class07

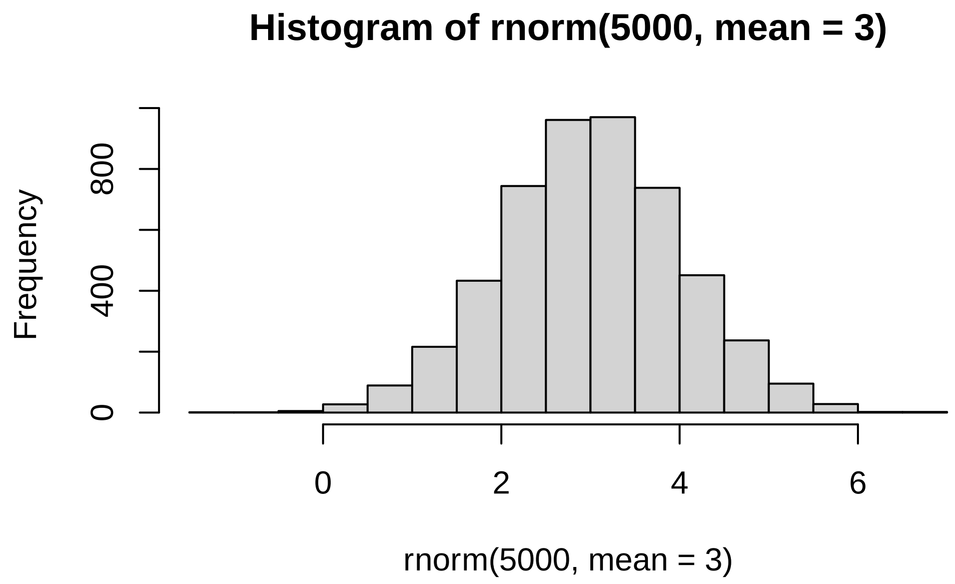
Sharlene Yang

Clustering

First let's make up some data to cluster so we can get a feel for these methods and how to work with them.

We can use the `rnorm()` function to get random numbers from a normal distribution around a given `mean`.

```r
hist(rnorm(5000, mean = 3))
```



Let's get 30 points with a mean of 3.

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

```
 [1]  1.854139  2.662916  3.763941  3.276596  1.383767  3.378643  4.011393
 [8]  2.270529  3.187053  3.166477  3.005245  2.688300  4.234360  4.141733
[15]  2.073393  3.349752  5.098710  2.722750  3.012937  3.495058  4.243105
[22]  4.414468  3.512653  4.469846  2.233505  1.943427  3.134889  3.198945
[29]  3.738780  2.869347 -2.456904 -4.087236 -4.520312 -3.938465 -2.806694
[36] -1.800357 -3.836458 -1.917171 -3.656225 -4.677399 -4.149475 -4.196859
[43] -1.691379 -1.851419 -2.628294 -2.269254 -4.471422 -3.615363 -2.097901
[50] -5.610092 -2.528510 -3.066923 -2.953418 -4.621749 -2.890803 -1.546576
[57] -3.735499 -2.620206 -2.287271 -4.369600
```

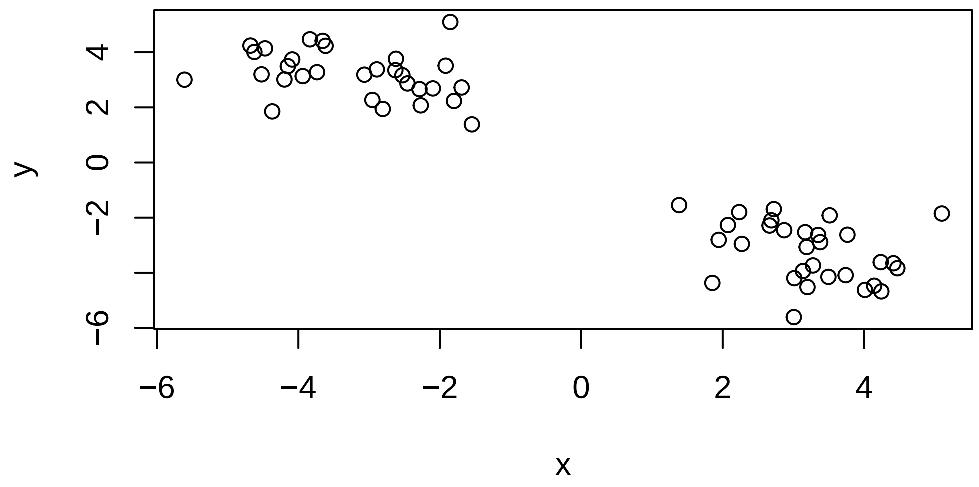
Put two of these together:

```r
x <- cbind(x=tmp, y= rev(tmp))
x
```

```
             x         y
 [1,]  1.854139 -4.369600
 [2,]  2.662916 -2.287271
 [3,]  3.763941 -2.620206
 [4,]  3.276596 -3.735499
 [5,]  1.383767 -1.546576
 [6,]  3.378643 -2.890803
 [7,]  4.011393 -4.621749
 [8,]  2.270529 -2.953418
 [9,]  3.187053 -3.066923
[10,]  3.166477 -2.528510
[11,]  3.005245 -5.610092
[12,]  2.688300 -2.097901
[13,]  4.234360 -3.615363
[14,]  4.141733 -4.471422
[15,]  2.073393 -2.269254
[16,]  3.349752 -2.628294
[17,]  5.098710 -1.851419
[18,]  2.722750 -1.691379
[19,]  3.012937 -4.196859
[20,]  3.495058 -4.149475
[21,]  4.243105 -4.677399
[22,]  4.414468 -3.656225
```

```
[23,]  3.512653 -1.917171
[24,]  4.469846 -3.836458
[25,]  2.233505 -1.800357
[26,]  1.943427 -2.806694
[27,]  3.134889 -3.938465
[28,]  3.198945 -4.520312
[29,]  3.738780 -4.087236
[30,]  2.869347 -2.456904
[31,] -2.456904  2.869347
[32,] -4.087236  3.738780
[33,] -4.520312  3.198945
[34,] -3.938465  3.134889
[35,] -2.806694  1.943427
[36,] -1.800357  2.233505
[37,] -3.836458  4.469846
[38,] -1.917171  3.512653
[39,] -3.656225  4.414468
[40,] -4.677399  4.243105
[41,] -4.149475  3.495058
[42,] -4.196859  3.012937
[43,] -1.691379  2.722750
[44,] -1.851419  5.098710
[45,] -2.628294  3.349752
[46,] -2.269254  2.073393
[47,] -4.471422  4.141733
[48,] -3.615363  4.234360
[49,] -2.097901  2.688300
[50,] -5.610092  3.005245
[51,] -2.528510  3.166477
[52,] -3.066923  3.187053
[53,] -2.953418  2.270529
[54,] -4.621749  4.011393
[55,] -2.890803  3.378643
[56,] -1.546576  1.383767
[57,] -3.735499  3.276596
[58,] -2.620206  3.763941
[59,] -2.287271  2.662916
[60,] -4.369600  1.854139
```

```r
plot(x)
```



K-means clustering.

Very popular clustering method that we cab use with the `kmeans()` function in base R.

```r
km <- kmeans(x, centers = 2)
km
```

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

Cluster means:
          x         y
1 -3.229975  3.217889
2  3.217889 -3.229975

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] 56.31718 56.31718
 (between_SS / total_SS =  91.7 %)
```

```
Available components:

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

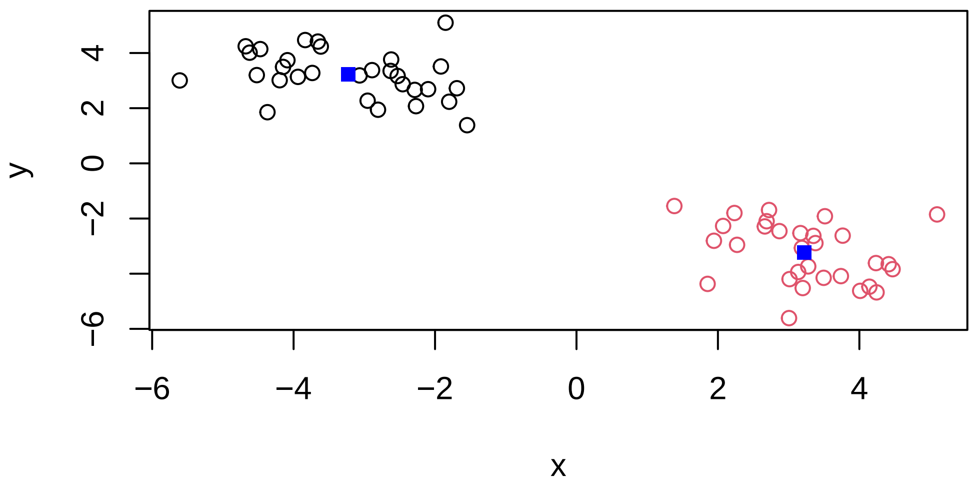
```
km$cluster
```

```
 [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 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
```

```
km$center
```

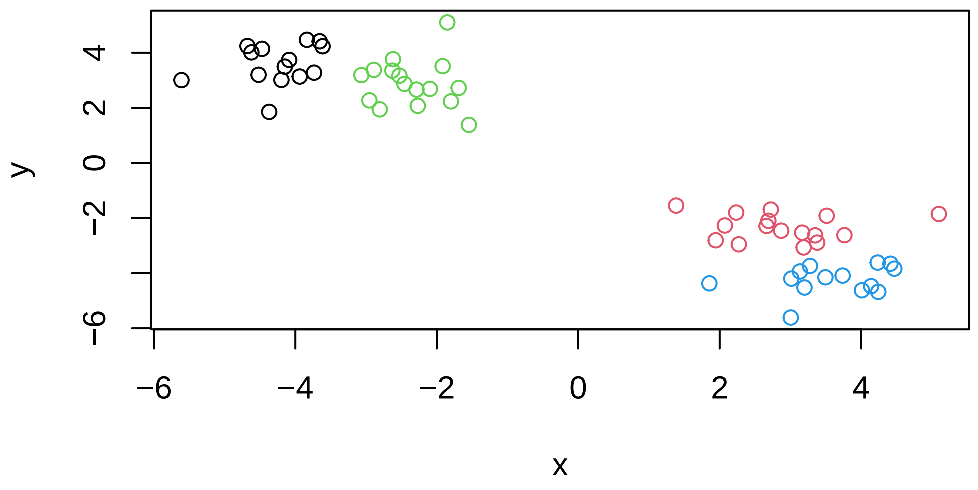
```
          x         y
1 -3.229975  3.217889
2  3.217889 -3.229975
```

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



Q. Let’s cluster into 3 groups or some `x` data and make a plot

```r
km <- kmeans(x, centers = 4)
plot(x, col=km$cluster)
```



Hierarchial Clustering

We can use the `hclust()` for Hierarchical Clustering Unlike `kmeans()`, where we could just pass in our data as input, we need to give give `hclust()` a "distance matrix".

We will use the `dist()` function to start with.

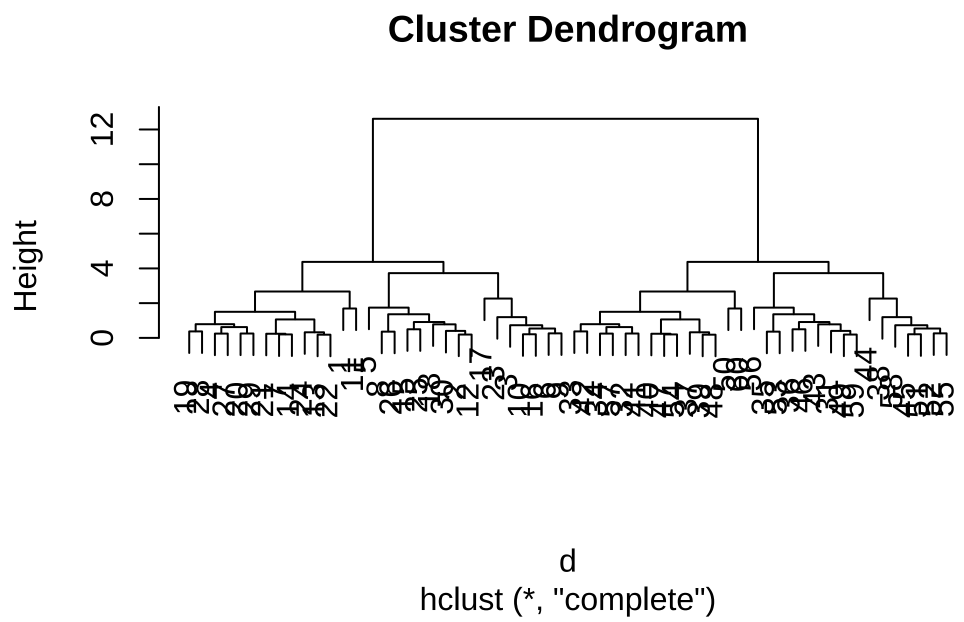
```r
d <- dist(x)
hc <- hclust(d)
hc
```

```

Call:
hclust(d = d)

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

```r
plot(hc)
```



I can now "cut" my tree with the `cutree()` to yield a cluster membership vector.

```r
grps <- cutree(hc, h= 8)
grps
```

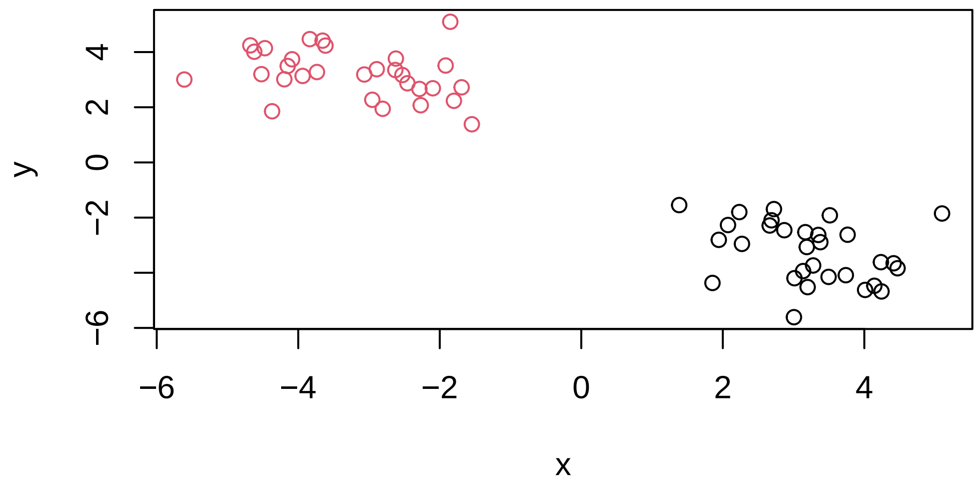
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

You can also tell `cutree()` to cut where it yields "k" groups.

```r
cutree(hc, k=2)
```

```
 [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
```

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



Principal Component Analysis (PCA)

Class 7 lab

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

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

```r
nrow(x)
```

```
[1] 17
```

```r
ncol(x)
```

```
[1] 5
```

There are 17 rows and 5 columns.

```r
View(x)
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
```

```r
tail(x)
```

```
                   X England Wales Scotland N.Ireland
12      Fresh_fruit     1102  1137      957       674
13          Cereals     1472  1582     1462      1494
14         Beverages      57    73       53        47
15      Soft_drinks     1374  1256     1572      1506
16 Alcoholic_drinks      375   475      458       135
17    Confectionery       54    64       62        41
```

To fix the name of the row, add `row.name = 1` to the `read.csv`

```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
```

Checking the dimensions again

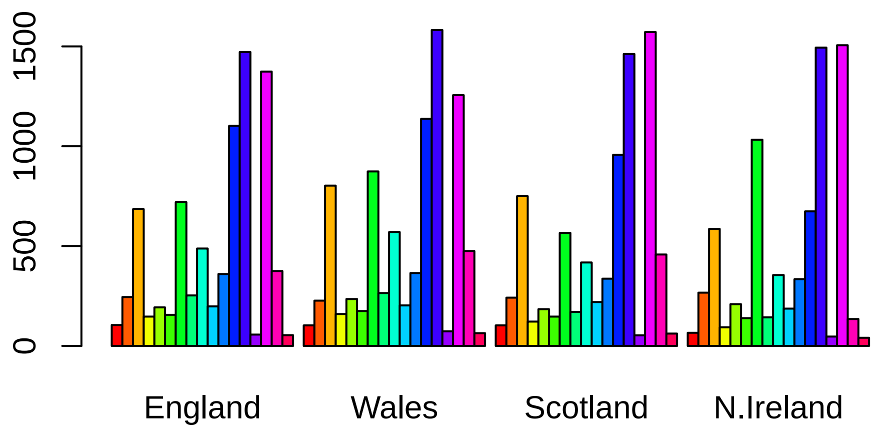
```
dim(x)
```

```
[1] 17  4
```

> 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?

I prefer the adding `row.names` because if you use the `-1` method and continuously run it, it will remove part of the table, which isn't what we want.

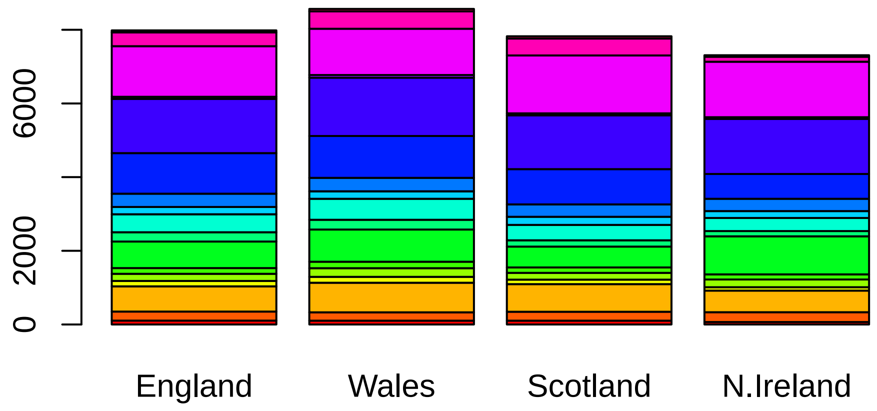
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

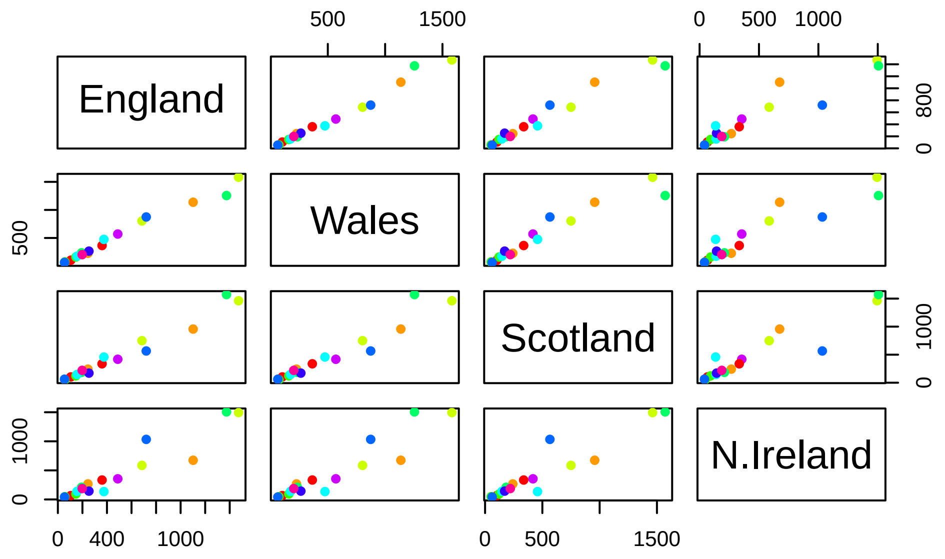
Changing the "besides" argument will change it to stacked bars.

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



> 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)
```



For the code, it graphs the countries on the x-axis or the y-axis. It compares the amount of products consumed/ used in each country. For example, row 1 column 2, it shows England on the y-axis and Wales on the x-axis. If a given point lies on the diagonal for a given plot, it means that both countries have used about the same amount of the product.

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

Main difference is that is has the distinct blue dot near the 1000 mark.

The main PCA function in base R is called `prcomp()` it expects the transpose of our 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
```

```
attributes(pca)
```

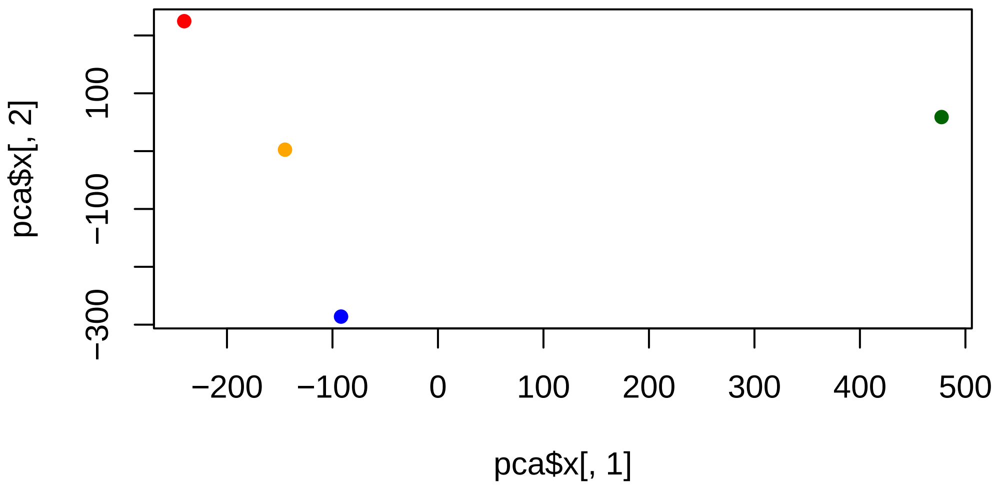
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```
pca$x
```

```
                 PC1         PC2         PC3           PC4
England   -144.99315    2.532999 -105.768945  2.842865e-14
Wales     -240.52915  224.646925   56.475555  7.804382e-13
Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

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


pca$x[, 2]
pca$x[, 1]