Class 7: Machine Learning 1

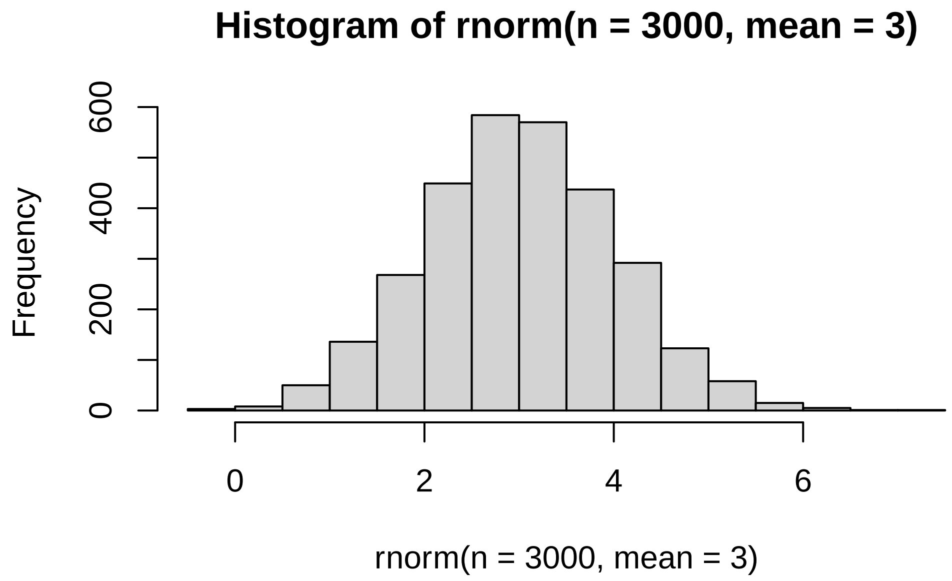
Amanda Wilpitz | A17463962

Today we will explore unsupervised machine learning methods including clustering and 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 the `rnorm()` function to help us here, which randomly generates data with normal distribution of mean and sd.

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

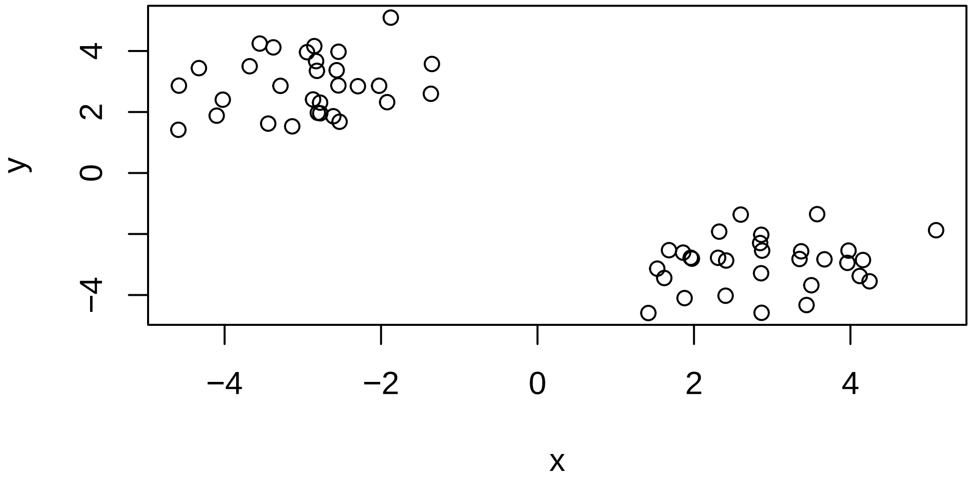


Now, make data `z` with two "clusters".

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

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

head(z)
```

```
             x        y
[1,] -1.875590 5.095042
[2,] -3.135288 1.530411
[3,] -2.819450 3.349256
[4,] -3.679415 3.500435
[5,] -3.376655 4.119039
[6,] -2.853156 4.160412
```

```r
plot(z)
```



K-means Clustering

The main function in "base" R for K-means clustering is called `kmeans()`. This will group an input data into input number of clusters that each center around its mean points.

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

k
```

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

Cluster means:
          x         y
1 -2.946903  2.890691
2  2.890691 -2.946903

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

Within cluster sum of squares by cluster:
[1] 47.09882 47.09882
 (between_SS / total_SS =  91.6 %)

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 (ie. which points lies in which cluster)?

```
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
```

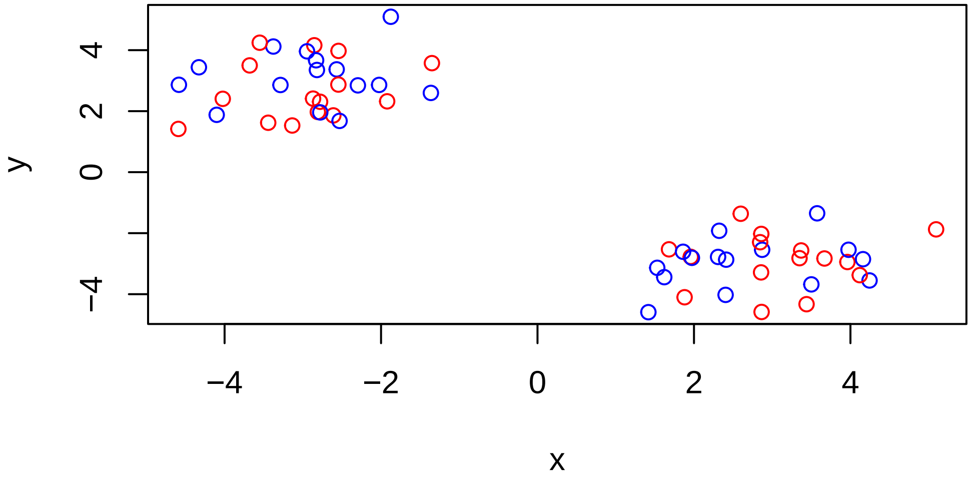
Q. Center of each cluster?

```
k$center
```

```
          x         y
1 -2.946903  2.890691
2  2.890691 -2.946903
```

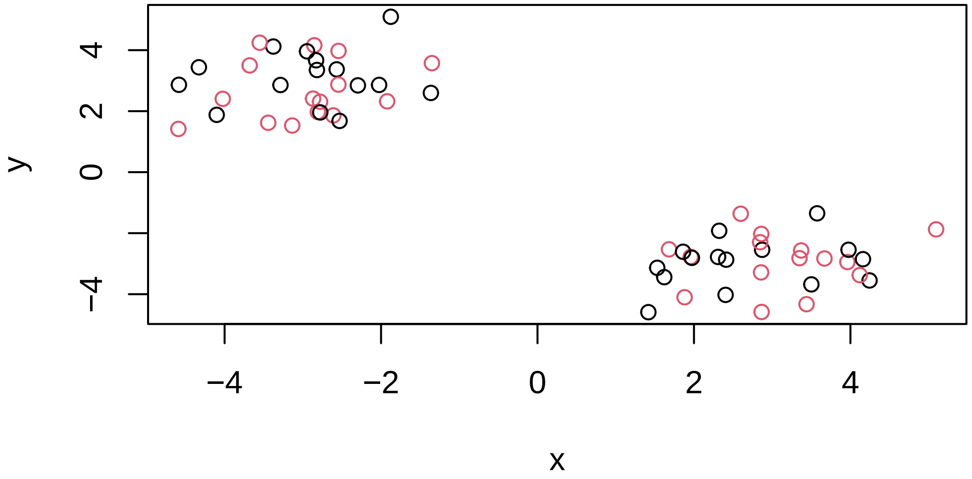
Q. Put this result info together and make a little "base R" plot of our clustering result. Also add the cluster center points to this plot.

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



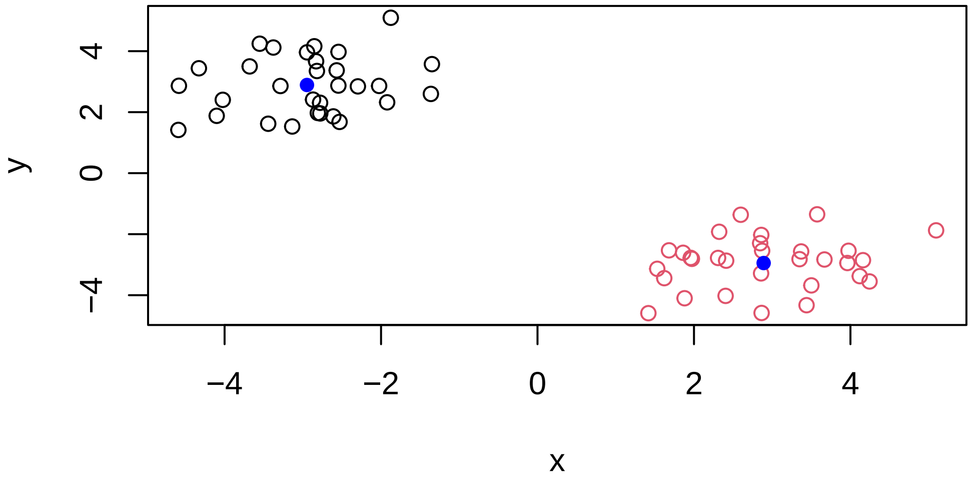
You can also color by number (1=black):

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



Color by membership/cluster:

```
plot(z, col=k$cluster)
points(k$centers, col= "blue", pch=16)
```



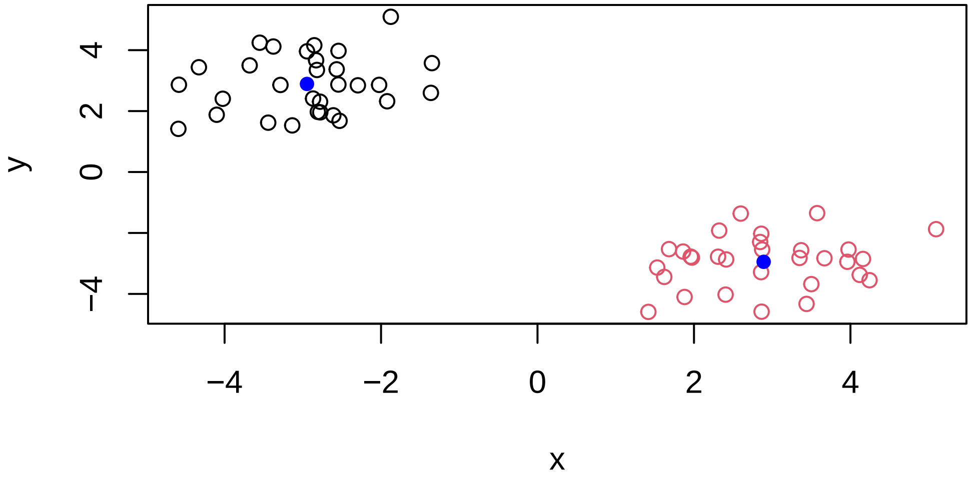
Q. Run k-means on our input z and define 4 clusters making the same results visualization plot as above (plot of z colored by cluster membership)

```
k4 <- kmeans(z, centers=4)
k4
```

```
K-means clustering with 4 clusters of sizes 14, 12, 16, 18

Cluster means:
          x         y
1  2.157563 -3.418860
2 -2.893370  3.871340
3  3.532177 -2.533941
4 -2.982592  2.236925

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

Within cluster sum of squares by cluster:
[1] 11.901815  9.782055 15.241262 18.025997
 (between_SS / total_SS =  95.1 %)
```

```
Available components:

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

```
plot(z, col=k$cluster)
points(k$centers, col = "blue", pch=16)
```



Better plot clustering will have a smaller tot.withinss value

```
k$tot.withinss
```

```
[1] 94.19764
```

```
k4$tot.withinss
```

```
[1] 54.95113
```

You can also plot a scree plot (x=number of clusters, y=tot.withinss) and "elbow"" in the plot will determine the number of clusters that you should have.

Hierarchical Clustering

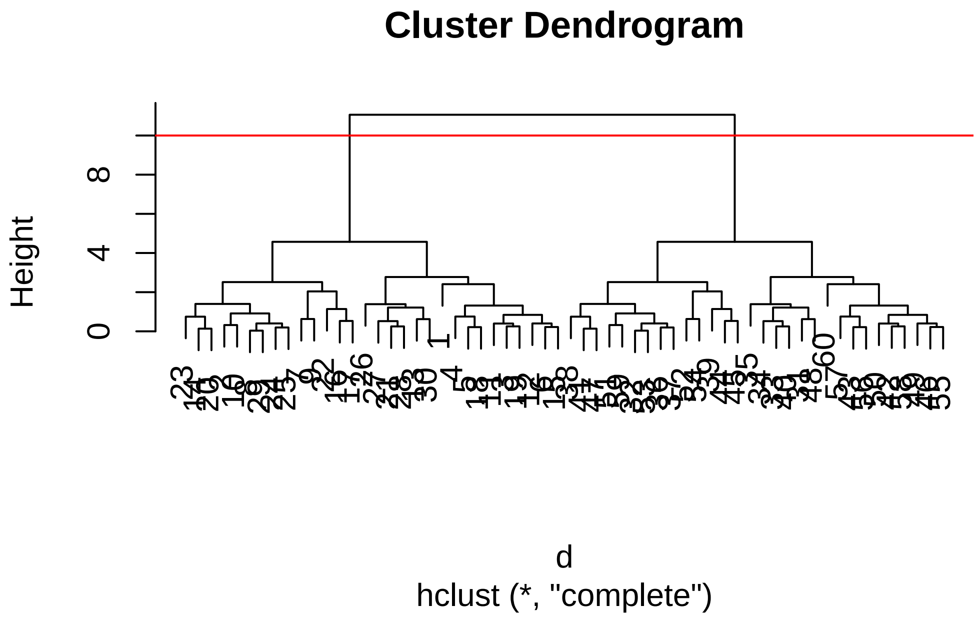
The main function in base R for this is called `hclust()`. It will take as input a distance matrix (you cannot just give your "raw" data as input - you have to first calculate a distance matrix from your data).

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

```

Call:
hclust(d = d)

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

Plot hclust, which will produce a hierarchical tree of the input values. All lower numbers <30 are on the left side and higher numbers 31-60 are on the right side. Each point starts as it's own cluster and the closest points are grouped together until there is one cluster left as the highest branch in the hierarchy. Higher branches = points farther apart.

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

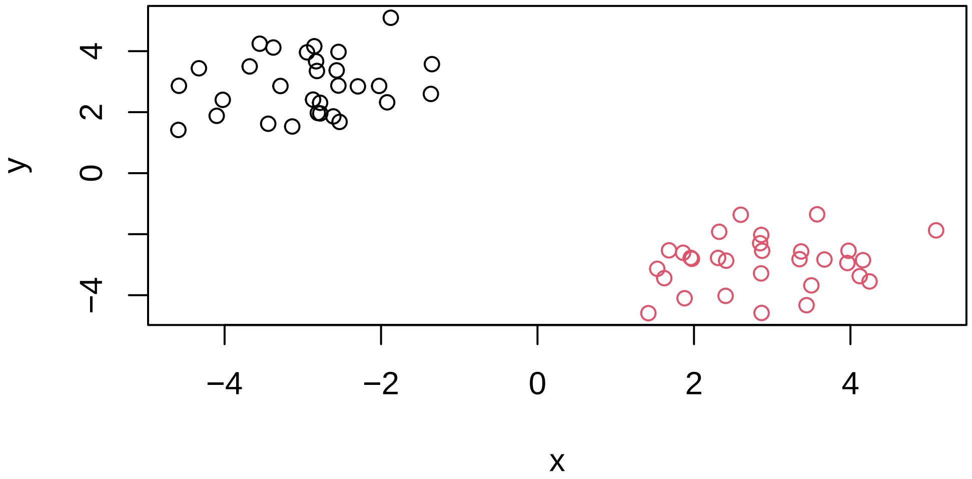


Once I inspect the "tree", I can "cut" the tree at a certain height to yield my groupings or clusters. THe function to do this is called `cutree()`.

```r
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 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
```

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



There are 4 methods to determine distance between clusters in hclust() - use trial and error.

Hands on with Principal Component Analysis (PCA): PCA w/ UK Food

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 to one another?

Data import

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

```
7      Fresh_potatoes      720   874      566      1033
8           Fresh_Veg      253   265      171       143
9           Other_Veg      488   570      418       355
10 Processed_potatoes      198   203      220       187
11      Processed_Veg      360   365      337       334
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
```

Checking your Data

> 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
nrow(x)
```

```
[1] 17
```

```r
ncol(x)
```

```
[1] 5
```

```r
dim(x)
```

```
[1] 17  5
```

Set `rownames()` to the first column and removes the first column with the -1 column index:

```r
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
```

Running this code multiple times will continuously remove the first column.

The code below will keep the first column (foods) as the row name, without deleting any other columns after multiple runs.

```
x <- read.csv(url, row.names=1)
head(x)
```

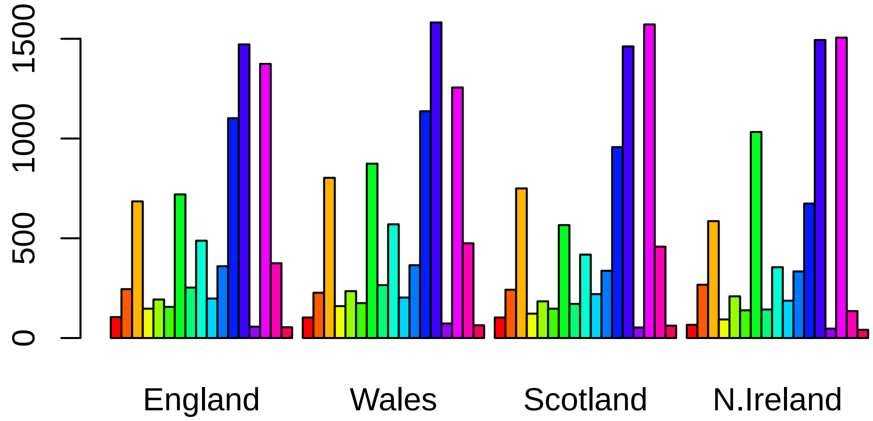
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars             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 one where you keep the first column as the row name, since it won't delete other columns especially when you are running the code again and again. This makes it more robust than removing the first column and setting it as the row names.

Spotting major differences and trends

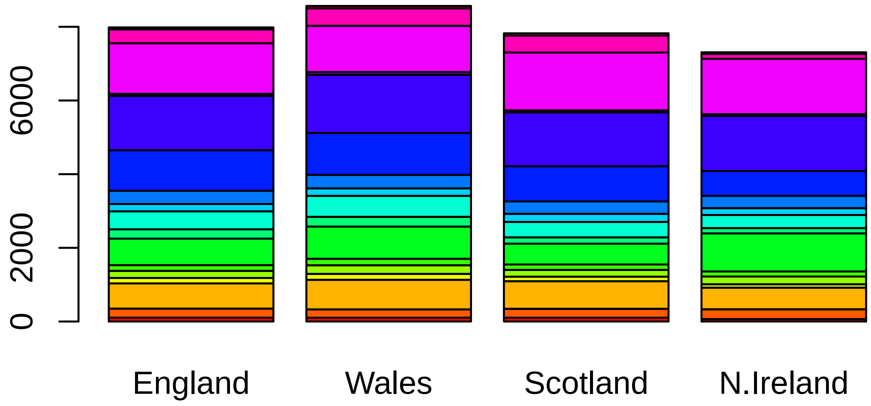
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

Changing beside to FALSE puts the bar plots on top of each other.

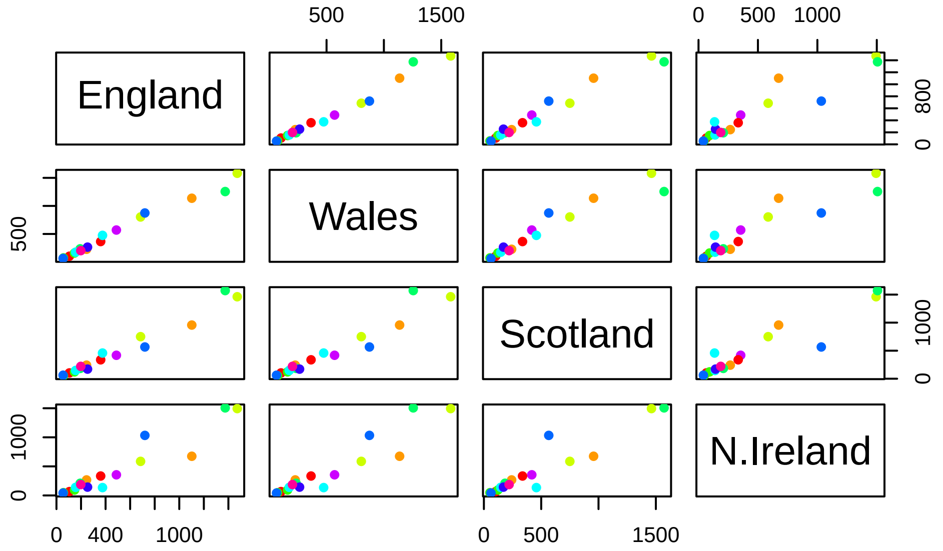
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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 label is the y-axis on its coresponding row and the x-axis in its corresponding column. For example, England is on the y-axis across the first row, Wales in on the x-axis on the second column. When two countries are similar, there are points on the diagonal. If the value is higher for the y-axis, the point is above the diagonal. If the value is higher for the x-axis, the points are below the diagonal.

```
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?

N. Ireland has the more differences from the other countries of the UK, since the column and row corresponding to N. Ireland has the most points off the diagonal.

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!

Principal component analysis (PCA) is a well established "multivariate statistical technique" used to reduce the dimensionality of a complex data set to a more manageable number (typically 2D or 3D). 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.

The main function for PCA in base R is called `prcomp()`. This function wants the transpose of our input data - ie. the important foods 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
```

Proportion of variance tells you percentage of data that is captured by that particular axis. PC1 has the highest variance. Cumulative proportion gives proportion of data that is captured by all the axis so far (ie. 96.5% captured by 2 axes)

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

```r
attributes(pca)
```

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

$class
[1] "prcomp"
```

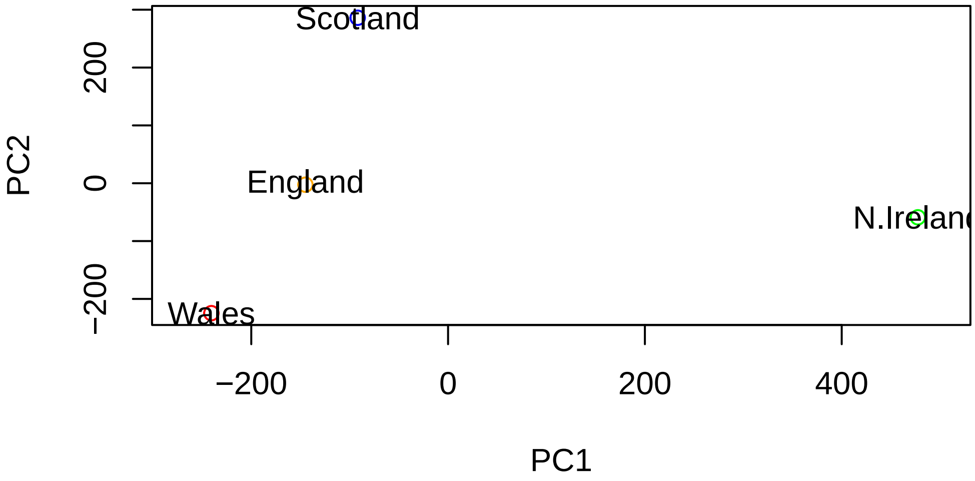
The `pca$x` result object is where we will focus first as this details how the countries are related to each other in terms of our new "axis" (aka "PCs", "eigenvectors", etc.)

```r
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
```

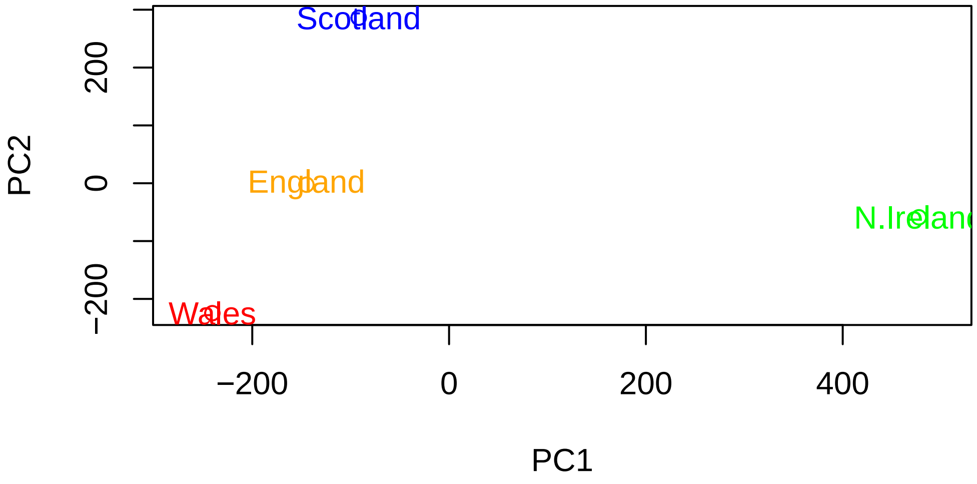
> Q7. Complete the code below to generate a plot of PC1 vs PC2. The second line adds text labels over the data points.

```r
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], col=c("orange", "red", "blue", "green"),
     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], col=c("orange", "red", "blue", "green"),
     xlab="PC1", ylab="PC2", xlim=c(-270,500))
text(pca$x[,1], pca$x[,2], colnames(x), col=c("orange", "red", "blue", "green"))
```



Digging deeper (variable loadings)

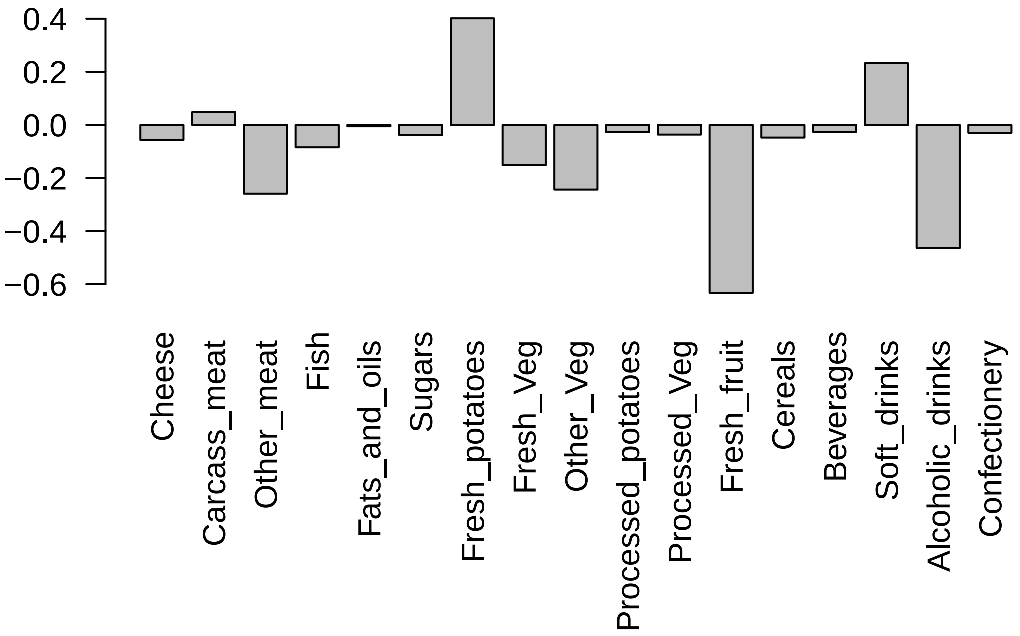
We can look at the so-called PC "loadings" result object to see how the original foods contribute to our new PCs (ie. how the original variables contribute to our new better PC variables). A positive loading is a positive correlation, while a negative loading is a negative correlation between food and a particular PC.

```
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 
```

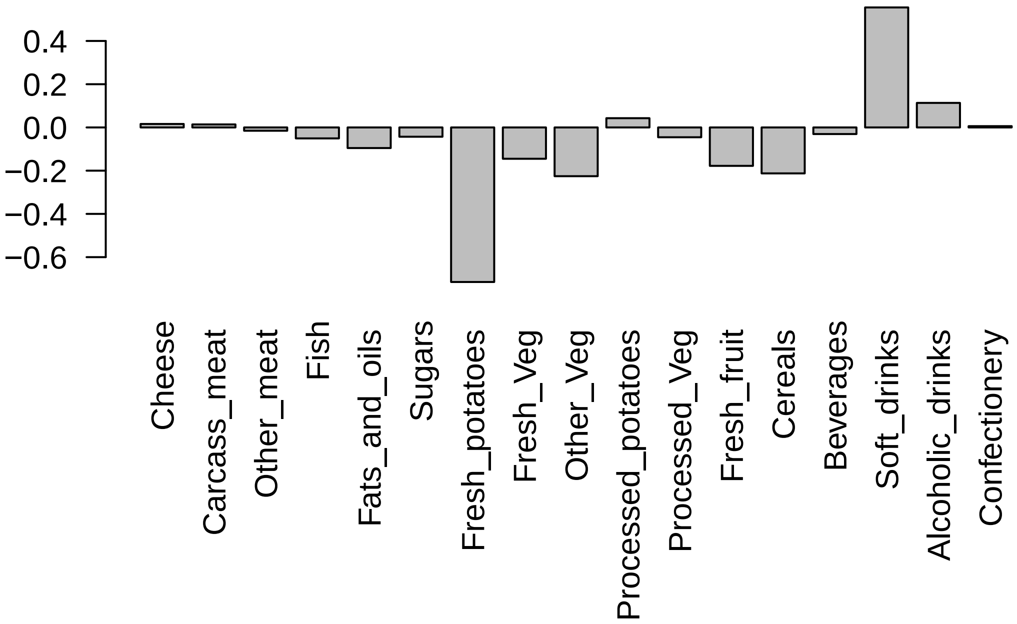
Plot a bar plot representing PC1

```
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?

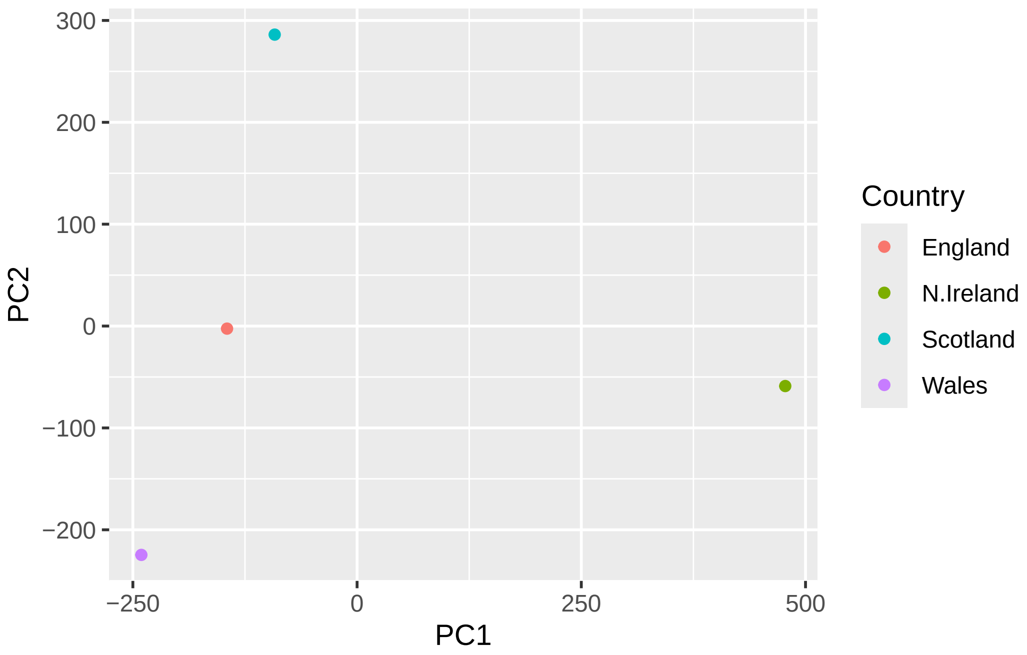
```
par(mar=c(10, 3, 0.35, 0))
barplot(pca$rotation[,2], las=2)
```



Soft drinks and fresh potatoes predominate. PC2 mainly tells us the contribution of each food category along the PC2 axis, with soft drinks having a positive correlation and fresh potatoes having a negative correlation.

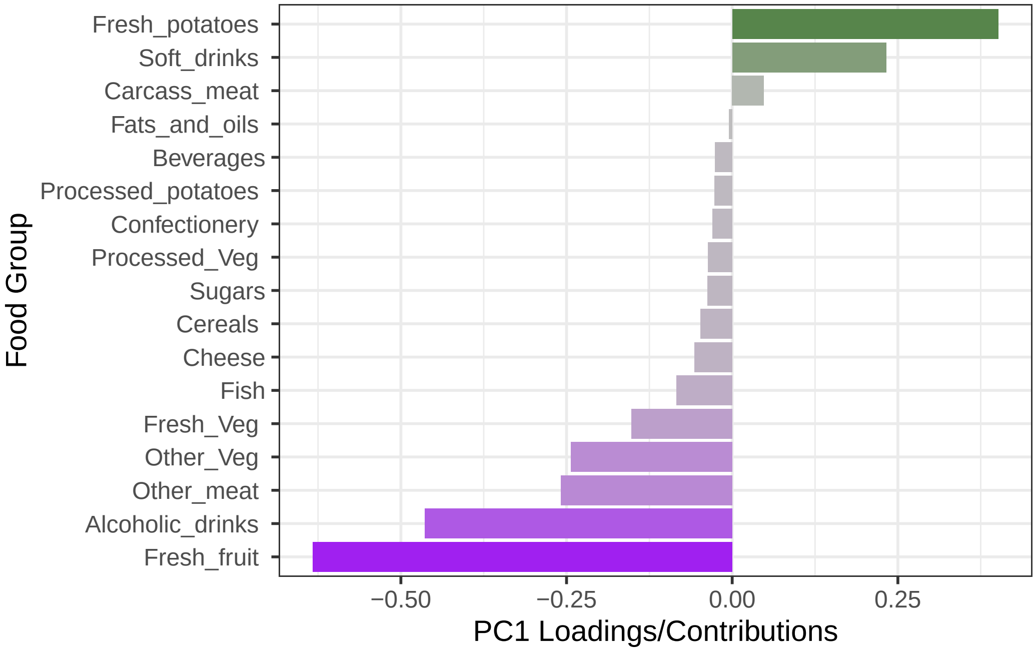
Using ggplot for these figures

```
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()
```



To plot our loadings plot with ggplot, we will convert it to a data frame and add row names as a new column called "Food":

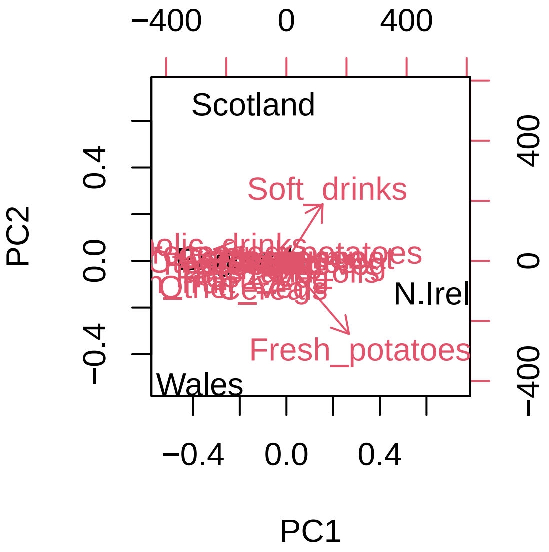
```
ld <- as.data.frame(pca$rotation)
ld_lab <- tibble::rownames_to_column(ld, "Food")

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

Another way to visualize PCA information is in a biplot. The data is organized in a central group of foods around the middle of each PC, with some on the periphery. Points closer to each other are more similar and longer arrows pointing towards/away the PC axis mean the variable contributes more to the PC.

```
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
```

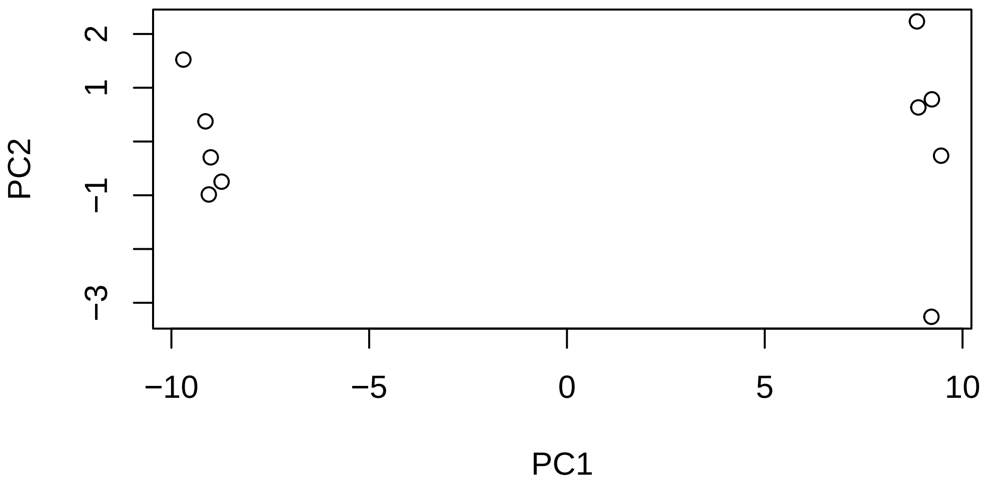
Q10: How many genes and samples are in this data set?

```
nrow(rna.data)
```

```
[1] 100
```

Let's make a PCA and plot the results:

```r
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")
```

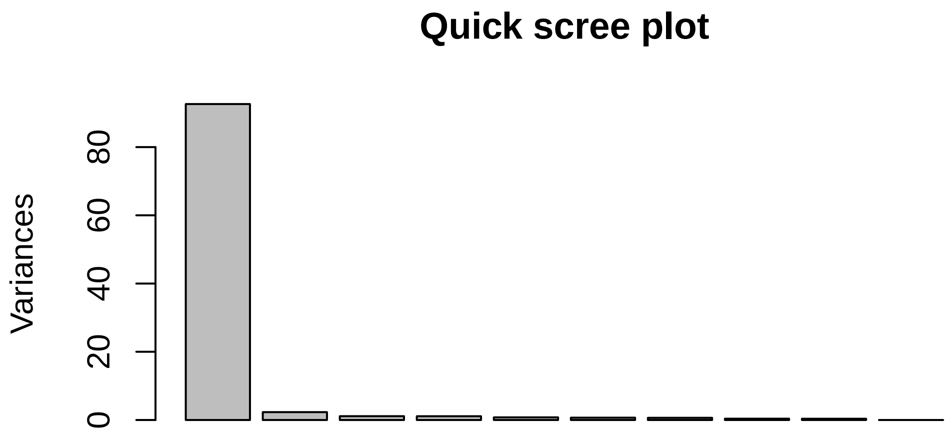


```r
summary(pca)
```

```
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
```

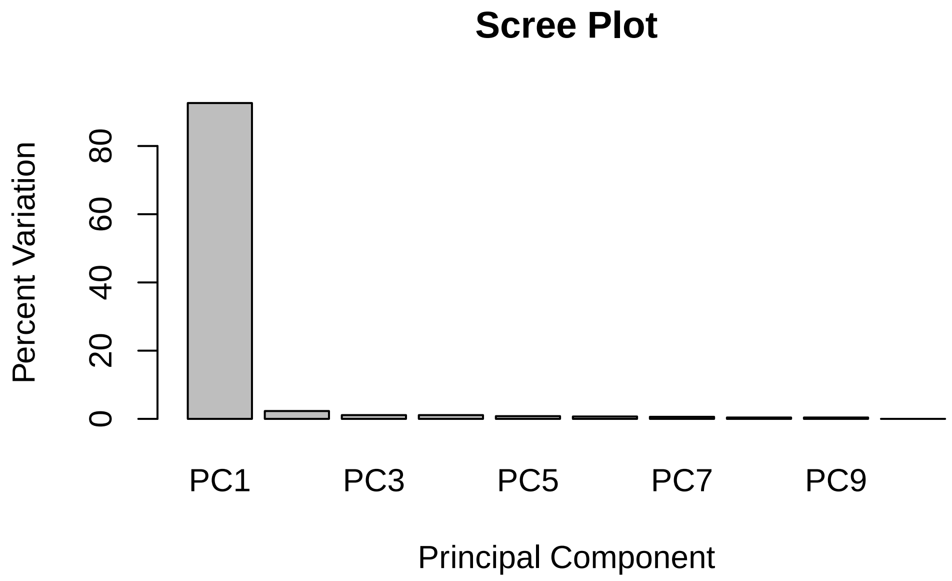
Quick plot of proportion of variance for each PC:

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



Let’s make our own scree plot:

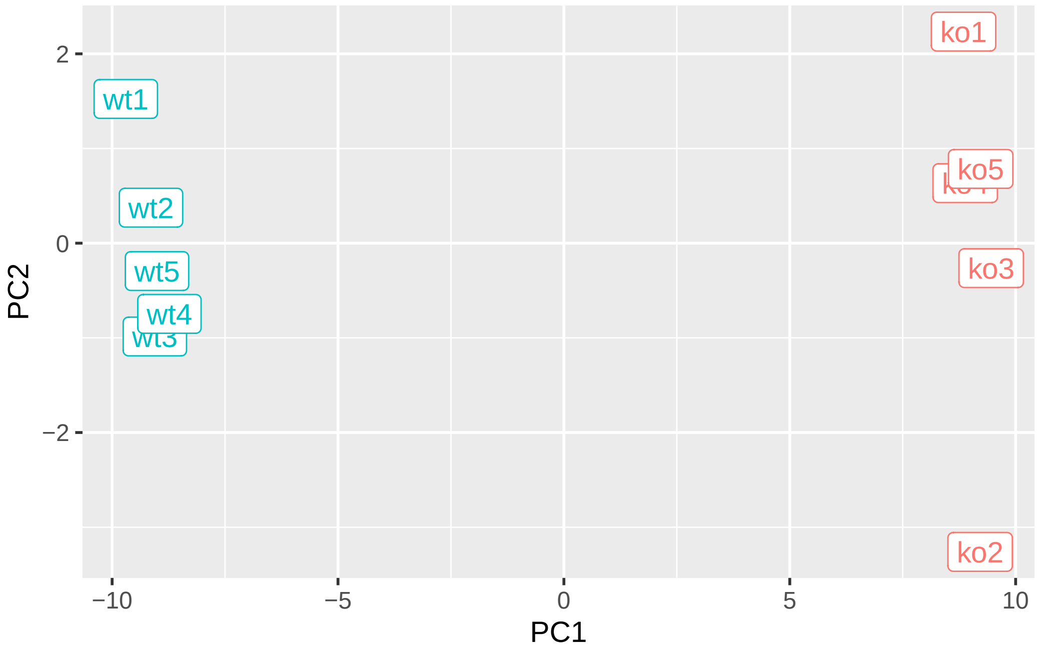
```
## Variance captured per PC
pca.var <- pca$sdev^2

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

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



We can see from the summary and the Scree plots that PC1 is where all the action is (92.6%).

Using ggplot to plot our RNA-seq data. Again, we must convert PCA to a data frame first:

```r
library(ggplot2)

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

df$samples <- colnames(rna.data)
df$condition <- substr(colnames(rna.data),1,2)

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


ko1
wt1
ko5
ko4
wt2
ko3
wt5
wt4
wt3
ko2
2
0
−2
−10
−5
0
5
10
PC1
PC2