Class 7: Machine Learning 1

Neha Deshpande (PID: A17567541)

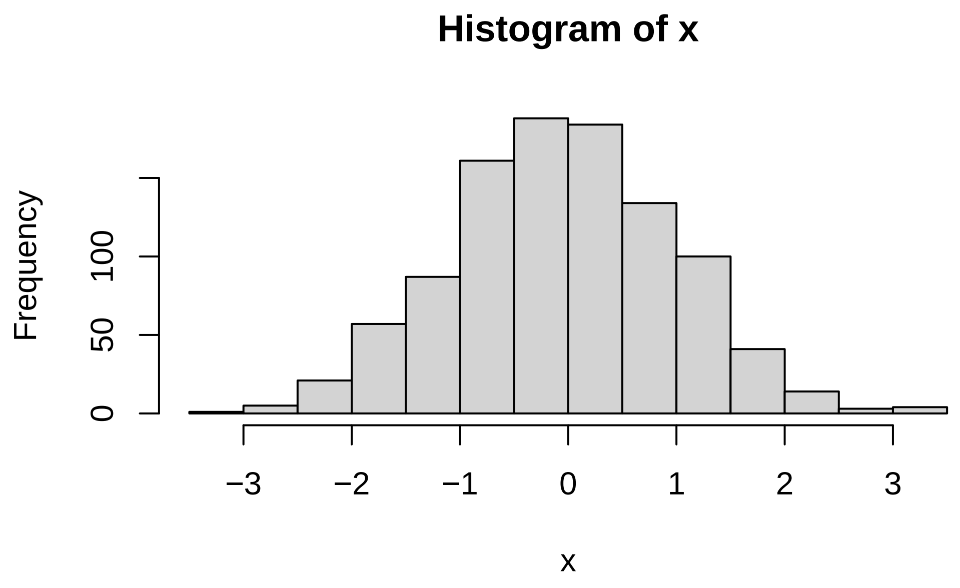
Clustering Methods

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

##Kmeans

First, lets make up some data (we know what the answer should be)

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



`rnorm()` makes up functions that are centered around the mean

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

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

```
 [1] -4.576049 -3.370114 -2.677610 -3.130324 -4.521176 -3.025933 -1.379772
 [8] -2.179554 -0.937749 -2.558719 -4.174534 -4.777958 -3.252388 -3.335312
[15] -1.253648 -4.068216 -2.016278 -3.729118 -1.228861 -2.411520 -2.626065
[22] -3.400340 -3.263280 -5.657453 -4.230287 -3.309544 -1.584433 -4.695490
[29] -3.783502 -2.133847  2.446586  3.666546  1.028455  3.505835  2.539684
[36]  2.074282  1.639406  1.167901  2.338831  3.899169  2.655990  2.895476
[43]  2.072174  1.193282  2.412571  1.468603  3.817459  3.195169  3.371968
[50]  3.991491  2.670344  1.374723  4.458681  1.952820  1.636139  1.393076
[57]  1.733518  2.529940  3.323006  2.108717
```

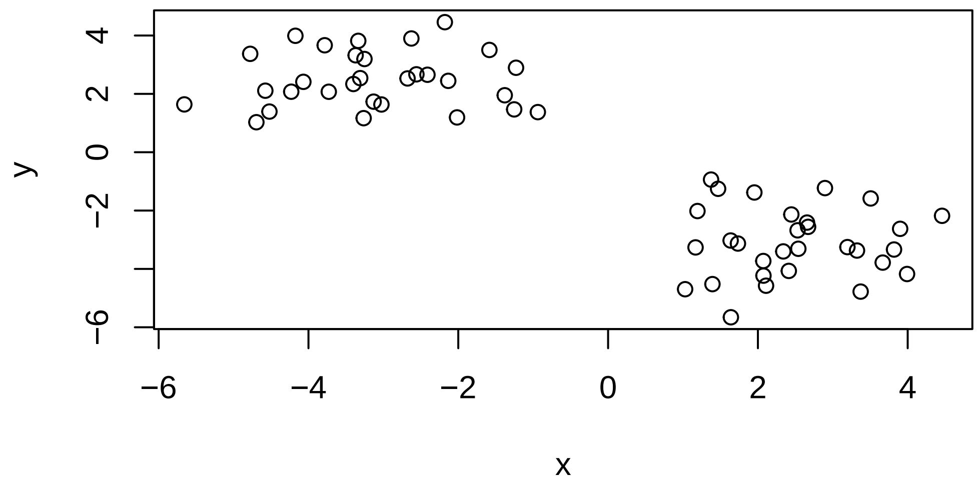
I will now make a wee x and y dataset with 2 groups of points. Use `rev()` function to reverse functions

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

```
              x        y
 [1,] -4.576049 2.108717
 [2,] -3.370114 3.323006
 [3,] -2.677610 2.529940
 [4,] -3.130324 1.733518
 [5,] -4.521176 1.393076
 [6,] -3.025933 1.636139
 [7,] -1.379772 1.952820
 [8,] -2.179554 4.458681
 [9,] -0.937749 1.374723
[10,] -2.558719 2.670344
[11,] -4.174534 3.991491
[12,] -4.777958 3.371968
[13,] -3.252388 3.195169
[14,] -3.335312 3.817459
[15,] -1.253648 1.468603
[16,] -4.068216 2.412571
[17,] -2.016278 1.193282
[18,] -3.729118 2.072174
[19,] -1.228861 2.895476
```

```
[20,] -2.411520  2.655990
[21,] -2.626065  3.899169
[22,] -3.400340  2.338831
[23,] -3.263280  1.167901
[24,] -5.657453  1.639406
[25,] -4.230287  2.074282
[26,] -3.309544  2.539684
[27,] -1.584433  3.505835
[28,] -4.695490  1.028455
[29,] -3.783502  3.666546
[30,] -2.133847  2.446586
[31,]  2.446586 -2.133847
[32,]  3.666546 -3.783502
[33,]  1.028455 -4.695490
[34,]  3.505835 -1.584433
[35,]  2.539684 -3.309544
[36,]  2.074282 -4.230287
[37,]  1.639406 -5.657453
[38,]  1.167901 -3.263280
[39,]  2.338831 -3.400340
[40,]  3.899169 -2.626065
[41,]  2.655990 -2.411520
[42,]  2.895476 -1.228861
[43,]  2.072174 -3.729118
[44,]  1.193282 -2.016278
[45,]  2.412571 -4.068216
[46,]  1.468603 -1.253648
[47,]  3.817459 -3.335312
[48,]  3.195169 -3.252388
[49,]  3.371968 -4.777958
[50,]  3.991491 -4.174534
[51,]  2.670344 -2.558719
[52,]  1.374723 -0.937749
[53,]  4.458681 -2.179554
[54,]  1.952820 -1.379772
[55,]  1.636139 -3.025933
[56,]  1.393076 -4.521176
[57,]  1.733518 -3.130324
[58,]  2.529940 -2.677610
[59,]  3.323006 -3.370114
[60,]  2.108717 -4.576049
```

```r
plot(x)
```



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

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

Cluster means:
          x         y
1 -3.109636  2.485395
2  2.485395 -3.109636

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 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

Within cluster sum of squares by cluster:
[1] 67.99947 67.99947
 (between_SS / total_SS =  87.4 %)

Available components:
```

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

Q. From your result object, how many points are in each cluster?

```r
k$size
```

```
[1] 30 30
```

Q. What “component” of your result object details the cluster membership

```r
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 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
```

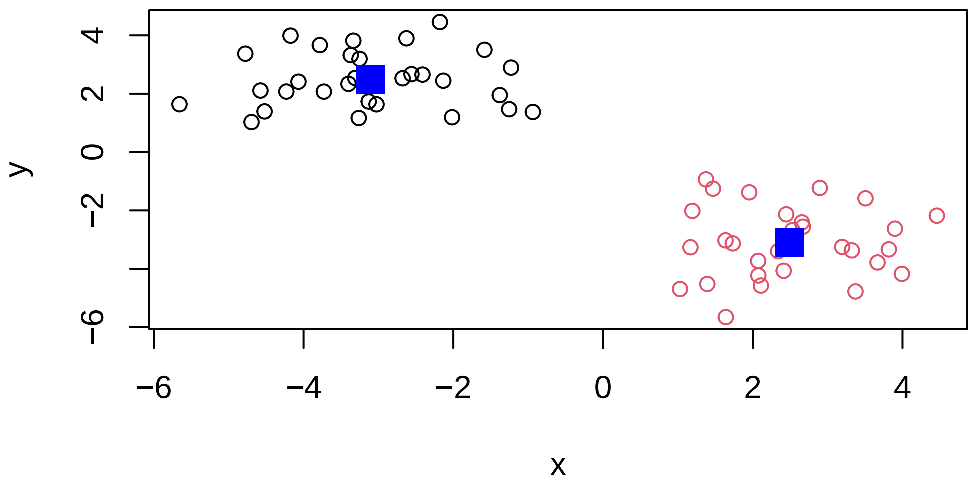
Q. Cluster centers?

```r
k$centers
```

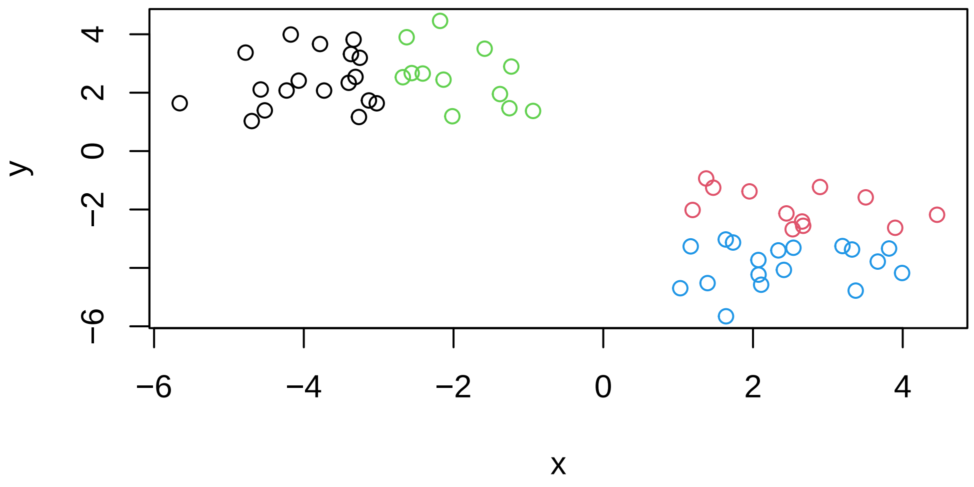
```
          x         y
1 -3.109636  2.485395
2  2.485395 -3.109636
```

Q. Plot of clustering results

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



```
k4<-kmeans(x, centers=4)
plot(x, col=k4$cluster)
```



A big limitation of kmmeans is that it does what you ask it to do even if you ask for silly clusters.

Hierarchical clustering

The main base R function for Hierarchical Clustering is `hclust()`. Unlike `kmeans()` you can just pass it to your data as input. You first need to calculate a distance matrix. It is a lot more flexible, because it allows you to represent a lot more different distances.

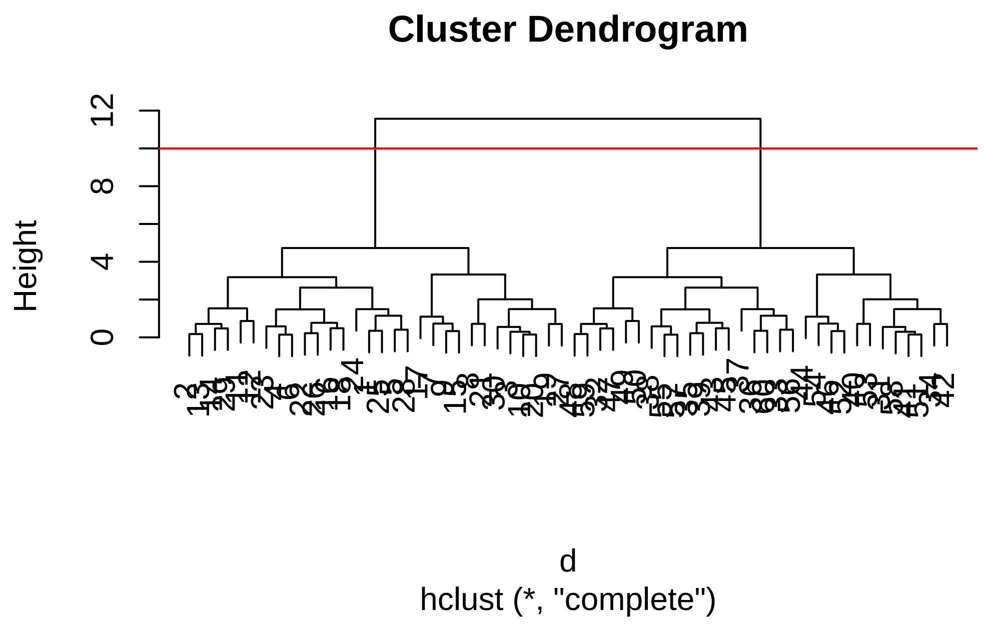
In kmeans, the data is divided into a predetermined number of clusters.

```r
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:

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



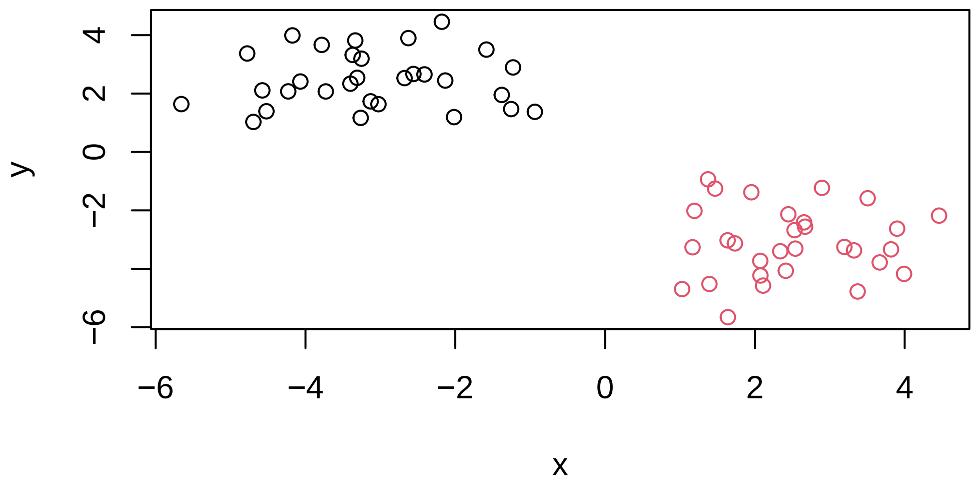
Depending on where you cut, it gives you the number of clusters. To make the cut and get our cluster membership vector we can use the `cutree()` function

```r
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 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
```

Make a plot of our data colored by hclust results

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



Principal Component Analysis (PCA)

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

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

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

> Q2. This is not a good approach to use, because the more times you run it, more columns are going to get deleted from the data set.
>
> Q1.

```r
dim(x)
```

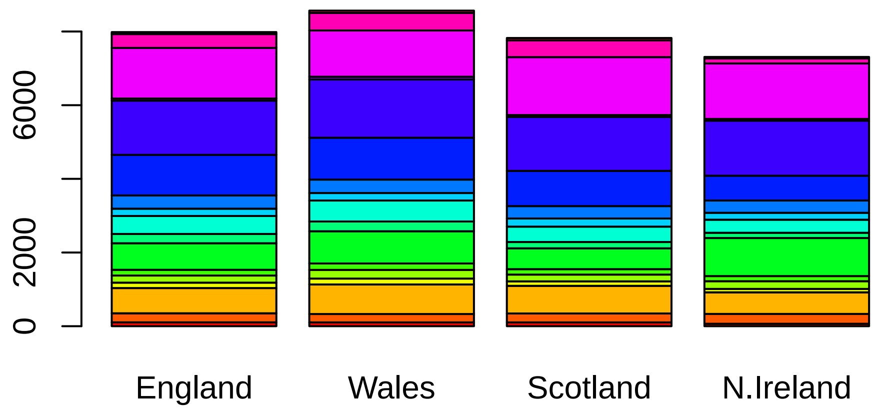
```
[1] 17  4
```

```
head(x)
```

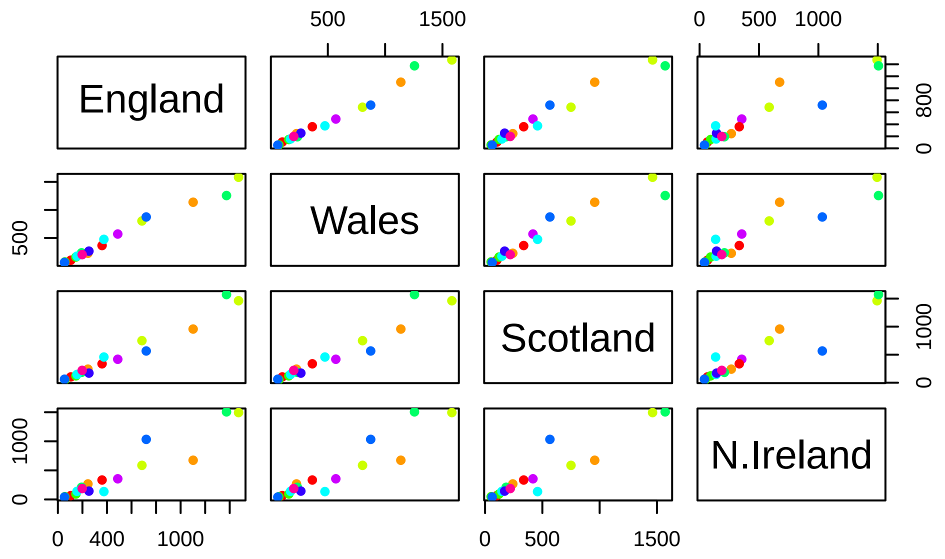
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars  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

Q3.

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



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



This plot is not great for many dimensions. This data set only has 17, but if they are a lot more, it would be a lot harder to analyze the data.

PCA to the rescue. The base function for this is `prcomp()`. Here we need to take the transpose of our input to get the food as columns and countries as rows. The summary tells you how well the PCA does at capturing the spread of the data.

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

PC1 is capturing 67% of the total spread over all the variables. But, PC1 against PC2 (using them as two dimensions for the new plot) covers 96.5% of the data. >Q. How much variance is captured in 2 PCs?

96.5%

To make our main "PC score plot" or "PC1 vs PC2 plot" or "ordination plot" or "PC plot"

```r
attributes(pca)
```

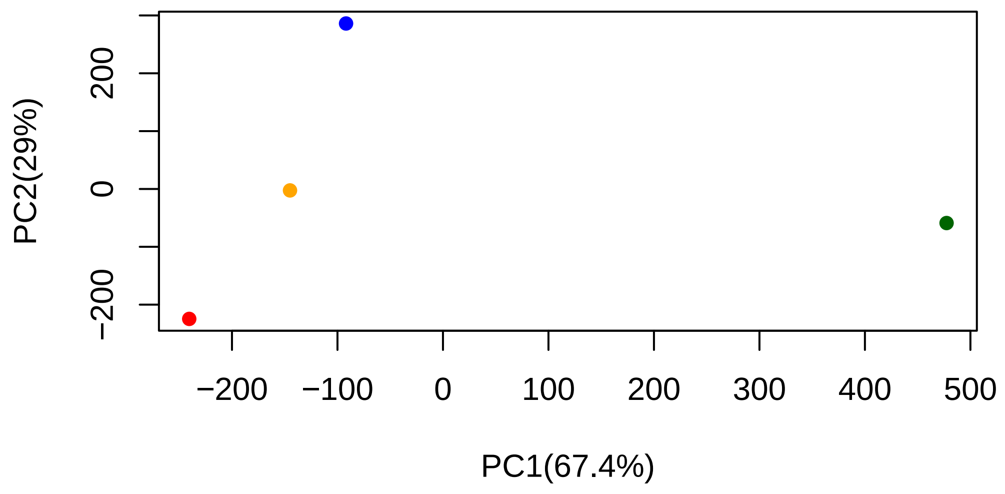
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"       

$class
[1] "prcomp"
```

We are after the **pca$x** component to make our main PCA plot

```r
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
```

```r
mycols<-c("orange","red","blue","darkgreen")
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 variables (the foods, in this case) contribute to the PCS.

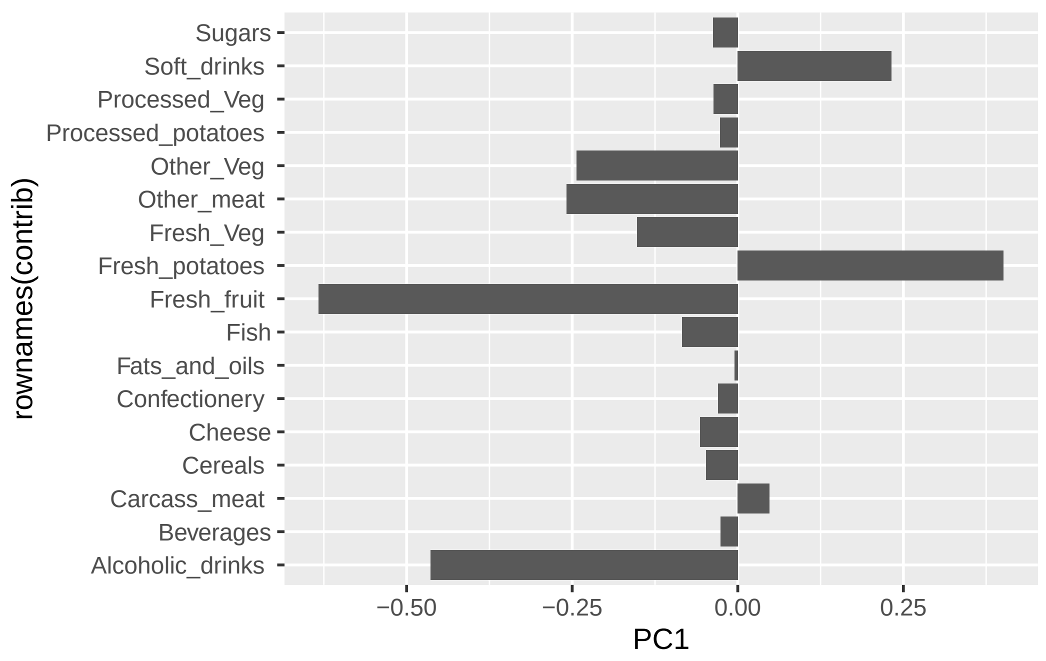
This is contained in the `pca$rotation` object- folks often call this the "loadings" or "contributions" to the PCs.

```
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()
```


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
rownames(contrib)
−0.50
−0.25
0.00
0.25
PC1