Class 007 - Machine Learning 1

R. D. Carias (PID: A18573289)

Table of contents

Background 1
K - Means Clustering 2

Annotation: `plot`(x, col = c("red","blue")) cycles one point red, one blue, one red, etc. 4

Annotations: Why this is "self-fulfilling": 7
Hierarchical Clustering 7

Notes: Notice that , 1-30 is one main branch and the other is 31-60. Recall, these diagrams can rotate at nodes. The y-axis shows the distance (height) at which observations or clusters are merged, indicating how dissimilar they were at the time of joining. 8
Dimensionality Reduction / Principle Component Analysis (PCA) 9
PCA of UK Food Data 10
Spotting Major Differnces and trends 12
Principal Component Analysis PCA - "To the rescue!" 15

Background

Today we are going to explore introduction to machine learning methods. e.g. **Clustering**, **Dimensionality**, and **Reduction**.

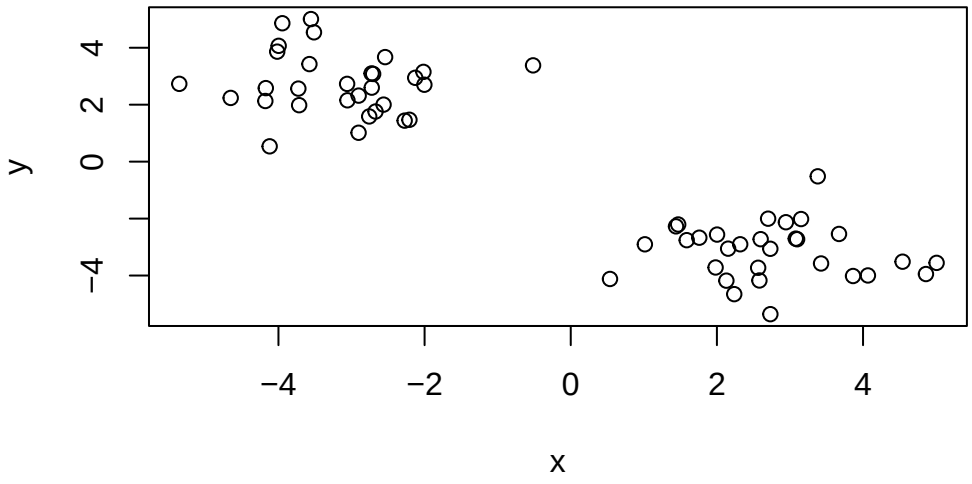
Q. Generate 30 random numbers cenetered at +3 and another 30 centered at -3?

Hint: rbind and cbind are partners in crime. r for row and c for columns.

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

tmp
```

```
 [1]  1.9821975  2.5645862  5.0067904  4.8607505  1.4702041  2.2373261
 [7]  2.5819675  3.0792137  4.0667997  4.5409758  3.0977127  2.0024138
[13]  2.7318258  3.3803393  2.3185982  1.7592992  3.8632237  1.0143633
[19]  2.1296908  1.4403239  3.1520449  3.4254915  0.5372165  2.1537446
[25]  2.9445129  3.6711947  2.5989917  2.7009718  2.7309531  1.5871557
[31] -2.7577510 -3.0602982 -2.0030078 -2.7229791 -2.5400504 -2.1281788
[37] -3.0557831 -4.1204656 -3.5761126 -2.0159306 -2.2726265 -4.1793405
[43] -2.9032936 -4.0168335 -2.6713664 -2.9023237 -0.5155960 -5.3570754
[49] -2.5615490 -2.7255102 -3.5136812 -3.9969604 -2.7040324 -4.1729393
[55] -4.6536222 -2.2074635 -3.9469887 -3.5549420 -3.7276991 -3.7166726
```

```r
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()`:

x: numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).

centers:
either the number of clusters, say k k, or a set of initial (distinct) cluster centers. If a number, a random set of (distinct) rows in x is chosen as the initial centers.

```
kmeans(x, centers = 2)
```

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

Cluster means:
          x         y
1  2.721029 -3.142702
2 -3.142702  2.721029

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 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] 61.31753 61.31753
 (between_SS / total_SS =  89.4 %)

Available components:

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

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

Q. What components of the results objects details the cluster sizes?

```
km$size
```

```
[1] 30 30
```

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

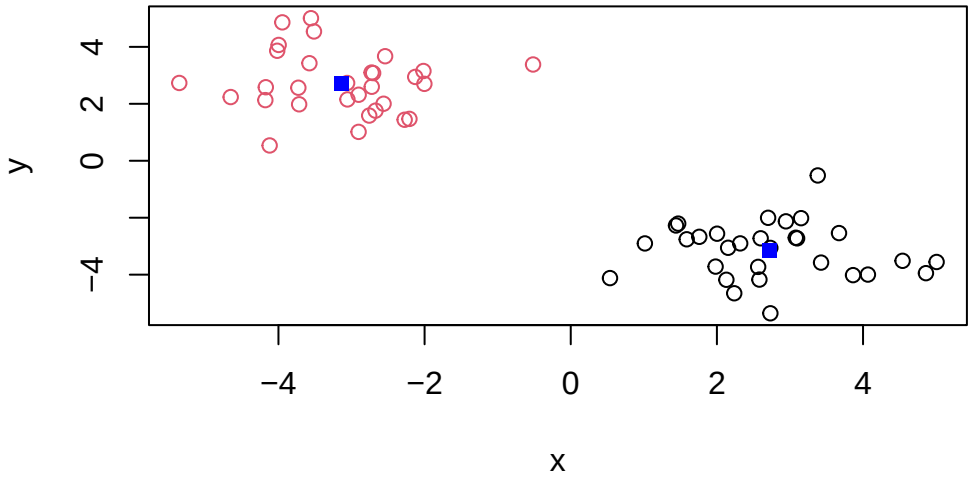
Q. What component of the reresults object details the cluster membership vector(i.e. our main results of which points lie in which cluster)?

```
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 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. Plot our clustering results with points colored by cluster and also add a blue point at the center of both centers.

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



Annotation: `plot(x, col = c("red","blue"))` cycles one point red, one blue, one red, etc.

Q. Run `kmeans()` again and this time produce 4 clusters and call your results `km4`. Can you make results like the figures above?

```
kmeans(x, centers = 4)
```

```
K-means clustering with 4 clusters of sizes 30, 7, 8, 15

Cluster means:
```

```
          x         y
1  2.721029 -3.142702
2 -3.994656  4.070837
3 -2.169411  3.078123
4 -3.264213  1.900669

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 4 4 3 3 3 3 4 4
[39] 2 3 4 4 4 2 4 4 3 2 4 3 2 2 3 4 4 4 2 2 4 4

Within cluster sum of squares by cluster:
[1] 61.317527  6.432363  4.665507 13.470189
 (between_SS / total_SS =  92.6 %)

Available components:

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

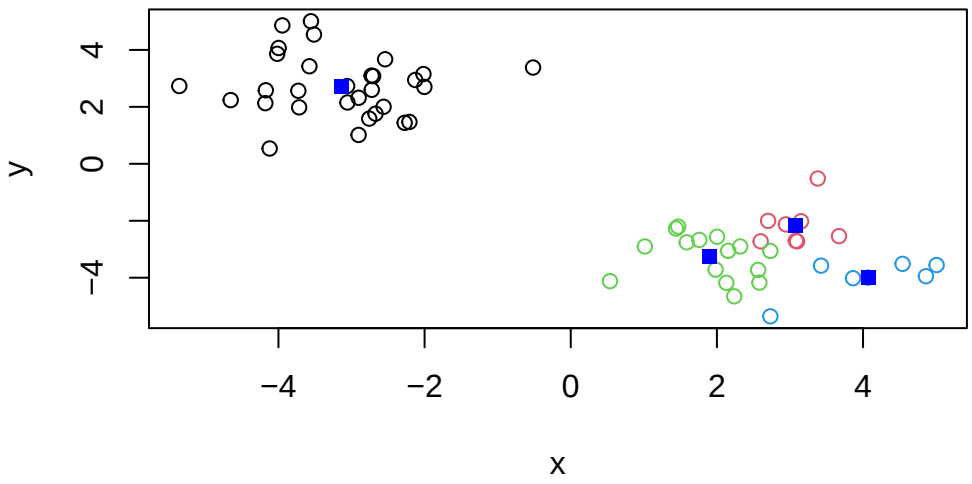
```r
km4 <- kmeans(x, centers = 4)

km4$size
```

```
[1] 30  8 15  7
```

```r
km4$cluster
```

```
 [1] 3 3 4 4 3 3 3 2 4 4 2 3 4 2 3 3 4 3 3 3 2 4 3 3 2 2 2 2 3 3 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 1 1
```

```r
plot(x, col = km4$cluster)
points(km4$centers, col='blue', pch=15)
```



The Metric

```
km$tot.withinss
```

```
[1] 122.6351
```

```
km4$tot.withinss
```

```
[1] 85.88559
```

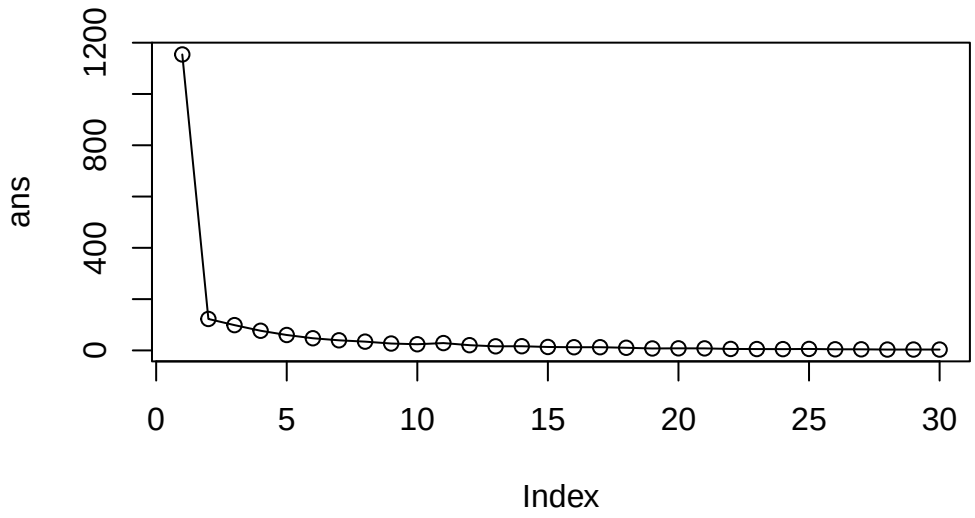
> Q. Let's try a different number of K (centers) from 1 to 30 and see what the best reult is?

Ex: Fore-loops

```
ans <- NULL

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

ans
```

```
 [1] 1154.135559  122.635055   98.548767   76.599231   60.120867   47.457687
 [7]   39.445197   34.179755   26.843067   23.972765   28.554923   20.395349
[13]   15.953724   16.286752   13.412604   12.405340   12.500702   10.242823
[19]    7.633666    8.118479    7.904693    5.708464    5.266358    4.935431
[25]    5.716144    4.389187    4.284550    3.285495    3.476075    3.298302
```

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



Annotations: Why this is "self-fulfilling":

You define "good clustering" as low within-cluster variance, then choose k based on which value minimizes within-cluster variance.

That is circular.

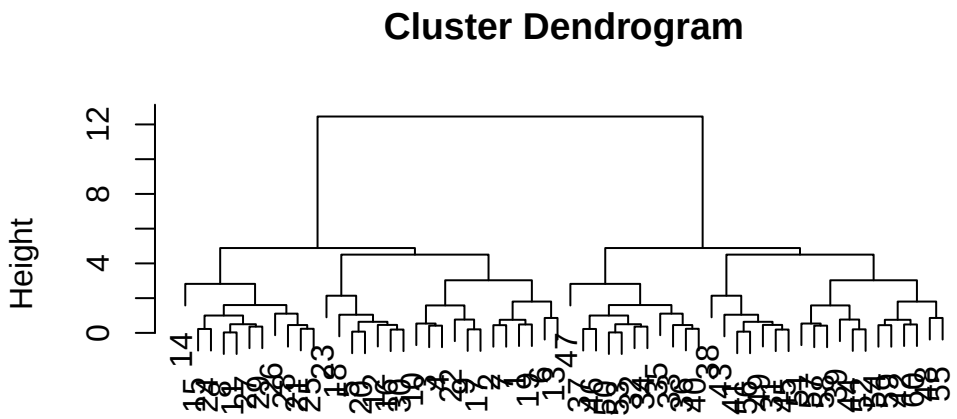
Hierarchical Clustering

The results from this will elucidate the structure of this hierarchy.

`hclust()`(d, method = "complete", members = NULL)

- **Unlike** `kmeans()`, which does all the work for you, you can't just pass 'hclust()' It wants/needs a distance matrix / dissimilarity matrix, this is like the output/return of `dist()` function. `dist()` automatically calculates euclidean distance.
- "Instead of giving you coordinates, we are giving distances from every other point."

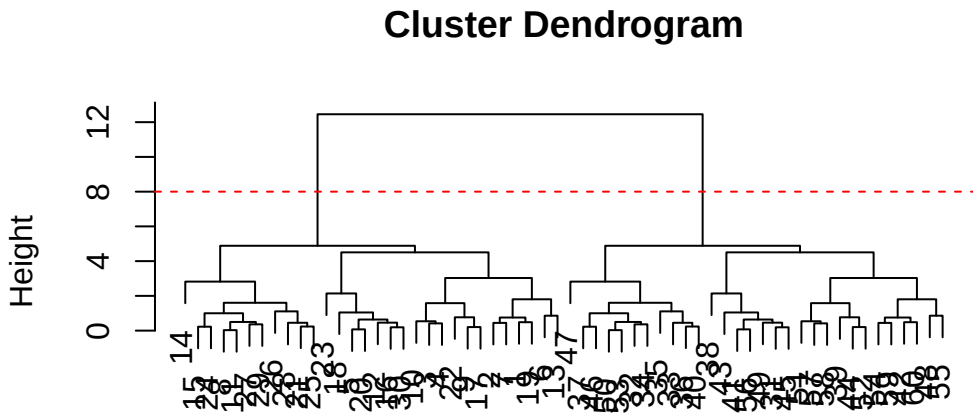
```
d <- dist(x)
hc <- hclust(d)
plot(hc)
```



Notes: Notice that , 1-30 is one main branch and the other is 31-60. Recall, these diagrams can rotate at nodes. The y-axis shows the distance (height) at which observations or clusters are merged, indicating how dissimilar they were at the time of joining.

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

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




```
#Not cut yet, just indicates where we are cutting. Why 8? This is where the two clusters we a
```

To accomplish this we use `cutree()` function on our `hclust` object.

```
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
```

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

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

Dimensionality Reduction / Principle Component Analysis (PCA)

- We are attempting to reduce the features of dimensionality while losing as little information as possible.

PCA = Eigenvectors ; Is "like" a big funnel. Where our output allows us to investigate our data.

PCA of UK Food Data

First, import the dataset required.

```
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?

```
dim(x)
```

```
[1] 17  5
```

One potential solution, is to set the row names, doing so "manually".

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

Removing the first columns can be accomplished by -1 to start indexing in column.

```
x <- x[,-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
```

The issue with this, you start to lose columns / data.

This can also be accomplished by arguing 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

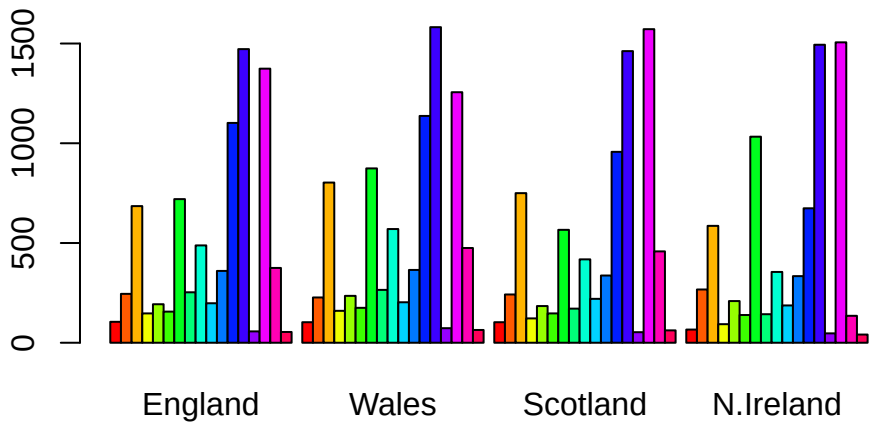
Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

- We rather argue and set the name immediately when we read in the csv. Doing so, allows us to start data d.f manipulations from the get go.

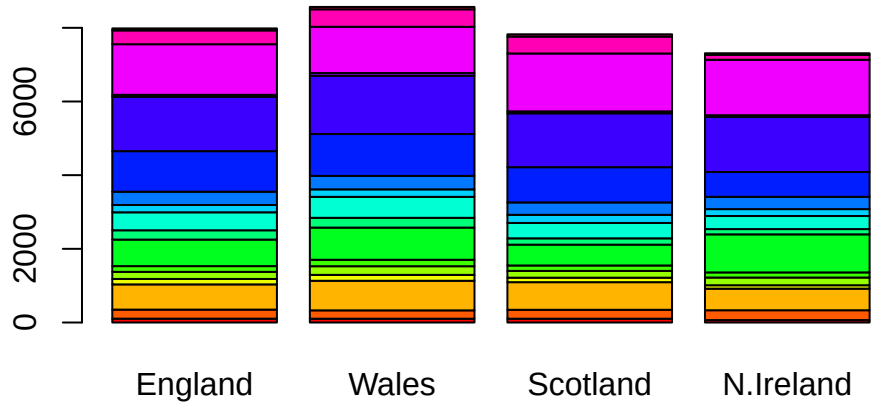
Spotting Major Differnces and trends

This is difficult, even in a wee little graph like this with 17D.

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



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



Q4: Changing what optional argument in the above ggplot() code results in a stacked barplot figure?

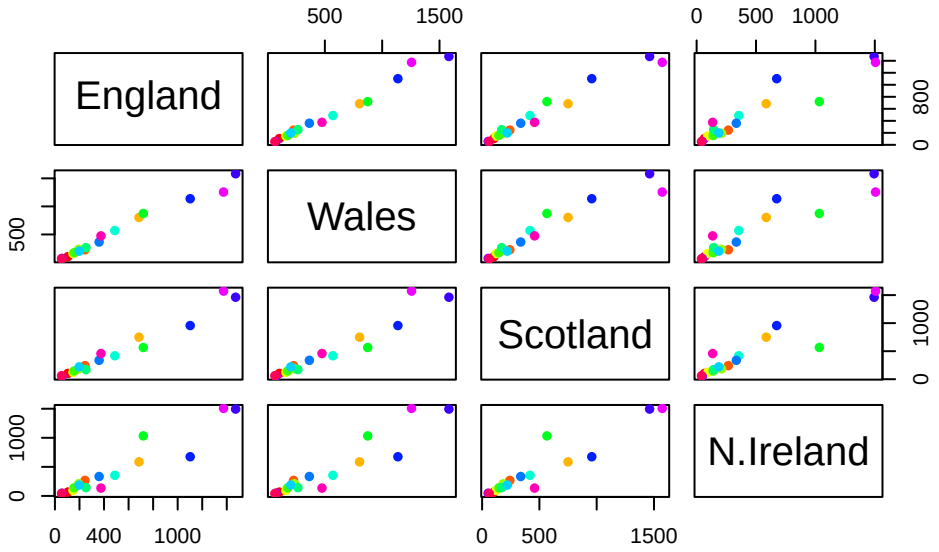
-

Q5: We can use the pairs() function to generate all pairwise plots for our countries. Can you make sense of the following code and resulting figure? What does it mean if a given point lies on the diagonal for a given plot?

-

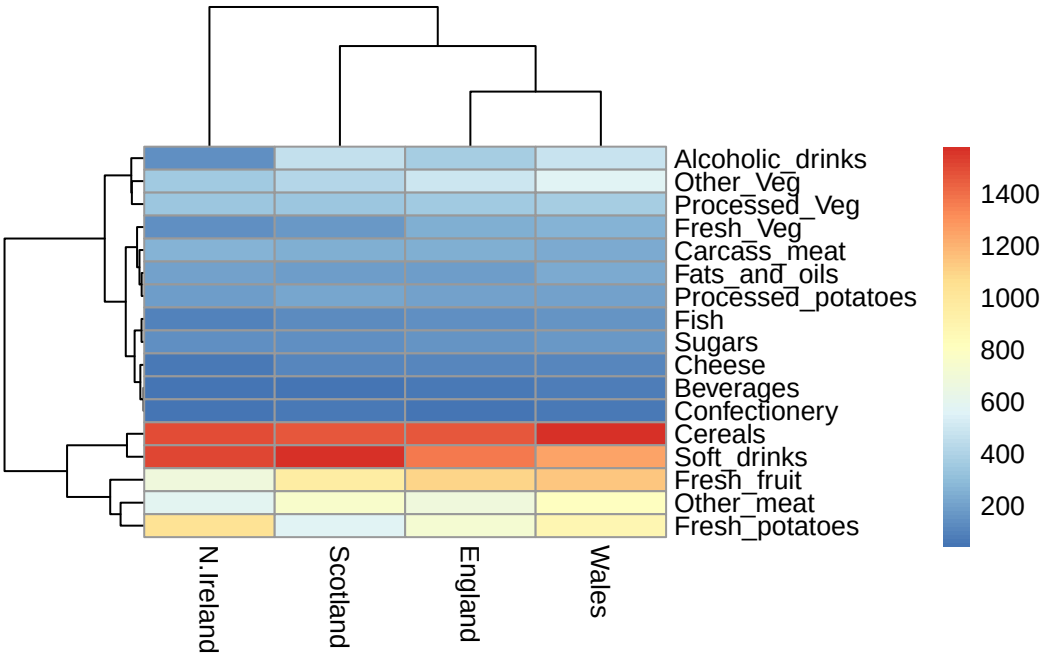
pairs(x, col=rainbow(nrow(x)), pch=16)

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



Observe the top row, from left to right. On the Y-axis for these plots, you have England's. All dots represent the food categories.

```
library(pheatmap)

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



Q6. Based on the pairs and heatmap figures, which countries cluster together and what does this suggest about their food consumption patterns? Can you easily tell what the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

•

Principal Component Analysis PCA - "To the rescue!"

The main PCA function in **base R**, which is probably the most difficult to use, is 'prcomp()'. This functions wants the transpose of our food data (i.e. the foods as columns and the counties as rows)

```
pca <- prcomp(t(x)) #t(x) = transpose matrix.
```

```
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
```

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

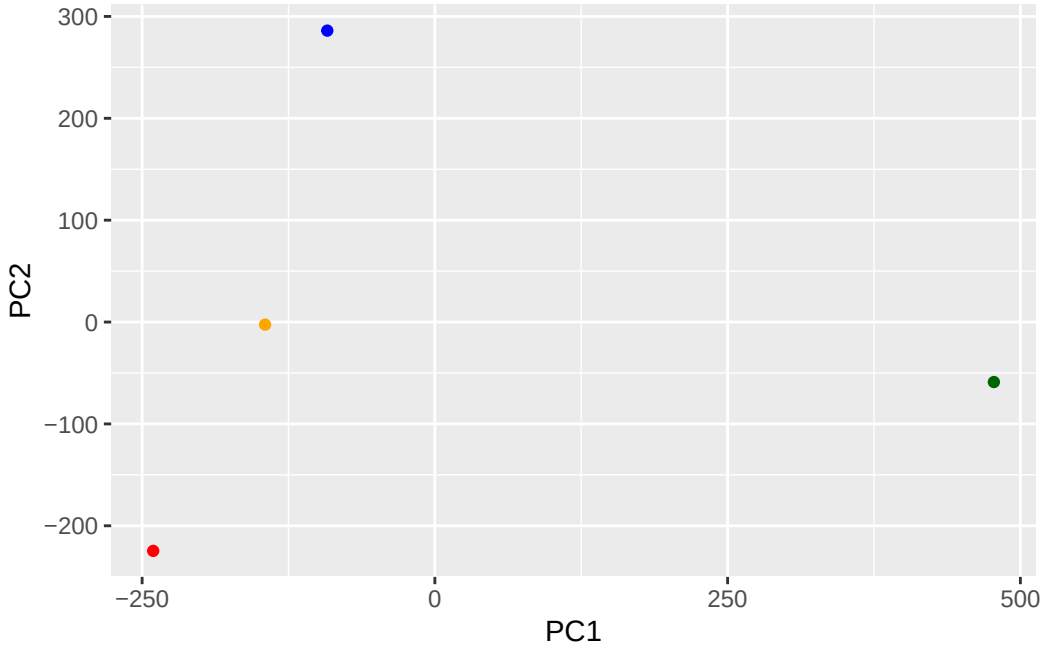
To make one of our main PCA result figures, we turn to `PCA$x` the scores along our new PCs. This is referred to as score plots, pc plots, ordination plots and various others.

```
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
```

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

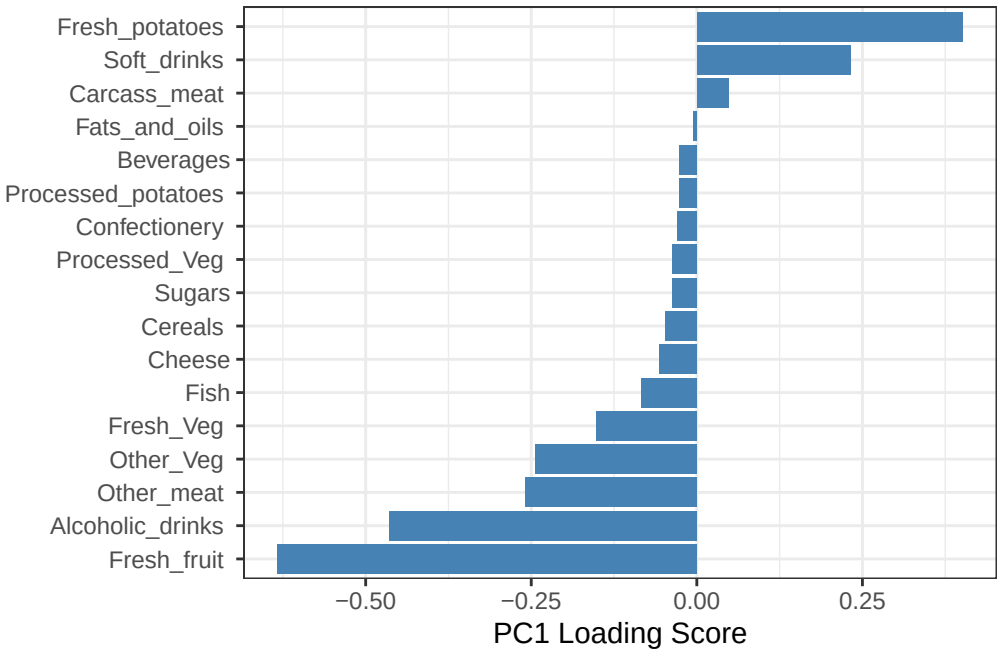
```
library(ggplot2)

ggplot(pca$x) + aes(PC1, PC2) + geom_point(col=my_cols)
```



The second major result figure is called a “loadings plot” of “variable contributions plot” or “weight plot” # Lots of Terminology

- How do the original variables weigh the respective axis. Think , weighted grading.

```
ggplot(pca$rotation) +
  aes(PC1, y = reorder(rownames(pca$rotation), PC1)) +
  geom_col(fill = "steelblue") +
  xlab("PC1 Loading Score") +
  ylab("") +
  theme_bw() +
  theme(axis.text.y = element_text(size = 9))
```



This can show us, why are these cells active compared to others, well, due to genes which are up- regulated compared to the other .