Read counts to DGE, Part I

Friederike Dündar, ABC

This script will show you how to:

- Read in `featureCounts` results into R.
- Use `DESeq2` not (!) for DGE (yet), but to:
  - normalize read counts for differences in sequencing depth
  - transform reads to the log2 scale including variance reduction
- Accompany each step by exploratory plots.

You can generate an html document out of this entire script by clicking the `Knit HTML` button in RStudio.

```
options(stringsAsFactors = FALSE) # this will change a global setting, but just for this session

library(knitr)
opts_chunk$set(echo = TRUE, message = FALSE,cache=FALSE) # tuning knitr output
```

featureCounts

We aligned five samples for the WT and SNF2 condition, respectively. You can find those files here: `~/mat/precomputed/results_alignment`.

How can you check which command was used to generate those `BAM` files?

```
## on the command line (!)
mkdir class/read_counts
cd class/read_counts
REF_DIR=~/class2019/mat/referenceGenomes/S_cerevisiae/

 # reads for yeast samples counted on the meta-feature level
~/class2019/mat/software/subread-1.6.0-Linux-x86_64/bin/featureCounts \
    -a ${REF_DIR}/Saccharomyces_cerevisiae.R64-1-1.81.gtf \
    -o featCounts_genes.txt \
     ~/mat/precomputed/results_alignment/*bam
```

Let's read the result file into R, i.e. download the table from our website or use the `scp` command to download the table that you generated on the server to your local machine.

Loading additional libraries:

```
library(ggplot2) # for making plots
```

```
## Warning: package 'ggplot2' was built under R version 3.5.2
```

```
library(magrittr) # for "pipe"-like coding in R
```

Reading the featureCounts result table into R:

```
# reading in featureCounts output
readcounts <- read.table("~/Downloads/featCounts_genes.txt",
                         header=TRUE)
head(readcounts)
```

```
##      Geneid Chr Start   End Strand Length
## 1   YDL248W  IV  1802  2953      +   1152
```

```
## 2 YDL247W-A  IV  3762  3836      +     75
## 3   YDL247W  IV  5985  7814      +   1830
## 4   YDL246C  IV  8683  9756      -   1074
## 5   YDL245C  IV 11657 13360      -   1704
## 6   YDL244W  IV 16204 17226      +   1023
##   X.home.classadmin.mat.precomputed.results_alignment.SNF2_1_Aligned.sortedByCoord.out.bam
## 1                                                                                      109
## 2                                                                                        0
## 3                                                                                        6
## 4                                                                                        6
## 5                                                                                        1
## 6                                                                                       79
##   X.home.classadmin.mat.precomputed.results_alignment.SNF2_2_Aligned.sortedByCoord.out.bam
## 1                                                                                       84
## 2                                                                                        1
## 3                                                                                        6
## 4                                                                                        6
## 5                                                                                        6
## 6                                                                                       59
##   X.home.classadmin.mat.precomputed.results_alignment.SNF2_3_Aligned.sortedByCoord.out.bam
## 1                                                                                      100
## 2                                                                                        1
## 3                                                                                        1
## 4                                                                                        1
## 5                                                                                        9
## 6                                                                                       49
##   X.home.classadmin.mat.precomputed.results_alignment.SNF2_4_Aligned.sortedByCoord.out.bam
## 1                                                                                      112
## 2                                                                                        0
## 3                                                                                        3
## 4                                                                                        4
## 5                                                                                        5
## 6                                                                                       60
##   X.home.classadmin.mat.precomputed.results_alignment.SNF2_5_Aligned.sortedByCoord.out.bam
## 1                                                                                       62
## 2                                                                                        3
## 3                                                                                        4
## 4                                                                                        4
## 5                                                                                        3
## 6                                                                                       37
##   X.home.classadmin.mat.precomputed.results_alignment.WT_1_Aligned.sortedByCoord.out.bam
## 1                                                                                    47
## 2                                                                                     0
## 3                                                                                     2
## 4                                                                                     1
## 5                                                                                     6
## 6                                                                                     9
##   X.home.classadmin.mat.precomputed.results_alignment.WT_2_Aligned.sortedByCoord.out.bam
## 1                                                                                    65
## 2                                                                                     0
## 3                                                                                     3
## 4                                                                                     3
## 5                                                                                     2
## 6                                                                                     8
```

```
##   X.home.classadmin.mat.precomputed.results_alignment.WT_3_Aligned.sortedByCoord.out.bam
## 1                                                                                      60
## 2                                                                                       1
## 3                                                                                       4
## 4                                                                                       2
## 5                                                                                       5
## 6                                                                                      12
##   X.home.classadmin.mat.precomputed.results_alignment.WT_4_Aligned.sortedByCoord.out.bam
## 1                                                                                      95
## 2                                                                                       0
## 3                                                                                       7
## 4                                                                                       4
## 5                                                                                       5
## 6                                                                                      30
##   X.home.classadmin.mat.precomputed.results_alignment.WT_5_Aligned.sortedByCoord.out.bam
## 1                                                                                      43
## 2                                                                                       0
## 3                                                                                       9
## 4                                                                                       0
## 5                                                                                       6
## 6                                                                                      14
```

We will use the `DESeq2` package to normalize the samples for differences in their sequencing depths. To make full use of `DESeq2`'s functions, we will have to turn the `data.frame` that we just generated via `read.table` into a specific R object that was defined by the authors of `DESeq2`.

```
# not available via install.packages(), but through bioconductor
# installation should not be done everytime the Rmd file is compiled,
# hence the eval=FALSE code chunk setting to turn off its evaluation
BiocManager::install("DESeq2")
```

```
library(DESeq2)
```

We will have to generate a `DESeqDataSet`; what is needed for this can be found out via `?DESeqDataSetFromMatrix`. The help indicates that we need two tables: `countData` and `colData`.

- `colData`: `data.frame` with all the variables you know about your samples, e.g., experimental condition, the type, and date of sequencing and so on. Its `row.names` should correspond to the unique sample names.
- `countData`: should contain a matrix of the actual values associated with the genes and samples. Is equivalent to `assay()`. Conveniently, this is almost exactly the format of the `featureCounts` output.

Preparing the count matrix for the DESeq2DataSet class:

```
# give meaningful and legible sample names
## the original names of the sample columns are something like:
## 'X.home.classadmin.mat.precomputed.results_alignment.WT_5_Aligned.sortedByCoord.out.bam'
orig_names <- names(readcounts)
## here, we use two regular expressions with the gsub function
## 1. ".*alignment\\." --> any character (.*) occurring before an instance of
## 'alignment' followed by a dot (alignment\\.) should be replaced with nothing ("")
## 2. then, we also replace every instance of '_Aligned' and all subsequent
## characters (.*) with nothing ("")
names(readcounts) <- gsub(".*alignment\\.", "" ,orig_names) %>% gsub("_Aligned.*", "", .)
```

```
# gene IDs should be stored as row.names
row.names(readcounts) <- gsub("-", ".", readcounts$Geneid)
```

```
# exclude the columns without read counts (columns 1 to 6 contain additional
# info such as genomic coordinates)
readcounts <- readcounts[,-c(1:6)]
```

Always check your data set after you manipulated it!

```
str(readcounts)
```

```
## 'data.frame':    7126 obs. of  10 variables:
##  $ SNF2_1: int  109 0 6 6 1 79 363 41 143 2 ...
##  $ SNF2_2: int  84 1 6 6 6 59 289 22 119 4 ...
##  $ SNF2_3: int  100 1 1 1 9 49 243 26 115 1 ...
##  $ SNF2_4: int  112 0 3 4 5 60 352 26 135 5 ...
##  $ SNF2_5: int  62 3 4 4 3 37 241 22 96 3 ...
##  $ WT_1  : int  47 0 2 1 6 9 192 25 239 2 ...
##  $ WT_2  : int  65 0 3 3 2 8 169 30 343 0 ...
##  $ WT_3  : int  60 1 4 2 5 12 190 18 251 1 ...
##  $ WT_4  : int  95 0 7 4 5 30 309 42 555 3 ...
##  $ WT_5  : int  43 0 9 0 6 14 171 27 323 1 ...
```

```
head(readcounts)
```

```
##           SNF2_1 SNF2_2 SNF2_3 SNF2_4 SNF2_5 WT_1 WT_2 WT_3 WT_4 WT_5
## YDL248W      109     84    100    112     62   47   65   60   95   43
## YDL247W.A      0      1      1      0      3    0    0    1    0    0
## YDL247W        6      6      1      3      4    2    3    4    7    9
## YDL246C        6      6      1      4      4    1    3    2    4    0
## YDL245C        1      6      9      5      3    6    2    5    5    6
## YDL244W       79     59     49     60     37    9    8   12   30   14
```

In addition to the read counts, we need to add some more information about the samples, e.g. which condition and replicate they represent. According to `?colData`, that type of "metadata" should be supplied in a `data.frame`, where the *rows* directly match the *columns* of the count data.

Here's how this could be generated in `R` matching the `readcounts` `data.frame` we already have:

```
sample_info <- DataFrame(condition = gsub("_[0-9]+", "", names(readcounts)),
                         row.names = names(readcounts) )
sample_info
```

```
## DataFrame with 10 rows and 1 column
##          condition
##        <character>
## SNF2_1        SNF2
## SNF2_2        SNF2
## SNF2_3        SNF2
## SNF2_4        SNF2
## SNF2_5        SNF2
## WT_1            WT
## WT_2            WT
## WT_3            WT
## WT_4            WT
## WT_5            WT
```

```
str(sample_info)
```

```
## Formal class 'DataFrame' [package "S4Vectors"] with 6 slots
##   ..@ rownames       : chr [1:10] "SNF2_1" "SNF2_2" "SNF2_3" "SNF2_4" ...
```

```
##   ..@ nrows          : int 10
##   ..@ listData       :List of 1
##   .. ..$ condition: chr [1:10] "SNF2" "SNF2" "SNF2" "SNF2" ...
##   ..@ elementType    : chr "ANY"
##   ..@ elementMetadata: NULL
##   ..@ metadata       : list()
```

Let's generate the DESeqDataSet:

```
DESeq.ds <- DESeqDataSetFromMatrix(countData = readcounts,
                                   colData = sample_info,
                                   design = ~ condition)
DESeq.ds
```

```
## class: DESeqDataSet
## dim: 7126 10
## metadata(1): version
## assays(1): counts
## rownames(7126): YDL248W YDL247W.A ... RPM1 Q0297
## rowData names(0):
## colnames(10): SNF2_1 SNF2_2 ... WT_4 WT_5
## colData names(1): condition
```

```
head(counts(DESeq.ds))
```

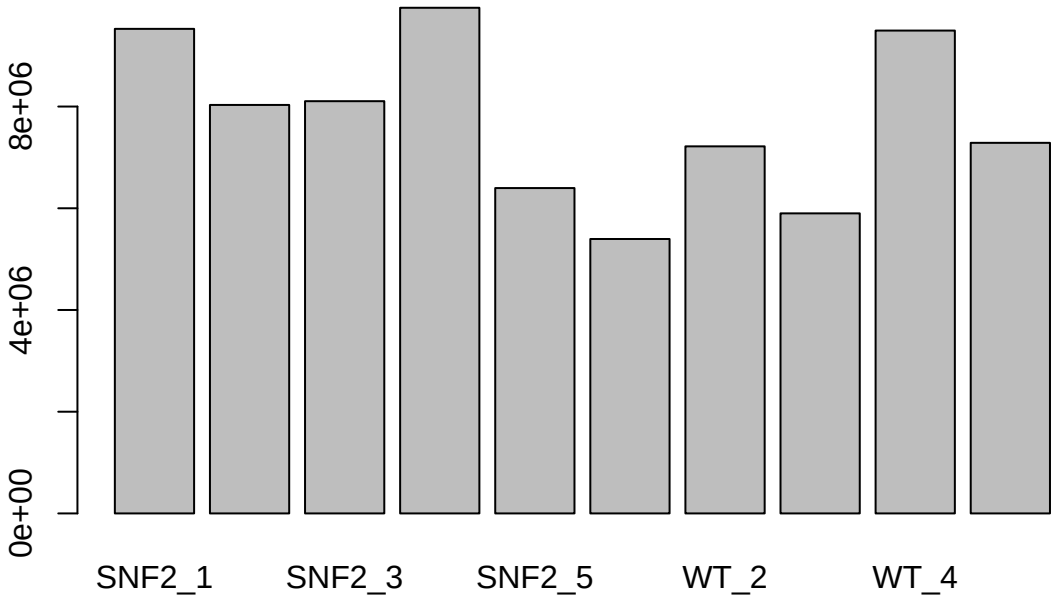
```
##           SNF2_1 SNF2_2 SNF2_3 SNF2_4 SNF2_5 WT_1 WT_2 WT_3 WT_4 WT_5
## YDL248W      109     84    100    112     62   47   65   60   95   43
## YDL247W.A      0      1      1      0      3    0    0    1    0    0
## YDL247W        6      6      1      3      4    2    3    4    7    9
## YDL246C        6      6      1      4      4    1    3    2    4    0
## YDL245C        1      6      9      5      3    6    2    5    5    6
## YDL244W       79     59     49     60     37    9    8   12   30   14
```

How many reads were counted for each sample (= library sizes)?

```
colSums(counts(DESeq.ds))
```

```
##  SNF2_1  SNF2_2  SNF2_3  SNF2_4  SNF2_5    WT_1    WT_2    WT_3    WT_4
## 9527395 8031694 8105366 9942250 6397208 5395963 7217177 5899432 9493974
##    WT_5
## 7286638
```

```
colSums(counts(DESeq.ds)) %>% barplot
```



Remove genes with no reads.

```
keep_genes <- rowSums(counts(DESeq.ds)) > 0
dim(DESeq.ds)
```

```
## [1] 7126   10
```

```
DESeq.ds <- DESeq.ds[ keep_genes, ]
dim(DESeq.ds)
```

```
## [1] 6680   10
```

```
counts(DESeq.ds) %>% str
```

```
##  int [1:6680, 1:10] 109 0 6 6 1 79 363 41 143 2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:6680] "YDL248W" "YDL247W.A" "YDL247W" "YDL246C" ...
##   ..$ : chr [1:10] "SNF2_1" "SNF2_2" "SNF2_3" "SNF2_4" ...
```

```
assay(DESeq.ds) %>% str
```

```
##  int [1:6680, 1:10] 109 0 6 6 1 79 363 41 143 2 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:6680] "YDL248W" "YDL247W.A" "YDL247W" "YDL246C" ...
##   ..$ : chr [1:10] "SNF2_1" "SNF2_2" "SNF2_3" "SNF2_4" ...
```

Now that we have the data in the right format, we can start using `DESeq2`'s functions, e.g. `estimateSizeFactors()` for calculating a factor that will be used to correct for sequencing depth differences.

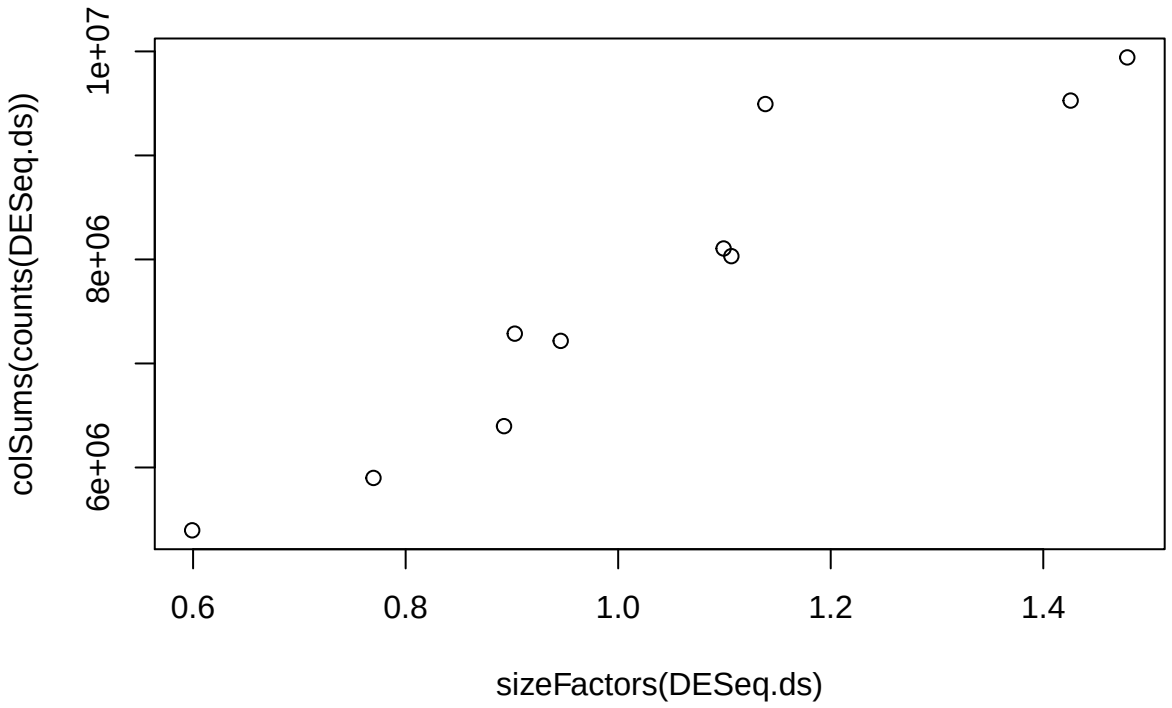
```
DESeq.ds <- estimateSizeFactors(DESeq.ds)
sizeFactors(DESeq.ds)
```

```
##    SNF2_1    SNF2_2    SNF2_3    SNF2_4    SNF2_5      WT_1      WT_2
## 1.4257668 1.1065006 1.0991960 1.4790643 0.8925667 0.5991287 0.9459106
##      WT_3      WT_4      WT_5
## 0.7697230 1.1385141 0.9026871
```

Check section 5.1.2 of the course notes to see the code for calculating the size factors yourself with base R

functions.

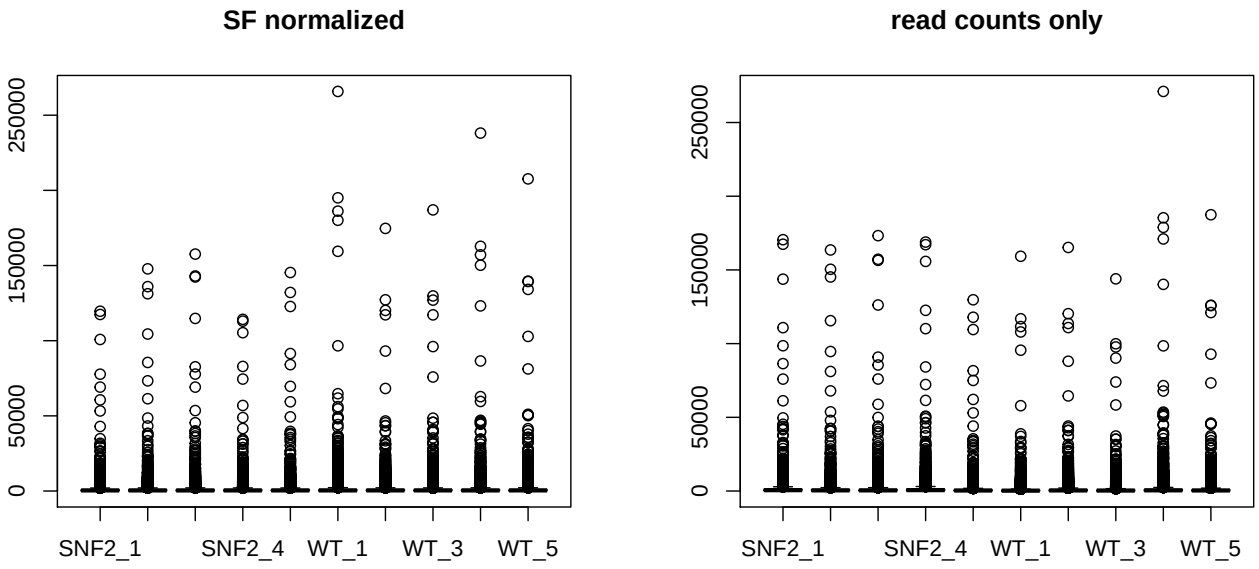
```
plot(sizeFactors(DESeq.ds), colSums(counts(DESeq.ds)))
```



The read counts normalized for sequencing depth can be accessed via `counts(DESeq.ds, normalized = TRUE)`.

Let's check whether the normalization helped to adjust global differences between the samples.

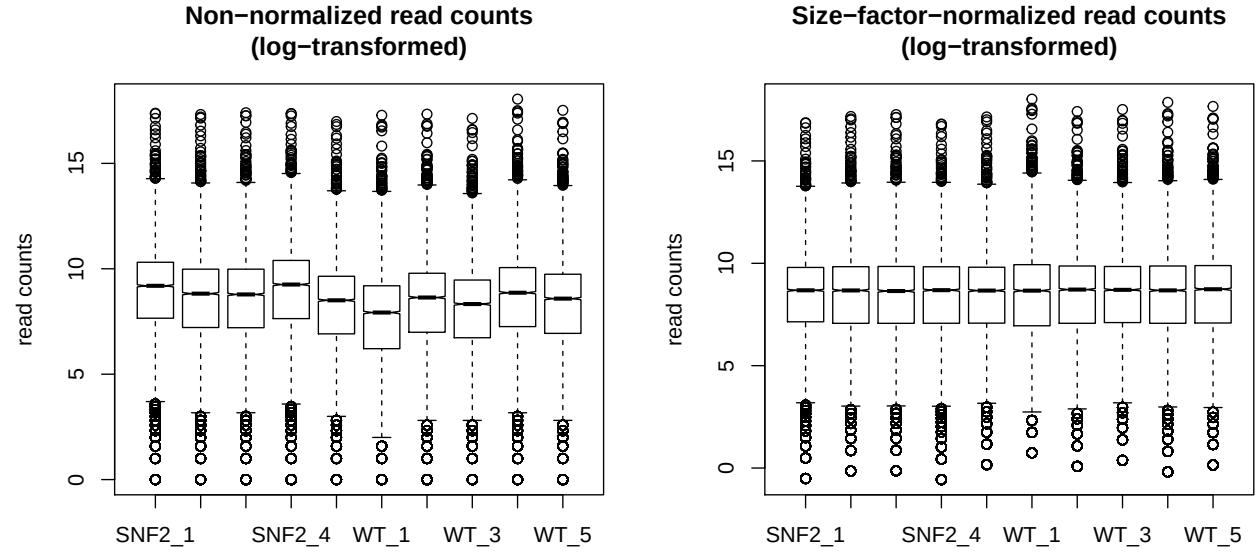
```
# setting up the plotting layout
par(mfrow=c(1,2))
counts.sf_normalized <- counts(DESeq.ds, normalized=TRUE)

# adding the boxplots
boxplot(counts.sf_normalized, main = "SF normalized")
boxplot(counts(DESeq.ds), main = "read counts only")
```



We can't really see anything. It is usually helpful to *transform* the normalized read counts to bring them onto more similar scales.

> To see the influence of the sequencing depth normalization, make two box plots of log2(read counts) - one for unnormalized counts, the other one for normalized counts (exclude genes with zero reads in all samples).

```
par(mfrow=c(1,2)) # to plot the two box plots next to each other
boxplot(log2(counts(DESeq.ds)), notch=TRUE,
        main = "Non-normalized read counts\n(log-transformed)",
        ylab="read counts")
boxplot(log2(counts(DESeq.ds, normalize= TRUE)), notch=TRUE,
        main = "Size-factor-normalized read counts\n(log-transformed)",
        ylab="read counts")
```



Understanding more properties of read count data

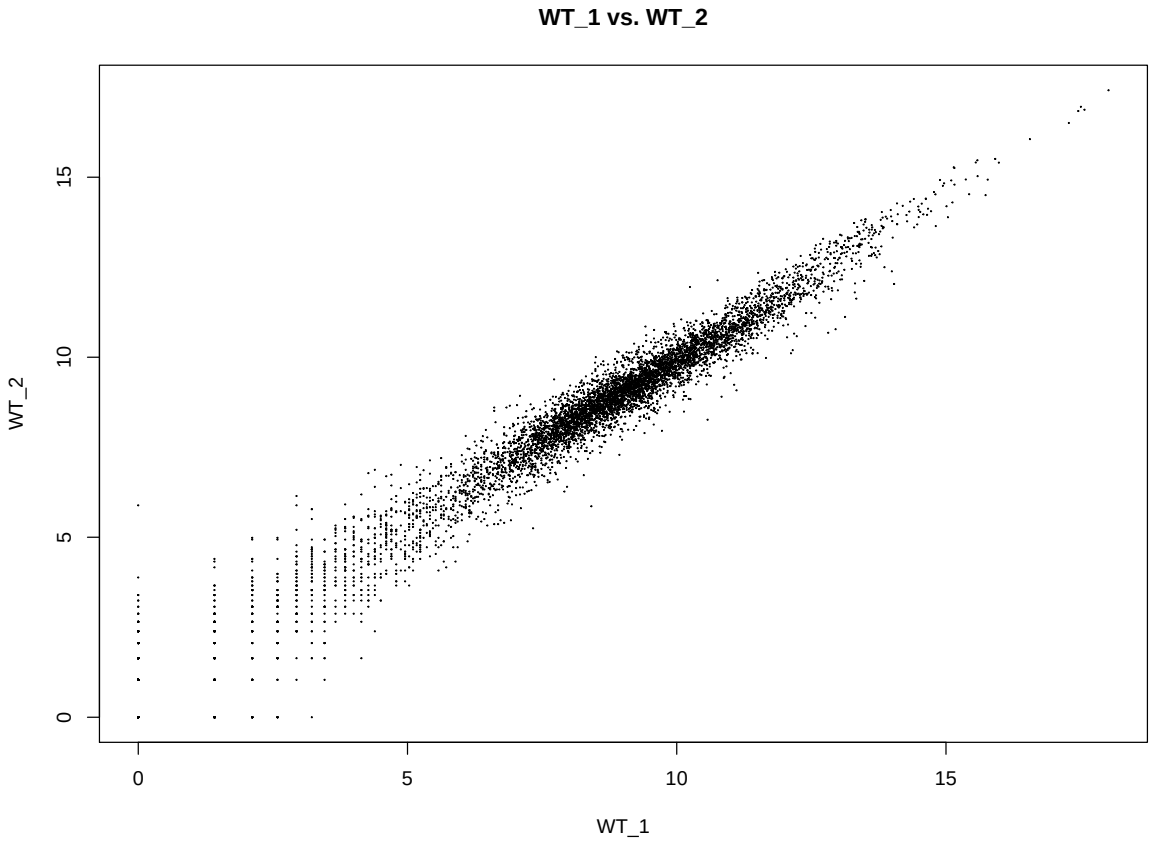
Characteristics we've touched upon so far:

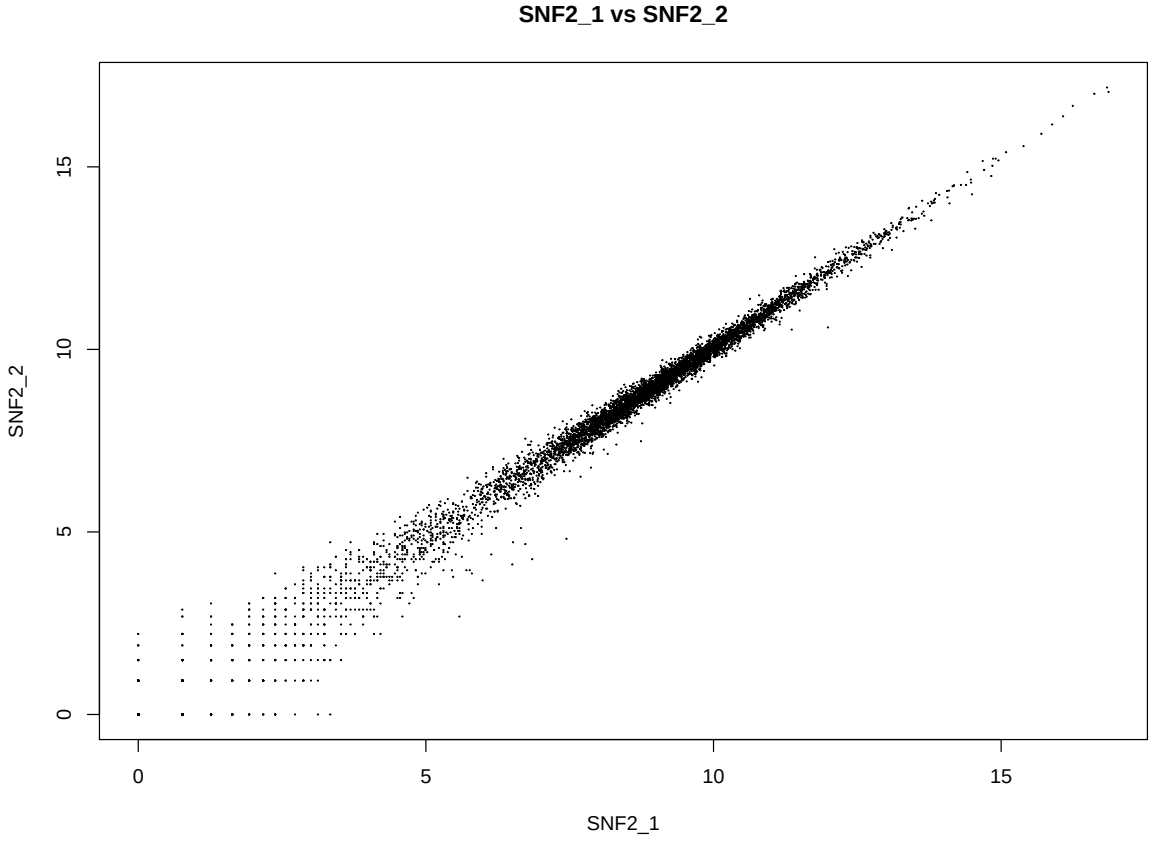
- zeros can mean two things: no expression or no detection
- fairly large dynamic range

  Make a scatterplot of log normalized counts against each other to see how well the actual values correlate which each other per sample and gene.

```
# non-normalized read counts plus pseudocount
log.counts <- log2(counts(DESeq.ds, normalized = FALSE) + 1)
# instead of creating a new object, we could assign the values to a distinct matrix
# normalized and log2-transformed read counts
assay(DESeq.ds, "log.norm.counts") <- log2(counts(DESeq.ds, normalized=TRUE) + 1)
```

```
par(mfrow=c(2,1))
DESeq.ds[, c("WT_1","WT_2")] %>% assay(.,  "log.norm.counts") %>%
    plot(., cex=.1, main = "WT_1 vs. WT_2")
DESeq.ds[, c("SNF2_1","SNF2_2")] %>% assay(.,  "log.norm.counts") %>%
    plot(., cex=.1, main = "SNF2_1 vs SNF2_2")
```


WT_1 vs. WT_2
WT_2
WT_1
0
5
10
15


SNF2_1 vs SNF2_2
SNF2_2
SNF2_1
0
5
10
15

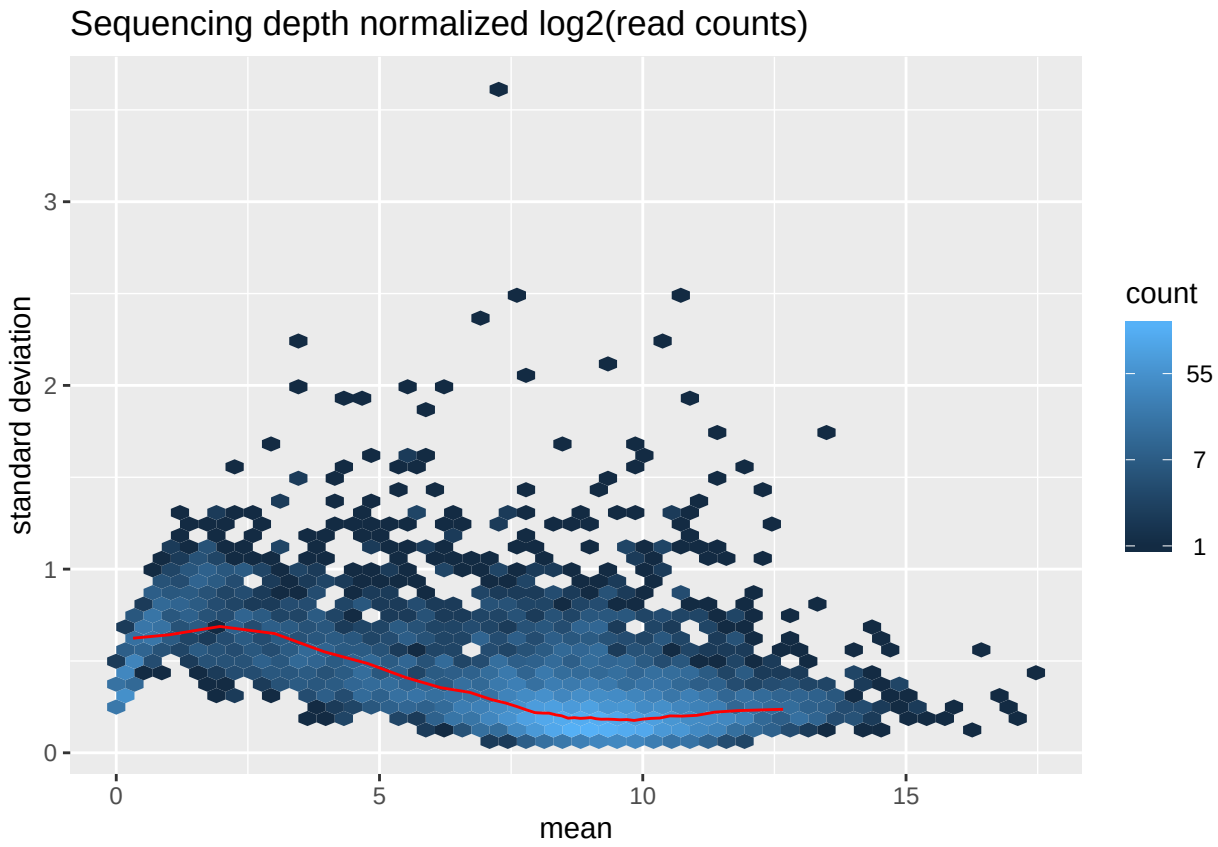
Every dot = one gene.

The fanning out of the points in the lower left corner (points below $2^5 = 32$) indicates that read counts correlate less well between replicates when they are low.

This observation indicates that the standard deviation of the expression levels may depend on the mean: the lower the mean read counts per gene, the higher the standard deviation.

This can be assessed visually; the package `vsn` offers a simple function for this.

```
par(mfrow=c(1,1))
# generate the base meanSdPlot using sequencing depth normalized log2(read counts)
msd_plot <- vsn::meanSdPlot(assay(DESeq.ds,  "log.norm.counts"),
                        ranks=FALSE, # show the data on the original scale
                        plot = FALSE) # return a ggplot2 object without printing it

# add a title and y-axis label to the ggplot2 object
msd_plot$gg +
  ggtitle("Sequencing depth normalized log2(read counts)") +
  ylab("standard deviation")
```



From the help for`meanSdPlot`: *The red dots depict the running median estimator (window-width 10 percent). If there is no variance-mean dependence, then the line formed by the red dots should be approximately horizontal.*

The plot here shows that there is some variance-mean dependence for genes with low read counts. This means that the data shows signs of *heteroskedasticity*.

Many tools expect data to be *homoskedastic*, i.e., all variables should have similar variances.

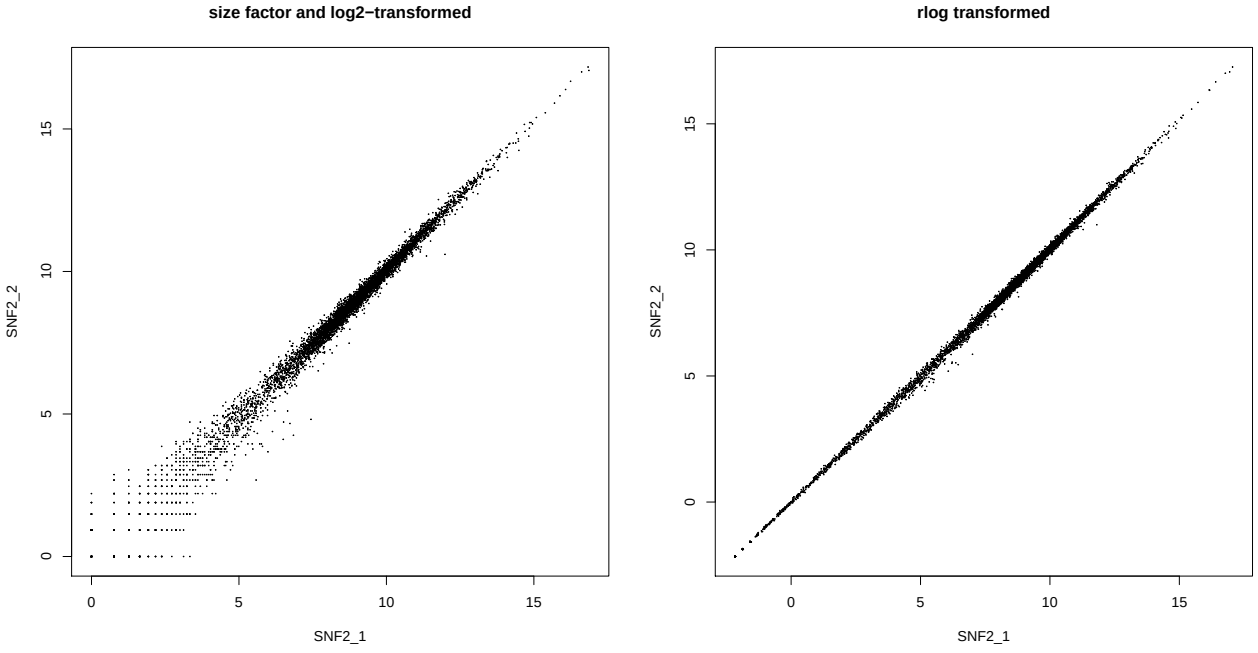
`DESeq2` offers two ways to shrink the log-transformed counts for genes with very low counts: `rlog` and

`varianceStabilizingTransformation` (`vst`).

We'll use `rlog` here as it is an optimized method for RNA-seq read counts: it transforms the size-factor-normalized read counts to the log2 scale while simultaneously minimizing the difference between samples for rows with small counts and taking differences between library sizes of the samples into account. `vst` tends to depend a bit more on the size factors, but generally, both methods should return similar results.

```
DESeq.rlog <- rlog(DESeq.ds, blind = TRUE) # this actually generates a
# different type of object
# set blind = FALSE if the conditions
# are expected to introduce strong differences in a large proportion of the genes
```

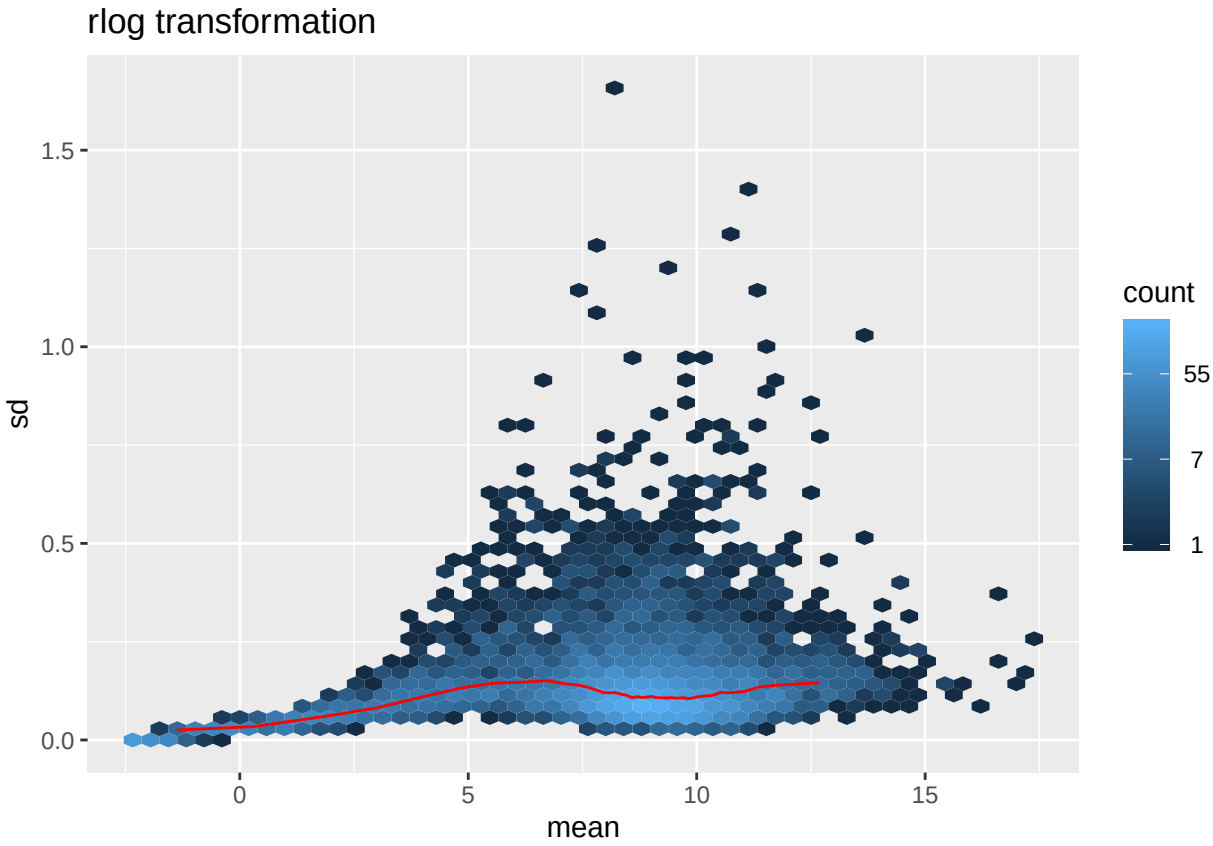
```
par(mfrow=c(1,2))
plot(assay(DESeq.ds[,1:2],  "log.norm.counts"), cex=.1,
     main = "size factor and log2-transformed")

# the rlog-transformed counts are stored in the accessor "assay"
plot(assay(DESeq.rlog)[,1],
     assay(DESeq.rlog)[,2],
     cex=.1, main = "rlog transformed",
     xlab = colnames(assay(DESeq.rlog[,1])),
     ylab = colnames(assay(DESeq.rlog[,2])) )
```




```
rlog.norm.counts <- assay(DESeq.rlog)
```

As you can see in the left plot the variance - that is higher for small read counts - is tightened significantly using `rlog`. What does the mean-sd-plot show?

```
# rlog-transformed read counts
msd_plot <- vsn::meanSdPlot( rlog.norm.counts, ranks=FALSE, plot = FALSE)
msd_plot$gg + ggtitle("rlog transformation")
```




```
# I don't want to save every time I compile; therefore eval=FALSE
save.image(file = "~/Documents/Teaching/2019_RNA-seq/Rclass.RData")
```

Similarity assessments and clustering

`pcaExplorer` lets you interact with the DESeq2-based plots and analyses. It has included hierarchical clustering of samples and PCA.

pcaExplorer

```
pcaExplorer::pcaExplorer(dds = DESeq.ds, dst = DESeq.rlog)
```

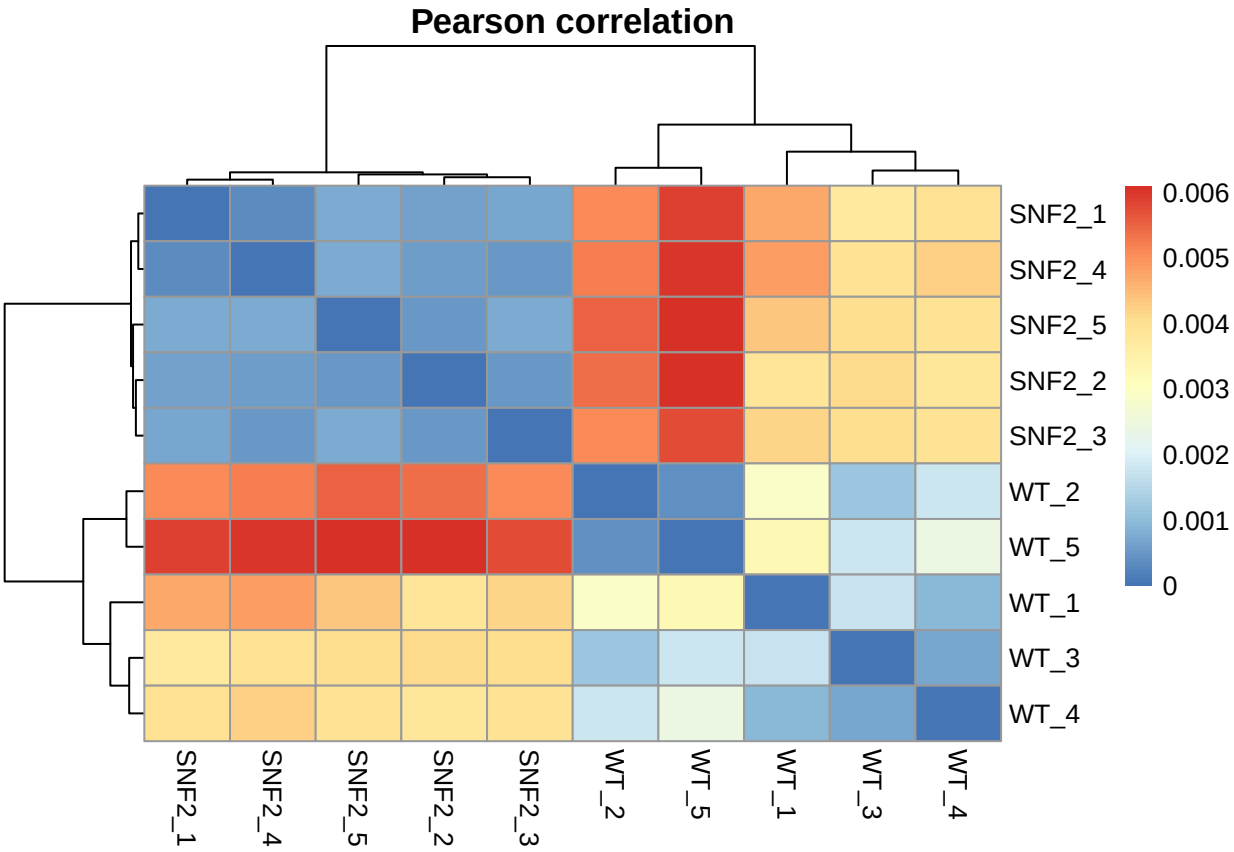
Sample clustering using Pearson correlation

The ENCODE consortium recommends that *"for messenger RNA, (. . .) biological replicates [should] display greater than 0.9 correlation for transcripts/features"*.

The Pearson correlation coefficient is a measure of the strength of the linear relationship between two variables and is often used to assess the similarity of RNA-seq samples in a pair-wise fashion. It is defined as the **covariance of two variables divided by the product of their standard deviation**.

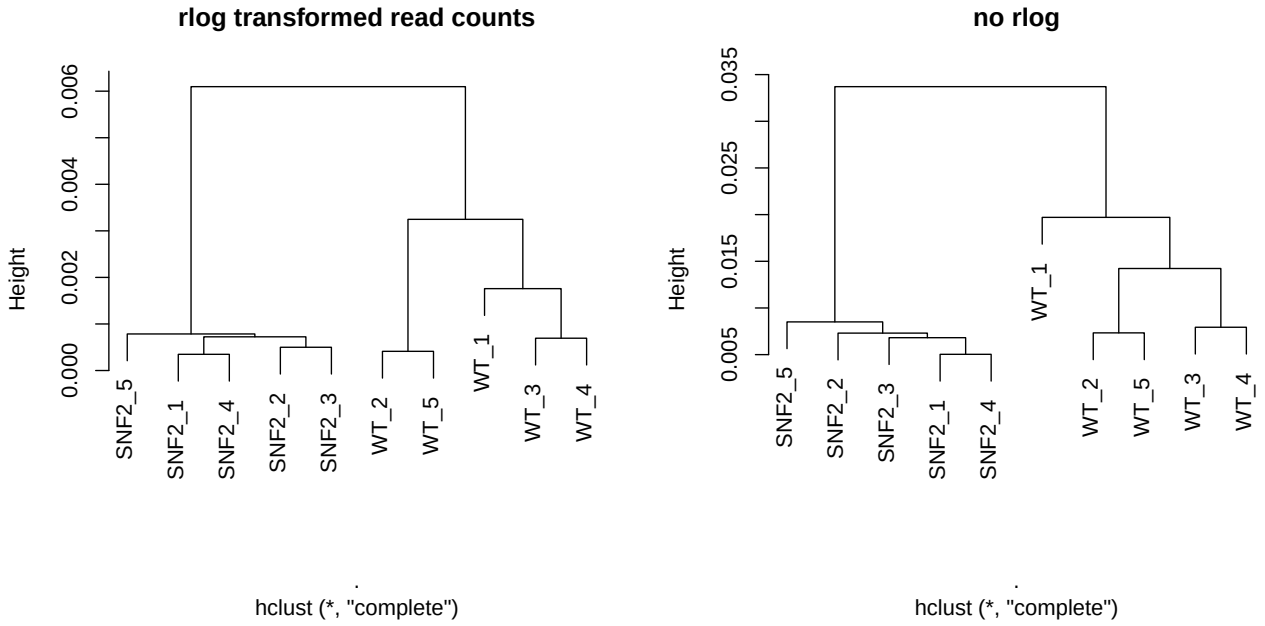
Mimicking `pcaExplorer`'s heatmap:

```
corr_coeff <- cor(rlog.norm.counts, method = "pearson")
as.dist(1-corr_coeff, upper = TRUE) %>% as.matrix %>%
    pheatmap::pheatmap(., main = "Pearson correlation")
```



Just plot the dendrogram, comparing the effects of the `rlog` transformation.

```
par(mfrow=c(1,2))
# Pearson corr. for rlog.norm values
as.dist(1 - corr_coeff) %>% hclust %>%
    plot( .,
        labels = colnames(rlog.norm.counts),
        main = "rlog transformed read counts")

# Pearson corr. for log.norm.values
as.dist( 1 - cor(assay(DESeq.ds, "log.norm.counts"),
    method = "pearson")) %>% hclust %>%
    plot( .,
        labels = colnames(assay(DESeq.ds, "log.norm.counts")),
        main = "no rlog")
```



How to do the PCA yourself (see the "protocol" part of pcaExplorer!)

```
rv <- rowVars(assay(DESeq.rlog)) # equivalent to rowVars(rlog.norm.counts)
top_variable <- order(rv, decreasing = TRUE)[seq_len(500)]
pca <- prcomp(t(assay(DESeq.rlog)[top_variable, ]))
head(pca$x)
```