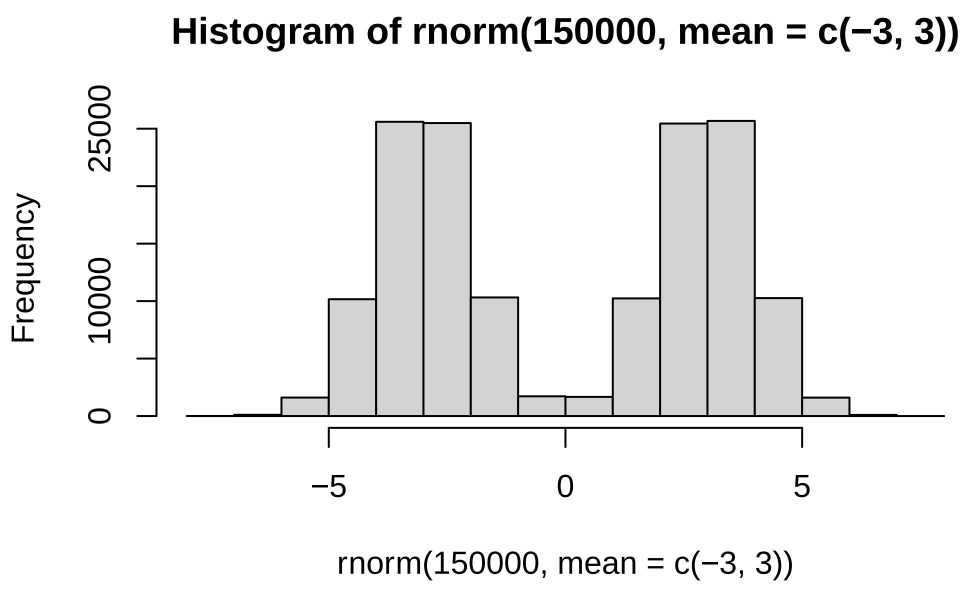
Class 07

Tim

Before we get into clustering methods let's make some sample data to cluster where we know what the answer should be

To help with this I will use the `rnorm()` function

```r
hist(rnorm(150000, mean = c(-3,3)))
```



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

```
 [1]  3.0673744  3.3921853  4.9633241  2.5177356  2.6219122  2.9027532
 [7]  2.8145837  2.4239096  3.1507842  2.7433154  3.0907909  3.7790824
```

```
[13]  1.1990955  1.5348185  4.4188325  3.1053706  2.5333590  2.0666045
[19]  4.0228372  1.9275282  3.7377766  3.7639632  3.5439448  4.0378093
[25]  2.3301366  1.8500144  2.3368215  3.0600333  4.1878627  4.5378783
[31] -1.8516578 -2.8796135 -3.9116693 -4.6042732 -0.7613992 -2.6786801
[37] -5.0145288 -3.5069505 -3.7206075 -3.5670613 -3.5723703 -2.1933569
[43] -4.6943384 -1.1339635 -4.0082621 -4.8991815 -4.1265986 -3.0615853
[49] -3.1763520 -3.6077591 -4.0878717 -1.8227651 -0.5573039 -2.5862250
[55] -3.1308867 -4.4812359 -3.5378964 -3.7234471 -2.0955326 -3.5964401
```

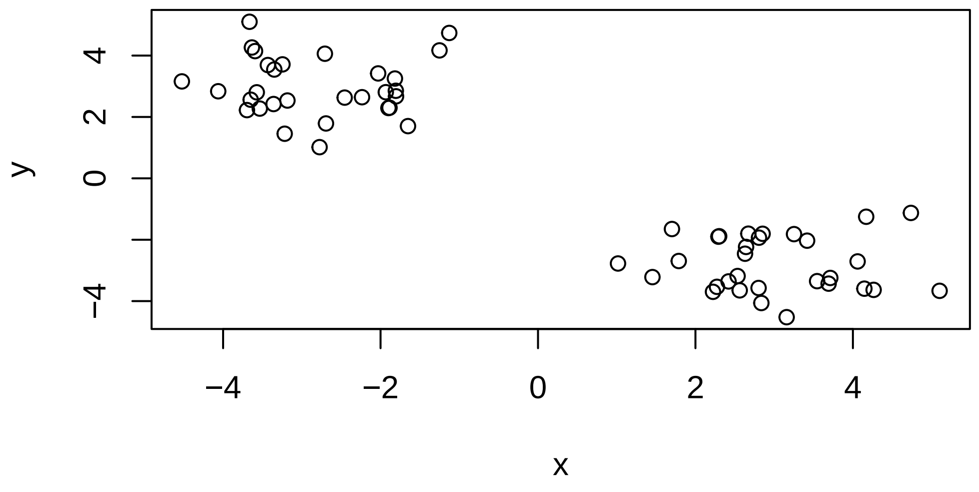
```r
n = 30
x <- c(rnorm(n, mean = 3), rnorm(n, mean = -3))
y <- rev(x)

z <- cbind(x,y)
z
```

```
             x         y
 [1,]  1.453668 -3.217778
 [2,]  5.101532 -3.664622
 [3,]  2.629980 -2.455965
 [4,]  3.712873 -3.245920
 [5,]  3.418594 -2.030575
 [6,]  2.853070 -1.806018
 [7,]  4.145809 -3.594470
 [8,]  4.263497 -3.634083
 [9,]  4.168928 -1.251198
[10,]  3.158374 -4.523698
[11,]  1.016139 -2.774650
[12,]  2.837173 -4.062118
[13,]  3.251956 -1.817334
[14,]  2.420578 -3.359221
[15,]  2.562483 -3.650044
[16,]  4.736963 -1.127376
[17,]  2.802062 -3.572527
[18,]  4.059955 -2.707002
[19,]  2.807184 -1.933435
[20,]  2.273081 -3.534090
[21,]  3.545364 -3.350019
[22,]  2.301536 -1.886402
[23,]  2.671902 -1.803242
[24,]  3.691481 -3.431428
[25,]  1.701671 -1.651517
```

```
[26,]  2.291465 -1.900011
[27,]  2.642904 -2.235410
[28,]  2.222915 -3.696900
[29,]  1.788067 -2.693113
[30,]  2.535225 -3.183098
[31,] -3.183098  2.535225
[32,] -2.693113  1.788067
[33,] -3.696900  2.222915
[34,] -2.235410  2.642904
[35,] -1.900011  2.291465
[36,] -1.651517  1.701671
[37,] -3.431428  3.691481
[38,] -1.803242  2.671902
[39,] -1.886402  2.301536
[40,] -3.350019  3.545364
[41,] -3.534090  2.273081
[42,] -1.933435  2.807184
[43,] -2.707002  4.059955
[44,] -3.572527  2.802062
[45,] -1.127376  4.736963
[46,] -3.650044  2.562483
[47,] -3.359221  2.420578
[48,] -1.817334  3.251956
[49,] -4.062118  2.837173
[50,] -2.774650  1.016139
[51,] -4.523698  3.158374
[52,] -1.251198  4.168928
[53,] -3.634083  4.263497
[54,] -3.594470  4.145809
[55,] -1.806018  2.853070
[56,] -2.030575  3.418594
[57,] -3.245920  3.712873
[58,] -2.455965  2.629980
[59,] -3.664622  5.101532
[60,] -3.217778  1.453668
```

```
plot(z)
```



K-means clustering

the function in base R for kmeana

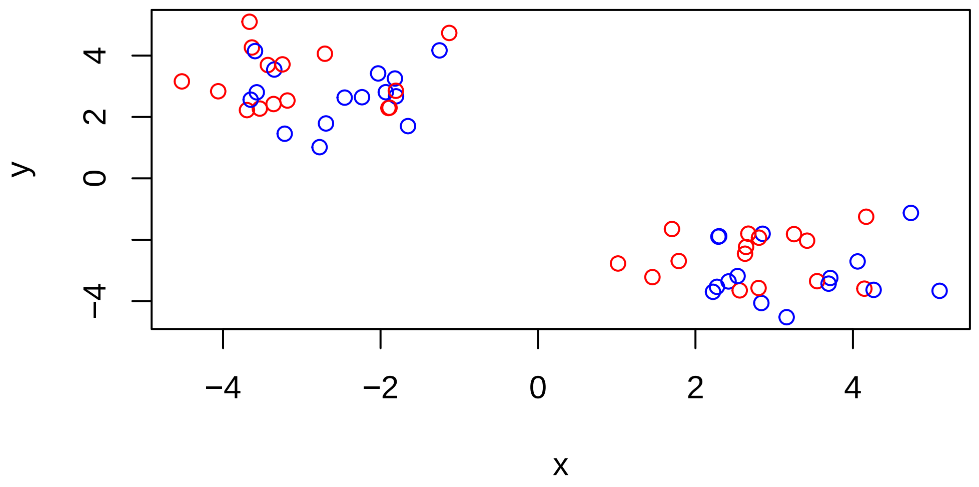
```
km <- kmeans(z, centers =2)
```

Q. print out the cluster membership vector (i.e. our main answer)

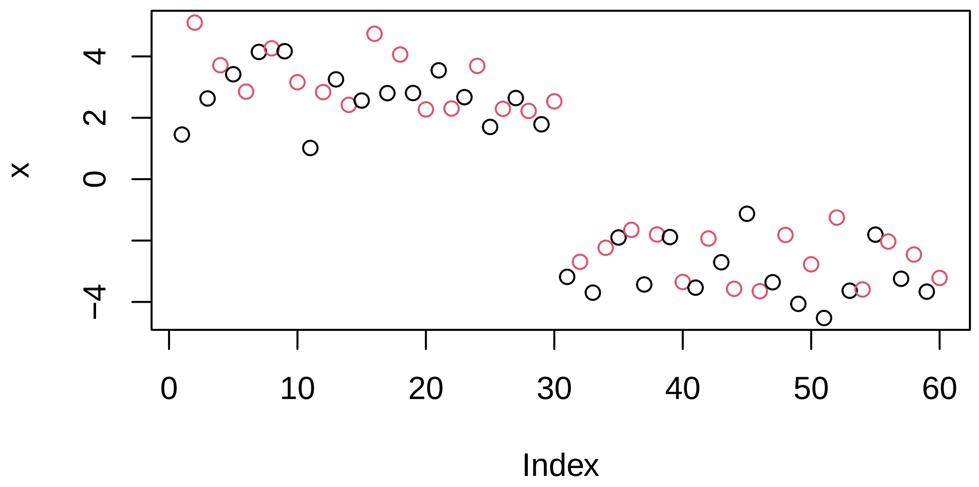
```
km$cluster
```

```
 [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
```

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

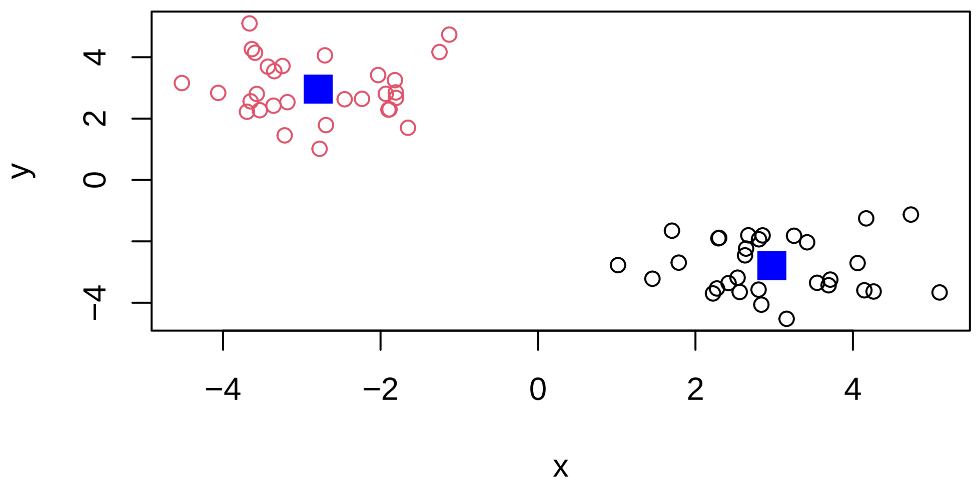


```
plot(x, col=c(1,2))
```



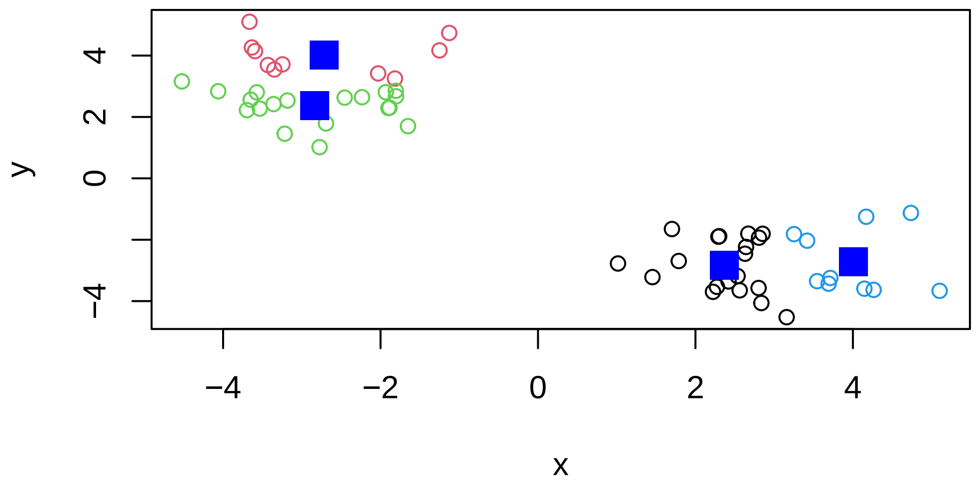
plot with clustering center

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



Q. can you cluster our data in `z` into 4 clusters?

```
km4 <- kmeans(z, centers = 4)
plot(z, col=km4$cluster)
points(km4$centers, col ="blue", pch=15, cex=2)
```



Hierarchical Clustering

the main function for hierarachical Clustering in base R is called `hclust`

unlike `kmeans()` I can not just pass in my data as input I first need a distance matrix from data.

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

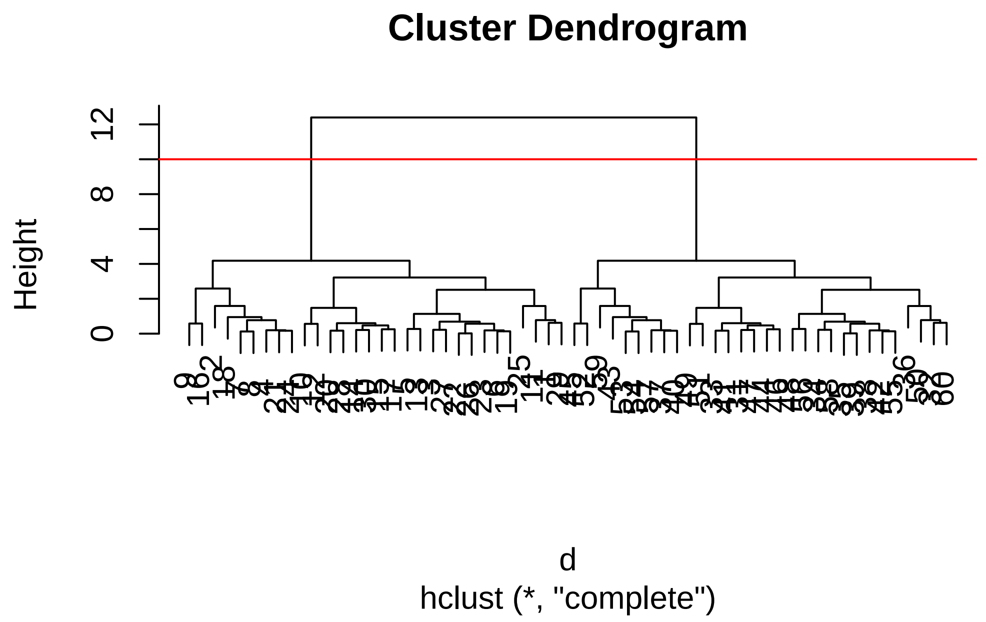
```

Call:
hclust(d = d)

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

there is a specific hclust plot() method..

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



the longer the cluster bar the larger the distance and the more space to scoare a difference between the clusters

to get my main clustering results (i.e. the membership vector) can "cut" my tree at a given height. To add this I will use the `cutree`

```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
```

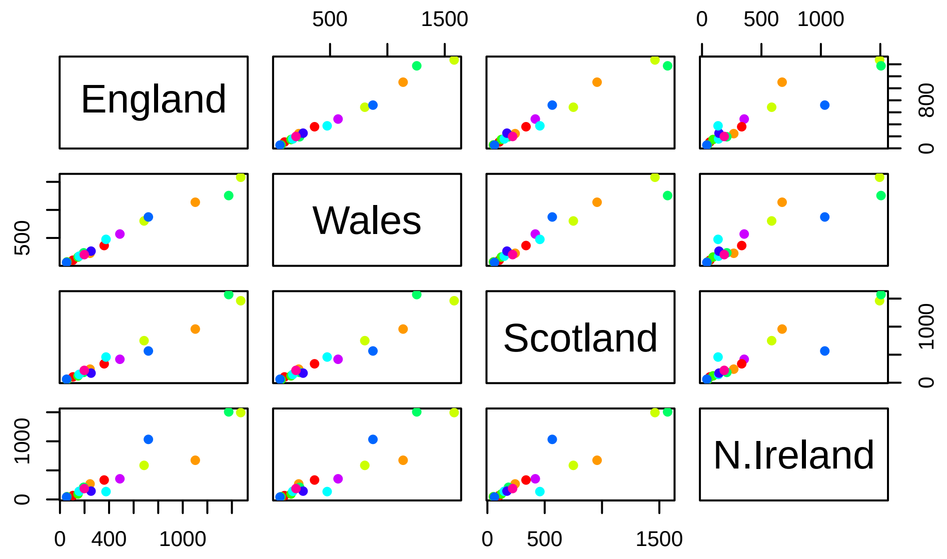
Principal Component Analysis

reduce the features dimensionality while only losing a small amount of info (follows the best fit line through points)

```r
url <- "https://tinyurl.com/UK-foods"
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
```

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



the main function to do PCA in base R is called `prcomp()`

t() function is trnapose to switch the x axis to make it y axis

```
t(x)
```

```
          Cheese Carcass_meat  Other_meat  Fish Fats_and_oils  Sugars
England      105           245         685   147           193     156
Wales        103           227         803   160           235     175
Scotland     103           242         750   122           184     147
N.Ireland     66           267         586    93           209     139
          Fresh_potatoes  Fresh_Veg  Other_Veg  Processed_potatoes
England              720        253        488                 198
Wales                874        265        570                 203
Scotland             566        171        418                 220
N.Ireland           1033        143        355                 187
          Processed_Veg  Fresh_fruit  Cereals  Beverages Soft_drinks
England             360         1102     1472         57        1374
Wales               365         1137     1582         73        1256
Scotland            337          957     1462         53        1572
N.Ireland           334          674     1494         47        1506
          Alcoholic_drinks  Confectionery
England                375             54
```

```
Wales                        475             64
Scotland                     458             62
N.Ireland                    135             41
```

```
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
```

#Proportion variance is how much each PC captured (i.e., PC1 captured 67% of the data 0.67)

#cumulative proportion you add PC1 to PC2 (0.67 + 0.29)

let's see what us inside our result object `pca` that we just calculated

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

```
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
```