Class 7 BGGN213

AUTHOR
Lisanne Stouthart (PID A69036187)

Quarto

Quarto enables you to weave together content and executable code into a finished document. To learn more about Quarto see https://quarto.org.

Class - Machine Learning 1

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.

dnorm(x, mean = 0, sd = 1, log = FALSE) pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE) qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE) rnorm(n, mean = 0, sd = 1)

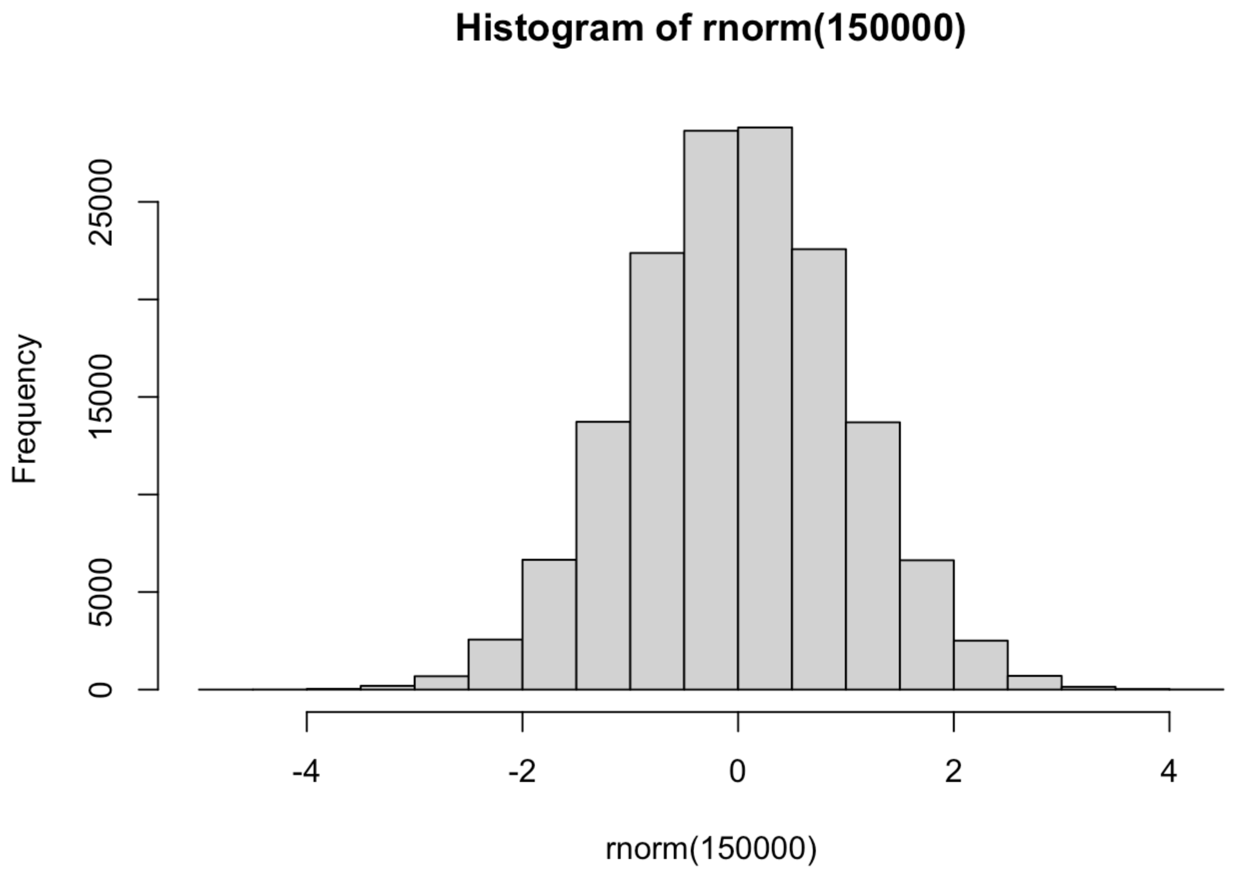
```
# rnorm()
# it has 3 arguments, where 2 out of the 3 arguments are fixed numbers.

rnorm(5)
```

```
[1] -0.04001234  0.68597051  1.07432277 -0.37158666  0.37560176
```

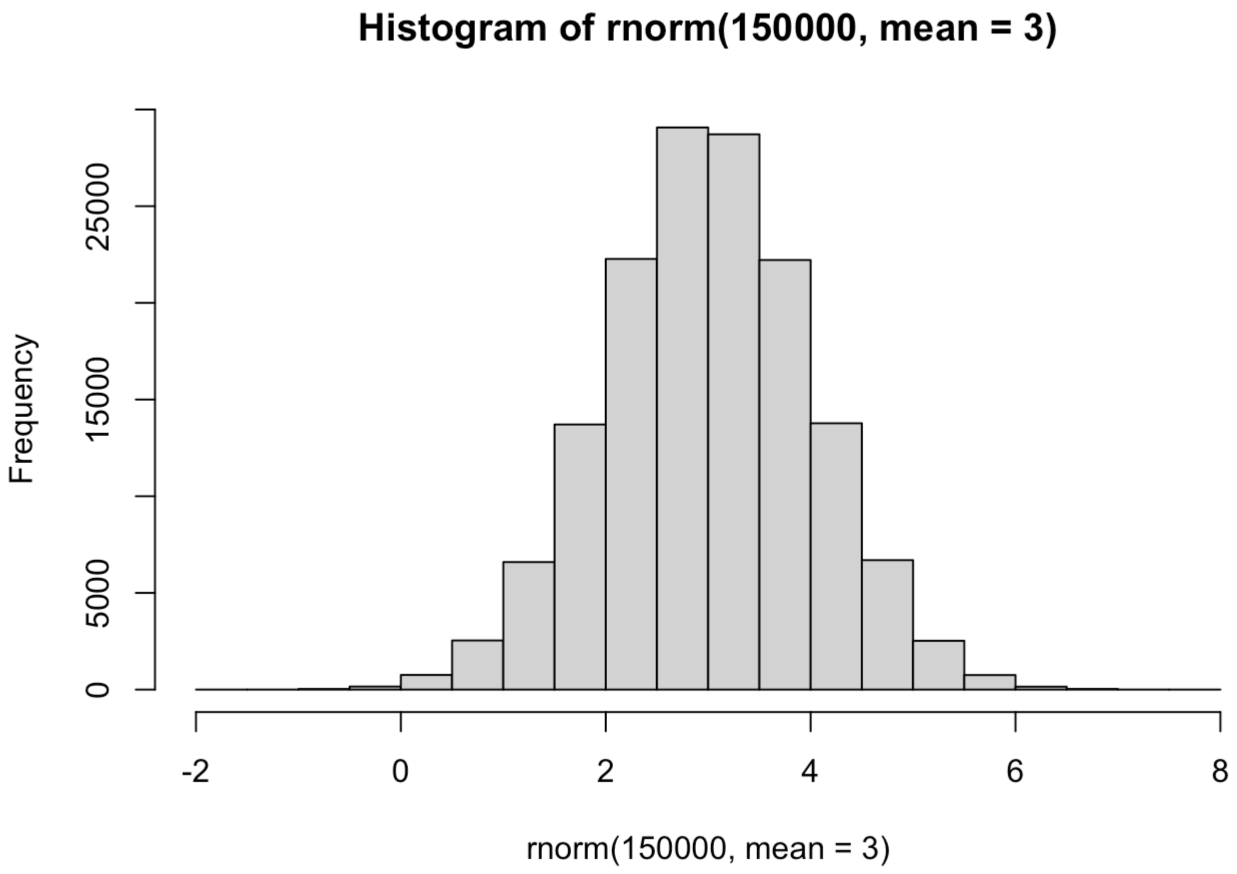
```
rnorm(15)
```

```
 [1] -0.83081203  0.36672117  1.52431777 -0.38404967 -1.27266604 -0.99357998
 [7] -0.40557806  0.01959658  1.48226903  0.80082539 -0.33246431  0.76634480
[13]  2.01612289  0.21182065 -0.99710142
```

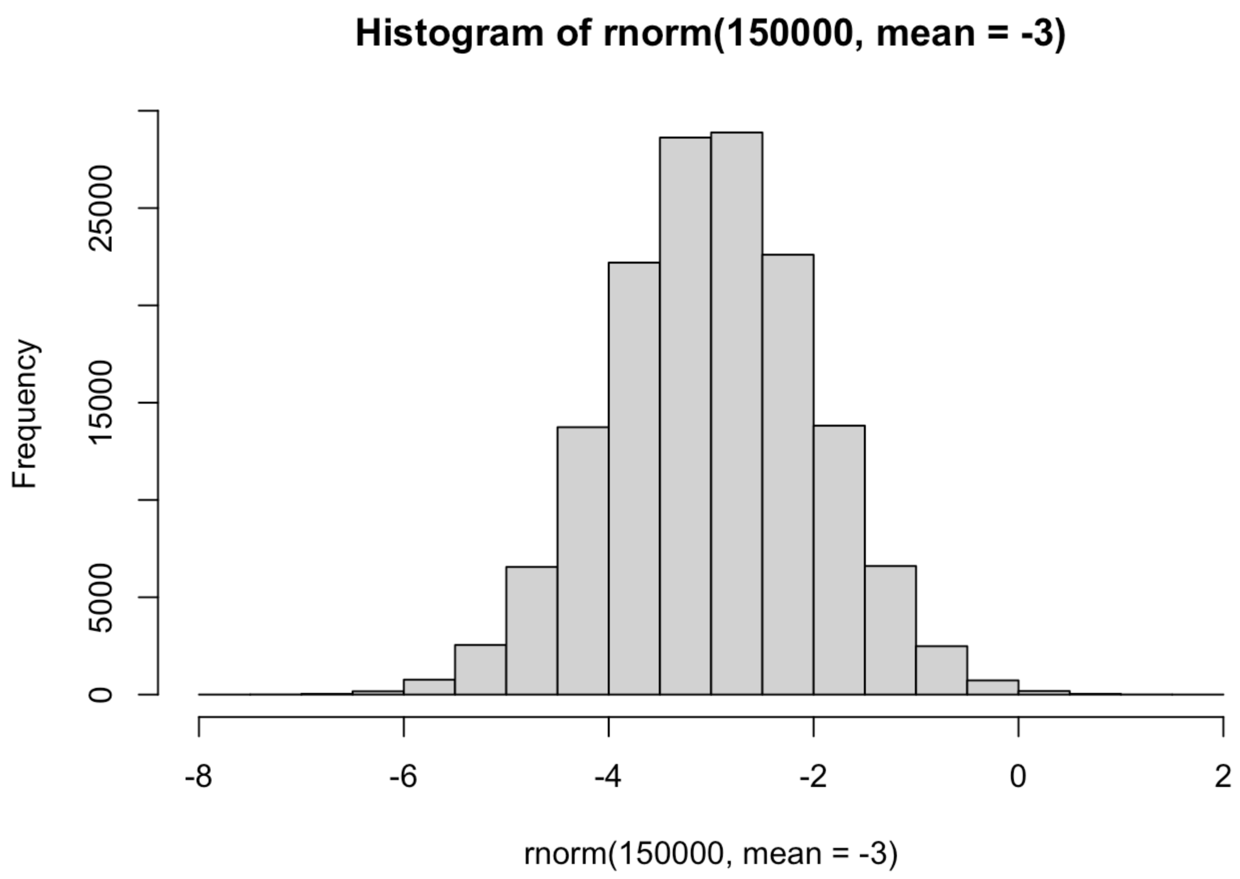
```
hist(rnorm(150000))
```



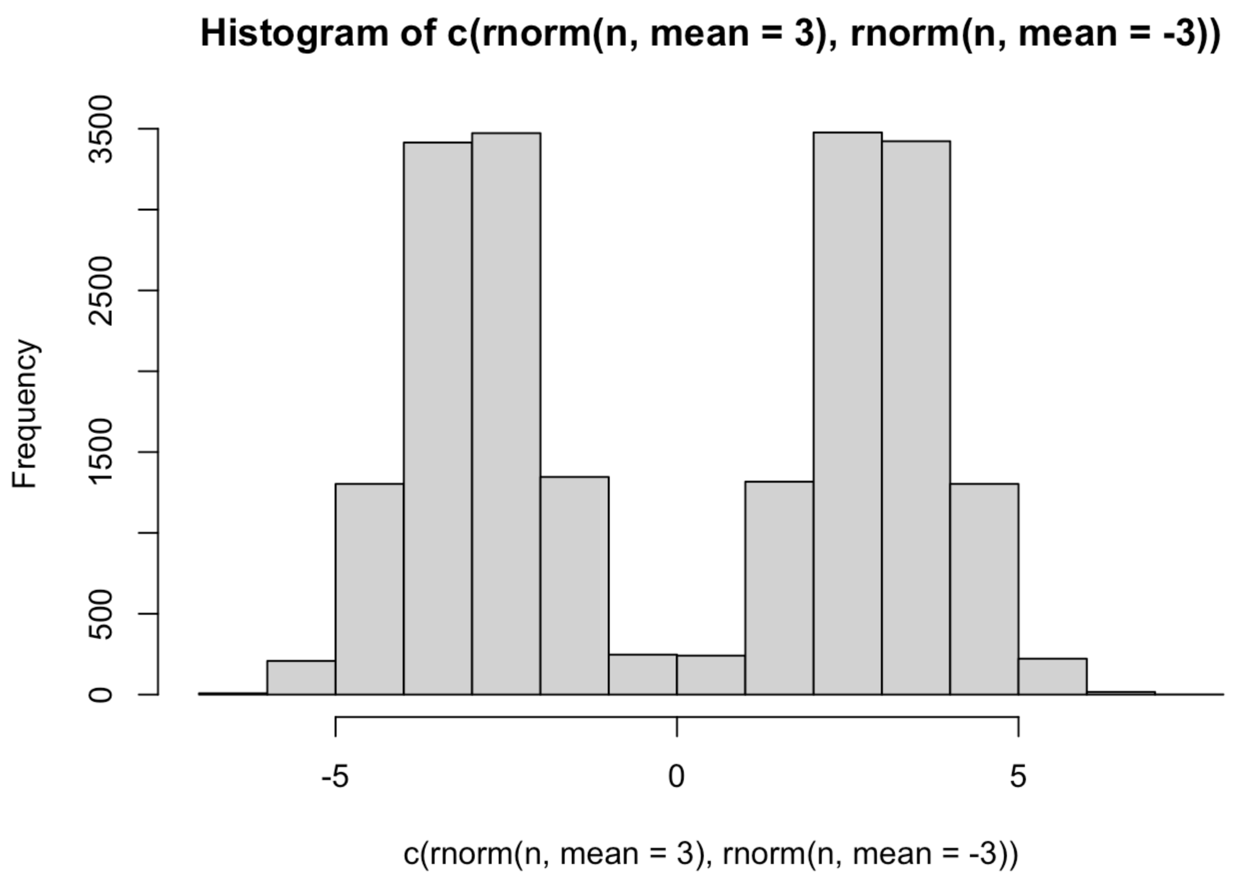
```r
hist(rnorm(150000, mean=3))
```



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



```r
n = 10000
hist(c(rnorm(n, mean=3), rnorm(n, mean=-3)))
```



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

```
 [1]  2.3875991  2.1519626  5.1819411  2.8475886  2.9156329  2.5451940
 [7]  1.3406880  2.7783780  4.6916243  3.6794071  2.2605492  3.8451309
[13]  1.2771424  2.8129824  2.4945789  1.5244540  4.2013811  0.9510227
[19]  4.7722170  3.6147070  3.1691506  2.2369681  3.5414158  2.2059774
[25]  2.6149123  3.5499059  2.1242428  3.1467887  2.5193566  4.2848344
[31] -4.4012033 -4.2292934 -3.2462846 -1.8146888 -4.3372222 -3.1793014
[37] -1.8153119 -3.6881381 -3.4270941 -3.6013313 -3.8240825 -4.6392731
[43] -2.2353999 -2.9303139 -3.2010897 -1.9443771 -5.0248378 -4.4468318
[49] -2.3281949 -2.6903217 -3.1669577 -2.6235725 -3.3530595 -2.2105007
[55] -2.8355858 -4.2625628 -4.0483448 -3.2362184 -3.5232899 -2.2982812
```

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

```
 [1]  2.4485957  2.8970112  3.6338778  2.9001816  2.9080854  3.6498481
 [7]  2.9218679  3.8375588  3.8146197  2.1855044  4.2121325  3.9143063
[13]  2.0037811  3.2844479  2.3215824  4.0638247  4.3940141  1.5648858
[19]  2.3756995  4.6339226  3.1604437  2.3822958  0.2377887  3.0933090
[25]  4.1708672  2.0677726  2.1395113  1.2399405  3.5271081  2.0080333
```

```
[31] -3.3041363 -2.8431167 -2.5284681 -4.1403048 -3.8434953 -2.7201755
[37] -1.7230558 -4.1083153 -3.3165389 -2.0127575 -4.3233424 -3.6131621
[43] -3.8747542 -2.3163600 -1.3576175 -2.1894470 -3.4805658 -1.0772638
[49] -1.2172450 -2.1500511 -2.5551595 -5.4395708 -2.1455182 -2.3519747
[55] -2.7612567 -1.8270459 -2.5132285 -5.6599396 -2.7580815 -2.5259105
```

```
y <- rev(x)
y
```

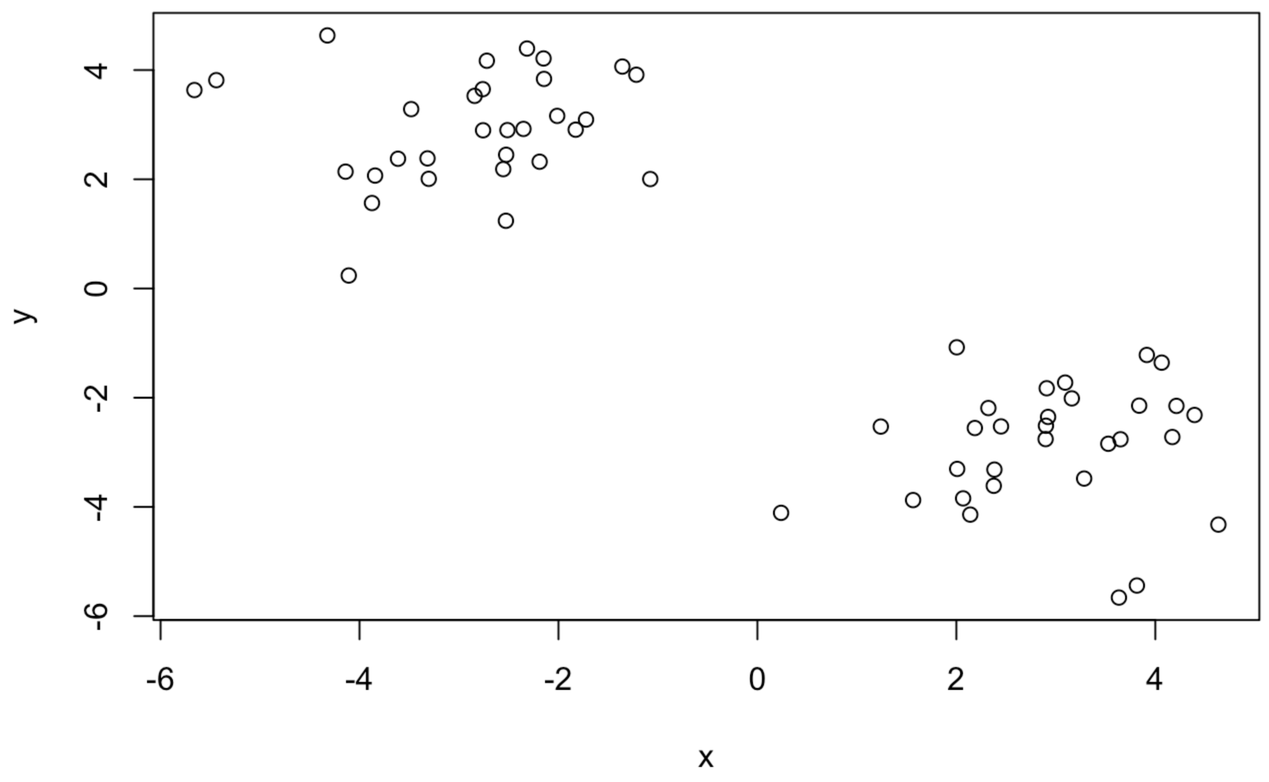
```
 [1] -2.5259105 -2.7580815 -5.6599396 -2.5132285 -1.8270459 -2.7612567
 [7] -2.3519747 -2.1455182 -5.4395708 -2.5551595 -2.1500511 -1.2172450
[13] -1.0772638 -3.4805658 -2.1894470 -1.3576175 -2.3163600 -3.8747542
[19] -3.6131621 -4.3233424 -2.0127575 -3.3165389 -4.1083153 -1.7230558
[25] -2.7201755 -3.8434953 -4.1403048 -2.5284681 -2.8431167 -3.3041363
[31]  2.0080333  3.5271081  1.2399405  2.1395113  2.0677726  4.1708672
[37]  3.0933090  0.2377887  2.3822958  3.1604437  4.6339226  2.3756995
[43]  1.5648858  4.3940141  4.0638247  2.3215824  3.2844479  2.0037811
[49]  3.9143063  4.2121325  2.1855044  3.8146197  3.8375588  2.9218679
[55]  3.6498481  2.9080854  2.9001816  3.6338778  2.8970112  2.4485957
```

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

```
              x          y
 [1,]  2.4485957 -2.5259105
 [2,]  2.8970112 -2.7580815
 [3,]  3.6338778 -5.6599396
 [4,]  2.9001816 -2.5132285
 [5,]  2.9080854 -1.8270459
 [6,]  3.6498481 -2.7612567
 [7,]  2.9218679 -2.3519747
 [8,]  3.8375588 -2.1455182
 [9,]  3.8146197 -5.4395708
[10,]  2.1855044 -2.5551595
[11,]  4.2121325 -2.1500511
[12,]  3.9143063 -1.2172450
[13,]  2.0037811 -1.0772638
[14,]  3.2844479 -3.4805658
[15,]  2.3215824 -2.1894470
[16,]  4.0638247 -1.3576175
[17,]  4.3940141 -2.3163600
[18,]  1.5648858 -3.8747542
[19,]  2.3756995 -3.6131621
[20,]  4.6339226 -4.3233424
[21,]  3.1604437 -2.0127575
[22,]  2.3822958 -3.3165389
[23,]  0.2377887 -4.1083153
[24,]  3.0933090 -1.7230558
[25,]  4.1708672 -2.7201755
```

```
[26,]  2.0677726 -3.8434953
[27,]  2.1395113 -4.1403048
[28,]  1.2399405 -2.5284681
[29,]  3.5271081 -2.8431167
[30,]  2.0080333 -3.3041363
[31,] -3.3041363  2.0080333
[32,] -2.8431167  3.5271081
[33,] -2.5284681  1.2399405
[34,] -4.1403048  2.1395113
[35,] -3.8434953  2.0677726
[36,] -2.7201755  4.1708672
[37,] -1.7230558  3.0933090
[38,] -4.1083153  0.2377887
[39,] -3.3165389  2.3822958
[40,] -2.0127575  3.1604437
[41,] -4.3233424  4.6339226
[42,] -3.6131621  2.3756995
[43,] -3.8747542  1.5648858
[44,] -2.3163600  4.3940141
[45,] -1.3576175  4.0638247
[46,] -2.1894470  2.3215824
[47,] -3.4805658  3.2844479
[48,] -1.0772638  2.0037811
[49,] -1.2172450  3.9143063
[50,] -2.1500511  4.2121325
[51,] -2.5551595  2.1855044
[52,] -5.4395708  3.8146197
[53,] -2.1455182  3.8375588
[54,] -2.3519747  2.9218679
[55,] -2.7612567  3.6498481
[56,] -1.8270459  2.9080854
[57,] -2.5132285  2.9001816
[58,] -5.6599396  3.6338778
[59,] -2.7580815  2.8970112
[60,] -2.5259105  2.4485957
```

```
# z <- rbind(x,y)
# z

plot(z)
```



K-means clustering

The function in base R (meaning that you dont have to install this) for k-means clustering is called ‘kmeans()’.

Perform k-means clustering on a data matrix. kmeans(x, centers, iter.max = 10, nstart = 1, algorithm = c(“Hartigan-Wong”, “Lloyd”, “Forgy”, “MacQueen”), trace = FALSE)

```
km <- kmeans(z, center = 2)
# is 30,30 because we set 30 points each dataset
# he has slightly different answers but that is because it is random
km
```

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

Cluster means:
          x         y
1  2.933094 -2.889262
2 -2.889262  2.933094

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] 67.34212 67.34212
 (between_SS / total_SS =  88.3 %)

Available components:

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

```
km$centers
```

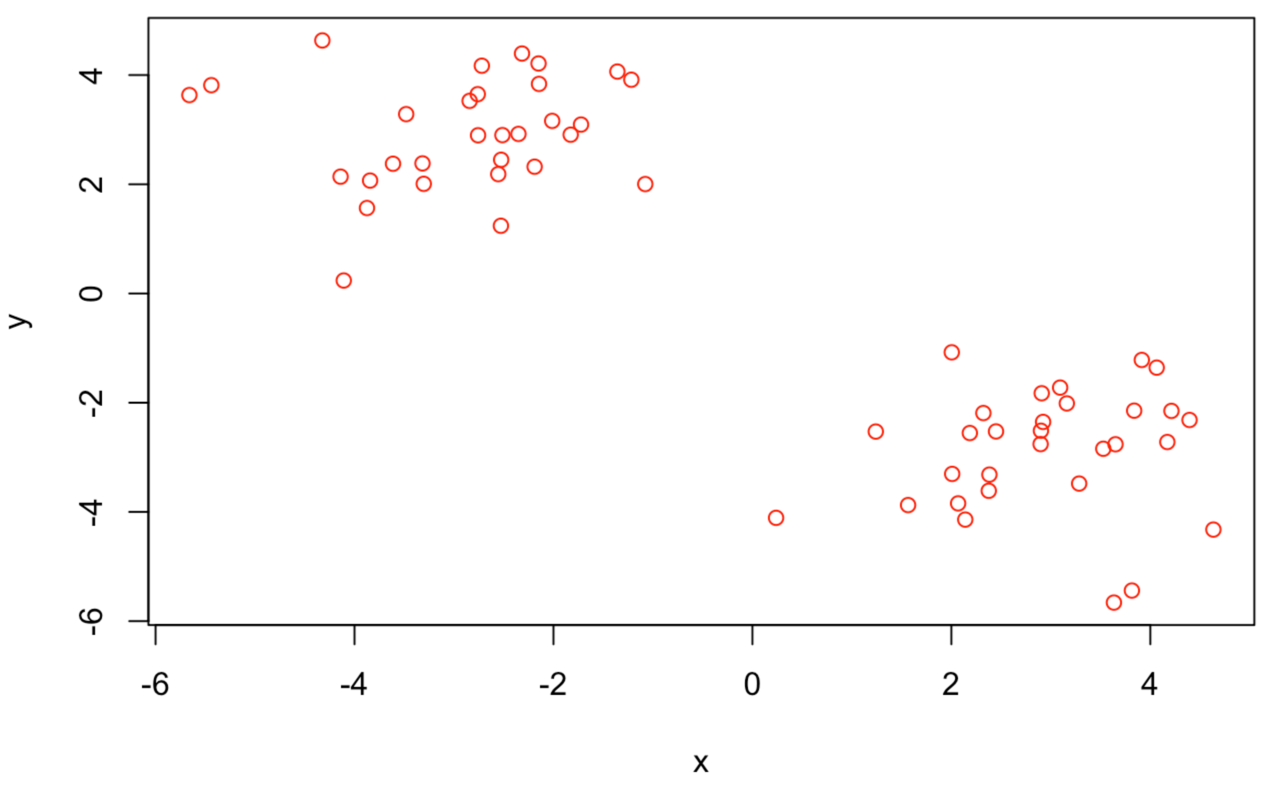
```
          x         y
1  2.933094 -2.889262
2 -2.889262  2.933094
```

```
km$cluster
```

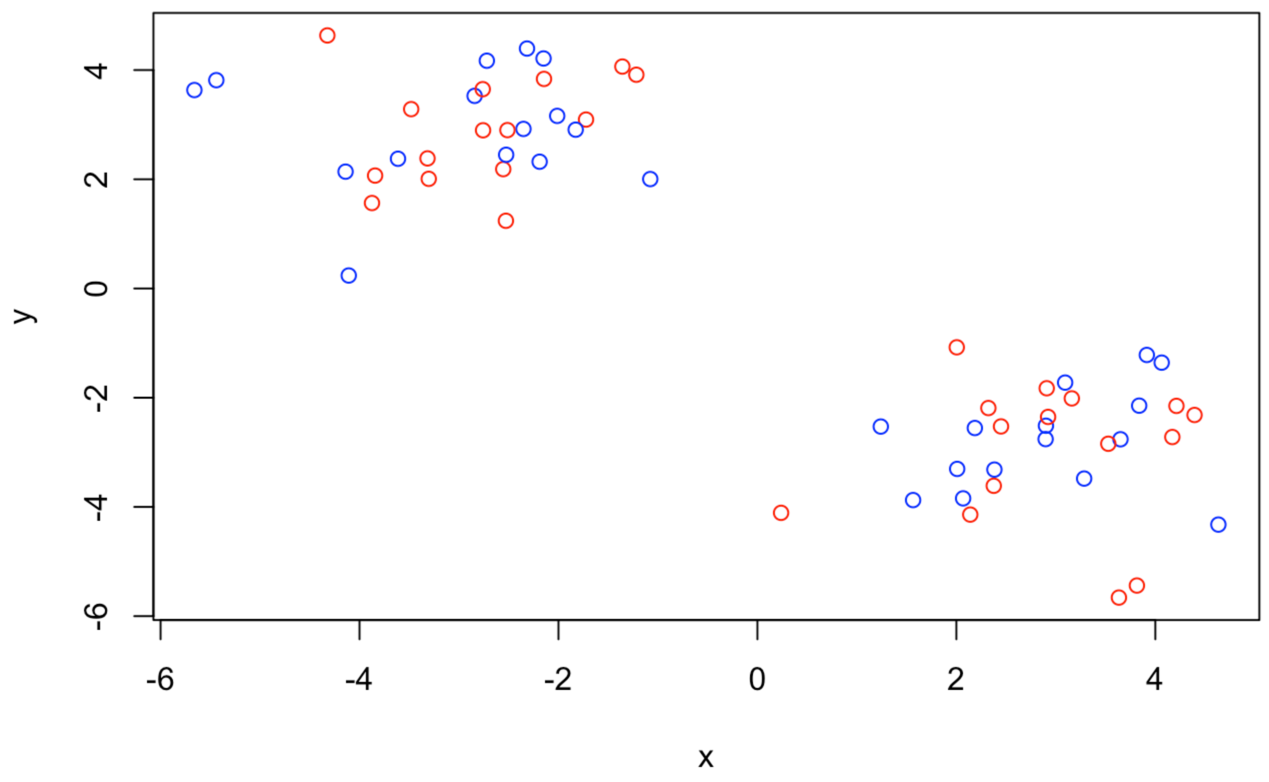
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Make color plot with clustering result and add cluster centers

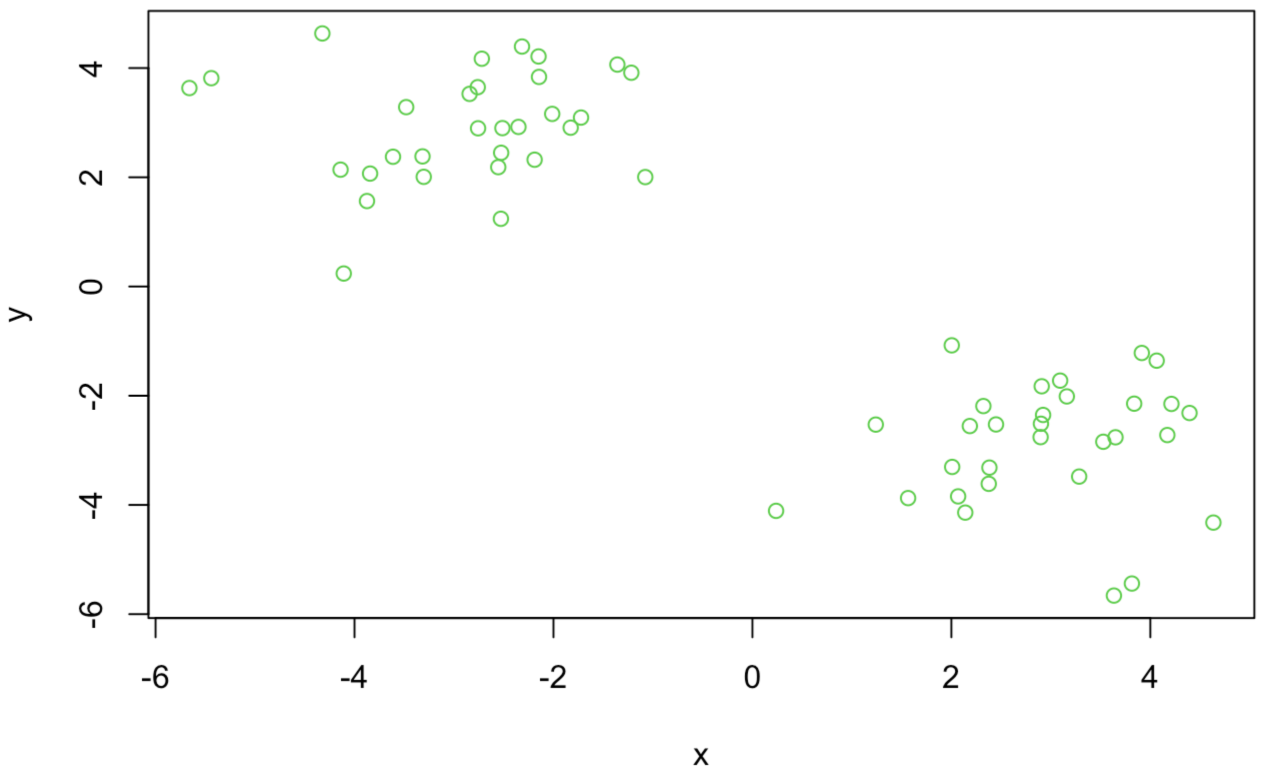
```
plot(z, col = "red")
```



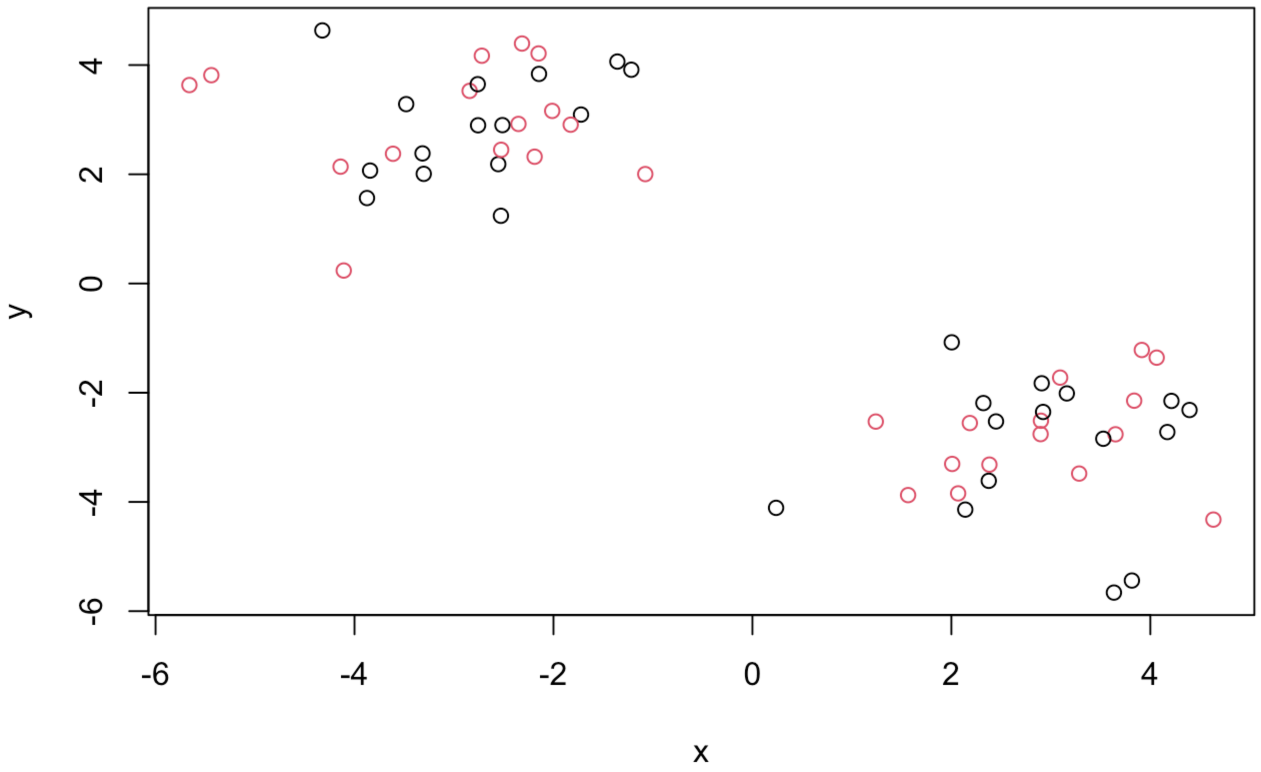
```r
plot(z, col=c("red", "blue"))
```



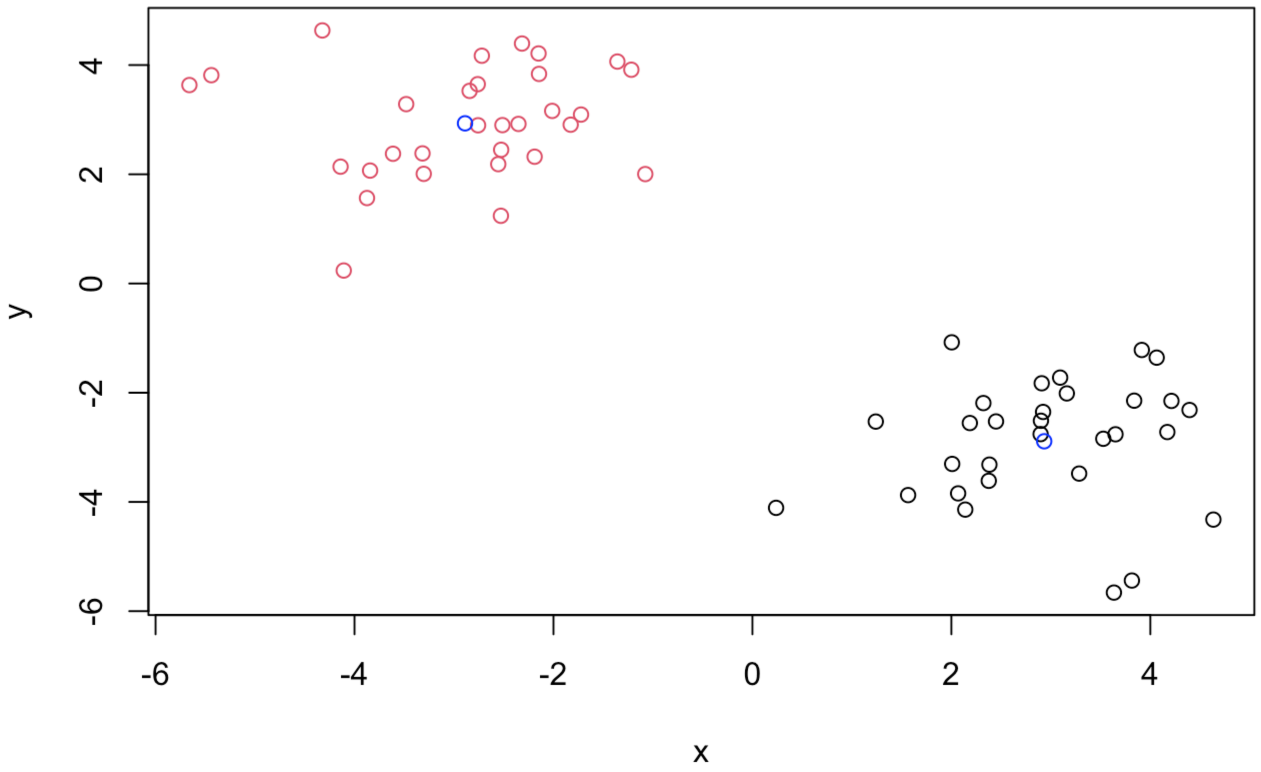
```
#this is random assigning of the blue and red, alternating

plot(z, col = 3) #green, color by number
```



```r
plot(z, col=c(1,2)) #green, color by number, random again
```

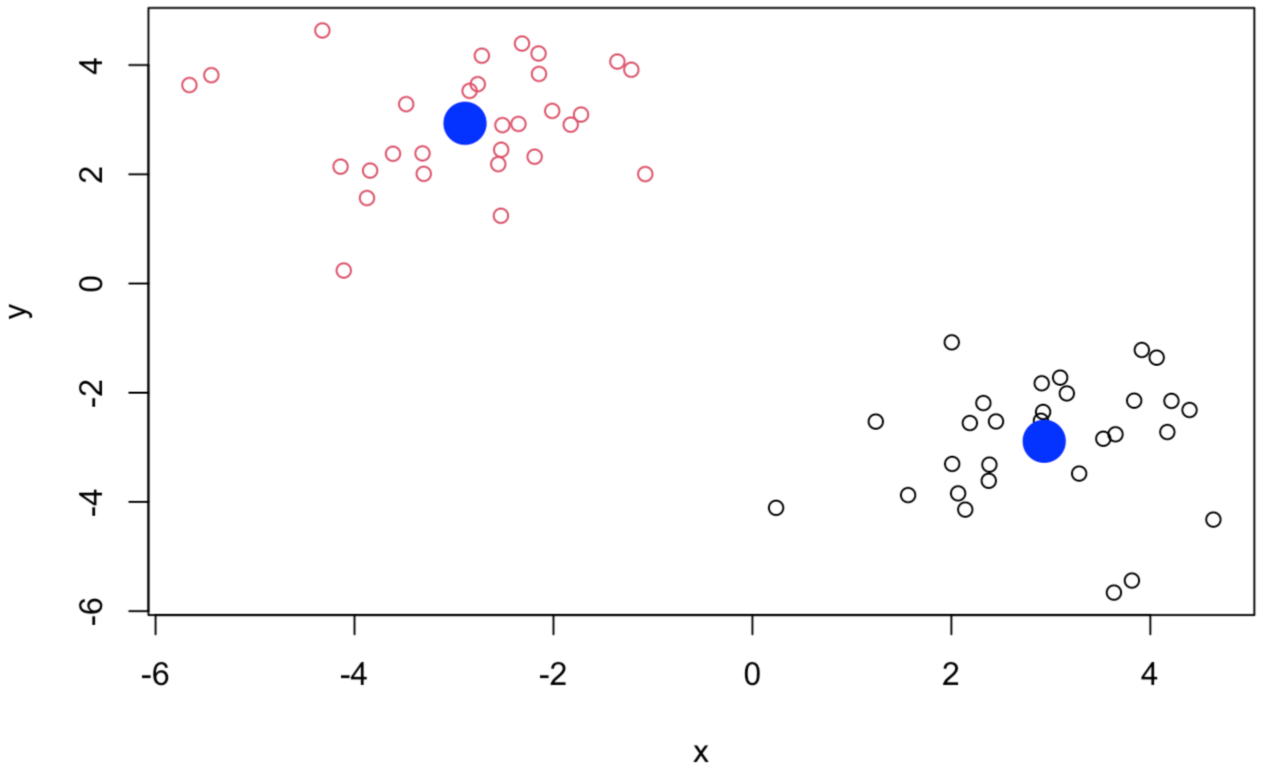


```
plot(z, col=km$cluster) #plot with clustering result

plot(z, col=km$cluster) +
points(km$centers, col="blue") #plot with clustering result and center result
```



```
integer(0)
```

```
plot(z, col=km$cluster) +
points(km$centers, col="blue", pch=16, cex=3) #to make center bit bigger
```

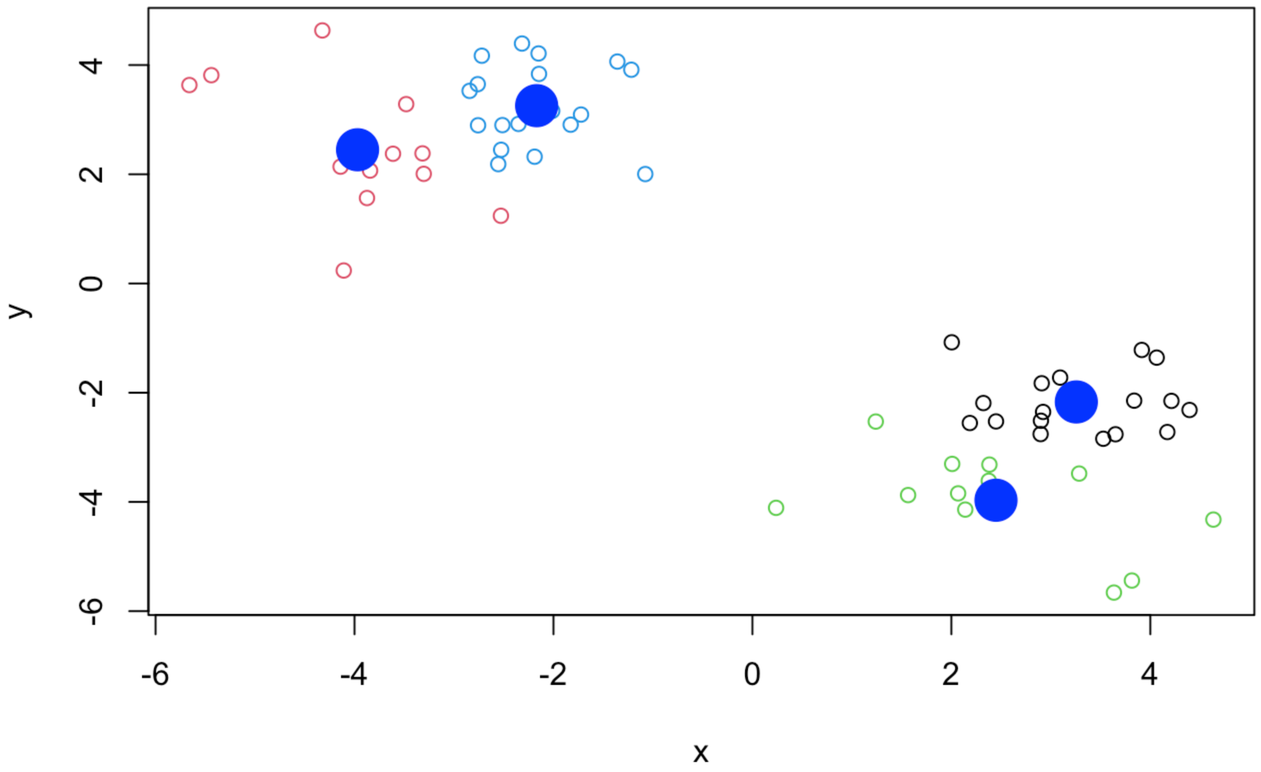


```
integer(0)
```

```
# if you play with number of pch, you can change the shape
# if you play with number of cex, you can change the size
```

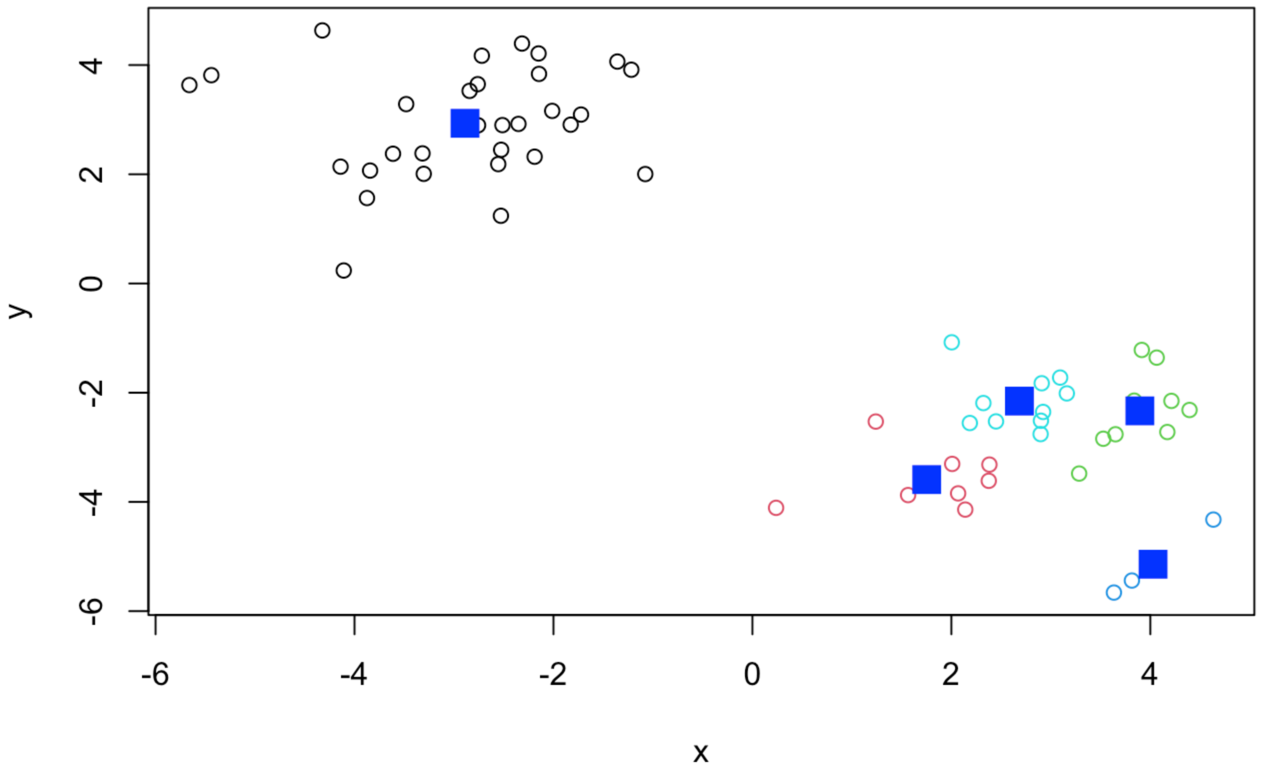
Can you cluster our data in ‘z’ into four clusters please?

```
km4 <- kmeans(z, center = 4)
plot(z, col=km4$cluster) +
  points(km4$centers, col="blue", pch=16, cex=3)
```



```
integer(0)
```

```
km5 <- kmeans(z, center = 5)
plot(z, col=km5$cluster) +
  points(km5$centers, col="blue", pch=15, cex=2)
```



```
integer(0)
```

Hierarchical Clustering

Hierarchical cluster analysis on a set of dissimilarities and methods for analyzing it. hclust(d, method = “complete”, members = NULL)

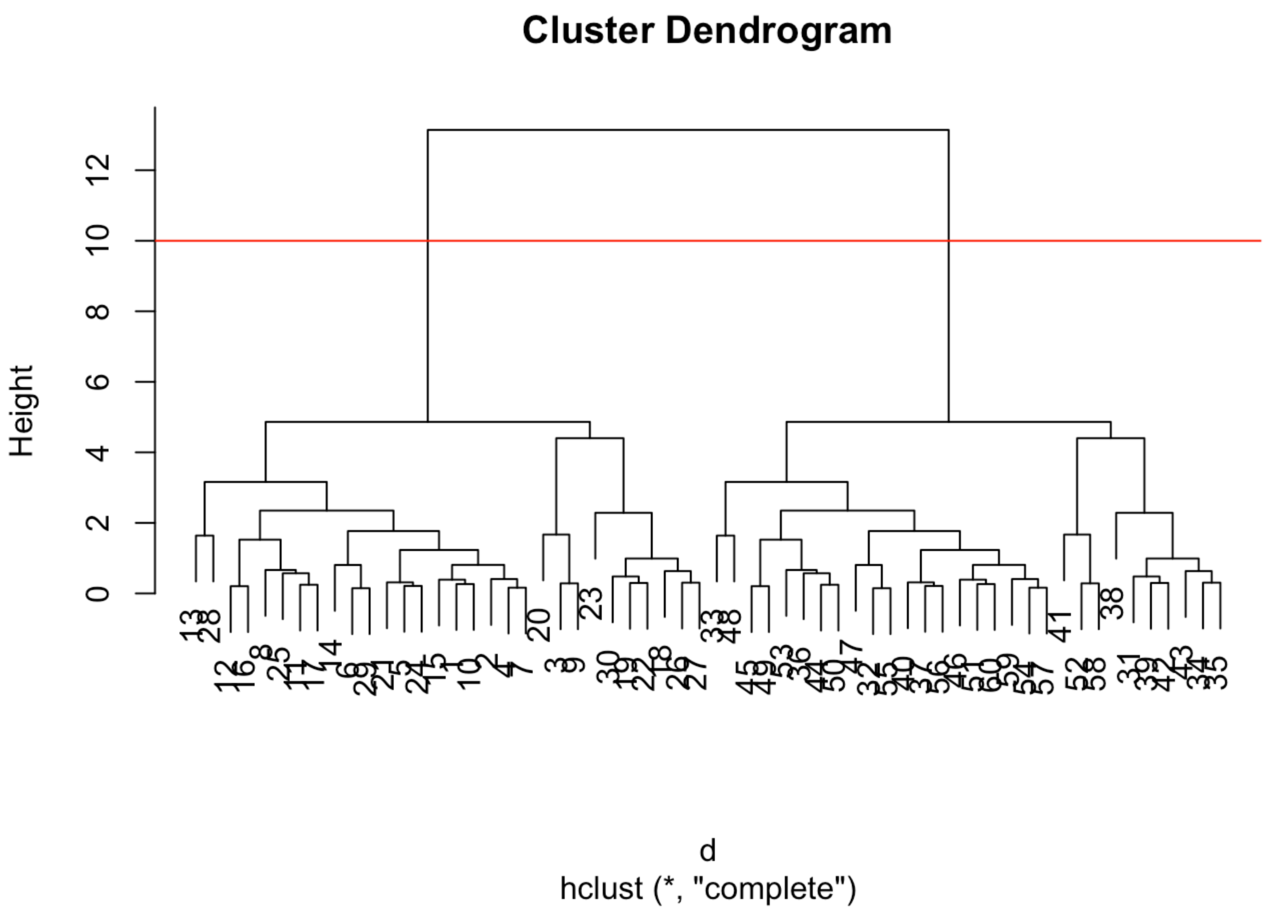
The main function for hierarchical clutering in base R is called ‘hclust()’. Unlike ‘kmeans()’ I can not just pass in my data as input I first need a distance matrix from my data.

```
d <-dist(z)
hc <- hclust(d)
hc # this is not really helpful
```

```
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")
```



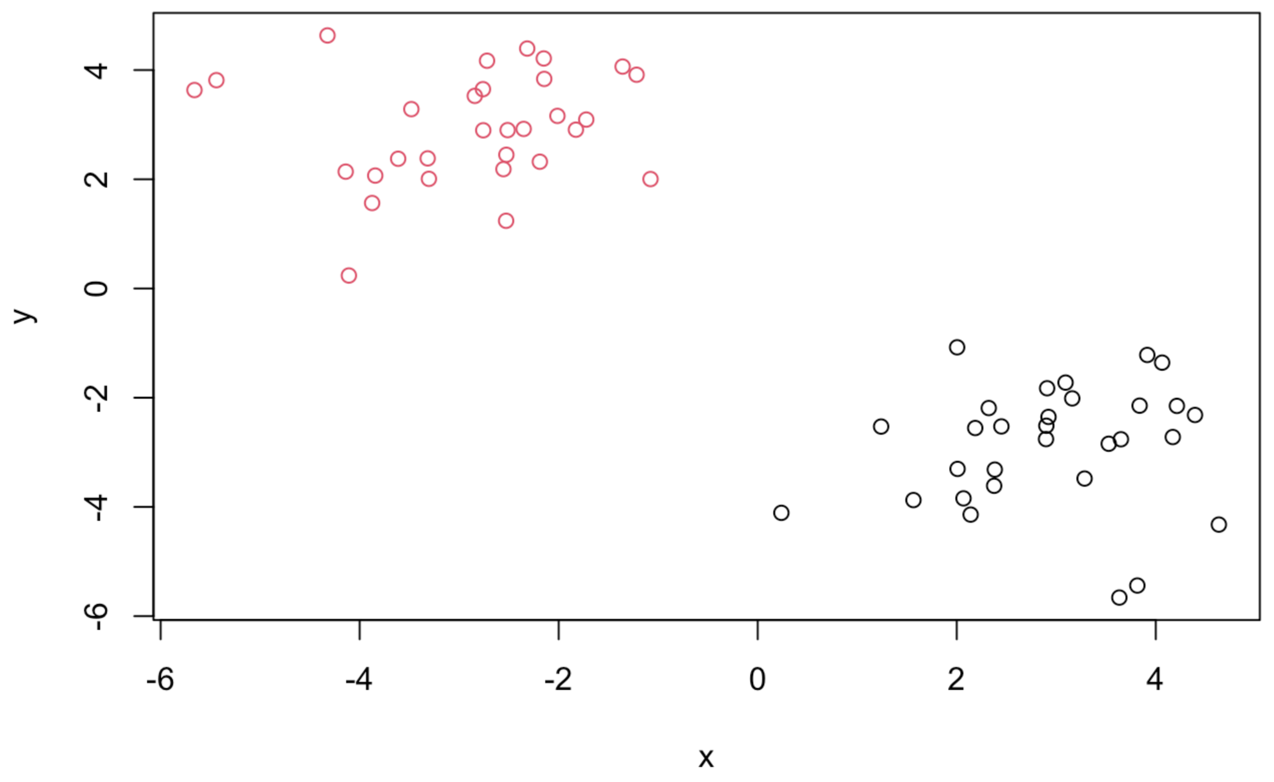
```
integer(0)
```

To get my main clustering result (i.e. the membership vector). I can "cut" my tree at a give height. To do 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 2 2
```

```r
plot(z, col=grps) #so 2 different ways to get the custering
```



Principal Component Analysis

Principal component analysis (PCA) is a well established “multivariate statistical technique” used to reduce the dimensionality of a complex data set to a more manageable number (typically 2D or 3D). This method is particularly useful for highlighting strong paterns and relationships in large datasets (i.e. revealing major similarities and diferences) that are otherwise hard to visualize. As we will see again and again in this course PCA is often used to make all sorts of bioinformatics data easy to explore and visualize.

PCA of UK food data

```
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
```

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

```
## Complete the following code to find out how many rows and columns are in x?
nrow(x) #answer = 17 rows
```

```
[1] 17
```

```
ncol(x) #answer = 5 columns
```

```
[1] 4
```

```
dim(x) #both combined
```

```
[1] 17  4
```

```
## Preview the first 6 rows
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
```

```
# Note how the minus indexing works
rownames(x) <- x[,1]
x <- x[,-1]
head(x)
```

```
    Wales Scotland N.Ireland
105   103      103        66
245   227      242       267
```

```
685   803      750       586
147   160      122        93
193   235      184       209
156   175      147       139
```

```
# Check dimensions again
dim(x) # answer is 17  4
```

```
[1] 17  3
```

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

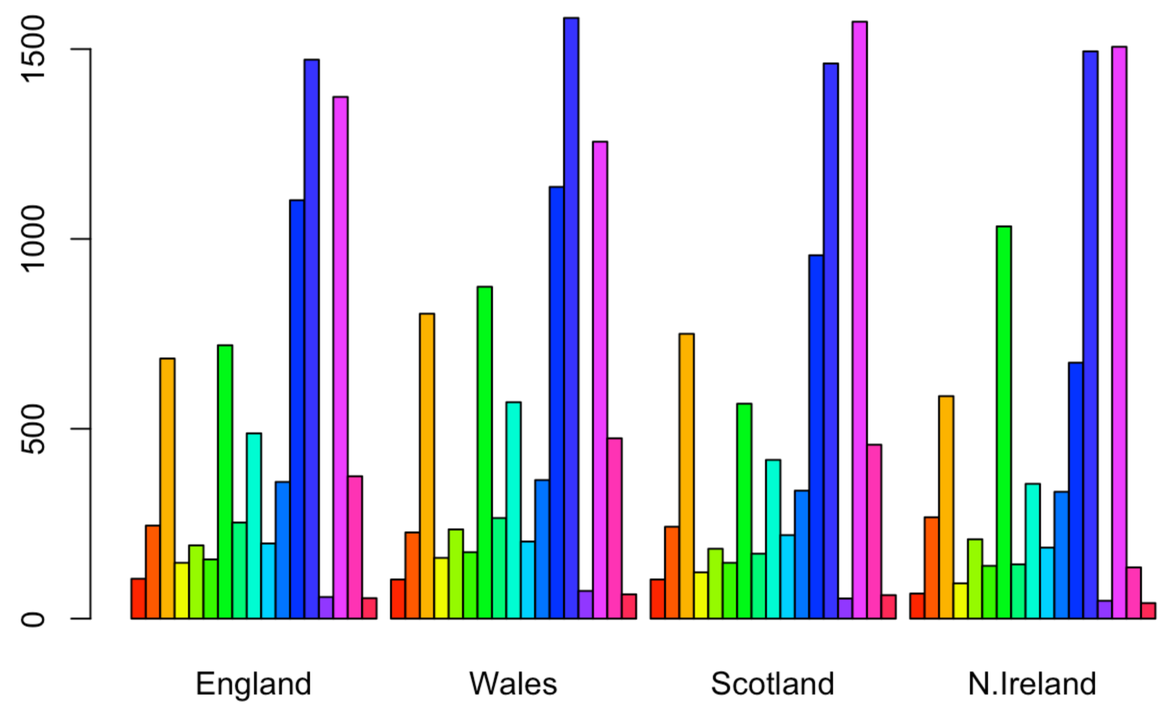
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars             156   175      147       139
```

Q2: Which approach to solving the ‘row-names problem’ mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

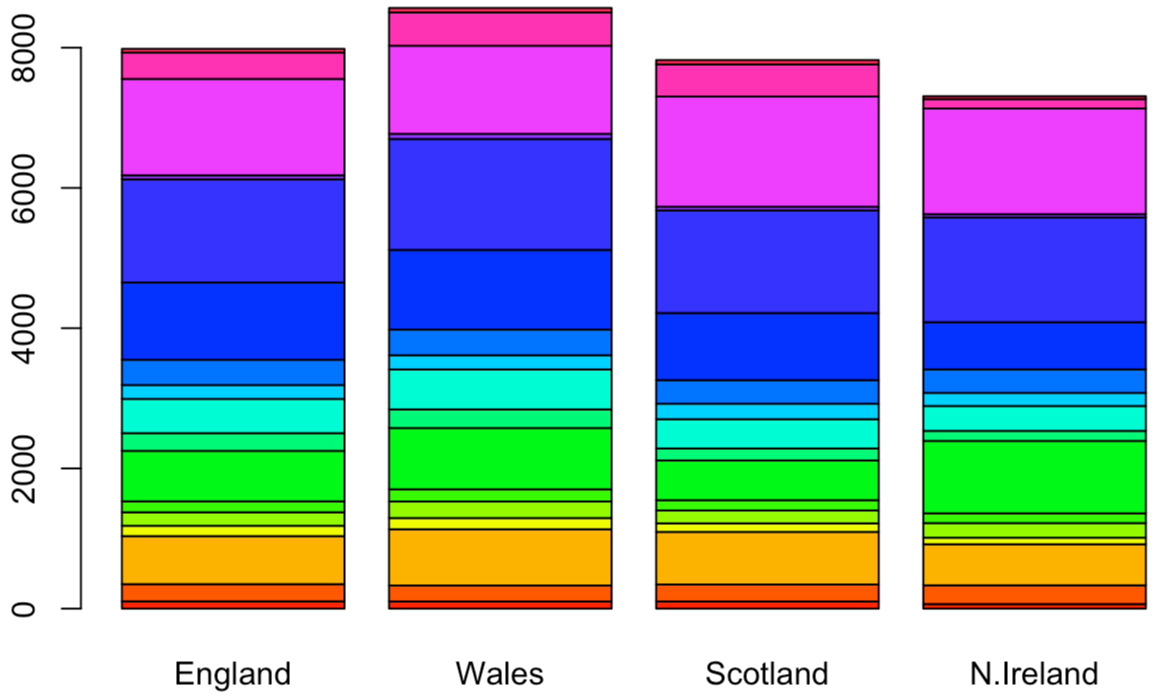
```
# x <- read.csv(url, row.names=1)
# This is safe
```

Q3: Changing what optional argument in the above barplot() function results in the following plot?

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

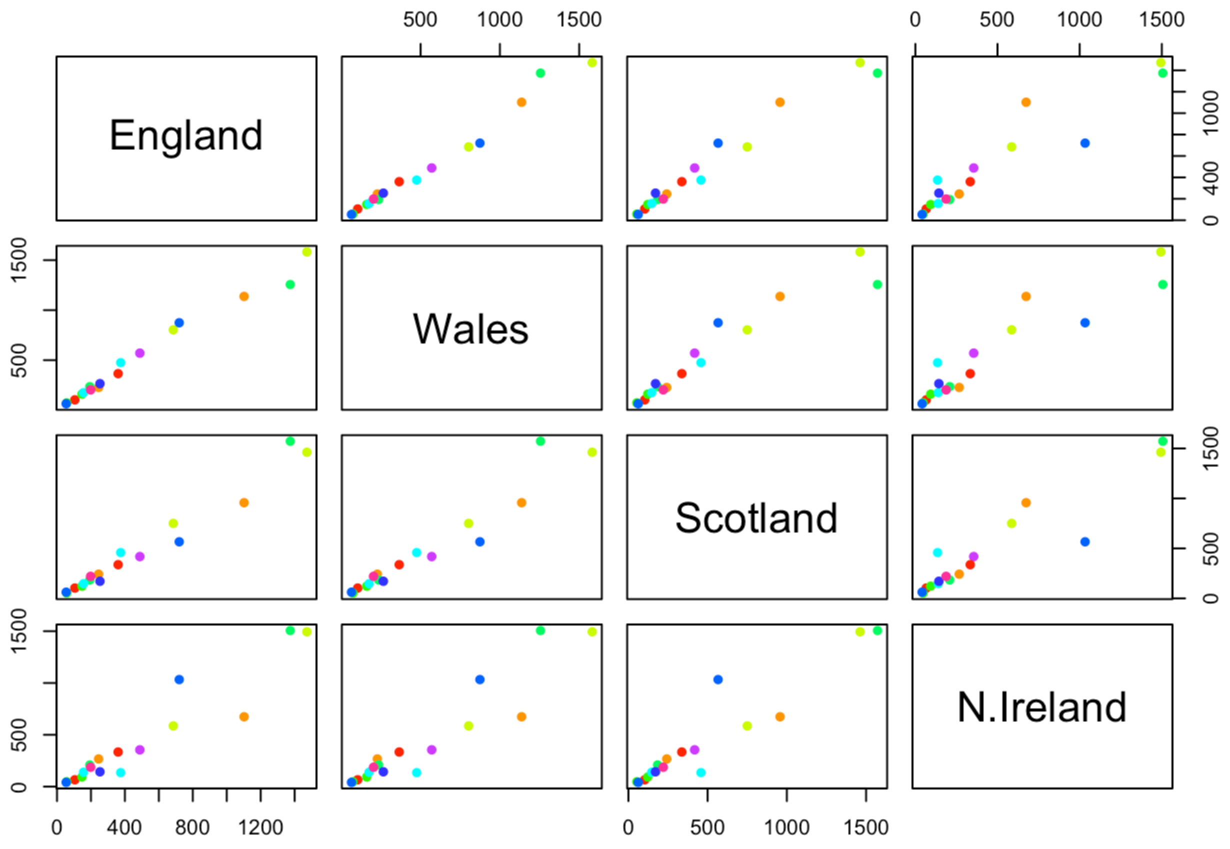


```r
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x))) #answer is change to F
```



Q5: Generating all pairwise plots may help somewhat. Can you make sense of the following code and resulting figure? What does it mean if a given point lies on the diagonal for a given plot?

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



```
# It always compares two countries. Column 2, row 1, compares england with wales.
# If it is not on the line, it is different between the countries. If the point i
```

Q6. What is the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

PCA to the rescue

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

```
# Use the prcomp() PCA function
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
```

Let's see what is inside our result object 'pca' that we just calculated:

```r
attributes(pca)
```

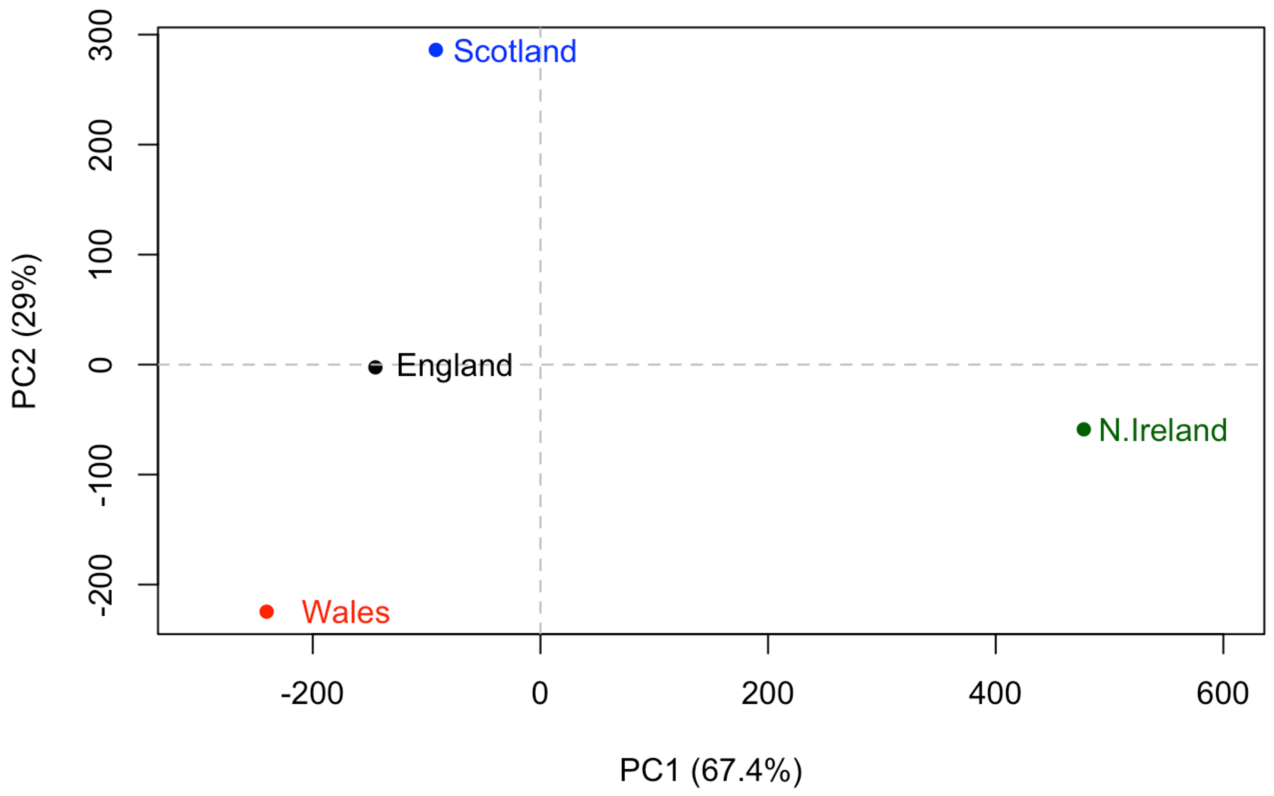
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"       

$class
[1] "prcomp"
```

Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```r
pca$x #if you want to data
```

```
                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
```

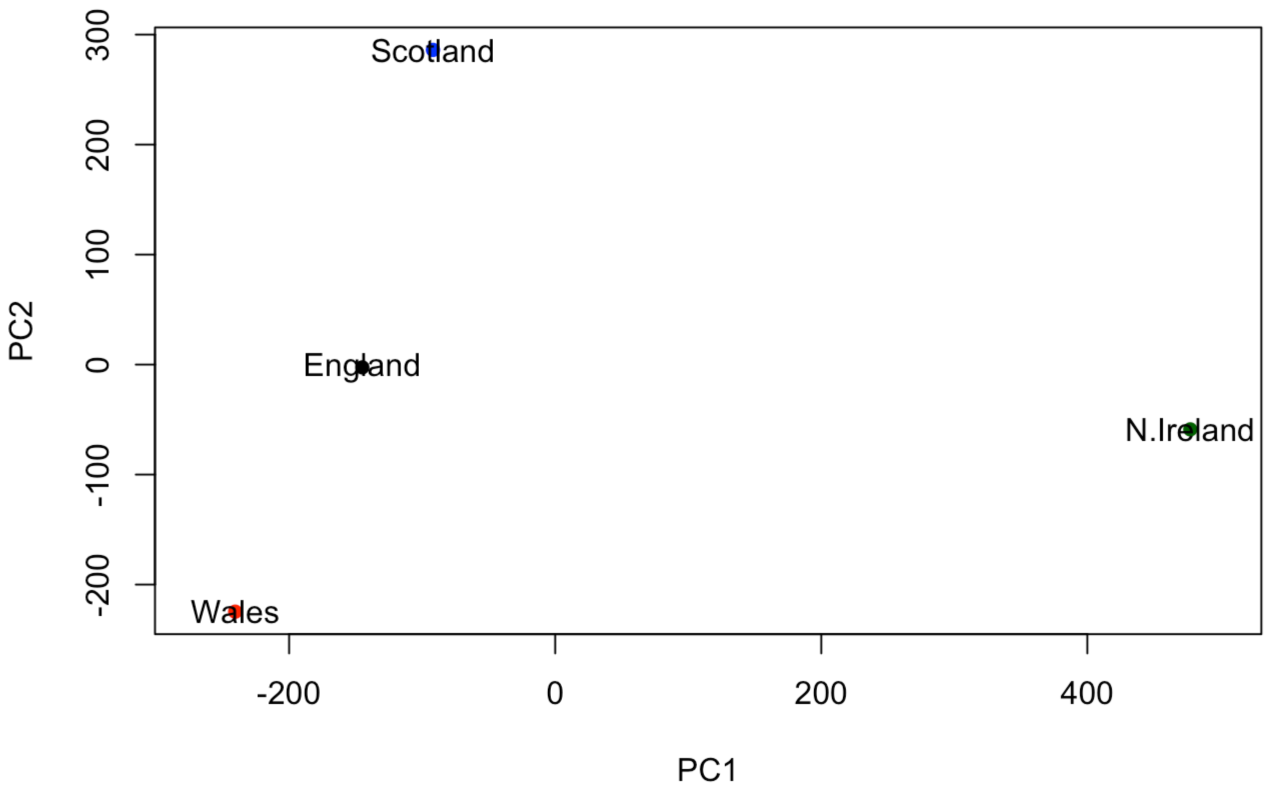
```r
# To make our main result figure, called a "PC plot" (or "score plot", or "ordina

# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2],
     col=c("black", "red", "blue", "darkgreen"),
     pch=16,
     xlab="PC1 (67.4%)", ylab="PC2 (29%)",
      xlim=c(-300,600)) +
abline(h=0, col="gray", lty=2) +
abline(v=0, col="gray", lty=2) +
text(pca$x[,1]+70, pca$x[,2], colnames(x),
     col=c("black", "red", "blue", "darkgreen"))
```



```
integer(0)
```

Q8. Customize your plot so that the colors of the country names match the colors in our UK and Ireland map and table at start of this document.

```
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500), col=c("black
text(pca$x[,1], pca$x[,2], colnames(x))
```

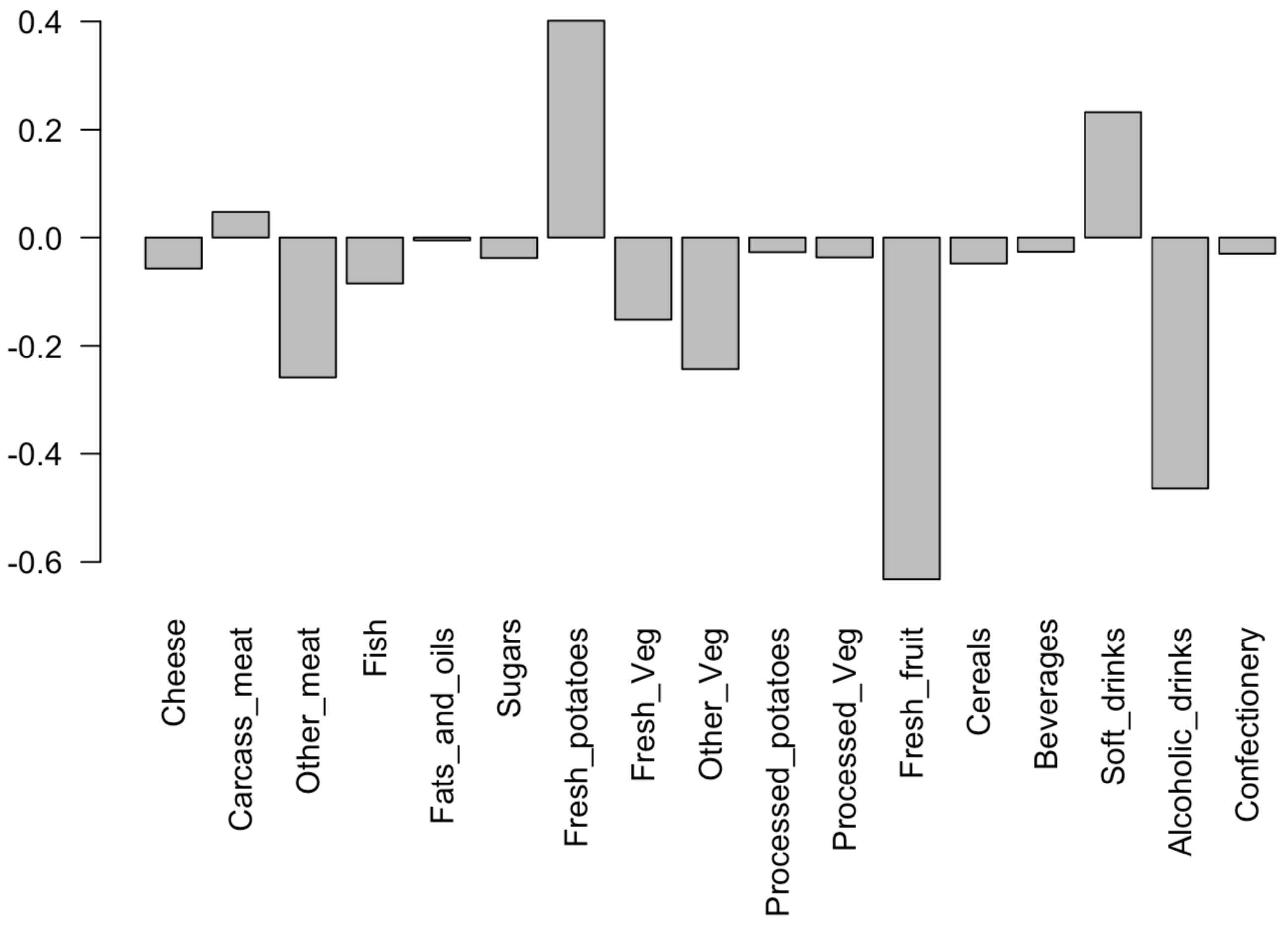


Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

```
pca$rotation
```

```
                            PC1          PC2         PC3          PC4
Cheese              -0.056955380  0.016012850  0.02394295 -0.409382587
Carcass_meat         0.047927628  0.013915823  0.06367111  0.729481922
Other_meat          -0.258916658 -0.015331138 -0.55384854  0.331001134
Fish                -0.084414983 -0.050754947  0.03906481  0.022375878
Fats_and_oils       -0.005193623 -0.095388656 -0.12522257  0.034512161
Sugars              -0.037620983 -0.043021699 -0.03605745  0.024943337
Fresh_potatoes       0.401402060 -0.715017078 -0.20668248  0.021396007
Fresh_Veg           -0.151849942 -0.144900268  0.21382237  0.001606882
Other_Veg           -0.243593729 -0.225450923 -0.05332841  0.031153231
Processed_potatoes  -0.026886233  0.042850761 -0.07364902 -0.017379680
Processed_Veg       -0.036488269 -0.045451802  0.05289191  0.021250980
Fresh_fruit         -0.632640898 -0.177740743  0.40012865  0.227657348
Cereals             -0.047702858 -0.212599678 -0.35884921  0.100043319
Beverages           -0.026187756 -0.030560542 -0.04135860 -0.018382072
Soft_drinks          0.232244140  0.555124311 -0.16942648  0.222319484
Alcoholic_drinks    -0.463968168  0.113536523 -0.49858320 -0.273126013
Confectionery       -0.029650201  0.005949921 -0.05232164  0.001890737
```

```
## 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 )
```



PCA of RNA-seq data

Q10: How many genes and samples are in this data set?