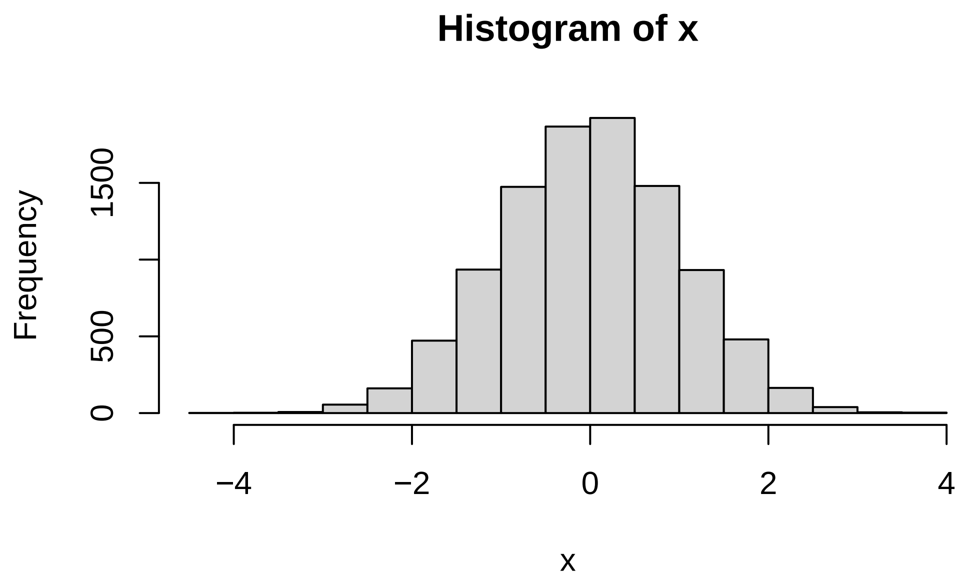
Class 07: Machine Learning 1

Nicholas Yousefi

K-means clustering

First we will test how this method works in R with some made up data.

```r
x <- rnorm(10000) # gives you 10 random numbers from a normal distribution
hist(x) # plot a histogram of the data
```



Let's make some numbers centered on -3, and some centered on +3

`cbind()`: puts vectors together into columns of a matrix (`rbind()` puts them into rows).

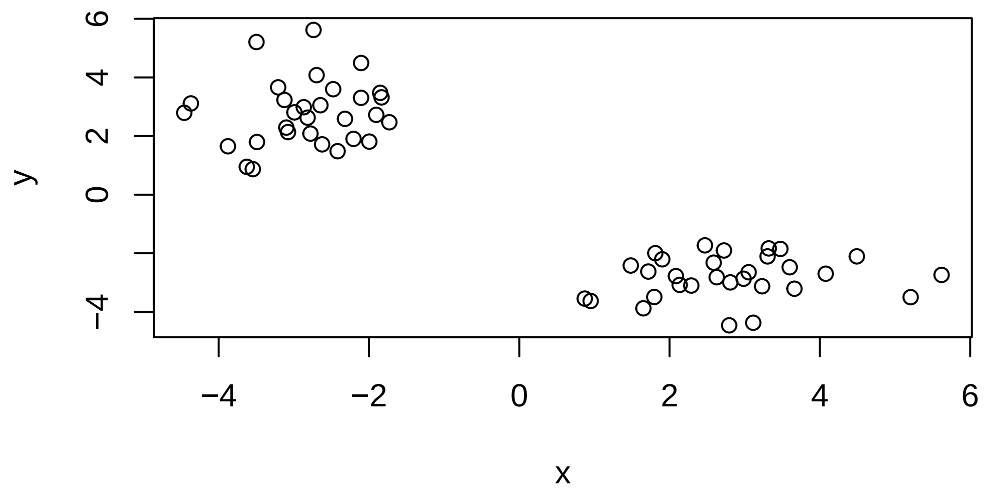
(The `rev()` function:

```r
rev(c("a", "b", "c"))
```

```
[1] "c" "b" "a"
```

)

```r
tmp <- c(rnorm(30, -3), rnorm(30, 3)) # see documentation; second argument is mean

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

plot(x)
```



Now, let's see how `kmeans()` works with this data...

```r
km <- kmeans(x, centers = 2, nstart = 20)
km
```

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

Cluster means:
```

```
          x         y
1 -2.816206  2.794710
2  2.794710 -2.816206

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] 51.9035 51.9035
 (between_SS / total_SS =  90.1 %)

Available components:

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

To get results out of `km` object, just do `km$available_component`.

```
km$centers
```

```
          x         y
1 -2.816206  2.794710
2  2.794710 -2.816206
```

> Q. How many points are in each cluster?

```
km$size
```

```
[1] 30 30
```

> Q. What 'component of your result object details - cluster assignment/membership?
> - cluster center?

```
km$cluster # cluster assignment/membership
```

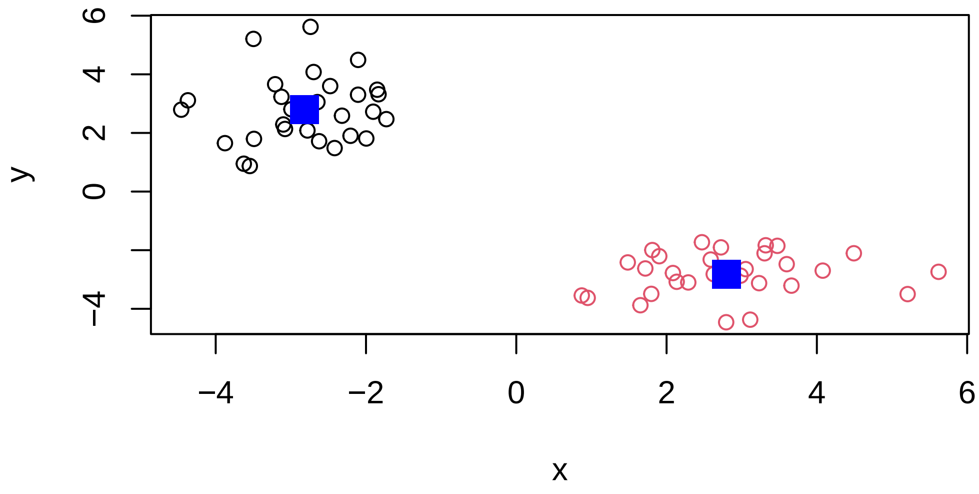
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
km$centers # cluster centers
```

```
          x         y
1 -2.816206  2.794710
2  2.794710 -2.816206
```

Q. Plot x colored by the kmeans cluster assignment and add cluster centers as blue points

```
plot(x, col=km$cluster) # plot of input data

# you can give col a vector of integers of the same length as the data, x, and it will col

points(km$centers, col="blue", pch=15, cex=2) # pch sets the plotting character (each char
```



Hierarchical Clustering

The `hclust()` function in R performs hierarchical clustering.

You can't just put in the data to `hclust()`. You need to give it a distance matrix, produced by `dist()`.

A distance matrix is a 60x60 matrix (in our case since we have 2 sets of 30 points). Inside it is the distance from the first point to every single other point in the data set. It is symmetrical.

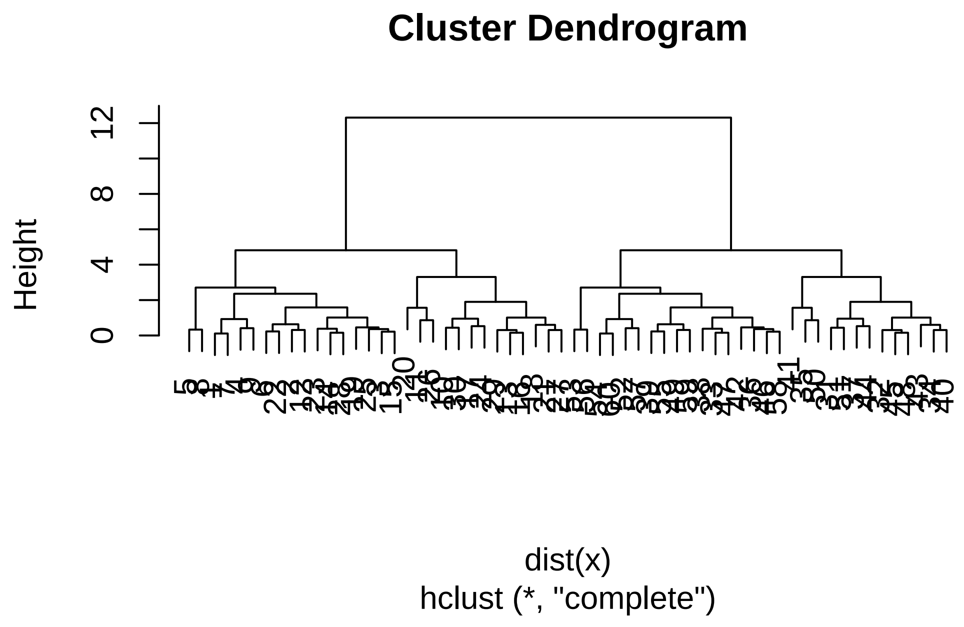
You can use different distance methods to calculate the distance between points. The default is Eucledian distance.

```
hc <- hclust(dist(x))
hc
```

```
Call:
hclust(d = dist(x))

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

There is a `plot()` method for `hclust` objects...

```
plot(hc) # when you call the plot function on an hc object, it creates a special plot call
```



Now to get my cluster membership vector I need to "cut" the tree to yield separate "branches" with the leaves on each branch being our clusters. To do this, we use the `cutree()` function.

```r
cutree(hc, h=8) # returns the membership vector, saying which cluster each point is in
```

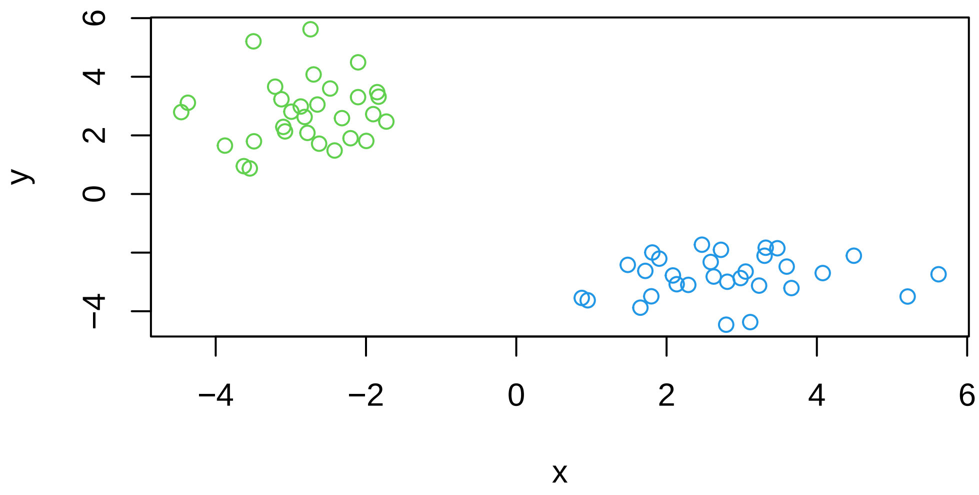
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Use `cutree()` with `k=2`. This finds the height to cut at so that you get 2 groups.

```r
grps <- cutree(hc, k=2)
```

Want to make a plot of our data colored by our hclust grps:

```r
plot(x, col=grps + 2)
```



Principal Component Analysis (PCA)

Import the Data:

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

```
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
```

> 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] 5
```

```
# note: could also use dim()
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
```

Uh oh! There should be 4 columns, not 5. Let's fix this:

```
rownames(x) <- x[,1] # set row names to the values in col 1 of x
x <- x[,-1] # replace x with all the columns in x except the first column
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
```

```r
dim(x)
```

```
[1] 17  4
```

There we go! That's better! Now, we have only 4 columns.

Alternatively, we could use an argument in the `read.csv()` function to set the first row of the CSV file to be the row names:

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

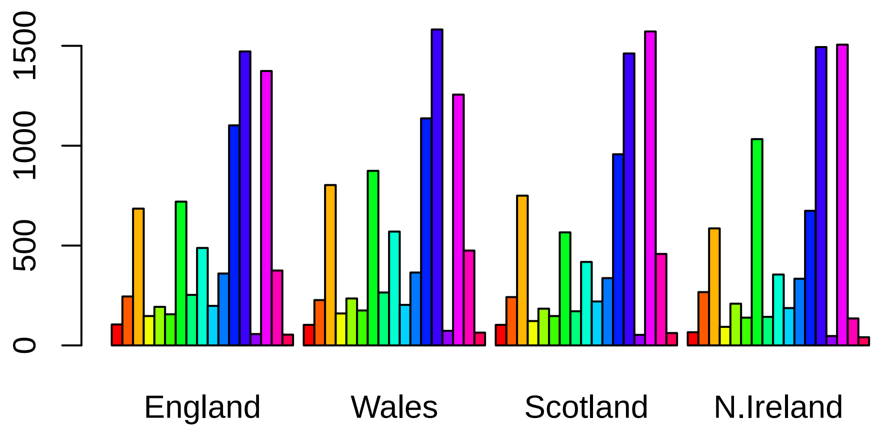
```
               England Wales Scotland N.Ireland
Cheese             105   103      103        66
Carcass_meat       245   227      242       267
Other_meat         685   803      750       586
Fish               147   160      122        93
Fats_and_oils      193   235      184       209
Sugars             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 `row.names` argument in `read.csv()`. If you run the other method multiple times, it will keep removing columns from x, and you may end up deleting the "England", "Wales", etc. columns if you run it multiple times by accident. The `row.names` version is more robust since if you run it multiple times, it doesn't keep deleting columns.

Let's try to visualize the data in a plot. Maybe that will be more useful…

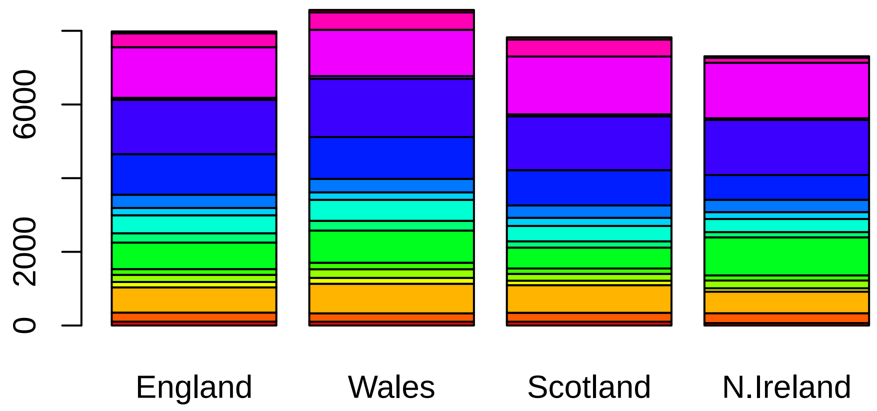
```r
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

To get the below plot, change the `beside` argument to `FALSE` (or just leave it out since it defaults to `FALSE`:

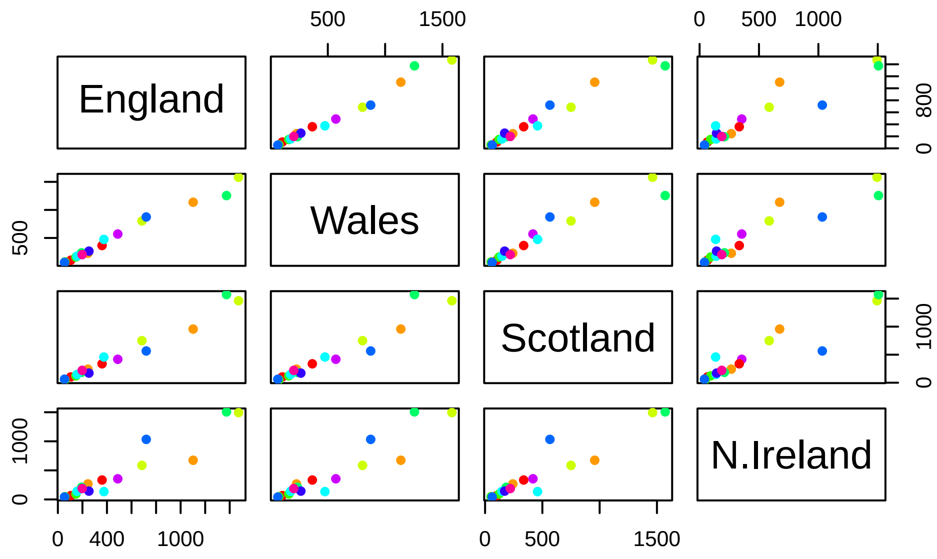
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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?

Pairs Plot:

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



```
# x is the data matrix, col sets the colors of the points (rainbow(10) gives a vector of 1
```

Take the plot in the upper right corner. It is England on the y-axis and N. Ireland on the x-axis. **For a given graph, to determine what is plotted on the y-axis, move to the left or right horizontally until you get to the name of the country. To determine the country plotted on the x-axis, move up or down vertically until you get to a country name.**

If, say, England and Wales both eat 20 units of a certain type of food (e.g. potatoes), that dot will be on the diagonal of the plot. The more points are on the diagonal, the more similar the two countries' eating habits are. **If a point lies on the diagonal, it means it is the same in both countries.**

Often, we divide, say, 20/20, so that anything that lies on the diagonal has a value of 1.

When we take:

$$log_2(\frac{20}{20}) = 0$$

we get 0. This means that the $\log_2$(fold change) is 0.

If we ate 10 food units in England and 20 in Wales, you have a $\log_2$(fold change) of:

$$log_2(\frac{10}{20}) = 0.5$$

Points above the line are consumed more in the country on the y-axis. Points below the line are consumed more in the country on the x-axis.

While this is kind of useful (i.e. not useful), it takes work to dig into the details here to find out what is different in these countries.

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

There seems to be a blue point in N. Ireland that is below the diagonal, indicating that people in N. Ireland eat less of it than the other countries.

PCA to the rescue:

Principal Component Analysis (PCA) can be a big help in these cases where we have lots of things that are being measured in a dataset (i.e. we have lots of dimensions; each column is a dimension).

The main PCA function in base R is called `prcomp()`.

This function wants the transpose of x (`t(x)`). That is, it wants England, Scotland, etc. to be in the rows, not the columns, and it wants the food items to be in the columns of our data matrix.

```r
pca <- prcomp(t(x))
summary(pca) # this function tells us how well we did (i.e. it tells us how much variance
```

```
Importance of components:
                            PC1      PC2      PC3       PC4
Standard deviation     324.1502 212.7478 73.87622 4.189e-14
Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```r
# in this case, PC1 captured about 67% of the spread of the data in one axis

# PC2 always captures less data than PC1 (it is designed that way)

#  Cumulative proportion is the sum of proportion of variance in all the PC's below this.
```

The above results shows that PCA captures 67% of the total variance in the original data in one PC and 96.5% in two PCs.

```
attributes(pca) # we are after x - it is what our data looks like in our new axis
```

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

$class
[1] "prcomp"
```

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

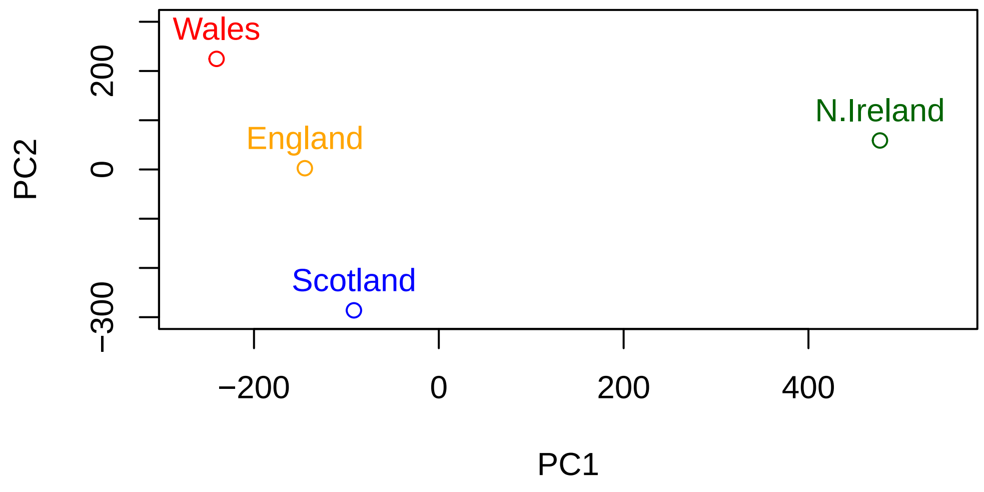
```
                 PC1         PC2         PC3           PC4
England   -144.99315    2.532999 -105.768945  2.842865e-14
Wales     -240.52915  224.646925   56.475555  7.804382e-13
Scotland   -91.86934 -286.081786   44.415495 -9.614462e-13
N.Ireland  477.39164   58.901862    4.877895  1.448078e-13
```

Let's plot our main results.

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

> 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.

```
colors = c("orange", "red", "blue", "darkgreen")
plot(pca$x[,1], pca$x[,2], col=colors, xlab="PC1", ylab="PC2", xlim=c(-270, 550), ylim=c(-
text(pca$x[,1], pca$x[,2], colnames(x), col=colors, pos=3)
```



The first 3 points are close together along PC1 (i.e. their x-values are close together). The 4th point is farther apart.

N. Ireland is pretty different along PC1.

Note: An alternative way to see how much variation in the original data each PC accounts for is to use the square of the standard deviation to calculate the percent variation:

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

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

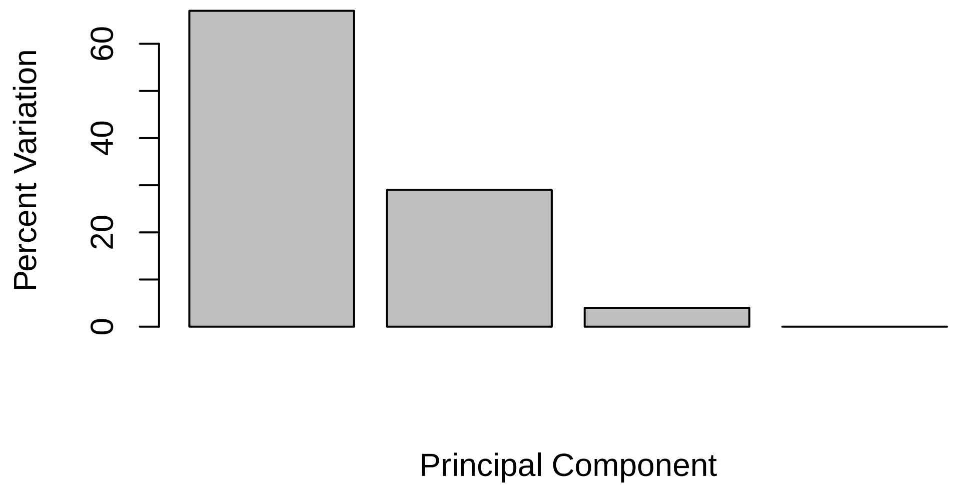
We can also use the second row of this table to get the same data:

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

```
                             PC1       PC2      PC3          PC4
Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

We can summarize this information in a bar plot of the % variation for each PC:

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



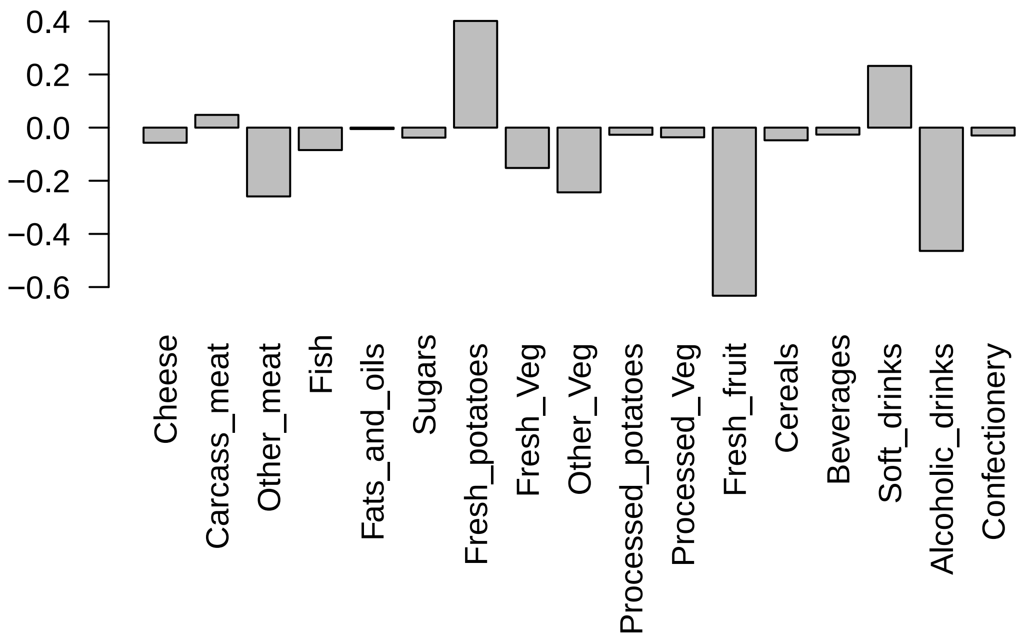
Variable Loadings

We want to know why N. Ireland is so different?

We can figure this out by looking at how the original variables in our data affect the principle components.

The loading scores tell us how much each original variable influences the PCs. They are returned in the `$rotation` component of the object returned by `prcomp()`. We can make a plot of the loading scores for PC1:

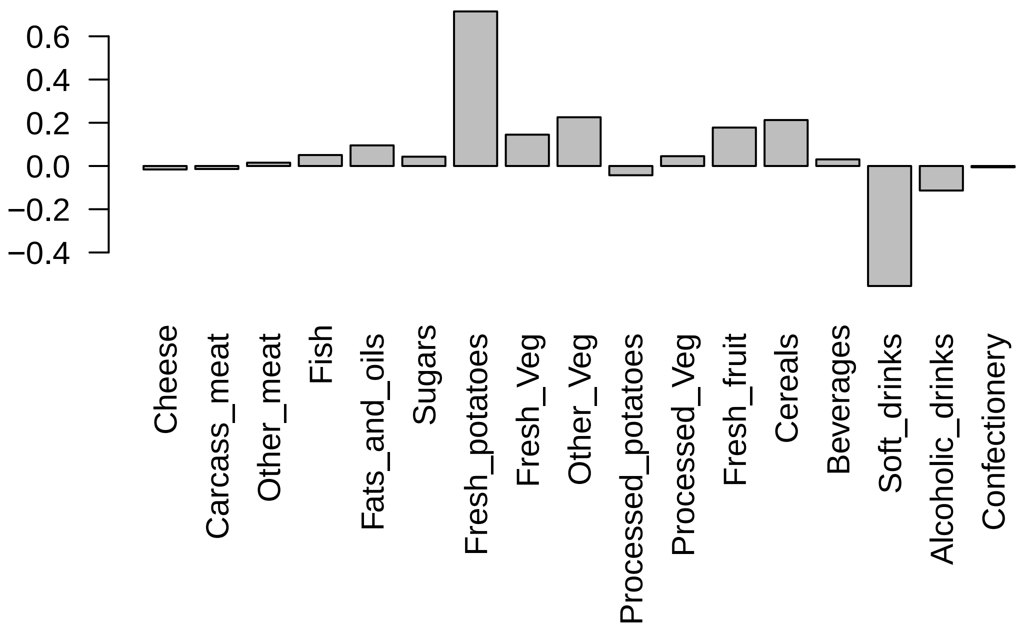
```r
par(mar=c(10, 3, 0.35, 0))
barplot(pca$rotation[,1], las=2)
```



The foods with the largest positive loading scores "push" N. Ireland to the right side of the plot (e.g. `Fresh_potatoes`, `Soft_drinks`). The foods with high negative scores push the other countries to the left side of the plot (e.g. `Fresh_fruit`, `Alcoholic_drinks`).

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

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

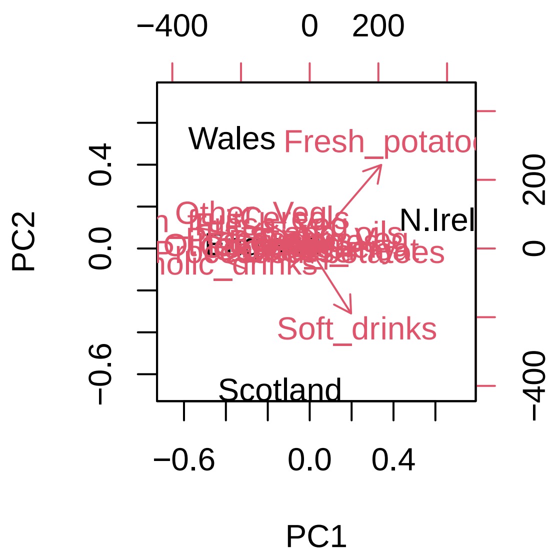


The 2 food groups that feature prominently are `Fresh_potatoes` and `Soft_drinks`. PC2 mainly tells us about the differences between Wales, England, and Scotland. It seems as though in Wales they like to eat a lot of fresh potatoes, whereas in Scotland they like to drink a lot of soft drinks.

Biplots

We can also make a biplot to see the different variables, as well as the main PCA plot using biplots:

```r
biplot(pca)
```



Recall earlier how we said that `Fresh_potatoes` and `Soft_drinks` push N. Ireland to the right and `Fresh_fruit` and `Alcoholic_drinks` push the other countries to the left. Notice how, here in this biplot, `Fresh_potatoes` and `Soft_drinks` are away from the big main cluster and are closer to N. Ireland. And `Fresh_fruit` and `Alcoholic_drinks` are closer to England, Scotland, and Wales. This plot shows which variables are associated with where different countries cluster on the plot.

PCA of RNA Seq Data

Let's import the RNA-seq data:

```r
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?

Number of genes:

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

```
[1] 100
```

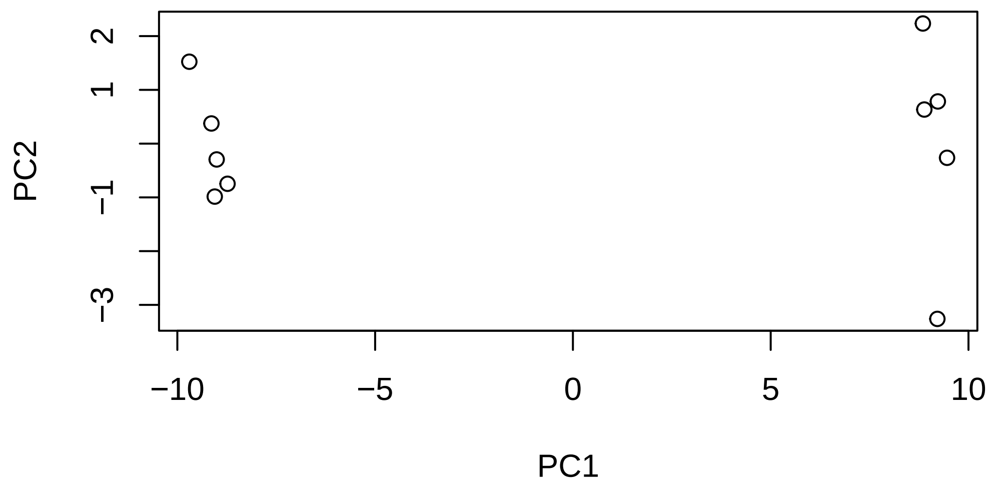
Number of samples:

```
ncol(rna.data)
```

```
[1] 10
```

Let's run PCA on this data!!!

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



Obviously, there are two separate groups, each with 5 samples (potentially representing our five Wild Types and five Knock-Outs). Let's see how much variance is accounted for by each PC:

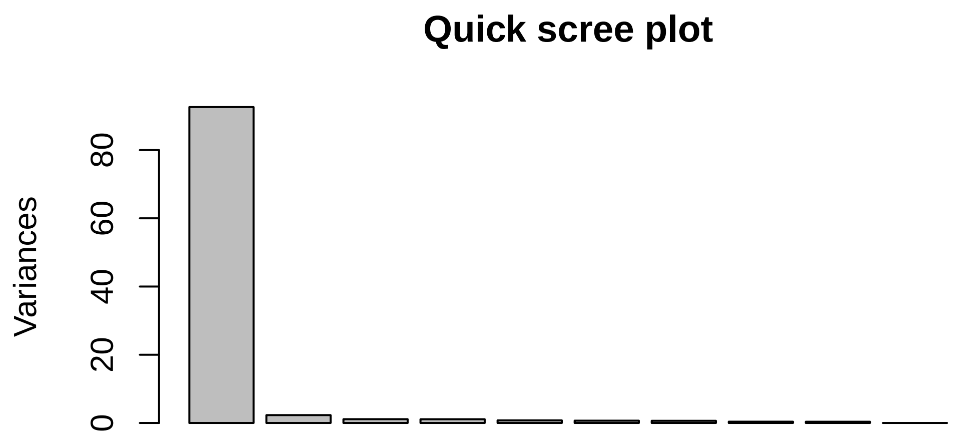
```r
summary(pca)
```

```
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 3.348e-15
Proportion of Variance 0.00385 0.00364 0.000e+00
Cumulative Proportion  0.99636 1.00000 1.000e+00
```

Wow! PC1 alone captured 92.6% of the data! That's almost all of it! We just reduced 100 dimentional data down to 1 dimension!

Let's create a scree plot of this data. We can do this by calling `plot()` on the object returned by `prcomp()`:

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



Alternatively, we can generate the scree plot manually:

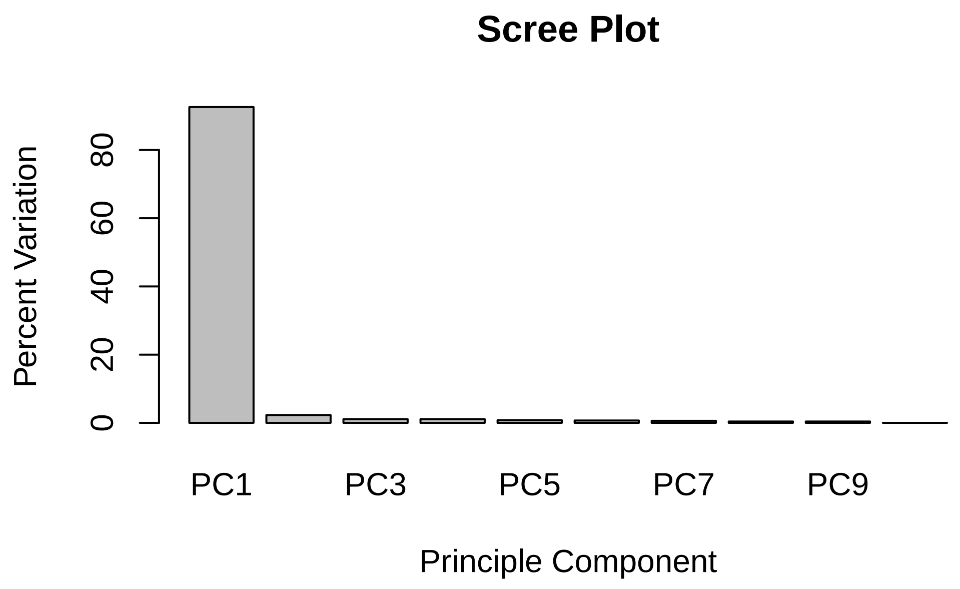
We can square `pca$dev` to calculate the variance of our data from the standard deviation. We can then calculate the percent variance.

```
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
```

Next, we generate a scree plot of this data as follows:

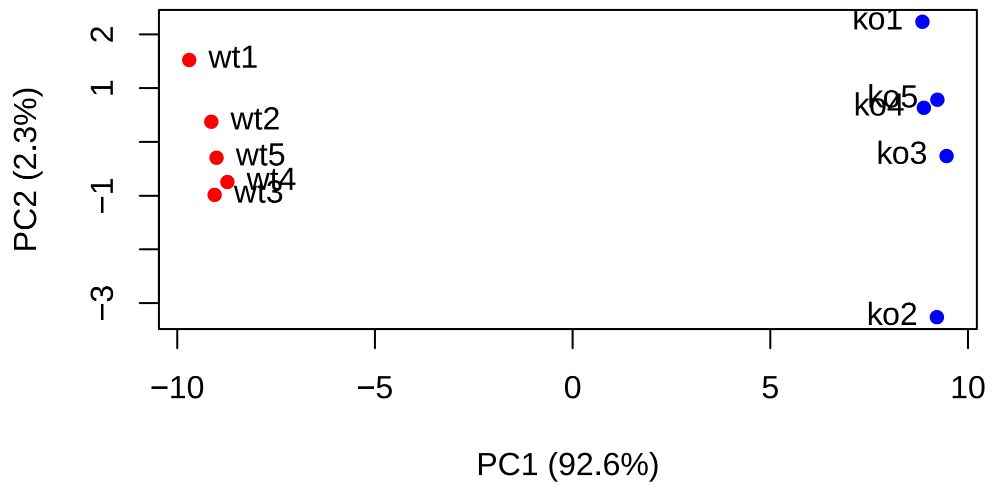
```
barplot(pca.var.per, main="Scree Plot", names.arg = paste0("PC", 1:10), xlab="Principle Co
```



As can be seen from both plots, PC1 captures the most variance.

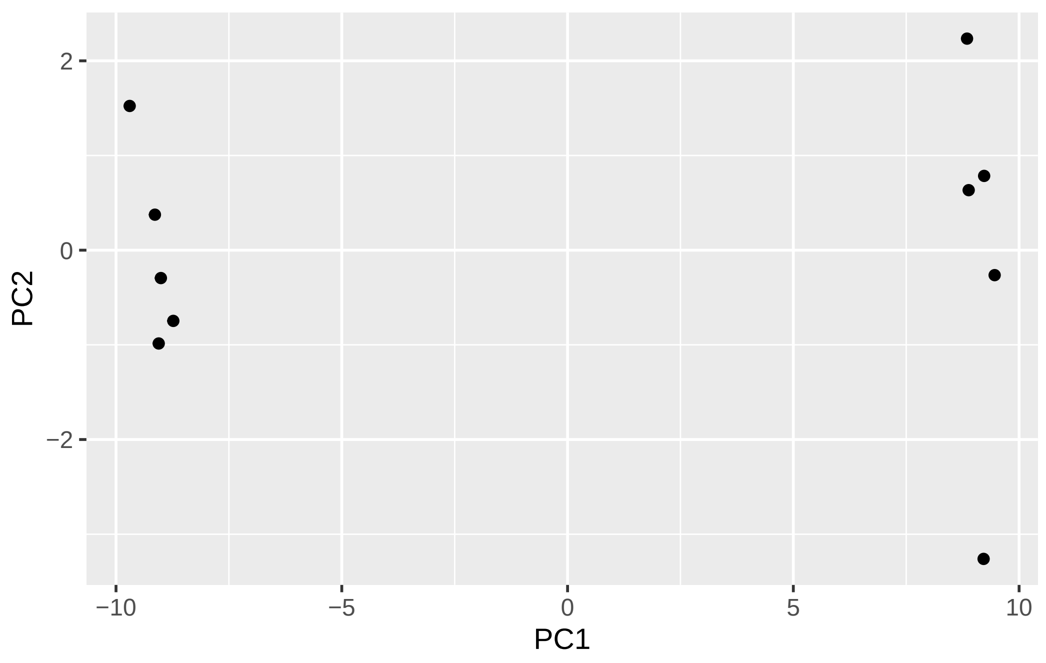
Let's now improve our main PCA plot so that we can see what each point represents:

```r
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], "%)"),

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



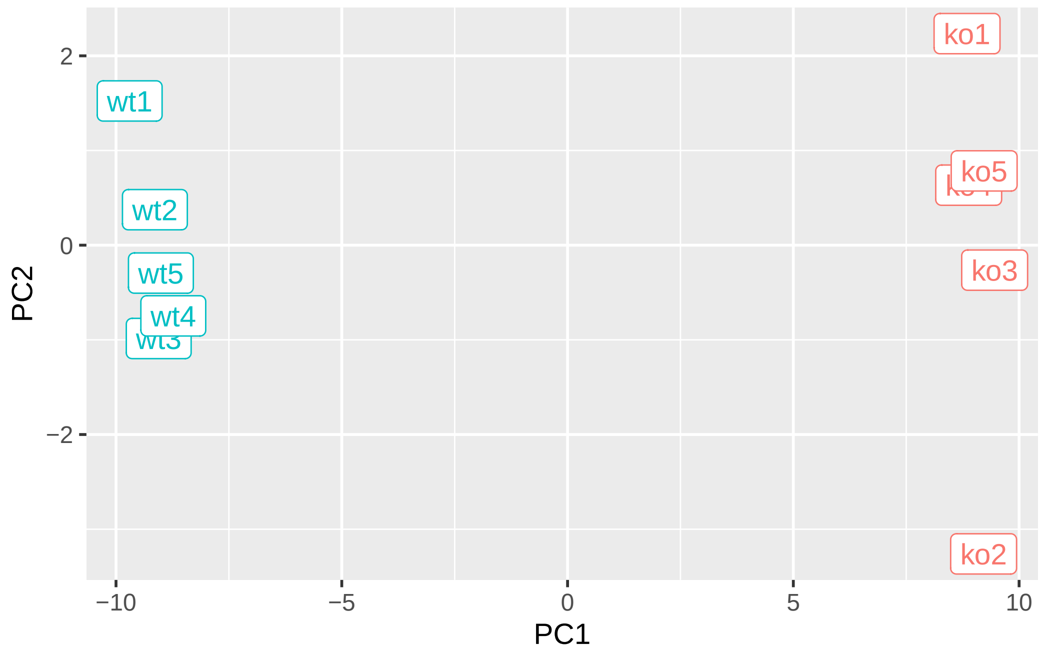
We can also use ggplot2 to make our graph, but we need to make a data frame of the data we want to graph. The data frame must include our PCA results for PC1 and PC2 as columns, as well as any additional info we want to give aesthetic mappings as more columns in our graph.

```
library(ggplot2)

df <- as.data.frame(pca$x) # pca$x is a matrix by default

ggplot(df) +
  aes(PC1, PC2) +
  geom_point()
```



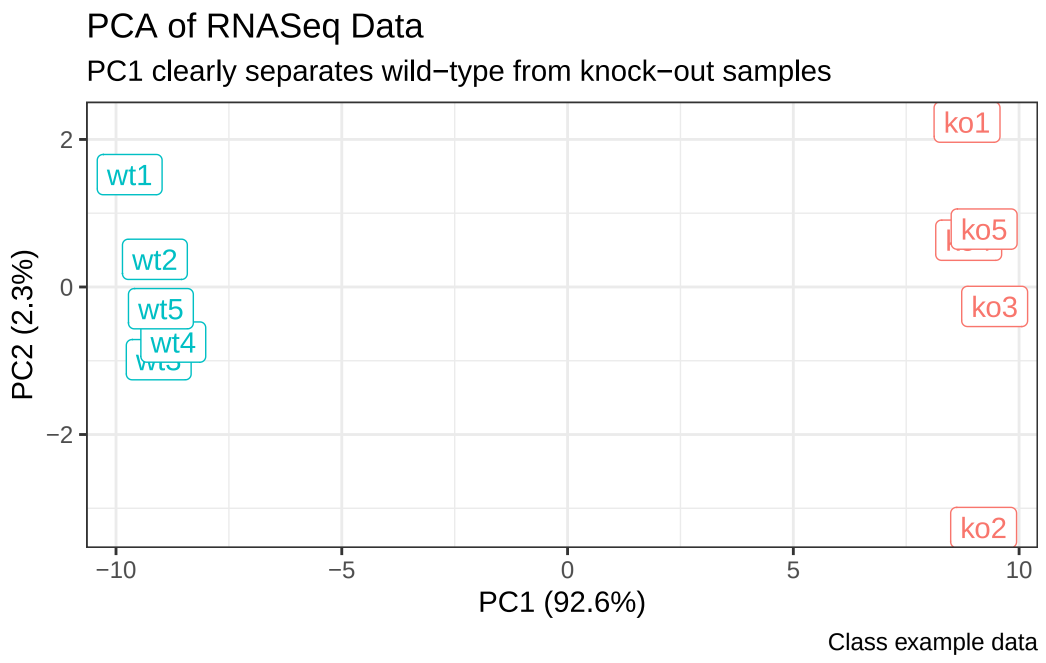
Let's add some info to our data frame so we can label the points and color them by if they are wt/ko:

```r
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 = F)
p
```



Now, let's make the graph look a bit nicer:

```
p + labs(title="PCA of RNASeq Data",
         subtitle = "PC1 clearly separates 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()
```



We can look at what are the top 10 genes that contrimute most to PC1 in either diretion (+ or -).

```r
loading_scores <- pca$rotation[,1]

# Find the top 10 genes that contribute most to PC1 in either direction
gene_scores <- abs(loading_scores)
gene_score_ranked <- sort(gene_scores, decreasing = T)

# show the naems of the top 10 genes
top_10_genes <- names(gene_score_ranked[1:10])
top_10_genes
```

```
 [1] "gene100" "gene66"  "gene45"  "gene68"  "gene98"  "gene60"  "gene21"
 [8] "gene56"  "gene10"  "gene90"
```

We should study these genes further! They seem to contrubute a lot to the differences between the wild-type and the knock-out.