Class 7: Clustering and PCA

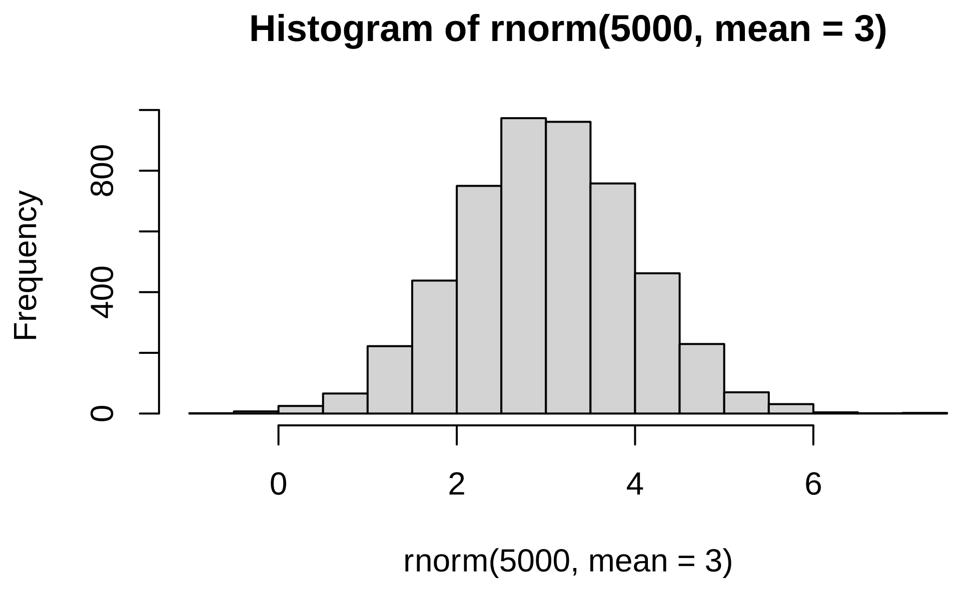
A16125573

Clustering

First let's make up dome 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))
```



```
# mean and sd has a default value, so should worry most about the one that doesn't (n)
```

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

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

```
 [1]  3.2084244  3.7054068  3.4705621  4.0661434  2.1897668  4.4194926
 [7]  3.1678866  1.9749433  2.1956443  3.9203972  2.7047327  2.7443981
[13]  3.8872643  3.6426726  4.9130119  1.9327703  4.5021159  3.8372291
[19]  4.2150368  0.3765644  3.3919821  0.8314147  3.7281302  3.4500863
[25]  3.6925781  3.3809910  2.2970447  3.1436025  1.2602399  2.2326780
[31] -3.9284313 -2.0047439 -3.7890276 -5.4954627 -4.2042352 -3.4685389
[37] -1.5980628 -3.9160934 -5.3001799 -1.5341460 -2.8632875 -2.9334831
[43] -2.9904012 -3.1754922 -2.1628485 -4.0954682 -2.8243610 -1.8922640
[49] -2.7839677 -3.2195891 -3.0894440 -3.5936179 -2.6009795 -3.7307964
[55] -3.2295156 -3.6557303 -2.7683935 -2.2659478 -2.6431553 -3.5518541
```

```
# cbind()
rev(c(rnorm(30, mean=3), rnorm(30, mean=-3))) # this reverses
```

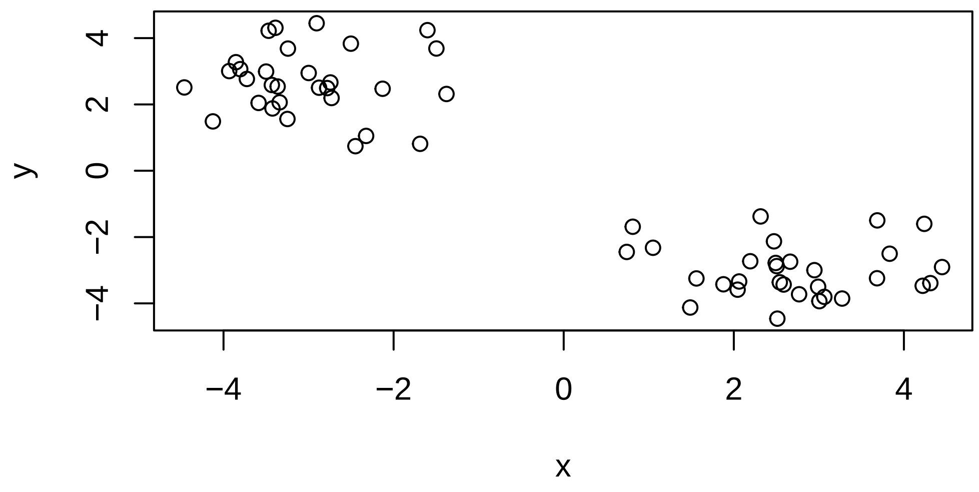
```
 [1] -4.957870 -2.039953 -2.822553 -3.083814 -3.793521 -4.010679 -1.624195
 [8] -3.772316 -2.706706 -1.521207 -2.610659 -4.526746 -3.185522 -3.375776
[15] -4.029854 -3.205724 -2.026053 -3.996235 -1.251944 -1.842881 -2.646989
[22] -3.327836 -2.469409 -4.091714 -2.925673 -2.805249 -2.553231 -3.095059
[29] -3.742598 -3.115158  3.729554  3.195838  4.005090  4.596818  2.562425
[36]  1.526517  4.807164  3.802973  3.453638  3.909998  2.142385  3.294790
[43]  3.250758  4.192773  3.154904  2.261173  2.372802  3.499841  1.658599
[50]  3.725033  2.692871  3.001185  3.191584  3.432434  3.625258  5.123551
[57]  3.204323  6.352056  4.107484  5.263905
```

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

```
 [1]  4.2396827  2.1934619  2.6626342  2.5859659  3.2736025  3.0643913
 [7]  0.7411759  1.0499690  3.0059509  2.3151920  1.4878670  0.8115113
[13]  4.3109357  2.4723246  1.5606790  2.9477691  4.2211109  2.5121841
[19]  3.6841331  3.6870870  2.9915633  2.5407447  2.4924749  2.7678030
[25]  2.5044406  2.0448785  1.8782077  4.4486437  3.8326469  2.0627920
[31] -3.3401594 -2.5026304 -2.9051613 -3.4244809 -3.5874678 -2.8765918
```

```
[37] -3.7255310 -2.7810177 -3.3638409 -3.4992866 -1.4970091 -3.2426324
[43] -4.4608747 -3.4696607 -2.9987337 -3.2484622 -2.1292967 -3.3894992
[49] -1.6883479 -4.1248860 -1.3782282 -3.9333013 -2.3230910 -2.4490755
[55] -3.8035898 -3.8542453 -3.4318094 -2.7434786 -2.7294987 -1.6021322
```

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



K-means clustering.

Very popular clustering method, especially for big datasets, that we can use with the `kmeans()` function in base R.

```
# kmeans() #needs 2 things/inputs

km <- kmeans(x, centers=2)
# centers = # of clusters

km
```

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

```
Cluster means:
          x         y
1 -3.016801  2.679727
2  2.679727 -3.016801

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] 47.68179 47.68179
 (between_SS / total_SS =  91.1 %)

Available components:

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

```r
# we know the answer should be 30, 30
# cluster means = the center points of the clusters
# clustering vector = shows which cluster each point belongs to
# available components = ?
```

```r
km$size
```

```
[1] 30 30
```

```r
# example data
tmp <- c(rnorm(30, 3), rnorm(30, -3))
x <- data.frame(x=tmp, y=rev(tmp))

#plot(x)

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

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

Cluster means:
```

```
          x         y
1  3.178960 -3.104514
2 -3.104514  3.178960

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] 55.20742 55.20742
 (between_SS / total_SS =  91.5 %)

Available components:

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

Q. How many points are in each cluster?

```
km$size
```

```
[1] 30 30
```

Q. How do we get to cluster membership/assignment?

```
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
```

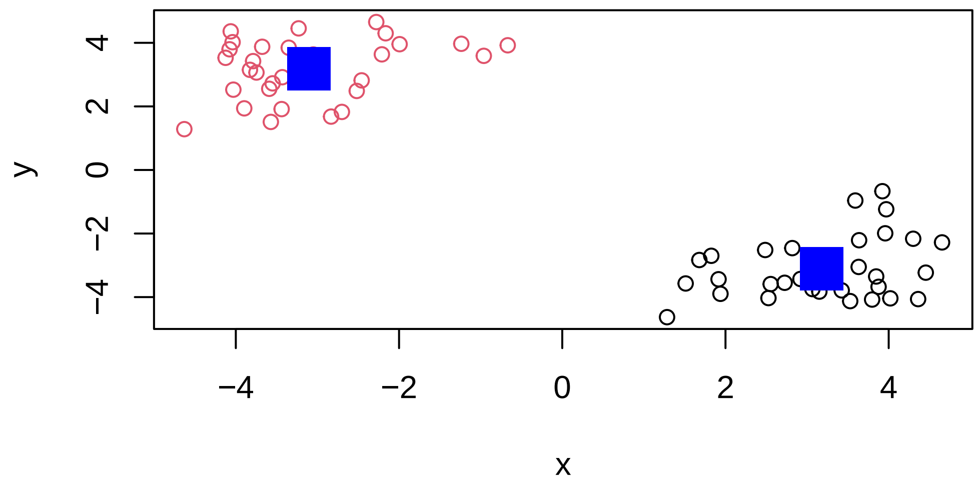
Q. How many cluster centers?

```
km$centers
```

```
          x         y
1  3.178960 -3.104514
2 -3.104514  3.178960
```

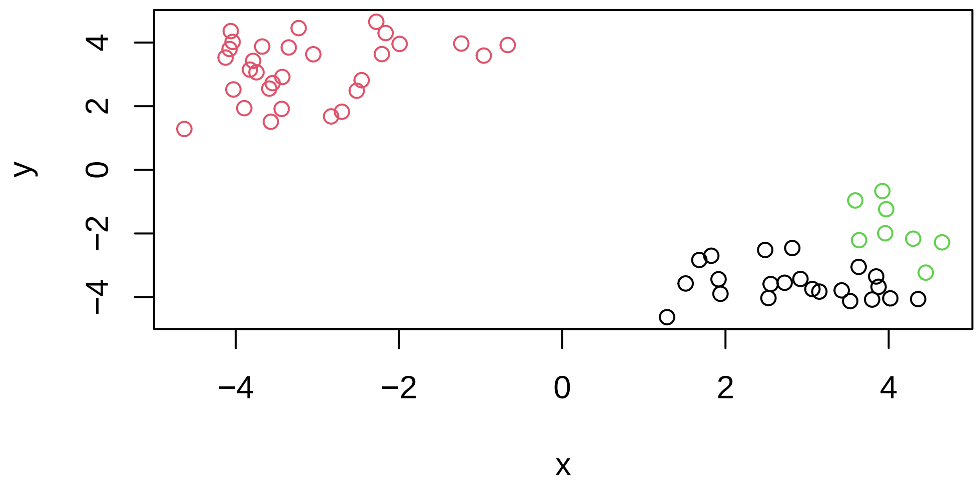
Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```r
# mycols <- km$cluster
plot(x, col=km$cluster)
points(km$centers, col="blue", pch=15, cex=3)
```



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

```r
km3 <- kmeans(x, centers=3)
plot(x, col=km3$cluster)
```



Hierarchical Clustering

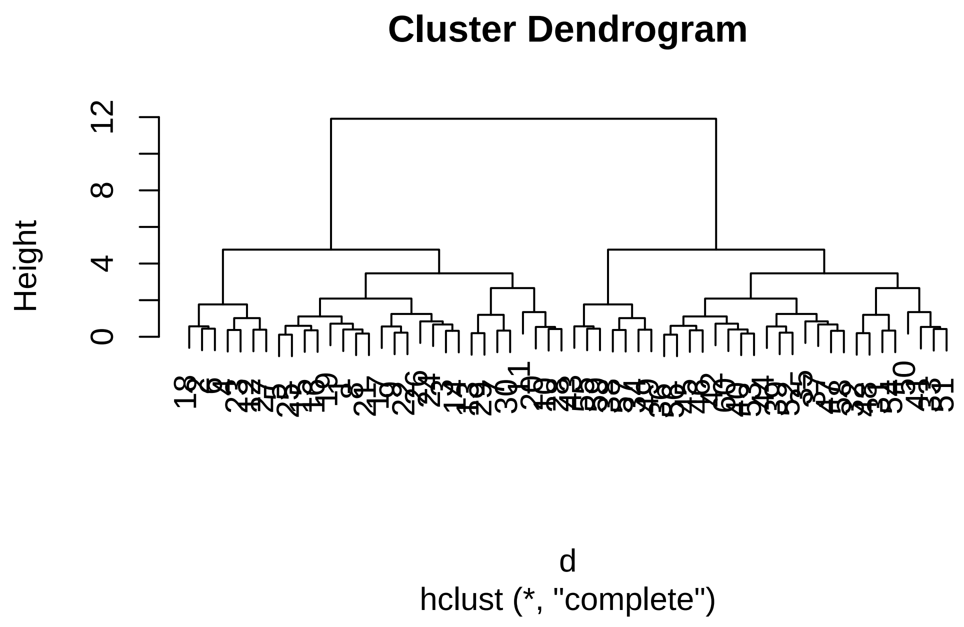
We can use the `hclust()` function for Hierarchical Clustering. Unlike `kmeans()`, where we could just pass in our data as input, we need to give `hclust()` a "distance matrix"

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

```
d <- dist(x) # a distance matrix
hc <- hclust(d)
hc
```

```

Call:
hclust(d = d)

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

```
plot(hc)
```



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

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

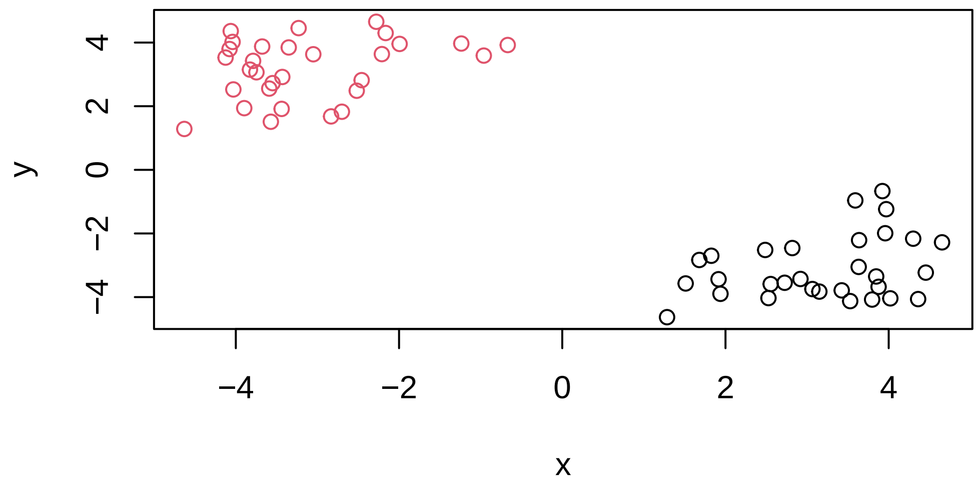
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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

```
cutree(hc, k=2) #cut it into 2 groups
```

```
 [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
```

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



Principal Component Analysis (PCA)

PCA of UK food

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

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

```
[1] 17
```

```r
ncol(x)
```

```
[1] 4
```

```r
dim(x)
```

```
[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
```

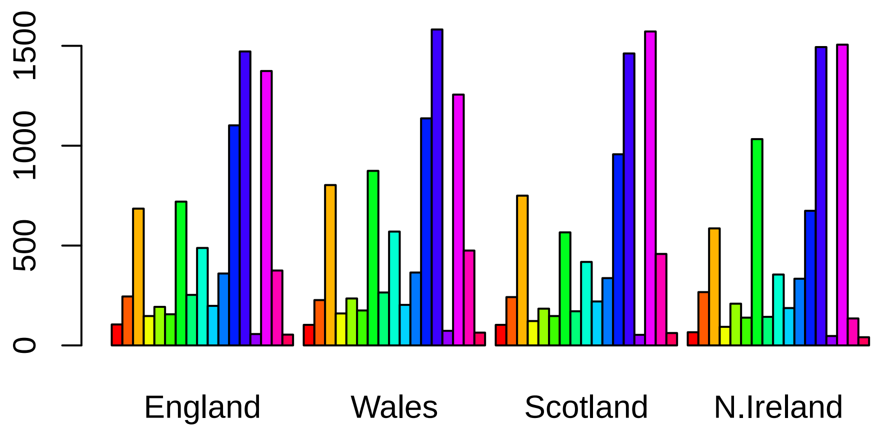
```
#rownames(x) <- x[,1]
#x <- x[,-1]
#head(x)
dim(x)
```

```
[1] 17  4
```

```
# I don't prefer this method because each time you run it, you would drop a column
```

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

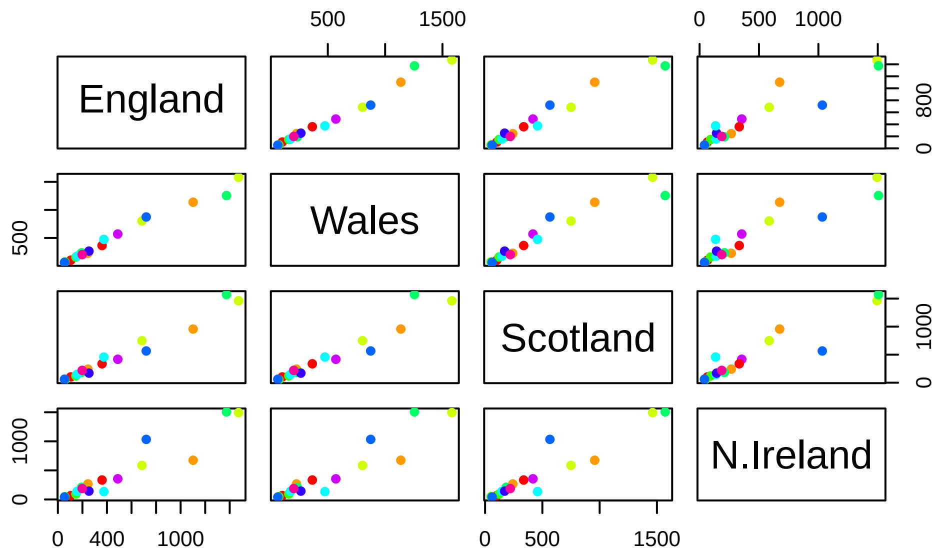
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



```
# changing beside to F will generate the other bar plot
```

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)
```



```
# the figure shows
# each colored dot is a food category
# if a points lies on the diagonal, it means that the the x and y axis countries comsumed
# points above the diagonal means that the y axis country has consumed more
```

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

```
# prcomp
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) # we want x
```

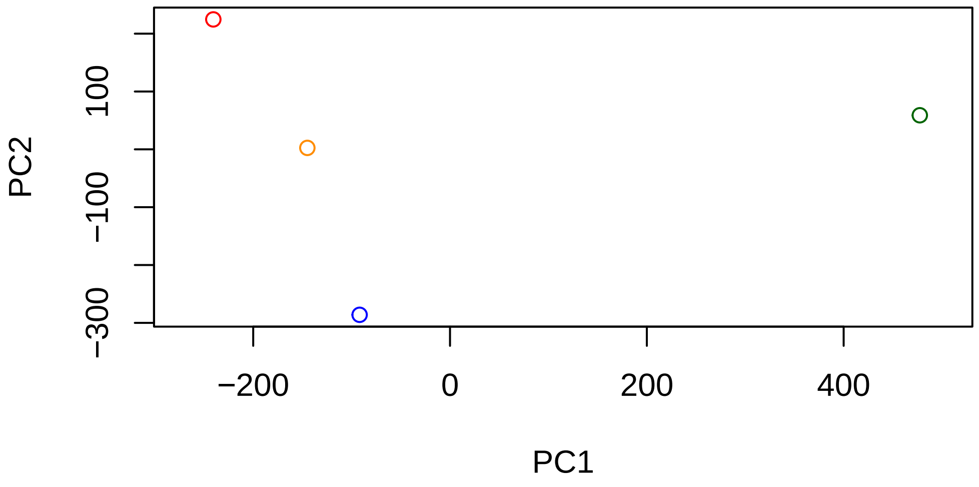
```
$names
```

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

$class
[1] "prcomp"
```

```
pca$x # shows where each country lies on the new axis
```

```
                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 PC1 vs PC2

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



```
# text(pca$x[,1], pca$x[,2], colnames(x))
```