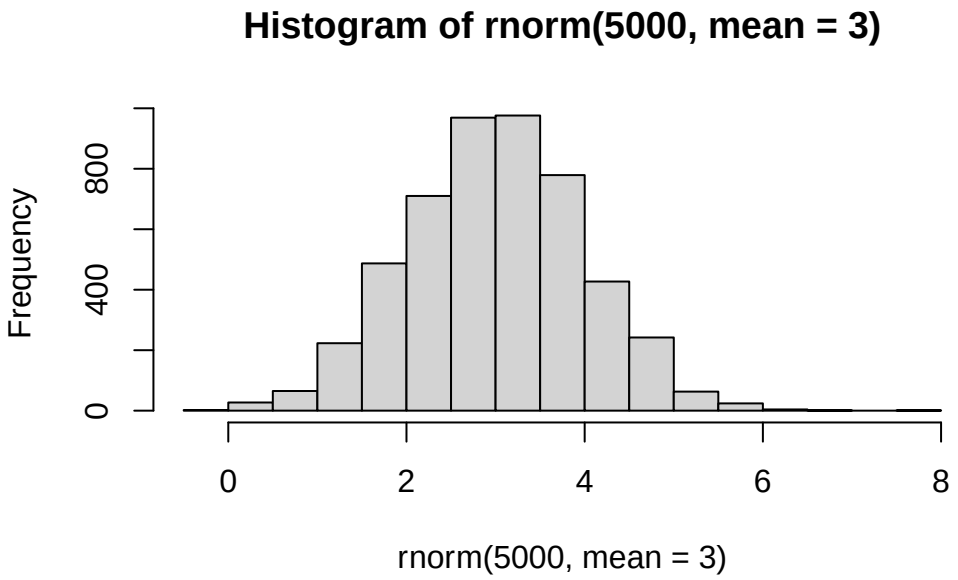
Clustering

First let's makeup some 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'.

```
hist(rnorm(5000,mean=3))
```




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

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

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

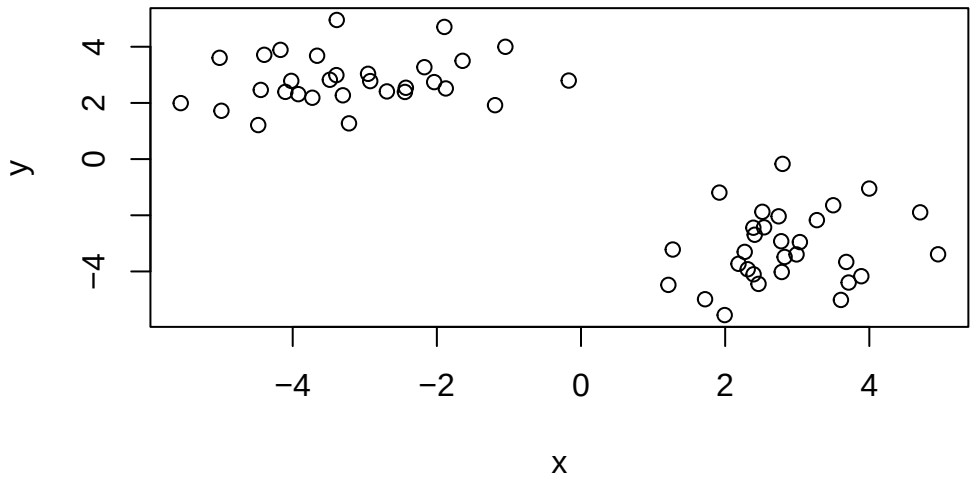
```
 [1]  3.6059709  2.7417943  3.0365300  2.7852153  2.4098417  4.7055276
 [7]  2.4627940  2.3943019  2.7960611  3.7134552  2.3126146  3.4990458
[13]  3.6801716  3.9987927  2.5400330  3.8890928  4.9539457  1.2120301
[19]  2.9904907  1.7209169  2.3909247  2.5158092  3.2723208  1.9933108
[25]  2.1849948  2.8247162  1.9197578  2.2711333  2.7776550  1.2728367
[31] -3.2190102 -2.9242796 -3.3041775 -1.1924530 -3.4846801 -3.7275224
```

```
[37] -5.5534776 -2.1732775 -1.8761589 -2.4430453 -4.9888259 -3.3939230
[43] -4.4782202 -3.3888683 -4.1694640 -2.4297753 -1.0492779 -3.6614196
[49] -1.6431405 -3.9223647 -4.3948481 -0.1703809 -4.1018595 -4.4429541
[55] -1.8957518 -2.6931376 -4.0199833 -2.9543630 -2.0370366 -5.0140950
```

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

```
 [1] -4.4082585 -2.3711487 -2.8779718 -2.7825818 -3.5346715 -5.2161048
 [7] -2.9562040 -1.3414632 -2.8428818 -3.4810844 -1.7192600 -1.8976365
[13] -1.3434320 -3.6717699 -2.0933268 -2.5584591 -1.4380299 -3.5894885
[19] -3.2588767 -2.6821879 -3.5962866 -1.3519058 -4.9701978 -3.5885502
[25] -3.4479279 -2.6490496 -1.9396412 -4.9204329 -3.6902266 -3.0983245
[31]  2.7381913  1.8132601  1.3430062  4.6981862  2.4285626  2.9728067
[37]  3.2112728  2.6902684  3.9683762  3.5130393  2.8229459  3.6464125
[43]  2.1136764  2.1245391  3.1008883  2.8800381  3.2103370  3.3776610
[49]  3.5612030  3.8381721  1.8677020  3.0578356  4.0982597  0.8809776
[55]  3.6582792  3.7332496  1.7006177  2.3471177  2.1343669  4.2805608
```

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



K means clustering

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

```
::: {.cell}

```{.r .cell-code}
#kmeans() need 2 inputs

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

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

Cluster means:
          x   y
1 2.829070 -3.158259
2 -3.158259 2.829070

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 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

Within cluster sum of squares by cluster:
[1] 70.88736 70.88736
 (between_SS / total_SS = 88.4 %)

Available components:

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

```{.r .cell-code}
  # centers = number of clusters
  # cluster means = the center points of the clusters
  # K-means clustering with 2 clusters of size 30, 30
```

:::

```{.r .cell-code}
  km$size
```

```
[1] 30 30
```

```
# Generate some example data for clustering

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 48, 12

Cluster means:
          x         y
1 -2.844720 -2.844720
2 -4.304434 -4.304434

Clustering vector:
 [1] 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1
[39] 1 1 2 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1

Within cluster sum of squares by cluster:
[1] 42.08990 12.01543
 (between_SS / total_SS =  43.1 %)

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] 48 12
```

Q. How many cluster centers?

```
km$centers
```

```
          x         y
1 -2.844720 -2.844720
2 -4.304434 -4.304434
```

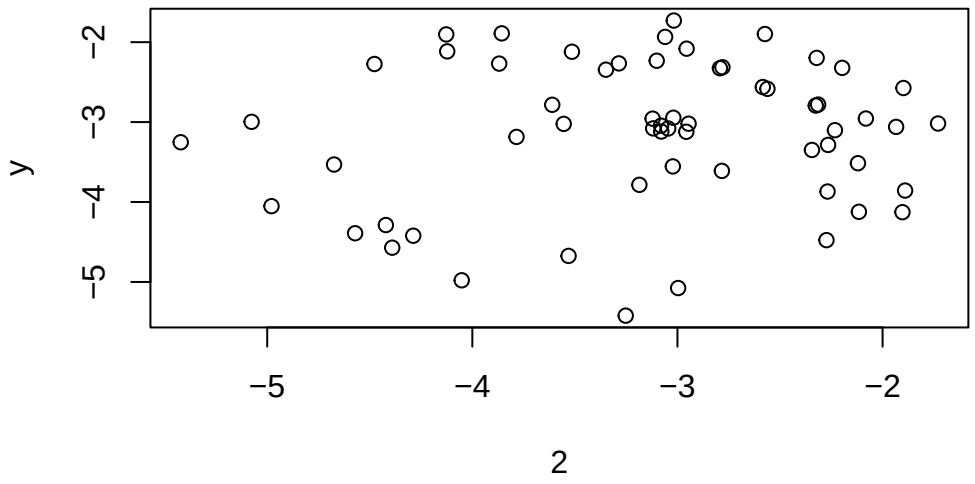
Q. How do we get to cluster membership/assignment?

```
km$cluster
```

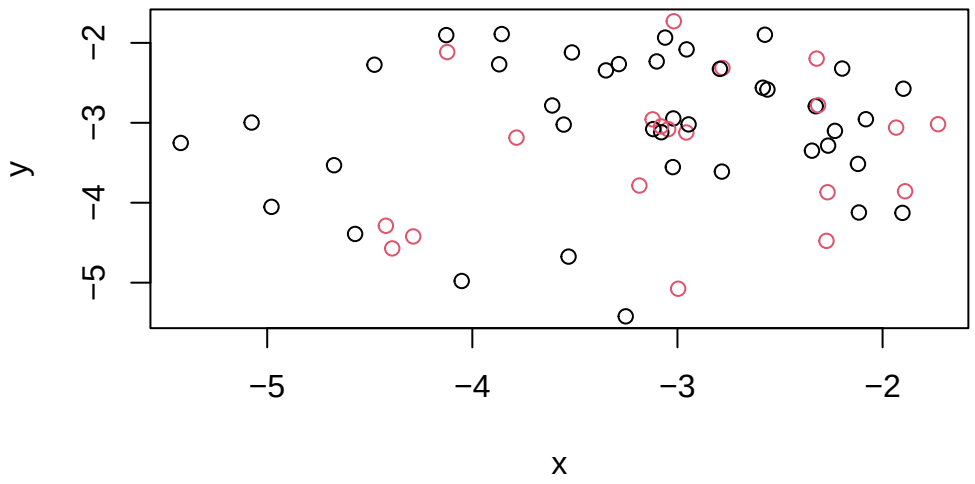
```
 [1] 1 2 1 1 1 1 1 1 2 1 1 1 1 1 2 1 1 2 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1 1 1
[39] 1 1 2 2 1 1 2 1 1 1 1 1 2 1 1 1 1 1 1 1 2 1
```

```
mycols<-c(1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2)

plot(x,col=1,2)
```




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

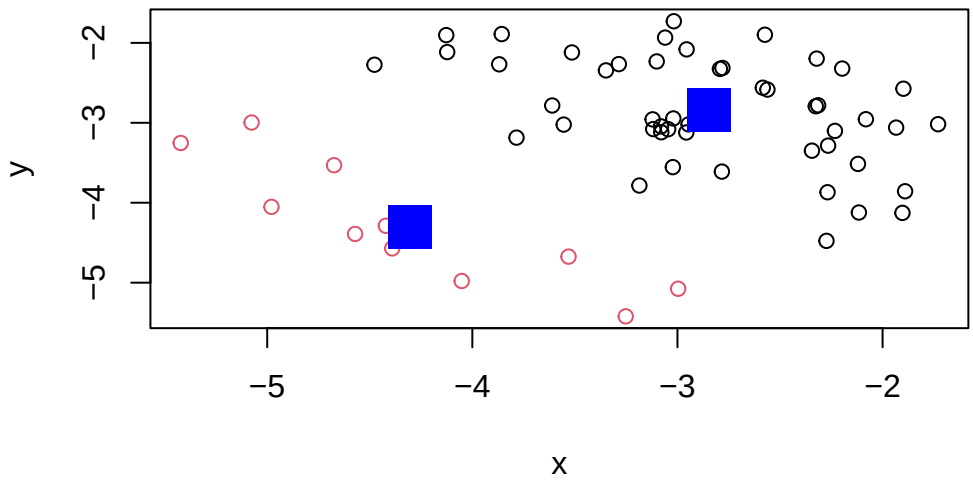



```
#Can color by name eg. col="red"
#Can color by number eg.col=1
#mycols<-c(1,2)

plot(x,col=mycols)
```

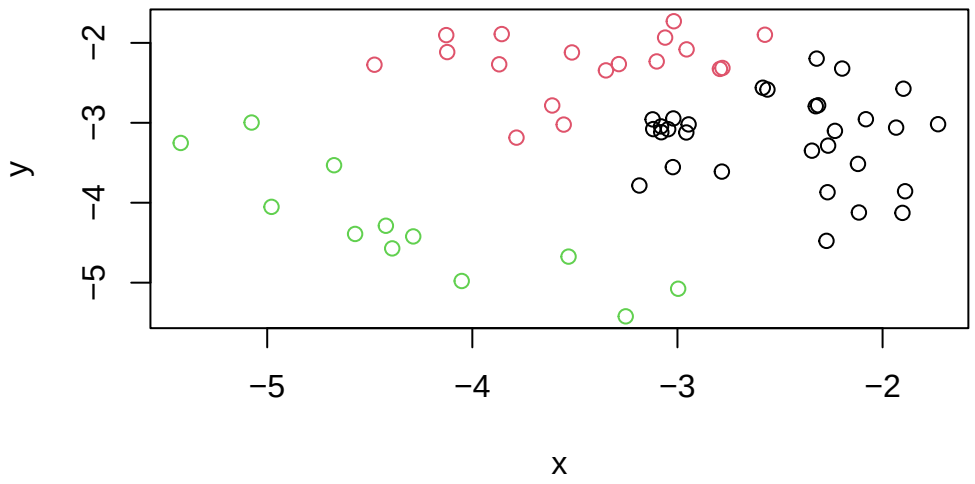
Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points.

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



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

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



Hierarchical Cluster

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

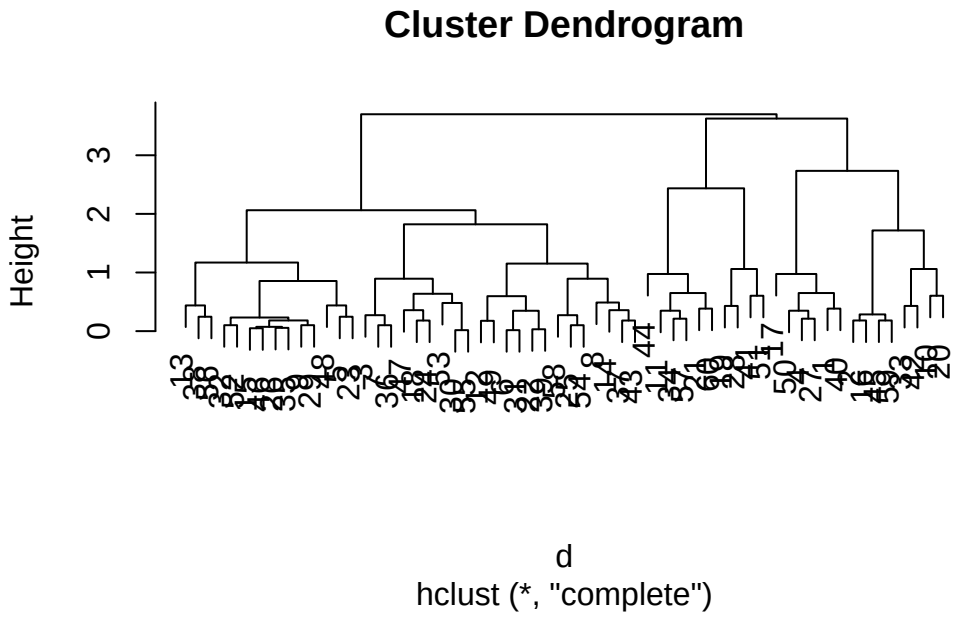
We will use the 'dist()' function to start with:

```
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 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
```

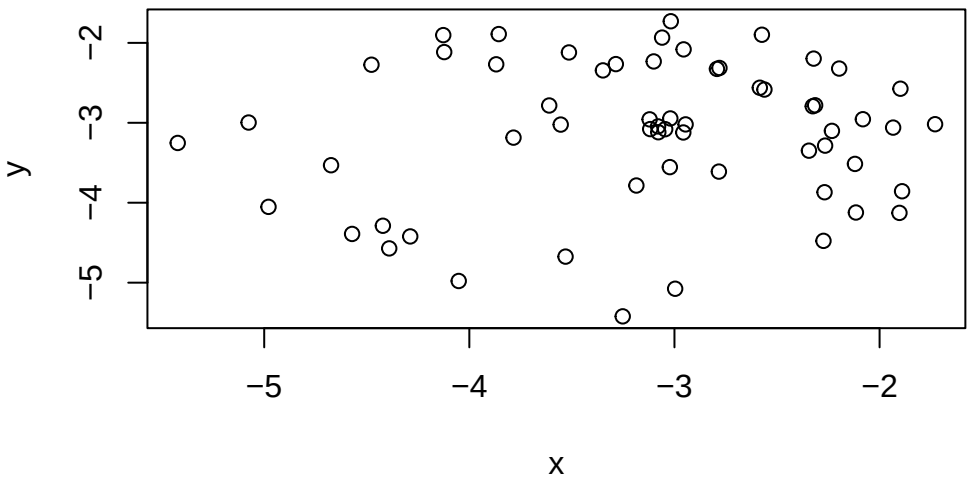
You can also tell 'cutree()' to cut where it yields "k" group.

```
cutree(hc, k=2)
```

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

```
#Cut into 2 groups. This is equivalent to the previous cutree(hc, h=8)
```

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



Principal Component Analysis (PCA)

Principal components are new low dimensional axis or surfaces closest to the observations, and reveals the most important structure The data have msx vairance along PC1 (then PC2) which make the first few PCs useful for visualizing our data

```
# Class 7 Lab

# 1. PCA of UK food data

::: {.cell}

```{.r .cell-code}
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names =1)
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
Fresh_potatoes	720	874	566	1033
Fresh_Veg	253	265	171	143
Other_Veg	488	570	418	355
Processed_potatoes	198	203	220	187
Processed_Veg	360	365	337	334
Fresh_fruit	1102	1137	957	674
Cereals	1472	1582	1462	1494
Beverages	57	73	53	47
Soft_drinks	1374	1256	1572	1506
Alcoholic_drinks	375	475	458	135
Confectionery	54	64	62	41

:::

Use this method instead o fx <-x[,1] head(x)

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

```
dim(x)
```

```
[1] 17  4
```

Q2. Which approach to solving the row-names problem mentioned above do you prefer and why? Is one apprach more robost than another under certian circumstance?

Answer: The apprach more accuarate is the x <- read.csv(url, row.names =1) function instead of the function fx <-x[,1] head(x)

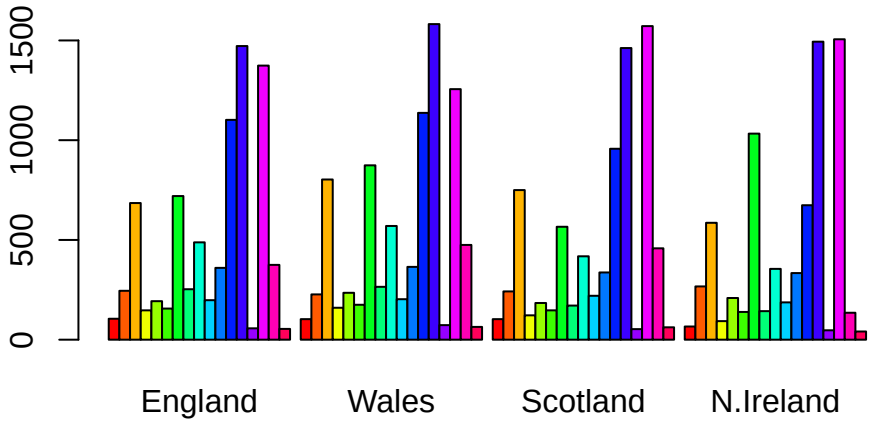
This is because the second function will lead to lost in row or columns when returned, while the first function returns all rows and columns and is more robost. So I would choose the first function for solving the row-names problem.

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

Answer: The beside function is changed to false in the barplot() code.

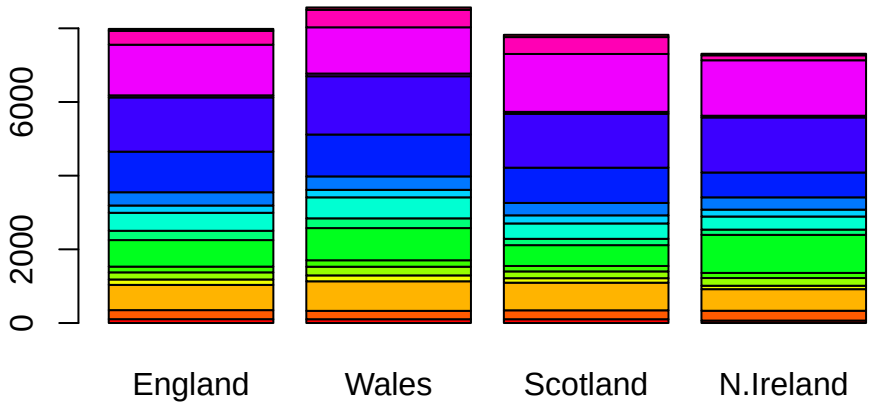
Spotting major differences and trends

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



#New Plot

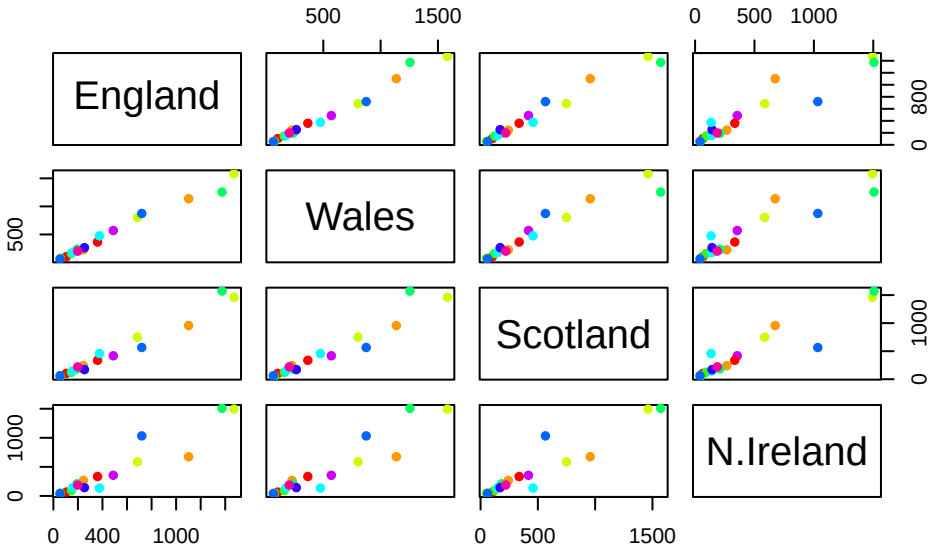
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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?

Answer: The figure shows each colored dot as a food category, and if a point lies on the diagonal line, it means that the countries in the x and y axis consumed the same amount for that food category. Points above the diagonal line means that the countries of the y axis consumes more food of a particular category.

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



The main PCA function in base R is called the 'prcomp()', and it expects the transpose of our data.

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

Answer: The main differences between N Irland and other countries of UK is that there is more orange dot consumed in England compared to that of Irland.

PCA to the rescue

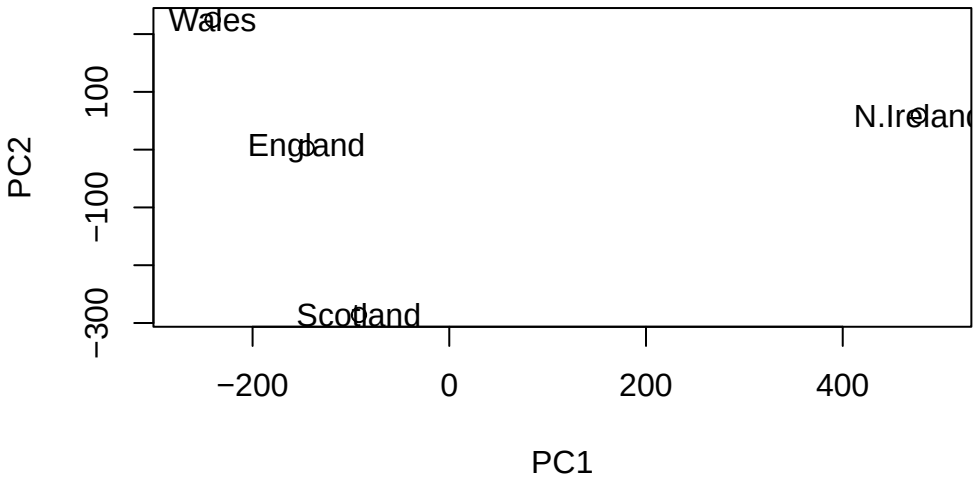
```
#Use the prcomp() PCA function
pca <- prcomp( t(x) )
summary(pca)
```

```
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
```

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

Answers:

```
# Plot PC1 vs PC2
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.

Answers:

The main PCA function in base R is called 'prcomp()' it expects the transpose of our data

```
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
```

```r
attributes(pca) # want x
```

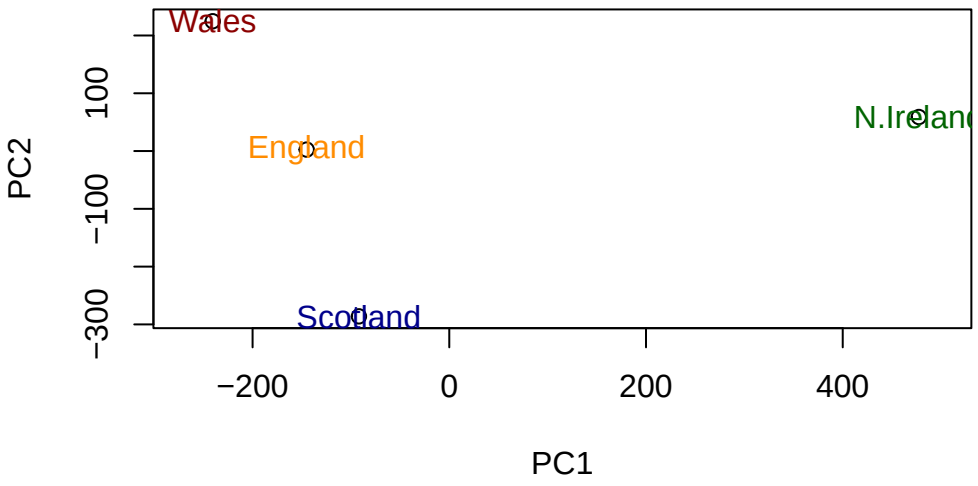
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

```r
pca$x # 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

```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), col=c("darkorange","darkred","darkblue","darkgreen"
```


Wales
England
Scotland
N.Ireland
PC2
100
−100
−300
−200
0
200
400
PC1