Machine Learning 1

Andrea Sama (A59010582)

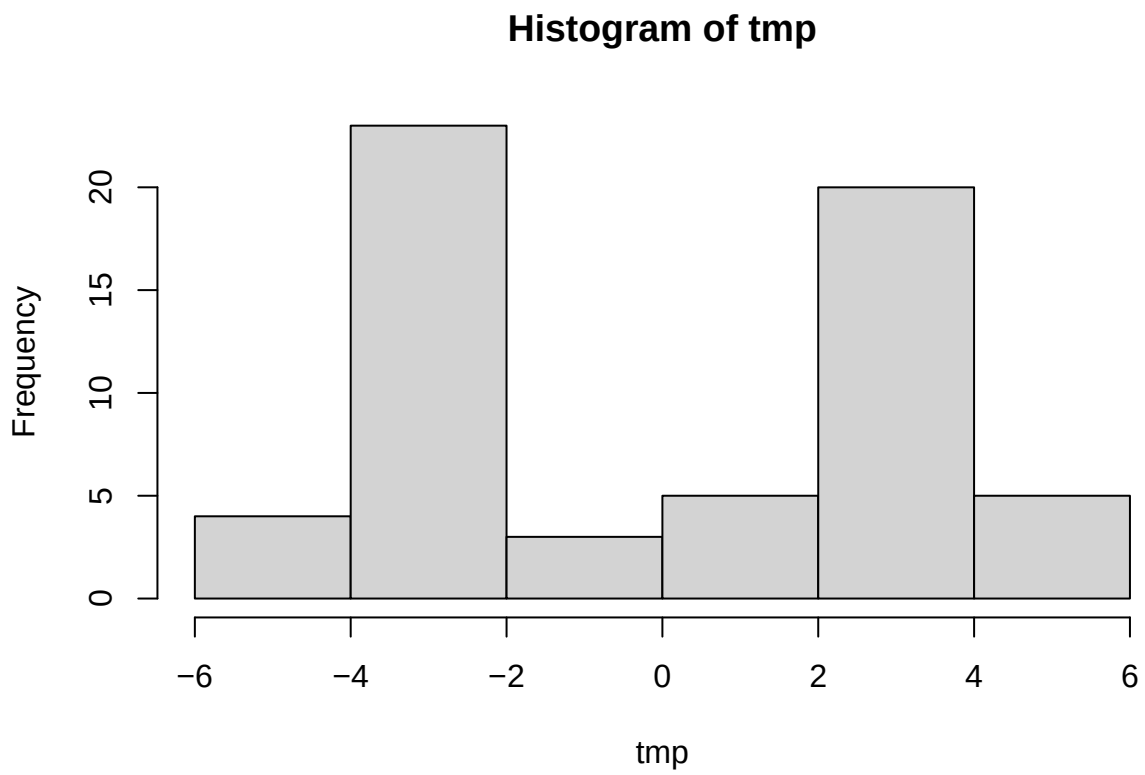
10/22/2021

#Clustering methods: finding groupings in our data

Kmeans clustering in R is done with the `kmeans()` function.

Here we makeup some data to test and learn with.

```
tmp <- c(rnorm(30, 3), rnorm(30, -3))
data <- cbind(tmp, rev(tmp))
#making up a data set that reverses the order of the original
hist(tmp)
```

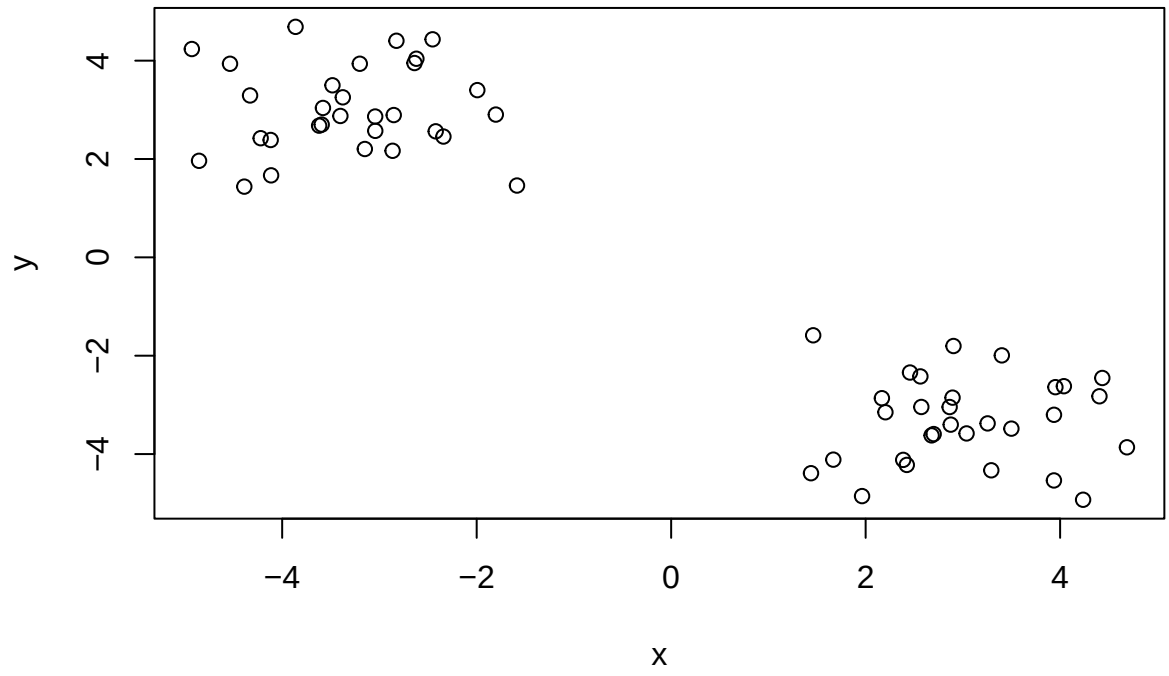



```
tmp <- c(rnorm(30, 3), rnorm(30, -3))
data <- cbind(x=tmp, y=rev(tmp))
#making up a data set that reverses the order of the original
data
```

```
##              x         y
##  [1,]  2.562858 -2.421067
##  [2,]  3.037968 -3.580785
##  [3,]  2.903236 -1.802971
##  [4,]  4.687219 -3.862460
##  [5,]  3.254189 -3.376585
##  [6,]  1.667573 -4.113747
##  [7,]  3.498929 -3.482879
##  [8,]  1.438199 -4.389516
##  [9,]  2.456289 -2.341891
## [10,]  4.433795 -2.453153
## [11,]  2.678093 -3.620107
## [12,]  3.292063 -4.330367
## [13,]  2.167188 -2.864460
## [14,]  2.203451 -3.150035
## [15,]  2.893668 -2.852623
## [16,]  1.963233 -4.854931
## [17,]  2.863779 -3.043702
## [18,]  2.874995 -3.401343
## [19,]  2.423159 -4.221084
## [20,]  4.038663 -2.619984
## [21,]  3.400377 -1.993160
## [22,]  3.951366 -2.639473
## [23,]  4.404283 -2.825230
## [24,]  2.386334 -4.118203
## [25,]  3.936858 -3.201660
## [26,]  3.936232 -4.536446
## [27,]  2.699409 -3.593646
## [28,]  2.572634 -3.042664
## [29,]  1.460530 -1.585237
## [30,]  4.236196 -4.928962
## [31,] -4.928962  4.236196
## [32,] -1.585237  1.460530
## [33,] -3.042664  2.572634
## [34,] -3.593646  2.699409
## [35,] -4.536446  3.936232
## [36,] -3.201660  3.936858
## [37,] -4.118203  2.386334
## [38,] -2.825230  4.404283
## [39,] -2.639473  3.951366
## [40,] -1.993160  3.400377
## [41,] -2.619984  4.038663
## [42,] -4.221084  2.423159
## [43,] -3.401343  2.874995
## [44,] -3.043702  2.863779
## [45,] -4.854931  1.963233
## [46,] -2.852623  2.893668
## [47,] -3.150035  2.203451
## [48,] -2.864460  2.167188
## [49,] -4.330367  3.292063
## [50,] -3.620107  2.678093
## [51,] -2.453153  4.433795
## [52,] -2.341891  2.456289
## [53,] -4.389516  1.438199
```

```
## [54,] -3.482879  3.498929
## [55,] -4.113747  1.667573
## [56,] -3.376585  3.254189
## [57,] -3.862460  4.687219
## [58,] -1.802971  2.903236
## [59,] -3.580785  3.037968
## [60,] -2.421067  2.562858
```

```
plot(data)
```



Run `kmeans()` set k (centers) to 2 (ie the number of clusters that we want) nstart 20. The thing with Kmeans is you have to tell it how many clusters you want.

```
km<-kmeans(data, centers=2, nstart=20)
km
```

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -3.308279  3.010759
## 2  3.010759 -3.308279
##
## 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 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 1 1
##
## Within cluster sum of squares by cluster:
## [1] 45.94827 45.94827
##  (between_SS / total_SS =  92.9 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q: How many points are in each cluster?

```
km$size
```

```
## [1] 30 30
```

Q. What 'component' of your result details cluster assignment/membership?

```
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 1 1
```

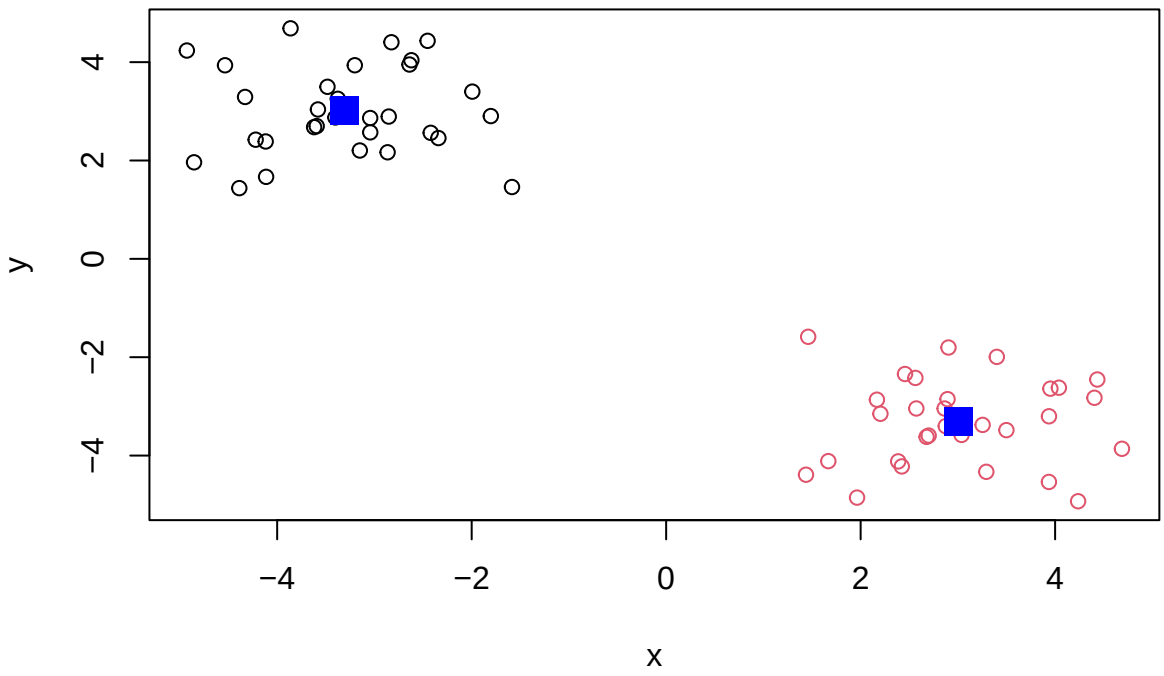
Q. What 'component' of your result object details cluster center?

```
km$centers
```

```
##           x         y
## 1 -3.308279  3.010759
## 2  3.010759 -3.308279
```

Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points.

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



#hclust We will use the `hclust()` function on the same data as before and see how this method works.

```
#hclust(data)
```

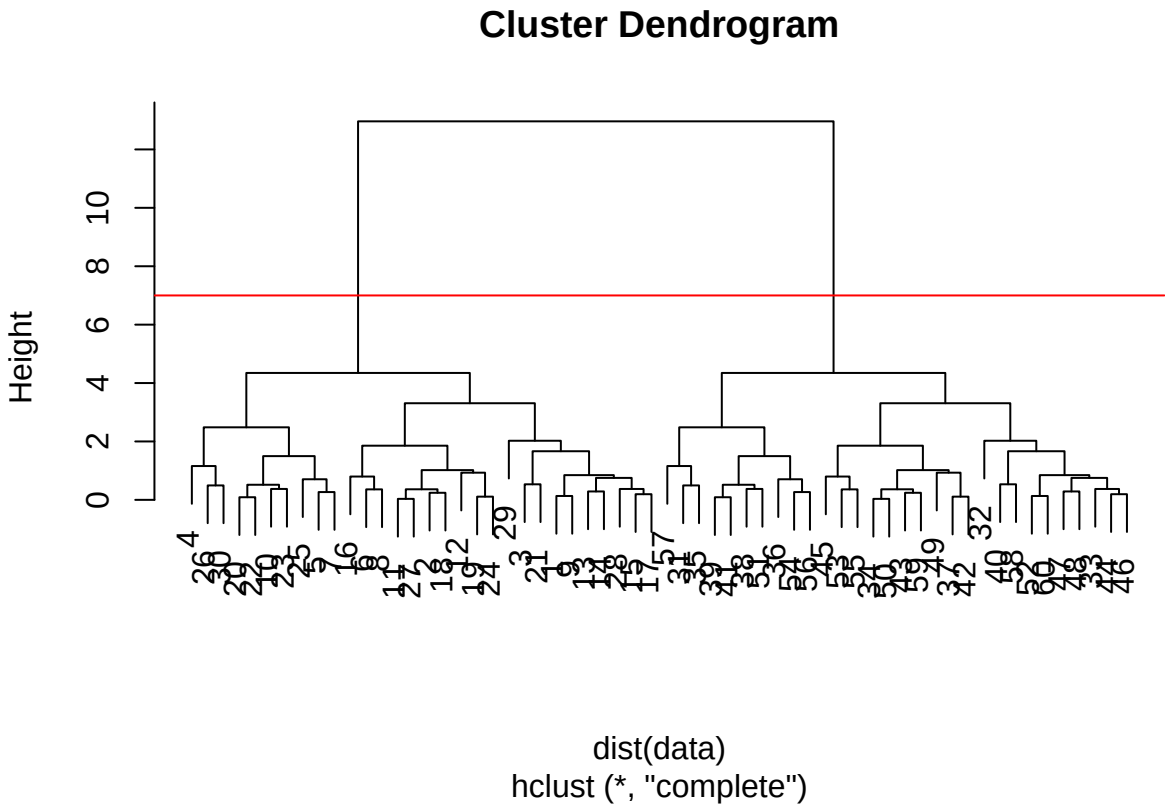
gives an error, needs a distance.

```
hc<-hclust(dist(data))
hc
```

```
##
## Call:
## hclust(d = dist(data))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

doesnt give a lot of information, but does provide a plot method.

```
plot(hc)
abline(h=7, col="red")
```



TO find our membership vector we need to "cut" the tree and for this we use the `cutree()` function and tell it the height to cut at.

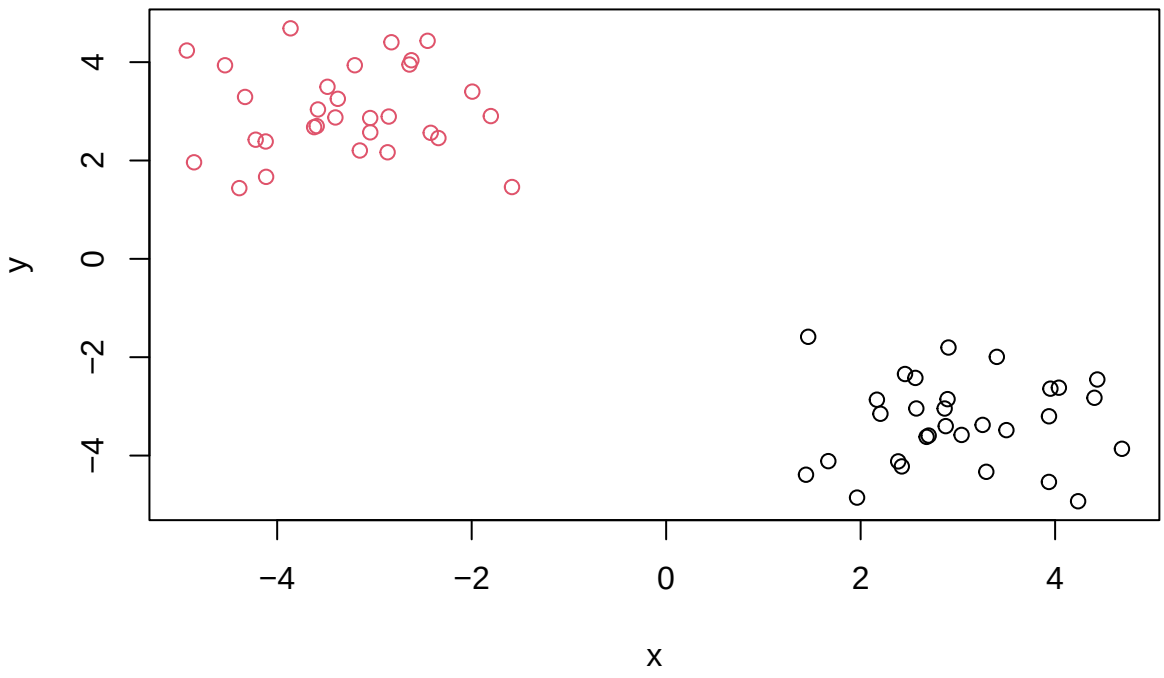
```
cutree(hc, h=7)
```

```
##  [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
```

If you dont know what the height is you can use cut at k=? use `cutree()` and state the number of k clusters that we want.

```
grps<- cutree(hc, k=2)
```

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



kmeans(x, centers=?) hclust(dist(x))

#Principal Component Analysis (PCA) PCA projects the features onto the principal components The motivation is to reduce the features dimensionality while only losing a small amount of information

New low dimension axis or surfaces closest to the observations

The first PCA:Line of best fit that lies closest to data that describes the spread of the data

#PCA of UK diets

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

How many rows and columns?

```
dim(x)
```

```
## [1] 17  5
```

```
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
```

The x is considered a column but we want it to be read as a row One option

```
x[,-1]
```

```
##    England Wales Scotland N.Ireland
## 1      105   103      103        66
## 2      245   227      242       267
## 3      685   803      750       586
## 4      147   160      122        93
## 5      193   235      184       209
## 6      156   175      147       139
## 7      720   874      566      1033
## 8      253   265      171       143
## 9      488   570      418       355
## 10     198   203      220       187
## 11     360   365      337       334
## 12    1102  1137      957       674
## 13    1472  1582     1462      1494
## 14      57    73       53        47
## 15    1374  1256     1572      1506
## 16     375   475      458       135
## 17      54    64       62        41
```

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

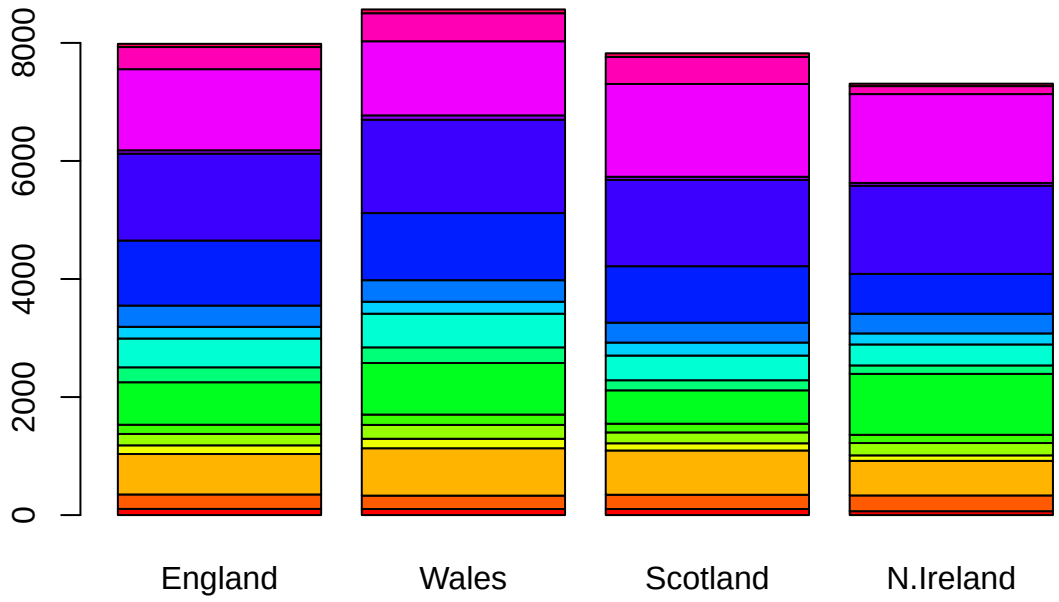
```
##                    England Wales Scotland N.Ireland
## Cheese                 105   103      103        66
## Carcass_meat           245   227      242       267
## Other_meat             685   803      750       586
## Fish                   147   160      122        93
## Fats_and_oils     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

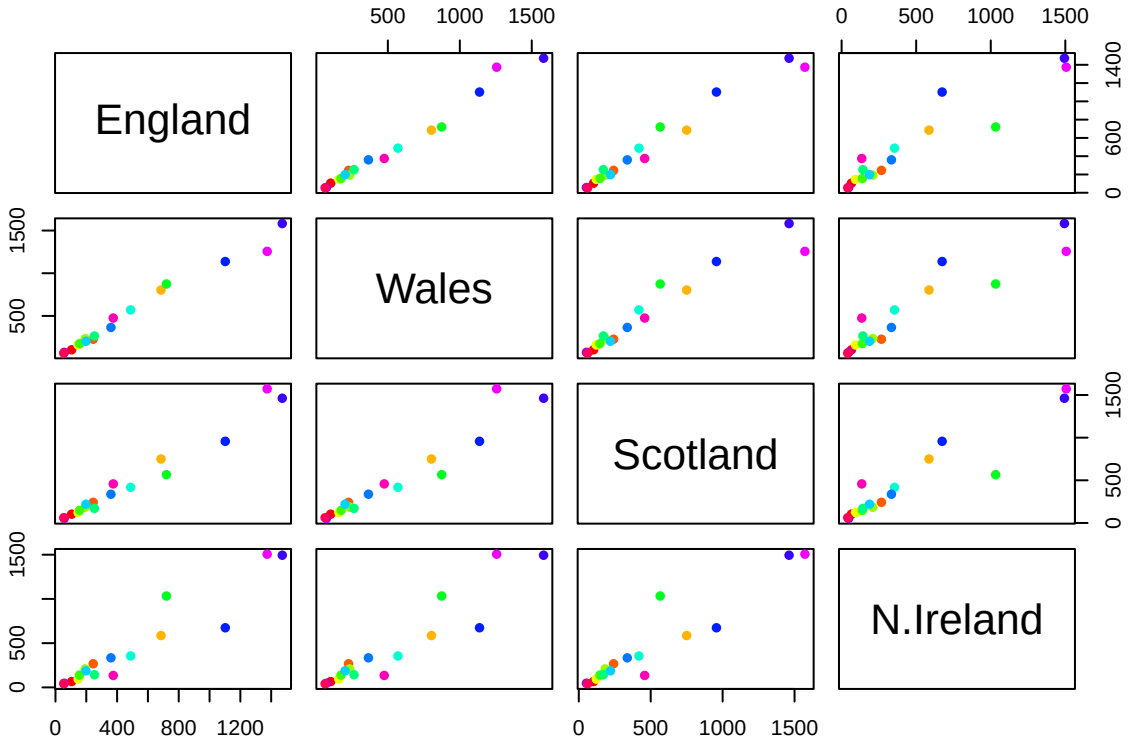
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)
x
```

```
##                     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
```

```
barplot(as.matrix(x), col=rainbow(17))
```




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



##PCA to the rescue!

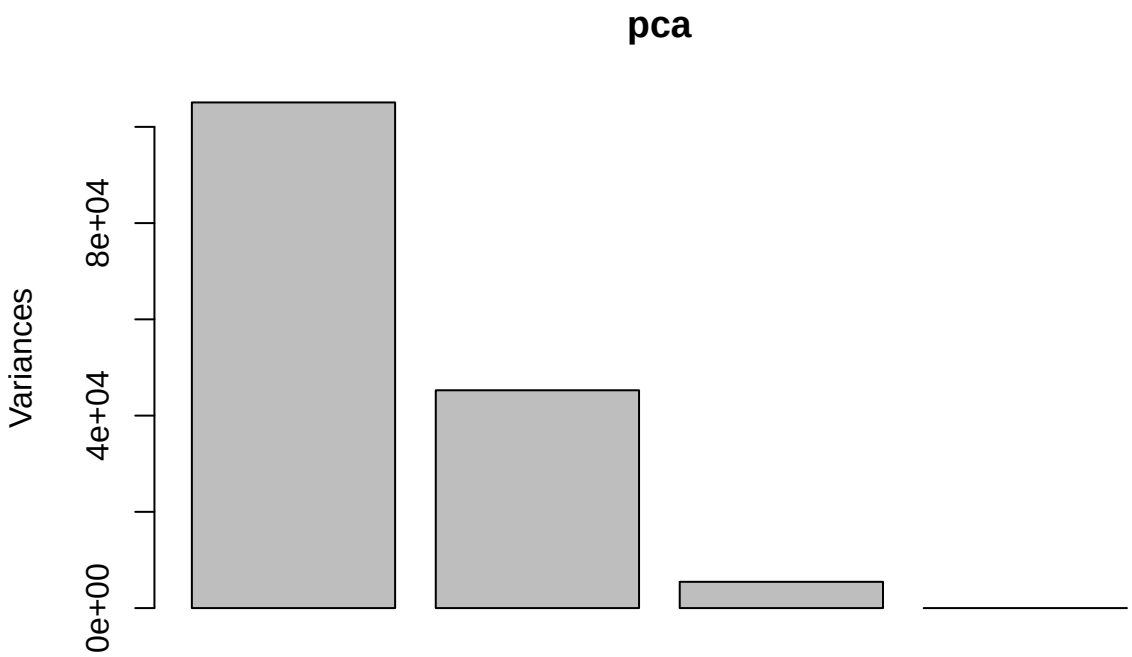
Here we will use the base R function for PCA, which is called `precomp()`. This function wants the transpose of our data.

```
pca<-prcomp( t(x) )
summary(pca)
```

```
## Importance of components:
##                             PC1      PC2      PC3        PC4
## Standard deviation     324.1502 212.7478 73.87622 4.189e-14
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

PC1 captures 67%, PC2 captures 29%, Cumulative proportion is the addition of the porportion of variances for the PCs

```
plot(pca)
```



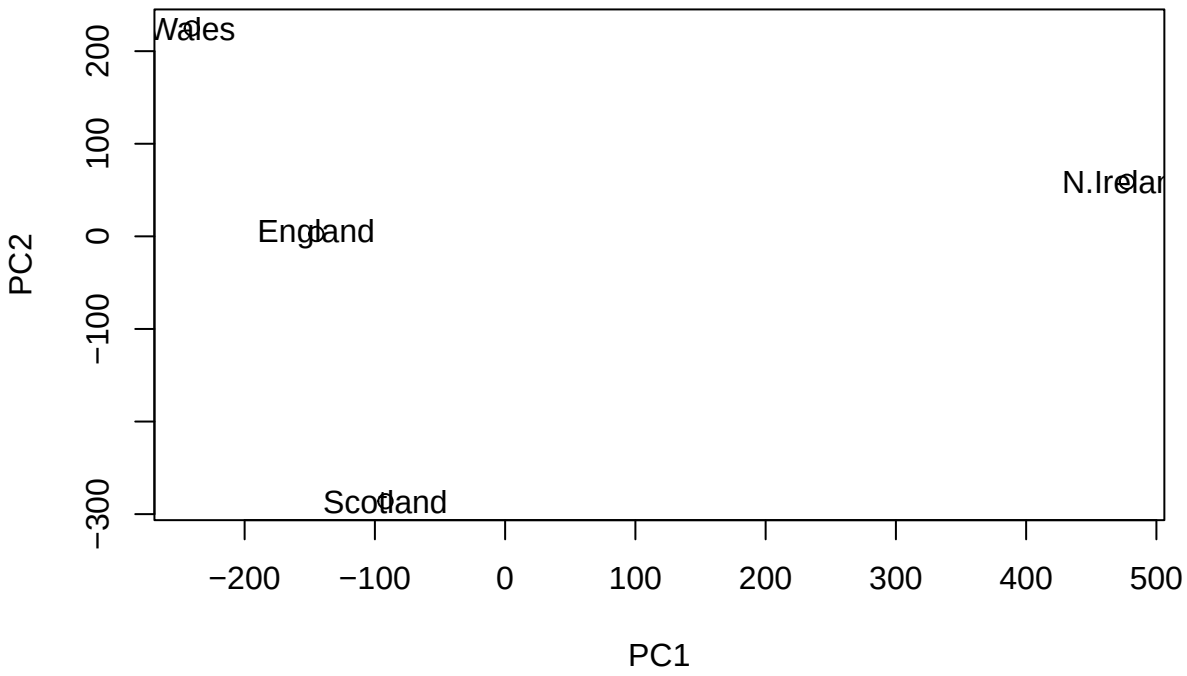
We want score plot (aka PCA plot). Basically a plot of PC1 vs PC2

```
attributes(pca)
```

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

We are after the pca$x component

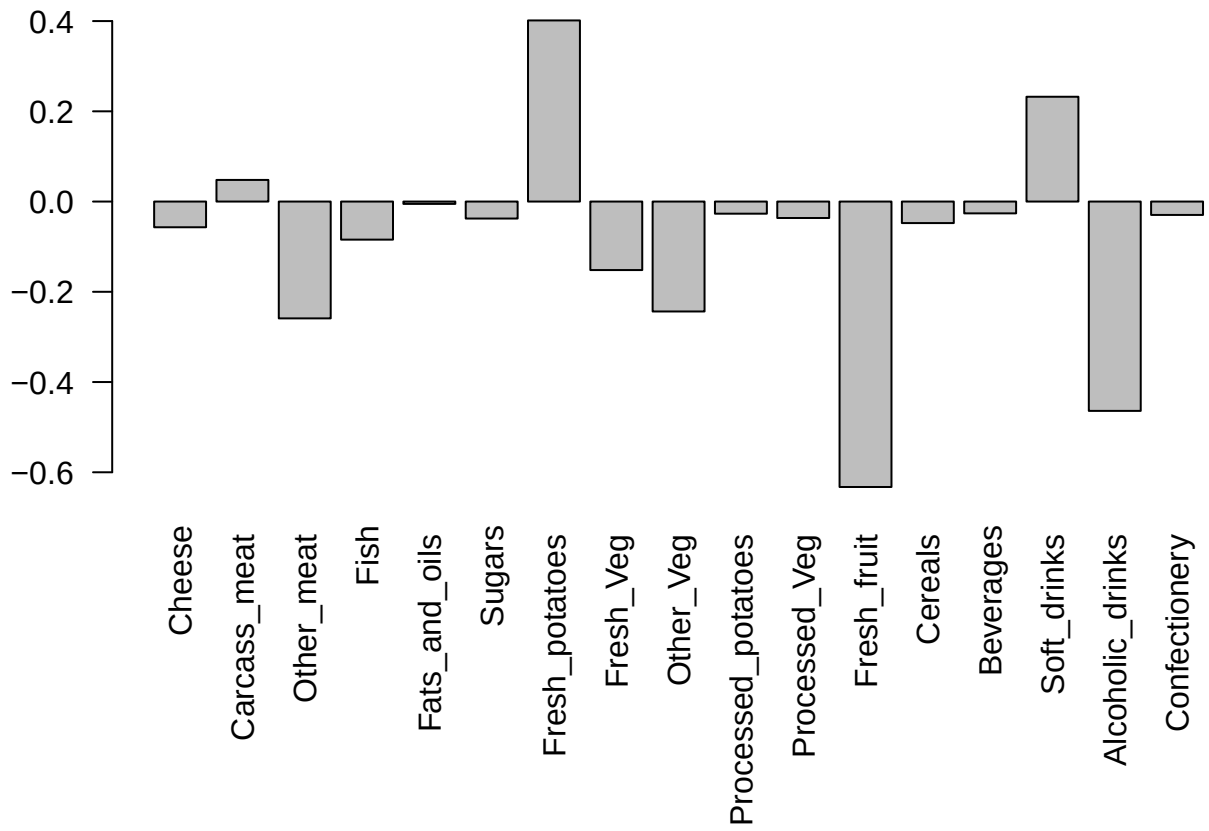
```
plot(pca$x[,1:2])
text(pca$x[,1:2], labels=colnames(x))
```



Northern Ireland is really different than these other three countries. PCA will tell us why they are different, how much each of the different foods contributed to the differences.

We can also examine the PCA "loadings", which tell us how much the original variables contribute to each new PC...

```
par(mar=c(10, 3, 0.35, 0))
barplot(pca$rotation[,1], las=2)
```



One more PCA for today

```
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <- read.csv(url2, row.names=1)
head(rna.data)
```

```
##        wt1 wt2  wt3  wt4 wt5 ko1 ko2 ko3 ko4 ko5
## gene1  439 458  408  429 420  90  88  86  90  93
## gene2  219 200  204  210 187 427 423 434 433 426
## gene3 1006 989 1030 1017 973 252 237 238 226 210
## gene4  783 792  829  856 760 849 856 835 885 894
## gene5  181 249  204  244 225 277 305 272 270 279
## gene6  460 502  491  491 493 612 594 577 618 638
```

```
ncol(rna.data)
```

```
## [1] 10
```

```
colnames(rna.data)
```

```
##  [1] "wt1" "wt2" "wt3" "wt4" "wt5" "ko1" "ko2" "ko3" "ko4" "ko5"
```

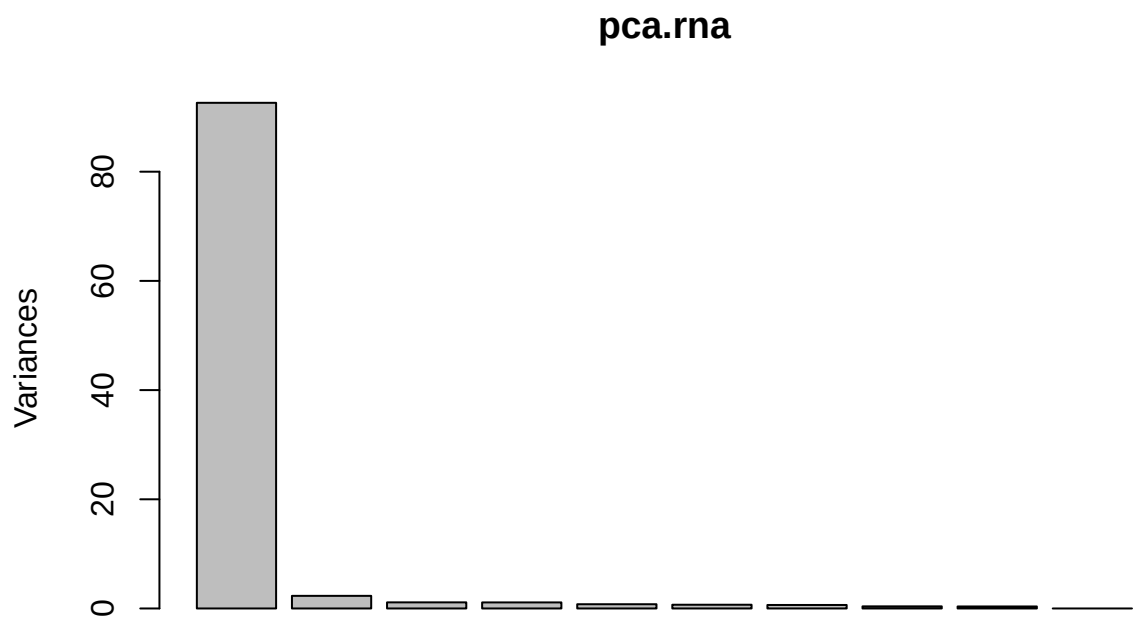
```
nrow(rna.data)
```

```
## [1] 100
```

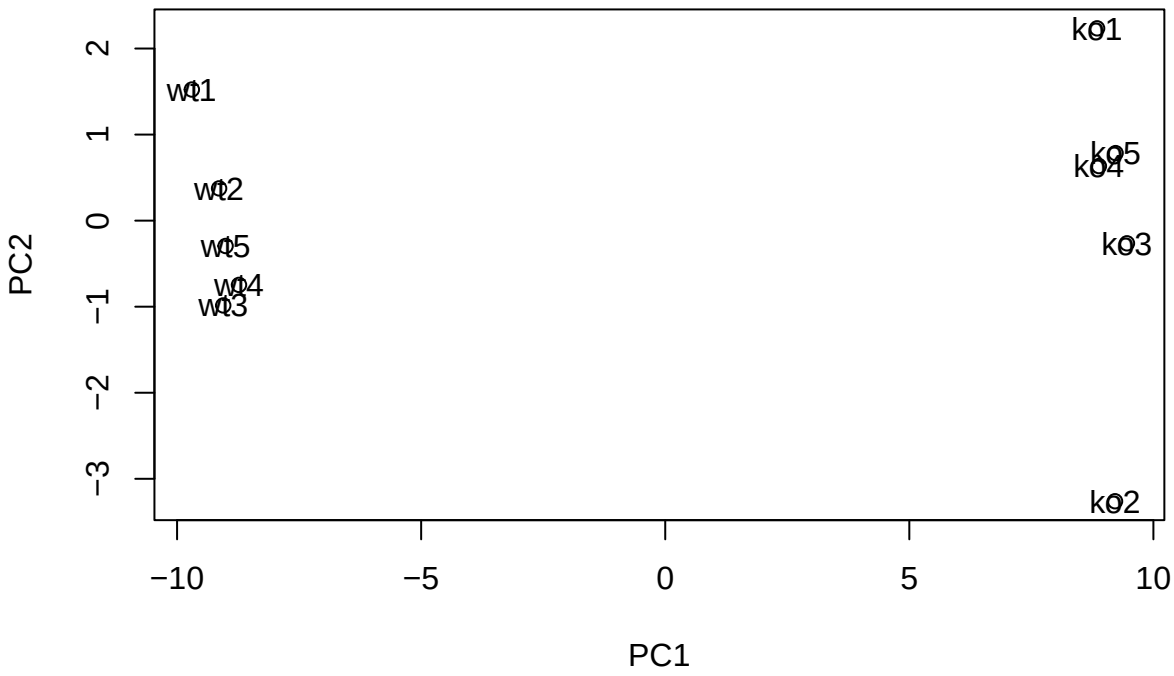
```
pca.rna <- prcomp( t(rna.data), scale=TRUE )
summary(pca.rna)
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.6237 1.5198 1.05787 1.05203 0.88062 0.82545 0.80111
## Proportion of Variance 0.9262 0.0231 0.01119 0.01107 0.00775 0.00681 0.00642
## Cumulative Proportion  0.9262 0.9493 0.96045 0.97152 0.97928 0.98609 0.99251
##                            PC8     PC9      PC10
## Standard deviation     0.62065 0.60342 3.348e-15
## Proportion of Variance 0.00385 0.00364 0.000e+00
## Cumulative Proportion  0.99636 1.00000 1.000e+00
```

```
plot(pca.rna)
```




```
plot(pca.rna$x[,1:2])
text(pca.rna$x[,1:2], labels=colnames(rna.data))
```



PC1 did a good job accounting for the separation between WT and KO. If you go to the loadings you can find what is responsible for the variation.

Lets knit to PDF and publish to github