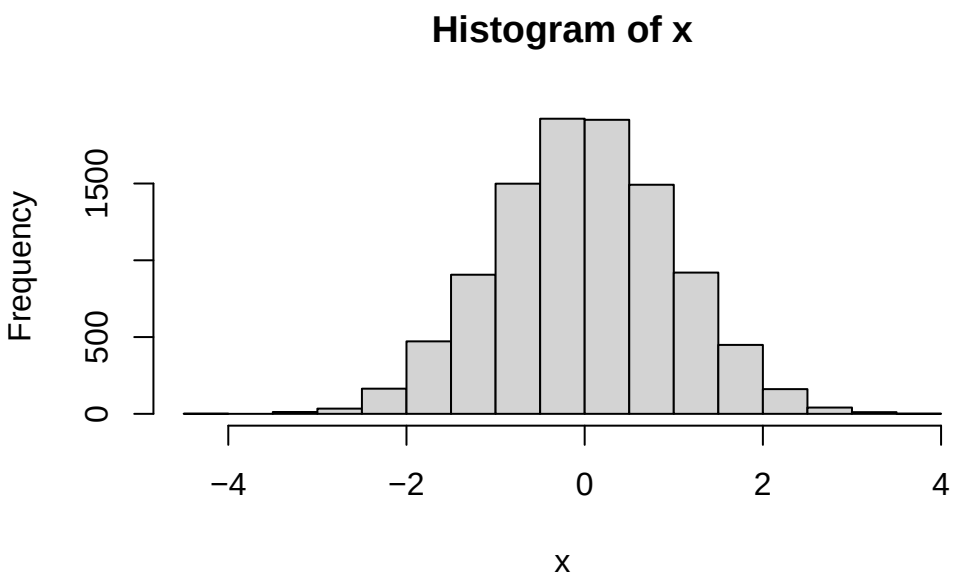
Class07 Lab

Mady Welch

K-means clustering

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

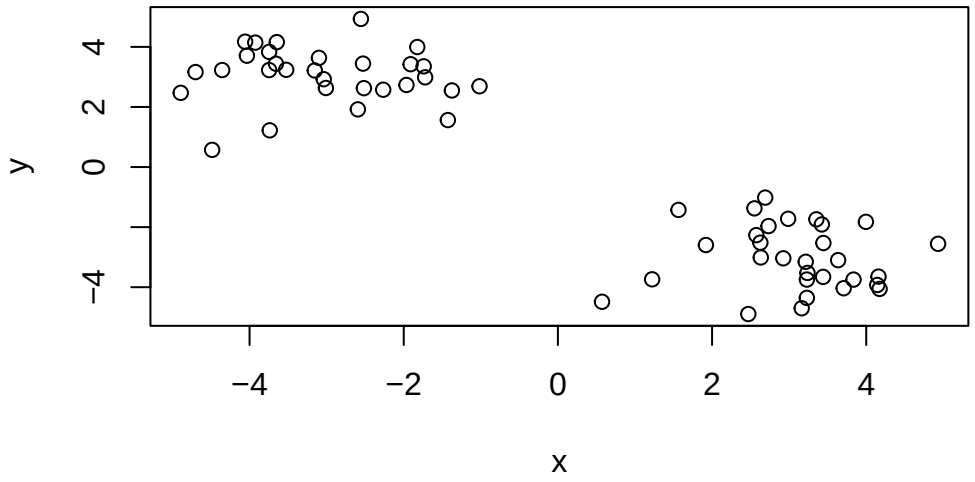
```
x <- rnorm(10000)
hist(x)
```



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

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

```r
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  3.059414 -3.008535
2 -3.008535  3.059414

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] 58.21475 58.21475
```

```
 (between_SS / total_SS =  90.5 %)

Available components:

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

```
km$centers
```

```
          x         y
1  3.059414 -3.008535
2 -3.008535  3.059414
```

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
```

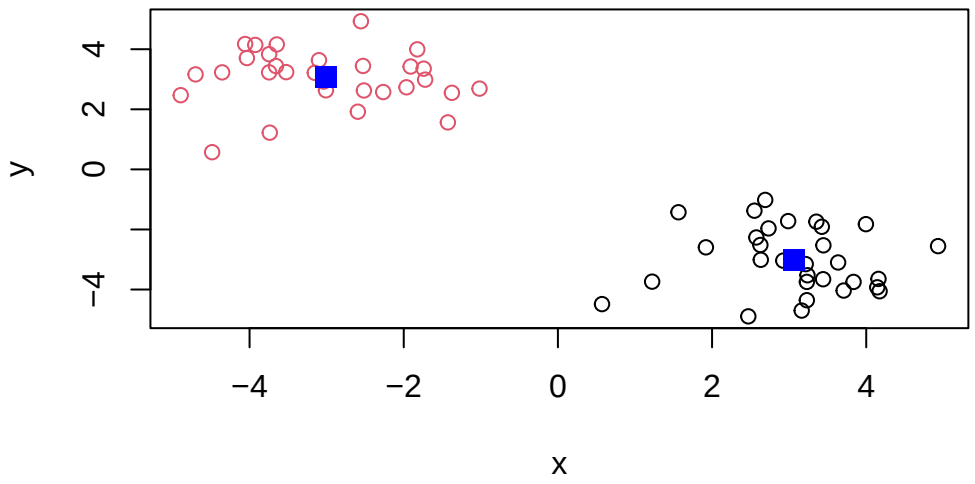
```
 [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
[39] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
km$centers
```

```
          x         y
1  3.059414 -3.008535
2 -3.008535  3.059414
```

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

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



Hierarchal Clustering

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

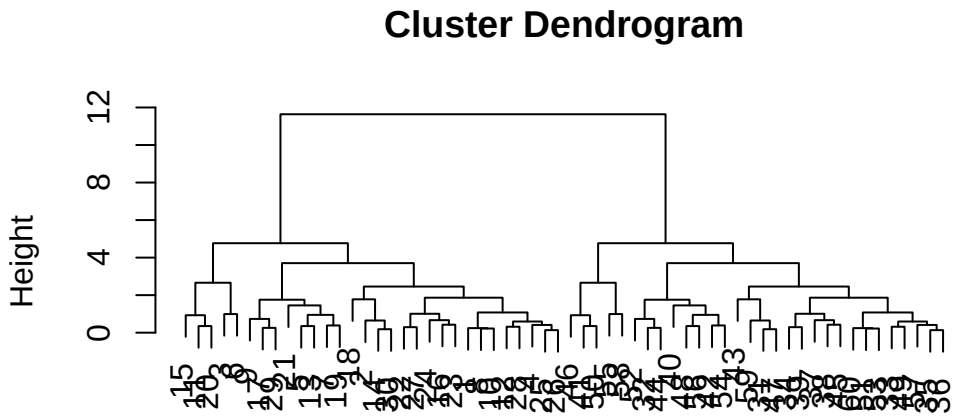
The `hclust()` function requires an input distance matrix, which I can get from the `dist()` function.

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

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

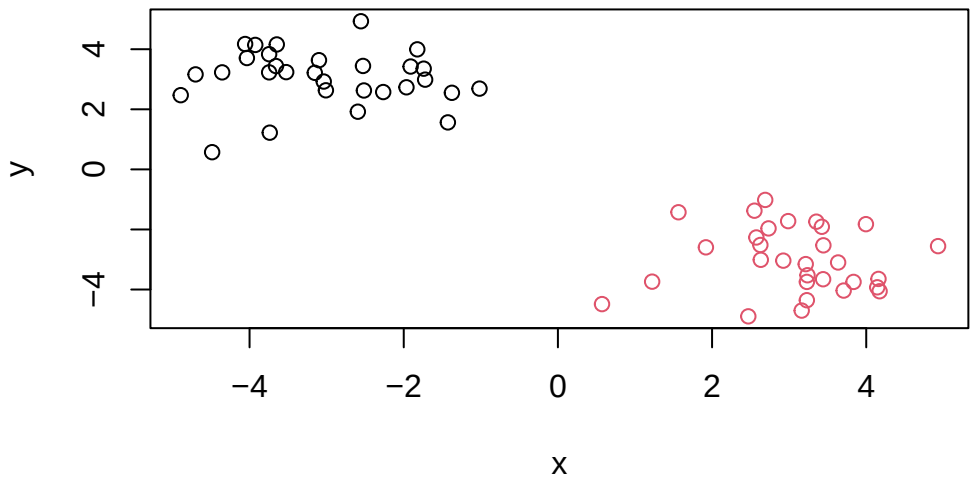
```
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[39] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

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)
```

>Q1. How many rows and columns are in your new data frame named x? What R functions could you use to answer this questions?
::: {.cell}
`{.r .cell-code} dim(x)`
::: {.cell-output .cell-output-stdout}
- 17 rows and 5 columns

It is always a good idea to examine your imported data to make sure it meets your expectations.

```
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
```

Change the "x" column to rownames:

```
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
```

- Now there are 17 rows and 4 columns

OR we could use `read.csv(url, row.names=1)`

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

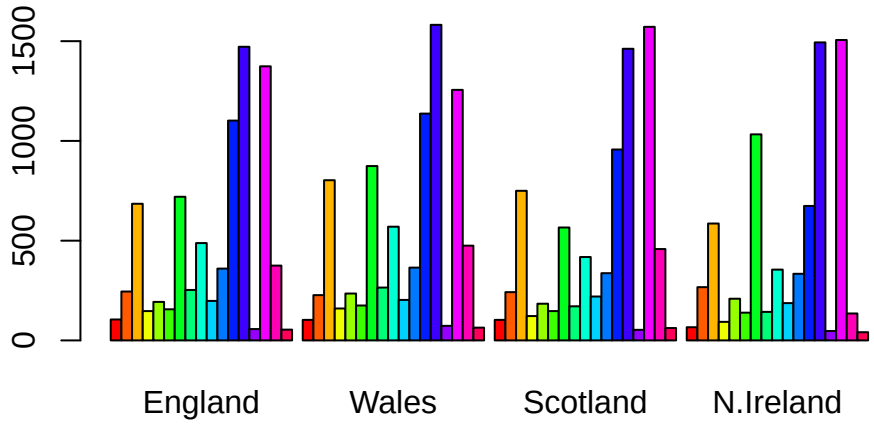
```
              England Wales Scotland N.Ireland
Cheese            105   103      103        66
Carcass_meat      245   227      242       267
Other_meat        685   803      750       586
Fish              147   160      122        93
Fats_and_oils     193   235      184       209
Sugars        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

- The second method is the better way to do it so we don't accidentally delete a column every time we call for x

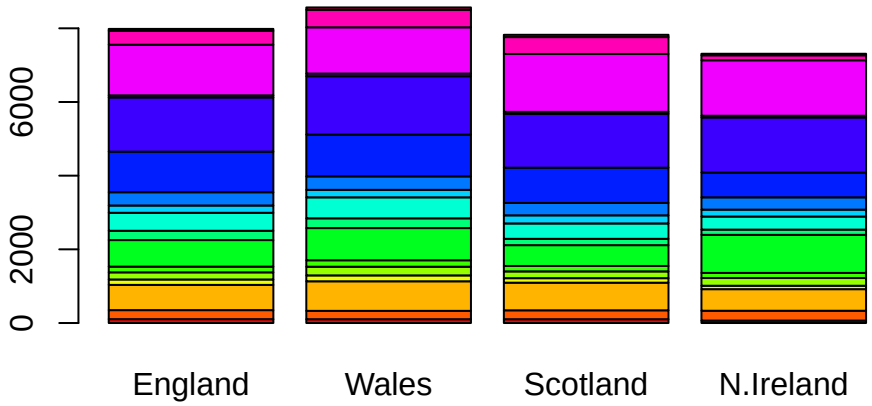
Barplot of the data:

```
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?

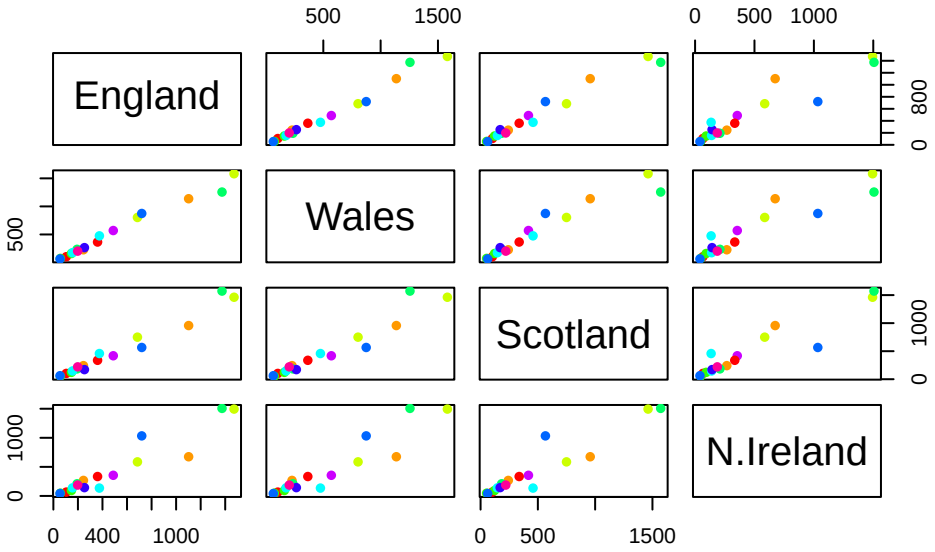
```
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



- If we change the beside argument to equal FALSE it will create the stacked plot

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



- If points lie on the diagonal that means they are the same

While this plot is kind of useful, it is difficult to interperet.

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

- N. Ireland points deviate from the diagonal more than other countries.

PCA to the rescue

Principal Component Analysis 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 input the transpose of our food matrix/table/data.frame, so we use `t()`

```
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 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)
```

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

$class
[1] "prcomp"
```

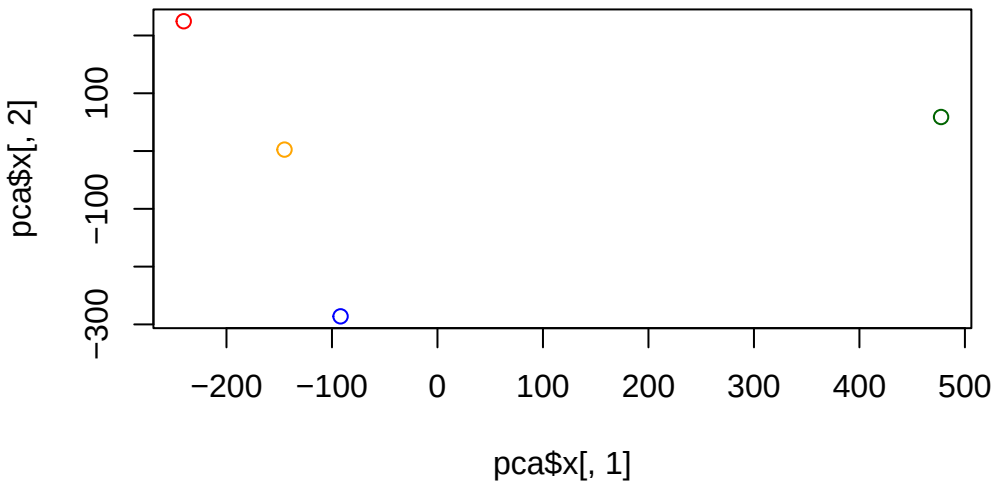
```
head(pca$x)
```

```
                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.

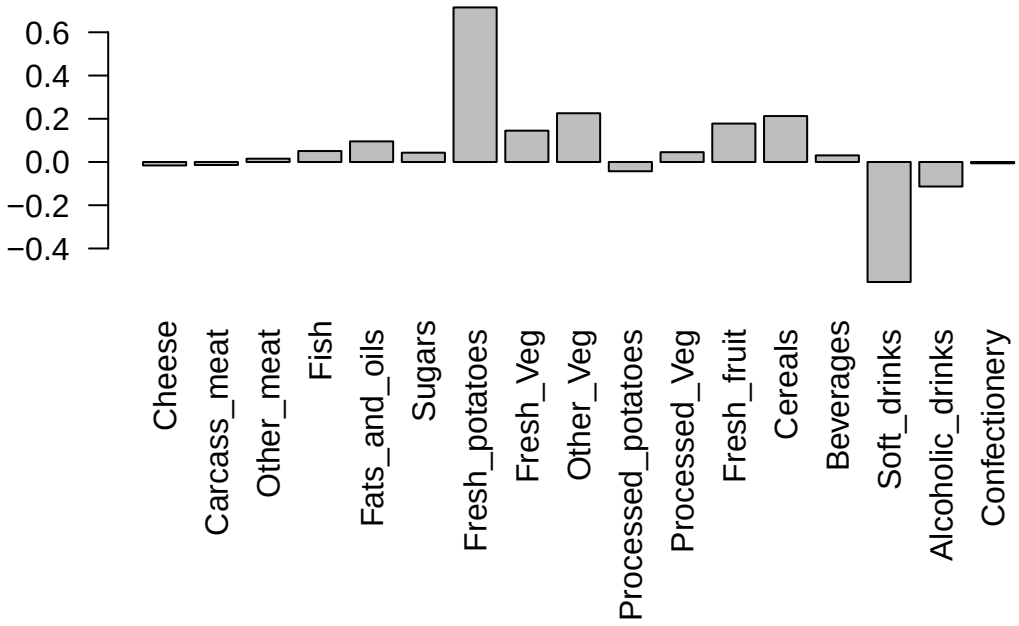
> Q7. Generate a plot of PC1 vs PC2 Q8. Customize your plot so that the colors of the points match the colors in our UK and Ireland map and table at start of this document.

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



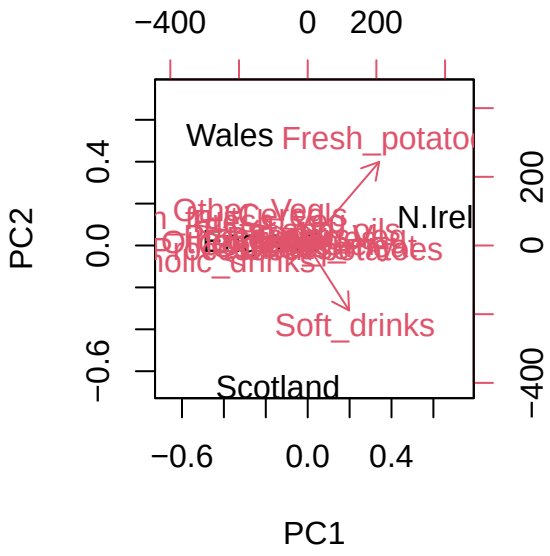
Q9. Generate a similar 'loadings plot' for PC2. What two food groups feature prominantely and what does PC2 maninly tell us about?

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



Biplots

```
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?

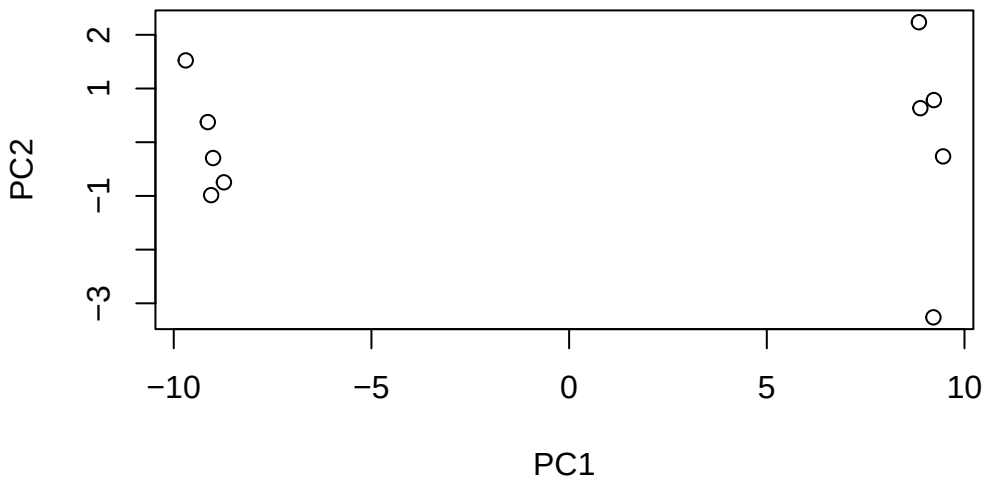
```
dim(rna.data)
```

```
[1] 100  10
```

- 100 rows and 10 columns

PCA and plot the results:

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



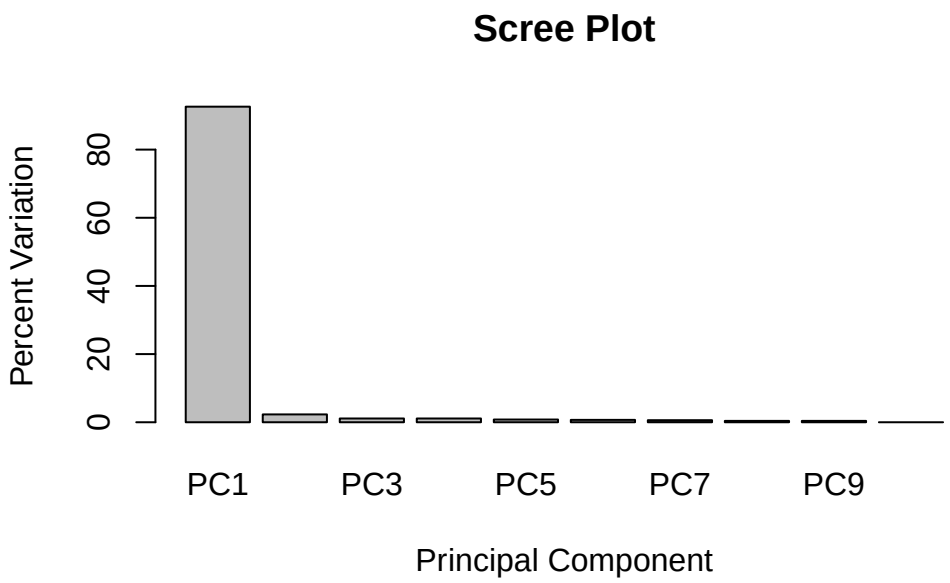

```
summary(pca)
```

```
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
```

- PC1 captures 92.6% of the original variance.

Let's make a plot:

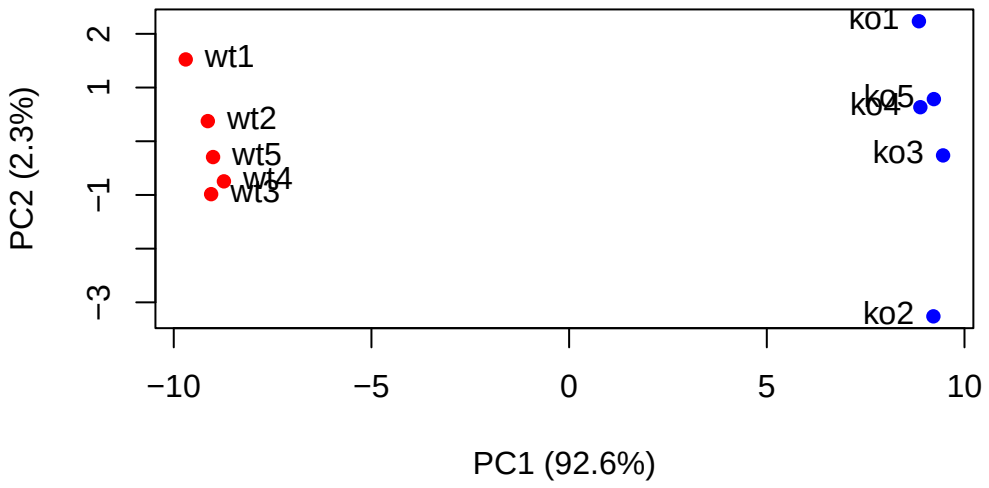
```
pca.var <- pca$sdev^2
pca.var.per <- round(pca.var/sum(pca.var)*100, 1)
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



- Once again shows that PC1 captures most of the variance.

We can make the plot a bit more useful:

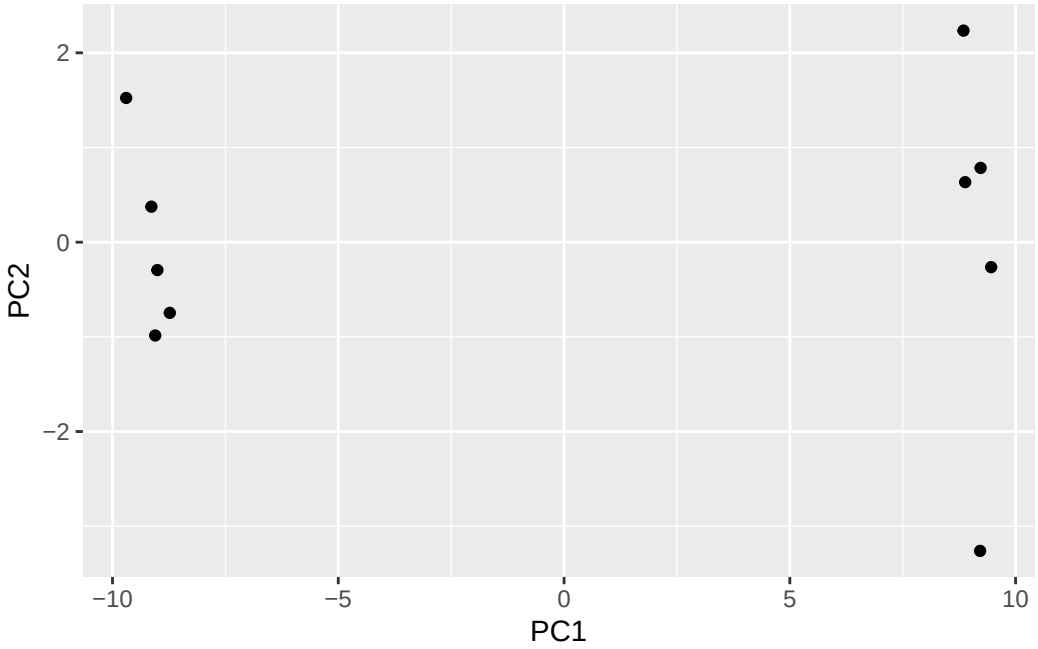
```
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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

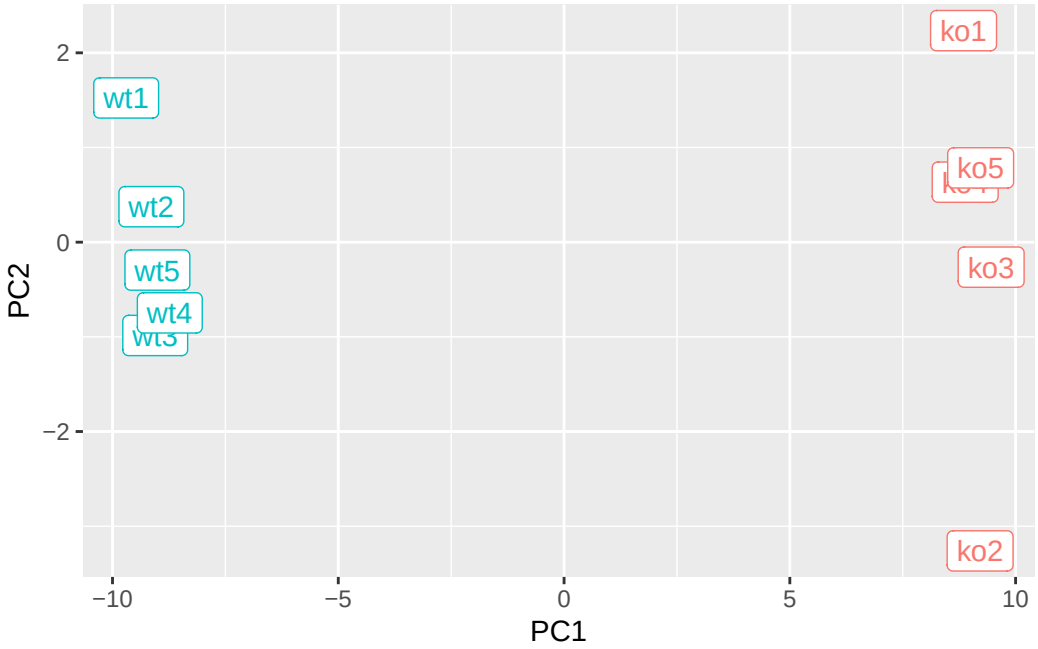
We could use ggplot2 here but we will first need a data.frame as input for the main ggplot() function. This data.frame will need to contain our PCA results and additional columns for any aesthetic mappings.

```
library(ggplot2)

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

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



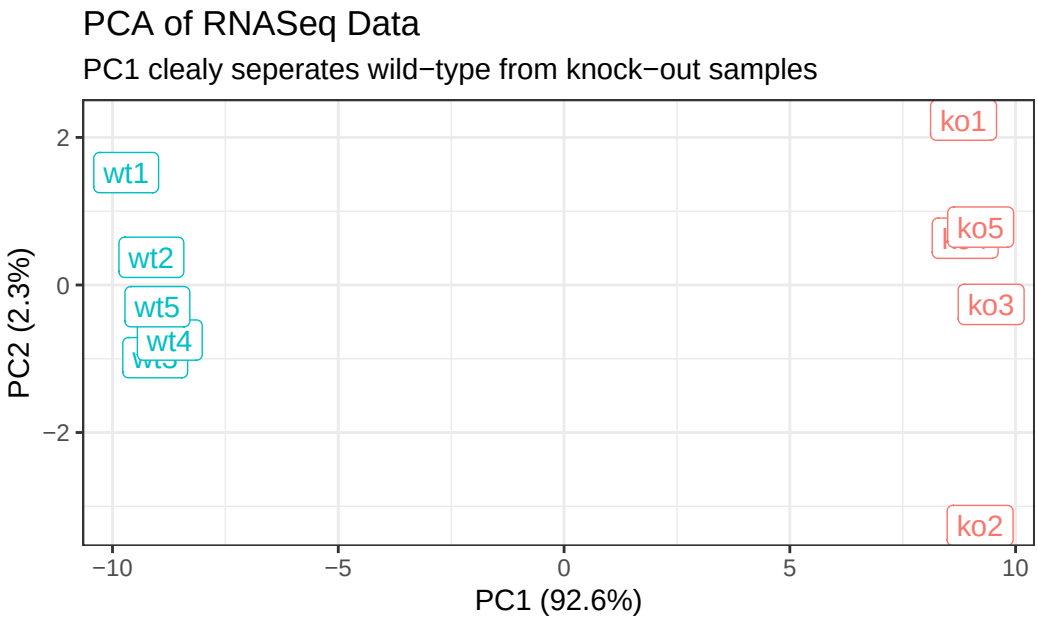
- If we want to add a condition specific color or labels for wild-type and knock-out sample we need to add this information to the data.frame

```
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
```



Now we can add a few more labels:

```
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()
```


PCA of RNASeq Data
PC1 clealy seperates wild−type from knock−out samples
ko1
wt1
ko5
wt2
wt5
ko3
wt4
ko2
2
0
−2
PC2 (2.3%)
−10
−5
0
5
10
PC1 (92.6%)
Class example data