Class 7: Machine learning 1

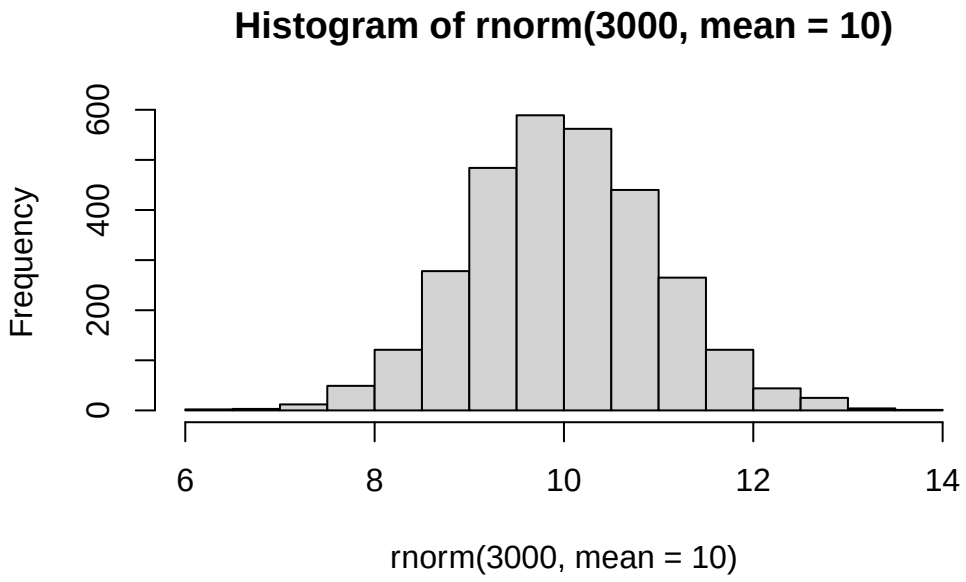
Vivian (PID:A18497911)

Background

Today we will begin our exploration of important machine learning methods with a focus on **clustering** and **dimensinallity reducation**

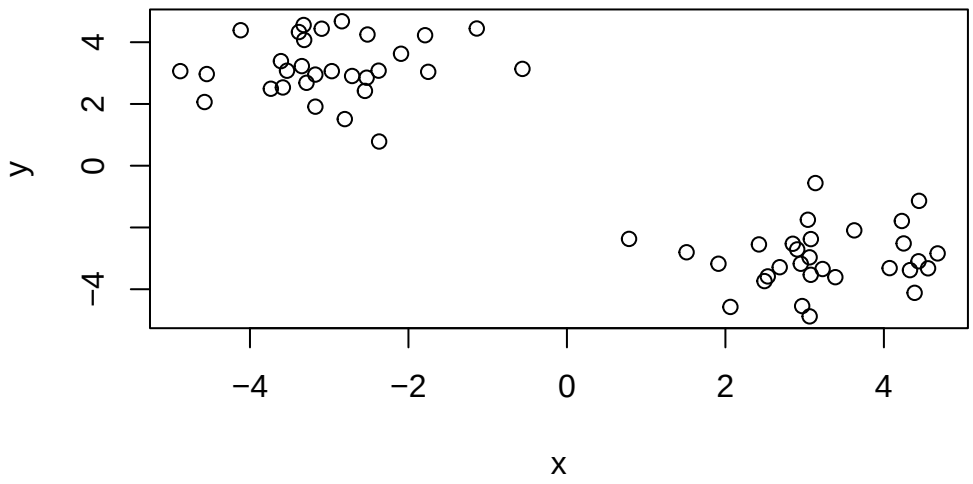
To start testing these methods let's, make up some sample data to cluster where we knoe what the answer should be.

```
hist(rnorm(3000, mean=10))
```



Q. Can you generate 30 numbers centered at +3 and 30 numbers at -3 taken at random from a normal distribution?

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

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()`, lets try it out:

```r
k <- kmeans(x, centers = 2)
k
```

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

Cluster means:
          x         y
1  3.205232 -2.989080
2 -2.989080  3.205232

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 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] 53.45464 53.45464
 (between_SS / total_SS =  91.5 %)

Available components:

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

Q. What componet of your kmeans rusults object has the cluster centers?

```
k$centers
```

```
          x         y
1  3.205232 -2.989080
2 -2.989080  3.205232
```

Q. What componet of your kmeans rusults object has the cluster size (i.e. how many points are in each cluster)?

```
k$size
```

```
[1] 30 30
```

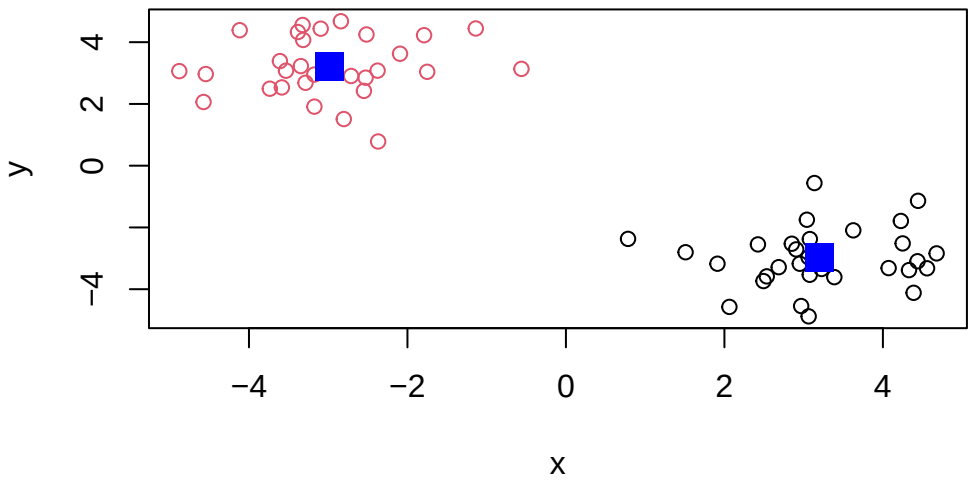
Q. What componet of your kmeans rusults object has the cluster membership vector (i.e. the main clustering result: which points are in which cluster)?

```
k$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 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 the results of clustering (i.e., or data colored bt the clusterting result) along with the cluster centers

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



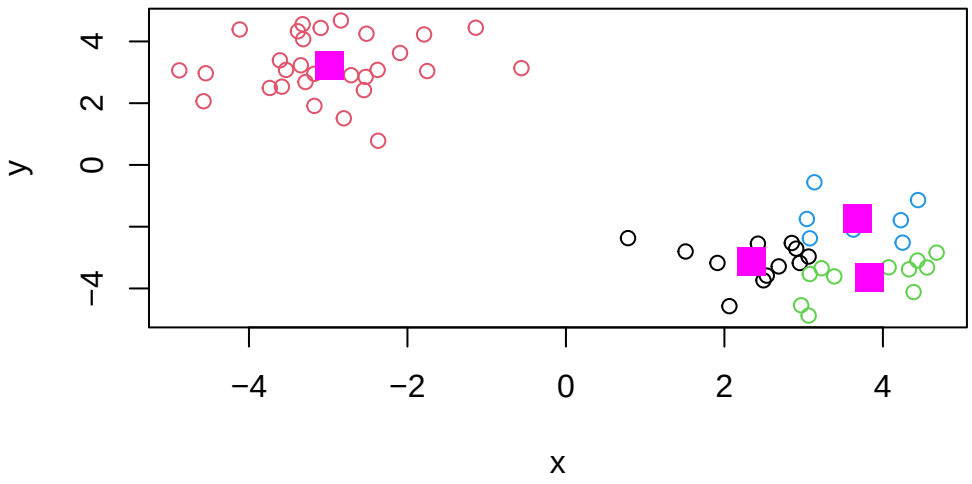
Q. Can you run **kmeans** again and cluster **x** into 4 clusters and plot the results just like we did above with coloring by cluster and the cluster centers shown in blue?

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

```
K-means clustering with 4 clusters of sizes 12, 30, 11, 7

Cluster means:
          x         y
1  2.347812 -3.121633
2 -2.989080  3.205232
3  3.834839 -3.635003
4  3.685711 -1.746825

Clustering vector:
 [1] 4 1 1 1 3 4 1 1 4 1 3 3 3 4 1 1 3 4 3 3 1 3 1 1 4 3 1 4 3 3 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]  9.305309 53.454640  8.584569  5.163645
 (between_SS / total_SS =  93.9 %)
```

```
Available components:

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

```
plot(x, col=k4$cluster)
points(k4$centers, col="magenta", pch=15, cex=2)
```



Key-point kmeans will always return the clustering that we ask for (this the "K" or "centers" in K-means)

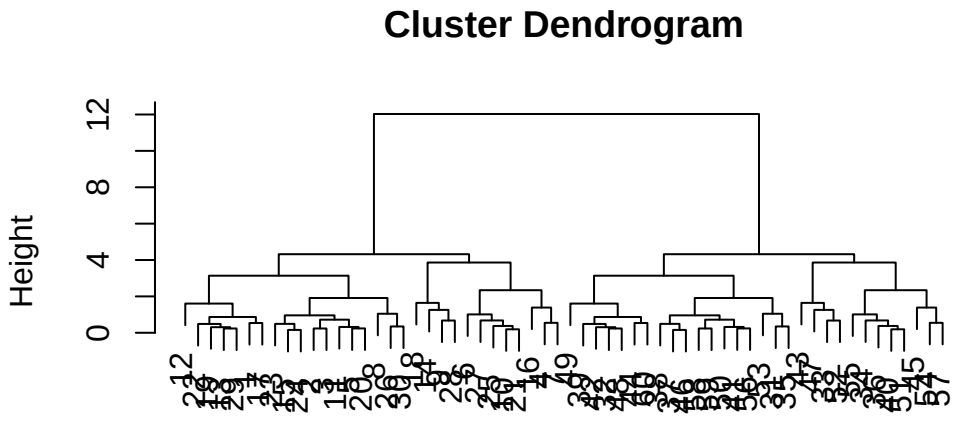
```
k$tot.withinss
```

```
[1] 106.9093
```

Hierarchial clustering

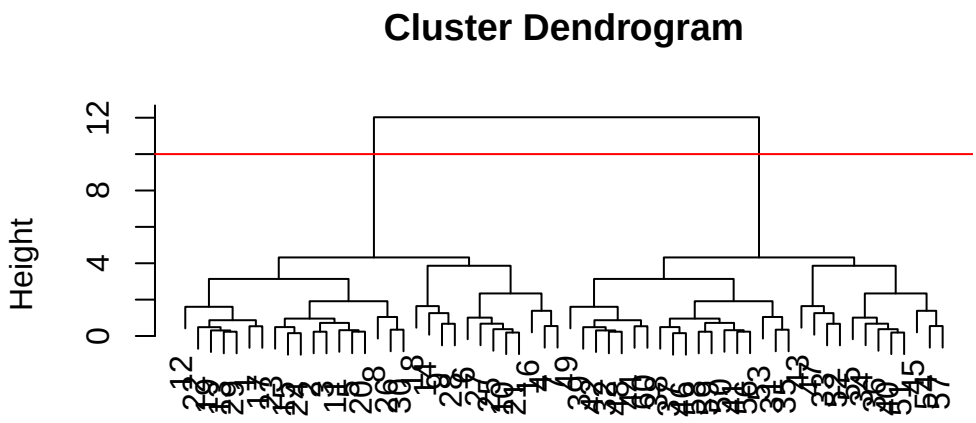
The main function for Hierarchical clustering in base R is called `hclust()`. One of the main differnces with respect to the `kmeans()` function is that you can not just pass your input data directly to `hclust()`- it needs a "distance matrix" as input. We can get this from lot's of places including the `dist()` function.

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



We can "cut" the dendrogram or "tree" at a given height to yield our "clusters". For this we use the function `cutree`

```
plot(hc)
abline(h=10, col="red")
```



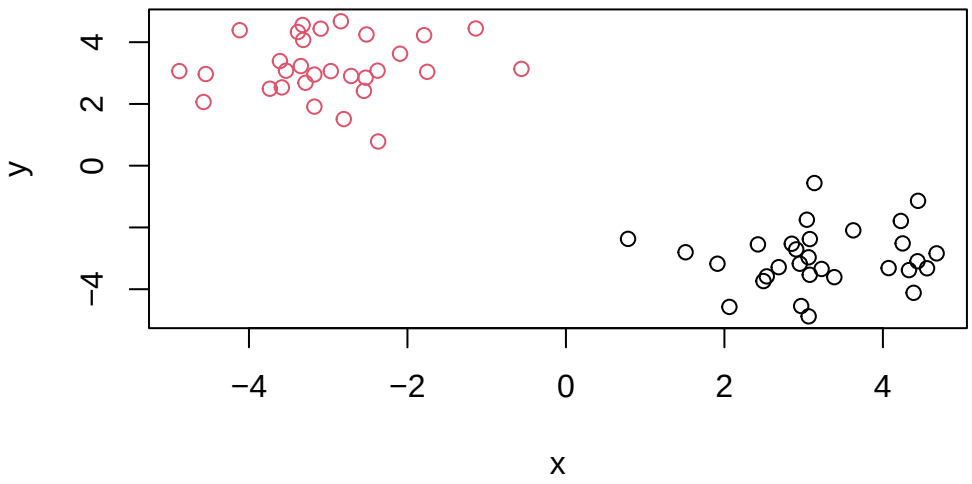

```
grps <- cutree(hc, h=10)
```

```
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
```

Q. Plot our data `x` colored by the clusterinv result from `hclust()` and `cutree`?

```
grps <- cutree (hc, h=10)
plot(x, col=grps)
```



Principal component Analysis (PCA)

PCA is a popular dimenionality reduction techniquw that iss widely used in bioinformatics.

##PCA of UK food

Read data on 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
```

It looks like the row names are not set properly. We can fix this

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

A better way to do this is fix the row names assignment at import time:

```
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
```

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  4
```

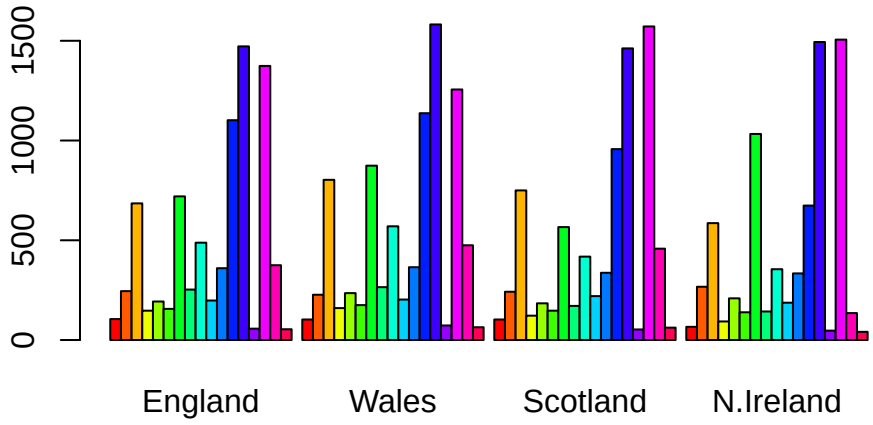
There are 17 rows and 4 columns.Used dim() function to find rows and columns after using row.names() to remove first column from being "x"

. 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?

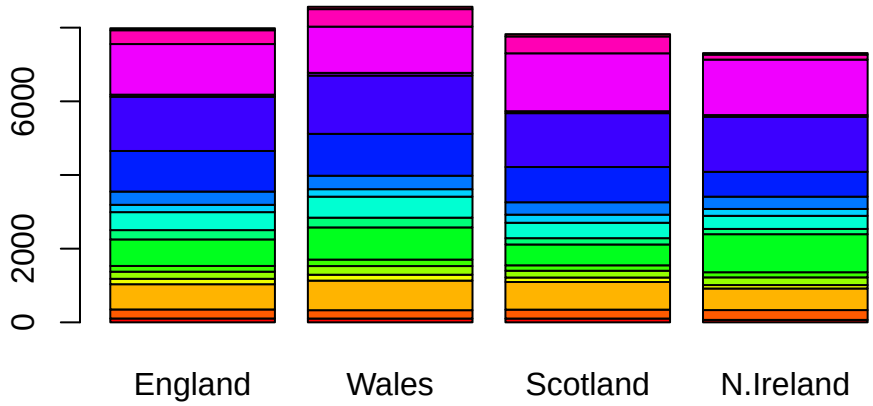
row.names() function is better in that it fixes the designated column, while the x <- x[,-1] function will keep taking away a column from the data

. Q3: Changing what optional argument in the above barplot() function results in the following plot?

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




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

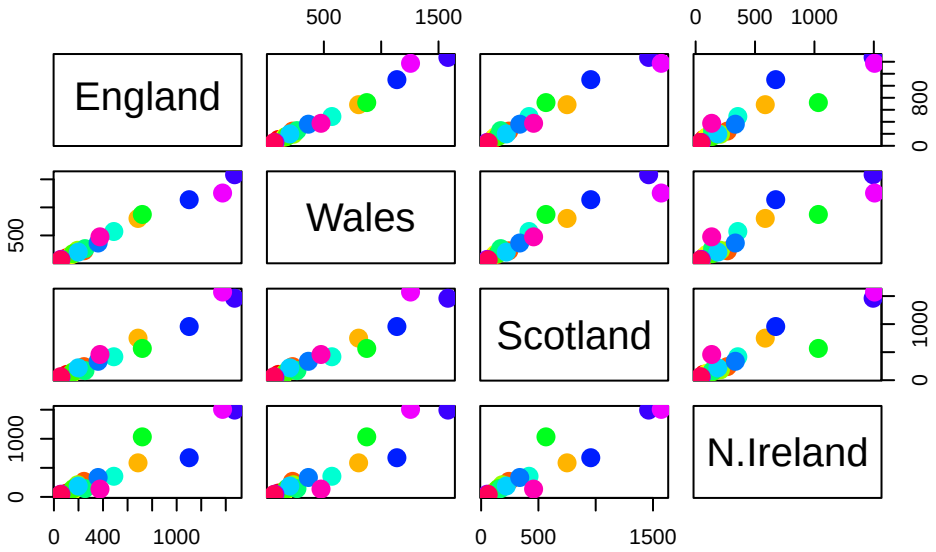


pairs plot and heatmaps

. Q4. is missing

.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?

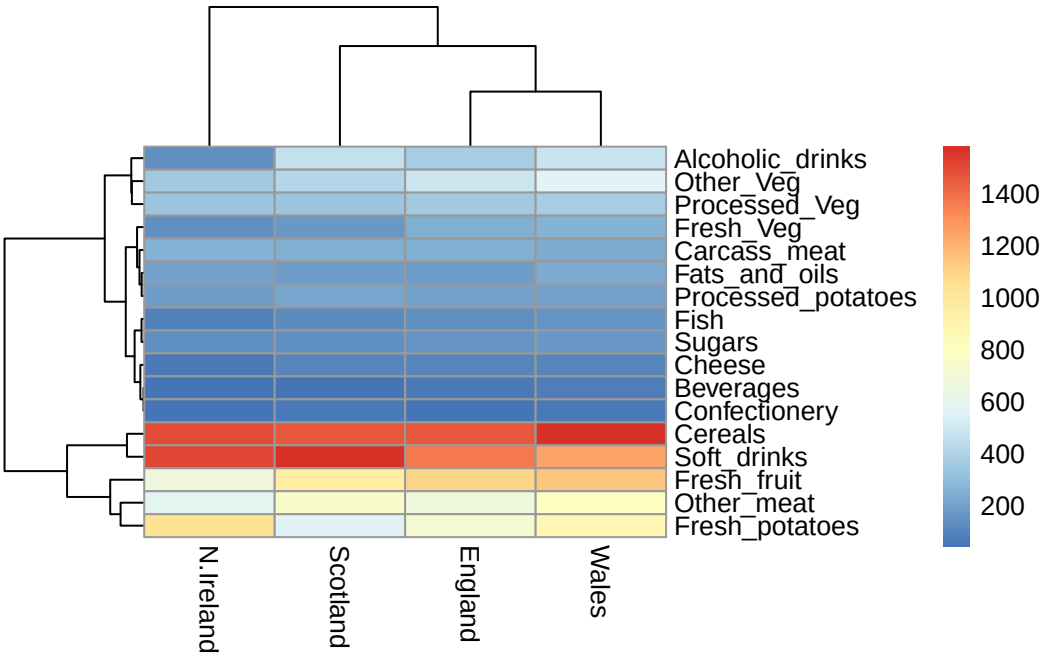
```
pairs(x, col=rainbow(17), pch=16, cex=2)
```



Heatmap

We can install the **pheatmap** pacage with the `install.package()` command that we used previously. Remember that we always run this in the console not a code chunk in our quarto doccument.

```
library(pheatmap)
pheatmap(x)
```



Of all of these plots really only `pairs()` plot was useful. tyhis however took a bit off work to interpret and well of scale when i am looking at much bigger datasets.

PCA the rescue

The main function in 'base R'bfor 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
```

Q. How much varance is captured in the first PC?

67.4%

Q. How many PCs do I need to capture at least 90% of the total varance in the dataset?

Two PCs capture 96.5% of total variance>

> Q. Plot our main PCA result. Folks can call thsis different things depending on their field of study e.g. “PC plot”, “ordienitaion plot” “Score plot”, “PC1 vs. P2”…

```
attributes(pca)
```

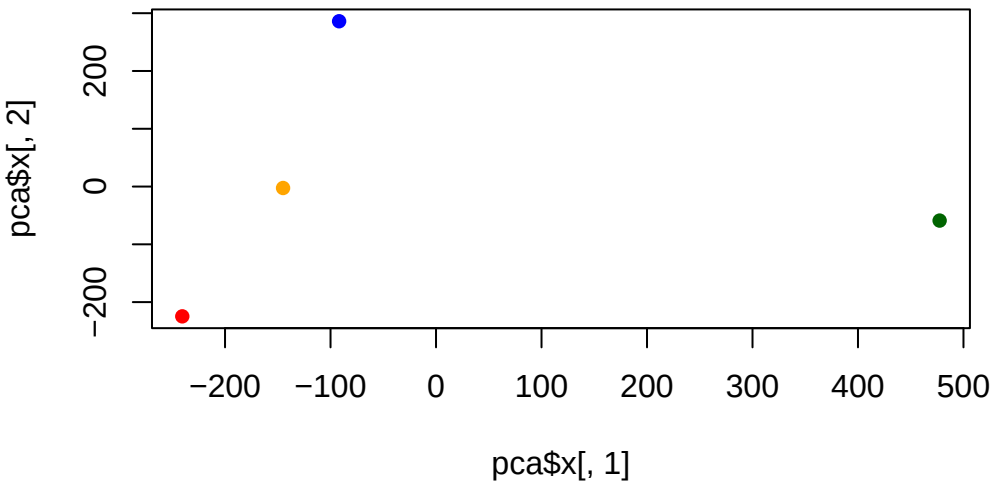
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

To generate oir PCA score plot we want the `pca$x` component of the resulting object

```
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
```

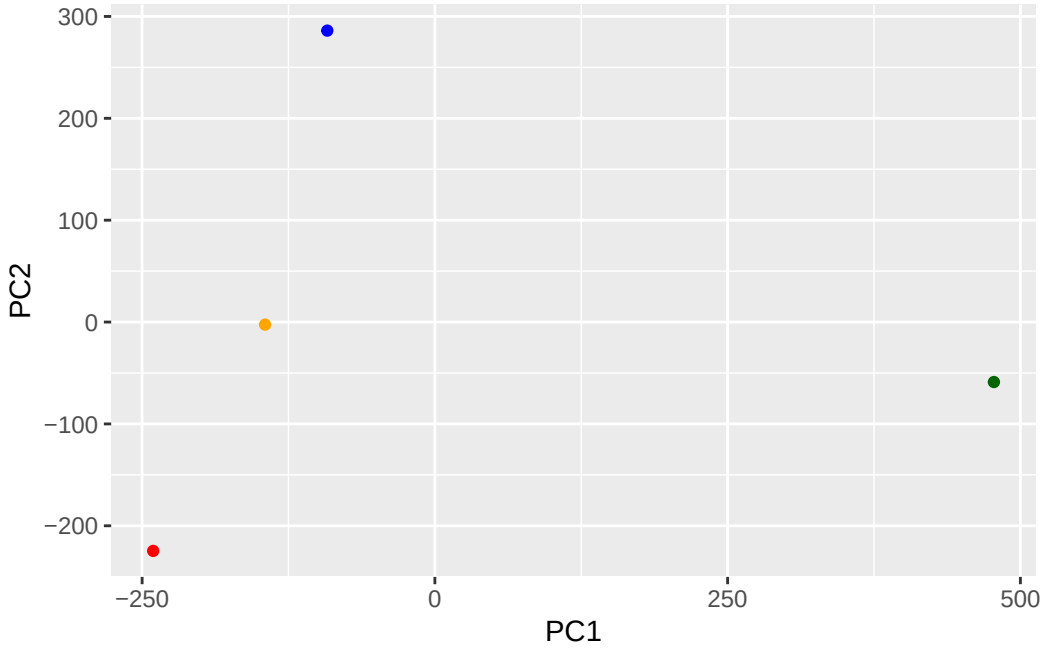
```
my_cols <- c("orange", "red","blue", "darkgreen")
plot(pca$x[,1], pca$x[,2], col=my_cols, pch=16)
```




```
library(ggplot2)
```

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

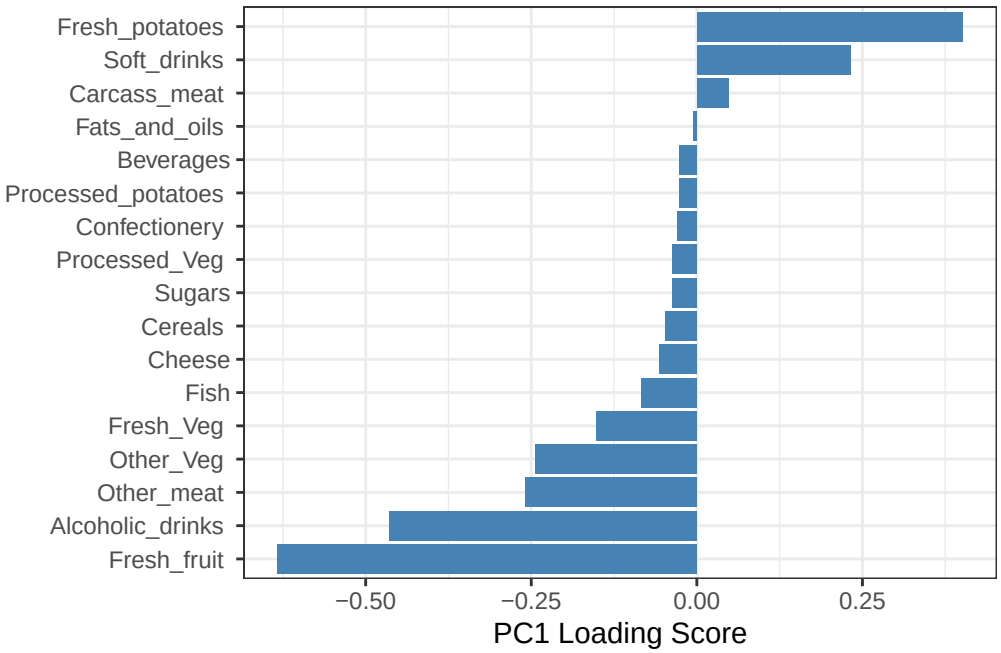
```
ggplot(pca$x) + aes(PC1, PC2) + geom_point(col=my_cols)
```



Digging deeper (variable loading)

How do the original variables (i.e. the 17 different foods) contribute to our new PCs

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


Fresh_potatoes
Soft_drinks
Carcass_meat
Fats_and_oils
Beverages
Processed_potatoes
Confectionery
Processed_Veg
Sugars
Cereals
Cheese
Fish
Fresh_Veg
Other_Veg
Other_meat
Alcoholic_drinks
Fresh_fruit
−0.50
−0.25
0.00
0.25
PC1 Loading Score