Class07: Machine Learning I

Patric Young

In this class, we will explore clustering and dimensionalitly reduction methods.

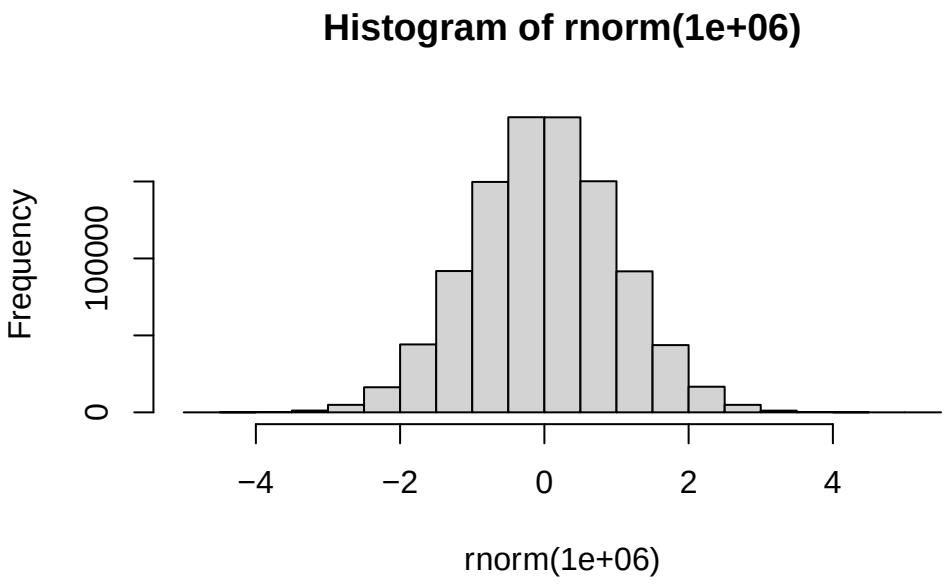
K-means

Make up some data input where we know what the answer should be.

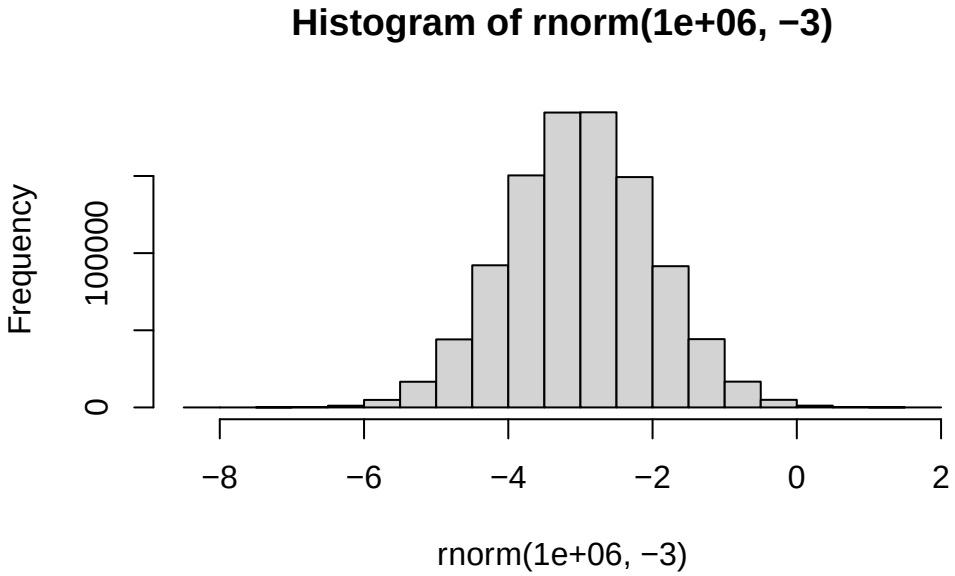
```
rnorm(10)
```

```
 [1] -0.2818977 -1.2230743 -2.0806455 -0.5147156 -0.9999234  0.9765028
 [7]  0.5977044 -0.9574892 -0.7613900 -0.9362476
```

```
hist(rnorm(1000000))
```




```r
hist(rnorm(1000000, -3))
```



```
tmp <- c(rnorm(30, -3), rnorm(30, +3))
rev(tmp)
```

```
 [1]  1.1231590  3.3001995  3.3233940  3.2075172  3.9859999  2.8362672
 [7]  2.7634545  2.2241246  3.1615870  3.2053750  2.9858055  1.6589509
[13]  3.2580545  2.2997760  4.0347388  3.2012833  2.7639195  2.9858991
[19]  2.6317338  2.6067408  3.0387654  2.8691472  3.0733612  2.3926644
[25]  3.2506170  4.2708893  5.2997571  0.3923686  2.3794689  3.5161459
[31] -1.8383950 -3.5187156 -3.6714231 -3.5866661 -3.1325229 -3.0286054
[37] -3.4406947 -5.0378522 -2.2680318 -2.0027523 -2.5896038 -2.4702351
[43] -3.2380719 -3.2574738 -3.2479199 -4.0585657 -3.7084236 -4.3478788
[49] -3.3263678 -2.1496279 -4.1157779 -2.5225674 -2.9998211 -1.8038913
[55] -3.2236820 -3.0933717 -3.2326470 -3.3746143 -2.9388862 -2.2165807
```

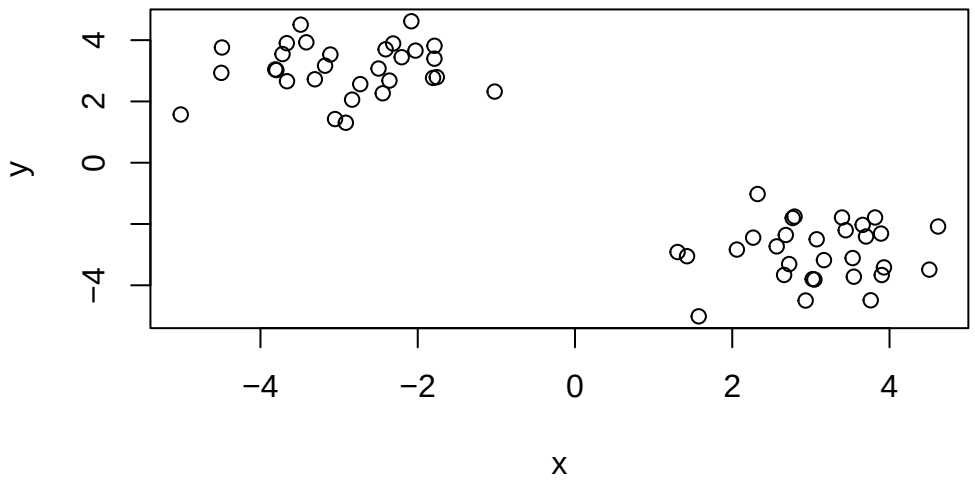
```
#x <- cbind(tmp, rev(tmp))
```

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

```
            x        y
[1,] -2.914377 1.304824
[2,] -3.797221 3.022648
[3,] -1.805180 2.768545
[4,] -2.834438 2.058803
[5,] -3.719608 3.546235
[6,] -2.204355 3.442822
```

Quick plot of x to see the two groups at -3, +3 and +3, -3

```
plot(x)
```



Use the `kmeans()` function setting k to 2 and nstart=20. How many things in here are not default. In this case its x and centers. How many clusters we want goes inside the parenthesis.

```
km <- kmeans(x, centers = 2, nstart=20)
km
```

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

Cluster means:
          x         y
1  3.069384 -2.906371
2 -2.906371  3.069384

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 1 1

Within cluster sum of squares by cluster:
[1] 45.86335 45.86335
 (between_SS / total_SS =  92.1 %)
```

```
Available components:

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

Q1. How many points are in each cluster?

```
km$size
```

```
[1] 30 30
```

Q2. What component if your result details

- 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 1 1
```

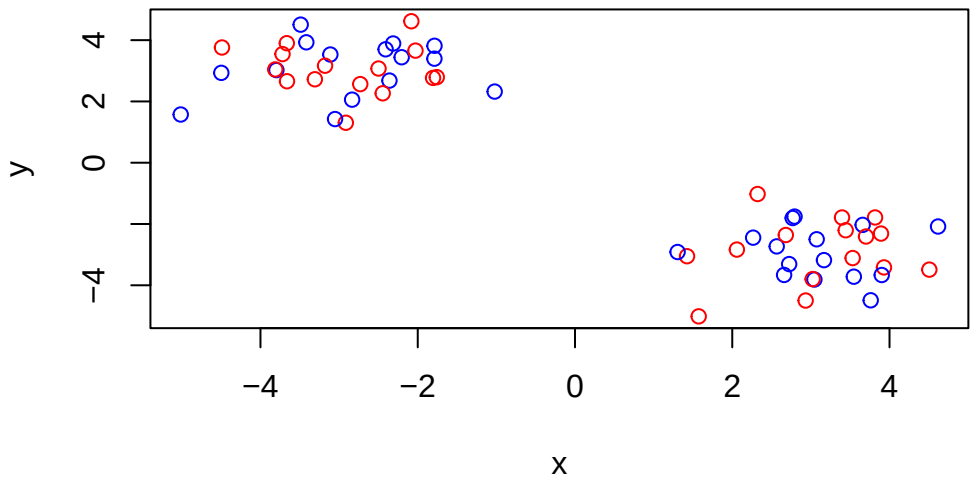
- cluster center?

```
km$center
```

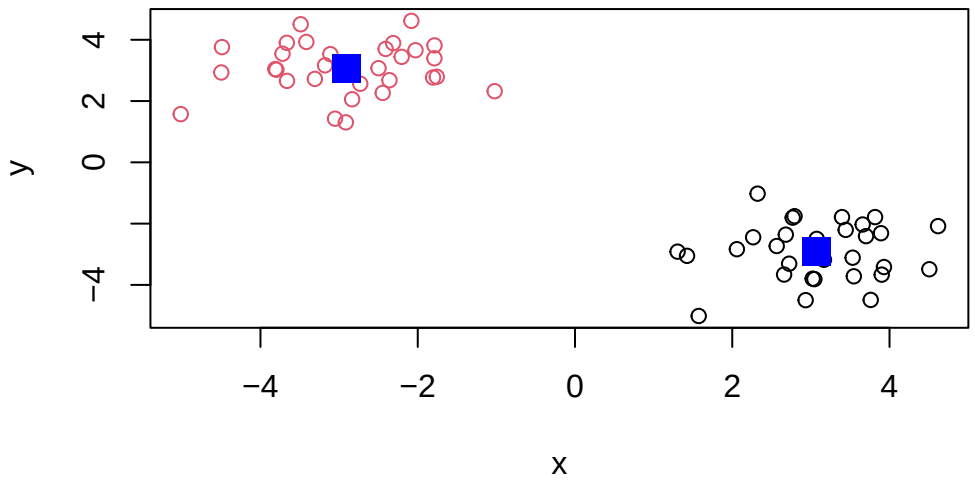
```
          x         y
1  3.069384 -2.906371
2 -2.906371  3.069384
```

Q3. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

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

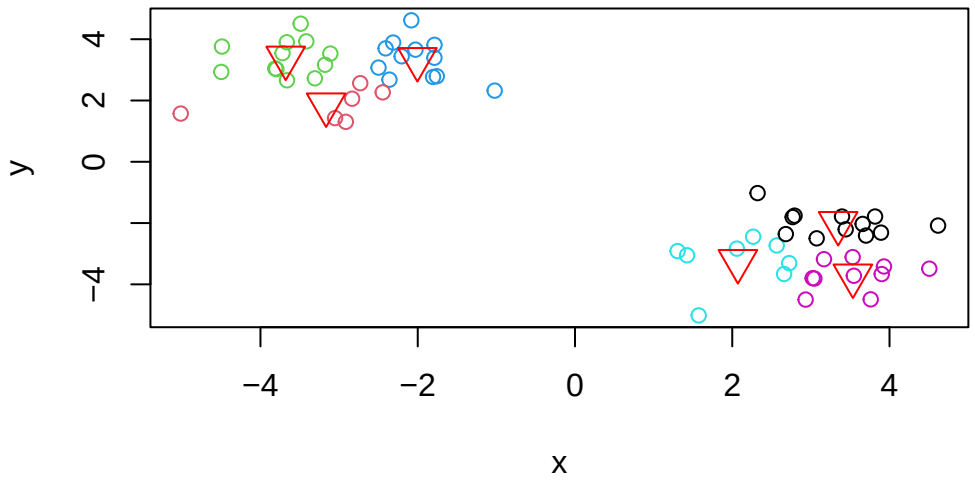



```
plot(x, col=km$cluster)
points(km$centers, col="blue", pch=15, cex=2)
```



Play with kmeans and ask for different number of clusters. kmeans calculates distance between all the points, then updates where the center is.

```
km <- kmeans(x, centers = 6, nstart=20)
plot(x, col=km$cluster)
points(km$centers, col="red", pch=6, cex=2)
```



Hierarchical Clustering

This is another very useful and widely employed clustering method which has the advantage over `kmeans()` in that it can help reveal the _____ of the true grouping in your data.

The `hclust()` function wants a distance matrix as input. Use `dist()`

kmeans(x, centers=2)

hclust(d)

cutree(hc, k=2)

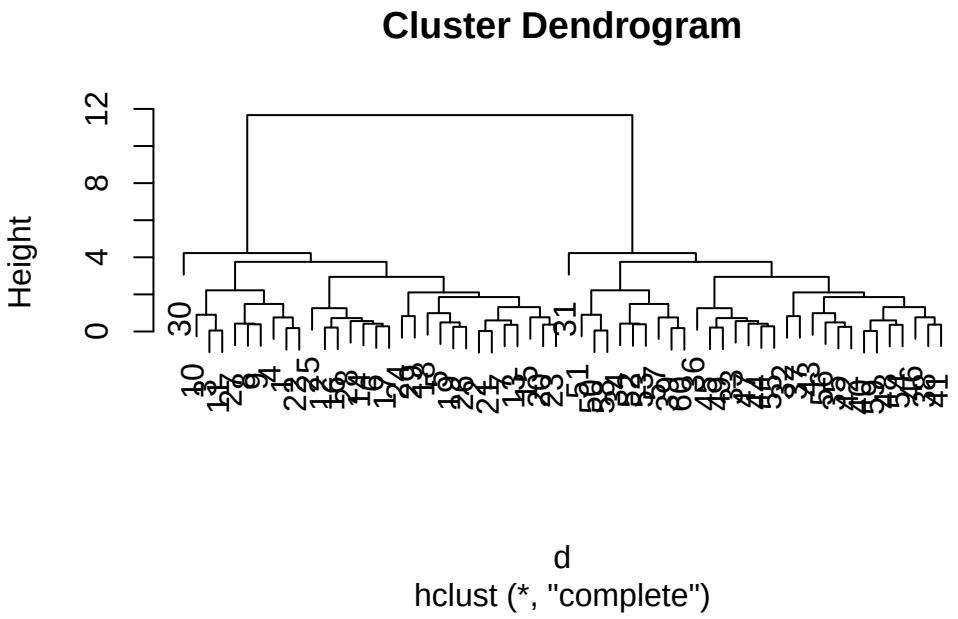
```
d <- dist(x)
hc <- hclust(d)
```

```
hc
```

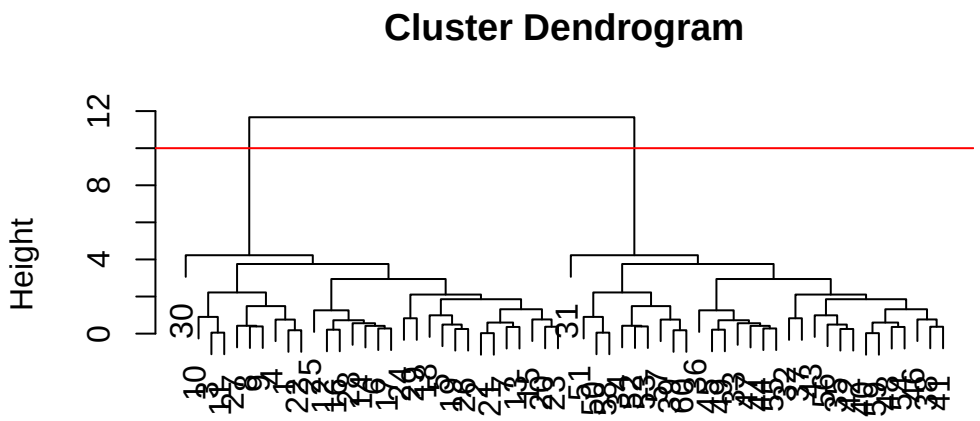
```
Call:
hclust(d = d)

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

There is a plot method for `hclust()` results.

```
plot(hc)
```




```
plot(hc)
abline(h=10, col="red")
```



To get my cluster membership vector, I had to "cut" my tree to yield sub-trees or branches with all the members of a given cluster residing on the same cut branch. The function to do this is called `cutree()`.

```
cutree(hc, h=10)
```

```
 [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
```

```
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 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
```

It is often helpful to use the `k=` argument to cutree rather than the `h=` height of cutting with `cutree()`. This will cut the tree to yield the number of clusters you want.

```
grps <- cutree(hc, k=4)
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 2 3 4 4 4 4 4 4 4
[39] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
```

KM <- kmeans(x, centers=2)

HC <- hclust(dist(x))

GRPS <- cutree(hc, k=2)

PCA <-prcomp(t(x))

Principal Component Analysis (PCA)

The base R function for PCA is called `prcomp()`. We use it to reduce dimensionality, which is everything you can measure about a data set. We want to visualize the most important things without losing information. Principal components are new low dimensional axis or surfaces closest to the observations. A line of best fit.

PCA of UK Foods

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

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

I could use these three below functions to find out the number of rows and columns: `dim(x)` or `ncol(x)` or `nrow(x)`

```
rownames(x) <- x[,1]
x <- x[,-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
```

```
dim(x)
```

```
[1] 17  4
```

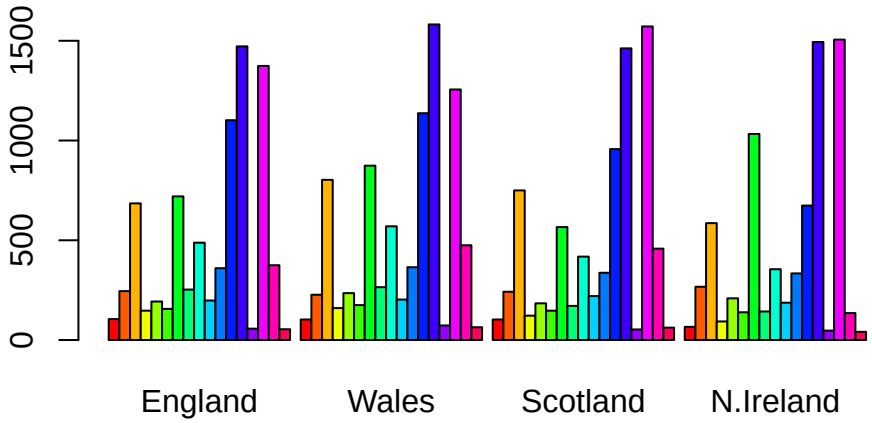
```
x <- read.csv(url, row.names=1)
head(x)
```

```
              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?

I believe the second approach to the "row-names problem" is the superior method because it is more efficient. Compared to the first method, which began deleting columns upon running because of the `x, -1`, the second method does not delete anything. Furthermore, there are less lines of text involved.

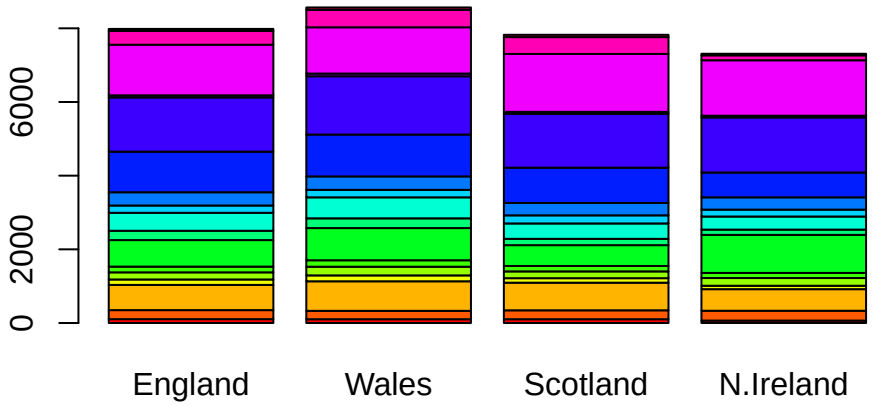
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

If we change `besides=T` to `besides=F`, the following plot is created. The bars for each country are wider and furthermore stacked on top of one another.

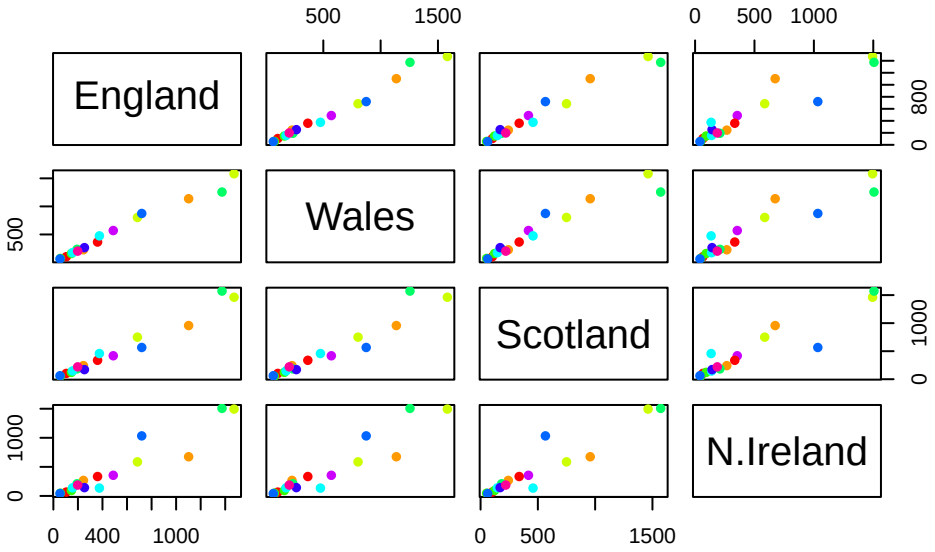
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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?

The following code further transforms the data and maps the data for all four countries onto pairwise plots. This allows us to see the distribution of single variables and the relationship between multiple variables. If a given point lies on the diagonal for a given plot, it means that that particular point has association with other points and clusters. It is part of a trend.

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



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

N. Ireland has a dramatically lower standard deviation compared to the other countries of the UK in terms of this data-set. Its STDEV is less than 1 while the other countries' STDEV is over 50. N. Ireland also has a Proportion of Variance of 0 while the other countries have values greater than 0.

```
pca <- prcomp( t(x))
summary(pca)
```

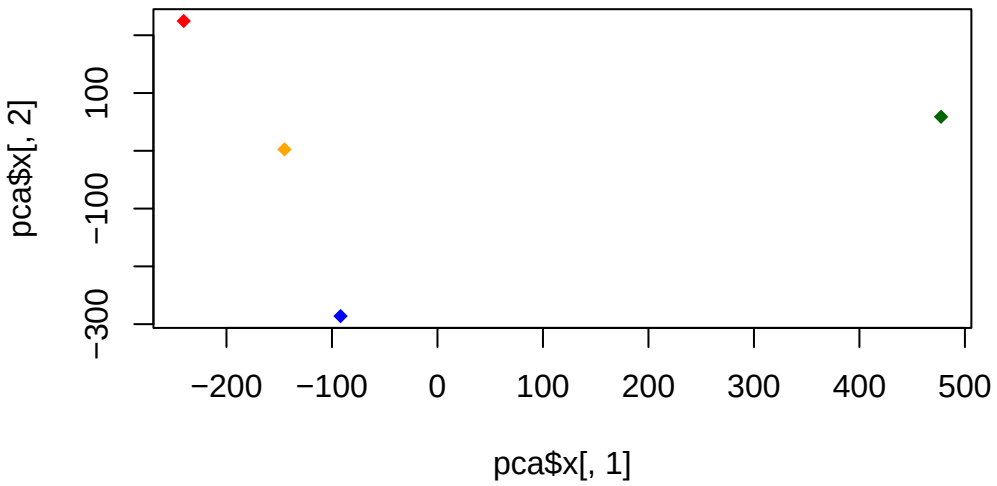
```
Importance of components:
                            PC1       PC2      PC3        PC4
Standard deviation     324.1502 212.7478 73.87622 4.189e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

A "PCA plot" (aka "Score Plot, PC1vPC2 plot, etc) is a plot comparing two plots

```
pca$x
```

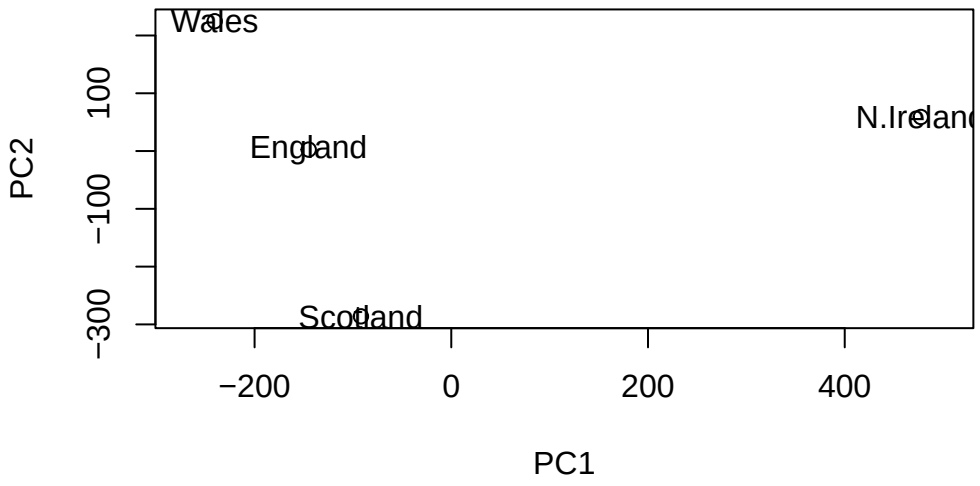
```
                PC1         PC2         PC3           PC4
England   -144.99315    2.532999 -105.768945  2.842865e-14
Wales     -240.52915  224.646925   56.475555  7.804382e-13
Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

```r
plot(pca$x[,1], pca$x[,2], col=c("orange", "red", "blue", "darkgreen"), pch=18)
```



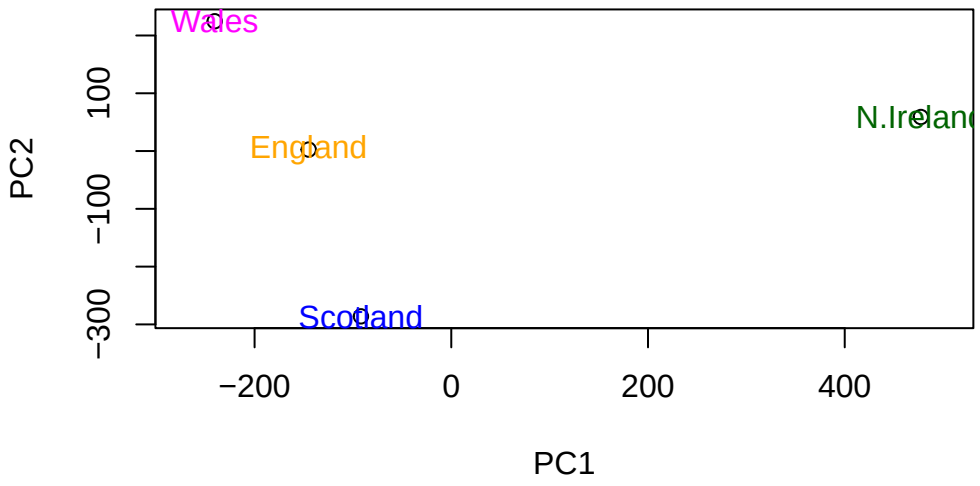
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points

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



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(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x),col=c("orange", "magenta", "blue", "darkgreen"), pc
```



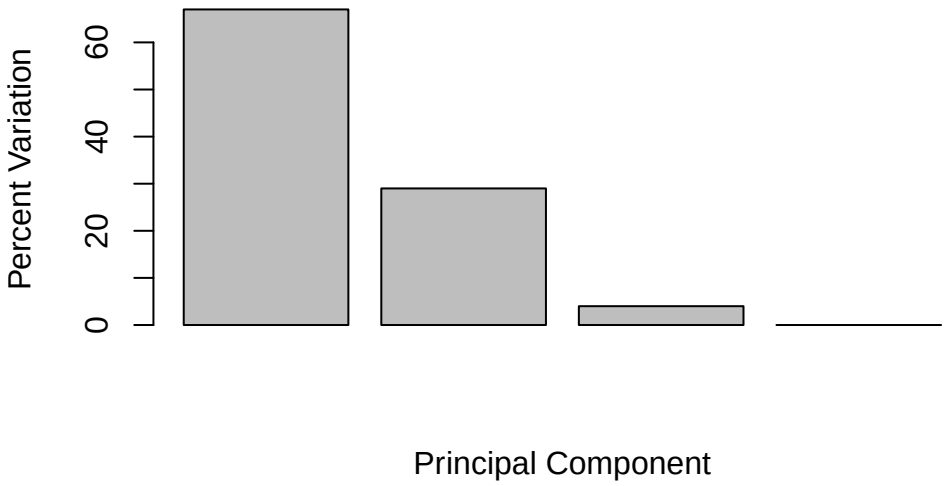

```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

```
[1] 67 29  4  0
```

```
z <- summary(pca)
z$importance
```

```
                             PC1       PC2      PC3           PC4
Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```




```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```