

# Class 7 Machine Learning I

Sophia Wang A16838155

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

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

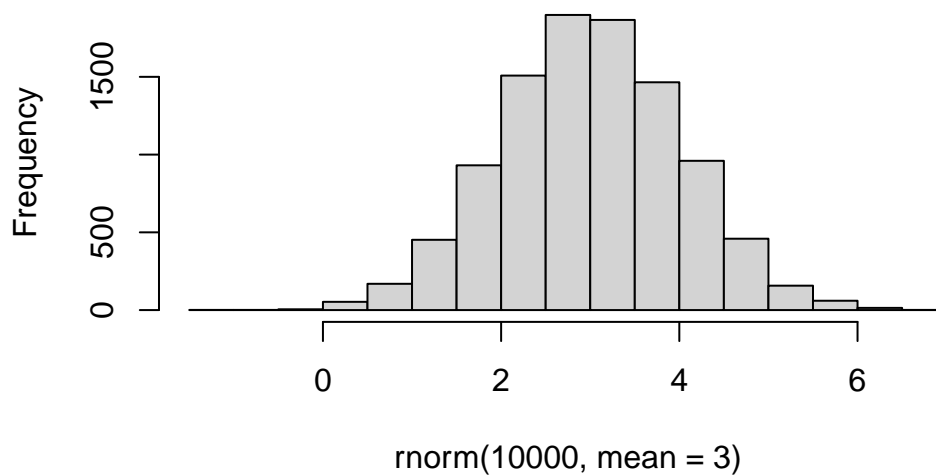
First I will make up some data with clear groups. For this I will use the `rnorm()` function:

```
rnorm(10)
```

```
[1] 0.57363444 0.68985836 -0.63738851 0.25769978 -0.01225789 0.43902090  
[7] 0.09254368 -0.06920554 -0.31056930 0.88075480
```

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

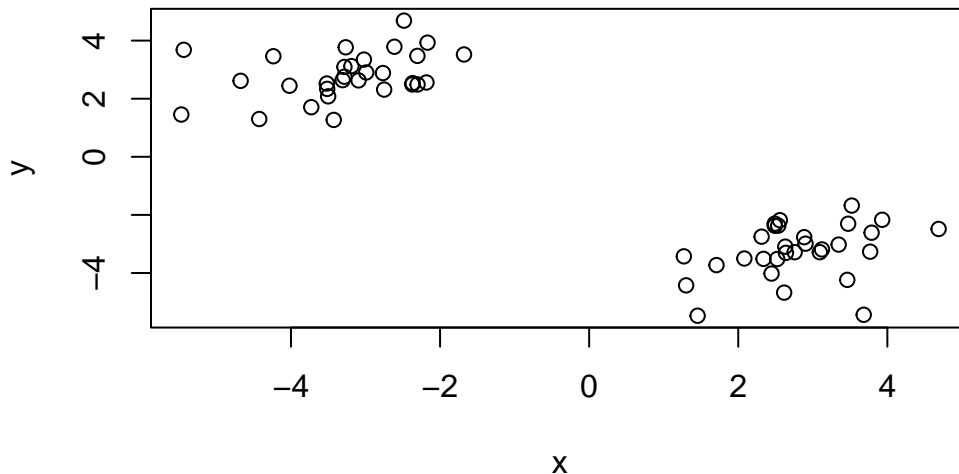
**Histogram of `rnorm(10000, mean = 3)`**



```
n=30
x <- c(rnorm(n,-3),rnorm(n,3))
y <- rev(x)
z <- cbind(x,y)
head(z)
```

```
      x      y
[1,] -2.749854 2.313384
[2,] -3.283442 2.757159
[3,] -2.766172 2.884232
[4,] -3.093804 2.631155
[5,] -2.304157 3.474425
[6,] -2.368678 2.536265
```

```
plot(z)
```



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/membership? - cluster center?

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

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

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

Cluster means:

	x	y
1	2.794209	-3.245936
2	-3.245936	2.794209

Clustering vector:

[illegible]

Within cluster sum of squares by cluster:

```
[1] 43.54446 43.54446
(between_SS / total_SS = 92.6 %)
```

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] 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
```

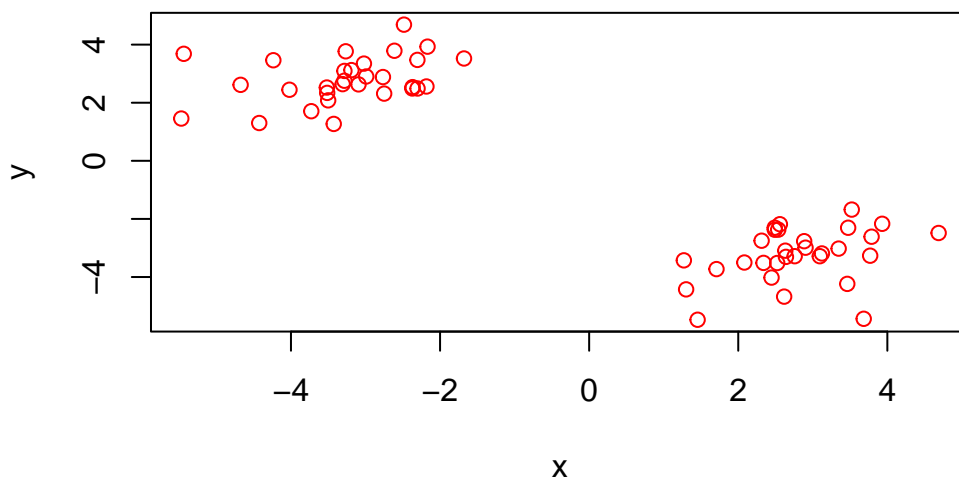
Cluster centers?

```
km$centers
```

```
      x      y  
1  2.794209 -3.245936  
2 -3.245936  2.794209
```

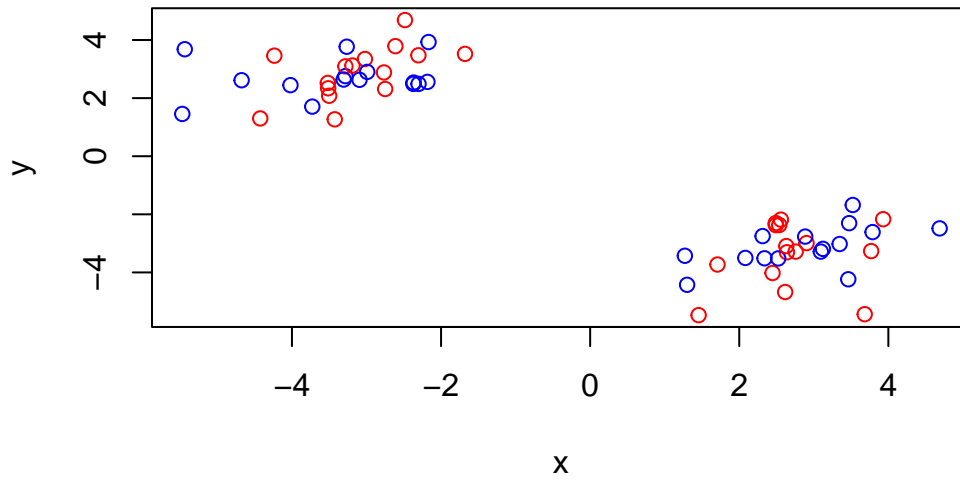
Q. Plot `z` colored by the kmeans cluster assignment and add cluster centers as blue points

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

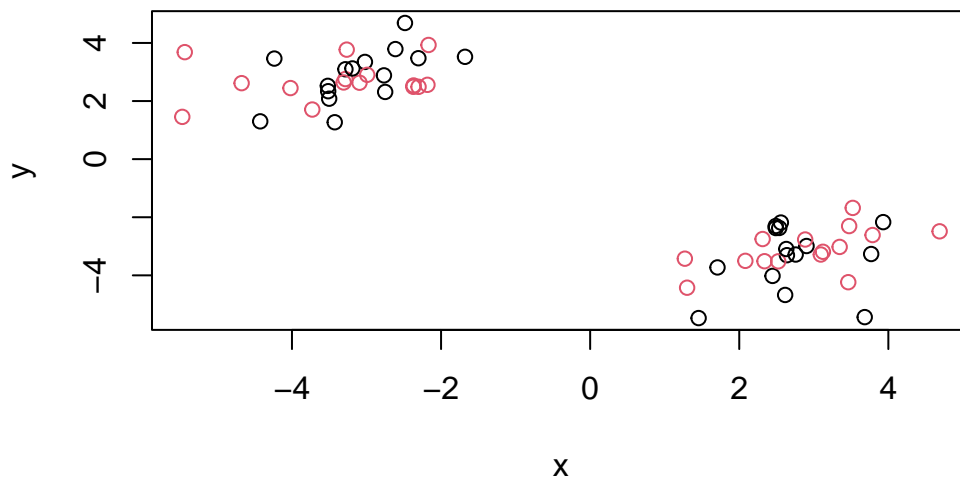


R will re-cycle the shorter color vector to be the same length as the longer (number of data points) in `z`

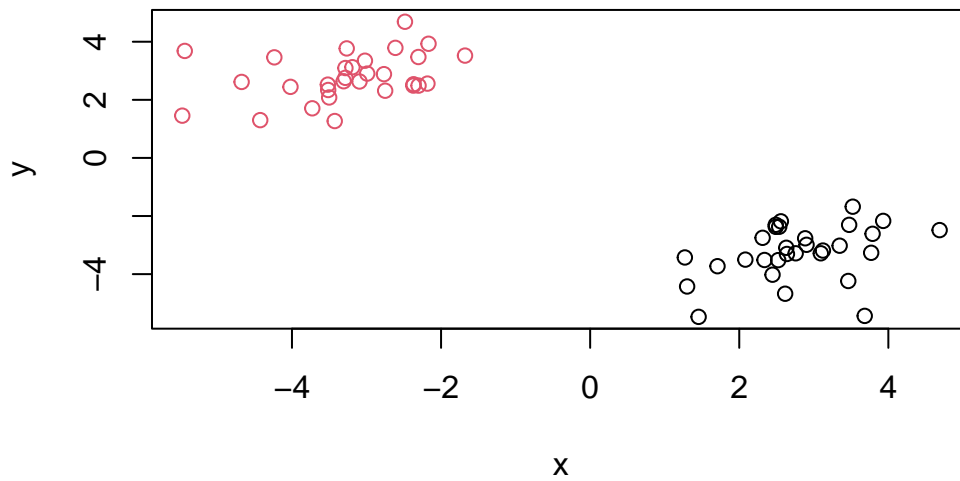
```
plot(z,col=c("red","blue"))
```



```
plot(z,col=c(1,2))
```

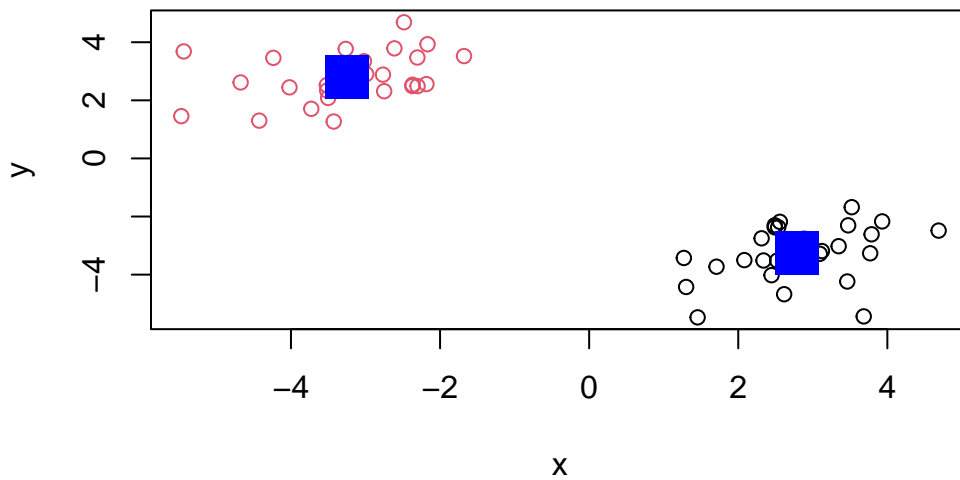


```
plot(z,col=km$cluster)
```



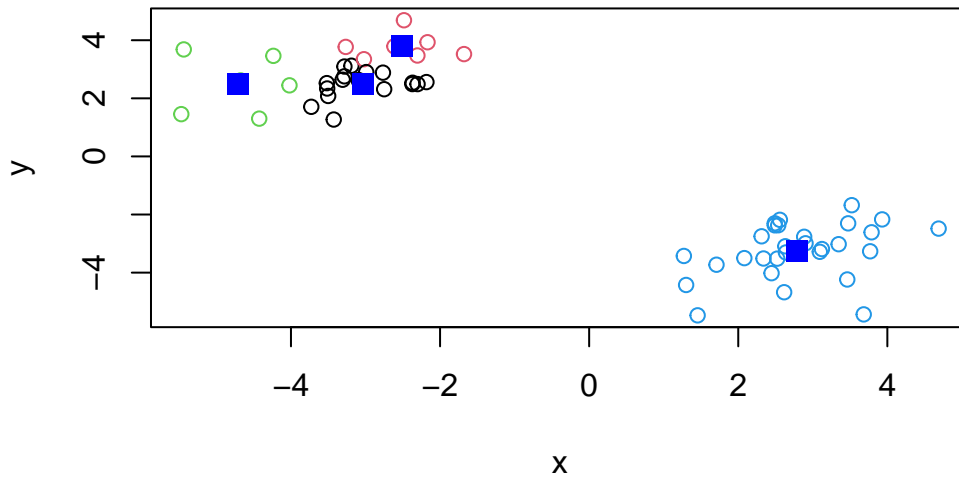
We can use the `points()` function to add new points to an existing plot...like the cluster centers.

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



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

```
km <- kmeans(z,4)
plot(z,col=km$cluster)
points(km$centers,col="blue",pch=15,cex=1.5)
```



##Hierarchical Clustering

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

First we need a distance matrix of our data to be clustered.

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

Call:

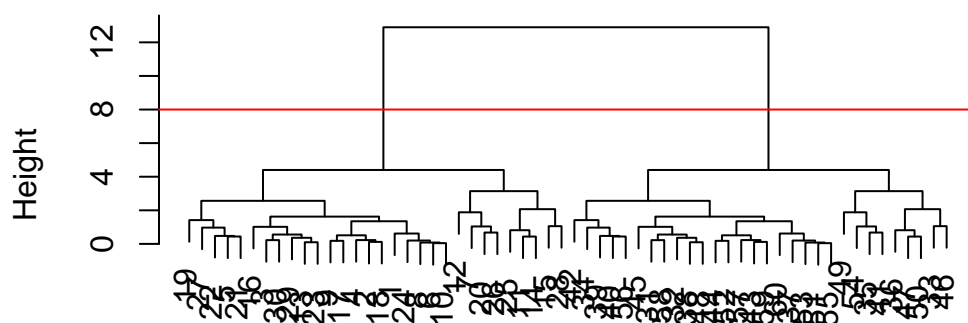
```
hclust(d = d)
```

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

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



## Cluster Dendrogram



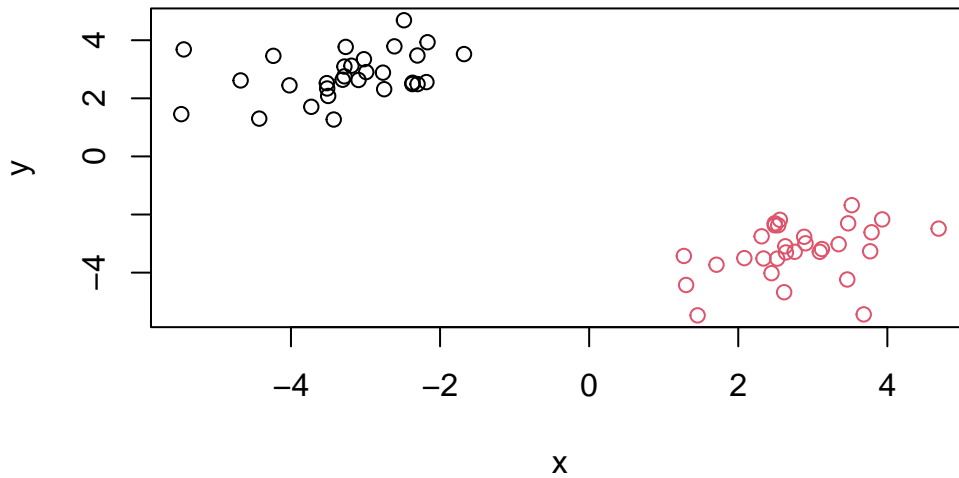
d  
hclust (\*, "complete")

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

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

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

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

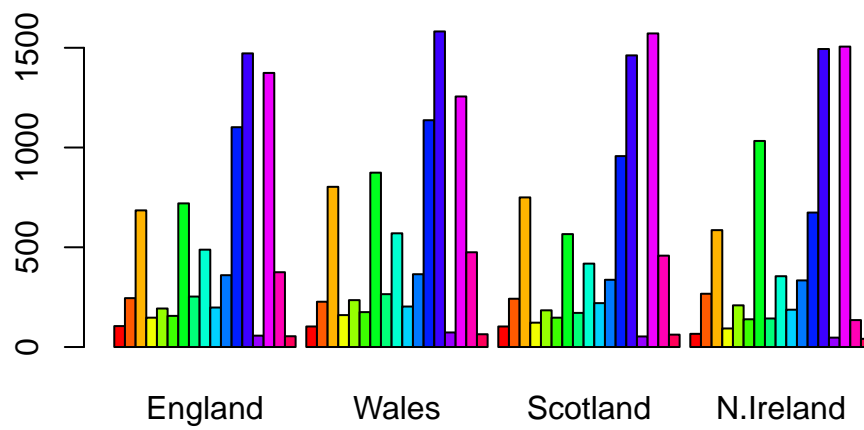


##PCA of UK food data Read data from the UK on food consumption in different parts of the UK.

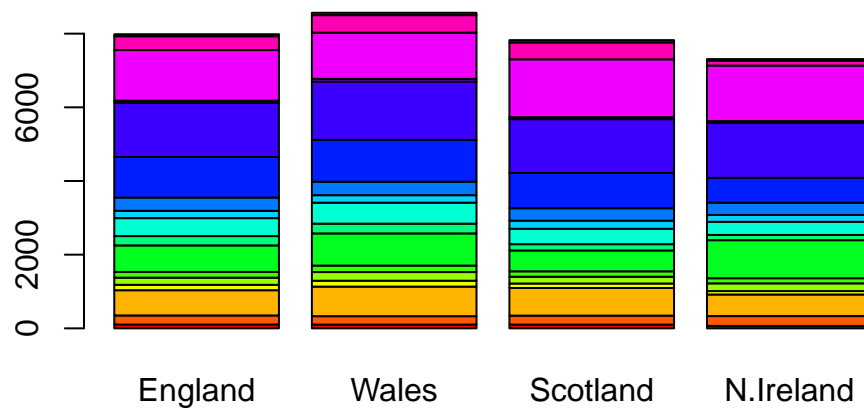
```
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url,,row.names=1)
head(x)
```

	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=T, col=rainbow(nrow(x)))
```

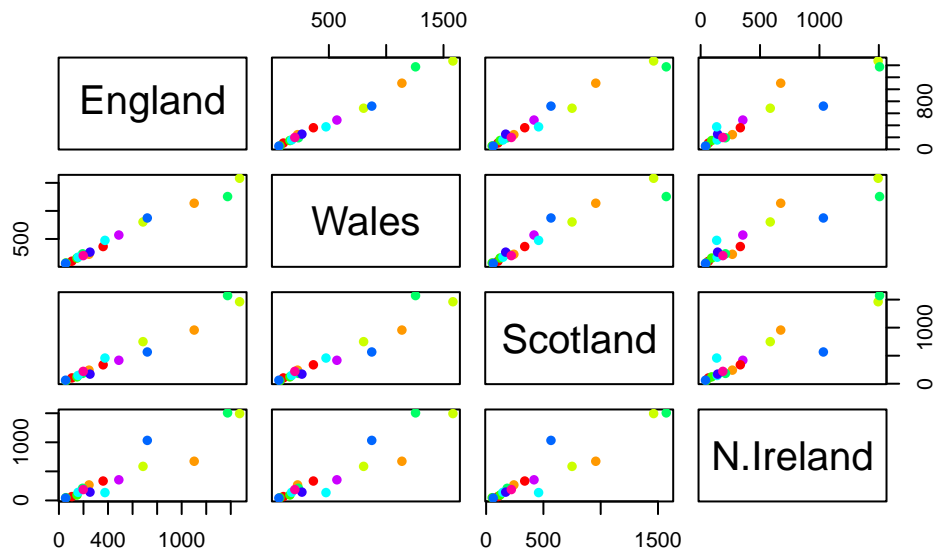


```
barplot(as.matrix(x), beside=FALSE, col=rainbow(nrow(x)))
```



A so-called **Paris** plot can be useful for small datasets like this

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



It's hard to see structure and trends in even this small dataset. How will we ever do this when we have big datasets with 1,000 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	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 this `pca` object that we created from running `prcomp()`

```
attributes(pca)
```

```
$names
```

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

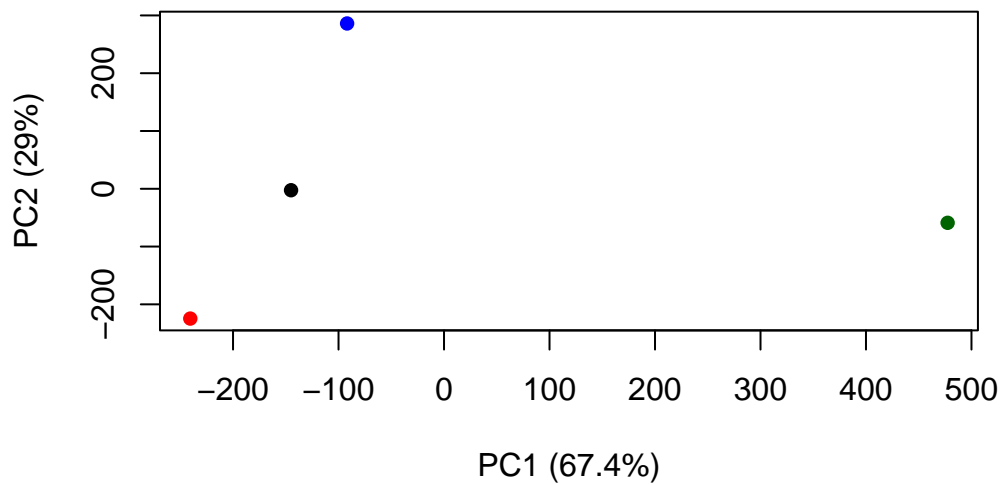
```
$class
```

```
[1] "prcomp"
```

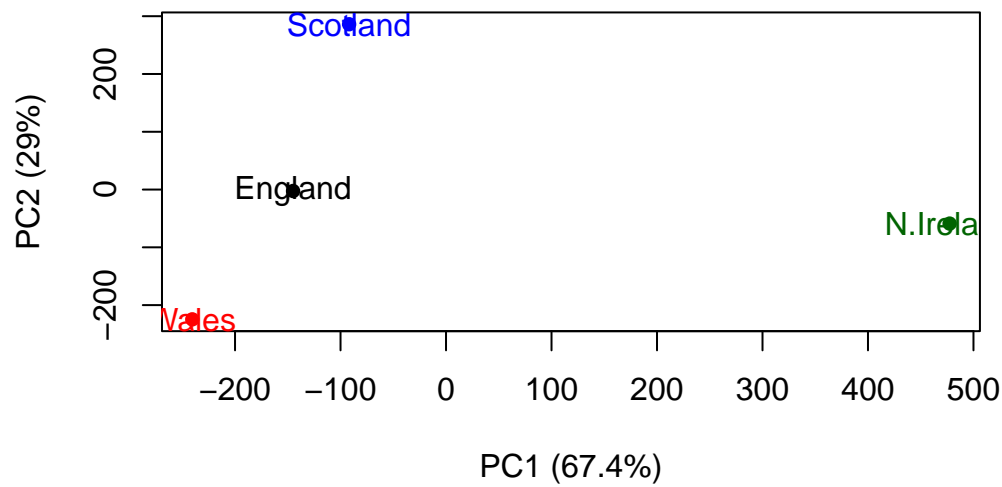
```
pca$x
```

	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

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



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

