Class07

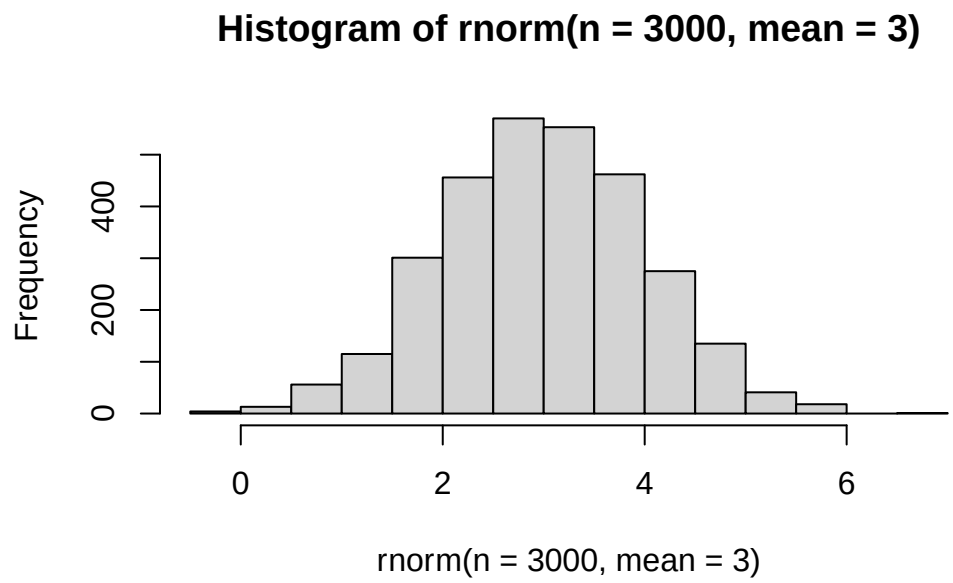
Jordan Prych (PID: A17080226)

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

Let's start my 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))
```

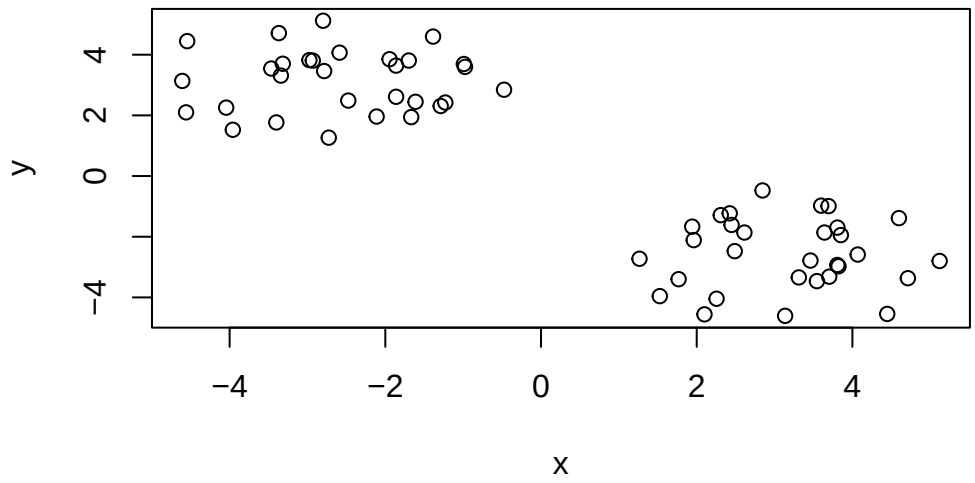


Make data `z` with two "clusters"

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

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

```
             x        y
[1,] -3.341485 3.311723
[2,] -3.400204 1.767580
[3,] -1.611900 2.447977
[4,] -3.317481 3.702444
[5,] -2.975029 3.821350
[6,] -2.785738 3.460876
```

```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  3.142086 -2.566440
2 -2.566440  3.142086

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] 67.66904 67.66904
 (between_SS / total_SS =  87.8 %)

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 lies in which cluster)?

```
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?

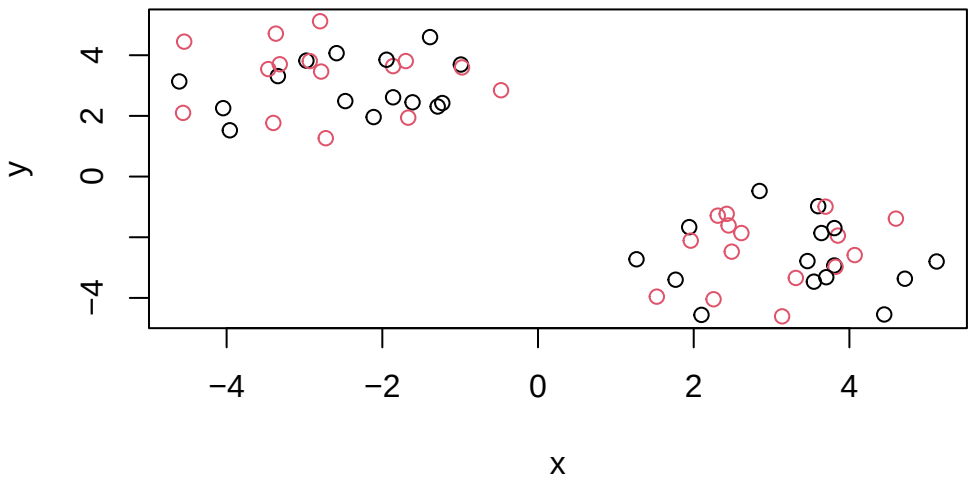
```
k$center
```

```
          x         y
1  3.142086 -2.566440
2 -2.566440  3.142086
```

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

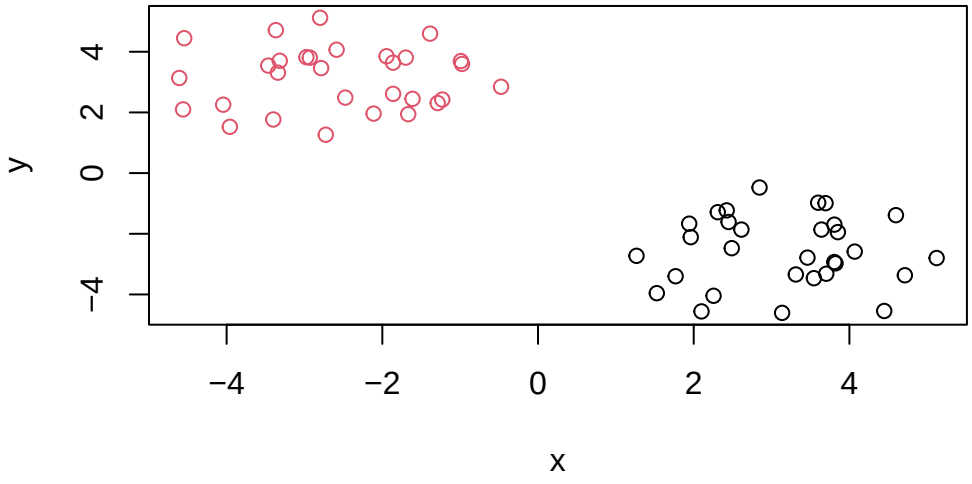
You can color by number.

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

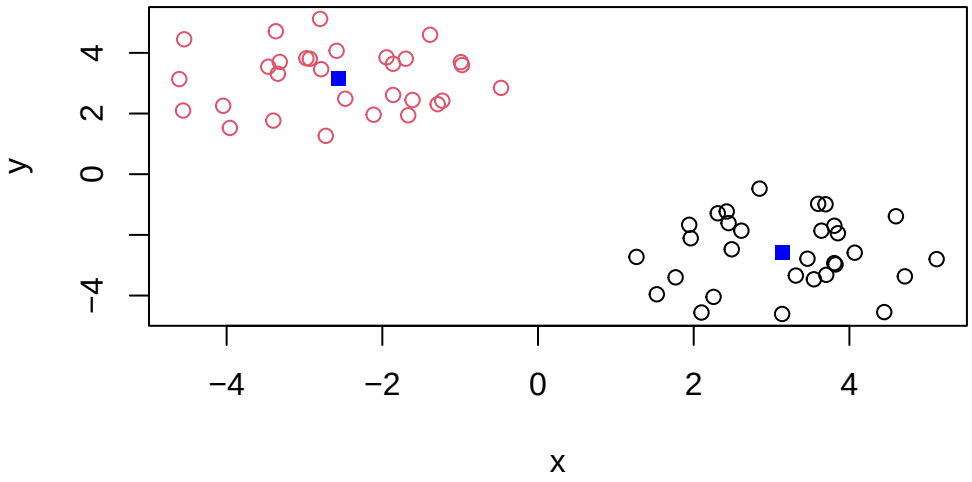


Plot colored by cluster membership:

```
plot(z, col=k$cluster)
```



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



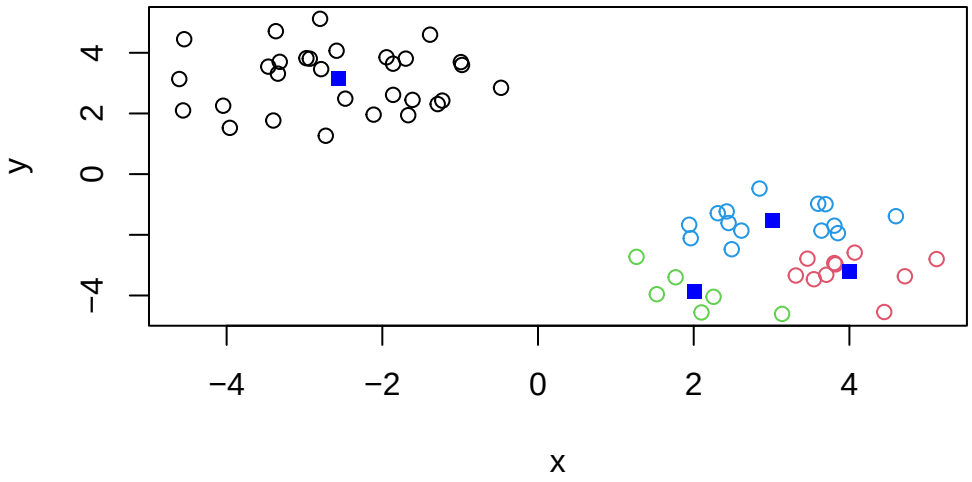
Q. Run K-means on our unput `z` and define 4 clusters making the same results vizualization plot as about (plot of z colored by cluster membership).

```
s <- kmeans(z, centers=4)
s
```

```
K-means clustering with 4 clusters of sizes 30, 10, 6, 14

Cluster means:
          x         y
1 -2.566440  3.142086
2  3.999417 -3.211509
3  2.008123 -3.882822
4  3.015692 -1.541512

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 4 4 4 4 4
[39] 4 3 4 3 2 3 3 4 2 4 4 4 2 4 3 4 2 2 2 4 3 2
```

```
Within cluster sum of squares by cluster:
[1] 67.669044  5.867674  4.768308 12.479024
 (between_SS / total_SS =  91.8 %)

Available components:

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

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



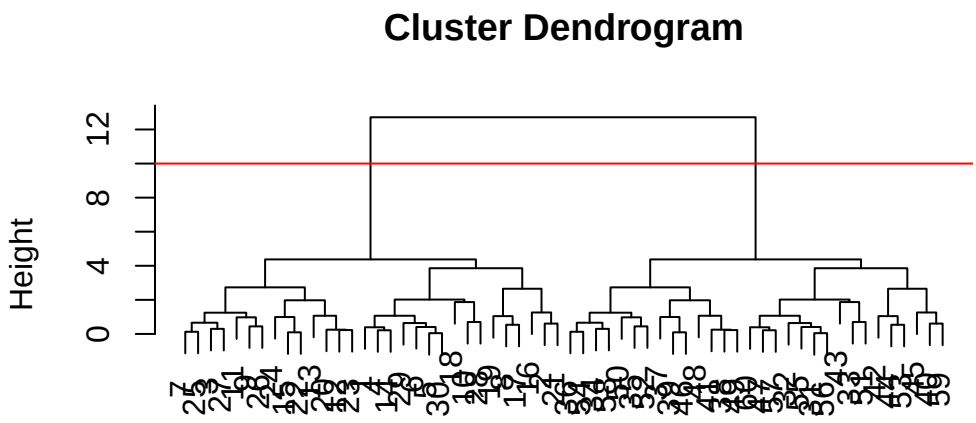
Heirarchial Clustering

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

```
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")
```

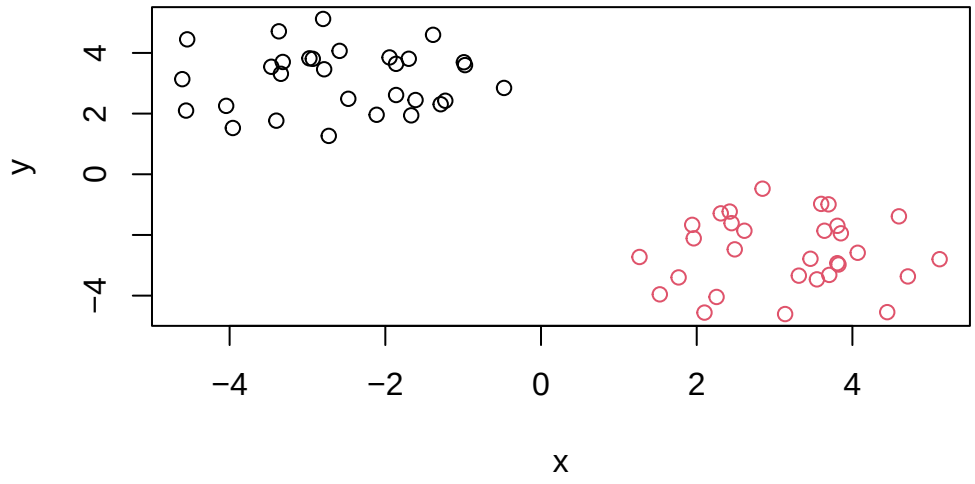


Once I inspect the cluster dendrogram "tree", I can "cut" the tree to yield my groupings or clusters. The function to do this is called `cutree()`

```
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 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(z, col=grps)
```



Hands on with Principal Component Analysis(PCA)

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

```
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 rown and columns are in your new data fram names x? What R functions could you use to answer this question?

```
nrow(x)
```

```
[1] 17
```

```
ncol(x)
```

```
[1] 4
```

```
dim(x)
```

```
[1] 17  4
```

Preview the first 6 rows with `head()` function.

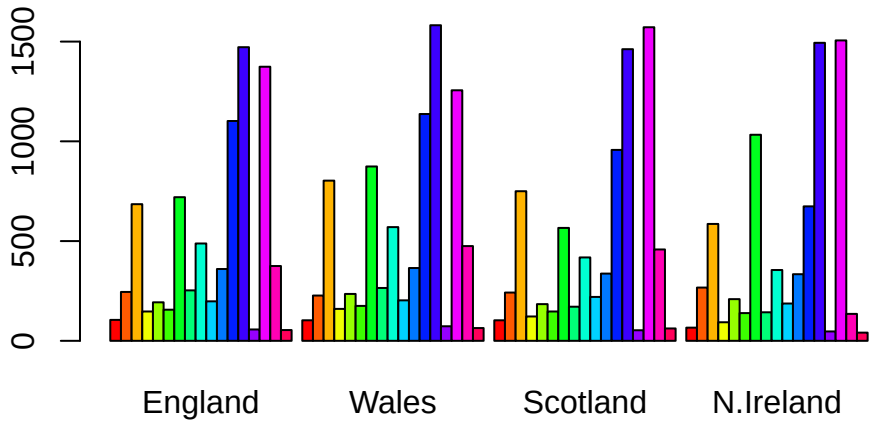
```
head(x)
```

```
               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 to solve the "row-names problem" using the row.names argument because this is a more robust method. If you run x <- x[,-1] multiple times, it will continuously remove columns until x is empty.

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

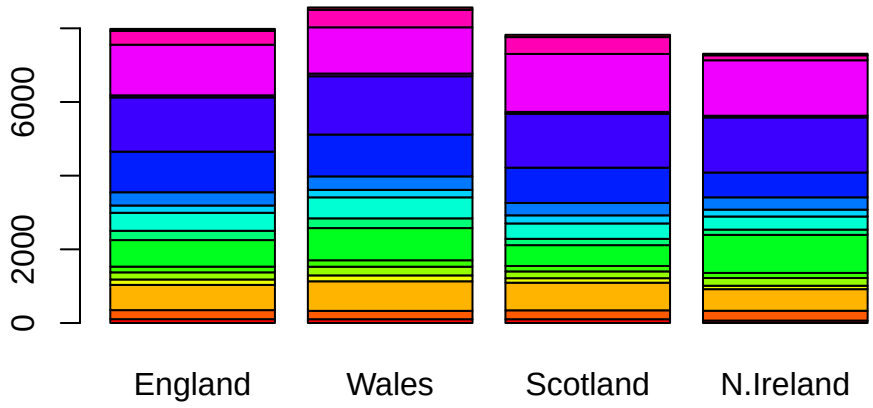


Q3. Changing what optional argument in the above barplot() function results in the following plot?

When you change the beside argument from `TRUE` to `FALSE`, this results each of the bars stacked on top of each other instead of beside each other in the bar plot.

An even worse plot:

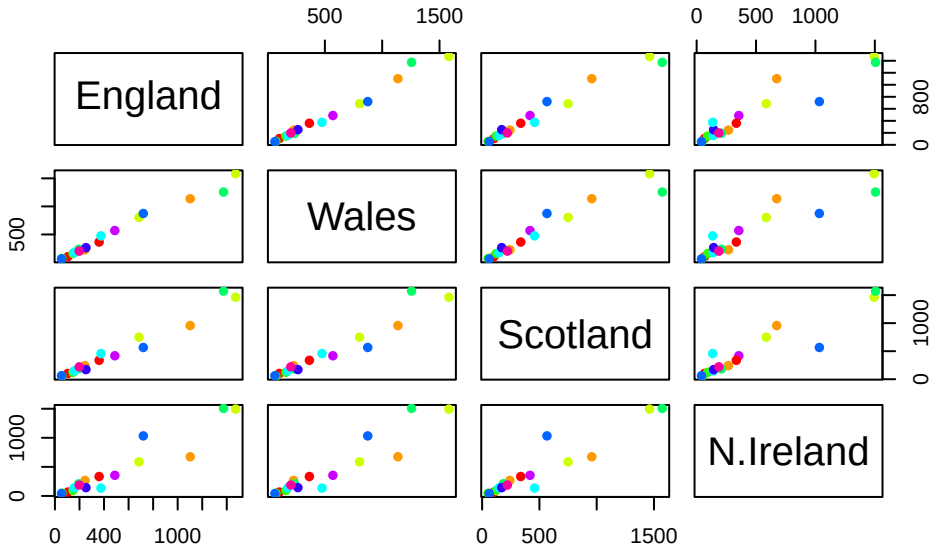
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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?

This plot attempts to plot each country against each other. Diagonal and horizontal countries are on the y-axis. Vertically, countries are on the x-axis. Points that lie on the straight line means that it is a similar amount of food consumed in both countries. If the point is not on the diagonal for a given plot means that more is consumer in one county than the other(depending on the plot). Each food type is plotted as a different rainbow color.

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



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

We can see that N. Ireland had an overall greater consumption of the "blue" variable compared to the other UK countries.

Looking at these types of "pairwise plots" can be helpful but it does not scale well and kid 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 want 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
```

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

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

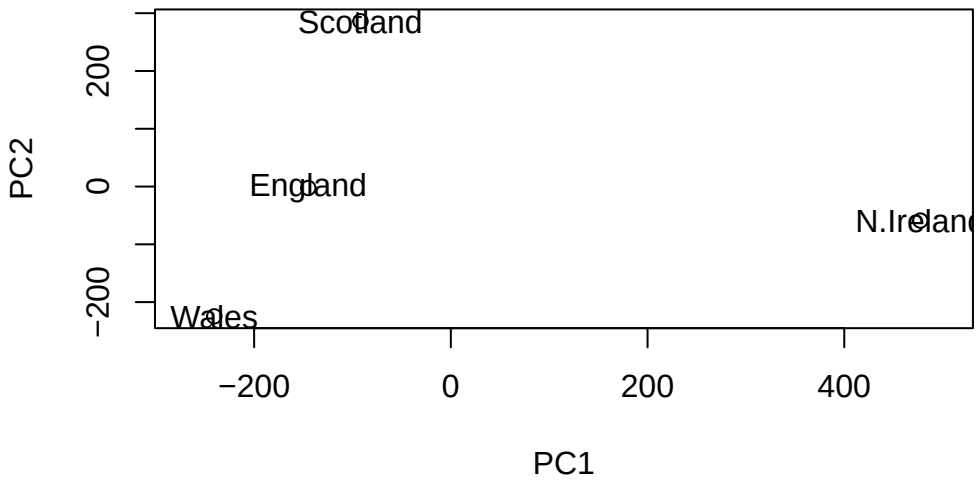
The `pca$x` result object is where we will focus first as this details how the countries are related to each other in terms of our new "axis" (a.k.a "PCs", "eigenvectors", etc.)

```
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
```

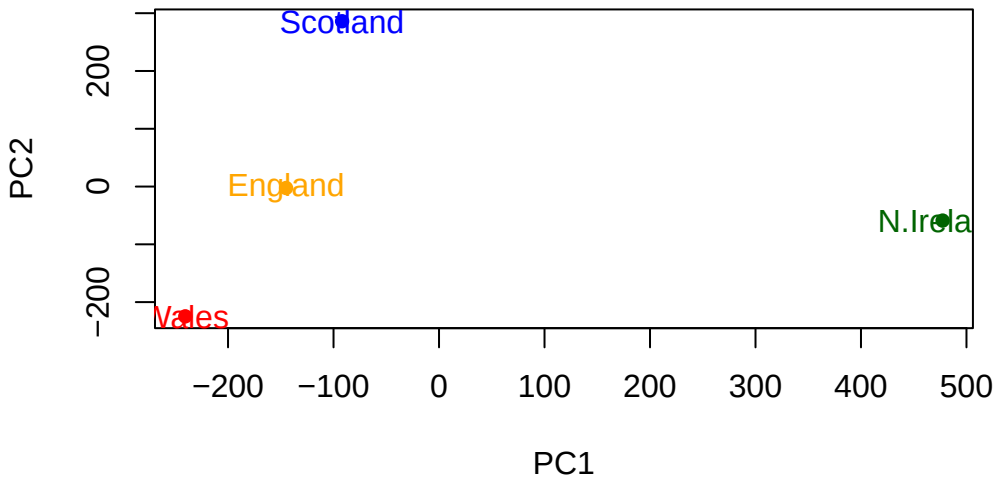
> Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

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

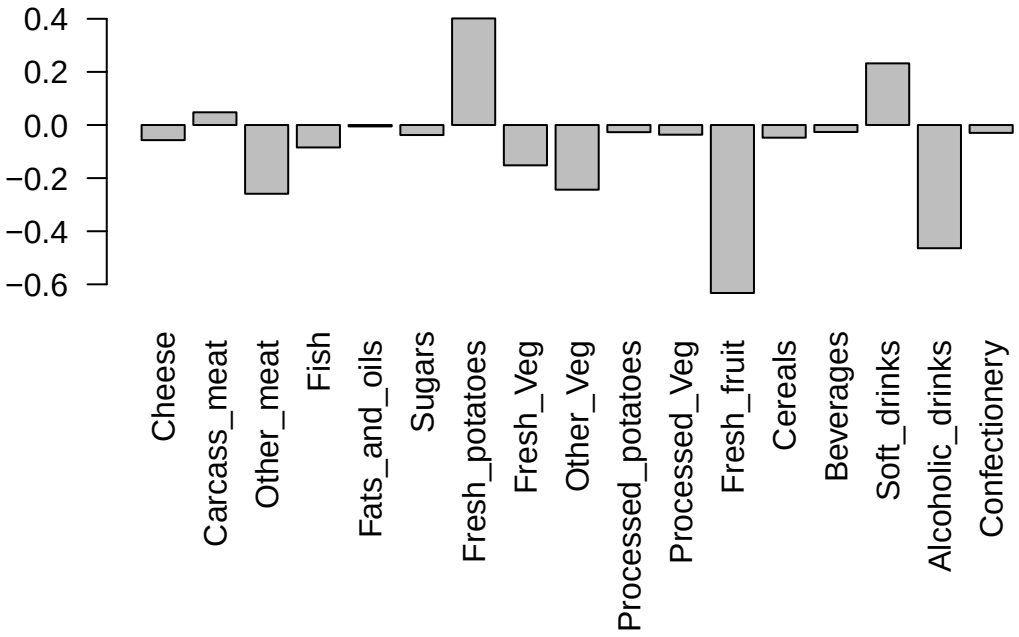


We can look at the so-called PC "loading" result object to see how the original foods contribute to our new PCs (i.e. how the original variables contribute to our new better variables). This plot is showing how variables are different.

```
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
```

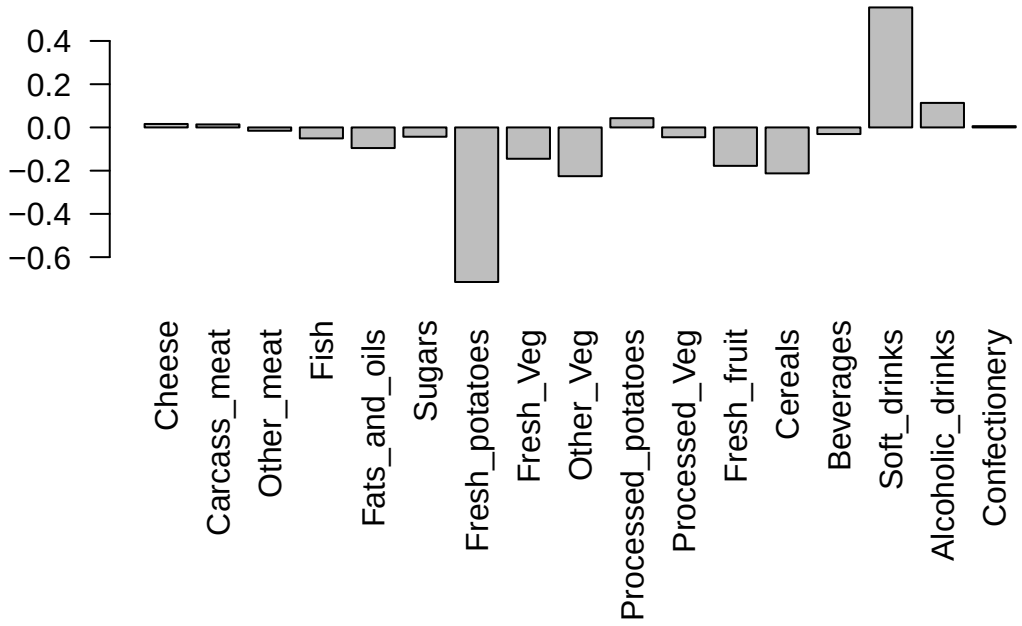
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



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

PC2 mainly tells us that the variance between countries can be summarized by the variance in soft drink consumption. The food groups that are featured prominently are soft drinks, fresh potatoes, and other vegetables.

```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```



PCA of RNA-seq Data

Here, we will use an example of RNA-seq data to demonstrate how this data can contain a PCA.

```
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
```

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

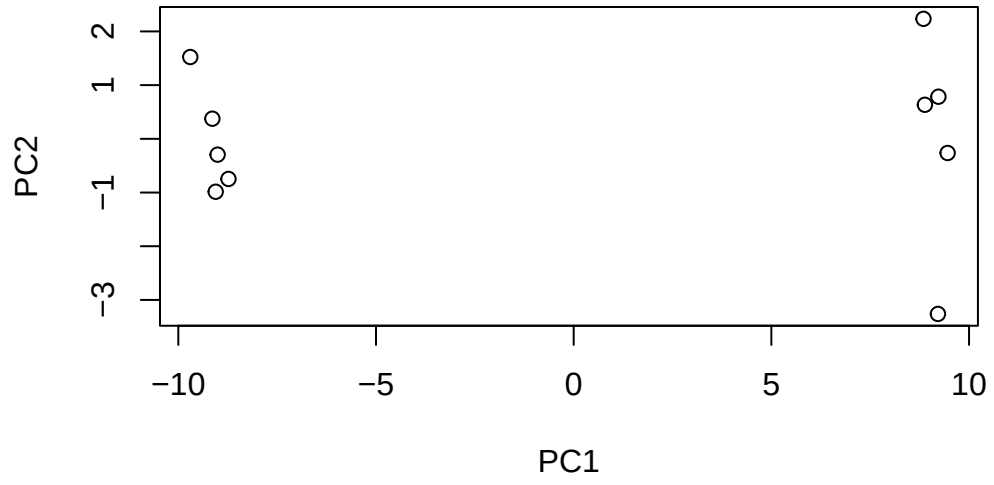
Genes are found in rows, while samples are found in columns. 100 genes, 10 samples.

```
dim(rna.data)
```

```
[1] 100  10
```

We now take the transpose of our data and plots PC1 vs. PC2.

```
pca <- prcomp(t(rna.data), scale=TRUE)
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
```

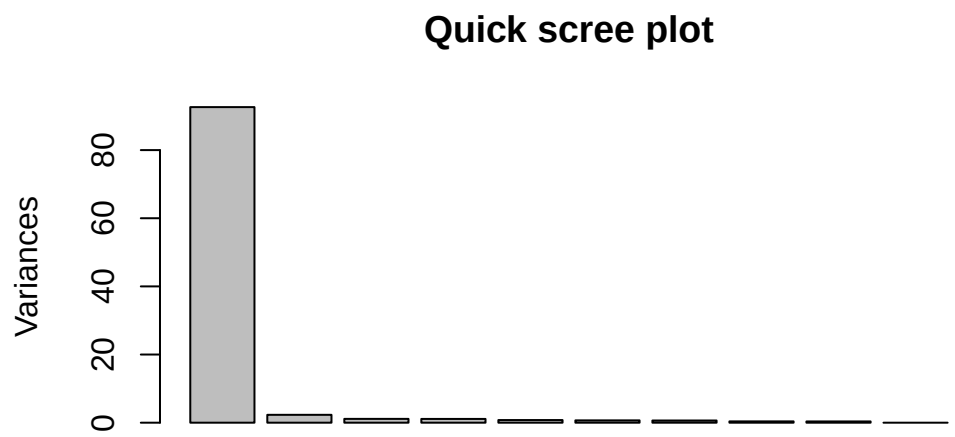



```
summary(pca)
```

```
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
```

We see above that PC1 holds most of the variation(92.6%), reducing the dimensional data down to one dimension. This can be further demonstrated with a scree plot.

```
plot(pca, main="Quick scree plot")
```

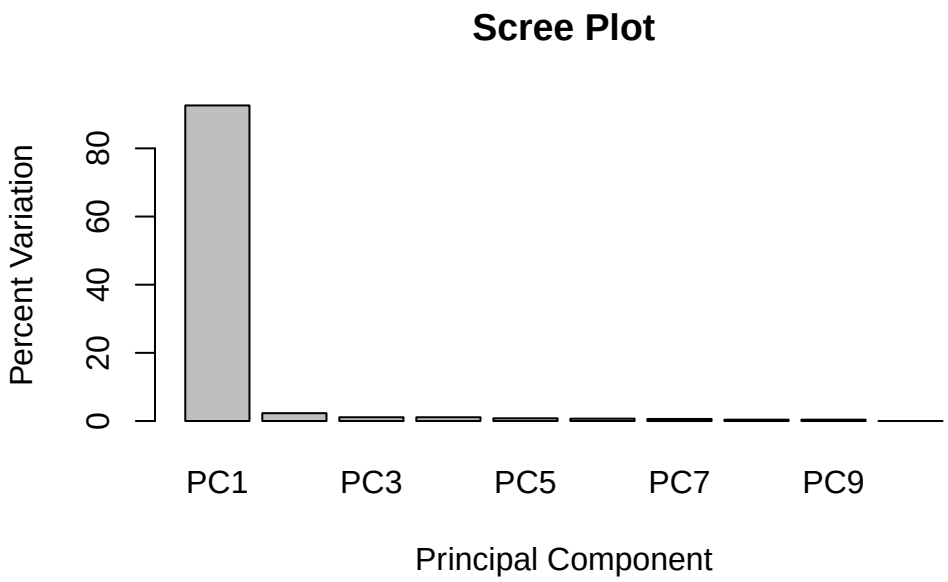


We can use `pca$sdev` to calculate how much variation in the original data each PC accounts for to generate a new scree plot.

```
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
pca.var.per
```

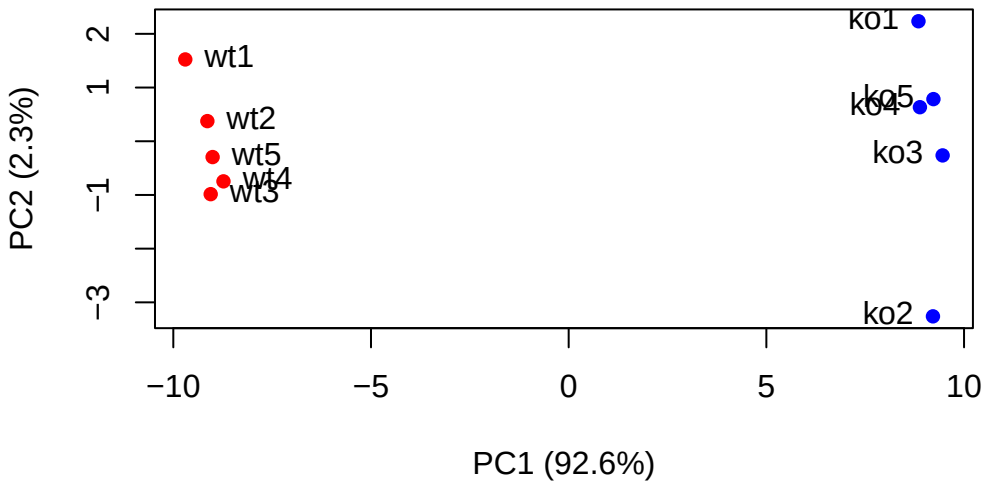
```
 [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

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



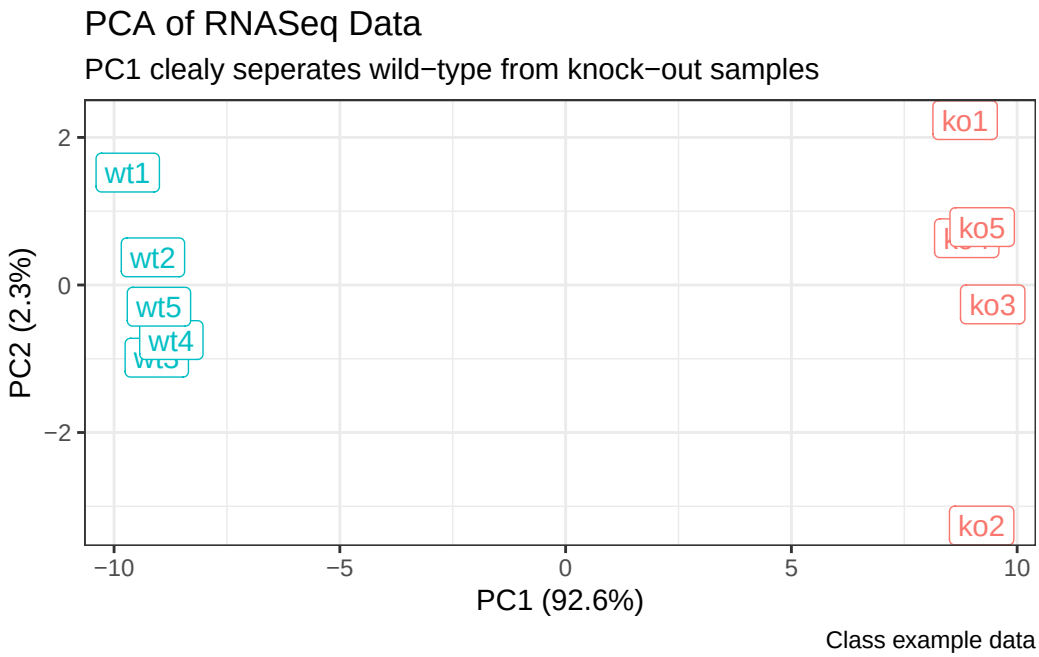
To make our original PCA more useful...

```
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

plot(pca$x[,1], pca$x[,2], col=colvec, pch=16,
     xlab=paste0("PC1 (", pca.var.per[1], "%)"),
     ylab=paste0("PC2 (", pca.var.per[2], "%)"))

text(pca$x[,1], pca$x[,2], labels = colnames(rna.data), pos=c(rep(4,5), rep(2,5)))
```



We can also plot this PCA using ggplot:

```
library(ggplot2)
df <- as.data.frame(pca$x)
df$samples <- colnames(rna.data)
df$condition <- substr(colnames(rna.data),1,2)
p <- ggplot(df) +
        aes(PC1, PC2, label=samples, col=condition) +
        geom_label(show.legend = FALSE)
p + labs(title="PCA of RNASeq Data",
       subtitle = "PC1 clealy seperates wild-type from knock-out samples",
       x=paste0("PC1 (", pca.var.per[1], "%)"),
       y=paste0("PC2 (", pca.var.per[2], "%)"),
       caption="Class example data") +
     theme_bw()
```


PCA of RNASeq Data
PC1 clealy seperates wild−type from knock−out samples
ko1
wt1
ko5
wt2
ko3
wt5
wt4
ko2
2
0
−2
PC2 (2.3%)
−10
−5
0
5
10
PC1 (92.6%)
Class example data