Class 7:Machine Learning 1

Yerin Go (A16272901)

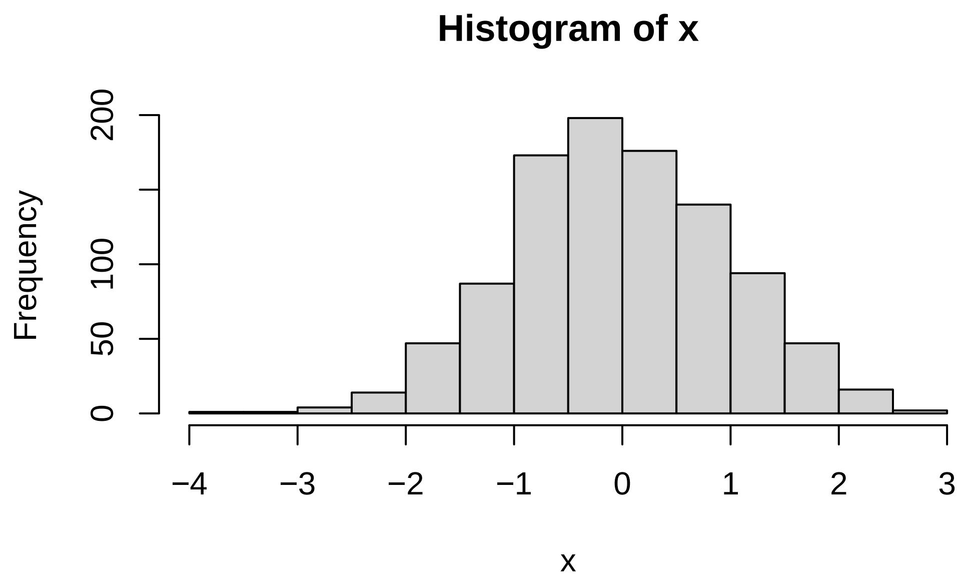
#Clustering Methods

The broad goal here is to find groupings (clusters) in your input data

##kmeans

First, let's make up some data to cluster.

```
x <- rnorm(1000)
hist(x)
```



Make a vector of length 60 with 30 points centered at -3 and 30 points centered at +3

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

```
 [1] -3.552130 -3.986926 -4.352605 -1.134901 -1.789953 -2.340931 -4.161691
 [8] -2.855190 -1.470861 -2.659238 -3.514743 -4.112074 -2.842861 -3.536923
[15] -4.008434 -3.952195 -4.196491 -3.601648 -3.950042 -1.098229 -4.237116
[22] -2.539294 -4.300440 -4.288732 -1.924482 -3.687145 -3.019116 -5.292654
[29] -2.622738 -1.878402  3.302369  1.615299  2.204554  5.286726  2.748953
[36]  4.012986  3.015273  3.140110  5.248738  2.862859  4.136961  3.848212
[43]  1.641793  2.797816  5.416417  2.419450  1.945332  2.819614  3.889826
[50]  3.401286  3.181459  4.889458  2.482277  1.519214  2.436674  2.262992
[57]  2.167018  2.982171  4.286173  4.357386
```

I will not make a wee x and y dataset with 2 groups of points.

```r
rev(c(1:5))
```

```
[1] 5 4 3 2 1
```

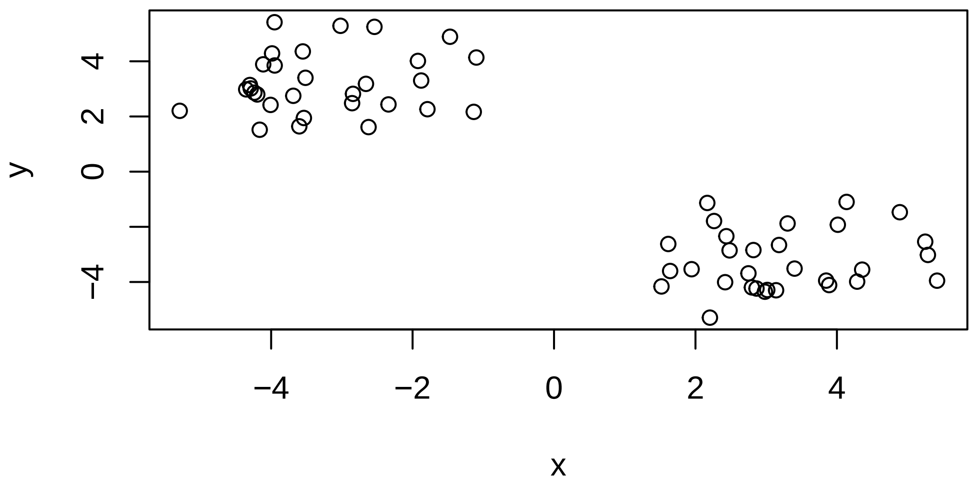
```r
x <- cbind(x=tmp, y=rev(tmp))
x
```

```
              x         y
 [1,] -3.552130  4.357386
 [2,] -3.986926  4.286173
 [3,] -4.352605  2.982171
 [4,] -1.134901  2.167018
 [5,] -1.789953  2.262992
 [6,] -2.340931  2.436674
 [7,] -4.161691  1.519214
 [8,] -2.855190  2.482277
 [9,] -1.470861  4.889458
[10,] -2.659238  3.181459
[11,] -3.514743  3.401286
[12,] -4.112074  3.889826
[13,] -2.842861  2.819614
[14,] -3.536923  1.945332
[15,] -4.008434  2.419450
[16,] -3.952195  5.416417
```

```
[17,] -4.196491  2.797816
[18,] -3.601648  1.641793
[19,] -3.950042  3.848212
[20,] -1.098229  4.136961
[21,] -4.237116  2.862859
[22,] -2.539294  5.248738
[23,] -4.300440  3.140110
[24,] -4.288732  3.015273
[25,] -1.924482  4.012986
[26,] -3.687145  2.748953
[27,] -3.019116  5.286726
[28,] -5.292654  2.204554
[29,] -2.622738  1.615299
[30,] -1.878402  3.302369
[31,]  3.302369 -1.878402
[32,]  1.615299 -2.622738
[33,]  2.204554 -5.292654
[34,]  5.286726 -3.019116
[35,]  2.748953 -3.687145
[36,]  4.012986 -1.924482
[37,]  3.015273 -4.288732
[38,]  3.140110 -4.300440
[39,]  5.248738 -2.539294
[40,]  2.862859 -4.237116
[41,]  4.136961 -1.098229
[42,]  3.848212 -3.950042
[43,]  1.641793 -3.601648
[44,]  2.797816 -4.196491
[45,]  5.416417 -3.952195
[46,]  2.419450 -4.008434
[47,]  1.945332 -3.536923
[48,]  2.819614 -2.842861
[49,]  3.889826 -4.112074
[50,]  3.401286 -3.514743
[51,]  3.181459 -2.659238
[52,]  4.889458 -1.470861
[53,]  2.482277 -2.855190
[54,]  1.519214 -4.161691
[55,]  2.436674 -2.340931
[56,]  2.262992 -1.789953
[57,]  2.167018 -1.134901
[58,]  2.982171 -4.352605
[59,]  4.286173 -3.986926
```

```
[60,]  4.357386 -3.552130
```

```
plot(x)
```



```
k <-kmeans(x, centers=2)
k
```

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

Cluster means:
          x         y
1 -3.230273  3.210646
2  3.210646 -3.230273

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] 70.16227 70.16227
 (between_SS / total_SS =  89.9 %)
```

```
Available components:

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

. Q. From your result object ‘k’ how many points are in each cluster?

```
k$size
```

```
[1] 30 30
```

. Q. What “component” of your results object details the cluster membership?

```
k$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
```

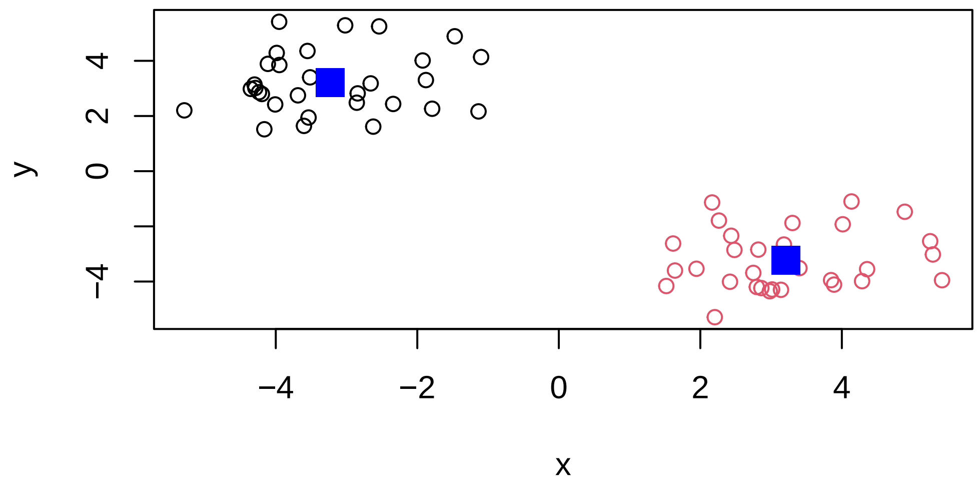
. Q. Cluster centers?

```
k$centers
```

```
          x         y
1 -3.230273  3.210646
2  3.210646 -3.230273
```

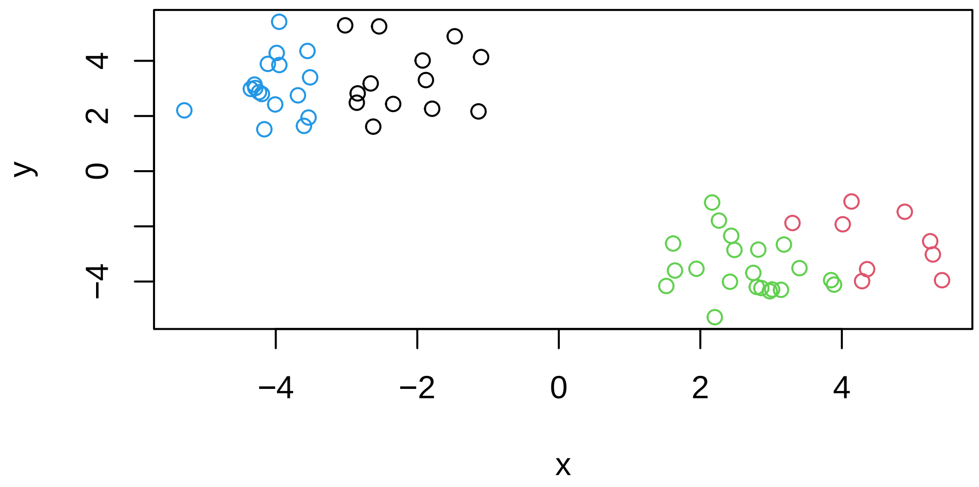
. Q. Plot of our clustering results?

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



We can cluster into 4 groups.

```
# kmeans
k4 <- kmeans(x, centers=4)
#plot results
plot(x, col=k4$cluster)
```



A big limitation of kmeans is that it does waht you ask even if you ask for silly clusters. # Hierarchical Clustering

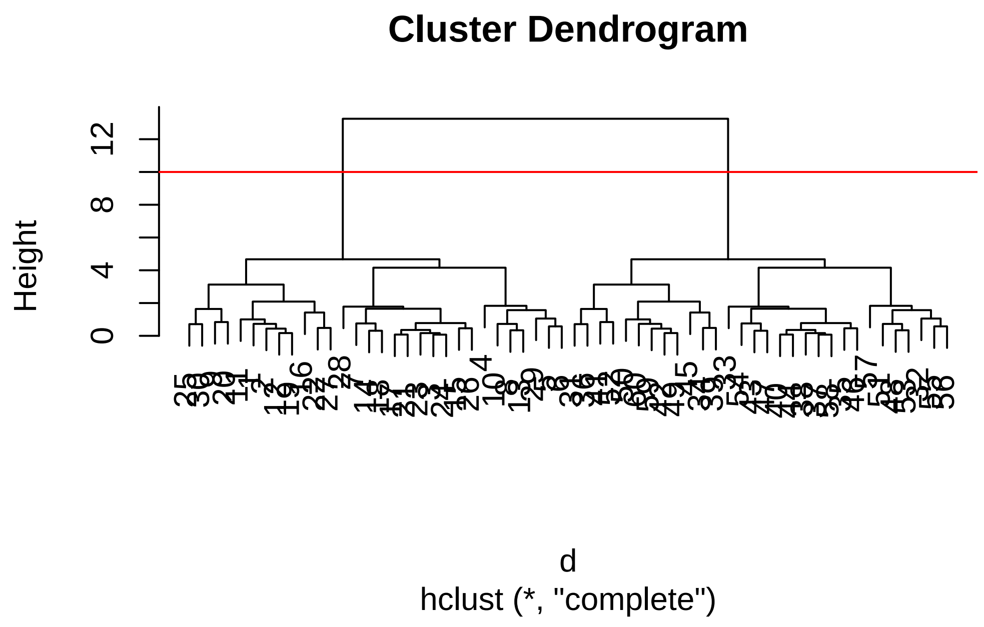
The main base R function for Hierarchical Clustering is `hclust()` Unlike `kmeans()` you cannot just pass it your data as input. You first need to calculate a distance matrix.

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

```

Call:
hclust(d = d)

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

Use `plot()` to view results.

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



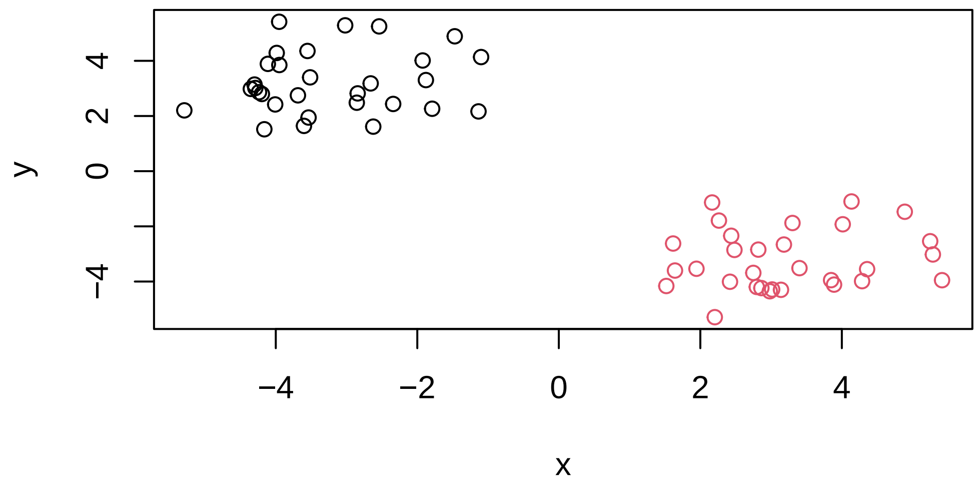
To make the "cut" and get our cluster membership vector we can use the `cutree()` function.

```r
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 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
```

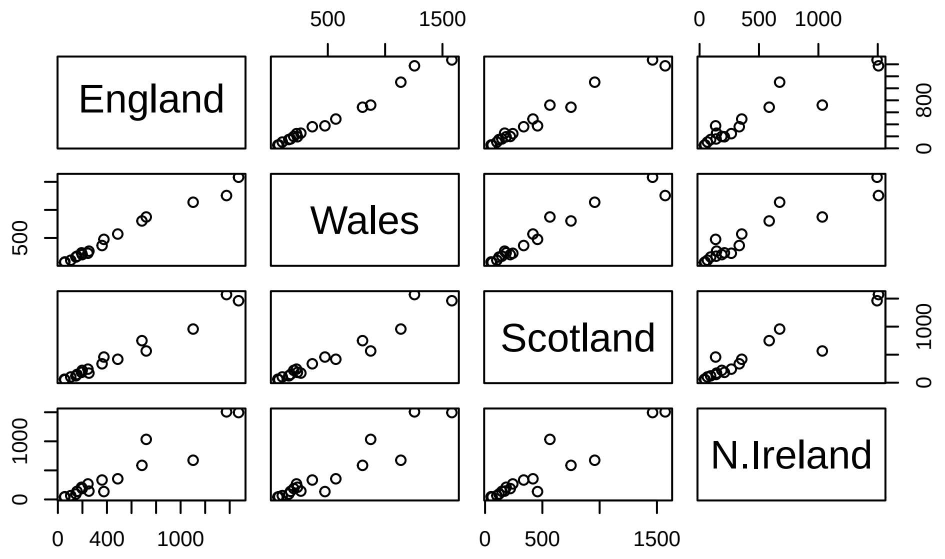
Make a plot of our data colored by hclust results.

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



#Principle Component Analysis (PCA) Here we will do PCA on some food data from the UK.

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



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

```
dim(x)
```

```
[1] 17  4
```

```
nrow(x)
```

```
[1] 17
```

```
ncol(x)
```

```
[1] 4
```

##PCA to the rescue The main "base" R function for PCA is called `prcomp()` Here we need to take the transpose of our input as we want the countries in the rows and foods as the columns.

```r
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
```

. Q. How much variance is captured to 2 PCs 96.5%

To make our main “PC score plot” or “PC1 vs. PC2 plot” or “PC plot” or “Ordination Plot”.

```r
attributes(pca)
```

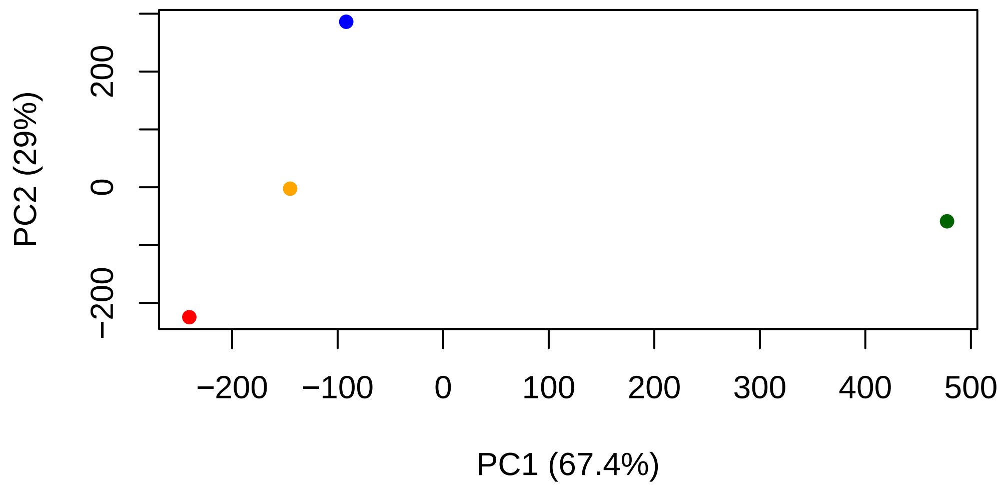
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"       

$class
[1] "prcomp"
```

We are after the `pca$x` result component to make out main PCA plot

```r
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
```

```r
mycols <- c("orange", "red", "blue","darkgreen")
plot(pca$x[,1], pca$x[,2], col=mycols, pch=16, xlab= "PC1 (67.4%)", ylab= "PC2 (29%)")
```



Another important result from PCA is how the original variables (in this case the foods) contribute to the PCs. This is contained in the `pca$rotation` object- folks often call this the “loadings” or “contributions” to the PCs.

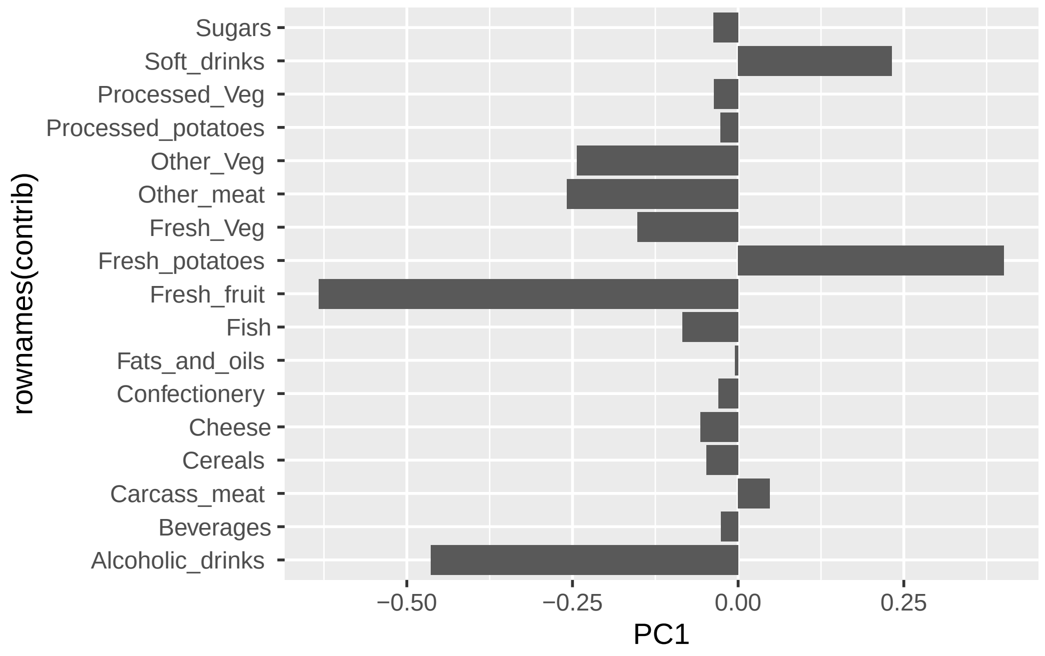
```
pca$rotation[,1]
```

```
          Cheese        Carcass_meat          Other_meat                Fish 
    -0.056955380         0.047927628        -0.258916658        -0.084414983 
  Fats_and_oils              Sugars      Fresh_potatoes           Fresh_Veg 
    -0.005193623        -0.037620983         0.401402060        -0.151849942 
      Other_Veg  Processed_potatoes       Processed_Veg         Fresh_fruit 
    -0.243593729        -0.026886233        -0.036488269        -0.632640898 
        Cereals           Beverages         Soft_drinks     Alcoholic_drinks 
    -0.047702858        -0.026187756         0.232244140        -0.463968168 
  Confectionery 
    -0.029650201 
```

We can make a plot along PC1.

```
library(ggplot2)
contrib <- as.data.frame(pca$rotation)
```

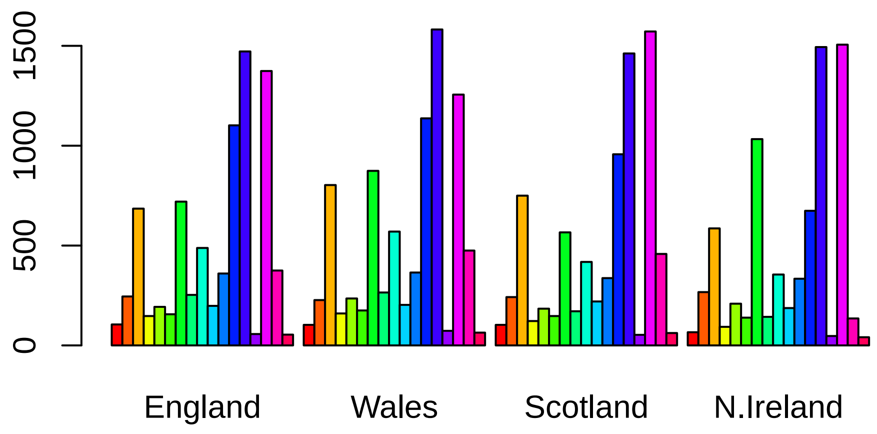
```
ggplot(contrib) +
 aes(PC1, rownames(contrib))+
  geom_col()
```



#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?

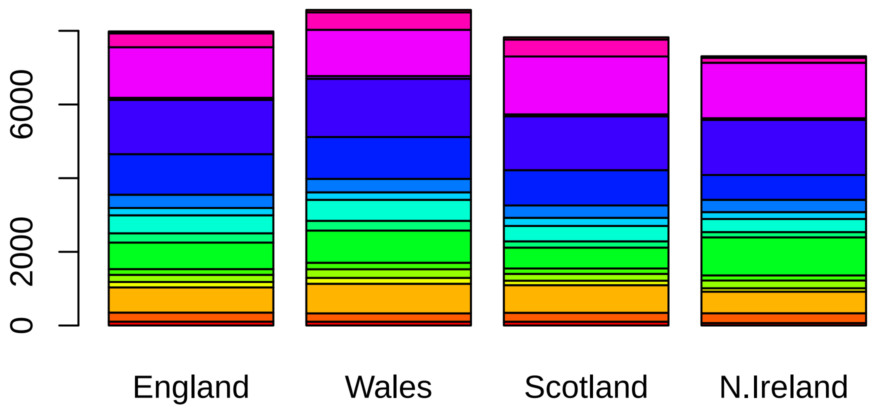
I prefer adding the part of the code `row.names=1` to the of the line of code `x <- read.csv(url, row.names=1)`. This approach is more robust than using the `rownames(x) <- x[,1]x <- x[,-1] head(x)` because everything I run the code the second way, it will override the function, therefore deleting the relative first row each time the code is run.

```
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?

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


0
2000
6000
England
Wales
Scotland
N.Ireland