Class 7: Machine Learning 1

Emily Chen (PID: A16925878)

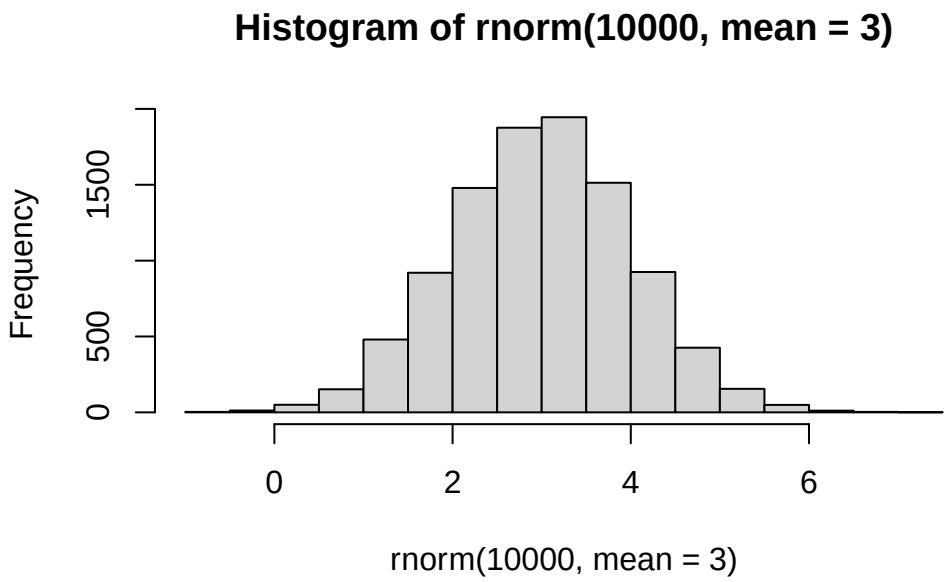
Table of contents

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

Clustering

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

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



Return 30 numbers centered on -3 So we are going to do rnorm() with 30 inside, and then for the mean we put -3 so that it will be centered at -3.

```
rnorm(30, mean=-3)
```

```
 [1] -2.927292 -3.205916 -1.995257 -2.492672 -3.668203 -3.997832 -5.065514
 [8] -2.245790 -1.685109 -2.851062 -2.484871 -2.606973 -2.599097 -1.430616
[15] -3.786621 -2.777023 -3.411738 -3.934057 -2.692508 -4.008403 -2.135482
[22] -4.117646 -3.596326 -2.145543 -4.158680 -4.960932 -3.236552 -2.916005
[29] -5.004536 -2.604609
```

```
rnorm(30, mean=+3)
```

```
 [1] 1.352944 1.642567 1.764674 1.134139 3.735790 5.082753 1.514705 2.275155
 [9] 4.232656 2.046336 3.609243 3.371938 3.546516 2.365242 2.757558 3.308907
[17] 1.799068 2.826527 4.550830 4.012148 5.103158 2.344181 4.940274 2.234244
[25] 2.184852 3.036562 4.350580 1.608979 4.093199 4.433550
```

To combine the two, we will create a vector using the c() function. Below we have 60 points in total

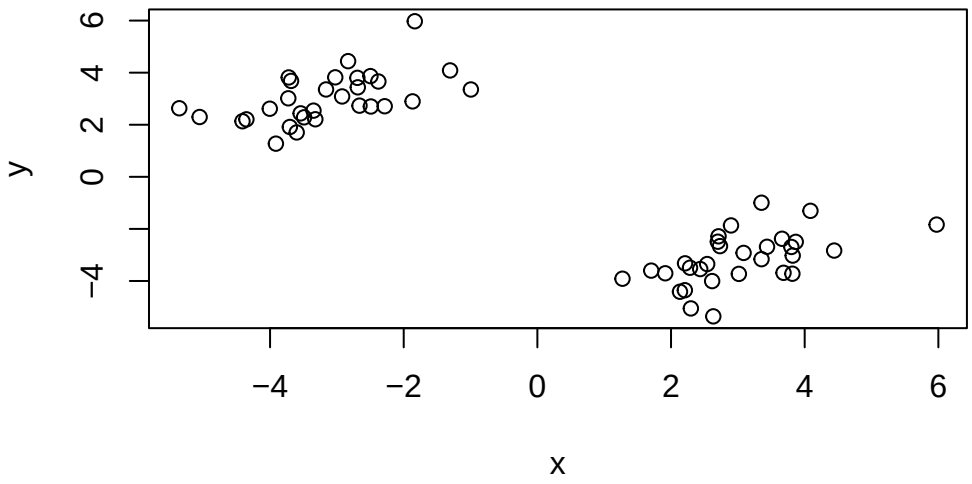
```r
tmp<-c(rnorm(30, mean=-3),
  rnorm(30, mean=+3) )

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

x
```

```
               x          y
 [1,] -3.6857495  3.6835618
 [2,] -2.4916507  2.7003328
 [3,] -4.0037068  2.6153352
 [4,] -3.1624108  3.3544061
 [5,] -1.8674317  2.8971687
 [6,] -2.2846241  2.7108179
 [7,] -3.4910438  2.2839814
 [8,] -2.3791313  3.6609349
 [9,] -3.7037302  1.9133511
[10,] -4.4126433  2.1339963
[11,] -3.3500325  2.5411868
[12,] -2.4976632  3.8663384
[13,] -1.8342443  5.9742427
[14,] -1.3062023  4.0858435
[15,] -3.0241445  3.8208766
[16,] -3.3222148  2.2097518
[17,] -3.7206556  3.8158973
[18,] -0.9949036  3.3535170
[19,] -5.3581478  2.6322209
[20,] -2.6871726  3.4362268
[21,] -3.5420838  2.4369105
[22,] -4.3538977  2.2067787
[23,] -3.6011636  1.7037760
[24,] -3.7270742  3.0149824
[25,] -2.6613964  2.7326862
[26,] -5.0537676  2.2966575
[27,] -2.8319816  4.4410430
[28,] -3.9125633  1.2736444
[29,] -2.6930328  3.8005552
[30,] -2.9218073  3.0857923
[31,]  3.0857923 -2.9218073
[32,]  3.8005552 -2.6930328
[33,]  1.2736444 -3.9125633
[34,]  4.4410430 -2.8319816
```

```
[35,]  2.2966575 -5.0537676
[36,]  2.7326862 -2.6613964
[37,]  3.0149824 -3.7270742
[38,]  1.7037760 -3.6011636
[39,]  2.2067787 -4.3538977
[40,]  2.4369105 -3.5420838
[41,]  3.4362268 -2.6871726
[42,]  2.6322209 -5.3581478
[43,]  3.3535170 -0.9949036
[44,]  3.8158973 -3.7206556
[45,]  2.2097518 -3.3222148
[46,]  3.8208766 -3.0241445
[47,]  4.0858435 -1.3062023
[48,]  5.9742427 -1.8342443
[49,]  3.8663384 -2.4976632
[50,]  2.5411868 -3.3500325
[51,]  2.1339963 -4.4126433
[52,]  1.9133511 -3.7037302
[53,]  3.6609349 -2.3791313
[54,]  2.2839814 -3.4910438
[55,]  2.7108179 -2.2846241
[56,]  2.8971687 -1.8674317
[57,]  3.3544061 -3.1624108
[58,]  2.6153352 -4.0037068
[59,]  2.7003328 -2.4916507
[60,]  3.6835618 -3.6857495
```

Make a plot of `x`

```
plot(x)
```



K-means

The main function in "base" R for K-means clustering is called `kmeans()`: There are two required arguments in this function and these are x which is the data and centers which is the number of cluster you want.

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

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

Cluster means:
          x         y
1  3.022760 -3.162542
2 -3.162542  3.022760

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] 55.18667 55.18667
```

```
 (between_SS / total_SS =  91.2 %)

Available components:

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

The `kmean()` function returns a "list" with 9 components , YOu can see 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

The number you get is the number of points 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 center

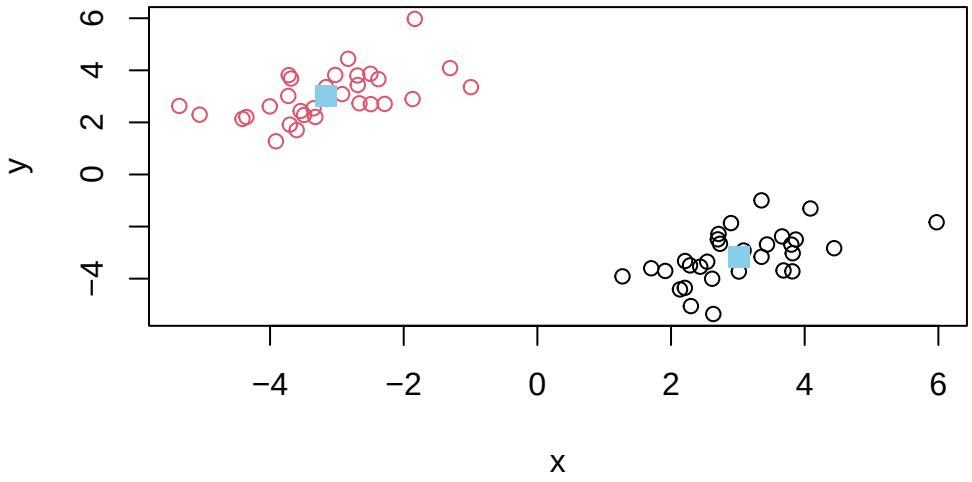
```
km$cluster
```

```
 [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. Make a plot of our `kmeans()` results showing cluster assignments using different color for each cluster/group of points and cluster center.

When we write the code `col=c("purple", "skyblue")`, it will tell R to alternate the color so one point purple and then the next skyblue, and we can't differentiate the two clusters. To do this, we will write the code `col=km$cluster` and that will color code our two cluster. pch refers to the shape of the point and then cex is the size of the point.

```
plot(x, col=km$cluster)
points(km$centers, col="skyblue", pch=15, cex=1.5)
```



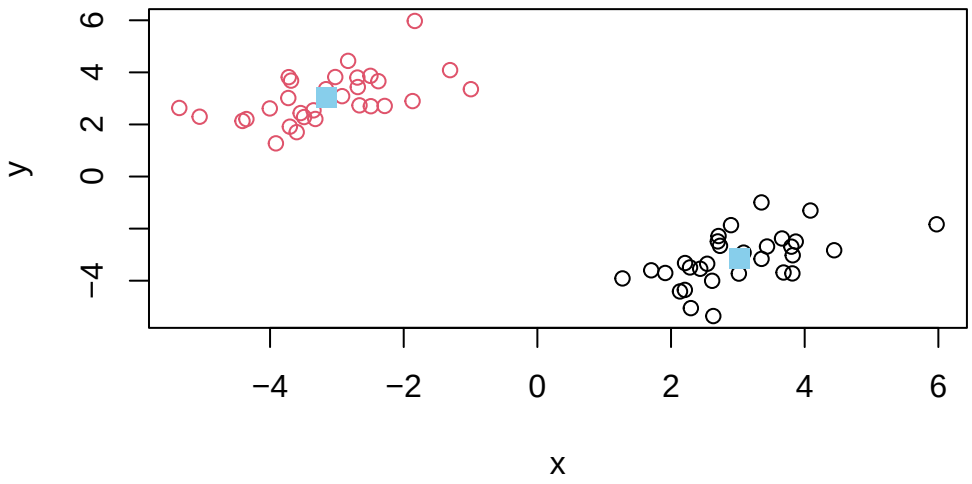
Q. Run `kmeans()` again on `x` and this cluster into 4 groups/clusters and plot the smae results like the figure as above.

```
km2<- kmeans(x,centers=4)
km2
```

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

Cluster means:
          x         y
1 -2.532012  3.612955
2 -3.987082  2.250967
```

```
3  2.250967 -3.987082
4  3.612955 -2.532012

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

Within cluster sum of squares by cluster:
[1] 18.626863  7.297622  7.297622 18.626863
 (between_SS / total_SS =  95.9 %)

Available components:

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

```r
plot(x, col=km$cluster)
points(km$centers, col="skyblue", pch=15, cex=1.4)
```



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 groupings do not exist. i.e its does what you tell it to do in terms of `center`

Hierarchical Clustering

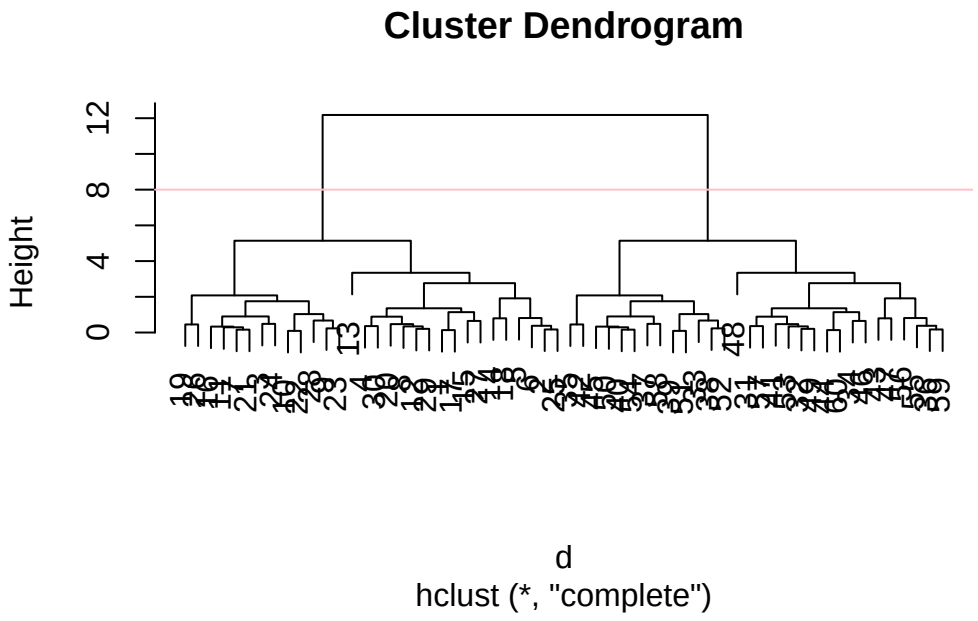
The main function in "base" R for hierarchical clustering is called `hclust()`. This functions has one required input d.You can't just pass our data set as is into `hclust()` you must give "distance matrix" as input. We can get this fro the `dist()` function in R.

```
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. When we plot it we get this speical graph that looks like a phylogentic tree.

```
plot(hc)
abline(h=8, col="pink")
```



To get our main cluster assignment (membership vector) we need to "cut" tree at the big goal posts. The code for this would be cutree()

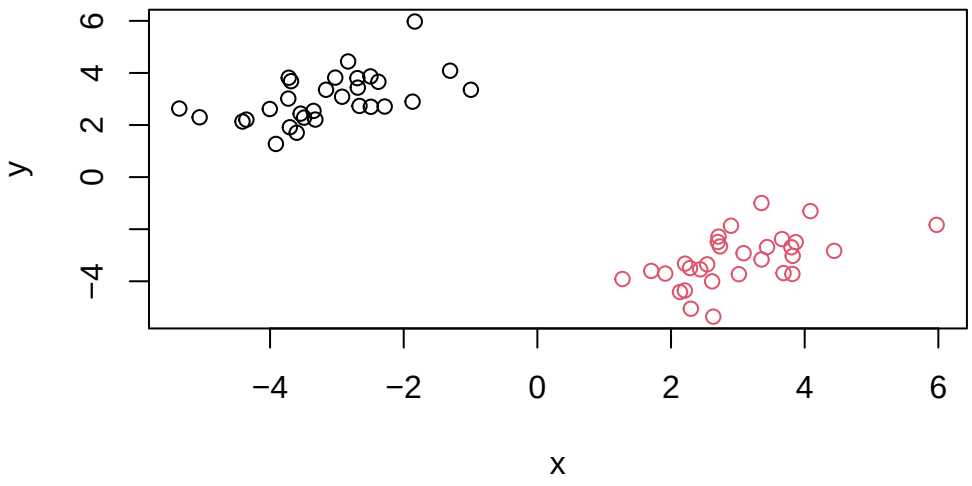
```
grps<- cutree(hc, h=8)
grps
```

```
 [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 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
```

```
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)

Principle Component Analysis (PCA)

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

Here we will anlayze 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
```

This way is not ideal because if we keep running it, we will get rid of one of the columns after every run...

```
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
```

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

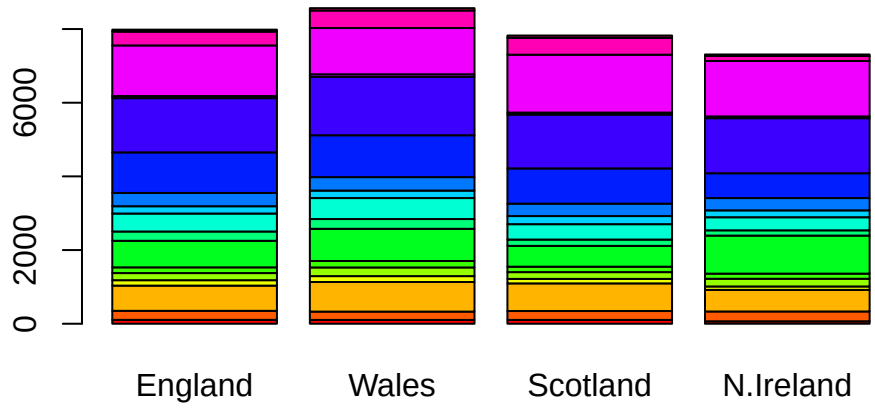
```
    Wales Scotland N.Ireland
105   103      103        66
245   227      242       267
685   803      750       586
147   160      122        93
193   235      184       209
156   175      147       139
```

A way to fix it is do what is this is to write this code

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

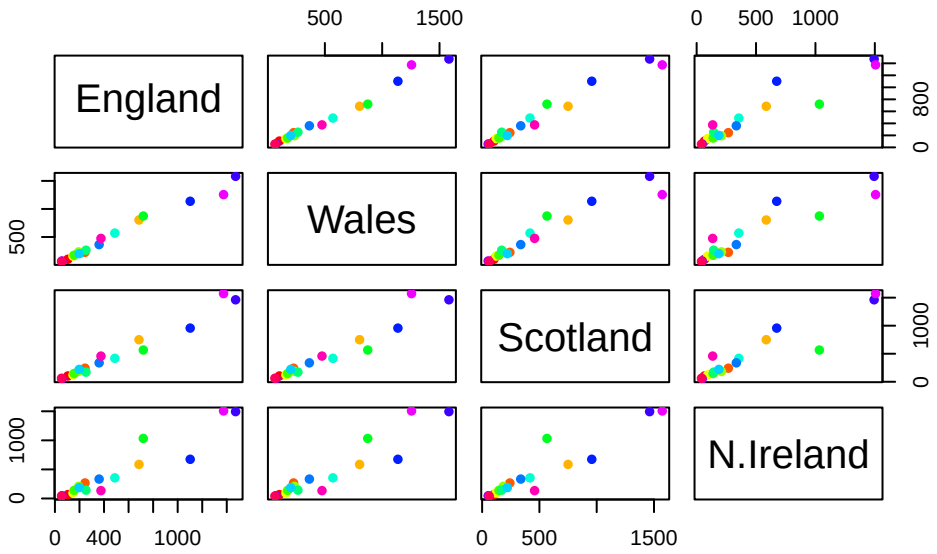
```
              England Wales Scotland N.Ireland
Cheese            105   103      103        66
Carcass_meat      245   227      242       267
Other_meat        685   803      750       586
Fish              147   160      122        93
Fats_and_oils     193   235      184       209
Sugars        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

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



One conventional plot that can be useful is called a "pairs" plot

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



For the second graph in the diagram, the y-axis is England, and the x-axis is Wales. Each dot on the graph represent the differnt food we are looking at in the data

PCA to the rescue

The main functions in the 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
```

The `prcomp()` function returns a list of options for our results with fice attributes/components.

```
attributes(pca)
```

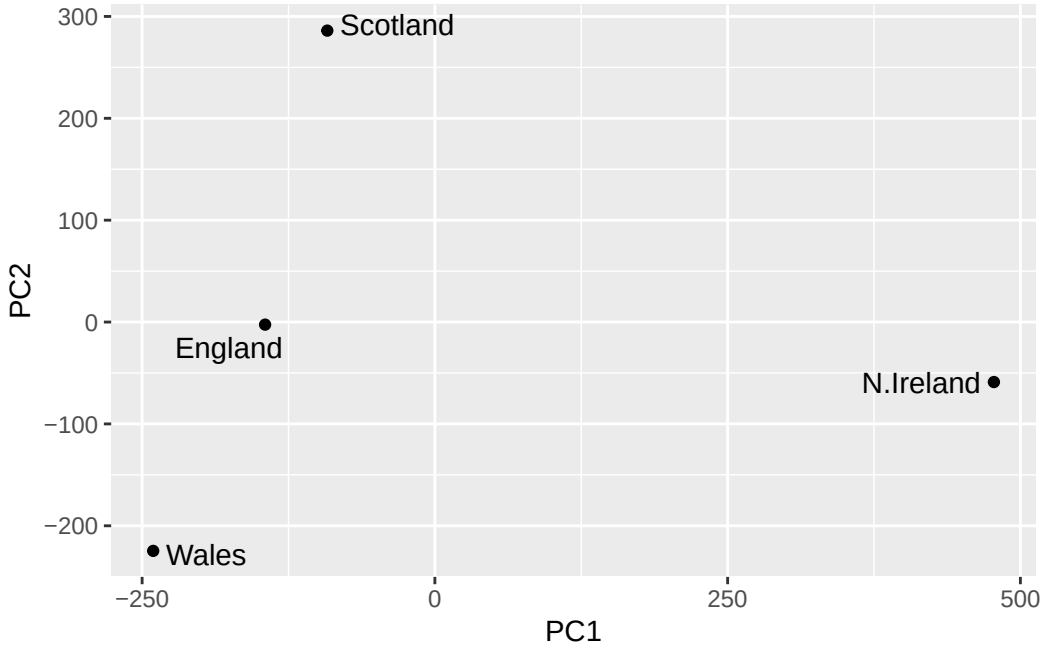
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

The two main "results" inhere are `pca$x` adn `pca$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
```

```
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()
```



The plot above is showing a plot that is comparing two different dimensions, PC1 and PC2. we are looking at the variance.

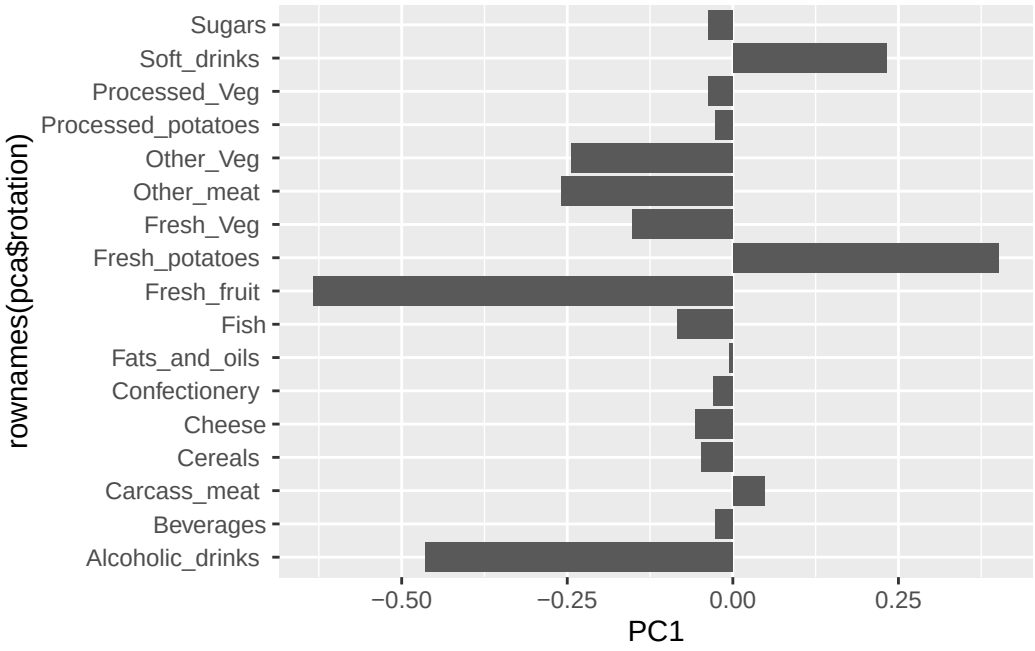
The second major result is curated in the `pca$rotaation` object or component. Lets plot this to see what PCA is picking up

```
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()
```



The barplot above shows the variance in the different food consumptions.PC1 is a dimension, and it will show the greatest amount of variance.

The PC number can be no more than the the number of food and also the countries.