Class 7: Machine Learning 1

Christopher Levinger (A17390693)

Table of contents

Clustering

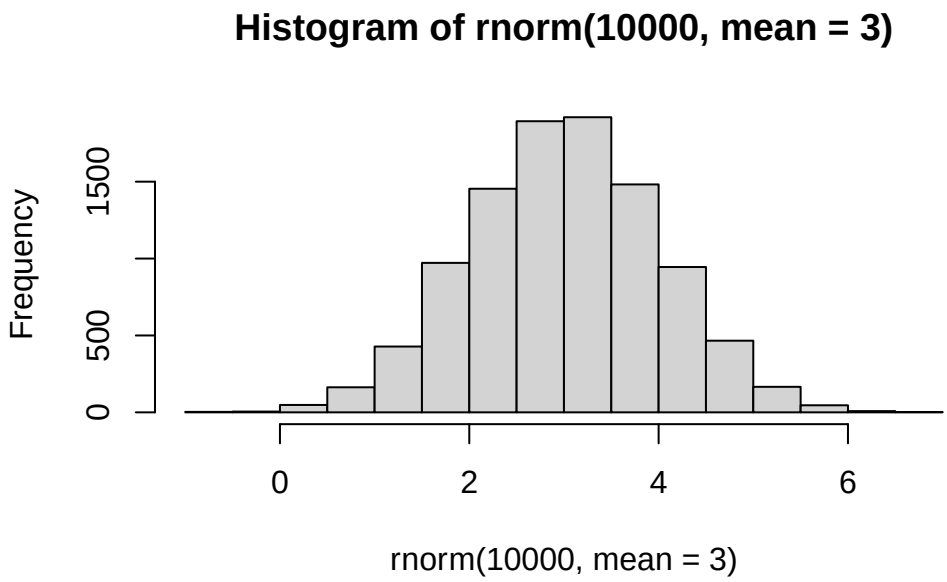
Today we will explore unsupervised machine learning methods starting with clustering and dimensionality reduction.

```
rnorm(10)
```

```
 [1] -0.4048155  0.8275925  1.3416045 -2.1657533  0.6076026 -0.1813111
 [7] -1.2650151  1.3234934  1.5547708  0.7079245
```

Return 1000 numbers centered on 3 in histogram

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



This histogram depicts a normally distributed set of data that is centered on a mean, median, and mode of 3, where all data points are symmetrically located around 3.

Return 30 numbers centered on -3

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

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

x
```

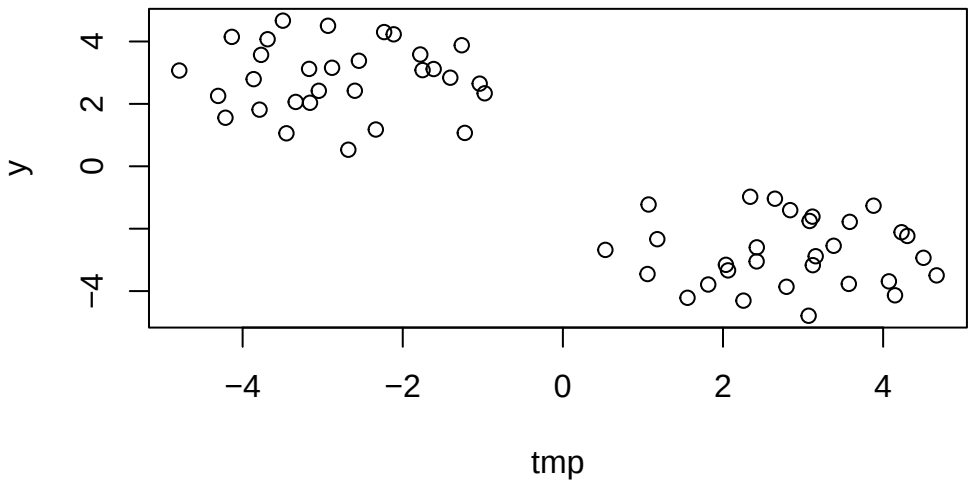
```
             tmp          y
 [1,] -4.3058905  2.2554221
 [2,] -3.1696305  3.1217749
 [3,] -1.6137528  3.1171719
 [4,] -3.8623507  2.7931088
 [5,] -3.7887958  1.8153137
 [6,] -1.2250413  1.0701916
 [7,] -1.7811717  3.5829862
 [8,] -2.5985172  2.4220115
 [9,] -3.0486716  2.4201229
[10,] -3.6879035  4.0722236
```

```
[11,] -2.5475974  3.3822178
[12,] -4.7912310  3.0694008
[13,] -2.3365477  1.1791638
[14,] -2.9337318  4.5037103
[15,] -3.7698315  3.5714186
[16,] -2.2324601  4.3021680
[17,] -1.0390912  2.6486194
[18,] -4.1350405  4.1482328
[19,] -2.1133736  4.2303716
[20,] -2.6798533  0.5309060
[21,] -3.4529917  1.0570523
[22,] -3.4972122  4.6682766
[23,] -3.1585347  2.0364805
[24,] -2.8831750  3.1576014
[25,] -1.2631833  3.8802977
[26,] -3.3369640  2.0630454
[27,] -1.4061342  2.8392847
[28,] -1.7526794  3.0820401
[29,] -0.9768403  2.3398549
[30,] -4.2145521  1.5566318
[31,]  1.5566318 -4.2145521
[32,]  2.3398549 -0.9768403
[33,]  3.0820401 -1.7526794
[34,]  2.8392847 -1.4061342
[35,]  2.0630454 -3.3369640
[36,]  3.8802977 -1.2631833
[37,]  3.1576014 -2.8831750
[38,]  2.0364805 -3.1585347
[39,]  4.6682766 -3.4972122
[40,]  1.0570523 -3.4529917
[41,]  0.5309060 -2.6798533
[42,]  4.2303716 -2.1133736
[43,]  4.1482328 -4.1350405
[44,]  2.6486194 -1.0390912
[45,]  4.3021680 -2.2324601
[46,]  3.5714186 -3.7698315
[47,]  4.5037103 -2.9337318
[48,]  1.1791638 -2.3365477
[49,]  3.0694008 -4.7912310
[50,]  3.3822178 -2.5475974
[51,]  4.0722236 -3.6879035
[52,]  2.4201229 -3.0486716
[53,]  2.4220115 -2.5985172
```

```
[54,]  3.5829862 -1.7811717
[55,]  1.0701916 -1.2250413
[56,]  1.8153137 -3.7887958
[57,]  2.7931088 -3.8623507
[58,]  3.1171719 -1.6137528
[59,]  3.1217749 -3.1696305
[60,]  2.2554221 -4.3058905
```

Make a plot of `x`

```
plot(x)
```



This plot simply depicts the values x representing tmp, plotted against y, which represents rev(tmp) or rev(x). Since the variable x consists of a vector of 30 elements, the variable simply represents the same set of values in vector form, except that the 30th element of vector x is the 1st element of vector y and so on. These values plotted against each other can be seen in the figure above. Here, it appears as though clusters begin to from, which will be helpful in our coming k means analysis. ### K-means

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

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

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

Cluster means:
        tmp         y
1  2.830570 -2.786758
2 -2.786758  2.830570

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] 68.84741 68.84741
 (between_SS / total_SS =  87.3 %)

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] 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 centers?

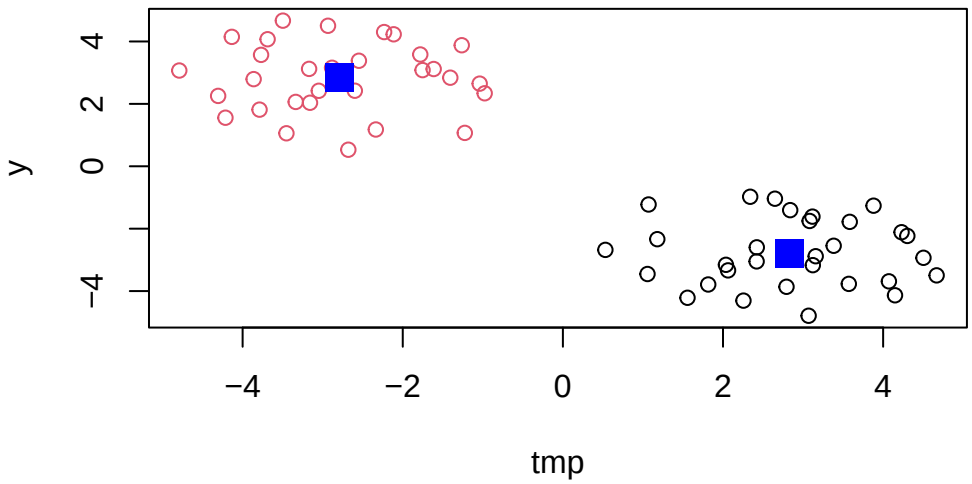
```
km$centers
```

```
        tmp         y
1  2.830570 -2.786758
2 -2.786758  2.830570
```

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

km$cluster will color each of them by cluster

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



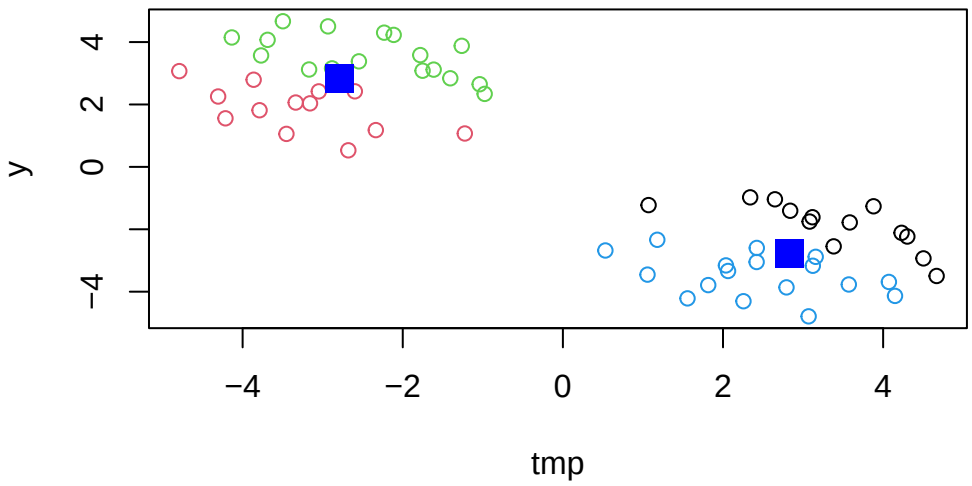
Here, using the same variables of x and y representing the reverse of all of the elements of x, we see two clearly defined clusters using the k means method. The k means clustering set

up involves randomly selecting a set of points based on the number of clusters selected and forming two clusters essentially based on sets of points that are close in distance to each other. It chooses these initial points to start with randomly and thus for the most efficient clustering, multiple iterations can be run. In this graph, we see two clearly defined clusters established with this method, where the dark blue square represents the cluster center. These clusters are representing the areas where x and rev(x) vector elements overlap and are close to each other and thus group in one cluster. Thus, these overlapping elements are likely the first elements of the x vector and the last elements of the y vector grouping together by being close in distance and vice versa for the other cluster.
>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)
km4
```

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

Cluster means:
        tmp         y
1  3.357476 -1.875559
2 -3.292303  1.866835
3 -2.400165  3.567544
4  2.427642 -3.483558

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

Within cluster sum of squares by cluster:
[1] 19.18660 17.62833 24.04844 24.24396
 (between_SS / total_SS =  92.2 %)

Available components:

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

```
plot(x, col=km4$cluster)
  points(km$centers, col="blue", pch=15, cex=2)
```



In this figure above, the same k means clustering method is used on the two vectors x and y, where y is the reverse of (x). In this case the number of clusters was set at 4, and created similar to the method described before, where random points were chosen and clsuters made based on the closest distance between these points. As clearly seen between this figure and the one above, there exist 2 distinct clusters in the data, and attempting to impose further division of the data into 4 clusters is not a "clear natural grouping" as seen above, where it appears some of the clusters in the top right seem to potentially overlap. The different clusters are depicted in their separate colors, but only 2 large clusters centers appear because the 3 in the top right are so close to each other and potentially overlap that one large cluster center, encompassing them all, thus showing how the previous cluster method in the last figure is more natural. > **keypoint** k-means clusterimg is super popular but can be miss-used. One big limitation is that it can impose a clustering patern 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`.

Hierarchal Clustering

The main function in "base" R for Hierarchical Clustering is called `hclust()`.

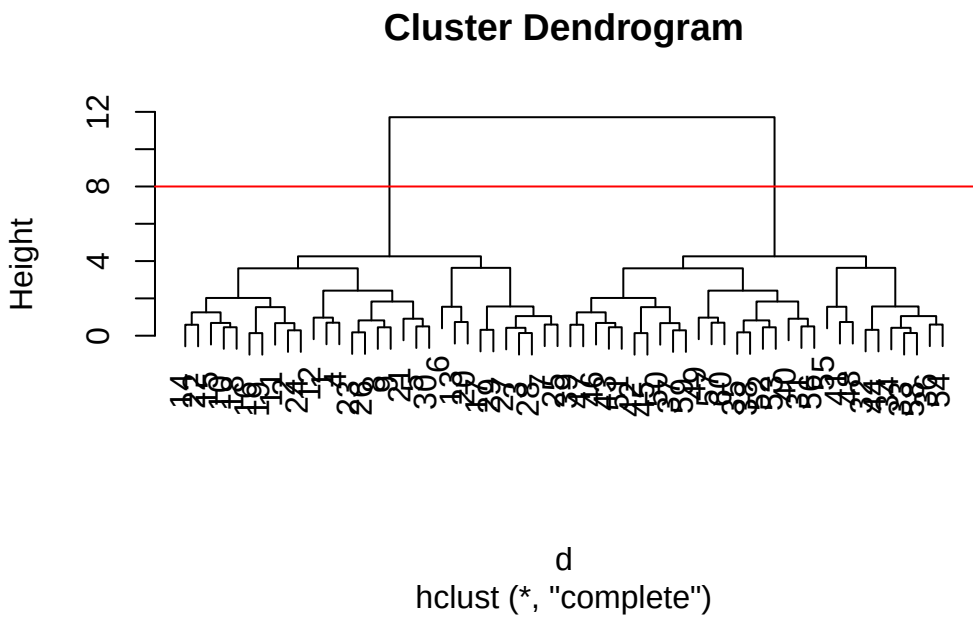
You can't just pass our dataset as is into `hclust()`. You must give "distance matrix" as input. We can get this from the `dist()` function in R.

```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")
```



This dendogram takes the input of our previous vector x as a distance matrix and uses bottom-up hierarchical clustering to construct this figure. It starts by connecting the closest points together seen in the roots of this dendogram and then starts connecting farther away points into larger clusters seen by the top of the dendogram, where the "cross bars" have a larger

height, therefore indicating a larger distance between the points used to connect and form the larger cluster. This dendogram depicts similar overall concepts to the k-means clustering in the way two distinct roots of this dendogram group separately with a small height indicating close distance between the elements of these smaller clusters on the left and right. Then connecting these faraway clusters into a larger cluster we see a cross bar with a very large height, showing a different cluster method than k means but representing similar conclusions in a different fashion, when the k means was specifically set to 2, representing the most natural grouping of the data.

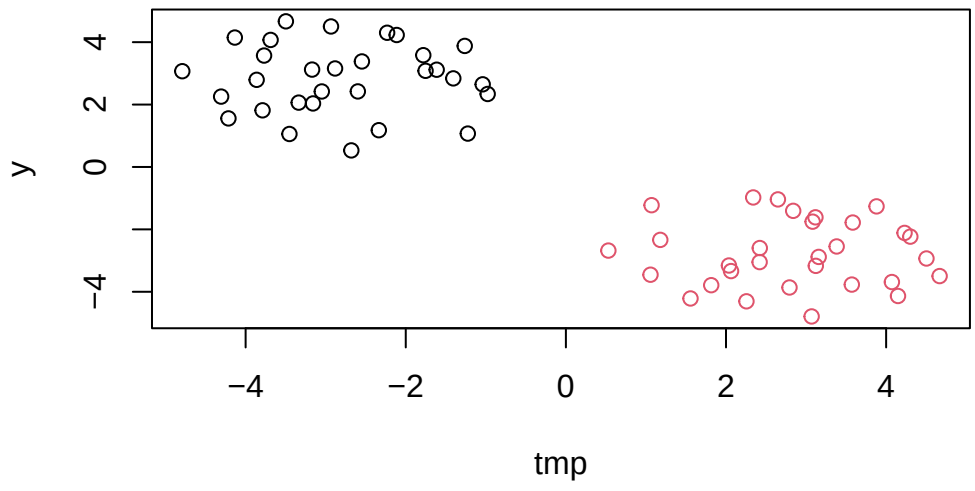
To get our main cluster assignment (membership vector), we need to "cut" the tree at the big goalposts.

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

```
table(grps)
```

```
grps
 1  2
30 30
```

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



Here we see a model more explicitly defining the grouping pattern seen in the dendogram between the two groups seen connected between that large cross bar with a large height in the previous figure These clusters did not have to be defined by a set value like k means, but by bottom up processes and slowly connecting points from small to larger and larger distances, revealing this natural groupoing, where the two main groups appeared, colored in black and red above. Hierarchical Clustering is distinct in that the dendogram (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 bionformatics.

Here we will analyze some data from the UK on food consumption.

Data input

```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
```

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

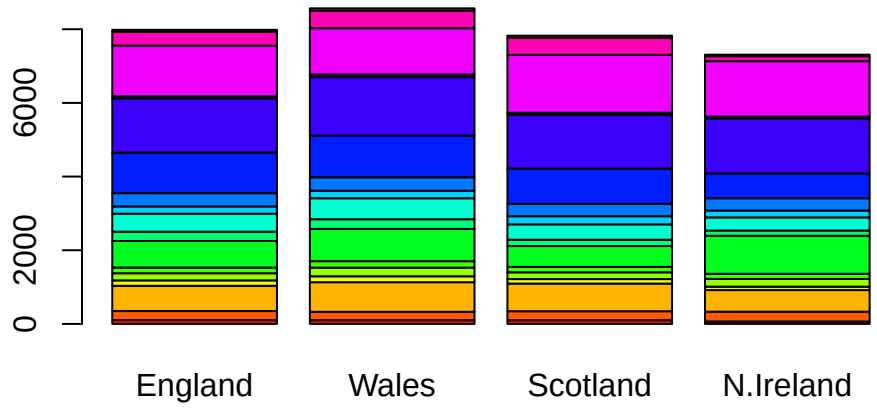
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars  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

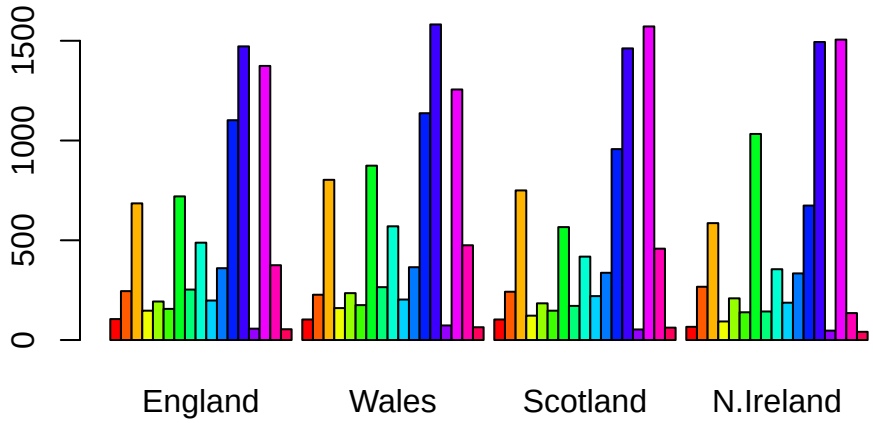
```
Fish               147   160       122        93
Fats_and_oils      193   235       184       209
Sugars             156   175       147       139
```

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



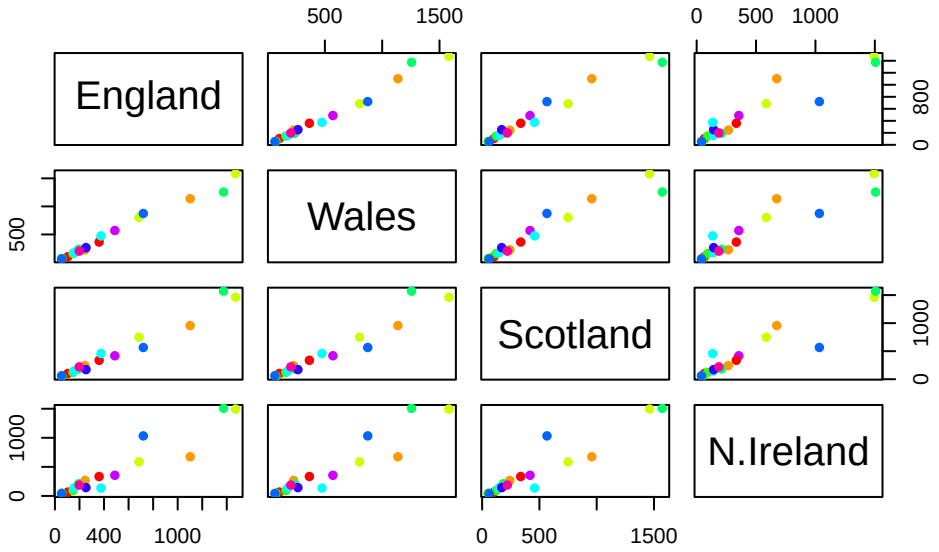
This figure above represents the colors of the rainbow corresponding to the rows seen in the previous code for the differnt food items consumed by these four countries seen on the horizontal. In this case, the area companied by each of these colors is a direct indicator of the relative amount of consumption of that food item for that country, such as the very large dark blue area indicating greater consumption of that food item compared to the orange with smaller area. However, this interpretation of areas for the different colors can be difficult to extrapolate more exact quantitative differences between different items and between different countries given the subjectivity of intepreting relative differences between the areas of the colors seen above.

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



This figure above gives a better approach to looking at differences in consumption of each of these food items were differences can be more easily extrapolated on a more exact quantitative analysis. Similar to before, the rows representing the different food items are colored with colors of the rainbow, but rather than using areas to depict relative consumption per country for each of these foods, the height of the colors used to depict relative consumption, again allowing for more exact extrapolation of differences in a food item across the 4 countries or difference between food items within a country or between countries. This figure helps show the greatest of these food items being in the purple and pink colors pretty consistently between countries.
One conventional plot that can be useful is called the "paris" plot.

```
pairs(x, col=rainbow(10), pch=16)
```



This plot allows for slightly better analysis than the previous two in allowing us to see more direct variation across these food items for each of the countries. For instance, the top row, from left to right indicates Wales plotted against England, followed by Scotland plotted against England, followed by N, and lastly Ireland plotted against England. To further make this point, the bottom row indicates England plotted against n. Ireland, then Wales plotted against N. Ireland, then Scotland plotted against n. Ireland from left to right. Similar to before, we see the different food items plotted with the colors of the rainbow. In each of these graphs we can see the variation between y and x with each of these variables being a country, where the differences in the x and y inputs across each of these graphs for a particular food item, represent the differences between their consumption for those two countries serving as x and y. Further, the mostly linear variation you can see in both of these plots give a visual representation of the variation between y and x, where a linear regression model could be made (although a residual plot may need to be looked at first to confirm appropriateness of linear model) to map these points to a line of best fit that could directly explain the variation between these two countries for each plot. Looking more specifically at these figures, the large cluster circles close to the origin seem pretty consistent across all the plots, where one could assume these food items do not have too much variation across each of the countries. However, for example the dark blue dot towards the right seems to have very different positions of x and y across the plots, indicating variation in this particular food items across the different countries.To better extrapolate the variation between each of the two countries across these plots, we can create a line of best fit represented by PC1 and look at the residuals from that line, essentially represented by PC2 seen below. ### 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 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 object of our results with 5 attributes/components.

```
attributes(pca)
```

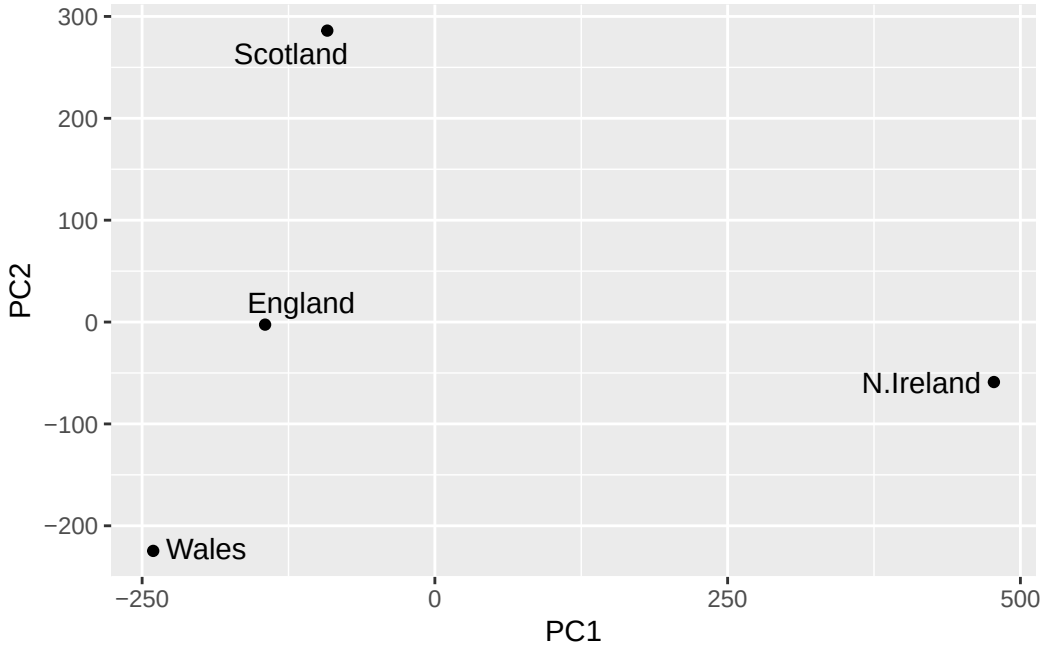
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$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 -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)
ggplot(pca$x) +
  aes(PC1, PC2, label=rownames(pca$x)) +
  geom_point() +
  geom_text_repel()
```



In this figure above, we see the results of plotting PC1 vs PC2 for these different countries. In this case, PC1 attempts to map a line of best fit to all the points of data for the different food items for the 4 countries, accounting for most of the variation, while PC2 helps account for variations not accounted for by that line, accounting for less variation among the data. Thus, looking at this figure, it appears that Northern Ireland is most dissimilar compared to the other 3 countries given its large horizontal distance away from them across the PC1 horizontal, which accounts for the most variation. There appears to be more similarity across the 3 other countries which are at a similar point along the PC1, but different points along the PC2, but such differences in PC2 indicate less dissimilarity compared to those for PC1, because PC1 accounts for more variation than PC2. Thus, although England appears closer to Wales than Scotland, and these variations do indicate some dissimilarity they do not do so to the extent as if they were along PC1, like N. Ireland.

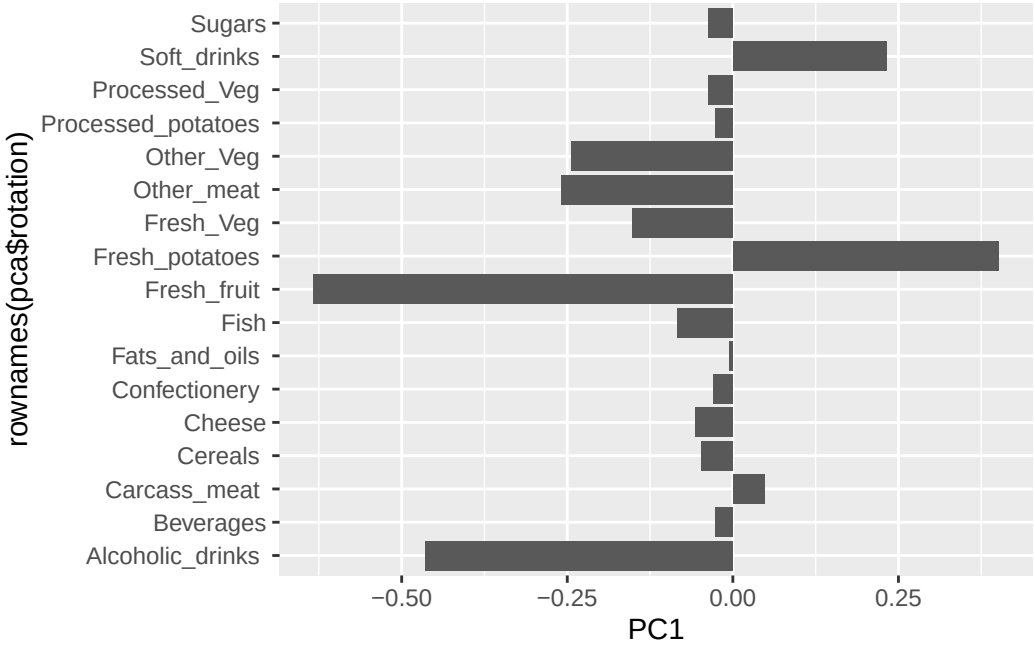
The second major result is conatined in the `pca$rotation` object of component. Let's plot this to see what PCA is picking up … i.e how much each of these elements contributes to PC1.

```
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()
```



These particular figure depicts relatively how much each food item contributes to the PC1 eigen-vector. The values closer to 0 have less contribution to PC1, while the values farther from 0 in either direction, positive or negative, have greater contribution to PC1. Thus, in this case, it appears that fresh fruit and alcohol have the highest contribution to PC1, while fats and oils appear to have the least contribution to PC1.

How each of these variables contribute to the PCs. Write notes about what each of these figures show. ”