Class 7: Machine Learning I

Nathaniel Nono (PID: A16782656)

Today we are going to leanr how to apply different machine learning methods

In-Class Section

Clustering

The goal here is to find groups/clusters in your data

`rnorm` function:

- Description: Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.
- Arguments:

`n` = number of observations. If length(n) > 1, the length is taken to be the number required

`mean` = vector of means

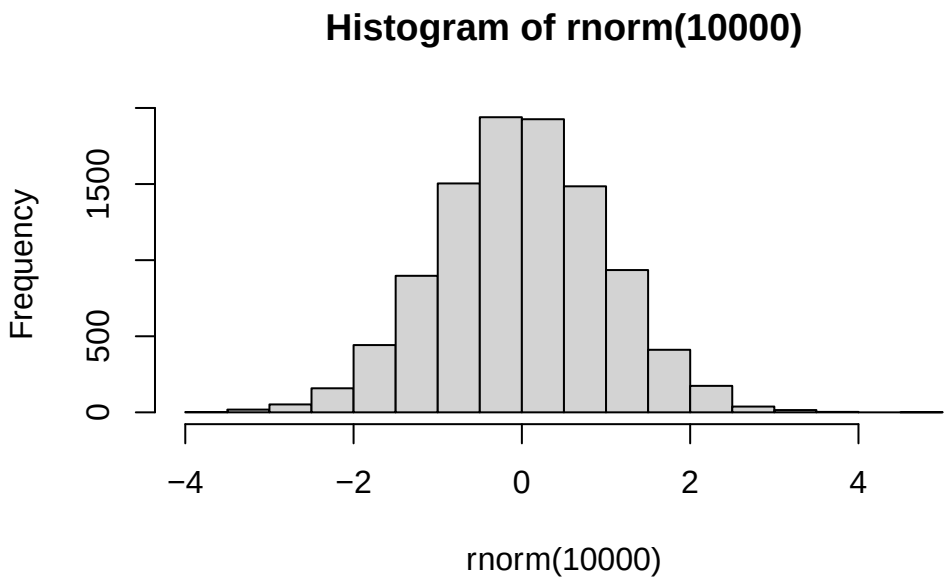
`sd` = vector of standard deviations

First I will make up some data with clear groups using the `rnorm` function

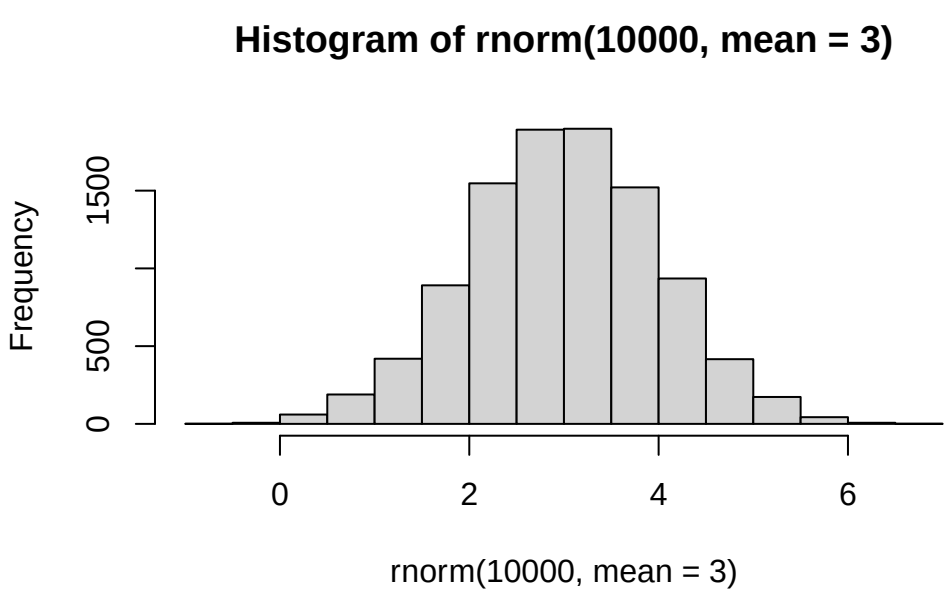
```
# 10 points of data
rnorm(10)
```

```
 [1]  0.5437189  1.0947691 -2.0716404  0.3461874 -1.0921275 -1.1606292
 [7] -1.0949762 -0.4194762  0.4460611 -0.9391080
```

```
#10,000 points of data and turning it into a histogram
hist(rnorm(10000))
```

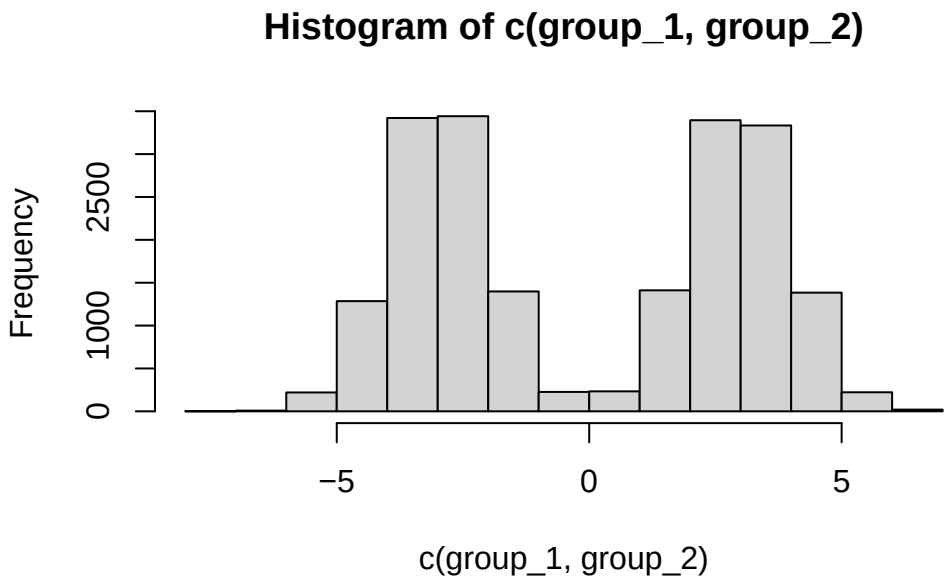



```
# Changing the center of the distribution to +3
hist(rnorm(10000, mean=3))
```



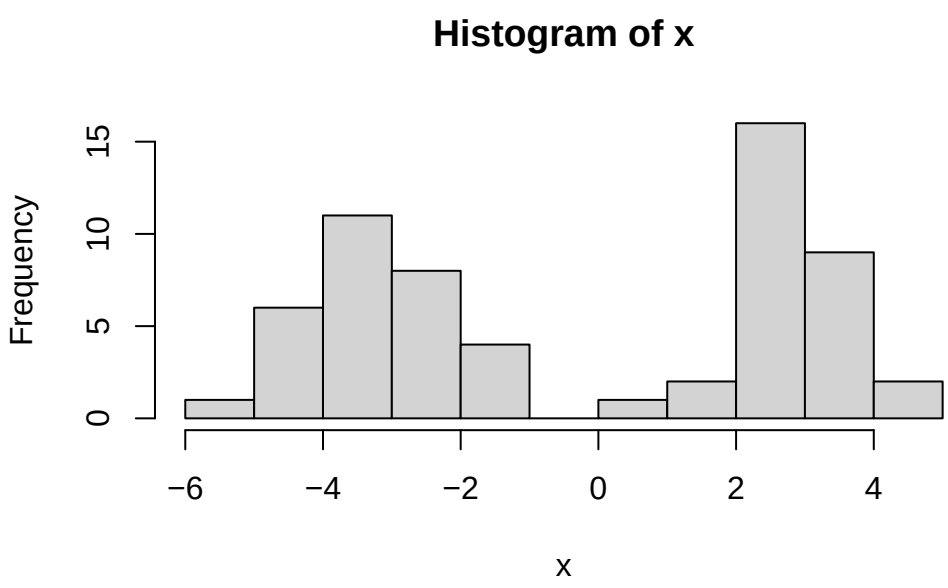
Make a vector with two peaks - Peak one = -3 - Peak two = +3

```r
group_1 <- rnorm(10000, mean=-3)
group_2 <- rnorm(10000, mean=3)
hist(c(group_1, group_2))
```




```r
# Alternative code (Uncheck to use):
# n <- 10000
# x <- c(rnorm(n, -3), rnorm(n, +3))
# hist(x)
```

```r
n <- 30
x <- c(rnorm(n, -3), rnorm(n, +3))
hist(x)
```




```
x
```

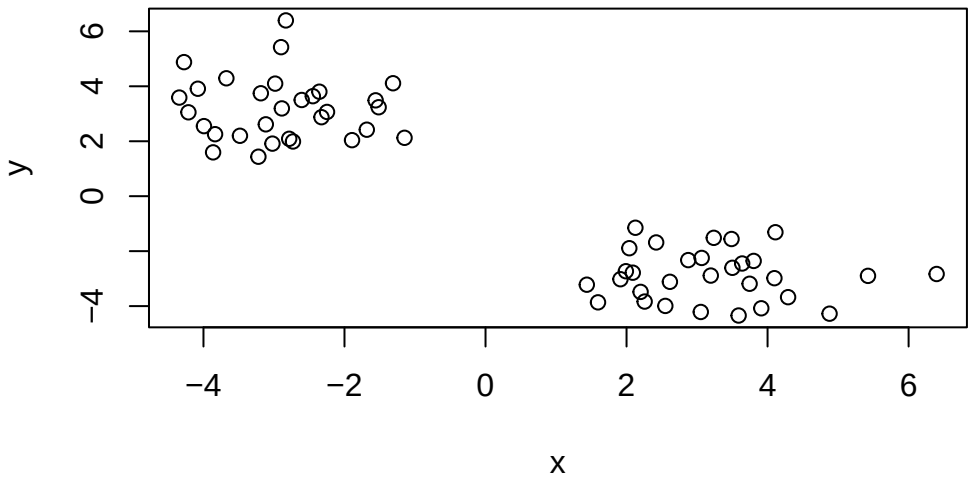
```
 [1] -4.4428483 -1.9039042 -3.6388306 -3.2664249 -4.0142334 -2.9851547
 [7] -3.3621190 -3.9674922 -3.2881689 -4.1542193 -3.2922257 -2.2499894
[13] -3.6768339 -3.6671017 -2.0808463 -3.2430802 -4.7464445 -4.7679244
[19] -2.6691196 -2.3070861 -3.5611817 -1.9340926 -2.0387202 -2.7107661
[25] -1.9128175 -1.3442168 -2.6171445 -4.2413823 -5.1279967 -3.3839093
[31]  3.0242548  1.4774144  2.5868132  4.4260515  2.5046395  2.4579156
[37]  2.2356827  2.8103161  2.5577242  2.3940157  2.7895956  2.3215096
[43]  1.7495177  3.0654416  3.2361465  2.9070313  4.1863418  3.7607838
[49]  3.1750910  3.2319922  0.3537028  2.6956532  2.3157631  3.0623092
[55]  2.4230248  3.4568104  2.9090461  2.8832368  2.1845829  3.2783394
```

```
n <- 30
x <- c(rnorm(n, -3), rnorm(n, +3))
# Reverses version of its argument
y <- rev(x)

# Takes the x and y coordinates and
z <- cbind(x, y)
head(z)
```

```
          x        y
```

```
[1,] -3.115865 2.615018
[2,] -2.606624 3.501089
[3,] -3.482803 2.199159
[4,] -3.675184 4.291211
[5,] -4.079642 3.910767
[6,] -2.730712 1.990641
```

```
plot(z)
```



K-Means

Use the `kmeans()` function setting k to 2 and nstart=20

Inspect/print the results

> Q. How many points are in each cluster?
>
> Q. What component of your result object details - Cluster size? - Cluster assignment/member - Cluster center?
>
> Q. Plot z colored by the kmeans colored by the kmeans cluster assignment and cluster centers as blue points

Tip: Use `?kmenas` to enter the help page. Focus on the arguments that do not have defaults - x = numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).

```
  - centers = either the number of clusters, say  , or a set of initial                    (d
```

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

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

Cluster means:
          x         y
1  3.184461 -2.884114
2 -2.884114  3.184461

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

Within cluster sum of squares by cluster:
[1] 62.91045 62.91045
 (between_SS / total_SS =  89.8 %)

Available components:

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

- Cluster means - Shows the means of each cluster. Shows how close each point is to the center
- **Cluster vector** - Shows how many points are in each cluster
- Sum of squares by cluster - Shows how "good" the kmenas are (i think)

Available components; Results in kmeans object `km`

```
attributes(km)
```

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

$class
[1] "kmeans"
```

What is the cluster size?

```
km$size
```

```
[1] 30 30
```

Cluster assignment/membership?

```
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
```

Cluster center

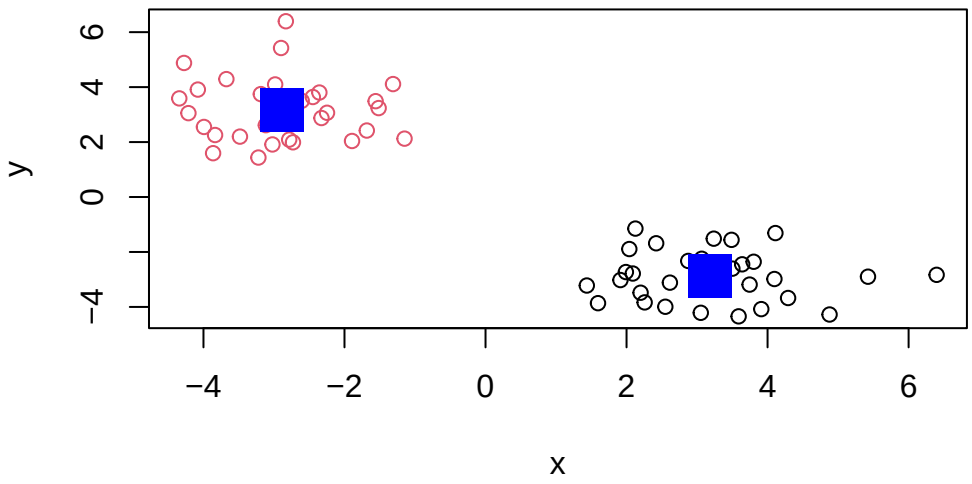
```
km$centers
```

```
          x         y
1  3.184461 -2.884114
2 -2.884114  3.184461
```

Q. Plot z colored by the kmeans colored by the kmeans cluster assignment and cluster centers as blue points

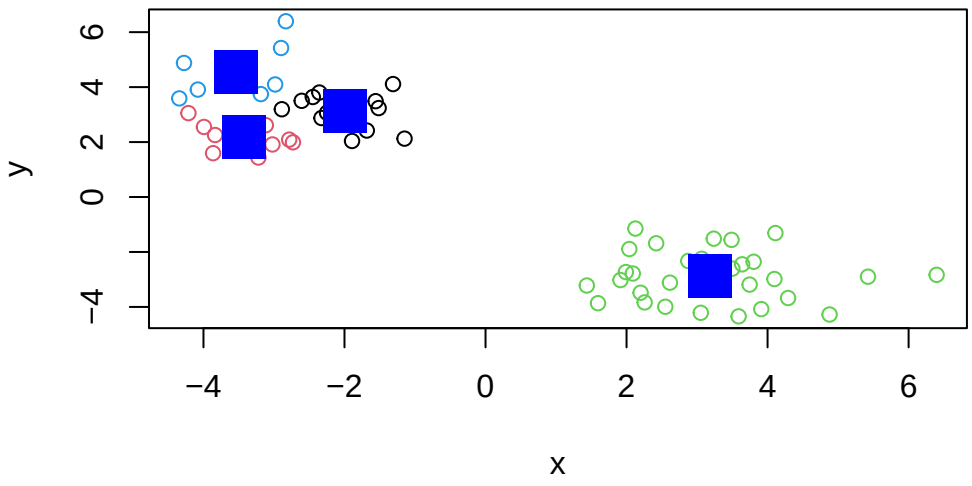
Remember, R will re-cycle the shorter color vector to be the same length as the longer (number of data points) in z

```
plot(z, col=km$cluster)
points(km$centers, col='blue', pch=15, cex=3)
```



Q. Can you run kmeans and ask for 4 cluster please and plot the results like we have done above? -> Problem because there is a lot of uncertainty

```
km4 <- kmeans(z, centers=4)
plot(z, col=km4$cluster)
points(km4$centers, col='blue', pch=15, cex=3)
```



How do you solve this problem?

- Scree plot - Systematically trying a range of different k values then finding the elbow point (The biggest seperation)

Hierarchical Clustering

Bottom up cluster - Starting up from the bottom then making a cluster at the top.

Let's take our same made-up data `z` and see how hclust works.

`hclust()` function needs a distance matrix

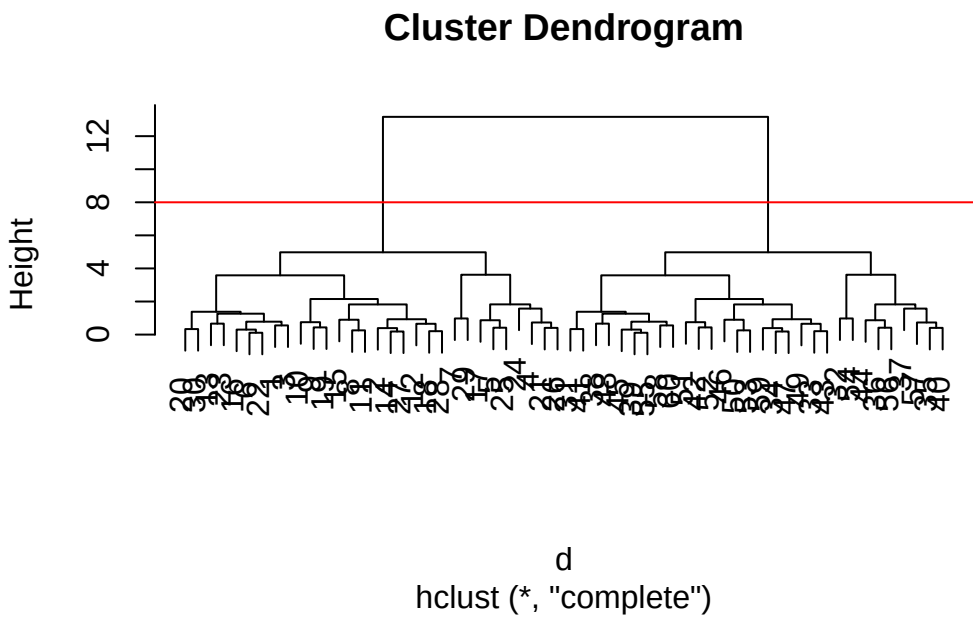
```
d <- dist(z)
#d

hc <- hclust(d)
hc
```

```
Call:
hclust(d = d)
```

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

Plotting the cluster dendrogram to show similarity

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



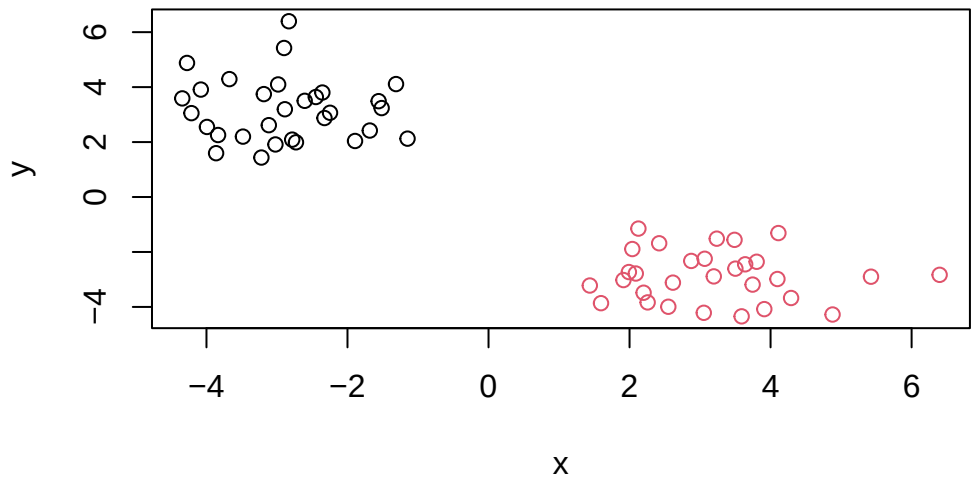
I can get my cluster membership vector by "cutting the tree" with the `cutree()` function like so. This will give us our clusters:

```
grps2 <- cutree(hc, h=8)
grps2
```

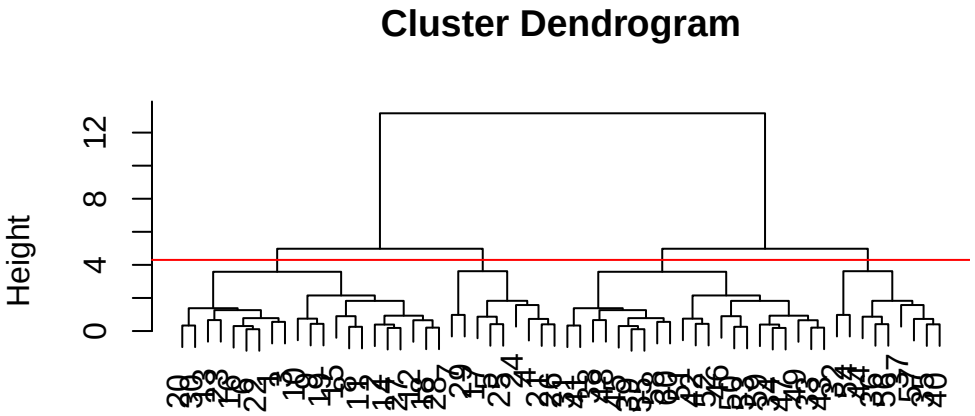
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Can you plot `z` colored by our hclust results:

```
plot(z, col=grps2)
```



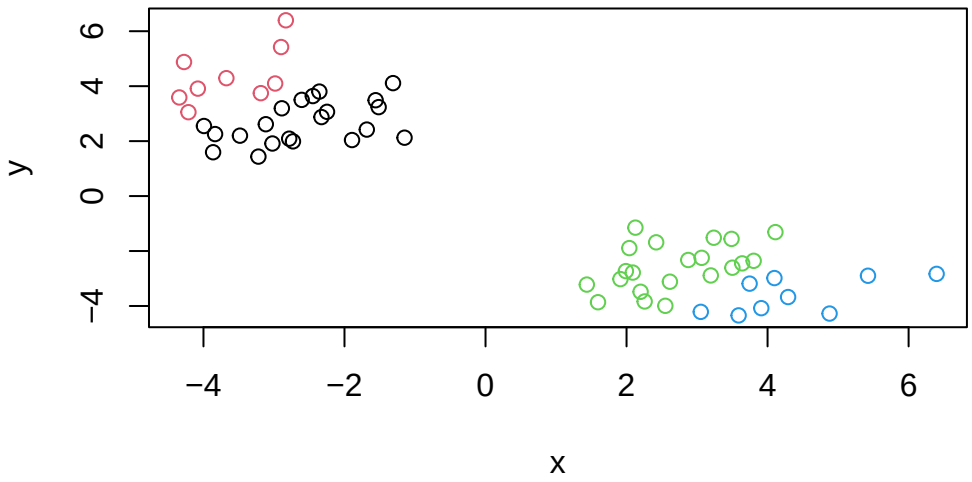

```
plot(hc)
abline(h=4.3, col='red')
```



```
grps4 <- cutree(hc, h=4.3)
grps4
```

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

```
plot(z, col=grps4)
```



Principtal Component Analysis (PCA)

Finding the dimensionaly reduction, visualization, and 'structure' analysis

Seen on the lab website

Lab Section

PCA of UK food data

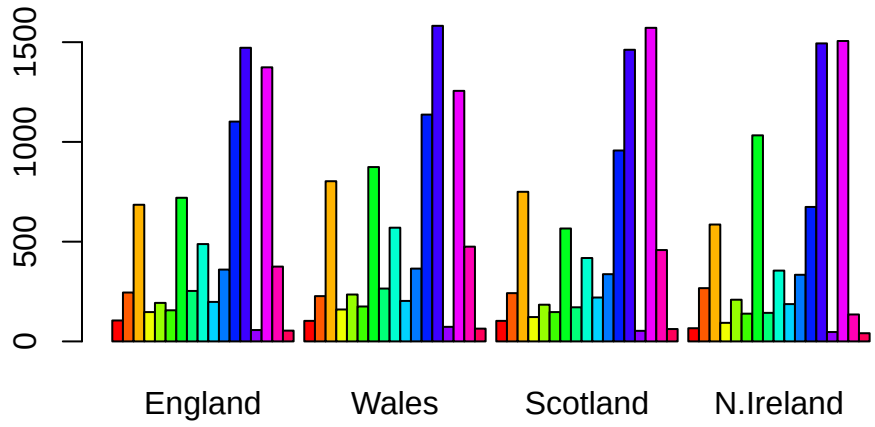
> Q1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?

Read data from the UK on food consumption in different parts of the UK

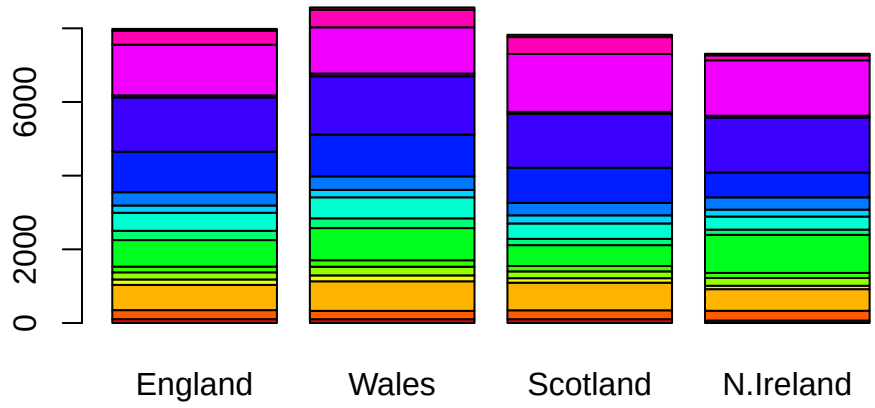
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names=1)

head(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
```

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

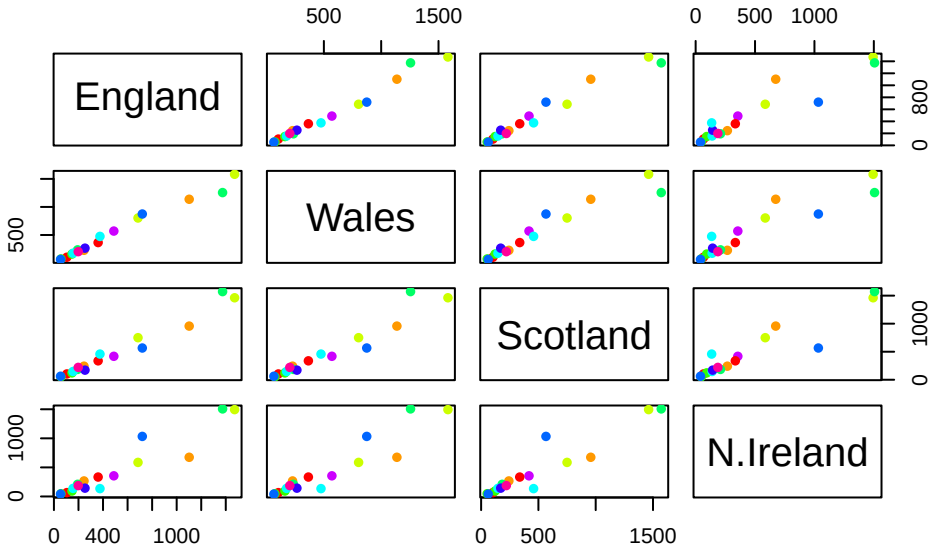



```
# beside = F; Stacked bar chart (More useless)
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



A so-called "Pairs" plot can be useful for small datasets like this one

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



It's hard to see structure and trends in even this small data set. How will we ever do this when we have big data sets with 1,000s or 10s of thousands of things we are measuring...

PCA to the Rescue

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

```
# transposing the values and performing a pca on it
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
```

- PC1: Captures 67.44% of the data

Let's see what is inside this `pca` object that we created from running `prcomp()`

```
attributes(pca)
```

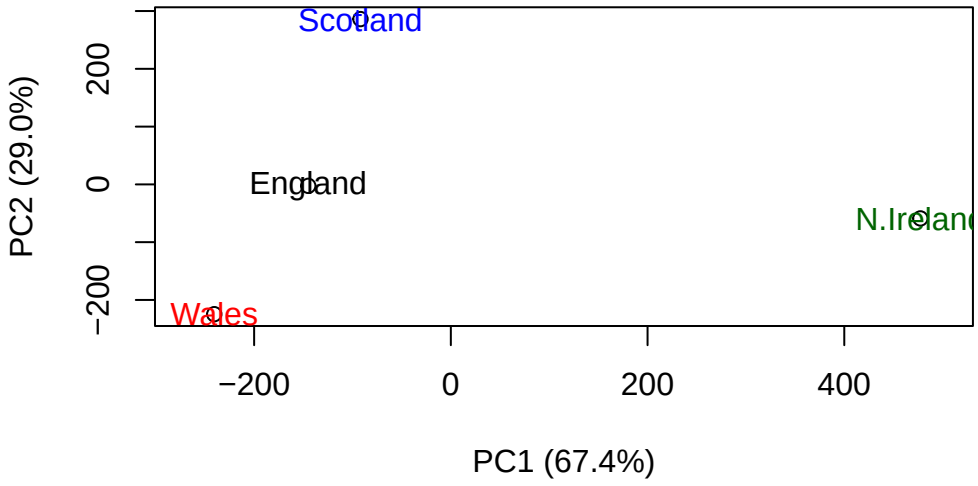
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```
pca$x
```

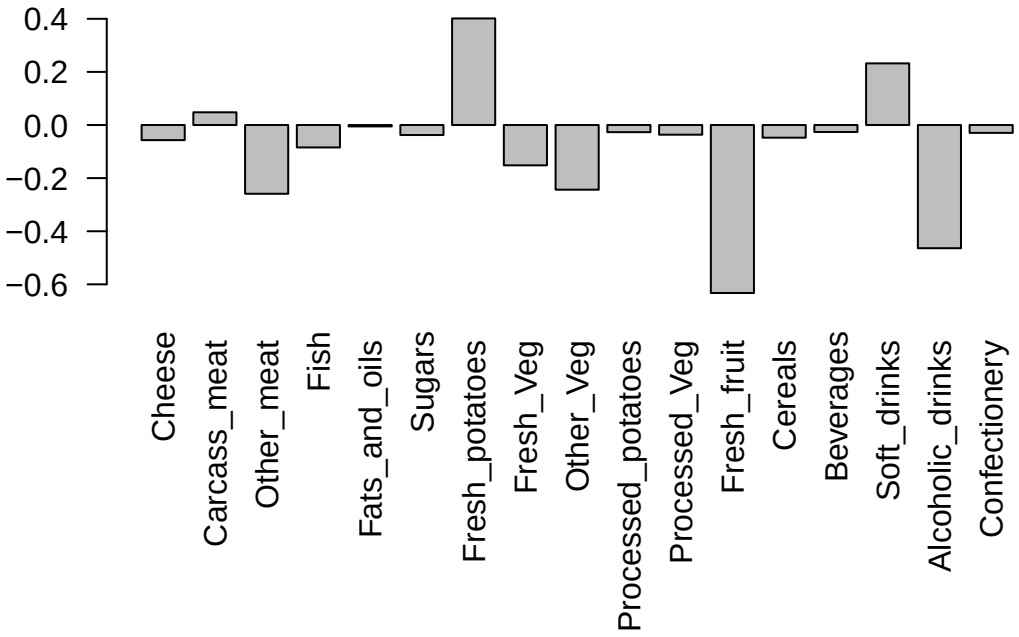
```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -9.152022e-15
Wales     -240.52915 -224.646925 -56.475555  5.560040e-13
Scotland   -91.86934  286.081786 -44.415495 -6.638419e-13
N.Ireland  477.39164  -58.901862  -4.877895  1.329771e-13
```

Want to get the first two columns and plot them against each other

```
plot(pca$x[,1], pca$x[,2], xlab="PC1 (67.4%)", ylab="PC2 (29.0%)", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x), col=c('black', 'red', 'blue', 'darkgreen'))
```




```
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```


0.4
0.2
0.0
−0.2
−0.4
−0.6
Cheese
Carcass_meat
Other_meat
Fish
Fats_and_oils
Sugars
Fresh_potatoes
Fresh_Veg
Other_Veg
Processed_potatoes
Processed_Veg
Fresh_fruit
Cereals
Beverages
Soft_drinks
Alcoholic_drinks
Confectionery