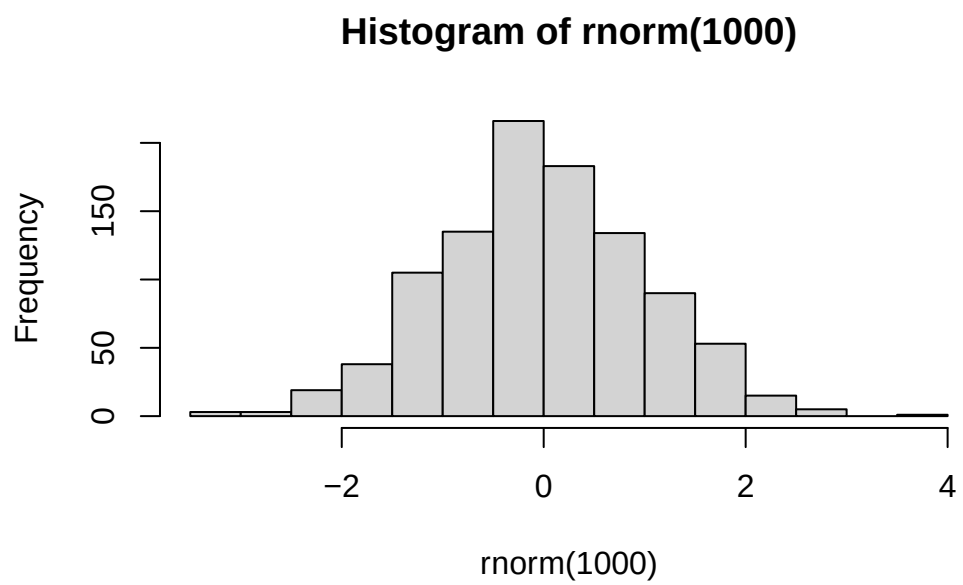
class 07 machine learning 1

Rachel Field (PID: A69042948

Today we will begin our exploration of some "classical" machine learning approaches. We will start with clustering.

Let's first make up some data to cluster where we know what the answer should be.

```
hist(rnorm(1000))
```




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

```
 [1] -3.4426506 -2.8051473 -1.2996608 -2.0808818 -3.7502202 -4.3352679
 [7] -5.1700773 -4.8275850 -3.7379117 -0.7592693 -2.8670495 -2.5121396
[13] -2.6445897 -0.8325340 -2.8978924 -3.6362068 -1.1006999 -4.0853093
```

```
[19] -2.9981698 -4.4083087 -3.7180388 -4.8326718 -3.6947553 -3.0101828
[25] -3.3064712 -2.3421018 -2.1331718 -2.9366260 -2.7165633 -4.1931131
```

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

```
 [1] 3.050287 4.702214 4.501628 1.468922 2.818786 3.317176 3.843900 4.244827
 [9] 3.601093 3.011034 1.988467 1.889176 3.151336 3.623194 3.669894 2.859638
[17] 2.167083 2.433095 4.069449 4.293882 3.612259 2.549594 3.928871 1.580968
[25] 2.356740 2.293461 3.702157 3.807237 2.819602 4.629174
```

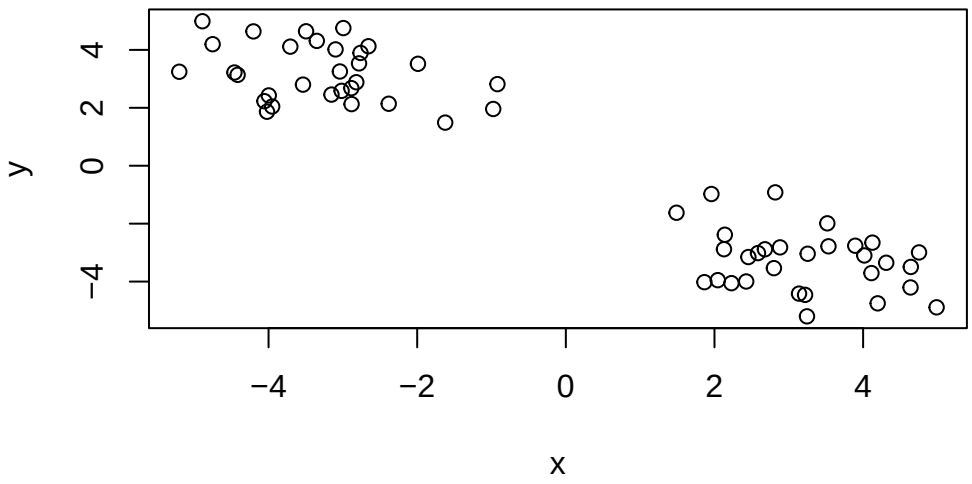
#makes normal distribution centered around a value #concatenate into a vector #print out side by side with cbind

```
x<-c(rnorm(30, mean=-3), rnorm(30, mean=3))
y<- rev(x)

x<- cbind(x,y)
head(x)
```

```
             x        y
[1,] -3.098649 4.016081
[2,] -3.707258 4.111991
[3,] -4.416490 3.137995
[4,] -2.993774 4.752525
[5,] -1.989411 3.518679
[6,] -5.201212 3.244759
```

A wee peak at x with `plot()`

```
plot(x)
```



Then main function in "base" R for K-means clustering called `kmeans()`

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

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

Cluster means:
          x         y
1  3.202836 -3.267422
2 -3.267422  3.202836

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] 60.22172 60.22172
 (between_SS / total_SS =  91.2 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
```

```
[6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How big are the clusters? (i.e. their size?)

```
k$size
```

```
[1] 30 30
```

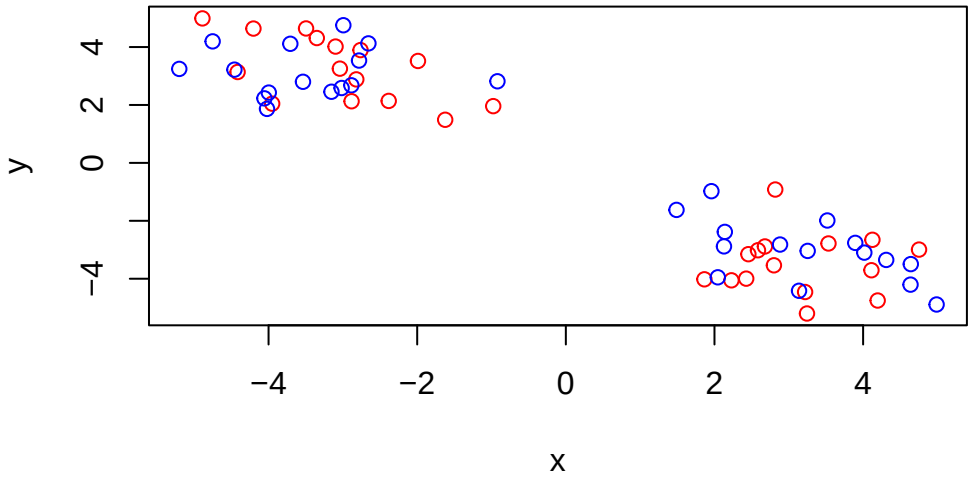
Q. What clusters do my data points reside in?

```
k$cluster
```

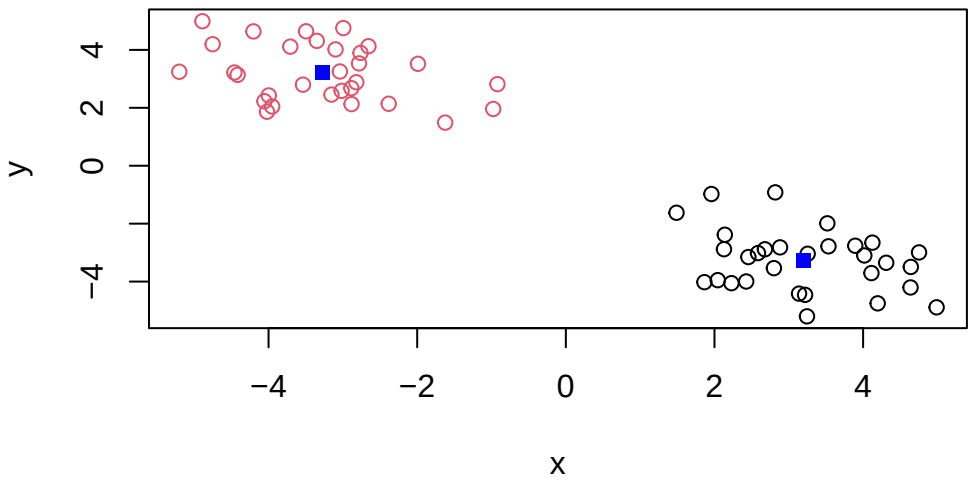
```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Q. Make a plot of our daa colored by clustered assignment. i.e. Make a result figure

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




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



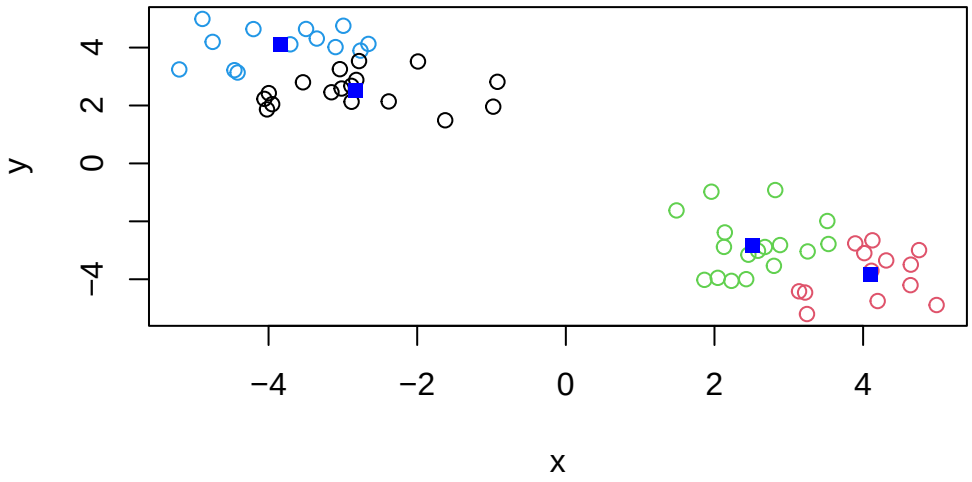
Q. Cluster with k-means into 4 clusters and plot your results as above.

```
k2 <- kmeans(x, centers = 4)
k2
```

```
K-means clustering with 4 clusters of sizes 17, 13, 17, 13

Cluster means:
          x         y
1 -2.825630  2.518081
2  4.098286 -3.845150
3  2.518081 -2.825630
4 -3.845150  4.098286

Clustering vector:
 [1] 4 4 4 4 1 4 4 4 1 1 1 4 1 1 1 1 4 1 4 1 4 1 4 1 1 1 1 1 1 4 2 3 3 3 3 3 3 2
[39] 3 2 3 2 3 2 3 3 3 3 2 3 3 3 2 2 2 3 2 2 2 2

Within cluster sum of squares by cluster:
[1] 20.99610 13.17364 20.99610 13.17364
 (between_SS / total_SS =  95.0 %)

Available components:
```

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

```
plot(x, col = k2$cluster)
points(k2$centers, col = "blue", pch = 15)
```



Q. Run kmeans with centers (i.e. values of k) equal to 1 to 6

```
k$tot.withinss
```

```
[1] 120.4434
```

Brute force

```
k.1<-kmeans(x, centers=1)$tot.withinss
k.2<-kmeans(x, centers=2)$tot.withinss
k.3<-kmeans(x, centers=3)$tot.withinss
k.4<-kmeans(x, centers=4)$tot.withinss
k.5<-kmeans(x, centers=5)$tot.withinss
k.6<-kmeans(x, centers=6)$tot.withinss

ans <- c(k.1,k.2,k.3,k.4,k.5,k.6)
```

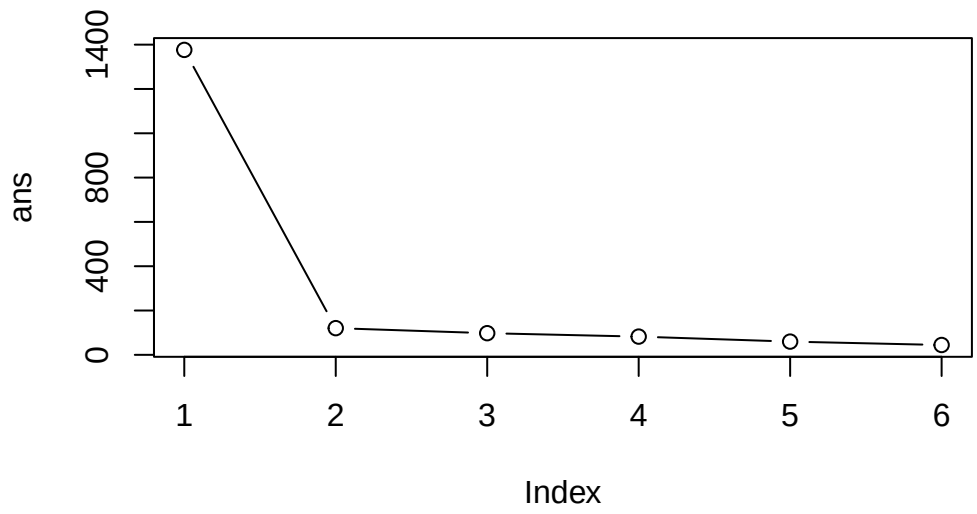
Or a for loop #make something called ans and it's empty so I can put stuff there #for i assuming the values of 1:6, assuming the value of 1 first

```
ans <- NULL
for(i in 1:6){
  ans <- c(ans, kmeans(x, centers= i)$tot.withinss)
}

ans
```

```
[1] 1376.37063  120.44343   97.75962   82.34543   59.72976   44.38370
```

Make a "scree-plot"

```
plot(ans, typ = "b")
```



#values decrease because as you add more clusters, the size of the clusters decreases. At value of 2, you get the. most bang for your book

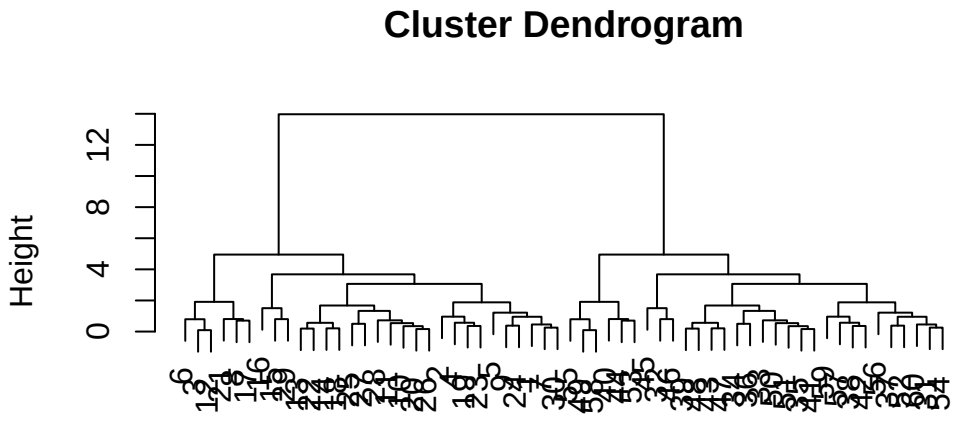
##Hierarchical Clustering The main function in "base" R for this is called `hclust()`

#kmeans works with euclidean distance, this can work with other things like sequence similarity, structure similarity, etc

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

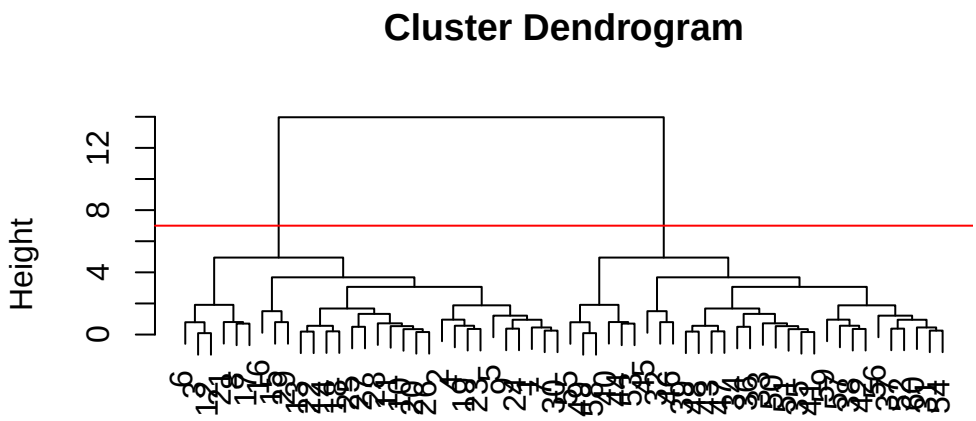
```
Call:
hclust(d = d)

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

```
plot(hc)
```



#low numbers on the left of the plot, high numbers on the right #tree shaped cluster used to interpret hierarchical clustering models

```
plot(hc)
abline(h=7, col="red")
```

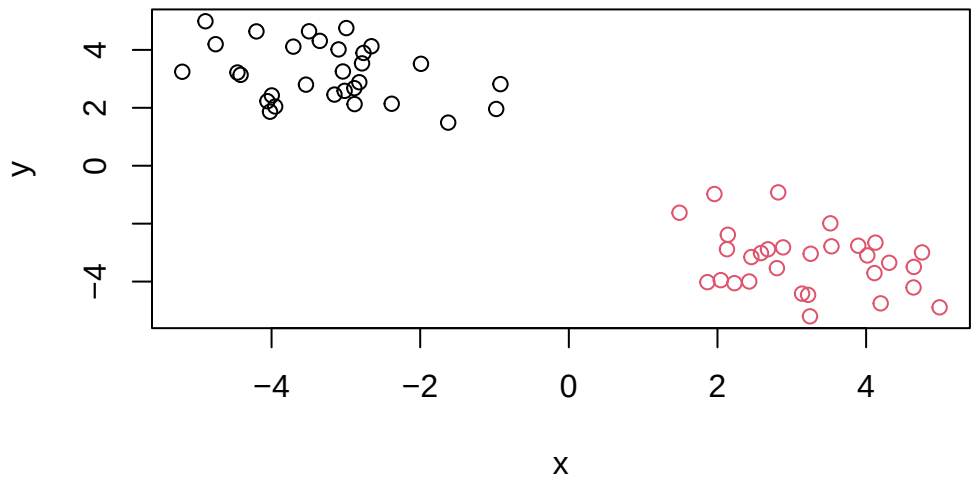


To obtain clusters from our `hclust` result object "hc" we "**cut** the tree to yield different sub branches, for this we use the `cutree()` function

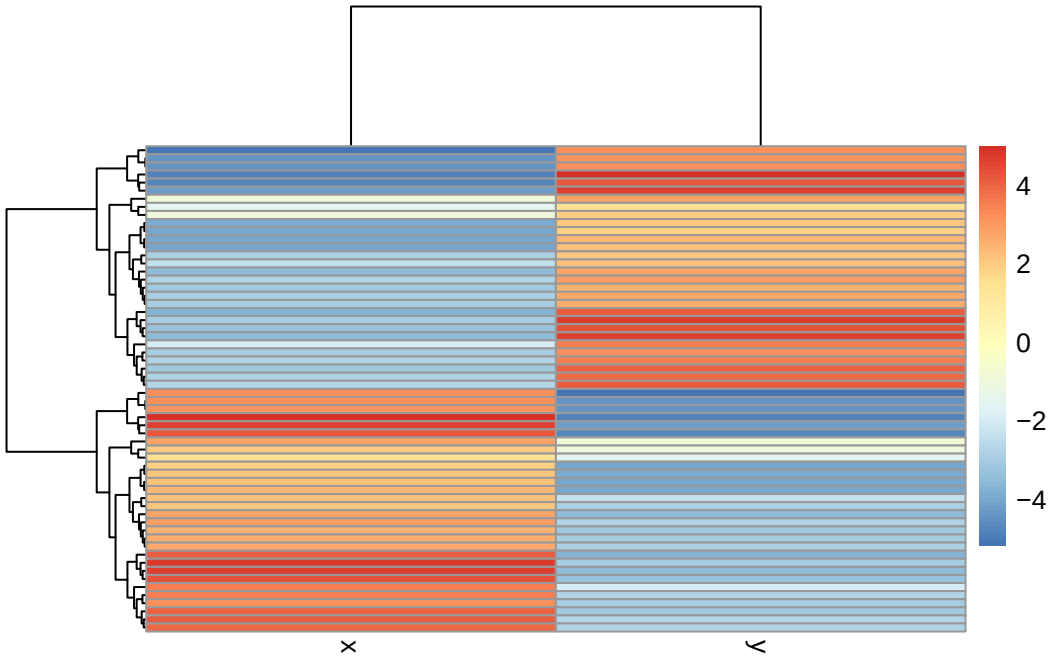
```
grps <- cutree(hc, h=7)
grps
```

```
 [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 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
```

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




```
library(pheatmap)
pheatmap(x)
```



##Principal Component Analysis (PCA)

PCA projects the features onto the principal components. The motivation is to reduce the features dimensionality while only losing a small amount of information.

The first principal component (PC) follows a best fit through the data points.

Principal components are new low dimesional axis (or surfaces) closest to the observations. These are better than the x and y axis because they describe more of the variance. The data have maximum variance along PC1 (then PC2 etc) which makes the first few PCs useful for visualizing our data and as a basis for further analysis

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

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

17 rows, 5 columns. dim(x)

```
dim(x)
```

```
[1] 17  5
```

```
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
```

```
tail(x)
```

```
                   X England Wales Scotland N.Ireland
12      Fresh_fruit    1102  1137      957       674
13          Cereals    1472  1582     1462      1494
14        Beverages      57    73       53        47
15      Soft_drinks    1374  1256     1572      1506
16 Alcoholic_drinks     375   475      458       135
17    Confectionery      54    64       62        41
```

```
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)
```

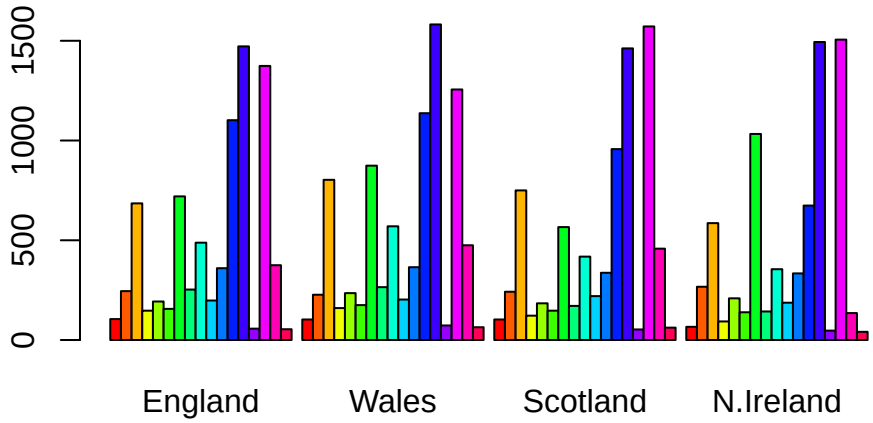
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars             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 prefer the second approach because when I run the first approach multiple times, it erases England and then other columns.

#Spotting major differences and trends

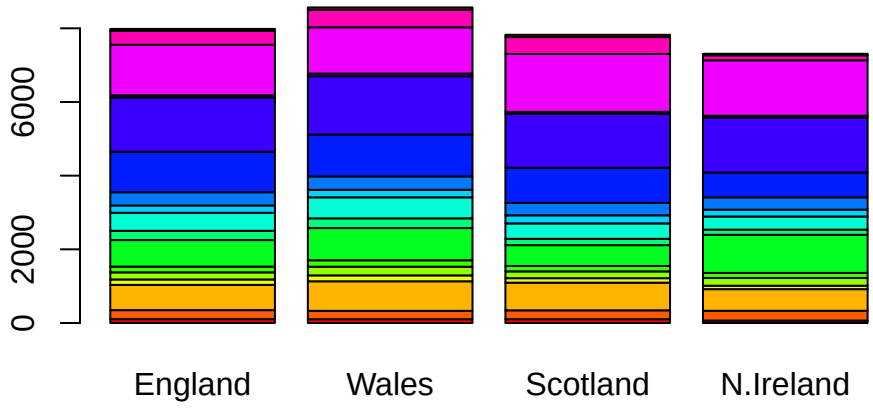
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

beside=F

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



#this is a bad plot, for ex the blue parts are not on a common axis

##Side-note: Using ggplot and the need for "tidy" data

Using the ggplot2 package for these relatively simple figures is somewhat more convoluted than using "base" R. This is due to the ggplot requirement for so-called "long" data rather than the conventional "wide" data format that we currently have. We will talk more about "long" format data a little later in the course when we introduce the dplyr package.

Data organized with one row per Food (17 observation) and multiple columns for Country (4 different measurements across a row):

To convert this to "long" format we want one row per measurement - maximizes rows (17x4=68), minimizes columns (with a singe Consumption measurement value per Country). We will do tidying with the pivot_longer() function from the tidyr package:

```
library(tidyr)

# Convert data to long format for ggplot with `pivot_longer()`
x_long <- x |>
          tibble::rownames_to_column("Food") |>
          pivot_longer(cols = -Food,
                       names_to = "Country",
                       values_to = "Consumption")
```

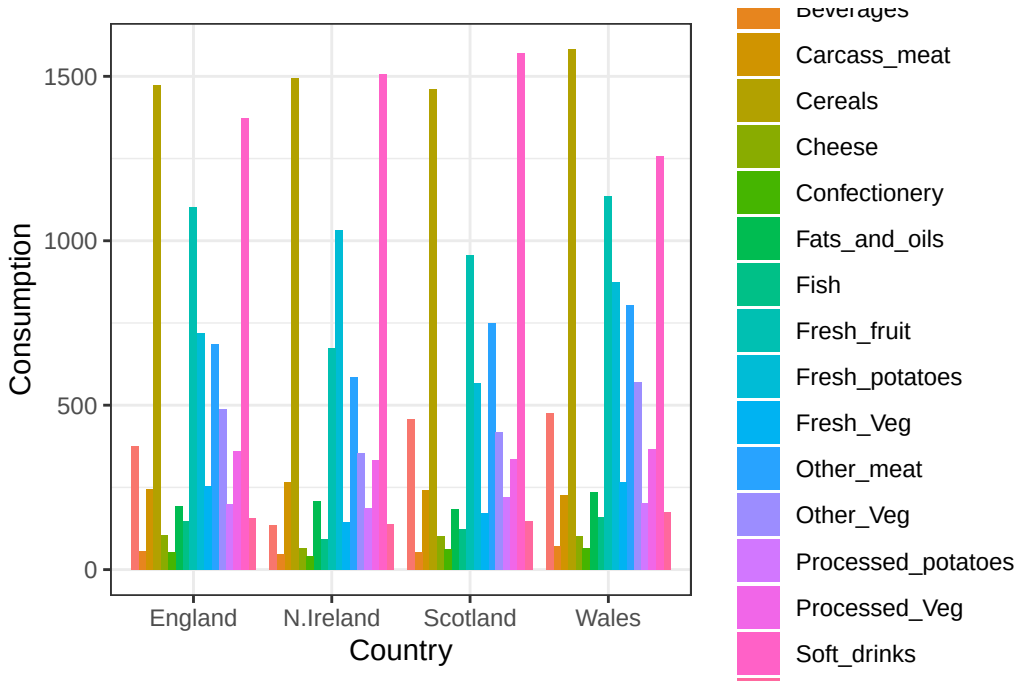
```r
dim(x_long)
```

```
[1] 68  3
```

```r
head(x_long)
```

```
# A tibble: 6 x 3
  Food            Country   Consumption
  <chr>           <chr>           <int>
1 "Cheese"        England           105
2 "Cheese"        Wales             103
3 "Cheese"        Scotland          103
4 "Cheese"        N.Ireland          66
5 "Carcass_meat " England           245
6 "Carcass_meat " Wales             227
```

```r
library(ggplot2)

ggplot(x_long) +
  aes(x = Country, y = Consumption, fill = Food) +
  geom_col(position = "dodge") +
  theme_bw()
```



Q4: Changing what optional argument in the above ggplot() code results in a stacked barplot figure?

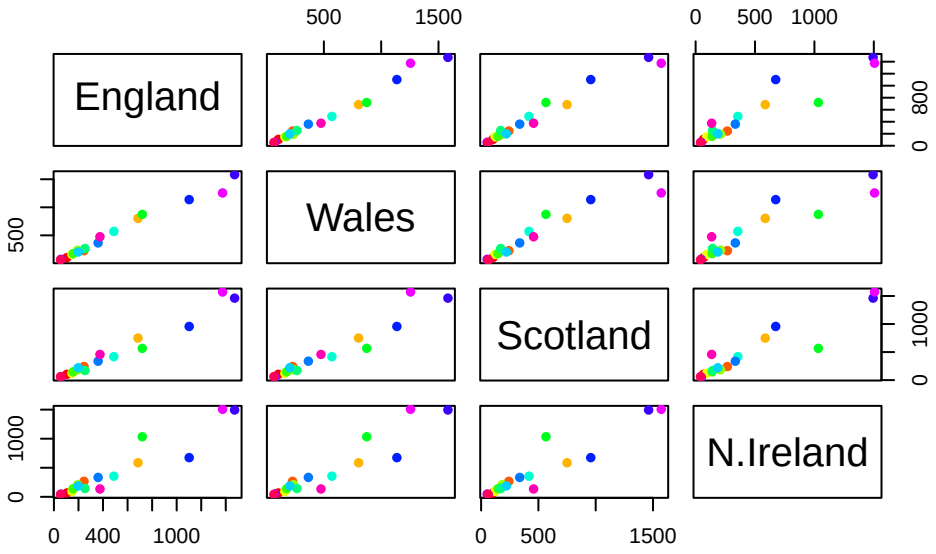
beside=F

##Pairs plots and heatmaps Pairs plots, also known as scatterplot matrices, are collections of bivariate scatter plots used to visualize relationships between multiple variables simultaneously. For small datasets, pairs plots are a particularly valuable exploratory analysis technique.

Q5: We can use the pairs() function to generate all pairwise plots for our countries. 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?

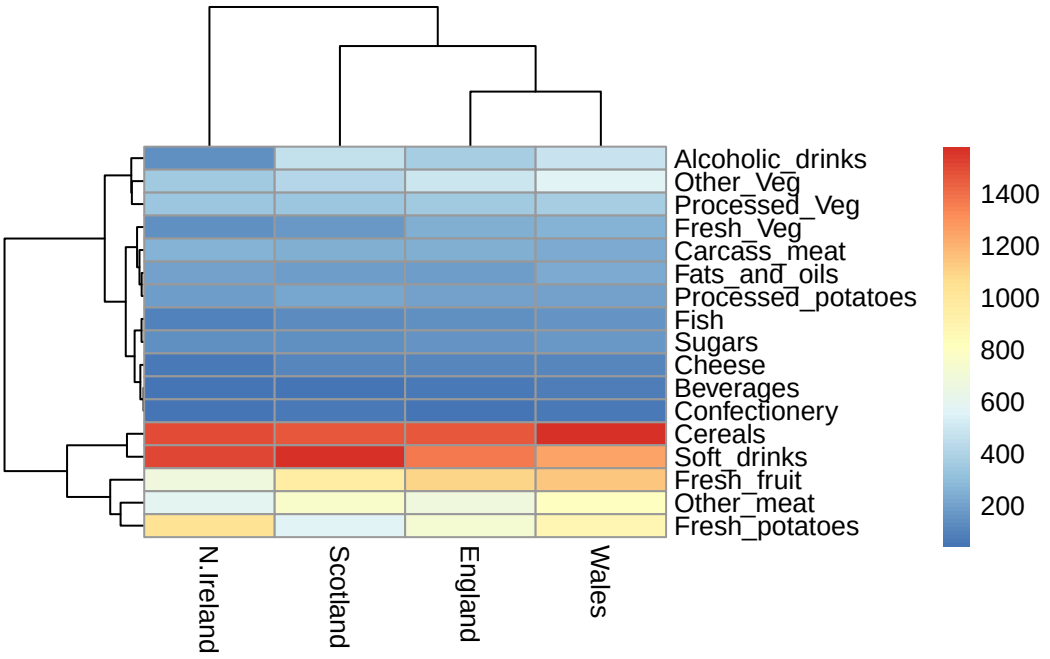
It is an outlier point, that has a higher value in one group (here, country) than the other.

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



#we can see outliers (think, we could do this for gene expression)

```
library(pheatmap)

pheatmap( as.matrix(x) )
```



In this heatmap, darker colors represent higher values (consumption in grams), and the dendrogram (tree diagram) on the left shows how foods cluster together, while the top dendrogram shows how countries cluster together.

> Q6. Based on the pairs and heatmap figures, which countries cluster together and what does this suggest about their food consumption patterns? Can you easily tell what the main differences between N. Ireland and the other countries of the UK in terms of this data-set?

It seems that England, and Wales are more similar, but it's hard to tell what is going on in the dataset.

It's not easy to notice patterns in these previous methods.

#PCA to the rescue

Helpful notes from lab sheet: Traditionally, we would use a series of pairwise plots (i.e. bivariate scatter plots) and analyse these to try and determine any relationships between variables, however the number of such plots required for such a task is clearly too large even for this small dataset. Therefore, for large data sets, this is not feasible or fun.

PCA generalizes this idea and allows us to perform such an analysis simultaneously, for many variables. In our example here, we have 17 dimensional data for 4 countries. We can thus 'imagine' plotting the 4 coordinates representing the 4 countries in 17 dimensional space. If there is any correlation between the observations (the countries), this will be observed in the 17 dimensional space by the correlated points being clustered close together, though of course

since we cannot visualize such a space, we are not able to see such clustering directly (see the lecture slides for a clear description and example of this).

To perform PCA in R there are actually lots of functions to chose from and many packages offer slick PCA implementations and useful graphing approaches. However here we will stick to the base R prcomp() function.

As we noted in the lecture portion of class, prcomp() expects the observations to be rows and the variables to be columns therefore we need to first transpose our data.frame matrix with the t() transpose function.

… The main function in "base" R for PCA is called `prcomp()`

As we want to do PCA on the food data for the different countries we will want the foods in the columns.

```
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
```

PC1 captures 67% of the variance.

Our result object is called `pca` and it has a `$x` component that we will look at first. (It is a list object so we need $ to get components of this)

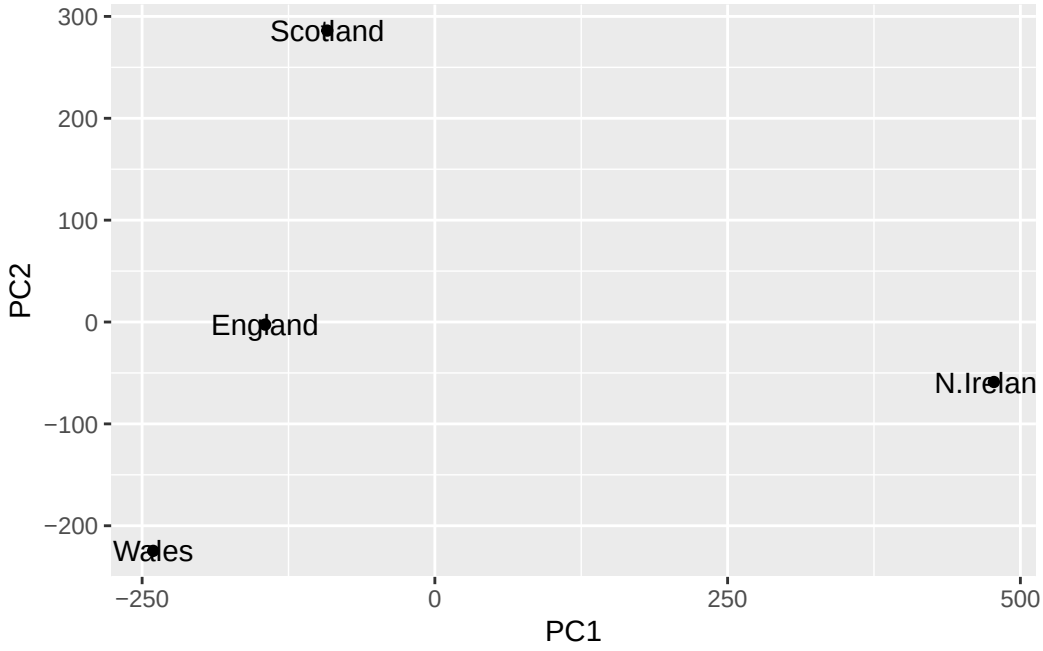
```
pca$x
```

```
                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
```

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

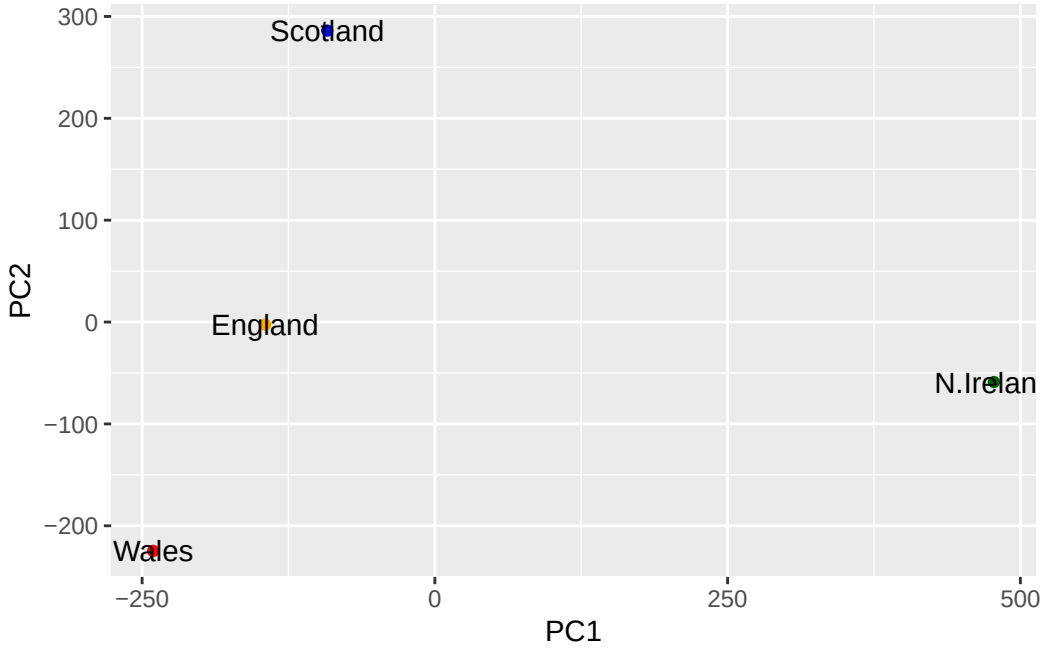
Score plot:

```
ggplot(pca$x, aes(PC1, PC2, label = rownames(pca$x))) +
  geom_point() +
  geom_text()
```



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.

```
library(ggplot2)

cols<-c("orange", "red", "blue", "darkgreen")

ggplot(pca$x, aes(PC1, PC2, label = rownames(pca$x))) +
  geom_point(col=cols) +
  geom_text()
```



Another major result out of PCA is the so-called "variable loadings" or how `$rotation` that tells us how the original variables (foods) contribute to PCs (i.e. our the new axis)

```
pca$rotation
```

	PC1	PC2	PC3	PC4
Cheese	-0.056955380	0.016012850	0.02394295	-0.409382587
Carcass_meat	0.047927628	0.013915823	0.06367111	0.729481922
Other_meat	-0.258916658	-0.015331138	-0.55384854	0.331001134
Fish	-0.084414983	-0.050754947	0.03906481	0.022375878
Fats_and_oils	-0.005193623	-0.095388656	-0.12522257	0.034512161
Sugars	-0.037620983	-0.043021699	-0.03605745	0.024943337
Fresh_potatoes	0.401402060	-0.715017078	-0.20668248	0.021396007
Fresh_Veg	-0.151849942	-0.144900268	0.21382237	0.001606882
Other_Veg	-0.243593729	-0.225450923	-0.05332841	0.031153231
Processed_potatoes	-0.026886233	0.042850761	-0.07364902	-0.017379680
Processed_Veg	-0.036488269	-0.045451802	0.05289191	0.021250980
Fresh_fruit	-0.632640898	-0.177740743	0.40012865	0.227657348
Cereals	-0.047702858	-0.212599678	-0.35884921	0.100043319
Beverages	-0.026187756	-0.030560542	-0.04135860	-0.018382072
Soft_drinks	0.232244140	0.555124311	-0.16942648	0.222319484
Alcoholic_drinks	-0.463968168	0.113536523	-0.49858320	-0.273126013
Confectionery	-0.029650201	0.005949921	-0.05232164	0.001890737

```
ggplot(pca$rotation) +
  aes(PC1, rownames(pca$rotation)) +
  geom_col()
```