Class 7: Machine Learning 1

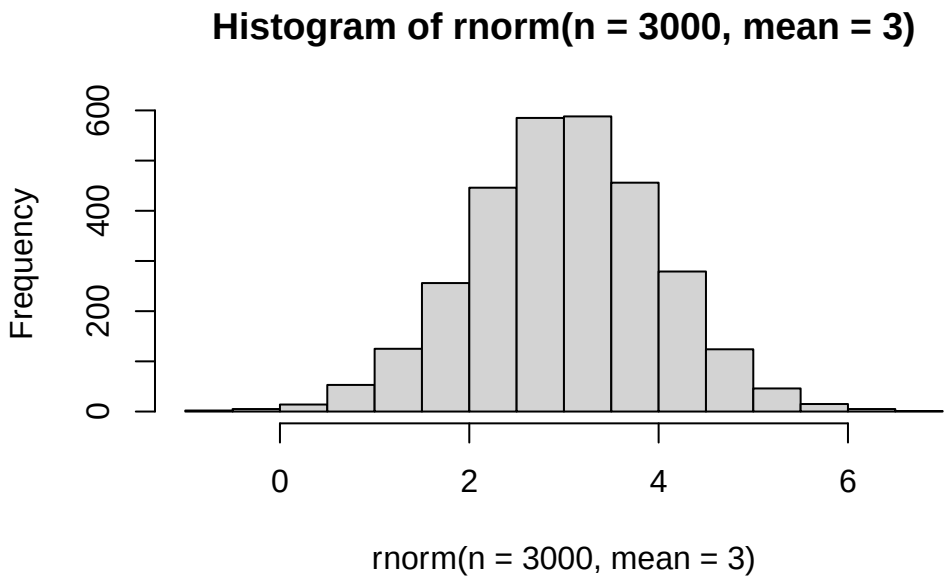
Lily Huynh (PID:A16929651)

Today we will explore unsupervised machine learning methods including clustering and dimensionallity reduction methods.

Let's start by making up some data (where we know there are clear groups) that we can use to test out different clustering methods.

We can use the `rnorm()` function to help us here

```
hist(rnorm(n=3000, mean=3))
```

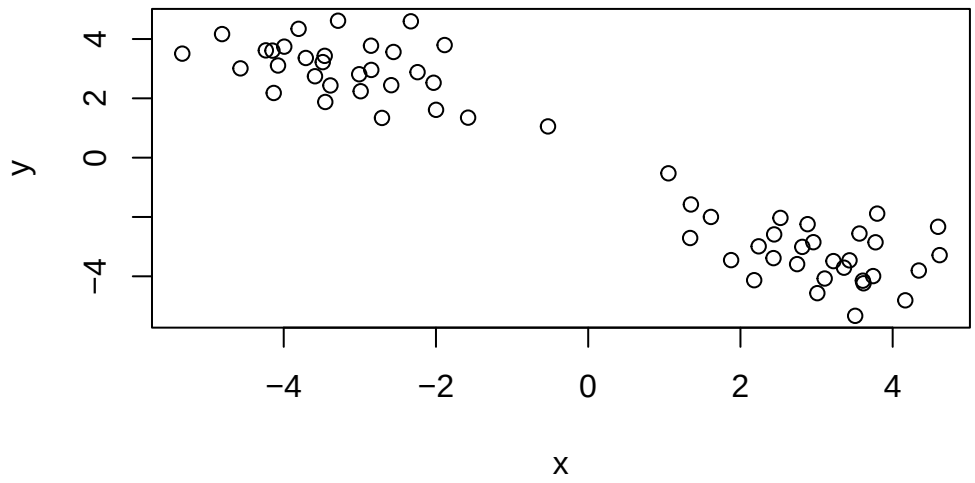


Make data with two "clusters"

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

z <- cbind(x=x, y=rev(x))
head(z)
```

```
             x        y
[1,] -4.147819 3.608259
[2,] -2.558164 3.563186
[3,] -2.587396 2.443149
[4,] -5.332697 3.507163
[5,] -3.993055 3.742930
[6,] -2.853371 3.774544
```

```r
plot(z)
```



How big is `z`

```r
nrow(z)
```

```
[1] 60
```

```
ncol(z)
```

```
[1] 2
```

K-means clustering

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

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

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

Cluster means:
          x         y
1  2.997167 -3.187425
2 -3.187425  2.997167

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] 58.09925 58.09925
 (between_SS / total_SS =  90.8 %)

Available components:

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

```
attributes(k)
```

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

$class
[1] "kmeans"
```

Q. How many points lie in each cluster?

```
k$size
```

```
[1] 30 30
```

Q. What component of our results tells us about the cluster membership (i.e. which point likes in which clusters)?

```
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
```

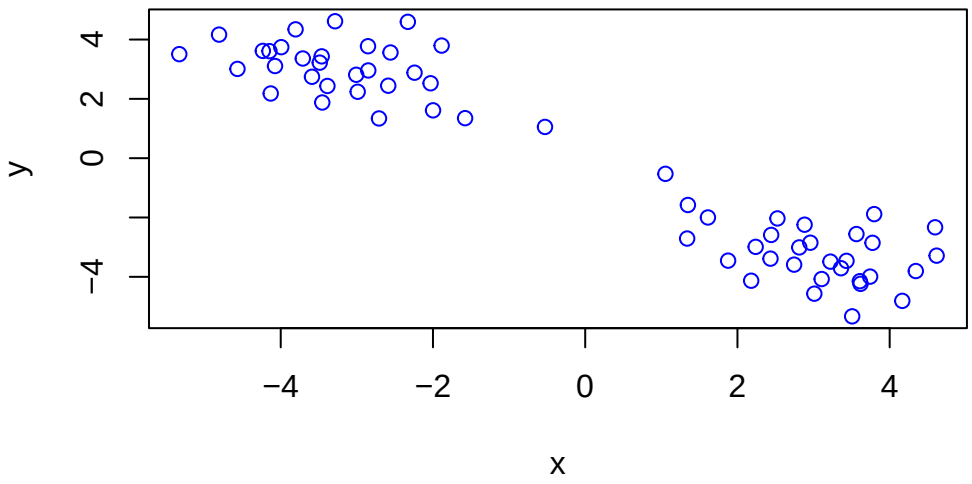
Q. Center of each cluster?

```
k$centers
```

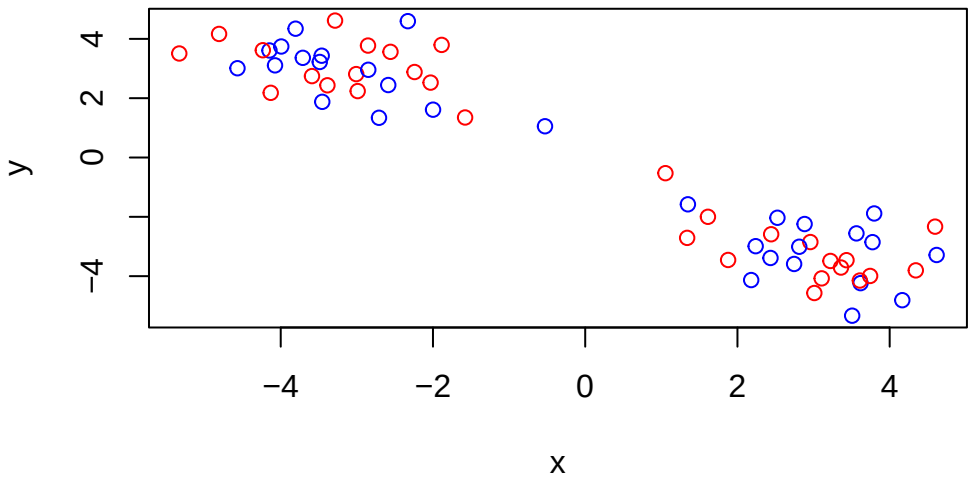
```
          x         y
1  2.997167 -3.187425
2 -3.187425  2.997167
```

Q. Put this result info together to make a little "base R" plot of our clustering result. Also add the cluster center points to this plot.

```
plot(z, col="blue")
```

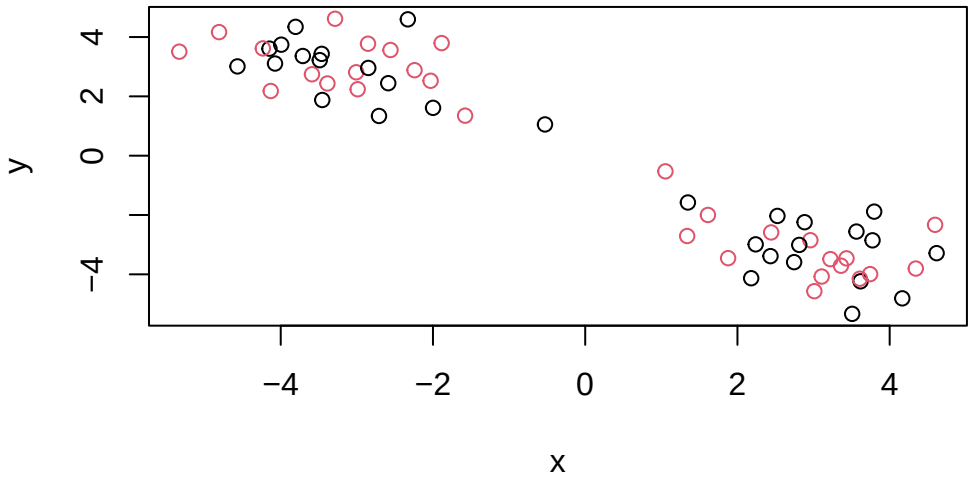



```
plot(z, col=c("blue", "red"))
```



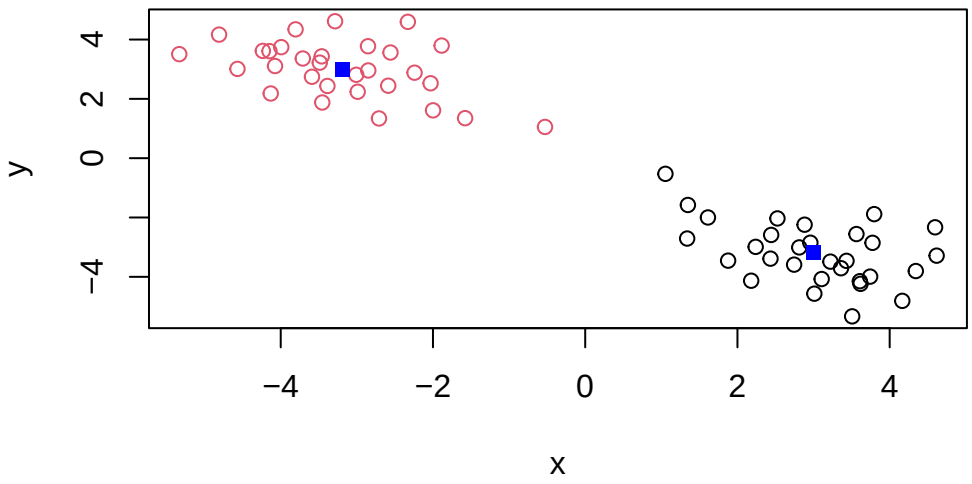
You can color by number

```
plot(z, col=c(1,2))
```



Plot colored by cluster membership:

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



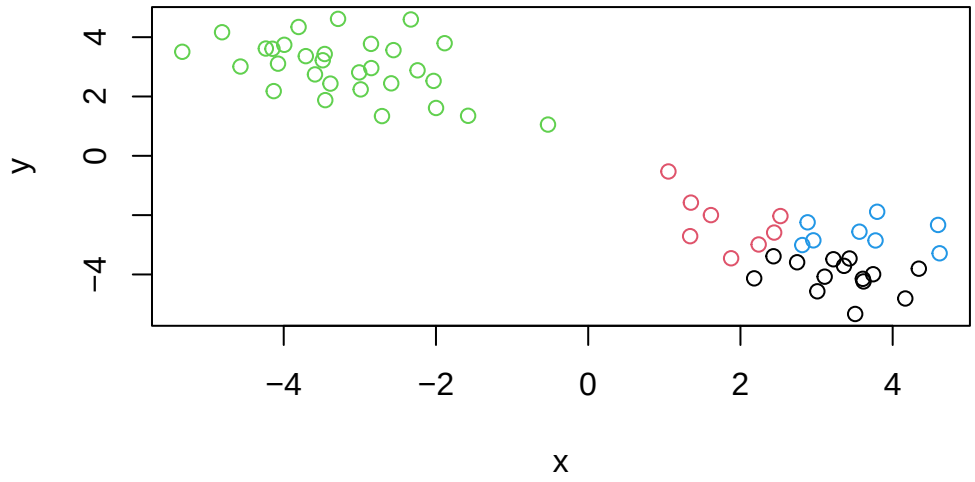
Q. Run kmeans on our input `z` and define 4 clusters, making the same results visualization plot as above (plot of z colored by cluster membership).

```
z
```

```
              x          y
 [1,] -4.1478192  3.6082594
 [2,] -2.5581643  3.5631858
 [3,] -2.5873962  2.4431488
 [4,] -5.3326974  3.5071626
 [5,] -3.9930548  3.7429299
 [6,] -2.8533711  3.7745439
 [7,] -3.8049439  4.3432741
 [8,] -4.1313432  2.1806506
 [9,] -3.4542220  1.8774245
[10,] -4.2351081  3.6172099
[11,] -1.9988317  1.6123787
[12,] -2.9890519  2.2394677
[13,] -3.7097882  3.3603682
[14,] -1.8861247  3.7964682
[15,] -0.5282306  1.0538448
[16,] -3.0082710  2.8122447
[17,] -4.0748208  3.1066399
```

```
[18,] -4.8100586  4.1656701
[19,] -4.5683454  3.0100129
[20,] -3.3867404  2.4342335
[21,] -2.7090978  1.3395143
[22,] -1.5776861  1.3491390
[23,] -2.8497230  2.9578108
[24,] -3.5893101  2.7445084
[25,] -3.4616632  3.4325825
[26,] -2.2420124  2.8821388
[27,] -2.3300596  4.5964264
[28,] -3.2863633  4.6168078
[29,] -3.4873008  3.2215261
[30,] -2.0311578  2.5254396
[31,]  2.5254396 -2.0311578
[32,]  3.2215261 -3.4873008
[33,]  4.6168078 -3.2863633
[34,]  4.5964264 -2.3300596
[35,]  2.8821388 -2.2420124
[36,]  3.4325825 -3.4616632
[37,]  2.7445084 -3.5893101
[38,]  2.9578108 -2.8497230
[39,]  1.3491390 -1.5776861
[40,]  1.3395143 -2.7090978
[41,]  2.4342335 -3.3867404
[42,]  3.0100129 -4.5683454
[43,]  4.1656701 -4.8100586
[44,]  3.1066399 -4.0748208
[45,]  2.8122447 -3.0082710
[46,]  1.0538448 -0.5282306
[47,]  3.7964682 -1.8861247
[48,]  3.3603682 -3.7097882
[49,]  2.2394677 -2.9890519
[50,]  1.6123787 -1.9988317
[51,]  3.6172099 -4.2351081
[52,]  1.8774245 -3.4542220
[53,]  2.1806506 -4.1313432
[54,]  4.3432741 -3.8049439
[55,]  3.7745439 -2.8533711
[56,]  3.7429299 -3.9930548
[57,]  3.5071626 -5.3326974
[58,]  2.4431488 -2.5873962
[59,]  3.5631858 -2.5581643
[60,]  3.6082594 -4.1478192
```

```r
k4 <- kmeans(z, centers = 4)
plot(z, col=k4$cluster)
```



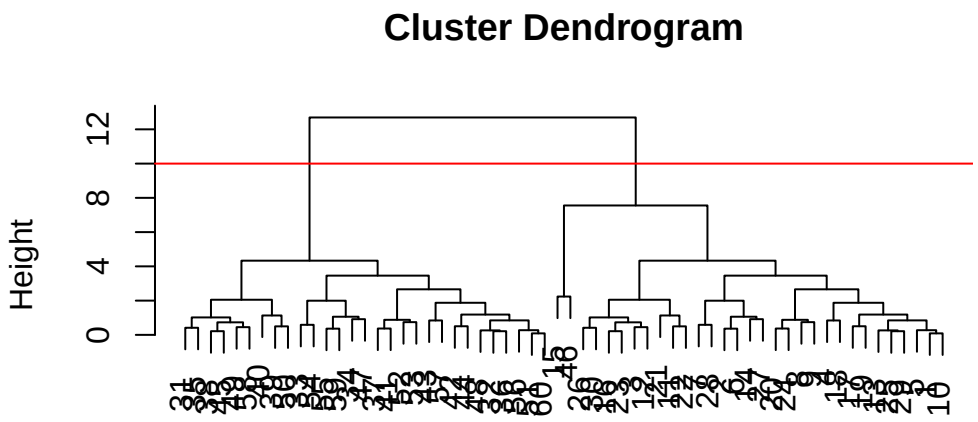
Hierarchical Clustering

The main function in base R for this called `hclust()` it will take as input a distance matrix (key point is that you can't just five your "raw" data as input - you have to first calculate a distance matrix from your data).

```r
d <- dist(z)
hc <- hclust(d)
hc
```

```
Call:
hclust(d = d)

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

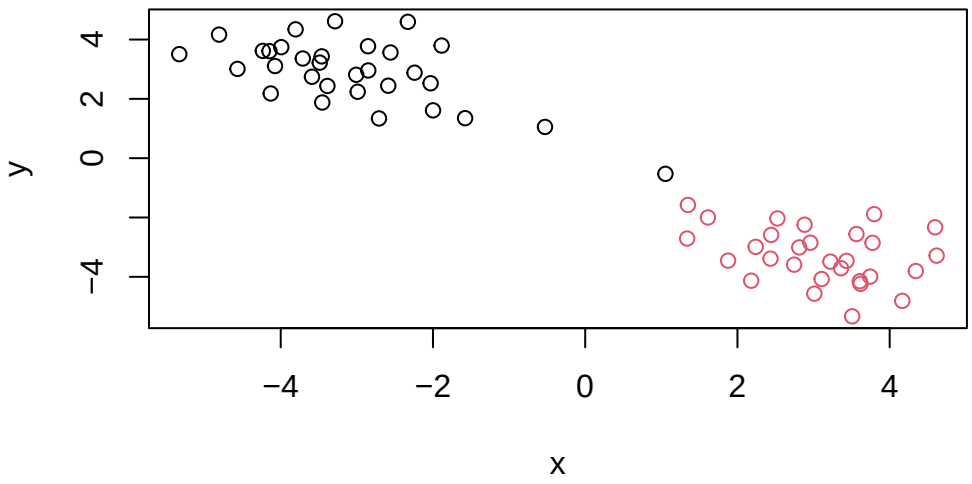
```
plot(hc)
abline(h=10, col="red")
```



Once I inspect the "tree", I can "cut" the tree to yield my groupings or clusters. The function to do this is called `cutree()`

```
groups <- cutree(hc, h=10)
```

```
plot(z, col=groups)
```



Hands on with Principal Component Analysis (PCA)

Let's examine some silly 17-dimensional data detailing food consumption in the UK (England, Scotland, Wales, and N. Ireland). Are these countries eating habits different or similar and if so how?

Data import

```
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
```

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

```
nrow(x)
```

```
[1] 17
```

```
ncol(x)
```

```
[1] 4
```

```
dim(x)
```

```
[1] 17  4
```

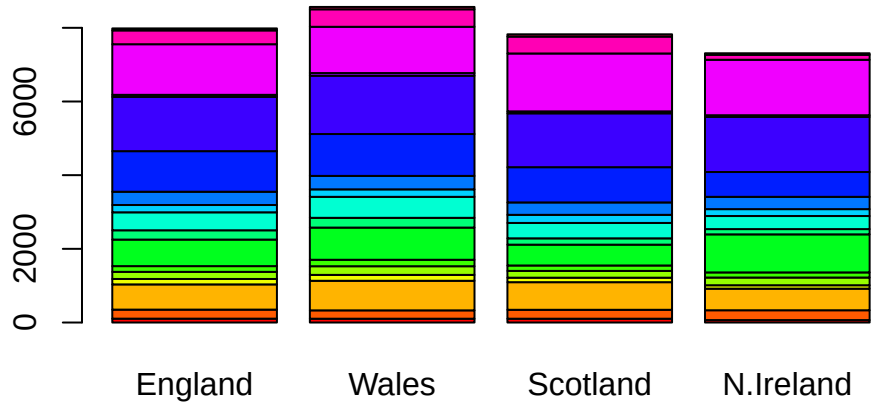
Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

I prefer doing nrow() and ncol() because I don't have to remember if dim() gives me the number of rows first or number of columns first. I think nrow() and ncol() is more robust because we can chose a specific value without needing the $ sign.

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

Changing the argument besides=T to beside=F changes it into a stacked graph.

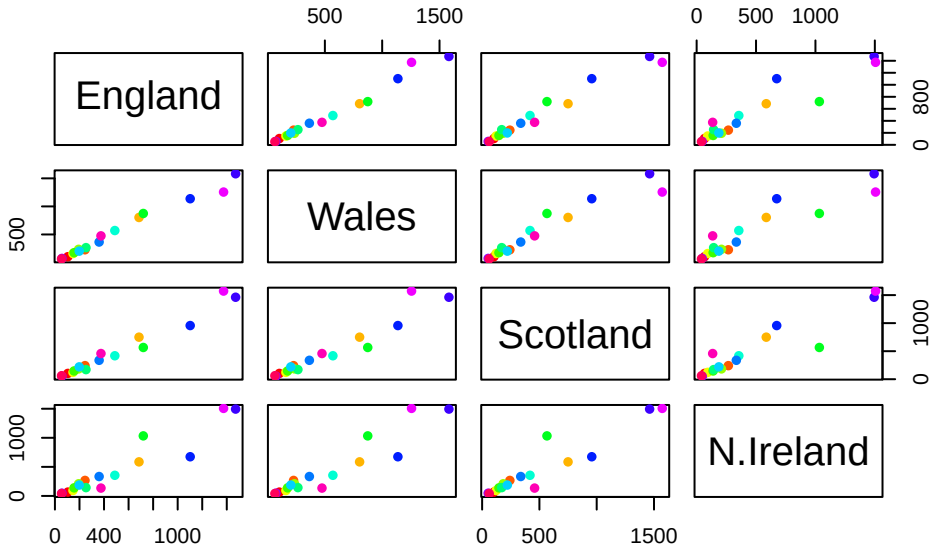
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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 pairwise plots are showing different plots with different x and y axis. For example, N.Ireland can be on the x axis or the y axis. A given data point that lies on the diagonal that has a positive correlation means that the compared countries on the x and y axis are similar.

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



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

The main different between N. Ireland and the other countries of the UK is the amount of Fresh_potatoes, Alcoholic_drinks, and Fresh_fruit they eat.

Looking at these types of "pairwise plots" can be helpful but it does not scale well and kind of sucks! There must be a better way…

PCA to the rescue!

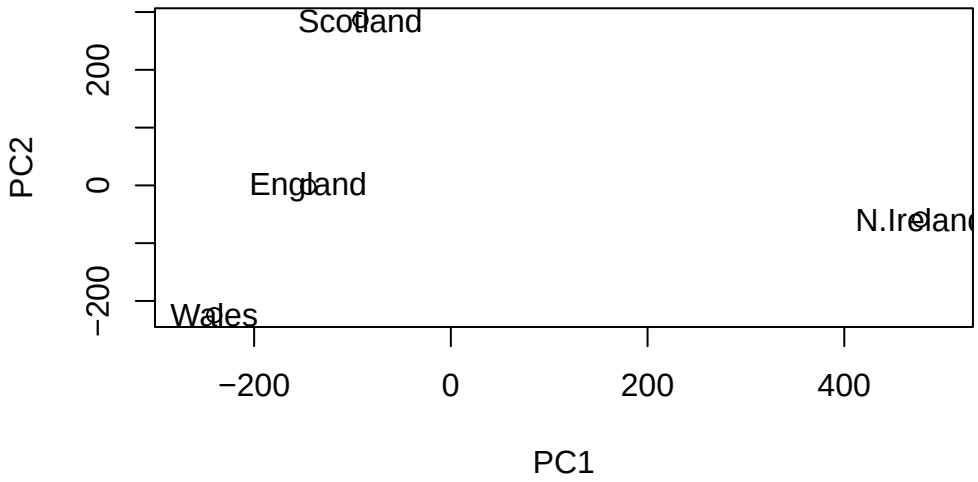
The main function for PCA in base R is called `prcomp()`. This function wants the transpose of our input data - i.e. the important food in as columns and the countries as rows.

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

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 3.176e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

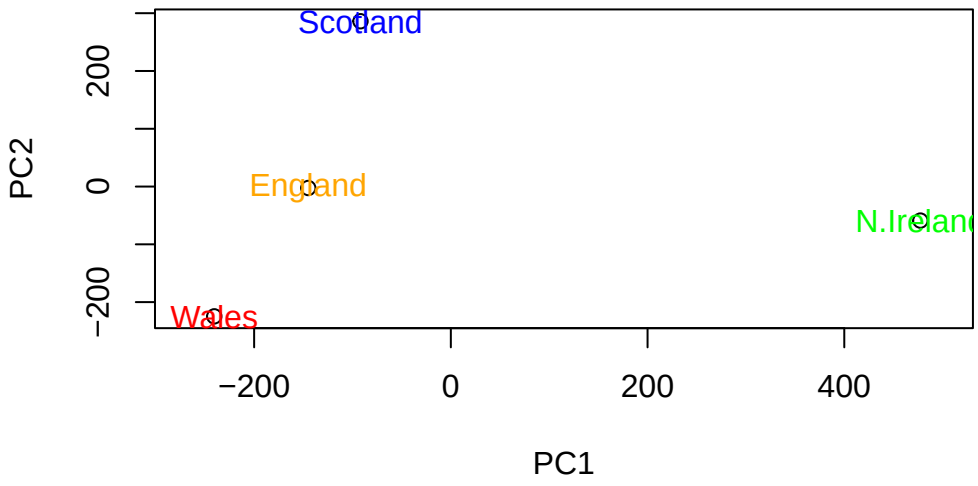
Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```
# 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.

```
# 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),col= c("orange", "red", "blue", "green"))
```



Let's see what is in our PCA result object `pca`

```
attributes(pca)
```

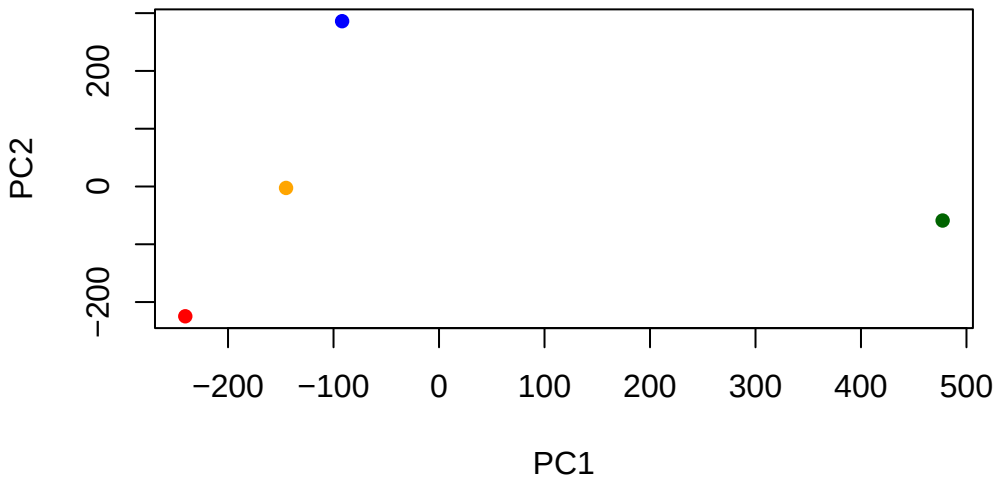
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$class
[1] "prcomp"
```

The `pca$x` result object is where we will focus on first, as this details how the countries are related to each other in terms of our new "axis" (aka. "PCs", "eigenvectors", etc.)

```
head(pca$x)
```

```
                PC1         PC2        PC3           PC4
England   -144.99315   -2.532999 105.768945 -4.894696e-14
Wales     -240.52915 -224.646925 -56.475555  5.700024e-13
Scotland   -91.86934  286.081786 -44.415495 -7.460785e-13
N.Ireland  477.39164  -58.901862  -4.877895  2.321303e-13
```

```
plot(pca$x[,1], pca$x[,2], pch=16,
     col= c("orange", "red", "blue", "darkgreen"),
     xlab="PC1", ylab="PC2")
```



We can look at the so-called PC "loadings" result object to see how the original foods contribute to our new PCs (i.e. how the original variables contribute to our new PC variables).

```
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
```

Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

Ans. The two prominantely food groups are fresh potatoes and soft drinks. PC2 mainly tells us that Fresh_potates is in the negative side of the plot and therefore pushes the countries down the plot. It also tells us that Soft_drinks is on the positive side of the plot and therefore pushes the countries to the top of the plot.

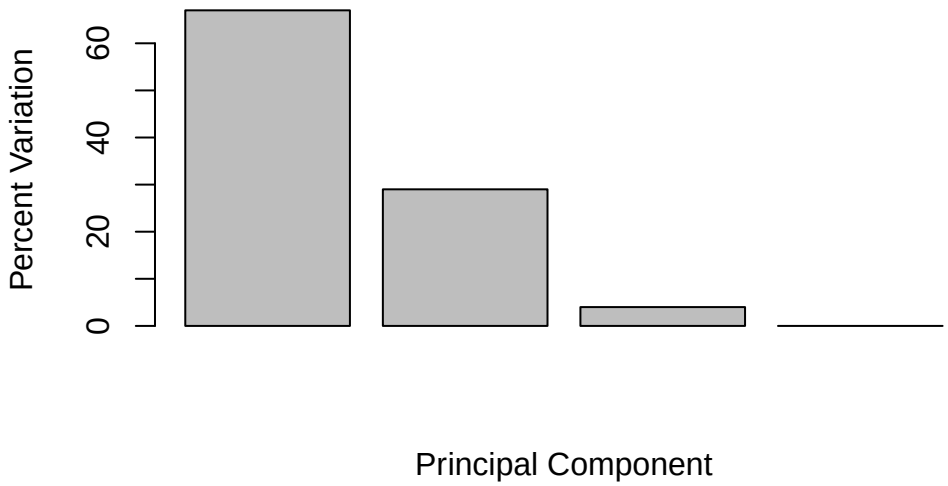
```
v <- round( pca$sdev^2/sum(pca$sdev^2) * 100 )
v
```

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

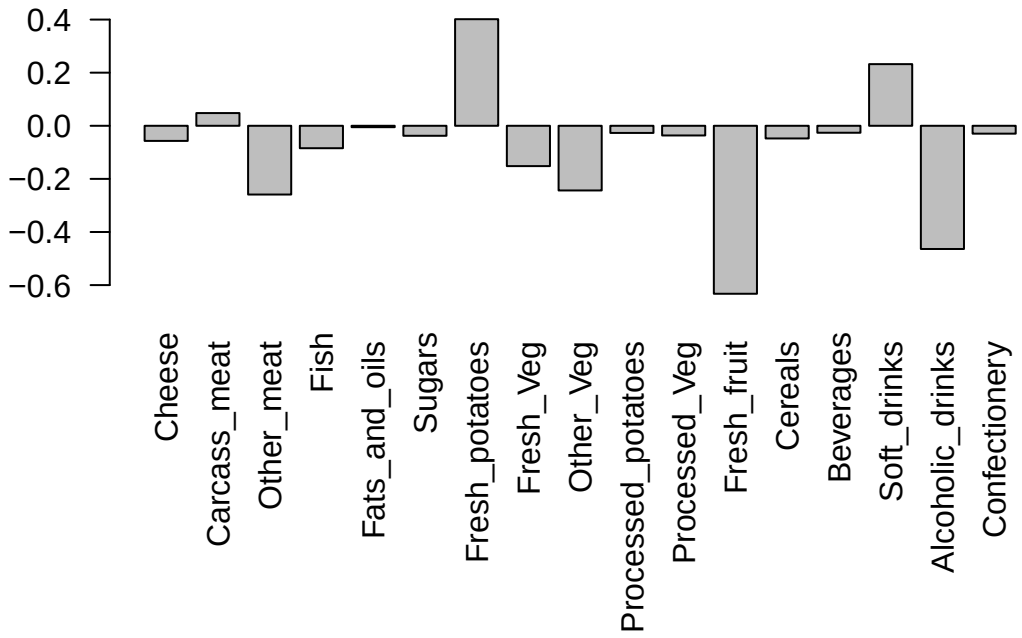
```
z <- summary(pca)
z$importance
```

```
                             PC1       PC2      PC3          PC4
Standard deviation     324.15019 212.74780 73.87622 3.175833e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

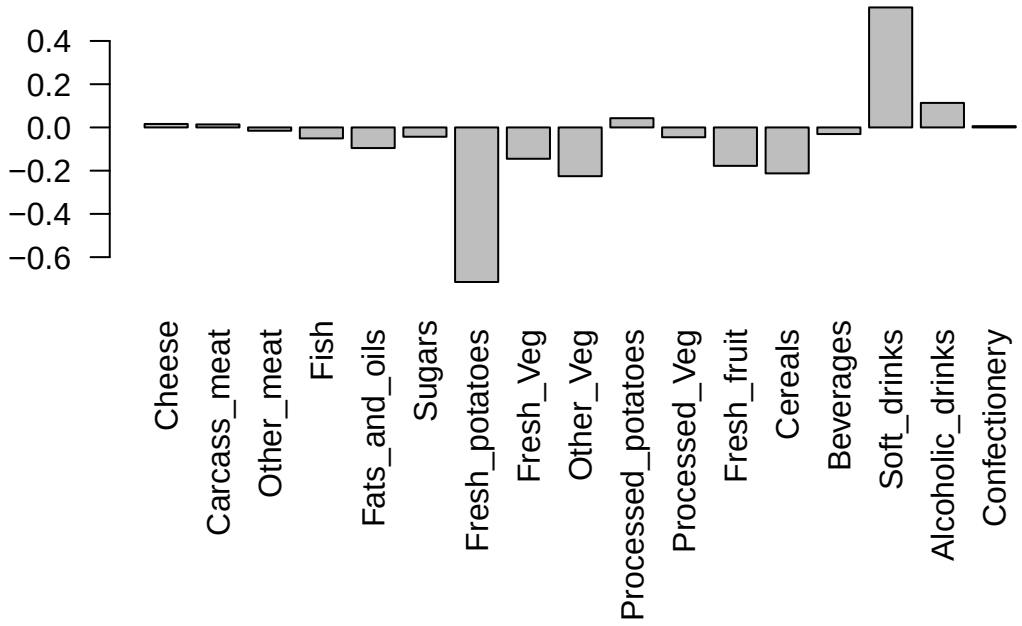
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



```r
## Lets focus on PC1 as it accounts for > 90% of variance
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```




```r
## Lets focus on PC2
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



Q10: How many genes and samples are in this data set?

Ans. There are 100 genes and 10 samples in this data set.

```
url2 <- "https://tinyurl.com/expression-CSV"
rna.data <- read.csv(url2, row.names=1)
head(rna.data)
```

```
       wt1 wt2  wt3  wt4 wt5 ko1 ko2 ko3 ko4 ko5
gene1  439 458  408  429 420  90  88  86  90  93
gene2  219 200  204  210 187 427 423 434 433 426
gene3 1006 989 1030 1017 973 252 237 238 226 210
gene4  783 792  829  856 760 849 856 835 885 894
gene5  181 249  204  244 225 277 305 272 270 279
gene6  460 502  491  491 493 612 594 577 618 638
```

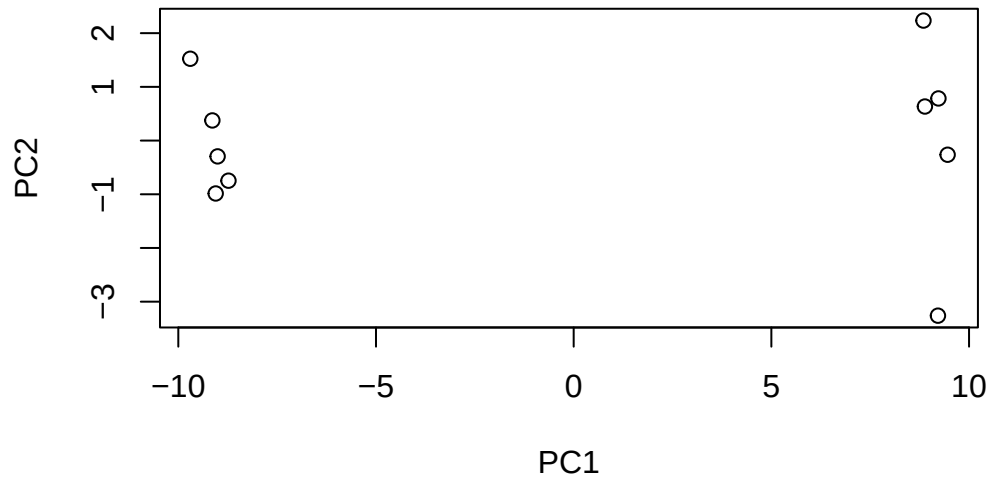
```
# The genes are rows
nrow(rna.data)
```

```
[1] 100
```

```r
# The samples are columns
ncol(rna.data)
```

```
[1] 10
```

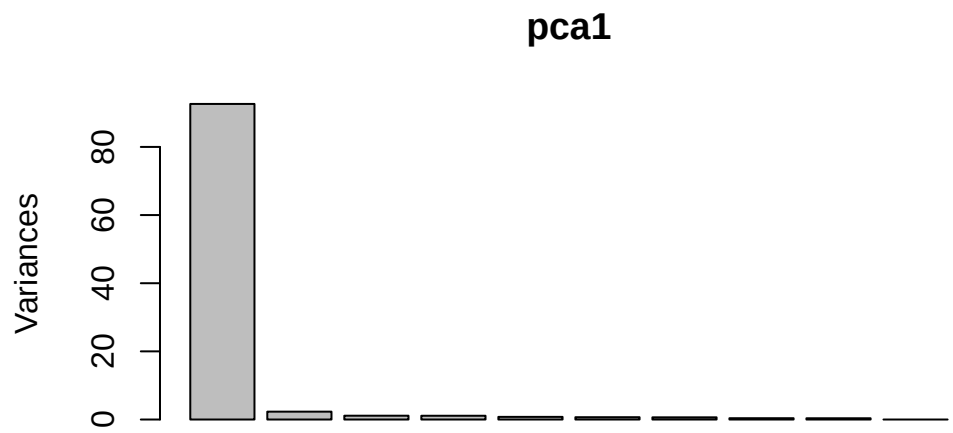
```r
pca1 <- prcomp(t(rna.data),, scale=TRUE)

plot(pca1$x[,1], pca1$x[,2], xlab="PC1", ylab="PC2")
```




```r
summary(pca1)
```

```
Importance of components:
                          PC1    PC2     PC3     PC4     PC5     PC6     PC7
Standard deviation     9.6237 1.5198 1.05787 1.05203 0.88062 0.82545 0.80111
Proportion of Variance 0.9262 0.0231 0.01119 0.01107 0.00775 0.00681 0.00642
Cumulative Proportion  0.9262 0.9493 0.96045 0.97152 0.97928 0.98609 0.99251
                           PC8     PC9      PC10
Standard deviation     0.62065 0.60342 3.457e-15
Proportion of Variance 0.00385 0.00364 0.000e+00
Cumulative Proportion  0.99636 1.00000 1.000e+00
```

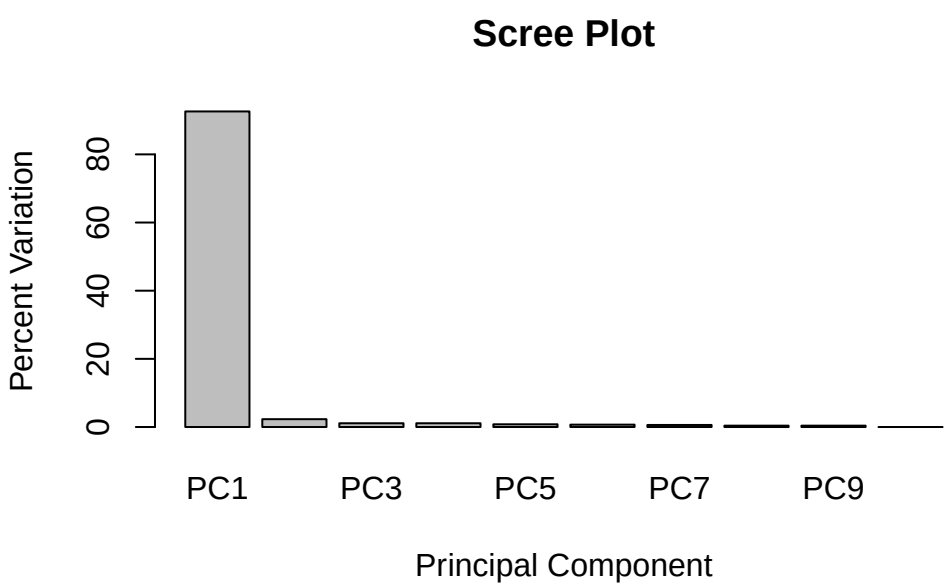
```r
plot(pca1)
```




```r
pca1.var <- pca1$sdev^2

pca1.var.per <- round(pca1.var/sum(pca1.var)*100, 1)
pca1.var.per
```

```
 [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

```r
barplot(pca1.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```


Scree Plot
Percent Variation
0
20
40
60
80
PC1
PC3
PC5
PC7
PC9
Principal Component