Differential Expression with DESeq2

Laura Pikkupeura

5/31/2019

Teachers and slides

- **Laura Pikkupeura**
- PhD studenst in Sandelin and Jensen Labs
- Mail: `laura.pikkupeura@bric.ku.dk`
- Background:
  - MSc Molecular Biomedicine
  - PhD in Stem Cell Biology, Genomics and Transcriptomics

Lecture Outline

The plan:

- Walkthrough of DESeq2 core functionalities using the `yeast` dataset.
- You will analyze the `IBD` dataset on your own afterwards.
- If time allows, we will see how Salmon $>$ tximport $>$ DESeq2 can form a complete DE pipeline

The yeast dataset

Background

- Dataset from the paper: *A global non-coding RNA system modulates fission yeast protein levels in response to stress* by Leng *et al*.
- URL to paper: http://www.ncbi.nlm.nih.gov/pubmed/24853205
- RNA-Seq data for **6** fission yeast samples.
- Biological triplicates before/after stress treatment
- The data is a genes-by-samples count matrix.

Loading the needed packages

First we load the packages we need for the analysis:

```
library(DESeq2)
```

```
## Loading required package: S4Vectors

## Loading required package: stats4

## Loading required package: BiocGenerics

## Loading required package: parallel

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall,
##     clusterEvalQ, clusterExport, clusterMap,
##     parApply, parCapply, parLapply, parLapplyLB,
##     parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
```

Looking at the data

First we load the files into R:

```
# Information about samples
design <- read_tsv("yeast_study_design.tab")
```

```
## Parsed with column specification:
## cols(
##   Sample = col_character(),
##   Replicate = col_character(),
##   Minute = col_character()
## )
```

```
# Count matrix (This is not a tibble!)
EM <- read.table("yeast_count_matrix.tab") %>% as.matrix
```

Looking at the data

`design` contains information about the study setup:

```
design
```

```
## # A tibble: 6 x 3
##   Sample     Replicate Minute
##   <chr>      <chr>     <chr>
## 1 GSM1368273 r1        min0
## 2 GSM1368274 r2        min0
## 3 GSM1368275 r3        min0
## 4 GSM1368279 r1        min30
## 5 GSM1368280 r2        min30
## 6 GSM1368281 r3        min30
```

Looking at the data

`EM` is the expression matrix, quantified as counts:

```
head(EM)
```

```
##              GSM1368273 GSM1368274 GSM1368275 GSM1368279
## SPAC212.11            8          4         25          9
## SPAC212.09c          23         31         49         91
## SPNCRNA.70            0          0          0          0
## SPAC212.12            1          0          0          6
## SPAC212.04c          37          5         21         33
## SPAC212.01c           2          0          2          2
##              GSM1368280 GSM1368281
## SPAC212.11            7         10
## SPAC212.09c          73         75
## SPNCRNA.70            0          1
## SPAC212.12            0          0
## SPAC212.04c          32         54
## SPAC212.01c           0          2
```

Preparing the data for DESeq2

DESeq2 is NOT based on the tidyverse, but on Bioconductors elaborate S4-system.

That means we have to do a little bit of preparation to set up our data for analysis:

- A `matrix`-object containing counts.
- A `data.frame`-object containing information on the samples.
- A `formula`-object pointing to the column in the design holding the groups.

Preparing the data for DESeq2

First we must save the data as DESeqDatSet-object:

```
dds <- DESeqDataSetFromMatrix(countData = EM, # Count matrix
                              colData = design, # Study design
                              design = ~ Minute) # Groups
```

```
## Warning in DESeqDataSet(se, design = design, ignoreRank):
## some variables in design formula are characters, converting
## to factors
```

```
dds
```

```
## class: DESeqDataSet
## dim: 6642 6
## metadata(1): version
## assays(1): counts
## rownames(6642): SPAC212.11 SPAC212.09c ...
##   SPMITTRNAGLU.01 SPMIT.11
## rowData names(0):
## colnames(6): GSM1368273 GSM1368274 ... GSM1368280
##   GSM1368281
## colData names(3): Sample Replicate Minute
```

Running DESeq2

DESeq2 can now be run with a magical one-liner:

```
dds <- DESeq(dds)
```

```
## estimating size factors

## estimating dispersions

## gene-wise dispersion estimates

## mean-dispersion relationship

## final dispersion estimates

## fitting model and testing
```

Running DESeq2

What on earth is going on?!

The `DESeq`-function runs the following series of functions:

- `estimateSizeFactors`: Calculate normalization or size factors
- `estimateDispersion`: Estimate dispersion using information sharing
- `nbinomWaldTest`: Testing each gene using the Wald Test

Running all of these adds new information to the `DESeqDataSet`:

```
dds
```

```
## class: DESeqDataSet
## dim: 6642 6
## metadata(1): version
## assays(4): counts mu H cooks
## rownames(6642): SPAC212.11 SPAC212.09c ...
##   SPMITTRNAGLU.01 SPMIT.11
## rowData names(22): baseMean baseVar ... deviance
##   maxCooks
## colnames(6): GSM1368273 GSM1368274 ... GSM1368280
##   GSM1368281
## colData names(4): Sample Replicate Minute sizeFactor
```

Inspecting results

Now we can inspect the results, returned as DESeq2Results-object:

```
res <- results(dds)
head(res)
```

```
## log2 fold change (MLE): Minute min30 vs min0
## Wald test p-value: Minute min30 vs min0
## DataFrame with 6 rows and 6 columns
##                        baseMean     log2FoldChange
##                       <numeric>          <numeric>
## SPAC212.11   10.0109632255266 -0.126118196529359
## SPAC212.09c  60.4782840168949   1.48254964538345
## SPNCRNA.70  0.194571047885225   1.27572753497477
## SPAC212.12  0.892558500282679    2.3621339611395
## SPAC212.04c  30.5572252350283   1.45684488478917
## SPAC212.01c   1.1370313450568   0.38321099262347
##                         lfcSE               stat
##                       <numeric>          <numeric>
## SPAC212.11  0.783799781633654 -0.160906138894928
## SPAC212.09c 0.416326151875937   3.56102934851241
## SPNCRNA.70   4.08008003243857  0.312672184082697
## SPAC212.12   2.92014920712879   0.80890865280889
## SPAC212.04c 0.530281333523057   2.74730561438068
```

Inspecting results

Let us use the built in overall summary:

```
summary(res)
```

```
##
## out of 6642 with nonzero total read count
## adjusted p-value < 0.1
## LFC > 0 (up)       : 1283, 19%
## LFC < 0 (down)     : 1071, 16%
## outliers [1]       : 1, 0.015%
## low counts [2]     : 129, 1.9%
## (mean count < 0)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Customizing results

Take notice of the default settings:

- By default DESeq2 used min30 vs min0 as the comparison: We might want to change this!
- By default DESeq2 used a p-value threshold of 0.1: We might want to change this!
- By default DESeq2 used a logFC treshold of 0: We might want to change this!

Customizing results

Pass more arguments to `results`:

```
res2 <- results(dds,
                contrast=c("Minute", "min30", "min0"), # Comparison
                lfcThreshold=0.25, # logFC cutoff
                alpha=0.05) # p-value cutoff
```

Customizing results

What happened to the amount of DE genes?

```
summary(res2)
```

```
##
## out of 6642 with nonzero total read count
## adjusted p-value < 0.05
## LFC > 0.25 (up)     : 650, 9.8%
## LFC < -0.25 (down)  : 393, 5.9%
## outliers [1]        : 1, 0.015%
## low counts [2]      : 258, 3.9%
## (mean count < 1)
## [1] see 'cooksCutoff' argument of ?results
## [2] see 'independentFiltering' argument of ?results
```

Exporting to the tidyverse

In many cases, we want to continue working on the results produced by DESeq2 using the tidyverse. This can be done in two ways:

Manually coerce the `DESeqResults`-object to a data.frame or tibble:

```
res3 <-res2 %>%
    as.data.frame %>%
    rownames_to_column("Gene") %>%
    as_tibble

res3
```

```
## # A tibble: 6,642 x 7
##    Gene  baseMean log2FoldChange lfcSE    stat  pvalue
##    <chr>    <dbl>          <dbl> <dbl>   <dbl>   <dbl>
##  1 SPAC~   10.0           -0.126  0.784  0       1
##  2 SPAC~   60.5            1.48   0.416  2.96    0.00307
##  3 SPNC~    0.195          1.28   4.08   0.251   0.802
##  4 SPAC~    0.893          2.36   2.92   0.723   0.469
##  5 SPAC~   30.6            1.46   0.530  2.28    0.0229
##  6 SPAC~    1.14           0.383  1.98   0.0673  0.946
##  7 SPAC~    0.225         -0.655  4.08  -0.0992  0.921
##  8 SPNC~    8.40          -0.781  0.913 -0.581   0.561
##  9 SPAC~    0.317         -1.70   4.03  -0.361   0.718
```

Exporting to the tidyverse

Or have `results` return a normal data.frame:

```
results(dds, tidy = TRUE) %>%
    as_tibble
```

```
## # A tibble: 6,642 x 7
##    row   baseMean log2FoldChange lfcSE   stat  pvalue
##    <chr>    <dbl>          <dbl> <dbl>  <dbl>   <dbl>
##  1 SPAC~   10.0           -0.126  0.784 -0.161 8.72e-1
##  2 SPAC~   60.5            1.48   0.416  3.56  3.69e-4
##  3 SPNC~    0.195          1.28   4.08   0.313 7.55e-1
##  4 SPAC~    0.893          2.36   2.92   0.809 4.19e-1
##  5 SPAC~   30.6            1.46   0.530  2.75  6.01e-3
##  6 SPAC~    1.14           0.383  1.98   0.194 8.46e-1
##  7 SPAC~    0.225         -0.655  4.08  -0.160 8.73e-1
##  8 SPNC~    8.40          -0.781  0.913 -0.855 3.93e-1
##  9 SPAC~    0.317         -1.70   4.03  -0.423 6.72e-1
## 10 SPAC~   83.0           -0.0855 0.265 -0.322 7.48e-1
## # ... with 6,632 more rows, and 1 more variable: padj <dbl>
```

Looking at top genes

Have a look at the top few genes:

```
arrange(res3, padj)
```

```
## # A tibble: 6,642 x 7
##    Gene  baseMean log2FoldChange lfcSE  stat    pvalue
##    <chr>    <dbl>          <dbl> <dbl> <dbl>     <dbl>
##  1 SPAC~   21418.           8.22 0.162  49.3 0.
##  2 SPCC~   14009.           6.01 0.132  43.8 0.
##  3 SPBC~    5899.           5.17 0.145  33.8 9.98e-251
##  4 SPAC~    4700.           7.80 0.228  33.2 2.20e-241
##  5 SPBC~    6758.           6.50 0.189  33.0 1.56e-238
##  6 SPAC~    9670.           7.83 0.259  29.2 6.75e-188
##  7 SPBC~    6834.           7.71 0.269  27.8 1.02e-169
##  8 SPCP~    4574.           6.79 0.242  27.1 1.90e-161
##  9 SPBC~    8371.           3.63 0.125  27.0 1.78e-160
## 10 SPAC~    6658.           5.30 0.188  26.9 3.25e-159
## # ... with 6,632 more rows, and 1 more variable: padj <dbl>
```

Diagnostics

Diagnostics

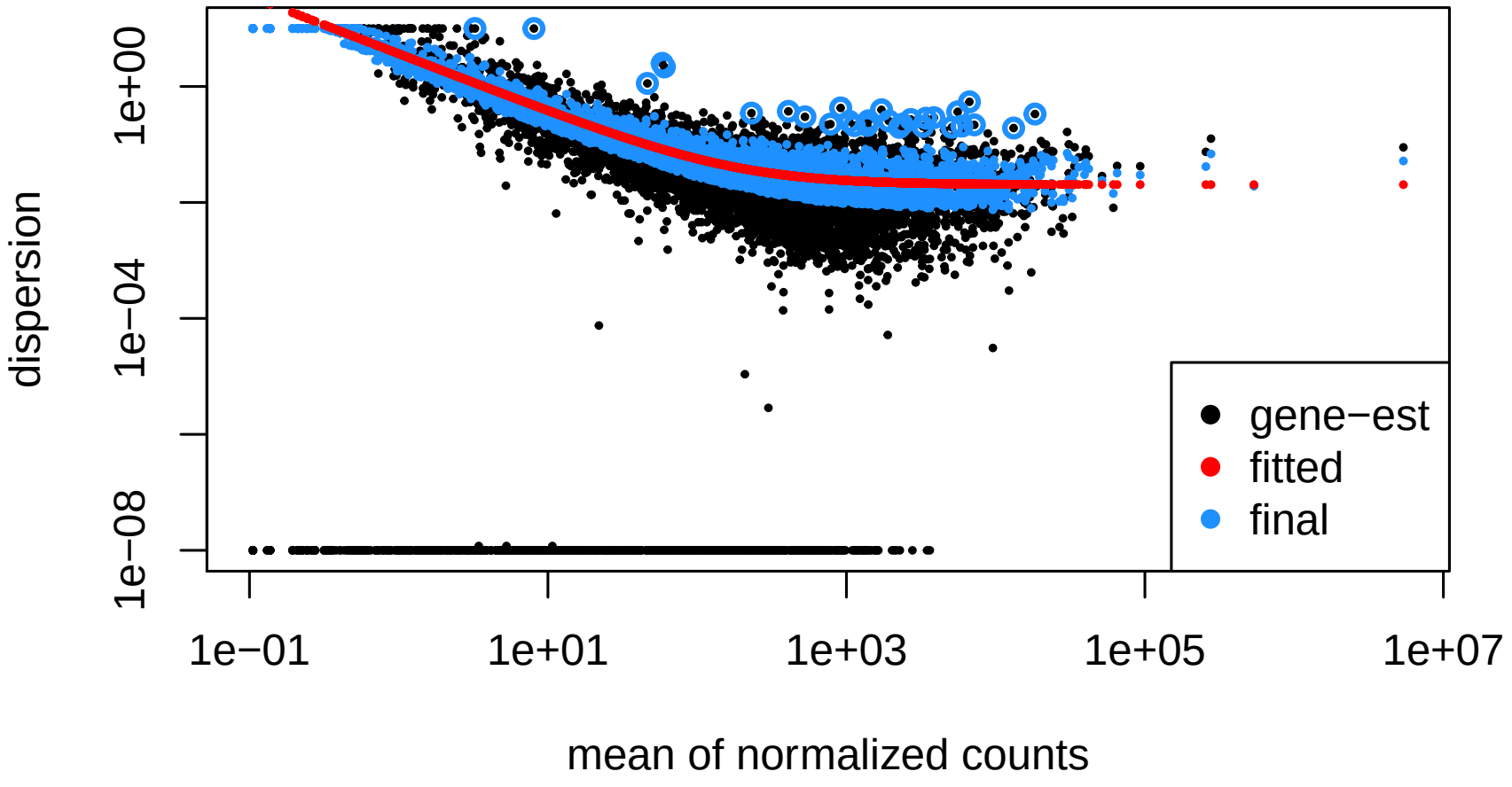
It is important to make sure the DESeq2 analysis was appropriate by inspecting some key diagnostic plots:

- Dispersion-plot
- MA-plot
- p-value distribution
- Independent filtering plot
- Volcano plot

Dispersion plot:

We use the built-in function to inspect to dispersion estimation:

```
plotDispEsts(dds)
```

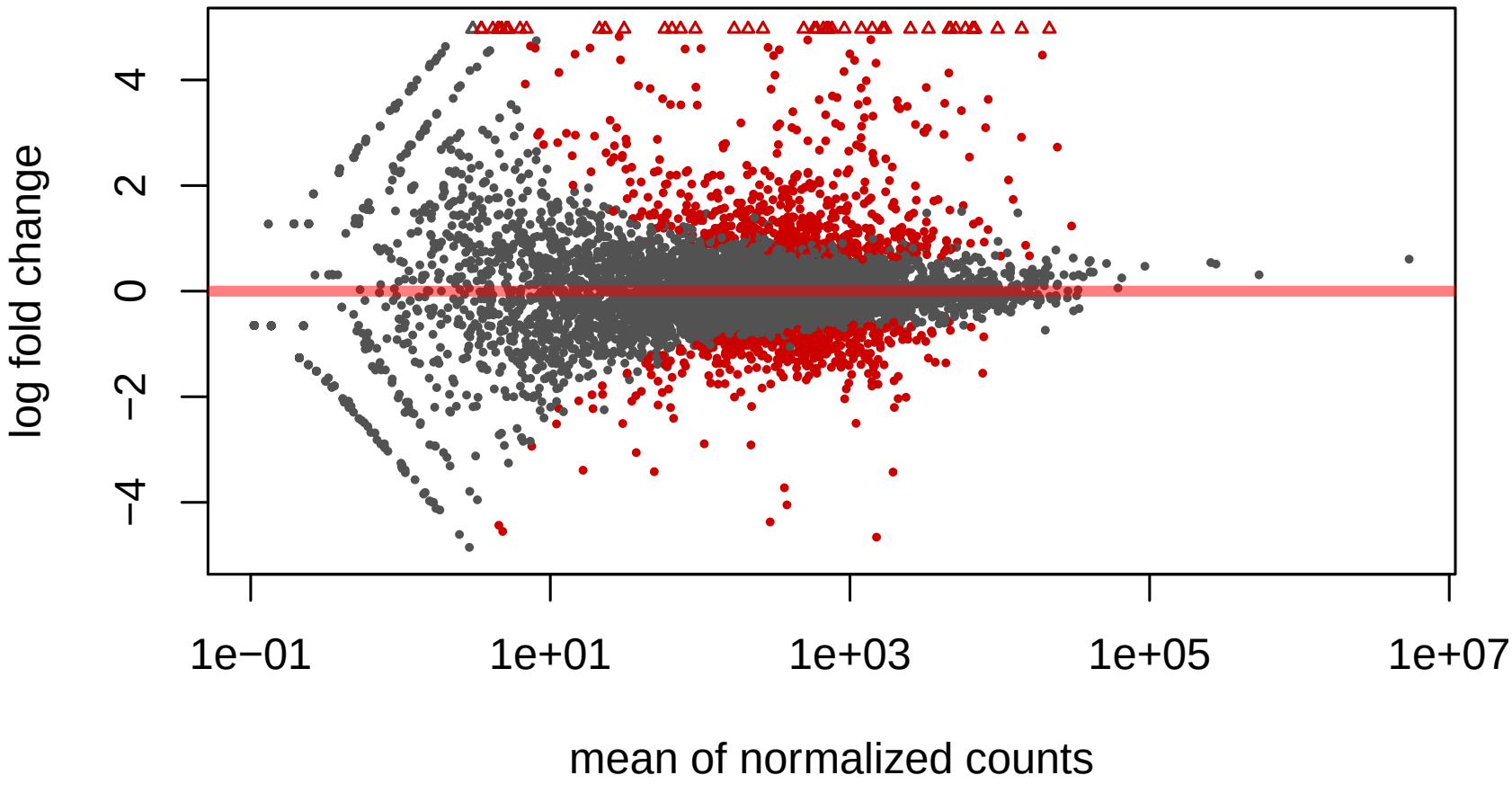


MA-plot

Similarly, we can use the built-in function for generating an MA-plot:

```
plotMA(res2)
```

want to see symm + most val are around 0 + less fold change in highly exp genes

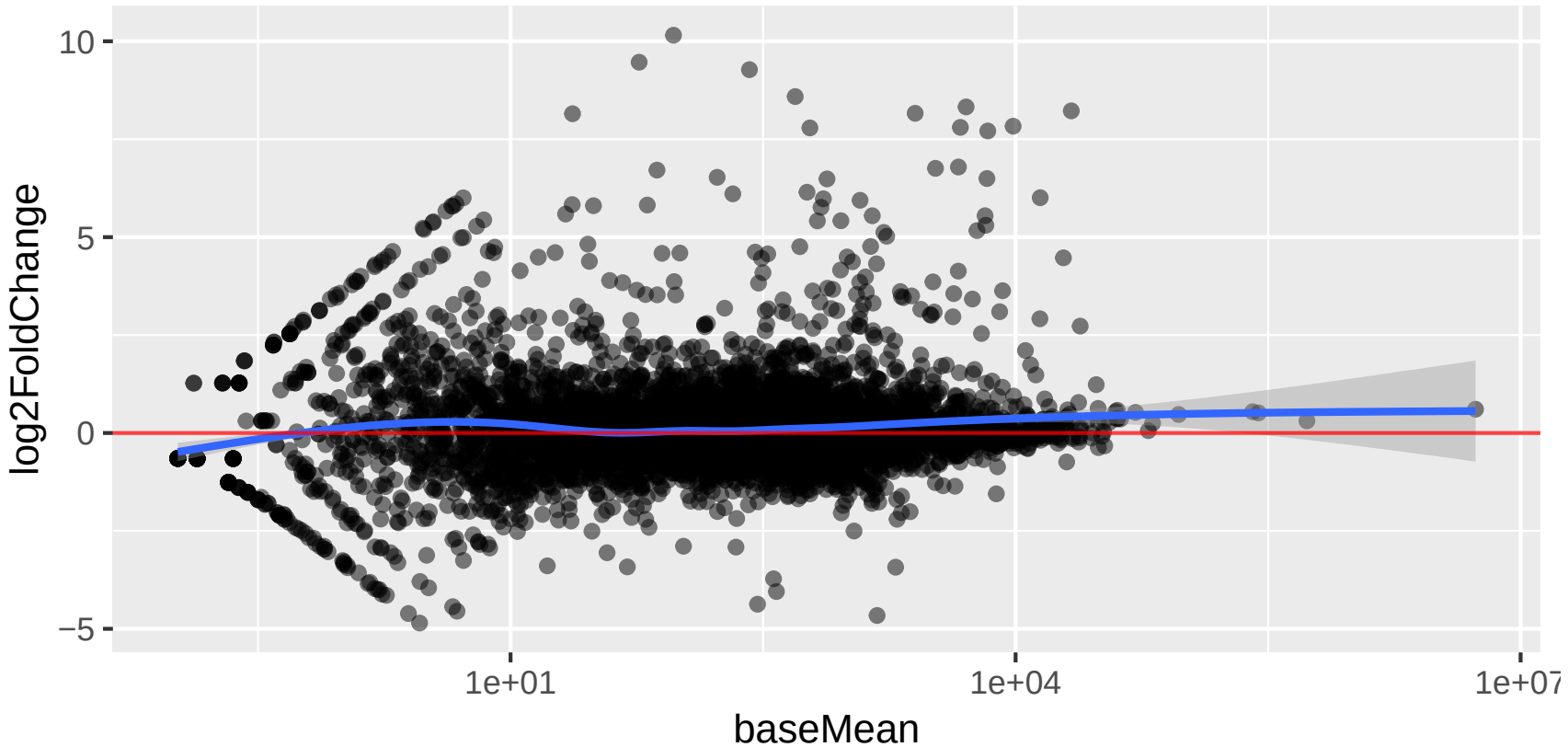


MA-plot

Or make our own using the tidyverse (Why are we using the res3 object?):

```
ggplot(res3, aes(x=baseMean, y=log2FoldChange)) +
    geom_point(alpha=0.5) + geom_smooth() + scale_x_log10() +
    geom_hline(yintercept = 0, alpha = 0.75, color="red")
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs"
```

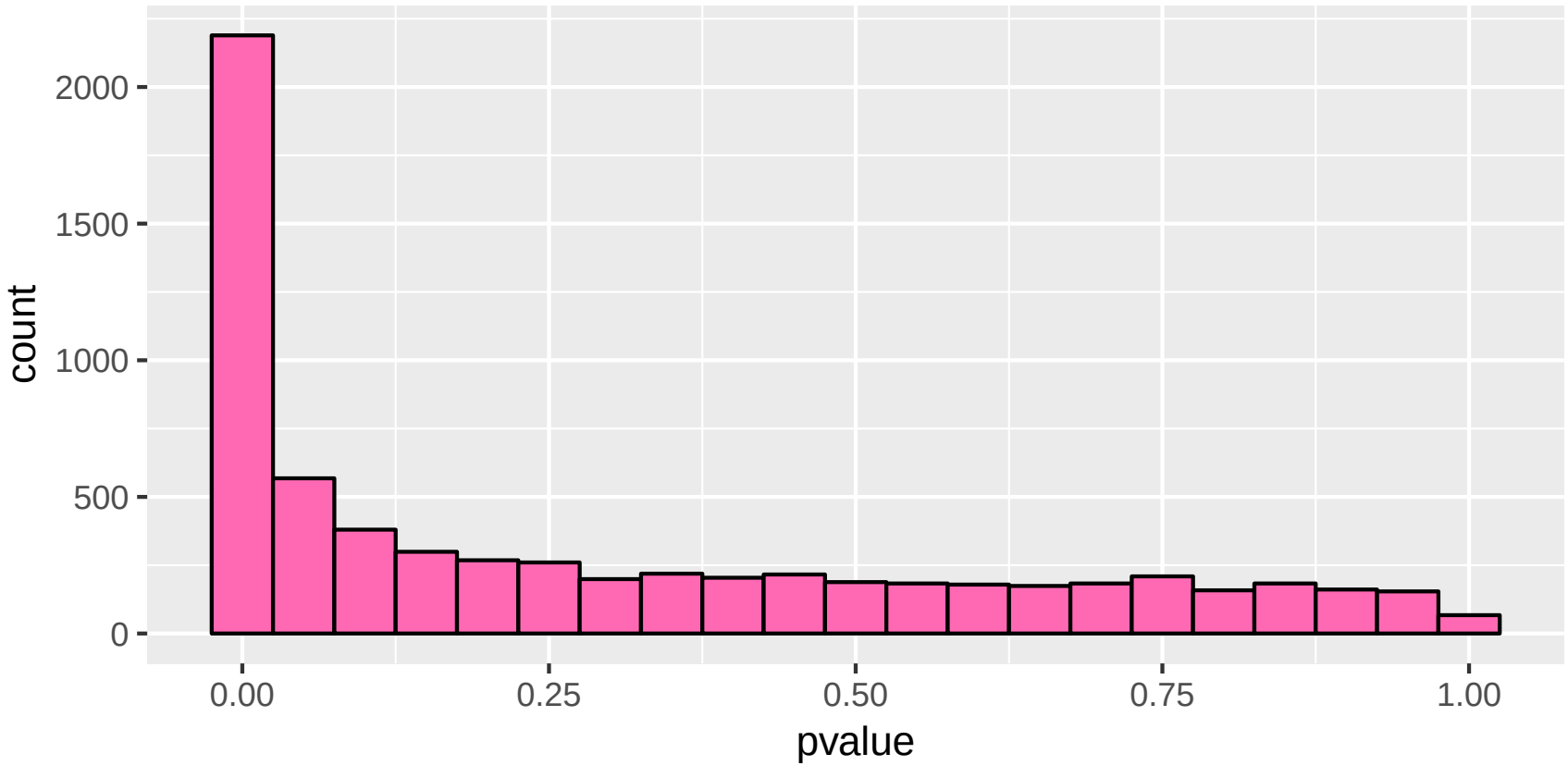


p-value distribution

This plot is easy to do with the tidyverse:

```
ggplot(as.data.frame(res), aes(x=pvalue)) +
    geom_histogram(binwidth = 0.05, fill="hotpink", color="black")
```

```
## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```

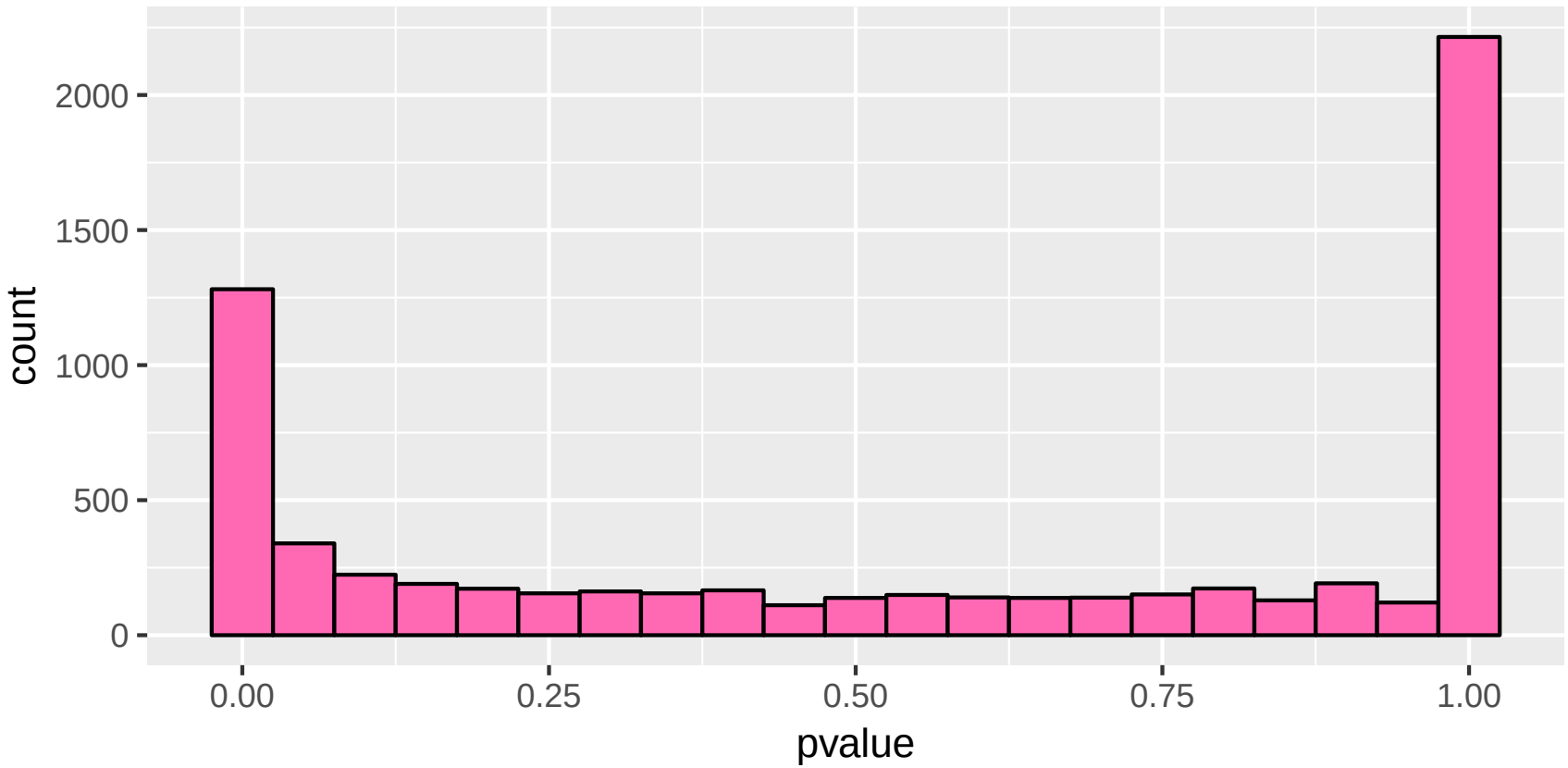


p-value distribution

Note, if you use a logFC threshold other than zero, the distribution will not be uniform:

```
ggplot(res3, aes(x=pvalue)) +
    geom_histogram(binwidth = 0.05, fill="hotpink", color="black")
```

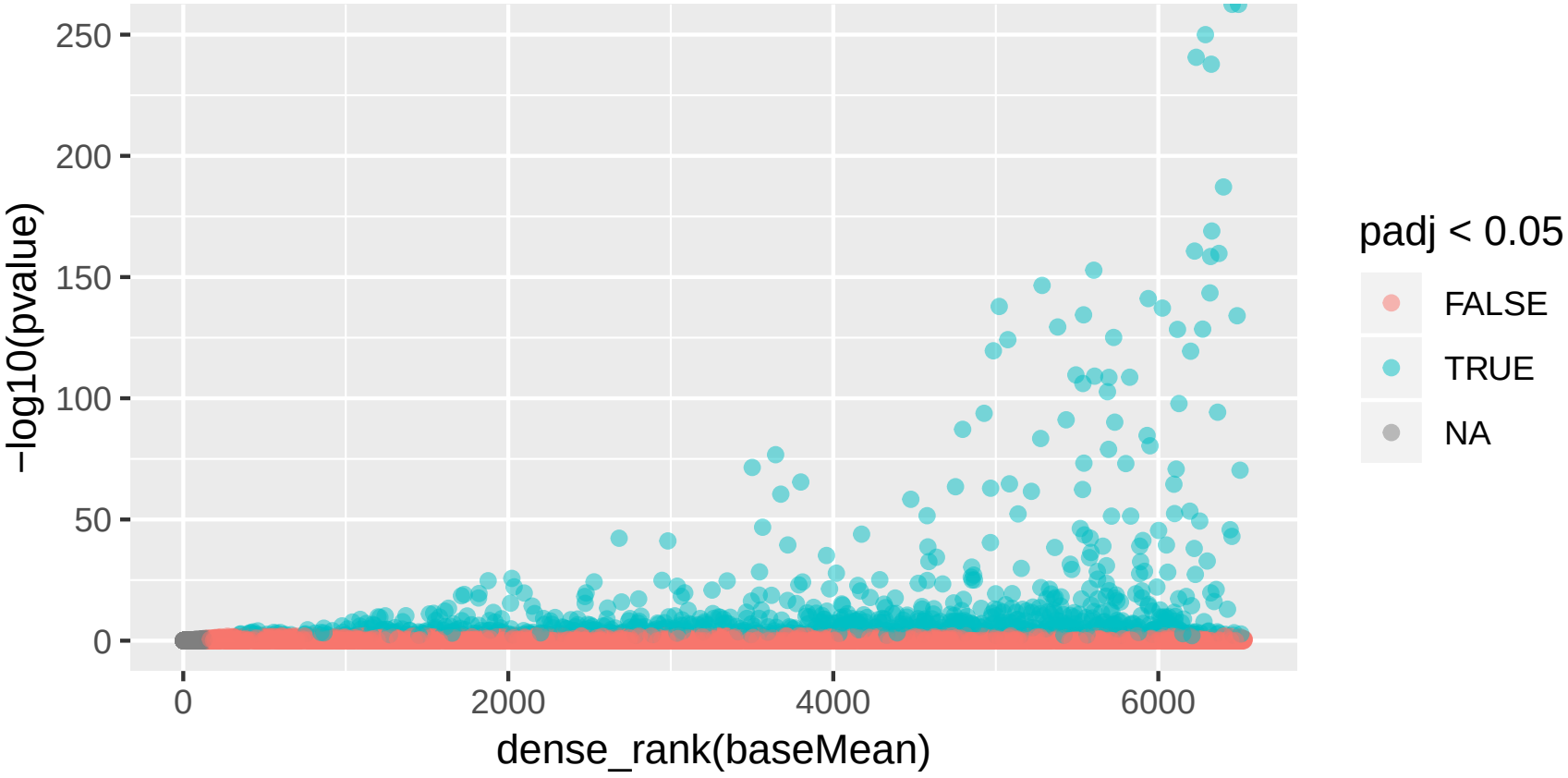
```
## Warning: Removed 1 rows containing non-finite values
## (stat_bin).
```



Independent filtering

```
ggplot(res3, aes(x=dense_rank(baseMean), y=-log10(pvalue),
                 color=padj < 0.05)) + geom_point(alpha=0.5)
```

```
## Warning: Removed 1 rows containing missing values
## (geom_point).
```

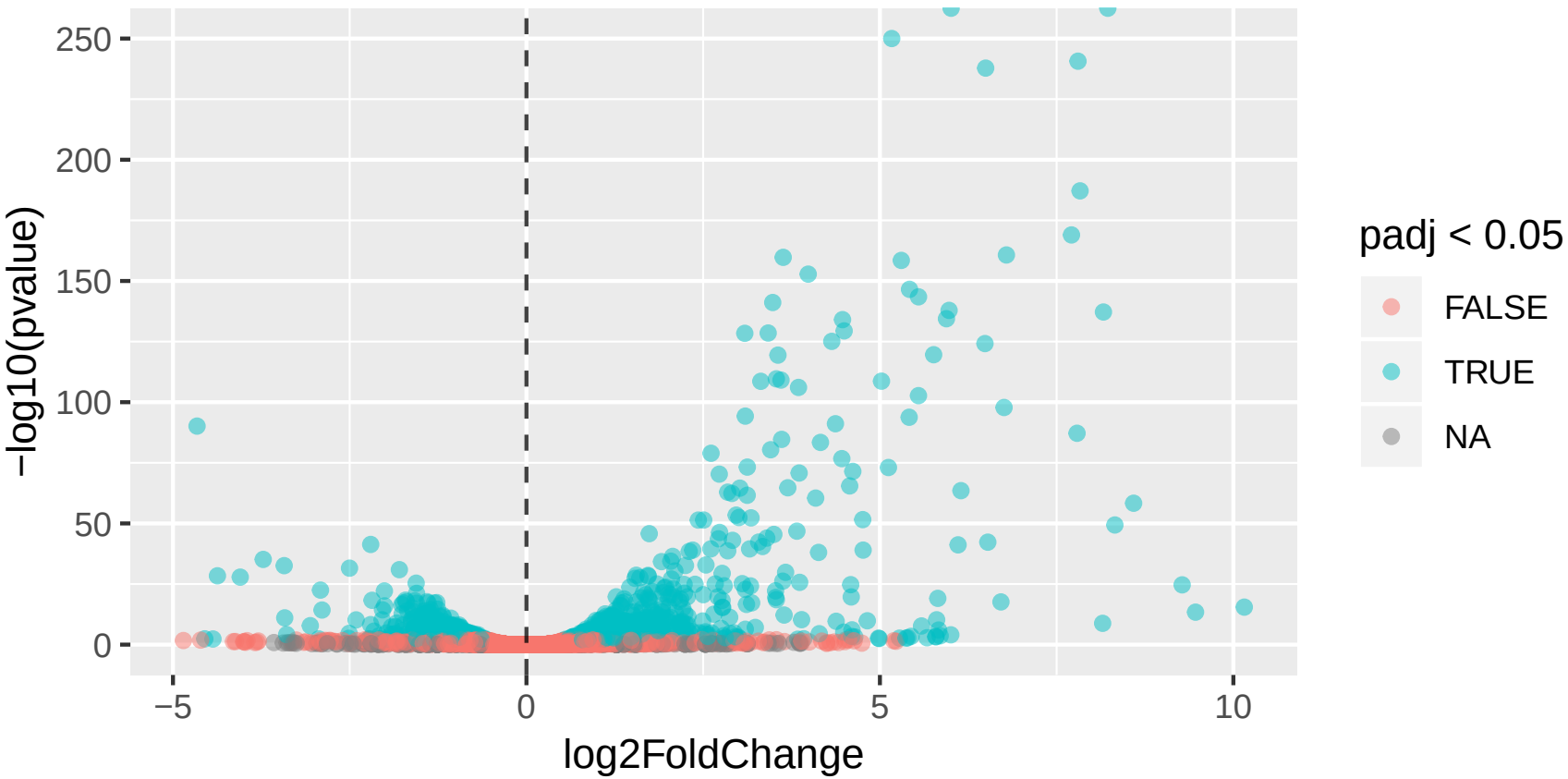


Volcano plot

This plot is easy to do with the tidyverse:

```
ggplot(res3, aes(x=log2FoldChange, y=-log10(pvalue),
                 color=padj < 0.05)) + geom_point(alpha=0.5) +
    geom_vline(xintercept = 0, alpha = 0.75, linetype="dashed")
```

```
## Warning: Removed 1 rows containing missing values
## (geom_point).
```



DESeq2 check-list

Make sure:

- You can assume most genes are not changing.
- You are comparing the correct groups.
- You are using the desired p-value and logFC thresholds.
- Shrink-plot indicating stable shrinkage procedure.
- MA-plot indicating appropriate normalization
- p-value histogram is uniform towards one, indicating well-specified model (i.e. no batch effects). Except if you use a logFC cutoff different from 0
- Check whether you are including many genes too lowly expressed to be DE using the independent filtering plot
- Look at the overall relation between effect sizes and significance using a volcano plot.
- If using Salmon (or Kallisto) use tximport together with DESeq2 - more on this later!

Enjoy your well-calibrated DESeq2 results!

The IBD dataset

Background

- Dataset from the paper: *Characterization of the enhancer and promoter landscape of inflammatory bowel disease from human colon biopsies* by Boyd *et al*.
- URL to paper: https://www.ncbi.nlm.nih.gov/pubmed/29695774
- CAGE-data for 12 Inflammatory Bowel Disease (IBD) patient colonic biopsies.
- The tissue samples have been categorized by pathologists:
  - `con`: Healthy individuals
  - `CDa`: Crohn's disease with active inflammation
  - `UCa`: Ulcerative Colitis with active inflammation
- The data contains gene-level CAGE counts.

Practical

Repeat the previous analysis on the CAGE data:

- Setup and fit the DESeq2 model
- Do two analyses: 1) p-value threshold of 0.05 and logFC treshold of 0 & 1) p-value threshold of 0.05 and logFC treshold of 0.5
- Produce and inspect all diagnostic plots.
- How many genes are DE between the groups? Are there more up- or downregulated genes?
- Which groups are more similar based on the number of DE genes?
- Manually inspect some of the top genes characterising: a) both CDa and UCa vs con and b) CDa vs UCa. What kinds of genes are they? What do they do?

Salmon => tximport => DESeq2

tximport

Salmon and DESeq2 are developed by the same people (Primarily Michael Love).

- To improve interconnectivity between the packages, they released the tximport package which can read the output of Salmon (and Kallisto).
- DESeq2 can directly use data from tximport
- This allows DESeq2 to include the modelled effective lengths, GC-biases, etc. at both transcript- and gene-levels.

Below is a small example of how to use DESeq2 with the tximport package:

```
library(tximport)
```

tximport

The course website includes an example salmon output. We first load the design:

```
design <- read_tsv("salmon_study_design.tab")
```

```
## Parsed with column specification:
## cols(
##   pop = col_character(),
##   center = col_character(),
##   assay = col_character(),
##   sample = col_character(),
##   experiment = col_character(),
##   run = col_character(),
##   condition = col_character()
## )
```

```
design
```

```
## # A tibble: 6 x 7
##   pop   center assay      sample experiment run    condition
##   <chr> <chr>  <chr>      <chr>  <chr>      <chr>  <chr>
## 1 TSI   UNIGE  NA20503.1~ ERS18~ ERX163094  ERR18~ A
## 2 TSI   UNIGE  NA20504.1~ ERS18~ ERX162972  ERR18~ A
```

tximport

Then we local all the files from the salmon folders:

```
quant_files <- file.path("salmon", design$run, "quant.sf.gz")
```

And read them in using tximport:

```
txi_transcripts <- tximport(quant_files, type="salmon", txOut = TRUE)
```

```
## reading in files with read_tsv
```

```
## 1 2 3 4 5 6
```

tximport to DESeq2

Then we can use the specialized import function:

```
DESeqDataSetFromTximport(txi=txi_transcripts,
                         colData=design,
                         design=~condition)
```

```
## Warning in DESeqDataSet(se, design = design, ignoreRank):
## some variables in design formula are characters, converting
## to factors

## using counts and average transcript lengths from tximport

## class: DESeqDataSet
## dim: 200401 6
## metadata(1): version
## assays(2): counts avgTxLength
## rownames(200401): ENST00000456328.2 ENST00000450305.2
##   ... ENST00000387460.2 ENST00000387461.2
## rowData names(0):
## colnames: NULL
## colData names(7): pop center ... run condition
```

The rest of the analysis is the same!

Going further

tximport has many advantages:

- tximport can do sophisticated aggregation of transcripts to genes. This requires a transcript-to-gene map. The vignette of the tximport package shows how to do this.
- Using the `DESeqDataSetFromTximport` function carries over all information from Salmon and tximport into DESeq2. This information is seamlessly included in the DESeq2 analysis using the standard pipeline.
- tximport also works for other pseudoaligners such as Kallisto and other DE tools such as edgeR and limma.