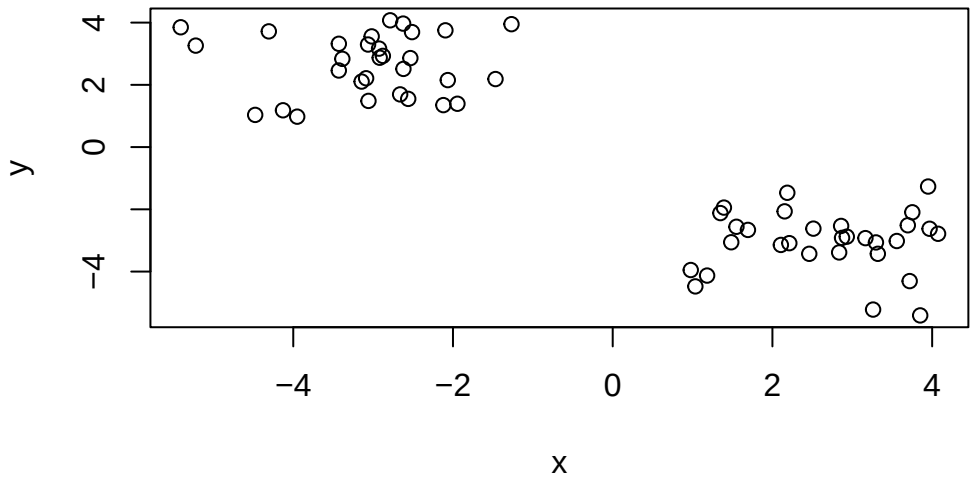
# Class07: Machine Learning 1

Gregory Jordan

## K-means Clustering

Let's make up some data to cluster.

```
tmp<-c(rnorm(n=30,-3),rnorm(30,3))
x<-cbind(x=tmp,y=rev(tmp))
plot(x)
```




```
#cbind binds objects by columns (rbind binds by rows)
#rev is the function to reverse the order of a vector
```

The function to do k-means clustering in base R is called `kmeans()`. we give this our input data for clustering and the argument `centers`, which we tell how many clusters we want.

Pass x into kmeans and centers argument to see how it clusters our previously made plot

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

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

Cluster means:
          x         y
1  2.646587 -3.036311
2 -3.036311  2.646587

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] 55.57198 55.57198
 (between_SS / total_SS =  89.7 %)

Available components:

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

```
#nstart is the input for how many random sets should be chosen
```

Q. How many points are in each cluster?

```
km$size
```

```
[1] 30 30
```

Q. What 'component' of your result object details - cluster size? - cluster assignment/membership? - cluster center?

```
#use $ notation to access different lists from k output
#cluster size
```

```
km$size
```

```
[1] 30 30
```

```
#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
```

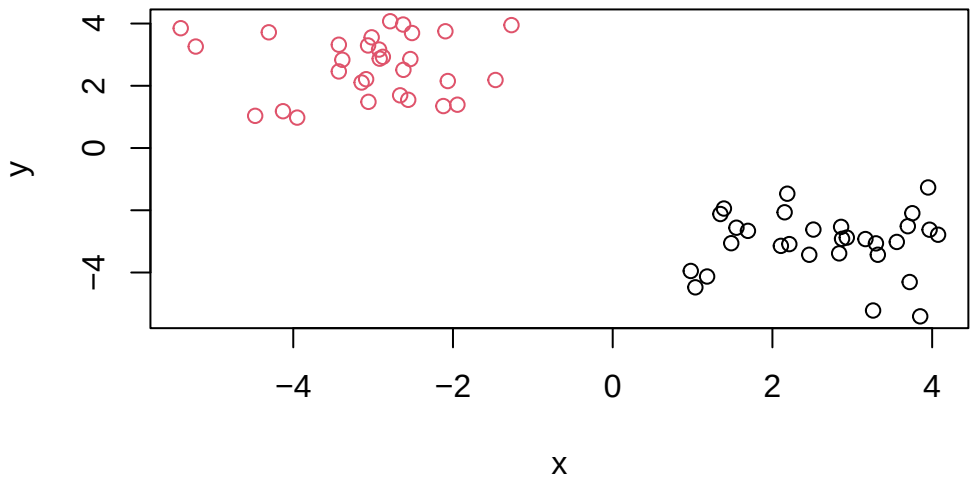
```
#cluster center
km$centers
```

```
          x         y
1  2.646587 -3.036311
2 -3.036311  2.646587
```

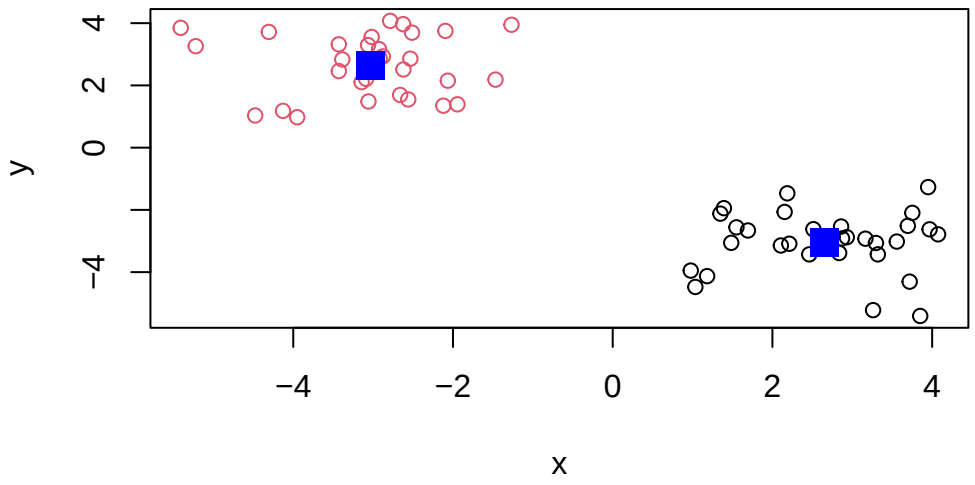
Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```
#using col makes a vector of values under the hood and maps it to the plot. for example co

#in this same regard, if you made col=c("red","blue") then your color vector would be [red
plot(x,col=km$cluster)
```




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



```
#pch is how you change types of points
#cex stands for character expansion and you put in a number to adjust scale of point
```

This showcases that if you are going to cluster by k means clustering you really have to know what your k value is. Or else you can be forcing your k value onto your data leading to false results. Here comes….heirarchial clustering!

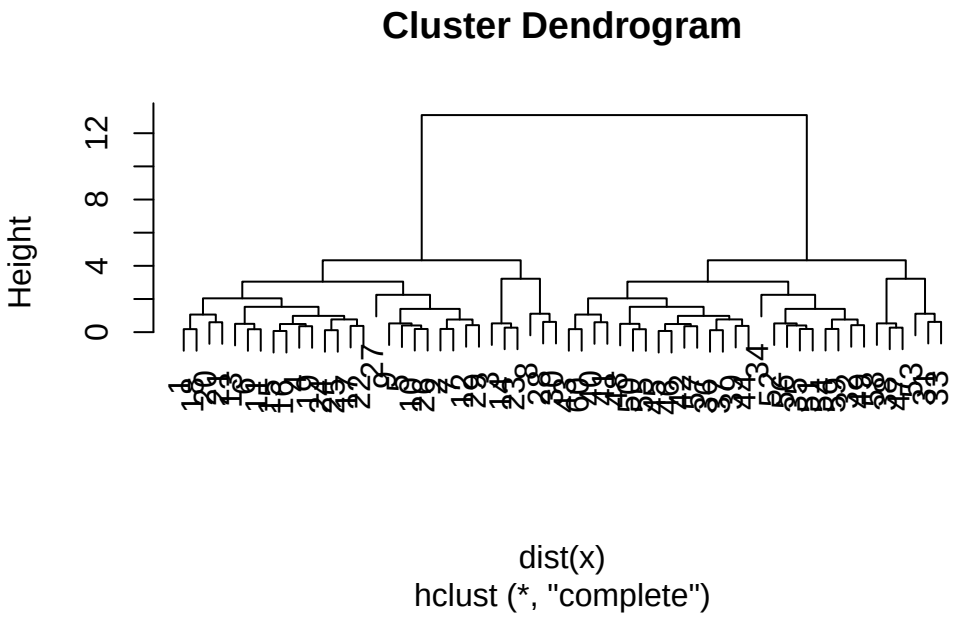
## Heirarchial Clustering

The `hclust()` function perfors heirarchial clustering. The big advantage here is I don't need to specify a K value!

to run `hclust()` I need to provide a distance matrix as input (not simply inputting the original data. we need our data in a dist matrix as the input)

```
#dist function takes a distance matrix of distance between points. if you had 30 points fr
hc<-hclust(dist(x))
#simply running hc just gives you your basic input and results. HOwever, there is a built
```

```
plot(hc)
```



```
#here we see a cluster dendogram. We start with 60/2 clusters = 30 clusters. Then we merge
```

To get my "main" result (cluster membership) I want to "cut" this tree to yield "branches" whose "leaves" are the members of the cluster.

```
#cutree function is used to get get the values of your clusters. It's like your clutting t
cutree(hc,h=8)
```

```
 [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
```

```
#h is your height parameter and you are specifying at what point you cut your cluster. in
```

more often we will use `cutree()` with k=2 for example to get the cut where we get 2 resulting groups

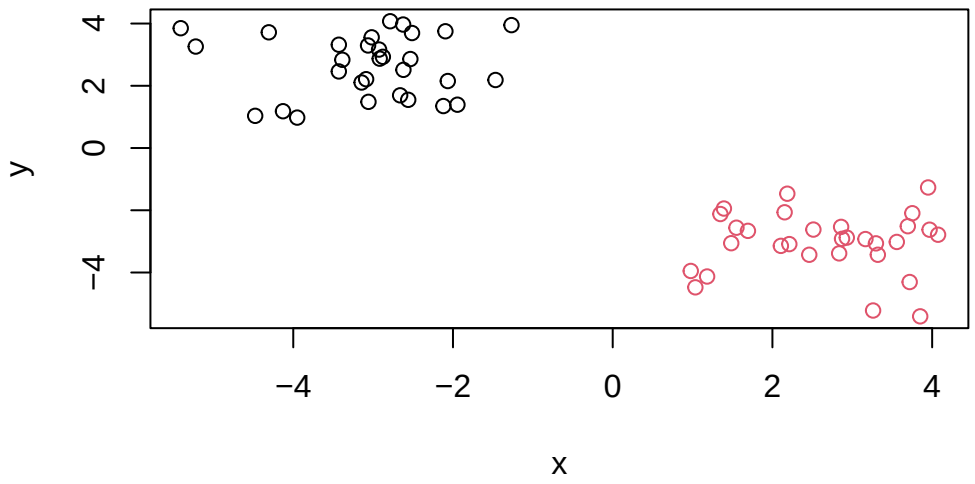
```
cutree(hc,k=2)
```

```
 [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 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
```

```
#use k=2 to cut the tree so we get 2 resulting groups
```

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

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




```
#look we see the same graph as with k means!
```

## Principal Component Analysis (PCA)

Read data for UK food from internet

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

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
```

```
#17 rows and 5 columns. dim function tells us the dimensions of our data
```

```
#we see the first column is the type of food. we would prefer these to be the row names
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
```

```
#make the row names equal to the food column
row.names(x) <- x$X
```

```
head(x)
```

```
                          X England Wales Scotland N.Ireland
Cheese                Cheese     105   103      103        66
Carcass_meat    Carcass_meat     245   227      242       267
Other_meat        Other_meat     685   803      750       586
Fish                    Fish     147   160      122        93
Fats_and_oils  Fats_and_oils     193   235      184       209
Sugars                Sugars     156   175      147       139
```

```
#now remove the first column
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
```

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 could have also just used the row.names = 1 when we read in x. this is MUCH safer beacuse we do not have to worry about running the code and accidentally lose the first column of x each time we run it which would happen if we did the -1 approach

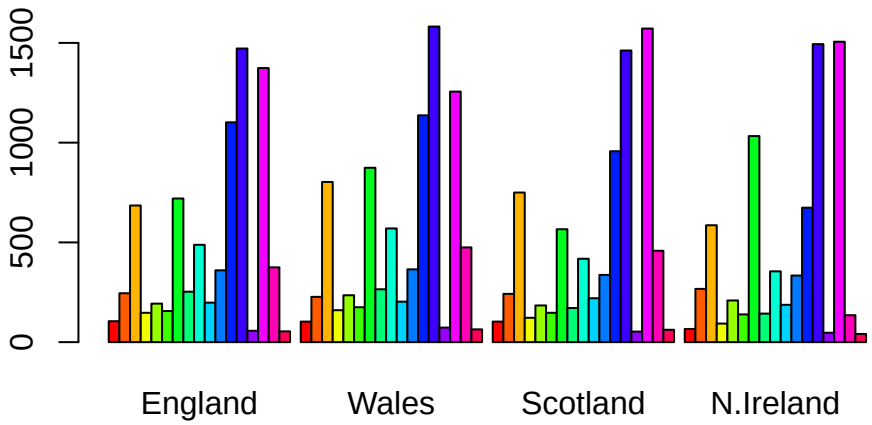
```
x<-read.csv("https://tinyurl.com/UK-foods",row.names = 1)
```

```
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
```

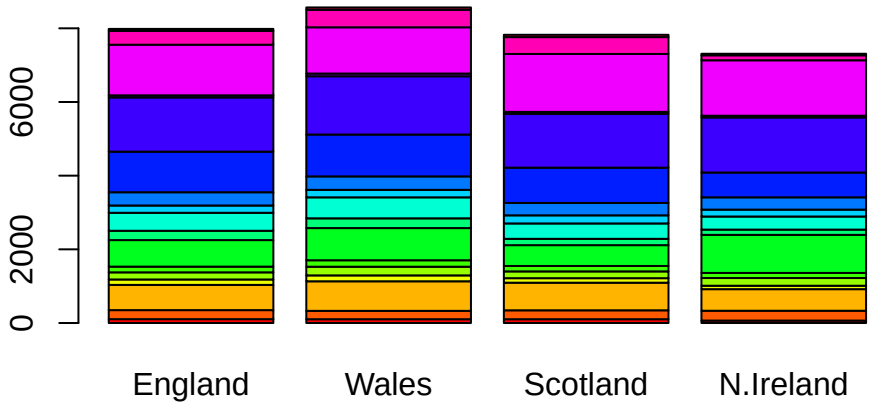
Explore the data - basically plot, plot and plot again

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




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

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

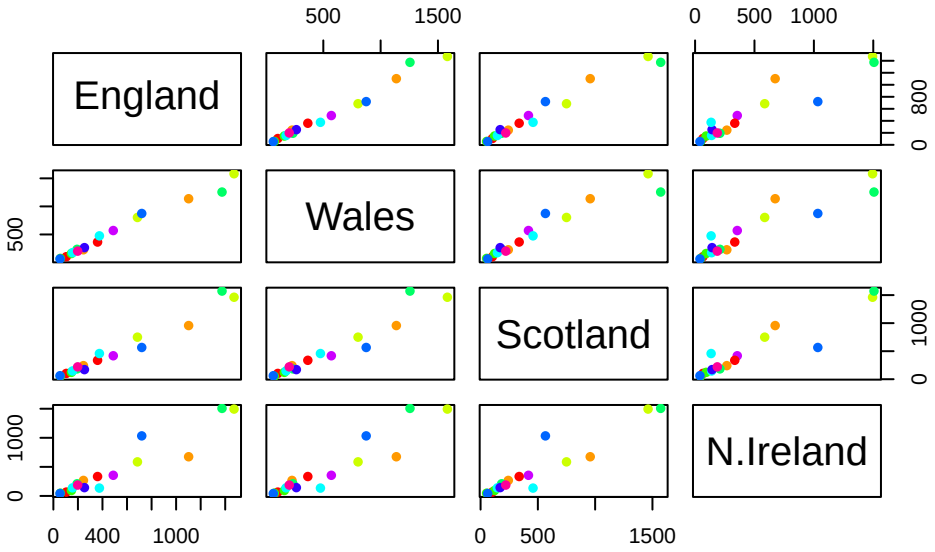



```
#the beside argument
```

A "pairs" plot is somewhat useful...

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)
```




if a given point lies along the diagnal then the values are the sae for each country. If the point lies more towards a country then that country has a higher value

Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

North Ireland has some points that lie significantly lower than the trend line skewed towards/away from them. Therefore, PCA analysis may be used to quantitatively ascertain some important differences.

This is really subjective though...and a lot of eye tiring work.

## PCA to the rescue!

The main function in base R to do PCA is called `prcomp()`

the issue with `prcomp()` is that it wants the transpose version of our data as input.

```
t(x)
```

```
         Cheese Carcass_meat  Other_meat  Fish Fats_and_oils  Sugars
England     105           245         685  147            193    156
Wales       103           227         803  160            235    175
```

```
Scotland     103           242         750  122            184    147
N.Ireland     66           267         586   93            209    139
          Fresh_potatoes  Fresh_Veg  Other_Veg  Processed_potatoes
England               720        253        488                 198
Wales                 874        265        570                 203
Scotland              566        171        418                 220
N.Ireland            1033        143        355                 187
          Processed_Veg  Fresh_fruit  Cereals  Beverages Soft_drinks
England               360         1102     1472        57         1374
Wales                 365         1137     1582        73         1256
Scotland              337          957     1462        53         1572
N.Ireland             334          674     1494        47         1506
          Alcoholic_drinks  Confectionery
England                375             54
Wales                  475             64
Scotland               458             62
N.Ireland              135             41
```

```
#t is to transpose our data. i.e. flip rows and columns
```

```
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
```

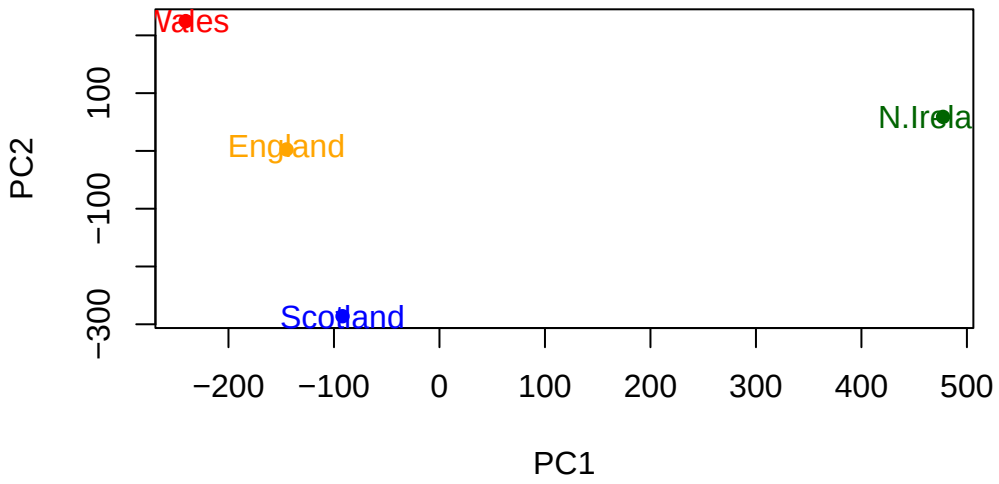
the .6744 value in PC1 shows us that 67% of the data is captured by the first principal component. Showcases that PC1 is the most important and retains a lot of data info. PC2 retains 29%, and thus the cumulative we see that together they retain 96.5% of what is going on with the data just by these principal components. PC3 is trash, PC4 is trash also, so we see that we can reduce this down to just PC1 and PC2 and retain all our info.

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

The object returned by `prcomp()` has our results that include a $x component. This is our "scores" along the PCs (i.e. the plot of our data along the new PC axis)

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

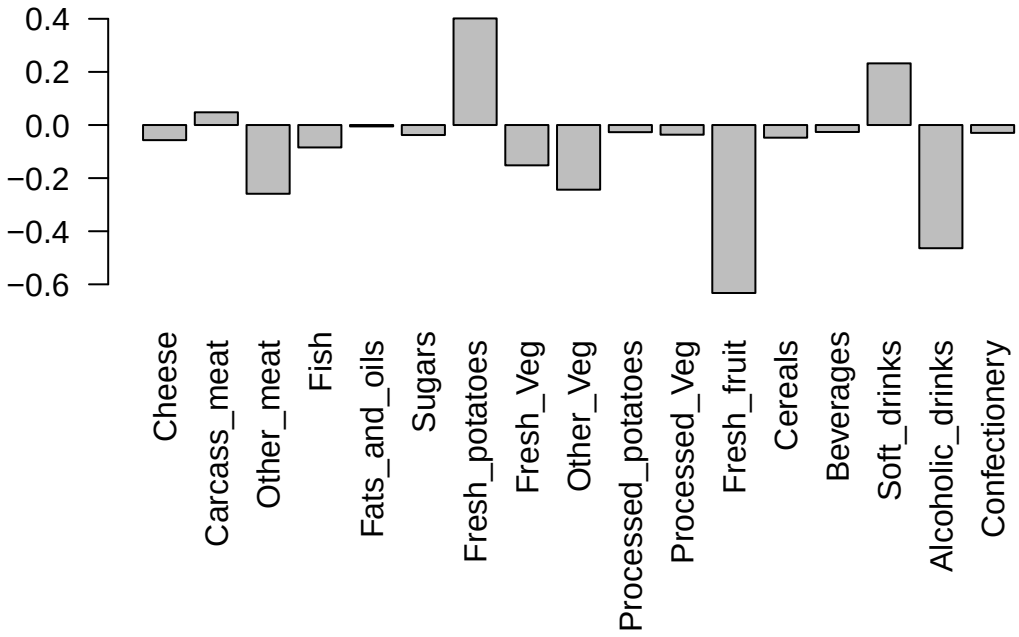
```
plot(pca$x[,1],pca$x[,2],xlab="PC1",ylab="PC2",col=c("orange","red","blue","darkgreen"),pc
text(pca$x[,1],pca$x[,2],colnames(x),col=c("orange","red","blue","darkgreen"))
```




Ireland is way far apart on PC1 which is the principal component so it is way different. The other 3 are apart on PC2, so there is variance but not as extreme as ireland because PC1 covers 67% of the data info while PC2 covers 29%.

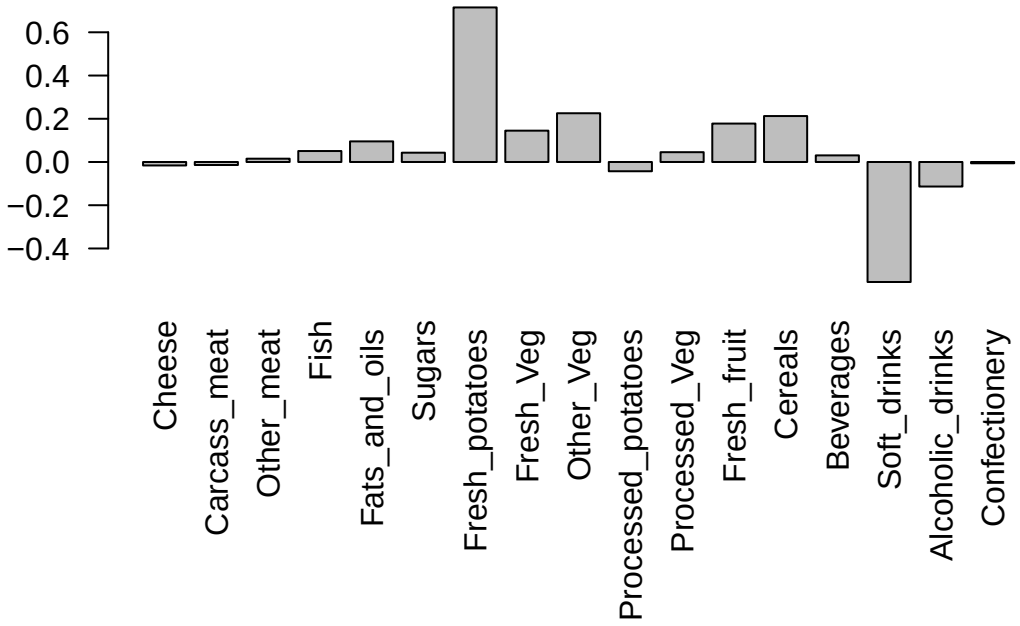
Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

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




```
#this is for PC1
```

```
#this is for PC2
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```




```
#the key difference here is that the second column from pca$rotation is used
```

As we can see, fresh_potatoes and soft_drinks feature prominently, with fresh_potatoes having a srong positive value on the PC2 component and soft_drinks having a strong negative value.