Class 08 Lab (Hands on with PCA)

Cindy Tran

1/25/2022

First up, kmeans()

Demo of using a kmeans() function R. First, we'll make up some data with a known structure.

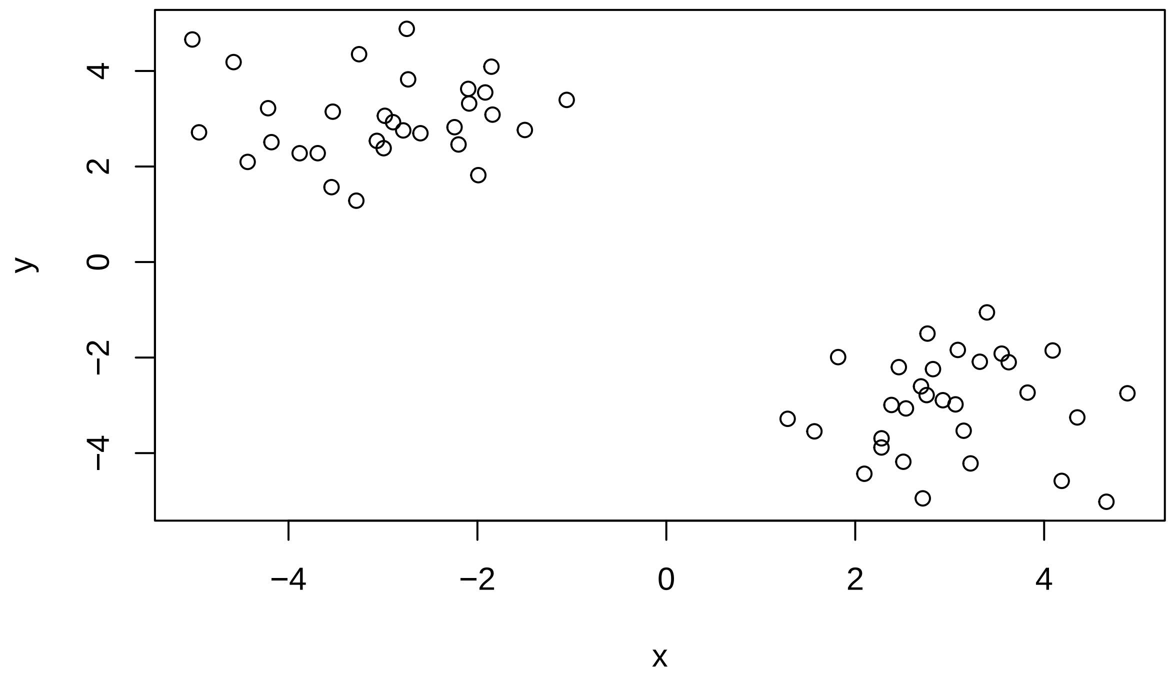
```r
tmp <- c(rnorm(30, -3), rnorm(30, 3))
tmp
```

```
##  [1] -3.064582 -1.054479 -5.018259 -4.947368 -4.216448 -2.242273 -3.252864
##  [8] -1.840153 -3.882097 -4.581467 -2.603311 -4.180810 -2.747890 -1.990400
## [15] -2.200072 -3.282043 -1.851823 -2.733582 -3.690920 -2.980103 -2.097329
## [22] -3.531481 -3.544700 -2.087204 -1.917406 -4.432514 -2.992661 -2.785482
## [29] -1.497685 -2.892918  2.927958  2.764994  2.755970  2.383580  2.096607
## [36]  3.550509  3.318986  1.567897  3.148368  3.625055  3.061729  2.278798
## [43]  3.824655  4.090380  1.284505  2.461614  1.819127  4.882160  2.509351
## [50]  2.696460  4.185999  2.277764  3.086049  4.351204  2.823317  3.220739
## [57]  2.715157  4.659633  3.394489  2.537130
```

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

```
##               x        y
##  [1,] -3.064582  2.537130
##  [2,] -1.054479  3.394489
##  [3,] -5.018259  4.659633
##  [4,] -4.947368  2.715157
##  [5,] -4.216448  3.220739
##  [6,] -2.242273  2.823317
##  [7,] -3.252864  4.351204
##  [8,] -1.840153  3.086049
##  [9,] -3.882097  2.277764
## [10,] -4.581467  4.185999
## [11,] -2.603311  2.696460
## [12,] -4.180810  2.509351
## [13,] -2.747890  4.882160
## [14,] -1.990400  1.819127
## [15,] -2.200072  2.461614
## [16,] -3.282043  1.284505
## [17,] -1.851823  4.090380
## [18,] -2.733582  3.824655
## [19,] -3.690920  2.278798
## [20,] -2.980103  3.061729
## [21,] -2.097329  3.625055
## [22,] -3.531481  3.148368
## [23,] -3.544700  1.567897
## [24,] -2.087204  3.318986
## [25,] -1.917406  3.550509
## [26,] -4.432514  2.096607
## [27,] -2.992661  2.383580
## [28,] -2.785482  2.755970
## [29,] -1.497685  2.764994
## [30,] -2.892918  2.927958
## [31,]  2.927958 -2.892918
## [32,]  2.764994 -1.497685
## [33,]  2.755970 -2.785482
## [34,]  2.383580 -2.992661
## [35,]  2.096607 -4.432514
## [36,]  3.550509 -1.917406
## [37,]  3.318986 -2.087204
## [38,]  1.567897 -3.544700
## [39,]  3.148368 -3.531481
## [40,]  3.625055 -2.097329
## [41,]  3.061729 -2.980103
## [42,]  2.278798 -3.690920
## [43,]  3.824655 -2.733582
## [44,]  4.090380 -1.851823
## [45,]  1.284505 -3.282043
## [46,]  2.461614 -2.200072
## [47,]  1.819127 -1.990400
## [48,]  4.882160 -2.747890
## [49,]  2.509351 -4.180810
## [50,]  2.696460 -2.603311
## [51,]  4.185999 -4.581467
## [52,]  2.277764 -3.882097
## [53,]  3.086049 -1.840153
## [54,]  4.351204 -3.252864
## [55,]  2.823317 -2.242273
## [56,]  3.220739 -4.216448
## [57,]  2.715157 -4.947368
## [58,]  4.659633 -5.018259
## [59,]  3.394489 -1.054479
## [60,]  2.537130 -3.064582
```

```
plot(x)
```



Now we have some made-up data in 'x'. Let's see how kmeans() works with this data

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

```
## K-means clustering with 2 clusters of sizes 30, 30
##
## Cluster means:
##           x         y
## 1 -3.004677  3.010006
## 2  3.010006 -3.004677
##
## 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] 53.50696 53.50696
##  (between_SS / total_SS =  91.0 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How many points are in each cluster?

```
k$size
```

```
## [1] 30 30
```

Q. How do we get to the cluster membership/assignment?

```
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
```

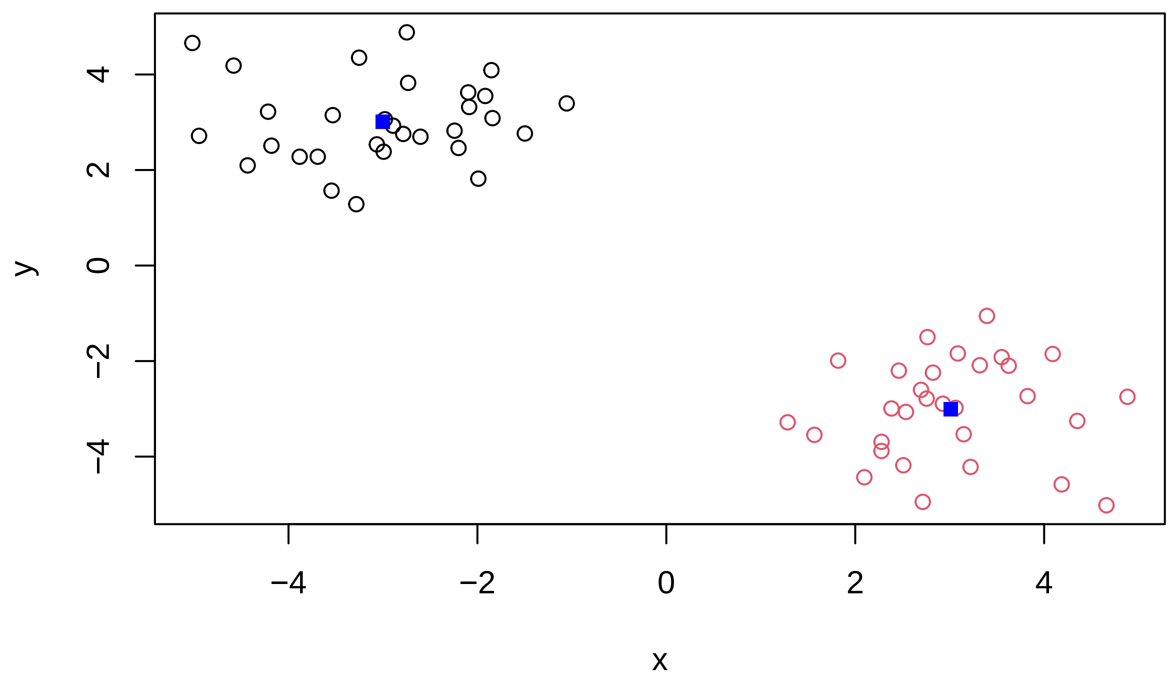
Q. What about cluster centers?

```
k$centers
```

```
##           x         y
## 1 -3.004677  3.010006
## 2  3.010006 -3.004677
```

Now we got to the main results. Let's use them to plot our data with the kmeans() result.

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



Now for Hierachical Clustering

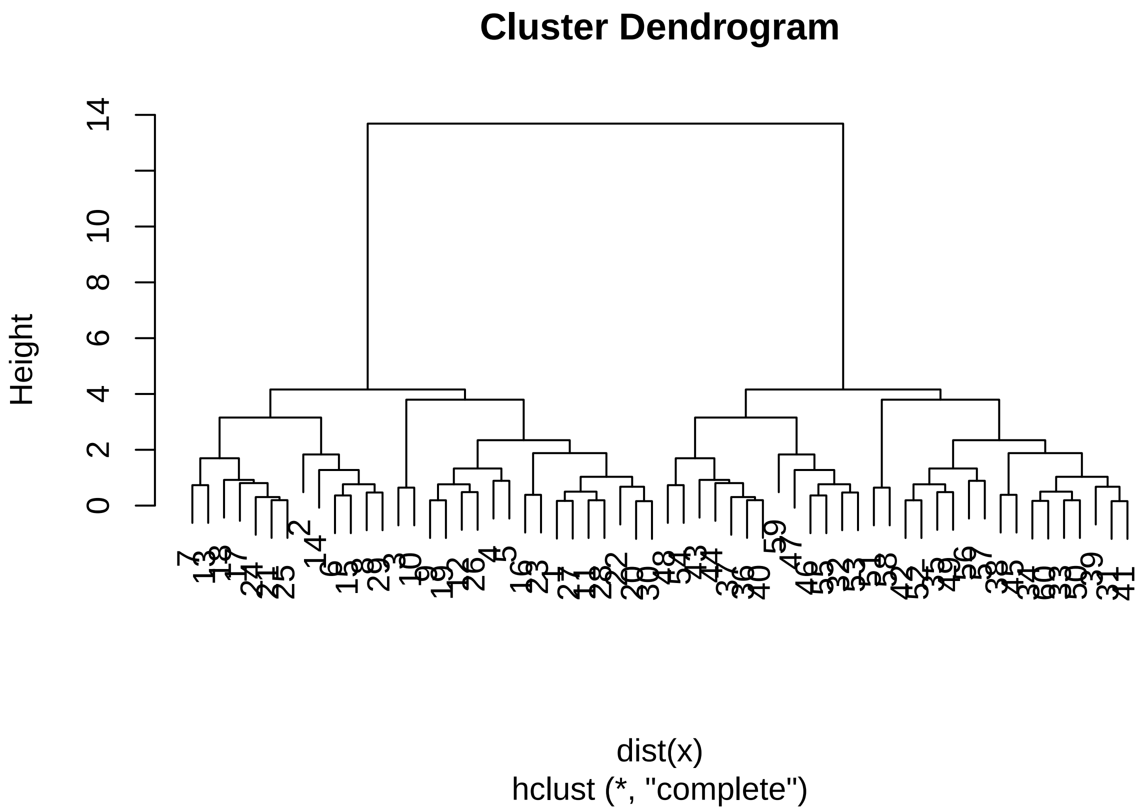
We will cluster the same data 'x' with the 'hclust()'. In this case, 'hclust()' requires a distance matrix as input.

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

```
##
## Call:
## hclust(d = dist(x))
##
## Cluster method   : complete
## Distance         : euclidean
## Number of objects: 60
```

Let's plot our hclust() result

```
plot(hc)
```



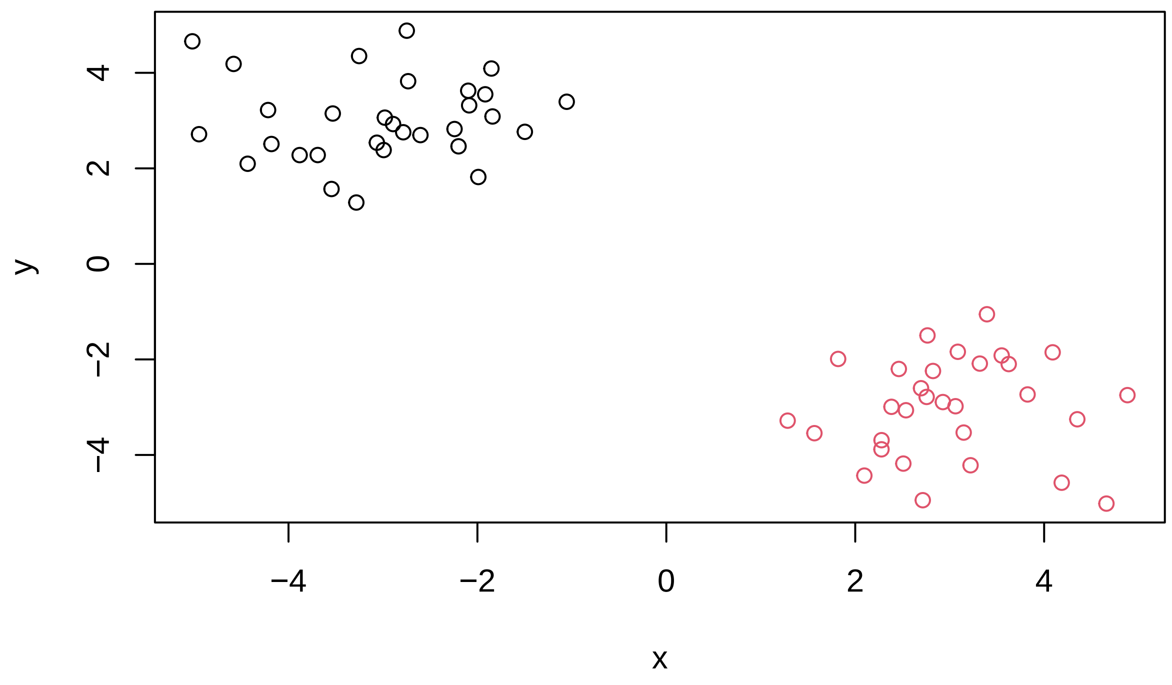
To get our cluster membership vector, we need to "cut" the dendrogram tree with 'cutree()'

```
grps <- cutree(hc, h = 8)
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
```

Now plot our data with the hclust() results.

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



Principal Component Analysis (PCA)

PCA of UK Food Data

Read data from website and try a few visualizations.

```
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
```

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

```r
nrow(x)
```

```
## [1] 17
```

```r
ncol(x)
```

```
## [1] 5
```

17 rows and 5 columns.

Preview the first 6 rows

```r
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
```

```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
```

Checking the dimensions again

```
dim(x)
```

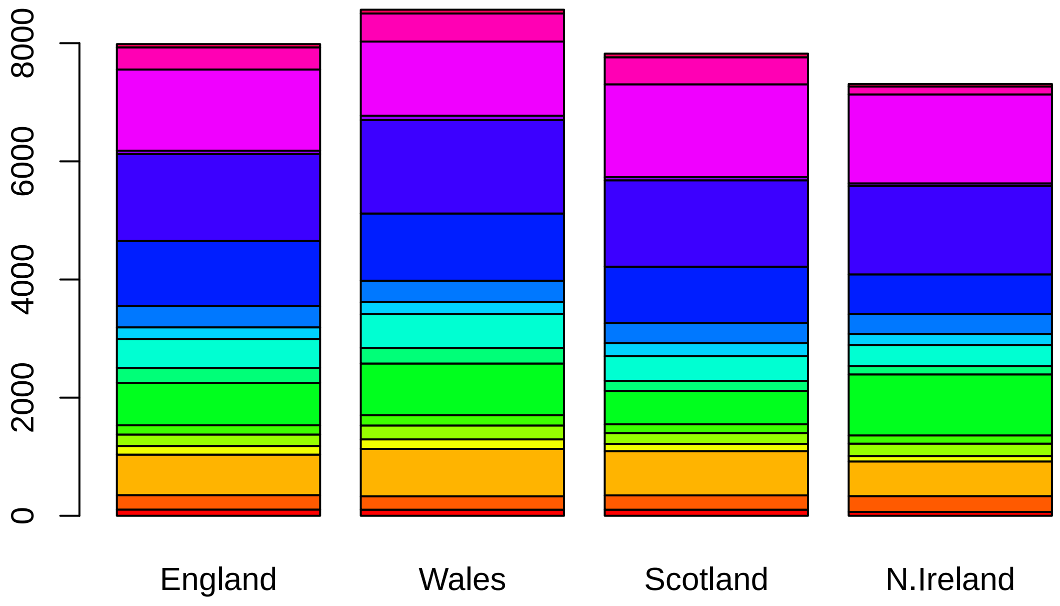
```
## [1] 17  4
```

17 rows, 4 columns

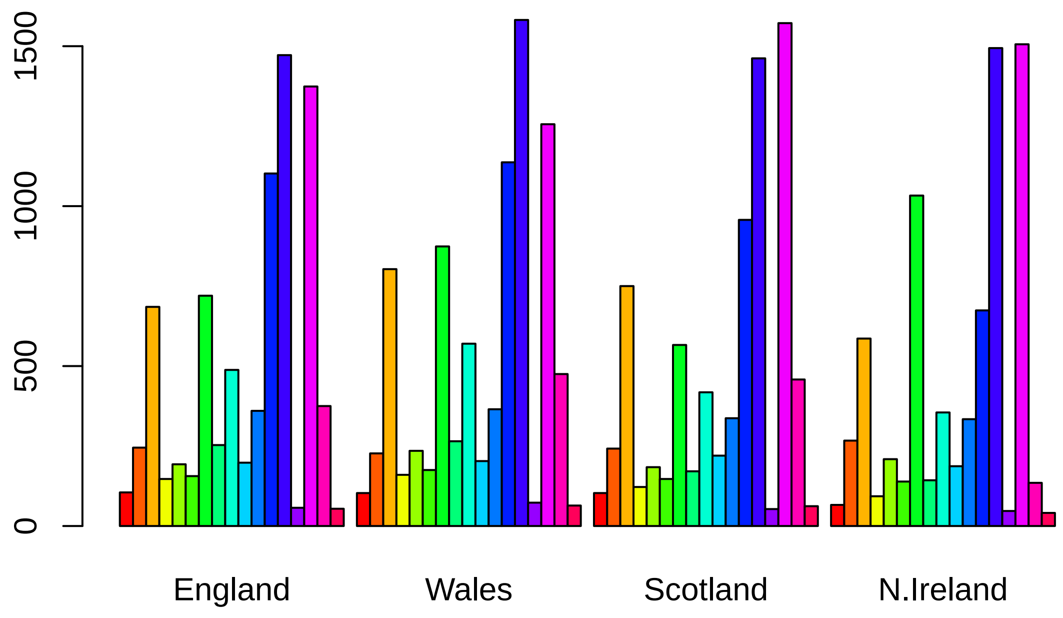
> Q2. Which approach to solving the 'row-names problem' above do you prefer and why? Is one approach more robust than another under certain circumstances?

I prefer the 'x <- read.csv(url, row.names = 1)' because if you run code block 'x <- x[,-1]' multiple times, it will change what column will be your column 1.

```
cols <- rainbow(nrow(x))
barplot(as.matrix(x), col = cols)
```



```
barplot(as.matrix(x), col = cols, beside = TRUE)
```

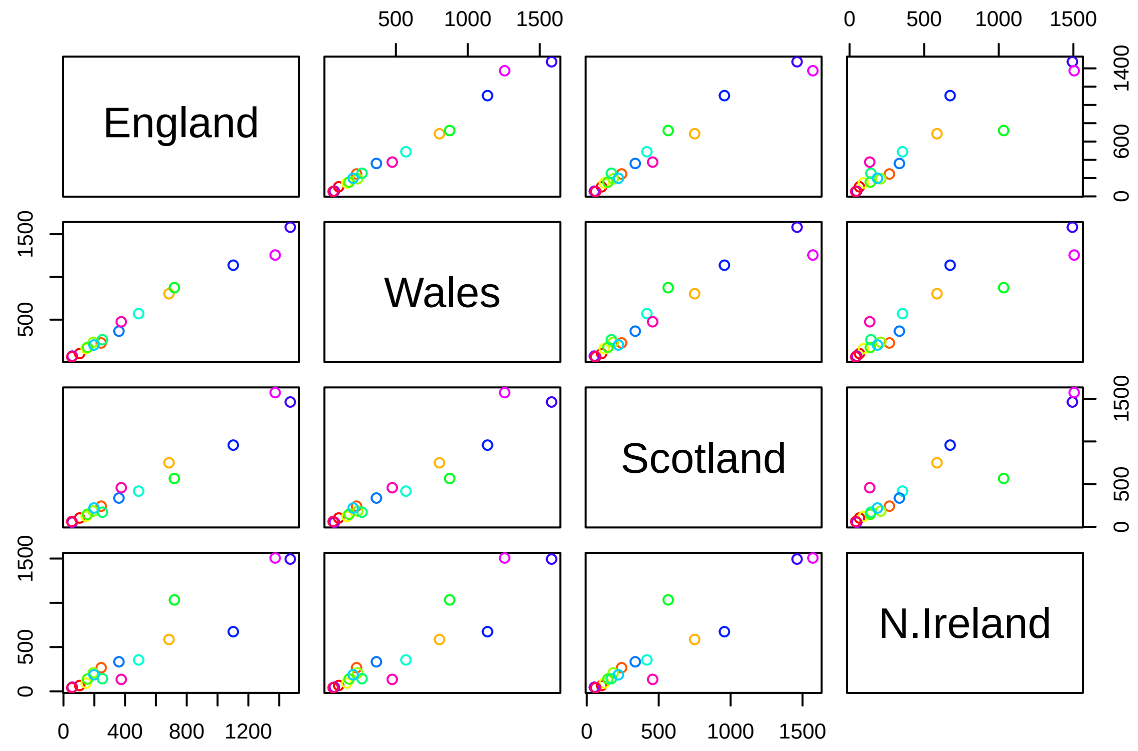


O3. Changing what optional argument in the barplot() function results in stacked bars?

beside = FALSE

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 = cols)
```



The closer the points are on the diagonal, the more similar the variables are to each other. If a given point lies on the diagonal, that means that the 2 variables being compared have the same values.

> Q6 What is the main difference between N. Ireland and the other countries of the UK in terms of this dataset?

Northern Ireland is more different from all of the other UK countries than those countries are from each other.

PCA to the rescue!!

The main base PCA function is called 'prcomp()'. We will need to give it the transpose of our input data.

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

There is a nice summary of how well PCA is doing.

```
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
```

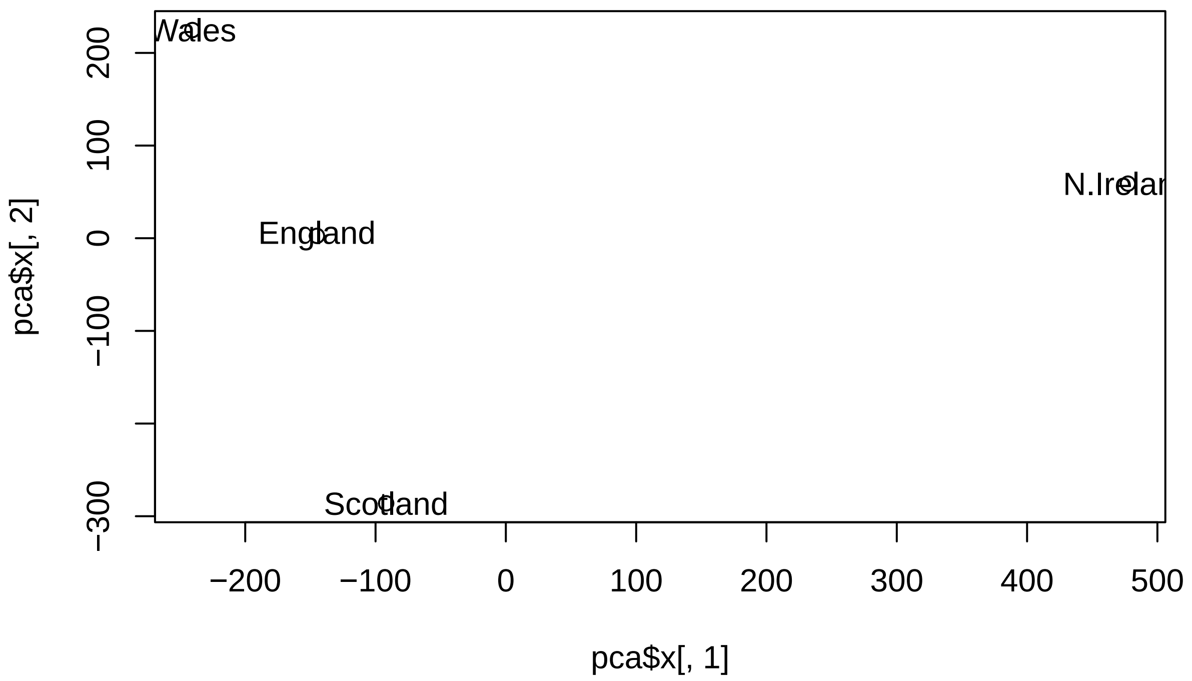
```
attributes(pca)
```

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

To make our new PCA plot (a.k.a. PCA score plot), we access 'pca$x'

> Q7. Generate a plot of PC1 vs PC2.

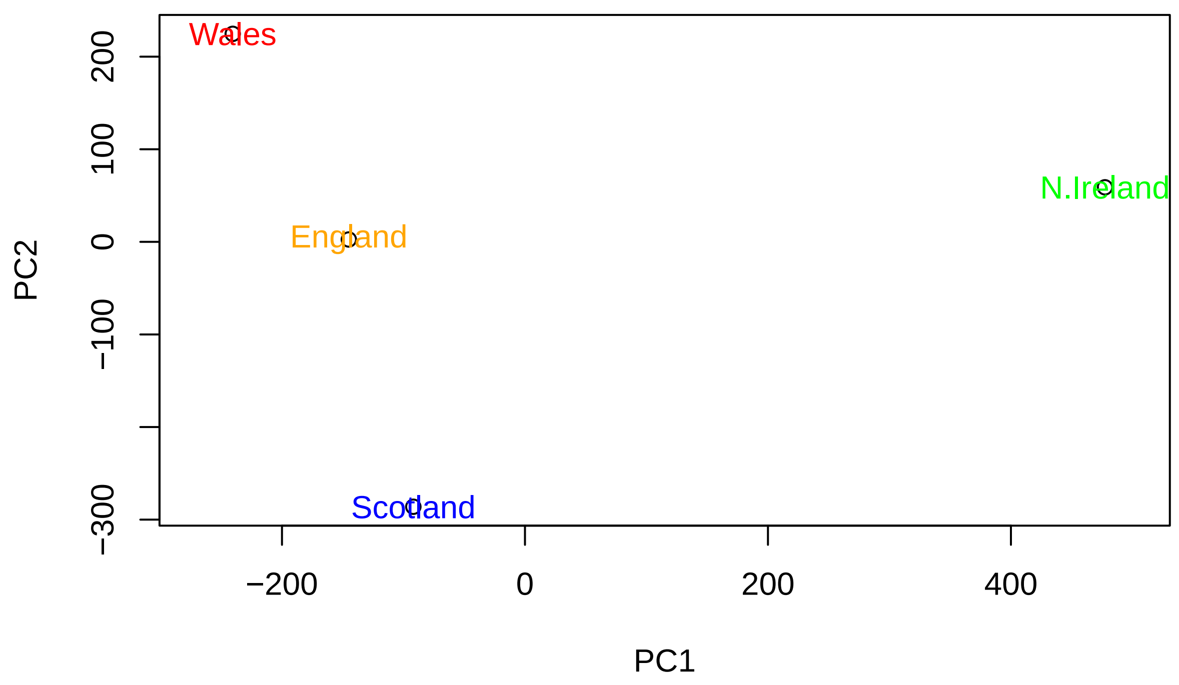
```
plot(pca$x[,1], pca$x[,2])
text(pca$x[,1], pca$x[,2], colnames(x))
```



> Q8. Customize your plot so that the colors of the country names match the colors in our UK and Ireland map and table at the start of this document.

Color up the plot

```
country_cols <- c("orange", "red", "blue", "green")
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 = country_cols)
```



Calculating how much variation in the original data each PC accounts for. Take the square of pcs$dev, which stands for "standard deviation."

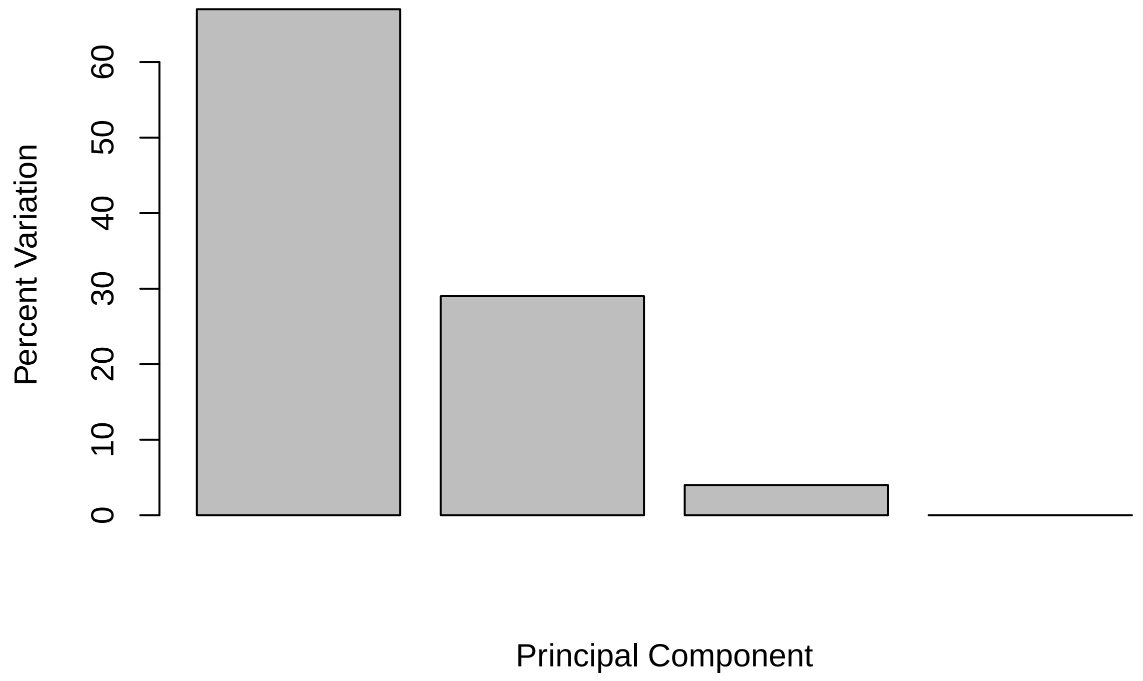
```
v <- round (pca$sdev^2/sum(pca$sdev^2) * 100)
v
```

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

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

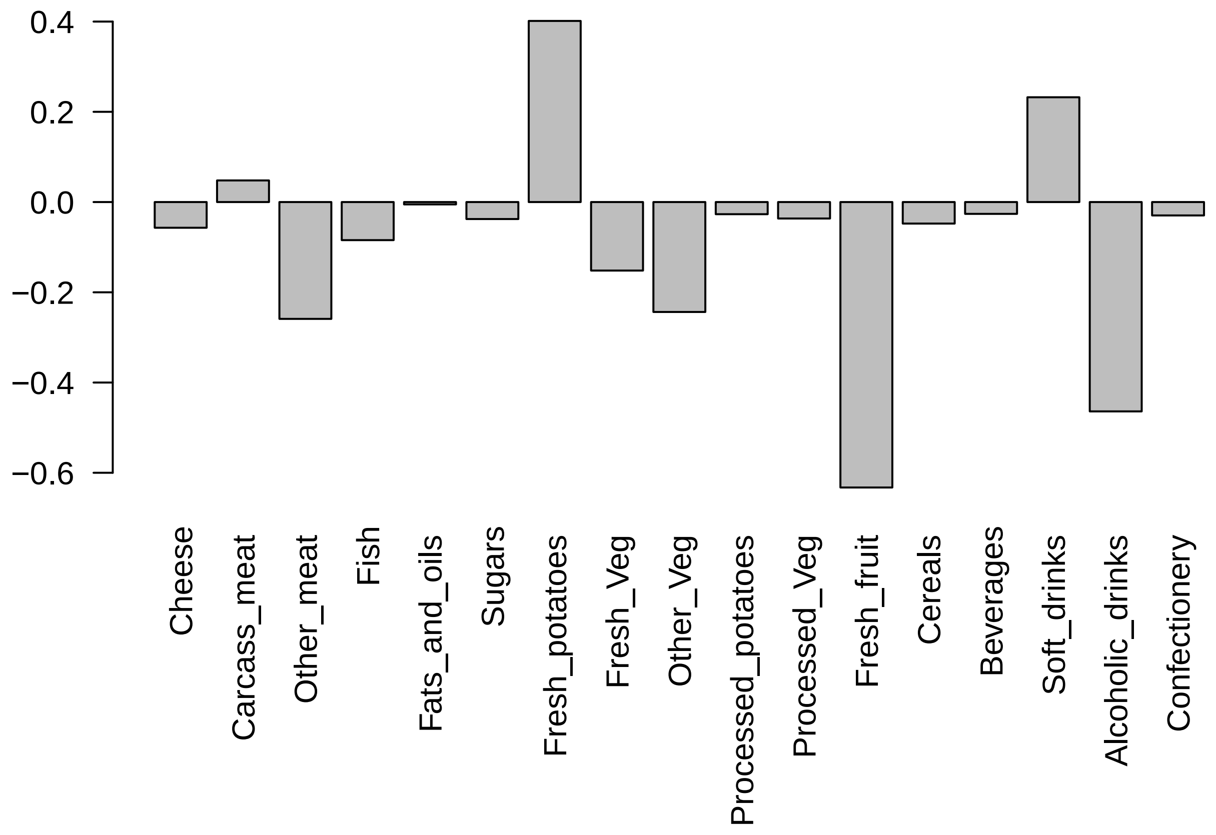
```
##                              PC1       PC2      PC3          PC4
## Standard deviation     324.15019 212.74780 73.87622 4.188568e-14
## Proportion of Variance   0.67444   0.29052  0.03503 0.000000e+00
## Cumulative Proportion    0.67444   0.96497  1.00000 1.000000e+00
```

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



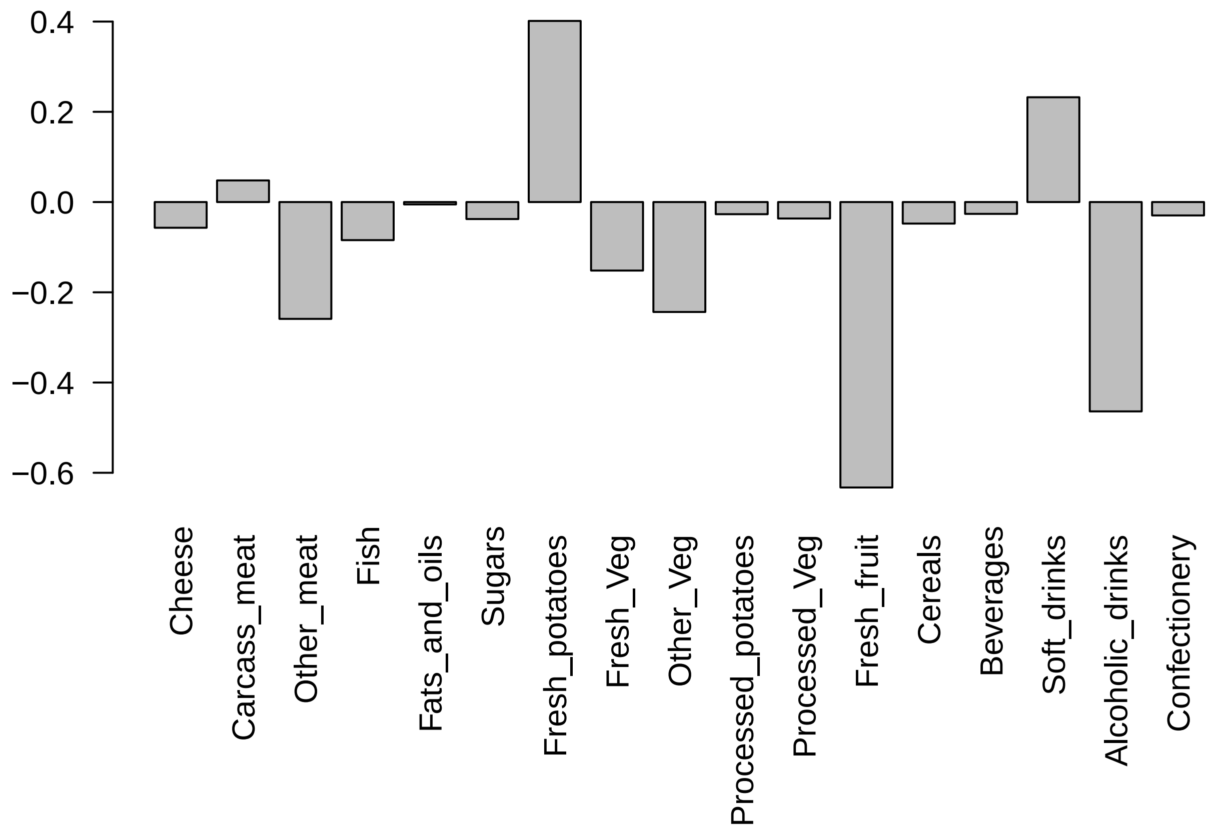
Digging Deeper (variable loadings)

```r
## 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 prominently, and what does PC2 mainly tell us about?

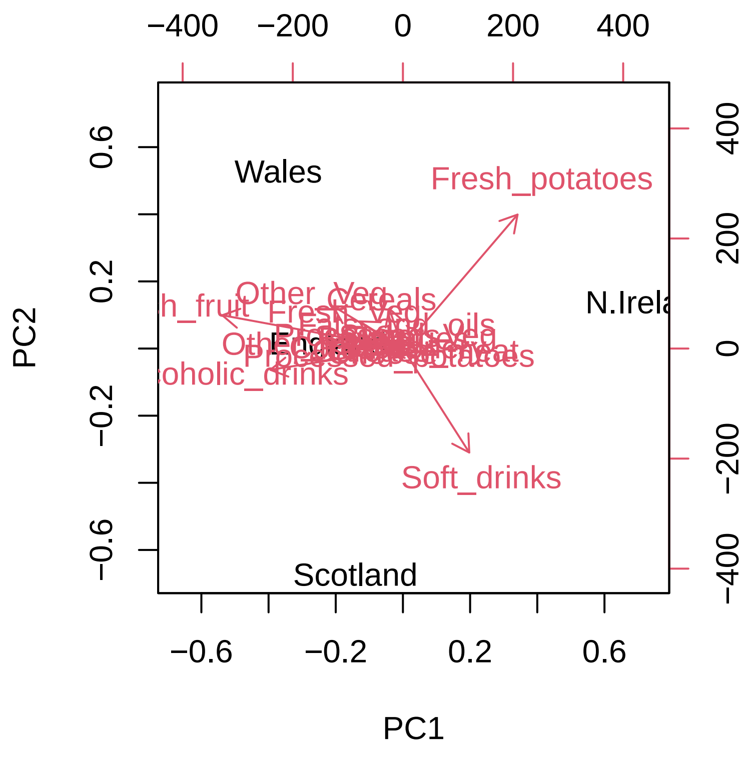
```
par(mar=c(10, 3, 0.35, 0))
barplot( pca$rotation[,1], las=2 )
```



Fresh_fruit and Alcoholic_drinks feature most prominently. PC2 accounts for the second-most amount of variation of the data.

Biplots

```
biplot(pca)
```



PCA of RNA-seq Data

Read in data from website

```
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)
```

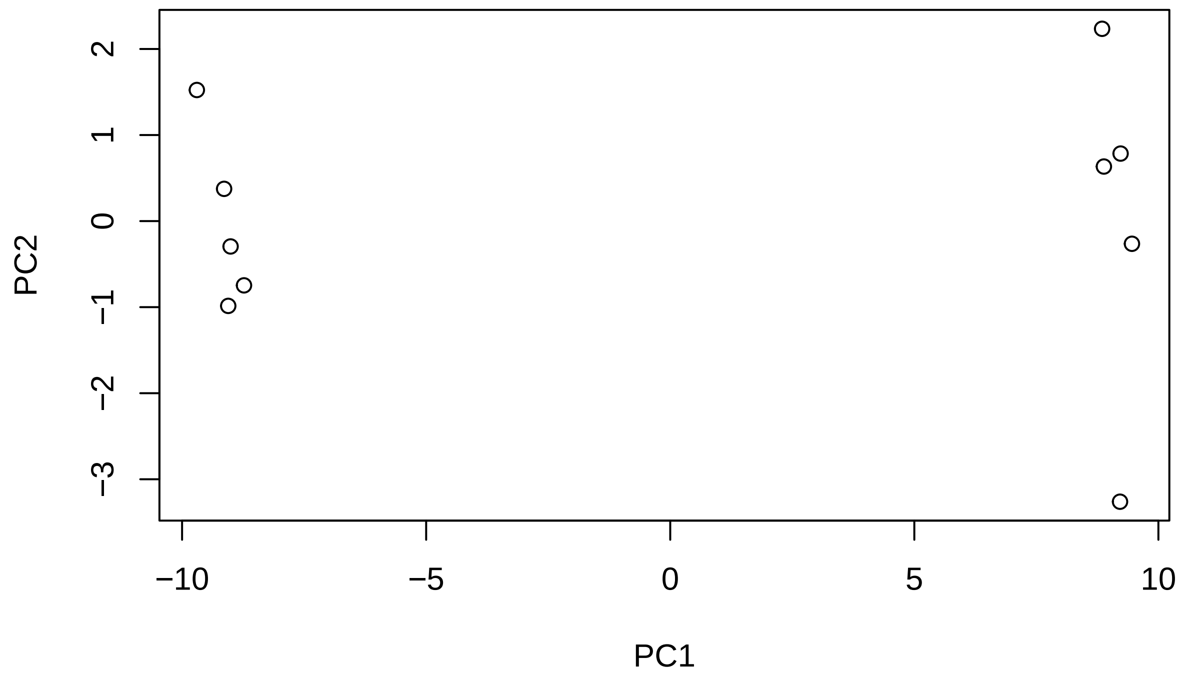
```
## [1] 100
```

```r
pca <- prcomp(t(rna.data), scale = TRUE)
summary(pca)
```

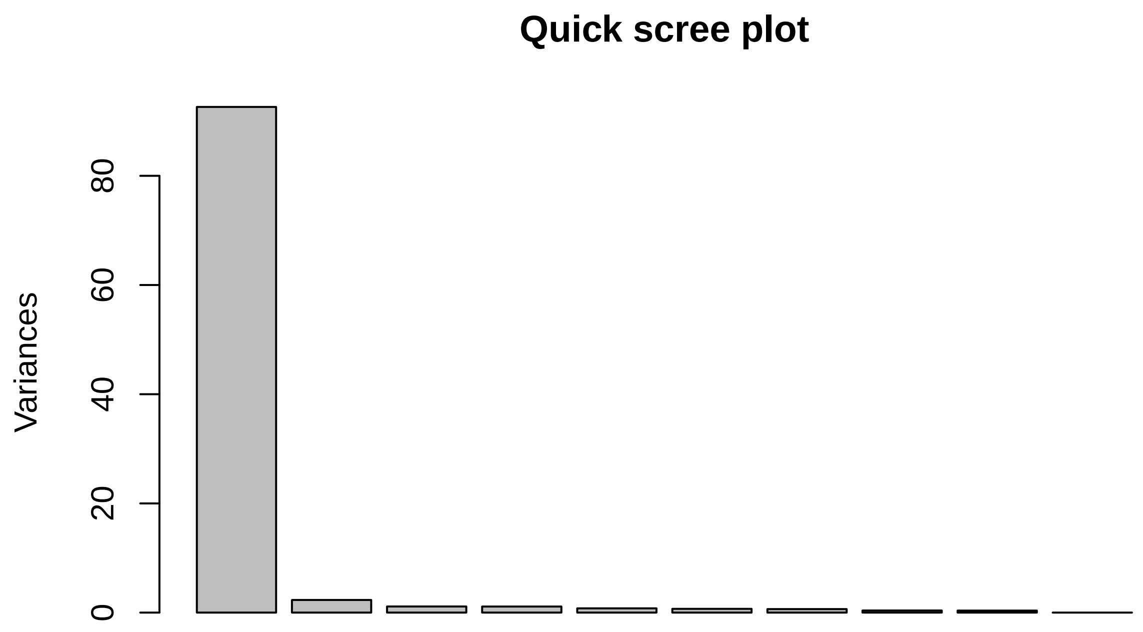
```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.6237 1.5198 1.05787 1.05203 0.88062 0.82545 0.80111
## Proportion of Variance 0.9262 0.0231 0.01119 0.01107 0.00775 0.00681 0.00642
## Cumulative Proportion  0.9262 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
```

Do our PCA plot of this RNA-Seq data

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



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

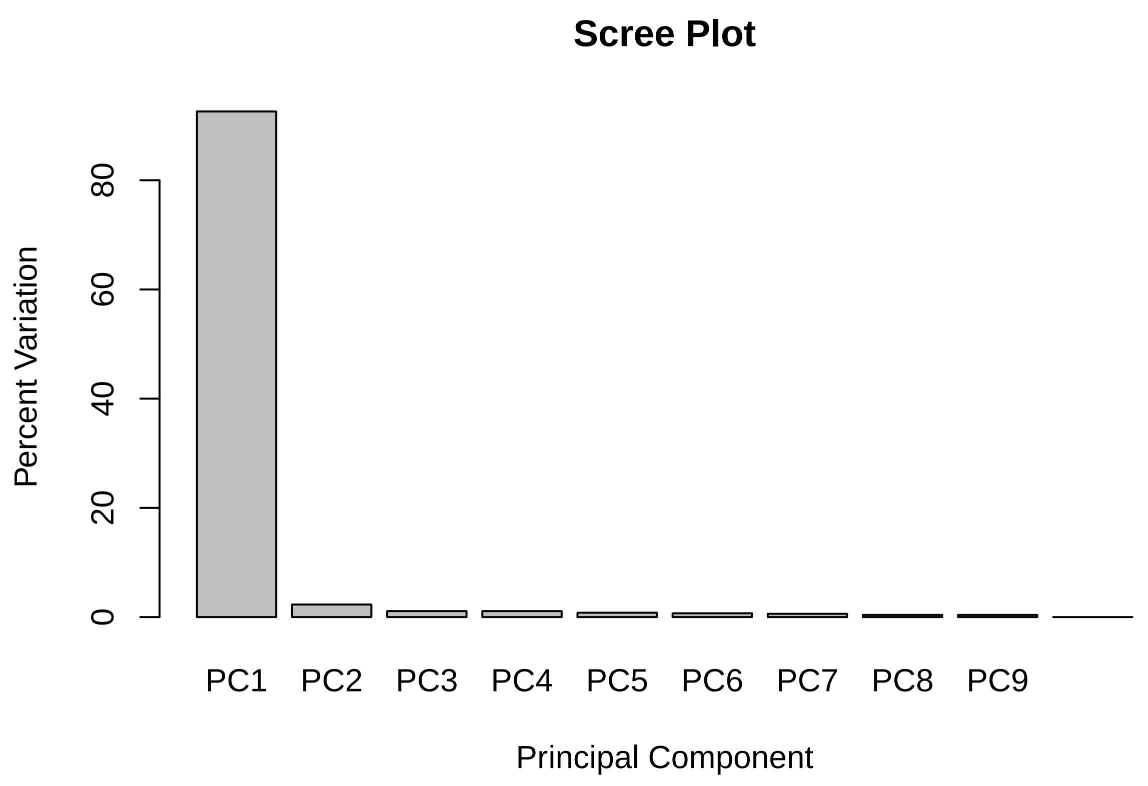


Making the above scree plot ourselves

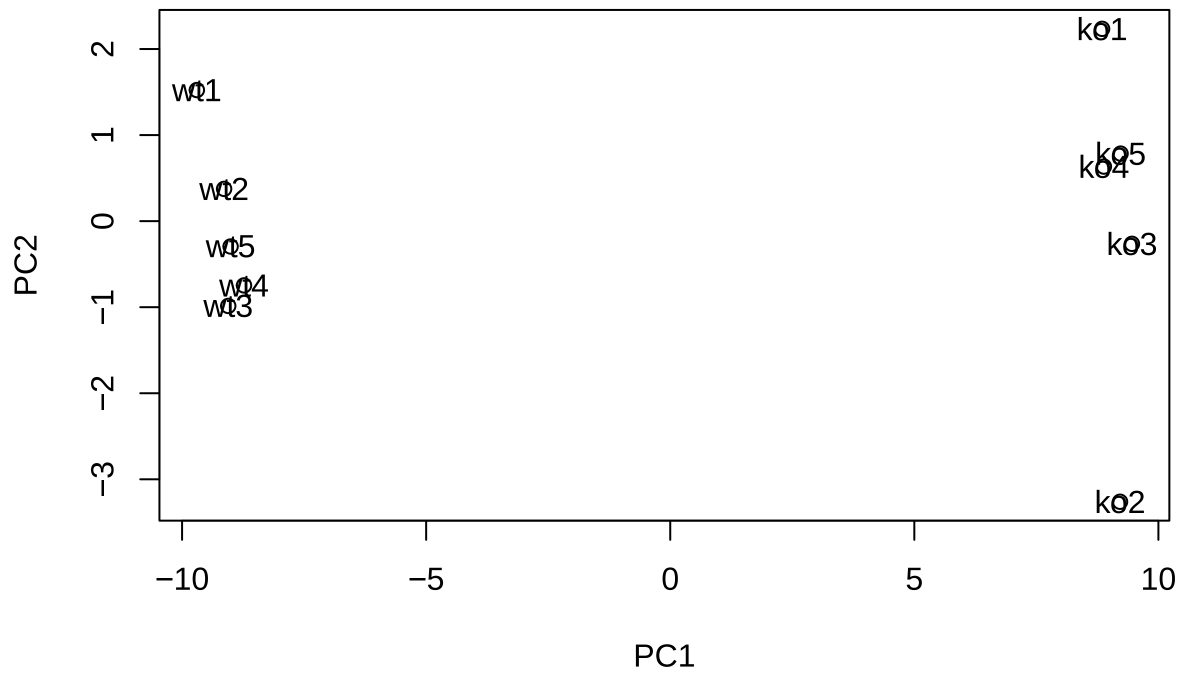
```
## Variance captured per PC
pca.var <- pca$sdev^2

## 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
```

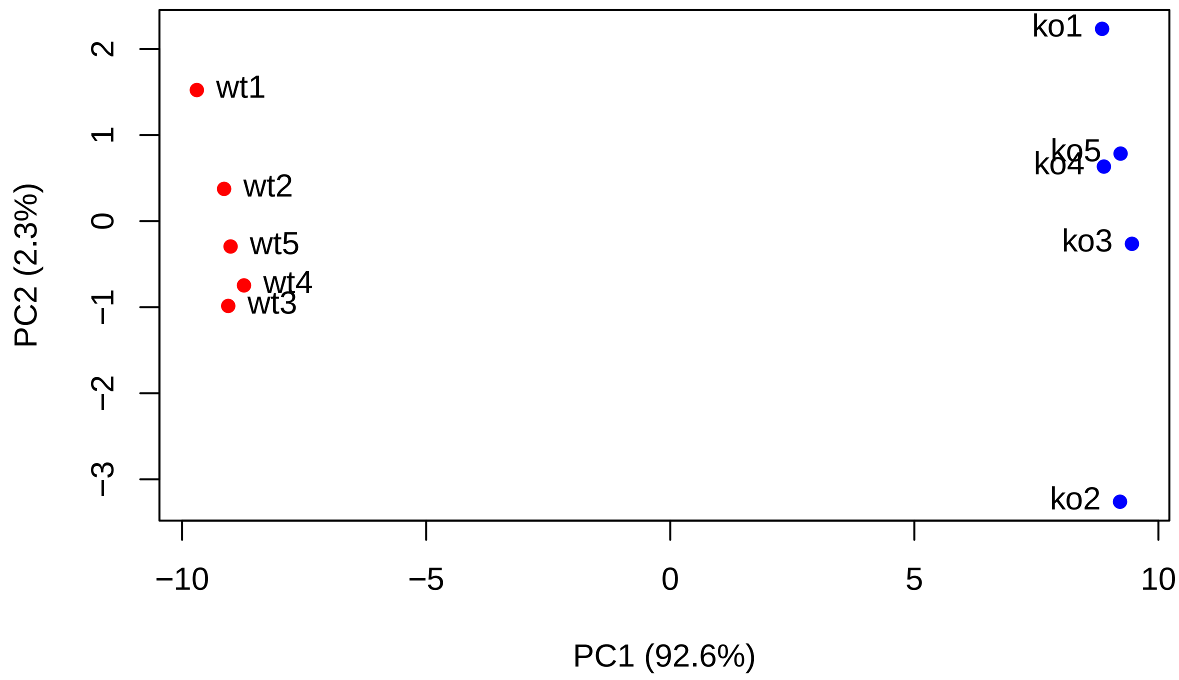
```
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



```
plot(pca$x[,1], pca$x[,2], xlab = "PC1", ylab = "PC2")
text(pca$x[,1], pca$x[,2], colnames(rna.data))
```

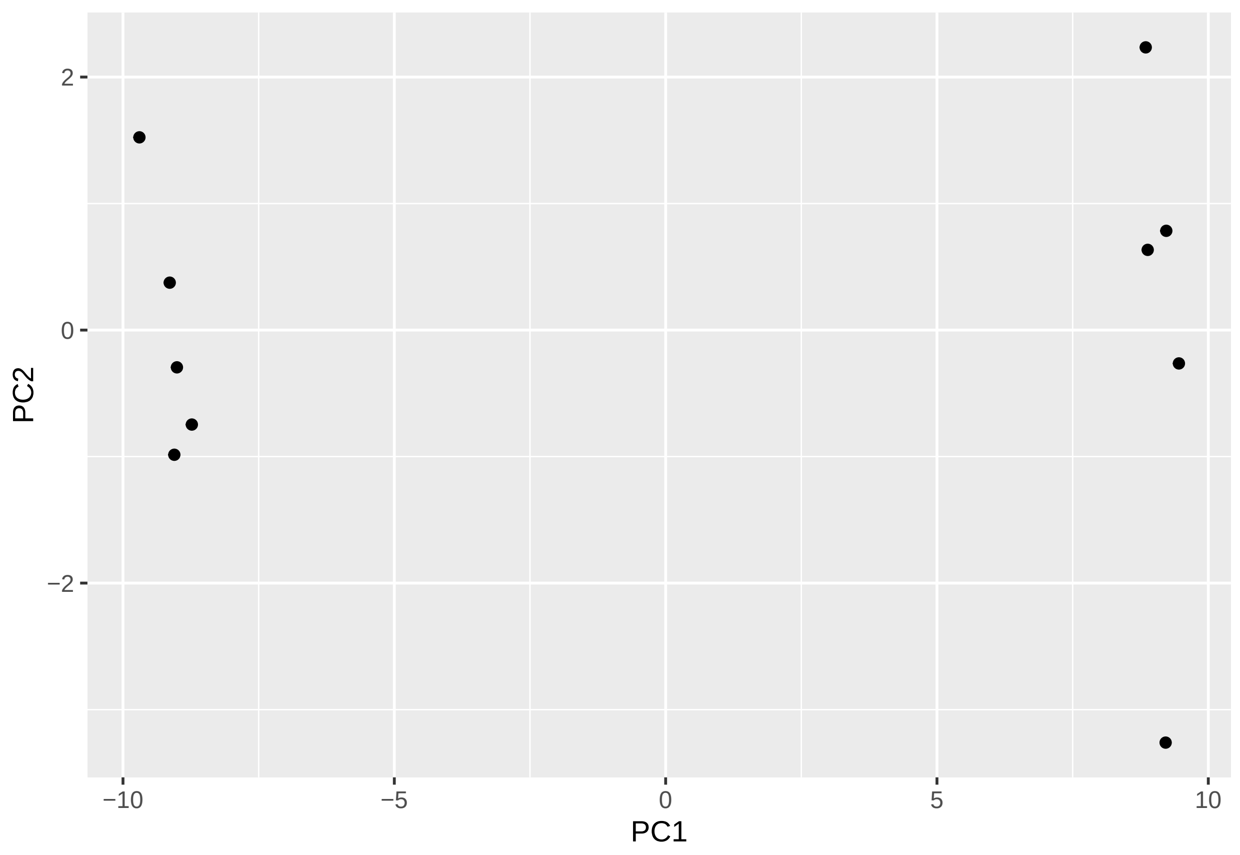


```
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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)))
```

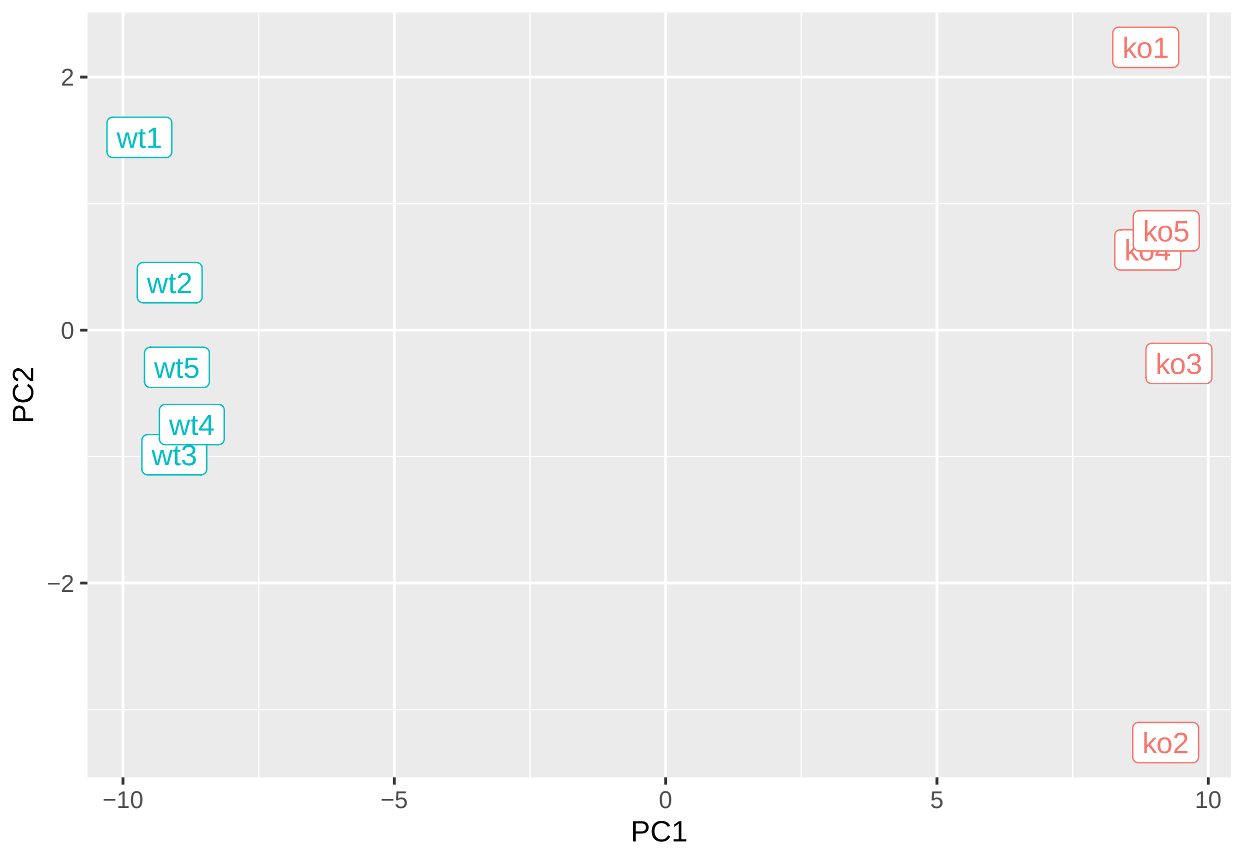


Using ggplot

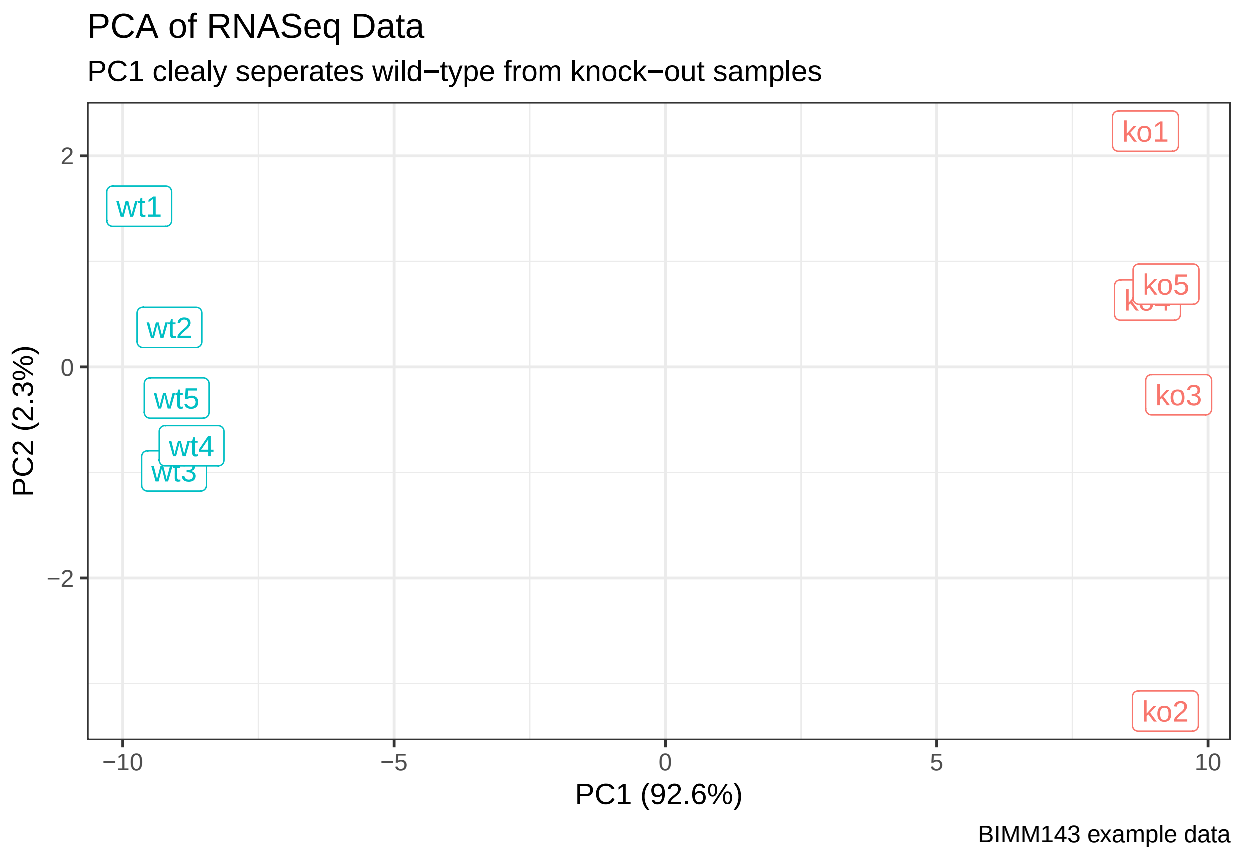
```r
library(ggplot2)

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

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



```r
# 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="BIMM143 example data") +
     theme_bw()
```



Optional: Gene Loadings

Finding the top 10 measurements (genes) that contribute most to pc1 in either direction (+ or -)

```
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"
```