RNASeq analysis

Table of content

Technical Prerequisites

Command-line interface

The first steps of the analyses - that are the most computationally demanding - will be performed on our Linux Server or Workstation. The interaction with the server is completely text-based, i.e., there will be no graphical user interface. We will instead be communicating entirly via command line using UNIX shell scripting language **bash**. You can find a good introduction into the **shell** basics at http://linuxcomand.org/lc3_learning_the_shell.php (//http://linuxcomand.org/lc3_learning_the_shell.php), https://git.embl.de/grp-bio-it/linuxcommandline (https://git.embl.de/grp-bio-it/linuxcommandline), or in the jupyter notebook Command_Line.ipynb in the same folder as this notebook.

Programs that will use the command line:

- FastQC http://www.bioinformatics.babraham.ac.uk/projects/fastqc/ (http://www.bioinformatics.babraham.ac.uk/projects/fastqc/)
- BCL2FASTQ http://emea.support.illumina.com/sequencing/sequencing_software/bcl2fastq-conversion-software.html (http://emea.support.illumina.com/sequencing/sequencing_software/bcl2fastq-conversion-software.html)
- featureCounts http://bioinf.wehi.edu.au/subread-package/ (http://bioinf.wehi.edu.au/subread-package/))
- MultiQC http://multiqc.info/docs/ (http://multiqc.info/docs/)
- QoRTs https://hartleys.github.io/QoRTs/ (https://hartleys.github.io/QoRTs/)
- RSeQC http://rseqc.sourceforge.net/ (http://rseqc.sourceforge.net/)
- samtools http://www.htslib.org/ (http://www.htslib.org/)
- STAR https://github.com/alexdobin/STAR (https://github.com/alexdobin/STAR)
- UCSC tools https://hgdownload.soe.ucsc.edu/admin/exe/ (https://hgdownload.soe.ucsc.edu/admin/exe/)

The only program with a graphical user interface will be IGV.

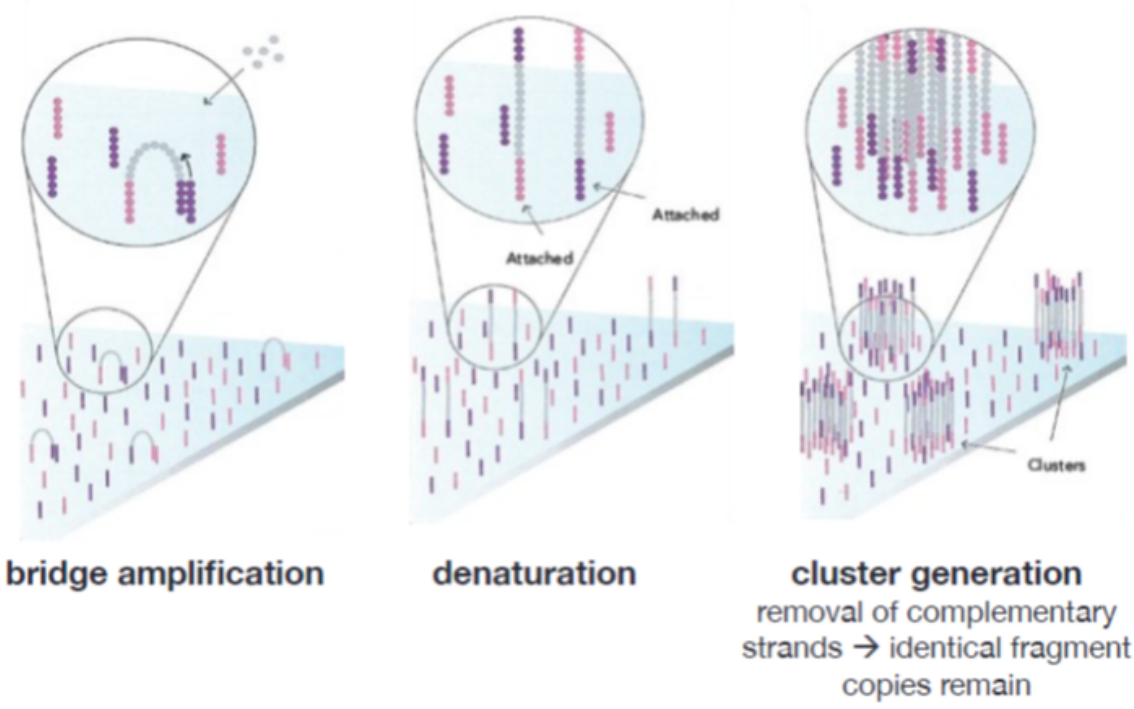
- igv browser https://www.broadinstitute.org/igv/ (https://www.broadinstitute.org/igv/)

R

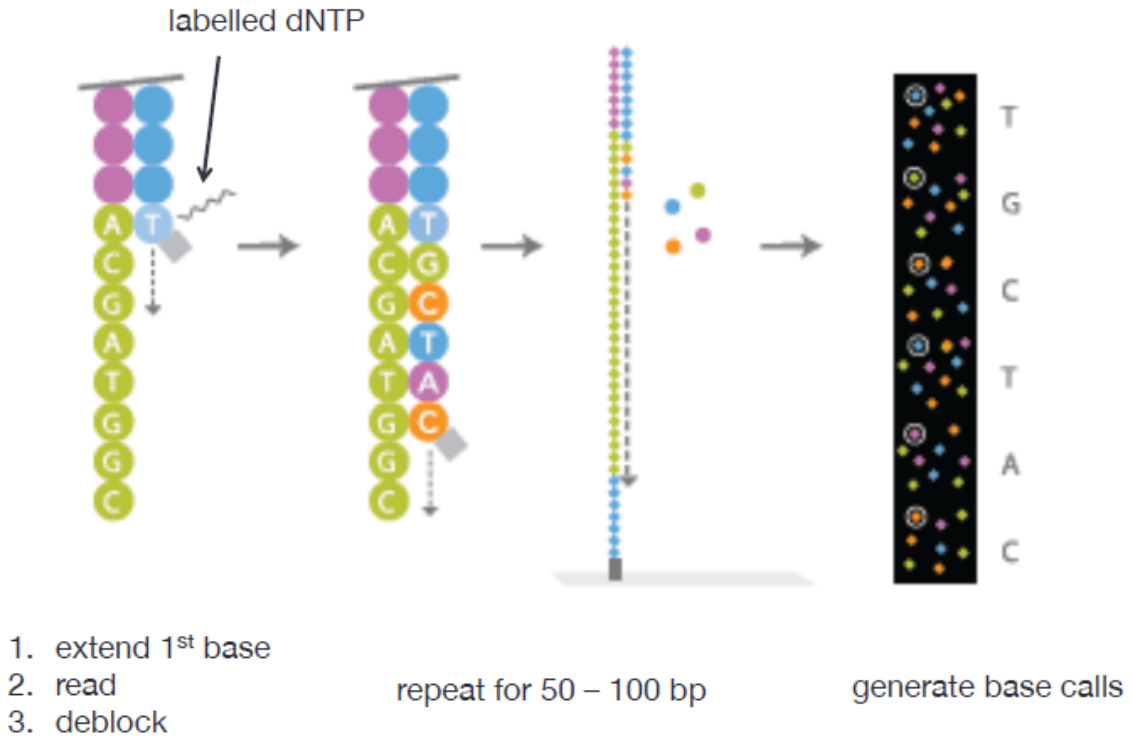
The second part of the analyses - where we will need support for statistics and visualization more than pure computation power - will mostly be done on the individual computers using the programming language R. You can download R from http://cran.rstudio.com/ (//http://cran.rstudio.com/). After you installed R, its highly recommended to install RStudio http://www.rstudio.com/products/rstudio/ (//http://www.rstudio.com/products/rstudio/), which will provide an interface to write commands at a prompt, construct script and view plots all in a single integrated environment. An introduction to R is available at https://descostesn.github.io/RIntroProgBio/ (https://descostesn.github.io/RIntroProgBio/).

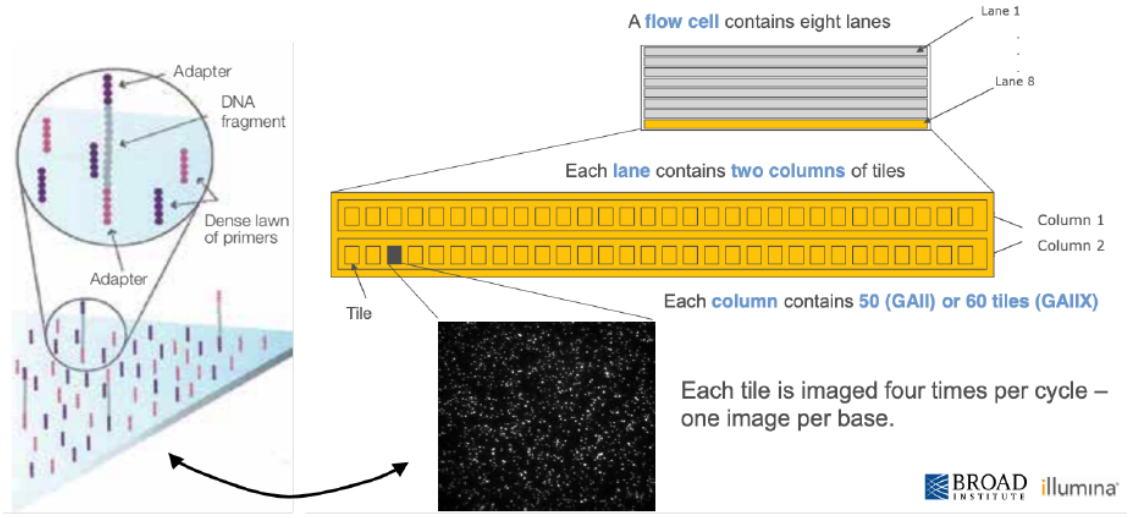
Illumina Sequencing

After hybridization of the DNA fragments to the flowcell through means of *adapters*, each fragment is massively and clonally amplified, forming *clusters* of double-stranded DNA. This step is neccessary to ensure that the sequencing signal will be strong enough to be detected for each base of each fragment.



The sequencing of the fragment is based on fluorophore labelled dNTPs with reversible terminator elements that will become incorporated and excited by a laser one at a time and thereby enable the optical identification of single bases. Images are taken after each round of nucleotide incorpration and bases are identified based on the recorded excitaton spectra. The information gets saved in bcl. files (base calling files).





https://www.illumina.com/documents/products/techspotlights/techspotlight_sequencing.pdf

https://www.broadinstitute.org/files/shared/illuminavids/sequencingSlides.pdf

Raw Sequencing Reads

Each sequencing run produces log files, instrument health data, run metrics, base call information (.bcl files), and other data. The base call information is demultiplexed to create the samples used in secondary analysis.

Sample Sheets

The sample sheet is a comma-delimited file (SampleSheet.csv) that stores the information needed to set up and analyze a sequencing experiment. The file includes a list of samples and their index sequences, as well as the workflow to be employed. Use Illumina Experiment Manager software to set up a sample sheet for your library prep protocol.

The Sample Sheet is a file format used by Illumina for storing biological sample information and metadata associated with a given experiment. The file is used widely across the Illumnina informatics ecosystem as an input to many pieces of software, such as bcl2fastq. The Sample Sheet for sequence data uses

American Standard Code for Information Interchange (ASCII) character encoding. This plain-text, comman-delimited format supports multiple sections with metadata describing experimental setup, demultiplexing settings, or analysis options. A sample sheet (SampleSheet.csv) records information about samples and corresponding index adapters. The bcl2fastq2 Conversion Software uses this information to demultiplex and convert BCL files. For most runs, a sample-sheet is optional. The default location is the root output folder, but you can use the command --sample-sheet to specify any CSV file in any location. When a sample sheet is not provided, the software assigns all reads to the default sample Undertermined_S0.

Sections

The Sample Sheet format is divided into multiple sections, which are denoted by a line starting with a section label. The section label starts with an open square bracket containing a string of text for the section name, and ending with a closing square bracket.

Header

The Header section is required, and must be located on the first line of the Sample Sheet file. The Header contains informational fields describing the context aroung which a sequencing run or analysis was performed.

Header records are represented as a series of key-value pairs. As such, each line requires exactly two fields. The first field in each line is the "key", which names the piece of metadata being recorded. Each key in the Header section must be unique. The second field in each line is the "value", which is the actual piece to metadata being recorded. Values do not necessarily need to be unique.

Example of a legal "Header" section containing records describing "Date" and "Investigator":

```
[Header]
Date, 2019-06-24
Worklfow, GenerateFASTQ
Investigator, Sebastian Heucke
```

Settings

Settings is an optional section used to control various parameters for some Illumina software, such as bcl2fastq. If no settings section is present in the Sample Sheet, the software will assume the default values for all options. Settings records are represented as a series of key-value pairs. As such, each line requires exactly two fields. The first field in each line is the "key", which is the name of the options being configured. Each option in the Settings section can only be used once. The second field in each line is the "value", which is the actual piece of metadata being recorded. Values do not necessarily need to by unique.

```
[Settings]
Adapter, AGATGGACGAGAG
AdapterRead2, AGTACTGAT
```

Reads

The Reads section describes the number of sequencing cycles used for read 1 and read 2. Reads records are represented as single positive interger per read. As such, each line requires exactly one field. For single-end sequencing, there should be exactly one record. For paired-end sequencing, there should be exactly two records with the first representing read 1 and the second representing read 2. Index reads are not included in this section.

```
[Reads]
20
75
```

Manifests

Manifests is a section required by some secondary analysis workflows for targeted sequencing to specify a manifest file that contains the targets of interest.

Settings records are represented as a series of key-filename pairs. As such, each line requires exactly two fields. The first field in each line is the "key" which is an arbitrary string. The second field in each line is the "filename", which is either the name or full path of the Manifest file including the file extension. The Manifest file must reside in the same run folder as the SampleSheet.csv file if a full path to the Manifest is not provided. Each key and filenmae in the Manifests section should be unique.

The Data section should also contain a Manifest column to assign a Manifest to each sample. Each sample can be assigned exactly one Manifest, but different samples may be assigned different Manifests.

```
[Manifests]
A, TruSeqAmplicons_1.txt
B, TruSeqAmplicons_2.txt
```

Data

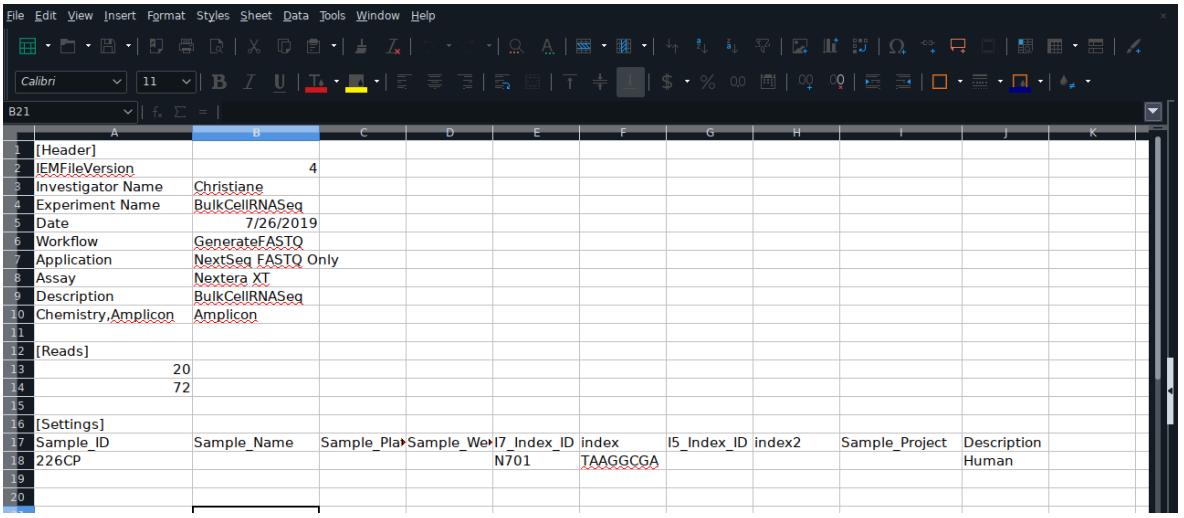
The Data section is required and must be located at the end of the Sample Sheet file. The Data section is a table and captures all sample-specific metadata.

The line immediately following the Data section label is a header line. The Data header line lists the names of each of the table columns in the section, separated by commas. No specific ordering of the column names is required and the are not case-sensitive. Each column name may only appear in data header line once.

At a minimum, the one column that is universally required is Sample_ID, which provides a unique string identifier for each sample. Immediately following the Data header line are the records for each sample. Each record spans exactly one line and must contain as many comma-separated fields as there are column names in the Data header line.

```
[Data]
Sample_ID, Sample_Name, Sample_project_, index
A1, Sample1,
```

Column	Description
Lane	When specified, the software generates FASTQ files only for the samples with the specified lane number.
Sample_ID	The sample ID.
Sample_Name	The sample name.
Sample_Project	The sample project name. The software creates a directory with the specified sample project name and puts the FASTQ files there. You can assign multiple sample to the same project.
index	The Index 1 (i7) index adapter sequence.
index2	The Index 2 (i5) index adapter sequence.



	A	B	C	D	E	F	G	H	I	J	K
1	[Header]										
2	IEMFileVersion	4									
3	Investigator Name	Christiane									
4	Experiment Name	BulkCellRNASeq									
5	Date	7/26/2019									
6	Workflow	GenerateFASTQ									
7	Application	NextSeq FASTQ Only									
8	Assay	Nextera XT									
9	Description	BulkCellRNASeq									
10	Chemistry,Amplicon	Amplicon									
11											
12	[Reads]										
13	20										
14	72										
15											
16	[Settings]										
17	Sample_ID	Sample_Name	Sample_Pla	Sample_We	I7_Index_ID	index	I5_Index_ID	index2	Sample_Project	Description	
18	226CP				N701	TAAGGCGA				Human	
19											
20											

The Sample_Project, Sample_ID, and Sample_Name columns accept alphanumeric characters, hyphens (~), and underscores. Do not use "all" or "unkonwn" as sample ID, all or undetermined as sample name, or all or default as the sample project name. Samples with these terms are omitted from the report.

Demultiplexing Scenarios

In order to demultiplex the data, first copy the entire run folder from the sequencer to the Linux Server. On the NextSeq500, the run folder will be inside the following directory on the hard disc - D:\Illumina\NextSeq Control Software\Temp\ It ought to be the only folder here as the NextSeq only retains fata from the most recent run - as soon as you start a new sequence run the data from the previous run is deleted. Copy the entire folder, including all its subdirectories. This folder contains the raw basecall (bcl) files. Do not change the name of the folder, which will be named as per the following convention - YYMMDD_InstrumentID_RunID_FlowcellID. The other requirement is a sample sheet - a simple comma separated file (csv) with the library chemistry, sample names and the index tag used for each sample, in addition to some other metrics describing the run. Copy the "SampleSheet.csv" into the top directory inside the sequence run folder on the server.

For each sample listed in a sample sheet, the software produces one FASTQ file for each read.

- When a sample sheet contains multiplexed samples, the software:
  - Places reads without a matching index adapter in the Undertermined_S0 FASTQ file.
  - Places reads with a valid index adapter sequences in the sample FASTQ file.
- When a sample sheet contains one unindexed sample, all reads are placed in the sample FASTQ files (one each for Read1 and Read2).
- When a sample sheet does not exist, or exists but has no Data section, all reads are placed in one FASTQ file named Undetermined_S0.
- When the Lane column in the Data section is not used, all lanes are converted. Otherwise only populated lanes are converted.

bcl2fastq2 Conversion

The Illumina bcl2fastq2 Conversion Software v2.20 demultiplexes sequencing data and converts base call(BCL) files into FASTQ files. For every cycle of a sequencing run, the Real-Time Analysis (RTA) software generates a BCL file containing base calls and associated quality scores (Q-scores). Most data analysis applications require FASTQ files as input.

The software uses input files, which are the output of a sequencing run, to convert BCL files into FASTQ files. For each cluster that passes filter (PF), the software writes one entry to one FASTQ file for each sample in each read.

- For a single-read run, the software creates one Read 1 FASTQ file per sample.

- For a paired-end run, the software creates one Read 1 and one Read 2 FASTQ file per sample.

The sample FASTQ files are compressed and appended with the *fastq.gz extension. Thus, per-cycle BCL files are converted into per-read FASTQ files that can be used as input for data analysis.

Demultiplexing Process Multiplexing adds a unique index adapter sequence to each sample during library prep, generating uniquely tagged libraries that can be identified and sorted for analysis. Demultiplexing then assigns clusters to a sample based on the index adapter sequence of the cluster.

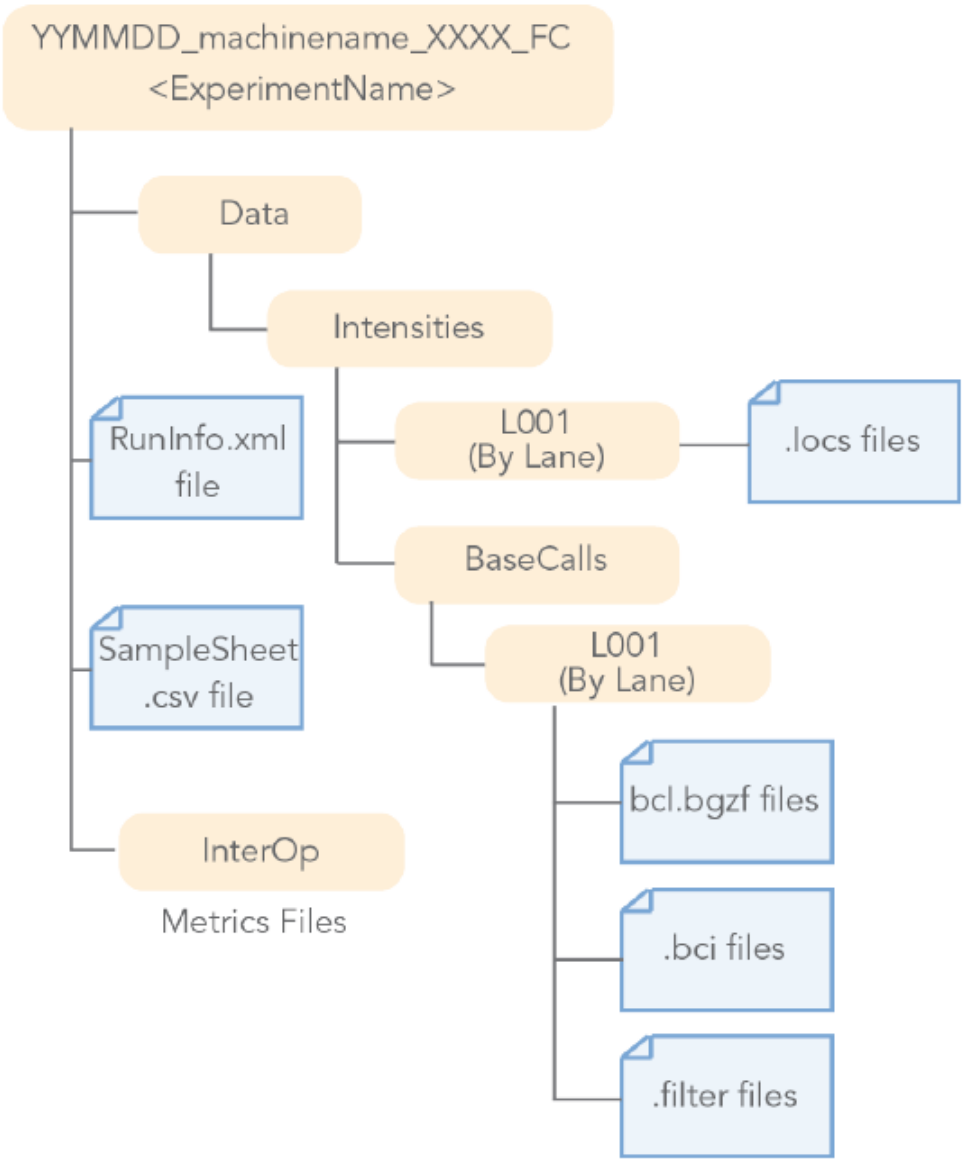
The bcl2fastq2 Conversion Software demultiplexes multiplexed samples as part of the conversion process. If samples are not multiplexed, the software skips demultiplexing and assigns all clusters in a flow cell lane to one sample.

Input Files

For each run, the control software generates an output folder to hold the BCL files and other sequencing data. The bcl2fastq2 Conversion Software uses this output as input. The output is recorded in various file formats. If a sample sheet is uploaded to the control software during run setup, it is included among the output.

All output files reside in the output folder. The output folder naming convention varies by system and can include the following variables separated by underscores:

- The six- or eight-digit date of the run in YYMMDD or YYYYMMDD format.
- The instrument or control computer ID consisting of any combination of alphanumeric characters and hyphens.
- A consecutively numberd experiment or run ID consisting of alphanumeric characters and hyphens.
- A consecutively numbered experiment or run ID consisting of at least one digit.
- The flow cell ID



Convert and Demultiplex BCL Files

1. Open a command-line window.
2. Type the folowing command and add options as needed.

```
nohup /usr/local/bin/bcl2fastq --run-folder-dir YYMMDD_InstrumentID_Ru
nID_FlowcellID \
-p 12 --output-dir YYMMDD_InstrumentID_RunID_FlowcellID/fastq_files --
no-lane-splitting

# nohup lets you start and close the terminal without canceling the jo
b
# -p 12 threads for processing (depends on server)
# -no-lane-splitting option, otherwise each output file from NextSeq i
s divided into four lanes.
```

FASTQ Files

BCL2FASTQ converts .bcl files into FASTQ files, which contain base call and quality information for all reads that pass filtering.

Naming Convention

FASTQ files are named with the sample name and the sample number, which is a numeric assignment based on the order that the sample is listed in the sample sheet. For example:

Data\Intensities\BaseCalls\SampleName_S1_L001_R1_001.fastq.gz

- SampleName—The sample name provided in the sample sheet. If a sample name is not provided, the file name includes the sample ID, which is a required field in the sample sheet and must be unique.
- S1—The sample number based on the order that samples are listed in the sample sheet starting with 1. In this example, S1 indicates that this sample is the first sample listed in the sample sheet.

NOTE

Reads that cannot be assigned to any sample are written to a FASTQ file for sample number 0, and excluded from downstream analysis.

- L001—The lane number.
- R1—The read. In this example, R1 means Read 1. For a paired-end run, there is at least one file with R2 in the file name for Read 2. When generated, index reads are I1 or I2.
- 001—The last segment is always 001.

File Compression FASTQ files are saved compressed in the GNU zip format (an open source file compression program), indicated by the .gz file extension.

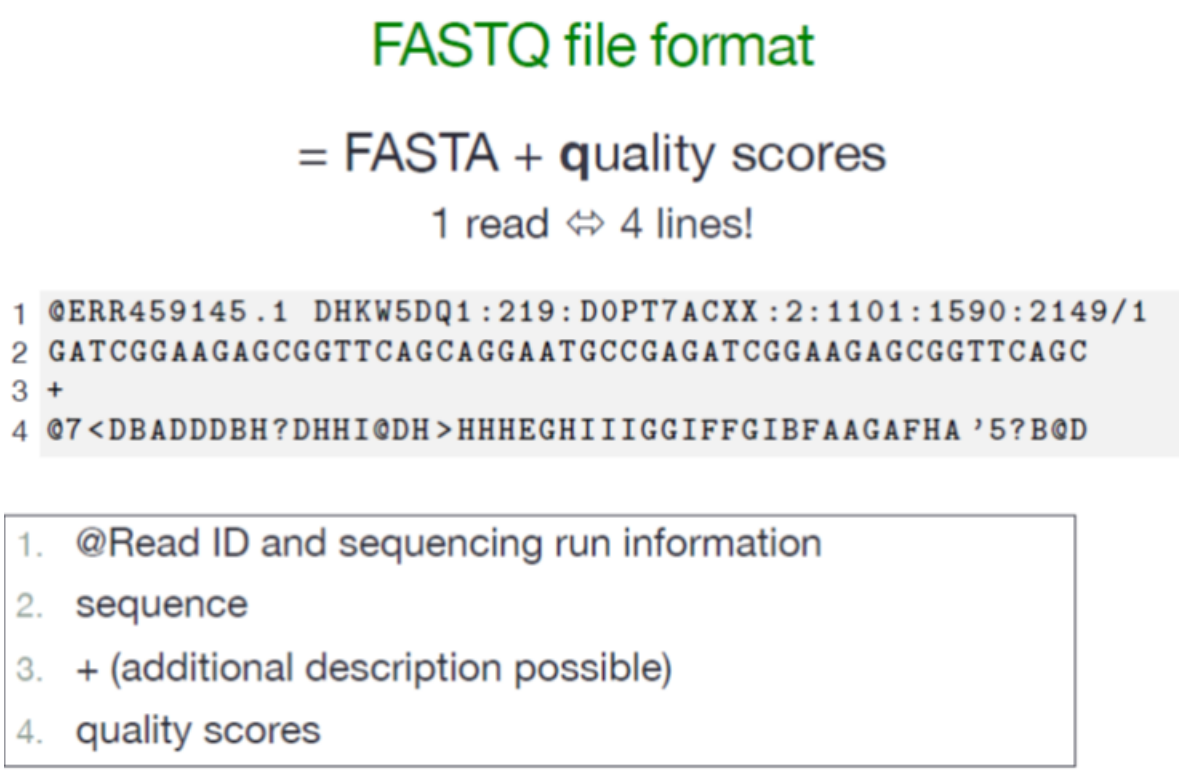
File Format

Each entry in a FASTQ file consists of 4 lines:

- Sequence identifier
- Sequence
- Quality score identifier line (consisting only of a +)
- Quality score The first line, identifying the sequence, contains the following elements.

```
@<instrument>:<run number>:<flowcell ID>:<lane>:<tile>:<x-pos>:<y-pos>:<UMI>
<read>:<is filtered>:<control number>:<index>
```

Element	Requirements	Description
@	@	Each sequence identifier line starts with @
<instrument>	Characters allowed: a-z, A-Z, 0-9 and underscore	Instrument ID
<run number>	Numerical	Run number on instrument.
<flowcell ID>	Characters allowed: a-z, A-Z, 0-9	
<lane>	Numerical	Lane number.
<tile>	Numerical	Tile number.
<x_pos>	Numerical	X coordinate of cluster.
<y_pos>	Numerical	Y coordinate of cluster.
<UMI>	Restricted characters: A/T/G/C/N	Optional, appears when UMI is specified in sample sheet. UMI sequences for Read 1 and Read 2, seperated by a plus [+].
<read>	Numerical	Read number. 1 can be single read or Read 2 of paired-end.
<is filtered>	Y or N	Y if the read is filtered (did not pass), N otherwise.
<control number>	Numerical	0 when none of the control bits are on, otherwise it is an even number.On HiSeq X and NextSeq systems, control specification is not performed and this number is always 0.
<index>	Restricted characters: A/T/G/C/N	Index of the read.



Control Values

If the read is not identified as a control, then the 10th column (`<control number>`) is zero. If the read is identified as a control, the number is greater than zero, and the value specifies what type of control it is. The value is the decimal representation of a bit-wise encoding scheme. In that scheme bit 0 has a decimal value of 1, bit 1 a value of 2, bit 2 a value of 4, and so on.

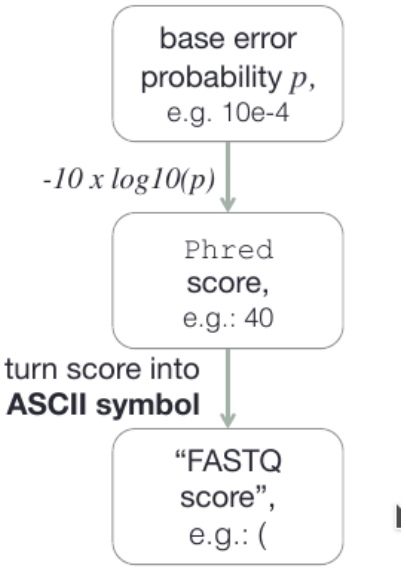
Quality Scores

Illumina sequencing is based on identifying the individual nucleotides by the fluorescence signal emitted upon their incorporation into the growing sequencing read. Once the sequencing run is complete, images taken during each DNA synthesis step are analyzed and the read clusters' fluorescence intensities are extracted and translated into the four letter code. The deduction of nucleotide sequences from the images aquired during sequencing is commonly referred to as base calling. Due to the imperfect nature of the sequencing process and limitations of the optical instruments, base calling will always have inherent uncertainty. This is the reason why FASTQ files store the DNA sequence of each read together with a position-specific quality score that represents the error probability, i.e., how likely it is that an individual base call may be incorrect. The score is called Phred score, Q, which is proportional to the probability p that a base call is incorrect, where $Q = -10 * \log10 (p)$. For example, a Phred score of 10 corresponds to one error in every ten base calls ($Q = -10 * \log10(0.1)$), or 90% accuracy; a Phred score of 20 corresponds to one error in every 100 base calls, or 99% accuracy. A higher Phred score thus reflects higher confidence in the reported base. To assign each base a unique score identifier (instead of numbers of varying character length), Phred scores are typically represented as ASCII characters. At http://ascii-code.com/ you can see which characters are assigned to what number.For raw reads, the range of scores will depend on the sequencing technology and the base caller used (Illumina, for example, used a tool called Bustard, or, more recently, RTA). Unfortunately, Illumina has been anything but consistent in how they a) calculated and b) ASCII-encoded the Phred score. In addition, Illumina now allows Phred scores for base calls with as high as 45, while 41 used to be the maximum score until the HiSeq X. This may cause issues with downstream applications that expect an upper limit of 41.

Base quality score

```
@ERR459145.1 DHKW5DQ1:219:D0PT7ACXX:2:1101:1590:2149/1
GATCGGAAGAGCGGTTCAGCAGGAATGCCGAGATCGGAAGAGCGGTTCAGC
+
@7<DBADDDBH?DHHI@DH>HHHEGHIIIGGIFFGIBFAAGAFHA'5?B@D
```

DEC	OCT	HEX	BIN	Symbol
32	040	20	00100000	
33	041	21	00100001	!
34	042	22	00100010	"
35	043	23	00100011	#
36	044	24	00100100	$
37	045	25	00100101	%
38	046	26	00100110	&
39	047	27	00100111	'
40	050	28	00101000	(
41	051	29	00101001	)
42	052	2A	00101010	*
43	053	2B	00101011	+



Quality score, Q(A)	Error probability, P(~A)
10	0.1
20	0.01
30	0.001

Quality Score Encoding

In FASTQ files, quality scores are encoded into a compact form, which uses only 1 byte per quality value. In this encoding, the quality score is represented as the character with an ASCII code equal to its value + 33. The following table demonstrates the relationship between the encoding character, its ASCII code, and the quality score represented.

Quality control of raw sequencing data

Since an analysis starts with the raw reads (stored in FASTQ files), the first step should be to check the overall quality of the sequenced reads.How successful was the actual sequencing? A poor RNAseq run will be characterized by the presence of:

Batch effects

Reasons

- variation in the sample processing (e.g., reagent batches, experimenters, pipetting accuracy)
- flowcell inconsistencies
- differences between sequencing runs (e.g., machine calibration)

Solutions

- appropriate experimental design (e.g. proper randomization)
- samples of the same experiment should have similar quality and quantity
- optimal experimental conditions (use of master mixes etc.)

Library preparation - PCR dependent biases

Reasons

- varying GC content can result in very distinct, library-specific fragment yields
- fragment size: small fragments are preferably hybridized to the flowcell
- low number of founder DNA-fragments will yield numerous duplicated fragments

Solutions

- optimizing cross-linking, sonication and the mRNA enrichment to ensure that the majority of the transcriptome is present in the sample
- limiting PCR cycles during library preparation to a minimum
- computational corrections for GC content and elimination of reads from identical DNA fragments

Sequencing erros and erorrs in base calling

Reasons

- loss of synchronized base incorporation into the single molecules within one cluster of clonally amplified DNA fragments (phasing and pre-phasing)
- mixed clusters
- signal intensity decay over time due to unstable reagents
- uneven signal intensities depending on the position on the flowcell
- overlapping emission frequency spectra of the four flourescently-labelled nucleotides

Solutions

- improvement of the sequencing chemistry and detection
- optimized software for base calling
- computational removal of bases with low base calling scores

Copy number variations and mappability

Reasons

- incomplete genome assemblies
- strain-specific differences from the reference assembly may lead to misrepresentation of individual loci
- repetitiveness of genomes and shortness of sequencing reads hinder uniqe read alignment

Solutions

- longer sequencing reads
- paired-end sequencing
- exclusion of blacklisted regions that are known to attract artificially high read numbers
- computational correction for mappability

Quality control of raw reads: FastQC

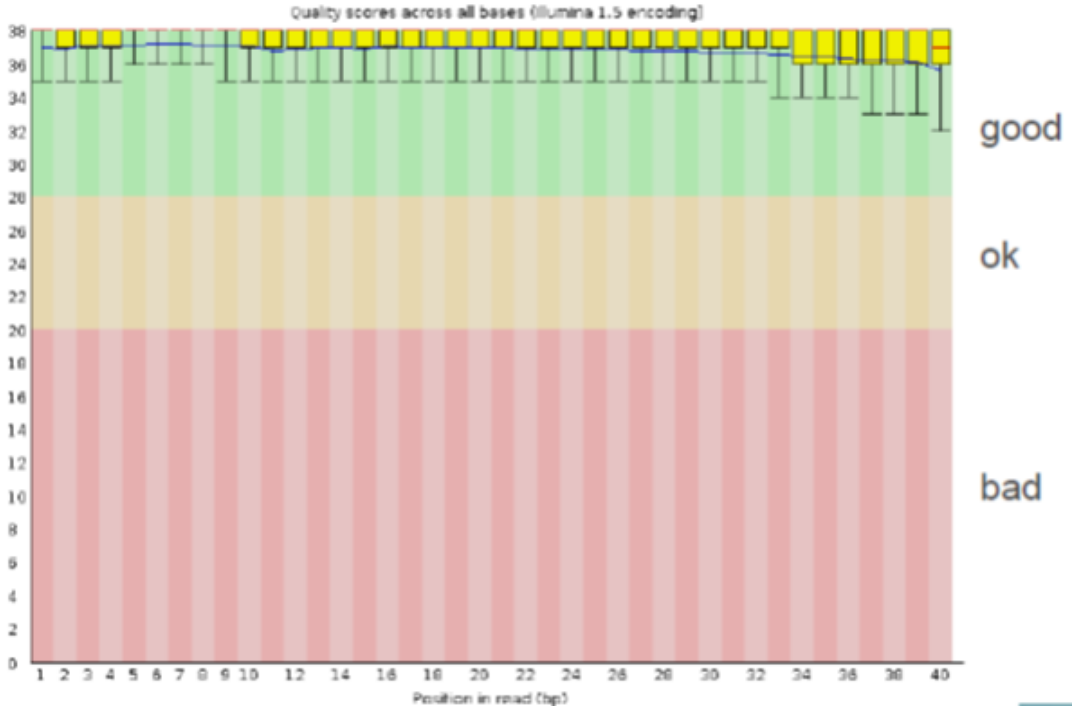
http://www.bioninformatics.babraham.ac.uk/projects/fastqc (http://www.bioninformatics.babraham.ac.uk/projects/fastqc)

FastQC aims to provide a simple way to do some quality checks on raw sequence data coming from high througput sequencing piplines. It provides a modular set of analysis which you can use to give a quick impression of whether your data has any problems of which you should be aware before doing any further analysis.

The main functions of FastQC are:

- Import of data from BAM, SAM or FastQ files (any variant)
- Providing a quick overview to tell you in which areas there may be problems
- summary graphs and tables to quickly assess your data

- export of results to an HTML based permanent report
- Offline operation to allow automated generation of reports without running the interactive application



FASTQC modules

- **Per *base* sequence quality**. This plot is based on the quality scores reported and stored by the sequencing platform (Phred scores). For Illumina sequencing, the reagents degrade throughout the sequening run which is why the last bases tend to get worse scores. *Warning*. Lower quartile for any base <10, or median for any base <25 .*Solution*. Trimming reads based on their average quality.
- **Per *tile* sequencing quality**. Check whether a part of the flowcell failed (e.g. due to bubbles or dirt on the flowcell). The plot shows the deviation from the average quality.*Warning* Any tile with a mean Phred score >2 for that base across all tiles. *Solution*. The FASTQ file contains the tile IDs for each read which can be used to filter out reads from flawed tiles. Note that variation in the Phred score of a flowcell is also a sign of overloading; tiles that are affected for only a few cycles can be ignored.
- **Per *sequence* quality scores**. Identify putative subsets of poor sequences. *Warning*. Most frequent mean quality <27. *Solution*. Quality trimming.
- **Per base sequence content**. Expectation: all bases should be sequenced as often as they are represented in the genome. Reasons why this assumption may not hold:
  - random hexamer priming during library preparation
  - contamination (e.g. adapter dimers)
  - bisulfite treatment (loss of cytosines) *Warning*. Difference between A and T, or G and C > 10% in any position. *Solution*. If overrepresented sequences are the cause of a failure, these can be removed.
- **Per sequence GC content**. The GC content of ech read shoudl be roughly normally distributed (maximum should correspond to the average GC content of the genome).*Warnings*. Deviations from the normal distribution for >15% of the reads.*Solution*. Sharp peaks on an otherwise smooth distribution usually indicate a specific contamination that should also be reported as on overrepresented sequence. Broader peaks outside the expectated distribution may represent contamination sequences from a species with a different GC genome content.
- **Per base N content**. Percentage of base calls at each position for which an N was called by the sequencer indicating lack of confidence to make a base call.*Warning*. Any position with an N content >5%. *Solution*. A common reason for large numbers of N is lack okf library complexity, i.e., if all reads start with the same bases, the sequencer may not be able to differentiate then sufficiently.
- **Sequence length distribution**. Determines the lengths of the sequences of the FASTQ file. The distribution of read sizes should be uniform for Illumina sequencing. *Warning*. Warning is raised if not all sequences are the same lenght; failure is issued if any sequence has zero lenght. *Solution*. For

Illumina sequencing, different read lengths should only occur if some sort of bioinformatic trimming has happend prior top FASTQC analysis.

- **Duplicate sequences**. The sequenced library should contain a random and complete representation of the genome or transcriptome, i.e., most sequences should occur only once. High duplication rates are indicative of PCR overamplification which may be caused by lack of starting material. Note that this module only takes the first 100000 sequences of each file into consideration. *Warning*. Non-unique sequences make up more than 20% of all reads. *Solution*. Unless you have reasons to expect sequences to be duplicated (i.e., specific enrichments of certain sequences), this plot is a strong indicator of suboptimal sample preparation. Duplication due to exessive sequencing will be reflefected by flattened lines in the plot; PCR overamplification of low complexity libraries are indicated by sharp peaks towards the right-hand side of the plot.
- **Overrepresented sequences**. All sequences which make up more than 0.1% of the first 100000 sequences are listed. *Warning*. Any sequence representing >0.1% of the total. *Solution*. Bioinformatic removal of contaminating sequences.
- **Adapter content**. This module specifically searches for a set of known adaptes. The plot iself shows a cumulative percentage count of the proportion of your library which has seen each of the adapter sequences at each position. *Warning*. Any adapter present in more than 5% of all reads. *Solution*. Bioinformatic removal of adapter sequences.
- **K-mer content**. The number of each 7-mer at each position is counted; then a binomial test is used to identify significant deviations from an even coverage at all positions (only 2% of the whole library is analyzed and the results are extrapolated). *Warning*. Any k-mer overrepresented with a binomial p-value <0.01. *Solution*. Libraries which derive from random priming will nearly always show k-mer bias at the start of the library due to an incomplete sampling of the possible random primers. Always check the results of the adatpter content and overrepresented sequences modules, too.

Script for Demultiplexing and FASTQC analysis

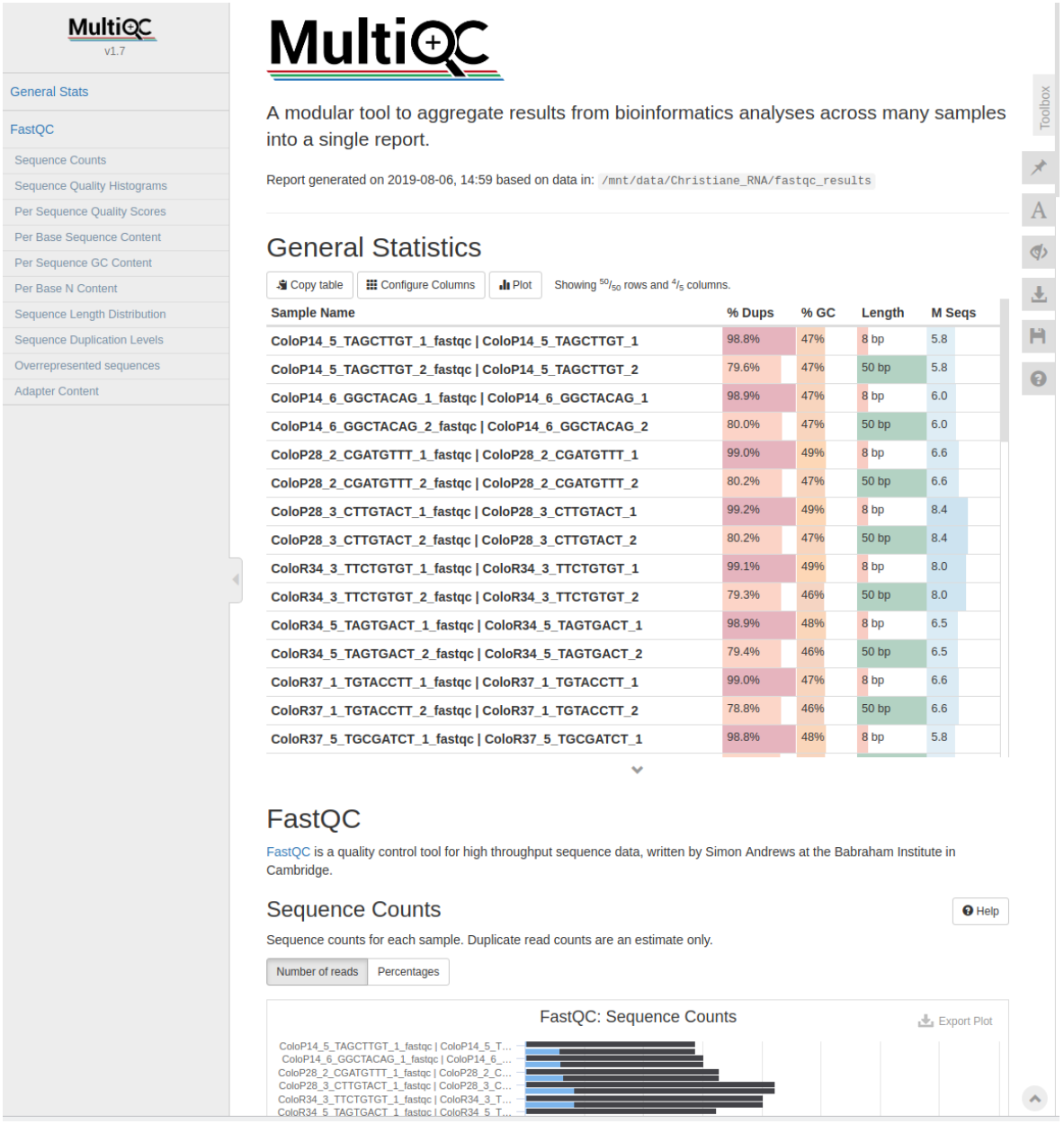
```
#!/bin/bash

echo "DemultiFastqcAndMultiQC Version 1.0"

# Check for file or '--help' argument:

if  [ -z "$1"  ]; then
    echo "Error: You must give 3 files as arguments." >&2
    echo "            Try 'show --help' to learn more." >&2
    exit 1
fi


# Functions

show-help() {

less << _EOF_

 DemultiFastqcAndMultiQC (Version 1.0)

 Help Page -- Press 'q' to exit.

 The Script when run in the folder where Read1 and Read2 fastq files a
re
 located first runs the program Je, which demultiplexes with the input
of
 a SampleBarcode.tsv in the format:
```

```
    ColoP14_5       TAGCTTGT
    ColoP14_6       GGCTACAG
    ColoR37_1       TGTACCTT
    ColoR37_5       TGCGATCT

 After that the FastQC is used to quality control checks on raw sequen
ce data.
 If there are FAIL OR WARN all the samples with the plots will be coll
ected in
 the fastqc_summary_biological_reads.pdf.

 To aggregate all the results the tool MultiQC is used, which creates
 an html
 report in the fastqc_results/QC/ folder.


 Commands:

    DemultiFastqcAndMultiQC.sh Read1.fastq Read2.fastq SampleBarcode.t
sv starts
    the script.

    DemultiFastqcAndMultiQC.sh --help = shows this help page.

_EOF_

}


# Display help page:

if [ "$1" == "--help" ]; then
    show-help
    exit
fi




exec 3>&1 4>&2
trap 'exec 2>&4 1>&3' 0 1 2 3
exec 1>Demulti_Fastqc_MultiQC.out 2>&1
# Every output/error from below will go to the file 'Demulti_Fastqc_Mu
ltiQC.out':

# run the script in the folder where the Read_1.fastq and Read_2.fastq
are located

# demultiplex using Je programm with barcode length 8 and Barcode fast
q files as Read_1

je demultiplex F1=R1.fastq F2=R2.fastq BF=Christiane_barcodes.tsv BPOS
=READ_1 BM=READ_1 LEN=8 ZT=4:0 O=./demultiplexed_fastq/

# run FastQC on all txt.gz (fastq files)
mkdir fastqc_results
for fastq in demultiplexed_fastq/*.gz
```

```
do
/home/sebastian/biotools/FastQC/fastqc /mnt/data/Christiane_RNA/${fast
q} --extract -o fastqc_results
done


# extract the sample names and barcodes out of the barcode file
awk '{print $1}' Christiane_barcodes.tsv  > samples.txt
awk '{print $2}' Christiane_barcodes.tsv  > barcode.txt

# show first 3 lines of files.txt
head -n3 samples.txt
head -n3 barcode.txt

# the following command extract all test results that did not pass (gr
ep -v PASS) and combines
# them with all images into a single PNG file

while read samples && read barcode <&3; do
        echo "$samples"; echo "$barcode"
        grep -v PASS ./fastqc_results/${samples}_${barcode}_2_fastqc/s
ummary.txt | \
        montage text:- ./fastqc_results/${samples}_${barcode}_2_fastq
c/Images/*png \
        -title x3 -geometry +0.1+0.1 -title ${samples} ${samples}.png
done <samples.txt 3<barcode.txt

# summarize all png files in pdf
convert *png fastqc_summary_biological_reads.pdf

#remove all png files
rm *.png

#change dir to run multiQC
cd fastqc_results/

rm *.zip

multiqc . --dirs -o QC/
```

FASTQC results

The results of the script above are a pdf report and a multiqc report.



DropSeq Pipeline

Preprocessing: tagging with barcodes & filtering using DropSeq

Convert the FASTQ files into an unaligned BAM file

The sequencer will gather data from both reads of the read pair. Read_1 is a *barcoded read*, containing the cell and molecular barcodes that will later identify this read as coming from particular transcript. Read_2 is the *biological read*, which contains a portion of the sequence of the sequence of the transcript observed. The raw reads form the sequencer must be converted into a Picard-queryname-sorted BAM file for each libray in the sequencer run. This has to be done because the FASTQ format can not hold the information about the cellular and molecular barcodes. BAM format has tag fields which can be used to hold this information.

```
# Create bam file without alignment
java -jar '-Djava.io.tmpdir='$tmpDir $PicardPath'picard.jar' FastqToSa
m F1=$ForwardRead  F2=$ReverseRead O=$RunName'.bam' SM=$RunName
# Sort bam file according to read ID
java -Xmx20g -jar '-Djava.io.tmpdir='$tmpDir $PicardPath'picard.jar' S
ortSam I=$RunName'.bam' O=$RunName'_sorted.bam' SORT_ORDER=queryname
```

Tag the reads in the BAM file with the cellular and molecular barcodes

The programm **TagBamWithReadSequenceExtended** transfers information from the barcoded read_1 over to the BAM record containing the genome read as a set of BAM_tags. The first 8 bases of the barcoded read contain the cell barcode, so we'll copy those bases over to a BAM tag (XC) on the genome read. Then we'll take the next 12 bases containing the molecular barcode and copy them over as another BAM tag (XM). Since we're now extracted all the information out of the barcoded read, we discard the read, converting the BAM to single-ended reads.

Additionally the program has a BASE_QUALITY option, which is the minimum base quality of all bases of the barcode being extracted. If more than NUM_BASES_BELOW_QUALITY bases falls below this quality, the read pair is discarded.

```
TagBamWithReadSequenceExtended
INPUT=my_unaligned_data.bam
OUTPUT=unaligned_tagged_Cell.bam
SUMMARY=unaligned_tagged_Cellular.bam_summary.txt
BASE_RANGE=1-8
BASE_QUALITY=10
BARCODED_READ=1
DISCARD_READ=False
TAG_NAME=XC
NUM_BASES_BELOW_QUALITY=1
```

```
TagBamWithReadSequenceExtended
INPUT=unaligned_tagged_Cell.bam
OUTPUT=unaligned_tagged_CellMolecular.bam
SUMMARY=unaligned_tagged_Molecular.bam_summary.txt
BASE_RANGE=9-20
BASE_QUALITY=10
BARCODED_READ=1
DISCARD_READ=True
TAG_NAME=XM
NUM_BASES_BELOW_QUALITY=1
```

Filter and trim the reads in the BAM file

In BAM format we can also do some trimming and filtering for the reads. The program **FilterBam** is used to remove reads where the cell or molecular barcode has low quality bases. During the run of **TagBamWithReadSequenceExtended**, and XQ tag was added to each read to represent the number of bases that have quality scores below the BASE_QUALITY threshold. These reads are then removed from the BAM.

```
FilterBam
TAG_REJECT=XQ
INPUT=unaligned_tagged_CellMolecular.bam
OUTPUT=unaligned_tagged_filtered.bam
```

The program **TrimStartingSequence** is one of two sequence cleanup programs designed to trim away any extra sequence that might have found a way into the reads. In this case, we trim the adapter that can occur 5' of the read. In our standard run, we look for at least 5 contiguous bases (NUM_BASES) of the adapter? (SEQUENCE) at the 5' end of the read with no errors (MISMATCHES), and hard clip those bases off the read.

```
TrimStartingSequence
INPUT=unaligned_tagged_filtered.bam
OUTPUT=unaligned_tagged_trimmed_smart.bam
OUTPUT_SUMMARY=adapter_trimming_report.txt
SEQUENCE=AAGCAGTGGTATCAACGCAGAGTGAATGGG
MISMATCHES=0
NUM_BASES=5
```

The **PolyATrimmer** program is the second sequence cleanup program designed to trim away trailing polA tails from reads. It searches for at least 6 (NUM_BASES) contigous A's in the read with 0 mismatches (MISMATCHES), and hard clips the read to remove these bases and all 3'bases of the polyA run.

```
PolyATrimmer
INPUT=unaligned_tagged_trimmed_smart.bam
OUTPUT=unaligned_mc_tagged_polyA_filtered.bam
OUTPUT_SUMMARY=polyA_trimming_report.txt
MISMATCHES=0
NUM_BASES=6
USE_NEW_TRIMMER=true
```

Convert the taggend and trimmed BAM file back into a FASTQ file for the alignment

Because the aligners take as input only FASTQ format we need to transform the trimmed and filtered BAM back to FASTQ format. To do this we extract the FASTQ files using Picard's **SamToFastq** program.

```
java -Xmx4g -jar /path/to/picard/picard.jar SamToFastq
INPUT=unaligned_mc_tagged_polyA_filtered.bam
FASTQ=unaligned_mc_tagged_polyA_filtered.fastq
```

DropSeq Script

This script was adjusted from the Rad script for our needs.

```
#!/bin/bash
exec 3>&1 4>&2
trap 'exec 2>&4 1>&3' 0 1 2 3
exec 1>log.out 2>&1
# Everything below will go to the file 'log.out':


## Run Script in Runfolder
RunId=$1
RunName=$2


MainPath=$(pwd)


pathname=$(pwd)
BarcodeInfo='BarcodeInfo.txt'
echo -e 'TAGCTTGT\nGGCTACAG\nTGTACCTT\nTGCGATCT\nTTGGTATG\nTCATTGAG\nT
TACTCGC\nTAGAACAC\nTACCACCA\nGAATCTGT\nGCTCCTTG\nGCTAACTC\nCGATGTTT\nC
TTGTACT\nTTCTGTGT\nTAGTGACT\nTGAACTGG\nTGGCTCAG\nTCGTTAGC\nTCATCCTA\nT
GCGTGAA\nGTACATCT\nGTAAGGTG\nGATTCATC' > BarcodeInfo.txt



#### Set paths to tools
DropSeqPipelinePath='/home/admin/RNASeq_test/packages/DropSeqPipelin
e/'
DropSeqPath='/home/admin/RNASeq_test/packages/Drop-seq_tools-1.12/'
PicardPath='/home/admin/RNASeq_test/packages/picard/'
StarExecutable='/home/admin/RNASeq_test/packages/STAR-2.6.0a/bin/Linux
_x86_64/STAR'
tmpPath='/home/admin/RNASeq_test/packages/'




RunDropSeq()
        {
        local ForwardRead=$1
        local ReverseRead=$2
        local RunName=$3
        local Organism=$4
        local tmpDir=$5
        echo $Organism
        echo $RunName
        if [ $Organism = "Human" ]; then
                ReferencePath='/home/admin/RNASeq_test/HG38/Homo_sapie
ns.GRCh38.dna.primary_assembly.fa'
                StarIndexPath='/home/admin/RNASeq_test/HG38/STARindex'
        elif [ $Organism = "Mouse" ]; then
                ReferencePath='/media/strauss/991ba020-c8ad-4483-860f-
7c5476093b78/DropSeqGenomes/GRCm38/GRCm38.fa'
                StarIndexPath='/media/strauss/991ba020-c8ad-4483-860f-
7c5476093b78/DropSeqGenomes/GRCm38/STAR_index'
        elif [ $Organism = "Pig" ]; then
                ReferencePath='/media/strauss/991ba020-c8ad-4483-860f-
7c5476093b78/DropSeqGenomes/Sscrofa11.1/Sscrofa11.1.fa'
                StarIndexPath='/media/strauss/991ba020-c8ad-4483-860f-
```

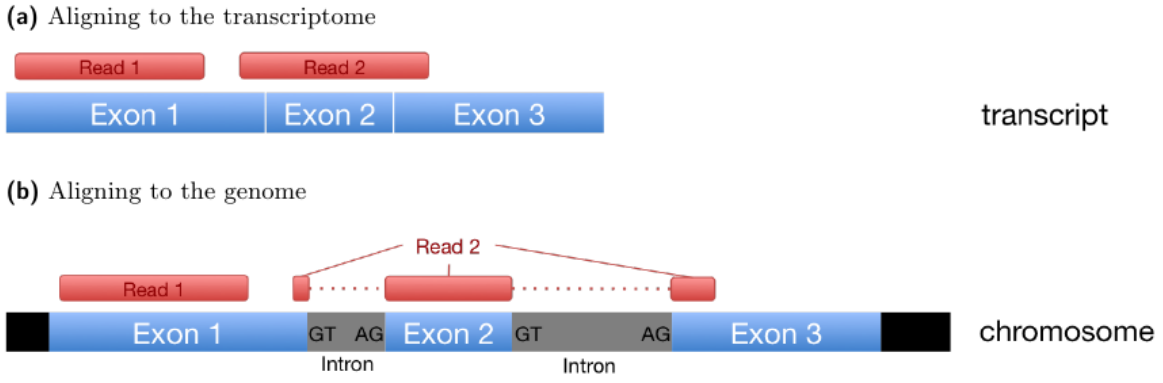
```
7c5476093b78/DropSeqGenomes/Sscrofa11.1/STAR_index'
        fi
        echo $ReferencePath
        echo $StarIndexPath
        mkdir $OutputFolder
        #### DropSeq Pipeline
        #### Create bam file without alignment
        java -jar '-Djava.io.tmpdir='$tmpDir $PicardPath'picard.jar' F
astqToSam F1=$ForwardRead  F2=$ReverseRead O=$RunName'.bam' SM=$RunNam
e
        ### Sort bam file according to read ID
        java -Xmx20g -jar '-Djava.io.tmpdir='$tmpDir $PicardPath'picar
d.jar' SortSam I=$RunName'.bam' O=$RunName'_sorted.bam' SORT_ORDER=que
ryname
        ### Start Dropseq tool
        $DropSeqPath'Drop-seq_alignment.sh' -g $StarIndexPath -r $Refe
rencePath -d $DropSeqPath -s $StarExecutable -o $OutputFolder -t $tmpD
ir $RunName'_sorted.bam'
        mv $pathname'/'$BarcodeInfo $OutputFolder
        cd $OutputFolder
        $DropSeqPath'DigitalExpression' -m 30g -t $tmpDir I=star_gene_
exon_tagged.bam O=DGE_Matrix.txt CELL_BARCODE_TAG=XC MOLECULAR_BARCODE
_TAG=XM CELL_BC_FILE=$BarcodeInfo
        mv $tmpDir'unaligned_tagged_CellMolecular.bam' $OutputFolder
        #sh $DropSeqPipelinePath'CreateSummaryTables.sh'
        samtools view star_gene_exon_tagged.bam | egrep 'INTERGENIC|IN
TRONIC|UTR|CODING' | cut -f12,13 > SummaryTable.txt
        sed -i 's/GE:Z.\+/XF:Z:EXONIC/' SummaryTable.txt
        sort -T $tmpDir -nk 1 SummaryTable.txt | uniq -c > Summary.txt
        samtools view unaligned_tagged_CellMolecular.bam | cut -f12 |
sort -T $tmpDir | uniq -c > UnfilteredReadCount.txt
        Rscript $DropSeqPipelinePath'CreateReadStatistics.R' $BarcodeI
nfo
        }


tmpDir=$tmpPath$RunName'_tmp/'
echo 'Creating temporary directory '$tmpDir
mkdir $tmpDir
Organism='Human'
echo 'Organism is set to '$Organism
ForwardRead=$(ls -1 | grep 'R1')
ReverseRead=$(ls -1 | grep 'R2')
echo $ForwardRead
echo $ReverseRead
OutputFolder=$MainPath'/'$RunName'_Output'

RunDropSeq $ForwardRead $ReverseRead $RunName $Organism $tmpDir
```

Read Alignment

In order to identify the transcript that are present in a specific sample, the genomic origin of the sequenced cDNA fragments must be determined. The assignment of sequencing reads to the most likely locus of origin is called *read alignment* or *mapping* and it is a crucial step in most types of high-throughput sequencing experiments.

The general challenge of short read alignment is to map millions of reads accurately and in a reasonable time, despite the presence of sequencing errors, genomic variation and repetitive elements. The different alignment programs employ various strategies that are meant to speed up the process (e.g., by indexing the reference genome) and find a balance between mapping fidelity and error tolerance.

The main challenge of RNA-seq data in particular is the spliced alignment of exon-exon-spanning reads and the presence of multiple different transcripts (isoforms) of the same gene. Some alignment programs tried to mitigate this problem by aligning to the transcriptome, but this approach is limited to known transcripts and thus heavily dependent on the annotation. Moreover, many reads will overlap with more than one isoform, introducing mapping ambiguity. Thus the most popular RNA-seq alignment programs (e.g., STAR, TopHat, GSNAP) use existing gene annotation for the placement of spliced reads in addition to attempting to identify novel splice events based on reads that cannot be aligned to the reference genome (or transcriptome). The identification of novel splice junctions is based on certain assumptions about transcript structures that may or may not be correct. Additionally, lowly expressed isoforms may have very few reads that span their specific splice junctions while, conversely, splice junctions that are supported by few reads are more likely to be false positives. Therefore, novel splice junctions will show a bias towards strongly expressed genes. Until reads routinely are sequenced longer, the alignment of spliced reads will therefore remain the most prevalent problem of RNA-seq data.



Reference genomes and annotation

Genome sequences and annotation are often generated by consortia such as ENCODE, The Mouse Genome Project, The Berkeley Drosophila Genome Project, and many more. The results of these efforts can either be downloaded from individual websites set up by the respective consortia or from more comprehensive data bases such as the one hosted by the UCSC (University of California https://genome.ucsc.edu/) or the European genome resource, Ensembl (http://www.emsembl.org). UCSC and Emsembl try to organize, unify, and provide access to a wide range of genomes and annotation data. The advantage of downloading data from UCSC or Ensembl is that even if you were to work with different species, the file formats and naming convetions will be consistent. Note that UCSC and Ensembl use slightly different naming conventions that can seriously affect downstrem analyses. Try to stick to one source. Reference sequences are usually stored in plain text **FASTA** files that can either be compressed with generic **gzip** command or, using the tool **faToTwoBit**, into **.2bit** format.

File formats for defining genomic regions

While the reference sequence is not much than a very long string of A/T/C/G/N, various file formats exist to store information about the location of transcription start site, exons, introns etc. All formats agree on having one line per genomic feature, but the nature of the information contained in each row can vary strongly between the formats.

GFF The General Feature Format has nine required fields; the first three fields form the basic **name, start, end** tuple that allows for the identification of the location in respect to the reference genome. Fields must be seperated by a single TAB, but no white space. All but the final field in each feature line must contain a value; missing values should be denoted with a '.'

There are two versions of the **GFF** format in use which are similar, but not compatible:

1. GFF version 2 (Sanger Institute; https://www.sanger.ac.uk (https://www.sanger.ac.uk))
2. GFF version 3 (Sequence Ontology Project; http://gmod.org/wiki/GFF3 (http://gmod.org/wiki/GFF3))

GFF2 files use the following fields:

1. **reference sequence**: coordinate system of the annotation (e.g., "chr1")
2. **source**: describes how the annotation was derived (e.g., name of annotation software)
3. **method**: annotation type (e.g., gene)
4. **start position**: 1-based integer, always less than or equal to the stop position
5. **stop position**: for zero-length features, such as insertion sites, start equals end and the implied site is to the right of the indicated base
6. **score**: e.g., sequence indentity
7. **strand**: "+" for the forward strand, "-" for the reverse strand, or "." for annotations that are not stranded
8. **phase**: codon phase for annotations linked to proteins; 0,1, or 2, indication the frame, or the number of bases that should be removed from the beginning of this feature to reach the first base of the next codon
9. **group**: contains the class and ID of an annotation which is the logical parant of the current one ("feature is composed of")

GFF3 files

1. **reference sequence**
2. **source**
3. **type**: contrained to be either: a term from the "lite" sequence ontology, SOFA
4. **start position**
5. **stop position**
6. **score**
7. **strand**
8. **phase**
9. **attributes**: list of feature attributes as TAG=VALUE pairs; spaces are allowed in the field, multiple TAG=VALUE pairs are separated by semicolon; the TAGS have predefiend meanings:

    - ID (must be unique(=)
    - Name (display name)
    - Alias (secondary name)
    - Parent
    - Target (the format okf the value is "target_id start end [strand]")
    - Gap (in CIGAR format)
    - Derives_from (database cross reference)
    - Ontology_term

GTF The Gene Transfer Format is based on the GFF, but is defined more strictly. The first eight GTF fields are the same as GFF2, but, as for GFF3, the 9th field has been expanded into a list of attributes. Contrary to GFF files, the TYPE VALUE pairs of GTF files are separated by one space and must end with a semi-colon (followed by exactly one space if another attribute is added afterwards).

Storing annotation information

- representing genome coordinates + description/name
- various formats (all are plain text files): GFF2, GFF3, GTF, BED, SAF…

see the course notes for details

GTF ("GFF2.5")

1. reference coordinate
2. source
3. annotation type
4. start position
5. end position
6. score
7. strand
8. frame/phase
9. attributes: <TYPE VALUE>

```
# GFF-version 2
IV      curated exon    5506900 5506996 . + .   Transcript B0273.1
IV      curated exon    5506026 5506382 . + .   Transcript B0273.1
IV      curated exon    5506558 5506660 . + .   Transcript B0273.1
IV      curated exon    5506738 5506852 . + .   Transcript B0273.1

# GFF-version 3
ctg123  .  exon  1300  1500  .  +  .  ID=exon00001
ctg123  .  exon  1050  1500  .  +  .  ID=exon00002
ctg123  .  exon  3000  3902  .  +  .  ID=exon00003
ctg123  .  exon  5000  5500  .  +  .  ID=exon00004
ctg123  .  exon  7000  9000  .  +  .  ID=exon00005
```

```
# example for the 9th field of a GTF file
    gene_id "Em:U62.C22.6"; transcript_id "Em:U62.C22.6.mRNA"; exon_number 1
```

BED format The BED format is the simplest way to store annotation tracks. It has three required fields (chromosome, start, end) and up to 9 optional fields (name, score, strand, thickStart, thickEnd, itemRgb, blockCount, blockSizes, blockStarts). The number of fields per line ca thus vary from three to twelve, but must be consistent within a file and must obey the order, lower-numbered fields must always be populated if higher-numbered fields are used. Fields seven to twelve are only neccessary if region should be drawn in a Genome Browser with the typical appearance known for gene tracks.

```
# 6 - column BED file defining transcript loci

 chr1 66999824 67210768 NM_032291 0 +
 chr1 33546713 33586132 NM_052998 0 +
 chr1 25071759 25170815 NM_013943 0 +
 chr1 48998526 50489626 NM_032785 0 -
```

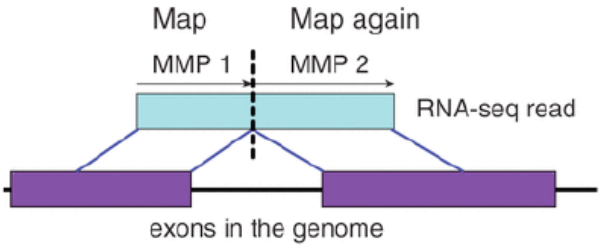
!Note Obtaining a correctly formatted GTF file may be one of the most difficult tasks in the entire analysis! Do take this seriously and invest the time to make sure that the GTF file you are using is correctly formatted.

Aligning reads using STAR

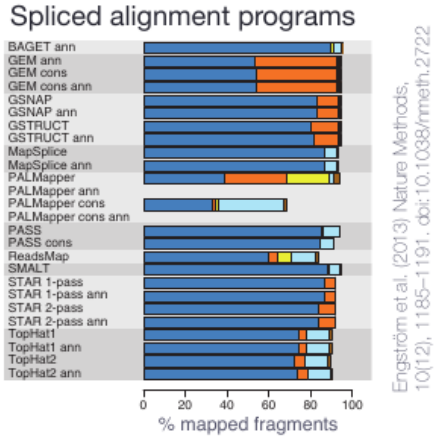
Numerous alignment programs have been published in the past (and will published in the future), and depending on your specific project, some aligners may be preferred over others. For example, detection of structural variants and fusion transcripts will require very specific settings or a dedicated alignment tool for that particular task.

For straight-forward RNAseq data that will be used for differential gene expression analysis, STAR has been shown to be very efficient and reasonably sensitive. Another popular aligner is TopHat, which is basically a sophisticated wrapper around the genomic aligner Bowtie.

Spliced Transcriptome Alignment to Reference (STAR)



- accurate & sensitive
- very fast
- memory intensive! (use it on the server!)



- MMP = maximal mappable prefix (aka maximum matching portion)
- reads are split when a continuous alignment is not possible
- the remaining unmappable portion is then aligned again
- finally, aligned portions of the original full-length reads are stitched together

2 main STAR modules

1. generate **genome index**

```
--runMode genomeGenerate
--genomeFastaFiles sacCer3.fa
--sjdbGTFfile sacCer3.gtf
```

needs to be done just 1x per transcriptome!

2. **align**

2.1. align to *reference* & identify novel splice junctions

2.2 *re-run* alignment including the novel splice junctions

```
$runSTAR -genomeDir STARindex/ \
         --readFilesIn $FASTQ_FILES \
         --readFilesCommand zcat \
```

```
--twopassMode
```

must be done for every sample

1. **Generate genome index** This step has to be done only once per genome type (and alignment program). The index files will comprise the genome sequence, suffix arrays (i.e., tables of k-mers), chromosome names and lengths, splice junctions coordinates, and information about the genes (e.g. the strand). Therefore, the main input for this step encompases the reference genome and an annotation file.

Create a STAR index

```
#!/bin/bash
exec 3>&1 4>&2
trap 'exec 2>&4 1>&3' 0 1 2 3
exec 1>log.out 2>&1
# Everything below will go to the file 'log.out':


#This step has to be done only once per genome.
#The index files will comprise the genome sequence, suffix arrays(tabl
es of k-mers),
#chromosom names and lengths, splice junctions coordinates, and inform
ation
#about the genes (strand).
#the main input for this step encompasses the reference genome and an
 annotation file.

#create a dictionary to store the index in

REF_DIR=/home/admin/RNASeq/HG38
mkdir ${REF_DIR}/STARindex

#set a variable for STAR access
runSTAR=/home/admin/RNASeq/packages/STAR-2.6.0a/bin/Linux_x86_64/STAR


#run STAR in "genomeGenerate" mode
${runSTAR} --runMode genomeGenerate \
        --genomeDir ${REF_DIR}/STARindex \ #index will be stored there
    --genomeFastaFiles ${REF_DIR}/*.fa \ #reference genome sequence
        --sjdbGTFfile ${REF_DIR}/*.gtf \ #annotation file
    --sjdbOverhang 71 #should be read length minus 1; length of the ge
nomic
    #sequence around the annotated junction to be used for the splice
    #junctions database
    --runThreadN 6 #can be used to define more processors
    --genomeSAsparseD 2
    --limitIObufferSize 80000000
```

Create Metadata for the DropSeq Pipeline

If you are using the DropSeq Pipeline from raw unaligned reads to an aligned BAM, it's necessary to have a number of different metadata files. These provide information about the organism you are running your experiment on, as well as genomic features like genes, transcripts, and exons that help extract DGE data from the reads.

CreateSequenceDictionary

The first file needed is the sequence dictionary. This is a list of the contigs in the fastq file and their lengths.

```
java -jar /path/to/picard/picard.jar CreateSequenceDictionary
REFERENCE=my.fasta
OUTPUT= my.dict
SPECIES=species_name
```

ConvertToRefFlat

The next file is the refFlat file, which is generated using the sequence dictionary generated above.

```
ConvertToRefFlat
ANNOTATIONS_FILE=my.gtf
SEQUENCE_DICTIONARY=my.dict
OUTPUT=my.refFlat
```

ReduceGTF

The may be useful if you need an easy to parse version of your annotations in a language like R, and is also used to generate the other metadata.

```
ReduceGTF
SEQUENCE_DICTIONARY=my.dict
GTF=my.gtf
OUTPUT=my.reduced.gtf
```

CreateIntervalsFiles

As a last step, we create interval files needed for various programs in the Drop-seq pipeline. This program generates a number of interval files for genes, exons, consensus introns, rRNA, and mt. The example below uses the human MT contig name, but if you use a different organism you should set that argument appropriately.

CreateIntervalsFiles

```
SEQUENCE_DICTIONARY=my.dict
REDUCED_GTF=my.reduced.gtf
PREFIX=my
OUTPUT=/path/to/output/files
MT_SEQUENCE=MT
```

Alignment DropSeq Pipeline

The DropSeq Folder contains the programs described above and also contains a script Drop-seq_alignment.sh that executes most of the processes described above and below.

Alignment-STAR

```
/path/to/STAR/STAR
--genomeDir /path/to/STAR_REFERENCE
--readFilesIn unaligned_mc_tagged_polyA_filtered.fastq
--outFileNamePrefix star
```

After STAR is done aligning reads, we know where the genome reads align, but we've lost track of what cell and molecular barcodes these reads have. This information is recovered by merging the BAM tags from the unaligned BAM to the aligned reads from STAR.

Before merging the SortSam picard program is invoked after alignment, to guarantee that the output from alignment is sorted in queryname order. As a side bonus, the output file is a BAM instead of SAM.

SortSam

This picard program is invoked after alignment, to guarantee that the output from alignment is sorted in queryname order. The output is also a BAM (compressed) instead of SAM (uncompressed).

```
java -Xmx4g -jar /path/to/picard/picard.jar SortSam
I=starAligned.out.sam
O=aligned.sorted.bam
SO=queryname
```

MergeBamAlignment

This Picard program merges the sorted alignment output from STAR (ALIGNED_BAM) with the unaligned BAM that had been previously tagged with molecular/cell barcodes (UNMAPPED_BAM). This recovers the BAM tags that were "lost" during alignment.

```
java -Xmx4g -jar /path/to/picard/picard.jar MergeBamAlignment
REFERENCE_SEQUENCE=my_fasta.fasta
UNMAPPED_BAM=unaligned_mc_tagged_polyA_filtered.bam
James Nemesh, McCarroll Lab
Drop-seq core computational protocol
V2.0.0; September 28 2018; page 9
ALIGNED_BAM=aligned.sorted.bam
OUTPUT=merged.bam
INCLUDE_SECONDARY_ALIGNMENTS=false
PAIRED_RUN=false
```

Alignment not using DropSeq

If you dont use DropSeq or you want to do bulk RNASeq alignment without using the UMIs, then the CLI script looks like this:

```
# make a folder to store the STAR output in
mkdir alignment_STAR

# list fastq.gz files separated by comma without whitespaces
FILES= `ls -m rawReads/*fastq.gz | sed 's/ //g'`
FILES= `echo $Files | sed 's/ //g'`

# execute STAR in the runMode "alignReads"
${runSTAR} --genomeDir ${REF_DIR}/STARindex/ \
--readFilesIn $FILES \
--readFilesCommand ycat \ # necessary because of gzipped fastq files
--outFileNamePrefix alignment_STAR/WT_1_ \
--outFilterMultimapNmax 1 \ # only reads with 1 match in the reference
will be returned as aligned
--outReadsUnmapped Fastx \ # will generate an extra output file with t
he unaligned reads
--outSAMtype BAM SortedByCoordinate \
--twopassMode Basic \ # STAR will perform mapping, then extract novel
 junctions which will be inserted into the genome index which will the
n be used to re-map all reads
--runThreadN 1 # can be increased if sufficient computational power is
available
```

!Note The default settings or the settings shown may not be optimal for your application (or even for this application)! Please, read the STAR manual (Dobin and Gingeras 2016) and decide which parameters are suitable for your data set!

Bam file format for aligned reads

- SAM(Sequence Alignment/Map) is a tab-delimited text file containing aligned reads. BAM is a binary compact form of SAM.
- BAM index file (.bai)
  - BAM files can be sorted by chromosomal coordinates and indexed for efficient retrieval of reads of reads for a given region
  - the index file must have a matching name
  - genome browser requires both BAM and the index file

BAM file indexing

Most downstream applictations will require a .BAM.BAI file together with every BAM file to quickly access the BAM files without having to load then into memory. To obtain these index files, simply run the **samtoools index** command for each BAM file once the mapping is finished.

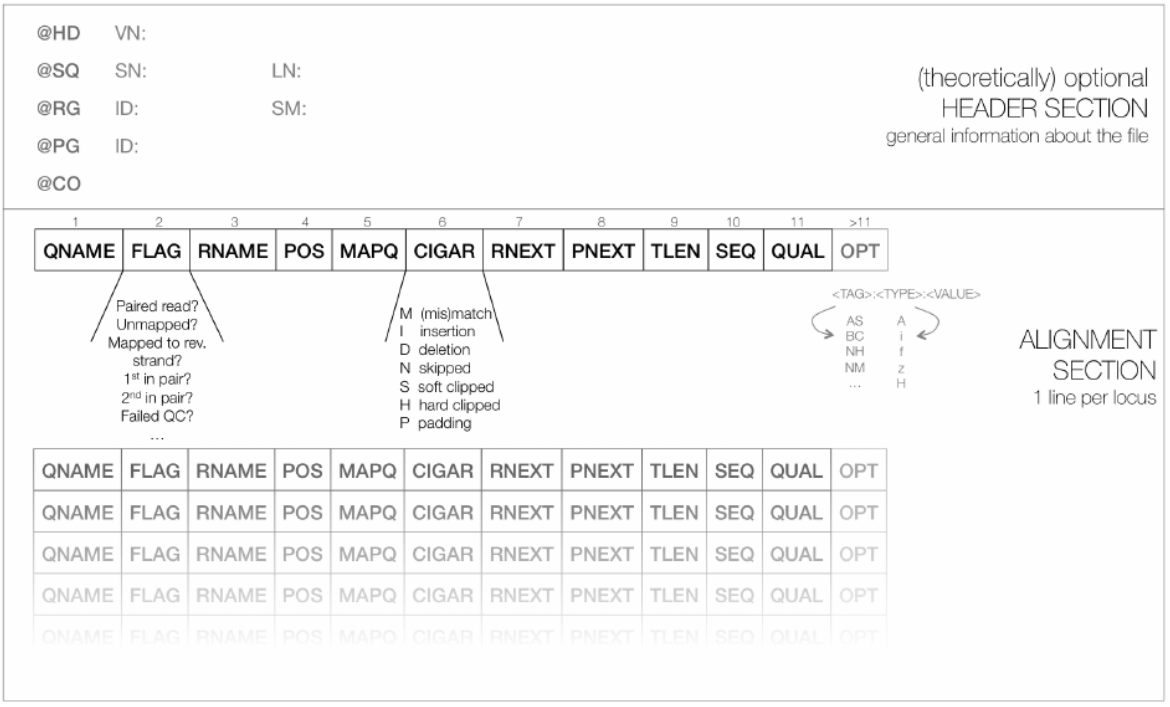
```
# export samtools path (for convenience)
export PATH=/home/classadmin/software/samtools-1.7:$PATH

# index the BAM file
samtools index alignment_STAR/WT_1_Aligned.sortedByCoord.out.bam
```

Storing aligned reads: SAM/BAM file format

The output option of STAR already indicates that the results of the alignment will be stored in a SAM or BAM file. The Sequence Alignment/Map (SAM) format is, in fact, a generic nucleotide alignment format that describes the alignment of sequencing reads (or query sequences) to a reference. The human readable, TAB-delimited SAM files can be compressed into the Binary Alignment/Map format. These BAM files are bigger than simply gzipped SAM files, because they have been optimized for fast random access rather then size redution. Postion-sorted BAM files can be indexed so that all reads aligning to a locus can be efficiently retrieved without loading the entire file into memory. To convert a BAM file into a SAM file, use samtools view:

```
export our local installation of samtools into your PATH
export PATH=~/mat/software/samtools-1.7/:$PATH
samtools view -h WT_1_Aligned.sortedByCoord.out.bam > WT_1_Aligned.sor
tedByCoord.out.sam
```

SAM files typically contain a short header section and very long alignment section where each row represents a single read alignment.



The SAM file header section

The header section includes information about how the alignment was generated and stored. All lines in the header section are tab-delimited and begin with the @ character, followed by **tag:value** pairs, where tag is a two-letter string that defines the content and the format of **value**. For example, the "@SQ" line in the header section contains the information about the names and lengths of the *reference sequences* to which the reads were aligned. For a hypothetical organsim with three chromosomes of lenght 1000 bp, the SAM header should contain:
@SQ SN:chr1 LN:1000
@SQ SN:chr2 LN:1000
@SQ SN:chr3 LN:1000

```
# The default behavior of samtools view is to not show the header sect
ion.
# samtools view -h will show both header an alignement section;
# samtools vieh -H will return the header section only

samtools view -H Sample1_Aligned.sortedByCoord.out.bam
@HD VN :1.4

 @SQ SN : chrI LN :230218
 @SQ SN : chrII LN :813184
 @SQ SN : chrIII LN :316620
 @SQ SN : chrIV LN :1531933
 @SQ SN : chrV LN :576874

 @PG ID : STAR VN : STAR_2 .4.0 e CL : STAR -- runThreadN 8 -- genomeD
ir STAR - sacCer3
-- readFilesIn Lane1 . fastq . gz , Lane2 . fastq . gz , Lane3 . fastq
. gz , Lane4 . fastq . gz ,
Lane5 . fastq . gz , Lane6 . fastq . gz , Lane7 . fastq . gz -- readFi
lesCommand zcat --
outFileNamePrefix Sample1_ -- outSAMtype BAM SortedByCoordinate --
outSAMunmapped Within
 -- outFil terMu ltimap Nmax 1

 @CO user command line : STAR -- genomeDir STAR - sacCer3 -- readFiles
In Lane1 .
fastq . gz , Lane2 . fastq . gz , Lane3 . fastq . gz , Lane4 . fastq .
gz , Lane5 . fastq . gz , Lane6
. fastq . gz , Lane7 . fastq . gz -- readFilesCommand zcat -- outFileN
amePrefix
Sample1_ -- ou tFilt erMult imapNm ax 1 -- outSAMunmapped Within -- ru
nThreadN 8
-- outSAMtype BAM SortedByCoordinate
```

The SAM file alignment section

The optional header section is followed by the alignment section where each line corresponds to one sequenced read. For each read, there are 11 mandatory fields that always appear in the same order:

```
<QNAME><FLAG><RNAME><POS><MAPQ><CIGAR><MRNM><MPOS><ISIZE><SEQ><QUAL>
```

If the corresponding information is unavailable or irrelevant, field values can be '0' or '* ', depending on the field, but they cannot be missing. After the 11 mandatory fields, a variable number of optional fields can be present.

Pos.	Field	Example entry	Description	NA value
1	QNAME	Read1	Query template(=read) name (PE:read pair name)	required
2	FLAG	83	Information about the read's mapping properties encoded as bit-wise flags	required
3	RNAME	chrI	Reference sequence name. This should match a @SQ line in the header.	*
4	POS	15364	1-based leftmost mapping position of the first matiching base. Set as 0 for an unmapped read without coordinates.	0
5	MAPQ	30	Mapping quality of the alignment. Should be Phred-scaled posterior probability that the position of the read is incorrect.	0

Pos.	Field	Example entry	Description	NA value
6	CIGAR	51M		
7	RNEXT	=	PE reads: reference sequence name of the next read. Set to "=" is both mates are mapped to the same chromosome.	*
8	PNEXT	15535	PE reads: leftmost mapping position of the next read.	0
9	TLEN	232	PE reads	inferred template length (fragment size).
10	SEQ	CCA...GGC	The sequence of the aligned read on the forward strand (not including indels).	*
11	QUAL	BBH...1+B	Base quality string in the FASTQ format	*
12ff	OPT	NM:i:0	Optional fields	

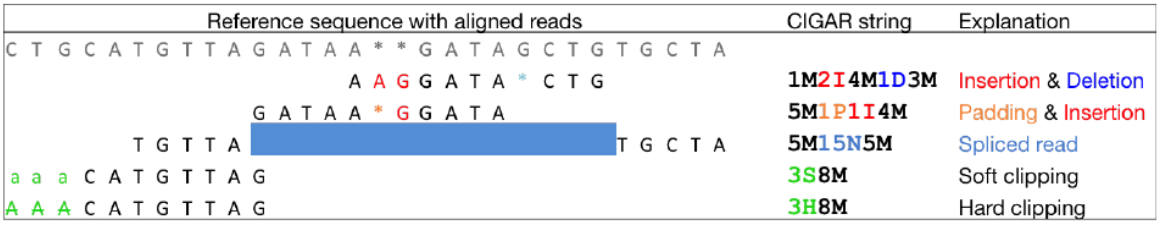
CIGAR (Concise Idiosyncratic Gapped Alignment Report) String

The sixth field of a SAM file contains a so-called **CIGAR** string indication which operations were to map the read to the reference sequence at that particular locus.

The following operations are defined in **CIGAR** format:

- M Alignment (can be a sequence match or mismatch!)
- I Insertion in the read compared to the reference
- D Delettion in the read compared to the reference
- N Skipped region from the reference. For mRNA-to-genome alignments, an N operation represents an intron otherwise N i not defined
- S Soft clipping
- H Hard clipping
- P Padding (silent deletion from padded reference)
- = Sequence match (not widely used)
- X Sequence mismatch (not widely used)

The sum of lengths of the M, I, S, =, X operations must equal the lenght of the read.



Manipulating SAM/BAM files

samtools is a powerful suite of tools designed to interact with **SAM** and **BAM** files.

```
# return a peek into a SAM or BAM file (note that a SAM file can also
 easily be inspected using the basic UNIX commands for any text file,
 such as cat, head, less etc.)
samtools view InFile.bam | head

# turn a BAM file into the human-readable SAM format (including the he
ader)
samtools view -h InFile.bam > InFile.sam

# compress a SAM file into a BAM format (-Sb is equivalent to -S -b)
samtools view -Sb InFile.bam > OutFile.bam

# generate an index for a BAM file (needed for many downstream tools)
samtools index InFile.bam

# to see all operations that can be done using samtools, type samtools
-help
```

The **samtools view** tool has many options that directly interpret some of the mandatory fields of its alignment section, such as the mapping quality, the location and the FLAG field values.

```
# get only unmapped reads
samtools view -h \ # show header
    -b \ # output a BAM file
    -f 4\ # include only reads where 0x4 bit is set
    Aligned.sortedByCoord.out.bam > unmapped_reads.bam

# get only mapped reads
samtools view -hb -F 4 \ # include only reads where the 0x4 bit is NOT
set
    Aligned.sortedByCoord.out.bam > mapped_reads.bam

# skip read alignments with mapping quality below 20
samtools view -h -b -q 20 Aligned.sortedByCoord.out.bam > high_mapq_re
ads.bam
```

If you would like to filter an alignment based on any optional tags, you will have to resort to means outside samtools. Looking for exact matches using grep can be particularly helpful here, but you should make sure that you make the regular expression as stringent as possible.

Here is an example for retrieving reads with only one alignment (aka uniquely aligned reads).

```
# STAR uses the NH:i tag to record the number of alignments found for
 a read NH:1 => 1 # alignemt; NH:2 => 2 alignemnts etc.

samtools view -h Aligned.sortedBy Coord.out.bam | \ # decompress the B
AM file
    egrep "^@\|\bNH:i:1\b" | \ # lines with either @ at the beginning
 of the line
    # or exact matches of NH:i:1 are returned

samtools view -S -b - > uniquely_aligned_reads.bam # turn the SAM line
s from stdin into a BAM file, - indicates standard input samtools
```

To filter out **reads with insert sizes greater than 1000bp**, one could make use of the CIGAR string. The example assume that the alignment program indicated large insertions with the N operator.

```
# for the sake of simplicity, let's work on the SAM file:
samtools view -h WT_1_Aligned.sortedByCoord.out.bam > WT_1_Aligned.sor
tedByCoord.out.sam

# here's an example using grep, excluding lines with at least four dig
its followed by N
egrep -v "[0-9][0-9][0-9][0-9]N" WT_1_Aligned.sortedByCoord.out.sam >
 smallInsert_reads.sam

# awk can be used to match a regex within a specified column
awk '!($6 ~ /[0-9][0-9][0-9][0-9]N/) {print $0}' WT_1_Aligned.sortedBy
Coord.out.sam > smallInsert_reads.sam
```

To retrieve **intron-spanning reads**, the commands will be similar:

```
# egrep allows for nicer regex syntax than grep
egrep "(^@|[0-9]+M[0-9]+N[0-9]+M)" WT_1_Aligned.sortedByCoord.out.sam
 > intron-spanning_reads.sam

# the same result achieved with awk
awk '$1 ~ /~@/ || $6 ~ /[0-9]+M[0-9]+N[0-9]+M/ {print $0}' WT_1_Aligne
d.sortedByCoord.out.sam > intron-spanning_reads.sam
```

Quality control of aligned reads

Once the reads have been aligned, the following properties should be assessed before downstream analyses are started:

- Could most reads be aligned?
- Are there any obvious biases of the read distributions?
- Are the replicate samples as similar to each other as expected?

Basic alignment assessments

There are numerous ways to do basic checks of the alignment success. An alignment of RNAseq reads is usually considered to have succeeded if the mapping rate is > 70%.

The very first QC of aligned reads should be to generally check the aligner's output. The **STAR** and **samtools index** commands generate the following files:

- Aligned.sortedByCoord.out.bam - information about the genomic loci of each read incl. its sequence
- Log.final.out - alignment statistics
- Log.progress.out - commands, parameters, and files used
- SJ.out.tab - genomic loci where splice junctions were detected and the number of reads overlapping with them
- Unmapped.out.mate1 text file with unmapped reads (similar to original **fastq** file)

Most aligners will return a summary of the basic stats of the aligned reads, such as the number of mapped reads. For, **STAR**, the information is stored in Log.final.out.

```
cat WT_1_Log.final.out
```

The number of *uniquely mapped reads* is usually the most important number. If you are handling more than two **BAM** file, it will certainly be wothwhile to visualize the alignment rate for all files, e.g., using MultiQC or your own, customized routine in R.

In addition to the log files generated by the mapping programm, there are numerous ways to obtain information about the numbers and kinds of reads stored in a **BAM** file, e.g., using **samtools** or **RSeQC**. The simples approach to finding out the number of alignments within a BAM file is to do a line count.

```
samtools view Aligned.sortedByCoord.out.bam | wc -l
```

Script for alignment QC

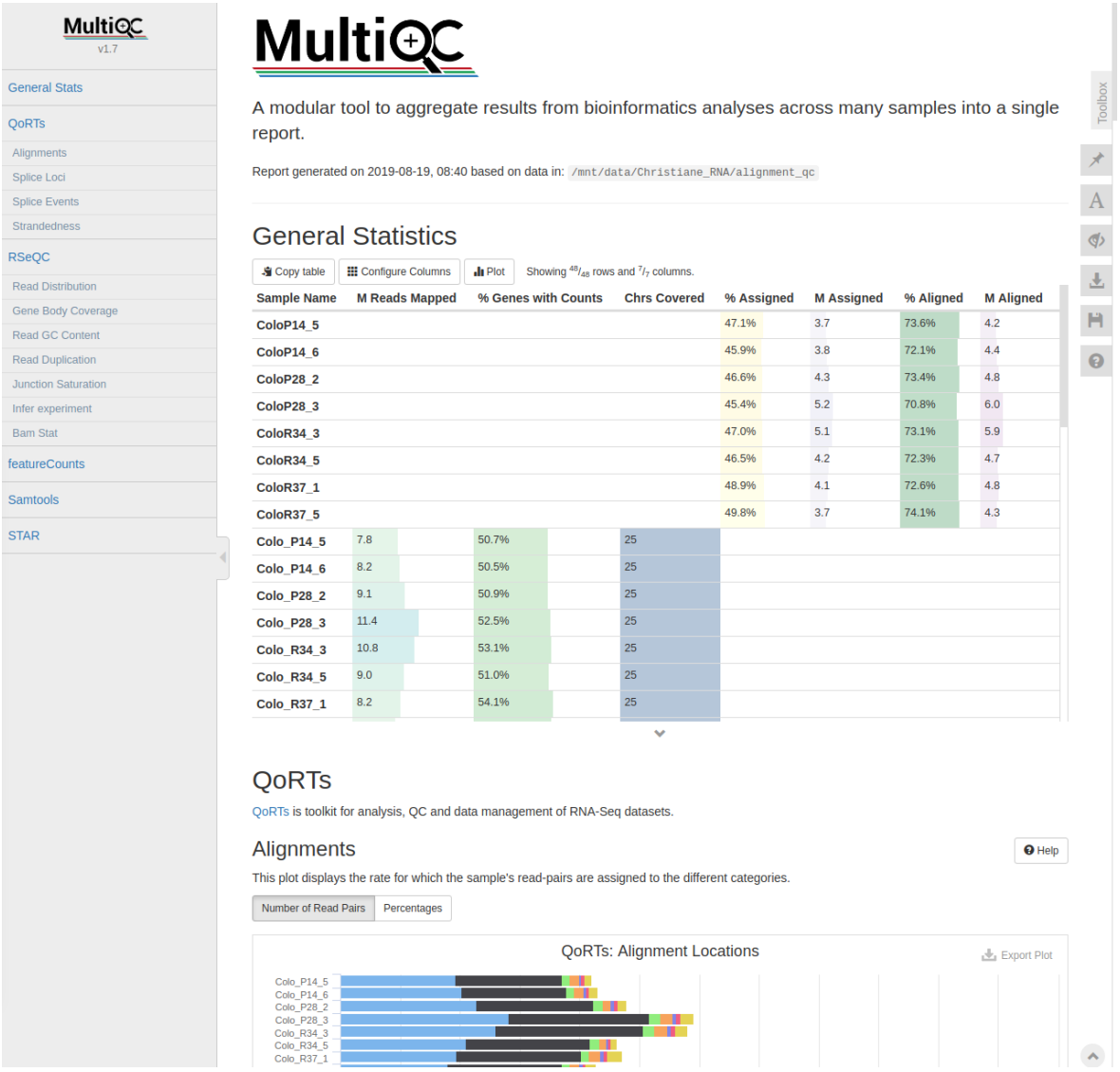
```
#!/bin/bash


# make one directory for every cancer cell line and collect the differ
ent QC outputs in the respective folder

# bam_stat
# Calculates reads mapping statistics for a BAM (or SAM) file. Note th
at
# uniquely mapped reads are determined on the basis of the mapping qua
lity.

# clipping_profile
# Estimates clipping profile of RNA-seq reads from BAM or SAM file. Th
is
# will fail if the aligner that was used does not support clipped mapp
ing
# (CIGAR strings must have S operation)

# geneBody_coverage
# Scales all transcripts to 100 bp, then calculates the coverage of ea
ch base.
# The read coverage should be uniform and ideally not show 5’ or 3’ bi
as
# since that would suggest problems with degraded RNA input or with cD
NA
# synthesis.

# infer_experiment
# Speculates how RNA-seq sequencing was configured, i.e., PE or SR and
# strand-specificity. This is done by subsampling reads and comparing
 their
# genome coordinates and strands with those of the transcripts from th
e ref-
# erence gene model. For non-strand-specific libraries, the strandedne
ss of
# the reads and the transcripts should be independent

# junction_annotation
# Compares detected splice junctions to the reference gene model, clas
sifying
# them into 3 groups: annotated, complete novel, partial novel (only o
ne of
# the splice sites is unannotated). The script differentiates between
 splice
# events (single read level) and splice junctions (multiple reads show
the same
# splicing event).

# junction saturation
# Similar concept to RPKM saturation: splice junctions are detected fo
r each
# sampled subset of reads. The detection of annotated splice sites sho
uld be
# saturated with the maximum coverage (= all supplied reads), otherwis
e
# alternative splicing analyses are not recommended because low abunda
nce
```

```
# splice sites will not be detected.

# read_distribution
# Calculates fractions of reads mapping to transcript features such as
exons,
# 5'UTR, 3' UTR, introns.

# read_duplication
# Determine read duplication rate, based on the sequence only
# (output.dup.seq.DupRate.xls) as well as on the alignment positions
# (output.dup.pos.DupRate.xls).

# read_GC
# Calculates % GC for all reads. Similar to FASTQC's GC distribution p
lot,
# the peak of the resulting histogram should coincide with the average
GC
# content of the genome.

# read_NVC
# Calculates the nucleotide composition across the reads; similar to F
ASTQC's
# per base sequence content analysis.

# read_quality
# Calculates distributions for base qualities across reads; similar to
FASTQC's
# per base sequence quality analysis.

# tin
# Calculates the transcript integrity number (TIN) (not to be confused
with
# the RNA integrity number, RIN, that is calculated before sequencing
 based
# on the 28S/18S ratio). TIN is calculated for each transcript and rep
resents
# the fraction of the transcript with uniform read coverage.

# QoRTs
# see http://hartleys.github.io/QoRTs/index.html


# MultiQC will eventually combine all the different QC stats into one
 document

# create the folders
mkdir alignment_qc
cd alignment_qc
mkdir Colo HT29 SW480

# get the STAR log files and copy them to the alignment folder
ln -s /mnt/data/Christiane_RNA/bam_files
for cell_line in Colo HT29 SW480
do
    for log_file in bam_files/${cell_line}/*Log*final.out
        do
                cp $log_file ${cell_line}/
    done
```

```
done

# samtools flagstat
for cell_line in Colo HT29 SW480
do
    for bam_file in bam_files/${cell_line}/*bam
        do
                samtools index $bam_file # the BAM file needs an index
.bai file
        samtools flagstat $bam_file > ${cell_line}/${cell_line}_${bam_
file: -9}_flagstat.out
    done
done

for cell_line in Colo HT29 SW480
do
        for bam_file in bam_files/${cell_line}/*bam
        do
                echo 'bam_stat.py'
                bam_stat.py -i $bam_file > ./${cell_line}/${cell_line}
_${bam_file: -9}_bam_stat.out
                echo 'clipping_profile.py'
                clipping_profile.py -i $bam_file -o ./${cell_line}/${c
ell_line}_${bam_file: -9}_clipping_profile -s "SE"
                echo 'geneBody_coverage.py'
                geneBody_coverage.py -r /mnt/data/Christiane_RNA/hg19_
UCSC_knownGene.bed -i $bam_file -o ./${cell_line}/${cell_line}_${bam_f
ile: -9}_genebody
                echo 'infer_experiment.py'
                infer_experiment.py -r /mnt/data/Christiane_RNA/hg19_U
CSC_knownGene.bed -i $bam_file > ./${cell_line}/${cell_line}_${bam_fil
e: -9}_infer_experiment.txt
                echo 'inner_distance.py'
                inner_distance.py -r /mnt/data/Christiane_RNA/hg19_UCS
C_knownGene.bed -i $bam_file -o ./${cell_line}/${cell_line}_${bam_fil
e: -9}_innerdist
                echo 'junction_annotation.py'
                junction_annotation.py -r /mnt/data/Christiane_RNA/hg1
9_UCSC_knownGene.bed -i $bam_file -o ./${cell_line}/${cell_line}_${bam
_file: -9}_juncannot
                echo 'junction_saturation.py'
                junction_saturation.py -r /mnt/data/Christiane_RNA/hg1
9_UCSC_knownGene.bed -i $bam_file -o ./${cell_line}/${cell_line}_${bam
_file: -9}_juncsat
                echo 'read_distribution.py'
                read_distribution.py -r /mnt/data/Christiane_RNA/hg19_
UCSC_knownGene.bed -i $bam_file > ./${cell_line}/${cell_line}_${bam_fi
le: -9}_read_distribution.txt
                echo 'read_duplication.py'
                read_duplication.py -i $bam_file -o ./${cell_line}/${c
ell_line}_${bam_file: -9}_read_dup
                echo 'read_GC.py'
                read_GC.py -i $bam_file -o ./${cell_line}/${cell_line}
_${bam_file: -9}_read_GC
                echo 'read_NVC.py'
                read_NVC.py -i $bam_file -o ./${cell_line}/${cell_lin
e}_${bam_file: -9}_read_NVC
                echo 'read_quality.py'
```

```
                read_quality.py -i $bam_file -o ./${cell_line}/${cell_
line}_${bam_file: -9}_read_Q
                echo 'tin.py'
                tin.py -i $bam_file -r /mnt/data/Christiane_RNA/hg19_U
CSC_knownGene.bed

        done
done

mkdir mRIN_input
mv *.summary.txt *.tin.xls ./mRIN_input/


for cell_line in Colo HT29 SW480
do
        for bam_file in bam_files/${cell_line}/*bam
        do
        java -Xmx4g -jar ./QoRTs-STABLE.jar QC --singleEnded --generat
ePdfReport $bam_file \
         /mnt/data/Christiane_RNA/hg19.gtf ./alignment_qc/${cell_lin
e}/${cell_line}_${bam_file: -9}
    done
done

# aggregate all QC results into one interactive html document

mkdir multiQC
cd multiQC

# link the individual cell lines
ln -s /mnt/data/Christiane_RNA/alignment_qc/Colo
ln -s /mnt/data/Christiane_RNA/alignment_qc/HT29
ln -s /mnt/data/Christiane_RNA/alignment_qc/SW480

# run MultiQC

multiqc . --dirs
```

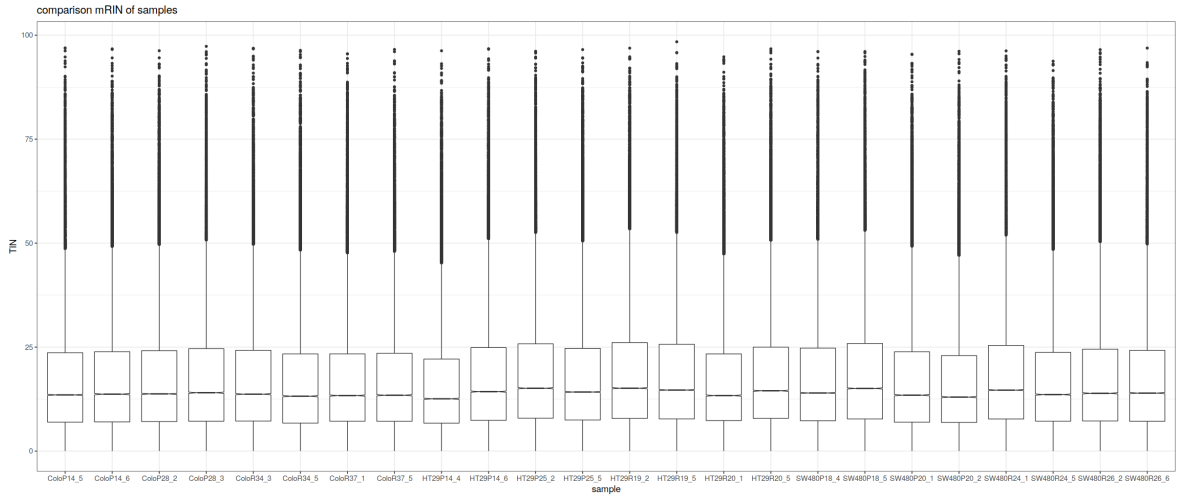
MultiQC report alignment



in silico mRIN calculation

The RNA integrity number that is calculated during library preparation to assess the RNA quality can be compared to RSeQC's tin.py script which measures mRNA degrdation *in silico* using the deviation from an expected uniform read distribution across the gene body as a proxy.

```r
## Compare the distribution of mRIN (= TIN) values across different sa
mples
#########################################################################
######
## This script will result in boxplots based on mRIN values calculated
by
## RSeQC on 4 example files, named UHR-RIN0 to UHR-RIN9.
## To run the script as is, download the xls files from the respective
folder
## into your current working directory.
#########################################################################
#######

## read in the RSeQC results
# list the respective files
TIN.files <- list.files(pattern = "xls")

# lapply will iterate over the list of file names in TIN.files, read t
he respective
# tables and return a list of data frames where each data frame corres
ponds to
# one of the original xls tables
TIN.list <- lapply(TIN.files, function(x) read.table(x, header = TRUE)
[c("geneID", "TIN")])

# to give meaningful names to each data frame, we can use regex on the
file names
# the regex will have to be adjusted if you use different files
names(TIN.list) <- gsub("(.*).tin.*", "\\1", TIN.files)

# make a long data frame that is suitable for ggplot2 plotting
TIN.df <- as.data.frame(do.call(rbind, TIN.list))

# add a column that indicates the sample type for each gene ID and TIN
value,
# here, I use the information from the original data frame's name in t
he TIN.list
# which is kept in the row.names of TIN.df
TIN.df$sample <- gsub("\\.[0-9]+", "", row.names(TIN.df))

# make the boxplots
library(ggplot2)
ggplot(data = TIN.df, aes(x = sample, y = TIN)) + geom_boxplot(notch=F
ALSE)

# excluding genes with TIN = 0, which are most likely due to lack of r
ead coverage
ggplot(data = subset(TIN.df, TIN > 0), aes(x = sample, y = TIN)) +
  geom_boxplot(notch=TRUE) +
  theme_bw(base_size = 14) +
  ggtitle("comparison mRIN of samples")
```

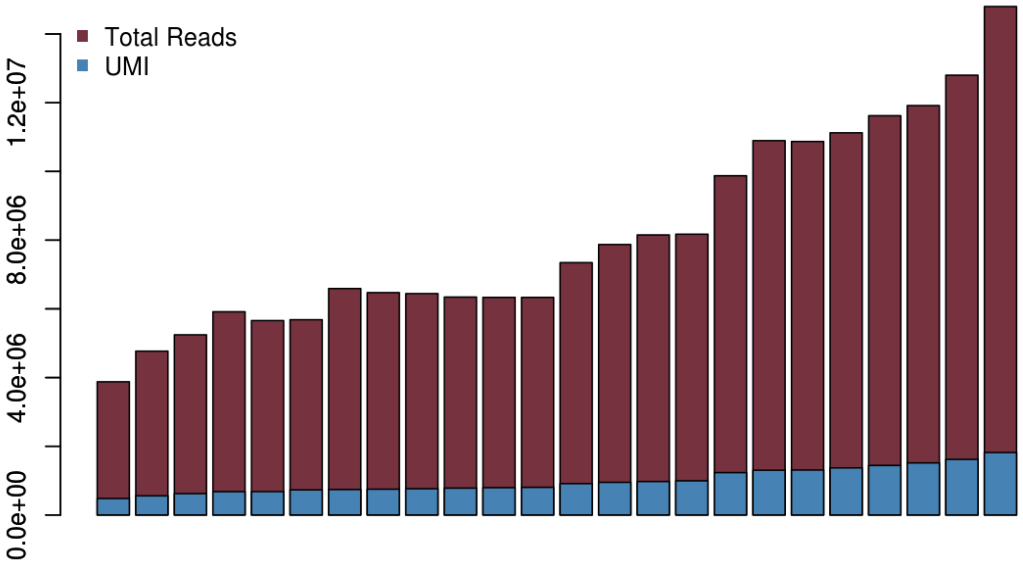


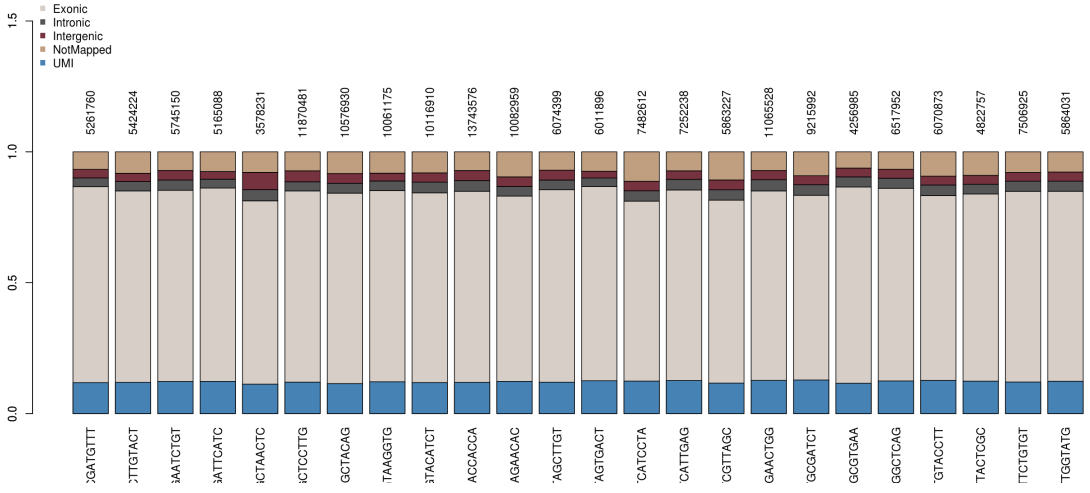
Read Statistics Dropseq Pipeline

The Dropseq Pipeline of the Rad group runs an R script which produces figures depicting read statistics.

```
args <- commandArgs(TRUE)

RawReadFile = "UnfilteredReadCount.txt"

BarcodeFile = args[1]

RawReads = read.table(RawReadFile,header=F,sep="\t",fill=NA)
rawData = unlist(strsplit(as.character(RawReads[,1]),"XC:Z:"))
ReadCounts = as.numeric(trimws(rawData[seq(1,length(rawData),2)]))
Samples  = as.character(rawData[seq(2,length(rawData),2)])
Rawdata = data.frame(ReadCounts=ReadCounts,Sample=Samples)



barcodesUsed = as.character(read.table(BarcodeFile,sep="\t",header=F)
[,1])


CountMat = read.table("DGE_Matrix.txt",header=T,sep="\t")
SumStat = read.table("Summary.txt",header=F,sep="\t")
barcodesCount = trimws(unlist(strsplit(as.character(SumStat[,1]),"XC:
Z:")))
CountsPerFeature = barcodesCount[seq(1,length(barcodesCount),2)]
BarcodePerFeature = barcodesCount[seq(2,length(barcodesCount),2)]

SumStat = data.frame(Counts=as.numeric(CountsPerFeature),Barcode=Barco
dePerFeature,Feature=as.character(SumStat[,2]))



ReducedMat = SumStat[SumStat[,"Barcode"] %in% barcodesUsed,]



InfoMat = data.frame(Barcode=barcodesUsed,Exonic=0,Intronic=0,Intergen
ic=0)
rownames(InfoMat) = InfoMat[,"Barcode"]


for (i in barcodesUsed){
          mat = ReducedMat[ReducedMat[,"Barcode"]==i,]
          coding=  mat[mat[,"Feature"]=="XF:Z:CODING","Counts"] +  mat
[mat[,"Feature"]=="XF:Z:EXONIC","Counts"] +mat[mat[,"Feature"]=="XF:Z:
UTR","Counts"]
          intronic =  mat[mat[,"Feature"]=="XF:Z:INTRONIC","Counts"]
          intergenic =  mat[mat[,"Feature"]=="XF:Z:INTERGENIC","Count
s"]
          if(!identical(numeric(0),coding))
             {
             InfoMat[i,"Exonic"] = coding
             }
          if(!identical(numeric(0),intronic))
             {
             InfoMat[i,"Intronic"] = intronic
             }
          if(!identical(numeric(0),intergenic))
             {
             InfoMat[i,"Intergenic"]=intergenic
```

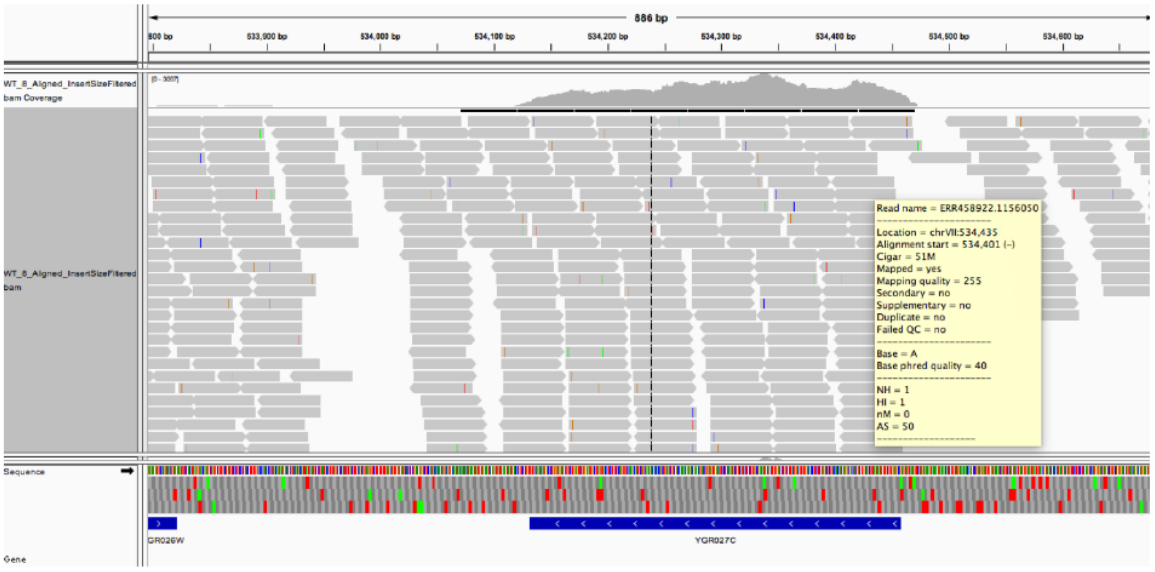
```
            }
          }



#InfoMat = data.frame(Barcode=barcodesUsed,Exonic=Exonic,Intronic=Intr
onic,Intergenic=Intergenic)
RawMat = Rawdata[Rawdata[,"Sample"] %in% barcodesUsed,]


merged  = merge(RawMat,InfoMat,by.x="Sample",by.y="Barcode")
UMICount = apply(CountMat[,as.character(merged[,"Sample"])],2,sum)
merged = data.frame(merged,UmiCount=UMICount)



colors =c("#D7CEC7","#565656","#76323F","#C09F80")

A = merged[,c("Exonic","Intronic","Intergenic","UmiCount")]/merged[,"R
eadCounts"]
B = t(as.matrix(A[,c("Exonic","Intronic","Intergenic")]))
D = rbind(1-apply(B,2,sum),B)
rownames(D) =  c("NotMapped","Exonic","Intronic","Intergenic")
D = D[c("Exonic","Intronic","Intergenic","NotMapped"),]


tiff("SummaryBarplot.tiff",2600,1200,res=150)
x = barplot(D,col=colors,ylim=c(0,1.6),las=3)
barplot(merged[,"UmiCount"]/merged[,"ReadCounts"],col="#4682b4",add=T,
xlab="",ylab="")
legend("topleft",c(rownames(D),"UMI"),col=c("#D7CEC7","#565656","#7632
3F","#C09F80","#4682b4"),pch=c(15,15,15,15,15),bty="n")
text(x,1.15,apply(InfoMat[,2:4],1,sum),srt=90)
dev.off()


Ordering = order(merged[,6])


tiff("BarplotReadCounts.tiff",1200,800,res=150)
barplot(merged[Ordering,2],col=colors[3])
barplot(merged[Ordering,6],add=T,col="#4682b4")
legend("topleft",c("Total Reads","UMI"),pch=c(15,15),col=c(colors[3],"
#4682b4"),bty="n")
dev.off()

write.table(merged,file="ReadCountDistribution.txt",quote=F,sep="\t",r
ow.names=F,col.names=T)
```



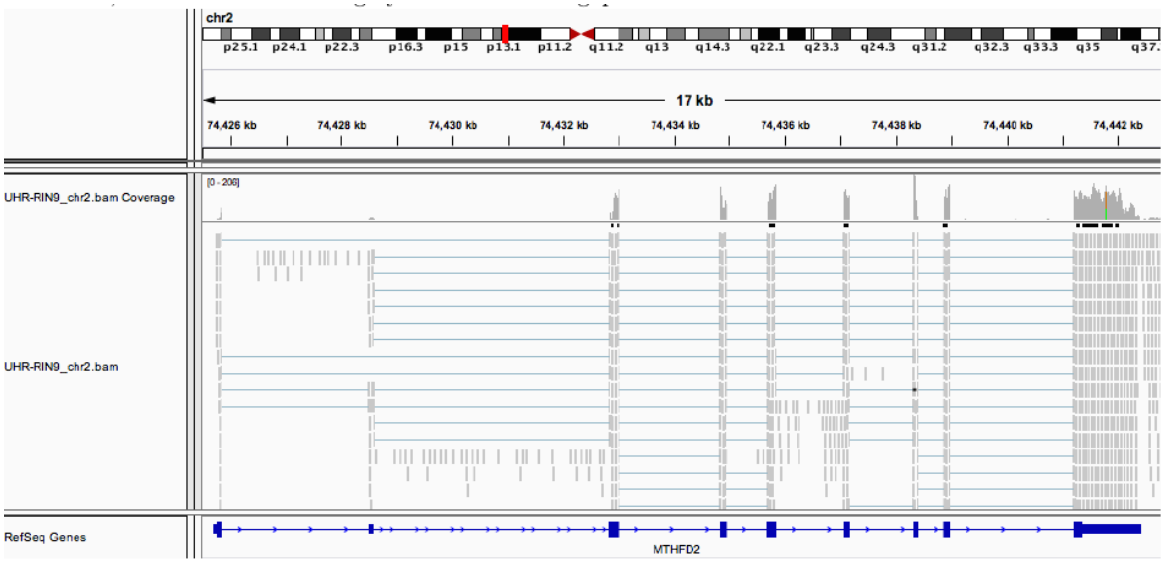


Visualization of aligned reads

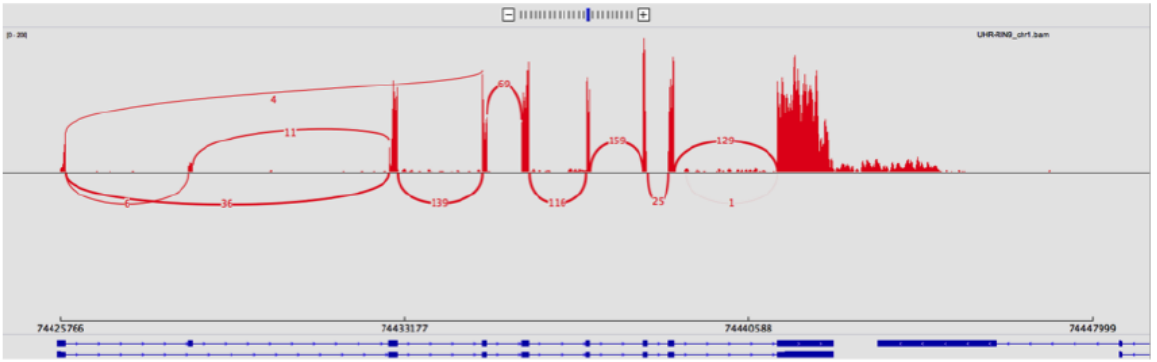
It is always a good idea to visually check the results, i.e., ensure the reads align to the expected regions, preferably without too many mismatches. The IGV Genome Browser can display numerous file formats, e.g., indexed BAM files with aligned reads and BED files with information about loci (such as genes).



On top of the read alignment display, IGV also produces a coverage line that allows for quick identification of highly covered regions. Blue lines within the reads indicate insertions with respect to the reference genome, red lines indicate deletions. Human genes, however, tend to have multiple introns, which means that exon-exon-spanning reads must be aligned with often lengthy gaps within them. In IGV this gaps are indicated by horizontal grey lines within a read:



If one is interested in the splice junctions of a particular gene, IGV can generate Sashimi plots:

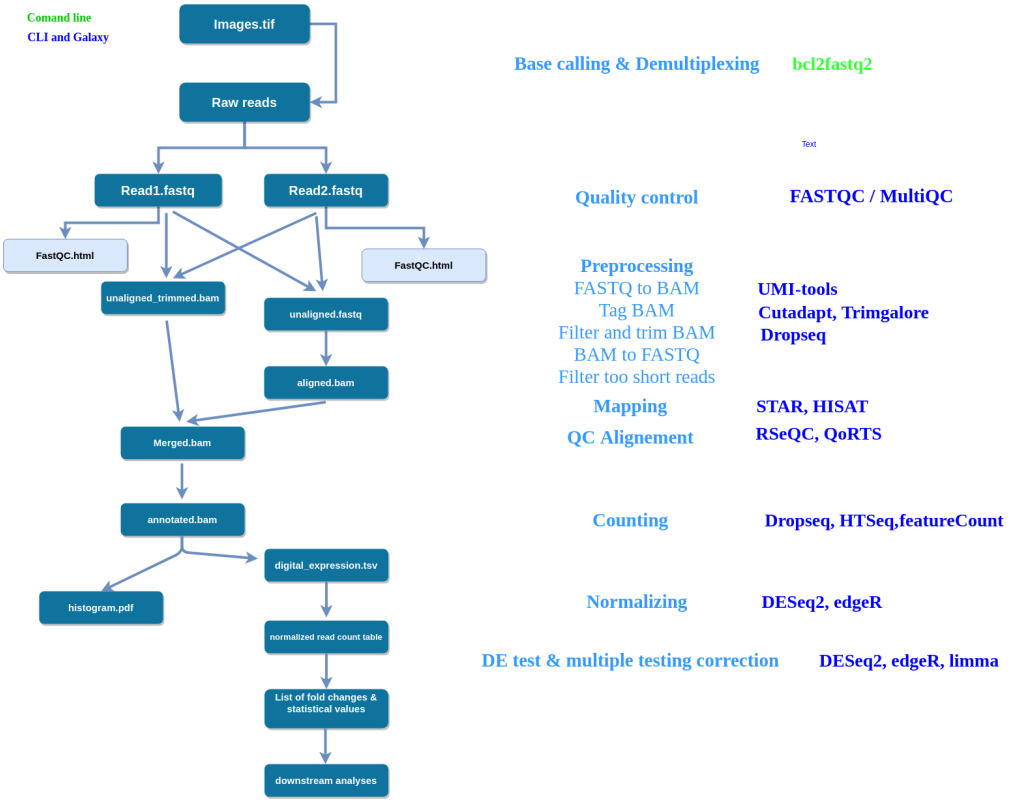


In the Sashimi plot, bar graphs indicate read coverage, arcs indicate splice junctions, and numbers represent the number of reads that contain the respective splice junction.

Read Quantification

The goal of our RNASeq is to perfom differential expression testing to determine which genes are expressed at different levels between conditions. These genes can offer biological insights into the proceses affected by the condition(s) of interest.

To determine the expression levels of genes, the RNAseq workflow follows the steps detailed in the image below. All steps were performed either on a Galaxy Server [Galaxy Server DKTK Blum (https://blum.galaxy.lafuga.genzentrum.lmu.de/galaxy/)](https://blum.galaxy.lafuga.genzentrum.lmu.de/galaxy/) or on the command line (Linux/Unix). The differential expression analysis and any downstream functional analysis are generally performed in R using R packages specifically designed for the complex statistical analyses required to determine whether genes are differentially expressed.



In the following, I will walk you through an end-to-end gene level RNAseq differential expression workflow. I t will start with the count matrix, exploratory data analysis for quality assessment and to explore relationship between sample. After that we perform differential expression analysis, and visually explore the results prior to performing downstream functional analysis.

There are two different ways of approaching the determination of individual transcripts expression levels, one can either assign all the reads to a given gene (effectively ignoring the presence of individual isoforms), or one can try to infer the quantity of individual transcripts.

Gene-based read counting

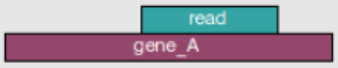
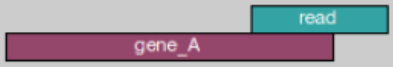
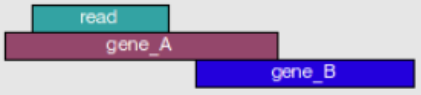
To obtain gene-level quantifications, one can either directly count reads overlapping with gene loci or use transcript-level quantification, followed by some way of aggregating the values per gene.

The most popular tools for gene quantification are **htseq-count** and **featureCounts**. htseq-count offers three different modes. The recommended mode is **union**, which counts overlaps even if a read only shares parts of its sequence with a genomic feature and disregards that overlap more than one feature. This is similar to **featureCounts** that calls a hit if any overlap (1bp or more) is found between the read and a feature and provides the option to either exclude multi-overlap reads or to count then for each feature that is overlapped.

In addition to the nature and lengths of the reads, gene expression quantification will be strongly affected by the underlying gene models that are usually supplied to the quantification programs via **GTF** or **BED** files.

When counting reads, make sure you know how the programm handles the following:

- overlap size (full read vs. partial overlap)
- multi-mapping reads
- reads overlapping multiple genomic features of the same kind
- reads overlapping introns

	union	intersection_strict	intersection_nonempty
	gene_A	gene_A	gene_A
	gene_A	no_feature	gene_A
	gene_A	no_feature	gene_A
	gene_A	gene_A	gene_A
	gene_A	gene_A	gene_A
	ambiguous	gene_A	gene_A
	ambiguous	ambiguous	ambiguous

Gene-based read counting using DropSeq Pipeline

The script of the DropSeq Pipeline adds annotation to the reads that is dependent on the genome read, such as any genes, exons that the read overlaps.

The DropSeq program **TagReadWithGeneExon** adds a BAM tag "GE" onto reads when the read overlaps the exon of a gene. This tag contains the name of the gene, as reported in the annotations file. You can use either a GTF or RefFlat annotation file.

```
TagReadWithGeneExon
I=merged.bam
O=star_gene_exon_tagged.bam
ANNOTATIONS_FILE=${refFlat}
TAG=GE
```

As Alternative for **TagReadWithGeneExon** you can use **TagReadWithGeneFunction**. This program provides a more flexible and informative set of tags for reads that allow downstream programs to measure not only digital expression of reads that overlap exons, but can leverage reads that overlap introns as well. This program provides 3 tags for each read, **gn**[gene name], **gs**[gene strand] and **gf**[gene function]. These tags can have more than one value, and the values are comma separated. These tags can also co-exist with the original tagger (**TagReadWithGeneExon**) as the tag names are different, so if you use those tags for other purposes, you can tag your BAM with both taggers.

```
TagReadWithGeneFunction
I=merged.bam
O=star_gene_exon_tagged.bam
ANNOTATIONS_FILE=${refFlat}
```

Generate digital expression matrix

To digitally count gene transcripts, a list of UMI's in each gene, within each Sample, is assembled, and UMI's within eit distance = 1 are merged together. The total number of unique UMI sequences is counted, and this number is reported as the number of transcripts of that gene for a given sample.

Extracting Digital Gene Expression data from an aligned library is done using the DropSeq program **DigitalExpression**. The input to this program is the aligned BAM from alignment workflow. There are two outputs available: the primary is the DGE matrix, with each a row for each gene, and a column for each cell. The secondary analysis is a summary of the DGE matrix on a per-sample level, indicating the number of genes and transcripts observed.

The file is ordered by list of sample barcodes (input sample barcode file or based on number of reads per sample), then the number of UMIs per gene, then alphabetically by gene when they have the same number of UMI's. There are no entries when a sample does not habe expression of a gene.

```
DigitalExpression
I=out_gene_exon_tagged.bam
O=out_gene_exon_tagged.dge.txt.gz
SUMMARY=out_gene_exon_tagged.dge.summary.txt
NUM_CORE_BARCODES=100
```

Gene-based read counting using featureCounts

```
#!/bin/bash

#counts number of reads overlapping with genes using featureCounts

for cell_line in Colo HT29 SW480
do

        /home/sebastian/biotools/featureCounts/bin/featureCounts \
                -a /mnt/data/Christiane_RNA/hg19.gtf \
        -o ./bam_files/${cell_line}/featureCounts_results.txt \
                ./bam_files/${cell_line}/*bam #use all BAM files in th
e folder of specific cell line

done
```

The output of featureCounts consists of two files:

1. The one defined by the -o parameter (e.g., featureCounts_results.txt) - this one contains the actual read counts per gene (with gene ID, genomic coordinates of the gene including strand and length); the first line starting with # contains the command that was used to generate the file.

2. A file with the suffix.summary: This file gives a quick overview about how many reads could be assigned to genes and the reason why some of them could not be assigned. This is a very useful file to double check the settings you've chosen for the counting.

So what does this count data acually represent?

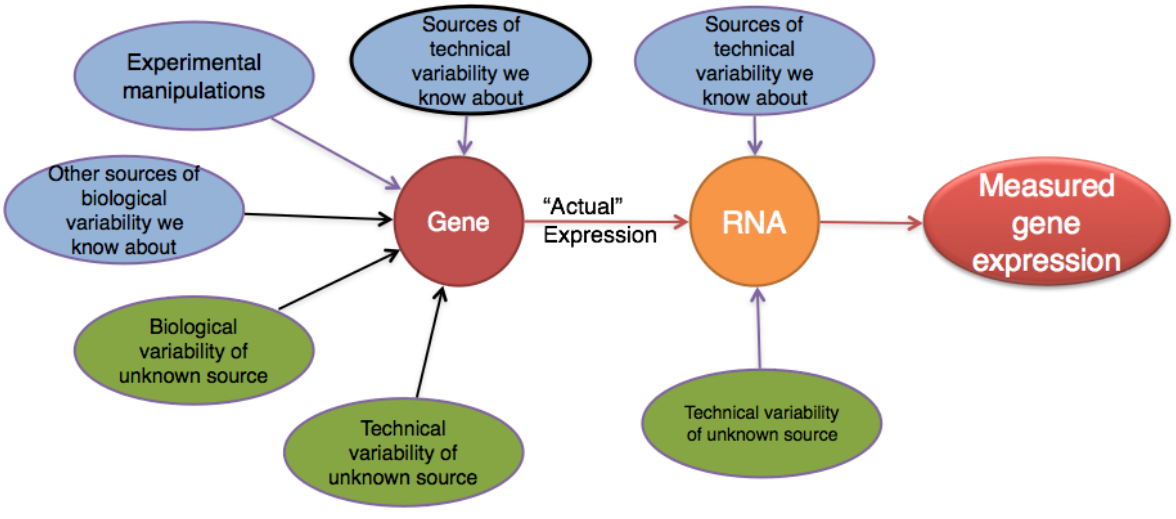
The count data used for differential expression analysis represent the number of sequence reads that originated from a particular gene. The higher the number of counts, the more reads associated with that gene, and the assumption that there was a higher level of that gene in the sample.

With differential expression analysis, we are looking for genes that change in expression between two or more groups.

- parental vs. resistent
- case vs. control
- correlation of expression with some variable or clinical outcome

Why does it not work to identify differentially expressed gene by ranking the genes by how different they are between the two groups (based on the fold change values)?

More ofthen than not, there is much more going on with your data than what you are anticipating. Genes that vary in expression level between samples is a consequence of not only the experimental variables of interest but also due to extraneous sources. The goal of differential expression analysis is to determine the relative role of these effects, and to separate the "interesting" from the "uninteresting".



Courtesy of Paul Pavlidis, UBC

The "uninteresting" presents as sources of variation in your data, and so even though the mean expression levels between sample groups may appear to be quite different, it is possible that the difference is not actually significant. We need to take into account the variation in the data, and where it might be coming from, when determining whether genes are differentially expressed.

The goal of differential expression analysis is to determine, for each gene, whether the differences in expression (counts) **between groups** is significant given the amount of variation observed **within groups** (replicates). To test for significance, we need an appropriate statistical model that accurately

performs normalization (to account for differences in sequencing depth, etc.) and variance modeling (to account for few numbers of replicates and large dynamic expression range).

RNAseq count distribution

To determine the appropriate statistical model, we need information about the distribution of counts. To get an idea about how RNAseq counts are distributed, plot the counts for a single sample.

In [305]:

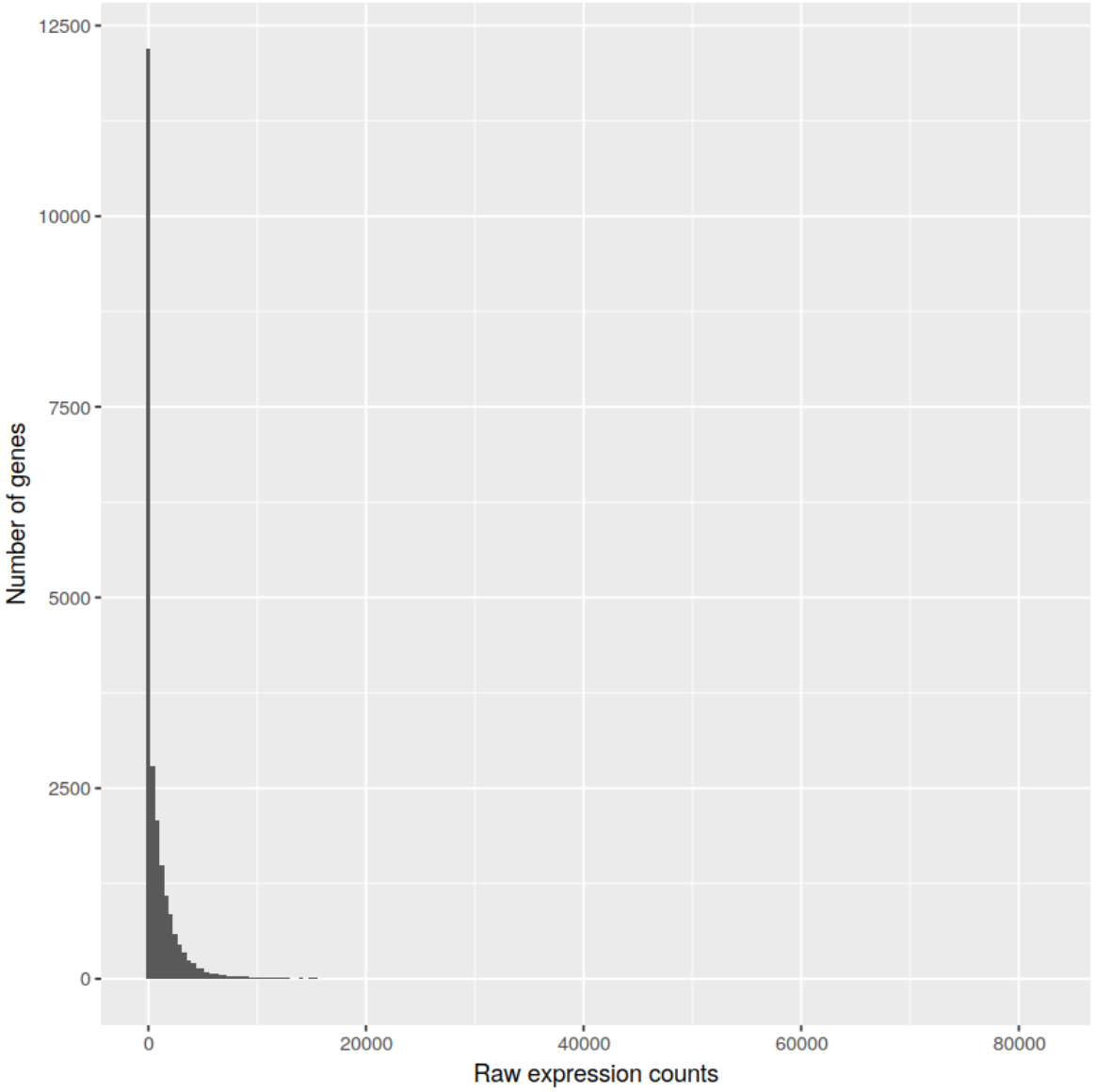
```
## Setup
### Bioconductor and CRAN libraries used
library(tidyverse)
library(RColorBrewer)
library(DESeq2)
library(pheatmap)
library(DEGreport)

# Load in data
data  <- read.table("Mov10_full_counts.txt", header=TRUE, row.names=1)
meta  <- read.table("Mov10_full_meta.txt", header=TRUE, row.names=1)

# Check classes of the data
class(data)
class(meta)
```

'data.frame'

'data.frame'

In [306]:

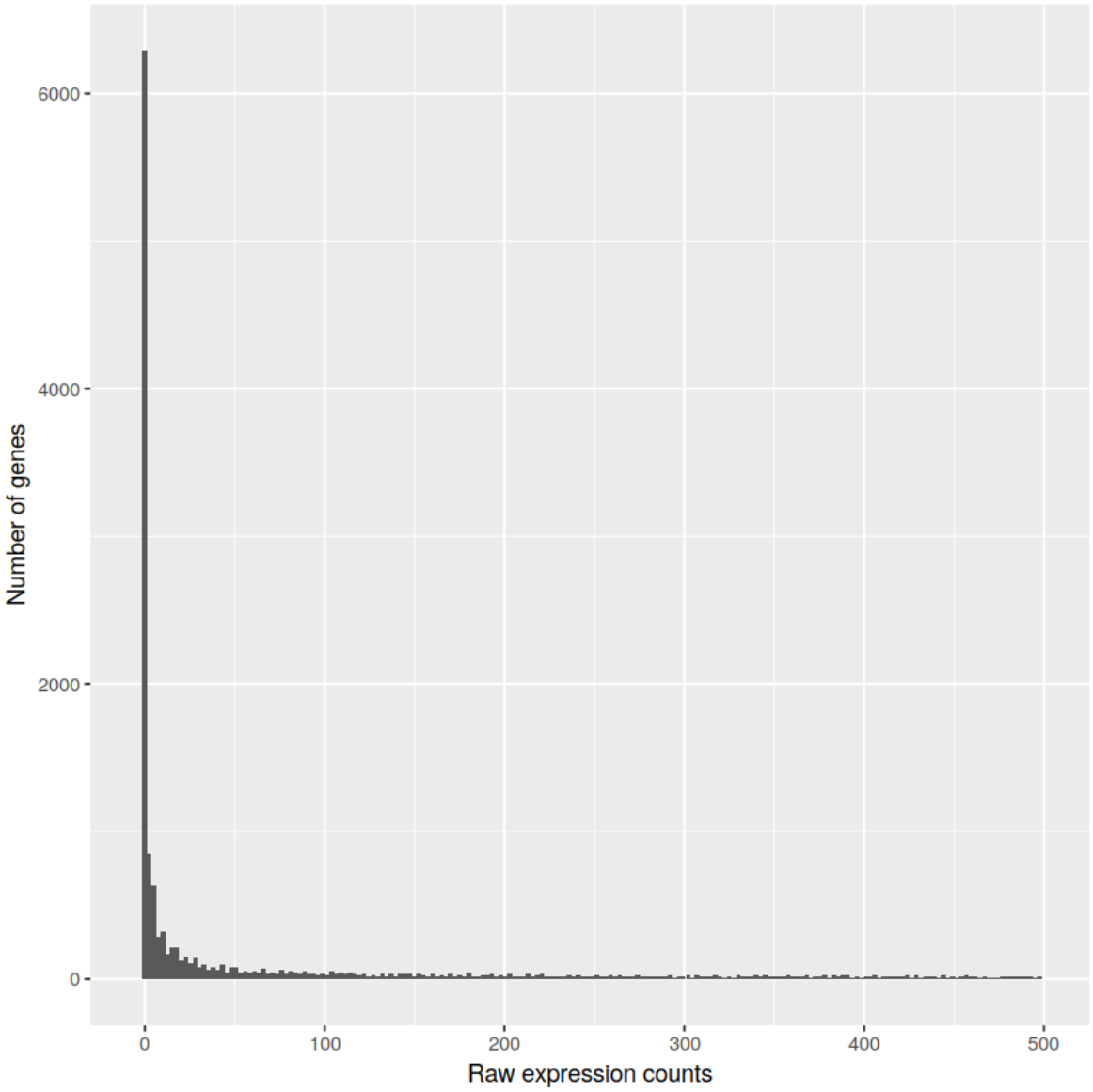
```
ggplot(data) +
  geom_histogram(aes(x = Mov10_oe_1), stat = "bin", bins = 200) +
  xlab("Raw expression counts") +
  ylab("Number of genes")
```



In [307]:

```
# zoom close to zero
ggplot(data) +
   geom_histogram(aes(x = Mov10_oe_1), stat = "bin", bins = 200) +
   xlim(-5, 500)  +
   xlab("Raw expression counts") +
   ylab("Number of genes")
```

```
Warning message:
"Removed 9121 rows containing non-finite values (stat_bin)."Warning message:
"Removed 2 rows containing missing values (geom_bar)."
```



These images illustrate some common features of RNAseq count data, including a **low number of counts associated with a large proportion of genes**, and a long right tail due to **lack of any upper limit for expression.**

Modeling count data

Count data is often modeled using the **binomial distribution**, which can give you the **probability of getting a number of heads upon tossing a coin a number of times**. However, not all count data can be fit with the binomial distribution. The binomial is based on discrete events and used in situations when you have a certain number of cases.

When **the number of cases is very large (i.e. people who buy lottery tickets), but the probability of an event is very small(probability of winning)**, the **Poisson distribution** is used to model these types of count data. The Poisson is similar to the binomial, but is based on continous events.

With RNAseq, a very large number of RNAs are represented and the probability of pulling out a particular transcript is very small. Thus, it would be an appropriate situation to use the Poisson distribution. However, a unique property of this distributiion is that the mean==variance. Realistically, with RNASeq data there is always some biological variation present across the replicates (within a sample class). Genes with larger average expression levels will tend to have larger observed variances across replicates.

If the proportions of mRNA stayed exactly constant between the biological replicates for each sample class, we could expect Poisson distribution (where mean==variance). But this doesn't happen in practice, and so the Poisson ditribution is only considered appropriate for a single biological sample.

The model that fits best, given this type of variability between replicates, is the Negative Binomial (NB) model. Essentially, **the NB model is a good approxiamtion for data where the mean < variance**, as is the case with RNAseq count data.

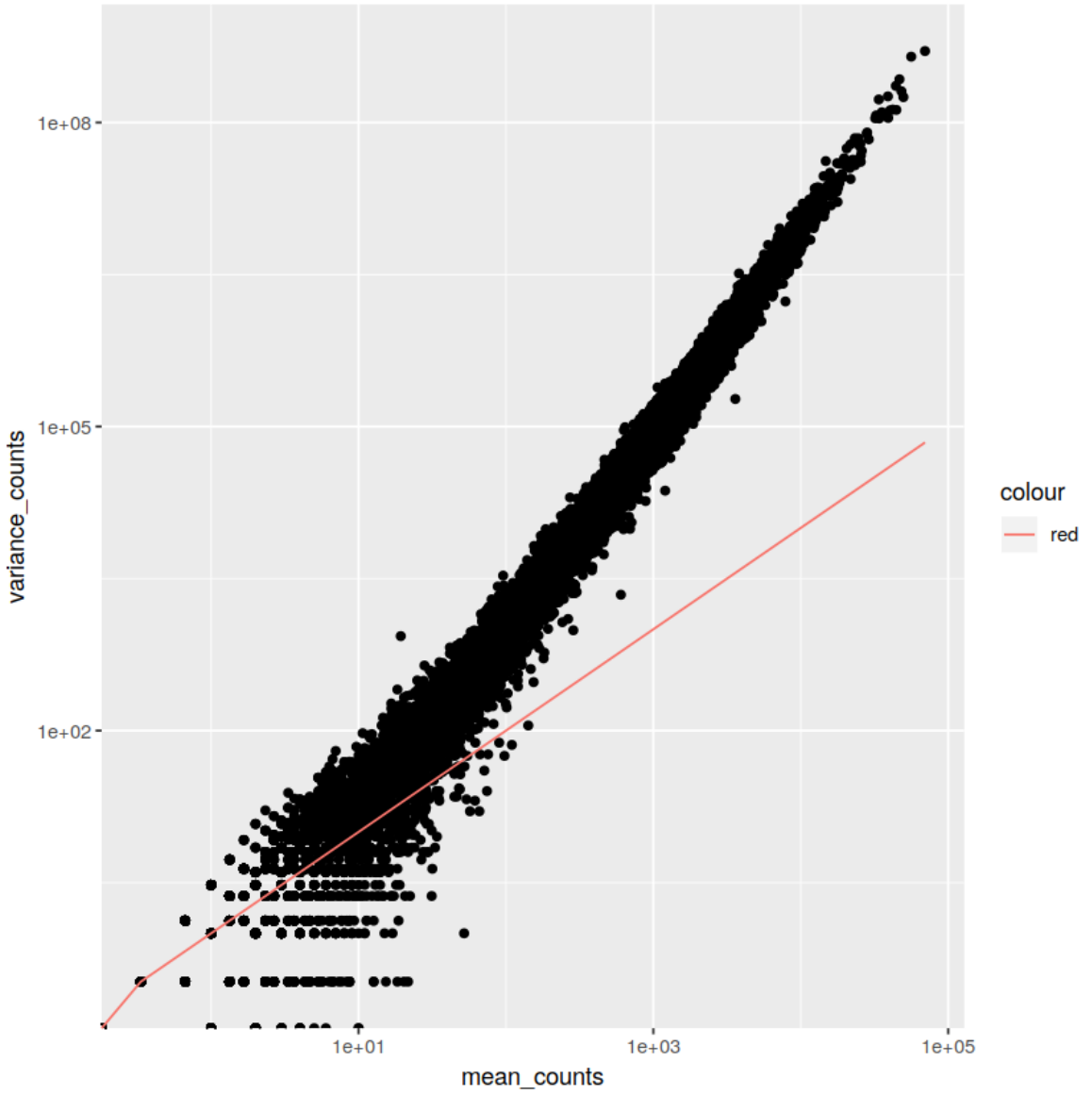
How do I know if my data should be modeled using the Poisson distribution or Negative binomial distribution?

If it's count data, it should fit the negative binomial. However it can be helpful to plot the mean versus the variance of your data. *Remember for the Poisson model, mean==variance, but for NB, mean < variance.

In [308]:

```
mean_counts <- apply(data[, 3:5], 1, mean)
variance_counts <- apply(data[, 3:5], 1, var)
df <- data.frame(mean_counts, variance_counts)

ggplot(df) +
        geom_point(aes(x=mean_counts, y=variance_counts)) +
        geom_line(aes(x=mean_counts, y=mean_counts, color="red")) +
        scale_y_log10() +
        scale_x_log10()
```

```
Warning message:
"Transformation introduced infinite values in continuous y-axis"Warning messa
ge:
"Transformation introduced infinite values in continuous x-axis"Warning messa
ge:
"Transformation introduced infinite values in continuous y-axis"Warning messa
ge:
"Transformation introduced infinite values in continuous x-axis"
```



Note that in the above figure, the variance across replicates tends to be greater than the mean (red line), especially for genes with large mean expression levels. This is a good indication that our data do not fit the Poisson distribtution an we need to account for this increase in variance using the negative binomial model.

Improving mean estimates (i.e. reducing variance) with biological replicates

The variance or scatter tends to reduce as we increase the number of biological replicates (the distribution will approach the Poisson distribution with increasing numbers of replicates), since standard deviations of averages are smaller than standard deviations individual observations. **The value of additional replicates is that as you add more data (replicates), you get increasingly precise estimates of group means, and ultimatly greater confidence in the ability to distinguish differences between sample classes (i.e. more DE genes)**.

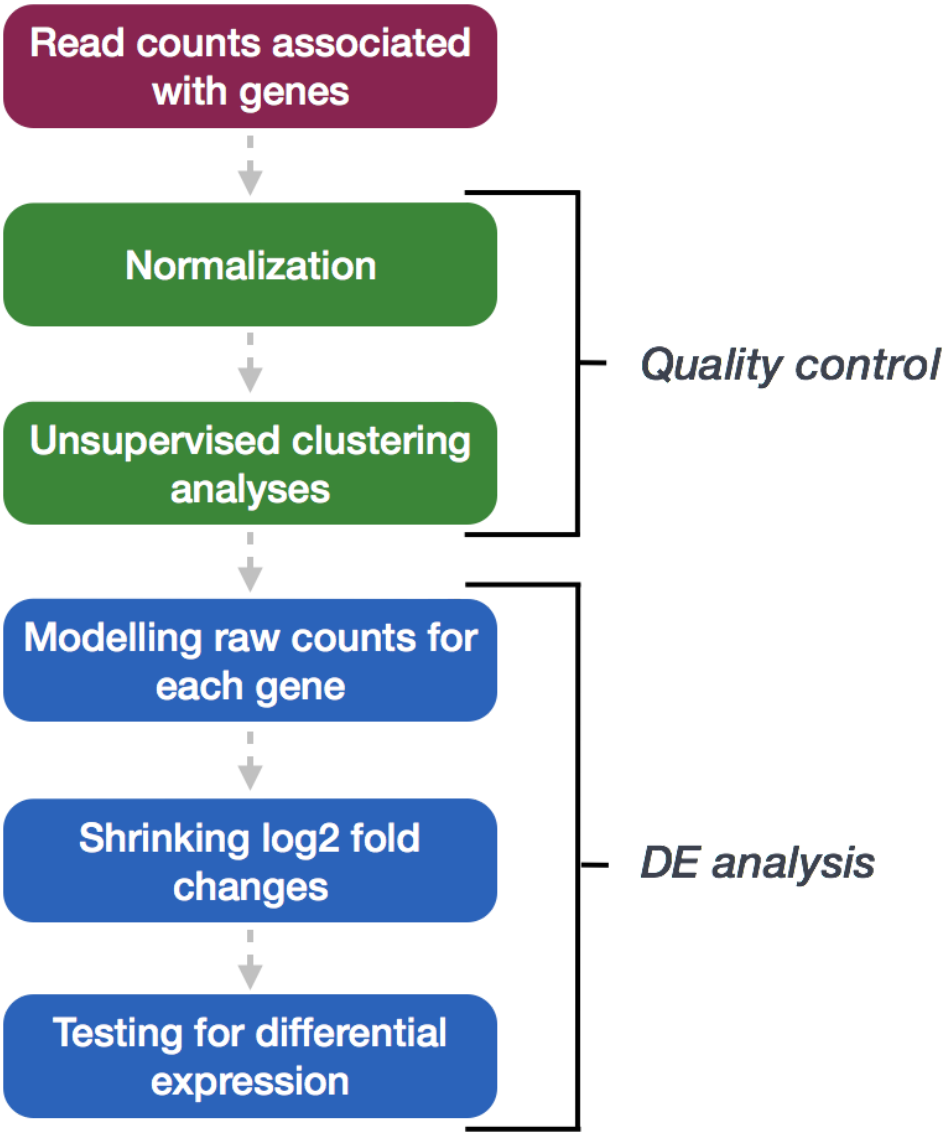
Note that an **increase in the number of replicates tends to return more DE genes than increasing the sequencing depth**. Therefore, generally more replicates are better than higher sequening depth, with the caveat that higher depth is required for detection of lowly expressed DE genes and for performing isoform-level differential expression. Generally, the minimum sequencing depth recommended is 20-30 million reads per sample.

Differential expression analysis workflow

To model counts appropriately when performing a differential expression analysis, theare are a number of software packages that have been developed for differential expression analysis of RNAseq data. Even as new methods are continously being developed a few tools are generally recommended as best practice, e.g. **DESeq2** and **EdgeR**. Both these tools use the negative binomial model, employ similar methods, and typically yield similar results. They are pretty stringent, and have a good balance between sensitivity and specificity (reducing both false positives and false negatives).

Limma-Voom is another set of tools often used for DE analysis, but this method may be less sensitive for small sample sizes. This method is recommended when the number of biological replicates per group grows large (>20).

The DESeq2 workflow first normalizes the count data to account for differences in library sizes and RNA composition between samples. Then, we will use the normalized counts to make some plots for QC at the gene and sample level. The final step is to use the appropriate functions from DESeq2 package to perform the differential expression analysis.



Exploratory Analyses

They are meant to familiarize yourself with the data at hand and to discover biases and unexpected variability.

- Do the experimental groups separate from each other?
- Is there a confounding factor (e.g. batch effect) that should be taken into account in the statistical analysis?
- Are there sample outliers that should be removed?

Typical exploratory analyses:

- corrleation of gene expression between different samples
- hierachical clustering
- dimensionality reduction (e.g. PCA)
- dot plots/box plots of individual genes

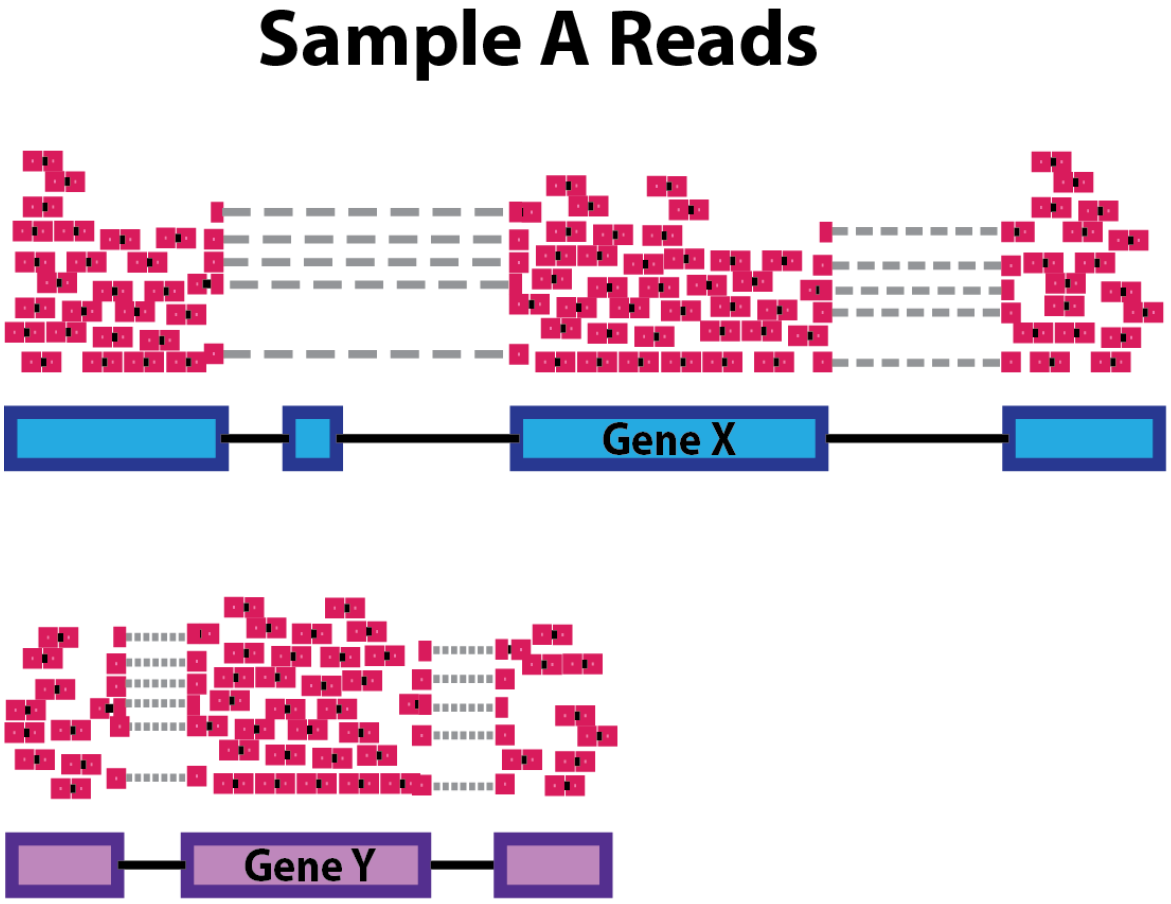
Normalizing and Transforming Read Counts

The first step in the DE workflow is count normalization, which is neccessary to make accurate comparisons of gene expression between samples.

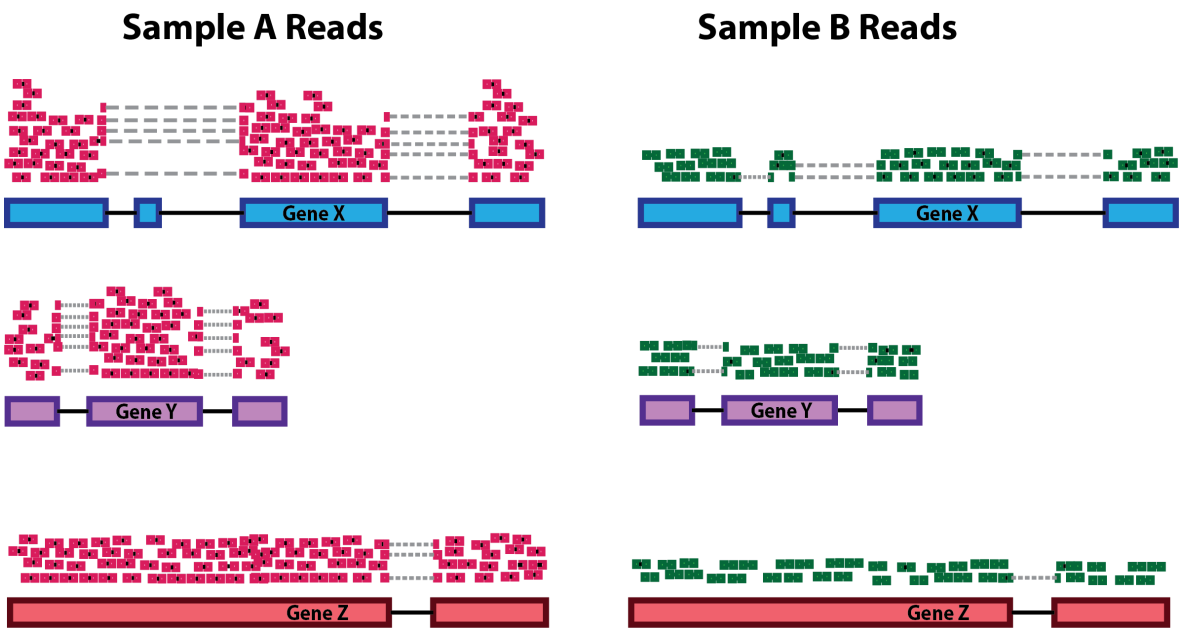
Normalization is the process of scaling raw count values to account for the "uninteresting" factors.

The numbers (or estimates) of reads overlapping with a given gene cannot be directly interpreted as absolute proxies of individual gene expression levels. The value that is obtained for a single gene in a single sample is based on the number of reads corresponding to that gene (or transcript), but there are numerous factors that influence the efficiency of amplification and sequencing of DNA framents. For RNAseq it is important to remember, even given a uniform sampling of a diverse transcript pool (creating a random and comprehensive sampling of the original transcript universe), the number of sequenced reads mapped to a gene depends on:

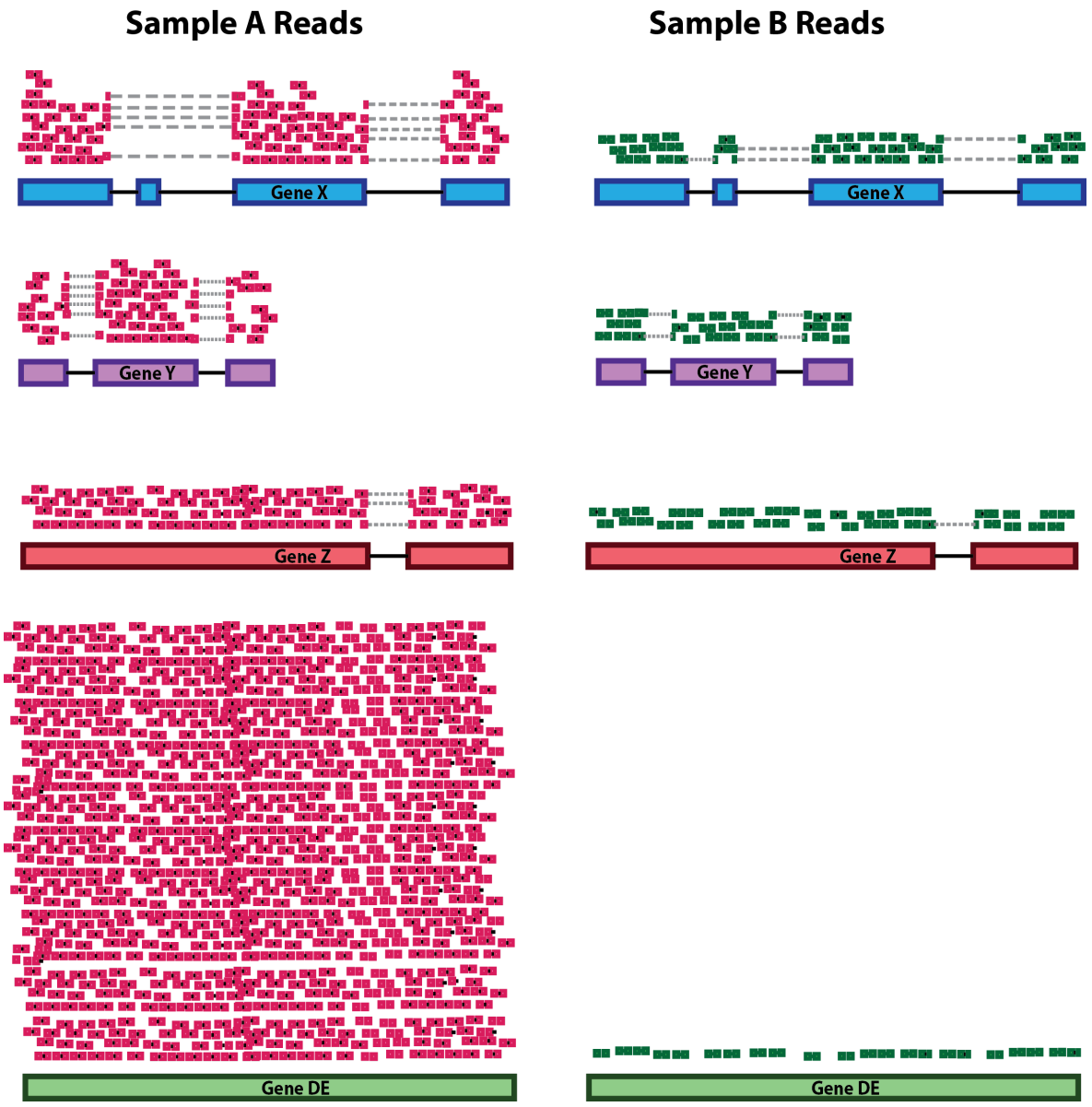
- **own expression level**: the metric of interest
- **Gene length**: Accounting for gene length is necessary for comparing expression between different genes within the sample. In the example *Gene X* and *Gene Y* have similar levels of expression, but the number of reads mapped to *Gene X* would be many more than the number mapped to *Gene Y* because *Gene X* is longer.



- **Sequencing depth**: Accounting for sequencing depth is necessary for comparison of gene expression between samples. In the example below, each gene appears to have doubled in expression, however this is a consequence of *Sample A* having double the sequencing depth.



- **RNA composition**: A few highly differentially expressed genes between samples, differences in the number of genes expressed between samples, or presence of contamination can skew some types of normalization methods. Accounting for RNA composition is recommended for accurate comparison of expression between samples, and is particularly important when performing differential expression analyses. In the example, if we were to divide each sample by the total number of counts to normalize, the counts would be greatly skewed by the DE gene, which takes up most of the counts for *Sample A*, but not *Sample B*. Most other genes for *Sample A* would be divided by the larger number of total counts and appear to be less expressed than those same genes in *Sample B*.



- the expression of all the other genes within the sample

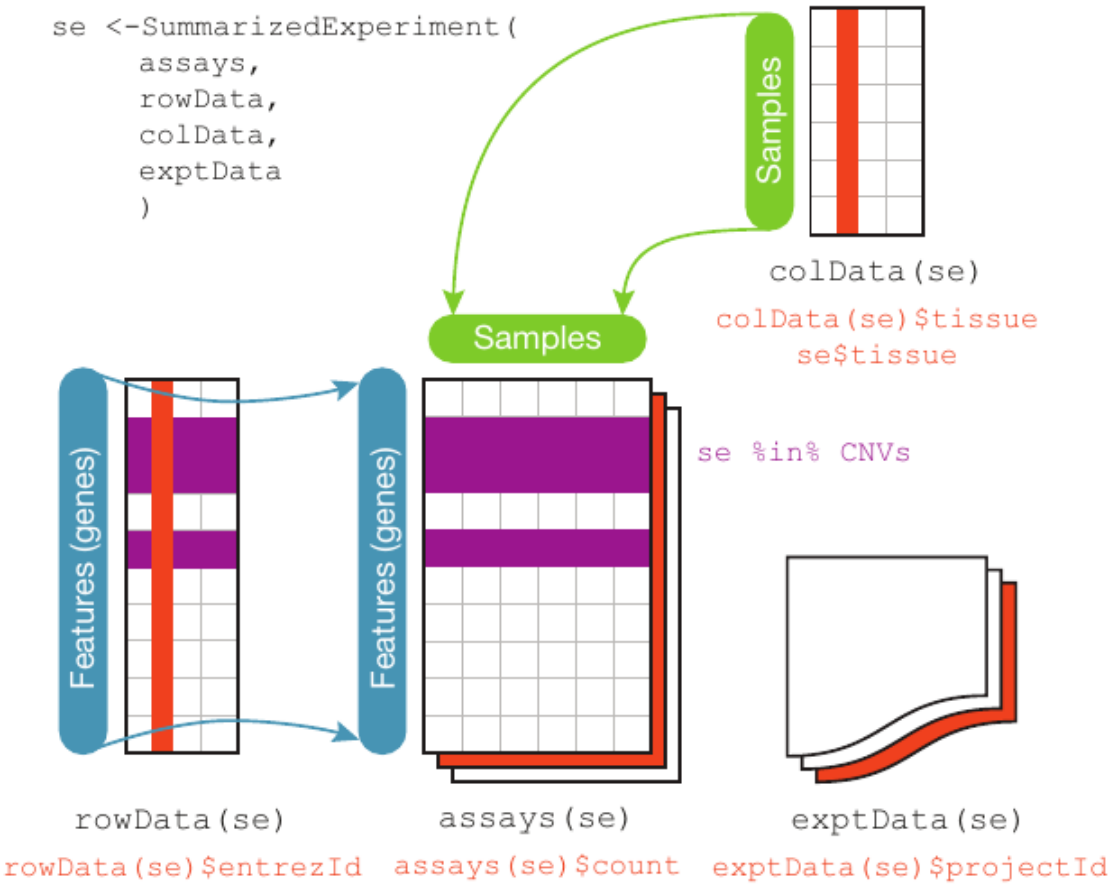
In order to compare the gene expression *between two conditions*, we must therefore calculate the fraction of reads assigned to each gene *relative* to the total number of reads and with respect to the entire RNA repertoire, which may vary drastically from sample to sample. While the number of sequenced reads is known, the total RNA library and its complexity is unknown and variation between samples may be due to contamination as well as biological reasons. The purpose of normalization is to eliminate systematic effects that are not associated with the biological differences of interest.

Normalization for sequencing depth differences with DESeq2

As input, the count-based statistical method expect data in form of a matrix of integer values. The value in the i-th row and the j-th column of the matrix tells how many reads (or fragments) have been assigned to gene i in sample j.

DESeq2 stores virtually all information associated with your experiment in one specific R object of the class DESeqDataSet. The DESeqDataSet is a slightly modified version of the **SummarizedExperiment** class. The **SummarizedExperiment** class was developed to enable the storage of both numeric matrices (e.g. raw read counts) together with plenty of metadata (e.g. conditions of every sample), which is a typically requirement of biological experiments. In short, these sophisticated objects can be thought of as containers where rows represent *features of interest* (e.g. genes, transcripts, exons) and columns represent *samples*. The data corresponding to features can be accessed with the **rowData(SummExpObject)** function, while the meta-data corresponding to the samples (columns) can be accessed via **colData(SummExpObject)**. The actual count data (and all additional numeric values) is stored in the **assay(SummObject)** slot.

- **colData** is a data.frame that can contain all the variables you know about your sample, such as experimental condition, the type and date of sequencing and so on. Its row.names should correspond to the *unique* sample names.
- **rowData** is meant to keep all the information about the genes, e.g. gene ID's, their genomic ranges etc.
- **assay** should contain a matrix of the actual values associated with the genes and samples. For DESeqDataSets, there is an additional specialized function just meant to return the raw counts (counts(DESeqDataSet)) will return the same as assay(DESeqDataSet, "counts")).



DESeq2 with Galaxy data of Colo cellline

In [1]:

```
# open R console or RStudio
library(magrittr) # this allows us to string commands together using %>%

# get the table of read counts by indicating the path to the file
readcounts <- read.table("/mnt/data/Christiane_RNA/bam_files/Colo/featureC
                         header = TRUE)
# show the first lines of the table
head(readcounts)
```

A data.frame: 6 × 14

Geneid	Chr	
<fct>	<fct>	
DDX11L1	chr1;chr1;chr1;chr15;chr15;chr15	11874;12613;13221;102516762
WASH7P	chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1;chr1	14362;14970;15796;16607;16858;17233;17606;17
FAM138F	chr1;chr1;chr1;chr19;chr19;chr19	34611;35277;357
FAM138A	chr1;chr1;chr1;chr19;chr19;chr19	34611;35277;357
OR4F5	chr1	
LOC729737	chr1;chr1;chr1	

In [3]:

```
# show the end of the table
tail(readcounts)
```

A data.frame: 6 × 14

	Geneid	Chr	
	<fct>	<fct>	
23363	TTTY3	chrY;chrY;chrY;chrY;chrY;chrY	
23364	CDY1B	chrY;chrY;chrY;chrY;chrY;chrY	
23365	CDY1	chrY;chrY;chrY;chrY;chrY;chrY	
23366	CSPG4P1Y	chrY;chrY;chrY;chrY	
23367	GOLGA2P2Y	chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY	263
23368	GOLGA2P3Y	chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY;chrY	263

In [2]:

```
# display the internal structure of the table object
str(readcounts)
```

```
'data.frame':   23368 obs. of  14 variables:
 $ Geneid                        : Factor w/ 23368 levels "1/2-SBSRNA4",..: 4
669 22264 5911 5906 14947 11257 10180 10171 10204 14941 ...
 $ Chr                           : Factor w/ 2320 levels "chr1","chr1;chr
1",..: 121 11 122 122 1 3 123 123 120 124 ...
 $ Start                         : Factor w/ 23205 levels "100000708;10000070
8;100000708;100008378;100008378;100008378;100013074;100013074;100013074;10001
3823;100013823;1"| __truncated__,..: 2238 4700 12083 12083 19253 3906 11596 1
1596 11597 12536 ...
 $ End                           : Factor w/ 23211 levels "100000849;10000084
9;100000849;100008468;100008468;100008468;100013125;100013125;100013125;10001
3839;100013839;1"| __truncated__,..: 2606 5046 12192 12192 19412 4333 11603 1
1603 11602 12556 ...
 $ Strand                        : Factor w/ 478 levels "-","-;-","-;-;-",..:
242 11 6 6 234 3 252 252 248 237 ...
 $ Length                        : int  3302 1769 2260 2260 918 5474 8740 874
0 12819 2817 ...
 $ ..bam_files.Colo.ColoP14_5.bam: int  0 0 0 0 0 4 0 0 0 0 ...
 $ ..bam_files.Colo.ColoP14_6.bam: int  0 0 0 0 0 11 0 0 0 0 ...
 $ ..bam_files.Colo.ColoP28_2.bam: int  0 0 0 0 0 0 0 0 0 0 ...
 $ ..bam_files.Colo.ColoP28_3.bam: int  0 0 0 0 0 8 0 0 0 0 ...
 $ ..bam_files.Colo.ColoR34_3.bam: int  0 0 0 0 0 19 0 0 0 0 ...
 $ ..bam_files.Colo.ColoR34_5.bam: int  0 0 0 0 0 9 0 0 0 0 ...
 $ ..bam_files.Colo.ColoR37_1.bam: int  0 0 0 0 0 1 0 0 0 0 ...
 $ ..bam_files.Colo.ColoR37_5.bam: int  0 1 0 0 0 24 0 0 0 0 ...
```

We have a table of 23368 rows (genes) and 14 columns, where as the last 8 columns correspond to the counts of our 8 samples.

The next step is to create the right format for DESeq2, the assay matrix only needs the values associated with the genes and samples.

In [4]:

```
# one of the requirements of the assay() slots is that the row.names corre
# the gene ID's and the col.names to the sample names
row.names(readcounts) <- readcounts$Geneid

# in addition, we need to exclude all columns that do not contain read cou
readcounts <- readcounts[, -c(1:6)] # deletes the first 6 columns

# give meaningful sample names
orig_names <- names(readcounts)
names(readcounts) <- c("ColoP14_5", "ColoP14_6", "ColoP28_2", "ColoP28_3",
                       "ColoR34_5", "ColoR37_1", "ColoR37_5")

# check the data after manipulation
str(readcounts)
```

```
'data.frame':   23368 obs. of  8 variables:
 $ ColoP14_5: int  0 0 0 0 0 4 0 0 0 0 ...
 $ ColoP14_6: int  0 0 0 0 0 11 0 0 0 0 ...
 $ ColoP28_2: int  0 0 0 0 0 0 0 0 0 0 ...
 $ ColoP28_3: int  0 0 0 0 0 8 0 0 0 0 ...
 $ ColoR34_3: int  0 0 0 0 0 19 0 0 0 0 ...
 $ ColoR34_5: int  0 0 0 0 0 9 0 0 0 0 ...
 $ ColoR37_1: int  0 0 0 0 0 1 0 0 0 0 ...
 $ ColoR37_5: int  0 1 0 0 0 24 0 0 0 0 ...
```

In [5]:
```
1 head(readcounts, n=3)
```

A data.frame: 3 × 8

	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37_1	ColoR37_5
	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
DDX11L1	0	0	0	0	0	0	0	0
WASH7P	0	0	0	0	0	0	0	1
FAM138F	0	0	0	0	0	0	0	0

In this count matrix, each row represents a gene, each column a sequenced RNA library, and the values give the raw numbers of fragments that were uniquely assigned to the respective gene in each library. We also have information on each of the samples (the columns of the count matrix).

Now that we have the read counts, we also need some information about the samples, which will be stored in **colData**. As described above, this should be a **data.frame**, where the rows match the column names of the count data we just generated. In addition each row should contain information about the condition of each sample (here: ColoP Parental and ColoR Resistent)

In [6]:
```
1 sample_info <- data.frame(condition = gsub("[0-9]*_[0-9]+", "", names(read
2                            row.names = names(readcounts))
3 sample_info
```

A data.frame: 8 × 1

	condition
	<fct>
ColoP14_5	ColoP
ColoP14_6	ColoP
ColoP28_2	ColoP
ColoP28_3	ColoP
ColoR34_3	ColoR
ColoR34_5	ColoR
ColoR37_1	ColoR
ColoR37_5	ColoR

We now have all the ingredients to prepare our data object in a form that is suitable for DESeq2 analysis, namely:

- readcounts: a table with the fragment counts
- sample_info: a table with the information about the samples

To construct the *DESeqDataSet* object from the matrix of counts and the sample information table, we use:

In [7]:

```r
# generate the DESeqDataSet
library(DESeq2)
DESeq.ds <- DESeqDataSetFromMatrix(countData = readcounts,
                                   colData = sample_info,
                                   design = ~ condition)

# check the result using the accessors
colData(DESeq.ds) %>% head()
```

```
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’:

    IQR, mad, sd, var, xtabs

The following objects are masked from ‘package:base’:

    anyDuplicated, append, as.data.frame, basename, cbind, colnames,
    dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
    grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
    order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
    rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
    union, unique, unsplit, which, which.max, which.min


Attaching package: ‘S4Vectors’

The following object is masked from ‘package:base’:

    expand.grid

Loading required package: IRanges
Loading required package: GenomicRanges
Loading required package: GenomeInfoDb
Loading required package: SummarizedExperiment
Loading required package: Biobase
Welcome to Bioconductor

    Vignettes contain introductory material; view with
    'browseVignettes()'. To cite Bioconductor, see
    'citation("Biobase")', and for packages 'citation("pkgname")'.

Loading required package: DelayedArray
Loading required package: matrixStats

Attaching package: ‘matrixStats’

The following objects are masked from ‘package:Biobase’:

    anyMissing, rowMedians

Loading required package: BiocParallel

Attaching package: ‘DelayedArray’

The following objects are masked from ‘package:matrixStats’:
```

```
    colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

The following objects are masked from ‘package:base’:

    aperm, apply, rowsum


DataFrame with 6 rows and 1 column
          condition
           <factor>
ColoP14_5     ColoP
ColoP14_6     ColoP
ColoP28_2     ColoP
ColoP28_3     ColoP
ColoR34_3     ColoR
ColoR34_5     ColoR
```

In [8]:
```
1 assay(DESeq.ds, "counts") %>% head()
```

A matrix: 6 × 8 of type int

	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37_1	ColoR37_
DDX11L1	0	0	0	0	0	0	0	
WASH7P	0	0	0	0	0	0	0	
FAM138F	0	0	0	0	0	0	0	
FAM138A	0	0	0	0	0	0	0	
OR4F5	0	0	0	0	0	0	0	
LOC729737	4	11	0	8	19	9	1	2

In [9]:
```
1 rowData(DESeq.ds) %>% head()
```

```
DataFrame with 6 rows and 0 columns
```

In [10]:
```
1 counts(DESeq.ds) %>% str
```

```
 int [1:23368, 1:8] 0 0 0 0 0 4 0 0 0 0 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:23368] "DDX11L1" "WASH7P" "FAM138F" "FAM138A" ...
  ..$ : chr [1:8] "ColoP14_5" "ColoP14_6" "ColoP28_2" "ColoP28_3" ...
```

Exploratory analysis and visualization

A useful initial step in an RNAseq analysis is often to assess overall similarity between samples:

- Which samples are similar to each other, which are different?
- Does this fit the expectation from the experiment design?
- What are the major sources of variation in the dataset?

To explore the similarity of our samples, we will be performing sample-level QC using Principal Component Analysis (PCA) and hierachical clustering methods. Our sample-level QC allows us to see how well our replicates cluster together, as well as, observe whether our experimental condition represents the major source of variation in the data. Performing sample-level QC can also identify any sample outliers, which may need to be explored further to determine whether they need to be removed prior to DE analysis.

There are two separate paths in this workflow; the one we will see first involves *transformation of the counts* in order to visually explore sample relationships. When using unsupervised clustering methods, log2-transformation of the normalized counts improves the distances/clustering for visualization. DESeq2

uses a **regularized log transform** (rlog) of the normalized counts for sample-level QC as it moderates the variance across the mean, improving the clustering. In the second part, we will go back to the original raw counts for *statistical testing*. This is critical because the statistical testing methods rely on original count data (not scaled or transformed) for caluclating the precision of measurements.

Pre-filtering the dataset

Our count matrix with our *DESeqDataSet* contains many rows with only zeros, and additionally many rows with only a few fragments total. In order to reduce the size of the object, and to increase the speed of our functions, we can remove the rows that have no or nearly no information about the amount of gene expression. Here we apply the most minimal filtering rule: removing rows of the *DESeqDataSet* that have no counts.

In [11]:
```
nrow(DESeq.ds)
```

23368

In [12]:
```
# remove genes without any counts
DESeq.ds <- DESeq.ds[ rowSums(counts(DESeq.ds)) > 0, ]
nrow(DESeq.ds)
```

14980

The variance stabilizing transformation and the rlog

Many common statistical methods for exploratory analysis of multidimensional data, for example *clustering* and *principal components analysis* (PCA), work best for data that generally has the same range of variance at different ranges of the mean values. When the expected amount of variance is approximately the same across different mean values, the data is said to be *homoskedastic*. For RNAseq counts, however, the expected variance grows with the mean. For example, if one performs PCA directly on a matrix of counts or normalized counts (e.g. correcting for differences in sequencing depth), the resulting plot typically depends mostly on the genes with *highest* counts because they show the largest absolute differences between samples. A simple and often used strategy to avoid this is to take the logarithm of the normalized count values plus a pseudocount of 1; however, depending on the choice of the pseudocount, now the genes with the very lowest counts will contribute a great deal of noise to the resulting plot, because taking the logarithm of small counts actually inflates their variance.

The logarithm with a small pseudocount amplifies differences when the values are close to 0. The low count genes with low signal-to-noise ratio will overly contribute to sample-sample distances and PCA plots.

As a solution, DESeq2 offers two transformations for count data that stabilize the variance across the mean: the variance stabilizing transformation (VST) for negative binomial data with a dispersion-mean trend (Anders and Huber 2010), implemented in the vst function, and the regularized-logarithm transformation or rlog (Love, Huber, and Anders 2014).

For genes with high counts, both the VST and the rlog will give similar result to the ordinary log2 transformation of normalized counts. For genes with lower counts, however, the values are shrunken towards a middle value. The VST or rlog-transformed data then become approximately homoskedastic (more flat trend in the meanSdPlot), and can be used directly for computing distances between samples, making PCA plots, or as input to downstream methods which perform best with homoskedastic data.

Which transformation to choose? The VST is much faster to compute and is less sensitive to high count outliers than the rlog. The rlog tends to work well on small datasets (n < 30), potentially outperforming the VST when there is a wide range of sequencing depth across samples (an order of magnitude difference). We therefore recommend the VST for medium-to-large datasets (n > 30). You can perform both transformations and compare the meanSdPlot or PCA plots generated, as described below.

Note that the two transformations offered by DESeq2 are provided for applications other than differential testing. For differential testing we recommend the DESeq function applied to raw counts, as described later in this workflow, which also takes into account the dependence of the variance of counts on the mean value during the dispersion estimation step.

Both vst and rlog return a DESeqTransform object which is based on the SummarizedExperiment class. The transformed values are no longer counts, and are stored in the assay slot. The colData that was attached to DESeq.ds is still accessible:

In [15]:
```
vsd <- vst(DESeq.ds, blind = FALSE)
head(assay(vsd), 3)
```

A matrix: 3 × 8 of type dbl

	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37_1	ColoR
WASH7P	7.504367	7.504367	7.504367	7.504367	7.504367	7.504367	7.504367	7.61
LOC729737	7.728574	7.876834	7.504367	7.774968	7.912350	7.828075	7.609512	8.04
LOC100288069	7.504367	7.504367	7.504367	7.600162	7.943150	7.504367	7.984095	7.83

In [16]:
```
colData(vsd)
```

```
DataFrame with 8 rows and 2 columns
          condition        sizeFactor
           <factor>         <numeric>
ColoP14_5     ColoP 0.910322553300013
ColoP14_6     ColoP 0.903889826097872
ColoP28_2     ColoP   1.0276191726606
ColoP28_3     ColoP  1.24872206240468
ColoR34_3     ColoR  1.29983176603371
ColoR34_5     ColoR 0.980446461529339
ColoR37_1     ColoR   1.0364297205781
ColoR37_5     ColoR 0.934367469873441
```

In [18]:
```
rld <- rlog(DESeq.ds, blind = FALSE)
head(assay(rld), 3)
```

A matrix: 3 × 8 of type dbl

	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37_1	ColoR
WASH7P	-2.130040	-2.130021	-2.130378	-2.130920	-2.131032	-2.130247	-2.130402	-2.12
LOC729737	3.010995	3.156271	2.914121	3.050456	3.198882	3.103493	2.936189	3.36
LOC100288069	2.275419	2.275482	2.274396	2.292073	2.602775	2.274780	2.652204	2.47

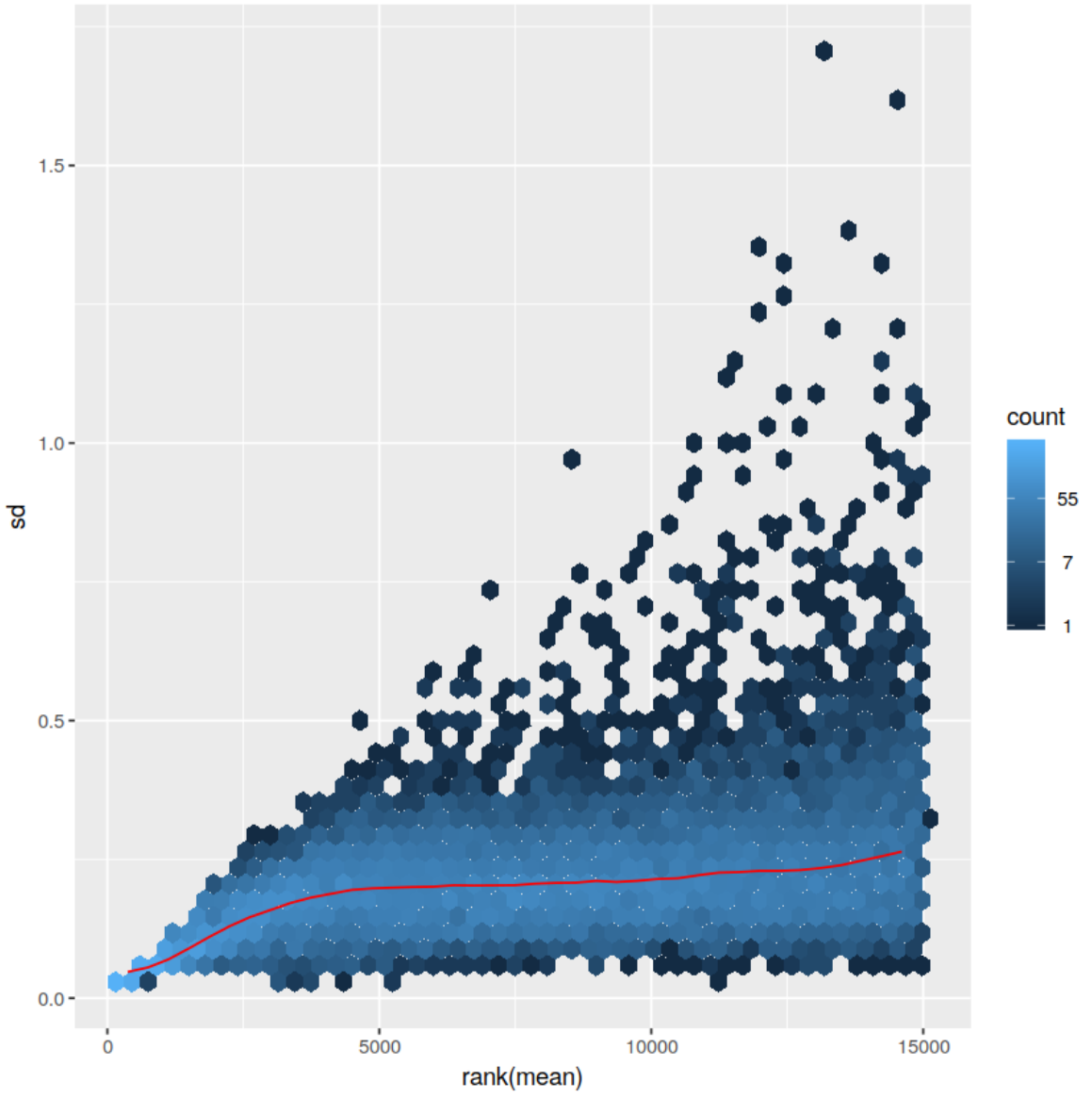
Effects of transformation on the variance

The figures below plot the standard deviation of the transformed data, across samples, against the mean, using the variance stabilizing and the regularized log transformation.

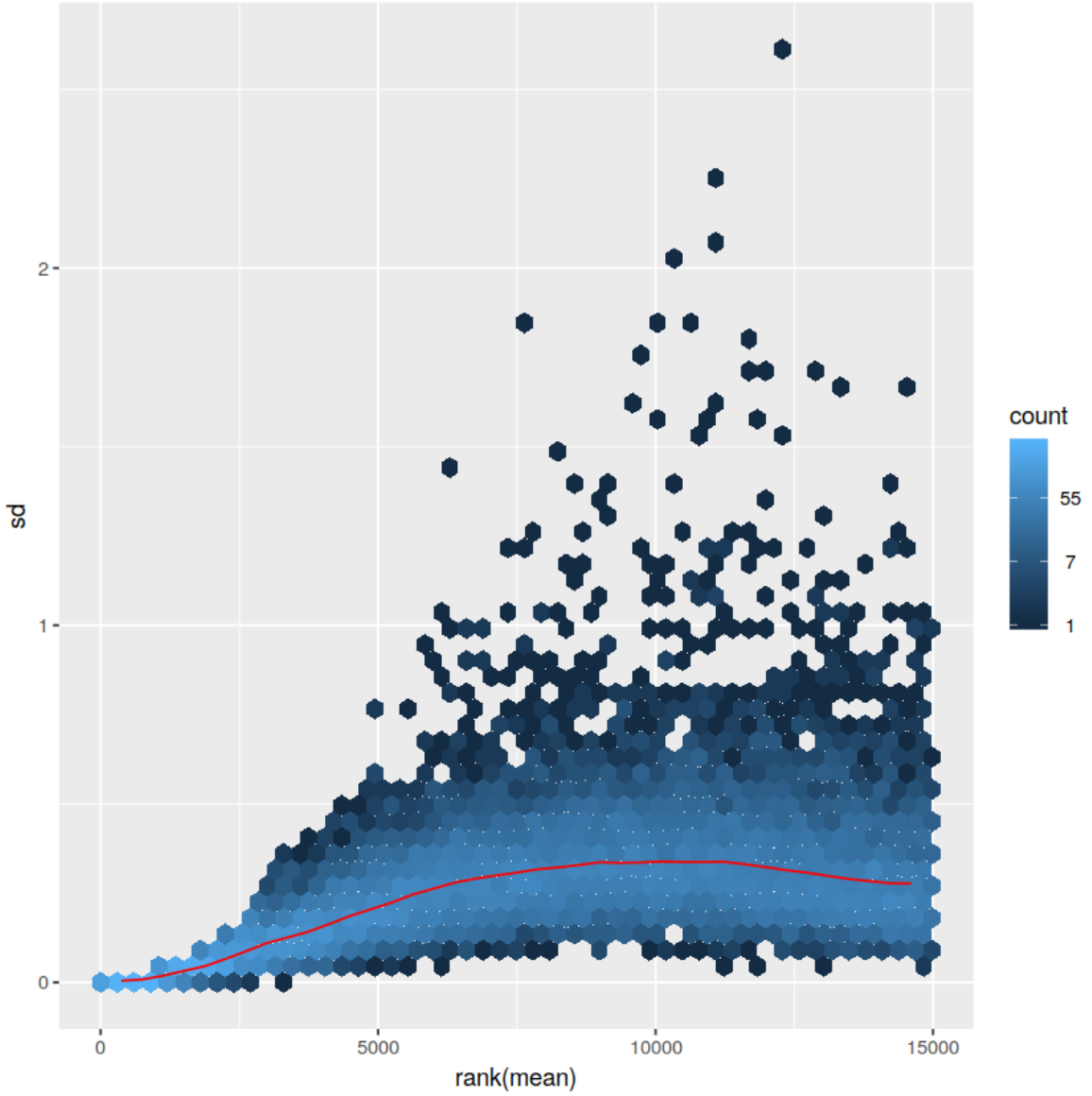
Note that the vertical axis in such plots is the square root of the variance over all samples, so including the variance due to experimental conditions. While a flat curve of the square root of varinace over the mean may seem the goal of such transformation, this may be unreasonable in the case of datasets with many true differences due to the experimental conditions.

In [20]:

```
1 library("vsn")
2 meanSdPlot(assay(vsd))
```



In [21]:
```
1 meanSdPlot(assay(rld))
```



In the above function calls, we specified blind = FALSE, which means that differences between cell lines and treatment (the variables in the design) will not contribute to the expected variance-mean trend of the experiment. The experimental design is not used directly in the transformation, only in estimating the global amount of variability in the counts. For a fully unsupervised transformation, one can set blind = TRUE (which is the default).

To show the effect of the transformation, in the figure below we plot the first sample against the second, first simply using the log2 function (after adding 1, to avoid taking the log of zero), and then using the VST and rlog-transformed values. For the log2 approach, we need to first estimate size factors to account for sequencing depth, and then specify normalized=TRUE. Sequencing depth correction is done automatically for the vst and rlog.

DESeq2's default method to normalize read counts to account for differences in sequencing depths is implemented in estimateSizeFactors()

In [23]:
```
# calculate the size factor and add it to the data set
DESeq.ds <- estimateSizeFactors(DESeq.ds)
sizeFactors(DESeq.ds)
```

ColoP14_5	0.910322553300013
ColoP14_6	0.903889826097872
ColoP28_2	1.0276191726606
ColoP28_3	1.24872206240468
ColoR34_3	1.29983176603371
ColoR34_5	0.980446461529339
ColoR37_1	1.0364297205781
ColoR37_5	0.934367469873441

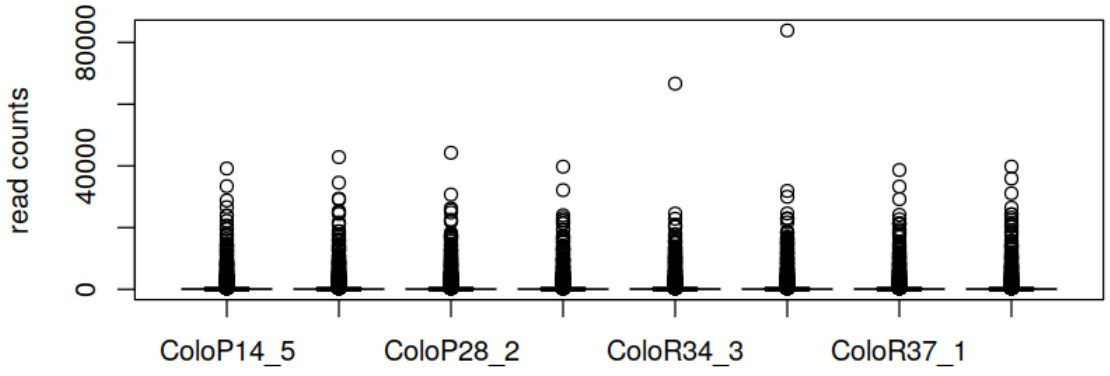
In [24]:
```
# if you check colData() again, you see that this now contains the sizeFac
colData(DESeq.ds)
```

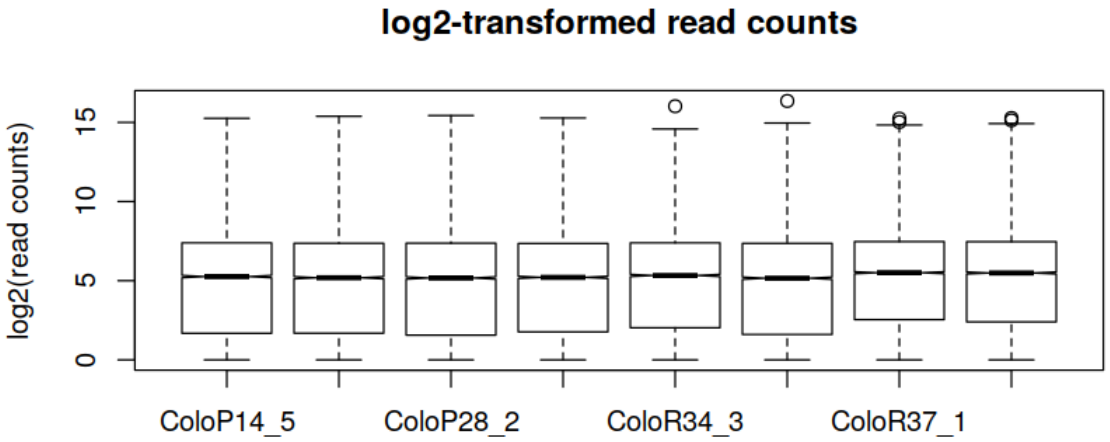
```
DataFrame with 8 rows and 2 columns
          condition        sizeFactor
           <factor>         <numeric>
ColoP14_5     ColoP 0.910322553300013
ColoP14_6     ColoP 0.903889826097872
ColoP28_2     ColoP   1.0276191726606
ColoP28_3     ColoP  1.24872206240468
ColoR34_3     ColoR  1.29983176603371
ColoR34_5     ColoR 0.980446461529339
ColoR37_1     ColoR   1.0364297205781
ColoR37_5     ColoR 0.934367469873441
```

In [26]:
```
# counts() allows you to immediately retrieve the _normalized_ read counts
counts.sf_normalized <- counts(DESeq.ds, normalized = TRUE)
```

In [27]:

```r
#transform size-factor normalized read counts to log2 scale using a pseudo
log.norm.counts <- log2(counts.sf_normalized + 1)

par(mfrow=c(2,1)) # to plot the following two images underneath each other

# first, boxplot of non-transformed read counts (one per sample)
boxplot(counts.sf_normalized, notch = TRUE,
        manin = "untransformed read counts", ylab = "read counts")

# box plots of log2-transformed read counts
boxplot(log.norm.counts, notch = TRUE,
        main = "log2-transformed read counts",
        ylab = "log2(read counts)")
```





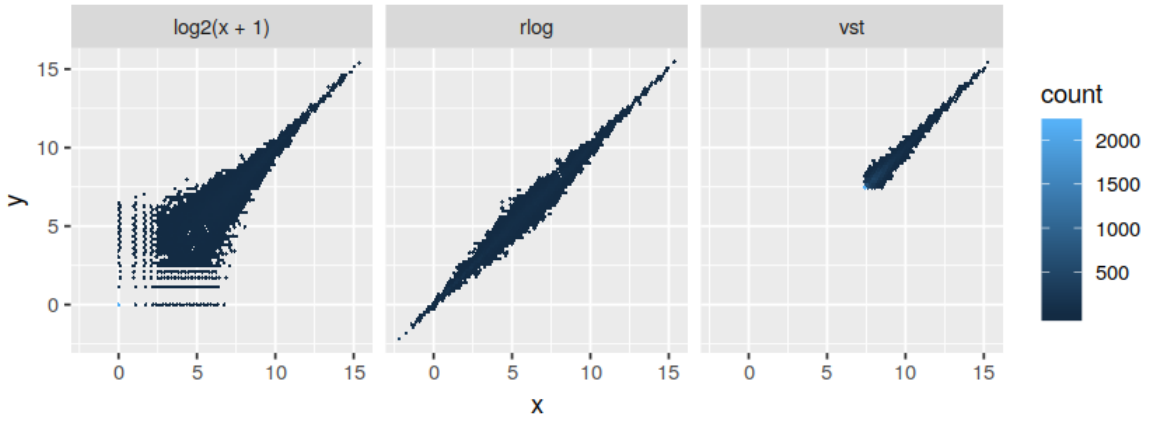
Scatterplot of transformed counts from two samples. To get an impression of how similar read counts are between replicates, it is often insightful to simply plot hte counts in a parwise manner. Shown below are scatterplots using the log2 transform of normalized counts (left), using the VST (middle), and using the rlog (right). While the rlog is on roughly the same scale as the log2 counts, the VST has a upward shift for the smaller values. It is the differences between samples (deviation from y=x in these scatterplots) which will contribute to the distance calculations and the PCA plot.

We can see how genes with low counts (bottom left-hand corner) seem to be excessively variable on the ordinary logarithmic scale, while the VST and rlog compress differences for the low count genes for which the data provide little information about differential expression.

In [25]:

```r
library("dplyr")
library("ggplot2")

df <- bind_rows(
  as_data_frame(log2(counts(DESeq.ds, normalized=TRUE)[, 1:2]+1)) %>%
         mutate(transformation = "log2(x + 1)"),
  as_data_frame(assay(vsd)[, 1:2]) %>% mutate(transformation = "vst"),
  as_data_frame(assay(rld)[, 1:2]) %>% mutate(transformation = "rlog"))

colnames(df)[1:2] <- c("x", "y")

ggplot(df, aes(x = x, y = y)) + geom_hex(bins = 80) +
  coord_fixed() + facet_grid( . ~ transformation)
```

```
Attaching package: ‘dplyr’

The following object is masked from ‘package:matrixStats’:

    count

The following object is masked from ‘package:Biobase’:

    combine

The following objects are masked from ‘package:GenomicRanges’:

    intersect, setdiff, union

The following object is masked from ‘package:GenomeInfoDb’:

    intersect

The following objects are masked from ‘package:IRanges’:

    collapse, desc, intersect, setdiff, slice, union

The following objects are masked from ‘package:S4Vectors’:

    first, intersect, rename, setdiff, setequal, union

The following objects are masked from ‘package:BiocGenerics’:

    combine, intersect, setdiff, union

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

Warning message:
“`as_data_frame()` is deprecated, use `as_tibble()` (but mind the new semanti
cs).
This warning is displayed once per session.”
```



Sample distances

A useful first step in an RNA-seq analysis is often to assess overall similarity between samples: Which samples are similar to each other, which are different? Does this fit to the expectation from the experiment's design?

We use the R function dist to calculate the Euclidean distance between samples. To ensure we have a roughly equal contribution from all genes, we use it on the VST data. We need to transpose the matrix of values using t, because the dist function expects the different samples to be rows of its argument, and different dimensions (here, genes) to be columns.

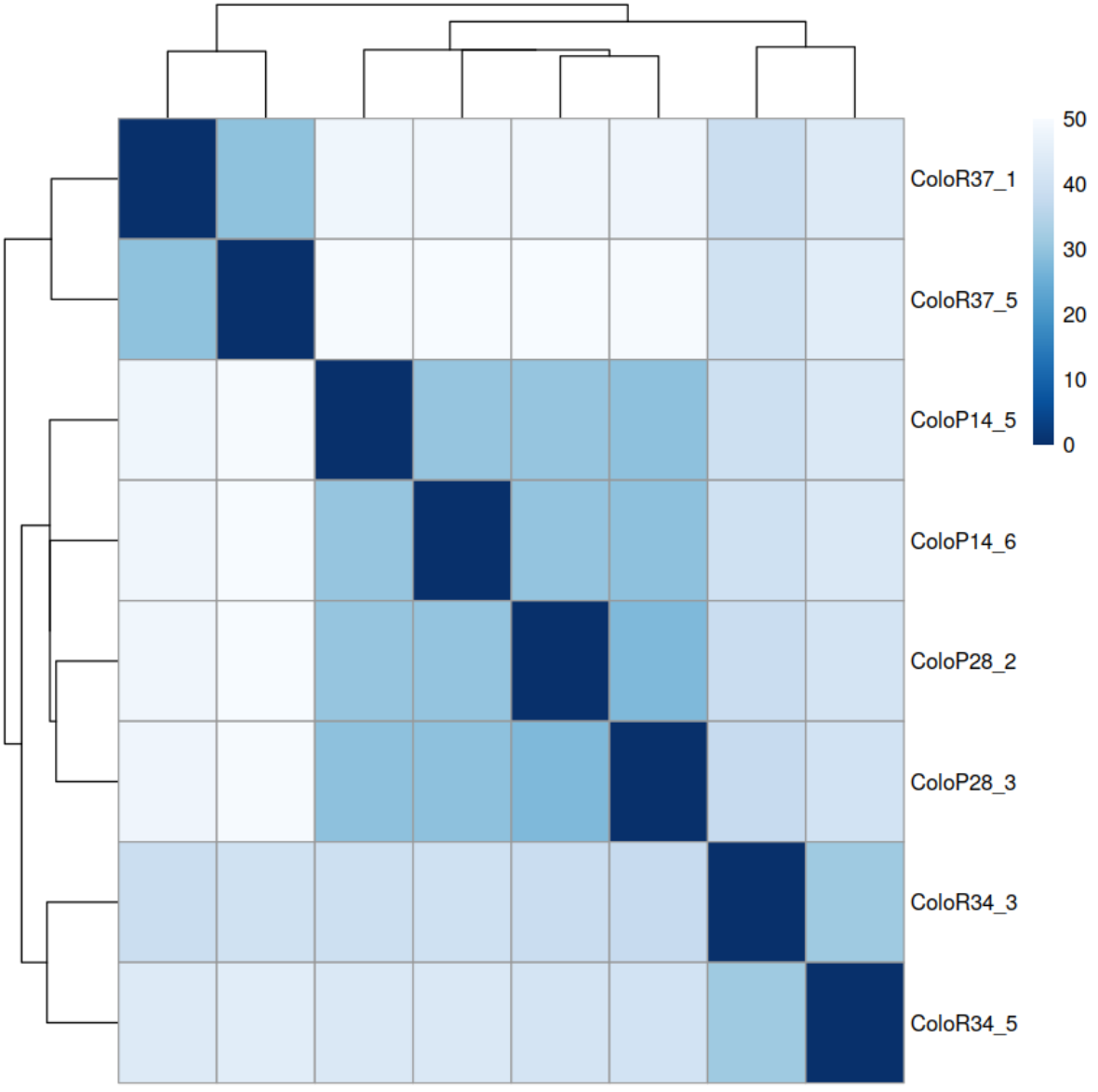
In [28]:
```
1 sampleDists <- dist(t(assay(vsd)))
2 sampleDists
```

```
          ColoP14_5 ColoP14_6 ColoP28_2 ColoP28_3 ColoR34_3 ColoR34_5 ColoR37
_1
ColoP14_6  30.22550
ColoP28_2  30.22813  30.18235
ColoP28_3  29.41748  29.36000  27.47038
ColoR34_3  39.46469  39.90620  38.49951  37.76663
ColoR34_5  42.57016  42.60511  41.18644  40.66440  31.46776
ColoR37_1  48.12649  48.40489  48.36155  48.04349  38.87592  43.36602
ColoR37_5  49.80286  50.01183  49.86288  49.63433  40.35677  44.68576  29.531
42
```

We visualize the distances in a heatmap in a figure below, using the function pheatmap from the pheatmap package.

In order to plot the sample distance matrix with the rows/columns arranged by the distances in our distance matrix, we manually provide sampleDists to the clustering_distance argument of the pheatmap function. Otherwise the pheatmap function would assume that the matrix contains the data values themselves, and would calculate distances between the rows/columns of the distance matrix, which is not desired. We also manually specify a blue color palette using the colorRampPalette function from the RColorBrewer package.

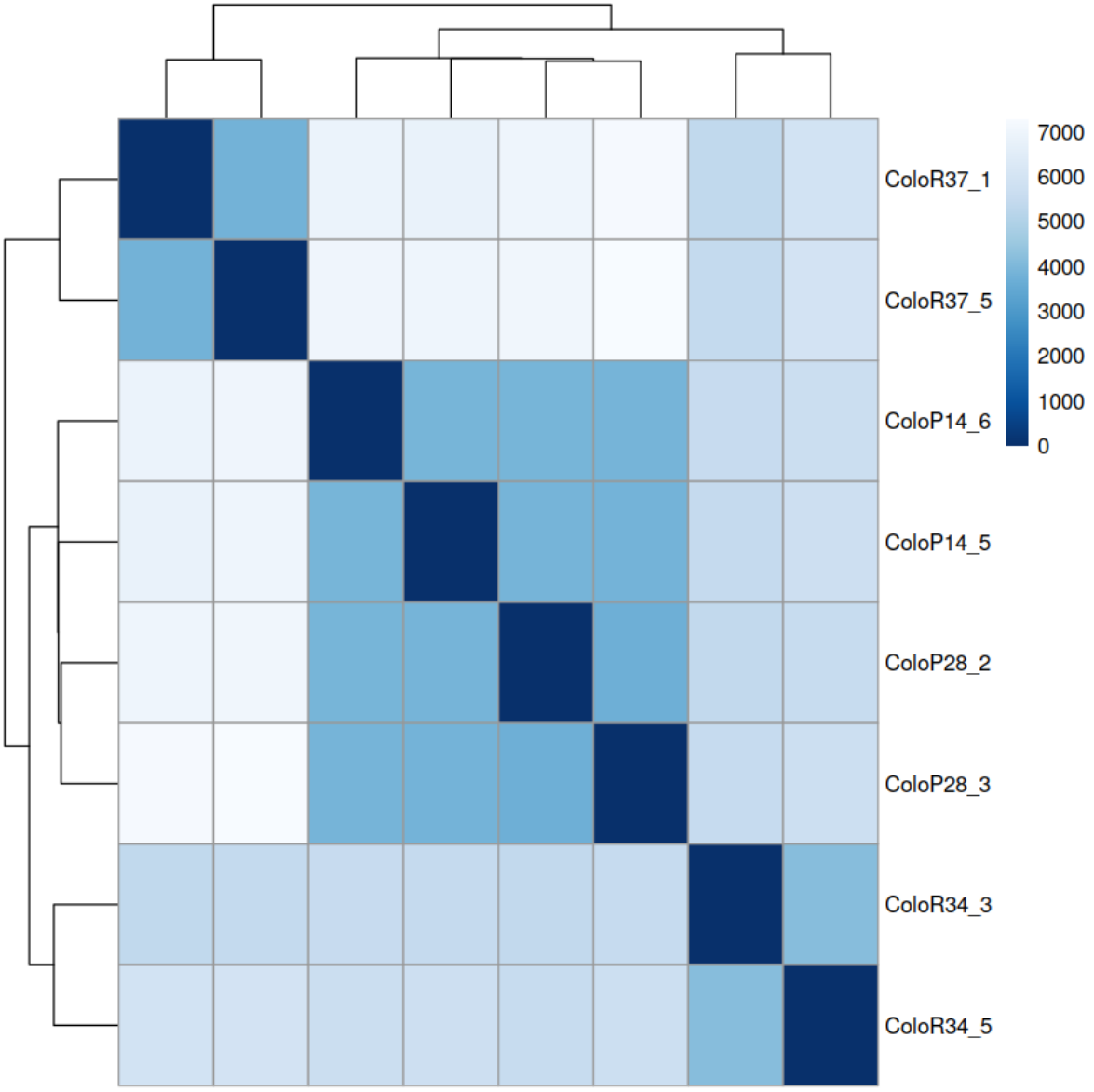
In [31]:
```
library("pheatmap")
library("RColorBrewer")

sampleDistMatrix <- as.matrix( sampleDists )
rownames(sampleDistMatrix) <- paste(names(readcounts))
colnames(sampleDistMatrix) <- NULL
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)
pheatmap(sampleDistMatrix,
         clustering_distance_rows = sampleDists,
         clustering_distance_cols = sampleDists,
         col = colors)
```



Another option for calculating sample distances is to use the Poisson Distance (Witten 2011), implemented in the PoiClaClu package. This measure of dissimilarity between counts also takes the inherent variance structure of counts into consideration when calculating the distances between samples.

The PoissonDistance function takes the original count matrix (not normalized) with samples as rows instead of columns, so we need to transpose the counts in DESeq.ds.

In [33]:
```
library("PoiClaClu")
poisd <- PoissonDistance(t(counts(DESeq.ds)))

samplePoisDistMatrix <- as.matrix( poisd$dd )
rownames(samplePoisDistMatrix) <- paste(names(readcounts))
colnames(samplePoisDistMatrix) <- NULL
pheatmap(samplePoisDistMatrix,
         clustering_distance_rows = poisd$dd,
         clustering_distance_cols = poisd$dd,
         col = colors)
```



Exploring global read count patterns

An important step before diving into the identification of differentially expressed genes is to check whether expectations about basic global patterns are met. For example, technical and biological replicates should show similar expression patterns while the expression patterns of, say, two experimental conditions should be more dissimilar. There are multiple ways to assess the similarity of expression patterns.

Pairwise correlation

The *Pearson correlation coefficient* is a measure of the strenght of the linear relationship between two variables and is often used to assess the similarity of RNAseq samples in a pair-wise fashion. It is defined as the covariance of two variables divided by the product of their standard deviation. The ENCODE consortium recommends that "for mRNA, biological replicates should display > 0.9 correlation for transcripts/features".

Hierachical clustering

To determine whether the different sample types can be separated in an unsupervised fashion (i.e., samples of different conditions are more dissimilar to each other than replicates within the same condition), hierarchical clustering can be used. Hierarchical clustering is typically based on pairwise comparisons of individual samples, which are grouped into “neighborhoods” of similar samples. The basis of hierarchical clustering is therefore a matrix of similarity metrics (which is different from the actual gene expression values!).

Hierarchical clustering requires two decisions:

1. How should the (dis)similarity between pairs be calculated?
2. How should the (dis)similarity be used for the clustering?

A common way to assess the (dis)similarity is the Pearson correlation coefficient, r, that we just described. The corresponding distance measure is $d = 1 - r$. Alternatively, the Euclidean distance is often used as a measure of distance between two vectors of read counts. The Euclidean distance is strongly influenced by differences of the scale: if two samples show large differences in sequencing depth, this will affect the Euclidean distance more than the distance based on the Pearson correlation coefficient.

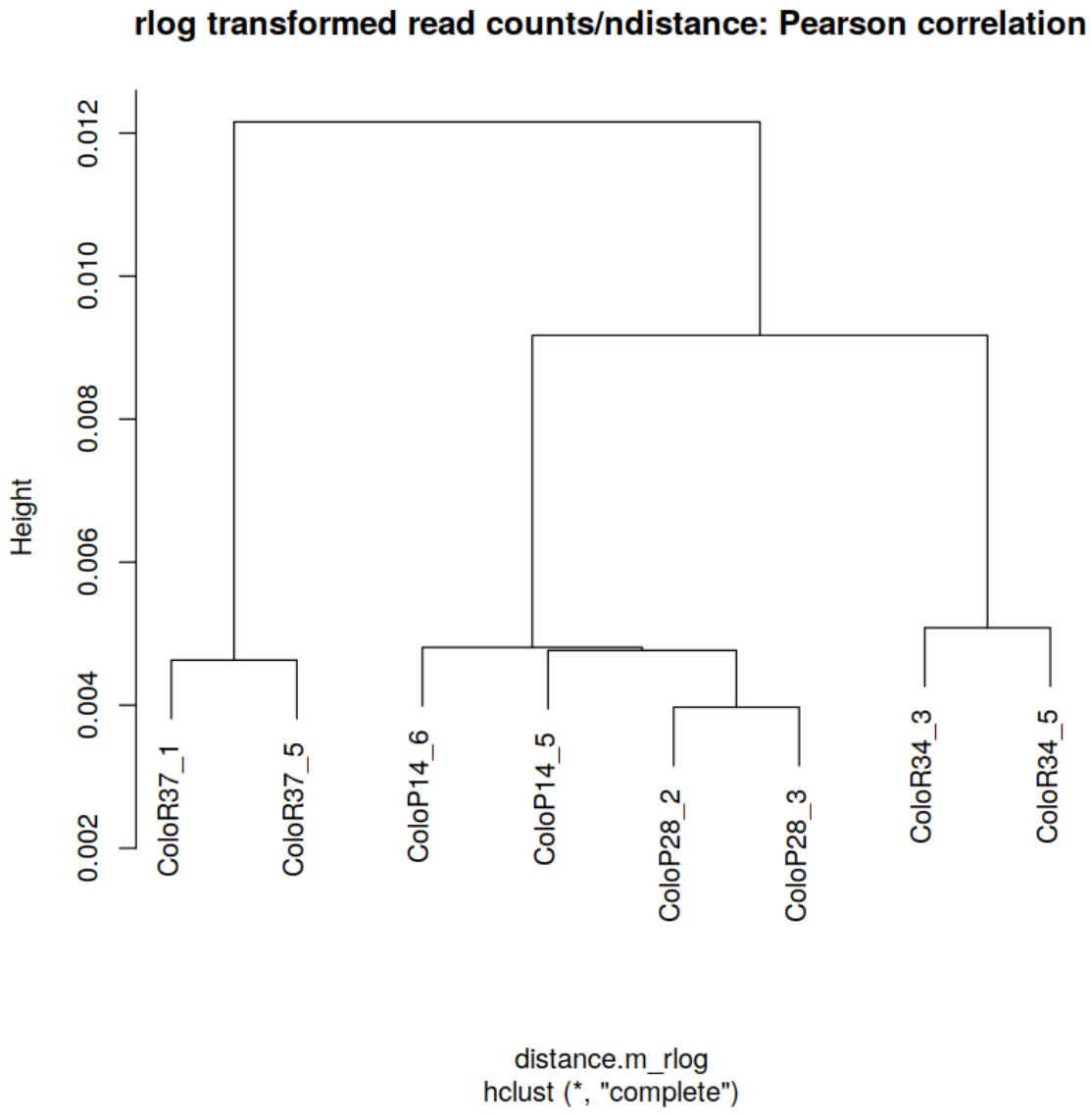
Just like there are numerous ways to calculate the distance, there are multiple options to decide on how the distances should be used to define clusters of samples. The most popular choices for the linkage function are

- complete: intercluster distance ≡ largest distance between any 2 members of either cluster
- average: intercluster distance ≡ average distance between any 2 members
- single: intercluster distance ≡ shortest distance between any 2 members

The result of hierachical clustering is a *dendrogram*. Clusters are obtained by cutting the dendrogram at a level where the jump between two consecutive nodes is large: connected components then form individual clusters.

In [36]:

```r
# in R pairwise correlations can be calculated with the cor() function
# obtain regularized log-transformed values
DESeq.rlog <- rlog(DESeq.ds, blind = TRUE)
rlog.norm.counts <- assay(DESeq.rlog)
distance.m_rlog <- as.dist(1 - cor(rlog.norm.counts, method = "pearson"))

# plot() can directly interpret the output of hclust()
plot(hclust(distance.m_rlog),
      labels = colnames(rlog.norm.counts),
      main = "rlog transformed read counts/ndistance: Pearson correlation"
```



In [309]:

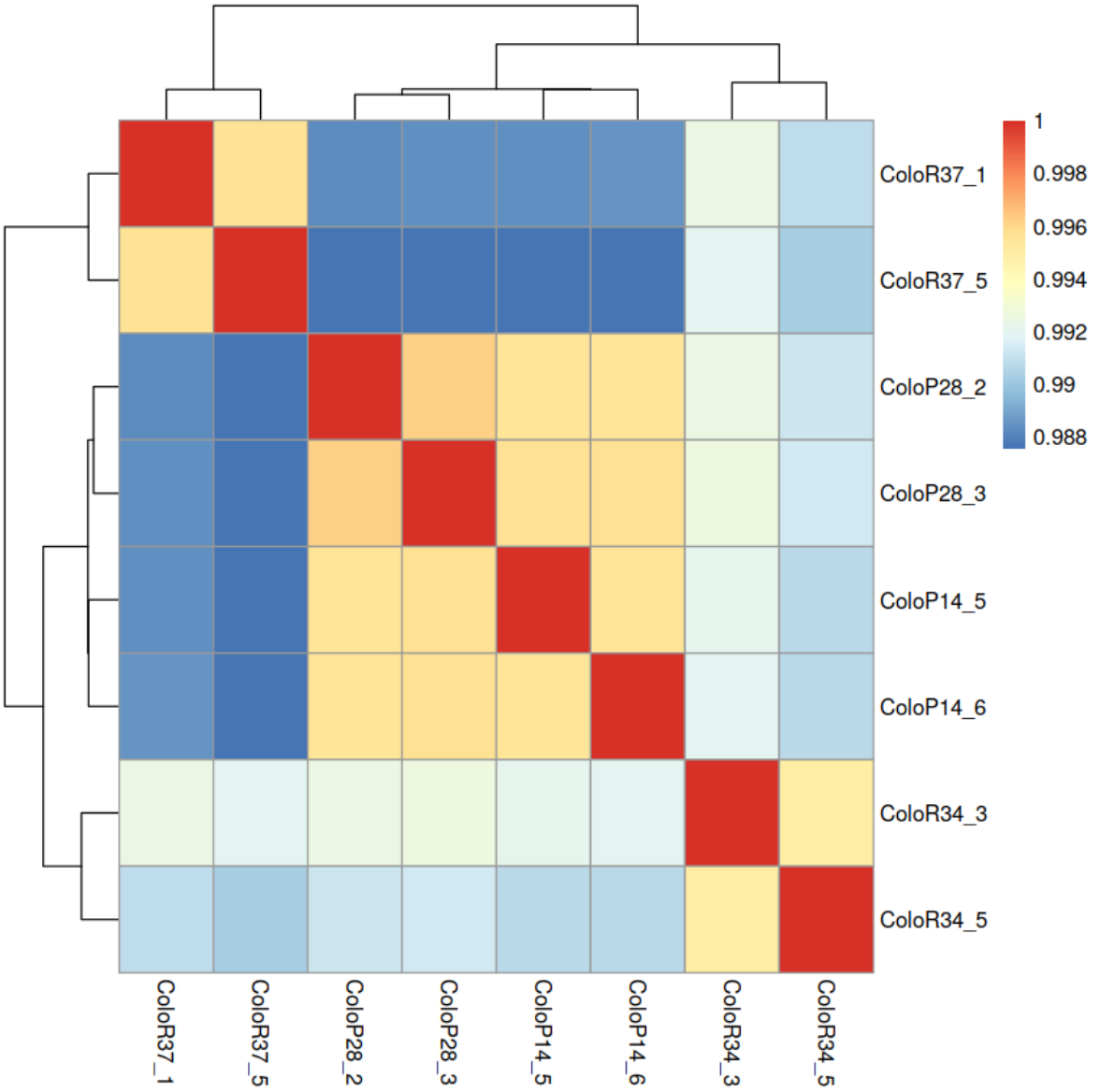
```
# Extract the rlog matrix from the object
rld_mat <- assay(rld)
rld_cor  <- cor(rld_mat) # cor() is a base R function for pairwise correla
head(rld_cor)
```

A matrix: 6 × 8 of type dbl

	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37_1	ColoR37_5
ColoP14_5	1.0000000	0.9954895	0.9954539	0.9957346	0.9920817	0.9907557	0.9884325	0.987649
ColoP14_6	0.9954895	1.0000000	0.9954284	0.9957179	0.9919960	0.9907661	0.9884715	0.987687
ColoP28_2	0.9954539	0.9954284	1.0000000	0.9962319	0.9924166	0.9911870	0.9882939	0.987566
ColoP28_3	0.9957346	0.9957179	0.9962319	1.0000000	0.9926306	0.9913805	0.9883345	0.987569
ColoR34_3	0.9920817	0.9919960	0.9924166	0.9926306	1.0000000	0.9950192	0.9924876	0.991923
ColoR34_5	0.9907557	0.9907661	0.9911870	0.9913805	0.9950192	1.0000000	0.9908071	0.990192

In [310]:

```
# Plot heatmap
pheatmap(rld_cor)
```



PCA plot

A complementary approach to determine whether samples display greater variability between experimental conditions than between replicates of the same treatment is principal components analysis. It is a typical example of dimensionality reduction approaches that have become very popular in the field of machine learning. The goal is to find groups of features (e.g., genes) that have something in common (e.g., certain patterns of expression across different samples), so that the information from thousands of features is captured and represented by a reduced number of groups.

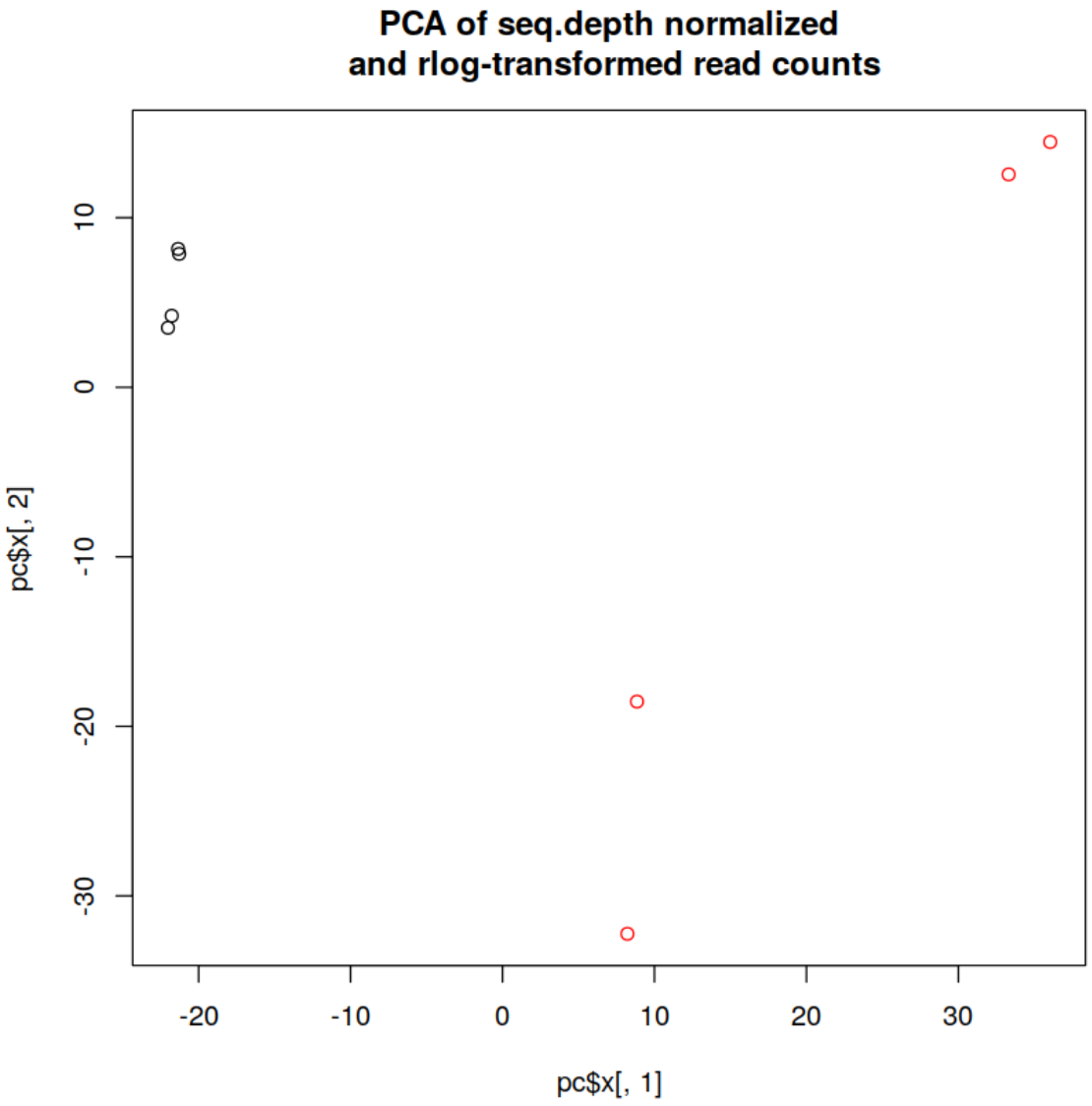
The result of PCA are principal components that represent the directions along which the variation in the original multi-dimensional data matrix is maximal. This way a few dimensions (components) can be used to represent the information from thousands of mRNAs. This allows us to, for example, visually represent the variation of the gene expression for different samples by using just the top two PCs as coordinates in a simple xy plot (instead of plotting thousands of genes per sample). Most commonly, the two principal components explaining the majority of the variability are displayed. It is also useful to identify unexpected patterns, such as batch effects or outliers. But keep in mind that PCA is not designed to discover unknown groupings; it is up to the researcher to actually identify the experimental or technical reason underlying the principal components.

The x-axis is the direction that separates the data points the most. The values of the samples in this direction are written PC1. The y-axis is a direction (it must be orthogonal to the first direction) that separates the data the second most. The values of the samples in this direction are written PC2. The percent of the total variance that is contained in the direction is printed in the axis label. Note that these percentages do not add to 100%, because there are more dimensions that contain the remaining variance (although each of these remaining dimensions will explain less than the two that we see). If you would like a more thorough description, I encourage you to explore StatQuest's video (https://www.youtube.com/watch?v=_UVHneBUBW0).

Depending on how much variation is explained by the first few principal components, you **may want to explore more (i.e. consider more components and plot pairwise combinations)**. Even if your samples do not separate clearly by the experimental variable, you may still get biologically releveant results from the DE analysis. If you are expecting very small effect sizes, then it's possible the signal is drowned out by extraneous sources of variation. In situations **where you can identify those sources, it is imortant to account for these in your model**, as it provides more power to the tool for detecting DE genes.

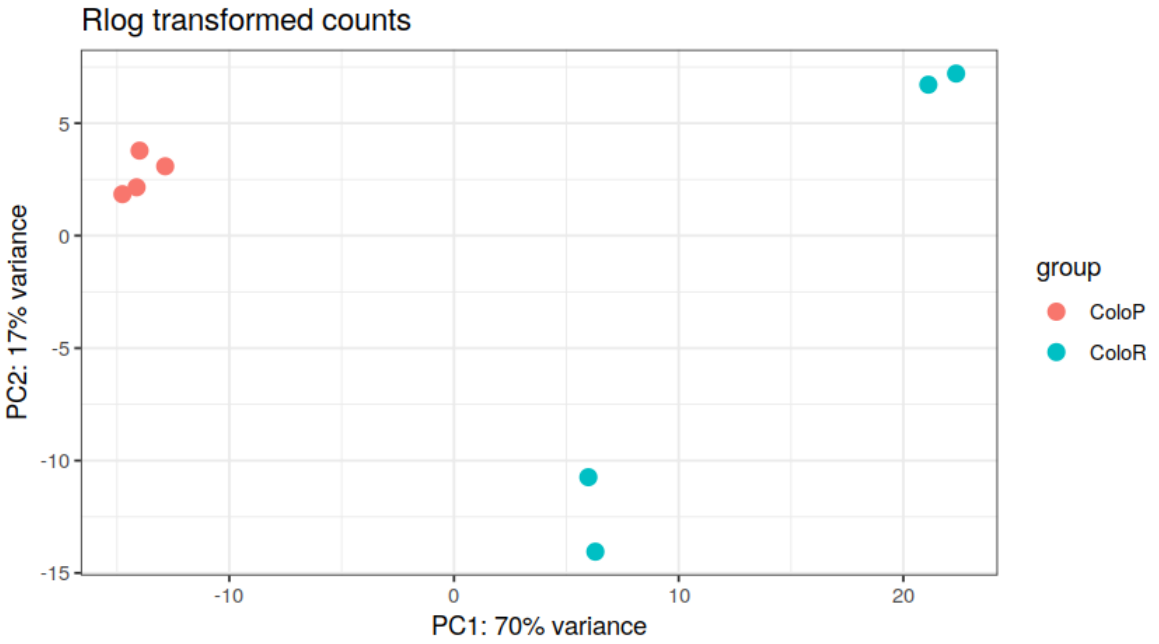
In [37]:

```r
# PCA can be performed in base R using the function prcomp().
pc <- prcomp(t(rlog.norm.counts))
plot(pc$x[,1], pc$x[,2],
    col = colData(DESeq.ds)[,1],
    main = "PCA of seq.depth normalized\n and rlog-transformed read counts
```



In [38]:

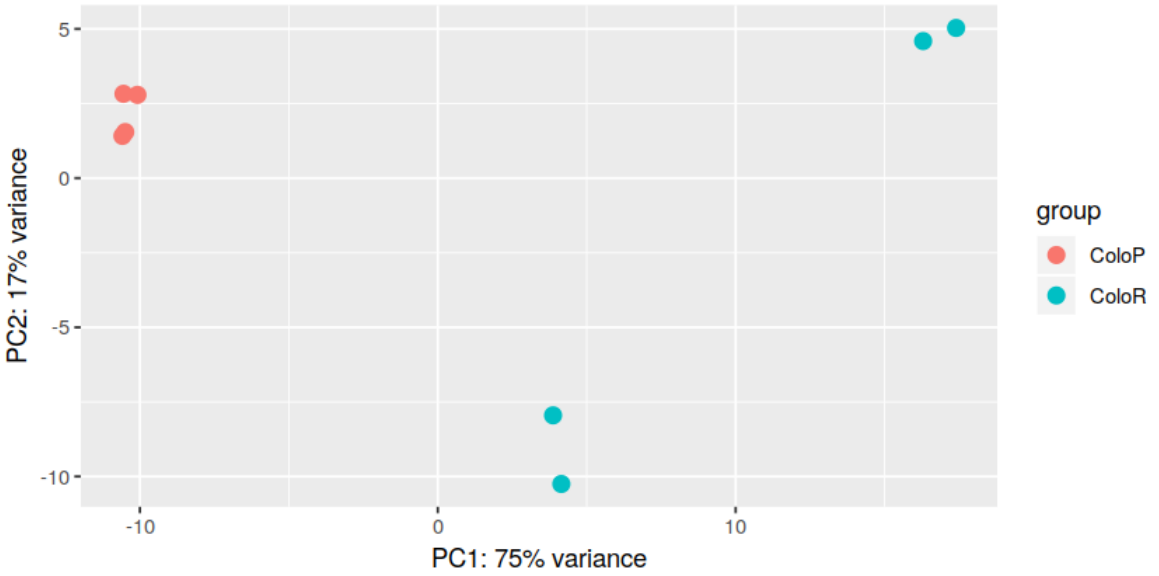
```r
# DESeq2 also offers a convenience function based on ggplot to do PCA dire
# on DESeqDataSet
# PCA
P <- plotPCA(DESeq.rlog)

# plot cosmetics
P <- P + theme_bw() + ggtitle("Rlog transformed counts")
print(P)
```



!Note PCA and clustering should be done on normalized and preferably transformed read counts, so that high variability of low read counts does not occlude potentially informative data trends.

In [44]:

```
# PCA plot using the VST data
plotPCA(vsd, intgroup = c("condition"))
```



In [42]:

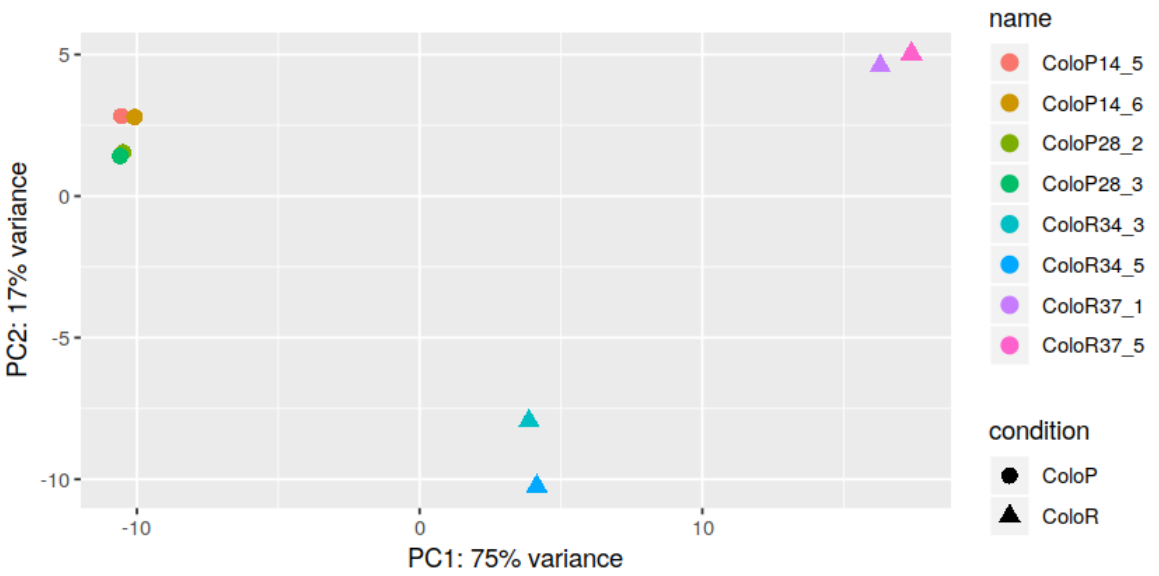
```
pcaData <- plotPCA(vsd, returnData = TRUE)
pcaData
```

A data.frame: 8 × 5

	PC1	PC2	group	condition	name
	<dbl>	<dbl>	<fct>	<fct>	<fct>
ColoP14_5	-10.550515	2.825792	ColoP	ColoP	ColoP14_5
ColoP14_6	-10.082600	2.790322	ColoP	ColoP	ColoP14_6
ColoP28_2	-10.492241	1.543791	ColoP	ColoP	ColoP28_2
ColoP28_3	-10.586360	1.414730	ColoP	ColoP	ColoP28_3
ColoR34_3	3.869987	-7.950152	ColoR	ColoR	ColoR34_3
ColoR34_5	4.147746	-10.249884	ColoR	ColoR	ColoR34_5
ColoR37_1	16.293303	4.592551	ColoR	ColoR	ColoR37_1
ColoR37_5	17.400680	5.032850	ColoR	ColoR	ColoR37_5

In [46]:

```r
# PCA plot using ggplot package
percentVar <- round(100 * attr(pcaData, "percentVar"))
ggplot(pcaData, aes(x = PC1, y = PC2, color = name, shape = condition)) +
  geom_point(size =3) +
  xlab(paste0("PC1: ", percentVar[1], "% variance")) +
  ylab(paste0("PC2: ", percentVar[2], "% variance")) +
  coord_fixed()
```

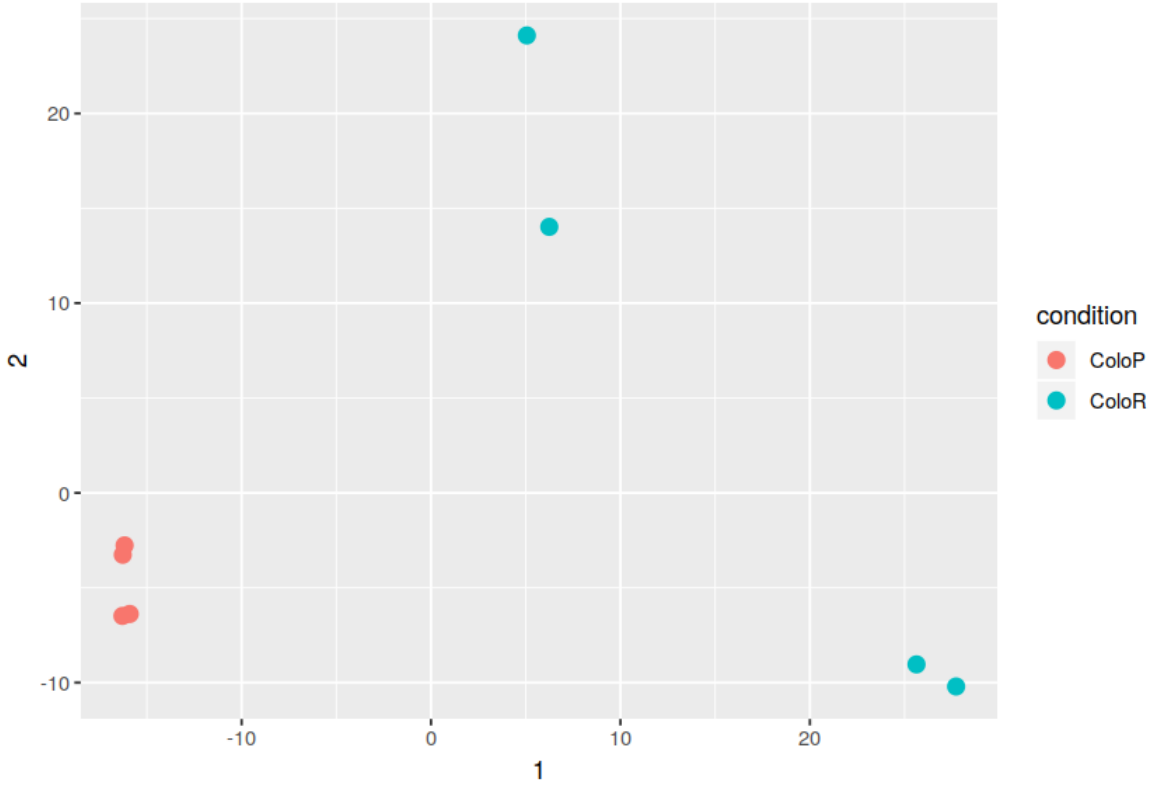


MDS plot

Another plot, very similar to the PCA plot, can be made using the multidimensional scaling (MDS) function in base R. This is useful when we don't have a matrix of data, but only a matrix of distances. MDS is a measn of visulalizing the level of similarity of individual cases of a dataset. MDS is used to translate "information about pairwise 'distances' among a set of n objects or individulas" into a configuration of n points mapped into an abstract Cartesian space. Given a distance matrix with the distances between each pair of objects in a set, and a chosen number of dimensions, N, an MDS algorithm places each object into N-dimensional space such that the between-object distances are preserved as well as possible. If *N* is one or two, then 2D scatter plots of the resulting points are possible. Here we compute the MDS for the distances calculated from the VST data and plot these in a figure below.

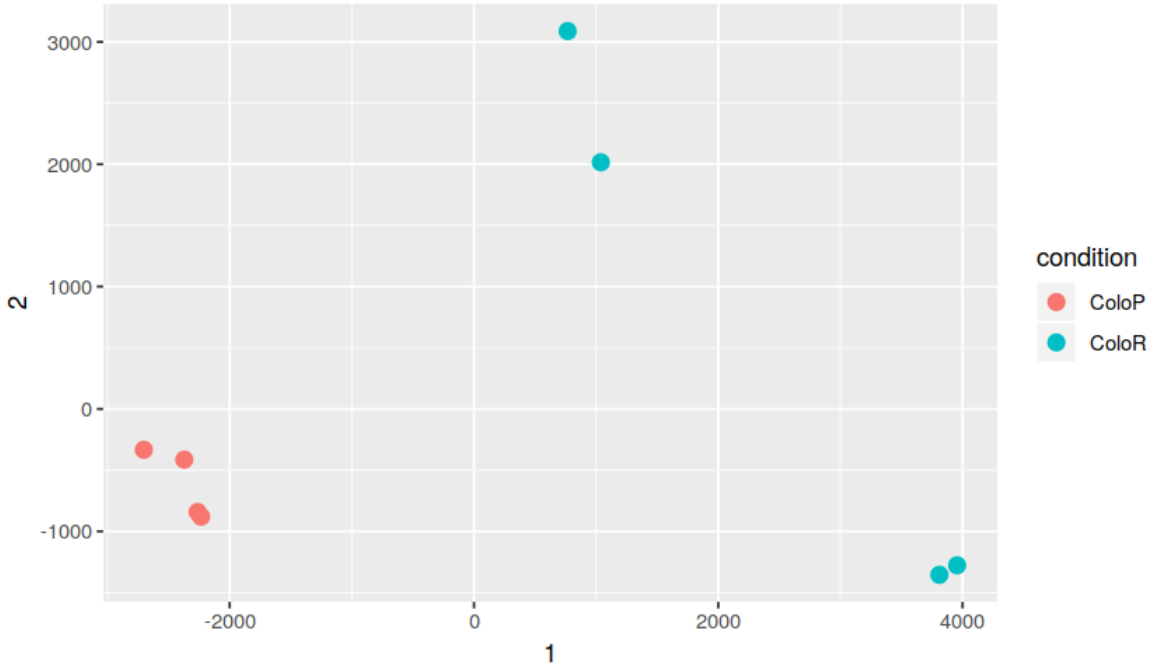
In [48]:

```
# MDS plot using VST data.
mds <- as.data.frame(colData(vsd))  %>%
         cbind(cmdscale(sampleDistMatrix))
ggplot(mds, aes(x = `1`, y = `2`, color = condition)) +
  geom_point(size = 3) + coord_fixed()
```



In [49]:

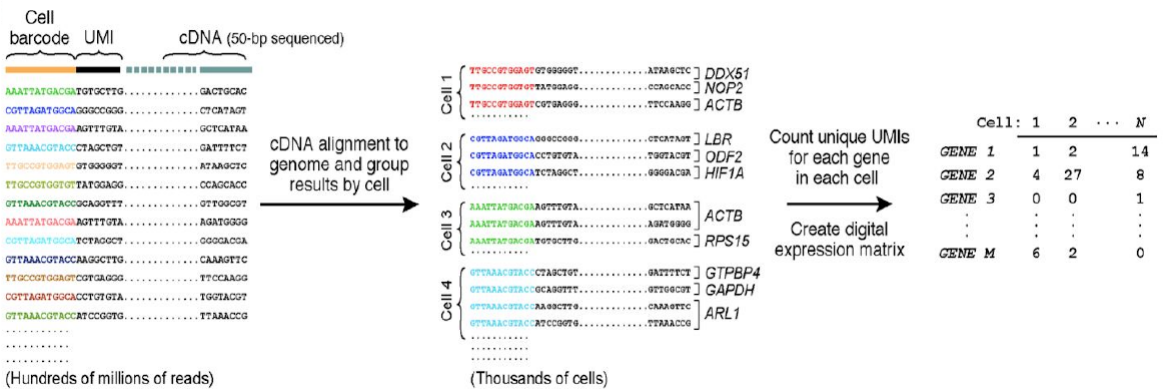
```
# MDS plot using the Poisson Distance
mdsPois <- as.data.frame(colData(DESeq.ds)) %>%
   cbind(cmdscale(samplePoisDistMatrix))
ggplot(mdsPois, aes(x = `1`, y = `2`, color = condition)) +
  geom_point(size = 3) + coord_fixed()
```



DESeq2 using Dropseq data

In this section I will repeat the analysis of the previous section, but using the data of the DropSeq Pipeline, which at the end will provide you with a DGE_matrix.txt file.

Overview of DGE extraction



To digitally count gene transcripts, a list of UMIs in each gene, within a sample, is assembled, and UMIs within edit distance = 1 are merged together. The total number of unique UMI sequences is counted and this number is reported as the number of transcripts of that gene for a given sample.

Digital Gene Expression

Extracting Digital Gene Expression (DGE) data from an aligned library is done using the Dropseq program DigitalExpression. The input to this program is the aligned BAM from the alignment workflow.

```
$DropSeqPath'DigitalExpression' -m 30g -t $tmpDir I=star_gene_exon_tag
ged.bam O=DGE_Matrix.txt CELL_BARCODE_TAG=XC MOLECULAR_BARCODE_TAG=XM
CELL_BC_FILE=$BarcodeInfo
```

There are two outputs available, the primary is the DGE matrix, with each row for each gene, and a column for each sample. The secondary analysis is a summary of the DGE matrix on a per-sample level, indication the number of genes and transcripts observed. This file is ordered by the list of sample barcodes, then the number of UMIs per gene, then alphabetically by gene when they have the same number of UMIs. There are no entries when a sample does not have expression of a gene.

In [1]:
```
# get the table of read counts by indicating the path to the DGE_Matrix.tx
readcounts_dp <- read.table("/mnt/data/Christiane_Dropseq/christinane_Outp
                            header = TRUE)
# show the first lines of the table
head(readcounts_dp)
```

A data.frame: 6 × 25

GENE	TAGCTTGT	GGCTACAG	TGTACCTT	TGCGATCT	TTGGTATG	TCATTGAG	TTACTCGC	TAGAAC
<fct>	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<i
A1CF	13	17	3	10	3	2	1	
A2M-AS1	0	0	0	0	0	0	0	
A4GALT	1	2	7	0	12	39	23	
AAAS	42	47	25	23	30	110	77	
AACS	50	44	13	22	16	25	37	
AADAC	0	1	0	0	2	9	44	

The samples are named with the barcode information and we have to rename them. The first column are the gene names.

In [2]:

```
# one of the requirements of the assay() slots is that the row.names corre
# the gene ID's and the col.names to the sample names
row.names(readcounts_dp) <- readcounts_dp$GENE

# in addition, we need to exclude all columns that do not contain read cou
readcounts_dp <- readcounts_dp[, c(2:25)] # extract column 2 to 25

# give meaningful sample names
orig_names <- names(readcounts_dp)
names(readcounts_dp) <- c("ColoP14_5", "ColoP14_6", "ColoR37_1", "ColoR37_
                          "HT29P14_6", "HT29R19_2", "HT29R19_5", "SW480P18_4"
"HT29P25_2", "HT29P25_5", "HT29R20_1", "HT29R20_5", "SW480P20_1", "SW480P2

# check the data after manipulation
str(readcounts_dp)
```

```
'data.frame':   29411 obs. of  24 variables:
 $ ColoP14_5 : int  13 0 1 42 50 0 0 1 0 4 ...
 $ ColoP14_6 : int  17 0 2 47 44 1 0 0 0 2 ...
 $ ColoR37_1 : int  3 0 7 25 13 0 0 0 1 4 ...
 $ ColoR37_5 : int  10 0 0 23 22 0 0 0 0 3 ...
 $ HT29P14_4 : int  3 0 12 30 16 2 0 0 1 4 ...
 $ HT29P14_6 : int  2 0 39 110 25 9 0 0 4 1 ...
 $ HT29R19_2 : int  1 0 23 77 37 44 0 0 10 4 ...
 $ HT29R19_5 : int  2 1 26 74 30 27 0 0 11 7 ...
 $ SW480P18_4: int  5 0 2 107 18 1 0 0 0 9 ...
 $ SW480P18_5: int  15 2 4 186 36 0 1 0 1 13 ...
 $ SW480R24_1: int  2 3 9 113 22 1 0 0 8 2 ...
 $ SW480R24_5: int  0 0 8 56 12 0 0 0 6 0 ...
 $ ColoP28_2 : int  17 0 1 41 54 0 0 0 0 7 ...
 $ ColoP28_3 : int  25 0 0 77 51 1 0 0 0 9 ...
 $ ColoR34_3 : int  11 0 2 47 23 0 0 0 0 7 ...
 $ ColoR34_5 : int  3 0 0 34 32 1 0 0 0 4 ...
 $ HT29P25_2 : int  1 0 39 78 32 2 0 0 7 3 ...
 $ HT29P25_5 : int  0 0 35 77 28 5 0 0 4 3 ...
 $ HT29R20_1 : int  0 0 6 25 15 6 0 0 1 1 ...
 $ HT29R20_5 : int  2 0 9 27 24 3 0 0 7 2 ...
 $ SW480P20_1: int  6 0 0 78 14 1 0 0 1 10 ...
 $ SW480P20_2: int  2 0 0 68 13 0 0 0 0 4 ...
 $ SW480R26_2: int  0 1 3 70 11 2 0 0 6 1 ...
 $ SW480R26_6: int  2 1 7 65 12 0 0 0 3 3 ...
```

In [3]:

```
head(readcounts_dp, n=3)
```

A data.frame: 3 × 24

	ColoP14_5	ColoP14_6	ColoR37_1	ColoR37_5	HT29P14_4	HT29P14_6	HT29R19_2	HT29R19_5
	<int>	<int>	<int>	<int>	<int>	<int>	<int>	<int>
A1CF	13	17	3	10	3	2	1	2
A2M-AS1	0	0	0	0	0	0	0	1
A4GALT	1	2	7	0	12	39	23	26

In [4]:

```
names(readcounts_dp)
```

'ColoP14_5' 'ColoP14_6' 'ColoR37_1' 'ColoR37_5' 'HT29P14_4' 'HT29P14_6' 'HT29R19_2' 'HT29R19_5' 'SW480P18_4' 'SW480P18_5' 'SW480R24_1' 'SW480R24_5' 'ColoP28_2' 'ColoP28_3' 'ColoR34_3' 'ColoR34_5' 'HT29P25_2' 'HT29P25_5' 'HT29R20_1' 'HT29R20_5' 'SW480P20_1' 'SW480P20_2' 'SW480R26_2' 'SW480R26_6'

In [5]:

```r
# create the colData table with the samples and the conditions
# parental (P) resistent (R) and cellline (Colo, HT29, SW480)
sample_info_dp <- data.frame(condition = c("P","P","R","R","P","P","R","R"
                             "P","P","R","R","P","P","R","R","P","P","R","R")
                             c("Colo", "Colo", "Colo","Colo","HT29", "HT29",
                              "SW480", "SW480", "SW480", "SW480",
                              "Colo", "Colo", "Colo","Colo","HT29", "HT29", "
                              "SW480", "SW480", "SW480", "SW480"),
                             row.names = names(readcounts_dp))
sample_info_dp
```

A data.frame: 24 × 2

	condition	cline
	<fct>	<fct>
ColoP14_5	P	Colo
ColoP14_6	P	Colo
ColoR37_1	R	Colo
ColoR37_5	R	Colo
HT29P14_4	P	HT29
HT29P14_6	P	HT29
HT29R19_2	R	HT29
HT29R19_5	R	HT29
SW480P18_4	P	SW480
SW480P18_5	P	SW480
SW480R24_1	R	SW480
SW480R24_5	R	SW480
ColoP28_2	P	Colo
ColoP28_3	P	Colo
ColoR34_3	R	Colo
ColoR34_5	R	Colo
HT29P25_2	P	HT29
HT29P25_5	P	HT29
HT29R20_1	R	HT29
HT29R20_5	R	HT29
SW480P20_1	P	SW480
SW480P20_2	P	SW480
SW480R26_2	R	SW480
SW480R26_6	R	SW480

In [8]:
```
# generate the DESeqDataSet
library(DESeq2)
library(magrittr)
DESeq_dp <- DESeqDataSetFromMatrix(countData = readcounts_dp,
                                   colData = sample_info_dp,
                                   design = ~ condition + cline)

# check the result using the accessors
colData(DESeq_dp) %>% head()
```

```
DataFrame with 6 rows and 2 columns
          condition    cline
           <factor> <factor>
ColoP14_5         P     Colo
ColoP14_6         P     Colo
ColoR37_1         R     Colo
ColoR37_5         R     Colo
HT29P14_4         P     HT29
HT29P14_6         P     HT29
```

In [9]:
```
assay(DESeq_dp, "counts") %>% head()
```

A matrix: 6 × 24 of type int

	ColoP14_5	ColoP14_6	ColoR37_1	ColoR37_5	HT29P14_4	HT29P14_6	HT29R19_2	HT29R19_5
A1CF	13	17	3	10	3	2	1	2
A2M-AS1	0	0	0	0	0	0	0	1
A4GALT	1	2	7	0	12	39	23	26
AAAS	42	47	25	23	30	110	77	74
AACS	50	44	13	22	16	25	37	30
AADAC	0	1	0	0	2	9	44	27

In [10]:
```
counts(DESeq_dp) %>% str
```

```
int [1:29411, 1:24] 13 0 1 42 50 0 0 1 0 4 ...
- attr(*, "dimnames")=List of 2
 ..$ : chr [1:29411] "A1CF" "A2M-AS1" "A4GALT" "AAAS" ...
 ..$ : chr [1:24] "ColoP14_5" "ColoP14_6" "ColoR37_1" "ColoR37_5" ...
```

In [11]:
```
# number of genes
nrow(DESeq_dp)
```

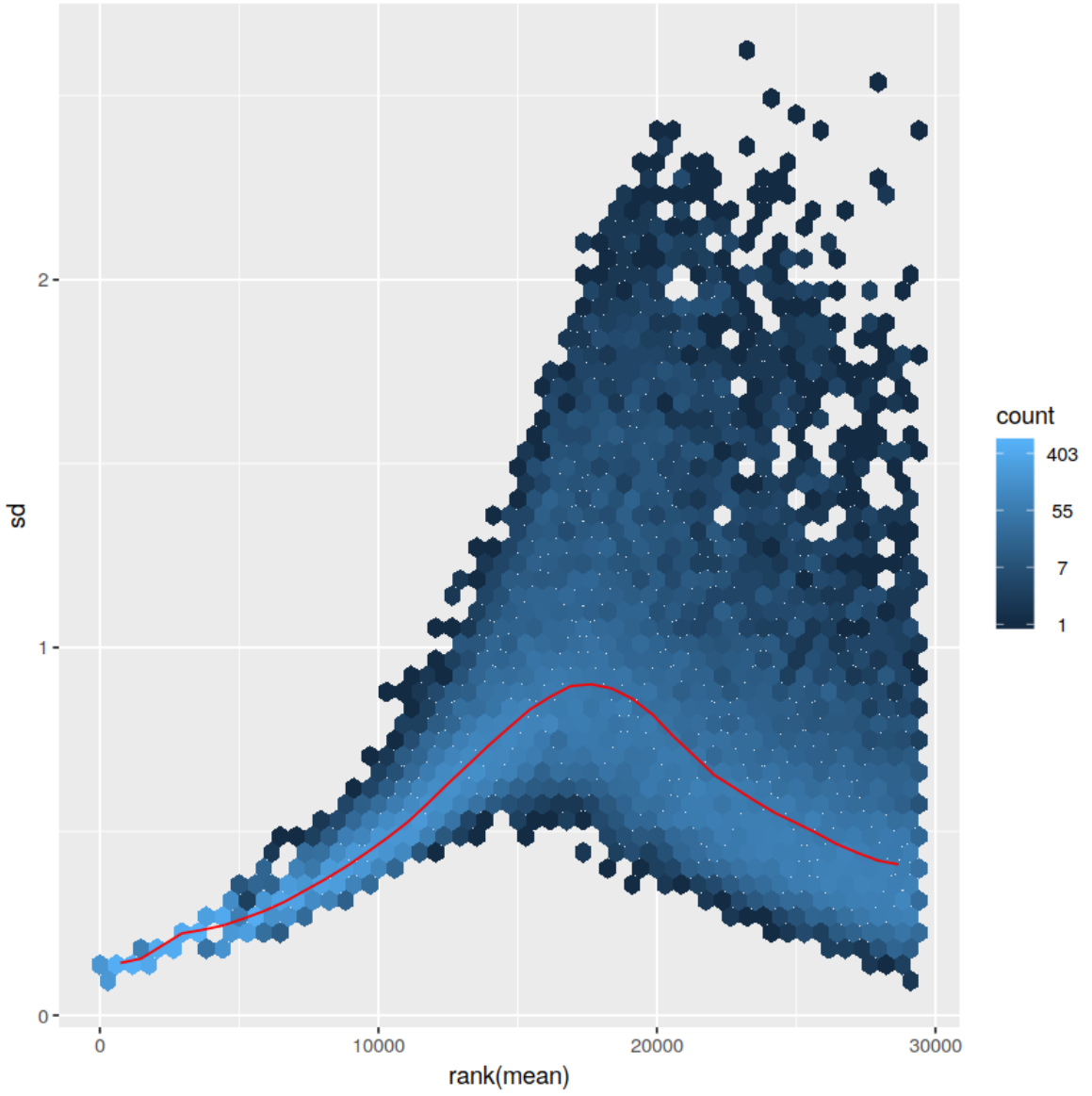
29411

In [12]:
```
# remove genes without any counts
DESeq_dp <- DESeq_dp[ rowSums(counts(DESeq_dp)) > 0, ]
nrow(DESeq_dp)
```

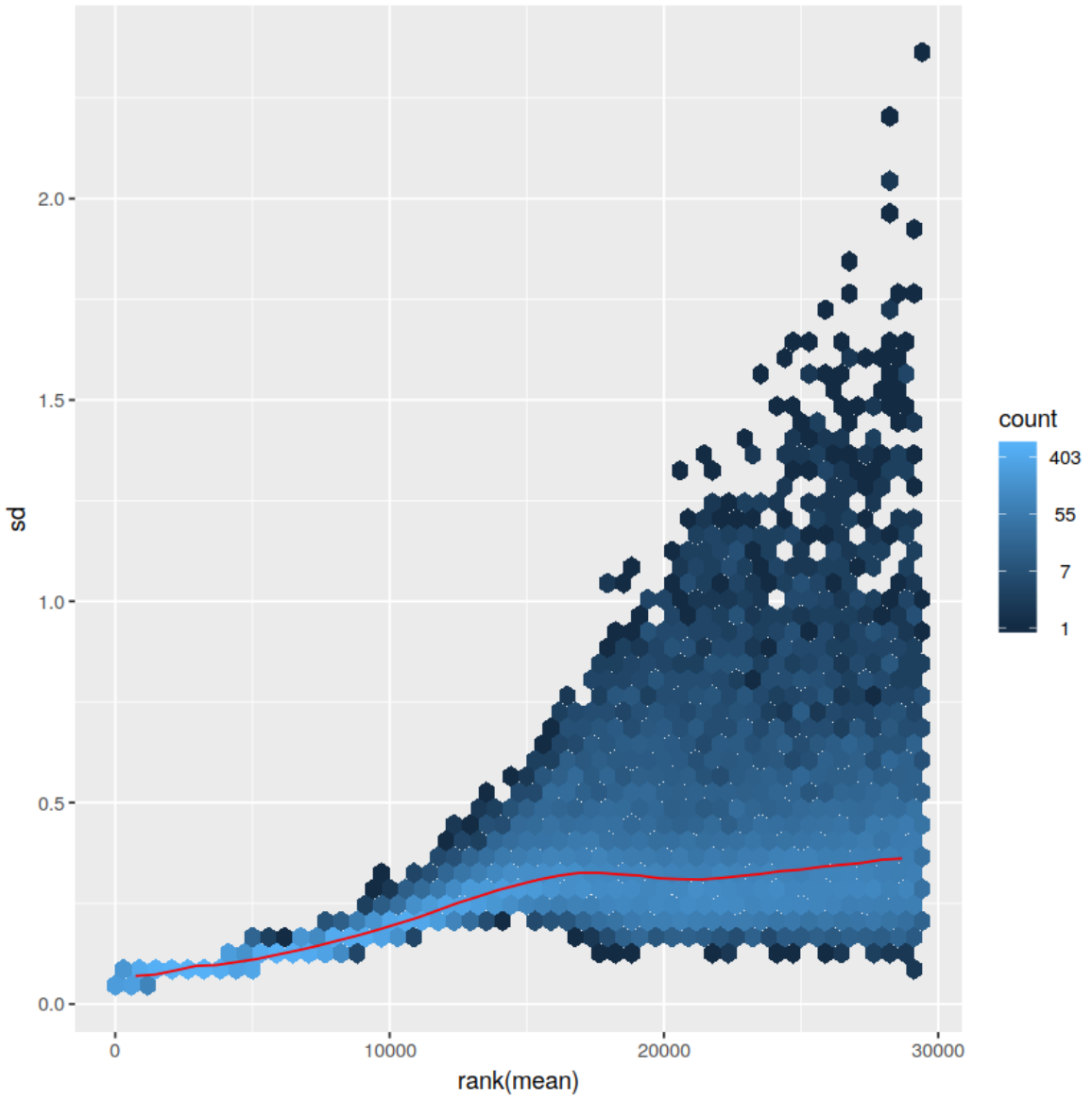
29411

In [13]:

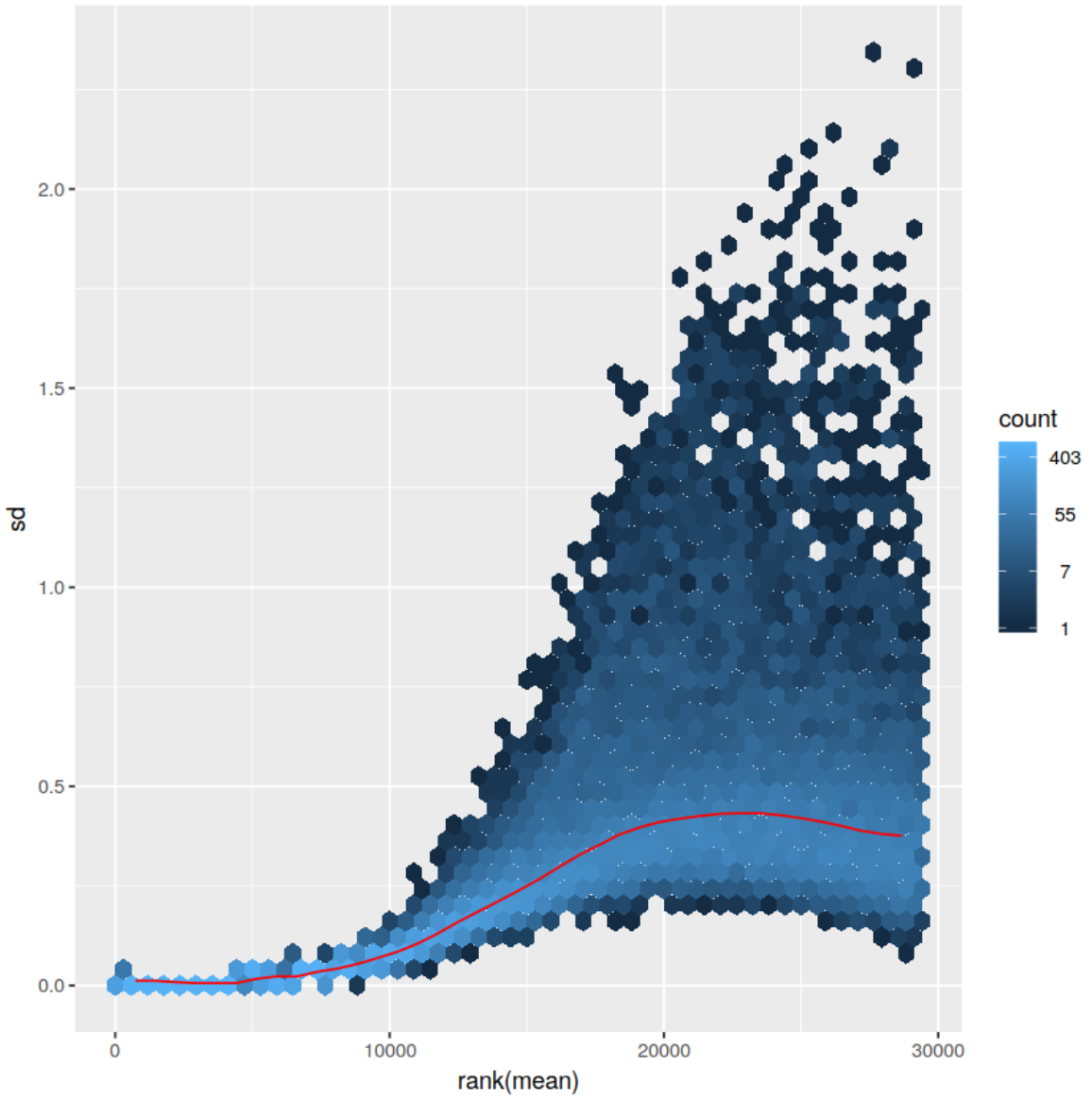
```r
# transformation of the data
# VST
vsd_dp <- vst(DESeq_dp, blind = FALSE)

# rlog
rld_dp <- rlog(DESeq_dp, blind = FALSE)

# normal
ntd_dp <- normTransform(DESeq_dp)
library("vsn")
meanSdPlot(assay(ntd_dp))
```



In [14]:
```
1 meanSdPlot(assay(vsd_dp))
```



In [15]:
```
1 meanSdPlot(assay(rld_dp))
```



In [16]:

```
# calculate the size factor and add it to the data set
DESeq_dp <- estimateSizeFactors(DESeq_dp)
sizeFactors(DESeq_dp)
```

ColoP14_5	0.71735596341219
ColoP14_6	0.693997156458135
ColoR37_1	0.841318167262475
ColoR37_5	0.754669106465864
HT29P14_4	0.517376607277679
HT29P14_6	1.7689881904958
HT29R19_2	1.6082936592719
HT29R19_5	1.54974805280366
SW480P18_4	1.43213008506613
SW480P18_5	2.04051950720212
SW480R24_1	1.45733051881396
SW480R24_5	0.832724300103576
ColoP28_2	0.795076506402101
ColoP28_3	1.01082314599897
ColoR34_3	1.04783666100118
ColoR34_5	0.832147495524851
HT29P25_2	1.67926081469593
HT29P25_5	1.35145842253194
HT29R20_1	0.652040241114447
HT29R20_5	1.08081336428269
SW480P20_1	0.817404698544088
SW480P20_2	0.702418900967911
SW480R26_2	1.11630763110992
SW480R26_6	0.864766346070931

In [17]:

```
# if you check colData() again, you see that this now contains the sizeFac
colData(DESeq_dp)
```

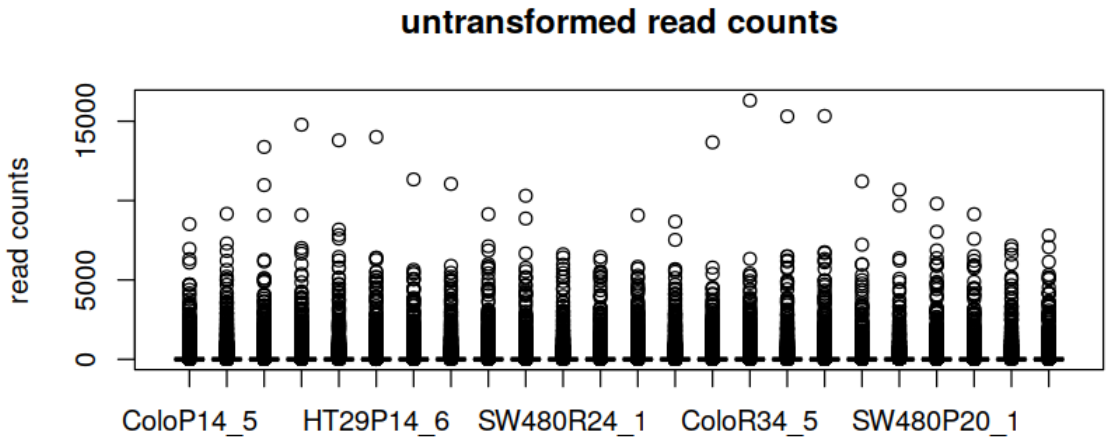
```
DataFrame with 24 rows and 3 columns
           condition    cline        sizeFactor
            <factor> <factor>         <numeric>
ColoP14_5          P     Colo  0.71735596341219
ColoP14_6          P     Colo 0.693997156458135
ColoR37_1          R     Colo 0.841318167262475
ColoR37_5          R     Colo 0.754669106465864
HT29P14_4          P     HT29 0.517376607277679
...              ...      ...               ...
HT29R20_5          R     HT29  1.08081336428269
SW480P20_1         P    SW480 0.817404698544088
SW480P20_2         P    SW480 0.702418900967911
SW480R26_2         R    SW480  1.11630763110992
SW480R26_6         R    SW480 0.864766346070931
```

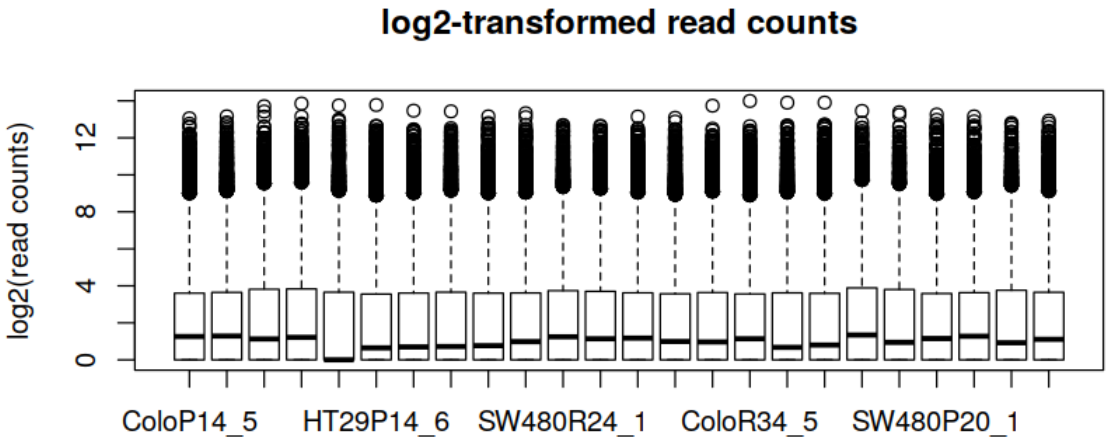
In [18]:

```
# counts() allows you to immediately retrieve the _normalized_ read counts
counts.sf_normalized_dp <- counts(DESeq_dp, normalized = TRUE)
```

In [19]:

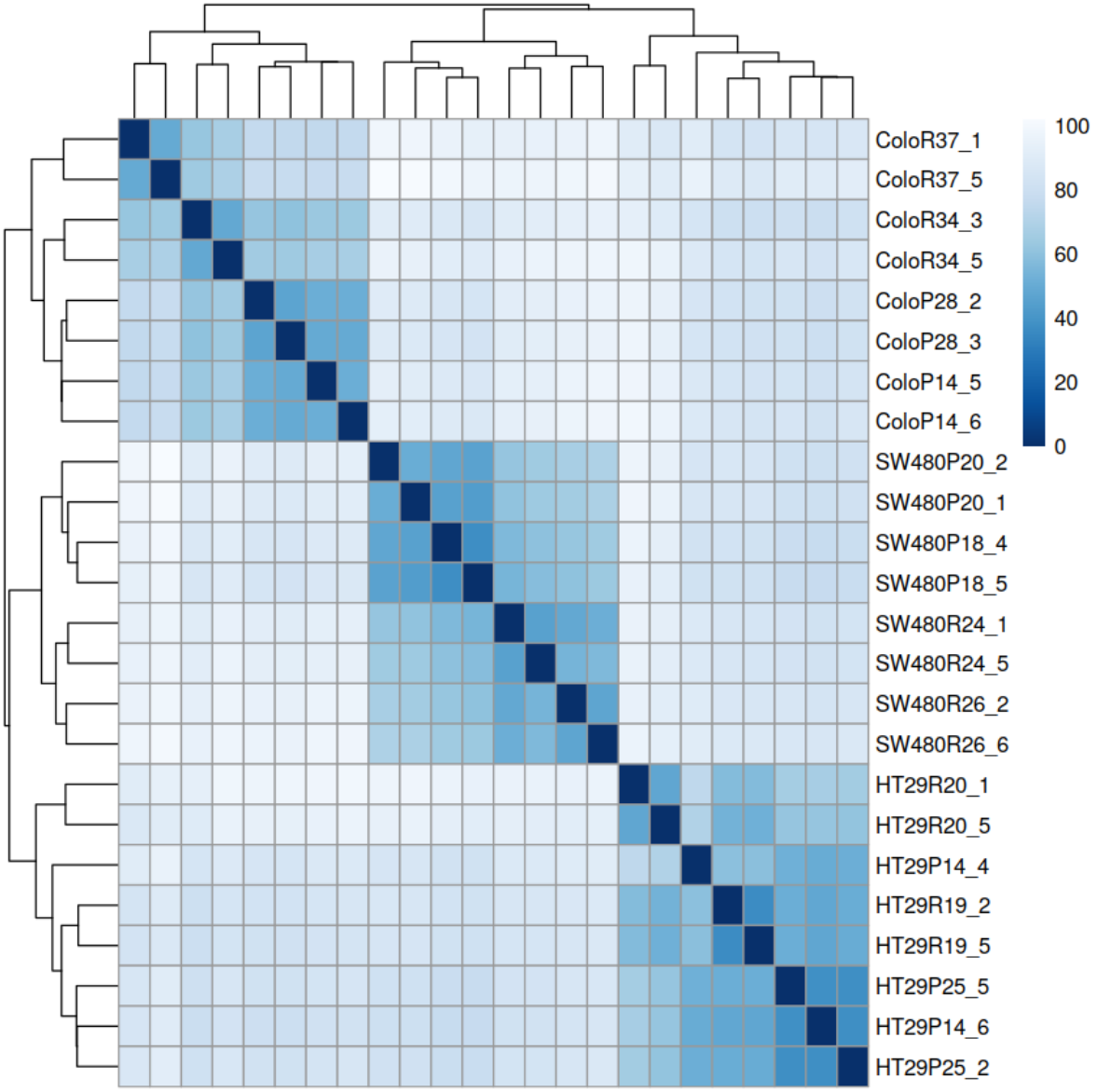
```r
#transform size-factor normalized read counts to log2 scale using a pseudo
log.norm.counts_dp <- log2(counts.sf_normalized_dp + 1)

par(mfrow=c(2,1)) # to plot the following two images underneath each other

# first, boxplot of non-transformed read counts (one per sample)
boxplot(counts.sf_normalized_dp, notch = FALSE,
        main = "untransformed read counts", ylab = "read counts")

# box plots of log2-transformed read counts
boxplot(log.norm.counts_dp, notch = FALSE,
        main = "log2-transformed read counts",
        ylab = "log2(read counts)")
```





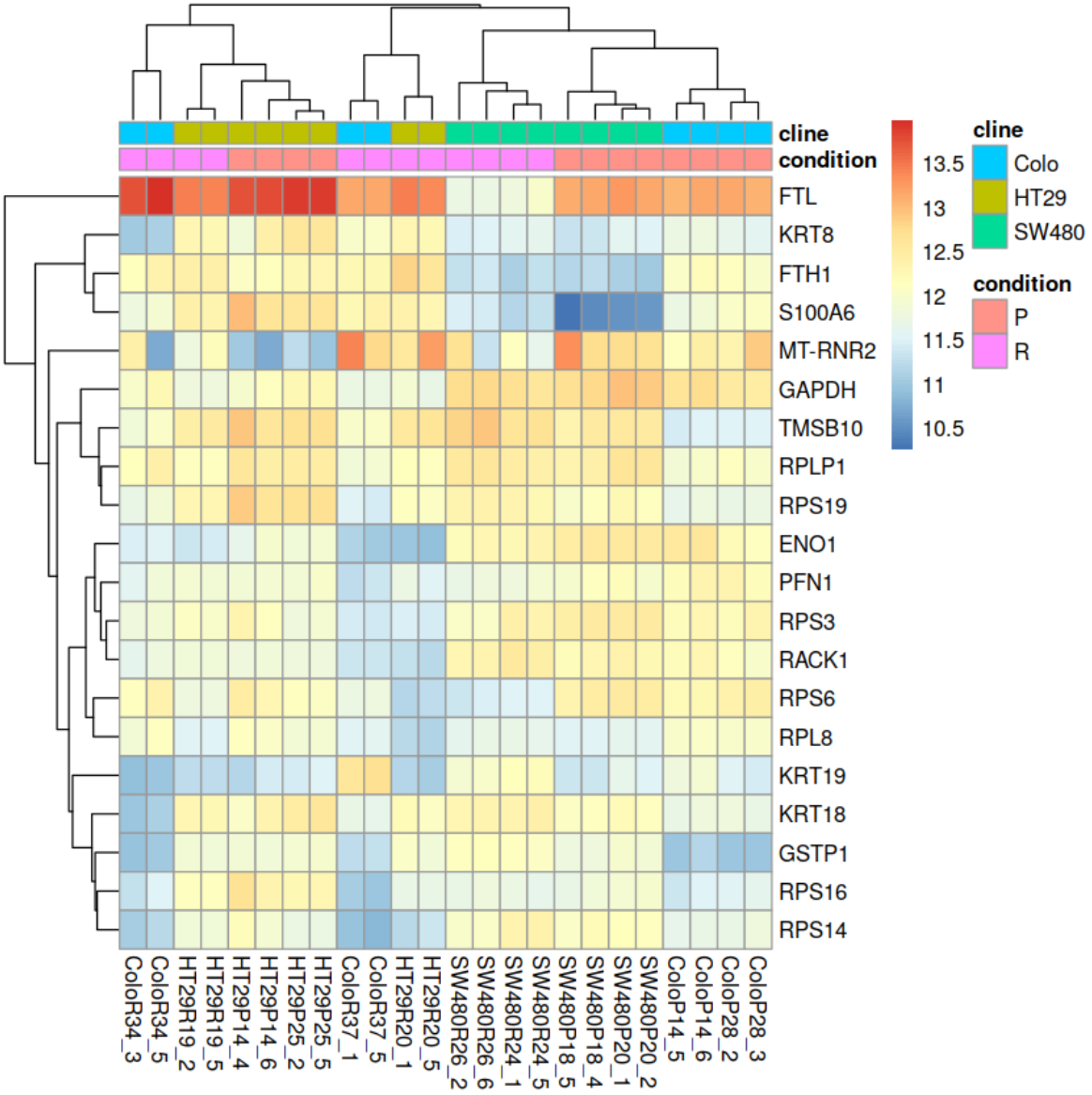
In [20]:

```
# calculate distances and create heatmap
library("pheatmap")
library("RColorBrewer")

sampleDists_dp <- dist(t(assay(vsd_dp)))
sampleDistMatrix_dp <- as.matrix( sampleDists_dp )
rownames(sampleDistMatrix_dp) <- paste(names(readcounts_dp))
colnames(sampleDistMatrix_dp) <- NULL
colors <- colorRampPalette( rev(brewer.pal(9, "Blues")) )(255)
pheatmap(sampleDistMatrix_dp,
         clustering_distance_rows = sampleDists_dp,
         clustering_distance_cols = sampleDists_dp,
         col = colors)
```



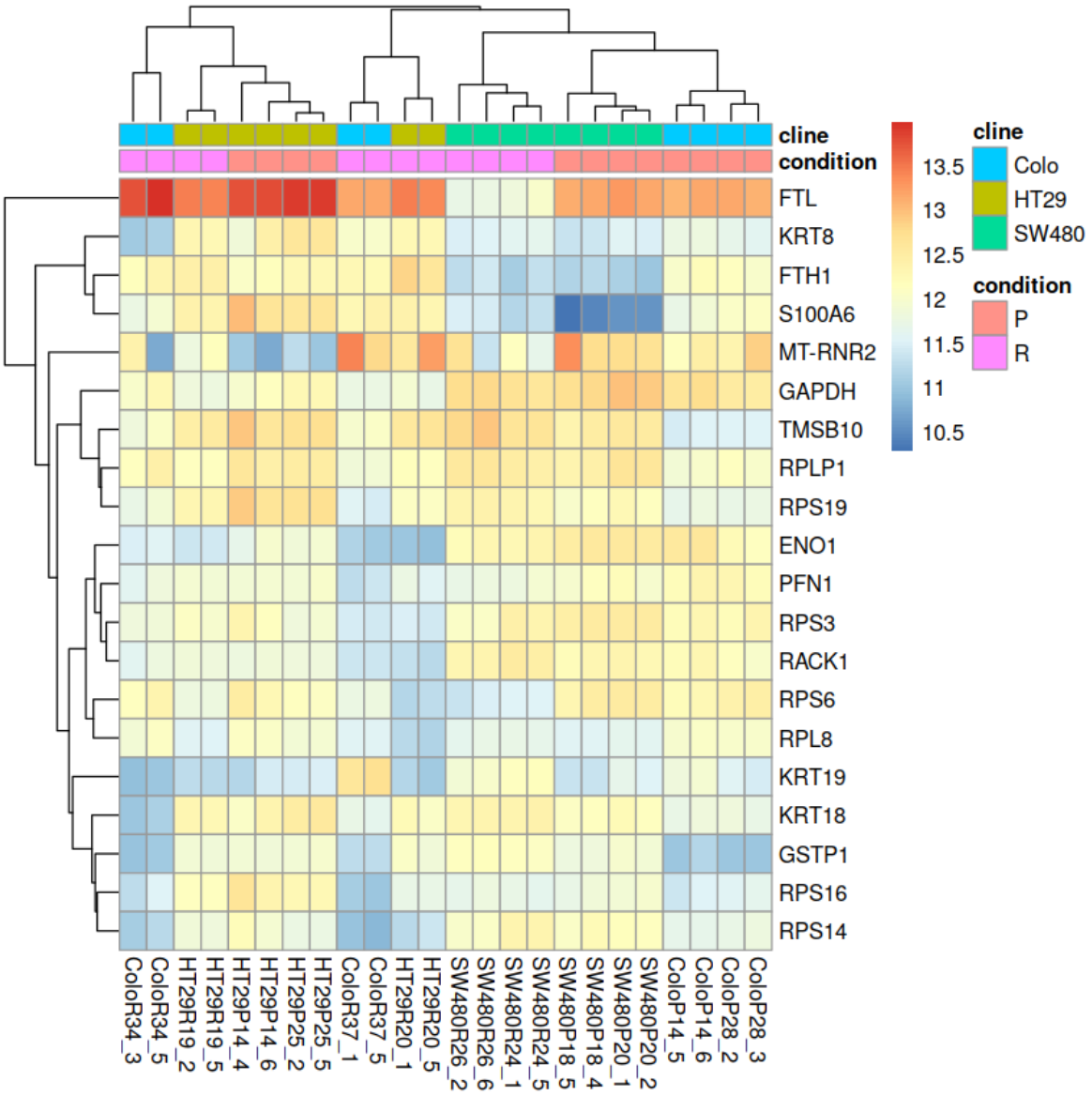
In [26]:

```
library("pheatmap")
select <- order(rowMeans(counts(DESeq_dp,normalized=TRUE)),
decreasing=TRUE)[1:20]
df_dp <- as.data.frame(colData(DESeq_dp)[,c("condition","cline")])
pheatmap(assay(ntd_dp)[select,], cluster_rows=TRUE, show_rownames=TRUE,
cluster_cols=TRUE, annotation_col=df_dp)

```



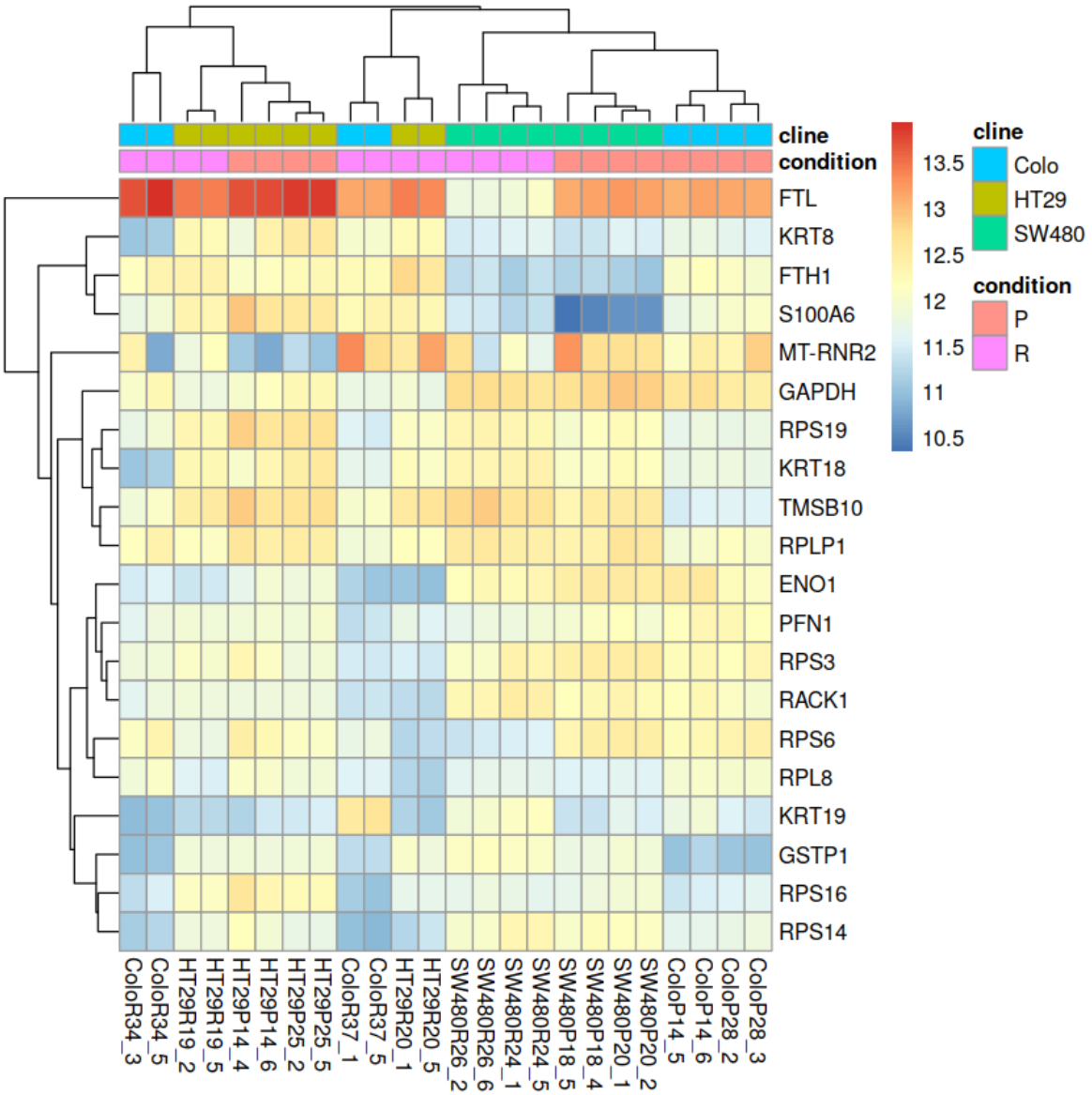
In [27]:

```
# with vsd
pheatmap(assay(vsd_dp)[select,], cluster_rows=TRUE, show_rownames=TRUE,
cluster_cols=TRUE, annotation_col=df_dp)
```



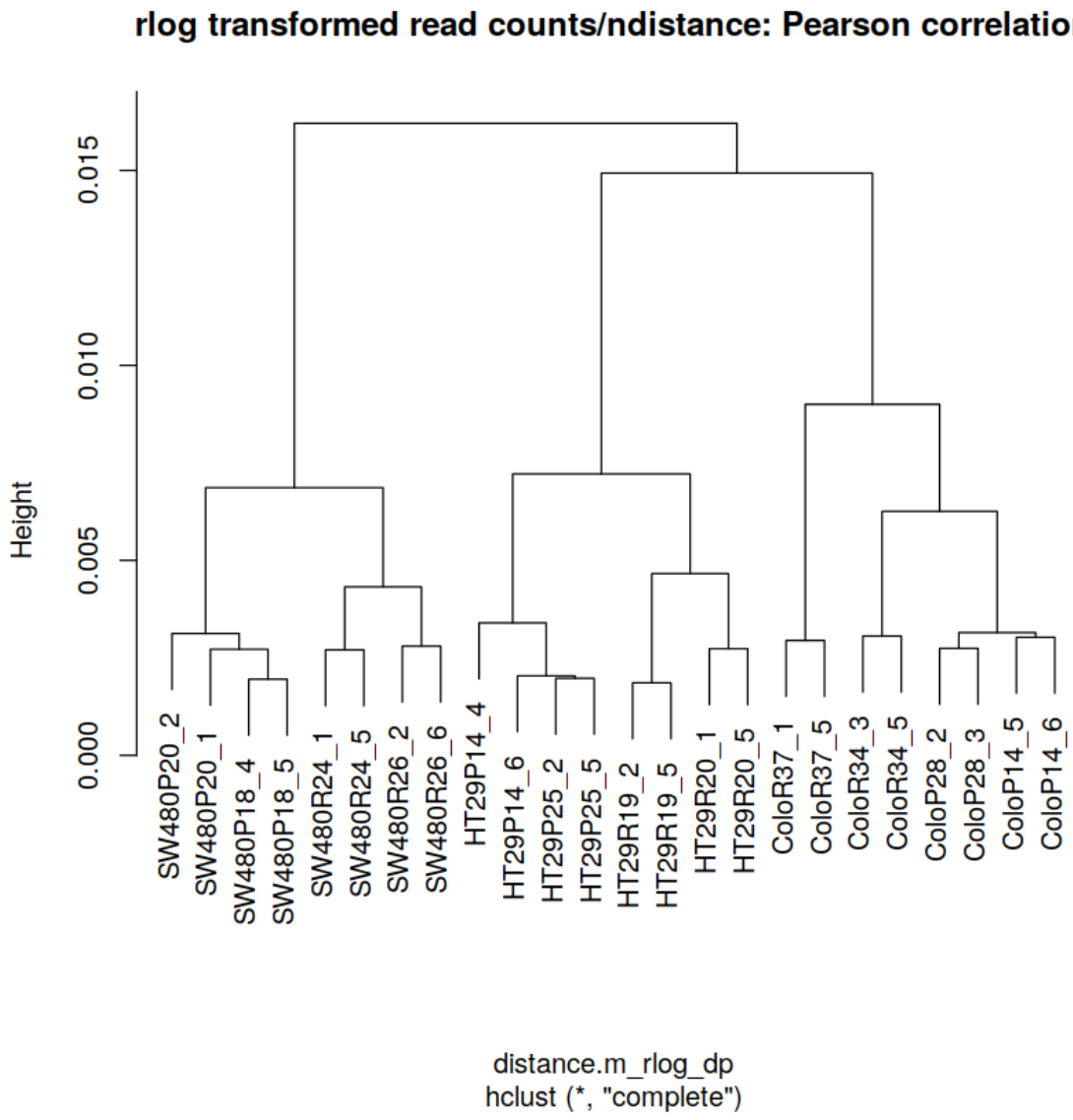
In [28]:

```
pheatmap(assay(rld_dp)[select,], cluster_rows=TRUE, show_rownames=TRUE,
cluster_cols=TRUE, annotation_col=df_dp)
```



In [29]:

```
# Dendrogram
# in R pairwise correlations can be calculated with the cor() function
# obtain regularized log-transformed values
DESeq.rlog_dp <- rlog(DESeq_dp, blind = TRUE)
rlog.norm.counts_dp <- assay(DESeq.rlog_dp)
distance.m_rlog_dp <- as.dist(1 - cor(rlog.norm.counts_dp, method = "pears

# plot() can directly interpret the output of hclust()
plot(hclust(distance.m_rlog_dp),
      labels = colnames(rlog.norm.counts_dp),
      main = "rlog transformed read counts/ndistance: Pearson correlation"
```



In [30]:

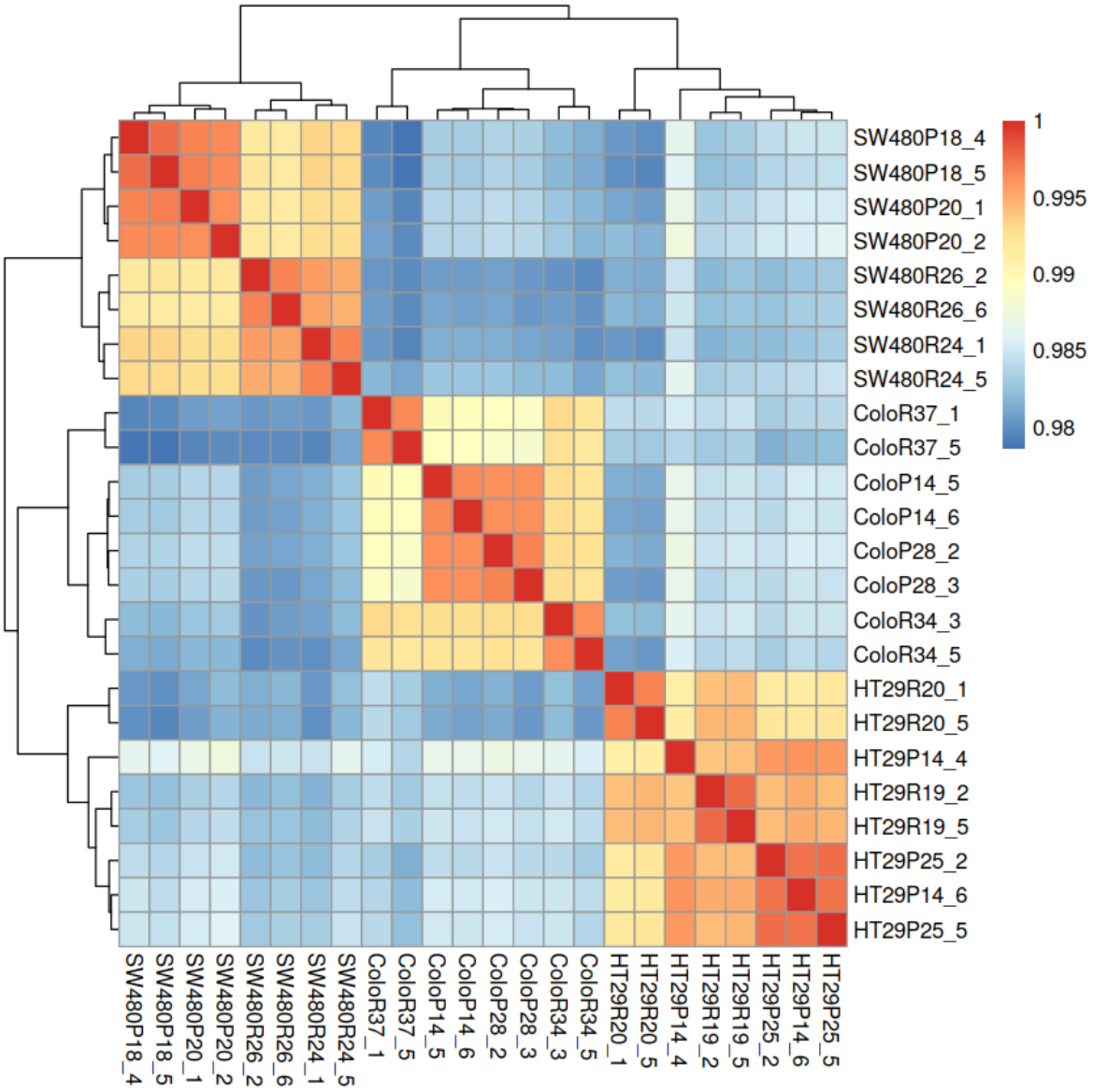
```
# Extract the rlog matrix from the object
rld_mat_dp <- assay(rld_dp)
rld_cor_dp  <- cor(rld_mat_dp) # cor() is a base R function for pairwise c
head(rld_cor_dp)
```

A matrix: 6 × 24 of type dbl

	ColoP14_5	ColoP14_6	ColoR37_1	ColoR37_5	HT29P14_4	HT29P14_6	HT29R19_2	HT29R19
ColoP14_5	1.0000000	0.9964855	0.9897347	0.9893489	0.9865840	0.9853417	0.9844809	0.9849
ColoP14_6	0.9964855	1.0000000	0.9896001	0.9891635	0.9865154	0.9851283	0.9842486	0.9847
ColoR37_1	0.9897347	0.9896001	1.0000000	0.9965395	0.9853515	0.9837694	0.9842263	0.9847
ColoR37_5	0.9893489	0.9891635	0.9965395	1.0000000	0.9839155	0.9821538	0.9828869	0.9834
HT29P14_4	0.9865840	0.9865154	0.9853515	0.9839155	1.0000000	0.9960303	0.9938418	0.9940
HT29P14_6	0.9853417	0.9851283	0.9837694	0.9821538	0.9960303	1.0000000	0.9949540	0.9949

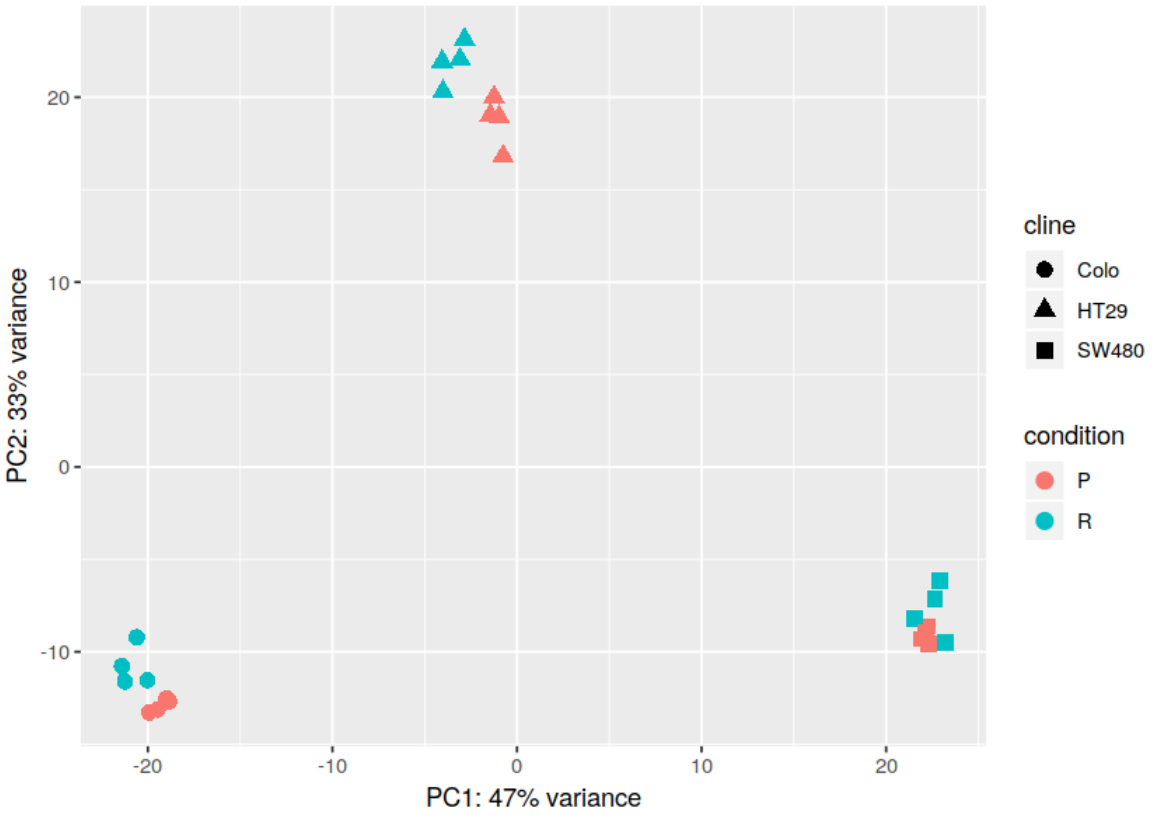
In [31]:

```
# Plot heatmap
pheatmap(rld_cor_dp)
```



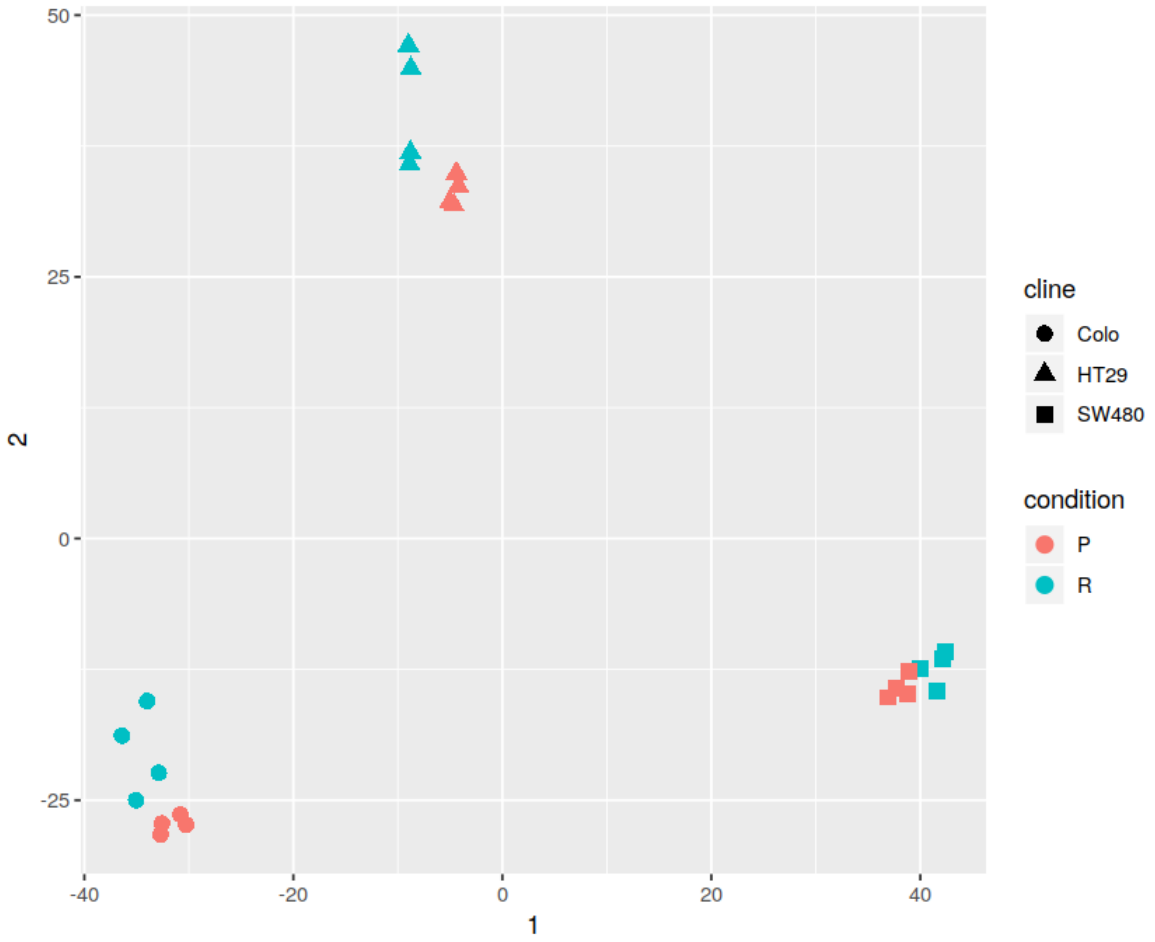
In [34]:

```
# PCA plot using ggplot package
library(ggplot2)
pcaData_dp <- plotPCA(vsd_dp, intgroup=c("condition", "cline"), returnData
percentVar_dp <- round(100 * attr(pcaData_dp, "percentVar"))
ggplot(pcaData_dp, aes(x = PC1, y = PC2, color = condition, shape = cline)
  geom_point(size =3) +
  xlab(paste0("PC1: ", percentVar_dp[1], "% variance")) +
  ylab(paste0("PC2: ", percentVar_dp[2], "% variance")) +
  coord_fixed()
```



In [35]:

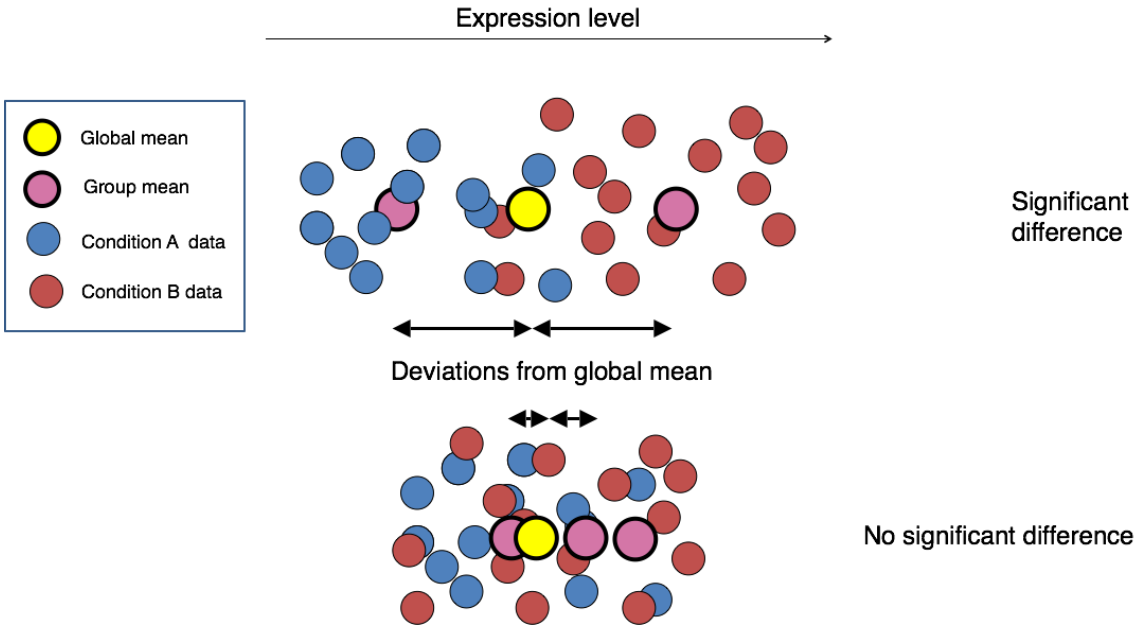
```
# MDS plot using VST data.
mds_dp <- as.data.frame(colData(vsd_dp))  %>%
         cbind(cmdscale(sampleDistMatrix_dp))
ggplot(mds_dp, aes(x = `1`, y = `2`, color = condition, shape = cline)) +
  geom_point(size = 3) + coord_fixed()
```



Differential Gene Expression Analysis (DGE)

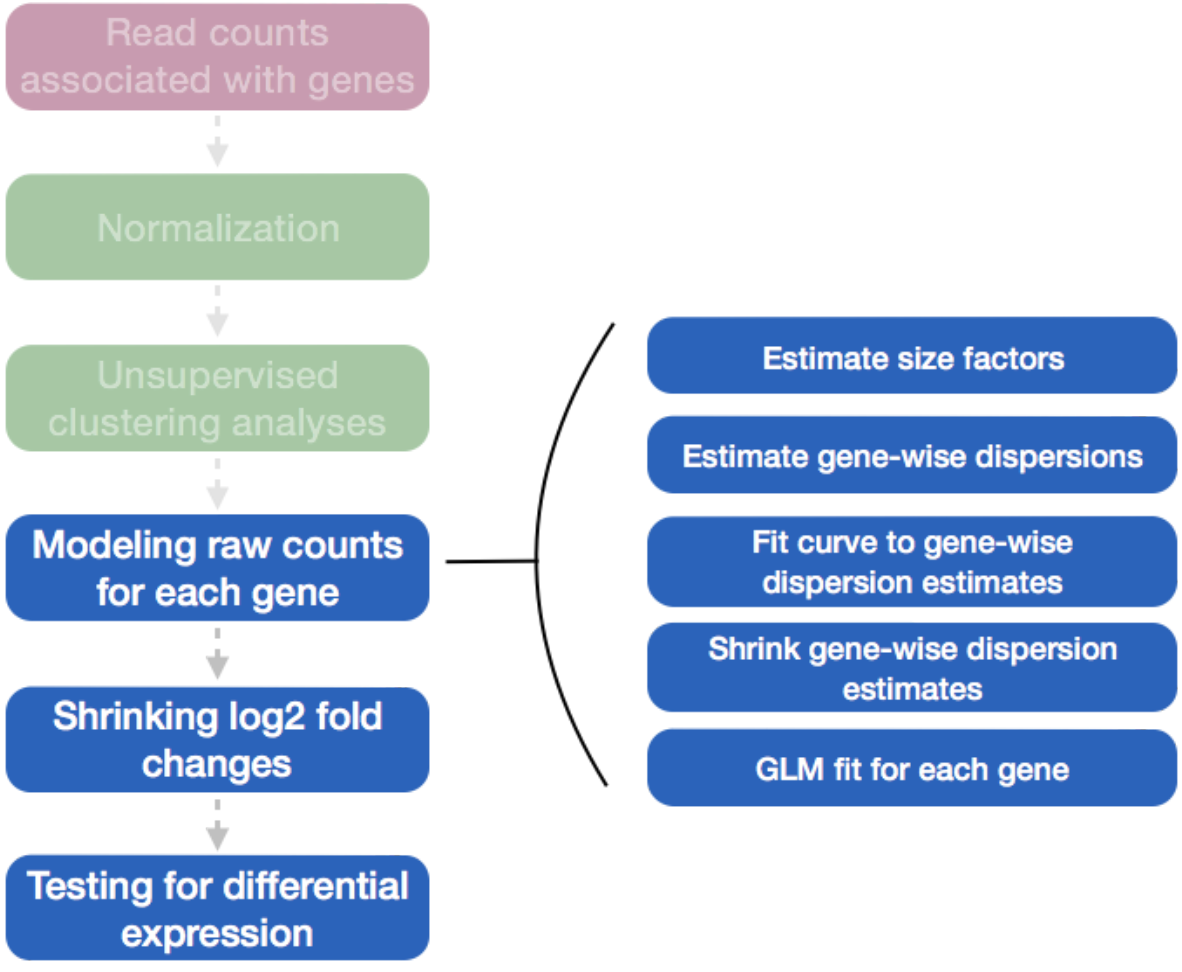
In addition to performing exploratory analyses on normalized measures of expression levels, numerous efforts have been dedicated to optimize statistical tests to decide whether a given genes expression varies between two (or more) conditions.

The final step in the differential expression analysis workflow is fitting the raw counts to the negative binomial model and performing the statistical test for differentially expressed genes. In this step we essentially want to determine whether the mean expression levels of different sample groups are significantly different.



The [DESeq2 paper (https://genomebiology.biomedcentral.com/articles/10.1186/s13059-014-0550-8)](https://genomebiology.biomedcentral.com/articles/10.1186/s13059-014-0550-8) paper was publised in 2014, but the package is continually updated and available for use in R through Bioconductor.It builds on ideas of dispersion estimation and use of Generalized Linear Models form the DSS and edgeR methods.

Differential expression analysis with DESeq2 involves multiple steps as displayed in the flowchart below in blue. Briefly, DESeq2 will model the raw counts, using normalization factors (size factors) to account for differences in library depth. Then, it will estimate the gene-wise dispersion and shrink these estimates to generate more accurate estimates of dispersion to model the counts. Finally, DESeq2 will fit the negative binomial model and perform hypothesis testing using the Wald test or Likelihood Ratio Test.



The two basic tasks of all DGE tools are:

1. Estimate the *magnitude* of differential expression between two or more conditions based on read counts from replicated samples, i.e., calculate the fold change of read counts, taking into account the differences in sequencing depth and variability.
2. Estimate the *significance* of the difference and correct for multiple testing.

The best performing tools tend to be **edgeR, DESeq2** and **limma-voom**. DESeq and limma-voom tend to be more conservative than edgeR (better control of false positives), but edgeR is recommended for experiments with fewer than 12 replicates. These tools are al based on the R language and make heavy use of numerous statistical methods that have been developed and implemented over the past two decades. They all follow the same approach, i.e., they estimate the gene expression difference for a given gene using regression-based models, followed by a statistical test based on the null hypothesis that the difference is close to zero (which would mean that there is no difference in the gene expression values that could be explained by the conditions).

Feature	DESeq2	edgeR	limmaVoom	Cuffdiff
Seq. depth normalization	Sample-wise size factor	Gene-wise trimmed median of means (TMM)	Gene-wise trimmed median of means (TMM)	FPKM-like or DESeq-like
Dispersion estimate	Cox-Reid approximate conditional inference with focus on maximum *individual* dispersion estimate	Cox-Reid approximate conditional inference moderated towards the *mean*	squeezes gene-wise residual variances towards the global variance	
Assumed distribution	Neg. binomial	Neg. binomial	*log*-normal	Neg. binomial
Test for DE	Wald test (2 factors); LRT for multiple factors	exact test for 2 factors; LRT for multiple factors	*t*-test	*t*-test
False positives	Low	Low	Low	High
Detection of differential isoforms	No	No	No	Yes
Support for multi-factored experiments	Yes	Yes	Yes	No
Runtime (3-5 replicates)	Seconds to minutes	Seconds to minutes	Seconds to minutes	Hours

!Note All statistical methods developed for read counts rely on approximations of various kinds, so that assumptions must be made about the data properties. edgeR and DESeq, for example, assume that the majority of the transcriptome is unchanged between the two conditions. If this assumption is not met by the data, both log2 fold change and the significance indicators are most likely incorrect!

Estimating the difference between read counts for a given gene

To determine whether the read count differences between different conditions for a given gene are greater then expected by chance, DGE tools must find a way to estimate that difference. edgeR, DESeq2 and limma-voom all use regression models that are applied to every single gene. Linear regression models usually take the following typical form: $Y = b_0 + b_1 * x + e$

Here, Y will entail *all* read counts (from all conditions) for a given gene. b_0 is called the *intercept*; *x* is the condition (for RNAseq, this is very often a discrete factor, e.g., "WT" or "mutant", or, in mathematical terms, 0 or 1), *e* is a term capturing the error or uncertainty, and b_1 is the coefficient that captures the difference. You may habe encountered linear models in different contexts, the are usually used to predict unknown values of *Y*, i.e., one often wants to find a function that returns *Y* at any given point along a certain trajectory captured by the model. In the case of RNAseq, we are actually more interested in the values that the coefficients take, more precisely, the parameter that explains the difference between groups of expression values belonging to different sets of *x*.

The very simple model shown above could be fitted in R using the function lm(rlog.norm~genotype), which will return estimates for both b_0 and b_1, so that the average expression values of the baseline genotype (e.g., WT=0) would correspond to $Y = b_0 + b_1 * 0 + e$. This is equivalent to $Y = b_0$ (assuming

that e is very small), thereby demonstrating why the intercept (b_0) can be interpreted as the average of our baseline group. b_1, on the other hand, will be the coefficient whose closeness to zero will be evaluated during the statistical testing step.

Instead of using a *linear* model, DESeq2 and edgeR rely on a *negative binomial* model to fit the observed read counts to arrive at the estimate for the difference. Originally, read counts had been modeled using the *Poisson* distribution because:

- individual reads can be interpreted as binary data (Bernoulli trials): they either originate from gene *i* or not
- we are trying to model the discrete probability distribution of the number of successes (success = read is present in the sequenced library).
- the pool of possible reads that could be present is large, while the proportion of reads belonging to gene *i* is quite small.

The convenient feature of a Poisson distribution is that *variance* = *mean*. Thus, if the RNAseq experiment gives us a precise estimate of the mean read counts per condition, we implicitly know what kind of variance to expect for read counts that are not truly changing between two conditions. This, in turn, then allows us to identify those genes that show greater differences between the two conditions than expected by chance.

While read counts of the same library preparation (=technical replicates) can indeed be well approximated by the Poisson distribution, it has been shown that biological replicates habe greater variance(noise) than expected. This *overdispersion* can be captured with the *negative binomial* distribution, which is a more general for of the Poisson distribution that allows the variance to exceed the mean. The square root of the dispersion is the coefficient of variation - SD/mean - after subtraction the variance we expect due to Poisson sampling.

In contrast to the Poisson distribution, we now need to estimate two parameters from the read counts: the mean as well as the dispersion. The precision of these estimates strongly depends on the number (and variation) of replicates - the more replicates, the better the grasp on the underlying mean expression values of unchanged genes and the variance that is due to biological variation rather than the experimental treatment. For most RNAseq experiments, only two to three replicates are available, which is not enough for reliable mean and variance estimates. Some tools therefore compensate for the lack of replication by borrowing information across genes with similar expression values and shrink a given gene's variance towards the regressed values. These fitted values of the mean and disperion are then used instead of the raw estimates to test for differential gene expression.

Testing the null hypothesis

The null hypothesis is that there is no systematic difference between the average read count values of the different conditions for a given gene. Which test is used to assign a p-value again depends on the tool, but generally you can think of them as some variation of the well-known t-test or ANOVA's.

Once you've obtained a list of p-values for all the genes of your data set, it is important to realize that you just performed the same type of test for thousands and thousands of genes. That means, that even if you decide to focus on genes with a p-value smaller than 0,05, if you've looked at 10000 genes your final list may contain 0.05 * 10000 = 500 false postive hits. To guard yourself against this, all the tools will offer some sort of correction for the multiple hypotheses you tested, e.g. in the form of the Benjamin-Hochberg fromula. You should should definitely rely on the adjusted p-values rather than the original ones to zoom into possible candidates for downstream analyses and follow-up studies.

Running DGE analysis tools

Prior to performing the differential expression analysis, it is a good idea to know what **sources of variation** are present in your data, either by exploration during QC and/or prior knowledge. Once you know the major sources of variation, you can remove them prior to analysis or control for them in the statistical model by including them in your **design formula**.

Design formula

A design formula tells the statistical software the known sources of variation to control for, as well as, the factor of interest to test for during differential expression testing. For example, if you know that sex is a significant source of variation in your data, then *sex* should be included in your model. **The design formula should have all of the factors in your metadata that account for major sources of variation in your data. The last factor entered in the formula should be the condition of interest.**

For example, suppose you have the following metadata:

	sex	age	litter	treatment
sample1	M	11	1	Ctrl
sample2	M	13	2	Ctrl
sample3	M	11	1	Treat
sample4	M	13	1	Treat
sample5	F	11	1	Ctrl
sample6	F	13	1	Ctrl
sample7	F	11	1	Treat
sample8	F	13	2	Treat

If you want to examine the expression differences between treatments,and you know that major sources of variation include *sex* and *age*, then your design formula would be:

```
design  <- ~sex + age + treatment
```

The tilde (~) should always proceed your factors and tells DESeq2 to model the counts using the following formula. Note the **factors included in the design formula need to match the column names in the metadata**.

Complex designs

DESeq2 also allows for the analysis of complex designs. You can explore interactions or 'the diference of differences' by specifying for it in the design formula. For example, if you want to explore the effect of sex on the treatment effect, you could specify for it in the design formula as follows:

```
design  <- ~sex + age + treatment + sex:treatment
```

Since the interaction term sex:treatment is last in the formula, the results output from DESeq2 will output results for this term.

DESeq2 workflow

For the Colo example data set, we would like to compare the effect of parental versus resistent cell line samples, with the parental values as the denominator for the fold change calculation.

In [50]:
```
# DESeq2 uses the levels of the condition to determine the order of the co
str(colData(DESeq.ds)$condition)
```

```
Factor w/ 2 levels "ColoP","ColoR": 1 1 1 1 2 2 2 2
```

In [51]:
```
# set ColoP as the first-level-factor
colData(DESeq.ds)$condition <- relevel(colData(DESeq.ds)$condition, "ColoP
```

In [52]:
```
# run analysis
DESeq.ds <- DESeq(DESeq.ds)
```

```
using pre-existing size factors
estimating dispersions
gene-wise dispersion estimates
mean-dispersion relationship
final dispersion estimates
fitting model and testing
```

With the 2 lines of code above the workflow for the differential gene expression analysis with DESeq2 is completed. To better understand each of the steps I will be taking a detailed look at each of them.

Step 1: Estimate size factors

The first step in the differential expression analysis is to estimate the size factors, which is exactly what we did to normalize the raw counts. To normalize the count data, DESeq2 calculates size factors for each sample using the *median of ratios method*

Step2: Estimate gene-wise dispersion

The next step in the differential expression analysis is the estimation of gene-wise dispersions. Before we get into the details, we should have a good idea about what dispersion is refering to in DESeq2.

What is dispersion? Dispersion is a measure of spread or variability in the data. Variance, standard deviation, IQR, among other measures, can all be used to measure dispersion. However, DESeq2 uses a specific measure of dispersion (α) related to the mean(μ) and variance of the data: var = μ + α * μ^2. For genes with moderate to high count values, the square root of dispersion will be equal to the coefficient of variation (var/μ). So 0.01 dispersion means 10% variation around the mean expected across biological replicates.

What does the DESeq2 dispersion represent? The DESeq2 dispersion estimates are **inversely related to the mean** and **directly related to variance. Based on this relationship, the dispersion is higher for small mean counts and lower for large mean counts.** The dispersion estimates for genes with the same mean will differ only based on their variance. **Therefore, the dispersion estimates reflect the variance in gene expression for a given mean value.**

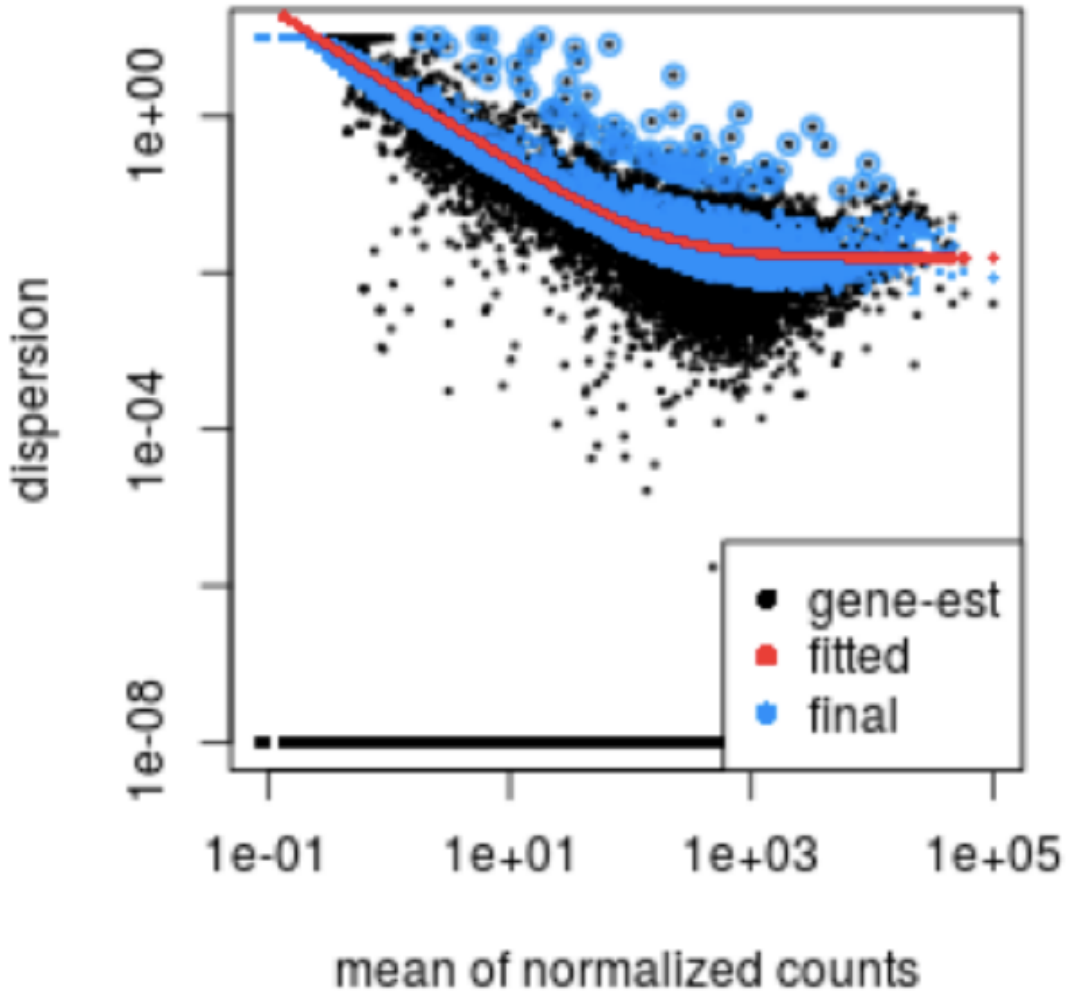
How does the dispersion relate to our model? To accurately model sequencing counts, we need to generate accurate estimates of within-group variation (variation between replicate of the same sample group) for each gene. With only a few (3-6) replicates per group, the **estimates of variation for each gene are often unreliable**(due to the large differences in dispersion for genes with similar means).

To address this problem, DESeq2 **shares information across genes** to generate more accurate estimates of variation based on the mean expression level of the gene using a method called 'shrinkage'. **DESeq2 assumes that genes with similar expression levels have similar dispersion.

To model the dispersion based on expression level (mean counts of replicates), the dispersion for each gene is estimated using maximum likelihood estimation. In other words, **given the count values of the replicates, the most likely estimate of dispersion is calculated.**

Step3: Fit curve to gene-wise dispersion estimates

The next step in the workflow is to fit a curve to the dispersion estimates for each gene. The idea behind fitting a curve to the data is that different genes will have different scales of biological variability, but, over all genes, there will be a distribtution of reasonable estimates of dispersion.This curve is displayed as a red line in the figure below, which plots the estimate for the **expected dispersion value for genes of a given expression strenght**. Each black dot is a gene with an associated mean expresssion level and maximum likelihood estimation (MLE) of the dispersion.



Step4: Shrink gene-wise dispersion estimates toward the values predicted by the curve

The next step in the workflow is to shrink the gene-wise dispersion estimates toward the expected dispersion values. The curve allows for more accurate identification of differentially expressed genes when sample sizes are small, and the strength of the shrinkage for each gene depends on:

- how close gene dispersion are from the curve
- sample size (more samples = less shrinkgage)

This shrinkage method is particularly important to reduce false positives in the differential expression analysis. Genes with low dispersion estimates are shrunken towards the curve, and the more accurate, higher shrunken values are output for fitting of the model and differential expression testing.

Dispersion estimates that are slightly above the curve are also shrunk toward the curve for better dispersion estimation; however, genes with extremly high dispersion values are not. This is due to the likelihood that the gene does not follow the modeling assumptions and has higher variability than others for biological or technical reasons. Shrinking the values toward the curve could result in false positives, so these values are not shrunken. These genes are shown surronded by blue circles.

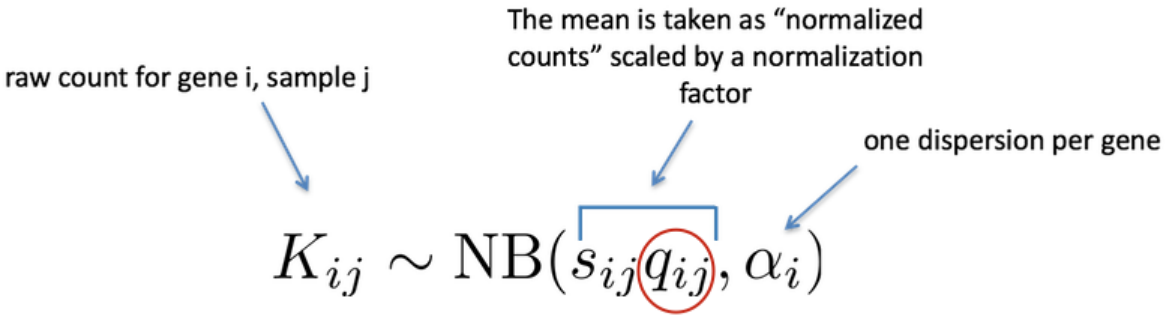
This is a good plot to examine to ensure your data is good fit for the DESeq2 model. You expect your data to generally scatter around the curve, with the dispersion decreasing with increasing mean expression levels. If you see a cloud or different shapes, then you might want to explore your data more to see if you have contamination or outlier samples.

!Note The input for the DGE analysis are the *raw* read counts(untransformed, not normalized for sequencing depth).

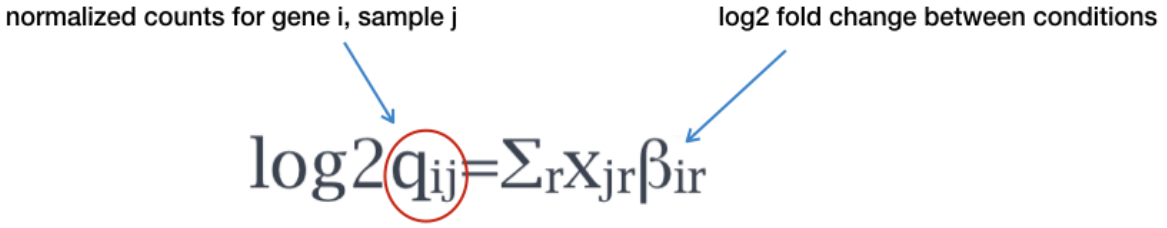
Step5: Model fitting and hypothesis testing

The final step in the DESeq2 workflow is fitting the Negative Binomial model for each gene and performing differential expression testing.

As discussed above, the count data generated by RNAseq exhibits overdispersion(variance>mean) and the statistical distribution used to model the counts needs to account for this overdispersion. DESeq2 uses a negative binomial distribution to model RNAseq counts using the equation below:



Modeling is a mathematically formalized way to approximate how the data behaves given a set pf parametes (i.e. size factor, dispersion). DESeq2 will use this formula as our model for each gene and fit the normalized count data to it. After the model is fit, coefficients are estimated for each sample group along with their standard error using the formula:



The coefficients are the estimates for the **log2 foldchanges** for each sample group. However, these estimates do not account for the large dispersion we observer with low read counts. To avoid this, the **log2 fold changes calculated by the model need to be adjusted.**

Shrunken log2 foldchanges (LFC)

To generate more accurate log2 foldchanges estimates, DESeq2 allows fot the **shrinkage of the LFC estimates toward zero** when the information for a gene is low, which could include:

- Low counts
- High dispersion values

As with the shrinkage of dispersion estimates, LFC shrinkage uses **information from all genes** to generate more accurate estimates.

To generate the shrunken log2 fold change estimates, you have to run an additional step on your results object with the function lfcShrink().

Hypothesis testing using the Wald test

The first step in hypothesis testing ist to set up a **null hypothesis** for each gene. In our case is, the null hypothesis that there is **no differential expression across the two sample groups(LFC=0)**. Notice that we can do this without observing any data, because it is based on a thought experiment. Second, we use a statistical test to determine if based on the observerd data, the null hypothesis is true.

With DESeq2, the Wald test is commonly used for hypothesis testing when compering two groups. A Wald test statistic is computed along with a probability that a test statistic at least as extreme as the observerd value were selected at random. This probability is called the p-value of the test. If the p-value is smaller we reject the null hypothesis and state that there is evidence against the null (i.e. the gene is differentially expressed).

Building the results table

The results table looks very much like a dataframe and in many ways it can be treated like one (i.e. when accessing/subsetting data). However, it is important to recognize that it is actually stored in a **DESeqResults** object. When we start visualizing our data, this information will be helpful.

In [54]:
```
res <- results(DESeq.ds)
res
```

```
log2 fold change (MLE): condition ColoR vs ColoP
Wald test p-value: condition ColoR vs ColoP
DataFrame with 14980 rows and 6 columns
                     baseMean     log2FoldChange             lfcSE
                    <numeric>          <numeric>         <numeric>
WASH7P        0.133780342349602  0.665050532904946   3.5338115409304
LOC729737      9.17720820896479   1.13771101427527  1.26619216142909
LOC100288069   5.95251706100896    5.2919073359345  1.91099703840421
LINC00115      2.92587670445247  -1.55966379292737  2.09351876640169
LOC643837      14.4153610520602   -1.6370537399031 0.943702930948073
...                         ...                ...               ...
CD24           2094.14935267275   1.16029432399119 0.179077578467571
TXLNG2P        68.8042737275944  -2.55247829342476 0.519417340513556
KDM5D          10.7071546375337 -0.494625510166897 0.993312988782063
EIF1AY         284.155746281196  -1.22750178220709 0.424478122989731
RPS4Y2       0.0961662911050586  0.665050532904946   3.5338115409304
                           stat               pvalue                 padj
                      <numeric>            <numeric>            <numeric>
WASH7P         0.188196378103924    0.850722711616258                   NA
LOC729737      0.898529503603302    0.368903323276919    0.644639231962783
LOC100288069    2.76918657098157  0.00561964448740934   0.0437544803911412
LINC00115     -0.744996327693827    0.456273939906555    0.714971496143793
LOC643837      -1.73471299729722   0.0827916530049437    0.275392324857151
...                          ...                  ...                  ...
CD24            6.47928307898863 9.21594569536474e-11 1.07291162076322e-08
TXLNG2P         -4.9141183675175 8.91828595898381e-07 3.44990910008285e-05
KDM5D         -0.497955343132455    0.618515518201271    0.824867302475707
EIF1AY         -2.89179044979141  0.00383053342400087    0.032802859837076
RPS4Y2         0.188196378103924    0.850722711616258                   NA
```

In [312]:
```
class(res)
```

'DESeqResults'

In [56]:
```
# as res is a DataFrame object, it carries metadata
mcols(res, use.names = TRUE)
```

```
DataFrame with 6 rows and 2 columns
                     type                                        description
                <character>                                        <character>
baseMean       intermediate        mean of normalized counts for all samples
log2FoldChange      results log2 fold change (MLE): condition ColoR vs ColoP
lfcSE               results         standard error: condition ColoR vs ColoP
stat                results         Wald statistic: condition ColoR vs ColoP
pvalue              results      Wald test p-value: condition ColoR vs ColoP
padj                results                           BH adjusted p-values
```

The first column, baseMean, is a just the average of the normalized count values, divided by the size factors, taken over all samples in the DESeqDataSet.

The column log2FoldChange is the effect size estimate. It tells us how much the gene's expression seems to have changed.This value is reported on a logarithmic scale to base 2: for example, a log2 fold change of 1.5 means that the gene's expression is increased by a multiplicative factor of 2 exp(1.5)≈2.82.

Of course, this estimate has an uncertainty associated with it, which is available in the column lfcSE, the standard error estimate for the log2 fold change estimate. We can also express the uncertainty of a particular effect size estimate as the result of a statistical test. The purpose of a test for differential expression is to test whether the data provides sufficient evidence to conclude that this value is really different from zero. DESeq2 performs for each gene a hypothesis test to see whether evidence is sufficient to decide against the null hypothesis that there is zero effect of the treatment on the gene and that the observed difference between treatment and control was merely caused by experimental variability (i.e., the type of variability that you can expect between different samples in the same treatment group). As usual in statistics, the result of this test is reported as a p value, and it is found in the column pvalue. Remember that a p value indicates the probability that a fold change as strong as the observed one, or even stronger, would be seen under the situation described by the null hypothesis.

We can also summarize the results with the following line of code, which reports some additional information, that will be covered in later sections.

In [55]:
```
# alpha set as significance cutoff used for optimizing the independent fil
DGE.results <- results(DESeq.ds, independentFiltering = TRUE, alpha = 0.05
summary(DGE.results)
```

```
out of 14980 with nonzero total read count
adjusted p-value < 0.05
LFC > 0 (up)       : 906, 6%
LFC < 0 (down)     : 743, 5%
outliers [1]       : 142, 0.95%
low counts [2]     : 2895, 19%
(mean count < 3)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

In [57]:
```
table(DGE.results$padj < 0.05)
```

```
FALSE  TRUE
10294  1649
```

If we want to raise the log2 fold change threshold, so that we test for genes that show more substantial changes due to treatment, we simply supply a value on the log2 scale. For example, by specifying lfcThreshold = 1, we test for genes that show significant effects of treatment on gene counts more than doubling or less than halving, because 21=2.

In [58]:
```
resLFC1 <- results(DESeq.ds, lfcThreshold=1)
table(resLFC1$padj < 0.1)
```

```
FALSE  TRUE
12640   165
```

In [59]:
```
# the DESeqResult object can basically be handled like a data.frame
head(DGE.results)
```

```
log2 fold change (MLE): condition ColoR vs ColoP
Wald test p-value: condition ColoR vs ColoP
DataFrame with 6 rows and 6 columns
                    baseMean     log2FoldChange             lfcSE
                   <numeric>          <numeric>         <numeric>
WASH7P       0.133780342349602  0.665050532904946   3.5338115409304
LOC729737     9.17720820896479   1.13771101427527  1.26619216142909
LOC100288069  5.95251706100896    5.2919073359345  1.91099703840421
LINC00115     2.92587670445247  -1.55966379292737  2.09351876640169
LOC643837     14.4153610520602   -1.6370537399031 0.943702930948073
NOC2L         103.448372511829 -0.859624158494743 0.383317185339642
                        stat              pvalue               padj
                   <numeric>           <numeric>          <numeric>
WASH7P        0.188196378103924   0.850722711616258                 NA
LOC729737     0.898529503603302   0.368903323276919  0.643990357554685
LOC100288069   2.76918657098157 0.00561964448740934 0.0427486714096368
LINC00115    -0.744996327693827   0.456273939906555                 NA
LOC643837     -1.73471299729722  0.0827916530049437  0.272172265948711
NOC2L         -2.24259227441907  0.0249231188923105  0.125065886105405
```

Multiple test correction

Note that we have p-values and p-adjusted values in the output. Which should use to identify significantly differentially expressed genes?

If we used the p-value directly from the Wald test with a significance cut-off of p<0.05, that means there is a 5% chance it is false positives. Each p-value is the result of a single test (single gene). The more genes we test, the more we inflate the false positive rate. **This is the multiple testing problem**. For example, if we test 20000 genes for differential expression, at p<0.05 we would expect to find 1000 genes by chance. If we found 3000 genes to be differentially expressed total, roughly one third of our genes are false positives. We would not want to sift through our "significant" genes to identify which ones are true positives.

DESeq2 helps reduce the number of genes tested by removing those genes unlikely to be significantly DE prior to testing, such as those with low number of counts and outlier samples (gene-level QC). However, we still need to correct for multiple testing to reduce the number of false positives, and there are a few common approaches:

- **Bonferroni:** The adjusted p-value is calculated by: p-value * m (m=total number of tests). **This is a very conserative approach with a high probability of false negatives**, so is generally not recommended.
- **FDR/Benjamini-Hochberg:** Benjamini and Hochberg (1995) defined the concept of FDR and created an algorithm to control the expected FDR below a specified level given a list of independent p-values. **An interpretation of the BH method for controlling the FDR is implemented in DESeq2 in which we rank the genes by p-value, then multiply each ranked p-value by m/rank.**
- **Q-value/Storey method:** The minimum FDR that can be attained when calling that feature significant. For example, if gene X has a q-value of 0.013 it means that 1.3% of genes that show p-values at least as small as gene X are false positives

In DESeq2, the p-values attained by the Wald test are corrected for multiple testing using the Benjamini and Hochberg method by default. There are options to use other methods in the results() function. The p-adjusted values should be used to determine significant genes. The significant gene can be output for visualization and/or functional analysis.

So what does FDR < 0.05 mean? By setting the FDR cutoff to <0.05, we're saying that the proportion of false positives we expect amongst our differentially expressed genes is 5%. For example, if you call 500 genes as differentially expressed with an FDR cutoff of 0.05, you expect 25 of them to be false positives.

In [63]:
```r
# how many genes are significantly expressed
sum(res$pvalue < 0.05, na.rm=TRUE)
```

3040

In [64]:
```r
# how many genes are not significantly expressed
sum(!is.na(res$pvalue))
```

14838

Now, assume for a moment that the null hypothesis is true for all genes, i.e., no gene is affected by the treatment with dexamethasone. Then, by the definition of the p value, we expect up to 5% of the genes to have a p value below 0.05. This amounts to 741 genes. If we just considered the list of genes with a p value below 0.05 as differentially expressed, this list should therefore be expected to contain up to 741 / 3040 = 24% false positives.

DESeq2 uses the Benjamini-Hochberg (BH) adjustment (Benjamini and Hochberg 1995) as implemented in the base R p.adjust function; in brief, this method calculates for each gene an adjusted p value that answers the following question: if one called significant all genes with an adjusted p value less than or equal to this gene's adjusted p value threshold, what would be the fraction of false positives (the false discovery rate, FDR) among them, in the sense of the calculation outlined above? These values, called the BH-adjusted p values, are given in the column padj of the res object.

The FDR is a useful statistic for many high-throughput experiments, as we are often interested in reporting or focusing on a set of interesting genes, and we would like to put an upper bound on the percent of false positives in this set.

Hence, if we consider a fraction of 10% false positives acceptable, we can consider all genes with an adjusted p value below 10% = 0.1 as significant. How many such genes are there?

In [65]:
```r
# how many genes are significantly expressed considering a 10%FDR
sum(res$padj < 0.1, na.rm=TRUE)
```

2124

We subset the results table to these genes and then sort it by the log2 fold change estimate to get the significant genes with the strongest down-regulation:

In [66]:
```r
# with strongest down-regulation
resSig <- subset(res, padj < 0.1)
head(resSig[ order(resSig$log2FoldChange), ])
```

```
log2 fold change (MLE): condition ColoR vs ColoP
Wald test p-value: condition ColoR vs ColoP
DataFrame with 6 rows and 6 columns
               baseMean    log2FoldChange             lfcSE                stat
              <numeric>         <numeric>         <numeric>           <numeric>
MZB1    50.1118067050585 -9.16578643157023 1.14206665368006 -8.02561426869045
PKIA    14.1084086698905 -7.33958124893185 1.47834332523188 -4.96473391779992
SNORA31 12.0006432093164 -7.10154500006678 1.50852736304281 -4.70760105122817
HABP2   11.6108240969898 -7.05523769892897 1.43247954222882 -4.92519264041398
WWC2    17.9461224042872 -6.95084252788737 1.33019949548687 -5.22541359508131
UGT2B4  10.5363535392266 -6.91447423196668  1.5264540079419 -4.52976257128727
                     pvalue                 padj
                  <numeric>            <numeric>
MZB1    1.01019174081494e-15 2.52011915096364e-13
PKIA    6.87953234260274e-07 2.80318011186586e-05
SNORA31 2.50649037299575e-06 8.58244770854345e-05
HABP2   8.42772851566214e-07 3.31738489817774e-05
WWC2    1.73766251769905e-07 8.32987710445225e-06
UGT2B4  5.90500073030719e-06 0.000175200798367173
```

In [67]:
```
# with strongest up-regulation
head(resSig[ order(resSig$log2FoldChange, decreasing = TRUE), ])
```

```
log2 fold change (MLE): condition ColoR vs ColoP
Wald test p-value: condition ColoR vs ColoP
DataFrame with 6 rows and 6 columns
                    baseMean   log2FoldChange            lfcSE
                   <numeric>        <numeric>        <numeric>
KIF6         56.8902915129279 9.29220239736187 1.14816835484566
GMPR          42.413897001456 8.86700110786199  1.3010016754105
LOC100127888 149.666566092671 8.74709173840927 1.33003060215044
SLMO1        28.2517488410203 8.28357801534713 1.27945692085089
ALDH1A2      26.5480947345279 8.19290503939129 3.13226649633766
PELI2        26.0431917533093 8.16348800833352   1.266198013784
                        stat               pvalue                 padj
                   <numeric>            <numeric>            <numeric>
KIF6         8.09306610667823 5.81812674360429e-16 1.48168294403789e-13
GMPR         6.81551859267529 9.39243249986429e-12  1.4001596936383e-09
LOC100127888 6.57660938347331 4.81296969231484e-11 6.00344301212822e-09
SLMO1         6.474292240991 9.52573791187807e-11 1.09355414508712e-08
ALDH1A2       2.6156475028452  0.00890584454886572   0.0616449851445836
PELI2         6.4472443642027 1.13902062401393e-10 1.27737505577489e-08
```

Summarizing results

To summarize the results table, a handy function in DESeq2 is *summary()*. This function when called with a DESeq results table as input, will summarize the results using the alpha threshold: FDR<0.05(padj/FDR is used even though the output says p-value < 0.05).

In [314]:
```
#summarize results
summary(res)
```

```
out of 14980 with nonzero total read count
adjusted p-value < 0.1
LFC > 0 (up)       : 1178, 7.9%
LFC < 0 (down)     : 946, 6.3%
outliers [1]       : 142, 0.95%
low counts [2]     : 2614, 17%
(mean count < 3)
[1] see 'cooksCutoff' argument of ?results
[2] see 'independentFiltering' argument of ?results
```

In addition to the number of genes up- and down-regulated at the default threshold, **the function also reports the number of genes that were tested (genes with non-zero total read count), and then number of genes not included in multiple test correction due to a low mean count.

Extracting significantly differentially expressed genes

What we noticed is that the FDR threshold on it's own doesn't appear to be reducing the number of significant genes. With large significant gene lists it can be hard to extract meaningful biological relevance. To help increase stingency, one con also **add a fold change threshold**. The summary() function doesn't have na argument for hold change threshold.

Let's first create variables that contain our threshold criteria:

In [315]:
```
# set thresholds
padj.cutoff <- 0.05
lfc.cutoff <- 0.58
```

The ldc.cutoff is set to 0.58; remember that we are working with log2 fold changes so this translates to an

actual fold change of 1.5 which is pretty reasonablle.

We can easily subset the results table to only include those that are significant using the *filter()* function, but first we will convert the results table into a tibble:

In [316]:
```
res_tableOE_tb <- res %>%
  data.frame() %>%
  rownames_to_column(var="gene") %>%
  as_tibble()
```

Now we can subset that table to only keep the significant genes using our pre-defined thresholds:

In [317]:

```
sigOE <- res_tableOE_tb %>%
        filter(padj < padj.cutoff & abs(log2FoldChange) > lfc.cutoff)
sigOE
```

A tibble: 1432 × 8

gene	baseMean	log2FoldChange	lfcSE	pvalue	padj	emsembl	entr
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<ch
C1orf159	53.54362	-1.4428782	0.6033942	1.148088e-03	1.301876e-02	ENSG00000131591	549
SDF4	768.17766	-1.1073720	0.1997889	4.286061e-09	3.274550e-07	ENSG00000078808	511
B3GALT6	276.37433	-1.1803212	0.2233851	1.652512e-08	1.080230e-06	ENSG00000176022	1267
UBE2J2	321.66968	-0.9738865	0.2356951	6.087835e-06	1.797529e-04	ENSG00000160087	1184
PUSL1	168.32943	-1.6315269	0.4713837	3.922649e-05	8.782135e-04	ENSG00000169972	1267
CPSF3L	143.20372	-1.2230809	0.3219479	1.644344e-05	4.193502e-04	NA	N
GLTPD1	52.60110	-0.9830747	0.5197426	6.604096e-03	4.958751e-02	NA	N
AURKAIP1	2619.38943	-1.2693385	0.1486322	1.672987e-18	5.680721e-16	ENSG00000175756	549
MRPL20	564.89691	-1.0036337	0.2179828	6.861881e-07	2.803180e-05	ENSG00000242485	550
ATAD3A	278.26067	-0.7929204	0.3323223	3.471442e-03	3.044111e-02	ENSG00000197785	552
SSU72	1053.36236	-0.6398395	0.1748561	8.773342e-05	1.694239e-03	ENSG00000160075	291
GNB1	542.66968	-1.2668507	0.2378892	1.186362e-08	8.193273e-07	ENSG00000078369	27
RER1	1175.71949	-1.1523870	0.2154719	1.195933e-08	8.212969e-07	ENSG00000157916	110
LRRC47	185.27518	-1.6848016	0.3650407	3.056679e-07	1.394211e-05	ENSG00000130764	574
C1orf174	193.94973	-0.7672243	0.2602130	7.627397e-04	9.333064e-03	ENSG00000198912	3394
KCNAB2	258.78508	-1.2868853	0.2943322	1.331306e-06	4.961550e-05	ENSG00000069424	85
ACOT7	1052.75484	-0.7968131	0.2057374	2.578348e-05	6.093469e-04	ENSG00000097021	113
ENO1	21538.51606	-1.0609730	0.1952750	9.907426e-09	6.960228e-07	ENSG00000074800	20
SLC25A33	281.95668	-1.0247748	0.3102118	1.385279e-04	2.473394e-03	ENSG00000171612	842
CTNNBIP1	501.11241	-0.8590725	0.2134744	1.183762e-05	3.208493e-04	ENSG00000178585	569
UBE4B	61.29269	-1.3851281	0.6297598	1.891524e-03	1.917247e-02	ENSG00000130939	102
PGD	779.25862	-0.9423126	0.1687678	4.533026e-09	3.420476e-07	ENSG00000142657	52
TNFRSF1B	347.21992	-0.8667455	0.2023970	3.931671e-06	1.254850e-04	ENSG00000028137	71
DHRS3	285.19948	-1.7768240	0.3231054	2.848930e-09	2.276165e-07	ENSG00000162496	92

gene	baseMean	log2FoldChange	lfcSE	pvalue	padj	emsembl	entr
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<ch
EFHD2	1009.60730	-0.7571524	0.1699574	2.291617e-06	7.980832e-05	ENSG00000142634	7918
SPEN	64.06980	-1.5134823	0.6651034	1.389525e-03	1.497845e-02	ENSG00000065526	2301
SDHB	1266.42211	-0.7230791	0.1592726	1.647405e-06	5.975633e-05	ENSG00000117118	639
UBR4	561.82755	-1.4242119	0.2744462	2.086267e-08	1.328256e-06	ENSG00000127481	2335
MRTO4	557.93110	-1.2634950	0.2086533	1.702427e-10	1.841634e-08	ENSG00000053372	5115
AKR7A3	151.70843	-2.9084293	0.3721781	2.947537e-16	7.832760e-14	ENSG00000162482	2297
⋮	⋮	⋮	⋮	⋮	⋮	⋮	
SH3BGRL	528.94070	1.9981721	0.2788476	5.775335e-14	1.283594e-11	ENSG00000131171	645
POF1B	62.76912	1.7453550	0.4155094	2.023155e-06	7.147702e-05	ENSG00000124429	7998
GLA	401.34750	0.7434275	0.2414326	5.406713e-04	7.137328e-03	ENSG00000102393	271
TCEAL4	646.84674	0.9243036	0.2220873	5.967948e-06	1.766397e-04	ENSG00000133142	7992
RNF128	650.17746	1.1569114	0.2905947	8.751385e-06	2.487836e-04	ENSG00000133135	7958
MORC4	625.54468	2.1258491	0.2495216	1.155575e-18	4.154630e-16	ENSG00000133131	7971
NUP62CL	141.74069	1.0462964	0.4617539	2.701775e-03	2.511520e-02	ENSG00000198088	5483
PLS3	153.65393	0.8995375	0.3472517	1.611086e-03	1.690464e-02	ENSG00000102024	535
IL13RA1	95.53785	1.2327130	0.7078868	5.478498e-03	4.284655e-02	ENSG00000131724	359
LAMP2	893.56485	2.2714062	0.2487791	4.678373e-21	1.972015e-18	ENSG00000005893	392
C1GALT1C1	426.19907	0.8479498	0.2037679	7.009173e-06	2.040003e-04	ENSG00000171155	2907
GPC4	70.33314	1.3563669	0.4999420	5.454157e-04	7.192191e-03	ENSG00000076716	223
HMGB3	659.83428	-1.0366300	0.3081328	1.088602e-04	2.013173e-03	ENSG00000029993	314
CETN2	510.92273	0.9896688	0.2979677	1.411027e-04	2.481783e-03	ENSG00000147400	106
FAM58A	424.25614	0.8038125	0.2383272	1.728897e-04	2.919066e-03	NA	N
BCAP31	7799.06885	0.8927730	0.1344006	7.121329e-12	1.101913e-09	ENSG00000185825	1013
ABCD1	50.01882	1.5784980	0.4952136	1.068743e-04	1.982446e-03	ENSG00000101986	21
SSR4	5261.72137	0.8151605	0.2032837	1.333346e-05	3.550940e-04	ENSG00000180879	674
ARHGAP4	181.79398	0.8127050	0.3178915	2.177576e-03	2.141488e-02	ENSG00000089820	39
EMD	375.08185	0.7384596	0.2544750	9.604321e-04	1.124902e-02	ENSG00000102119	201
ATP6AP1	1707.33330	1.5675756	0.2432736	1.120256e-11	1.606781e-09	ENSG00000071553	53

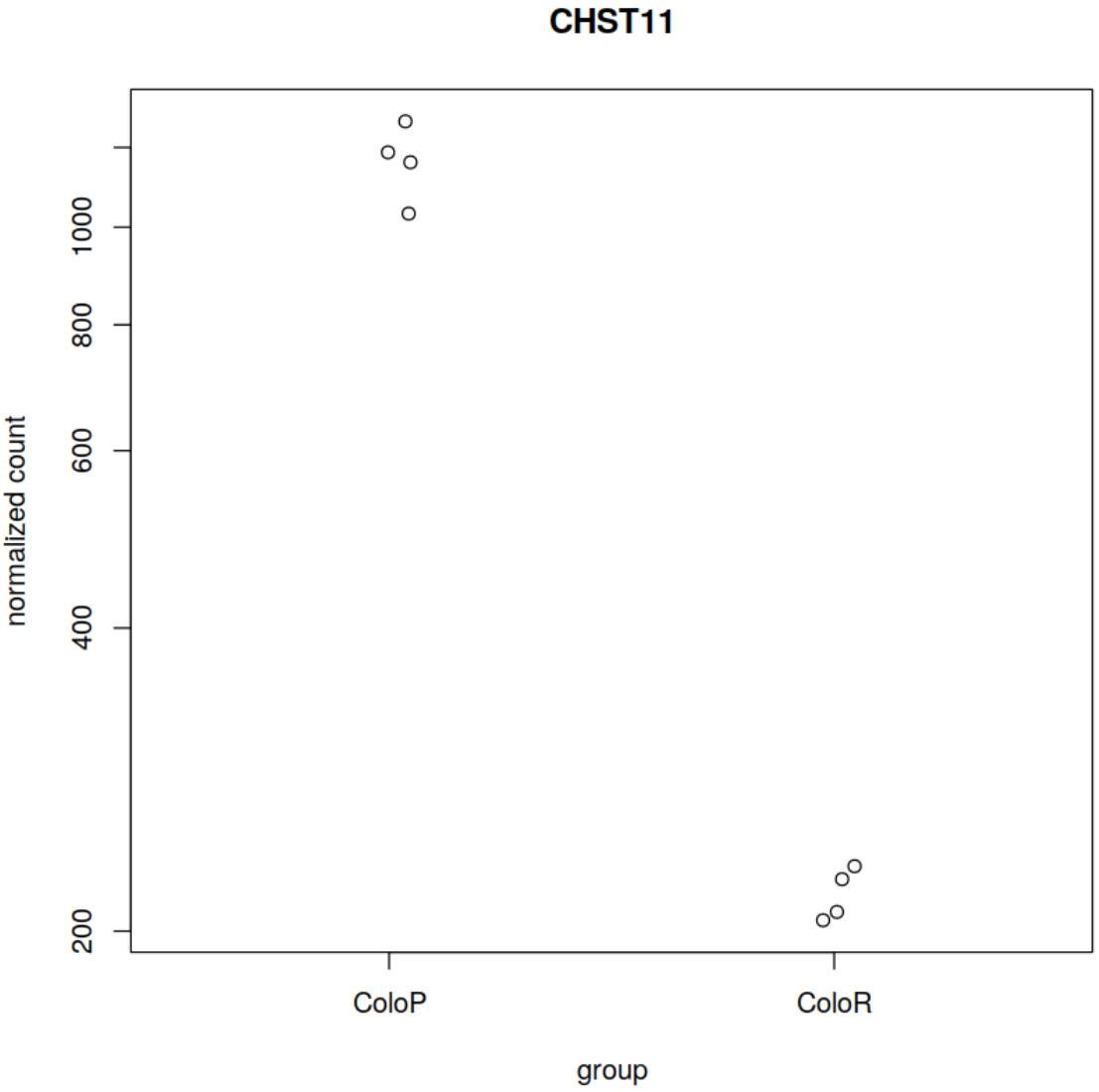
gene	baseMean	log2FoldChange	lfcSE	pvalue	padj	emsembl	entr
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<ch
GDI1	50.18781	1.7444650	0.4923626	2.759612e-05	6.433084e-04	ENSG00000203879	26(
FAM50A	781.88015	0.7457304	0.2185646	1.717073e-04	2.909237e-03	ENSG00000071859	91:
FAM3A	222.11348	1.0744692	0.2790323	1.667482e-05	4.237691e-04	ENSG00000071889	603
G6PD	386.81733	1.0336655	0.2793695	3.276241e-05	7.471786e-04	ENSG00000160211	25:
MPP1	279.54684	-2.0163512	0.2726551	9.922038e-15	2.332442e-12	ENSG00000130830	43:
FUNDC2	559.18254	0.7307175	0.2582112	1.222915e-03	1.360228e-02	ENSG00000165775	659
CD24	2094.14935	1.1179624	0.1807600	9.215946e-11	1.072912e-08	ENSG00000272398	1001339
TXLNG2P	68.80427	-2.3374237	0.5407256	8.918286e-07	3.449909e-05	NA	N
EIF1AY	284.15575	-0.9710376	0.4478429	3.830533e-03	3.280286e-02	ENSG00000198692	908

Exploratory plots following DGE analysis

Counts plot

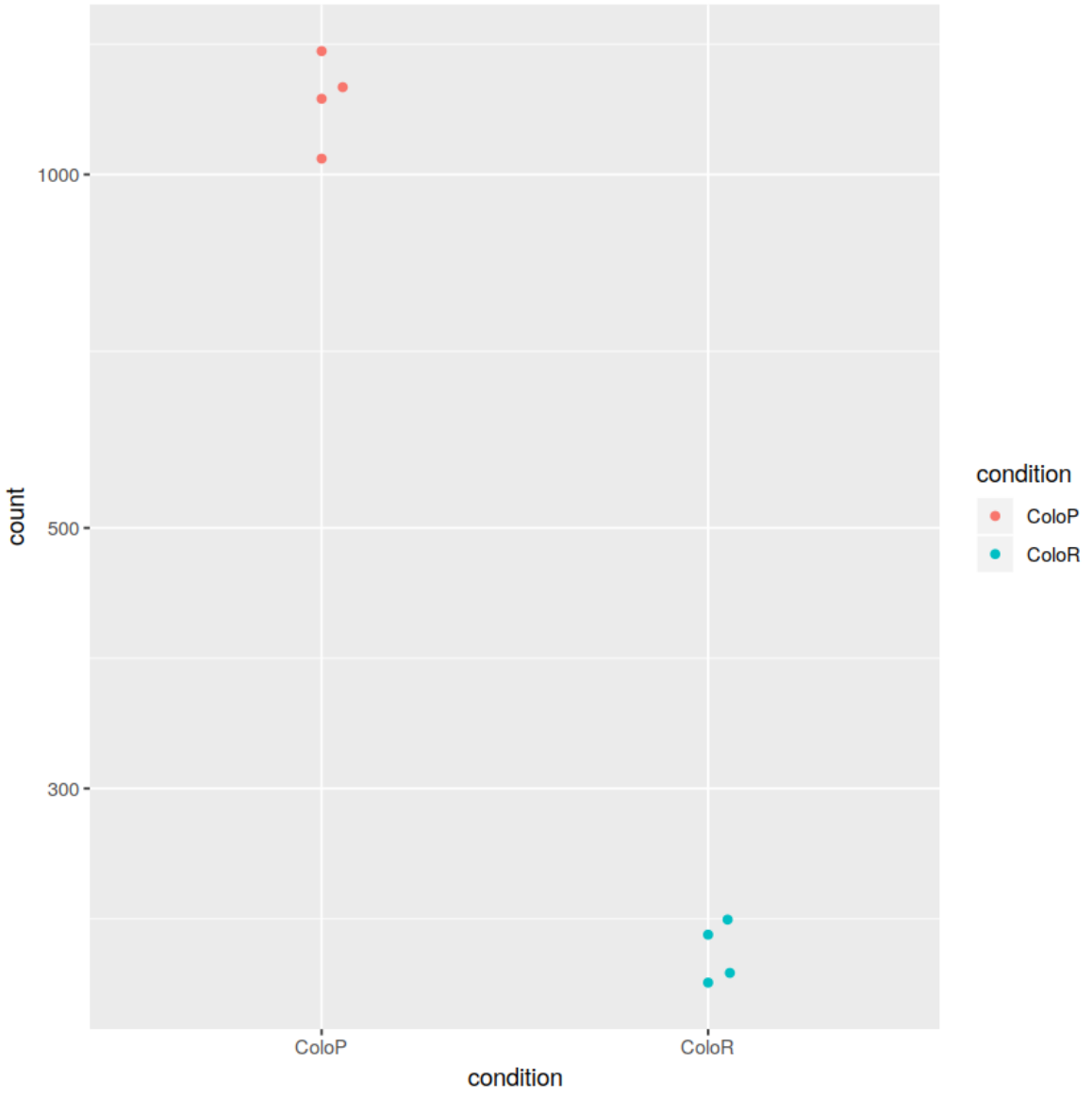
In [69]:

```
topGene <- rownames(res)[which.min(res$padj)]
plotCounts(DESeq.ds, gene = topGene, intgroup=c("condition"))
```



In [73]:

```
library("ggbeeswarm")
geneCounts <- plotCounts(DESeq.ds, gene = topGene, intgroup = c("condition
                         returnData = TRUE)
ggplot(geneCounts, aes(x = condition, y = count, color = condition)) +
  scale_y_log10() +  geom_beeswarm(cex = 3)
```



Read counts of single genes

An important sanity check of your data and the DGE analysis is to see whether genes about which you have prior knowledge behave as expected. In a knockdown experiment you want to see your gene among the most strongly downregulated genes.

In [90]:

```
# get annotation package
library(org.Hs.eg.db)

# list the types of keywords that are available to query the annotation da
keytypes(org.Hs.eg.db)
```

```
Loading required package: AnnotationDbi

Attaching package: ‘AnnotationDbi’

The following object is masked from ‘package:dplyr’:

    select
```

'ACCNUM' 'ALIAS' 'ENSEMBL' 'ENSEMBLPROT' 'ENSEMBLTRANS' 'ENTREZID' 'ENZYME' 'EVIDENCE' 'EVIDENCEALL' 'GENENAME' 'GO' 'GOALL' 'IPI' 'MAP' 'OMIM' 'ONTOLOGY' 'ONTOLOGYALL' 'PATH' 'PFAM' 'PMID' 'PROSITE' 'REFSEQ' 'SYMBOL' 'UCSCKG' 'UNIGENE' 'UNIPROT'

In [91]:

```
# list columns than can be retrieved from the annotation data base
columns(org.Hs.eg.db)
```

'ACCNUM' 'ALIAS' 'ENSEMBL' 'ENSEMBLPROT' 'ENSEMBLTRANS' 'ENTREZID' 'ENZYME' 'EVIDENCE' 'EVIDENCEALL' 'GENENAME' 'GO' 'GOALL' 'IPI' 'MAP' 'OMIM' 'ONTOLOGY' 'ONTOLOGYALL' 'PATH' 'PFAM' 'PMID' 'PROSITE' 'REFSEQ' 'SYMBOL' 'UCSCKG' 'UNIGENE' 'UNIPROT'

In [93]:

```
# make a batch retrieval for all DE genes
anno <- select(org.Hs.eg.db,
               keys = DGEgenes, keytype = "SYMBOL", # to retrieve all gene
               columns = c("ENTREZID", "GENENAME", "ALIAS"))

# check what genes pop up among the top downregulated genes
DGE.results.sorted_logFC <- DGE.results[order(DGE.results$log2FoldChange),
head(DGE.results.sorted_logFC)
```

```
'select()' returned 1:many mapping between keys and columns

log2 fold change (MLE): condition ColoR vs ColoP
Wald test p-value: condition ColoR vs ColoP
DataFrame with 6 rows and 6 columns
               baseMean    log2FoldChange             lfcSE                stat
              <numeric>         <numeric>         <numeric>           <numeric>
MZB1    50.1118067050585 -9.16578643157023 1.14206665368006 -8.02561426869045
PKIA    14.1084086698905 -7.33958124893185 1.47834332523188 -4.96473391779992
SNORA31 12.0006432093164 -7.10154500006678 1.50852736304281 -4.70760105122817
HABP2   11.6108240969898 -7.05523769892897 1.43247954222882 -4.92519264041398
WWC2    17.9461224042872 -6.95084252788737 1.33019949548687 -5.22541359508131
UGT2B4  10.5363535392266 -6.91447423196668  1.5264540079419 -4.52976257128727
                      pvalue                 padj
                   <numeric>            <numeric>
MZB1    1.01019174081494e-15  2.4621877470516e-13
PKIA    6.87953234260274e-07 2.73874182559015e-05
SNORA31 2.50649037299575e-06 8.38515813016479e-05
HABP2   8.42772851566214e-07 3.24112629572454e-05
WWC2    1.73766251769905e-07 8.13839350936463e-06
UGT2B4  5.90500073030719e-06 0.000171173358548686
```

In [94]:

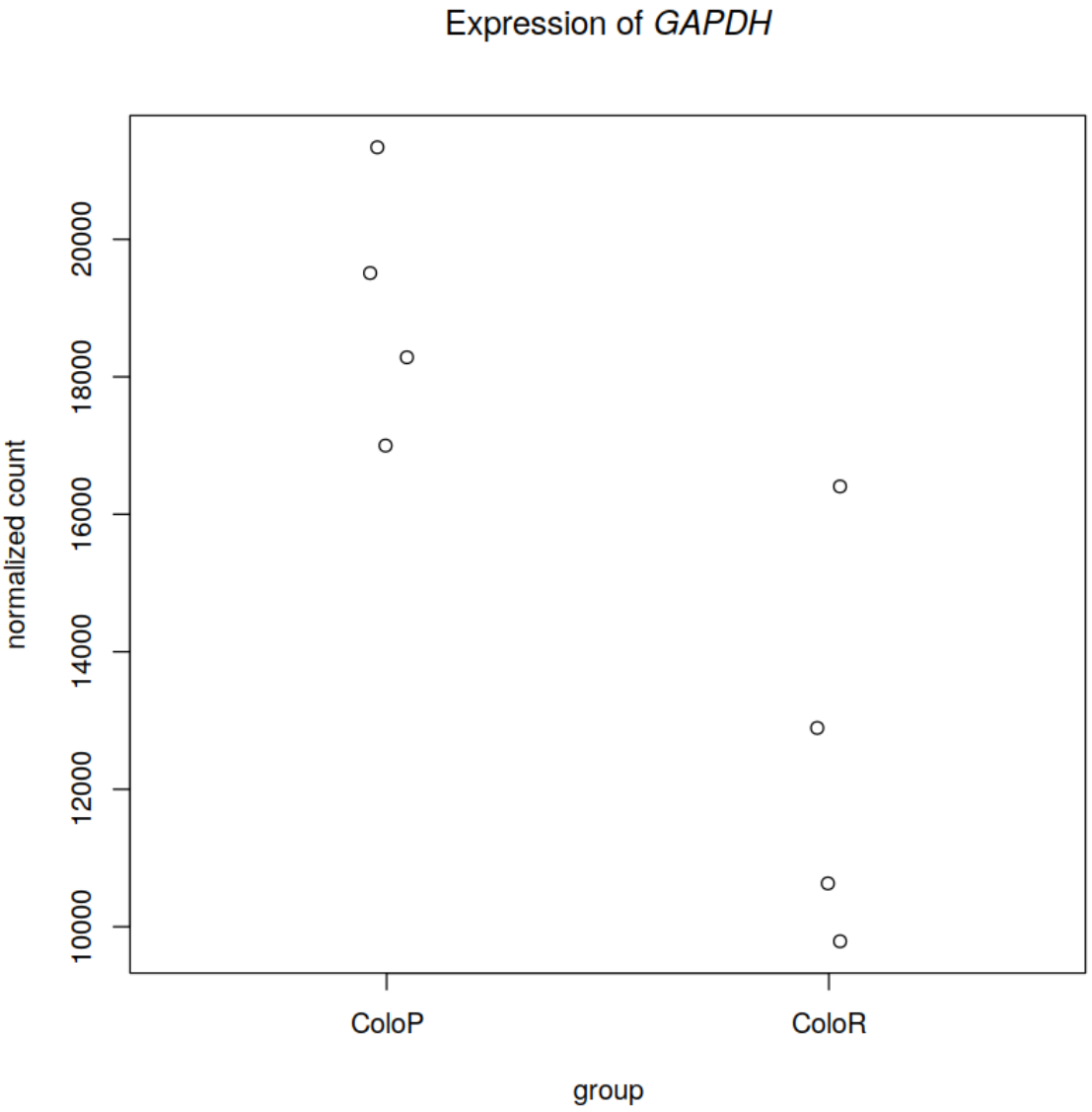
```
DGEgenes_logFC <- rownames(subset(DGE.results.sorted_logFC,padj < 0.05))
head(anno[match(DGEgenes_logFC,anno$SYMBOL),])

# find GAPDH
subset(anno, SYMBOL == "GAPDH")

# DESeq2 offers a wrapper function to plot read counts for single genes
library ( grDevices ) # for italicizing the gene name
plotCounts(dds = DESeq.ds,
           gene = "GAPDH",
           normalized = TRUE, transform = FALSE,
           main = expression(atop("Expression of "*italic("GAPDH"))))
```

A data.frame: 6 × 4

	SYMBOL	ENTREZID	GENENAME	ALIAS
	<chr>	<chr>	<chr>	<chr>
226	MZB1	51237	marginal zone B and B1 cell specific protein	MEDA-7
1195	PKIA	5569	cAMP-dependent protein kinase inhibitor alpha	PRKACN1
1426	SNORA31	677814	small nucleolar RNA, H/ACA box 31	ACA31
1232	HABP2	3026	hyaluronan binding protein 2	FSAP
1034	WWC2	80014	WW and C2 domain containing 2	BOMB
1619	UGT2B4	7363	UDP glucuronosyltransferase family 2 member B4	HLUG25

A data.frame: 4 × 4

	SYMBOL	ENTREZID	GENENAME	ALIAS
	<chr>	<chr>	<chr>	<chr>
3838	GAPDH	2597	glyceraldehyde-3-phosphate dehydrogenase	G3PD
3839	GAPDH	2597	glyceraldehyde-3-phosphate dehydrogenase	GAPD
3840	GAPDH	2597	glyceraldehyde-3-phosphate dehydrogenase	HEL-S-162eP
3841	GAPDH	2597	glyceraldehyde-3-phosphate dehydrogenase	GAPDH



Using ggplot2 to plot multiple genes

Often it is helpful to check the expression of mutltiple genes of interest at the same time. We are going to plot the normalized count values for the **top 20 differentially expressed genes (by padj values).**

To do this, wh first need to determine the gene names of our top 20 genes by ordering our results and extracting the top 20 genes (by padj values):

In [318]:

```
## Order results by padj values
top20_sigOE_genes <- res_tableOE_tb %>%
        arrange(padj) %>%   #Arrange rows by padj values
        pull(gene) %>%      #Extract character vector of ordered genes
        head(n=20)       #Extract the first 20 genes
```

In [347]:

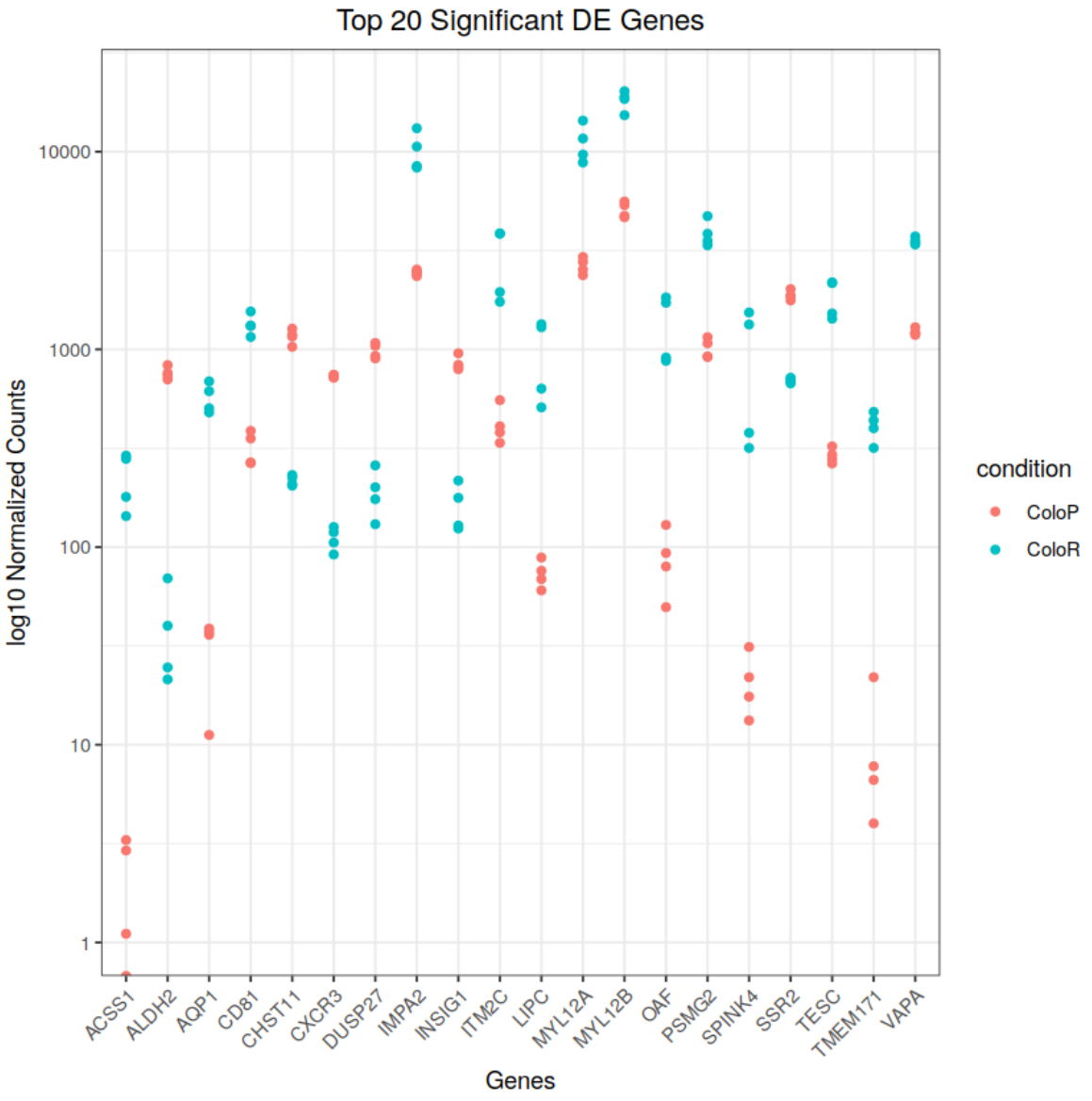
```r
## normalized counts for top 20 significant genes

# Create tibbles including row names
mov10_meta <- sample_info %>%
  rownames_to_column(var="samplename") %>%
  as_tibble()

normalized_counts <- counts(DESeq.ds, normalized=TRUE)

normalized_counts <- normalized_counts %>%
  data.frame() %>%
  rownames_to_column(var="gene") %>%
  as_tibble()

# normalized counts for top 20 significant genes
top20_sigOE_norm <- normalized_counts %>%
        filter(gene %in% top20_sigOE_genes)


# Gathering the columns to have normalized counts to a single column
gathered_top20_sigOE <- top20_sigOE_norm %>%
  gather(colnames(top20_sigOE_norm)[2:9], key = "samplename", value = "nor

gathered_top20_sigOE <- inner_join(mov10_meta, gathered_top20_sigOE)
```

```
Joining, by = "samplename"
```

In [348]:

```
## plot using ggplot2
ggplot(gathered_top20_sigOE) +
        geom_point(aes(x = gene, y = normalized_counts, color = condition)) +
        scale_y_log10() +
        xlab("Genes") +
        ylab("log10 Normalized Counts") +
        ggtitle("Top 20 Significant DE Genes") +
        theme_bw() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    theme(plot.title = element_text(hjust = 0.5))
```

```
Warning message:
"Transformation introduced infinite values in continuous y-axis"
```



Heatmap

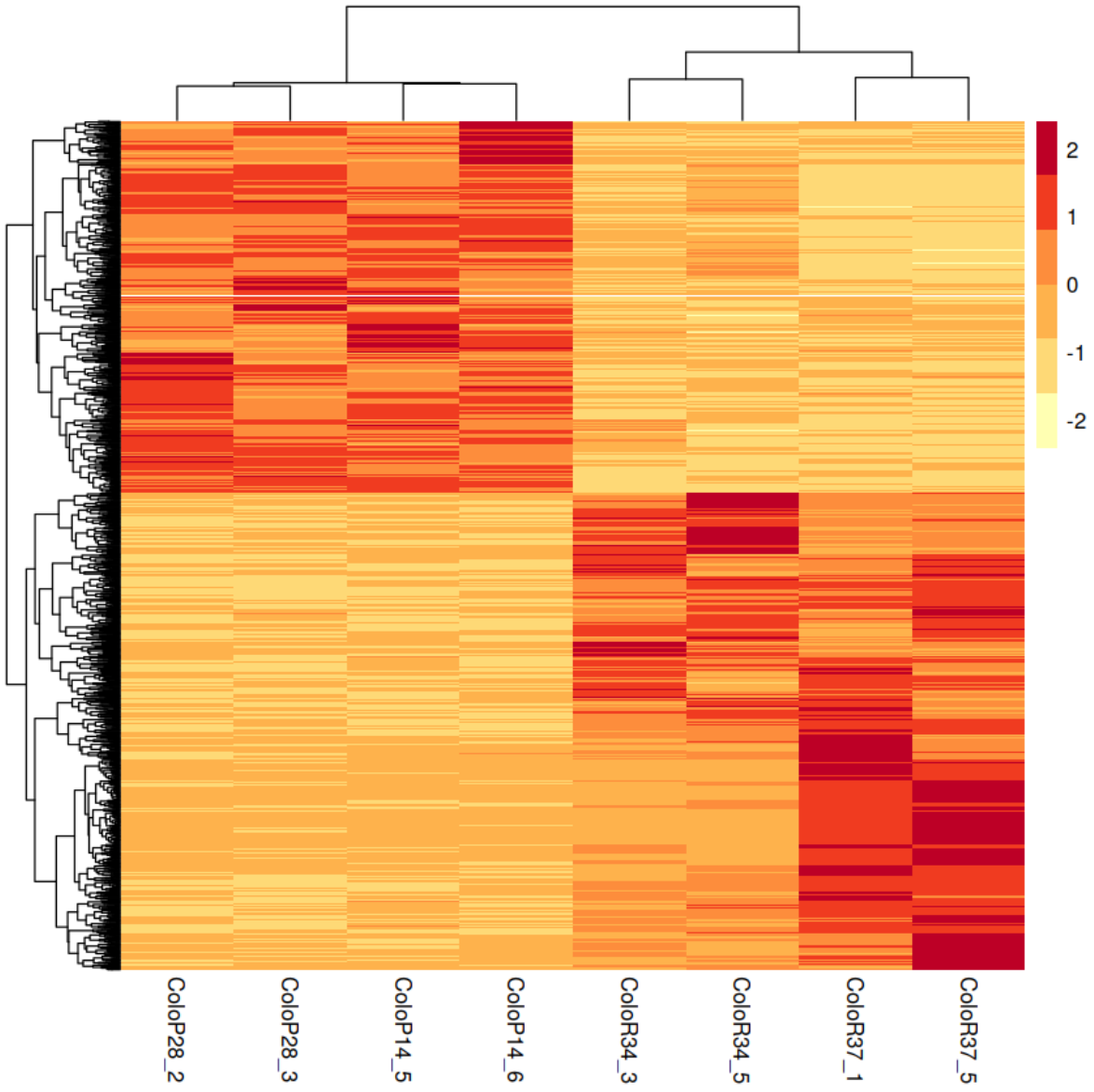
In addition to plotting subsets, we could also extract the normalized values of *all* the significant genes and plot a heatmap of their expression using *pheatmap()*.

In [355]:

```
### Extract normalized expression for significant genes from the OE and co
norm_OEsig <- normalized_counts[,c(1,2:9)] %>%
              filter(gene %in% sigOE$gene) %>%
          data.frame() %>%
          column_to_rownames(var = "gene")
```

In [356]:

```
### Set a color palette
heat_colors <- brewer.pal(6, "YlOrRd")

### Run pheatmap
pheatmap(norm_OEsig,
         color = heat_colors,
         cluster_rows = T,
         show_rownames = F,
         border_color = NA,
         fontsize = 10,
         scale = "row",
         fontsize_row = 10,
         height = 20)
```



Volcano plot

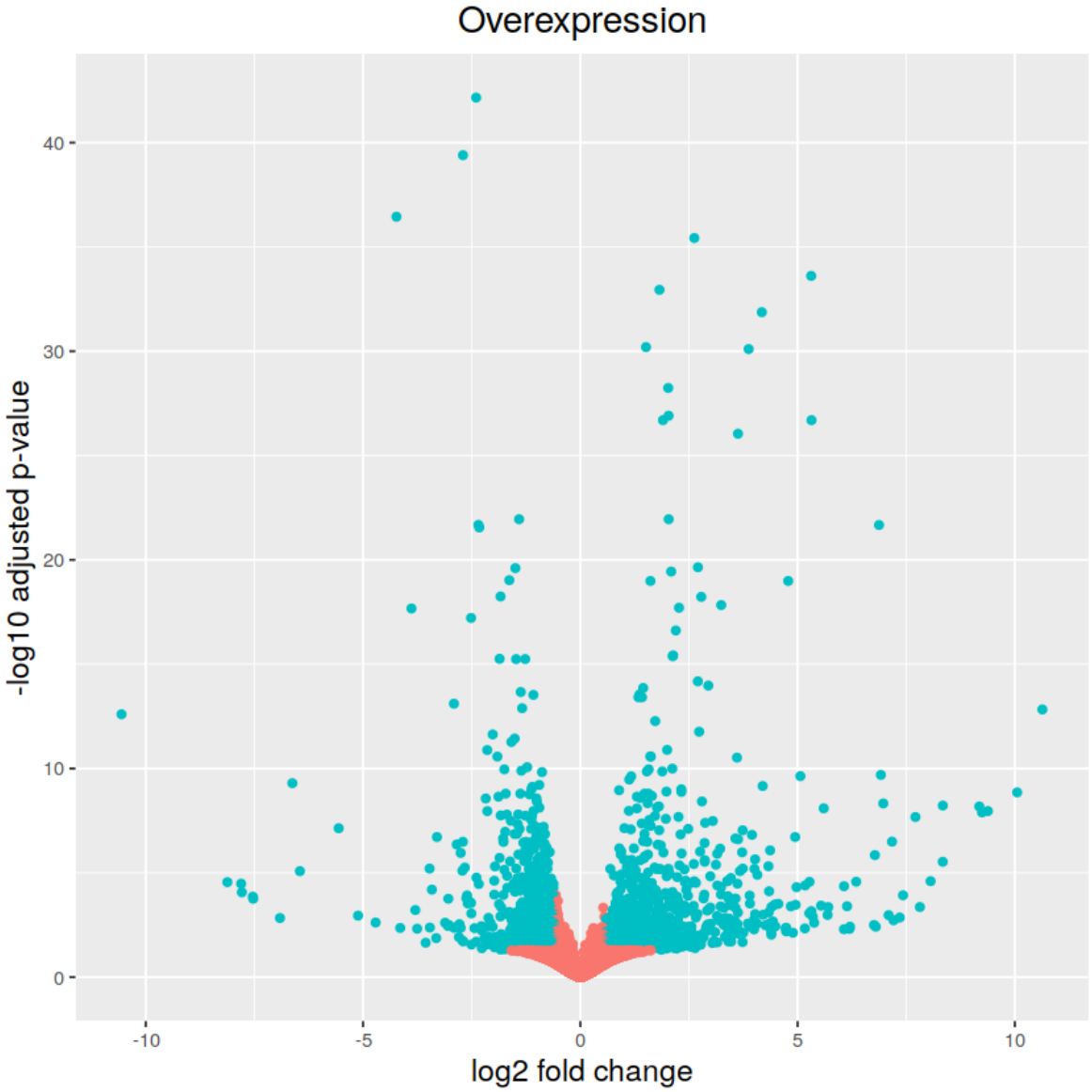
The above plot would be great to look at the expression levels of a good number of genes, but for more of a global view there are other plots we can draw. A commonly used one is a volcano plot; in which you have the log transformed adjusted p-values plotted on the y-axis and log2 fold change values on the x-axis.

To generate a volcano plot, we first need to have a column in our results data indicationg whether or not the gene is considered differentially expressed based on p-adjusted values.

In [373]:
```
## Obtain logical vector where TRUE values denote padj values < 0.05 and f

res_tableOE_tb <- res_tableOE_tb %>%
                  mutate(threshold_OE = padj < 0.05 & abs(log2FoldChange)
```

In [377]:
```
## Volcano plot
ggplot(res_tableOE_tb) +
        geom_point(aes(x = log2FoldChange, y = -log10(padj), colour = thre
        ggtitle("Overexpression") +
        xlab("log2 fold change") +
        ylab("-log10 adjusted p-value") +
        #scale_y_continuous(limits = c(0,50)) +
        theme(legend.position = "none",
              plot.title = element_text(size = rel(1.5), hjust = 0.5),
              axis.title = element_text(size = rel(1.25)))
```

```
Warning message:
"Removed 2756 rows containing missing values (geom_point)."
```



This is a great way to get an overall picture of what is going on, but what if we also wanted to know where the top 10 genes (lowes padj) in our DE list are located on this plot? We could label those dots with the gene name on the Volcano plot using *geom_text_repel()*.

First, we need to order the res_tableOE tibble by padj, and add an additional column ot it, to include on those gene names we want to use to label the plot.

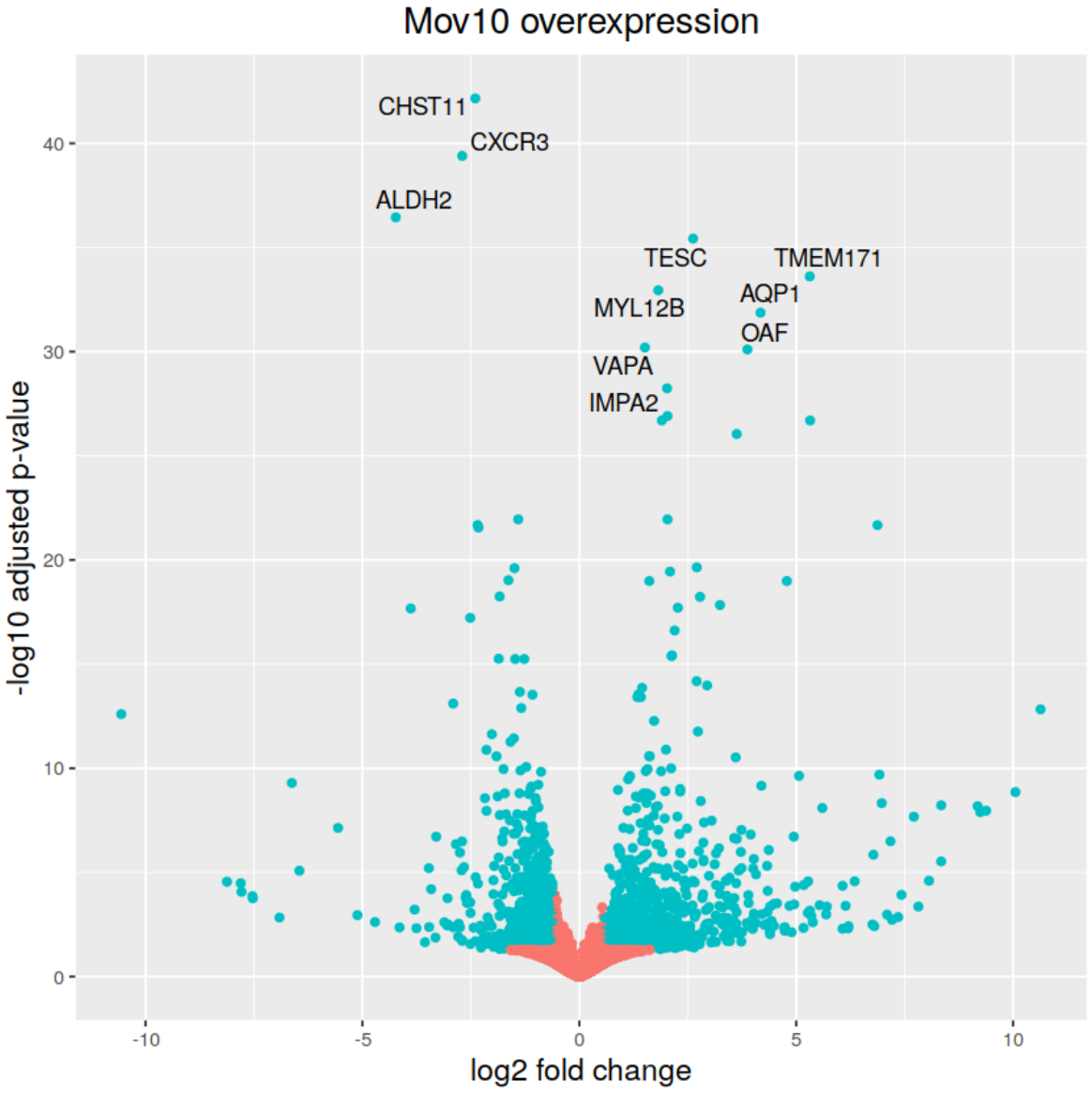
In [378]:

```
## Create a column to indicate which genes to label
res_tableOE_tb <- res_tableOE_tb %>% arrange(padj) %>% mutate(genelabels =

res_tableOE_tb$genelabels[1:10] <- res_tableOE_tb$gene[1:10]
```

In [380]:

```
library(ggrepel)
ggplot(res_tableOE_tb, aes(x = log2FoldChange, y = -log10(padj))) +
        geom_point(aes(colour = threshold_OE)) +
        geom_text_repel(aes(label = genelabels)) +
        ggtitle("Mov10 overexpression") +
        xlab("log2 fold change") +
        ylab("-log10 adjusted p-value") +
        theme(legend.position = "none",
              plot.title = element_text(size = rel(1.5), hjust = 0.5),
              axis.title = element_text(size = rel(1.25)))
```

```
Warning message:
"Removed 2756 rows containing missing values (geom_point)."Warning message:
"Removed 2756 rows containing missing values (geom_text_repel)."
```

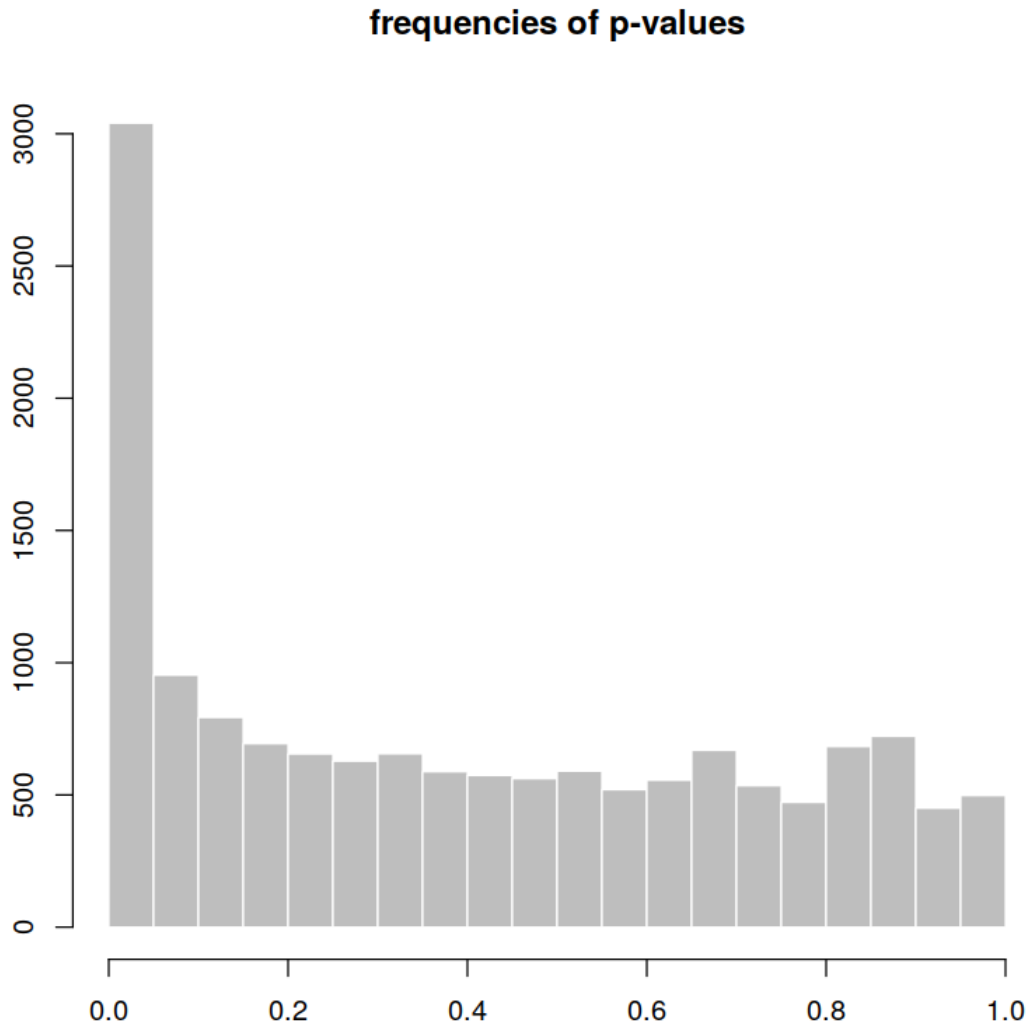


Histograms

Histograms are a simple and fast way of getting a feeling for how frequently certain values are present in a data set. A common example is a histogram of p-values.

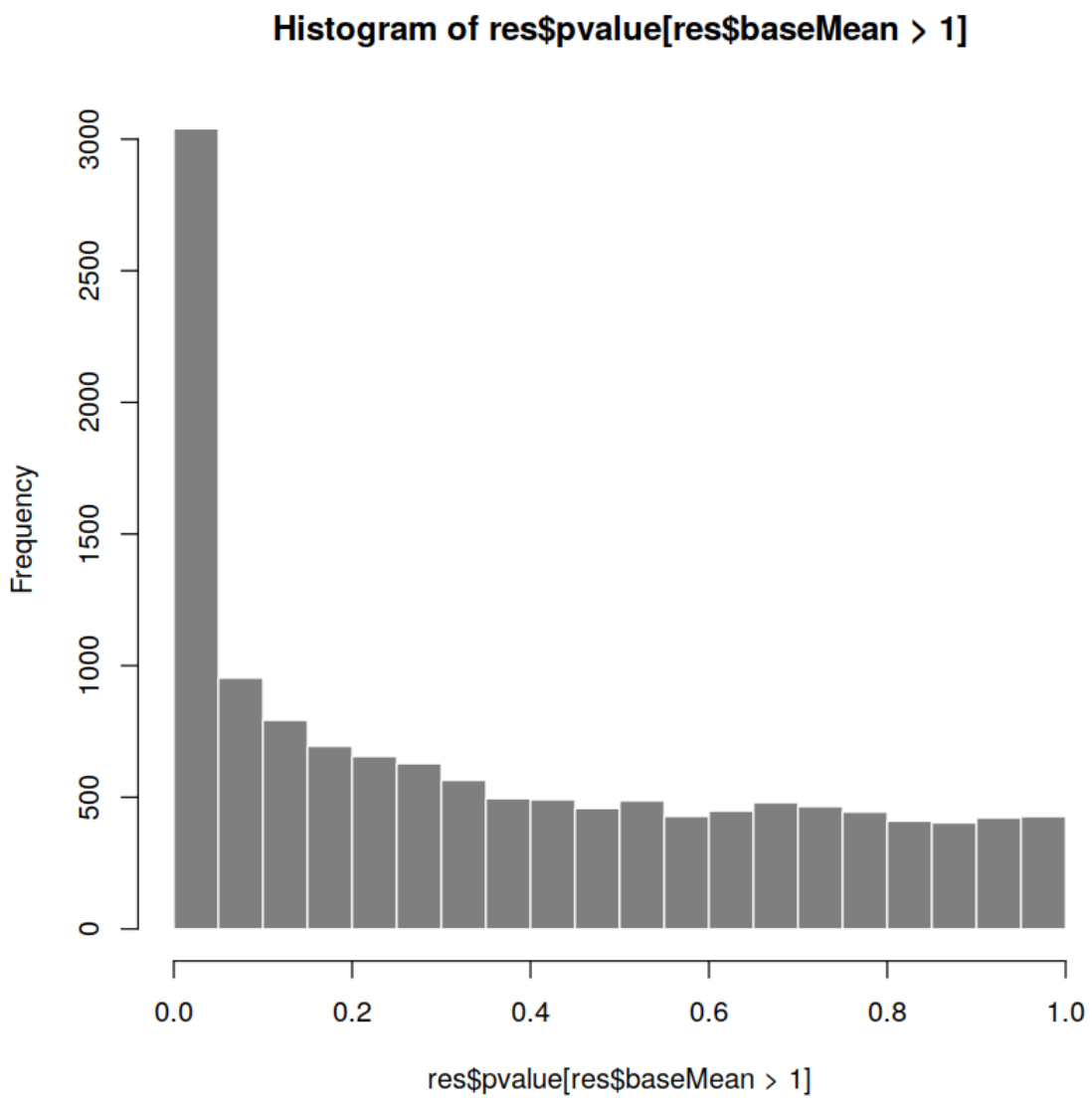
In [74]:

```
hist(DGE.results$pvalue,
     col = "grey", border = "white", xlab = "", ylab = "",
     main = "frequencies of p-values")
```



In [75]:

```
# best formed by excluding genes with very small counts, which might other
# generate spikes in the histogram.
hist(res$pvalue[res$baseMean > 1], breaks = 0:20/20,
     col = "grey50", border = "white")
```



MA plot

An MA-plot (Dudoit et al. 2002) provides a useful overview for the distribution of the estimated coefficients in the model, e.g. the comparisons of interest, across all genes. On the y-axis, the “M” stands for “minus” – subtraction of log values is equivalent to the log of the ratio – and on the x-axis, the “A” stands for “average”. You may hear this plot also referred to as a mean-difference plot, or a Bland-Altman plot.

The MA shows the mean of the normalized counts versus the log2 foldchanges for all genes tested. The genes that are signigicantly DE are colored to be easily identified. In addition to the comparison described above, this plot allows us to evaluate the magnitude of fold changes and how they are distributed relative to mean expression. Generally, we would expect to see significant genes across the full range of expression levels.

Before making the MA-plot, we use the lfcShrink function to shrink the log2 fold changes for the comparison of dex treated vs untreated samples. There are three types of shrinkage estimators in DESeq2, which are covered in the DESeq2 vignette. Here we specify the apeglm method for shrinking

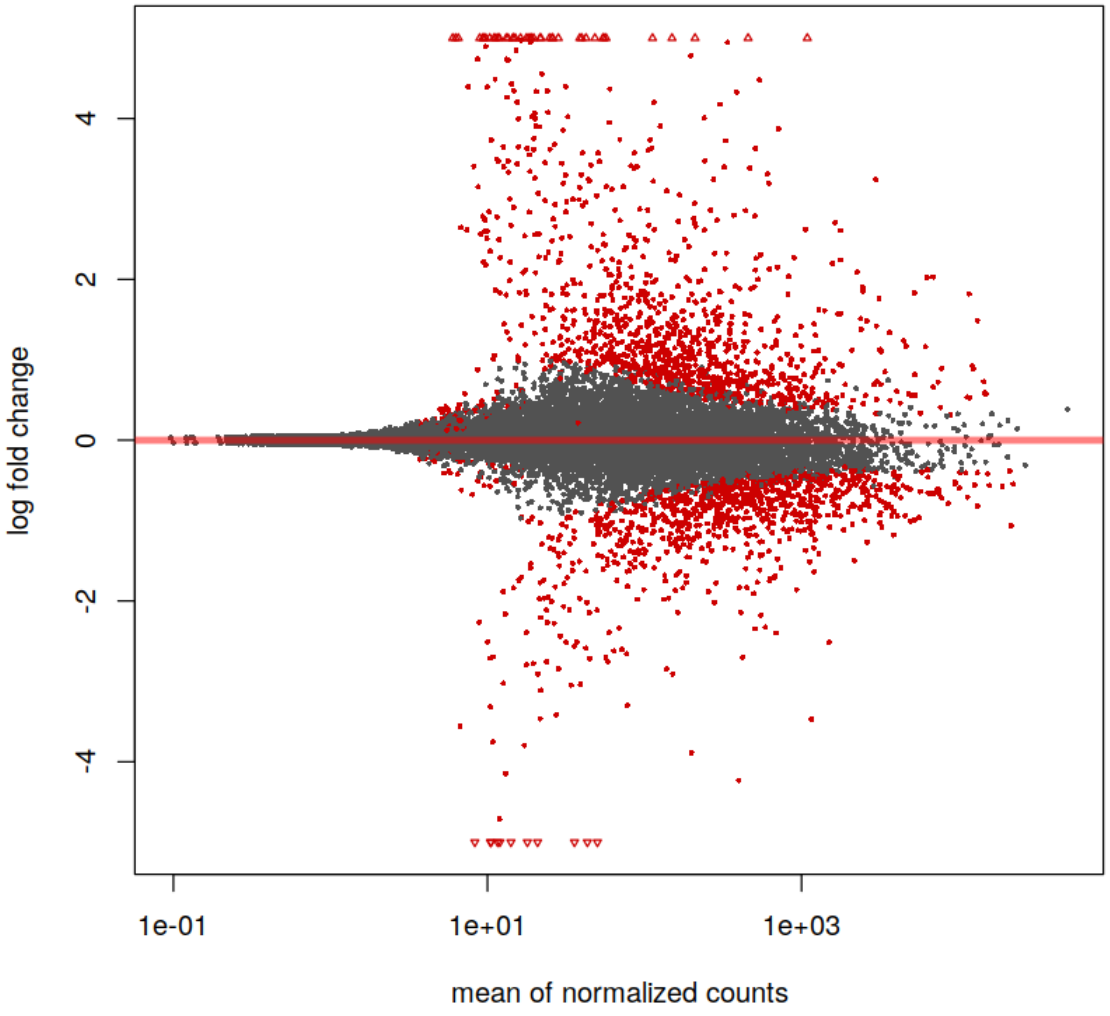
coefficients, which is good for shrinking the noisy LFC estimates while giving low bias LFC estimates for true large differences (Zhu, Ibrahim, and Love 2018). To use apeglm we specify a coefficient from the model to shrink, either by name or number as the coefficient appears in resultsNames(DESeq.ds).

In [76]:
```
library("apeglm")
resultsNames(DESeq.ds)
```

'Intercept' 'condition_ColoR_vs_ColoP'

In [79]:
```
res <- lfcShrink(DESeq.ds, coef="condition_ColoR_vs_ColoP", type="apeglm")
plotMA(res, ylim = c(-5, 5))
```

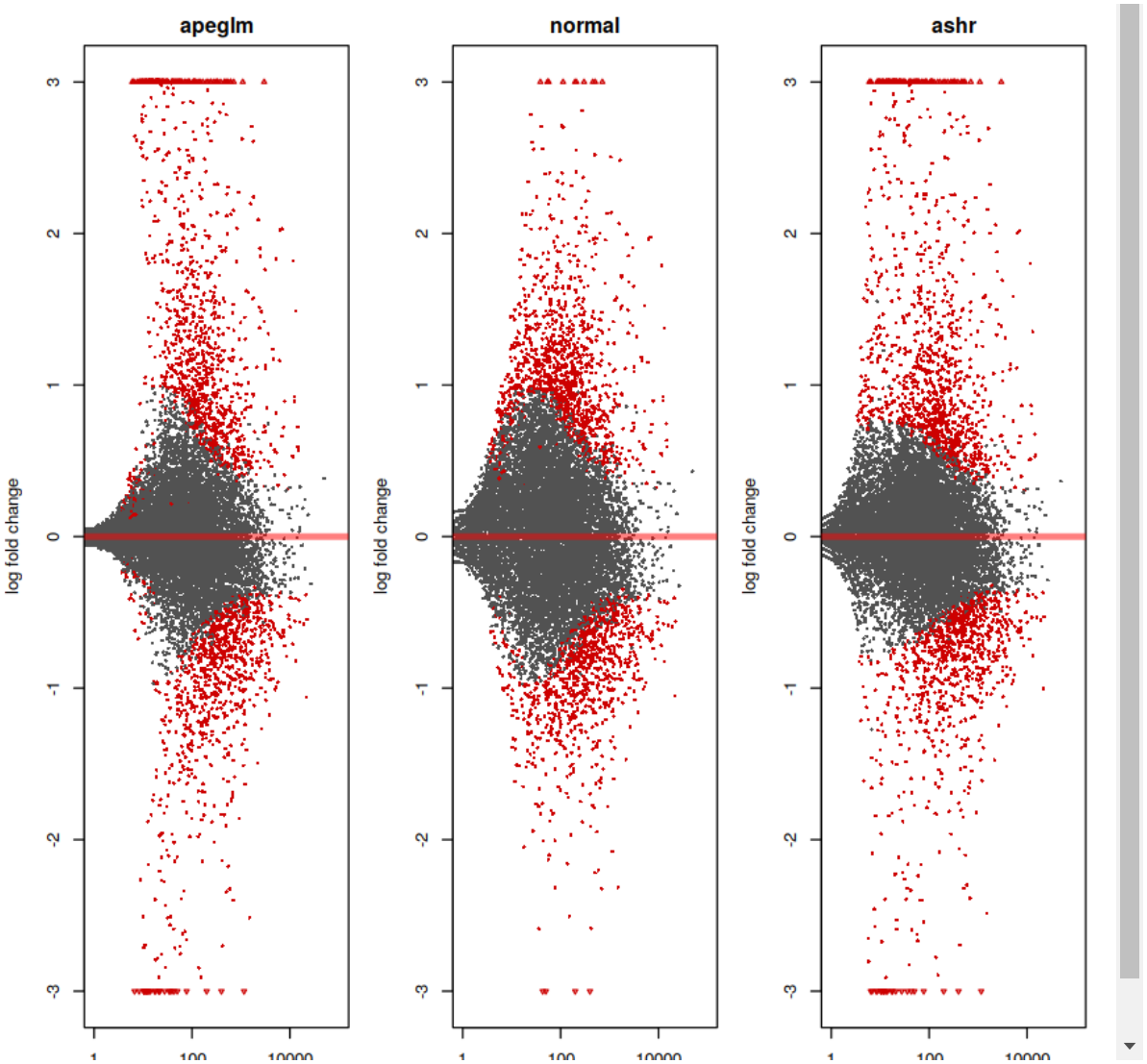
```
using 'apeglm' for LFC shrinkage. If used in published research, please cite:
    Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distributions for
    sequence count data: removing the noise and preserving large differences.
    Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895 (https://doi.org/10.1093/bioinformatics/bty895)
```



In [80]:

```
#MA Plots
# because we are interested in treated vs untreated, we set 'coef=2'
resNorm <- lfcShrink(DESeq.ds, coef="condition_ColoR_vs_ColoP", type="norm
resAsh <- lfcShrink(DESeq.ds, coef="condition_ColoR_vs_ColoP", type="ashr"
resLFC <- lfcShrink(DESeq.ds, coef="condition_ColoR_vs_ColoP", type="apegl


par(mfrow=c(1,3), mar=c(4,4,2,1))
xlim <- c(1,1e5); ylim <- c(-3,3)
plotMA(resLFC, xlim=xlim, ylim=ylim, main="apeglm")
plotMA(resNorm, xlim=xlim, ylim=ylim, main="normal")
plotMA(resAsh, xlim=xlim, ylim=ylim, main="ashr")
```

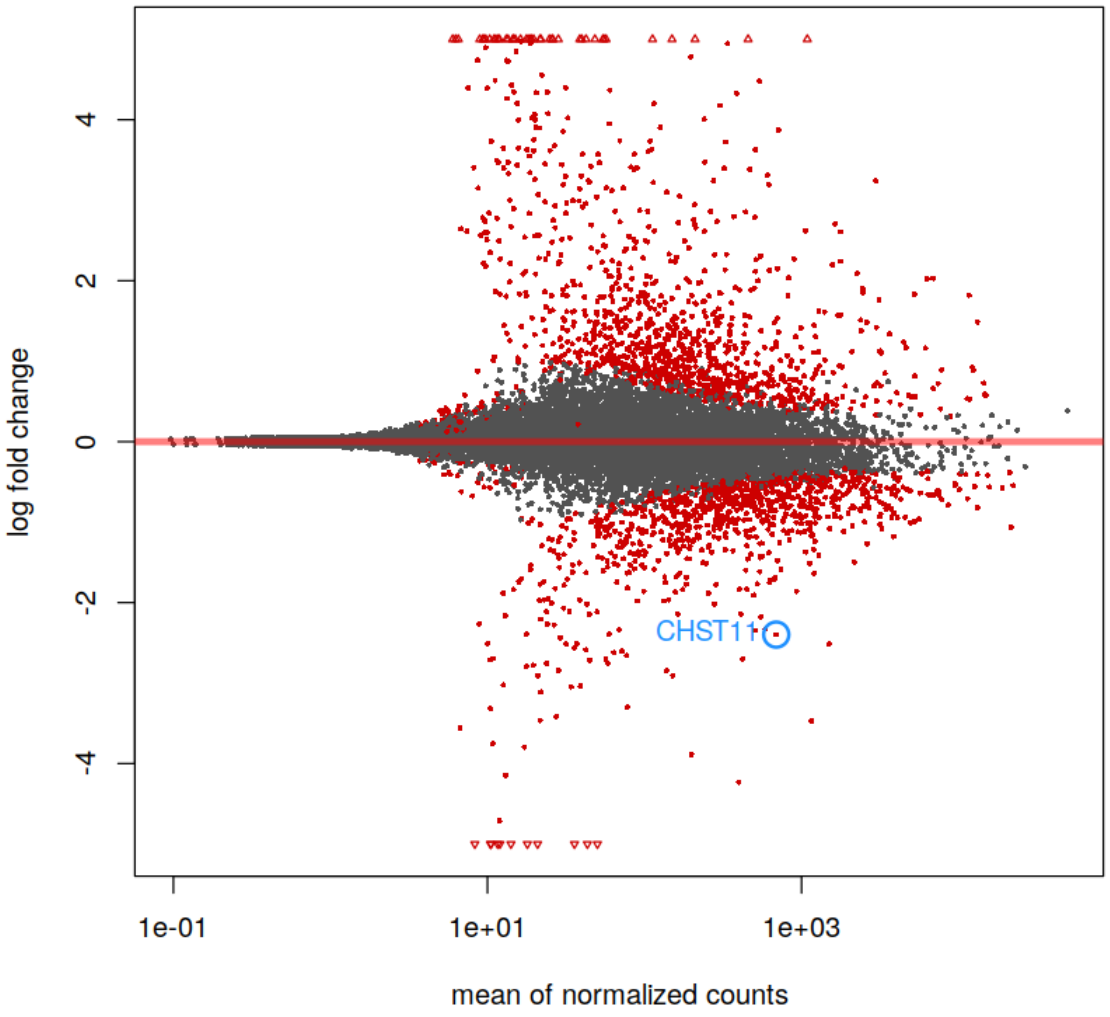
```
using 'normal' for LFC shrinkage, the Normal prior from Love et al (2014).

Note that type='apeglm' and type='ashr' have shown to have less bias than typ
e='normal'.
See ?lfcShrink for more details on shrinkage type, and the DESeq2 vignette.
Reference: https://doi.org/10.1093/bioinformatics/bty895 (https://doi.org/10.
1093/bioinformatics/bty895)
using 'ashr' for LFC shrinkage. If used in published research, please cite:
    Stephens, M. (2016) False discovery rates: a new deal. Biostatistics, 18:
2.
    https://doi.org/10.1093/biostatistics/kxw041 (https://doi.org/10.1093/bio
statistics/kxw041)
using 'apeglm' for LFC shrinkage. If used in published research, please cite:
    Zhu, A., Ibrahim, J.G., Love, M.I. (2018) Heavy-tailed prior distribution
s for
    sequence count data: removing the noise and preserving large differences.
    Bioinformatics. https://doi.org/10.1093/bioinformatics/bty895 (https://do
i.org/10.1093/bioinformatics/bty895)
```



The DESeq2 package uses a Bayesian procedure to moderate (or “shrink”) log2 fold changes from genes with very low counts and highly variable counts, as can be seen by the narrowing of the vertical spread of points on the left side of the MA-plot. As shown above, the lfcShrink function performs this operation. For a detailed explanation of the rationale of moderated fold changes, please see the DESeq2 paper (Love, Huber, and Anders 2014).

In [81]:

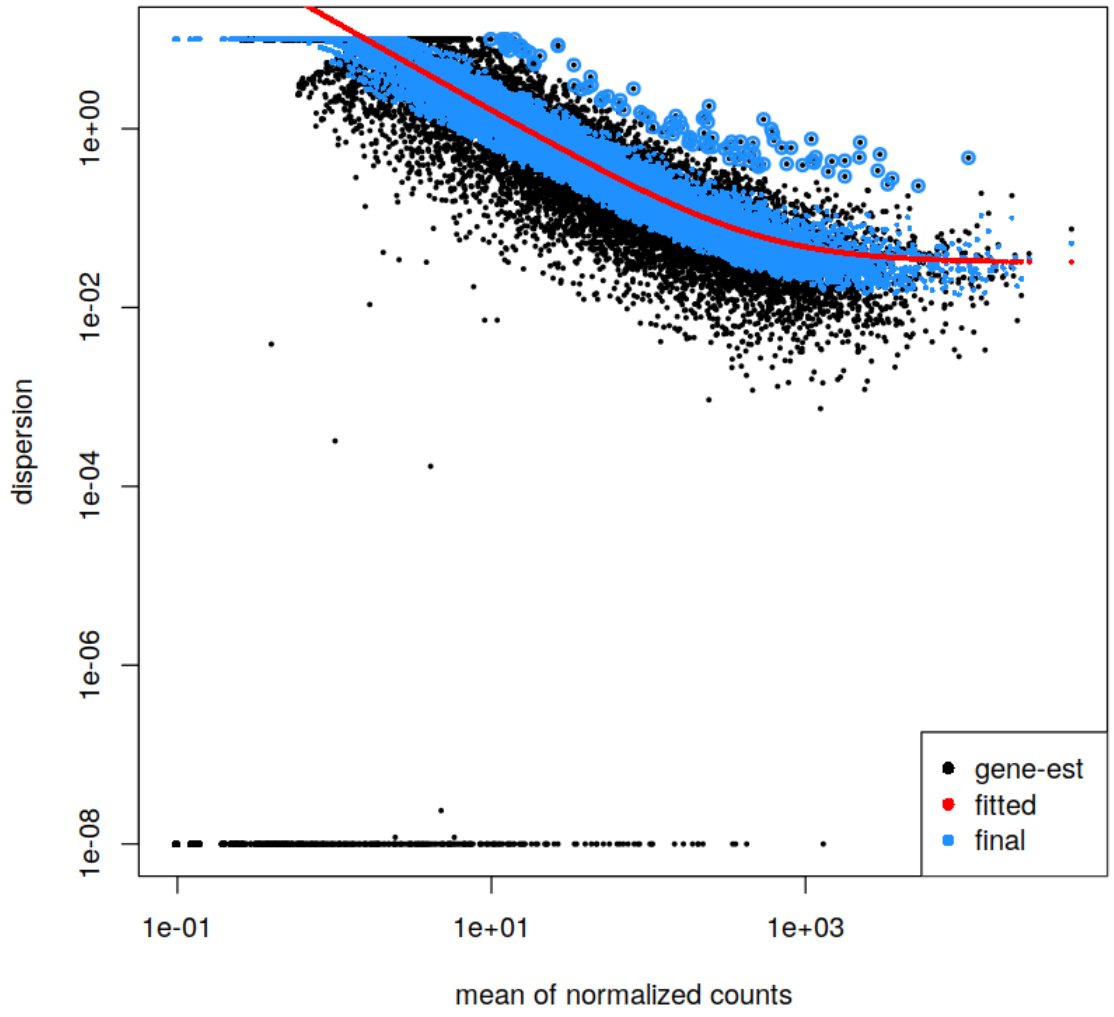
```
# label individual points on the MA-plot
plotMA(res, ylim = c(-5,5))
topGene <- rownames(res)[which.min(res$padj)]
with(res[topGene, ], {
  points(baseMean, log2FoldChange, col="dodgerblue", cex=2, lwd=2)
  text(baseMean, log2FoldChange, topGene, pos=2, col="dodgerblue")
})
```



Dispersion plot

Plotting the dispersion estimates is a useful diagnostic. The dispersion plot below is typical, with the final estimates shrunk from the gene-wise estimates towards the fitted estimates. Some gene-wise estimates are flagged as outliers and not shrunk towards the fitted value, (this outlier detection is described in the manual page for estimateDispersionsMAP). The amount of shrinkage can be more or less than seen here, depending on the sample size, the number of coefficients, the row mean and the variability of the gene-wise estimates.

In [123]:
```
1 plotDispEsts(DESeq.ds)
```

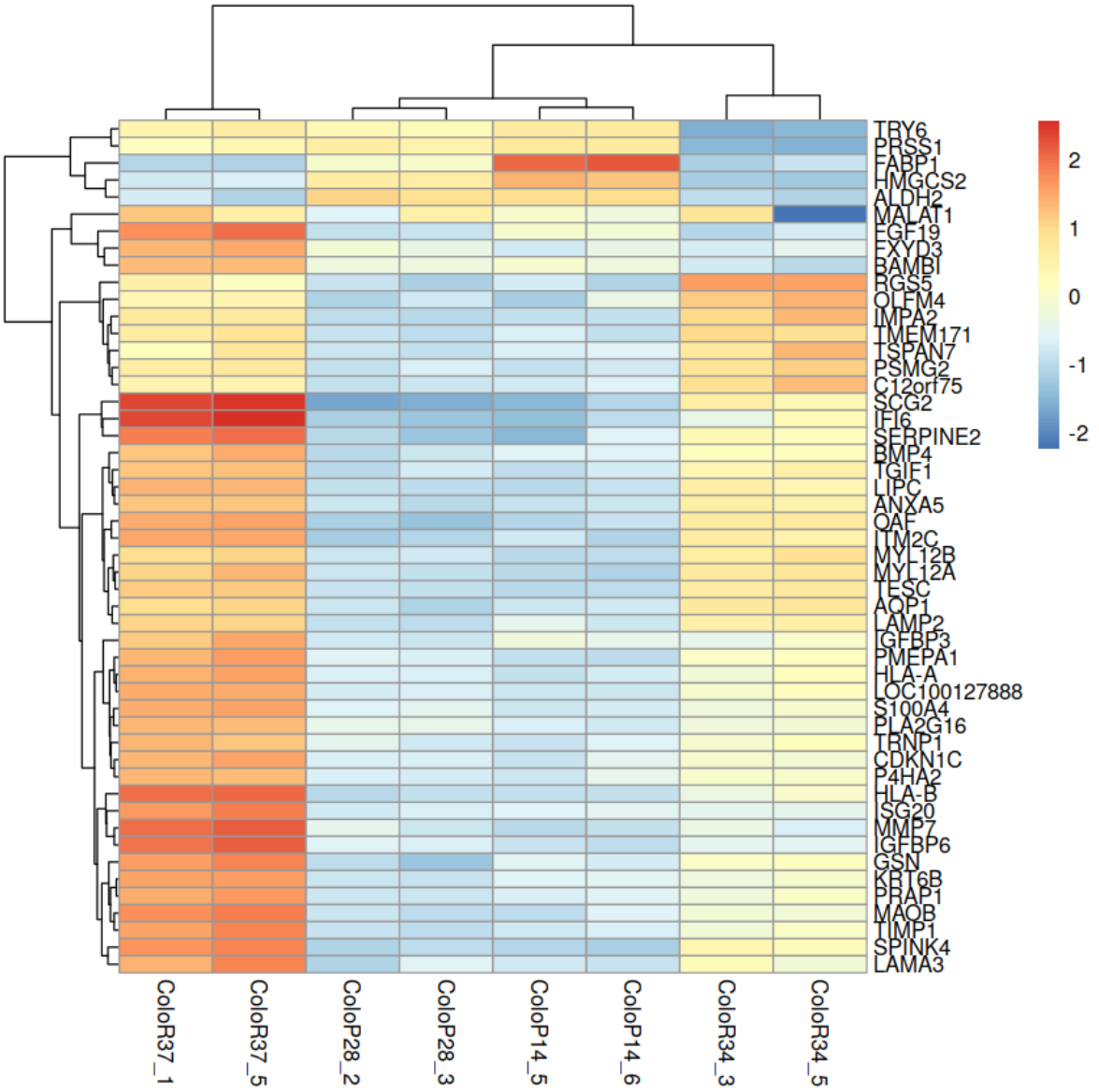


Gene clustering

In the sample distance heatmap made previously, the dendrogram at the side shows us a hierarchical clustering of the samples. Such a clustering can also be performed for the genes. Since the clustering is only relevant for genes that actually carry a signal, one usually would only cluster a subset of the most highly variable genes. Here, for demonstration, let us select the 20 genes with the highest variance across samples. We will work with the VST data.

In [84]:

```
library("genefilter")
topVarGenes <- head(order(rowVars(assay(vsd)), decreasing = TRUE), 50)
mat  <- assay(vsd)[ topVarGenes, ]
mat  <- mat - rowMeans(mat)
anno <- as.data.frame(colData(vsd)[, c("condition")])
pheatmap(mat)
```

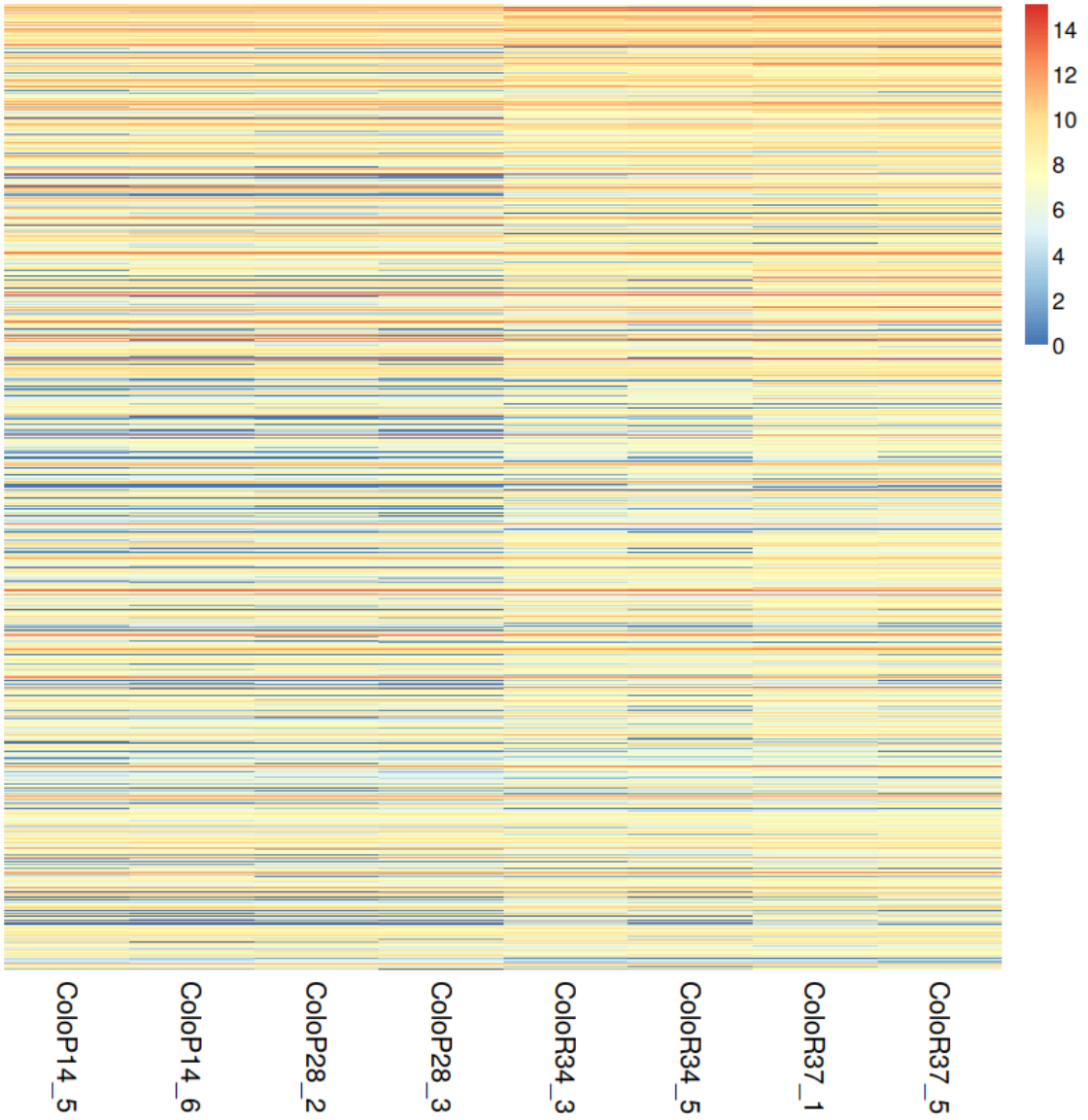


Heatmaps

Heatmaps are a popular means to visualize the expression values across the individual samples. The following commands can be used to obtain heatmaps for *rlog*-normalized read counts for genes that show differential expression with adjusted p-values < 0.05.

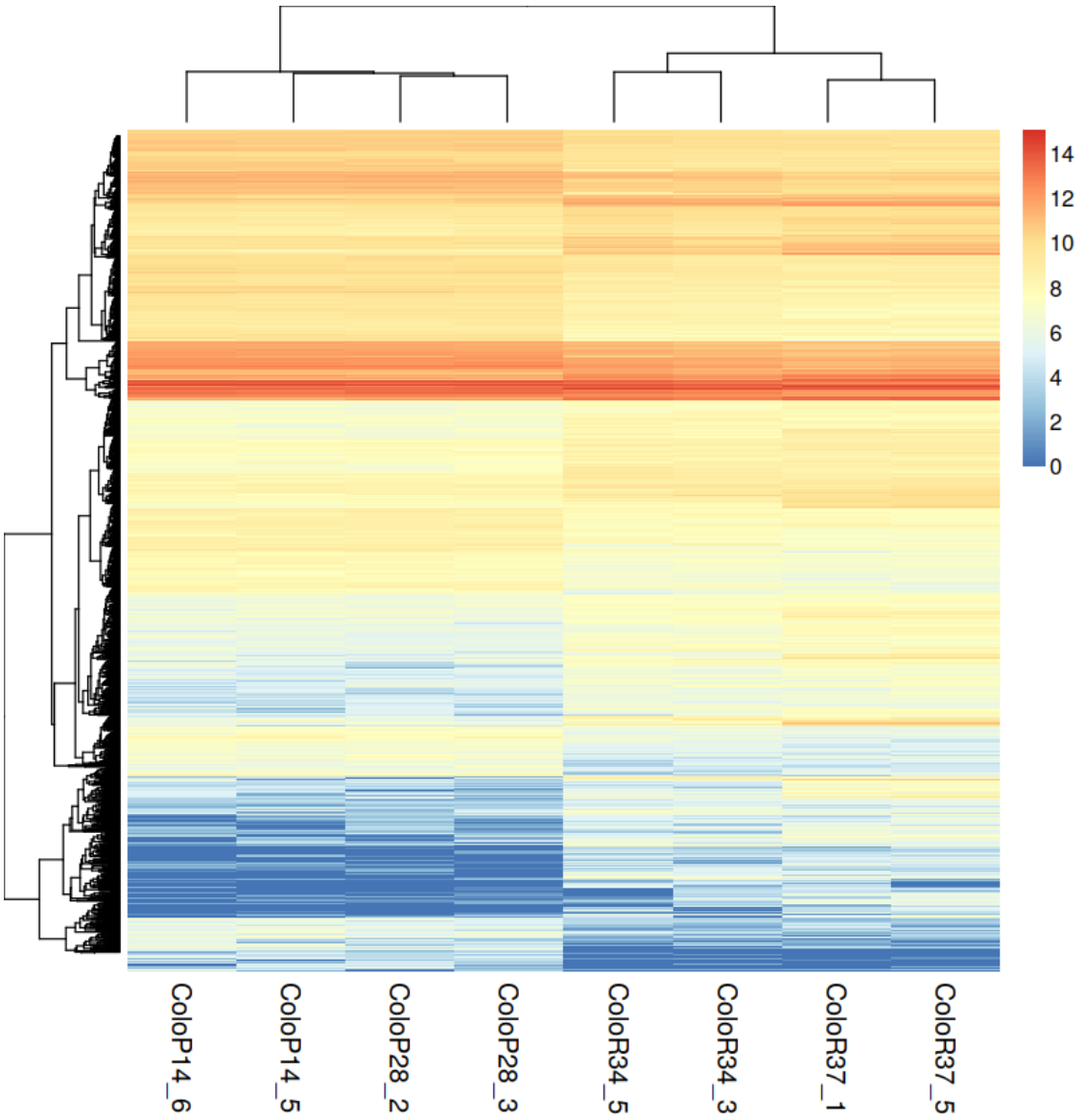
In [86]:

```
library ( NMF )


# aheatmap needs a matrix of values, e.g. a matrix of DE genes with the
# transformed read counts for each replicate
# sort the results according to the adjusted p-value
DGE.results.sorted <- DGE.results[order(DGE.results$padj), ]

# identify genes with the desired adjusted p-value cut-off
DGEgenes <- rownames(subset(DGE.results.sorted, padj < 0.05))

# extract the normalized read counts for DE into a matrix
hm.mat_DGEgenes <- log.norm.counts[DGEgenes, ]

# plot the normalized read counts of DE genes sorted by the adjusted p-val
aheatmap(hm.mat_DGEgenes, Rowv = NA, Colv = NA)
```



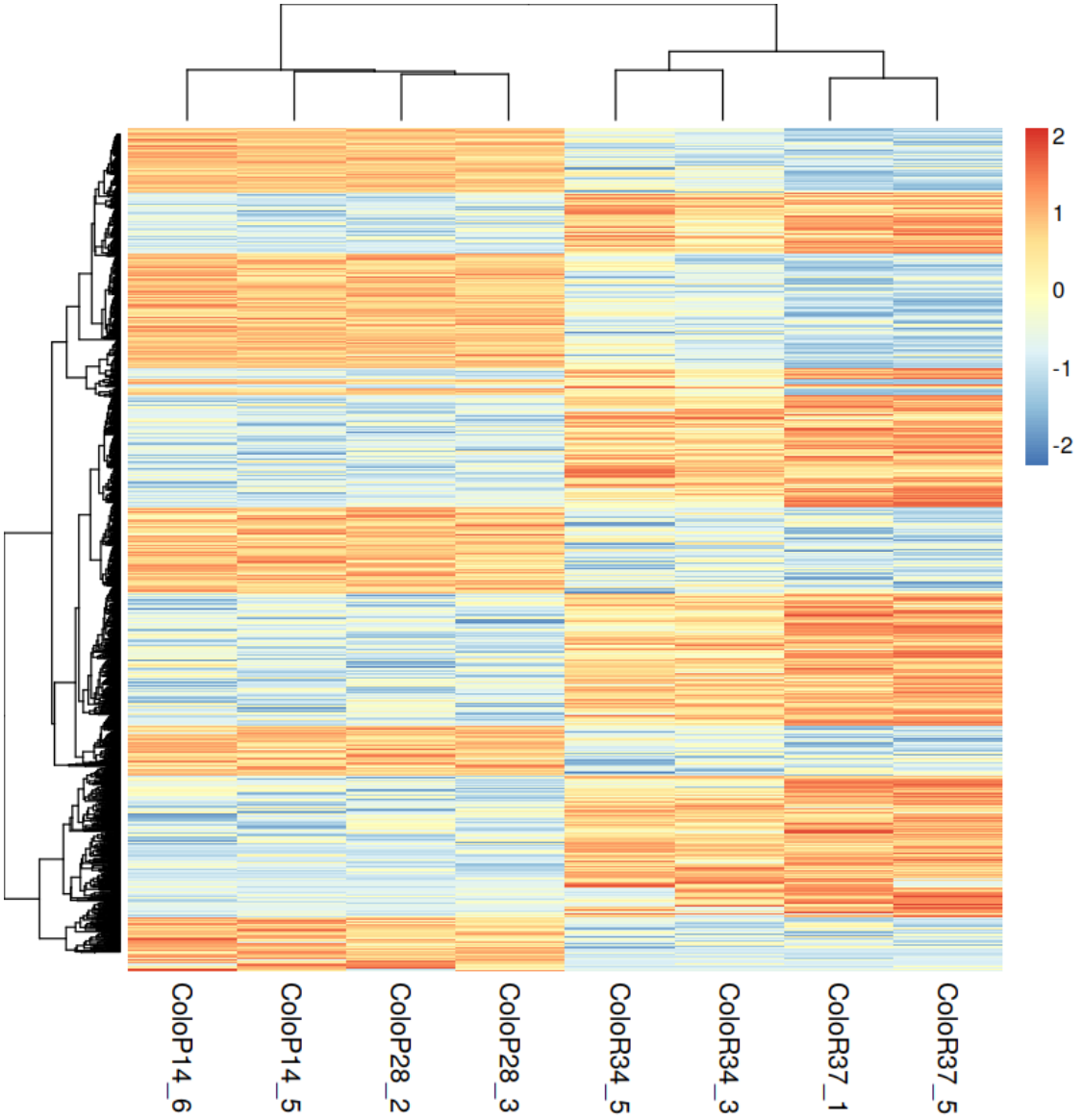
In [87]:

```
# combine the heatmap with hierachical clustering
aheatmap(hm.mat_DGEgenes,
         Rowv = TRUE, Colv = TRUE, # add dendrograms to rows and columns
         distfun = "euclidean", hclustfun = "average")
```



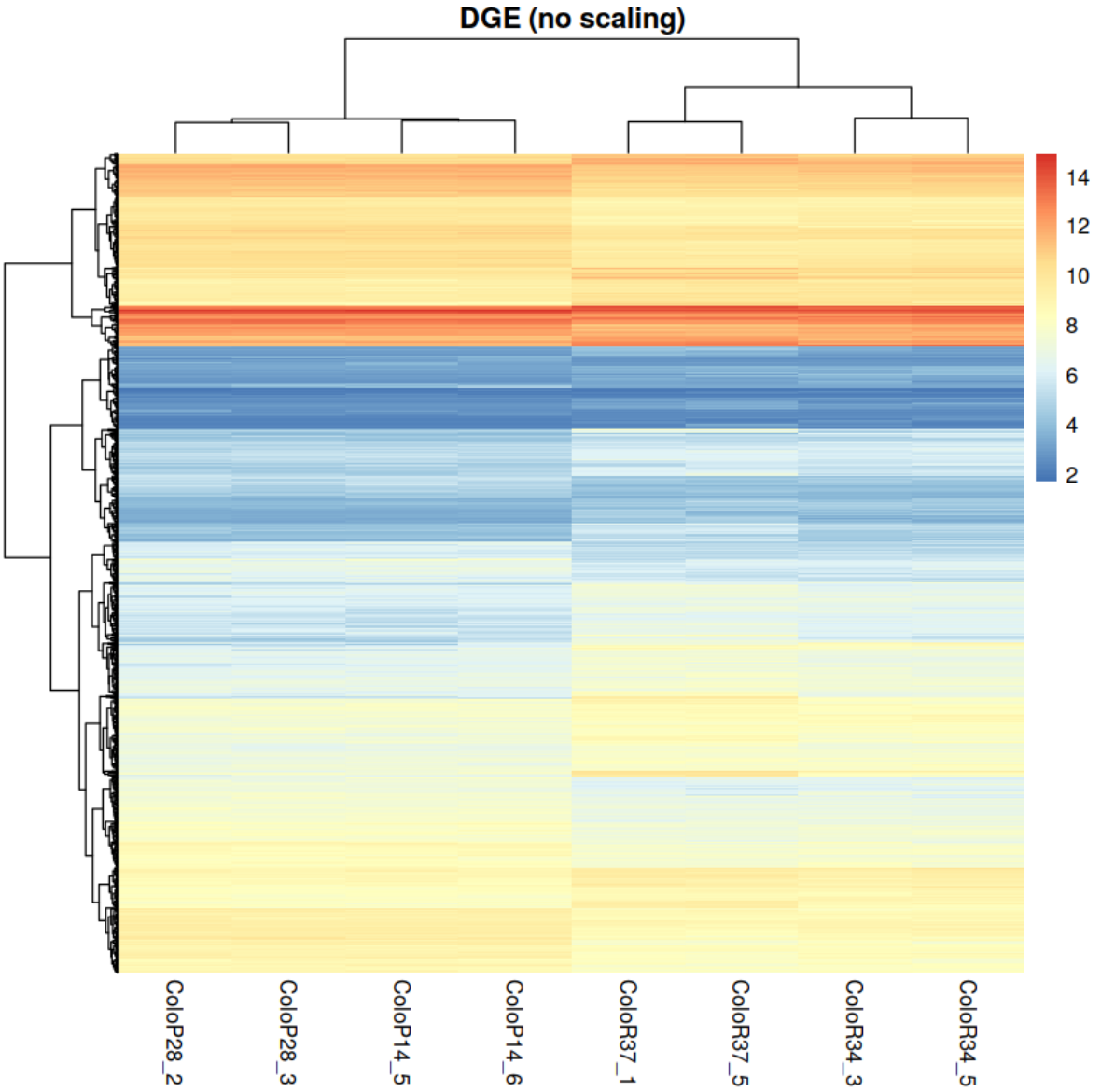
In [88]:

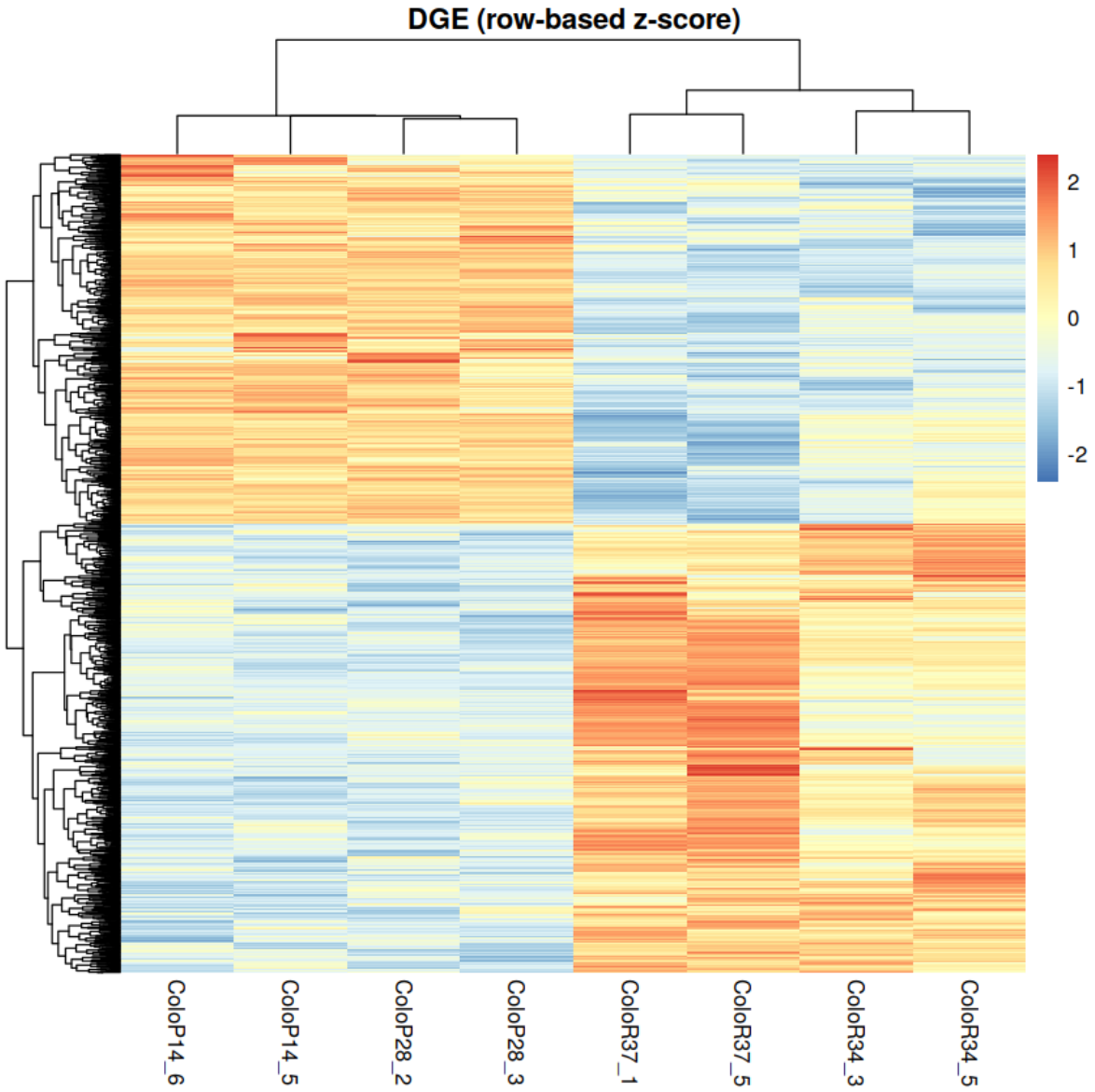
```
# scale the read counts per gene to emphasize the sample-type-specific dif
aheatmap(hm.mat_DGEgenes,
         Rowv = TRUE, Colv = TRUE,
         distfun = "euclidean", hclustfun = "average",
         scale = "row") # values are transformed into distances from the c
                        # of the row-specific average: (actual value-mean
                        # the group) / standard deviation
```



In [89]:

```
# extract rlog-transformed values of DE genes into a matrix
rlog.dge <- DESeq.rlog[DGEgenes,] %>% assay

library(pheatmap)
# heatmap of DEG sorted by p.adjust
pheatmap(rlog.dge, scale="none", show_rownames = FALSE, main = "DGE (no sc


pheatmap(rlog.dge, scale="row", show_rownames = FALSE, main = "DGE (row-ba
```





Annotating and exporting results

Our result table so far only contains gene Symbol, but alternative gene names and Ensembl gene IDs may be more informative for interpretation. Bioconductor's annotation packages help with mapping various ID schemes to each other.

In [96]:

```
# load the annotation packages
library("AnnotationDbi")
library("org.Hs.eg.db") # this is the Homo sapies (HS) organism (org) anno

columns(org.Hs.eg.db)
```

'ACCNUM' 'ALIAS' 'ENSEMBL' 'ENSEMBLPROT' 'ENSEMBLTRANS' 'ENTREZID' 'ENZYME' 'EVIDENCE' 'EVIDENCEALL' 'GENENAME' 'GO' 'GOALL' 'IPI' 'MAP' 'OMIM' 'ONTOLOGY' 'ONTOLOGYALL' 'PATH' 'PFAM' 'PMID' 'PROSITE' 'REFSEQ' 'SYMBOL' 'UCSCKG' 'UNIGENE' 'UNIPROT'

We can use the mapIds function to add individual columns to our results table. We provide the row names of our results table as a key, and specify that keytype=ENSEMBL. The column argument tells the mapIds function which information we want, and the multiVals argument tells the function what to do if there are multiple possible values for a single input value. Here we ask to just give us back the first one that occurs in the database. To add the gene symbol and Entrez ID, we call mapIds twice.

In [99]:

```
res$emsembl <- mapIds(org.Hs.eg.db,
                      keys=row.names(res),
                      column="ENSEMBL",
                      keytype="SYMBOL",
                      multiVals="first")
res$entrez <- mapIds(org.Hs.eg.db,
                      keys=row.names(res),
                      column="ENTREZID",
                      keytype="SYMBOL",
                      multiVals="first")
```

```
'select()' returned 1:many mapping between keys and columns
'select()' returned 1:many mapping between keys and columns
```

In [100]:

```
resOrdered <- res[order(res$pvalue),]
head(resOrdered)
```

```
log2 fold change (MAP): condition ColoR vs ColoP
Wald test p-value: condition ColoR vs ColoP
DataFrame with 6 rows and 7 columns
               baseMean    log2FoldChange             lfcSE
              <numeric>         <numeric>         <numeric>
CHST11  689.926481445679 -2.39799681429977 0.168819965434453
CXCR3    421.72696206591  -2.7014753966521 0.197214819037292
ALDH2   398.521929721851 -4.23176089654189 0.321400251272042
TESC    1057.17534944271  2.62329355121026 0.202389500299887
TMEM171  209.71611760604  5.31269217295704 0.422438223051737
MYL12B  11666.1236183521  1.82118881484595 0.145552200540915
                     pvalue                 padj         emsembl      entrez
                  <numeric>            <numeric>     <character> <character>
CHST11  5.68197271332737e-47 6.94564344477138e-43 ENSG00000171310       50515
CXCR3   6.54971212394784e-44 4.00318405015692e-40 ENSG00000186810        2833
ALDH2   8.78731576760633e-41 3.58053826477399e-37 ENSG00000111275         217
TESC    1.22300984785528e-39 3.73751809504575e-36 ENSG00000088992       54997
TMEM171 1.00125647001092e-37  2.4478718178827e-34 ENSG00000157111      134285
MYL12B  5.58848163637809e-37 1.13855999205143e-33 ENSG00000118680      103910
```

Exporting results

You can save the results table in a CSV file that you can then share or load with a spreadsheet program. The call to *as.data.frame* is necessary to convert the *DataFrame* object to a *data.frame* object that can be processed by *write.csv*.

In [101]:

```
resOrderedDF <- as.data.frame(resOrdered)[1:100, ]
write.csv(resOrderedDF, file = "results.csv")
```

A more sophisticated way for exporting results the Bioconductor package ReportingTools (Huntley et al. 2013). ReportingTools will automatically generate dynamic HTML documents, including links to external databases using gene identifiers and boxplots summarizing the normalized counts across groups. See the ReportingTools vignettes for full details. The simplest version of creating a dynamic ReportingTools report is performed with the following code:

In [103]:
```
library("ReportingTools")
htmlRep <- HTMLReport(shortName="report", title="My report",
                      reportDirectory="./report")
publish(resOrderedDF, htmlRep)
url <- finish(htmlRep)
browseURL(url)
```

```
Loading required package: knitr

Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2

Registered S3 method overwritten by 'R.oo':
  method        from
  throw.default R.methodsS3

Attaching package: 'ReportingTools'

The following object is masked from 'package:NMF':

    name
```

Removing hidden batch effects

Suppose we did not know that there were different cell lines involved in the experiment, only that there was treatment with Folfiri. The cell line effect on the counts then would represent some hidden and unwanted variation that might be affecting many or all of the genes in the dataset. We can use statistical methods designed for RNA-seq from the sva package (Leek 2014) or the RUVSeq package (Risso et al. 2014) in Bioconductor to detect such groupings of the samples, and then we can add these to the DESeqDataSet design, in order to account for them.

The SVA package uses the term surrogate variables for the estimated variables that we want to account for in our analysis, while the RUV package uses the terms factors of unwanted variation with the acronym “Remove Unwanted Variation” explaining the package title. We first use SVA to find hidden batch effects and then RUV following.

Using SVA with DESeq2

Below we obtain a matrix of normalized counts for which the average count across samples is larger than 1. As we described above, we are trying to recover any hidden batch effects, supposing that we do not know the cell line information. So we use a full model matrix with the dex variable, and a reduced, or null, model matrix with only an intercept term. Finally we specify that we want to estimate 2 surrogate variables. For more information read the manual page for the svaseq function by typing ?svaseq.

In [116]:
```
library("sva")
dat  <- counts(DESeq.ds, normalized = TRUE)
idx  <- rowMeans(dat) > 1
dat  <- dat[idx, ]
mod  <- model.matrix(~ condition, colData(DESeq.ds))
mod0 <- model.matrix(~   1, colData(DESeq.ds))
svseq <- svaseq(dat, mod, mod0, n.sv = 2)
```

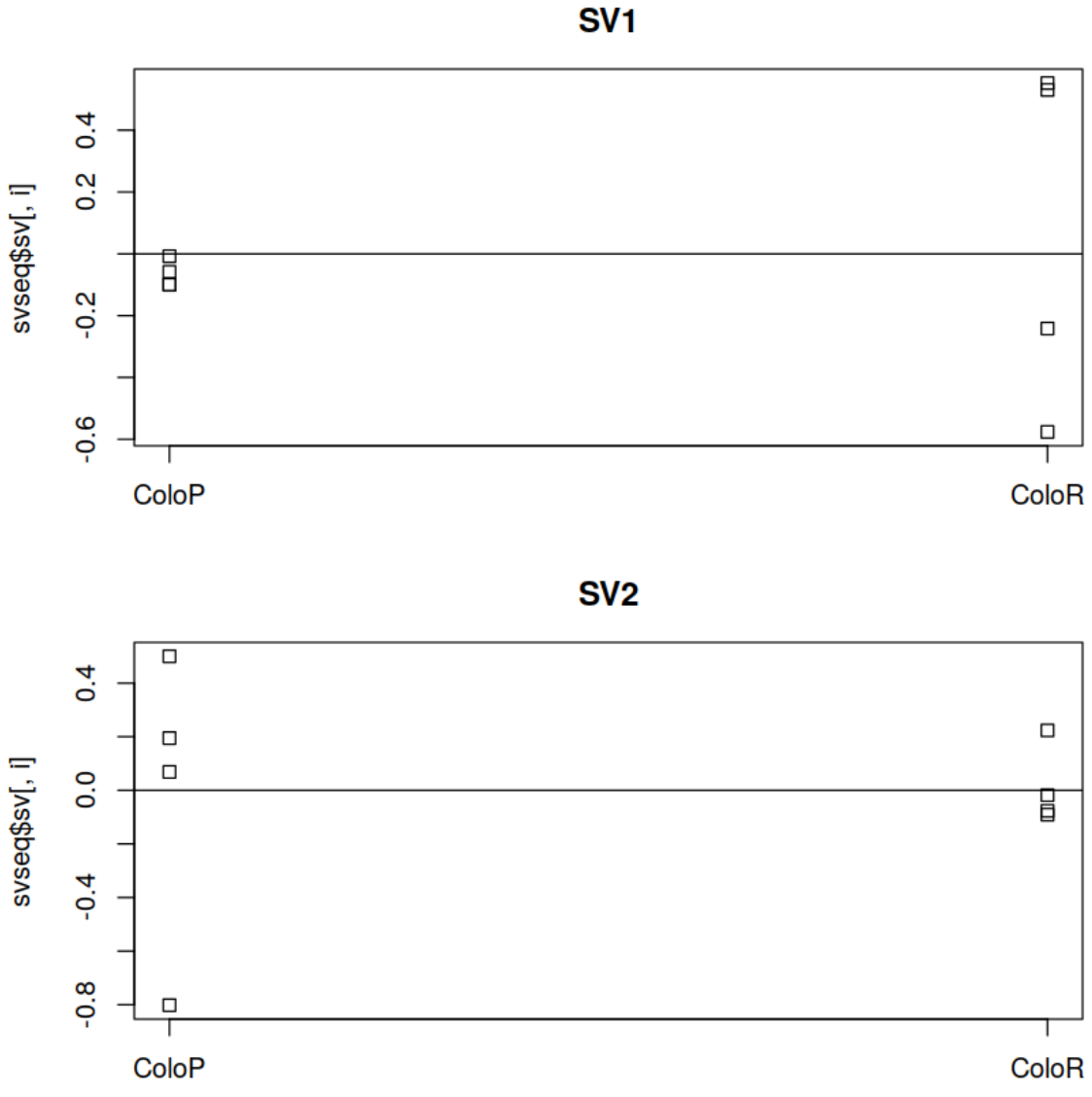
```
Number of significant surrogate variables is:  2
Iteration (out of 5 ):1  2  3  4  5
```

In [117]:
```
svseq$sv
```

A matrix: 8 × 2 of type dbl

-0.007648087	0.49997424
-0.099128309	-0.80180526
-0.100113795	0.19492057
-0.058347060	0.06861288
-0.241562656	0.22361380
-0.576036806	-0.09088248
0.552583982	-0.01797054
0.530252731	-0.07646322

In [118]:
```
par(mfrow = c(2, 1), mar = c(3,5,3,1))
for (i in 1:2) {
  stripchart(svseq$sv[, i] ~ DESeq.ds$condition, vertical = TRUE, main = p
  abline(h = 0)
 }
```



Finally, in order to use SVA to remove any effect on the counts from our surrogate variables, we simply add these two surrogate variables as columns to the DESeqDataSet and then add them to the design:

```
ddssva <- dds
ddssva\$SV1 <- svseq\$sv[,1]
ddssva\$SV2 <- svseq\$sv[,2]
design(ddssva) <- ~ SV1 + SV2 + dex
```

We could then produce results controlling for surrogate variables by running DESeq with the new design.

edgeR

edgeR is very similar to DESeq2, both packages rely on the negative binomial distribution to model the raw read counts in a gene-wise manner while adjusting the dispersion estimates on trends seen across all samples and genes. The methods are, however not identical, and the results may vary.

edgeR requires a matrix of read counts where the row names = gene IDs and the column names = sample IDs. Thus, we can use the same object we used for DESeq2 (readcounts). In addition, we need to specify the sample types, similarly to what we did for DESeq2.

In [124]:
```
library(edgeR)

sample_info.edger <- factor(c(rep("ColoP", 4), rep("ColoR", 4)))
sample_info.edger <- relevel(sample_info.edger, ref = "ColoP")


# DGEList() is the function that converts the count matrix into an edgeR o
edgeR.DGElist <- DGEList(counts = readcounts, group = sample_info.edger)

# check the result
head(edgeR.DGElist$counts)
```

```
Loading required package: limma

Attaching package: 'limma'

The following object is masked from 'package:DESeq2':

    plotMA

The following object is masked from 'package:BiocGenerics':

    plotMA
```

A matrix: 6 × 8 of type int

	ColoP14_5	ColoP14_6	ColoP28_2	ColoP28_3	ColoR34_3	ColoR34_5	ColoR37_1	ColoR37_
DDX11L1	0	0	0	0	0	0	0	
WASH7P	0	0	0	0	0	0	0	
FAM138F	0	0	0	0	0	0	0	
FAM138A	0	0	0	0	0	0	0	
OR4F5	0	0	0	0	0	0	0	
LOC729737	4	11	0	8	19	9	1	2

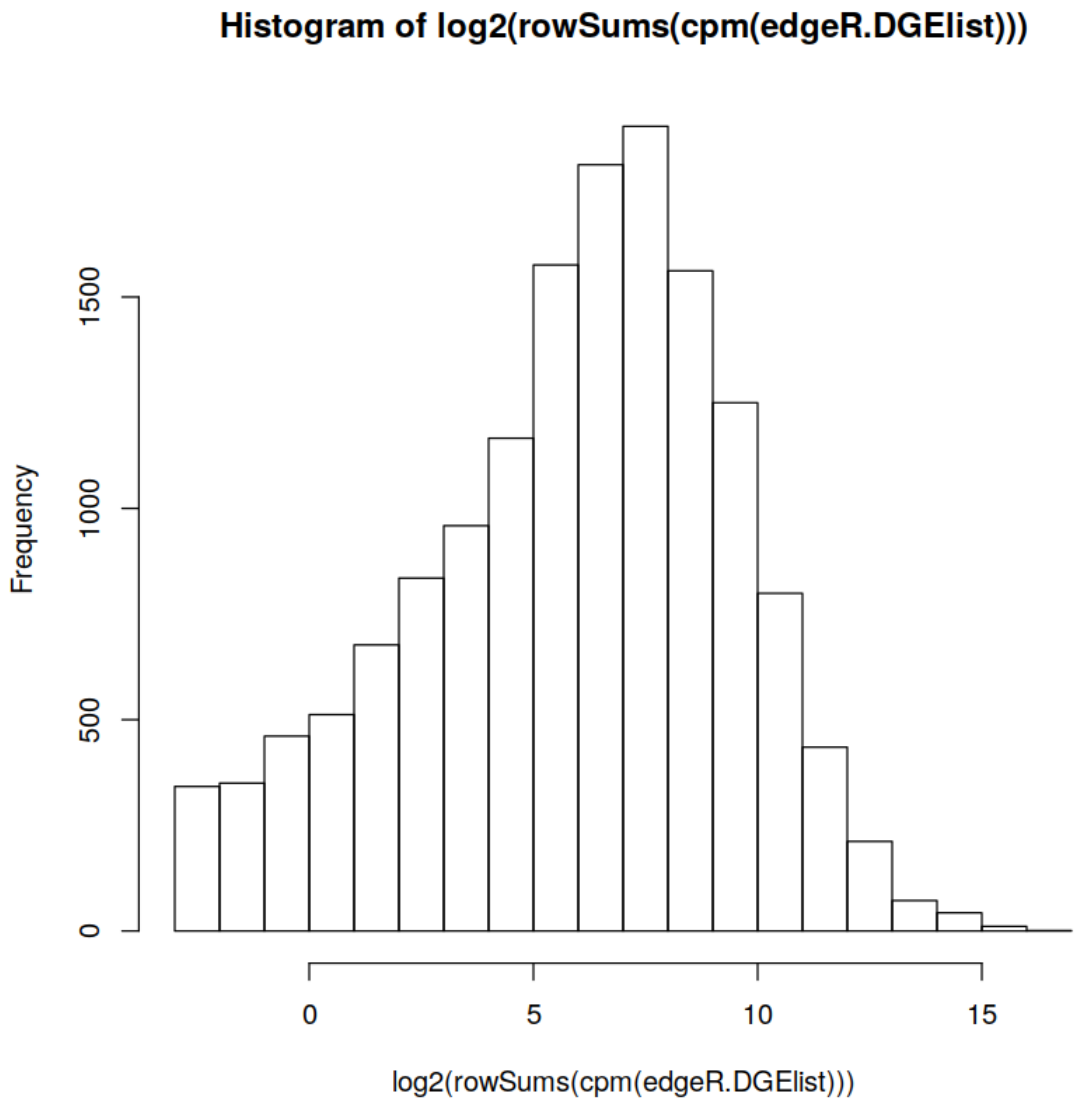
In [125]:
```
1 edgeR.DGElist$samples
```

A data.frame: 8 × 3

	group	lib.size	norm.factors
	<fct>	<dbl>	<dbl>
ColoP14_5	ColoP	3731677	1
ColoP14_6	ColoP	3811916	1
ColoP28_2	ColoP	4273587	1
ColoP28_3	ColoP	5224082	1
ColoR34_3	ColoR	5118330	1
ColoR34_5	ColoR	4203339	1
ColoR37_1	ColoR	4066180	1
ColoR37_5	ColoR	3745915	1

edgeR recommends removing genes with almost no covergage, in order to determine a sensible cutoff

In [126]:
```
1 # get an impression of the coverage across samples
2 hist(log2(rowSums(cpm(edgeR.DGElist))))
```



In [127]:
```
summary(log2(rowSums(cpm(edgeR.DGElist))))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   -Inf    -Inf   3.129    -Inf   7.214  16.553 
```

In [158]:
```
# remove genes that do not have one count per million in at least 4 sample
keep <- rowSums(cpm(edgeR.DGElist) > 0) >=4
edgeR.DGElist <- edgeR.DGElist[keep,]
```

In [159]:
```
# recompute library sizes after filtering
edgeR.DGElist$samples$lib.size <- colSums(edgeR.DGElist$counts)
edgeR.DGElist$samples
```

A data.frame: 8 × 3

	group	lib.size	norm.factors
	<fct>	<dbl>	<dbl>
ColoP14_5	ColoP	3707814	0.9986699
ColoP14_6	ColoP	3789352	0.9720278
ColoP28_2	ColoP	4248831	0.9837622
ColoP28_3	ColoP	5192212	0.9790675
ColoR34_3	ColoR	5078441	1.0413423
ColoR34_5	ColoR	4176183	0.9545256
ColoR37_1	ColoR	4021352	1.0479770
ColoR37_5	ColoR	3705025	1.0267479

In [160]:
```
# calculate normalization factor standard "TMM" if for DESeq2 then use "RL
edgeR.DGElist <- calcNormFactors ( edgeR.DGElist , method = "TMM")
edgeR.DGElist$samples
```

A data.frame: 8 × 3

	group	lib.size	norm.factors
	<fct>	<dbl>	<dbl>
ColoP14_5	ColoP	3707814	0.9992893
ColoP14_6	ColoP	3789352	0.9751147
ColoP28_2	ColoP	4248831	0.9831335
ColoP28_3	ColoP	5192212	0.9826025
ColoR34_3	ColoR	5078441	1.0423411
ColoR34_5	ColoR	4176183	0.9586103
ColoR37_1	ColoR	4021352	1.0420939
ColoR37_5	ColoR	3705025	1.0202437

To determine the differential expression in a gene-wise manner, edgeR first estimates the dispersion and subsequently tests whether the observed gene counts fit the respective negative binomial model.

In [161]:

```
# specify the design setup
design <- model.matrix(~sample_info.edger )

# estimate the dispersion for all read coounts across all samples
edgeR.DGElist <- estimateDisp(edgeR.DGElist,design = design)

# fit the negative binomial model
edger_fit <- glmFit(edgeR.DGElist, design)

# perform the testing for every gene using the neg.binomial model
edger_lrt <- glmLRT(edger_fit)

#extract results from edger_lrt$table plus adjusted p-values
DGE.results_edgeR <- topTags(edger_lrt, n = Inf,
                             sort.by = "PValue", adjust.method = "BH") # t
```

limma-voom

Limma was originally developed for the analysis of microarry gene expression data using linear models. The functions of the **limma** package have continuously been developed for much more than a decade and have laid the foundation for many widely used statistical concepts for differential gene expression analysis. In order to use the functionalities that had specifically beed developed for microarray-based data, implemented "precision weights" that are meant to transform the finicky count data (with all its statistically annoying properties including heteroskedasticity into more tractable normally distributed data. The two main differences to **edgeR** and **DESeq** are:

- count values are transformed to log-cpm;
- instead of negative binomial models, linear models are used (on the log-cpm values plus "precision weights").

The steps limma takes are:

1. For every sample and gene, calculate the counts per million reads and log-transform these.
2. Fit a linear model to the log-cpm value takig the experimental design into account (e.g., conditions, batches etc.).
3. Use the resulting residual standard deviations for every gene to fit a global mean-variance trend across all genes and samples.
4. To obtain a "precision weight" for *single* gene observation (i.e., for every sample!), the fitted log-cpm values from step 2 are used to predict the counts for every gene and every sample. The meand-variance trend (step 3) is then used to interpolate the corresponding standard deviation for these predicted counts.
5. The squared inverse of this observation-wise estimated standard deviation is used as a penalty (inverse weight) during the test for differenetial expression. These penalty values are the above mentioned "precision weights".

Like **DESeq** and **edgeR**, **lima** starts with a matrix of raw read counts where ech gene is represented by a row and the columns represent samples. **limma** assumes that rows with zero or very low counts habe been removed. In addition, size factors for sequencing depth can be calculated using **edgeR's calcNormFactors()** function.

In [162]:

```
library(limma)

rownames(design) <- colnames(edgeR.DGElist)
voomTransformed <- voom(edgeR.DGElist, design, plot = FALSE)

# fit a linear model for each gene
voomed.fitted <- lmFit(voomTransformed, design = design)

# compute moderated t-statistics, moderated F-statistics, and log-odds of
voomed.fitted <- eBayes(voomed.fitted)

# extract gene list with LogFC and statistical measures
colnames(design) # check how the coefficient is named
DGE.results_limma <- topTable(voomed.fitted, coef = "sample_info.edgerColo
                              number = Inf, adjust.method = "BH",
                              sort.by = "logFC")
```

'(Intercept)' 'sample_info.edgerColoR'

The **logFC** column gives the value of contrast. Usually this represents a log2-fold change between two or more experimental conditions although sometimes it represents a log2-expression level. The **AveExpr** column gives the average log2-expression level for that gene across all the arrays and channels in the experiment. Column t is the moderated *t-statistic*. Column **P.value** is the associated *p-value* and **adj.P.Value** is the *p-value* adjusted for multiple testing. The most popular form of adjustment is "BH" which is Benjamini and Hochberg's method to control the false discovery rate. The B-statistic is the log-odds that the gene is differentially expressed. Suppose for example that B=1.5. The odds of differential expression is *exp*(1.5) = 4.48, i.e., about for and half to one. The probalitity that the gene is differentially expressed is 4.48/(1+4.48) = 0.82, i.e., the probability is about 82% that this gene is differentially expressed. A B-statistic is automatically adjusted for multiple testing by assuming that 1% of the genes, or some other percentage specified by the user in the call to eBayes(), are expected to be differentially expressed. The p-values and B-statistics will normally rank genes in the same order. In fact, if the data contains no missing values or quality weights, then the order will be precisely the same.

Evaluation DGE Results

Once you have obtained a table of genes that show signs of differential expression, you habe reached one of the most important milestones of RNA-seq analysis! To evaluate how confident you can be in that list of DE genes, you should look at serveral aspects of the analyses you die and perform basic checks on your results:

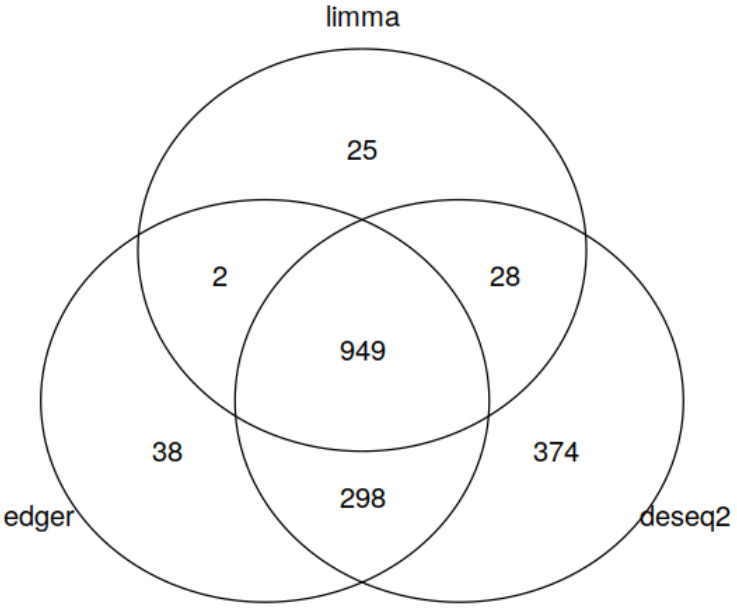
1. Did the unsupervised clustering and PCA analysis reproduce the major trends of the initial experiment? For example, did replicatees of the same condition cluster together and were they separated from the replicates of the other conditions's?
2. How well do different DGE programs agree on the final list of DE genes? You may want to consider performing downstream analyses only on the list of genes that were identified as DE by more than one tool.
3. DO the results of the DGE analysis agree with results from small-scale experiments? Can you reproduce qPCR results (and vice versa: can you reproduce the results of the DGE analysis with qPCR)?
4. How robust are the observed fold changes? Can they explain the effects you see on a phenotypic level?

!Note If your RNAseq results are suggesting expression changes that differ dramatically from everything you would have expected based on prior knowledge, you should be very cautious!

The following code and images should just give you some examples of typical follow-up visualizations that can be done. To avoid having to load full libraries, we will use the syntax **R library::function**, i.e., **gplots::venn** uses the function **venn()** of the **gplots** package. This will directly call that function without loading all other functions of **gplots** into the working environment.

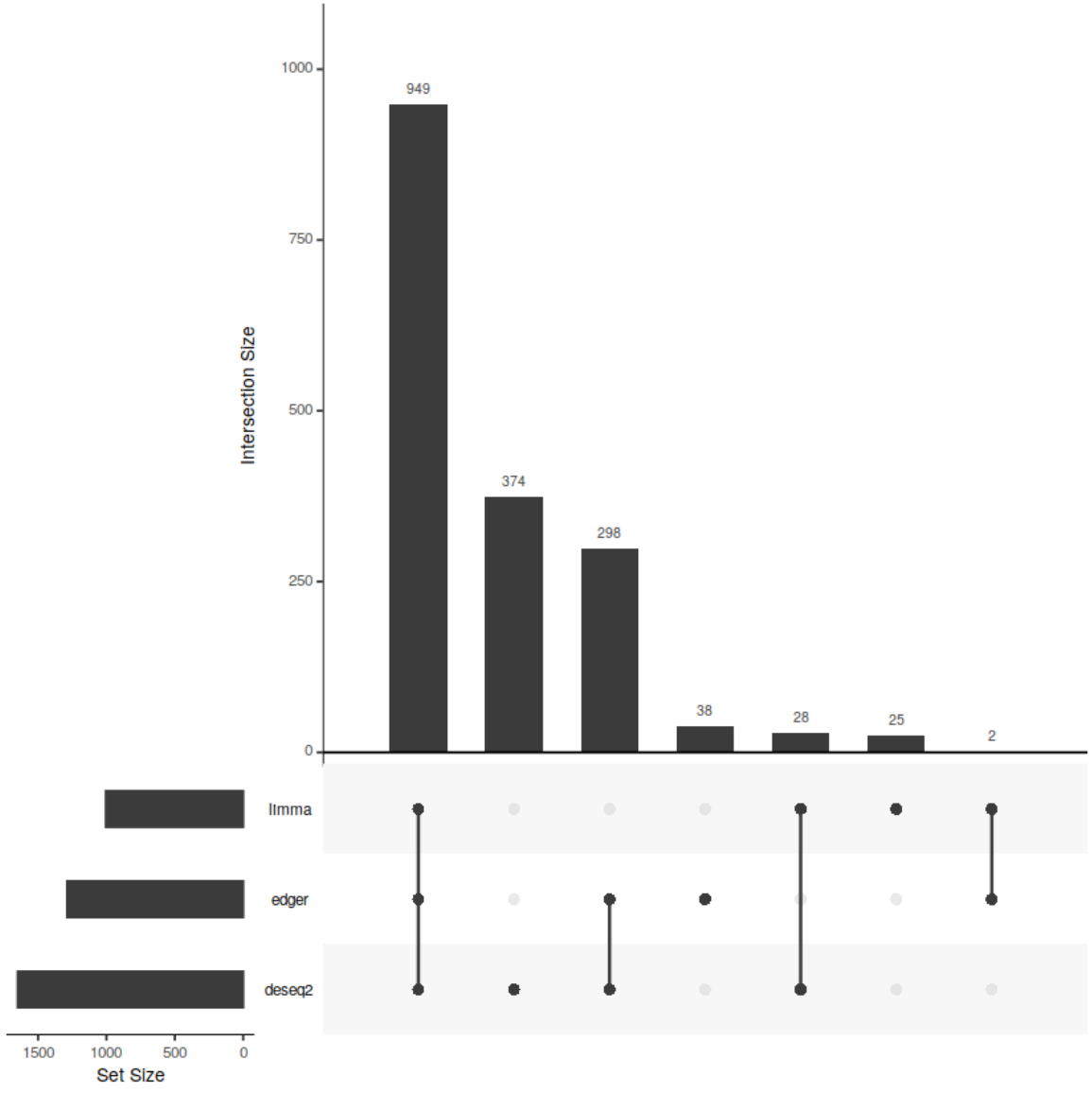
In [163]:

```
# make a Venn diagram
DE_list <- list(edger = rownames(subset(DGE.results_edgeR$table, FDR<=0.05
                deseq2 = rownames(subset(DGE.results, padj<=0.05)),
                limma = rownames(subset(DGE.results_limma, adj.P.Val<=0.05
gplots::venn(DE_list)
```



In [164]:

```
# more sophisticated venn alternative
DE_gns <- UpSetR::fromList(DE_list)
UpSetR::upset(DE_gns, order.by = "freq")
```



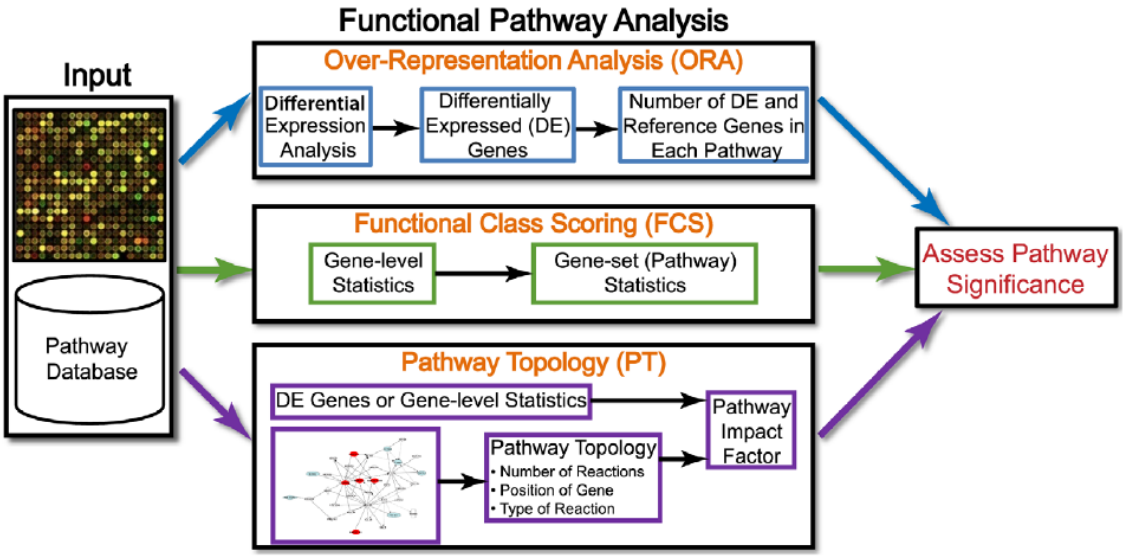
Functional Analyses

The output of RNAseq differential expression is a list of significant differentially expressed genes (DGEs). To gain greater biological insight on the differentially expressed genes there are various analyses that can be done.

Most downstream analyses will be very specific to your question of interest and the model system you are studying. Generally, most downstream analyses are aimed at elucidating the functions of the differentially expressed genes and to identify possible patterns among them.

- enrichments of certain gene ontology (GO) terms encompassing the three classes of GO terms: biological processes, cell components and molecular functions
- enrichments of certain pathways such as those defined by MSigDB, STRING of KEGG
- identification of specific "master" regulators or transcription factors that may underlie a bulk of the changes that are seen.

Generally for any differential expression analysis, it is useful to interpret the resulting gene lists using freely available web- and R-based tools. While tools for functional analysis span a wide variety of techniques, they can loosely be categorized into three main types: over-representation analysis, functional class scoring, and pathway topology.



Enrichments are typically assessed by either:

- over-representation analysis (ORA)
- gene set enrichment analyses (GSEA)

The goal of functional analysis is provide biological insight, so it's necessary to analyze results in the context of the experimental hypothesis.

!Notethat all tools described below are there to validate experimental results and to make hypotheses. These tools suggest genes/pathways that may be involved with your condition of interest; however, you should NOT use these tools to make conclusions about the pathways involved in your experimental process. You will need to perform experimental validation of any suggested pathways.

All types of enrichment analyses are based on comparisons *between genes*, that means that the gene-specific biases such as gene length may cause spurious findings. In fact it is described that long transcripts and genes are more likely to (a) be detected as differentially expressed and (b) tend to be over-represented in most commonly used databases because their length makes it more likely that they are detected across many different experiments. The **goseq** package therefor tries to correct for the inherently increased likelihood of long genes to be present on your list of interest.

Over-representation analyses

These types of analyses rely on a filtered list of genes of interest, e.g. all genes that pass the DE filter. This list is compared to the genes that are known to be part of a specific pathway or a generic set of interest, e.g. "Glycolysis". A statistical test (e.g. hypergeometric test) is then used to determine whether the overlap between the gene list of interest and the known pathway is greater than expected by chance. While this approach is relatively straight-forward, there are serious limitations including the fact that both magnitude and direction of the change of individual genes are completely disregarded; the only measure that matters is the presence of absence of a given gene within the list that are being compared.

Gene Ontology project

One of the most widely-used categorizations is the **Gene Ontology (GO)** established by the Gene Ontology project. The Gene Ontology project is a collaborative effort to address the need for consistent descriptions of gene products across databases. The Gene Ontology Consortium maintains the GO terms, and these GO terms are incorporated into gene annotations in m any of the popular repositories for animal, plant and microbial genomes.

Tools that inverstigate enrichment of biological functions or interactions often use the Gene Ontology (GO) categorizations, i.e. the GO terms to determine whether any have significantly modified representations in a given list of genes. Therefore, to best use and interpret the result from these

functional analysis tools, it is helpful to have a good understanding of the GO terms themselves and their organization.

GO Ontologies

To describe the roles of genes and gene products, GO terms are organized into three independent controlled vocabularies (ontologies) in a species-independent manner:

- **Biological process**: refers to the biological role involving the gene or gene product, and could include "transcription", "signal transduction", and "apoptosis". A biological process generally involves a chemical or physical change of the starting material or input.
- **Molecular function**: represents the biochemical activity of the gene product, such activities could include "ligand", "GTPase", and "transporter".
- **Cellular component**: refers to the location in the cell of the gene product. Cellular components could include "nucleus", "lysosome", and "plasma membrane".

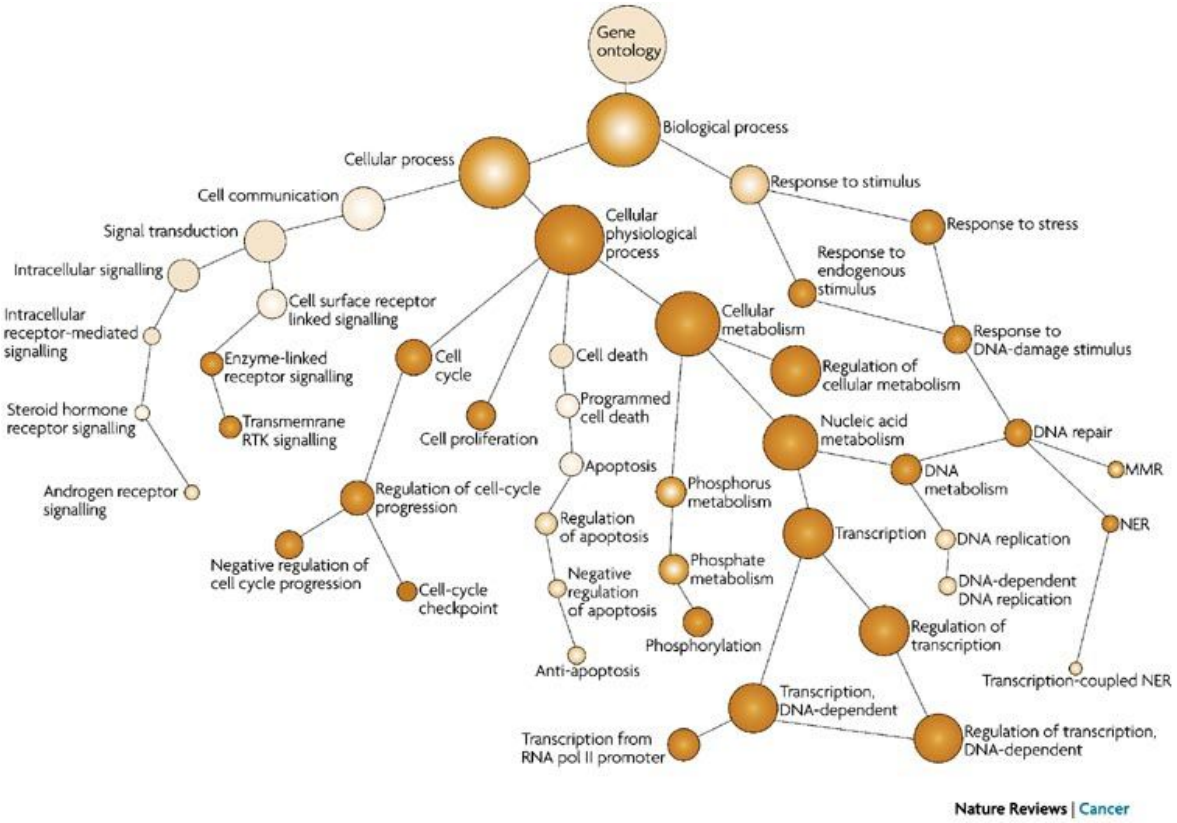
Each GO term has a term name (e.g. **DNA repair**) and a unique term accession number (**GO:0005125**), and a single gene product can be associated with many GO terms, since a single gene product "may function in several processes, contain domains that cary out diverse molecular functions, and participate in multiple alternative interactions with other proteins, organells or locations in the cell."

GO term hierachy

Some gene products are well-researched, with vast quantities of data available regarding their biological processes and functions. However, other gene products habe very little data available about their roles in the cell.

For example, the protein "p53", would contain a wealth of information on it's roles in the cell, whereas another protein might only be known as a "membrane-bound protein" with no other information available.

The GO ontologies were developed to describe and query biological knowledge with differing levels of information available. To do this, GO ontologies are loosely hierachical, ranging from general, 'parent' terms to more specific 'child' terms. The GO ontologies are "loosely" hierachical since 'child' terms can have multiple 'parent' terms.



Gene set entichment analyses

To address some of the limitations of the ORA approach, functional scoring algorithms typically do not require a pre-selected list of genes; instead, the require a fairly exhaustive list of all genes that could make up your "universe" of genes. These genes should have some measure of change by which they will be sorted. The basic assumption is that although large changes in individual genes can have significant effects on pathways, weaker but *coordinated changes* in *sets of functionally related genes* can also have significant effects. Therefore, the gene-level statistics for all genes in a pathway are aggregated into a single pathway-level statistic (e.g. the sum of all log-fold changes), which will then be evaluated.

While GSEA does take the magnitude and direction of change into consideration, pathways are regarded as independent units despite the fact that many pathways share individual genes.

!Note The vast majority of downstream analyses are **hypothesis-genereating** tools. Most gene set tests are not robust the changes in sample size, gene size, experimental design and fold-change biases.

goseq

In order to perform a GO analysis of your RNAseq data, **goseq** only requires a simple named vector, which contains two pieces of information.

1. **Measured genes**: all genes for which RNAseq data was gathered for your experiment. Each element of your vector should be named by a unique gene identifier.
2. **Differentially expressed genes**: each element of your vector should be either a 1 or o 0, where 1 indicates that the gene is differentially expressed and 0 that it is not.

To begin the analysis, **goseq** first needs to quantify the lenght bias present in the dataset under consideration. This is done by calculating a Probability Weighting Function or PWF which can be thought of as a function which gives the probability that a gene will be differentially expressed (DE), based on its length alone. The PWF is calculated by fitting a monotonic spline to the the binary data series of differential expression (1=DE, 0=Not DE) as a function of gene length. The PWF is used to weight the chance of selecting each gene when forming a null distribution for GO category membership. The fact that the PWF is calculated directly from the dataset under consideration makes this approach robust, only correcting for the length bias present in the data.

In [209]:
```
# format the DE genes into a vector suitable for use with goseq
DGE_results <- results(DESeq.ds,independentFiltering = TRUE, alpha = 0.05)
DGEgenes <- rownames(subset(DGE_results, padj < 0.05))
gene.vector <- row.names(DGE_results) %in% DGEgenes %>% as.integer
names(gene.vector) <- row.names(DGE_results)
table(gene.vector)
```

```
gene.vector
    0     1
13331  1649
```

nullp plots the resulting fit, allowing verification of the goodness of fit before continuing the analysis. The output of nullp contains all the data used to create the PWF, as well as the PWF itself. It is a data frame with 3 columns, named "DEgenes", "bias.data" and "pwf" with the rownames set to the gene names. Each row corresponds to a gene with the DEgenes column specifying if the gene is DE (1 for DE, 0 for not DE), the bias.data column giving the numeric value of the DE bias being accounted for (usually the gene length or number of counts) and the pwf column giving the genes value on the probability weighting function.

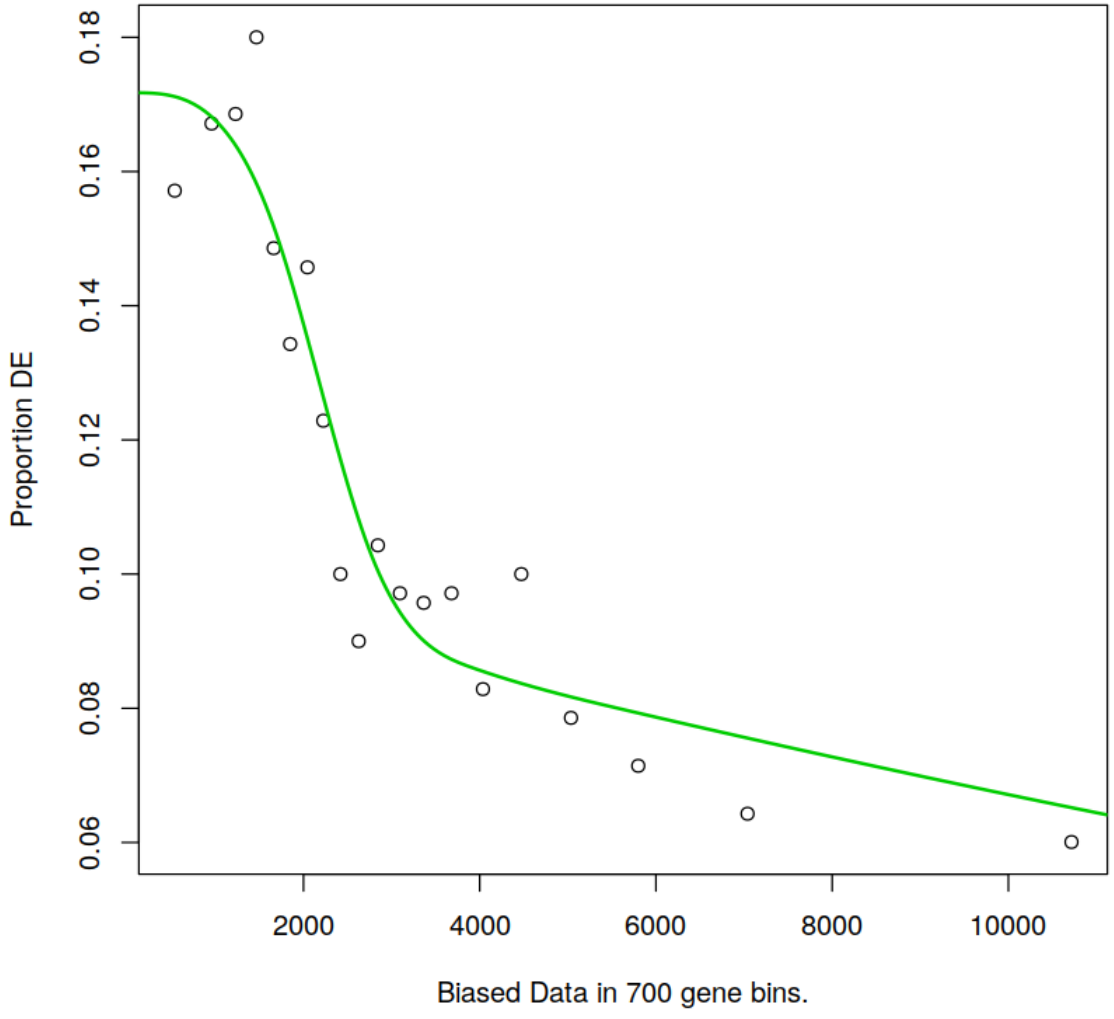
In [198]:

```
# Fitting the Probability Weighting Function (PWF)
library(goseq)
pwf <- nullp(gene.vector, "hg19", "geneSymbol")
head(pwf)
```

```
Warning message in library():
"libraries '/usr/local/lib/R/site-library', '/usr/lib/R/site-library' contain
no packages"Loading hg19 length data...
```

A data.frame: 6 × 3

	DEgenes	bias.data	pwf
	<int>	<dbl>	<dbl>
WASH7P	0	NA	NA
LOC729737	0	NA	NA
LOC100288069	1	NA	NA
LINC00115	0	NA	NA
LOC643837	0	1550	0.1556621
NOC2L	0	2818	0.1013010



Using the Wallenius approximation. To start we will use the default method, to calculate the over and under expressed GO categories among DE genes. The resulting object is ordered by GO category over representation amongst DE genes.

In [199]:

```
GO.wall=goseq(pwf,"hg19","geneSymbol")
head(GO.wall)
```

```
Fetching GO annotations...
For 2472 genes, we could not find any categories. These genes will be exclude
d.
To force their use, please run with use_genes_without_cat=TRUE (see documenta
tion).
This was the default behavior for version 1.15.1 and earlier.
Calculating the p-values...
'select()' returned 1:1 mapping between keys and columns
```

A data.frame: 6 × 7

	category	over_represented_pvalue	under_represented_pvalue	numDEInCat	numInCat	te
	<chr>	<dbl>	<dbl>	<int>	<int>	<ch
10780	GO:0044444	3.956705e-23	1	1007	7141	cytoplasm p
18605	GO:1903561	1.565804e-21	1	331	1670	extracellu vesi
10280	GO:0043230	1.750708e-21	1	331	1671	extracellu organe
14470	GO:0070062	2.059654e-21	1	329	1658	extracellu exoso
2432	GO:0005737	2.258883e-21	1	1110	8168	cytopla
2375	GO:0005615	3.708884e-20	1	380	2028	extracellu spa

Using random sampling. It may sometimes be desirable to use random sampling to generate the null distribution for category membership. For example, to check consistency against results from Wallenius approximation.

In [210]:

```
GO.samp=goseq(pwf,"hg19","geneSymbol",method="Sampling",repcnt=1000)
head(GO.samp)
```

```
Fetching GO annotations...
For 2472 genes, we could not find any categories. These genes will be exclude
d.
To force their use, please run with use_genes_without_cat=TRUE (see documenta
tion).
This was the default behavior for version 1.15.1 and earlier.
Running the simulation...

100 %

Calculating the p-values...
'select()' returned 1:1 mapping between keys and columns
```

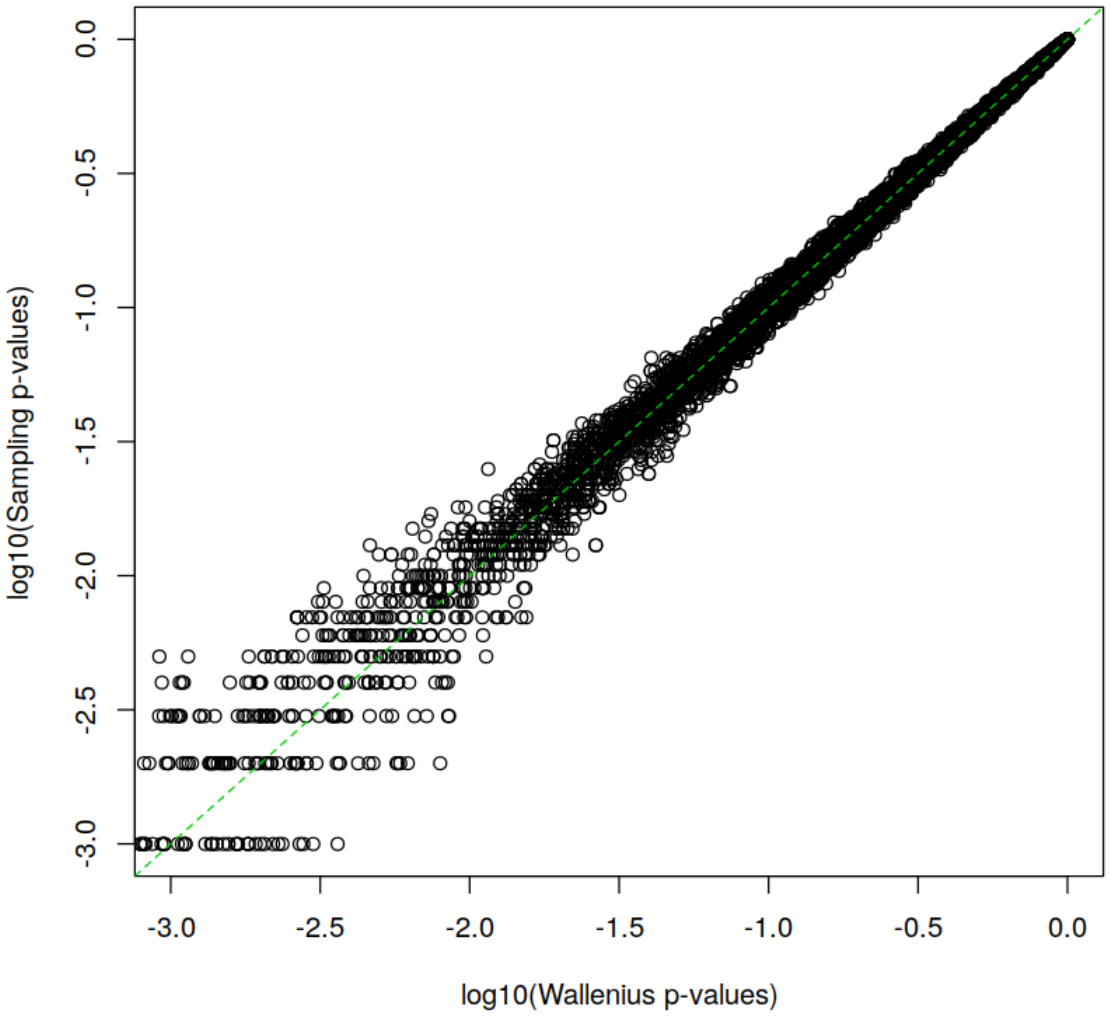
A data.frame: 6 × 7

	category	over_represented_pvalue	under_represented_pvalue	numDEInCat	numInCat	term
	<chr>	<dbl>	<dbl>	<int>	<int>	<chr>
145	GO:0000313	0.000999001	1	24	80	organella ribosom
150	GO:0000323	0.000999001	1	98	520	lytic vacuol
545	GO:0001775	0.000999001	1	171	950	ce activatio
592	GO:0001836	0.000999001	1	18	50	release c cytochrom c fro mitochondri
834	GO:0002252	0.000999001	1	152	824	immun effecto proces
839	GO:0002263	0.000999001	1	106	515	ce activatio involved i immun respons

Plotting the p-values against one another, we see that there is little difference between the two methods.

In [212]:

```
plot(log10(GO.wall[,2]), log10(GO.samp[match(GO.wall[,1],GO.samp[,1]),2]),
 xlab="log10(Wallenius p-values)",ylab="log10(Sampling p-values)",
 xlim=c(-3,0))
abline(0,1,col=3,lty=2)
```



Ignoring lenght bias. goseq also allows for one to perform a GO analysis without correcting for RNAseq length bias. In practice, this is only useful for assessing the effect of length bias on your results. You should NEVER use this option as yout final analysis.

In [213]:

```
GO.nobias=goseq(pwf,"hg19","geneSymbol",method="Hypergeometric")
head(GO.nobias)
```

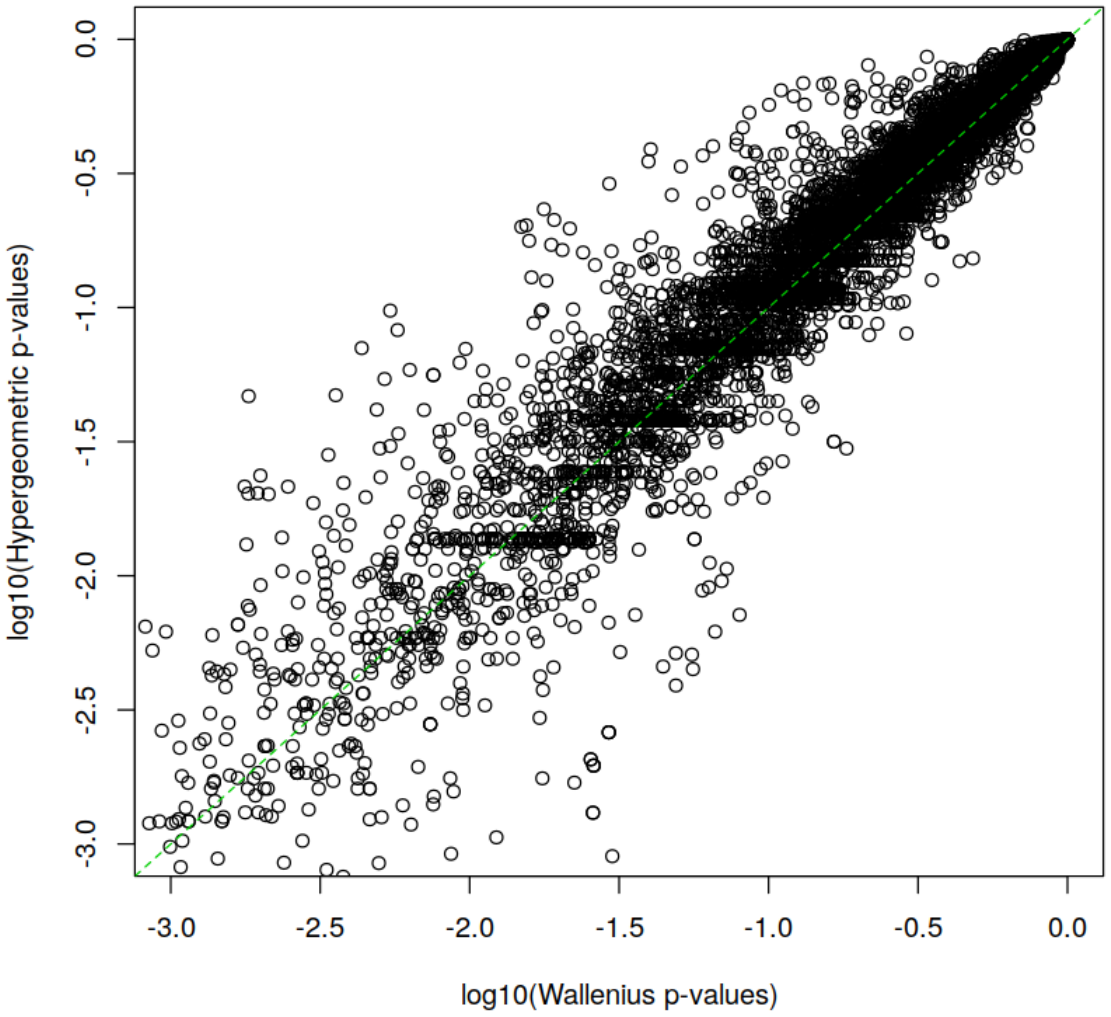
```
Fetching GO annotations...
For 2472 genes, we could not find any categories. These genes will be exclude
d.
To force their use, please run with use_genes_without_cat=TRUE (see documenta
tion).
This was the default behavior for version 1.15.1 and earlier.
Calculating the p-values...
'select()' returned 1:1 mapping between keys and columns
```

A data.frame: 6 × 7

	category	over_represented_pvalue	under_represented_pvalue	numDEInCat	numInCat	te
	<chr>	<dbl>	<dbl>	<int>	<int>	<ch
18605	GO:1903561	3.163313e-25	1	331	1670	extracellu vesi
10280	GO:0043230	3.534236e-25	1	331	1671	extracellu organe
14470	GO:0070062	3.765618e-25	1	329	1658	extracellu exoso
2375	GO:0005615	2.313043e-24	1	380	2028	extracellu spa
10764	GO:0044421	1.321696e-22	1	394	2170	extracellu region p
10780	GO:0044444	2.288605e-22	1	1007	7141	cytoplasr p

In [214]:

```
plot(log10(GO.wall[,2]), log10(GO.nobias[match(GO.wall[,1],GO.nobias[,1]),
 xlab="log10(Wallenius p-values)", ylab="log10(Hypergeometric p-values)",
 xlim=c(-3,0), ylim=c(-3,0))
abline(0,1,col=3,lty=2)
```



Limiting GO categories. By default, goseq tests all three major Gene Ontology branches; Cellular Components, Biological Processes and Molecular Functions. However, it is possible to limit testing to any combination of the major branches by using the test.cats argument to the goseq function. This is done by specifying a vector consisting of some combination of the strings "GO:CC", "GO:BP" and "GO:MF".

In [215]:

```
GO.MF=goseq(pwf,"hg19","geneSymbol",test.cats=c("GO:MF"))
head(GO.MF)
```

```
Fetching GO annotations...
For 2472 genes, we could not find any categories. These genes will be exclude
d.
To force their use, please run with use_genes_without_cat=TRUE (see documenta
tion).
This was the default behavior for version 1.15.1 and earlier.
Calculating the p-values...
'select()' returned 1:1 mapping between keys and columns
```

A data.frame: 6 × 7

	category	over_represented_pvalue	under_represented_pvalue	numDEInCat	numInCat	
	<chr>	<dbl>	<dbl>	<int>	<int>	
3211	GO:0050839	1.587721e-07	1.0000000	74	369	cell a molecule
2892	GO:0045296	2.949123e-06	0.9999988	58	279	cadherin
3702	GO:0098632	2.335715e-05	0.9999964	12	28	cell-cell a mediato
192	GO:0003674	3.030108e-05	0.9999818	1370	11410	molecular_
2238	GO:0031625	4.453476e-05	0.9999787	49	245	ubiquitir ligase
3098	GO:0048037	8.325428e-05	0.9999544	67	349	cofactor

Making sense of the results. Having performed the GO analysis, you may now wish to interpret the results. If you wish to identify categories significantly enriched/unenriched below some p-value cutoff, it is necessary to first apply some kind of multiple hypothesis testing correction. For example, GO categories over enriched using a .05 FDR cutoff Benjamini and Hochberg.

In [205]:

```
library(GO.db)

enriched.GO=GO.wall$category[p.adjust(GO.wall$over_represented_pvalue,
method="BH")<.05]
for(go in enriched.GO[1:10]){
 print(GOTERM[[go]])

 cat("--------------------------------------\n")}
```

```
GOID: GO:0044444
Term: cytoplasmic part
Ontology: CC
Definition: Any constituent part of the cytoplasm, all of the contents
    of a cell excluding the plasma membrane and nucleus, but including
    other subcellular structures.
Synonym: cytoplasm component
--------------------------------------
GOID: GO:1903561
Term: extracellular vesicle
Ontology: CC
Definition: NA
Synonym: microparticle
--------------------------------------
GOID: GO:0043230
Term: extracellular organelle
Ontology: CC
Definition: Organized structure of distinctive morphology and function,
    occurring outside the cell. Includes, for example, extracellular
    membrane vesicles (EMVs) and the cellulosomes of anaerobic bacteria
    and fungi.
--------------------------------------
GOID: GO:0070062
Term: extracellular exosome
Ontology: CC
Definition: A vesicle that is released into the extracellular region by
    fusion of the limiting endosomal membrane of a multivesicular body
    with the plasma membrane. Extracellular exosomes, also simply
    called exosomes, have a diameter of about 40-100 nm.
Synonym: exosome
Synonym: extracellular vesicular exosome
--------------------------------------
GOID: GO:0005737
Term: cytoplasm
Ontology: CC
Definition: All of the contents of a cell excluding the plasma membrane
    and nucleus, but including other subcellular structures.
--------------------------------------
GOID: GO:0005615
Term: extracellular space
Ontology: CC
Definition: That part of a multicellular organism outside the cells
    proper, usually taken to be outside the plasma membranes, and
    occupied by fluid.
Synonym: intercellular space
--------------------------------------
GOID: GO:0044421
Term: extracellular region part
Ontology: CC
Definition: Any constituent part of the extracellular region, the space
    external to the outermost structure of a cell. For cells without
    external protective or external encapsulating structures this
    refers to space outside of the plasma membrane. This term covers
    constituent parts of the host cell environment outside an
    intracellular parasite.
Synonym: extracellular structure
--------------------------------------
GOID: GO:0031982
```

```
Term: vesicle
Ontology: CC
Definition: Any small, fluid-filled, spherical organelle enclosed by
    membrane.
Synonym: membrane-bounded vesicle
Synonym: membrane-enclosed vesicle
Synonym: GO:0031988
Secondary: GO:0031988
-------------------------------------
GOID: GO:0005576
Term: extracellular region
Ontology: CC
Definition: The space external to the outermost structure of a cell.
    For cells without external protective or external encapsulating
    structures this refers to space outside of the plasma membrane.
    This term covers the host cell environment outside an intracellular
    parasite.
Synonym: extracellular
-------------------------------------
GOID: GO:0016020
Term: membrane
Ontology: CC
Definition: A lipid bilayer along with all the proteins and protein
    complexes embedded in it an attached to it.
-------------------------------------
```

In [216]:

```
# retrieving the GO categories assigned to each gene
go_gns <- getgo( rownames(DGE_results), 'hg19', 'geneSymbol') %>% stack
# in principle, the gene information could be added to the goseq results:
merge(GO.wall, go_gns, by.x = "category", by.y = "values") %>% dim

```

```
Warning message in stack.default(.):
"non-vector elements will be ignored"

1527338  8
```

In [219]:

```
# export a list of over-represented GO terms plus the corresponding p-valu
# which is the input type that REVIGO needs
subset(GO.wall, over_represented_pvalue < 0.01,
select = c("category","over_represented_pvalue")) %>%
write.table(., file = "~/projects/RNASeq/Enriched_GOterms_goseq.txt",
quote = FALSE, row.names = FALSE, col.names = FALSE)
```

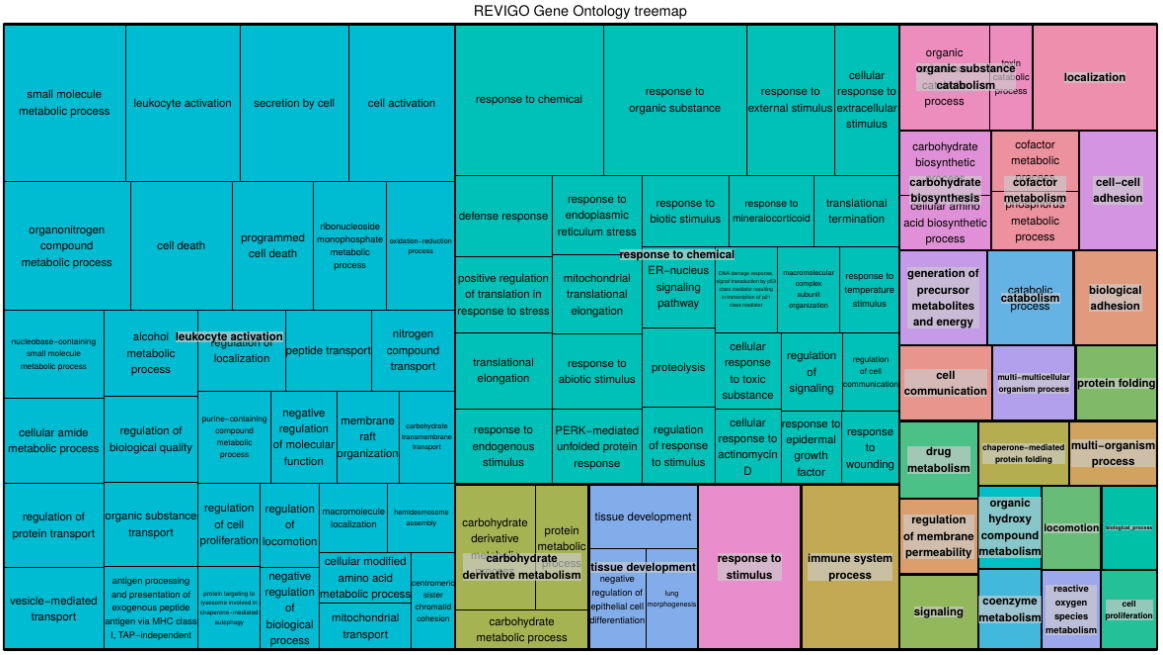
REVIGO

REVIGO (http://revigo.irb.hr/) is a web-based tool that can take a list of GO terms, collaps redundant terms by semantic similarity, and summarize them graphically.

Open the Enriched_GOterms_goseq.txt and copy the contents into the REVIGO search box, and submit.

After the program runs, there may not be output to the screen, but you can click on the Treemap tab. At the bottom of the Treemap tab should be a link to an R script to create the treemap; click to download the script.

In RStudio, pull-down the File menu and choose Open File, then navigate to the REVIGO_treemap.r script to open. In the REVIGO_treemap.r script tab, scroll down the script and replace the tmPlot() command to treemap(), then execute the script. It will genereate a pdf.file with your results.



GO and Gene set enrichment of KEGG pathways using ClusterProfiler

ClusterProfiler offers access to a set of R packages that all deal with over-representation analyses and gene set enrichment analyses. To perform GSEA using the pathways annotated and curated by KEGG (Kyoto Encyclopedia of genes and Genomes) you will need to provide a list of all genes that you analyzed. Those genes should be sorted by some sort of metric (e.g. log2FC). This sorted vector of gene-value pairs is then compared to the KEGG pathways and those pathways that seem to have a consistent enrichment for positive (or negeative) fold changes across all of the pathway's genes will be highlighted.

Gene Ontology analysis

In **clusterProfiler**, groupGO is designed for gene classification based on GO distribution at a specific level. The input parameters of *gene* is a vector of gene IDs. It expects entrezgene.

In [208]:

```
library(clusterProfiler)
# clusterProfiler requires a sorted vector where the values correspond
# to the measure used for the sorting
DGE_cp_results <- DGE.results[order(-1*DGE.results$log2FoldChange),]
genes_for_cp <- DGE_cp_results$log2FoldChange
names(genes_for_cp) <- row.names(DGE_cp_results)
head(genes_for_cp)
```

KIF6	9.29220239736187
GMPR	8.86700110786199
LOC100127888	8.74709173840927
SLMO1	8.28357801534713
ALDH1A2	8.19290503939129
PELI2	8.16348800833352

In [232]:

```
# transform Symbol to EntrezID
eg = bitr(row.names(DGE.results), fromType="SYMBOL", toType="ENTREZID", Or
head(eg)
names(genes_for_cp) <- eg$ENTREZID
head(genes_for_cp)
```

```
'select()' returned 1:many mapping between keys and columns
Warning message in bitr(row.names(DGE.results), fromType = "SYMBOL", toType =
"ENTREZID", :
"10.84% of input gene IDs are fail to map..."
```

A data.frame: 6 × 2

	SYMBOL	ENTREZID
	<chr>	<chr>
1	WASH7P	653635
2	LOC729737	729737
3	LOC100288069	100288069
4	LINC00115	79854
6	NOC2L	26155
7	KLHL17	339451

653635	9.29220239736187
729737	8.86700110786199
100288069	8.74709173840927
79854	8.28357801534713
26155	8.19290503939129
339451	8.16348800833352

In [553]:

```
library("DOSE")
data(geneList)
gene <- names(genes_for_cp)[abs(geneList) > 1.5]
head(gene)
```

'653635' '729737' '100288069' '79854' '26155' '339451'

In [556]:

```
ggo <- groupGO(gene = gene,
               'org.Hs.eg.db',
               ont = "BP",
               level = 2,
               readable = TRUE)
head(ggo)
```

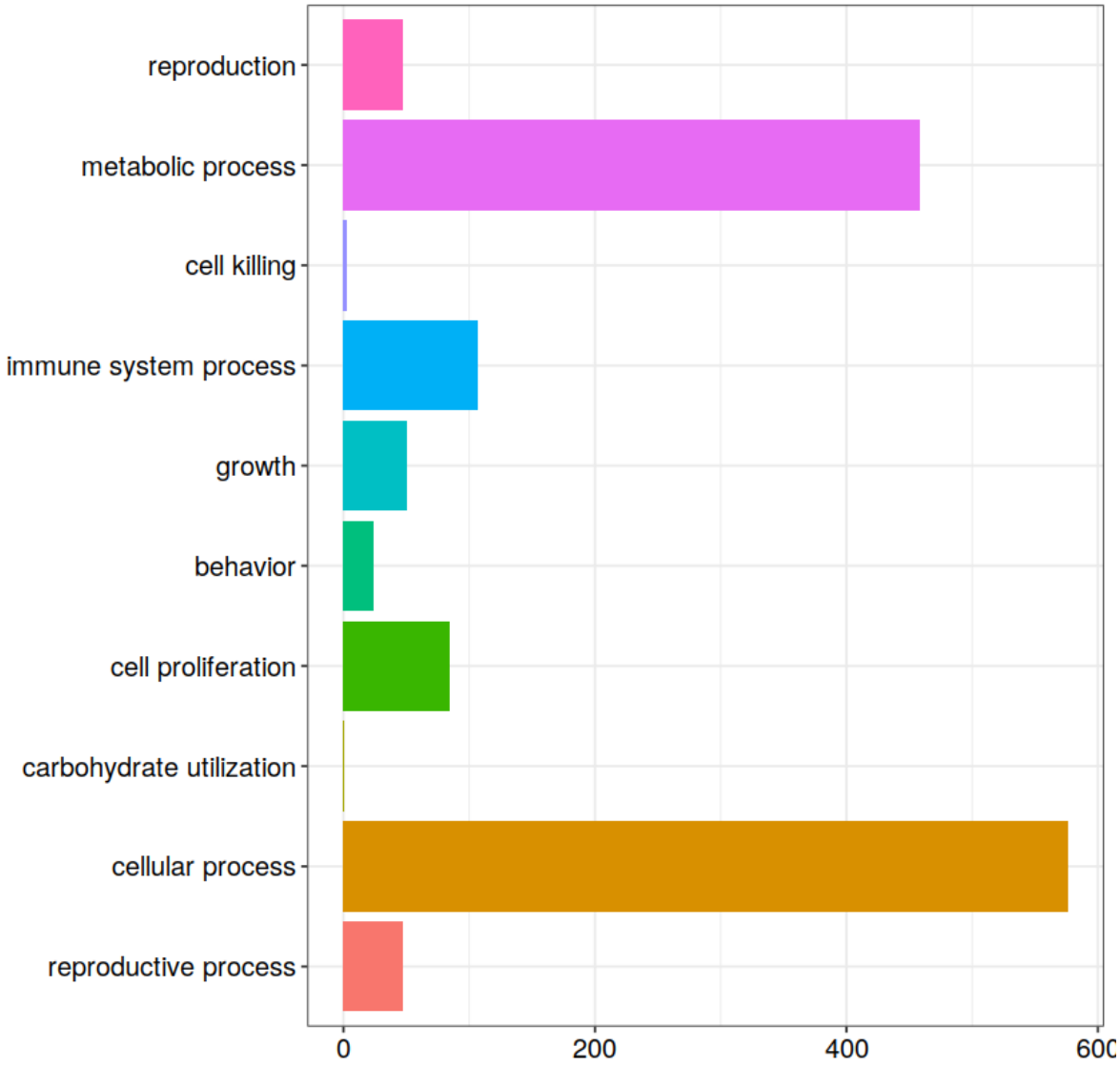
A data.frame: 6 × 5

	ID	Description	Count	GeneRatio	
	<fct>	<fct>	<int>	<fct>	
GO:0000003	GO:0000003	reproduction	47	47/741	
GO:0008152	GO:0008152	metabolic process	458	458/741	NOC2L/KLHL17/PLEKHN1/HES4/ISG15/AGRN/TNFR
GO:0001906	GO:0001906	cell killing	3	3/741	
GO:0002376	GO:0002376	immune system process	107	107/741	
GO:0006791	GO:0006791	sulfur utilization	0	0/741	
GO:0006794	GO:0006794	phosphorus utilization	0	0/741	

GO over-representation test

Over-representation test is used as approach to identify biological themes. To determine whether any terms annotate a specified list of genes at frequency greter than expected by chance, clusterProfiler calculates a p-value using hypergeomtric distribution. P-values are adjusted for multiple comparison, and p-values are also calculated for FDR control.

In [557]:
```
1 barplot(ggo, drop=TRUE, showCategory=10)
```



clusterProfiler GO

We will be using clusterProfiler to perform over-representation analysis on GO terms associated with our list of significant genes. The tool takes as input a significant gene list and a background gene list and performs statistical enrichment analysis using hypergeometric testing. The basic arguments allow the user to select the appropriate organism and GO ontoligy (BP, CC, MF) to test.

Running clusterProfiler

To run clusterProfiler GO over-representation analysis, we will change our gene names into Ensembl IDs, since the tool works a bit easier with the Ensembl IDs.

In [399]:
```
1 # load libraries
2 library(org.Hs.eg.db)
3 library(DOSE)
4 library(pathview)
5 library(clusterProfiler)
6 library(AnnotationHub)
7 library(ensembldb)
8 library(tidyverse)
9 library(annotables)
```

In [400]:

```
## Explore the grch37 table loaded by the annotables library
grch37
```

A tibble: 66978 × 9

ensgene	entrez	symbol	chr	start	end	strand
<chr>	<int>	<chr>	<chr>	<int>	<int>	<int>
ENSG00000000003	7105	TSPAN6	X	99883667	99894988	-1
ENSG00000000005	64102	TNMD	X	99839799	99854882	1
ENSG00000000419	8813	DPM1	20	49551404	49575092	-1
ENSG00000000457	57147	SCYL3	1	169818772	169863408	-1
ENSG00000000460	55732	C1orf112	1	169631245	169823221	1

In [421]:

```
# Return the IDs for the gene symbols in the DE results
idx <- grch37$symbol %in% res_tableOE_tb$gene
ids <- grch37[idx, ]
ids
```

A tibble: 16587 × 9

ensgene	entrez	symbol	chr	start	end	strand	
<chr>	<int>	<chr>	<chr>	<int>	<int>	<int>	
ENSG00000000003	7105	TSPAN6	X	99883667	99894988	-1	pro
ENSG00000000419	8813	DPM1	20	49551404	49575092	-1	pro
ENSG00000000457	57147	SCYL3	1	169818772	169863408	-1	pro
ENSG00000000460	55732	C1orf112	1	169631245	169823221	1	pro

In [422]:

```
# The gene names can map to more than one Ensembl ID (some genes change ID
# so we need to remove duplicate IDs prior to assessing enriched GO terms
non_duplicates <- which(duplicated(ids$symbol) == FALSE)

ids <- ids[non_duplicates, ]
## Merge the IDs with the results
res_ids <- inner_join(res_tableOE_tb, ids, by=c("gene"="symbol"))

```

To perform the over-representation analysis, we need a list of background genes and a list of significant genes. FOr our backgroudn dataset we will use all genes tested for differential expression (all genes in our results table). For our significant gene list we will use genes with p-adjusted values less than 0.05 (we could include a fold change threshold too if we have many DE genes).

In [423]:
```r
# Create background dataset for hypergeometric testing using all genes tes
allOE_genes <- as.character(res_ids$ensgene)
# Extract significant results
sigOE <- dplyr::filter(res_ids, padj < 0.05)
sigOE_genes <- as.character(sigOE$ensgene)
```

In [523]:
```r
# Run GO enrichment analysis
ego <- enrichGO(gene = sigOE_genes,
universe = allOE_genes,
keyType = "ENSEMBL",
OrgDb = org.Hs.eg.db,
ont = "BP",
pAdjustMethod = "BH",
qvalueCutoff = 0.05,
readable = TRUE)
# Output results from GO analysis to a table
cluster_summary <- data.frame(ego)
```

In [524]:
```r
# write the summary in a csv.file
write.csv(cluster_summary, "clusterProfiler_MF.csv")
```

Visualzing clusterProfiler results

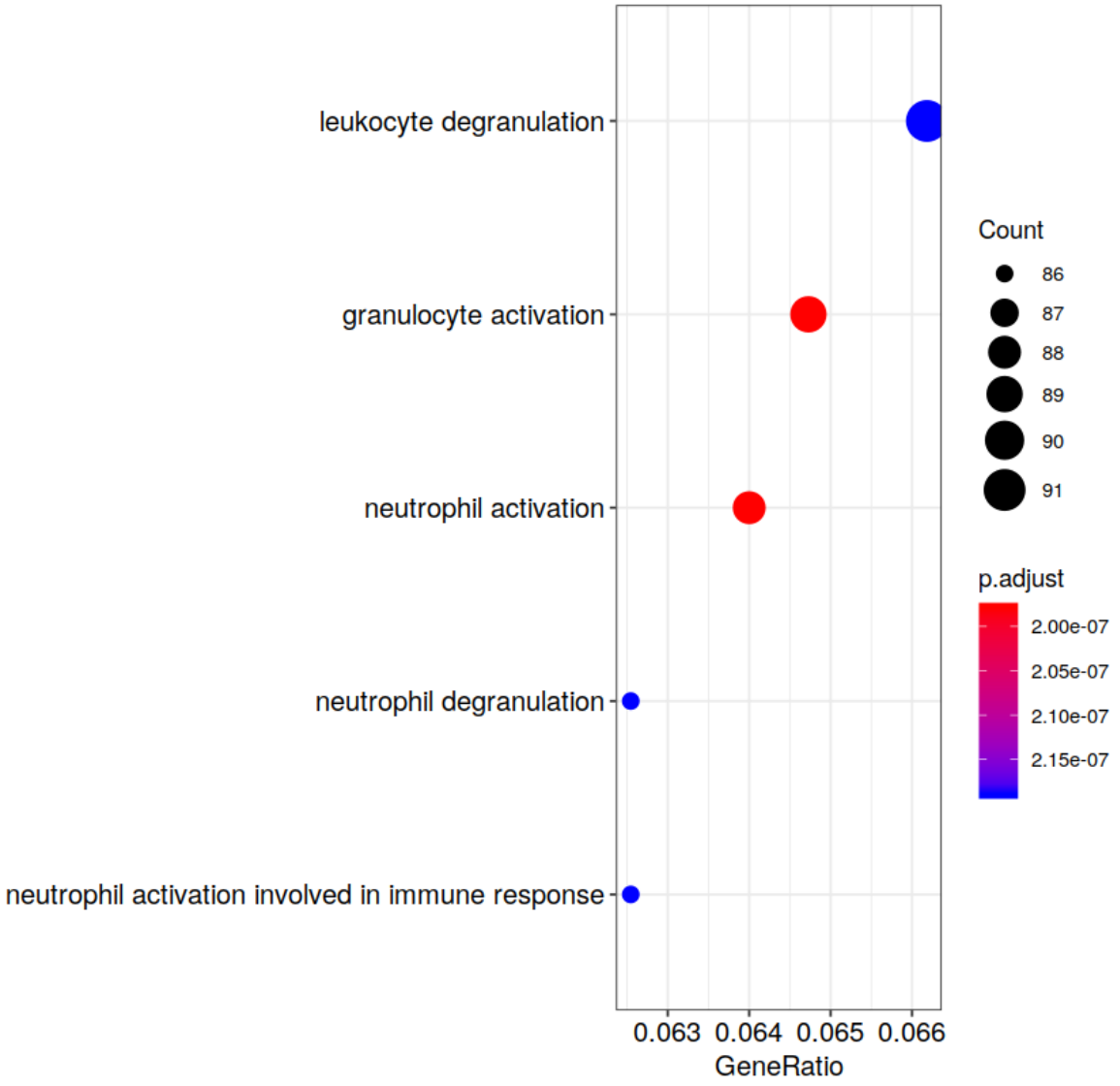
clusterProfiler has a variety of options for viewing the over-represented GO terms. We will explore the dotplot, enrichment plot, and the category netplot

The **dotplot** shows the number of genes associated with the first 5 terms and the p-adjused values for these terms (color).

In [525]:

```
# Dotplot
dotplot(ego, showCategory=5)
```

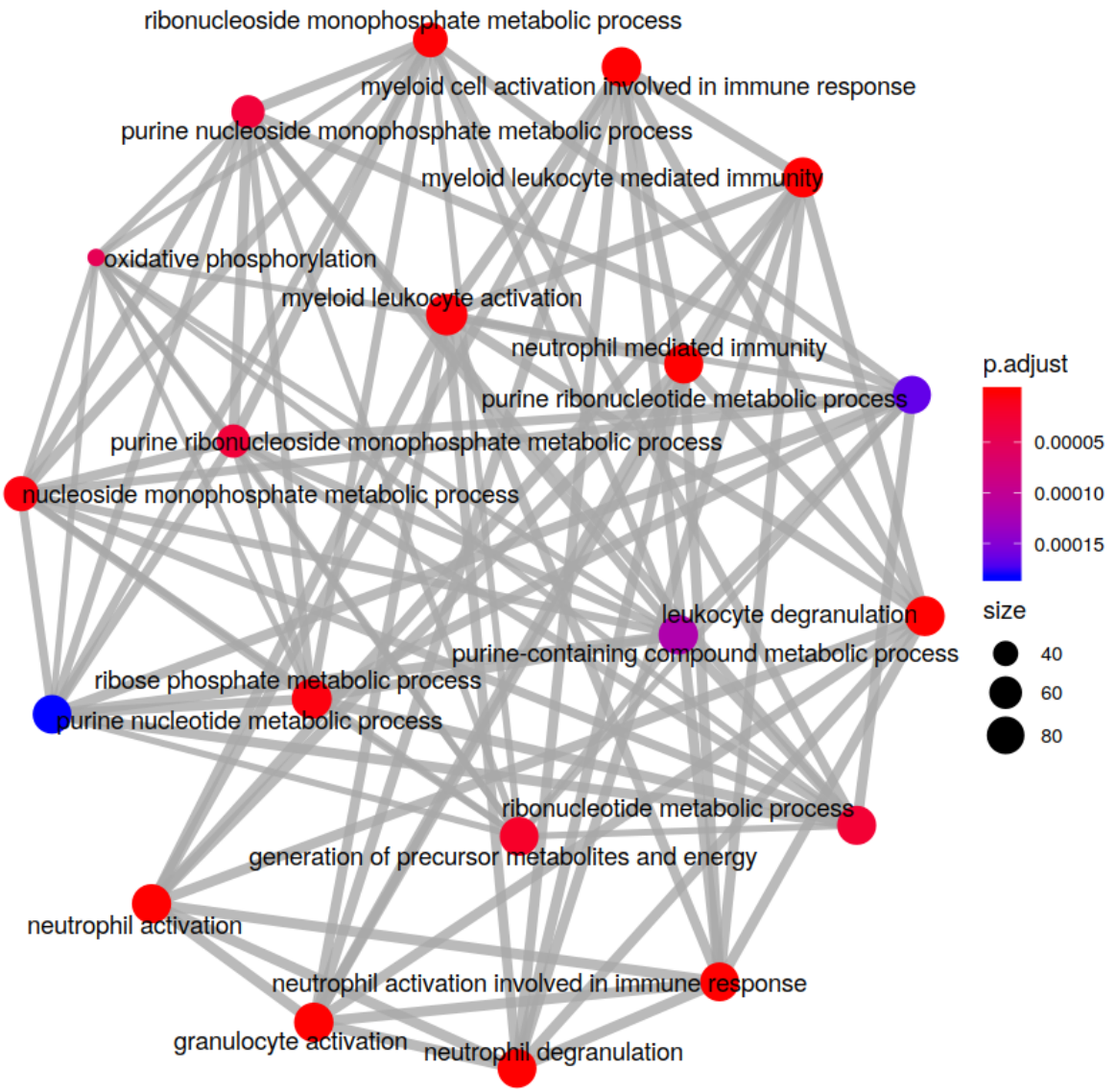
```
wrong orderBy parameter; set to default `orderBy = "x"`
```



The next plot is the **enrichment GO plot**, which shows the relationship between the top 20 most significant enriched GO terms (padj.), by grouping similar terms together. The color represents the p-values relative to the other displayed terms (brigther red is more significant) and the size of the terms represents the number of genes that are significant from our list.

In [526]:

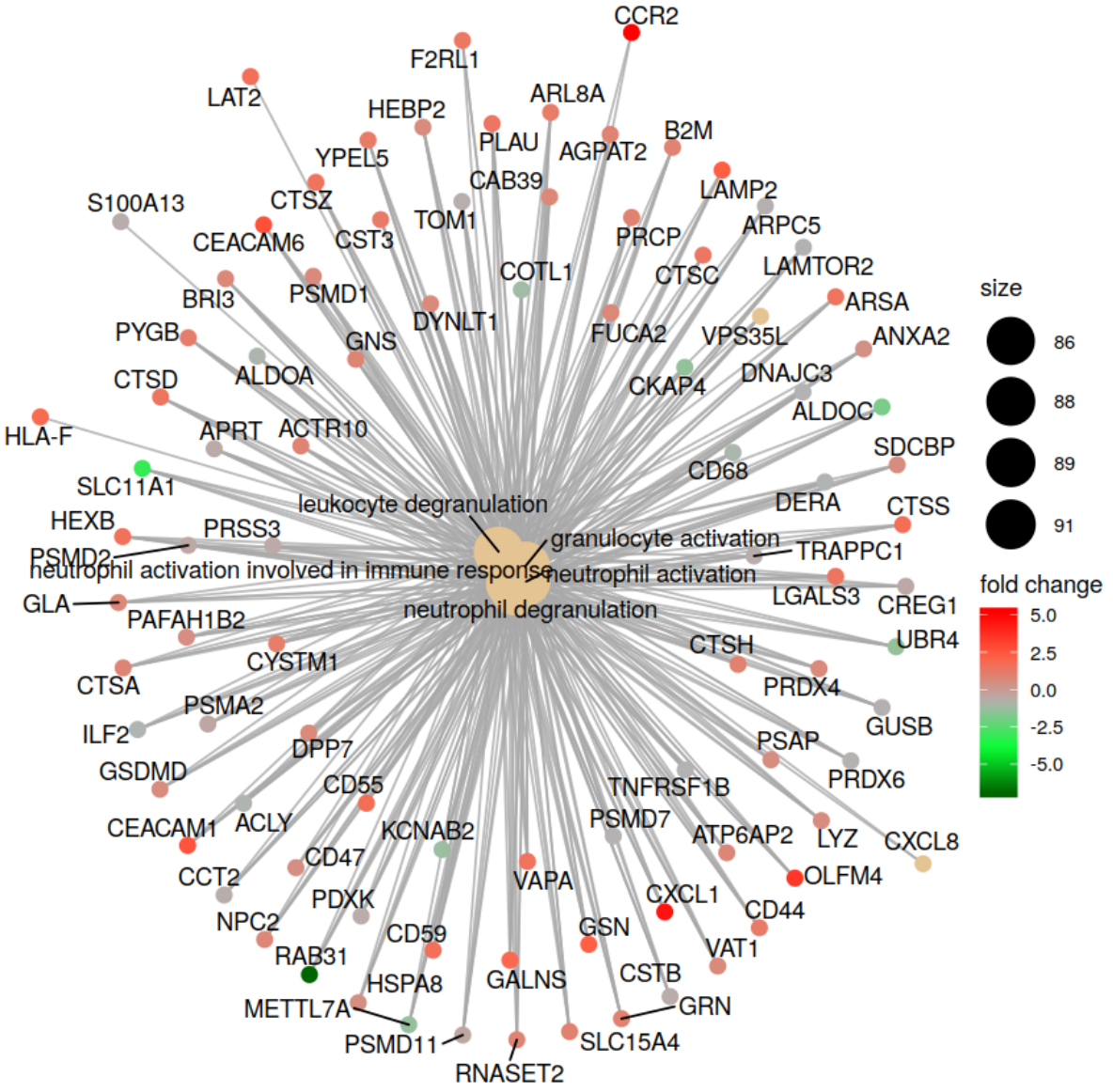
```
# Enrichmap clusters the 20 most significant (by padj) GO terms to visuali
emapplot(ego, showCategory = 20)
```



Finally, the **category netplot** shows the relationships between the genes associated with the top five most significant GO terms and the fold changes of significant genes associated with these terms (color). The size of the GO terms reflects the pvalues of the terms, with the more signigicant terms being larger. This plot is particularly useful for hypothesis generation in identifying genes that may be important to several of the most affected processes.

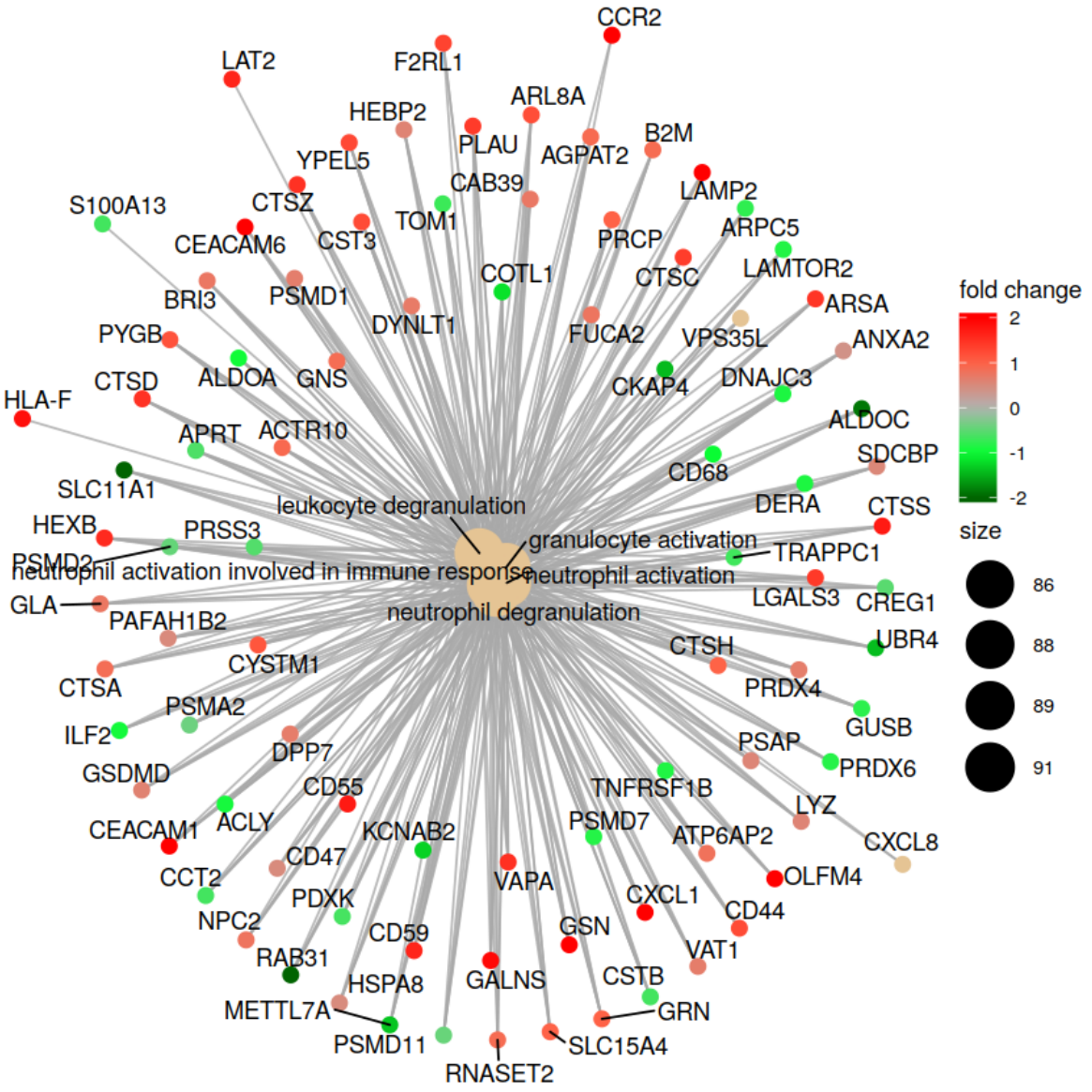
In [527]:

```
# To color genes by log2 fold changes, we need to extract the log2 fold ch
OE_foldchanges <- sigOE$log2FoldChange

names(OE_foldchanges) <- sigOE$gene

# Cnetplot details the genes associated with one or more terms - by defaul
cnetplot(ego,
categorySize="pvalue",
showCategory = 5,
foldChange=OE_foldchanges,
vertex.label.font=6)

```



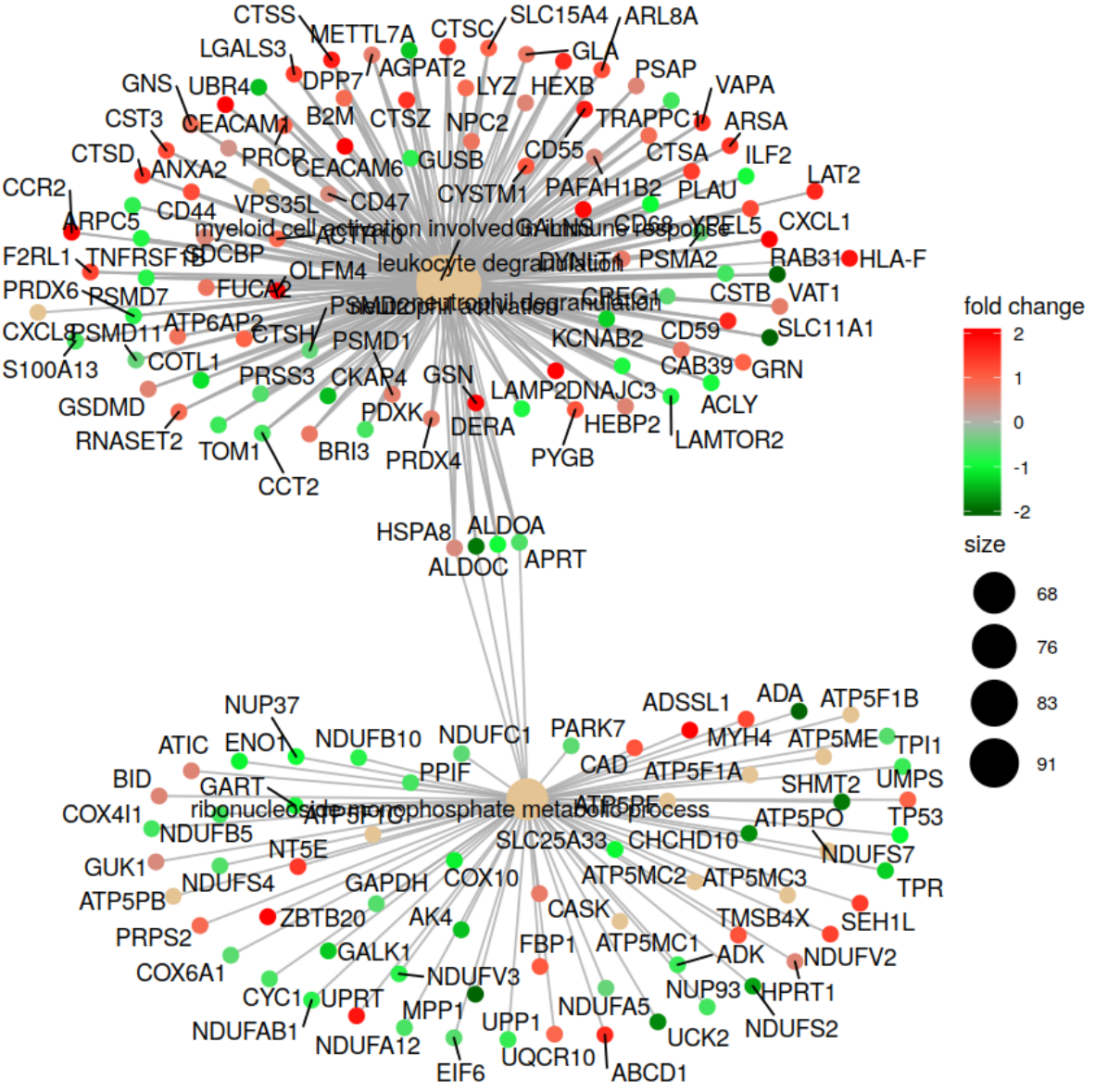
In [528]:

```
# If some of the high fold changes are getting drowned out due to a large
OE_foldchanges <- ifelse(OE_foldchanges > 2, 2, OE_foldchanges)
OE_foldchanges <- ifelse(OE_foldchanges < -2, -2, OE_foldchanges)
cnetplot(ego,
categorySize="pvalue",
showCategory = 5,
foldChange=OE_foldchanges,
vertex.label.font=6)

```



In [529]:

```
# If you are interested in significant processes that are not among the to
# you can subset your ego dataset to only display these processes
# Subsetting the ego results without overwriting original `ego` variable
ego2 <- ego
ego2@result <- ego@result[c(1,3,4,8,9),]
# Plotting terms of interest
cnetplot(ego2,
categorySize="pvalue",
foldChange=OE_foldchanges,
showCategory = 5,
vertex.label.font=6)

```

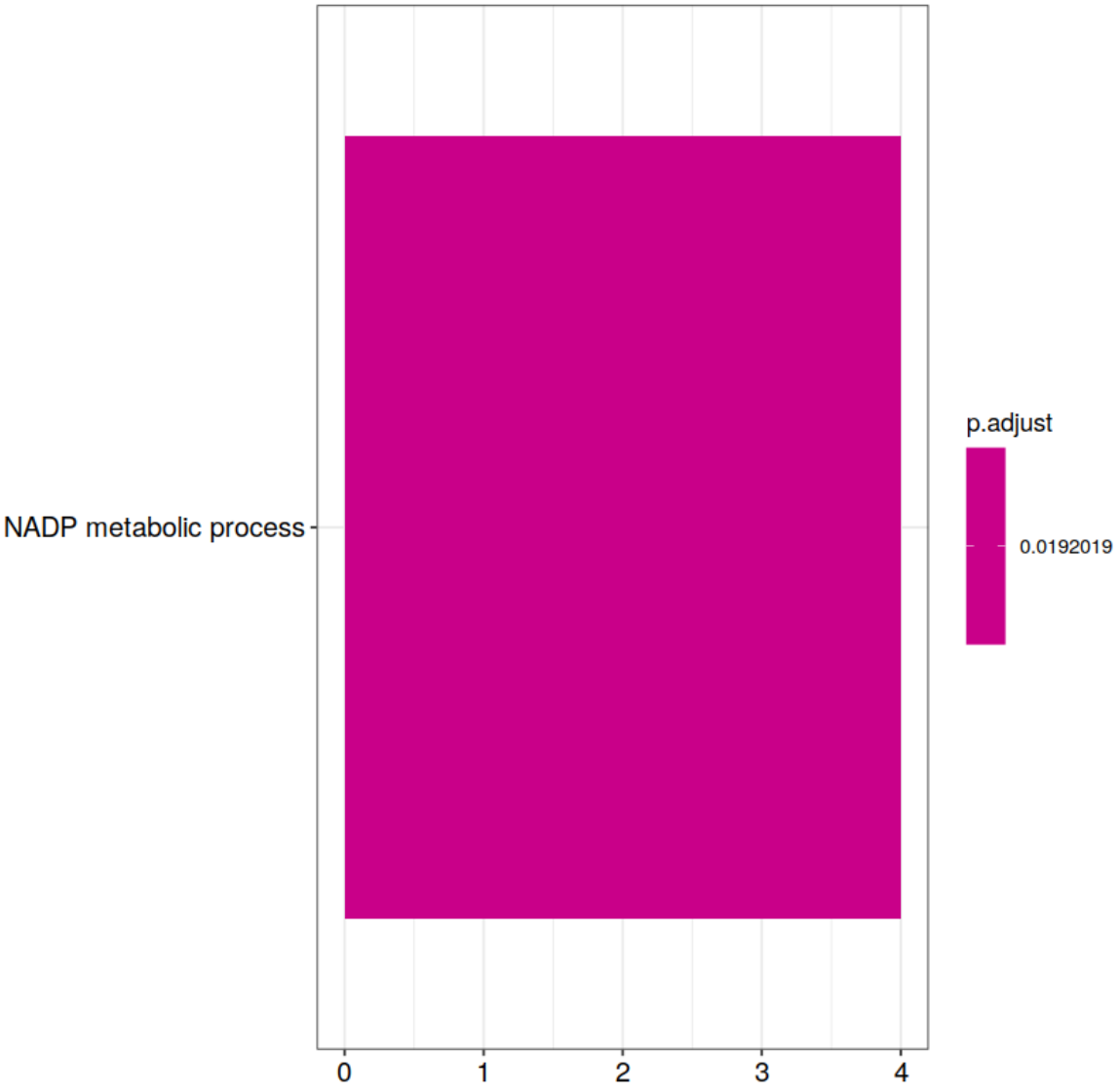


In [530]:

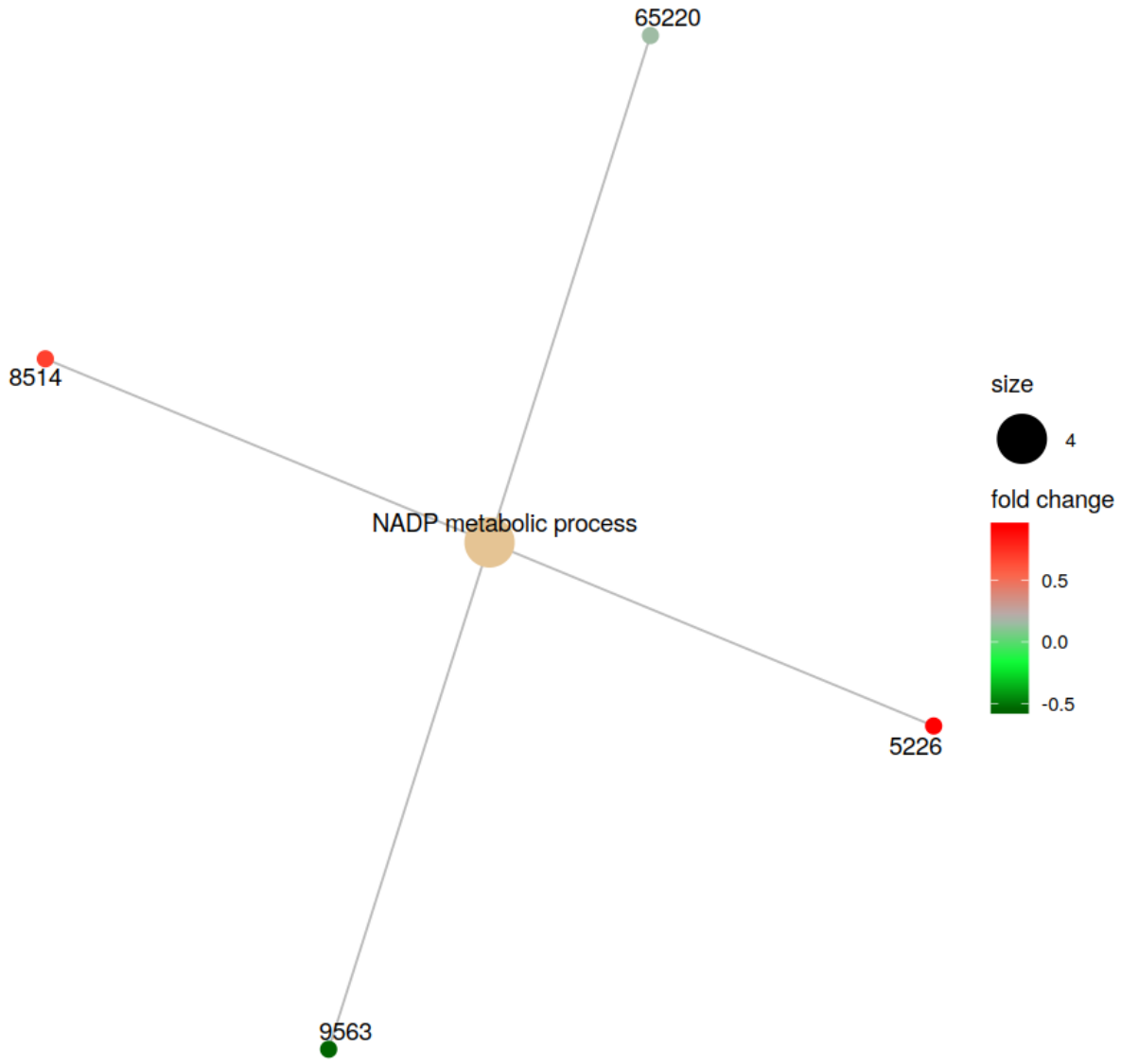
```r
# Disease Ontology Semantic and Enrichment analysis
data(geneList, package = "DOSE")
de <- names(genes_for_cp)[1:100]
ego <- enrichGO(de, 'org.Hs.eg.db', ont = 'BP', pvalueCutoff=0.05)
ego
```

```
#
# over-representation test
#
#...@organism    Homo sapiens
#...@ontology    BP
#...@keytype     ENTREZID
#...@gene        chr [1:100] "653635" "729737" "100288069" "79854" "26155" "3
39451" "84069" ...
#...pvalues adjusted by 'BH' with cutoff <0.05
#...1 enriched terms found
'data.frame':   1 obs. of  9 variables:
 $ ID         : chr "GO:0006739"
 $ Description: chr "NADP metabolic process"
 $ GeneRatio  : chr "4/82"
 $ BgRatio    : chr "30/17913"
 $ pvalue     : num 1.02e-05
 $ p.adjust   : num 0.0192
 $ qvalue     : num 0.0181
 $ geneID     : chr "65220/8514/9563/5226"
 $ Count      : int 4
#...Citation
  Guangchuang Yu, Li-Gen Wang, Yanyan Han and Qing-Yu He.
  clusterProfiler: an R package for comparing biological themes among
  gene clusters. OMICS: A Journal of Integrative Biology
  2012, 16(5):284-287
```

In [531]:
```
1 barplot(ego, showCategory=8)
```



In [532]:
```
1 cnetplot(ego, categorySize="pvalue", foldChange=geneList)
```



Functional class scoring tools

Functional class scoring (FCS) tools, such as GSEA most often use the gene-level statistics or log2 fold changes for all genes from the differential expression results, then look to see whether gene sets for particular biological pathways are enriched among the large positive or negative fold changes.

The hypothesis of FCS methods is that large changes in individual genes can have significant effects on pathways (and will be detected via ORA), weaker but coordinated changes in sets of functionally related genes (i.e., pathways) can also have significant effects. Thus, rather than setting an arbitrary threshold to identify 'significant genes', **all genes are considered** in the analysis. The gene-level statistics from the dataset are aggregated to generate a single pathway-level statistics and statistical significance of each pathway is reported. This type of analysis can be particularly helpful if the differential expression analysis only outputs a small list of significant DE genes.

GO Gene Set Enrichment Analysis

A common approach in analyzing gene expression profiles was identifying differential expressed genes that are deemed interesting. The enrichment analysis above are based on these differential expressed genes. This approach will find gene where the difference is large, but will not detect a situation where the difference is small, but evidenced in coordinated way in a set of related genes. Gene Set Enrichment Analysis (GSEA) addresses this limitation. All genes can be used in GSEA; GSEA aggregates the per

gene statistics across gene within a gene set, therefor making it possible to detect situations where all genes in a predefined set change in a small but coordinated way. Since it is likely that many releveant phenotypic differences are manifested by small but consistent changes in a set of genes.

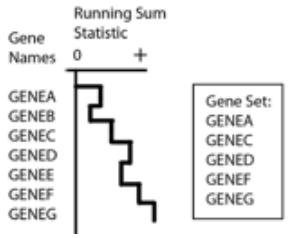
Genes are ranked based on their phenotypes. Give a priori defined set of genes *S* (e.g. genes sharing the same GO or KEGG category), the goal of GSEA is to determine whether the members of genes *S* are randomly distributed throughout the ranked gene list *L* or or primarily found at the top or bottom.

There are three key elements of the GSEA method:

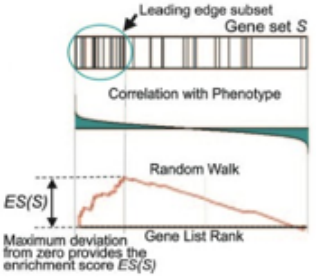
- **Calculation of an Enrichment Score.** The enrichment score (ES) represent the degree to which a set *S* is over-represented at the top or bottom of the ranked list *L* . The score ist calculated by walking down the list *L*, increasing a running-sum statistic when we encounter a gene in S and decreasing when it is not. The magnitude of the increment depends on the gene statistics (e.g. corrrelation of the gene with phenotype). The ES is the maximum deviation from zero encountered in the random walk; its statistical analysis and visualisation of functional profiles for gene and clusters corresponds to a weighted Kolmogorov-Smirnov-like statistic.
- **Estimation of Significance Level of *ES*** The p-value of the *ES* is calculated using permutation test. Specifically, we permute the gene labels of the gene list *L* and recompute the *ES* of the gene set for the permutated data, which generate a null distribution for the *ES*. The p-value of the observed *ES* is then calculated relative to this null distribution.
- **Adjustment for Multiple Hypothesis Testing**. When the entire *GO* or *KEGG* gene sets is evaluated, clusterProfiler adjust the estimated significance level to account for multiple hypothesis testing and also q-values were calculated for FDR control.

Algorithm

GSEA first ranks the genes based on a measure of each gene's differential expression with respect to the two phenotypes (for example, tumor vs. normal) or correlation with a continous phenotype. Then the entire ranked list is used to assess how the genes of each gene set are distributed across the ranked list. To do this, GSEA walks down the ranked list of genes, increasing a running-sum statistic when a gene belongs to the set and decreasing it when the gene does not.



The enrichment score (ES) is the maximum deviation from zero encountered during that walk. The ES reflects the degree to which the genes in a gene set are overrepresented at the top or bottom of the entire ranked list of genes. A set that is not enriched will have its genes spread more or less unifomly through the ranked list. An enriched set, on the other hand, will have a larger portion of its genes at one or the other end of the ranked list. The extend of enrichment is captured mathematically as the ES statistic.



Next, GSEA estimates the statistical significance of the ES by a permutation test. To do this, GSEA creates a version of the data set with phenotype labels randomly scrambled, produces the corresponding ranked list, and recomputes the ES of the gene set for this permuted data set. GSEA repeats this many

times (1000 is the default) and produces an empirical null distribution of ES scores. Alternatively, permutations may be generated by creating "random" gene sets (genes randomly selected from those in the expression dataset) of equal size to the gene set under analysis.

The nominal p-value estimates the statistical significance of a single gene set's enrichment score, based on the permutation-generated null distribution. The nominal p-value is the probability under the null distribution of obtaining an ES value that is as strong or stronger that that observed for your experiment under the permutation-generated null distribution.

Typically, GSEA is run with a large number of gene sets. For example, the MSigDB collection and subcollections each contain hundreds to thousands of gene sets. This has implications when comparing enrichment results for the many sets:

The ES must be adjusted to account for differences in the gene set sizes and in corrleations between gene sets and the expression data set. The resulting normalized enrichment scores (NES) allow you to compare the analysis results across gene sets.

The nominal p-value need to be corrected to adjust for multiple hypothesis testing. For a large number of sets (rule of thumb: more than 30), it's recommended paying attention to the False Discovery Rate (FDR) q-values: consider a set significantly enriched if its NES has an FDR q-value below 0.25.

In [465]:
```
1 head(res_ids)
```

A tibble: 6 × 18

gene	baseMean	log2FoldChange	lfcSE	pvalue	padj	emsembl	entrez.x	th
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<chr>	
CHST11	689.9265	-2.397997	0.1688200	5.681973e-47	6.945643e-43	ENSG00000171310	50515	
CXCR3	421.7270	-2.701475	0.1972148	6.549712e-44	4.003184e-40	ENSG00000186810	2833	
ALDH2	398.5219	-4.231761	0.3214003	8.787316e-41	3.580538e-37	ENSG00000111275	217	
TESC	1057.1753	2.623294	0.2023895	1.223010e-39	3.737518e-36	ENSG00000088992	54997	
TMEM171	209.7161	5.312692	0.4224382	1.001256e-37	2.447872e-34	ENSG00000157111	134285	
MYL12B	11666.1236	1.821189	0.1455522	5.588482e-37	1.138560e-33	ENSG00000118680	103910	

In [466]:

```
## Remove any NA values
res_entrez <- dplyr::filter(res_ids, entrez.x != "NA")
head(res_entrez)
```

A tibble: 6 × 18

gene	baseMean	log2FoldChange	lfcSE	pvalue	padj	emsembl	entrez.x	th
<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>	<chr>	
CHST11	689.9265	-2.397997	0.1688200	5.681973e-47	6.945643e-43	ENSG00000171310	50515	
CXCR3	421.7270	-2.701475	0.1972148	6.549712e-44	4.003184e-40	ENSG00000186810	2833	
ALDH2	398.5219	-4.231761	0.3214003	8.787316e-41	3.580538e-37	ENSG00000111275	217	
TESC	1057.1753	2.623294	0.2023895	1.223010e-39	3.737518e-36	ENSG00000088992	54997	
TMEM171	209.7161	5.312692	0.4224382	1.001256e-37	2.447872e-34	ENSG00000157111	134285	
MYL12B	11666.1236	1.821189	0.1455522	5.588482e-37	1.138560e-33	ENSG00000118680	103910	

In [467]:

```
# Remove any Entrez duplicates
res_entrez <- res_entrez[which(duplicated(res_entrez$entrez.x) == F), ]

```

In [515]:

```
# Extract the foldchanges
foldchanges <- res_entrez$log2FoldChange

# Name each fold change with the corresponding Entrez ID
names(foldchanges) <- res_entrez$entrez.x

## Sort fold changes in decreasing order
foldchanges <- sort(foldchanges, decreasing = TRUE)

head(foldchanges)

```

221458	10.6341541825349
2766	10.0543426417089
57161	9.24522878178835
91133	8.34200718472065
3861	8.06151178279155
1307	7.81595228715757

In [496]:

```r
# GSEA using gene sets from KEGG pathways
gseaKEGG <- gseKEGG(geneList = foldchanges, # ordered named vector of fold
    organism = "hsa", # supported organisms
    nPerm = 1000, # default number permutations
    minGSSize = 120, # minimum gene set size (# genes in set) - change to
    pvalueCutoff = 0.07, # padj cutoff value
    verbose = FALSE)

# Extract the GSEA results
gseaKEGG_results <- gseaKEGG@result
head(as.data.frame(gseaKEGG))
```

```
Warning message in fgsea(pathways = geneSets, stats = geneList, nperm = nPer
m, minSize = minGSSize, :
"There are ties in the preranked stats (7.66% of the list).
The order of those tied genes will be arbitrary, which may produce unexpected
results."
```

A data.frame: 5 × 11

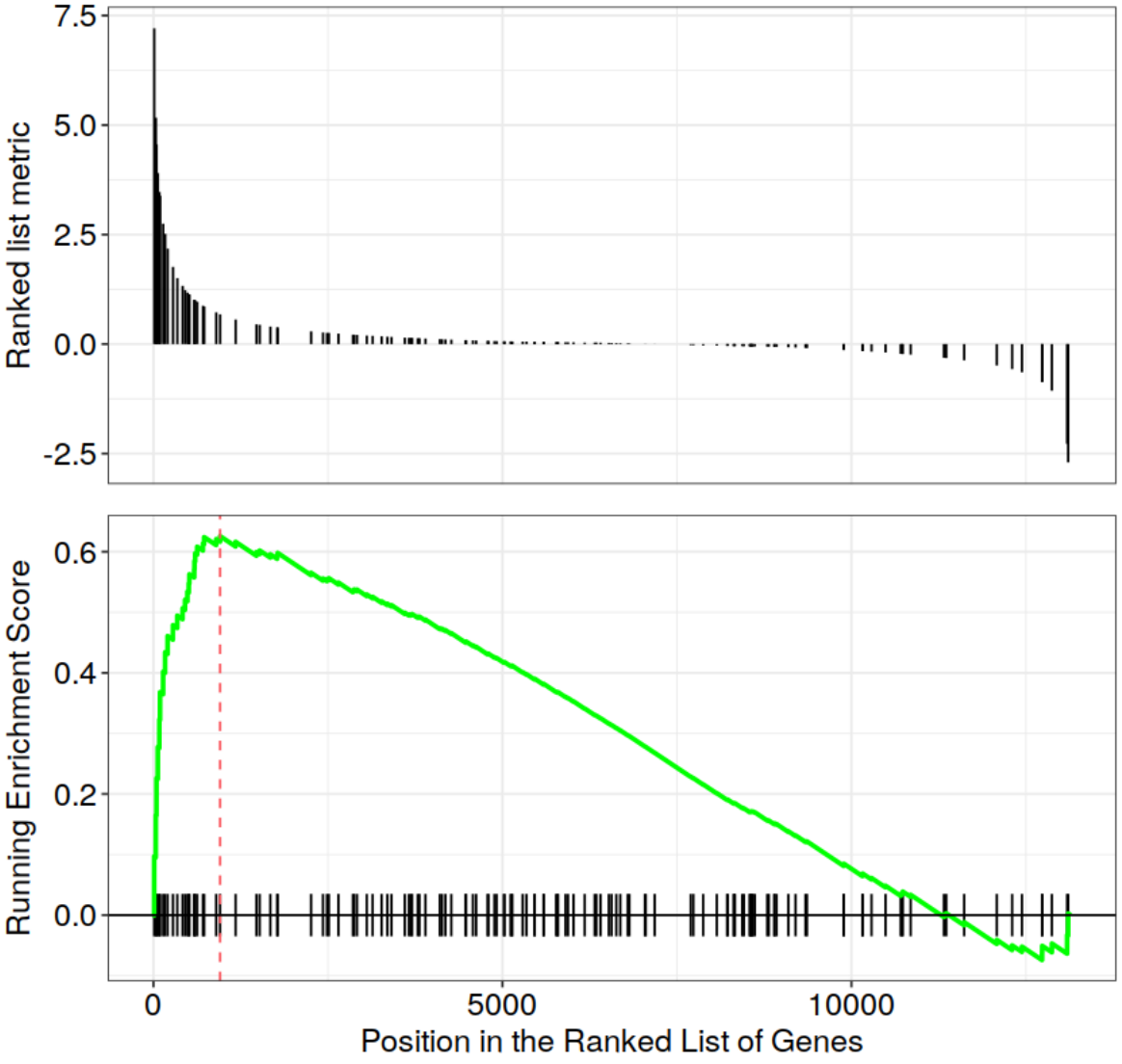
	ID	Description	setSize	enrichmentScore	NES	pvalue	p.adjust	qva
	<chr>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<
hsa04060	hsa04060	Cytokine-cytokine receptor interaction	140	0.6258059	1.754113	0.001218027	0.06455542	0.0435
hsa03010	hsa03010	Ribosome	129	-0.7063218	-2.362904	0.005128205	0.06666667	0.0450
hsa03013	hsa03013	RNA transport	139	-0.4572133	-1.538354	0.005405405	0.06666667	0.0450
hsa05010	hsa05010	Alzheimer disease	141	-0.4405963	-1.479332	0.005714286	0.06666667	0.0450
hsa04714	hsa04714	Thermogenesis	172	-0.4682229	-1.650324	0.006289308	0.06666667	0.0450

In [497]:

```r
# write results to csv file
write.csv(gseaKEGG_results, "gsea_kegg.csv", quote=F)
```

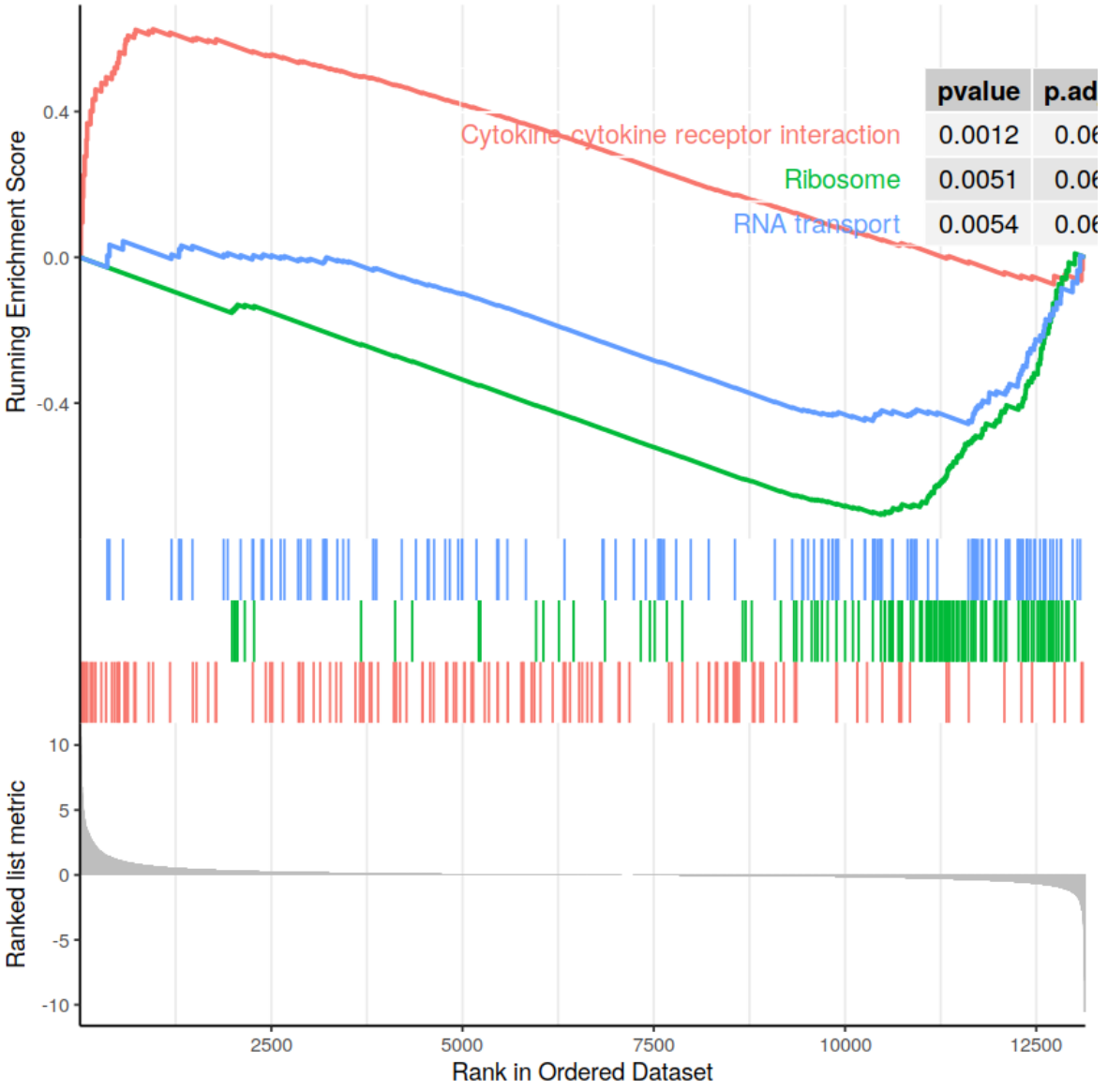
In [500]:

```
# Plot the GSEA plot for a single enriched pathway, `hsa04060`
gseaplot(gseaKEGG, geneSetID = 'hsa04060')

```



In [502]:

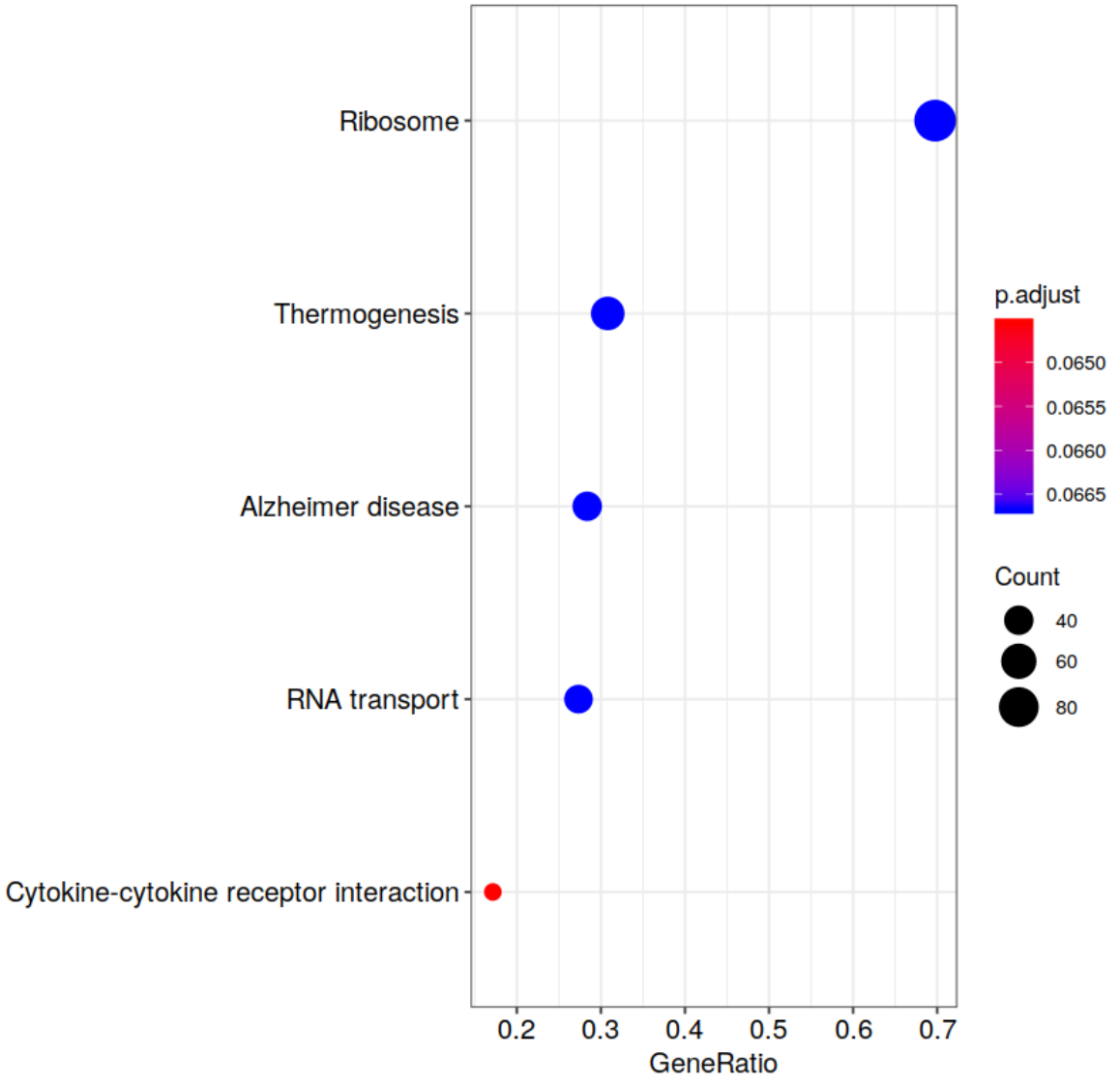
```
# another method to plot GSEA results which also allows multiple gene sets
gseaplot2(gseaKEGG, geneSetID = 1:3, pvalue_table = TRUE)
```



In [509]:

```
# dot plot depict the enrichment scores and gene counts per gene set (for

dotplot(gseaKEGG)
```

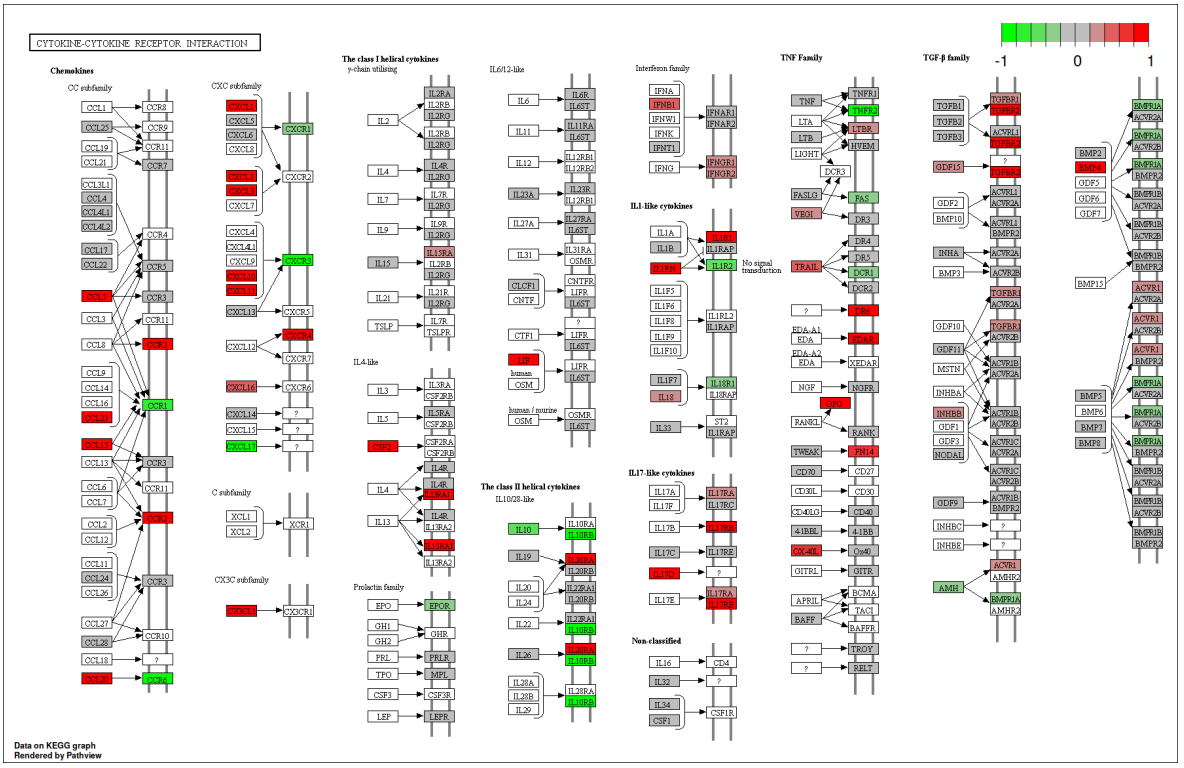
```
wrong orderBy parameter; set to default `orderBy = "x"`
```



In [503]:

```
# Use the Pathview R package to integrate the KEGG pathway data
# Output images for a single significant KEGG pathway
detach("package:dplyr", unload=TRUE) # first unload dplyr to avoid conflic
pathview(gene.data = foldchanges,
pathway.id = "hsa04060",
species = "hsa",
limit = list(gene = 2, # value gives the max/min limit for foldchanges
cpd = 1))
```

```
Warning message:
"'dplyr' namespace cannot be unloaded:
  namespace 'dplyr' is imported by 'ggraph', 'dbplyr', 'broom', 'AnnotationHu
b', 'BiocFileCache', 'DEGreport', 'europepmc' so cannot be unloaded"Info: Dow
nloading xml files for hsa04060, 1/1 pathways..
Info: Downloading png files for hsa04060, 1/1 pathways..
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa04060.pathview.png
```



In [504]:

```
## Output images for all significant KEGG pathways
get_kegg_plots <- function(x) {
pathview(gene.data = foldchanges, pathway.id = gseaKEGG_results$ID[x], spe
limit = list(gene = 2, cpd = 1))
}
purrr::map(1:length(gseaKEGG_results$ID), get_kegg_plots)

```

```
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa04060.pathview.png
Info: Downloading xml files for hsa03010, 1/1 pathways..
Info: Downloading png files for hsa03010, 1/1 pathways..
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa03010.pathview.png
Info: Downloading xml files for hsa03013, 1/1 pathways..
Info: Downloading png files for hsa03013, 1/1 pathways..
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa03013.pathview.png
Info: Downloading xml files for hsa05010, 1/1 pathways..
Info: Downloading png files for hsa05010, 1/1 pathways..
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa05010.pathview.png
Info: Downloading xml files for hsa04714, 1/1 pathways
```

In [518]:

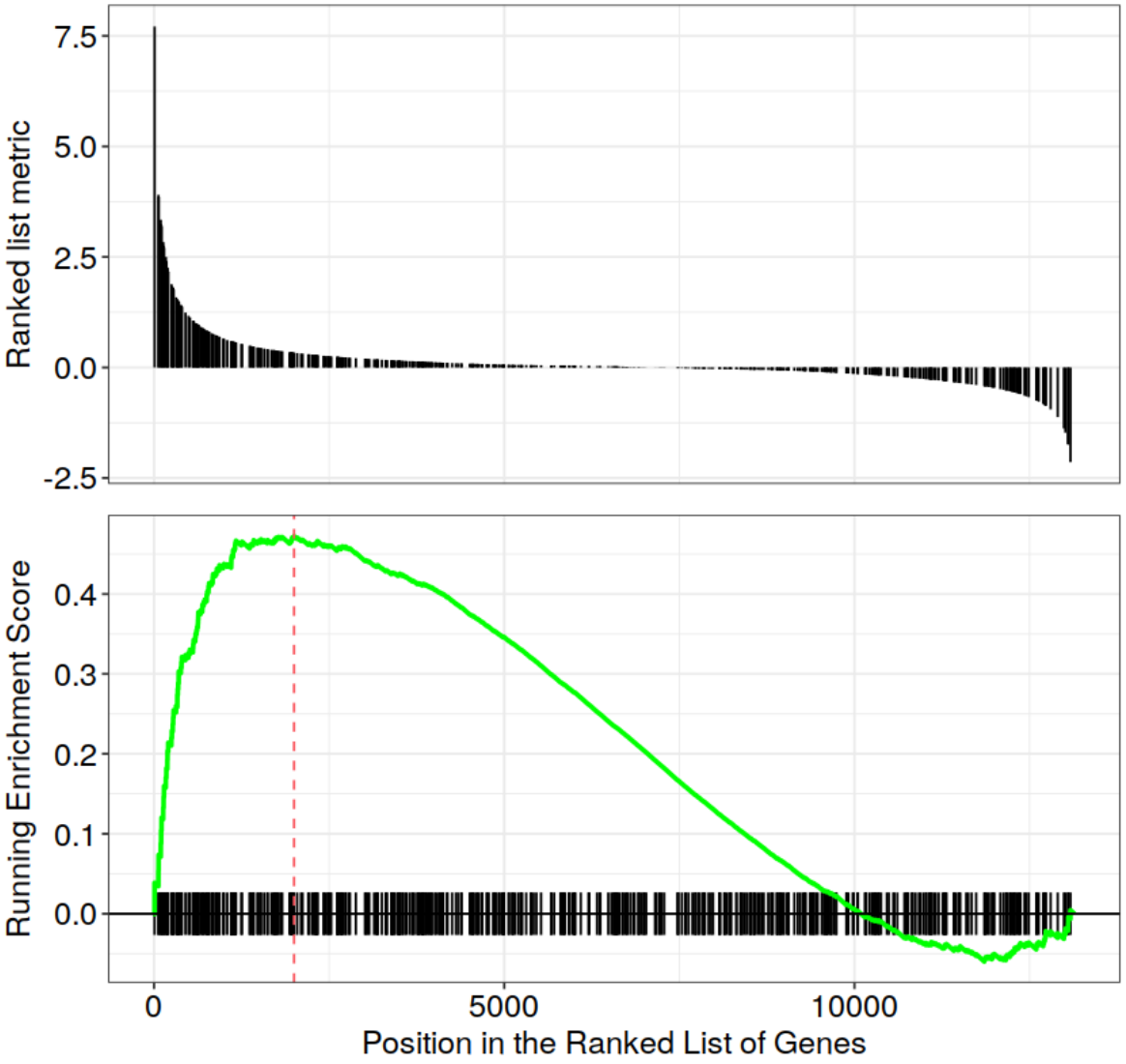
```r
# GSEA using gene sets associated with BP Gene Ontology terms
gseaGO <- gseGO(geneList = foldchanges,
    OrgDb = org.Hs.eg.db,
    ont = 'BP',
    nPerm = 1000,
    minGSSize = 20,
    pvalueCutoff = 0.07,
    verbose = FALSE)
gseaGO_results <- gseaGO@result
head(as.data.frame(gseaGO_results))
```

```
Warning message in fgsea(pathways = geneSets, stats = geneList, nperm = nPer
m, minSize = minGSSize, :
"There are ties in the preranked stats (7.66% of the list).
The order of those tied genes will be arbitrary, which may produce unexpected
results."
```

A data.frame: 6 × 11

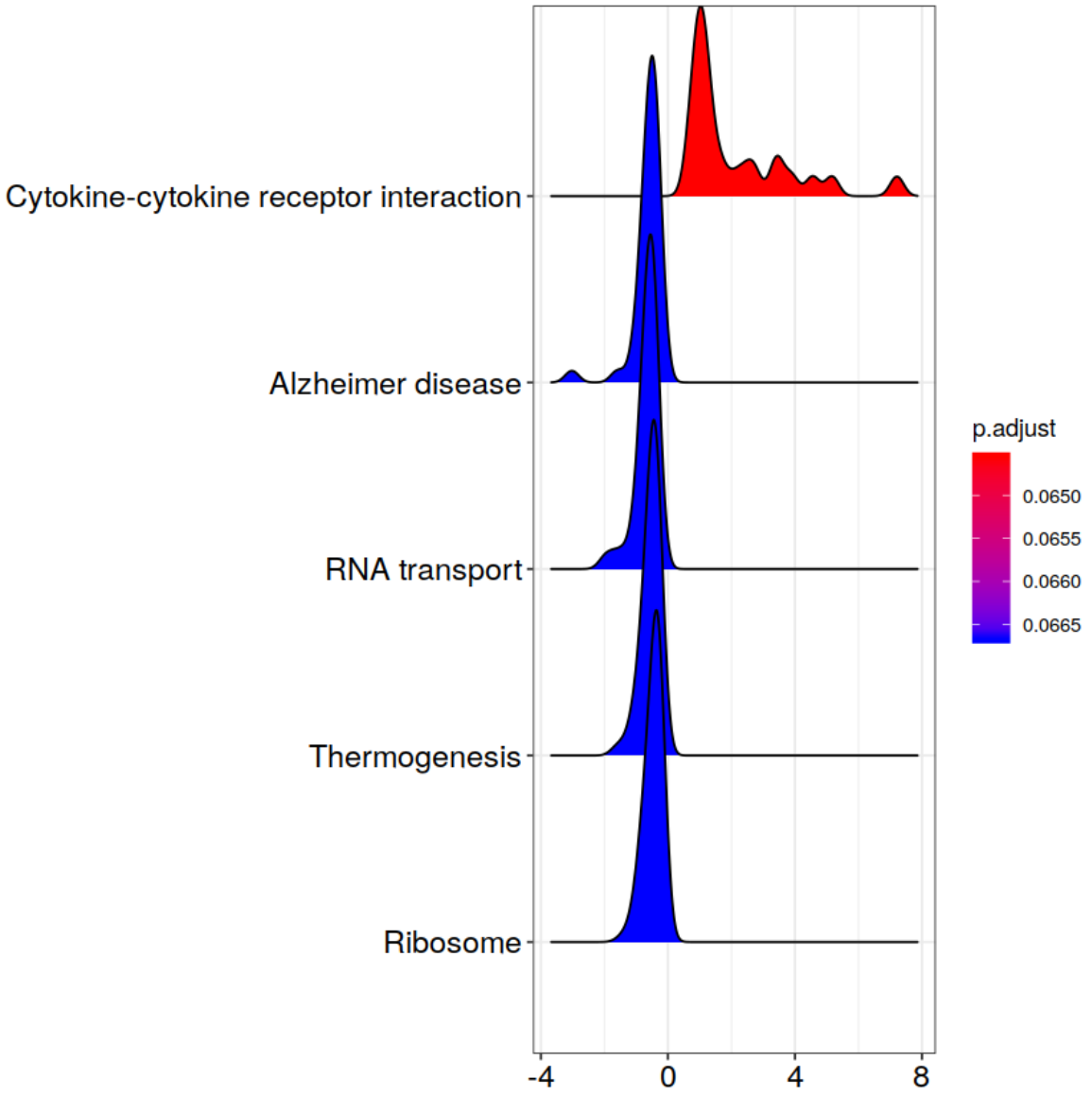
	ID	Description	setSize	enrichmentScore	NES	pvalue	p.adjust	
	<chr>	<chr>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	
GO:0043065	GO:0043065	positive regulation of apoptotic process	500	0.4723397	1.450798	0.001034126	0.05060415	0.
GO:0051046	GO:0051046	regulation of secretion	500	0.4819020	1.480169	0.001034126	0.05060415	0.
GO:0030155	GO:0030155	regulation of cell adhesion	470	0.5499016	1.684149	0.001037344	0.05060415	0.
GO:0031344	GO:0031344	regulation of cell projection organization	471	0.4626606	1.416711	0.001038422	0.05060415	0.
GO:0032101	GO:0032101	regulation of response to external stimulus	494	0.5426989	1.665935	0.001038422	0.05060415	0.
GO:0000904	GO:0000904	cell morphogenesis involved in differentiation	469	0.4655308	1.424179	0.001040583	0.05060415	0.

In [519]:
```
1 gseaplot(gseaGO, geneSetID = 'GO:0043065')
```



In [510]:
```
1 ridgeplot(gseaKEGG)
```

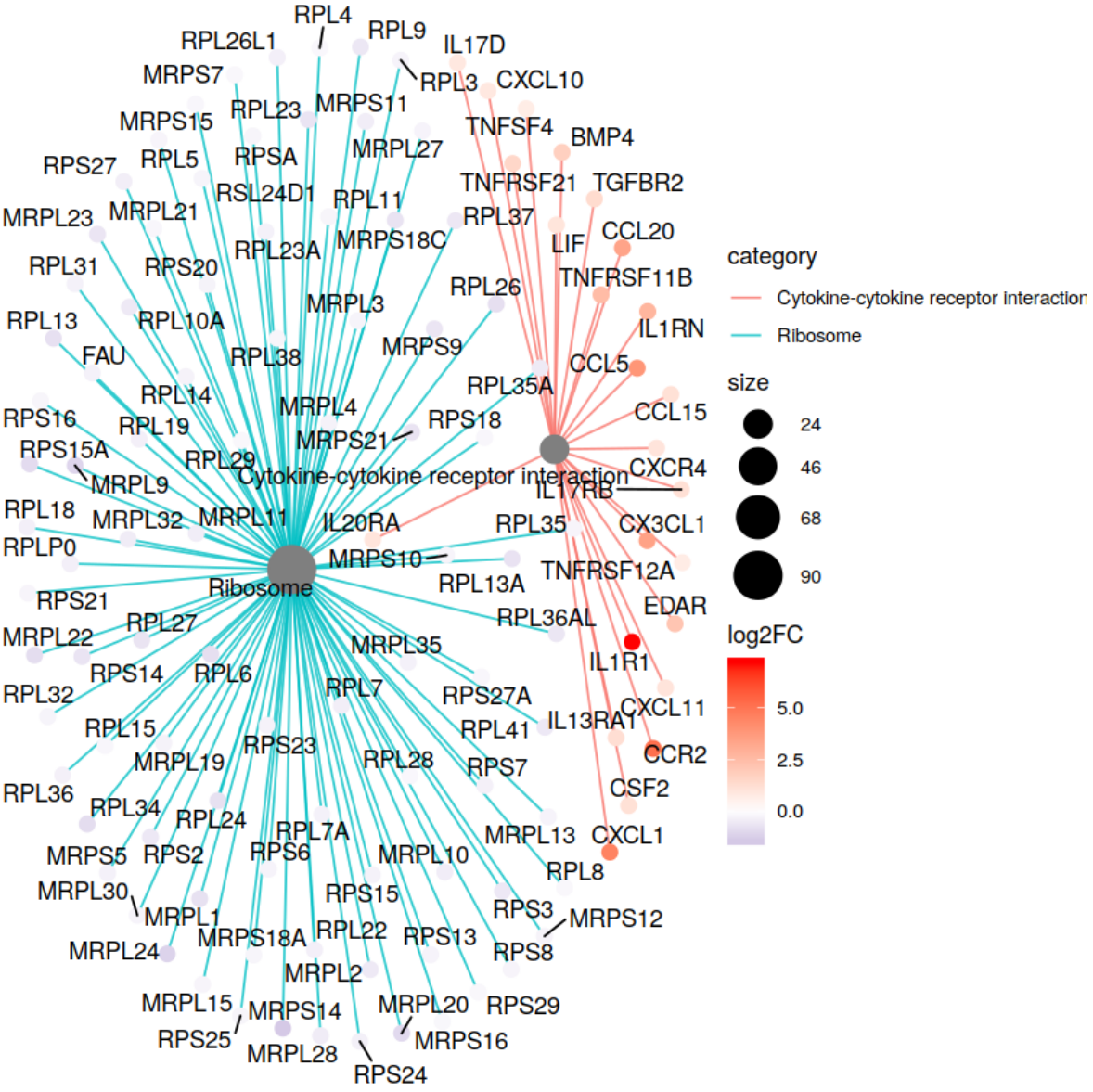
```
Picking joint bandwidth of 0.216
```



In [511]:

```
## convert gene ID to Symbol
edox <- setReadable(gseaKEGG, 'org.Hs.eg.db', 'ENTREZID')
cnetplot(edox, showCategory = 2, colorEdge = TRUE, foldChange = foldchange
scale_colour_gradient2(name = "log2FC", low = "navyblue", high = "red", mi
```

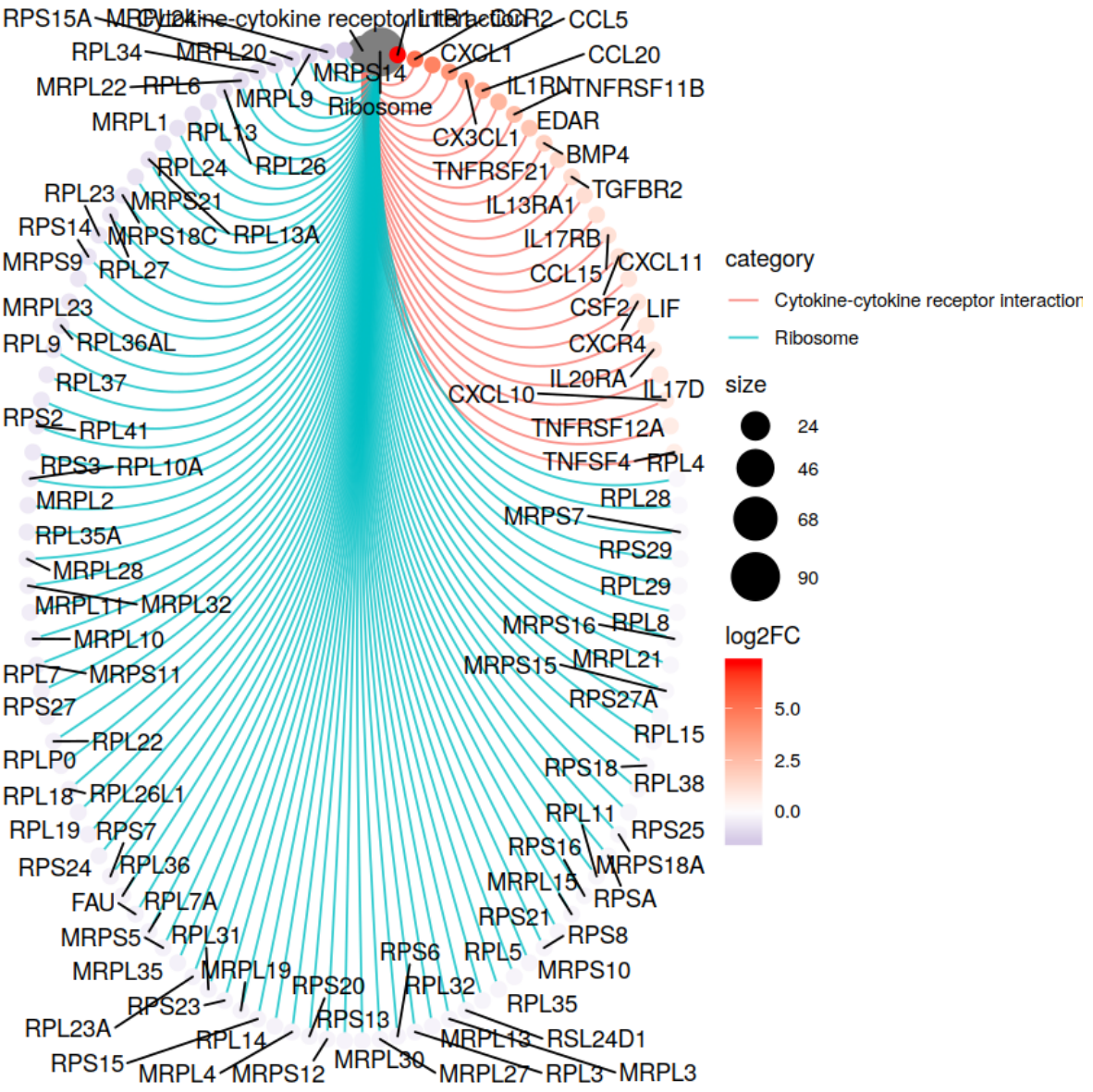
```
Scale for 'colour' is already present. Adding another scale for 'colour',
which will replace the existing scale.
```



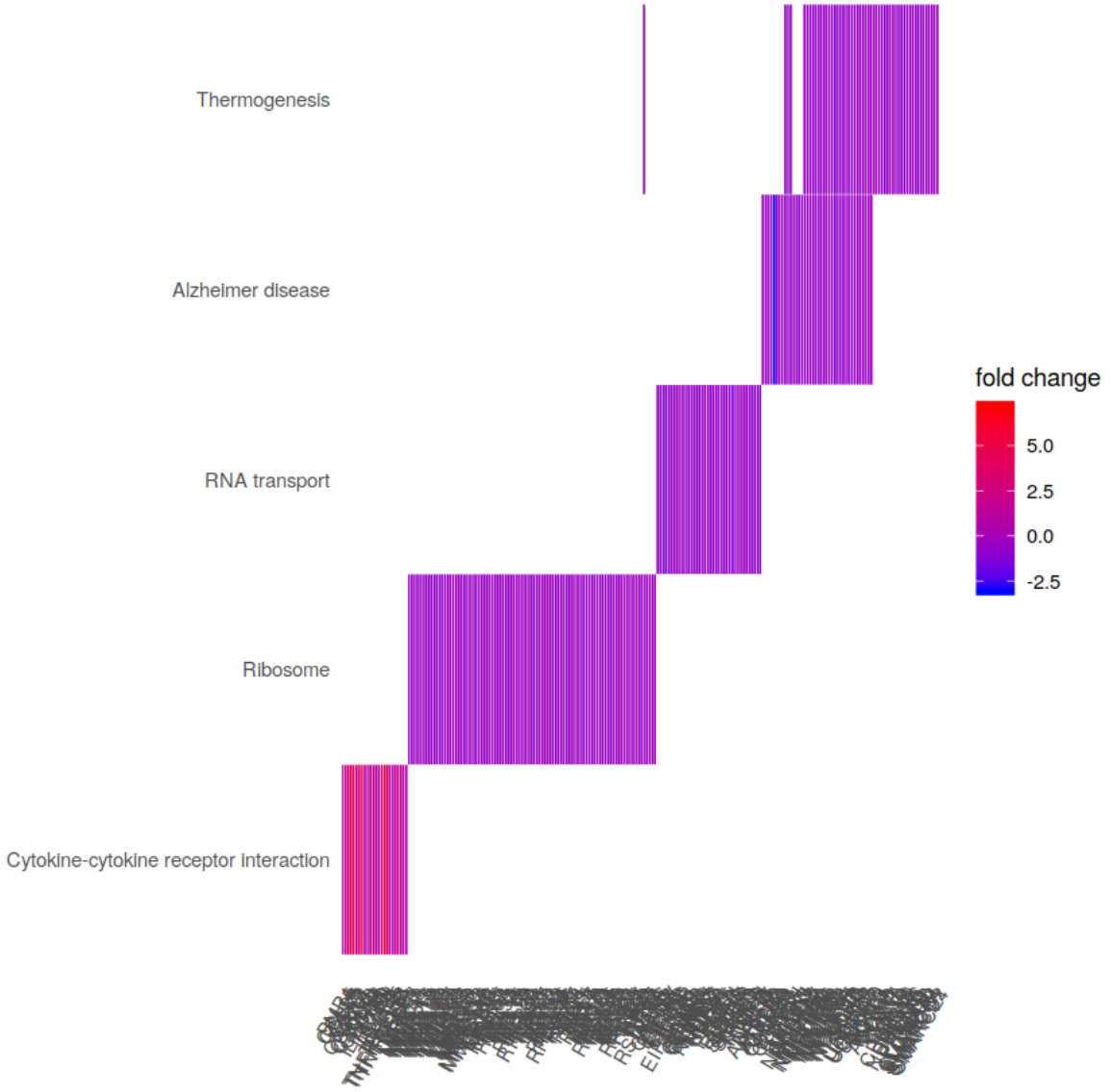
In [512]:

```
cnetplot(edox, showCategory = 2, colorEdge = TRUE,circular = TRUE, foldCha
scale_colour_gradient2(name = "log2FC", low = "navyblue", high = "red", mi
```

```
Scale for 'colour' is already present. Adding another scale for 'colour',
which will replace the existing scale.
```



In [542]:
```
1 heatplot(edox, showCategory = 5, foldChange = foldchanges)
```



In [300]:
```
1 # view KEGG pathway in web browser
2 browseKEGG(kk, 'hsa05200')
```

Pathway topology tools

The last main type functional analysis technique is pathway topology analysis. Pathway topology analysis often takes into account gene interaction information along with the fold changes and adjusted p-values from differential expression analysis to identify dysregulated pathways. Depending on the tool, pathway topology tools explore how genes interact with each other (e.g. activation, inhibition, phosphorylation, ubiquitination, etc.) to determine the pathway-level statistics. Pathway topology-based methods utilize the number and type of interactions between gene product (our DE genes) and other gene products to infer gene function or pathway association.

SPIA

The SPIA (Signaling Pathway Impact Analysis) tool can be used to integrate the lists of differentially expressed genes, their fold changes, and pathway topology to identify affected pathways.

In [536]:

```r
# Significant genes is a vector of fold changes where the names are ENTREZ
background_entrez <- res_entrez$entrez.x
sig_res_entrez <- res_entrez[which(res_entrez$padj < 0.05), ]
sig_entrez <- sig_res_entrez$log2FoldChange
names(sig_entrez) <- sig_res_entrez$entrez.x
head(sig_entrez)

```

50515	-2.39799681429977
2833	-2.7014753966521
217	-4.23176089654189
54997	2.62329355121026
134285	5.31269217295704
103910	1.82118881484595

In [538]:

```
library(SPIA)
spia_result <- spia(de=sig_entrez, all=background_entrez, organism="hsa")
```

```
Loading required package: KEGGgraph

Attaching package: 'KEGGgraph'

The following object is masked from 'package:graphics':

    plot


Done pathway 1 : RNA transport..
Done pathway 2 : RNA degradation..
Done pathway 3 : PPAR signaling pathway..
Done pathway 4 : Fanconi anemia pathway..
Done pathway 5 : MAPK signaling pathway..
Done pathway 6 : ErbB signaling pathway..
Done pathway 7 : Calcium signaling pathway..
Done pathway 8 : Cytokine-cytokine receptor int..
Done pathway 9 : Chemokine signaling pathway..
Done pathway 10 : NF-kappa B signaling pathway..
Done pathway 11 : Phosphatidylinositol signaling..
Done pathway 12 : Neuroactive ligand-receptor in..
Done pathway 13 : Cell cycle..
Done pathway 14 : Oocyte meiosis..
Done pathway 15 : p53 signaling pathway..
Done pathway 16 : Sulfur relay system..
Done pathway 17 : SNARE interactions in vesicula..
Done pathway 18 : Regulation of autophagy..
Done pathway 19 : Protein processing in endoplas..
Done pathway 20 : Lysosome..
Done pathway 21 : mTOR signaling pathway..
Done pathway 22 : Apoptosis..
Done pathway 23 : Vascular smooth muscle contrac..
Done pathway 24 : Wnt signaling pathway..
Done pathway 25 : Dorso-ventral axis formation..
Done pathway 26 : Notch signaling pathway..
Done pathway 27 : Hedgehog signaling pathway..
Done pathway 28 : TGF-beta signaling pathway..
Done pathway 29 : Axon guidance..
Done pathway 30 : VEGF signaling pathway..
Done pathway 31 : Osteoclast differentiation..
Done pathway 32 : Focal adhesion..
Done pathway 33 : ECM-receptor interaction..
Done pathway 34 : Cell adhesion molecules (CAMs)..
Done pathway 35 : Adherens junction..
Done pathway 36 : Tight junction..
Done pathway 37 : Gap junction..
Done pathway 38 : Complement and coagulation cas..
Done pathway 39 : Antigen processing and present..
Done pathway 40 : Toll-like receptor signaling p..
Done pathway 41 : NOD-like receptor signaling pa..
Done pathway 42 : RIG-I-like receptor signaling ..
Done pathway 43 : Cytosolic DNA-sensing pathway..
Done pathway 44 : Jak-STAT signaling pathway..
Done pathway 45 : Natural killer cell mediated c..
Done pathway 46 : T cell receptor signaling path..
Done pathway 47 : B cell receptor signaling path..
Done pathway 48 : Fc epsilon RI signaling pathwa..
Done pathway 49 : Fc gamma R-mediated phagocytos..
Done pathway 50 : Leukocyte transendothelial mig..
Done pathway 51 : Intestinal immune network for ..
Done pathway 52 : Circadian rhythm - mammal..
Done pathway 53 : Long-term potentiation..
Done pathway 54 : Neurotrophin signaling pathway..
Done pathway 55 : Retrograde endocannabinoid sig..
```

```
Done pathway 56 : Glutamatergic synapse..
Done pathway 57 : Cholinergic synapse..
Done pathway 58 : Serotonergic synapse..
Done pathway 59 : GABAergic synapse..
Done pathway 60 : Dopaminergic synapse..
Done pathway 61 : Long-term depression..
Done pathway 62 : Olfactory transduction..
Done pathway 63 : Taste transduction..
Done pathway 64 : Phototransduction..
Done pathway 65 : Regulation of actin cytoskelet..
Done pathway 66 : Insulin signaling pathway..
Done pathway 67 : GnRH signaling pathway..
Done pathway 68 : Progesterone-mediated oocyte m..
Done pathway 69 : Melanogenesis..
Done pathway 70 : Adipocytokine signaling pathwa..
Done pathway 71 : Type II diabetes mellitus..
Done pathway 72 : Type I diabetes mellitus..
Done pathway 73 : Maturity onset diabetes of the..
Done pathway 74 : Aldosterone-regulated sodium r..
Done pathway 75 : Endocrine and other factor-reg..
Done pathway 76 : Vasopressin-regulated water re..
Done pathway 77 : Salivary secretion..
Done pathway 78 : Gastric acid secretion..
Done pathway 79 : Pancreatic secretion..
Done pathway 80 : Carbohydrate digestion and abs..
Done pathway 81 : Bile secretion..
Done pathway 82 : Mineral absorption..
Done pathway 83 : Alzheimer's disease..
Done pathway 84 : Parkinson's disease..
Done pathway 85 : Amyotrophic lateral sclerosis ..
Done pathway 86 : Huntington's disease..
Done pathway 87 : Prion diseases..
Done pathway 88 : Cocaine addiction..
Done pathway 89 : Amphetamine addiction..
Done pathway 90 : Morphine addiction..
Done pathway 91 : Alcoholism..
Done pathway 92 : Bacterial invasion of epitheli..
Done pathway 93 : Vibrio cholerae infection..
Done pathway 94 : Epithelial cell signaling in H..
Done pathway 95 : Pathogenic Escherichia coli in..
Done pathway 96 : Shigellosis..
Done pathway 97 : Salmonella infection..
Done pathway 98 : Pertussis..
Done pathway 99 : Legionellosis..
Done pathway 100 : Leishmaniasis..
Done pathway 101 : Chagas disease (American trypa..
Done pathway 102 : African trypanosomiasis..
Done pathway 103 : Malaria..
Done pathway 104 : Toxoplasmosis..
Done pathway 105 : Amoebiasis..
Done pathway 106 : Staphylococcus aureus infectio..
Done pathway 107 : Tuberculosis..
Done pathway 108 : Hepatitis C..
Done pathway 109 : Measles..
Done pathway 110 : Influenza A..
Done pathway 111 : HTLV-I infection..
Done pathway 112 : Herpes simplex infection..
Done pathway 113 : Epstein-Barr virus infection..
Done pathway 114 : Pathways in cancer..
Done pathway 115 : Transcriptional misregulation ..
Done pathway 116 : Viral carcinogenesis..
Done pathway 117 : Colorectal cancer..
Done pathway 118 : Renal cell carcinoma..
Done pathway 119 : Pancreatic cancer..
Done pathway 120 : Endometrial cancer..
Done pathway 121 : Glioma..
Done pathway 122 : Prostate cancer..
Done pathway 123 : Thyroid cancer..
```

```
Done pathway 124 : Basal cell carcinoma..
Done pathway 125 : Melanoma..
Done pathway 126 : Bladder cancer..
Done pathway 127 : Chronic myeloid leukemia..
Done pathway 128 : Acute myeloid leukemia..
Done pathway 129 : Small cell lung cancer..
Done pathway 130 : Non-small cell lung cancer..
Done pathway 131 : Asthma..
Done pathway 132 : Autoimmune thyroid disease..
Done pathway 133 : Systemic lupus erythematosus..
Done pathway 134 : Rheumatoid arthritis..
Done pathway 135 : Allograft rejection..
Done pathway 136 : Graft-versus-host disease..
Done pathway 137 : Arrhythmogenic right ventricul..
Done pathway 138 : Dilated cardiomyopathy..
Done pathway 139 : Viral myocarditis..
```

SPIA outputs a table showing significantly dysregulated pathways based on over-representation and signaling perturbations accumulation. The table shows the following information:

- **pSize**: the number of genes on the pathway
- **NDE**: the number of DE genes per pathway
- **tA**: the observed total perturbation accumulation in the pathway
- **pNDE**: the probability to observe at least NDE genes on the pathway using hypergeometric model (similar to ORA)
- **pPERT**: the probability to observe a total accumulation more extreme than tA only by chance
- **pG**: the p-value obtained by combining pNDE and pPERT
- **pGFdr and PGFWER**: are the False Discovery Rate and Bonferroni adjusted global p-values, respectively
- **Status**: gives the direction in which the pathway is perturbed (activated or inhibited)
- **KEGGLINK**: gives a web link to KEGG website that **displays the pathway image** with the differentially expressed genes highlighted in red

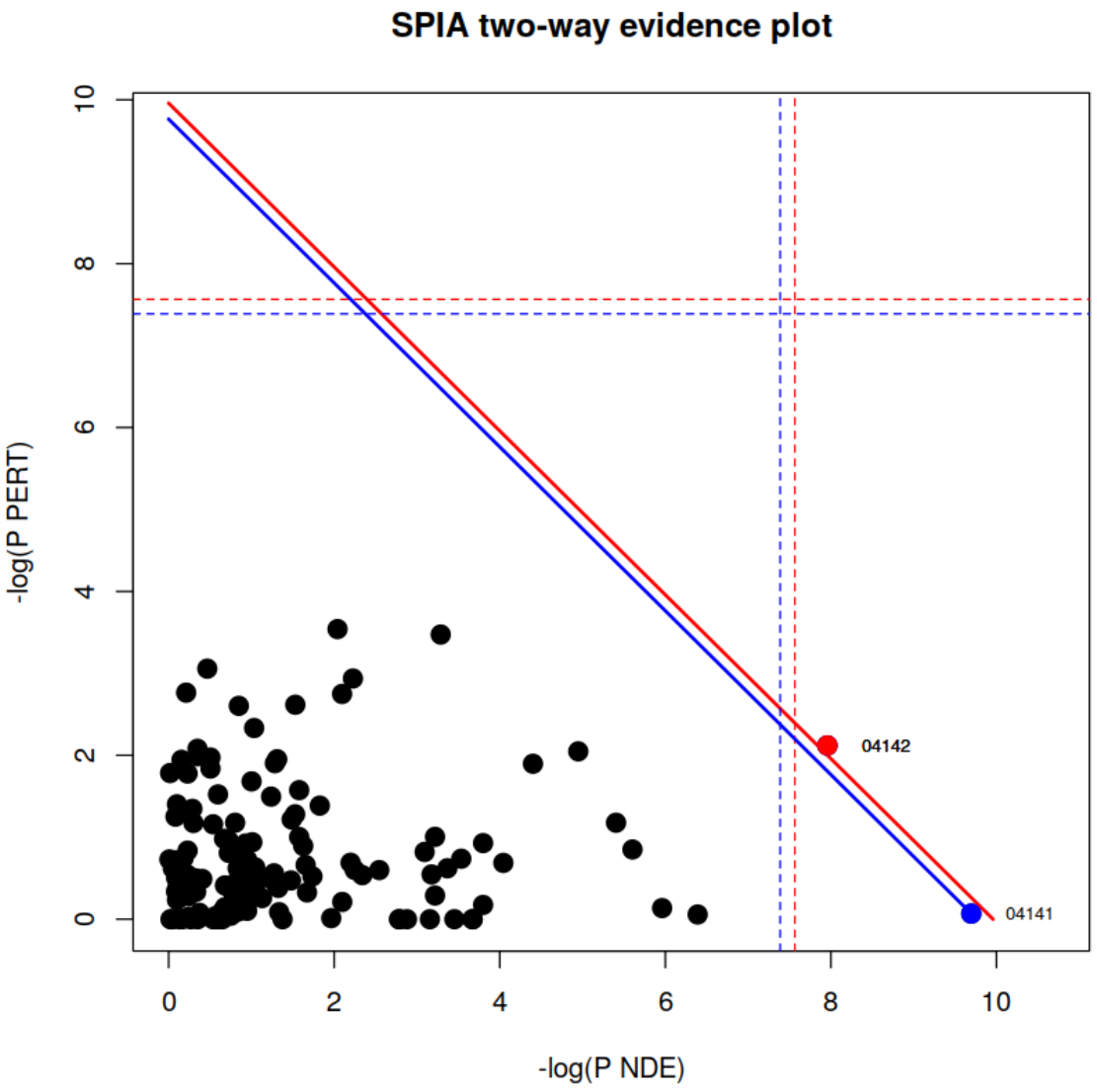
In [539]:
```
1 head(spia_result,n=20)
```

A data.frame: 20 × 12

Name	ID	pSize	NDE	pNDE	tA	pPERT	pG	pGFdr	pGFWER
<chr>	<chr>	<int>	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Lysosome	04142	114	26	3.500043e-04	1.4193465	0.120	0.0004652745	0.04177465	0.0628120
Protein processing in endoplasmic reticulum	04141	151	34	6.155982e-05	-0.5337849	0.934	0.0006188837	0.04177465	0.0835493
Bile secretion	04976	43	11	7.093523e-03	-2.2404665	0.129	0.0073173274	0.22647179	0.9878392
Antigen processing and presentation	04612	54	11	3.734782e-02	8.8810176	0.031	0.0089858376	0.22647179	1.0000000
p53 signaling pathway	04115	64	15	4.493784e-03	-4.6609554	0.308	0.0104951268	0.22647179	1.0000000
Viral myocarditis	05416	51	13	3.685691e-03	4.3767077	0.427	0.0117314528	0.22647179	1.0000000
Epstein-Barr virus infection	05169	179	34	1.677292e-03	0.5673962	0.944	0.0117932137	0.22647179	1.0000000
Viral carcinogenesis	05203	181	31	1.225846e-02	1.2676597	0.150	0.0134205504	0.22647179	1.0000000
Parkinson's disease	05012	91	20	2.576420e-03	-1.0333830	0.873	0.0159630708	0.23944606	1.0000000
Focal adhesion	04510	151	22	1.299390e-01	44.8568483	0.029	0.0247992844	0.33479034	1.0000000
Cytokine-cytokine receptor interaction	04060	132	20	1.078692e-01	13.0331866	0.053	0.0352417210	0.43251203	1.0000000
HTLV-I infection	05166	221	31	1.229931e-01	22.4802468	0.064	0.0460053098	0.48855002	1.0000000
Small cell lung cancer	05222	74	15	1.753469e-02	7.2752138	0.503	0.0505448304	0.48855002	1.0000000
Aldosterone-regulated sodium reabsorption	04960	26	7	2.239302e-02	3.1460303	0.395	0.0506644466	0.48855002	1.0000000
Alcoholism	05034	142	24	2.914751e-02	10.8901220	0.478	0.0734735153	0.64546467	1.0000000
Arrhythmogenic right ventricular cardiomyopathy (ARVC)	05412	48	10	4.001306e-02	0.5886369	0.366	0.0764995165	0.64546467	1.0000000
Chemokine signaling pathway	04062	130	18	2.163374e-01	28.7235144	0.073	0.0813037923	0.64564776	1.0000000
Alzheimer's disease	05010	137	23	3.448388e-02	-2.7605867	0.537	0.0923859051	0.66209678	1.0000000
Thyroid cancer	05216	26	7	2.239302e-02	-0.8691405	0.840	0.0935495640	0.66209678	1.0000000
ECM-receptor interaction	04512	49	10	4.536623e-02	3.3532227	0.440	0.0980884121	0.66209678	1.0000000

In [550]:

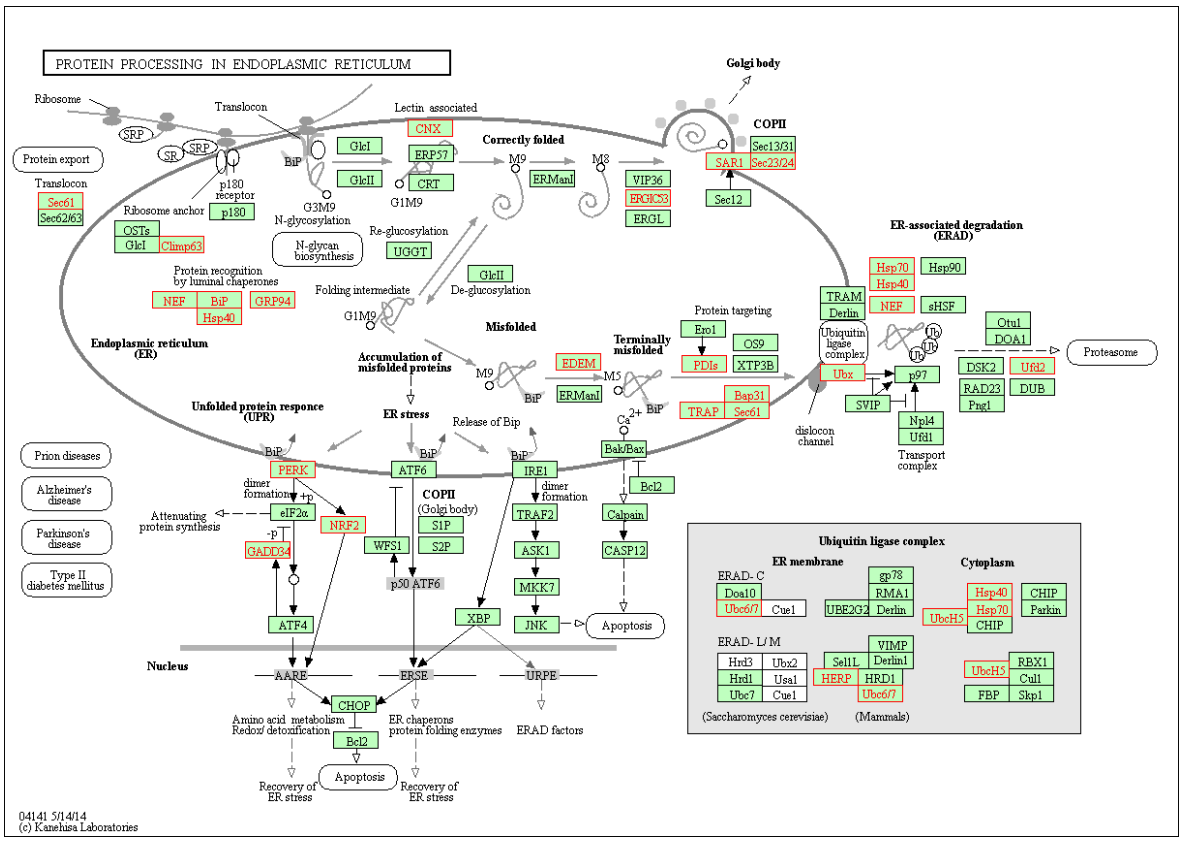
```
# view the significantly dysregulated pathways by viewing the over-represe
# for each pathway
plotP(spia_result, threshold=0.07)

# In this plot, each pathway is a point and the coordinates are the log of
# and the p-value from perturbations, pPERT. The oblique lines in the plot
# based on the combined evidence.
```



In [37]:

```
# If we choose to explore the significant genes from our dataset occuriing
# we can subset our SPIA results

# Look at pathway 03013 and view kegglink
subset(spia_result, ID == "04142")
```

```
Error in subset(spia_result, ID == "04142"): object 'spia_result' not found
Traceback:

1. subset(spia_result, ID == "04142")
```



Gene set enrichment analysis using GAGE and Pathview

Gene set enrichment analysis using [GAGE (Generally Applicable Gene-set Enrichemnt for Pathway Analysis (http://bioconductor.org/packages/release/bioc/html/gage.html)](http://bioconductor.org/packages/release/bioc/html/gage.html) and [Pathview (http://bioconductor.org/packages/release/bioc/html/pathview.html)](http://bioconductor.org/packages/release/bioc/html/pathview.html) tools are performed using a slightly different type of algoritm.

GAGE assumes a gene set comes from a different distribution then the background and uses two-sample t-test to account for the gene set specific variance as well as the background variance. The two-sample t-test used by GAGE identifies gene sets with modest but consistent changes in gene expression level.

Pathviews allows for the integration of the data generated by GAGE and visualization of the pathways from the dataset.

Exploring enrichment of KEGG pathways

In [558]:

```
# Loading the packages needed for GAGE and Pathview analysis
library(gage)
library(pathview)
library(gageData)
library(org.Hs.eg.db)

# Create datasets with KEGG gene sets to test
kegg_human <- kegg.gsets(species = "human", id.type = "kegg")
names(kegg_human)

kegg.gs <- kegg_human$kg.sets[kegg_human$sigmet.idx]
head(kegg.gs)

```

'kg.sets' 'sigmet.idx' 'sig.idx' 'met.idx' 'dise.idx'

$`hsa00970 Aminoacyl-tRNA biosynthesis`

'10056' '10352' '10667' '118672' '123263' '123283' '124454' '16' '1615' '2058' '2193'
'23395' '23438' '25973' '2617' '283459' '3035' '3376' '3735' '4141' '4511' '4553' '4555'
'4556' '4558' '4563' '4564' '4565' '4566' '4567' '4568' '4569' '4570' '4571' '4572' '4573'
'4574' '4575' '4576' '4577' '4578' '4579' '4677' '51067' '51091' '51520' '5188' '54938'
'55157' '55278' '55699' '57038' '57176' '57505' '5859' '5917' '6301' '6897' '7407' '7453'
'79587' '79731' '80222' '833' '8565' '92935'

$`hsa02010 ABC transporters`

'10057' '10058' '10060' '10257' '10347' '10349' '10350' '10351' '1080' '11194' '1244'
'154664' '1672' '19' '20' '21' '215' '22' '225' '23456' '23457' '23460' '23461' '24'
'26154' '340273' '368' '4363' '5243' '5244' '5825' '5826' '64137' '64240' '64241' '6833'
'6890' '6891' '85320' '8647' '8714' '89845' '94160' '9429' '9619'

$`hsa03008 Ribosome biogenesis in eukaryotes`

'100169751' '100169753' '100169754' '100169755' '100169756' '100169757' '100169758'
'100169759' '100169760' '100169761' '100169762' '100169763' '100169764' '100169765'
'100169766' '100169767' '100169768' '100288562' '10171' '101929601' '101929627' '10199'
'102157402' '10248' '10436' '10482' '10528' '10556' '10557' '10607' '10775' '10799'
'10813' '10885' '10940' '134430' '138716' '1457' '1459' '1460' '166378' '1736' '2091'
'22803' '23160' '23195' '23560' '25996' '26354' '26851' '27341' '283106' '28987' '29102'
'29107' '29889' '3692' '4549' '4550' '4809' '4931' '51068' '51077' '51096' '51119' '51367'
'51602' '54433' '54464' '54552' '54913' '55127' '55131' '55226' '55272' '55341' '55505'
'55651' '55781' '55813' '55916' '55998' '56000' '56001' '57455' '5822' '5901' '6023'
'643802' '65083' '6949' '728343' '7514' '780851' '780852' '780853' '79631' '81691'
'83732' '84128' '84135' '84916' '92856' '9724' '9790'

$`hsa03010 Ribosome`

'100169751' '100169753' '100169754' '100169755' '100169756' '100169757' '100169758'
'100169759' '100169760' '100169761' '100169762' '100169763' '100169764' '100169765'
'100169766' '100169767' '100169768' '100526842' '100529097' '100529239' '10573' '11222'
'11224' '124995' '140032' '140801' '200916' '2197' '219927' '23521' '25873' '28998'
'29074' '29088' '29093' '3921' '4549' '4550' '4736' '51021' '51023' '51065' '51069'
'51073' '51081' '51116' '51121' '51187' '51263' '51264' '51318' '51373' '54460' '54948'
'55052' '55168' '55173' '6122' '6123' '6124' '6125' '6128' '6129' '6130' '6132' '6133'
'6134' '6135' '6136' '6137' '6138' '6139' '6141' '6142' '6143' '6144' '6146' '6147' '6150'
'6152' '6154' '6155' '6156' '6157' '6158' '6159' '6160' '6161' '6164' '6165' '6166' '6167'
'6168' '6169' '6170' '6171' '6173' '6175' '6176' '6181' '6182' '6183' '6187' '6188' '6189'
'6191' '6192' '6193' '6194' '6201' '6202' '6203' '6204' '6205' '6206' '6207' '6208' '6209'
'6210' '6217' '6218' '6222' '6223' '6224' '6227' '6228' '6229' '6230' '6231' '6232' '6233'
'6234' '6235' '63875' '63931' '64928' '64960' '64963' '64965' '64968' '64969' '64979'
'64981' '64983' '65003' '65005' '65008' '7311' '79590' '9045' '9349' '9553' '9801'

$`hsa03013 RNA transport`

'100101267' '10073' '10189' '10209' '10248' '10250' '10284' '10289' '10419' '10460'
'10482' '10556' '10557' '10605' '10762' '10775' '10799' '10921' '10940' '11097' '11102'
'11171' '11218' '11260' '1207' '129401' '132430' '138716' '140886' '1915' '1917' '1964'
'1965' '1967' '1968' '1973' '1974' '1975' '1977' '1978' '1979' '1981' '1982' '1983'
'22794' '22916' '22985' '23165' '23191' '23225' '23279' '2332' '23511' '23636' '253314'
'25929' '26019' '26827' '26851' '26986' '26999' '29107' '340529' '348995' '3646' '3837'
'387082' '4116' '4686' '4927' '4928' '5042' '50628' '51068' '51095' '51367' '51808'
'53371' '5411' '54913' '54960' '55110' '55520' '55706' '55746' '55916' '55998' '56000'
'56001' '57122' '57187' '57510' '5901' '5903' '5905' '59343' '5976' '60528' '6396'
'645974' '65109' '65110' '6606' '6607' '6612' '6613' '7175' '728343' '728689' '7329'
'7341' '7514' '780851' '780852' '780853' '79023' '7919' '79228' '79760' '79833' '79897'
'79902' '80145' '8021' '80336' '8086' '8087' '81929' '84305' '84321' '8480' '8487' '8563'
'8637' '8661' '8662' '8663' '8664' '8665' '8666' '8667' '8668' '8669' '8672' '8761' '8890'
'8891' '8892' '8893' '8894' '9086' '91181' '94026' '9470' '9513' '9631' '9669' '96764'
'9688' '9775' '9818' '9883' '9939' '9972' '9984'

$`hsa03015 mRNA surveillance pathway`

'100529063' '10189' '10250' '10284' '10482' '10767' '10898' '10914' '10921' '10978'
'11051' '11052' '124540' '132430' '140886' '1477' '1478' '1479' '2107' '22794' '22916'
'22985' '23049' '23283' '23293' '23381' '23708' '26019' '26528' '26986' '28227' '29101'
'29107' '2935' '29894' '340529' '390748' '4116' '4440' '4686' '5042' '51585' '51692'
'53918' '53981' '5411' '5499' '5500' '5501' '55012' '55110' '5515' '5516' '5518' '5519'
'5520' '5521' '5522' '5523' '5525' '5526' '5527' '5528' '5529' '55339' '55844' '55916'
'55998' '56000' '56001' '56903' '5976' '645974' '64895' '65109' '65110' '728343' '7919'
'79869' '80335' '80336' '8106' '81608' '8189' '84305' '8731' '8732' '8761' '9775' '9887'
'9939'

Now that we have our pathways to test, we need to bring in our data. We will use the log2 fold changes output by differential expression analysis to determine particular pathways are enriched. GAGE requirez the genes have Entrez IDs, so we will use these IDs for the analysis.

In [559]:

```
# Run GAGE
keggres = gage(foldchanges, gsets=kegg.gs, same.dir=T)
names(keggres)
head(keggres$greater) #Pathways that are up-regulated
```

'greater' 'less' 'stats'

A matrix: 6 × 6 of type dbl

	p.geomean	stat.mean	p.val	q.val	set.size	exp1
hsa04142 Lysosome	0.005342084	2.573883	0.005342084	0.6222262	116	0.005342084
hsa04668 TNF signaling pathway	0.005580504	2.562855	0.005580504	0.6222262	102	0.005580504
hsa04060 Cytokine-cytokine receptor interaction	0.010365248	2.328639	0.010365248	0.6877766	140	0.010365248
hsa04510 Focal adhesion	0.012336800	2.257796	0.012336800	0.6877766	153	0.012336800
hsa04380 Osteoclast differentiation	0.032582010	1.855313	0.032582010	0.8429584	95	0.032582010
hsa04064 NF-kappa B signaling pathway	0.032649385	1.857448	0.032649385	0.8429584	82	0.032649385

In [560]:
```
1 head(keggres$less) #Pathways that are down-regulated
```

A matrix: 6 × 6 of type dbl

	p.geomean	stat.mean	p.val	q.val	set.size	exp1
hsa03010 Ribosome	2.487081e-08	-5.698442	2.487081e-08	5.546191e-06	129	2.487081e-08
hsa03013 RNA transport	2.496195e-03	-2.834603	2.496195e-03	2.301006e-01	139	2.496195e-03
hsa00280 Valine, leucine and isoleucine degradation	3.095524e-03	-2.813985	3.095524e-03	2.301006e-01	44	3.095524e-03
hsa03040 Spliceosome	4.693023e-03	-2.626176	4.693023e-03	2.326613e-01	124	4.693023e-03
hsa00190 Oxidative phosphorylation	5.216620e-03	-2.592156	5.216620e-03	2.326613e-01	96	5.216620e-03
hsa00970 Aminoacyl-tRNA biosynthesis	6.281894e-03	-2.551462	6.281894e-03	2.334770e-01	43	6.281894e-03

In [582]:
```
1 # Explore genes that are up-regulated
2 sel_up <- keggres$greater[, "q.val"] < 0.8 & !is.na(keggres$greater[, "q.v
3 path_ids_up <- rownames(keggres$greater)[sel_up]
4 path_ids_up
```

'hsa04142 Lysosome' 'hsa04668 TNF signaling pathway'
'hsa04060 Cytokine-cytokine receptor interaction' 'hsa04510 Focal adhesion'

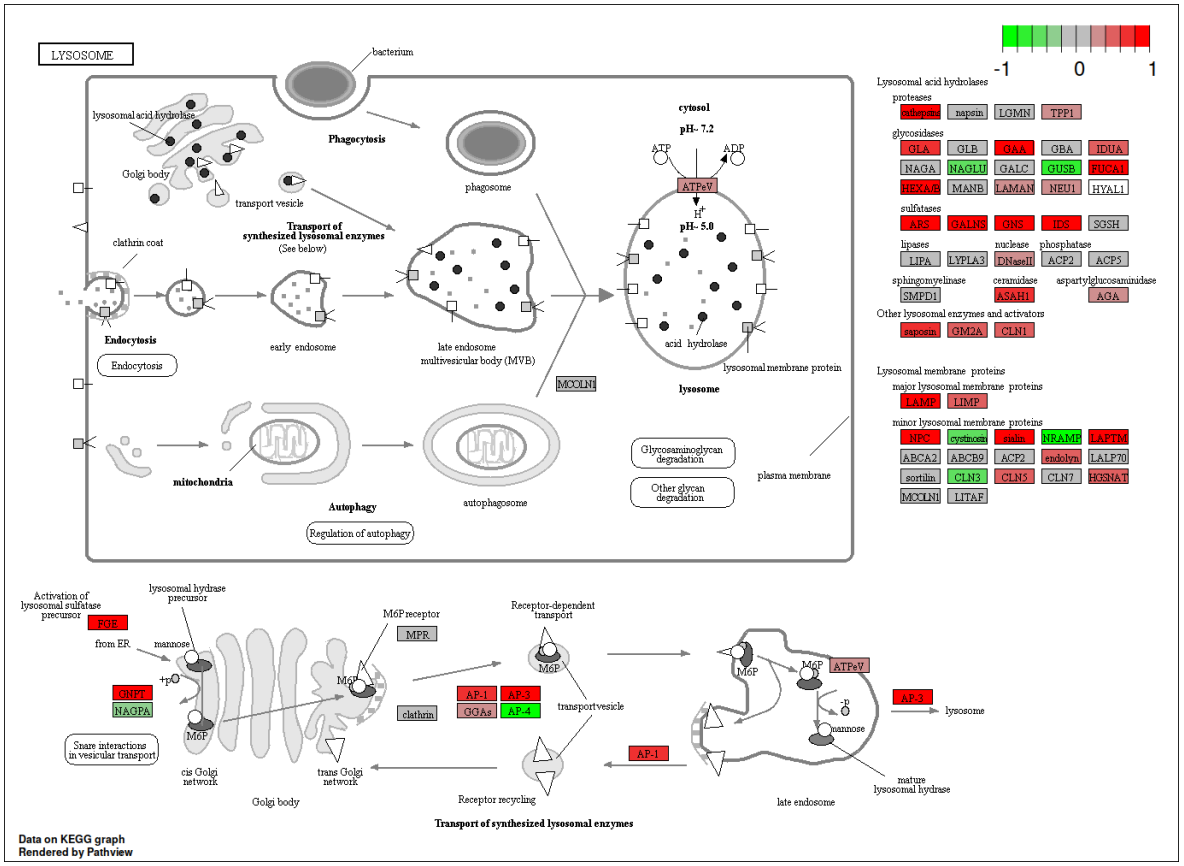
In [583]:
```
1 # Get the pathway IDs for the significantly up-regulated pathways
2 keggresids = substr(path_ids_up, start=1, stop=8)
3 keggresids
```

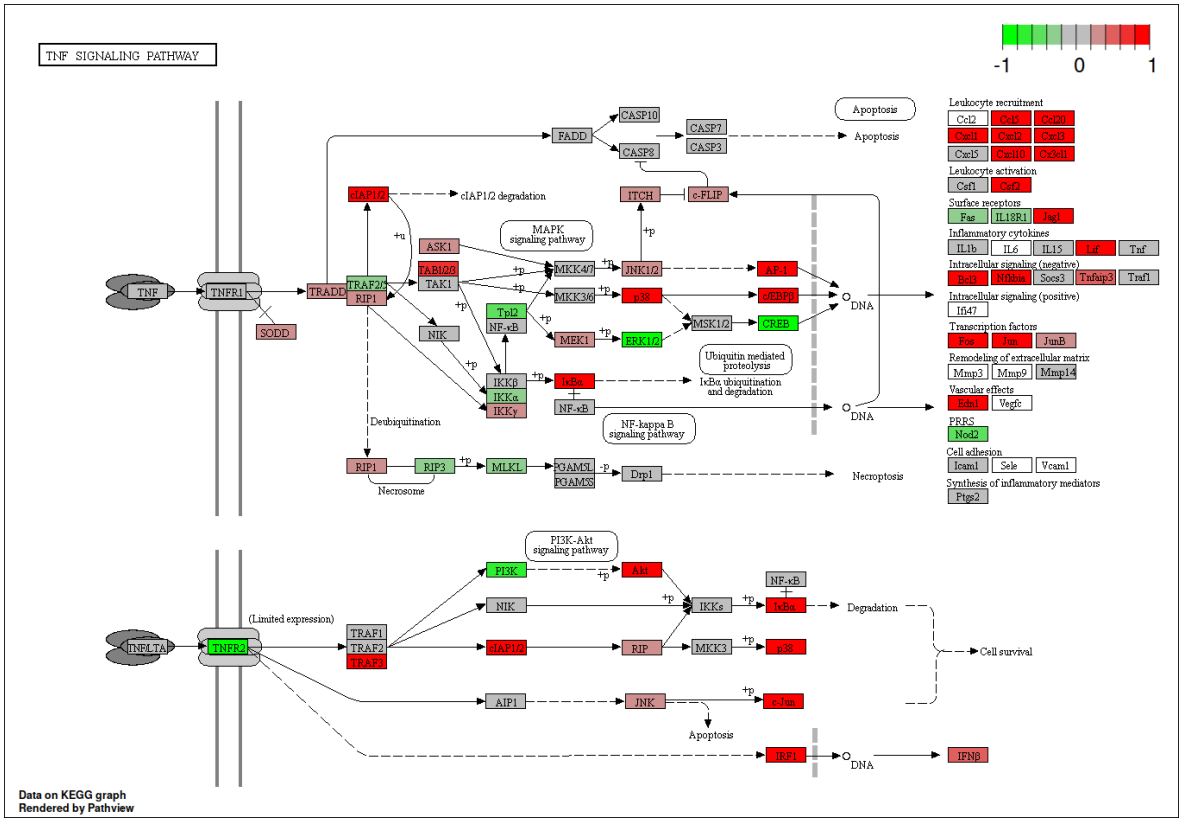
'hsa04142' 'hsa04668' 'hsa04060' 'hsa04510'

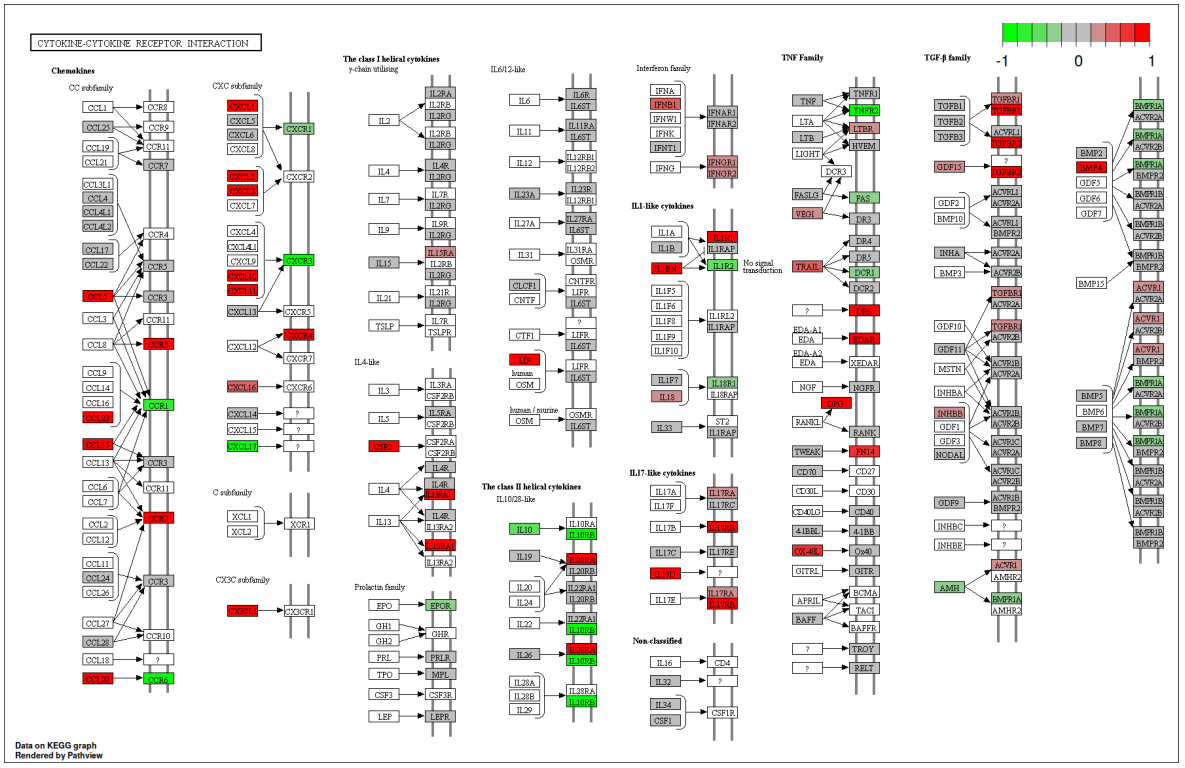
Now that we have the IDs for the pathways that are significantly up-regulated in our dataset, we can visualize these pathways and the genes identified form our dataset causing these pathways to be enriched using Pathview.

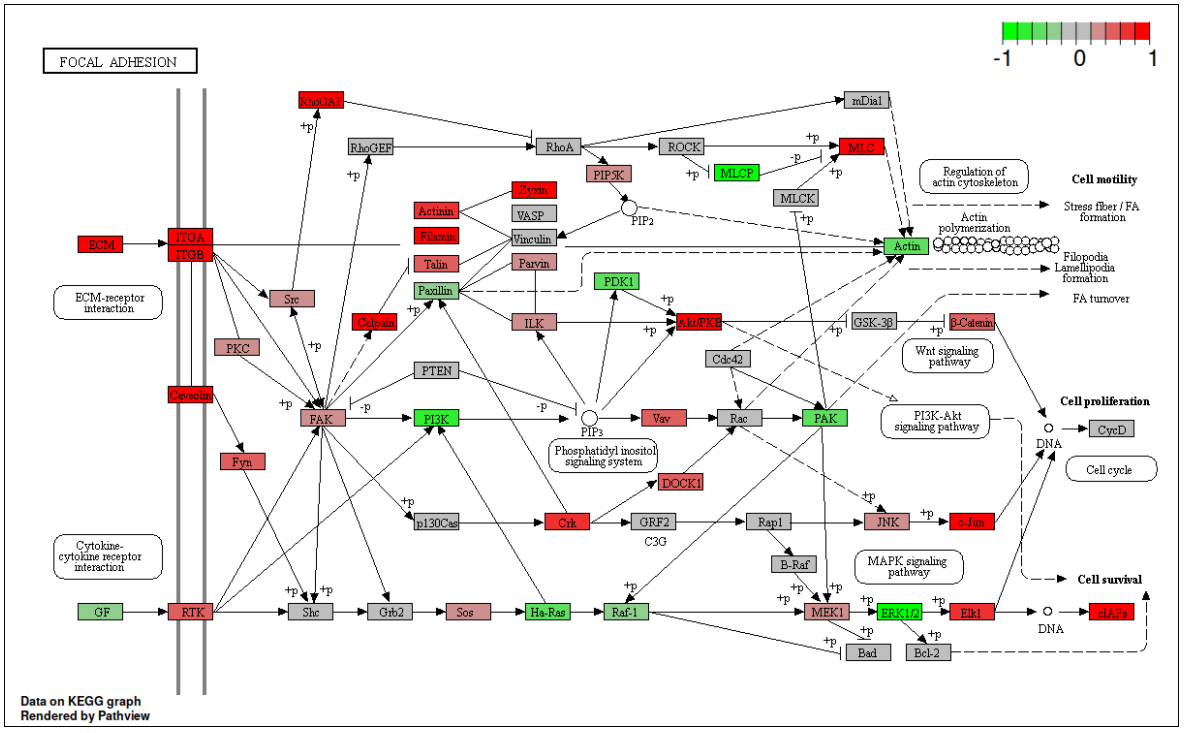
In [585]:
```
1 # Run Pathview
2 # Use Pathview to view significant up-regulated pathways
3 pathview(gene.data = foldchanges, pathway.id=keggresids, species="hsa")
```

```
Info: Downloading xml files for hsa04142, 1/1 pathways..
Info: Downloading png files for hsa04142, 1/1 pathways..
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa04142.pathview.png
Info: some node width is different from others, and hence adjusted!
Info: Downloading xml files for hsa04668, 1/1 pathways..
Info: Downloading png files for hsa04668, 1/1 pathways..
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa04668.pathview.png
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa04060.pathview.png
Info: Downloading xml files for hsa04510, 1/1 pathways..
Info: Downloading png files for hsa04510, 1/1 pathways..
'select()' returned 1:1 mapping between keys and columns
Info: Working in directory /home/sebastian/projects/RNASeq
Info: Writing image file hsa04510.pathview.png
```


LYSOSOME
bacterium
lysosomal acid hydrolase
Golgi body
Phagocytosis
phagosome
transport vesicle
Transport of synthesized lysosomal enzymes
(See below)
clathrin coat
Endocytosis
Endocytosis
early endosome
late endosome
multivesicular body (MVB)
MCOLN1
cytosol
pH~ 7.2
ATP
ADP
ATPeV
H+
pH~ 5.0
acid hydrolase
lysosomal membrane protein
lysosome
mitochondria
Autophagy
Regulation of autophagy
autophagosome
Glycosaminoglycan degradation
Other glycan degradation
plasma membrane
-1
0
1
Lysosomal acid hydrolases
proteases
cathepsins
napsin
LGMN
TPP1
glycosidases
GLA
GLB
GAA
GBA
IDUA
NAGA
NAGLU
GALC
GUSB
FUCA1
HEXA/B
MANB
LAMAN
NEU1
HYAL1
sulfatases
ARS
GALNS
GNS
IDS
SGSH
lipases
LIPA
LYPLA3
nuclease
DNaseII
phosphatase
ACP2
ACP5
sphingomyelinase
SMPD1
ceramidase
ASAH1
aspartylglucosaminidase
AGA
Other lysosomal enzymes and activators
saposin
GM2A
CLN1
Lysosomal membrane proteins
major lysosomal membrane proteins
LAMP
LIMP
minor lysosomal membrane proteins
NPC
cystinosin
sialin
NRAMP
LAPTM
ABCA2
ABCB9
ACP2
endolyn
LALP70
sortilin
CLN3
CLN5
CLN7
HGSNAT
MCOLN1
LITAF
Activation of lysosomal sulfatase precursor
FGE
from ER
lysosomal hydrase precursor
mannose
GNPT
NAGPA
M6P
M6P receptor
MPR
clathrin
AP-1
AP-3
GGAs
AP-4
Receptor-dependent transport
transport vesicle
Snare interactions in vesicular transport
cis Golgi network
trans Golgi network
Golgi body
Receptor recycling
AP-1
AP-3
lysosome
mannose
mature lysosomal hydrase
late endosome
Transport of synthesized lysosomal enzymes
Data on KEGG graph
Rendered by Pathview


TNF SIGNALING PATHWAY
-1
0
1
TNF
TNFR1
SODD
TRADD
TRAF2/5
RIP1
FADD
CASP10
CASP8
CASP7
CASP3
Apoptosis
cIAP1/2
cIAP1/2 degradation
ITCH
c-FLIP
ASK1
TAB1/2/3
TAK1
MAPK signaling pathway
MKK4/7
MKK3/6
JNK1/2
p38
AP-1
C/EBPβ
CREB
MSK1/2
DNA
Tpl2
NF-κB
MEK1
ERK1/2
NIK
IKKβ
IKKα
IKKγ
IκBα
NF-κB
Ubiquitin mediated proteolysis
IκBα ubiquitination and degradation
NF-kappa B signaling pathway
Deubiquitination
RIP1
RIP3
MLKL
PGAM5L
PGAM5S
Drp1
Necrosome
Necroptosis
Leukocyte recruitment
Ccl2
Ccl5
Ccl20
Cxcl1
Cxcl2
Cxcl3
Cxcl5
Cxcl10
Cx3cl1
Leukocyte activation
Csf1
Csf2
Surface receptors
Fas
IL18R1
Jag1
Inflammatory cytokines
IL1b
IL6
IL15
Lif
Tnf
Intracellular signaling (negative)
Bcl3
NFkbia
Socs3
Tnfaip3
Traf1
Intracellular signaling (positive)
Ifi47
Transcription factors
Fos
Jun
JunB
Remodeling of extracellular matrix
Mmp3
Mmp9
Mmp14
Vascular effects
Edn1
Vegfc
PRRS
Nod2
Cell adhesion
Icam1
Sele
Vcam1
Synthesis of inflammatory mediators
Ptgs2
TNFLTA
TNFR2
(Limited expression)
TRAF1
TRAF2
TRAF3
PI3K-Akt signaling pathway
PI3K
Akt
NIK
IKKs
NF-κB
IκBα
Degradation
cIAP1/2
RIP
MKK3
p38
Cell survival
AIP1
JNK
c-Jun
Apoptosis
IRF1
IFNβ
Data on KEGG graph
Rendered by Pathview





In [586]:

```
#Acquire datasets
data(go.sets.hs)
head(names(go.sets.hs))

```

'GO:0000002 mitochondrial genome maintenance' 'GO:0000003 reproduction'
'GO:0000012 single strand break repair' 'GO:0000018 regulation of DNA recombination'
'GO:0000019 regulation of mitotic recombination' 'GO:0000022 mitotic spindle elongation'

In [587]:
```
data(go.subs.hs)
names(go.subs.hs)
head(go.subs.hs$MF)
```

'BP' 'CC' 'MF'

13299 13300 13301 13302 13303 13304

In [588]:
```
# Use gage to explore enriched biological processes
# Biological process
go_bp_sets = go.sets.hs[go.subs.hs$BP]
```

In [589]:
```
# Run GAGE
go_bp_res = gage(foldchanges, gsets=go_bp_sets, same.dir=T)
class(go_bp_res)
names(go_bp_res)
head(go_bp_res$greater)
```

'list'

'greater' 'less' 'stats'

A matrix: 6 × 6 of type dbl

	p.geomean	stat.mean	p.val	q.val	set.size	exp1
GO:0040012 regulation of locomotion	0.0002348654	3.515962	0.0002348654	0.6270366	327	0.0002348654
GO:0051270 regulation of cellular component movement	0.0006421656	3.233070	0.0006421656	0.6270366	342	0.0006421656
GO:0008285 negative regulation of cell proliferation	0.0006672386	3.220136	0.0006672386	0.6270366	393	0.0006672386
GO:2000145 regulation of cell motility	0.0009505929	3.118941	0.0009505929	0.6270366	310	0.0009505929
GO:0030334 regulation of cell migration	0.0013154689	3.021375	0.0013154689	0.6270366	289	0.0013154689
GO:0019221 cytokine-mediated signaling pathway	0.0016088983	2.962724	0.0016088983	0.6270366	255	0.0016088983

In [597]:

```
go_df_enriched <- data.frame(go_bp_res$greater)
GO_enriched_BP <- subset(go_df_enriched, q.val < 0.7)
GO_enriched_BP
```

A data.frame: 1618 × 6

	p.geomean	stat.mean	p.val	q.val	set.size	exp1
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
GO:0040012 regulation of locomotion	0.0002348654	3.515962	0.0002348654	0.6270366	327	0.0002348654
GO:0051270 regulation of cellular component movement	0.0006421656	3.233070	0.0006421656	0.6270366	342	0.0006421656
GO:0008285 negative regulation of cell proliferation	0.0006672386	3.220136	0.0006672386	0.6270366	393	0.0006672386
GO:2000145 regulation of cell motility	0.0009505929	3.118941	0.0009505929	0.6270366	310	0.0009505929
GO:0030334 regulation of cell migration	0.0013154689	3.021375	0.0013154689	0.6270366	289	0.0013154689
GO:0019221 cytokine-mediated signaling pathway	0.0016088983	2.962724	0.0016088983	0.6270366	255	0.0016088983
GO:0001944 vasculature development	0.0018007094	2.920198	0.0018007094	0.6270366	392	0.0018007094
GO:0001568 blood vessel development	0.0019183011	2.900747	0.0019183011	0.6270366	372	0.0019183011
GO:0045087 innate immune response	0.0020181379	2.885275	0.0020181379	0.6270366	381	0.0020181379
GO:0030155 regulation of cell adhesion	0.0023813084	2.838794	0.0023813084	0.6270366	204	0.0023813084
GO:0051048 negative regulation of secretion	0.0032865936	2.768801	0.0032865936	0.6270366	77	0.0032865936
GO:0032101 regulation of response to external stimulus	0.0036143405	2.698244	0.0036143405	0.6270366	249	0.0036143405
GO:0042063 gliogenesis	0.0041123328	2.665925	0.0041123328	0.6270366	116	0.0041123328
GO:0034097 response to cytokine stimulus	0.0045448414	2.616122	0.0045448414	0.6270366	381	0.0045448414
GO:0022603 regulation of anatomical structure morphogenesis	0.0046319653	2.607844	0.0046319653	0.6270366	453	0.0046319653
GO:0033993 response to lipid	0.0049782567	2.583894	0.0049782567	0.6270366	382	0.0049782567
GO:0044057 regulation of system process	0.0051944747	2.571385	0.0051944747	0.6270366	286	0.0051944747
GO:0000165 MAPK cascade	0.0054888231	2.549161	0.0054888231	0.6270366	417	0.0054888231
GO:0050776 regulation of immune response	0.0063565011	2.498112	0.0063565011	0.6270366	368	0.0063565011
GO:0001501 skeletal system development	0.0066943312	2.481733	0.0066943312	0.6270366	262	0.0066943312
GO:0048732 gland development	0.0067699879	2.480001	0.0067699879	0.6270366	206	0.0067699879
GO:0034330 cell junction organization	0.0069557774	2.474323	0.0069557774	0.6270366	156	0.0069557774
GO:0048514 blood vessel morphogenesis	0.0070279628	2.462852	0.0070279628	0.6270366	324	0.0070279628
GO:0010001 glial cell differentiation	0.0071409730	2.472432	0.0071409730	0.6270366	99	0.0071409730
GO:0007044 cell-substrate junction assembly	0.0072310591	2.504816	0.0072310591	0.6270366	50	0.0072310591

	p.geomean	stat.mean	p.val	q.val	set.size	exp1
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
GO:0007229 integrin-mediated signaling pathway	0.0073349032	2.507395	0.0073349032	0.6270366	44	0.0073349032
GO:0052547 regulation of peptidase activity	0.0075270183	2.442124	0.0075270183	0.6270366	197	0.0075270183
GO:0031581 hemidesmosome assembly	0.0079264359	2.760388	0.0079264359	0.6270366	12	0.0079264359
GO:0006954 inflammatory response	0.0080163137	2.415341	0.0080163137	0.6270366	311	0.0080163137
GO:0048545 response to steroid hormone stimulus	0.0082984096	2.407793	0.0082984096	0.6270366	177	0.0082984096
⋮	⋮	⋮	⋮	⋮	⋮	⋮
GO:0010821 regulation of mitochondrion organization	0.2880896	0.5613277	0.2880896	0.6841046	42	0.2880896
GO:0046822 regulation of nucleocytoplasmic transport	0.2882422	0.5593548	0.2882422	0.6841046	127	0.2882422
GO:0034308 primary alcohol metabolic process	0.2882648	0.5695659	0.2882648	0.6841046	15	0.2882648
GO:0007569 cell aging	0.2882814	0.5601318	0.2882814	0.6841046	64	0.2882814
GO:0035094 response to nicotine	0.2883042	0.5632549	0.2883042	0.6841046	20	0.2883042
GO:0030521 androgen receptor signaling pathway	0.2883622	0.5597942	0.2883622	0.6841046	58	0.2883622
GO:0052646 alditol phosphate metabolic process	0.2886677	0.5614234	0.2886677	0.6841046	24	0.2886677
GO:0010885 regulation of cholesterol storage	0.2886727	0.5675371	0.2886727	0.6841046	11	0.2886727
GO:0034332 adherens junction organization	0.2887196	0.5586998	0.2887196	0.6841046	62	0.2887196
GO:0072332 intrinsic apoptotic signaling pathway by p53 class mediator	0.2899426	0.5564892	0.2899426	0.6865726	32	0.2899426
GO:0014068 positive regulation of phosphatidylinositol 3-kinase cascade	0.2910603	0.5532278	0.2910603	0.6886174	32	0.2910603
GO:0021515 cell differentiation in spinal cord	0.2912458	0.5578185	0.2912458	0.6886174	16	0.2912458
GO:0001708 cell fate specification	0.2913521	0.5519106	0.2913521	0.6886174	42	0.2913521
GO:0070482 response to oxygen levels	0.2928317	0.5456153	0.2928317	0.6916824	189	0.2928317
GO:0051898 negative regulation of protein kinase B signaling cascade	0.2938158	0.5485051	0.2938158	0.6935739	19	0.2938158
GO:0046341 CDP-diacylglycerol metabolic process	0.2944887	0.5526482	0.2944887	0.6945231	11	0.2944887
GO:0060840 artery development	0.2945850	0.5421643	0.2945850	0.6945231	43	0.2945850
GO:0050690 regulation of defense response to virus by virus	0.2958909	0.5406979	0.2958909	0.6971675	23	0.2958909
GO:0046006 regulation of activated T cell proliferation	0.2964698	0.5411469	0.2964698	0.6979864	18	0.2964698
GO:0019722 calcium-mediated signaling	0.2966540	0.5354891	0.2966540	0.6979864	63	0.2966540

	p.geomean	stat.mean	p.val	q.val	set.size	exp1
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
GO:0043392 negative regulation of DNA binding	0.2968064	0.5365935	0.2968064	0.6979864	32	0.2968064
GO:0007568 aging	0.2969762	0.5337141	0.2969762	0.6979864	145	0.2969762
GO:0019098 reproductive behavior	0.2974838	0.5335990	0.2974838	0.6987453	45	0.2974838
GO:0007141 male meiosis I	0.2979573	0.5393974	0.2979573	0.6991438	11	0.2979573
GO:0006476 protein deacetylation	0.2980230	0.5321391	0.2980230	0.6991438	46	0.2980230
GO:0050432 catecholamine secretion	0.2984004	0.5365949	0.2984004	0.6994002	17	0.2984004
GO:0033674 positive regulation of kinase activity	0.2986304	0.5285889	0.2986304	0.6994002	346	0.2986304
GO:0010677 negative regulation of cellular carbohydrate metabolic process	0.2986868	0.5341168	0.2986868	0.6994002	16	0.2986868
GO:0043552 positive regulation of phosphatidylinositol 3-kinase activity	0.2990816	0.5317590	0.2990816	0.6998763	19	0.2990816
GO:0002703 regulation of leukocyte mediated immunity	0.2992600	0.5280145	0.2992600	0.6998763	58	0.2992600

In [598]:
```
1 write.table(GO_enriched_BP, "GAGE_GO_BP.txt", quote=F)
```

Additional Resources for functional analysis

gProfileR (https://biit.cs.ut.ee/gprofiler/gost) is another tool for performing ORA, similar to clusterProfiler.

GeneMANIA (http://genemania.org/)

GenePattern (http://software.broadinstitute.org/cancer/software/genepattern/) (need to register)

WebGestalt (http://www.webgestalt.org/) (need to register)

AmiGO (http://amigo.geneontology.org/amigo)

GSEA (http://software.broadinstitute.org/gsea/index.jsp)

Sources used for this "patchwork" notebook

- https://descostesn.github.io/RIntroProgBio/whatisR.html (https://descostesn.github.io/RIntroProgBio/whatisR.html)
- http://r-statistics.co/R-Tutorial.html (http://r-statistics.co/R-Tutorial.html)
- http://mccarrolllab.org/dropseq/ (http://mccarrolllab.org/dropseq/)
- https://github.com/broadinstitute/Drop-seq/releases (https://github.com/broadinstitute/Drop-seq/releases)
- https://github.com/CGATOxford/UMI-tools/blob/master/doc/Single_cell_tutorial.md (https://github.com/CGATOxford/UMI-tools/blob/master/doc/Single_cell_tutorial.md)
- https://galaxyproject.github.io/training-material/topics/introduction/tutorials/galaxy-intro-ngs-data-managment/tutorial.html (https://galaxyproject.github.io/training-material/topics/introduction/tutorials/galaxy-intro-ngs-data-managment/tutorial.html)
- https://broadinstitute.github.io/2019_scWorkshop/index.html (https://broadinstitute.github.io/2019_scWorkshop/index.html)
- https://bioconductor.org/packages/release/bioc/html/edgeR.html (https://bioconductor.org/packages/release/bioc/html/edgeR.html)

- https://github.com/friedue/course_RNA-seq2019 (https://github.com/friedue/course_RNA-seq2019)
- https://github.com/hbctraining/GCC-BOSC-2018 (https://github.com/hbctraining/GCC-BOSC-2018)
- http://bioconductor.org/packages/release/workflows/vignettes/rnaseqGene/inst/doc/rnaseqGene.html (http://bioconductor.org/packages/release/workflows/vignettes/rnaseqGene/inst/doc/rnaseqGene.html)
- http://girke.bioinformatics.ucr.edu/GEN242/pages/mydoc/systemPipeRNAseq.html (http://girke.bioinformatics.ucr.edu/GEN242/pages/mydoc/systemPipeRNAseq.html)
- https://www.labome.com/method/RNA-seq.html (https://www.labome.com/method/RNA-seq.html)
- https://github.com/MonashBioinformaticsPlatform/RSeQC (https://github.com/MonashBioinformaticsPlatform/RSeQC)
- https://github.com/griffithlab/rnaseq_tutorial/wiki (https://github.com/griffithlab/rnaseq_tutorial/wiki)
- https://hartleys.github.io/QoRTs/ (https://hartleys.github.io/QoRTs/)
- https://www.bioinformatics.babraham.ac.uk/projects/fastqc/ (https://www.bioinformatics.babraham.ac.uk/projects/fastqc/)
- https://www.bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html (https://www.bioconductor.org/packages/devel/bioc/vignettes/DESeq2/inst/doc/DESeq2.html)
- https://galaxyproject.org/tutorials/rb_rnaseq/ (https://galaxyproject.org/tutorials/rb_rnaseq/)
- https://scrnaseq-course.cog.sanger.ac.uk/website/index.html (https://scrnaseq-course.cog.sanger.ac.uk/website/index.html)
- https://galaxyproject.github.io/training-material/topics/introduction/tutorials/galaxy-intro-101/tutorial.html (https://galaxyproject.github.io/training-material/topics/introduction/tutorials/galaxy-intro-101/tutorial.html)
- https://github.com/hbctraining/DGE_workshop (https://github.com/hbctraining/DGE_workshop)
- https://yulab-smu.github.io/clusterProfiler-book/index.html (https://yulab-smu.github.io/clusterProfiler-book/index.html)
- https://docs.gdc.cancer.gov/Data/Bioinformatics_Pipelines/DNA_Seq_Variant_Calling_Pipeline/ (https://docs.gdc.cancer.gov/Data/Bioinformatics_Pipelines/DNA_Seq_Variant_Calling_Pipeline/)
- https://bioconductor.org/packages/release/bioc/html/limma.html (https://bioconductor.org/packages/release/bioc/html/limma.html)
- https://github.com/alexdobin/STAR (https://github.com/alexdobin/STAR)
- https://bioconductor.org/packages/release/bioc/html/goseq.html (https://bioconductor.org/packages/release/bioc/html/goseq.html)
- https://guangchuangyu.github.io/2016/01/go-analysis-using-clusterprofiler/ (https://guangchuangyu.github.io/2016/01/go-analysis-using-clusterprofiler/)
- https://software.broadinstitute.org/software/igv/ (https://software.broadinstitute.org/software/igv/)
- https://bookdown.org/yihui/rmarkdown/ (https://bookdown.org/yihui/rmarkdown/)
- https://www.nature.com/articles/nmeth.3252 (https://www.nature.com/articles/nmeth.3252)
- https://en.wikipedia.org/wiki/Multidimensional_scaling (https://en.wikipedia.org/wiki/Multidimensional_scaling)

In [605]:

```
# report the version numbers of R and packages used in the session
sessionInfo()
```

```
R version 3.6.1 (2019-07-05)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.04.3 LTS

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] parallel  stats4    stats     graphics  grDevices utils     datasets
[8] methods   base

other attached packages:
 [1] gProfileR_0.6.7             gageData_2.22.0
 [3] gage_2.34.0                 SPIA_2.36.0
 [5] KEGGgraph_1.44.0            annotables_0.1.91
 [7] ensembldb_2.8.0             AnnotationFilter_1.8.0
 [9] GenomicFeatures_1.36.4      AnnotationHub_2.16.0
[11] BiocFileCache_1.8.0         dbplyr_1.4.2
[13] ggrepel_0.8.1               DEGreport_1.20.0
[15] forcats_0.4.0               stringr_1.4.0
[17] purrr_0.3.2                 readr_1.3.1
[19] tidyr_0.8.3                 tibble_2.1.3
[21] tidyverse_1.2.1             enrichplot_1.4.0
[23] pathview_1.24.0             DOSE_3.10.2
[25] GO.db_3.8.2                 rtracklayer_1.44.0
[27] goseq_1.36.0                geneLenDataBase_1.20.0
[29] BiasedUrn_1.07              clusterProfiler_3.12.0
[31] edgeR_3.26.5                limma_3.40.2
[33] sva_3.32.1                  mgcv_1.8-28
[35] nlme_3.1-140                ReportingTools_2.24.0
[37] knitr_1.23                  org.Hs.eg.db_3.8.2
[39] AnnotationDbi_1.46.0        NMF_0.21.0
[41] cluster_2.1.0               rngtools_1.4
[43] pkgmaker_0.27               registry_0.5-1
[45] genefilter_1.66.0           apeglm_1.6.0
[47] ggbeeswarm_0.6.0            PoiClaClu_1.0.2.1
[49] RColorBrewer_1.1-2          pheatmap_1.0.12
[51] ggplot2_3.2.0               vsn_3.52.0
[53] DESeq2_1.24.0               SummarizedExperiment_1.14.0
[55] DelayedArray_0.10.0         BiocParallel_1.18.0
[57] matrixStats_0.54.0          Biobase_2.44.0
[59] GenomicRanges_1.36.0        GenomeInfoDb_1.20.0
[61] IRanges_2.18.1              S4Vectors_0.22.0
[63] BiocGenerics_0.30.0         magrittr_1.5

loaded via a namespace (and not attached):
  [1] Hmisc_4.2-0                    Rsamtools_2.0.0
  [3] foreach_1.4.4                  crayon_1.3.4
  [5] MASS_7.3-51.4                  backports_1.1.4
  [7] GOSemSim_2.10.0                rlang_0.4.0
  [9] XVector_0.24.0                 readxl_1.3.1
 [11] GOstats_2.50.0                 rjson_0.2.20
 [13] bit64_0.9-7                    glue_1.3.1
 [15] mixsqp_0.1-97                  vipor_0.4.5
 [17] UpSetR_1.4.0                   haven_2.1.1
```

```
 [19] tidyselect_0.2.5              XML_3.98-1.20
 [21] GenomicAlignments_1.20.1      xtable_1.8-4
 [23] evaluate_0.14                 bibtex_0.4.2
 [25] cli_1.1.0                     zlibbioc_1.30.0
 [27] hwriter_1.3.2                 rstudioapi_0.10
 [29] rpart_4.1-15                  fastmatch_1.1-0
 [31] shiny_1.3.2                   xfun_0.8
 [33] clue_0.3-57                   caTools_1.17.1.2
 [35] pbdZMQ_0.3-3                  KEGGREST_1.24.0
 [37] interactiveDisplayBase_1.22.0 biovizBase_1.32.0
 [39] logging_0.10-108              Biostrings_2.52.0
 [41] png_0.1-7                     reshape_0.8.8
 [43] zeallot_0.1.0                 withr_2.1.2
 [45] bitops_1.0-6                  ggforce_0.2.2
 [47] RBGL_1.60.0                   plyr_1.8.4
 [49] cellranger_1.1.0              GSEABase_1.46.0
 [51] PFAM.db_3.8.2                 coda_0.19-3
 [53] pillar_1.4.2                  gplots_3.0.1.1
 [55] GlobalOptions_0.1.0           pscl_1.5.2
 [57] GetoptLong_0.1.7              europepmc_0.3
 [59] vctrs_0.2.0                   generics_0.0.2
 [61] urltools_1.7.3                tools_3.6.1
 [63] foreign_0.8-72                beeswarm_0.2.3
 [65] munsell_0.5.0                 tweenr_1.0.1
 [67] fgsea_1.10.0                  httpuv_1.5.1
 [69] compiler_3.6.1                GenomeInfoDbData_1.2.1
 [71] gridExtra_2.3                 lattice_0.20-38
 [73] AnnotationForge_1.26.0        later_0.8.0
 [75] dplyr_0.8.3                   jsonlite_1.6
 [77] affy_1.62.0                   GGally_1.4.0
 [79] scales_1.0.0                  graph_1.62.0
 [81] lazyeval_0.2.2                promises_1.0.1
 [83] doParallel_1.0.14             latticeExtra_0.6-28
 [85] R.utils_2.9.0                 checkmate_1.9.4
 [87] cowplot_1.0.0                 dichromat_2.0-0
 [89] BSgenome_1.52.0               igraph_1.2.4.1
 [91] yaml_2.2.0                    survival_2.44-1.1
 [93] numDeriv_2016.8-1.1           ashr_2.2-32
 [95] SQUAREM_2017.10-1             htmltools_0.3.6
 [97] memoise_1.1.0                 VariantAnnotation_1.30.1
 [99] locfit_1.5-9.1                viridisLite_0.3.0
[101] digest_0.6.20                 assertthat_0.2.1
[103] mime_0.7                      rappdirs_0.3.1
[105] repr_1.0.1                    emdbook_1.3.11
[107] RSQLite_2.1.1                 data.table_1.12.2
[109] blob_1.2.0                    R.oo_1.22.0
[111] preprocessCore_1.46.0         splines_3.6.1
[113] Formula_1.2-3                 labeling_0.3
[115] OrganismDbi_1.26.0            ProtGenerics_1.16.0
[117] RCurl_1.95-4.12               broom_0.5.2
[119] hms_0.5.0                     modelr_0.1.5
[121] colorspace_1.4-1              ConsensusClusterPlus_1.48.0
[123] base64enc_0.1-3               BiocManager_1.30.4
[125] mnormt_1.5-5                  shape_1.4.4
[127] nnet_7.3-12                   Rcpp_1.0.1
[129] circlize_0.4.6                truncnorm_1.0-8
[131] Nozzle.R1_1.1-1               IRdisplay_0.7.0
[133] R6_2.4.0                      grid_3.6.1
[135] ggridges_0.5.1                acepack_1.4.1
[137] curl_3.3                      gdata_2.18.0
[139] affyio_1.54.0                 DO.db_2.9
[141] Matrix_1.2-17                 qvalue_2.16.0
[143] ggbio_1.32.0                  iterators_1.0.10
[145] htmlwidgets_1.3               polyclip_1.10-0
[147] triebeard_0.3.0               biomaRt_2.40.4
[149] gridGraphics_0.4-1            rvest_0.3.4
[151] ComplexHeatmap_2.0.0          htmlTable_1.13.1
[153] codetools_0.2-16              lubridate_1.7.4
```

```
[155] gtools_3.8.1                prettyunits_1.0.2
[157] psych_1.8.12                gridBase_0.4-7
[159] R.methodsS3_1.7.1           gtable_0.3.0
[161] DBI_1.0.0                   httr_1.4.0
[163] KernSmooth_2.23-15          stringi_1.4.3
[165] progress_1.2.2              reshape2_1.4.3
[167] farver_1.1.0                uuid_0.1-2
[169] annotate_1.62.0             viridis_0.5.1
[171] hexbin_1.27.3               Rgraphviz_2.28.0
[173] xml2_1.2.0                  ggdendro_0.1-20
[175] rvcheck_0.1.3               bbmle_1.0.20
[177] IRkernel_1.0.2              geneplotter_1.62.0
[179] ggplotify_0.0.3             Category_2.50.0
[181] bit_1.1-14                  ggraph_1.0.2
[183] pkgconfig_2.0.2             lasso2_1.2-20
```

In [ ]:

```
1
```