Class 7: Machine Learning 1

Ellice Wang (PID: A16882742)

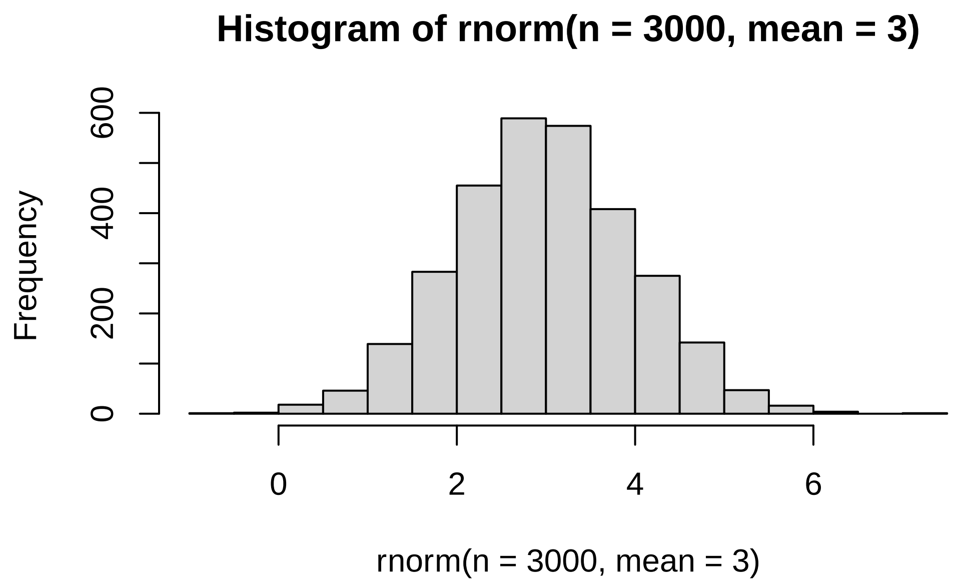
2025-01-28

Today we will explore unsupervised machine learning methods including clustering & dimensionality 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 'rnorm()' function to help us here:

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

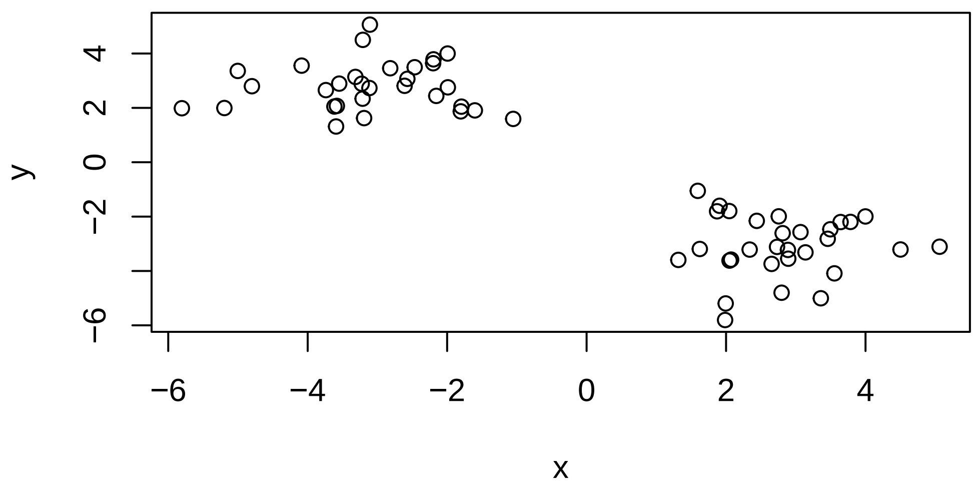


Make data with two "clusters"

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

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

```
             x        y
[1,] -3.190996 1.623297
[2,] -1.989774 2.755709
[3,] -2.155930 2.440371
[4,] -1.993812 3.998618
[5,] -2.200725 3.642545
[6,] -1.804729 1.871042
```

```
plot(z)
```



K-means clustering

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

```r
k <- kmeans(z, center = 2)
k
```

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

Cluster means:
          x         y
1  2.794307 -3.086687
2 -3.086687  2.794307

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 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] 59.59596 59.59596
 (between_SS / total_SS =  89.7 %)

Available components:

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

```r
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?

```r
k$size
```

```
[1] 30 30
```

> Q. What component of our results tells us about the cluster membership (i.e. which point lives 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 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
```

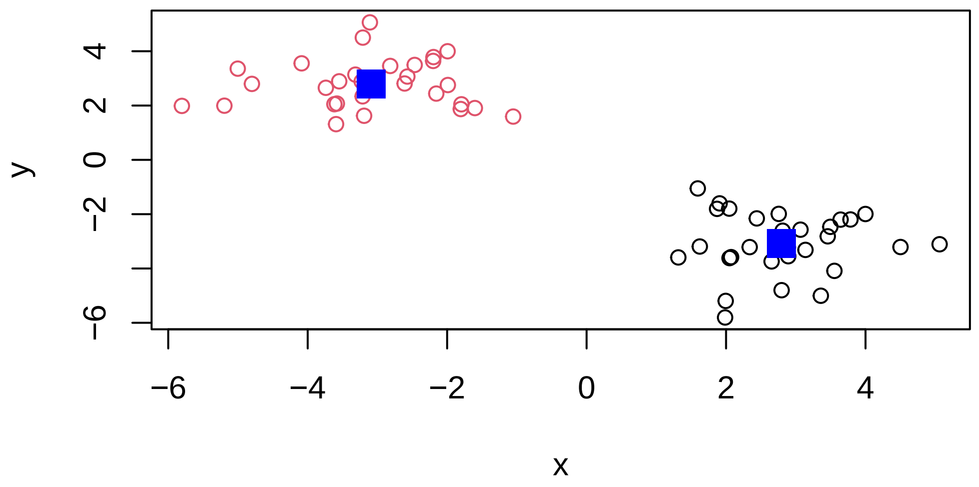
Q. Center of each cluster?

```
k$centers
```

```
          x         y
1  2.794307 -3.086687
2 -3.086687  2.794307
```

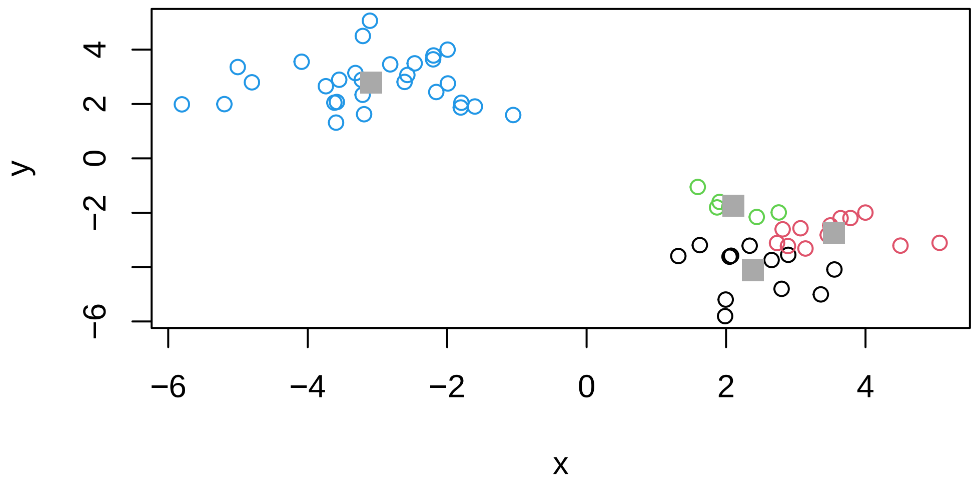
Q. Put this result info together and make a little "base R" plot of our clustering result. Also add the cluster center points to this plot.

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



Q. Run kmeans on our input 'z' and define 4 clusters

```
four_cluster <- kmeans(z, centers=4)
plot(z, col=four_cluster$cluster)
points(four_cluster$centers, col="darkgrey", pch=15, cex=1.5)
```



```
# total sum of squares which tells us how spread out it is
# the smaller the totss, the better the visualization
four_cluster$totss
```

```
[1] 1156.775
```

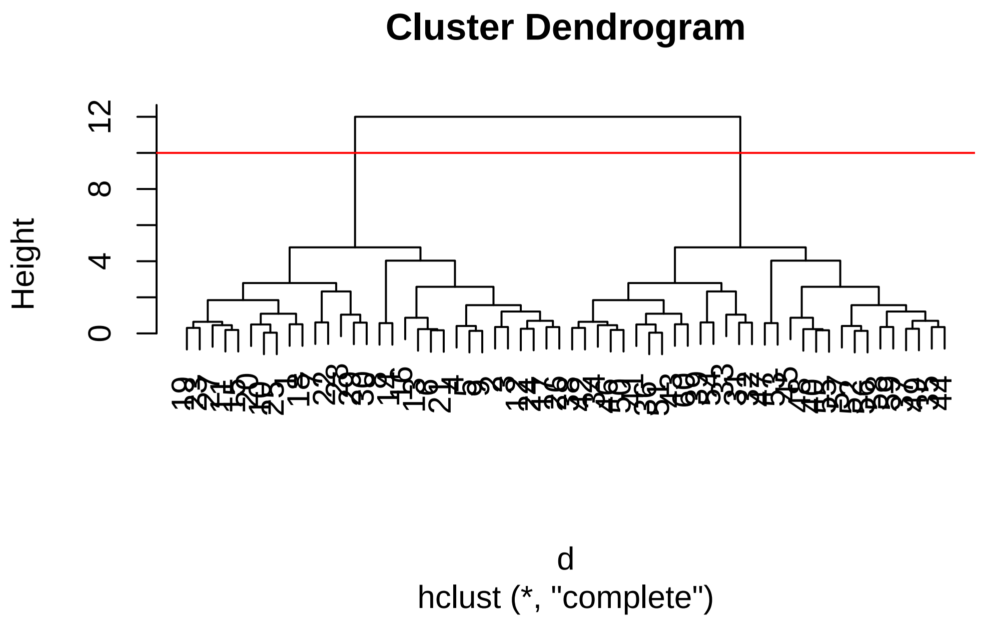
Hierarchical Clustering

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 give your "raw" data as input - must first calculate a distance matrix from data)

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

```
Call:
hclust(d = d)

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

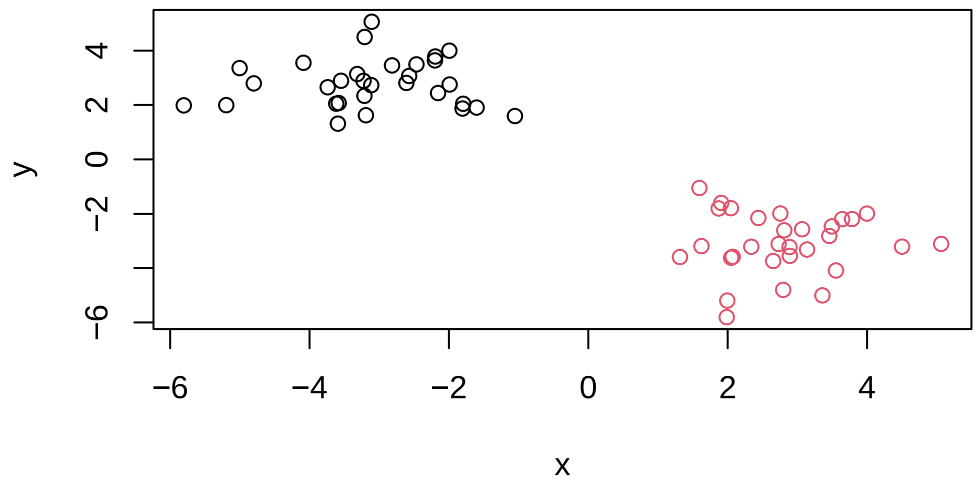
```
plot(hc)
abline(h=10, col="red")
```



Once I inspect the “tree” I can “cut” the tree to yield my groupings or clusters. The function to this is called ‘cutree()’

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

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



Hands on with Principle Component Analysis (PCA)

Let’s examine some 17-dimensional data.

```r
# load in dataframe of UK foods
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url)

# explore the data
dim(x)
```

```
[1] 17  5
```

> Q1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?
> There are 17 rows and 5 columns in the data frame. I used ‘dim()’ to answer this question, but ‘nrow()’ and ‘ncol()’ would also work.

```r
# preview first 6 rows
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
```

```
# removes first column names

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
```

```
# check dimensions of edited dataframe
dim(x)
```

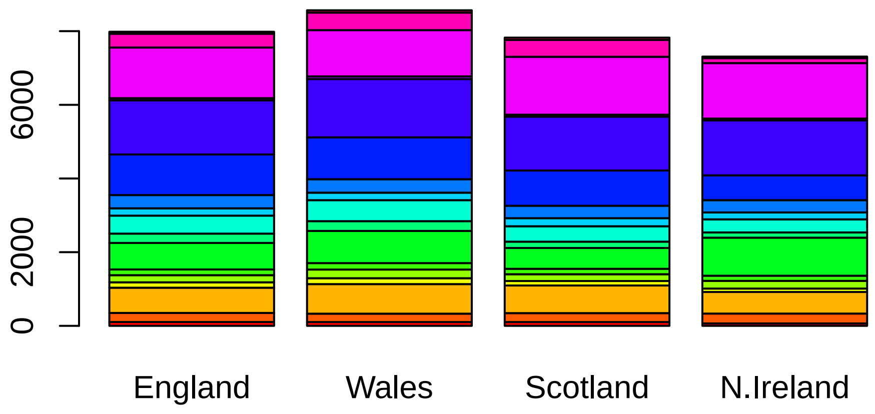
```
[1] 17  4
```

```
# alternative way to edit dataframe

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 using the method that sets the 'row.names' argument when loading in the dataframe. It is a more streamlined way to input data. I would use it after checking what the data looked like and determining whether it should be used. Setting 'x <- x[,-1]' multiple times will remove the first column multiple times.

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



```
test <- as.matrix(x)
color <- rainbow(nrow(x))
test <- cbind(test, color)
test
```

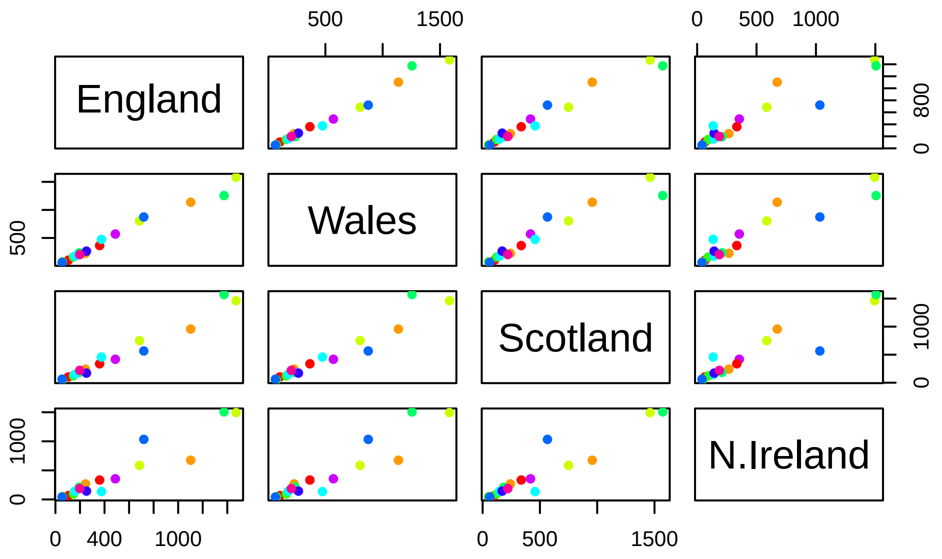
```
                    England Wales  Scotland N.Ireland color
Cheese              "105"   "103"  "103"    "66"      "#FF0000"
Carcass_meat        "245"   "227"  "242"    "267"     "#FF5A00"
Other_meat          "685"   "803"  "750"    "586"     "#FFB400"
Fish                "147"   "160"  "122"    "93"      "#F0FF00"
Fats_and_oils       "193"   "235"  "184"    "209"     "#96FF00"
Sugars              "156"   "175"  "147"    "139"     "#3CFF00"
```

```
Fresh_potatoes       "720"    "874"  "566"     "1033"     "#00FF1E"
Fresh_Veg            "253"    "265"  "171"     "143"      "#00FF78"
Other_Veg            "488"    "570"  "418"     "355"      "#00FFD2"
Processed_potatoes   "198"    "203"  "220"     "187"      "#00D2FF"
Processed_Veg        "360"    "365"  "337"     "334"      "#0078FF"
Fresh_fruit          "1102"   "1137" "957"     "674"      "#001EFF"
Cereals              "1472"   "1582" "1462"    "1494"     "#3C00FF"
Beverages            "57"     "73"   "53"      "47"       "#9600FF"
Soft_drinks          "1374"   "1256" "1572"    "1506"     "#F000FF"
Alcoholic_drinks     "375"    "475"  "458"     "135"      "#FF00B4"
Confectionery        "54"     "64"   "62"      "41"       "#FF005A"
```

> Q3: Changing what optional argument in the above barplot() function results in the following plot?
> instead of 'beside=T' I set 'beside=F' to generate the stacked barplot

> 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 result is a matrix of scatterplots of the dataframe comparing the amount of different food groups each country eats. If a given point is on the diagonal for a given plot, it means that the two countries each similar amounts of that food. For example, England vs Wales plots are in the 2x1 and 1x2 positions. The blue dot lies on the diagonal for these two countries, indicating that England and Wales consume similar amounts of the food represented by the blue dot.

```r
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?
Northern Ireland's biggest difference is their consumption of the foods represented by the orange and blue dots. Orange represents carcass meat and the blue represents fresh fruit.

Looking at these types of "pairwise plots" can be helpful but it does not scale well & kind of sucks...

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 foods in as columns and the countries as rows.

```r
# transpose data
head(t(x))
```

```
          Cheese Carcass_meat  Other_meat  Fish Fats_and_oils  Sugars
England      105          245         685   147           193     156
Wales        103          227         803   160           235     175
Scotland     103          242         750   122           184     147
N.Ireland     66          267         586    93           209     139
```

```
          Fresh_potatoes  Fresh_Veg  Other_Veg  Processed_potatoes
England              720        253        488                 198
Wales                874        265        570                 203
Scotland             566        171        418                 220
N.Ireland           1033        143        355                 187
          Processed_Veg  Fresh_fruit  Cereals  Beverages Soft_drinks
England             360         1102     1472         57        1374
Wales               365         1137     1582         73        1256
Scotland            337          957     1462         53        1572
N.Ireland           334          674     1494         47        1506
          Alcoholic_drinks  Confectionery
England                375             54
Wales                  475             64
Scotland               458             62
N.Ireland              135             41
```

```
# 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 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 results

```
attributes(pca)
```

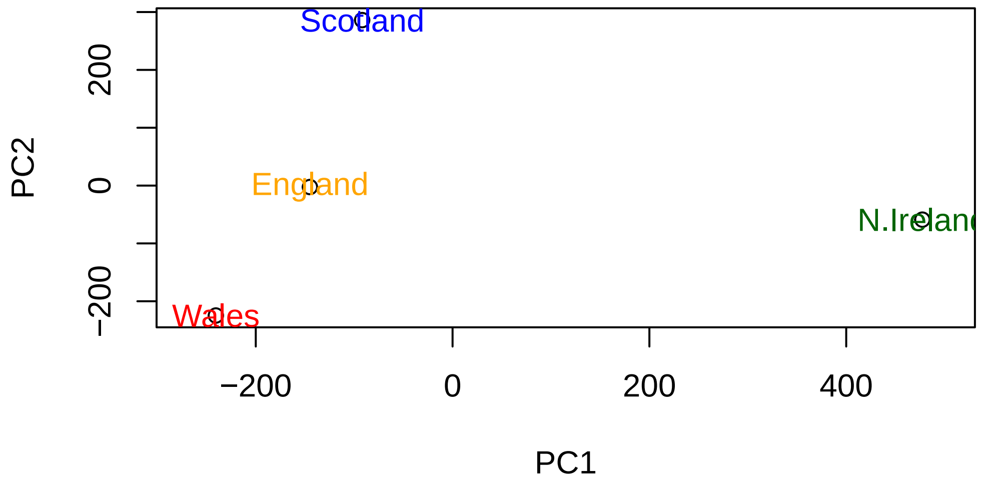
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$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).

```
pca$x[,2]
```

```
    England        Wales     Scotland   N.Ireland
  -2.532999 -224.646925  286.081786  -58.901862
```

```
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",
                                              "darkgreen"))
```



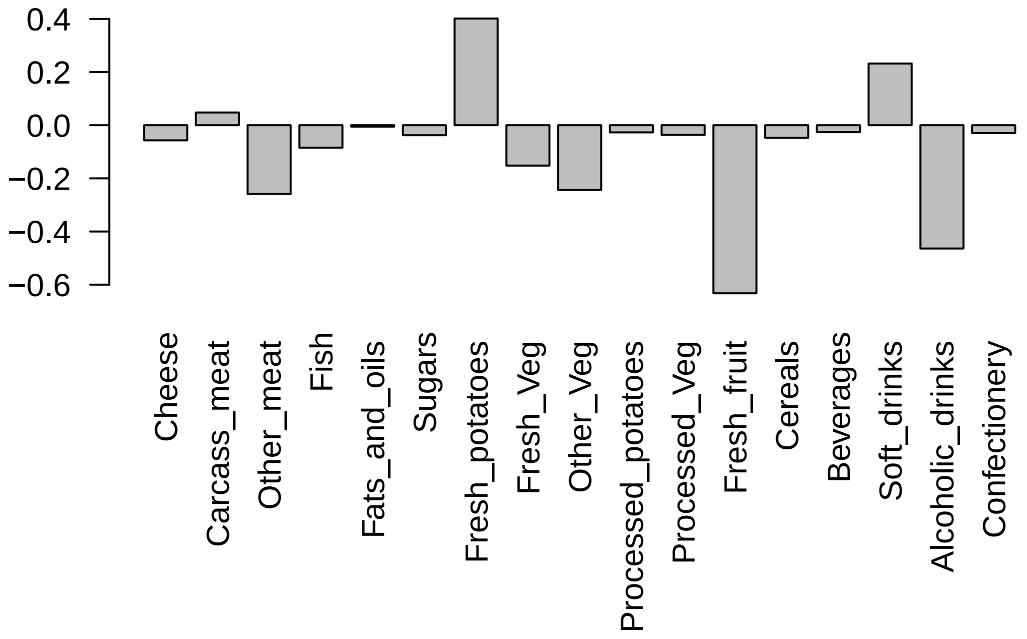
We can look at the so-called PC "loadings" result to see how the original foods contribute to our new PCs (i.e. how the original variables contribute to our new better variables)

```
pca$rotation
```

```
                         PC1          PC2         PC3          PC4
Cheese           -0.056955380  0.016012850  0.02394295 -0.694538519
Carcass_meat      0.047927628  0.013915823  0.06367111  0.489884628
Other_meat       -0.258916658 -0.015331138 -0.55384854  0.279023718
Fish             -0.084414983 -0.050754947  0.03906481 -0.008483145
Fats_and_oils    -0.005193623 -0.095388656 -0.12522257  0.076097502
Sugars           -0.037620983 -0.043021699 -0.03605745  0.034101334
```

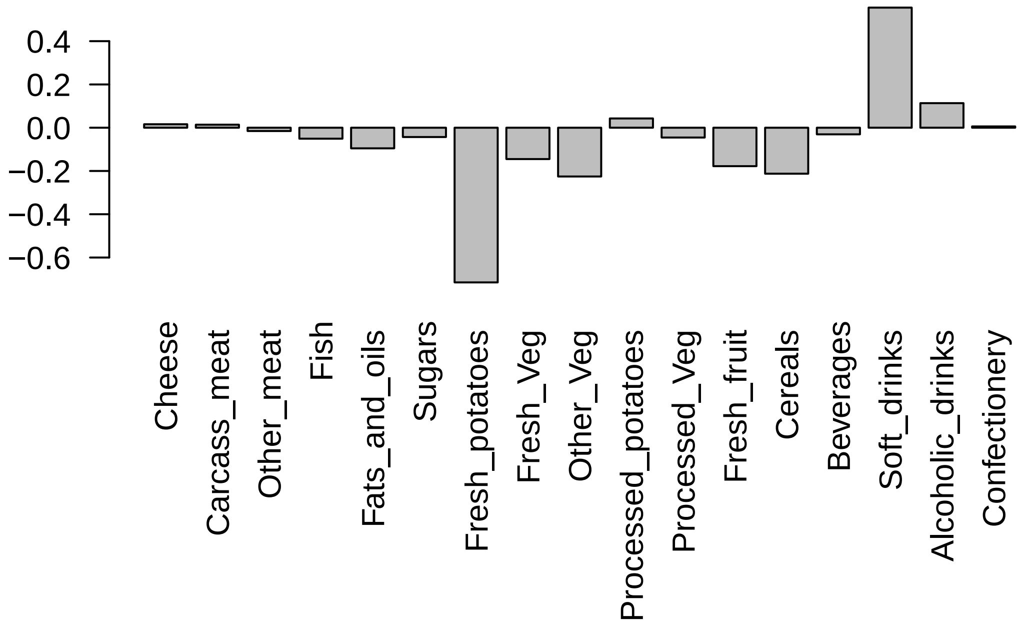
```
Fresh_potatoes       0.401402060 -0.715017078 -0.20668248 -0.090972715
Fresh_Veg           -0.151849942 -0.144900268  0.21382237 -0.039901917
Other_Veg           -0.243593729 -0.225450923 -0.05332841  0.016719075
Processed_potatoes  -0.026886233  0.042850761 -0.07364902  0.030125166
Processed_Veg       -0.036488269 -0.045451802  0.05289191 -0.013969507
Fresh_fruit         -0.632640898 -0.177740743  0.40012865  0.184072217
Cereals             -0.047702858 -0.212599678 -0.35884921  0.191926714
Beverages           -0.026187756 -0.030560542 -0.04135860  0.004831876
Soft_drinks          0.232244140  0.555124311 -0.16942648  0.103508492
Alcoholic_drinks    -0.463968168  0.113536523 -0.49858320 -0.316290619
Confectionery       -0.029650201  0.005949921 -0.05232164  0.001847469
```

```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 )
```

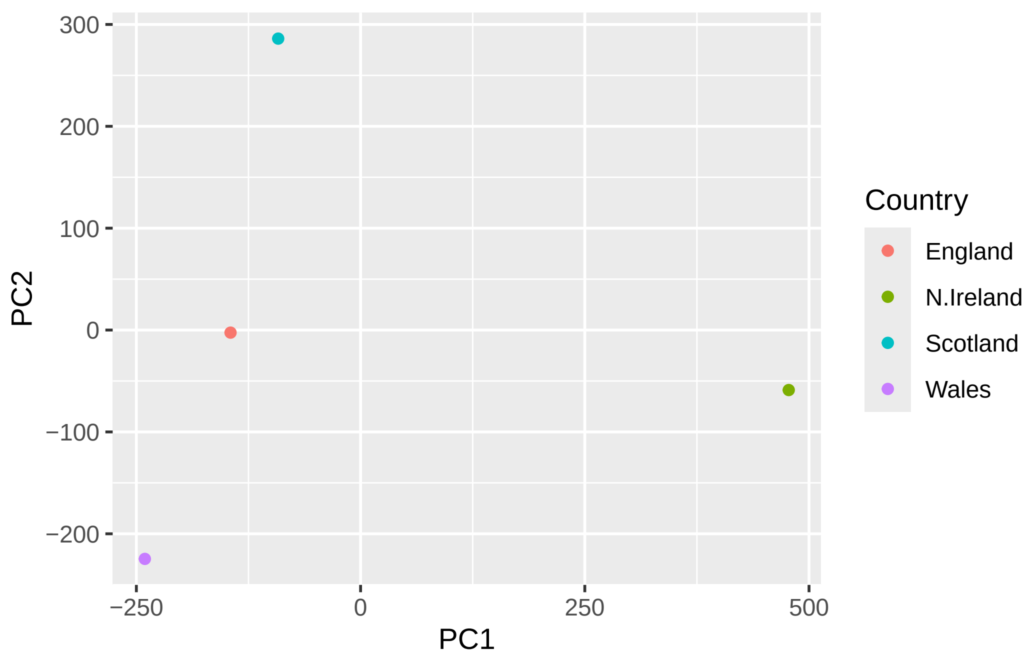


Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?
Fresh potatoes and soft drinks feature prominantly (similar to PC1). The other food groups feature less prominantly than in the first loading graph. PC2 is the vector that is perpendicular to PC1 and tells us about the second largest source of variation in the data not captured by PC1.

```
# loading plot for PC2
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,2], las=2 )
```

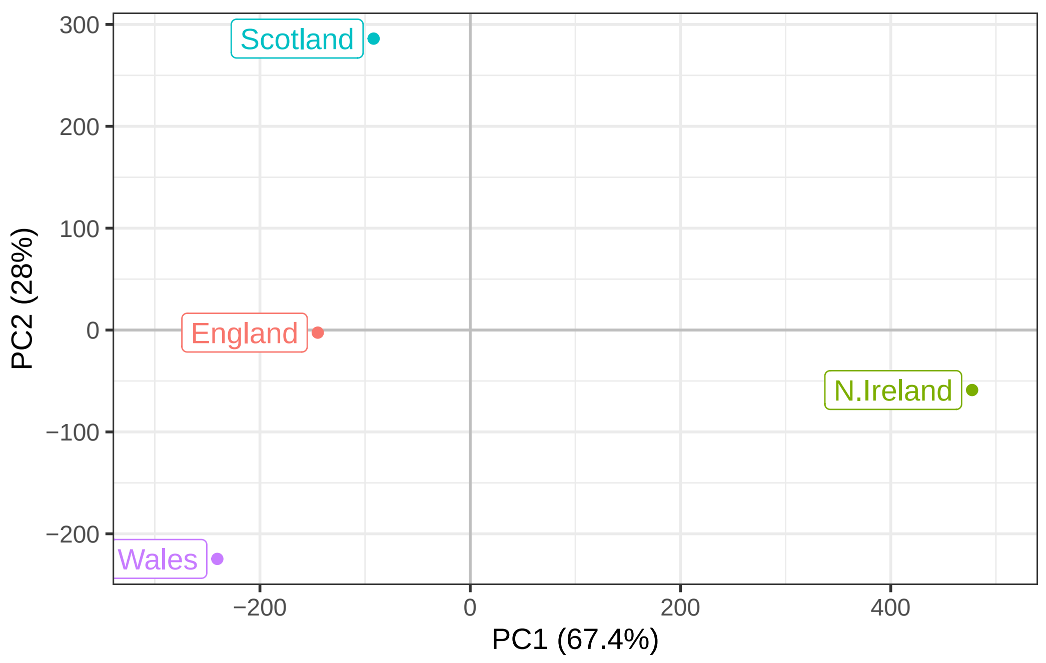


```
library(ggplot2)

df <- as.data.frame(pca$x)
df_lab <- tibble::rownames_to_column(df, "Country")

# Our first basic plot
ggplot(df_lab) +
  aes(PC1, PC2, col=Country) +
  geom_point()
```



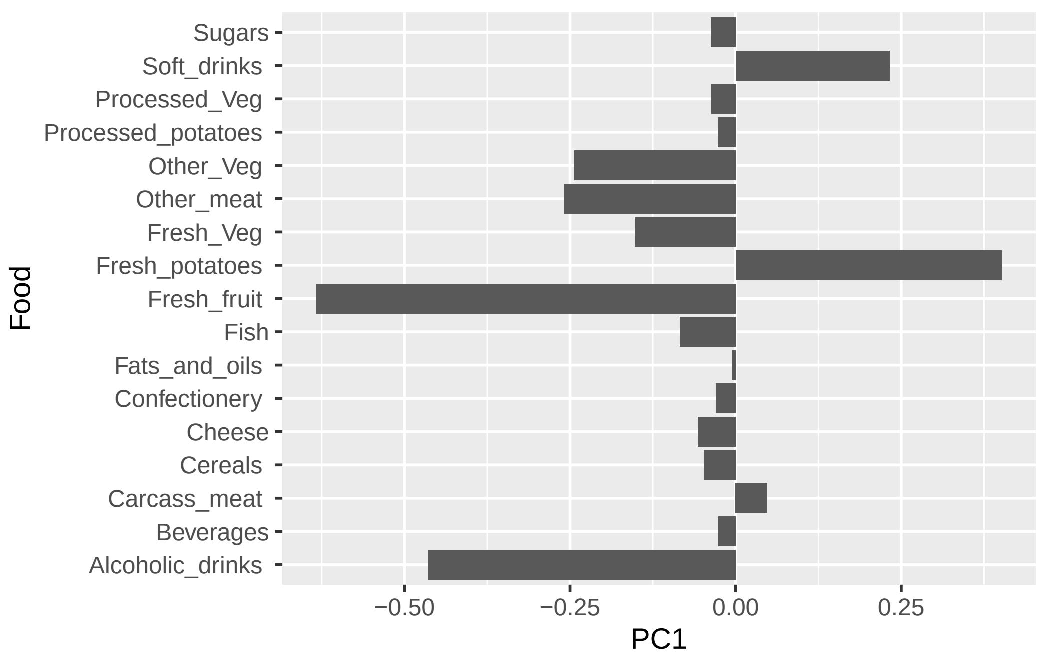
Nicer looking ggplot

```r
ggplot(df_lab) +
  aes(PC1, PC2, col=Country, label=Country) +
  geom_hline(yintercept = 0, col="gray") +
  geom_vline(xintercept = 0, col="gray") +
  geom_point(show.legend = FALSE) +
  geom_label(hjust=1, nudge_x = -10, show.legend = FALSE) +
  expand_limits(x = c(-300,500)) +
  xlab("PC1 (67.4%)") +
  ylab("PC2 (28%)") +
  theme_bw()
```



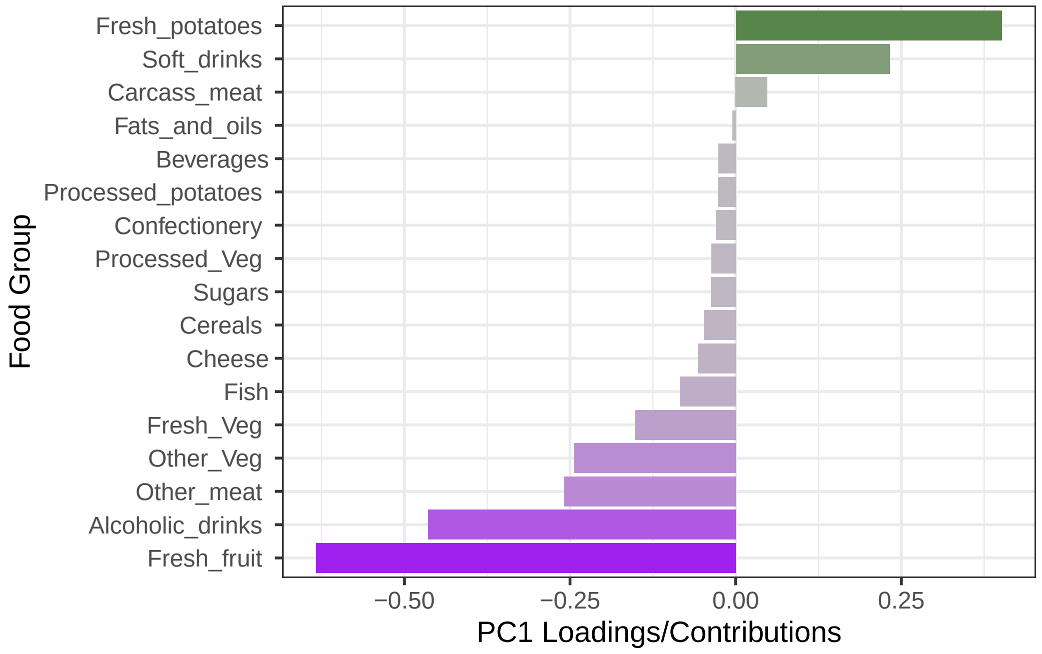
loading plots

```
ld <- as.data.frame(pca$rotation)
ld_lab <- tibble::rownames_to_column(ld, "Food")

ggplot(ld_lab) +
  aes(PC1, Food) +
  geom_col()
```



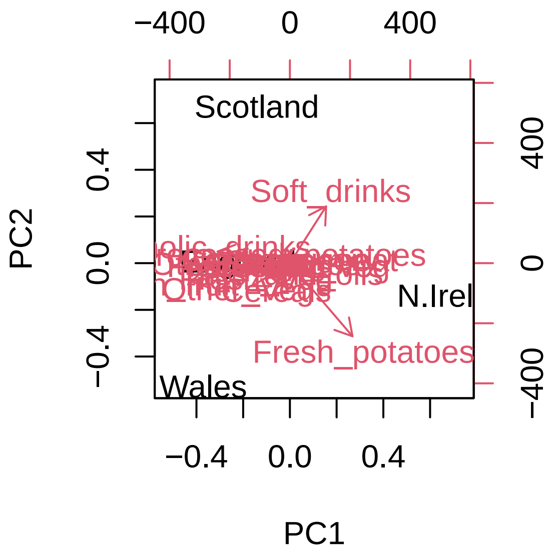
Nicer looking loading plot

```
ggplot(ld_lab) +
  aes(PC1, reorder(Food, PC1), bg=PC1) +
  geom_col() +
  xlab("PC1 Loadings/Contributions") +
  ylab("Food Group") +
  scale_fill_gradient2(low="purple", mid="gray", high="darkgreen", guide=NULL) +
  theme_bw()
```



Biplots are another way to visualize the information.

```
biplot(pca)
```



PCA of RNA-seq data

```
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
```

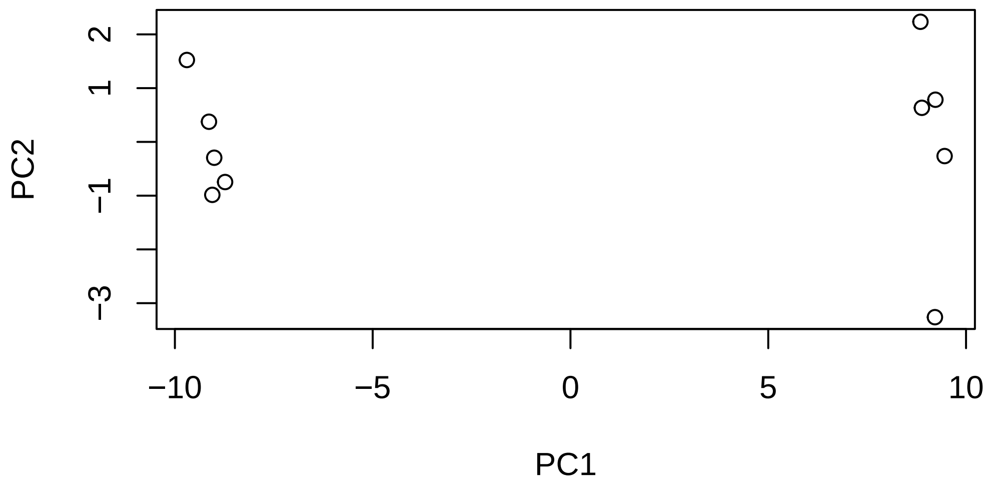
> Q10: How many genes and samples are in this data set?
> There are 100 genes and 10 samples in this dataset.

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

```
[1] 100  10
```

Plot a PCA graph of the RNA-seq data

```
## Again we have to take the transpose of our data
pca <- prcomp(t(rna.data), scale=TRUE)

## Simple un polished plot of pc1 and pc2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
```



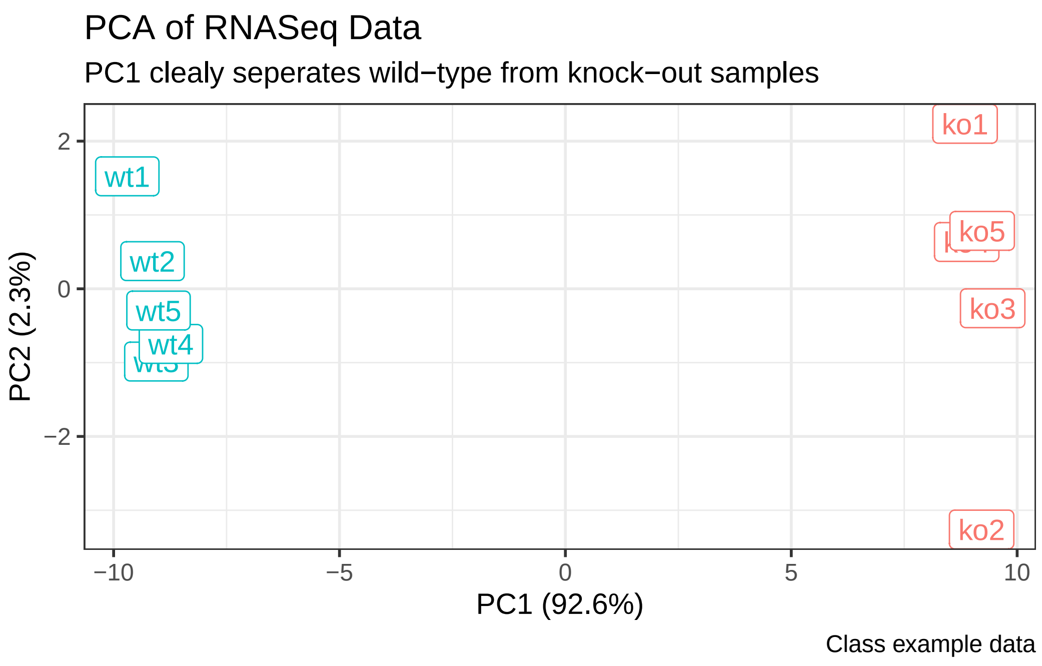
```r
summary(pca)
```

```
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
```

Using ggplot

```r
library(ggplot2)

## Variance captured per PC
pca.var <- pca$sdev^2

## Percent variance is often more informative to look at
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
```

```
df <- as.data.frame(pca$x)

# Add a 'wt' and 'ko' "condition" column to our plot
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) +
# add titles and labels and change theme to make graph look nicer
        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()
p
```


PCA of RNASeq Data
PC1 clealy seperates wild−type from knock−out samples
wt1
wt2
wt5
wt4
wt3
ko1
ko5
ko4
ko3
ko2
PC2 (2.3%)
PC1 (92.6%)
Class example data