CLASS07LAB

Kenny Dang (PID:A18544481)

Table of contents

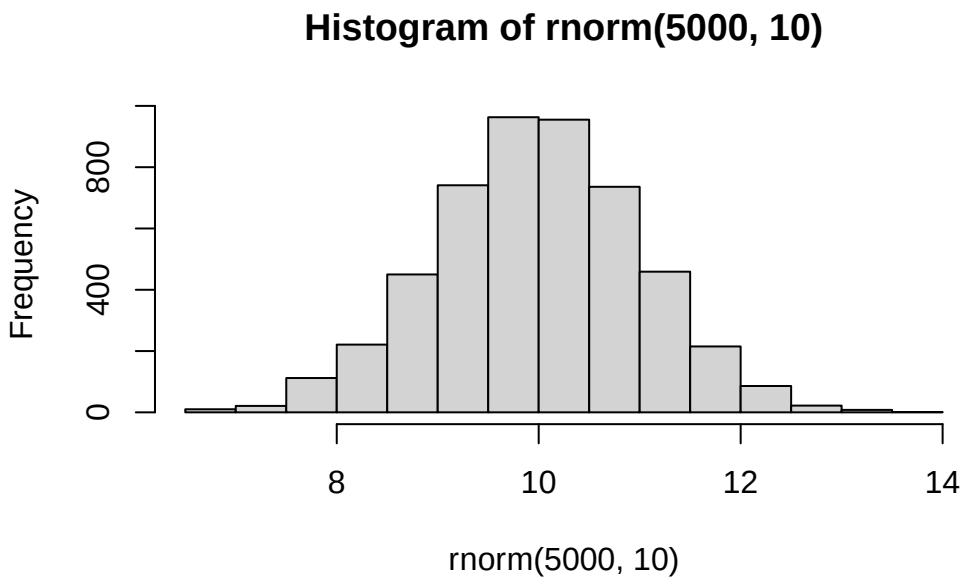
Background

Today we will begin our exploration of some important machine learning methods namely **clustering** and **dimensionality reduction**.

Let's make up some input data for clustering where we know what the natural "clusters" are

The function `rnorm()` can be useful here

```
hist(rnorm(5000, 10))
```



Q. generate 30 random numbers centered at +3

```
rnorm(30, 3)
```

```
 [1] 2.0542401 2.4052712 2.7457525 3.5872095 3.9423985 2.7076045 5.2555077
 [8] 4.0473963 3.5779885 1.0420229 2.9272717 2.6256387 3.6368413 3.1462461
[15] 3.0437061 2.5675229 3.1589716 1.6362379 3.0093894 1.9809580 2.0473208
[22] 2.8166909 3.0905863 0.4654629 4.0237686 1.0069908 3.8198654 2.4056627
[29] 4.3841729 2.9492916
```

```
rnorm(30, -3)
```

```
 [1] -2.967133 -4.021971 -4.110622 -4.488216 -4.887471 -2.377331 -2.783336
 [8] -3.440173 -1.204396 -2.312854 -3.917803 -2.131494 -3.587699 -2.761018
[15] -1.826163 -4.188538 -3.722375 -2.544363 -2.766732 -2.652181 -3.181742
[22] -2.219501 -2.523061 -4.034884 -2.789410 -3.975466 -2.688950 -4.793095
[29] -1.708511 -2.374850
```

Q. generate 30 random numbers centered at -3

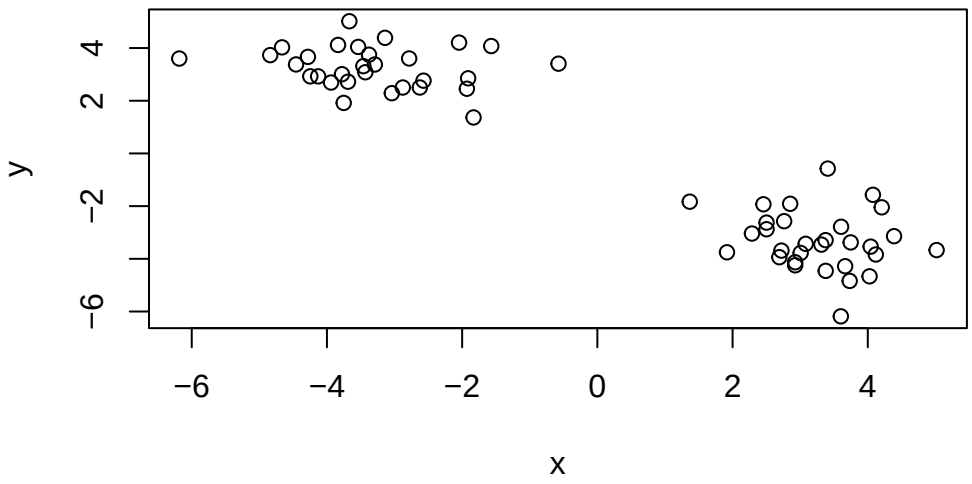
```r
rnorm(30, -3)
```

```
 [1] -1.777520 -4.140917 -1.774876 -5.312634 -4.141177 -2.782727 -3.797258
 [8] -3.549059 -2.875090 -3.242899 -1.292197 -1.258298 -4.210296 -2.860625
[15] -5.373097 -3.009103 -2.618855 -3.000159 -4.561248 -3.799247 -3.201922
[22] -2.334720 -3.335410 -3.988362 -1.929129 -3.228630 -2.168280 -1.947502
[29] -2.438893 -2.801740
```

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

```
 [1]  2.7629104  3.0073085  2.4561542  2.8518897  1.3668192  3.4076399
 [7]  4.3885233  2.2865293  4.0262808  3.6650137  3.7472455  3.3757702
[13]  2.5022426  3.6049127  4.2063669  4.1185069  3.7322512  4.0758104
[19]  2.7228184  3.3131628  5.0180000  3.3773544  2.6920654  2.9257244
[25]  1.9175155  2.9262398  3.0813309  2.5020964  4.0424829  3.6013261
[31] -6.1840059 -3.5375884 -2.6258571 -3.4316400 -4.2459021 -3.7520175
[37] -4.1298729 -3.9389956 -4.4566709 -3.6682198 -3.4608607 -3.6885365
[43] -1.5694351 -4.8387632 -3.8358297 -2.0455855 -2.7833793 -2.8779446
[49] -3.2914955 -3.3776871 -4.2817127 -4.6640497 -3.0405037 -3.1410172
[55] -0.5746713 -1.8314864 -1.9114645 -1.9298437 -3.7774934 -2.5715312
```

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




```
rev(letters)
```

```
 [1] "z" "y" "x" "w" "v" "u" "t" "s" "r" "q" "p" "o" "n" "m" "l" "k" "j" "i" "h"
[20] "g" "f" "e" "d" "c" "b" "a"
```

K-means clustering

The main function in "base R" for K-means clustering is called `kmeans()`:

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

```
 [1]  2.1135720  3.6158853  2.4787535  2.4675816  2.7336211  2.2229199
 [7]  4.2971295  3.8115781  3.5280431  3.6766100  2.9062024  2.8540052
[13]  1.8126444  2.9327472  1.7735714  5.4115532  3.7300674  1.9784879
[19]  2.0026474  3.2831227  4.2585156  3.6120882  3.3870936  3.4087585
[25]  3.0558583  3.8278558  1.2324359  2.3060119  3.9329290  4.2407432
[31] -3.8350760 -3.2086842 -2.8386195 -1.9800089 -3.5164724 -5.1243917
[37] -2.0077950 -3.3000137 -2.7340901 -3.1914377 -3.6743000 -1.9821901
[43] -3.4025908 -3.2589365 -4.6283293 -1.7735148 -2.4853686 -3.0210035
[49] -3.4584055 -3.2208825 -0.5950501 -2.9217113 -2.7530519 -4.4630877
[55] -0.5300384 -3.6872615 -3.3777957 -1.2901281 -2.6817141 -1.9917312
```

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

km <- kmeans(x, 2)
km
```

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

Cluster means:
          x         y
1 -2.897789  3.096434
2  3.096434 -2.897789

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] 58.1931 58.1931
 (between_SS / total_SS =  90.3 %)

Available components:

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

Q. What component of the results object details the cluster sizes?

```
km$size
```

```
[1] 30 30
```

Q. What components of the results object details the cluster centers?

```
km$centers
```

```
          x         y
1 -2.897789  3.096434
2  3.096434 -2.897789
```

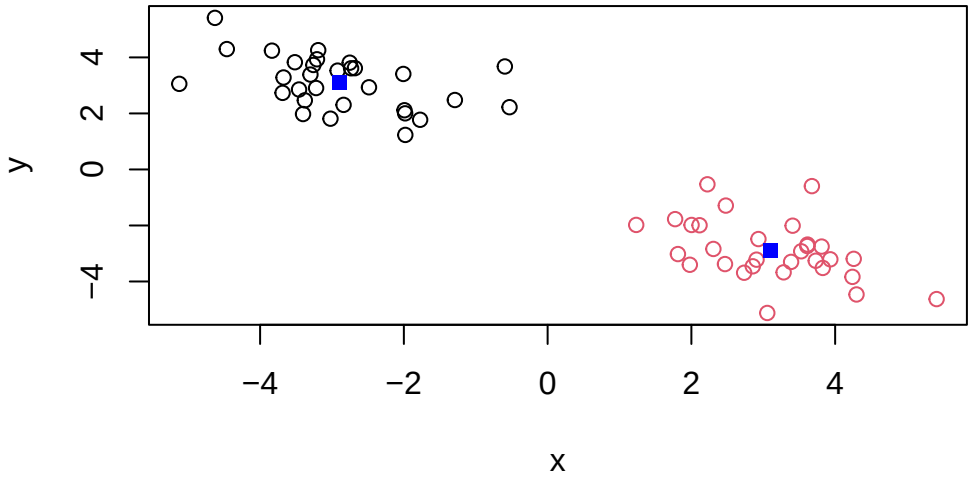
Q. What components of the results object details the cluster membership vector (i.e. our main result of which points lie in which cluster)?

```
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
```

Q. Plot our clustering results with points colored by cluster and also add the cluster centers as new points colored blue?

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



Q. Run `kmeans()` again and this time produce 4 clusters (and call your result object `k4`) and make a results figure like above?

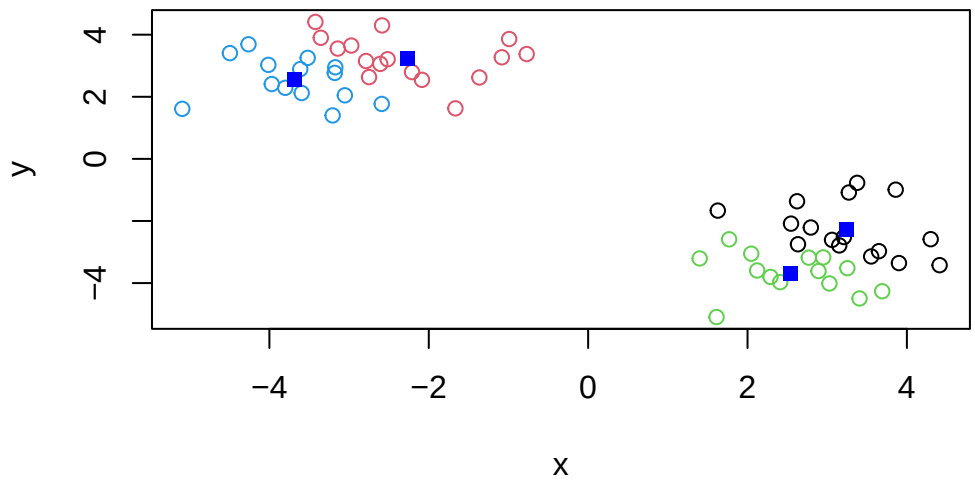
```
tmp <- c(rnorm(30, 3), rnorm(30, -3))
tmp
```

```
 [1]  1.6271412  3.1512364  3.3759280  2.1208613  3.2531435  3.8595764
 [7]  2.6331144  3.9001141  2.7690917  4.2985652  2.0469197  2.7947260
[13]  3.5519860  3.2712822  3.0606659  1.3996534  2.8906526  1.6113790
[19]  2.4100433  1.7693580  2.2886160  3.4056802  2.9495075  3.0279465
```

```
[25]  3.6905303  3.2095993  3.6490156  4.4105835  2.5458692  2.6216015
[31] -1.3652304 -2.0856477 -3.4229848 -2.9723223 -2.5164077 -4.2623923
[37] -4.0131036 -3.1729648 -4.4958000 -3.8012648 -2.5903803 -3.9702341
[43] -5.0933930 -3.6129621 -3.2067594 -2.6089682 -1.0833707 -3.1421123
[49] -2.2105648 -3.0531360 -2.5862919 -3.1797476 -3.3553524 -2.7497070
[55] -0.9914016 -3.5195475 -3.5949381 -0.7714653 -2.7873127 -1.6660227
```

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

k4 <- kmeans(x,4)
plot(x, col=k4$cluster)
points(k4$centers, col ="blue", pch=15)
```



The metric

```r
km$tot.withinss
```

```
[1] 116.3862
```

```r
k4$tot.withinss
```

```
[1] 61.3798
```

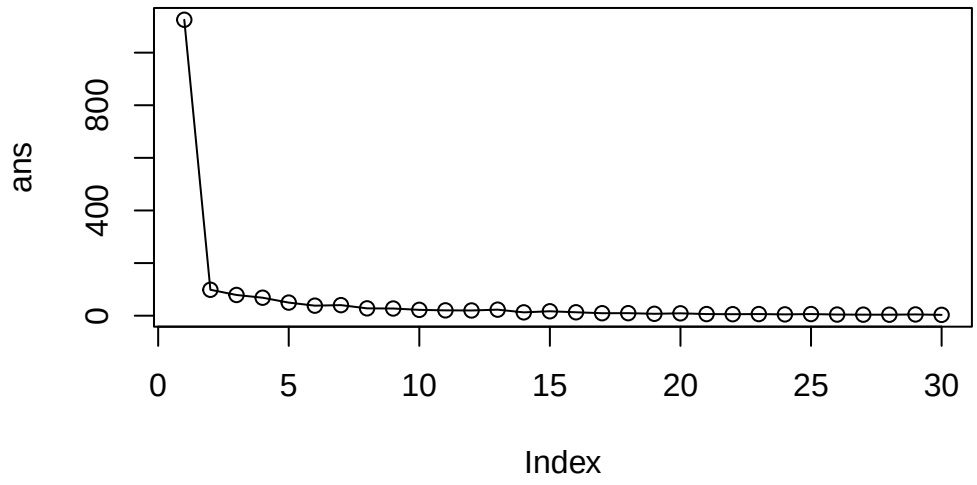
Q. Let's try different number of K(centers) from 1 to 30 and see what the best result is?

```
i <- 1
ans <- NULL
for (i in 1:30) {
ans <- c(ans, kmeans(x, centers = i)$tot.withinss)
}

ans
```

```
 [1] 1124.984125   98.587872   78.741858   68.342661   49.738623   38.097450
 [7]   40.156861   27.884147   27.305555   22.198904   20.303992   19.853151
[13]   22.934456   12.656174   16.817965   12.945824    9.419698    9.633449
[19]    7.184957    9.014951    6.190677    5.708762    6.188585    4.991252
[25]    6.154052    4.322487    3.988995    3.781830    4.770160    3.120925
```

```
plot(ans, typ="o")
```

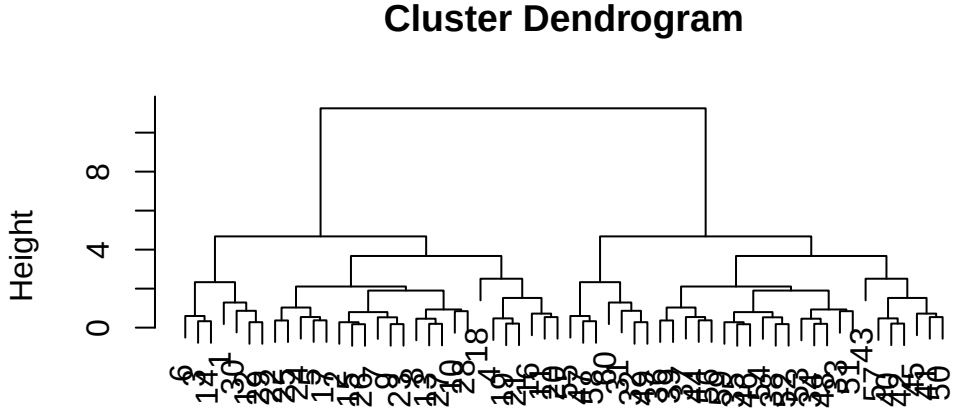


Key-point: K-means will impose a clustering structure on your data even if it is not there - it will always give you the answer you asked for even if that answer is silly!

Hierarchical Clustering

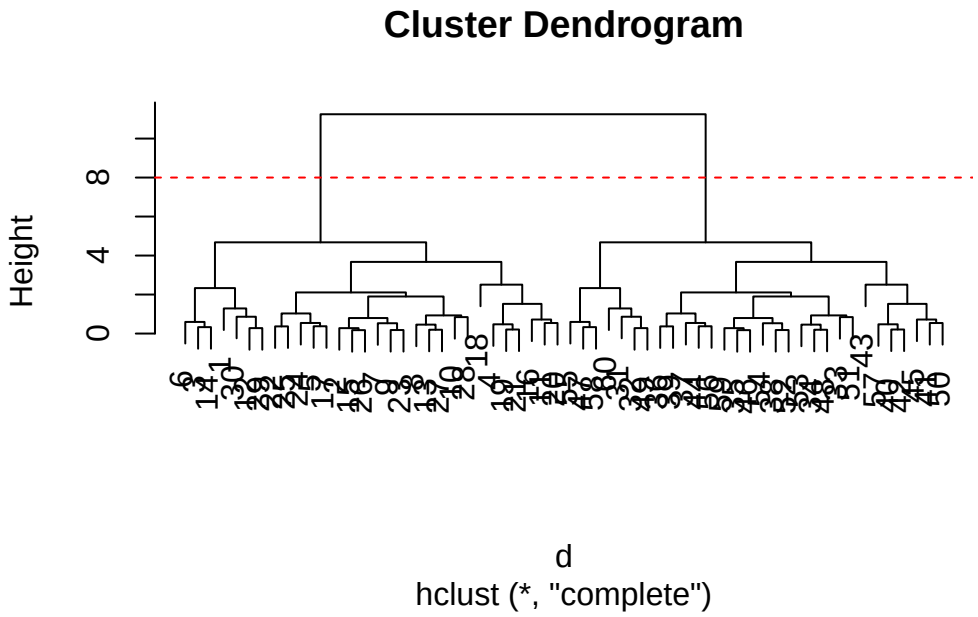
The main function for Hierarchical Clustering is called `hclust()`. Unlike `kmeans()` (which does all the work for you) you can't just pass `hclust()` our raw input data. It needs a "distance matrix" like the one returned from the `dist()` function.

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



To extract our cluster membership vector from a `hclust()`result object we have to "cut" our tree at a given hieght to yield separate "groups"/"branches".

```
plot(hc)
abline(h=8, col="red", lty=2)
```



To do this we use the `cutree()` function on our `hclust()` object:

```
grps <- cutree(hc, h=8)
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 2
```

should be two columns

```
tab <- table(grps, km$cluster)
tab[, c(2,1)]
```

```
grps  2  1
   1 30  0
   2  0 30
```

PCA of UK food data

Import the dataset of 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
```

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

There are 17 rows and 5 columns. I used the dim() function to find the answer for this.

```
dim(x)
```

```
[1] 17  5
```

One solution to set the row names is to do it by hand...

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

```
 [1] "Cheese"              "Carcass_meat "       "Other_meat "
 [4] "Fish"                "Fats_and_oils "      "Sugars"
 [7] "Fresh_potatoes "     "Fresh_Veg "          "Other_Veg "
[10] "Processed_potatoes " "Processed_Veg "      "Fresh_fruit "
[13] "Cereals "            "Beverages"           "Soft_drinks "
[16] "Alcoholic_drinks "   "Confectionery "
```

To remove the first column I can use the minus index trick

```
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 to set the row names to the first column with `read.csv()`

```
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

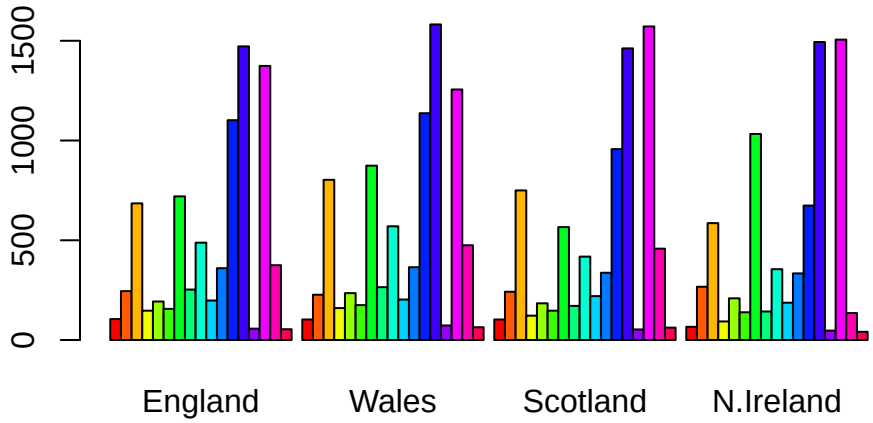
> Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

I prefer the second method more which has to do with the "read.csv()" function because that gives you a consistent answer. The first method we used does also give you the right answer but it can give you an inaccurate answer if you run it multiple times. The second method being "x <- x[,-1]".

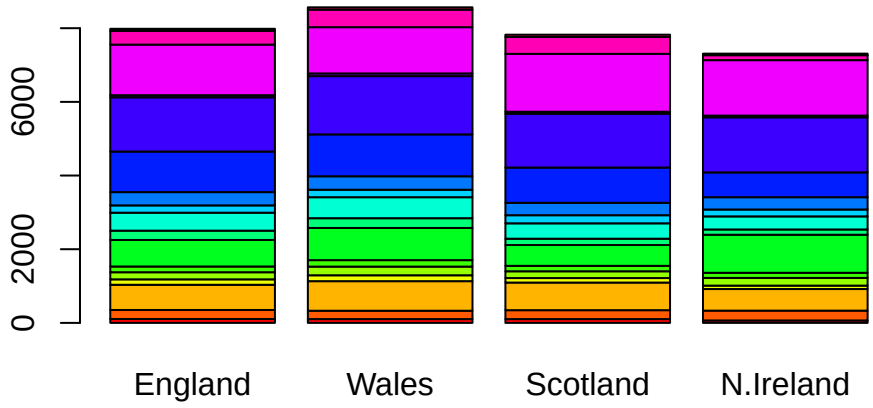
Spotting major differences and trends

Is difficult even in this wee 17D dataset...

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




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

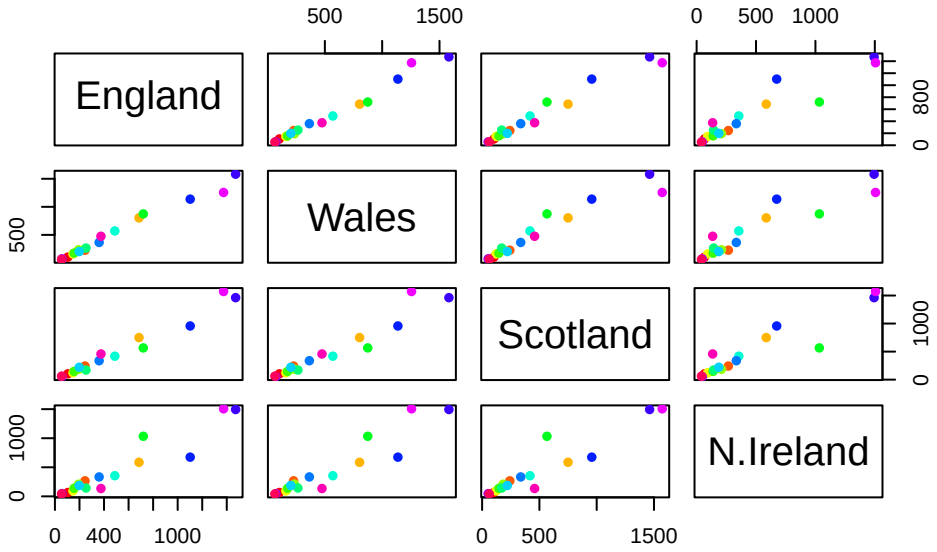


Q3: Changing what optional argument in the above barplot() function results in the following plot?

setting the "beside" equal to FALSE

Pairs plots and heatmaps

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



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?

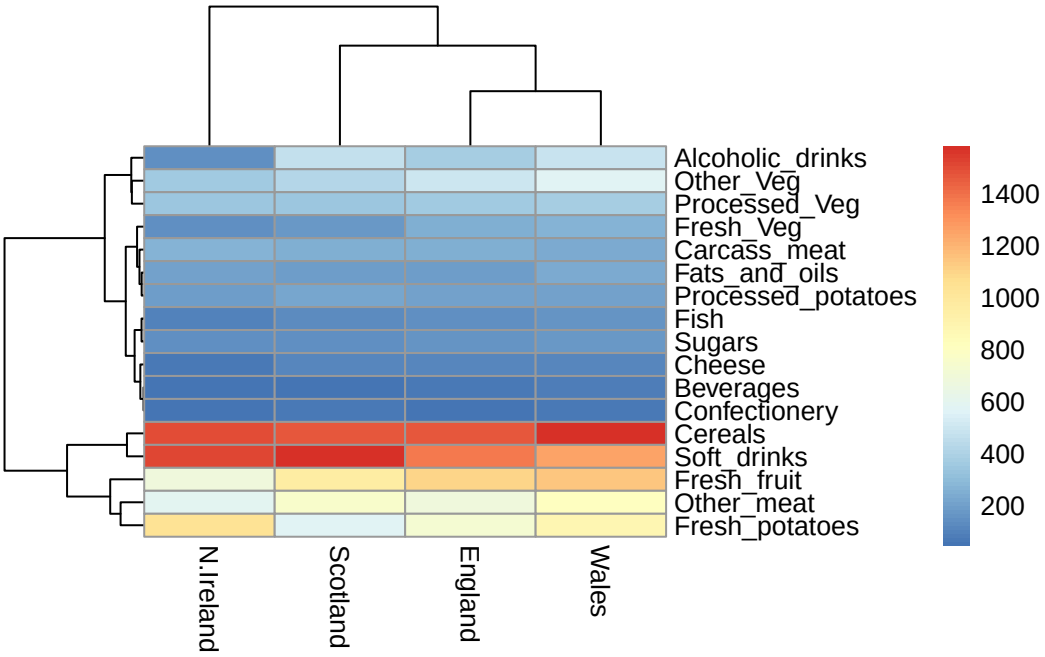
The x axis and y axis represents the countries. Each small panel represents two countries simultaneously. Each dot represents a data point in regards to food consumption. Most of the panels appear to have a positive correlation since there seems to be a trend in which they are increasing together. In regards to the first row, the y-axis represents the England population and the x-axis represents the country that is right below it. So for example, the first small panel to the right of England is comparing data of Wales versus England. The rest of the panels on this visual follows a similar design. The diagonal panels are the country plotted against itself and so that is why those boxes show the country name instead of the data as the data would just be a straight line. If a point lies on the diagonal for a given plot, this means that the two countries being compared have the same value for that observation. But if the data point is above the diagonal, this means that the country on the y-axis has a larger value.

And if the data point is below the diagonal, this means that the country on the x-axis has a larger value.

```
library(pheatmap)
```

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

```
pheatmap( as.matrix(x) )
```



Q6. Based on the pairs and heatmap figures, which countries cluster together and what does this suggest about their food consumption patterns? Can you easily tell what the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

The countries that cluster together are England, Wales, and Scotland. This means that they have very similar food consumption patterns across the categories in the dataset. The main differences between Norther Ireland and the other countries of the UK are that Northern Ireland has a more distinct cluster in comparison to the other three which means that its overall consumption differs systematically rather than by random variation. Northern Ireland has a more consistent food consumption pattern in comparison to the other countries and so this is why they have a more distinct cluster.

PCA to the rescue

The main PCA function in "base R" is called `prcomp()`. This function wants the transpose of our food data as input (i.e. the foods as columns and the countries as rows).

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

```
Standard deviations (1, .., p=4):
[1] 3.241502e+02 2.127478e+02 7.387622e+01 3.175833e-14

Rotation (n x k) = (17 x 4):
                            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
```

```
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
```

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

To make one of main PCA result figures we turn to `pca$x` the scores along our new PCs. This is called "PC plot" or "score plot" or "Ordination plot" ...

```
pca$x
```

```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -4.894696e-14
Wales     -240.52915 -224.646925 -56.475555  5.700024e-13
Scotland   -91.86934  286.081786 -44.415495 -7.460785e-13
N.Ireland  477.39164  -58.901862  -4.877895  2.321303e-13
```

```
my_cols <- c("orange", "red", "blue", "darkgreen")
my_cols
```

```
[1] "orange"    "red"       "blue"      "darkgreen"
```

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

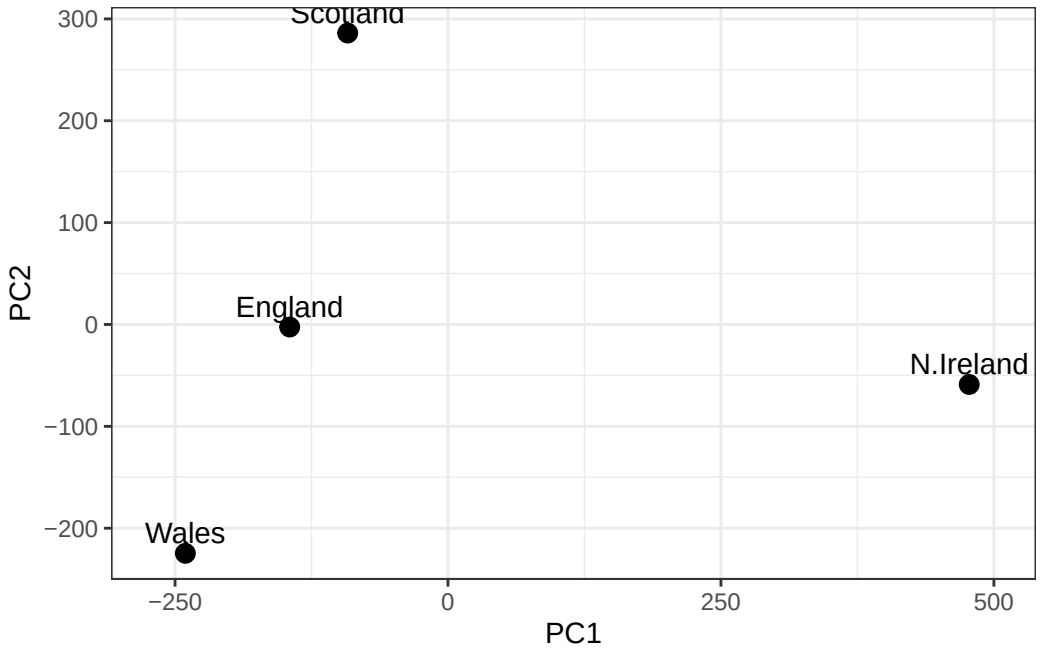
```
library (ggplot2)
```

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

```
# Create a data frame for plotting
df <- as.data.frame(pca$x)
df$Country <- rownames(df)

# Plot PC1 vs PC2 with ggplot
ggplot(pca$x)+
  aes(x = PC1, y = PC2, label = rownames(pca$x)) +
  geom_point(size = 3) +
```

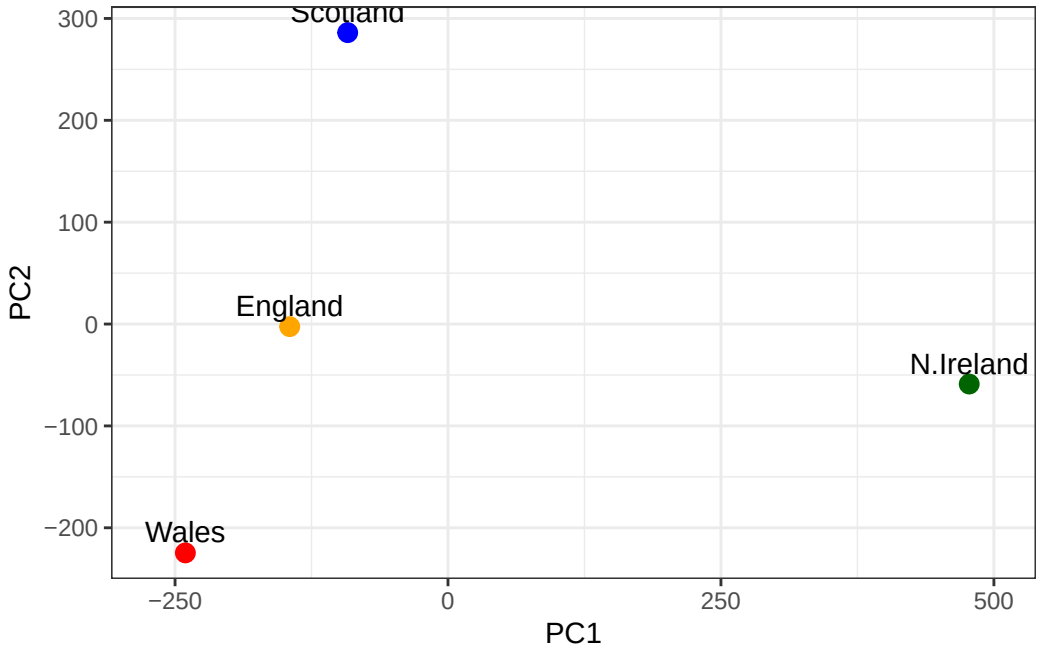
```
  geom_text(vjust = -0.5) +
  xlim(-270, 500) +
  xlab("PC1") +
  ylab("PC2") +
  theme_bw()
```



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

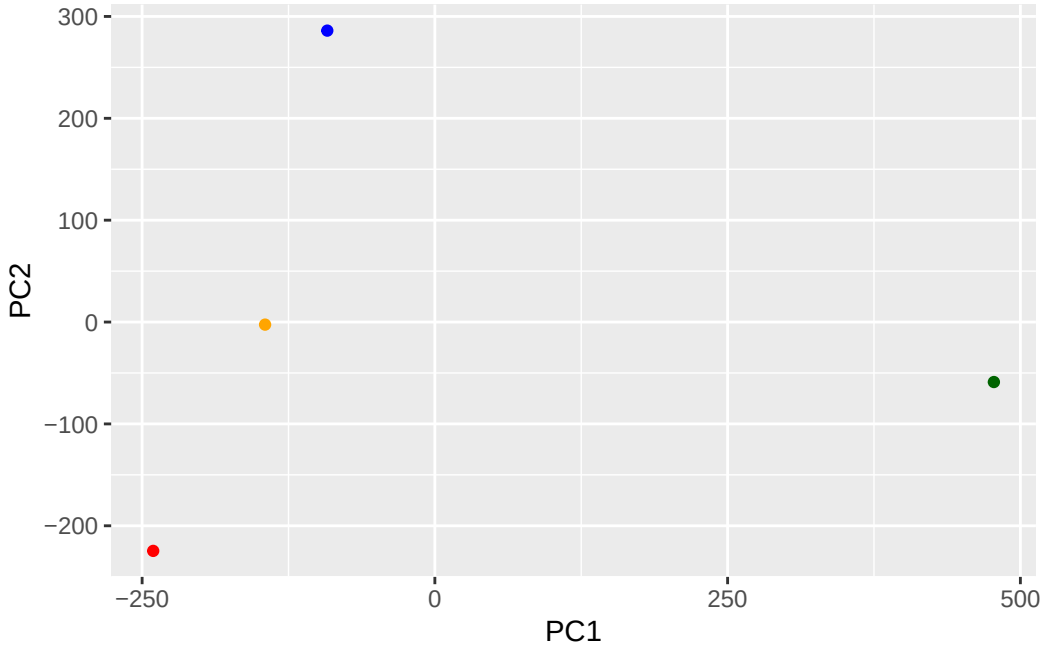
```
# Create a data frame for plotting
df <- as.data.frame(pca$x)
df$Country <- rownames(df)

# Plot PC1 vs PC2 with ggplot
ggplot(pca$x) +
  aes(x = PC1, y = PC2, label = rownames(pca$x)) +
  geom_point(size = 3, col = my_cols) +
  geom_text(vjust = -0.5) +
  xlim(-270, 500) +
  xlab("PC1") +
  ylab("PC2") +
  theme_bw()
```



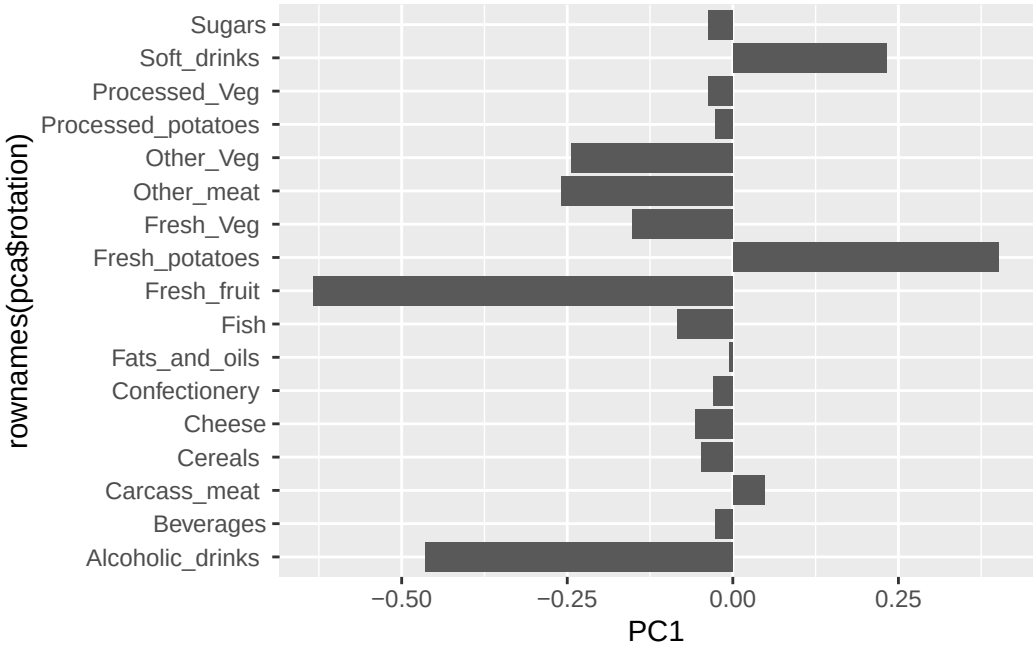

```
library (ggplot2)

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



The second major result figure is called a "loadings plot" of "variable contributions plot" or "weight plot"

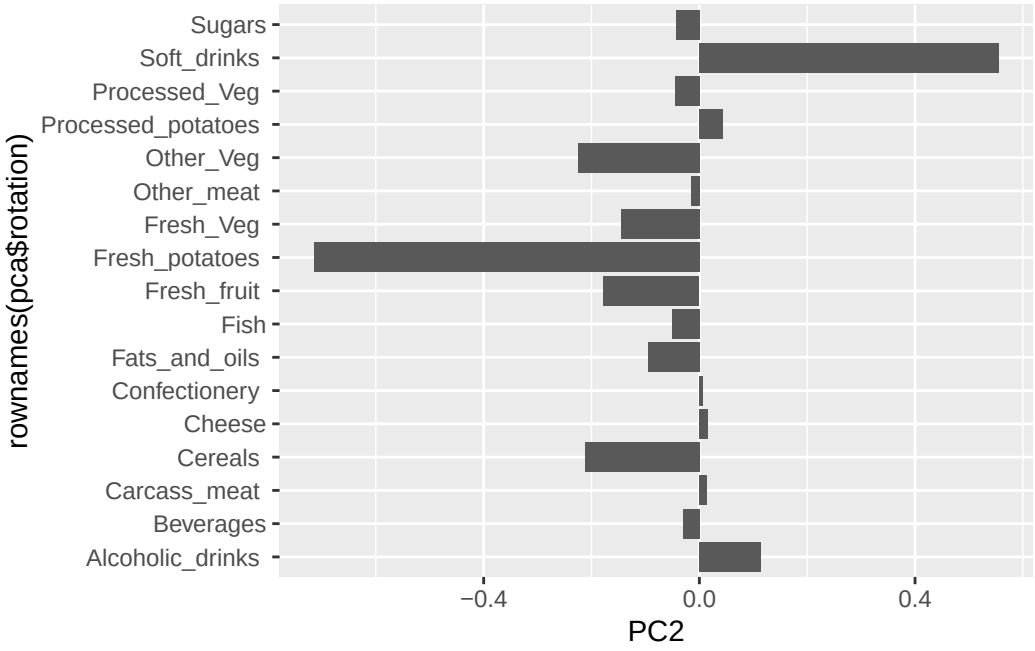
```
ggplot(pca$rotation) +
  aes(PC1, rownames(pca$rotation)) +
        geom_col()
```



Q9. Generate a similar `loadings plot` for PC2. What two food groups feature predominately and what does PC2 mainly tell us about?

The two food groups that feature predominately are soft drinks (strong positive loading) and fresh potatoes (negative loading). PC2 tells us that diets with more soft drinks have high PC2 values while diets with more fresh potatoes have low PC2 values. It is essentially comparing processed/sugary beverages vs. more traditional staple foods.

```
ggplot(pca$rotation) +
  aes(PC2, rownames(pca$rotation)) +
        geom_col()
```


rownames(pca$rotation)
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
−0.4
0.0
0.4
PC2