SysBio Day 2 Hands On

Contents

Overview

In today's practice we will be focusing on data visualization and basic summary statistics.

This examples are intended to be as simple as possible (e.g. using base R), even though there are many packages tailored for plotting in R, for instance one of the most common is `ggplot2` which offers more functionalities but needs more time to explain and understand. Because of this, it can't be covered in the available time for this course. Nevertheless, once you understand the basics, it's easy to find tutorials or cheatsheets online that cover more advanced plotting techniques and/or functionalities.

You may find for instance useful cheatsheets for `ggplot2` and other common R packages in the RStudio website: https://www.rstudio.com/resources/cheatsheets/

1) Loading packages and data

Most of the methods explained during the class are part of `R-basics`. Thus their installation are not required.

Still, we will need to install the following packages

- `RColorBrewer`
- `Rtsne`

```
install.packages('RColorBrewer')
install.packages('Rtsne')
```

Loading necessary packages

```
library(Rtsne)
library(RColorBrewer)
```

Load the data

Load the expression and annotation data we saved yesterday:

```
data = read.csv('my_data.csv', row.names=1, stringsAsFactors=F, check.names=F)
meta = read.csv('my_p_data.csv', row.names=1, stringsAsFactors=F)
```

2) Subsetting the data

For today we will use only the treated samples (aka 10000nM):

```
sample_tags = as.character(meta$title)

## Drug concentration
# Extract the part describing drug concentration
drug_conc = sapply(sample_tags, function(tag){
  strsplit(tag, split="_")[[1]][3]
  })
# Replace nanoMolar with empty string
drug_conc = gsub("nM$", "", drug_conc)
# Convert to numbers
drug_conc = as.numeric(drug_conc)
# Max conc
drug_conc.max = max(drug_conc)

# Subsetting
selected_samples = drug_conc == drug_conc.max
data = data[, selected_samples]
meta = meta[selected_samples, ]

dim(data)
dim(meta)
```

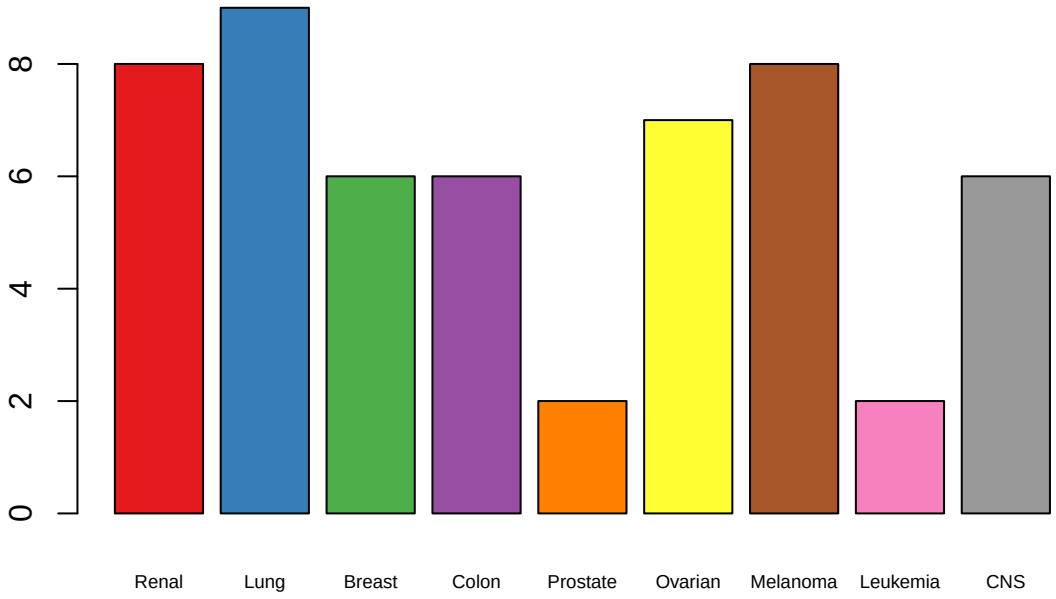
3) Preparing a color palette

```
# List of unique tissues in our data set
tissues = unique(meta$tissue.ch1)

# List of unique colors for each tissue
tissue_colors = brewer.pal(length(tissues), 'Set1')

# List of tissue-assigned colors for each sample
sample_colors = tissue_colors[as.integer(factor(meta$tissue.ch1))]
```

4) Generating barplots

Let's create a barplot to see how many samples we have per tissue. First we will count how many samples per tissue we have in our data set and generate a list of colors for each one.

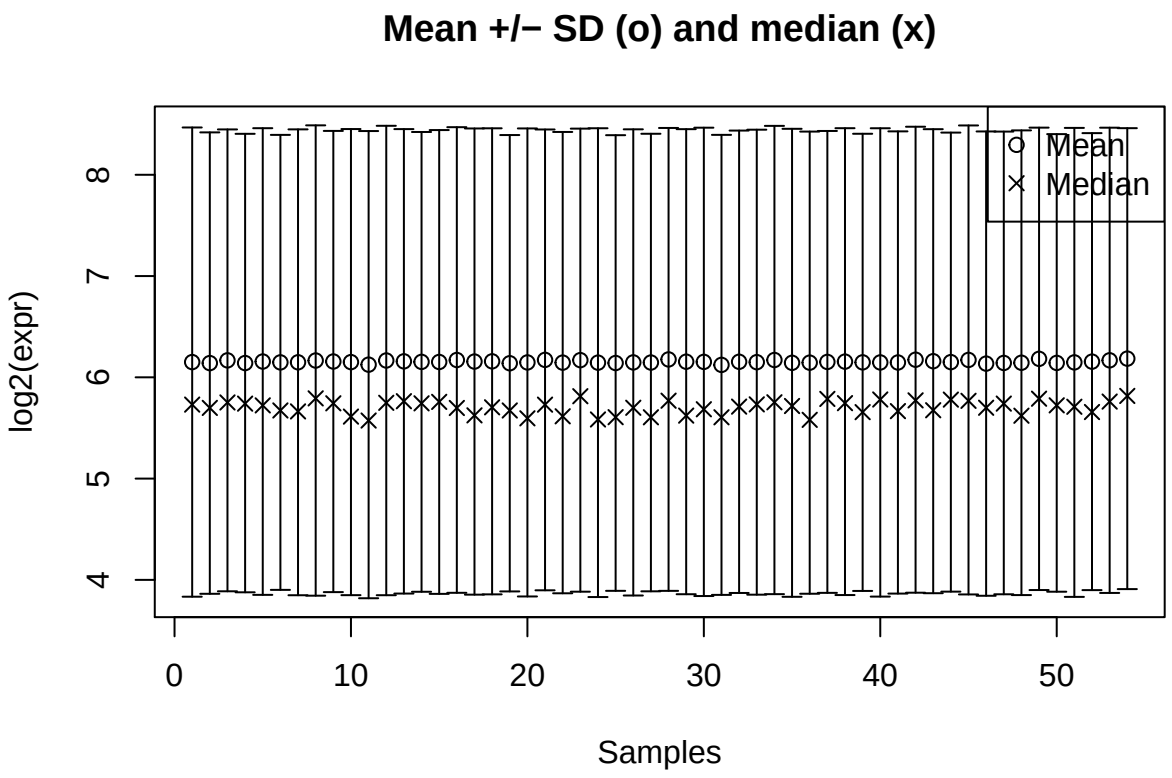
```
tissue_counts = sapply(tissues, function(x){sum(meta$tissue.ch1 == x)})
barplot(tissue_counts, cex.names=.6, col=tissue_colors)
```



5) Basic statistics

Let us first compute some summary statistics and plot them across samples in order to get a preliminary idea of our expression data.

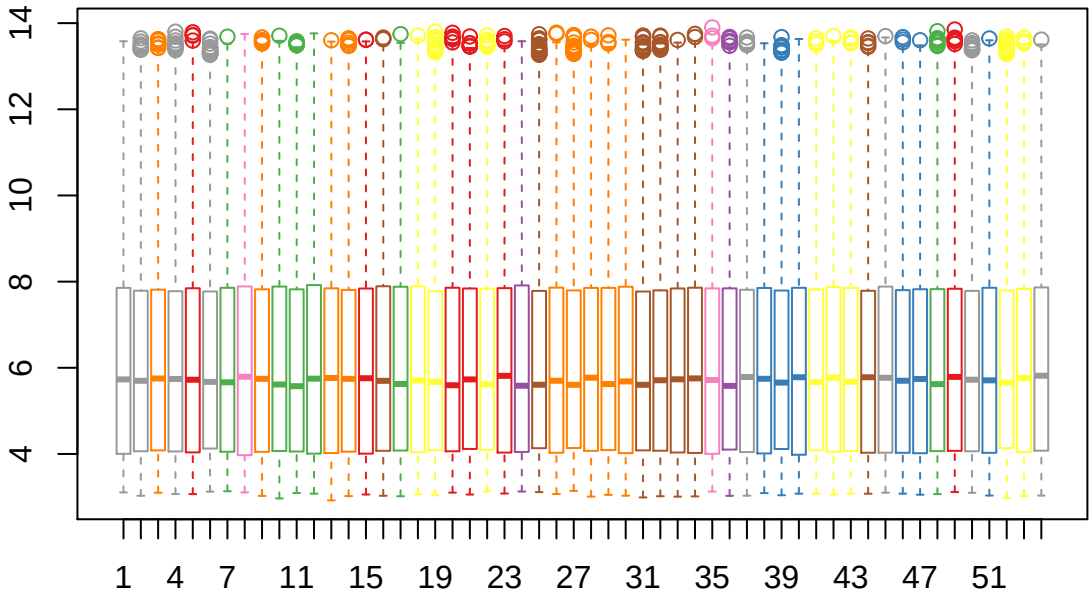
```
c_mean = colMeans(data)
c_median = apply(data, 2, median)
c_std = apply(data, 2, sd)

rng = 1:length(c_mean)

plot(rng, c_mean, ylim=range(c(c_mean - c_std, c_mean + c_std)),
     main='Mean +/- SD (o) and median (x)', xlab='Samples', ylab='log2(expr)')
points(rng, c_median, pch=4)

# Despite R being an **statistics-designed language**, it's surprising they
# don't provide a straightforward way to plot errorbars (w/o external packages)
arrows(rng, c_mean - c_std, rng, c_mean + c_std, length=0.05, angle=90, code=3)

# Adding legend
legend('topright', c('Mean', 'Median'), pch=c(1, 4))
```



6) Boxplots

Let's see now how this data looks like using a box plot, coloring by tissue:

```
boxplot(data, border=sample_colors,
        names=NULL)
```

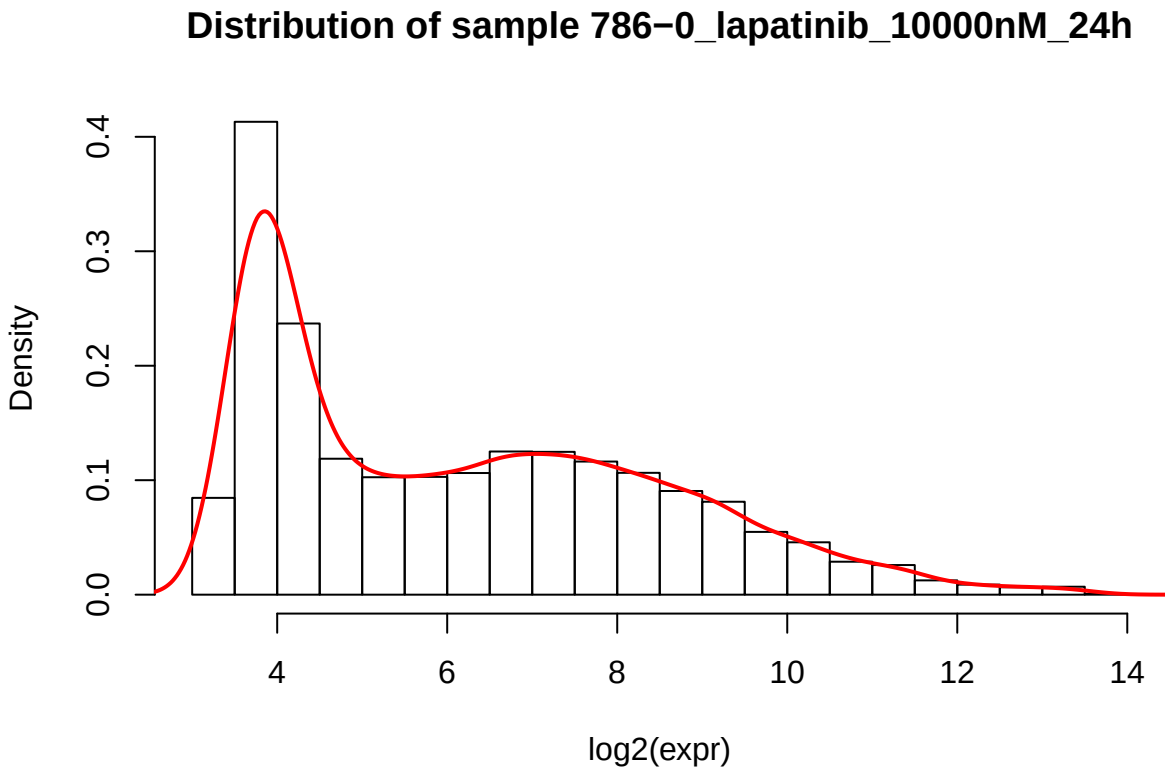



```
# Doesn't work and I hate R
#legend(0.03, 0.025, names(tissue_counts), col=tissue_colors)
```

7) Histogram + density

Now we select a sample and see how the expression data distribution looks like. To do this we can plot the histogram or the density plot (or both).

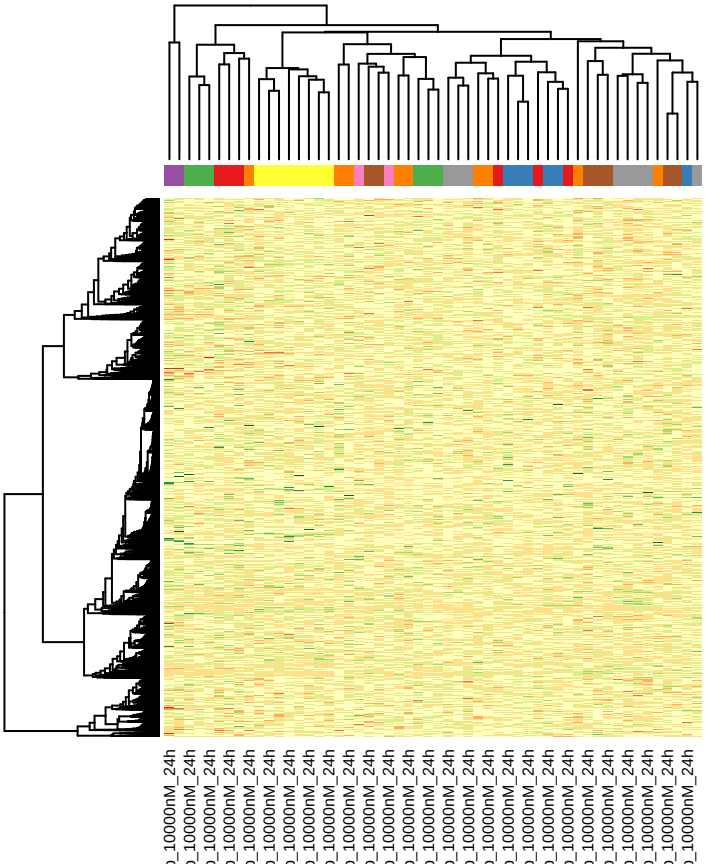
```
sample_no = 1
hist(data[, sample_no], breaks=25, prob=T, xlab='log2(expr)',
     main=paste0('Distribution of sample ', colnames(data)[sample_no]))
lines(density(data[, sample_no]), lw=2, col='red')
```



8) Heatmap + hierarchical clustering

Now we will plot the heatmap of our expression profiles along with the hierarchical clustering. On top of that, we will show the tissue of origin with colors (which can nicely be done in a single function in base R):

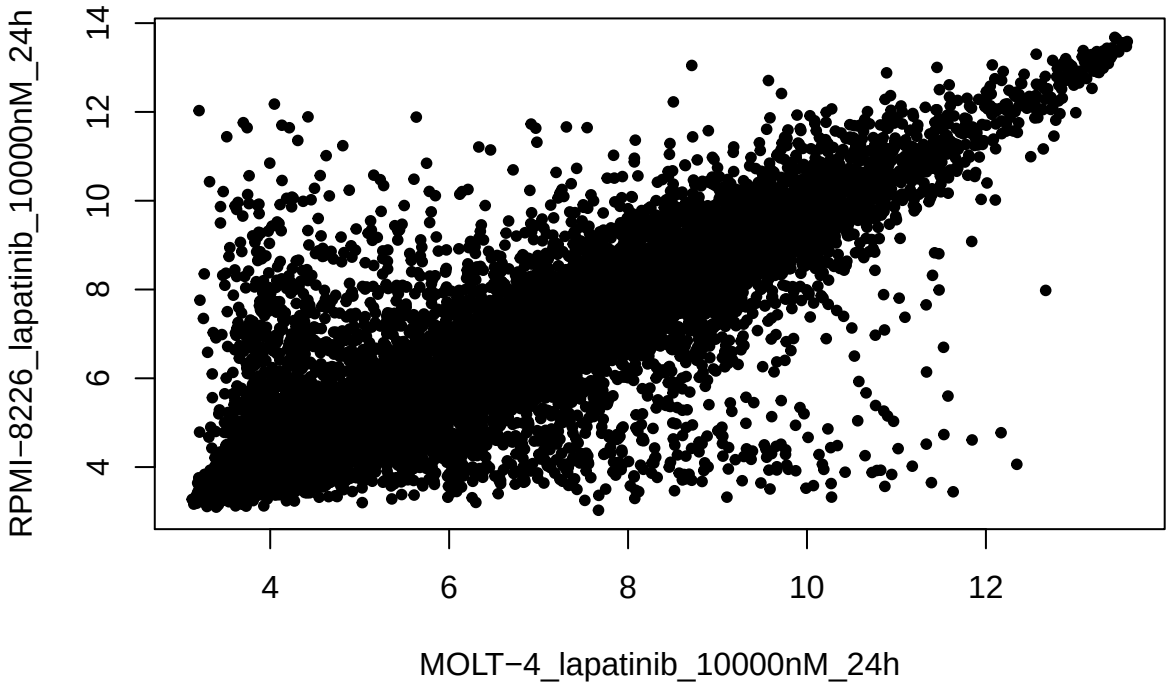
```
heatmap(as.matrix(data), labRow=F, col=brewer.pal(11, 'RdYlGn'),
        ColSideColors=sample_colors)
```



9) Scatter plot and correlation

Leukemia samples (pink in this example) seem to cluster together, let's see how well they correlate:

```
usecols = colnames(data)[meta$tissue.ch1 == 'Leukemia']

plot(data[, usecols[1]], data[, usecols[2]], pch=20, xlab=usecols[1],
     ylab=usecols[2])
```




```
cor(data[, usecols])
```

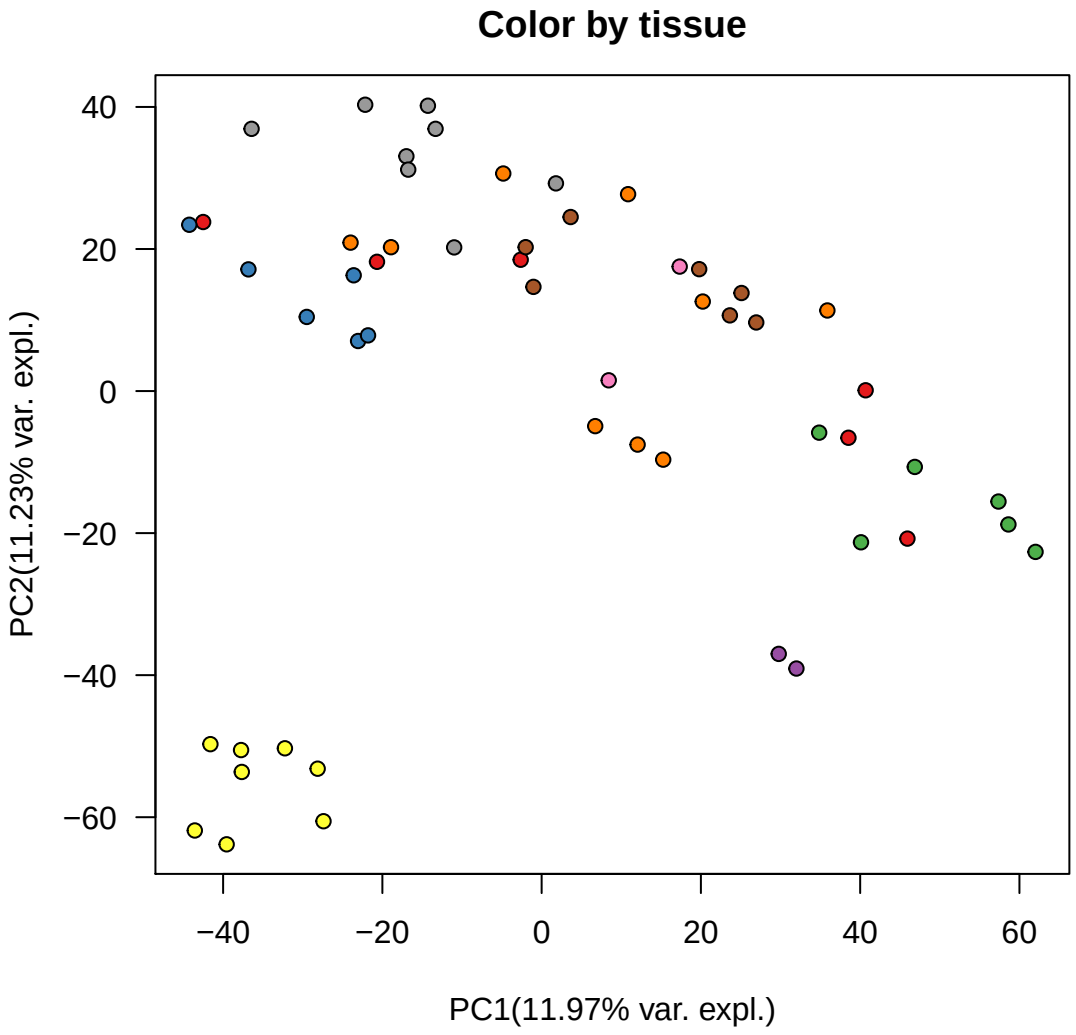
10) PCA on gene expression profiles of treated cell lines

Next, a Principal Component Analysis (PCA) is performed on the data to assess the main sources of variability across the gene expression profiles. In order to do this, we first define a simple function that shows the most relevant information from the PCA (i.e. the first 2 principal components and how much variance each one explains).

```
plotPCA <- function(pca, pchs=21, cols="white", PCs=c("PC1", "PC2")){
  # stopifnot(pca)
  # stopifnot(length(PCs) == 2)
  tmp <- summary(pca)
    # importance: variance explained
    PCx <- format(round(tmp$importance[2, PCs[1]] * 100, digits=2), nsmall=2)
    PCy <- format(round(tmp$importance[2, PCs[2]] * 100, digits=2), nsmall=2)
  x <- data.frame(pca$x)

  # Plot
  plot(x[, PCs[1]], x[, PCs[2]], las=1,
       pch=pchs, bg=cols,
       xlab=paste0(PCs[1], "(", PCx, "% var. expl.)"),
       ylab=paste0(PCs[2], "(", PCy, "% var. expl.)"))
}

# Perform PCA
# Note: the matrix must be transposed, so samples are rows
pca.n <- prcomp(t(data))
#pca.s <- prcomp(t(data.scaled))
#pca.q <- prcomp(t(data.qnorm))
```

Now let's plot it!

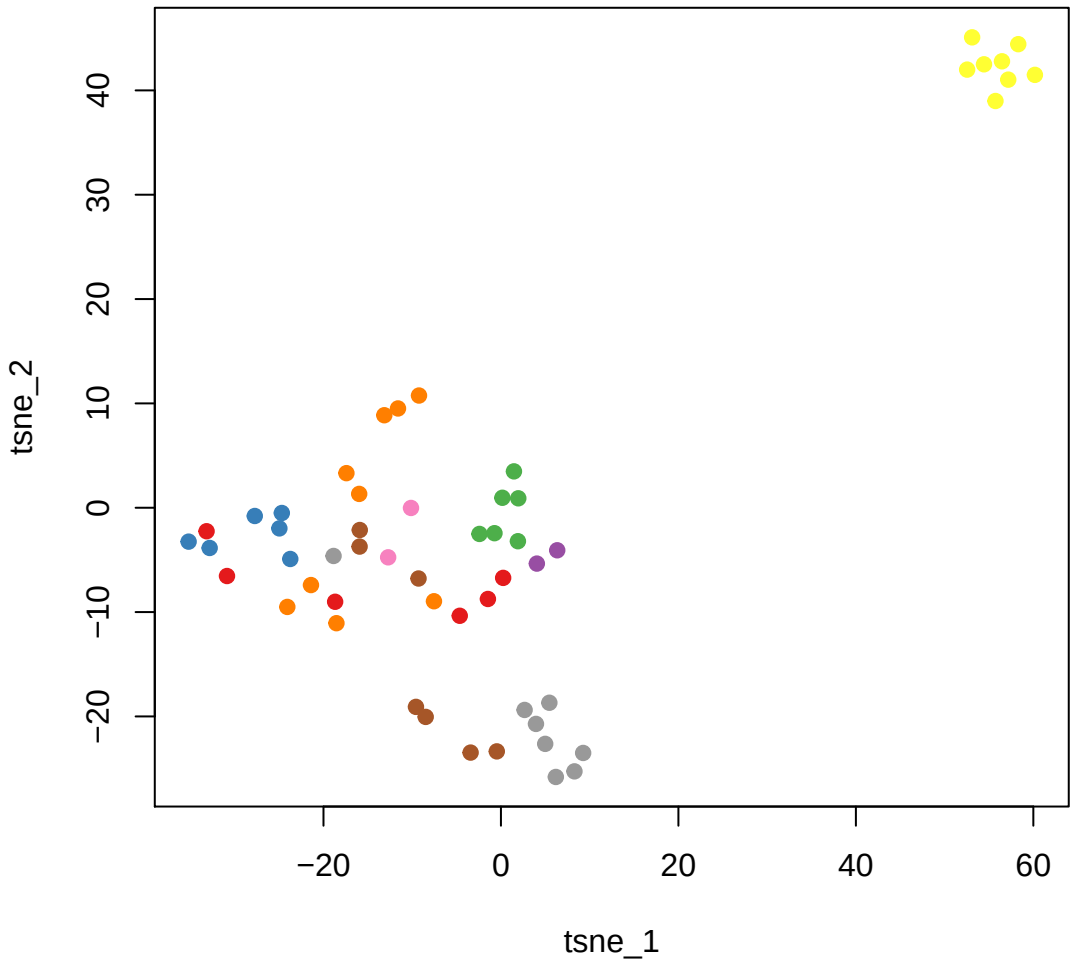
```
plotPCA(pca.n, cols=sample_colors, PCs=c("PC1", "PC2"))
title("Color by tissue", line=1)
```



11) t-SNE

t-distributed Stochastic Neighbour Embedding is another dimensionality reduction algorithm (originally developed by Laurens Van der Maaten) that has gained a lot of importance in the last years. One of its major improvements from the common PCA is that is able to reveal some non-linear relations across (multi-dimensional) data points. As a drawback, the parameter choice is non-trivial and the obtained results may vary depending on the pseudo-random number generator seed (stochastic implies randomness by definition).

```
# perplexity should be 3 * perplexity < nrow(X) - 1 (default=50)
# theta is speed/accuracy ratio. 1 is max speed, 0 max accuracy (default=0.5)
tsne_data = Rtsne(t(data), perplexity=5, theta=0.5) # NOTE: Transposed matrix!
plot(tsne_data$Y[, 1], tsne_data$Y[, 2], col=sample_colors, pch=19,
     xlab='tsne_1', ylab='tsne_2')
```


tsne_2
40
30
20
10
0
−10
−20
−20
0
20
40
60
tsne_1