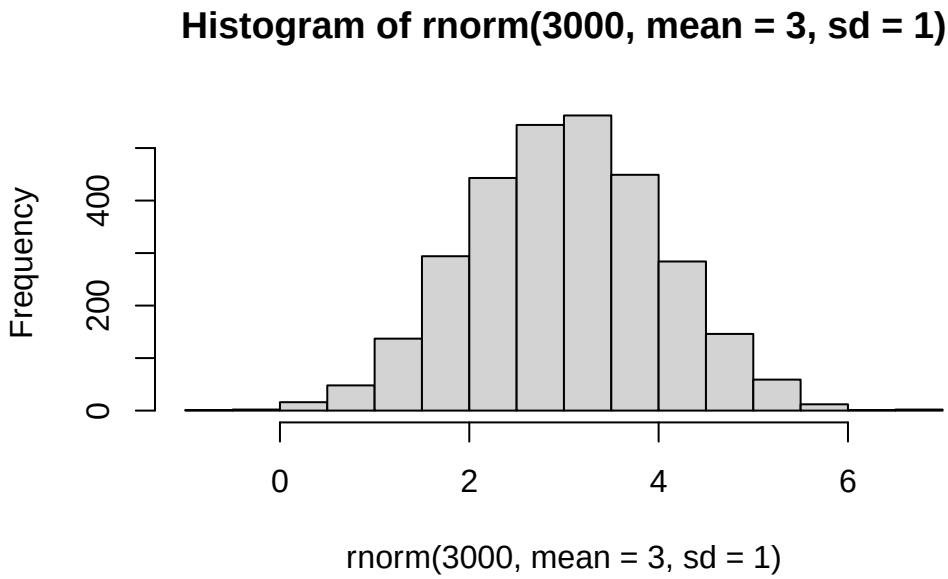
Class07Lab: Machine Learning 1

Nancy Leon-Rivera

Today we will delve into unsupervised machine learning with a initial focus on clustering and dimensionality reduction.

Make up 2 clusters of data point with known results The `rnorm()` function will help us here...

```
hist(rnorm(3000, mean =3, sd = 1))
```



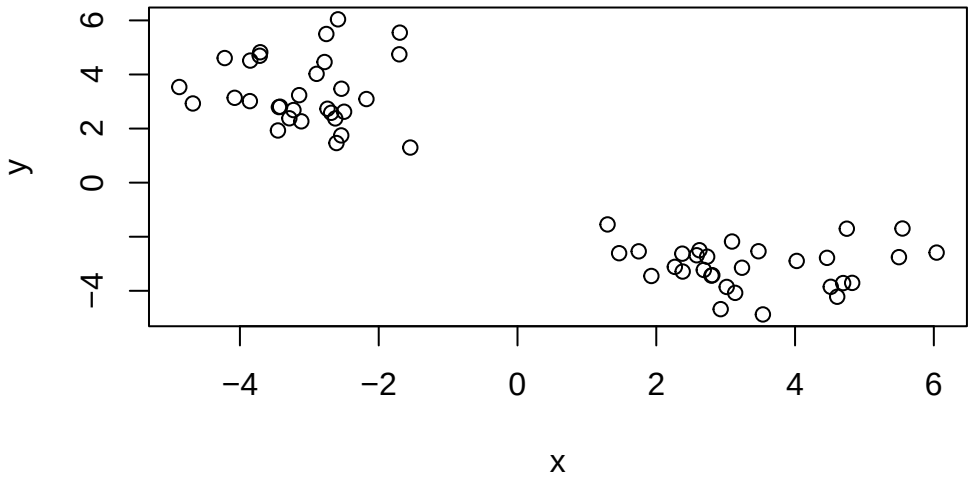
Lets get some data centered at x3,-3, y-3,3

```
#c(, , ) how you add things together in a vector
#Combine 30+3 values with 30-3 calues
x <- c(rnorm(30, mean =3, sd = 1), rnorm(30, mean =-3, sd = 1))
#bind these values together
```

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

```
            x         y
[1,] 6.040205 -2.586698
[2,] 2.807586 -3.421572
[3,] 4.023633 -2.895305
[4,] 1.464785 -2.608920
[5,] 3.472883 -2.537227
[6,] 4.745852 -1.702934
```

```r
plot(z)
```



##K means Now we can see how K-means cluster's this data. The main function for K-means clustering in "base R" is called `kmeans()`

```r
km <- kmeans(z, centers = 2)
km
```

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

```
Cluster means:
          x         y
1 -3.081420  3.368609
2  3.368609 -3.081420

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] 65.23347 65.23347
 (between_SS / total_SS =  90.5 %)

Available components:

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

```
#clustering total was 60 si it divides it by 2
```

```
attributes(km)
```

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

$class
[1] "kmeans"
```

Q. what size is each cluster? size

```
km$size
```

```
[1] 30 30
```

Q. The cluster membership vector (i.e. the answer: cluster to which each point is allocated)

```
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
```

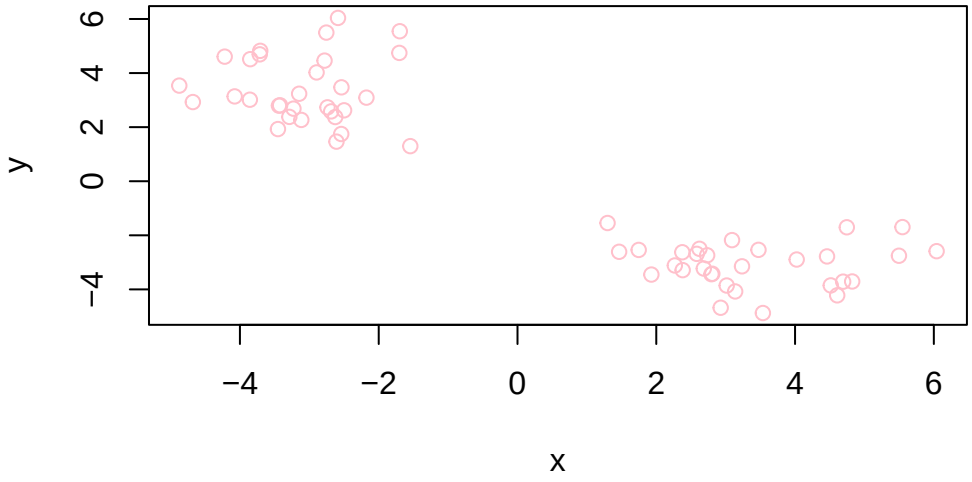
Q. Clusters centers

```
km$centers
```

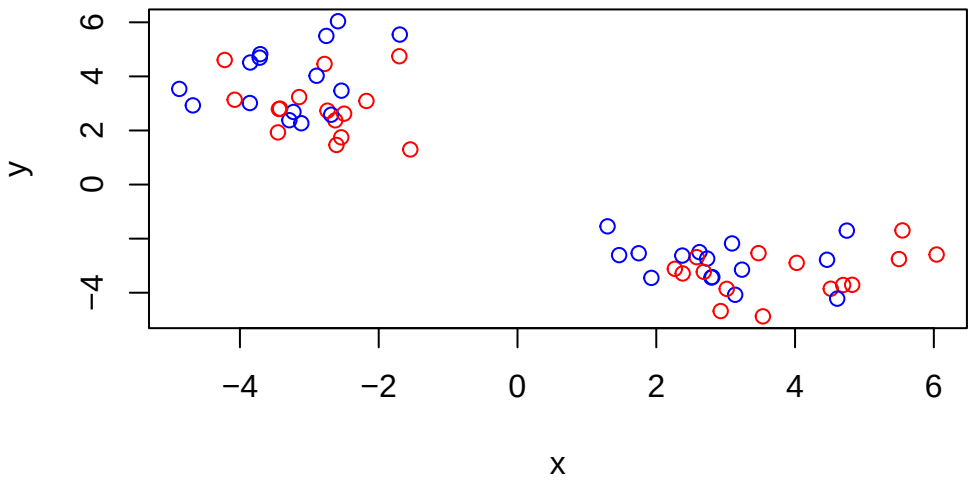
```
          x         y
1 -3.081420  3.368609
2  3.368609 -3.081420
```

Q. Make a results figure, plot data **z** colored by cluster membership and show the cluster centers.

```
plot(z, col="pink")
```

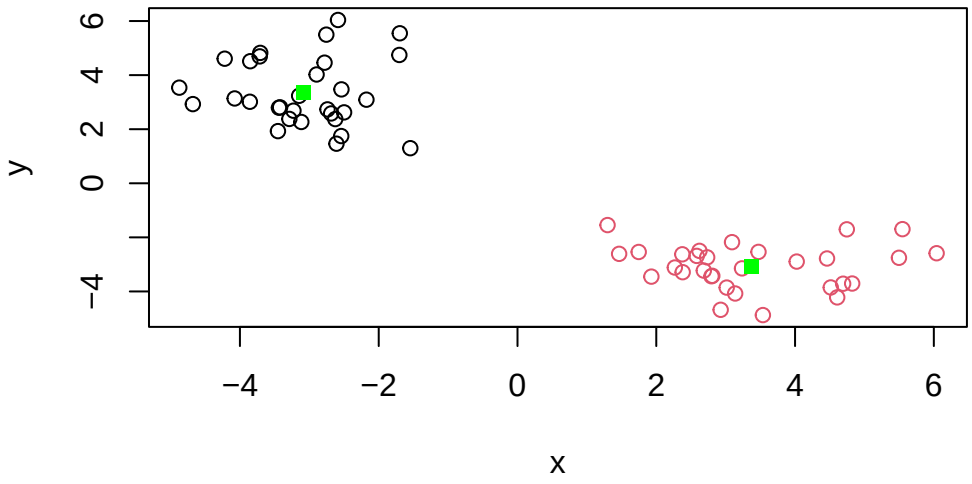



```
plot(z, col=c("red", "blue"))
```



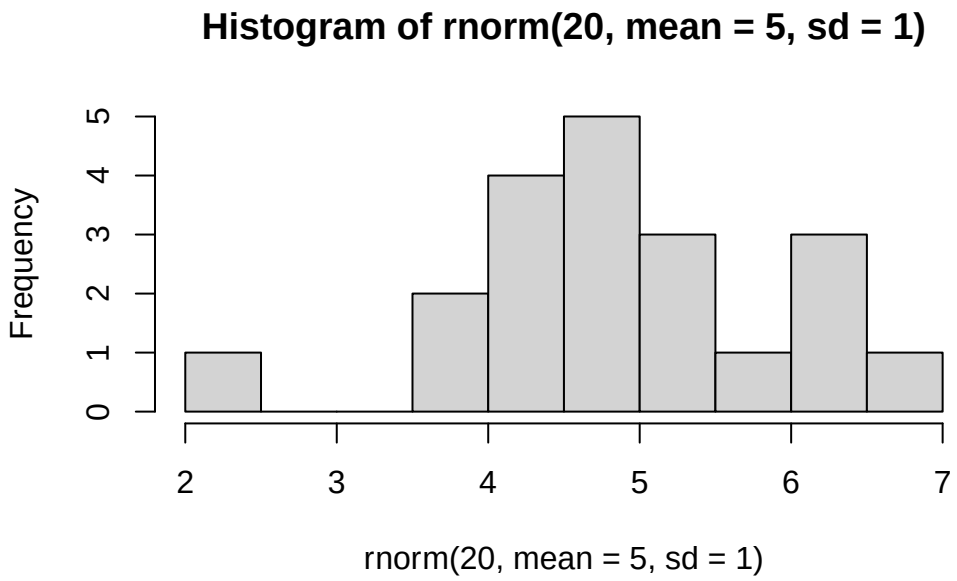
You can specify color based on a number, where 1 is black, 2 is red…

```
plot(z, col=km$cluster)
points(km$centers, col="green", pch=15)
```



Q. Re-run your K-means clustering and as for 4 clusters and plot the results as above

```
hist(rnorm(20, mean =5, sd = 1))
```



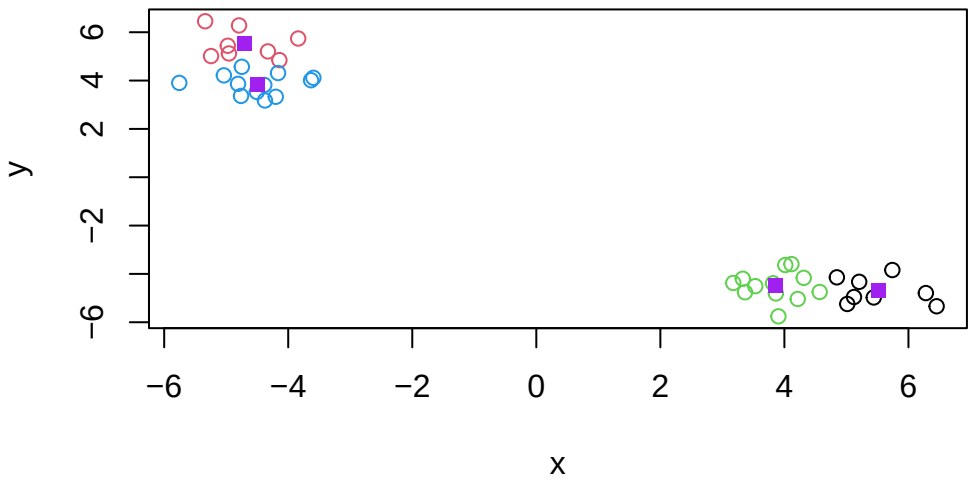

```
x <- c(rnorm(20, mean =5, sd = 1), rnorm(20, mean =-5, sd = 1))
#bind these values together
y <- cbind(x=x, y=rev(x))
head(y)
```

```
            x         y
[1,] 3.817075 -4.386371
[2,] 6.454590 -5.339043
[3,] 3.863172 -4.809098
[4,] 5.013769 -5.244476
[5,] 5.440645 -4.974968
[6,] 4.569118 -4.748031
```

```
km2 <- kmeans(y, centers = 4)
km2
```

```
K-means clustering with 4 clusters of sizes 8, 8, 12, 12

Cluster means:
          x         y
1  5.513034 -4.702094
2 -4.702094  5.513034
3  3.850701 -4.497637
4 -4.497637  3.850701

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

Within cluster sum of squares by cluster:
[1] 4.519741 4.519741 5.923578 5.923578
 (between_SS / total_SS =  98.8 %)

Available components:

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

```r
plot(y, col=km2$cluster)
points(km2$centers, col="purple", pch=15)
```

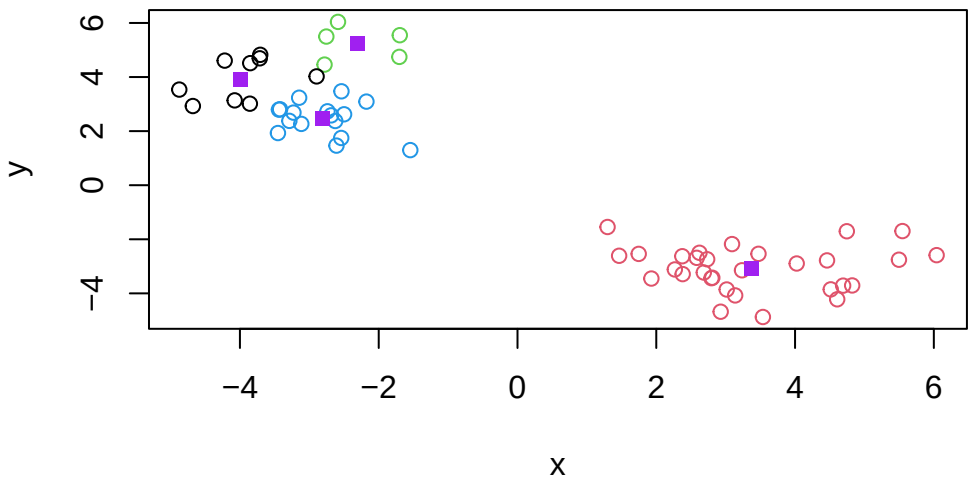



```
km4 <- kmeans(z, centers=4)
km4
```

```
K-means clustering with 4 clusters of sizes 9, 30, 5, 16

Cluster means:
          x         y
1 -3.985946  3.919798
2  3.368609 -3.081420
3 -2.304273  5.258391
4 -2.815482  2.468008

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 2 4 1 4 1 4 4 4 4
[39] 4 4 4 1 1 1 4 4 3 1 1 3 4 1 4 3 3 4 4 1 4 3

Within cluster sum of squares by cluster:
[1]  7.520329 65.233466  2.892346  9.737955
 (between_SS / total_SS =  93.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
```

```
[6] "betweenss"    "size"         "iter"         "ifault"
```

```
plot(z, col=km4$cluster)
points(km4$centers, col="purple", pch=15)
```



##Hierarachial clustering

The main "base R" function for this is `hclust()`. Unlike `kmeans()` you can't just give your dataset an input, you need to provide a distance matrix.

We can use the `dist()` function fo this

```
#dim(z) give you num of columns and rows
#dist creates a distance matrix
d <-  dist(z)
hc <- hclust(d)
hc
```

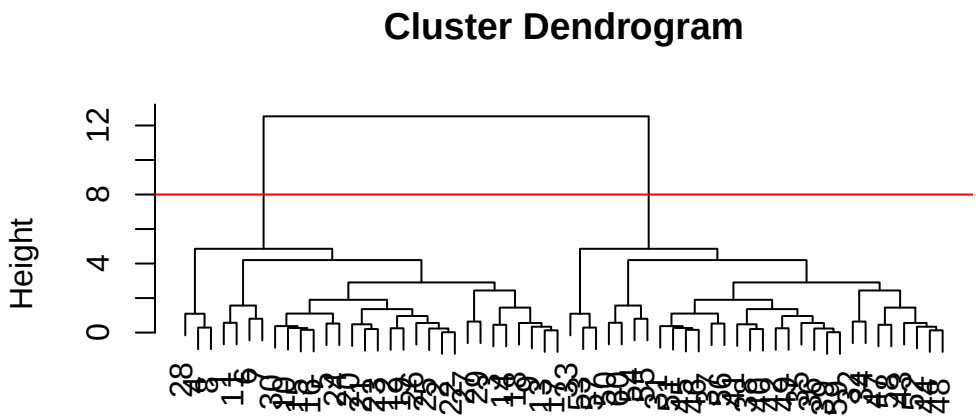
```
Call:
hclust(d = d)

Cluster method   : complete
```

```
Distance         : euclidean
Number of objects: 60
```

```
#hclust()
```

There is a custom plot() for hclust object, lets see it.

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

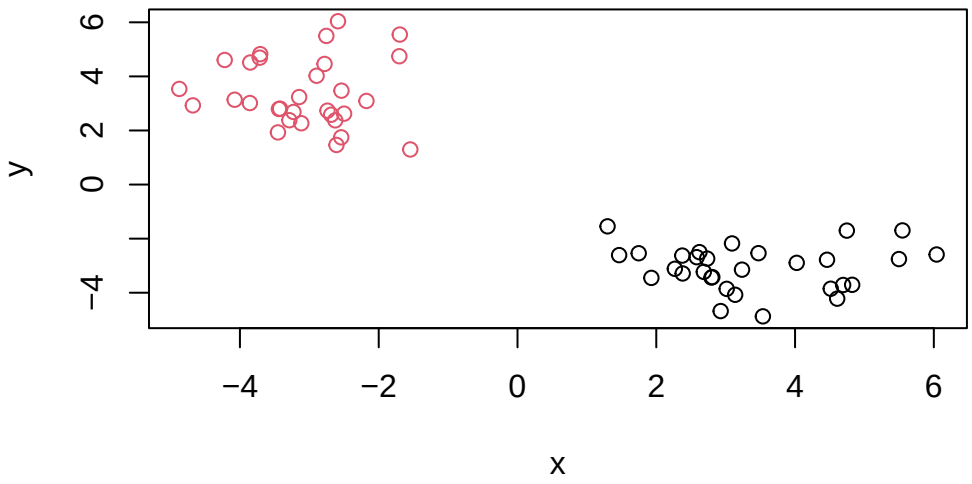


The function to extract clusters/grps from a hclust object/tree is called `cutree()`:

```
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
```

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




```
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
```

##Principal Component Analysis (PCA) The main function for PCA in base R for PCA is call `prcomp()`. There are many, many add on packages with PCA functions tailored to particular data types (RNAseq, protein, structure, metagenomics, etc…)

##PCA of UK Food Data Read the data into R, it's a CSV file and we can use `read.csv()` to read it:

```
# read via x <- read.csv("UK_foods.csv")
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)
#I would like the food names as row names not their own column of data(first column curently)
rownames(x) <-  x[,1]
y <- x[ ,-1]
y
```

```
                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 do it at the time of data import with `read.csv()`.

```
food <- read.csv(url, row.names =1)
food
```

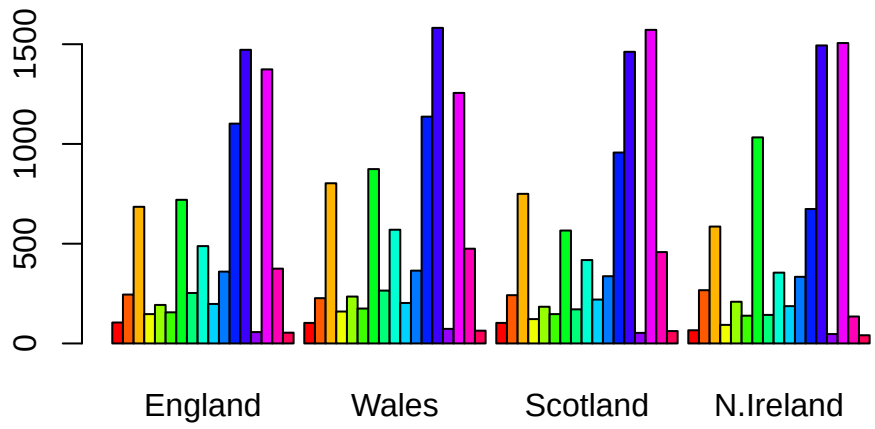
	England	Wales	Scotland	N.Ireland
Cheese	105	103	103	66
Carcass_meat	245	227	242	267
Other_meat	685	803	750	586
Fish	147	160	122	93
Fats_and_oils	193	235	184	209
Sugars	156	175	147	139
Fresh_potatoes	720	874	566	1033
Fresh_Veg	253	265	171	143
Other_Veg	488	570	418	355
Processed_potatoes	198	203	220	187
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

Let's make some plots and dig into the data a little.

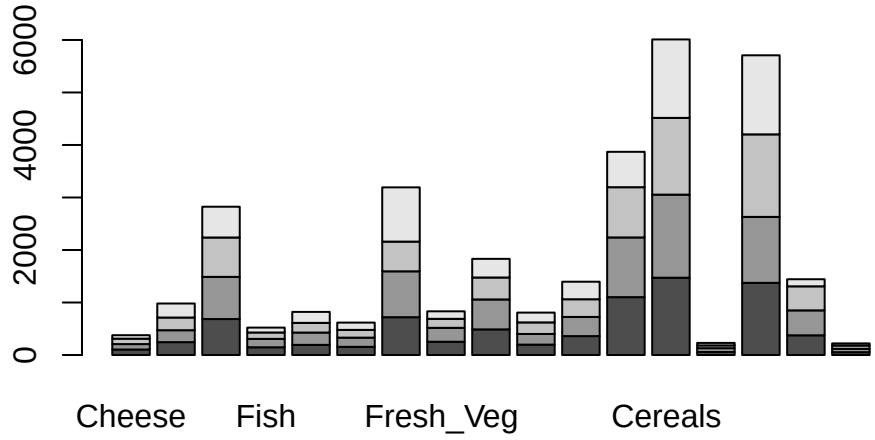
```
rainbow(nrow(food))
```

```
 [1] "#FF0000" "#FF5A00" "#FFB400" "#F0FF00" "#96FF00" "#3CFF00" "#00FF1E"
 [8] "#00FF78" "#00FFD2" "#00D2FF" "#0078FF" "#001EFF" "#3C00FF" "#9600FF"
[15] "#F000FF" "#FF00B4" "#FF005A"
```

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

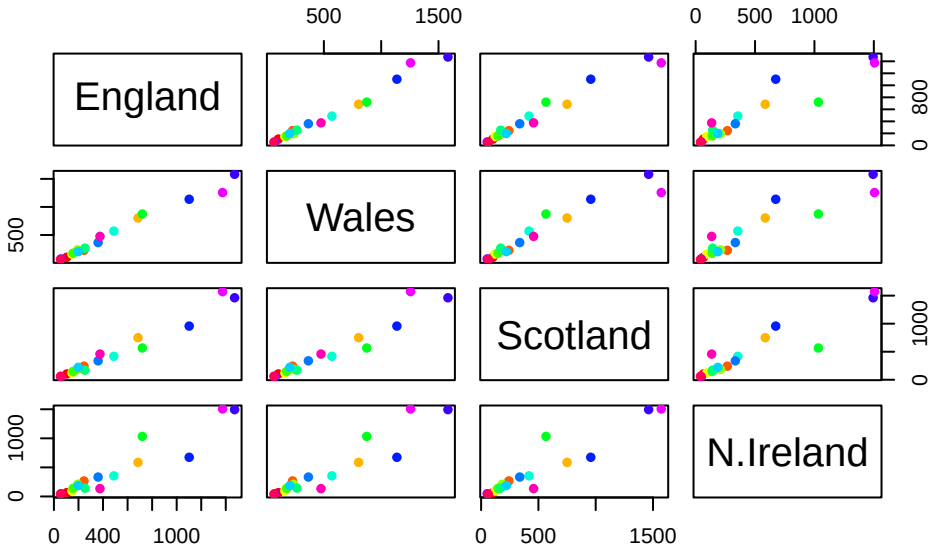



```
barplot(as.matrix(t(food), beside=T))
```



How a bout a "pairs" plot where we plot each country against all other countries.

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



pairwise plot only works for 5 coutnries or less so ther has to be a better way...

##PCA to the rescue!!

We can run a Principal Component Analysis (PCA) for this data with the `prcomp()` function.

We need to take the transpose of this data to get the foods in the columns and the countries in the rows

```
#need to place countries on the columns and food on rows so a flip
pca <- prcomp(t(food))
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
```

What is in my `pca` result object?

```
#attributes will tell me what in this object
attributes(pca)
```

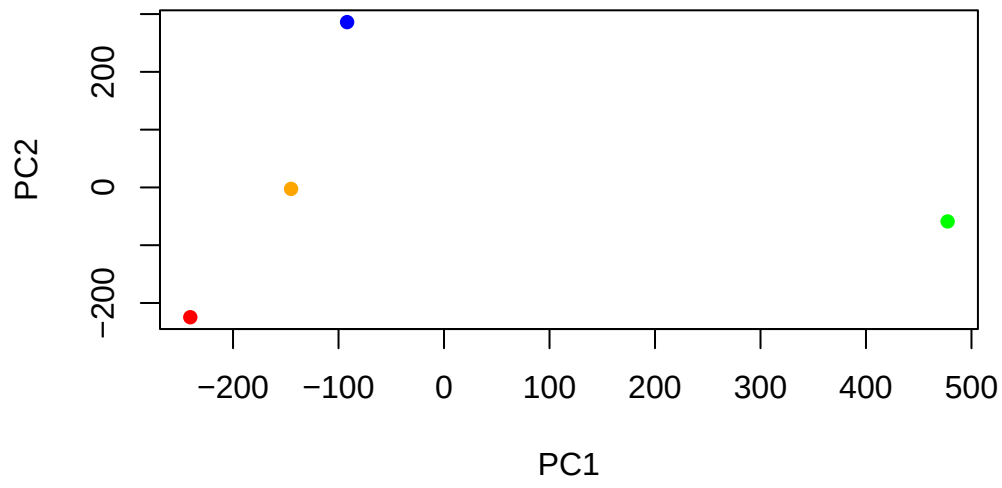
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```
#The scores along the new PCs

pca$x
```

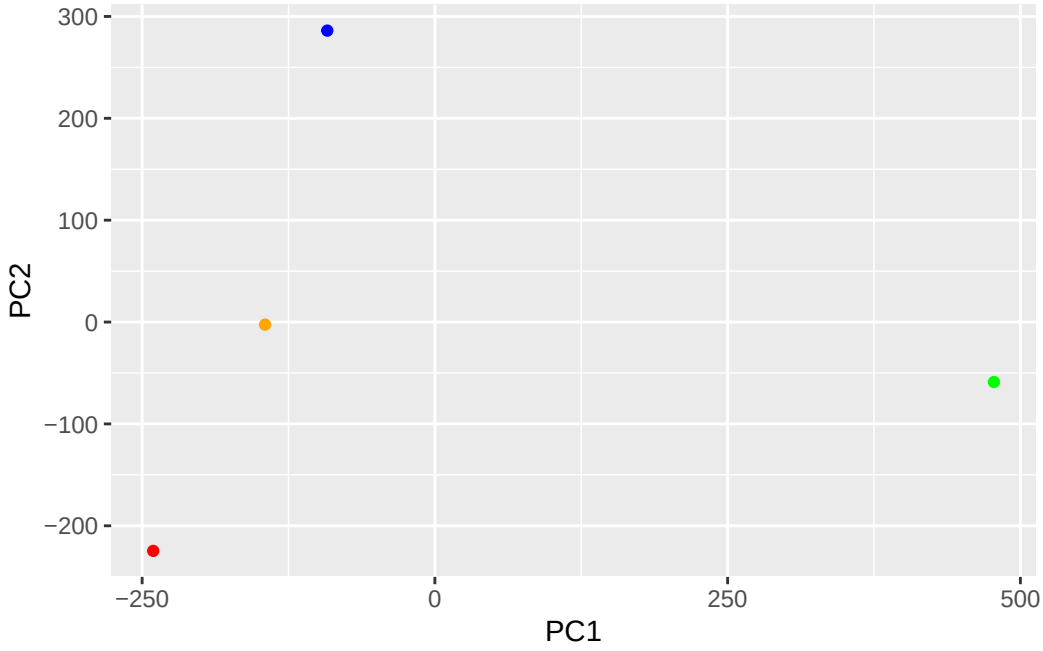
```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -4.894696e-14
Wales     -240.52915 -224.646925 -56.475555  5.700024e-13
Scotland   -91.86934  286.081786 -44.415495 -7.460785e-13
N.Ireland  477.39164  -58.901862  -4.877895  2.321303e-13
```

To make my main results figure, often called a PC plot (or score plot or oridenation plot, or PC1 vs PC2 plot ect.)

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

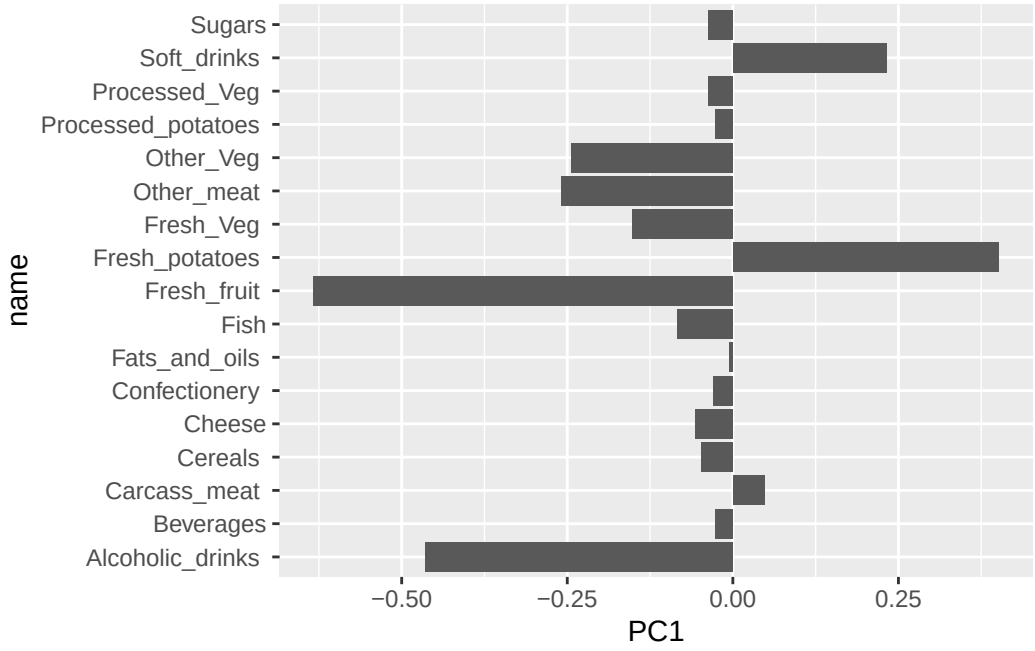



```
library(ggplot2)
data <- as.data.frame(pca$x)
ggplot(data) + aes(PC1, PC2) + geom_point(col = c("orange", "red", "blue", "green"))
```



To see the contributions of the original variables (food) to these new PC we can look at the `pca$rotation` component of our results object.

```
loadings <-  as.data.frame(pca$rotation)
loadings$name <- rownames(loadings)

ggplot(loadings) + aes(PC1, name) + 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
name
−0.50
−0.25
0.00
0.25
PC1