Class 07: Machine Learning 1

Mitchell Sullivan (PID: A18595276)

Table of contents

Background 1
K-means Clustering 2
Hierarchical Clustering 7
PCA of UK food data 8
Spotting major differences and trends 10
    Pairs plots and heatmaps 12
PCA to the rescue 13

Background

Today we will begin our exploration of some important machine learning methods, namely **clustering** and **dimensionality reduction**.

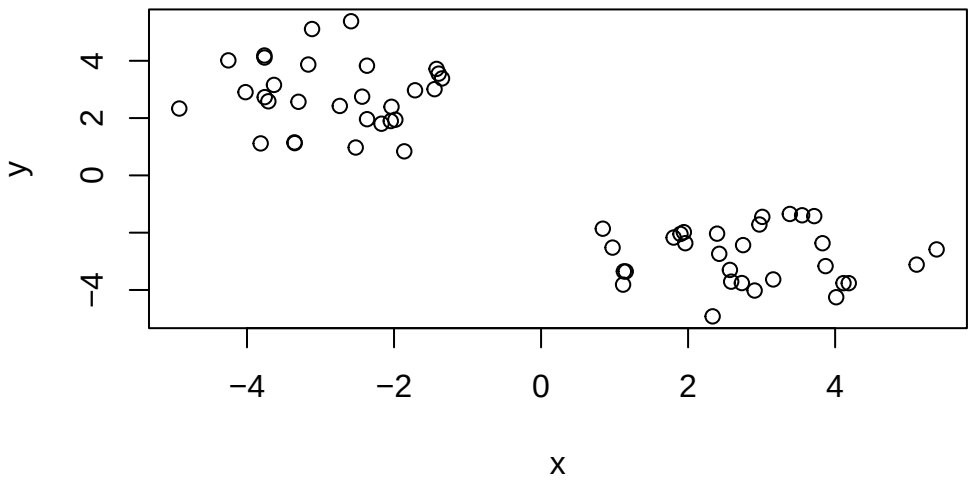
Let's make up some input data for clustering where we know what the natural clusters are.

The function `rnorm()` can be useful here.

```
rnorm3pos <- rnorm(30, mean =  3)
rnorm3neg <- rnorm(30, mean = -3)
```

```
tmp <- c(rnorm3pos, rnorm3neg)
x <- cbind(x = tmp, y = rev(tmp))
```

```
plot(x)
```



K-means Clustering

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

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

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

Cluster means:
          x         y
1 -2.811689  2.792943
2  2.792943 -2.811689

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] 67.92911 67.92911
 (between_SS / total_SS =  87.4 %)
```

```
Available components:

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

Q. What component of the results object details the cluster sizes?

```
km$size
```

```
[1] 30 30
```

Q. What component of the results object details the cluster centers?

```
km$centers
```

```
          x         y
1 -2.811689  2.792943
2  2.792943 -2.811689
```

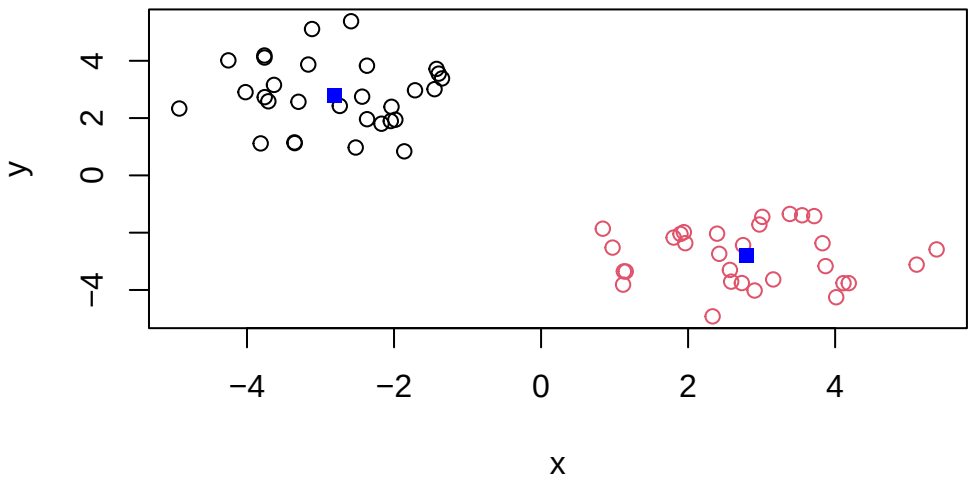
Q. What component of the results object details the cluster membership vector (i.e. our main result of which points lie in which cluster)?

```
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. Plot our sclustering results with points colored by cluster and also add the cluster centers as new points colored blue

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



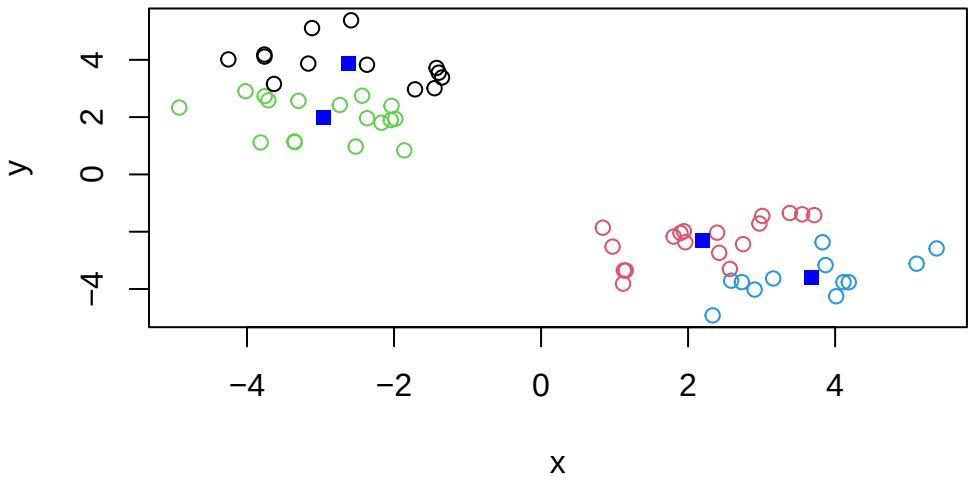
Q. Run `kmeans()` again and this time produce 4 clusters (call your result object `k4`) and make a results figure like above?

```
k4 <- kmeans(x, centers = 4)
k4
```

```
K-means clustering with 4 clusters of sizes 13, 18, 17, 12

Cluster means:
          x         y
1 -2.613059  3.868344
2  2.198523 -2.294220
3 -2.963583  1.970578
4  3.684573 -3.587893

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

Within cluster sum of squares by cluster:
[1] 20.08473 24.23929 20.40808 15.73988
 (between_SS / total_SS =  92.5 %)
```

```
Available components:

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

```
plot(x, col = k4$cluster)
points(k4$centers, col = "blue", pch = 15)
```



The metric total within sums of squares gives us more information on the "fit" of the clustering

```
km$tot.withinss
```

```
[1] 135.8582
```

```
k4$tot.withinss
```

```
[1] 80.47198
```

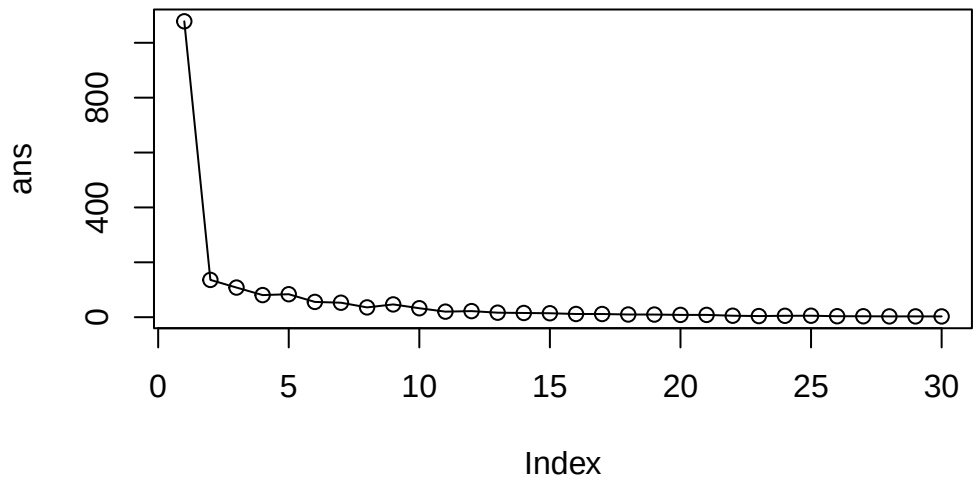
Key-point Because K-means will impose the number of groups you specify, it can artificially form clusters that aren't there.

Q. Let's try different numbers of K (centers) from 1 to 30 and see what the best result is

```
ans <- NULL
for (i in 1:30) {
  ans <- c(ans, kmeans(x, centers = i)$tot.withinss)
}

ans
```

```
 [1] 1078.215265  135.858221  107.908277   80.471975   83.589202   55.639257
 [7]   52.712983   35.665568   46.539064   32.624309   20.145512   22.143162
[13]   16.384573   15.277305   14.298740   11.647889   11.524407    9.814527
[19]    9.610789    8.261072    8.303838    5.335909    4.038065    4.998548
[25]    5.272589    3.454351    3.353336    2.889346    2.891236    2.737496
```

```
plot(ans, typ = "o")
```

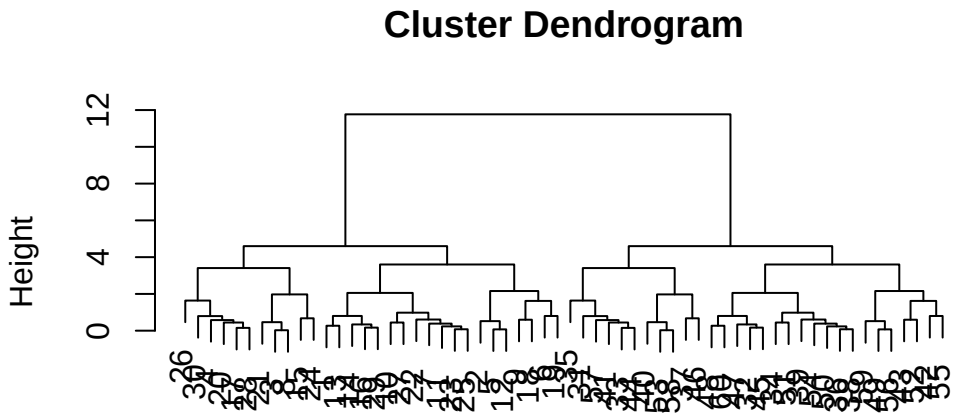


Using this method, we can find where the biggest step down in total within sums of squares and pick the best number of clusters.

Hierarchical Clustering

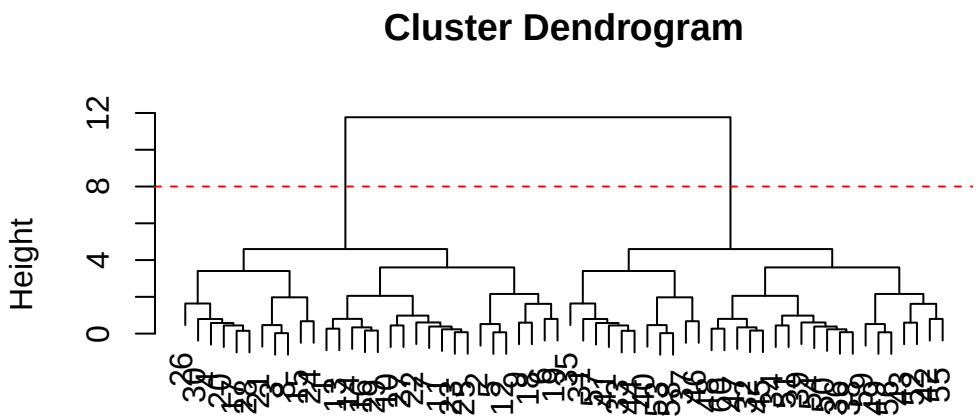
The main function for Hierarchical Clustering is called `hclust()`. Unlike `kmeans()` (which does all the work for you), you can't just pass `hclust()` our raw input data. It needs a "distance matrix" like the one returned from the `dist()` function.

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



To extract our cluster membership vector from an `hclust()` result object, we have to "cut" our tree at a given height to yield separate "groups" or "branches".

```
plot(hc)
abline(h = 8, col = "red", lty = 2)
```



To do this, we use the `cutree()` function on our `hclust()` object:

```
grps <- cutree(hc, h = 6)
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
```

```
table(grps, km$cluster)
```

```
grps  1  2
   1  0 30
   2 30  0
```

PCA of UK food data

Import the dataset of food consumption in the UK

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
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
7      Fresh_potatoes     720   874      566      1033
8           Fresh_Veg     253   265      171       143
9           Other_Veg     488   570      418       355
10 Processed_potatoes     198   203      220       187
11      Processed_Veg     360   365      337       334
12        Fresh_fruit    1102  1137      957       674
13            Cereals    1472  1582     1462      1494
14          Beverages      57    73       53        47
15        Soft_drinks    1374  1256     1572      1506
16   Alcoholic_drinks     375   475      458       135
17      Confectionery      54    64       62        41
```

Q1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

There are 17 rows and 5 columns. Using the `dim()` function will tell us this.

```
dim(x)
```

```
[1] 17  5
```

One solution to set the row names is to do it by hand...

```
rownames(x) <- x[,1]
head(x)
```

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

```
dim(x)
```

```
[1] 17  5
```

A better way to do this is to set the row names to the first column with `read.csv()`.

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

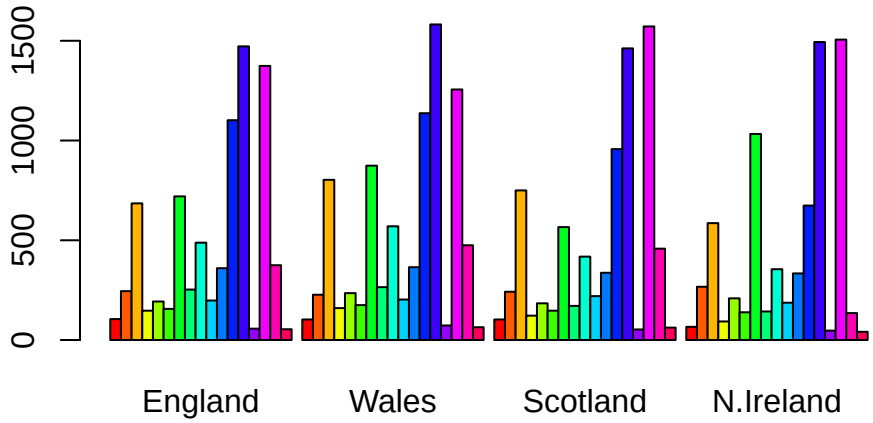
	England	Wales	Scotland	N.Ireland
Cheese	105	103	103	66
Carcass_meat	245	227	242	267
Other_meat	685	803	750	586
Fish	147	160	122	93
Fats_and_oils	193	235	184	209
Sugars	156	175	147	139
Fresh_potatoes	720	874	566	1033
Fresh_Veg	253	265	171	143
Other_Veg	488	570	418	355
Processed_potatoes	198	203	220	187
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

> 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?

I prefer using the `read.csv()` argument because it won't delete a column when run more than once.

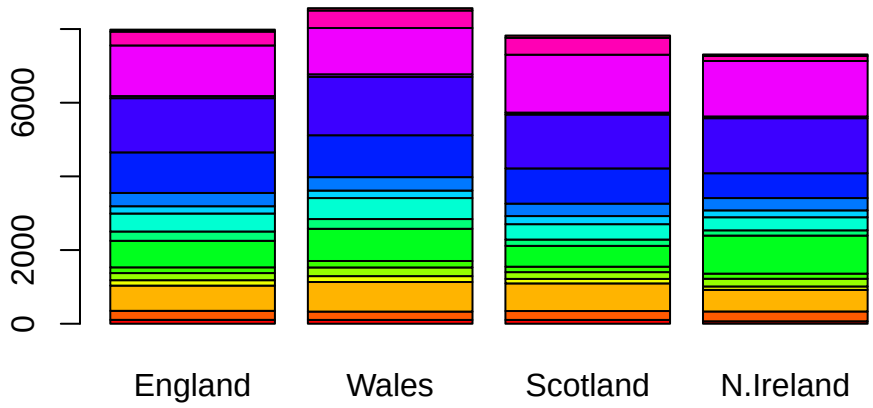
Spotting major differences and trends

```
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?

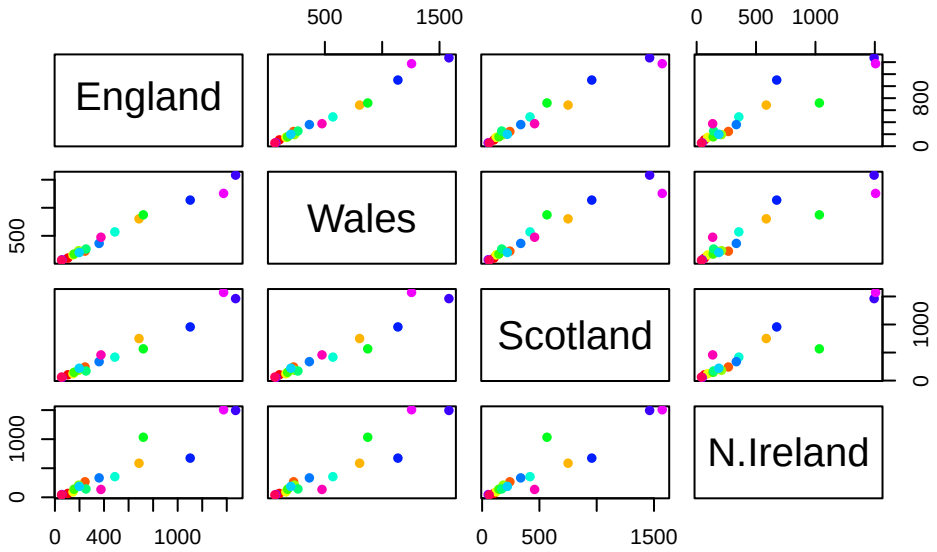
```
barplot(as.matrix(x), col=rainbow(nrow(x)))
```



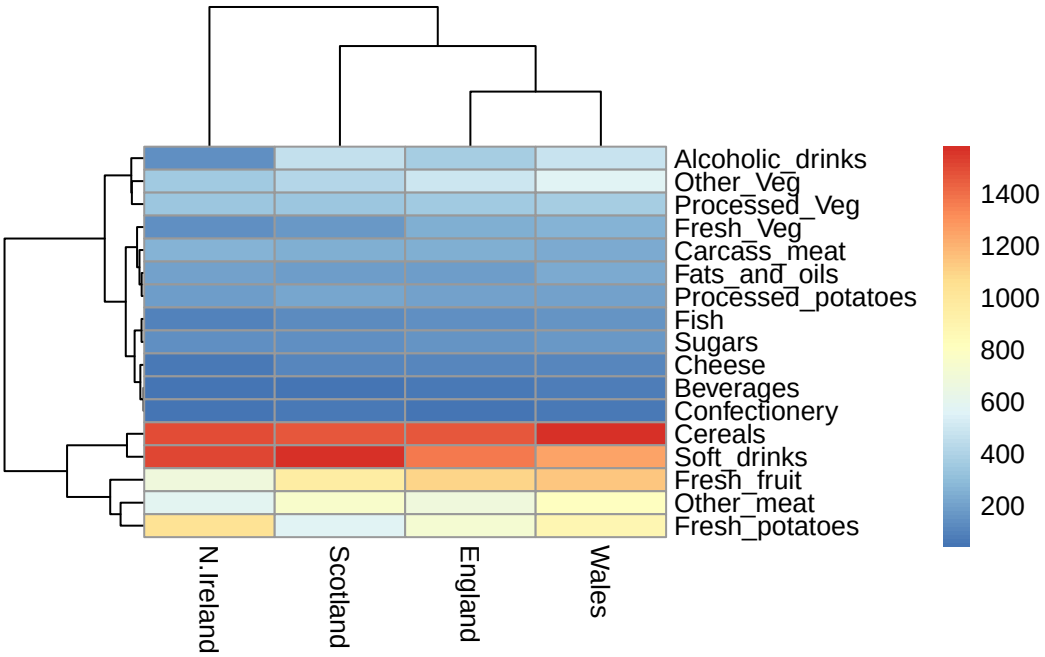
Removing `beside = T` stacks the bars

Pairs plots and heatmaps

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




```
library(pheatmap)

pheatmap( as.matrix(x) )
```



PCA to the rescue

The main PCA function in "base R" is called `prcomp()`. This function wants the transpose of our food data as input (i.e. the foods as columns and the countries as rows).

```
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
```

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

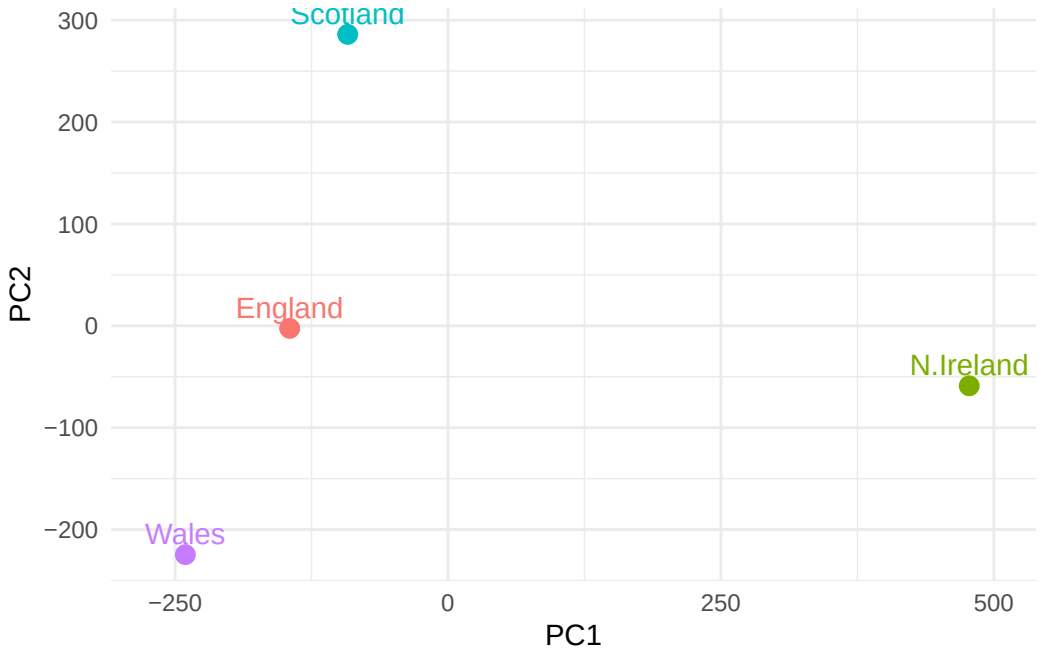
To make one of our main PCA result figures, we turn to `pca$x` (the scores along our PCs). This is called "PC plot", "score plot", or "ordination plot" ...

```
library(ggplot2)
```

```
Warning: package 'ggplot2' was built under R version 4.4.3
```

```
my_cols <- c("orange", "red", "blue", "darkgreen")
```

```
ggplot(pca$x) +
  aes(x = PC1, y = PC2, label = rownames(pca$x), colour = rownames(pca$x)) +
  geom_point(size = 3) +
  geom_text(vjust = -0.5) +
  xlim(-270, 500) +
  xlab("PC1") +
  ylab("PC2") +
  theme_minimal() +
  theme(legend.position = "none")
```



The second major result figure is called a "loadings plot", "variable contributions plot", or "weight plot".

```r
ggplot(pca$rotation) +
  aes(x = PC1,
      y = reorder(rownames(pca$rotation), PC1)) +
  geom_col(fill = "grey30") +
  xlab("PC1 Loading Score") +
  ylab("") +
  theme_minimal() +
  theme(axis.text.y = element_text(size = 8.5))
```