Class 7: Machine Learning 1

Alyssa Hayashi

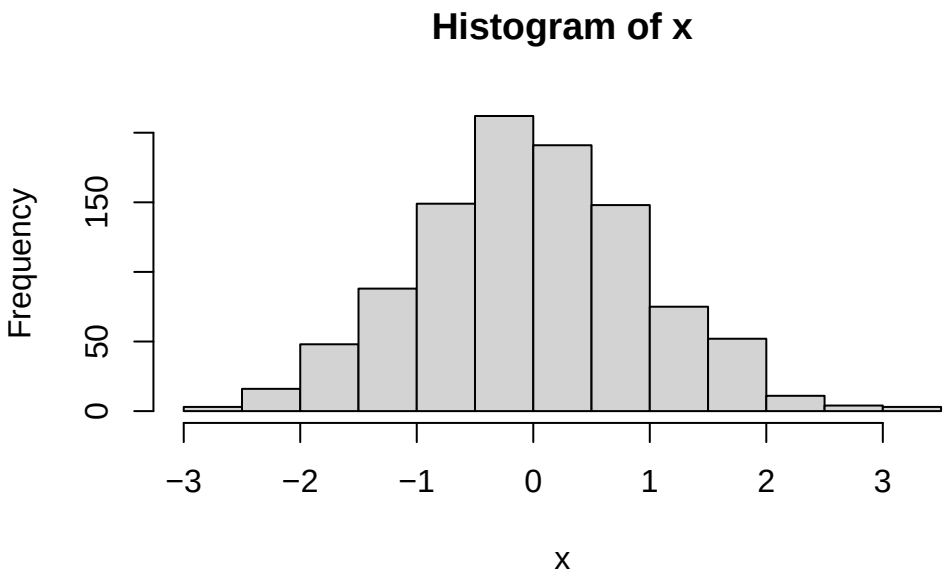
Clustering Methods

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

##Kmeans

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

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



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

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

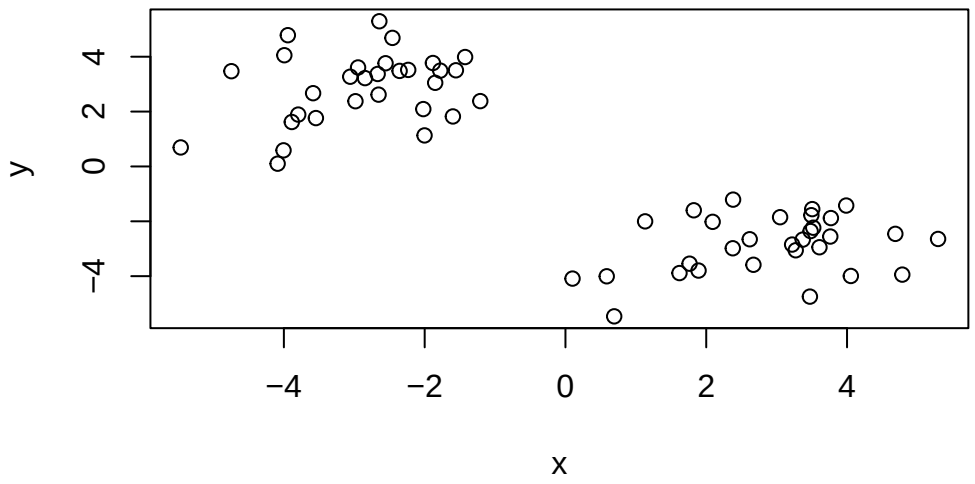
```
 [1] -5.4654287 -2.4578113 -3.0573653 -2.9463360 -2.5555743 -2.2337278
 [7] -1.7780594 -2.0196049 -1.5562221 -4.0051479 -2.6549887 -1.5989093
[13] -3.5450027 -2.9834527 -3.5847736 -4.0889276 -2.0017765 -1.4253807
[19] -1.8827668 -1.2100202 -2.6677473 -4.7456262 -3.9437298 -2.8478409
[25] -3.8875276 -2.6437114 -3.9936078 -1.8505998 -3.7962376 -2.3560163
[31]  3.4825200  1.8906814  3.0492871  4.0544682  5.2934483  1.6198860
[37]  3.2200539  4.7851298  3.4726593  3.3708642  2.3815219  3.7707897
[43]  3.9884467  1.1315131  0.1002699  2.6703867  2.3771791  1.7631722
[49]  1.8227790  2.6176273  0.5863878  3.5057416  2.0911169  3.4930918
[55]  3.5201021  3.7621265  3.6082200  3.2711775  4.6862477  0.6925739
```

I will now make a wee x and y dataset with 2 groups of points

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

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

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




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

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

Cluster means:
          x         y
1  2.869316 -2.859464
2 -2.859464  2.869316

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] 79.14885 79.14885
 (between_SS / total_SS =  86.1 %)

Available components:

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

Q. from your result object 'k' how many points are in each cluster?

```
k$size
```

```
[1] 30 30
```

Q. What "component" of your result object details the cluter membership.

```
k$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
```

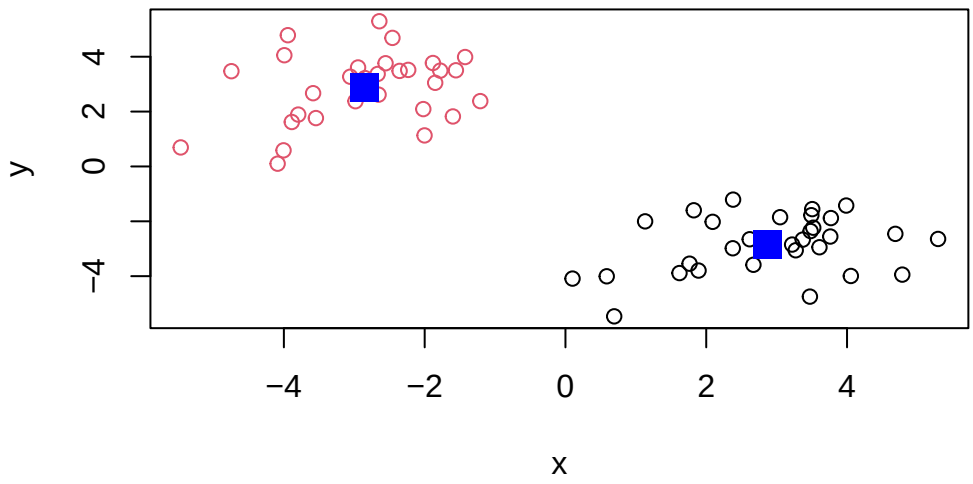
Cluster centers

```
k$centers
```

```
          x         y
1  2.869316 -2.859464
2 -2.859464  2.869316
```

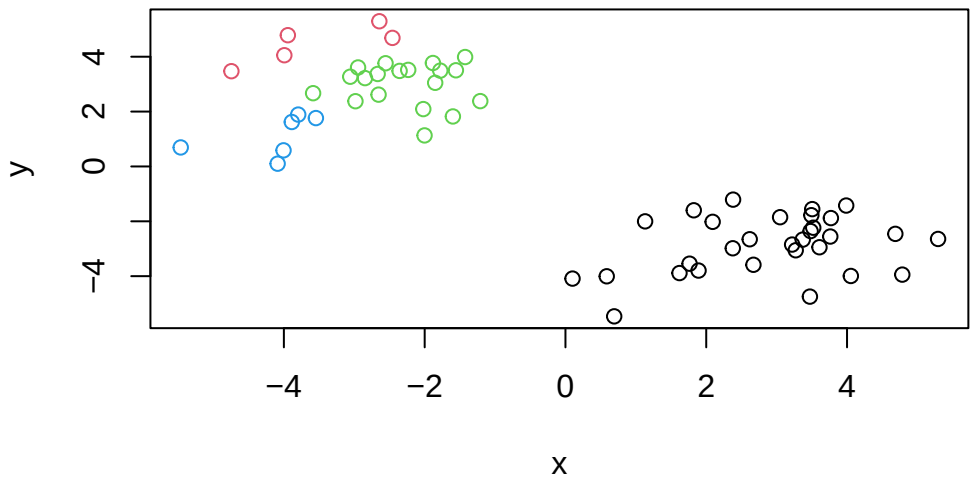
Plot of our clustering results

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



We can cluster into 4 groups

```
#kmeans
k4 <- kmeans(x, center=4)
#plot results
plot(x, col=k4$cluster)
```



limitation of kmeans is that it does what you ask even if you ask silly clusters

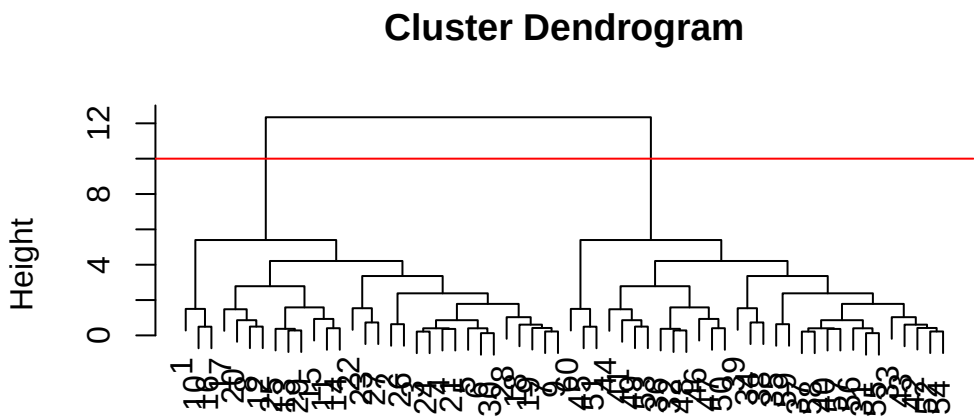
Hierarchical Clustering

The maain base R function for Hierarchical Clustering is 'hclust'. Unlike 'kmeans()' you can not just pass it your data as input.. You first need to calculate a distance matrix.

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

use `plot()` to view results

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



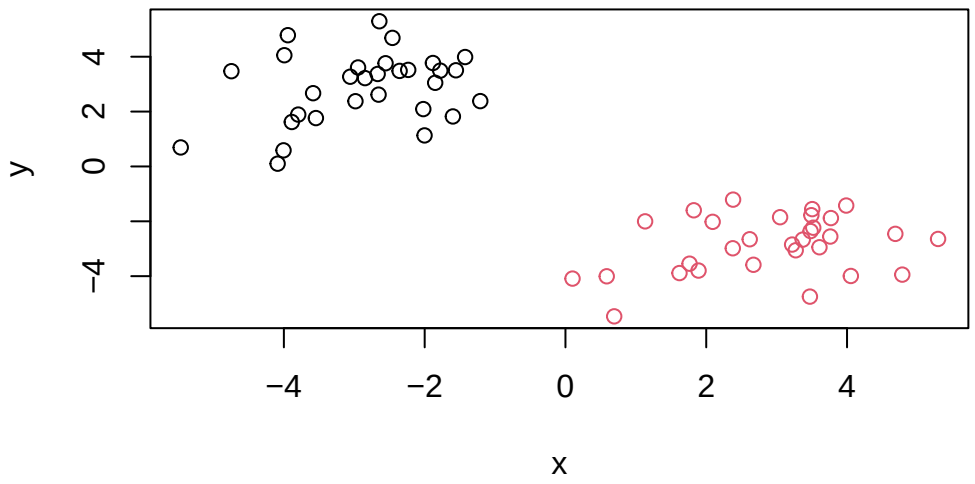
Make the cut and get our clusster membership vector we can use the cutree() function

```
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 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)
```



Principle Component Analysis

Here we will do Principal Component Analysis (PCA) on food data from the uk

> 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?

USing the row.names=1 approach is preferred since the second one results in a row being deleted and if ran multiple times rows that are desired will get deleted.

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

```
#rownaames(x) x[,1]
#x<- x[,-1]
#x
```

> 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)
```

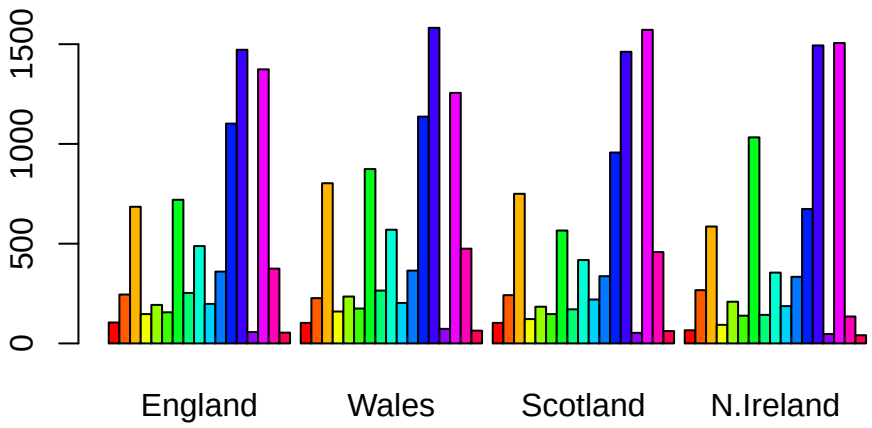
```
[1] 17
```

```
ncol(x)
```

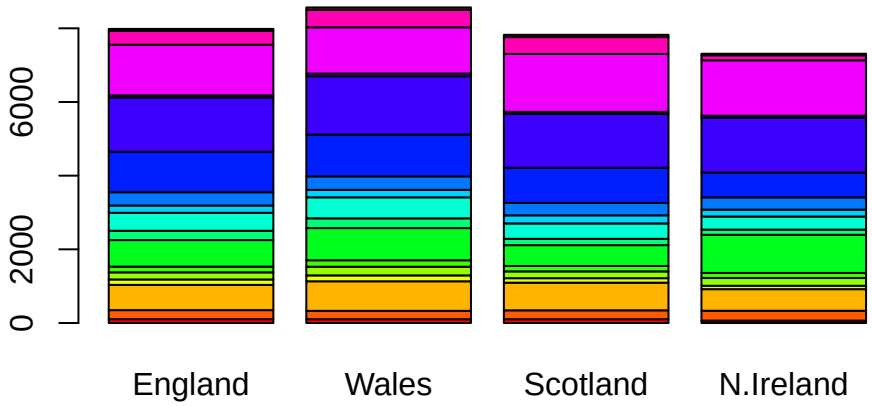
```
[1] 4
```

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)))
```



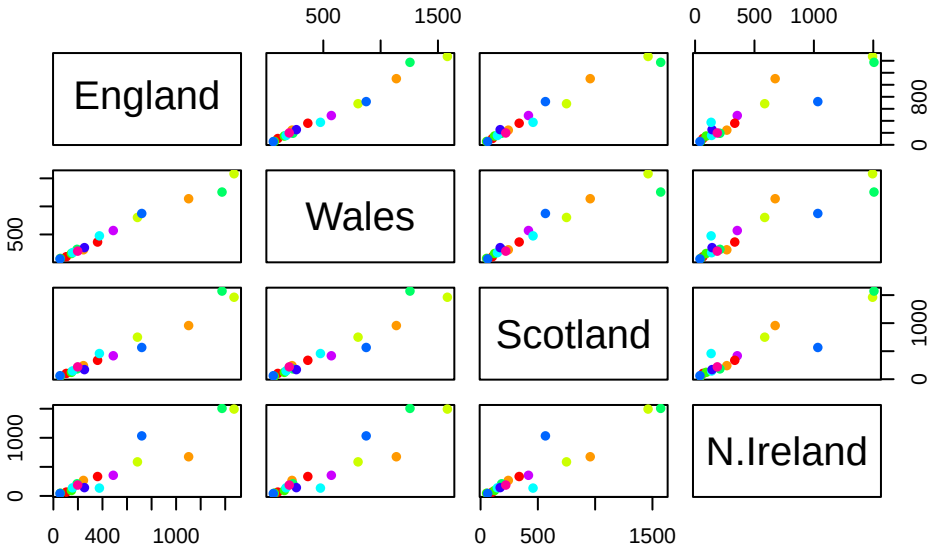

```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```




```
#changing the beside argument from T to F causes the bar plot to a stacked bar plot.
```

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)
```



PCA to the rescue

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

```
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
```

Q. How much varriance is captured in 2 pcs

96.5%

To make or main "PC score plot" or "PC1 vs PC2 plot" or "PC plot" or "ordinated plot"

```
attributes(pca)
```

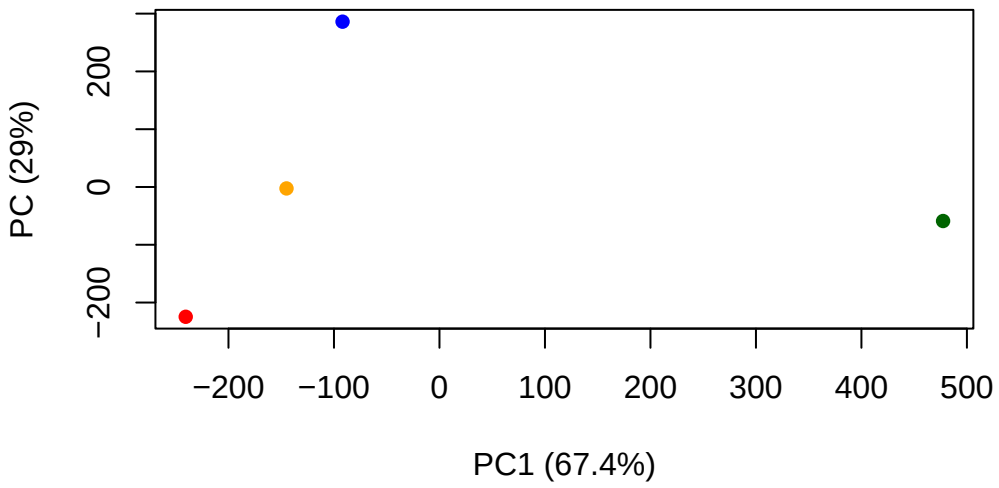
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

We are after the `pca$x` result component to make. or man PCA 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
```

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



Another important result from pca is how the original variables (in this case the food) contributes to the pca

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

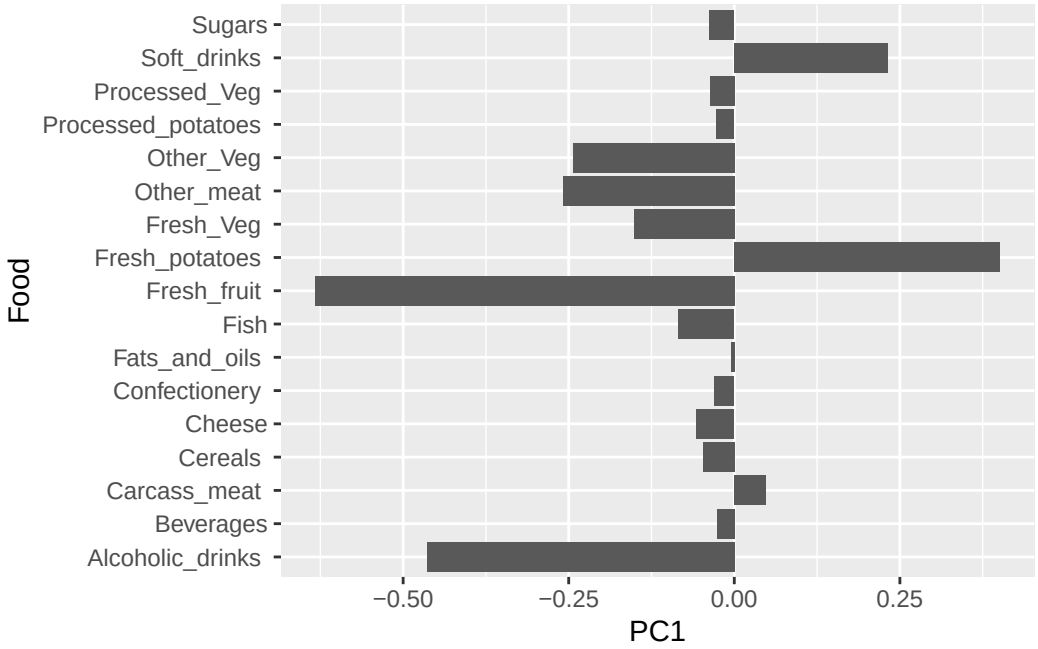
```
pca$rotation
```

	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)
contributions<- as.data.frame(pca$rotation)
ggplot(contributions)
```

```r
ld <- as.data.frame(pca$rotation)
ld_lab <- tibble::rownames_to_column(ld, "Food")

ggplot(ld_lab) +
  aes(PC1, Food) +
  geom_col()
```


Food
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