Class 07 - Machine Learning 1

Kristiana Wong A16281367

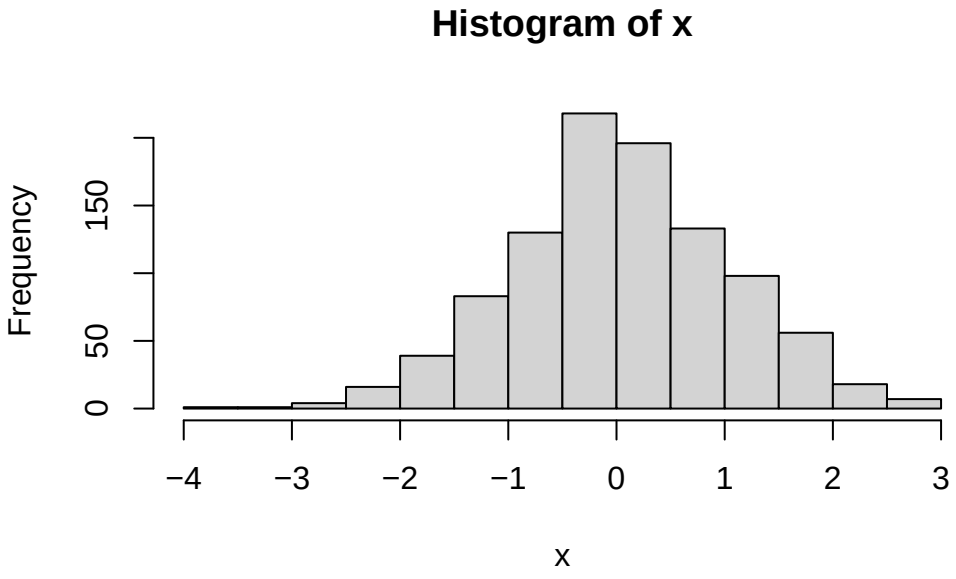
Clustering Methods

The broad goal here is to find groupings (clusters) in your input data

Kmeans

First, let's make up some data to cluster.

```
x <- rnorm(1000)
hist(x)
```



Normally, these distributions values will be around 0, or where ever you set the average to be.

Make a vector of length 60 with 30 points centered at -3 and 30 points at +3.

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

I will now make a small x and y dataset with 2 groups of points. 'rev()' function reverses the vector order.

```
rev(c(1:5))
```

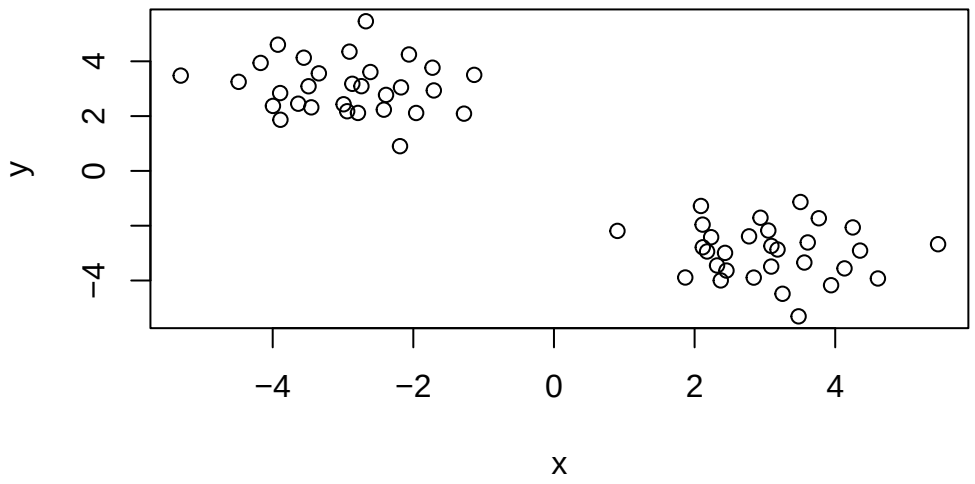
```
[1] 5 4 3 2 1
```

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

```
                x          y
 [1,] -3.4510370  2.3192527
 [2,] -3.5579733  4.1320213
 [3,] -2.3880180  2.7743944
 [4,] -3.4902170  3.0897218
 [5,] -2.7382205  3.0920173
 [6,] -3.6364909  2.4534040
 [7,] -5.3079480  3.4788878
 [8,] -3.9964416  2.3721125
 [9,] -2.9916866  2.4328205
[10,] -2.6751342  5.4626032
[11,] -4.1715599  3.9412426
[12,] -2.9406009  2.1775065
[13,] -2.1895012  0.9028581
[14,] -2.0625284  4.2495867
[15,] -2.9079065  4.3531909
[16,] -1.2795956  2.0901881
[17,] -1.7276784  3.7664877
[18,] -4.4842056  3.2506439
[19,] -1.9600939  2.1139031
[20,] -2.6107056  3.6092374
[21,] -1.7086290  2.9368793
[22,] -3.8939631  2.8412479
[23,] -3.8900656  1.8673738
```

```
[24,] -2.4193813  2.2350511
[25,] -2.7874071  2.1166376
[26,] -2.1752513  3.0481830
[27,] -1.1344785  3.5053685
[28,] -3.9260102  4.6068942
[29,] -2.8678400  3.1794578
[30,] -3.3433386  3.5605386
[31,]  3.5605386 -3.3433386
[32,]  3.1794578 -2.8678400
[33,]  4.6068942 -3.9260102
[34,]  3.5053685 -1.1344785
[35,]  3.0481830 -2.1752513
[36,]  2.1166376 -2.7874071
[37,]  2.2350511 -2.4193813
[38,]  1.8673738 -3.8900656
[39,]  2.8412479 -3.8939631
[40,]  2.9368793 -1.7086290
[41,]  3.6092374 -2.6107056
[42,]  2.1139031 -1.9600939
[43,]  3.2506439 -4.4842056
[44,]  3.7664877 -1.7276784
[45,]  2.0901881 -1.2795956
[46,]  4.3531909 -2.9079065
[47,]  4.2495867 -2.0625284
[48,]  0.9028581 -2.1895012
[49,]  2.1775065 -2.9406009
[50,]  3.9412426 -4.1715599
[51,]  5.4626032 -2.6751342
[52,]  2.4328205 -2.9916866
[53,]  2.3721125 -3.9964416
[54,]  3.4788878 -5.3079480
[55,]  2.4534040 -3.6364909
[56,]  3.0920173 -2.7382205
[57,]  3.0897218 -3.4902170
[58,]  2.7743944 -2.3880180
[59,]  4.1320213 -3.5579733
[60,]  2.3192527 -3.4510370
```

```
plot(x)
```




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

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

Cluster means:
          x         y
1 -2.957130  3.065324
2  3.065324 -2.957130

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 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 2

Within cluster sum of squares by cluster:
[1] 54.54648 54.54648
 (between_SS / total_SS =  90.9 %)

Available components:

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

Q1. From your result object 'k' how many points are in each cluster?

```
k$size
```

```
[1] 30 30
```

Q2. What "component" of your result object details the cluster membership?

```
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 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
```

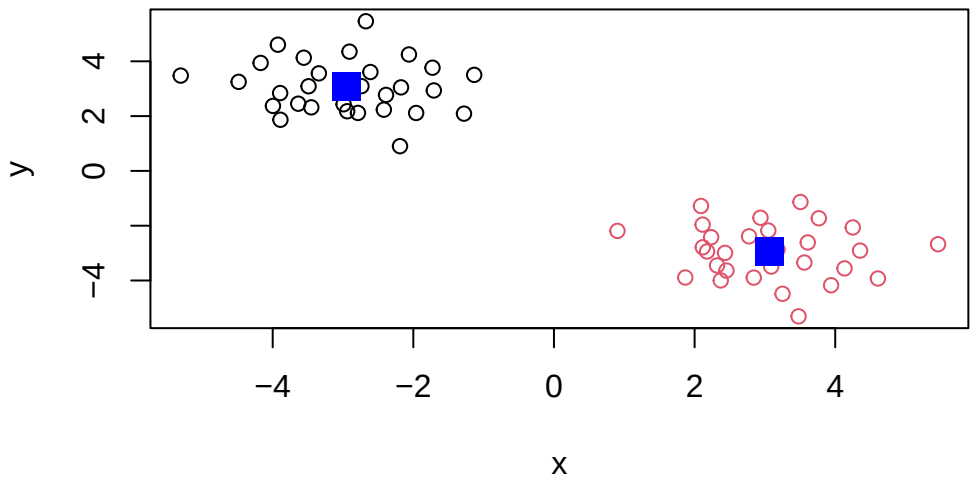
Q3. Cluster centers?

```
k$centers
```

```
          x         y
1 -2.957130  3.065324
2  3.065324 -2.957130
```

Q4. Plot of our clustering results

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

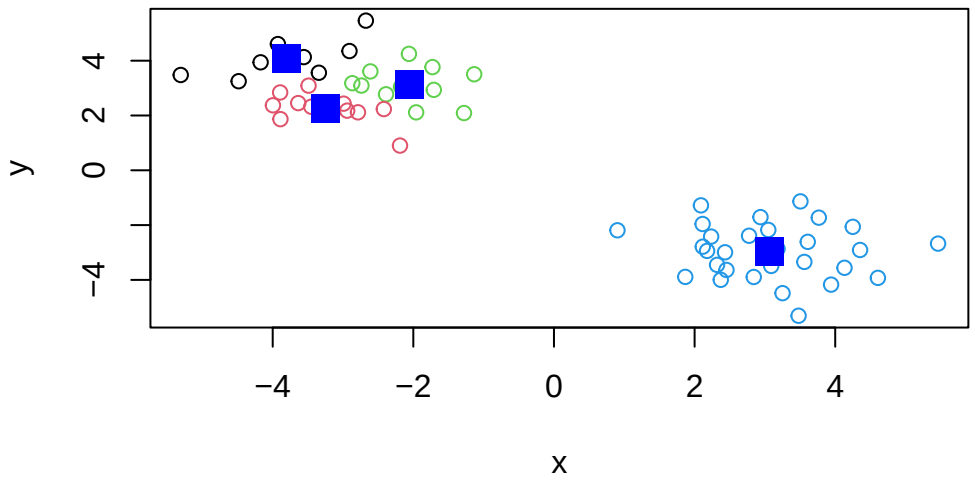



```
a <- kmeans(x, centers = 4)
a
```

```
K-means clustering with 4 clusters of sizes 8, 11, 11, 30

Cluster means:
          x         y
1 -3.796760  4.098253
2 -3.244254  2.255272
3 -2.059367  3.124155
4  3.065324 -2.957130

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

Within cluster sum of squares by cluster:
[1]  8.826311  6.953074  7.563028 54.546478
 (between_SS / total_SS =  93.5 %)

Available components:
```

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

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



A big limitation is that is does what you ask even if you ask for silly clusters.

Hierarchical Clustering

The main base R function for Hierarchical Clustering is 'hclust()'. Unlike 'kmeans()' you cannot just pass it your dataset as an input. You first need to calculate a distance matrix.

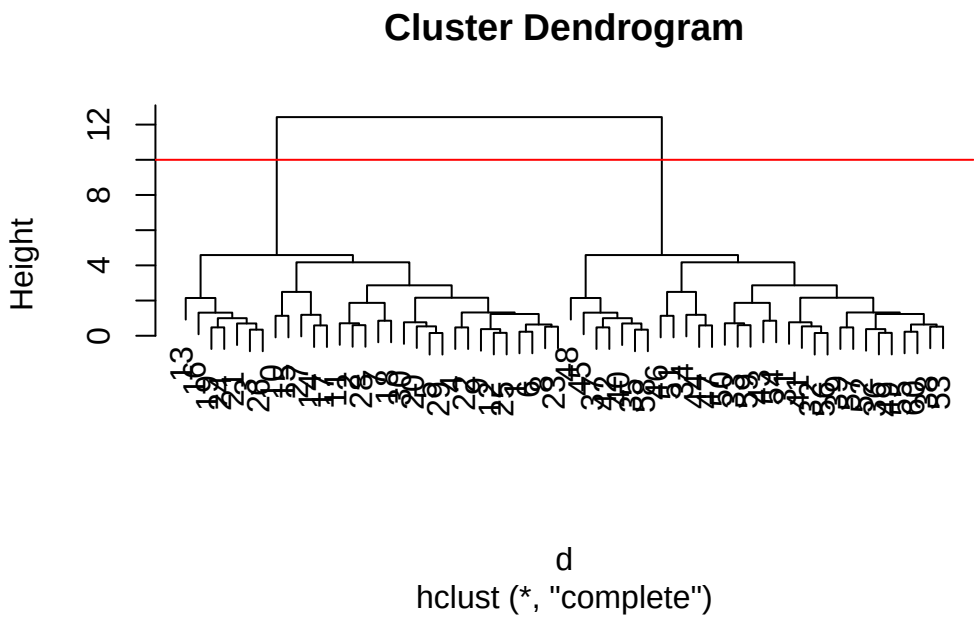
```
d <- dist(x)
hc <- hclust(d)
hc
```

```
Call:
hclust(d = d)
```

```
Cluster method   : complete
Distance         : euclidean
Number of objects: 60
```

Use plot() to view results

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



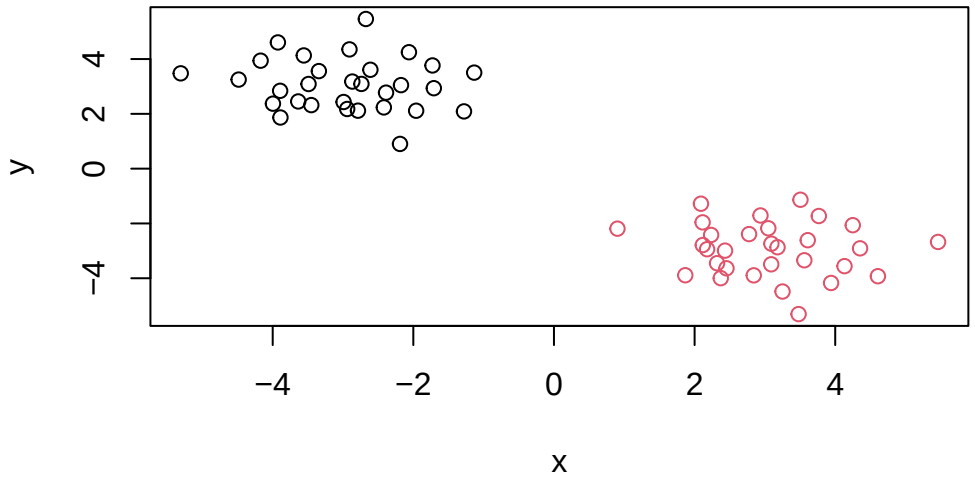
To make the "cut" and get our cluster membership vector we can use the 'cutree()' function.

```
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 2
```

Make a plot of our data colored by hclust results.

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



#Principal Component Analysis (PCA) Here we will do Principal Component Analysis (PCA for short) on some food data from the UK.

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

```
                   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?

```
#Code for previewing columns and rows in the UK food dataset
nrow(UK)
```

```
[1] 17
```

```
ncol(UK)
```

```
[1] 5
```

Now we will check our data to make sure that everything is as expected.

```
#Preview the first 6 rows of data of the UK food dataset
head(UK)
```

```
              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
```

The rows weren't as we expected, so we should change the columns. Note that the below code though isn't very good. Rather, we should edit the orginial CSV import to have the rows be -1.

```
# Note how the minus indexing works
#rownames(UK) <- UK[,1]
#UK <- UK[,-1]
```

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

```
                    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
```

```
head(UK)
```

```
              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
```

Now lets check the dimensions again

```
dim(UK)
```

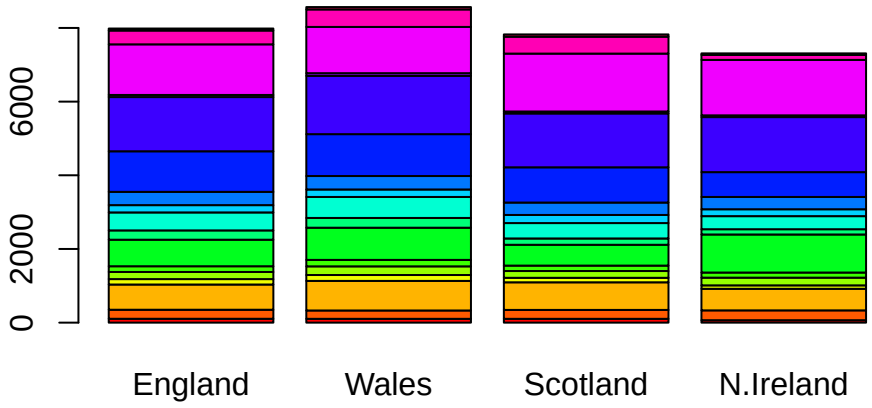
```
[1] 17  4
```

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 prefer to edit the CSV file, since the actual edit to the output can manipulate the data to have the incorrect dimensions.

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

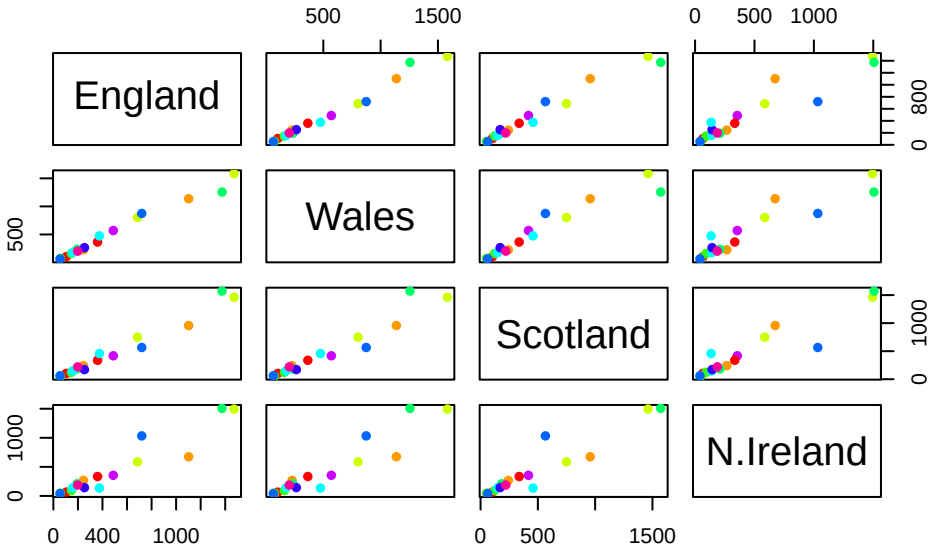
```
#Changing T -> F changes the plot to the one above
barplot(as.matrix(UK), beside=F, col=rainbow(nrow(UK)))
```



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?

The following code generates pairwise plots, pch changes the plotting character. If a given point lies on the diagonal, it means it is being compared to itself.

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



##PCA to the rescue

The main "base" R function for PCA is called 'prcomp()'.

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

```
Importance of components:
                            PC1      PC2      PC3        PC4
Standard deviation     324.1502 212.7478 73.87622 3.176e-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 variance is captured in 2 PCs? 96.5%

To make our main "PC score plot" or PC1 vs PC2 plot (aka "PC1" v.s. "PC2" plot or "PC plot" or "ordination plot".)

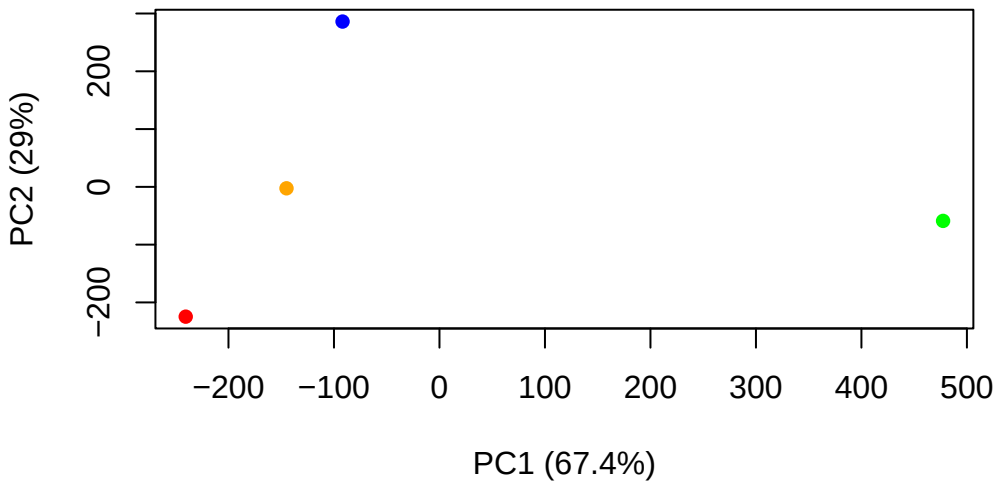
```
attributes(pca)
```

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

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

We are after the 'pca$x' result component to make our main PCA plot.

```
mycols <- c("orange", "red", "blue", "green")
plot(pca$x[,1], pca$x[,2], col=mycols, pch=16, xlab="PC1 (67.4%)", ylab="PC2 (29%)")
```



Another important result from PCA is how the original variable (in this case the foods) contribute to the PCs.

This is the contained in the 'pca$rotation' object - this is often called the "loadings" or "contributions" to the PCs.

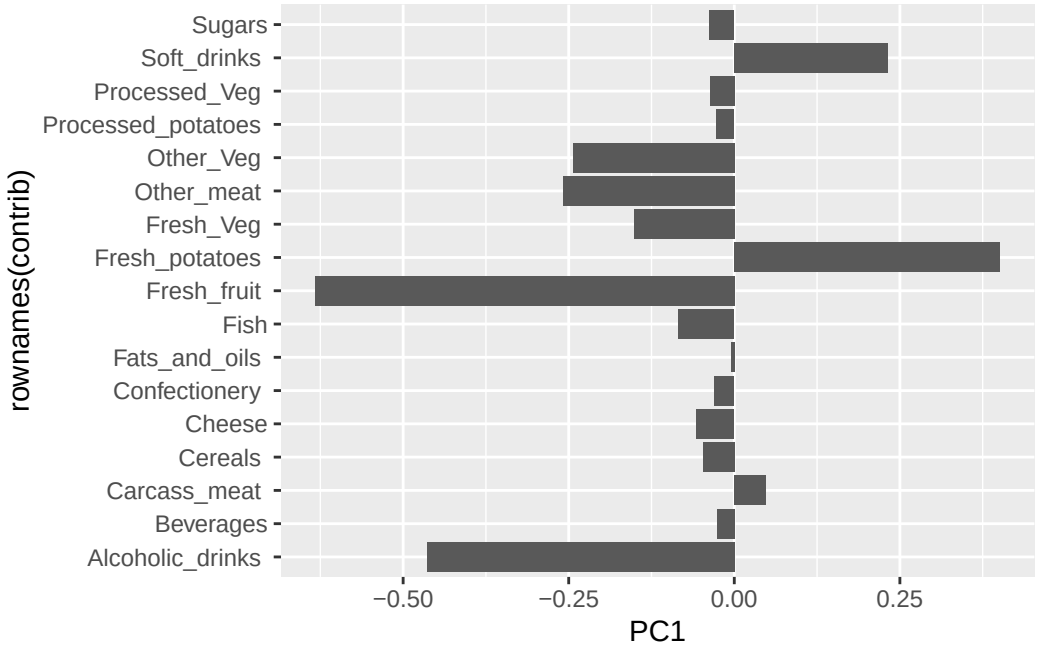
```
pca$rotation
```

```
                    PC1          PC2         PC3          PC4
Cheese       -0.056955380  0.016012850  0.02394295 -0.694538519
Carcass_meat  0.047927628  0.013915823  0.06367111  0.489884628
Other_meat   -0.258916658 -0.015331138 -0.55384854  0.279023718
```

```
Fish                -0.084414983 -0.050754947  0.03906481 -0.008483145
Fats_and_oils       -0.005193623 -0.095388656 -0.12522257  0.076097502
Sugars              -0.037620983 -0.043021699 -0.03605745  0.034101334
Fresh_potatoes       0.401402060 -0.715017078 -0.20668248 -0.090972715
Fresh_Veg           -0.151849942 -0.144900268  0.21382237 -0.039901917
Other_Veg           -0.243593729 -0.225450923 -0.05332841  0.016719075
Processed_potatoes  -0.026886233  0.042850761 -0.07364902  0.030125166
Processed_Veg       -0.036488269 -0.045451802  0.05289191 -0.013969507
Fresh_fruit         -0.632640898 -0.177740743  0.40012865  0.184072217
Cereals             -0.047702858 -0.212599678 -0.35884921  0.191926714
Beverages           -0.026187756 -0.030560542 -0.04135860  0.004831876
Soft_drinks          0.232244140  0.555124311 -0.16942648  0.103508492
Alcoholic_drinks    -0.463968168  0.113536523 -0.49858320 -0.316290619
Confectionery       -0.029650201  0.005949921 -0.05232164  0.001847469
```

We can make a plot along PC1.

```
library(ggplot2)

contrib <- as.data.frame(pca$rotation)
ggplot(contrib) +
  aes(PC1, rownames(contrib)) +
  geom_col()
```


rownames(contrib)
Sugars
Soft_drinks
Processed_Veg
Processed_potatoes
Other_Veg
Other_meat
Fresh_Veg
Fresh_potatoes
Fresh_fruit
Fish
Fats_and_oils
Confectionery
Cheese
Cereals
Carcass_meat
Beverages
Alcoholic_drinks
−0.50
−0.25
0.00
0.25
PC1