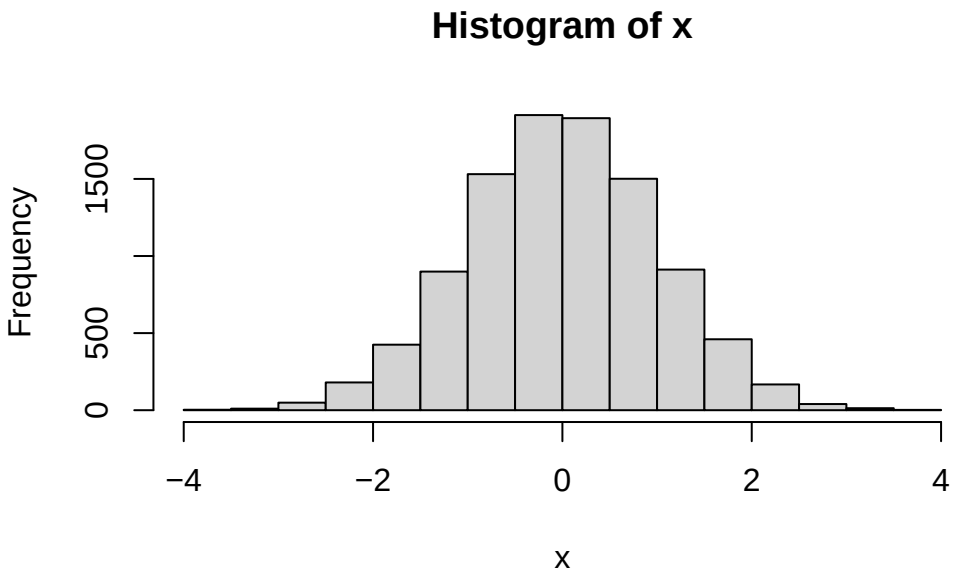
Class 07: Machine Learning 1

Kaitlyn Powell

K-menas clustering

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

```
x <- rnorm(10000)
hist(x)
```

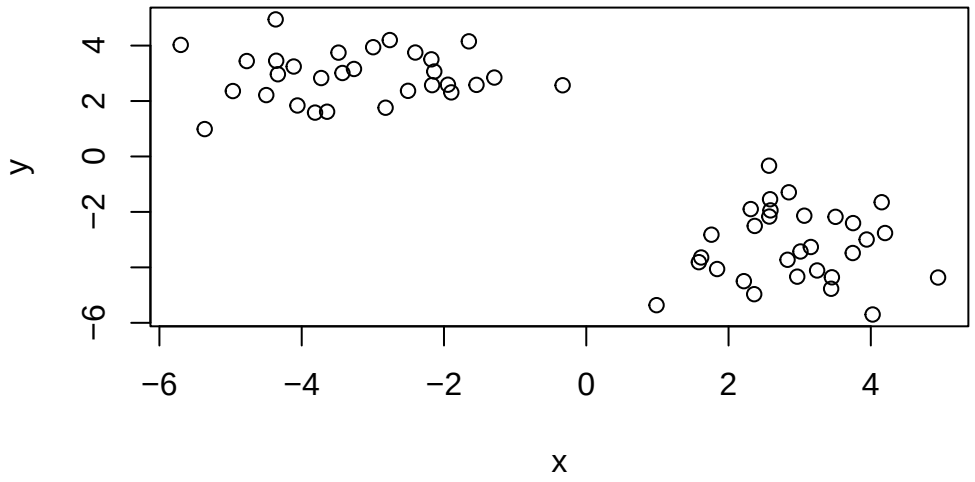


Let's make some numbers centered on -3

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

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

```r
tmp <- c(rnorm(30, -3), rnorm(30, +3))

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  2.921088 -3.217296
2 -3.217296  2.921088

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] 73.46651 73.46651
 (between_SS / total_SS =  88.5 %)

Available components:

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

```
km$centers
```

```
          x         y
1  2.921088 -3.217296
2 -3.217296  2.921088
```

Q. How many points are in each cluster?

```
km$size
```

```
[1] 30 30
```

Q. What 'component' of your result object details

- cluster size?
- cluster assignment/membership?
- cluster center?

```
km$cluster
```

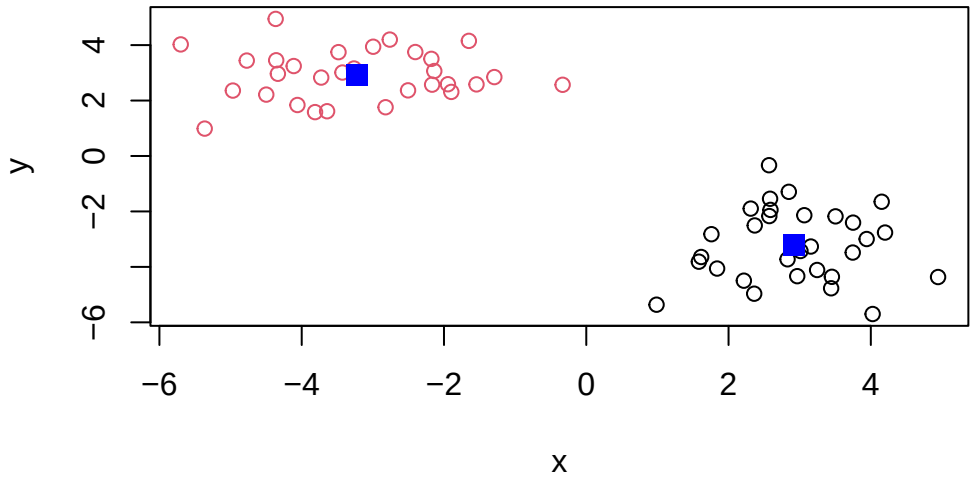
```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
km$centers
```

```
          x         y
1  2.921088 -3.217296
2 -3.217296  2.921088
```

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

```r
plot(x, col=km$cluster)
points(km$centers, col= "blue", pch=15, cex =1.5)
```



Hierarchical Clustering

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

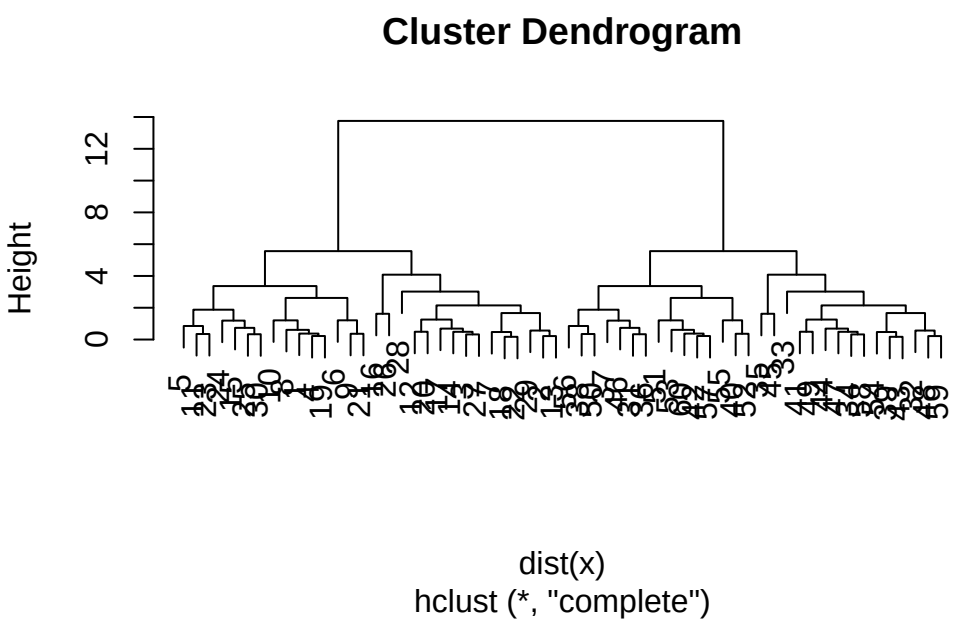
The `hclust()` function requires an input distance matrix, which I can get from the `dist()` function.

```r
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)
```



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 cluster. To do this we use the `cutree()` function.

```
cutree(hc, h=8)
```

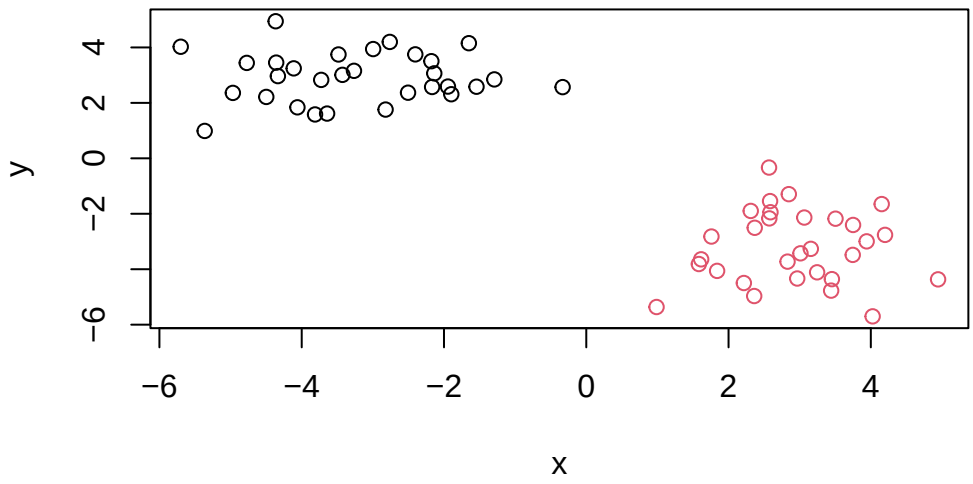
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

Use `cutree()` with a k=2.

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

A plot of our data colored by our hclust grps.

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



Principal Component Analysis (PCA)

```
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
```

Part 1: PCA of UK Food 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?

There are 17 rows and 5 columns in the data frame named x. R functions that can be used in order to answer this question include `nrow()`, `ncol()`, or `dim()`.

Complete the following code to find out how many rows and columns are in x?

```
dim(x)
```

```
[1] 17  5
```

Preview the 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
```

Note how the minus indexing works

```
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
```

```
dim(x)
```

```
[1] 17  4
```

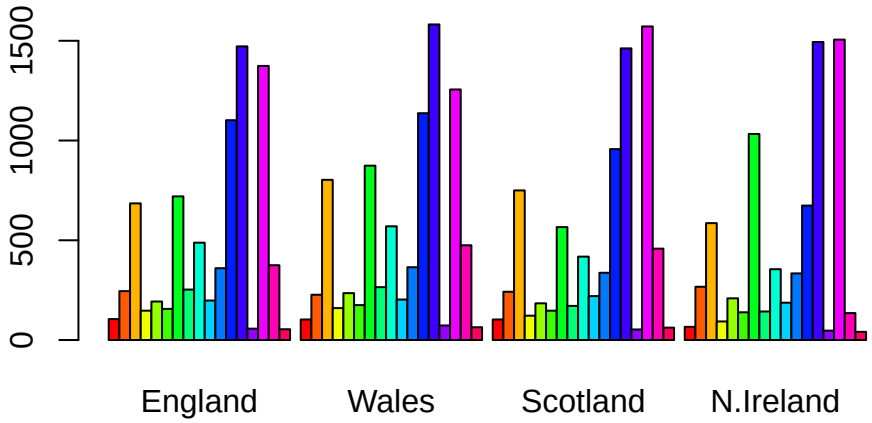
```
x <- read.csv(url, row.names=1)
head(x)
```

```
               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 second approach to solving the 'row-names problem' mentioned above because it is easier to see what variables I am working with. The first approach seems to leave more room for error because it seems that it would be easy to mis-type a number.

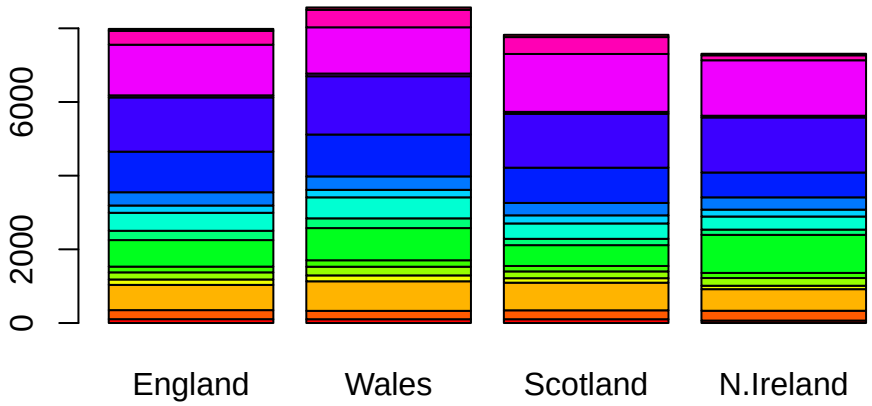
```
barplot(as.matrix(x), beside=T, col=rainbow(nrow(x)))
```



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

Changing the `beside` argument to "FALSE" in the barplot() function results in this plot.

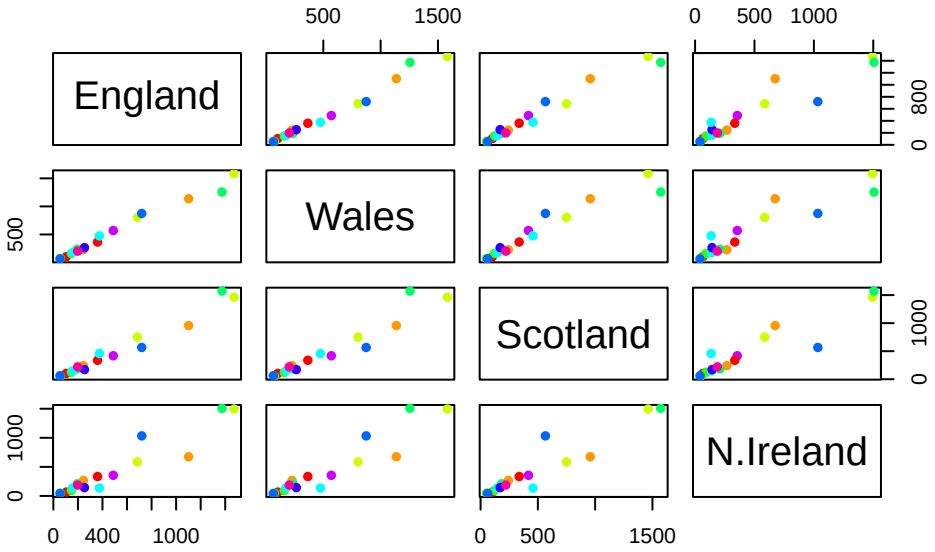
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



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 code plots all possible pairs of countries against each other in which it shows all plot options, using different pairs of countries as each axis. Each point on the plot represents a specific food group. If a given point lies on the diagonal for a given plot, it indicates that the same amount of a certain food group is consumed by the two countries being compared. If a point lies above the diagonal, it means that the country on the y-axis consumes more of that food group. Similarly, if a point lies below the diagonal, it means that the country on the x-axis consumes more of that food group.

```
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?

In terms of this data-set, the main difference between N. Ireland and the other countries of the UK is that, they tend to consume the most food groups in different amounts (either more or less) when compared to the other countries.

While this is kind of useful it takes work to dig into the details here to find out what is different in these countries.

PCA to the rescue

Principal Component Analysis (PCA for short) can be a big help in these cases where we have lots of things that are being measured in a dataset.

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

The `prcomp()` function wants as the input of the transpose of our food matrix/table/data frame.

```
pca <- prcomp(t(x))
summary(pca)
```

```
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
```

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

```
attributes(pca)
```

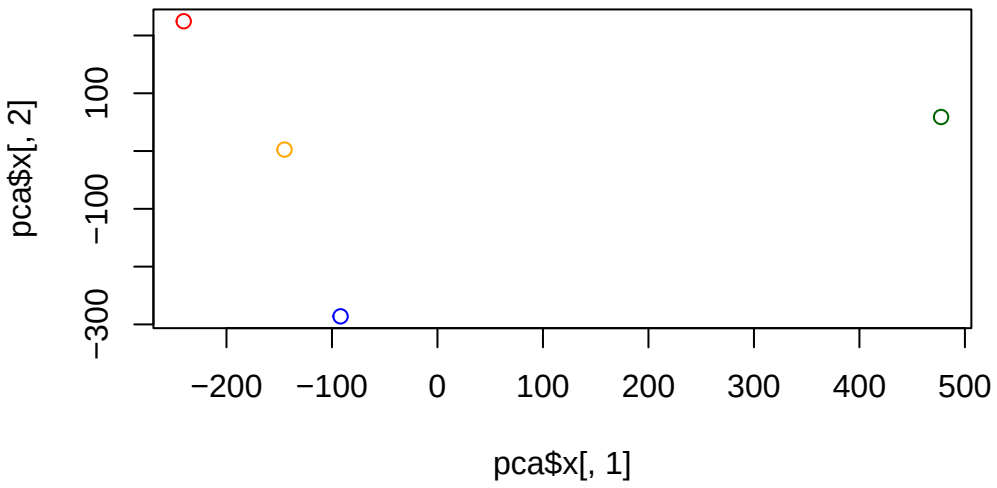
```
$names
[1] "sdev"     "rotation" "center"   "scale"    "x"

$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.

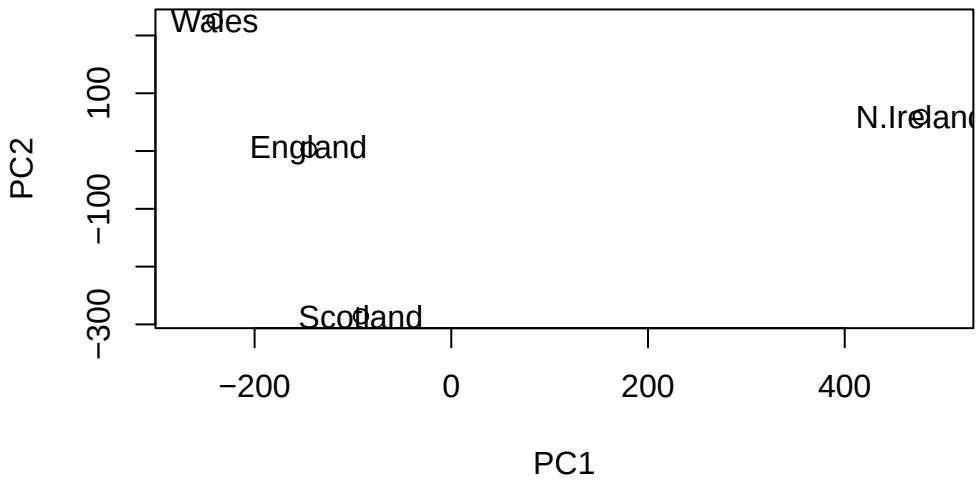
```
plot(pca$x[,1], pca$x[,2], col=c("orange",
                                 "red",
                                 "blue",
                                 "darkgreen"))
```



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

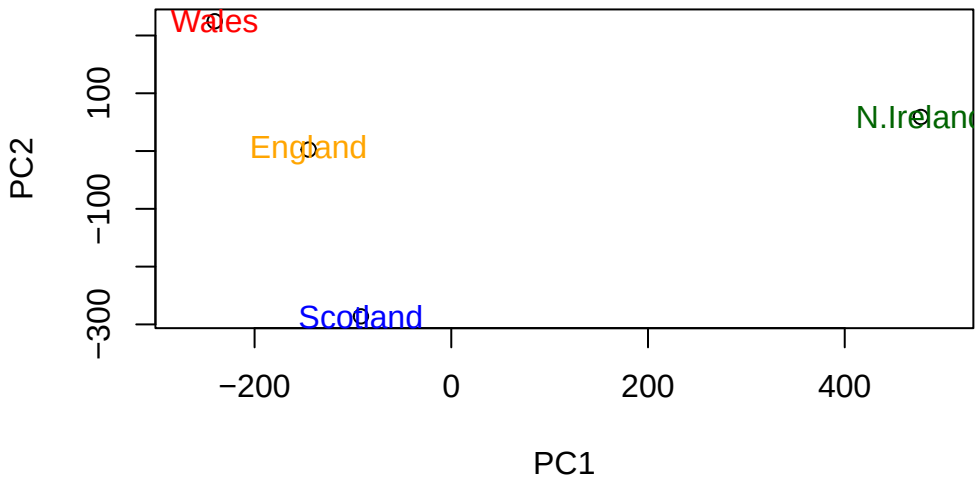
Plot PC1 vs PC2

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




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

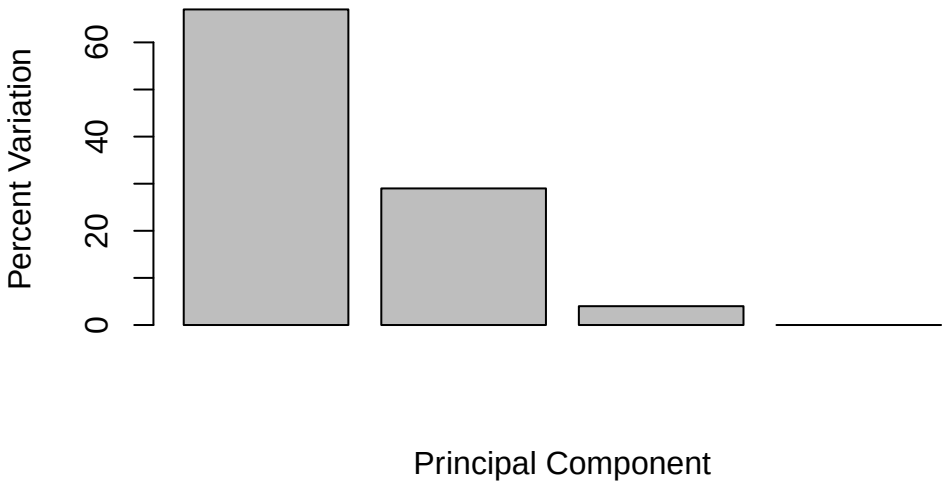
```
[1] 67 29  4  0
```

or the second row here...

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

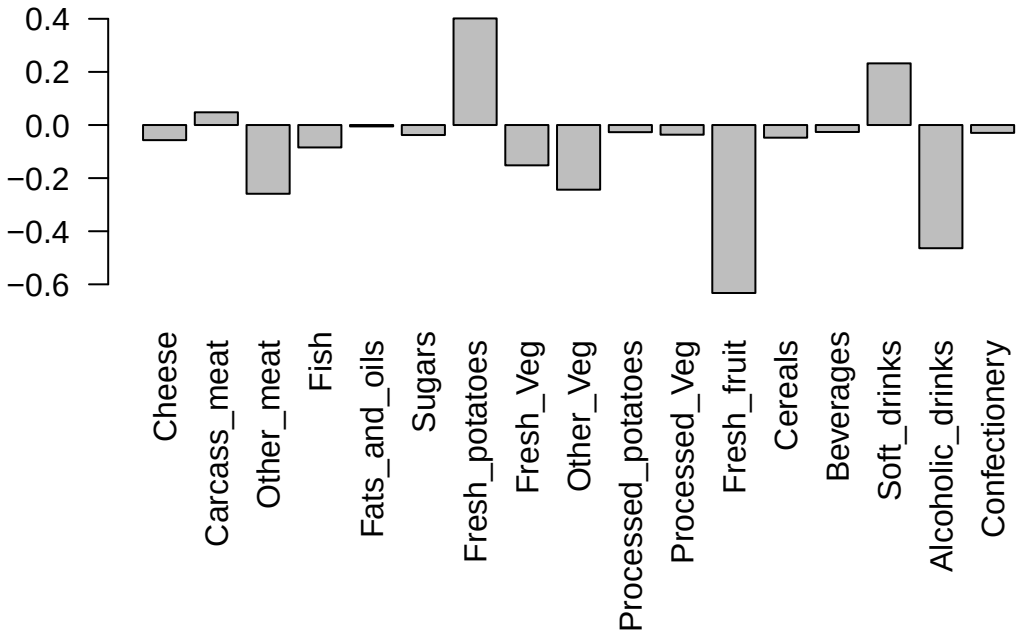
```
                             PC1       PC2      PC3           PC4
Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

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



Lets focus on PC1 as it accounts for $>$ 90% of variance

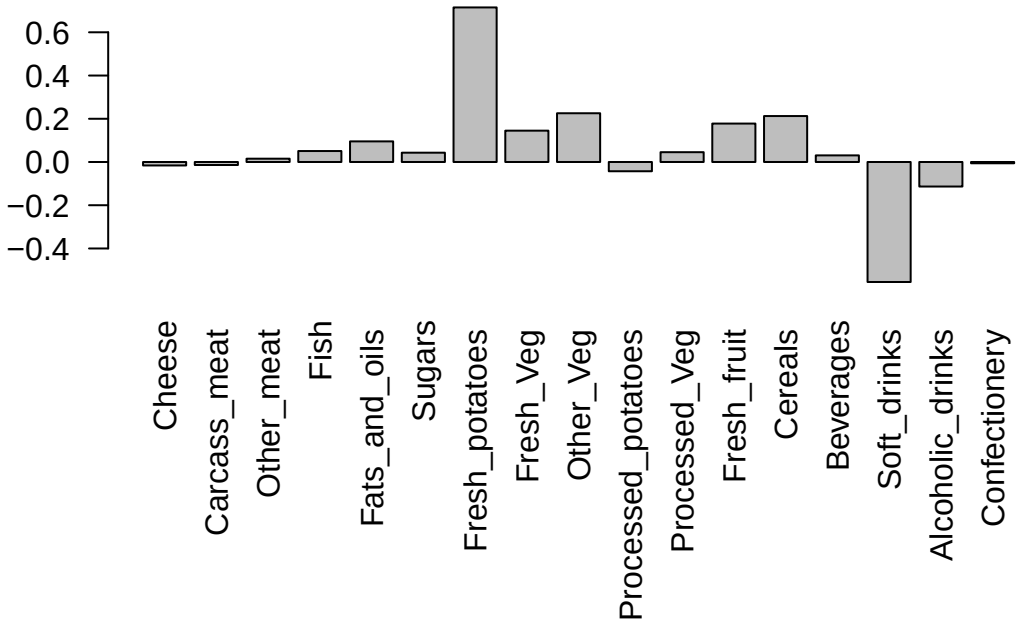
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



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

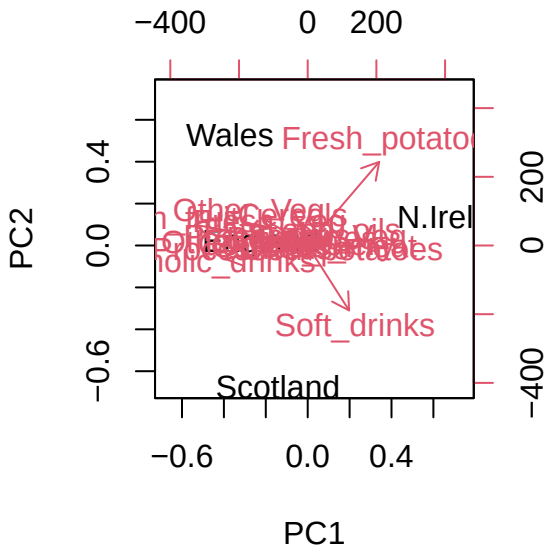
PC2 mainly features the food groups, fresh potatoes and soft drinks.

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



The inbuilt biplot() can be useful for small datasets

```
biplot(pca)
```



Part 2: PCA of RNA-Seq Data

Here we apply PCA to some example RNA-Seq data of a know-out experiment.

First we read the dataset:

```
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 data set.

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

```
[1] 100  10
```

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

```
[1] 100
```

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

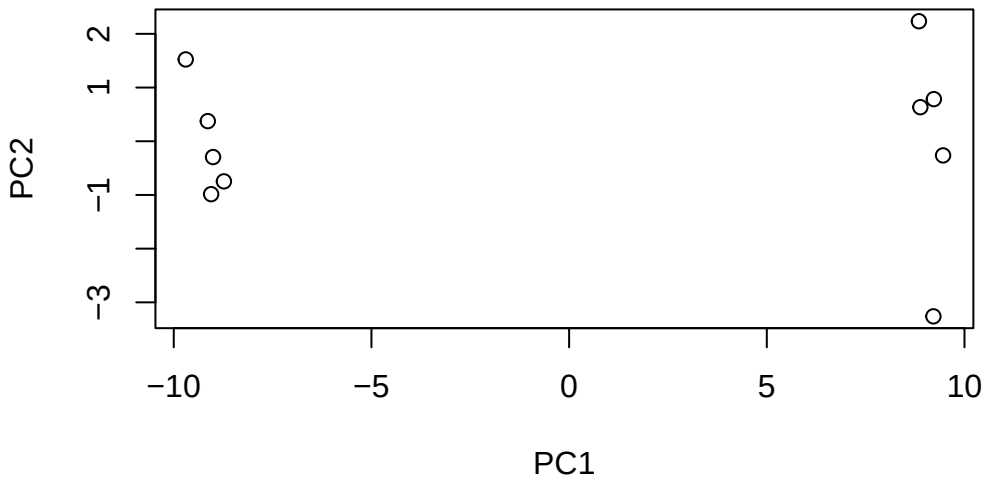
```
[1] 10
```

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

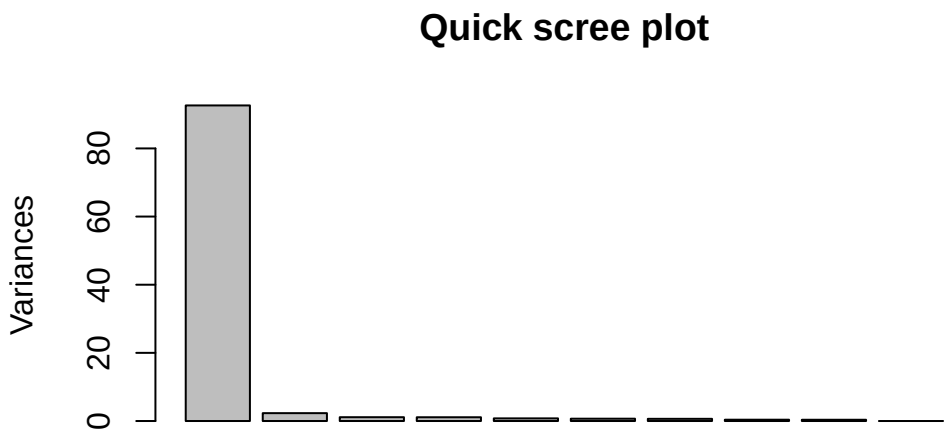
```
  plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2")
```




```
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
```

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



Variance captured per PC

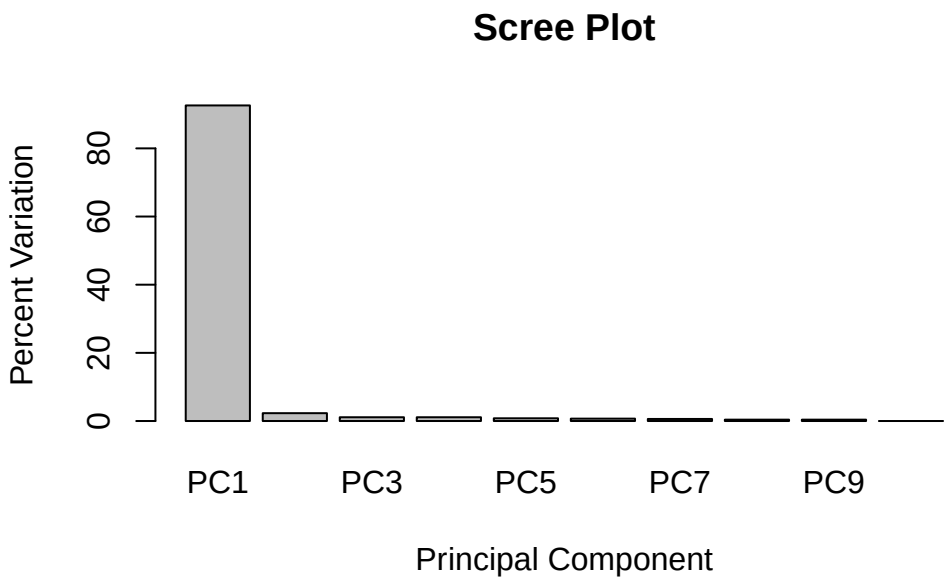
```
pca.var <- pca$sdev^2
```

Percent variance is often more informative to look at

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

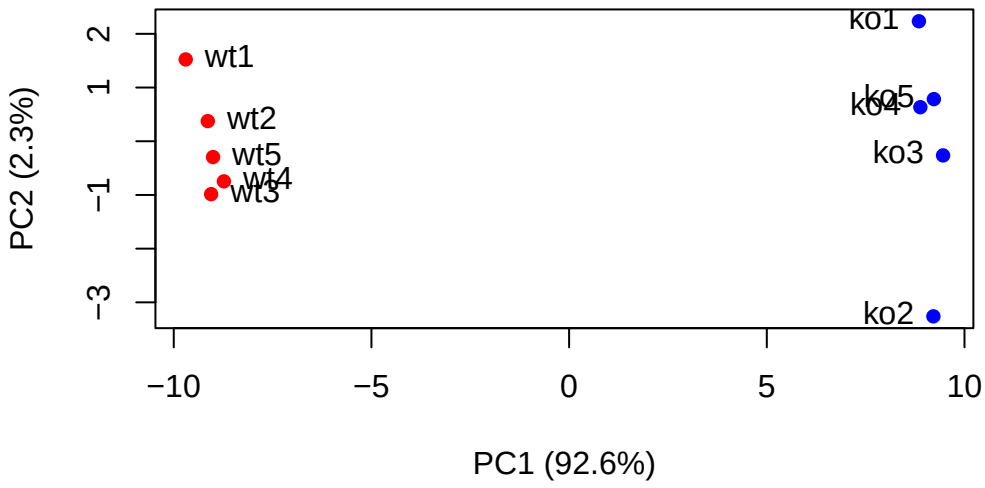
```
 [1] 92.6  2.3  1.1  1.1  0.8  0.7  0.6  0.4  0.4  0.0
```

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



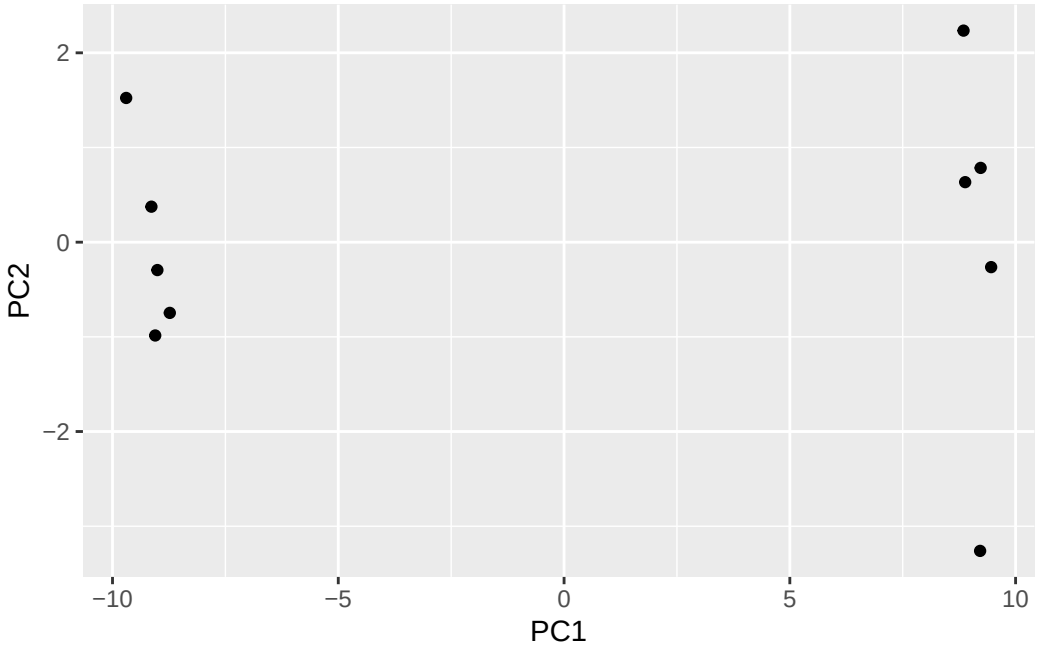
A vector of colors for wt and ko samples

```
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)))
```



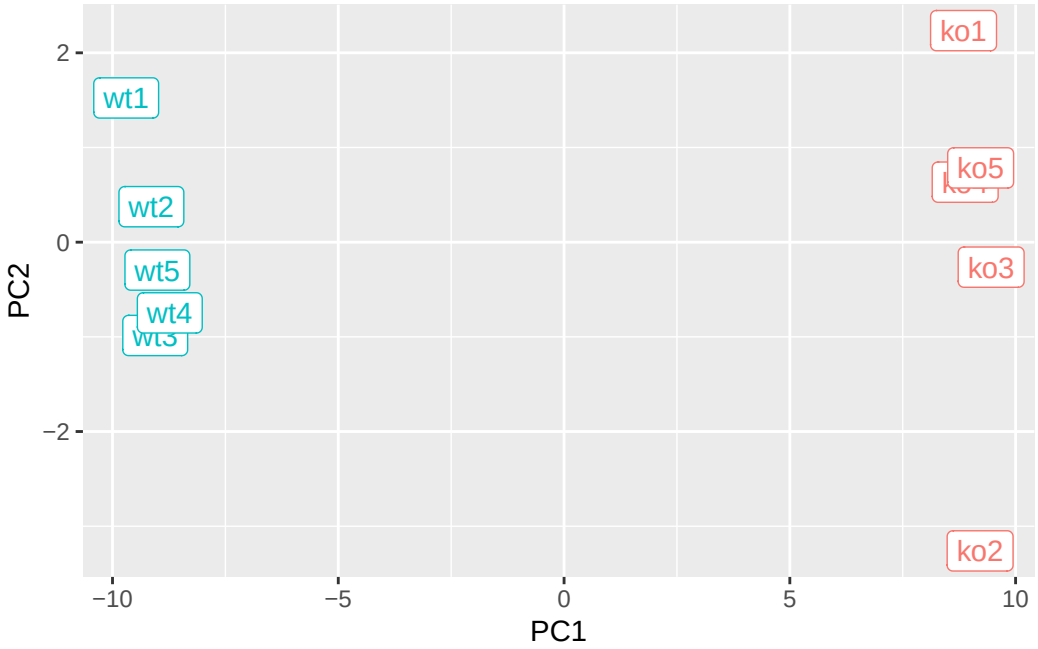
Using ggplot

```
library(ggplot2)

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

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

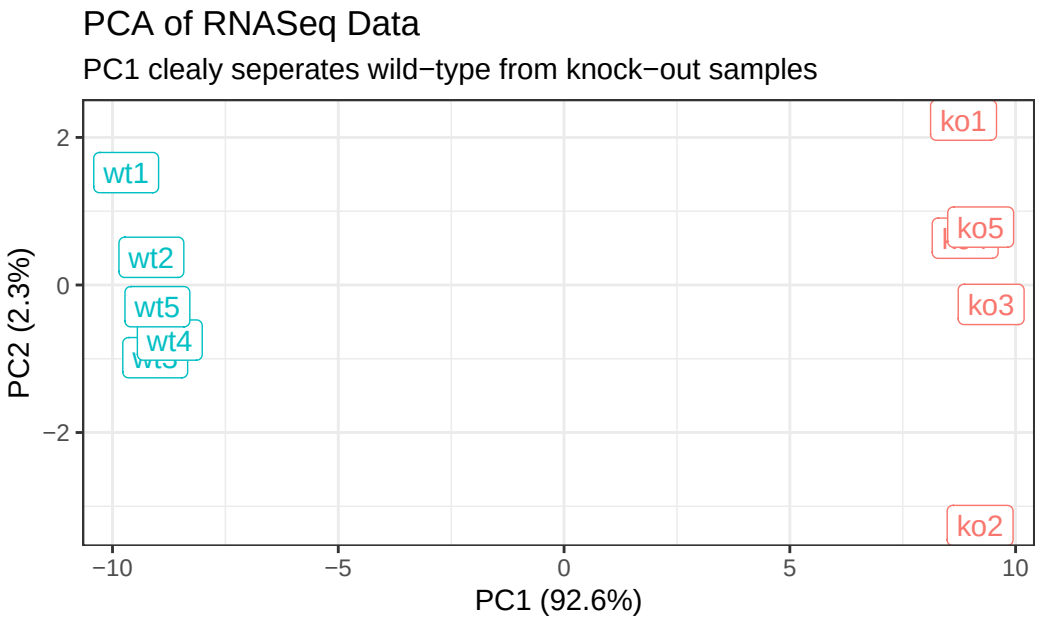


Add a ‘wt’ and ‘ko’ “condition” column

```
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
```




```
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()
```



Optional: Gene loadings

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

Find the top 10 measurements (genes) that contribute

most to PC1 in either direction (+ or -)

```
gene_scores <- abs(loading_scores)
gene_score_ranked <- sort(gene_scores, decreasing=TRUE)
```

show the names 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"
```