

```r
x <- (read.csv(url, row.names = 1))
head(x)
```
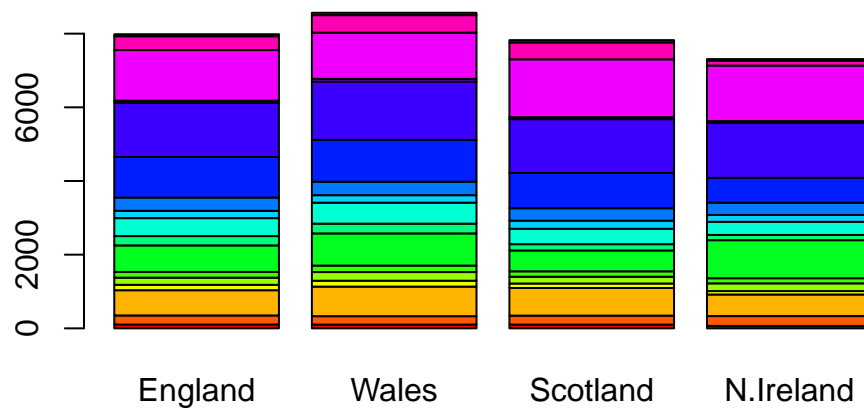
```
              England Wales Scotland N.Ireland
Cheese            105   103      103        66
Carcass_meat      245   227      242       267
Other_meat        685   803      750       586
Fish              147   160      122        93
Fats_and_oils     193   235      184       209
Sugars            156   175      147       139
```

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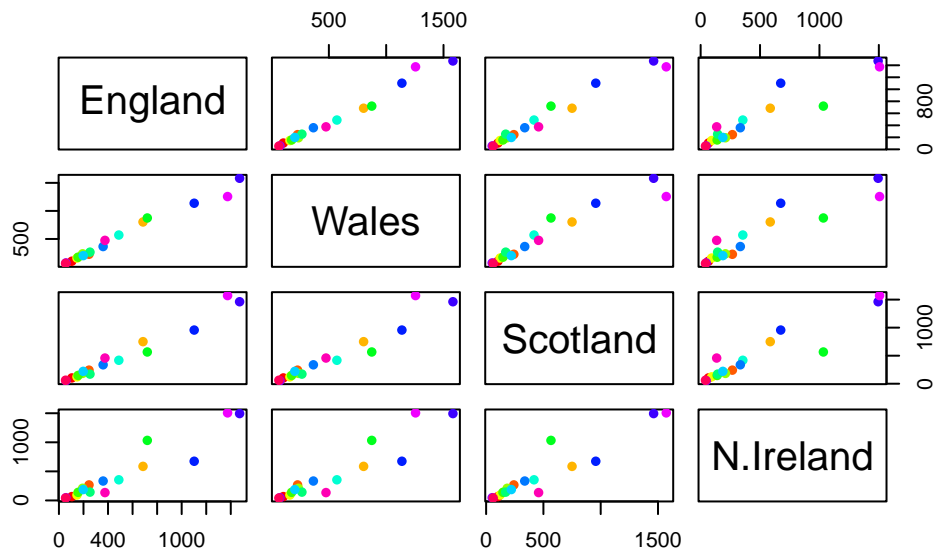
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```


One conventional plot that can be useful is called a "paris" plot.

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


Follow which block borders each graph. That shows which two places are being compared and on what axis.

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 3.176e-14
Proportion of Variance  0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion   0.6744   0.9650  1.00000 1.000e+00
```

Proportion of Variance at PC1 means you could capture all 17 points in one axis and you'd have 67% of the data there.

Cumulative Proportion on PC2 combines PC1 and PC2 Proportion of Variance. It means you can plot those two together(? on different axes??). and you'd capture 96% of the data.

The `prcomp()` function returns a list object of our results with five attributes/components

```
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 -4.894696e-14
Wales      -240.52915 -224.646925 -56.475555  5.700024e-13
Scotland    -91.86934  286.081786 -44.415495 -7.460785e-13
N.Ireland   477.39164  -58.901862  -4.877895  2.321303e-13
```
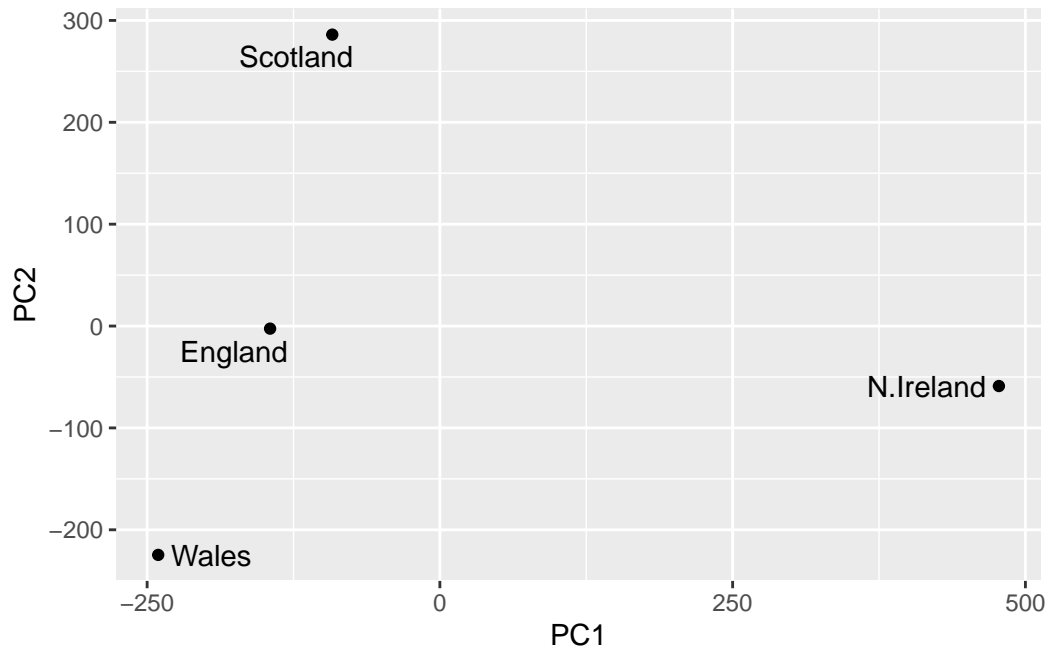
```
library(ggplot2)
library(ggrepel)

# Make a plot of pca$x with PC1 vs PC2
ggplot(pca$x) +
  aes(PC1, PC2, label=rownames(pca$x)) +
  geom_point() +
  geom_text_repel()
```

This graph factors in all 17 point of each cluster and combines them into 4 points. By plotting PC1 vs PC2 we get points on a graph that show how similar/dissimilar the data of each cluster was from one another.

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

```
ggplot(pca$rotation) +
  aes(PC1, rownames(pca$rotation)) +
  geom_col()
```
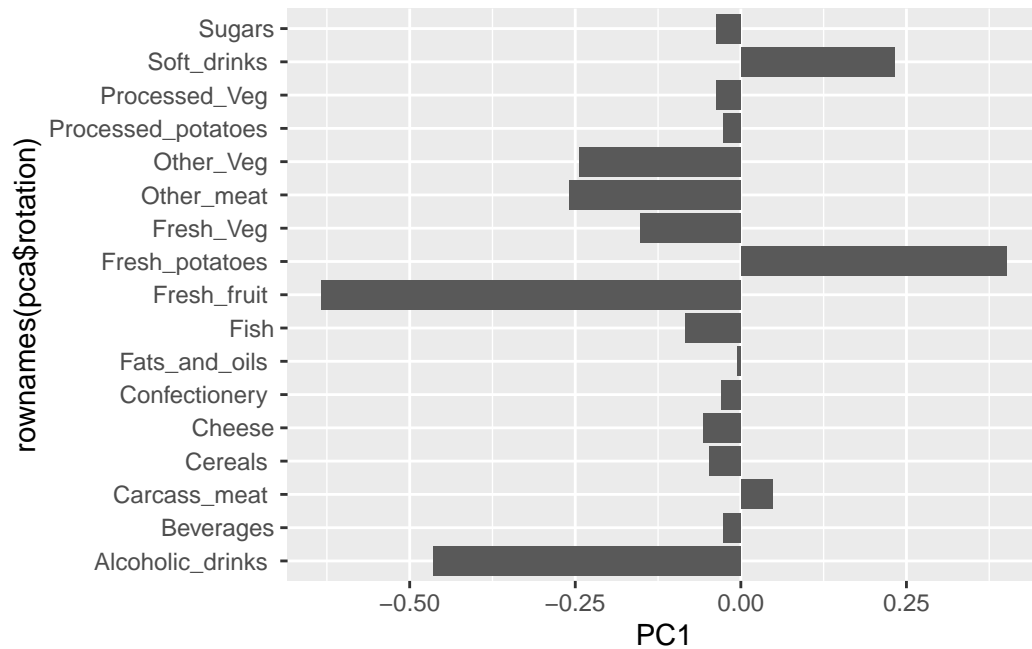
In this graph, any bar going in the positive direction shows that N. Ireland eats more of those, and bars going in the negative direction show that England, Scotland, and Wales eat more of that. The negative and positive skew can be seen in the x-axis of the previous graph.