Class 7: Machine Learning 1

Yoonjin Lim (PID: A16850635)

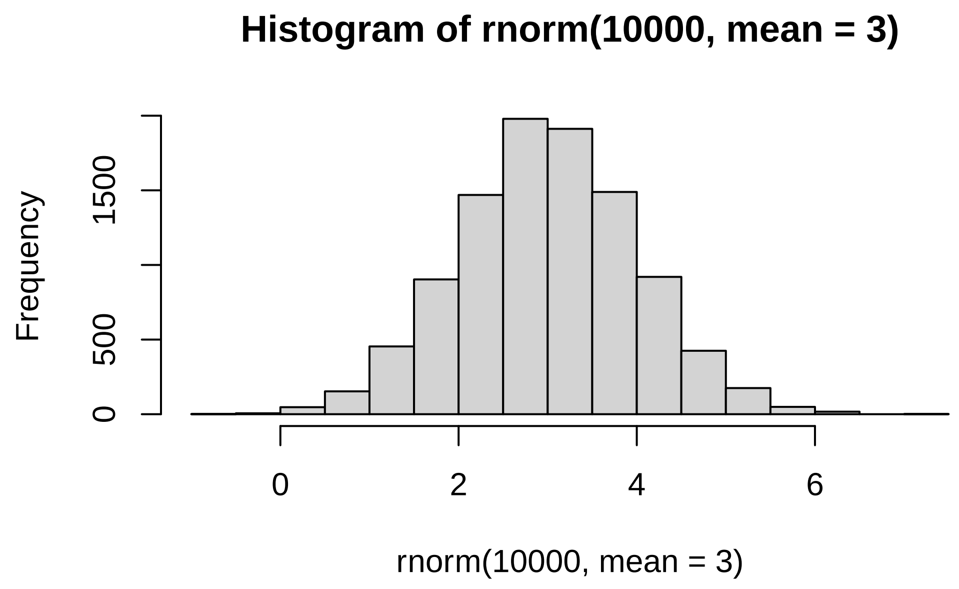
Table of contents

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.

```r
hist(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
```

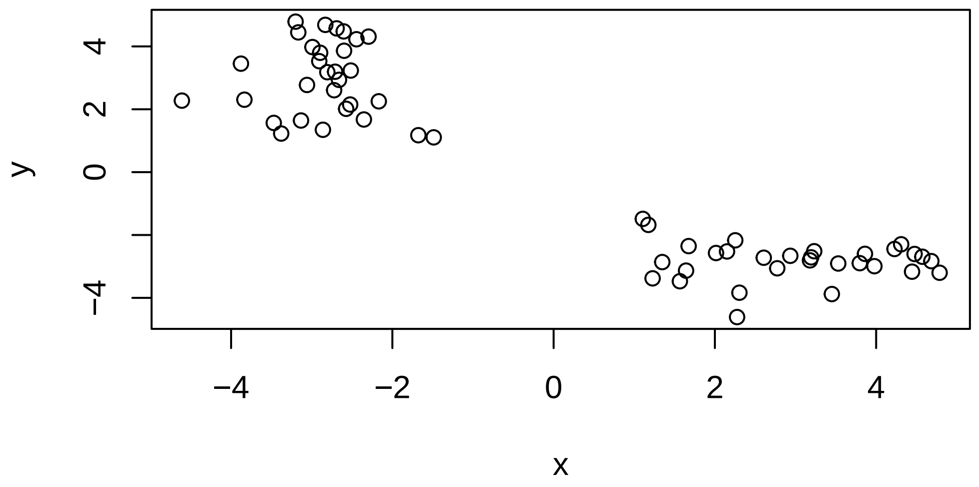
```
              x        y
 [1,] -2.991297 3.978344
 [2,] -3.834980 2.304359
 [3,] -2.691131 4.573042
 [4,] -2.603694 4.476778
 [5,] -2.598901 3.861289
 [6,] -3.470747 1.565748
 [7,] -3.166711 4.445206
 [8,] -2.806797 3.179574
 [9,] -2.168453 2.252648
[10,] -3.877895 3.450803
[11,] -2.661681 2.935304
[12,] -2.573306 2.014116
[13,] -2.906548 3.530135
[14,] -2.861164 1.347440
```

```
[15,] -2.710405  3.193785
[16,] -2.353080  1.674420
[17,] -2.895673  3.797106
[18,] -3.058580  2.773551
[19,] -2.515713  3.231854
[20,] -2.831384  4.684583
[21,] -2.523303  2.150658
[22,] -1.487225  1.106359
[23,] -3.132889  1.642689
[24,] -2.722205  2.606078
[25,] -2.444967  4.227515
[26,] -3.378592  1.227771
[27,] -4.610479  2.275412
[28,] -1.677803  1.174826
[29,] -3.200248  4.787185
[30,] -2.295523  4.310386
[31,]  4.310386 -2.295523
[32,]  4.787185 -3.200248
[33,]  1.174826 -1.677803
[34,]  2.275412 -4.610479
[35,]  1.227771 -3.378592
[36,]  4.227515 -2.444967
[37,]  2.606078 -2.722205
[38,]  1.642689 -3.132889
[39,]  1.106359 -1.487225
[40,]  2.150658 -2.523303
[41,]  4.684583 -2.831384
[42,]  3.231854 -2.515713
[43,]  2.773551 -3.058580
[44,]  3.797106 -2.895673
[45,]  1.674420 -2.353080
[46,]  3.193785 -2.710405
[47,]  1.347440 -2.861164
[48,]  3.530135 -2.906548
[49,]  2.014116 -2.573306
[50,]  2.935304 -2.661681
[51,]  3.450803 -3.877895
[52,]  2.252648 -2.168453
[53,]  3.179574 -2.806797
[54,]  4.445206 -3.166711
[55,]  1.565748 -3.470747
[56,]  3.861289 -2.598901
[57,]  4.476778 -2.603694
```

```
[58,]  4.573042 -2.691131
[59,]  2.304359 -3.834980
[60,]  3.978344 -2.991297
```

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 -2.835046  2.959299
2  2.959299 -2.835046
```

```
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] 51.89665 51.89665
 (between_SS / total_SS =  90.7 %)

Available components:

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

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

```r
attributes(km)
```

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

$class
[1] "kmeans"
```

> Q. How many points in each cluster?

```r
km$size
```

```
[1] 30 30
```

> Q. Cluster assignment/memebership vector?

```r
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
```

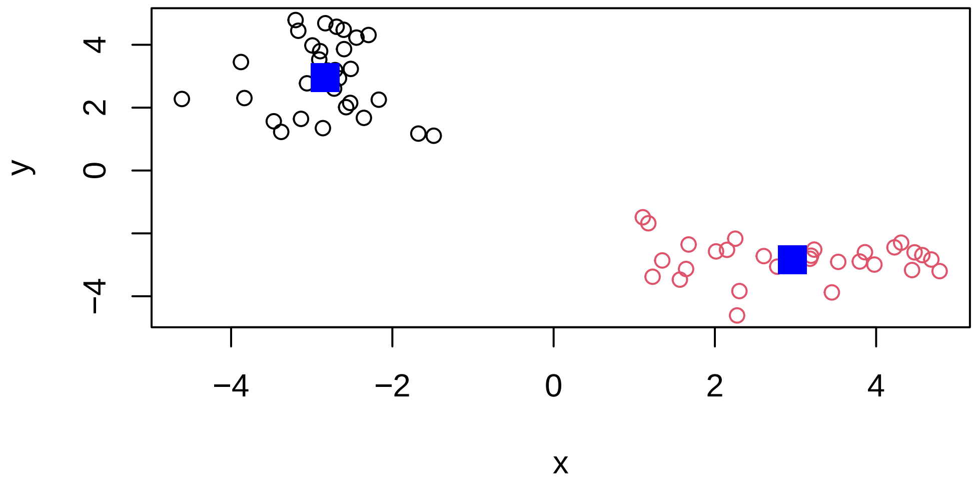
> Q. Cluster centers?

```
km$centers
```

```
          x         y
1 -2.835046  2.959299
2  2.959299 -2.835046
```

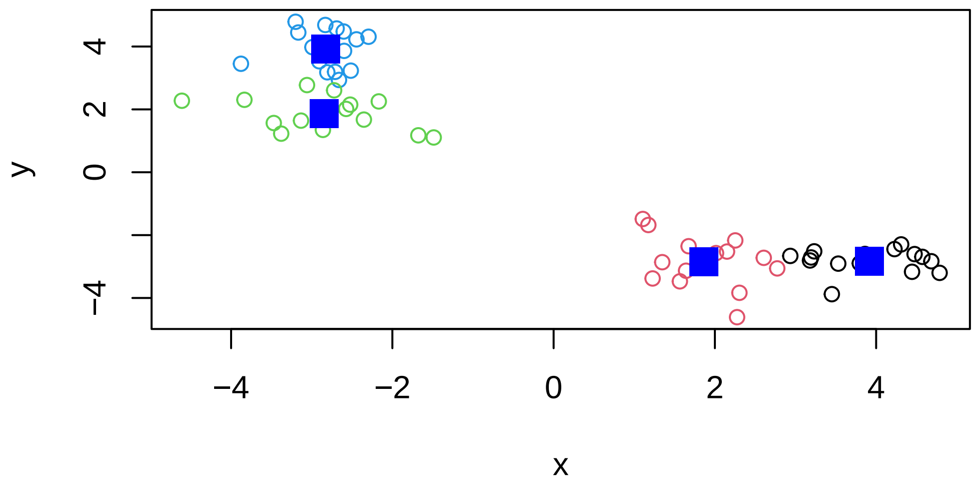
Q. Make a plot of our `kmeans()` results showing cluster assignment using different colors for each cluster/group of points and cluster centers in blue.

```
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)
```



key-point: K-means clustering is super popular but can be miss-used. One big limitation is that it can impose a clustering pattern on your data even if clear natural grouping don'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 "distancee matrix" as input. We can get this from the `dist()` function in R.

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

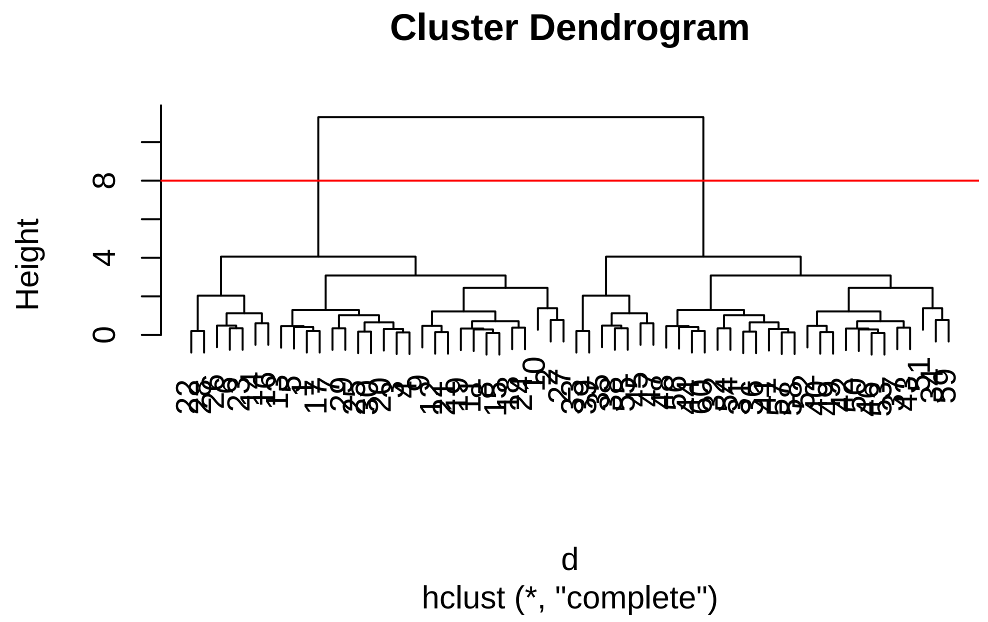
```

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.

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



To get our main cluster assignment (membership vector), we need to "cut" the tree at the big goal posts...

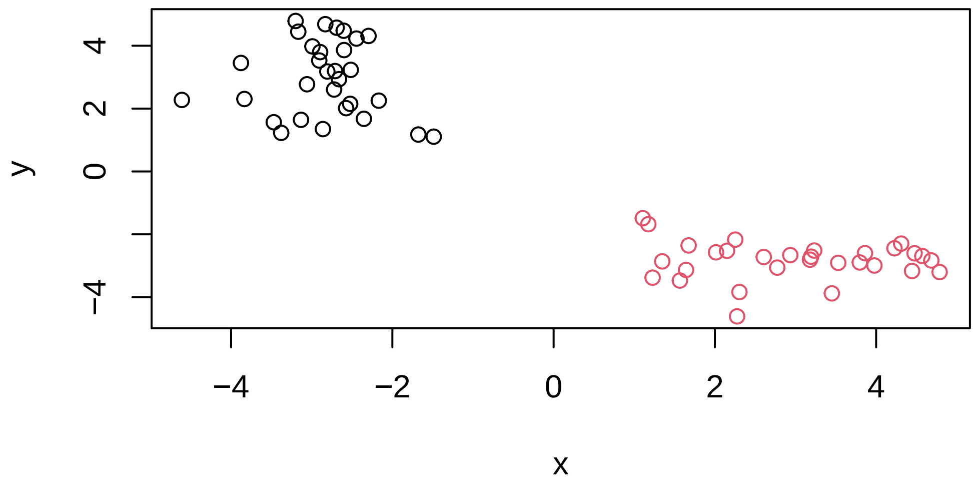
```r
grps <- cutree(hc, h=8)
grps
```

```
 [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
```

```r
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).

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
```

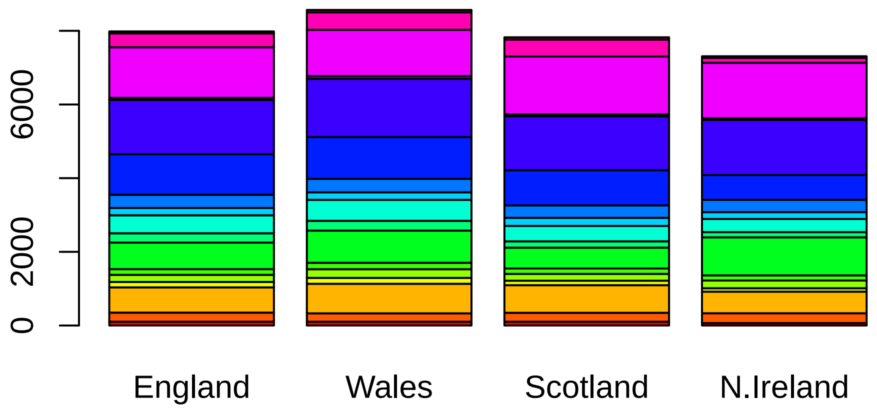
```
rownames(x) <- x[,1]
x <- x[,-1]
head(x)
```

```
              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)
```

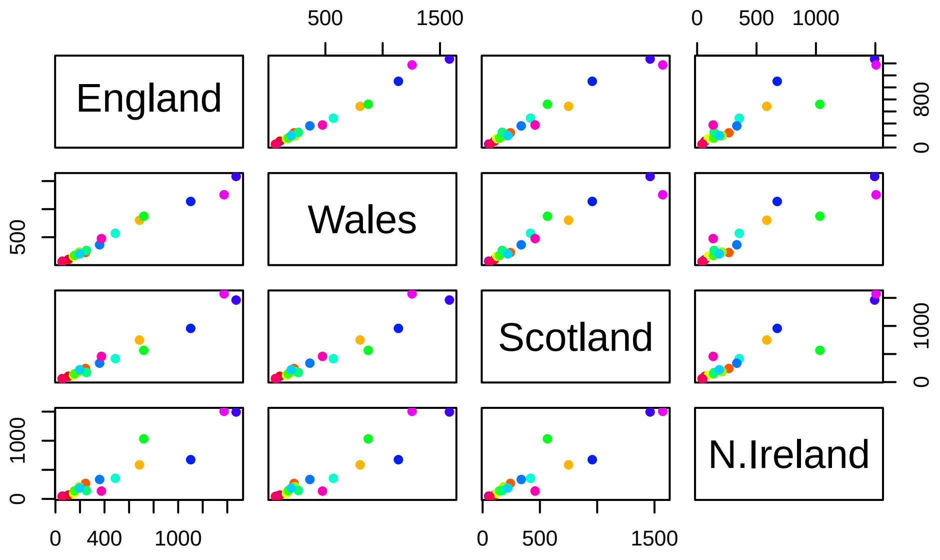
```
              England Wales Scotland N.Ireland
Cheese            105   103      103        66
Carcass_meat      245   227      242       267
Other_meat        685   803      750       586
Fish              147   160      122        93
Fats_and_oils     193   235      184       209
Sugars        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

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



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

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



PCA to the rescue

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

```r
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
```

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

```r
attributes(pca)
```

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

$class
[1] "prcomp"
```

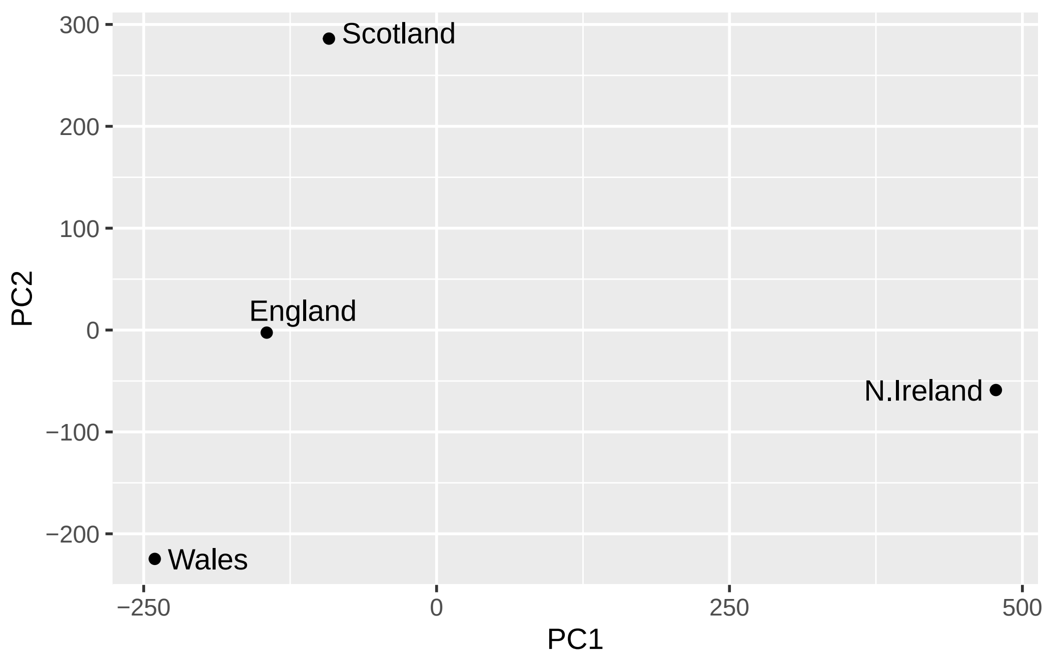
The two main "results" in here are `pca$x` and `pca$rotation`. The first of these (`pca$x`) contains the scores of the data on the new PC axis - we use these to make our "PCA plot".

```r
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
```

```r
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()
```

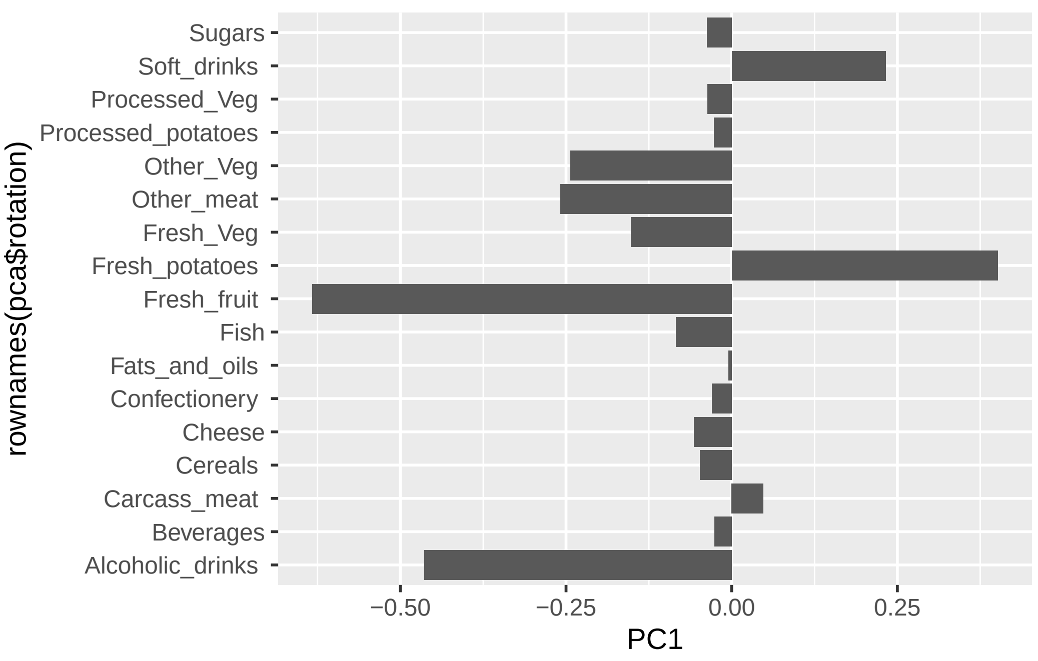


Notes: Using the ggplot2 packages, this results in a figure describing lists of vectors and the matrix between four different countries, to mainly represent the variation: how similar and dissimilar they are from each other. For example, Scotland, England, and Wales are on same area (the negative PC1), while N.Ireland are plotted in the other area (the positive PC1)

So first we will need to take whatever it is we want to plot and convert it to a data.frame with the as.data.frame() function. Then to make our plotting life easier we will also add the food labels as a column (called "Food") to this data frame with the rownames_to_column() function from the tibble package (you might need to install this):

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()
```



Notes: based on the graph above, this plot specifically describes the proportion of each component of food in bar representation, based on PC1 value. Therefore, for example, fresh fruit and alcoholic drinks, being represented significantly high toward the negative PC1, is likely being consumed in three countries, Scotland, England, and Wales. On the other hand, soft drinks and fresh potatos, being represented significantly high toward the positive PC1, is likely consumed in N.Ireland.