Class 07: Machine Learning 1 & PCA Lab

Seong Tae Gwon

2/14/2022

Part 1. Machine Learning 1 (Following the Tuesday's lab review video)

First up kmeans(): Demo using kmeans() fucntion in base R

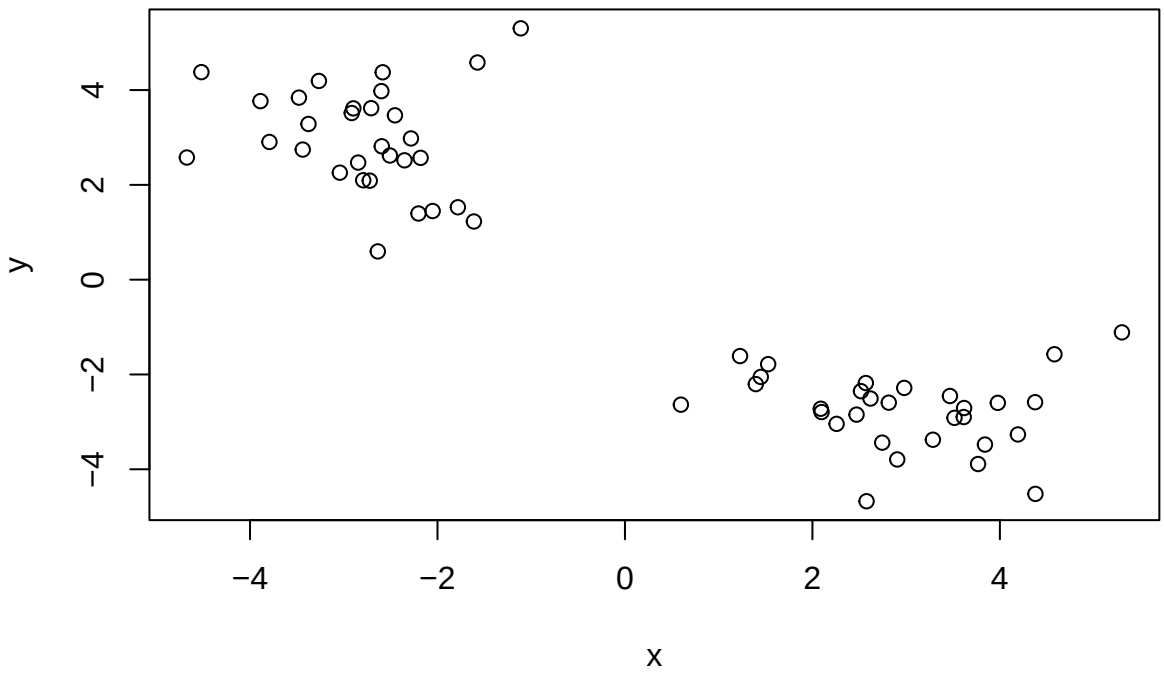
First make up some data with a known structure.

```
tmp <- c(rnorm(30,-3), rnorm(30,3))
x <- cbind(x=tmp, y=rev(tmp))
x
```

```
##                x          y
##  [1,] -1.6115752  1.2279429
##  [2,] -2.2030345  1.3959836
##  [3,] -3.0417992  2.2578015
##  [4,] -1.7825462  1.5277415
##  [5,] -3.4782201  3.8407806
##  [6,] -3.7930558  2.9050364
##  [7,] -3.3764399  3.2852185
##  [8,] -2.1801854  2.5699251
##  [9,] -3.4376639  2.7449388
## [10,] -3.2652663  4.1924784
## [11,] -2.5075731  2.6196297
## [12,] -2.3521676  2.5180662
## [13,] -2.7077415  3.6183651
## [14,] -3.8889867  3.7672183
## [15,] -4.5189583  4.3788602
## [16,] -2.9149310  3.5154736
## [17,] -2.6367429  0.5965891
## [18,] -2.4539144  3.4671005
## [19,] -2.0516195  1.4489648
## [20,] -2.5947008  2.8143849
## [21,] -2.7226336  2.0892372
## [22,] -4.6741944  2.5774021
## [23,] -2.5847987  4.3754889
## [24,] -2.2831427  2.9794204
## [25,] -2.8465686  2.4705936
## [26,] -2.5987847  3.9776937
## [27,] -2.8977791  3.6143606
## [28,] -1.5735457  4.5819692
```

```
## [29,] -2.7924097  2.0976675
## [30,] -1.1116413  5.3022764
## [31,]  5.3022764 -1.1116413
## [32,]  2.0976675 -2.7924097
## [33,]  4.5819692 -1.5735457
## [34,]  3.6143606 -2.8977791
## [35,]  3.9776937 -2.5987847
## [36,]  2.4705936 -2.8465686
## [37,]  2.9794204 -2.2831427
## [38,]  4.3754889 -2.5847987
## [39,]  2.5774021 -4.6741944
## [40,]  2.0892372 -2.7226336
## [41,]  2.8143849 -2.5947008
## [42,]  1.4489648 -2.0516195
## [43,]  3.4671005 -2.4539144
## [44,]  0.5965891 -2.6367429
## [45,]  3.5154736 -2.9149310
## [46,]  4.3788602 -4.5189583
## [47,]  3.7672183 -3.8889867
## [48,]  3.6183651 -2.7077415
## [49,]  2.5180662 -2.3521676
## [50,]  2.6196297 -2.5075731
## [51,]  4.1924784 -3.2652663
## [52,]  2.7449388 -3.4376639
## [53,]  2.5699251 -2.1801854
## [54,]  3.2852185 -3.3764399
## [55,]  2.9050364 -3.7930558
## [56,]  3.8407806 -3.4782201
## [57,]  1.5277415 -1.7825462
## [58,]  2.2578015 -3.0417992
## [59,]  1.3959836 -2.2030345
## [60,]  1.2279429 -1.6115752
```

```
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 -2.762754  2.958620
## 2  2.958620 -2.762754
##
## 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 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
##
## Within cluster sum of squares by cluster:
## [1] 54.97308 54.97308
##  (between_SS / total_SS =  89.9 %)
##
## Available components:
##
## [1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss"
## [6] "betweenss"    "size"         "iter"         "ifault"
```

Q. How many points are in each cluster?

```
k$size
```

Answer:

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

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

```
k$cluster
```

Answer:

```
##  [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
```

Q. What about cluster centers?

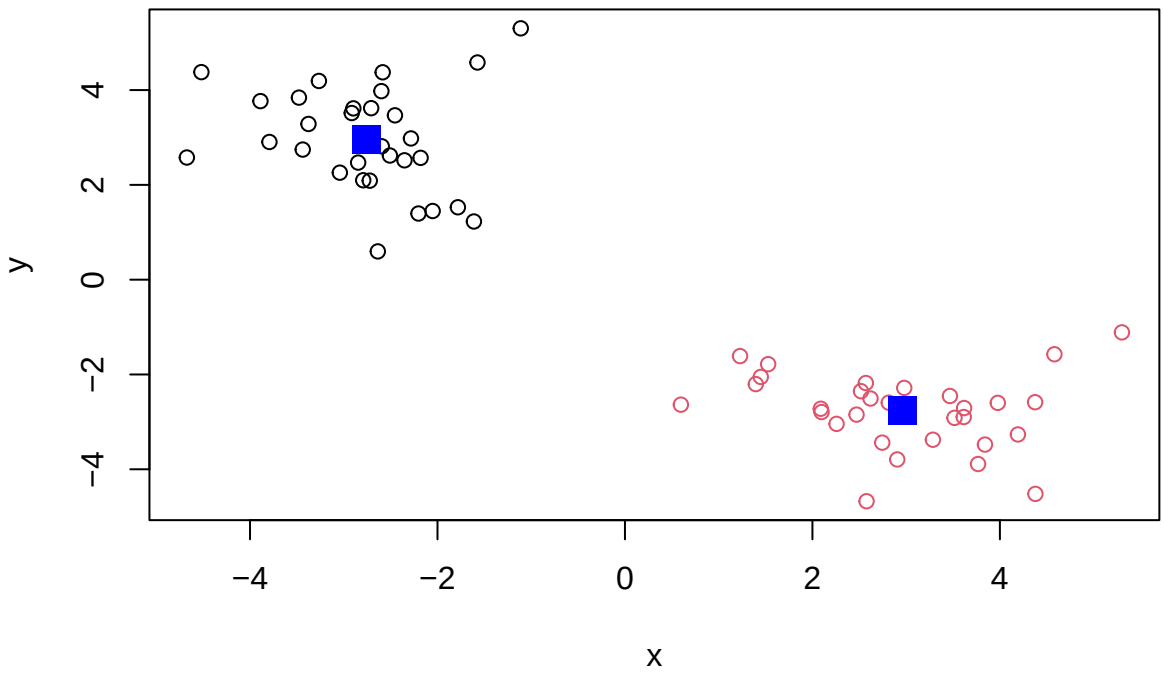
```
k$centers
```

Answer:

```
##           x         y
## 1 -2.762754  2.958620
## 2  2.958620 -2.762754
```

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

```
plot(x, col=k$cluster)
points(k$centers, col="blue", pch=15, cex=2)
```



Now for Hierarchical Clustering

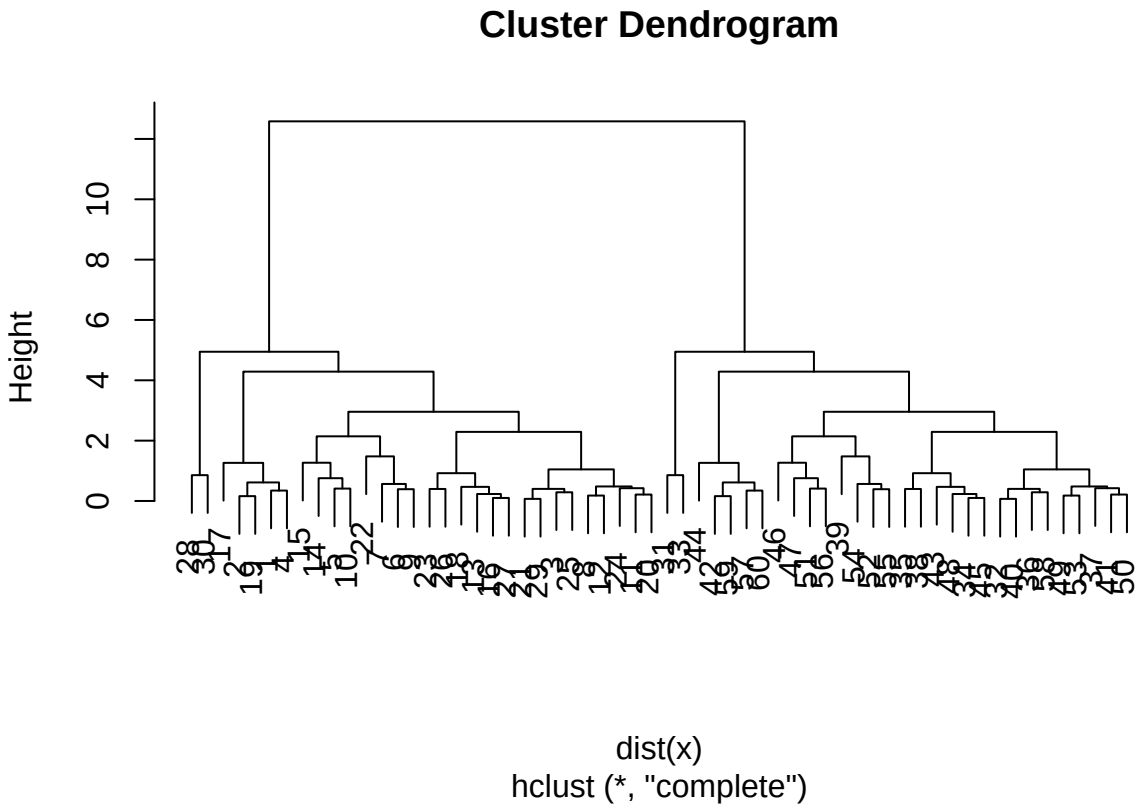
We will cluster the same data `x` with the `hclust()`. In this case, `hclust()` requires a dsitance matrix as input.

```
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)
```



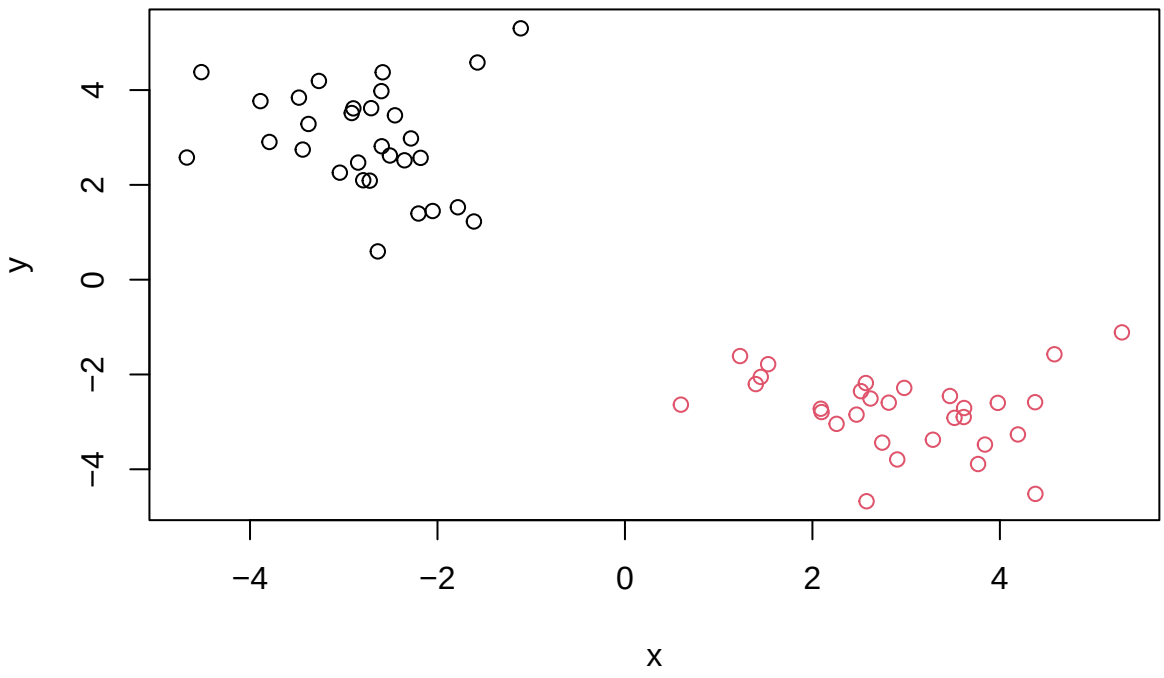
To get our cluster membership vector, we need to "cut" the tree with the `cutree()`.

```
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 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
```

Now, plot our data with the hclust() results.

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



Part 2. Hands on with Principal Component Analysis (PCA)

1. PCA of UK food data

Data import

Read the provided UK_foods.csv input file.

```
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?

```
nrow(x)
```

Answer:

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

```
ncol(x)
```

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

Checking your data

Use the `View()` function to display all the data, or the `head()` and `tail()` functions to print only a portion of the data (by default 6 rows from either the top or bottom of the data set respectively).

```
#View(x)
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
```

Hmm, it looks like the row-names here were not set properly as we were expecting 4 columns (one for each of the 4 countries of the UK - not 5 as reported from the dim() function). Fix this up with 2 following methods.

One method: set the rownames() to the first column and then remove the troublesome first column (with the -1 column index).

```
# 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
```

Second method: set the row.names argument of read.csv() to be the first column

```
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
```

```
dim(x)
```

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

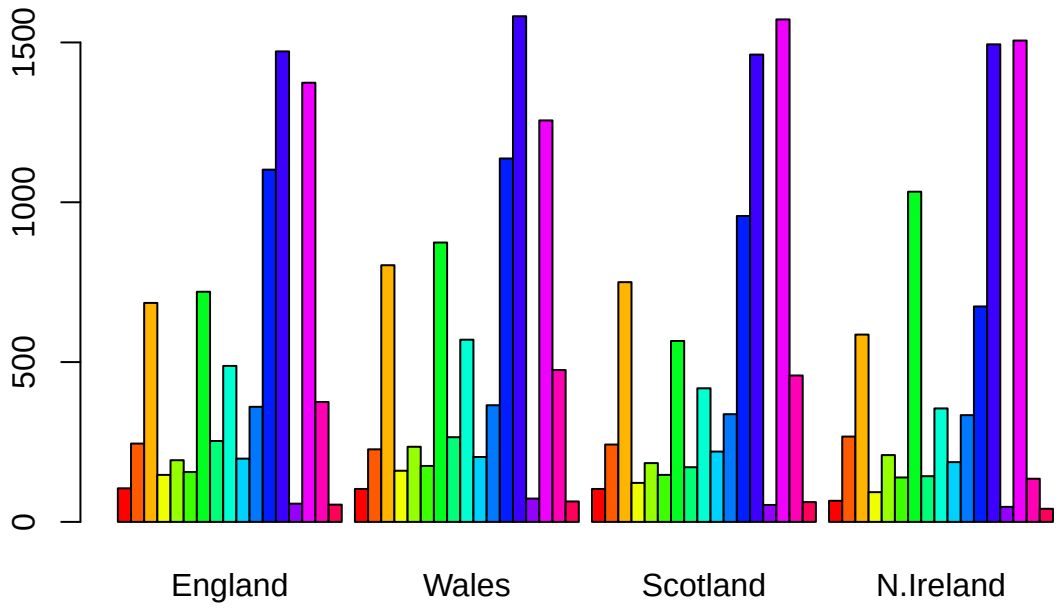
Q2. Which approach to solving the 'row-names problem' mentioned above do you prefer and why? Is one approach more robust than another under certain circumstances?

Answer: I prefer the second approach because the first approach deletes a column of the data set every time I run the code.

Spotting major differnces and trends

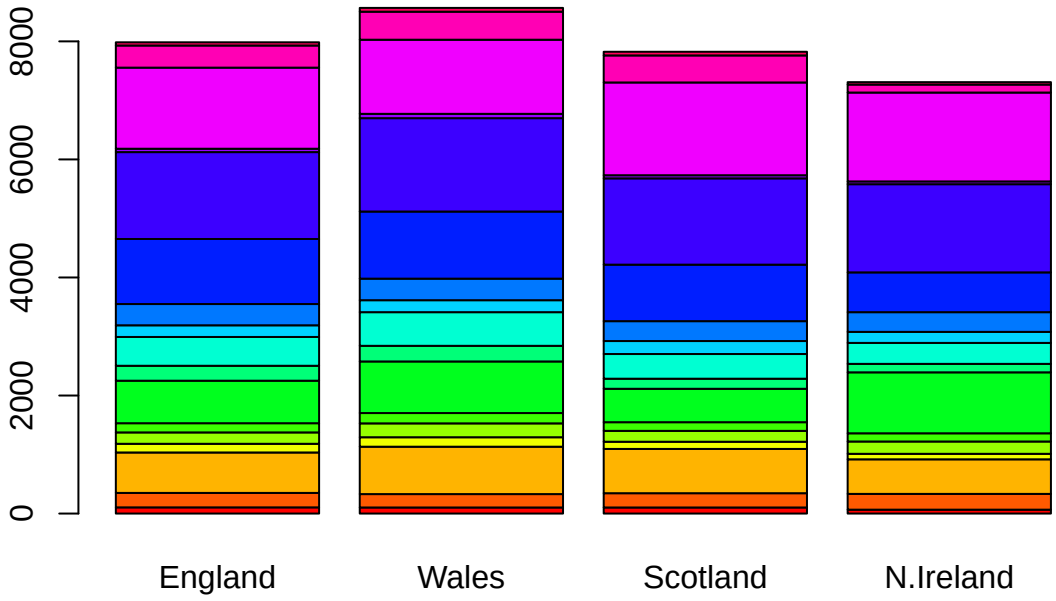
Quick glance of the data set with a regular bar plot.

```
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?

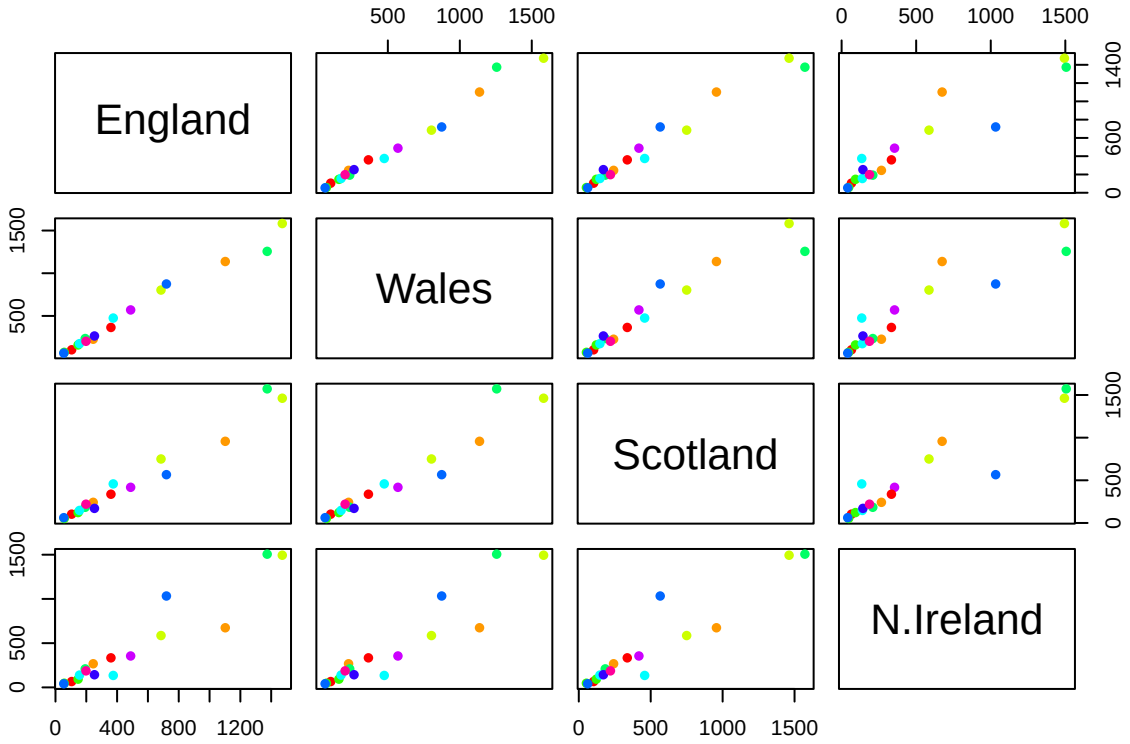
```
# Setting beside=F in the previous barplot() code
barplot(as.matrix(x), beside=F, col=rainbow(nrow(x)))
```



Answer:

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)
```



Answer: Each row represents a pairwise comparison between the labeled country (y-axis) and the corresponding country in each column (x-axis). For example, the second plot in the first row is a comparison between England and Wales. 17 different colors in each plot represent each value (food) of the column (country). The diagonal in each plot represents how similar x and y values are. If a given point doesn't lie on the diagonal, two countries have a difference in food data (lower the diagonal = y country has more, above the diagonal = x country has more).

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

Answer: N. Ireland's food data differs from the other countries since points in every plot don't lie on the diagonal.

PCA to the rescue

To perform PCA, we will use the base R `prcomp()` function. `prcomp()` expects the *observations* as rows and the *variables* as columns. First, we want to transpose our data.frame matrix with the `t()` transpose function.

```
# Use the prcomp() PCA function
pca <- prcomp( t(x) )
summary(pca)
```

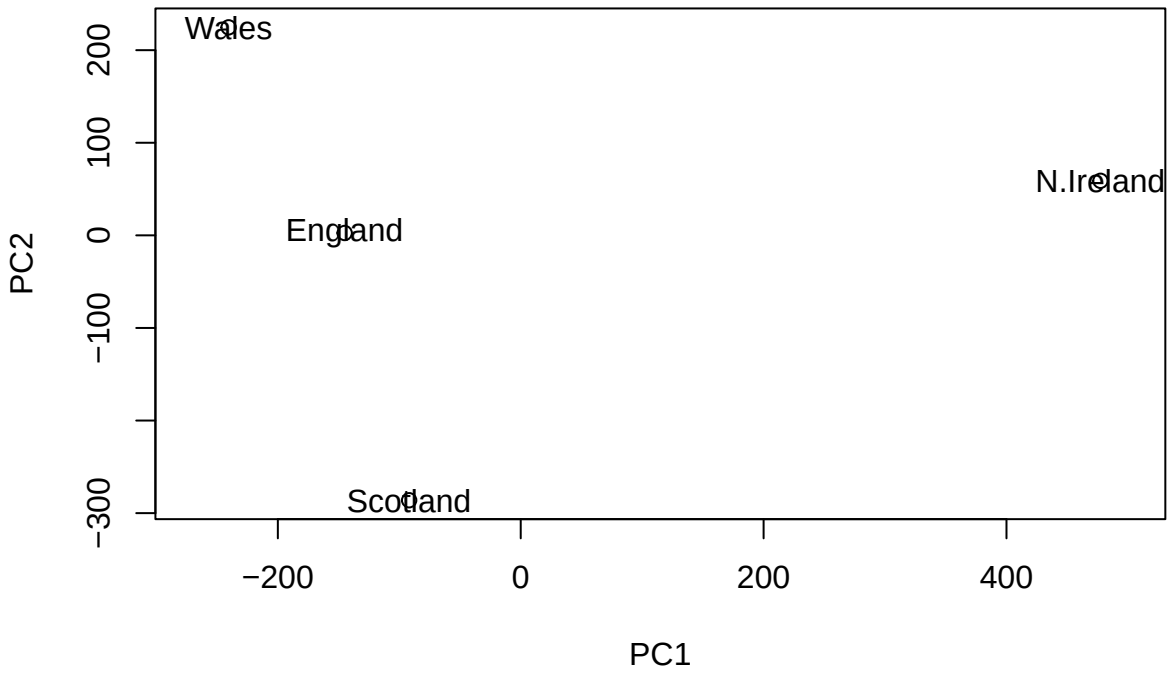
```
## Importance of components:
##                             PC1      PC2      PC3       PC4
## Standard deviation     324.1502 212.7478 73.87622 4.189e-14
## Proportion of Variance   0.6744   0.2905  0.03503 0.000e+00
## Cumulative Proportion    0.6744   0.9650  1.00000 1.000e+00
```

```
# Look inside the PCA object
attributes(pca)
```

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

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

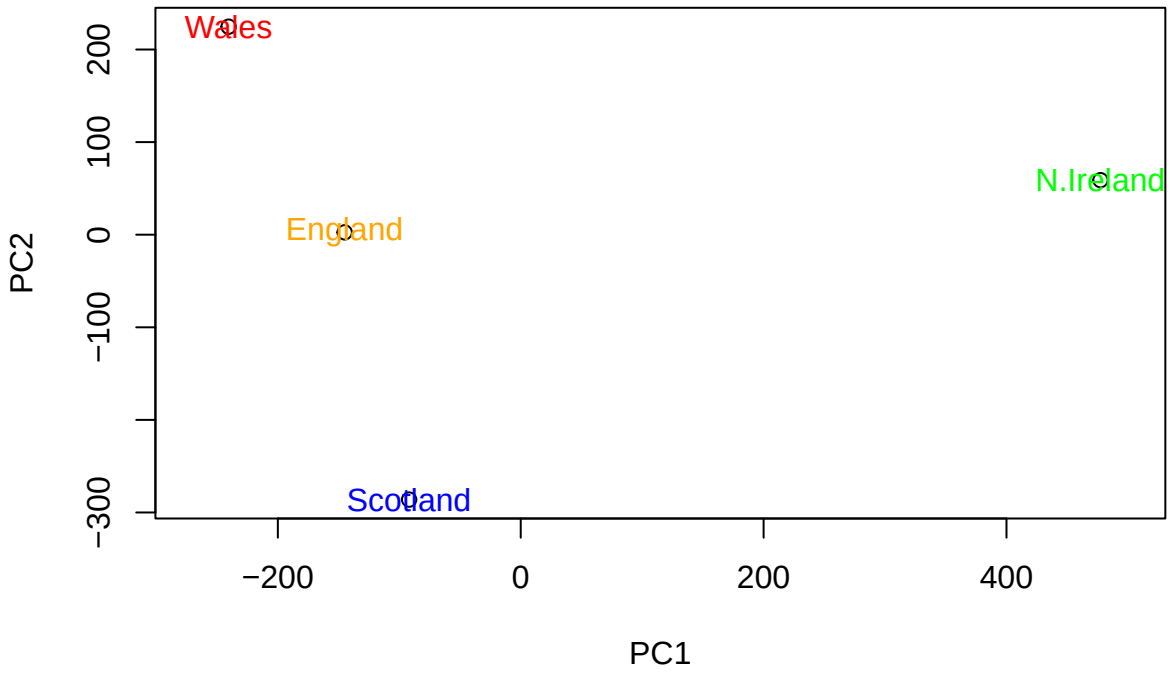
```
# To make our new PCA plot (a.k.a. PCA score plot), we access `pca$x`
# Plot PC1 vs PC2
plot(pca$x[,1], pca$x[,2], xlab="PC1", ylab="PC2", xlim=c(-270,500))
# Add column names to the plot
text(pca$x[,1], pca$x[,2], colnames(x))
```



Answer:

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))
color <- c("orange", "red", "blue", "green")
text(pca$x[,1], pca$x[,2], colnames(x), col=color)
```



Answer:

Once the principal components have been obtained, we can use them to map the relationship between variables (i.e. countries) in therms of these major PCs (i.e. new axis that maximally describe the original data variance).

Below we can use the square of pca$sdev , which stands for "standard deviation", to calculate how much variation in the original data each PC accounts for.

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

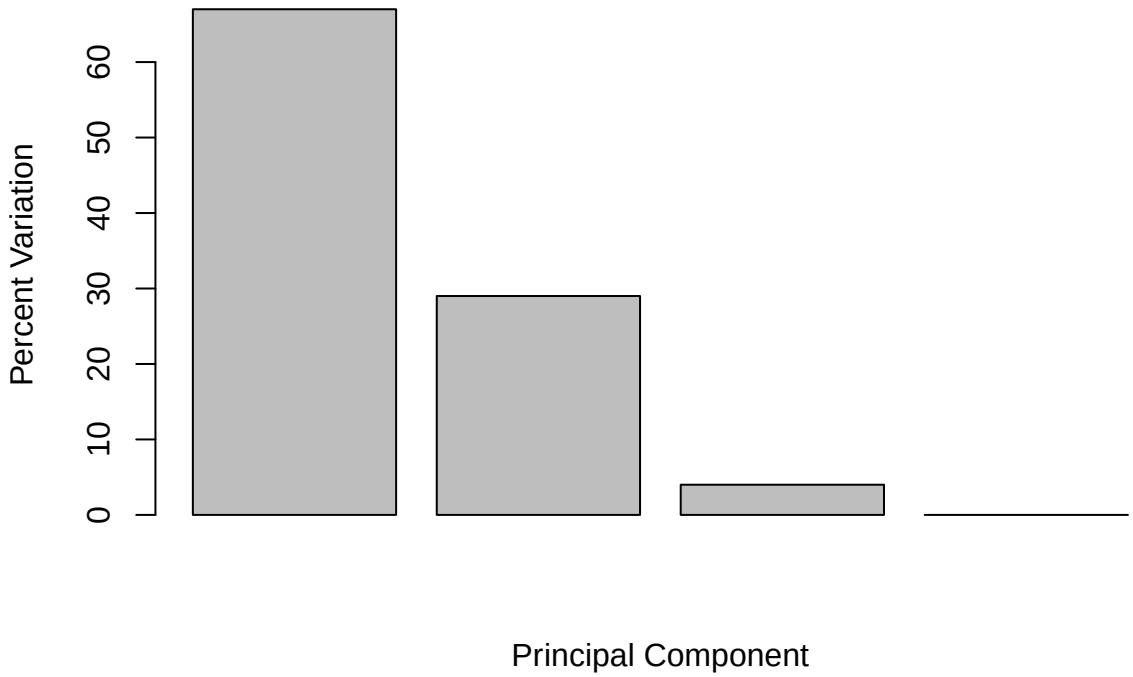
```
## [1] 67 29  4  0
```

```
## 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
```

This information can be summarized in a plot of the variances (eigenvalues) with respect to the principal component number (eigenvector number).

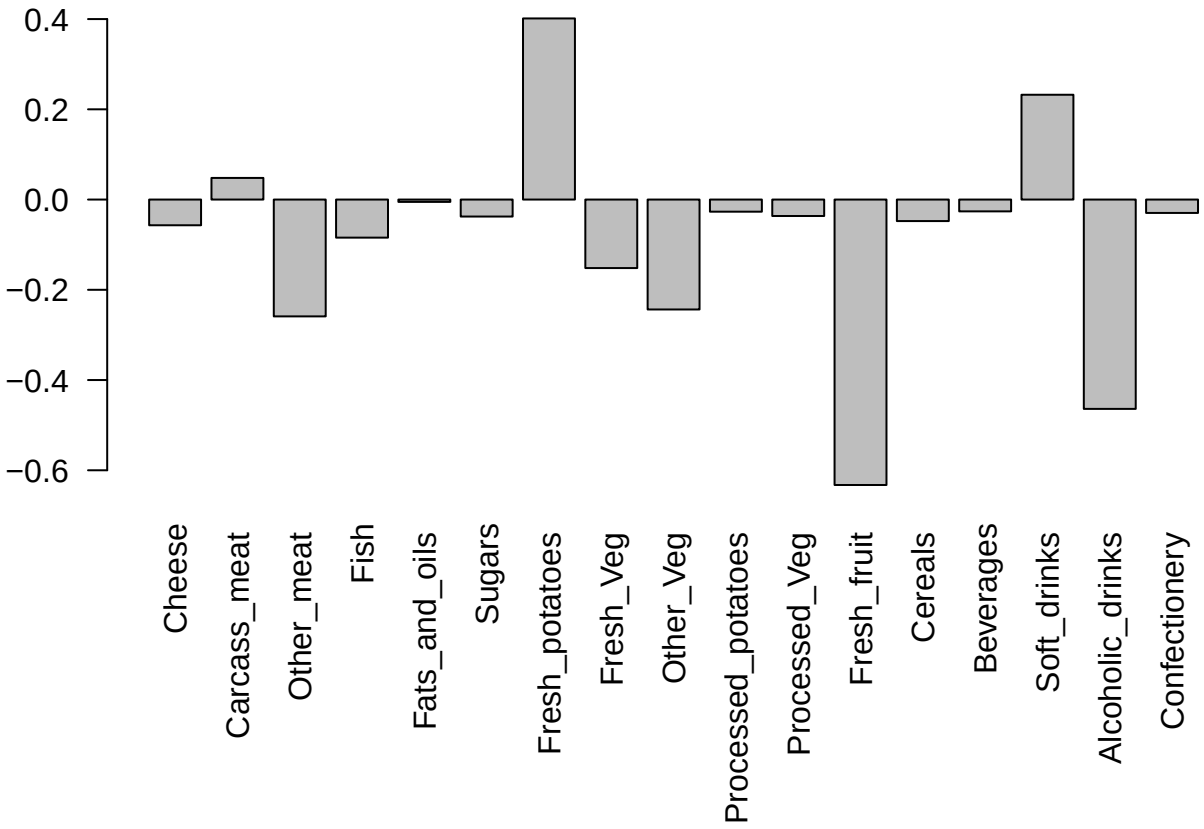
```
barplot(v, xlab="Principal Component", ylab="Percent Variation")
```



Digging deeper (variable loadings)

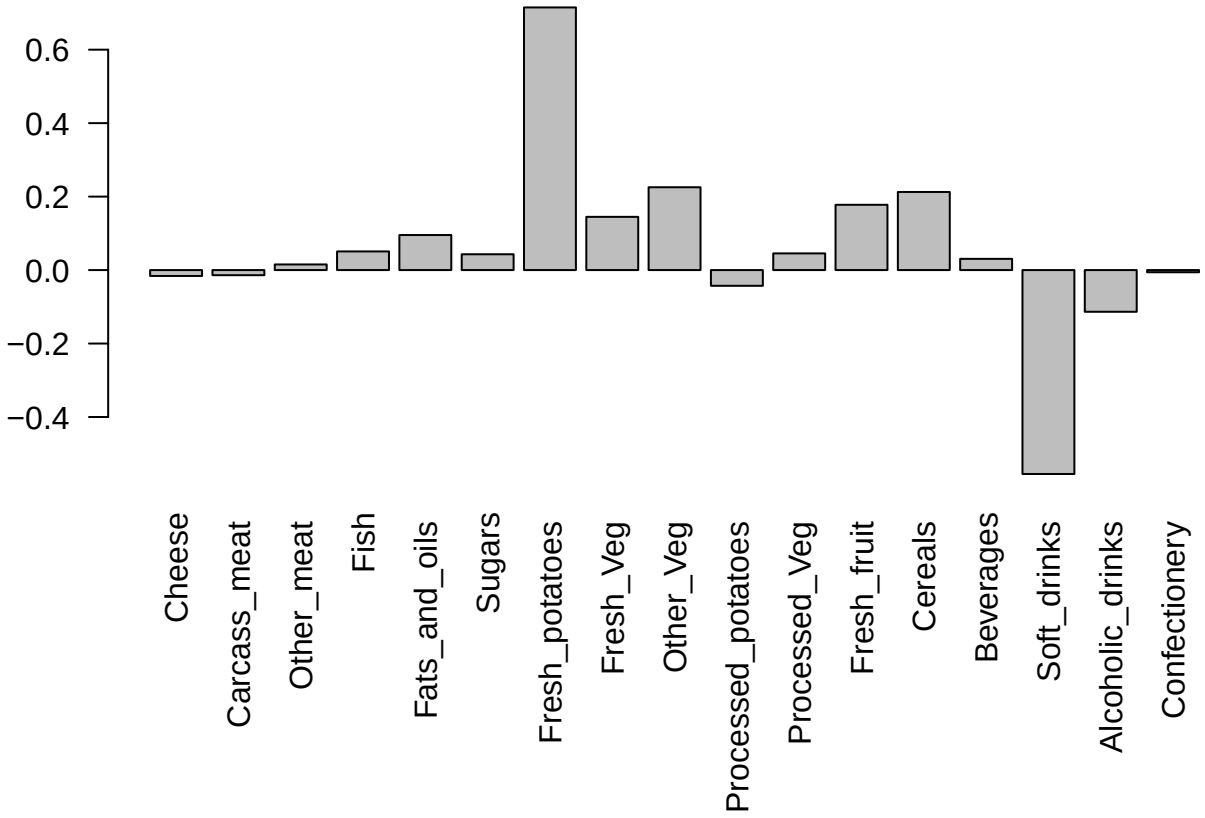
We can also consider the influence of each of the original variables upon the principal components (typically known as loading scores). This information can be obtained from the prcomp() returned $rotation component. It can also be summarized with a call to biplot().

```
## 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?

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

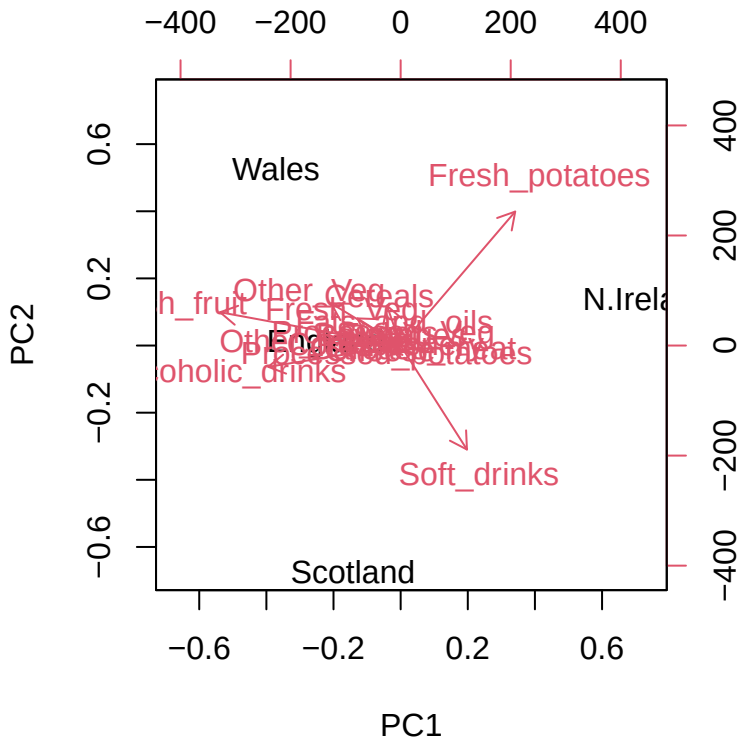


Answer:
Two food groups that feature prominently are `Fresh_potatoes` and `Soft_drinks`. PC2 mainly tells us that there is lower variance in the other food groups. This is illustrated by the similar distributions and loading scores closer to 0.

Biplots

Another way to see this information together with the main PCA plot.

```
# The inbuild biplot() can be useful for small datasets
biplot(pca)
```



Two food groups `Fresh_potatoes` and `Soft_drinks` feature prominently here.

2. PCA of RNA-seq data

Read a small RNA-seq count data set into a data frame called rna.data where the columns are individual samples (i.e. cells) and rows are measurements taken for all the samples (i.e. genes).

```
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?

```
# Samples
nrow(x)
```

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

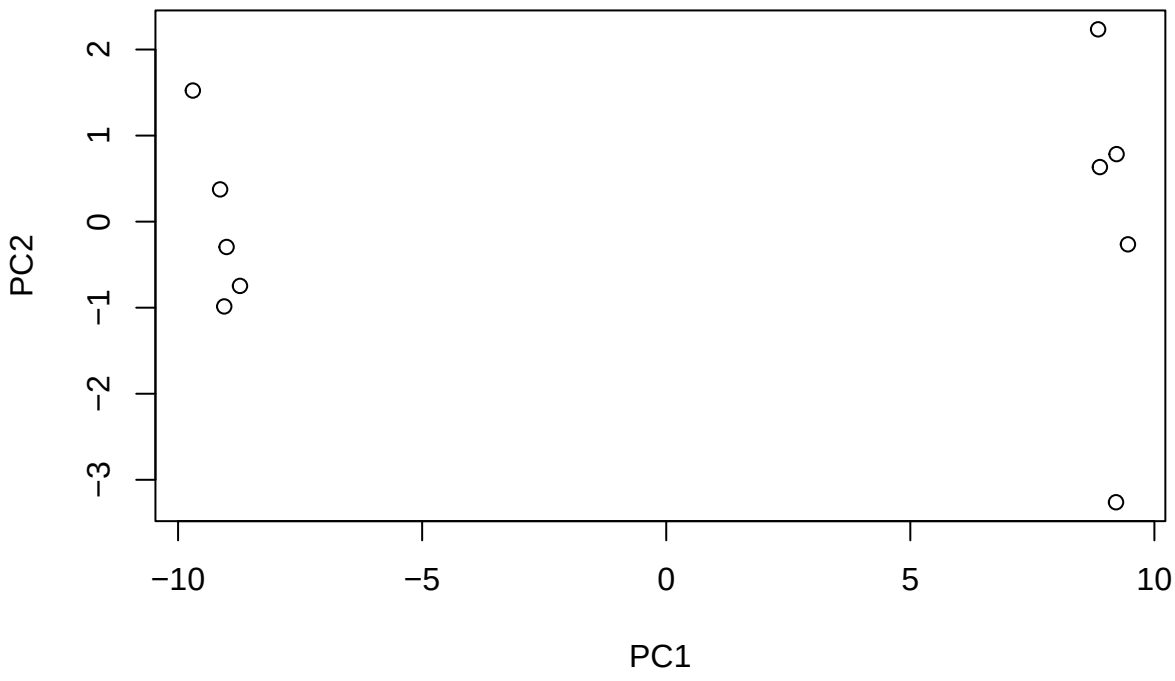
```
# Genes
ncol(x)
```

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

Answer: There are 17 samples and 4 genes in this data set.

Generating barplots etc. to make sense of this data is really not an exciting or worthwhile option to consider. So lets do PCA and plot the results.

```
## 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")
```



Examine a summary of how much variation in the original data each PC accounts for.

```
summary(pca)
```

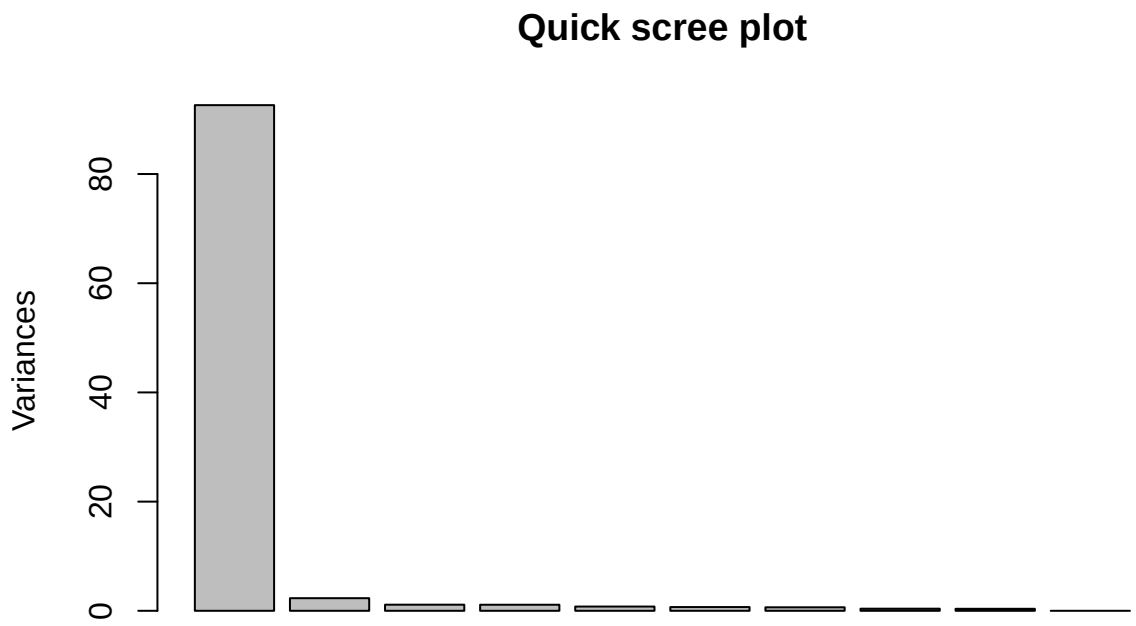
```
## Importance of components:
##                           PC1    PC2     PC3     PC4     PC5     PC6     PC7
## Standard deviation     9.6237 1.5198 1.05787 1.05203 0.88062 0.82545 0.80111
## Proportion of Variance 0.9262 0.0231 0.01119 0.01107 0.00775 0.00681 0.00642
```

```
## 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
```

Based on these results, PC1 is where all the action is (accounts for 92.6% of the variations)

A quick barplot summary of this Proportion of Variance for each PC can be obtained by calling the plot() function directly on our prcomp result object.

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



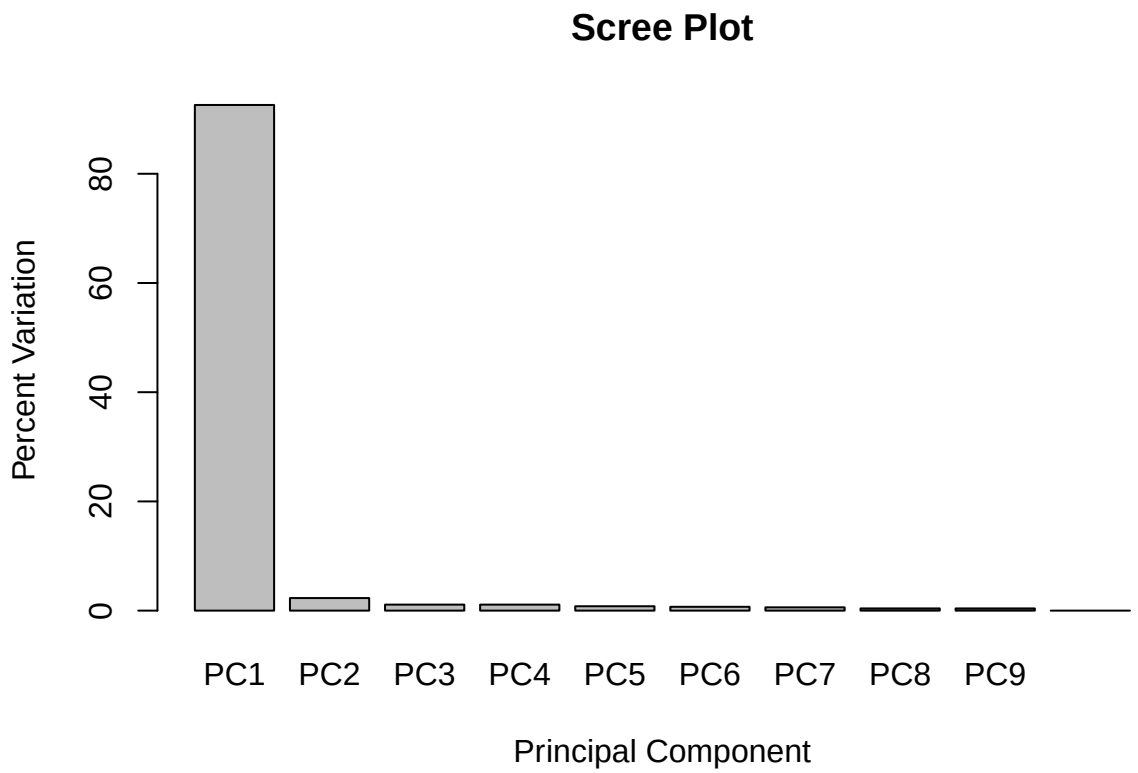
We can use the square of `pra$sdev` to calculate how much variation in the original data each PC accounts for.

```
## 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
```

Generate a screen plot.

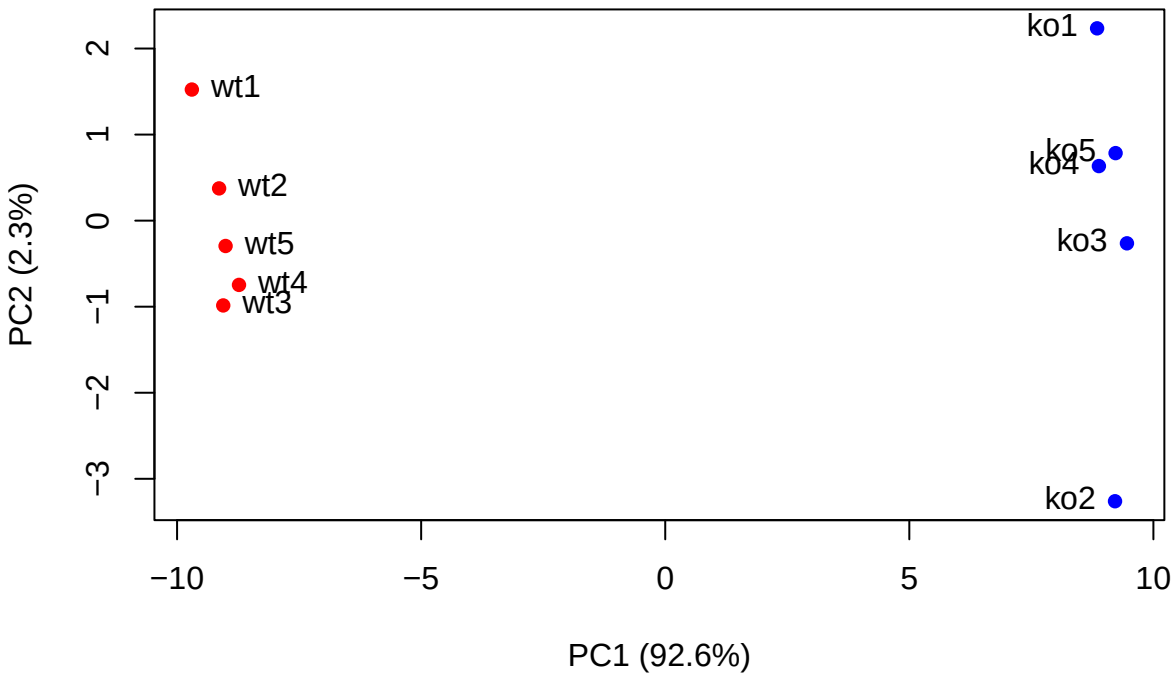
```
barplot(pca.var.per, main="Scree Plot",
        names.arg = paste0("PC", 1:10),
        xlab="Principal Component", ylab="Percent Variation")
```



Based on this plot, PC1 is where all the action is

Now lets make our main PCA plot a bit more attractive and useful.

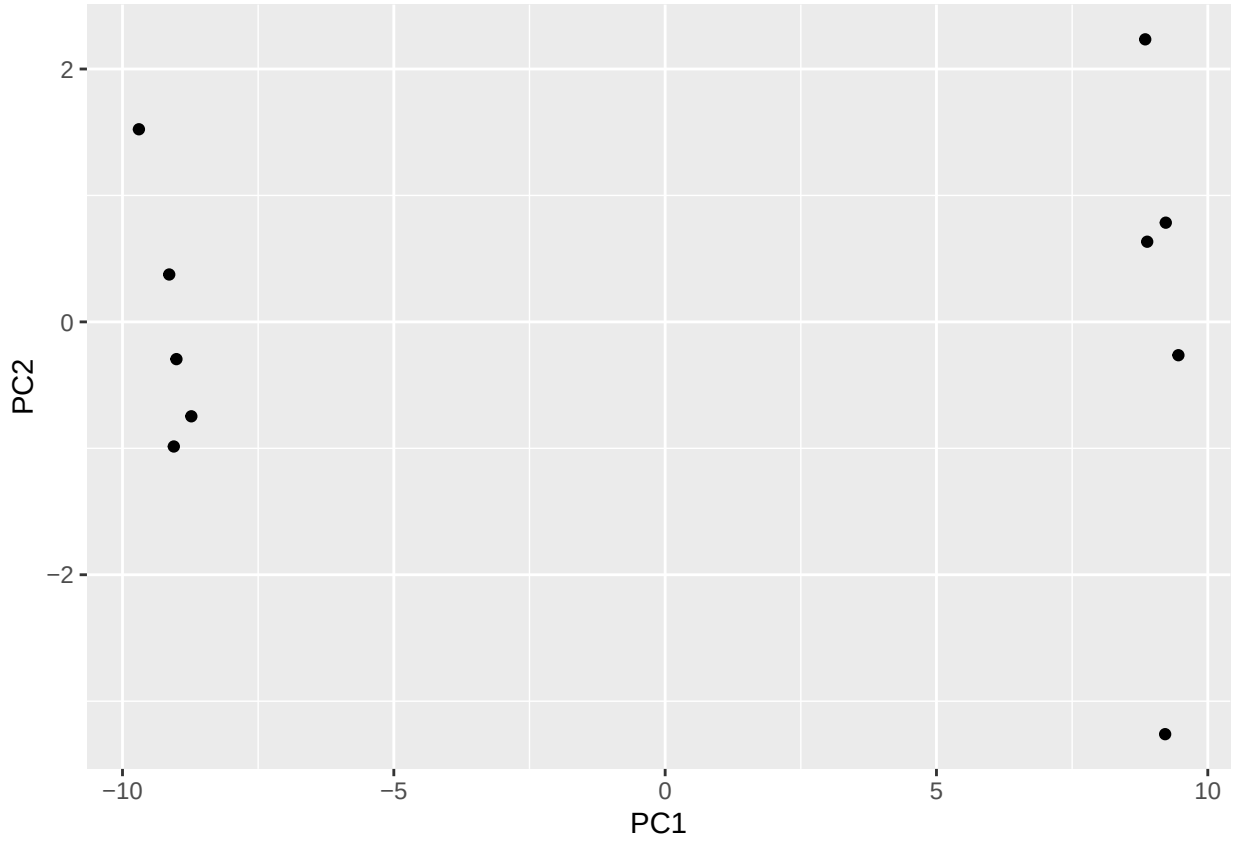
```
## A vector of colors for wt and ko samples
colvec <- colnames(rna.data)
colvec[grep("wt", colvec)] <- "red"
colvec[grep("ko", colvec)] <- "blue"

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

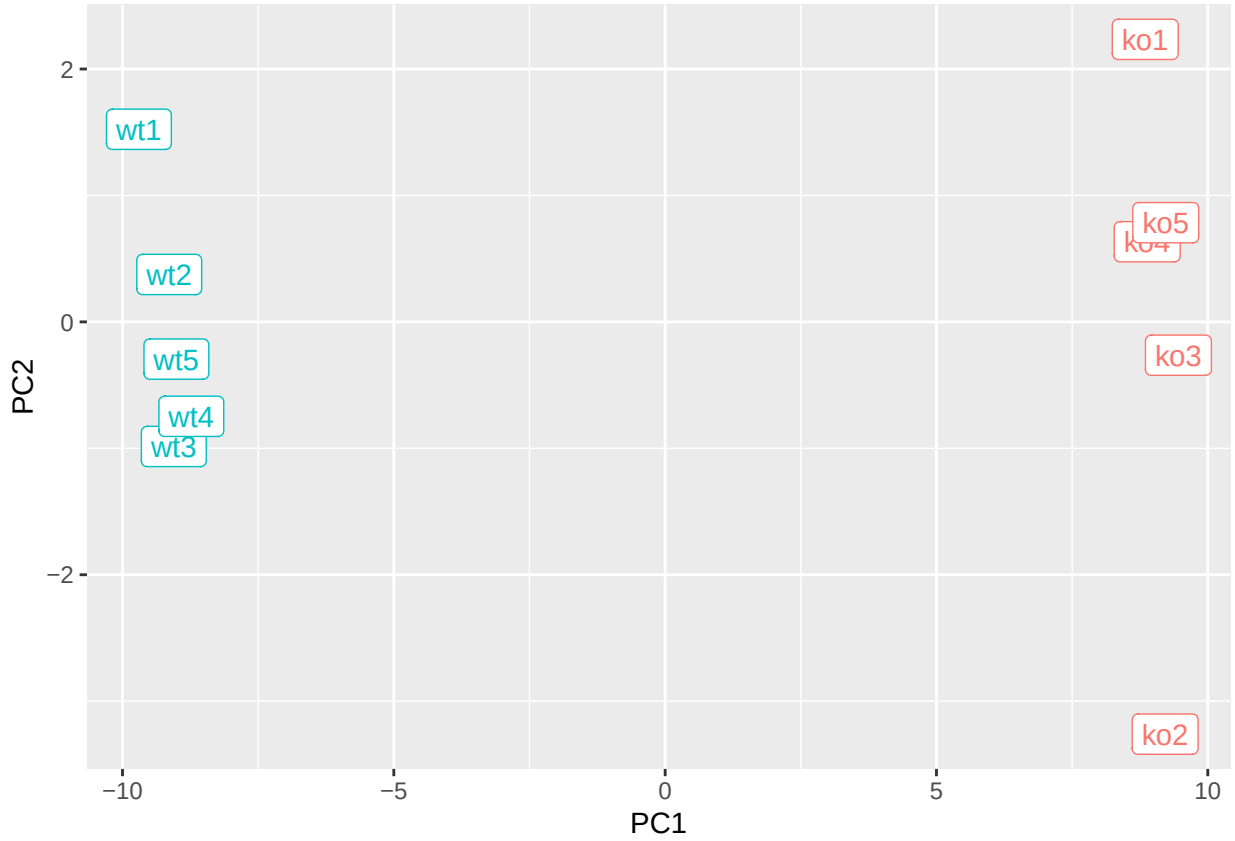
We need a data.frame as input for the main ggplot() function. This data.frame will need to contain our PCA results (specifically pca$x) and additional columns for any other aesthetic mappings we will want to display.

```
library(ggplot2)

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

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



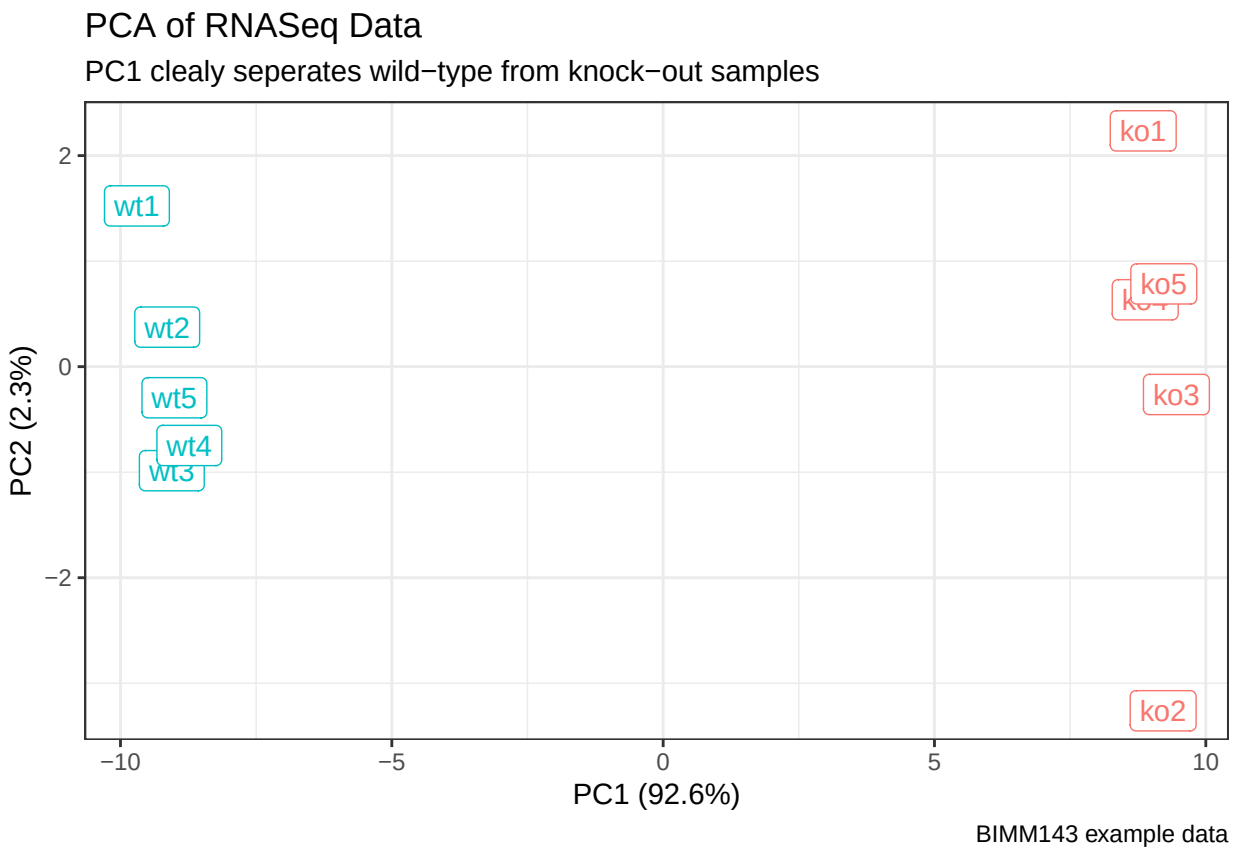
Add a condition-specific color and label aesthetics for wild-type and knock-out samples.

```
# 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
```



Finally add some spit and polish.

```
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

let's find 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"
```

These may be the genes that we would like to focus on for further analysis (if their expression changes are significant - we will deal with this and further steps of RNA-Seq analysis in subsequent classes).