Machine Learning 1

Angelita Rivera (PID A15522236)

10/21/2021

First up is clustering methods

#Kmeans clustering

The function in base R to do Kmeans clustering is called 'kmeans()'.

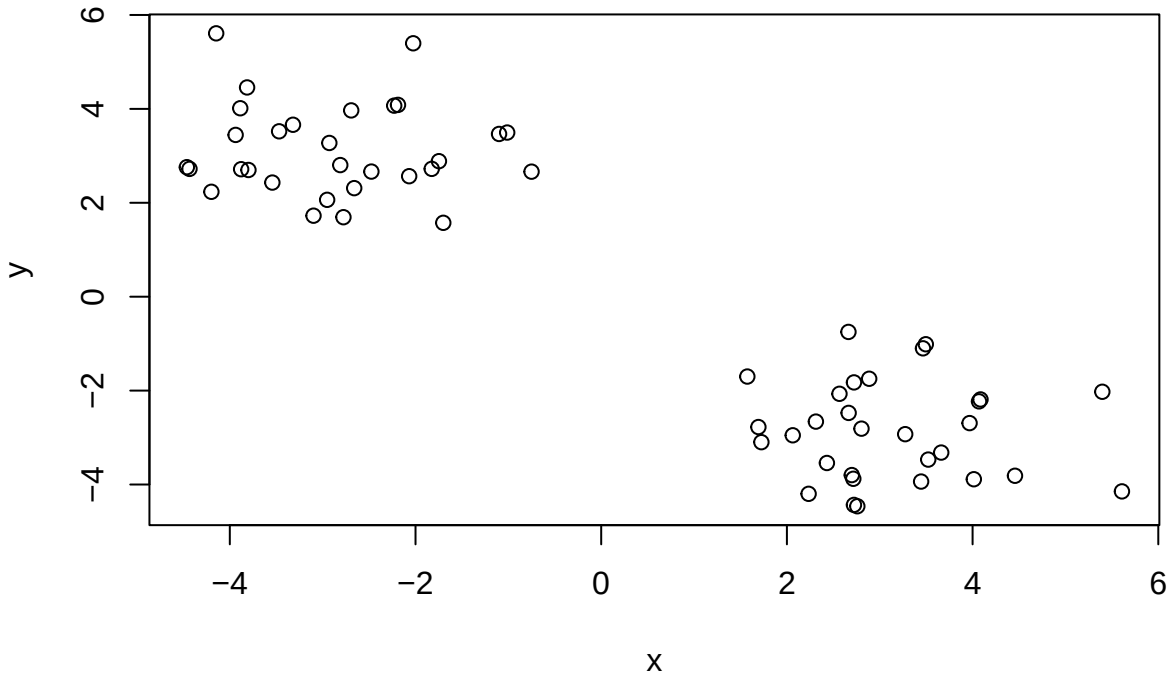
First make up some data where we know what the answer should be:

```
#rnorm() gives 30 points, separated around -3 or 3
tmp <-  c(rnorm(30, -3), rnorm(30,3))
x <-  cbind(x = tmp, y = rev(tmp))
#The rev() function does the reverse of the function
x
```

```
##                x          y
##  [1,] -4.4628410  2.7585336
##  [2,] -3.8874885  4.0134911
##  [3,] -2.9281597  3.2742282
##  [4,] -1.0125554  3.4961178
##  [5,] -2.7763685  1.6930961
##  [6,] -3.8138671  4.4567430
##  [7,] -3.7993193  2.6989030
##  [8,] -2.4753802  2.6649526
##  [9,] -4.4343684  2.7222104
## [10,] -2.1886001  4.0852044
## [11,] -3.9373897  3.4456207
## [12,] -4.1468983  5.6087043
## [13,] -2.9518537  2.0629983
## [14,] -3.3196600  3.6621205
## [15,] -2.0247579  5.3970888
## [16,] -0.7503547  2.6630338
## [17,] -2.6602087  2.3122042
## [18,] -2.0682779  2.5663033
## [19,] -1.0999454  3.4658443
## [20,] -2.6924110  3.9675714
## [21,] -3.4689063  3.5222993
## [22,] -1.7476228  2.8863525
## [23,] -3.0985216  1.7253514
## [24,] -2.2295016  4.0691324
## [25,] -1.7008168  1.5738121
## [26,] -2.8103250  2.8033119
## [27,] -4.1985016  2.2339437
## [28,] -3.5418962  2.4308283
```

```
## [29,] -1.8254238  2.7224642
## [30,] -3.8780184  2.7151575
## [31,]  2.7151575 -3.8780184
## [32,]  2.7224642 -1.8254238
## [33,]  2.4308283 -3.5418962
## [34,]  2.2339437 -4.1985016
## [35,]  2.8033119 -2.8103250
## [36,]  1.5738121 -1.7008168
## [37,]  4.0691324 -2.2295016
## [38,]  1.7253514 -3.0985216
## [39,]  2.8863525 -1.7476228
## [40,]  3.5222993 -3.4689063
## [41,]  3.9675714 -2.6924110
## [42,]  3.4658443 -1.0999454
## [43,]  2.5663033 -2.0682779
## [44,]  2.3122042 -2.6602087
## [45,]  2.6630338 -0.7503547
## [46,]  5.3970888 -2.0247579
## [47,]  3.6621205 -3.3196600
## [48,]  2.0629983 -2.9518537
## [49,]  5.6087043 -4.1468983
## [50,]  3.4456207 -3.9373897
## [51,]  4.0852044 -2.1886001
## [52,]  2.7222104 -4.4343684
## [53,]  2.6649526 -2.4753802
## [54,]  2.6989030 -3.7993193
## [55,]  4.4567430 -3.8138671
## [56,]  1.6930961 -2.7763685
## [57,]  3.4961178 -1.0125554
## [58,]  3.2742282 -2.9281597
## [59,]  4.0134911 -3.8874885
## [60,]  2.7585336 -4.4628410
```

```
plot(x)
```



Q. Can we use kmeans() to cluster this data setting k 2 and nstart to 20?

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1  3.123254 -2.864341
## 2 -2.864341  3.123254
##
## 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 1 1
##
## Within cluster sum of squares by cluster:
## [1] 60.09625 60.09625
##  (between_SS / total_SS =  89.9 %)
##
## 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
```

There are 30 points in each cluster.

Q. What 'component' of your result object 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 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
```

The function 'km$cluster' gives the cluster assignments/membership.

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

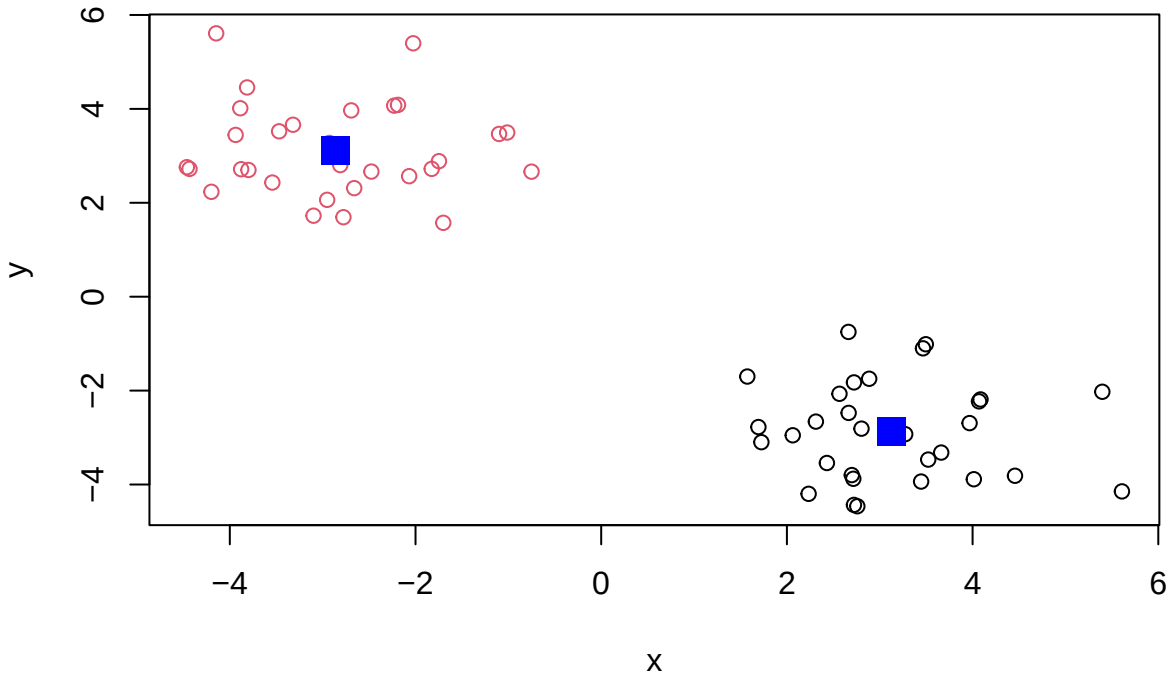
```
km$centers
```

```
##           x         y
## 1  3.123254 -2.864341
## 2 -2.864341  3.123254
```

The function 'km$center' gives the cluster center.

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

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



#Hierarchical Clustering

A big limitation with k-means is that we have to tell it K (the number of clusters we want).

Analyze the same data with hclust()

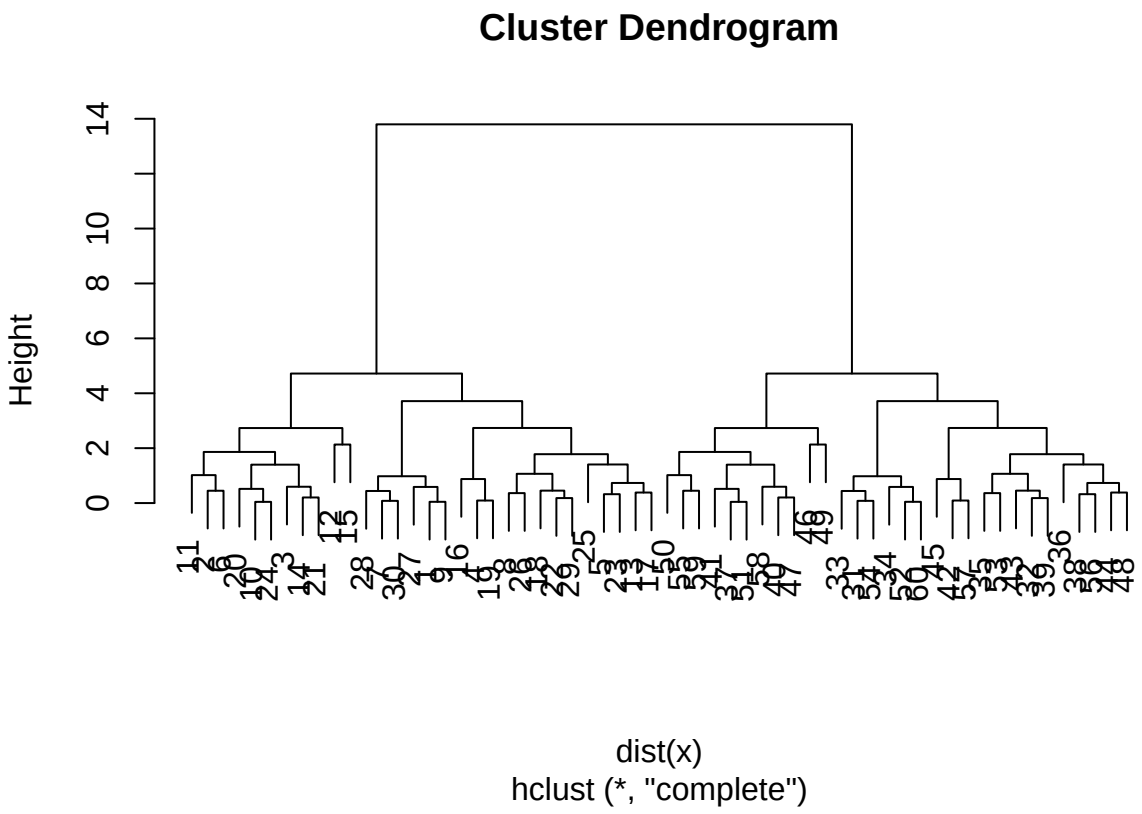
Demonstrate the use of dist(), hclust(), plot(), and cutree() functions to do clustering, generate dendrograms and return cluster assignment membership vector. . .

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

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

There is a plot method for hclust() result objects. Let's see it.

```
plot(hc)
```



To get our membership vector we have to do a wee bit more work. We have to "cut" the tree where we think it makes sense. For this we use the 'cutree()' function,

```
cutree(hc, h = 6)
```

```
##  [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 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
```

```
#h=6 means it's cutting at height 6 ... from tree above.
```

You could also call 'cutree()' setting k = the number of grps/clusters you want.

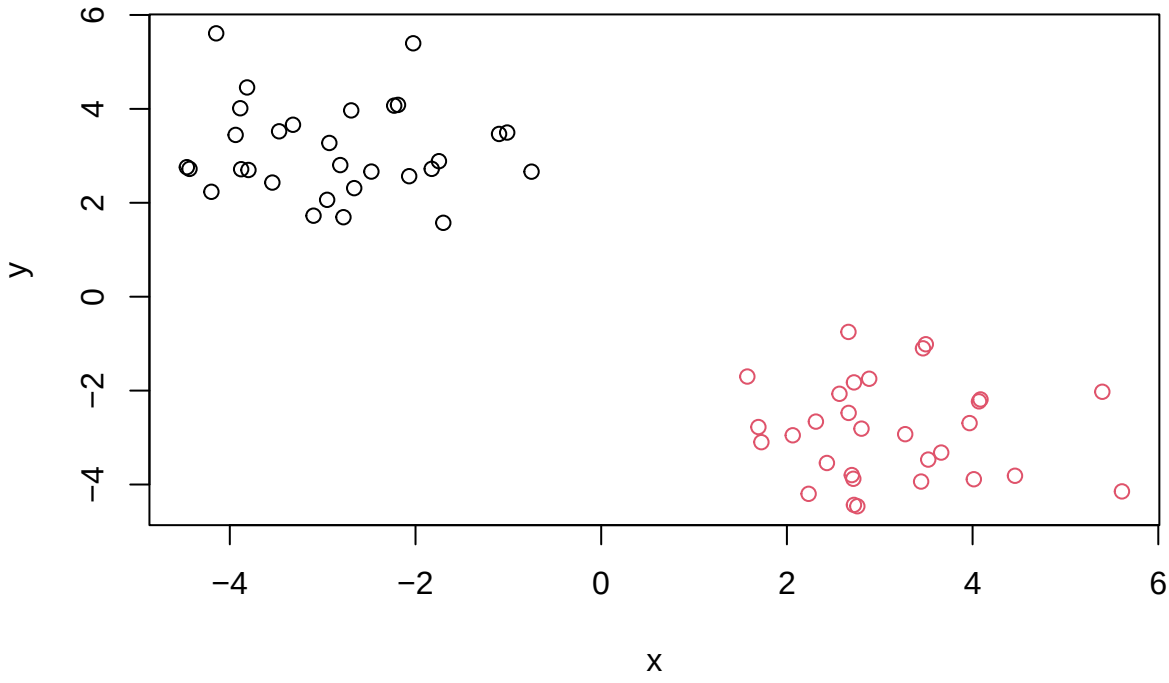
```
cutree(hc, k=2)
```

```
##  [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 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
```

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

Make our results plot.

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



Class 8 Lab #Principal Component Analysis

Read data on food stuffs from 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
```

Q1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

```
nrow(x)
```

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

```
ncol(x)
```

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

Checking your data: Preview the first 6 rows of the dataset.

```
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
```

Uh-oh! The row-names are incorrectly set as the first column of our data frame. Let's try to fix it by getting rid of the extra first column.

One way we could do it is:

```
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
```

But, if we run it again, we lose a country. So, it's dangerous! Let's find another way to do this.

```
url <- "https://tinyurl.com/UK-foods"
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
```

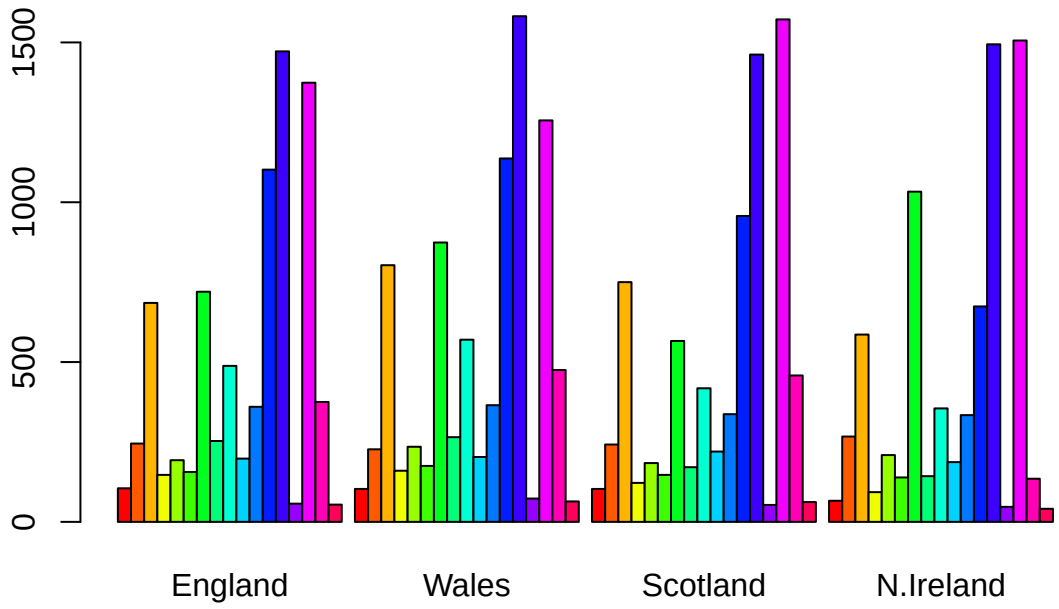
Here, we argued with the initial reading function to get rid of the column #1 while we imported the data. This is a safer way of rearranging the columns.

> 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?

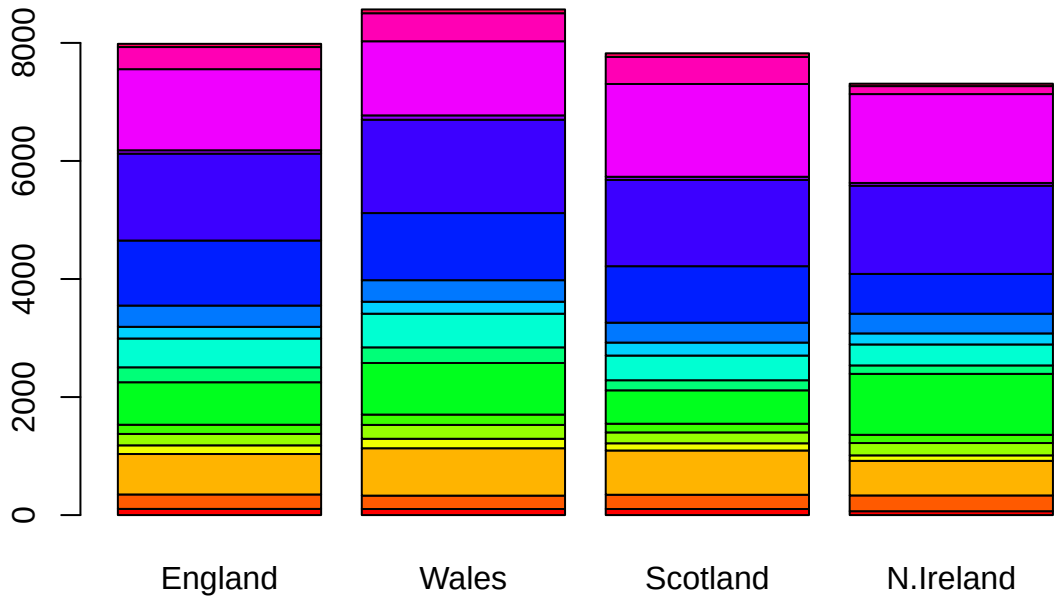
As mentioned above, it is safer and therefore better to use the second method. This is because if we use the first and keep running the code, we will eventually loose all of our data. Whereas the second method, let's us edit our display as we're importing it to R.

> 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)))
```



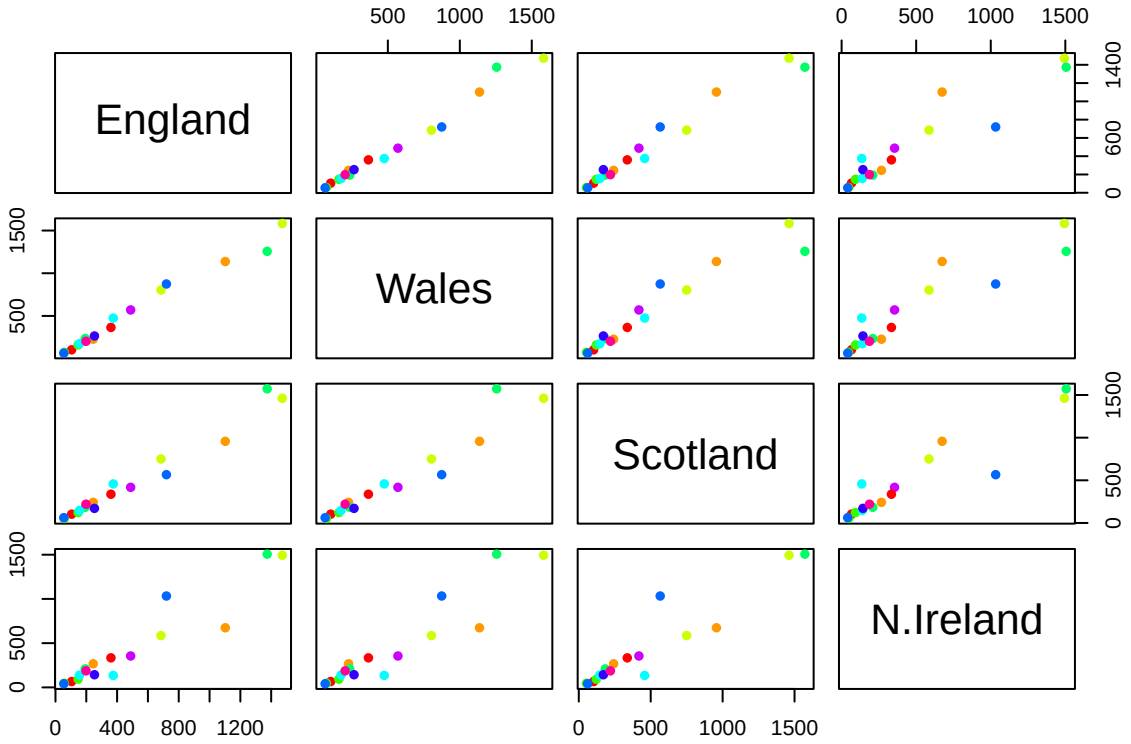

```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



To change between plots, you set the argument 'beside' to FALSE instead of TRUE.

Q5. Generating all pairwise plots may help somewhat. 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(10), pch=16)
```



The function let's us read 'pairs' of all the countries. For example, in the first row, England is being compared all three places (Wales, Scotland, and N. Ireland). This is a matrix of scatter plots that shows us the different pairwise comparisons of all the different variables (i.e. different food stuffs represented are shown by different colored points). I belive, If a point lies on the diagonal it's following the expected plot/more similar.

#PCA to the rescue! > Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

The two countries have the most differences; because their data points are farther apart (deviating from the linear path).

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

```
attributes(pca)
```

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

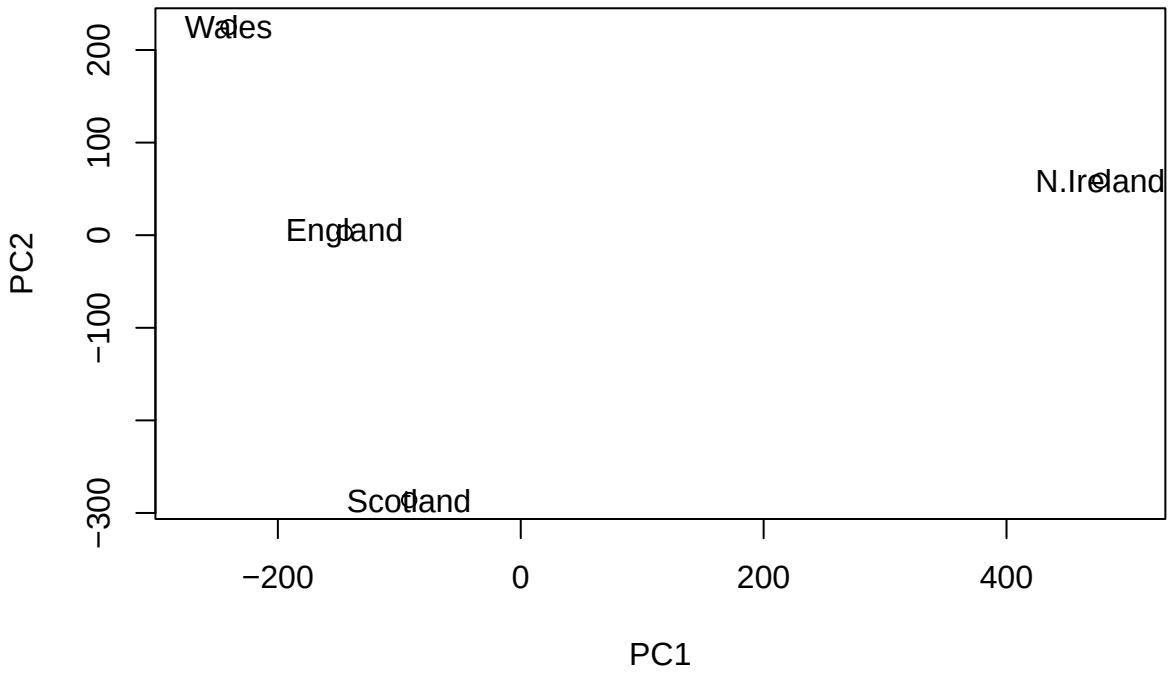
```
##
## $class
## [1] "prcomp"
```

Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
pca$x
```

```
##                  PC1         PC2         PC3           PC4
## England   -144.99315    2.532999 -105.768945  2.842865e-14
## Wales     -240.52915  224.646925   56.475555  7.804382e-13
## Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
## N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

```
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```

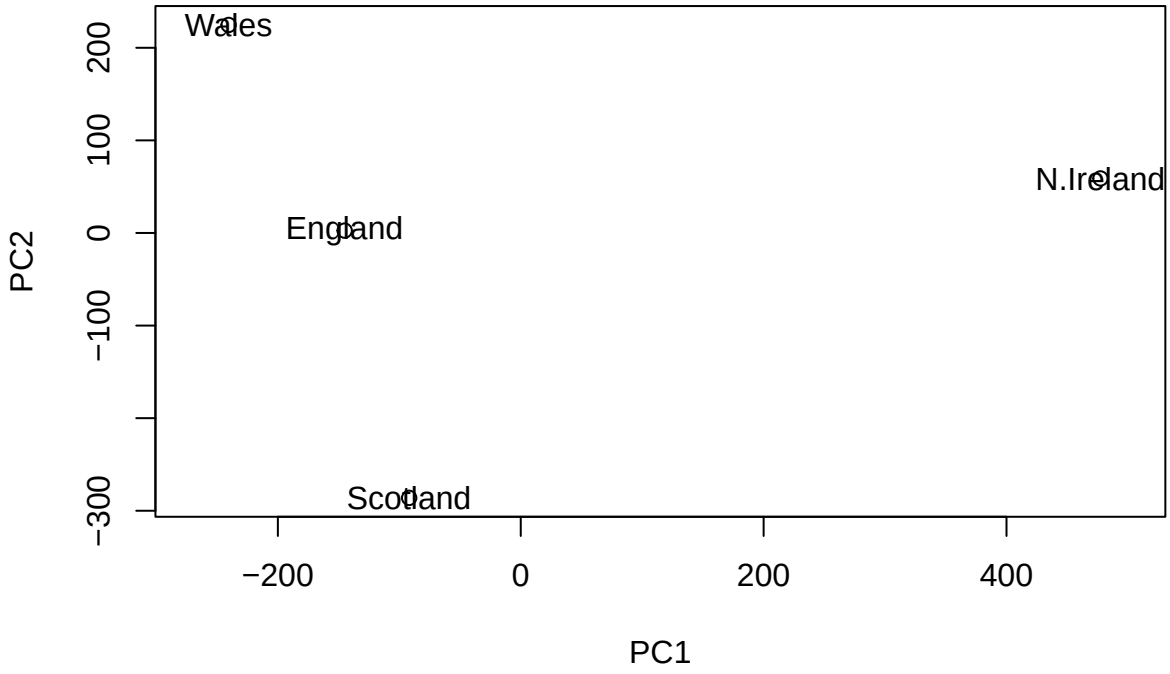


Q8. Customize your plot so that the colors of the country names match the colors in our UK and Ireland map and table at start of this document.

```
pca$x
```

```
##                 PC1         PC2         PC3           PC4
## England   -144.99315    2.532999 -105.768945  2.842865e-14
## Wales     -240.52915  224.646925   56.475555  7.804382e-13
## Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
## N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

```
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x))
```

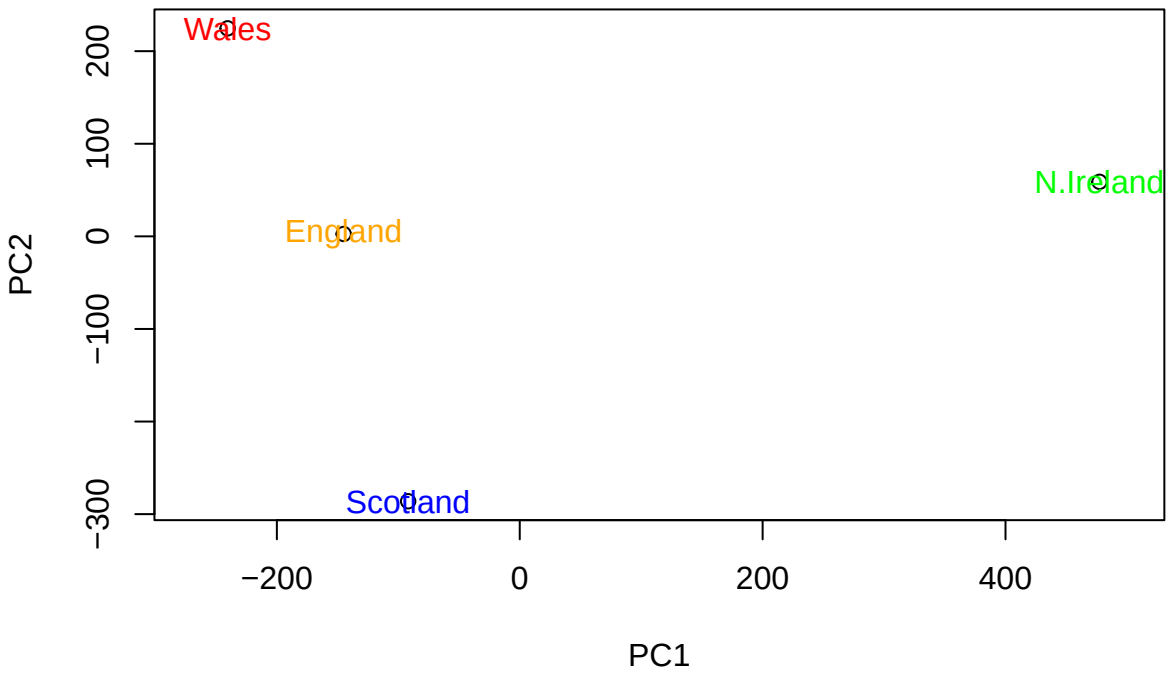



```
color <- c("orange", "red", "blue", "green")

pca$x
```

```
##                 PC1         PC2         PC3           PC4
## England   -144.99315    2.532999 -105.768945  2.842865e-14
## Wales     -240.52915  224.646925   56.475555  7.804382e-13
## Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
## N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

```
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x), col=color)
```



Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

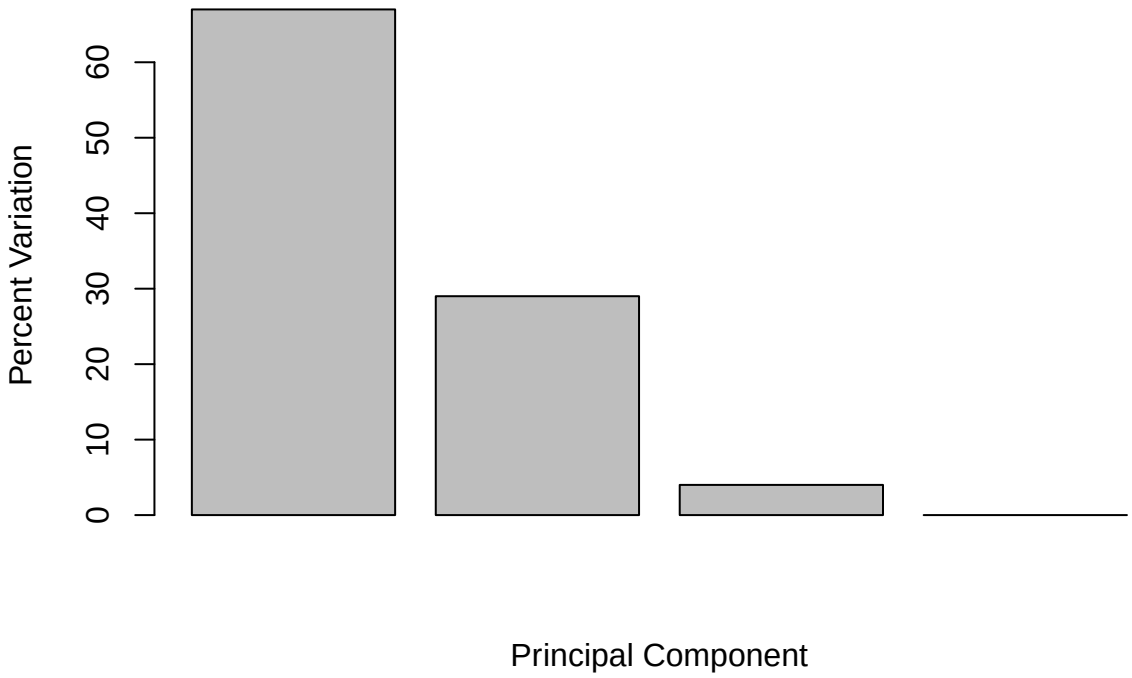
```
#v = how much variation in the original data each PC accounts for
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

```
## [1] 67 29  4  0
```

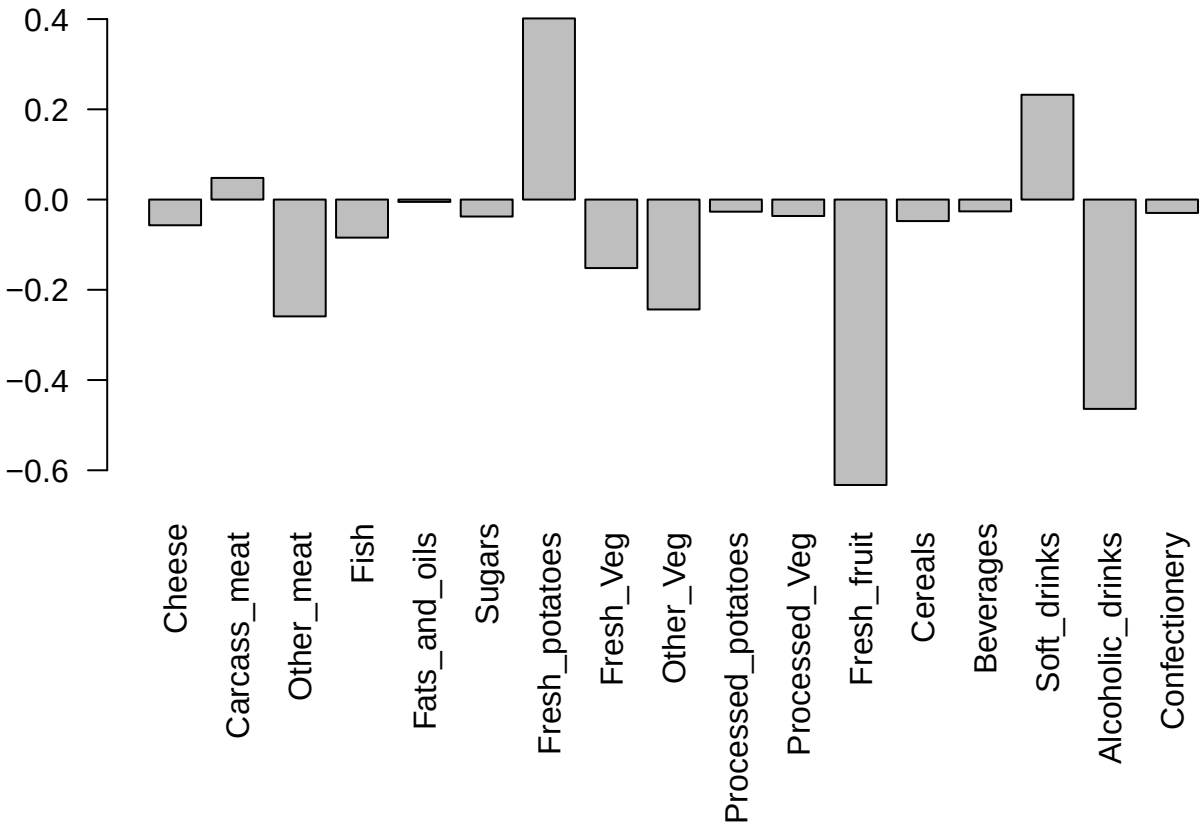
```
# or the second row here...
z <- summary(pca)
z$importance
```

```
##                             PC1       PC2      PC3          PC4
## Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
## Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
## Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

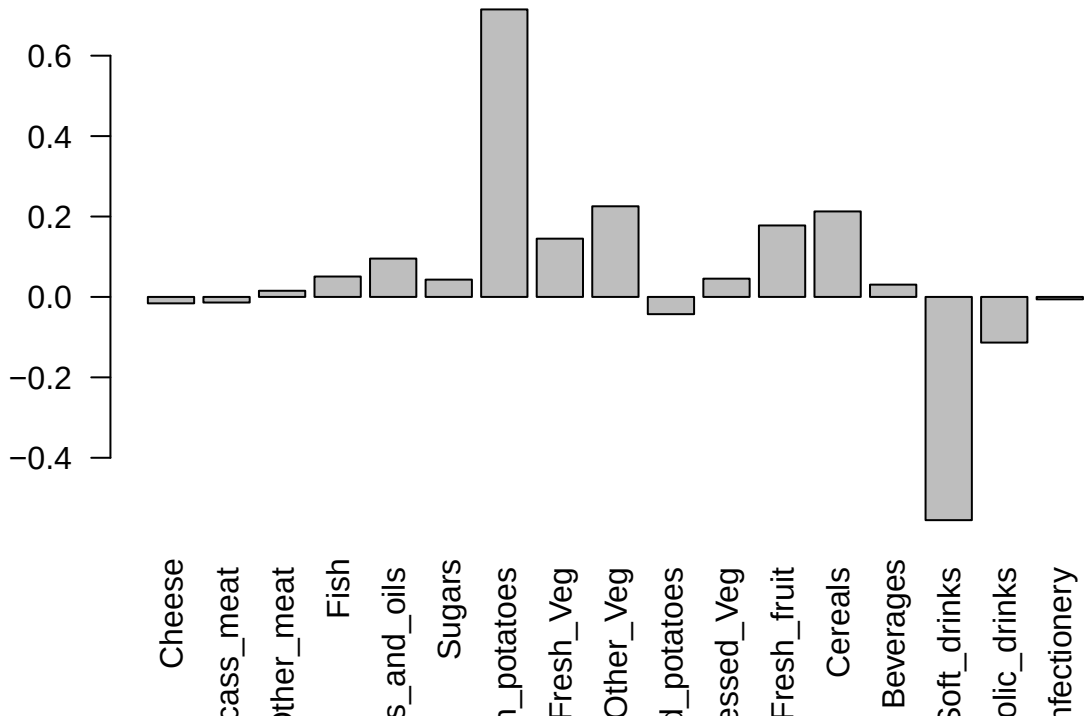
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```




```
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```




```
barplot( pca$rotation[,2], las=2 )
```



It features Fresh_potatoes and Soft_drinks. PC2 tells us that about the variation.

Q10. How many genes and samples are in this data set?

```
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
```

```
dim(rna.data)
```

```
## [1] 100  10
```

There is 100 genes and 10 samples.