Lab 07 Machine Learning 1
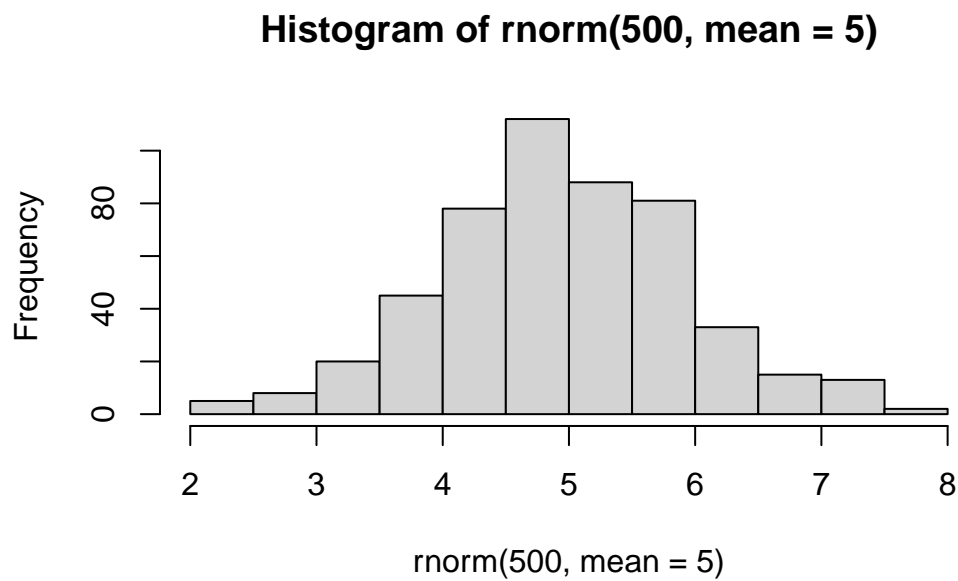
Blinda Sui (PID: A17117043)

Today we will explore some fundamental machine learning methods including clustering and dimensionality reduction.

K-means clustering

To see how this works let's first makeup some data to cluster where we know what the answer should be. We can use the `rnorm()` function to help here:

```
hist(rnorm(500, mean=5))
```

Histogram of rnorm(500, mean = 5)

```
x <- c(rnorm(30, mean=-3),rnorm(30, mean=3))
y <- rev(x) #reverse x
```

```
cbind(x, y)
```

```
              x          y
 [1,] -3.395773  2.528748
 [2,] -4.783878  2.493567
 [3,] -2.882670  3.835322
 [4,] -3.440399  3.250271
 [5,] -3.315120  3.859012
 [6,] -4.232051  3.225137
 [7,] -3.678088  4.246030
 [8,] -1.942925  2.513592
 [9,] -2.525976  2.760413
[10,] -3.450852  2.973410
[11,] -2.694751  4.889120
[12,] -1.624324  5.123273
[13,] -4.573737  2.471611
[14,] -2.098139  2.571778
[15,] -2.408505  2.236146
[16,] -3.734090  2.895889
[17,] -2.224290  3.137766
[18,] -3.106477  1.499477
[19,] -6.407909  4.320899
[20,] -6.133922  2.816261
[21,] -1.437689  4.498610
[22,] -3.519751  2.007183
[23,] -2.469261  2.403540
[24,] -3.684403  1.818132
[25,] -5.318315  2.522348
[26,] -3.230949  3.915822
[27,] -3.354363  3.724570
[28,] -2.927445  2.214595
[29,] -1.939310  2.607254
[30,] -1.837975  3.346758
[31,]  3.346758 -1.837975
[32,]  2.607254 -1.939310
[33,]  2.214595 -2.927445
[34,]  3.724570 -3.354363
[35,]  3.915822 -3.230949
```

```
[36,]   2.522348 -5.318315
[37,]   1.818132 -3.684403
[38,]   2.403540 -2.469261
[39,]   2.007183 -3.519751
[40,]   4.498610 -1.437689
[41,]   2.816261 -6.133922
[42,]   4.320899 -6.407909
[43,]   1.499477 -3.106477
[44,]   3.137766 -2.224290
[45,]   2.895889 -3.734090
[46,]   2.236146 -2.408505
[47,]   2.571778 -2.098139
[48,]   2.471611 -4.573737
[49,]   5.123273 -1.624324
[50,]   4.889120 -2.694751
[51,]   2.973410 -3.450852
[52,]   2.760413 -2.525976
[53,]   2.513592 -1.942925
[54,]   4.246030 -3.678088
[55,]   3.225137 -4.232051
[56,]   3.859012 -3.315120
[57,]   3.250271 -3.440399
[58,]   3.835322 -2.882670
[59,]   2.493567 -4.783878
[60,]   2.528748 -3.395773
```

```
plot(x)
```
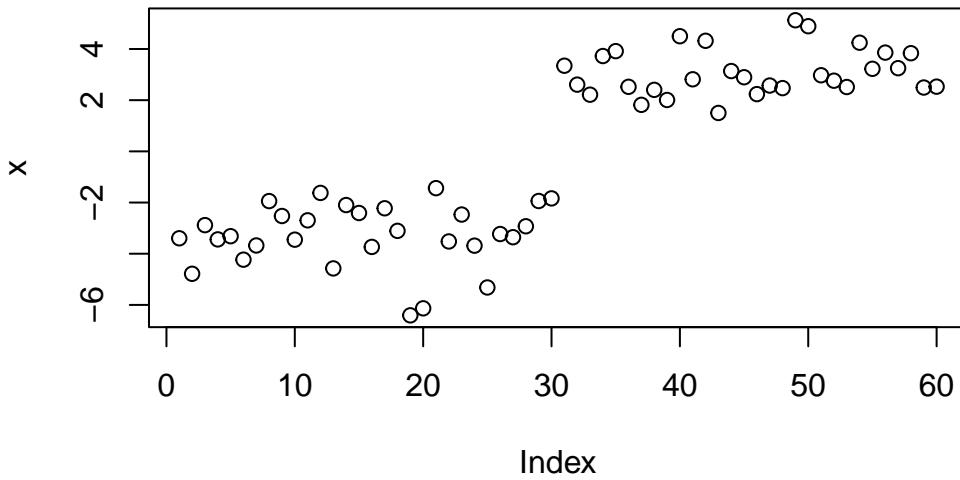
The function for K-means clustering in "base" R is `kmeans()`

```
kmeans(x, centers = 2)
```

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

Cluster means:
       [,1]
1 -3.279111
2  3.090218

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] 44.53988 24.28101
 (between_SS / total_SS =  89.8 %)

Available components:

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

4

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

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

Cluster means:
        [,1]
1  3.090218
2 -3.279111

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 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] 24.28101 44.53988
 (between_SS / total_SS =  89.8 %)

Available components:

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

#:) Clustering vector: the first 30 belongs to cluster 1 and last 30 belongs to cluster 2 #:) Within cluster sume of squares by cluster (WCSS): For k=2 clusters, these are the WCSS values for cluster 1 and cluster 2. Lower = tighter cluster. Here, cluster 2 is looser (30.12 > 16.66). #:) (between_SS / total_SS = 92.4%): the proportion of total variance "explained" by the separation between the two cluster centroids = only 7.6% of the variance remains within clusters.

To get the results of the returned list object we can use the dollar $ syntax.

Q. How many points are in each cluster?

```
k$size
```

```
[1] 30 30
```

Q. What 'component' of your result object details - cluster assignment/membership? - cluster center?
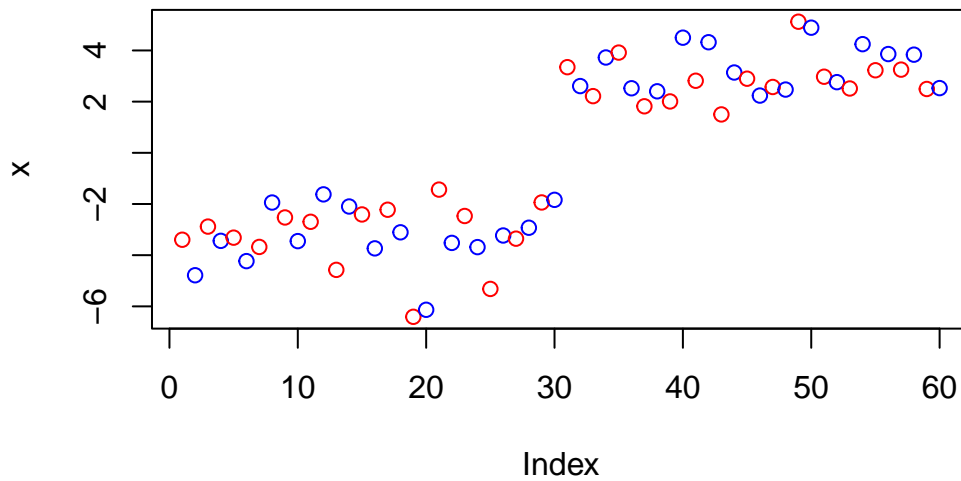
5

```
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 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 1 1
```
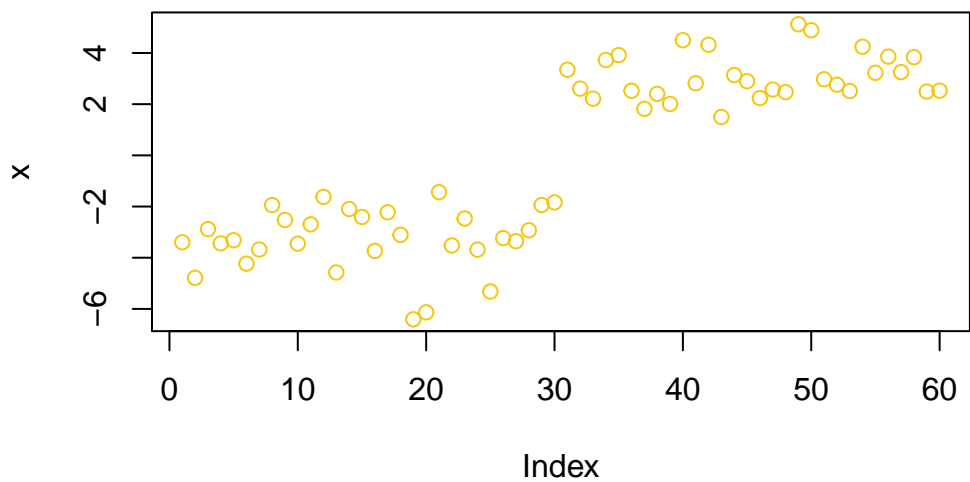
```
k$centers
```

```
        [,1]
1  3.090218
2 -3.279111
```

Q. Make a clustering results figure of the data colored by cluster membership and show cluster centers.
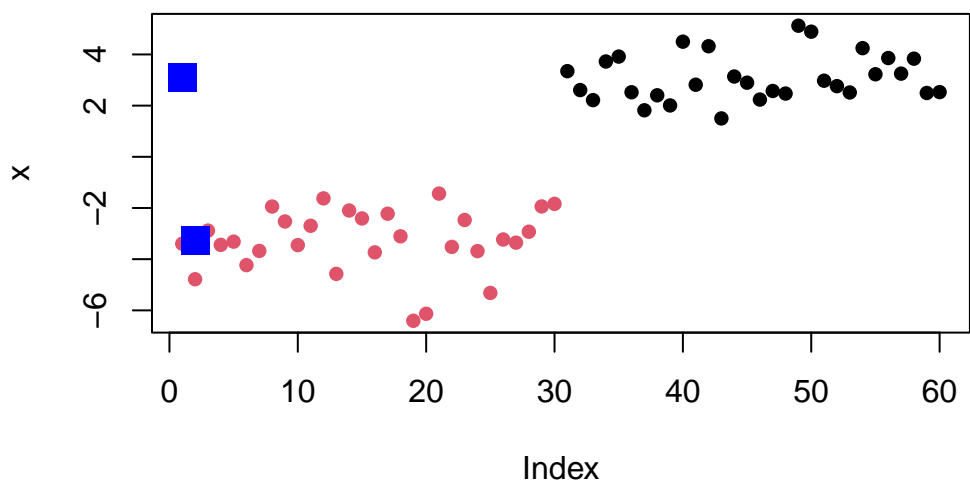
```
plot(x, col = c("red", "blue"))
```



```
plot(x, col = 7)
```

```
plot(x, col = k$cluster, pch=16)
points(k$centers, col="blue", pch=15, cex=2)
```

K-means clustering is very popular as it is very fast and relatively straight forward: it takes numeric data as input and returns the clusterm membership vector etc.

The "issue" is we tell `kmeans()` how many clusters we want!

> Q. Run kmeans again and cluster into 4 frps/clusters and plot the results like we did above?

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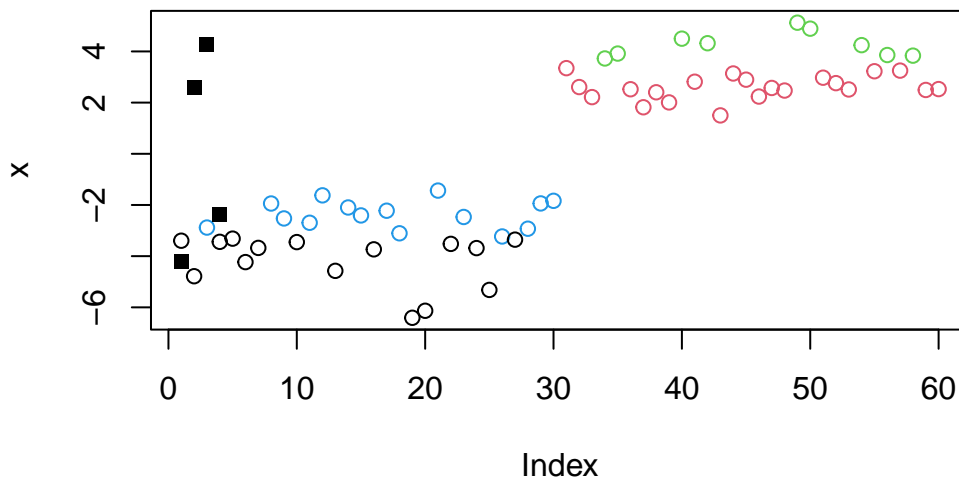

Scree plot to pick k `center` value

brute-force

```
k1 <- kmeans(x, centers = 1)
k2 <- kmeans(x, centers = 2)
k3 <- kmeans(x, centers = 3)
k4 <- kmeans(x, centers = 4)
k5 <- kmeans(x, centers = 5)
```
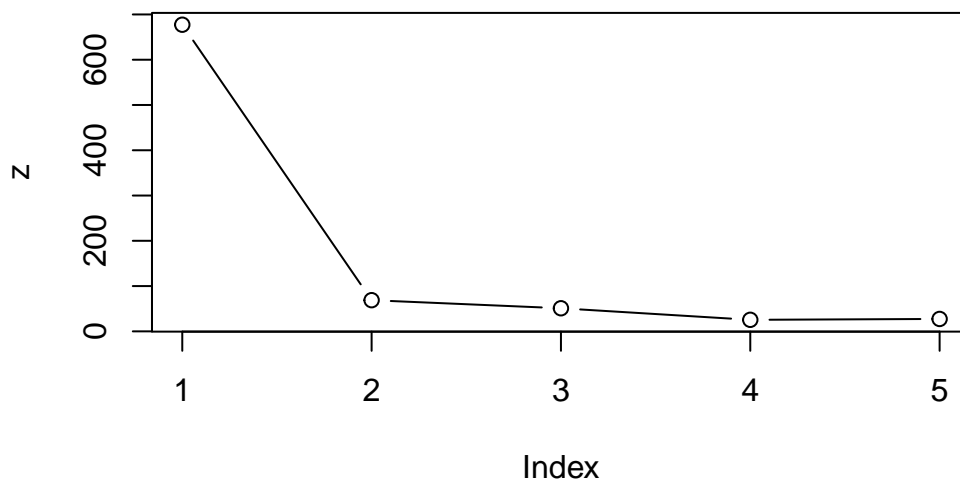
```r
z <- c(k1$tot.withinss,
       k2$tot.withinss,
       k3$tot.withinss,
       k4$tot.withinss,
       k5$tot.withinss)

plot(z, typ = "b")
```
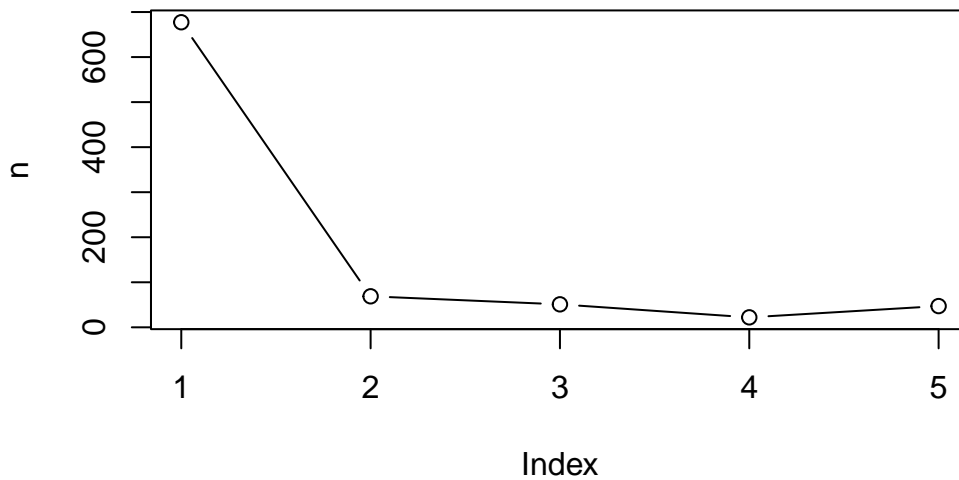


```r
n <- NULL
for(i in 1:5) {
  n <- c(n, kmeans(x, centers = i)$tot.withinss)
}

plot(n, typ = "b")
```

Hierarchical CLustering

The main "base" R function for Hierarchical Clustering is called `cluster()`. Here we can't just input our data, we need to first calculate a distance matrix (e.g.`dist()`) for our data and use this as input to `hclust()`
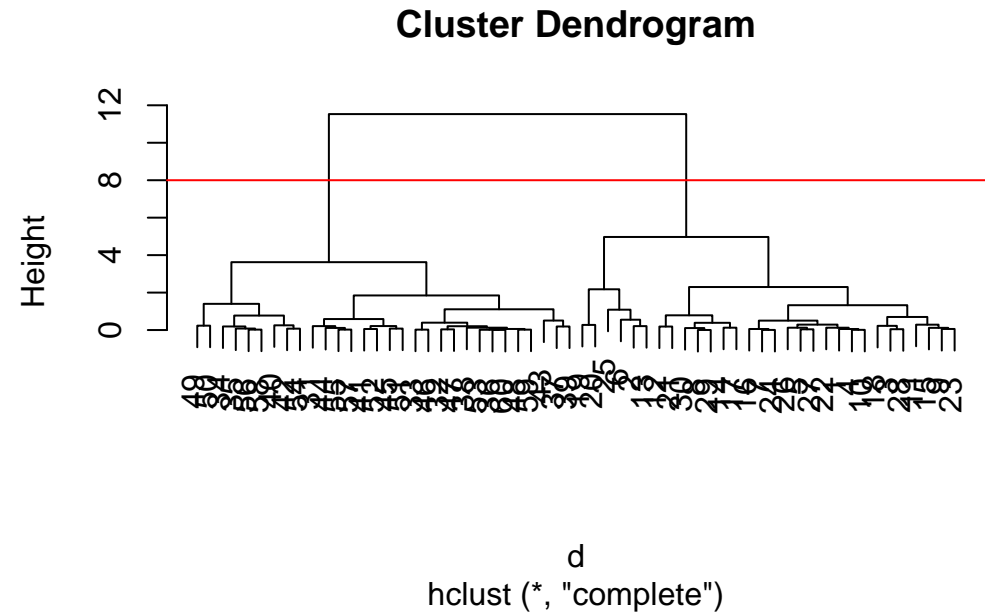
```r
d <- dist(x)
hc <- hclust(d)
hc
```

```
Call:
hclust(d = d)

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

There is a plot method for hclust results, lets try it

```
plot(hc)
abline(h=8, col="red") #abline(): to add straight lines to an existing plot.
```

Cluster Dendrogram

d
hclust (*, "complete")

To get our cluster "membership" vector (i.e. our main clustering result) we can "cut" the tree at a given height or at height that yields a given "k" gorups.

```
cutree(hc, h=8)
```

```
 [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 1 1 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
```
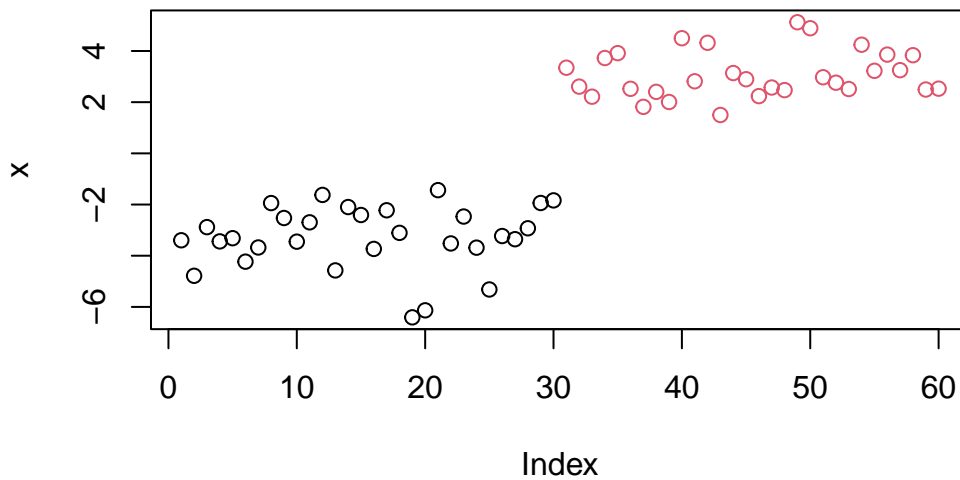
```
grps <- cutree(hc, k=2)
```

Q. Plot the data with our hclust result coloring

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

Principal Component Analysis (PCA)

PCA of UK food data

Import food data from an online CSV file:

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

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

```r
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
```
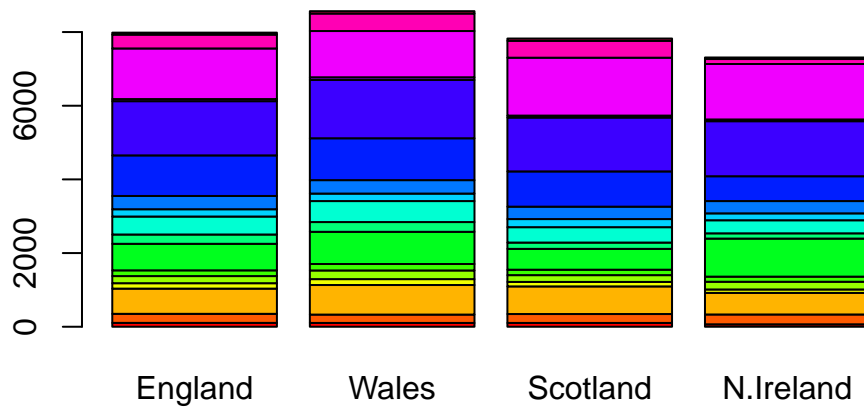
some base figures

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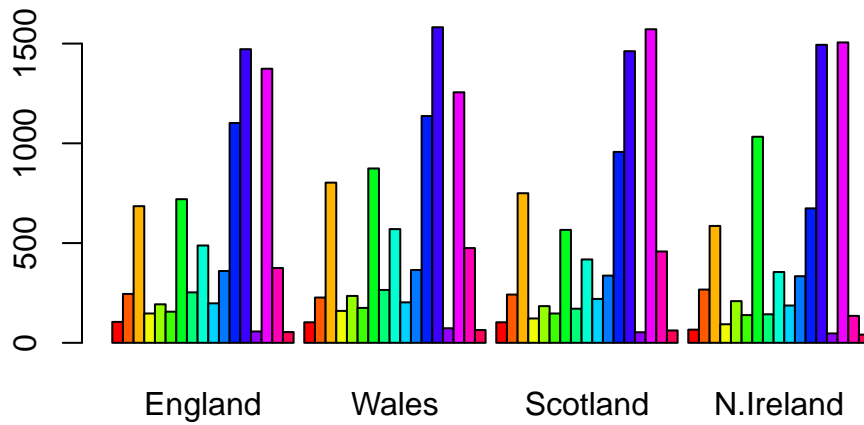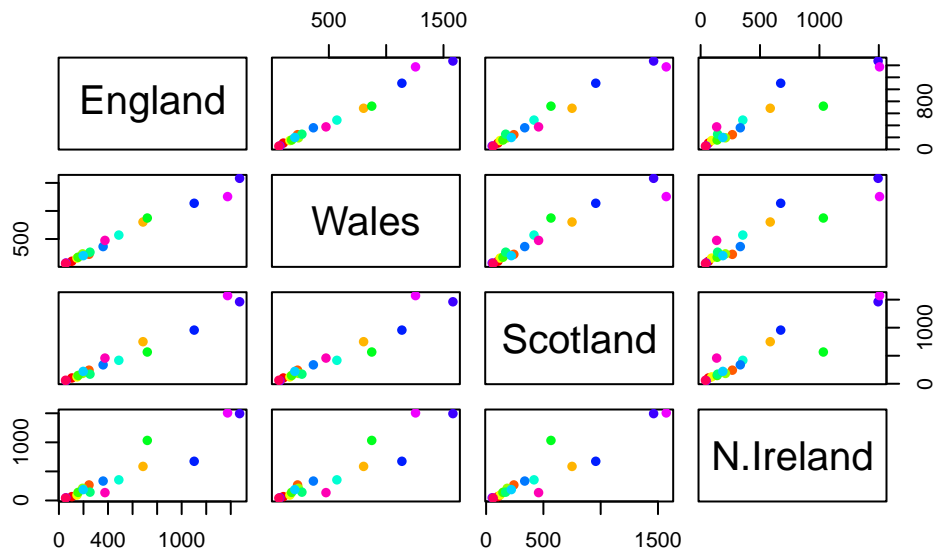


```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```

There is one plot that can be useful for small datasets:

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


Main point: It can be difficult to spot major trends and patterns even in relatively small multivariate datasets (here we only have 17 dimensions, typically we have 1000s).

PCA to the rescue

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

I will take the transpose of our data so the "food" are in the columns:

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

```
Importance of components:
                          PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 2.921e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```
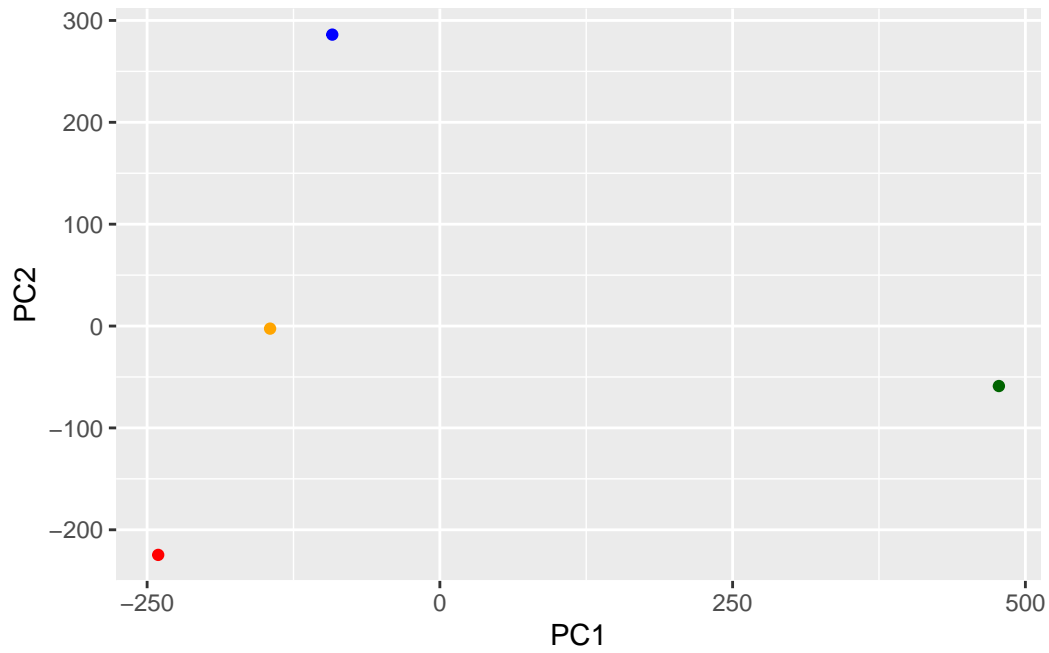
15

```
cols <- c("orange", "red", "blue", "darkgreen")
plot(pca$x[,1], pca$x[,2], col=cols, pch=16)
```
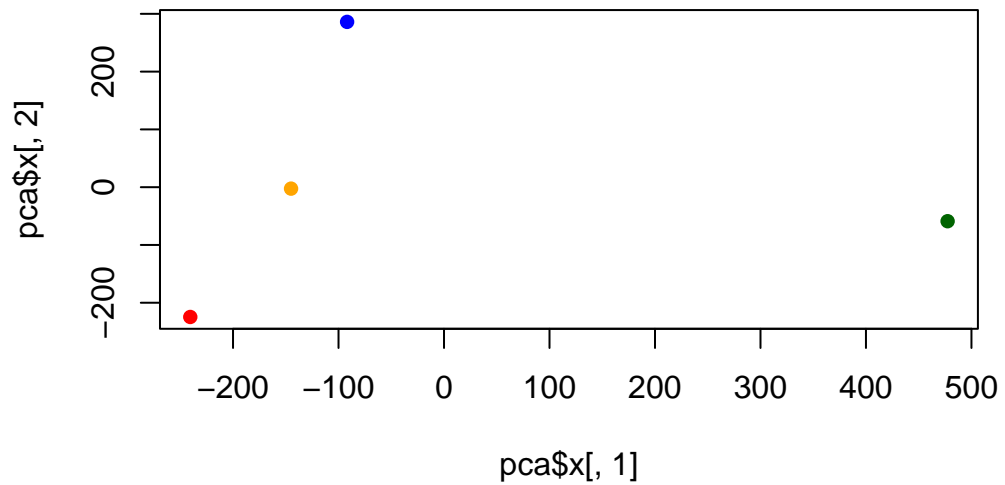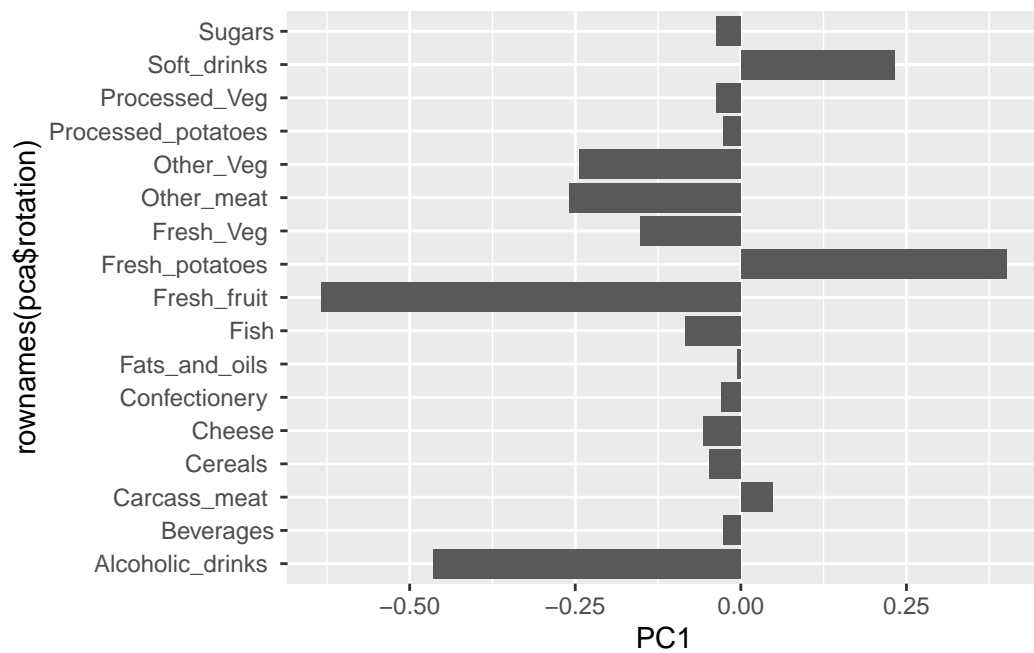


```
library(ggplot2)
```

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

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


17

PCA looks super useful and we will come back to describe this further next day ;-)