# Class 7: Machine Learning 1

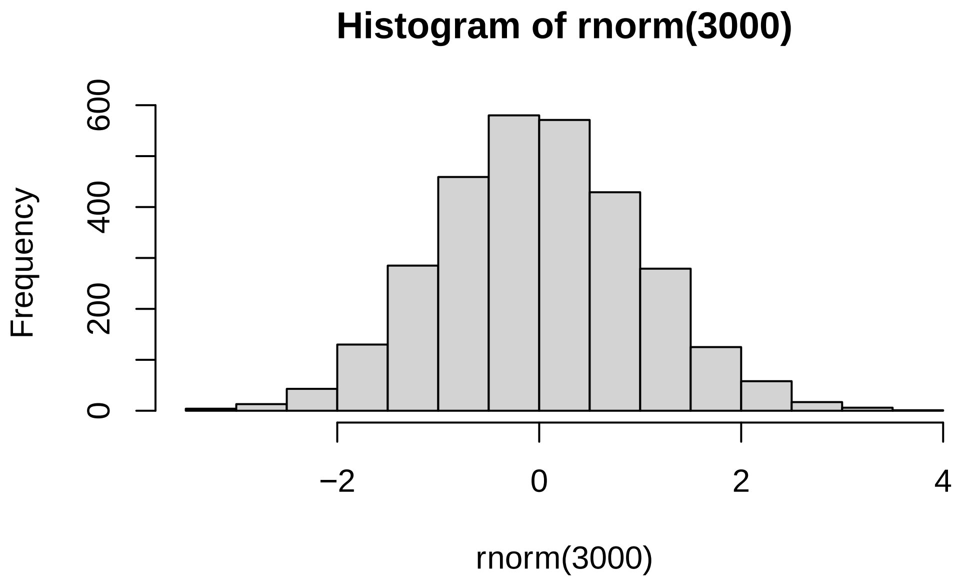
Snehita Vallumchetla (PID A16853399)

Today we will explore unsupervied machine learning methods including clustering and dimentionallity 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(3000))
```

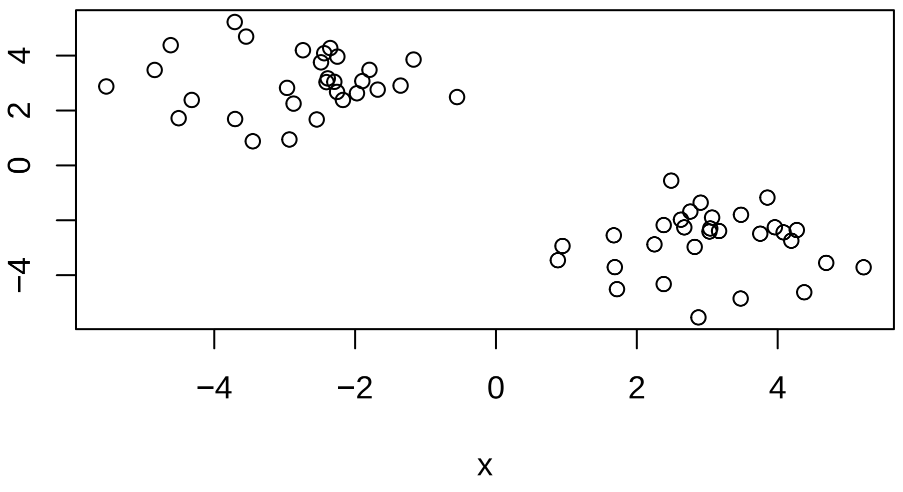


We make data with two "cluster"

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

#cbind, causes columns to "bind together" there is also row bind for the binding of rows

z <- cbind(x = x, rev(x))

head(z)
```

```
             x
[1,] -2.173028 2.381752
[2,] -2.873963 2.250486
[3,] -2.966235 2.821831
[4,] -3.709090 5.220407
[5,] -2.545036 1.673360
[6,] -2.740639 4.193083
```

```r
plot(z)
```



How big is z

```r
nrow(z)
```

```
[1] 60
```

```r
ncol(z)
```

```
[1] 2
```

## K-means clustering

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

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

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

Cluster means:
          x
1 -2.792214  3.024492
2  3.024492 -2.792214

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 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] 70.8648 70.8648
 (between_SS / total_SS =  87.7 %)

Available components:

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

```r
# shows that each cluster has a size of 30
# clustering vector: it is the membership vector, which cluster do the points belong to
# available components: all of the stuff that is available
```

```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
# size shows how many points are in each vector
k$size
```

```
[1] 30 30
```

> Q. What component of our results tell us about the cluster membership (i.e. whcih points lie in which cluster)

```r
# shows the cluster that each point is allocated to
k$cluster
```

```
 [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 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
```

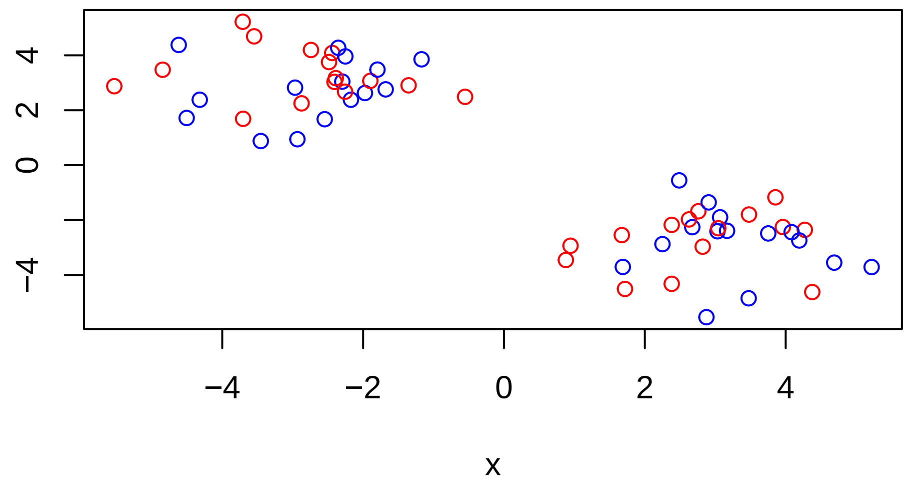
> Q. Center of each cluster

```r
k$centers
```

```
          x
1 -2.792214  3.024492
2  3.024492 -2.792214
```

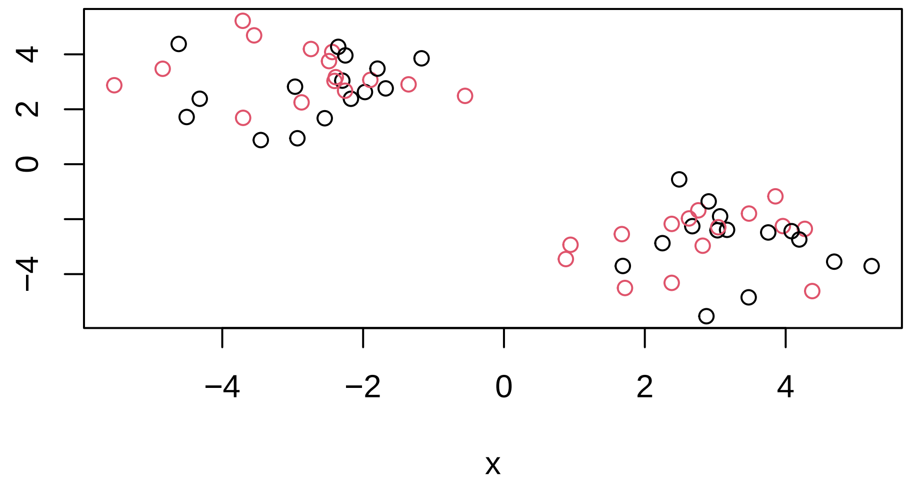
> Q. Put this result information together and make a little "base R" plot of our clustering results. Also add the cluster center points to the plot.

```r
plot(z, col = c('blue', 'red'))
```

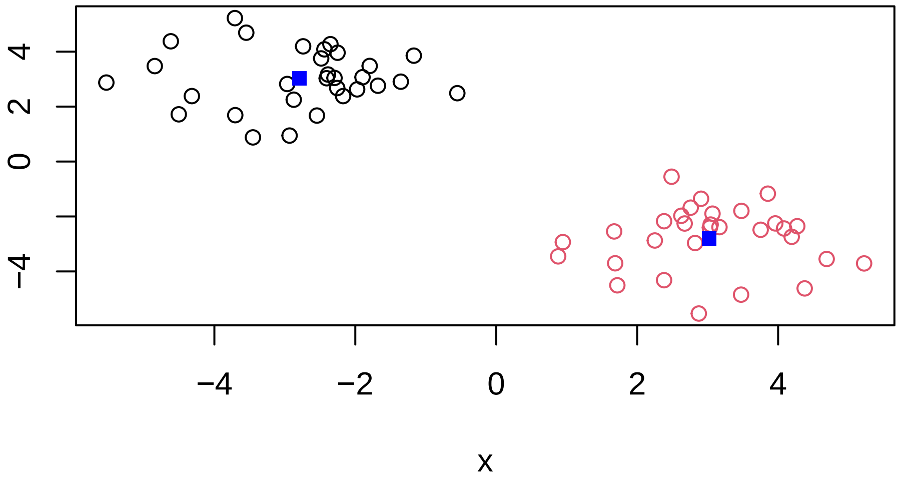


You are able to color plots by the number

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

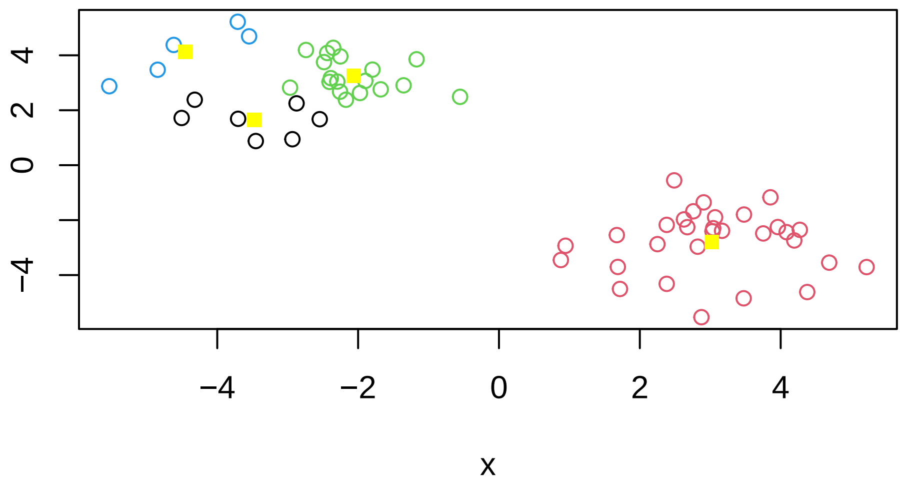


```r
# points are now colored by cluster membership
plot(z, col = k$cluster)
points(k$centers, col = 'blue', pch = 15)
```



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

```r
k2 = kmeans(z, centers = 4)

plot(z, col = k2$cluster)

points(k2$center, col = 'yellow', pch = 15)
```



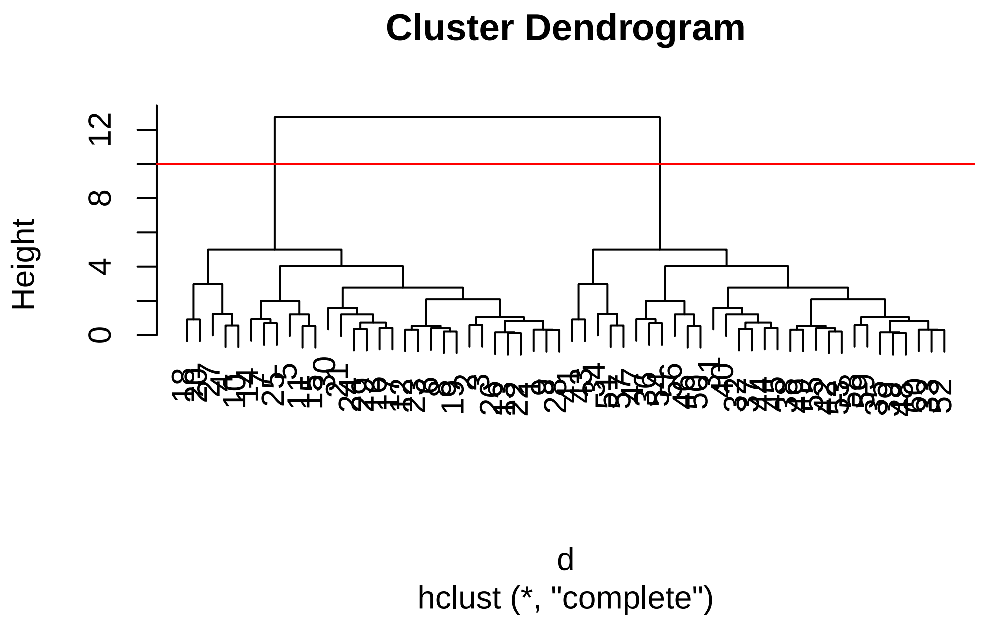
**Hierarchical Clustering:**

The main function in base R for this is called `hclust()` it will take as input a distance matrix (key point is that you cant just give 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 
```

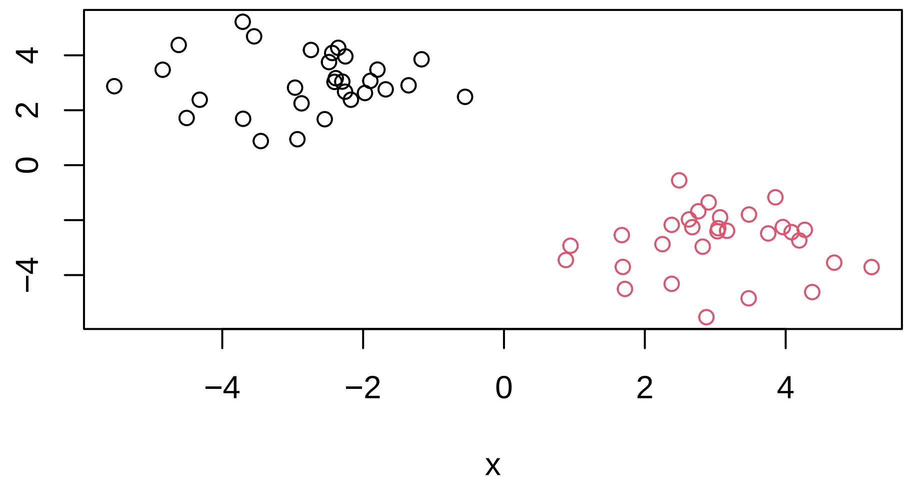
```r
plot(hc)
abline(h = 10, col = 'red')
```



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

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

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



## Dimensionality Rediction and PCA:

PCA is used to reduce the dimensions in data allowing for something that is able to be visualized. Plot data on a new axis which makes them better for viewing relationships. First principle component describes the most that it can, and the second describes anything else. Filters data, and maintains the main essence of the data.

Let's examine some silly 17-dimensional data detailing food consumption in the UK. Are these countries eating habits similar or different to one another.

```r
url <- "https://tinyurl.com/UK-foods"
x <- read.csv(url, row.names = 1)
```

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

```r
# can use the dim fucntion which outputs 17 rows, and 5 columns
dim(x)
```

```
[1] 17  4
```

```
# preview the first 6 rows
head(x)
```

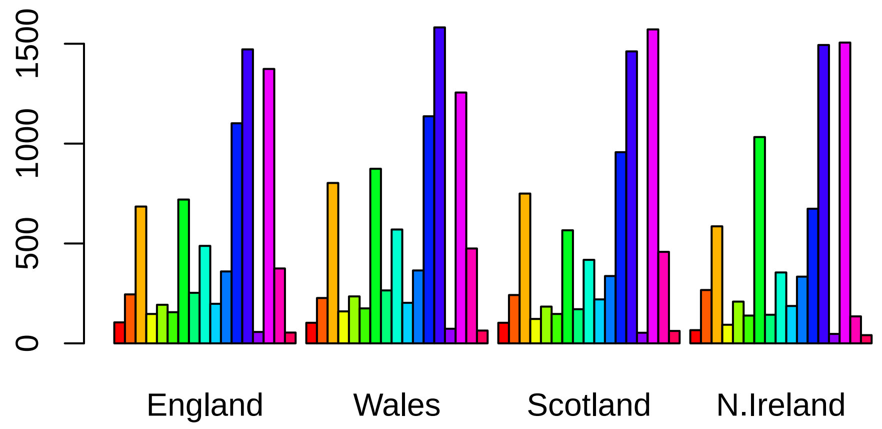
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars  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

If the method using `x <- x[,-1]` was utilized, and the same cell was run multiple times, it would keep removing rows, which would be problematic, when rendering the document, or when checking your code by rerunning all of your code.

Can visualize food consumption with respect to each country

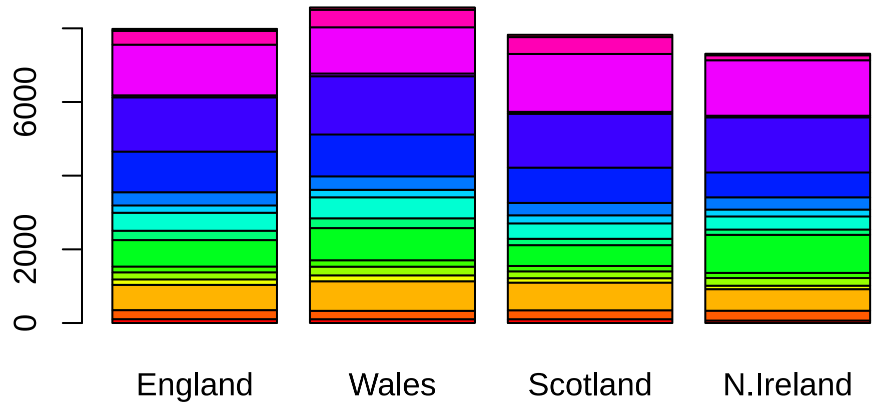
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

This barplot results in not transposing the data frame, which results in a much worse visualization of the data, since it is much harder to see differences between the variables!

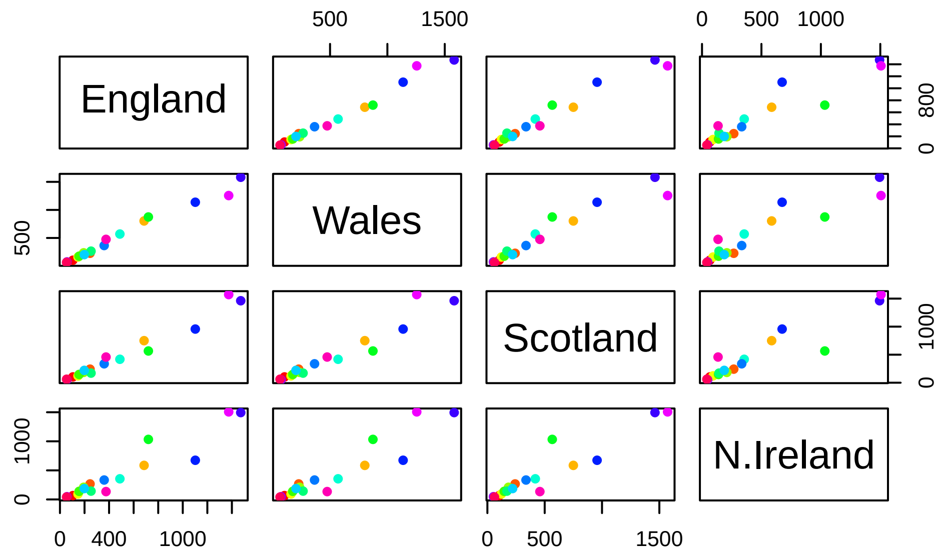
```
# even worse at visualization compared to the other type of barplot
# unable to visualize the difference between variables
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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?

Pairwise plots compare the different consumption of food type between two different countries, and do it for all of the plots showing the relationship between consumption among each of the countries. For the first row, england is on the y axis for the plots on the first row, the x-axis for each row changes based on country, which allows visualization of consumption between them. If points fall on the same line, it shows similar consumption between the two countries.

```
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?

Looking at the green point among England and Ireland, you can see that there is more consumption in Ireland vs England since it is further right on the X-axis, and there is less consumption in Ireland in comparison to England of the blue point since it is further left on the X-axis

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

## PCA to the rescue!

The main function for PCA in base R is called `prcomp()`. This function will want the food categories to be in the columns, not in the rows, needs us to transpose our input data -i.e. the food categories in columns and the countries as the rows.

```r
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
```

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

```
attributes(pca)
```

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

$class
[1] "prcomp"
```

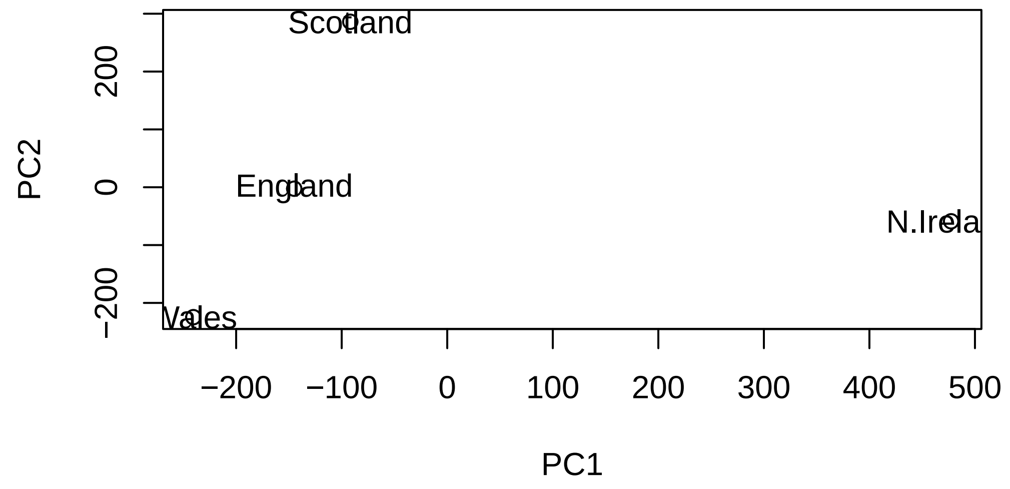
The `pca$x` result object is where we will focus first as this details how the countries are related to each other in terms of our new "axis" (a.k.a. the "PCs", "eigenvectors", etc).

```
head(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.

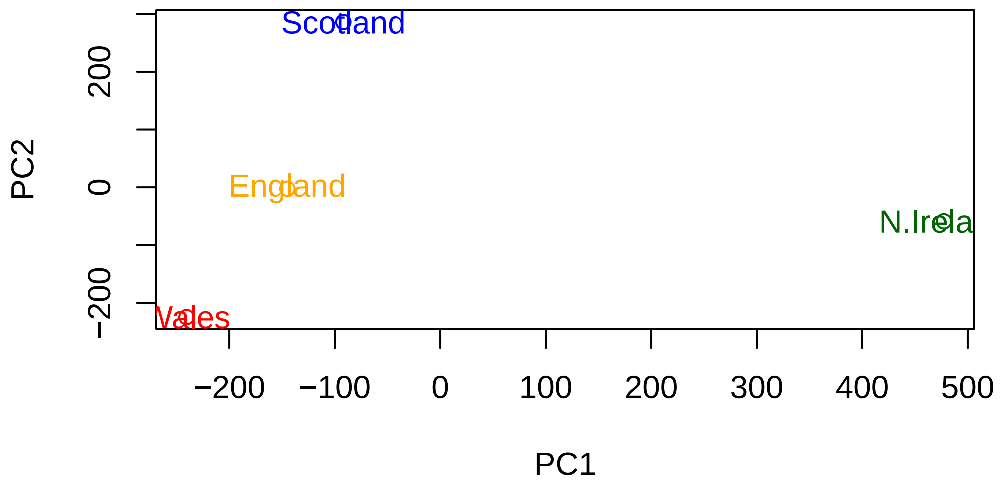
```
plot(pca$x[,1], pca$x[,2],
     xlab = 'PC1', ylab = 'PC2')

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.

We will now be plotting PC1 vs PC2

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

text(pca$x[,1], pca$x[,2], colnames(x), col = c("orange", "red", "blue", "darkgreen"))
```

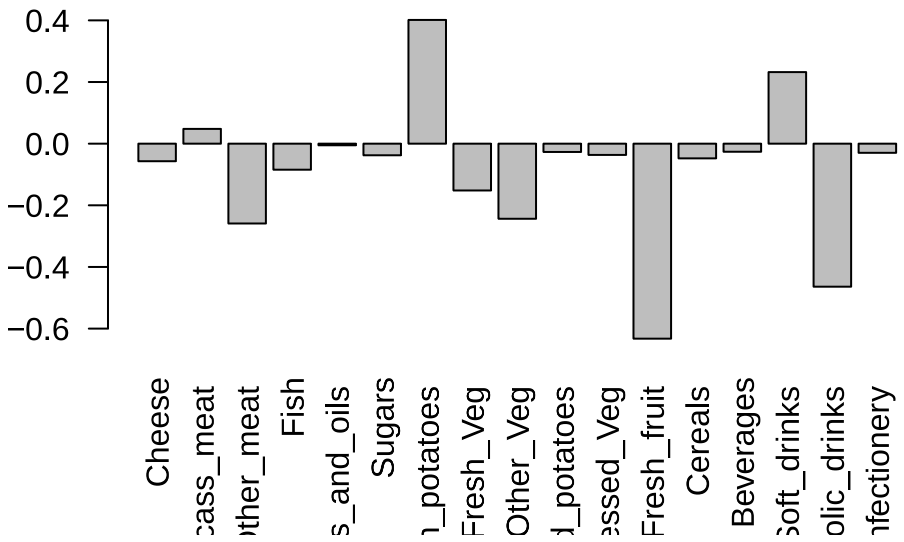


We can look at so called PC loadings result object to see how the original foods contribute to our new PCs (i.e. how the original variables contribute to our new better variables)

```
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 
```

```
barplot( pca$rotation[,1], las=2 )
```

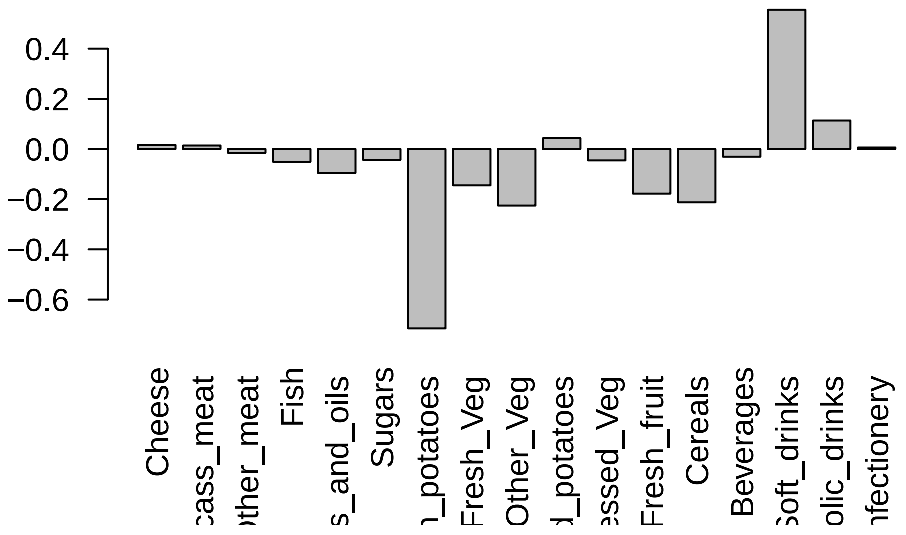


Q9: Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

```
pca$rotation[,2]
```

```
          Cheese        Carcass_meat          Other_meat                Fish 
     0.016012850         0.013915823        -0.015331138        -0.050754947 
  Fats_and_oils              Sugars      Fresh_potatoes           Fresh_Veg 
    -0.095388656        -0.043021699        -0.715017078        -0.144900268 
       Other_Veg  Processed_potatoes       Processed_Veg         Fresh_fruit 
    -0.225450923         0.042850761        -0.045451802        -0.177740743 
         Cereals           Beverages         Soft_drinks     Alcoholic_drinks 
    -0.212599678        -0.030560542         0.555124311         0.113536523 
   Confectionery 
     0.005949921 
```

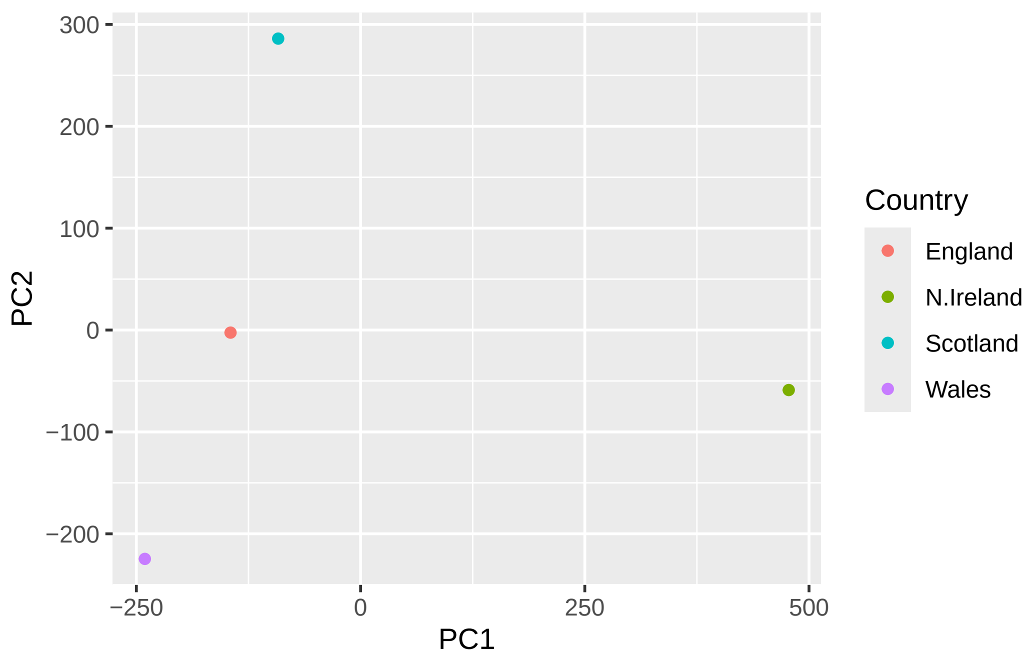
```
barplot( pca$rotation[,2], las=2 )
```



PC2 shows tells us how far up or down the plot the country will be. With a higher positive score, i.e. soft drinks causes Scotland to be further up on the graph. A higher negative score on PC2 shows that the country will be further down on the graph, for example a lower consumption in potatoes causes the country like Whales to be closer to the bottom of the graph.

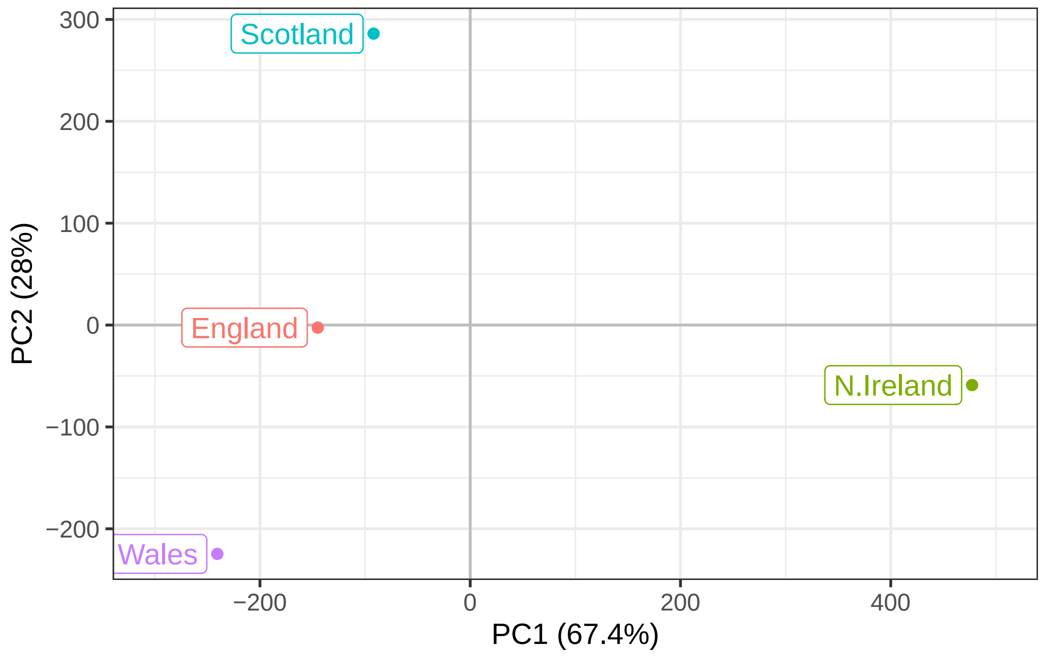
Using ggplot for the figures:

```
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()
```



making a nicer plot!

```
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()
```



**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
```

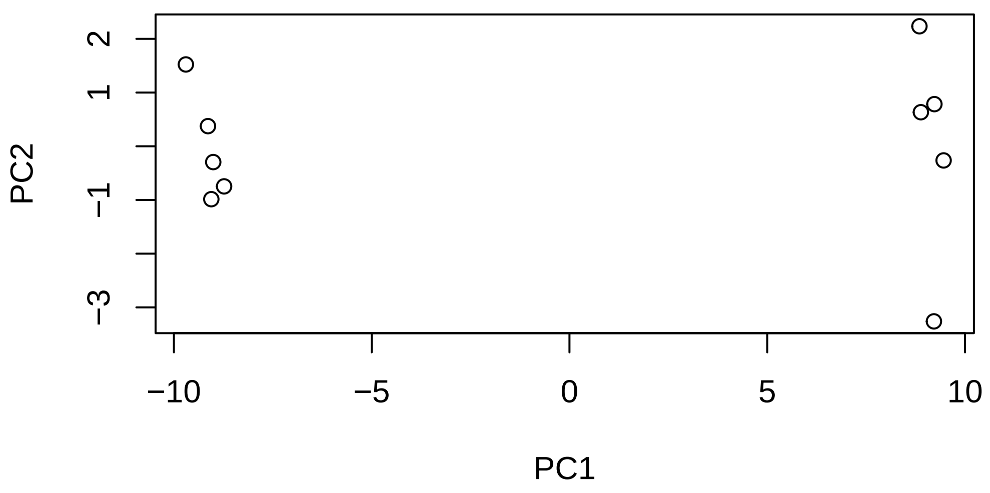
Q10: How many genes and samples are in this data set?

We can determine the number of genes using the `dim()` function which outputs the number of rows and columns, respectively. The number of rows corresponds to the number of genes, which are 100 in this dataset, and the number of columns corresponds to the number of samples which is 10 in this dataset.

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

```
[1] 100  10
```

```
## Again we have to take the transpose of our data
pca1 <- prcomp(t(rna.data), scale=TRUE)

## Simple un polished plot of pc1 and pc2
plot(pca1$x[,1], pca1$x[,2], xlab="PC1", ylab="PC2")
```

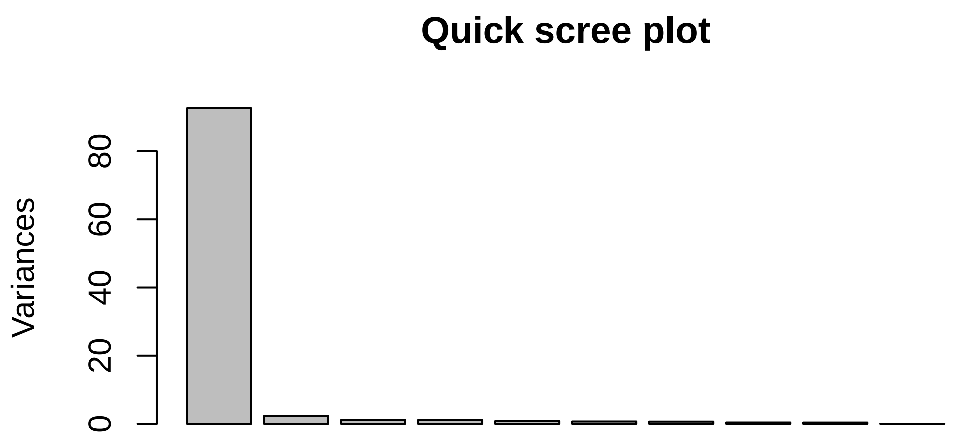


```
summary(pca1)
```

```
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 3.345e-15
Proportion of Variance 0.00385 0.00364 0.000e+00
Cumulative Proportion  0.99636 1.00000 1.000e+00
```

Make a barplot to quickly visualize this data

```r
plot(pca1, main="Quick scree plot")
```



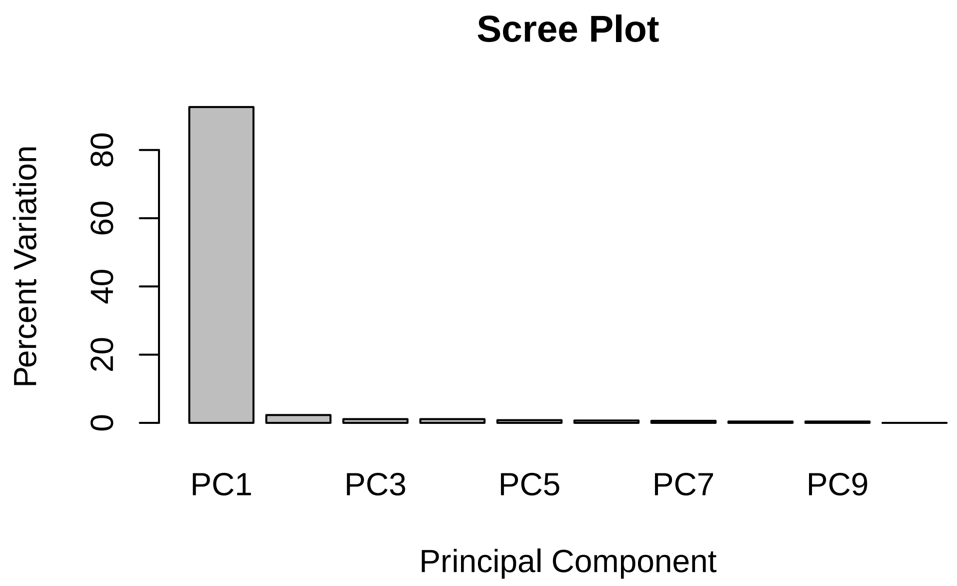
Based on this distribution it looks like there is some skew so we can account for this using `pca$stdev` which accounts for the standard deviation of the data!

```r
## Variance captured per PC
pca1.var <- pca1$sdev^2

## Percent variance is often more informative to look at
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
```

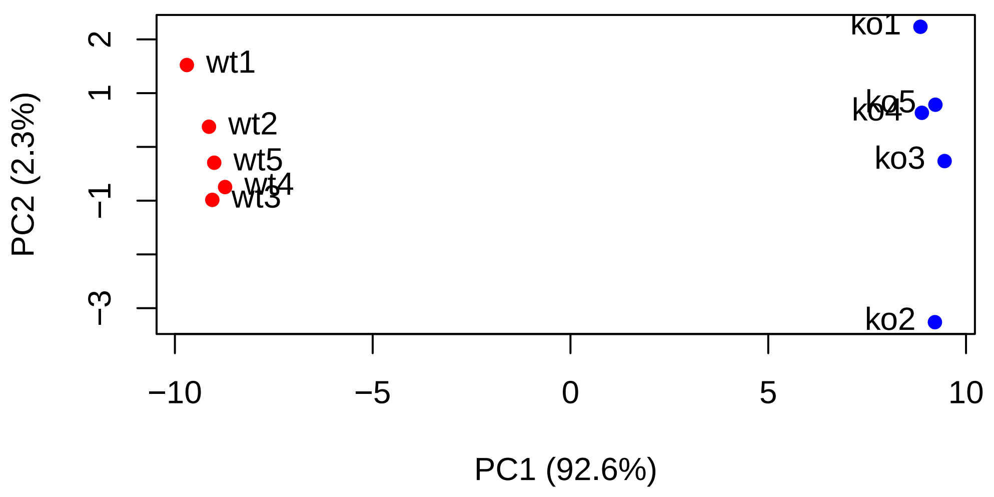
Using this information we can make another scree plot, to account for the variance!

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



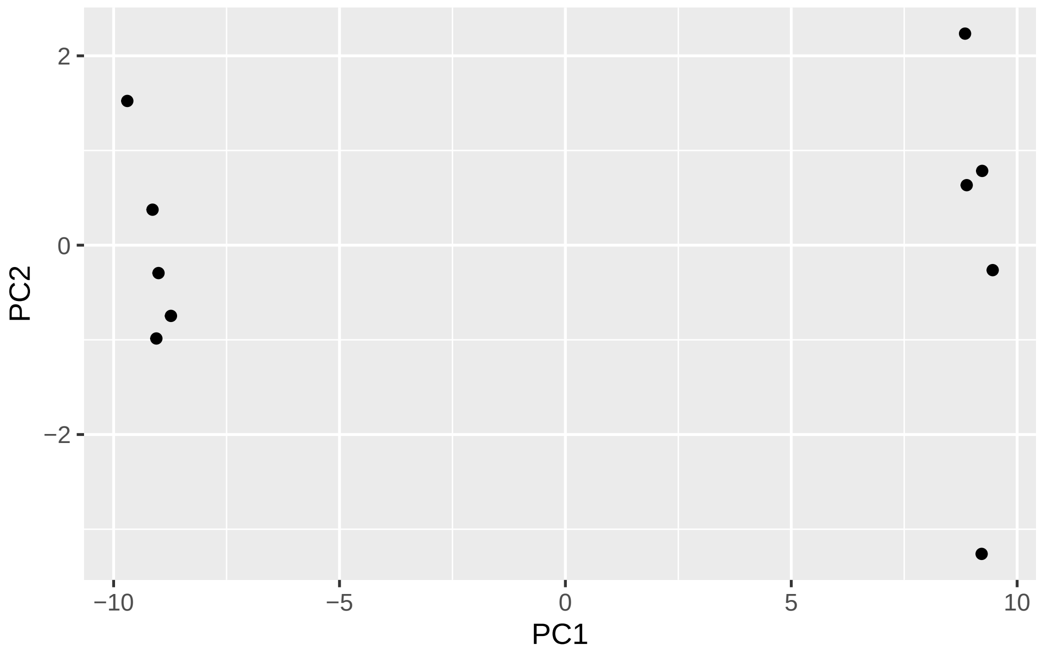
Making a more useful PCA plot:

```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

plot(pca1$x[,1], pca1$x[,2], col=colvec, pch=16,
     xlab=paste0("PC1 (", pca1.var.per[1], "%)"),
     ylab=paste0("PC2 (", pca1.var.per[2], "%)"))

text(pca1$x[,1], pca1$x[,2], labels = colnames(rna.data), pos=c(rep(4,5), rep(2,5)))
```

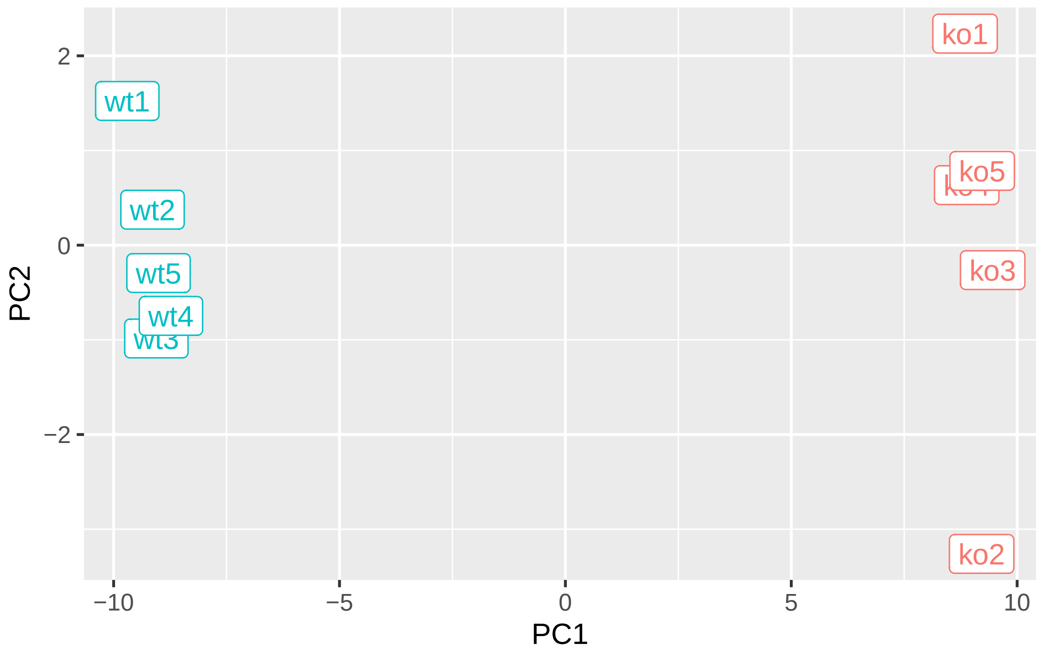


We can also use ggplot to make the figures even more presentable and attractive:

```
library(ggplot2)

df.rna <- as.data.frame(pca1$x)

# Our first basic plot
ggplot(df.rna) +
  aes(PC1, PC2) +
  geom_point()
```



```
# Add a 'wt' and 'ko' "condition" column
df.rna$samples <- colnames(rna.data)
df.rna$condition <- substr(colnames(rna.data),1,2)

plotrna <- ggplot(df.rna) +
        aes(PC1, PC2, label=samples, col=condition) +
        geom_label(show.legend = FALSE)
plotrna
```

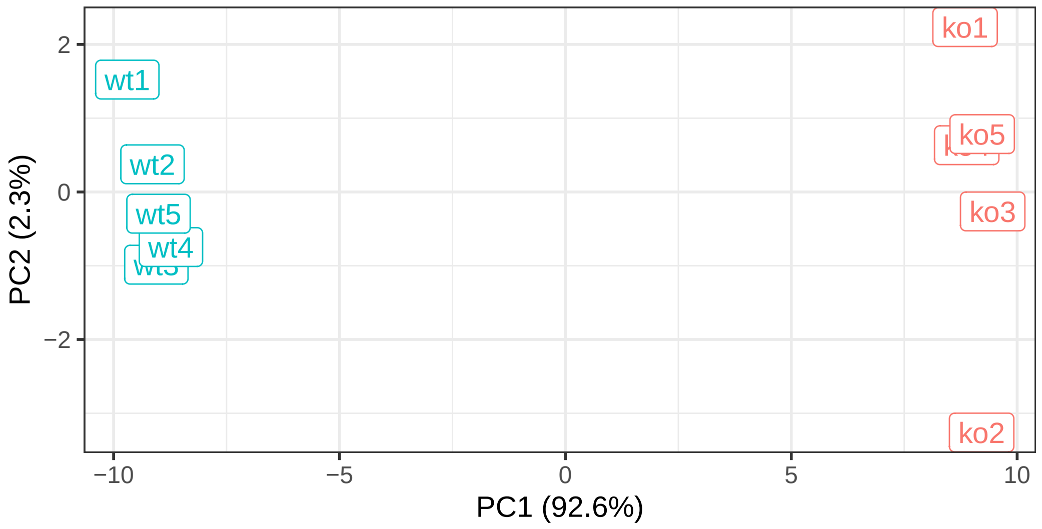


Lastly add final features to polish figure!

```
plotrna + labs(title="PCA of RNASeq Data",
       subtitle = "PC1 clealy seperates wild-type from knock-out samples",
       x=paste0("PC1 (", pca1.var.per[1], "%)"),
       y=paste0("PC2 (", pca1.var.per[2], "%)"),
       caption="Class example data") +
     theme_bw()
```

PCA of RNASeq Data

PC1 clealy seperates wild−type from knock−out samples



Class example data