Class 7: Machine Learning I

Arman Farahani A17497672

Today we are going to learn how to apply different machine learning methods, begining with clustering:

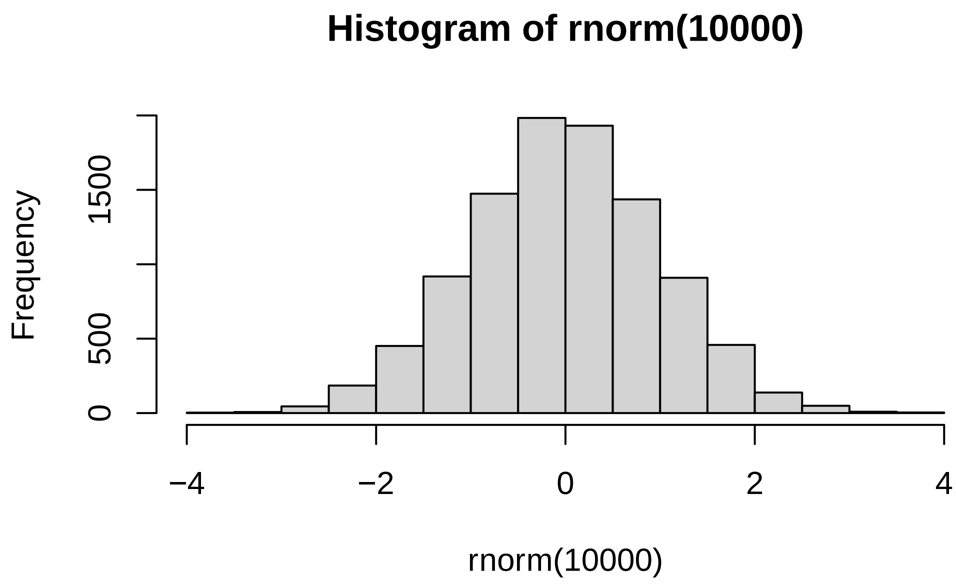
The goal hee is to find groups/clusters in your input data.

First II will make up some data with clear groups. For tthis I will use the `rnorm()` function.

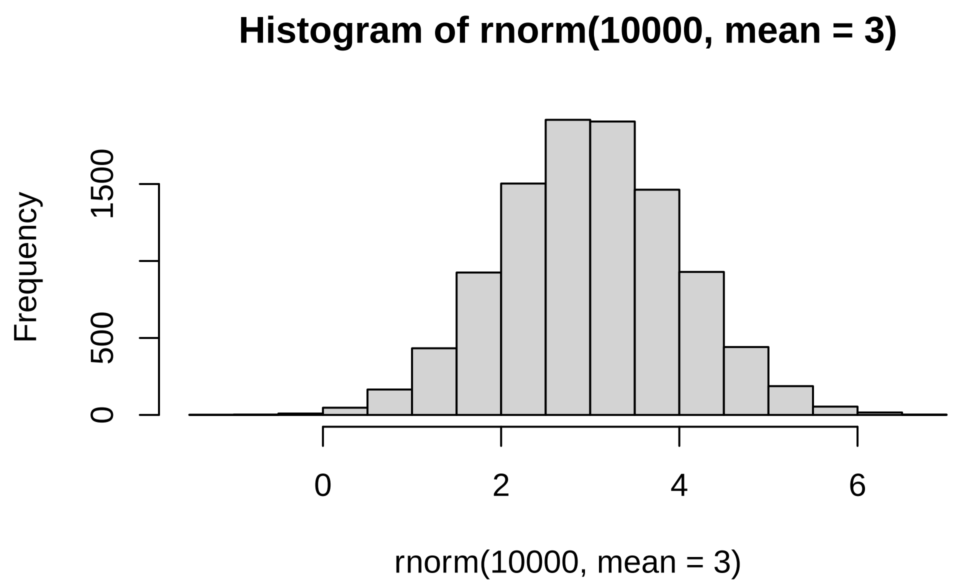
```
rnorm(10)
```

```
 [1]  0.02034549 -0.23323336 -0.05477835  0.60860457 -0.61076463  1.17173344
 [7] -1.23321090 -0.47579293 -1.66379373  1.34730321
```

```
hist( rnorm(10000) )
```



```r
hist( rnorm(10000, mean=3)  )
```



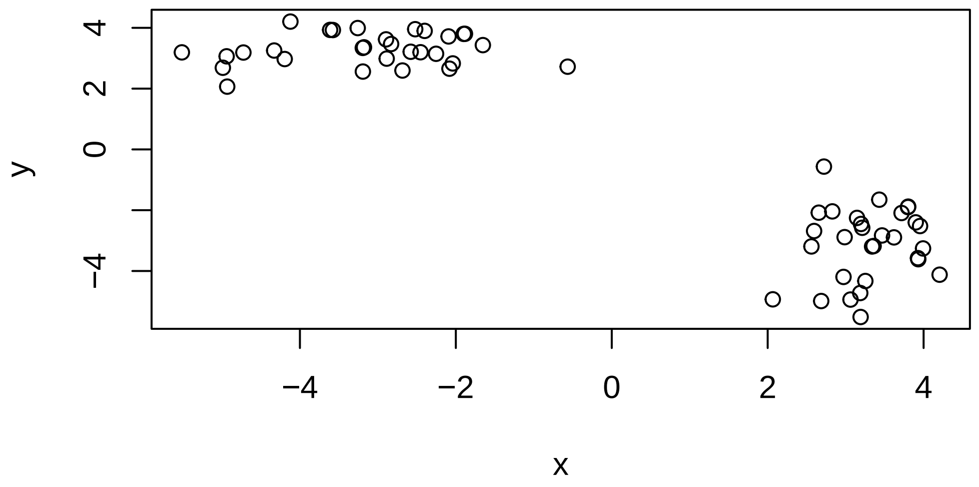
```r
n <- 30
x <- c(rnorm(n, -3), rnorm(n, +3))
y <- rev(x)
y
```

```
 [1]  3.3588777  3.6200603  3.4674440  2.6866279  3.1461406  3.9929117
 [7]  3.8993859  3.8000065  2.9869769  2.0658241  2.5613461  3.1968006
[13]  3.0624894  4.2054771  3.7173125  3.9316315  2.5955059  3.1875076
[19]  2.6555183  3.8025302  3.1922416  3.2527911  3.2128774  3.9546613
[25]  2.7213989  2.8293756  3.4316537  3.9275001  3.3391575  2.9726882
[31] -4.1949643 -3.1932639 -3.5767757 -1.6532270 -2.0391735 -0.5655745
[37] -2.5211202 -2.5788479 -4.3300632 -5.5138948 -1.8826669 -2.0816715
[43] -4.7246441 -2.6844769 -3.6125355 -2.0931389 -4.1216388 -4.9398170
[49] -2.4527720 -3.1920844 -4.9323473 -2.8869290 -1.8985437 -2.3999277
[55] -3.2586782 -2.2533460 -4.9883190 -2.8301746 -2.8950352 -3.1778834
```

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

```
             x        y
[1,] -3.177883 3.358878
[2,] -2.895035 3.620060
[3,] -2.830175 3.467444
[4,] -4.988319 2.686628
[5,] -2.253346 3.146141
[6,] -3.258678 3.992912
```

```r
plot(z)
```



Q. How many points are in each cluster?

Q. What 'component' of your result objet details -cluster size -cluster assignment/membership -cluster center

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

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

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

```
Cluster means:
          x         y
1 -3.115785  3.292491
2  3.292491 -3.115785

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] 48.19589 48.19589
 (between_SS / total_SS =  92.7 %)

Available components:

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

Results in kmeans object `km`

```
attributes(km)
```

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

$class
[1] "kmeans"
```

cluster size?

```
km$size
```

```
[1] 30 30
```

cluster assignment/membership?

```
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
```

cluster center

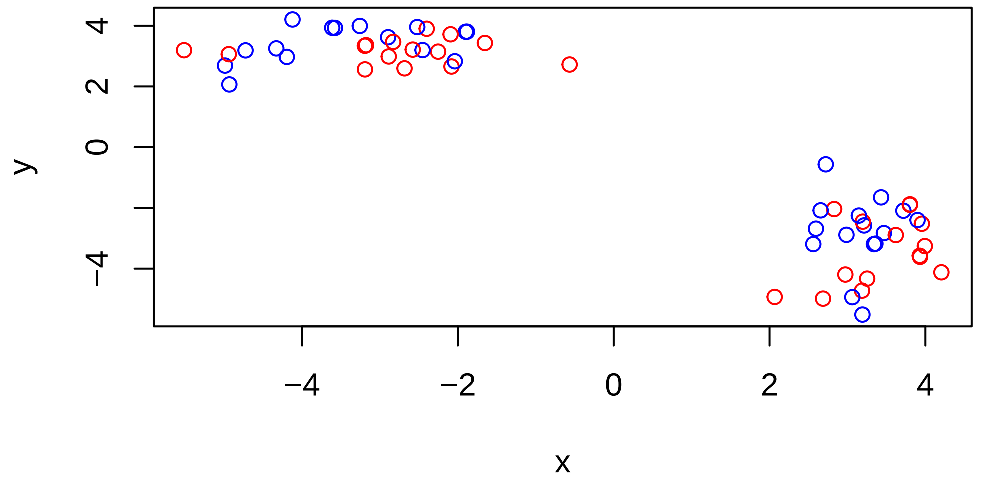
```
km$center
```

```
          x         y
1 -3.115785  3.292491
2  3.292491 -3.115785
```

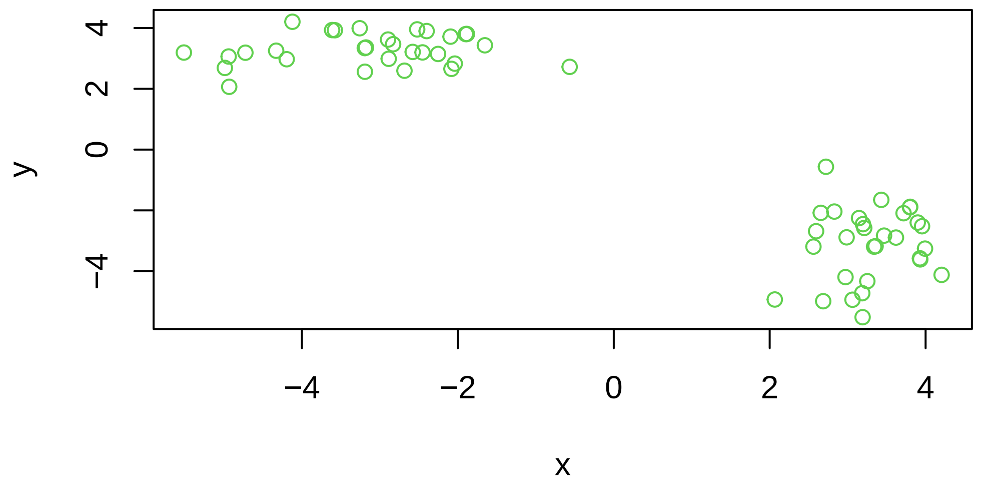
> Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

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

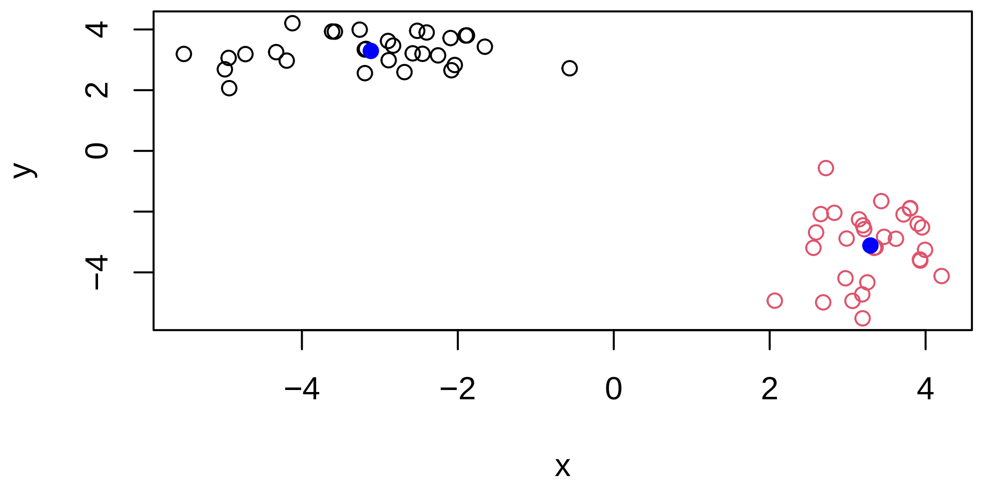
```
plot(z, col=c("red", "blue"))
```



```
plot(z, col=3)
```

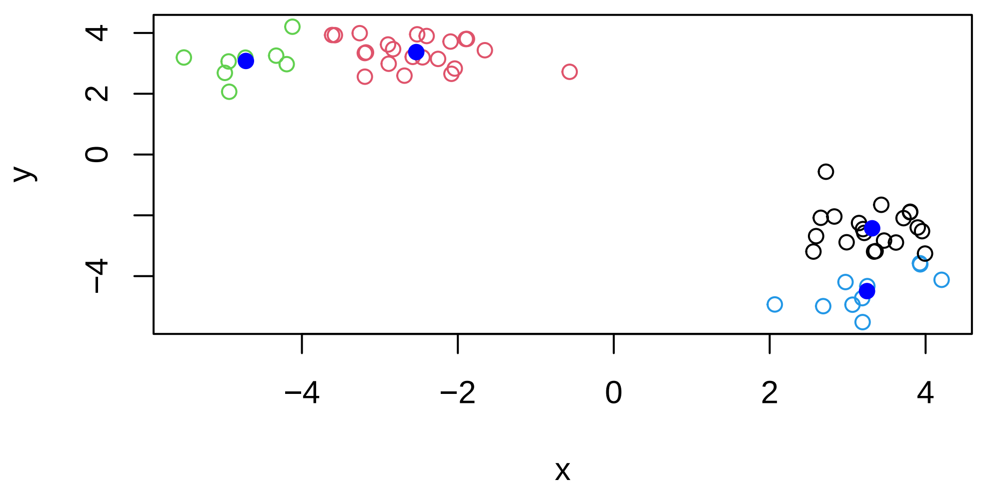


```
plot(z, col=km$cluster)
points(km$centers, col="blue", pch=19)
```



Q. Can you run kmeans and ask for 4 clusters please and plot the results like we have done above?

```
km1 <- kmeans(z, center=4)
plot(z, col=km1$cluster)
points(km1$centers, col="blue", pch=19)
```



Hierarchical clustering

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

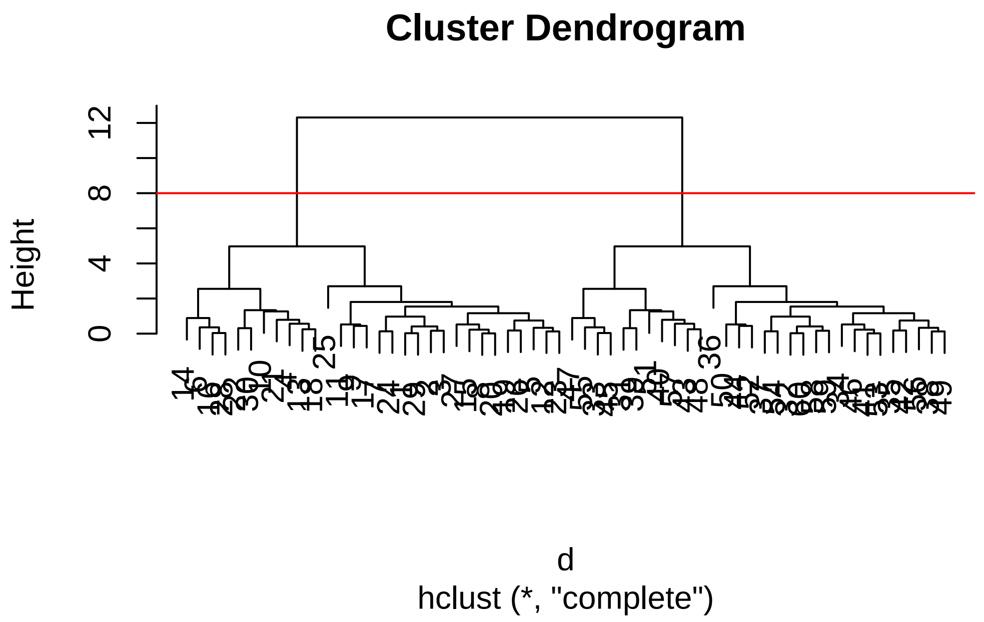
```
d <- dist(z)
hc <- hclust(d)
hc
```

```

Call:
hclust(d = d)

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

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



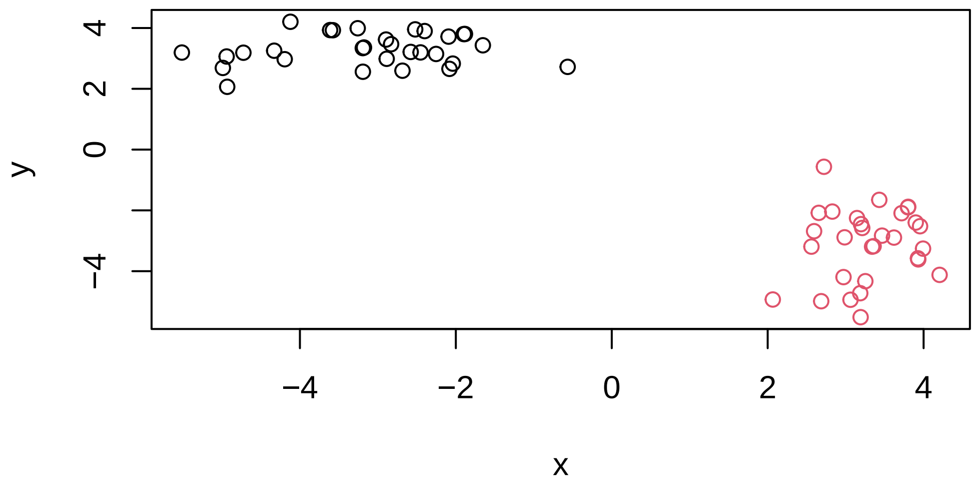
I can get my cluster membership vector by "cutting the tree" with the `cutree()` function like so:

```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 2 2
```

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

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



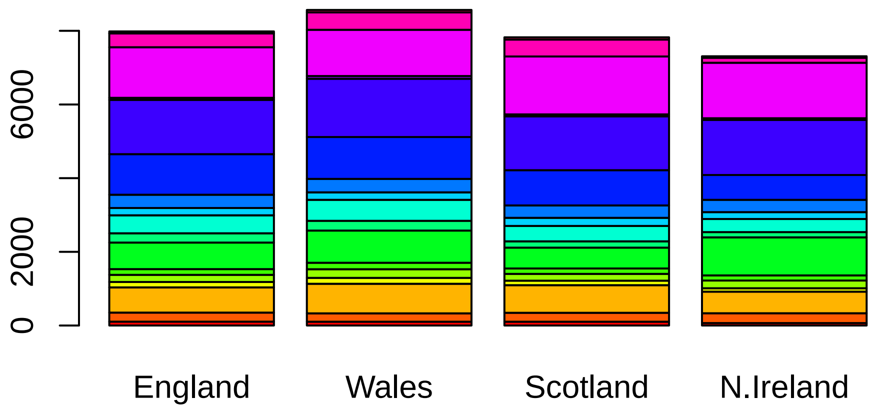
PCA of UK food data

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)
```

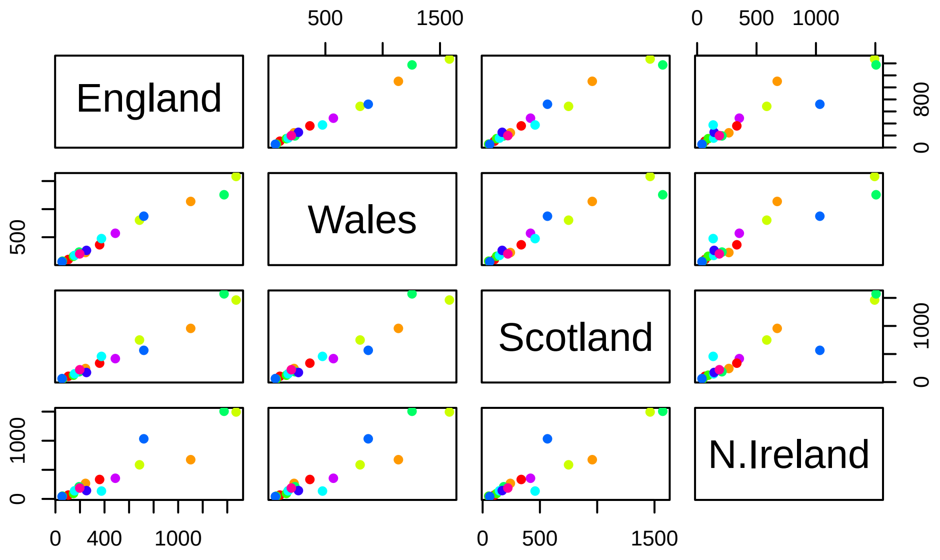
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars  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

```
barplot(as.matrix(x), beside=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 is hard to see structure and trends in even this small dataset. How will we ever do this when we have big datasets with 1,000s or 10s of thousands of things we are measuring…

PCA to the rescue

Let's see how PCA deals with this dataset. So main function in base R to do PCA is called `prcomp()`

```
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
```

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

```
attributes(pca)
```

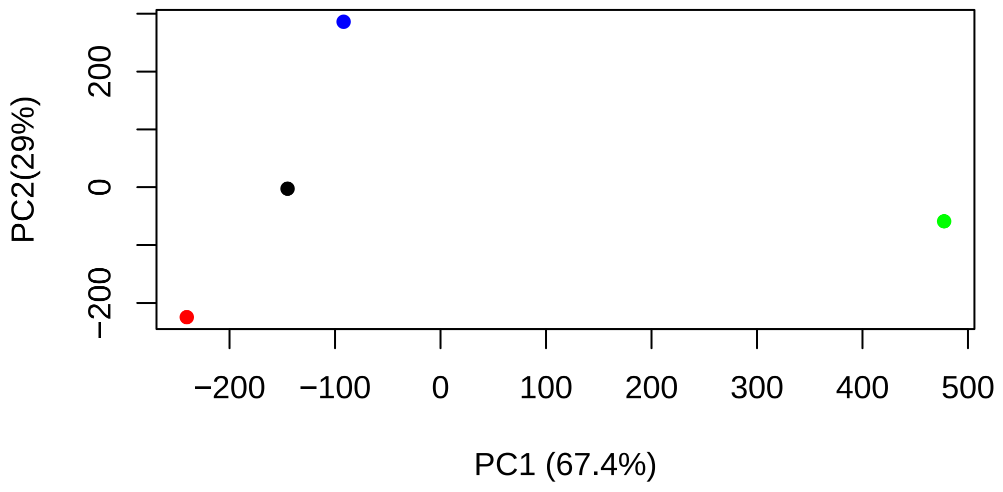
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"       

$class
[1] "prcomp"
```

```
pca$x
```

```
                 PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -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
```

```
plot(pca$x[,1], pca$x[,2], col=c("black", "red", "blue", "green"), pch=16, xlab="PC1 (67.4%)"
```



Quarto

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

Running Code

When you click the **Render** button a document will be generated that includes both content and the output of embedded code. You can embed code like this:

```
1 + 1
```

```
[1] 2
```

You can add options to executable code like this

```
[1] 4
```

The `echo: false` option disables the printing of code (only output is displayed).