machine learning 1

Onyeka (PID: A59011964)

10/22/2021

#clustering methods

Kmeans clustering in R is one with the 'kmeans()' function Here we makeup some data to test and learn with

```
tmp <- c(rnorm(30, 3), rnorm(30, -3))
data <- cbind(x=tmp, y=rev(tmp))
data
```

```
##                x          y
##  [1,]  3.6457037 -3.1157619
##  [2,]  1.1910822 -3.4389380
##  [3,]  2.2870111 -2.2975798
##  [4,]  2.8826480 -4.4206933
##  [5,]  3.6630774 -3.4447617
##  [6,]  4.7402614 -3.0075912
##  [7,]  3.2585136 -0.5305349
##  [8,]  3.4284655 -2.2325141
##  [9,]  2.5906188 -3.6578168
## [10,]  2.8712816 -1.9209183
## [11,]  1.8785811 -1.9276798
## [12,]  3.7019664 -2.1538364
## [13,]  3.5723386 -3.0365687
## [14,]  5.1467891 -2.7824709
## [15,]  2.3522528 -2.9948344
## [16,]  2.9406942 -5.2746681
## [17,]  3.7212032 -3.4105293
## [18,]  2.2019197 -4.9961240
## [19,]  4.1322486 -2.5649461
## [20,]  2.9577741 -1.7396571
## [21,]  4.3384140 -3.1443359
## [22,]  3.6692001 -2.0493533
## [23,]  2.9438236 -5.3364504
## [24,]  3.3356486 -3.5837341
## [25,]  4.3277976 -5.0320634
## [26,]  4.3693973 -1.2062081
## [27,]  2.7734564 -2.8306055
## [28,]  1.9892811 -2.6397185
## [29,]  1.4311055 -2.9190240
## [30,]  2.8776286 -5.0904796
## [31,] -5.0904796  2.8776286
## [32,] -2.9190240  1.4311055
```

```
## [33,] -2.6397185  1.9892811
## [34,] -2.8306055  2.7734564
## [35,] -1.2062081  4.3693973
## [36,] -5.0320634  4.3277976
## [37,] -3.5837341  3.3356486
## [38,] -5.3364504  2.9438236
## [39,] -2.0493533  3.6692001
## [40,] -3.1443359  4.3384140
## [41,] -1.7396571  2.9577741
## [42,] -2.5649461  4.1322486
## [43,] -4.9961240  2.2019197
## [44,] -3.4105293  3.7212032
## [45,] -5.2746681  2.9406942
## [46,] -2.9948344  2.3522528
## [47,] -2.7824709  5.1467891
## [48,] -3.0365687  3.5723386
## [49,] -2.1538364  3.7019664
## [50,] -1.9276798  1.8785811
## [51,] -1.9209183  2.8712816
## [52,] -3.6578168  2.5906188
## [53,] -2.2325141  3.4284655
## [54,] -0.5305349  3.2585136
## [55,] -3.0075912  4.7402614
## [56,] -3.4447617  3.6630774
## [57,] -4.4206933  2.8826480
## [58,] -2.2975798  2.2870111
## [59,] -3.4389380  1.1910822
## [60,] -3.1157619  3.6457037
```

Run 'kmenas()' set k to 2 nstart 20. the thing with kmenas is you have to tell it how mnay cluster you have.

```
km<- kmeans(data, centers =2, nstart=20)
km
```

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1  3.174006 -3.092680
## 2 -3.092680  3.174006
##
## 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 2 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 2 2 2 2
##
## Within cluster sum of squares by cluster:
## [1] 68.86709 68.86709
##  (between_SS / total_SS =  89.5 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. how many points are there in each cluster?

```
km$cluster
```

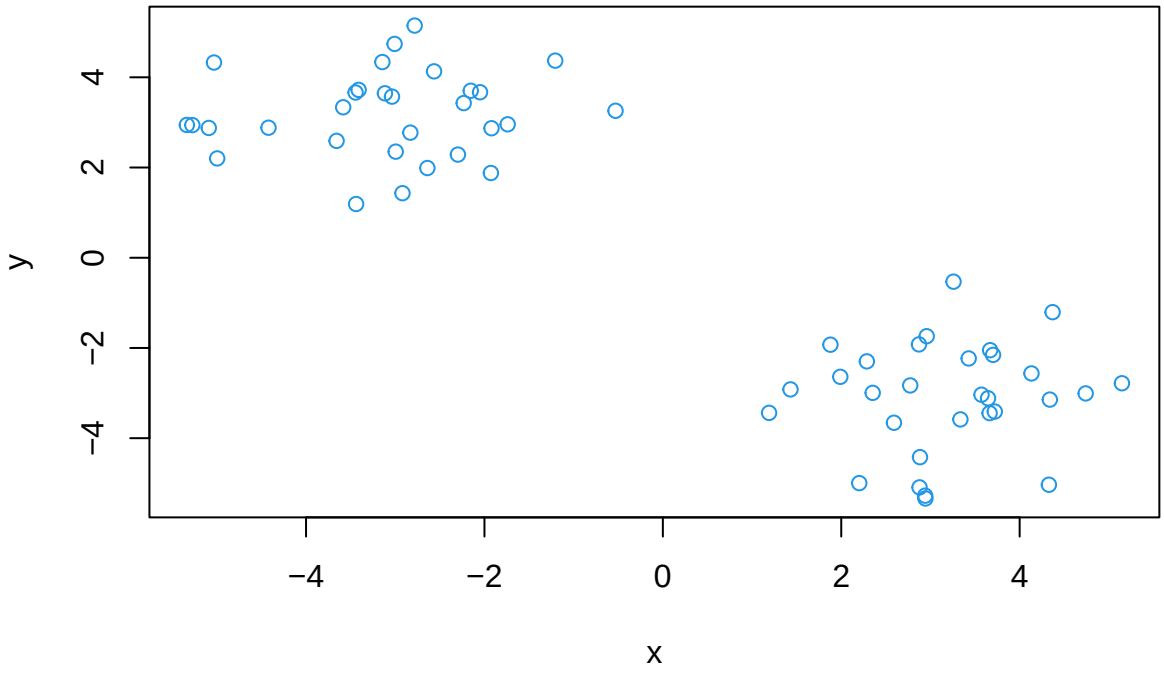
```
##  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
## [39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Q. what component of your result object details cluster center?

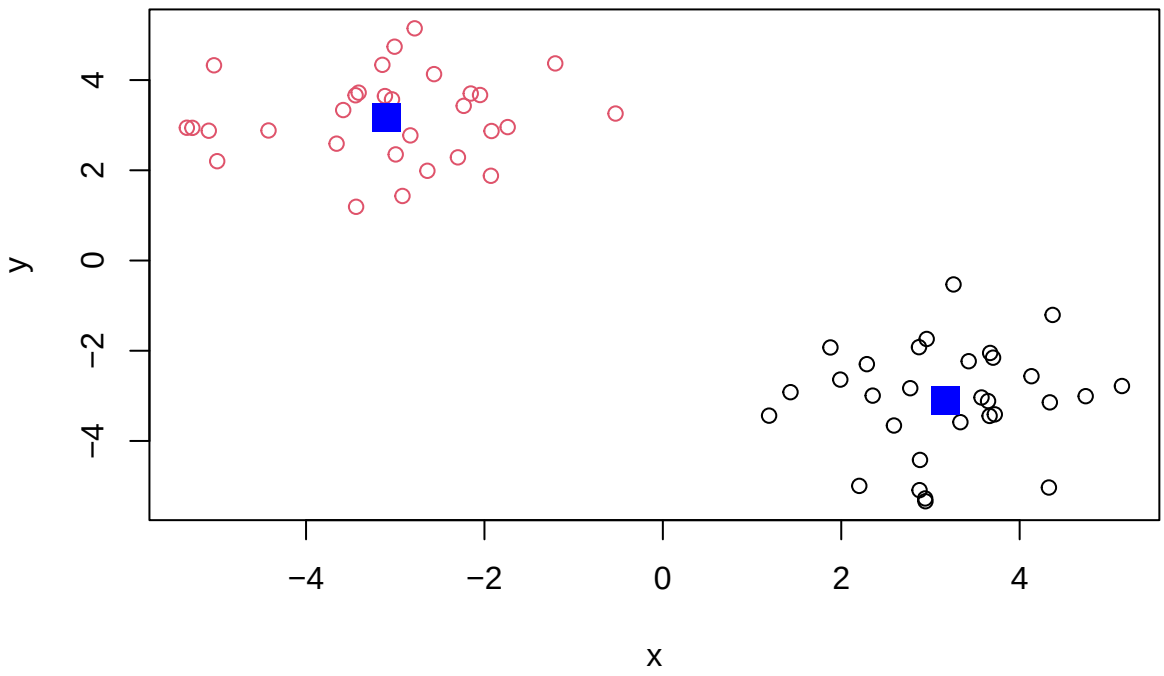
```
km$centers
```

```
##           x         y
## 1  3.174006 -3.092680
## 2 -3.092680  3.174006
```

```
plot(data, col=4)
```




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



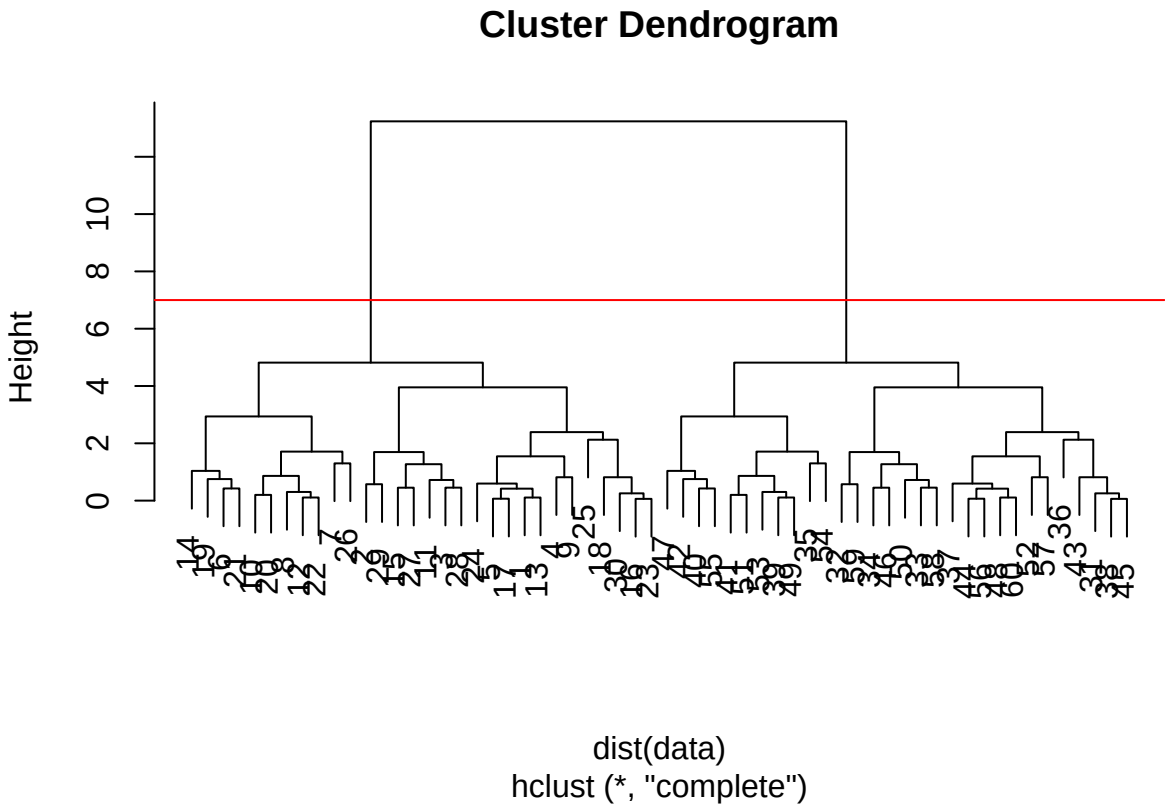
hclust

we will use the '(hclust)'function on the same data as before and see how this method works

```
hc<- hclust (dist(data))
hc
```

```
##
## Call:
## hclust(d = dist(data))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

```
plot(hc)
abline(h=7, col="red")
```



to find our membership vector we need to “cut” the treet and doe this we use the ‘cutree()’ function and tell it the height to cut at

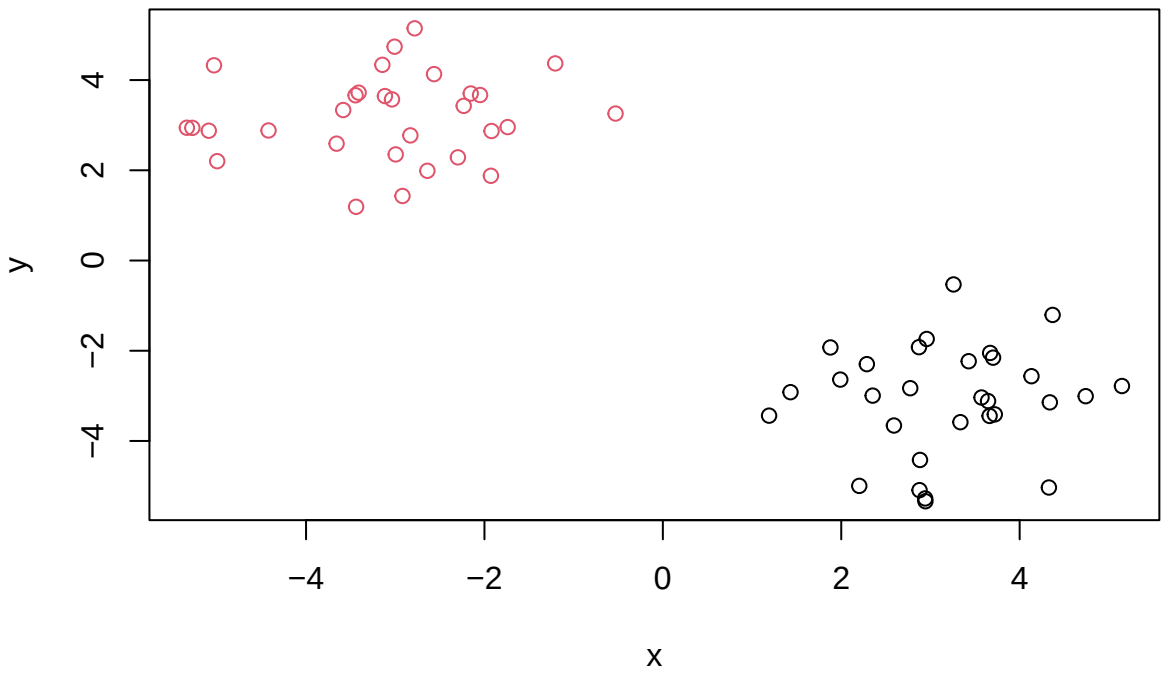
we can use ‘cutree()’ and state the number of k clusters we want

```
cutree(hc, h=4)
```

```
##  [1] 1 1 1 1 1 2 2 2 1 2 1 2 1 2 1 1 1 1 2 2 2 2 1 1 1 2 1 1 1 1 3 3 3 3 4 3 3 3
## [39] 4 4 4 4 3 3 3 3 4 3 4 3 4 3 4 4 4 3 3 3 3 3
```

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

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



#principal cluster analysis (PCA)

PCA is a super useful analysis method when you have lots of dimensions in your data.....

#PCA of UK food data

import the data from a csv file how many role and column

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

```
dim(x)
```

```
## [1] 17  5
```

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

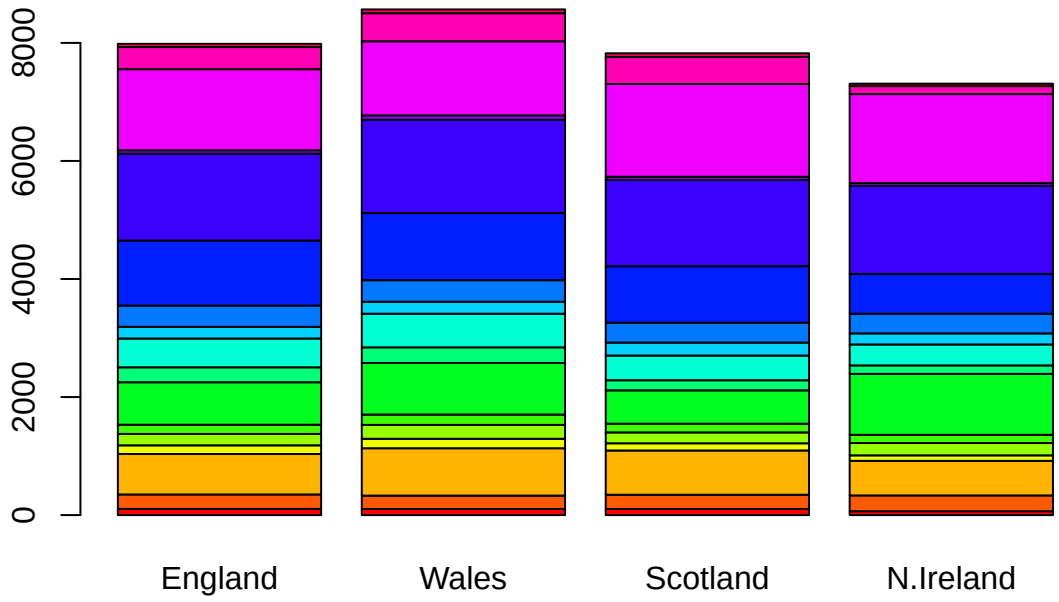
```
##                     England Wales Scotland N.Ireland
## Cheese                  105   103      103        66
## Carcass_meat            245   227      242       267
## Other_meat              685   803      750       586
## Fish                    147   160      122        93
## Fats_and_oils           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
```

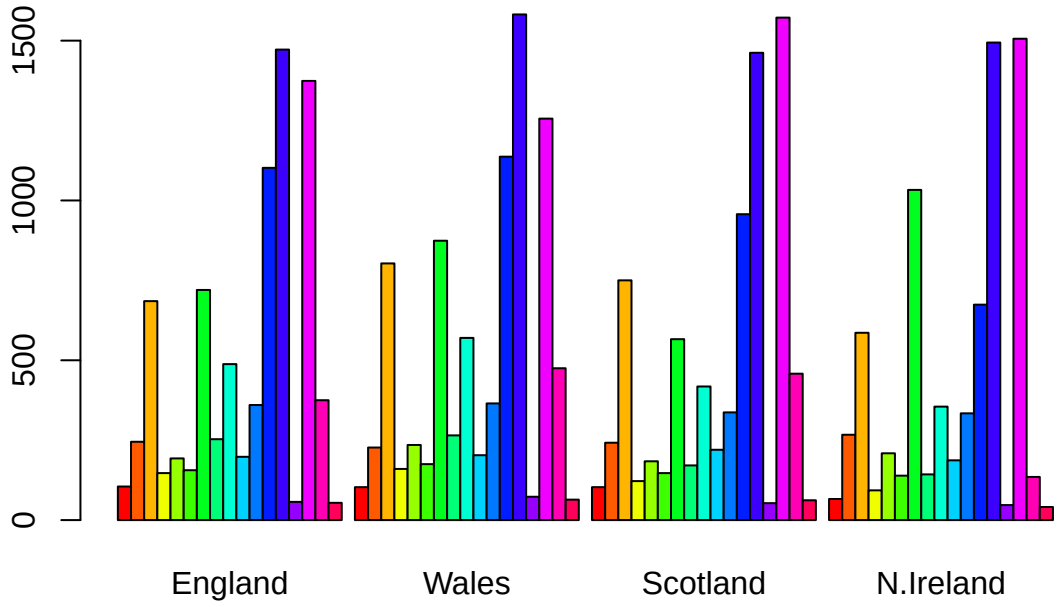
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names = 1)
x
```

```
##                     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
```

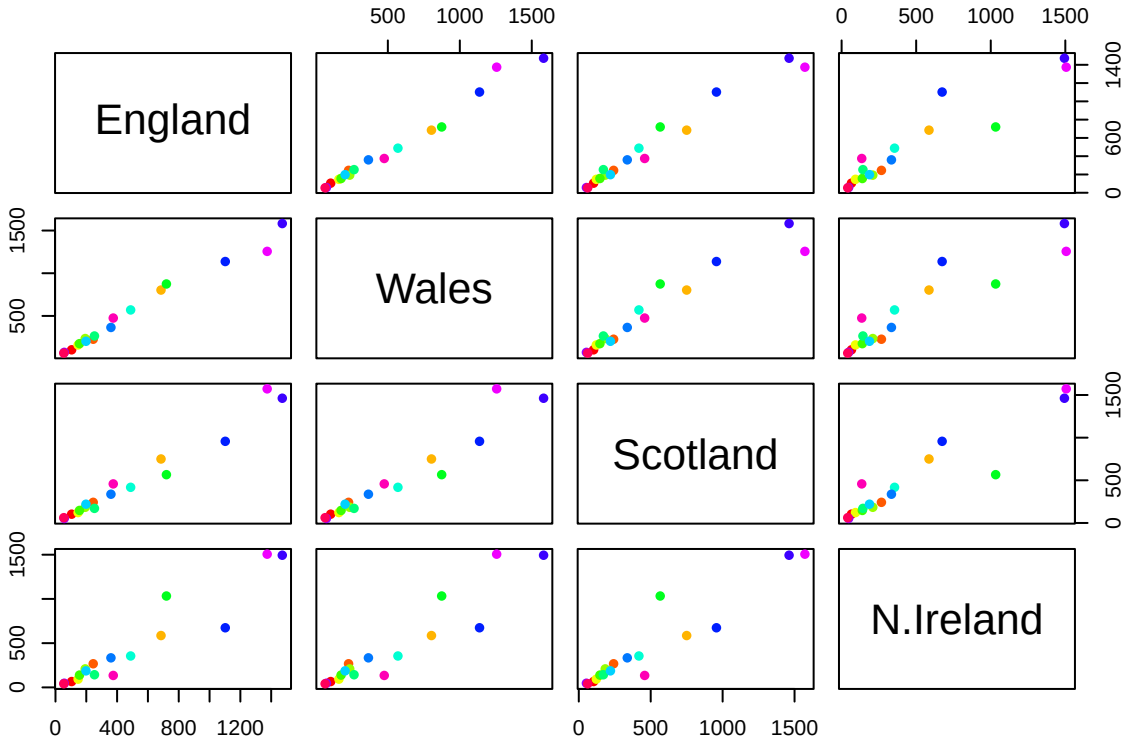
```
barplot( as.matrix(x), col = rainbow(17))
```




```
barplot(as.matrix(x), col = rainbow(17), beside=TRUE)
```




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




```
#paris(x,)
```

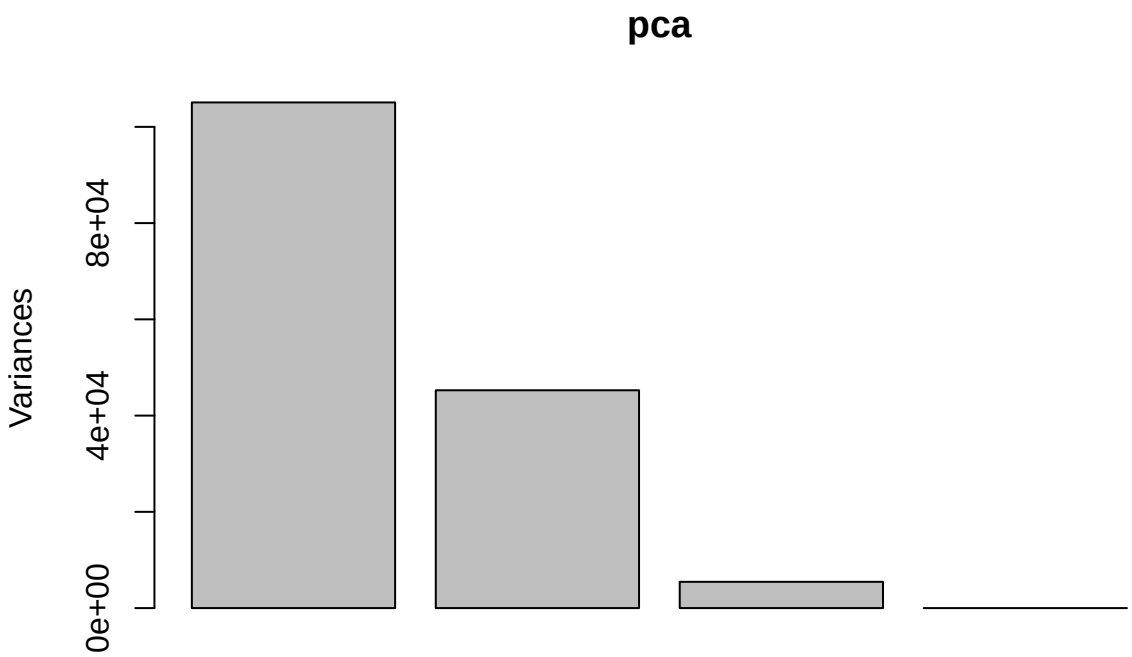
##PCA to the rescue

Here we will sue the nase R dfunction for PCS,w hcih is called 'prcomp()'. The functions wabts the transpose of data

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

```
## Importance of components:
##                             PC1      PC2      PC3       PC4
## Standard deviation     324.1502 212.7478 73.87622 4.189e-14
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```
plot(pca)
```



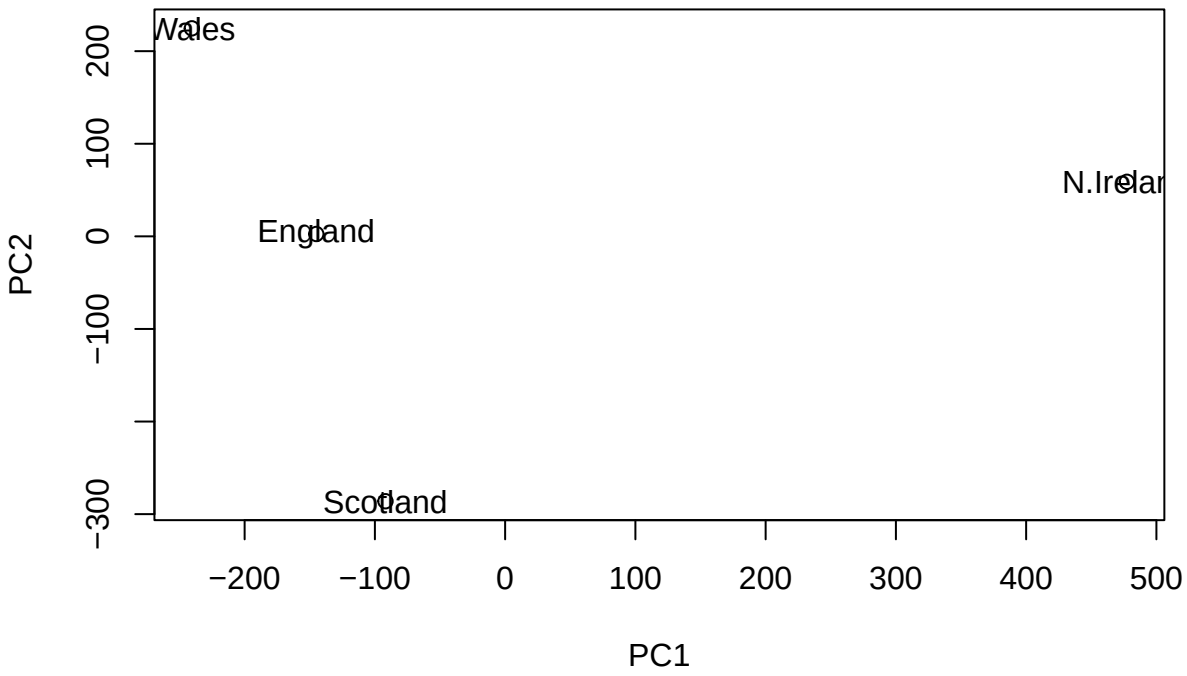
we want score plot (a.k.a PCS plot). Basically of PC1 vs PC2

```
attributes(pca)
```

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

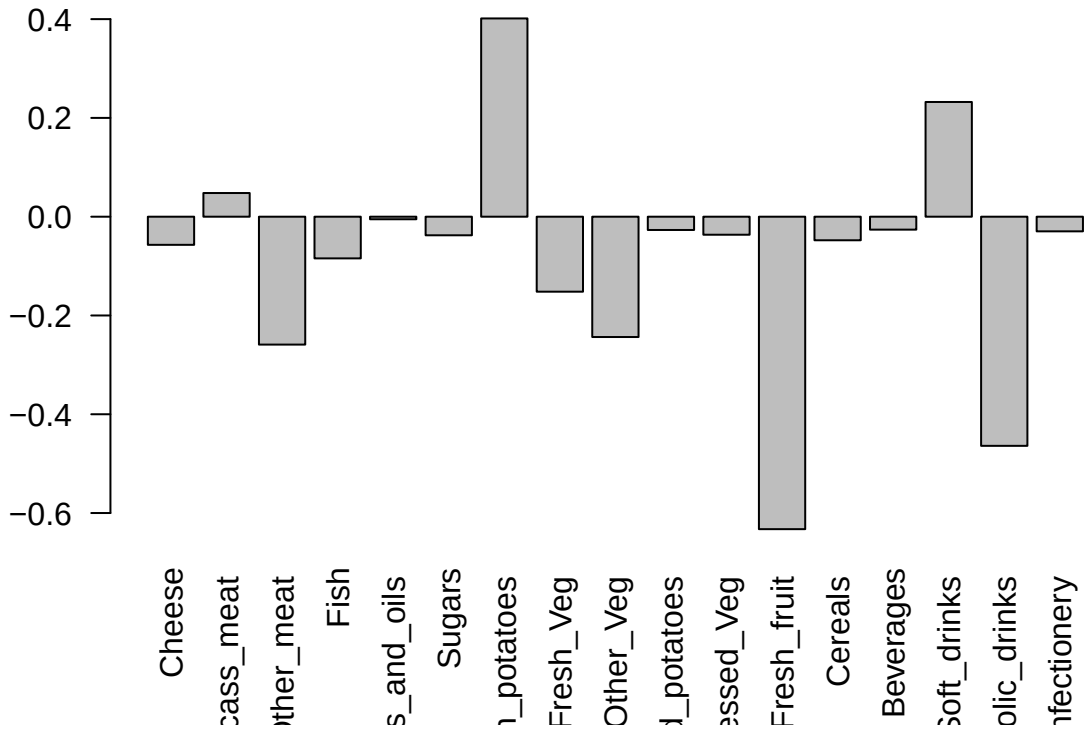
we are after the pcs$x component for this plot

```
plot(pca$x[,1:2])
text(pca$x[,1:2], labels = colnames(x))
```



we can also examine the PCA "loadings", which tell us how much the original variables

```
barplot(pca$rotation[,1], las=2)
```



##One more PCS for today

```
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <- read.csv(url2, row.names=1)
head(rna.data)
```

```
##        wt1 wt2  wt3  wt4 wt5 ko1 ko2 ko3 ko4 ko5
## gene1  439 458  408  429 420  90  88  86  90  93
## gene2  219 200  204  210 187 427 423 434 433 426
## gene3 1006 989 1030 1017 973 252 237 238 226 210
## gene4  783 792  829  856 760 849 856 835 885 894
## gene5  181 249  204  244 225 277 305 272 270 279
## gene6  460 502  491  491 493 612 594 577 618 638
```

```
nrow(rna.data)
```

```
## [1] 100
```

```
ncol(rna.data)
```

```
## [1] 10
```

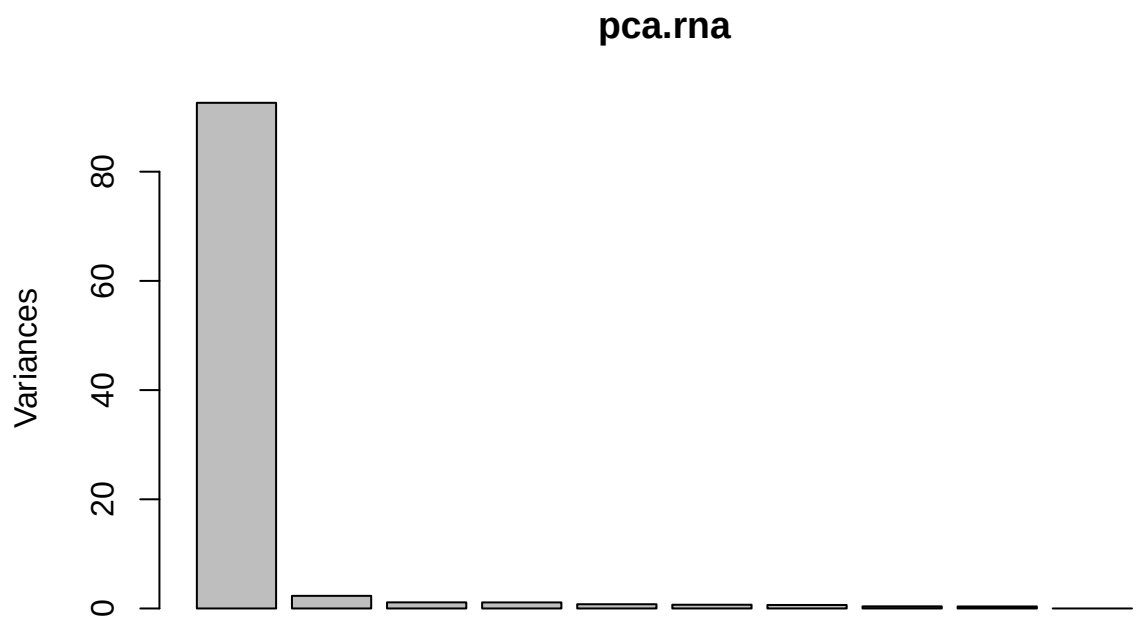
```
colnames(rna.data)
```

```
##  [1] "wt1" "wt2" "wt3" "wt4" "wt5" "ko1" "ko2" "ko3" "ko4" "ko5"
```

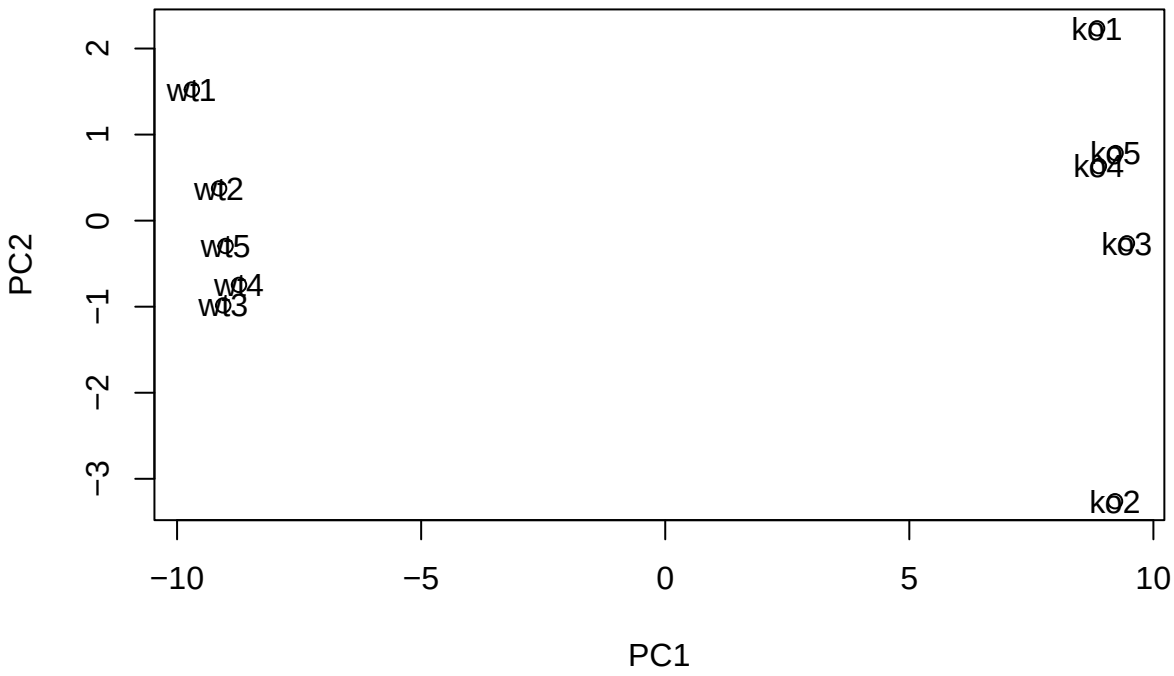
```
pca.rna <- prcomp( t(rna.data), scale=TRUE )
summary(pca.rna)
```

```
## Importance of components:
##                            PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.6237 1.5198 1.05787 1.05203 0.88062 0.82545 0.80111
## Proportion of Variance 0.9262 0.0231 0.01119 0.01107 0.00775 0.00681 0.00642
## Cumulative Proportion  0.9262 0.9493 0.96045 0.97152 0.97928 0.98609 0.99251
##                            PC8     PC9      PC10
## Standard deviation     0.62065 0.60342 3.348e-15
## Proportion of Variance 0.00385 0.00364 0.000e+00
## Cumulative Proportion  0.99636 1.00000 1.000e+00
```

```
plot(pca.rna)
```




```
plot(pca.rna$x[,1:2])
text(pca.rna$x[,1:2], labels=colnames(rna.data))
```


ko1
wt1
ko5
ko4
wt2
wt5
ko3
wt4
wt3
ko2
PC2
PC1
−10
−5
0
5
10
−3
−2
−1
0
1
2